SMU
SINGAPORE MANAGEMENT UNIVERSITY

School of
Computing and
Information Systems

IS483: IS Project Experience (Business Analytics)

AY 2021/2022 Semester 1

Project CDG

Technical User Guide for (*Data Analysts*)

Team Members:

Name	ID	Email	Role
Choo Hui Xing	01378884	hxchoo.2018@scis.smu.edu.sg	Back-end Developer
Iris Har Jing Ru	01375564	iris.har.2018@scis.smu.edu.sg	Data Engineer
Ivy Hau Jia Yi	01350274	ivy.hau.2018@scis.smu.edu.sg	UI/UX Designer
Shazarifah Shawal	01373871	shazarifah.2018@scis.smu.edu.sg	Data Analyst
Shermin Tan	01374546	shermin.tan.2018@scis.smu.edu.sg	Project Manager
Sonia Johari	01376980	soniajohari.2018@scis.smu.edu.sg	Front-end Developer

Table of Contents

1.0 Brief Overview

This document entails a full step-by-step guide on replicating the analysis process for Project CDG. Do take the following document as reference and refer to the actual coding files as final.

2.0 Data Collection

2.1 Overall Data Sources

Please refer to the git repository [https://github.com/SMUxCDG/For-CDG-Ver.git] (*System* > *Data*) for all the data sources that will be used for these analyses.

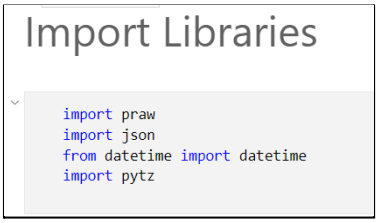
2.2 Data Scraping

2.2.1 Reddit

Step 1: Open reddit_scrape_CDG.ipynb

Navigate to *System* > *Code* > *Data Scraping* > *reddit_scrape_CDG.ipynb* to open the reddit scraping file.

Step 2: Import the relevant libraries

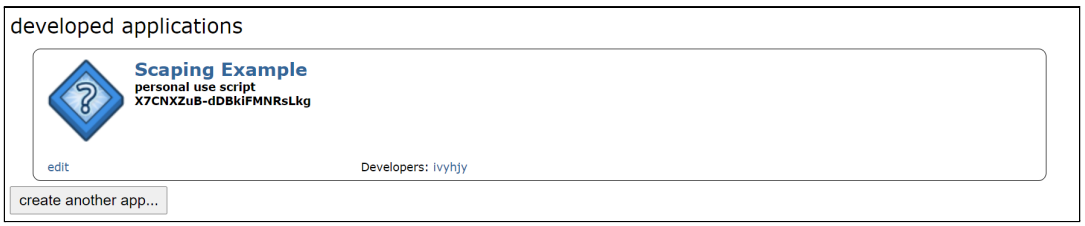


Step 3: Input the relevant IDs for scraping

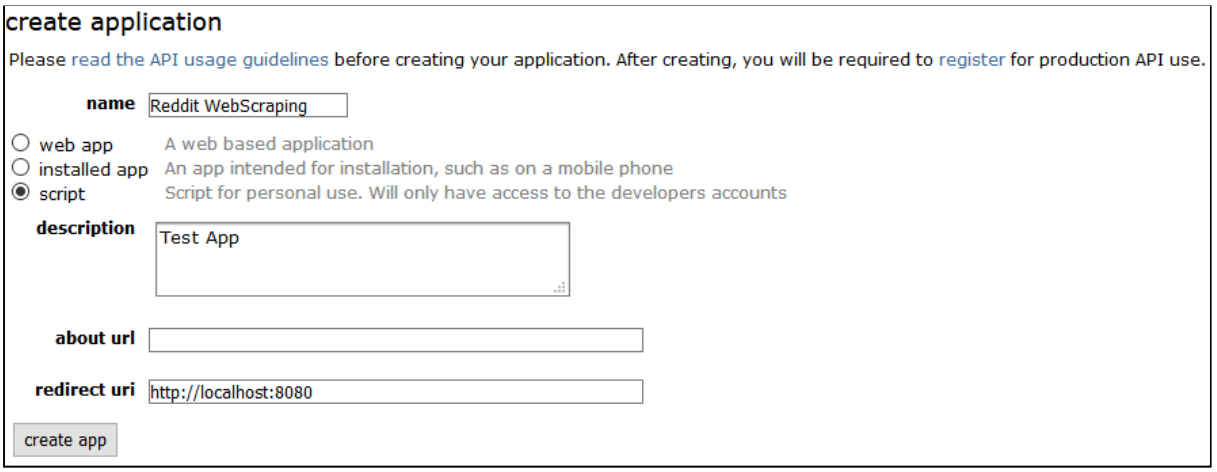
Before scraping data from reddit, a Reddit instance needs to be created with the client_id, client_secret, and the user_agent.

To get the authentication information we need to create a reddit app by navigating to this page and clicking create app or create another app.

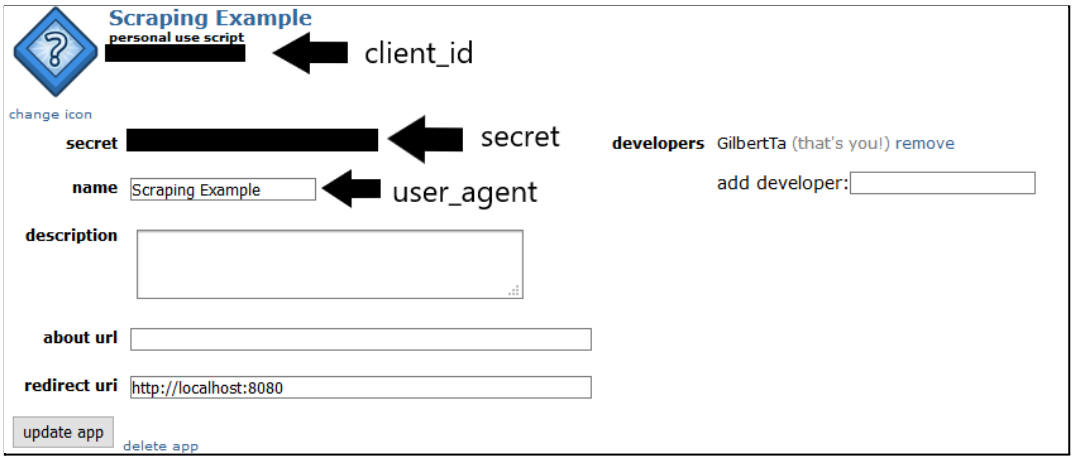
To get the authentication information, a Reddit app needs to be created at https://www.reddit.com/prefs/apps and clicking on “create another app”.



A form will be shown after clicking on “create another app”. Do fill in the specific details such as name. Do click on the “script” option and the redirect uri to http://localhost:8080 shown in the figure below.



After creating the app, copy the client_id, secret, and user_agent to a text file to later copy it to the script.



Copy the details to the script below.

Input Relevant IDs for Scraping

After getting the client ID, client secret, and user agent as per stated in the user guide, fill in the details in the code below.

```
reddit = praw.Reddit(client_id='',
                     client_secret='',
                     user_agent='')
```

Step 4: Run the scraping function

The function is created in the script. The function will scrape for the keyword in the specific subreddit page and export as "subredditpage_keyword.json". Do edit the path file to specify the specific folder the data would be exported to. The code to edit is commented as shown in the figure below.

Create Function to scrape reddit

scape_comment(subreddit, keywords) takes 2 elements, subreddit and keywords.

Subreddit refers to the subreddit page you wish to scrape from. Keywords refers to the keywords you wish to use to scrape in the stated subreddit page.

```
def scrape_comments(subreddit, keyword):
    sub = reddit.subreddit(subreddit)

    sub_dict = {}
    sub_lst = []
    sub_comments = []
    comment_dict = {}

    for submission in sub.search(keyword, limit = None):
        sub_dict['title'] = submission.title
        sub_dict['time created'] = pytz.utc.localize(datetime.utcfromtimestamp(submission.created_utc)).astimezone(pytz.timezone("Asia/Singapore")).strftime("%d/%m/%Y")
        sub_dict['score']= submission.score
        sub_dict['id'] = submission.id
        sub_dict['url']= submission.url

        submission.comments.replace_more(limit = None)
        for comment in submission.comments.list():
            comment_dict['time created'] = pytz.utc.localize(datetime.utcfromtimestamp(comment.created_utc)).astimezone(pytz.timezone("Asia/Singapore")).strftime("%d/%m/%Y")
            comment_dict['author'] = str(comment.author)
            comment_dict['score'] = comment.score
            comment_dict['comment'] = comment.body
            sub_comments.append(comment_dict)
            comment_dict = {}
        sub_dict['comments'] = sub_comments

        sub_comments = []
        sub_lst.append(sub_dict)
        sub_dict = {}
        comment_dict = {}

    # The code below write the file to current directory
    # Change the path file to the relevant data file you wish to save to
    with open(f"{subreddit}_{keyword}.json", 'w', encoding = 'utf-8') as file:
        file.write(json.dumps(sub_lst, indent=4))
```

Run the function with the following code. State the subreddit page and keywords for the Reddit scraping. For example "scrape_comments("Singapore", "taxi")" for taxi keywords in r/Singapore. scrape_comments can only take in one subreddit page and one keyword.

Run Function

Insert the relevant reddit page and keywords you wish to scrape and run the function below.

```
#State the subreddit page and keywords for the function
scrape_comments('subreddit page', 'keywords')
```

2.3 Data Pre-Processing

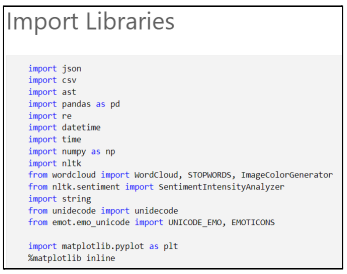
2.3.1 Data Cleaning

2.3.1.1 Reddit

Step 1: Open *Reddit Cleaning_CDG.ipynb*

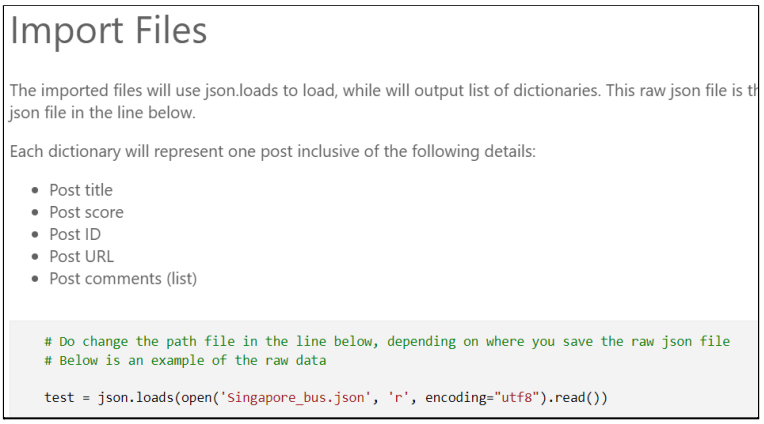
Navigate to *System* > *Code* > *Data Cleaning* > *Web Scrap* > *Reddit Cleaning_CDG.ipynb* to open the reddit cleaning file.

Step 2: Load Libraries



Step 3: Import Files

Here, the raw scraped data file will be loaded in order to clean it. The code in the figure below is an example of one of the loaded files. Do edit to include all the raw data files, as well as the correct file path in the code.



Step 4: Consolidate all data into a single dataframe

Below cleaning the data, the relevant json data files that were loaded will be consolidated into a single dataframe using the function that was created in the figure below.

Before running the function, the relevant variables containing the raw json data will be saved into a list. The list will then be looped and concat into a new dataframe to contain both the title and comments of each reddit post into the column "text".

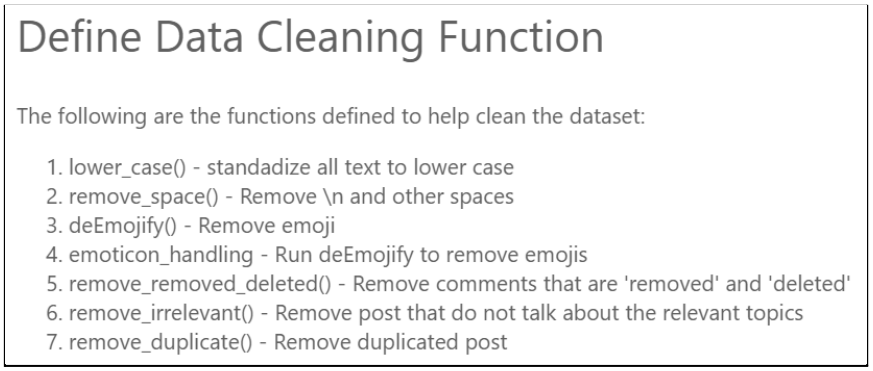
```
# Input all the imported variables that has the raw json data into a list
# The line below is an example

datasets = [test]
```

```
# This line will create an empty dataframe and loop the raw json data to save it into the dataframe

dataframe = pd.DataFrame()

for d in datasets:
    dataframe = pd.concat([dataframe, consolidate_text(d, 'title'), consolidate_text(d, 'comment')], axis = 0)
```

```
# This code checks if the data is properly loaded as a dataframe

dataframe.describe
```

Step 5: Define Data Cleaning Function

Next data cleaning functions will be created to clean the data frame. The figure below shows the relevant data cleaning to be performed.



Step 6: Execute Data Cleaning Functions

Next, the data cleaning functions will be executed, shown in the figure below.

Run the Data Cleaning Functions

```
dataframe_clean = lower_case(dataframe)
dataframe_clean = remove_space(dataframe_clean)
dataframe_clean = emoticon_handling(dataframe_clean)
remove_removed_deleted(dataframe_clean)
dataframe_clean = remove_irrelevant(dataframe_clean, keyword)
dataframe_clean.drop_duplicates(subset = ['text'], inplace = True)
```

Step 7: Categorizing Text into its Business Units

After cleaning, the text will be labelled with its business units, under the column "category" for further analysis.

Categorize Relevant Post into Groups

The relevant post will be categorized into the following groups:

1. Bus
2. MRT
3. Taxi
4. Private Hire
5. Car Rental

Define Function for Categorization

```
def categorize(df, group, group_text):

  t = []
  id = []
  d = []
  s = []
  cat = []

  for w in group:
    for i in range(len(df['text'])):
      if w in df['text'].iloc[i]:
        t.append(df['text'].iloc[i])
        id.append(df['author'].iloc[i])
        d.append(df['date'].iloc[i])
        s.append(df['score'].iloc[i])
        cat.append(str(group_text))

  df_new = pd.DataFrame()

  df_new['text'] = t
  df_new['author'] = id
  df_new['date'] = d
  df_new['score'] = s
  df_new['category'] = cat

  return df_new
```

Step 8: Execute Categorize() Function

Before running the categorize function, keywords for each business unit will be saved into their individual list as shown below. This is so the function can filter out the relevant text based on the keywords that are specified.

Run Categorization Function

```
# The code below are the keywords used to categorise the different business units.
# Do include or remove the following keywords if needed.

bus = ['bus']
mrt = ['mrt', 'train']
taxi = ['cab', 'cabbie', 'taxi']
private_hire = ['grab', 'uber', 'lyft', 'bluesg', 'comfortdelgro', 'delgro', 'gojek', 'tada']
car_rental = ['car rent', 'rent car', 'car rental', 'rental car']
bus_char = ['bus rent', 'rent bus', 'bus rental', 'rental bus', 'chartered bus', 'charter bus', 'bus charter', 'bus chartered']
```

After creating the relevant keywords, it will be used to run the categorize function as shown below. The function will take in the cleaned data frame from before, the keyword for that specific business unit, and the name of the business unit it is to be labelled as. The figure below shows how the code will run.

```
# The code below runs the categorize function to group the text accordingly.

bus_df = categorize(dataframe_clean, bus, 'bus')
mrt_df = categorize(dataframe_clean, mrt, 'mrt')
taxi_df = categorize(dataframe_clean, taxi, 'taxi')
private_hire_df = categorize(dataframe_clean, private_hire, 'private hire')
car_rental_df = categorize(dataframe_clean, car_rental, 'car rental')
bus_char_df = categorize(dataframe_clean, bus_char, 'bus charter')
```

After grouping the business units, it will be saved into a single dataframe again, and double checked for duplicates. These duplicates will then be dropped.

```
# The code below join the categorized dataframe into a single dataframe

dataframe_clean_cat = pd.concat([bus_df, mrt_df, taxi_df,private_hire_df,  car_rental_df, bus_char_df], axis = 0)
```

```
# The code below removes any duplicates present

dataframe_clean_cat.drop_duplicates(subset = ['text'], inplace = True)
```

Step 9: Export

These final cleaned data will then be exported, which will be necessary for later analysis. Do edit the file path to specify the specific file to be saved into.

Export the Final Clean Data

```
# The code below will export the dataframes into csv.
# Do state the path file you wish to save the file in.

dataframe_clean.to_csv(r'dataframe_clean.csv', index = False, encoding = 'utf-8-sig')
dataframe_clean_cat.to_csv(r'dataframe_clean_cat.csv', index = False, encoding = 'utf-8-sig')
```

2.3.1.2 Survey

Step 1: Open *System* > *Code* > *Data Cleaning* > *Survey* > *CSISG_Results* > *Data Cleaning.ipynb*

Step 2: Import raw survey data in python

```
df = pd.read_excel (r'../../../Data/Survey/raw_survey_data_402.xlsx')
df
```

0	4	2021-09-01 00:38:08	2021-09-01 00:48:28	anonymous	nan	22 - 25	Female	Full-time Student;	Below $500	Yes	SMU	North	NaN	Yishun	NaN
1	5	2021-09-01 00:44:47	2021-09-01 01:01:51	anonymous	nan	22 - 25	Female	Employed Part-time;Full-time Student;	Below $500	Yes	SMU	West	NaN	NaN	NaN

Step 3: Rename all relevant columns

```
demographic_df = df.iloc[:, np.r_[0:1, 5:17]]
demographic_df.columns = ['id', 'age', 'gender', 'employment_status', 'monthly_income','tertiary_student', 'instituition

demographic_df[['east', 'north', 'north-east', 'west','central']] = demographic_df[['east', 'north', 'north-east', 'west
demographic_df['location']=demographic_df['east'].astype(str)+demographic_df['north'].astype(str)+demographic_df['north-
demographic_df = demographic_df.drop(columns=['east', 'north', 'north-east', 'west','central'])

demographic_df['employment_status'] = demographic_df['employment_status'].str[:-1]
lst_col = 'employment_status'
demographic_df = demographic_df.assign(**{lst_col:demographic_df[lst_col].str.split(';')})
demographic_df.employment_status = demographic_df.employment_status.fillna('')
employment_status_df = pd.DataFrame(demographic_df.employment_status.values.tolist()).add_prefix('employment_status_')
demographic_df = demographic_df.merge(employment_status_df, left_index=True, right_index=True)
demographic_df.drop('employment_status', axis=1, inplace=True)

demographic_df.head()
```

Step 4: Remove unnecessary characters like ‘;’, ‘[’

Step 5: Create new columns when necessary as shown in the example below

```

perception_cdg_df['aware_cdg_cnt'] = perception_cdg_df['aware_cdg'].str.len()
perception_cdg_df['used_cdg_cnt'] = perception_cdg_df['used_cdg'].str.len()
```

2.3.2 Exploratory Data Analysis

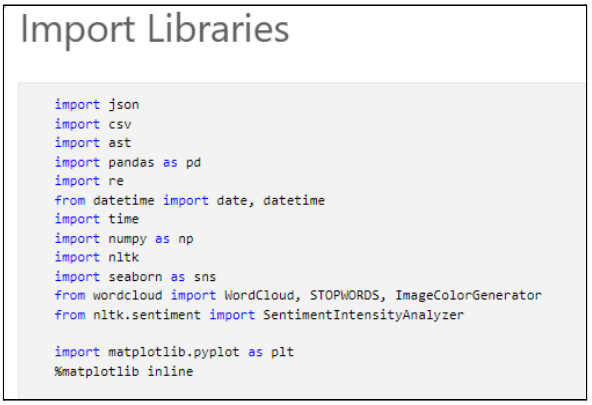
2.3.2.1 Reddit

The exploratory data analysis performed on Reddit data will generally look into the frequency of discussion found in each of the business units. It will also look at the top 10 discussions, and a word cloud of the discussion. This provides an idea of what is concerning Singaporeans based on what was commonly discussed.

