```python
# --- College Earnings Predictor: notebook bootstrap (Cell 1) ---
# Purpose: make the notebook portable, set constants, and prep artifact paths.

from __future__ import annotations
import os, sys, json, datetime as dt
from pathlib import Path

# ↳ 1) Find the repo root (df-jsx) no matter where Jupyter was launched
def find_repo_root(start: Path = Path.cwd()) -> Path:
    p = start.resolve()
    while p != p.parent:
        # heuristics: both server/routers and client/ exist in the project root
        if (p / "server" / "routers").exists() and (p / "client").exists():
            return p
        p = p.parent
    return start.resolve()

REPO_ROOT = find_repo_root()
print(f"[paths] REPO_ROOT = {REPO_ROOT}")

# Ensure repo root is importable if you want local modules
if str(REPO_ROOT) not in sys.path:
    sys.path.insert(0, str(REPO_ROOT))

# ↳ 2) Notebook-local data directories (raw downloads & scratch)
NB_DIR = REPO_ROOT / "notebooks" / "college_earnings"
RAW_DATA_DIR = NB_DIR / "data"
OUTPUTS_DIR = NB_DIR / "outputs"
RAW_DATA_DIR.mkdir(parents=True, exist_ok=True)
OUTPUTS_DIR.mkdir(parents=True, exist_ok=True)

# ↳ 3) Model identifiers & constants (edit as needed)
MODEL_NAME   = "college_earnings"
VERSION      = "v1_75k_5y"      # keep lowercase "k" to match artifact folder name
HORIZON      = "p6"             # ~6 years (proxy for 5–6y)
TARGET_USD   = 75_000           # threshold for ≥ $75k
RANDOM_SEED  = 42

# ↳ 4) Artifact directory (where the FastAPI route already looks)
ARTIFACT_DIR = REPO_ROOT / "server" / "routers" / "models" / MODEL_NAME / VERSION
ARTIFACT_DIR.mkdir(parents=True, exist_ok=True)

# Optional: training report filename to export later via nbconvert
REPORT_PDF_PATH = ARTIFACT_DIR / "training_report.pdf"

print(f"[paths] RAW_DATA_DIR     = {RAW_DATA_DIR}")
print(f"[paths] OUTPUTS_DIR      = {OUTPUTS_DIR}")
print(f"[paths] ARTIFACT_DIR     = {ARTIFACT_DIR}")
print(f"[paths] REPORT_PDF_PATH  = {REPORT_PDF_PATH}")

# ↳ 5) Helpers for consistent saving/logging
def save_json(data: dict, path: Path) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    path.write_text(json.dumps(data, indent=2))
```

```
    print(f"[save] {path.relative_to(REPO_ROOT)} ({path.stat().st_size} bytes)")

def utcnow() -> str:
    return dt.datetime.utcnow().replace(microsecond=0).isoformat() + "Z"

def log(msg: str) -> None:
    print(f"[{dt.datetime.now().strftime('%H:%M:%S')}] {msg}")

# ↳ 6) Planned artifact filenames (for later cells to use)
ENCODERS_JSON      = ARTIFACT_DIR / "encoders.json"
FIXED_EFFECTS_JSON = ARTIFACT_DIR / "fixed_effects.json"
RAND_STATE_JSON    = ARTIFACT_DIR / "random_state.json"
RAND_CIP_JSON      = ARTIFACT_DIR / "random_cip.json"
CALIB_JSON         = ARTIFACT_DIR / "calibration.json"
THRESHOLDS_JSON    = ARTIFACT_DIR / "thresholds.json"
METADATA_JSON      = ARTIFACT_DIR / "metadata.json"

# ↳ 7) (Optional) S3 settings if you later want to upload from the notebook
USE_S3_UPLOAD = bool(int(os.getenv("EARNINGS_USE_S3_UPLOAD", "0")))  # set 1 to enable
S3_BUCKET     = os.getenv("EARNINGS_S3_BUCKET", "your-bucket-name")
S3_PREFIX     = f"models/{MODEL_NAME}/{VERSION}/"

print(f"[s3] USE_S3_UPLOAD={USE_S3_UPLOAD}  bucket={S3_BUCKET}  prefix={S3_PREFIX}")

# Sanity ping
log("Notebook bootstrap complete. Proceed to data ingest…")
```

```
[paths] REPO_ROOT = /Users/sheilamcgovern/Desktop/Projects2025/df-jsx
[paths] RAW_DATA_DIR     = /Users/sheilamcgovern/Desktop/Projects2025/df-jsx/notebook
s/college_earnings/data
[paths] OUTPUTS_DIR      = /Users/sheilamcgovern/Desktop/Projects2025/df-jsx/notebook
s/college_earnings/outputs
[paths] ARTIFACT_DIR     = /Users/sheilamcgovern/Desktop/Projects2025/df-jsx/server/r
