

Appendix C - Model code and sources

Submission to the Productivity Commission | Inquiry into GST Distribution Reforms

Generated 2026-09-30. Attached separately from the submission.

What this is

model_2001.py is the twenty-five-year real-terms model, 2001-02 to 2025-26, and is the basis for every modelled figure in the submission (sections 4.3, 4.5, 5.3, 5.4, 6.2, 6.3 and Appendix B). All parameters are declared in a single block at the head of the file and may be substituted.

Reproducing the figures

python model_2001.py main run, contribution shares, threshold vs discount (s 4.5)
 python model_2001.py --scenarios the five stress scenarios (s 5.4, 6.3) and the projection
 python model_2001.py --aud the same on the real Australian dollar basis (s 4.3)

The model needs Python 3.9 or later and NO third-party package. All five scenarios take a single shock start date (SHOCK_FROM_FY), so sections 5.4 and 6.3 cannot diverge.

The data pipeline

The pipeline is split so that the model can be run without redistributing a third-party workbook:

```
data/CMO-Historical-Data-Annual.xlsx --[extract_inputs.py]--> inputs.csv
inputs.csv --[model_2001.py]----> the figures
```

inputs.csv holds the 360 cells the model actually consumes: calendar years 1990 to 2025, five price series, on both the real and the nominal sheet. It is about 11 KB of plain text, carries its own provenance header including the source checksum, and is attached to this submission. Everything else the model needs - calendar 2026 prices, exchange rates, Australian CPI, baseline royalty levels - is declared as a constant in model_2001.py.

Values are written with Python's repr(), which round-trips exactly, so inputs.csv reproduces the workbook's figures bit for bit. Both the extract stage and the model refuse to run on an incomplete file rather than returning quietly smaller numbers.

Status of the figures

Section 7 of the submission lists every input that is estimated rather than sourced. Outputs marked [EST] there are directional and should not be cited as findings. The effective real return applied throughout is 4.35% - the Future Fund's 4.5% real mandate net of an assumed 10% domestic allocation at 1.5 percentage points below it. That domestic allocation is an assumption of ours, not Future Fund policy.

Correction and re-use

Any party may substitute official inputs and re-run the model; we would welcome the Commission substituting its own data. fetch_sources.py downloads each source file from its publisher and reports whether the bytes match those used here. At the time of writing the Productivity Commission and Western Australian files verify byte-identical, and the World Bank has issued a later vintage of its workbook which, on re-extraction, yields the same 360 values.

Sources

Every external input behind the submission, with its publisher, download location, vintage and SHA-256 checksum, so that any party can confirm they are working from the same files.

fetch_sources.py downloads each of these and verifies the checksum.

How the pipeline uses them

```

data/CMO-Historical-Data-Annual.xlsx
| extract_inputs.py      (needs openpyxl)
v
inputs.csv               360 values, 11 KB, plain text
| model_2001.py          (Python standard library only)
v
every modelled figure in the submission

```

model_2001.py reads inputs.csv and nothing else. The other files below are cited in the submission or used by the auxiliary scripts; they are not needed to reproduce the modelled figures.

Files

1. World Bank — Commodity Price Data (the Pink Sheet), annual prices

The only source the model itself consumes. Sheets Annual Prices (Real) and Annual Prices (Nominal), calendar years 1990–2025, five series (Coal Australian; Liquefied natural gas Japan; Iron ore cfr spot; Copper; Gold) = the 360 values in inputs.csv.

File:

CMO-Historical-Data-Annual.xlsx

Vintage used:

updated 3 March 2026 (actual data to calendar 2025)

Size:

399,410 bytes

SHA-256:

32b3821b0909c9db7e30dda54b20967e7c4e49258d76143ab0f6aab1f786290e9

Landing page:

<https://www.worldbank.org/en/research/commodity-markets>

Direct URL:

<https://thedocs.worldbank.org/en/doc/74e8be41ceb20fa0da750cda2f6b9e4e-0050012026/related/CMO-Historical-Data-Annual.xlsx>

Note on vintage. The World Bank overwrites this file in place, so the direct URL now serves a later vintage (SHA-256 ed4564bca630c9b3198fc093571d58d594b82a7f79c293901e142d1d176c75d6, 3,175,855 bytes) than the one used here. Re-running extract_inputs.py against that later vintage produces the same 360 values — zero changed rows — so nothing in the submission depends on which of the two is downloaded. The checksum above records what was actually used.

2. World Bank — Commodity Price Data (the Pink Sheet), monthly prices

Used only to extend calendar 2026 from actual monthly data (2026M01–2026M07); those seven-month averages are recorded as constants in model_2001.py (EXTEND_2026) rather than read at run time.

File:

CMO-Historical-Data-Monthly.xlsx

Vintage used:

updated August 2026

Size:

577,979 bytes

SHA-256:

7902a77505ebdc5d202ce65f666c2ee1b04b626f042d7738ed3e6f7d112c8433

Direct URL:

<https://thedocs.worldbank.org/en/doc/74e8be41ceb20fa0da750cda2f6b9e4e-0050012026/related/CMO-Historical-Data-Monthly.xlsx>

3. Productivity Commission — GST distribution reforms: Interim report

The source of every Commission figure cited in the submission. Licensed CC BY 4.0 (report p. ii): "this copyright work is licensed under a Creative Commons Attribution 4.0 International licence."

File:

PC_interim_report.pdf

Size:

1,678,805 bytes

SHA-256:

24a1fbb32989772c5266e5e8d16f0ab5563aa70c46b11ed9da042148c7fbf57a

Landing page:

<https://www.pc.gov.au/inquiries-and-research/gst-reforms/interim/>

Direct URL:

<https://assets.pc.gov.au/2026-08/gst-reforms-interim.pdf>

Printed page numbers cited in the submission run 8 behind the PDF page numbers (printed p. 57 = PDF p. 65).

4. Productivity Commission — Interim report chart data workbook

Underlies Figures 1, 2.5, 2.7, 3.7, 3.8 and 4.4 as cited.

File:

PC_interim_chart_data.xlsx

Size:

164,063 bytes

SHA-256:

4836a960025d688051d2e41ace951eb877cef502957a1f6986e4fdb6a6a5c6df

Direct URL:

<https://assets.pc.gov.au/2026-08/gst-reforms-interim-chart-data.xlsx>

Verified byte-identical to the published file.

5. Government of Western Australia (DMPE) — Major Commodities Resource Data File 2025

Sheets Iron Ore – Prices and Iron Ore – Q&V, used for the currency-basis test at \$4.3 and to calibrate the Western Australian iron ore baseline.

File:

WA_major_commodities_2025.xlsx

Published:

25 May 2025; last updated 28 May 2026

Size:

1,634,511 bytes

SHA-256:

11d329d7b721dd5e95fb07ceec59425ee6869c36ceb2e25e42cb47d68105864a

Landing page:

<https://www.wa.gov.au/government/publications/major-commodities-data-file>

Direct URL:

<https://www.wa.gov.au/system/files/2026-05/2025-major-commodities-data-file.xlsx>

Verified byte-identical to the published file.

