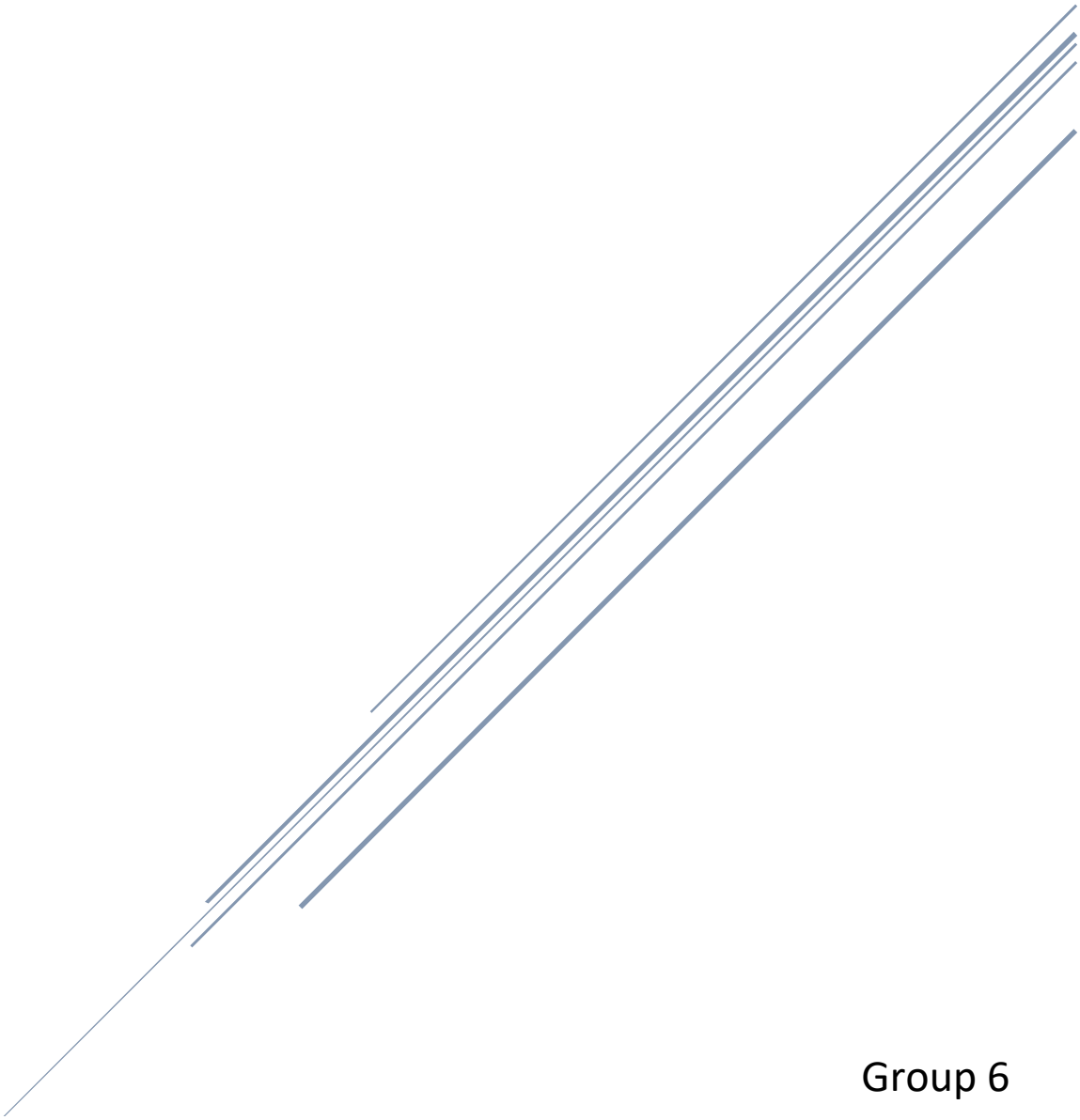
MACHINE LEARNING

GDipSA48, CA



Group 6

1) Ganesh Baskaran (E0384989)
2) Gao Yukun (E0390019)
3) Khin Thu Zar Zaw (E0395896)
4) Lai Weixin (E0390031)
5) Lee Hwa Yang David (E0388058)
6) Liu Ruiming (E0387088)
7) Ng Hui Ling Iris (E0390005)
8) Reshma R Nair (E0384926)

Table of Contents

1. Preliminaries

1.1 Supervised Machine Learning

1.1.1 Data Source

i. https://www.kaggle.com/johnddddd/customer-satisfaction

ii. US Airline passenger satisfaction survey for year 2015

iii. 129,880 records, 24 columns

1. Id
2. **Satisfaction (Dependent Variable)**
3. Gender
4. Customer Type
5. Age
6. Type of Travel
7. Class
8. Flight Distance
9. Inflight Wifi Service
10. Departure/Arrival Time Convenient
11. Ease of Online Booking
12. Gate Location
13. Food and Drink
14. Online Boarding
15. Seat Comfort
16. Inflight Entertainment
17. On-board Service
18. Leg-Room Service
19. Baggage Handling
20. Checkin Service
21. Inflight Service
22. Cleanliness
23. Departure Delay in Minutes
24. Arrival Delay in Minutes

1.1.2 Problem Statement

To be able to classify / predict customer satisfaction with over 90% accuracy from other measurable parameters

1.2 Unsupervised Machine Learning Data Source

1.2.1 Data Source

i. https://www.kaggle.com/pavanraj159/predicting-a-pulsar-star

ii. Prediction of pulsar stars

iii. 17,898 records, 8 columns

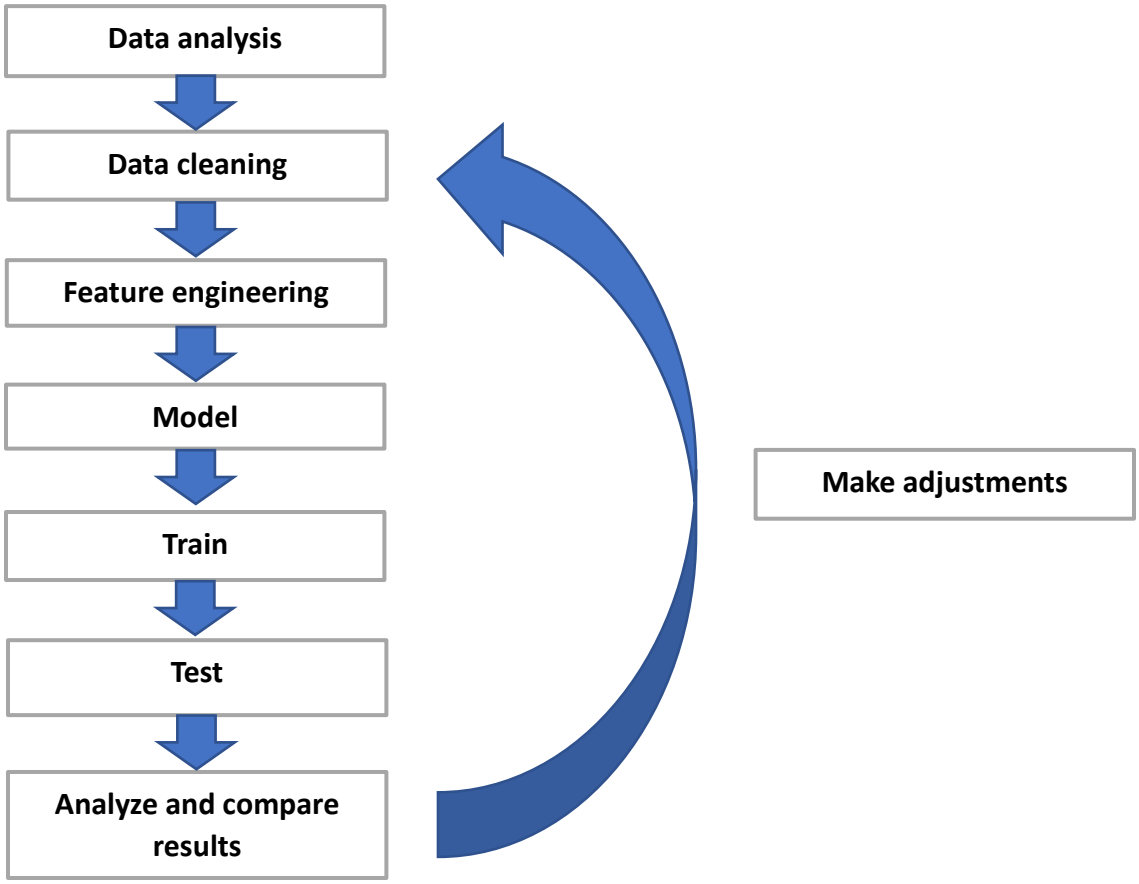
1. Mean of the integrated profile
2. Standard deviation of the integrated profile
3. Excess kurtosis of the integrated profile
4. Skewness of the integrated profile
5. Mean of the DM-SNR curve
6. Standard deviation of the DM-SNR curve
7. Excess kurtosis of the DM-SNR curve
8. Skewness of the DM-SNR curve
9. **Target Class (Dependent variable)**

