[ANL312 // TG02]

[ECA]

[APPLICATION OF TEXT MINING ON MARKET INTELLIGENCE]

[TAN E-ZHEN // B1881568]

3.2 Data Understanding

The second phase in the CRISP-DM framework is data understanding. This phase is where the data required for the model is collected, and exploratory data analysis is conducted on the collected data to understand it and evaluate the data quality, such as if there are any null or duplicate entries.

3.2.1 Data Collection

For the purposes of this project, a mock-up dataset was created from 4 sources – selected attributes from the Airline Passenger Satisfaction (Klein, 2020) and Customer Churn (Kumar, 2020) datasets on Kaggle, customer reviews posted on Grab Singapore's Facebook page obtained by web scraping, and randomised integer values generated by *numpy* in Python. The attributes in the final dataset are summarised as follows:

Attribute	Measurement	Description
Gender	Categorical	Gender of customer (Male or Female)
Age	Continuous	Age of customer
AccountWeeks	Continuous	Number of weeks that the customer has had an active account
Subscription	Flag	Whether the customer subscribed to the Daily Value Plan (Yes or No)
Reviews	Text	Text of customer review
AvgRides	Continuous	Average number of rides the customer takes per month (over 12-month period or since account opening, whichever is smaller)
AvgSpend	Continuous	Average amount (in SGD) the customer spends on Grab rides per month (over 12-month period or since account opening, whichever is smaller)
LastRating	Categorical	Rating given for the last ride (1, 2, 3, 4, or 5)
Churn	Flag	Whether the customer will book a Grab ride in the next 30 days

Table 1: Attributes of dataset

The steps to collect all the relevant data are as follows:

Step 1: Scrape Grab Singapore's Facebook page

Firstly, to scrape Grab Singapore's Facebook page for the customer reviews, the *selenium* and *BeautifulSoup* packages in Python were used. Although Facebook's Graph API can scrape Facebook pages, this function is now only available to the administrators of the page. As such, for this report, the scraping of the Facebook page had to be done relatively more manually.

One limitation of *BeautifulSoup* is that it can only scrape static webpages. However, for the Facebook page, to view past reviews, the page needs to be scrolled. Hence, the *selenium* package was used to control the browser and automate the scrolling.

