Python Cheatsheet

MATH60033A · Session 07 · Four papers, one toolkit

Thierry Warin, PhD · HEC Montréal · A2026

Everything you need to type, in one place: today's set-up in a folder of your own, every name from sessions 1 to 7, and every step of the four papers' scripts.

Session 07: a folder of your own

Today is not worked in the course repository. Make an empty folder — `Desktop/session-07` — and open it in VS Codium (**File > Open Folder**). Then, in its terminal (**View > Terminal**):

```
# macOS
open /Applications/Python\ 3*/Install\ Certificates.command   # python.org Python only, once
python3 -m venv .venv
source .venv/bin/activate
pip install pandas statsmodels scikit-learn matplotlib
curl -LO https://warint.github.io/quantitative-methods/session-07-scripts.zip
unzip session-07-scripts.zip
python check_setup.py

# Linux
python3 -m venv .venv
source .venv/bin/activate
pip install pandas statsmodels scikit-learn matplotlib
curl -LO https://warint.github.io/quantitative-methods/session-07-scripts.zip
unzip session-07-scripts.zip
python check_setup.py

# Windows (PowerShell)
py -m venv .venv
.venv\Scripts\Activate.ps1
pip install pandas statsmodels scikit-learn matplotlib
Invoke-WebRequest https://warint.github.io/quantitative-methods/session-07-scripts.zip
 ↪  -OutFile session-07-scripts.zip
Expand-Archive session-07-scripts.zip -DestinationPath .
python check_setup.py
```

Every line of the check should say `OK`. Then **Python: Select Interpreter** (`Ctrl`/`Cmd` + `Shift` + `P`) and choose this folder's `.venv`. Run a script with the **Run** triangle, or `python r1_fraiberger_2021.py`; one step at a time, select its lines and press `Shift` + `Enter`.

No `requirements.txt` and no `qmib` today: the four packages above are the whole toolkit, and each script reads its paper's data from the course site — you need the internet.

The four things you always do

```
import pandas as pd

# 1 - build a DataFrame
df = pd.DataFrame({"country": ["FR", "DE", "IT"],
                   "gdp_pc_eur": [37_000, 46_000, 32_000]})

# 2 - look at what is in it
```

```
df.head(); df.shape; df.dtypes; df.info(); df.describe()

# 3 - read a file in: from your folder, or straight from a web address
df = pd.read_csv("data.csv")
DATA = "https://warint.github.io/quantitative-methods/data/session-07/"
d = pd.read_csv(DATA + "paper1_fraiberger_2021.csv")

# 4 - write a CSV out
df.to_csv("out.csv", index=False)   # index=False, or you get a stray column
```

A package missing? With `(.venv)` showing: `pip install` and its name — `pip install openpyxl`, say. Never `sudo`, never outside the environment.

What you have learned so far

Sessions 1 to 7, in the order you met them. Every line below has run on the course data.

Session 1 · Syllabus and Europe 2031

The workstation itself — the editor, the environment, and loading a course dataset with one line. That is section 1 above.

Session 2 · Exploratory data analysis

```
# -- Load and shape -----------------------------------------------
core = qmib.load("core")            # a course dataset, by name
core = pd.read_parquet("data/spine/core.parquet")
d = core.dropna(subset=["gdp_pc_eur", "productivity_idx"])
# subset= matters: without it a row missing ANY column goes
both = left.merge(right, on=["geo", "time"], how="inner")
n_rows, n_cols = core.shape

# -- Describe -----------------------------------------------------
gdp.mean(), gdp.median()      # the gap between them is the finding
gdp.std(ddof=1), gdp.var(ddof=1)  # ddof=1 divides by n-1
gdp.quantile([0.25, 0.5, 0.75])    # IQR is the third minus the first
from scipy import stats
stats.trim_mean(gdp, 0.05)        # mean after cutting 5% off each tail
```

