CFG- ASSIGNMENT

Question 1

Advantages and disadvantages of Jupyter Notebook (.ipynb) and Python files (.py) for data analysts and scientists.

Jupyter Notebook (.ipynb):

A Jupyter Notebook provides an interactive computing environment, allowing users to create documents containing live code, visualizations, and narrative text. It runs on web browsers and supports various programming languages, including Python.

Advantages of Jupyter Notebook for data analysts/scientists

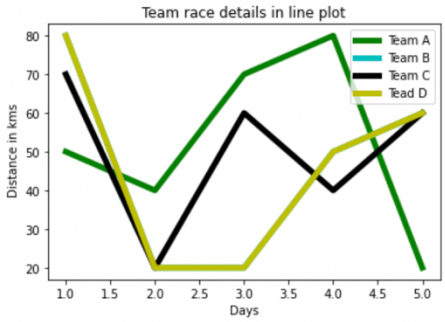
1. Interactivity: Users can execute code cells individually, facilitating iterative data exploration and step-by-step result understanding. See example below.

```
In [11]: test_df['Year']

Out[11]: 0       1916
         1       1920
         2       1925
         3       1927
         4       1929
                 ...
         1333    1999
         1334    1999
         1335    1999
         1336    1999
         1337    1999
         Name: Year, Length: 1338, dtype: int64

In [12]: test_df['Year'] == 1920

Out[12]: 0       False
         1        True
         2       False
         3       False
         4       False
                 ...
         1333    False
         1334    False
         1335    False
         1336    False
         1337    False
         Name: Year, Length: 1338, dtype: bool

In [13]: test_df[test_df['IMDB Score'] > 5]
```

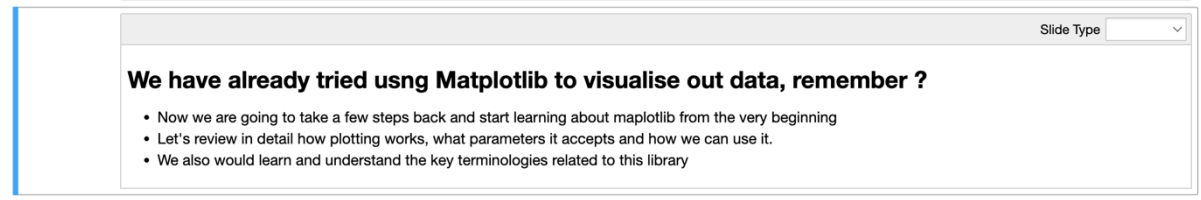
2. Data Visualisation: Visualizations can be displayed inline, making it easy to create and analyze charts and plots alongside the code.

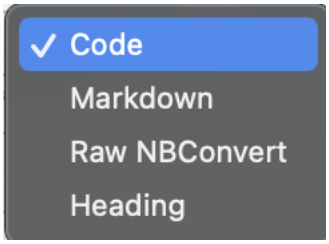
```
In [ ]:                                                    Slide Type
import matplotlib.pyplot as plt

x = [1,2,3,4,5]

y1 = [50,40,70,80,20]
y2 = [80,20,20,50,60]
y3 = [70,20,60,40,60]
y4 = [80,20,20,50,60]

plt.plot(x,y1,'g',label='Team A', linewidth=5)
plt.plot(x,y2,'c',label='Team B',linewidth=5)
plt.plot(x,y3,'k',label='Team C',linewidth=5)
plt.plot(x,y4,'y',label='Tead D',linewidth=5)

plt.title('Team race details in line plot')
plt.ylabel(' Distance in kms')
plt.xlabel('Days')

plt.legend()


# Various lines present in the graph have unique colors, and each of them denotes details of different teams.
# The line representing Team B is overwritten by the line representing Team D,
# since both vehicles have covered the same distance in their respective days.
```

```
Out[14]: <matplotlib.legend.Legend at 0x2161cdb8490>
```



3. Documentation and Communication: The inclusion of rich-text elements like Markdown enables effective documentation, explanations, and collaboration.





