Python Cheatsheet

MATH60033A · Session 05 · Regularisation

Thierry Warin, PhD · HEC Montréal · A2026

Everything you need to type, in one place. The first four sections are the same every week; the last one is every name from sessions 1 to 5.

Starting Python in VS Codium

Open the course folder once — **File > Open Folder**, then pick the repository. Everything below assumes you are inside it.

Open a terminal with **View > Terminal** (`Ctrl` + `` ` `` on Windows and Linux, `Cmd` + `` ` `` on macOS). Create the environment the first time only:

```
python3 -m venv .venv      # macOS
python3 -m venv .venv      # Linux
py -m venv .venv           # Windows (PowerShell)
```

Then activate it — **every time you open a new terminal**:

```
source .venv/bin/activate  # macOS
source .venv/bin/activate  # Linux
.venv\Scripts\activate     # Windows (PowerShell)
```

You will see `(.venv)` at the start of the prompt. If you do not, nothing below will find pandas. Install the packages once, with the environment active:

```
pip install -r requirements.txt
```

Finally tell the editor which Python to use: `Ctrl`/`Cmd` + `Shift` + `P`, type **Python: Select Interpreter**, and choose the one inside `.venv`.

A new file, and running it

File > New File, then save it with a `.py` ending — `session05.py`, say. Write your code, save it, and in the terminal:

```
python session05.py     # macOS, Linux and Windows alike, inside .venv
```

Inside the environment the command is `python` on all three platforms. Outside it, macOS and Linux need `python3` and Windows needs `py`, which is one more reason to activate first.

For trying one line at a time, type `python` on its own to get the `>>>` prompt, and `exit()` to leave. The editor's **Run** triangle does the same thing as the command above.

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()          # the first five rows
df.shape           # (rows, columns)
df.dtypes          # what type each column holds
df.info()          # types, and how many values are missing
```

```
df.describe()      # count, mean, sd and quartiles, per numeric column

# 3 - read a file in
df = pd.read_csv("data.csv")
df = pd.read_excel("data.xlsx")                    # needs openpyxl - see below
df = pd.read_parquet("data/spine/core.parquet")

import qmib
df = qmib.load("core")                             # a course dataset, by name

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

Installing a package you do not have

Everything this course needs is already in `requirements.txt`. When you meet a `ModuleNotFoundError` anyway, or a line above says a reader needs something extra, install it **into the environment** — never outside it:

```
pip install openpyxl                # one package, by name
pip install -r requirements.txt     # or the whole course list again
```

Two rules, and both are about where it lands:

- Your prompt must show `(.venv)` **first**. `pip -V` prints the path it is about to install into; that path has to contain `.venv`.
- Type `pip`, not `pip3`, and never `sudo`. Inside an activated environment `pip` is already the environment's own.

Then check it arrived:

```
pip show openpyxl                   # name, version and location - or nothing at all
python -c "import openpyxl; print(openpyxl.__version__)"
```

What you have learned so far

Sessions 1 to 5, 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
```