Cited in the text but not held as a data file

- Terms of Reference, received from the Treasurer 24 September 2025 (Interim Report, p. v)
- *Treasury Laws Amendment (Making Sure Every State and Territory Gets Their Fair Share of GST) Act 2018*
- Australian Government, *Interim response to PC inquiry into horizontal fiscal equalisation* (5 July 2018)
- Australian Government, 2023-24 MYEFO (13 December 2023)
- Commonwealth Grants Commission, 2024 Update; *History of General Revenue Assistance to the States* (2016)
- Government of Western Australia, Annual Report on State Finances
- Queensland Government, progressive coal royalty tiers announcement (June 2022)
- Future Fund, Annual Report 2024-25
- S. Eslake (Corinna Economic Advisory), address to the National Press Club, 8 May 2024

Inputs that are assumptions, not sources

Listed in full at §7 of the submission. In summary: baseline royalty levels for each commodity; the investment return (the Future Fund's mandate expressed in real terms, 4.5%, applied as an effective 4.35% after an assumed 10% domestic allocation at 1.5 percentage points below mandate); and the exchange-rate and CPI series used by the --aud sensitivity, which are approximations pending substitution of RBA Statistical Table F11 and ABS 6401.0.

inputs.csv - the data the model reads

The 360 values model_2001.py consumes, printed in full, so that the data is available whether or not inputs.csv is attached. That file holds the same values in long format (basis, year, series, value); extract_inputs.py regenerates it from the World Bank workbook. Values are reproduced exactly as published - none is rounded here.

360 values | inputs.csv sha256 9efce28a565996e4b6704635c12d41f197e51be7a98b4d4b69a66b00d78f667a

```
# rows: 360
# Inputs consumed by model_2001.py. Generated by extract_inputs.py on 2026-09-29.
# Source: World Bank, Commodity Price Data (the Pink Sheet), annual prices (updated 3 March 2026)
# File: CMO-Historical-Data-Annual.xlsx
# sha256: 32b3821b0909cdb7e30dda54b20967e7c4e49258d76143ab0f6aab1f786290e9
# Sheets: Annual Prices (Real) -> real_usd_2010 | Annual Prices (Nominal) -> nominal_usd
# Values are real US dollars (2010 currency) and nominal US dollars,
# written with Python repr() so that they round-trip exactly.
# See SOURCES.md for the download location and every other input.
```

Annual Prices (Real) - constant 2010 US dollars

basis key: real_usd_2010 (Coal \$/mt | Copper \$/mt | Gold \$/troy oz | IronOre \$/dmu | LNG \$/mmbtu)

Year	Coal	Copper	Gold	IronOre	LNG
1990	48	3219	464	39.3	4.4
1991	48.4	2855	442	42.4	4.87
1992	46.2	2734	412	39.7	4.31
1993	36.3	2216	417	33.7	4.06
1994	38.6	2756	459	31.6	3.8
1995	42.8	3194	418	30.9	3.75
1996	42.2	2545	430	33.3	4.07
1997	40.8	2650	385	35.1	4.55
1998	35.6	2013	358	37.7	3.67
1999	32.1	1952	346	34.2	3.89
2000	33	2279	351	36.2	5.92
2001	42.2	2061	354	39.2	6.05
2002	33.4	2061	410	38.7	5.66
2003	32.8	2235	457	40.1	5.94
2004	62.3	3370	481	44.6	6.03
2005	54.3	4195	507	74.1	6.83
2006	54.6	7475	672	77.1	7.87
2007	68.9	7459	730	128.9	8.05
2008	123.6	6764	848	151.7	12.18
2009	74.5	5339	1009	82.9	9.26
2010	99	7535	1225	145.9	10.85
2011	109.4	7950	1413	151.1	13.2
2012	87.4	7223	1515	116.6	15.02
2013	77	6681	1286	123.3	14.54
2014	64.8	6340	1169	89.6	14.81
2015	60.2	5630	1186	57.1	11.17
2016	70.3	5177	1328	62.1	7.84
2017	91	6342	1293	73.8	8.85
2018	105.1	6411	1246	68.5	10.48
2019	78.3	6042	1400	94.3	10.62
2020	62.1	6306	1808	111.3	8.49
2021	126.2	8516	1645	147.8	9.84
2022	293.7	7512	1533	103.3	15.69
2023	151.4	7441	1703	105.7	12.61
2024	118.3	7943	2074	95	11.16
2025	93.4	8568	2964	86.3	10.37

Annual Prices (Nominal) - current US dollars

basis key: nominal_usd (Coal \$/mt | Copper \$/mt | Gold \$/troy oz | IronOre \$/dmu | LNG \$/mmbtu)

Year	Coal	Copper	Gold	IronOre	LNG
1990	39.7	2661	383	32.5	3.64
1991	39.7	2339	362	34.8	3.99
1992	38.6	2281	344	33.1	3.6
1993	31.3	1913	360	29.1	3.51
1994	32.3	2307	384	26.5	3.18
1995	39.4	2936	384	28.4	3.45
1996	38.1	2295	388	30	3.67
1997	35.1	2277	331	30.2	3.91
1998	29.2	1654	294	31	3.02
1999	25.9	1573	279	27.6	3.14
2000	26.3	1813	279	28.8	4.71
2001	32.3	1578	271	30	4.63
2002	25.3	1559	310	29.3	4.28
2003	26.1	1779	364	32	4.73
2004	52.9	2866	409	37.9	5.13
2005	47.6	3679	445	65	5.99
2006	49.1	6722	604	69.3	7.08
2007	65.7	7118	697	123	7.68
2008	127.1	6956	872	156	12.53
2009	71.8	5150	973	80	8.94
2010	99	7535	1225	145.9	10.85
2011	121.4	8828	1569	167.8	14.66
2012	96.4	7962	1670	128.5	16.55
2013	84.6	7332	1411	135.4	15.96
2014	70.1	6863	1266	97	16.04

2015	58.9	5510	1161	55.9	10.93
2016	66.1	4868	1249	58.4	7.37
2017	88.5	6170	1258	71.8	8.61
2018	107	6530	1269	69.8	10.67
2019	77.9	6010	1392	93.8	10.56
2020	60.8	6174	1770	108.9	8.31
2021	138.1	9317	1800	161.7	10.76
2022	344.9	8822	1801	121.3	18.43
2023	172.8	8490	1943	120.6	14.39
2024	136.1	9142	2388	109.4	12.84
2025	108.4	9947	3442	100.2	12.04

model_2001.py

The model. Standard library only.

