

**Singapore University of Social Sciences**

**Group-Based Assignment (GBA)**

**ANL252: Python for Data Analytics**

Dr Munish Kumar

August 28, 2022

We, members of group 8, do hereby declare that we each contributed to this assignment and that we collectively agree to a shared grade.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | PI Number | Contribution | Signature |
| Tan Wee Min, Andre (Chen Weimin) | E2070932 | I did questions 1a to 1e. | Andre |
| Kynan Tan | Z2181984 | I did questions 1a to 1e. | Kynan |
| Lee Pei Shi | Y2210991 | I did questions 1a to 1e. | Belinda |
| Tng Ting Xu | K2181584 | I did questions 1a to 1e. | Ting Xu |
| Tan Wenying, Audrey (Chen Wenying) | K2170805 | I did questions 1a to 1e. | Audrey |

**Question 1a)**

**Code in text:**

**Libraries**

#import pandas

import pandas as pd

#import matplotlib

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

#import seaborn

import seaborn as sns

#import ast

import ast

#import datetime

from datetime import date, time, datetime

#import numpy

import numpy as np

#import missingno

import missingno as msno

**Import Excel**

# reading data frame from the csv file and replacing symbols as null values

df = pd.read\_csv('GBA\_Data.csv', engine='python').replace(to\_replace =["?" , "-" , "--"], value = np.nan)

**Question 1b)**

The variables of origin, destination, type, yob, age, and gender have null values. However, the percentage of null values per column is low as it falls between 0.1% to 7%. Yob and age are correlated, while origin and destination are correlated as well.

The null values were all skewed to a certain extend and the mean values cannot be used to replace these null values. Null values in gender were replaced with the mode which is Male. Since, origin and destination have no correlation with other variables and as it is not as important in data visualisation, their null values were replaced by the mode, as the mode is not affected by extreme high or low values.

There were some null ages in concession (ages: 55 to 65) and regular types (ages: 20-71). There were no null ages in ad-hoc types (ages: 20 to 71). As the age influences the type purchased, the overall median age could not be used to replace these null values. The null ages were replaced with the corresponding median age for each type. These null ages had corresponding null yobs. The null yobs were replaced by the new corresponding ages. For example, when type is concession, the null age was replaced by the median concession age which was 58 years old. Then the null yob for concession was replaced by the corresponding yob for this age of 58, which was 1962.

The null types were replaced with regular as none of the ages for the null types were concession types (ages: 55-65) and since there are more regular records, the mode of regular was used to replace these null values. This action of replacing of all values instead of dropping them was to achieve consistency across data cleaning.

**Code in text:**

**Data Preparation**

display(df)

#get sum of null for columns with at least one null value

df[df.columns[df.isnull().any()]].isnull().sum()

#Find out the percentage of missing values in each column with null values in the df

percent\_missing = df[df.columns[df.isnull().any()]].isnull().sum() \* 100 / len(df)

df1 = pd.DataFrame({'percent\_missing': percent\_missing})

display(df1)

# Visualize the correlation between the number of

# missing values in different columns as a heatmap

msno.heatmap(df) #dark blue shows more correlation

#check the skewness of the columns. Use median instead of mean to replace null if it is skewed.

print(df.skew())

**Gender, Origin and Destination with null values are replaced by the mode**

#change gender with null to the mode

for column in ['gender', 'origin', 'destination']:

df[column].fillna(df[column].mode()[0], inplace=True)

**Replace null in each type by the Median age for each type**

#check the data types and change data type of columns as per required

df.dtypes

#change age to float first then integer

df['age']=df['age'].astype('float').astype('Int64')

#group by type to find the median, mean, min and max of ages.

result = round(df.groupby('type').agg({'age': ['median','mean', 'min', 'max']}),0)

print("Mean, min, and max values of age grouped by type")

print(result)

#calculate the number of null values in age per type

result3= df.groupby('type').agg({'age': lambda x: x.isnull().sum()})

print(result3)

#since we will be filling concession with median age 58, we need to find the corresponding year to fill yob

year1= df.loc[df['age'] == 58, 'yob'].iloc[0]

#since we will be filling regular with median age 35, we need to find the corresponding year to fill yob

year2= df.loc[df['age'] == 35, 'yob'].iloc[0]