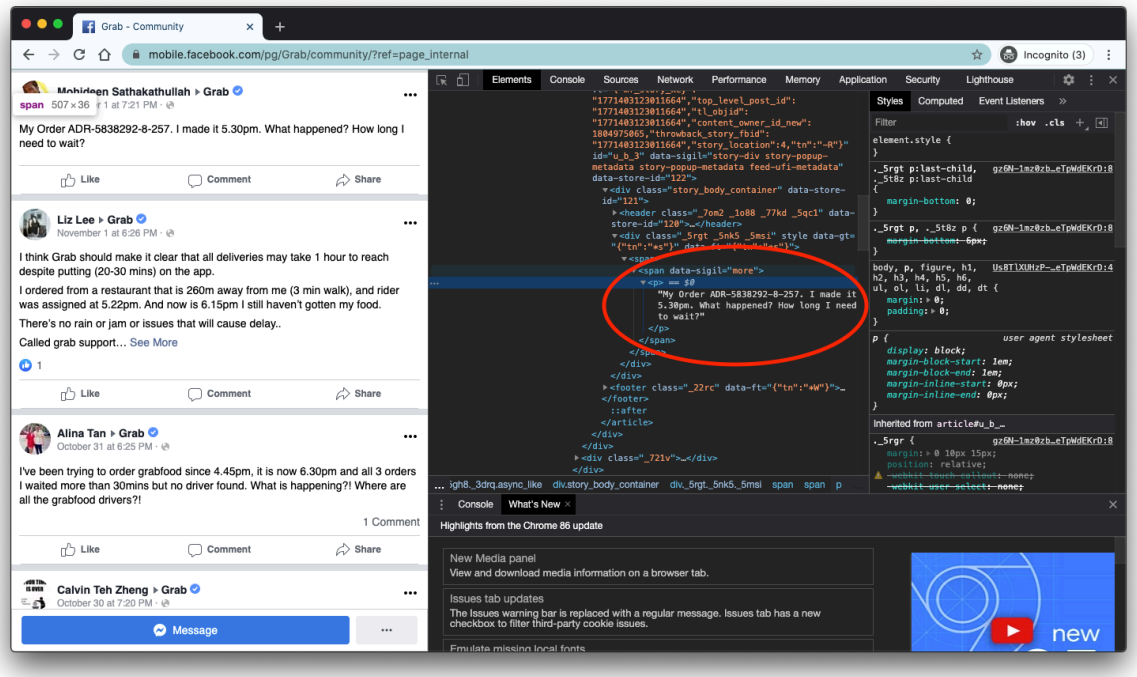


Fig. 2.1: Inspecting the elements of the Facebook page with Google Chrome

In general, for web scraping, the mobile version of websites are easier to scrape, as the websites are made more simply with less JavaScript. Hence, the mobile version of Grab Singapore's Facebook page was scraped for this report. After inspecting the elements of the page with Google Chrome, it can be seen in the above figure that the text are under the tag '' with the attribute 'data-sigil="more"'. This information was then used to scrape the page for the text reviews as follows:

```
import requests
import re
import json
import time
import collections
from bs4 import BeautifulSoup

import csv
import pandas as pd

from credentials import username, password

from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.chrome.options import Options
```

```
payload = {'email': username, 'pass': password} # FaceBook username and password
grab_url = 'https://mobile.facebook.com/pg/Grab/community/?ref=page_internal&mt_nav=0'

class FaceBookBot():

    def __init__(self):
        options = webdriver.ChromeOptions()
        options.add_argument('--disable-notifications')
        self.driver = webdriver.Chrome('../../../chromedriver', options=options)

    def login(self, username, password):
        self.driver.get("https://mobile.facebook.com/login")

        time.sleep(5)

        email_in = self.driver.find_element_by_xpath('//*[@id="m_login_email"]')
        email_in.send_keys(username)

        password_in = self.driver.find_element_by_xpath('//*[@id="m_login_password"]')
        password_in.send_keys(password)

        login_btn = self.driver.find_element_by_xpath('//*[@name="login"]')
        login_btn.click()

        time.sleep(5)
```

```
bot = FaceBookBot()
bot.login(payload['email'], payload['pass'])

bot.driver.get(grab_url)

time.sleep(3)

for i in range(1, 200):
    bot.driver.execute_script('window.scrollTo(0, document.body.scrollHeight);')
    time.sleep(3)

    if i % 10 == 0:
        print(f'{i} times scrolled.')

page = bot.driver.page_source
soup = BeautifulSoup(page, 'html.parser')

contents = soup.find_all('span', attrs={'data-sigil': 'more'})

posts_contents = []
for posts in contents:
    posts_contents.append(posts.text)

print(f'{len(posts_contents)} reviews collected.')

df = pd.DataFrame(data={'grab_reviews': posts_contents})
df.to_csv('grab_reviews.csv', index=False)
```

Fig. 2.2: Code for scraping Grab Singapore's Facebook Page – 'Community' tab

After running the above code, 405 reviews were scraped. However, due to insufficient time in between some of the scrolls, there were several duplicate entries. After removing the duplicates with the *pandas* function `drop_duplicates`, there were 392 unique reviews remaining with no null values.

```
df_grab = pd.read_csv('datasets/grab_reviews.csv')
df_grab.head()
```

	grab_reviews
0	My Order ADR-5838292-8-257. I made it 5.30pm. ...
1	I think Grab should make it clear that all del...
2	I've been trying to order grabfood since 4.45p...
3	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...
4	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...

```
df_grab.drop_duplicates(inplace=True)
df_grab.reset_index(drop=True, inplace=True)
```

```
len(df_grab)
```

```
392
```

```
sum(df_grab['grab_reviews'].isnull())
```

```
0
```

Fig. 2.3: Code to remove duplicates

Step 2: Airline Passenger Satisfaction Dataset

Two attributes in the Airline Passenger Satisfaction dataset obtained from Kaggle (Klein, 2020) were used – 'Gender' and 'Age' – to replicate customer demographics for this project. 392 rows were randomly sampled to match the number of customer reviews scraped in the previous step. In the *pandas* function `sample`, the `random_state` was set to 42 to ensure reproducibility (refer to Appendix A for the screenshot of the code).

Step 3: Customer Churn Dataset

Three attributes in the Customer Churn dataset obtained from Kaggle (Kumar, 2020) were used – 'AccountWeeks', 'DataPlan', and 'Churn' – to replicate customer usage of the Grab ride-hailing platform and the history of customer churn. The 'DataPlan' attribute was then renamed to 'Subscription'.

For this dataset, the distribution of the target, 'Churn', was highly imbalanced. Hence, instead of randomly sampling 392 rows from the entire dataset, half of the rows required were sampled from the rows where 'Churn' = 1, and the other half from the rows where 'Churn' = 0. Similar

to the Airline Passenger Satisfaction dataset, the `random_state` was set to 42 to ensure reproducibility (refer to Appendix A for the screenshot of the code).

Step 4: Joining the above 3 DataFrames and renaming the columns

The *pandas* function `concat` was used to concatenate the three DataFrames into one. The 'DataPlan' and 'grab_reviews' columns were renamed to 'Subscription' and 'Reviews' respectively.

```
data = pd.concat([airline_sample, churn_sample, df_grab], axis=1)
data.head()
```

	Gender	Age	AccountWeeks	DataPlan	Churn	grab_reviews
0	Female	26	157	1	0	My Order ADR-5838292-8-257. I made it 5.30pm. ...
1	Male	22	10	0	0	I think Grab should make it clear that all del...
2	Female	59	65	0	1	I've been trying to order grabfood since 4.45p...
3	Female	32	82	0	1	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...
4	Male	35	95	0	1	Hi Sir Anthony Tan I would like to enquire wi...

```
data.rename(columns={'DataPlan': 'Subscription', 'grab_reviews': 'Reviews'},
            inplace=True)
data.head()
```

	Gender	Age	AccountWeeks	Subscription	Churn	Reviews
0	Female	26	157	1	0	My Order ADR-5838292-8-257. I made it 5.30pm. ...
1	Male	22	10	0	0	I think Grab should make it clear that all del...
2	Female	59	65	0	1	I've been trying to order grabfood since 4.45p...
3	Female	32	82	0	1	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...
4	Male	35	95	0	1	Hi Sir Anthony Tan I would like to enquire wi...

