Contents

[Introduction 4](#_Toc124080655)

[Data Dictionary of dataset 4](#_Toc124080656)

[Objective 4](#_Toc124080657)

[Part A 4](#_Toc124080658)

[Solution 4](#_Toc124080659)

[Code 5](#_Toc124080660)

[Code 5](#_Toc124080661)

[Part B 6](#_Toc124080662)

[Solution 6](#_Toc124080663)

[Observation 6](#_Toc124080664)

[Output 7](#_Toc124080665)

[Observation 7](#_Toc124080666)

[Explaining reasoning for solution 8](#_Toc124080667)

[Implementing ways to treat missing values 8](#_Toc124080668)

[Output 8](#_Toc124080669)

[Output 9](#_Toc124080670)

[Observation 9](#_Toc124080671)

[Code 9](#_Toc124080672)

[Output 9](#_Toc124080673)

[Observation 9](#_Toc124080674)

[“yob” column 9](#_Toc124080675)

[Code 9](#_Toc124080676)

[Output 10](#_Toc124080677)

[Code 10](#_Toc124080678)

[Output 10](#_Toc124080679)

[“age” column 10](#_Toc124080680)

[Code 10](#_Toc124080681)

[Code 11](#_Toc124080682)

[“gender” column 11](#_Toc124080683)

[Output 11](#_Toc124080684)

[Code 11](#_Toc124080685)

[Output 11](#_Toc124080686)

[Is there still any missing value? 12](#_Toc124080687)

[Output 12](#_Toc124080688)

[Observation 12](#_Toc124080689)

[Interesting Observation 12](#_Toc124080690)

[Part C 12](#_Toc124080691)

[Solution 13](#_Toc124080692)

[Treating Issue 1 13](#_Toc124080693)

[Output 13](#_Toc124080694)

[Output 13](#_Toc124080695)

[“yob” column 13](#_Toc124080696)

[Output 14](#_Toc124080697)

[Output 14](#_Toc124080698)

[Output 14](#_Toc124080699)

[Treating Issue 2 14](#_Toc124080700)

[Output 15](#_Toc124080701)

[Output 15](#_Toc124080702)

[Observation 15](#_Toc124080703)

[Treating issue 3 15](#_Toc124080704)

[Output 16](#_Toc124080705)

[Output 16](#_Toc124080706)

[Removing the missing valued records 16](#_Toc124080707)

[Output 16](#_Toc124080708)

[Output 17](#_Toc124080709)

[Observation 17](#_Toc124080710)

[Part D 17](#_Toc124080711)

[Solution 17](#_Toc124080712)

[Output 17](#_Toc124080713)

[Output 18](#_Toc124080714)

[Part E 18](#_Toc124080715)

[Solution 18](#_Toc124080716)

[Code 18](#_Toc124080717)

[Graph 19](#_Toc124080718)

[Observation 19](#_Toc124080719)

[Creating another Visualization 19](#_Toc124080720)

[Code 19](#_Toc124080721)

[Graph 20](#_Toc124080722)

[Observation 20](#_Toc124080723)

[Code 20](#_Toc124080724)

[Graph 20](#_Toc124080725)

[Observation 21](#_Toc124080726)

[Code 21](#_Toc124080727)

[Graph 21](#_Toc124080728)

[Observation 21](#_Toc124080729)

[Code 21](#_Toc124080730)

[Graph 22](#_Toc124080731)

[Observation 22](#_Toc124080732)

[Code 22](#_Toc124080733)

[Graph 23](#_Toc124080734)

[Observation 23](#_Toc124080735)

[Interesting Insights 23](#_Toc124080736)

# Introduction

The given dataset is about the commuters’ journeys. The data is comprised of 670009 records. There are 10 different variable columns which were supposed to be analyzed to get several insights from the data.

# Data Dictionary of dataset

|  |  |
| --- | --- |
| **Variable** | **Description** |
| origin | Start location identifier |
| destination | End location identifier |
| start | Start time of commuter |
| end | End time of commuter |
| id | Device/Vehicle identifier |
| type | Commuter profile type |
| subscriber | Subscribing commuter (Yes/No) |
| yob | Commuter year of birth |
| age | Commuter Age |
| gender | Commuter Gender |

# Objective

The object of this GBA is to analyze the dataset, perform necessary cleaning, and get familiar with the process of data preparation.

