5. Exploring Python

Exploring Python

Jupyter Notebook

Up till now we have been using the Spyder IDE or the Python coding environment. I hope you have become familiar with it. Now there are many others out there built for different purposes. Like PyCharm and VS Code, and many many more. However, if you are learning Python and applying it to some data science application, then you have to know about Jupyter Notebooks.

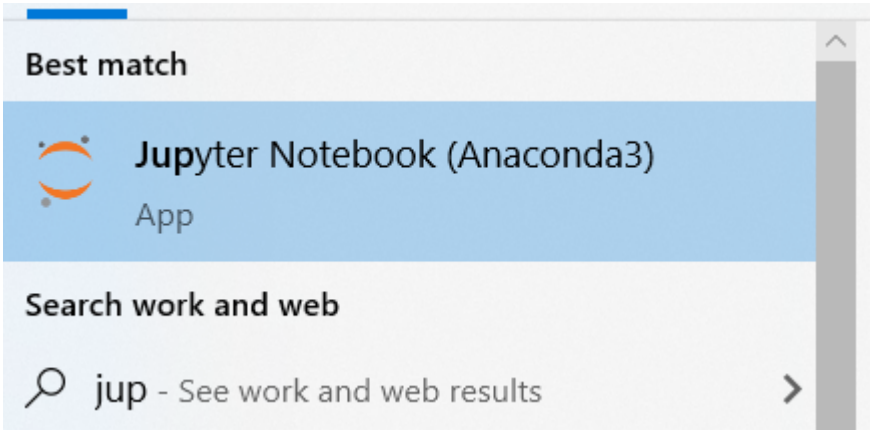
A Jupyter Notebook is an interactive web applications that facilitate creating and sharing documents that contain live code, equations, visualizations, and narrative text. They are widely used in data science, scientific research, and education.

Many teachers use it to present content, as you will see in week 2. Many tutorials on line use Jupyter Notebooks to convey ideas and principles. So it is quite important to understand how to use it, if you want to follow a tutorial or lesson.

There is a newer version of Jupyter Notebook called Jupyter Lab, but essentialy the same thing. The latter just have more features.

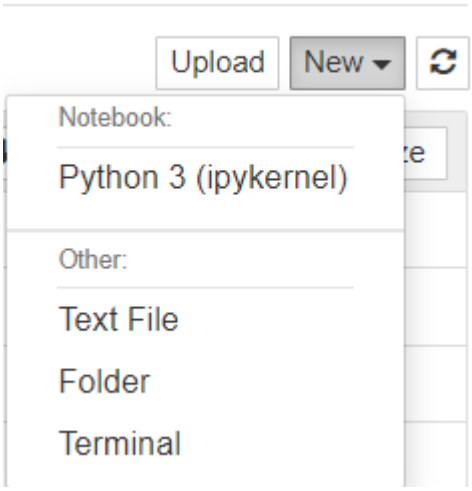
Getting Started

So the call thing about the Anaconda workspace you installed is that it has Jupyter Notebook installed already for you. So if go to you windows app, and type "Jupyter", you should see:

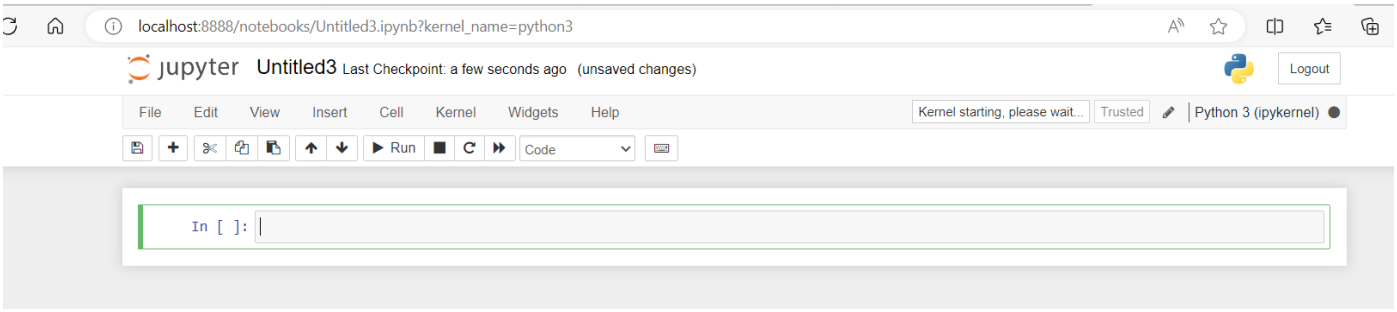


Click on that and you should see a windows terminal pop up and a browser window:

Go to the "New" tab and click on "Python 3 (ipykernal")



You should then see the following:



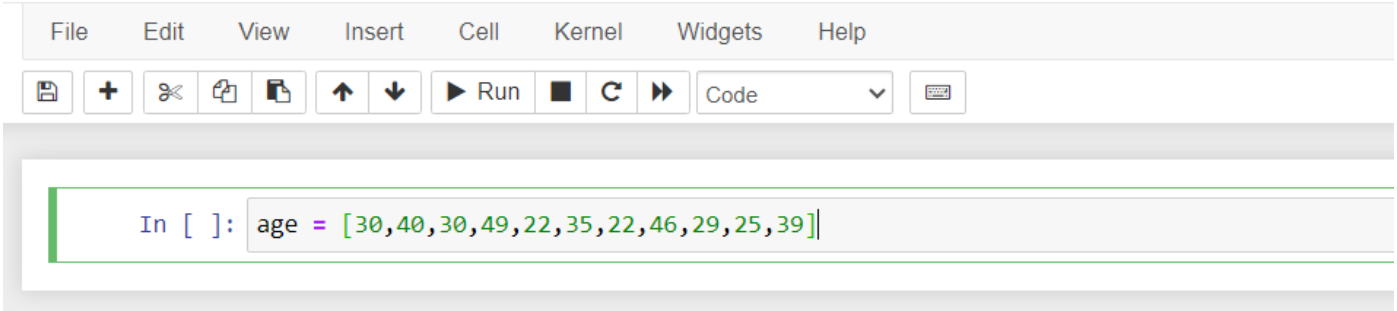
So what is the difference between Jupyter Notebook and a Python script file.

Well it is more interactive, it is like a combination of a Python script and the console. You can also write in two modes, the is code the other is like a word document. This helps to explain what your doing throughout the document.

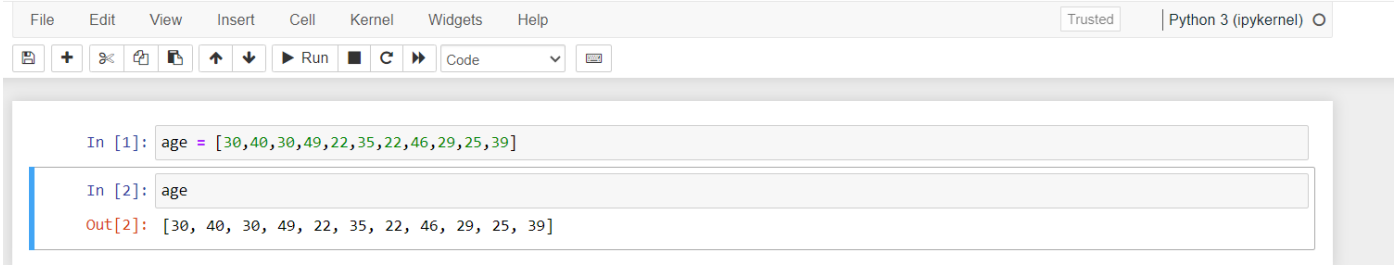
Let us see a few examples in action.

Let us go back to the one of our first lessons when we were using the country_data.csv file. We first learnt to input this data into lists. Let us do the same in a Jupyter Notebook. Enter the following list and click on "Run":

```
age = [30,40,30,49,22,35,22,46,29,25,39]
```



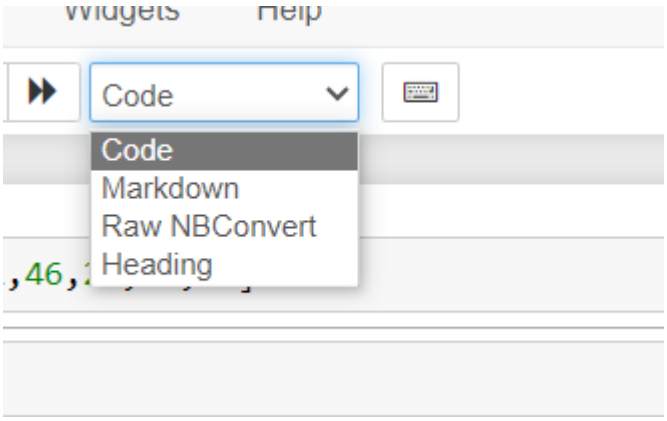
Now to print the data of "age" to the screen we can just input "age" and click on "Run":