#When age is null and type is concession, change the age to the median 58

df['age'] = np.where((df['age'].isnull()) & (df['type'] == 'Concession'), 58, df['age'])

#When age is null and type is regular, change the age to the median 35

df['age'] = np.where((df['age'].isnull()) & (df['type'] == 'Regular'), 35, df['age'])

#When yob is null and type is concession, change the yob to the corresponding 1962

df['yob'] = np.where((df['yob'].isnull()) & (df['type'] == 'Concession'), year1, df['yob'])

#When yob is null and type is regular, change the yob to the corresponding 1985

df['yob'] = np.where((df['yob'].isnull()) & (df['type'] == 'Regular'), year2, df['yob'])

**Replace types with null values with Regular**

#creating ranges for the ages

bins = [19, 30, 40, 50, 60, 70, 120]

labels = ['19-29', '30-39', '40-49', '50-59', '60-69', '70+']

df['agerange'] = pd.cut(df.age, bins, labels = labels,include\_lowest = True)

display(df)

#group by type and agerange to find agerange for each type

result5 = df.groupby(['type','agerange'], dropna=False).size()

print(result5)

#group by type to find subscriber for each type.

result6 = df.groupby(['type','subscriber'], dropna=False).size()

print(result6)

#find the percentage of subscribers under each type

result6= result6 / result6.groupby(level=0).sum()

print(result6)

#As the agerange for null values is not seniors, the type will not be concession

#It will be either Ad-Hoc or Regular

#As there are more regular Records than Ad-Hoc for both subscribers, to replace null values to Regular

#When type is null, change the type to Regular

df['type'] = np.where((df['type'].isnull()), 'Regular', df['type'])

#Checking the number of nulls left

df[df.columns[df.isnull().any()]].isnull().sum()

**Question 1c)**

To achieve consistency in data cleaning, replacements are done instead of a combination of dropping and replacing.

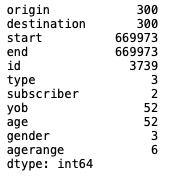
**First data quality issue and subsequent implementation**

Firstly, for data preparation, nunique() method was used to check the number of distinct elements in each column as seen in Figure 1.

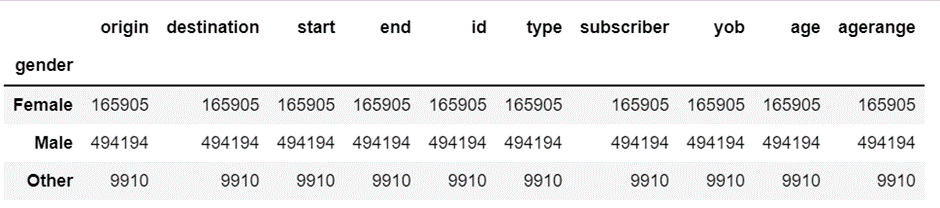
One of the data quality issues with this data was there were three values under the gender column.

It should be limited to have only two gender ‘Male’ or ‘Female’. However, the dataset includes an entry for 'Other' as seen in Figure 2. Since the mode for gender is ‘Male’, ‘Other’ was replaced with ‘Male’. A check was done in Figure 3 to ensure that ‘Other’ was no longer present.