# Part A

In the requirements, it is given that in the dataset, ‘-’, ‘--’, and ‘?’ are considered as missing values, and the variable columns of the dataset are noted as in the data dictionary in order. As part of data preparation, read the dataset in as a Pandas dataframe, with the above considerations.

# Solution

# Importing the necessory library

import pandas as pd

# Reading the dataset as Pandas dataframe

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

commuters.head()

By using this code, we imported the required dataset in a variable and saw the top 5 rows of the data.

Text

Description automatically generated

After that, we checked the basic information about the data. This was done by using the following code:

## Code

# Checking the basic information about the data

commuters.info()

Output:

Table

Description automatically generated

## Code

# Using describe() to get some more information about the features

commuters.describe()

Table

Description automatically generated

With the given considerations, the dataset was read.

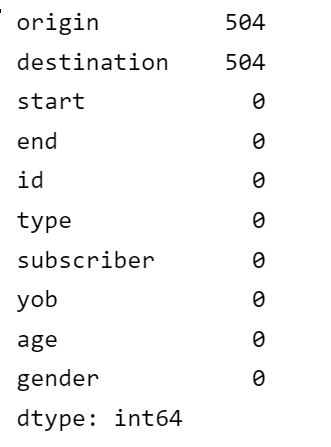
# Part B

Identify the variable columns which have missing values. As part of data preparation, implement ways to treat them, and explain your rationale. State any interesting observation(s).

# Solution

# Checking the null values in the dataset.

commuters.isna().sum()



## Observation

There are two columns in the dataset which have blank values in the samples. Both columns are of float datatype.

To identify the variables which contained the missing values, following code was written:

# Making a userdefined function to check which feature columns have any of the missing values.

def check\_missing\_values(variable\_column):

    '''

    Check any of the missing value in the variable column.

    '''

    missing\_value\_1 = "-" in variable\_column

    missing\_value\_2 = "--" in variable\_column

    missing\_value\_3 = "?" in variable\_column

    to\_return = [True]

    if missing\_value\_1:

        to\_return.append("-")

    if missing\_value\_2:

        to\_return.append("--")

    if missing\_value\_3:

        to\_return.append("?")

    if missing\_value\_1 or missing\_value\_2 or missing\_value\_3:

        return to\_return

    else:

        return False

# Storing the results in a dictionary by calling a function on all columns

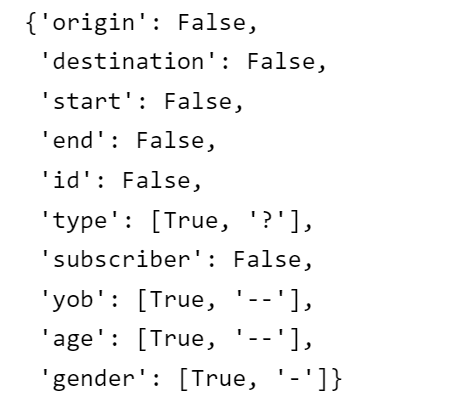
result = {}

for column in commuters.columns:

    result[column] = check\_missing\_values(list(commuters[column].unique()))

result

## Output



## Observation

We have identified that the feature columns `origin` and `destination` have the blank values in them by using `isna()`. Also, there are the columns `type`, `yob`, `age`, and `gender` which have the missing values. The function is returning the columns which have missing values along with the missing values which are present in those columns.

# Explaining reasoning for solution

When we are dealing with the missing values in a dataset, we can perform different operations on them. They could be:

1. Fill the missing values with some appropriate value (Also known as Imputation)
2. Remove the samples with the missing values altogether

In the case of excessively large number of samples present, we can skip the samples with the missing values. Otherwise, we prefer to fill them with some appropriate values.

The missing values can be filled with the maximum number of values that are present in the column. They can also be filled with the average of all the available values in the case of numeric column. In some cases, we can fill the missing values with the label "missing" to identify that these were the cells which are filled during Imputation.