Note: Several datasets were tried but this was selected as it provided many interesting visual insights into the data

1.2.2 Problem Statement

To be able to determine if a particular radio-wave signal gives a positive identification of a pulsar star or whether it is due to radio frequency interference

1.3 High Level Overview of Machine Learning Process Applied



1.3.1 Data Cleaning

i. Removing null data
   a. By using "**dataframe.dropna()**"
ii. Binning (Conversion of continuous data to categorical data)
   a. Make a copy of the dataframe and map the values
iii. Scaling (Standardizing values)
   a. By using "**StandardScalar().fit_transform(dataframe)**"

1.3.2 Feature Engineering

i. Identify correlations between features and outputs:
   a. Generate heatmap or correlation table
   b. Remove features according to the following criteria:
      i. Features that have a **strong correlation (>0.8)** with other features.
      ii. Features that have a **weak correlation (<0.1)** with other features.

2. Supervised Machine Learning

2.1 Logistic Regression

2.1.1 Code

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import datetime
```

```
# read in data from sample file
df = pd.read_excel('satisfaction_2015.xlsx')
df.head() # show the first five rows
```

```
df.shape
```

```
df.info()
```

```
df.isnull().sum()
```

```
df.describe()
```

```
#convert categorical values into numerical values
df_cat = df
#satisfied=1, dissatisfied=0
df_cat.loc[df['satisfaction_v2'] == 'neutral or dissatisfied',␣
 ↪'satisfaction_v2'] = 0
df_cat.loc[df['satisfaction_v2'] == 'satisfied', 'satisfaction_v2'] = 1
#male=1, female=0
df_cat.loc[df['Gender'] == 'Female', 'Gender'] = 0
df_cat.loc[df['Gender'] == 'Male', 'Gender'] = 1
#loyal=1, disloyal=0
df_cat.loc[df['Customer Type'] == 'disloyal Customer', 'Customer Type'] = 0
df_cat.loc[df['Customer Type'] == 'Loyal Customer', 'Customer Type'] = 1
#Personal Travel=0, Business Travel=1
df_cat.loc[df['Type of Travel'] == 'Personal Travel', 'Type of Travel'] = 0
df_cat.loc[df['Type of Travel'] == 'Business travel', 'Type of Travel'] = 1
#Eco=0, Eco Plus=1, Business=2
df_cat.loc[df['Class'] == 'Eco', 'Class'] = 0
df_cat.loc[df['Class'] == 'Eco Plus', 'Class'] = 1
df_cat.loc[df['Class'] == 'Business', 'Class'] = 2

df_cat.info()
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 2:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
X_train
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
print(datetime.datetime.now())
logReg = LogisticRegression(solver = 'lbfgs', max_iter = 10000)
logReg.fit(X_train, y_train)
print(datetime.datetime.now())
```

```
y_pred=logReg.predict(X_test)
y_pred.shape
```

```
#LogReg result, before removing null data, binning, scaling, and feature␣
 ↪engineering
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
#removing null data
df_cat=df_cat.dropna()
df_cat.shape
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 2:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
print(datetime.datetime.now())
logReg = LogisticRegression(solver = 'lbfgs', max_iter=10000)
logReg.fit(X_train, y_train)
print(datetime.datetime.now())
```

```
y_pred=logReg.predict(X_test)
y_pred.shape
```

```
#LogReg result, before binning, scaling, and feature engineering
accuracy_score(y_test, y_pred)
```

```
#Binning

#Categorize age, by an interval of 10 years
df_cat.loc[df['Age'] < 10, 'Age'] = 0
```

```
df_cat.loc[(df['Age'] >= 10) & (df['Age'] < 20), 'Age'] = 1
df_cat.loc[(df['Age'] >= 20) & (df['Age'] < 30), 'Age'] = 2
df_cat.loc[(df['Age'] >= 30) & (df['Age'] < 40), 'Age'] = 3
df_cat.loc[(df['Age'] >= 40) & (df['Age'] < 50), 'Age'] = 4
df_cat.loc[(df['Age'] >= 50) & (df['Age'] < 60), 'Age'] = 5
df_cat.loc[(df['Age'] >= 60) & (df['Age'] < 70), 'Age'] = 6
df_cat.loc[(df['Age'] >= 70) & (df['Age'] < 80), 'Age'] = 7
df_cat.loc[(df['Age'] >= 80) & (df['Age'] < 90), 'Age'] = 8
#Categorize flight distance, by an interval of 500
df_cat.loc[df['Flight Distance'] < 500, 'Flight Distance'] = 0
df_cat.loc[(df['Flight Distance'] >= 500) & (df['Flight Distance'] < 1000),␣
 ↪'Flight Distance'] = 1
df_cat.loc[(df['Flight Distance'] >= 1000) & (df['Flight Distance'] < 1500),␣
 ↪'Flight Distance'] = 2
df_cat.loc[(df['Flight Distance'] >= 1500) & (df['Flight Distance'] < 2000),␣
 ↪'Flight Distance'] = 3
df_cat.loc[df['Flight Distance'] >= 2000, 'Flight Distance'] = 4
#Categorize Departure Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Departure Delay in Minutes'] < 7.5, 'Departure Delay in␣
 ↪Minutes'] = 0
df_cat.loc[(df['Departure Delay in Minutes'] >= 7.5) & (df['Departure Delay in␣
 ↪Minutes'] < 15), 'Departure Delay in Minutes'] = 1
df_cat.loc[(df['Departure Delay in Minutes'] >= 15) & (df['Departure Delay in␣
 ↪Minutes'] < 22.5), 'Departure Delay in Minutes'] = 2
df_cat.loc[(df['Departure Delay in Minutes'] >= 22.5) & (df['Departure Delay in␣
 ↪Minutes'] < 30), 'Departure Delay in Minutes'] = 3
df_cat.loc[df['Departure Delay in Minutes'] >= 30, 'Departure Delay in␣
 ↪Minutes'] = 4
#Categorize Arrival Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Arrival Delay in Minutes'] < 7.5, 'Arrival Delay in Minutes'] = 0
df_cat.loc[(df['Arrival Delay in Minutes'] >= 7.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 15), 'Arrival Delay in Minutes'] = 1
df_cat.loc[(df['Arrival Delay in Minutes'] >= 15) & (df['Arrival Delay in␣
 ↪Minutes'] < 22.5), 'Arrival Delay in Minutes'] = 2
df_cat.loc[(df['Arrival Delay in Minutes'] >= 22.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 30), 'Arrival Delay in Minutes'] = 3
df_cat.loc[df['Arrival Delay in Minutes'] >= 30, 'Arrival Delay in Minutes'] = 4
df_cat.head()
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 2:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
print(datetime.datetime.now())
logReg = LogisticRegression(solver = 'lbfgs')
logReg.fit(X_train, y_train)
print(datetime.datetime.now())
```

```
y_pred=logReg.predict(X_test)
y_pred.shape
```

```
#LogReg result, before scaling, and feature engineering
accuracy_score(y_test, y_pred)
```

