

Appendix D: Programs of the rebuild and revision (verbatim listing)

Companion to Cieszewski (2026), Artificial Intelligence as a Research Instrument and an Editorial Burden, MCFNS 18(1). A verbatim listing of the programs that produce and check the results of this paper, from the files deposited in Appendix C (scripts folder). Run order: compute_normalization.py, revision_checks.py, nash_search.py, make_figures.py, build_tex.py, build_docx.py, render_docx.py, verify_numbers.py; for Appendix B: redo_standardized.py, compute_ugapay.py, make_pay_figures.py, pay_sensitivity.py, appendix_a.py, with paynames.py as their shared name key; this listing: build_program_appendix.py. Paths inside the programs refer to the author's machine; the Appendix C README states how to point them at the package.

Contents: 1. appendix_a.py (42,381 bytes); 2. build_docx.py (5,638 bytes); 3. build_program_appendix.py (4,466 bytes); 4. build_tex.py (18,847 bytes); 5. check_authorship_fields.py (4,580 bytes); 6. check_layout.py (14,584 bytes); 7. compile.py (4,253 bytes); 8. compute_normalization.py (1,129 bytes); 9. compute_ugapay.py (8,139 bytes); 10. data_core.py (8,330 bytes); 11. figtrim.py (1,193 bytes); 12. make_figures.py (11,494 bytes); 13. make_package.py (16,910 bytes); 14. make_pay_figures.py (18,267 bytes); 15. manuscript_content.py (33,245 bytes); 16. nash_search.py (1,576 bytes); 17. ooxml_fallback.py (8,239 bytes); 18. pay_sensitivity.py (6,210 bytes); 19. paynames.py (1,155 bytes); 20. pdfclean.py (1,613 bytes); 21. profiles_admin_warnell.py (5,950 bytes); 22. redo_standardized.py (20,129 bytes); 23. render_docx.py (2,129 bytes); 24. revision_checks.py (6,247 bytes); 25. verify_numbers.py (27,795 bytes)

1. appendix_a.py (SHA-256 6d2dac72d615f028...; 221 lines)

```
"""Appendix B (formerly Supplement S2): the University of Georgia pay exercise, moved out of the main text at review.
```

```
Builds the block list shared by the standalone DOCX/PDF (Cieszewski2026_AI_AppendixB_pay) and the LaTeX fragment build/appendix_a.tex that build_tex.py appends to the Article from build/ugapay.json, build/pay_figures.json and the companion workbook. Descriptive; not used for inference in the Article. Revision 3: sequential sample flow, screening criteria stated, observed pay primary, full model tables with repeated headers, self-explanatory captions, narrowed conclusion.
```

```
"""
import json, subprocess, os, unicodedata, re
APPX='B'
TAXTAB='9' if 'verification_case' in os.environ.get('MS_MD','') else 'A6' # standalone appendix; TeX resolves the shared label
from pathlib import Path
from pdfclean import strip_pdf_dates
import pandas as pd
from docx import Document
from docx.shared import Inches, Pt, RGBColor
from docx.oxml.ns import qn
from docx.oxml import OxmlElement
ROOT=Path(__file__).resolve().parents[1]
u=json.loads((ROOT/'build/ugapay.json').read_text());pf=json.loads((ROOT/'build/pay_figures.json').read_text())
ONES=[c for c,v in sorted(u['by_college'].items()) if v['n_persons_incl_imputed']==1] # one-person standardization groups
XL=ROOT/'UGA_StanfordList_Salaries_FY2021-2025.xlsx'
panel=pd.read_excel(XL,sheet_name='Std panel');lg=pd.read_excel(XL,sheet_name='Scientists long');pre=pd.read_csv(ROOT/'build/prewindow_retirees.csv');idx=pd.read_excel(XL,sheet_name='Authorship indices (321)')
pc=u['panel_counts']
ps=json.loads((ROOT/'build/pay_sensitivity.json').read_text())
cc=u['college_check']
checked=pd.read_csv(ROOT/'build/college_checked.csv')
changed_before=int((checked.changed&checked.in_pay_analysis_before).sum())
n_before=int(checked.in_pay_analysis_before.sum())
def ptext(p):
    return 'p < 0.001' if p<0.001 else f'p = {p:.3f}' if p<0.01 else f'p = {p:.2f}'
def corr(k):
    v=pf['rho'][k]['observed'];return f"{v['rho']:.2f}, {ptext(v['p'])}"
n_matched=int(lg.auth.nunique());n_profiles=int(len(idx));n_unmatched=n_profiles-n_matched
norm=lambda s:unicodedata.normalize('NFKD',str(s)).encode('ascii','ignore').decode().strip().lower()
pre_in_list=sum(norm(a) in set(map(norm,idx.auth)) for a in pre.auth);pre_carried=int(panel[panel.imputed&~panel.auth.isin(lg.auth)].auth.nunique())
assert pc['panel_persons_total']==n_matched+pre_carried,(pc['panel_persons_total'],n_matched,pre_carried)
never=panel[~panel.auth.isin(panel[panel.use].auth)]
pre_drop_text=(f" The remaining {pre_in_list-pre_carried} were dropped because no college could be found." if pre_in_list>pre_carried else "")
```

```

BLOCKS=[]
def appx(text):
    return re.sub(r'(?:(?<=Appendix )B\b|(?<=Table )B(?=\d)|(?<=Figure )B(?=\d)|\bB(?=\.\d))',APPX,text)

def P(text,style=None):BLOCKS.append(('para' if style is None else style,appx(text)))
def table(headers,rows,widths):BLOCKS.append(('table',headers,[appx(str(v)) for v in row] for row in
rows],widths))
def FIG(f,cap):BLOCKS.append(('figure',f,appx(cap)))
P('Appendix B: Public pay records of University of Georgia scientists on the 2024 citation list: a descriptive
exercise','Title')
P('Companion to: Cieszewski, C.J., 2026. Artificial Intelligence as a Research Instrument and an Editorial
Burden: Scientific Judgment, Forestry Scientometrics, and Academic Incentives. Math. Comput. For. Nat.-Res. Sci.
18(1).')
P('B.1 Status','Heading 1')
P(f'This appendix reports an analysis included in the Discussion of an earlier draft and separated in response to
author-commissioned AI critiques. It compares public fiscal-year pay with career bibliometric measures for
University of Georgia (UGA) scientists on the 2024 data year of the Stanford/Ioannidis list. It also compares
deans with the listed scientists of their colleges. The main text uses these results as an illustration, not as a
causal test. The analysis describes the data and suggests hypotheses for further study. Colleges were first
inferred from subfields, the inference shown to be unreliable in Table {TAXTAB} in Appendix A, and then checked
person by person against public university pages (Section B.3). The outcome is recorded fiscal-year gross pay,
which is not a salary rate, and {pc["rows_imputed"]} of the {pc["rows"]} person-years are modeled rather than
observed. The cross-sectional design cannot identify the criteria or causal processes that determine pay.
Observed pay is the primary basis of every figure and the active-only model columns. Modeled retiree values are
reported separately.')
P('B.2 Data and sample flow','Heading 1')
P(f'The State of Georgia publishes the name, title, and fiscal-year gross pay of every University System employee
on the Open Georgia portal, open.ga.gov [[cite:OpenGeorgia_2026]]. I matched the {n_profiles} nominal UGA
profiles of the 2024 data year by surname and given name, ignoring diacritics, to the UGA pay records for fiscal
years 2021 to 2025. This gave {n_matched} matched profiles and {n_unmatched} unmatched. One name match was a
different person, a university employee with the same surname and first name and a non-academic title, and is
excluded (Section B.3). Three of the unmatched profiles belong to universities in the country of Georgia, so the
nominal UGA count includes them but the pay results do not. The {n_unmatched} unmatched profiles were searched in
an additional extract for fiscal years 2015 to 2020, and {pre_in_list} were found. The transcribed historical
list contains {len(pre)} rows, with {pre_in_list} matching a listed profile after transcription. Rows without a
listed profile are excluded. Of the matching profiles, {pre_carried} had a college found and were included in the
panel.{pre_drop_text} The {n_unmatched-pre_in_list} profiles unmatched in either window were not verified
individually. Failure to find a payroll record does not by itself show that a person has retired, died, or left,
and the list gives last-known affiliations. Table B1 shows the successive selection steps. Two workbook sheets
were prepared before the match ignoring diacritics: Scientists x FY, with 190 persons, and Not found in OpenGA,
with 131 profiles. These sheets predate seven matches, which had no college before the check and now have one,
and are retained unchanged so the work can be traced. The panel sheet (Std panel) is the source to use.')
rows=[['Listed UGA profiles in the 2024 data year',str(n_profiles),'all'],
['Matched to FY2021 to 2025 pay records (diacritic-insensitive name match)',str(n_matched),'continue'],
['Not matched in FY2021 to 2025',str(n_unmatched),'searched in FY2015 to 2020'],
[' found in FY2015 to 2020 records',str(pre_in_list),'continue'],
[' carried into the panel (college found)',str(pre_carried),'continue'],
*([' dropped (no college assignable)',str(pre_in_list-pre_carried),'excluded']] if pre_in_list>pre_carried
else []),
[' not found in either window',str(n_unmatched-pre_in_list),'excluded, unverified'],
['Persons in the assembled panel',str(pc['panel_persons_total']),f'{n_matched} + {pre_carried}'],
['Persons with no usable person-year (every year administrative, temporary or retiree title, below the pay floor,
or a partial exit year, or the one payroll match that was a different
person)',str(pc['persons_never_usable']),'excluded'],
['Persons with at least one usable person-year',str(pc['persons']),'panel'],
[' with at least one observed screened year (observed persons)',str(pc['persons_active']),'primary'],
[' with modeled years only (pre-window retirees)',str(pc['persons_imputed']-pc['persons_both']),'secondary'],
[' observed persons who also have modeled years (retired inside the window)',str(pc['persons_both']),'counted
among observed'],
['Person-years: observed / modeled / total',f"{pc['rows_observed']} / {pc['rows_imputed']} / {pc['rows']}",'',],
['Persons / person-years in models E',f"{u['n_persons']} / {u['n_obs']}",'pooled'],
['Persons / person-years in models E, observed pay only',f"{u['models']['E_active_mixed']['n_persons']} /
{u['models']['E_active_mixed']['n_obs']}",'primary'],
['Persons / person-years in models F (at least one single-author and one last-author paper)',f"{u['models']
['F_mixed']['n_persons']} / {u['models']['F_mixed']['n_obs']}",'pooled']]
P(f'Table B1. Successive sample selection from the {n_profiles} nominal UGA profiles of the 2024 data year,
including three unmatched profiles at universities in the country of Georgia (Stanford/Ioannidis career database,
version 8) to each figure and model sample. FY = state fiscal year (July to June). A usable person-year is an
observed screened year (Section B.3) or a modeled year. Observed persons have at least one observed screened
year. Source: workbook sheets Std panel and Scientists long; build/redo_standardized.py;
build/compute_ugapay.py.','Caption');table(['Step','Persons (or persons / person-years)','Use'],rows,
[.64,.2,.16])
P('Table B2 gives Spearman correlations between observed standardized pay and papers, citations, and citations
per paper in the three authorship classes. Sections B.3 and B.4 provide the definitions and model estimates.')
def rr(k):
    v=pf['rho'][k]['observed'];return [f"{v['rho']:.2f}",('<0.001' if v['p']<0.001 else f"{v['p']:.3f}" if
v['p']<0.01 else f"{v['p']:.2f}"),str(v['n'])]
P('Table B2. '+'Spearman rank correlations of standardized pay with career papers, citations, and citations per
paper for University of Georgia scientists on the 2024 list. The three authorship classes are all positions, last

```

author, and single author. Only persons with observed pay are included.', 'Caption'); table(['Feature of the publication record', 'rho', 'p', 'n'], [

```

    ['Papers, all positions (np)'] + rr('np'), ['Papers as last author (np_last)'] + rr('np_last'), ['Papers as single author (nps)'] + rr('nps'),
    ['Citations, all positions (nc)'] + rr('nc'), ['Citations to last-author papers (nc_last)'] + rr('nc_last'),
    ['Citations to single-author papers (ncs)'] + rr('ncs'),
    ['Citations per paper, all positions (nc / np)'] + rr('cpp_all'), ['Citations per paper, last author (nc_last / np_last)'] + rr('cpp_last'), ['Citations per paper, single author (ncs / nps)'] + rr('cpp_single')
], [.55, .15, .15, .15])
P(f"Pay: mean standardized fiscal-year gross pay over each person's observed screened years, fiscal 2021 to 2025. Standardization divides pay by the five-year mean for observed listed scientists of the same college and multiplies it by their university-wide mean ({u['U_mean']:.0f} dollars). Values remain in dollars and are rescaled to a common college mean. Persons: listed profiles matched to public pay records with at least one non-administrative, non-temporary year above 50,000 dollars (Appendix B, Table B1). The value of n is smaller where an authorship class has no paper. Authorship classes come from the version 8 career table (papers 1960 to 2024, citations 1996 to 2024). All positions: np, nc. Last author: np_last = npsfl minus cpsf and nc_last = ncsfl minus ncsf. These are papers on which the person is last but not first or sole author, and their citations. Single author: nps, ncs. Citations per paper of a class are class citations divided by class papers. Regressions of log pay on log papers, with year and career-age controls, give elasticity {u['lnp_range'][0]:.2f} to {u['lnp_range'][1]:.2f} in every specification (Appendix B, Table B3). Source: Open Georgia pay records; build/make_pay_figures.py and build/compute Ugapay.py.", 'Note')

P('B.3 Screening, college assignment, outcome, and standardization', 'Heading 1')
P(f"Screening. A person-year enters the observed base only if its recorded title and pay meet the following conditions. Administrative titles are excluded: dean, associate or assistant dean, department chair or head, director, provost, and chief academic affairs officer. Temporary, retiree, former, limited-term, affiliate, occasional, and part-time classes are also excluded. Gross pay must be at least 50,000 dollars and at least 80% of the previous screened year. A lower year is treated as a partial exit year and excluded. Appointment fraction, contract length, and duties are not observed. Thus, screening approximates a regular full-time academic appointment but does not verify that status, and the earlier full-time label is withdrawn. Persons with administrative titles in some years contribute only their non-administrative years. In the panel, {int(never.auth.nunique())} persons had no usable year. Their most frequent titles were '+', '.join(f'{k} ({v} person-years)' for k,v in never.titles.value_counts().head(4).items())+'.')
P(f"College assignment. The citation database has no department field, so colleges were first inferred from subfields, with Warnell and Odum corrections, the same inference criticized in the main text (Table {TAXTAB} in Appendix A). Because college sets the standardization denominator and the dean comparison, every person was then looked up with AI assistance on public university pages, i.e., department and college pages, profiles, news releases and CVs, or in a publication affiliation. Of {cc['persons_in_csv']} persons checked, {cc['confirmed']} were confirmed and {cc['probable']} were probable because of joint appointments or indirect or older sources. The inference was wrong for {cc['changed_from_heuristic']} persons, including {changed_before} of the {n_before} then in the analysis, for example psychology-labelled scientists moving to Family and Consumer Sciences and plant scientists to CAES. The false payroll match and the retiree row without a profile on the citation list were excluded. School of Computing faculty are counted with Franklin. Three small units keep their own group, each with one usable person: the journalism college (Grady), the medical partnership (Medicine), and the CCRC research center. The file college_checked.csv in Appendix C gives each person's college, source, and a short quotation. Colleges and usable persons: '+', '.join(f'{c} {n}' for c,n in sorted(panel[panel.use].groupby('college').auth.nunique().items(), key=lambda x:-x[1]))+'.')
# academic rank from the recorded titles of the observed person-years (B.5 limitation 8; the main text cites the Professor count)
def _rank(t):
    t=str(t).upper()
    return 'junior' if ('ASSISTANT PROF' in t or 'ASSOCIATE PROF' in t) else ('professor' if 'PROFESSOR' in t else 'other')
_obs=panel[panel.use&~panel.imputed].assign(rank=lambda d:d.titles.map(_rank)).groupby('auth')['rank'].agg(set)
RANKS={ 'n':int(_obs.size), 'professor':int(sum(s=='professor' for s in _obs)), 'junior':int(sum('junior' in s for s in _obs))}
RANKS['other']=RANKS['n']-RANKS['professor']-RANKS['junior']
if 'verification case' not in os.environ.get('MS_MD',''):
    assert f"{RANKS['professor']} of the {RANKS['n']} persons held the title Professor in every observed year" in (ROOT/'build'/'manuscript.md').read_text(encoding='utf-8'), RANKS
dean_college_names={'CAES':'Agriculture (CAES)', 'FACS':'Family and Consumer Sciences (FACS)', 'Franklin':'Arts and Sciences (Franklin)', 'Grady':'Journalism (Grady)', 'Odum':'Ecology (Odum)', 'SPIA':'Public and International Affairs (SPIA)', 'Terry':'Business (Terry)', 'Vet Med':'Veterinary Medicine', 'Warnell':'Forestry (Warnell)'}
DEANS_CAP=f"Deans and the listed scientists of their colleges on the standardized pay scale, college-years without a dean transition only. Left: for each college, the mean standardized pay of its observed screened listed scientists over the same clean fiscal years as its dean (blue) and of its dean (orange), joined by a gray bar and labelled with the ratio dean / scientists. In this panel, pay is standardized over those same years, so the scientists of every college are at U = {pf['dean_ratio_plot']['U']:.0f} dollars. Right: the same ratio for each clean fiscal year, FY2021 (lightest blue) to FY2025 (darkest blue); vertical line at ratio 1. Because standardization multiplies a dean and the scientists of one college by the same factor, the ratios equal the raw within-college ratios. Colleges: {'', '.join(dean_college_names.get(c,c) for c in sorted(pf['dean_ratio_by_college'], key=pf['dean_ratio_by_college'].get, reverse=True))}. Source: workbook sheet Std dean vs sci; build/make_pay_figures.py."
P(f"Outcome and standardization. The outcome is recorded fiscal-year gross pay. Values were standardized for differences in college pay levels using pay_std = pay / F_c * U, where F_c is the mean gross pay of observed screened person-years in college c over FY2021 to 2025. U = {u['U_mean']:.0f} dollars is the mean of the same person-years over all colleges. Modeled values enter neither F_c nor U. In the {len(ONES)} groups with one usable person ({',', '.join(ONES)}), F_c is that person's own mean over the observed screened years, so their mean standardized observed pay equals U. This transformation removes only differences in college means and does not make positions, contracts, or labor markets comparable. It leaves every within-college ratio unchanged because

```

dean and scientist pay in the same college are multiplied by the same factor. Thus, the dean pay ratios in Figure B1 are raw within-college ratios. Standardization affects only the pooling of scientists across colleges.")

FIG('deans_vs_scientists_std.png', 'Figure B1. '+DEANS_CAP) # first cited here (B.3), so the deans figure precedes the salaries figure

P(f'Modeled retiree pay. Scientists who retired within the window ({pc['persons_both']} persons) and before it ({pc['persons_imputed']-pc['persons_both']} persons) were retained by projecting their last screened pay forward at the yearly raise of the observed screened scientists of the same college. For fiscal 2016 to 2020, the median observed raise came from the historical extract. The raise was zero for fiscal 2021, and the college-year mean raise, computed with the checked colleges, was used for fiscal 2022 to 2025. These values estimate earnings under assumed continued employment and are not recorded university pay. Retirement was inferred from a missing screened record or a retiree title and was not verified. In every figure, observed persons are plotted using only the mean of their observed screened years. Thus, the {pc['persons_both']} in-window retirees appear as observed persons, and their modeled years are not used. The {pc['persons_imputed']-pc['persons_both']} pre-window retirees appear as hollow triangles at the mean of their modeled FY2021 to 2025 values. Five-year means use the years available for each person (1 to 5).")

P('Definitions. Authorship counts come from the version 8 career table in three classes. All positions: np = papers 1960 to 2024 and nc = citations 1996 to 2024. Last author: np_last = papers on which the person is last but not first or sole author (npsfl minus cpsf), and nc_last = their citations (ncsfl minus ncsf). Single author: nps and ncs. build/check_authorship_fields.py recomputes np_last, nc_last, nps, and ncs from the raw version 8 release for all '+str(n_profiles)+' profiles and finds no difference. Shares are np_last / np, nc_last / nc, nps / np and ncs / nc. Citations per paper of a class are class citations divided by class papers. If the class has no paper, this value is undefined and the person is omitted from that panel. The figures add 1 to last-author and single-author counts to show zeros on the logarithmic axis. Log citations per single-author and per last-author paper are undefined for zero papers or zero citations, so those person-years are omitted from regression model F (six person-years have single-author papers but no single-author citations). Lines are fitted by ordinary least squares through observed persons only, using the plotted scales (log-log where both axes are logarithmic).')

P('B.4 Results', 'Heading 1')

P(f'The pay-record comparison [{cite:OpenGeorgia_2026}] gives the following results. Of the nominal University of Georgia profiles, {n_matched} matched pay records by name, and {pc['persons_active']} had at least one observed year meeting the screening rules. These {pc['persons_active']} persons, with pay averaged over their observed years, provide the basis for the observed-pay results reported here, while pooled models also include modeled retiree pay. Table B1 gives the sample flow, and Sections B.3 to B.5 give screening criteria, modeled values, model tables, the dean comparison, and limitations. I standardized each pay value for differences in college pay levels, dividing by the mean for observed listed scientists of the same college over the study window and multiplying by the university-wide mean for that group. Thus, the values remain in dollars and are rescaled to a common college mean. Table B2 gives results for all positions, last author, and single author, where last author means papers on which the person is last but not first or sole author. Fig. B2 gives the scatter plots. Standardized pay goes about equally with papers in all positions (Spearman rho = {corr('np')}, n = {pf['rho'] ['np']}['observed'] ['n']}) and last-author papers ({corr('np_last')}), while single-author papers have a weak association ({corr('nps')})). The elasticity is {u['lnp_range'][0]:.2f} to {u['lnp_range'][1]:.2f} across the regression specifications with year and career-age controls. For citations, the association is strongest for last-author papers ({pf['rho'] ['nc_last'] ['observed'] ['rho']:+.2f}), followed by all positions ({pf['rho'] ['nc'] ['observed'] ['rho']:+.2f}) and single-author papers ({pf['rho'] ['ncs'] ['observed'] ['rho']:+.2f}). Citations per paper have a positive association for last-author papers ({corr('cpp_last')}), a weak and nonsignificant one in all positions ({corr('cpp_all')}), and none for single-author papers ({corr('cpp_single')}). The last-author share of papers correlates with pay ({pf['rho_shares'] ['last_pap_share'] ['observed'] ['rho']:+.2f}, {ptext(pf['rho_shares'] ['last_pap_share'] ['observed'] ['p'])}) and is significant in the pooled models, but not on observed pay only. Citations per last-author paper are positive under both estimators of model F, also on observed pay only (Table B3 and Section B.5).")

def rs(k,g):

 v=pf['rho'][k][g];return f"rho = {v['rho']:+.2f} (p {'< 0.001' if v['p']<0.001 else '=' %3f'%v['p'] if v['p']<0.01 else '=' %2f'%v['p']}, n = {v['n']})"

caps=([('pay_authorship_output.png',f'Standardized gross pay of UGA scientists on the 2024 list, by authorship class. Columns from left to right: all positions, last author, single author, as defined in Section B.3. The three rows, a to c: standardized pay (pay_std, dollars), averaged over the person's observed screened fiscal years 2021 to 2025. Pay is compared with citations per author (a), citations per paper (b, omitting persons with no paper in the class), and papers (c). Last-author and single-author counts have 1 added to show zeros on the logarithmic axes. Blue dots: {pf['n_observed']} observed persons. Orange hollow triangles: {pf['n_modeled_only']} pre-window retirees with modeled pay only. Line: least-squares fit through observed persons. Text above each panel: Spearman rho, p-value, and n for observed persons. Source: workbook sheets Std panel and Authorship indices (321); build/make_pay_figures.py."))

for f,cap in caps:FIG(f,f'Figure B2. {cap}') # precedes the results paragraph in the source so that LaTeX can set it at the top of the page whose text discusses it

maxdiff=max(abs(v['all'] ['rho']-v['observed'] ['rho']) for k,v in pf['rho'].items() if not k.startswith('exhibit_') and v['all'] ['rho'] is not None and v['observed'] ['rho'] is not None)

P(f'Figure B2 compares standardized pay with career citations per author (row a), citations per paper (row b), and papers (row c). Columns are the three authorship classes of Section B.3: all positions, last author, and single author. Figure B1 compares deans with the listed scientists of their colleges. For observed persons, Spearman correlations of pay with citations per author were {rs('nc','observed')} for all positions, {rs('nc_last','observed')} for last author, and {rs('ncs','observed')} for single author. With citations per paper, they were {rs('cpp_all','observed')} for all positions, {rs('cpp_last','observed')} for last author, and {rs('cpp_single','observed')} for single author. With papers, they were {rs('np','observed')} for all positions, {rs('np_last','observed')} for last author, and {rs('nps','observed')} for single author. The largest coefficient change is {maxdiff:.3f} when the persons with modeled pay only are included as well (values in pay_figures.json, Appendix C).")

labels=('Intercept':'Intercept', 'C(yr)[T.2022]':'Fiscal 2022 (vs 2021)', 'C(yr)[T.2023]':'Fiscal 2023 (vs 2021)', 'C(yr)[T.2024]':'Fiscal 2024 (vs 2021)', 'C(yr)[T.2025]':'Fiscal 2025 (vs 2021)', 'lnp':'log np (papers, all positions)', 'lcpp':'log citations per paper (all positions)', 'lage':'log career age (2025 minus first publication year)', 'single_pap':'Single-author share of papers (nps / np)', 'single_cit':'Single-author share of citations (ncs / nc)', 'last_pap':'Last-author share of papers (np_last / np)', 'last_cit':'Last-author share of citations

```

(nc_last / nc)', 'lcpp_single': 'log citations per single-author paper', 'lcpp_last': 'log citations per last-author
paper')
def fmt(m, k):
    p = m['params'].get(k)
    if p is None: return ''
    pv = 'p<0.001' if p['p'] < 0.001 else 'p=%3f'%p['p']
    return '%+.3f (%3f; %s) %'(p['coef'], p['se'], pv)
keys = ['Intercept', 'C(yr [T.2022])', 'C(yr [T.2023])', 'C(yr [T.2024])', 'C(yr [T.2025])', 'lnp', 'lcpp', 'lage', 'single_pap', 'single_cit', 'last_pap', 'last_cit', 'lcpp_single', 'lcpp_last']
mods = [('E_active_mixed', 'E observed, person RI'), ('E_active_ols_cluster', 'E observed, OLS cl.'), ('E_mixed', 'E pooled, person RI'), ('E_ols_cluster', 'E pooled, OLS cl.'), ('F_mixed', 'F pooled, person RI'), ('F_ols_cluster', 'F pooled, OLS cl.')]
rows = [[labels[k]] + [fmt(u['models'][m], k) for m, _ in mods] for k in keys]
rows.append(['Person-years / persons'] + [f"{u['models'][m]['n_obs']} / {u['models'][m]['n_persons']}" for m, _ in mods])
P('Table B3. Complete estimates for models of log standardized pay: coefficient (standard error; p-value). Dependent variable: natural logarithm of pay_std (Section B.3) for each person-year. Model E uses log np, log citations per paper (all positions), log career age, and the four authorship shares. Model F uses log np, log career age, log citations per single-author paper, and log citations per last-author paper. It includes person-years with at least one paper in each class. Every model includes fiscal-year effects, with fiscal 2021 as the reference. Person RI: linear mixed model with a random intercept per person, fitted by restricted maximum likelihood. OLS cl.: ordinary least squares with standard errors clustered by person. Observed: observed screened person-years only (primary). Pooled: observed and modeled person-years together. Coefficients on logged predictors are elasticities. Coefficients on shares give the change in log pay from share 0 to share 1. Year coefficients give log differences from fiscal 2021. Sources: Open Georgia pay records and the version 8 career table; build/compute_ugapay.py (statsmodels).', 'Caption'); table(['Term'] + [l for _, l in mods], rows, [.22] + [.13]*6)
P(f"Sensitivity. The associations hold without the {ps['definitions']['variants']['base']['PAY_FLOOR']:.0f}-dollar pay floor, without the rule that removes partial exit years, and without either rule (Table B4). Without both rules, the elasticity rises to {ps['neither']['lnp_range'][0]:.2f} to {ps['neither']['lnp_range'][1]:.2f}. The one large change is the SPIA ratio in fiscal 2021, which reaches {ps['no_floor']['ratio_max']:.2f} without the floor. One partial year below {ps['definitions']['variants']['base']['PAY_FLOOR']:.0f} dollars lowers the mean in a college with three listed scientists, which is why the floor is used.")
variants = [('base', 'Base'), ('no_floor', 'No floor'), ('no_exit_rule', 'No exit rule'), ('neither', 'Neither')]
rows = [['Observed persons'] + [f"{ps[variant]['n_observed']}" for variant, _ in variants]]
for label, k in (('Papers rho, all positions', 'np'), ('Papers rho, last author', 'np_last'), ('Papers rho, single author', 'nps'), ('Citations per paper rho, all positions', 'cpp_all'), ('Citations per paper rho, last author', 'cpp_last'), ('Citations per paper rho, single author', 'cpp_single')):
    rows.append([label] + [f"{ps[variant]['rho'][k]['rho']:.2f}" for variant, _ in variants])
rows.append(['Elasticity range'] + [f"{ps[variant]['lnp_range'][0]:.2f} to {ps[variant]['lnp_range'][1]:.2f}" for variant, _ in variants])
for label, m in (('Model F on observed pay, citations per last-author paper, person random intercept', 'F_active_mixed'), ('Model F on observed pay, citations per last-author paper, OLS clustered', 'F_active_ols_cluster')):
    rows.append([label] + [f"{ps[variant]['models'][m]['params']['lcpp_last']['coef']:.3f} ({ptext(ps[variant]['models'][m]['params']['lcpp_last']['p']))}" for variant, _ in variants])
rows.append(['Dean ratio range'] + [f"{ps[variant]['ratio_min']:.2f} to {ps[variant]['ratio_max']:.2f}" for variant, _ in variants])
P('Table B4. Sensitivity to the pay floor and partial-exit-year rule, with checked colleges in every variant. The whole pay pipeline is rerun for each variant.', 'Caption'); table(['Measure'] + [label for _, label in variants], rows, [.36] + [.16]*4)
P(f"Note. Columns give the screening variants. Base: {ps['definitions']['variants']['base']['PAY_FLOOR']:.0f}-dollar floor and {float(ps['definitions']['variants']['base']['PAY_EXIT_RATIO']):.0%} exit rule. No floor retains the exit rule. No exit rule retains the floor. The Neither variant removes both rules. The rho rows are Spearman correlations of mean standardized observed pay with papers and with citations per paper, one row per authorship class. The elasticity range is the range of the log np coefficient over the six Table B3 specifications. The two model F rows give the coefficient on log citations per last-author paper, with its p-value, separately for the person random-intercept estimator and OLS with errors clustered by person. The dean range covers clean annual within-college ratios. Source: build/pay_sensitivity.json; build/pay_sensitivity.py.", 'Note')

gr = pd.read_excel(XL, sheet_name='Greene FY2015-2025'); wr = panel[(panel.college == 'Warnell') & panel.use & ~panel.imputed]
gy = (int(r.fy):float(r.salary) for r in gr.itertuples()); wm = (int(fy):float(g.pay.mean()) for fy, g in wr.groupby('fy'))
greene = ', '.join(f"{gy[y]:.0f} dollars in fiscal {y} against a mean of {wm[y]:.0f} for the {int((wr.fy==y).sum())} listed Warnell scientists with observed pay ({gy[y]/wm[y]:.2f} times)" for y in (2021, 2022, 2023))
P(f"Dean premia. In each fiscal year the university carried 17 to 20 academic deans and 52 to 58 associate or assistant deans. Comparing each dean with the observed screened listed scientists of the same college, in college-years without a dean transition, the ratio of dean pay to mean scientist pay ranged from {u['ratio_min']:.2f} to {u['ratio_max']:.2f}: '+'; '.join(f"{c} {lo:.2f} to {hi:.2f}" for c, (lo, hi) in sorted(u['ratio_by_college'].items(), key=lambda x: -x[1][1])) + f". The comparison is with listed scientists, not with the faculty of the college; the scientist means use observed screened person-years only (in Figure B1 over the same fiscal years as the dean); and differences between colleges may reflect market structure or joint appointments. In raw dollars, the dean of the forestry school (Warnell) was paid {greene}; fiscal 2024 and 2025 are dean-transition years and are excluded. The listed scientists are not named in this appendix.")
P('B.5 Limitations and the hypothesis', 'Heading 1')
P(f"(1) College was checked for every person, but joint appointments and the {cc['probable']} probable cases may still sit in the less relevant college, which would change the standardization denominator and the dean

```

comparison. (2) The sample contains listed profiles with matched pay rather than UGA faculty, and unmatched profiles were not verified. (3) Gross pay is not a salary rate, and screening only approximates a regular appointment. (4) Modeled retiree pay is included in the pooled models. Observed-pay results are the observed columns of Table B3, the blue markers of Figure B2, and the dean comparison of Figure B1. (5) Career records through 2024 are compared with pay in fiscal 2021 to 2025. This cross-sectional comparison between persons cannot determine cause and effect. (6) Last-author position has not been validated here as a measure of leading the work reported in a paper. (7) Nonsignificant coefficients do not show that an effect is absent. (8) Academic rank was not used as a variable. Rank tracks career output, the last-author share, and pay, so the volume and last-author associations could arise from promotion alone. Of the {RANKS['n']} persons with observed pay, {RANKS['professor']} held the title Professor in every observed year, {RANKS['junior']} held the title Associate or Assistant Professor in at least one observed year, and {RANKS['other']} held research titles.")

pv=lambda m,k:u['models'][m]['params'][k]['p']
cf=lambda m,k:u['models'][m]['params'][k]['coef']

P(f"The results show a positive association between publication volume and standardized pay across the reported specifications. The elasticity of log np is between {u['lnp_range'][0]:.2f} and {u['lnp_range'][1]:.2f}, including the observed-pay models. In the rank correlations, papers were associated about equally with pay in all positions and the last-author class, while citations and citations per paper were strongest in the last-author class. The single-author class had the weakest associations, and the lowest value ({pf['rho']['cpp_single']['observed']['rho']+.2f}, p = {pf['rho']['cpp_single']['observed']['p']:.2f}) was for citations per single-author paper. The differences between these correlations were not tested. Associations with authorship shares and citation-intensity measures depend on the predictor, sample, and estimator. For last-author share of papers, the observed-pay coefficients are {cf('E_active_mixed','last_pap'):.2f} and {cf('E_active_ols_cluster','last_pap'):.2f}, with p = {pv('E_active_mixed','last_pap'):.2f} and {pv('E_active_ols_cluster','last_pap'):.2f}. The pooled models give p = {pv('E_mixed','last_pap'):.2f} and {pv('E_ols_cluster','last_pap'):.2f}. In model F, citations per last-author paper have coefficients {cf('F_mixed','lcpp_last'):.3f} and {cf('F_ols_cluster','lcpp_last'):.3f}, with {ptext(pv('F_mixed','lcpp_last'))} and {ptext(pv('F_ols_cluster','lcpp_last'))}. This is an association with citation intensity. Fitted to observed pay only ({u['models']['F_active_mixed']['n_obs']} person-years of {u['models']['F_active_mixed']['n_persons']} persons), model F gives {cf('F_active_mixed','lcpp_last'):.3f} and {cf('F_active_ols_cluster','lcpp_last'):.3f} for citations per last-author paper ({ptext(pv('F_active_mixed','lcpp_last'))} and {ptext(pv('F_active_ols_cluster','lcpp_last'))}), and about zero for citations per single-author paper (p = {pv('F_active_mixed','lcpp_single'):.2f} and {pv('F_active_ols_cluster','lcpp_single'):.2f}). These initial associations do not identify pay criteria or show a preference for group leadership over per-paper impact. Testing hypotheses about pay criteria requires verified appointments, salary rates, an unselected faculty population, and a longitudinal design, and no such test is made here. The results are consistent with the hypothesis that institutional rewards give more weight to volume and to last-author work than to the per-paper impact of single-author work, but other hypotheses are also possible.")