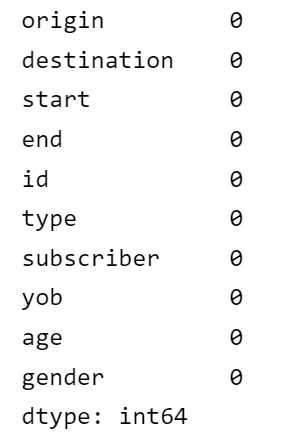
# Implementing ways to treat missing values

# Removing the samples with missing values in the “origin” column

commuters.dropna(subset=['origin'], inplace=True)

commuters.isna().sum()

## Output



commuters.info()

Now from the output shown above, we can see that the `type` column has `?` missing value. As the `type` is of `object` datatype, we will be filling it with the maximum value that appears in the `type` column.

commuters['type'].value\_counts()

## Output

Table

Description automatically generated with medium confidence

## Observation

`type` column has a value: `Regular` which appeared most of the times. So, we are filling the missing value with `Regular`.

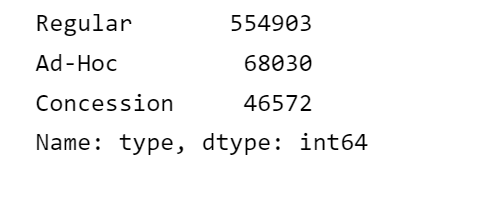
## Code

commuters['type'] = commuters['type'].replace({"?": "Regular"})

# Checking the "type" column for missing values

commuters['type'].value\_counts()

## Output



## Observation

Now we see that, the missing value does not exist anymore. Similar procedure will be applied for other columns with missing values.

## “yob” column

## Code

commuters['yob'].value\_counts()

## Output

Table

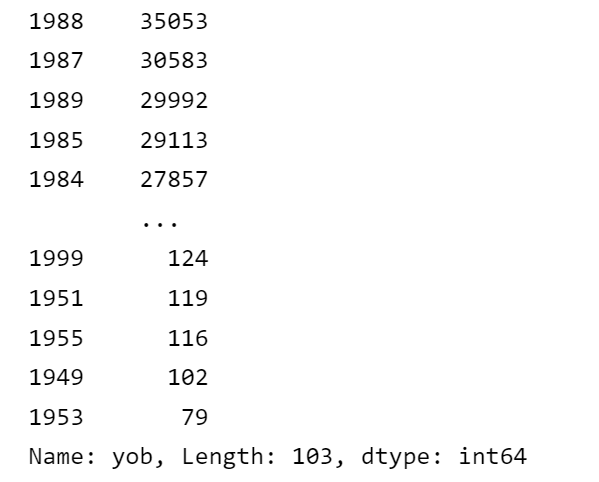
Description automatically generated

## Code

commuters['yob'] = commuters['yob'].replace({"--": 1988})

commuters['yob'].value\_counts()

## Output



## “age” column

## Code

commuters['age'].value\_counts()

Table

Description automatically generated

## Code

commuters['age'] = commuters['age'].replace({"--": 32})

commuters['age'].value\_counts()

Table

Description automatically generated

## “gender” column

commuters['gender'].value\_counts()

## Output

Table

Description automatically generated with medium confidence

## Code

commuters['gender'] = commuters['gender'].replace({"-": "Male"})

commuters['gender'].value\_counts()

## Output

Text

Description automatically generated with low confidence

## Is there still any missing value?

# Checking whether any of the column still contains some missing values or not

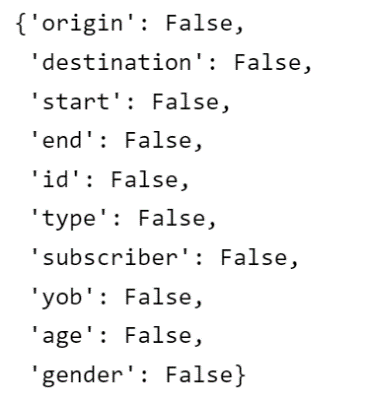
result = {}

for column in commuters.columns:

    result[column] = check\_missing\_values(list(commuters[column].unique()))

result

## Output



## Observation

All the missing values are now gone, and the dataset is now filled.

