UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI ECONOMIA "GIORGIO FUÀ"

Corso di Laurea triennale in

Economia Aziendale

"Basic Econometrics" in Python

Relatore:

Prof. Claudia Pigini

Rapporto Finale di:

Alessandro Bachetti

Anno Accademico 2022/2023

"Basic Econometrics" in Python

Alessandro Bachetti

Maggio 2023

Introduzione

Questo lavoro nasce con l'intento di replicare e tradurre in linguaggio di programmazione Python i principali esempi, prodotti attraverso l'utilizzo del software gretl, presentati all'interno del manuale "Basic Econometrics" del Prof. Riccardo "Jack" Lucchetti, Dipartimento di Scienze Economiche e Sociali (DiSES), Università Politecnica delle Marche.
Il testo si sviluppa come segue: il capitolo 1 tratta l'applicazione in gretl, e successivamente in Python, del Teorema del Limite Centrale, richiamando brevemente la teoria relativa a questo argomento.
Il capitolo 2 tratta la regressione lineare, mentre il capitolo 3 illustra le principali diagnostiche riguardanti il metodo dei minimi quadrati ordinari applicato in un contesto di analisi cross-section.
Infine, il capitolo 4 tratta l'applicazione in gretl, e relativa conversione in Python, dei modelli dinamici ADL ed ECM per l'analisi delle serie storiche, con relativi metodi di diagnostica.

Indice

Capitolo 1

Inferenza Statistica

Cosa significa fare inferenza?
A differenza dell'analisi descrittiva di un fenomeno, che si limita a descrivere come questo si è manifestato, l'inferenza pone una domanda più complessa e profonda sul fenomeno stesso: "come si è generato?". Fare inferenza significa stabilire, attraverso determinate metodologie e tecniche, se l'analisi di un campione estratto da una popolazione sia capace di descrivere la popolazione stessa. Ciò che interessa è capire come la natura abbia generato il campione estratto, al fine di trovare il Data Generating Process (DGP) del campione.

1.1 Correttezza e consistenza di uno stimatore

Si supponga di avere in natura un dato **DGP** che stabilisca la creazione di un dato evento e che la sua legge di moto sia caratterizzata da un parametro chiamato θ.
L' obiettivo è quantificare questo parametro attraverso uno stimatore $\hat{\theta}$ in modo tale che, costruendo una funzione basata su quest'ultimo, sia possibile avere dei risultati molto simili a quelli che si hanno a disposizione, permettendoci così di poter ricreare e in futuro prevedere le realizzazioni che un determinato fenomeno proporrà.
Si definisce stimatore corretto quello stimatore $\hat{\theta}$, con un supporto contenente dei valori simili a quelli assunti dal parametro oggetto di stima θ, tale che:

$$E(\hat{\theta}) = \theta$$

Nel caso in cui $E(\hat{\theta}) \neq \theta$ la differenza tra i due valori prende il nome di **bias**.

Uno degli argomenti più importanti dell'inferenza statistica è la **Convergenza in probabilità**.
Una variabile casuale, e quindi anche lo stimatore $\hat{\theta}$, converge in probabilità se:

$$\hat{\theta} \xrightarrow{Pr} \theta \iff \lim_{n \to +\infty} P[|\hat{\theta} - \theta|] < \epsilon] = 1$$

La formula indica che, all'aumentare del numero di osservazioni n, lo stimatore $\hat{\theta}$ assumerà valori sempre più vicini a quelli assunti da θ. Se questa condizione è verificata si definisce θ come il *plim* di $\hat{\theta}$.

1.2 La Legge dei Grandi Numeri e il Teorema del Limite Centrale

Prendendo in esempio la media campionaria, definita come

$$\bar{X_n} = \frac{1}{n} \sum_{i=1}^{n} X_i$$

quale sarà il suo *limite in probabilità*?
Per rispondere a questa domanda viene in aiuto la **Legge dei Grandi Numeri di Khinchin.**
La **LLN** (Law of Large Numbers) pone dei vincoli sulla distribuzione delle singole osservazioni campionarie X_i, nello specifico:

- Indipendenza tra le variabili casuali;
- Tutte le variabili hanno distribuzione identica;
- $E[X_i] = m$.

Le variabili sono, quindi, **i.i.d**, ovvero *Indipendenti e Identicamente Distribuite*. Le condizioni poste in precedenza permettono di stabilire che $\bar{X_n} \xrightarrow{Pr} m$.

Un altro importantissimo concetto dell'inferenza statistica è il **Teorema del Limite Centrale**.
Prima, però, è necessario introdurre il concetto di **Convergenza in distribuzione**. La convergenza in distribuzione è verificata sotto la seguente condizione:

$$X_n \xrightarrow{d} X \iff F_{X_n}(x_i) \to F_X(x_i)$$

la quale indica che si ha una convergenza in distribuzione quando la funzione di ripartizione della v.c. X_n tende ad assumere la forma della funzione di ripartizione di X. Bisogna fare attenzione a non confondere la convergenza in distribuzione con la convergenza in probabilità. Quest'ultima, essendo legata al valore che X_n assume con l'aumentare delle osservazioni, porta con sè anche tutta la struttura probabilistica di X_n a convergere con quella di X. La convergenza in probabilità comporta, quindi, anche la convergenza in distribuzione. Non vale, invece, la relazione inversa; in questo caso il vincolo è posto soltanto sulla struttura probabilistica e non sui valori che X_n assumerà.

Avendo spiegato brevemente la convergenza in distribuzione è possibile, dunque, trattare il **Teorema del Limite Centrale** o **CLT** (Central Limit Theorem) di Lindeberg-Lévy.
In breve, ipotizzando:

- X_i campione di variabili casuali i.i.d
- $E(X_i) = m$
- $V(X_i) = \sigma^2$

è possibile dimostrare la seguente relazione:

$$\sqrt{n}(\bar{X_n} - m) \xrightarrow{d} N(0, \sigma^2)$$

Per maggiori approfondimenti e dimostrazioni su questo argomento e quelli precedentemente trattati si invita caldamente a consultare il manuale "Basic Econometrics" del Professor Riccardo "Jack" Lucchetti.[1]

[1] Il manuale è disponibile gratuitamente all'indirizzo web : http://www2.econ.univpm.it/servizi/hpp/lucchetti/didattica/basic.pdf

1.3 Introduzione al software gretl e breve esempio CLT

Questa sezione tratterà del software libero e open-source **gretl**, utilizzato nei corsi di Elementi di Econometria UNIVPM per analisi statistico-econometriche.
Nello specifico si andrà a spiegare l'esempio a pagina 69 del manuale"basic econometrics" citato precedentemente.
Il codice fornito nell'esempio è il seguente:

```
set verbose off
clear

#characteristics of the event

scalar p = 0.5
scalar n = 100
scalar lo = 36
scalar hi = 45

#true probability via the binomial distribution

matrix bin = pdf(B, p , n, seq(lo, hi)) #Binomial
    probabilities
scalar true = sumc(bin)

#approximation via the Central Limit Theorem

scalar m = p*n                    #mean
scalar s = sqrt(p*(1-p)*n)        #standard error
scalar z0 = (lo - 0.5 - m)/s      #subtract 0.5 to compensate
    for continuity
scalar z1 = (hi + 0.5 - m)/s      #add 0.5 to compensate for
    continuity

scalar appr= cnorm(z1) - cnorm(z0) #"cnorm" = Normal
    distribution function

#printout

printf "probability of \"heads\" = %g\n", p
printf "number of tosses = %g\n", n
printf "probability of heads between %d and %d:\n" , lo, hi
printf "true = %g, approximate via CLT =%g\n", true, appr
```

Questo codice Gretl esegue il calcolo della probabilità di ottenere una testa in un determinato numero di lanci di una moneta (un evento binomiale).