outers/models/college_earnings/v1_75k_5y
[paths] REPORT_PDF_PATH  = /Users/sheilamcgovern/Desktop/Projects2025/df-jsx/server/r
outers/models/college_earnings/v1_75k_5y/training_report.pdf
[s3] USE_S3_UPLOAD=False  bucket=your-bucket-name  prefix=models/college_earnings/v1_
75k_5y/
[19:49:18] Notebook bootstrap complete. Proceed to data ingest…
```

```
# --- Data Ingest (FoS + Institution join) ---

import zipfile
import pandas as pd
from pathlib import Path
import numpy as np

# 0) find FoS ZIP (you already have it)
fos_zips = sorted([p for p in RAW_DATA_DIR.glob("*.zip") if "Field-of-Study" in p.name
                   key=lambda p: p.stat().st_mtime, reverse=True)
assert fos_zips, "Put the 'Most-Recent-Cohorts-Field-of-Study_*.zip' in notebooks/col
fos_zip = fos_zips[0]
log(f"FoS ZIP: {fos_zip.name}")

# 1) read FoS CSV (pick the largest CSV inside)
with zipfile.ZipFile(fos_zip, "r") as zf:
```

```
    fos_csv = max([m for m in zf.namelist() if m.lower().endswith(".csv")],
                  key=lambda m: zf.getinfo(m).file_size)
    log(f"FoS CSV: {fos_csv}")
    with zf.open(fos_csv) as f:
        peek = pd.read_csv(f, nrows=5)
        fos_cols = {c.upper(): c for c in peek.columns}

# aliases for FoS
def has(colnames):  # returns original-case name if present
    for c in colnames:
        if c.upper() in fos_cols:
            return fos_cols[c.upper()]
    return None

fos_use = {
    "UNITID":      has(["UNITID"]),
    "INSTNM":      has(["INSTNM"]),
    "CIPCODE":     has(["CIPCODE"]),
    "CIPTITLE":    has(["CIPDESC","CIPTITLE"]),
    "CREDLEV":     has(["CREDLEV"]),
    "CREDDESC":    has(["CREDDESC"]),
    "CONTROL":     has(["CONTROL"]),
    "COUNT":       has(["COUNTOVERALL","IPEDSCOUNT1","IPEDSCOUNT2"]),
    "EARN_5YR":    has(["EARN_MDN_5YR"]),
    "EARN_4YR":    has(["EARN_MDN_4YR","EARN_MDN_HI_4YR"]),
    "EARN_2YR":    has(["EARN_MDN_HI_2YR"]),
    "EARN_1YR":    has(["EARN_MDN_HI_1YR","EARN_MDN_1YR"]),
}

earn_col = fos_use["EARN_5YR"] or fos_use["EARN_4YR"] or fos_use["EARN_2YR"] or fos_us
assert fos_use["UNITID"] and fos_use["CIPCODE"] and fos_use["CREDLEV"], "Missing UNIT
assert earn_col, "No earnings column found (looked for EARN_MDN_5YR/4YR/2YR/1YR)."

with zipfile.ZipFile(fos_zip, "r") as zf, zf.open(fos_csv) as f:
    usecols = [v for v in [fos_use["UNITID"], fos_use["INSTNM"], fos_use["CIPCODE"],
                           fos_use["CREDLEV"], fos_use["CREDDESC"], fos_use["CONTROL"]
                           fos_use["COUNT"], earn_col] if v]
    fos = pd.read_csv(f, usecols=usecols, low_memory=False)

# normalize FoS
rename_map = {}
if fos_use["UNITID"]:   rename_map[fos_use["UNITID"]] = "unitid"
if fos_use["INSTNM"]:   rename_map[fos_use["INSTNM"]] = "instnm"
if fos_use["CIPCODE"]:  rename_map[fos_use["CIPCODE"]] = "cip4"
if fos_use["CIPTITLE"]: rename_map[fos_use["CIPTITLE"]] = "ciptitle"
if fos_use["CREDLEV"]:  rename_map[fos_use["CREDLEV"]] = "credlev"
if fos_use["CREDDESC"]: rename_map[fos_use["CREDDESC"]] = "creddesc"
if fos_use["CONTROL"]:  rename_map[fos_use["CONTROL"]] = "control"
if fos_use["COUNT"]:    rename_map[fos_use["COUNT"]] = "countoverall"
rename_map[earn_col] = "earn_median"

fos = fos.rename(columns=rename_map)
fos["cip4"] = fos["cip4"].astype(str).str.replace(r"[^0-9]", "", regex=True).str[:4]
fos["earn_median"] = pd.to_numeric(fos["earn_median"], errors="coerce")

# 2) map labels used in your app
```

```
cred_map = {2: "Associate", 3: "Bachelor", 5: "Master", 7: "Professional", 6: "Doctora
fos["degree_level"] = fos["credlev"].map(cred_map)
control_map = {1: "Public", 2: "Private Nonprofit", 3: "Private For-profit"}