656 lines | sha256 b51826f9a831681eed1d5d837778dae59aa836462d9321adce59b95dadcd406d

```

1  """
2  Twenty-five-year parallel-world model -- 2001-02 to 2025-26
3  =====
4
5  Appendix C to the submission to the Productivity Commission's inquiry into GST
6  Distribution Reforms. This file is the basis for every modelled figure in the
7  submission (sections 4.3, 4.5, 5.3, 5.4, 6.2, 6.3 and Appendix B).
8
9  Why the window is 2001-02 to 2025-26:
10  1. It is the Commission's own window (Table 4.3, Figure 2.7, and the $65.4bn
11     counterfactual). A submission run on a different window cannot be placed
12     in the Commission's tables.
13  2. A short window excludes the strongest available evidence. The 2014-16 iron
14     ore collapse -- the period in which Western Australia's assessed relativity
15     reached its record low of about 0.30 -- is the clearest test of downside
16     protection, and a 2018-26 window does not contain it.
17  3. The mechanism assumes mean reversion. Eight years cannot test that
18     assumption; twenty-five years spanning a full commodity cycle can.
19
20  All quantities are in REAL terms. Returns are the Future Fund's mandate
21  expressed in real terms (CPI + 4-5%), not an actual return series.
22
23  Price basis (must agree with section 4.3 of the submission):
24  default World Bank Pink Sheet "Annual Prices (Real)" -- real US dollars,
25          2010 currency.
26  --aud World Bank nominal US dollars / AUD-USD rate / Australian CPI --
27          real Australian dollars, 2024 currency.
28  The difference is material, not rounding. The Australian dollar is a
29  commodity currency: it was near parity at the 2011 peak and near US$0.75 at
30  the 2015 trough, so converting compresses the amplitude of the cycle at both
31  ends. Because the mechanism engages only outside a +/-20% band, compressing
32  amplitude reduces contributions faster than it reduces claims. Section 4.3
33  recommends the Australian dollar basis and reports what changes under it.
34
35  Status of inputs (section 7 of the submission lists these in full):
36  [OFFICIAL] World Bank Pink Sheet annual prices, actual data to calendar 2025
37             (CM0-Historical-Data-Annual.xlsx, vintage 2026-03-03);
38             Western Australian DMPE official royalties and volumes.
39  [PARTIAL] Calendar 2026 = actual monthly series 2026M01-2026M07, deflated
40             (see EXTEND_2026).
41  [EST] Baseline royalty levels for each state (Queensland and New South
42         Wales coal inferred from known peak years).
43  [EST] The exchange-rate and CPI series used by --aud (see the warnings
44         on AUD_USD and AU_CPI_INFLATION).
45
46  Usage:
47  python model_2001.py                # main run + threshold vs discount + scenarios
48  python model_2001.py --aud          # the same on the real Australian dollar basis
49  python model_2001.py --scenarios    # scenarios only (source of sections 5.4 / 6.3)
50  """
51
52  import csv
53  import sys
54  import statistics as st
55  from pathlib import Path
56
57  if hasattr(sys.stdout, "reconfigure"):
58      sys.stdout.reconfigure(encoding="utf-8")
59
60  # The model reads one plain-text file and needs no third-party package.
61  # inputs.csv is produced by extract_inputs.py from the World Bank workbook and
62  # carries its own provenance header; SOURCES.md records where that workbook
63  # came from and its checksum.
64  INPUTS = Path(__file__).parent / "inputs.csv"
65
66  # =====
67  # Parameters
68  # =====
69  FY_START, FY_END = 2001, 2025          # FY2001-02 .. FY2025-26
70  BASELINE_WINDOW = 10                  # rolling 10-year real average price
71  BAND = 0.20                          # retained band, +/-20%
72  UPSIDE_SHARE = 0.70                  # share of the excess above the band contributed
73  PAYOUT_CAP = 0.05                    # downside payment cap = 5% of fund assets a year
74  TRANSITION_YEARS = 6
75  SPEND_RULE = 0.035
76  REAL_RETURN = 0.045                  # Future Fund mandate CPI+4-5% -> 4.5% real
77  REAL_RETURN_LOW = 0.030              # sensitivity
78  # 10% of the fund is assumed held in domestic assets returning 1.5 percentage
79  # points below the mandate. The effective real return applied by run() and
80  # project() is therefore blended(REAL_RETURN) = 4.35%, not 4.5%. This is an
81  # assumption of ours, not Future Fund policy; it is disclosed in section 7.
82  LOCAL_ALLOCATION, LOCAL_SPREAD = 0.10, -0.015
83
84  # Shock start year for the stress scenarios. Sections 5.4 and 6.3 must cite the

```

```

85 # same value; holding it in one constant is what guarantees they cannot diverge.
86 SHOCK_FROM_FY = 2014 # = FY2014-15
87
88 # Baseline royalties (A$100 million, real 2024-25 currency): the annual royalty
89 # each commodity yields when its price equals the rolling baseline.
90 # [EST] Queensland and New South Wales are inferred by dividing a known peak
91 # year by that year's price ratio; Western Australian iron ore is
92 # calibrated to official DMPE sales values.
93 #
94 # Known proxy errors in the price series, disclosed in section 7:
95 # Coal: the World Bank "Coal, Australian" series is THERMAL coal fob
96 # Newcastle. Queensland's royalties are predominantly METALLURGICAL coal,
97 # which the Commission assesses separately (its Table 3.4). Queensland's
98 # contribution share is therefore sensitive to the 2021-23 energy episode
99 # rather than to the metallurgical cycle.
100 # LNG: the series is the Japanese cif import price, and most Australian LNG
101 # is offshore and taxed by the Commonwealth under the PRRT rather than by a
102 # state royalty. Only Queensland's coal-seam-gas LNG clearly sits within a
103 # mechanism operating on state royalties, so the "Multi-state" row should
104 # be read as an upper bound.
105 # Iron ore: the World Bank series is CFR spot including freight, whereas the
106 # Commission's Figure 3.7 is fob; and the series transitions from annual
107 # contract pricing to spot in 2009-10 (nominal 79.98 -> 145.86), so part of
108 # the apparent increase is a change of measurement, not of price.
109 COMMODITIES = {
110     "Iron ore (WA)": dict(state="WA", series="IronOre", base=65.0),
111     "Coal (QLD)": dict(state="QLD", series="Coal", base=40.0),
112     "Coal (NSW)": dict(state="NSW", series="Coal", base=15.0),
113     "LNG (multi-state)": dict(state="Multi-state", series="LNG", base=12.0),
114     "Copper (SA/NSW)": dict(state="SA", series="Copper", base=4.0),
115     "Gold (WA)": dict(state="WA", series="Gold", base=8.0),
116 }
117
118 # Calendar 2025 is now ACTUAL data: the annual workbook
119 # (CMO-Historical-Data-Annual.xlsx, updated 3 March 2026) contains the complete
120 # 2025 average, so no value is entered by hand for that year.
121 #
122 # Calendar 2026 is not complete and must still be extended, but it is extended
123 # from ACTUAL monthly data rather than estimated:
124 # source data/CMO-Historical-Data-Monthly.xlsx, Monthly Prices,
125 # 2026M01-2026M07 (seven months)
126 # nominal year to date: Coal 129.29 / LNG 12.34 / IronOre 103.21 /
127 # Copper 13154.43 / Gold 4605.43
128 # deflated to real 2010 US dollars by the MUV index 1.1711
129 # (= the 2025 actual of 1.1610 extended one year at the 0.87% increase
130 # observed from 2024 to 2025; the 2026 MUV is not yet published).
131 # To refresh: replace the nominal values above with the new year-to-date
132 # averages from the monthly workbook and divide by that year's MUV.
133 # Note that 2026 covers seven months only, and that copper and gold have moved
134 # sharply during the year, so the full-year average may differ.
135 FIRST_CY = 1990 # first year inputs.csv covers
136 LAST_ACTUAL_CY = 2025 # last year of actual annual data
137 MUV_2026 = 1.1711 # [EST] extrapolated; replace when published
138 EXTEND_2026 = dict(Coal=110.4, LNG=10.5, IronOre=88.1, Copper=11232.8, Gold=3932.7)
139
140 # =====
141 # Exchange rates and inflation, used only by the --aud sensitivity
142 # =====
143 # [EST] The two tables below are approximations for a directional sensitivity
144 # test and should not be cited. Before they are relied upon they must be
145 # replaced with the official series:
146 # AUD_USD -> RBA Statistical Table F11 "Exchange Rates",
147 # A$/US$ calendar-year averages
148 # AU_CPI_INFLATION -> ABS 6401.0 Table 1 "All groups CPI, Australia",
149 # calendar-year average annual change
150 # Substituting them requires editing only these two dictionaries.
151 AUD_USD = {
152     1990: .781, 1991: .779, 1992: .735, 1993: .680, 1994: .732, 1995: .741,
153     1996: .783, 1997: .744, 1998: .629, 1999: .646, 2000: .581, 2001: .518,
154     2002: .543, 2003: .654, 2004: .736, 2005: .762, 2006: .753, 2007: .838,
155     2008: .853, 2009: .792, 2010: .919, 2011: 1.033, 2012: 1.036, 2013: .968,
156     2014: .903, 2015: .752, 2016: .744, 2017: .767, 2018: .748, 2019: .695,
157     2020: .690, 2021: .751, 2022: .694, 2023: .664, 2024: .660, 2025: .650,
158     2026: .650,
159 }
160 AU_CPI_INFLATION = { # calendar-year CPI change (%)
161     1991: 3.2, 1992: 1.0, 1993: 1.8, 1994: 1.9, 1995: 4.6, 1996: 2.6,
162     1997: 0.3, 1998: 0.9, 1999: 1.5, 2000: 4.5, 2001: 4.4, 2002: 3.0,
163     2003: 2.8, 2004: 2.3, 2005: 2.7, 2006: 3.5, 2007: 2.3, 2008: 4.4,
164     2009: 1.8, 2010: 2.8, 2011: 3.4, 2012: 1.8, 2013: 2.4, 2014: 2.5,
165     2015: 1.5, 2016: 1.3, 2017: 2.0, 2018: 1.9, 2019: 1.6, 2020: 0.9,
166     2021: 2.8, 2022: 6.6, 2023: 5.6, 2024: 3.2, 2025: 2.8, 2026: 2.6,
167 }
168 CPI_BASE_YEAR = 2024 # real-currency base year for --aud
169
170
171 def _cpi_index():
172     idx = {1990: 100.0}
173     for y in range(1991, 2027):