Step 1: Open *Reddit EDA_CDG.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Exploratory Data Analysis (EDA)* > *Reddit EDA_CDG.ipynb* to open the reddit EDA file.

Step 2: Import Libraries



Import Libraries

```
import json
import csv
import ast
import pandas as pd
import re
from datetime import date, datetime
import time
import numpy as np
import nltk
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.sentiment import SentimentIntensityAnalyzer

import matplotlib.pyplot as plt
%matplotlib inline
```

Step 3: Separate the Data Frame into Before and After Covid

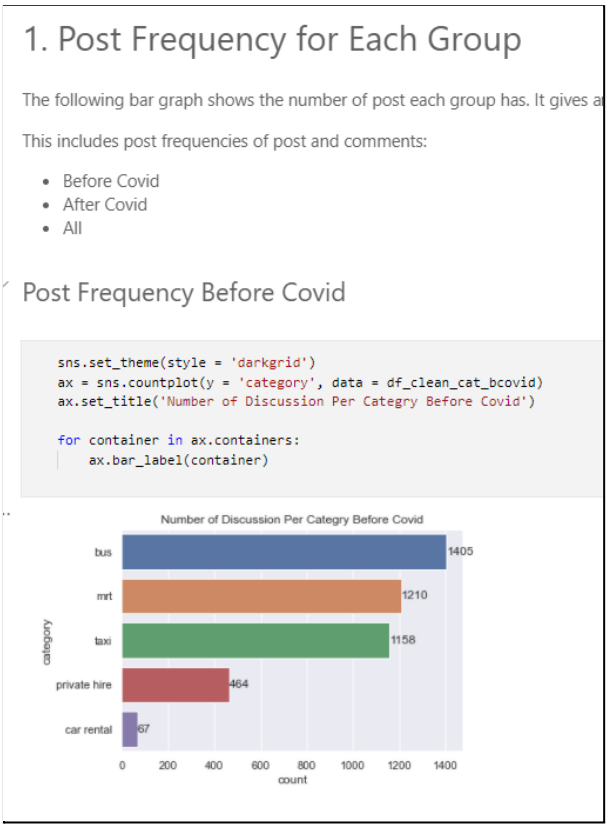
The dataframe will be separated into before and after covid to check if there is a difference in discussion between these time periods. If this is not the focus of the analysis, this step can be skipped and run the subsequent code on the all dataframe.

```
def after_covid(df):

  df_new = pd.DataFrame()
  t = []
  id = []
  d = []
  s = []
  cat = []

  for i in range(len(df['text'])):
    if datetime.strptime(df['date'].iloc[i], "%d/%m/%Y") > datetime.strptime("31/12/2019", "%d/%m/%Y"):
      t.append(df['text'].iloc[i])
      id.append(df['author'].iloc[i])
      d.append(df['date'].iloc[i])
      s.append(df['score'].iloc[i])
      cat.append(df['category'].iloc[i])

  df_new['text'] = t
  df_new['author'] = id
  df_new['date'] = d
  df_new['score'] = s
  df_new['category'] = cat

  return df_new
```

```
df_clean_cat_bcovid =  before_covid(df_clean_cat)
df_clean_cat_acovid =  after_covid(df_clean_cat)
```

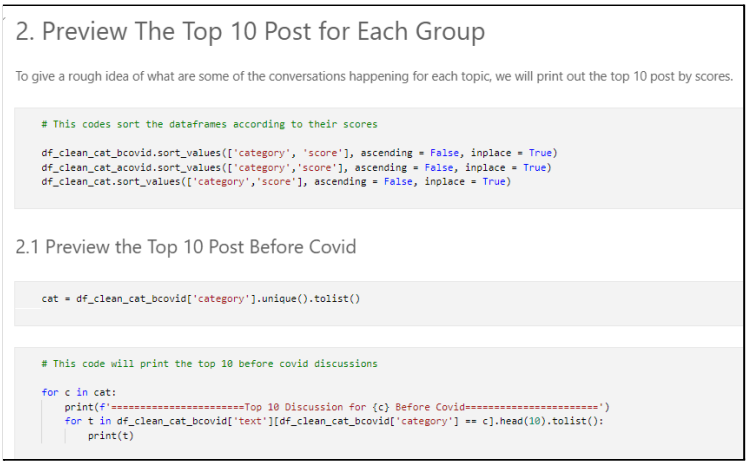
Step 4: Post Frequency

This code will be performed to look at the frequency of discussion in each business unit. If the before and after covid is not the focus of the analysis, the code can be run on just the "all" data frame.



Step 5: Top 10 Post

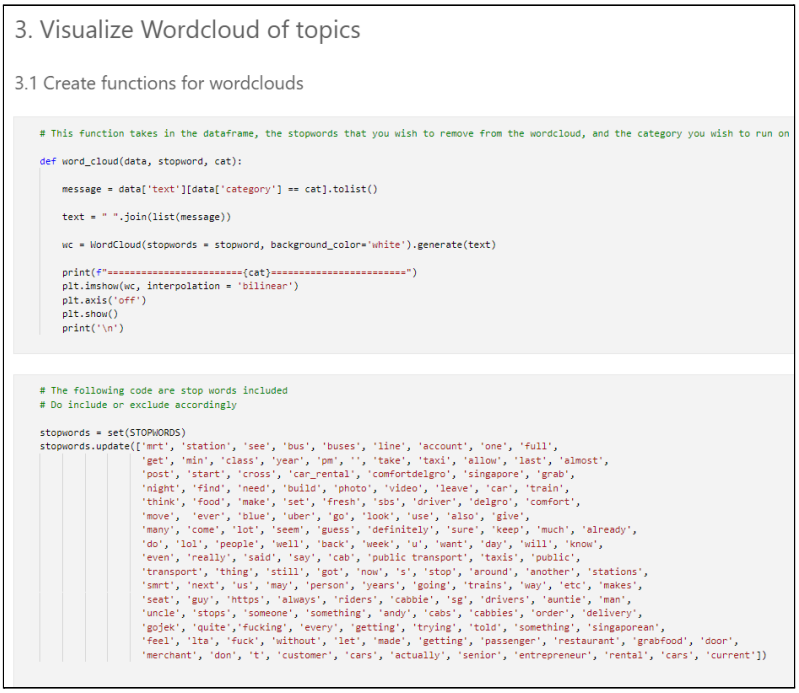
Next, the code will run to look at the top 10 discussions in each business unit to get a general idea of the discussions with the most scores (upvotes).



Step 6: Word Cloud Visualization

To further understand what was discussed in each business unit, word cloud will be used. This will help give information on the common keywords in these discussions and an insight on the concerns of Singaporeans.

The word_cloud() function will take in the dataframe, the list of stopwords, as well as the specific business unit to visualize. Do edit the stop words to include or exclude any relevant stopwords.



3. Visualize Wordcloud of topics

3.1 Create functions for wordclouds

```
# This function takes in the dataframe, the stopwords that you wish to remove from the wordcloud, and the category you wish to run on

def word_cloud(data, stopword, cat):

    message = data['text'][data['category'] == cat].tolist()

    text = " ".join(list(message))

    wc = WordCloud(stopwords = stopword, background_color='white').generate(text)

    print(f"======================={cat}=======================")
    plt.imshow(wc, interpolation = 'bilinear')
    plt.axis('off')
    plt.show()
    print('\n')
```

```
# The following code are stop words included
# Do include or exclude accordingly

stopwords = set(STOPWORDS)
stopwords.update(['mrt', 'station', 'see', 'bus', 'buses', 'line', 'account', 'one', 'full',
                  'get', 'min', 'class', 'year', 'pm', '', 'take', 'taxi', 'allow', 'last', 'almost',
                  'post', 'start', 'cross', 'car_rental', 'comfortdelgro', 'singapore', 'grab',
                  'night', 'find', 'need', 'build', 'photo', 'video', 'leave', 'car', 'train',
                  'think', 'food', 'make', 'set', 'fresh', 'sbs', 'driver', 'delgro', 'comfort',
                  'move',  'ever', 'blue', 'uber', 'go', 'look', 'use', 'also', 'give',
                  'many', 'come', 'lot', 'seem', 'guess', 'definitely', 'sure', 'keep', 'much', 'already',
                  'do', 'lol', 'people', 'well', 'back', 'week', 'u', 'want', 'day', 'will', 'know',
                  'even', 'really', 'said', 'say', 'cab', 'public transport', 'taxis', 'public',
                  'transport', 'thing', 'still', 'got', 'now', 's', 'stop', 'around', 'another', 'stations',
                  'smrt', 'next', 'us', 'may', 'person', 'years', 'going', 'trains', 'way', 'etc', 'makes',
                  'seat', 'guy', 'https', 'always', 'riders', 'cabbie', 'sg', 'drivers', 'auntie', 'man',
                  'uncle', 'stops', 'someone', 'something', 'andy', 'cabs', 'cabbies', 'order', 'delivery',
                  'gojek', 'quite','fucking', 'every', 'getting', 'trying', 'told', 'something', 'singaporean',
                  'feel', 'lta', 'fuck', 'without', 'let', 'made', 'getting', 'passenger', 'restaurant', 'grabfood', 'door',
                  'merchant', 'don', 't', 'customer', 'cars', 'actually', 'senior', 'entrepreneur', 'rental', 'cars', 'current'])
```

Step 7: Export

Lastly, the before and after dataframe will be exported to be used in later analysis. If this is not the focus of the analysis, the exporting of the data can be skipped.

Export Dataframe

Export the data needed for sentiment analysis.

```
df_clean_cat_bcovid.to_csv(r'df_clean_cat_bcovid.csv', index = False, encoding = 'utf-8-sig')
df_clean_cat_acovid.to_csv(r'df_clean_cat_acovid.csv', index = False, encoding = 'utf-8-sig')
```

3.0 Data Analysis

3.1 Factor Analysis

Step 1: *Open System > Code > Data Analysis > Factor Analysis + Clustering > survey_data_402 > Factor Analysis.ipynb*

Step 2: Import packages

```
In [1]:
#Import Packages

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Step 3: Load survey csv file "cleaned_survey_402.csv"

```
In [2]:
df = pd.read_csv('cleaned_survey_402.csv')
df
```

Out[2]:

	id	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_m
0	4	22 - 25	Female	Below $500	Yes	SMU	North	Yishun	Full-time Student	NaN	['MRT', 'Bu
1	5	22 - 25	Female	Below $500	Yes	SMU	West	Bukit Batok	Full-time Student	Employed Part-time	['Bus', 'MRT', 'Tax 'Ride-hailing service

Step 4: Clean the dataset

1. Drop columns with open ended answers (as those will not be used for Clustering analysis)
2. Replace the "&" signs from the data
    a. Those signs are come about from Microsoft Survey's concatenating of selected answers from multi-select answers, eg: if respondent choose "option A" and "option B" for a question, the result recorded would be "option A&option B".
    b. Remove the ";" signs
    c. Drop rows of respondents' who are not tertiary students (as they are not part of our target audience)

Data Cleaning

```
In [3]:
# drop open ended columns & id
columns = ['difficulties_ride_hailing','reason_least_preferred_mot', 'circ_least_preferred_mot', 'occasion_chartered_bus', '
df.drop(columns, axis=1, inplace = True )
```

```
In [4]:
# replace [ & ] with empty
col_name = ['frequently_used_mot', 'preferred_mot', 'factors_mot_ranking', 'reason_bus', 'prevent_bus', 'reason_mrt',
            'mrt_lines', 'prevent_mrt', 'way_hail', 'reason_taxi', 'preference_rank_taxi', 'prevent_taxi', 'reason_ride_hail
           'ride_hailing_type', 'preference_rank_ride_hailing', 'prevent_ride_hailing',
           'preference_rank_rental_car', 'reason_rental_car_company', 'reason_rental_car', 'type_rental_car', 'biz_model_ren
           'biz_model_reason_rental', 'prevent_rental', 'aware_cdg', 'used_cdg', 'incentives_cdg']

for i in col_name:
    df[i] = df[i].str.replace(r"'", '')
    df[i] = df[i].str.replace(r'[', '')
    df[i] = df[i].str.replace(r']', '')
```

In [5]:
```
lst_col = ['reason_bus', 'reason_mrt', 'reason_taxi', 'reason_ride_hailing', 'reason_rental_car_company', 'reason_rental_car
for i in lst_col:
    df = df.assign(**{i:df[i].str.split(';')})
```

In [7]:
```
# drop rows if not tertiary_student
df.drop(df.index[df['tertiary_student'] == 'No'], inplace = True)
# drop rows if missing tertiary_student
df.dropna(subset=['tertiary_student'], inplace = True)
```

Step 5: Check for null values and fill them up

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
age                          0
gender                       0
monthly_income               0
tertiary_student             0
instituition                 0
region                       0
location                     0
type_of_student             20
type_of_employment         286
frequently_used_mot          0
frequency_mot                0
```

In [11]:
```
# fill missing values
df['frequency_rental'].fillna('0', inplace = True)
df['duration_rental'].fillna('0', inplace = True)
df['difficulties_rental_car'].fillna('Have not rent before', inplace = True)
df['convenient_rental_car'].fillna('Have not rent before', inplace = True)
df['affordable_rental_car'].fillna('Have not rent before', inplace = True)
df['reasonable_rental_car'].fillna('Have not rent before', inplace = True)
df['max_price_rental_car'].fillna('$3.01 - $5', inplace = True)
df['accessibility_cdg'].fillna('Somewhat satisfied', inplace = True)
df['cs_cdg'].fillna('Somewhat satisfied', inplace = True)
df['cost_cdg'].fillna('Neutral', inplace = True)
df['comfort_cdg'].fillna('Somewhat satisfied', inplace = True)
df['reliability_cdg'].fillna('Somewhat satisfied', inplace = True)
df['info_cdg'].fillna('Somewhat satisfied', inplace = True)
df['overall_rating_cdg'].fillna('Somewhat satisfied', inplace = True)
df['type_of_student'].fillna('Full-time Student', inplace = True)
df['type_of_employment'].fillna('Not employed', inplace = True)
```

Step 6: Extract dataframe as new csv named “cleaned_334.csv”

In [13]:
```
df.to_csv('cleaned_334.csv', index=False)
```

Step 7: Run newly extracted dataset to further clean dataset

Further cleaning in excel

In [14]:
```
df = pd.read_csv('cleaned_334_edited.csv')
df.head()
```

Out[14]:

	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_mot	frequency_mot	pre
0	22 - 25	Female	Below $500	Yes	SMU	North	Yishun	Full-time Student	Not employed	MRT, Bus	Once a week	se Bu
1	22 - 25	Female	Below $500	Yes	SMU	West	Bukit Batok	Full-time Student	Employed Part-time	Bus, MRT, Taxi, Ride-hailing services	3 - 4 days a week	se Bu

Step 8: Encode data with Label Encoder → to encode data with multiple answers, so each categorical value will have a label to it.

a. Eg: north, east, west might become 1,2,3

Label Encoding

In [16]:
```
def transform_agree_disagree(x):
    if x == 'Strongly Disagree':
        return -2
    elif x == 'Disagree':
        return -1
    elif x == 'Neutral':
        return 0
    elif x == 'Agree':
        return 1
    else:
        return 2
```

In [17]:
```
def transform_yes_no(x):
    if x == 'Yes':
        return 1
    elif x == 'No':
        return 0
    else:
        return -1
```

In [18]:
```
bin_features = ['own_driving_license', 'rent_before']
train_bin = df[bin_features]
train_bin['own_driving_license'] = df['own_driving_license'].apply(transform_yes_no)
train_bin['rent_before'] = df['rent_before'].apply(transform_yes_no)
train_bin = train_bin.astype('int64')
train_bin.head()
```

Out[18]:

	own_driving_license	rent_before
0	1	0
1	1	1
2	1	0
3	0	0
4	0	0

In [19]:
```
agree_disagree_col = ['affordable_bus', 'accesible_bus', 'safe_bus', 'affordable_mrt', 'accessible_mrt', 'waiting_mrt', 'saf
                      'affordable_taxi', 'accessible_taxi', 'safe_taxi', 'customer_service_taxi',
                       'affordable_ride_hailing', 'safe_ride_hailing', 'customer_service_ride_hailing']
train_bin_2 = df[agree_disagree_col]
for col in agree_disagree_col:
    train_bin_2[col] = df[col].apply(transform_agree_disagree)

train_bin_2.head()
```

Out[19]:

	affordable_bus	accesible_bus	safe_bus	affordable_mrt	accessible_mrt	waiting_mrt	safe_mrt	affordable_taxi	accessible_taxi	safe_taxi	customer_service_t
0	1	2	2	1	2	2	1	0	0	1	
1	1	1	1	1	1	1	1	-1	-1	1	
2	1	1	1	1	1	1	1	-1	1	2	
3	1	1	1	1	1	1	1	1	1	1	
4	0	0	0	0	0	0	0	0	0	0	

Step 9: Encode nominal features

In [20]:
```
#encode nominal features
from sklearn.preprocessing import LabelEncoder
train_final = df.copy()

cat_features=[x for x in train_final.columns if train_final[x].dtype=="object"]

le=LabelEncoder()

for col in cat_features:
    if col in train_final.columns:
        i = train_final.columns.get_loc(col)
        train_final.iloc[:,i] = train_final.apply(lambda i:le.fit_transform(i.astype(str)), axis=0, result_type='expand')
```

In [21]:
```
train_final
```

Out[21]:

	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_mot	frequency_mot
0	1	0	4	0	17	2	37	0	2	23	3
1	1	0	4	0	17	5	3	0	1	14	0

Step 10: Drop unwanted columns

In [22]:
```
col_drop = ['own_driving_license', 'rent_before', 'affordable_bus', 'accesible_bus', 'safe_bus', 'affordable_mrt', 'accessib
                    'affordable_taxi', 'accessible_taxi', 'safe_taxi', 'customer_service_taxi',
                     'affordable_ride_hailing', 'safe_ride_hailing', 'customer_service_ride_hailing' ]
train_final = train_final.drop(columns=col_drop)
train_final.head()
```

Out[22]:

	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_mot	frequency_mot	pref
0	1	0	4	0	17	2	37	0	2	23	3	
1	1	0	4	0	17	5	3	0	1	14	0	
2	1	1	4	0	16	5	3	0	2	5	2	
3	0	0	4	0	17	1	30	0	2	1	2	
4	1	0	5	0	15	1	21	0	2	23	0	

Step 11: Combine datasets

In [23]:
```
train_final = pd.concat([train_final, train_bin, train_bin_2], axis = 1)
train_final.head()
```

Out[23]:

	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_mot	frequency_mot	pref
0	1	0	4	0	17	2	37	0	2	23	3	
1	1	0	4	0	17	5	3	0	1	14	0	
2	1	1	4	0	16	5	3	0	2	5	2	
3	0	0	4	0	17	1	30	0	2	1	2	
4	1	0	5	0	15	1	21	0	2	23	0	

Step 12: Split the dataset into two dataframes, perceptions and patterns. This splitting was for us to also split up our analysis later on as we are diving into the individual factors.

```
#Travel Patterns/Behaviour
travel_patterns = train_final[demo_cols + patts_cols]

#Travel Perceptions & Preferences
travel_perceptions = train_final[demo_cols + percept_cols]
```

Get the dataframes

In [28]:

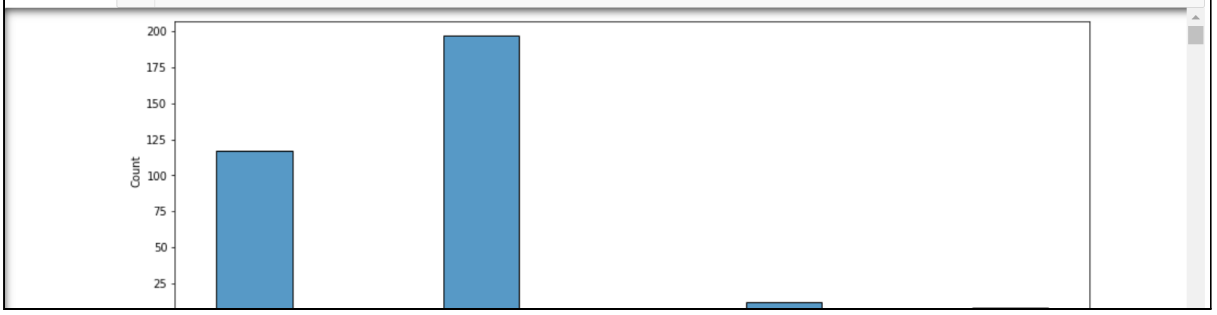
```
#Cluster into different groups of data
demo_cols = ['age', 'gender', 'monthly_income', 'instituition', 'region', 'location', 'type_of_student', 'type_of_employment

patts_cols = ['frequently_used_mot', 'frequency_mot', 'factors_mot_ranking', 'overall_avg_spending', 'research', 'comfortabl
              'frequently_used_mot_cnt', 'frequency_bus', 'waiting_time_bus', 'chartered_bus', 'reason_bus_travel_to_mrt_stati
              'reason_bus_for_work_school', 'frequency_mrt', 'spend_mrt', 'mrt_lines',
              'mrt_lines_cnt', 'reason_mrt_for_work_school', 'reason_mrt_recreation_destination', 'reason_mrt_to_bus_stop',  '
              'preference_rank_ride_hailing','own_driving_license', 'rent_before', 'frequency_rental', 'duration_rental']

percept_cols = ['preferred_mot', 'factors_mot_ranking','most_preferred_mot','least_preferred_mot', 'affordable_bus','accesib
                'safe_bus','prevent_bus','affordable_mrt','accessible_mrt','waiting_mrt','safe_mrt','prevent_mrt', 'reason_t
                 'reason_taxi_convenient', 'reason_taxi_cheaper', 'reason_taxi_comfort_of_privacy',
                'preference_rank_taxi','affordable_taxi','accessible_taxi','safe_taxi','customer_service_taxi','prevent_taxi
                'most_preferred_taxi','least_preferred_taxi','reason_ride_hailing_promotion', 'reason_ride_hailing_easy_book
                'preference_rank_ride_hailing','affordable_ride_hailing',
                'safe_ride_hailing','customer_service_ride_hailing','prevent_ride_hailing','used_ride_hailing_svc_cnt',
                'most_preferred_ride_hailing','least_preferred_ride_hailing','preference_rank_rental_car',
                'reason_chosen_rental_car_company_convenient', 'reason_chosen_rental_car_company_cheap', 'reason_chosen_rent
                'reason_rental_car_convenience', 'reason_rental_car_car_too_costly', 'reason_rental_car_occasions', 'reason_
                'difficulties_rental_car','convenient_rental_car','affordable_rental_car','reasonable_rental_car',
                'type_rental_car','max_price_rental_car','biz_model_rental','biz_model_reason_rental','biz_model_important_r
                'prevent_rental','type_rental_car_heard_cnt']
```

Step 13: Conducting EDA

EDA

In [30]:

```
#checking of ditribuition
all_features = travel_patterns.columns
for col in all_features:
    plt.subplots(figsize = (15,5))
    sns.histplot(data = travel_patterns, x=travel_patterns[col])
    plt.show()
```



Step 14: Check skewness

a. If the skewness is between -0.5 and 0.5, the data are fairly symmetrical

b. If the skewness is between -1 and – 0.5 or between 0.5 and 1, the data are moderately skewed

c. If the skewness is less than -1 or greater than 1, the data are highly skewed

```
In [31]: travel_patterns.skew()

Out[31]: age                          0.856712
         gender                       1.091091
         monthly_income              -0.748466
         instituition                -0.152224
         region                       0.241637
         location                     0.210914
         type_of_student              4.592400
         type_of_employment          -2.548440
         frequently_used_mot          0.567429
         frequency_mot                0.646453
         factors_mot_ranking          0.096528
         overall_avg_spending         0.239636
```

```
In [32]: travel_perceptions.skew()

Out[32]: age                          0.856712
         gender                       1.091091
         monthly_income              -0.748466
         instituition                -0.152224
         region                       0.241637
         location                     0.210914
         type_of_student              4.592400
         type_of_employment          -2.548440
         preferred_mot                0.462095
         factors_mot_ranking          0.096528
         most_preferred_mot           0.035732
         least_preferred_mot          1.506505
```

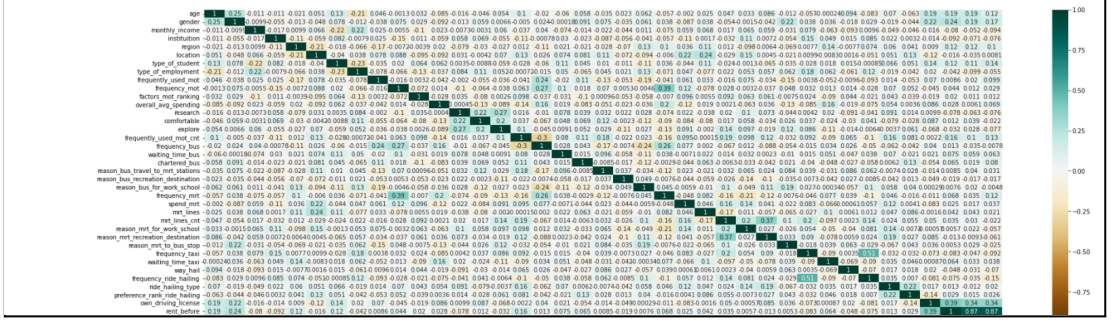
Conclusion: The skewness of respondent data because they are considered the insights and therefore it is not needed for us to address this.

Further conclusion: No need to transform observed variables because in general they do not follow normal distribution. In our survey data, no distribution to follow and no outlier in terms of age, gender, income which transformation can also help to address.

Step 15: Check correlation of the columns

Correlation HeatMap

```
In [34]: #Travel Patterns corr heatmap
         plt.figure(figsize=(30,10))
         sns.heatmap(travel_patterns.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x1b01e79f320>
```



```
In [35]: #Travel Perceptions/Preferences corr heatmap
         plt.figure(figsize=(40,20))
         sns.heatmap(travel_perceptions.corr(), vmin=-1, vmax=1, annot=True, cmap='BrBG')

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1b01f4f67f0>
```

Step 16: Install relevant packages - Factor Analyzer

FACTOR ANALYSIS

In [37]:
```
# pip install factor_analyzer
```

In [38]:
```
#Import and Install packages

from factor_analyzer import FactorAnalyzer
```

Step 17: We will be using three different tests to check the suitability of the datasets to undergo a factor analysis.

a. Adequacy Test - Evaluating the "factorability" of the dataset

b. Barlett's Test - Checks whether or not the observed variables intercorrelated at all using the observed correlation matrix against the identity matrix. If the test found statistically insignificant, you should not employ a factor analysis.

c. Kaiser-Meyer-Olkin (KMO) Test - Measures the suitability of data for factor analysis. Determines the adequacy for each observed variable and for the complete model. KMO estimates the proportion of variance among all the observed variables.

In [40]:
```
#Bartlett's Test - Travel perceptions/preferences df

from factor_analyzer.factor_analyzer import calculate_bartlett_sphericity
percept_chi_square_value, percept_p_value = calculate_bartlett_sphericity(travel_perceptions)
percept_chi_square_value, percept_p_value
```

Out[40]: (9548.947278167601, 0.0)

In [41]:
```
#KMO Test - Travel patterns df

from factor_analyzer.factor_analyzer import calculate_kmo
patts_kmo_all, patts_kmo_model = calculate_kmo(travel_patterns)

patts_kmo_model
```

Out[41]: 0.5865026760594407

In [42]:
```
#KMO Test - Travel perceptions/preferences df

from factor_analyzer.factor_analyzer import calculate_kmo
percept_kmo_all, percept_kmo_model = calculate_kmo(travel_perceptions)

percept_kmo_model
```

Out[42]: 0.7306047717036648

Step 18: In fact, Factor Analysis can only be done on the *Travel Perceptions/Preferences df*.

a. Bartlett's Results: p-value = 0.0 Hence, this was statistically significant, indicating that the observed correlation matrix is not an identity matrix. KMO Results: Value of KMO is > 0.6: 0.7855376109339436. Hence, we may proceed with the planned factor analysis.

Step 19: Conduct factor analysis on travel perceptions df and choosing eigenvalues

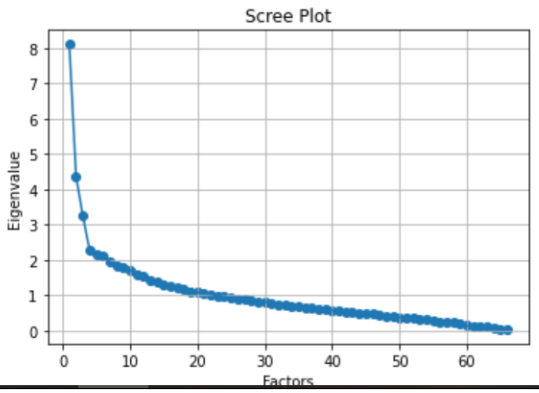
FACTOR ANALYSIS ON TRAVEL PERCEPTIONS/PREFERENCES df

```
#Choosing the number of factors

#Create factor analysis object and perform factor analysis
fa_percept = FactorAnalyzer(55, rotation=None)
#fa.set_params(rotation=None)
fa_percept.fit(travel_perceptions)

#Check Eigenvalues
ev_percept, v_percept = fa_percept.get_eigenvalues()
ev_percept
```

Step 20: Conduct a scree plot to identify the optimal number of factors from the dataset

In [44]:

```
# Create scree plot using matplotlib

plt.scatter(range(1,travel_perceptions.shape[1]+1),ev_percept)
plt.plot(range(1,travel_perceptions.shape[1]+1),ev_percept)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```



Step 21: Export loading scores excel file to see the scores of the various columns to each identified factor, hence allowing us to name each factor.

a. Identify the top 3-5 variables (as shown in column A) with the highest score for each factor (as shown in row 1).

In [45]:

```
# Create factor analysis object and perform factor analysis
fa_percept_varimax = FactorAnalyzer(20, rotation="varimax")
#fa.set_params(rotation="varimax")
fa_percept_varimax.fit(travel_perceptions)
fa_percept_varimax.loadings_

loadings_score_percept = pd.DataFrame(fa_percept_varimax.loadings_,index=travel_perceptions.columns)
loadings_score_percept.to_csv("travel_perceptions_loading_scores.csv")
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
2	age	0.147348	-0.12284	0.002884	-0.05911	-0.08835	0.064586	0.073677	-0.1787	-0.11847	0.023541	0.01918	-0.18762	0.04262	0.261832	0.062455	0.100707	0.124745	-0.18338
3	gender	0.213986	-0.07333	-0.04927	-0.02815	-0.10139	0.086186	0.088082	0.018871	-0.05221	-0.00901	0.120873	-0.14967	0.117421	0.028425	-0.00066	-0.02133	0.338894	-0.09027
4	monthly_i	-0.0801	-0.08198	0.030037	-0.0361	-0.01969	-0.09779	-0.07364	-0.15993	-0.09985	-0.10143	0.016646	-0.06802	-0.02789	-0.40708	-0.04473	0.002423	0.044874	0.069421
5	instituitior	-0.08913	0.063538	0.015323	-0.00822	-0.01226	0.085329	-0.06703	0.144599	0.011949	0.164637	-0.03559	0.015631	-0.05815	0.053589	-0.09895	0.000176	0.059714	0.043363
6	region	0.108477	-0.02476	0.021939	-0.01038	0.058793	-0.07616	-0.13882	0.051105	0.001474	-0.18669	-0.13349	0.160297	0.2345	0.043283	-0.11253	-0.05038	-0.01564	-0.00514
7	location	-0.00973	-0.09948	0.043005	0.169	-0.08794	0.039663	0.258409	-0.0744	-0.03551	-0.04313	0.226005	-0.06367	-0.30858	-0.13723	0.048646	0.107068	-0.13892	-0.08764
8	type_of_s	0.103036	0.110224	-0.07912	0.050978	0.039152	0.014257	0.10389	0.038544	-0.05214	-0.07802	-0.07219	0.011773	-0.13469	0.487944	-0.04744	-0.13731	0.098885	-0.02599
9	type_of_e	-0.01421	-0.0482	-0.0435	-0.03052	-0.03085	0.001585	0.064897	0.024161	0.119366	0.038978	-0.07244	0.01839	-0.0067	-0.49897	0.074888	-0.0853	0.042355	0.027797
10	preferred	0.010864	0.02589	0.973588	0.003764	0.027278	0.026139	0.07954	0.021621	-0.02082	0.022863	0.006854	-0.015	0.01084	0.005062	-0.00401	0.041199	0.004973	-0.04396
11	factors_m	0.018926	0.04979	0.03515	-0.03704	0.028041	-0.08104	0.033877	-0.03517	-0.03774	0.11905	0.125479	0.069661	0.108198	0.253055	0.042594	-0.01541	0.101136	0.051975
12	most_pref	0.011197	0.050347	0.965711	-0.0144	0.003989	0.022135	0.055055	0.036183	-0.0326	0.03838	-0.03239	-0.00856	0.030277	-0.01015	-0.03403	0.024781	-0.0098	-0.03237
13	least_pref	0.296553	0.036619	0.001259	0.031164	0.107292	-0.09476	0.104046	-0.01943	-0.00652	-0.08334	-0.01391	0.021461	-0.04276	-0.02514	0.071612	-0.0787	-0.03009	0.031839
14	affordable	0.068673	0.726931	-0.05873	0.063147	-0.04463	0.096769	0.055443	0.024515	-0.10225	-0.13593	-0.02707	0.156492	-0.05617	0.073591	0.082545	0.021112	0.082228	-0.2622
15	accesible_	-0.00818	0.595282	-0.04072	-0.02458	-0.01792	0.12897	0.100854	-0.09765	-0.00128	0.144201	0.030399	-0.00795	0.036719	0.124511	-0.06874	-0.04293	-0.05296	0.051699
16	safe_bus	-0.02373	0.502553	0.000368	-0.02818	-0.02695	0.526076	0.005633	-0.0594	0.044157	0.089333	-0.0053	-0.05387	-0.00946	-0.03341	-0.06469	-0.10081	-0.1206	-0.10817

Step 22: Using the results from the loading score sheet, compile the identified columns into the various and name them.

a. For example, “factor 1” has the columns, affordable_bus and affordable_mrt as the top 2 factors with the highest loading scores. Therefore, factor 1 will be renamed as “affordability” and the data frame is created as such.

In [47]:
```
#!pip install pingouin
import pingouin as pg

#Create the factors

convenience = travel_perceptions[['convenient_rental_car', 'difficulties_rental_car', 'reason_chosen_rental_car_company_conv
affordability = travel_perceptions[['affordable_bus', 'affordable_mrt']]
safety = travel_perceptions[['safe_taxi', 'safe_ride_hailing', 'safe_bus', 'safe_mrt']]
comfort = travel_perceptions[['reason_taxi_comfort_of_privacy', 'reason_ride_hailing_ comfort_of_privacy']]
promotion = travel_perceptions[['reason_ride_hailing_promotion', 'reason_taxi_promotion']]
customer_service = travel_perceptions[['customer_service_taxi','customer_service_ride_hailing']]
accessible = travel_perceptions[['reason_ride_hailing_accessible', 'reason_taxi_more_accessible', 'reason_ride_hailing_easy_
```

Step 23: Check Cronbach alpha score to further confirm if these factors are suitable to be used in the following analyses.

a. Those with the scores above 0.6 are deemed as suitable.

b. Even though we choose 20 as the optimal number of factors, some of the factors have similar and overlap variables and cronbach alpha not good enough (not more than 0.6) → hence we have come to the conclusion of having 6 factors of travel perception

```
#Get cronbach alpha
#accessibility_alpha = pg.cronbach_alpha(accessibility)
convenience_alpha = pg.cronbach_alpha(convenience)
affordability_alpha = pg.cronbach_alpha(affordability)
safety_alpha = pg.cronbach_alpha(safety)
comfort_alpha = pg.cronbach_alpha(comfort)
promotion_alpha = pg.cronbach_alpha(promotion)
customer_service_alpha = pg.cronbach_alpha(customer_service)
accessible_alpha = pg.cronbach_alpha(accessible)


print("Convenience: " + str(convenience_alpha))
print("Affordability: " + str(affordability_alpha))
print("Safety: " + str(safety_alpha))
print("Comfort: " + str(comfort_alpha))
print("Promotion: " + str(promotion_alpha))
print("Customer Service: " + str(customer_service_alpha))
print("Accessible: " + str(accessible_alpha))
```

```
Convenience: (0.9310080616309302, array([0.919, 0.942]))
Affordability: (0.75849421948858, array([0.701, 0.805]))
Safety: (0.7368714667359193, array([0.688, 0.78 ]))
Comfort: (0.7277015437392782, array([0.662, 0.78 ]))
Promotion: (0.6063669606366966, array([0.512, 0.683]))
```

Step 24: Prepare and create the new perceptions dataset for clustering analysis, by combining the identified factors along with the demographic factors.

a. Export dataset to csv named "*perception_dataset.csv*"

PREPARE TRAVEL PERCEPTION DATASET FOR CLUSTERING

In [53]:
```
perception_dataset= pd.concat([train_final[demo_cols], convenience, affordability,safety,comfort,promotion, customer_service

perception_dataset.to_csv('perception_dataset.csv', index=False)

```

In [54]:
```
perception_dataset.head()
```

Out[54]:

	age	gender	monthly_income	instituition	region	location	type_of_student	type_of_employment	convenient_rental_car	difficulties_rental_car	reason_chose
0	1	0	4	17	2	37	0	2	0	0	
1	1	0	4	17	5	3	0	1	1	1	
2	1	1	4	16	5	3	0	2	0	0	
3	0	0	4	17	1	30	0	2	0	0	
4	1	0	5	15	1	21	0	2	0	0	

3.2 Text Analysis

This section entails a combination of Sentiment and Topic Modelling Analysis.

3.2.1 Sentiment Analysis

3.2.1.1 Reddit

Step 1: Open *reddit_sentiment_CDG.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Sentiment Analysis* > *reddit_sentiment_CDG.ipynb* to open the reddit sentiment analysis file.

Step 2: Import Libraries

```
import json
import csv
import ast
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
import re
from datetime import date, datetime
import time
import numpy as np
import nltk
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.sentiment import SentimentIntensityAnalyzer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize, word_tokenize
import statistics
sentiment = SentimentIntensityAnalyzer()

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline
```

Step 3: Load Files

Import the before and after covid data previously exported in the Reddit EDA script. If that is not the focus of the analysis, the dataframe_clean_cat.csv can be loaded instead, previously exported in the Reddit Data Cleaning script, which does not separate the data before and after covid. A sample prelabelled twitter dataset will also be loaded for the purpose of training the LSTM (Long Short Term Memory) model for sentiment analysis. This twitter dataset can be found under *System* > *Data* > *Scrapping* > *Reddit* > *Twitter_Data.csv.*

Import Files

```
# These are the data loaded for sentiment analysis
df_clean_cat_bcovid = pd.read_csv(r'df_clean_cat_bcovid.csv')
df_clean_cat_acovid = pd.read_csv(r'df_clean_cat_acovid.csv')

# This is the training data for LSTM model
df = pd.read_csv('Twitter_Data.csv')
```

```
df_clean_cat_all = pd.concat([df_clean_cat_bcovid,df_clean_cat_acovid], axis = 0)
```

Step 4: Model Training

Before executing the sentiment analysis, the model has to be trained. The code below shows a glimpse of the code used to perform for model training. The model training process consists of cleaning of twitter data first, tokenizing the text, executing the data training, and checking the accuracy, precision, and F1 Scores.

```
max_words = 5000
max_len=50

def tokenize_pad_sequences(text):
    '''
    This function tokenize the input text into sequnences of intergers and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=max_len)
    # return sequences
    return X, tokenizer

print('Before Tokenization & Padding \n', df['clean_text'][0])
X, tokenizer = tokenize_pad_sequences(df['clean_text'])
print('After Tokenization & Padding \n', X[0])
```

Execute Data Training

```
# Convert categorical variable into dummy/indicator variables.
y = pd.get_dummies(df['category'])
# Train and Test split
X_train, X_test,y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 42)
# Extracting validation set from the train set
valid_size=1000
X_valid, y_valid = X_train[-valid_size:], y_train[-valid_size:]
X_test, y_test = X_train[:-valid_size], y_train[:-valid_size]

print('Train Set ->', X_train.shape, y_train.shape)
print('Validation Set ->', X_valid.shape, y_valid.shape)
print('Test Set ->', X_test.shape, y_test.shape)
```

The code below runs the model and calculates the relevant accuracies and precisions scores. The definition and calculations of each of these scores are as follows:

- Precision = True Positive/(True Positive + False Positive)
    - Ratio of correctly predicted positive observations to the total predicted positive observations.
- Recall = True Positive/(True Positive + False Negative)
    - Ratio of correctly predicted positive observations to all the actual positive observations.
- Accuracy = (True Positive + True Negative)/(True Positive + False Positive + False Negative + True Negative)
    - A ratio of correctly predicted observation to the total observations.
- F1 Score = 2*((Precision*Recall)/(Precision + Recall))
    - Weighted average of Precision and Recall. Needed when you want to seek balance between Precision and Recall.

