SPC707P Machine and Deep Learning — Week 02 Project

Kavit Tolia

October 3, 2025

1 Exercise 1

Write an expression to calculate the variance of `my_array` and compare it to `np.var(my_array)`. First, let's recall the definition of the variance:

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

One can notice that this is basically the mean of the array $(x_i - \overline{x})^2$. You can do this in 4 steps:

1. Write an expression for the mean of `my_array`
2. Define a new array as the difference between `my_array` and its mean
3. Square your new array
4. Calculate the mean of `new_array_squared`. Compare this to the variance calculated with `numpy`.

Solution.

```
import numpy as np
my_array = np.array([10, 15, 15, 20, 25, 30, 32])
arr_mean = np.sum(my_array) / my_array.size
new_array = my_array - arr_mean
new_array_squared = new_array ** 2
my_array_var = np.sum(new_array_squared) / new_array_squared.size
print(f"My calculated variance = {my_array_var:.2f}, using numpy we get {np.var(my_array):.2f}")
```

Output: `My calculated variance = 58.86, using numpy we get 58.86`

2 Exercise 2

Write an expression to find the mode of `my_array`.
Solution.

```
import numpy as np
my_array = np.array([10, 15, 15, 20, 25, 30, 32])
unique_arr = np.unique(my_array)
freq_arr = np.zeros(unique_arr.size)
for idx, val in enumerate(unique_arr):
    freq_arr[idx] = np.sum(my_array == val)
max_freq = np.max(freq_arr)
my_mode = unique_arr[freq_arr == max_freq]
if len(my_mode) == 1:
    print(f"The mode of my_array is {my_mode[0]}")
else:
    print(f"The modes of my_array are {my_mode}")
```

Output: `The mode of my_array is 15`

3 Exercise 3

1. Define a simple function to greet someone (use default values for arguments).
2. Call the function without any input to verify that the default values work.
3. Call the function with different inputs to verify that the function works as expected.
4. Define another function that determines whether a number is even or odd.
5. Call the function and store the result, then print the result.
6. Define a function that calls the first two functions and greets someone and displays whether their age is even or odd.

Solution.

```
def say_hello(output='Hello World!'):
    print(output)

say_hello()
say_hello('Now say this!')

def even_or_odd(num):
    if (num % 2 == 0):
        return 'even'
    else:
        return 'odd'

test_num = 3
result = even_or_odd(test_num)
print(f"The number {test_num} is {result}.")

def greet_person(age, greeting='Hello World!'):
    say_hello(greeting)
    print(f"Your age is {even_or_odd(age)}")

greet_person(4)
```

Output:

```
Hello World!
Now say this!
The number 3 is odd.
Hello World!
Your age is even
```

4 Exercise 4

The purpose of this exercise is to get you to play around with `pandas` `DataFrame` and to consolidate the knowledge that you already have.

- Generate 5 samples with 100,000 correlated random numbers distributed according to Gaussian distributions (you can choose whatever covariance matrix you like, although you can start with an identity matrix for simplicity). You may want to use the `numpy` function `random.multivariate_normal` for this purpose.
- Read these into a DataFrame.
- Create a 6^{th} column in your DataFrame: the values should be the second column plus the fourth column.
- Verify that the covariance (and correlation) matrices are what you would expect.

- Display your data.

Solution.

```
import numpy as np
import pandas as pd
num_var = 5
means = np.zeros(num_var)
cov_matrix = np.identity(num_var)
num_samples = 100000
col_names = ['Var1', 'Var2', 'Var3', 'Var4', 'Var5']
random_multivariate = np.random.multivariate_normal(means, cov_matrix, num_samples)
df = pd.DataFrame(random_multivariate, columns=col_names)
df = df.assign(
    Var6 = df['Var2'] + df['Var4']
)
print("The covariances are what we expect within sampling error. The variance of Var6 is 2 which is correct as it's the sum of 1 and 1")
display(df.cov(numeric_only=True))
display(df.head())
```