# CONTROL can be numeric (1/2/3) or strings ("Public", "Private Nonprofit", etc.)
ctrl_num = pd.to_numeric(fos["control"], errors="coerce")
ctrl_str = fos["control"].astype(str).str.strip().str.lower()

pp_from_num = np.where(ctrl_num == 1, "Public",
                np.where(ctrl_num.isin([2, 3]), "Private", np.nan))

pp = np.where(ctrl_num.notna(), pp_from_num,
     np.where(ctrl_str.str.contains("public", na=False), "Public",
     np.where(ctrl_str.str.contains("private", na=False), "Private", np.nan)))

fos["public_private"] = pd.Series(pp, index=fos.index)


# 3) load Institution "Most Recent" to get state
inst_zips = sorted([p for p in RAW_DATA_DIR.glob("*.zip")
                    if "Institution" in p.name or "Most-Recent-Institution" in p.name
                   key=lambda p: p.stat().st_mtime, reverse=True)
assert inst_zips, "Download the 'Most Recent' Institution ZIP to RAW_DATA_DIR as well
inst_zip = inst_zips[0]
log(f"Institution ZIP: {inst_zip.name}")

with zipfile.ZipFile(inst_zip, "r") as zf:
    inst_csv = max([m for m in zf.namelist() if m.lower().endswith(".csv")],
                   key=lambda m: zf.getinfo(m).file_size)
    log(f"Institution CSV: {inst_csv}")
    with zf.open(inst_csv) as f:
        inst_peek = pd.read_csv(f, nrows=5)
        inst_cols = {c.upper(): c for c in inst_peek.columns}

stabbr_col = inst_cols.get("STABBR") or inst_cols.get("STATE")
assert stabbr_col, "Could not find STABBR/STATE in institution file."
with zipfile.ZipFile(inst_zip, "r") as zf, zf.open(inst_csv) as f:
    inst = pd.read_csv(f, usecols=[inst_cols["UNITID"], stabbr_col])
inst = inst.rename(columns={inst_cols["UNITID"]: "unitid", stabbr_col: "state"})
inst["state"] = inst["state"].astype(str).str.upper().str.strip()

# 4) merge FoS + state
df = fos.merge(inst, on="unitid", how="left")

# 5) target & basic cohort filter
df["target"] = (df["earn_median"] >= TARGET_USD).astype("Int64")
if "countoverall" in df.columns:
    df = df[df["countoverall"].fillna(0).astype("Int64") >= 30].copy()

log(f"Rows after join/filter: {len(df):,}")
df.head(3)
```

```
[19:49:28] FoS ZIP: Most-Recent-Cohorts-Field-of-Study_04172025.zip
[19:49:28] FoS CSV: Most-Recent-Cohorts-Field-of-Study.csv
[19:49:30] Institution ZIP: Most-Recent-Cohorts-Institution_05192025.zip
[19:49:30] Institution CSV: Most-Recent-Cohorts-Institution_05192025.csv
[19:49:30] Rows after join/filter: 39,051
```

	unitid	instnm	control	cip4	ciptitle	credlev	creddesc	countoverall	earn_m
29	100654.0	Alabama A & M University	Public	1410	Electrical, Electronics and Communications Eng...	3	Bachelor's Degree	33.0	9(
30	100654.0	Alabama A & M University	Public	1419	Mechanical Engineering.	3	Bachelor's Degree	41.0	8:
31	100654.0	Alabama A & M University	Public	1499	Engineering, Other.	5	Master's Degree	30.0	

```python
# --- Modeling + Calibration + Artifact Export (one cell) ---

import numpy as np, pandas as pd
from sklearn.model_selection import GroupShuffleSplit
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, brier_score_loss

rng = np.random.default_rng(RANDOM_SEED)

# 0) Minimal sanity
req = ["degree_level","state","cip4","public_private","target"]
missing_cols = [c for c in req if c not in df.columns]
assert not missing_cols, f"Missing required columns: {missing_cols}"
dfm = df.dropna(subset=["degree_level","state","cip4"]).copy()
dfm["target"] = dfm["target"].astype(int)

# 1) Fixed + group features
# use countoverall as a light size proxy (optional)
if "countoverall" in dfm.columns:
    q = pd.qcut(dfm["countoverall"].fillna(0), q=4, duplicates="drop")
    dfm["size_bin"] = q.astype(str)
else:
    dfm["size_bin"] = "NA"

fixed_feats  = ["degree_level","public_private","size_bin"]
