Econometric analysis with Python (practice)

Programa de Doctorat en Economia, Universitat de Barcelona

David Moriña

Python practice 3

Exercise 1

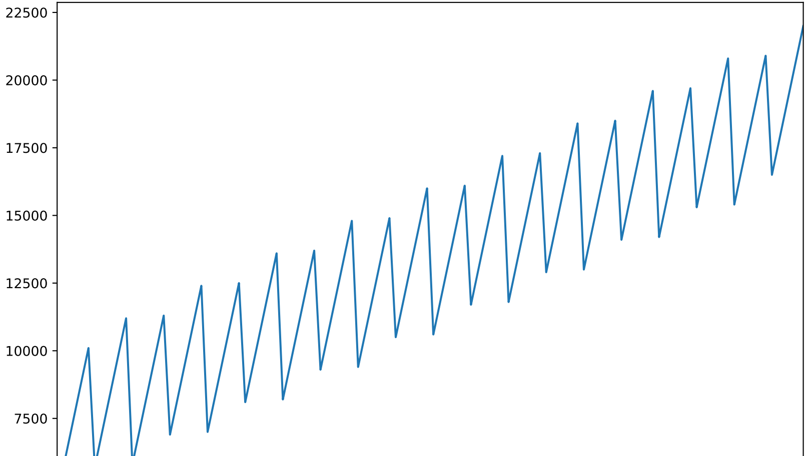
- Load the dataset "salesdata.csv" into the Python session and convert the column "Date" to *datetime*

```
import pandas as pd
import statsmodels.api as sm
import matplotlib.pyplot as plt

df=pd.read_csv('/home/dmorina/Insync/dmorina@ub.edu/OneDriv
 ↪  Biz/Docència/UB/2023-2024/PyEcon/3. Python for
 ↪  data analysis/practice/data/salesdata.csv')

df.index=pd.to_datetime(df['Date'])
```

Exercise 1

- Plot the data. Is it stationary? Why?

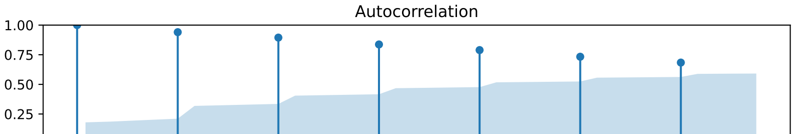
```
df['Sales'].plot()
plt.show()
```



Exercise 1

- Plot the autocorrelation and partial autocorrelation coefficients and discuss if the process is stationary

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 1, 1)
fig =
↪ sm.graphics.tsa.plot_acf(df['Sales'].diff().dropna(),
↪ lags=40, ax=ax1)
ax2 = fig.add_subplot(2, 1, 2)
fig =
↪ sm.graphics.tsa.plot_pacf(df['Sales'].diff().dropna(),
↪ lags=40, ax=ax2)
plt.show()
```



Exercise 1

- Plot the autocorrelation and partial autocorrelation coefficients and discuss if the process is stationary

```
from statsmodels.tsa.stattools import adfuller

def adf_test(timeseries):
    print("Results of Dickey-Fuller Test:")
    dftest = adfuller(timeseries, autolag="AIC")
    dfoutput = pd.Series(
        dftest[0:4],
        index=[
            "Test Statistic",
            "p-value",
            "#Lags Used",
            "Number of Observations Used",
        ],
    )
    for key, value in dftest[4].items():
        dfoutput["Critical Value (%s)" % key] = value
```

Exercise 1

- Plot the autocorrelation and partial autocorrelation coefficients and discuss if the process is stationary

```
adf_test(df['Sales'])
```

```
Results of Dickey-Fuller Test:
Test Statistic                   -0.702501
p-value                           0.846101
#Lags Used                       11.000000
Number of Observations Used     108.000000
Critical Value (1%)              -3.492401
Critical Value (5%)              -2.888697
Critical Value (10%)             -2.581255
dtype: float64
```

```
adf_test(df['Sales'].diff().diff(periods=6).dropna())
```

```
Results of Dickey-Fuller Test:
Test Statistic                 -1.015100e+01
```

Exercise 1

- Fit an appropriate SARIMA model consistent with the previous ACF and PACF profiles

```
model=sm.tsa.statespace.SARIMAX(endog=df['Sales'],
↪  order=(0, 1, 0), seasonal_order=(0, 0, 1, 6))
results=model.fit()
print(results.summary())
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  9.04016D+00    |proj g|=  8.95066D-0
```

Exercise 1

- Is the fitted model good enough?