```
#feature scaling
from sklearn.preprocessing import StandardScaler
features = ['Gender', 'Customer Type', 'Age', 'Type of Travel', 'Class','Flight␣
 ↪Distance','Inflight wifi service',
            'Departure/Arrival time convenient','Ease of Online booking','Gate␣
 ↪location','Food and drink','Online boarding',
           'Seat comfort','Inflight entertainment','On-board service','Leg room␣
 ↪service','Baggage handling','Checkin service',
           'Inflight service','Cleanliness','Departure Delay in␣
 ↪Minutes','Arrival Delay in Minutes']
# Separating out the features
x = df_cat.loc[:, features].values
# Separating out the target
y = df_cat.loc[:,['satisfaction_v2']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

pd.DataFrame(data=x, columns = features).head()
```

```
from sklearn.model_selection import train_test_split

#X = df_cat.iloc[:, 0:22]
#y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 42)
pd.DataFrame(data=X_test, columns = features).head()
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
print(datetime.datetime.now())
logReg = LogisticRegression(solver = 'lbfgs')
logReg.fit(X_train, y_train)
print(datetime.datetime.now())
```

```
y_pred=logReg.predict(X_test)
y_pred.shape
```

```
#LogReg result, before feature engineering
accuracy_score(y_test, y_pred)
```

```
#Feature engineering
plt.matshow(df_cat.corr())
```

```
import seaborn as sns
sns.set() # setting seaborn default for plots
```

```
corr = df_cat.corr()
sns.heatmap(corr)
```

```
corr.style.background_gradient(cmap='coolwarm')
```

```
#drop features whose correlation with satisfaction < 0.1, and features whose
 ↪correlation with other features > 0.8
df_cat.drop('id', axis=1, inplace=True)
df_cat.drop('Gender', axis=1, inplace=True)
df_cat.drop('Arrival Delay in Minutes', axis=1, inplace=True)
df_cat.drop('Departure/Arrival time convenient', axis=1, inplace=True)
df_cat.drop('Gate location', axis=1, inplace=True)
df_cat.drop('Departure Delay in Minutes', axis=1, inplace=True)
df_cat.head()
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 2:22]
y = df_cat['satisfaction_v2']
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
print(datetime.datetime.now())
logReg = LogisticRegression(solver = 'lbfgs')
logReg.fit(X_train, y_train)
print(datetime.datetime.now())
```

```
y_pred=logReg.predict(X_test)
y_pred.shape
```

```
#LogReg result
accuracy_score(y_test, y_pred)
```

2.1.2 Data Dictionary

As shown in code above

2.1.3 Number of records in the dataset

129,487 records (after dropping records with blank cells)

2.1.4 Number of records in the training and test datasets

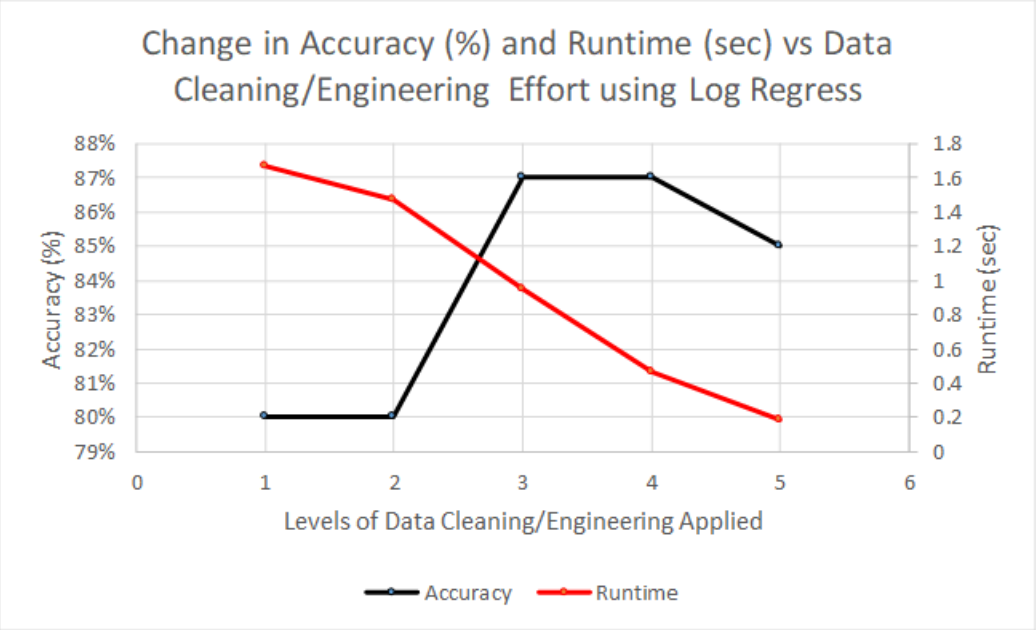
Training dataset: 97,115

Test dataset: 32,372

2.1.5 Training duration

Refer to 2.1.6 below

2.1.6 Improvements after data engineering, feature engineering, model engineering



S/N	Technique Applied
1	None
2	Drop null data
3	Drop null data & binning
4	Drop null data, binning, scaling
5	Drop null data, binning, scaling, feature engineering

2.1.7 Other findings and observations

S/N	C value	solver	Iterations	Accuracy	Precision	Recall	Duration (sec)	Remarks
1	1	lbfgs	100	0.78987	0.74191	0.79106	1.87	
2	100	lbfgs	100	0.78639	0.73275	0.79942	1.65	No convergence. Need to increase iteration
3	0.01	lbfgs	10000	0.87319	0.86939	0.83299	13.35	
4	100	lbfgs	10000	0.87366	0.87008	0.83334	19	
5	10000	lbfgs	10000	0.87337	0.869218	0.83367	14.51	
6	1	lbfgs	10000	0.8735	0.86976	0.83343	17.62	
7	1	lbfgs	1000000	0.8735	0.86976	0.83343	17.79	

Table 2.1.8.1 Variation of C-value and Number of Iterations using lbfgs solver

S/N	C value	solver	Iterations	Accuracy	Precision	Recall	Duration (sec)	Remarks
1	0.1	newton-cg	100	0.87273	0.86892	0.83233	4.09	
2	0.5	newton-cg	100	0.87309	0.86931	0.83281	4.38	
3	1	newton-cg	100	0.87317	0.86943	0.83287	4.3	
4	1	newton-cg	10000	0.87317	0.86943	0.83287	4.24	
5	1	newton-cg	1000000	0.87317	0.86943	0.83287	4.31	
6	100	newton-cg	10000	0.87319	0.86948	0.83268	4.44	
7	10000	newton-cg	10000	0.87319	0.86948	0.83268	4.49	

Table 2.1.8.2 Variation of C-value and Number of Iterations using newton-cg solver

i. Using all the available dimensions, the maximum achievable accuracy is ~ 87% using both lbfgs and newton-cg solver.
ii. The newton-cg solver seems to converge significantly faster requiring less than 5 secs compared to the lbfgs solver that can require around 3 times longer.

2.2 K-NN Classification

2.2.1 Code

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
```

```
# read in data from sample file
df = pd.read_excel('satisfaction_2015.xlsx')
df.head() # show the first five rows
```

```
df.shape
```

```
df.info()
```

```
df.isnull().sum()
```

```
df.describe()
```

```
#drop null-value entries
df=pd.read_excel('satisfaction_2015.xlsx').dropna()
df.info()
```

```
df.corr()
```

```
#categorical variable and binning

df_cat = df
#satisfied=1, dissatisfied=0
df_cat.loc[df['satisfaction_v2'] == 'neutral or dissatisfied',␣
 ↪'satisfaction_v2'] = 0
df_cat.loc[df['satisfaction_v2'] == 'satisfied', 'satisfaction_v2'] = 1
#male=1, female=0
df_cat.loc[df['Gender'] == 'Female', 'Gender'] = 0
df_cat.loc[df['Gender'] == 'Male', 'Gender'] = 1
#loyal=1, disloyal=0
df_cat.loc[df['Customer Type'] == 'disloyal Customer', 'Customer Type'] = 0
df_cat.loc[df['Customer Type'] == 'Loyal Customer', 'Customer Type'] = 1
#Personal Travel=0, Business Travel=1
df_cat.loc[df['Type of Travel'] == 'Personal Travel', 'Type of Travel'] = 0
df_cat.loc[df['Type of Travel'] == 'Business travel', 'Type of Travel'] = 1
#Eco=0, Eco Plus=1, Business=2
df_cat.loc[df['Class'] == 'Eco', 'Class'] = 0
df_cat.loc[df['Class'] == 'Eco Plus', 'Class'] = 1
df_cat.loc[df['Class'] == 'Business', 'Class'] = 2
```

```
#Categorize age, by an interval of 10 years
df_cat.loc[df['Age'] < 10, 'Age'] = 0
df_cat.loc[(df['Age'] >= 10) & (df['Age'] < 20), 'Age'] = 1
df_cat.loc[(df['Age'] >= 20) & (df['Age'] < 30), 'Age'] = 2
df_cat.loc[(df['Age'] >= 30) & (df['Age'] < 40), 'Age'] = 3
df_cat.loc[(df['Age'] >= 40) & (df['Age'] < 50), 'Age'] = 4
df_cat.loc[(df['Age'] >= 50) & (df['Age'] < 60), 'Age'] = 5
df_cat.loc[(df['Age'] >= 60) & (df['Age'] < 70), 'Age'] = 6
df_cat.loc[(df['Age'] >= 70) & (df['Age'] < 80), 'Age'] = 7
df_cat.loc[(df['Age'] >= 80) & (df['Age'] < 90), 'Age'] = 8
#Categorize flight distance, by an interval of 500
df_cat.loc[df['Flight Distance'] < 500, 'Flight Distance'] = 0
df_cat.loc[(df['Flight Distance'] >= 500) & (df['Flight Distance'] < 1000),␣
 ↪'Flight Distance'] = 1
df_cat.loc[(df['Flight Distance'] >= 1000) & (df['Flight Distance'] < 1500),␣
 ↪'Flight Distance'] = 2
df_cat.loc[(df['Flight Distance'] >= 1500) & (df['Flight Distance'] < 2000),␣
 ↪'Flight Distance'] = 3
df_cat.loc[df['Flight Distance'] >= 2000, 'Flight Distance'] = 4
#Categorize Departure Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Departure Delay in Minutes'] < 7.5, 'Departure Delay in␣
 ↪Minutes'] = 0
df_cat.loc[(df['Departure Delay in Minutes'] >= 7.5) & (df['Departure Delay in␣
 ↪Minutes'] < 15), 'Departure Delay in Minutes'] = 1
df_cat.loc[(df['Departure Delay in Minutes'] >= 15) & (df['Departure Delay in␣
 ↪Minutes'] < 22.5), 'Departure Delay in Minutes'] = 2
df_cat.loc[(df['Departure Delay in Minutes'] >= 22.5) & (df['Departure Delay in␣
 ↪Minutes'] < 30), 'Departure Delay in Minutes'] = 3
df_cat.loc[df['Departure Delay in Minutes'] >= 30, 'Departure Delay in␣
 ↪Minutes'] = 4
#Categorize Arrival Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Arrival Delay in Minutes'] < 7.5, 'Arrival Delay in Minutes'] = 0
df_cat.loc[(df['Arrival Delay in Minutes'] >= 7.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 15), 'Arrival Delay in Minutes'] = 1
df_cat.loc[(df['Arrival Delay in Minutes'] >= 15) & (df['Arrival Delay in␣
 ↪Minutes'] < 22.5), 'Arrival Delay in Minutes'] = 2
df_cat.loc[(df['Arrival Delay in Minutes'] >= 22.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 30), 'Arrival Delay in Minutes'] = 3
df_cat.loc[df['Arrival Delay in Minutes'] >= 30, 'Arrival Delay in Minutes'] = 4
df_cat.head(100)
```

```
df_cat.isnull().sum()
```

