Machine Learning for Economics and Finance

Python Exercises

Ole Wilms

April 24, 2024

Contents

Important Instructions

- The purpose of these exercises is to get to know Python by solving some basic programming exercises
- In case you struggle with some problems, please post your questions on the OpenOlat Forum.
- Particularly difficult questions are marked by (D). Don't worry if you cannot solve these questions right away. Throughout the course, these programming concepts will become easier to understand.
- Sample solutions to the exercises will be provided next week. However, I strongly encourage all students to work on the exercises beforehand.

Task 1: Constructing a dataset

1. Create different kinds of vectors with 6 entries each:
   - vector a: a vector with only ones (hint: you can use the `np.repeat()` function)
   - vector b: a vector of integers that goes from 1 to 6 (hint: you can use the `np.arange()` function)
   - vector c: a vector where each entry is drawn from a normal distribution with mean 2 standard deviation 5.
   - vector d: a vector where each entry consists of one of the words in "*Machine Learning for Economics and Finance*".

[1]:
```
import numpy as np

n = 6
a = np.repeat(1, 6)
b = np.arange(1, n+1)
c = np.random.normal(2, 5, size=6)
d = ["Machine", "Learning", "for", "Economics", "and", "Finance"]
```

2. Stack vector b into a matrix $M1$ of dimension 2 x 3 where you fill in by column. Stack the same vector into a matrix $M2$ of dimension 3 x 2 where you fill in by row.

[2]:
```
M1 = np.reshape(b, (2, 3))
M2 = np.reshape(b, (3, 2))
```

3. Add the two matrices. You will obtain an error message. What's going wrong? Solve the problem using the transpose function `np.transpose()`.

[3]:
```
#result = M1 + M2                  # Both matrices have different dimensions,
result = np.transpose(M1) + M2     # so we need to reshape (transpose) them.
print(result)
```

```
[[ 2  6]
 [ 5  9]
 [ 8 12]]
```

4. Create a vector *train_sample* with 4 entries by randomly sampling 4 values from vector b without replacement (that is, you cannot draw the same number twice). For this you can use the function `np.random.choice()`. Run the code that creates the vector multiple times. Explain what's happening. Fix the issue by using the function `np.random.seed()`.

[4]:
```
# Set the random seed for reproducibility
np.random.seed(2)

# Randomly sample 4 values from vector b without replacement
sample_train = np.random.choice(b, size=4, replace=False)
print(sample_train)
```

```
[5 2 4 3]
```

5. Put vectors a, b, c and d together in a dataframe called *df*.

[5]:
```
import pandas as pd

# Combine arrays into a DataFrame
df = pd.DataFrame({'a': a, 'b': b, 'c': c, 'd': d})
print(df)
```

```
   a  b          c          d
0  1  1   8.371047    Machine
1  1  2  10.463605   Learning
2  1  3   2.738497        for
3  1  4  -4.247608  Economics
4  1  5  14.106393        and
5  1  6   0.939076    Finance
```

6. Name the columns of *df* *'Ones'*, *'Seq'*, *'Normal'* and *'Coursename'* respectively (hint: you can use the function `pd.DataFrame()`). Provide a summary of the dataframe using the `describe()`function.

[6]:
```
# Combine arrays into a DataFrame with named columns
df = pd.DataFrame({'Ones': a, 'Seq': b, 'Normal': c, 'Coursename': d})
print(df)
```

```
   Ones  Seq     Normal Coursename
0     1    1   8.371047    Machine
1     1    2  10.463605   Learning
2     1    3   2.738497        for
3     1    4  -4.247608  Economics
4     1    5  14.106393        and
5     1    6   0.939076    Finance
```

[7]:
```
# Provide a summary of the DataFrame
summary = df.describe()
# Alternative:  summary = df.describe(include='all')
# Note: Coursename is a object (string) column
print(summary)
```

```
       Ones       Seq     Normal
count   6.0  6.000000   6.000000
mean    1.0  3.500000   5.395168
std     0.0  1.870829   6.787166
min     1.0  1.000000  -4.247608
25%     1.0  2.250000   1.388931
50%     1.0  3.500000   5.554772
75%     1.0  4.750000   9.940466
max     1.0  6.000000  14.106393
```

7. (D) Add a column called *'Int'* to the dataframe which checks whether column *'Normal'* is larger than 0. If that is the case *'Int'* should contain a *TRUE*, if that is not the case *'Int'* should contain a FALSE. Proceed as follows:

- Create a new column named *'Int'* in the DataFrame, initializing all elements to True. Use a loop to iterate through each row of the DataFrame. For each row, check if the corresponding value in the *'Normal'* column is greater than 0. If it is, retain the *TRUE* value in the *'Int'* column; otherwise, replace it with *FALSE*."

