

**ANL252 Python for Data Analytics**

**July 2022 Presentation**

**GBA**

|  |  |
| --- | --- |
| TG: | 09 |
| Group: | 9 |
| Student 1 Name: | Shawn Ng Zhen Xiang (Q2172585) |
| Student 2 Name: | Pay Jiarre Lyn (Y1981896) |
| Student 3 Name: | Lynn Tan |
| Submission Date: | 28 Aug 2022 |

**Q1(a)** The vast commuter journey dataset is rich with interesting insights that can be explored by python. However, the probability of missing values grows as the size of the dataset increases and this affects data exploration and modeling. In this case, values ‘-’, ‘--’, and ‘?’ are seen as missing values and we will be using python to source them out below. We will first begin by understanding the data.

**# Importing Libraries**

import pandas as pd

import numpy as np

import datetime

from datetime import datetime

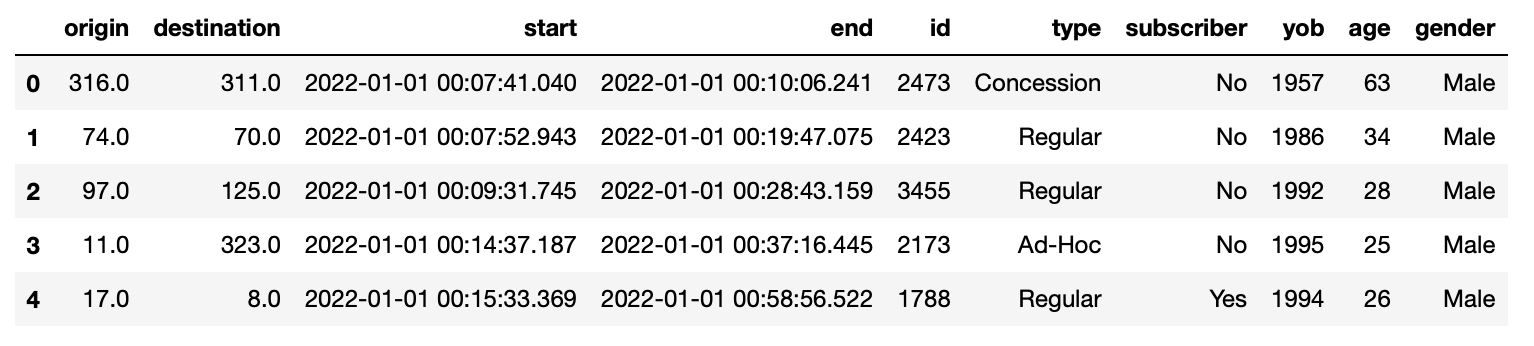
import math

**# Reading File to Pandas DataFrame**

df = pd.read\_csv(r'GBA\_Data.csv', low\_memory = False)

**# Show the first 5 rows of the dataframe**

df.head()



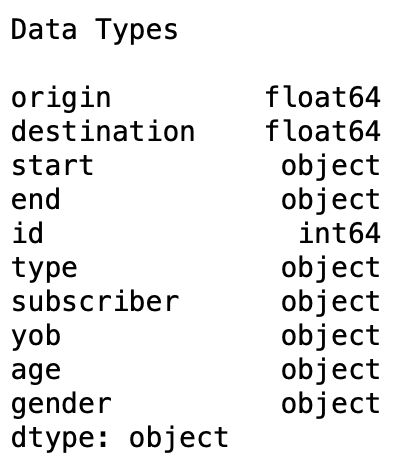
**# Show the number of rows and columns of the data frame**

print(f'Data Shape \n\n{df.shape}')



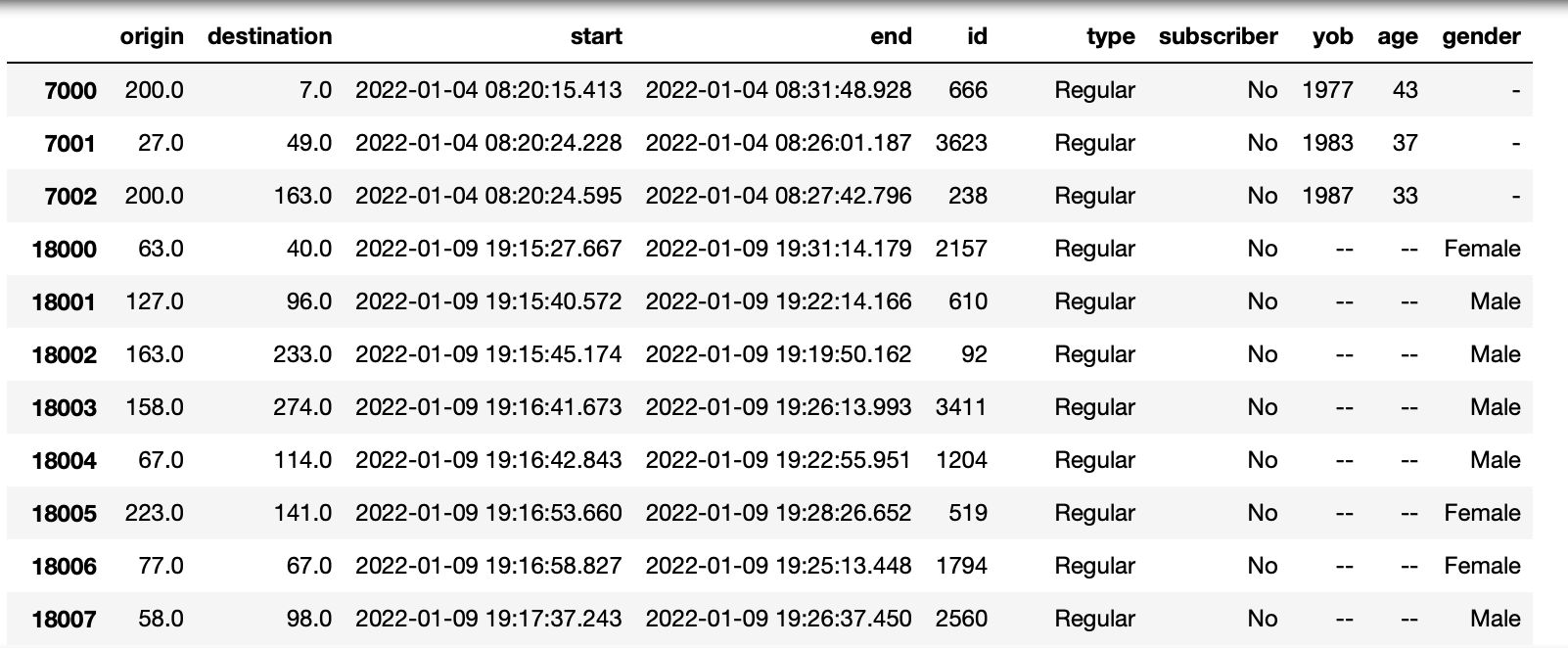
**# Check data types**

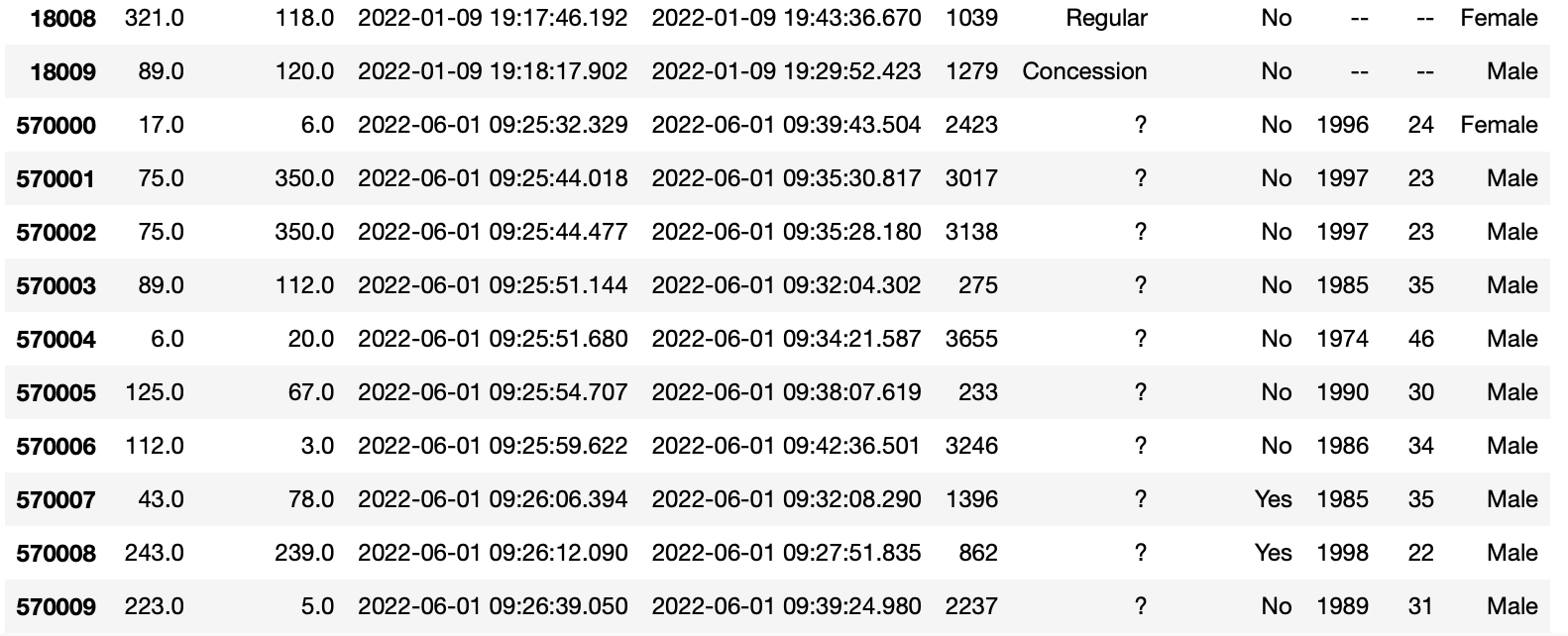
print(f'Data Types \n\n{df.dtypes}')



**# Check condition for values = '?', '--', '-'**

df[df.isin(['?', '--', '-']).any(axis=1)]

****



From the above, we can see that the missing values exist in columns under type, yob, age and gender. Some instances of such missing values may be due to users neglecting to fill out the fields or lost data during manual transfer of data from an older database.