Si ricorda che la distribuzione binomiale viene utilizzata per calcolare la probabilità di avere x successi in n prove indipendenti, ognuna con una distribuzione Bernoulliana.
La funzione di probabilità della VC Binomiale risulta:

$$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$$

Nella prima riga, con il comando `set verbose off` si indica al software di non visualizzare i messaggi di output mentre il codice viene eseguito. La seconda riga, `clear`, cancella qualsiasi variabile che potrebbe essere stata definita in precedenza. Il codice definisce poi quattro variabili scalari:

- `p` è la probabilità di ottenere una testa (in questo caso è impostata su 0,5, il che significa che la moneta è equilibrata)
- `n` è il numero di lanci di moneta
- `lo` e `hi` rappresentano gli estremi dell'intervallo di interesse per il conteggio delle teste, quindi l'intevallo di successi.

Il codice, successivamente, calcola la probabilità dell'evento attraverso la distribuzione binomiale utilizzando la funzione `pdf` (Probability Distribution Function) di Gretl. Si specifica attraverso il parametro `B` di utilizzare una variabile casuale binomiale e a seguire si forniscono probabilità, numero di lanci e l'intervallo di interesse in forma di vettore colonna (`seq` genera un vettore riga, con il simbolo ' viene trasposto).
Questa operazione corrisponde al calcolo seguente:

$$\sum_{x=36}^{45} \binom{n}{x} p^x (1-p)^{n-x}$$

Dopo aver fatto tutto ciò, si assegna alla variabile matrice `bin` il risultato in forma di vettore colonna e gli elementi sommati di questa matrice alla variabile scalare `true` con il comando `sumc`.
Successivamente, si calcola l'approssimazione della probabilità attraverso il teorema del limite centrale (utilizzando la funzione `cnorm` di Gretl per la distribuzione normale standard).

Lanciando il codice si ottiene il seguente output:

```
probability of "heads" = 0,5
number of tosses = 100
probability of heads between 36 and 45:
true = 0,182342, approximate via CLT =0,182194
```

1.4 CLT in Python

```
1  import math
2  import scipy.stats as stats
3
4  p = 0.5
5  n = 100
6  lo = 36
7  hi = 45
8
9  # true probability via the binomial distribution
10 binomial = stats.binom(n, p)
11 bin = binomial.pmf(range(lo, hi+1))
12 true = bin.sum()
13
14 # approximation via the Central Limit Theorem
15 m = p * n   # mean
16 s = math.sqrt(p * (1 - p) * n)  # standard error
17 z0 = (lo - 0.5 - m) / s
18 z1 = (hi + 0.5 - m) / s
19 appr = stats.norm.cdf(z1) - stats.norm.cdf(z0)
20
21 # printout
22 print(f"probability of heads = {p}")
23 print(f"number of tosses = {n}")
24 print(f"probability of heads between {lo} and {hi}:")
25 print(f"true = {true}, approximate via CLT = {appr}")
```

Attraverso questo codice in linguaggio di programmazione Python si è in grado di riprodurre, con gli stessi risultati, il precedente esercizio in gretl.
Importando i **moduli** `math` e `scipy.stats` siamo in grado di utilizzare le funzioni dei moduli, cioè `sqrt` , `stats.binom` , `binomial.pmf` , `bin.sum` per svolgere specifiche operazioni matematico-statistiche.
In Python, `stats.binom(n, p)` è una funzione del modulo `scipy.stats` che restituisce una distribuzione binomiale discreta alla riga 10, richiedendo come parametri `n` (numero di tentativi) e `p` (probabilità dell'evento successo). Nella riga 11, attraverso il metodo `.pmf` si ottiene la *probability mass function*, cioè la funzione di densità. La funzione `binomial.pmf` prende come argomenti i valori `k`, cioè il numero di successi (in questo caso in un `range` che va da `lo` a `hi` incluso) e `n`, cioè il numero di prove, non specificato perché sottointeso essere `n=100`.
Attraverso la funzione integrata in Python `.sum` si sommano gli elementi contenuti nell'array di valori assegnato alla variabile `bin`.
Con la funzione `stats.norm` si genera una variabile casuale normale. Applicando il metodo `.cdf` andiamo a calcolare la funzione di ripartizione nei

punti `z1` e `z2`, cioè i due punti presi in esempio per i quali vogliamo calcolare la *Cumulative Distribution Function*.
Infine, attraverso il `print(f" ")` è possibile includere nella stringa interna al `print`, attraverso delle parentesi graffe, le variabili che abbiamo definito in precedenza
Lanciando il programma si ottiene, quindi, il seguente output:

```
probability of heads = 0.5
number of tosses = 100
probability of heads between 36 and 45:
true = 0.182341987801863, approximate via CLT =
   0.18219431204637543
```

Capitolo 2

OLS come strumento per fare inferenza

In questo capitolo verrà affrontato il metodo di stima più conosciuto in ambito statistico-econometrico: il modello **OLS - Ordinary Least Squares**.
Verrà trattata in particolare la replica di questo metodo di stima attraverso il linguaggio di programmazione Python.

2.1 Descrizione e principali proprietà del metodo

Come detto nell'introduzione il metodo OLS, anche conosciuto in italiano come metodo dei **Minimi Quadrati Ordinari**, è il mezzo più semplice ed efficace che permette di effettuare le prime stime anche a chi è alla prime armi, consentendo così di muovere i primi passi nel mondo dell'inferenza statistica. Questo metodo fornisce la stima dei parametri che caratterizzano un modello di regressione lineare.
Il modello è definito dalla formula (in forma vettoriale):

$$\underset{n\times 1}{\hat{y}} = \underset{n\times k}{X}\ \underset{k\times 1}{\hat{\beta}}$$

è caratterizzato dallo stimatore $\underset{k\times 1}{\hat{\beta}}$, dove:

$$\underset{k\times 1}{\hat{\beta}} = (\underset{k\times n}{X'}\ \underset{n\times k}{X})^{-1}\ \underset{k\times n}{X'}\ \underset{n\times 1}{y}$$

Quest'ultimo è, sotto determinate condizioni (vedi Ipotesi Classiche), **CAN** (Consistent and Asymptotically Normal) e **BLUE** (Best Linear Unbiased Estimator).

2.2 OLS in gretl

Alla tabella 3.1 a pagina 83 di "Basic Econometrics" è possibile incontrare per la prima volta all'interno del manuale una regressione OLS effettuata attraverso il software gretl. Il programma permette di effettuare questa stima attraverso due metodi:

- attraverso la GUI (Graphical User Interface);
- attraverso una riga di comando da eseguire nel terminale gretl

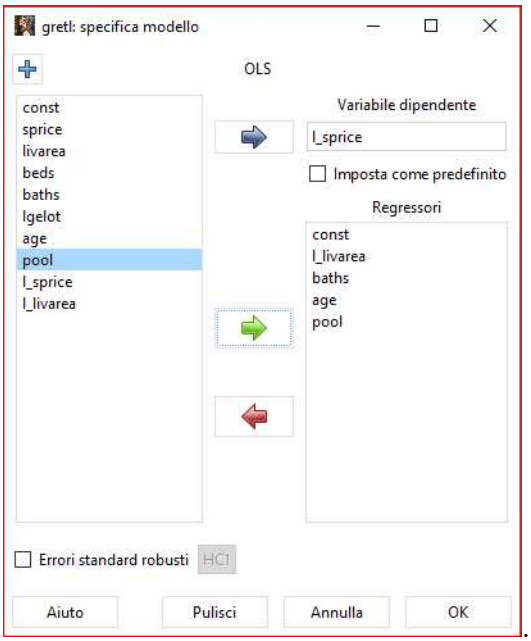
Scaricando il dataset proposto nell'esempio[2] è possibile notare una sequenza di variabili:

Legenda	
const	constant, aggiunta automaticamente
sprice	selling price of home, dollars
livarea	living area, hundreds of square feet
beds	number of beds
baths	number of baths
lgelot	=1 if lot size > 0.5 acres, 0 otherwise
age	age of home at time of sale, years
pool	=1 if home has pool, 0 otherwise

Quindi, dopo aver aperto il file `.gdt` con gretl e aver visualizzato le variabili in esame si dovrà fare click con il tasto destro sulle variabili `sprice` e `livarea`, selezionando l'opzione che permette di costruire una nuova variabile basata sul logaritmo naturale delle precedenti. Si ricorda che la trasformazione in logaritmi permette di "linearizzare" e quindi smussare le differenze delle rilevazioni contenute nelle variabili in esame. In questo caso i coefficienti $\hat{\beta}$ stimati devono essere interpretati come l'elasticità della variabile dipendente a quelle esplicative, ovvero di quanto varia `sprice` rispetto a un incremento unitario della variabile esplicativa. Il software, quindi, rinomina `sprice` in `l_sprice` e `livarea` in `l_livarea`.
Dopo aver eseguito i passaggi precedenti è possibile eseguire una stima OLS. Come accennato è possibile svolgere questa operazione attraverso due metodi. Il primo prevede l'utilizzo del pulsante  , posto nella parte inferiore della finestra di apertura del software; una volta cliccato sul pulsante è possibile accedere alla finestra di selezione delle variabili di stima.