Output: `The covariances are what we expect within sampling error. The variance of Var6 is 2 which is correct as it's the sum of 1 and 1`

	Var1	Var2	Var3	Var4	Var5	Var6
Var1	0.994469	0.000490	0.004513	0.000934	-0.001600	0.001423
Var2	0.000490	0.994359	0.006232	-0.000122	-0.000678	0.994237
Var3	0.004513	0.006232	1.007970	0.002121	-0.004293	0.008353
Var4	0.000934	-0.000122	0.002121	1.002105	-0.000528	1.001983
Var5	-0.001600	-0.000678	-0.004293	-0.000528	0.997289	-0.001205
Var6	0.001423	0.994237	0.008353	1.001983	-0.001205	1.996220

	Var1	Var2	Var3	Var4	Var5	Var6
0	1.254480	-0.315118	0.482757	2.670053	1.055085	2.354936
1	0.386099	0.259366	0.424522	-0.021064	0.006300	0.238302
2	0.227913	1.515974	-1.020592	-0.443084	-0.058855	1.072891
3	-0.205500	-2.201970	-0.334635	1.025902	0.734656	-1.176068
4	-0.347330	-0.185261	1.599332	-0.915768	0.105614	-1.101029

5 Exercise 5

Let's analyse some data from the euro/dollar and pound/dollar exchange rates.

1. Read the euro–dollar exchange rate file into a DataFrame and merge it with the pound–dollar DataFrame. You may want to use the `pandas rename` function to rename the columns.
2. Check the dataset for `NaN` and drop any instances of `NaN`.
3. Plot histograms and time series from the DataFrame.
4. Calculate mean, variance, covariance, and correlation.

5. Plot a scatter plot of the euro–dollar vs. the pound–dollar exchange. You may notice the graph showing two separate populations. Use the slice function to try to separate the two populations *(hint: the ideal index will be between 2000 and 3000).*

6. Calculate the coefficients for the linear regression $y = mx + q$ for the three samples (the total sample, and the two samples found with the split in point 5), and plot them on top of a scatter graph:

$$m = \frac{\sigma_{xy}}{\sigma_x^2}$$

$$q = \overline{y} - m\overline{x}$$

   You may want to use `stats.linregress` from `scipy` to calculate the coefficients.

7. Write a sentence of your interpretation of the graph.

Solution.

1.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
df_eur = pd.read_csv('data/euro-dollar-exchange-rate-historical-chart.csv')
df_gbp = pd.read_csv('data/pound-dollar-exchange-rate-historical-chart.csv')
df = pd.merge(df_eur, df_gbp, on='date')
df.rename(columns={'␣value_x': 'eurusd', '␣value_y': 'gbpusd'}, inplace=True)
df.set_index('date', inplace=True)
df.head()
```

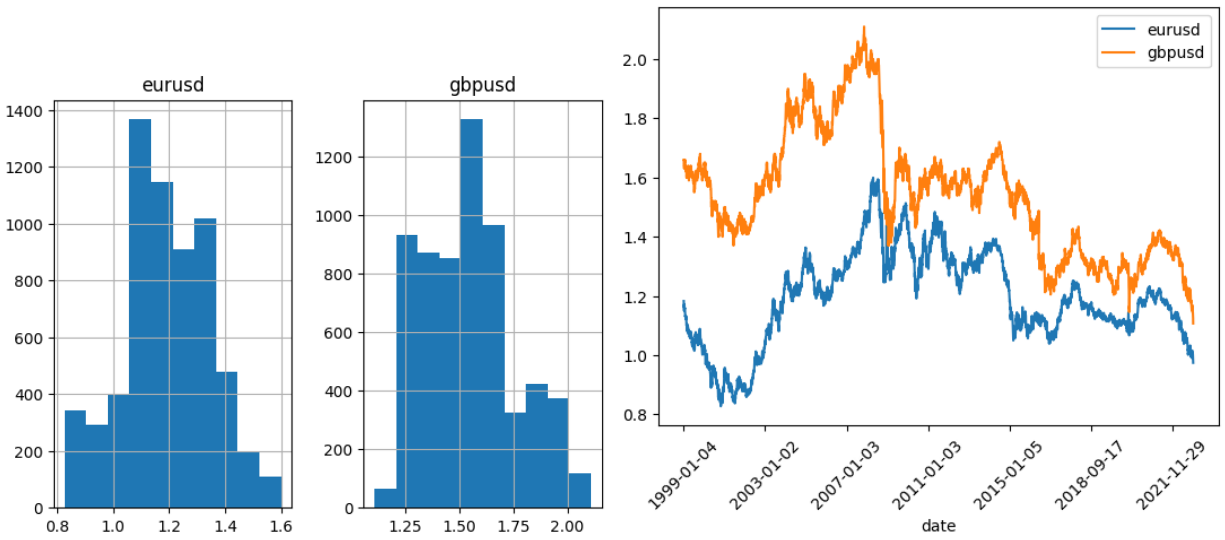
2.