**Q1(b)** Understanding the idea of missing values is crucial for efficient data management. If the user does not manage missing data correctly, inaccurate data analysis will have a significant impact on the modeling phase, hence impacting the findings. Other than the missing values identified in part (a), more missing values were also seen from the data set.

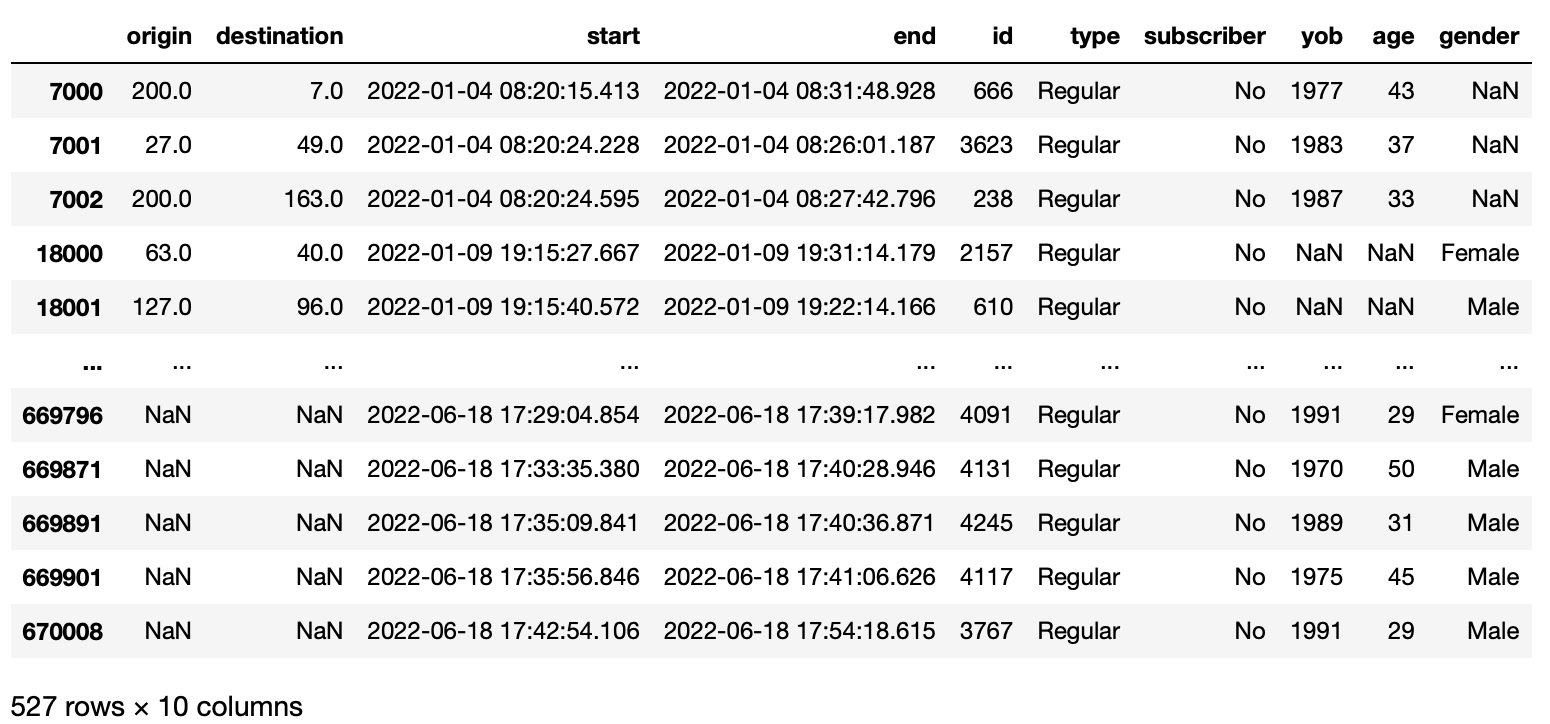
First, for the missing values ‘-’, ‘--’, and ‘?’

**# Replace missing values as null values**

df = df.apply(lambda x: x.replace({'-': np.nan, '?': np.nan, '--': np.nan}))

**# Display all missing values**

df[df.isna().any(axis = 1)]



1. Impute missing Age and YOB variables with Mean

The missing values in **Age** coincide with **YOB**, with all the data values missing. This occurred <3mins on the 2022-01-09 19:15 - 2022-01-09-19:18 from the time the users started their travel. Mean imputation is one of the strategies that replaces missing values with the average of the specified column. This method is selected as the 10 missing values will not necessarily skew the entire 670009 observations. We decided to replace this missing value with the mean based on the given date and hour.

**# Changing the data type as the value "-" made the data type for age to be an object**

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

**# Find the mean age given the date and hour**

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

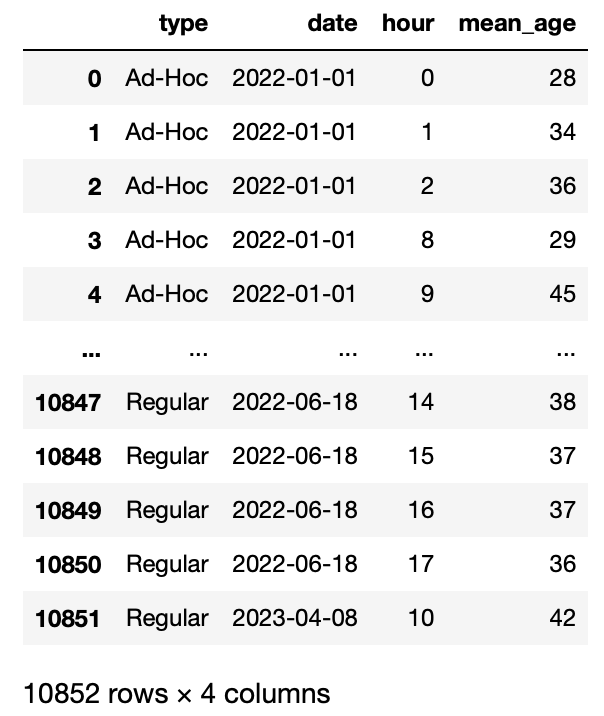
df['hour'] = pd.to\_datetime(df['start']).dt.hour

consumer\_age\_data = df.groupby(['type', 'date', 'hour'], as\_index = False).agg({'age':['mean']})

consumer\_age\_data.columns = ['type', 'date', 'hour', 'mean\_age']

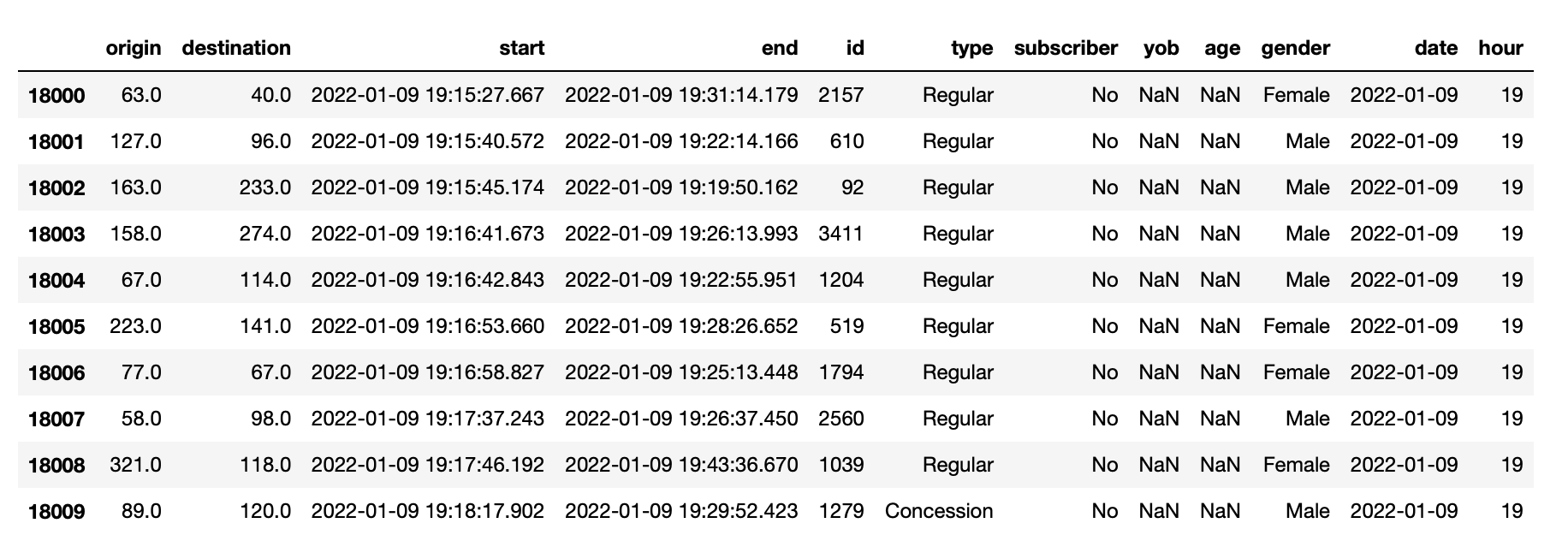
consumer\_age\_data['mean\_age'] = consumer\_age\_data['mean\_age'].astype(int)

consumer\_age\_data.head()



**# Display Missing Values in Age and YOB**

display(df.loc[df['age'].isna()])



