Data Technician

Name: Athiramol Padinjarekudiyil Sukumaran

Course Date: 08/09/25-11/09/25

Table of contents

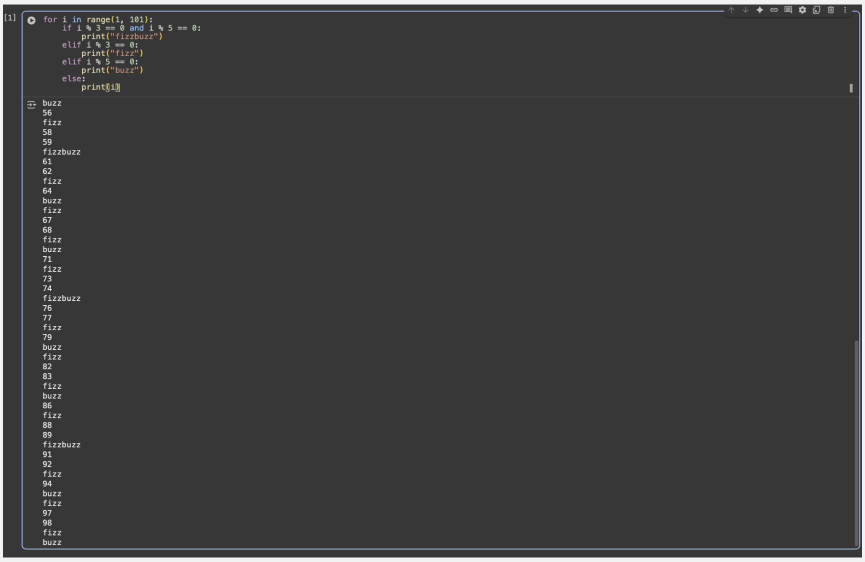
Day 2: Task 1

It is a common software development interview question to create the below with a certain programming language. Create the below using Python syntax, test it and past the completed syntax and output below.

FizzBuzz:

Go through the integers from 1 to 100.
If a number is divisible by 3, print "fizz."
If a number is divisible by 5, print "buzz."
If a number is both divisible by 3 and by 5, print "fizzbuzz."
Otherwise, print just the number.

Paste your completed work to the right



```
[1]
for i in range(1, 101):
    if i % 3 == 0 and i % 5 == 0:
        print("fizzbuzz")
    elif i % 3 == 0:
        print("fizz")
    elif i % 5 == 0:
        print("buzz")
    else:
        print(i)
```

```
buzz
56
fizz
58
59
fizzbuzz
61
62
fizz
64
buzz
fizz
67
68
fizz
buzz
71
fizz
73
74
fizzbuzz
76
77
fizz
79
buzz
fizz
82
83
fizz
buzz
86
fizz
88
89
fizzbuzz
91
92
fizz
94
buzz
fizz
97
98
fizz
buzz
```

Day 3: Task 1

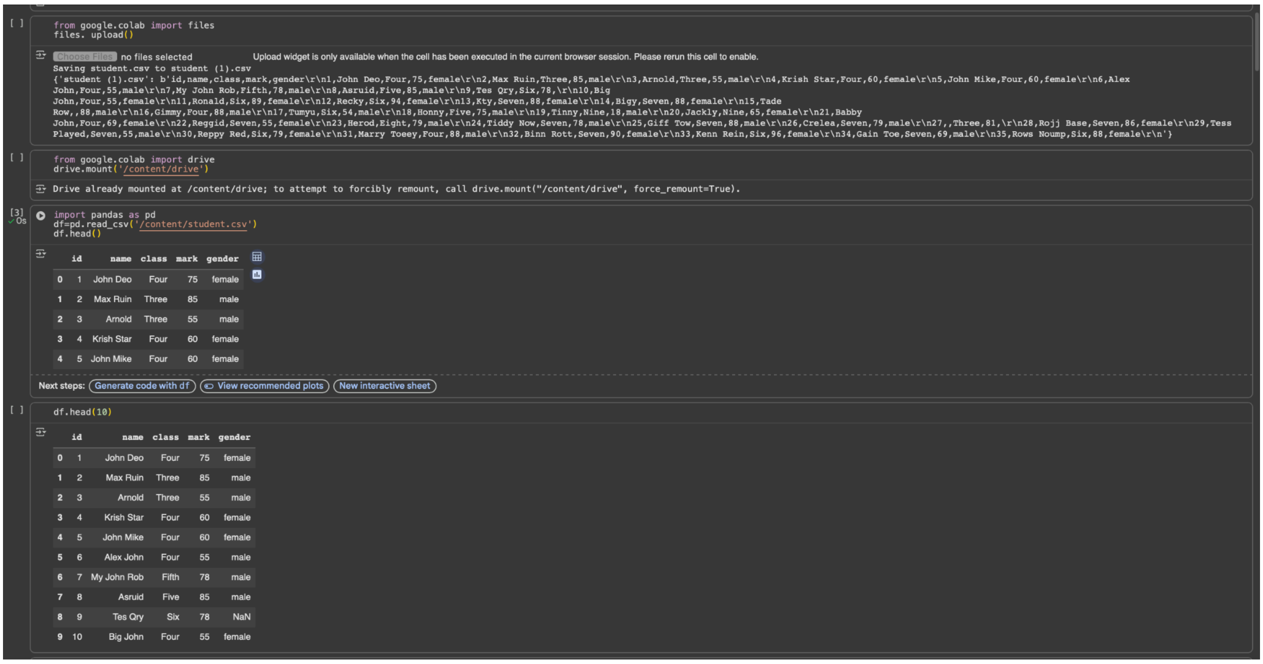
Using the ‘student.csv’ which can be downloaded here, complete the below exercises as a group and paste your input and output. Although this is a group activity, everyone should have the below answered so it supports your portfolio:

Exercise 1: Loading and Exploring the Data

1. Question: "Write the code to read a CSV file into a Pandas DataFrame."
2. Question: "Write the code to display the first 5 rows of the DataFrame."

3. Question: "Write the code to get the information about the DataFrame."
4. Question: "Write the code to get summary statistics for the DataFrame."

1. df= pd.read_csv(‘filename.csv’)
2. Df.head()
3. Df.info()
4. Df.describe()



```
[ ]  from google.colab import files
     files. upload()
```

```
no files selected          Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving student.csv to student (1).csv
{'student (1).csv': b'id,name,class,mark,gender\r\n1,John Deo,Four,75,female\r\n2,Max Ruin,Three,85,male\r\n3,Arnold,Three,55,male\r\n4,Krish Star,Four,60,female\r\n5,John Mike,Four,60,female\r\n6,Alex
John,Four,55,male\r\n7,My John Rob,Fifth,78,male\r\n8,Asruid,Five,85,male\r\n9,Tes Qry,Six,78,\r\n10,Big
John,Four,55,female\r\n11,Ronald,Six,89,female\r\n12,Recky,Six,94,female\r\n13,Kty,Seven,88,female\r\n14,Bigy,Seven,88,female\r\n15,Tade
Row,,88,male\r\n16,Gimmy,Four,88,male\r\n17,Tumyu,Six,54,male\r\n18,Honny,Five,75,male\r\n19,Tinny,Nine,18,male\r\n20,Jackly,Nine,65,female\r\n21,Babby
John,Four,69,female\r\n22,Reggid,Seven,55,female\r\n23,Herod,Eight,79,male\r\n24,Tiddy Now,Seven,78,male\r\n25,Giff Tow,Seven,88,male\r\n26,Crelea,Seven,79,male\r\n27,,Three,81,\r\n28,Rojj Base,Seven,86,female\r\n29,Tess
Played,Seven,55,male\r\n30,Reppy Red,Six,79,female\r\n31,Marry Toeey,Four,88,male\r\n32,Binn Rott,Seven,90,female\r\n33,Kenn Rein,Six,96,female\r\n34,Gain Toe,Seven,69,male\r\n35,Rows Noump,Six,88,female\r\n'}
```