P('References of Appendix B','Heading 1')

P('Georgia Department of Audits and Accounts, 2025. Open Georgia: salaries and travel reimbursements, University of Georgia, fiscal years 2015 to 2025. <https://open.ga.gov/>')

P('Ioannidis, J.P.A., 2025. August 2025 data-update for "Updated science-wide author databases of standardized citation indicators". Elsevier Data Repository, version 8. <https://doi.org/10.17632/btchxktzyw.8>')

P('Workbook: UGA_StanfordList_Salaries_FY2021-2025.xlsx (sheets Scientists long, Std factors, Std panel, Imputed retirees, Std dean vs sci, Std by college, Std regression); scripts build/redo_standardized.py, build/compute_ugapay.py, build/make_pay_figures.py, build/build_appendix_s2.py (Appendix C).')

```
def render_docx():
    from manuscript_content import plain
    REFS=json.loads((ROOT/'build/references.json').read_text(encoding='utf-8'))
    from docx import Document
    from docx.shared import Inches,Pt,RGBColor
    from docx.oxml.ns import qn
    from docx.oxml import OxmlElement
    doc=Document();sec=doc.sections[0];sec.left_margin=sec.right_margin=Inches(.8);sec.top_margin=sec.bottom_marg
in=Inches(.75)
    pr=sec._sectPr;ln=OxmlElement('w:lnNumType');ln.set(qn('w:countBy'),'1');ln.set(qn('w:restart'),'continuous')
;pr.append(ln)
    for name,size in [('Normal',10.5),('Title',14),('Heading 1',12),('Heading 2',11),('Caption',10.5)]:
        s=doc.styles[name];s.font.name='Times New
Roman';s.font.size=Pt(size);s.font.color.rgb=RGBColor(0,0,0);s.paragraph_format.space_after=Pt(5)
        tp=doc.styles['Title'].element.get_or_add_pPr()
        for b in list(tp.findall(qn('w:pBdr'))):tp.remove(b)
        for b in BLOCKS:
            if b[0]=='table':
                headers,rows,widths=b[1:]
                t=doc.add_table(rows=1,cols=len(headers));t.style='Table Grid';t.autofit=False
                for c,h in zip(t.rows[0].cells,headers):c.text=h;c.paragraphs[0].runs[0].bold=True
                trPr=t.rows[0]._tr.get_or_add_trPr();h=OxmlElement('w:tblHeader');h.set(qn('w:val'),'true');trPr.appe
nd(h)
                for r in rows:
                    cells=t.add_row().cells
                    for c,v in zip(cells,r):c.text=str(v)
                for row in t.rows:
                    for c,w in zip(row.cells,widths):c.width=Inches(6.9*w)
                    for c in row.cells:
                        for p in c.paragraphs:
                            p.paragraph_format.space_after=Pt(1)
                            for run in p.runs:run.font.size=Pt(9.5)
            elif b[0]=='figure':
                z=doc.add_paragraph();z.add_run().add_picture(str(ROOT/'figures'/b[1]),width=Inches(6.9));doc.add_par
agraph(b[2],'Caption')
```

```

elif b[0]=='Caption':
    p=doc.add_paragraph(plain(b[1],REFS,{}),'Caption')
    p.paragraph_format.space_after=Pt(3);p.paragraph_format.keep_with_next=True
elif b[0]=='Note':
    p=doc.add_paragraph(plain(b[1],REFS,{}))
    p.paragraph_format.space_before=Pt(3)
    else:doc.add_paragraph(plain(b[1],REFS,{}),None if b[0] in ('para','Note') else b[0])
out=ROOT/f'Cieszewski2026_AI_Appendix{APPX}_pay.docx';doc.save(out)
env=os.environ.copy();env.update(TMP=str(ROOT/'build/tmp'),TEMP=str(ROOT/'build/tmp'))
subprocess.run([r'C:' +chr(92)+'Program
Files'+chr(92)+'LibreOffice'+chr(92)+'program'+chr(92)+'soffice.exe', '-env:UserInstallation='+ (ROOT/'build/
lo_profile').as_uri(), '--headless', '--norestore', '--convert-to', 'pdf', '--
outdir', str(ROOT), str(out)], cwd=ROOT, env=env, capture_output=True, timeout=120, check=True)
assert out.with_suffix('.pdf').is_file(), 'Pay appendix PDF was not produced'
strip_pdf_dates(out.with_suffix('.pdf'))
for ext in ('docx', 'pdf'): (ROOT/f'Cieszewski2026_AI_AppendixA_pay.{ext}').unlink(missing_ok=True)
print('Appendix '+APPX+' DOCX/PDF written')

def render_tex():
    from manuscript_content import tex_escape, latex
    from figtrim import fig_width # trimmed PNGs at native width
    E=latex;out=[];cap=None;stop=False
    for b in BLOCKS:
        if b[0]=='Title' or (b[0]=='para' and b[1].startswith('Companion to')):continue
        if b[0]=='Heading 1' and b[1].startswith('References of Appendix '+APPX):stop=True
        if stop:continue
        if b[0]=='Heading 1':
            # no FloatBarrier before B.5: it flushed Tables B3 and B4 onto part-empty float pages
            out.append(chr(92)+'subsection*{' +E(b[1])+'}')
        elif b[0]=='Caption':cap=E(b[1].split(' ',1)[1])
        elif b[0]=='table':
            headers,rows,widths=b[1:];widths=[w/sum(widths) for w in widths];cm=17.78-(len(headers)-1)*.211
            spec='@{'+''.join('>' +chr(92)+'raggedright'+chr(92)+'arraybackslash)p{' +f'{cm*w:.3f}'+ 'cm}' for w
            in widths)+'@{'
            out+=[chr(92)+'begin{table}!htb]',chr(92)+'caption{' +(cap or '')
            +'}',r'\nointerlineskip\centerline{\footnotesize\begin{tabular}{' +spec+'}',chr(92)+'toprule', ' &
            '.join(chr(92)+'textbf{' +E(x)+'}' for x in headers)+' '+chr(92)+chr(92),chr(92)+'midrule']
            out+=[ ' & '.join(E(str(x)) for x in row)+' '+chr(92)+chr(92) for row in rows]
            out+=[chr(92)+'bottomrule',chr(92)+'end{tabular}'],'chr(92)+'end{table}'];cap=None
        elif b[0]=='Note':
            assert out[-1]==chr(92)+'end{table}'
            out.insert(len(out)-1,r'\nointerlineskip\vspace{3pt}{\footnotesize\raggedright '+E(b[1])+r'\par}')
        elif b[0]=='figure':
            # the four-row figure is set at the top or bottom of a text page: without a figure title, with one-
            line x labels and a short caption it no longer fills a page, and a float page of its own left a fifth of that
            page empty
            placement='!t' if 'pay_authorship_output' in b[1] else '!htb'
            out+=[chr(92)+'begin{figure}'+placement,chr(92)+'centering',chr(92)+'includegraphics[width='+f'{fig_w
            idth(ROOT/'figures'/b[1],5.4 if 'deans' in b[1] else 6.0):.3f}'+ 'in]
            {figures/' +b[1]+'}',chr(92)+'caption{' +E(b[2].split(' ',1)[1])+'}',chr(92)+'end{figure}']]
            else:out.append(E(b[1]).replace('Table '+TAXTAB+' in Appendix A','Table~'+chr(92)+'ref{tab:taxonomy} in
            Appendix A')) # in the TeX the number is resolved by LaTeX; TAXTAB serves the standalone DOCX/PDF
            out.append(chr(92)+'FloatBarrier')
            text=chr(10).join(x+chr(10) for x in out)
            (ROOT/'build/appendix_a.tex').write_text(text,encoding='ascii');print('appendix_a.tex written')

if __name__=='__main__':
    render_docx();render_tex()

```

2. build_docx.py (SHA-256 015c2bda7435fb7c...; 86 lines)

"""Render the same manuscript using python-docx when available.

The --fallback flag uses explicit OOXML when the host prohibits the configured Python/python-docx runtime. Backend is recorded, never silently misrepresented.

"""

```

import argparse,json,sys
from manuscript_content import *

```

```

def python_docx_build():
    from docx import Document
    from docx.shared import Inches,Pt,RGBColor
    from docx.enum.text import WD_ALIGN_PARAGRAPH
    from docx.oxml import OxmlElement
    from docx.oxml.ns import qn
    doc=Document();sec=doc.sections[0]
    sec.page_width=Inches(8.5);sec.page_height=Inches(11)

```

```

sec.left_margin=sec.right_margin=Inches(.8);sec.top_margin=sec.bottom_margin=Inches(.75)
ln=OxmlElement('w:lnNumType');ln.set(qn('w:countBy'),'1');ln.set(qn('w:restart'),'continuous');ln.set(qn('w:d
istance'),'360');sec._sectPr.append(ln) # line numbers, as MCFNS requests for review copies
for name,size in [(('Normal',10.5),('Title',15),('Heading 1',12),('Heading 2',11),('Caption',10.5))]:
    s=doc.styles[name];s.font.name='Times New Roman';s.font.size=Pt(size);s.font.color.rgb=RGBColor(0,0,0)
    s.paragraph_format.space_after=Pt(5);s.paragraph_format.line_spacing=1.05
# The installed default template adds a decorative blue rule to Title.
title_pr=doc.styles['Title'].element.get_or_add_pPr()
for border in list(title_pr.findall(qn('w:pBdr'))):title_pr.remove(border)
doc.styles['Normal'].paragraph_format.widow_control=True
for name in ['Heading 1','Heading 2','Caption']:doc.styles[name].paragraph_format.keep_with_next=True
blocks,refs,lookup=load_content()
def p(text,style=None):return doc.add_paragraph(plain(text,refs,lookup),style)
p(blocks[0]['text'],'Title');p(AUTHOR);p(AFFILIATION);p(EMAIL);p('MCFNS 18(1) companion draft')
for b in blocks[1:]:
    k=b['type']
    if k=='section':
        if b['text'].startswith('Appendix '):doc.add_page_break()
        p(b['text'],'Heading 1') # retain the literal Appendix A: heading in the review copy
    elif k=='subsection':p(b['text'],'Heading 2')
    elif k in ('paragraph','equation'):p(b['text'])
    elif k=='where':
        p('where:')
        for i,(_sym,txt) in enumerate(b['items']):doc.add_paragraph('- '+sym+' '+plain(txt,refs,lookup)+('.'
if i==len(b['items'])-1 else ('; and' if i==len(b['items'])-2 else ';'))
    elif k=='enumerate':
        for t in b['items']:doc.add_paragraph(plain(t,refs,lookup),'List Number')
    elif k=='figure':
        z=doc.add_paragraph();z.paragraph_format.keep_with_next=True
        z.add_run().add_picture(str(ROOT/'figures'/b['file']),width=Inches(6.9))
        cap=p('Figure '+str(b['number']))+'.
'+b['caption'],'Caption');cap.paragraph_format.keep_with_next=False
    elif k=='table':
        p('Table '+str(b['number'])+'. '+b['caption'],'Caption')
        t=doc.add_table(rows=1,cols=len(b['headers']));t.autofit=False
        widths=[6.9*w/sum(b['widths']) for w in b['widths']]
        for col,w in zip(t.columns,widths):col.width=Inches(w)
        for rowno,vals in enumerate([b['headers']] + b['rows']):
            cells=t.rows[0].cells if rowno==0 else t.add_row().cells
            trpr=t.rows[rowno].tr.get_or_add_trPr();cant=OxmlElement('w:cantSplit');trpr.append(cant)
            if rowno==0:repeat=OxmlElement('w:tblHeader');trpr.append(repeat)
            for j,(cell,value) in enumerate(zip(cells,vals)):
                cell.width=Inches(widths[j]);cell.text=plain(str(value),refs,lookup)
                tcpr=cell._tc.get_or_add_tcPr();sh=OxmlElement('w:shd');sh.set(qn('w:fill'),'E8ECEA' if
rowno==0 else 'FFFFFF');tcpr.append(sh)
                borders=OxmlElement('w:tcBorders')
                for edge in ['top','left','bottom','right']:
                    el=OxmlElement('w:'+edge);el.set(qn('w:val'),'single');el.set(qn('w:sz'),'4');el.set(qn('
w:color'),'D9D9D9');borders.append(el)
                tcpr.append(borders)
            for pp in cell.paragraphs:
                pp.paragraph_format.space_after=Pt(3);pp.paragraph_format.space_before=Pt(3)
                if b['key']=='position':
                    pp.paragraph_format.keep_with_next=rowno<len(b['rows'])
                for run in pp.runs:run.font.size=Pt(10);run.bold=rowno==0
        p(b['note'])
p('References','Heading 1')
for r in refs:p(r['text']+(' '+r['url']+ '.' if r['url'] else ''))
h=sec.header.paragraphs[0];h.text='Cieszewski | AI as a research instrument | MCFNS companion draft'
h.style=doc.styles['Caption'];h.paragraph_format.keep_with_next=False
fp=sec.footer.paragraphs[0];fp.alignment=WD_ALIGN_PARAGRAPH.RIGHT
field=OxmlElement('w:fldSimple');field.set(qn('w:instr'),'PAGE');fp._p.append(field)
path=ROOT/'Cieszewski2026_AI_draft.docx';doc.save(path)
return 'python-docx'

```

```

def main():
    ap=argparse.ArgumentParser();ap.add_argument('--fallback',action='store_true');args=ap.parse_args()
    if args.fallback:
        from ooxml_fallback import build
        build();backend='standard-library OOXML fallback (python-docx runtime blocked)'
    else:backend=python_docx_build()
    (ROOT/'build/docx_build.json').write_text(json.dumps({'backend':backend,'source':'manuscript.md +
manuscript_content.py + references.json','font_points':10.5,'page_inches':
[8.5,11],'body_width_inches':6.9},indent=2))
    print('DOCX built:',backend)

```

```

if __name__=='__main__':main()

```

3. build_program_appendix.py (SHA-256 9638dc002cd49c40...; 49 lines)

"""Appendices D and E: verbatim listing of the programs that produce and check the results of this paper.

D: the rebuild and revision programs (build/*.py) that produce the Article, Appendix B, the checks, and the figures.

E: the original analysis pipeline as archived in Appendix C (july_pipeline/*.py), unedited.

Writes DOCX and PDF (LibreOffice) into the project directory.

"""

```
import os, subprocess, hashlib
from pathlib import Path
from make_package import PUBLICATION_SCRIPTS
from pdfclean import strip_pdf_dates
from docx import Document
from docx.shared import Pt, Inches, RGBColor
from docx.oxml.ns import qn
from docx.oxml import OxmlElement
ROOT=Path(__file__).resolve().parents[1]
PROGRAMS=PUBLICATION_SCRIPTS
JULY=sorted((ROOT/'AppendixC_package/july_pipeline').glob('*.py'))

def build(name,title,intro,files):
    doc=Document();sec=doc.sections[0];sec.left_margin=sec.right_margin=Inches(.7);sec.top_margin=sec.bottom_margin=Inches(.7)
    for n,sz in [('Normal',10.5),('Title',14),('Heading 1',11)]:
        s=doc.styles[n];s.font.name='Times New Roman';s.font.size=Pt(sz);s.font.color.rgb=RGBColor(0,0,0)
        tp=doc.styles['Title'].element.get_or_add_pPr()
        for b in list(tp.findall(qn('w:pBdr'))):tp.remove(b)
        doc.add_paragraph(title,'Title');doc.add_paragraph(intro)
        doc.add_paragraph('Contents: '+' '.join(f'{i+1}. {p.name} ({p.stat().st_size:,} bytes)' for i,p in enumerate(files)))
        for i,p in enumerate(files,1):
            src=p.read_text(encoding='utf-8',errors='replace')
            doc.add_paragraph(f'{i}. {p.name} (SHA-256 {hashlib.sha256(p.read_bytes()).hexdigest()[:16]})...; {len(src.splitlines())} lines','Heading 1')
            for line in src.splitlines():
                para=doc.add_paragraph();para.paragraph_format.space_after=Pt(0);para.paragraph_format.space_before=Pt(0);para.paragraph_format.line_spacing=1.0
                run=para.add_run(line.replace('\t',' ') if line else ' ');run.font.name='Courier New';run.font.size=Pt(7.5)
                rpr=run._element.get_or_add_rPr();rf=rpr.find(qn('w:rFonts'))
                if rf is None:rf=OxmlElement('w:rFonts');rpr.append(rf)
                for k in ('w:ascii','w:hAnsi','w:cs','w:eastAsia'):rf.set(qn(k),'Courier New')
            out=ROOT/f'{name}.docx';doc.save(out)
            env=os.environ.copy();env.update(TMP=str(ROOT/'build/tmp'),TEMP=str(ROOT/'build/tmp'))
            subprocess.run([r'C:\Program Files\LibreOffice\program\soffice.exe','-env:UserInstallation='+str(ROOT/'build/lo_profile').as_uri(),'--headless','--norestore','--convert-to','pdf','--outdir',str(ROOT),str(out)],cwd=ROOT,env=env,capture_output=True,timeout=600,check=True)
            strip_pdf_dates(out.with_suffix('.pdf'))
            print(name,len(files),'files',(ROOT/f'{name}.pdf').exists())
```

build('Cieszewski2026_AI_AppendixD_programs','Appendix D: Programs of the rebuild and revision (verbatim listing)',

'Companion to Cieszewski (2026), Artificial Intelligence as a Research Instrument and an Editorial Burden, MCFNS 18(1). A verbatim listing of the programs that produce and check the results of this paper, from the files deposited in Appendix C (scripts folder). Run order: compute_normalization.py, revision_checks.py, nash_search.py, make_figures.py, build_tex.py, build_docx.py, render_docx.py, verify_numbers.py; for Appendix B: redo_standardized.py, compute_uapay.py, make_pay_figures.py, pay_sensitivity.py, appendix_a.py, with paynames.py as their shared name key; this listing: build_program_appendix.py. Paths inside the programs refer to the author's machine; the Appendix C README states how to point them at the package.',PROGRAMS)

build('Cieszewski2026_AI_AppendixE_original_pipeline','Appendix E: The original analysis pipeline as archived (verbatim listing)',

'Companion to Cieszewski (2026), MCFNS 18(1). The Python programs of the original project that produced the broad report (S1) and the final report, as archived in Appendix C (july_pipeline folder), unedited. They are listed for the documentary record of the workflow studied in the Article; the verification programs in Appendix D recompute every number of the Article independently of them.',JULY)

Remove superseded listings only after both replacements were rendered successfully.

for stem in

('Cieszewski2026_AI_AppendixC_programs','Cieszewski2026_AI_AppendixD_july_pipeline','Cieszewski2026_AI_AppendixE_july_pipeline'):

for ext in ('pdf','docx'):(ROOT/f'{stem}.{ext}').unlink(missing_ok=True)

4. build_tex.py (SHA-256 5931be3da1372a6f...; 238 lines)

"""Generate the canonical MCFNS TeX from the shared manuscript content (mcfnsApr2024v2 header as in the Weiskittel 18(1) paper), with Appendix A (forestry numbers), Appendix B (pay exercise, from build/appendix_a.tex), and Appendix C (package)

```

in one column after the references. The author edits the .tex by hand; those edits are carried into
build/manuscript.md
(and the other build files) before a rebuild, and write_guarded() never overwrites a hand-edited .tex."""
import json, re, os, hashlib, shutil, sys, time
from manuscript_content import *
from figtrim import fig_width # trimmed PNGs are included at their native width, so the type keeps its size
COLUMN_IN=(7*72-20)/2/72 # \columnwidth of mcfnsApr2024v2: (7in - 20pt)/2
MS_TEX=os.environ.get('MS_TEX','Cieszewski2026_AI.tex') # set MS_MD and MS_TEX to build a variant (Opt-B)
B=chr(92)

def write_guarded(target,text):
    """Write the generated TeX without ever overwriting the author's hand edits.
    build/texout/<name>.generated keeps the bytes of the last generated file. If the file on disk differs from
    it, the
    author has edited the file since the last build: the new text then replaces it only if it reproduces the file
    on
    disk exactly, or if TEX_PORTED_SHA256 equals the SHA-256 of the file on disk, set after its edits were
    carried into
    the sources and checked against build/texout/<name>.candidate.tex. Otherwise the build stops and nothing is
    overwritten. A replaced hand-edited file is first copied to build/tmp/tex_backups/."""
    out=ROOT/'build'/'texout';out.mkdir(exist_ok=True)
    last=out/(target.name+'.generated');cand=out/(target.stem+'.candidate.tex')
    cand.write_text(text,encoding='ascii');new=cand.read_bytes()
    if target.exists():
        disk=target.read_bytes()
        if (not last.exists() or disk!=last.read_bytes()) and disk!=new:
            if os.environ.get('TEX_PORTED_SHA256','')!=hashlib.sha256(disk).hexdigest():
                print(target.name+' has hand edits that are not in the sources. Nothing was overwritten. Carry
the edits into '
                        'build/manuscript.md (or the build file that makes that part), compare
'+cand.relative_to(ROOT).as_posix()+
                        ' with '+target.name+', and set TEX_PORTED_SHA256 to the SHA-256 of '+target.name+'.')
                sys.exit(1)
        keep=ROOT/'build'/'tmp'/'tex_backups';keep.mkdir(parents=True,exist_ok=True)
        shutil.copy2(target,keep/(target.stem+'_'+time.strftime('%Y%m%d-%H%M%S')+'.tex'))
        target.write_bytes(new);last.write_bytes(new)

PREAMBLE=r'''% MCFNS 18(1) companion manuscript. Generated by build/build_tex.py from build/manuscript.md,
% build/manuscript_content.py, build/references.json and build/appendix_a.tex; hand edits are carried into them
before a rebuild.
\documentclass[twocolumn,letterpaper,sort]{article} \pagestyle{myheadings}
\pdfminorversion=7
\pdfinfo{mitdate=1}
\pdftrailerid{}
\pdfsuppresswarningpagegroup=1
\usepackage[T1]{fontenc}
\usepackage{lmodern} % vector fonts with a searchable text layer (the bitmap EC fonts broke copying, search,
and indexing)
\usepackage[utf8]{inputenc}
\usepackage[breaklinks]{hyperref}
\usepackage{graphicx}
\usepackage{amsmath}
\usepackage{booktabs}
\usepackage{array}
\usepackage{natbib}
\usepackage{caption}
\usepackage{float}
\usepackage{placeins} % FloatBarrier keeps the appendix floats inside their subsections
\usepackage{mcfnsApr2024v2}
\usepackage{xurl} % long URLs in the reference list may break anywhere
\usepackage{dblfloatfix} % full-width floats may sit at the top or bottom of the page after their citation
\usepackage{balance} % \balance is issued at the end of the references: balances the last two-column page
before the one-column appendices
\widowpenalty=10000 \clubpenalty=10000
\predisplaypenalty=10000 \displaywidowpenalty=10000 \postdisplaypenalty=10000 % an equation stays with its
introductory sentence and with "where:" (author's rule)
\makeatletter\newenvironment{wherelist}{\@beginparpenalty=\@M\begin{description}}{\end{description}}\makeatother
% no page break between "where:" and the first variable
\setlength{\bibsep}{2pt plus 1pt} % compact reference list
\captionsetup{font=normalsize,labelfont=bf,strut=0} % strut=0: the skips below are measured from the caption's
glyphs, not from a strut
\captionsetup[table]{skip=3pt} % MCFNS house rule (Cieszewski): table caption 3 pt above the table (the tabular
follows with \nointerlineskip)
\captionsetup[figure]{skip=9pt} % figure caption 9 pt below the graph
\setlength{\tabcolsep}{3pt}
\renewcommand{\arraystretch}{1.15}
\setcounter{topnumber}{4}
\setcounter{dbltopnumber}{4}
\setcounter{totalnumber}{6}
\renewcommand{\topfraction}{0.95}

```

```

\renewcommand{\dbltopfraction}{0.95}
\renewcommand{\bottomfraction}{0.8}
\setcounter{bottomnumber}{2}
\renewcommand{\textfraction}{0.05}
\renewcommand{\floatpagefraction}{0.85}
\renewcommand{\dblfloatpagefraction}{0.7}

% MCFNS preamble
\year=2026 \setcounter{page}{36} \def\issueno{1} % MCFNS 18(1): Vol. 18 = 2026-2008, Issue 1 -> "Published:
Apr. 30, 2026" (mcfnsApr2024v2); first page 36: third paper of the issue (MCFNS numbers pages through the volume)
\def\submit{Mar.~30, 2026} % EDITOR: set the submission date
\def\accept{Apr.~22, 2026}
\def\lasterrata{Apr.~28, 2026}
\def\citename{Cieszewski}
\def\citeetal{}
\def\citeemail{mail@cjci.net}
\def\editors{Editor: \href{mailto:Bogdan.Strimbu@oregonstate.edu} {Bogdan Strimbu}}
\author{ \href{mailto:\citeemail}{Chris~J.~\citename}} }
\affiliation{\small \it Quantfor Co., Watkinsville, GA, USA} % EDITOR: affiliation given by the author
'''

```

```

APPENDIX_B=[
('Reproducibility package and program listings (Appendices D and E)',None),
('Appendix C is the archive Cieszewski2026_AI_AppendixC_package.zip published with this paper
[[cite:Record_2026]]. It contains the unchanged cohort extract, StanfordList_Forestry_Ecology_career_2020-
2024.xlsx, with 23,374 author-release records from versions 3, 5, 6, 7, and 8 of Elsevier Data Repository record
10.17632/btchxktzyw. It also contains the whole-database country-by-subfield-by-year file,
world_subfield_year.csv, and the original outputs used for verification: auth6.json, authorship.json,
integrity.json, and refs.json. The record includes the text of the broad report (S1), the final report, the
handoff, and a running log kept during the work, with instructions quoted or paraphrased. Original figure files
are included with their hashes, among them ff4_group.png, the unused count-share plot that an earlier draft of
this paper mistook for the report figure (Section A.5). The pay workbook of Appendix B, the checked college list
college_checked.csv with its sources, and the original, unedited analysis programs are also included. Programs
that recompute every numerical table cell and figure of this paper are compute_normalization.py,
revision_checks.py, nash_search.py, make_figures.py, manuscript_content.py, build_tex.py, and verify_numbers.py.
Those for Appendix B are redo_standardized.py, compute Ugapay.py, make_pay_figures.py, pay_sensitivity.py, and
appendix_a.py, with paynames.py as their shared name key. Generated outputs are statistics.json,
normalization.json, annual_ri.csv, revision_checks.json, nash_search.json, rank_vectors.json, ugapay.json,
pay_figures.json, pay_sensitivity.json, and verify_output.txt. SHA-256 hashes are provided for every file. The
README gives the field-name mapping and selection conditions.',None),
('Appendix D (Cieszewski2026_AI_AppendixD_programs.pdf) lists the verification programs verbatim. Appendix E
(Cieszewski2026_AI_AppendixE_original_pipeline.pdf) lists the 86 original analysis programs verbatim. Both are
separate files because of their length. The run order is compute_normalization.py, revision_checks.py,
nash_search.py (requires the raw releases), make_figures.py, redo_standardized.py, compute Ugapay.py,
make_pay_figures.py, pay_sensitivity.py, and appendix_a.py. The source text of this paper is not included, so
build_tex.py and verify_numbers.py cannot be rerun from the package. The output of the final check,
verify_output.txt, ends with RESULT PASS.',None)]

```

```

def attach_floats(body):
    """Move each float environment into the nearest preceding paragraph, immediately after the cross-reference
    that
    first cites it (the \ref{tab:key} or \ref{fig:key} token). A float that LaTeX meets inside a paragraph is
    attached
    to the line containing that point, so a paragraph or sentence that straddles a page break no longer pushes
    the
    float a page later (house rule: after, and as close as possible to, the first mention). Floats not referenced
    in
    the preceding paragraph stay where the manuscript placed them; source order (and so numbering) is
    unchanged."""
    out=[]
    for item in body:
        m=re.match(r'\\begin\{(table|figure)*\\}',item)
        if not m: out.append(item);continue
        lab=re.search(r'\\label\{((?:tab|fig):[^\}]+)\}',item).group(1)
        # nearest preceding paragraph (skip section commands and other floats)
        for j in range(len(out)-1,-1,-1):
            prev=out[j]
            if re.match(r'\\(section|subsection|begin|end|label|balance)',prev): continue
            k=prev.find('\\ref{'+lab+'}')
            if k<0: break
            cut=k+len('\\ref{'+lab+'}')
            m3=re.match(r'[A-Z]?[()\\.,;:]*',prev[cut:]);cut+=m3.end() # keep a panel letter (Figure 1A) and
            closing punctuation with the reference
            # keep manuscript order when several floats are cited at the same point: go past floats already
            attached there
            while True:
                m2=re.match(r'\\s*\\begin\{(table|figure)*\\}.*?\\end\{\\1*\\}',prev[cut:],re.S)
                if not m2: break
                cut+=m2.end()
            out[j]=prev[cut:]+ '\\n'+item+'\\n'+prev[cut:].lstrip()

```

```

        item=None;break
        if item is not None: out.append(item)
    return out

def build():
    blocks,refs,lookup=load_content();title=blocks[0]['text']
    abs_pars=[];in_abs=False    # every paragraph between "## Abstract" and the next heading (the abstract may run
    to several paragraphs)
    for b in blocks[1:]:
        if b['type']=='section':in_abs=(b['text']=='Abstract');continue
        if in_abs and b['type']=='paragraph':abs_pars.append(b['text'])
    abstract='\n\n'.join(abs_pars);kw=None
    body=[];seen_abstract=False;section=None;skip=False;appendix_start=None
    for b in blocks[1:]:
        kind=b['type']
        if kind=='section':
            current=b['text']
            if current=='Abstract':skip=True;continue
            if current=='Keywords':skip='kw';continue
            skip=False
            m=re.match(r'^\d+ (.*)$',current)
            app=re.match(r'^Appendix [A-Z]: (.*)$',current)
            if app:
                if appendix_start is None:appendix_start=len(body)
                body.append('\section{'+latex(app.group(1))+'}');continue
                body.append('\section{'+latex(m.group(1))+'}') if m else ('\section{'+latex(current)+'}')
            elif skip=='kw' and kind=='paragraph':kw=b['text'];skip=False
            elif skip:continue
            elif kind=='subsection':
                if re.match(r'^[A-Z]\.\d+ ',b['text']):
                    # no \FloatBarrier here: a barrier before every subsection flushed Table A2 and Figure A1 onto
                    two
                    # float pages; without it the deferred floats top the next page and the text runs on below
                    body.append('\subsection{'+latex(b['text'])+'}')
                    else:body.append('\subsection{'+latex(re.sub(r'^\d+\.\d+ ','',b['text']))+'}')
            elif kind=='paragraph':body.append(latex(b['text']))
            elif kind=='equation':
                # the display ends the paragraph that introduces it (no blank line before it): with
                \predisplaypenalty and
                # \displaywidowpenalty at 10000 no page break can separate the introductory sentence from the
                equation
                eq='\begin{equation}\n'+b['tex']+'\n\\label{eq:ri}\n\\end{equation}'
                if body and not body[-1].rstrip().startswith('\n'):body[-1]+='\n'+eq
                else:body.append(eq)
            elif kind=='where':
                # variable list in the form of the author's MCFNS papers (15(1)): "where:" on the line after the
                equation
                # (same paragraph, so no indent), then a description list whose labels are "- $symbol$" and whose
                items
                # end with ";", "; and", and "."
                items=b['items'];lines=['where:',r'\begin{wherelist}']
                for i,(sym,txt) in enumerate(items):lines.append(r'\item[- $'+sym+'$ '+latex(txt)+'.' if
                i==len(items)-1 else ('; and' if i==len(items)-2 else ';'))
                lines.append(r'\end{wherelist}');body[-1]+='\n'+'\n'.join(lines)
                elif kind=='enumerate':body.append('\begin{enumerate}\n'+'\n'.join('\item '+latex(t) for t in
                b['items'])+'\n\\end{enumerate}')
                elif kind=='table':
                    single=b['key'] in ('checklistB',)    # a short prose table is set in one column so that it can follow
                    its citation on the same page (a table* met in the right column drifts a page)
                    widths=b['widths'];widths=[w/sum(widths) for w in widths];cm=(8.35 if single else 17.78)-
                    (len(widths)-1)*.211
                    spec='@{}'+'.join('>{\raggedright\\arraybackslashp{'+f'{cm*w:.3f}'+cm}' for w in widths)+'@{}'
                    tab=['\begin{table}[!htb]' if single else '\begin{table*}[!tb]','\caption{'+latex(b['caption'])+'}'+
                    '\label{tab:'+b['key']+'}',r'\nointerlineskip\centerline{\footnotesize\begin{tabular}{'+spec+'}',r'\toprule',' &
                    '.join('\textbf{'+latex(x)+'}' for x in b['headers'])+r' \\',r'\midrule']
                    tab+=['.join(latex(str(x)) for x in row)+r' \\' for row in b['rows']]
                    tab+=[r'\bottomrule',r'\end{table}'],r'\nointerlineskip\vspace{3pt}{\footnotesize\raggedright
                    '+latex(b['note'])+r'\par}',r'\end{table}' if single else r'\end{table*}'];body.append('\n'.join(tab))
                elif kind=='figure':
                    wmax=7.0 if b['key']=='normalization' else 5.6    # the two-panel country figure spans the text width
                    (7 in); the others are set at 5.6 in
                    body.append('\n'.join(['\\begin{figure*}[!tb]',r'\centering','\
                    \includegraphics[width='+f'{fig_width(ROOT/'figures'/b['file'],wmax):.3f}"+in]{figures/'+b['file']+'}','\
                    \caption{'+latex(b['caption'])+'}\\label{fig:'+b['key']+'}',r'\end{figure*}'])))
                    if appendix_start is None:appendix_start=len(body)
                    appendix_body=body[appendix_start:];body=attach_floats(body[:appendix_start])    # appendix floats stay at
                    their markers, after the citing paragraph (one column, [!htb]); attaching them after the \ref would split
                    sentences

```

```

    appendix_body=[s.replace(r'\begin{table*}[!tb]',r'\begin{table}[!
htb]').replace(r'\end{table*}',r'\end{table}').replace(r'\begin{figure*}[!tb]',r'\begin{figure}[!
htb]').replace(r'\end{figure*}',r'\end{figure}') for s in appendix_body]
    out=[PREAMBLE]
    out.append('\def\yourtitle{{{+latex(title)+}}}')
    out.append('\def\yourkeywords{{{+latex(kw)+}}}')
    out.append('\def\yourabstract\n\n'+latex(abstract)+'\n') # the author's layout of the abstract block
    out.append(r'''\title{\Large\bf\uppercase\yourtitle}
\begin{document}\markright{\hfil{{\href{mailto://\citeemail}{\citename}\citeetal}}~(\issueyear)/\mcfnshead}}
\twocolumn[
\begin{@twocolumnfalse}
\hypersetup{pdftitle={\mcfnshead},pdfauthor={\citename~(\issueyear)},pdfsubject={\yourtitle},pdfkeywords={\yourkw
ords}}
\maketitle
\hrule\begin{abstract}\yourabstract\par\medskip\noindent{\bf Keywords: }\yourkeywords
\end{abstract}\hrule\vspace{.3truein}
\end{@twocolumnfalse}
]''')
    out+=body
    hintfile=ROOT/'build/'('balance_hint.json' if MS_TEX=='Cieszewski2026_AI.tex' else
'balance_hint_vc.json');hint=json.loads(hintfile.read_text()) if hintfile.exists() else {}
    out.append(r'\begin{thebibliography}{99}')
    cited=set(re.findall(r'\cite[pt]?{([^\}]+)}', '\n'.join(body+appendix_body)+(ROOT/'build/
appendix_a.tex')).read_text(encoding='ascii'));cited={k.strip() for g in cited for k in g.split(',')})
    refs=[r for r in refs if r['key'] in cited]
    for r in refs:
        if r['key']==hint.get('balance_before'):out.append(r'\balance')
        out.append('\bibitem[+tex_escape(r['author'])+(r['year'])+]{+r['key']+}\n'+latex(r['text'])+
('\n\\url{+r['url']+}' if r['url'] else ''))
        if not hint:out.append(r'\balance')
        out.append(r'\end{thebibliography}')
        # All manuscript appendix blocks follow the references, in one column.
        out.append(r'''\onecolumn
\raggedbottom
\appendix
\makeatletter\renewcommand{\@secntformat}[1]{Appendix~\csname the#1\endcsname:\ } \makeatother
\setcounter{table}{0}\renewcommand{\thetable}{A\arabic{table}}
\setcounter{figure}{0}\renewcommand{\thefigure}{A\arabic{figure}}''')
    out+=appendix_body
    out.append(r'''\FloatBarrier
\setcounter{table}{0}\renewcommand{\thetable}{B\arabic{table}}
\setcounter{figure}{0}\renewcommand{\thefigure}{B\arabic{figure}}
\section{Public pay records of University of Georgia scientists on the 2024 citation list: a descriptive
exercise}\label{app:pay}''')
    out.append((ROOT/'build/appendix_a.tex').read_text(encoding='ascii'))
    out.append('\section{{{+latex(APPENDIX_B[0][0])}}}\label{app:package}')
    for text,_ in APPENDIX_B[1:]:out.append(latex(text))
    out.extend([r'\label{docend}',r'\end{document}'])
    text='\n\n'.join(out)+'\n';text.encode('ascii')
    write_guarded(ROOT/MS_TEX,text)
    expected=[]
    for b in blocks:
        if b['type']=='table':
            expected.append('Table '+str(b['number'])+'. '+plain(b['caption'],refs,lookup))
            expected.extend(plain(str(x),refs,lookup) for row in [b['headers']] + b['rows'] for x in row)
            expected.append(plain(b['note'],refs,lookup))
        elif b['type']=='figure':expected.append('Figure '+str(b['number'])+'. '+plain(b['caption'],refs,lookup))
        elif b['type']=='where':expected+= [plain(t,refs,lookup) for _,t in b['items']]
        elif b['type']=='enumerate':expected+= [plain(t,refs,lookup) for t in b['items']]
        else:expected.append(plain(b['text'],refs,lookup))
    expected.append('References')
    expected.extend(plain(r['text'],refs,lookup)+(r['url']+'. ' if r['url'] else '') for r in refs)
    if
MS_TEX=='Cieszewski2026_AI.tex':(ROOT/'build/content_manifest.json').write_text(json.dumps({'blocks':blocks,'refe
rences':refs,'lookup':lookup,'expected_text':expected},indent=2),encoding='utf-8')
    print('Built MCFNS TeX:',len(text),'ASCII characters;',len(expected),'content units')

if __name__=='__main__':build()

```

5. check_authorship_fields.py (SHA-256 84c702fc247f314b...; 68 lines)

"""Cross-check of the authorship fields used in Appendix B against the source releases.