4. Code Reusability: Complex tasks can be broken down into reusable code cells, enhancing code readability and reusability.

5. Data Exploration and Prototyping: Real-time code experimentation enables rapid prototyping and interactive data exploration.

<u>Disadvantages of Jupyter Notebooks for data analysts/scientists</u>

version control challenges, as tracking changes with traditional systems like Git can be cumbersome due to the underlying JSON format.

```
# Reference: https://svds.com/jupyter-notebook-best-practices-for-data-science/
import os
from subprocess import check_call

def post_save(model, os_path, contents_manager):
    """post-save hook for converting notebooks to .py scripts"""
    if model['type'] != 'notebook':
        return # only do this for notebooks
    d, fname = os.path.split(os_path)
    check_call(['jupyter', 'nbconvert', '--to', 'script', fname], cwd=d)
    check_call(['jupyter', 'nbconvert', '--to', 'html', fname], cwd=d)

c.FileContentsManager.post_save_hook = post_save
```

1. Execution order dependencies may also lead to confusion and errors during sharing or rerunning the notebook.

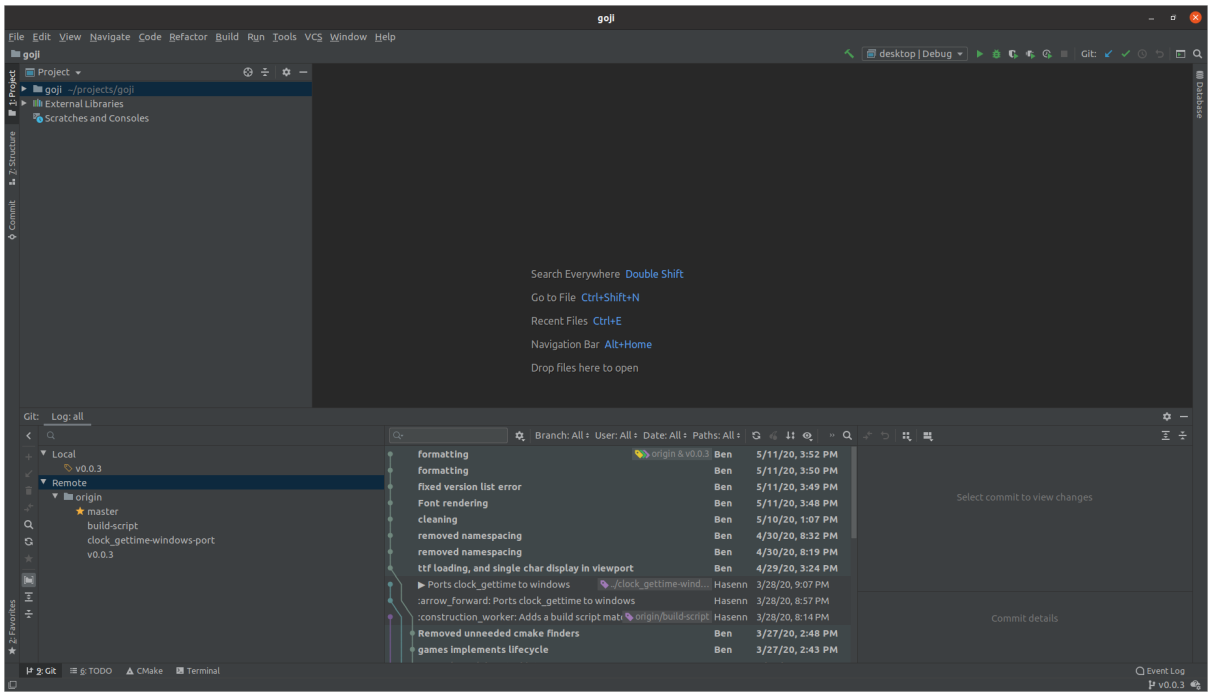
2. The only option available to run multiple cells is to use the “Run Cell” button at the top of the page, or you can use the keyboard shortcut “Shift+Enter”.