Session 3 · Regression diagnostics

```
# -- Fit ----------------------------------------------------------
import statsmodels.formula.api as smf
fit = smf.ols("gdp_pc_eur ~ productivity_idx", data=d).fit()
# read the ~ as "explained by"; the intercept is added for you
fit = model.fit()                 # .fit() does the arithmetic

# -- Read the fit -------------------------------------------------
fit.params                        # the coefficients, in units of y
fit.rsquared                      # share of variance explained
fit.resid, fit.fittedvalues       # what is left, and what was predicted
fit.rsquared_adj                  # penalised for each extra predictor
np.sqrt(fit.mse_resid)            # the RSE, in the units of y
print(qmib.regtable([m1, m2], names=["simple", "+ investment"],
                    order=["productivity_idx", "gfcf_meur", "Intercept"]))
# one column per model, stars, SEs beneath, fit stats at the foot.
# .as_latex() for a paper, .as_html() for a slide.
```

```
# -- Diagnose ----------------------------------------------------
influence = fit.get_influence()    # the whole diagnostic bundle
leverage = influence.hat_matrix_diag        # sums to p, always
cooks_d = influence.cooks_distance[0]       # screen at 4/n
influence.resid_studentized_internal        # on a common scale

# -- Choose and validate -----------------------------------------
fit.aic, fit.bic                  # lower is better, and only relatively

# -- Plot --------------------------------------------------------
import matplotlib.pyplot as plt
plt.scatter(d["productivity_idx"], d["gdp_pc_eur"], s=12, alpha=0.6)
plt.xlabel("Productivity index"); plt.ylabel("GDP per capita (EUR)")
# say the units, every time
plt.axhline(0, color="black", lw=1)
plt.axvline(2 * 2 / len(d), color="red", ls="--")   # the 2p/n screen
plt.stem(cooks_d)                 # one spike per observation
```

Session 4 · Logistic regression

```
# -- Load and shape ----------------------------------------------
X = pd.get_dummies(d[["grade"]], drop_first=True)

# -- Describe ----------------------------------------------------
pd.crosstab(d["grade"], d["default"])
d["default"].value_counts(normalize=True)    # how unbalanced is it

# -- Fit ---------------------------------------------------------
fit = smf.logit("default ~ fico + dti", data=d).fit()
from statsmodels.miscmodels.ordinal_model import OrderedModel
fit = OrderedModel(y, X, distr="logit").fit(method="bfgs")
import statsmodels.api as sm
fit = sm.MNLogit(y, sm.add_constant(X)).fit()
X = sm.add_constant(X)            # the intercept, with no formula

# -- Read the fit ------------------------------------------------
p = fit.predict(newdata)          # probability e^xb/(1+e^xb), not a class
fit.get_prediction(newdata).summary_frame()  # with an interval
fit.params                        # log-odds, as fitted
np.exp(fit.params)                # odds ratios: 1.65 = odds 65% higher

# -- Choose and validate -----------------------------------------
lr = 2 * (full.llf - reduced.llf)  # the likelihood-ratio statistic
stats.chi2.sf(lr, df=2)           # its p-value

# -- Plot --------------------------------------------------------
fig, axes = plt.subplots(1, 2, figsize=(7.4, 2.9))
```

Session 5 · Regularisation

```
# -- Load and shape ----------------------------------------------
from sklearn.preprocessing import StandardScaler
Z = StandardScaler().fit(Xtr).transform(Xtr)   # fit on TRAIN only
pipe = make_pipeline(StandardScaler(), Lasso(alpha=0.8)).fit(X, y)
# standardise first; the scaler never sees the held-out fold
```

```
# -- Fit ------------------------------------------------------------
Ridge(alpha=10)                     # alpha is lambda; shrinks, keeps all
Lasso(alpha=0.8)                    # sets weak ones to exactly 0
ElasticNet(alpha=1.0, l1_ratio=0.5)  # l1_ratio: 1 = lasso, 0 = ridge

# -- Read the fit ---------------------------------------------------
model.coef_, model.alpha_          # scikit-learn's spelling of .params

# -- Choose and validate --------------------------------------------
LassoCV(cv=5).fit(Ztr, ytr).alpha_      # lambda by cross-validation
ElasticNetCV(l1_ratio=[.1, .5, .9, 1], cv=5)  # and the mix too
alphas, coefs, _ = lasso_path(Z, y)     # the whole coefficient path
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, random_state=7)
KFold(n_splits=5, shuffle=True, random_state=7)
mean_squared_error(yte, model.predict(Zte)) ** 0.5    # RMSE
alphas = np.logspace(-3, 2, 100)   # a grid even in log-lambda
```