[8]:
```
# Add a column 'Int' to the DataFrame with all elements set to True
df['Int'] = True

# Write a loop to iterate through each row of the DataFrame
for index, row in df.iterrows():
    # Check if the value in the 'Normal' column for the current row is
    # larger than 0
    if row['Normal'] > 0:
        # If yes, set the corresponding value in the 'Int' column to True
        df.at[index, 'Int'] = True
    else:
        # If no, set the corresponding value in the 'Int' column to False
        df.at[index, 'Int'] = False

print(df)
```

```
   Ones  Seq     Normal Coursename    Int
0     1    1   8.371047    Machine   True
1     1    2  10.463605   Learning   True
2     1    3   2.738497        for   True
3     1    4  -4.247608  Economics  False
4     1    5  14.106393        and   True
5     1    6   0.939076    Finance   True
```

8. (D) Can you think of an easier way to construct the column *'Int'* instead of the loop described above? If yes, add this column and call it *'Int2'*

[9]:
```
# Add a column 'Int2' to the DataFrame based on the condition
df['Int2'] = df['Normal'] > 0
print(df)
```

```
   Ones  Seq     Normal Coursename    Int   Int2
0     1    1   8.371047    Machine   True   True
1     1    2  10.463605   Learning   True   True
2     1    3   2.738497        for   True   True
3     1    4  -4.247608  Economics  False  False
4     1    5  14.106393        and   True   True
5     1    6   0.939076    Finance   True   True
```

9. (D) Now we use our vector *train_sample* to construct two distinct datasets from *df*. The numbers in *train_sample* refer to the rows of our dataframe *df* that we want to use for the first dataset while all other rows can be used for the second dataset. Construct a new dataframe called *df_train* that only contains the rows in *train_sample*. Note that you can simply use square brackets to extract rows from a dataframe. Make sure that you extract all columns but only the rows that are in *train_sample*. Your object *df_train* should have 4

rows and as many columns as *df*.

[10]:
```
df_train = df.loc[sample_train]
print(df_train)
```

```
   Ones  Seq     Normal Coursename    Int   Int2
5     1    6   0.939076    Finance   True   True
2     1    3   2.738497        for   True   True
4     1    5  14.106393        and   True   True
3     1    4  -4.247608  Economics  False  False
```

10. (D) Construct another dataframe called *df_test* which contains the other two rows of *df* that are not in *df_train*. Note that you can use `~df.index.isin()` to select all rows that are *NOT* in *train_sample*.

[11]:
```
df_test = df.loc[~df.index.isin(sample_train)]
print(df_test)
```

```
   Ones  Seq     Normal Coursename   Int  Int2
0     1    1   8.371047    Machine  True  True
1     1    2  10.463605   Learning  True  True
```

Task 2: Working data from the *ISLR2* library

1. Install and load the library *ISLP*.

[12]:
```
# Note: There are different options based on your OS!
# https://islp.readthedocs.io/en/latest/installation.html

#pip install ISLP
```

[13]:
```
import ISLP
```

2. Load the dataset *Auto* and save it into an object called *Auto*. Use the help function to obtain information about the variables in *Auto*.

[14]:
```
from ISLP import load_data

# Load the Auto dataset
Auto = load_data('Auto')

# Obtain information about the variables in Auto
#help(Auto)
```

3. Provide a summary of *Auto* using the `describe()` function. Do you think all the variables in *Auto* could be readily used for a linear regression model?

