**ANL252** 

**Python for Data Analytics**

**Group-based Assignment July 2022 Presentation**

|  |  |
| --- | --- |
| **Name** | **PI Number** |
| **Chloe Tan Ying Yin** | **W2082681** |
| **Tricia Ang Li Ying** | **J2081605** |
| **Koo Chi Hang** | **K2081853** |
| **Timothy Yap Wei Yu** | **K2111660** |
| **Reta Rajakumar** | **M2170763** |

**Tutorial Group: T09 Group 4**

**Instructor’s Name: Munish Kumar**

**Submission Date: 28 August 2022**

# Table of Contents

[**Question 1(a)**](#_hfokkdl0djtt) **2**

[**Question 1(b)**](#_lanmds9ia0lh) **3**

[**Question 1(c)**](#_u46ttj2ngi0m) **6**

[**Question 1(d)**](#_b3zac65z1lc) **9**

[**Question 1(e)**](#_26vx1f6vifz) **10**

[**References**](#_p9sg2bpiasc) **20**

# 

# Question 1(a)

*#To import the necessary libraries that are required for the code to run*

*import pandas as pd*

*import numpy as np*

*import matplotlib.pyplot as plt*

*#the strings in the 'start' and 'end' columns are parsed as DateTime objects*

*#the missing values of ‘-’, ‘--’, and ‘?’ are explicitly declared and parsed as NaN values*

*PATH = r"/Users/twly/Downloads/GBA\_data.csv" #replace r"..." path with own file path*

GBA\_data *= pd.read\_csv(PATH,parse\_dates=[2,3] , na\_values=['-','--','?'])*

# 

# Question 1(b)

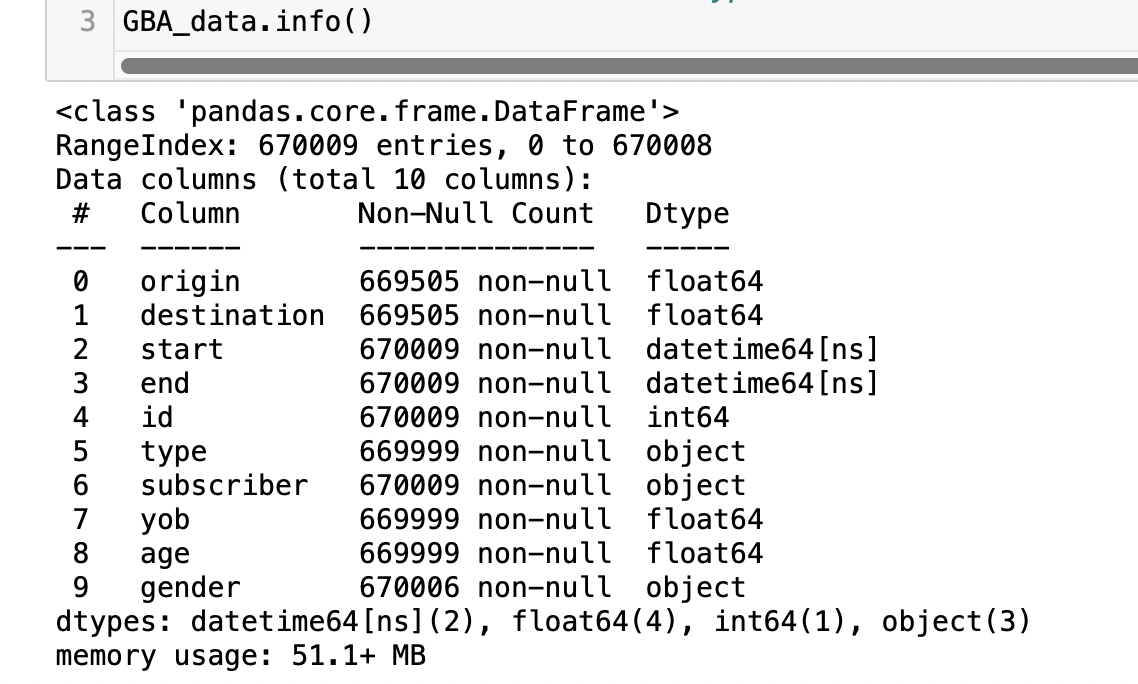
By using df.info, we can look at the information, data type, and shape for the data frame. There are more than 600,000 entries in this dataset. We then use df.isnull().sum(axis = 0) to locate the missing values in dataframe.

Code:

GBA\_data.info()

Output:

*Figure 1: Output for GBA\_data.info()*



Code:

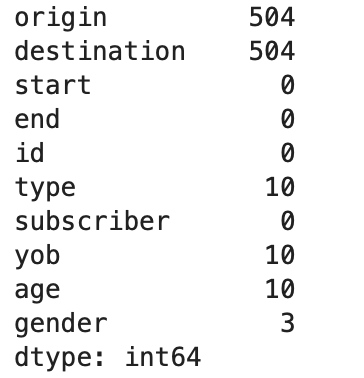
*#To locate missing data in the dataframe*

*#When axis = 0, it adds up the values in a column*

*GBA\_data.isnull().sum(axis = 0)*

Output:

*Figure 2: Locating missing data in dataframe*



Findings:

The ‘origin’, ‘destination’, ‘type’, ‘yob’, ‘age’ and ‘gender’ columns 504, 504, 10, 10, 10 and 3 missing values respectively. This adds up to a total sum of 1041 missing values in the data set. There are equal number of missing values in origin and destination columns. Similarly for type, year of birth and age columns.