```
from scipy import stats

resid = results.resid
stats.normaltest(resid)
```

```
NormaltestResult(statistic=67.70963068189998, pvalue=1.9817
```

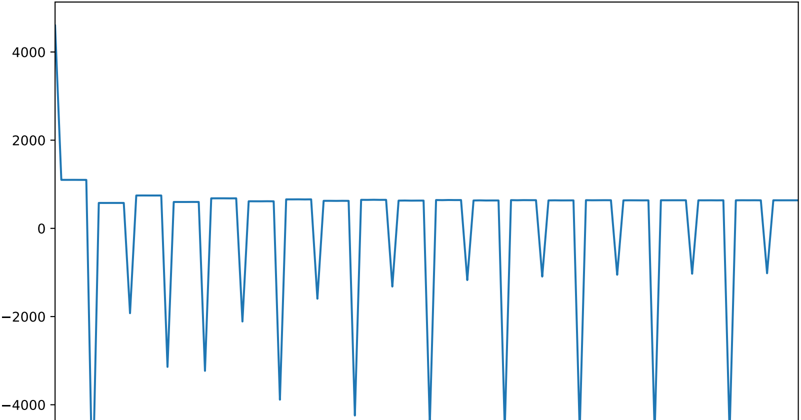
```
sm.stats.durbin_watson(resid)
```

```
2.1576221381903653
```

Exercise 1

- Is the fitted model good enough?

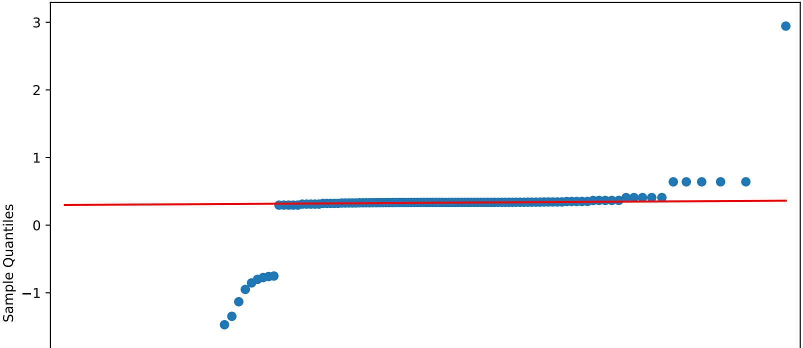
```
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
ax = results.resid.plot(ax=ax)
```



Exercise 1

- Is the fitted model good enough?

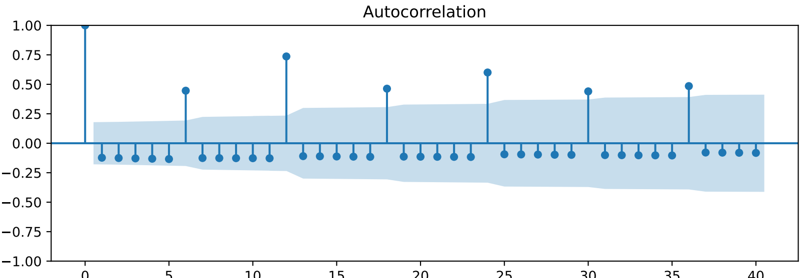
```
from statsmodels.graphics.api import qqplot

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)
fig = qqplot(resid, line="q", ax=ax, fit=True)
```



Exercise 1

- Is the fitted model good enough?

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 1,1)
fig = sm.graphics.tsa.plot_acf(resid, lags=40,
 ↪  ax=ax1)
ax2 = fig.add_subplot(2, 1, 2)
fig = sm.graphics.tsa.plot_pacf(resid, lags=40,
 ↪  ax=ax2)
```



Exercise 2

- Try to extend the Bayesian model introduced before in order to reproduce the linear model to explain the price of the train ticket according to the fare, recategorizing this variable in two categories (“Flexible”, “Adulto ida” and “Mesa” together). Fit the regular linear regression model in Python and compare the estimates.

```
import pandas as pd
import os

os.chdir("/home/dmorina/Insync/dmorina@ub.edu/OneDrive
 ↪ Biz/Docència/UB/2023-2024/PyEcon/3. Python for
 ↪ data analysis/practice/data/")
renfe = pd.read_csv("renfe.csv")
renfe.head()
```

	insert_date	origin	destination	start_date
0	2019-04-22 08:00:25	MADRID	SEVILLA	2019-04-28 08:30:0

Exercise 2

Categorical variables with more than two categories have to be expressed as a combination of dummy variables.

```
renfe['fare1'] = 0
renfe.loc[renfe['fare'] == "Adulto ida", 'fare1'] = 1
renfe.loc[renfe['fare'] == "Flexible", 'fare1'] = 1
renfe.loc[renfe['fare'] == "Mesa", 'fare1'] = 1
renfe.head()
```

	insert_date	origin	destination	start_date
0	2019-04-22 08:00:25	MADRID	SEVILLA	2019-04-28 08:30:0
1	2019-04-22 10:03:24	MADRID	VALENCIA	2019-05-20 06:45:0
2	2019-04-25 19:19:46	MADRID	SEVILLA	2019-05-29 06:20:0
3	2019-04-24 06:21:57	SEVILLA	MADRID	2019-05-03 08:35:0
4	2019-04-19 21:13:55	VALENCIA	MADRID	2019-05-10 09:40:0