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout
from keras.metrics import Precision, Recall

vocab_size = 5000
embedding_size = 32

# Build model
model3 = Sequential()
model3.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model3.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model3.add(MaxPooling1D(pool_size=2))
model3.add(Bidirectional(LSTM(32)))
model3.add(Dropout(0.4))
model3.add(Dense(3, activation='softmax'))

print(model3.summary())

# Compile model
model3.compile(loss='categorical_crossentropy', optimizer='adam',
               metrics=['accuracy', Precision(), Recall()])

# Train model
num_epochs = 1
batch_size = 32
history3 = model3.fit(X_train, y_train,
                      validation_data=(X_valid, y_valid),
                      batch_size=batch_size, epochs=num_epochs)

# Evaluate model on the test set
loss, accuracy, precision, recall = model3.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('CNN + LSTM Accuracy  : {:.2f}'.format(100 * accuracy), '%')
print('CNN + LSTM Precision : {:.2f}'.format(100 * precision), '%')
print('CNN + LSTM Recall    : {:.2f}'.format(100 * recall), '%')
print('CNN + LSTM F1 Score  : {:.2f}'.format(f1_score(precision, recall)), '%')
```

Step 5: Create and Execute Sentiment Analysis

The code below will create the function to run the sentiment analysis and create a new column for the labelled sentiment. The function will then be performed on the dataframes. If the before and after covid is not the focus of the analysis, these two data frames can be skipped and run only on the all dataframe.

```python
def predict_class(text):
    '''Function to predict sentiment class of the passed text'''

    sentiment_classes = ['Negative', 'Neutral', 'Positive']
    max_len=50

    # Transforms text to a sequence of integers using a tokenizer object
    xt = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    xt = pad_sequences(xt, padding='post', maxlen=max_len)
    # Do the prediction using the loaded model
    yt = model3.predict(xt).argmax(axis=1)
    # Print the predicted sentiment
    # Print('The predicted sentiment is', sentiment_classes[yt[0]])
    return sentiment_classes[yt[0]]
```

```python
def sentiment_text(df):
    sent = []

    for t in df['text']:
        sent.append(predict_class([t]))

    df['sentiment'] = sent
```

Execute LSTM Sentiment Analysis

Sentiment analysis will be run on bcovid, acovid, and all dataframes. This also applies

```python
sentiment_text(df_clean_cat_bcovid)
sentiment_text(df_clean_cat_acovid)
sentiment_text(df_clean_cat_all)
```

Step 6: Visualize Sentiment Frequency

Next, the frequency of sentiments in each business unit will be visualised to get a rough gauge of the general sentiment of the discussions. The figure below shows an example of the code run on one of the business units. The before and after covid codes can be skipped if that is not the focus of the analysis.

Visualize Sentiment Frequncy

Bus

```python
# Before Covid

sns.set_theme(style = 'darkgrid')
ax = sns.countplot(x = 'sentiment', data = df_clean_cat_bcovid[df_clean_cat_bcovid['category'] == 'bus'],
                   order = ['Negative', 'Neutral', 'Positive'], palette = ['#ad5353', '#5985b5', '#87c993'])
ax.set_title('Sentiments Before Covid')
sns.color_palette('pastel')

for container in ax.containers:
    ax.bar_label(container)
```

```python
# After Covid

sns.set_theme(style = 'darkgrid')
ax = sns.countplot(x = 'sentiment', data = df_clean_cat_acovid[df_clean_cat_acovid['category'] == 'bus'],
                   order = ['Negative', 'Neutral', 'Positive'], palette = ['#ad5353', '#5985b5', '#87c993'])
ax.set_title('Sentiments After Covid')

for container in ax.containers:
    ax.bar_label(container)
```

```python
# All

sns.set_theme(style = 'darkgrid')
ax = sns.countplot(x = 'sentiment', data = df_clean_cat_all[df_clean_cat_all['category'] == 'bus'],
                   order = ['Negative', 'Neutral', 'Positive'], palette = ['#ad5353', '#5985b5', '#87c993'])
ax.set_title('Sentiments After Covid')
```

Step 7: Preview Top Negatives Discussions Visualize Wordcloud

Next, the top negative discussion will be previewed to get a general idea of what are the topic of discussions that lead to the negative sentiments to discover possible gaps in the transportation. The wordcloud of the negative discussions also help to discover these possible gaps. The before and after covid codes can be skipped if that is not the focus of the analysis.

Execute Functions to Preview Discussions

Bus

```
# Before Covid

t = df_clean_cat_bcovid[df_clean_cat_bcovid['sentiment']=='Negative']
p = t[t['category'] == 'bus']

print('==========================Bus Before Covid Title==========================')
for index,row in p[:5].iterrows():
    print(row['text'])
wc_neg(df_clean_cat_bcovid, stopwords, 'bus')
```

```
# After Covid

t = df_clean_cat_acovid[df_clean_cat_acovid['sentiment']=='Negative']
p = t[t['category'] == 'bus']

print('==========================Bus After Covid Title==========================')
for index,row in p[:5].iterrows():
    print(row['text'])
wc_neg(df_clean_cat_acovid, stopwords, 'bus')
```

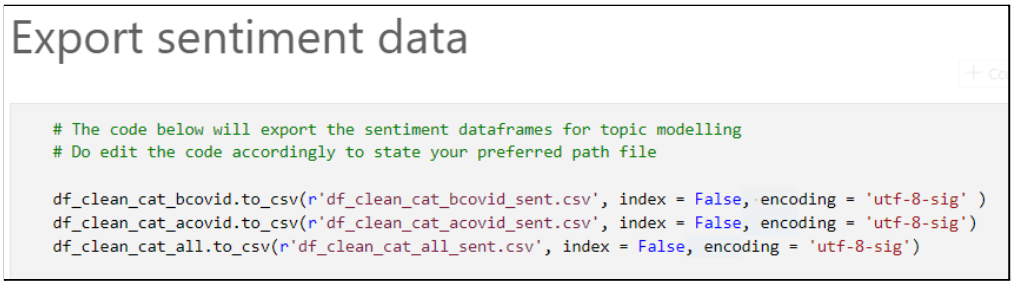
```
# All

t = df_clean_cat_all[df_clean_cat_all['sentiment']=='Negative']
p = t[t['category'] == 'bus']

print('==========================Bus Title==========================')
for index,row in p[:5].iterrows():
    print(row['text'])
wc_neg(df_clean_cat_all, stopwords, 'bus')
```

Step 8: Export the Sentiment Data

Export the sentiment data which will be used for topic modelling. Do edit the file path to the correct file path.



Export sentiment data

```
# The code below will export the sentiment dataframes for topic modelling
# Do edit the code accordingly to state your preferred path file

df_clean_cat_bcovid.to_csv(r'df_clean_cat_bcovid_sent.csv', index = False, encoding = 'utf-8-sig' )
df_clean_cat_acovid.to_csv(r'df_clean_cat_acovid_sent.csv', index = False, encoding = 'utf-8-sig')
df_clean_cat_all.to_csv(r'df_clean_cat_all_sent.csv', index = False, encoding = 'utf-8-sig')
```

Step 9: Export Sentiment Data for Dashboard

Export the data for the dashboard. A function will be created to remove the stopwords from the and then export the file in the relevant folder. Do edit the file path to the correct file path.

Export data needed for Dashboard

Here, we will prepare the data for dashboard purposes.

We will remove the stopwords in df_clean_cat_all dataframe before it will be loaded into PowerBI.

```
def df_wc_no_stopwords(data, stopwords):
    output = []
    output_df = data
    #t = data[data['sentiment']=='Negative']
    #p = t[t['category'] == cat]

    for text in data['text']:
        s_text = text.split()
        result_text = [word for word in s_text if word.lower() not in stopwords]
        result = ' '.join(result_text)
        output.append(result)

    output_df['text'] = pd.Series(output)

    return output_df
```

```
df_clean_cat_all_export = df_wc_no_stopwords(df_clean_cat_all, stopwords)
```

```
# The following code will export the file for dashboard
# Do change the code accordingly to your preferred path file

df_clean_cat_all_export.to_csv(r'df_clean_cat_all_export.csv', index = False)
```

3.2.1.2 Survey

Step 1: Open *Text Analysis (Survey) LSTM.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Sentiment Analysis* > *Text Analysis (Survey) LSTM.ipynb* to open the text analysis file.

Step 2: Import Libraries

```
import json
import csv
import ast
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import seaborn as sns
import re
from datetime import date, datetime
import time
import numpy as np
import nltk
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from nltk.sentiment import SentimentIntensityAnalyzer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
%matplotlib inline

from wordcloud import WordCloud
```

Step 3: Read Files

Update file path accordingly. For this file in particular, there were blanks cells that needed to be filled.

```
#fill blanks with empty string
df_survey = pd.read_excel('../../../Data/Survey/raw_survey_data_402.xlsx')
df_survey.fillna('', inplace=True)
df_survey.describe()
```

Step 4: Cleaning Twitter data to train Machine Learning(ML) model

We used Twitter data to train our chosen machine learning model, LSTM, to label our survey results to identify their sentiments.

```
#cleaning
import re    # RegEx for removing non-letter characters

import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
from nltk.stem.porter import *


def tweet_to_words(tweet):
    ''' Convert tweet text into a sequence of words '''

    # convert to lowercase
    text = tweet.lower()
    # remove non letters
    text = re.sub(r"[^a-zA-Z0-9]", " ", text)
    # tokenize
    words = text.split()
    # remove stopwords
    words = [w for w in words if w not in stopwords.words("english")]
    # apply stemming
    words = [PorterStemmer().stem(w) for w in words]
    # return list
    return words

print("\nOriginal tweet ->", df['clean_text'][0])
print("\nProcessed tweet ->", tweet_to_words(df['clean_text'][0]))
```

Step 5: Training ML model

ML model is trained on cleaned Twitter dataset.

```
# Apply data processing to each tweet
X = list(map(tweet_to_words, df['clean_text']))

max_words = 5000
max_len=50

def tokenize_pad_sequences(text):
    '''
    This function tokenize the input text into sequnences of intergers and then
    pad each sequence to the same length
    '''
    # Text tokenization
    tokenizer = Tokenizer(num_words=max_words, lower=True, split=' ')
    tokenizer.fit_on_texts(text)
    # Transforms text to a sequence of integers
    X = tokenizer.texts_to_sequences(text)
    # Pad sequences to the same length
    X = pad_sequences(X, padding='post', maxlen=max_len)
    # return sequences
    return X, tokenizer

print('Before Tokenization & Padding \n', df['clean_text'][0])
X, tokenizer = tokenize_pad_sequences(df['clean_text'])
print('After Tokenization & Padding \n', X[0])
```

Metrics such as accuracy and precision are used to measure the performance of the model.

```
from keras.models import Sequential
from keras.layers import Embedding, Conv1D, MaxPooling1D, Bidirectional, LSTM, Dense, Dropout
from keras.metrics import Precision, Recall

vocab_size = 5000
embedding_size = 32

# Build model
model3 = Sequential()
model3.add(Embedding(vocab_size, embedding_size, input_length=max_len))
model3.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model3.add(MaxPooling1D(pool_size=2))
model3.add(Bidirectional(LSTM(32)))
model3.add(Dropout(0.4))
model3.add(Dense(3, activation='softmax'))

print(model3.summary())

# Compile model
model3.compile(loss='categorical_crossentropy', optimizer='adam',
               metrics=['accuracy', Precision(), Recall()])

# Train model
num_epochs = 1
batch_size = 32
history3 = model3.fit(X_train, y_train,
                      validation_data=(X_valid, y_valid),
                      batch_size=batch_size, epochs=num_epochs)

# Evaluate model on the test set
loss, accuracy, precision, recall = model3.evaluate(X_test, y_test, verbose=0)
# Print metrics
print('')
print('CNN + LSTM Accuracy  : {:.2f}'.format(100 * accuracy), '%')
print('CNN + LSTM Precision : {:.2f}'.format(100 * precision), '%')
print('CNN + LSTM Recall    : {:.2f}'.format(100 * recall), '%')
print('CNN + LSTM F1 Score  : {:.2f}'.format(f1_score(precision, recall)), '%')
```

Step 6: Labelling the data

Using the trained ML model, run the model to label the survey data results and obtain the sentiments for each question.

```
#LSTM sentiment

col_name = ["Based off your ranking choices above, why do you prefer not to take the last 2 least preferred mode of
            "Under what circumstances will you take your last 2 least preferred mode of transport?",
            "On what occasion would you charter a bus?",
            "Do you ever experienced difficulties finding a ride? Can you briefly describe when and where? (E.g. MRT
            "Why do you prefer these ride-hailing services?",
            "Do you ever experienced difficulties finding a ride? Can you briefly describe when and where? (E.g. MRT
            "If Yes, please briefly describe the problem.",
            "If No, what are the difficulties that you've faced? Are there any locations where you think that it sho
            "What are your perceptions of the companies that you've selected above?",
            "What are your opinions on ComfortDelGro?",
            "How do you think ComfortDelGro can improve their services?"
]
stop = stopwords.words('english')
newStopWords = ['NA','na','Na','nil','Nil','NIL']
stop.extend(newStopWords)
#include NA, na, nil, Nil, NIL, Na

for col in col_name:
    df_survey[col+'sentiment'] = sentiment_text(df_survey,col)
    print(df_survey[col+'sentiment'])
    #sentiment_text(df_survey,col)
df_survey.head()
```

Step 7: Visualize Sentiment Frequency

Run the code for each question to visualize the frequency of the 3 sentiments: Positive, Neutral, Negative.

```
#On what occasion would you charter a bus?sentiment
fig = plt.figure()
ax = fig.add_axes([1,1,1,1])
ax.set_title("On what occasion would you charter a bus?sentiment")
ax.set_ylabel('Sentiments')
ax.set_ylabel('Number of Sentiments')
ax.bar("Negative", len(df_survey[df_survey["On what occasion would you charter a bus?sentiment"]=="Negative"]))
ax.bar("Neutral", len(df_survey[df_survey["On what occasion would you charter a bus?sentiment"]=="Neutral"]))
ax.bar("Positive", len(df_survey[df_survey["On what occasion would you charter a bus?sentiment"]=="Positive"]))
```

Step 8: Visualize Word Clouds

Run the following codes to obtain the overall word cloud for the different questions.

```
word_cloud(df_survey, stop, "Why do you prefer these ride-hailing services?")
```

The following codes are to obtain the word clouds for the different questions based on a particular sentiment such as 'Negative' and 'Positive'. This is to zoom into the areas of weaknesses and strengths based on the public responses.

```
wc_neg(df_survey, stop,
       'Why do you prefer these ride-hailing services?sentiment',
       'Why do you prefer these ride-hailing services?')
```

```
wc_pos(df_survey, stop,
       'Why do you prefer these ride-hailing services?sentiment',
       'Why do you prefer these ride-hailing services?')
```

3.2.1.3 CSISG Data

For CSISG data from Year ***2015 to 2019***:

Step 1: Open *Text Analysis (CSISG - 2015 to 2019) LSTM.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Sentiment Analysis* > *Text Analysis (CSISG - 2015 to 2019) LSTM.ipynb* to open the text analysis file.

For CSISG data from Year ***2020***:

Step 1: *Open Text Analysis (CSISG - 2020) LSTM.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Sentiment Analysis* > *Text Analysis (CSISG - 2020) LSTM.ipynb* to open the text analysis file.

Repeat Steps 2 to 8 from Section 3.2.1.2 to obtain sentiment analysis and visualizations.

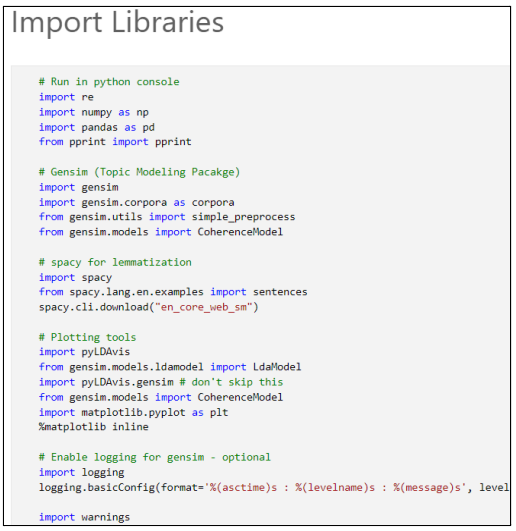
3.2.2 Topic Modelling

3.2.2.1 Reddit

Step 1: Open *reddit_topic_modelling_CDG.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Topic Modelling* > *reddit_topic_modelling_CDG.ipynb* to open the reddit topic modelling file.

Step 2: Import Libraries


Import Libraries

```
# Run in python console
import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim (Topic Modeling Pacakge)
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel

# spacy for lemmatization
import spacy
from spacy.lang.en.examples import sentences
spacy.cli.download("en_core_web_sm")

# Plotting tools
import pyLDAvis
from gensim.models.ldamodel import LdaModel
import pyLDAvis.gensim # don't skip this
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level

import warnings
```

Step 3: Load stopwords

Load the stopwords and include or exclude any relevant stopwords.

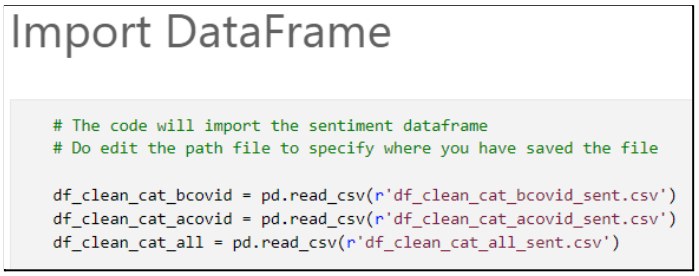
```
# Stopwords will be loaded here
# Do edit the stopwords you wish to use in the code below

stop_new = ['mrt', 'station', 'see', 'bus', 'buses', 'line', 'account', 'one', 'full',
            'get', 'min', 'class', 'year', 'pm', '', 'take', 'taxi', 'allow', 'last', 'almost',
            'post', 'start', 'cross', 'car_rental', 'comfortdelgro', 'singapore', 'grab',
            'night', 'find', 'need', 'build', 'photo', 'video', 'leave', 'car', 'train',
            'think', 'food', 'make', 'set', 'fresh', 'sbs', 'driver', 'delgro', 'comfort',
            'move',  'ever', 'blue', 'uber', 'go', 'look', 'use', 'also', 'give',
            'many', 'come', 'lot', 'seem', 'guess', 'definitely', 'sure', 'keep', 'much', 'already',
            'do', 'lol', 'people', 'well', 'back', 'week', 'u', 'want', 'day', 'will', 'know',
            'even', 'really', 'said', 'say', 'cab', 'public transport', 'taxis', 'public',
            'transport', 'thing', 'still', 'got', 'now', 's', 'stop', 'around', 'another',
            'smrt', 'next', 'us', 'may', 'person', 'years', 'going', 'trains', 'way',
            'seat', 'guy', 'https', 'always', 'riders', 'cabbie', 'sg', 'drivers', 'auntie', 'man',
            'uncle', 'stops', 'someone', 'something', 'andy', 'cabs', 'cabbies', 'order', 'delivery',
            'gojek', 'quite','fucking', 'every', 'getting', 'trying', 'told', 'something', 'singaporean',
            'feel', 'lta', 'fuck', 'without', 'let', 'made', 'getting', 'passenger', 'restaurant', 'grabfood',
            'merchant', 'don', 't', 'customer', 'cars', 'actually', 'senior', 'entrepreneur', 'rental', 'cars', 'current']

