`statsmodels` è una \*\*libreria Python\*\* per \*\*modelli statistici\*\*, che contiene vari moduli specializzati (composti da <u>classi e funzioni, ed anche dataset</u>) per la \*\*stima\*\* di diversi modelli statistici, nonché per l'esecuzione di \*\*test statistici\*\* e \*\*l'esplorazione\*\* di dati statistici.

`statsmodels` mette a disposizione vari \*stimatori\* di modelli ("\*\*estimator\*\*", stesso termine usato da \*scikit-learn\*); ad esempio:

- `smf.OLS()` → lo stimatore OLS (minimi quadrati ordinari)

- `smf.Logit()` → lo stimatore Logit (massima verosimiglianza)

- `tsa.ARIMA()` → lo stimatore ARIMA per serie temporali

- `GLM()` → stimatore di modelli lineari generalizzati

Per ogni stimatore è disponibile un ampio elenco di statistiche sui risultati.

La famosa libreria python `scikit-learn` (che fornisce ad esempio il noto stimatore generico `LinearRegression`) ha meno funzioni specifiche per le serie temporali (rispetto a `statsmodels`) e non sarà quindi usata in questo notebook.

## Le due sottolibrerie di `statsmodels`

La libreria `statsmodels` ha <u>due sotto-librerie</u>:<br>

📦 `statsmodels.api`

È la (sotto)libreria principale di `statsmodels`:

- contiene i modelli statistici generali (\*\*OLS\*\*, Logit, Probit, GLM, ecc)

- strumenti per analisi statistica in senso ampio

- \*\*non è focalizzata sulle serie temporali\*\*.

La si usa per regressioni classiche, ANOVA, modelli lineari generalizzati, test statistici, ecc e la si importa così:

```python

import statsmodels.api as sm

```

Con questa convenzione di import (`as sm`) si può poi richiamare:

- `sm.OLS(...)` → regressione lineare OLS

- `sm.Logit(...)` → regressione logistica

- `sm.add\_constant(...)` → aggiungere l’intercetta

- `sm.datasets.get\_rdataset(...)` → dataset di esempio

- ecc

<br>

📦 `statsmodels.tsa` (\*tsa = time series analysis\*)

È la (sotto) libreria \*\*dedicata alle serie temporali\*\*.<br>

Contiene i modelli e le funzioni specifiche per:

- AR, MA, ARMA, ARIMA, SARIMA

- VAR (Vector AutoRegression)

- analisi della stazionarietà (Dickey-Fuller test, ecc.)

- decomposizione stagionale (seasonal\_decompose)

- funzioni di autocorrelazione (ACF, PACF)

Si importa in due modi tipici:

```python

import statsmodels.tsa.api as tsa

```

oppure importando direttamente le sue librerie specifiche, ad es.:<br>

```python

from statsmodels.tsa import tsatools

from statsmodels.tsa.ar\_model import AutoReg

from statsmodels.tsa.arima.model import ARIMA

from statsmodels.tsa.seasonal import seasonal\_decompose

```

---

A differenza di `statsmodels.api`, `statsmodels.tsa` è dunque \*\*dedicata alla analisi e previsione di serie temporali\*\*.

11.2. *statsmodels.formula.api*

Attenzione!  
Per la costruzione ed il fit dei modelli di regressione **OLS**, anzichè la libreria statsmodels.api, noi in realtà useremo nel seguito la libreria statsmodels.formula.api, importata as smf (anzichè as sm, per distinguerle), che permette di specificare modelli statistici usando una **sintassi simile a R, basata su formule**.  
In pratica: invece di passare array/vettori di variabili (come in statsmodels.api), si può scrivere una **formula testuale in stile *patsy*** ( ad esempio: ).

### Patsy

\*\*Cos’è Patsy?\*\*

`Patsy` è una libreria Python (\*\*<u>usata “dietro le quinte” da `statsmodels.formula.api`</u>\*\*) che permette di <u>specificare</u> modelli statistici usando \*\*formule testuali in stile R\*\*.

Ad esempio, data una formula tipo $y \sim x1 + x2$, Patsy la trasforma in una \*\*MATRICE di DISEGNO\*\* (cioè la matrice 𝑋 con le variabili indipendenti e le eventuali trasformazioni).

Quando si usa `statsmodels.formula.api`, in realtà <u>è Patsy a interpretare la stringa $y \sim x1 + x2$</u>.

👉 Dietro le quinte succede questo:

- La formula $y \sim x1 + x2$ viene passata a Patsy.

- Patsy costruisce:

- la matrice di disegno X (predittori, con intercetta inclusa di default)

- il vettore y (risposta).

- `statsmodels` riceve 𝑋 e 𝑦 già pronti e stima il modello con il metodo scelto (OLS, GLM, ecc.).

Si può facilmente verificare che `statsmodels.formula.api` e Patsy (`patsy.dmatrices`) fanno lo stesso lavoro nel modo seguente:

```python

import patsy

import statsmodels.api as sm

# utilizzo esplicito di Patsy

y, X = patsy.dmatrices("y ~ x1 + x2", data=df) # esplode la formula e crea la matrice di disegno ed il vettore risposta

# costruzione e fit del modello con 'ols' di 'stasmodels.api' ('statsmodels.formula.api' non è più necessaria!)

model = sm.OLS(y, X).fit()

```

\*\*Esempi di sintassi Patsy (sintassi R)\*\*

```python

# import come 'smf' per distinguere da 'import statsmodels.api as sm'

import statsmodels.formula.api as smf

# modello lineare

smf.ols("y ~ x1 + x2", data=df)

# senza intercetta

smf.ols("y ~ x1 + x2 - 1", data=df)

# con interazione

smf.ols("y ~ x1 \* x2", data=df) # equivalente a y ~ x1 + x2 + x1:x2

# con trasformazioni

smf.ols("y ~ np.log(x1) + I(x2\*\*2)", data=df)

```

\*\*Perché è utile?\*\*

È molto più leggibile rispetto a costruire manualmente le matrici.

Permette di specificare formule complesse con pochi caratteri.

Dà la stessa comodità che si ha in R con la sintassi `lm(y ~ x1 + x2)`.

👉 In sintesi: Patsy è il “traduttore” che legge le formule in testo e le converte in strutture matematiche per `statsmodels`.

\*\*La matrice di disegno con Patsy\*\*

Quando si usa Patsy esplicitamente, come visto prima, \*\*in realtà si ottengono due oggetti\*\*:

```python

import patsy

y, X = patsy.dmatrices("y ~ x1 + x2", data=df)

```

X → la matrice di design dei predittori (inclusa l’intercetta se non la si toglie con -1)<br>

y → la variabile risposta

👉 Quindi Patsy può restituire sia y che X, ma la \*\*design matrix in senso stretto è solo 𝑋\*\*.