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

```
import seaborn as sns
sns.set() # setting seaborn default for plots
```

```
def bar_chart(feature):
    satisfied = df_cat[df_cat['satisfaction_v2']==1][feature].value_counts()
    dissatisfied = df_cat[df_cat['satisfaction_v2']==0][feature].value_counts()
    df1 = pd.DataFrame([satisfied,dissatisfied])
    df1.index = ['satisfied','dissatisfied']
    print(df1)
    df1.plot(kind='bar',stacked=True, figsize=(10,5))
```

```
bar_chart('Gender')
#this feature does not significantly affect the satisfaction level
```

```
bar_chart('Customer Type')
#disloyal customers(0) have a higher chance to give a negative rating
```

```
bar_chart('Age')
#Age group affects satisfaction level.
```

```
bar_chart('Type of Travel')
#This affects satisfaction level
```

```
bar_chart('Class')
#This affects satisfaction level
```

```
bar_chart('Flight Distance')
#This affects satisfaction level
```

```
bar_chart('Inflight wifi service')
#This affects satisfaction level
```

```
bar_chart('Departure/Arrival time convenient')
#This affects satisfaction level, but seems not much
```

```
bar_chart('Ease of Online booking')
#This affects satisfaction level
```

```
bar_chart('Gate location')
#This affects satisfaction level, but seems not much
```

```
bar_chart('Food and drink')
#This affects satisfaction level
```

```
bar_chart('Online boarding')
#This affects satisfaction level a lot
```

```
bar_chart('Seat comfort')
#This affects satisfaction level a lot
```

```
bar_chart('Inflight entertainment')
```

```
bar_chart('On-board service')
#This affects satisfaction level
```

```
bar_chart('Leg room service')
#This affects satisfaction level
```

```
bar_chart('Baggage handling')
#This affects satisfaction level a lot
```

```
bar_chart('Checkin service')
#This affects satisfaction level a lot
```

```
bar_chart('Inflight service')
#This affects satisfaction level
```

```
bar_chart('Cleanliness')
#This affects satisfaction level
```

```
bar_chart('Departure Delay in Minutes')
#This affects satisfaction level
```

```
bar_chart('Arrival Delay in Minutes')
```

```
#drop unnecessary features
df_cat.drop('id', axis=1, inplace=True)
df_cat.drop('Gender', axis=1, inplace=True)
df_cat.head()
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
```

```
from pandas.plotting import scatter_matrix

scatter_matrix(df_cat.iloc[:,1:22], alpha=0.2, figsize=(25, 25), diagonal='kde')
```

```
plt.matshow(df_cat.corr())
```

```
#sns.heatmap(df_cat.corr)
```

```
#rs = np.random.RandomState(0)
#df = pd.DataFrame(rs.rand(22, 22))
corr = df_cat.corr()
corr.style.background_gradient(cmap='coolwarm')
```

```
#for using all columns
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 0:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
# import the model from sklearn
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score

# we choose k = 3 here, due to the small number of data we have
# there isn't a k suitable for all. Generally we want to choose a small odd␣
 ↪number.
# large k is generally more costy
knn = KNeighborsClassifier(n_neighbors = 5)

knn.fit(X_train, y_train) #
```

```
y_pred = knn.predict(X_test)
y_pred
```

```
accuracy_score(y_test, y_pred)
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_matrix(y_test,y_pred)
```

```
#for dropping unnecessary features
df_cat.drop('Arrival Delay in Minutes', axis=1, inplace=True)
df_cat.drop('Departure/Arrival time convenient', axis=1, inplace=True)
df_cat.drop('Gate location', axis=1, inplace=True)
df_cat.drop('Departure Delay in Minutes', axis=1, inplace=True)
df_cat.head()
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 0:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
# import the model from sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# we choose k = 3 here, due to the small number of data we have
# there isn't a k suitable for all. Generally we want to choose a small odd␣
 ↪number.
# large k is generally more costy
knn = KNeighborsClassifier(n_neighbors = 5)

knn.fit(X_train, y_train) # train our model to the data set we have. The␣
 ↪training set is all the data points
```

```
y_pred = knn.predict(X_test)
y_pred
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_matrix(y_test,y_pred)
```

```
y_test
```

```
#KNN result
accuracy_score(y_test, y_pred)
```

```
df_cat['satisfaction_v2'].value_counts()
```

```
#for balancing data
df_cat=df_cat.drop(df_cat.query('satisfaction_v2 == 0').sample(frac=0.3).index)
df_cat['satisfaction_v2'].value_counts()
```

```
from sklearn.model_selection import train_test_split

X = df_cat.iloc[:, 0:22]
y = df_cat['satisfaction_v2']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
X_train.head(10)
```

```
print(y_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
```

```
y_pred = knn.predict(X_test)
y_pred
```

```
from sklearn.metrics import confusion_matrix,accuracy_score
confusion_matrix(y_test,y_pred)
```

```
accuracy_score(y_test, y_pred)
```