```

```

174     idx[y] = idx[y - 1] * (1 + AU_CPI_INFLATION[y] / 100)
175     return idx
176
177
178 # =====
179 # Data
180 # =====
181 SERIES = ("Coal", "Copper", "Gold", "IronOre", "LNG")
182
183
184 def _read_inputs(basis_key):
185     """Read one price basis out of inputs.csv.
186
187     basis_key is 'real_usd_2010' or 'nominal_usd'. The file is produced by
188     extract_inputs.py; to work from a different World Bank vintage, replace
189     the workbook in data/ and re-run that script."""
190     if not INPUTS.exists():
191         raise RuntimeError(
192             f"{INPUTS.name} not found. Generate it with 'python extract_inputs.py' "
193             f"(needs the World Bank workbook in data/ -- see SOURCES.md).")
194
195     out, total, declared = {}, 0, None
196     with INPUTS.open(encoding="utf-8") as fh:
197         body = []
198         for line in fh:
199             if line.startswith("#"):
200                 if line.startswith("# rows:"):
201                     declared = int(line.split(":", 1)[1])
202                 continue
203             body.append(line)
204         rdr = csv.DictReader(body)
205         want_cols = {"basis", "year", "series", "value"}
206         if set(rdr.fieldnames or ()) != want_cols:
207             raise RuntimeError(f"{INPUTS.name} has columns {rdr.fieldnames}, "
208                               f"expected {sorted(want_cols)}")
209         for row in rdr:
210             total += 1
211             if row["basis"] != basis_key:
212                 continue
213             out.setdefault(int(row["year"]), {})[row["series"]] = float(row["value"])
214
215     # A truncated file would otherwise pass unnoticed whenever the rows it lost
216     # belong to the other price basis.
217     if declared is None:
218         raise RuntimeError(f"{INPUTS.name} has no '# rows:' header; regenerate it "
219                           f"with 'python extract_inputs.py'")
220     if total != declared:
221         raise RuntimeError(f"{INPUTS.name} declares {declared} rows but contains "
222                           f"{total}; the file is truncated or has been edited")
223
224     # Silently dropping cells would make the model return quietly smaller
225     # numbers. An incomplete or truncated file must fail here, not drift in
226     # the results.
227     if not out:
228         raise RuntimeError(f"{INPUTS.name} contains no rows for basis '{basis_key}'")
229     gap_y = sorted(set(range(FIRST_CY, LAST_ACTUAL_CY + 1)) - set(out))
230     if gap_y:
231         raise RuntimeError(f"{INPUTS.name}[{basis_key}] is missing years: {gap_y}")
232     want = set(SERIES)
233     gap_s = {y: sorted(want - set(v)) for y, v in sorted(out.items()) if set(v) != want}
234     if gap_s:
235         raise RuntimeError(f"{INPUTS.name}[{basis_key}] is missing series: {gap_s}")
236     return out
237
238
239 def load_prices(basis="usd"):
240     """basis='usd': World Bank real US dollars, 2010 currency -- the default,
241         and the basis used throughout the body of the submission.
242         basis='aud': nominal US dollars / exchange rate / Australian CPI = real
243         Australian dollars, 2024 currency -- the sensitivity."""
244     if basis == "usd":
245         out = _read_inputs("real_usd_2010")
246         out[2026] = EXTEND_2026
247         return out
248
249     nom = _read_inputs("nominal_usd")
250     # The World Bank real series is deflated to 2010 US dollars by the MUV
251     # index. EXTEND_2026 is expressed in real dollars, so it is restored to
252     # nominal by multiplying by that index. [EST] see MUV_2026.
253     nom[2026] = {k: v * MUV_2026 for k, v in EXTEND_2026.items()}
254
255     cpi = _cpi_index()
256     out = {}
257     for y, d in nom.items():
258         if y not in AUD_USD:
259             continue
260         defl = cpi[y] / cpi[CPI_BASE_YEAR]
261         out[y] = {k: (v / AUD_USD[y]) / defl for k, v in d.items()}
262     return out

```

```

263
264
265 # Retained under the original name so that existing callers (plots.py,
266 # plots_en.py) continue to work unchanged.
267 def load_real_prices():
268     return load_prices("usd")
269
270
271 def fy_price(cy_prices, series, fy):
272     """Financial-year price = (calendar t + calendar t+1) / 2, because a
273     financial year spans two calendar years."""
274     a = cy_prices.get(fy, {}).get(series)
275     b = cy_prices.get(fy + 1, {}).get(series)
276     vals = [v for v in (a, b) if v is not None]
277     return sum(vals) / len(vals) if vals else None
278
279
280 def build_ratios(cy_prices):
281     """For each commodity and financial year: real price / rolling 10-year
282     real average price."""
283     ratios, baselines = {}, {}
284     for name, c in COMMODITIES.items():
285         s = c["series"]
286         ratios[name], baselines[name] = {}, {}
287         for fy in range(FY_START, FY_END + 1):
288             window = [fy_price(cy_prices, s, y)
289                       for y in range(fy - BASELINE_WINDOW, fy)]
290             window = [w for w in window if w]
291             p = fy_price(cy_prices, s, fy)
292             if window and p:
293                 base = sum(window) / len(window)
294                 ratios[name][fy] = p / base
295                 baselines[name][fy] = base
296     return ratios, baselines
297
298
299 def shocked(ratios, level, from_fy=SHOCK_FROM_FY):
300     """Stress scenario: from from_fy onward, every commodity's price ratio is
301     held at level (a structural decline rather than a cycle)."""
302     if level is None:
303         return ratios
304     return {name: {fy: (level if fy >= from_fy else r) for fy, r in series.items()}}
305     for name, series in ratios.items()
306
307
308 # =====
309 # Model
310 # =====
311 def blended(r):
312     """Mandate return net of the domestic allocation: the EFFECTIVE real return
313     actually applied. blended(0.045) = 0.0435."""
314     return (1 - LOCAL_ALLOCATION) * r + LOCAL_ALLOCATION * (r + LOCAL_SPREAD)
315
316
317 def run(ratios, real_return=REAL_RETURN, band=BAND, share=UPSIDE_SHARE):
318     """Two design asymmetries, disclosed in section 7 of the submission:
319     contributions are taken at share (0.70) of the excess above the band,
320     while claims are calculated on 100% of the shortfall below it; and the
321     six-year ramp applies to CONTRIBUTIONS only, not to claims.
322     Both understate the fund's balance, so both are conservative. They mean
323     the mechanism can face a claim in its first years that it cannot meet.
324     The historical run raises no claim in the first six years, so this does
325     not show there; on the --aud basis a claim falls on an empty fund in
326     2002-03. This is deliberate, not an oversight -- see section 6.4."""
327     ceil, floor = 1 + band, 1 - band
328     r = blended(real_return)
329     assets = 0.0
330     rows = []
331     for i, fy in enumerate(range(FY_START, FY_END + 1)):
332         ramp = min(1.0, (i + 1) / TRANSITION_YEARS)
333         inflow = claim = raw = 0.0
334         trig = []
335         by_state = {}
336         for name, c in COMMODITIES.items():
337             ratio = ratios[name].get(fy)
338             if ratio is None:
339                 raise RuntimeError(
340                     f"{name} has no price ratio for FY{fy}; skipping it "
341                     f"silently would make that year's figures too small.")
342             b = c["base"]
343             raw += b * ratio
344             up = b * max(0.0, ratio - ceil) * share * ramp
345             inflow += up
346             by_state[c["state"]] = by_state.get(c["state"], 0.0) + up
347             dn = b * max(0.0, floor - ratio)
348             if dn > 0:
349                 claim += dn
350                 trig.append(f"{name} ({ratio:.2f})")
351     paid = min(claim, PAYOUT_CAP * assets)