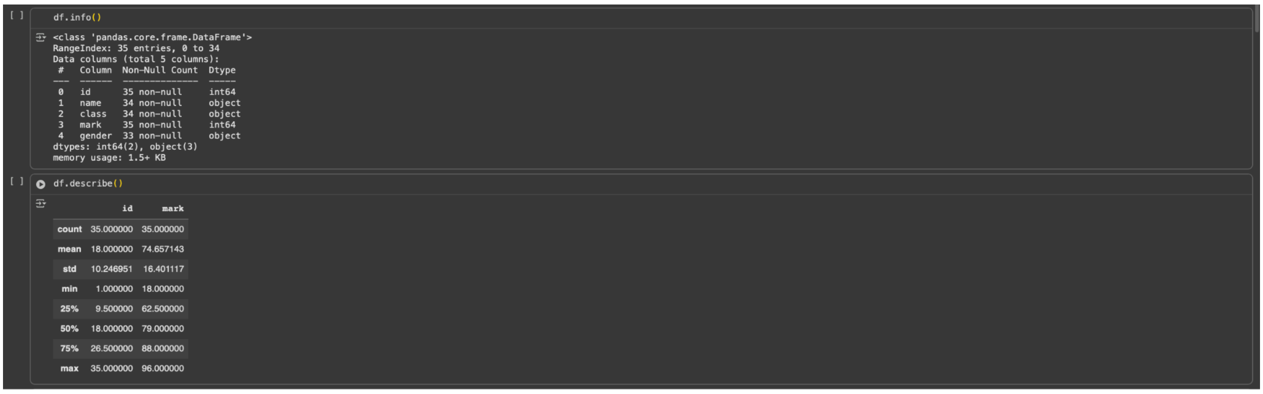
```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[3]  import pandas as pd
0s   df=pd.read_csv('/content/student.csv')
     df.head()
```

	id	name	class	mark	gender
0	1	John Deo	Four	75	female
1	2	Max Ruin	Three	85	male
2	3	Arnold	Three	55	male
3	4	Krish Star	Four	60	female
4	5	John Mike	Four	60	female

```
[ ]  df.head(10)
```

	id	name	class	mark	gender
0	1	John Deo	Four	75	female
1	2	Max Ruin	Three	85	male
2	3	Arnold	Three	55	male
3	4	Krish Star	Four	60	female
4	5	John Mike	Four	60	female
5	6	Alex John	Four	55	male
6	7	My John Rob	Fifth	78	male
7	8	Asruid	Five	85	male
8	9	Tes Qry	Six	78	NaN
9	10	Big John	Four	55	female



```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35 entries, 0 to 34
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      35 non-null     int64
 1   name    34 non-null     object
 2   class   34 non-null     object
 3   mark    35 non-null     int64
 4   gender  33 non-null     object
dtypes: int64(2), object(3)
memory usage: 1.5+ KB
```

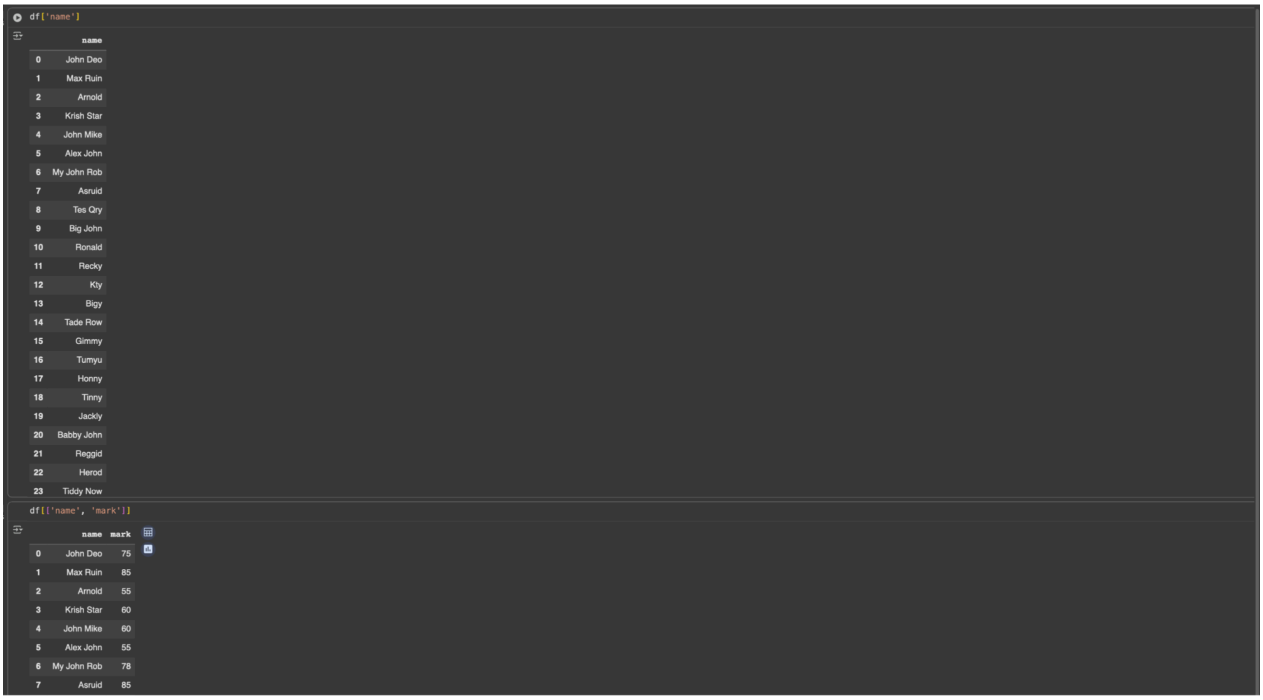
```
[ ]  df.describe()
```

	id	mark
count	35.000000	35.000000
mean	18.000000	74.657143
std	10.246951	16.401117
min	1.000000	18.000000
25%	9.500000	62.500000
50%	18.000000	79.000000
75%	26.500000	88.000000
max	35.000000	96.000000

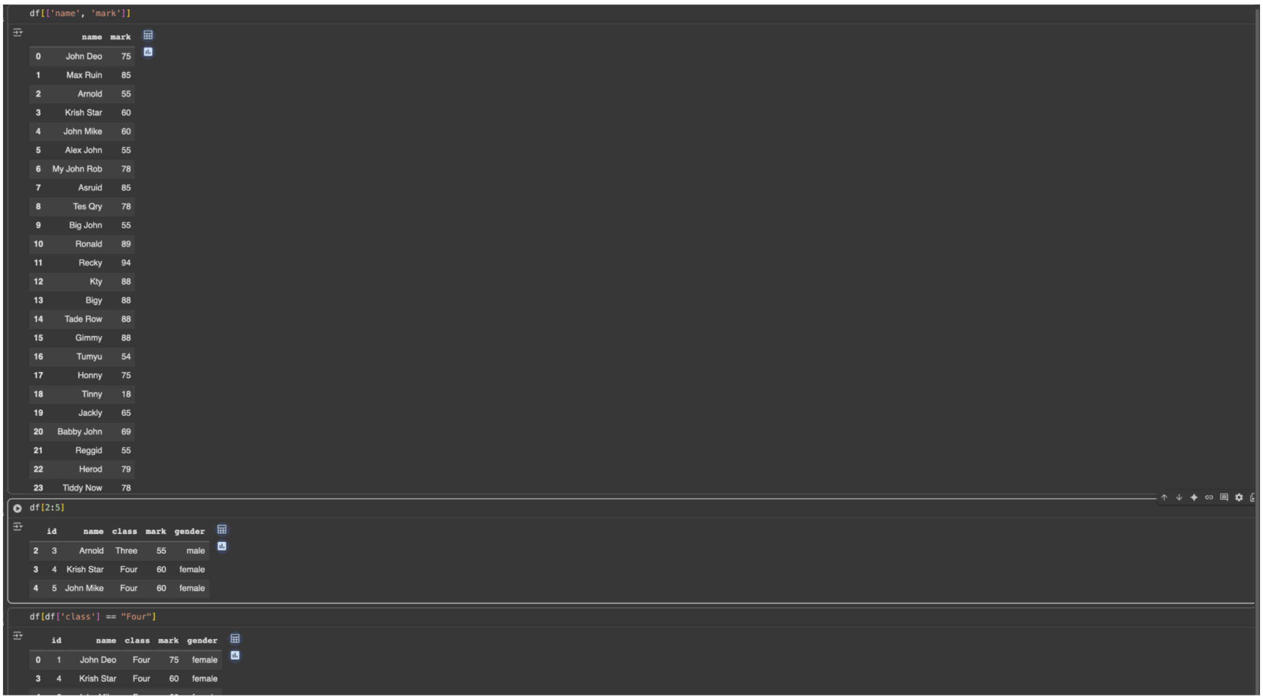
Exercise 2: Indexing and Slicing

1. Question: "Write the code to select the 'name' column."
2. Question: "Write the code to select the 'name' and 'mark' columns."
3. Question: "Write the code to select the first 3 rows."
4. Question: "Write the code to select all rows where the 'class' is 'Four'."