group_feats  = ["state","cip4"]

# 2) Train/val split grouped by institution to reduce leakage
groups = dfm.get("unitid", pd.Series(range(len(dfm))))
gss = GroupShuffleSplit(n_splits=1, test_size=0.2, random_state=RANDOM_SEED)
(train_idx, val_idx), = gss.split(dfm, groups=groups)
tr, va = dfm.iloc[train_idx].copy(), dfm.iloc[val_idx].copy()
y_tr, y_va = tr["target"].values, va["target"].values

# 3) Baseline fixed-effects logistic (sanity)
Xtr_fixed = pd.get_dummies(tr[fixed_feats], drop_first=False)
```

```
Xva_fixed = pd.get_dummies(va[fixed_feats], drop_first=False).reindex(columns=Xtr_fixe

clf_fixed = LogisticRegression(penalty="l2", C=1.0, max_iter=300, solver="liblinear",
clf_fixed.fit(Xtr_fixed, y_tr)
p_va_fixed = clf_fixed.predict_proba(Xva_fixed)[:,1]
print(f"AUC (fixed):  {roc_auc_score(y_va, p_va_fixed):.3f} | Brier: {brier_score_loss

# 4) GLMM-ish with group dummies (state, cip4) + ridge-like shrink
Xtr = pd.get_dummies(tr[fixed_feats + group_feats], drop_first=False)
Xva = pd.get_dummies(va[fixed_feats + group_feats], drop_first=False).reindex(columns=

clf = LogisticRegression(penalty="l2", C=0.5, max_iter=500, solver="liblinear", randor
clf.fit(Xtr, y_tr)
p_va = clf.predict_proba(Xva)[:,1]
print(f"AUC (fx+grp): {roc_auc_score(y_va, p_va):.3f} | Brier: {brier_score_loss(y_va,

# 5) Extract coefficients and split into fixed vs group parts
coef = pd.Series(clf.coef_[0], index=Xtr.columns)
intercept = float(clf.intercept_[0])

fixed_cols = Xtr_fixed.columns
group_state_cols = [c for c in Xtr.columns if c.startswith("state_")]
group_cip_cols   = [c for c in Xtr.columns if c.startswith("cip4_")]

fixed_coefs = coef.loc[fixed_cols].to_dict()

# 6) Empirical-Bayes-ish shrinkage for random intercepts by group size
def group_total(s, key_col, n_col):
    if n_col not in s.columns: return s.groupby(key_col).size().rename("n")
    return s.groupby(key_col)[n_col].sum().rename("n")

state_sizes = group_total(tr, "state", "countoverall")
cip_sizes   = group_total(tr, "cip4", "countoverall")

def shrink(effect, n, k=200.0):
    w = float(n) / float(n + k) if pd.notnull(n) else 0.0
    return float(w * effect)

random_state = {}
for c in group_state_cols:
    st = c.split("state_",1)[1]
    eff = coef[c]
    n = state_sizes.get(st, 0.0)
    random_state[st] = shrink(eff, n)

random_cip = {}
for c in group_cip_cols:
    cp = c.split("cip4_",1)[1]
    eff = coef[c]
    n = cip_sizes.get(cp, 0.0)
    random_cip[cp] = shrink(eff, n)

# 7) Simple Platt scaling on validation probs
eps = 1e-8
def logit(p):
    p = np.clip(p, eps, 1-eps)
```

```
    return np.log(p/(1-p))

from sklearn.linear_model import LogisticRegression as LR
platt = LR(penalty=None, max_iter=1000, solver="lbfgs")

platt.fit(logit(p_va).reshape(-1,1), y_va)

def platt_calibrate(p):
    z = logit(np.asarray(p)).reshape(-1,1)
    return platt.predict_proba(z)[:,1]

p_va_cal = platt_calibrate(p_va)
print(f"AUC (cal):    {roc_auc_score(y_va, p_va_cal):.3f} | Brier: {brier_score_loss(y

# 8) Thresholds & encoders
low_cut, high_cut = 0.33, 0.66

encoders = {
    "degree_levels": sorted(dfm["degree_level"].dropna().unique().tolist()),
    "states": sorted(dfm["state"].dropna().unique().tolist()),
    "cip4": sorted(dfm["cip4"].dropna().unique().tolist()),
    "public_private": sorted(dfm["public_private"].dropna().unique().tolist()),
    "size_bins": sorted(dfm["size_bin"].dropna().unique().tolist()),
    "fixed_feature_columns": list(fixed_cols)
}

fixed_effects = {
    "intercept": intercept,
    "coefficients": fixed_coefs
}

calibration = {
    "type": "platt",
    "coef": float(platt.coef_[0][0]),
    "intercept": float(platt.intercept_[0]),
    "note": "Input is raw model probability logit."