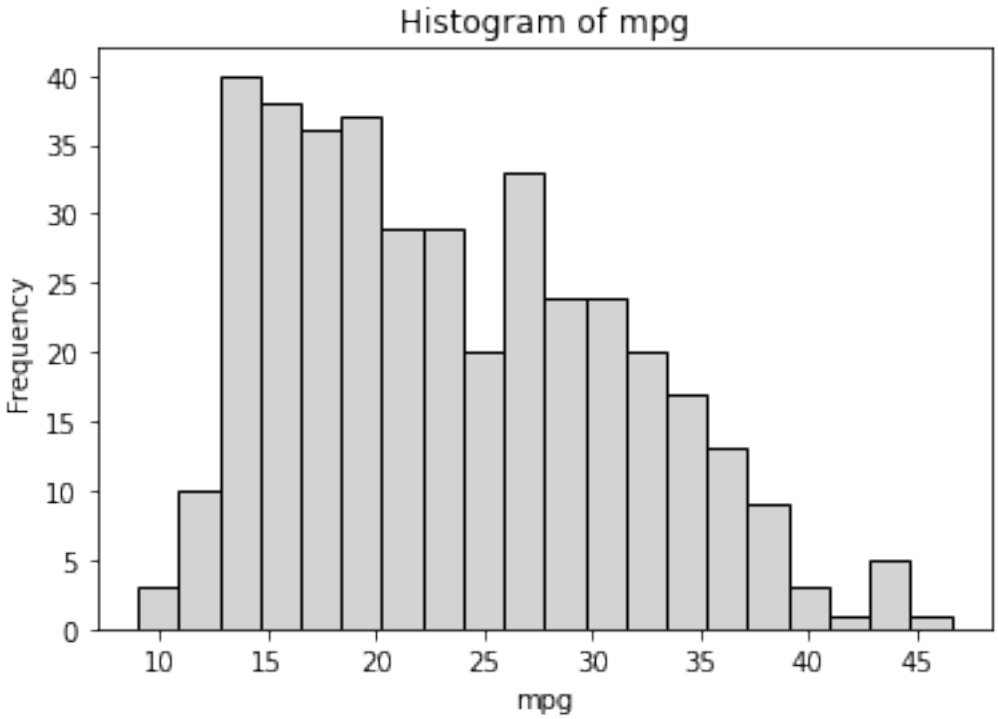
[15]:
```
# Note: You can transpose '.T' the output to optain a horizontal output.
print(Auto.describe().T)
```

```
              count         mean         std     min       25%      50%  \
mpg           392.0    23.445918    7.805007     9.0    17.000    22.75
cylinders     392.0     5.471939    1.705783     3.0     4.000     4.00
displacement  392.0   194.411990  104.644004    68.0   105.000   151.00
horsepower    392.0   104.469388   38.491160    46.0    75.000    93.50
weight        392.0  2977.584184  849.402560  1613.0  2225.250  2803.50
acceleration  392.0    15.541327    2.758864     8.0    13.775    15.50
year          392.0    75.979592    3.683737    70.0    73.000    76.00
origin        392.0     1.576531    0.805518     1.0     1.000     1.00

                  75%     max
mpg            29.000    46.6
cylinders       8.000     8.0
displacement  275.750   455.0
horsepower    126.000   230.0
weight       3614.750  5140.0
acceleration   17.025    24.8
year           79.000    82.0
origin          2.000     3.0
```

4. The goal of the following exercises is to understand the relation between the variable *'mpg'* and *'horsepower'*:

- Provide a histogram of *'mpg'* using the function `hist()`. Hint: For creating plots and visualizations, the `matplotlib` package is a common choice.
- Compute the pearson correlation between *'mpg'* and *'horsepower'*. For this, first select the two respective columns using `Auto["mpg","horsepower"]` and then use the function `corr()`. Is there a positive or negative relationship between the two variables?
- Provide a plot with *'horsepower'* on the x-axis and *'mpg'* on the y-axis. Do you think a linear regression model is well suited to predict *'mpg'* using *'horsepower'* ?