**Data Preparation**

There are 2 main ways to treat missing data.

**1. Delete the entire observations of missing data**

The .dropna() function will allow us to remove the rows that contains NULL values. This will help to improve data quality and provide more accurate data for interpretation, analysis and decision making in an organisation (Stedman, n.d.).

Code:

*#The columns with any NULL values are dropped on the current DataFrame*

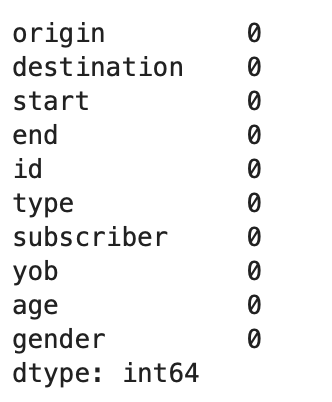
*GBA\_data.dropna(axis=0, how="any", inplace=True)*

*#to verify that the missing values have indeed been removed*

*GBA\_data.isnull().sum(axis = 0)*

Output:

*Figure 3: Verifying missing values has been removed in dataframe*



**2. Replace certain observations of missing data**

The .fillna() function will allow us to replace all missing values with a specified value that is indicated in the parameter. We can also specify a specific column where a set of missing values should be replaced by the fillna() function.

For example, we can replace the missing values in the gender column to “others” to prevent any loss of data relating to the identified rows.

Code:

*GBA\_data["gender"].fillna(value = "others")*

We can also choose to **ignore observations of missing data**. However, more often than not, this will affect the accuracy of data quality and provide faulty information which may lead to flawed business decisions and misguided strategies for organisations (Stedman, n.d.)

**Rationale for using method 1 to delete rows with missing values**

In this dataset, we will choose to remove the rows that contains NULL values with .dropna() function. This is because there are over 600,000 entries and excluding the total 1041 missing values is not significant to affect data visualisation.

# Question 1(c)

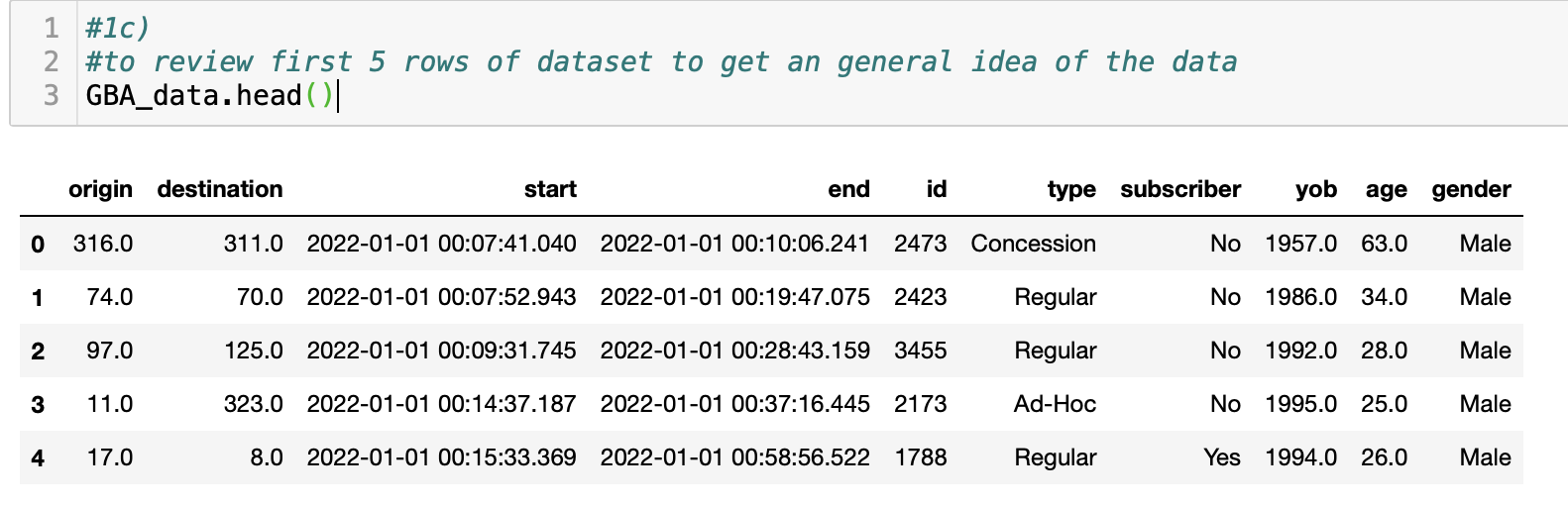
Use df.head() to review first 5 rows of dataset to get an general idea of the data.

Code:

GBA\_data.head()

Output:

*Figure 4: Overview of first 5 rows in dataframe*



**First data quality issue: start and end time format**

The start and end columns are expressed in yyyy-mm-dd hh:mm:ss:milliseconds. There is no need for milliseconds as such time precision is not valuable for our data visualisation.