stop_words.extend(stop_new)
```

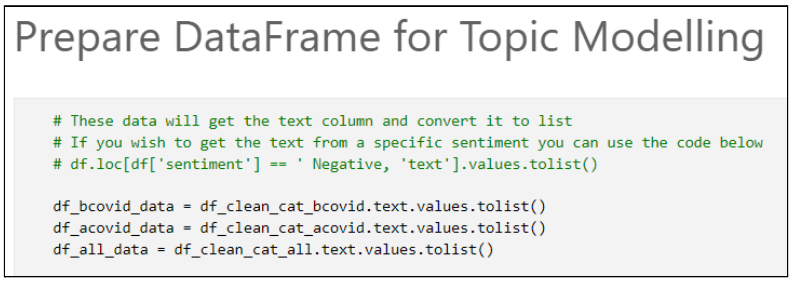
Step 4: Import Data Frame

Import the relevant data frame exported in the reddit sentiment analysis. The before and after covid codes can be skipped if that is not the focus of the analysis.



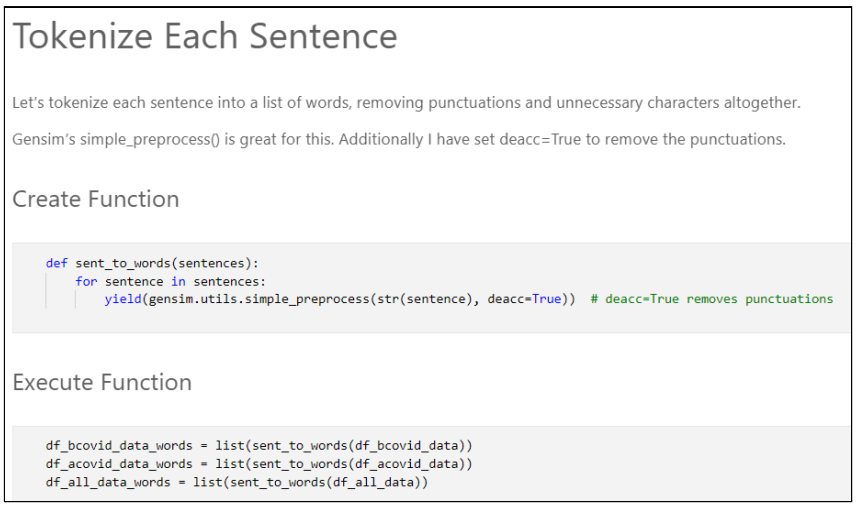
Step 5: Prepare data for Topic Modelling

Before running the topic modelling, the data have to be transformed to a list first. The figure below shows the code used to get the text values to change it to list. The figure below also shows how to run a topic modelling for text in specific sentiment. If analysis before different periods, before and after covid is not the focus of the analysis, the code for *df_bcovid_data* and *df_acovid_data* can be skipped.



Step 6: Tokenize Each Sentence

Next, the text that was converted to a list will be tokenized. If analysis before different periods, before and after covid is not the focus of the analysis, the code for *df_bcovid_data_words* and *df_acovid_data_words* can be skipped.



Tokenize Each Sentence

Let's tokenize each sentence into a list of words, removing punctuations and unnecessary characters altogether.

Gensim's simple_preprocess() is great for this. Additionally I have set deacc=True to remove the punctuations.

Create Function

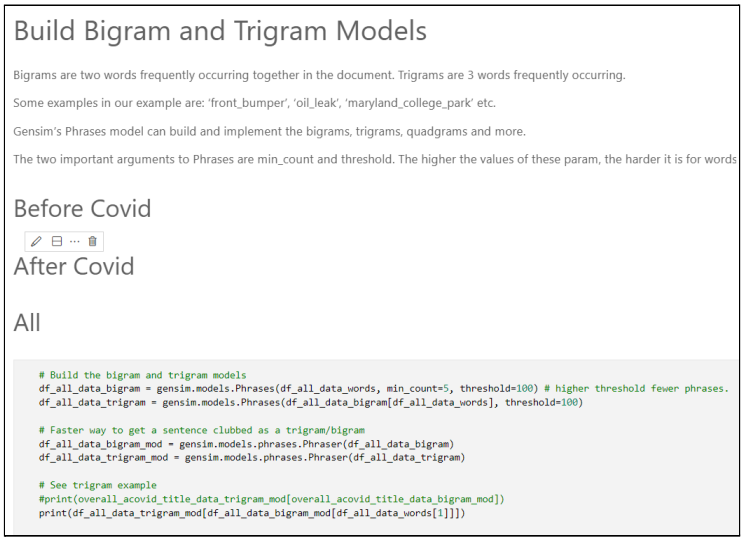
```
def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))  # deacc=True removes punctuations
```

Execute Function

```
df_bcovid_data_words = list(sent_to_words(df_bcovid_data))
df_acovid_data_words = list(sent_to_words(df_acovid_data))
df_all_data_words = list(sent_to_words(df_all_data))
```

Step 7: Build Bigram and Trigram Models

Next, the bigram and trigram models will be built to find out two or three words frequently together in the document. The figure below shows the code used to find the bigram and trigram for the overall (*df_all_data_trigram*). For the codes for the specific time periods (before and after covid), do refer to the scripts.



Build Bigram and Trigram Models

Bigrams are two words frequently occurring together in the document. Trigrams are 3 words frequently occurring.

Some examples in our example are: 'front_bumper', 'oil_leak', 'maryland_college_park' etc.

Gensim's Phrases model can build and implement the bigrams, trigrams, quadgrams and more.

The two important arguments to Phrases are min_count and threshold. The higher the values of these param, the harder it is for words

Before Covid

After Covid

All

```
# Build the bigram and trigram models
df_all_data_bigram = gensim.models.Phrases(df_all_data_words, min_count=5, threshold=100) # higher threshold fewer phrases.
df_all_data_trigram = gensim.models.Phrases(df_all_data_bigram[df_all_data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
df_all_data_bigram_mod = gensim.models.phrases.Phraser(df_all_data_bigram)
df_all_data_trigram_mod = gensim.models.phrases.Phraser(df_all_data_trigram)

# See trigram example
#print(overall_acovid_title_data_trigram_mod[overall_acovid_title_data_bigram_mod])
print(df_all_data_trigram_mod[df_all_data_bigram_mod[df_all_data_words[1]]])
```

Step 8: Remove Stopwords

Stopwords will be needed to be removed from the text as well. The text below shows the function that would be used to remove the stopwords.

```
# Define functions for stopwords, bigrams, trigrams and lemmatization
def remove_stopwords(texts):

    return [[word for word in simple_preprocess(str(doc)) if word not in stop_words] for doc in texts]

def make_bigrams(texts, bigram_mod):
    return [bigram_mod[doc] for doc in texts]

def make_trigrams(texts, trigram_mod, bigram_mod):
    return [trigram_mod[bigram_mod[doc]] for doc in texts]

def lemmatization(texts, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV']):
    """https://spacy.io/api/annotation"""
    texts_out = []
    for sent in texts:
        doc = nlp(" ".join(sent))
        texts_out.append([token.lemma_ for token in doc if token.pos_ in allowed_postags])
    return texts_out
```

Step 9: Execute function to remove stop words and run bigram and trigram model

After building the bigram, trigram, and stopwords functions, it is ready to be performed. The figure below shows the code used to find the bigram and trigram for the overall (*df_all_data_words_trigram*). For the codes for the specific time periods (before and after covid), do refer to the scripts.

```
# Remove Stop Words
df_all_data_words_nostops = remove_stopwords(df_all_data_words)

# Form Bigrams
df_all_data_words_bigrams = make_bigrams(df_all_data_words_nostops, df_all_data_bigram_mod)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
df_all_data_lemmatized = lemmatization(df_all_data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(df_all_data_lemmatized[:1])
```

Step 10: Create id2words and corpus

The topic model takes in two variables, id2words and corpus. Thus, id2words and corpus must be built first. The figure below shows the code for the overall (*df_all_data_id2words*). For the codes for the specific time periods (before and after covid), do refer to the scripts.

```
# Create Dictionary
df_all_data_id2word = corpora.Dictionary(df_all_data_lemmatized)

# Create Corpus
df_all_data_texts = df_all_data_lemmatized

# Term Document Frequency
df_all_data_corpus = [df_all_data_id2word.doc2bow(text) for text in df_all_data_texts]

# View
print(df_all_data_corpus[:1])
```

Step 11: Build LDA Model

Next is to build the LDA model that will be used to run the topic modelling.

```
# Build LDA model
def lda_model(corpus, id2word, n_topics):

    model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=n_topics,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
    return model
```

Step 12: Visualize Coherence Values

One last step before running the LDA model is to identify the ideal number of topics to run. A coherence value will be calculated and visualised as a line chart. The ideal number of topics will be the point at which the line graph shows a sharp increase.

```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of topics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model=LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topics)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

Before Covid

```
model_list, coherence_values = compute_coherence_values(dictionary=df_bcovid_data_id2word, corpus=df_bcovid
# Show graph
import matplotlib.pyplot as plt
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
```

Step 13: Visualize Topics

Lastly, it is to visualize the topic models using the code shown in the figure below. If before and after covid are not the focus of the analysis, do use the all code instead, available in the script.

Visualize the topics

Now that the LDA model is built, the next step is to examine the produced topics and the associated keywords. There is n

```
# lda_model takes in 3 elements, the corpus, id2word, and the number of topics
# Do edit the number of topics base on the chart that was visualized ealier to get the ideal number of topics.

df_bcovid_data_ldamodel = lda_model(df_bcovid_data_corpus, df_bcovid_data_id2word, 15)
df_acovid_data_ldamodel = lda_model(df_acovid_data_corpus, df_acovid_data_id2word, 20)
df_all_data_ldamodel = lda_model(df_all_data_corpus, df_all_data_id2word, 20)
```

```
df_bcovid_data_doc_lda = df_bcovid_data_ldamodel[df_bcovid_data_corpus]
df_acovid_data_doc_lda = df_acovid_data_ldamodel[df_acovid_data_corpus]
df_all_data_doc_lda = df_all_data_ldamodel[df_all_data_corpus]
```

Before Covid

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(df_bcovid_data_ldamodel,
                              df_bcovid_data_corpus,
                              df_bcovid_data_id2word)
vis
```

3.2.2.2 Survey

Step 1: Open *Survey_topic_modelling.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Topic Modelling* > *Survey_topic_modelling.ipynb* to open the topic modelling file.

Step 2: Import Libraries

Importing the relevant libraries to conduct topic modelling.

```
# Run in python console
import re
import numpy as np
import pandas as pd
from pprint import pprint

# Gensim (Topic Modeling Pacakge)
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
from gensim.models.ldamodel import LdaModel

# spacy for lemmatization
import spacy
from spacy.lang.en.examples import sentences
spacy.cli.download("en_core_web_sm")

# Plotting tools
import pyLDAvis
import pyLDAvis.gensim # don't skip this
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt
%matplotlib inline

# Enable logging for gensim - optional
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.ERROR)

import warnings
warnings.filterwarnings("ignore",category=DeprecationWarning)

# NLTK Stop words
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
nltk.download('words')
words = set(nltk.corpus.words.words())
```

Step 3: Load stopwords

Load the stopwords include or exclude any relevant stopwords.

```
stop_new = ['mrt', 'station', 'see', 'bus', 'buses', 'line', 'account', 'one', 'full',
                    'get', 'min', 'class', 'year', 'pm', '', 'take', 'taxi', 'allow', 'last', 'almost',
                    'post', 'start', 'cross', 'car_rental', 'comfortdelgro', 'singapore', 'grab',
                    'night', 'find', 'need', 'build', 'photo', 'video', 'leave', 'car', 'train',
                    'think', 'food', 'make', 'set', 'fresh', 'sbs', 'driver', 'delgro', 'comfort',
                    'move',  'ever', 'blue', 'uber', 'go', 'look', 'use', 'also', 'give',
                    'many', 'come', 'lot', 'seem', 'guess', 'definitely', 'sure', 'keep', 'much', 'already',
                    'do', 'lol', 'people', 'well', 'back', 'week', 'u', 'want', 'day', 'will', 'know',
                    'even', 'really', 'said', 'say', 'cab', 'public transport', 'taxis', 'public',
                    'transport', 'thing', 'still', 'got', 'now', 's', 'stop', 'around', 'another',
                    'smrt', 'next', 'us', 'may', 'person', 'years', 'going', 'trains', 'way',
                    'seat', 'guy', 'https', 'always', 'riders', 'cabbie', 'sg', 'drivers', 'auntie', 'man',
                    'uncle', 'stops', 'someone', 'something', 'andy', 'cabs', 'cabbies', 'order', 'delivery',
                    'gojek', 'quite','fucking', 'every', 'getting', 'trying', 'told', 'something', 'singaporean',
                    'feel', 'lta', 'fuck', 'without', 'let', 'made', 'getting', 'passenger', 'restaurant', 'grabfood',
                    'merchant', 'don', 't', 'customer', 'cars', 'actually', 'senior', 'entrepreneur', 'rental', 'cars', 'current']

stop_words.extend(stop_new)
```

Step 4: Import Data Frame

Import Data Frame or file to conduct topic modelling on. Update path accordingly.

```
df =  pd.read_excel('../../../Data/Survey/raw_survey_data_402.xlsx')
```

Step 5: Tokenize Each Sentence

By conducting tokenization, each sentence is converted into a list of words. This removes punctuations and unnecessary characters. The following code snippet shows the code written to get a text value to change it to a list.

```
df_data_words = list(sent_to_words(df))
```

Step 6: Build Bigram and Trigram Models

Bigrams are two words that often appear together in a document. Trigrams are three words that occur frequently. Examples of this example are `front_bumper` and ` oil_leak`. Gensim's Phrases model allows you to create and implement bigrams, trigrams, quadgrams, and more. The two important arguments to Phrase are min_count and threshold. The greater the value of these parameters, the more difficult it is to combine words into a bigram.

```
# Build the bigram and trigram models
df_data_bigram = gensim.models.Phrases(df_data_words, min_count=5, threshold=100) # higher threshold fewer phrases.
df_data_trigram = gensim.models.Phrases(df_data_bigram[df_data_words], threshold=100)

# Faster way to get a sentence clubbed as a trigram/bigram
df_data_bigram_mod = gensim.models.phrases.Phraser(df_data_bigram)
df_data_trigram_mod = gensim.models.phrases.Phraser(df_data_trigram)

# See trigram example
#print(overall_acovid_title_data_trigram_mod[overall_acovid_title_data_bigram_mod])
print(df_data_trigram_mod[df_data_bigram_mod[df_data_words[1]]])
```

Step 8: Execute function to remove stop words and run bigram and trigram model

The following code snippet removes the stopwords, makes bigrams and lemmatization and calls them sequentially.

```
# Remove Stop Words
df_data_words_nostops = remove_stopwords(df_data_words)

# Form Bigrams
df_data_words_bigrams = make_bigrams(df_data_words_nostops, df_data_bigram_mod)

# Initialize spacy 'en' model, keeping only tagger component (for efficiency)
# python3 -m spacy download en
nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])

# Do lemmatization keeping only noun, adj, vb, adv
df_data_lemmatized = lemmatization(df_data_words_bigrams, allowed_postags=['NOUN', 'ADJ', 'VERB', 'ADV'])

print(df_data_lemmatized[:1])
```

Step 9: Create id2words and corpus

The topic model requires two variables, id2words and corpus. Therefore, id2words and corpus are created beforehand.

```
# Create Dictionary
df_data_id2word = corpora.Dictionary(df_data_lemmatized)

# Create Corpus
df_data_texts = df_data_lemmatized

# Term Document Frequency
df_data_corpus = [df_data_id2word.doc2bow(text) for text in df_data_texts]

# View
print(df_data_corpus[:1])
```

Step 10: Build LDA model

After creating the dictionary and corpus, we need to build the LDA model and define the number of topics for the model to generate.

```
# Build LDA model
def lda_model(corpus, id2word, n_topics):

    model = gensim.models.ldamodel.LdaModel(corpus=corpus,
                                            id2word=id2word,
                                            num_topics=n_topics,
                                            random_state=100,
                                            update_every=1,
                                            chunksize=100,
                                            passes=10,
                                            alpha='auto',
                                            per_word_topics=True)
    return model
```

```
df_data_ldamodel = lda_model(df_data_corpus, df_data_id2word, 30)
```

```
df_data_doc_lda = df_data_ldamodel[df_data_corpus]
```

The optimal number of topics can be achieved by running the following and identifying the highest coherence score possible (aka the elbow model).

```
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model=LdaModel(corpus=corpus, id2word=dictionary, num_topics=num_topics)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

```
model_list, coherence_values = compute_coherence_values(dictionary=df_data_id2word,
                                                         corpus=df_data_corpus,
                                                         texts=df_data_lemmatized,
                                                         start=1, limit=40, step=1)
# Show graph
import matplotlib.pyplot as plt
limit=40; start=1; step=1;

x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```

Step 11: Calculate Model Perplexity and Topic Coherence

To evaluate the performance of the model, calculation of the model's perplexity and coherence score of the topics is carried out. The combination of model perplexity and topic coherence is a convenient mean to assess how good a given topic model is.

The following code snippet calculates the above mentioned metrics. Perplexity measures how surprised a model reacts to unseen data. Topic Coherence measures score a single topic by measuring the degree of semantic similarity between high scoring words in the topic.

```
# Compute Perplexity
print('\nPerplexity: ', df_data_ldamodel.log_perplexity(df_data_corpus))  # a measure of how good the model is. lower the better.

# Compute Coherence Score
df_data_coherence_model_lda = CoherenceModel(model=df_data_ldamodel, texts=df_data_lemmatized, dictionary=df_data_id2word, coherence='c_v')
df_data_coherence_lda = df_data_coherence_model_lda.get_coherence()
print('\nCoherence Score: ', df_data_coherence_lda)
```

Step 12: Visualize Topics

After building the model, to best view and analyse the results, a visualisation is generated using pyLDAvis package's interactive chart. It also works well with jupyter notebooks.

```
# Visualize the topics
pyLDAvis.enable_notebook()
vis = pyLDAvis.gensim.prepare(df_data_ldamodel,
                              df_data_corpus,
                              df_data_id2word)
vis
```

3.2.2.3 CSISG Data

For CSISG data from **2015 to 2019**:

Step 1: Open *CSISG_2015_to_2019_topic_modelling.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Topic Modelling* > *CSISG_2015_to_2019_topic_modelling.ipynb* to open the topic modelling file.

For CSISG data from **2020**:

Step 1: Open *CSISG_2020_topic_modelling.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Topic Modelling* > *CSISG_2020_topic_modelling.ipynb* to open the topic modelling file.

Repeat Steps 2 to 12 from Section 3.2.2.2 to conduct topic modelling on the dataset.

3.2.2.4 Combined Data Sources

Step 1: Open *CSISG_2015_to_2019_topic_modelling.ipynb*

Navigate to *System* > *Code* > *Data Analysis* > *Topic Modelling* > *Combined_topic_modelling.ipynb* to open the topic modelling file.

Repeat Steps 2 to 12 from Section 3.2.2.2 to conduct topic modelling on the dataset.

3.3 Clustering Analysis

Step 1: Open *System* > *Code* > *Data Analysis* > *Factor Analysis + Clustering* > *survey_data_402* > *Clustering Analysis.ipynb*

Step 2: Load all the relevant packages and libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import preprocessing
from sklearn import metrics
from sklearn.metrics import pairwise_distances

from sklearn.cluster import AgglomerativeClustering
from kmodes.kmodes import KModes

import plotly.express as px

pd.set_option('mode.chained_assignment', None)
```

Step 3: Load datasets

```
In [2]:
original_dataset_labels = pd.read_csv(r'../../../Data/Survey/cleaned_334_edited.csv')
original_dataset_labels
```

Out[2]:

	age	gender	monthly_income	tertiary_student	instituition	region	location	type_of_student	type_of_employment	frequently_used_mot	...	accessibility_
0	22 - 25	Female	Below $500	Yes	SMU	North	Yishun	Full-time Student	Not employed	MRT, Bus	...	Somev satis
1	22 - 25	Female	Below $500	Yes	SMU	West	Bukit Batok	Full-time Student	Employed Part-time	Bus, MRT, Taxi, Ride-hailing services	...	Ne

```
In [3]:
perception = pd.read_csv(r'../../../Data/Survey/perception_dataset.csv')
perception
```

Out[3]:

	age	gender	monthly_income	instituition	region	location	type_of_student	type_of_employment	convenient_rental_car	difficulties_rental_car	...	safe_mr
0	1	0	4	17	2	38	0	2	0	0	...	
1	1	0	4	17	4	3	0	1	1	1	...	
2	1	1	4	16	4	3	0	2	0	0	...	
3	0	0	4	17	1	30	0	2	0	0	...	