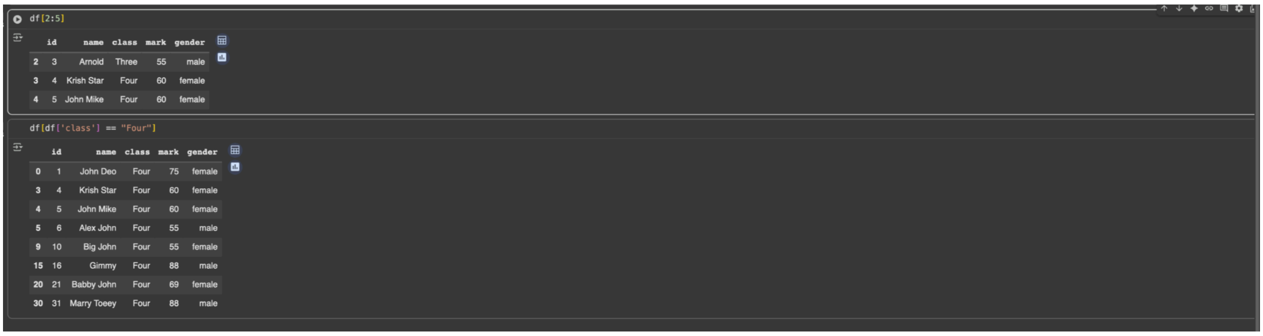
1. Df[‘name’]
2. Df[[‘name’, ‘mark’]
3. Df[: 3] or df.head(3)
4. Df[df[‘class’] == “Four”]



```
df['name']

    name
0   John Deo
1   Max Ruin
2   Arnold
3   Krish Star
4   John Mike
5   Alex John
6   My John Rob
7   Asruid
8   Tes Qry
9   Big John
10  Ronald
11  Recky
12  Kty
13  Bigy
14  Tade Row
15  Gimmy
16  Tumyu
17  Honny
18  Tinny
19  Jackly
20  Babby John
21  Reggid
22  Herod
23  Tiddy Now

df[['name', 'mark']]

           name  mark
0      John Deo    75
1      Max Ruin    85
2        Arnold    55
3    Krish Star    60
4     John Mike    60
5     Alex John    55
6   My John Rob    78
7        Asruid    85
```



```
df[['name', 'mark']]

           name  mark
0      John Deo    75
1      Max Ruin    85
2        Arnold    55
3    Krish Star    60
4     John Mike    60
5     Alex John    55
6   My John Rob    78
7        Asruid    85
8       Tes Qry    78
9      Big John    55
10       Ronald    89
11        Recky    94
12          Kty    88
13         Bigy    88
14     Tade Row    88
15        Gimmy    88
16        Tumyu    54
17        Honny    75
18        Tinny    18
19       Jackly    65
20   Babby John    69
21       Reggid    55
22        Herod    79
23    Tiddy Now    78

df[2:5]

   id        name  class  mark  gender
2   3      Arnold  Three    55    male
3   4  Krish Star   Four    60  female
4   5   John Mike   Four    60  female

df[df['class'] == "Four"]

   id        name  class  mark  gender
0   1    John Deo   Four    75  female
3   4  Krish Star   Four    60  female
```



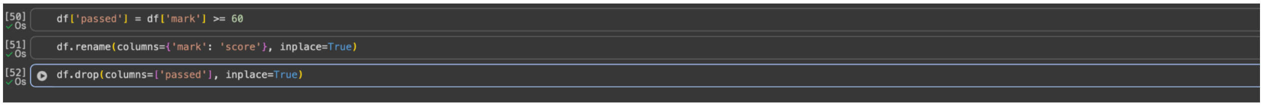
```
df[2:5]

   id        name  class  mark  gender
2   3      Arnold  Three    55    male
3   4  Krish Star   Four    60  female
4   5   John Mike   Four    60  female

df[df['class'] == "Four"]

    id         name  class  mark  gender
0    1     John Deo   Four    75  female
3    4   Krish Star   Four    60  female
4    5    John Mike   Four    60  female
5    6    Alex John   Four    55    male
9   10     Big John   Four    55  female
15  16        Gimmy   Four    88    male
20  21   Babby John   Four    69  female
30  31  Marry Toeey   Four    88    male
```

Exercise 3: Data Manipulation

1. Question: "Write the code to add a new column 'passed' that indicates whether the student passed (mark >= 60)."

2. Question: "Write the code to rename the 'mark' column to 'score'."
3. Question: "Write the code to drop the 'passed' column."

1. Df[‘passed’] = df[‘mark’] >= 60
2. Df.rename(columns=(‘mark’:’score’), inplace = True)
3. Df.drop(columns=[’passed’], inplace = True)

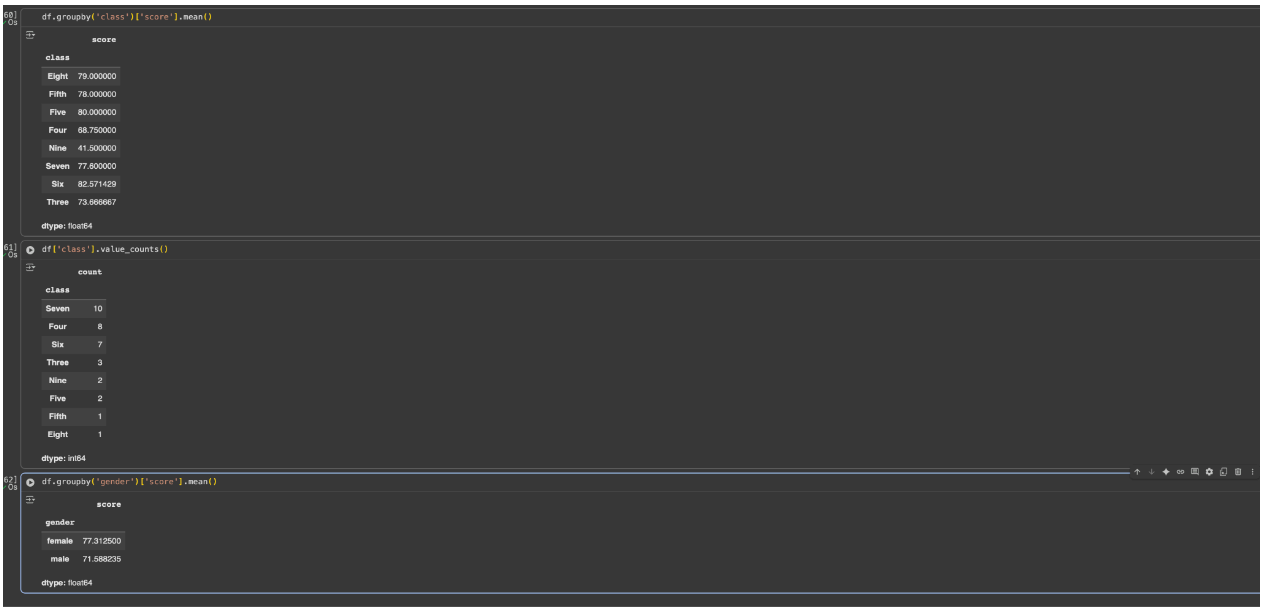


```
[50] df['passed'] = df['mark'] >= 60
[51] df.rename(columns={'mark': 'score'}, inplace=True)
[52] df.drop(columns=['passed'], inplace=True)
```

Exercise 4: Aggregation and Grouping

1. Question: "Write the code to group the DataFrame by the 'class' column and calculate the mean 'mark' for each group."
2. Question: "Write the code to count the number of students in each class."
3. Question: "Write the code to calculate the average mark for each gender."