**# Impute missing age with mean**

def impute\_age(consumer\_data, imputation\_data):

**# Impute missing age**

merge\_data = pd.merge(consumer\_data, imputation\_data, on = ['type', 'date', 'hour'], how = 'left')

merge\_data.loc[merge\_data['age'].isna(), 'age'] = merge\_data['mean\_age']

merge\_data.drop(columns = ['mean\_age'], inplace = True)

**# Impute YOB**

merge\_data.loc[merge\_data['yob'].isna(), 'yob'] = 2020 - merge\_data['age']

merge\_data[['yob', 'age']] = merge\_data[['yob', 'age']].astype(int)

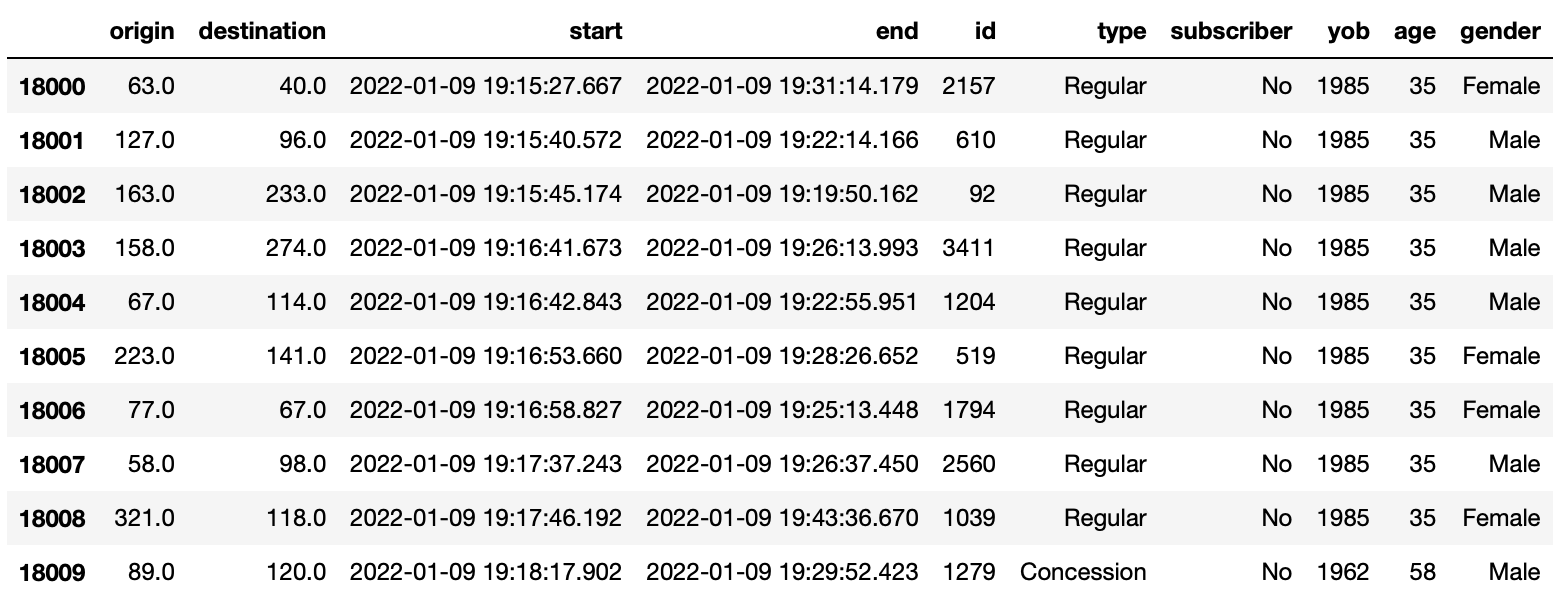
merge\_data.drop(columns = ['date', 'hour'], inplace = True)

return merge\_data

**# Display Imputed Age and YOB Value**

df = impute\_age(df, consumer\_age\_data)

df.iloc[18000: 18010]

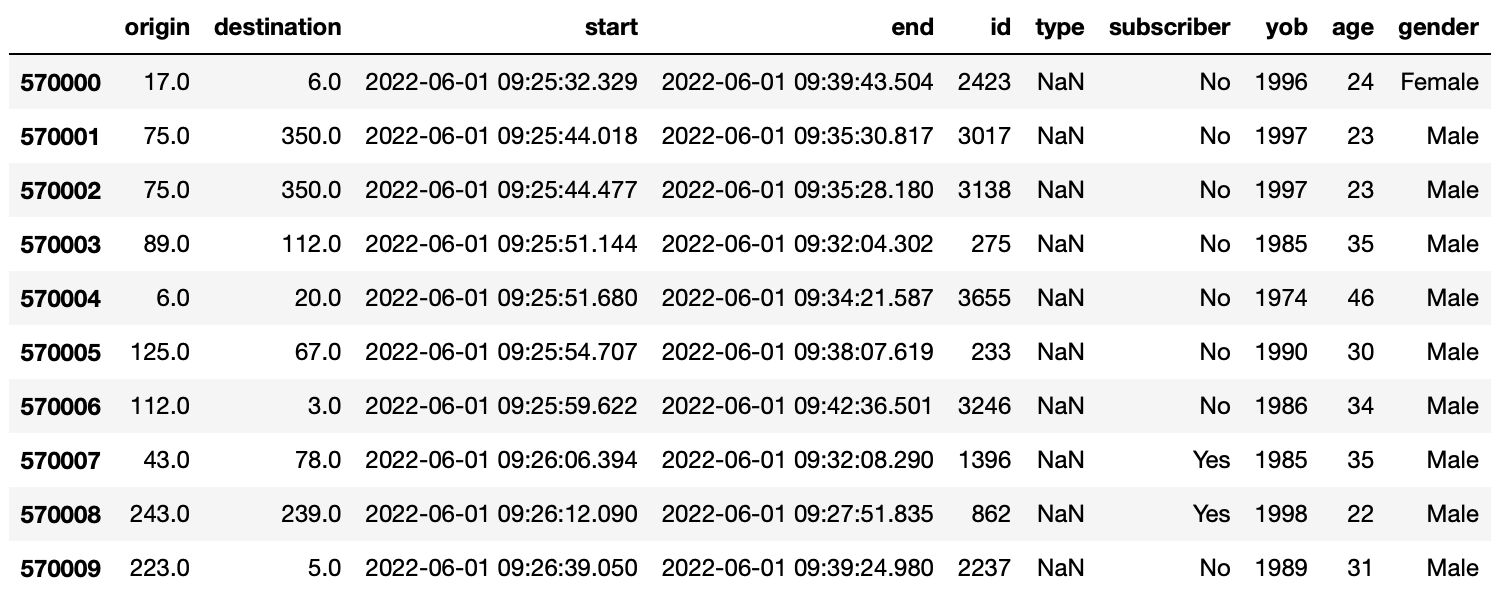


2. Impute missing Type variables with Mode of the specific age group

For the missing values in type, we decided to fill them in based on the most popular type according to the age group. We will first bin the age group into four different categories, namely ‘Below 30’, ‘Between 30 to 39’, ‘Between 40 to 49’ and ‘Above 50’. From there, we will find out the type count for each category, retrieve the mode and sort them into the missing values accordingly. Substituting missing values with mode values is recommended when the data is skewed.

**# Display Missing Values in Type**

df.loc[df['type'].isna()]



**# Age Categorisation**

def age\_categorisation(x):

if x >= 50:

return '50 and above'

elif x >= 40:

return 'Between 40 to 49'

elif x >= 30:

return 'Between 30 to 39'

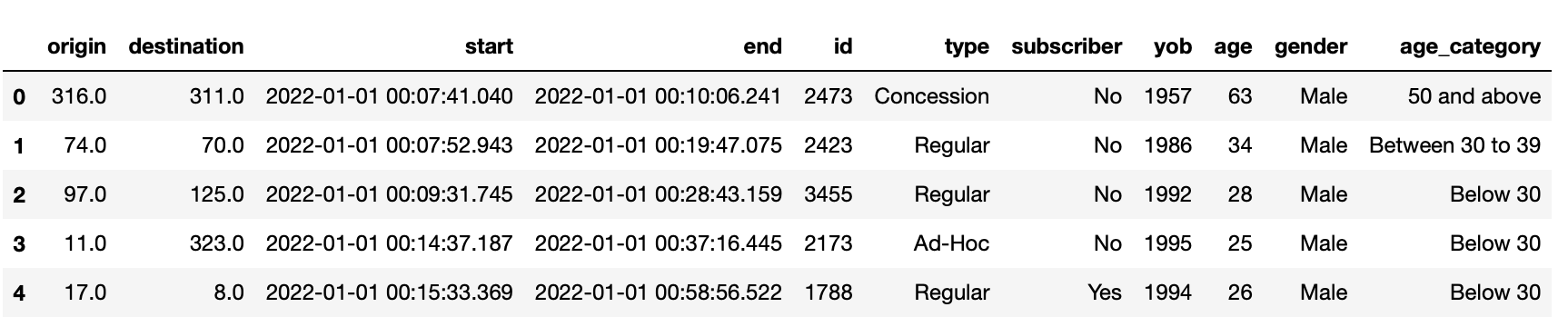
else:

return 'Below 30'

**# Display Age Categorisation**

df['age\_category'] = df['age'].apply(age\_categorisation)

df.head()



**# Identifying the Mode of type of the relevant age category**

age\_category\_type\_data = df.groupby(['age\_category'])['type'].value\_counts().reset\_index(name = 'Count')

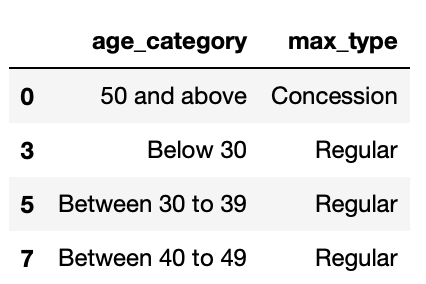
age\_category\_type\_data.sort\_values(['age\_category', 'Count'], inplace = True)

age\_category\_type\_data.rename(columns = {'type': 'max\_type'}, inplace = True)

age\_category\_type\_data.drop\_duplicates(subset = ['age\_category'], keep = 'last', inplace = True)

age\_category\_type\_data.drop(columns = ['Count'], inplace = True)

age\_category\_type\_data



From the above, we can see that for age\_category Below 30, between 30 to 39 and Between 40 to 49, the most frequent type is “Regular”. The type “Concession” only falls under age\_category “50 and above”.