[16]:
```
import matplotlib.pyplot as plt

# Histogram of 'mpg'
plt.figure(figsize=(6, 4))
plt.hist(Auto['mpg'], bins=20, color='lightgray', edgecolor='black')
plt.xlabel('mpg')
plt.ylabel('Frequency')
plt.title('Histogram of mpg')
plt.show()
```

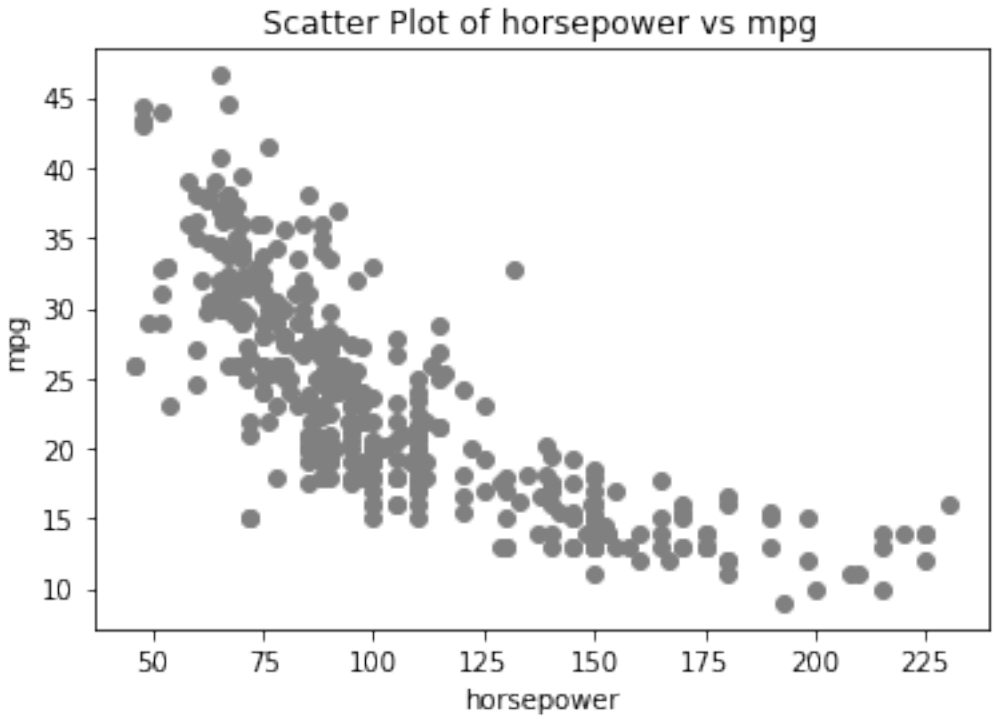


[17]:
```
# Pearson correlation between 'mpg' and 'horsepower'
pearson_corr = Auto[['mpg', 'horsepower']].corr().loc['mpg', 'horsepower']

print(f"Pearson correlation between mpg and horsepower: {pearson_corr}")
print(f"Pearson correlation between mpg and horsepower: {pearson_corr:.3f}")
# Note: The f prefix before the string indicates that it's an f-string.
```

```
# Inside the string, you can embed Python expressions by enclosing them
# in curly braces {}.
# Inside the expression you can call a variable and formats the value as
# a floating-point number with e.g. 3 decimal places.

# So, in the given example:
# {pearson_corr} embeds the value of the variable pearson_corr into the string.
# :.3f formats the embedded value as a floating-point number with 3 decimal␣
 ↪places.
```

```
Pearson correlation between mpg and horsepower: -0.7784267838977761
Pearson correlation between mpg and horsepower: -0.778
```

[18]:
```
# Plot of 'horsepower' vs 'mpg'
plt.figure(figsize=(6, 4))
plt.scatter(Auto['horsepower'], Auto['mpg'], color='gray')
plt.xlabel('horsepower')
plt.ylabel('mpg')
plt.title('Scatter Plot of horsepower vs mpg')
plt.show()
```



Task 3: Working with external data

1. Load the dataset `'return_data.csv'` which contains historical returns of Apple (*'ret_apple'*), the index return of the *S&P500* which is a broad portfolio of stocks in the US (*'ret_index'*), as well as the return of a riskless investment in government bonds (*'rf'*). Make sure that you set the right working director when you try to load in the data. In the dataset, a number of 0.1 corresponds to a return of 10%.