Code:

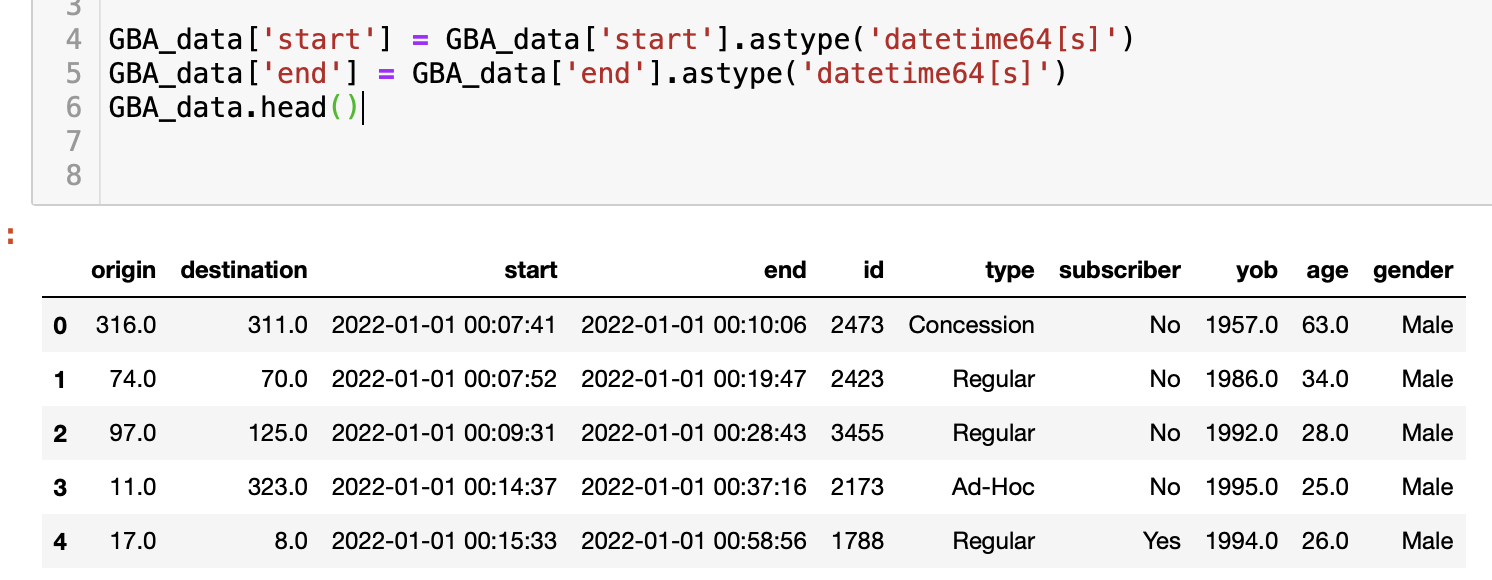
GBA\_data['start'] = GBA\_data['start'].astype('datetime64[s]')

GBA\_data['end'] = GBA\_data['end'].astype('datetime64[s]')

GBA\_data.head()

Output:

*Figure 5: Amend time format*

****

**Second data quality issue: memory usage**

Memory usage should be optimized so that our programs can run faster.

Categorical data type is useful in saving memory usage, as compared to object data type. To reduce memory usage, we convert 'object' and 'int64' type to 'category'. The year of birth column is also not necessary as we already have an age column. Hence we should remove the column. Memory usage has been reduced from 51.1 mb to 37.7 mb.

Code:

GBA\_data['type']= GBA\_data['type'].astype('category')

GBA\_data['subscriber']= GBA\_data['subscriber'].astype('category')

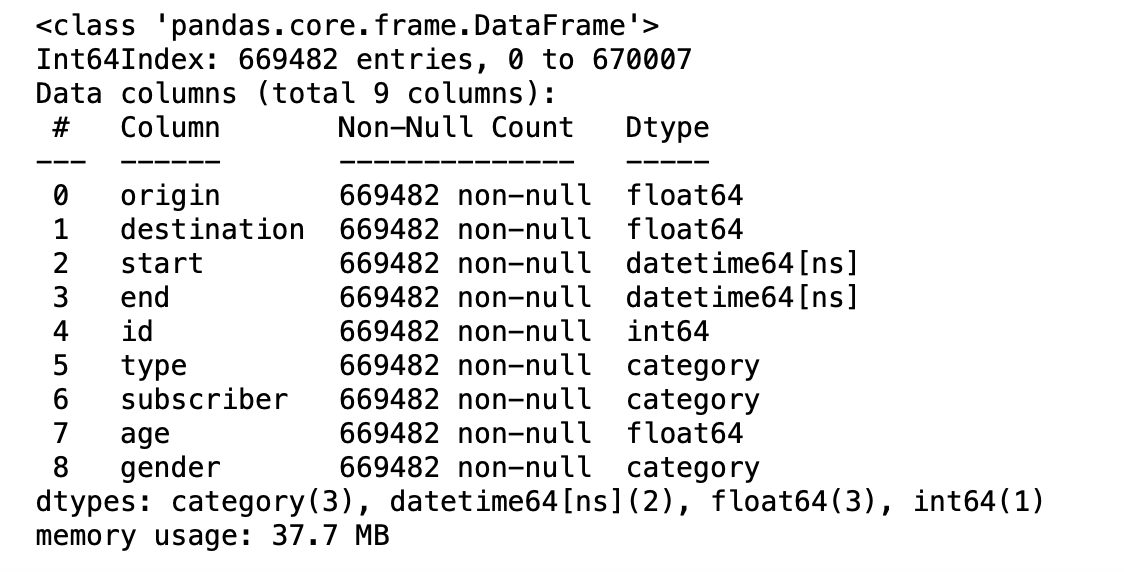
GBA\_data['gender']= GBA\_data['gender'].astype('category')

del GBA\_data['yob']

GBA\_data.info()

Output:

*Figure 6: Reduced memory usage*



**Third data quality issue: data type**

Origin, destination, and age is shown as a float data type. It is unnecessary to use decimals for origin, destination, and age, we should remove decimals so that data analysis and plotting will be easier to understand. We should change the data type to integer.