**# Impute missing age with mode**

def impute\_type(consumer\_data, imputation\_data):

# Impute missing type

merge\_data = pd.merge(consumer\_data, imputation\_data, on = ['age\_category'], how = 'left')

merge\_data.loc[merge\_data['type'].isna(), 'type'] = merge\_data['max\_type']

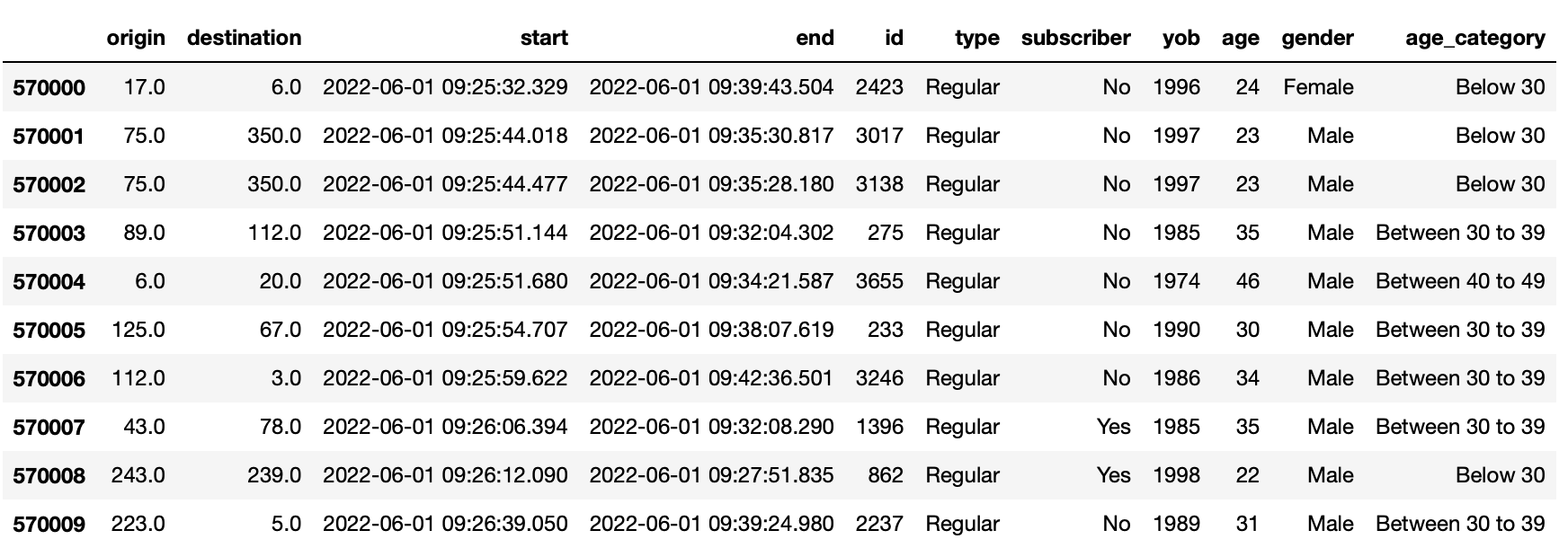
merge\_data.drop(columns = ['max\_type'], inplace = True)

return merge\_data

**# Display Imputed Type Value**

df = impute\_type(df, age\_category\_type\_data)

df.iloc[570000: 570010]

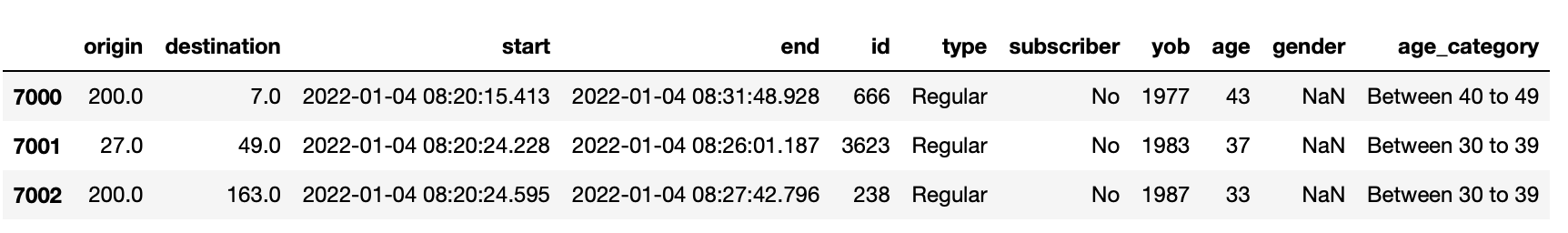


3. Impute missing Gender variables with Others

The missing values for **gender** were 3 values that were also within a minute that happened on the same day, 2022-04-01 08:20 from the time the users started their travel. Under “gender”, there are fields such as ‘Male’, ‘Female’ and ‘Others’. The assumption here was that those unknown or blank values indicated as ‘Others’ as such input will not have any significant analysis to commuter journeys.

**# Display Missing Values in Gender**

df.loc[df['gender'].isna()]



**# Display the unique values**

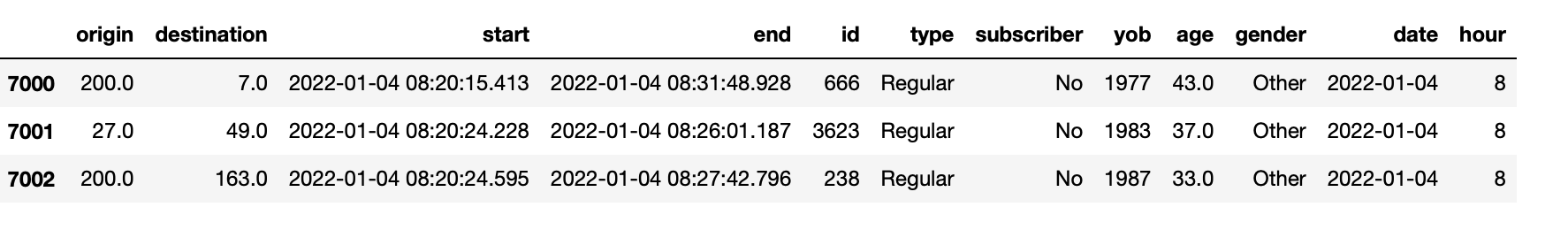
df['gender'].unique()



**# Impute Gender Values**

df.loc[df['gender'].isna(), 'gender'] = 'Other'

df.iloc[7000:7003]



4. Removing Missing Values in Origin and Destination

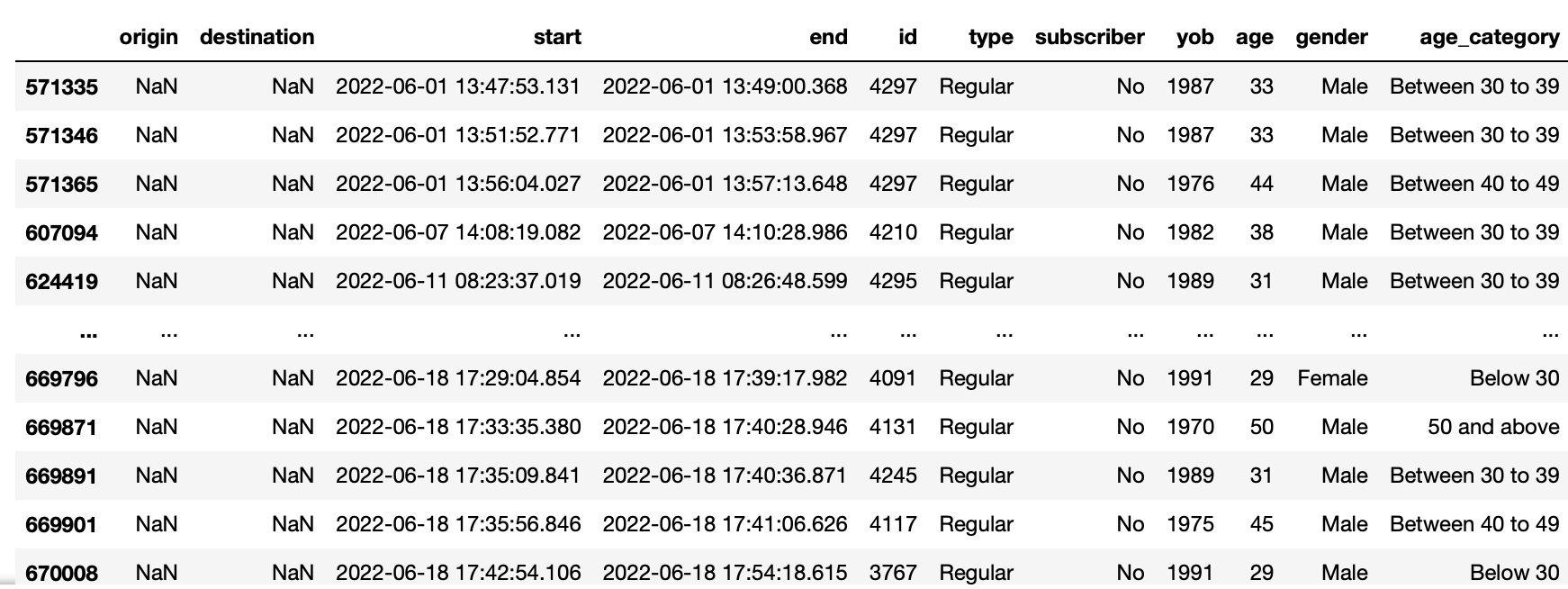
Lastly there is no pattern for missing values in **Origin** and **Destination**. We noticed that there were a total of 504 missing values that were sporadic and did not occur at the same time.