[19]:
```
import os

# Get the directory of the current Jupyter notebook file
notebook_dir = os.path.dirname(os.path.abspath('__file__'))

# Set the working directory to the directory of the Jupyter notebook file
os.chdir(notebook_dir)
```

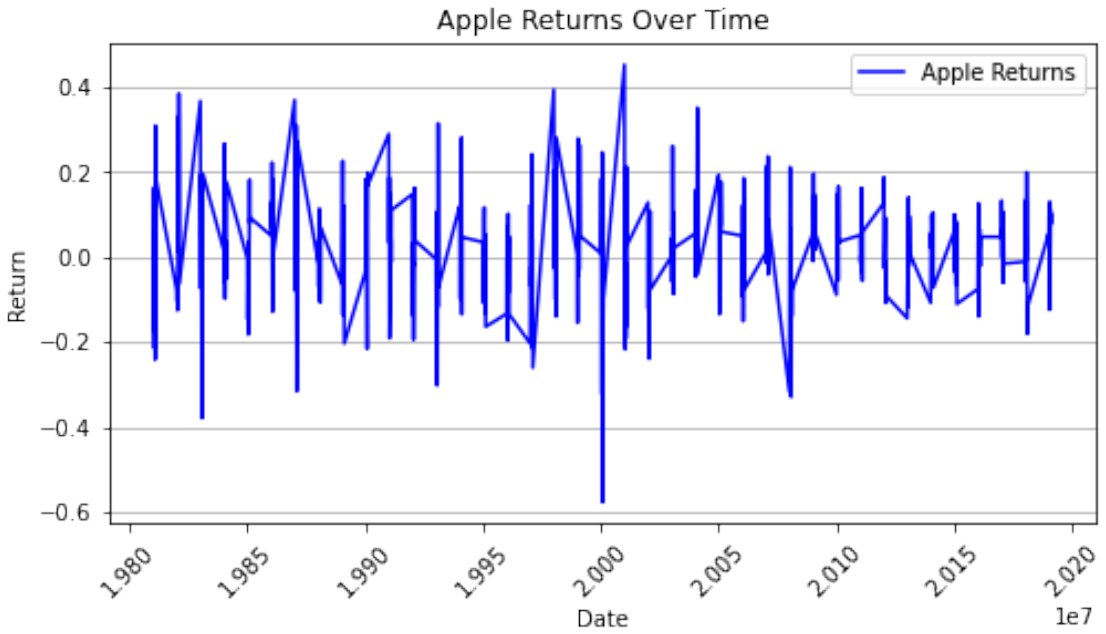
[20]:
```
# Load the dataset
df = pd.read_csv('return_data.csv')
df.head()
```

[20]:

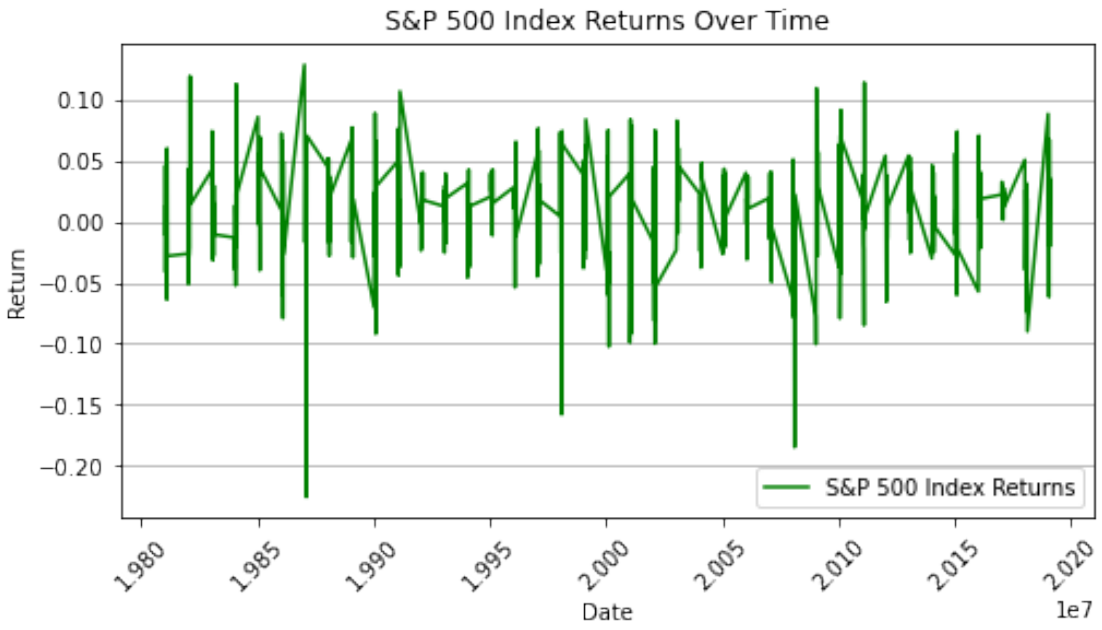
	date	ret_apple	ret_index	rf
0	19810130	-0.170018	-0.040087	0.0104
1	19810227	-0.061674	0.015521	0.0107
2	19810331	-0.075117	0.046184	0.0121
3	19810430	0.157360	-0.011268	0.0108
4	19810529	0.164474	0.013550	0.0115