1. Df.groupby(‘class’)[‘score’].mean()
2. Df[‘class’].value_counts()
3. Df.groupby(‘gender’)[‘score’].mean()

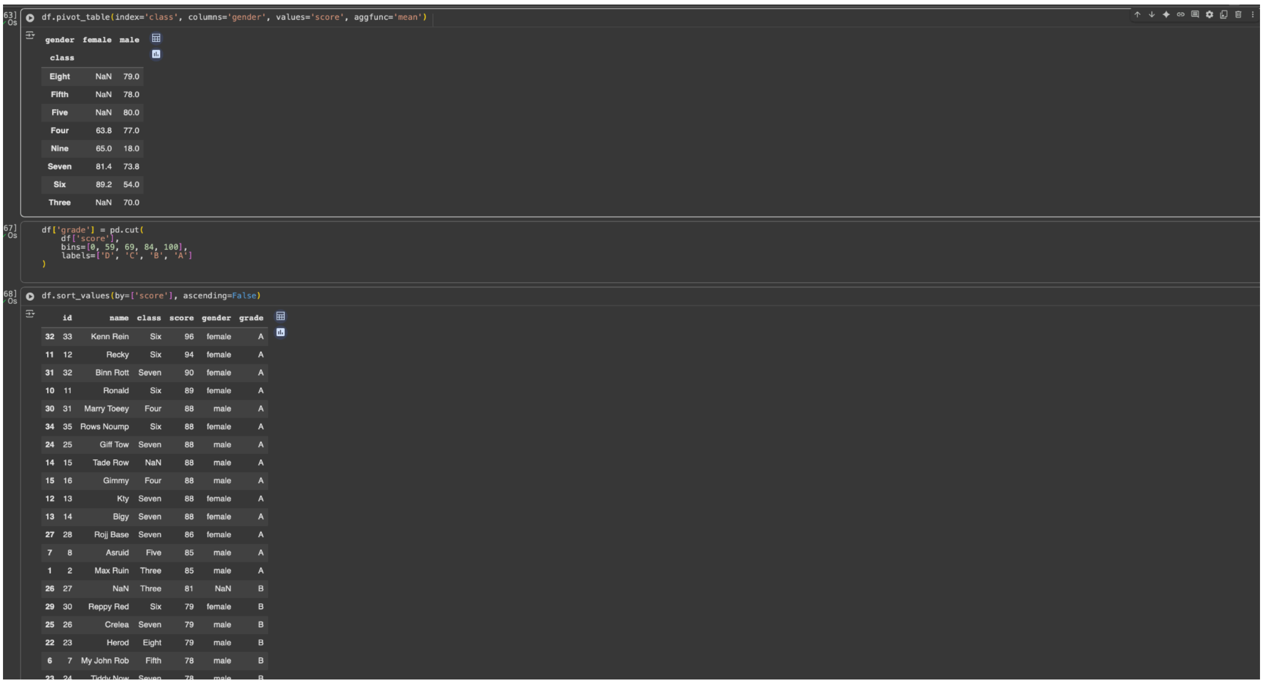


```
60] df.groupby('class')['score'].mean()

            score
   class
   Eight    79.000000
   Fifth    78.000000
   Five     80.000000
   Four     68.750000
   Nine     41.500000
   Seven    77.600000
   Six      82.571429
   Three    73.666667

   dtype: float64

61] df['class'].value_counts()

          count
   class
   Seven     10
   Four       8
   Six        7
   Three      3
   Nine       2
   Five       2
   Fifth      1
   Eight      1

   dtype: int64

62] df.groupby('gender')['score'].mean()

             score
   gender
   female    77.312500
   male      71.588235

   dtype: float64
```

Exercise 5: Advanced Operations

1. Question: "Write the code to create a pivot table with 'class' as rows, 'gender' as columns, and 'mark' as values."
2. Question: "Write the code to create a new column 'grade' where marks >= 85 are 'A', 70-84 are 'B', 60-69 are 'C', and below 60 are 'D'."
3. Question: "Write the code to sort the DataFrame by 'mark' in descending order."

1. Df.pivot_table(index=’class’, columns=’gender’, values=’score’, aggfunc=’mean’)
2. Df[‘grade’]=pd.cut(df[‘score’], bins=[0, 59, 69, 84, 100], labels=['D’, ‘C’, ‘B’, ‘A’])
3. Df.sort_values(by=['score’], ascending = False)



```
df.pivot_table(index='class', columns='gender', values='score', aggfunc='mean')
```

```
gender  female  male
 class
 Eight     NaN  79.0
 Fifth     NaN  78.0
  Five     NaN  80.0
  Four    63.8  77.0
  Nine    65.0  18.0
 Seven    81.4  73.8
   Six    89.2  54.0
 Three     NaN  70.0
```

```
df['grade'] = pd.cut(
    df['score'],
    bins=[0, 59, 69, 84, 100],
    labels=['D', 'C', 'B', 'A']
)
```

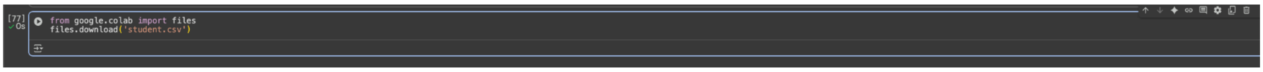
```
df.sort_values(by=['score'], ascending=False)
```

```
    id         name   class  score  gender  grade
32  33    Kenn Rein     Six     96  female      A
11  12        Recky     Six     94  female      A
31  32    Binn Rott   Seven     90  female      A
10  11       Ronald     Six     89  female      A
30  31  Marry Toeey    Four     88    male      A
34  35   Rows Noump     Six     88  female      A
24  25     Giff Tow   Seven     88    male      A
14  15     Tade Row     NaN     88    male      A
15  16        Gimmy    Four     88    male      A
12  13          Kty   Seven     88  female      A
13  14         Bigy   Seven     88  female      A
27  28    Rojj Base   Seven     86  female      A
 7   8       Asruid    Five     85    male      A
 1   2     Max Ruin   Three     85    male      A
26  27          NaN   Three     81     NaN      B
29  30    Reppy Red     Six     79  female      B
25  26       Crelea   Seven     79    male      B
22  23        Herod   Eight     79    male      B
 6   7  My John Rob   Fifth     78    male      B
23  24    Tiddy Now   Seven     78    male      B
```

Exercise 6: Exporting Data

1. Question: "Write the code to save the DataFrame with the new 'grade' column to a new CSV file."

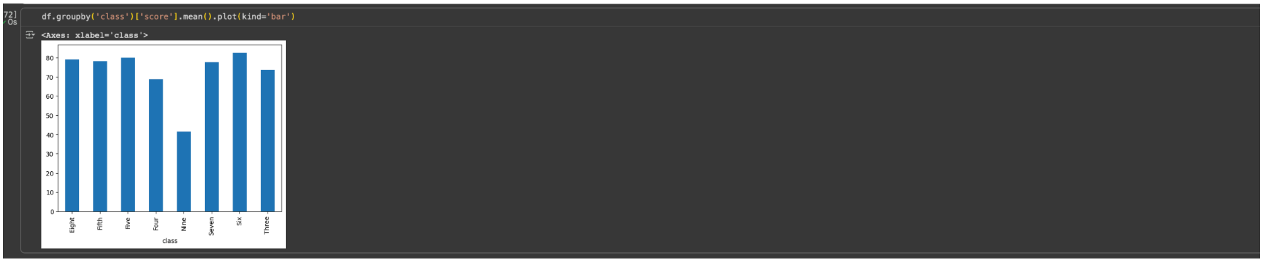
From google.colab import files
Files.download(‘new_filename.csv’)



```
from google.colab import files
files.download('student.csv')
```