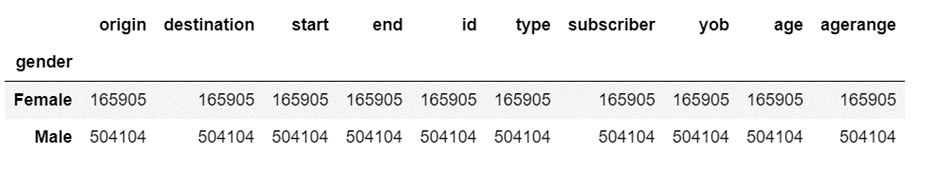
**Figure 1**



**Figure 2**



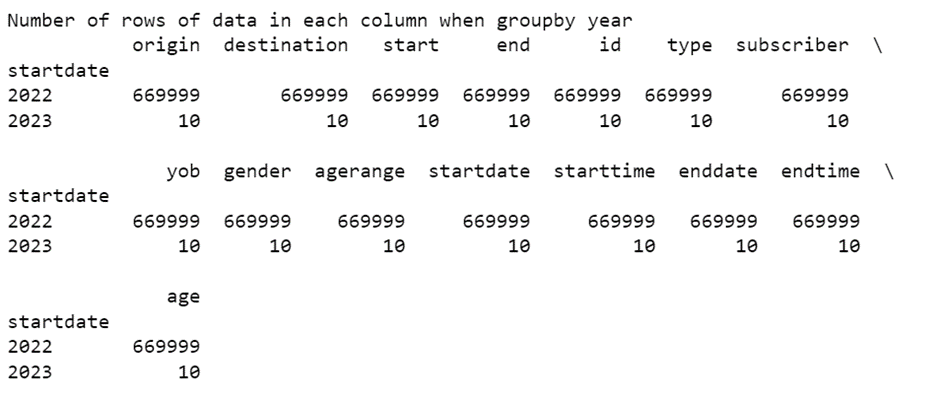
**Figure 3**



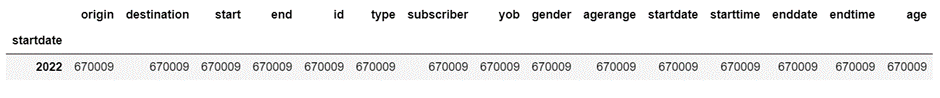
**Second data quality issue and subsequent implementation**

The second data issue was that there were data from 2023 that may have been entered incorrectly. This can be seen in Figure 4. Since the current year is ‘2022’, it is incorrect for the dataset to consist of data collected from ‘2023’ and hence, we have chosen to replace the value of ‘2023’ with the year ‘2022’. As seen in Figure 5, a check was done to ensure that ‘2023’ was no longer present.

**Figure 4**



**Figure 5**

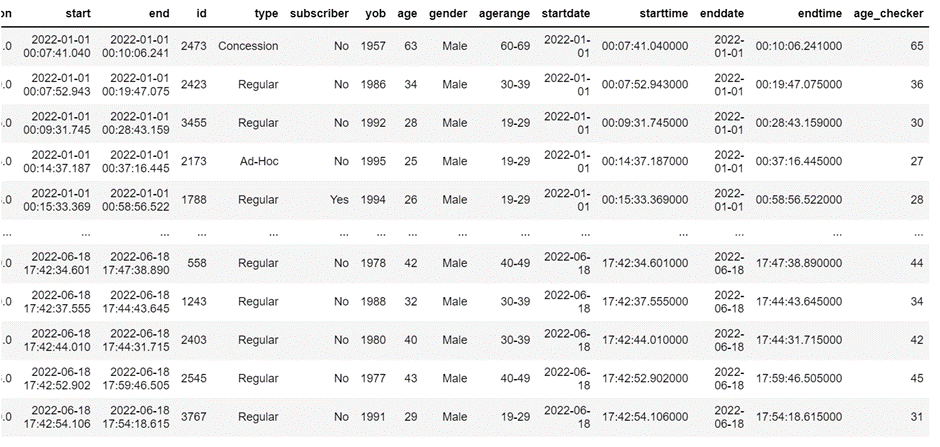


**Third data quality issue and subsequent implementation**

The third quality issue suggests that the age was calculated based on the year 2020 because the original data calculation was not based on the year 2022. This issue was rectified by recalculating through subtraction of yob from 2022 to create the new column *age\_checker*.

The new column *age\_checker* can be seen in Figure 6. The old column age was removed using the drop() method, and the new column was renamed from *age\_checker* to *age* using the rename() method.