Session 6 · Panel data and interactions

```
# -- Fit ------------------------------------------------------------
from linearmodels.panel import PanelOLS
fit = PanelOLS.from_formula("y ~ x + EntityEffects", data=panel).fit()
RandomEffects.from_formula("y ~ 1 + x", data=panel).fit()
smf.ols("price ~ living_area + I(living_area**2)", data=d).fit()
smf.ols("wage ~ age * female", data=d).fit()
# a * b expands to a + b + a:b

# -- Choose and validate --------------------------------------------
big.compare_f_test(small)          # nested models only
np.linalg.pinv(v_fe - v_re)        # for the Hausman statistic
```

Session 7 · Four papers, one toolkit

```
# -- Load and shape -------------------------------------------------
DATA = "https://warint.github.io/quantitative-methods/data/session-07/"
firms = pd.read_csv(DATA + "paper4_topalova_khandelwal_2011.csv")
y = np.log(d.fdi_stock_musd)       # log FDI: a skewed outcome, read in %
pd.Categorical(d.event, categories=["shrinking", "continuing", "growing"], ordered=True)
d["tfp_third"] = pd.qcut(d["tfp1000"], 3, labels=["low", "middle", "high"])
d.loc[d["foreign"] == 1, "owner"] = "foreign"   # only those rows change
dat = groups[groups["event"].isin(order)]   # only the three ordered events
dat = groups[groups["event"].isin(order)].copy()   # then add columns safely

# -- Describe -------------------------------------------------------
d["iso"].nunique()                 # 25 countries

# -- Plot -----------------------------------------------------------
plt.savefig("p4_resid_vs_fitted.png", dpi=110, bbox_inches="tight")   # saved in your folder
```

The four papers, in the course's code

No new method today: each paper's data goes into the models of sessions 3 to 5, with the same lines as in their lectures — only the data and the variable names change. Below is every step of the four scripts in the zip, run in your session-07 folder. The guides in `07-replication-workshop/02-practice/replications/` explain each step and show what you should see.

The imports, once for all four

```
# each script opens with the imports it needs; all of them:
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
from statsmodels.miscmodels.ordinal_model import OrderedModel
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV, ElasticNetCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

Paper 1 · Fraiberger et al. (2021) · news sentiment and stock returns

OLS, then the binary, ordinal and multinomial logits · `starter/r1_fraiberger_2021.py`

```
# Step 1 · Load the data: one row is a country-day
d = pd.read_csv(DATA + "paper1_fraiberger_2021.csv")
d = d.dropna(subset=["F0_index", "L1_sent1_best", "L1_vix"])
d["up"] = (d["F0_index"] > 0).astype(int)            # 1 if the market rose that day
print(f"{len(d):,} country-days, {d['iso'].nunique()} countries, share of up days
 ↪  {d['up'].mean():.3f}")

# Step 2 · Three lines of Python (session 03): today's return on yesterday's news
model = smf.ols("F0_index ~ L1_sent1_best", data=d).fit()
print(model.params)

# Step 3 · How much did the line explain? (session 03)
print(f"R-squared = {model.rsquared:.3f}")

# Step 4 · Two predictors: what "controlling for" means (session 03)
model2 = smf.ols("F0_index ~ L1_sent1_best + L1_vix", data=d).fit()
print(model2.params)