df.shape



**# Display missing values**

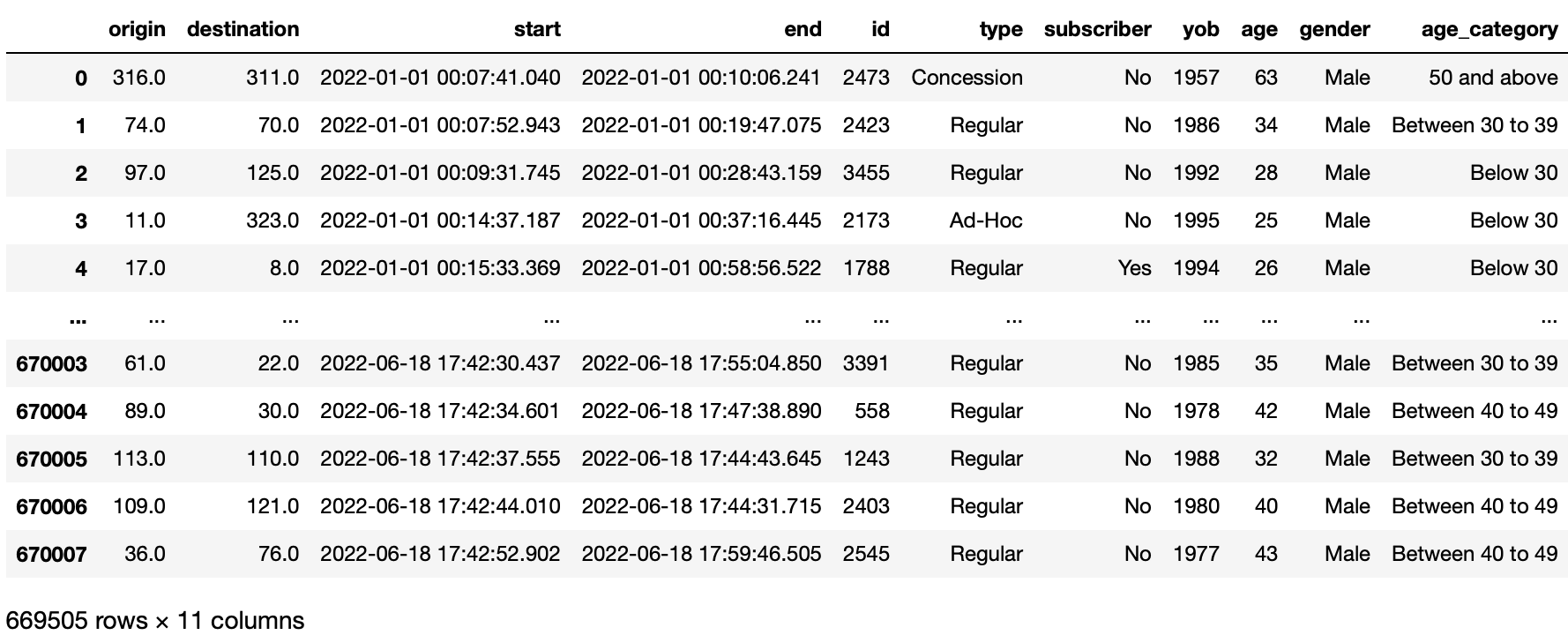
df[df.isna().any(axis = 1)]



**# Dropping missing Origin and Destination Values**

df = df.dropna()

df



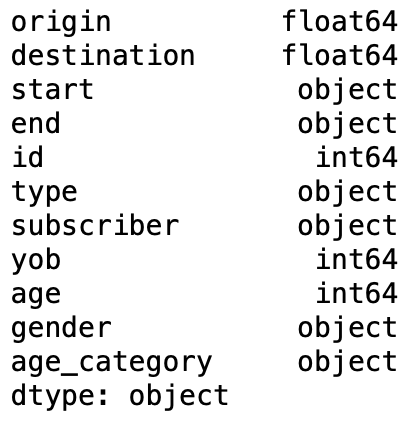
Hence for Origin and Destination, because of the lack of data and information given we decided to drop the Data.

**1(c)**

Data Issue 1 - Data Formats

**#Data Issue 1 - Data Preparation for Data Formats**

print(df.dtypes)



**# Changing to the right data format**

df["start"] = pd.to\_datetime(df["start"], format="%Y-%m-%d %H:%M:%S")

df["end"] = pd.to\_datetime(df["start"], format="%Y-%m-%d %H:%M:%S")

df['id'] = df['id'].astype('float')

df['type'] = df['type'].astype('category')

df['subscriber'] = df['subscriber'].astype('category')

df['gender'] = df['gender'].astype('category')

df['yob'] = df['yob'].astype('float')

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

df['origin'] = df['origin'].astype('float')

df['destination'] = df['destination'].astype('float')

**# Display New Data Format**

print(df.dtypes)

A screenshot of a computer

Description automatically generated with low confidence

Essentially to change all the formats of the data mentioned above to the correct data formats. It is necessary to change the datatype to the necessary so that we are able to better manipulate the values of the data given. I.e. calculations for integers/floats, or datetime manipulation to be able to simply extract years out of the given values and the format of the years.

It also helps us categorize the datatypes as we manipulate it for graphs and analysis.

Data Issue 2 - Data with the wrong date range

There are also data lapses that contain start and end dates in the year 2023, which is yet to occur. Our solution is to omit values that have the date range above the current date. There are 2023 values seen in the dataset.

**#Identifying Wrong Date Range**

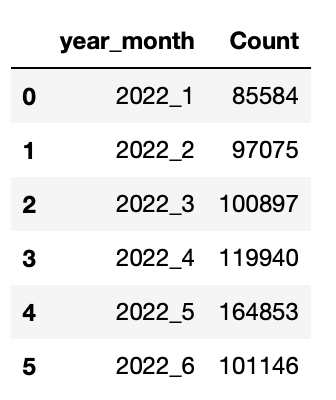
df['year'] = df['start'].dt.year

df['month'] = df['start'].dt.month

df['year\_month'] = df['year'].astype(str) + "\_" + df['month'].astype(str)

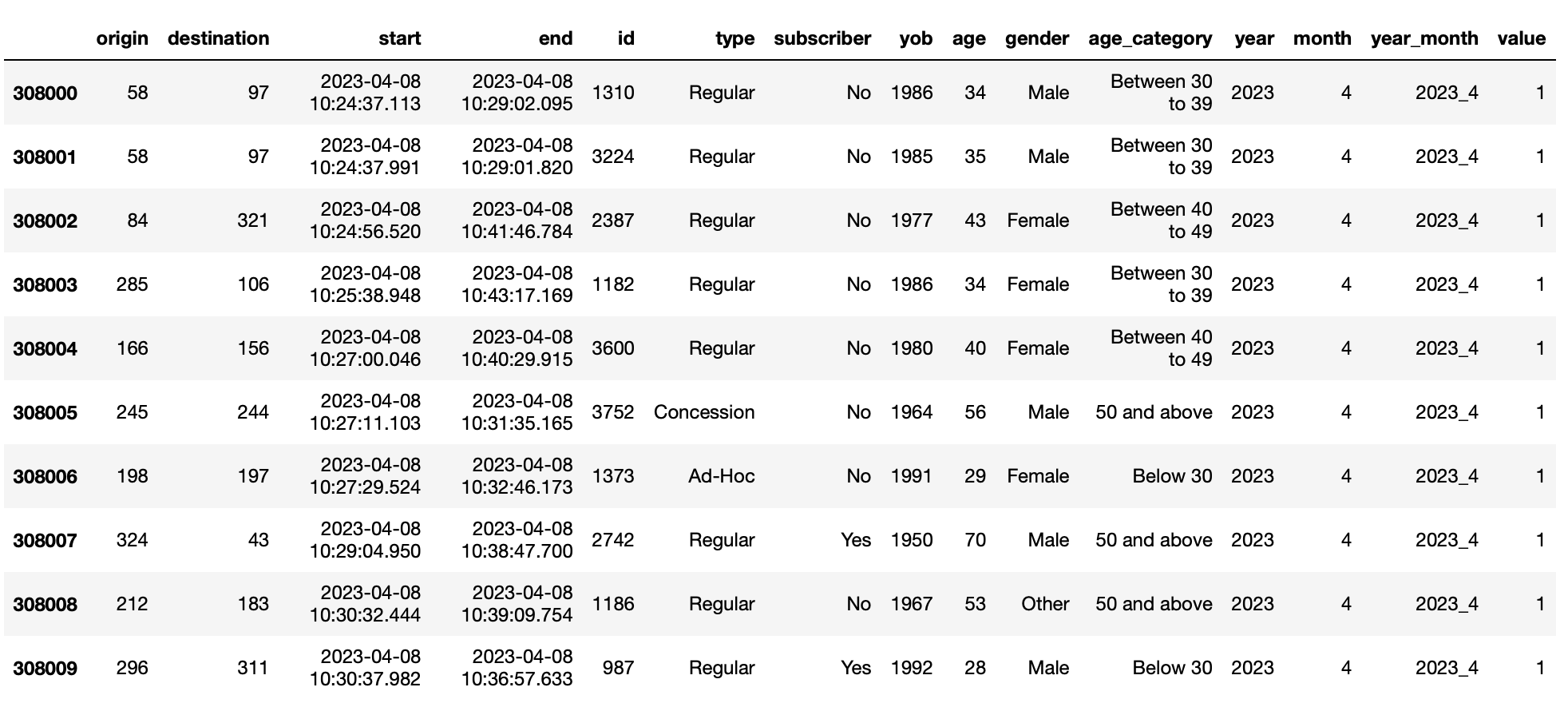
df['value'] = 1

df.groupby(['year\_month'])['value'].count().reset\_index(name = 'Count')