The workbook sheet 'Authorship indices (321)' carries the exclusive parts of the release's nested authorship sets:

np_last = npsfl - cpsf (papers on which the person is the last but not the first or sole author), nc_last = ncsfl - ncsf, nps and ncs (single-authored papers and their citations). This script recomputes them from the raw version 8 release (all subfields, the source of the 321 UGA profiles) and from the frozen Forestry/Ecology cohort

```

extract (the 28 UGA profiles it contains), reports any disagreement, and counts the profiles whose npsfl exceeds
np (pre-1960 papers, the publication-window discrepancy of the main text). The slow reads are cached in
build/tmp.
Usage: python build/check_authorship_fields.py
"""
import sys, json, unicodedata
from pathlib import Path
import pandas as pd
ROOT = Path(__file__).resolve().parents[1]
sys.path.insert(0, str(ROOT / 'build'))
from data_core import read_extract, n
norm = lambda s: unicodedata.normalize('NFKD', str(s)).encode('ascii', 'ignore').decode().strip().lower()
idx = pd.read_excel(ROOT / 'UGA_StanfordList_Salaries_FY2021-2025.xlsx', sheet_name='Authorship indices (321)')
def key(name):
    """(surname, first initial) for 'Last, First M.' and 'First M. Last' spellings alike."""
    s = norm(name).replace('.', ' ')
    if ',' in s:
        last, first = [x.strip() for x in s.split(',', 1)]
    else:
        parts = s.split(); last, first = parts[-1], ' '.join(parts[:-1])
    return (last.replace(' ', ''), first[:1])
def compare(rows, get, label):
    by = {}
    for r in rows: by.setdefault(key(get(r, 'authfull')), []).append(r)
    bad = []; miss = []; over = []
    for t in idx.itertuples():
        c = by.get(key(t.auth), [])
        if not c: miss.append(t.auth); continue
        r = next((x for x in c if get(x, 'np') == t.np), c[0]) # two UGA profiles can share surname and
initial; np disambiguates
        np_last = get(r, 'npsfl') - get(r, 'cpsf'); nc_last = get(r, 'ncsfl') - get(r, 'ncsf')
        if np_last != t.np_last or nc_last != t.nc_last or get(r, 'nps') != t.nps or get(r, 'ncs') != t.ncs:
            bad.append((t.auth, np_last, t.np_last, nc_last, t.nc_last, get(r, 'nps'), t.nps, get(r, 'ncs'),
t.ncs))
        if get(r, 'npsfl') > get(r, 'np'): over.append((t.auth, get(r, 'np'), get(r, 'npsfl'), get(r,
'firstyr'))
    print(f'{label}: profiles matched {len(idx) - len(miss)}, unmatched {len(miss)}, disagreements {len(bad)}')
    for b in bad[:10]: print(' disagreement (auth, np_last raw/wb, nc_last raw/wb, nps raw/wb, ncs raw/wb):',
b)
    print(' npsfl > np (pre-1960 papers):', len(over), over[:5])
    return bad
# 1. frozen cohort extract (Forestry/Ecology records only; holds a minority of the 321 UGA profiles)
CACHE = ROOT / 'build' / 'tmp' / 'extract_2024_georgia.json'
if CACHE.exists():
    rows = json.loads(CACHE.read_text(encoding='utf-8'))
else:
    rows = [r for r in read_extract() if str(r.get('year')) == '2024' and 'georgia' in str(r.get('inst_name',
')).lower()]
    CACHE.parent.mkdir(exist_ok=True); CACHE.write_text(json.dumps(rows), encoding='utf-8')
rows = [r for r in rows if 'university of georgia' in str(r.get('inst_name', '')).lower()]
compare(rows, lambda r, k: r.get(k) if k in ('authfull', 'firstyr') else n(r, k), 'frozen cohort extract
(Forestry/Ecology only)')
# 2. raw version 8 release (all subfields)
RAW = Path(r'C:
\D\Dropbox\Administrative\FAR\StanfordList\V8_2024\Table_1_Authors_career_2024_pubs_since_1788_wopp_extracted_202
508.xlsx')
CACHE8 = ROOT / 'build' / 'tmp' / 'v8_georgia.json'
if CACHE8.exists() or RAW.exists():
    if CACHE8.exists():
        raw = json.loads(CACHE8.read_text(encoding='utf-8'))
    else:
        from openpyxl import load_workbook
        wb = load_workbook(RAW, read_only=True); ws = wb['Data'] if 'Data' in wb.sheetnames else
wb[wb.sheetnames[0]]
        it = ws.iter_rows(values_only=True); head = [str(h) for h in next(it)]
        want = ['authfull', 'inst_name', 'np6024', 'nps', 'cpsf', 'npsfl', 'ncs', 'ncsf', 'ncsfl', 'firstyr']
        pos = {w: head.index(w) for w in want if w in head}
        raw = [{w: row[i] for w, i in pos.items()} for row in it if row[pos['inst_name']] and 'university of
georgia' in str(row[pos['inst_name']]).lower()]
        CACHE8.write_text(json.dumps(raw), encoding='utf-8')
    compare(raw, lambda r, k: r['np6024'] if k == 'np' else r[k], 'raw version 8 release (all subfields)')
else:
    print('raw version 8 release not on disk; second pass skipped')

```

6. check_layout.py (SHA-256 6a421b6880f6274d...; 232 lines)

"""Measure the house typesetting rules on a compiled PDF.

Usage: python build/check_layout.py [vc] (vc = the archived verification-case draft)

```

Reports, per table: caption-to-top-rule gap and bottom-rule-to-note gap (target 3 pt each);
per figure: image-to-caption gap (target 9 pt); float page vs. first-mention page; trailing floats;
balance of the last two-column page; appendix titles; text-size range inside figures vs. body size.
"""
import sys,re,json
from pathlib import Path
ROOT=Path(__file__).resolve().parents[1];B=ROOT/'build'
sys.path.insert(0,str(B/'vendor'));import pymupdf as fitz
VARIANT=sys.argv[1] if len(sys.argv)>1 else ''
TEX='Cieszewski2026_AI_verification_case.tex' if VARIANT=='vc' else 'Cieszewski2026_AI.tex'
doc=fitz.open(B/'texout'/TEX.replace('.tex','.pdf'))
tex=(ROOT/TEX).read_text(encoding='utf-8')
ok=True
def bad(msg):
    global ok; ok=False; print(' FAIL',msg)

# ---- body font size ----
sizes={}
for p in doc[1:6]:
    for b in p.get_text('dict')['blocks']:
        for l in b.get('lines',[]):
            for s in l['spans']:
                sizes[round(s['size'],1)]=sizes.get(round(s['size'],1),0)+len(s['text'])
body=max(sizes,key=sizes.get);print('body font size',body,'pt')

# ---- captions, rules, notes ----
def caption_lines(page):
    out=[];blocks=page.get_text('dict')['blocks'];weight={}
    for b in blocks:
        for l in b.get('lines',[]):
            for s in l['spans']: weight[s['font']]=weight.get(s['font'],0)+len(s['text'])
    body_font=max(weight,key=weight.get) if weight else None
    for b in blocks:
        for l in b.get('lines',[]):
            t=''.join(s['text'] for s in l['spans']).strip()
            m=re.match(r'(Table|Figure)\s+([A-Z]?[d+])[:.]',t)
            # a caption label is set in the bold label font (labelfont=bf), not in the page's body font: running
            # can also start a line with "Table A4." (the PDF names its fonts F41, F60, ..., so the name says
            # nothing)
            if m and l['spans'][0]['font']!=body_font: out.append((m.group(1),m.group(2),l['bbox'],l['spans'][0]
            ['size']))
    return out
def hrules(page):
    """Horizontal rules (thin filled rects or lines), as (y,x0,x1)."""
    rs=[]
    for d in page.get_drawings():
        r=d['rect']
        if r.height<1.5 and r.width>60: rs.append((round(r.y0,2),r.x0,r.x1))
    return sorted(set(rs))
def text_lines(page):
    out=[]
    for b in page.get_text('dict')['blocks']:
        for l in b.get('lines',[]):
            t=''.join(s['text'] for s in l['spans']).strip()
            if t: out.append((l['bbox'],t,l['spans'][0]['size']))
    return out
def images(page):
    return [page.get_image_bbox(im) for im in page.get_images(full=True)]

Z=10
def ink_gaps(page,y,x0,x1):
    """Render 15 pt above and below a horizontal rule; return (gap above, gap below) from the rule to the nearest
    ink."""
    clip=fitz.Rect(x0,y-15,x1,y+15)
    pix=page.get_pixmap(matrix=fitz.Matrix(Z,Z),clip=clip,colorspace=fitz.csGRAY)
    w,h=pix.width,pix.height;s=pix.samples
    rows=[sum(1 for v in s[r*w:(r+1)*w] if v<128)/w for r in range(h)]
    rule=[i for i,f in enumerate(rows) if f>0.8]
    if not rule: return None,None
    r0,r1=rule[0],rule[-1]
    above=[i for i in range(0,r0) if rows[i]>0];below=[i for i in range(r1+1,h) if rows[i]>0]
    return ((r0-above[-1]-1)/Z if above else None,(below[0]-r1-1)/Z if below else None)

print('\nTables, measured on the rendered page from rule to nearest ink (caption above top rule, note below
bottom rule; target 3 pt):')
seen_t={}
for pi,page in enumerate(doc):
    caps=caption_lines(page);rules=hrules(page)

```

```

for kind,num,bbox,sz in caps:
    if kind!='Table' or num in seen_t: continue
    seen_t[num]=pi+1
    x0,y0,x1,y1=bbox
    nextcap=min([c[2][1] for c in caps if c[2][1]>y1+1]+[page.rect.height])
    tab=[r for r in rules if r[0]>y1-1 and r[0]<nextcap and r[1]<x0+5]
    if not tab: bad(f'Table {num} p{pi+1}: no rules found below caption');continue
    top=tab[0];rw=top[2]-top[1]
    # the note is a small-type line below the table's last rule that starts at the table's left edge and runs
    # across most of its width (a cell of the first column is narrow); a table without a note carries its
    # source in the caption (appendix tables) and only the caption gap is checked
    lines=text_lines(page)
    note=next((l for l in sorted(lines,key=lambda l:l[0][1]) if l[0][1]>tab[-1][0] and l[0][1]<nextcap and
abs(l[0][0]-top[1])<2 and l[2]<body-0.5 and (l[0][2]-l[0][0])>0.6*rw and l[0][1]-tab[-1][0]<8),None)
    if note is None:
        ga,_=ink_gaps(page,top[0],top[1],top[2])
        print(f' Table {num} p{pi+1}: caption ink -> top rule {ga} pt; no note (source in caption)')
        if ga is None or abs(ga-3)>0.6: bad(f'Table {num} caption gap {ga}')
        continue
    bot=tab[-1]
    ga,_=ink_gaps(page,top[0],top[1],top[2]);_,gb=ink_gaps(page,bot[0],bot[1],bot[2])
    print(f' Table {num} p{pi+1}: caption ink -> top rule {ga} pt; bottom rule -> note ink {gb} pt')
    if ga is None or abs(ga-3)>0.6: bad(f'Table {num} caption gap {ga}')
    if gb is None or abs(gb-3)>0.6: bad(f'Table {num} note gap {gb}')

print('\nFigures (image bottom to caption top; target 9 pt):')
seen_f={};figsizes={}
for pi,page in enumerate(doc):
    caps=caption_lines(page);ims=images(page)
    for kind,num,bbox,sz in caps:
        if kind!='Figure' or num in seen_f: continue
        seen_f[num]=pi+1
        x0,y0,x1,y1=bbox
        # column-width figures: image in left minipage, caption in right; gap rule then N/A (side by side)
        side=[im for im in ims if im.x1<x0-5 and im.y0<y1 and im.y1>y0-40]
        above=[im for im in ims if im.y1<=y0+1 and im.x1>x0-5]
        if side and not above:
            print(f' Figure {num} p{pi+1}: caption beside image (column layout)');continue
        if not above: print(f' Figure {num} p{pi+1}: no image above caption?');continue
        im=max(above,key=lambda r:r.y1)
        # ink: first dark row below the image bottom within the caption's width
        clip=fitz.Rect(x0,im.y1,x1,y1+2)
        pix=page.get_pixmap(matrix=fitz.Matrix(Z,Z),clip=clip,colorspace=fitz.csGRAY);w=pix.width;s=pix.samples
        rows=[r for r in range(pix.height) if any(v<128 for v in s[r*w:(r+1)*w])]
        g=rows[0]/Z if rows else None
        print(f' Figure {num} p{pi+1}: image bottom -> caption ink {g} pt (bbox {y0-im.y1:.1f})')
        if g is None or abs(g-9)>0.6: bad(f'Figure {num} caption gap {g}')

# ---- float position vs. first mention ----
print('\nFloat page vs. first mention in text:')
app_start=next(i+1 for i,pg in enumerate(doc) if i>0 and 'Appendix A:' in pg.get_text())
ment={}
for pi,page in enumerate(doc):
    # body-size text only: a "Table 6" inside a table cell or note (small type) is a cross-reference or a quoted
    # July-report label, not a first mention in the text
    txt=' '.join(l[1] for l in text_lines(page) if l[2]>=body-0.5)
    for m in re.finditer(r'(Table|Figure|Fig\.)s?\s+([A-Z]?[d+])',txt):
        if re.search(r'(report'?s|S1)\s*$",txt[max(0,m.start()-12):m.start()]): continue # a July-report float,
not one of ours
        k=('Figure' if m.group(1).startswith('Fig') else 'Table',m.group(2)) # "Fig. 1A" in the text counts as
a mention of Figure 1
        ment.setdefault(k,[]).append(pi+1)
        for m in re.finditer(r'(Tables|Figures)\s+([A-Z]?[d+])\s*(?:and|,|-)\s*([A-Z]?[d+])',txt):
            for n in (m.group(2),m.group(3)): ment.setdefault((m.group(1)[:1],n),[]).append(pi+1)
for kind,seen in (('Table',seen_t),('Figure',seen_f)):
    for num,fp in sorted(seen.items(),key=lambda kv:(len(kv[0]),kv[0])):
        pages=sorted(set(ment.get((kind,num),[])))
        if num[1].isalpha(): pages=[q for q in pages if q>=app_start] # appendix floats: mentions inside the
appendix (main-text notes point forward to them)
        first=pages[0] if pages else None
        flag='' if first is None or fp>=first else ' <-- float before first mention'
        if first is not None and fp-first>1:
            # pages between the mention and the float must be occupied by earlier floats; an empty one means the
float could have been closer
            between=list(range(first+1,fp))
            empty=[q for q in between if not caption_lines(doc[q-1])]
            flag=f' <-- pages {empty} between mention and float carry no float' if empty else f' (pages
{between} carry earlier floats)'
            if not empty: print(f' {kind} {num}: float p{fp}, mentions p{pages}{flag}');continue
            print(f' {kind} {num}: float p{fp}, mentions p{pages}{flag}')

```

```

        if flag: bad(f'{kind} {num} placement')

# ---- trailing floats / appendix titles / balance ----
print('\nAppendix titles:')
for pi,page in enumerate(doc):
    for l in text_lines(page):
        if re.match(r'Appendix [A-Z][:~]',l[1]) and l[2]>body: print(f'  p{pi+1}: {l[1][:70]}')
last=None
for i,page in enumerate(doc):
    if 'Appendix A:' in page.get_text() and i>0: last=i-1;break
page=doc[last];W=page.rect.width
L=[b for b in page.get_text('blocks') if b[6]==0 and b[0]<W/2 and b[1]>60]
R=[b for b in page.get_text('blocks') if b[6]==0 and b[0]>=W/2 and b[1]>60]
lb=max(b[3] for b in L);rb=max(b[3] for b in R)
print(f'\nLast two-column page p{last+1}: left column bottom {lb:.1f}, right {rb:.1f}, difference {abs(lb-
rb):.1f} pt')
if abs(lb-rb)>26: bad('last two-column page not balanced') # within two lines: club/widow penalties forbid a
one-line split of a reference
txt=page.get_text()
if re.search(r'^(Table|Figure) \d+[:~]',txt,re.M): bad('float on last two-column page after references?')
# trailing floats: any main-text float on a page after the references page
for pi in range(last+1,len(doc)):
    for kind,num,bbox,sz in caption_lines(doc[pi]):
        if not num[1].isalpha(): bad(f'{kind} {num} trails on p{pi+1}')
# last page of the whole document: any float after the last text?
lastpage=doc[len(doc)-1]
print('last page',len(doc),'has',len(images(lastpage)), 'images; last lines:',[l[1][:50] for l in
text_lines(lastpage)[-2:]])

# ---- figure font size (from PNG dpi and matplotlib pt) ----
print('\nFigure text size (matplotlib font pt x scale):')
for m in re.finditer(r'\\includegraphics\\[width=([\\d.]+)in\\]\\{([\\^]+)\\}',tex):
    w=float(m.group(1));f=m.group(2)
    for cand in (ROOT/'build'/f,ROOT/f,ROOT/'figures'/f):
        if cand.exists():
            px=fitz.open(cand)[0].rect.width # points at 72 dpi?
            import PIL.Image as I
            im=I.open(cand);dpi=im.info.get('dpi',(300,300))[0];win=im.width/dpi
            scale=w/win
            print(f' {f}: native {win:.2f}in -> {w}in, scale {scale:.2f}; 9pt->{9*scale:.1f} 8pt->{8*scale:.1f}
(body {body})')
            if 9*scale>body+0.3: bad(f'{f} text larger than body')
            break
# ---- no calendar date after the Apr. 30, 2026 publication date in the text (header lines of page 1 excepted)
----
print('\nDates after the publication date (Apr. 30, 2026) in the text:')
MON=['January','February','March','April','May','June','July','August','September','October','November','December']
ABBR={m[:3]:i+1 for i,m in enumerate(MON)};ABBR['Sept']=9
late=[]
for pi,page in enumerate(doc):
    t=re.sub(r's+',',',page.get_text())
    if pi==0: t=re.sub(r'(Submitted|Accepted|Published|Last Correction): [A-Z][a-z]{2,4}\\.? \\d{1,2}, \\d{4}',',',t)
    t=t.replace('July to June','') # the definition of the state fiscal year is not a date
    for m in re.finditer(r'\\b(?:\\d{1,2}) )?(January|February|March|April|May|June|July|August|September|October|
November|December|Jan|Feb|Mar|Apr|Jun|Jul|Aug|Sep|Sept|Oct|Nov|Dec)\\b\\.?(?: (\\d{1,2})\\b,)?(?: (\\d{4}))?',t):
        mon=ABBR[m.group(2)[:4]] if m.group(2).startswith('Sept') else m.group(2)[:3];yr=int(m.group(4)) if
m.group(4) else None
        if (yr is None and mon>4 and m.group(1)) or (yr is not None and (yr>2026 or (yr==2026 and mon>4))) or (yr
is None and m.group(2) in MON[4:] and not m.group(3)):
            late.append(f'p{pi+1}: {m.group(0)}')
        late+= [f'p{pi+1}: {d}' for d in re.findall(r'2026-(?:0[5-9]|1[0-2])-\\d\\d',t)]
print(' none' if not late else '\n'.join(' '+x for x in late))
if late: bad('calendar dates after the publication date in the text')

# ---- an equation stays with its introductory sentence and its "where:" (the author's rule) ----
print('\nEquations (introductory sentence, number, and "where:" on one page):')
nums=[]
for pi,page in enumerate(doc):
    lines_=[l for bl in page.get_text('dict')['blocks'] for l in bl.get('lines',[]) if ''.join(s_['text'] for s_
in l['spans']).strip()]
    right=max((l['bbox'][2] for l in lines_),default=0)
    for l in lines_:
        sps=sorted((s_ for s_ in l['spans'] if s_['text'].strip()),key=lambda s_:s_['bbox'][0])
        for k_,sp in enumerate(sps):
            # an equation number: "(n)" flush with the right edge of the text block and set off from any text to
its left
            # by more than 20 pt, which an in-text enumeration such as "(1) contemporaneous files" never is
            if re.fullmatch(r'\\(\\d+\\)',sp['text'].strip()) and sp['bbox'][2]>=right-1.5 and (k_==0 or sp['bbox']
[0]-sps[k_-1]['bbox'][2]>20):

```

```

        nums.append((pi,sp['bbox'][1],sp['text'].strip()))
for k,m in enumerate(re.finditer(r'([\^\\n]+\n\\begin\{equation\}.*?\\end\{equation\}(\nwhere:)?',tex,re.S)):
    lead=re.split(r'(?<=[.?!])\s+',re.sub(r'\${^$}*$','',m.group(1)).strip())[-1];words=' '.join(lead.split()
[:4])
    if k>=len(nums): bad(f'equation {k+1}: number not found in the PDF');continue
    pi,y,num=nums[k];lines=sorted(text_lines(doc[pi]),key=lambda z:z[0][1])
    above=re.sub(r'(\w)- (\w)',r'\1\2',' '.join(z[1] for z in lines if z[0][3]<=y+1))
    below=[z[1] for z in lines if z[0][1]>y+1]
    lead_ok=words in above[-400:];where_ok=(not m.group(2)) or any(x.startswith('where:') for x in below[:3])
    print(f' Eq. {num} p{pi+1}: introductory sentence "{lead[:60]}" {"on the same page" if lead_ok else "NOT on
the same page"}'+('; "where:" follows' if m.group(2) and where_ok else ''))
    if not lead_ok: bad(f'Eq. {num} separated from its introductory sentence')
    if not where_ok: bad(f'Eq. {num} separated from its "where:" list')
print('\nRESULT:', 'PASS' if ok else 'FAIL')
sys.exit(0 if ok else 1)

```

7. compile.py (SHA-256 a278945b02b3f83d...; 64 lines)

""One-command typesetting: appendix_a.py -> build_tex.py -> pdflatex x3, then balance the last two-column page.

The balance package needs `\\balance` issued in the FIRST column of the last two-column page. The reference list spans pages, so the position is found from the first compile: the first reference printed on the last two-column page is recorded in `build/balance_hint.json`, `build_tex.py` inserts `\\balance` before that `\\bibitem`, and the document is compiled again. Usage: `python build/compile.py` (from any directory).

```

"""
import subprocess,sys,os,json,re,shutil
VARIANT=sys.argv[1] if len(sys.argv)>1 else '' # '' -> build/manuscript.md -> Cieszewski2026_AI.tex (the paper);
'vc' -> the archived verification-case draft
from pathlib import Path
from pdfclean import strip_pdf_dates
ROOT=Path(__file__).resolve().parents[1];B=ROOT/'build';OUT=B/'texout';OUT.mkdir(exist_ok=True)
sys.path.insert(0,str(B/'vendor'));import pymupdf as fitz
MIKTEX=r'C:\Users\Administrator\AppData\Local\Programs\MiKTeX\miktex\bin\x64'
env=os.environ.copy();env['PATH']=MIKTEX+os.pathsep+env['PATH'];env['MIKTEX_AUTOINSTALL']='no'
TEX='Cieszewski2026_AI_verification_case.tex' if VARIANT=='vc' else 'Cieszewski2026_AI.tex'
env['MS_MD']='manuscript_verification_case.md' if VARIANT=='vc' else 'manuscript.md';env['MS_TEX']=TEX
def run(cmd):
    p=subprocess.run(cmd,cwd=ROOT,env=env,capture_output=True,text=True)
    if p.returncode: print(p.stdout[-3000:]);raise SystemExit(cmd[0]+' failed')
def latex3():
    for _ in range(3): run(['pdflatex','-interaction=nonstopmode','-halt-on-error','-output-directory=build/
texout',TEX])
def last_twocol_first_ref():
    """Return the reference key whose entry starts the left column of the last two-column page."""
    refs=json.loads((B/'references.json').read_text(encoding='utf-8'))
    doc=fitz.open(OUT/TEX.replace('.tex','.pdf'))
    last=None
    for i,page in enumerate(doc):
        if 'Appendix A:' in page.get_text() and i>0: last=i-1;break
    page=doc[last];W=page.rect.width
    # line starts in the left column: a reference that begins inside a text block (after the tail of the previous
    # entry, or after a URL set in another font) is found as well as one that begins a block
    lines=sorted((l['bbox'][1],re.sub(r'\s+', ' ', ' '.join(s['_text'] for s in l['spans']))).strip())
        for bl in page.get_text('dict')['blocks'] for l in bl.get('lines',[]) if l['bbox'][0]<W/2 and
l['bbox'][1]>60)
    # first reference whose author string (before the year) begins a line in this page's left column
    starts=[t[:40] for _,t in lines if t]
    for r in refs:
        head=r['text'][:40]
        if any(s.startswith(head[:25]) for s in starts): return r['key'],last+1
    # the reference list starts in the left column of the last page: \balance goes before the first cited entry
    if 'References' in ' '.join(s for s in starts):
        cited=re.findall(r'\\bibitem\[([^\]]*)\]{([^\]]+)\}',(ROOT/TEX).read_text(encoding='utf-8'))
        if cited: return cited[0],last+1
    return None,last+1
run([sys.executable,str(B/'appendix_a.py')])
hint=B/('balance_hint_vc.json' if VARIANT=='vc' else 'balance_hint.json')
if hint.exists(): hint.unlink()
for stale in (ROOT/TEX.replace('.tex','.aux'),ROOT/TEX.replace('.tex','.out')):
    # MiKTeX reads a .aux in the current directory before the one in -output-directory: a leftover from a manual
    # compile in the project root would feed stale table/figure numbers into every \ref
    if stale.exists(): stale.unlink()
run([sys.executable,str(B/'build_tex.py')]);latex3()
key,page=last_twocol_first_ref()
if key:
    hint.write_text(json.dumps({'balance_before':key,'last_twocolumn_page':page}))
    run([sys.executable,str(B/'build_tex.py')]);latex3()
    print('balanced with \\balance before',key,'on page',page)

```

```

else: print('no reference starts the left column of the last two-column page; \\balance left at the end of the
list')
compiled=OUT/TEX.replace('.tex','.pdf')
strip_pdf_dates(compiled)
with fitz.open(compiled) as doc:
    print('compiled:',compiled.name,len(doc),'pages')
if not VARIANT:
    shutil.copy2(OUT/'Cieszewski2026_AI.pdf',ROOT/'Cieszewski2026_AI_MCFNS18-1.pdf')

```

8. compute_normalization.py (SHA-256 78aefb0c4d70f896...; 17 lines)

```

"""Recompute the analytical exhibit and 2024/pooled country RI from source files."""
import json, csv
from data_core import ROOT, read_extract, calculate, normalization

def main():
    rows=read_extract();stats=calculate(rows);norm=normalization(rows)
    (ROOT/'build/statistics.json').write_text(json.dumps(stats,indent=2),encoding='utf-8')
    (ROOT/'build/normalization.json').write_text(json.dumps(norm,indent=2),encoding='utf-8')
    with (ROOT/'build/normalization_2024.csv').open('w',newline='',encoding='utf-8') as f:
        writer=csv.DictWriter(f,fieldnames=list(norm['2024']['records'])
    [0]));writer.writeheader();writer.writerows(norm['2024']['records'])
    print('Extract
rows:',len(rows),'Forestry:',stats['Forestry_n'],'Ecology:',stats['Ecology_n'],'cohort:',stats['cohort_n'])
    print('2024 world denominator:',norm['2024']['all_total'],'Forestry:',norm['2024']['forestry_total'])
    print('Rank diagnostics:',stats['ranks'])
    print('Non-nested cells:',stats['non_nested'])
    for r in norm['2024']['records'][:25]: print(r['country'],r['forestry'],r['all'],round(r['ri'],4))

if __name__=='__main__':main()

```

9. compute_ugapay.py (SHA-256 37ae1bae2651d79a...; 83 lines)

""Numbers for Section 6 (Table ugapay and the institutional-evidence paragraphs), standardized-pay version.

```

Reads the companion workbook sheets written by build/redo_standardized.py ('Std panel', 'Std factors',
'Std dean vs sci', 'Std by college', 'Authorship indices (321)') and writes build/ugapay.json.
Standardized pay = pay / (college five-year mean of non-administrative list scientists) x (university-wide
mean of the same group). Retirees are carried forward from their last full-time year. Requires pandas,
openpyxl, statsmodels.
"""
import json, os, warnings
import numpy as np, pandas as pd
import statsmodels.formula.api as smf
from pathlib import Path
warnings.filterwarnings('ignore')
ROOT = Path(__file__).resolve().parents[1]
import sys; sys.dont_write_bytecode = True; sys.path.insert(0, str(ROOT / 'build')); from paynames import
join_idx # accent-insensitive name join
XL = Path(os.environ.get('PAY_XL', ROOT / 'UGA_StanfordList_Salaries_FY2021-2025.xlsx'))
OUT = Path(os.environ.get('PAY_OUT', ROOT / 'build'))
OUT.mkdir(parents=True, exist_ok=True)
panel = pd.read_excel(XL, sheet_name='Std panel')
fac = pd.read_excel(XL, sheet_name='Std factors')
ratios = pd.read_excel(XL, sheet_name='Std dean vs sci')
byc = pd.read_excel(XL, sheet_name='Std by college')
idx = pd.read_excel(XL, sheet_name='Authorship indices (321)')
summ = json.loads((OUT / 'std_summary.json').read_text())

d = join_idx(panel[panel.use], idx)
d['lstd'] = np.log(d.pay_std); d['lnp'] = np.log(d.np); d['lcpp'] = np.log(d.cpp_all); d['yr'] = d.fy.astype(str)
d['lage'] = np.log((2025 - d.firstyr).clip(lower=1))
# authorship classes: all positions (np, nc); last author (np_last = npsfl - cpsf, nc_last = ncsfl - ncsf, as
stored in the
# workbook and confirmed against the raw release by build/check_authorship_fields.py); single author (nps, ncs)
d['single_pap'] = d.I_pap_single_vs_all; d['single_cit'] = d.I_cit_single_vs_all; d['last_pap'] =
d.share_pap_last; d['last_cit'] = d.share_cit_last
d['lcpp_single'] = np.log(d.cpp_single.replace(0, np.nan)); d['lcpp_last'] = np.log(d.cpp_last.replace(0,
np.nan))
fE = 'lstd ~ lnp + lcpp + lage + single_pap + single_cit + last_pap + last_cit + C(yr)'
fF = 'lstd ~ lnp + lage + lcpp_single + lcpp_last + C(yr)'
dE = d.dropna(subset=['lstd', 'lnp', 'lcpp', 'lage', 'single_pap', 'single_cit', 'last_pap',
'last_cit']).reset_index(drop=True)
dF = dE.dropna(subset=['lcpp_single', 'lcpp_last']).reset_index(drop=True)
mE1 = smf.mixedlm(fE, dE, groups=dE['auth']).fit(reml=True)
mE2 = smf.ols(fE, dE).fit(cov_type='cluster', cov_kwds={'groups': pd.factorize(dE.auth)[0]})

```

```

mF1 = smf.mixedlm(fF, dF, groups=dF['auth']).fit(reml=True)
mF2 = smf.ols(fF, dF).fit(cov_type='cluster', cov_kws={'groups': pd.factorize(dF.auth)[0]})
terms = [('lnp', 'log papers (all positions)', mE1, mE2), ('lcpp', 'log citations per paper (all positions)',
mE1, mE2), ('lage', 'log career age', mE1, mE2),
('single_pap', 'Single-author share of papers', mE1, mE2), ('single_cit', 'Single-author share of
citations', mE1, mE2),
('last_pap', 'Last-author share of papers', mE1, mE2), ('last_cit', 'Last-author share of citations',
mE1, mE2),
('lcpp_single', 'log citations per single-author paper', mF1, mF2), ('lcpp_last', 'log citations per
last-author paper', mF1, mF2)]
rows = [{'term': k, 'label': lab, 'c1': float(m1.params[k]), 'p1': float(m1.pvalues[k]), 'c2':
float(m2.params[k]), 'p2': float(m2.pvalues[k])} for k, lab, m1, m2 in terms]
# active-only robustness for the elasticity range
dA = dE[~dE.imputed]; mA = smf.mixedlm(fE, dA, groups=dA['auth']).fit(reml=True)
clean = ratios[ratios.flag.fillna('') == '']
clean = clean[clean.ratio_dean_to_sci_active.notna()]
out = {'n_obs': int(len(dE)), 'n_persons': int(dE.auth.nunique()), 'n_colleges': int(dE.college.nunique()),
'n_obs_F': int(len(dF)),
'n_active_persons': int(dA.auth.nunique()), 'n_imputed_persons': int(dE[dE.imputed].auth.nunique()),
'rows': rows, 'lnp_active': float(mA.params['lnp']), 'lnp_range': [round(min(mE1.params['lnp'],
mE2.params['lnp']), mF1.params['lnp'], mF2.params['lnp'], mA.params['lnp']), 3), round(max(mE1.params['lnp'],
mE2.params['lnp'], mF1.params['lnp'], mF2.params['lnp'], mA.params['lnp']), 3)],
'U_mean': summ['U_mean'], 'factors': {r.college: {'mean5': float(r.mean5), 'multiplier':
float(r.multiplier_U_over_mean5), 'n_persons': int(r.n_persons)} for r in fac.itertuples() if not
str(r.college).startswith('UNIVERSITY')},
'ratio_min': float(clean.ratio_dean_to_sci_active.min()), 'ratio_max':
float(clean.ratio_dean_to_sci_active.max()),
'ratio_by_college': {c: [float(g.ratio_dean_to_sci_active.min()), float(g.ratio_dean_to_sci_active.max())]
for c, g in clean.groupby('college')},
'by_college': {r.college: {'std_mean_incl_imputed': float(r.std_mean_incl_imputed), 'dean_std_mean':
float(r.dean_std_mean) if pd.notna(r.dean_std_mean) else None, 'n_persons_incl_imputed':
int(r.n_persons_incl_imputed)} for r in byc.itertuples()),
'dean_inc_range': [float(clean.dean_inc_pct.min()), float(clean.dean_inc_pct.max())]}

# Full parameter tables for Supplement S2: every term of each model, plus an active-only model.
def table(m): return {k: {'coef': float(m.params[k]), 'se': float(m.bse[k]), 'p': float(m.pvalues[k])} for k in
m.params.index if not k.startswith('Group')}
mA2 = smf.ols(fE, dA, groups=None).fit(cov_type='cluster', cov_kws={'groups': pd.factorize(dA.auth)[0]})
# observed-pay counterpart of model F: does the pooled result depend on the modeled retiree pay?
dFo = dF[~dF.imputed]
mF3 = smf.mixedlm(fF, dFo, groups=dFo['auth']).fit(reml=True)
mF4 = smf.ols(fF, dFo).fit(cov_type='cluster', cov_kws={'groups': pd.factorize(dFo.auth)[0]})
out['models'] = {'E_mixed': {'n_obs': int(len(dE)), 'n_persons': int(dE.auth.nunique()), 'params': table(mE1),
'sigma_person': float(mE1.cov_re.iloc[0,0])**0.5, 'sigma_resid': float(mE1.scale)**0.5},
'E_ols_cluster': {'n_obs': int(len(dE)), 'n_persons': int(dE.auth.nunique()), 'params': table(mE2), 'r2':
float(mE2.rsquared)},
'F_mixed': {'n_obs': int(len(dF)), 'n_persons': int(dF.auth.nunique()), 'params': table(mF1)},
'F_ols_cluster': {'n_obs': int(len(dF)), 'n_persons': int(dF.auth.nunique()), 'params': table(mF2), 'r2':
float(mF2.rsquared)},
'E_active_mixed': {'n_obs': int(len(dA)), 'n_persons': int(dA.auth.nunique()), 'params': table(mA)},
'E_active_ols_cluster': {'n_obs': int(len(dA)), 'n_persons': int(dA.auth.nunique()), 'params': table(mA2),
'r2': float(mA2.rsquared)},
'F_active_mixed': {'n_obs': int(len(dFo)), 'n_persons': int(dFo.auth.nunique()), 'params': table(mF3)},
'F_active_ols_cluster': {'n_obs': int(len(dFo)), 'n_persons': int(dFo.auth.nunique()), 'params': table(mF4),
'r2': float(mF4.rsquared)}
out['dropped_from_regression'] = sorted(set(d[d.use if 'use' in d else slice(None)].auth) - set(dE.auth)) if
False else sorted(set(panel[panel.use].auth) - set(dE.auth))
out['panel_counts'] = {'persons': int(panel[panel.use].auth.nunique()), 'rows': int(panel.use.sum()),
'rows_observed': int((panel.use & ~panel.imputed).sum()), 'rows_imputed': int((panel.use & panel.imputed).sum()),
'persons_active': int(panel[panel.use & ~panel.imputed].auth.nunique()), 'persons_imputed':
int(panel[panel.use & panel.imputed].auth.nunique()),
'persons_both': int(len(set(panel[panel.use & ~panel.imputed].auth) & set(panel[panel.use &
panel.imputed].auth))),
'persons_never_usable': int(panel.auth.nunique() - panel[panel.use].auth.nunique()), 'panel_persons_total':
int(panel.auth.nunique())}
out.update({k: summ.get(k) for k in ('college_check', 'college_source', 'raise_source')})
(OUT / 'ugapay.json').write_text(json.dumps(out, indent=1), encoding='utf-8')
print(json.dumps({k: v for k, v in out.items() if k not in ('rows', 'factors', 'by_college')}, indent=1))
for r in rows: print(f"{r['label']:42s} {r['c1']:+.3f} ({r['p1']:+.3f}) {r['c2']:+.3f} ({r['p2']:+.3f})")
print({c: [round(x, 2) for x in v] for c, v in out['ratio_by_college'].items()})

