statsmodels

Regression you can read: fit, read, check, predict

Python cheatsheet · MATH60033A · Quantitative Methods in IB · used in sessions 2 to 7, 10, 11 and 13

```
import statsmodels.formula.api as smf
```

+ import statsmodels.api as sm (no formula)

1 · Write the model as a formula

smf.ols("gdp ~ prod + invest", data=d)

outcome	~	predictors
left of ~	explained by	joined by +

intercept added for you · .fit() estimates

`smf.ols("y ~ x", data=d).fit()`
ordinary least squares: the line that makes the squared misses smallest.

`"y ~ x + z"`
two predictors: each slope now **holds the other fixed**.

`"y ~ C(region)"`
a categorical predictor: one dummy per level, minus a reference.

`"y ~ x + I(x**2)"`
a squared term: `I()` means *compute this*. The effect of x depends on x.

`"y ~ a * b"`
an interaction: expands to `a + b + a:b`, a slope per group.

`"np.log(y) ~ x"`
functions work inside the formula.

> Read ~ as *explained by*. Write the formula you would say aloud.

2 · Read the fit

OLS Regression Results
No. Observations: 428
R-squared: 0.627
Adj. R-sq.: 0.626
share of variance explained

	coef	std err	P>|t|
Intercept	-86118	4375	0.000
productivity	1111.3	41.5	0.000

slope: y units per x unit — p < 0.05

`fit.summary()`
the whole table: coefficients, p-values, R², n.

`fit.summary().tables[1]`
only the coefficient table — what fits on a slide.

`fit.params`
the estimates, as a Series you can compute with.

```
fit.params["productivity_idx"]   # 1111.3 €/point
```

`fit.pvalues · fit.conf_int()`
p-values · 95% intervals, one row per coefficient.

`fit.rsquared · fit.rsquared_adj`
share of variance explained · the same, penalised for extra predictors.

`fit.aic · fit.bic · fit.nobs`
information criteria (lower is better) · rows used.

> A slope is a sentence with units: *one more point of x goes with b more units of y, holding the others fixed*.

3 · Predict

`fit.predict(new)`
fitted values for new rows: a DataFrame with the predictor columns.

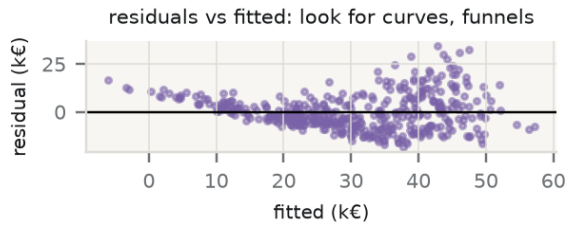
```
fit.predict(pd.DataFrame({"productivity_idx": [100, 120]}))
```

`fit.get_prediction(new).summary_frame()`
the prediction **with** its interval.

> Predict only inside the range of the data you fitted.

4 · Check the fit — diagnostics

Session 3: is the line adequate, valid, robust?

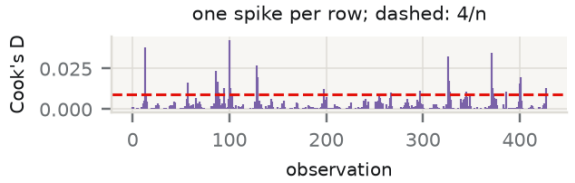


`fit.fittedvalues · fit.resid`
what the line predicts · what it missed, row by row.

`infl = fit.get_influence()`
the whole diagnostic bundle, computed once.

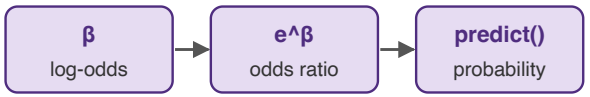
`infl.hat_matrix_diag`
leverage: how unusual each row's x is. They sum to the number of parameters.

`infl.cooks_distance[0]`
influence: how far the fit moves without that row.



> A tall spike is a row to **look at**, not to delete: report the fit with and without it.