2.2.2 Data Dictionary

As shown in code above

2.2.3 Number of records in the dataset

129,487 records (after dropping records with blank cells)

2.2.4 Number of records in the training and test datasets

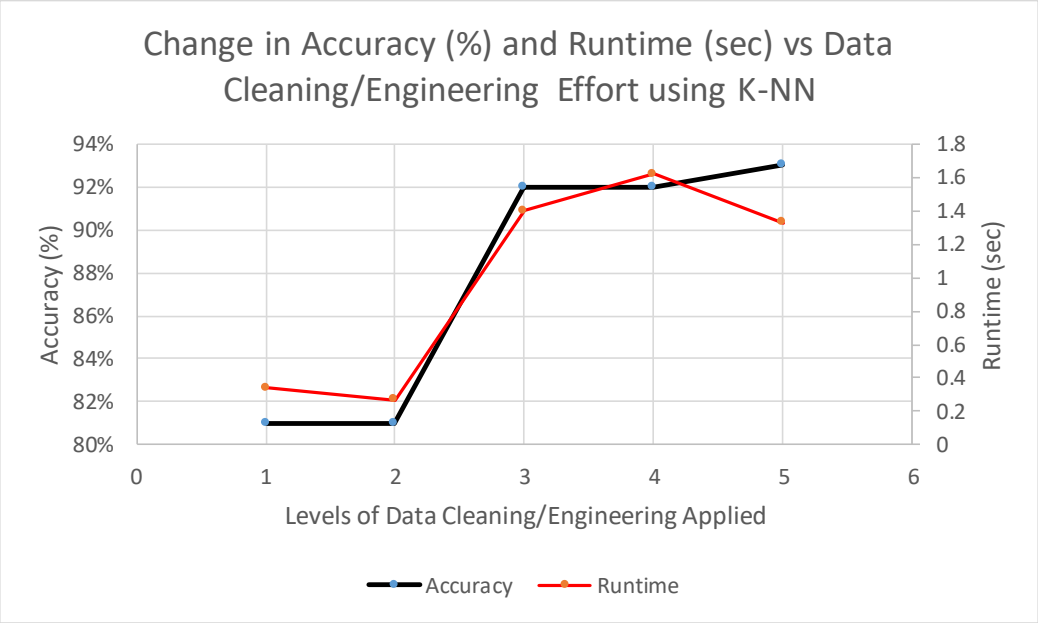
Training dataset: 97,115

Test dataset: 32,372

2.2.5 Training duration

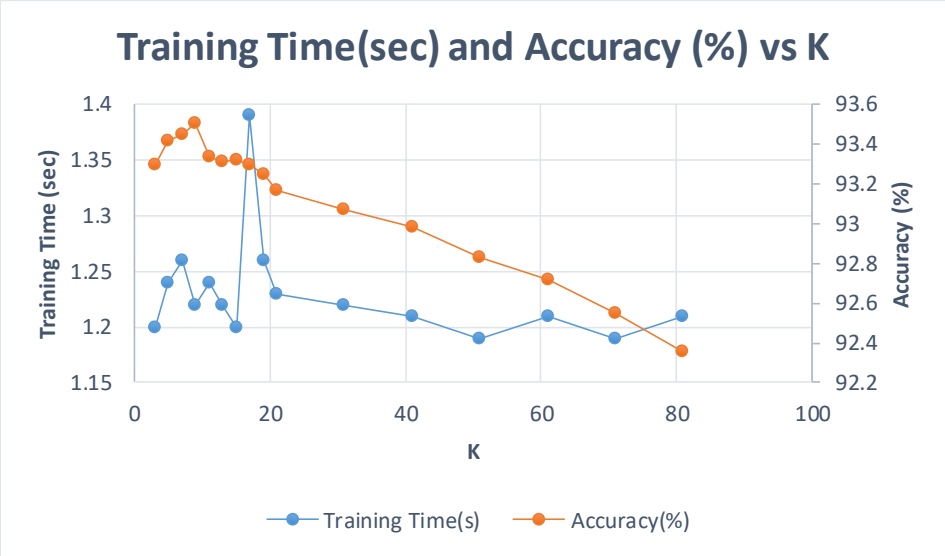
Refer to 2.2.6 below

2.2.6 Improvements after data engineering, feature engineering, model engineering



S/N	Technique Applied
1	None
2	Drop null data
3	Drop null data & binning
4	Drop null data, binning, scaling
5	Drop null data, binning, scaling, feature engineering

2.2.7 Other findings and observations



Increasing number of K does not significantly increase training time. The accuracy will increase and reach maximum when K=9 then decrease constantly.

2.3 Support Vector Machines

2.3.1 Code

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
```

```
# read in data from sample file
df = pd.read_excel('satisfaction_2015.xlsx')
df.head() # show the first five rows
```

```
df.shape
```

```
df.info()
```

```
df.isnull().sum()
```

```
df.describe()
```

```
#categorical variable and binning

df_cat = df
#satisfied=1, dissatisfied=0
df_cat.loc[df['satisfaction_v2'] == 'neutral or dissatisfied',␣
 ↪'satisfaction_v2'] = 0
df_cat.loc[df['satisfaction_v2'] == 'satisfied', 'satisfaction_v2'] = 1
#male=1, female=0
df_cat.loc[df['Gender'] == 'Female', 'Gender'] = 0
df_cat.loc[df['Gender'] == 'Male', 'Gender'] = 1
#loyal=1, disloyal=0
df_cat.loc[df['Customer Type'] == 'disloyal Customer', 'Customer Type'] = 0
df_cat.loc[df['Customer Type'] == 'Loyal Customer', 'Customer Type'] = 1
#Personal Travel=0, Business Travel=1
df_cat.loc[df['Type of Travel'] == 'Personal Travel', 'Type of Travel'] = 0
df_cat.loc[df['Type of Travel'] == 'Business travel', 'Type of Travel'] = 1
#Eco=0, Eco Plus=1, Business=2
df_cat.loc[df['Class'] == 'Eco', 'Class'] = 0
df_cat.loc[df['Class'] == 'Eco Plus', 'Class'] = 1
df_cat.loc[df['Class'] == 'Business', 'Class'] = 2
df_cat.info()
```

```
df_cat.head()
```

```
df_cat['Arrival Delay in Minutes']=df_cat['Arrival Delay in Minutes'].
 ↪astype(int) #converting the Arrival Delay in Minutes from float type to type␣
 ↪int
```

```
df_cat.head()
```

```
X=df_cat.iloc[:,2:22]
y=df_cat['satisfaction_v2']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
from sklearn.svm import SVC

svClassifier = SVC(gamma='auto')
svClassifier.fit(X_train, y_train)
```

Time Taken:

```
y_pred = svClassifier.predict(X_test)
y_pred.shape # predicted result
```

SVM Result before binning,scaling and feature engineering

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

1 Drop Null values

```
#drop null-value entries
df=df.dropna()
df.info()
```

```
X=df_cat.iloc[:,2:23]
y=df_cat['satisfaction_v2']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

```
from sklearn.svm import SVC

svClassifier = SVC(kernel='rbf',gamma="auto")
svClassifier.fit(X_train, y_train)
```

```
y_pred = svClassifier.predict(X_test)
y_pred.shape # predicted result
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

2 Binning