Fig. 2.4: Code to join the DataFrames and rename the columns

Step 4: Randomised Integer Values

To mock-up the inputs for 'AvgRides', 'AvgSpend' and 'LastRating', the `random.randint` function from the *numpy* package was used, with the seed set at 42 to ensure reproducibility.

```
np.random.seed(42)
data['AvgRides'] = np.random.randint(1, 35, data.shape[0])
data['AvgSpend'] = np.random.randint(5, 200, data.shape[0])
data['LastRating'] = np.random.randint(1, 6, data.shape[0])
```

Fig. 2.5: Code to generate random integer values

After going through the above four steps, the dataset for the report was generated. The columns were then reordered for ease of reference and saved for use in the model.

```
data = data[['Gender', 'Age', 'AccountWeeks', 'Subscription', 'Reviews',
             'AvgRides', 'AvgSpend', 'LastRating', 'Churn']]
```

```
cat_dict = {0: 'No', 1: 'Yes'}
data['Subscription'] = data['Subscription'].map(cat_dict)
data['Churn'] = data['Churn'].map(cat_dict)
data.head()
```

	Gender	Age	AccountWeeks	Subscription	Reviews	AvgRides	AvgSpend	LastRating	Churn
0	Female	26	157	Yes	My Order ADR-5838292-8-257. I made it 5.30pm. ...	29	156	4	No
1	Male	22	10	No	I think Grab should make it clear that all del...	15	196	2	No
2	Female	59	65	No	I've been trying to order grabfood since 4.45p...	8	181	3	Yes
3	Female	32	82	No	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...	21	103	1	Yes
4	Male	35	95	No	Hi Sir Anthony Tan I would like to enquire wi...	19	40	5	Yes

Fig. 2.6: Sample of final dataset

3.2.2 Data Exploration

This step of exploring the dataset generated will provide an overview of the dataset, and Grab's customer demographic as a result, and if there are any attributes that are unbalanced. The distribution of all the attributes (excluding 'Reviews') are as follows:

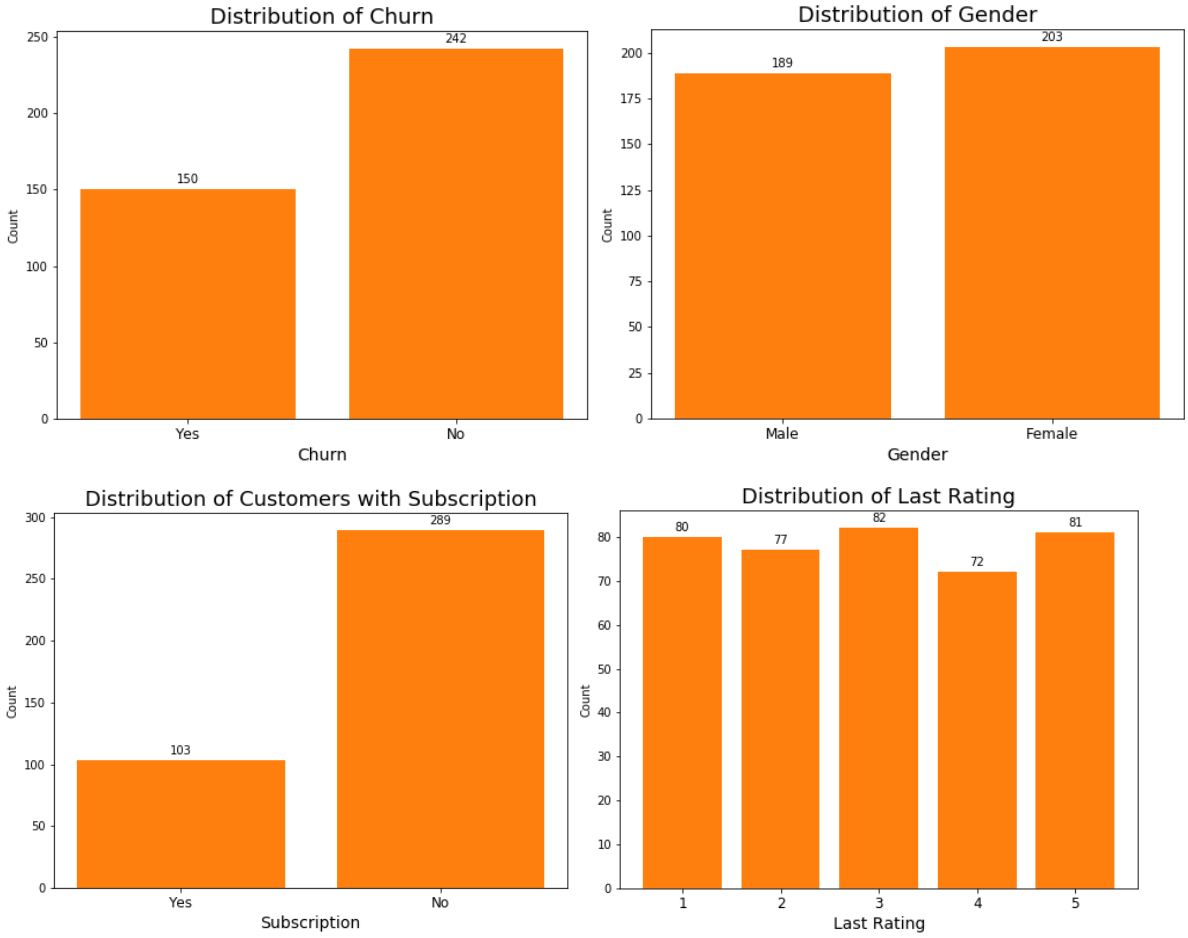
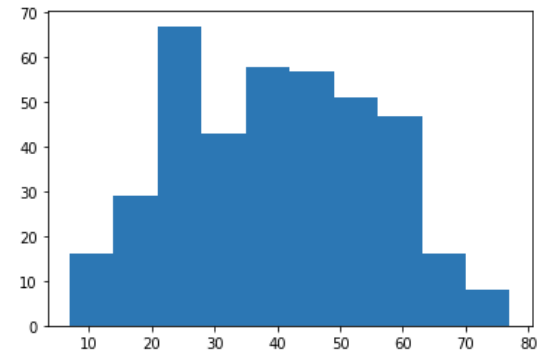
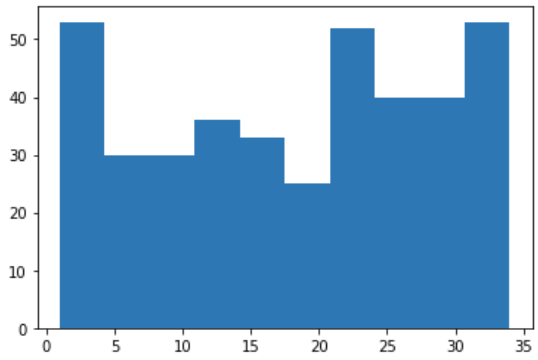