Exercise 2

```
import rpy2.robjects as robjects
from rpy2.robjects import r, pandas2ri
from rpy2.robjects.packages import importr
pandas2ri.activate()

# import the jags package
R2jags = importr('R2jags')

robjects.globalenv["renfe"] = renfe
robjects.r('''
model <- function() {
  # likelihood
  for (i in 1:N) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- beta0 + beta1*fare1[i]
  }
  # priors
  beta0 ~ dnorm(0, 0.01)
```

Exercise 2

```
r_f = robjects.globalenv['fit']
print(r_f)
```

```
Inference for Bugs model at "/tmp/RtmpYAGPRL/modelab1c67ad6
 5 chains, each with 100 iterations (first 10 discarded), n
 n.sims = 45 iterations saved
            mu.vect   sd.vect       2.5%        25%
beta0        61.559     0.400     60.582     61.425     61.
beta1         5.876     0.912      4.134      5.229      6.
sigma        51.459    35.495     23.955     24.118     25.
deviance 254989.615 22315.885 237512.873 237514.397 237575.
              97.5%  Rhat n.eff
beta0        62.501 1.000    45
beta1         8.094 1.011    45
sigma       123.785 0.948    45
deviance 297012.097 0.948    45

For each parameter, n.eff is a crude measure of effective s
```

Exercise 2

```
import numpy as np
import statsmodels.formula.api as sm

model = sm.ols("price ~ fare1", data=renfe).fit()
print(model.summary())
```

```
                              OLS Regression Results
=========================================================
Dep. Variable:                      price   R-squared:
Model:                                OLS   Adj. R-squared:
Method:                     Least Squares   F-statistic:
Date:                dt., 19 de març 2024   Prob (F-statisti
Time:                            16:00:05   Log-Likelihood:
No. Observations:                   25798   AIC:
Df Residuals:                       25796   BIC:
Df Model:                               1
Covariance Type:                nonrobust
=========================================================
```

Exercise 3

- Import the files "TravelTimes_to_5975375_RailwayStation.shp", "metro.shp" and "roads.shp" into the Python session

```
import geopandas as gpd
import matplotlib.pyplot as plt

# Filepaths
grid_fp =
↪ r"/home/dmorina/Insync/dmorina@ub.edu/OneDrive
↪ Biz/Docència/UB/2023-2024/PyEcon/3. Python for
↪ data
↪ analysis/practice/data/TravelTimes_to_5975375_RailwaySt
roads_fp =
↪ r"/home/dmorina/Insync/dmorina@ub.edu/OneDrive
↪ Biz/Docència/UB/2023-2024/PyEcon/3. Python for
↪ data analysis/practice/data/roads.shp"
metro_fp =
↪ r"/home/dmorina/Insync/dmorina@ub.edu/OneDrive
```

Exercise 3

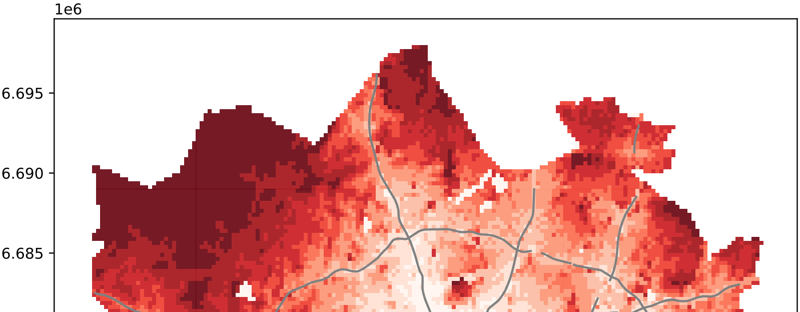
- Visualize metro and roads travel times on top of "car_r_t" column

```
# Get the CRS of the grid
gridCRS = grid.crs

# Reproject geometries using the crs of travel time
↪  grid
roads['geometry'] =
↪  roads['geometry'].to_crs(crs=gridCRS)
metro['geometry'] =
↪  metro['geometry'].to_crs(crs=gridCRS)
```

Exercise 3

- Visualize metro and roads travel times on top of "car_r_t" column

```
my_map = grid.plot(column="car_r_t", linewidth=0.03,
    cmap="Reds", scheme="quantiles", k=9, alpha=0.9)
roads.plot(ax=my_map, color="grey", linewidth=1.5)
metro.plot(ax=my_map, color="red", linewidth=2.5)
plt.show()
```



Exercise 3

- Save the figure as a *png* file with resolution of 1200 dpi

```
my_map = grid.plot(column="car_r_t", linewidth=0.03,
↪  cmap="Reds", scheme="quantiles", k=9, alpha=0.9)
roads.plot(ax=my_map, color="grey", linewidth=1.5)
metro.plot(ax=my_map, color="red", linewidth=2.5)
plt.savefig("/home/dmorina/Insync/dmorina@ub.edu/OneDrive
↪  Biz/Docència/UB/2023-2024/PyEcon/3. Python for
↪  data analysis/practice/static_map.png", dpi=1200)
```