Code:

GBA\_data['origin']= GBA\_data['origin'].astype('int64')

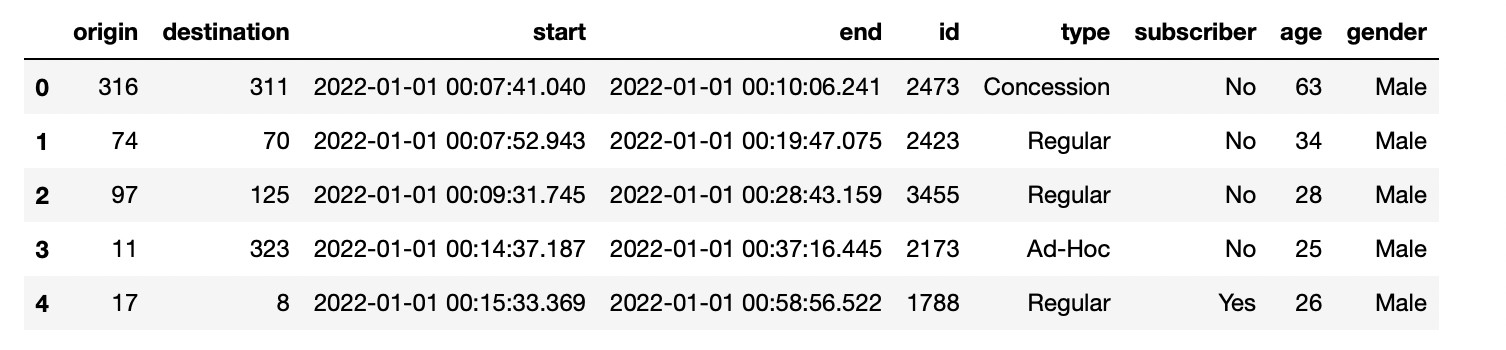
GBA\_data['destination']= GBA\_data['destination'].astype('int64')

GBA\_data['age']= GBA\_data['age'].astype('int64')

GBA\_data.head()

Output:

*Figure 7: Amended data type from float to integer for origin, destination and age.*



# Question 1(d)

The code for the user-defined function is appended below:

Code:

*#a copy of clean\_df is created for the purpose of this question*

df\_1d = GBA\_data.copy()

*#from the start column, the hour of travel is extracted in a 24-hour clock format in a new column, i.e. ‘start\_hour’*

df\_1d['start\_hour'] = df\_1d['start'].dt.hour

*#this parses the values in the start\_hour column as values expressed in the 12-hour clock format*

df\_1d['start\_time'] = df\_1d['start\_hour'].replace([0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23],['12am','1am','2am','3am','4am','5am','6am','7am','8am','9am','10am','11am','12pm','1pm','2pm','3pm','4pm','5pm','6pm','7pm','8pm','9pm','10pm','11pm'])

*#a function is defined to derive the mode value in the 'start hour' column*

*#the output is printed in a formatted string for the sake of readability*

def most\_freq\_hour():

mode\_hour = df\_1d['start\_time'].mode()

print(f'The highest number of commuters start their journey at {mode\_hour.iloc[0]}')

most\_freq\_hour()

Output:

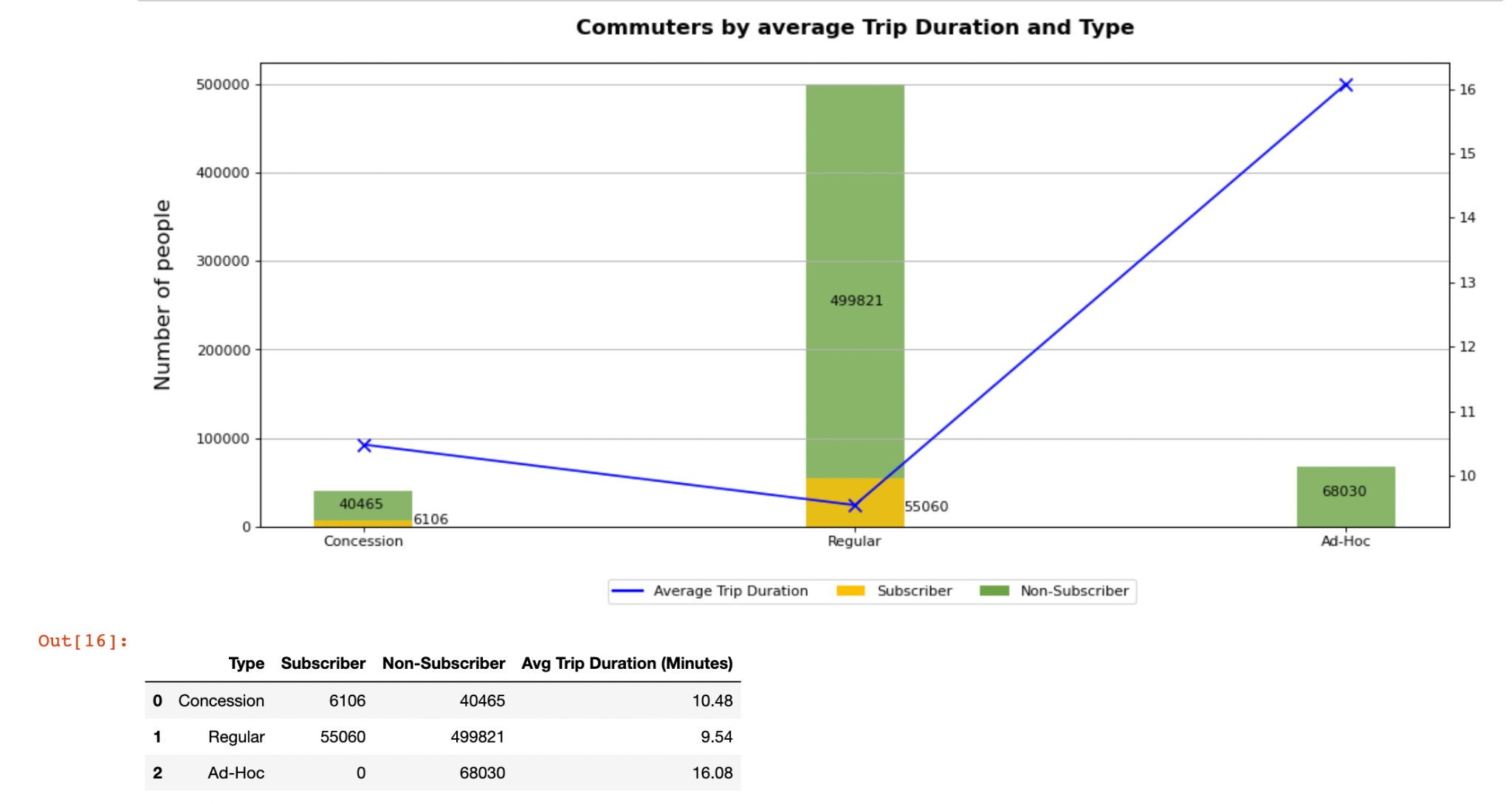
*Figure 8: Output for user-defined function*