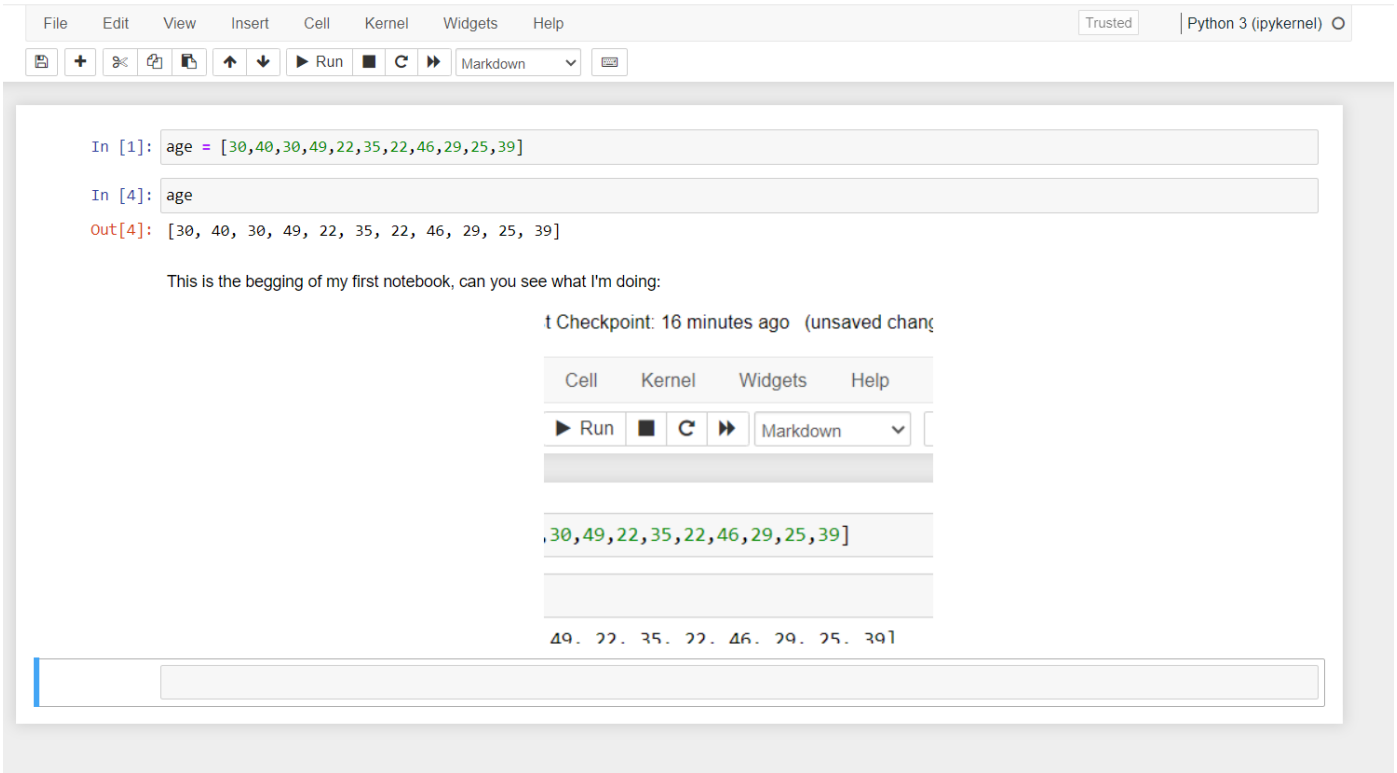
Similar to what we get in a Python console. However, now you can remove or add a cell, like you do in a script. Note you can also use the `print()` function to display it but it is not necessary.

Instead of press run you can press "Shift" + "F5" to run a cell.