```

10. data_core.py (SHA-256 c1ef45f92de1a01e...; 132 lines)

```

"""Read-only source access and deterministic statistics; standard library only."""
from pathlib import Path
import os, csv, json, math, statistics, zipfile
import xml.etree.ElementTree as ET

```

```

ROOT = Path(__file__).resolve().parents[1]
_JULY_SRC = Path(r'C:\Users\Administrator\AppData\Local\Temp\claude\C--D-Dropbox-OJS-18-1-\5931ceb7-66f6-4bbb-97b6-a37204817c7f\scratchpad\src')
# Source root: the July working tree if it still exists, else the frozen copy in the Appendix C package (same files;
# the package keeps the extract at inputs/ and the documentary record at inputs/record/). Override with
CIESZEWSKI_SOURCE_ROOT.
SRC = Path(os.environ.get('CIESZEWSKI_SOURCE_ROOT', str(_JULY_SRC if _JULY_SRC.exists() else ROOT /
'AppendixC_package' / 'inputs')))
def src_file(name):
    """Locate a source file under SRC in either layout (July tree: deliverables/, root; package: root,
record/)."""
    for sub in ('', 'deliverables', 'record', 'data'):
        p = SRC / sub / name
        if p.exists(): return p
    return SRC / name
EXTRACT = src_file('StanfordList_Forestry_Ecology_career_2020-2024.xlsx')
NS = {'s': 'http://schemas.openxmlformats.org/spreadsheetml/2006/main'}
YEARS = [2020, 2021, 2022, 2023, 2024]

def read_extract():
    """Parse the Data worksheet directly, preserving numbers and empty cells."""
    with zipfile.ZipFile(EXTRACT) as z:
        strings=[]
        if 'xl/sharedStrings.xml' in z.namelist():
            strings=[''.join(n.itertext()) for n in ET.fromstring(z.read('xl/sharedStrings.xml'))]
        wb=ET.fromstring(z.read('xl/workbook.xml'))
        rels={r.get('Id'):r.get('Target') for r in ET.fromstring(z.read('xl/_rels/workbook.xml.rels'))}
        sheet=next(x for x in wb.find('s:sheets',NS) if x.get('name')=='Data')
        target=rels[sheet.get('{http://schemas.openxmlformats.org/officeDocument/2006/relationships}id')]
        target=target.lstrip('/') if target.startswith('/') else 'xl/'+target
        rows=[]; header=None
        with z.open(target) as f:
            for event,el in ET.iterparse(f,events=('end',)):
                if el.tag!='{'+NS['s']+'}row': continue
                vals={}
                for c in el:
                    ref=c.get('r',''); letters=''.join(x for x in ref if x.isalpha()); ix=0
                    for x in letters: ix=ix*26+ord(x)-64
                    v=c.find('s:v',NS); value=v.text if v is not None else ''
                    if c.get('t')=='s': value=strings[int(value)]
                    elif c.get('t')=='inlineStr': value=''.join(c.find('s:is',NS).itertext())
                    elif value!='':
                        try: value=float(value); value=int(value) if value.is_integer() else value
                        except ValueError: pass
                    vals[ix]=value
                if header is None: header=[str(vals.get(i,'')) for i in range(max(vals)+1)]
                else: rows.append({h:vals.get(i,'') for i,h in enumerate(header)})
                el.clear()
    return rows

def n(r,k):
    v=r.get(k,0)
    return float(v) if v not in ('',None) else 0.0

def forestry(r): return str(r['sm-subfield-1']).strip()=='Forestry'
def ecology(r): return str(r['sm-subfield-1']).strip()=='Ecology' and str(r['sm-subfield-2']).strip()=='Forestry'
def cohort(rows): return [r for r in rows if forestry(r) or ecology(r)]
def summed(rows,k): return sum(n(r,k) for r in rows)

def rates(rows):
    out={'n':len(rows)}
    for label,p,c in [ ('solo','nps','ncs'), ('sen','cpsf','ncsf'), ('all','np','nc') ]:
        ps,cs=summed(rows,p),summed(rows,c)
        out.update({label+'_p':ps,label+'_ppa':ps/len(rows),label+'_cpa':cs/len(rows),label+'_cpp':cs/ps})
    return out

def rank(values):
    order=sorted(range(len(values)),key=lambda i:-values[i]); out=[0.0]*len(values); i=0
    while i<len(order):
        j=i+1
        while j<len(order) and values[order[j]]==values[order[i]]: j+=1
        for k in range(i,j): out[order[k]]=(i+1+j)/2
        i=j
    return out

def rho(a,b):
    x,y=rank(a),rank(b); mx,my=statistics.mean(x),statistics.mean(y)

```

```

    return sum((u-mx)*(v-my) for u,v in zip(x,y))/math.sqrt(sum((u-mx)**2 for u in x)*sum((v-my)**2 for v in y))

def calculate(rows):
    d=cohort(rows); d24=[r for r in d if r['year']==2024]
    out={'broad':len(rows),'Forestry_n':sum(forestry(r) for r in rows),'Ecology_n':sum(ecology(r) for r in
rows),'cohort_n':len(d),'n2024':len(d24)}
    out['years']={str(y):rates([r for r in d if r['year']==y]) for y in YEARS}
    out['groups']={'cohort':rates(d),'Forestry':rates([r for r in d if forestry(r)]),'Ecology':rates([r for r in
d if ecology(r)])}
    p,c=summed(d,'np'),summed(d,'nc');out['position']={}
    for label,pk,ck,psub,csup in [('Single','nps','ncs',None,None),('First','cpsf','ncsf','nps','ncs'),
('Other','np','nc','cpsf','ncsf')]:
        pv=summed(d,pk)-(summed(d,psub) if psub else 0); cv=summed(d,ck)-(summed(d,csup) if csup else 0)
        out['position'][label]={'p':pv,'c':cv,'paper_share':100*pv/p,'citation_share':100*cv/c,'cpp':cv/pv}
        allv=[n(r,'nc') for r in d24];out['ranks']={};ta=set(sorted(range(len(d24)),key=lambda i:(-allv[i],str(d24[i]
['authfull']))):[25])
        for label,k in [('Key','ncsf1'),('Lead','ncsf'),('Single','ncs')]:
            v=[n(r,k) for r in d24];tb=set(sorted(range(len(d24)),key=lambda i:(-v[i],str(d24[i]['authfull']))):[25])
            s=sorted(v,reverse=True)
            out['ranks'][label]={'rho':rho(allv,v),'overlap':len(ta&tb),'cutoff':s[24],'boundary_tie':s[24]==s[25]}
            ar,sr=rank(allv),rank([n(r,'ncs') for r in d24]);out['people']={}
        for name in ['Bettinger, Pete','Cieszewski, Chris J.','Peterson, Chris Jon']:
            hits=[i for i,r in enumerate(d24) if r['authfull']==name]
            assert len(hits)==1,(name,len(hits))
            i=hits[0];r=d24[i]
            out['people'][name]={**rates([r]),'h':n(r,'h'),'hm':n(r,'hm'),'rank_all':ar[i],'rank_single':sr[i]}
            nc=sorted(allv);nn=len(nc);cs=sum(nc)
            out['integrity']={'self_med':statistics.median(n(r,'self%') for r in d24)*100,'self_max':max(n(r,'self%') for
r in d24)*100,'self_gt25':sum(n(r,'self%')>.25 for r in d24),'cprat_med':statistics.median(n(r,'cprat') for r in
d24),'rw_papers':summed(d24,'np_rw'),'rw_auth':sum(n(r,'np_rw')>0 for r in
d24),'pctcited_med':statistics.median(100*n(r,'np cited')/n(r,'np') for r in
d24),'h_med':statistics.median(n(r,'h') for r in d24),'hm_med':statistics.median(n(r,'hm') for r in
d24),'gini':sum((2*i-nn-1)*v for i,v in
enumerate(nc,1))/(nn*cs),'top10':100*sum(nc[int(.9*nn):])/cs,'top1':100*sum(nc[int(.99*nn):])/cs}
            countries=sorted(set(str(r['cntry']) for r in d if r['cntry']))
            out['countries']={k:rates([r for r in d if r['cntry']==k]) for k in countries}
            out['non_nested']={k:sum(n(r,a)<n(r,b) for r in d) for k,a,b in [('first_p','cpsf','nps'),
('other_p','np','cpsf'),('first_c','ncsf','ncs'),('other_c','nc','ncsf')]}
        return out

def normalization(rows):
    with (SRC/'data/world_subfield_year.csv').open(encoding='utf-8-sig',newline='') as f:
w=list(csv.DictReader(f))
    countkey=next(k for k in w[0] if k not in ('cntry','sml','year'))
    result={}
    for label,years in [('2024',[2024]),('pooled',YEARS)]:
        a={};f={}
        for r in w:
            if int(r['year']) not in years: continue
            cc=r['cntry'].strip().lower(); value=int(float(r[countkey]));a[cc]=a.get(cc,0)+value
            if r['sml']=='Forestry': f[cc]=f.get(cc,0)+value
        total=sum(a.values());ft=sum(f.values())
        extract_f=sum(forestry(r) and r['year'] in years for r in rows)
        assert extract_f==ft,(label,extract_f,ft)
        cc_extract={}
        for r in rows:
            if forestry(r) and r['year'] in years:
                cc=str(r['cntry']).strip().lower();cc_extract[cc]=cc_extract.get(cc,0)+1
            assert all(cc_extract.get(cc,0)==v for cc,v in f.items())
            records=[{'country':cc,'forestry':f.get(cc,0),'all':av,'forestry_share':f.get(cc,0)/ft,'all_share':av/
total,'ri':f.get(cc,0)*total/(ft*av)} for cc,av in a.items()]
            records.sort(key=lambda r:(-r['forestry'],r['country']))
            result[label]={'forestry_total':ft,'all_total':total,'unknown_all':a.get('',0),'unknown_forestry':f.get('
',0),'records':records}
        return result

if __name__=='__main__':
    rows=read_extract(); print(len(rows),list(rows[0])); print(rows[0])

```

11. figtrim.py (SHA-256 e33963dd7f771b75...; 22 lines)

"""Trim the white canvas margin of a figure PNG so that the caption skip is measured from the graph itself.

matplotlib leaves 3 to 12 pt of white canvas around the drawn area; with the figure caption set 9 pt below the image, that margin would add to the visible gap. trim() crops each PNG to its ink plus PAD_PT, keeping the dpi metadata, so the drawn type keeps its size when the file is included at its native width (see fig_width()).

"""

```

from pathlib import Path
from PIL import Image
import numpy as np
PAD_PT=1.0
def trim(path,thresh=250):
    path=Path(path);im=Image.open(path);dpi=im.info.get('dpi',(300,300))
    a=np.array(im.convert('L'));ink=a<thresh
    rows=np.where(ink.any(axis=1))[0];cols=np.where(ink.any(axis=0))[0]
    pad=int(round(PAD_PT/72*dpi[0]))
    box=(max(int(cols[0])-pad,0),max(int(rows[0])-pad,0),min(int(cols[-1])+pad+1,a.shape[1]),min(int(rows[-1])+pad+1,a.shape[0]))
    im.crop(box).save(path,dpi=dpi)
    return box
def fig_width(path,nominal_in):
    """Width (inches) at which the trimmed PNG prints at its native size, capped at the nominal width."""
    im=Image.open(path);dpi=im.info.get('dpi',(300,300))[0]
    return min(im.width/dpi,nominal_in)

```

12. make_figures.py (SHA-256 0630eff5f1d69404...; 122 lines)

"""Create the paper figure and supplementary diagnostic plots at 300 dpi with 7-9 pt type at the printed size.

Figure A1 (publishing_base_impact.png, text width): A, representation index of the 25 top countries by primary-Forestry count (2024 data year, pooled value as a hollow marker); B, citations per single-authored paper against citations per paper in all positions for the same 25 countries, every point labelled "Country single/all" with labels placed automatically so that none overlaps a point or another label.
 authorship_rank_sensitivity.png: rank sensitivity to the authorship basis; not used in the paper, retained for the archived verification-case draft.
 integrity_diagnostics.png: Lorenz curve and hm against h; not used in the paper.
 Revision: the former normalization and country-impact plots are the two panels of Figure A1, panel B extended from the 16 countries with at least 40 records to the 25 countries of panel A. All plotted values come from statistics.json / normalization.json / rank_vectors.json or the read-only cohort extract.

```

"""
import sys,json,os
from pathlib import Path
from data_core import ROOT,read_extract,cohort,n
os.environ['MPLCONFIGDIR']=str(ROOT/'build/tmp/matplotlib')
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from manuscript_content import COUNTRIES
from figtrim import trim # crop the white canvas margin so the caption skip is measured from the graph
plt.rcParams.update({'font.family':'DejaVu Sans','font.size':9,'axes.titlesize':9.5,'axes.labelsize':9,'xtick.labelsize':8,'ytick.labelsize':8,'legend.fontsize':8})
TEXTWIDTH=7.0 # MCFNS text width in inches; Figure 1 is drawn at this width so that its type is at true size
GREEN='#2a5e45';BLUE='#2a78d6';ORANGE='#eb6834';GRAY='0.5';LIGHT='#b9d3c3'

def place_labels(ax,fig,pts,labels,fontsize=6.3):
    """Label every point, choosing for each the first of the candidate offsets (near to far, eight directions) whose text box lies inside the allowed region (the axes, the gap to the neighbouring panel on the left, and the margin to the figure edge on the right) and overlaps neither a point nor a label placed earlier; the most crowded points are labelled first; a thin gray connector joins a label that had to move away from its point."""
    fig.canvas.draw();rend=fig.canvas.get_renderer();axbox=ax.get_window_extent(rend);fb=fig.bbox
    region=(axbox.x0+2,axbox.y0-2,fb.x1-3,axbox.y1+2) # inside the axes on the left (tick labels live there), out to the figure edge on the right
    pt=[ax.transData.transform((x,y)) for x,y in pts];pt_boxes=[(px-3.5,py-3.5,px+3.5,py+3.5) for px,py in pt]
    dirs=[(1,0,'left','center'),(-1,0,'right','center'),(0,1,'center','bottom'),(0,-1,'center','top'),(0.8,0.8,'left','bottom'),(-0.8,0.8,'right','bottom'),(0.8,-0.8,'left','top'),(-0.8,-0.8,'right','top')]
    cands=[(d*sx,d*sy,ha,va) for d in (5,10,16,24,34,46,60,76,94) for sx,sy,ha,va in dirs]
    def overlap(b,o):return max(0,min(b[2],o[2])-max(b[0],o[0]))*max(0,min(b[3],o[3])-max(b[1],o[1]))
    def hits(b,others):return any(overlap(b,o)>0 for o in others)
    crowd=[sum(1 for q in pt if abs(q[0]-p[0])<40 and abs(q[1]-p[1])<40) for p in pt]
    order=sorted(range(len(pts)),key=lambda i:-crowd[i])
    placed=[]
    for i in order:
        (x,y),lab=pts[i],labels[i];others=[b for j,b in enumerate(pt_boxes) if j!=i];chosen=None;best=None
        for dx,dy,ha,va in cands:
            t=ax.annotate(lab,xy=(x,y),xytext=(dx,dy),textcoords='offsetpoints',ha=ha,va=va,fontsize=fontsize,linespacing=0.95)
            bb=t.get_window_extent(rend);b=(bb.x0-1,bb.y0-1,bb.x1+1,bb.y1+1);t.remove()
            inside=b[0]>=region[0] and b[2]<=region[2] and b[1]>=region[1] and b[3]<=region[3]
            if inside and not hits(b,placed) and not hits(b,others):chosen=(dx,dy,ha,va,b);break
            if inside: # remember the least-overlapping candidate in case no free spot exists
                cost=sum(overlap(b,o) for o in placed)+sum(overlap(b,o) for o in others)+0.02*(abs(dx)+abs(dy))
                if best is None or cost<best[0]:best=(cost,dx,dy,ha,va,b)
            if chosen is None:chosen=best[1:] if best else (5,0,'left','center',None)

```

```

        dx,dy,ha,va,b=chosen;far=abs(dx)>7 or abs(dy)>7
        t=ax.annotate(lab,xy=(x,y),xytext=(dx,dy),textcoords='offset
points',ha=ha,va=va,fontsize=fontsize,linespacing=0.95,arrowprops=dict(arrowstyle='-',lw=0.4,color=GRAY,shrinkA=0
,shrinkB=2.5) if far else None)
        if b is None:bb=t.get_window_extent(rend);b=(bb.x0,bb.y0,bb.x1,bb.y1)
        placed.append(b)

def main():
    a=json.loads((ROOT/'build/statistics.json').read_text());b=json.loads((ROOT/'build/
normalization.json').read_text())
    rows=[r for r in b['2024']['records'] if r['country'][:25];pooled={r['country']:r['ri'] for r in b['pooled']
['records']}
    # Figure 1: A, representation index (log axis, reference at 1, pooled value hollow); B, citations per paper,
single against all, same 25 countries
    fig=plt.figure(figsize=(TEXTWIDTH,4.1))
    gs=fig.add_gridspec(1,2,width_ratios=[1.0,1.18],left=0.155,right=0.985,top=0.915,bottom=0.175,wspace=0.30)
    # bottom margin holds the x labels and, below them, the legend of panel A
    axA=fig.add_subplot(gs[0]);axB=fig.add_subplot(gs[1])
    yy=list(range(len(rows)));vals=[r['ri'] for r in rows]
    axA.hlines(yy,1,vals,color=GREEN,linewidth=1.5)
    axA.scatter(vals,yy,s=16,color=GREEN,zorder=3,label='2024 data year')
    axA.scatter([pooled[r['country']] for r in
rows],yy,s=22,facecolors='none',edgecolors=ORANGE,linewidths=1.0,zorder=4,label='pooled 2020-2024')
    axA.set_yticks(yy,[f'{COUNTRIES[r['country']] }({r['forestry'])}' for r in
rows],fontsize=7.2);axA.invert_yaxis()
    for y,v,r in zip(yy,vals,rows):
        pv=pooled[r['country']];right=v>=1
        axA.annotate(f'{v:.2f}',(max(v,pv) if right else 1.0,y),xytext=(5,0),textcoords='offset
points',va='center',ha='left',fontsize=7.5)
        axA.set_xscale('log');axA.set_xlim(0.12,45);axA.set_xticks([0.2,0.5,1,2,5,10,20],
['0.2','0.5','1','2','5','10','20'])
        axA.axvline(1,color=GRAY,linewidth=1);axA.set_xlabel('RI (log scale);reference = 1')
        axA.set_title('RI, 2024 data year',loc='left',pad=8)
        axA.spines[['top','right','left']].set_visible(False);axA.tick_params(axis='y',length=0)
        h,l=axA.get_legend_handles_labels();fig.legend(h,l,loc='lower
left',bbox_to_anchor=(0.10,0.0),ncol=2,frameon=False,columnspacing=1.0,handletextpad=0.3)
        sel=[(r['country'],a['countries'][r['country']]) for r in rows]
        xs=[v['all_cpp'] for k,v in sel];ys=[v['solo_cpp'] for k,v in sel]
        lo=min(xs+ys)*0.76;hi=max(xs+ys)*1.08 # room below and left of the crowded low-rate countries for their
labels
        axB.plot([lo,hi],[lo,hi],color=GRAY,linewidth=1)
        axB.scatter(xs,ys,s=22,color=LIGHT,edgecolors=GREEN,linewidths=1.0,zorder=3)
        axB.set_xlim(lo,hi);axB.set_ylim(lo,hi)
        axB.set_xlabel('All papers');axB.set_ylabel('Single author papers');axB.set_title('Citations per
paper',loc='left',pad=8)
        axB.spines[['top','right']].set_visible(False)
        place_labels(axB,fig,list(zip(xs,ys)),[f'{COUNTRIES.get(k,k)}\n{y:.1f}/{x:.1f}' for (k,v),x,y in
zip(sel,xs,ys)]) # two-line labels: country, then single/all
        posB=axB.get_position();fig.text(0.012,0.965,'A',fontsize=11,fontweight='bold',va='top');fig.text(posB.x0-
0.075,0.965,'B',fontsize=11,fontweight='bold',va='top')
        fig.savefig(ROOT/'figures/publishing_base_impact.png',dpi=300);plt.close(fig);trim(ROOT/'figures/
publishing_base_impact.png')
        for old in ('normalization_2024.png','country_impact_slice.png):(ROOT/'figures'/old).unlink(missing_ok=True)
    # superseded by the two-panel figure

    # Diagnostic plot, not used in the paper: rank sensitivity (left) and rank changes, single minus total
(right)
    rv=json.loads((ROOT/'build/rank_vectors.json').read_text());diff=[s-t for s,t in
zip(rv['ranks_single'],rv['ranks_total'])]
    fig,(ax1,ax2)=plt.subplots(1,2,figsize=(5.6,2.6),gridspec_kw={'width_ratios':[1,1.25]})
    labels=[('Key','Key (legacy)'),('Lead','Single-or-first'),('Single','Single')]
    vals=[a['ranks'][k][r['rho']] for k,_ in labels];ov=[a['ranks'][k]['overlap'] for k,_ in labels]
    ax1.barh(range(3),vals,color=[BLUE if v>0 else ORANGE for v in vals]);ax1.set_yticks(range(3),[l for _,l in
labels]);ax1.invert_yaxis()
    for i,(v,o) in enumerate(zip(vals,ov)):ax1.annotate(f'{v:+.2f}; {o}/25',
(max(v,0),i),xytext=(4,0),textcoords='offset points',va='center',fontsize=8)
    ax1.set_xlim(-0.15,1.25);ax1.set_xticks([0,.25,.5,.75,1]);ax1.axvline(0,color=GRAY,linewidth=.8);ax1.set_xlab
el('rho with total-citation rank')
    ax1.set_title('Spearman rho; top-25 overlap',loc='left');ax1.spines[['top','right']].set_visible(False)
    ax2.hist(diff,bins=range(-850,900,50),color=LIGHT,edgecolor=GREEN,linewidth=.8)
    ax2.axvline(0,color=GRAY,linewidth=.8);ax2.set_xlabel('Rank change\ (single-author minus
total)');ax2.set_ylabel('Profiles')
    ax2.set_title('Rank change, 854 profiles',loc='left');ax2.spines[['top','right']].set_visible(False)
    fig.subplots_adjust(left=.19,right=.97,top=.88,bottom=.28,wspace=.40)
    fig.savefig(ROOT/'figures/authorship_rank_sensitivity.png',dpi=300);plt.close(fig);trim(ROOT/'figures/
authorship_rank_sensitivity.png')

    # Diagnostic plot, not used in the paper: Lorenz curve and h versus hm
    r=[x for x in cohort(read_extract()) if x['year']==2024];nc=sorted(n(x,'nc') for x in r)
    fig,(ax1,ax2)=plt.subplots(1,2,figsize=(5.6,2.8))

```

```

tot=sum(nc);c=0;pts=[(0,0)]
for i,v in enumerate(nc,1):c+=v;pts.append((i/len(nc),c/tot))
ax1.plot([p[0] for p in pts],[p[1] for p in pts],color=GREEN,linewidth=1.6);ax1.plot([0,1],
[0,1],color=GRAY,linewidth=1)
ax1.set_xlabel('Cumulative author share');ax1.set_ylabel('Cumulative citation share');ax1.set_title(f'Lorenz
curve; Gini = {a['integrity']['gini']:.2f}",loc='left');ax1.set_aspect('equal')
ax2.scatter([n(x,'h') for x in r],[n(x,'hm') for x in r],s=6,color=LIGHT,edgecolors='none')
ax2.plot([0,130],[0,130],color=GRAY,linewidth=1);ax2.plot([0,130],[0,65],color=BLUE,linewidth=1)
ax2.set_xlim(0,130);ax2.set_ylim(0,130);ax2.set_xlabel('h');ax2.set_ylabel('hm');ax2.set_title('hm against
h',loc='left')
for ax in (ax1,ax2):ax.spines[['top','right']].set_visible(False)
fig.subplots_adjust(left=.10,right=.98,top=.89,bottom=.17,wspace=.32)
fig.savefig(ROOT/'figures/integrity_diagnostics.png',dpi=300);plt.close(fig);trim(ROOT/'figures/
integrity_diagnostics.png')
(ROOT/'build/figure_build.json').write_text(json.dumps({'backend':'matplotlib','normalization_backend':'matpl
otlib','print_width_inches':7.0,'minimum_font_points':6.3,'dpi':300,'country_rule':'panel B shows the 25 top
countries by primary-Forestry count of panel A (2024 data year); rates from the pooled author-release records
2020 to 2024','countries_shown':[k for k,_ in sel],'source':'statistics.json, normalization.json,
rank_vectors.json, read-only cohort extract','replaced':['normalization_2024.png and country_impact_slice.png
(merged into publishing_base_impact.png)','ff4_group.png (stale count-share image)','ff_ratio_scatter.png
(overlapping labels)','ff_authrank.png (font scale and legacy benchmark)','ff_integrity.png (font scale and
interpretive wording)'],'dropped_in_revision':['group_nine_rates.png (duplicated the nine-rate
table)']},indent=2))
print('figures written; countries in Figure 1B:',[k for k,_ in sel])

if __name__=='__main__':main()

```

13. make_package.py (SHA-256 e91bc8bc0345fce1...; 310 lines)

"""Refresh Appendix C from current publication artifacts and unchanged original inputs.

Rebuild scripts/, outputs/, and manuscript/ (the response letters) from explicit inputs. The source text of the paper and copies of its published files are not packaged. Keep the original record folders and cohort extract untouched. Clean only published copies, check publication dates, write SHA256SUMS.txt, and archive with fixed ZIP timestamps. README_AppendixC.md is maintained directly in the package folder.

Usage: python build/make_package.py

```

"""
import copy
import hashlib
import json
import os
import re
import shutil
import tempfile
import zipfile
import xml.etree.ElementTree as ET
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
PKG = ROOT / 'AppendixC_package'
B = ROOT / 'build'
ZIP_EPOCH = (2026, 4, 30, 12, 0, 0) # the issue date, Apr. 30, 2026, 12:00, as in the other archives published
in l8(1)
EXCLUDED_SCRIPTS = {'check_opinion_restructure.py', '_patch_tex.py', 'check_rewrite.py',
'build_editorial_docx.py', 'archive_review.py'} # internal audit, retired, and non-paper
programs
PUBLICATION_SCRIPTS = sorted(p for p in B.glob('*.py') if p.name not in EXCLUDED_SCRIPTS)
SCRIPTS = PUBLICATION_SCRIPTS + [B / n for n in ('roster_overrides.json', 'balance_hint.json',
'college_checked.csv')]
OUTPUTS = [
'statistics.json', 'normalization.json', 'revision_checks.json', 'rank_vectors.json',
'nash_search.json', 'ugapay.json', 'pay_figures.json', 'pay_sensitivity.json',
'std_summary.json', 'content_manifest.json', 'figure_build.json',
'numeric_traceability.json', 'source_manifest.json', 'annual_ri.csv',
'prewindow_retirees.csv', 'verify_output.txt',
]
MANUSCRIPT = [ # the paper's published files: checked by the publication guard, not packaged
'Cieszewski2026_AI.tex', 'Cieszewski2026_AI_MCFNS18-1.pdf',
'Cieszewski2026_AI_draft.docx', 'Cieszewski2026_AI_draft.pdf',
'Cieszewski2026_AI_AppendixB_pay.docx', 'Cieszewski2026_AI_AppendixB_pay.pdf',
'Cieszewski2026_AI_AppendixD_programs.docx', 'Cieszewski2026_AI_AppendixD_programs.pdf',
'Cieszewski2026_AI_AppendixE_original_pipeline.docx', 'Cieszewski2026_AI_AppendixE_original_pipeline.pdf',
'mcfnsApr2024v2.sty',
]
PAY_WORKBOOK = 'UGA StanfordList_Salaries_FY2021-2025.xlsx'
EXTRACT = 'StanfordList_Forestry_Ecology_career_2020-2024.xlsx'
RECORD_DIRS = ('inputs/record', 'inputs/data', 'inputs/july_figures', 'july_pipeline')

```



```

        new = re.sub(r'[ \t]+([.,;])', r'\1', new)
        new = re.sub(r'[ \t]{2,}', ' ', new).strip()
    if new != old:
        print(f'    workbook {sheet.title}!{cell.coordinate}: {old!r} -> {new!r}')
        cell.value = new
    assert hits == 1, ('Notes retrieval provenance', hits)
    workbook.save(path)
finally:
    workbook.close()

```

```

def normalize_office_zip(path):
    """Remove core created/modified properties; sort members with fixed ZIP times.

    Each member keeps its original compression method. Other XML parts are copied
    verbatim. Only packaged project documents and the packaged pay workbook use this.
    """
    fd, temporary = tempfile.mkstemp(prefix=path.stem + '-', suffix=path.suffix, dir=path.parent)
    os.close(fd)
    temporary = Path(temporary)
    try:
        with zipfile.ZipFile(path) as source, zipfile.ZipFile(temporary, 'w') as target:
            for member in sorted(source.infolist(), key=lambda item: item.filename):
                data = source.read(member)
                if member.filename == 'docProps/core.xml':
                    core = ET.fromstring(data)
                    for child in list(core):
                        if child.tag in ('{' + DCTERMS + '}created', '{' + DCTERMS + '}modified'):
                            core.remove(child)
                    data = ET.tostring(core, encoding='utf-8', xml_declaration=True)
                info = copy.copy(member)
                info.date_time = ZIP_EPOCH
                info.extra = b'' # Also discard optional ZIP timestamp extra fields.
                target.writestr(info, data)
        os.replace(temporary, path)
    finally:
        temporary.unlink(missing_ok=True)

def publication_guard():
    """Fail with all publication-date hits; count retained original-record dates."""
    from openpyxl import load_workbook
    from pdfclean import fitz
    problems = []
    july_count = 0

    def check_text(label, text):
        for pattern in (PROJECT_DATES, PROJECT_MONTH_WORDING):
            for match in pattern.finditer(text):
                problems.append(f'{label}: {match.group()!r}')

    # Check folder and file names as well as content, outside the frozen record.
    paths = sorted(PKG.rglob('*'))
    for path in paths:
        if not frozen(path):
            check_text(str(path.relative_to(ROOT)), path.relative_to(PKG).as_posix())
    published = [ROOT / name for name in MANUSCRIPT if name.endswith('.pdf')]
    for path in [p for p in paths if p.is_file()] + published:
        label = str(path.relative_to(ROOT))
        record = path.is_relative_to(PKG) and frozen(path)
        if record:
            july_count += len(JULY_DATES.findall(path.name))
            if path.suffix.lower() in TEXT_SUFFIXES:
                july_count += len(JULY_DATES.findall(path.read_text(encoding='utf-8-sig')))
            elif path.suffix.lower() == '.pdf':
                with fitz.open(path) as doc:
                    july_count += len(JULY_DATES.findall(' '.join(page.get_text() for page in doc)))
                    july_count += len(JULY_DATES.findall(str(doc.metadata) + doc.get_xml_metadata()))
            continue
        check_text(label + ' name', path.name)
        if path.suffix.lower() in TEXT_SUFFIXES:
            check_text(label, path.read_text(encoding='utf-8-sig'))
        elif path.suffix.lower() == '.pdf':
            with fitz.open(path) as doc:
                for key in ('creationDate', 'modDate'):
                    if doc.metadata.get(key):
                        problems.append(f'{label}: PDF {key} is present')
                kind, value = doc.xref_get_key(-1, 'Info')

```

```

        if kind == 'xref':
            keys = doc.xref_get_keys(int(value.split()[0]))
            for key in ('CreationDate', 'ModDate'):
                if key in keys:
                    problems.append(f'{label}: PDF /{key} key remains')
        if doc.get_xml_metadata() or doc.xref_xml_metadata():
            problems.append(f'{label}: PDF XMP remains')
        check_text(label + ' metadata', str(doc.metadata))
        for number, page in enumerate(doc, 1):
            check_text(f'{label} page {number}', page.get_text())
    elif path.suffix.lower() in ('.docx', '.xlsx'):
        if path == PKG / 'inputs' / EXTRACT:
            continue # The unchanged extract is an original input, not a project output.
        with zipfile.ZipFile(path) as archive:
            members = archive.infolist()
            if [m.filename for m in members] != sorted(m.filename for m in members):
                problems.append(f'{label}: Office ZIP members are not sorted')
            for member in members:
                if member.date_time != ZIP_EPOCH:
                    problems.append(f'{label}: Office ZIP timestamp on {member.filename}')
            if 'docProps/core.xml' in archive.namelist():
                xml = archive.read('docProps/core.xml')
                core = ET.fromstring(xml)
                for tag in ('created', 'modified'):
                    if core.find('{ ' + DCTERMS + '}' + tag) is not None:
                        problems.append(f'{label}: dcterms:{tag} remains')
                check_text(label + ' core.xml', xml.decode('utf-8'))
            if path.suffix.lower() == '.docx':
                for member in members:
                    if member.filename.startswith('word/') and member.filename.endswith('.xml'):
                        tree = ET.fromstring(archive.read(member))
                        check_text(label + ' ' + member.filename, ''.join(tree.itertext()))
        if path.name == PAY_WORKBOOK:
            workbook = load_workbook(path, read_only=True, data_only=False)
            try:
                for sheet in workbook:
                    for row in sheet:
                        for cell in row:
                            if isinstance(cell.value, str):
                                check_text(f'{label} {sheet.title}!{cell.coordinate}', cell.value)
            finally:
                workbook.close()
        print(f'INFO retained original-record July dates: {july_count} occurrences (content, PDF metadata, and
names).')
        if problems:
            raise AssertionError('Publication date/metadata guard failed:\n' + '\n'.join(problems))
        print('PASS publication date/metadata guard; frozen original records excluded.')

def main():
    if not CLEAN_FILE.exists(): raise SystemExit('make_package.py builds the deposit and needs the private
settings file build/tmp/publication_cleaning.json')
    from pdfclean import strip_pdf_dates
    required = SCRIPTS + [B / n for n in OUTPUTS]
    required += [ROOT / n for n in MANUSCRIPT] + [ROOT / PAY_WORKBOOK, PKG / 'README_AppendixC.md']
    required += [ROOT / source for source, _, _ in RESPONSE_COPIES]
    missing = [str(p.relative_to(ROOT)) for p in required if not p.is_file()]
    assert not missing, ('Build required publication inputs before packaging', missing)

    # Rebuild only the mutable package folders. Resolve every recursive deletion
    # and reject symlinks/junctions that could reach original inputs or the workspace.
    assert PKG.resolve() == ROOT / 'AppendixC_package', 'Package root is redirected'
    for name in ('scripts', 'outputs', 'manuscript'):
        target = PKG / name
        assert target.resolve() == PKG / name, ('Redirected package folder', target)
        if target.exists():
            for child in target.rglob('*'):
                assert child.resolve().is_relative_to(target), ('Redirected package member', child)
            shutil.rmtree(target)
        target.mkdir()
    for path in SCRIPTS:
        put(path, PKG / 'scripts' / path.name)
    for name in OUTPUTS:
        put(B / name, PKG / 'outputs' / name)
    stale = {x.split()[0] for x in json.loads(B /
'figure_build.json').read_text(encoding='utf-8')).get('dropped_in_revision', [])} # entries read 'name.png
(reason)'
    for path in sorted(p for p in (ROOT / 'figures').glob('*.png') if p.name not in stale): # no figure dropped
in revision
        put(path, PKG / 'outputs' / 'figures' / path.name)

```

```

for source, destination, substitutions in RESPONSE_COPIES:
    text = substitute_once((ROOT / source).read_bytes().decode('utf-8'), substitutions, source)
    (PKG / 'manuscript' / destination).write_bytes(text.encode('utf-8')) # bytes: line endings stay as in
the original
put(ROOT / PAY_WORKBOOK, PKG / 'inputs' / PAY_WORKBOOK)
clean_pay_workbook(PKG / 'inputs' / PAY_WORKBOOK)
for path in sorted(PKG.rglob('*')):
    if not path.is_file() or frozen(path) or path == PKG / 'inputs' / EXTRACT:
        continue
    if path.suffix.lower() in ('.docx', '.xlsx'):
        normalize_office_zip(path)
    elif path.suffix.lower() == '.pdf':
        strip_pdf_dates(path)

files = sorted((p for p in PKG.rglob('*') if p.is_file() and p != PKG / 'SHA256SUMS.txt'),
               key=lambda p: p.relative_to(PKG).as_posix())
lines = [hashlib.sha256(p.read_bytes()).hexdigest() + ' ' + p.relative_to(PKG).as_posix() for p in files]
(PKG / 'SHA256SUMS.txt').write_text('\n'.join(lines) + '\n', encoding='utf-8')
publication_guard() # Refuse to publish an archive until every content/metadata check passes.
zpath = ROOT / 'Cieszewski2026_AI_AppendixC_package.zip'
temporary = zpath.with_suffix('.zip.tmp')
try:
    with zipfile.ZipFile(temporary, 'w', compression=zipfile.ZIP_DEFLATED) as archive:
        for path in sorted(files + [PKG / 'SHA256SUMS.txt'], key=lambda p: p.relative_to(PKG).as_posix()):
            info = zipfile.ZipInfo('AppendixC_package/' + path.relative_to(PKG).as_posix(), ZIP_EPOCH)
            info.compress_type = zipfile.ZIP_DEFLATED
            info.create_system = 3
            info.external_attr = 0o100644 << 16
            archive.writestr(info, path.read_bytes())
    with zipfile.ZipFile(temporary) as archive:
        assert archive.namelist() == sorted(archive.namelist())
        assert all(item.date_time == ZIP_EPOCH for item in archive.infolist())
    os.replace(temporary, zpath)
finally:
    temporary.unlink(missing_ok=True)
print('package:', len(files) + 1, 'files;', zpath.name, round(zpath.stat().st_size / 1e6, 1), 'MB')

if __name__ == '__main__':
    main()

```

14. make_pay_figures.py (SHA-256 ef2499e63ef5a82e...; 203 lines)

"""Figures for Appendix B (standardized-pay exercise). Reads the companion workbook sheets written by build/redo_standardized.py and writes two PNGs into figures/ plus build/pay_figures.json by default (PAY_XL, PAY_FIG_OUT and PAY_OUT override the workbook, figure directory and JSON directory).