# Step 5 · The residual plot is the instrument (session 03)
plt.scatter(model2.fittedvalues, model2.resid, s=12, alpha=0.6)
plt.axhline(0, color="black", lw=1)
plt.xlabel("Fitted values"); plt.ylabel("Residuals")
plt.show()

# Step 6 · Binary logit: log-odds (session 04, LR 6) - does the market go up today?
m2 = smf.logit("up ~ L1_sent1_best", data=d).fit(disp=0)
print(m2.params.round(4))

# Step 7 · Binary logit: odds (session 04, LR 7)
print(np.exp(m2.params["L1_sent1_best"]).round(4))

# Step 8 · Binary logit: predicted probability (session 04, LR 8)
print(m2.predict(pd.DataFrame({"L1_sent1_best": [-1, 0, 1]})).round(3))

# Step 9 · Ordinal logit (session 04, OLR 3) - the day's return in thirds: down < flat < up
d["day"] = pd.qcut(d["F0_index"], 3, labels=["down", "flat", "up"])
order = ["down", "flat", "up"]
```

```
d["day_c"] = pd.Categorical(d["day"], categories=order, ordered=True)
om = OrderedModel(d["day_c"], d[["L1_sent1_best", "L1_vix"]],
 ↪  distr="logit").fit(method="bfgs", disp=0)
print(om.summary())

# Step 10 · Ordinal logit: log-odds and odds (session 04, OLR 4)
print(om.params[["L1_sent1_best", "L1_vix"]].round(3))
print(np.exp(om.params[["L1_sent1_best", "L1_vix"]]).round(3))

# Step 11 · Ordinal logit: probabilities (session 04, OLR 5)
new = pd.DataFrame({"L1_sent1_best": [-1, 1], "L1_vix": [20, 20]})
print(pd.DataFrame(np.asarray(om.predict(new)), columns=order, index=["bad news", "good
 ↪  news"]).round(3))

# Step 12 · Multinomial logit (session 04, MLR 3) - the same three days, without their order
levels = ["flat", "down", "up"]
d["day_m"] = pd.Categorical(d["day"], categories=levels).codes
X = d[["L1_sent1_best"]].astype(float)
X["L1_vix"] = d["L1_vix"].astype(float)
X = sm.add_constant(X)
mn = sm.MNLogit(d["day_m"], X).fit(disp=0)
print(mn.summary().tables[1])          # the coefficient table

# Step 13 · Multinomial logit: log-odds and relative risk ratios (session 04, MLR 4)
print(pd.concat({"log-odds": mn.params, "RRR": np.exp(mn.params)}, axis=1)
      .set_axis(["log-odds: down", "log-odds: up", "RRR: down", "RRR: up"], axis=1).round(3))

# Step 14 · Multinomial logit: probabilities (session 04, MLR 5)
grid = pd.DataFrame({"const": 1.0, "L1_sent1_best": [-2, -1, 0, 1, 2], "L1_vix": 20.0})
out = pd.DataFrame(mn.predict(grid).values, columns=levels, index=grid["L1_sent1_best"])
print(out.round(3))
```

Paper 2 · Saganowski et al. (2019) · what happens to a group next

the binary, ordinal and multinomial logits · `starter/r2_saganowski_2019.py`

```
# Step 1 · Load the groups, and what happened to each one next week
FILE = DATA + "paper2_saganowski_2019.csv"     # the authors' file, unchanged
raw = pd.read_csv(FILE)
print(raw.shape)
print(raw["event_type"].value_counts(normalize=True).round(3))   # the base rates

# Step 2 · Three features a person can read
groups = pd.DataFrame({
    "event": raw["event_type"],
    "log2_size": np.log2(raw["g1_size"]),            # +1 = the group is twice as big
    "avg_degree": raw["g1_avg_group_degree_total"],  # ties per member, inside the group
    "aging": raw["g1_IlhanAging"]})                  # how long its members have been around
print(groups[["log2_size", "avg_degree", "aging"]].median().round(2))