Step 4: Before applying Agglomerative Hierarchical Clustering, we have to normalize the data so that the scale of each variable is the same.

Why is this important? Well, if the scale of the variables is not the same, the model might become biased towards the variables with a higher magnitude.

```
In [6]:
from sklearn.preprocessing import normalize
data_scaled = normalize(perception)
data_scaled = pd.DataFrame(data_scaled, columns=perception.columns)
data_scaled.head()
```

Out[6]:

	age	gender	monthly_income	instituition	region	location	type_of_student	type_of_employment	convenient_rental_car	difficulties_rental_car	...
0	0.023762	0.000000	0.095050	0.403961	0.047525	0.902972	0.0	0.047525	0.000000	0.000000	...
1	0.053376	0.000000	0.213504	0.907393	0.213504	0.160128	0.0	0.053376	0.053376	0.053376	...
2	0.055902	0.055902	0.223607	0.894427	0.223607	0.167705	0.0	0.111803	0.000000	0.000000	...
3	0.000000	0.000000	0.114379	0.486111	0.028595	0.857843	0.0	0.057190	0.000000	0.000000	...
4	0.037878	0.000000	0.189389	0.568166	0.037878	0.795432	0.0	0.075755	0.000000	0.000000	...

5 rows × 28 columns

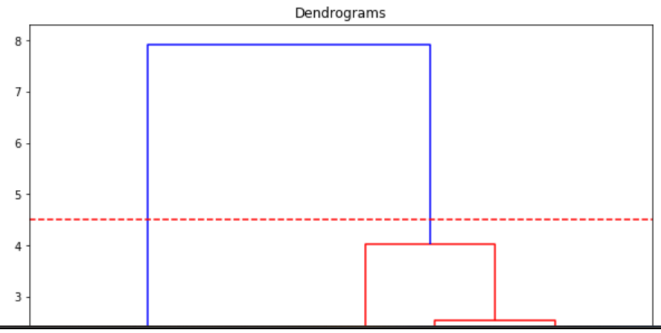
Step 5: To identify the total number of clusters required to be used as a criteria for the clustering, we first have to plot out a dendrogram.

To identify the dendrogram: Find the vertical line with maximum distance (is the blue line) and set the threshold in such a way that it cuts the tallest vertical line and cut the dendrogram The number of clusters will be the number of vertical lines which are being intersected by the line drawn using the threshold. The threshold is the maximum number of data points a sub-cluster in the leaf node of the CF tree can hold.

```
In [7]:
# draw the dendrogram to help us decide the number of clusters for this particular problem

import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(data_scaled, method='ward'))
plt.axhline(y=4.5, color='r', linestyle='--')

Out[7]: <matplotlib.lines.Line2D at 0x1798677a668>
```



Step 6: Run the Clustering algorithm on the dataset

```
In [8]:
# Cluster!
cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')

AHC_cluster_labels = cluster.fit_predict(data_scaled)
data_scaled['cluster'] = AHC_cluster_labels

In [9]:
data_scaled

Out[9]:
```

	age	gender	monthly_income	instituition	region	location	type_of_student	type_of_employment	convenient_rental_car	difficulties_rental_car	...
0	0.023762	0.000000	0.095050	0.403961	0.047525	0.902972	0.0	0.047525	0.000000	0.000000	...
1	0.053376	0.000000	0.213504	0.907393	0.213504	0.160128	0.0	0.053376	0.053376	0.053376	...
2	0.055902	0.055902	0.223607	0.894427	0.223607	0.167705	0.0	0.111803	0.000000	0.000000	...
3	0.000000	0.000000	0.114379	0.486111	0.028595	0.857843	0.0	0.057190	0.000000	0.000000	...

Step 7: Run Accuracy models to check the accuracy of the algorithms. Earlier on we also ran other clustering models, K-Modes by Cao and Huang, however by using the 3 accuracy calculations as shown here, we have identified Agglomerative Hierarchical Clustering to produce the best results and have therefore only included this model in this user guide.

```
In [10]:
# Silhouette Coefficient
# Closer to -1 suggests incorrect clustering, while closer to +1 shows that each cluster is very dense.
print("Silhouette Coefficient for AHC: " + str(metrics.silhouette_score(data_scaled, AHC_cluster_labels)))

print("")

# Calinski Harabaz Index
# The score is higher when clusters are dense and well separated, which relates to a standard concept of a cluster.
# (AKA the higher the better)
print("CH Score for AHC: " + str(metrics.calinski_harabasz_score(data_scaled, AHC_cluster_labels)))

print("")

# Davies-Bouldin Index
# The minimum score is zero, with lower values indicating better clustering
print("Davies-Bouldin Index for AHC: " + str(metrics.davies_bouldin_score(data_scaled, AHC_cluster_labels)))

Silhouette Coefficient for AHC: 0.6837267330069038

CH Score for AHC: 1198.4738972520595

Davies-Bouldin Index for AHC: 0.462419903496877
```

Step 8: (Not relevant to current dataset) Plot scatter plots to better understand the relationship between the variables, as well as identify any interesting insights. However, this will be more relevant for quantitative data whereby an example such as: a relationship between variable and variable y have a positive gradient, indicating that they are positively correlated to one another will be insightful. However, with the current dataset (that is majority categorical), such plots are not representative of the dataset, whereby the numbers currently plotted are from the encoding of data, and they are not actually ordinal (aka the order of the numbers is not significant, or even relevant).

```
In [16]:
fig, ax = plt.subplots(3, figsize=(20, 10))
ax[0].scatter(x = data_scaled['region'], y = data_scaled['reason_ride_hailing_accessible'], c=data_scaled['cluster'] )
ax[0].set_xlabel("Region")
ax[0].set_ylabel("Accessibility of Renting A Car")

ax[1].scatter(x = data_scaled['region'], y = data_scaled['reason_ride_hailing_easy_booking'], c=data_scaled['cluster'])
ax[1].set_xlabel("Region")
ax[1].set_ylabel("Ease of Renting A Car")


ax[2].scatter(x = data_scaled['region'], y = data_scaled['reason_chosen_rental_car_company_convenient'], c=data_scaled['clus
ax[2].set_xlabel("Region")
ax[2].set_ylabel("Convenience of Chosen Rental Company")

plt.show()
```


Step 9: Therefore, to properly interpret the clustering results of this dataset, we decided to make use of **(stacked) bar chart**s to understand the breakdown and visualise the various clusters instead. But before we can do that, we first have to prepare the data set as well.

Interpreting Clustering Results

```
In [ ]:
all_features = perception.columns
all_features
```

```
In [ ]:
data = original_dataset_labels[all_features]
data['cluster'] = AHC_cluster_labels
data
```

```
In [ ]:
data['cluster'].replace(to_replace = [0,1,2], value =['A', 'B', 'C'], inplace=True)
data.head()
#data.to_csv('agg_clustering_perception.csv')
```

```
In [ ]:
data.groupby('cluster').describe()
```

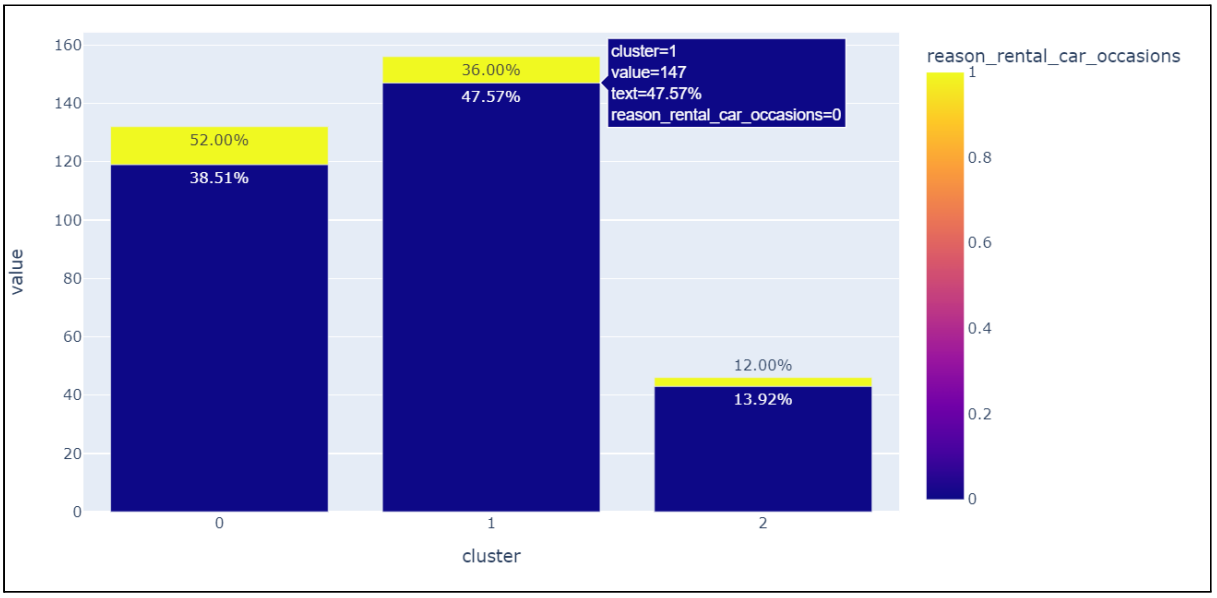
Step 10: The group decided to break down the visualisation of the various clusters and those within each clusters' answers to the various survey questions (aka the various variables) by each factor as identified during the earlier Factor Analysis.

For example, for Convenience:

- Prepare a dataframe for convenience,

```
df_convenience = data[["convenient_rental_car", "difficulties_rental_car", "reason_chosen_rental_car_company_convenient", 'r
all_features = df_convenience.drop('cluster', axis=1).columns
```

- Plot a stacked bar chart for each individual variable (aka column) as shown below, whereby each column is a cluster made up of stacked charts of the various values.

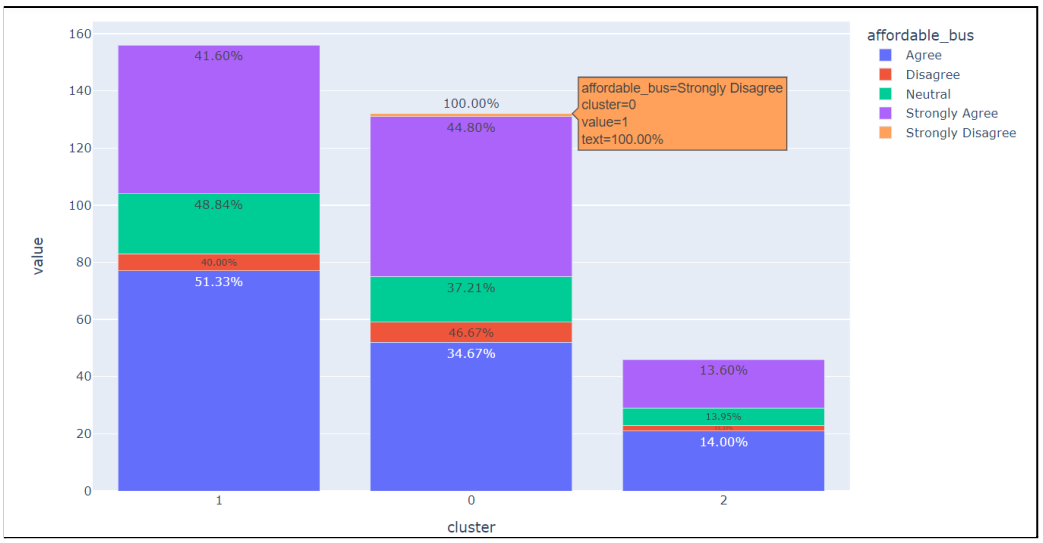


This step will ***repeat*** **for the various factors, Convenience, Affordability, Safety, Comfort, Promotion, Customer Service, Accessibility and lastly, Demographics.**

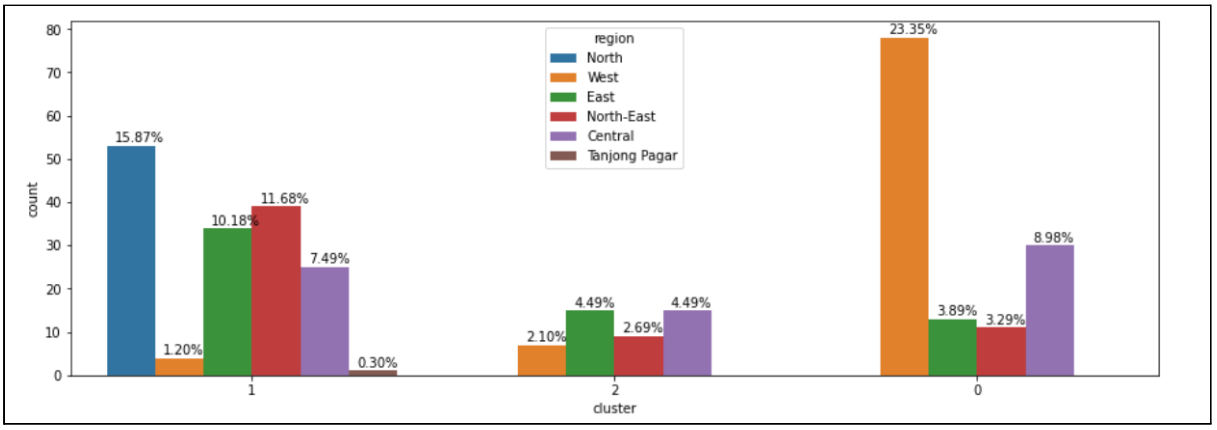
The code for each chart is the same as above, except for the values inputted to be plotted, and you can refer to the jupyter notebook for clustering for the exact codes.

Step 11: Understanding the results

We can first compare the breakdown of responses by each cluster for the various variables (columns of data) to understand the makeup of each cluster. For example, cluster A might have a higher significant percentage of respondents disagreeing that prices of transport in Singapore are affordable, possibly indicating that they are more likely to be influenced by prices as they are more sensitive to it.



We can also understand the makeup of each cluster for example in demographic variables, majority of those who live in the West belong to Cluster 1, indicating that those who live in the west have a general or average behaviour as encompassed by cluster A's characteristics.



3.3.1 Plot Clusters into Map

In this section, the following steps are to aid to plot the clusters into a geographical map and visualize the clusters based on real population dataset.

Step 1: Open *System* > *Code* > *Data Analysis* > *Geospatial Analysis* > *Clustering_Results* > *Clustering_Results.Rproj*.

Step 2: Open *Clustering_Results.Rmd*.

Step 3: Load relevant packages and libraries

```
#Getting Started

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
packages = c('sp', 'rgdal', 'sf', 'tidyverse', 'tmap',
'ggplot2','stringr', 'maptools', 'raster', 'spatstat', 'sfheaders',
'dplyr')
for(p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
  }
  library(p, character.only = T)
}
```
```

Step 4: Import Data

```
#Import Data

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
mpsz = st_read(dsn = "data/geospatial", layer = "MP14_SUBZONE_WEB_PL")
agg_clusters <- read_csv("data/aspatial/agg_clustering_perception.csv")
resident_edu <- read_csv("data/aspatial/(Education) Students Aged 5 Years
and Over by Planning Area (2020).csv")
```
```

Step 5: Clean Population Dataset for Tertiary Students ONLY

```
##Tertiary Students by Planning Area (2020)

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
resident_edu <- na.omit(resident_edu)

tertiary_students <- resident_edu %>%
  mutate(`Planning Area of Residence CAPS` = toupper(`Planning Area of
Residence`)) %>%
  dplyr::select('Planning Area of Residence CAPS', 'Polytechnic Diploma',
'Professional Qualification and Other Diploma', 'University')

tertiary_students <- tertiary_students %>%
  mutate(`Total` = rowSums(.[2:4])) %>%
  dplyr::select('Planning Area of Residence CAPS', 'Total')
```
```

Step 6: Check the Total number and Percentage of Tertiary Students

```
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Check the total number of population of tertiary students
sum(tertiary_students$Total)
```

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Check the percentage of tertiary students over the whole population
(tertiary students) by Planning Area
tertiary_students <- tertiary_students %>%
  mutate(`Percentage_Pop` = ((Total/186884)*100))
```
```

Step 7: Clean Agg_Clusters Dataset

```
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
agg_clusters <- agg_clusters %>%
  #remove location == "Prefer not to say"
  subset(location != "Prefer not to say")

agg_clusters_CLEAN <- agg_clusters %>%
  #change to upper CAPS for location
  mutate(LOCATION = toupper(location)) %>%
  mutate(REGION = toupper(region)) %>%
  #remove unwanted columns
  dplyr::select(2, 29:31)

#rename some location to proper planning areas naming convention
agg_clusters_CLEAN$LOCATION <- gsub("KALLANG / WHAMPAO", "KALLANG",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("CENTRAL AREA / CBD", "DOWNTOWN
CORE", agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("KATONG", "MARINE PARADE",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("POTONG PASIR", "TOA PAYOH",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("SIMEI", "TAMPINES",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("TAMAN JURONG", "JURONG WEST",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("UBI", "GEYLANG",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("WOODLEIGH", "TOA PAYOH",
agg_clusters_CLEAN$LOCATION)
agg_clusters_CLEAN$LOCATION <- gsub("YEW TEE", "CHOA CHU KANG",
agg_clusters_CLEAN$LOCATION)
```
```

Step 8: Raking: Balancing Weights to Predict Actual Numbers of Tertiary Students in Clusters

```
##Raking: Balancing Weights

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Get frequency of each planning area and its percentage of overall
survey_results_percent <- agg_clusters_CLEAN %>%
  mutate(count = 1) %>%
  group_by(LOCATION) %>%
  count(LOCATION, count) %>%
  mutate(freq = ifelse(is.na(count), NA, n)) %>%
  mutate(freq_percentage = freq/318*100) %>%
  dplyr::select(1,5)

pop_vs_sample <- left_join(survey_results_percent, tertiary_students, by
= c("LOCATION" = "Planning Area of Residence CAPS"))

#Balancing Weights (Target (Population) /Actual (Survey) = Weight)
pop_vs_sample_weights <- pop_vs_sample %>%
  mutate(weights = Percentage_Pop/freq_percentage) %>%
  dplyr::select(1,5)
```
```

Step 9: Merge Datasets with Singapore Geospatial Map

```
##Filtering and Mapping by Clusters

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#merge with mpsz_3414
mpsz_3414_agg_clusters <- left_join(mpsz_3414_plnarea,
agg_clusters_CLEAN, by = c("PLN_AREA_N" = "LOCATION"))
```
```

Step 10: Split Clusters to plot map

```
###Cluster 0

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
cluster0_mpsz3414 <- mpsz_3414_agg_clusters %>%
  filter(cluster == 0) %>%
  mutate(count = 1)
#Frequency of No. of Respondents in each Plan Area
cluster0_mpsz3414_PAfreq <- cluster0_mpsz3414 %>%
  group_by(PLN_AREA_N) %>%
  count(PLN_AREA_N, count) %>%
  mutate(freq = ifelse(is.na(count), NA, n)) #%>%
  #Percentage of respondents of Cluster 0 by Plan Area
  #scales::percent(freq/318, accuracy=0.001))
  #mutate(freq_percentage = round(freq/318*100, 2))

#Merge with weights
cluster0_mpsz3414_PAfreq <- left_join(cluster0_mpsz3414_PAfreq,
pop_vs_sample_weights, by = c("PLN_AREA_N" = "LOCATION"))

cluster0_mpsz3414_PAfreq <- cluster0_mpsz3414_PAfreq %>%
  #calculate weighted value
  mutate(weighted_percentage = freq*weights) %>%
  dplyr::select(1,4,7)
```
```

Repeat the same steps for Cluster 1 and Cluster 2 by changing the naming convention and cluster filter to '1' and '2'.

Do take note that Cluster 0, 1, 2 is representing Cluster A, B, C from the Section 3.3.

Step 11: Map Visualization

```
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
tmap_mode("view")

#Cluster 0
tm_shape(cluster0_mpsz3414_PAfreq) + tm_fill("weighted_percentage", style
= "jenks", n= 6, title = "Cluster 0: Weighted Percentage of Respondents")
+ tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Cluster 1
tm_shape(cluster1_mpsz3414_PAfreq) + tm_fill("weighted_percentage", style
= "jenks", n= 6, title = "Cluster 1: Weighted Percentage of Respondents")
+ tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Cluster 2
tm_shape(cluster2_mpsz3414_PAfreq) + tm_fill("weighted_percentage", style
= "jenks", n= 6, title = "Cluster 2: Weighted Percentage of Respondents")
+ tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)
```
```