Cliccando su OK si ottiene lo stesso output presente nel manuale:

[2]Il dataset è disponibile in formato `.gdt` all'indirizzo web: http://www.learneconometrics.com/gretl/poe5/data/stockton5.gdt



Modello 1: OLS, usando le osservazioni 1–2610
Variabile dipendente: l_sprice

	Coefficiente	Errore Std.	rapporto t	p-value
const	8,85359	0,0483007	183,3	0,0000
l_livarea	1,03696	0,0232121	44,67	0,0000
baths	−0,00515142	0,0130688	−0,3942	0,6935
age	−0,00238675	0,000270997	−8,807	0,0000
pool	0,106793	0,0226757	4,710	0,0000

Media var. dipendente	11,60193	SQM var. dipendente	0,438325
Somma quadr. residui	157,8844	E.S. della regressione	0,246187
R^2	0,685027	R^2 corretto	0,684544
$F(4, 2605)$	1416,389	P-value(F)	0,000000
Log-verosimiglianza	−42,58860	Criterio di Akaike	95,17721
Criterio di Schwarz	124,5127	Hannan–Quinn	105,8041

Il secondo metodo prevede l'utilizzo del terminale gretl. Dopo aver caricato il dataset contenente le variabili, è possibile eseguire una stima OLS attraverso una riga di comando nel terminale, scrivendo:

```
1 ols l_sprice const l_livarea baths age pool
```

avendo cura di specificare come primo argomento la variabile dipendente e come argomenti successivi le variabili indipendenti inclusa la costante, è

possibile riprodurre lo stesso esempio. Per conoscere le varie opzioni di stima (si veda `--robust` per stima con errori standard robusti) si consiglia di consultare il manuale gretl disponibile nella sezione "aiuto" del programma.

2.3 OLS in Python

Dopo aver trattato la stima OLS con gretl, in questa sezione si mostrerà come effettuare una stima in Python. Verranno utilizzate alcune tra le più famose librerie software e moduli per l'analisi di dati in Python e ne verranno spiegati la sintassi e il funzionamento.
In primo luogo occorre creare un file `.csv` utilizzando gretl. Questo passaggio si rende necessario per procedere alla lettura successiva dei dati con il programma in Python.
Dopo aver caricato il dataset nel software è necessario seguire la procedura: **File** $\longrightarrow$ **Esporta dati**$\longrightarrow$ **Selezione di tutte le variabili** $\longrightarrow$ **Selezione formato:** `CSV` $\longrightarrow$**OK** Tornando a Python, quindi, è possibile importare all'interno del ambiente di lavoro la prima libreria che verrà utilizzata: `pandas`.

```
1 import pandas as pd
```

`pandas` è una delle librerie per la manipolazione e l'analisi di dati più note. Permette di gestire tabelle numeriche, in questo caso il file `.csv` creato precedentemente, per procedere successivamente alla stima OLS in formato vettoriale.
`statsmodels` è un modulo Python che permette di effettuare stime e test statistici sui dati. In particolare, si utilizza `statsmodels.api` per l'analisi cross-section dei dati, in questo caso si occuperà della stima OLS vera e propria. È possibile importare tale modulo attraverso il seguente comando:

```
1 import statsmodels.api as sm
```

Ora che si hanno tutte le componenti necessarie è possibile scrivere il programma.

```
1  import pandas as pd
2  import statsmodels.api as sm
3
4  #assegnazione alla variabile "data" dei dati nel .csv
5
6  data = pd.read_csv("stockton5.csv")
7
8  #specificazione variabile dipendente e regressori
9
10 Y = data['l_sprice'].to_numpy()
11 X = data[['l_livarea','baths','age','pool']].to_numpy()
12
13
14 #aggiungo costante a X
15
16 X = sm.add_constant(X)
17
18 model = sm.OLS(Y, X).fit()
19 print(model.summary())
```

Nella riga 6 si procede alla lettura dei valori contenuti nel file `.csv` e alla assegnazione degli stessi alla variabile `data` con la funzione `pd.read_csv` della libreria `pandas`. Per leggere il file è necessario indicare la directory in cui si trova. Si consiglia di crearlo nella stessa directory in cui si trova il file `.py` associato al programma per evitare di specificare il percorso, consentendo, così, di inserire solo il nome del file.
Alle righe 10 e 11 si assegnano alle variabili `Y` e `X` i valori contenuti nelle colonne del `.csv` corrispondenti alla variabile dipendente e ai regressori. In queste due righe si utilizza la funzione `.to_numpy` appartenente alla libreria `pandas`. La funzione permette di convertire i dati delle variabili `Y` e `X` salvati con struttura `pandas.DataFrame` in dati con struttura `NumPy array`. Questo passaggio si rende necessario per evitare dei `FutureWarning` riguardanti la futura deprecazione del metodo di indexing multi-dimensionale con dati in formato `pandas.DataFrame`. Questa conversione risulta, però, meno chiara poiché non permette di mantenere il nome delle variabili in considerazione.
Attraverso il metodo `add_constant` del modulo `statsmodels.api` si aggiunge una colonna di 1 all' array `X`. Quest'ultimo è un passaggio fondamentale poiché consente di calcolare l'intercetta della retta di regressione. Infine, alle righe 18 e 19 si procede alla stima OLS e al print.
Con il comando `sm.OLS(Y , X)`, specificando l' array della variabile dipendente e l' array dei regressori, si effettua la stima OLS. Con il metodo `.fit()`

si specifica alla funzione OLS di eseguire, tra i possibili metodi, una stima utilizzando degli errori distribuiti come White Noise e di stimare una matrice Var-Cov non robusta dei regressori. Non è necessario passare alcun argomento al metodo `.fit()` poiché utilizza queste specifiche di default. Ricordando che:

$$\hat{y}_i = \texttt{const} + x1_i\beta_1 + x2_i\beta_2 + x3_i\beta_3 + x4_i\beta_4$$

si ridefiniscono le variabili per rendere congrui gli output di gretl con quelli forniti da Python:

Legenda		
gretl	Python	Descrizione
l_sprice	y	selling price of home, dollars
const	const	constant, aggiunta automaticamente
l_livarea	x1	living area, hundreds of square feet
baths	x2	number of baths
age	x3	age of home at time of sale, years
pool	x4	=1 if home has pool, 0 otherwise

Lanciando il programma, infine, si ottiene un output simile a questo:

Dep. Variable:	y	**R-squared:**	0.685
Model:	OLS	**Adj. R-squared:**	0.685
Method:	Least Squares	**F-statistic:**	1416.
Date:	....	**Prob (F-statistic):**	0.00
Time:	....	**Log-Likelihood:**	-42.589
No. Observations:	2610	**AIC:**	95.18
Df Residuals:	2605	**BIC:**	124.5
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	8.8536	0.048	183.301	0.000	8.759	8.948
x1	1.0370	0.023	44.673	0.000	0.991	1.082
x2	-0.0052	0.013	-0.394	0.693	-0.031	0.020
x3	-0.0024	0.000	-8.807	0.000	-0.003	-0.002
x4	0.1068	0.023	4.710	0.000	0.062	0.151

Omnibus:	131.881	**Durbin-Watson:**	1.224
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	456.090
Skew:	0.110	**Prob(JB):**	9.15e-100
Kurtosis:	5.036	**Cond. No.**	391.