```
df.info() # No NaNs found from this!
<class 'pandas.core.frame.DataFrame'>
Index: 6257 entries, 1999-01-04 to 2022-09-23
Data columns (total 2 columns):
# Column Non-Null Count Dtype
--- ------ -------------- -----
0 eurusd 6257 non-null float64
1 gbpusd 6257 non-null float64
dtypes: float64(2)
memory usage: 146.6+ KB
```

3.

```
df.hist()
df.plot(rot=45)
```

Figure shown on next page →



4.

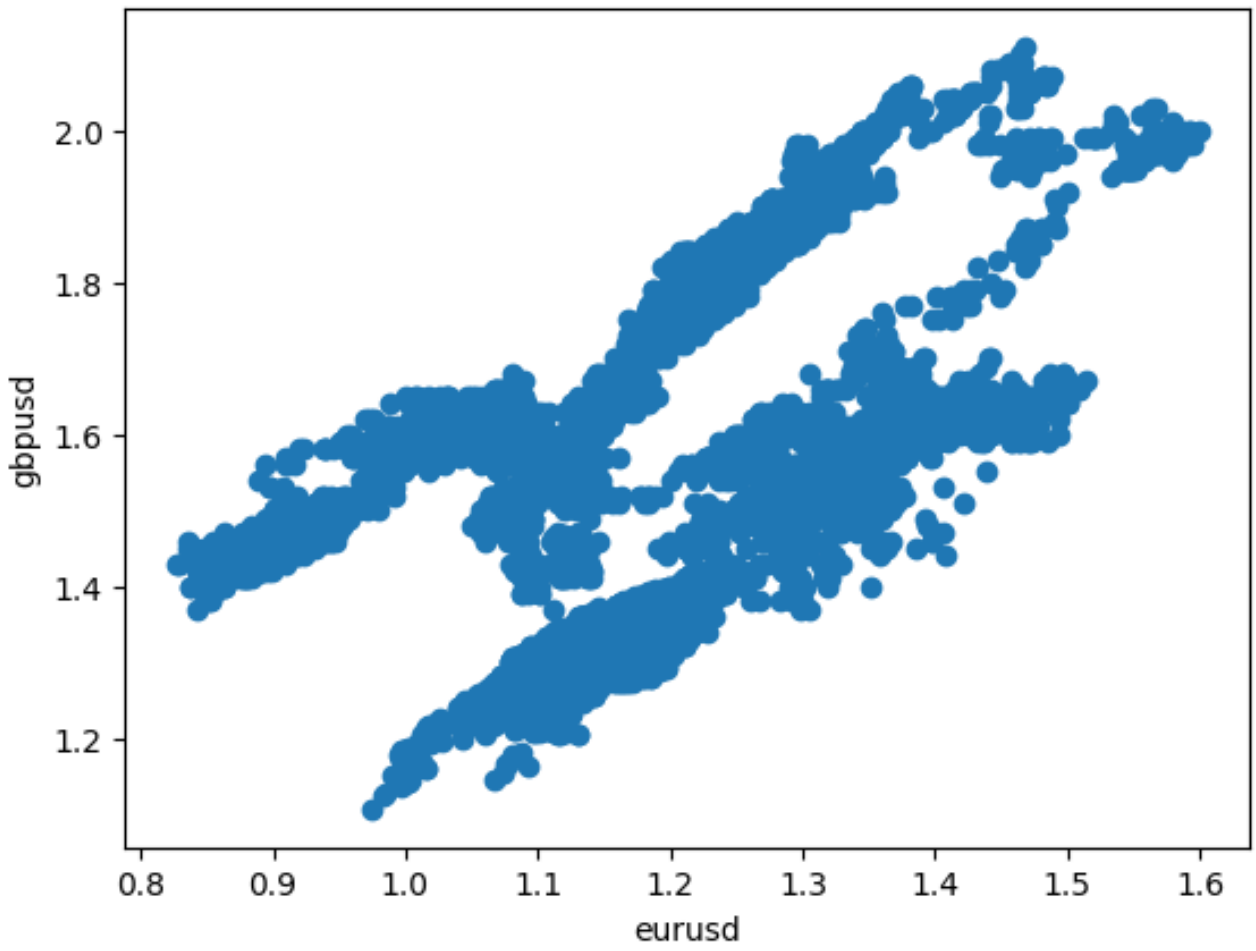
```
summary = pd.DataFrame({
    'mean': df.mean(),