```
#categorize age, by an interval of 10 years
```

```
df_cat.loc[df['Age'] < 10, 'Age'] = 0
df_cat.loc[(df['Age'] >= 10) & (df['Age'] < 20), 'Age'] = 1
df_cat.loc[(df['Age'] >= 20) & (df['Age'] < 30), 'Age'] = 2
df_cat.loc[(df['Age'] >= 30) & (df['Age'] < 40), 'Age'] = 3
df_cat.loc[(df['Age'] >= 40) & (df['Age'] < 50), 'Age'] = 4
df_cat.loc[(df['Age'] >= 50) & (df['Age'] < 60), 'Age'] = 5
df_cat.loc[(df['Age'] >= 60) & (df['Age'] < 70), 'Age'] = 6
df_cat.loc[(df['Age'] >= 70) & (df['Age'] < 80), 'Age'] = 7
df_cat.loc[(df['Age'] >= 80) & (df['Age'] < 90), 'Age'] = 8
#Categorize flight distance, by an interval of 500
df_cat.loc[df['Flight Distance'] < 500, 'Flight Distance'] = 0
df_cat.loc[(df['Flight Distance'] >= 500) & (df['Flight Distance'] < 1000),␣
 ↪'Flight Distance'] = 1
df_cat.loc[(df['Flight Distance'] >= 1000) & (df['Flight Distance'] < 1500),␣
 ↪'Flight Distance'] = 2
df_cat.loc[(df['Flight Distance'] >= 1500) & (df['Flight Distance'] < 2000),␣
 ↪'Flight Distance'] = 3
df_cat.loc[df['Flight Distance'] >= 2000, 'Flight Distance'] = 4
#Categorize Departure Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Departure Delay in Minutes'] < 7.5, 'Departure Delay in␣
 ↪Minutes'] = 0
df_cat.loc[(df['Departure Delay in Minutes'] >= 7.5) & (df['Departure Delay in␣
 ↪Minutes'] < 15), 'Departure Delay in Minutes'] = 1
df_cat.loc[(df['Departure Delay in Minutes'] >= 15) & (df['Departure Delay in␣
 ↪Minutes'] < 22.5), 'Departure Delay in Minutes'] = 2
df_cat.loc[(df['Departure Delay in Minutes'] >= 22.5) & (df['Departure Delay in␣
 ↪Minutes'] < 30), 'Departure Delay in Minutes'] = 3
df_cat.loc[df['Departure Delay in Minutes'] >= 30, 'Departure Delay in␣
 ↪Minutes'] = 4
#Categorize Arrival Delay in Minutes, by an interval of 7.5
df_cat.loc[df['Arrival Delay in Minutes'] < 7.5, 'Arrival Delay in Minutes'] = 0
df_cat.loc[(df['Arrival Delay in Minutes'] >= 7.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 15), 'Arrival Delay in Minutes'] = 1
df_cat.loc[(df['Arrival Delay in Minutes'] >= 15) & (df['Arrival Delay in␣
 ↪Minutes'] < 22.5), 'Arrival Delay in Minutes'] = 2
df_cat.loc[(df['Arrival Delay in Minutes'] >= 22.5) & (df['Arrival Delay in␣
 ↪Minutes'] < 30), 'Arrival Delay in Minutes'] = 3
df_cat.loc[df['Arrival Delay in Minutes'] >= 30, 'Arrival Delay in Minutes'] = 4
df_cat.head(48850)
```

```
X=df_cat.iloc[:,2:23]
y=df_cat['satisfaction_v2']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)

from sklearn.svm import SVC

svClassifier = SVC(kernel='rbf',gamma="auto")
svClassifier.fit(X_train, y_train)
y_pred = svClassifier.predict(X_test)
y_pred.shape # predicted result
```

```
# SVM Result before scaling and feature engineering
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

3 Feature Scaling

```
from sklearn.preprocessing import StandardScalar
```

```
features=['Gender','Customer Type','Age','Type of Travel','Class','Flight␣
 ↪Distance','Inflight wifi service','Departure/Arrival time convenient','Ease␣
 ↪of Online booking','Gate location','Food and drink','Online boarding','Seat␣
 ↪comfort',
         'Inflight entertainment','On-board service','Leg room␣
 ↪service        Baggage handling','Checkin service','Inflight␣
 ↪service','Cleanliness','Departure Delay in Minutes','Arrival Delay in␣
 ↪Minutes']
#seperating out the features
x=df_cat.loc[:,features].values
#seperating out the target
y=df_cat.loc[:,[''satisfaction_v2]].values
#Standardizing the features
x=StandardScalar().fit_transform(x)
pd.DataFrame(data=x,columns=features).head()
```

```
#X=df_cat.iloc[:,2:23]
#y=df_cat['satisfaction_v2']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 42)

from sklearn.svm import SVC

svClassifier = SVC(kernel='rbf',gamma="auto")
svClassifier.fit(X_train, y_train)
```

```
y_pred = svClassifier.predict(X_test)
y_pred.shape # predicted result
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

4 FEATURE ENGINEERING

```
plt.matshow(df_cat.corr())
sns.heatmap(corr)
corr.style.background_gradient(cmap='coolwarm')
```

```
#drop unnecessary features
df_cat.drop('Arrival Delay in Minutes', axis=1, inplace=True)
df_cat.drop('Departure/Arrival time convenient', axis=1, inplace=True)
df_cat.drop('Gate location', axis=1, inplace=True)
df_cat.drop('Departure Delay in Minutes', axis=1, inplace=True)
df_cat.head()
```

```
X=df_cat.iloc[:,2:]
y=df_cat['satisfaction_v2']

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x, y, random_state = 42)

from sklearn.svm import SVC

svClassifier = SVC(kernel='rbf',gamma="auto")
svClassifier.fit(X_train, y_train)
```

2.3.2 Data Dictionary

As shown in code above

2.3.3 Number of records in the dataset

129,487 records (after dropping records with blank cells)

2.3.4 Number of records in the training and test datasets

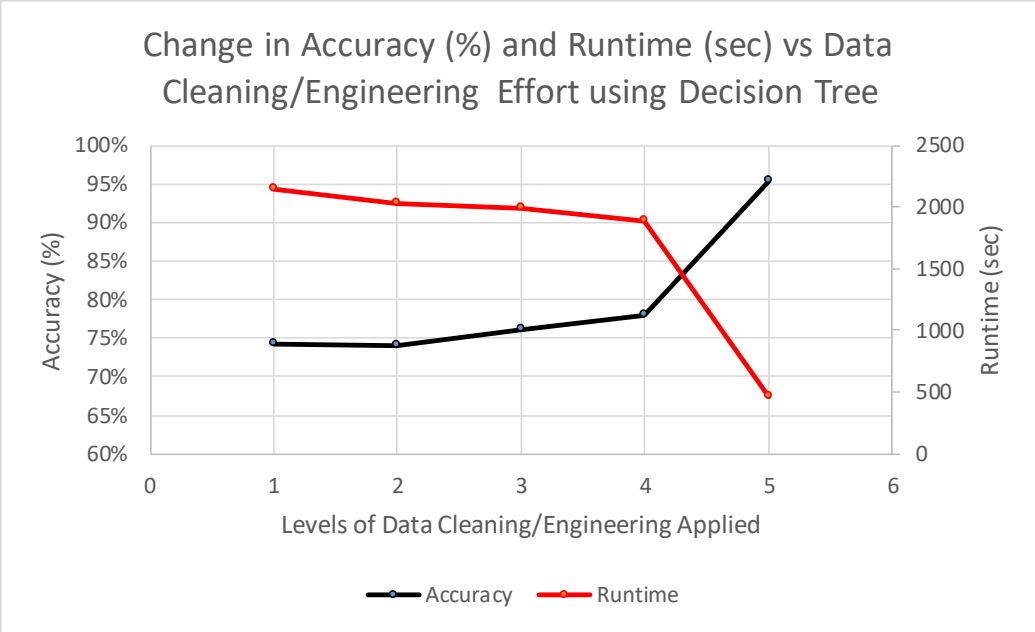
Training dataset: 90,641

Test dataset: 38,846

2.3.5 Training duration

Refer to 2.3.6

2.3.6 Improvements after data engineering, feature engineering, model engineering



2.3.7 Other findings and observations

Attributes	C	Gamma	Kernel	Duration (sec)	Accuracy	Precision	Recall	Remarks
All	1	Auto	rbf	2146	0.73838	0.86497	0.47073	
All	0.5	Auto	rbf	2129	0.6777	0.91684	0.28313	
All	0.1	Auto	rbf	1497	0.56655	0.96	0.001423	
All	1	Auto	linear	NA	NA	NA	NA	Simulation halted after >2 hrs on learning
All	0.5	Auto	linear	NA	NA	NA	NA	Simulation halted after >2 hrs on learning
All	0.1	Auto	linear	NA	NA	NA	NA	Simulation halted after >2 hrs on learning
All	1	Auto	poly	NA	NA	NA	NA	Simulation halted after >2 hrs on learning
All	0.5	Auto	poly	NA	NA	NA	NA	Simulation halted after >2 hrs on learning
All	0.1	Auto	poly	NA	NA	NA	NA	Simulation halted after >2 hrs on learning

Table 2.3.7.1 Variation of C-value and kernels

i. Using all the available dimensions, the maximum achievable accuracy is ~ 73% and this was achievable using only the rbf solver. All other solvers were stopped after over 2hrs of run-time without convergence

ii. The accuracy was noted to decrease with decreasing value of C (between 0 and 1) as can be expected due to having an increasingly smaller margin

iii. SVM appears to require a fairly long time to converge at a result. The duration scales up significantly with the size of the dataset. Even with increased time, convergence is also not guaranteed as evidenced by several of our trials which saw no results in spite of running the computer overnight

iv. Attribute reduction was however noted to give a significant boost in accuracy and reduction in time as evidenced by the

2.4 Decision Trees and / or Random Forest

2.4.1 Code