Fig. 2.7: Distribution of categorical attributes

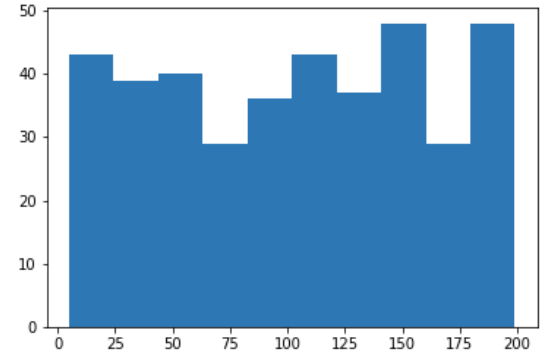
```
plt.hist(data['Age'])
plt.show()
```




```
plt.hist(data['AvgRides'])
plt.show()
```




```
plt.hist(data['AvgSpend'])
plt.show()
```




```
plt.hist(data['AccountWeeks'])
plt.show()
```

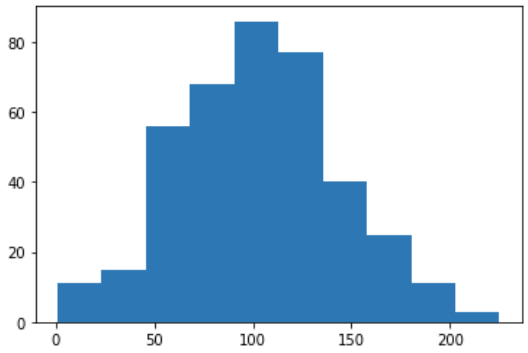


Fig. 2.8: Distribution of continuous attributes

From Fig. 2.8 above, it can be seen that the range of each continuous attribute is different. For example, 'Age' has a much smaller range than 'AvgSpend'. As such, to compare between attributes with small and large ranges, data transformation will be needed in the next stage to normalise the ranges of the continuous attributes.

3.2.3 Data Quality

As seen in the figure below, there are no null values or duplicated rows in the dataset.

```
Number of null values in Gender: 0
Number of null values in Age: 0
Number of null values in AccountWeeks: 0
Number of null values in Subscription: 0
Number of null values in Reviews: 0
Number of null values in AvgRides: 0
Number of null values in AvgSpend: 0
Number of null values in LastRating: 0
Number of null values in Churn: 0
```

```
# check for duplicate rows
duplicate_row = data[data.duplicated()]
duplicate_row
```

Gender	Age	AccountWeeks	Subscription	Reviews	AvgRides	AvgSpend	LastRating	Churn

Fig. 2.9: Python check to ensure no null or duplicated rows

3.3 Data Preparation

The third phase of CRISP-DM is data preparation, which is the most time-consuming phase. In this stage, transformations will be applied to the data where required. For structured data, this may involve scaling the numerical values or changing the data type. For text data, this will be where the pre-processing occurs, such as case normalisation and the removal of stop words.

For the dataset used in this report, there are two types of data – structured and unstructured. As seen in Table 1, other than the 'Reviews' column, all the columns are structured data.