# Step 3 · Binary logit: log-odds (session 04, LR 6)
groups["dissolve"] = (groups["event"] == "dissolving").astype(int)
m2 = smf.logit("dissolve ~ log2_size", data=groups).fit(disp=0)
print(m2.params.round(4))

# Step 4 · Binary logit: odds (session 04, LR 7)
```

```
print(np.exp(m2.params["log2_size"]).round(4))

# Step 5 · Binary logit: predicted probability (session 04, LR 8)
print(m2.predict(pd.DataFrame({"log2_size": [2, 3, 4]})).round(3))

# Step 6 · Ordinal logit: the code (session 04, OLR 3)
order = ["shrinking", "continuing", "growing"]
dat = groups[groups["event"].isin(order)].copy()
dat["event_c"] = pd.Categorical(dat["event"], categories=order, ordered=True)
om = OrderedModel(dat["event_c"], dat[["log2_size", "avg_degree", "aging"]],
 ↪  distr="logit").fit(method="bfgs", disp=0)
print(om.summary())

# Step 7 · Ordinal logit: log-odds -> odds (session 04, OLR 4)
print(om.params[["log2_size", "avg_degree", "aging"]].round(3))
print(np.exp(om.params[["log2_size", "avg_degree", "aging"]]).round(3))

# Step 8 · Ordinal logit: probabilities (session 04, OLR 5)
new = pd.DataFrame({"log2_size": [2, 4], "avg_degree": [2.7, 2.7], "aging": [8.5, 8.5]})
print(pd.DataFrame(np.asarray(om.predict(new)), columns=order, index=["4 members", "16
 ↪  members"]).round(3))

# Step 9 · Multinomial logit: the code (session 04, MLR 3)
levels = ["dissolving", "shrinking", "growing", "continuing"]
ml = groups[groups["event"].isin(levels)].copy()
ml["event_c"] = pd.Categorical(ml["event"], categories=levels).codes
X = ml[["log2_size", "avg_degree", "aging"]].astype(float)
X = sm.add_constant(X)
mn = sm.MNLogit(ml["event_c"], X).fit(disp=0)
print(mn.summary().tables[1])          # the coefficient table

# Step 10 · Multinomial logit: log-odds -> relative risk ratios (session 04, MLR 4)
print(pd.concat({"log-odds": mn.params, "RRR": np.exp(mn.params)}, axis=1)
      .set_axis(["log-odds: shrinking", "log-odds: growing", "log-odds: continuing",
                 "RRR: shrinking", "RRR: growing", "RRR: continuing"], axis=1).round(3))

# Step 11 · Multinomial logit: probabilities (session 04, MLR 5)
grid = pd.DataFrame({"const": 1.0, "log2_size": [2, 3, 4, 5],
                     "avg_degree": 2.7, "aging": 8.5})
out = pd.DataFrame(mn.predict(grid).values, columns=levels, index=grid["log2_size"])
print(out.round(3))
```

Paper 3 · Blonigen and Piger (2014) · the determinants of FDI

OLS, then ridge, lasso and elastic net, compared on held-out rows · `starter/r3_blonigen_piger_2014.py`

```
# Step 1 · Load the data: one row per country pair, log FDI, 56 candidates
fdi = pd.read_csv(DATA + "paper3_blonigen_piger_2014.csv")
YCOL = "log_fdi"
XCOLS = [c for c in fdi.columns if c not in ("parent", "host", "year", "fdi_stock_musd")]
d = fdi[fdi.fdi_stock_musd > 0].dropna(subset=XCOLS).copy()   # positive FDI, complete rows
d["log_fdi"] = np.log(d["fdi_stock_musd"])
for v in ["parent_gdp", "host_gdp", "distance"]:               # gravity variables in logs
    d[v] = np.log(d[v])
print(f"country pairs: {len(d)}   candidate predictors: {len(XCOLS)}")
```