```
# import the things we need first
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
import datetime

print(datetime.datetime.now().time())
```

```
df = pd.read_excel('satisfaction_2015.xlsx').dropna()
df.head()
```

```
data = df.iloc[:, :]
target = df['satisfaction_v2']
```

```
#change the flight distance and the age
# make a copy of the dataframe
data_copy = data
# categorize by their Age
data_copy.loc[df['Age']< 18, 'Age'] = 0
data_copy.loc[(df['Age'] >= 18) & (df['Age'] < 30), 'Age'] = 1
data_copy.loc[(df['Age'] >= 30) & (df['Age'] < 50), 'Age'] = 2
data_copy.loc[df['Age'] >= 50, 'Age'] = 3
# categorize by their Flight Distance
data_copy.loc[df['Flight Distance']< 1000, 'Flight Distance'] = 0
data_copy.loc[(df['Flight Distance'] >= 1000) & (df['Flight Distance'] < 5000),␣
↪'Flight Distance'] = 1
data_copy.loc[(df['Flight Distance'] >= 5000) & (df['Flight Distance'] <␣
↪10000), 'Flight Distance'] = 2
data_copy.loc[df['Flight Distance'] >= 10000, 'Flight Distance'] = 3
# categorize by their Departure Delay in Minutes
data_copy.loc[df['Departure Delay in Minutes']< 10, 'Departure Delay in␣
↪Minutes'] = 0
```

```
data_copy.loc[(df['Departure Delay in Minutes'] >= 10) & (df['Departure Delay␣
↪in Minutes'] < 30), 'Departure Delay in Minutes'] = 1
data_copy.loc[df['Departure Delay in Minutes'] >= 30, 'Departure Delay in␣
↪Minutes'] = 2
# categorize by their Departure Delay in Minutes
data_copy.loc[df['Arrival Delay in Minutes']< 10, 'Arrival Delay in Minutes'] =␣
↪0
data_copy.loc[(df['Arrival Delay in Minutes'] >= 10) & (df['Arrival Delay in␣
↪Minutes'] < 30), 'Arrival Delay in Minutes'] = 1
data_copy.loc[df['Arrival Delay in Minutes'] >= 30, 'Arrival Delay in Minutes']␣
↪= 2
```

```
# to customize what value for each feature to be mapped to
# we can provide a dictionary that has all the mapping rules


mydict = {
    "satisfied": 1,
    "neutral or dissatisfied": 0,
    "Male": 0,
    "Female": 1,
    "Loyal Customer": 0,
    "disloyal Customer": 1,
    "Business travel":0,
    "Personal Travel":1,
    "Eco":0,
    "Eco Plus":1,
    "Business":2
}

#we should only map certain rows
for i in [1,2,3,5,6]:
    data_copy.iloc[:, i] = data_copy.iloc[:, i].map(mydict)
```

```
from sklearn.model_selection import train_test_split

# split our data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(data_copy.iloc[:,2:24],␣
 ↪data['satisfaction_v2'], random_state = 42)
target = data_copy['satisfaction_v2']
```

```
# import decision tree model from sklearn
from sklearn.tree import DecisionTreeClassifier

# instantiate a decision tree model. All parameters can be omited to use␣
 ↪default ones.
# details please check https://scikit-learn.org/stable/modules/generated/
 ↪sklearn.tree.DecisionTreeClassifier.html
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train) # train our model
y_pred = dt.predict(X_test) # let the model predict the test data
```

```
X_test
```

```
y_test
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
print(y_pred) # what the model predict as satisfaction labels
print(y_test) # true labels
```

```
# show the decision tree model
# import graphviz and sklearn.tree first
from sklearn import tree
import graphviz
from graphviz import Source

Source(tree.export_graphviz(dt, class_names=True, out_file=None, feature_names=␣
 ↪X_train.columns)) # display the tree, with no output file
```

