

**ANL252**

**Python for Data Analytics**

**Group-Based Assignment**

**July 2022 Presentation**

**Submitted by:**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Muhammad Azfar Bin Razak** | **H2170546** |
| **Shun Pei Ling Felicia** | **H1882537** |
| **Wong Hang Sin** | **Q2071045** |
| **Muhammad Irfan Muhaimin Bin Ahmad Jasman** | **K2172687** |

**Tutorial Group: ­­­­­­­­­­­­ T 09**

**Instructor’s Name: Dr. Kumar Munish**

**Submission Date: 28/08/2022**

**Declaration Page**

We, members of group 6, do hereby declare that we each contributed to this assignment and that we collectively agree to a shared grade.

|  |  |  |
| --- | --- | --- |
| Name | Contribution | Signature |
| Muhammad Azfar Bin Razak (Team Lead) | I did questions d, e |  |
| Shun Pei Ling Felicia | I did questions a, c |  |
| Muhammad Irfan Muhaimin Bin Ahmad Jasman | I did question b |  |
| Wong Hang Sin | I did question d |  |

**Contents**

[Question 1(a) 1](#_Toc112585923)

[Code 1](#_Toc112585924)

[Question 1(b) 2](#_Toc112585925)

[Question 1(c) 8](#_Toc112585926)

[1st Data Quality Issue 8](#_Toc112585927)

[Code 8](#_Toc112585928)

[2nd Data Quality Issue 10](#_Toc112585929)

[Code 10](#_Toc112585930)

[3rd Data Quality Issue 12](#_Toc112585931)

[Code 12](#_Toc112585932)

[Question 1(d) 14](#_Toc112585933)

[Code 14](#_Toc112585934)

[Output Cell 14](#_Toc112585935)

[Question 1(e) 15](#_Toc112585936)

[Visual 1 - Number of commuters that start their journey at the certain hour of the day 15](#_Toc112585937)

[Figure 1: Frequency of Start Journey of Commuters 15](#_Toc112585938)

[Table 1: Count of Commuters 15](#_Toc112585939)

[Code 16](#_Toc112585940)

[Visual 2 - Count of Commuters By Age Group 17](#_Toc112585941)

[Figure 2: Frequency of Commuters by Age Group 17](#_Toc112585942)

[Table 2: Age Group 17](#_Toc112585943)

[Code 17](#_Toc112585944)

[Visual 3 - Count of Gender 19](#_Toc112585945)

[Figure 3: Count of Commuters by Gender % 19](#_Toc112585946)

[Table 3: Count of Commuters by Gender 19](#_Toc112585947)

[Code 19](#_Toc112585948)

[References 21](#_Toc112585949)

# Question 1(a)

|  |  |
| --- | --- |
| Cell | Code |
| 1 | # import relevant modules with pd as an alias for pandas and plt as an alias for matplotlib.pyplot  import pandas as pd  import numpy as np  import random  import matplotlib.pyplot as plt  from datetime import date  # read csv file "GBA\_Data.csv" into a dataframe name employee\_data  journey\_data = pd.read\_csv("GBA\_Data.csv")  # capture dataframe  df = pd.DataFrame(journey\_data)  df |

# Question 1(b)

Identifying Missing Values ‘-’, ‘--’, and ‘?’

There are a few ways to treat the missing value, either by replacing them or deleting them. However, before treating them, we would need to identify the variable columns for which these missing values denoted as ‘-’, ‘--’, and ‘?’are present. After importing the data and capturing it as a DataFrame, we will use a for-loop to identify these variable columns as shown in Cell 2 below. Based on the output of the for-loop, it is noted that 4 variables namely, gender, type, age and yob, contain the missing values.

|  |  |
| --- | --- |
| Cell | Code |
| 2 | # identify columns with '-', '--', '?' denoted as missing values  for cols in df.columns:  if '?' in set(df[cols]) or '-' in set(df[cols]) or '--' in set(df[cols]):  print(cols) |

For ease in the later part of the programming, we will replace the missing values with np.nan as shown in Cell 3.