```

```

352     eq = SPEND_RULE * assets if ramp >= 1.0 else 0.0
353     assets = max(0.0, assets * (1 + r) + inflow - paid - eq)
354     rows.append(dict(fy=fy, ramp=ramp, raw=raw, inflow=inflow, claim=claim,
355                     paid=paid, eq=eq, assets=assets,
356                     net=raw - inflow + paid, trig=trig, states=by_state))
357     return rows
358
359
360 def project(assets, annual_claim, real_return=REAL_RETURN, years=50):
361     """Beyond the window: how the fund behaves if the shock persists and no
362     further contribution is ever made. This is the test of what the claim
363     that the fund 'cannot be drained' actually means."""
364     r = blended(real_return)
365     out = []
366     for k in range(1, years + 1):
367         paid = min(annual_claim, PAYOUT_CAP * assets)
368         eq = SPEND_RULE * assets
369         assets = max(0.0, assets * (1 + r) - paid - eq)
370         out.append((k, assets, paid, paid / annual_claim if annual_claim else 1.0))
371     return out
372
373
374 def cv(xs):
375     m = st.mean(xs)
376     return st.pstdev(xs) / m if m else 0.0
377
378
379 def mayc(xs):
380     """Mean absolute year-on-year change, in levels (A$100 million)."""
381     return st.mean([abs(xs[i] - xs[i - 1]) for i in range(1, len(xs))])
382
383
384 def mayc_pct(xs):
385     """Mean absolute year-on-year change, as a percentage -- the measure the
386     Commission actually uses in Table 4.3 and Figure 4.4.
387     Note that proportional scaling does NOT change this quantity, so on the
388     Commission's measure a mining discount reduces volatility by zero."""
389     return st.mean([abs(xs[i] - xs[i - 1]) / xs[i - 1] for i in range(1, len(xs))])
390
391
392 # =====
393 # Scenarios -- the single source for sections 5.4 and 6.3
394 # =====
395 # (label, post-shock price ratio or None, real return)
396 SCENARIOS = [
397     ("Historical (cyclical)", None, REAL_RETURN),
398     ("Structural decline - all commodities 0.5x from 2014-15", 0.5, REAL_RETURN),
399     ("Lost two decades - zero real return, prices as history", None, 0.0),
400     ("Correlated failure - structural decline AND zero return", 0.5, 0.0),
401     ("Severe collapse - prices 0.3x, zero real return", 0.3, 0.0),
402 ]
403
404
405 def scenario_rows(ratios):
406     out = []
407     for label, level, rr in SCENARIOS:
408         rows = run(shocked(ratios, level), real_return=rr)
409         claims = sum(x["claim"] for x in rows)
410         paid = sum(x["paid"] for x in rows)
411         out.append(dict(label=label, level=level, rr=rr, rows=rows,
412                        claims=claims, paid=paid,
413                        cov=(paid / claims if claims else 1.0),
414                        end=rows[-1]["assets"],
415                        eq=sum(x["eq"] for x in rows),
416                        inflow=sum(x["inflow"] for x in rows)))
417     return out
418
419
420 # =====
421 # Reporting
422 # =====
423 def fyl(fy):
424     return f"{fy}-{str(fy + 1)[2:]}"
425
426
427 def bn(x):
428     """A$100 million units -> $bn."""
429     return x / 10.0
430
431
432 def print_scenarios(ratios):
433     print("\n" + "=" * 78)
434     print(f"STRESS SCENARIOS (shock from FY{fyl(SHOCK_FROM_FY)}) "
435           f"-- sections 5.4 and 6.3")
436     print("=" * 78)
437     scs = scenario_rows(ratios)
438     print(f"  {'Scenario':<58}{'Coverage':>9}{'Fund at end':>13}")
439     print(" " + "-" * 78)
440     for s in scs:

```

```

441     print(f"    {s['label']}<58>{s['cov']*100:>8.0f}%{bn(s['end'])>11.1f}bn")
442     print()
443     print(f"    {'':<58}{Claims':>10}{Paid':>9}{Equalisn':>10}{Inflow':>9}")
444     for s in scs:
445         print(f"        {s['label']}<58>{bn(s['claims'])>9.1f}b{bn(s['paid'])>8.1f}b"
446               f"bn(s['eq'])>9.1f}b{bn(s['inflow'])>8.1f}b")
447
448     # Scenarios 4 and 5 share a fund path: the 5% cap binds throughout, so only
449     # the denominator (total claims) differs.
450     s4, s5 = scs[3], scs[4]
451     if abs(s4["end"] - s5["end"]) < 0.05:
452         print(f"\n    [note] Scenarios 4 and 5 share one fund path (both end at "
453               f"bn(s4['end']):.1f}bn, both pay {bn(s4['paid']):.1f}bn.")
454         print("        The 5% cap binds throughout, so only the claim total differs.")
455         print("        They are one path with two claim totals, not two degrees")
456         print("        of resilience, and must not be presented as a gradient.")
457
458     # What "the fund cannot be drained" actually means
459     struct = scs[1]
460     annual_claim = sum(c["base"] for c in COMMODITIES.values()) * (1 - BAND - 0.5)
461     print("\n" + "-" * 78)
462     print("    Testing 'the fund cannot be drained': the structural-decline scenario,"
463           " extrapolated beyond the end of the window")
464     print(f"    (annual claim held at {bn(annual_claim):.2f}bn, no further inflow)")
465     print(f"    {'Years on':>9}{Fund':>12}{Payment':>11}{Coverage':>11}")
466     for k, a, p, c in project(struct["end"], annual_claim):
467         if k in (1, 10, 20, 30, 50):
468             print(f"        {k:>9}{bn(a)>10.1f}bn{bn(p)>9.2f}bn{c*100:>10.1f}%")
469     _draw = PAYOUT_CAP + SPEND_RULE
470     _eff = blended(REAL_RETURN)
471     print(f"    -> Drawings {PAYOUT_CAP*100:.0f}% + {SPEND_RULE*100:.1f}% = {_draw*100:.1f}% "
472           f"exceed the effective real return of {_eff*100:.2f}%")
473     print(f"    (mandate {REAL_RETURN*100:.1f}% net of the "
474           f"{LOCAL_ALLOCATION*100:.0f}% domestic allocation), so the balance decays")
475     print(f"    geometrically at about {(_draw-_eff)*100:.1f}% a year.")
476     print("    It never reaches zero, but that is a mathematical property and")
477     print("    not a fiscal one: the payment halves within two decades. The")
478     print("    payment is not permanent; what the proportional cap guarantees")
479     print("    is that the fund can never be called on for more than it holds.")
480     return scs
481
482
483 def print_threshold_vs_discount(ratios):
484     rows = run(ratios)
485     raw = [x["raw"] for x in rows]
486     b_l, b_p = mayc(raw), mayc_pct(raw)
487
488     print("\n" + "-" * 78)
489     print("THRESHOLD vs MINING DISCOUNT -- comparison with Figure 4.4 (25-year window)")
490     print("-" * 78)
491     print(f"    Baseline mean absolute year-on-year change: "
492           f"bn(b_l:.1f) (levels, A$100m) / {b_p*100:.1f}% (percentage)")
493     print(f"\n    {'Instrument':<26}{Excluded':>10}{Lvl redn':>10}{Effcy':>7}")
494     print(f"    {'Pct redn':>10}{Effcy':>7}")
495     print("    " + "-" * 70)
496     for d in (0.10, 0.25, 0.50):
497         n = [v * (1 - d) for v in raw]
498         rl = (1 - mayc(n) / b_l) * 100
499         rp = (1 - mayc_pct(n) / b_p) * 100
500         print(f"    {'Mining discount ' + str(int(d*100)) + '%':<26}{d*100:>9.1f}%{rl:>9.1f}%"
501               f"rl/(d*100)>7.2f}{rp:>9.1f}%{rp/(d*100)>7.2f}")
502     print()
503     for band in (0.15, 0.20, 0.25):
504         for share in (0.70, 1.00):
505             rr = run(ratios, band=band, share=share)
506             n = [x["raw"] - x["inflow"] for x in rr]
507             e = sum(x["inflow"] for x in rr) / sum(x["raw"] for x in rr) * 100
508             rl = (1 - mayc(n) / b_l) * 100
509             rp = (1 - mayc_pct(n) / b_p) * 100
510             tag = f"Threshold +/- {int(band*100)}%, {int(share*100)}% share"
511             print(f"    {tag:<26}{e:>9.1f}%{rl:>9.1f}%{rl/e:>7.2f}{rp:>9.1f}%{rp/e:>7.2f}")
512
513     print("\n    [key] On the levels measure a discount's efficiency is always 1.00.")
514     print("    That is an artefact of the MEASURE, not a finding. Table 4.3 and")
515     print("    Figure 4.4 use the PERCENTAGE measure, and proportional scaling")
516     print("    does not change year-on-year percentages, so on the Commission's")
517     print("    own measure a discount reduces volatility by 0.0%. Only the")
518     print("    Commission's two axes (below) may be used externally.")
519
520     # Figure 4.4 (printed p.80), as published by the Commission
521     print("\n" + "-" * 78)
522     print("    Figure 4.4 (printed p.80) on its own two axes -- the Commission's")
523     print("    own framework for assessing a proposal")
524     print("-" * 78)
525     pc_eq = {"pre2018": 100, 10: 95, 25: 88, 50: 75, "current": 73}
526     pc_vol = {"pre2018": 4.6, 10: 4.1, 25: 3.6, 50: 3.3, "current": 2.9}
527     print(f"    {'Instrument':<26}{Equalisn':>10}{Forgone':>9}{PC vol':>9}")
528     print(f"    {'Redn':>8}{Effcy':>7}")
529     effs = []

```

```

530     for d in (10, 25, 50):
531         forgone = pc_eq["pre2018"] - pc_eq[d]
532         red = (pc_vol["pre2018"] - pc_vol[d]) / pc_vol["pre2018"] * 100
533         effs.append(red / forgone)
534         print(f"    {'Mining discount ' + str(d) + '%':<26}{pc_eq[d]:>9}%{forgone:>7}pp"
535               f"{pc_vol[d]:>8.1f}%{red:>7.1f}%{red/forgone:>7.2f}")
536     print(f"    -> On the Commission's own numbers the efficiencies are "
537           f"'{' / '.join(f'{e:.2f}' for e in effs)}'; none is 1.00.")
538     print("    A discount scales ASSESSED MINING REVENUE, which passes through")
539     print("    the redistribution formula non-linearly; it is not a proportional")
540     print("    scaling of each state's GST revenue.")
541
542
543 def print_main(ratios, basis):
544     rows = run(ratios)
545     raw = [x["raw"] for x in rows]
546     net = [x["net"] for x in rows]
547
548     label = ("World Bank real US dollars, 2010 currency" if basis == "usd"
549             else f"real Australian dollars, {CPI_BASE_YEAR} currency "
550             f"(exchange rate and CPI are [EST])")
551     print("=" * 78)
552     print("TWENTY-FIVE-YEAR PARALLEL-WORLD MODEL -- 2001-02 to 2025-26")
553     print("=" * 78)
554     print(f"    Price basis: {label}")
555     print(f"    Band +/-{BAND*100:.0f}% | upside share {UPSIDE_SHARE*100:.0f}%")
556     print(f"    f" | payout cap {PAYOUT_CAP*100:.0f}%/yr"
557           f"    f" | real return: mandate {REAL_RETURN*100:.1f}% -> effective "
558           f"    f"{blended(REAL_RETURN)*100:.2f}% ({LOCAL_ALLOCATION*100:.0f}% domestic)")
559     print(f"    Baseline = rolling {BASELINE_WINDOW}-year real average price")
560
561     print(f"\n{'Fin year':<10}{ 'Rent':>9}{ 'Inflow':>9}{ 'Claim':>9}"
562           f"{'Paid':>8}{ 'Fund':>11} Triggered by)")
563     for x in rows:
564         t = "; ".join(x["trig"][:34])
565         print(f"{'f'f'yl(x['fy']):<10}{x['raw']:>9.1f}{x['inflow']:>9.1f}"
566               f"{'f'f'x['claim']:>9.1f}{x['paid']:>8.1f}{x['assets']:>11.1f} {t}")
567
568     claims = sum(x["claim"] for x in rows)
569     paid = sum(x["paid"] for x in rows)
570     inflow = sum(x["inflow"] for x in rows)
571     eqtot = sum(x["eq"] for x in rows)
572     print("\n" + "=" * 78)
573     print("RESULTS (A$100 million units; $bn shown alongside)")
574     print("=" * 78)
575     print(f"    Cumulative contributions to the fund {inflow:>9.1f} = ${bn(inflow):.1f}bn")
576     print(f"    Cumulative payments to resource states {paid:>9.1f} = ${bn(paid):.1f}bn")
577     print(f"    Cumulative equalisation distributions {eqtot:>9.1f} = ${bn(eqtot):.1f}bn")
578     print(f"    Fund at end of window {rows[-1]['assets']:>9.1f}"
579           f"    f" = ${bn(rows[-1]['assets']):.1f}bn")
580     print(f"    Coverage of downside claims {(paid/claims*100 if claims else 100):>9.1f} %")
581
582     low = run(ratios, real_return=REAL_RETURN_LOW)
583     print(f"    Fund at end (mandate {REAL_RETURN_LOW*100:.1f}% -> effective "
584           f"    f" {blended(REAL_RETURN_LOW)*100:.2f}%) {low[-1]['assets']:>7.1f}"
585           f"    f" = ${bn(low[-1]['assets']):.1f}bn")
586
587     # The permanent obligation: the Commission's $5.7bn permanent-guarantee
588     # cost at p.86 is the denominator.
589     perm = SPEND_RULE * rows[-1]["assets"]
590     print(f"\n    Sustainable annual payment ({SPEND_RULE*100:.1f}% spending rule) "
591           f"    f"${bn(perm):.2f}bn/yr")
592     print(f"    = {bn(perm)/5.7*100:.0f}% of the Commission's $5.7bn permanent guarantee")
593     print(f"    Assets required to cover $5.7bn in full: ${5.7/SPEND_RULE:.0f}bn")
594
595     trig = [x for x in rows if x["claim"] > 0]
596     print(f"\n    Downside protection triggered in {len(trig)} years:")
597     for x in trig:
598         gap = x["claim"] - x["paid"]
599         flag = f" [shortfall {gap:.1f}]" if gap > 0.05 else ""
600         print(f"    {'f'f'yl(x['fy'])} claim {x['claim']:>5.1f} / paid {x['paid']:>5.1f}{flag}"
601               f"    for t in x['trig']:"
602               f"    print(f"        - {t}")
603
604     print(f"\n    Volatility (CV) {cv(raw):.3f} -> {cv(net):.3f}"
605           f"    f" (down {(1-cv(net)/cv(raw))*100:.1f}%)")
606     print(f"    Volatility (mean abs yr-on-yr, levels) {mayc(raw):.1f} -> {mayc(net):.1f}"
607           f"    f" (down {(1-mayc(net)/mayc(raw))*100:.1f}%)")
608     print("    [note] These measure the SYNTHETIC NATIONAL RESOURCE-RENT SERIES, not")
609     print("    GST. Figure 2.7 measures year-on-year change in each state's")
610     print("    assessed relativity; Table 4.3 and Figure 4.4 measure change in")
611     print("    real GST revenue per capita. The CGC's three-year averaging and")
612     print("    the redistribution formula sit between them. They are related")
613     print("    measures but they are not the same criterion.")
614
615     tot = {}
616     for x in rows:
617         for s, v in x["states"].items():
618             tot[s] = tot.get(s, 0.0) + v

