

**Data Technician**

|  |
| --- |
|  |

|  |
| --- |
| Name: |
| Course Date: |
|  |

**Table of contents**

[Day 2: Task 1 3](#_Toc1765240926)

[Day 3: Task 1 3](#_Toc873857632)

[Exercise 1: Loading and Exploring the Data 4](#_Toc1363711086)

[Exercise 2: Indexing and Slicing 4](#_Toc1679584900)

[Exercise 3: Data Manipulation 4](#_Toc998449442)

[Exercise 4: Aggregation and Grouping 5](#_Toc267294297)

[Exercise 5: Advanced Operations 5](#_Toc1232452443)

[Exercise 6: Exporting Data 6](#_Toc1251390377)

[Exercise 7: If finished early try visualising the results 6](#_Toc66584577)

[Day 4: Task 1 7](#_Toc1452452084)

[Day 4: Task 2 8](#_Toc934594145)

[Course Notes 8](#_Toc1227711066)

[Additional Information 9](#_Toc1836543406)

# 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 | #Fizzbuzz activity  for num in range(1,101): #chekcing number from 1 to 101 range      if num % 3 == 0 and num % 5 == 0: #checking number devide by 3 and 5          print("fizzbuzz")      elif num % 3 == 0: #checking number devide by 3          print("fizz")      elif num % 5 == 0:# checking number devide 5          print("buzz")      else:          print(num) |

# **Day 3: Task 1**

Download the ‘student.csv’, 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."

|  |
| --- |
| # import pandas library  import pandas as pd  1).  # Load the CSV file  df = pd.read\_csv("student.csv")  print(f"Successfully loaded '{csv\_filename}' into a DataFrame. Now let's explore it.")  2). # Display the first 5 rows if the DataFrame exists  if not df.empty:      print("First 5 rows (df.head()):")      display(df.head())  3).  #3. To get the information about the DataFrame  if not df.empty:      # Get a summary of the DataFrame      print("DataFrame Information (df.info()):")      df.info()  else:       print("DataFrame is empty.")  4).  #4.the code to get summary statistics for the DataFrame."  if not df.empty:      # Get summary statistics for numerical columns      print("\nSummary Statistics (df.describe()):")      display(df.describe())  else:       print("DataFrame is empty.") |

### **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.  # Select the 'name' column  if not df.empty:      name\_column = df['name']      print("Specific Column (df['name']):")      print(name\_column)  else:       print("DataFrame is empty.")      2).  #The code to select the 'name' and 'mark' columns."  if not df.empty:      name\_column = df['name']      mark\_column = df['mark']      print("Specific Columns (df['name']) (df['mark']):")      print(name\_column, mark\_column)  else:       print("DataFrame is empty.")    3).#Write the code to select the first 3 rows  if not df.empty:      print("First 3 rows (df.head()):")      display(df.head(3))    4). The code to select all rows where the 'class' is 'Four'."  # Select the 'name' column  if not df.empty:      print("Specific Columns (df[['Class', 'Four']]):")  #Print the specified columns      print(df[df["class"] == "Four"])  else:       print("DataFrame is empty.") |

### **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). # Add 'passed' column (Pass if mark >= 60, otherwise Fail)  df["passed"] = df["mark"].apply(lambda x: "Pass" if x >= 60 else "Fail")  display(df.head())    2). "Write the code to rename the 'mark' column to 'score'."  # Rename the 'mark' column to 'score'  df.rename(columns={"mark": "score"}, inplace=True)  display(df.head())    3.Write the code to drop the 'passed' column  # Drop the 'passed' column  df.drop(columns=["passed"], inplace=True)  display(df.head()) |

### **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.  #Renaming the column to mark  df.rename(columns={"score": "mark"}, inplace=True)  display(df.head())  # Group by 'class' and calculate the mean of 'mark'  class\_mean\_mark = df.groupby("class")["mark"].mean()  print(class\_mean\_mark)    2) Write the code to count the number of students in each class  # Count the number of students in each class  students\_per\_class = df["class"].value\_counts()  print(students\_per\_class)    3). Write the code to calculate the average mark for each gender  # Group by 'gender' and calculate the mean 'mark'  average\_mark\_by\_gender = df.groupby("gender")["mark"].mean()  display(average\_mark\_by\_gender) |

### **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).  # Create the pivot table  pivot\_table = df.pivot\_table(values="mark", index="class", columns="gender", aggfunc="mean")  print(pivot\_table)    2). 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'."  # Function to assign grades based on marks  def assign\_grade(mark):  if mark >= 85:  return "A"  elif mark >= 70:  return "B"  elif mark >= 60:  return "C"  else:  return "D"  display(df.head())  # Apply the function to create the 'grade' column  df["grade"] = df["mark"].apply(assign\_grade)  display(df.head())    3). Write the code to sort the DataFrame by 'mark' in descending order  # Apply the function to create the 'grade' column  df["grade"] = df["mark"].apply(assign\_grade)  print(df)    # Sort the DataFrame by 'mark' in descending order  df\_sorted = df.sort\_values("mark", ascending=False)  print("\n \*\*\*\*\*\*\*Marks in ascending order \*\*\*\*\*\*\*\n",df\_sorted) |