|  |  |
| --- | --- |
| Cell | Code |
| 3 | # replace missing values with np.nan for ease  df.replace(["-", "--", "?"], [np.nan, np.nan, np.nan], inplace=True) |

Replacing Missing Value from Type Column

Firstly, we would print out the rows containing the missing values from the type column using .isnull(). Based on the output, we realized that the age of the users of these rows ranges from 22 to 46 years old. Further analysis of the original dataset has suggested that the age range for concession, regular and ad-hoc type of users falls between 55 to 65 years old, 25 to 71 years old and all ages respectively. With that, since the age of the users for the missing values under type range between 22 to 46 years of age and which falls in the range observed for regular and ad-hoc users, we can reasonably assume the profile type of these users and replace them with either regular or ad-hoc randomly. This is shown below in Cell 5, where the values, Regular and Ad-hoc, are assigned to randomtype.

By assigning dftype to df[‘type’].isnull(), it would allow us to change the missing value on the type column. As we have already figure out from which row that has missing value from type column, we could use .loc[] method to print out the row to ensure and check whether the data have been replaced to the random numbers.

The final output is then verified using the .loc method to access the rows which were affected.

|  |  |
| --- | --- |
| Cell | Code |
| 4 | # print type with missing data  print(df[df['type'].isnull()]) |
| 5 | # Replace missing values from type column  ## Random Type  ## Realised that the age range for the missing value of type are from 22-46.  ## Age range for concession is 55-65  ## Age range for regular is 25-71  ## Age range for Ad-Hoc is all age    ##Hence, will only random type to regular and ad-hoc  randomtype = ['Regular', 'Ad-hoc']    # replace missing values  np.random.seed(0)  dftype = df['type'].isnull()  df.loc[dftype, 'type'] = np.random.choice (randomtype, size = dftype.sum())    # print to check row that initial have missing values from type column have been randomise based on regular and ad-hoc  df.loc[570000:570009,] |

Replacing Missing Value from Gender Column

Similar to type, the missing values would be replaced. In the case of Gender, we have discovered 3 rows under which the gender column contains the missing values. These rows contain an id 666, 3623 and 238.

An in-depth analysis using the .loc() method in Cell 7 on rows containing only the id 666 have revealed interesting patterns. In other words, there has been another user record of id 666 that is of the same age and uses the shared mobility device from the same start location as the row containing id 666 for which the gender column has a missing value. It was revealed that it belongs to a male user. Taking these similar characteristics into consideration, we can safely presume that the row containing id 666 and its respective missing gender value pertains to the same male user.

In this case, we use .loc() method to assess the row and replace the missing value to ‘Male’.

For the remaining two rows where the gender is unknown and where there are no distinct patterns, we have decided to change it to ‘Other’ as we find it fit to classify them by using .fillna() method. Once modified, we print the rows using the .loc method to display the affected rows and check whether the missing values have been replaced to ‘Other’.

|  |  |
| --- | --- |
| Cell | Code |
| 6 | # print gender with missing values  print(df[df['gender'].isnull()]) |
| 7 | ## analysis of individual column  df.loc[(df['id'] == 666) & (df['age'] == 43)] |
| 8 | ## Replace missing values from gender  ## since gender is unknown, best to classify the missing value to others  ## Except for id 666 as we found out that the user is a male    df.update(df.loc[(df['id'] == 666)].replace(np.nan, 'Male'))  df["gender"].fillna("Other", inplace = True)    ## Print to check row that initial have missing data from gender column have changed to others  df.loc[7000:7002,] |

Replacing Missing Values from Age Column

Similarly, we will use the .isnull() method to identify the rows that contain missing values under the yob and age columns.