```python
# Step 2 · OLS: three lines of Python (session 3)
model = smf.ols("log_fdi ~ host_gdp", data=d).fit()
print(model.params)

# Step 3 · How much did the line explain?
print(f"R-squared = {model.rsquared:.3f}")

# Step 4 · Two predictors: what "controlling for" means
model2 = smf.ols("log_fdi ~ host_gdp + distance", data=d).fit()
print(model2.params)
print(f"R-squared = {model2.rsquared:.3f}")

# Step 5 · Ridge (session 5)
Xc, yc = d[XCOLS].astype(float), d[YCOL].astype(float)
ridge = make_pipeline(StandardScaler(),                          # same scale for all
                      RidgeCV(alphas=np.logspace(-3, 3, 60))   # try 60 values of lambda
                      ).fit(Xc, yc)
print(f"lambda = {ridge[-1].alpha_:.2f}")
print(f"ridge kept {np.sum(ridge[-1].coef_ != 0)} of {len(XCOLS)} - it never drops a
 ↪  predictor")

# Step 6 · Lasso
lasso = make_pipeline(StandardScaler(),
                      LassoCV(cv=5, random_state=0, max_iter=50_000)   # lambda by 5-fold CV
                      ).fit(Xc, yc)
kept = [c for c, b in zip(XCOLS, lasso[-1].coef_) if b != 0]
print(f"lambda = {lasso[-1].alpha_:.4f}")
print(f"lasso kept {len(kept)} of {len(XCOLS)}")

# Step 7 · Elastic net
enet = make_pipeline(StandardScaler(),
                     ElasticNetCV(l1_ratio=0.5,            # half lasso, half ridge
                                  cv=5, random_state=0, max_iter=50_000)
                     ).fit(Xc, yc)
kept = [c for c, b in zip(XCOLS, enet[-1].coef_) if b != 0]
print(f"lambda = {enet[-1].alpha_:.4f}")
print(f"elastic net kept {len(kept)} of {len(XCOLS)}")

# Step 8 · Which one predicts best? Fit on 70% of the pairs, score on 30%
Xtr, Xte, ytr, yte = train_test_split(d[XCOLS], d[YCOL], test_size=0.3, random_state=7)
def test_rmse(model):                    # fit on 70% of rows, score on the other 30%
    pipe = make_pipeline(StandardScaler(), model).fit(Xtr, ytr)   # standardise first
    kept = int(np.sum(np.abs(pipe[-1].coef_) > 1e-8))             # non-zero coefficients
    return kept, mean_squared_error(yte, pipe.predict(Xte)) ** 0.5
results = {
    "OLS, all predictors": test_rmse(LinearRegression()),
    "ridge":       test_rmse(RidgeCV(alphas=np.logspace(-3, 3, 60))),   # lambda by CV
    "lasso":       test_rmse(LassoCV(cv=5, random_state=0, max_iter=50_000)),
    "elastic net": test_rmse(ElasticNetCV(l1_ratio=0.5, cv=5, random_state=0,
     ↪  max_iter=50_000)),
}
res = pd.DataFrame(results, index=["predictors kept", "test RMSE"]).T.astype({"predictors
 ↪  kept": int})
print(res.round(3))
```

Paper 4 · Topalova and Khandelwal (2011) · tariffs and productivity

OLS, the three logits, then ridge, lasso and elastic net · `starter/r4_topalova_khandelwal_2011.py`

```
# Step 1 · Load the firms: one row is one firm in one year, 1989-1996
firms = pd.read_csv(DATA + "paper4_topalova_khandelwal_2011.csv")
cols = ["tfp1000", "lagtariff", "laginputariff", "prgroup", "govown",
        "foreign", "medium", "small", "age"]
d = firms[firms["yr"] < 1997].dropna(subset=cols)       # no gaps in what we use
print(f"{len(d):,} firm-years, {d['companyname'].nunique():,} firms")
print(d[["tfp1000", "lagtariff", "laginputariff"]].describe().round(3))