Exercise 7: If finished early try visualising the results

Df.groupby(‘class’)[‘score’].mean().plot(kind=’bar’)



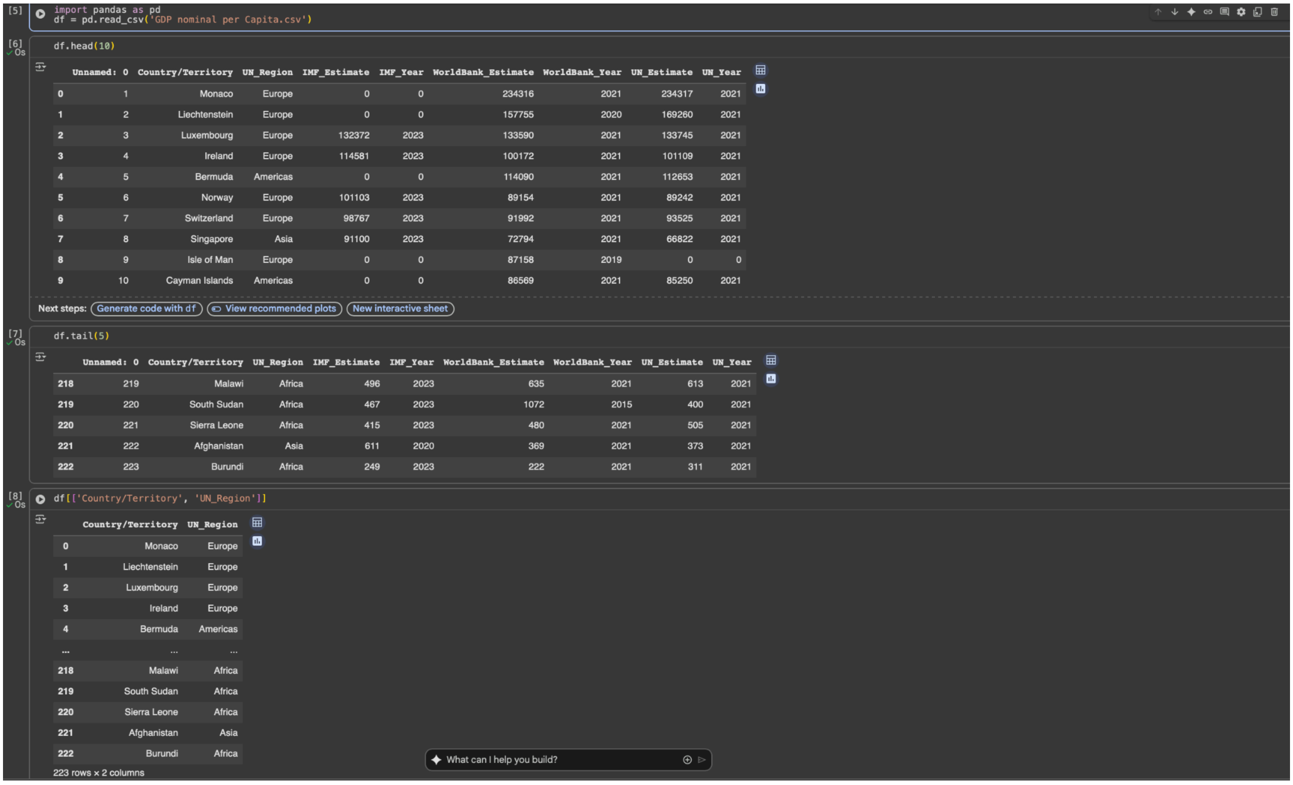
```
df.groupby('class')['score'].mean().plot(kind='bar')
```

```
<Axes: xlabel='class'>
```

Day 4: Task 1

Using the ‘GDP (nominal) per Capita.csv’ which can be downloaded here, complete the below exercises and paste your input and output. Work individually, but we will work and support each other in the room.

- Read and save the ‘GDP (nominal) per Capita’ data to a data frame called “df” in Jupyter notebook
- Print the first 10 rows
- Print the last 5 rows
- Print ‘Country/Territory’ and ‘UN_Region’ columns



```
[5]
import pandas as pd
df = pd.read_csv('GDP nominal per Capita.csv')
```

```
[6]
df.head(10)
```

	Unnamed: 0	Country/Territory	UN_Region	IMF_Estimate	IMF_Year	WorldBank_Estimate	WorldBank_Year	UN_Estimate	UN_Year
0	1	Monaco	Europe	0	0	234316	2021	234317	2021
1	2	Liechtenstein	Europe	0	0	157755	2020	169260	2021
2	3	Luxembourg	Europe	132372	2023	133590	2021	133745	2021
3	4	Ireland	Europe	114581	2023	100172	2021	101109	2021
4	5	Bermuda	Americas	0	0	114090	2021	112653	2021
5	6	Norway	Europe	101103	2023	89154	2021	89242	2021
6	7	Switzerland	Europe	98767	2023	91992	2021	93525	2021
7	8	Singapore	Asia	91100	2023	72794	2021	66822	2021
8	9	Isle of Man	Europe	0	0	87158	2019	0	0
9	10	Cayman Islands	Americas	0	0	86569	2021	85250	2021

```
[7]
df.tail(5)
```

	Unnamed: 0	Country/Territory	UN_Region	IMF_Estimate	IMF_Year	WorldBank_Estimate	WorldBank_Year	UN_Estimate	UN_Year
218	219	Malawi	Africa	496	2023	635	2021	613	2021
219	220	South Sudan	Africa	467	2023	1072	2015	400	2021
220	221	Sierra Leone	Africa	415	2023	480	2021	505	2021
221	222	Afghanistan	Asia	611	2020	369	2021	373	2021
222	223	Burundi	Africa	249	2023	222	2021	311	2021

```
[8]
df[['Country/Territory', 'UN_Region']]
```

	Country/Territory	UN_Region
0	Monaco	Europe
1	Liechtenstein	Europe
2	Luxembourg	Europe
3	Ireland	Europe
4	Bermuda	Americas
...	...	...
218	Malawi	Africa
219	South Sudan	Africa
220	Sierra Leone	Africa
221	Afghanistan	Asia
222	Burundi	Africa