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors
 is correctly specified.
```

2.4 L'importanza di pandas nell'analisi dei dati

Pandas è una libreria open-source creata sulla base di NumPy, la quale fornisce diverse strutture e funzioni per gestire dati e serie storiche. Le strutture che Pandas è capace di fornire sono:

- `Series`
- `DataFrame`

Le `Series` sono strutture dati uni-dimensionali capaci di contenere dati in qualsiasi formato. E' possibile creare una serie con il codice seguente:

```
1 import numpy as np
2 import pandas as pd
3
4 # creazione serie vuota
5 ser = pd.Series()
6
7 # creazione array
8 data = np.array(['a', 'b', 'c', 'd', 'e'])
9 ser = pd.Series(data)
```

Ottenendo il seguente output:

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

I `DataFrame`, invece, sono strutture bi-dimensionali composte da righe e colonne. Solitamente la struttura `DataFrame` è generata importando dei dati da un file, come a esempio un `.csv`. In questo esempio, però, è possibile vedere come creare manualmente un `DataFrame`:

```
import numpy as np
import pandas as pd

# creazione dati di lists.
data = {'Name': ['Claudia', 'Marco', 'Alessandro', 'Giulio'],
        'Age': [20, 21, 23, 18]}

# Creazione DataFrame
df = pd.DataFrame(data)

# Print output.
print(df)
```

L'output risulta essere il seguente:

```
         Name  Age
0     Claudia   20
1       Marco   21
2  Alessandro   23
3      Giulio   18
```

Capitolo 3

Test di diagnostica per regressioni Cross-Section

La fase di diagnostica è fondamentale per capire quanto il modello di regressione ipotizzato sia corretto rispetto ai dati osservati. Lo scopo di effettuare dei test diagnostici è quello di aggredire, e quindi verificare, la tenuta del modello nei confronti delle ipotesi su cui si regge, in questo caso le ipotesi classiche OLS.

3.1 Introduzione ai principali test diagnostici

Come detto in precedenza, l'obiettivo è aggredire le ipotesi su cui si fonda il modello di regressione lineare. Nello specifico in questo capitolo verranno trattati i principali test riguardanti le regressioni con dati in formato cross-section (serie di osservazioni individuali). I test sono:

1. Test sulla linearità dei parametri;
2. Test su omoschedasticità dei residui;
3. Test su break strutturali.

3.2 Diagnostica su linearità dei parametri

Il test RESET (**RE**gression **S**pecification **E**rror **T**est) è uno strumento utile per verificare la bontà, in fase di specificazione, della scelta dei regressori per il modello. Allo stesso tempo viene effettuato sia un test sulle variabili omesse che sulla linearità del modello.

Nell'esempio fornito nel manuale si procede al calcolo della statistica test attraverso il metodo del *Moltiplicatore di Lagrange* che prevede l'utilizzo del numero di osservazioni campionarie moltiplicate per l'indice R^2 calcolato sulla regressione ausiliaria.

$$LM = n * R^2 \sim \chi_2^2$$

Per utilizzare questo metodo sarà necessario:

1. Svolgere la stima OLS del modello in analisi;
2. Salvare i residui stimati $\hat{e}$ e le previsioni $\hat{y}$;
3. Generare un vettore di $\hat{y}^2$ e $\hat{y}^3$;
4. Eseguire nuovamente una stima OLS ponendo come variabile dipendente i residui della stima $\hat{e}$ e aggiungendo ai regressori i quadrati e i cubi delle stime fatte in precedenza. La risultante regressione ausiliaria sarà:
$$\hat{e}_i = \gamma' \mathbf{x}_i + \delta_1 \hat{y}_i^2 + \delta_2 \hat{y}_i^3 + u_i$$
Con γ pari al precedente vettore dei parametri OLS β.

3.3 Test RESET per la linearità in gretl

Dopo aver effettuato l'import del dataset `stockton5.gdt` all'interno del software gretl è possibile replicare l'esempio 4.1 a pagina 112 del manuale "Basic Econometrics" nel seguente modo:

```
1  set verbose off
2
3  #regressione ols da testare
4  ols l_sprice const l_livarea baths age pool
5
6  series yhat = $yhat      #salvataggio serie delle previsioni
7  series ehat = $uhat      #salvataggio serie dei residui
8  series yh2 = $yhat^2     #serie delle previsioni al quadrato
9  series yh3 = $yhat^3     #serie delle previsioni al cubo
10
11 #regressione ausiliaria
12 ols ehat const l_livarea baths age pool yh2 yh3
13
14 #salvataggio indice R2 dell'ultima regressione eseguita
15 scalar r2aux = $rsq
16
17 #calcolo statistica LM e print
18 scalar LM = $nobs*r2aux
19 printf "La statistica test LM vale: %g",LM
```

Alla riga 4 si esegue la prima regressione OLS attraverso il comando `ols` di gretl, specificando variabile dipendente e a seguire tutti i regressori, inclusa la costante `const`.
Nelle righe 6 e 7 si assegnano alle variabili `series` i valori delle previsioni e dei residui rispettivamente attraverso le funzioni `$yhat` e `$uhat`.
La specificazione `series`, eseguita precedentemente, si rende necessaria per conformare i valori restituiti dalle funzioni alla variabile a cui vengono assegnati.
Dopo aver eseguito la regressione ausiliaria, nella riga 15 si salva il valore del suo indice R2 all'interno della variabile `r2aux` attraverso la funzione `$rsq`, la quale si occuperà di restituire il valore R2 dell'ultimo modello stimato prima del comando.
Alla riga 18 si calcola la statistica LM moltiplicando `$nobs` per `$rsq`. La funzione `$nobs` di gretl produce in output il numero delle osservazioni campionarie dell'ultimo modello stimato prima del comando.
Alla riga 19, infine, si effettua il print della statistica test attraverso il comando `printf` al quale si passa come specificatore di formato `%g`, consigliato all'interno del manuale di gretl, il quale stabilisce il numero dei decimali dopo la virgola.
Lanciando il programma avremo il seguente output:

Modello 3: OLS, usando le osservazioni 1–2610
Variabile dipendente: ehat

	Coefficiente	Errore Std.	rapporto t	p-value
const	195,825	81,6622	2,398	0,0166
l_livarea	38,9198	17,1107	2,275	0,0230
baths	−0,205739	0,0861412	−2,388	0,0170
age	−0,0900385	0,0393760	−2,287	0,0223
pool	3,97729	1,76245	2,257	0,0241
yh2	−3,42319	1,40727	−2,433	0,0151
yh3	0,103615	0,0399778	2,592	0,0096

Media var. dipendente	0,000000	SQM var. dipendente	0,245999
Somma quadr. residui	152,9225	E.S. della regressione	0,242381
R^2	0,031428	R^2 corretto	0,029195
$F(6, 2603)$	14,07679	P-value(F)	7,85e–16
Log-verosimiglianza	−0,917127	Criterio di Akaike	15,83425
Criterio di Schwarz	56,90399	Hannan–Quinn	30,71192

La statistica test LM vale: 82,0263

3.4 Test RESET in Python

Per effettuare il test RESET di Ramsey si utilizza la funzione `linear_reset` della libreria `statsmodels`.
Nello specifico, è possibile eseguire il test RESET sulla regressione OLS trattata nell'esempio in gretl, precedentemente analizzato, attraverso il seguente codice:

```
1  import pandas as pd
2  import statsmodels.api as sm
3  from statsmodels.stats.diagnostic import linear_reset
4
5  #assegnazione alla variabile "data" dei dati nel .csv
6  data = pd.read_csv("stockton5.csv")
7
8  #specificazione variabile dipendente e regressori
9  Y = data['l_sprice'].to_numpy()
10 X = data[['l_livarea','baths','age','pool']].to_numpy()
11
12
13 #aggiungo costante a X
14 X = sm.add_constant(X)
15
16 model = sm.OLS(Y, X).fit()
17
18 print(model.summary())
19
20 print(linear_reset(model, power = 3, test_type = "fitted"))
```

Attraverso il comando presente nella riga 3 si effettua l'import della funzione `linear_reset`.