Now you will notice there is drop-down that says code, if you click on you will see a few text options:



Now I added some text, not code, and then an image at the end:

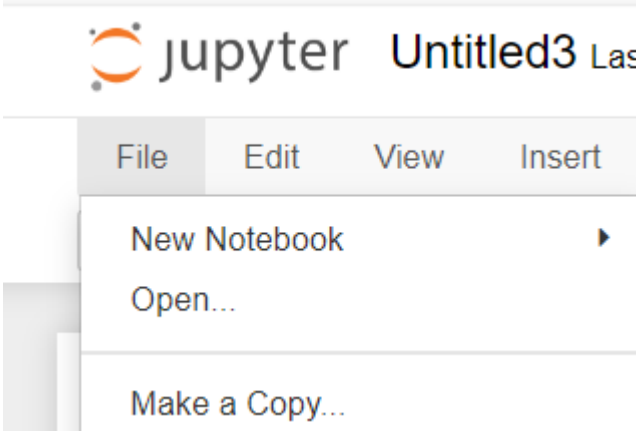


This is what makes Jupyter Notebook really usefull for teaching as you can easily add non-code items to you Jupyter Notebook, and essentially tell a story of your code.

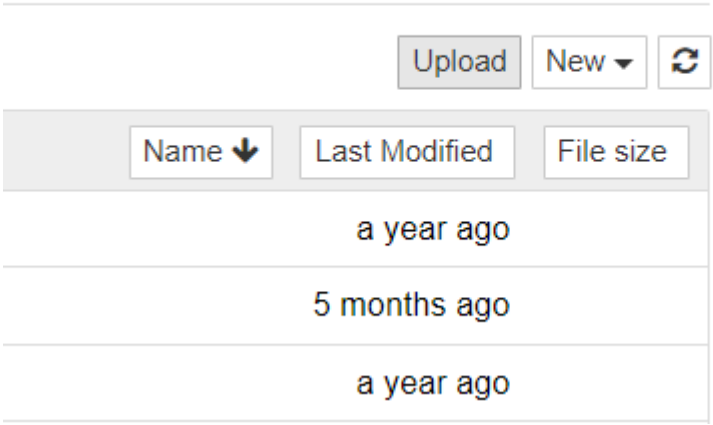
Pandas

Let us try read in some data.

Let us first upload the file, do that by going to File->Open:



Then click on the "Upload" button and the select the "country_data.csv":



Go back to your Jupyter Notebook and input the following code:

```
import pandas as pd

df = pd.read_csv("country_data.csv")

df
```

When you click on "Run", you get the following:

```
In [5]: import pandas as pd

        df = pd.read_csv("country_data.csv")

        df
```