}

thresholds = {"low": low_cut, "high": high_cut}

metadata = {
    "model": MODEL_NAME,
    "version": VERSION,
    "target": f"Pr(median earnings ≥ ${TARGET_USD:,} at ~5 years; FoS 5-year median us
    "trained_at": utcnow(),
    "notes": [
        "Fixed effects: degree_level, public_private, size_bin.",
        "Group effects: state, cip4 via L2 + empirical shrink.",
        "Calibration: Platt on validation grouped by UNITID split."
    ],
    "counts": {
        "train": int(len(tr)),
        "valid": int(len(va)),
        "pos_rate_train": float(np.mean(y_tr)),
        "pos_rate_valid": float(np.mean(y_va))
    },
```

```
    "metrics_valid": {
        "auc_fixed": float(roc_auc_score(y_va, p_va_fixed)),
        "auc_fx_grp": float(roc_auc_score(y_va, p_va)),
        "auc_cal":    float(roc_auc_score(y_va, p_va_cal)),
        "brier_cal":  float(brier_score_loss(y_va, p_va_cal))
    }
}

# 9) Save artifacts
save_json(encoders, ENCODERS_JSON)
save_json(fixed_effects, FIXED_EFFECTS_JSON)
save_json(random_state, RAND_STATE_JSON)
save_json(random_cip, RAND_CIP_JSON)
save_json(calibration, CALIB_JSON)
save_json(thresholds, THRESHOLDS_JSON)
save_json(metadata, METADATA_JSON)

log("Artifacts written. You can now hit /api/predictors/infer with real scores.")
```

```
AUC (fixed):  0.711 | Brier: 0.167
AUC (fx+grp): 0.893 | Brier: 0.113
AUC (cal):    0.893 | Brier: 0.113
[save] server/routers/models/college_earnings/v1_75k_5y/encoders.json (5228 bytes)
[save] server/routers/models/college_earnings/v1_75k_5y/fixed_effects.json (622 byte
s)
[save] server/routers/models/college_earnings/v1_75k_5y/random_state.json (1634 byte
s)
[save] server/routers/models/college_earnings/v1_75k_5y/random_cip.json (10083 bytes)
[save] server/routers/models/college_earnings/v1_75k_5y/calibration.json (138 bytes)
[save] server/routers/models/college_earnings/v1_75k_5y/thresholds.json (33 bytes)
[save] server/routers/models/college_earnings/v1_75k_5y/metadata.json (718 bytes)
[19:49:37] Artifacts written. You can now hit /api/predictors/infer with real scores.
```

```
# Build CIP4 -> label map (most common title per code)
from collections import Counter
import json, os

# ensure codes are the same normalized 4-digit strings you use in artifacts
fos["_cip4"] = fos["cip4"].astype(str).str.replace(r"\D", "", regex=True).str[:4]

label_map = {}
for code, titles in fos.groupby("_cip4")["ciptitle"]:
    title = Counter(titles.dropna().str.strip()).most_common(1)[0][0]
    label_map[code] = title

# optional: hand-fix a couple common names
label_map.setdefault("1101", "Computer Science")
label_map.setdefault("5203", "Accounting")

labels_path = ARTIFACT_DIR / "cip4_labels.json"
labels_path.write_text(json.dumps(label_map, ensure_ascii=False, indent=2))
print("[save]", labels_path, f"({os.path.getsize(labels_path)} bytes)")
```

```
[save] /Users/sheilamcgovern/Desktop/Projects2025/df-jsx/server/routers/models/colleg
e_earnings/v1_75k_5y/cip4_labels.json (21378 bytes)
```