# Question 1(e)

**Chart 1**

*Figure 9: [Chart 1] Commuters by Average Trip Duration and Type*



**Insight 1: Although regular commuters are the majority of total consumers, they have the lowest average trip duration.**

The dual axis chart showcases the breakdown of commuters by average trip duration and commuter type. Majority of the regular, ad-hoc, and concession commuters are made up of non-subscribers, with the largest majority of non-subscribers also being regular commuters (499,821 commuters). Furthermore, none of the Ad-Hoc commuters are subscribers.

Ad-hoc commuters spend the most average time on their commute, having the longest average trip duration (16.08 mins), followed by concession commuters (10.48 mins) and regular commuters (9.54 mins).

Code:

from matplotlib.patches import Patch

from matplotlib.lines import Line2D

*# define plot settings*

COL = "trip\_duration\_min"

TITLE = "Commuters by average Trip Duration and Type"

Y\_LABEL = "Number of people"

*# category = df\_1d["type"].unique()*

*# Create basic dictionary for values to be stored*

subs\_dict = {

"type" : df\_1d["type"].unique(),

"type\_axis": [1,2,3],

"Yes" :[],

"No" :[]

}

*# Calculate values for bar plots*

for choice in df\_1d["subscriber"].unique():

df\_choice = df\_1d[df\_1d["subscriber"]==choice]

subs\_dict[str(choice)] = [len(df\_choice[df\_choice["type"]==T]) for T in df\_1d["type"].unique() ]

*# Calculate average time duration for each type (this coding style might be a bit advance)*

avg\_trip\_dur = [round( df\_1d[df\_1d["type"]==T]["trip\_duration\_min"].mean() , 2 ) for T in df\_1d["type"].unique() ]

*# create figure*

ax = plt.figure(figsize=(15, 6), dpi=80)

*# plot bars*

plt1 = plt.bar(subs\_dict["type"], subs\_dict["No"],alpha=0.8 ,label="Non-Sub",width = 0.2,color=color\_dict["nsub"])

plt2 = plt.bar(subs\_dict["type"], subs\_dict["Yes"],alpha=0.8,label="Sub",width = 0.2,color=color\_dict["sub"])

*# plot display settings for bar plots*

plt.ylabel(Y\_LABEL,labelpad=15,fontsize=15)

plt.grid(True,which="Major",axis='y')

*# Display values of each bar and set placement coordinates*

for i,value in enumerate(subs\_dict["No"]):

plt.text(i-0.05,int(round(value/2,0)),str(value))

for i,value in enumerate(subs\_dict["Yes"]):

if i ==0: plt.text(i+0.1,int(round(value/2,0))-1000,str(value))

elif i ==1: plt.text(i+0.1,int(round(value/2,0))-10000,str(value))

else: continue

*# Duplicate/mirror X axis and plot line and scatter points*

ax2 = plt.twinx()

plt.plot(subs\_dict["type"],avg\_trip\_dur,c="#0000FF")

plt.scatter(subs\_dict["type"],avg\_trip\_dur,marker='x',s=80,c="#0000FF")

*# Plot settings*

plt.title(label=TITLE,pad=20,fontsize=15,weight='bold')

*# define legends and place it at the bottom*

legend\_elements = [

Line2D([0], [0], color="#0000FF", lw=2, label='Average Trip Duration'),

Patch(facecolor=color\_dict["sub"], edgecolor=None,label='Subscriber'),

Patch(facecolor=color\_dict["nsub"], edgecolor=None,label='Non-Subscriber')]

ax.legend(handles=legend\_elements, bbox\_to\_anchor=(0.7, 0.05),ncol=3)

plt.show()