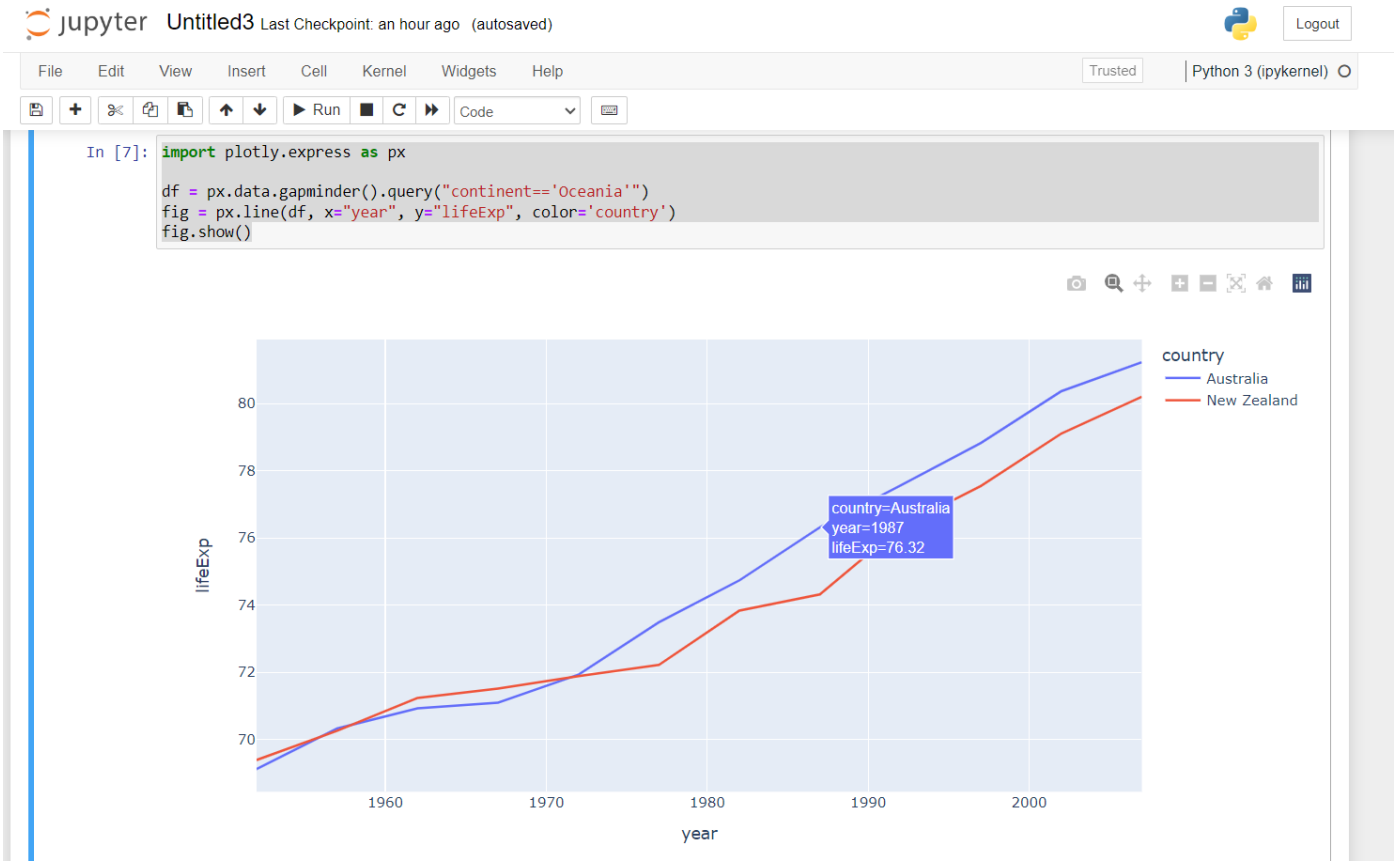
Out[5]:

	Unnamed: 0	Age	Gender	Country
0	0	39	M	South Africa
1	1	25	M	Botswana
2	2	29	F	South Africa
3	3	46	M	South Africa
4	4	22	F	Kenya
5	5	35	F	Mozambique
6	6	22	F	Lesotho
7	7	49	M	Kenya
8	8	30	M	Kenya
9	9	40	F	Egypt
10	10	30	M	Sudan

Which is quite a nice representation.

Visualization

Now we are going to look at plotting some graphs.



Numerical Calculations - stdev & R^2

Stdev

One of the outputs we showed was the standard deviation of the age list from the excel data set. Now we can do this in Python too quite easily with a built-in function but for the purposes of learning we are going to do a manual approach since we learnt all the tools needed now to do it. To work out the population standard deviation we need to do the following:

1. Calculate the mean (average) of each data set.
2. Subtract the deviance of each piece of data by subtracting the mean from each number.
3. Square each deviation.
4. Add all the squared deviations.
5. Divide the value obtained in step four by the number of items in the data set.
6. Calculate the square root of the value obtained in step five.

We have partially done this before. Let us complete the entire process:

Step 1. Given the age list we first need to find the mean of the list:

```
age=[39, 25, 29, 46, 22, 35, 22, 49, 30, 40, 30]

mean = sum(age)/len(age)

print(mean)
```

Step 2. Now we need to subtract the mean from value in the list and square the result:

We are first going to create an empty list that will store the squared differences called sq*diff*list and then we are going to create a for loop to run through each value in the age list, subtract the mean, square the result and store in the sq*diff*list variable:

```
# Create empty list
sq_diff_list = []

# Setup for loop
for i in range(len(age)):

    # Take average from each value in age list and square out
come. Note we square use double asterisks
    # sq_diff we are just using as a temporary variable to st
ore the result
    sq_diff = (age[i] - mean)**2

    # add the results to the `sq_diff_list` list
    sq_diff_list.append(sq_diff)

    # print the result
    print(sq_diff)
```

Now we need to work out the mean of the sq*diff*list and square the result:

```
mean_sq_diff = sum(sq_diff_list)/len(sq_diff_list)
print(mean_sq_diff**0.5)
```