5 · Binary logit — a 0/1 outcome

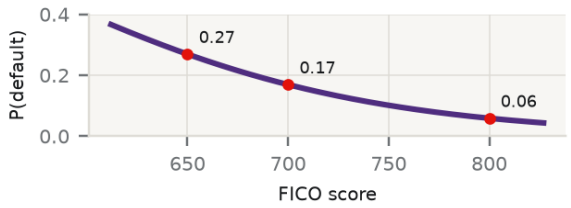



```
m = smf.logit("default ~ fico",
data=loans).fit(disp=0)
```
maximum likelihood; `disp=0` silences the iteration log.

```
m.params
```
log-odds: read the sign, not the size.

```
np.exp(m.params["fico"])
```
the **odds ratio**: 0.988 = odds of default 1.2% lower per point.

```
m.predict(pd.DataFrame({"fico": [650, 700, 800]}))
```
the **probability**: the whole linear predictor through the S-curve.



> An odds ratio of 1.5 is *odds* 50% higher — not a probability 50% higher.

6 · Which model for which outcome?

```
a number → smf.ols
```
GDP, productivity, a return: the slope is in units of y.

```
yes / no → smf.logit
```
default, exports, an up day: odds ratios and probabilities.

```
ordered categories → OrderedModel
```
unlikely < somewhat < very likely; low < middle < high.

```
unordered categories → sm.MNLogit
```
programme, entry mode, owner type: everything against a baseline.

7 · Ordinal logit — ordered categories



a positive β shifts y* right: higher categories

```
from statsmodels.miscmodels.ordinal_model import
OrderedModel
```

```
d["y_c"] = pd.Categorical(d["y"],
categories=order, ordered=True)
```
give the categories their order first.

```
om = OrderedModel(d["y_c"], d[["x1", "x2"]],
distr="logit")
```
one slope per predictor, shared by every cut: *proportional odds*.

```
om = om.fit(method="bfgs", disp=0)
```

```
np.exp(om.params[["x1", "x2"]])
```
odds of a **higher** category, at any cut.

```
om.predict(new)
```
one probability per category; each row sums to 1.

8 · When statsmodels complains

```
PatsyError: ... not defined
```
a name in the formula is not a column of `data=`: check the spelling.

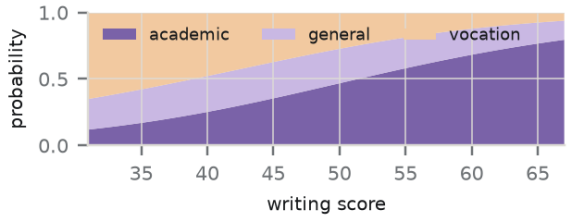
```
Optimization terminated / Iterations
```
the logit's log: add `disp=0` to `.fit()`.

```
Singular matrix / perfect separation
```
two predictors are copies, or one predicts the outcome perfectly.

```
ConvergenceWarning
```
try another optimiser: `.fit(method="bfgs")`.

```
AIC comparison looks too good
```
the models dropped different rows: `dropna` on the union first.

9 · Multinomial logit — unordered categories




```
y = pd.Categorical(ml["prog"],
categories=levels).codes
```
code 0 — the first level — is the **baseline**.

```
X = sm.add_constant(X)
```
no formula here: add the intercept yourself.

```
mn = sm.MNLogit(y, X).fit(disp=0)
```
one equation per category, each **against the baseline**.

```
np.exp(mn.params)
```
relative risk ratios: multiply P(category) / P(baseline).

```
mn.predict(grid)
```
probabilities for every category — what to report.

> Say *relative to the baseline* every time: a negative coefficient does not mean that category's probability falls.

10 · Compare models

```
LR = 2 * (full.llf - small.llf)
```
likelihood-ratio statistic for nested models; `.llf` is the log-likelihood.

```
stats.chi2.sf(LR, df=q)
```
its p-value, with q = coefficients dropped (from scipy).

```
full.compare_f_test(small)
```
the F-test for nested OLS models.

```
a.aic, b.aic
```
any two models — but only **on the same rows**.

```
1 - m.llf / null.llf
```
McFadden's pseudo-R^2 for a logit; 0.2–0.4 is already good.