Based on the output, we have observed that these rows containing missing values under yob and age columns are predominantly regular type users with only 1 concession type. As mentioned above, the analysis has shown that the age range for concession, regular and ad-hoc types varies from 55 to 65 years old, 25 to 71 years old and all ages respectively. The analysis is performed using the .loc method as shown in Cell 10 to access rows containing certain conditions to better understand its characteristics. In this case, we will fill up the missing value based on randomization from the age range of 55 to 65 years old for the concession type user using the .randint() method. On the other hand, the ages of the remaining regular type users would be randomized between the age range of 25 - 71 years old. This is illustrated in Cell 11 and 12 respectively. Similarly, we would then print the rows using the .loc method to verify that the missing values have been replaced randomly within the appropriate ranges.

|  |  |
| --- | --- |
| Cell | Code |
| 9 | # print age and yob with missing values  print(df[df['yob'].isnull()])  print(df[df['age'].isnull()]) |
| 10 | # demonstration of how an analysis of a row is done to identify patterns  df.loc[(df['id'] == 2157) & (df['type'] == 'Regular')] |
| 11 | ## Replacing missing values from age  ## Since most type of the missing value from age are regular and 1 concession  # replace missing value under age for concession type using random range between 55 to 65  random.seed(0)  df.loc[df.type == 'Concession', 'age'] = df.loc[df.type == 'Concession', 'age'].fillna(random.randint(55,65))  # Print to check whether initial missing value from age have been replaced appropriately as intended  df.loc[18000:18009,] |
| 12 | # assign age range to randomage  randomage = range (25,71)  # replace missing values under age for regular type using random range between 25 to 71  np.random.seed(0)  dfage = df['age'].isnull()  df.loc[dfage, 'age'] = np.random.choice (randomage, size = dfage.sum())  # Print to check whether initial missing value from age have been replaced appropriately as intended  df.loc[18000:18009,] |

Replacing Missing Values from Yob Column

The yob column should tally with the age column. In this case, an analysis of the dataset has revealed that the age of the users are reflected as of the year 2020. Since we have obtained their ages in the previous section, we can deduce the yob of the users by using year 2020 as the base year using arithmetic operators.

However, prior to that, we would have to determine whether the data type of the variables are classified correctly so that mathematical operations can be performed with ease. The .info() method is used to provide a basic overview of the DataFrame. Based on its output, we have discovered that the data types for age and yob, have been classified wrongly as ‘object’. These variables are quantitative in nature and hence, should be classified as integers. The .astype() method is used to convert them to integers so that mathematical operations can be conducted. In this case, the age column for rows containing the missing yob value will be deducted from the value of 2020 to compute its corresponding yob. The output is then verified using .loc method to assess the affected rows.

|  |  |
| --- | --- |
| Cell | Code |
| 13 | # obtain basic overview of DataFrame  df.info() |
| 14 | # convert datatype for age and yob to integer  df['age'].replace(np.nan, 0, inplace=True)  df['yob'].replace(np.nan, 0, inplace=True)  df['age'] = df['age'].astype(int)  df['yob'] = df['yob'].astype(int)  # verify datatype for age and yob  df.info() |
| 15 | # use mathematical operation to calculate yob from age  df\_yob = df.loc[18000:18009,]  df\_yob.loc[18000:18009,]['yob'] = 2020 - df\_yob.loc[18000:18009,]['age'] |
| 16 | # Print to check whether initial missing values from yob have been changed appropriately as intended  df.loc[18000:18009,] |

# Question 1(c)

## 1st Data Quality Issue

One of the data quality issues identified relates to the mixed data types for certain variables. This issue has been picked and raised in Cell 13 above where age and yob variables are classified incorrectly as ‘object’. It shall be reiterated here for the purpose of this question.

A solution would be to convert the data types of these variables to integers to better suit its quantitative nature. If these data were to be stored as string, it would not be possible to conduct arithmetic mathematical operations for statistical analysis.

Explanation of Code

The .info() method is used initially to provide us with basic information about our DataFrame. By using the .info() method, we were able to obtain information such as the data types for each of the variables. Based on the output, we have noticed that two variables namely, age and yob, have been classified wrongly as ‘object’. Age and yob are quantitative in nature and hence, should be classified as integers.

|  |  |
| --- | --- |
| Cell | Code |
| 13 | # obtain basic overview of DataFrame  df.info() |

The .astype() method shall be used to convert the data types for age and yob. ‘int’ is placed within the round brackets of the .astype() method to change the data type to integer. By casting those two columns to .astype(int) method, Python would transform the type of all its data values under age and yob columns to integer.

