

**ANL252**

**Python for Data Analytics**

**Group-based Assignment**

**July 2022 Presentation**

|  |  |
| --- | --- |
| **Name** | **PI No.** |
| **Derence Ng Jun Jie** | **H1911247** |
| **Ho Qin Yi Amanda** | **E1780807** |
| **Muhammad Saifuddin Bin Muhammad Shahiddin** | **Q2210021** |
| **Shawnald Yeong Pheng Kiat** | **J2170823** |
| **Teo Takafumi** | **W1972267** |

**Tutorial Group: T04**

**Instructor’s Name: Chua Poh Chai (Cai Bao Cai)**



**Question 1 (a)**

import pandas as pd #alias for pandas

import numpy as np #alias for numpy

import matplotlib.pyplot as plt #alias for matplotlib

import datetime

#After running this, it prompts that there are a mix data types in columns 7 and 8

data1 = pd.read\_csv('GBA\_data.csv')

#Hence, here we will convert columns 7 and 8 which are "yob" and "age" to string.

data1 = pd.read\_csv ('GBA\_data.csv', dtype={ 'yob': 'str', 'age': 'str'})

data1

**Question 1 (b)**

#We will convert all ‘-’, ‘--’ and ‘?’ into blank cells to subsequently drop all blank cells.

data1.replace('-', np.nan, inplace = True)

data1.replace('--', np.nan, inplace = True)

data1.replace('?', np.nan, inplace = True)

data1.isnull().sum(axis = 0/1)

#As part of the data preparation, we have identified the variable columns with missing values as “origin”, “destination”, “type”, “yob”, “age”, and “gender”. The missing values are as described below:

#origin column contains 504 missing values

#destination column contains 504 missing values

#type column contains 10 missing values

#yob column contains 10 missing values

#age column contains 10 missing values

#gender column contains 3 missing values

#To deal with missing data, one should choose to either apply imputation or deletion/removal of the data.

#Since the missing data cannot be determined, we decide to remove the missing data from the data set.

#In particular, we will apply the listwise deletion method to remove the rows containing the missing data. Listwise deletion method is advantageous to large sample volume.

#To delete entire observations that have blanks using the Listwise deletion method.

data1.dropna(how='any', inplace= True)

#check to see if NaN values are replaced.

data1.isnull().sum(axis = 0)

#We choose to remove rows instead because it is illogical to delete columns as it will result in too little variable columns to compare.

**Question 1 (c)**

**1st Data Issue**

The first major issue identified from the dataset is that the date and time were both in the ‘start’ and ‘end’ column. In hindsight, this may pose a potential problem when we need to process the dataset later on. Hence, to treat the data, we decided to split the data under column ‘start’ to ‘start\_date’ and ‘start\_time’ as well as for ‘end’ to ‘end\_date’ and ‘end\_time’

#Splitting the ‘start’ and ‘end’ column into 'start\_date', 'start\_time', 'end\_date' and 'end\_time'.

data1[['start\_date','start\_time']]=data1.start.str.split(' ',expand=True)

data1[['end\_date','end\_time']]=data1.end.str.split(' ',expand=True)

#Subsequently we drop columns 'start' and 'end' since we no longer require them as we have added the split column as mentioned above.

data1.drop(columns=['start', 'end'], inplace = True)

data1

**2nd Data Issue**

The second major issue with the given dataset is that there are outliers present. The entire dataset revolved around the year 2022, however a small part of the data that was collected were in 2023, therefore it would constitute as outliers. We would want to focus on the dataset for the year 2022 to cater to a better data quality. Hence, we will remove the data that was collected in 2023 and only focus on the data that was in 2022.

#Here, we will locate the data that were collected in 2023 and drop it, such that we’ll only use data that were collected in 2022

#to locate data that was collected in 2023

data1.loc[(data1['start\_date'] >= '2023-01-01')

& (data1['start\_date'] < '2023-12-31')]

#to drop the data that was collected in 2023

data1.drop(axis = 0, index = [308000, 308001, 308002, 308003, 308004, 308005, 308006, 308007, 308008, 308009], inplace = True)

data1

**3rd Data Issue**

No aggregation of total time travelled for each individual, only given start time and end time. Consolidation of aggregate will provide a better narration and insight on the given dataset. Hence, in order to treat the dataset, we have to compute the duration of time of service used by the customer by subtracting the end time and the start time with the result in minutes.