This process was quite tedious and in practice you would not do this as there is a built-in Python function to do it as shown below:

```
import numpy
age=[39, 25, 29, 46, 22, 35, 22, 49, 30, 40, 30]
print(numpy.std(age))
```

R^2

Unfortunately, R-squared calculation is not implemented in numpy… so we will be doing it by hand using the following code:

```
import numpy as np
hours = [29, 9, 10, 38, 16, 26, 50, 10, 30, 33, 43, 2, 39, 1
5, 44, 29, 41, 15, 24, 50]
results = [65, 7, 8, 76, 23, 56, 100, 3, 74, 48, 73, 0, 62, 3
7, 74, 40, 90, 42, 58, 100]
x = np.asarray(hours)
y = np.asarray(results)
sum_x = np.sum(x)
sum_x_2 = np.sum(np.square(x))
sum_y = np.sum(y)
sum_y_2 = np.sum(np.square(y))
sum_xy = np.sum(x*y)
print(f"sum_x = {sum_x}")
print(f"sum_x_2 = {sum_x_2}")
print(f"sum_y = {sum_y}")
print(f"sum_y_2 = {sum_y_2}")
print(f"sum_xy = {sum_xy}")
n = len(x)
print(f"n = {n}")
top = n*sum_xy - sum_x*sum_y
print(f"top = {top}")
bot_a = np.sqrt(n*sum_x_2 - np.square(sum_x))
bot_b = np.sqrt(n*sum_y_2 - np.square(sum_y))
bot = bot_a*bot_b
print(f"bot = {bot}")
R_2 = np.square(top/bot)
print(f"R_2 = {R_2}")
```

Output:

```
sum_x = 553
sum_x_2 = 19345
```

```
sum_y = 1036
sum_y_2 = 72414
sum_xy = 36814
n = 20
top = 163372
bot = 174378.40331875964
R_2 = 0.8777480188408425
```

Practically, we should just use a Python library called sklearn that can do this for us but doing the manual approach allows us to learn more about Python.

```
import numpy as np
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt

# Data
hours = [29, 9, 10, 38, 16, 26, 50, 10, 30, 33, 43, 2, 39, 1
5, 44, 29, 41, 15, 24, 50]
results = [65, 7, 8, 76, 23, 56, 100, 3, 74, 48, 73, 0, 62, 3
7, 74, 40, 90, 42, 58, 100]

# Fit a linear regression model
model = np.polyfit(hours, results, 1)
predict = np.poly1d(model)

# Calculate R-squared
r2 = r2_score(results, predict(hours))
print("R-squared:", r2)

# Scatter plot
plt.scatter(hours, results, label='Actual data')

# Regression line plot
plt.plot(hours, predict(hours), color='red', label='Regressio
n line')

# Labels and title
plt.xlabel('Hours')
plt.ylabel('Results')
plt.title('Scatter Plot with Regression Line')

# Show legend
plt.legend()
```

```
# Display the plot
plt.show()
```

Output:

```
0.8777480188408424
```

Some issues with Jupyter Notebook

While Jupyter Notebooks are a popular and powerful tool for data analysis, scientific computing, and interactive programming in Python, there are some downsides and reasons why they might not be recommended for beginners, especially when just starting to learn Python:

In Jupyter Notebooks, code can be executed out of order, leading to confusion for beginners. The order in which cells are executed matters, and this can make it challenging to understand the flow of the program. Some other issues are variables and their values persist across cells, which can make it difficult to track the state of the program.

Automation

Python

Let us say we have 10 csv files with 2 columns each, x and y. We want to add up all the values in column y and store it in a list. Let us write a python script to do it first then in bash.