Finally, the .info() method is used to verify the data types for age and yob variables to ensure that they have been modified appropriately. The results have shown our intended outcome as the respective data types have been changed to ‘int64’, which means integer.

|  |  |
| --- | --- |
| Cell | Code |
| 14 | # convert datatype for age and yob to integer  df['age'].replace(np.nan, 0, inplace=True)  df['yob'].replace(np.nan, 0, inplace=True)  df['age'] = df['age'].astype(int)  df['yob'] = df['yob'].astype(int)  # verify datatype for age and yob  df.info() |

## 2nd Data Quality Issue

There is too much data under the start and end variable columns. The start and end time contains information from the date and hours up to its milliseconds. In particular, the data under these columns are too specific to the milliseconds, which renders it meaningless and insignificant.

An approach to this would be to remove these milliseconds information. Since the start and end time have included the date, hours and minutes, additional information on its milliseconds becomes invaluable and its impact on the resulting duration would be too small to generate meaningful results and conclusions. Instead, it would only lengthen the time needed to process such big data. This issue becomes severe, especially with the sheer volume of our dataset. Hence, it would only be reasonable to remove these irrelevant values.

Explanation of Code

We apply df[‘column name’] on the start and end columns to obtain a better view of each of these individual columns.

|  |  |
| --- | --- |
| Cell | Code |
| 17 | # assess start column of DataFrame  df['start'] |
| 18 | # assess end column of DataFrame  df['end'] |

In order to remove the milliseconds information as represented by the values after the ‘.’, we apply the .split() method. The .map() function performs the lambda function object for each of the values and returns a list of values altered by the lambda function. In this case, the .split() method splits the string at the separator ‘.’ and returns part of the argument string before it. Values after the ‘.’ would be removed, while keeping the values before the ‘.’.

|  |  |
| --- | --- |
| Cell | Code |
| 19 | # remove milliseconds which is represented by values after '.'  df['start'] = df['start'].map(lambda x:x.split('.')[0])  df['end'] = df['end'].map(lambda x:x.split('.')[0]) |

The output is then verified using the df[‘column name’] on the start and end columns to ensure that the milliseconds have been removed appropriately.

|  |  |
| --- | --- |
| Cell | Code |
| 20 | # check output of start column  df['start'] |
| 21 | # check output of end column  df['end'] |

## 3rd Data Quality Issue

There are empty fields found under the “origin” and “destination” variable columns. The context of the data pertains to the use of shared mobility rides. Empty fields under these columns could imply a functional failure of the shared mobility device to capture the origin and destination codes. This is further supported by the fact that there were records under their corresponding start and end columns, which indicated the existence of a ride, however, they were not captured by the system in terms of its origin and destination information.

Removing these incomplete data would be a feasible option. Further analysis on these data have suggested that it may have occurred at random and carried no pattern as to why only these values are missing. By eliminating these incomplete data, it would help minimize the distortion of the validity of the results and conclusions. Since the original dataset is huge and where there is only a small data sample of 505 records containing the null values, omitting these records would not result in a substantial loss of information or biasness.

Explanation of Code

The null values under the ‘origin’ and ‘destination’ columns are denoted by NaN in Python. We first use the isnull().sum() method to check for the existence and count the amount of null values for each cell in the DataFrame. However, since our aim is to identify the number of missing values under the variable columns, the axis = 0 is added to obtain the sum of the column instead.

|  |  |
| --- | --- |
| Cell | Code |
| 22 | # count the number of null values in each column of DataFrame  df.isnull().sum(axis = 0) |

Through the output returned from the above code, we discovered that the origin and destination variable columns contain 504 null values each. The .dropna method is hence applied on the variable ‘origin’ and ‘destination’  to remove records with empty fields under the origin and destination columns as shown below.

|  |  |
| --- | --- |
| Cell | Code |
| 23 | # remove rows containing NaN under origin and destination columns using dropna. method  df = df.dropna(subset=['origin', 'destination']) |