```
1  from statsmodels.stats.diagnostic import linear_reset
```

La funzione `linear_reset` è definita come segue:

```
1  linear_reset(res, power=3, test_type='fitted', use_f=False,
2               cov_type='nonrobust', cov_kwargs=None)
```

Attraverso il parametro `res` la funzione richiede in input dei dati in formato `RegressionResultsWrapper` che in questo caso sono forniti dalla variabile `model`.
Con il parametro `power` si richiede la potenza massima a cui elevare le previsioni OLS da inserire nella regressione ausiliaria.
Il parametro `test_type` stabilisce quali regressori aggiuntivi verranno utilizzati nella regressione ausiliaria. Fornendo come argomento `fitted` verranno

aggiunte le potenze delle previsioni specificate nell'argomento precedente.
`use_f = False` specifica alla funzione di utilizzare una distribuzione Chi-quadro per il calcolo della statistica test.
L'argomento `cov_type = nonrobust` indica alla funzione di utilizzare una matrice delle covarianze non robusta e accetta qualsiasi argomento `cov_type` ammesso dalla funzione `OLS` della libreria `statsmodels`.
Infine, `cov_kwargs` è un argomento impostato di default su `None` e che accetta qualsiasi argomento `cov_kwargs` ammesso dalla funzione `OLS` della libreria `statsmodels`.
Eseguendo il programma si ottiene il seguente output:

Dep. Variable:	y	**R-squared:**	0.685
Model:	OLS	**Adj. R-squared:**	0.685
Method:	Least Squares	**F-statistic:**	1416.
Date:	....	**Prob (F-statistic):**	0.00
Time:	....	**Log-Likelihood:**	-42.589
No. Observations:	2610	**AIC:**	95.18
Df Residuals:	2605	**BIC:**	124.5
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	8.8536	0.048	183.301	0.000	8.759	8.948
x1	1.0370	0.023	44.673	0.000	0.991	1.082
x2	-0.0052	0.013	-0.394	0.693	-0.031	0.020
x3	-0.0024	0.000	-8.807	0.000	-0.003	-0.002
x4	0.1068	0.023	4.710	0.000	0.062	0.151

Omnibus:	131.881	**Durbin-Watson:**	1.224
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	456.090
Skew:	0.110	**Prob(JB):**	9.15e-100
Kurtosis:	5.036	**Cond. No.**	391.

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.

<Wald test (chi2): statistic=84.46075639443279,p-value=4.566461287755668e-19,
                   df_denom=2>
```

3.5 Diagnostica sull' eteroschedasticità

La diagnostica sull'eteroschedasticità si occupa di testare se i residui della stima presentano una varianza costante oppure se quest'utlima varia da individuo a individuo. Avere dei residui eteroschedastici è una condizione deleteria per la stima poichè causerebbe dei calcoli errati di tutte le statistiche test sulla significatività dei parametri e di altri test sul modello impedendo, quindi, di poter effettuare qualsiasi inferenza sui dati. Nello specifico verrà trattato il test di White per verificare se il nostro modello è affetto da eteroschedasticità.

3.6 Test di White per l'eteroschedasticità in gretl

Dopo aver caricato il dataset `stockton5.gdt` è possibile replicare i risultati dell'esempio 4.4 a pagina 125 del manuale "Basic Econometrics" utilizzando la seguente riga di comando:

```
1 ols l_sprice const l_livarea baths age pool
2 modtest --white
```

Dopo aver effettuato la stima OLS sul dataset, attraverso il comando `modtest --white` che dovrà essere scritto immediatamente dopo il comando `ols`, si ottiene la stima OLS effettuata sulla regressione ausiliaria del test di White. In coda alla stima verrà fornita in output anche la statistica test LM con il relativo p-value.

Eseguendo il codice si riceve, quindi, il seguente output:

Test di White per l'eteroschedasticità OLS, usando le osservazioni 1-2610
Variabile dipendente: uhat$\hat{2}$

	Coefficiente	Errore Std.	rapporto t	p-value
const	0,806236	0,160351	5,028	5,30e-07
l_livarea	−0,710993	0,134434	−5,289	1,33e-07
baths	0,0936181	0,0514158	1,821	0,0688
age	0,00464135	0,00119047	3,899	9,91e-05
pool	0,267089	0,110699	2,413	0,0159
sq_l_livarea	0,159277	0,0307527	5,179	2,40e-07
X2_X3	−0,0409056	0,0240448	−1,701	0,0890
X2_X4	−0,00163418	0,000527404	−3,099	0,0020
X2_X5	−0,164525	0,0498979	−3,297	0,0010
sq_baths	0,00360873	0,00513648	0,7026	0,4824
X3_X4	0,000204179	0,000252621	0,8082	0,4190
X3_X5	0,0653657	0,0290985	2,246	0,0248
sq_age	−4,93515e-08	4,95338e-06	−0,009963	0,9921
X4_X5	0,00245662	0,000753929	3,258	0,0011

R^2 = 0,054056
Statistica test: TR$\hat{2}$ = 141,085963,
con p-value = P(Chi-quadro(13) > 141,085963) = 0,000000

3.7 Test di White in Python

Per effettuare il test di White si utilizza la funzione `het_white` della libreria `statsmodels`.
Nello specifico, è possibile eseguire il test White sulla regressione OLS trattata nell'esempio in gretl, precedentemente analizzato, attraverso il seguente codice:

```
import pandas as pd
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_white


#assegnazione alla variabile "data" dei dati nel .csv
data = pd.read_csv("stockton5.csv")

#specificazione variabile dipendente e regressori
Y = data['l_sprice'].to_numpy()
X = data[['l_livarea','baths','age','pool']].to_numpy()


#aggiungo costante a X
X = sm.add_constant(X)

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

print(model.summary(),"\n")

print(het_white(model.resid,X))
# Valori output : lagrange multiplier stat,
#                 p-value lagrange multi test,
#                 f-stat,
#                 p-value fstat
```

La funzione `het_white` richiede soltanto due argomenti in input: `resid` ed `exog`.
Con il parametro `resid` si richiede l'input di un array di dati contenenti i residui della stima OLS, che in questo caso vengono forniti dall'attributo `.resid` applicato all'oggetto `model`.
Il parametro `exog` richiede in input un array di dati contente i regressori (o variabili esogene) per il calcolo della matrice Var-Cov. In questo caso l'array fornito è `X`.
Lanciando il programma verrà restituito il seguente output:

Dep. Variable:	y	**R-squared:**	0.685
Model:	OLS	**Adj. R-squared:**	0.685
Method:	Least Squares	**F-statistic:**	1416.
Date:	....	**Prob (F-statistic):**	0.00
Time:	....	**Log-Likelihood:**	-42.589
No. Observations:	2610	**AIC:**	95.18
Df Residuals:	2605	**BIC:**	124.5
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	8.8536	0.048	183.301	0.000	8.759	8.948
x1	1.0370	0.023	44.673	0.000	0.991	1.082
x2	-0.0052	0.013	-0.394	0.693	-0.031	0.020
x3	-0.0024	0.000	-8.807	0.000	-0.003	-0.002
x4	0.1068	0.023	4.710	0.000	0.062	0.151

Omnibus:	131.881	**Durbin-Watson:**	1.224
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	456.090
Skew:	0.110	**Prob(JB):**	9.15e-100
Kurtosis:	5.036	**Cond. No.**	391.

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.