## Interesting Observation

Following are some interesting observations made during the process:

1. When performing the replacement in the data, we encountered the issue of `chained indexing`. While the process of chain indexing, we got `SettingWithCopy` warning. This occurred because in this case, it's hard to predict whether it will return a view or a copy. That's why we changed this method with the replacing method.
2. While finding the missing values, the blank values were not being compared as they had not any specific datatype. So, for getting rid of those missing data, we removed the samples with the missing values.
3. After handling with the missing values, our dataset is now filled and does not contain any missing values.

# Part C

As part of data preparation, identify ***three (3)*** other data quality issues in the data. Similarly, suggest and implement ways to treat them, and explain your rationale.

# Solution

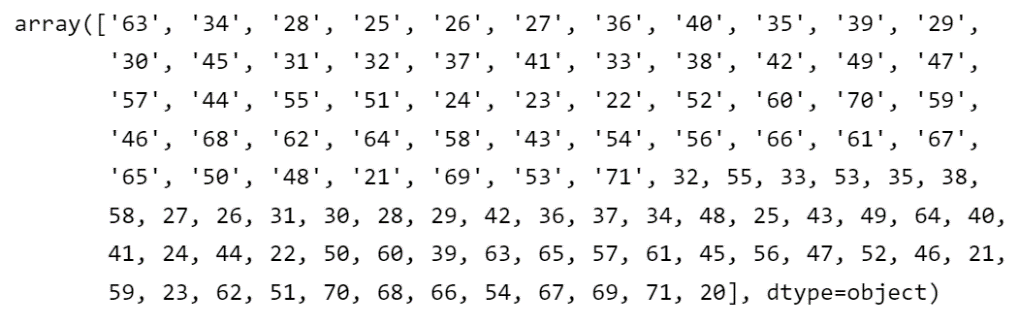
The other quality issues encountered are following:

1. **Mixed Datatypes**: There is `type mismatch` in `age` and `yob` columns. Some of the values are in string format and some are in the numeric format. This causes the problem in dealing with the same kind of data. For example, if we want to sort the list, it would not possible because of the data type mixing.
2. **Create Dummy Variables**: There are some categorical variable columns which need to be reduced during data preparation. Keeping all our data in numeric format is one of the quality assurances about the data.
3. **Too much Data**: I think that there is too much data in the dataset which is causing some difficulty in getting information from some attributes of data. Due to the massive amount of data, some other data quality issues are becoming more severe. It sometimes become difficult to extract information of such type of columns which have too much unique values as we cannot compare them with some other independent variable columns. Sometimes these kinds of features need transformation to be properly scaled.

# Treating Issue 1

commuters['age'].unique()

## Output



We will now resolve this issue:

commuters['age'] = commuters['age'].astype('int64')

commuters['age'].unique()

## Output

Text

Description automatically generated

# “yob” column

commuters['yob'].unique()

## Output

Table

Description automatically generated

commuters['yob'] = commuters['yob'].astype('int64')

commuters['yob'].unique()

## Output

Table

Description automatically generated

commuters.info()

## Output

Table

Description automatically generated

# Treating Issue 2

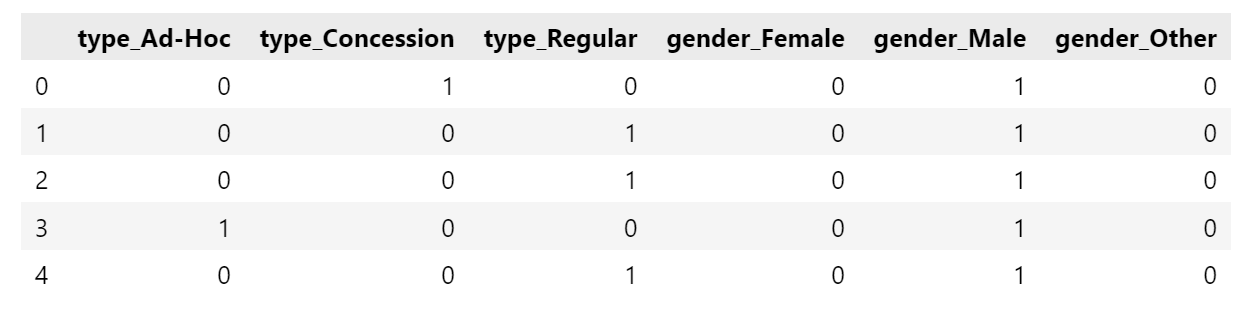
Now we will see how to convert categories into numbers