Python File (.py):

A Python file is a plain text file containing Python code, following a traditional script-based approach executed from top to bottom.

Advantages for data analysts/scientists

1. Version Control: Python files work seamlessly with version control systems like Git, enabling easy tracking of changes and collaboration.



2. Scripting and Automation: Ideal for creating scripts and automating data analysis tasks, especially when tasks require a specific order of execution.

For example, Suppose we have a folder with multiple CSV files, and we want to combine them into a single file. We can create a Python script to automate this task.

```
# Script to combine multiple CSV files into a single file

import os
import pandas as pd

def combine_csv_files(input_folder, output_file):
    all_data = pd.DataFrame()
    for file in os.listdir(input_folder):
        if file.endswith('.csv'):
            file_path = os.path.join(input_folder, file)
            data = pd.read_csv(file_path)
            all_data = all_data.append(data, ignore_index=True)

    all_data.to_csv(output_file, index=False)

# Example Usage
input_folder = 'data_files'
output_file = 'combined_data.csv'
combine_csv_files(input_folder, output_file)
```

3. Code Reusability and Packaging: Functions and modules in Python files can be reused in other scripts and projects, promoting code organization and maintainability.

4. Production Deployment: Python files are preferable for deploying data analysis pipelines or models to production environments due to their simplicity and integration ease.

Disadvantages of Python for data analysts/scientists

1. Lack of Interactivity: Unlike Jupyter Notebooks, Python files lack interactivity during code execution, making data exploration and visualization more manual.

2. Less Documentation Support: While comments are possible, Python files do not support rich-text elements like Markdown, making comprehensive documentation harder.

3. Limited Data Visualization Inline: Python files may require external libraries or additional code for inline data visualization, unlike Jupyter Notebooks.

Question 2

pandas series

A pandas series is a data structure consisting of key-value pairs, where the keys or labels serve as indices, and the corresponding values are stored at those indices. This structure resembles a Python dictionary but offers greater flexibility for data manipulation and editing. We use pandas.Series() to initialize a series object using Pandas.

```python
import pandas

##### INTIALIZATION #####

#STRING SERIES
fruits = pandas.Series(["apples", "oranges", "bananas"])

print("Fruit series:")
print(fruits)

#FLOAT SERIES
temperature = pandas.Series([32.6, 34.1, 28.0, 35.9])

print("\nTemperature series:")
print(temperature)

#INTEGER SERIES
factors_of_12 = pandas.Series([1,2,4,6,12])

print("\nFactors of 12 series:")
print(factors_of_12)

print("Type of this data structure is:", type(factors_of_12))
```

Pandas Dataframe

A pandas DataFrame is a two-dimensional data structure akin to a spreadsheet. Alternatively, it can be visualized as a collection of two or more series sharing common indices.

```python
import pandas as pd

##### INITIALIZATION #####

fruits_jack = ["apples", "oranges", "bananas"]
fruits_john = ["guavas", "kiwis", "strawberries"]
index = ["a", "b", "c"]
all_fruits = {"Jack's": fruits_jack, "John's": fruits_john}

fruits_default_index = pd.DataFrame(all_fruits)
print("Dataframe with default indices:\n", fruits_default_index, "\n")

new_fruits = pd.DataFrame(all_fruits, index = index)
print("Dataframe with given indices:\n", new_fruits, "\n")
```

Question 3

Data comes in various formats, but fundamentally, all data can be classified into two structures: rectangular and non-rectangular. Rectangular data takes the shape of a table, where each value corresponds to a specific row and column. This structure is commonly used in data frames. On the other hand, non-rectangular data does not have a neat arrangement in rows and columns; instead, it is often a combination of distinct data structures where members within the same structure share some similarities. Non-rectangular data is typically stored in lists or other nested structures, where each entry may have different attributes or may contain multiple values, making it non-tabular or non-rectangular in nature.