Every pay figure has the same three columns, the authorship classes of the version 8 career table, left to right: all positions (np, nc); last author (np_last = npsfl - cpsf and nc_last = ncsfl - ncsf: papers on which the person is the last but not the first or sole author, and the citations to them); single author (nps, ncs). build/check_authorship_fields.py confirms the workbook's np_last and nc_last against the raw release for all 321 profiles.

Figure B2 (pay_authorship_output.png; its title "Author salaries" is in the caption only): three rows, a) pay against citations per author, b) pay against citations per paper, c) pay against papers, the same three classes in every row.

Figure B1 (deans_vs_scientists_std.png): deans against the listed scientists of their colleges. The left panel compares mean dean pay in clean college-years with mean observed screened scientist pay in exactly those fiscal years, weighted by person-years, both standardized over those years, so the scientists of every college sit at U and the dean at U times the ratio; the right panel retains the annual ratios. Observed pay is primary: a person with any observed usable year is plotted from the mean of those observed years only (blue dot); a person with modeled years only (pre-window retiree) is plotted from the mean of the modeled years (orange hollow triangle). Each panel states n and the Spearman correlation for the observed persons; the correlations for all persons are written to pay_figures.json. Requires pandas, openpyxl, numpy, matplotlib, scipy.

```

"""
import json, os
import numpy as np, pandas as pd
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter, FixedLocator, NullLocator, NullFormatter
from scipy.stats import spearmanr
from pathlib import Path
ROOT = Path(__file__).resolve().parents[1]
import sys; sys.dont_write_bytecode = True; sys.path.insert(0, str(ROOT / 'build')); from figtrim import trim #
crop the white canvas margin

```

```

from paynames import join_idx # accent-insensitive name join
XL = Path(os.environ.get('PAY_XL', ROOT / 'UGA StanfordList Salaries FY2021-2025.xlsx'))
OUT = Path(os.environ.get('PAY_FIG_OUT', ROOT / 'figures')); OUT.mkdir(parents=True, exist_ok=True)
JSON_OUT = Path(os.environ.get('PAY_OUT', ROOT / 'build')); JSON_OUT.mkdir(parents=True, exist_ok=True)
BLUE, ORANGE = '#2a78d6', '#eb6834'
INK, INK2, GRID = '#0b0b0b', '#52514e', '#d9d8d4'
plt.rcParams.update({'font.size': 9, 'axes.titlesize': 9.5, 'axes.labelsize': 9, 'xtick.labelsize': 8,
'ytick.labelsize': 8, 'legend.fontsize': 8,
'axes.edgecolor': GRID, 'axes.linewidth': 0.8, 'axes.spines.top': False,
'axes.spines.right': False, 'text.color': INK, 'axes.labelcolor': INK2, 'xtick.color': INK2, 'ytick.color':
INK2})
W = 6.0 # print width in inches; fonts are true size at this width (body text is 10 pt)

panel = pd.read_excel(XL, sheet_name='Std panel')
idx = pd.read_excel(XL, sheet_name='Authorship indices (321)')
ratios = pd.read_excel(XL, sheet_name='Std dean vs sci')
use = panel[panel.use].copy()
obs = use[~use.imputed].groupby('auth').agg(pay_std=('pay_std', 'mean'), n_years=('fy', 'size')).reset_index();
obs['modeled'] = False
mod_only = use[use.imputed & ~use.auth.isin(obs.auth)].groupby('auth').agg(pay_std=('pay_std', 'mean'),
n_years=('fy', 'size')).reset_index(); mod_only['modeled'] = True
per = join_idx(pd.concat([obs, mod_only], ignore_index=True), idx)
# the two position counts get +1 for the logarithmic axes (ranks, and so the correlations, are unchanged)
per['np_last1'] = per.np_last.clip(lower=0) + 1; per['nps1'] = per.nps + 1
per['nc_last1'] = per.nc_last.clip(lower=0) + 1; per['ncs1'] = per.ncs + 1
per['last_pap_share'] = per.np_last / per.np; per['last_cit_share'] = per.nc_last / per.nc
per['single_pap_share'] = per.nps / per.np; per['single_cit_share'] = per.ncs / per.nc
# same-person annual standardized raises, observed non-administrative years only, FY2022-2025
act = panel[panel.base].sort_values(['auth', 'fy']).copy()
act['prev'] = act.groupby('auth').pay_std.shift(); act['inc'] = 100 * (act.pay_std / act.prev - 1)
inc = act.dropna(subset=['inc']).groupby('auth').agg(inc_mean=('inc', 'mean'), n_inc=('inc',
'size')).reset_index()
per = per.merge(inc, on='auth', how='left')
kfmt = FuncFormatter(lambda v, p: f'{v/1000:.0f}k')
YL = (40000, 480000); YT = [50000, 100000, 150000, 200000, 300000, 400000]

def rho(x, y):
    m = np.isfinite(x) & np.isfinite(y)
    if m.sum() < 8: return None, None, int(m.sum())
    r, p = spearmanr(x[m], y[m]); return float(r), float(p), int(m.sum())
def rtxt(r, p, n, tag=''):
    if r is None: return f'{tag}n = {n}'
    return f'{tag}rho {r:+.2f}, ' + ('p<0.001' if p < 0.001 else (f'p={p:.3f}' if p < 0.01 else f'p={p:.2f}')) +
f' (n {n})'

def scatter_panel(ax, x, y, df, xlabel, logx=True, logy=True, fit=True, key=None, stats=None, prefix=''):
    a = df[~df.modeled]; r = df[df.modeled]
    ax.scatter(a[x], a[y], s=20, c=BLUE, alpha=0.75, linewidths=0, label='observed pay, mean of observed years',
zorder=3)
    ax.scatter(r[x], r[y], s=28, facecolors='none', edgecolors=ORANGE, linewidths=1.2, marker='^', label='modeled
pay only, pre-window retirees', zorder=3)
    if logx:
        ax.set_xscale('log'); ax.xaxis.set_minor_locator(NullLocator());
    ax.xaxis.set_minor_formatter(NullFormatter())
    lo, hi = np.nanmin(df[x][df[x] > 0]), np.nanmax(df[x]); cand = [1, 3, 10, 30, 100, 300, 1000, 3000,
10000, 30000]
    ticks = [t for t in cand if lo * 0.8 <= t <= hi * 1.25]
    if np.log10(hi / lo) > 1.6 and len(ticks) > 4: ticks = [t for t in ticks if t in (1, 10, 100, 1000,
10000)]
    if len(ticks) < 3: ticks = [t for t in cand if lo * 0.5 <= t <= hi * 2] # a narrow range (career
citations) still gets three labelled ticks
    ax.xaxis.set_major_locator(FixedLocator(ticks)); ax.xaxis.set_major_formatter(FuncFormatter(lambda v, p:
f'{v/1000:g}k' if v >= 1000 else f'{v:g}'))
    if logy:
        ax.set_yscale('log'); ax.yaxis.set_minor_locator(NullLocator());
    ax.yaxis.set_minor_formatter(NullFormatter())
    ax.yaxis.set_major_locator(FixedLocator(YT)); ax.yaxis.set_major_formatter(kfmt); ax.set_ylim(*YL)
    xa = (np.log(a[x]) if logx else a[x]).values; ya = (np.log(a[y]) if logy else a[y]).values
    xd = (np.log(df[x]) if logx else df[x]).values; yd = (np.log(df[y]) if logy else df[y]).values
    if fit:
        m = np.isfinite(xa) & np.isfinite(ya)
        if m.sum() > 8:
            b1, b0 = np.polyfit(xa[m], ya[m], 1); xs = np.linspace(xa[m].min(), xa[m].max(), 50)
            ax.plot(np.exp(xs) if logx else xs, np.exp(b0 + b1 * xs) if logy else b0 + b1 * xs, color=INK2,
lw=1.6, zorder=4, label='least-squares fit, observed persons')
        ra, pa, na = rho(xa, ya); rd, pd, nd = rho(xd, yd)
        ax.set_title(prefix + rtxt(ra, pa, na), loc='left', fontsize=6.8, color=INK2, pad=3) # observed persons; the
statistic sits above the panel so that no data point is covered
        if key is not None and stats is not None: stats[key] = {'n': na, 'rho': ra, 'p': pa}, 'all':
{'n': nd, 'rho': rd, 'p': pd}}

```

```

ax.set_xlabel(xlabel); ax.grid(True, color=GRID, lw=0.6, alpha=0.8); ax.set_axisbelow(True)

stats = {'n_persons': int(len(per)), 'n_observed': int((~per.modeled).sum()), 'n_modeled_only':
int(per.modeled.sum()), 'rho': {}, 'rho_raise': {}, 'rho_shares': {}}
# columns: (plotted column, statistics key, axis label)
# one-line x labels: 'All Papers (np)', the other two without 'papers'; citations -> 'cite all', 'cite last',
'cite single'
PAPERS = [('np', 'np', 'All Papers (np)'), ('np_last1', 'np_last', 'last-author (np_last + 1)'), ('nps1', 'nps',
'single-author (nps + 1)')]
CITES = [('nc', 'nc', 'cite all (nc)'), ('nc_last1', 'nc_last', 'cite last (nc_last + 1)'), ('ncs1', 'ncs', 'cite
single (ncs + 1)')]
CPP = [('cpp_all', 'cpp_all', 'cite all per paper'), ('cpp_last', 'cpp_last', 'cite last per paper'),
('cpp_single', 'cpp_single', 'cite single per paper')]
positive = lambda src, x: src[src[x].notna() & (src[x] > 0)]

(OUT / 'pay_exhibit.png').unlink(missing_ok=True) # the former main-text Figure 3 duplicated row c of Figure B2

# ----- Figure B2: author salaries, three rows by authorship class -----
pi = per[per.inc_mean.notna() & (per.n_inc >= 2) & ~per.modeled]
ROWS = [('pay_std', CITES, 'pay_std, dollars', True, per), ('pay_std', CPP, 'pay_std, dollars', True, per),
('pay_std', PAPERS, 'pay_std, dollars', True, per)] # short y labels stay within the axes height
FIG_HEIGHT = 5.23
fig, axes = plt.subplots(len(ROWS), 3, figsize=(W, FIG_HEIGHT), sharey='row') # three rows at the existing
panel height; title in caption only
for r, (y, cols, ylab, logy, src) in enumerate(ROWS):
    for c, (ax, (x, k, lab)) in enumerate(zip(axes[r], cols)):
        scatter_panel(ax, x, y, positive(src, x), lab, logx=True, logy=logy, key=k, stats=stats['rho'],
prefix=('abc'[r] + ' ') if c == 0 else '') # row letter in the first panel's title
        axes[r, 0].set_ylabel(ylab)
fig.subplots_adjust(left=0.12, right=0.99, top=1 - 0.2085 / FIG_HEIGHT, bottom=0.452 / FIG_HEIGHT, hspace=0.54,
wspace=0.22) # absolute top and bottom margins in inches; panel height about 1.12 in
fig.align_ylabels(axes[:, 0])
fig.canvas.draw(); rend = fig.canvas.get_renderer(); gap = 4 / 72 * fig.dpi # at least 4 pt between texts,
checked on the drawn canvas
for r in range(len(ROWS)):
    xl = [axes[r, c].xaxis.label.get_window_extent(rend) for c in range(3)]
    assert all(xl[c].x1 + gap < xl[c + 1].x0 for c in range(2)) and xl[0].x0 >= 0 and xl[2].x1 <= fig.bbox.x1,
f'x labels of row {r} overlap or leave the figure'
    if r < len(ROWS) - 1: assert all(axes[r, c].get_tightbbox(rend).y0 > axes[r + 1, c].get_tightbbox(rend).y1 +
gap for c in range(3)), f'row {r} x labels touch the statistics of row {r + 1}'
    assert all(axes[0, c].get_tightbbox(rend).y1 <= fig.bbox.y1 for c in range(3)) and all(axes[-1,
c].get_tightbbox(rend).y0 >= 0 for c in range(3)), 'text outside the canvas'
fig.savefig(OUT / 'pay_authorship_output.png', dpi=300); plt.close(fig); trim(OUT / 'pay_authorship_output.png')
# Annual-raise statistics remain in the JSON record only; there is no raise panel.
for x, k, _ in PAPERS:
    src = positive(pi, x)
    r, p, n = rho(np.log(src[x].values), src.inc_mean.values)
    result = {'n': n, 'rho': r, 'p': p}
    stats['rho_raise']['raise_' + k] = {'observed': result, 'all': result.copy()}
stats['n_raise_persons'] = int(len(pi))
(OUT / 'pay_shares_raises.png').unlink(missing_ok=True) # obsolete figure
# authorship shares (model terms of Table B3): correlations only, no panel
o = per[~per.modeled]
for k in ('last_pap_share', 'last_cit_share', 'single_pap_share', 'single_cit_share'):
    r, p, n = rho(o[k].values.astype(float), np.log(o.pay_std.values)); stats['rho_shares'][k] = {'observed':
{'n': n, 'rho': r, 'p': p}}

# ----- Figure B1: deans vs scientists (clean college-years only) -----
rr = ratios[ratios.ratio_dean_to_sci_active.notna() & (ratios.flag.fillna('') == '')]
# Scientists and deans use exactly the same clean fiscal years within each college.
# Pool scientist person-years, rather than taking an unweighted mean of annual means.
clean_years = rr[['college', 'fy']].drop_duplicates()
scientists = panel[panel.base].merge(clean_years, on=['college', 'fy'], how='inner', validate='many_to_one')
bc = rr.groupby('college').agg(dean=('dean_std', 'mean'), n_years=('fy', 'size')).reset_index()
bc = bc.merge(scientists.groupby('college').pay_std.mean().rename('sci'), on='college', validate='one_to_one')
bc['ratio'] = bc.dean / bc.sci; bc = bc.sort_values('ratio')
# A college with fewer than five clean years has a scientists' mean over those years that differs from U, because
pay
# changes from year to year. The left panel therefore standardizes over the same years (F_c = the scientists'
mean of
# those years): the scientists of every college sit at U, the dean at U times the ratio, and the ratios do not
change.
U = float(pd.read_excel(XL, sheet_name='Std factors').set_index('college').loc['UNIVERSITY (all colleges,
pooled)', 'mean5'])
bc['sci_plot'] = U; bc['dean_plot'] = U * bc.ratio
names = {'CAES': 'Agriculture', 'Franklin': 'Arts and Sciences', 'Vet Med': 'Veterinary Medicine', 'SPIA':
'Public and Intl Affairs', 'Terry': 'Business', 'Odum': 'Ecology', 'Warnell': 'Forestry', 'Public Health':
'Public Health', 'Pharmacy': 'Pharmacy', 'Education': 'Education', 'Engineering': 'Engineering', 'FACS': 'Family
and Consumer Sci.', 'Grady': 'Journalism'}

```

```

fig, (a1, a2) = plt.subplots(1, 2, figsize=(5.4, 3.9), gridspec_kw={'width_ratios': [1.55, 1]}, sharey=True)
yy = np.arange(len(bc))
ratio_labels = []
for i, r in enumerate(bc.itertuples()):
    a1.plot([r.sci_plot, r.dean_plot], [i, i], color=GRID, lw=2, zorder=1)
    a1.scatter(r.sci_plot, i, s=30, c=BLUE, zorder=3); a1.scatter(r.dean_plot, i, s=30, c=ORANGE, zorder=3)
    ratio_labels.append(a1.text(r.dean_plot + 7000, i, f'{r.ratio:.2f}x', va='center', fontsize=8, color=INK2))
a1.set_yticks(yy); a1.set_yticklabels([names.get(c, c) for c in bc.college]); a1.xaxis.set_major_formatter(kfmt)
a1.set_xlabel('standardized pay, dollars'); a1.grid(True, axis='x', color=GRID, lw=0.6); a1.set_axisbelow(True)
lo, hi = float(bc[['sci_plot', 'dean_plot']].min().min()), float(bc[['sci_plot', 'dean_plot']].max().max())
span = max(hi - lo, 1.0)
a1.set_xlim(lo - 0.08 * span, hi + 0.35 * span); a1.set_ylim(-0.8, len(bc) - 0.2)
years = [2021, 2022, 2023, 2024, 2025]; shades = ['#c9dcf3', '#9dbfe8', '#6ealdd', '#4188d9', '#1f5fb0']
for i, c in enumerate(bc.college):
    g = rr[rr.college == c]
    for y, sh in zip(years, shades):
        v = g[g.fy == y].ratio_dean_to_sci_active
        if len(v): a2.scatter(v.iloc[0], i, s=30, c=sh, edgecolors='white', linewidths=0.6, zorder=3)
a2.axvline(1, color=GRID, lw=1); a2.set_xlabel('dean / scientists, by year'); a2.grid(True, axis='x', color=GRID,
lw=0.6); a2.set_axisbelow(True)
a2.set_xlim(0.9, 2.6); a2.set_xticks([1.0, 1.5, 2.0, 2.5]); a2.tick_params(labelleft=False)
h1 = [plt.Line2D([], [], marker='o', ls='', color=BLUE, ms=6, label='listed scientists of the college, mean over
the same fiscal years'), plt.Line2D([], [], marker='o', ls='', color=ORANGE, ms=6, label='dean')]
h2 = [plt.Line2D([], [], marker='o', ls='', color=sh, ms=6, label=f'FY{y}') for y, sh in zip(years, shades)]
fig.legend(handles=h1, loc='lower center', ncol=2, frameon=False, bbox_to_anchor=(0.5, 0.045), columnspacing=1.2,
handletextpad=0.4)
fig.legend(handles=h2, loc='lower center', ncol=5, frameon=False, bbox_to_anchor=(0.5, -0.005),
columnspacing=1.2, handletextpad=0.4)
fig.suptitle('Deans and the listed scientists of their colleges, standardized pay', x=0.02, ha='left',
fontsize=9.5, color=INK)
fig.tight_layout(rect=(0, 0.1, 1, 0.95)); fig.canvas.draw()
# Reserve the measured label width plus four pixels inside the left axes.
left, right = a1.get_xlim(); renderer = fig.canvas.get_renderer()
for label in ratio_labels:
    fraction = (label.get_window_extent(renderer).width + 4) / a1.bbox.width
    assert fraction < 1, 'Figure B1 ratio label is wider than the left axes'
    right = max(right, left + (label.get_position()[0] - left) / (1 - fraction))
a1.set_xlim(left, right)
fig.savefig(OUT / 'deans_vs_scientists_std.png', dpi=300); plt.close(fig); trim(OUT /
'deans_vs_scientists_std.png')

stats.update({'dean_ratio_by_college': {r.college: float(r.ratio) for r in bc.itertuples()},
'dean_ratio_n_years': {r.college: int(r.n_years) for r in bc.itertuples()},
'dean_ratio_years': {c: sorted(int(y) for y in g.fy.unique()) for c, g in
clean_years.groupby('college')},
'dean_ratio_sci_mean': {r.college: float(r.sci) for r in bc.itertuples()},
'dean_ratio_dean_mean': {r.college: float(r.dean) for r in bc.itertuples()},
'dean_ratio_plot': {'U': U, 'dean': {r.college: float(r.dean_plot) for r in bc.itertuples()},
'definition': 'Figure B1 left panel standardized over the clean years of each
college: scientists at U, dean at U times the ratio'},
'definitions': {'observed person': 'mean standardized pay over the observed non-administrative
usable years FY2021 to 2025 (persons with any such year; modeled years of the same person are not used)',
'modeled-only person': 'mean of the carried-forward values FY2021 to 2025 for pre-
window retirees with no observed usable year',
'authorship classes': 'all positions (np, nc); last author (np_last = npsfl - cpsf,
nc_last = ncsfl - ncsf); single author (nps, ncs)',
'zero handling': 'last-author and single-author counts are plotted with 1 added;
citations-per-paper panels omit persons with no paper of that class (panel n stated); statistics keys carry the
field name without the +1 because ranks are unchanged',
'fit': 'ordinary least squares on the plotted scales (log-log where both axes are
logarithmic), observed persons only',
'raise': 'mean of year-to-year percentage changes of standardized pay over
consecutive observed non-administrative years, persons with at least two changes'}})
(JSON_OUT / 'pay_figures.json').write_text(json.dumps(stats, indent=1), encoding='utf-8')
print(json.dumps({k: v for k, v in stats.items() if k in ('n_persons', 'n_observed', 'n_modeled_only',
'n_raise_persons')}))
for k, v in stats['rho'].items(): print(k, v['observed'], v['all'])
for k, v in stats['rho_raise'].items(): print(k, v['observed'])
for k, v in stats['rho_shares'].items(): print(k, v['observed'])

```

15. manuscript_content.py (SHA-256 a0a69a46c8eef2a6...; 186 lines)

"""Shared text, tables, captions and references for both manuscript formats."""

```

import json, re, os
from data_core import ROOT

```

```

COUNTRIES={'usa':'United
States', 'can':'Canada', 'fin':'Finland', 'aus':'Australia', 'deu':'Germany', 'swe':'Sweden', 'fra':'France', 'gbr':'Uni

```

ted Kingdom','jpn':'Japan','che':'Switzerland','esp':'Spain','nzl':'New Zealand','chn':'China','ita':'Italy','aut':'Austria','nld':'Netherlands','dnk':'Denmark','pol':'Poland','svn':'Slovenia','prt':'Portugal','tur':'Turkey','cze':'Czechia','svk':'Slovakia','zaf':'South Africa','bra':'Brazil')
 AUTHOR='Chris J. Cieszewski'
 AFFILIATION='Quantfor Co., Watkinsville, GA, USA' # as in the typeset paper
 EMAIL='mail@cjci.net'

RELEASES={'2020':('3','19 Oct 2021'),'2021':('5','3 Nov 2022'),'2022':('6','4 Oct 2023'),'2023':('7','16 Sep 2024'),'2024':('8','19 Sep 2025')}

```
def make_tables(stats,norm):
    t={}
    rc=json.loads((ROOT/'build/revision_checks.json').read_text(encoding='utf-8'))
    OPTB='verification_case' not in os.environ.get('MS_MD','manuscript.md') # the paper (manuscript.md) has the
    Exhibit A/B structure; the archived verification-case draft has the earlier section numbers
    S31='A.5' if OPTB else '3.1';S5='A.5' if OPTB else '5';S23='A.4' if OPTB else '2.3'
    def table(key,caption,headers,rows,widths,note=''):
        t[key]={'key':key,'caption':caption,'headers':headers,'rows':rows,'widths':widths,'note':note}
        table('fields','Fields used from the career release: names in the extract and release (version 8),
        definitions in the release key sheet, and time windows.',['Extract name','Release column (v8)','Definition
        (release key)','Window'],[
            ['np','np6024','number of papers','papers 1960 to the data year'],
            ['nc','nc9624','total citations','citations 1996 to the data year'],
            ['nps / ncs','nps / ncs','single-authored papers / their citations','no paper or citation window
            stated'],
            ['cpsf / ncsf','cpsf / ncsf','single-or-first-authored papers / their citations','as nps / ncs'],
            ['npsfl / ncsfl','npsfl / ncsfl','single, first, or last-authored papers / their citations','as nps /
            ncs'],
            ['h / hm','h24 / hm24','Hirsch index [[cite:Hirsch_2005]] / coauthorship-adjusted hm index
            [[cite:Schreiber_2008]]','as of the end of the data year'],
            ['self%','self%','self-citation share of total citations','no window stated'],
            ['cprat','cprat','total citations divided by distinct citing papers','no window stated'],
            ['np cited','np6024 cited9624','papers cited at least once','papers 1960 to the data year'],
            ['np_rw','np6024_rw','papers marked as retractions in the Retraction Watch database','papers 1960 to the
            data year; versions 7 and 8 only'],
            ['sm-subfield-1 / -2; sm-field','same','first and second Science-Metrix subfield; top-ranked
            field','profile of the data year'],
            ['firstyr / lastyr','same','year of first / most recent publication',''],
            ],[.17,.19,.36,.28], 'Release column names end with the data-year suffix (6024 = 1960 to 2024, 9624 = 1996 to
            2024 in version 8; earlier versions use their own year). Citation columns other than the _rw and _d columns are
            provided with and without self-citations (suffix ns). This paper uses the all-citations columns unless stated
            otherwise. The extract removes the year suffixes from column names to combine the five releases. The different
            windows for np and the authorship paper counts explain the negative residuals in Section '+S23+'. Source: Key
            sheet of each release; Appendix C README.')
        table('deltas','Changes in the 2026 source record from the broad report (supplementary file S1, source day
        label 6) to the final report two days later (source day label 8) and this paper, with the purpose of each
        change.','Aspect|Broad report (source day label 6)|Final report (source day label 8) or this Article|Initiator|
        Analytical purpose'.split('|')),[
            ['Cohort','Forestry OR Ecology in either subfield slot; 23,374 author-release records; reported ecology
            dominance about 5:1','Forestry-anchored union (primary Forestry, or primary Ecology with secondary Forestry);
            3,886 records','Author','Align selection with the forestry question'],
            ['Career summaries','Accumulated career sums in summary columns; maps already averaged','Per-release
            averages and pooled career rates; no aggregate citation sums displayed','Author','Avoid presenting repeated
            careers as interval production'],
            ['Rank display','Top 20; residual bands of about 10%','Straight top-25 country listings','Author','Expose
            individual entries'],
            ['Measures','Counts of scientists, papers, citations','Nine rates: single / single-or-first / all
            authorship sets by papers per record, citations per record, citations per paper','Author','Distinguish scale from
            citation intensity'],
            ['Authorship','No position discrimination','Single / first / other decomposition; rank
            sensitivity','Author','Expose dependence on the credit rule'],
            ['Normalization','None against the whole list','Country representation index (a location quotient,
            Equation 1)','Companion brief','Compare Forestry share with overall listed share'],
            ['Reference cases','None','Three named profiles, now confined to one illustration','Author','Test
            interpretation against known profiles'],
            ['Coverage','No Scholar comparison','Documented Scholar snapshots; unique-work estimates
            withdrawn','Author','Identify differential coverage and counting limits'],
            ['Diagnostics','No integrity block','Self-citation, cprat, retractions, Gini, h and hm as descriptive
            indicators','Author, via supplied analysis guide','Separate screening from accusation'],
            ['Presentation and register','Small labels; service-oriented report','Regenerated figures; journal
            argument and reproducible exhibit','Author; companion brief; rebuild inspection','Make comparisons inspectable;
            separate evidence from interpretation']
            ],[.15,.225,.27,.14,.215], 'Sources: S1, the final report, handoff sections 4 and 6--8, and the running log
            (all in Appendix C). Initiator identifies who directed the recorded change: the author, the companion brief for
            this paper, or the later rebuild. It does not assign sole intellectual authorship. Normalization and adaptation
            for the journal followed the final report. Abbreviations: cprat, ratio of total citations to distinct citing
            papers; h, Hirsch index; hm, coauthorship-adjusted h index (Schreiber).')
        table('evidence','Recorded episodes of the work, the evidence in the record for each, the class of failure,
        and the lesson.','Episode|Evidence in the record (Appendix C)|Class|Lesson'.split('|')),[
```

['Broad Forestry-or-Ecology cohort','S1 Section 2 (population definition); the author\'s first either-slot mapping and later NAMING instruction in the log; S1 reports ecology dominance; 23,374 versus 3,886 records (Section A.1)','Revised research objective (author)','A correct extraction can answer the question asked rather than the one intended'],

['Northern Europe as intensity leader','First-draft sentence reported in log; final report Section 4.2 and its region figure; regional citations per paper (South America 63.0, Western Europe 53.7, Northern Europe 39.5); log entry "Adversarial verify workflow" says fixed but quotes no corrected sentence (Section '+S31+')','Interpretation error (tool)','Agreement of table values does not verify written comparisons'],

['Nash identity','Returned profile J. E. Nash (environmental engineering); intended J. F. Nash (game theory); author correction in handoff; full-file surname search of all five releases, nash_search.json (Section '+S5+')','AI implementation error (identity matching)','A name match does not verify identity'],

['Qualitative-oriented label','Classification rule in S1 (absence of a methods subfield); resulting 94% label','Interpretation error (concept)','A rule can run correctly without measuring the stated concept'],

['USDA and United States Unknown counts','USDA affiliation sequence 32 / 73 / 36 in S1 Table 8a; Unknown counts 28 to 110 in Table 7','Source data problem, unresolved','Sudden changes require source checking before explanation'],

['Old figure file','Final report Figure 4 uses ff_ratio_group.png (nine rates); ff4_group.png (count shares) is not embedded in the final report; original build_docx6.py and figure hashes in Appendix C (Section '+S31+')','Interpretation error (later rebuild)','Open the embedded image: an earlier draft of this paper mistook an unused file for the report figure'],

['Scholar raw counts','Log entry on reading only the first page; author-supplied counts 211 and 361; dedup_scholar.py and dedup_bettinger.py; Bettinger estimate about 271 in the log versus about 300 in the final report','AI implementation error, then conflicting reported estimates','A written counting rule does not reconcile conflicting estimates'],

['Georgia Tech enrollment','Author assumption in the log; contrary AI evidence; assumption withdrawn','Interpretation error (author), corrected by the tool','Oversight runs in both directions'],

['Top-20 to top-25 display','Instruction in the log; final report Table 5 (scientist and publication counts by country)','Presentation preference','Neither party made an error']

], [.2, .36, .19, .25], 'Class labels distinguish AI implementation error, revised research objective, source data problem, interpretation error (tool, author, or later rebuild), and presentation preference. Log entries are identified by their original headings in the running log (Appendix C, record folder). S1 is the broad report.')

table('cohort','Citation database releases used, listed author profiles, and credited career publications, by data year and nested authorship set.', ['Data year','Version (published)','Listed profiles','Single papers','Single-or-first papers','All papers'], [

[y,f"{RELEASES[y][0]} ({RELEASES[y][1]})","f"{v['n']:,}","f"{v['solo_p']:,}.0f","f"{v['sen_p']:,}.0f","f"{v['all_p']:,}.0f"] for y,v in stats['years'].items()

], [.12, .2, .16, .16, .2, .16], 'Data year: last year of career data in the release. Version and publication date: Elsevier Data Repository record 10.17632/btchxktzyw.N, where N is the version. Version 5 replaced version 4 for the 2021 data year. Listed profiles: author-release records in the cohort requiring Forestry (primary Forestry, or primary Ecology with secondary Forestry). Single papers: sum of nps (single-authored papers, no window stated); single-or-first papers: sum of cpsf; all papers: sum of np (papers 1960 to the data year). These are career counts summed over profiles in each release, rather than unique publications produced in that year. The panel cannot separate entry, exit, reclassification, affiliation change, and database revision. Source: cohort extract; checked against auth6.json (Appendix C).')

table('normalization','Representation index of primary-Forestry scientists by country, relative to all listed scientists, for the 2024 data year and pooled 2020--2024 data.', ['Country','Forestry count','All listed count','Forestry share (%)','All listed share (%)','RI 2024','RI pooled'], [

[COUNTRIES.get(r['country'],r['country']),f"{r['forestry']:,}","f"{r['all']:,}","f"{100*r['forestry_share']:.2f}","f"{100*r['all_share']:.2f}","f"{r['ri']:.2f}","f"{next(p['ri']) for p in norm['pooled']]['records'] if p['country']==r['country']:.2f}"] for r in norm['2024']]['records'] if r['country']

], [.25], [.22, .13, .14, .14, .115, .115], 'Forestry count F_c: listed profiles with primary subfield Forestry in country c. All listed count A_c: listed profiles of every primary subfield in country c. Shares are F_c/F and A_c/A, with world totals F = 733 and A = 230,333. A includes 252 records with unknown country; no primary-Forestry record has an unknown country. RI = (F_c/F)/(A_c/A), Eq. (1); RI = 1 means equal shares. RI pooled: the same ratio computed from the five releases 2020 to 2024 together, giving larger releases more weight. Top 25 countries by F_c; ties ordered by ISO code. Brazil, India, Indonesia, and Norway tie at four profiles for the 25th place; ISO order keeps Brazil and leaves out India, Indonesia, and Norway. Indonesia\'s RI of 22.05 would be the highest in the table. Annual values for every country and year are in annual_ri.csv (Appendix C). Source: world_subfield_year.csv, checked against the Forestry extract.')

table('nine','Nine rates computed by population and nested authorship set from the pooled 2020--2024 releases.', ['Population','Authorship set','Papers / record','Citations / record','Citations / paper'], [

[(('Whole cohort' if g=='cohort' else g+group'),label,f"{v[p+'ppa']:.1f}","f"{v[p+'cpa']:.1f}","f"{v[p+'cpp']:.1f}"] for g,v in stats['groups'].items()

for p,label in [(('solo','Single'),('sen','Single-or-first'),('all','All')]

], [.22, .22, .18, .2, .18], 'Whole cohort: the 3,886 author-release records requiring Forestry; Forestry group: the 3,271 with primary Forestry; Ecology group: the 615 with primary Ecology and secondary Forestry. Authorship sets are nested: single (fields nps, ncs), single-or-first (cpsf, ncsf), all (np, nc). Record means author-release record. Papers per record = sum of class papers / number of records; citations per record = sum of class citations / number of records; citations per paper = sum of class citations / sum of class papers. Within a set, citations per record equal papers per record times citations per paper. Thus, the three columns divide a measure into its components and are not independent measures. The values are ratios of pooled career-profile totals, not annual productivity, unique-document counts, or averages of individual ratios. All records, including those without single-authored papers, enter per-record denominators. Source: extract; auth6.json and final report Table 8.')

pa=rc['position_all'];pd=rc['position_drop_negative'];pp=rc['window']['position_excl_pre1960']

table('position','Pooled cohort (3,886 author-release records, 2020--2024) divided into mutually exclusive authorship positions.', ['Position','Papers (%)','Citations (%)','Citations / paper'], [

[label,f"{v['paper_share']:.0f}","f"{v['citation_share']:.0f}","f"{v['cpp']:.1f}"] for label,v in stats['position'].items())