**# Displaying data values above today's date**

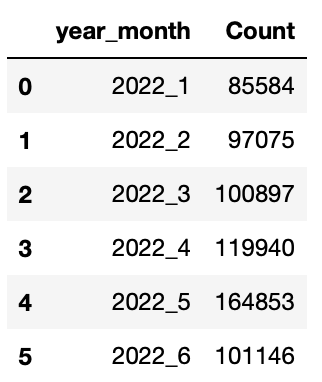
df.loc[df['start'].dt.date > datetime.date.today()]



**# Removing data values from wrong date range**

df = df.loc[df['start'].dt.date < datetime.date.today()]

df.groupby(['year\_month'])['value'].count().reset\_index(name = 'Count')



Data Issue 3 - Inaccurate data

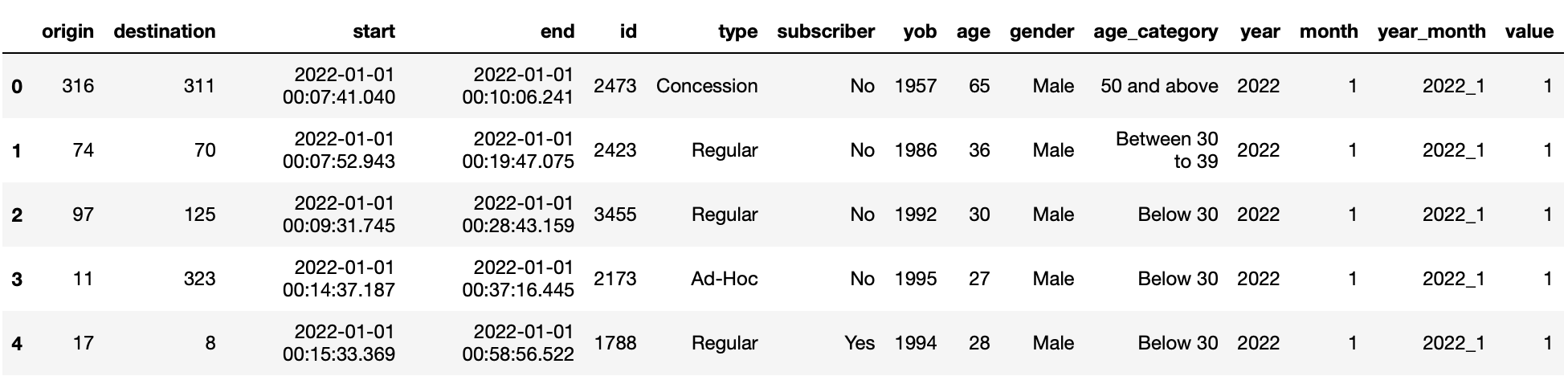
Age column is also not accurate as it does not take into account time that has passed from the time that the users signed up. Hence the need to change the time from The start and end date is 2022, but YOB and AGE is from 2020. In order to give real time age, we decided to take today’s date and extract the year from it and subtract the user’s birth year such that we’d have the most up to date data on the actual age of the users.

**#Recalculate the YOB and Age from today’s date (2022)**

current\_year = datetime.date.today().year

df['age'] = current\_year - df['yob']

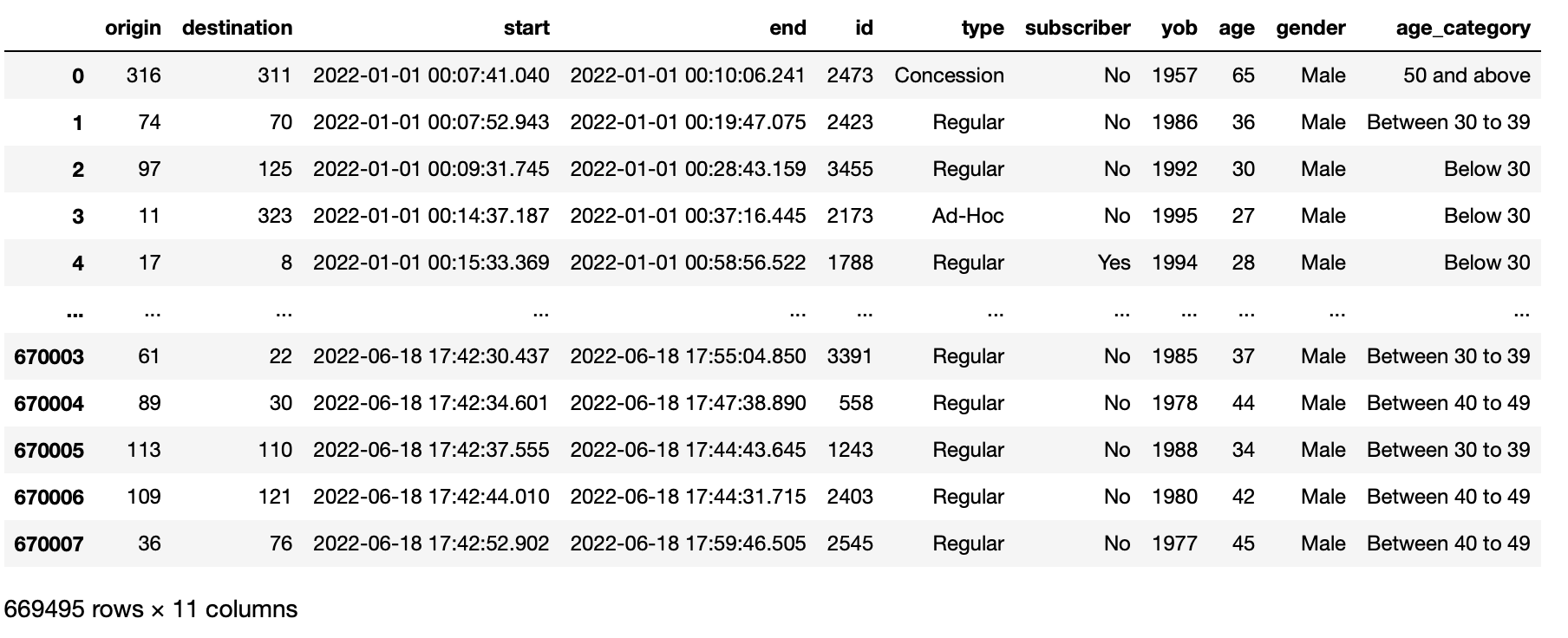
df.head()



**#Final Data Set: Dropping Additional columns not required in modelling.**

df.drop(columns = ['year', 'month', 'year\_month', 'value'], inplace = True)

df

****

**1(d)**

Prep the initial data to ensure that the date time format is accurate.

Create a new column which contains the start time such that it doesn’t affect the original data

Change format of date time to a 12 hour clock

Group the time slots to better understand which hour has the most number of people leaving.

Sort the group with the highest number first.

Create a user defined function peak() to define the peak hour where most commuters start their journey. Of which the defined function brings the highest value of the grouped values.

The answer would be 5pm

Code in Words

def peak(df):

df['start\_time'] = pd.to\_datetime(df["start"], infer\_datetime\_format=True)

df['start\_time'] = df['start\_time'].dt.strftime("%I %p")

df\_grouped = df.groupby(['start\_time'])['id'].count().reset\_index(name = 'count')

df\_grouped.set\_index('start\_time', inplace = True)

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

print(f"The peak departure time is {df\_grouped['count'].idxmax()}")

peak(df)

Output



**1(e)**

Age\_group and Commuter Types (By Count and Percentage Count)

Firstly, we would like to find out the travel behaviours of the age\_group based on the customer profile type. This is essential to gain an in-depth understanding of the distinctive customer profile, in order to plan marketing efforts and maximize its effectiveness in terms of customer service, customer satisfaction, sales, and revenue. Different age\_group behave very differently as they travel around, and various commuter types have different sets of offerings and benefits valued for them.

**# Plotting bargraph with plotly, setting x-axis as unit, y-axis as id, categories by type**

fig = px.histogram(df, x="age\_category", y="id",

color='type',

barmode='group',

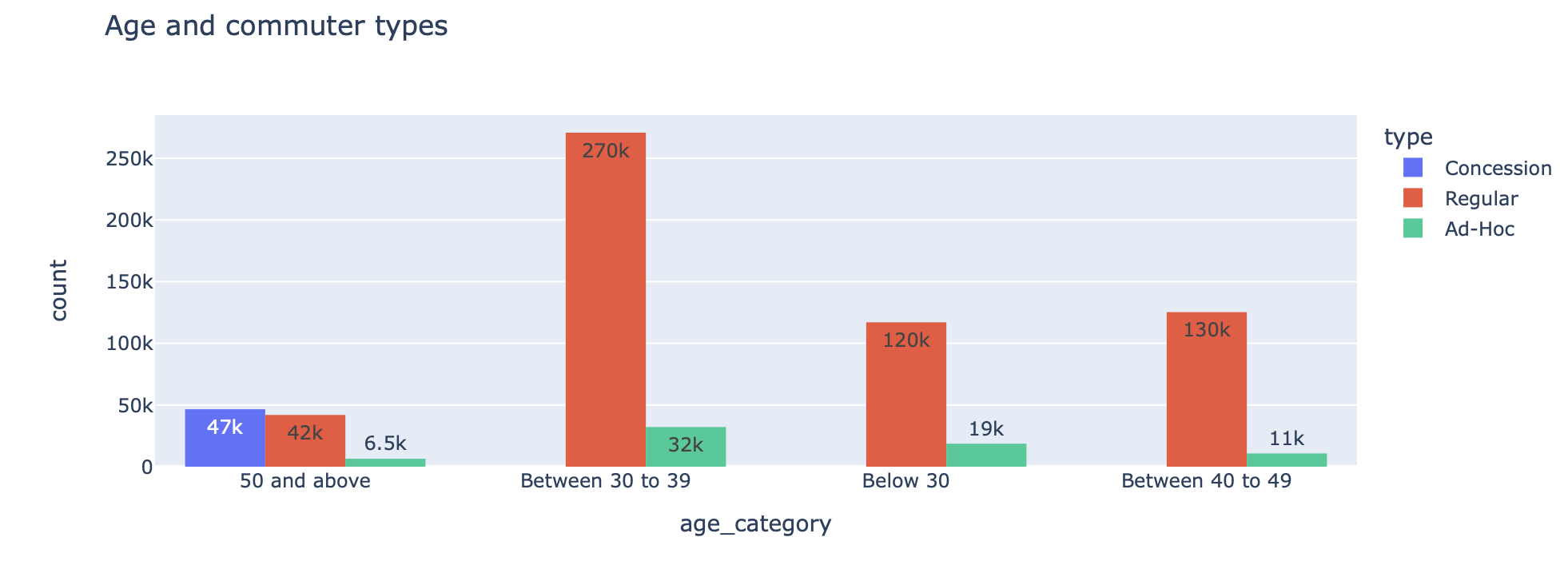
histfunc='count',

height=400,

title = "Age and commuter types",

text\_auto = ".2s")

fig.show()