### **Exercise 6: Exporting Data**

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

|  |
| --- |
| # Mount Google Drive  from google.colab import drive  drive.mount('/content/drive')  # Define the file path where the DataFrame will be saved  file\_path = '/content/drive/My Drive/df\_transformed.csv'  # Change path if needed    # Save the DataFrame as a CSV file without the index  df\_transformed.to\_csv(file\_path, index=False)    # Optional: Confirm that the file was saved successfully  import os  print("File saved:", os.path.exists(file\_path)) |

### **Exercise 7: If finished early try visualising the results**

|  |
| --- |
|  |

# **Day 4: Task 1**

Using the ‘GDP (nominal) per Capita.csv’ which can be downloaded from the shared Folder, 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 Jyputer notebook
* Print the first 10 rows
* Print the last 5 rows
* Print ‘Country/Territory’ and ‘UN\_Region’ columns

|  |
| --- |
| Please add Panda library  import pandas as pd  Upload CV file to Google colab.  Create a dataframe object to read CSV file.  1). df = pd.read\_csv("GDP (nominal) per Capita.csv",encoding= 'unicode\_escape',  index\_col=0)  print(f"Successfully loaded '{df}' into a DataFrame. Now let's explore it.")    2).  # Display the first 10 rows if the DataFrame exists  if not df.empty:      print("First 10 rows (df.head()):")      display(df.head(10))    3).# Display the last 5 rows if the DataFrame exists  if not df.empty:      print("Last 5 rows (df.tail()):")      display(df.tail())    4). # Print ‘Country/Territory’ and ‘UN\_Region’ columns  if not df.empty:      print("Specific Columns (df[['Country/Territory', 'UN\_Region']]):")  # Print the specified columns  print(df[['Country/Territory', 'UN\_Region']]) |

# **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 on the shared Folder. 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.](https://justit831-my.sharepoint.com/:f:/g/personal/danpe_justit_co_uk/Er0ybU9i0AZKiuGaCWZyj2ABoqKD23zwLGdJf3WlaixpRA?e=QVj2Bs)