commuters.head()

dummies = pd.get\_dummies(commuters[["type", "gender"]])

dummies.head()

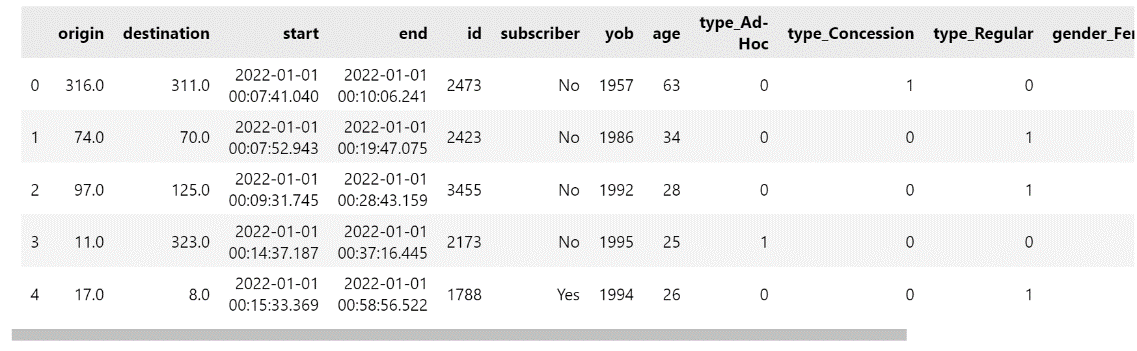
## Output



dummy\_dataframe = commuters.drop(["type", "gender"], axis=1)

pd.concat([dummy\_dataframe, dummies], axis=1).head()

## Output



Table

Description automatically generated

## Observation

Now we have the encoded the values of the columns and the nominal nature of variables is reduced to numbers. We can use it in our dataset and then use for further procedure. If we treat any numerical column as a category, we can use astype to change them to 'category', 'str' or 'object' before converting it to dummy.

# Treating issue 3

For treating this issue, we can transform our variables so that they can be matched with other variables in the dataset.

But for achieving some sort of precision, we can also remove all the data with missing or ambiguous value so that our dataset can get rid of the records with missing values and the massive nature of data can also be reduced along with that.

# Reading the dataset again

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

df.head()

len(df)

## Output



# Checking whether any of the column contains some missing values or not

result = {}

for column in df.columns:

    result[column] = check\_missing\_values(list(df[column].unique()))

result

## Output

Text

Description automatically generated with medium confidence

## Removing the missing valued records

# remove rows using the drop() function

df.drop(df.index[df['type'] == '?'], inplace=True)

df.drop(df.index[df['yob'] == '--'], inplace=True)

df.drop(df.index[df['age'] == '--'], inplace=True)

df.drop(df.index[df['gender'] == '-'], inplace=True)

len(df)

## Output

A picture containing logo

Description automatically generated

# Checking whether any of the column contains some missing values or not

result = {}

for column in df.columns:

    result[column] = check\_missing\_values(list(df[column].unique()))

result

## Output

Text

Description automatically generated with low confidence

## Observation

Here we can see that the input data now does not contain any assumed values. Also, some amount of data is reduced which is not affecting much as compared to our initial amount of data. So, sometimes we need to remove the ambiguous data to retain the precision.