**Figure 6**



**Code in text:**

#check the number of unique values for each column

df.nunique()

**Data quality issue 1: There should be only 2 types of gender**

#Gender has 3 unique values, check the categories

df.gender.unique()

**Suggest and implement ways to treat issue 1**

#Find the number of rows with others

result3 = df.groupby(df.gender).count()

result3

#Replace Other with Male

df['gender'] = df['gender'].str.replace('Other','Male')

#check that replacement was done

result4 = df.groupby(df.gender).count()

Result4

**Data quality issue 2: Data from 2023 was captured even though it is in the future**

#change to datetime format

df['startdate'] = pd.to\_datetime(df['start']).dt.date

df['starttime'] = pd.to\_datetime(df['start']).dt.time

df['enddate'] = pd.to\_datetime(df['end']).dt.date

df['endtime'] = pd.to\_datetime(df['end']).dt.time

df.dtypes

#change to datetime format

df['startdate'] = pd.to\_datetime(df['startdate'], errors='coerce')

#group by startdate to find the number of rows with 2023

result2 = df.groupby(df.startdate.dt.year).count()

print("Number of rows of data in each column when groupby year")

print(result2)

**Suggest and implement ways to treat issue 2**

#replace year 2023 with 2022

df['startdate']= df['startdate'].mask(df['startdate'].dt.year ==2023,

df['startdate']+pd.offsets.DateOffset(year=2022))

#check results

result3 = df.groupby(df.startdate.dt.year).count()

result3

**Data quality issue 3: The age is based off year 2020 instead of 2022 even though data is from 2022**

#create a new column with the correct age based off 2022

now = 2022

df['age\_checker'] = now - df['yob'].astype('int')

df

#check if the data in age matches age\_checker

df['age'].equals(df['age\_checker'])

#Check the number of non matches per row between age and age\_checker

s = (df.age.eq(df.age\_checker)

.value\_counts()

.rename({True:'match', False: 'no match'}))

s

**Suggest and implement ways to treat issue 3**

#Drop column age

df.drop(['age'], axis=1, inplace=True)

#Rename age\_checker as age

df.rename(columns={'age\_checker': 'age'}, inplace=True)

**Question 1d)**

**Code in text:**

**Converting the datetime**

#convert endtime and starttime to format of hours, mins, sec

df['endtime'] = df['endtime'].apply(lambda x: time.strftime(x, '%H:%M:%S'))

df['starttime'] = df['starttime'].apply(lambda x: time.strftime(x, '%H:%M:%S'))

display(df)

#Express starttime in the 12-hour clock

df['starttime12'] = pd.to\_datetime(df['starttime']).dt.strftime('%I %p')

df['starttime12']

#dataframe groupby start time in 12 hour clock and number of commuters per hour

df\_sorted= df.groupby(['starttime12'], as\_index=False)['type'].size()

#user defined function to groupby starttime and count the number of rows matching the starttime

def sort():

print("The time with the maximum number of commuters:")

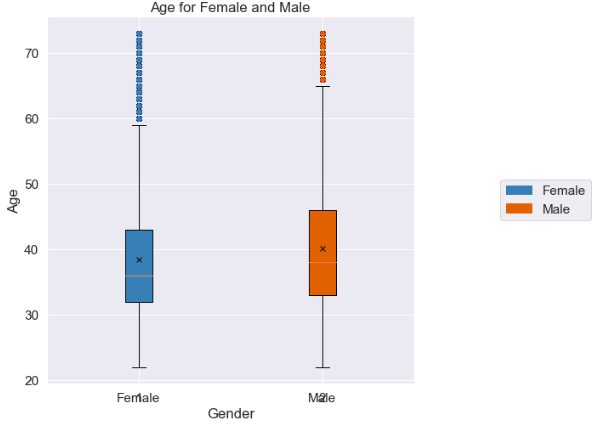
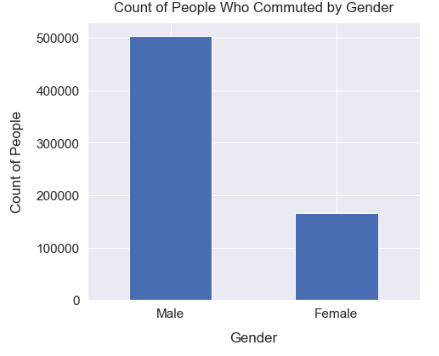
print(df\_sorted.loc[df\_sorted['size'].idxmax()].values[0])

sort()

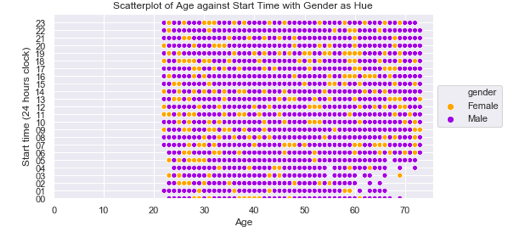
**Question 1e)**

**Insight 1 on Gender:**

**Figure 7 Figure 8**



**Figure 9**



Based on our observations from the bar chart, boxplot and scatterplot, Male users make up a significantly larger proportion of total users of the shared mobility service as approximately 75% of users are Male. Additionally, 50% of Male are aged between mid 30s to late 40s.