(141.08596287579718, 1.2751343330090841e-23, 11.411406426477324,
2.5510023311979512e-24)
```

3.8 Diagnostica per break strutturali

La diagnostica per i break strutturali tra le osservazioni si rende necessaria per prevenire delle stime che non tengono conto di un cambio di andamento tra diversi sotto-campioni. Questo tipo di analisi è spesso utilizzata in contesti time-series perché più suscettibili a cambi improvvisi dei valori osservati di un dato fenomeno(chiamati, appunto, break strutturali.)
Il test di Chow è lo strumento più utilizzato per tale verifica. Il test può essere eseguito attraverso l'utilizzo, tra i regressori della stima, di una "**splitdummy**" (una variabile con valori 0 e 1 che andrà a "selezionare" le osservazioni che apparterranno a uno dei sotto-campioni) per dividere il campione originale

in diversi sotto-gruppi in cui il cambio di andamento delle osservazioni è ritenuto più plausibile. La procedura per il calcolo di tale test può essere divisa in 3 step:

1. Calcolo degli SSR della regressione OLS sull'intero campione (dummy inclusa) (**SSRd**) ;
2. Calcolo degli SSR della regressione OLS sul sotto-campione in cui la dummy assume valore 1 (**SSR1**) ;
3. Calcolo degli SSR della regressione OLS sul sotto-campione in cui la dummy assume valore 0 (**SSR2**) .

La statistica test Chow risulta, quindi:

$$F = \frac{SSRd - (SSR1 + SSR2)}{SSR1 + SSR2} \frac{n - 2k}{k} \xrightarrow{d} F_{k,n-2k}$$

con k pari al numero dei regressori e n pari al numero delle osservazioni campionarie.
Le ipotesi di tale test sono, quindi:

$$\text{CHOW} : \begin{cases} H0 = \text{break assente} \\ H1 = \text{break presente} \end{cases}$$

3.9 Test di Chow per break strutturali in gretl

In gretl è possibile eseguire il test di Chow attraverso il seguente comando:

```
1 ols l_sprice const l_livarea baths age --quiet
2 chow pool --dummy
```

Come evidenziato dal codice è possibile eseguire il test attraverso il comando `chow` avendo cura di eseguirlo subito dopo una regressione OLS, la quale non dovrà contenere tra i regressori la variabile dummy che abbiamo scelto per suddividere il campione completo. L'argomento del comando `chow` risulta essere, dunque, la variabile dummy in esame e come opzione è necessario specificare `--dummy`, questo permetterà al programma di eseguire un test di Chow attraverso delle interazioni tra la dummy e i parametri `beta` del modello.

In particolare risulta:

$$\begin{aligned}\texttt{l\_sprice} = \beta_0 + \beta_1\texttt{l\_livarea} + \beta_2\texttt{baths} + \beta_3\texttt{age} \\ +\beta_4\texttt{pool} + \beta_5\texttt{pool*l\_livarea} + \\ +\beta_6\texttt{pool*baths} + \beta_7\texttt{pool*age}\end{aligned}$$

Eseguendo il codice si riceve, quindi, il seguente output:

Regressione aumentata per il test Chow OLS, usando le osservazioni 1-2610
Variabile dipendente: l_sprice

	Coefficiente	Errore Std.	rapporto t	p-value
const	8,86963	0,0497162	178,4	0,0000
l_livarea	1,03371	0,0238848	43,28	0,0000
baths	−0,00644016	0,0134051	−0,4804	0,6310
age	−0,00255117	0,000275490	−9,260	4,13e-20
pool	−0,225375	0,218878	−1,030	0,3033
po_l_livarea	0,0602648	0,0983527	0,6127	0,5401
po_baths	0,00861632	0,0584199	0,1475	0,8828
po_age	0,00546735	0,00155200	3,523	0,0004

Media var. dipendente	11,60193	SQM var. dipendente	0,438325
Somma quadr. residui	157,0758	E.S. della regressione	0,245698
R^2	0,686640	R^2 corretto	0,685797
$F(4, 2605)$	814,5083	P-value(F)	0,000000
Log-verosimiglianza	−35,88788	Criterio di Akaike	87,77576
Criterio di Schwarz	134,7126	Hannan–Quinn	104,7788

Test Chow per differenza strutturale rispetto a pool
F(4, 2602) = 8.91581 con p-value 0,0000

Con un p-value minore di 0.05 rifiutiamo l'ipotesi nulla H0, pertanto siamo in presenza di break strutturale rispetto alla variabile `pool`.

3.10 Test di Chow in Python

Al fine di riprodurre il test di Chow in Python è stata creata una funzione denominata `ChowTest` .
Attraverso il seguente codice è possibile definire la funzione in questione:

```
import pandas as pd
import statsmodels.api as sm
import scipy

def ChowTest(data, y_variable_name, subgroup_variable_name):
  '''data: un dataframe contenente tutti i dati inclusa la
                                variabile dipendente y
     y_variable_name: nome della variabile dipendente y
     subgroup_variable_name: varibile dummy su cui testare il
                                break'''
  X=data.drop(y_variable_name,axis=1)
  for column in X.columns:
    # Creazione nome della nuova variabile interagita
    new_var_name = "d_" + column

    if column != subgroup_variable_name:
      # Moltiplicazione della variabile per la dummy
      X[new_var_name]=X[column]*data[subgroup_variable_name]

  X=sm.add_constant(X)

  y=data[y_variable_name]

  #Regressione aumentata
  model_dummy = sm.OLS(y,X).fit()
  print(model_dummy.summary())
  SSRa=model_dummy.ssr

  X_1=data.drop(y_variable_name,axis=1).drop(
                                subgroup_variable_name,axis=1)
  X_1=sm.add_constant(X_1)

  #Regressione originale
  model_restr=sm.OLS(y,X_1).fit()
  SSRb=model_restr.ssr

  k=X.shape[1] - X_1.shape[1]
  n=X.shape[0]

  #Calcolo Chow
  chow = ((SSRb-SSRa)/SSRa)*((n-2*k)/k)

  p = scipy.stats.f.cdf(chow, k, n-2*k)
```

```
41
42    print(f'''Test Chow per differenza strutturale rispetto a
43                                        {subgroup_variable_name}
44    F({k}, {n-2*k}) = {chow} con p-value {1-p}''')
45
46  #Assegnazione alla variabile "data" dei dati nel .csv
47  data = pd.read_csv('stockton5_2.csv')
48
49  #Esecuzione del test
50  ChowTest(data,'l_sprice','pool')
```

Dopo aver importato i moduli necessari per eseguire le operazioni all'interno del codice e aver creato un file `.csv` contenente le sole variabili da analizzare (in questo caso `stockton5_2`), si procede alla definizione della funzione attraverso il comando `def` seguito dal nome e dagli argomenti della funzione stessa. Gli argomenti della funzione sono:

- `data`: un dataframe contenente tutti i dati inclusa la variabile dipendente y;
- `y_variable_name`: nome della variabile dipendente y;
- `subgroup_variable_name`: varibile dummy su cui testare il break.

Utilizzando il metodo `.drop`, applicato alla variabile `X`, si indica al programma di rimuovere dal `DataFrame` la variabile specificata all'interno delle parentesi indicando successivamente di eliminare la corrispondente colonna da `X` attraverso l'argomento `axis=1`. Successivamente si definisce un ciclo `for`, il quale si occupa di scorrere ogni colonna (in questo caso la variabile locale `column`) di `X` (elemento iterabile) e, a ogni iterazione, di creare una nuova variabile `new_var_name` definita dalla somma tra la stringa `"d_"` e il nome presente all'interno della variabile `column`. Lo statement `if` alla riga 14 del codice verifica se il nome della colonna assegnata a `column` è diverso dalla variabile `subgroup_variable_name` fornita come argomento alla funzione: in caso la condizione restituisca il valore `True` si procede al prodotto tra la colonna assegnata a `column` e la variabile `subgroup_variable_name` contenuta nel `DataFrame data`; in caso la condizione restituisca il valore `False`, invece, si esce dal `if`.
Dopo aver aggiunto la costante a `X` si procede all'assegnazione della variabile dipendente a `y`. Seguendo la procedura di calcolo definita all'interno del manuale d'uso di gretl, si procede allo svolgimento della regressione aumentata di tutte le interazioni regressori-dummy (esclusa l'interazione dummy-dummy) e della regressione senza variabile dummy salvando i valori degli **SSR** di ciascuna. La statistica test calcolata alla riga 38 risulta essere quindi un test F

che considera gli SSR del modello aumentato come quelli vincolati e gli SSR del modello senza dummy come quelli liberi. Alla riga 40 si procede, quindi, al calcolo del p-value del test attraverso la funzione `scipy.stats.f.cdf` specificando tra gli argomenti il valore su cui calcolare la *Cumulative Distribution Function* e i gradi di libertà *dfn* e *dfd* del test in esame.

Lanciando il programma si ottiene il seguente output:

Dep. Variable:	l_sprice	**R-squared:**	0.687
Model:	OLS	**Adj. R-squared:**	0.686
Method:	Least Squares	**F-statistic:**	814.5
Date:	...	**Prob (F-statistic):**	0.00
Time:	...	**Log-Likelihood:**	-35.888
No. Observations:	2610	**AIC:**	87.78
Df Residuals:	2602	**BIC:**	134.7
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
const	8.8696	0.050	178.405	0.000	8.772	8.967
l_livarea	1.0337	0.024	43.279	0.000	0.987	1.081
baths	-0.0064	0.013	-0.480	0.631	-0.033	0.020
age	-0.0026	0.000	-9.260	0.000	-0.003	-0.002
pool	-0.2254	0.219	-1.030	0.303	-0.655	0.204
d_l_livarea	0.0603	0.098	0.613	0.540	-0.133	0.253
d_baths	0.0086	0.058	0.147	0.883	-0.106	0.123
d_age	0.0055	0.002	3.523	0.000	0.002	0.009

Omnibus:	128.711	**Durbin-Watson:**	1.226
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	440.691
Skew:	0.101	**Prob(JB):**	2.02e-96
Kurtosis:	5.003	**Cond. No.**	1.75e+03

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors
 is correctly specified.
[2] The condition number is large, 1.75e+03. This might indicate that
there are strong multicollinearity or other numerical problems.

Test Chow per differenza strutturale rispetto a pool
F(4, 2602) = 8.915806058916424 con p-value 3.7819152642892817e-07
```