Step 12: Export to csv for Power BI

```
####Export to csv for PowerBI
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
cluster0_mpsz3414_PAfreq_csv <- cluster0_mpsz3414_PAfreq %>%
  dplyr::select(1,3) %>%
  st_set_geometry(NULL)

cluster0_mpsz3414_PAfreq_csv$PLN_AREA_N <- sub("^", "Singapore, ",
cluster0_mpsz3414_PAfreq_csv$PLN_AREA_N)

write.csv(cluster0_mpsz3414_PAfreq_csv,
"data/aspatial/Cluster0_PowerBI.csv", #row.names = FALSE)
```
```

Repeat the following to export for Cluster 1 and 2 by changing the naming convention and file directory. This export will be the final files for the PowerBI.

3.4 Geospatial Analysis

In this section, the following steps will be used to analyze the CSISG dataset in a geospatial format. There will be 4 R Markdown files in total to analyze 4 different sectors namely, Bus, MRT/LRT, Taxi

and Booking Application. Do ***repeat*** the steps again to extract the individual sector results as the following steps shown will be for 1 sector only.

3.4.1 Understanding Overall Scores for CSISG aggregated scoring results

Step 1: Open *System > Code > Data Analysis > Geospatial Analysis > CSISG_Results > CSISG_Results.Rproj*.

Step 2: Open *Agg_Bus_CSISG.Rmd, Agg_MRT_LRT_CSISG.Rmd, Agg_Taxi_CSISG.Rmd, Agg_Booking_App_CSISG.Rmd*. The following steps below entails 1 of the Rmd file as above, and the steps will have to be ***repeated*** for the other 3 Rmd files.

Step 3: Import Packages and Libraries

```
# Getting Started

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
packages = c('sp', 'rgdal', 'sf', 'tidyverse', 'tmap',
'ggplot2','stringr', 'maptools', 'raster', 'spatstat', 'sfheaders',
'dplyr')
for(p in packages){
  if(!require(p, character.only = T)){
    install.packages(p)
  }
  library(p, character.only = T)
}
```
```

Step 4: Import Data

```
#Import Data

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
mpsz = st_read(dsn = "data/geospatial", layer = "MP14_SUBZONE_WEB_PL")
agg_bus <- read_csv("data/aspatial/agg_bus.csv")
```
```

Step 5: Initial cleaning for planning area dataset

```
#Data Cleaning and Segmentation

##Singapore Planning Area w/ Geometry Points

```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#transform CRS = 3414
mpsz_3414 <- st_set_crs(mpsz, 3414)

mpsz_3414_plnarea <- mpsz_3414 %>%
  group_by(PLN_AREA_N) %>%
  summarise()
```

###Visualize Maps Function
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Visualization
tmap_mode("view")
```
```

Step 6: Extract overall scores for Year 2016 - 2019

```
###agg_bus overall for Year 2016 - 2019
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
agg_bus_overall <- agg_bus %>%
  dplyr::select(3,5:9)

agg_bus_overall_PA <- aggregate(agg_bus_overall, by =
list(agg_bus_overall$planning_area), FUN=mean)
names(agg_bus_overall_PA)[1] <- 'planning_area'
agg_bus_overall_PA <- agg_bus_overall_PA %>%
  dplyr::select(1, 3:7)

agg_bus_overall_PA_mpsz3414 <- left_join(mpsz_3414_plnarea,
agg_bus_overall_PA, by = c("PLN_AREA_N" = "planning_area"))
agg_bus_overall_PA_mpsz3414 <- agg_bus_overall_PA_mpsz3414 %>%
  group_by(PLN_AREA_N) %>%
  filter(! is.na(customer_expectations_score))
```
```

Step 7: Plot Visualization for Overall

```
####Visualization for agg_bus overall
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Customer Expectations Score
tm_shape(agg_bus_overall_PA_mpsz3414) +
tm_fill("customer_expectations_score", style = "jenks", n= 6, title =
"Average Customer Expectations Score (Bus) by Planning Area from 2016 -
2019") + tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Customer Satisfaction Score
tm_shape(agg_bus_overall_PA_mpsz3414) +
tm_fill("customer_satisfaction_score", style = "jenks", n= 6, title =
"Average Customer Satisfaction Score (Bus) by Planning Area from 2016 -
2019") + tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Customer Loyalty Trust Score
tm_shape(agg_bus_overall_PA_mpsz3414) +
tm_fill("customer_loyalty-_user_trust_score", style = "jenks", n= 6,
title = "Average Customer Loyalty User Trust Score (Bus) by Planning Area
from 2016 - 2019") + tm_text("PLN_AREA_N", size = 0.65) +
tm_shape(mpsz_3414) + tm_borders(lwd = 0.1, alpha = 1) +
tm_legend(legend.outside=TRUE) #+ tm_text("PLN_AREA_N", size = 0.2)

#Perceived Overall Quality Score
tm_shape(agg_bus_overall_PA_mpsz3414) +
tm_fill("perceived_overall_quality_score", style = "jenks", n= 6, title =
"Average Perceived Overall Quality Score by (Bus) Planning Area from 2016
- 2019") + tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Perceived Value Score
tm_shape(agg_bus_overall_PA_mpsz3414) + tm_fill("perceived_value_score",
style = "jenks", n= 6, title = "Average Perceived Value Score (Bus) by
Planning Area from 2016 - 2019") + tm_text("PLN_AREA_N", size = 0.65) +
tm_shape(mpsz_3414) + tm_borders(lwd = 0.1, alpha = 1) +
tm_legend(legend.outside=TRUE) #+ tm_text("PLN_AREA_N", size = 0.2)
```
```

Step 8: Export Overall to csv for PowerBI

```
####Export to csv for PowerBI
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
agg_bus_overall_PA_mpsz3414_csv <- agg_bus_overall_PA_mpsz3414 %>%
  dplyr::select(1,3:7) %>%
  st_set_geometry(NULL)

agg_bus_overall_PA_mpsz3414_csv$PLN_AREA_N <- sub("^", "Singapore, ",
agg_bus_overall_PA_mpsz3414_csv$PLN_AREA_N)

write.csv(agg_bus_overall_PA_mpsz3414_csv,
"data/aspatial/agg_bus_overall_PowerBI.csv", row.names = FALSE)
```
```

Repeat the following steps to extract the results for all 4 sectors. Do remember to change the naming convention for the raw datasets. You may refer to the Rmd files for reference.

3.4.2 Understanding Expectations vs Satisfaction Scores

As continued from the previous section, do ***repeat*** the following steps for the ***4 individual Rmd files***.

Step 1: Data Clean to extract the comparison between Expectations and Satisfaction

````
####Expectations vs Satisfaction
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
agg_bus_overall_ExpectvsSatisf <- agg_bus_overall_PA_mpsz3414 %>%
  dplyr::select(1,2,3,6)

agg_bus_overall_ExpectvsSatisf_comparison <-
agg_bus_overall_ExpectvsSatisf %>%
  mutate(PositiveOrNegative = case_when(customer_expectations_score >
customer_satisfaction_score ~ "Yes",
                                        customer_expectations_score <
customer_satisfaction_score ~ "No",
                                        TRUE ~ "NA"))

agg_bus_overall_ExpectvsSatisf_positive <-
agg_bus_overall_ExpectvsSatisf_comparison %>%
  filter(PositiveOrNegative == "Yes") %>%
  mutate(difference = customer_expectations_score -
customer_satisfaction_score)

agg_bus_overall_ExpectvsSatisf_positive %>% arrange(desc(difference))

#Visualize difference
tm_shape(agg_bus_overall_ExpectvsSatisf_positive) + tm_fill("difference",
style = "jenks", n= 6, title = "Bus:(AVG) Expectations vs Satisfaction
Scores Difference (2016 - 2019)") + tm_text("PLN_AREA_N", size = 0.65) +
tm_shape(mpsz_3414) + tm_borders(lwd = 0.1, alpha = 1) +
tm_legend(legend.outside=TRUE) #+ tm_text("PLN_AREA_N", size = 0.2)
```
````

3.4.3 Comparing the varying scoring results by year

Continuing from the previous section, we can also analyze the varying scoring results individually by year. With that, we can see the comparison between each year respectively.

Step 1: Clean Data by Year

```
###Individually by year
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#2019
agg_bus_2019 <- agg_bus %>%
  filter(year == 2019) %>%
  dplyr::select(1,3,5:9)

agg_bus_2019_PA <- aggregate(agg_bus_2019, by =
list(agg_bus_2019$planning_area), FUN=mean)
names(agg_bus_2019_PA)[1] <- 'planning_area'
agg_bus_2019_PA <- agg_bus_2019_PA %>%
  dplyr::select(1:2,4:8)

agg_bus_2019_PA_mpsz3414 <- left_join(mpsz_3414_plnarea, agg_bus_2019_PA,
by = c("PLN_AREA_N" = "planning_area"))
agg_bus_2019_PA_mpsz3414 <- agg_bus_2019_PA_mpsz3414 %>%
  group_by(PLN_AREA_N) %>%
  filter(year == 2019)
  #drop_na()
```

Repeat the following by filtering the year for 2018, 2017 and 2016, to extract the dataset for the respective years.
```

Step 2: Compare the scores over the years (Bar Plot)

```
#Customer Expectations Score
ggplot(agg_bus_2016_2019_comparison_PA, aes(planning_area,
customer_expectations_score,  fill=year), xlab="Age Group") +
geom_bar(stat="identity", position="dodge", width=.6) +
scale_x_discrete(labels=abbreviate) +
geom_text(aes(label=round(customer_expectations_score, 1)), vjust=0,
size=3) + labs(title="Comparison of Customer Expectation Scores (Bus)
from 2016 to 2019")

#Customer Satisfaction Score
ggplot(agg_bus_2016_2019_comparison_PA, aes(planning_area,
customer_satisfaction_score,  fill=year), xlab="Age Group") +
geom_bar(stat="identity", position="dodge", width=.6) +
scale_x_discrete(labels=abbreviate) +
geom_text(aes(label=round(customer_satisfaction_score, 1)), vjust=0,
size=3) + labs(title="Comparison of Customer Satisfaction Scores (Bus)
from 2016 to 2019")

#Customer Loyalty User Trust Score
ggplot(agg_bus_2016_2019_comparison_PA, aes(planning_area,
agg_bus_2016_2019_comparison_PA$`customer_loyalty-_user_trust_score`,
fill=year), xlab="Age Group") + geom_bar(stat="identity",
position="dodge", width=.6) + scale_x_discrete(labels=abbreviate) +
geom_text(aes(label=round(agg_bus_2016_2019_comparison_PA$`customer_loyal
ty-_user_trust_score`, 1)), vjust=0, size=3) + labs(title="Comparison of
Customer Loyalty User Trust Scores (Bus) from 2016 to 2019")

#Perceived Overall Quality Score
ggplot(agg_bus_2016_2019_comparison_PA, aes(planning_area,
perceived_overall_quality_score,  fill=year), xlab="Age Group") +
geom_bar(stat="identity", position="dodge", width=.6) +
scale_x_discrete(labels=abbreviate) +
geom_text(aes(label=round(perceived_overall_quality_score, 1)), vjust=0,
size=3) + labs(title="Comparison of Perceived Overall Quality Scores
(Bus) from 2016 to 2019")

#Perceived Value Score
ggplot(agg_bus_2016_2019_comparison_PA, aes(planning_area,
perceived_value_score,  fill=year), xlab="Age Group") +
geom_bar(stat="identity", position="dodge", width=.6) +
scale_x_discrete(labels=abbreviate) +
geom_text(aes(label=round(perceived_value_score, 1)), vjust=0, size=3) +
labs(title="Comparison of Perceived Value Scores (Bus) from 2016 to
2019")
```

Step 3: Plot Visualization for the Respective Years

```
####Visualization for 2019 agg_bus
```{r echo=TRUE, eval=TRUE, warning=FALSE, message=FALSE}
#Customer Expectations Score
tm_shape(agg_bus_2019_PA_mpsz3414) +
tm_fill("customer_expectations_score", style = "jenks", n= 6, title =
"Average Customer Expectations Score (Bus) by Planning Area in 2019") +
tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) + tm_borders(lwd
= 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Customer Satisfaction Score
tm_shape(agg_bus_2019_PA_mpsz3414) +
tm_fill("customer_satisfaction_score", style = "jenks", n= 6, title =
"Average Customer Satisfaction Score (Bus) by Planning Area in 2019") +
tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) + tm_borders(lwd
= 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Customer Loyalty Trust Score
tm_shape(agg_bus_2019_PA_mpsz3414) +
tm_fill("customer_loyalty-_user_trust_score", style = "jenks", n= 6,
title = "Average Customer Loyalty User Trust Score (Bus) by Planning Area
in 2019") + tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Perceived Overall Quality Score
tm_shape(agg_bus_2019_PA_mpsz3414) +
tm_fill("perceived_overall_quality_score", style = "jenks", n= 6, title =
"Average Perceived Overall Quality Score (Bus) by Planning Area in 2019")
+ tm_text("PLN_AREA_N", size = 0.65) + tm_shape(mpsz_3414) +
tm_borders(lwd = 0.1, alpha = 1) + tm_legend(legend.outside=TRUE) #+
tm_text("PLN_AREA_N", size = 0.2)

#Perceived Value Score
tm_shape(agg_bus_2019_PA_mpsz3414) + tm_fill("perceived_value_score",
style = "jenks", n= 6, title = "Average Perceived Value Score (Bus) by
Planning Area in 2019") + tm_text("PLN_AREA_N", size = 0.65) +
tm_shape(mpsz_3414) + tm_borders(lwd = 0.1, alpha = 1) +
tm_legend(legend.outside=TRUE) #+ tm_text("PLN_AREA_N", size = 0.2)
```
```

Repeat the following to extract the visualizations for the other respective years.

3.5 Correlation Analysis

Step 1: Regrouping metrics into the different factors identified in Factor Analysis and get the average value of all metrics under the factors of each sub-sector

Step 2: Load the aggregated datasets

Load the relevant data ¶

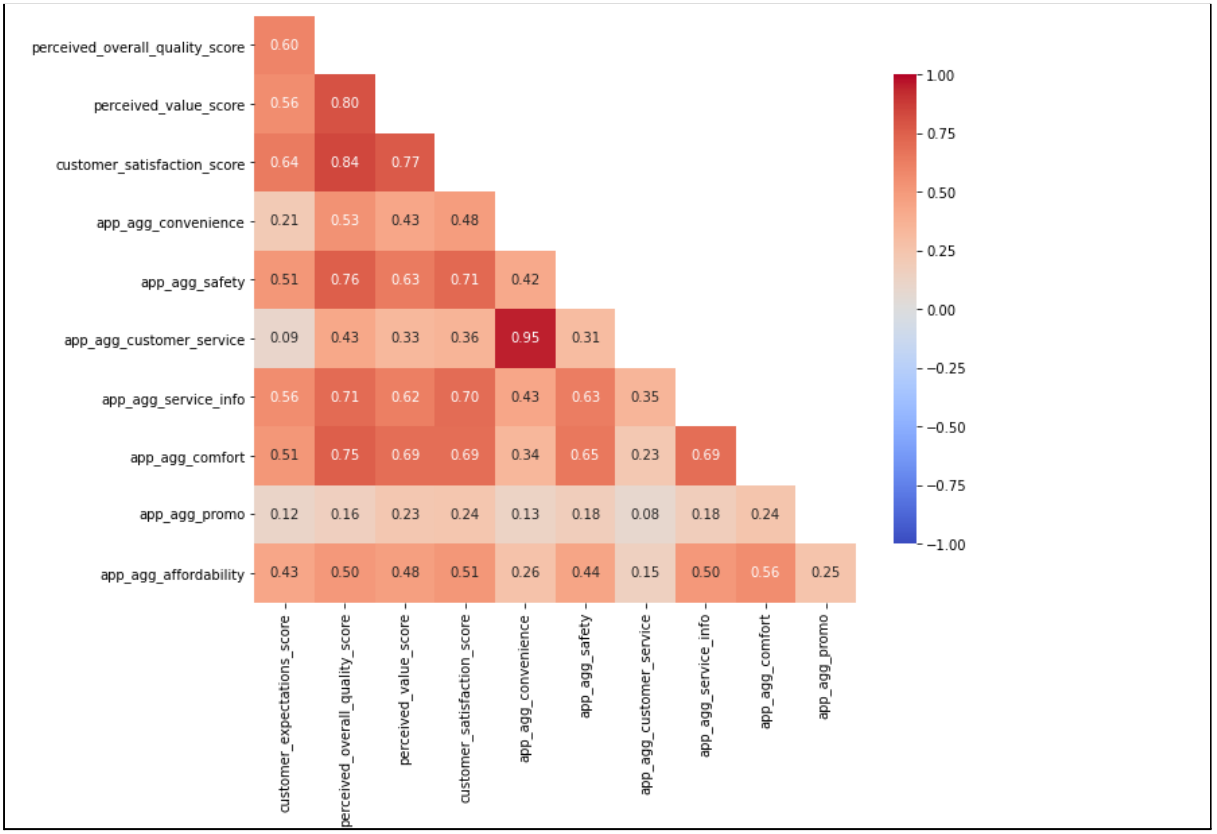
```
df_mrt_lrt = pd.read_csv(r'../../../Data/CSISG/Correlation Data/agg_mrt_lrt.csv')
df_taxi = pd.read_csv(r'../../../Data/CSISG/Correlation Data/agg_taxi.csv')
df_app = pd.read_csv(r'../../../Data/CSISG/Correlation Data/agg_app.csv')
df_bus = pd.read_csv(r'../../../Data/CSISG/Correlation Data/agg_bus.csv')
```

Step 3: Using spearman method to get the correlation matrix

```
df_mrt_lrt_corr_matrix = df_mrt_lrt.corr(method = 'spearman').iloc[7:, 2:6]
df_mrt_lrt_corr_matrix
```

	customer_expectations_score	perceived_overall_quality_score	perceived_value_score	customer_satisfaction_score
train_agg_service_info	0.125020	0.249564	0.287731	0.187195
train_agg_customer_service	0.154376	0.326536	0.311167	0.233210
train_agg_convenience	0.206474	0.366520	0.350211	0.351679
train_agg_comfort	0.075100	0.263847	0.269251	0.155622
train_agg_safety	0.363029	0.592499	0.469011	0.602082

Step 4: Visualise using heatmap



Step 5: Get the correlation of each factor group by planning area

```
df_app_area_corr_matrix = round(df_app.groupby(['planning_area']).corr(), 2).dropna(axis = 0, how = 'all').fillna("")
df_app_area_corr_matrix = df_app_area_corr_matrix.drop(['customer_expectations_score', 'customer_expectations_score', 'p
df_app_area_corr_matrix = df_app_area_corr_matrix.drop(['app_agg_convenience', 'app_agg_safety', 'app_agg_customer_servi
```