Finally, we apply the same method used previously, isnull().sum() method, to verify for any null values. Since we have dropped the rows containing null values under the specified columns, the output of the code shown below has returned a ‘0’ for all columns to indicate that there are no null values in all columns.

|  |  |
| --- | --- |
| Cell | Code |
| 24 | # check for number of null values in each column of DataFrame  df.isnull().sum(axis = 0) |

# Question 1(d)

We created a user function that prints the hour by PM or AM and counts where the most commuters begin their journey by using defining method and if, else if, else statement.

|  |
| --- |
| Code |
| # Defining a method that will display the highest number of commuters start their journey and the time.  def HighestFrequency(df):        # Make a copy of start      start\_column = df['start'].copy()        # extracting the hour and converting it to int      start\_column = start\_column.map(lambda x:int(x.split(':')[0].split()[1]))        # By using mode it will return the highest number (most occuring number) of commuters start their journey      highest\_hour = start\_column.mode()[0]      # if the hour is less than 12      if highest\_hour < 12:          print("The Highest number of commuters that starts their jorney is",start\_column.value\_counts()[highest\_hour],"which starts at",start\_column.mode()[0],"AM")        # if the hour is equal to 12      elif highest\_hour == 12:          print("The Highest number of commuters that starts their jorney is",start\_column.value\_counts()[highest\_hour],"which starts at",start\_column.mode()[0],"PM")        # if the hour is greater than 12      else:          print("The Highest number of commuters that starts their jorney is",start\_column.value\_counts()[highest\_hour],"which starts at",start\_column.mode()[0]%12,"PM")  # Calling the method  HighestFrequency(df) |

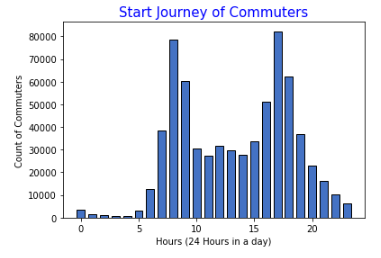
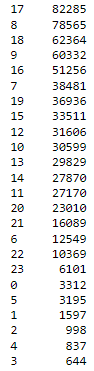
## Output Cell



# Question 1(e)

## Visual 1 - Number of commuters that start their journey at the certain hour of the day

We saw in the preceding question (d) that the highest number of commuters begin their journey around 5 p.m., thus we'd want to see whether there is another trend of highest commuter count by the hour during the 24 hour period.

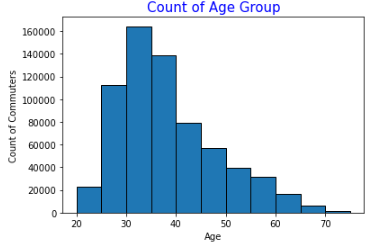
|  |  |
| --- | --- |
| Figure 1: Frequency of Start Journey of Commuters | Table 1: Count of Commuters |

According to Figure 1, the highest frequency for commuters who begin their travel in the morning is between 8 and 9 a.m., whereas the highest frequency for commuters who begin their journey in the evening is between 5 and 6 p.m. According to Table 1, most of the commuters begin their journey in the morning (78,565 and 60,332 travelers, respectively), while the largest frequency of commuters begin their journey in the evening (82,285 and 62,364 commuters, respectively).