    'std': df.std()
})
for col in df.columns:
    summary[f"corr_{col}"] = df.corr()[col]
    summary[f"cov_{col}"] = df.cov()[col]
summary
```

	mean	std	corr_eurusd	cov_eurusd	corr_gbpusd	cov_gbpusd
eurusd	1.192415	0.155381	1.000000	0.024143	0.554301	0.018375
gbpusd	1.543773	0.213346	0.554301	0.018375	1.000000	0.045517

5.

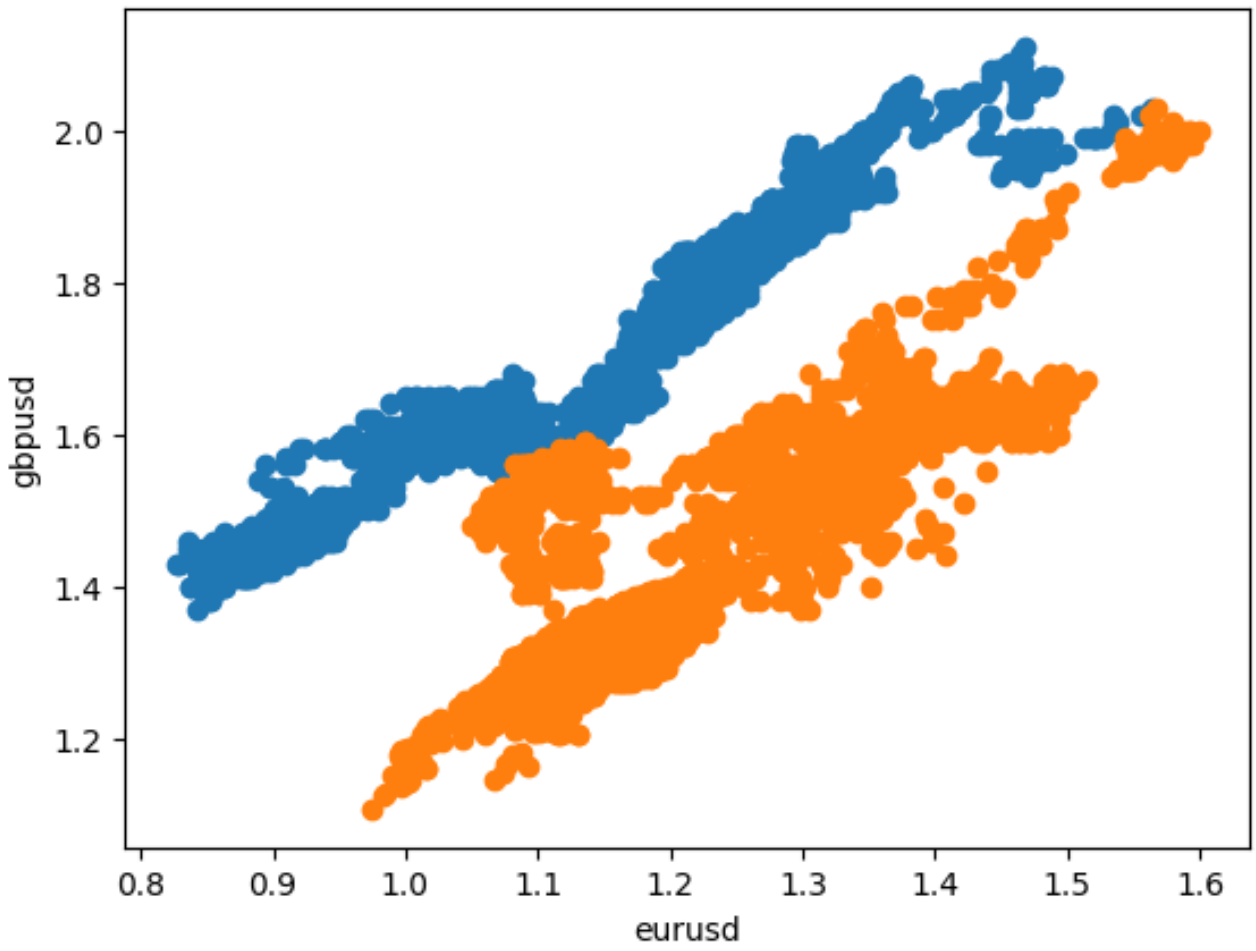
```
plt.scatter(df['eurusd'], df['gbpusd'])
plt.xlabel('eurusd')
plt.ylabel('gbpusd')
```

Figure shown on next page →