Structured data

For the continuous attributes, as mentioned in the previous section, data transformation will be needed to normalise the variables to a standard range. One method to do so would be to use min-max normalisation, which transforms a variable's original range into a newly specified range, namely [0, 1] in this case. The *scikit-learn* package in Python has a `MinMaxScaler` function that scales each feature to a given range, [0,1] by default.

```
minmax = MinMaxScaler()
x_num_mm = minmax.fit_transform(X_num)
```

```
X_num = pd.DataFrame(x_num_mm)
X_num.rename(columns={0: 'Age', 1: 'AccountWeeks',
                      2: 'AvgRides', 3: 'AvgSpend'}, inplace=True)
X_num
```

	Age	AccountWeeks	AvgRides	AvgSpend
0	0.271429	0.699552	0.848485	0.778351
1	0.214286	0.040359	0.424242	0.984536
2	0.742857	0.286996	0.212121	0.907216
3	0.357143	0.363229	0.606061	0.505155
4	0.400000	0.421525	0.545455	0.180412
...	...	...	...	...
387	0.928571	0.511211	0.848485	0.221649
388	0.271429	0.430493	0.727273	0.123711
389	0.428571	0.358744	0.606061	0.742268
390	0.585714	0.511211	0.272727	0.061856
391	0.642857	0.390135	0.242424	0.123711

392 rows × 4 columns

Fig. 3.1: Min-Max Normalisation of Continuous Attributes

For the categorical attributes, they cannot be directly used as inputs in neural networks. Instead, dummy variables need to be created for each attribute, with each variable being binary.

```
X_cat = pd.get_dummies(X_cat)
X_cat
```

	Gender_Female	Gender_Male	Subscription_No	Subscription_Yes	LastRating_1	LastRating_2
0	1	0	0	1	0	0
1	0	1	1	0	0	1
2	1	0	1	0	0	0
3	1	0	1	0	1	0
4	0	1	1	0	0	0
...	...	...	...	...	...	...
387	1	0	1	0	0	1
388	0	1	1	0	0	0
389	1	0	1	0	0	0
390	1	0	1	0	0	0
391	0	1	1	0	0	0

392 rows × 12 columns

Fig 3.2: Dummy variable for each categorical attribute

However, creating dummy variables will result in the dummy variable trap, where there is an extra variable for each attribute. For example, for 'Gender_Male', 0 would indicate that the customer is a male, while 1 would indicate that the customer is female. As such, 'Gender_Female' is redundant, and should be removed. Similarly, for the 'LastRating' attribute, 'LastRating_5' can be removed, as 0 in all other columns – 'LastRating_1', 'LastRating_2', 'LastRating_3', and 'LastRating_4' – would mean that the customer's last rating was 5. Hence, one dummy variable for each categorical attribute should be dropped, and the dropped variable will then be used as the benchmark.

After transforming both continuous and categorical attributes, the resulting dataset is as follows:

	Age	AccountWeeks	AvgRides	AvgSpend	Gender_Male	Subscription_Yes	LastRating_1	LastRating_2	LastRating_3	LastRating_4
0	0.271429	0.699552	0.848485	0.778351	0	1	0	0	0	1
1	0.214286	0.040359	0.424242	0.984536	1	0	0	1	0	0
2	0.742857	0.286996	0.212121	0.907216	0	0	0	0	1	0
3	0.357143	0.363229	0.606061	0.505155	0	0	1	0	0	0
4	0.400000	0.421525	0.545455	0.180412	1	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...
387	0.928571	0.511211	0.848485	0.221649	0	0	0	1	0	0
388	0.271429	0.430493	0.727273	0.123711	1	0	0	0	0	0
389	0.428571	0.358744	0.606061	0.742268	0	0	0	0	0	0
390	0.585714	0.511211	0.272727	0.061856	0	0	0	0	1	0
391	0.642857	0.390135	0.242424	0.123711	1	0	0	0	1	0

Fig. 3.3: Inputs for modelling

For the target attribute, ‘Churn’, the values were encoded with scikit-learn’s `LabelEncoder`. This function encodes the target labels, ‘Yes’ and ‘No’ with values ‘1’ and ‘0’ respectively.

```
labelencoder_y = LabelEncoder()
Y = labelencoder_y.fit_transform(Y)
Y.reshape(-1, 1)
```

```
array([[0],
       [0],
       [1],
       [1],
       [1],
       [0],
       [1],
       [0],
       [1],
       [0],
       [0],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
```

Fig. 3.4: Encoding target values

Unstructured data

Generally, an objective of the use of the text reviews is to carry out text mining on the text and obtain structured data. The result can then be used in the existing model to compare if the additional information from the text data will improve the accuracy of the predictive model.

For this report, sentiment analysis will be carried out on the text reviews scraped to obtain a categorical attribute of ‘LastReview’, where sentiment will be positive (‘pos’), neutral (‘neu’) or negative (‘neg’).

Additionally, the text reviews were obtained from Facebook, a social media platform. As such, in this report, the VADER tool was used to conduct unsupervised sentiment analysis on each review to obtain the sentiment. The rationale behind using the tool for this report is because the tool is trained for social media text, where it takes into account the emoticons, punctuation, and even commonly used acronyms (Hutto & Gilbert, 2014). As a result of using the tool, text pre-processing, such as case-normalisation, will not be essential, as VADER takes into account the case of the text in determining the sentiment of the text.

```python
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

analyzer = SentimentIntensityAnalyzer()

def vader_sentiment(df_column):
    sentiment_result = []

    for review in df_column:
        vs = analyzer.polarity_scores(review)

        if vs['compound'] >= 0.05:
            sentiment_result.append('pos')

        elif vs['compound'] <= -0.05:
            sentiment_result.append('neg')

        else:
            sentiment_result.append('neu')

    return sentiment_result
```

```python
data['LastReview'] = vader_sentiment(data['Reviews'])
```

```python
data['LastReview'].value_counts()
```

```
pos    191
neg    144
neu     57
Name: LastReview, dtype: int64
```

```python
data.head()
```