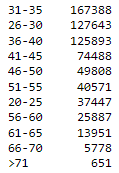
|  |  |
| --- | --- |
| Cell | Code |
| 1 | # Making a copy of the start column  start\_column\_2 = df['start'].copy()  # Checking and displaying the copy of start column  start\_column\_2 |
| 2 | # extracting the hour data  start\_column\_2 = start\_column\_2.map(lambda x:int(x.split(':')[0].split()[1]))  # Checking after splitting the hour  start\_column\_2 |
| 3 | # Count of Age  start\_count = start\_column\_2.value\_counts()  print(start\_count) |
| 4 | # Storing the hours 0-23 in x-axis  x = pd.Series(range(0, 24))  # Storing the number of commuters in y-axis  y = [start\_column\_2.value\_counts()[x] for x in range(24)]  # Plotting of bar graph to show the number of commuters start journey  plot = plt.bar(x,y, align='center', width = 0.7, color = '#4472C4', edgecolor ='black')  # Sets font1 as blue colour with size 15  font1 = {'color':'blue', 'size':15}  # Sets font2 as black colour with size 10  font2 = {'color':'black', 'size':10}  # Sets chart title using font1  plt.title('Start Journey of Commuters', fontdict = font1)  # Sets x-axis label  plt.xlabel("Hours (24 Hours in a day)", fontdict = font2)  # Sets y-axis label  plt.ylabel("Count of Commuters", fontdict = font2)  # Display the bar chart  plt.show() |

## Visual 2 - Count of Commuters By Age Group

We would want to depict the age group of commuters from the data set provided, as it is noticed that the age range varies from 20 to 71 years old. As a result, we would like to identify the age group that commute the most.



### Figure 2: Frequency of Commuters by Age Group



### Table 2: Age Group

Figure 2 shows that the distribution is skewed to the right, as demonstrated to be positively skewed. The distribution has a high number of occurrences in the younger age groups and a low number in the older age groups. All of the data displayed has a value greater than zero. It can also be seen that the majority of commuters are between the ages of 30 and 35.

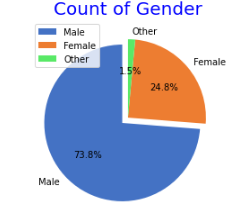
According to Table 2, the largest age group that travels amount to 167,388 people, with the majority of them being between the ages of 31 and 35. This data is consistent with the results observed in Figure 1. Commuters in this age group are often strongly associated with being in the workforce where work usually begins in the morning and ends in the evening. They may be in the phase of working towards financial stability and hence, may not have own a personal vehicle yet. As such, most of them may have opted for the shared mobility devices for transport, thereby suggesting the two peak periods identified in Figure 1.

|  |  |
| --- | --- |
| Cell | Code |
| 1 | # Assigning ages to age group using conditions  # Using Numpy Select to Set Values using Multiple Conditions  # Creating a list of conditions  conditions = [      (df['age'] >= 20) & (df['age'] <26), (df['age'] >= 26) & (df['age'] <31),      (df['age'] >= 31) & (df['age'] <36), (df['age'] >= 36) & (df['age'] <41),      (df['age'] >= 41) & (df['age'] <46), (df['age'] >= 46) & (df['age'] <51),      (df['age'] >= 51) & (df['age'] <56), (df['age'] >= 56) & (df['age'] <61),      (df['age'] >= 61) & (df['age'] <66), (df['age'] >= 66) & (df['age'] <71),      (df['age'] >= 71),  ]  # Creating corresponding values to fill  age\_values = ['20-25', '26-30', '31-35', '36-40',                '41-45', '46-50', '51-55', '56-60',                '61-65', '66-70',                '>71']  df['Age\_Group'] = np.select(conditions, age\_values)  df |
| 2 | # Table Count of Age Group  age\_group\_count = df['Age\_Group'].value\_counts()  print(age\_group\_count) |
| 3 | %matplotlib inline  # Plotting and displaying histogram  plt.hist(df['age'], bins=range(20,80,5), edgecolor ='black')  # Sets font1 as blue colour with size 15  font1 = {'color':'blue', 'size':15}  # Sets font2 as black colour with size 10  font2 = {'color':'black', 'size':10}  # Sets chart title  plt.title('Count of Age Group', fontdict = font1)  # Sets x-axis label  plt.xlabel("Age", fontdict = font2)  # Sets y-axis label  plt.ylabel("Count of Commuters", fontdict = font2) |

## 

## Visual 3 - Count of Gender

When we first study the data set, we see that there is a significant difference in gender. As a result, we'd want to show the data and observe whether there is a significant difference in commuters based on gender, therefore a pie chart would be a good choice as there is only 3 types of gender in the data set.

|  |  |
| --- | --- |
| Figure 3: Count of Commuters by Gender % | Table 3: Count of Commuters by Gender |

Figure 3 is a pie chart displaying the number of commuters by gender in the form of degrees. We used Python to calculate the proportion of each gender who travels. It also shows that more than 73.8% of commuters are male.