# Step 2 · Three lines of Python (session 3): productivity on the output tariff
model = smf.ols("tfp1000 ~ lagtariff", data=d).fit()
print(model.params)

# Step 3 · How much did the line explain? (session 3)
print(f"R-squared = {model.rsquared:.3f}")

# Step 4 · Two predictors: what "controlling for" means (session 3)
model2 = smf.ols("tfp1000 ~ lagtariff + laginputariff", data=d).fit()
print(model2.params)
print(f"R-squared = {model2.rsquared:.3f}")

# Step 5 · Residuals vs fitted (session 3): structure the straight line missed?
fit = model2
plt.scatter(fit.fittedvalues, fit.resid, s=12, alpha=0.6)
plt.axhline(0, color="black", lw=1)
plt.xlabel("Fitted values"); plt.ylabel("Residuals")
plt.savefig("p4_resid_vs_fitted.png", dpi=110, bbox_inches="tight")   # saved in your folder
plt.show()

# Step 6 · Binary logit: log-odds (session 4, LR 6) - is the firm more productive than the
 ↪  median firm?
d["high_tfp"] = (d["tfp1000"] > d["tfp1000"].median()).astype(int)   # 1 = above the median
m2 = smf.logit("high_tfp ~ lagtariff", data=d).fit(disp=0)
print(m2.params.round(4))

# Step 7 · Binary logit: odds (session 4, LR 7)
print(np.exp(m2.params["lagtariff"]).round(4))

# Step 8 · Binary logit: predicted probability (session 4, LR 8)
print(m2.predict(pd.DataFrame({"lagtariff": [0.4, 0.7, 1.0]})).round(3))   # inside the data:
 ↪  0.1 to 3.5

# Step 9 · Ordinal logit (session 4, OLR 3) - productivity in thirds: low < middle < high
d["tfp_third"] = pd.qcut(d["tfp1000"], 3, labels=["low", "middle", "high"])
order = ["low", "middle", "high"]
d["tfp_c"] = pd.Categorical(d["tfp_third"], categories=order, ordered=True)
om = OrderedModel(d["tfp_c"], d[["lagtariff", "laginputariff", "age"]],
 ↪  distr="logit").fit(method="bfgs", disp=0)
print(om.summary())

# Step 10 · Ordinal logit: log-odds and odds (session 4, OLR 4)
print(om.params[["lagtariff", "laginputariff", "age"]].round(3))
print(np.exp(om.params[["lagtariff", "laginputariff", "age"]]).round(3))
```

```
# Step 11 · Ordinal logit: probabilities (session 4, OLR 5)
new = pd.DataFrame({"lagtariff": [1.0, 0.4], "laginputariff": [0.3, 0.3], "age": [20, 20]})
print(pd.DataFrame(np.asarray(om.predict(new)), columns=order, index=["tariff 1.0", "tariff
 ↪  0.4"]).round(3))

# Step 12 · Multinomial logit (session 4, MLR 3) - who owns the firm? No natural order
d["owner"] = "stand-alone"
d.loc[d["prgroup"] == 1, "owner"] = "business group"
d.loc[d["govown"] == 1, "owner"] = "state-owned"
d.loc[d["foreign"] == 1, "owner"] = "foreign"
levels = ["stand-alone", "business group", "state-owned", "foreign"]
d["owner_c"] = pd.Categorical(d["owner"], categories=levels).codes
X = d[["medium", "small"]].astype(float)
X["tfp1000"] = d["tfp1000"].astype(float)
X = sm.add_constant(X)
mn = sm.MNLogit(d["owner_c"], X).fit(disp=0)
print(mn.summary().tables[1])          # the coefficient table