	Gender	Age	AccountWeeks	Subscription	Reviews	AvgRides	AvgSpend	LastRating	Churn	LastReview
0	Female	26	157	Yes	My Order ADR-5838292-8-257. I made it 5.30pm. ...	29	156	4	No	neu
1	Male	22	10	No	I think Grab should make it clear that all del...	15	196	2	No	pos
2	Female	59	65	No	I've been trying to order grabfood since 4.45p...	8	181	3	Yes	neg
3	Female	32	82	No	WE SHOULD BOYCOTT GRAB LOUSLY FARES NO INCENTI...	21	103	1	Yes	neg
4	Male	35	95	No	Hi Sir Anthony Tan I would like to enquire wi...	19	40	5	Yes	pos

Fig. 3.5: Code to categorise sentiment of text reviews

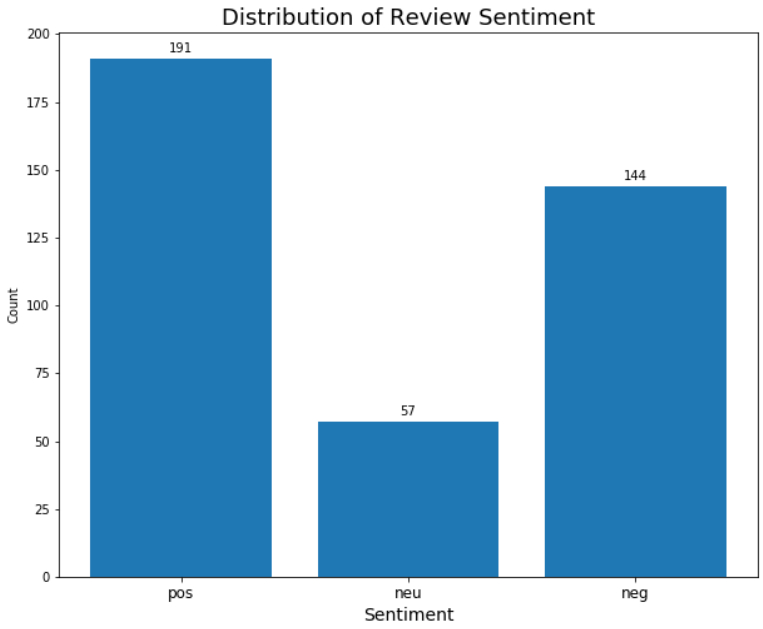


Fig. 3.6: Distribution of Sentiment

With the new categorical attribute ‘LastReview’, the step of creating dummy variables for the attribute was carried out.

ge	AccountWeeks	AvgRides	AvgSpend	Gender_Male	Subscription_Yes	LastRating_1	LastRating_2	LastRating_3	LastRating_4	LastReview_neg	LastReview_
29	0.699552	0.848485	0.778351	0	1	0	0	0	1	0	
86	0.040359	0.424242	0.984536	1	0	0	1	0	0	0	
57	0.286996	0.212121	0.907216	0	0	0	0	1	0	1	
43	0.363229	0.606061	0.505155	0	0	1	0	0	0	1	
00	0.421525	0.545455	0.180412	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	
71	0.511211	0.848485	0.221649	0	0	0	1	0	0	0	
29	0.430493	0.727273	0.123711	1	0	0	0	0	0	1	
71	0.358744	0.606061	0.742268	0	0	0	0	0	0	1	
14	0.511211	0.272727	0.061856	0	0	0	0	1	0	0	
57	0.390135	0.242424	0.123711	1	0	0	0	1	0	1	

12 columns

Fig. 3.7: Inputs for model with sentiment

3.4 Modelling

The modelling phase of the CRISP-DM framework involves applying appropriate models to achieve the data mining goal. More than one model may be needed to compare the performance.

Predicting customer churn is a binary classification problem – whether the customer will take a Grab ride in the next 30 days. After the data preparation phase, all the inputs are now numerical. Hence, the data is suitable for most predictive modelling methods.

Several models were tested – multinomial Naïve Bayes, logistic regression, and neural networks. The training dataset consists of 70% of the dataset, with the remaining 30% used as the test dataset that was unseen by the models. Each model was run twice – once excluding the customer reviews sentiment, and once including the sentiment.

The inputs used are summarised in the following table. The sentiment inputs that were excluded in the first run of each model are indicated with an asterisk (*).

Input	Measurement	Description
Age	Continuous	Age of customer (scaled to range [0, 1])
AccountWeeks	Continuous	Number of weeks that the customer has had an active account (scaled to range [0, 1])
AvgRides	Continuous	Average number of rides the customer takes per month (over 12-month period or since account opening, whichever is smaller) (scaled to range [0, 1])
AvgSpend	Continuous	Average amount (in SGD) the customer spends on Grab rides per month (over 12-month period or since account opening, whichever is smaller) (scaled to range [0, 1])
Gender_Male	Flag	Customer is male (‘0’ = No or ‘1’ = Yes)
Subscription_Yes	Flag	Customer subscribed to the Daily Value Plan (‘0’ = No or ‘1’ = Yes)
LastRating_1	Flag	1-star rating given for the last ride (‘0’ = No or ‘1’ = Yes)
LastRating_2	Flag	2-star rating given for the last ride (‘0’ = No or ‘1’ = Yes)