*# prepare data for data table*

df\_dict = {

"Type": subs\_dict["type"],

"Subscriber": subs\_dict["Yes"],

"Non-Subscriber": subs\_dict["No"],

"Avg Trip Duration (Minutes)": avg\_trip\_dur,

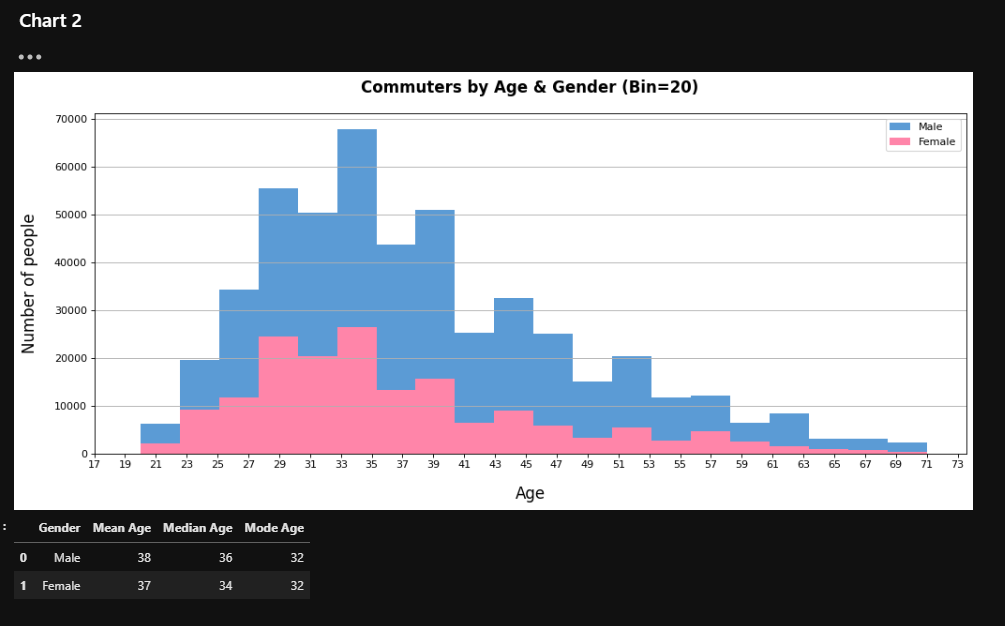
}

*# make data table into data frame and display it*

pd.DataFrame(df\_dict)

**Chart 2**

*Figure 10: [Chart 2] Commuters by Age & Gender*



**Insight 2: Majority of commuters are male, in the age range of 23-41**

The histogram showcases the number of commuters by binned age ranges (2 years) segmented by gender. There is a larger proportion of male commuters compared to female, with majority of commuters falling in the 23-41 age range. These commuters are likely working adults who are making their daily commute to and fro from work. There is a low number of senior commuters (>59 years old).

Code:

*# define plot settings*

BINS = 20

COL = "age"

TITLE = "Commuters by Age & Gender (Bin=20)"

Y\_LABEL = "Number of people"

X\_LABEL = "Age"

*# Data splits*

df\_interested = df\_1d[df\_1d["gender"].isin(["Male","Female"])]

m\_df = df\_interested[df\_interested["gender"]=="Male"]

f\_df = df\_interested[df\_interested["gender"]=="Female"]

*# create figure and plot histograms*

plt.figure(figsize=(15, 6), dpi=80)

plt.hist(m\_df[COL],bins = BINS,label="Male",color=color\_dict["Male"])

plt.hist(f\_df[COL],bins = BINS,label="Female",color=color\_dict["Female"])

*# Plots settings*

plt.title(label=TITLE,pad=20,fontsize=15,weight='bold')

plt.xlabel(X\_LABEL,labelpad=15,fontsize=15)

plt.ylabel(Y\_LABEL,labelpad=15,fontsize=15)

plt.xticks([i for i in range(int(df\_1d[COL].min())-3,int(df\_1d[COL].max()+3),2)])

plt.legend()

plt.grid(True,which="Major",axis='y')

plt.show()

*# prepare data for data table*

df\_dict = {

"Gender":df\_interested["gender"].unique(),

"Mean Age":[int(round(m\_df[COL].mean(),0)), int(round(f\_df[COL].mean(),0))],

"Median Age":[int(m\_df[COL].median().item()), int(f\_df[COL].median().item())],

"Mode Age":[int(m\_df[COL].mode().item()), int(f\_df[COL].mode().item())]