```

```

619     g = sum(tot.values())
620     print("\n" + "=" * 78)
621     print("CONCENTRATION OF CONTRIBUTIONS -- 25-year cumulative")
622     print("=" * 78)
623     for s, v in sorted(tot.items(), key=lambda kv: -kv[1]):
624         print(f"  {s:<13}{v:>8.1f}  {v/g*100:>5.1f}%  {' '*int(v/g*46)}")
625     print(f"  {'VIC/TAS/ACT':<13}{0.0:>8.1f}  {0.0:>5.1f}%  <- beneficiaries only")
626     wa = tot.get("WA", 0.0)
627     print(f"\n  Western Australia's net position: contributes ${bn(wa):.1f}bn, "
628           f"draws ${bn(paid):.1f}bn")
629     print("  in downside payments, plus its population share of the fund's")
630     print("  equalisation distributions.")
631
632
633 def main():
634     argv = sys.argv[1:]
635     basis = "aud" if "--aud" in argv else "usd"
636     only_scen = "--scenarios" in argv
637
638     cy = load_prices(basis)
639     ratios, _ = build_ratios(cy)
640
641     if not only_scen:
642         print_main(ratios, basis)
643         print_threshold_vs_discount(ratios)
644     print_scenarios(ratios)
645
646     print("\n" + "=" * 78)
647     print("[EST] Baseline royalty levels, 2025-26 prices, and the coal and LNG")
648     print("  price-series proxies. See section 7 of the submission.")
649     if basis == "aud":
650         print("[EST] AUD_USD and AU_CPI_INFLATION are approximations and must be")
651         print("  replaced with RBA Table F11 and ABS 6401.0 before citation.")
652     print("=" * 78)
653
654
655 if __name__ == "__main__":
656     main()

```

extract_inputs.py

Builds inputs.csv from the World Bank workbook. Needs openpyxl.

145 lines | sha256 f73820aa9e902c7e9b4b2181cecee68bd9b41812fc2875ca594cff3f0b8823c4

```

1  """
2  Extract stage: World Bank workbook -> inputs.csv
3
4  The submission undertakes that any party may substitute official inputs and
5  re-run the model. That undertaking is only meaningful if the model can be run
6  without redistributing a 0.4 MB third-party workbook, so the pipeline is split
7  in two:
8
9      extract_inputs.py    reads CMO-Historical-Data-Annual.xlsx -> inputs.csv
10     model_2001.py        reads inputs.csv only                    -> the figures
11
12 inputs.csv holds the 360 cells the model actually consumes: calendar years
13 1990 to 2025, five price series, on both the real and the nominal sheet. It is
14 about 11 KB, is plain text, and carries its own provenance header, so it can be
15 attached to the submission and read in place. model_2001.py needs no
16 third-party package at all once it exists.
17
18 Values are written with repr(), which round-trips exactly in Python, so the
19 extracted CSV reproduces the workbook's figures bit for bit.
20
21 Run: python extract_inputs.py
22 """
23 import csv
24 import hashlib
25 from datetime import date
26 from pathlib import Path
27
28 import openpyxl
29
30 HERE = Path(__file__).parent
31 DATA = HERE / "data"
32 OUT = HERE / "inputs.csv"
33
34 SOURCE = "CMO-Historical-Data-Annual.xlsx"
35 SOURCE_PUBLISHER = "World Bank"
36 SOURCE_TITLE = "Commodity Price Data (the Pink Sheet), annual prices"
37 SOURCE_VINTAGE = "updated 3 March 2026"
38
39 FIRST_CY, LAST_CY = 1990, 2025
40
41 SHEETS = {
42     "Annual Prices (Real)": "real_usd_2010",
43     "Annual Prices (Nominal)": "nominal_usd",
44 }
45
46 # Columns are located by HEADING, not by a fixed index: the World Bank shifts
47 # them between vintages.
48 SERIES = {
49     "Coal, Australian": "Coal",                # thermal coal fob Newcastle ($/mt)
50     "Liquefied natural gas, Japan": "LNG",      # Japanese cif import price ($/mmbtu)
51     "Iron ore, cfr spot": "IronOre",           # CFR spot ($/dmtu)
52     "Copper": "Copper",                        # ($/mt)
53     "Gold": "Gold",                            # ($/troy oz)
54 }
55
56
57 def sha256(path):
58     return hashlib.sha256(path.read_bytes()).hexdigest()
59
60
61 def read_sheet(name):
62     wb = openpyxl.load_workbook(DATA / SOURCE, data_only=True, read_only=True)
63     ws = wb[name]
64     rows = list(ws.iter_rows(values_only=True))
65     wb.close()
66
67     hdr = next(i for i, r in enumerate(rows[:12])
68                 if r[1] and "Crude oil" in str(r[1]))
69     col = {}
70     for j, n in enumerate(rows[hdr]):
71         key = SERIES.get(str(n).strip()) if n else ""
72         if key and key not in col:
73             col[key] = j
74     missing = set(SERIES.values()) - set(col)
75     if missing:
76         raise RuntimeError(f"{SOURCE}[{name}] is missing columns: {sorted(missing)}. "
77                             f"The World Bank has renamed or moved them; update SERIES.")
78
79     out = {}
80     for r in rows:
81         try:
82             y = int(str(r[0]).strip())
83         except (TypeError, ValueError):
84             continue