Pandas represents rectangular data because it is an efficient and flexible way to organize and manipulate structured data. Rectangular data, also known as tabular data, is organized in rows and columns, where each row corresponds to a record or observation, and each column represents a specific attribute or variable. It offers easy indexing, efficient computation, and seamless integration with various data sources and tools commonly used in data analysis. The tabular structure allows for flexible handling of different data types and enables efficient data manipulation and analysis.

Rectangular data

Spreadsheets

	A	B	C
1	name	gender	date
2	Dezik	Male	1951-07-22
3	Dezik	Male	1951-07-29
4	Tsygan	Male	1951-07-22
5	Lisa	Female	1951-07-29
6	Chizhik	Male	1951-08-15

CSV

```
name,gender,date
Dezik,Male,1951-07-22
Dezik,Male,1951-07-29
Tsygan,Male,1951-07-22
Lisa,Female,1951-07-29
Chizhik,Male,1951-08-15
```

Non-rectangular formats

JSON

```
{
    "name": "Darth Vader",
    "species": "Human",
    "homeworld": "Tatooine",
    "films": [
        "Revenge of the Sith",
        "Return of the Jedi",
        "The Empire Strikes Back",
        "A New Hope"
    ]
}
```

XML

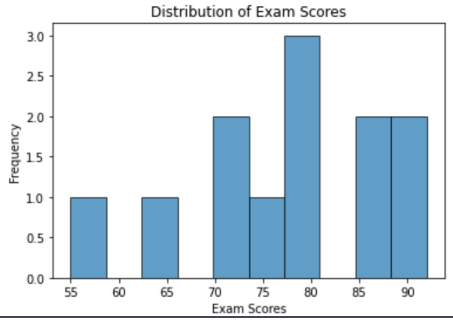
```
<note>
  <from>Teacher</from>
  <to>Student</to>
  <heading>Almost there</heading>
  <body>It's the final chapter!</body>
</note>
```

Question 4

1. Examples of how figures could be of use to the data scientist to identify patterns in the data/ highlight the important parts of a dataset:

 # A histogram is useful for analyzing the distribution of a numeric variable. Suppose you have a dataset containing exam scores of students. Using a histogram with matplotlib, you can visualize the distribution of scores to identify patterns like whether the scores are normally distributed, skewed, or bimodal.#

```
import matplotlib.pyplot as plt

# Sample data of exam scores
exam_scores = [75, 80, 65, 90, 85, 70, 55, 78, 92, 87, 80, 72]

# Create a histogram to visualize the distribution
plt.hist(exam_scores, bins=10, edgecolor='black', alpha=0.7)
plt.xlabel('Exam Scores')
plt.ylabel('Frequency')
plt.title('Distribution of Exam Scores')
plt.show()
```



 # Box Plot for Outlier Detection: A box plot is an effective way to identify outliers in a dataset. Suppose you have a dataset containing the salaries of employees in a company. Using a box plot with matplotlib, you can detect any unusually high or low salaries that might need further investigation

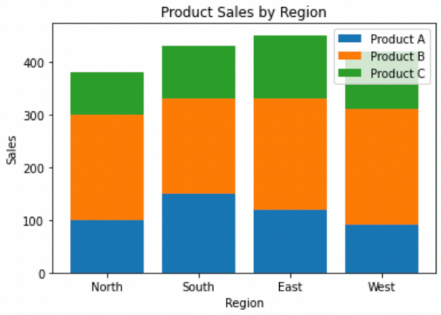
```
# Sample data of employee salaries
salaries = [30000, 40000, 45000, 50000, 80000, 90000, 110000, 150000, 200000]

# Create a box plot to visualize the distribution and identify outliers
plt.boxplot(salaries, vert=False)
plt.xlabel('Salary')
plt.title('Employee Salary Distribution')
plt.show()
```

You may also choose to plot the box plot for both male and female employees to check for Gender Pay gap.