Download and extract the `csv_files.zip` folder and put it in your project directory.

```
import pandas as pd
import os

# List to store the sum of 'y' column values for each file
sum_list = []

# Directory containing CSV files
directory = './csv_files/'

# Loop through each CSV file in the directory
for filename in os.listdir(directory):
    print(f"Processing...Filename = {filename}")
    if filename.endswith('.csv'):
        file_path = os.path.join(directory, filename)


        # Read the CSV file using pandas
```

```
        df = pd.read_csv(file_path)

        # Calculate the sum of 'y' column and append to the l
ist
        total_sum = df['y'].sum()
        sum_list.append(total_sum)

# Print the list of sums
print("Sum of 'y' column values for each file:", sum_list)
```

Output:

```
Processing...Filename = file_1.csv
Processing...Filename = file_10.csv
Processing...Filename = file_2.csv
Processing...Filename = file_3.csv
Processing...Filename = file_4.csv
Processing...Filename = file_5.csv
Processing...Filename = file_6.csv
Processing...Filename = file_7.csv
Processing...Filename = file_8.csv
Processing...Filename = file_9.csv
Processing...Filename = read_csv.sh
Sum of 'y' column values for each file: [4945, 5029, 5400, 47
53, 5248, 4860, 5084, 49
79, 5581, 5217]
```

Bash

Doing the same with a bash script:

```
#!/bin/bash

# Directory containing CSV files
directory='csv_files/'
#directory='/c/Users/BBarsch.CSIR/css2024_day03/csv_files'

# List to store the sum of 'y' column values for each file
sum_list=()

# Loop through each CSV file in the directory
for filename in "$directory"/*.csv; do
    if [ -f "$filename" ]; then
```

```
        # Use awk to sum the 'y' column values
        echo $filename
        total_sum=$(awk -F',' 'NR > 1 {sum += $2} END {print
sum}' "$filename")

        # Append the sum to the list
        sum_list+=("$total_sum")
    fi
done

# Print the list of sums
echo "Sum of 'y' column values for each file: ${sum_list[@]}"
```

Output:

```
csv_files//file_1.csv
csv_files//file_10.csv
csv_files//file_2.csv
csv_files//file_3.csv
csv_files//file_4.csv
csv_files//file_5.csv
csv_files//file_6.csv
csv_files//file_7.csv
csv_files//file_8.csv
csv_files//file_9.csv
Sum of 'y' column values for each file: 4945 5029 5400 4753 5
248 4860 5084 4979 5581 5217
```

Bash with Python

Any time you get access to a web server your access will most likey be in some format of terminal - most likely a Bash terminal. While Bash is quite useful and you can do many things with it, it is quite limited to the functionality of Python and what Pandas can do. Many times you would manage your files with Bash but run a Python script to do the processing of it.

The script below combines the two codes we did before.

```
#!/bin/bash

# This is a comment so the terminal will ignore this line
# ... you can write anything here
```

```
echo "Process files"

winpty python read_csv_files.py
```

Output:

```
$ ./run_python.sh
Process files
Processing...Filename = file_1.csv
Processing...Filename = file_10.csv
Processing...Filename = file_2.csv
Processing...Filename = file_3.csv
Processing...Filename = file_4.csv
Processing...Filename = file_5.csv
Processing...Filename = file_6.csv
Processing...Filename = file_7.csv
Processing...Filename = file_8.csv
Processing...Filename = file_9.csv
Sum of 'y' column values for each file: [4945, 5029, 5400, 47
53, 5248, 4860, 5084, 4979, 5581, 5217]
```

This is a very simple calculation, imagine you had a lot more data and more complete problems to solve...if so then a supercomputer is the way to go! Remember all the code we have learnt up to now is sequential. Things are done one after the other. If we distribute that work load it can be done quicker.

Pandas Profiling:

Open Anaconda Prompt and input the following:

1. `conda create --name eda`
2. `conda activate eda`
3. `conda install conda-forge::ydata-profiling`
4. `jupyter notebook`

Code:

```
import numpy as np
import pandas as pd
from ydata_profiling import ProfileReport

df = pd.DataFrame(np.random.rand(100, 5), columns=["a", "b",
"c", "d", "e"])
```

```
profile = ProfileReport(df, title="Pandas Profiling Report")

profile
```

Resources

Github Desktop

https://survey.stackoverflow.co/2023/#technology

https://automatetheboringstuff.com/

https://www.offerzen.com/reports/software-developer-south-africa

https://streamlit.io/

https://pypi.org/project/pygwalker/

https://openrefine.org/

https://www.kaggle.com/

"Kaggle is an online community platform for data scientists and machine learning enthusiasts. Kaggle allows users to collaborate with other users, find and publish datasets, use GPU integrated notebooks, and compete with other data scientists to solve data science challenges."

https://www.datacamp.com/courses/intro-to-python-for-data-science

CHPC Winter School

https://events.chpc.ac.za/event/123/