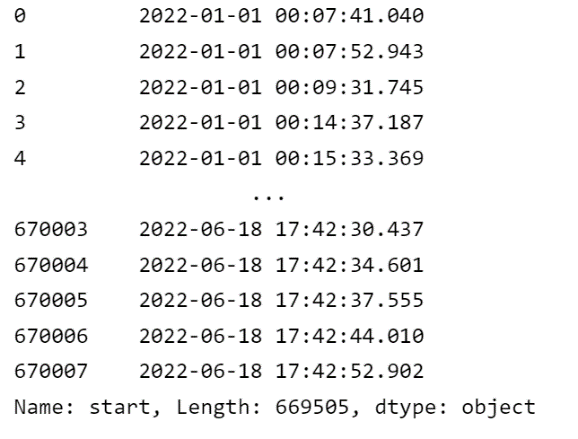
# Part D

Develop a user-defined function that will print the hour, expressed in the 12-hour clock format (e.g., 12am, 1pm), whereby the highest number of commuters start their journey.

# Solution

commuters['start']

## Output



def get\_highest\_starting\_hour():

    start\_times = []

    for start\_date\_and\_time in commuters['start']:

        start\_date\_and\_time = start\_date\_and\_time.split(' ')

        start\_times.append(start\_date\_and\_time[1])

    occurences = {}

    for start\_time in start\_times:

        start\_time = start\_time.split(':')[0]

        if start\_time in occurences:

            occurences[start\_time] += 1

        else:

            occurences[start\_time] = 1

    highest\_starting\_hour = int(max(occurences, key=occurences.get))

    if highest\_starting\_hour > 12:

        return str(highest\_starting\_hour - 12) + "pm"

    elif highest\_starting\_hour == 0:

        return "12am"

    elif highest\_starting\_hour == 12:

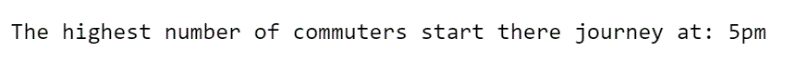
        return "12pm"

    else:

        return highest\_starting\_hour + "am"

print(f"The highest number of commuters start there journey at: {get\_highest\_starting\_hour()}")

## Output



# Part E

Write a Python code to create appropriate visualisations of the commuter data. Analyse the results and then discuss three (3) interesting insights.

# Solution

## Code

import matplotlib.pyplot as plt

import numpy as np

%matplotlib inline

commuters['type'].value\_counts()

commuters['type'].value\_counts().plot(kind="bar",

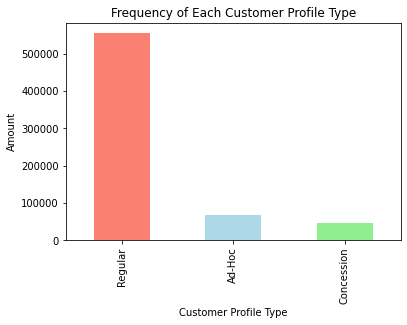
                                      color=['salmon', 'lightblue', 'lightgreen']);

plt.title('Frequency of Each Customer Profile Type')

plt.xlabel('Customer Profile Type')

plt.ylabel('Amount');

## Graph



## Observation

We can see that a greater number of commuters are of `Regular` type rather than Concession of Ad-Hoc. It means that the ratio of regular customers is more than the others.

## Creating another Visualization

## Code

pd.crosstab(commuters.subscriber, commuters.type).plot(kind="bar", figsize=(10, 6), color=["salmon", "lightblue", "lightgreen"])

plt.title("Subscribers Values for each Customer Profile Type")

plt.xlabel("Subscriber")

plt.ylabel("Count");

## Graph

Chart

Description automatically generated with low confidence

## Observation

We can see here that the more `Regular` commuters are `non-Subscribing` commuters. Whereas here we can see that `Ad-Hoc` type of commuters are less in amount but all of them are non-Subscribing. We cannot see any of `Ad-Hoc` commuter who is Subscribing as well.

## Code

pd.crosstab(commuters['age'], commuters['gender']).plot(kind="bar", figsize=(15, 8), color=['salmon', 'lightblue', 'lightgreen'])

plt.title("Age Frequency of each gender")

plt.xlabel("Age")

plt.ylabel("Count");

plt.yticks(np.arange(0, 30000, 1000));

## Graph

Chart, histogram

Description automatically generated

## Observation

Here we can infer that more of the commuters are Males. Also, the maximum number of commuters are of age `32`. This graph is clearly showing that we have very rare number of commuters who have `other` gender. Also, after age 60, there is a smaller number of Females. The range of age can be seen from 20 to 71.