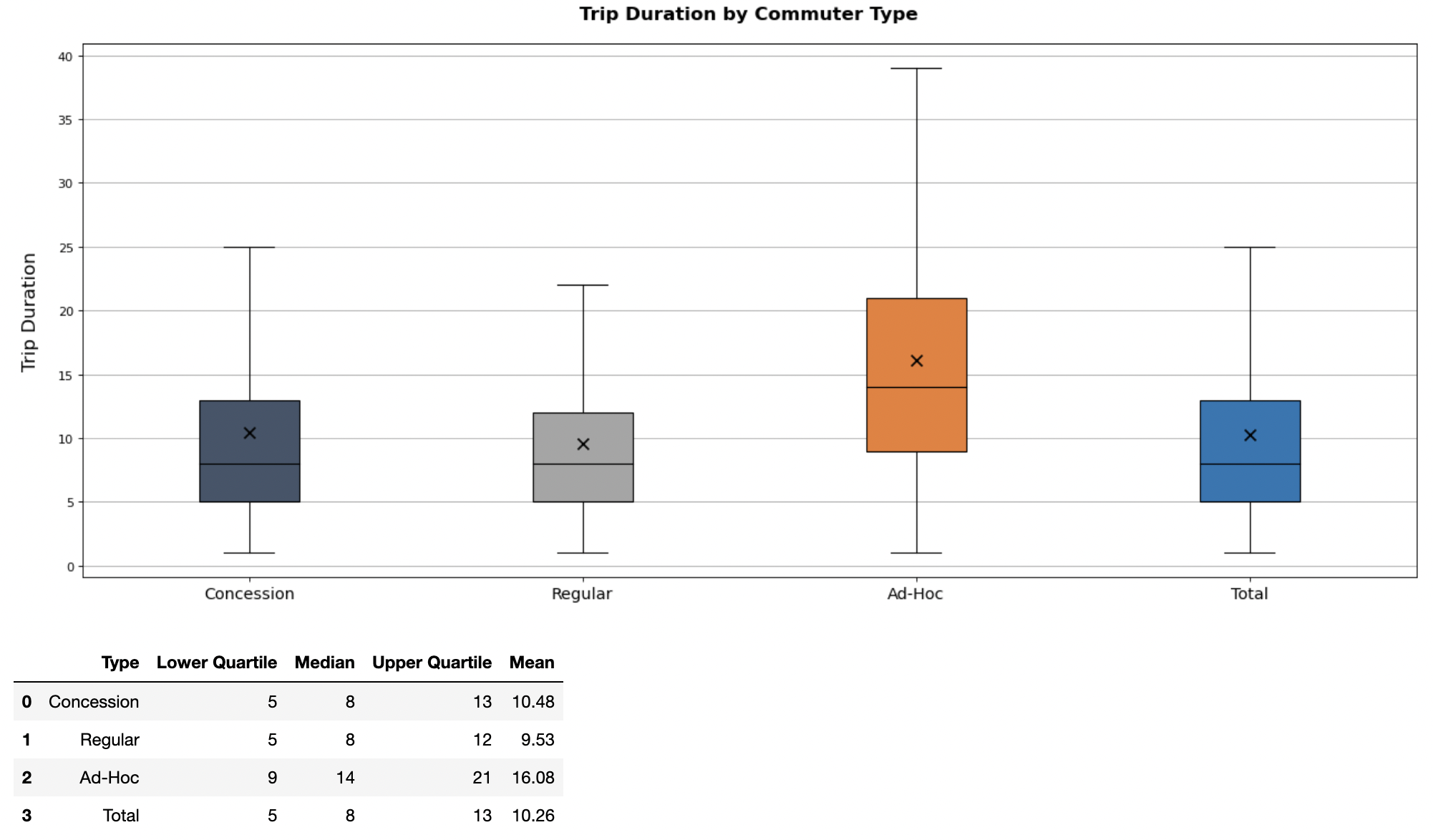
}

*# make into dataframe and display data table*

pd.DataFrame(df\_dict)

**Chart 3**

*Figure 11: [Chart 3] Trip Duration by Customer Type*



**Insight 3: Ad-Hoc commuters have the highest and largest range in the duration of their trips.**

The box plot chart depicts the commuters’ trip duration segmented by customer type with outliers removed. Ad-hoc consumers have the largest dispersion in the duration of trip timings, with a interquartile range of approximately 9-21 mins. The distribution of the trip duration of Regular and Concession commuters are similar, with Concession commuters having a slightly larger interquartile range (5-13 mins).

Ad-Hoc commuters (16.08 mins) have significantly higher mean trip duration in comparison to the total trip duration (10.26 mins), whilst Regular (9.53 mins) and Concession (10.48 mins) commuters fall within similar means.

All commuter types are are slightly positively skewed. The Ad-Hoc commuters has the larget positive skew, meaning that which shows that the Ad-Hoc commuters travelling longer distances are more varied.

Overall, the Ad-Hoc commuters take longer trips as compared to the other types of commuters. Furthermore, the average duration range of all commuters is from 5-13 mins.

Code:

from numpy import percentile

*# define data*

COL = "trip\_duration\_min"

TITLE = "Trip Duration by Commuter Type"

Y\_LABEL = "Trip Duration"

*# Data splits*

LIMIT\_DURATION = 200

df\_interested = df\_1d[df\_1d["trip\_duration\_min"]<=LIMIT\_DURATION]

*# initialise vairables*

data = []

mean\_dat = []

x\_axis = list(df\_interested["type"].unique()) + ["Total"]

*# data preparations for boxplots*

for T in df\_interested["type"].unique():

data.append(df\_interested[df\_interested["type"]==T]["trip\_duration\_min"])

mean\_dat.append(round(df\_interested[df\_interested["type"]==T]["trip\_duration\_min"].mean(),2))

data.append(df\_interested["trip\_duration\_min"])

mean\_dat.append(round(df\_interested["trip\_duration\_min"].mean(),2))

*# data plottings*

ax = plt.figure(figsize=(15, 6), dpi=80).add\_axes([0, 0, 1, 1])

bp = ax.boxplot(data,patch\_artist=True,whis=1.5,showfliers=False,widths=0.3)

ax.scatter( x = [1,2,3,4], y = mean\_dat,

color = 'black',zorder=3,marker='x',s=80)

*# Boxplot settings*

for i,T in enumerate(x\_axis):

if str(T) != "Total": bp["boxes"][i].set\_facecolor( color\_dict[str(T)] )

bp["medians"][i].set\_color('black')