#To change start time and end time to datetime to do arithmetic

#identify the type of data start time and end time is currently in

data1.info()

#change start time and end time to datetime

data1['start\_time'] = pd.to\_datetime(data1['start\_time'], format='%H:%M:%S.%f')

data1['end\_time'] = pd.to\_datetime(data1['end\_time'], format='%H:%M:%S.%f')

#to check if it has been changed

data1.info()

#subtract end time and start time to find the aggregate of total time travelled

data1['time travelled'] = (data1['end\_time'] - data1['start\_time'])

data1

**Question 1 (d)**

#For question 1d, we will re-run and re-read the original csv.

#This time round we will name it data2.

#Similarly, it prompts that there are a mix data types in columns 7 and 8.

data2= pd.read\_csv('GBA\_data.csv')

#Hence, here we will convert columns 7 and 8 which are "yob" and "age" to string.

data2= pd.read\_csv ('GBA\_data.csv', dtype={ 'yob': 'str', 'age': 'str'})

data2

def modeHour(): #creating def function

data2['start'] = pd.to\_datetime(data2['start']) #convert start column to datetime format

data2['start'].dt.strftime('%I %p') #converting datetime to 12hr format

data2['starttime'] = data2.start.dt.strftime('%I %p') #creating starttime as 12 hour format

start = data2['starttime'].mode() #highest frequency is mode therefore, finding mode

return start # return the mode value to function

modeHour()



**Question 1 (e)**

Figure 1

#Here we plot a bar chart for the number of consumers in the different types split by gender (Figure 1).

fig, ax = plt.subplots(figsize=(12, 8)) #size of chart

types = ['Concession', 'Regular', 'Ad-Hoc'] #Column value

xpos = np.arange(len(types))

bar\_width = 0.5

maleconces = data1[(data1['type'] == 'Concession') & (data1['gender'] == 'Male')].count()[0] #count of data for male and concession

malereg = data1[(data1['type'] == 'Regular') & (data1['gender'] == 'Male')].count()[0] #count of data for male and regular

maleadhoc = data1[(data1['type'] == 'Ad-Hoc') & (data1['gender'] == 'Male')].count()[0] #count of data for male and adhoc

femaleconces = data1[(data1['type'] == 'Concession') & (data1['gender'] == 'Female')].count()[0] #count of data for female and concession

femalereg = data1[(data1['type'] == 'Regular') & (data1['gender'] == 'Female')].count()[0] #count of data for female and regular

femaleadhoc = data1[(data1['type'] == 'Ad-Hoc') & (data1['gender'] == 'Female')].count()[0] #count of data for female and adhoc

male = (maleconces, malereg, maleadhoc) #Consolidate male values

female = (femaleconces, femalereg, femaleadhoc) #Consolidate female values

b1 = ax.bar(xpos, male,width=bar\_width, label='Male', color = '#2b8cbe') #male column

b2 = ax.bar(xpos + bar\_width, female,width=bar\_width, label='Female', color = '#c51b8a') # female column

ax.set\_xticks(xpos + bar\_width / 3) #position

ax.set\_xticklabels(types) # label for bar

ax.legend(loc = 'center right') # legend position

ax.spines['top'].set\_visible(False) #axis style

ax.spines['right'].set\_visible(False) #axis style

ax.spines['left'].set\_visible(False) #axis style

ax.spines['bottom'].set\_color('#DDDDDD') #axis style

ax.tick\_params(bottom=False, left=False) #axis style

ax.set\_axisbelow(True) #axis style

ax.yaxis.grid(True, color='#EEEEEE') #axis style

ax.xaxis.grid(False) #axis style

ax.set\_xlabel('Type', labelpad=20) #x axis label

ax.set\_ylabel('Number of Commuters', labelpad=20) #y axis label

ax.set\_title('Number of commuters by Gender against Customer Profile Type', pad=25) #chart title

fig.tight\_layout()

for bar in ax.patches:

bar\_value = bar.get\_height() #get bar value

text = f'{bar\_value:,}' #data point is bar value

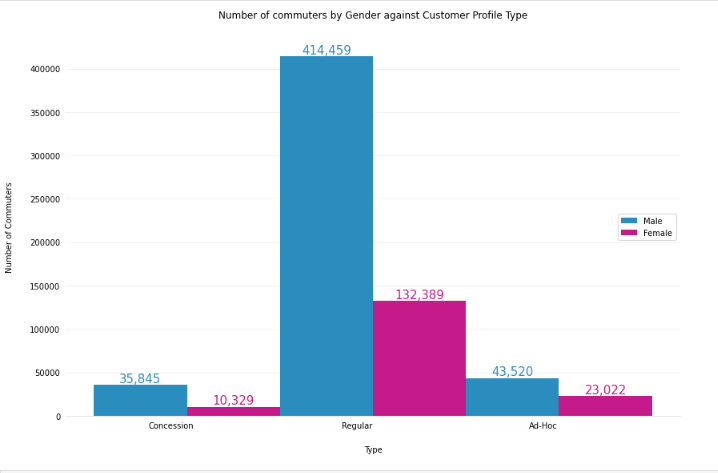
text\_x = bar.get\_x() + bar.get\_width() / 2 #position data value

text\_y = bar.get\_y() + bar\_value

bar\_color = bar.get\_facecolor() #default style

ax.text(text\_x, text\_y, text, ha='center', va='bottom', color=bar\_color,

size=15)

**

*Figure 1: Number of Commuters by Gender against Customer Profile Type*

Figure 1 is the output of the chart created using python as codes above. This chart is created to find out information on the majority of the commuters. By finding out this information, more marketing can be done to specific target groups. Assuming this is a car sharing application dataset, this bar chart will help the company to make decisions in ways to generate more users. The bar chart does not only show the type of commuters, Concession, Regular and Ad-Hoc but it is also separated by gender, Male and Female. The very noticeable insight from this chart is the number of commuters in the Regular type. Over 80% of the commuters fall under this commuter type. This shows that most of the commuters are regular users of the application. If concession is the ideal marketing type, the company could promote and market an incentive model catered for the regular commuters to encourage the switch to concession.

By splitting the value by gender, we are able to gain additional insights. We are able to see that the majority of users are also male. This can help in decision making on targeting the correct market. The company can consider to focus their marketing to target the male population as they are the majority or can also focus on promoting usage to the female population to increase the number of female commuters. A promo can be given to first time users to encourage them to commute this way more often.

Figure 2

#Here we wish to gain insights on the drivers in terms of age thus we will convert the age to int and then we extract the column and name it data2.

data1 = data1.astype({"age":"int"})

data3 = data1.age

data3

#Here we plot a histogram for the distribution of age (Figure 2).

Age = (data3)

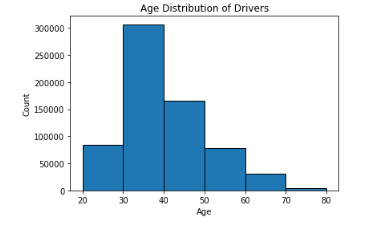
bins = (20, 30, 40, 50, 60, 70, 80)

plt.title('Age Distribution of Drivers')

plt.xlabel('Age')

plt.ylabel('Count')

plt.hist(Age, bins=bins, edgecolor='black')

****

*Figure 2: Age Distribution of Drivers*

Let's assume the organisation, which is a car rental service, wishes to determine the current age distribution of their service. As such, the data were cleaned and populated based on age group.

In Figure 2, a histogram was created with a bin width of 10 and range from 20 years old to 80 years old.

Based on Figure 2, the key takeaway is that there is a higher likelihood of drivers renting a car between 30 years old to 40 years old. As the histogram is skewed to the right, the frequency is higher on the lower age group as compared to the higher age group, with the mean being 30 to 40 years old. The organisation could use such data to determine the target group for advertisement.

The rationale for such data is largely due to the monetary limitation of individuals in their younger age. As individuals work longer into their life, the chances of them renting a car is lower, assuming individuals have higher buying power to afford their own car and this explains and correlates to the right skewed histogram. As such, the advertisement of the rental service should be targeted towards the younger age group.