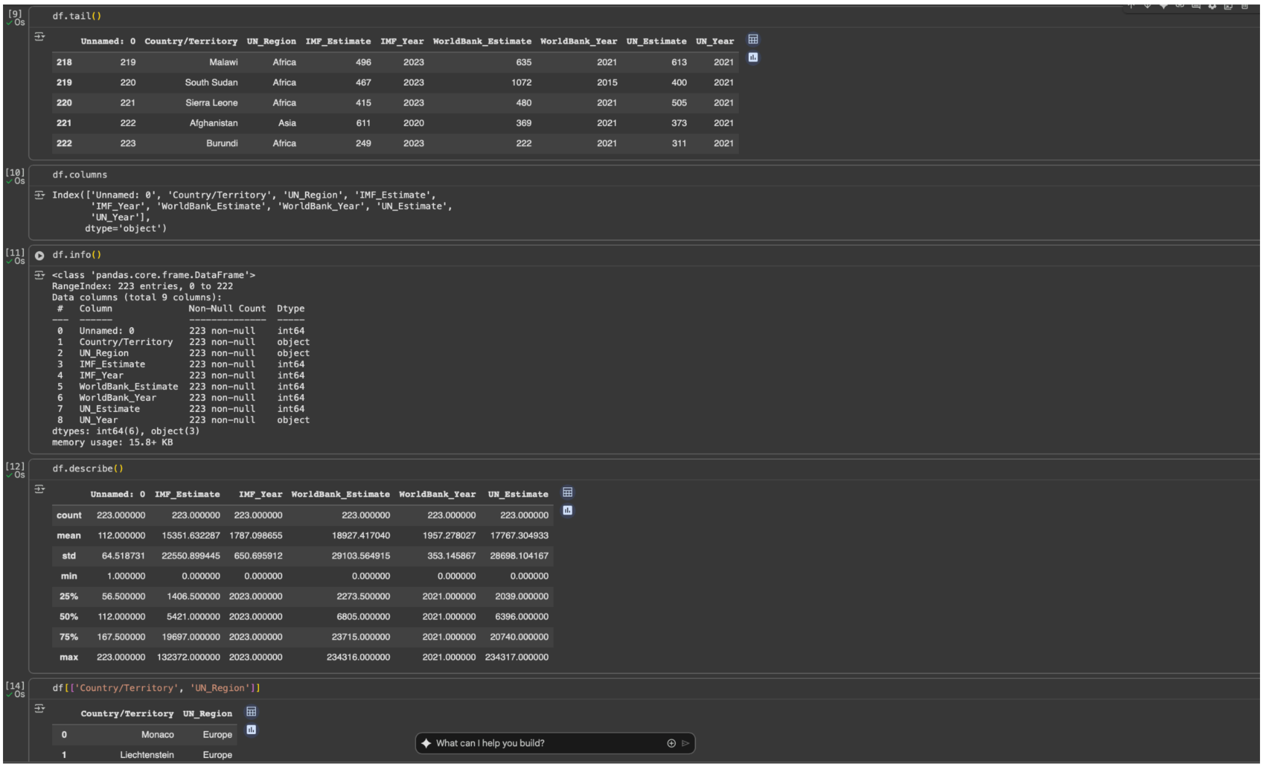
223 rows × 2 columns

Day 4: Task 2

Back with ‘GDP (nominal) per Capita’. As a group, import and work your way through the Day_4_Python_Activity.ipynb notebook which can be found here. There are questions to answer, but also opportunities to have fun with the data – paste your input and output below.

Once complete, and again as a group, work with some more data and have some fun –there is no set agenda for this section, other than to embed the skills developed this week. Paste your input and output below and upon return we’ll discuss progress made.

Additional data found here.



[9]

```
df.tail()
```

	Unnamed: 0	Country/Territory	UN_Region	IMF_Estimate	IMF_Year	WorldBank_Estimate	WorldBank_Year	UN_Estimate	UN_Year
218	219	Malawi	Africa	496	2023	635	2021	613	2021
219	220	South Sudan	Africa	467	2023	1072	2015	400	2021
220	221	Sierra Leone	Africa	415	2023	480	2021	505	2021
221	222	Afghanistan	Asia	611	2020	369	2021	373	2021
222	223	Burundi	Africa	249	2023	222	2021	311	2021

[10]

```
df.columns
```

```
Index(['Unnamed: 0', 'Country/Territory', 'UN_Region', 'IMF_Estimate',
       'IMF_Year', 'WorldBank_Estimate', 'WorldBank_Year', 'UN_Estimate',
       'UN_Year'],
      dtype='object')
```

[11]

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 223 entries, 0 to 222
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   Unnamed: 0          223 non-null    int64 
 1   Country/Territory   223 non-null    object
 2   UN_Region           223 non-null    object
 3   IMF_Estimate        223 non-null    int64 
 4   IMF_Year            223 non-null    int64 
 5   WorldBank_Estimate  223 non-null    int64 
 6   WorldBank_Year      223 non-null    int64 
 7   UN_Estimate         223 non-null    int64 
 8   UN_Year             223 non-null    object
dtypes: int64(6), object(3)
memory usage: 15.8+ KB
```

[12]

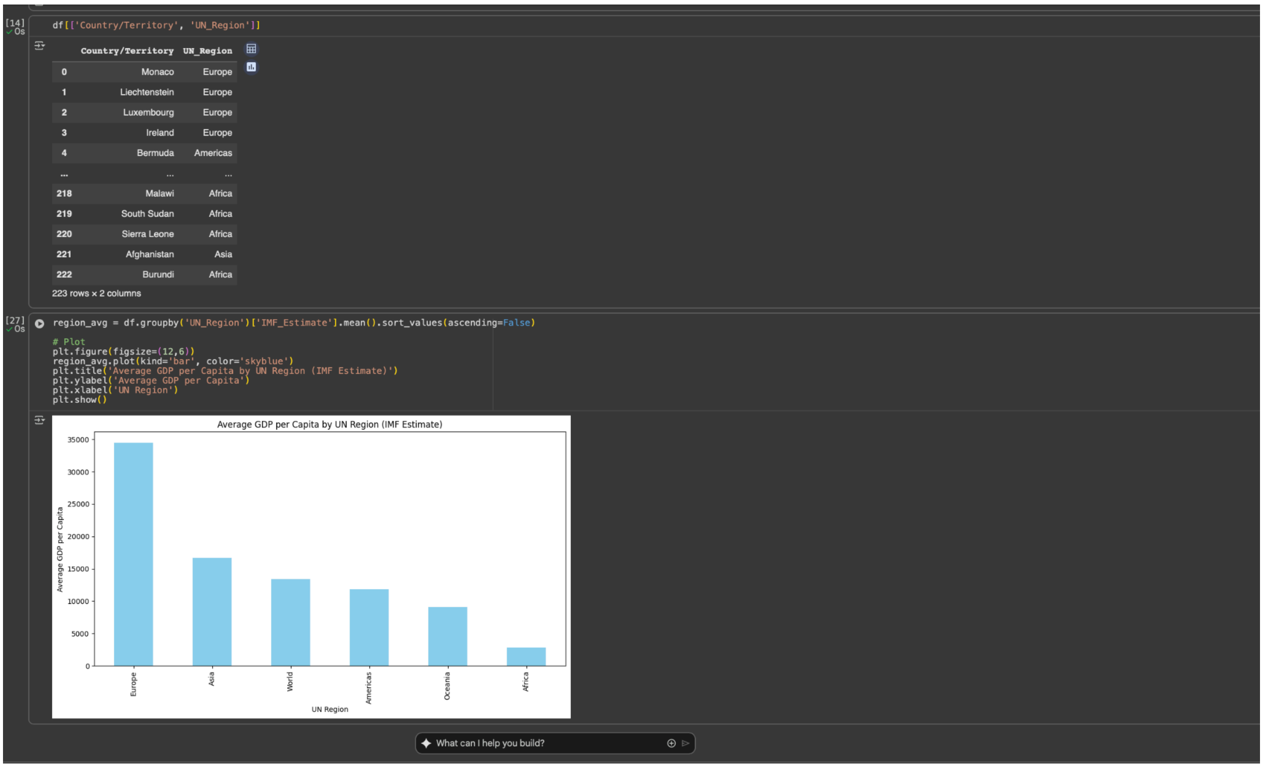
```
df.describe()
```

	Unnamed: 0	IMF_Estimate	IMF_Year	WorldBank_Estimate	WorldBank_Year	UN_Estimate
count	223.000000	223.000000	223.000000	223.000000	223.000000	223.000000
mean	112.000000	15351.632287	1787.098655	18927.417040	1957.278027	17767.304933
std	64.518731	22550.899445	650.695912	29103.564915	353.145867	28698.104167
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	56.500000	1406.500000	2023.000000	2273.500000	2021.000000	2039.000000
50%	112.000000	5421.000000	2023.000000	6805.000000	2021.000000	6396.000000
75%	167.500000	19697.000000	2023.000000	23715.000000	2021.000000	20740.000000
max	223.000000	132372.000000	2023.000000	234316.000000	2021.000000	234317.000000

[14]

```
df[['Country/Territory', 'UN_Region']]
```

	Country/Territory	UN_Region
0	Monaco	Europe
1	Liechtenstein	Europe



[14]