**# To show Per Percentage Count**

df\_stack = df.groupby(['age\_category','type']).size().reset\_index()

df\_stack['Percentage']=df.groupby(['age\_category','type']).size().groupby(level=0).apply(lambda x:100 \* x/float(x.sum())).values

df\_stack.columns= ['age\_category','type', 'Counts', 'Percentage']

df\_stack['Percentage'] = df\_stack['Percentage'].astype(float, errors = 'raise')

df\_stack.round(2)

fig = px.bar(df\_stack, x="age\_category", y="Percentage",color='type',

title="Bar Plot",

template="plotly\_white",

height=600,

text = df\_stack['Percentage'].round(2))

**# Updating chart labels**

fig.update\_layout(barmode = "relative", title = "Percentage count of type by age category")

# Formatting layout

fig.update\_traces(textposition='inside')

fig.update\_layout(uniformtext\_minsize=8, uniformtext\_mode='hide')

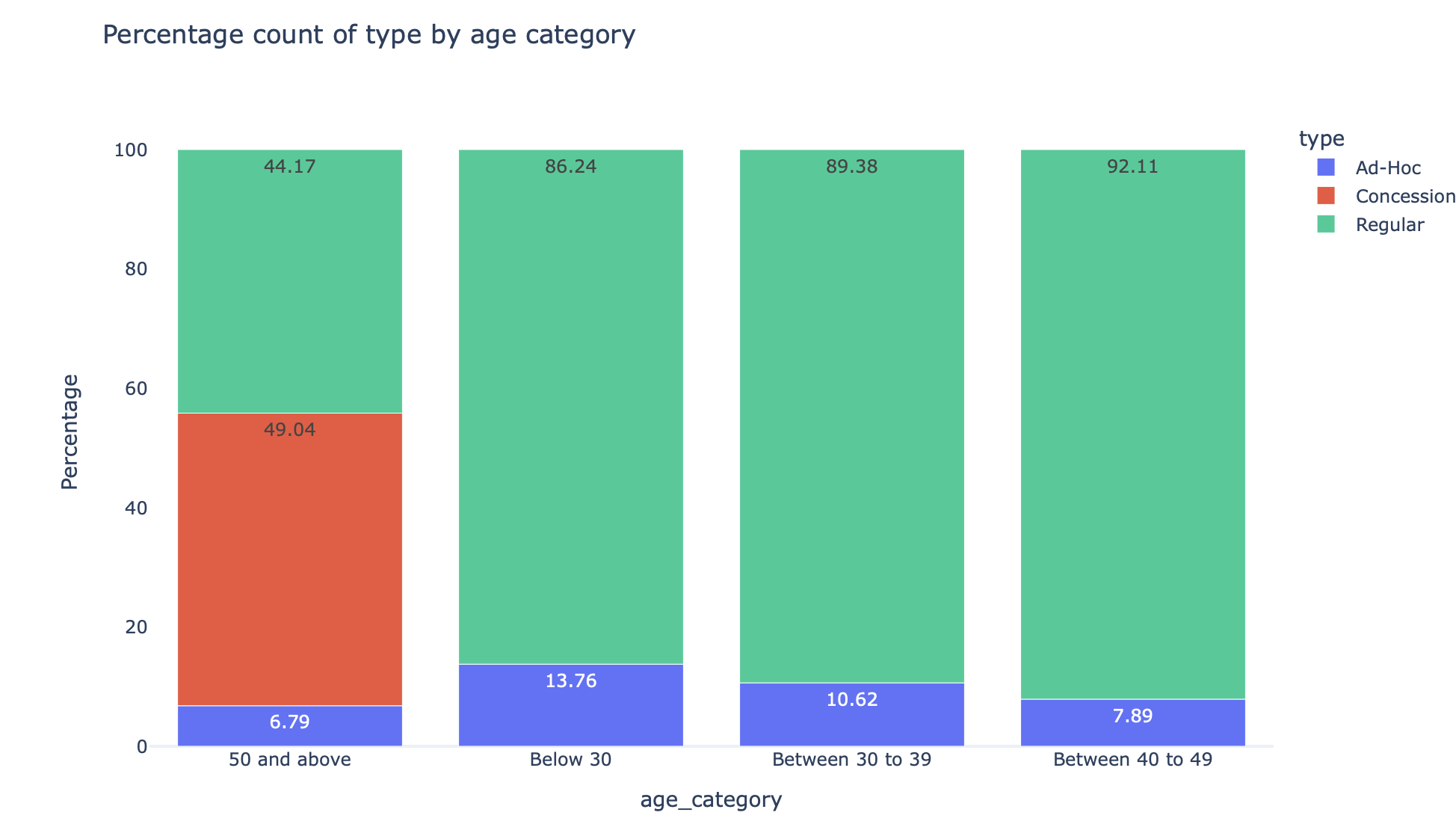
fig.update\_layout(plot\_bgcolor='white')

fig.update\_yaxes(showline=False,showgrid=False)

fig.update\_xaxes(showline=False,showgrid=False)

**#Show graph**

fig.show()



Analysis:

From both graphs above, we can see that commuters with the profile type ‘Regular’ are the most popular type across the age\_group ‘Below 30, Between 30 to 39 and Between 40 to 49’. For age\_group ‘50 and above’, it consists of a mixture of the 3 types, where the majority of the commuter type is ‘Concession’.

A small percentage of ‘Adhoc’ types can be seen among the various age\_groups. As its name suggests, "adhoc" refers to a temporary form that has been established or utilized for a specific and immediate purpose, with no prior record. It is usually lesser among the rest of the commuter types

In Singapore’s context, we define ‘Concession’ type as monthly unlimited train and/or basic bus travels. ‘Regular’ type refers to commuters who topups their passes on an often basis while ‘Adhoc’ refers to commuters on instances where they paid cash on the public transports instead of utilising their passes.

‘Concession’ type are seen largely in age\_group 50 and above as older adults and elderlies will want to enjoy the convenience and ease of these passes. Besides, such passes are also more subsidised for this particular age\_group.

To encourage the use of ‘Concession’ passes for the other age\_groups, marketing efforts can be projected to show the value offering and savings one can get compared to ‘Regular’ passess. This includes showing a comparison of the monthly prices of the two passes. However, this also boils down to the travel frequency and the destination. For instance, an adult who travels to and fro from work in town everyday may be recommended to have the ‘Concession’ pass to speed her his traveling time and not worry about whether any occurences of low funds are in his pass.

Which age group travel longer

The next analysis is to identify which age group travels longer

**#Identifying the travel duration**

import plotly.express as px

df[['start', 'end']] = df[['start', 'end']].astype('datetime64[ns]')

df['duration'] = (df['end'] - df['start'])/np.timedelta64(1, 'm')

df.groupby(['age\_category'])['duration'].describe()

Table

Description automatically generated

According to the graph, the maximum journey time is significantly longer than the average. Such observations are referred to be outliers. We will employ winsorisation to reduce the impact of outliers by assigning them a lesser weight (99 percentile). This reduces the impact of outliers by substituting them for less severe numbers.