```

```

85     if FIRST_CY <= y <= LAST_CY:
86         out[y] = {k: r[c] for k, c in col.items()
87                 if isinstance(r[c], (int, float))}
88
89     # Fail here rather than let the model return quietly smaller numbers.
90     gap_y = sorted(set(range(FIRST_CY, LAST_CY + 1)) - set(out))
91     if gap_y:
92         raise RuntimeError(f"{SOURCE}[{name}] is missing years: {gap_y}")
93     want = set(SERIES.values())
94     gap_s = {y: sorted(want - set(v)) for y, v in sorted(out.items()) if set(v) != want}
95     if gap_s:
96         raise RuntimeError(f"{SOURCE}[{name}] is missing series: {gap_s}")
97     return out
98
99
100 def build():
101     src = DATA / SOURCE
102     if not src.exists():
103         raise SystemExit(f"{src} not found. Run fetch_sources.py first, or see SOURCES.md.")
104     digest = sha256(src)
105
106     data = {basis: read_sheet(sheet) for sheet, basis in SHEETS.items()}
107
108     expect = len(SHEETS) * (LAST_CY - FIRST_CY + 1) * len(set(SERIES.values()))
109
110     with OUT.open("w", newline="", encoding="utf-8") as fh:
111         for line in (
112             f"# rows: {expect}",
113             f"# Inputs consumed by model_2001.py. Generated by extract_inputs.py ",
114             f"on {date.today().isoformat()}.",
115             f"# Source: {SOURCE_PUBLISHER}, {SOURCE_TITLE} ({SOURCE_VINTAGE})",
116             f"# File: {SOURCE}",
117             f"# sha256: {digest}",
118             f"# Sheets: {' | '.join(f'{s} -> {b}' for s, b in SHEETS.items())}",
119             f"# Values are real US dollars (2010 currency) and nominal US dollars,",
120             f"# written with Python repr() so that they round-trip exactly.",
121             f"# See SOURCES.md for the download location and every other input.",
122         ):
123             fh.write(line + "\n")
124         w = csv.writer(fh)
125         w.writerow(["basis", "year", "series", "value"])
126         n = 0
127         for basis in SHEETS.values():
128             for y in sorted(data[basis]):
129                 for s in sorted(data[basis][y]):
130                     w.writerow([basis, y, s, repr(data[basis][y][s])])
131                     n += 1
132
133     if n != expect:
134         raise RuntimeError(f"wrote {n} rows, expected {expect}")
135
136     print(f"Saved {OUT.name}")
137     print(f" rows    {n} ({len(SHEETS)} bases x {LAST_CY-FIRST_CY+1} years "
138           f"x {len(set(SERIES.values()))} series)")
139     print(f" size    {OUT.stat().st_size:,} bytes")
140     print(f" source  {SOURCE} sha256 {digest[:16]}")
141     print(f" output  sha256 {sha256(OUT)[:16]}")
142
143
144 if __name__ == "__main__":
145     build()

```

fetch_sources.py

Downloads each source and verifies its checksum. Standard library only.

131 lines | sha256 b56244cc7ec1687007a28c9fb4eb5c76213eed5dfc464cb0603fc79de5878abf

```

1  """
2  Download every source file and verify its checksum against SOURCES.md.
3
4  The submission undertakes that any party may substitute official inputs and
5  re-run the model. This script is what makes that checkable: it fetches each
6  input from its publisher and reports whether the bytes match what was used.
7
8      python fetch_sources.py          report only -- nothing is written
9      python fetch_sources.py --download fetch into data/ and verify
10
11 A MISMATCH is not necessarily an error. The World Bank overwrites its workbook
12 in place, so the direct URL serves whatever vintage is current. When that
13 happens, re-run extract_inputs.py and compare inputs.csv: at the time of
14 writing, the later vintage produces the same 360 values.
15
16 Standard library only.
17 """
18 import hashlib
19 import sys
20 import urllib.request
21 from pathlib import Path
22
23 HERE = Path(__file__).parent
24 DATA = HERE / "data"
25 TIMEOUT = 180
26 UA = "Mozilla/5.0 (compatible; PC-submission-source-check/1.0)"
27
28 # name, local filename, url, expected sha256, note
29 SOURCES = [
30     ("World Bank -- Pink Sheet, annual prices",
31      "CMO-Historical-Data-Annual.xlsx",
32      "https://thedocs.worldbank.org/en/doc/74e8be41ceb20fa0da750cda2f6b9e4e-0050012026"
33      "/related/CMO-Historical-Data-Annual.xlsx",
34      "32b3821b0909cddb7e30dda54b20967e7c4e49258d76143ab0f6aab1f786290e9",
35      "the only file the model consumes; overwritten in place by the publisher"),
36
37     ("World Bank -- Pink Sheet, monthly prices",
38      "CMO-Historical-Data-Monthly.xlsx",
39      "https://thedocs.worldbank.org/en/doc/74e8be41ceb20fa0da750cda2f6b9e4e-0050012026"
40      "/related/CMO-Historical-Data-Monthly.xlsx",
41      "7902a77505ebdc5d202ce65f666c2ee1b04b626f042d7738ed3e6f7d112c8433",
42      "used only for the CY2026 extension, which is held as a constant"),
43
44     ("Productivity Commission -- Interim report",
45      "PC_interim_report.pdf",
46      "https://assets.pc.gov.au/2026-08/gst-reforms-interim.pdf",
47      "24a1fbb32989772c5266e5e8d16f0ab5563aa70c46b11ed9da042148c7fbf57a",
48      "CC BY 4.0"),
49
50     ("Productivity Commission -- Interim report chart data",
51      "PC_interim_chart_data.xlsx",
52      "https://assets.pc.gov.au/2026-08/gst-reforms-interim-chart-data.xlsx",
53      "4836a960025d688051d2e41ace951eb877cef502957a1f6986e4fdb6a6a5c6df",
54      "CC BY 4.0"),
55
56     ("Govt of Western Australia (DMPE) -- Major Commodities Data File 2025",
57      "WA_major_commodities_2025.xlsx",
58      "https://www.wa.gov.au/system/files/2026-05/2025-major-commodities-data-file.xlsx",
59      "11d329d7b721dd5e95fb07ceec59425ee6869c36ceb2e25e42cb47d68105864a",
60      "section 4.3 currency-basis test"),
61 ]
62
63
64 def sha256_bytes(b):
65     return hashlib.sha256(b).hexdigest()
66
67
68 def sha256_file(p):
69     return hashlib.sha256(p.read_bytes()).hexdigest()
70
71
72 def get(url):
73     req = urllib.request.Request(url, headers={"User-Agent": UA})
74     with urllib.request.urlopen(req, timeout=TIMEOUT) as r:
75         return r.read()
76
77
78 def main():
79     download = "--download" in sys.argv[1:]
80     DATA.mkdir(exist_ok=True)
81     ok = mismatch = failed = 0
82
83     print(f"{'Source':<62} {'Local':<10} {'Remote':<10}")
84     print("-" * 86)

```

```

85
86     for name, fname, url, expect, note in SOURCES:
87         local = DATA / fname
88         have = sha256_file(local) if local.exists() else None
89         local_tag = ("match" if have == expect else "differs") if have else "absent"
90
91         try:
92             blob = get(url)
93             live = sha256_bytes(blob)
94             remote_tag = "match" if live == expect else "new vintage"
95         except Exception as e:
96             live, remote_tag = None, f"error"
97             failed += 1
98
99         print(f"{name[:60]:<62} {local_tag:<10} {remote_tag:<10}")
100        print(f" {fname}")
101        print(f"  expected sha256 {expect}")
102        if have:
103            print(f"    local    sha256 {have}")
104        if live:
105            print(f"    remote  sha256 {live} ({len(blob):,} bytes)")
106            if live != expect:
107                print(f"      -> the publisher has issued a different file. Re-run "
108                    f"extract_inputs.py and diff inputs.csv before relying on it.")
109                mismatch += 1
110            else:
111                ok += 1
112        else:
113            print(f"    remote  could not be fetched: {url}")
114        if note:
115            print(f"    note: {note}")
116
117        if download and live:
118            local.write_bytes(blob)
119            print(f"      -> written to {local.relative_to(HERE)}")
120        print()
121
122    print("-" * 86)
123    print(f"{ok} verified, {mismatch} differ from the recorded vintage, "
124          f"{failed} could not be fetched.")
125    if not download:
126        print("Report only. Pass --download to write the files into data/.")
127    return 1 if failed else 0
128
129
130 if __name__ == "__main__":
131     raise SystemExit(main())

```