LastRating_3	Flag	3-star rating given for the last ride (‘0’ = No or ‘1’ = Yes)
LastRating_4	Flag	4-star rating given for the last ride (‘0’ = No or ‘1’ = Yes)
LastReview_neg*	Flag	Sentiment of last review given was negative (‘0’ = No or ‘1’ = Yes)
LastReview_neu*	Flag	Sentiment of last review given was neutral (‘0’ = No or ‘1’ = Yes)
Churn	Flag	Whether the customer will book a Grab ride in the next 30 days (‘0’ = No or ‘1’ = Yes)

Table 2: Inputs used in the models

Multinomial Naïve Bayes

```
model_nb = MultinomialNB(alpha=2)
# pipe_nb = Pipeline([('prep', col_transform_2), ('m', model_nb)])
model_nb.fit(x_train_nos, y_train)
y_pred_nb = model_nb.predict(x_test_nos)
print(f'Accuracy score for Multinomial Naive Bayes model: {accuracy_score(y_test, y_pred_nb):.2f}')
```

```
Accuracy score for Multinomial Naive Bayes model: 0.56
```

```
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.56      1.00      0.72        66
           1       0.00      0.00      0.00        52

    accuracy                           0.56       118
   macro avg       0.28      0.50      0.36       118
weighted avg       0.31      0.56      0.40       118
```

Fig. 4.1: Multinomial Naïve Bayes model without sentiment

```
model_nb = MultinomialNB()
model_nb.fit(x_train_s, y_train)
y_pred_nb = model_nb.predict(x_test_s)
print(f'Accuracy score for Multinomial Naive Bayes model: {accuracy_score(y_test, y_pred_nb):.2f}')
```

```
Accuracy score for Multinomial Naive Bayes model: 0.57
```

```
print(classification_report(y_test, y_pred_nb))
```

```
              precision    recall  f1-score   support

           0       0.57      0.96      0.72        45
           1       0.50      0.06      0.11        34

    accuracy                           0.57        79
   macro avg       0.54      0.51      0.41        79
weighted avg       0.54      0.57      0.45        79
```

Fig. 4.2: Multinomial Naïve Bayes model with sentiment

Logistic Regression

```
logistic = LogisticRegression(solver='lbfgs')
logistic.fit(x_train_nos, y_train)
y_pred_logistic = logistic.predict(x_test_nos)
```

```
print(classification_report(y_test, y_pred_logistic))
```

```
              precision    recall  f1-score   support

           0       0.58      1.00      0.73        66
           1       1.00      0.08      0.14        52

    accuracy                           0.59       118
   macro avg       0.79      0.54      0.44       118
weighted avg       0.76      0.59      0.47       118
```

Fig. 4.3: Logistic Regression model without sentiment

```
logistic_s = LogisticRegression(solver='lbfgs')
logistic_s.fit(x_train_s, y_train)
y_pred_logistic_s = logistic_s.predict(x_test_s)
```

```
print(classification_report(y_test, y_pred_logistic_s))
```

```
              precision    recall  f1-score   support

           0       0.60      0.96      0.74        45
           1       0.71      0.15      0.24        34

    accuracy                           0.61        79
   macro avg       0.66      0.55      0.49        79
weighted avg       0.65      0.61      0.52        79
```

Fig. 4.4: Logistic Regression with sentiment

Neural Network – Multi-layer Perceptron (MLP) Classifier

```
model_nn = MLPClassifier(activation='tanh', solver='adam', alpha = 0.0002, learning_rate_init=0.005)
model_nn.fit(x_train_nos, y_train)
y_pred = model_nn.predict(x_test_nos)
```

```
print(f'Accuracy score for MLPClassifier: {accuracy_score(y_test, y_pred):.2f}')
```

```
Accuracy score for MLPClassifier: 0.56
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.57      0.85      0.68        66
           1       0.50      0.19      0.28        52

    accuracy                           0.56       118
   macro avg       0.54      0.52      0.48       118
weighted avg       0.54      0.56      0.50       118
```

Fig. 4.5: MLP Classifier without sentiment

```
model_nn = MLPClassifier(activation='tanh', solver='adam')
model_nn.fit(x_train_s, y_train)
y_pred = model_nn.predict(x_test_s)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.59      0.89      0.71        45
           1       0.55      0.18      0.27        34

    accuracy                           0.58        79
   macro avg       0.57      0.53      0.49        79
weighted avg       0.57      0.58      0.52        79
```

```
print(f'Accuracy score for MLPClassifier: {accuracy_score(y_test, y_pred):.2f}')
```

```
Accuracy score for MLPClassifier: 0.58
```