2. To get to know the data, construct three plots each having the date on the x-axis and the respective return time series on the y-axis.

[21]:
```
# Plot for Apple returns
plt.figure(figsize=(8, 4))
plt.plot(df['date'], df['ret_apple'], color='blue', label='Apple Returns')
plt.xlabel('Date')
plt.ylabel('Return')
plt.title('Apple Returns Over Time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(axis='y')
plt.show()
```



[22]:
```
# Plot for S&P 500 index returns
plt.figure(figsize=(8, 4))
plt.plot(df['date'], df['ret_index'], color='green', label='S&P 500 Index␣
 ↪Returns')
plt.xlabel('Date')
plt.ylabel('Return')
plt.title('S&P 500 Index Returns Over Time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(axis='y')
plt.show()
```



[23]:
```
# Plot for risk-free returns
plt.figure(figsize=(8, 4))
plt.plot(df['date'], df['rf'], color='red', label='Risk-Free Returns')
plt.xlabel('Date')
plt.ylabel('Return')
plt.title('Risk-Free Returns Over Time')
plt.xticks(rotation=45)
plt.legend()
plt.grid(axis='y')
plt.show()
```

3. Compute the means and the standard deviations of the three time series and interpret the results.

[24]:
```
# Compute means and standard deviations
mean_returns = df[['ret_apple', 'ret_index', 'rf']].mean()
std_returns = df[['ret_apple', 'ret_index', 'rf']].std()

print("Mean Returns:")
print(mean_returns)
print("\nStandard Deviations:")
print(std_returns)
```

```
Mean Returns:
ret_apple    0.022536
ret_index    0.009536
rf           0.003318
dtype: float64

Standard Deviations:
ret_apple    0.131109
ret_index    0.043376
rf           0.002795
dtype: float64
```

4. What was the maximum loss in a single month when holding Apple stocks? What are the maximum losses for the *S&P500* and the risk-free rate? Interpret.

[25]:
```
# Compute maximum losses
max_loss_apple = df['ret_apple'].min()
max_loss_sp500 = df['ret_index'].min()
max_loss_rf = df['rf'].min()

print(f"Maximum Loss for Apple stocks: {max_loss_apple:.3f}")
print(f"Maximum Loss for S&P 500 index: {max_loss_sp500:.3f}")
print(f"Maximum Loss for Risk-Free rate: {max_loss_rf:.3f}")
```

```
Maximum Loss for Apple stocks: -0.577
Maximum Loss for S&P 500 index: -0.225
Maximum Loss for Risk-Free rate: 0.000
```

5. Compute the pearson correlation between *'ret_apple'* and *'ret_index'* using the function `cor()`. Interpret the result.

[26]:
```
# Compute Pearson correlation between 'ret_apple' and 'ret_index'
pearson_corr = df['ret_apple'].corr(df['ret_index'])

print(f"Pearson correlation between ret_apple and ret_index: {pearson_corr:.
↪3f}")
```

```
Pearson correlation between ret_apple and ret_index: 0.466
```