**# Reduce Outliers**

winsorize\_parameters = df.groupby(['age\_category'])['duration'].quantile(0.99).reset\_index(name = '99\_quantile')

**# Updating Format**

df = pd.merge(df, winsorize\_parameters, on = ['age\_category'], how = 'left')

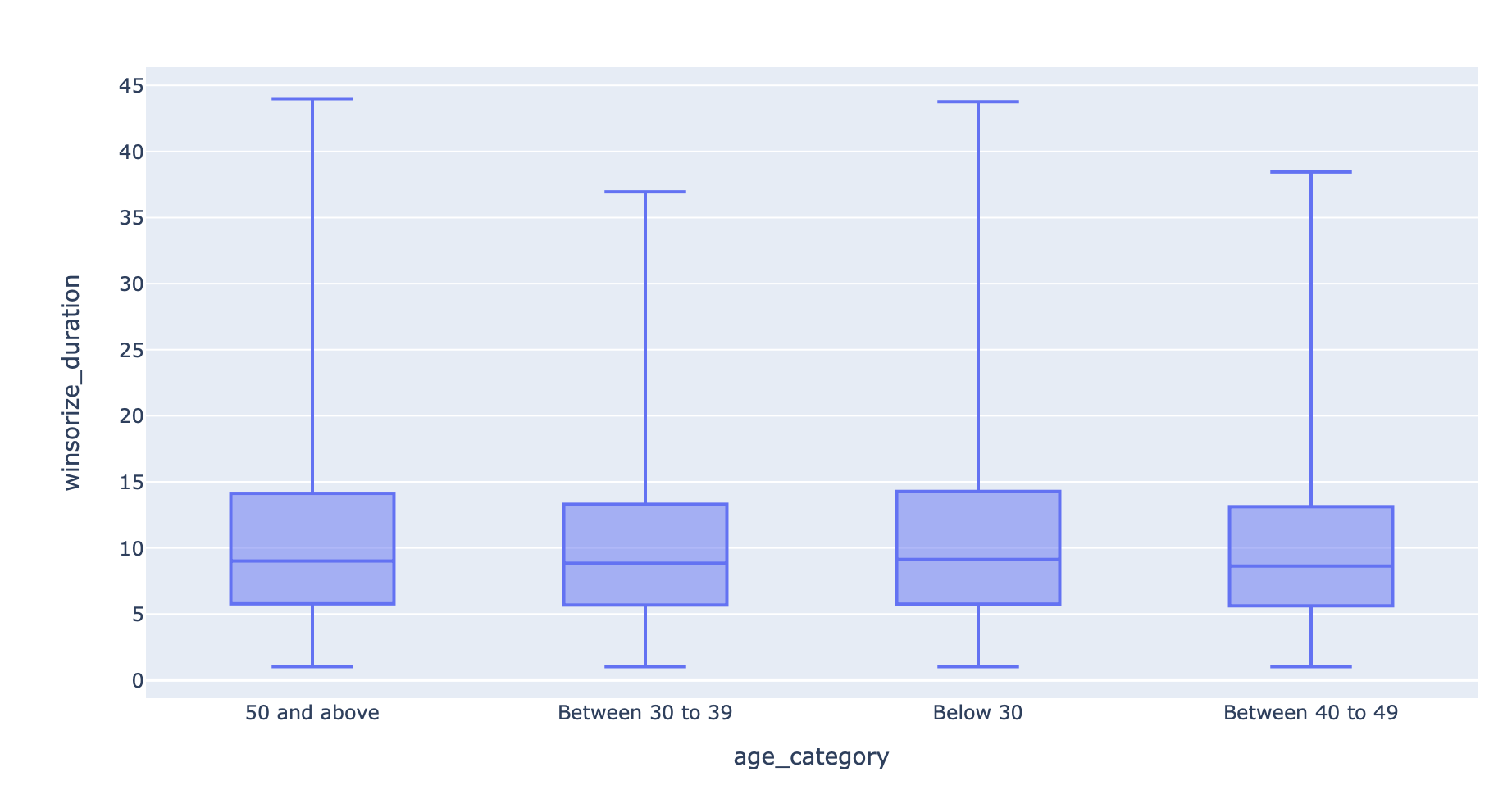
df['winsorize\_duration'] = df.apply(lambda x: x['99\_quantile'] if(x['duration'] >= x['99\_quantile']) else x['duration'], axis = 1)

**# Diplay Graph**

fig = px.box(df, x="age\_category", y="winsorize\_duration")

fig.update\_traces(boxpoints=False)

fig.show()



Analysis:

From the boxplot above, we can tell that commuters aged 50 and above and below 30 travels the longest compared to those aged between 30 to 39 and 40 to 49. We can also see that the general bulk of users from 1st quartile to 3rd fair between a similar range. This tells us that the middle age groups tend to not travel as long as the younger and older age groups.

Younger adults are more likely to travel longer to City Centre to work as better prestigious companies are also located there. This is also likely related to them generally having higher educational qualifications and better income, which are leading factors to higher and longer public transport use. For adults and seniors aged 50 and above, travelling time may be longer because these individuals may not want to spend extra money on other forms of transportation. They do not mind longer traveling hours as long as they can get to their destination on time.

Subscribers vs Commuter Types

**# To show Per Percentage Count**

df\_stack = df.groupby(['type','subscriber']).size().reset\_index()

df\_stack['Percentage']=df.groupby(['type','subscriber']).size().groupby(level=0).apply(lambda x:100 \* x/float(x.sum())).values

df\_stack.columns= ['type','subscriber', 'Counts', 'Percentage']

df\_stack['Percentage'] = df\_stack['Percentage'].astype(float, errors = 'raise')

df\_stack.round(2)

fig = px.bar(df\_stack, x="type", y="Percentage",color='subscriber',

title="Bar Plot",

template="plotly\_white",

height=600,

text = df\_stack['Percentage'].round(2))

**# Updating chart labels**

fig.update\_layout(barmode = "relative", title = "Subscribers vs commuters type")

**# Formatting layout**

fig.update\_traces(textposition='inside')

fig.update\_layout(uniformtext\_minsize=8, uniformtext\_mode='hide')

fig.update\_layout(plot\_bgcolor='white')

fig.update\_yaxes(showline=False,showgrid=False)

fig.update\_xaxes(showline=False,showgrid=False)

**#Show graph**

fig.show()

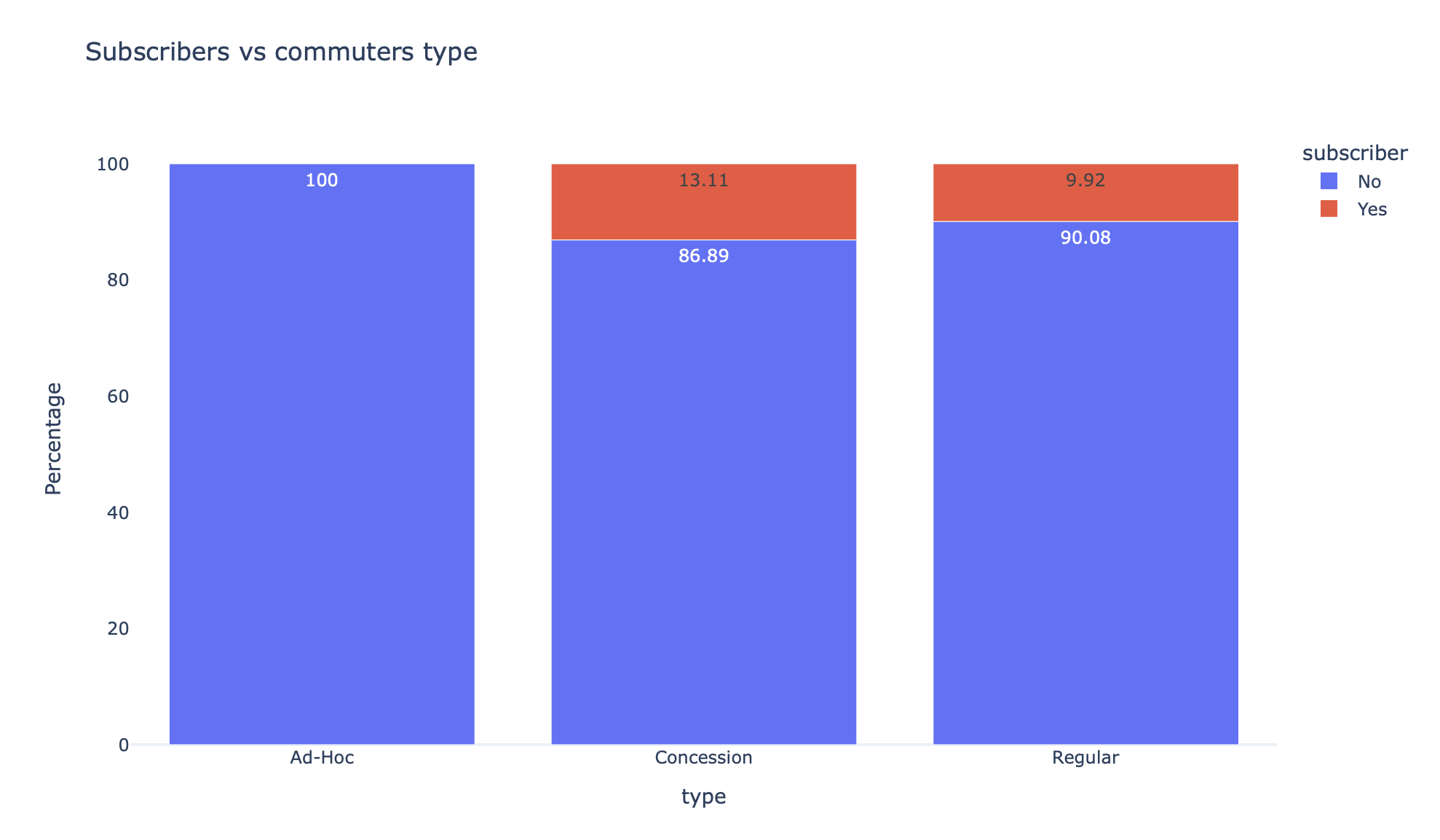


fig = px.histogram(df, x="type", y="id",

color='subscriber',

barmode='group',

histfunc='count',

height=400,

title = "Subscribers vs commuters type",

text\_auto = ".2s")

fig.show()

Chart, bar chart

Description automatically generated

Analysis:

From the graphs above, we can also see that none of the ad-hoc users are subscribers, which is not an anomaly because these ad-hoc users would be deemed as the users that use the service the least as and when they decide to.

With regards to concession and regular commuters the percentages of subscribers is about 10% and 13% respectively. With this data we can see that the pick-up rate for regular passes and concession passes do not convert into subscribers of the service, this would mean that the subscription portion of the business might not be too convincing in terms of perks that majority of its heavier users don’t see a benefit in subscribing, either that or there is lack of awareness in getting the users to subscribe.

In accordance with the mobility-as-a-service (MaaS) paradigm, the primary purpose of subscription plans is to provide commuters multiple subscription plans based on their transport needs, allowing them to plan their journeys and pay for numerous modes of transportation through a single platform. Such subscription programs are likely to permit customization based on a user's particular requirements.

Hence, the company can focus on the target offerings and re-evaluate its marketing components as well as the current offerings of the subscription to determine a better course of action to get more subscribers.