Table 3 shows that there are 493,845 males, 165,749 females, and 9911 other people. It demonstrates that the 73.8% given in Figure 3 computes to 493,845 males, who have the greatest number of commuters by gender of all three gender categories.

The significant difference among the gender proportion may be attributed to their characteristics and perception of safety. Males are risk-takers and hence, are willing to explore the shared mobility devices due to its speed and convenience. Females, on the other hand, are generally more risk-averse and tend to opt for safer alternatives. This is especially so during the day, where the number of cars and pedestrians on road and street respectively tend to be unusually high and crowded. Females generally perceive these bikes as a threat to their safety and others and may hence choose to take the public transport instead.

|  |  |
| --- | --- |
| Cell | Code |
| 1 | # Storing the gender types in x2  x2 = list(df['gender'].value\_counts().keys())  # Storing the number of commuters in y2  y2 = list(df['gender'].value\_counts())  # Sets the colours, font size with colour and explode size  colours = ['#4472C4', '#ED7D31', '#59E967']  font1 = {'color':'blue', 'size':20}  exp = [0.1,0,0]  # plotting of pie chart with the starting angle of 90 degrees with respect to the assigned colours, add in percentage and explode  plt.pie(y2, labels = x2, startangle = 90, colors = colours, autopct= '%2.1f%%', explode = exp)  # Sets the pie chart's title and legend  plt.title('Count of Gender', fontdict = font1)  plt.legend()  # Count of gender and displaying its value  values = df['gender'].value\_counts()  print(values) |

# References

[1] Peres, G. (2021, February 5). Pandas: print column name with missing values. Retrieved from stackoverflow: https://stackoverflow.com/questions/37366717/pandas-print-column-name-with-missing-values

[2] How to Convert Strings to Integers in Pandas DataFrame. (2021, June 4). Retrieved from Data to Fish: https://datatofish.com/string-to-integer-dataframe/

[3] Drop rows with all zeros in pandas data frame. (2016, August 5). Retrieved from stackoverflow: https://stackoverflow.com/questions/22649693/drop-rows-with-all-zeros-in-pandas-data-frame

[4] Zach. (2021, May 14). Pandas: How to Drop Rows that Contain a Specific Value. Retrieved from Statology: https://www.statology.org/pandas-drop-rows-with-value/

[5] Agarwal, M. (n.d.). Pythonic Data Cleaning With Pandas and NumPy. Retrieved from Real Python: https://realpython.com/python-data-cleaning-numpy-pandas/

[6] Set Pandas Conditional Column Based on Values of Another Column. (2021, August 9). Retrieved from datagy: https://datagy.io/pandas-conditional-column/

[7] Python Export to Excel. (2022, January 15). Retrieved from DelftStack: https://www.delftstack.com/howto/python/python-write-to-an-excel-spreadsheet/#:~:text=file%20in%20Python.-,Export%20Data%20to%20Excel%20With%20the%20DataFrame.,()%20function%20in%20Pandas%20DataFrame%20

[8] How to Replace Values in Pandas DataFrame. (2021, July 23). Retrieved from Data to Fish: https://datatofish.com/replace-values-pandas-dataframe/#:~:text=(3)%20Replace%20multiple%20values%20with,value'%2C...%5D

[9] pandas.DataFrame.mode. (2022). Retrieved from pandas: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mode.html

[10] Pandas – Count Missing Values in Each Column. (n.d.). Retrieved from Data Science Parichay: https://datascienceparichay.com/article/pandas-count-missing-values-in-each-column/

[11] Robinson, R. (2012, August 9). matplotlib: disregard outliers when plotting. Retrieved from stackoverflow: https://stackoverflow.com/questions/11882393/matplotlib-disregard-outliers-when-plotting