]+[['All','100','100',f"{stats['groups']['cohort']['all_cpp']:.1f}"]], [.4,.19,.19,.22], f"Single: nps papers and ncs citations. First: first author of a multi-authored paper, cpsf minus nps papers and ncsf minus ncs citations. Other: every non-first coauthored position, np minus cpsf papers and nc minus ncsf citations. Papers (%) and citations (%) are shares of the pooled totals. Separately rounded shares may not sum to 100. Citations per paper: class citations divided by class papers. Window limitation (Section {S23}): np counts papers from 1960 to the data year, while no window is stated for nps and cpsf. Thus, the other-paper residual is approximate for the {rc['window']]['records_first_before_1960']} records with a first publication before 1960. It is negative for five of them (one profile, Adler). Without those {rc['window']]['records_first_before_1960']} records, paper shares are {pp['paper_share'][0]:.1f} / {pp['paper_share'][1]:.1f} / {pp['paper_share'][2]:.1f}% and citations per paper are {pp['cpp'][0]:.1f} / {pp['cpp'][1]:.1f} / {pp['cpp'][2]:.1f}. Excluding only the five negative-residual records gives {pd['paper_share'][0]:.1f} / {pd['paper_share'][1]:.1f} / {pd['paper_share'][2]:.1f}% and {pd['cpp'][0]:.1f} / {pd['cpp'][1]:.1f} / {pd['cpp'][2]:.1f} (full cohort {pa['paper_share'][0]:.1f} / {pa['paper_share'][1]:.1f} / {pa['paper_share'][2]:.1f}%). Source: extract; authorship.json and final report Table 9.")

table('taxonomy','University of Georgia scientists on the 2024 list whose forestry-school or related affiliation is not shown by the Science-Metrix subfield labels.', ['Scientist','Primary subfield in the database'],'Why a forestry reader would expect a different label'), [

['Bettinger','Forestry (secondary: Ecology)','Operations research and GIS; the secondary label does not describe the work'],

['Cieszewski','Forestry','Biometrics and modeling; there is no biometrics label'],

['Peterson','Ecology (secondary: Forestry)','Forest disturbance ecology in the forestry-anchored union'],

['Teskey','Plant Biology and Botany','Tree physiology, Warnell (retired)'],

['Markewitz','Agronomy and Agriculture','Forest soils, Warnell'],

['Grossman','Fishes','Stream ecology, Warnell (emeritus)'],

['Yabsley','Mycology and Parasitology','Wildlife disease, Warnell'],

['Woosnam','Sport, Leisure and Tourism','Nature-based tourism, Warnell'],

['Helfman','Fishes','Odum School of Ecology (emeritus); grouped with Warnell in the running log until the affiliation check']

], [.16,.3,.54], 'Warnell: Warnell School of Forestry and Natural Resources; Odum: Odum School of Ecology; both at the University of Georgia. Primary subfield: sm-subfield-1 of the 2024 data year. This classification is based on journals and does not provide a school faculty list. The database has no department field. Surnames and initials were matched to the school directory and checked by hand. Source: extract; running log; final report Tables 12 and 13.')

i=stats['integrity']

table('integrity','Integrity indicators describing the 854 author-release records of the 2024 data year.', ['Indicator','Value','Interpretive limit'], [

['Self-citation: median / maximum',f"{i['self_med']:.0f}% / {i['self_max']:.0f}%", 'Author-level proportions; not misconduct labels'],

['Authors above 25% self-citation',str(i['self_gt25']), 'Threshold is a descriptive screen'],

['Citations per distinct citing paper: median',f"{i['cprat_med']:.2f}", 'Does not rule out citation manipulation'],

['Retracted-paper occurrences / author records',f"{i['rw_papers']:.0f} / {i['rw_auth']}", 'Unique retracted documents are not identified'],

['Papers ever cited: median author share',f"{i['pctcited_med']:.0f}%", 'Median of author ratios; not a pooled share'],

['Citation Gini',f"{i['gini']:.2f}", 'Inequality within this selected population'],

['Citation share: top 10% / top 1%',f"{i['top10']:.0f}% / {i['top1']:.0f}%", 'Upper tails use the original script convention'],

['h / hm: separate cohort medians',f"{i['h_med']:.0f} / {i['hm_med']:.0f}", 'Credit-rule sensitivity; not fractions of merit']

], [.49,.17,.34], 'Self-citation: field self% of the release (share of an author\'s citations that are self-citations). Citations per distinct citing paper: field cprat (total citations 1996 to 2024 divided by the number of distinct citing papers). Retracted-paper occurrences: sum of np_rw (papers marked as retractions in the Retraction Watch database) over author records, and number of records with np_rw above zero. Papers ever cited: field np cited divided by np, per author. Gini: Gini coefficient of nc (total citations) across the 854 records, pairwise definition. Top 10% and top 1%: shares of total citations in records above the floor(0.9n) and floor(0.99n) positions in ascending order of nc. h: Hirsch index (field h); hm: coauthorship-adjusted index of Schreiber (field hm). Retraction fields are available only in the 2023 and 2024 releases. These indicators describe the data and do not test the stability of the main results or show that there are no integrity problems. Source: extract and integrity.json.')

table('coverage','Career-database and Google Scholar profiles recorded for two reference scientists. The definitions differ, so the comparison does not measure omissions from the database.', ['Measure','Cieszewski DB','Cieszewski Scholar','Bettinger DB','Bettinger Scholar'], [

['Raw listed works','74','211','207','361'],

['Career citations','1,878','4,131','3,495','10,619'],

['h-index','21','29','30','49'],

['DB / Scholar citations',f"{1878/4131*100:.0f}%", 'Reference',f"{3495/10619*100:.0f}%", 'Reference']

], [.3,.16,.19,.16,.19], 'DB: the career release with data through 2024 (raw listed works = np, papers 1960 to 2024; career citations = nc, citations 1996 to 2024; h-index = h). Scholar: Google Scholar profiles documented in the 2026 project record (raw listed works = number of profile entries; citations and h-index as displayed). These include citing material, document types, and duplicates excluded by the database. DB / Scholar citations: database total divided by Scholar total, using records with different definitions. The original estimates of unique Scholar works (about 170 and 270 to 300) are withdrawn because the rule for removing duplicates was never written down. The raw counts are retained. Source: extract, refs.json, final report Table 11, and running log.')

table('checklist','Disclosure and verification checklist for an AI-assisted submission, based on this example.', ['Item','What the editor should find in the manuscript or its package'], [

['Tools and tasks','Tools, model versions where recorded, dates, and the tasks assisted: retrieval, code, analysis, prose, checking; unknown versions reported as unavailable'],

['Definitions','Every inclusion predicate, denominator, and category label written out once, with its local meaning'],

```

['Sources and scripts','Frozen extracts, release identifiers, and executable scripts that produce every
numbered claim and figure'],
['Claim-level checks','For each comparative statement, the measure it names and the check that recovered
it'],
['Unresolved items','Identities, coverage gaps, and source artifacts flagged in the text, not only in a
log'],
['Independence','No certification of a narrative by the generator that wrote it; adversarial checks named
as such; author-requested AI critiques not presented as peer review'],
['Responsibility','A statement that the author can defend every claim, and the interests that bear on the
interpretation']
],[.22,.78], 'The items give the competence, verification, and disclosure conditions of Section 1 in a form
an editor can apply to any submission. Reviewers remain responsible for assessing scientific validity.')
    table('checklistB','The four characteristics of research production (Section 1) as conditions an editor can
look for in a manuscript before review.',['Condition','What the editor should find in the manuscript or its
package'],[
    ['Competence','The author can defend every aspect of the research. Every selection condition,
denominator, and category label is defined once, with its meaning in this study, and the limits of inference are
stated beside each claim'],
    ['Verification','Clear methods, selection and processing of data, and a tractable problem, so that what
is presented can be checked as authentic: saved unchanged inputs and executable scripts that produce every
numbered claim and figure; the measure and confirming check for each comparison; no certification of a narrative
by the generator that wrote it'],
    ['Replicability','The information needed to do the same kind of work by the same methods with your own
data or in your own environment: the data sources and their releases; the software or tool used for each task,
stated as a plain fact (e.g., SAS used to compute, Mathematica to derive); the settings and checks applied. A
statement that AI was used does not provide this information'],
    ['Contribution','A relevant result that the author can explain and defend and the readers can benefit
from: a question the author can state, and what the analysis lets readers understand that makes it worth
publishing, whether its purpose concerns theory, methods, explanation, or practical use']
],[.24,.76], 'All conditions are required. A manuscript that meets competence, verification, and
replicability but not contribution is trustworthy within its stated limits and still not worth publishing. A
manuscript that claims a contribution without the other three has transferred the burden of establishing its
basis to the reviewer. None of the four requires additional unpaid checking by reviewers. All four require work
by the author before submission. Disclosure of AI use is not one of them (Section 5).')
    return t

def make_figures():
    return {
        'normalization':{'file':'publishing_base_impact.png','caption':'Publishing base and impact for the 25
countries with the most listed primary-Forestry scientists in the 2024 data year (count after the name; ties
ordered by ISO code). Brazil, India, Indonesia, and Norway tie at four profiles for the 25th place; ISO order
keeps Brazil and leaves out India, Indonesia, and Norway. Indonesia's RI of 22.05 would be the highest in panel
A. A: representation index  $SRI_{c\$} = (\$F_{c\$}/\$F\$)/(\$A_{c\$}/\$A\$)$ , Eq. (1).  $\$F_{c\$}$ ,  $\$A_{c\$}$ : listed profiles with primary
subfield Forestry and with all subfields in country  $\$c\$$ .  $\$F\$ = 733$  and  $\$A\$ = 230,333$ : world totals. The vertical
line marks  $SRI_{c\$} = 1$ , equal shares of listed Forestry scientists and listed scientists overall. Filled green
markers and labels: 2024 data year. Hollow orange markers: ratio pooled over the five releases 2020 to 2024. The
axis is logarithmic. A high index shows specialization within the list and does not measure national quality.
Countries with few profiles change most between the two bases. B: citations per single-authored paper (ncs / nps)
against citations per paper in all positions (nc / np) for the same countries, pooled author-release records 2020
to 2024. Labels give the country and the two rates as single/all. The gray line marks equality. Rates for
countries with few profiles are based on few records, and the panel suggests different publishing cultures but
does not explain them. Data: Table [[ref:tab:normalization]] (A) and Appendix C (B); source:
world_subfield_year.csv, the extract, statistics.json; build/make_figures.py.'},
        # The archived verification-case draft still uses this figure; the paper does not.
        'ranks':{'file':'authorship_rank_sensitivity.png','caption':'Sensitivity of citation rankings to the
authorship basis for the 854 author-release records of the 2024 data year. Left: Spearman rank correlation (rho)
between total-citation ranks (nc) and ranks on three alternative bases, with the number of records common to the
top 25 of both rankings. Key (legacy) = citations to single, first, or last-author papers (ncsf1); single-or-
first = ncsf; single = ncs. Ties receive average ranks. Right: histogram of each record's rank change from total
citations to single-authored citations (single-author rank minus total-citation rank). Negative values mean a
better single-author rank. The histogram uses 50-position bins, with median absolute change 247. The database
composite indicator already includes authorship information, so differences from total-citation ranks are
expected and do not imply a problem with that indicator. Source: extract; rank_vectors.json (Appendix C).'},
        'integrity':{'file':'integrity_diagnostics.png','caption':'Citation concentration and coauthorship
adjustment for the 854 author-release records of the 2024 data year. Left: Lorenz curve of total citations (nc),
the cumulative share of citations held by the cumulative share of records ordered from least to most cited, with
the equality diagonal; the Gini coefficient is twice the area between the curves. Right: coauthorship-adjusted
index hm (Schreiber) against the Hirsch index h for every record, with the reference lines hm = h (gray) and hm =
h/2 (blue); a record on the lower line has, on average, half of the credit of its h-core papers assigned to
coauthors. Neither panel tests misconduct or measures intellectual contribution. Source: extract and
integrity.json (Appendix C).'}
    }

def where_ri(norm):
    """Variable list printed after Equation 1: "where:" followed by an itemized list, the form of the author's
MCFNS
papers (e.g., MCFNS 13(1) and 15(1)). Items are (TeX symbol, plain symbol, text without the leading
symbol)."""
    F=norm['2024']['forestry_total'];A=norm['2024']['all_total'];unk=norm['2024']['unknown_all']

```

```

    return [(r'\mathrm{RI}_c', 'RI_c', 'is the representation index of country $c$, a dimensionless ratio equal to
    unity when the country holds the same share of the listed Forestry scientists as of all listed scientists'),
            ('F_c', 'F_c', 'is the number of listed profiles with primary subfield Forestry in country $c$'),
            ('F', 'F', 'is the number of listed profiles with primary subfield Forestry in all countries ($F$ =
    {F:,} in the 2024 data year)'),
            ('A_c', 'A_c', 'is the number of listed profiles of all primary subfields in country $c$'),
            ('A', 'A', 'is the number of listed profiles of all primary subfields in all countries, including the
    {unk} records without a country ($A$ = {A:,} in the 2024 data year)')]

def load_content():
    stats=json.loads((ROOT/'build/statistics.json').read_text(encoding='utf-8'))
    norm=json.loads((ROOT/'build/normalization.json').read_text(encoding='utf-8'))
    tables=make_tables(stats,norm);figures=make_figures();refs=json.loads((ROOT/'build/
    references.json').read_text(encoding='utf-8'))
    chunks=re.split(r'\n\s*\n', (ROOT/'build'/os.environ.get('MS_MD','manuscript.md')).read_text(encoding='utf-8')
    .strip());blocks=[];counters={'table':0,'figure':0};lookup={};prefix=''
    for chunk in chunks:
        text=' '.join(chunk.splitlines()).strip()
        if text.startswith('# '):blocks.append({'type':'title','text':text[2:]})
        elif text.startswith('### '):blocks.append({'type':'subsection','text':text[4:]})
        elif text.startswith('## '):
            blocks.append({'type':'section','text':text[3:]})
            app=re.match(r'## Appendix ([A-Z]):',text)
            if app:
                prefix=app.group(1);counters={'table':0,'figure':0}
            elif text.startswith('{{table:}}') or text.startswith('{{figure:}}'):
                kind,key=text[2:-2].split(':');counters[kind]+=1
                b=(tables if kind=='table' else figures)
            [key].copy();b.update(type=kind,number=prefix+str(counters[kind]),key=key);blocks.append(b)
            lookup['{{tab' if kind=='table' else 'fig'}}'+':'+key]=prefix+str(counters[kind])
            elif text=='{{equation:ri}}':blocks.append({'type':'equation','text':f'RI_c = (F_c / F) / (A_c / A) = (F_c
    / A_c) / (F / A)', 'tex':r'\mathrm{RI}_c=\frac{F_c/F}{A_c/A}=\frac{F_c/A_c}{F/A}\quad'})
            elif text=='{{where:ri}}':blocks.append({'type':'where','items':where_ri(norm)})
            elif re.match(r'^\d+\.\s+',text): # consecutive "1. ..." chunks form one numbered list
                item=re.sub(r'^\d+\.\s+', '',text)
                if blocks and blocks[-1]['type']=='enumerate':blocks[-1]['items'].append(item)
                else:blocks.append({'type':'enumerate','items':[item]})
            else:blocks.append({'type':'paragraph','text':text})
    return blocks,refs,lookup

def plain(text,refs,lookup):
    rd={r['key']:r for r in refs}
    text=re.sub(r'\$([^\$]+\$)',lambda m:re.sub(r'\mathrm{([^\$]*)}',r'\1',m.group(1)),text) # inline
    mathematics as plain symbols (DOCX, checks)
    def sub(m):
        kind,keys=m.group(1),m.group(2)
        if kind=='ref':return lookup[keys]
        pieces=[]
        for key in keys.split(';'):
            r=rd[key];pieces.append(r['author']+ ('+r['year']+)' if kind=='text' else r['author']+
    '+r['year'])
        result='; '.join(pieces)
        return '('+result+)' if kind=='cite' else result
    return re.sub(r'\[([cite|text|ref]:([^\$]+\$))\]',sub,text).replace('--','').replace('\ ',' ')

def tex_escape(text):
    # Keep the ASCII TeX contract while preserving names in T1 fonts.
    accents={'ä':r'\{a}','ö':r'\{o}','ü':r'\{u}','Ä':r'\{A}','Ö':r'\{O}','Ü':r'\{U}','ñ':r'\{n}','ó':r'\{o}'}
    o}', '$':r'\$','!':r'\!','\':r'\','"':r'\',"':r'\''}'
    return ''.join({'&':r'\&','%':r'\%','_':r'\_','#':r'\#','$':r'\$','{':r'\{','}':r'\}','~':r'\textasciitilde}','^':r'\textasciicircum}','\':r'\textbackslash'}).get(c,accents.get(c,c))
    for c in text)

def latex(text):
    """Prose to TeX: [[cite:...]], [[text:...]], [[ref:...]] become natbib/ref commands; a $...$ span is inline
    mathematics and passes through unescaped (the prose sources use no other dollar signs); everything else is
    escaped."""
    out=[];i=0
    for m in re.finditer(r'\[([cite|text|ref]:([^\$]+\$))\]|\\$[^\$]+\$|\\ ',text):
        out.append(tex_escape(text[i:m.start()]))
        if m.group(0).startswith('$') or m.group(0)=='\\ ':out.append(m.group(0));i=m.end();continue # inline
    math; TeX control space after an abbreviation
    kind,keys=m.groups()
    out.append('\\'+{'cite':'citep','text':'citet','ref':'ref'}[kind]+'{'+keys.replace(';',' ','')
    +'})';i=m.end()
    out.append(tex_escape(text[i:]));return ''.join(out)

```

16. nash_search.py (SHA-256 8283fc2873e9c38f...; 21 lines)

```
"""Search every available science-wide career release for surname Nash (reviewer request: full files, not the
forestry extract)."""
import json,time
from pathlib import Path
from openpyxl import load_workbook
B=Path(r'C:\D\Dropbox\Administrative\FAR\StanfordList')
files={'v3 (2020 data)':B/'btchxktzyw-3/Table_1_Authors_career_2020_wopp_extracted_202108.xlsx',
       'v5 (2021 data)':B/'v5_2021/Table_1_Authors_career_2021_pubs_since_1788_wopp_extracted_202209b.xlsx',
       'v6 (2022 data)':B/'V6_2022/Table_1_Authors_career_2022_pubs_since_1788_wopp_extracted_202310.xlsx',
       'v7 (2023 data)':B/'V7-2023/Table_1_Authors_career_2023_pubs_since_1788_wopp_extracted_202408.xlsx',
       'v8 (2024 data)':B/'V8_2024/Table_1_Authors_career_2024_pubs_since_1788_wopp_extracted_202508.xlsx'}
out={}
for lab,p in files.items():
    t=time.time();wb=load_workbook(p,read_only=True);ws=wb[wb.sheetnames[0]] if 'Data' not in wb.sheetnames else
wb['Data']
    rows=ws.iter_rows(values_only=True);hdr=[str(x) for x in next(rows)]
    ia=hdr.index('authfull');ii=hdr.index('inst_name');ic=hdr.index('cntry');isf=hdr.index('sm-
subfield-1');inp=next(i for i,h in enumerate(hdr) if h.startswith('np'));ifi=hdr.index('firstyr')
    hits=[];total=0
    for r in rows:
        total+=1;a=r[ia]
        if a and
str(a).startswith('Nash,'):hits.append({'authfull':a,'inst':r[ii],'cntry':r[ic],'subfield':r[isf],'np':r[inp],'fi
rstyr':r[ifi]})
    out[lab]={ 'sheet':ws.title,'rows':total,'nash':hits};print(lab,total,len(hits),round(time.time()-t))
json.dump(out,open('nash_search.json','w'),indent=1,default=str)
```

17. ooxml_fallback.py (SHA-256 e9b0b4ba708064b0...; 82 lines)

"""Explicit fallback for a session without an executable python-docx runtime.

Uses the read-only python-docx default template from the locally cached wheel.
No third-party package is downloaded. This is not reported as python-docx
execution. The normal build_docx.py path remains the requested implementation.

```
"""
import io,zipfile,struct,html,xmle.etree.ElementTree as ET
from manuscript_content import *

TEMPLATE_WHEEL=r'C:\Users\Administrator\AppData\Local\pip\Cache\http-
v2\9\9\1\d\e\991de6b156d764c62f0de8035b40aa71a5a3959e54ef1e9b5be454fe.body'
W='http://schemas.openxmlformats.org/wordprocessingml/2006/main'
R='http://schemas.openxmlformats.org/officeDocument/2006/relationships'
def esc(s):return html.escape(str(s),quote=True)
def para(s,style='Normal',keep=False,center=False):
    prop='<w:pStyle w:val="'+style+'"/>'+'(<w:keepNext/>' if keep else '')+'(<w:jc w:val="center"/>' if center
else '')
    return '<w:p><w:pPr>'+prop+'</w:pPr><w:r><w:t xml:space="preserve">'+esc(s)+'</w:t></w:r></w:p>'

def build():
    with zipfile.ZipFile(TEMPLATE_WHEEL) as z:template=z.read('docx/templates/default.docx')
    with zipfile.ZipFile(io.BytesIO(template)) as z:files={n:z.read(n) for n in z.namelist()}
    ET.register_namespace('w',W)
    styles=ET.fromstring(files['word/styles.xml']);ns={'w':W}
    styles.attrib.pop('{http://schemas.openxmlformats.org/markup-compatibility/2006}Ignorable',None)
    for s in styles.findall('w:style',ns):
        sid=s.get('{'+W+'}styleId')
        if sid not in ['Normal','Title','Heading1','Heading2','Caption']:continue
        for tag in ['rPr','pPr']:
            old=s.find('w:'+tag,ns)
            if old is not None:s.remove(old)
        size={'Normal':21,'Title':30,'Heading1':24,'Heading2':22,'Caption':21}[sid]
        bold='<w:b>' if sid in ['Title','Heading1','Heading2'] else ''
        rp=ET.fromstring('<w:rPr xmlns:w="'+W+'"><w:rFonts w:ascii="Times New Roman" w:hAnsi="Times New
Roman"/><w:color w:val="000000"/><w:sz w:val="'+str(size)+'"/><w:szCs w:val="'+str(size)+'"/>'+bold+'</w:rPr>')
        keep='<w:keepNext/>' if sid in ['Heading1','Heading2'] else ''
        pp=ET.fromstring('<w:pPr xmlns:w="'+W+'">'+keep+'<w:widowControl/><w:spacing w:before="'+('180' if sid in
['Heading1','Heading2'] else '0')+'"" w:after="100" w:line="264" w:lineRule="auto"/></w:pPr>')
        s.append(pp);s.append(rp)
    files['word/styles.xml']=ET.tostring(styles,encoding='utf-8',xml_declaration=True)
    blocks,refs,lookup=load_content();body=[];rel=[];medias={}
    def p(s,style='Normal',keep=False):body.append(para(plain(s,refs,lookup),style,keep))
    p(blocks[0]['text'],'Title');p(AUTHOR);p(AFFILIATION);p(EMAIL);p('MCFNS 18(1) companion draft')
    for b in blocks[1:]:
        k=b['type']
        if k=='section':p(b['text'],'Heading1')
        elif k=='subsection':p(b['text'],'Heading2')
```

```

elif k in ['paragraph','equation']:p(b['text'])
elif k=='table':
    p('Table '+str(b['number'])+'. '+b['caption'],'Caption',True)
    widths=[round(9936*w/sum(b['widths'])) for w in b['widths'];widths[-1]=9936-sum(widths[:-1])
    borders=''.join('<w:'+e+' w:val="single" w:sz="4" w:color="D9D9D9"/>' for e in
['top','left','bottom','right','insideH','insideV'])
    tab='<w:tbl><w:tblPr><w:tblW w:w="9936"
w:type="dxa"/><w:tblBorders'+borders+'</w:tblBorders><w:tblLayout w:type="fixed"/><w:tblCellMar><w:top w:w="55"
w:type="dxa"/><w:left w:w="70" w:type="dxa"/><w:bottom w:w="55" w:type="dxa"/><w:right w:w="70"
w:type="dxa"/></w:tblCellMar></w:tblPr><w:tblGrid>'+''.join('<w:gridCol w:w="'+str(w)+'"/>' for w in
widths)+'</w:tblGrid>'
    for rowno,row in enumerate([b['headers']] + b['rows']):
        tab+=''<w:tr><w:trPr><w:cantSplit/>'+('<w:tblHeader/>' if rowno==0 else '')+'</w:trPr>'
        for j,(val,w) in enumerate(zip(row,widths)):
            tab+=''<w:tc><w:tcPr><w:tcW w:w="'+str(w)+'" w:type="dxa"/><w:shd w:fill="'+('E8CEEA' if
rowno==0 else 'FFFFFF')+'"/><w:VAlign w:val="center"/></w:tcPr>'
            tab+=''<w:p><w:pPr>'+('<w:keepNext/>' if b['key']=='position' and rowno<len(b['rows']) else
'')+('<w:spacing w:after="20" w:line="240" w:lineRule="auto"/><w:jc w:val="'+('left' if j==0 or b['key']=='deltas'
else 'center')+'"/></w:pPr><w:r><w:rPr><w:sz w:val="20"/>'+('<w:b/>' if rowno==0 else
'')+('</w:rPr><w:t>'+esc(plain(str(val),refs,lookup))+'</w:t></w:r></w:p></w:tc>'
            tab+=''</w:tr>'
        body.append(tab+'</w:tbl>');p(b['note'])
    elif k=='figure':
        data=(ROOT/'figures'/b['file']).read_bytes();px,py=struct.unpack('>II',data[16:24]);cx=6309360;cy=rou
nd(cx*py/px);rid='rIdFig'+str(b['number']);medias['word/media/'+b['file']]=data
        rel.append((rid,R+'image','media/'+b['file']))
        body.append('<w:p><w:pPr><w:keepNext/><w:spacing w:before="80"
w:after="30"/></w:pPr><w:r><w:drawing><wp:inline distT="0" distB="0" distL="0" distR="0"><wp:extent cx="'+str(cx)
+'" cy="'+str(cy)+'"/><wp:docPr id="'+str(b['number'])+'" name="'+b['file']+'"
descr="'+esc(plain(b['caption'],refs,lookup))+'"/><a:graphic><a:graphicData
uri="http://schemas.openxmlformats.org/drawingml/2006/picture"><pic:pic><pic:nvPicPr><pic:cNvPr id="0"
name="'+b['file']+'"/><pic:cNvPicPr></pic:nvPicPr><pic:blipFill><a:blip
r:embed="'+rid+'"/><a:stretch><a:fillRect/></a:stretch></pic:blipFill><pic:spPr><a:xfrm><a:off x="0"
y="0"/><a:ext cx="'+str(cx)+'" cy="'+str(cy)+'"/></a:xfrm><a:prstGeom
prst="rect"><a:avLst/></a:prstGeom></pic:spPr></pic:pic></a:graphicData></a:graphic></wp:inline></w:drawing></
w:r></w:p>')
        p('Figure '+str(b['number'])+'. '+b['caption'],'Caption')
        p('References','Heading1')
        for r in refs:p(r['text']+((' '+r['url']+'.' if r['url'] else ''))
        rel.extend([(('rIdHdr',R+'/header','header1.xml'),('rIdFtr',R+'/footer','footer1.xml')])
        body.append('<w:sectPr><w:footerReference w:type="default" r:id="rIdFtr"/><w:pgSz w:w="12240"
w:h="15840"/><w:pgMar w:top="1080" w:right="1152" w:bottom="1080" w:left="1152" w:header="480"
w:footer="480"/><w:cols w:space="720"/></w:sectPr>')
        files['word/document.xml']=('<?xml version="1.0" encoding="UTF-8" standalone="yes"?><w:document
xmlns:w="'+W+'" xmlns:r="'+R+'" xmlns:wp="http://schemas.openxmlformats.org/drawingml/2006/wordprocessingDrawing"
xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"
xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture"><w:body>'+''.join(body)+'</w:body></
w:document>').encode()
        files['word/header1.xml']=('<w:hdr xmlns:w="'+W+'">+para('Cieszewski | AI as a research instrument | MCFNS
companion draft')+'</w:hdr>').encode()
        files['word/footer1.xml']=('<w:ftr xmlns:w="'+W+'"><w:p><w:pPr><w:jc w:val="right"/></w:pPr><w:fldSimple
w:instr="PAGE"><w:r><w:t>1</w:t></w:r></w:fldSimple></w:p></w:ftr>').encode()
        rr=ET.fromstring(files['word/_rels/document.xml.rels']);rn='{http://schemas.openxmlformats.org/package/2006/
relationships}'
        for rid,typ,target in rel:ET.SubElement(rr,rn+'Relationship',{'Id':rid,'Type':typ,'Target':target})
        ET.register_namespace('','http://schemas.openxmlformats.org/package/2006/relationships')
        files['word/_rels/document.xml.rels']=ET.tostring(rr,encoding='utf-8',xml_declaration=True)
        ct=ET.fromstring(files['Content_Types.xml']);cn='{http://schemas.openxmlformats.org/package/2006/content-
types}'
        for name,typ in
[('/',word/header1.xml','header'),('/',word/footer1.xml','footer')]:ET.SubElement(ct,cn+'Override',
{'PartName':name,'ContentType': 'application/vnd.openxmlformats-officedocument.wordprocessingml.'+typ+'.xml'})
        if not any(x.get('Extension')=='png' for x in ct):ET.SubElement(ct,cn+'Default',
{'Extension': 'png','ContentType': 'image/png'})
        ET.register_namespace('','http://schemas.openxmlformats.org/package/2006/content-types')
        files['Content_Types.xml']=ET.tostring(ct,encoding='utf-8',xml_declaration=True)
        files.update(medias)
        with zipfile.ZipFile(ROOT/'Cieszewski2026_AI_draft.docx','w',zipfile.ZIP_DEFLATED) as z:
            for name,data in files.items():z.writestr(name,data)

if __name__=='__main__':build()

```

18. pay_sensitivity.py (SHA-256 d2d182dbb7bc3466...; 103 lines)

"""Run the four checked-college screening variants in isolated workbook copies.

Run the checked production pipeline first. The base variant must reproduce every collected production value (relative and absolute tolerance 1e-9, NaNs equal).

Only build/tmp/sensitivity/ and build/pay_sensitivity.json are written.

```
"""
import json
import math
import os
import shutil
import subprocess
import sys
from pathlib import Path

ROOT = Path(__file__).resolve().parents[1]
BUILD = ROOT / 'build'
WORKBOOK = ROOT / 'UGA StanfordList_Salaries_FY2021-2025.xlsx'
VARIANTS = {'base': ('50000', '0.8'), 'no_floor': ('0', '0.8'),
             'no_exit_rule': ('50000', 'none'), 'neither': ('0', 'none')}
FEATURES = ('np', 'np_last', 'nps', 'nc', 'nc_last', 'ncs', 'cpp_all', 'cpp_last', 'cpp_single')
MODELS = ('E_active_mixed', 'E_active_ols_cluster', 'E_mixed', 'E_ols_cluster',
          'F_mixed', 'F_ols_cluster', 'F_active_mixed', 'F_active_ols_cluster')
TERMS = ('lnp', 'lcpp', 'last_pap', 'single_cit', 'lcpp_last', 'lcpp_single')

def collect(directory):
    u = json.loads((directory / 'ugapay.json').read_text(encoding='utf-8'))
    pf = json.loads((directory / 'pay_figures.json').read_text(encoding='utf-8'))
    assert u.get('college_source') == 'checked', f'Run the checked production pipeline first: {directory}'
    models = {}
    for name in MODELS:
        model = u['models'][name]
        models[name] = {'n_obs': model['n_obs'], 'n_persons': model['n_persons'],
                       'params': {term: {key: model['params'][term][key] for key in ('coef', 'se', 'p')}
                                for term in TERMS if term in model['params']}}
    return {'panel_counts': u['panel_counts'], 'n_observed': pf['n_observed'],
           'n_modeled_only': pf['n_modeled_only'],
           'rho': {feature: {key: pf['rho'][feature]['observed'][key] for key in ('rho', 'p', 'n')}
                  for feature in FEATURES},
           'lnp_range': u['lnp_range'], 'models': models,
           'ratio_min': u['ratio_min'], 'ratio_max': u['ratio_max'],
           'dean_ratio_by_college': pf['dean_ratio_by_college']}

def assert_equivalent(actual, expected, path='base'):
    if isinstance(expected, dict):
        assert isinstance(actual, dict) and actual.keys() == expected.keys(), f'{path}: keys differ'
        for key in expected: assert_equivalent(actual[key], expected[key], f'{path}.{key}')
    elif isinstance(expected, list):
        assert isinstance(actual, list) and len(actual) == len(expected), f'{path}: lengths differ'
        for i, (a, e) in enumerate(zip(actual, expected)): assert_equivalent(a, e, f'{path}[{i}]')
    elif isinstance(expected, (int, float)) and not isinstance(expected, bool):
        assert isinstance(actual, (int, float)) and not isinstance(actual, bool), f'{path}: nonnumeric value'
        assert (math.isnan(actual) and math.isnan(expected)) or math.isclose(actual, expected, rel_tol=1e-9,
abs_tol=1e-9), f'{path}: {actual!r} != {expected!r}'
    else:
        assert actual == expected, f'{path}: {actual!r} != {expected!r}'

def main():
    published = collect(BUILD)
    results = {}
    for variant, (floor, exit_ratio) in VARIANTS.items():
        work = BUILD / 'tmp' / 'sensitivity' / variant
        work.mkdir(parents=True, exist_ok=True)
        xl = work / WORKBOOK.name
        shutil.copy2(WORKBOOK, xl)
        env = os.environ.copy()
        env.update(PAY_XL=str(xl), PAY_OUT=str(work), PAY_FIG_OUT=str(work),
                  PAY_FLOOR=floor, PAY_EXIT_RATIO=exit_ratio, PAY_COLLEGES='checked',
                  PYTHONDONTWRITEBYTECODE='1', MPLCONFIGDIR=str(work / 'mplconfig'))
        log_path = work / 'pipeline.log'
        print(f'Running {variant}; log: {log_path.relative_to(ROOT)}', flush=True)
        with log_path.open('w', encoding='utf-8') as log:
            for script in ('redo_standardized.py', 'compute_ugapay.py', 'make_pay_figures.py'):
                log.write(f'\n{script}\n'); log.flush()
                subprocess.run([sys.executable, str(BUILD / script)], cwd=ROOT, env=env,
                               stdout=log, stderr=subprocess.STDOUT, check=True)
        results[variant] = collect(work)
        if variant == 'base': assert_equivalent(results[variant], published)
    results['definitions'] = {
        'college_source': 'checked; build/college_checked.csv is shared by all variants',
    }
```

```

    'variants': {name: {'PAY_FLOOR': int(floor), 'PAY_EXIT_RATIO': ratio, 'PAY_COLLEGES': 'checked'}
                  for name, (floor, ratio) in VARIANTS.items()},
    'panel_counts': 'copied from ugapay.json; observed and modeled person-years and persons',
    'n_observed / n_modeled_only': 'Figure B2 persons, copied from pay_figures.json',
    'rho': 'observed-person Spearman rho, p and n for the nine Table B2 features; no modeled-only persons',
    'lnp_range': 'copied from ugapay.json without changing its existing model selection or rounding',
    'models': 'coef, se and p of the requested terms where present; model-specific n_obs and n_persons',
    'ratio_min / ratio_max': 'minimum / maximum clean annual dean-to-scientist ratio from ugapay.json',
    'dean_ratio_by_college': 'Figure B1 left: mean dean pay divided by person-year-weighted mean observed
screened scientist pay over the same clean fiscal years',
    'base_validation': 'all collected base values compared recursively with production build/ugapay.json and
build/pay_figures.json; relative and absolute tolerance 1e-9; NaNs equal',
}
(BUILD / 'pay_sensitivity.json').write_text(json.dumps(results, indent=2) + '\n', encoding='utf-8')
print('\nvariant      observed PY   modeled PY   observed persons   modeled only   lnp range      annual dean
ratio')
for name in VARIANTS:
    r = results[name]; pc = r['panel_counts']
    lo, hi = r['lnp_range']
    print(f'{name:14s} {pc[\"rows_observed\"]:11d} {pc[\"rows_imputed\"]:11d} {r[\"n_observed\"]:17d}
{r[\"n_modeled_only\"]:13d} {lo:.3f}..{hi:.3f} {r[\"ratio_min\"]:13f}..{r[\"ratio_max\"]:13f}')
print('PASS: base matches production for every collected value (tolerance 1e-9).')

if __name__ == '__main__':
    main()

```

19. paynames.py (SHA-256 83b400e6c25bf9da...; 19 lines)

"""Name key for joining the pay panel to the citation records of the 2024 release.