```
df_app_area_corr_matrix
```

2]:

planning_area		customer_expectations_score	perceived_overall_quality_score	perceived_value_score	customer_satisfaction_sc
	app_agg_convenience	0.45	0.76	0.67	0
	app_agg_safety	0.62	0.89	0.69	0
	app_agg_customer_service	0.30	0.61	0.54	0
ANGMOKIO	app_agg_service_info	0.70	0.92	0.92	0
	app_agg_comfort	0.77	0.84	0.8	0
	app_agg_promo	-0.16	-0.07	0.16	-0
	app_agg_affordability	0.54	0.77	0.85	0
	app_agg_convenience	1.00	1.00	-1	1
	app_agg_safety	1.00	1.00	-1	1
	app_agg_customer_service	1.00	1.00	-1	1
BEDOK	app_agg_service_info	1.00	1.00	-1	1
	app_agg_comfort	-1.00	-1.00	1	-1
	app_agg_promo	1.00	1.00	-1	1

4.0 Application - Dashboard

This section entails the steps that are needed after the export of files from the respective analyses for the Power BI dashboard.

4.1 Data Sources

For reference, navigate to *Dashboard* > *data*.

1. *cleaned_survey*.csv
2. *csisg_agg_combined.xlsx* (average csisg score)
3. *reddit_csisg_text_combined.xlsx* (perception about singapore transportation)
4. *correlation_all.xlsx* (csisg top correlated factors)
5. *survey_agg_clustering.xlsx* (survey factor breakdown)
6. *population_agg_clustering.xlsx* (survey projected population numbers)

4.2 Manipulation

a. reddit_csisg_text_combined.xlsx

The text data provided by CSISG was added into the extracted reddit text and labelled as positive sentiment.

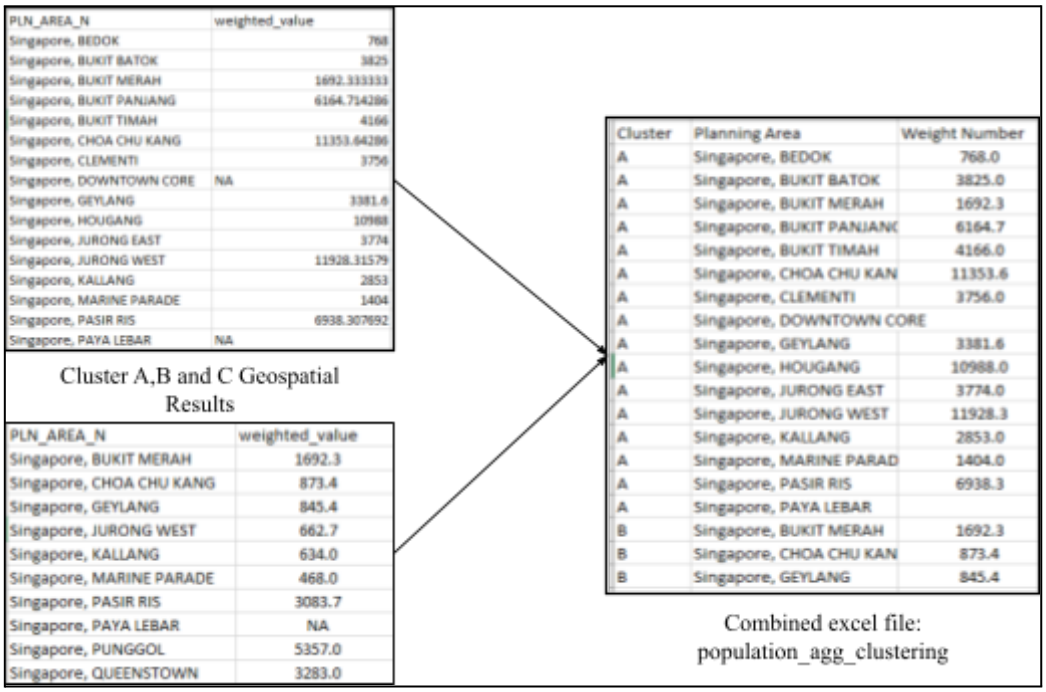
b. correlation_all.xlsx

The correlation results are manually combined on excel as shown below.

Source	Scores	Factors	Correlation Numbers
Bus	Customer Expectations Score	Service Information	0.426121
Bus	Customer Expectations Score	Customer Service	0.421063
Bus	Customer Expectations Score	Convenience	0.382501
Bus	Customer Expectations Score	Comfort	0.435537
Bus	Customer Expectations Score	Safety	0.289151
Train	Customer Expectations Score	Service Information	0.278647
Train	Customer Expectations Score	Customer Service	0.285749
Train	Customer Expectations Score	Convenience	0.287681
Train	Customer Expectations Score	Comfort	0.205009
Train	Customer Expectations Score	Safety	0.282822
Taxi	Customer Expectations Score	Service Information	-0.002808
Taxi	Customer Expectations Score	Customer Service	0.109519
Taxi	Customer Expectations Score	Convenience	-0.180264
Taxi	Customer Expectations Score	Comfort	-0.019998
Taxi	Customer Expectations Score	Safety	-0.129835
Taxi	Customer Expectations Score	Affordability	0.014357
Booking A	Customer Expectations Score	Convenience	0.13308
Booking A	Customer Expectations Score	Safety	0.23973
Booking A	Customer Expectations Score	Customer Service	0.059335

c. population_agg_clustering.xlsx

The geospatial population analysis is manually combined and an additional column "Cluster" is added and filled with the cluster name.



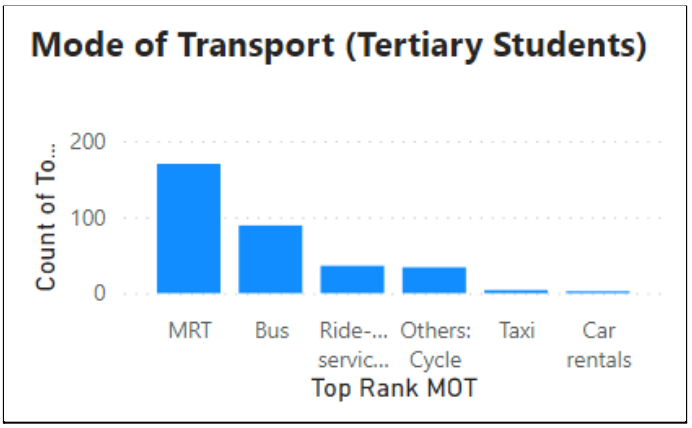
4.3 Visualization

4.3.1 Mode of Transport (Tertiary Students)

Source: *cleaned_survey.csv*

Step 1: Create a clustered column chart

Step 2: Fill axis with “Top Rank MOT” and values as count of “Top Rank MOT”

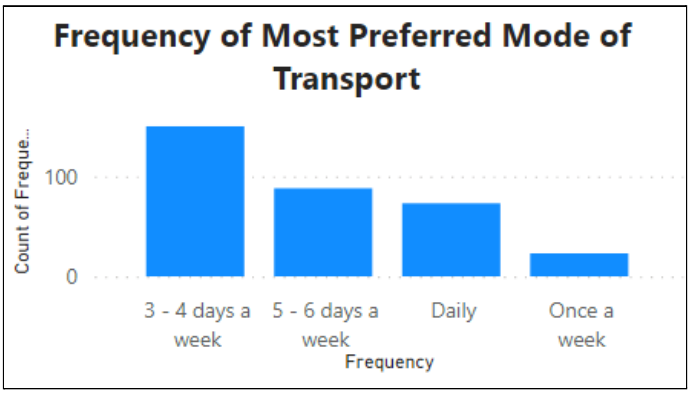


4.3.2 Frequency of Most Preferred Mode of Transport

Source: *cleaned_survey.csv*

Step 1: Create a clustered column chart

Step 2: Fill axis with “Frequency” and values as count of “Frequency”



4.3.3 Most Important Factor in Deciding MOT

Source: *cleaned_survey.csv*

Step 1: Create a table

Step 2: Drag values “Most Important Factor” and change Total to count.

Most Important Factor in Deciding MOT

Most Important Factor	Total
Comfort	8
Physical Accessibility	25
Price	183
Reliability	5
Safety	25
Service Information	2

4.3.4 Average CSISG Scores

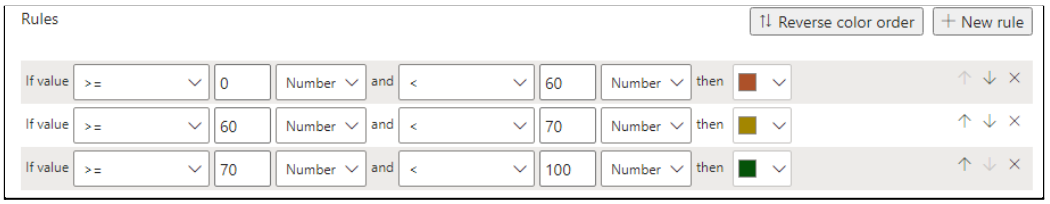
Source: *csisg_agg_combined.xlsx*

Step 1: Create a filled map.

Step 2: Fill location with "Planning Area" and add tooltips as average of "Scoring".

Step 3: Format the colour of different planning areas by setting the default color in the format style "Rules" based on the field of Average of Scoring and summarization "Average".

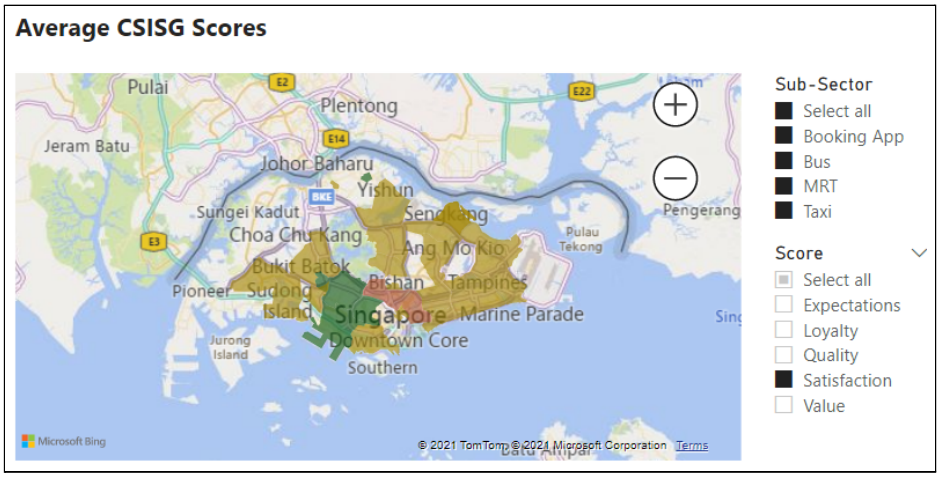
Step 4: Add appropriate rule ranges with red, yellow and green.



Step 5: Ensure "Singapore," is added for "Planning Area".

Sub-Sector	Planning Area	Score	Scoring
Bus	Singapore, ANG MO KIO	Expectations	*71.9533333333333*
Bus	Singapore, BEDOK	Expectations	*71.84125*
Bus	Singapore, BISHAN	Expectations	*61.43*
Bus	Singapore, BUKIT BATOK	Expectations	*66.6585714285714*
Bus	Singapore, BUKIT MERAH	Expectations	*70.73*
Bus	Singapore, BUKIT PANJANG	Expectations	*64.835*
Bus	Singapore, BUKIT TIMAH	Expectations	*74.155*

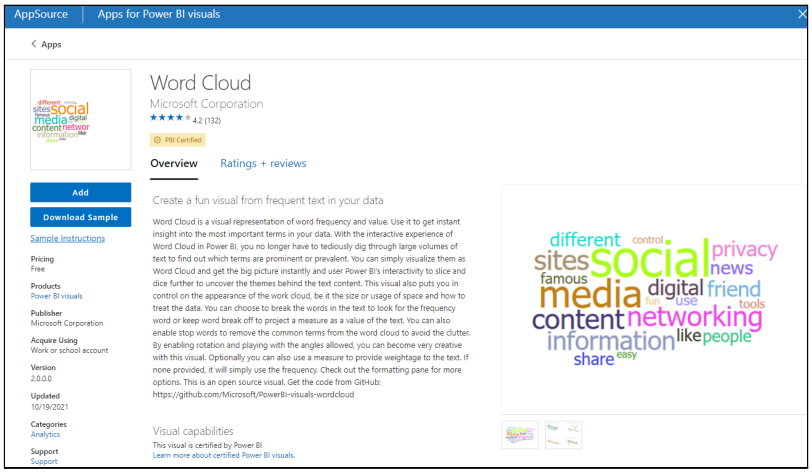
Step 6: To create filters for the map, add slicers with fields "Sub-Sector" and "Score".



4.3.5 Perception About Singapore Transportation

Source: *reddit_csisg_text_combined.xlsx*

Step 1: To get the word cloud plugin, click on "Get More Visuals" and search "Word Cloud" to download.



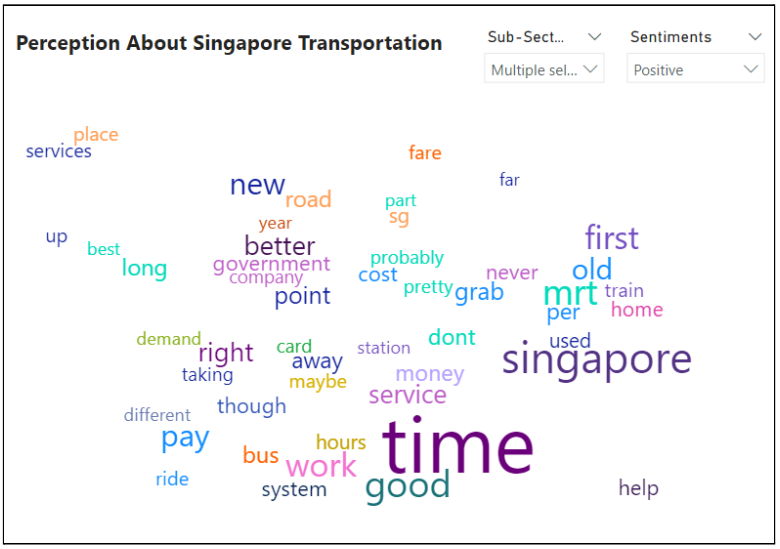
Step 2: Create a word cloud.

Step 3: Fill the category with "Text".

Step 4: Format by turning on "Stop Words" and turning off "Rotate Text" for better visualisation.

Step 5: To set the number of words, format "General" to the preferred number.

Step 6: To create filters, add a slicer with fields “Sub-Sector” and “Sentiments”.

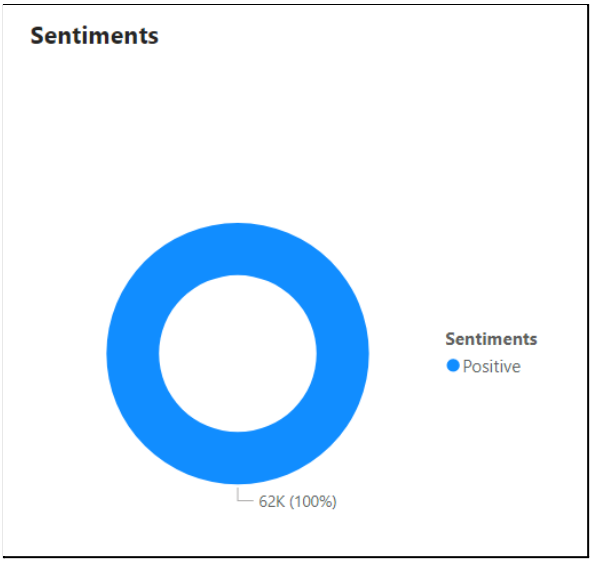


4.3.6 Sentiments

Source: *reddit_csisg_text_combined.xlsx*

Step 1: Create a donut chart

Step 2: Fill values with “Score” and add legend “Sentiments”



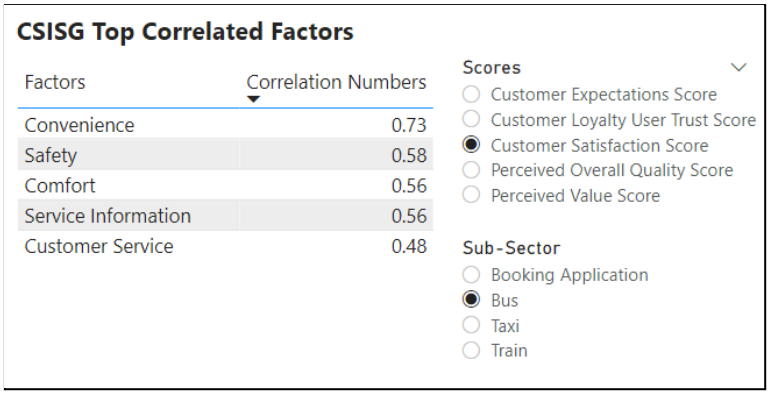
4.3.7 CSISG Top Correlated Factors

Source: *correlation_all.xlsx*

Step 1: Create a table.

Step 2: Fill values with “Factors” and “Correlation Numbers”.

Step 3: To create filters, add slicers with fields “Scores" and “Sub-Sector”.



4.3.8 Survey Projected Population Numbers

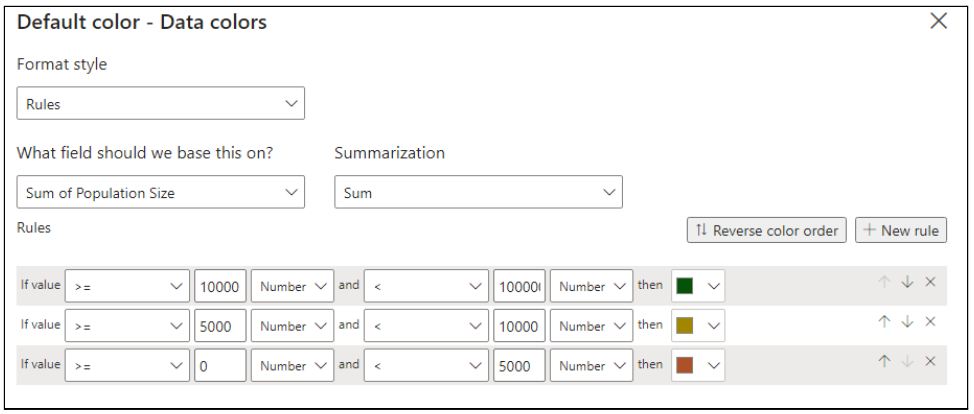
Source: *population_agg_clustering.xlsx*

Step 1: Create a filled map.

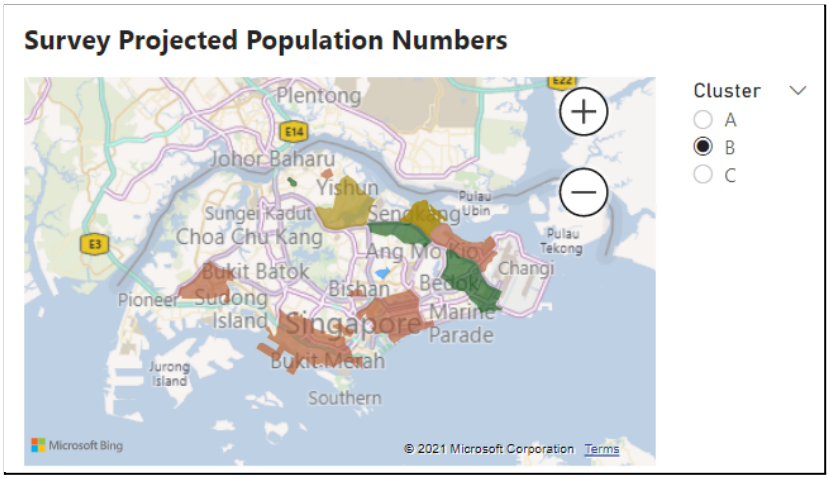
Step 2: Fill location with “Planning Area” and add tooltips as sum of “Population Size”.

Step 3: Format the colour of different planning areas by setting the default color in the format style “Rules” based on Sum of Population Size and summarization “Sum”.

Step 4: Add appropriate rule ranges with red, yellow and green.



Step 5: Create a filter by adding slicers with the field “Cluster”.



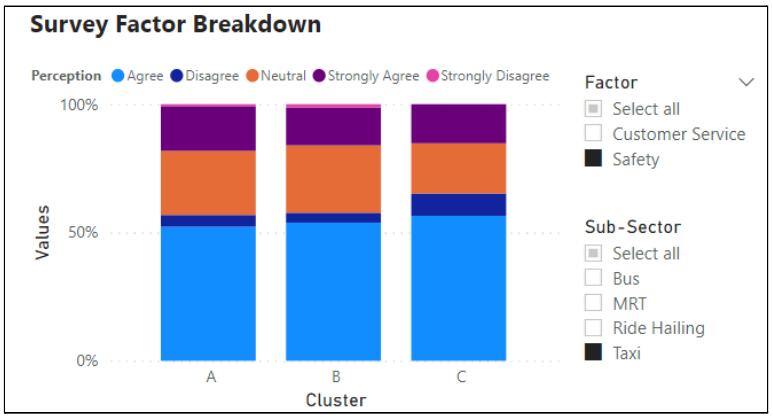
4.3.9 Survey Factor Breakdown

Source: *survey_agg_clustering.xlsx*

Step 1: Create a 100% stacked column chart

Step 2: Fill the axis with "Cluster" and legend as "Perception".

Step 3: Create filters by adding slicers with fields "Factor" and "Sub-Sector".



4.4 Updating data

Step 1: To update values, re-run analysis

Step 2: Go to File → Options and settings → Data source settings

Step 3: Select on the chosen data source → Change Source…

Step 4: Browse and select the updated source and refresh the visualization.