```
row_slice = slice(2300, df.shape[0])
post_df = df.iloc[row_slice]
pre_df = df.iloc[:row_slice.start]
plt.scatter(pre_df['eurusd'], pre_df['gbpusd'])
plt.scatter(post_df['eurusd'], post_df['gbpusd'])
plt.xlabel('eurusd')
plt.ylabel('gbpusd')
```

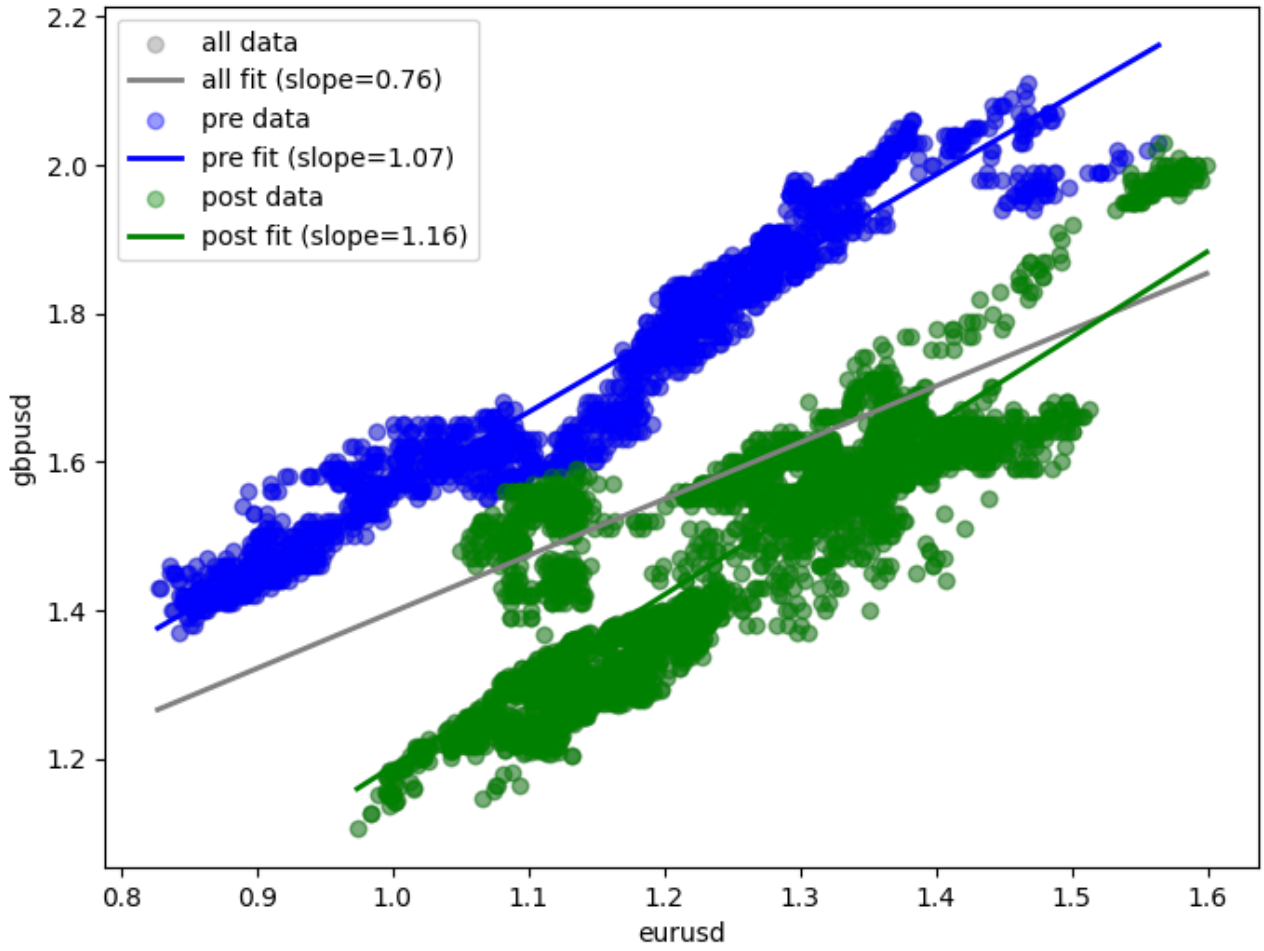
Figure shown on next page →



6.

```
datasets = {
    "all": df,
    "pre": pre_df,
    "post": post_df
}
colors = {
    "all": "gray",
    "pre": "blue",
    "post": "green"
}
plt.figure(figsize=(8,6))
for label, d in datasets.items():
    x = d["eurusd"]
    y = d["gbpusd"]
    plt.scatter(x, y, alpha=0.4, label=f"{label} data", color=colors[label])
    res = stats.linregress(x, y)
    x_fit = np.linspace(x.min(), x.max(), 100)
    y_fit = res.intercept + res.slope * x_fit
    plt.plot(x_fit, y_fit, color=colors[label], lw=2,
           label=f"{label} fit (slope={res.slope:.2f})")
plt.xlabel("eurusd")
plt.ylabel("gbpusd")
plt.legend()
plt.show()
```

Figure shown on next page →



7. There seems to be two different regimes (pre-2008 and post-2008), where the behaviour of the the variables seems to change somewhat. This makes it clear how important the data window is in any time series analysis. You could also say that a linear regression done using either pre-2008 or post-2008 data would provide a higher R^2 compared to the R^2 for the entire period.