A few names are spelled differently in the payroll-matched workbook sheets (accents removed) and in the release ('Díaz-Pérez', with a dotless i, is 'Diaz-Perez' in the pay sheets). The key folds accents, maps letters that Unicode does not decompose, lowercases, and keeps letters only. Used by redo_standardized.py, compute_ugapay.py and make_pay_figures.py.

```

"""
import re, unicodedata

_EXTRA = str.maketrans({'ı': 'i', 'İ': 'I', 'Ł': 'L', 'Ø': 'O', 'ø': 'O', 'đ': 'd', 'Đ': 'D', 'ß': 'ss', 'æ':
'ae', 'Æ': 'AE', 'œ': 'oe', 'Œ': 'OE'})

def auth_key(s):
    return re.sub('[^a-z]', '', unicodedata.normalize('NFKD', str(s).translate(_EXTRA)).encode('ascii',
'ignore').decode()).lower()

def join_idx(frame, idx):
    """frame joined to the citation records idx by auth_key; the frame keeps its own 'auth' spelling."""
    i = idx.assign(_k=idx.auth.map(auth_key)).drop(columns='auth')
    assert not i._k_duplicated().any(), 'two citation records share a name key'
    return frame.assign(_k=frame.auth.map(auth_key)).merge(i, on='_k').drop(columns='_k')

```

20. pdfclean.py (SHA-256 d7506332efb77790...; 38 lines)

"""Remove production timestamps and XMP from generated publication PDFs."""

```

from pathlib import Path
import os
import re
import sys
import tempfile
import time

sys.path.insert(0, str(Path(__file__).resolve().parent / 'vendor'))
import pymupdf as fitz

# a /CreationDate or /ModDate entry of the document information dictionary, with its value: null, a literal
# string such as (D:20260101000000-04'00'), or a hexadecimal string
DATE_ENTRY = re.compile(r'/(?:CreationDate|ModDate)\s*(?:null|\\((?:\\.|\\[^\])\\))*\\)|<[0-9A-Fa-f\\s]*>\\s*")

def strip_pdf_dates(path):
    """Replace a PDF atomically with a copy without date entries or XMP metadata."""
    path = Path(path)
    fd, temporary = tempfile.mkstemp(prefix=path.stem + '-', suffix='.pdf', dir=path.parent)

```

```

os.close(fd)
temporary = Path(temporary)
try:
    with fitz.open(path) as doc:
        kind, value = doc.xref_get_key(-1, 'Info')
        if kind == 'xref': # delete the two date entries from the information dictionary itself
            number = int(value.split()[0])
            doc.update_object(number, DATE_ENTRY.sub('', doc.xref_object(number, compressed=False))
            doc.del_xml_metadata()
            doc.save(temporary, garbage=4, deflate=True, no_new_id=True)
        for attempt in range(40): # Dropbox or a virus scanner may hold a just-written file for a moment
            try:
                os.replace(temporary, path); break
            except PermissionError:
                if attempt == 39: raise
                time.sleep(0.5)
finally:
    temporary.unlink(missing_ok=True)

```

21. profiles_admin_warnell.py (SHA-256 54e42e056b6c959a...; 72 lines)

```

"""Authorship profiles of (a) UGA Stanford-list scientists holding administrative titles and (b) Warnell School
members.
Writes two sheets into the companion workbook and prints compact tables. Requires pandas, openpyxl.
"""
import json, pandas as pd, numpy as np
from pathlib import Path
ROOT = Path(__file__).resolve().parents[1]
XL = ROOT / 'UGA_StanfordList_Salaries_FY2021-2025.xlsx'
idx = pd.read_excel(XL, sheet_name='Authorship indices (321)').rename(columns={'auth': 'authfull'})
long = pd.read_excel(XL, sheet_name='Scientists long')
roster = pd.read_excel(XL, sheet_name='Scientists x FY')[['auth', 'school']]

ADMIN = r'DEAN|CHAIR|HEAD|DIRECTOR|PROVOST|PRESIDENT|CHIEF'
# admin titles per person: which years, which titles, latest pay
t = long.groupby(['auth', 'fy']).agg(title=('title', lambda s: ' | '.join(sorted(set(s)))), pay=('salary',
'sum')).reset_index()
t['is_admin'] = t.title.str.contains(ADMIN)
adm = t[t.is_admin].groupby('auth').agg(admin_titles=('title', lambda s: ' | '.join(sorted(set(s)))),
admin_years=('fy', lambda s: ', '.join(map(str, sorted(s)))).reset_index()
pay = t.pivot_table(index='auth', columns='fy', values='pay', aggfunc='sum'); pay.columns = [f'pay_FY{c}' for c
in pay.columns]

_R = json.load(open(ROOT / 'build' / 'roster_overrides.json', encoding='utf-8'))
WARNELL = _R['warnell']; WARNELL_NOTE = _R['warnell_note']

def profile(df):
    p = df.copy()
    p['first_pap_share'] = p.np_first / p.np; p['last_pap_share'] = p.np_last / p.np; p['other_pap_share'] =
p.np_other / p.np
    p['first_cit_share'] = p.nc_first / p.nc; p['last_cit_share'] = p.nc_last / p.nc; p['other_cit_share'] =
p.nc_other / p.nc
    p['subfield_pct'] = 100 * p['rank sm-subfield-1'] / p['sm-subfield-1 count']
    cols = ['authfull', 'sm-subfield-1', 'firstyr', 'np', 'nc', 'h', 'hm', 'self%', 'nps', 'ncs', 'np_first',
'nc_first', 'np_last', 'nc_last', 'np_other', 'nc_other',
'I_pap_single_vs_all', 'I_cit_single_vs_all', 'first_pap_share', 'first_cit_share', 'last_pap_share',
'last_cit_share', 'other_pap_share', 'other_cit_share',
'I_pap_single_vs_last', 'I_cit_single_vs_last', 'cpp_single', 'cpp_first', 'cpp_last', 'cpp_all',
'rank sm-subfield-1', 'sm-subfield-1 count', 'subfield_pct']
    return p[cols].rename(columns={'authfull': 'auth', 'I_pap_single_vs_all': 'single_pap_share',
'I_cit_single_vs_all': 'single_cit_share', 'I_pap_single_vs_last': 'single_vs_last_pap', 'I_cit_single_vs_last':
'single_vs_last_cit'})

prof = profile(idx)
admin_prof = prof.merge(adm, on='auth').merge(roster, on='auth', how='left').merge(pay, left_on='auth',
right_index=True, how='left')
admin_prof = admin_prof.sort_values(['admin_titles', 'auth'])
warn_prof = prof[prof.auth.isin(WARNELL)].copy(); warn_prof['warnell_status'] = warn_prof.auth.map(WARNELL_NOTE)
warn_prof = warn_prof.merge(pay, left_on='auth', right_index=True,
how='left').merge(t.groupby('auth').title.agg(lambda s: ' | '.join(sorted(set(s)))).rename('titles_FY21_25'),
left_on='auth', right_index=True, how='left')
warn_prof = warn_prof.set_index('auth').loc[[w for w in WARNELL if w in set(warn_prof.auth)]].reset_index()

# standardized pay (pay / college five-year mean of non-administrative list scientists)
try:
    fac = pd.read_excel(XL, sheet_name='Std factors'); F = dict(zip(fac.college, fac.mean5))
    def col_of(school):

```

```

        s = str(school)
        for k in ['Franklin', 'CAES', 'Education', 'Engineering', 'Odum', 'Pharmacy', 'Public Health', 'SPIA',
'Terry', 'Vet Med', 'Warnell']:
            if s.startswith(k): return k
        return None
    for df in (admin_prof,):
        df['college_std'] = df.school.map(col_of);
        for y in [2021, 2022, 2023, 2024, 2025]:
            df[f'pay_std_FY{y}'] = df[f'pay_FY{y}'] / df.college_std.map(F)
        for y in [2021, 2022, 2023, 2024, 2025]:
            warn_prof[f'pay_std_FY{y}'] = warn_prof[f'pay_FY{y}'] / F['Warnell']
except Exception as e:
    print('std pay not added:', e)

# comparison groups: admin vs non-admin (matched, full-time, non-admin) vs all 321
non_admin = prof[~prof.auth.isin(admin_prof.auth) & prof.auth.isin(t.auth)]
def med(df): return df[['np', 'nc', 'h', 'hm', 'self%', 'single_pap_share', 'single_cit_share',
'first_pap_share', 'last_pap_share', 'last_cit_share', 'other_pap_share', 'cpp_single', 'cpp_first', 'cpp_last',
'cpp_all', 'subfield_pct']].median()
comp = pd.DataFrame({'admin title holders (n=%d)' % len(admin_prof): med(admin_prof), 'other matched scientists
(n=%d)' % len(non_admin): med(non_admin), 'all 321': med(prof), 'Warnell 7': med(warn_prof)}).T

with pd.ExcelWriter(XL, mode='a', engine='openpyxl', if_sheet_exists='replace') as xw:
    admin_prof.to_excel(xw, sheet_name='Profiles admin holders', index=False)
    warn_prof.to_excel(xw, sheet_name='Profiles Warnell', index=False)
    comp.reset_index().rename(columns={'index': 'group (medians)'}).to_excel(xw, sheet_name='Profiles
comparison', index=False)
pd.set_option('display.width', 260); pd.set_option('display.max_columns', 40); pd.set_option('display.max_rows',
100)
show = ['auth', 'admin titles', 'school', 'sm-subfield-1', 'np', 'nc', 'h', 'single_pap_share',
'single_cit_share', 'first_pap_share', 'last_pap_share', 'last_cit_share', 'cpp_single', 'cpp_last', 'cpp_all',
'subfield_pct', 'pay_FY2025']
print(admin_prof[show].round(2).to_string(index=False))
print()
showw = ['auth', 'warnell_status', 'firstyr', 'np', 'nc', 'h', 'hm', 'self%', 'nps', 'ncs', 'np_first',
'np_last', 'np_other', 'single_pap_share', 'single_cit_share', 'first_pap_share', 'last_pap_share',
'last_cit_share', 'single_vs_last_pap', 'single_vs_last_cit', 'cpp_single', 'cpp_first', 'cpp_last', 'cpp_all',
'rank sm-subfield-1', 'sm-subfield-1 count', 'subfield_pct', 'pay_FY2021', 'pay_FY2025', 'titles_FY21_25']
print(warn_prof[showw].round(2).T.to_string())
print()
print(comp.round(2).T.to_string())

```

22. redo_standardized.py (SHA-256 191649fc99c8c92d...; 256 lines)

"""Standardized-pay rerun of every salary analysis.

Standardization (author's specification): each salary is divided by its college's five-year (FY2021-2025) mean pay of the observed screened NON-administrative Stanford-list scientists and multiplied by the university-wide mean of the same group, so figures stay in dollars, keep the within-college relations, and are rescaled to a common college mean. Colleges default to build/college_checked.csv, checked against public UGA pages; PAY_COLLEGES=heuristic retains the legacy assignments and calculation order. Retirees are carried forward using HIST_RAISE for FY2016-2021 (FY2021 = 0). For FY2022-2025, checked colleges use mean same-person raw-pay percentage changes between consecutive screened base years after partial-exit screening, with the year's mean across colleges as fallback; heuristic mode uses the legacy 'Dean vs Sci by college-year' sheet. Writes sheets: 'Std factors', 'Std panel', 'Imputed retirees', 'Std dean vs sci', 'Std by college', 'Std regression', 'Std Warnell', 'Std admin holders'. Requires pandas, openpyxl, statsmodels.

"""

```

import json, os, re, unicodedata, warnings
import numpy as np, pandas as pd
import statsmodels.formula.api as smf
from pathlib import Path
warnings.filterwarnings('ignore')
ROOT = Path(__file__).resolve().parents[1]
import sys; sys.dont_write_bytecode = True; sys.path.insert(0, str(ROOT / 'build')); from paynames import
auth_key, join_idx # accent-insensitive name key
XL = Path(os.environ.get('PAY_XL', ROOT / 'UGA_StanfordList_Salaries_FY2021-2025.xlsx'))
OUT = Path(os.environ.get('PAY_OUT', ROOT / 'build'))
PAY_FLOOR = float(os.environ.get('PAY_FLOOR', '50000'))
_exit_ratio = os.environ.get('PAY_EXIT_RATIO', '0.8').strip().lower()
PAY_EXIT_RATIO = None if _exit_ratio == 'none' else float(_exit_ratio)
COLLEGE_SOURCE = os.environ.get('PAY_COLLEGES', 'checked').strip().lower()
assert COLLEGE_SOURCE in ('checked', 'heuristic'), 'PAY_COLLEGES must be checked or heuristic'
OUT.mkdir(parents=True, exist_ok=True)
YEARS = [2021, 2022, 2023, 2024, 2025]
ADMIN = r'DEAN|CHAIR|HEAD|DIRECTOR|PROVOST|PRESIDENT|CHIEF'
NONFULL = r'TEMPORARY|RETIREE|FORMER|LIMITED TERM|AFFILIATE|OCCASIONAL|PART TIME'
# median same-person raises of active non-administrative list scientists, from the FY2015-2020 pull

```

```

HIST_RAISE = {2016: 5.5, 2017: 6.99, 2018: -2.17, 2019: 6.72, 2020: 1.49, 2021: 0.0}

long = pd.read_excel(XL, sheet_name='Scientists long')
roster = pd.read_excel(XL, sheet_name='Scientists x FY')[['auth', 'school']]
deans_all = pd.read_excel(XL, sheet_name='Deans & President')
deans = deans_all[deans_all.title.str.startswith('DEAN')].copy()
idx = pd.read_excel(XL, sheet_name='Authorship indices (321)')
ratios = pd.read_excel(XL, sheet_name='Dean vs Sci by college-year') if COLLEGE_SOURCE == 'heuristic' else None
pre = pd.read_csv(ROOT / 'build' / 'prewindow_retirees.csv')
names = pd.read_excel(XL, sheet_name='Not found in OpenGA')[['authfull', 'school']].rename(columns={'authfull': 'auth'})

def college(s):
    s = str(s)
    if s.startswith('Franklin (Communication)'): return None
    if s.startswith('Franklin'): return 'Franklin'
    if s.startswith('CAES'): return 'CAES'
    if s.startswith('Education'): return 'Education'
    if s.startswith('Engineering') or s.startswith('Provost'): return 'Engineering'
    if s == 'Environment & Design': return 'Environment & Design'
    if s.startswith('Odum'): return 'Odum'
    if s.startswith('Pharmacy'): return 'Pharmacy'
    if s.startswith('Public Health'): return 'Public Health'
    if s in ('SPIA', 'Terry', 'Vet Med', 'FACS'): return s
    if s.startswith('Warnell'): return 'Warnell'
    return None

dc = {'SPIA': 'SPIA', 'Terry': 'Terry', 'Public Health': 'Public Health', 'Franklin': 'Franklin', 'Engineering': 'Engineering', 'FACS': 'FACS', 'Odum School of Ecology': 'Odum', 'Warnell': 'Warnell', 'Environment & Design': 'Environment & Design', 'Warnell (interim)': 'Warnell', 'Vet Med': 'Vet Med', 'CAES': 'CAES', 'Pharmacy': 'Pharmacy', 'Education': 'Education', 'CAES (interim)': 'CAES'}
deans.loc[deans.og_name == 'ALTIZER, SONIA M', 'school'] = 'Odum School of Ecology'
if COLLEGE_SOURCE == 'checked':
    dc = {'**dc', 'Grady': 'Grady', 'Law': 'Law'} # the checked colleges include Grady; every college with a listed scientist gets its dean
deans['college'] = deans.school.map(dc)

py = long.drop(columns=[c for c in long.columns if c == 'school']).merge(roster, on='auth', how='left')
ROSTER = json.load(open(ROOT / 'build' / 'roster_overrides.json', encoding='utf-8')) # name-keyed overrides live in a data file, not in the listing
for _a, _s in ROSTER['affiliation_overrides'].items(): py.loc[py.auth == _a, 'school'] = _s
py['college'] = py.school.map(college)
py = py.groupby(['auth', 'fy']).agg(pay=('salary', 'sum'), titles=('title', lambda s: ' + '.join(sorted(set(s)))), college=('college', 'first'),
                                **({'school': ('school', 'first')} if COLLEGE_SOURCE == 'checked' else {})).reset_index()
pre = pre.merge(names, on='auth', how='left')
for _a, _s in ROSTER['prewindow_school_overrides'].items(): pre.loc[pre.auth == _a, 'school'] = _s
pre['college'] = pre.school.map(college)

if COLLEGE_SOURCE == 'checked':
    checked = pd.read_csv(ROOT / 'build' / 'college_checked.csv', keep_default_na=False, encoding='utf-8-sig')
    checked['_auth'] = checked.auth.map(auth_key)
    for col in ('college', 'status', 'heuristic_college'): checked[col] = checked[col].str.strip()
    assert checked._auth.ne('').all(), 'Empty normalized auth in college_checked.csv'
    conflicts = checked.groupby('_auth').college.nunique()
    assert not (conflicts > 1).any(), 'Conflicting checked colleges for: ' + ', '.join(checked.loc[checked._auth.isin(conflicts[conflicts > 1].index), 'auth'])
    checked = checked.drop_duplicates('_auth').set_index('_auth')
    assert checked.status.isin(['confirmed', 'probable', 'unresolved']).all(), 'Invalid status in college_checked.csv'
    required = pd.concat([roster.auth, py.auth, pre.auth]).drop_duplicates()
    missing = sorted(a for a in required if auth_key(a) not in checked.index)
    assert not missing, 'Missing persons in college_checked.csv: ' + '; '.join(missing)
    college_check = {'persons_in_csv': int(len(checked)),
                    **{s: int(checked.status.eq(s).sum()) for s in ('confirmed', 'probable', 'unresolved')},
                    'changed_from_heuristic': int(checked.college.ne(checked.heuristic_college).sum()),
                    'no_college': int(checked.college.eq('').sum())}
    for frame in (py, pre):
        frame['college_heuristic'] = frame.college
        frame['college'] = frame.auth.map(auth_key).map(checked.college).replace({'': None})

py['is_admin'] = py.titles.str.contains(ADMIN)
py['nonfull'] = py.titles.str.contains(NONFULL) | (py.pay < PAY_FLOOR)
py['base'] = (~py.is_admin) & (~py.nonfull) & py.college.notna()
py['imputed'] = False

# In checked mode, remove every partial exit year before factors and raises are computed.
# The heuristic path keeps the original ordering, including its later conditional removal.

```

```

if COLLEGE_SOURCE == 'checked' and PAY_EXIT_RATIO is not None:
    for auth, g in py.groupby('auth'):
        b = g[g.base].sort_values('fy')
        while len(b) >= 2 and b.iloc[-1].pay < PAY_EXIT_RATIO * b.iloc[-2].pay:
            py.loc[b.index[-1], 'base'] = False
            b = b.iloc[:-1]

# --- 1. factors: college five-year mean and university-wide mean of non-administrative full-time list scientists
---
factors = py[py.base].groupby('college').agg(mean5=('pay', 'mean'), median5=('pay', 'median'),
n_person_years=('pay', 'size'), n_persons=('auth', 'nunique')).reset_index()
U = float(py[py.base].pay.mean()); U_med = float(py[py.base].pay.median())
factors['multiplier_U_over_mean5'] = U / factors.mean5
F = dict(zip(factors.college, factors.mean5))
def std(pay, col): return pay / F[col] * U if col in F and pd.notna(pay) else np.nan

# --- 2. raises for imputation ---
if COLLEGE_SOURCE == 'checked':
    changes = py[py.base].sort_values(['auth', 'fy']).rename(columns={'pay': 'pay_raw'}).copy()
    changes['prev_pay'] = changes.groupby('auth').pay_raw.shift()
    changes['prev_fy'] = changes.groupby('auth').fy.shift()
    changes = changes[(changes.fy == changes.prev_fy + 1) & changes.fy.between(2022, 2025)].copy()
    changes['sci_inc_mean_pct'] = 100 * (changes.pay_raw / changes.prev_pay - 1)
    inc = changes.groupby(['college', 'fy']).sci_inc_mean_pct.mean().reset_index()
    raise_source = 'HIST_RAISE FY2016-2021; checked-college mean consecutive screened-base raw-pay changes
FY2022-2025; fallback: year mean across colleges'
else:
    inc = ratios[['college', 'fy', 'sci_inc_mean_pct']].copy()
    raise_source = 'HIST_RAISE FY2016-2021; Dean vs Sci by college-year sci_inc_mean_pct FY2022-2025 (legacy
heuristic colleges)'
inc_all = inc.groupby('fy').sci_inc_mean_pct.mean().to_dict()
def raise_pct(c, y):
    if y <= 2021: return HIST_RAISE[y]
    v = inc[(inc.college == c) & (inc.fy == y)].sci_inc_mean_pct
    v = float(v.iloc[0]) if len(v) and pd.notna(v.iloc[0]) else np.nan
    return v if pd.notna(v) else float(inc_all.get(y, 0.0))

imputed = []
# (a) in-window retirees: last base year < 2025 and no later base year, no later admin year
for auth, g in py.groupby('auth'):
    b = g[g.base].sort_values('fy')
    if not len(b): continue
    last = b.iloc[-1]; later = g[g.fy > last.fy]
    # step back over partial exit years (already screened above in checked mode)
    dropped = []
    while PAY_EXIT_RATIO is not None and len(b) >= 2 and last.pay < PAY_EXIT_RATIO * b.iloc[-2].pay:
        dropped.append(int(last.fy)); b = b.iloc[:-1]; last = b.iloc[-1]; later = g[g.fy > last.fy]
    later_base = later[later.base & ~later.fy.isin(dropped)]
    if last.fy < 2025 and not len(later_base) and not later.is_admin.any():
        py.loc[(py.auth == auth) & py.fy.isin(dropped), 'base'] = False
        pay = float(last.pay); c = last.college
        for y in range(int(last.fy) + 1, 2026):
            pay *= 1 + raise_pct(c, y) / 100
            imputed.append({'auth': auth, 'fy': y, 'pay': pay, 'titles': 'IMPUTED (in-window; last full-time
FY%d)' % last.fy, 'college': c, 'is admin': False, 'nonfull': False, 'base': False, 'imputed': True,
'last_ft_fy': int(last.fy), 'last_ft_pay': float(last.pay), 'kind': 'in-window'})
# (b) pre-window retirees from the FY2015-2020 pull
for _, r in pre.iterrows():
    if pd.isna(r.college): continue
    pay = float(r.last_ft_pay)
    for y in range(int(r.last_ft_fy) + 1, 2026):
        pay *= 1 + raise_pct(r.college, y) / 100
        if y >= 2021:
            imputed.append({'auth': r.auth, 'fy': y, 'pay': pay, 'titles': 'IMPUTED (pre-window; last full-time
FY%d)' % r.last_ft_fy, 'college': r.college, 'is admin': False, 'nonfull': False, 'base': False, 'imputed': True,
'last_ft_fy': int(r.last_ft_fy), 'last_ft_pay': float(r.last_ft_pay), 'kind': 'pre-window'})
imp = pd.DataFrame(imputed)
if len(imp):
    if COLLEGE_SOURCE == 'checked':
        provenance = pd.concat([py[['auth', 'school', 'college_heuristic']], pre[['auth', 'school',
'college_heuristic']]]).drop_duplicates('auth').set_index('auth')
        for col in ('school', 'college_heuristic'): imp[col] = imp.auth.map(provenance[col])
        key = set(zip(imp.auth, imp.fy)); py.loc[(a, y) in key for a, y in zip(py.auth, py.fy)], 'base'] = False
        panel = pd.concat([py, imp[py.columns.tolist()], ignore_index=True) if len(imp) else py.copy()
        panel['pay_std'] = [std(p, c) for p, c in zip(panel.pay, panel.college)]
        panel['use'] = panel.base | panel.imputed
        deans['pay_std'] = [std(p, c) for p, c in zip(deans.salary, deans.college)]

# --- 3. standardized dean vs scientists per college-year, active and active+imputed ---

```

```

rows = []
for (c, y), g in panel[panel.use].groupby(['college', 'fy']):
    d = deans[(deans.college == c) & (deans.fy == y)]; gb = g[g.base]
    dean_std = d.pay_std.sum() if len(d) else np.nan
    rows.append({'college': c, 'fy': y, 'n_active': len(gb), 'n_incl_imputed': len(g),
                'sci_std_mean_active': gb.pay_std.mean(), 'sci_std_median_active': gb.pay_std.median(),
                'sci_std_mean_incl_imputed': g.pay_std.mean(),
                'dean_std': dean_std, 'dean_names': ' / '.join(d.og_name) if len(d) else '',
                'ratio_dean_to_sci_active': dean_std / gb.pay_std.mean() if len(d) and len(gb) else np.nan,
                'ratio_dean_to_sci_incl_imputed': dean_std / g.pay_std.mean() if len(d) and len(g) else np.nan,
                'flag': 'dean seat partial/transition;' if (len(d) > 1 or (len(d) and d.salary.sum() < 150000))
                else ''})
std_ratios = pd.DataFrame(rows).sort_values(['college', 'fy'])
# same-person raises: deans vs active scientists, standardized (multiplicative constants cancel within college)
dr = deans.sort_values(['og_name', 'fy']); dr['prev'] = dr.groupby('og_name').pay_std.shift(); dr['dean_inc_pct']
= 100 * (dr.pay_std / dr.prev - 1)
dr = dr[(dr.salary > 150000) & (dr.prev > 150000 * U / dr.college.map(F).fillna(U))]
sr = panel[panel.base].sort_values(['auth', 'fy']); sr['prev'] = sr.groupby('auth').pay_std.shift(); sr['inc'] =
100 * (sr.pay_std / sr.prev - 1)
inc_tab = sr.dropna(subset=['inc']).groupby(['college', 'fy']).agg(sci_inc_mean_pct=('inc', 'mean'),
sci_inc_median_pct=('inc', 'median'), n_inc=('inc', 'size')).reset_index()
inc_tab = inc_tab.merge(dr.groupby(['college', 'fy']).dean_inc_pct.mean().reset_index(), on=['college', 'fy'],
how='left')
inc_tab['diff_dean_minus_sci_pp'] = inc_tab.dean_inc_pct - inc_tab.sci_inc_mean_pct
std_ratios = std_ratios.merge(inc_tab, on=['college', 'fy'], how='left')

# --- 4. by-college summary, raw vs standardized (five-year pooled, active) ---
bc = panel[panel.base].groupby('college').agg(n_persons=('auth', 'nunique'), raw_mean=('pay', 'mean'),
raw_median=('pay', 'median'), std_mean=('pay_std', 'mean'), std_median=('pay_std', 'median')).reset_index()
bci = panel[panel.use].groupby('college').agg(n_persons_incl_imputed=('auth', 'nunique'),
std_mean_incl_imputed=('pay_std', 'mean'), std_median_incl_imputed=('pay_std', 'median')).reset_index()
bd = deans.groupby('college').agg(dean_raw_mean=('salary', 'mean'), dean_std_mean=('pay_std',
'mean')).reset_index()
by_college = bc.merge(bci, on='college', how='left').merge(bd, on='college', how='left')
by_college['dean_std_over_sci_std_mean'] = by_college.dean_std_mean / by_college.std_mean

# --- 5. regression on log standardized pay (active; active + imputed) ---
d = join_idx(panel[panel.use], idx)
d['lstd'] = np.log(d.pay_std); d['lnp'] = np.log(d.np); d['lcpp'] = np.log(d.cpp_all); d['yr'] = d.fy.astype(str)
d['lage'] = np.log((2025 - d.firstyr).clip(lower=1))
d['single_pap'] = d.I_pap_single_vs_all; d['single_cit'] = d.I_cit_single_vs_all; d['last_pap'] =
d.share_pap_last; d['last_cit'] = d.share_cit_last
d['lcpp_single'] = np.log(d.cpp_single.replace(0, np.nan)); d['lcpp_last'] = np.log(d.cpp_last.replace(0,
np.nan))
fE = 'lstd ~ lnp + lcpp + lage + single_pap + single_cit + last_pap + last_cit + C(yr)'
fF = 'lstd ~ lnp + lage + lcpp_single + lcpp_last + C(yr)'
out = []
def rec(name, m, n):
    for k in m.params.index:
        if k.startswith('C(yr)') or k == 'Intercept' or 'Var' in k: continue
        out.append({'model': name, 'n_obs': n, 'term': k, 'coef': round(float(m.params[k]), 3), 'se':
round(float(m.bse[k]), 3), 'p': round(float(m.pvalues[k]), 3)})
    for label, dd in [(('active only', d[~d.imputed]), ('active + imputed retirees', d))]:
        for fname, f in [(('E all shares', fE), ('F cites/paper by position', fF))]:
            ddf = dd.dropna(subset=['lstd', 'lnp', 'lcpp', 'lage', 'single_pap', 'single_cit', 'last_pap',
'last_cit'] + [t for t in ['lcpp_single', 'lcpp_last'] if t in f]).reset_index(drop=True)
            m1 = smf.ols(f, ddf).fit(cov_type='cluster', cov_kws={'groups': pd.factorize(ddf.auth)[0]});
            rec(f'{fname} | {label} | OLS, person-clustered SE', m1, len(ddf))
            m2 = smf.mixedlm(f, ddf, groups=ddf['auth']).fit(reml=True); rec(f'{fname} | {label} | person random
intercept', m2, len(ddf))
reg = pd.DataFrame(out)
p24 = d[(d.fy == 2024) & (~d.imputed)]
corr = p24[['lstd', 'lnp', 'lcpp', 'single_pap', 'single_cit', 'last_pap', 'last_cit', 'lcpp_single',
'lcpp_last', 'lage']].corr(method='spearman')['lstd'].round(3)

# --- 6. standardized profiles: Warnell and administrative holders ---
def pay_wide(frame, col):
    w = frame.pivot_table(index='auth', columns='fy', values=col, aggfunc='sum'); w.columns = [f'{col}_FY{c}' for
c in w.columns]; return w
allp = panel.copy()
WARNELL = ROSTER['warnell']
wp = allp[allp.auth.isin(WARNELL)]
warn = pay_wide(wp, 'pay').join(pay_wide(wp, 'pay_std')).join(wp.groupby('auth').titles.agg(lambda s: ' |
'.join(sorted(set(s)))).rename('titles')).reindex(WARNELL).reset_index()
adm_names = py[py.is_admin].auth.unique()
ap = allp[allp.auth.isin(adm_names)]
admin = pay_wide(ap, 'pay').join(pay_wide(ap, 'pay_std')).join(ap.groupby('auth').agg(college=('college',
'first'), titles=('titles', lambda s: ' | '.join(sorted(set(s)))))').reset_index()

```

```

with pd.ExcelWriter(XL, mode='a', engine='openpyxl', if_sheet_exists='replace') as xw:
    fac = pd.concat([factors, pd.DataFrame({'college': 'UNIVERSITY (all colleges, pooled)', 'mean5': U,
'median5': U_med, 'n_person_years': int(py.base.sum()), 'n_persons': int(py[py.base].auth.nunique()),
'multiplier_U_over_mean5': 1.0})]), ignore_index=True)
    fac.to_excel(xw, sheet_name='Std factors', index=False)
    panel.sort_values(['college', 'auth', 'fy']).to_excel(xw, sheet_name='Std panel', index=False)
    (imp if len(imp) else pd.DataFrame()).to_excel(xw, sheet_name='Imputed retirees', index=False)
    std_ratios.to_excel(xw, sheet_name='Std dean vs sci', index=False)
    by_college.to_excel(xw, sheet_name='Std by college', index=False)
    blank = pd.DataFrame([''] * len(reg.columns)), columns=reg.columns)
    c = corr.reset_index().rename(columns={'index': 'model', 'lstd': 'n_obs'}).reindex(columns=reg.columns,
fill_value='')
    hdr = pd.DataFrame(['Spearman corr with log standardized pay, FY2024 active (col2)'] + [''] *
(len(reg.columns) - 1)], columns=reg.columns)
    pd.concat([reg, blank, hdr, c], ignore_index=True).to_excel(xw, sheet_name='Std regression', index=False)
    warn.to_excel(xw, sheet_name='Std Warnell', index=False)
    admin.to_excel(xw, sheet_name='Std admin holders', index=False)
json.dump({'factors': F, 'U_mean': U, 'U_median': U_med, 'n_imputed_persons': int(imp.auth.nunique()) if len(imp)
else 0, 'n_imputed_rows': int(len(imp)), 'hist_raises': HIST_RAISE,
'excluded_prewindow': ROSTER['excluded_prewindow'], 'college_source': COLLEGE_SOURCE, 'raise_source':
raise_source,
**({'college_check': college_check} if COLLEGE_SOURCE == 'checked' else {})),
open(OUT / 'std_summary.json', 'w'), indent=1)
pd.set_option('display.width', 250); pd.set_option('display.max_rows', 200)
print('U (university mean, non-admin FT list scientists, FY2021-25) = %.0f ; median %.0f' % (U, U_med))
print(factors.round(2).to_string(index=False)); print()
print('imputed persons:', imp.auth.nunique() if len(imp) else 0, '(in-window %d, pre-window %d)' % (imp[imp.kind
== 'in-window'].auth.nunique(), imp[imp.kind == 'pre-window'].auth.nunique()))
print(by_college.round(0).to_string(index=False)); print()
print(std_ratios[['college', 'fy', 'n_active', 'n_incl_imputed', 'sci_std_mean_active',
'sci_std_mean_incl_imputed', 'dean_std', 'ratio_dean_to_sci_active', 'ratio_dean_to_sci_incl_imputed',
'sci_inc_median_pct', 'dean_inc_pct', 'flag']].round(2).to_string(index=False)); print()
print(reg.to_string(index=False)); print(); print(corr.to_string()); print()
print(warn[['auth']] + [c for c in warn.columns if c.startswith('pay_std')]).round(0).to_string(index=False))

```

23. render_docx.py (SHA-256 8a028714239ebb49...; 38 lines)

""Windows rendering: user-specified LibreOffice plus PyMuPDF page PNGs.