Fig. 4.6: MLP Classifier without sentiment

Neural Network – Keras

```
model_nos = Sequential()
model_nos.add(Dense(32, input_dim=x_train_nos.shape[1], activation='relu'))
model_nos.add(Dropout(rate=0.2))
model_nos.add(Dense(64, activation='tanh'))
model_nos.add(Dropout(rate=0.2))
model_nos.add(Dense(1, activation='sigmoid'))
model_nos.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_nos.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_58 (Dense)             (None, 32)                352
_________________________________________________________________
dropout_39 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_59 (Dense)             (None, 64)                2112
_________________________________________________________________
dropout_40 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_60 (Dense)             (None, 1)                 65
=================================================================
Total params: 2,529
Trainable params: 2,529
Non-trainable params: 0
_________________________________________________________________
```

```
t_start = time.time()

history_nos = model_nos.fit(x_train_nos, y_train, epochs=100, batch_size=10)

t_end = time.time()
print(f'Model took {t_end-t_start} seconds to train.')
```

```
Epoch 1/100
274/274 [==============================] - 0s 758us/step - loss: 0.6705 - accuracy: 0.6168
Epoch 2/100
274/274 [==============================] - 0s 115us/step - loss: 0.6464 - accuracy: 0.6387
Epoch 3/100
274/274 [==============================] - 0s 107us/step - loss: 0.6332 - accuracy: 0.6350
Epoch 4/100
274/274 [==============================] - 0s 118us/step - loss: 0.6356 - accuracy: 0.6387
Epoch 5/100
274/274 [==============================] - 0s 96us/step - loss: 0.6483 - accuracy: 0.6350
Epoch 6/100
274/274 [==============================] - 0s 99us/step - loss: 0.6253 - accuracy: 0.6496
Epoch 7/100
274/274 [==============================] - 0s 99us/step - loss: 0.6506 - accuracy: 0.6350
Epoch 8/100
274/274 [==============================] - 0s 97us/step - loss: 0.6289 - accuracy: 0.6569
Epoch 9/100
274/274 [==============================] - 0s 103us/step - loss: 0.6312 - accuracy: 0.6423
Epoch 10/100
```

```
y_pred = model_nos.predict_classes(x_test_nos)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.56      0.74      0.64        66
           1       0.43      0.25      0.32        52

    accuracy                           0.53       118
   macro avg       0.50      0.50      0.48       118
weighted avg       0.50      0.53      0.50       118
```

```
print(f'Accuracy score for NN model: {accuracy_score(y_test, y_pred):.2f}')
```

```
Accuracy score for NN model: 0.53
```

Fig. 4.7: Keras model without sentiment

```
model = Sequential()
model.add(Dense(16, input_dim=x_train_s.shape[1], activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(8, activation='tanh'))
model.add(Dropout(rate=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_61 (Dense)             (None, 16)                208
_________________________________________________________________
dropout_41 (Dropout)         (None, 16)                0
_________________________________________________________________
dense_62 (Dense)             (None, 8)                 136
_________________________________________________________________
dropout_42 (Dropout)         (None, 8)                 0
_________________________________________________________________
dense_63 (Dense)             (None, 1)                 9
=================================================================
Total params: 353
Trainable params: 353
Non-trainable params: 0
_________________________________________________________________
```

```
t_start = time.time()

history = model.fit(x_train_s, y_train,
                    epochs=100, batch_size=200)

t_end = time.time()
print(f'Model took {t_end-t_start} seconds to train.')
```

```
Epoch 1/100
313/313 [==============================] - 0s 550us/step - loss: 0.7377 - accuracy: 0.4505
Epoch 2/100
313/313 [==============================] - 0s 11us/step - loss: 0.7420 - accuracy: 0.4473
Epoch 3/100
313/313 [==============================] - 0s 11us/step - loss: 0.7327 - accuracy: 0.4601
Epoch 4/100
313/313 [==============================] - 0s 11us/step - loss: 0.7148 - accuracy: 0.5048
Epoch 5/100
313/313 [==============================] - 0s 12us/step - loss: 0.7110 - accuracy: 0.4952
Epoch 6/100
313/313 [==============================] - 0s 11us/step - loss: 0.7049 - accuracy: 0.5240
Epoch 7/100
313/313 [==============================] - 0s 11us/step - loss: 0.6965 - accuracy: 0.5176
Epoch 8/100
313/313 [==============================] - 0s 12us/step - loss: 0.7013 - accuracy: 0.5208
Epoch 9/100
313/313 [==============================] - 0s 10us/step - loss: 0.6978 - accuracy: 0.5527
Epoch 10/100
```

```
y_pred = model.predict_classes(x_test_s)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.58      1.00      0.73        45
           1       1.00      0.03      0.06        34

    accuracy                           0.58        79
   macro avg       0.79      0.51      0.39        79
weighted avg       0.76      0.58      0.44        79
```

```
print(f'Accuracy score for NN model: {accuracy_score(y_test, y_pred):.2f}')
```

```
Accuracy score for NN model: 0.58
```

Fig. 4.8: Keras model with sentiment

3.5 Evaluation

The fifth phase of the framework is the evaluation of the models trained and tested in the previous phase to select the best model that has the best performance to address the data mining problem.

Generally, the model selected should have a higher accuracy than the baseline accuracy, which is the percentage of the majority class in the dataset. This is because this would be the accuracy of a model that only predicts the majority class. For this dataset, the baseline accuracy will be 61%. As the distribution of churn in the dataset is relatively balanced, the accuracy score will be an appropriate measure to evaluate model performance.

The table below summarises the metrics in the figures in the previous section:

Model		**Accuracy**
Multinomial Naïve Bayes	*Without sentiment*	56%
	With sentiment	57%
Logistic Regression	*Without sentiment*	59%
	With sentiment	61%
Neural Network – MLP Classifier	*Without sentiment*	56%
	With sentiment	58%
Neural Network – Keras	*Without sentiment*	53%
	With sentiment	58%

Table 3: Summary of model performance

From Table 3, it can be concluded that the additional sentiment input does help with increasing the accuracy of the model. For the Keras model, it increased the accuracy by 5 percentage points. Overall, the best model is logistic regression including the sentiment input, with an accuracy of 61%, which is on par with the baseline.