Conversely, Female users make up a significantly smaller proportion of total users at approximately 25%. Additionally, 50% of Female are aged between early 30s to early 40s.

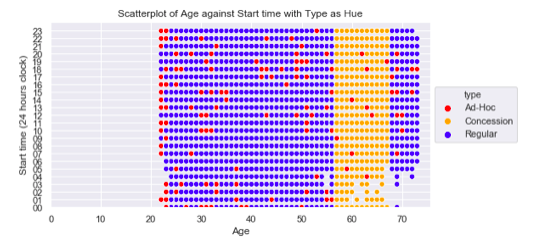
From the boxplot, it can be interpreted that the minimum age for both Male and Female is approximately early 20s. However, the maximum age for Male is higher than the maximum age for Female, around 65 and late 50s respectively.

Additionally, Female users are observed to be utilizing more shared mobility services between 7am to 6pm and lesser between 8pm to 5am while Male commuters are observed to have generally travelled equally across all times.

These observations are consistent with a normal work force travelling to work in the morning and returning home in the evening. More Male users utilizing shared mobility services past midnight could be attributed to manual labor / shift jobs which are typically male dominated.

**Insight 2 on Types:**

**Figure 10**



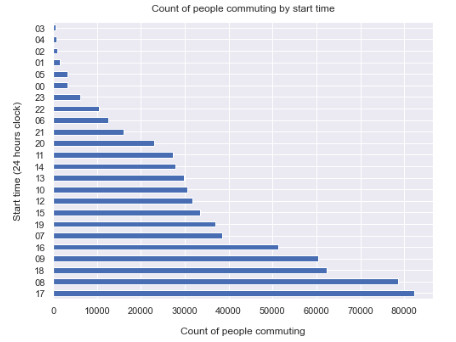
Based off volume of users, the profile types are mainly dominated by Regular users, followed by Concession users, and lastly Ad-hoc users. The majority of Regular users were mostly between the age range of 25 to 55 and 68 to 75. There is a rather equal spread of Regular users across the age range of 25 to late 50s travelling around the clock.

Based on our observations, all Concession type users were seniors aged between 55 to 65. This could indicate that concession-based fares are only available for seniors aged between 55 to 65. However, there were no observations of seniors aged above 65 with Concession.

Users who were categorised under ad-hoc type were scattered across all age groups. Senior users aged 69 and above do not travel between 1 am to 4 am, and were either regular or ad-hoc types.

**Insight 3 on Travelling Times:**

**Figure 11**



From the bar chart, the timings that recorded the highest volume of commuters are between 8 am to 9am and 4 pm to 6 pm. This could be attributed to workers utilizing shared mobility services for travelling to and from work and are considered peak hours. Additionally, the workforce is more punctual in ending their workday, as compared to starting their workday. This is evident from the spreading of people commuting across a wider start time in themorning.

Moreover, there was a significant decrease in the number of users beyond 10 pm. This could also be attributed to more users at home after work and the closure of shared mobility services such as trains and public buses. However, the volume of users commuting started increasing from 6 am when public transport reopens for the day.