*# Chart settings*

plt.title(label=TITLE,pad=20,fontsize=15,weight='bold')

plt.ylabel(Y\_LABEL,labelpad=15,fontsize=15)

plt.xticks([1,2,3,4],labels=x\_axis,fontsize=13)

plt.grid(True,which="Major",axis='y')

plt.show()

df\_dict = {

"Type":x\_axis,

"Lower Quartile":[],

"Median":[],

"Upper Quartile":[],

"Mean":[]

}

*# Calculation of data tables (mean,upper/lower quartile, median)*

for T in df\_interested["type"].unique():

df\_t = df\_interested[df\_interested["type"]==T]

quartiles = percentile(df\_t["trip\_duration\_min"], [25, 50, 75])

df\_dict["Upper Quartile"] = df\_dict["Upper Quartile"] + [int(quartiles[2])]

df\_dict["Median"] = df\_dict["Median"] + [int(df\_t["trip\_duration\_min"].median())]

df\_dict["Lower Quartile"] = df\_dict["Lower Quartile"] + [int(quartiles[0])]

df\_dict["Mean"] = df\_dict["Mean"] + [round(df\_t["trip\_duration\_min"].mean(),2)]

*# Calculate Overall/Total mean,upper/lower quartile, median*

quartiles = percentile(df\_interested["trip\_duration\_min"], [25, 50, 75])

df\_dict["Upper Quartile"] = df\_dict["Upper Quartile"] + [int(quartiles[2])]

df\_dict["Median"] = df\_dict["Median"] + [int(df\_interested["trip\_duration\_min"].median())]

df\_dict["Lower Quartile"] = df\_dict["Lower Quartile"] + [int(quartiles[0])]

df\_dict["Mean"] = df\_dict["Mean"] + [round(df\_interested["trip\_duration\_min"].mean(),2)]

*# make into display data table*

pd.DataFrame(df\_dict)

**Chart 3**

from numpy import percentile

*# define data*

COL = "trip\_duration\_min"

TITLE = "Trip Duration by Customer type"

Y\_LABEL = "Trip Duration"

*# Data splits*

LIMIT\_DURATION = 200

df\_interested = df\_1d[df\_1d["trip\_duration\_min"]<=LIMIT\_DURATION]

*# initialise vairables*

data = []

mean\_dat = []

x\_axis = list(df\_interested["type"].unique()) + ["Total"]

*# data preparations for boxplots*

for T in df\_interested["type"].unique():

data.append(df\_interested[df\_interested["type"]==T]["trip\_duration\_min"])

mean\_dat.append(round(df\_interested[df\_interested["type"]==T]["trip\_duration\_min"].mean(),2))

data.append(df\_interested["trip\_duration\_min"])

mean\_dat.append(round(df\_interested["trip\_duration\_min"].mean(),2))

*# data plottings*

ax = plt.figure(figsize=(15, 6), dpi=80).add\_axes([0, 0, 1, 1])

bp = ax.boxplot(data,patch\_artist=True,whis=1.5,showfliers=False,widths=0.3)

ax.scatter( x = [1,2,3,4], y = mean\_dat,

color = 'black',zorder=3,marker='x',s=80)

*# Boxplot settings*

for i,T in enumerate(x\_axis):

if str(T) != "Total": bp["boxes"][i].set\_facecolor( color\_dict[str(T)] )

bp["medians"][i].set\_color('black')

*# Chart settings*

plt.title(label=TITLE,pad=20,fontsize=15,weight='bold')

plt.ylabel(Y\_LABEL,labelpad=15,fontsize=15)

plt.xticks([1,2,3,4],labels=x\_axis,fontsize=13)

plt.grid(True,which="Major",axis='y')

plt.show()

df\_dict = {

"Type":x\_axis,

"Lower Quartile":[],

"Median":[],

"Upper Quartile":[],

"Mean":[]

}

*# Calculation of data tables (mean,upper/lower quartile, median)*

for T in df\_interested["type"].unique():

df\_t = df\_interested[df\_interested["type"]==T]

quartiles = percentile(df\_t["trip\_duration\_min"], [25, 50, 75])

df\_dict["Upper Quartile"] = df\_dict["Upper Quartile"] + [int(quartiles[2])]

df\_dict["Median"] = df\_dict["Median"] + [int(df\_t["trip\_duration\_min"].median())]

df\_dict["Lower Quartile"] = df\_dict["Lower Quartile"] + [int(quartiles[0])]

df\_dict["Mean"] = df\_dict["Mean"] + [round(df\_t["trip\_duration\_min"].mean(),2)]

*# Calculate Overall/Total mean,upper/lower quartile, median*

quartiles = percentile(df\_interested["trip\_duration\_min"], [25, 50, 75])

df\_dict["Upper Quartile"] = df\_dict["Upper Quartile"] + [int(quartiles[2])]

df\_dict["Median"] = df\_dict["Median"] + [int(df\_interested["trip\_duration\_min"].median())]

df\_dict["Lower Quartile"] = df\_dict["Lower Quartile"] + [int(quartiles[0])]

df\_dict["Mean"] = df\_dict["Mean"] + [round(df\_interested["trip\_duration\_min"].mean(),2)]

*# make into display data table*

pd.DataFrame(df\_dict)

# References

Craig, S. (n.d.). *Data Cleansing*. https://www.techtarget.com/searchdatamanagement/definition/data-scrubbing