Figure 3

data1['age'] = pd.to\_numeric(data1['age']) #converting age column from object to integer

agegroup = [] #creating new column to group age

for age in data1['age']:

if age < 30:

group = ['Age below 30'] # grouping age below 30

elif age >= 30 and age < 40:

group = ['Age between 30 to 39'] #grouping age between 30 to 39

elif age >= 40 and age < 50:

group = ['Age between 40 to 49'] #grouping age between 40 to 49

elif age >= 50 and age < 60:

group = ['Age between 50 to 59'] #grouping age between 50 to 59

elif age >= 60:

group = ['Age 60 and above'] #grouping age 60 and above

agegroup = agegroup + group

data1['agegroup'] = agegroup

sub30 = len(data1[(data1['subscriber'] == 'Yes') & (data1['agegroup'] == 'Age below 30')]) #count subscriber below 30

sub3040 = len(data1[(data1['subscriber'] == 'Yes') & (data1['agegroup'] == 'Age between 30 to 39')]) #count subscriber between 30 to 39

sub4050 = len(data1[(data1['subscriber'] == 'Yes') & (data1['agegroup'] == 'Age between 40 to 49')]) #count subscriber between 40 to 49

sub5060 = len(data1[(data1['subscriber'] == 'Yes') & (data1['agegroup'] == 'Age between 50 to 59')]) #count subscriber between 50 to 59

sub60 = len(data1[(data1['subscriber'] == 'Yes') & (data1['agegroup'] == 'Age 60 and above')]) #count subscriber 60 and above

nosub30 = len(data1[(data1['subscriber'] == 'No') & (data1['agegroup'] == 'Age below 30')]) #count non subscriber below 30

nosub3040 = len(data1[(data1['subscriber'] == 'No') & (data1['agegroup'] == 'Age between 30 to 39')]) #count non subscriber between 30 to 39

nosub4050 = len(data1[(data1['subscriber'] == 'No') & (data1['agegroup'] == 'Age between 40 to 49')]) #count non subscriber between 40 to 49

nosub5060 = len(data1[(data1['subscriber'] == 'No') & (data1['agegroup'] == 'Age between 50 to 59')]) #count non subscriber between 50 to 59

nosub60 = len(data1[(data1['subscriber'] == 'No') & (data1['agegroup'] == 'Age 60 and above')]) #count non subscriber 60 and above

raw = pd.DataFrame([['Age below 30', sub30, nosub30], ['Age between 30 to 39', sub3040, nosub3040], ['Age between 40 to 49', sub4050, nosub4050],

['Age between 50 to 59', sub5060, nosub5060], ['Age 60 and above', sub60, nosub60]],

columns=['Age Group', 'Yes', 'No']) #create data frame for graph values

raw.plot.barh(stacked=True, title='Subscribers by Age Group', color=("orange", "cyan"), x= 'Age Group') #plot the graph

plt.show()

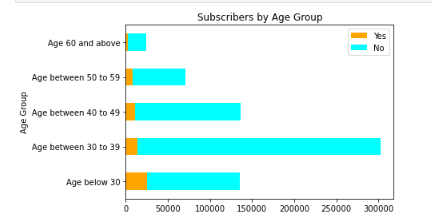


Figure 3 is a horizontal stacked bar which shows the number of subscribers by age group. Using the column age from the dataset, a code has been made to group the age by different ranges. This is to narrow down the age group where commuters use the application.

From the visualisation we are able to gain insight that the majority of commuters are in the age group “Age between 30 to 39”. This gives us information on the commuters demographic. However, there are clear subscriber and non-subscriber differences where the number of subscribers is very little as compared to the non-subscribing commuters. In return, the company might want to look into giving incentive to new subscribers or lower any subscription fee in an effort to attract more subscribers and thus respectfully gain loyal customers for long term benefits.

**References**

Dixon, M. (2020, March 3) *The best way to handle missing data.* Retrieved from:

<https://seleritysas.com/blog/2020/03/03/the-best-way-to-handle-missing-data/>

Wu, K. Y. (2022). *ANL252 Python for data analytics (study guide).* Singapore University of

Social Sciences.