2. How figures can be used to tell a story and create business presentations

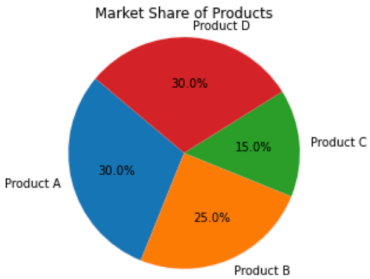
 # Stacked Bar Chart for Sales by Region: Suppose you have a dataset with sales data for different products across regions. Using a stacked bar chart with matplotlib, you can present the total sales for each product category and compare sales across different regions.

```
2]:
e data for product sales by region
 = ['North', 'South', 'East', 'West']
_sales = {
oduct A': [100, 150, 120, 90],
oduct B': [200, 180, 210, 220],
oduct C': [80, 100, 120, 110]


e a stacked bar chart to visualize product sales by region
(regions, product_sales['Product A'], label='Product A')
(regions, product_sales['Product B'], bottom=product_sales['Product A'], label='Product B')
(regions, product_sales['Product C'], bottom=[sum(x) for x in zip(product_sales['Product A'], product_sales['Product B'
bel('Region')
bel('Sales')
le('Product Sales by Region')
end()
w()
```



 # Pie Chart for Market Share Analysis: Another example of a pie chart, this time showcasing the market share of different products in percentages.

```
# Sample data for market share of products
products = ['Product A', 'Product B', 'Product C', 'Product D']
market_share = [30, 25, 15, 30]

# Create a pie chart to visualize the market share of products
plt.pie(market_share, labels=products, autopct='%1.1f%%', startangle=140)
plt.axis('equal')
plt.title('Market Share of Products')
plt.show()
```



Question 5

a. 15 columns and 891 rows

```
]: titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   survived     891 non-null    int64
 1   pclass       891 non-null    int64
 2   sex          891 non-null    object
 3   age          714 non-null    float64
 4   sibsp        891 non-null    int64
 5   parch        891 non-null    int64
 6   fare         891 non-null    float64
 7   embarked     889 non-null    object
 8   class        891 non-null    object
 9   who          891 non-null    object
 10  adult_male   891 non-null    bool
 11  deck         203 non-null    object
 12  embark_town  889 non-null    object
 13  alive        891 non-null    object
 14  alone        891 non-null    bool
dtypes: bool(2), float64(2), int64(4), object(7)
memory usage: 92.4+ KB
```

b.

```
In [7]: titanic.describe(include= 'all')
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
count	891.000000	891.000000	891	714.000000	891.000000	891.000000	891.000000	889	891	891	891	203	889	891	891
unique	NaN	NaN	2	NaN	NaN	NaN	NaN	3	3	3	2	7	3	2	2
top	NaN	NaN	male	NaN	NaN	NaN	NaN	S	Third	man	True	C	Southampton	no	True
freq	NaN	NaN	577	NaN	NaN	NaN	NaN	644	491	537	537	59	644	549	537
mean	0.383838	2.308642	NaN	29.699118	0.523008	0.381594	32.204208	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	0.486592	0.836071	NaN	14.526497	1.102743	0.806057	49.693429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	0.000000	1.000000	NaN	0.420000	0.000000	0.000000	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	0.000000	2.000000	NaN	20.125000	0.000000	0.000000	7.910400	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	0.000000	3.000000	NaN	28.000000	0.000000	0.000000	14.454200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	1.000000	3.000000	NaN	38.000000	1.000000	0.000000	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	1.000000	3.000000	NaN	80.000000	8.000000	6.000000	512.329200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

c. 177 missing values in age, 688 missing values in deck and 2 missing values in embark_town