**Code in text:**

**Data Visualization**

#Plot a bar chart

sns.set(font\_scale=1.4)

df['gender'].value\_counts().plot(kind='bar', figsize=(7, 6), rot=0)

plt.xlabel("Gender", labelpad=14)

plt.ylabel("Count of People", labelpad=14)

plt.title("Count of People Who Commuted by Gender", y=1.02);

#Extract all the rows with Gender = F or female

contain\_values = df[df['gender'].str.contains('Female')]

#For these rows with Gender =F, filter out the Salary

contain\_values = contain\_values.filter(['age'])

#Extract all the rows with Gender = M or Male

contain\_values2 = df[df['gender'].str.contains('Male')]

#For these rows with Gender =M, filter out the Salary

contain\_values2 = contain\_values2.filter(['age'])

#Combining data of salaries from M and F into numpy array

data = np.array([contain\_values, contain\_values2], dtype=object)

#Set the size of the plot

fig = plt.figure(figsize =(6, 6))

#Creating axes instance

ax = fig.add\_axes([0, 0, 1, 1])

#Creating plot

bp = ax.boxplot(data)

#X and Y Labels

plt.xlabel('Gender')

plt.ylabel('Age')

#Adding title

plt.title("Age for Female and Male")

#Adding ticks

plt.xticks([1, 2], ['Female', 'Male'])

#Plot the boxplot

box = plt.boxplot(data, #array to be plotted

patch\_artist=True, #fill with color

flierprops={'markeredgecolor': 'None'}, #no marker edger for outliers

showmeans=True, #show the mean

meanprops={"marker":"x","markerfacecolor":"black", "markeredgecolor":"black"}) #set the marker type and colour of the mean

#Fill the outliers with two colours

cols = ['steelblue', 'chocolate']

for f, fc in zip(box['fliers'], cols):

f.set\_markerfacecolor(fc)

#Fill the boxplot with two colours

colors = ['steelblue', 'chocolate']

for patch, color in zip(box['boxes'], colors):

patch.set\_facecolor(color)

#set legend

orange\_patch=mpatches.Patch(color='steelblue',label="Female") #Set colour for Females in legend

blue\_patch=mpatches.Patch(color='chocolate',label="Male") #Set colour for Males in legend

plt.legend(loc='center right', bbox\_to\_anchor=(1.5, 0.5), ncol=1, handles=[orange\_patch, blue\_patch])

#show plot

plt.show()

#convert to datetime format in 24hours

df['starttime'] = pd.to\_datetime(df.loc[:, 'starttime']).dt.strftime('%H')

#sort by 24 hour format

df.sort\_values(by=['starttime'], inplace=True, ascending=True)

display(df)

#Plot scatterplot with Seaborn

#To set the colours per Unit

colour\_dict = dict({'Male':'blueviolet',

'Female':'orange'})

#Set the order in the legend

orders=['Female','Male']

#To set the side of the scatterplot

sns.set(rc={"figure.figsize":(8, 4)})

#To display the scatterplot

g =sns.scatterplot(x="age", y="starttime",

hue="gender",

data=df,

palette=colour\_dict,

hue\_order=orders)

#Rename axis

plt.xlabel('Age')

plt.ylabel('Start time (24 hours clock)')

#To start the x and y axis at point 0

plt.xlim(0)

plt.ylim(0)

#To set the title

plt.title('Scatterplot of Age against Start Time with Gender as Hue')

#To set the legend location of scatterplot

g.legend(loc='center left', bbox\_to\_anchor=(1, 0.5), ncol=1)

#Plot scatterplot with Seaborn

#To set the colours per Unit

colour\_dict = dict({'Ad-Hoc':'red',

'Concession':'orange',

'Regular': 'blue'})

#Set the order in the legend

orders=['Ad-Hoc','Concession','Regular']

#To set the side of the scatterplot

sns.set(rc={"figure.figsize":(8, 4)})

#To display the scatterplot

g =sns.scatterplot(x="age", y="starttime",

hue="type",

data=df,

palette=colour\_dict,

hue\_order=orders)

#Rename axis

plt.xlabel('Age')

plt.ylabel('Start time (24 hours clock)')

#To start the x and y axis at point 0

plt.xlim(0)

plt.ylim(0)

#To set the title

plt.title('Scatterplot of Age against Start time with Type as Hue')

#To set the legend location of scatterplot

g.legend(loc='center left', bbox\_to\_anchor=(1, 0.5), ncol=1)

#Count of people commuting at a certain hour

df['starttime'].value\_counts().plot(kind='barh', figsize=(8, 6))

plt.xlabel("Count of people commuting", labelpad=14)

plt.ylabel("Start time (24 hours clock)", labelpad=14)

plt.title("Count of people commuting by start time", y=1.02);

**The End**