The packaged skill renderer requires unavailable pdf2image/Poppler. This local adapter follows the same render-and-inspect process with all files inside cwd.

```

"""
import sys,os,subprocess,json
from pathlib import Path
from pdfclean import strip_pdf_dates
ROOT=Path(__file__).resolve().parents[1]
sys.path.insert(0,str(ROOT/'build/vendor'))
import pymupdf as fitz

def main():
    temp=ROOT/'build/tmp';temp.mkdir(exist_ok=True)
    env=os.environ.copy();env.update(TMP=str(temp),TEMP=str(temp),TMPDIR=str(temp))
    profile=ROOT/'build/lo_profile'
    cmd=[r'C:\Program Files\LibreOffice\program\soffice.exe','-env:UserInstallation='+profile.as_uri(), '--
headless', '--norestore', '--convert-to', 'pdf', '--outdir', str(ROOT), str(ROOT/'Cieszewski2026_AI_draft.docx')]
    logs=[]
    for attempt in range(2):
        p=subprocess.run(cmd, cwd=ROOT, env=env, capture_output=True, text=True, timeout=60)
        logs.append({'attempt': attempt+1, 'returncode': p.returncode, 'stdout': p.stdout, 'stderr': p.stderr})
        if p.returncode==0 and (ROOT/'Cieszewski2026_AI_draft.pdf').exists(): break
        (ROOT/'build/render_log.json').write_text(json.dumps(logs, indent=2))
        strip_pdf_dates(ROOT/'Cieszewski2026_AI_draft.pdf')
        pdf=fitz.open(ROOT/'Cieszewski2026_AI_draft.pdf'); qa=ROOT/'build/qa'; qa.mkdir(exist_ok=True)
        pages=[]
        # Clear only stale PNGs produced by this renderer inside its own QA folder.
        for old in qa.glob('page-*.png'):
            if old.resolve().parent != qa.resolve():
                raise RuntimeError('QA output path escaped the workspace')
            old.unlink()
        for i, page in enumerate(pdf, 1):
            page.get_pixmap(matrix=fitz.Matrix(1.4, 1.4)).save(qa/f'page-{i:02d}.png')
            pages.append({'page': i, 'words': len(page.get_text().split()), 'images': len(page.get_image_info()), 'text': pa
ge.get_text()})
        (ROOT/'build/pdf_pages.json').write_text(json.dumps(pages, indent=2), encoding='utf-8')
        print('Rendered', len(pdf), 'pages; image counts:', [(x['page'], x['images']) for x in pages if x['images']])

if __name__ == '__main__': main()

```

24. revision_checks.py (SHA-256 4cbc8b90f740756c...; 72 lines)

```
"""Checks added for the revision (reviewer requests). Writes build/revision_checks.json."""
import json,statistics,math
from collections import Counter
from data_core import read_extract,n,ROOT
raw=read_extract()
f=[r for r in raw if str(r['sm-subfield-1']).strip()=='Forestry'];e=[r for r in raw if str(r['sm-subfield-1']).strip()=='Ecology' and str(r['sm-subfield-2']).strip()=='Forestry'];d=f+e
d24=[r for r in d if r['year']==2024]
def rnk(vals):return [1+sum(x>v for x in vals)+(sum(x==v for x in vals)-1)/2 for v in vals]
def rho(a,b):return statistics.correlation(rnk(a),rnk(b))
out={}
# 1 duplicate names
dup={}
for name in ['Keane, Robert E.','Schmidt, Olaf']:
    dup[name]=[k:r[k] for k in ['inst_name','cntry','sm-subfield-1','sm-subfield-2','np','nc','firstyr','h']]
for r in d24 if r['authfull']==name]
out['duplicates']=dup
# sensitivity: drop the four duplicate rows
def shares(frame):
    pt=sum(n(r,'np') for r in frame);ct=sum(n(r,'nc') for r in frame)
    s=sum(n(r,'nps') for r in frame);fs=sum(n(r,'cpsf')-n(r,'nps') for r in frame);o=sum(n(r,'np')-n(r,'cpsf')
for r in frame)
    sc=sum(n(r,'ncs') for r in frame);fc=sum(n(r,'ncsf')-n(r,'ncs') for r in frame);oc=sum(n(r,'nc')-n(r,'ncsf')
for r in frame)
    return {'n':len(frame),'paper_share':[100*s/pt,100*fs/pt,100*o/pt],'cit_share':[100*sc/ct,100*fc/ct,100*oc/
ct],'cpp':[sc/s,fc/fs,oc/o],'all_cpp':ct/pt}
out['position_all']=shares(d)
neg=[r for r in d if n(r,'np')<n(r,'cpsf')]
out['negative_rows']=[k:r[k] for k in ['year','authfull','np','cpsf','nps','nc','ncsf','ncs']] for r in neg]
out['position_drop_negative']=shares([r for r in d if not (n(r,'np')<n(r,'cpsf'))])
# 2 rank analysis details
allv=[n(r,'nc') for r in d24];sv=[n(r,'ncs') for r in d24];lv=[n(r,'ncsf') for r in d24]
out['rank']={'n':len(d24),'zero_nps':sum(n(r,'nps')==0 for r in d24),'zero_ncs':sum(n(r,'ncs')==0 for r in
d24),'zero_cpsf':sum(n(r,'cpsf')==0 for r in d24),
    'distinct_ncs_values':len(set(sv)),'tied_ncs_records':sum(1 for v in sv if
sv.count(v)>1),'distinct_nc_values':len(set(allv)),'tied_nc_records':sum(1 for v in allv if allv.count(v)>1),
    'rho_single':rho(allv,sv),'rho_lead':rho(allv,lv),
    'rho_single_nonzero':rho([a for a,s in zip(allv,sv) if s>0],[s for s in sv if s>0]),'n_nonzero':sum(1 for s in
sv if s>0),
    'rho_single_ns':rho([n(r,'nc (ns)') for r in d24],[n(r,'ncs (ns)') for r in d24]),'rho_lead_ns':rho([n(r,'nc
(ns)') for r in d24],[n(r,'ncsf (ns)') for r in d24])}
for g,frame in ([('Forestry',[r for r in d24 if r in f]),('Ecology',[r for r in d24 if r in e])):
    a=[n(r,'nc') for r in frame];s=[n(r,'ncs') for r in frame];l=[n(r,'ncsf') for r in frame]
    out['rank'][g]={'n':len(frame),'rho_single':rho(a,s),'rho_lead':rho(a,l)}
# excluding duplicate-name rows
keep=[r for r in d24 if r['authfull'] not in dup]
out['rank']['excl_duplicates']={'n':len(keep),'rho_single':rho([n(r,'nc') for r in keep],[n(r,'ncs') for r in
keep]),'rho_lead':rho([n(r,'nc') for r in keep],[n(r,'ncsf') for r in keep])}
ar=rnk(allv);sr=rnk(sv);diff=[s-a for a,s in zip(ar,sr)]
out['rank']['rank_change_single_minus_total']={'median':statistics.median(diff),'q10':sorted(diff)
[int(.1*len(diff))],'q90':sorted(diff)[int(.9*len(diff))],'abs_median':statistics.median(abs(x) for x in
diff),'share_abs_gt_200':sum(abs(x)>200 for x in diff)/len(diff)}
json.dump({'ranks_total':ar,'ranks_single':sr,'ranks_lead':rnk(lv)},open(ROOT/'build/rank_vectors.json','w'))
# 3 RI small-count sensitivity: RI for 2024 vs pooled from normalization.json
norm=json.loads((ROOT/'build/normalization.json').read_text())
ri24=[r['country']:r for r in norm['2024']]['records'];rip=[r['country']:r for r in norm['pooled']]['records']]
out['ri_compare']={c:{'f24':ri24[c]['forestry'],'ri24':ri24[c]['ri'],'ri_pooled':rip[c]['ri'] if c in rip else
None} for c in ['fin','swe','svk','svn','usa','chn','gbr','can','aus','nzl','dnk']}

# window compatibility of the authorship fields (Key sheet of the release: np6024 = papers 1960-2024;
nps/cpsf/npsfl carry no stated window)
pre=[r for r in d if n(r,'firstyr')<1960]
out['window']={'records_first_before_1960':len(pre),'profiles_first_before_1960':len(set(r['authfull'] for r in
pre)),
    'records_npsfl_gt_np':sum(n(r,'npsfl')>n(r,'np') for r in
d),'records_npsfl_gt_np_pre1960':sum(n(r,'npsfl')>n(r,'np') for r in pre),
    'records_cpsf_gt_np':sum(n(r,'cpsf')>n(r,'np') for r in
d),'records_cpsf_gt_np_pre1960':sum(n(r,'cpsf')>n(r,'np') for r in pre),
    'adler':[{k:r[k] for k in ['year','firstyr','lastyr','np','nps','cpsf','npsfl','nc','ncs','ncsf','ncsfl']} for r
in d if r['authfull']=='Adler, Erich'],
    'position_excl_pre1960':shares([r for r in d if n(r,'firstyr')>=1960])}
# annual representation index by country and data year (sensitivity to the aggregation period)
import csv
from data_core import SRC
with open(SRC/'data/world_subfield_year.csv',encoding='utf-8-sig',newline='') as fp:w=list(csv.DictReader(fp))
ann={}
for y in range(2020,2025):
    ww=[r for r in w if int(r['year'])==y];A=sum(int(r['n']) for r in ww);F=sum(int(r['n']) for r in ww if
r['sm1']=='Forestry')
```

```

cc={}
for r in ww:
    c=r['cntry'];cc.setdefault(c,[0,0]);cc[c][1]+=int(r['n'])
    if r['sm1']=='Forestry':cc[c][0]+=int(r['n'])
for c,(fc,ac) in cc.items():
    if fc:ann.setdefault(c,{})[y]={ 'F':fc, 'A':ac, 'RI':(fc/F)/(ac/A) }
rows=[{ 'country':c, 'year':y, 'forestry':v['F'], 'all':v['A'], 'ri':round(v['RI'],4) } for c in sorted(ann) for y,v in
sorted(ann[c].items())]
with open(ROOT/'build/annual_ri.csv','w',newline='',encoding='utf-8') as fp:
    wr=csv.DictWriter(fp,fieldnames=['country','year','forestry','all','ri']);wr.writeheader();wr.writerows(rows)
out['annual_ri_range']={c:{ 'min':round(min(v['RI'] for v in ann[c].values()),2), 'max':round(max(v['RI'] for v in
ann[c].values()),2), 'years':len(ann[c]), 'min_F':min(v['F'] for v in ann[c].values()) } for c in
['fin','swe','svk','svn','usa','chn','gbr','can','aus','nzl','dnk','deu','cze','zaf','bra','prt','tur','pol']}
json.dump(out,open(ROOT/'build/revision_checks.json','w'),indent=1,default=str)
print(json.dumps({k:v for k,v in out.items() if k!
='negative_rows'},indent=1,default=str));print(out['negative_rows'])

```

25. verify_numbers.py (SHA-256 990e3b29e1c46ea7...; 299 lines)

"""Independent recomputation plus source/artifact/TeX/DOCX integrity checks.

Run from any directory. Writes verify_output.txt even if an assertion fails.
Uses only the standard library for all numerical checks and DOCX extraction.
PyMuPDF is used for the PDF checks when the PDF exists.

"""

```

from pathlib import Path
import sys,json,csv,re,math,statistics,hashlib,zipfile,traceback,posixpath
import xml.etree.ElementTree as ET
from collections import Counter
from data_core import ROOT,SRC,EXTRACT,read_extract,n,src_file
B=chr(92)
from manuscript_content import load_content,plain,latex,make_tables,COUNTRIES

```

```

LOG=[]
def ok(msg):LOG.append('PASS '+msg);print(LOG[-1])
def near(a,b):assert math.isclose(float(a),float(b),rel_tol=1e-10,abs_tol=1e-9),(a,b)
def normtext(t):return re.sub(r'\s+', ' ',t).strip()
def rnk(vals):return [1+sum(x>v for x in vals)+(sum(x==v for x in vals)-1)/2 for v in vals]

```

```

def verify_exhibit_b_numbers(main):
    u=json.loads((ROOT/'build/ugapay.json').read_text())
    pf=json.loads((ROOT/'build/pay_figures.json').read_text())
    sensitivity=json.loads((ROOT/'build/pay_sensitivity.json').read_text())
    pc=u['panel_counts'];base=sensitivity['base']
    assert base['panel_counts']==pc and base['lnp_range']==u['lnp_range']
    assert (base['ratio_min'],base['ratio_max'])==(u['ratio_min'],u['ratio_max'])
    assert base['n_observed']==pf['n_observed']==pc['persons_active']
    for key,value in base['rho'].items():assert value==pf['rho'][key]['observed'],key
    n_matched=pc['panel_persons_total']-(pc['persons_imputed']-pc['persons_both'])
    section=lambda start,end:main.split(start,1)[1].split(end,1)[0]
    sections={3:section('## 3 Exhibit B:', '## 4 '),4:section('## 4 ', '## 5 '),
              6:section('## 6 Conclusions', '## Acknowledgments')}
    triples=[]
    for keys,label in [ (('np','np_last','nps'),'papers'), (('nc','nc_last','ncs'),'citations'),
                       (('cpp_all','cpp_last','cpp_single'),'citations per paper')]:
        values=[f"{pf['rho'][k]['observed']['rho']:.2f}" for k in keys]
        triples.append(f"{values[0]}, {values[1]}, and {values[2]} for {label}")
    required={3:triples+[
        f"{n_matched} matched the records by name, and {pc['persons_active']} persons had usable observed pay",
        f"elasticity of {u['lnp_range'][0]:.2f} to {u['lnp_range'][1]:.2f}",
        f"last-author share of papers correlates with pay at {pf['rho_shares']['last_pap_share']['observed']
['rho']:.2f}",
        f"{base['ratio_min']:.1f} to {base['ratio_max']:.1f} times",
        4:[f"up to {base['ratio_max']:.1f} times",6:[f"up to {base['ratio_max']:.1f} times"]]}
    for number,strings in required.items():
        for text in strings:assert text in sections[number],(number,text)
    pay=(ROOT/'build/appendix_a.tex').read_text(encoding='ascii')
    for stale in ('0.20 to 0.26','+0.25, p = 0.003','n = 147','1.1 to 2.4','up to 2.4 times','Author salaries'):
        assert stale not in pay,stale
    ok('Exhibit B numbers match the recomputed JSON')

```

```

def verify():
    raw=read_extract();f=[r for r in raw if str(r['sm-subfield-1']).strip()=='Forestry'];e=[r for r in raw if
str(r['sm-subfield-1']).strip()=='Ecology' and str(r['sm-subfield-2']).strip()=='Forestry'];d=f+e
    assert [len(raw),len(f),len(e),len(d)]==[23374,3271,615,3886]
    d24=[r for r in d if r['year']==2024];assert len(d24)==854
    assert len(set(r['authfull'] for r in d24))==852

```

```

duplicates={name for name in set(r['authfull'] for r in d24) if sum(x['authfull']==name for x in d24)>1}
assert duplicates=={'Keane, Robert E.', 'Schmidt, Olaf'}
ok('Cohorts from raw XLSX: 23,374 broad; 3,271 Forestry + 615 Ecology = 3,886; 854 profiles / 852 name
strings in 2024; repeated-name limitation disclosed')
source_a=json.loads((SRC/'data/auth6.json').read_text(encoding='utf-8'))
source_p=json.loads((SRC/'data/authorship.json').read_text(encoding='utf-8'))
source_i=json.loads((SRC/'data/integrity.json').read_text(encoding='utf-8'))
saved=json.loads((ROOT/'build/statistics.json').read_text())
def rates(frame):
    out={'n':len(frame)}
    for lab,pk,ck in [ ('solo','nps','ncs'), ('sen','cpsf','ncsf'), ('all','np','nc') ]:
        p=sum(n(r,pk) for r in frame);c=sum(n(r,ck) for r in frame)
        out.update({lab+'_p':p,lab+'_ppa':p/len(frame),lab+'_cpa':c/len(frame),lab+'_cpp':c/p})
    return out
independent={'groups':{},'years':{}}
for label,frame in [ ('cohort',d), ('Forestry',f), ('Ecology',e) ]:
    v=rates(frame);independent['groups'][label]=v
    target=source_a['cohort'] if label=='cohort' else source_a['group'][label]
    for k,x in v.items():near(x,target[k]);near(x,saved['groups'][label][k])
for j,y in enumerate(range(2020,2025)):
    v=rates([r for r in d if r['year']==y]);independent['years'][str(y)]=v
    for k,x in v.items():near(x,source_a['cohort_by'][k][j]);near(x,saved['years'][str(y)][k])
ok('Every annual count and all 27 population rates reproduce auth6.json and generated statistics.json')
independent['position']={};pt=sum(n(r,'np') for r in d);ct=sum(n(r,'nc') for r in d)
for label,pk,ck,subp,subc in [ ('Single','nps','ncs',None,None), ('First','cpsf','ncsf','nps','ncs'),
('Other','np','nc','cpsf','ncsf') ]:
    p=sum(n(r,pk)-(n(r,subp) if subp else 0) for r in d);c=sum(n(r,ck)-(n(r,subc) if subc else 0) for r in d)
    pp,cp,cpp=100*p/pt,100*c/ct,c/p;key=label.lower()
    independent['position'][label]={ 'p':p, 'c':c, 'paper_share':pp, 'citation_share':cp, 'cpp':cpp }
    near(p,source_p['blocks']['cohort'][key]['p']);near(c,source_p['blocks']['cohort'][key]
['c']);near(cpp,source_p['cpp']['cohort'][key])
    near(pp,source_p['share_p']['cohort'][key]);near(cp,source_p['share_c']['cohort'][key])
    assert [(round(v['paper_share']),round(v['citation_share']),round(v['cpp'],1)) for v in
independent['position'].values())]==[(7,6,37.2),(18,21,49.6),(74,73,43.6)]
assert round(ct/pt,1)==44.2
nonnested=sum(n(r,'np')<n(r,'cpsf') for r in d);assert nonnested==5
ok('Authorship shares / cites per paper: 7/6/37.2; 18/21/49.6; 74/73/43.6; all 44.2; five non-nested paper
records disclosed')
allv=[n(r,'nc') for r in d24];ar=rank(allv);topall=set(sorted(range(len(d24)),key=lambda i:-allv[i])[:25])
ranks={}
for key,col,expected,overlap in [ ('Key','ncsf',.87,15), ('Lead','ncsf',.55,14), ('Single','ncs',-.03,5) ]:
    vals=[n(r,col) for r in d24];rv=rank(vals);corr=statistics.correlation(ar,rv)
    top=set(sorted(range(len(d24)),key=lambda i:-vals[i])[:25]);assert round(corr,2)==expected;assert
len(topall&top)==overlap
    near(corr,saved['ranks'][key]['rho']);assert sorted(vals,reverse=True)[24]!=sorted(vals,reverse=True)[25]
    ranks[key]=(corr,overlap)
    near(ranks['Lead'][0],source_p['corr']['lead: single+first (ncsf)'];near(ranks['Single'][0],source_p['corr']
['single (ncs)'])
ok('Independent pairwise tie ranks: rho +0.87 / +0.55 / -0.03; top-25 overlap 15 / 14 / 5; no boundary ties')
sr=rank([n(r,'ncs') for r in d24])
for name,ra,rs in [ ('Bettinger, Pete',446,638), ('Cieszewski, Chris J.',706.5,234) ]:
    ix=next(i for i,r in enumerate(d24) if r['authfull']==name);assert ar[ix]==ra and sr[ix]==rs,
(name,ar[ix],sr[ix])
assert source_p['cies']['rank_all']==706
ok('Reference reversal independently verified: Bettinger 446 / 638; Cieszewski 706.5 / 234 (all / single).
Source integer truncation to 706 corrected in prose.')
for code in saved['countries']:
    calc=rates([r for r in d if r['cntry']==code])
    for k,v in calc.items():near(v,saved['countries'][code][k])
ok('All country figure and prose rates independently recomputed')
with (SRC/'data/world_subfield_year.csv').open(encoding='utf-8-sig',newline='') as
fp:w=list(csv.DictReader(fp))
norm=json.loads((ROOT/'build/normalization.json').read_text());rn={}
for label,years in [ ('2024',{2024}), ('pooled',set(range(2020,2025))) ]:
    ww=[r for r in w if int(r['year']) in years];total=sum(int(r['n']) for r in ww);ff=sum(int(r['n']) for r
in ww if r['sml']=='Forestry')
    assert ff==sum(r['year'] in years for r in f);near(total,norm[label]['all_total']);near(ff,norm[label]
['forestry_total'])
    counts={}
    for r in ww:
        cc=r['cntry'];counts.setdefault(cc,[0,0]);counts[cc][1]+=int(r['n'])
        if r['sml']=='Forestry':counts[cc][0]+=int(r['n'])
    for cc,(fc,ac) in counts.items():
        assert fc==sum(r['year'] in years and r['cntry']==cc for r in f)
        idx=(fc/ff)/(ac/total);savedrow=next(r for r in norm[label]['records'] if
r['country']==cc);near(idx,savedrow['ri'])
        assert [fc,ac]==[savedrow['forestry'],savedrow['all']]
        near(savedrow['forestry_share'],fc/ff);near(savedrow['all_share'],ac/total)
        if label=='2024':rn[cc]=idx
    assert [norm['2024']['all_total'],norm['2024']['forestry_total'],norm['2024']['unknown_all'],norm['2024']
['unknown_forestry']]==[230333,733,252,0]

```

```

assert {c:round(rn[c],2) for c in ['usa','swe','fin','chn']}=={'usa':.85,'swe':3.35,'fin':11.69,'chn':.28}
ok('Every country RI, both years scopes, and primary-Forestry country counts independently reconciled to raw
inputs')
i={'self_med':100*statistics.median(n(r,'self%') for r in d24),'self_max':100*max(n(r,'self%') for r in
d24),'self_gt25':sum(n(r,'self%')>.25 for r in d24),'cprat_med':statistics.median(n(r,'cprat') for r in
d24),'rw_papers':sum(n(r,'np_rw') for r in d24),'rw_auth':sum(n(r,'np_rw')>0 for r in
d24),'pctcited_med':statistics.median(100*n(r,'np cited')/n(r,'np') for r in
d24),'h_med':statistics.median(n(r,'h') for r in d24),'hm_med':statistics.median(n(r,'hm') for r in d24)}
xs=sorted(allv);nn=len(xs);ss=sum(xs);i.update(gini=sum(abs(x-y) for x in xs for y in
xs)/(2*nn*ss),top10=100*sum(xs[int(.9*nn):])/ss,top1=100*sum(xs[int(.99*nn):])/ss)
independent['integrity']=i
for k,v in i.items():near(v,source_i[k]);near(v,saved['integrity'][k])
ok('Integrity metrics verified, including independent pairwise Gini calculation and author-median cited-paper
proportion')
# Numeric source traces for historical statements, not statistical validation.
before=src_file('v2_report_text.txt').read_text(encoding='utf-8');final=src_file('final_report_text.txt').rea
d_text(encoding='utf-8');log=src_file('session_memory_stanford-list-
forestry.md').read_text(encoding='utf-8');handoff=src_file('HANDOFF_StanfordList_Forestry.md').read_text(encoding
='utf-8')
bf=normtext(before);fl=normtext(final)
assert '33 29 32 73 36 203' in bf and 'Unknown 42 49 49 28 110 278' in bf
assert '21,931' in before and round(21931/len(raw)*100)==94
assert 'about five to one' in bf
for s in ['211','361','4,131','10,619','≈170','≈300']:assert s in final or s in log,s
assert 'Northern Europe' in log and 'WRONG' in log and 'Georgia Tech' in log
ok('Historical claims traced to S1/log/final: affiliation sequences, broad-cohort 94% label, reported 5:1,
Scholar observations, recorded corrections')
weiskittel=Path(r'C:\D\Dropbox\OJS\18(1)\Aaron\Weiskittel2026_AI.tex').read_text(encoding='utf-8')
wt=normtext(weiskittel)
# what this paper cites from the companion paper: the index values, the workforce it is built on, and the
upload advice
for s in ['1.0:1 in 2002','4.5:1 in 2024','9.4:1 by','2032','US forestry researcher workforce','zero-data-
retention']:assert s in wt,s
ok('Weiskittel-specific numerical and policy discussion checked against supplied companion TeX; projections
described as projections')
tex=(ROOT/'Cieszewski2026_AI.tex').read_text(encoding='ascii');texclean=re.sub(r'(?<!\n)%[^\n]*','',tex)
stack=[]
for m in re.finditer(r'\\(begin|end){([{}]+)}\\',texclean):
    kind,env=m.groups()
    if kind=='begin':stack.append(env)
    else:assert stack and stack.pop()==env,env
assert not stack
braces=0
for m in re.finditer(r'(?<!\n){[{}]+}',texclean):
    braces+=1 if m.group()=='{' else -1;assert braces>=0
assert braces==0
labels=set(re.findall(r'\\label{([{}]+)}\\',texclean));used=set(re.findall(r'\\(?:ref|pageref){([{}]+)
\\',texclean));assert used<=labels
cites=set(k.strip() for group in re.findall(r'\\cite(?:p|t|alt|author|yearpar){([{}]+)}\\',texclean) for k
in group.split(','));defined=set(re.findall(r'\\bibitem(?:\\[{}]+\\)?{([{}]+)}\\',texclean));assert
cites==defined,(cites-defined,defined-cites)
manifest=json.loads((ROOT/'build/content_manifest.json').read_text());blocks,refs,lookup=load_content()
shared_figures=[b for b in blocks if b['type']=='figure']
image_paths=lambda text:re.findall(r'\\includegraphics(?:\\[{}]*\\)?{([{}]+)}\\',text)
expected_main=['figures/'+b['file'] for b in shared_figures if not str(b['number'])[0].isalpha()]
expected_appendix=['figures/'+b['file'] for b in shared_figures if str(b['number'])[0].isalpha()]
# Appendix B is generated separately from the shared manuscript blocks.
pay_tex=(ROOT/'build/appendix_a.tex').read_text(encoding='ascii')
expected_appendix+=image_paths(re.sub(r'(?<!\n)%[^\n]*','',pay_tex))
mainclean,appendixclean=texclean.split(B+'appendix',1)
assert not re.search(r'\\begin{equation}\\',mainclean)
main_figures=image_paths(mainclean);figures=image_paths(appendixclean)
assert Counter(main_figures)==Counter(expected_main),(main_figures,expected_main)
assert Counter(figures)==Counter(expected_appendix),(figures,expected_appendix)
for fpath in main_figures+figures:assert (ROOT/fpath).is_file()
assert '\\u2014' not in tex and '---' not in tex
assert len(re.findall(r'\\begin{table}*',mainclean))==1 # the single-column checklist follows its
citation on the same page
assert len(re.findall(r'\\begin{table}\\',appendixclean))==10
assert r'\\label{tab:checklistB}' in mainclean
assert all(r'\\label{tab:'+key+'}' in appendixclean for key in
('cohort','normalization','position','fields','evidence','taxonomy'))
assert r'\\label{eq:ri}' in appendixclean
assert r'\\documentclass[twocolumn,letterpaper,sort]{article}' in tex and r'\\usepackage{mcfnsApr2024v2}' in
tex and r'\\usepackage{balance}' in tex and 'Appendix~' in tex
assert (ROOT/'mcfnsApr2024v2.sty').read_bytes()==Path(r'C:
\\D\\Dropbox\\OJS\\18(1)\\Aaron\\mcfnsApr2024v2.sty').read_bytes()
ok(f'TeX: ASCII only, no em dashes; balanced braces/environments; cited references only, all defined; defined
refs; 1 main-text checklist, 6 forestry and 4 pay appendix tables; {len(main_figures)} main-text and
{len(figures)} appendix figures; Eq. (1) in Appendix A; exact MCFNS style copy')
with zipfile.ZipFile(ROOT/'Cieszewski2026_AI_draft.docx') as z:

```

```

root=ET.fromstring(z.read('word/document.xml'));ns={'w':'http://schemas.openxmlformats.org/
wordprocessingml/2006/main'}
paragraphs=[''.join(p.itertext()) for p in root.findall('.//w:p',ns)]
doc_text=normtext(' '.join(paragraphs))
for x in manifest['expected_text']:assert normtext(x) in doc_text,('DOCX missing',x[:100])
# build_tex.attach_floats sets each float inside the paragraph that first cites it; lift the floats out
again
# so that every paragraph can be matched verbatim
tex_nofloat=re.sub(r'\n\\begin\\{(table|figure)\\(.*?)\\}.*?\\end\\{\\1\\2\\}\\n',' ',tex,flags=re.S)
for b in blocks:
    if b['type'] in ['title','paragraph']:assert latex(b['text']) in tex_nofloat,('paragraph not verbatim
in TeX',b['text'][:80])
    elif b['type']=='table':
        assert latex(b['caption']) in tex and latex(b['note']) in tex
        for row in [b['headers']] + b['rows']:assert ' & '.join(latex(str(x)) for x in row) in tex or
all(latex(str(x)) in tex for x in row)
    elif b['type']=='figure':assert latex(b['caption']) in tex
    for r in refs:
        if r['key'] in cites:assert latex(r['text']) in tex # the TeX lists cited references only
        # Cell-level comparison against independently recomputed table data.
        ts=make_tables(independent,norm);doc_tables=root.findall('.//w:tbl',ns)
        for j,key in [(1,'cohort'),(2,'normalization'),(3,'position')]: # checklist first, then the recomputed
Appendix A tables
        cells=[normtext(''.join(c.itertext())) for c in tr.findall('w:tc',ns)] for tr in
doc_tables[j].findall('w:tr',ns)]
        want=[plain(str(x),refs,lookup) for x in row] for row in [ts[key]['headers']] + ts[key]['rows']]
        assert cells==want, (key,cells,want)
        # python-docx assigns image1.png etc.; resolve actual OOXML relationships.
        rels=ET.fromstring(z.read('word/_rels/document.xml.rels'))
        targets={rel.attrib['Id']:rel.attrib['Target'] for rel in rels}
        blips=root.findall('.//{http://schemas.openxmlformats.org/drawingml/2006/main}blip')
        docx_figures=['figures/'+b['file'] for b in shared_figures]
        assert len(blips)==len(docx_figures), (len(blips),len(docx_figures)) # Appendix B has its own standalone
DOCX
        for fpath,blip in zip(docx_figures,blips):
            rid=blip.attrib['{http://schemas.openxmlformats.org/officeDocument/2006/relationships}embed']
            member=posixpath.normpath(posixpath.join('word',targets[rid]))
            assert z.read(member)==(ROOT/fpath).read_bytes(), (fpath,member)
        ok('All shared content units in DOCX; prose/captions/references in TeX; table cells checked against
recomputation; embedded figure bytes match')
        # Claim-specific prose checks (not just presence of isolated numbers).
        main=(ROOT/'build/manuscript.md').read_text(encoding='utf-8')
        # C4 adds South America while retaining every earlier regional value.
        # C22 qualifies the nominal affiliation selection without changing any count.
        required=['3,271 author-release records whose primary Science-Metrix subfield is Forestry', 'plus the 615
whose primary subfield is Ecology', '3,886 in all', '23,374 records', '854 records under 852 distinct names',
'Spearman correlation of +0.55 with the single-or-first-author citation ranking and -0.03 with the single-author
ranking', 'median absolute change is 247 positions, and 57% of the profiles move', 'has an RI of 0.85', 'Canada
has 95 profiles and an RI of 3.11', 'Finland's 57 give 11.69 and Sweden's 37 give 3.35', 'China has 11 profiles
and 0.28', 'the United Kingdom 19 and 0.29', 'Slovakia ranges from 3.23 to 14.68', '52 such records (15
profiles', 'first publication in 1956 (np = 7, cpsf = 8, and npsfl = 10', '7.0%, 18.4%, and 74.6%', '186,177 to
230,333 profiles', '7%, 18%, and 74% of the papers and 6%, 21%, and 73% of the citations, with 37.2, 49.6, and
43.6 citations per paper against 44.2 overall', 'Northern Europe (39.5) near the bottom, below South America
(63.0), Western Europe (53.7), Northern America (48.0), Australia and New Zealand (43.1), and Central Europe
(40.5)', '-0.25 in the Forestry group and -0.15 in the Ecology group', '45 profiles have no single-authored
citations and 37 have no single-authored papers', '809 profiles with single-authored citations gives +0.06',
'Suppose we want to ask questions about '])
        for s in required:assert s in main,s
        verify_exhibit_b_numbers(main)
        assert '[TODO' not in main and '[TODO' not in (ROOT/'build/references.json').read_text(encoding='utf-8')
        rc=json.loads((ROOT/'build/revision_checks.json').read_text());assert round(rc['rank']
['rho_single'],2)==-0.03 and rc['rank']['zero_ncs']==45 and rc['rank']['zero_nps']==37 and rc['rank']
['distinct_ncs_values']==455 and rc['rank']['distinct_nc_values']==805
        assert round(rc['rank']['rank_change_single_minus_total']['abs_median'])==247 and round(100*rc['rank']
['rank_change_single_minus_total']['share_abs_gt_200'])==57 and rc['rank']['n_nonzero']==809
        assert round(rc['rank']['Forestry']['rho_single'],2)==-0.25 and round(rc['rank']['Ecology']
['rho_single'],2)==-0.15 and abs(rc['rank']['excl_duplicates']['rho_single']-rc['rank']['rho_single'])<0.01
        w=rc['window'];assert w['records_first_before_1960']==52 and w['profiles_first_before_1960']==15 and
w['records_npsfl_gt_np']==20 and w['records_npsfl_gt_np_pre1960']==20 and w['records_cpsf_gt_np_pre1960']==5
        assert (ROOT/'build/annual_ri.csv').exists() and round(rc['annual_ri_range']['svk']['min'],2)==3.23 and
round(rc['annual_ri_range']['fin']['max'],2)==13.44
        assert all(r['authfull']=='Adler, Erich' and r['np']==7 and r['cpsf']==8 for r in rc['negative_rows']) and
len(rc['negative_rows'])==5
        ns=json.loads((ROOT/'build/nash_search.json').read_text());assert [v['rows'] for v in
ns.values()]==[186177,194983,204643,217097,230333] and all(17<=len(v['nash'])<=22 for v in ns.values())
        assert not any('Princeton' in str(h['inst']) or h['subfield'] in ('Economics','Applied
Mathematics','Mathematics') for v in ns.values() for h in v['nash'])
        ok('Key numerical prose checked in context; no TODO placeholders; revision checks (ties, zero counts, group
correlations, duplicate sensitivity, Adler residual, full-file Nash search) reconciled')
        mainbody=main.split('## 1 Introduction')[1].split('## Acknowledgments')[0];mainbody=re.sub(r'\{\\(.*?)\\}\\{\\(
[.*?\\]\\}',',',mainbody)

```

```

wc=len(re.findall(r"\b[\w]+(?:[-'][\w]+)*\b",mainbody));aw=len(main.split('## Abstract')[1].split('##
Keywords')[0].split());assert 2500<=wc<=5000 and aw<=360
ok(f'Length: {wc} main-text words (headings included), {aw} abstract words')
# The migration audit is an internal project check, absent from the reproduction package.
audit_available=False
if Path(__file__).with_name('check_opinion_restructure.py').is_file():
    from check_opinion_restructure import BASE,check as check_opinion
    audit_available=BASE.is_file()
if audit_available:
    check_opinion()
    ok('Opinion restructuring: numeric prose multiset contained, citation keys retained, protected inputs
unchanged')
else:
    LOG.append('SKIP Opinion restructuring: internal audit/baseline not supplied; not a reproduction
step.');
```

print(LOG[-1])

Every numeric-bearing shared unit has a source family in a readable ledger.

```

ledger=[];section='Front matter'
for bi,b in enumerate(blocks):
    if b['type']=='section':section=b['text']
    texts=[b['text']] if 'text' in b else ([t for _,t in b['items']] if b['type']=='where' else
(list(b['items']) if b['type']=='enumerate' else [b['caption'],b.get('note','')]+[' | '.join(map(str,row)) for
row in b.get('rows',[])])])
    for text in texts:
        tokens=re.findall(r'(?![A-Za-z])\d+(?:[.,]\d+)*(?:\d+)?%',text)
        if not tokens:continue
        family='cohort extract; auth6.json; authorship.json; integrity.json; computed normalization'
        if section.startswith('2 ') or section.startswith('Appendix A:'):family='cohort extract; auth6.json;
authorship.json; integrity.json; computed normalization; the record of Section A.5 (v2_report_text.txt;
final_report_text.txt; HANDOFF; session memory)'
        elif section.startswith('3 '):family='UGA-StanfordList-Salaries-FY2021-2025.xlsx via
build/ugapay.json and build/pay_figures.json (Open Georgia pay records; V8 career table)'
        elif section.startswith('4 ') or section.startswith('5 '):family='Weiskittel2026_AI.tex; cited
reference metadata; author methodological argument'
        elif section in ['Acknowledgments','Data and code availability']:family='project log/handoff; build
date and disclosed production record'
        elif b['type'] in ['title','section','subsection']:family='manuscript structural numbering'
        ledger.append({'block':bi,'section':section,'numbers':tokens,'source_family':family,'text':text})
    (ROOT/'build/numeric_traceability.json').write_text(json.dumps(ledger,indent=2),encoding='utf-8')
    inputs=[EXTRACT,SRC/'data/world_subfield_year.csv',SRC/'data/auth6.json',SRC/'data/
authorship.json',SRC/'data/integrity.json',SRC/'data/
refs.json',src_file('v2_report_text.txt'),src_file('final_report_text.txt'),src_file('HANDOFF-StanfordList-Forest
ry.md'),src_file('session_memory_stanford-list-forestry.md'),Path(r'C:
\D\Dropbox\OJS\18(1)\Aaron\Weiskittel2026_AI.tex'),Path(r'C:\D\Dropbox\OJS\18(1)\Aaron\mcfnsApr2024v2.sty')]
    source_manifest=[{'path':str(p),'bytes':p.stat().st_size,'sha256':hashlib.sha256(p.read_bytes()).hexdigest()}
for p in inputs]
    (ROOT/'build/source_manifest.json').write_text(json.dumps(source_manifest,indent=2))
    old=json.loads((ROOT/'build/before_pdf_hash.json').read_text(encoding='utf-8-
sig'));protected=ROOT/'StanfordList-Forestry-Ecology-Representation-Report.pdf'
    assert hashlib.sha256(protected.read_bytes()).hexdigest().upper()==old['Hash']
    ok('Read-only source hashes recorded; preserved original PDF hash unchanged')
    if (ROOT/'Cieszewski2026_AI_draft.pdf').exists():
        sys.path.insert(0,str(ROOT/'build/vendor'));import pymupdf as fitz
        pdf=fitz.open(ROOT/'Cieszewski2026_AI_draft.pdf');assert 12<=len(pdf)<=30,len(pdf)
        pdftext=normtext(' '.join(b[4] for p in pdf for b in p.get_text('blocks') if b[6]==0 and b[1]<740 and
b[3]>50 and b[0]>50)) # b[0]>50 drops the margin line numbers
        def pdfnorm(s):return re.sub(r'-\s+','- ',normtext(s.replace('\u00ad','').replace('\u2013','-')))
        for unit in manifest['expected text']:
            assert pdfnorm(unit) in pdfnorm(pdftext),('PDF content missing',unit[:120])
        for s in ['11.69','3.35','0.85','0.28','7,650' if '7,650' in main else '854','Competence']:assert s in
pdftext,s
        shown=sum(len(p.get_image_info()) for p in pdf);assert shown==len(shared_figures),
(shown,len(shared_figures))
        for p in pdf:
            assert '\ufffd' not in p.get_text()
            for info in p.get_image_info():
                bb=fitz.Rect(info['bbox']);assert bb.x0>=0 and bb.y0>=0 and bb.x1<=p.rect.width+1 and
bb.y1<=p.rect.height+1
            ok(f'PDF opens: {len(pdf)} pages, figures placed, all shared text units present, no replacement glyphs or
out-of-page images')
        docx_backend=json.loads((ROOT/'build/docx_build.json').read_text())['backend']
        figure_record=json.loads((ROOT/'build/figure_build.json').read_text())
        production=[
            'PRODUCTION DOCX backend: '+docx_backend+'.',
            'PRODUCTION Normalization plotting backend: '+figure_record['normalization_backend']+'.',
            'PRODUCTION Other analytical figures: '+figure_record['backend']+'.',
            'PRODUCTION The MCFNS proof is compiled by build/compile.py and measured separately by
build/check_layout.py.',
            'LIMITATION Documentary traceability of the original episodes is not independent external validation; the
interaction transcripts were not archived. Nash: no matching profile identified in the five full releases under
the documented search (nash_search.json); Schreiber reference verified against the publisher page; Scholar
unique-work estimates withdrawn.',
```

```
        'RESULT PASS: numerical, source, structural, and content checks above. See quality_review.md for the
item-by-item visual/editorial review.'
    ]
    LOG.extend(production)
    for line in production:print(line)

if __name__=='__main__':
    try:verify()
    except Exception:
        LOG.append('FAIL\n'+traceback.format_exc());print(LOG[-1]);status=1
    else:status=0
    (ROOT/'build/verify_output.txt').write_text('\n'.join(LOG)+'\n',encoding='utf-8')
    sys.exit(status)
```