Capitolo 4

Modelli dinamici e diagnostiche

I modelli dinamici sono uno strumento di inferenza piuttosto utile nel caso in cui il fenomeno in esame presenti *persistenza* tra le osservazioni. Con persistenza si intende quella capacità, tipica delle osservazioni time-series, di avere degli scostamenti piccoli per osservazioni vicine e scostamenti ampi tra i valori nel caso in cui le osservazioni siano distanti temporalmente. I modelli dinamici ci permettono quindi di capire quanto le osservazioni precedenti influenzano quelle al tempo corrente.

4.1 Auto Regressive Distributed Lags

Il primo modello in esame è l'**ADL**(p,q) (anche definito **ARDL**).
Il modello **ARDL**(p,q) (**A**uto **R**egressive **D**istributed **L**ags) permette di ricondurre alla forma chiusa (OLS) la stima dei parametri β che caratterizzano la relazione di breve e lungo periodo del fenomeno in esame.
Definito dalla formula:

$$y_t = \sum_{i=1}^{p} \alpha_i y_{t-i} + \sum_{i=0}^{q} \beta_i' \mathbf{x}_{t-i} + \epsilon_t$$

il modello è caratterizzato dai seguenti parametri:

- y_t= variabile dipendente al tempo t ;
- α_i= coefficienti dei ritardi della variabile dipendente Y_{t-i} ;
- β_i= coefficienti dei ritardi delle variabili esplicative $\mathbf{x}_{t-i}$;
- ϵ_t= termine di errore al tempo t.

4.2 Error Correction Model

L' **ECM** (**E**rror **C**orrection **M**odel) è un modello derivato dall'ADL, di fatto non è altro che una sua riparametrizzazione. L'ECM è utile quando si vuole analizzare l'equilibrio a lungo termine tra le variabili, ma anche la velocità con cui le variabili si correggono quando si verificano deviazioni dalla relazione di lungo termine.
Partendo da un ADL(1,1) è possibile definire l'ECM come segue:

$$\Delta y_t = \beta_0' \Delta \mathbf{x}_t + (\alpha - 1) \left[y_{t-1} - \frac{(\beta_0 + \beta_1)'}{1-\alpha} \right] \mathbf{x}_{t-1}$$

dove $\frac{(\beta_0+\beta_1)'}{1-\alpha}$ rappresenta il termine di correzione dell'errore.

4.3 ARDL in gretl

Dopo aver importato il dataset[3] proposto nell'esempio a pagina 149 del manuale "basic econometrics" è possibile replicare il risultato in gretl attraverso questo codice:

```
1 set echo off
2 set messages off
3 logs YER PCR
4 y = l_YER
5 c = l_PCR
6 ols c const c(-1) y(0 to -3)
7 modtest 8 --autocorr --quiet
```

Attraverso il comando `logs` è possibile creare una nuova serie contenete il logaritmo della variabile passata come argomento, in questo caso le variabili sono due: `YER` e `PCR`; dopo aver fatto ciò la funzione restituirà due nuove variabili con lo stesso nome delle precedenti più il prefisso " `l_` ".
Dopo aver assegnato alle variabili `y` e `c` i logaritmi delle serie precedenti si procede alla stima OLS. In gretl è possibile specificare i "ritardi" (lags) di una variabile attraverso l'utilizzo delle parentesi tonde, poste subito dopo la variabile in questione, specificando al loro interno il singolo ritardo o un intervallo di lags. Infine, attraverso il comando `modtest 8 --autocorr --quiet` si eseguono una serie di test sull'autocorrelazione dei residui con ordine di controllo pari al valore inserito come argomento del comando, in questo caso 8.

[3] Il dataset è disponibile a questo indirizzo in formato `.csv`: https://eabcn.org/sites/default/files/awm19up18.csv

Lanciando il programma si ottiene, quindi, il seguente output:

adl: OLS, usando le osservazioni 1970:4–2017:4 ($T = 189$)
Variabile dipendente: c

	Coefficiente	Errore Std.	rapporto t	p-value
const	0,0687220	0,0144032	4,771	0,0000
c_1	0,963061	0,0184605	52,17	0,0000
y	0,609100	0,0496618	12,26	0,0000
y_1	−0,752277	0,0838447	−8,972	0,0000
y_2	0,292231	0,0835855	3,496	0,0006
y_3	−0,118342	0,0496405	−2,384	0,0181

Media var. dipendente	13,54363	SQM var. dipendente	0,267142
Somma quadr. residui	0,002424	E.S. della regressione	0,003639
R^2	0,999819	R^2 corretto	0,999814
$F(5, 183)$	202578,5	P-value(F)	0,000000
Log-verosimiglianza	796,2943	Criterio di Akaike	−1580,589
Criterio di Schwarz	−1561,138	Hannan–Quinn	−1572,709
$\hat{\rho}$	0,015128	h di Durbin	0,215018

Test Breusch-Godfrey per Autocorrelazione fino all'ordine 8

Statistica test: LMF	=	1,703551
con p-value	=	P(F(8,175) > 1,70355) = 0,1
Statistica alternativa: $TR\hat{2}$	=	13,655259
con p-value	=	P(Chi-quadro(8) > 13,6553) = 0,0912
Ljung-Box Q'	=	11,2944
con p-value	=	P(Chi-quadro(8) > 11,2944) = 0,186

4.4 ARDL in Python

È possibile riprodurre il precedente esempio in Python utilizzando la funzione `ARDL` della libreria `statsmodels.tsa.api` attraverso il codice seguente:

```
1  import pandas as pd
2  import numpy as np
3  import statsmodels.api as sm
4  from statsmodels.tsa.api import ARDL
5  from statsmodels.stats.diagnostic import acorr_ljungbox,
                                            acorr_lm
6
7  data=pd.read_csv('awm19up18.csv')
8  y=np.log(data.YER)
9  c=np.log(data.PCR)
10 y=sm.add_constant(y)
11
12
13 model=ARDL(c, 1, y ,{'const':None,'YER':3}).fit()
14 print(model.summary(),'\n')
15
16 print(acorr_lm(model.resid,8),'\n')
17 print(acorr_ljungbox(model.resid,[8]),'\n')
```

Dopo aver letto, alla riga 7, il nostro dataset in formato `.csv` si procede alla creazione delle variabili `y` e `c` attraverso la funzione matematica `np.log` della libreria `numpy`. La funzione `np.log` trasforma le variabili `array` passate come argomento nel loro logaritmo naturale, in questo caso la funzione è applicata alle variabili `YER` e `PCR` accessibili attraverso i comandi `data.YER` e `data.PCR`. Dopo aver aggiunto la constante tra i regressori si procede al calcolo della regressione attraverso il comando `ARDL(endog, lags, exog, order)`. Analizzando la funzione Python precedente è possibile notare la presenza di quattro argomenti utili al fine della stima:

- `endog`: array 1-D contenente i valori della variabile dipendente;
- `lags`: numero dei ritardi della variabile dipendente da inserire nel modello, possono essere specificati intervalli (attraverso un valore `int`) o specifici valori (attraverso una lista `[...]`);
- `exog`: array 2-D o `DataFrame` contenente le variabili esplicative del modello;
- `order`: numero dei ritardi delle variabili esplicative, se `int` applica a tutte le variabili i lags da 0...`int`. Se `dict` applica uno specifico

ordine a una specifica variabile, dove la `chiave` del dizionario è il nome della variabile e il `valore` associato alla chiave è il numero di lags da applicare.