# Step 13 · Multinomial logit: log-odds and relative risk ratios (session 4, MLR 4)
pd.set_option("display.width", 130)       # six columns: print them side by side
pd.set_option("display.max_columns", 10)
print(pd.concat({"log-odds": mn.params, "RRR": np.exp(mn.params)}, axis=1)
      .set_axis(["log-odds: group", "log-odds: state", "log-odds: foreign",
                 "RRR: group", "RRR: state", "RRR: foreign"], axis=1).round(3))

# Step 14 · Multinomial logit: probabilities (session 4, MLR 5)
grid = pd.DataFrame({"const": 1.0, "medium": 0.0, "small": 0.0,
                     "tfp1000": [d["tfp1000"].median(), -0.5, 0.0, 0.5]})
out = pd.DataFrame(mn.predict(grid).values, columns=levels, index=grid["tfp1000"].round(2))
print(out.round(3))

# Step 15 · The candidates for ridge, lasso and elastic net (session 5)
d = d.copy()
d["age2"] = d["age"] ** 2
years = pd.get_dummies(d["yr"], prefix="yr", drop_first=True).astype(float)   # 1989 is the
 ↪  base
d = pd.concat([d, years], axis=1)
YCOL = "tfp1000"
XCOLS = cols[1:] + ["age2"] + list(years.columns)
Xc, yc = d[XCOLS].astype(float), d[YCOL].astype(float)
print(f"{len(XCOLS)} candidates:", XCOLS)

# Step 16 · Ridge (session 5)
ridge = make_pipeline(StandardScaler(),                          # same scale for all
                      RidgeCV(alphas=np.logspace(-3, 3, 60))    # try 60 values of lambda
                      ).fit(Xc, yc)
print(f"lambda = {ridge[-1].alpha_:.2f}")
print(f"ridge kept {np.sum(ridge[-1].coef_ != 0)} of {len(XCOLS)} - it never drops a
 ↪  predictor")

# Step 17 · Lasso (session 5)
lasso = make_pipeline(StandardScaler(),
                      LassoCV(cv=5, random_state=0, max_iter=50_000)   # lambda by 5-fold CV
                      ).fit(Xc, yc)
```

```
kept = [c for c, b in zip(XCOLS, lasso[-1].coef_) if b != 0]
print(f"lambda = {lasso[-1].alpha_:.4f}")
print(f"lasso kept {len(kept)} of {len(XCOLS)}:", kept)

# Step 18 · Elastic net (session 5)
enet = make_pipeline(StandardScaler(),
                     ElasticNetCV(l1_ratio=0.5,            # half lasso, half ridge
                                  cv=5, random_state=0, max_iter=50_000)
                     ).fit(Xc, yc)
kept = [c for c, b in zip(XCOLS, enet[-1].coef_) if b != 0]
print(f"lambda = {enet[-1].alpha_:.4f}")
print(f"elastic net kept {len(kept)} of {len(XCOLS)}:", kept)

# Step 19 · The four, on rows they never saw (session 5, as reviewed in session 6)
Xtr, Xte, ytr, yte = train_test_split(Xc, yc, test_size=0.3, random_state=7)
def test_rmse(model):                  # fit on 70% of rows, score on the other 30%
    pipe = make_pipeline(StandardScaler(), model).fit(Xtr, ytr)   # standardise first
    kept = int(np.sum(np.abs(pipe[-1].coef_) > 1e-8))             # non-zero coefficients
    return kept, mean_squared_error(yte, pipe.predict(Xte)) ** 0.5
results = {
    "OLS, all predictors": test_rmse(LinearRegression()),
    "ridge":       test_rmse(RidgeCV(alphas=np.logspace(-3, 3, 60))),   # lambda by CV
    "lasso":       test_rmse(LassoCV(cv=5, random_state=0, max_iter=50_000)),
    "elastic net": test_rmse(ElasticNetCV(l1_ratio=0.5, cv=5, random_state=0,
     ↪  max_iter=50_000)),
}
res = pd.DataFrame(results, index=["predictors kept", "test RMSE"]).T.astype({"predictors
 ↪  kept": int})
print(res.round(3))
```