```
df[['Country/Territory', 'UN_Region']]
```

	Country/Territory	UN_Region
0	Monaco	Europe
1	Liechtenstein	Europe
2	Luxembourg	Europe
3	Ireland	Europe
4	Bermuda	Americas
...	...	...
218	Malawi	Africa
219	South Sudan	Africa
220	Sierra Leone	Africa
221	Afghanistan	Asia
222	Burundi	Africa

223 rows × 2 columns

[27]

```
region_avg = df.groupby('UN_Region')['IMF_Estimate'].mean().sort_values(ascending=False)

# Plot
plt.figure(figsize=(12,6))
region_avg.plot(kind='bar', color='skyblue')
plt.title('Average GDP per Capita by UN Region (IMF Estimate)')
plt.ylabel('Average GDP per Capita')
plt.xlabel('UN Region')
plt.show()
```

Course Notes

It is recommended to take notes from the course, use the space below to do so, or use the revision guide shared with the class:

What is Python?

- Python is a high-level, versatile programming language created by Guido van Rossum and released in 1991.
- It emphasizes readability and simplicity with English-like syntax.
- It is free to use and maintained by a global community.
- It has a large ecosystem of libraries for different fields like web development, data science, artificial intelligence, and automation.

Why Learn Python?

Python is a great language for learning because it is:

- **Easy to Learn:** Its simple and readable syntax allows you to start programming quickly.
- **Easy to Use:** You can write new software faster with Python, making it ideal for rapid development.
- **Easy to Access:** It is free, open-source, and works on multiple platforms.

Working with Python

- For this course, you will use the cloud-based platform **Google Colab**, which includes a Jupyter notebook and requires no installation.
- The file extension for a Jupyter notebook is `.ipynb`.

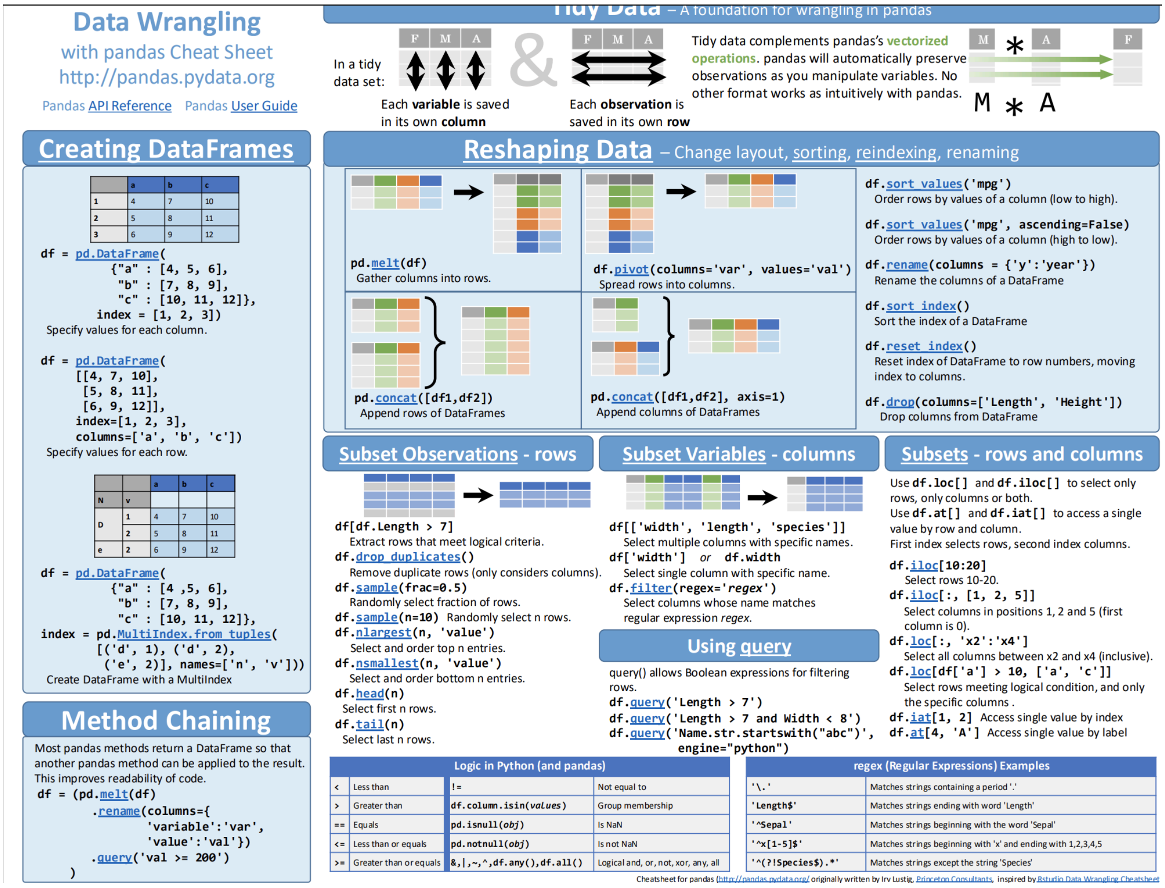
Basic Python Concepts

- **`print()` Function:** This is the command used to send a message to the screen.
- **Quotes:** You must use either single quotes (`''`) or double quotes (`""`) for string data types, but you cannot mix them in the same string.
- **Data Types:**
  - A value in quotes, like `"3.14"`, is a **string (`str`)** type.
  - A value without quotes, like `3.14`, is a **float (numeric)** type.
- **Escape Characters:** A backslash (`\`) is an escape character.
  - `\n` creates a new line.
  - To use a single backslash in a string, you must double it: `\\`.
- **Multiple Arguments:** You can use multiple arguments in a `print()` function by separating them with commas.

- **Comments:** Comments are remarks inserted into the code for humans, not for the program itself. They are not run at execution. In Python, a comment is created by starting a line with the hash character (`#`).
- **Variables:**
  - A variable is declared automatically when you assign a value to it.
  - The structure is `variable_name = value` (e.g., `my_age = 33`).
- **`input()` Function:** The `input()` function reads data entered by the user and returns it as a string.
- **Type Casting:** This is the process of converting one data type to another. You can use functions like `int()` or `float()` to manually convert a string that looks like a number into a numeric type.

Algorithms

- An **algorithm** is a step-by-step procedure designed to perform an operation to achieve a desired result.
- **Key properties of an algorithm:**
  - **Unambiguous:** Each step must be clear and lead to only one meaning.
  - **Input:** Must have zero or more well-defined inputs.
  - **Output:** Must have one or more well-defined outputs that match the desired result.
  - **Finiteness:** Must terminate after a finite number of steps.
  - **Feasibility:** Must be achievable with available resources.
  - **Independent:** The step-by-step directions should not depend on any specific programming code.



Data Wrangling
with pandas Cheat Sheet
http://pandas.pydata.org

Pandas API Reference   Pandas User Guide

Tidy Data – A foundation for wrangling in pandas

F
M
A
&
F
M
A
M
*
A
F
M * A

In a tidy data set:

Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
          {"a" : [4, 5, 6],
           "b" : [7, 8, 9],
           "c" : [10, 11, 12]},
        index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
N	v			
D	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
          {"a" : [4 ,5, 6],
           "b" : [7, 8, 9],
           "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d', 1), ('d', 2),
         ('e', 2)], names=['n', 'v']))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable':'var',
                'value':'val'})
        .query('val >= 200')
     )
```

Reshaping Data – Change layout, sorting, reindexing, renaming

`pd.melt(df)`
Gather columns into rows.

`df.pivot(columns='var', values='val')`
Spread rows into columns.

`pd.concat([df1,df2])`
Append rows of DataFrames

`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg', ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(columns=['Length', 'Height'])`
Drop columns from DataFrame

Subset Observations - rows

`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)` Randomly select n rows.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

Subset Variables - columns

`df[['width', 'length', 'species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

Using query

query() allows Boolean expressions for filtering rows.

```
df.query('Length > 7')
df.query('Length > 7 and Width < 8')
df.query('Name.str.startswith("abc")',
            engine="python")
```

Subsets - rows and columns

Use `df.loc[]` and `df.iloc[]` to select only rows, only columns or both.
Use `df.at[]` and `df.iat[]` to access a single value by row and column.
First index selects rows, second index columns.