|  |
| --- |
| # number of countries per region  countires\_count= df["UN\_Region"].value\_counts()  print(countires\_count)    2 #What is European Union[n 1]?  if not df.empty:      # Get a summary of the DataFrame      print("DataFrame Information (df.info()):")      #df.info()      #df.describe()      #print("Specific Columns (df[['Europe', 'First\_Contry']]):")    # Display the first few rows to understand the structure      df.head()      print(df.head())    #Print the specified columns     # cnt\_name = df['Un\_Region'][0]      #print(cnt\_name)  else:       print("DataFrame is empty.")  .    europe\_df = df[df['UN\_Region'] == 'Europe'].copy()  print(europe\_df)    europe\_df['UN\_Estimate'] = pd.to\_numeric(europe\_df['UN\_Estimate'], errors='coerce')    europe\_avg = europe\_df['UN\_Estimate'].mean()  print(europe\_avg)  # Countries in Europe below avarege  #if not df.empty:  europe\_df = df[df['UN\_Region'] == 'Europe'].copy()  #print(europe\_df)    europe\_df['UN\_Estimate'] = pd.to\_numeric(europe\_df['UN\_Estimate'], errors='coerce')    europe\_avg = europe\_df['UN\_Estimate'].mean()  #print("List of all Europe Contries Average",europe\_avg)  below\_avg\_europe = europe\_df[europe\_df['UN\_Estimate'] < europe\_avg]  #print("List of contires below average in Europe",below\_avg\_europe)  print(below\_avg\_europe[['Country/Territory', 'UN\_Estimate']])  #else:       #print("DataFrame is empty.")    # Filter the DataFrame for Europe only europe\_df = df[df['UN\_Region'] == 'Europe'].copy()    # Convert UN\_Estimate column to numeric (in case of parsing issues) europe\_df['UN\_Estimate'] = pd.to\_numeric(europe\_df['UN\_Estimate'], errors='coerce')    # Extract UK GDP value uk\_gdp\_row = europe\_df[europe\_df['Country/Territory'] == 'United Kingdom'] if not uk\_gdp\_row.empty:     uk\_gdp = uk\_gdp\_row['UN\_Estimate'].values[0]     print("UK GDP:", uk\_gdp)        # Filter countries with GDP higher than the UK     higher\_gdp\_df = europe\_df[europe\_df['UN\_Estimate'] > uk\_gdp]        if not higher\_gdp\_df.empty:         print("European countries with GDP higher than the UK:")         print(higher\_gdp\_df[['Country/Territory', 'UN\_Estimate']])     else:         print("No European country has a higher GDP than the UK.") else:     print("United Kingdom data not found in the dataset.")      # Step 1: Calculate the average of the UN\_Estimate  average\_un\_estimate = df.groupby('Country/Territory')['IMF\_Estimate'].mean()  #Step 2: Filter countries where the UN\_Estimate is below the average  below\_average\_countries = df[df['IMF\_Estimate'] < average\_un\_estimate.mean()]  #print(average\_un\_estimate)  #print(below\_average\_countries)  # Step 3: Group the filtered countries by 'Region' and apply aggregation (e.g., count the number of countries per region)  #grouped = below\_average\_countries.groupby('Country/Territory')  #print(grouped)  # Step 4: Display the result  #print("Group by region with countries below the average IMF world estimate:")  print(below\_average\_countries[['Country/Territory','IMF\_Estimate','UN\_Region']])    # Step 1: Calculate the IMF\_Estimate 0  grouped\_df = df.groupby('Country/Territory')['IMF\_Estimate'].sum()  # Use sum() if needed to aggregate  # Filter countries where the IMF\_Estimate is 0  countries\_with\_zero\_estimate = grouped\_df[grouped\_df == 0]  # Display the result  print(countries\_with\_zero\_estimate)    highest\_un\_estimate\_country = df.loc[df['UN\_Estimate'].idxmax()]  # Display the country and its UN Estimate  print("Country with the highest UN Estimate:")  print(highest\_un\_estimate\_country)    highest\_wrlbnk\_estimate = df.loc[df['WorldBank\_Estimate'].idxmax()]  # Display the country and its UN Estimate  print("The highest World Bank Estimate:")  print(highest\_wrlbnk\_estimate)    highest\_imf\_estimate = df.loc[df['IMF\_Estimate'].idxmax()]  # Display the country and its UN Estimate  print("The highest IMF Estimate:")  print(highest\_imf\_estimate)    mean\_value = df[df['WorldBank\_Estimate'] != 0]['WorldBank\_Estimate'].mean()  # Fill the 0 values in the 'WorlBank\_Estimate' column with the calculated mean  df['WorldBank\_Estimate'] = df['WorldBank\_Estimate'].replace(0, mean\_value)  # Display the updated DataFrame  print(df)    # Drop the temporary 'avg\_worldbank\_un' column if not needed    # Calculate the average of 'Worldbank\_Estimate' and 'UN\_Estimate' columns  mean\_value = df[df['WorldBank\_Estimate'] != 0]['WorldBank\_Estimate'].mean()    # Fill the 0 values in the 'UN\_Estimate' column with the calculated mean  df['WorldBank\_Estimate'] = df['WorldBank\_Estimate'].replace(0, mean\_value)    drop\_columns = ['WorldBank\_Estimate']  df = df.drop(columns=drop\_columns, errors='ignore')    # Display the updated DataFrame  print(df)      mean\_value = df[df['IMF\_Estimate'] != 0]['IMF\_Estimate'].mean()  # Fill the 0 values in the 'UN\_Estimate' column with the calculated mean  df['IMF\_Estimate'] = df['IMF\_Estimate'].replace(0, mean\_value)  # Display the updated DataFrame  print(df)    if not df.empty:      # Count missing values per column      print("Missing values per column (df.isnull().sum()):")      missing\_per\_column = df.isnull().sum()      print(missing\_per\_column)        # Count total missing values in the DataFrame      total\_missing = missing\_per\_column.sum()      # Alternative: df.isnull().sum().sum()      print(f"\nTotal missing values in the DataFrame: {total\_missing}")  else:       print("DataFrame is empty.")      # Fill the null values in 'imf' column with the calculated average    # Calculate the average of 'Worldbank\_Estimate' and 'UN\_Estimate' columns  mean\_value\_IMF = df[df['IMF\_Estimate'] != 0]['IMF\_Estimate'].mean()    # Fill the 0 values in the 'UN\_Estimate' column with the calculated mean  df['IMF\_Estimate'] = df['IMF\_Estimate'].replace(0, mean\_value)    # Display the updated DataFrame  print(df)    **Importing Matploitlib and Seaborn library for visualisation:**  import matplotlib.pyplot as plt  import seaborn as sns    Histogram chart:    df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].hist(figsize=(12,9))  plt.show()    df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].hist(bins=5, figsize=(12,9))  plt.show()      Add Bins to Histogram:    df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].hist(bins=15, figsize=(15,12))  #23400/15 = 15300  plt.show()    Correlation Heatmap      corr = df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].corr()  plt.figure(figsize=(9,6))  sns.heatmap(corr, annot=True)  plt.show()    corr = df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].corr()  plt.figure(figsize=(9,6))  sns.heatmap(corr, annot=True, fmt=".2f", cmap = 'GnBu', annot\_kws={"size": 12})  plt.show()    corr = df[["IMF\_Estimate", "UN\_Estimate", "WorldBank\_Estimate"]].corr()  plt.figure(figsize=(9,6))  sns.heatmap(corr, annot=True, cmap = 'Purples')  plt.title("Correlation Map")  plt.show()    corr = df.select\_dtypes(include=[int, float]).corr()    plt.figure(figsize=(9,6))    sns.heatmap(corr, annot=True, cmap = 'Purples')    plt.show()    Bar plot:            Scatter Plot    Boxplot and Outliers            Create another dataframe called data excluding 5 countries with highest UN estimate  addCode addText      Removing outliers |

|  |
| --- |
| **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:

|  |
| --- |
| **Additional Information** |

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.**