```
: missing_values = titanic.isnull().sum()
  missing_values
```

```
: survived         0
  pclass           0
  sex              0
  age            177
  sibsp            0
  parch            0
  fare             0
  embarked         2
  class            0
  who              0
  adult_male       0
  deck           688
  embark_town      2
  alive            0
  alone            0
  dtype: int64
```

```
# Drop rows where the "age" column is empty
titanic_cleaned = titanic.dropna(subset=['age'])
titanic_cleaned
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
885	0	3	female	39.0	0	5	29.1250	Q	Third	woman	False	NaN	Queenstown	no	False
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

714 rows × 15 columns

d.

```
In [13]: titanic['age'].max()
Out[13]: 80.0
```

```
In [17]: #Since the highest age is 80, my bin will cut off at 80

         # Define the age ranges and corresponding labels
         age_ranges = [0, 11, 21, 31, 41, 51, 61, 71, 81]
         age_labels = ['0-10', '11-20', '21-30', '31-40', '41-50', '51-60', '61-70', '71-80']

         # Create a new column "Age Range" based on the age_bins
         titanic['Age Range'] = pd.cut(titanic['age'], bins=age_ranges, labels=age_labels, right=False)

         # Display the DataFrame with the "Age Range" column
         print(titanic)
```

```
     survived  pclass     sex   age  sibsp  parch     fare embarked   class  \
0           0       3    male  22.0      1      0   7.2500        S   Third
1           1       1  female  38.0      1      0  71.2833        C   First
2           1       3  female  26.0      0      0   7.9250        S   Third
3           1       1  female  35.0      1      0  53.1000        S   First
4           0       3    male  35.0      0      0   8.0500        S   Third
..        ...     ...     ...   ...    ...    ...      ...      ...     ...
885         0       3  female  39.0      0      5  29.1250        Q   Third
886         0       2    male  27.0      0      0  13.0000        S  Second
887         1       1  female  19.0      0      0  30.0000        S   First
889         1       1    male  26.0      0      0  30.0000        C   First
890         0       3    male  32.0      0      0   7.7500        Q   Third

        who  adult_male deck  embark_town alive  alone Age Range
0      male        True  NaN  Southampton    no  False     21-30
1    female       False    C    Cherbourg   yes  False     31-40
2    female       False  NaN  Southampton   yes   True     21-30
3    female       False    C  Southampton   yes  False     31-40
4      male        True  NaN  Southampton    no   True     31-40
..      ...         ...  ...          ...   ...    ...       ...
885  female       False  NaN   Queenstown    no  False     31-40
886    male        True  NaN  Southampton    no   True     21-30
887  female       False    B  Southampton   yes   True     11-20
889    male        True    C    Cherbourg   yes   True     21-30
890    male        True  NaN   Queenstown    no   True     31-40

[714 rows x 16 columns]
```

I excluded child from the list

In [18]:

```
# Replace 'man' with 'male' and 'woman' with 'female' in the 'who' column
titanic['who'] = titanic['who'].replace({'man': 'male', 'woman': 'female'})



titanic_filtered = titanic[titanic['who'].isin(['male', 'female'])]



# Group by "Age Range" and count the number of passengers
grouped_titanic = titanic_filtered.groupby(['Age Range','who'])['who'].count().reset_index(name='Passenger Count')

# Display the grouped DataFrame
print(grouped_titanic)
```

```
   Age Range     who  Passenger Count
0       0-10  female                0
1       0-10    male                0
2      11-20  female               34
3      11-20    male               63
4      21-30  female               82
5      21-30    male              149
6      31-40  female               54
7      31-40    male              101
8      41-50  female               31
9      41-50    male               53
10     51-60  female               14
11     51-60    male               28
12     61-70  female                3
13     61-70    male               15
14     71-80  female                0
15     71-80    male                4
```