Dopo aver eseguito la stima si procede al calcolo delle statistiche test sull'autocorrelazione attraverso le funzioni `acorr_lm` e `acorr_ljungbox` della libreria `statsmodels.stats.diagnostic`.
La funzione `acorr_lm(resid, nlags)` esegue un test LM per l'autocorrelazione calcolato come `(nobs-ddof)*r2`, dove `r2` è l'indice R2 della regressione ausiliaria dei residui; richiede i seguenti parametri:

- `resid`: array dei residui del modello in analisi. In questo esempio è possibile accedere ai residui del modello ARDL attraverso `model.resid`;
- `nlags`: ordini di controllo per l'autocorrelazione dei residui.

L'output della funzione `acorr_lm` è il seguente:

- `lm`: valore della statistica test LM ;
- `lmpval`: p-value della statistica LM ;
- `fval`: valore della statistica F-test alternativa al test LM ;
- `fpval`: p-value della statistica `fval`.

Infine, la funzione `acorr_ljungbox(x, lags)` esegue un test Ljung-Box per l'autocorrelazione. La funzione richiede i seguenti parametri:

- `x`: array dei residui del modello in analisi
- `lags`: se `int` restituisce le `lbstat` da 0...`int`, se `list [...]` restituisce il valore `lbstat` e il p-value dei soli ordini di controllo inseriti nella lista.

L'output della funzione `acorr_ljung_box` è il seguente:
`DataFrame` con colonne:

- `lb_stat`: statistica del test Ljung-Box;
- `lb_pvalue`: p-value della statistica.

Lanciando il programma si ottiene il seguente output:

Dep. Variable:	PCR	**No. Observations:**	192
Model:	ARDL(1, 3)	**Log Likelihood**	805.726
Method:	Conditional MLE	**S.D. of innovations**	0.004
Date:	...	**AIC**	-1597.452
Time:	...	**BIC**	-1574.686
Sample:	3	**HQIC**	-1588.231
	192		

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0687	0.014	4.771	0.000	0.040	0.097
PCR.L1	0.9631	0.018	52.169	0.000	0.927	0.999
YER.L0	0.6091	0.050	12.265	0.000	0.511	0.707
YER.L1	-0.7523	0.084	-8.972	0.000	-0.918	-0.587
YER.L2	0.2922	0.084	3.496	0.001	0.127	0.457
YER.L3	-0.1183	0.050	-2.384	0.018	-0.216	-0.020

```
(10.68084955988661, 0.22044653536308648, 1.3482821217940848,
 0.22280626000337148)

   lb_stat     lb_pvalue
8  11.29439    0.18557
```

4.5 ECM in gretl

La riproduzione dell'esempio a pagina 153 del manuale è possibile attraverso le seguenti righe di comando in gretl:

```
set echo off
set messages off
open awm19up18.csv --quiet
logs YER PCR
y = l_YER
c = l_PCR
series d_c = c-c(-1)
series d_y = y-y(-1)

ols d_c const d_y(0 to -2) c(-1) y(-1)
```

Questo codice non differisce molto da quello utilizzato per il modello ARDL, difatti l'ECM non è altro che un'estensione di quest'ultimo. Attraverso il comando `series` è possibile creare delle nuove variabili all'interno del data-set contenenti le variazioni della variabile dipendente e di quella esplicativa.

Successivamente si procede alla stima OLS del modello seguendo le stesse procedure utilizzate per la regressione ADL(p,q).
Eseguendo il programma si ottiene, quindi, il seguente output:

Modello 1: OLS, usando le osservazioni 1970:4–2017:4 ($T = 189$)
Variabile dipendente: d_c

	Coefficiente	Errore Std.	rapporto t	p-value
const	0,0687220	0,0144032	4,771	0,0000
d_y	0,609100	0,0496618	12,26	0,0000
d_y_1	−0,173888	0,0522364	−3,329	0,0011
d_y_2	0,118342	0,0496405	2,384	0,0181
c_1	−0,0369392	0,0184605	−2,001	0,0469
y_1	0,0307119	0,0177545	1,730	0,0854

Media var. dipendente	0,004976	SQM var. dipendente	0,005557
Somma quadr. residui	0,002424	E.S. della regressione	0,003639
R^2	0,582529	R^2 corretto	0,571122
$F(5, 183)$	51,07068	P-value(F)	5,86e–33
Log-verosimiglianza	796,2943	Criterio di Akaike	−1580,589
Criterio di Schwarz	−1561,138	Hannan–Quinn	−1572,709
$\hat{\rho}$	0,015128	Durbin–Watson	1,954127

4.6 ECM in Python

La riproduzione in Python del precedente esempio dell'ECM in gretl è possibile utilizzando la funzione `UECM` della libreria `statsmodels.tsa.api`. Attraverso questo codice è possibile replicare i risultati forniti da gretl:

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.tsa.api import UECM

data=pd.read_csv('awm19up18.csv')
y=np.log(data.YER)
c=np.log(data.PCR)
y=sm.add_constant(y)

ECM = UECM(c,1,y,{'const':None,'YER':3}).fit()
print(ECM.summary())
```

Anche in questo caso la procedura non differisce molto da quella utilizzata per il modello ARDL(p,q).

La funzione `UECM` richiede i seguenti parametri:

- `endog`: array 1-D contenente i valori della variabile dipendente;
- `lags`: numero dei ritardi della variabile dipendente da inserire nel modello, possono essere specificati intervalli (attraverso un valore `int`) o specifici valori (attraverso una lista `[...]`);
- `exog`: array 2-D o `DataFrame` contenente le variabili esplicative del modello;
- `order`: numero dei ritardi delle variabili esplicative, se `int` applica a tutte le variabili i lags da 0...`int`. Se `dict` applica uno specifico ordine a una specifica variabile, dove la `chiave` del dizionario è il nome della variabile e il `valore` associato alla chiave è il numero di lags da applicare.

Come è possibile notare gli argomenti da fornire alla funzione sono gli stessi utilizzati dalla funzione `ARDL`.
Eseguendo il programma otteniamo, infine, il seguente output:

Dep. Variable:	D.PCR	**No. Observations:**	192
Model:	UECM(1, 3)	**Log Likelihood**	805.726
Method:	Conditional MLE	**S.D. of innovations**	13.470
Date:	...	**AIC**	-1597.452
Time:	...	**BIC**	-1574.686
Sample:	3	**HQIC**	-1588.231
	192		

	coef	**std err**	**z**	**P> |z|**	**[0.025**	**0.975]**
const	0.0687	0.014	4.771	0.000	0.040	0.097
PCR.L1	-0.0369	0.018	-2.001	0.047	-0.073	-0.001
YER.L1	0.0307	0.018	1.730	0.085	-0.004	0.066
D.YER.L0	0.6091	0.050	12.265	0.000	0.511	0.707
D.YER.L1	-0.1739	0.052	-3.329	0.001	-0.277	-0.071
D.YER.L2	0.1183	0.050	2.384	0.018	0.020	0.216