`df.iloc[10:20]`
Select rows 10-20.

`df.iloc[:, [1, 2, 5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[:, 'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.loc[df['a'] > 10, ['a', 'c']]`
Select rows meeting logical condition, and only the specific columns .

`df.iat[1, 2]` Access single value by index

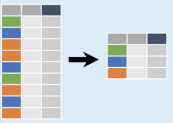
`df.at[4, 'A']` Access single value by label

Logic in Python (and pandas)			
<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

regex (Regular Expressions) Examples	
'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
'^(?!Species$).*'	Matches strings except the string 'Species'

Cheatsheet for pandas (http://pandas.pydata.org/ originally written by Irv Lustig, Princeton Consultants, inspired by Rstudio Data Wrangling Cheatsheet

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.
`rank(method='dense')`
Ranks with no gaps.
`rank(method='min')`
Ranks. Ties get min rank.
`rank(pct=True)`
Ranks rescaled to interval [0, 1].
`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.
`cumsum()`
Cumulative sum.
`cummax()`
Cumulative max.
`cummin()`
Cumulative min.
`cumprod()`
Cumulative product.

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable
`len(df)`
# of rows in DataFrame.
`df.shape`
Tuple of # of rows, # of columns in DataFrame.
`df['w'].nunique()`
# of distinct values in a column.
`df.describe()`
Basic descriptive and statistics for each column (or GroupBy).
`df.info()`
Prints a concise summary of the DataFrame.
`df.memory_usage()`
Prints the memory usage of each column in the DataFrame.
`df.dtypes()`
Prints a Series with the dtype of each column in the DataFrame.

pandas provides a large set of summary functions that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.
`count()`
Count non-NA/null values of each object.
`median()`
Median value of each object.
`quantile([0.25,0.75])`
Quantiles of each object.
`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.
`max()`
Maximum value in each object.
`mean()`
Mean value of each object.
`var()`
Variance of each object.
`std()`
Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.
`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.
`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.
`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

Vector function
Vector function

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.
`abs()`
Absolute value.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.
`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

➕

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

➕

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

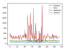
`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

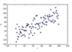
```
pd.merge(ydf, zdf, how='outer',
            indicator=True)
.query('_merge == "left_only"')
.drop(columns=['_merge'])
```
Rows that appear in ydf but not zdf (Setdiff).

Plotting

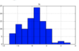
`df.plot()`
Plot a line graph for the DataFrame.



`df.plot.scatter(x='w', y='h')`
Plot a scatter graph of the DataFrame.



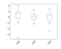
`df.plot.hist()`
Plot a histogram of the DataFrame.



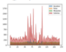
`df.plot.pie()`
Plot a pie chart of the DataFrame.

`df.plot.bar()`
Plot a line graph for the DataFrame.

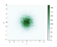
`df.plot.boxplot()`
Plot a scatter graph of the DataFrame.



`df.plot.area()`
Plot an area graph of the DataFrame.



`df.plot.hexbin()`
Plot a hexbin graph of the DataFrame.



`df.plot(subplots=True)`
Separate into different graphs for each column in the DataFrame.
`df.plot(title="Graph of A against B")`
Sets the title of the graph.

`df.plot(cumulative=True)`
Creates a cumulative plot
`df.plot(bins=30)`
Set the number of bins into which data is grouped (histograms)

`df.plot(stacked=True)`
Stacks the data for the columns on top of each other. (bar, barh and area only)
`df.plot(alpha=0.5)`
Sets the transparency of the plot to 50%.

`df.plot(subplots=True, title=['col1', 'col2', 'col3'])`
Arguments can be combined for more flexibility when graphing, this would plot a separate line graph for of column of a 3-columned DataFrame. The first string in the list of titles applies to the graph of the left-most column.

Changing Type

`pd.to_numeric(data)`
Convert non-numeric types to numeric.
`pd.to_datetime(data)`
Convert non-datetime types to datetime type
`pd.to_timedelta(data)`
Convert non- timedelta types to timedelta

`df.astype(type)`
Convert data to (almost) any given type including categorical
`df.infer_objects()`
Attempts to infer a better type for object type data.
`df.convert_dtypes()`
Convert columns to best possible dtypes

Datetime

With a Series containing data of type datetime, the dt accessor is used to get various components of the datetime values:

`s.dt.year`
Extract the year
`s.dt.month`
Extract the month as an integer.

`s.dt.day`
Extract the day (int) from the date.
`s.dt.quarter`
Find which quarter the date lies in.
`s.dt.hour`
Extract the hour.
`s.dt.minute`
Extract the minute.
`s.dt.second`
Extract the second.

Mapping

Apply a mapping to every element in a DataFrame or Series, useful for recategorizing or transforming data.

`s.map(lambda x: 2*x)`
Returns a copy of the series where every entry is doubled
`df.apply(lambda s: s.max() - s.min(), axis=1)`
Returns a Series with the difference of the maximum and minimum values of each row of the DataFrame

Series String Operations

Similar to python string operations, except these are vectorized to apply to the entire Series efficiently.

`s.str.count(pattern)`
Returns a series with the integer counts in each element.
`s.str.get(index)`
Returns a series with the data at the given index for each element.
`s.str.join(sep)`
Returns a series where each element has been concatenated.
`s.str.title()`
Converts the first character of each word to be a capital.
`s.str.len()`
Returns a series with the lengths of each element.

`s.str.cat()`
Concatenate elements into a single string
`s.str.partition(sep)`
Splits the string on the first instance of the separator
`s.str.slice(start, stop, step)`
Slices each string
`s.str.replace(pat, rep)`
Use regex to replace patterns in each string.
`s.str.isalnum()`
Checks whether each element is alpha-numeric

Input/Output

Common file types for data input include CSV, JSON, HTML which are human-readable, while the common output types are usually more optimized for performance and scalability such as feather, parquet and HDF.

`df = pd.read_csv(filepath)`
Read data from csv file
`df = pd.read_html(filepath)`
Read data from html file
`df = pd.read_excel(filepath)`
Read data from xls (and related) files
`df = pd.read_sql(filepath)`
Read data from sql file
`pd.read_clipboard()`
Read text from clipboard

`df.to_parquet(filepath)`
Write data to parquet file
`df.to_feather(filepath)`
Write data to feather file
`df.to_hdf(filepath)`
Write data to HDF file
`df.to_clipboard()`
Copy object to the system clipboard

Frequently Used Options

Pandas offers some 'options' to globally control how Pandas behaves, display etc. Options can be queried and set via:
`pd.options.`*`option_name`* (where *option_name* is the name of an option). For example:
`pd.options.display.max_rows = 20`
Set the **`display.max_rows`** option to 20.

Functions

`get_option(option)`
Fetch the value of the given option.
`set_option(option)`
Set the value of the given option.
`reset_option(options)`
Reset the values of all given options to default settings.
`describe_option(options)`
Print descriptions of given options.
`option_context(options)`
Execute code with temporary option settings that revert to prior settings after execution.

Display options

`display.max_rows`
The maximum number of rows displayed in pretty-print.
`display.max_columns`
The maximum number of columns displayed in pretty-print.
`display.expand_frame_repr`
Controls whether the DataFrame representation stretches across pages.
`display.large_repr`
Controls whether a DataFrame that exceeds maximum rows/columns is truncated or summarized
`display.precision`
The output display precision in decimal places.
`display.max_colwidth`
The maximum width of columns, longer cells will be truncated.
`display.max_info_columns`
The maximum number of columns displayed after calling **`info()`**.
`display.chop_threshold`
Sets the rounding threshold to zero when displaying a Series/DataFrame.
`display.colheader_justify`
Controls how column headers are justified.

We have included a range of additional links to further resources and information that you may find useful, these can be found within your revision guide.

END OF WORKBOOK

Please check through your work thoroughly before submitting and update the table of contents if required.

Please send your completed work booklet to your trainer.