## Code

# Create figure

plt.figure(figsize=(12,8))

# Scatter for subsctibing customer

plt.scatter(commuters.type[commuters.subscriber=="Yes"],

            commuters.id[commuters.subscriber=="Yes"],

            c="lightblue"); # axis always come as (x, y)

## Graph



## Observation

From the given information we can infer that there are no `Ad-Hoc` type of commuters who are subscribers. But we can see the graph of those commuters who are non-subscriber. This will show us that the non-subscribers are of each type.

## Code

# Create figure

plt.figure(figsize=(12,8))

# Scatter for subsctibing customer

plt.scatter(commuters.type[commuters.subscriber=="Yes"],

            commuters.id[commuters.subscriber=="Yes"],

            c="lightblue") # axis always come as (x, y)

# Scatter for non-subscribing customer

plt.scatter(commuters.type[commuters.subscriber=="No"],

            commuters.id[commuters.subscriber=="No"],

            c="salmon") # define it as a scatter figure

# Add some helpful info

plt.title("Subscriber in function of Type and Id")

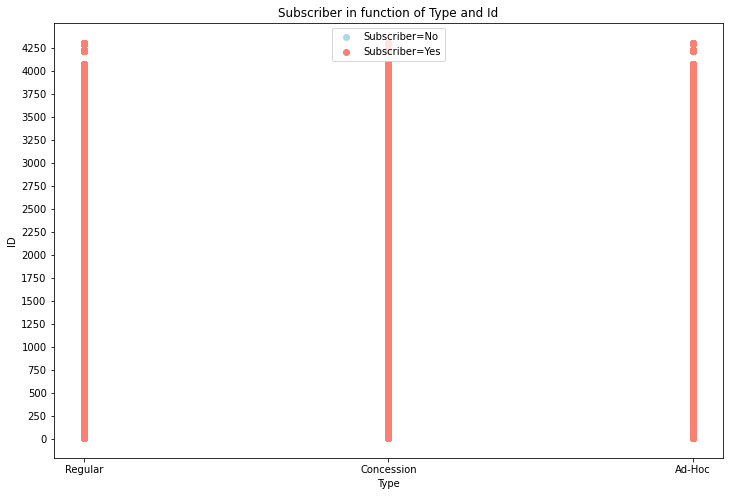
plt.xlabel("Type")

plt.legend(["Subscriber=No", "Subscriber=Yes"])

plt.ylabel("ID")

plt.yticks(np.arange(0, 4500, 250));

## Graph



## Observation

Now we can see that all the `lightblue` dots are hidden below the `salmon` dots. This is because the `non-subscribing` commuters are in all type of commuters and in all genders (will be shown below).

## Code

pd.crosstab(commuters['subscriber'], commuters['gender']).plot(figsize=(10, 6), kind='bar', color=['salmon', 'lightblue', 'lightgreen']);

# Add some helpful info

plt.title("Subscriber Frequency of Each Gender")

plt.xlabel("Subscriber")

plt.legend(["Female", "Male", "Other"])

plt.ylabel("Count");

## Graph

Chart, bar chart

Description automatically generated

## Observation

When we compare the subscribing and non-subscribing commuters, we can get to know that a greater number of Males and females are un-subscribing. Commuters with `other` as gender are only non-subscribing commuters.

Now as a whole we can infer that a greater number of consumers are `Un-Subscribing` altogether.

# Interesting Insights

After observing the whole dataset, we have got different interesting information about the data. Some interesting insights which we have got are:

1. More number of commuters are `Males`.
2. More number of commuters are `Un-Subscribing`.
3. We mostly have the `Regular` type of customers.
4. We can feel that the commuters of age range `25-40` are in greater amount. This means that we have more young commuters than the other aged ones.
5. Different commuters start their journey at different times but mostly the journey is started in evening time.