```
print(datetime.datetime.now().time())
```

2.4.2 Data Dictionary

Refer to code above

2.4.3 Number of records in the dataset

129,487 records (after dropping records with blank cells)

2.4.4 Number of records in the training and test datasets

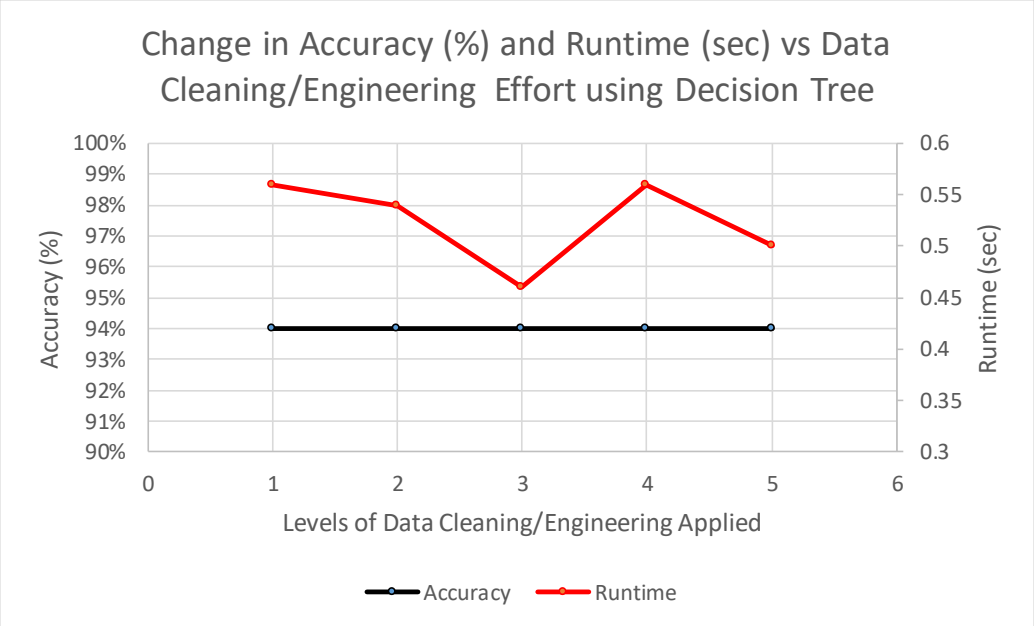
Training dataset: 97,115

Test dataset: 32,372

2.4.5 Training duration

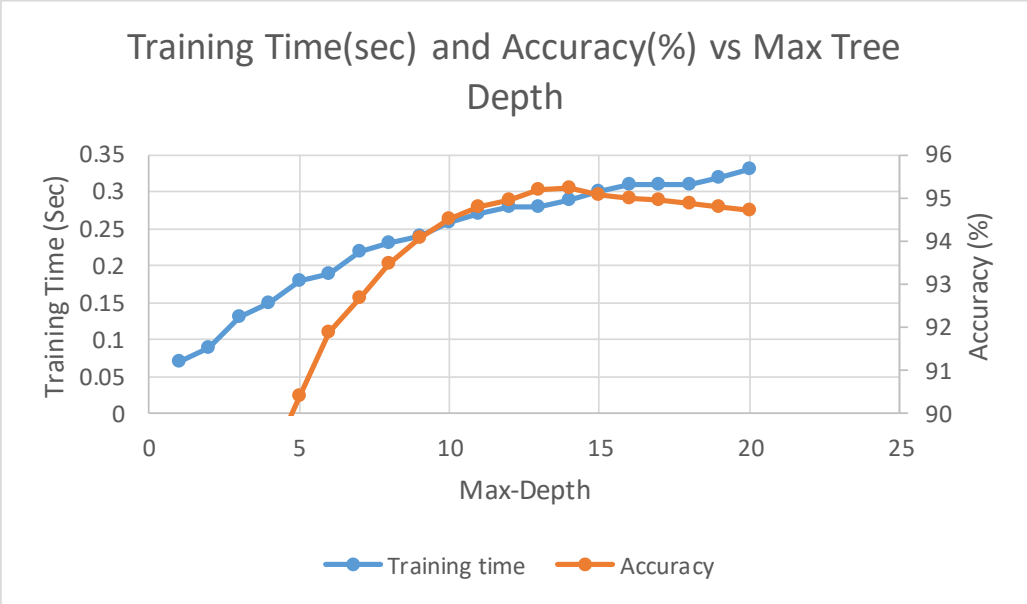
Refer to 2.4.6 below

2.4.6 Improvements after data engineering, feature engineering, model engineering



S/N	Technique Applied
1	None
2	Drop null data
3	Drop null data & binning
4	Drop null data, binning, scaling
5	Drop null data, binning, scaling, feature engineering

2.4.7 Other findings and observations



When max-depth increases, training time will also increase because more computational power is required. However, accuracy will reach maximum when max-depth=14 and it will decrease thereafter. This is due to overfitting.

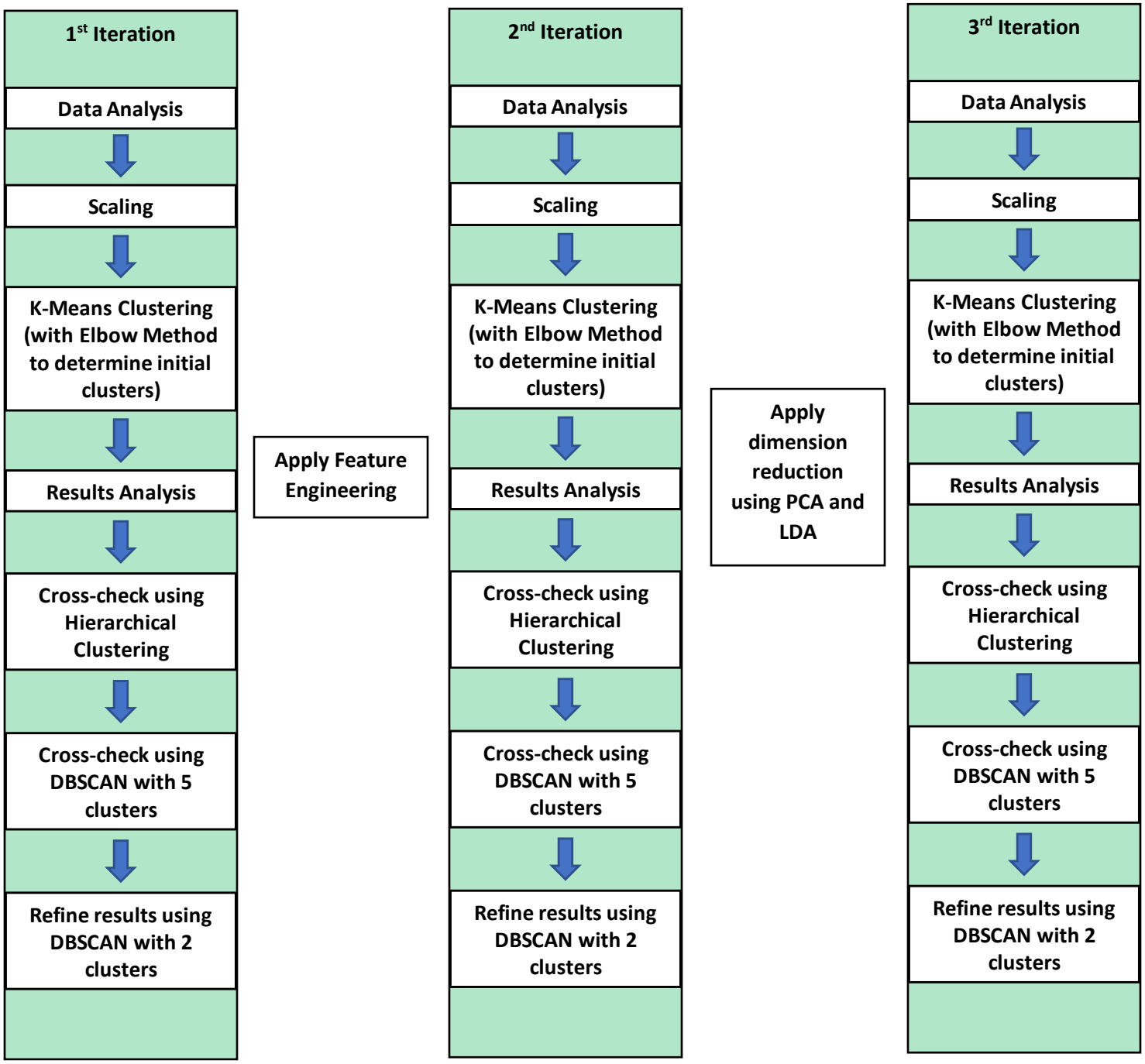
3. Unsupervised Machine Learning

3.1 Dataset characteristics

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
count	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000	17898.000000
mean	111.079968	46.549532	0.477857	1.770279	12.614400	26.326515	8.303556	104.857709	0.091574
std	25.652935	6.843189	1.064040	6.167913	29.472897	19.470572	4.506092	106.514540	0.288432
min	5.812500	24.772042	-1.876011	-1.791886	0.213211	7.370432	-3.139270	-1.976976	0.000000
25%	100.929688	42.376018	0.027098	-0.188572	1.923077	14.437332	5.781506	34.960504	0.000000
50%	115.078125	46.947479	0.223240	0.198710	2.801839	18.461316	8.433515	83.064556	0.000000
75%	127.085938	51.023202	0.473325	0.927783	5.464256	28.428104	10.702959	139.309331	0.000000
max	192.617188	98.778911	8.069522	68.101622	223.392140	110.642211	34.539844	1191.000837	1.000000

- <class 'pandas.core.frame.DataFrame'>
- RangeIndex: 17898 entries, 0 to 17897
- Data columns (total 9 columns):
- Mean of the integrated profile 17898 non-null float64
- Standard deviation of the integrated profile 17898 non-null float64
- Excess kurtosis of the integrated profile 17898 non-null float64
- Skewness of the integrated profile 17898 non-null float64
- Mean of the DM-SNR curve 17898 non-null float64
- Standard deviation of the DM-SNR curve 17898 non-null float64
- Excess kurtosis of the DM-SNR curve 17898 non-null float64
- Skewness of the DM-SNR curve 17898 non-null float64
- target_class 17898 non-null int64
- dtypes: float64(8), int64(1)
- memory usage: 1.2 MB

3.2 Workflow Applied for Unsupervised Learning



Data was also analyzed using the following modelling methods

1) KNN,
2) Decision Tree,
3) Random Forest,
4) Naïve Bayes
5) SVM

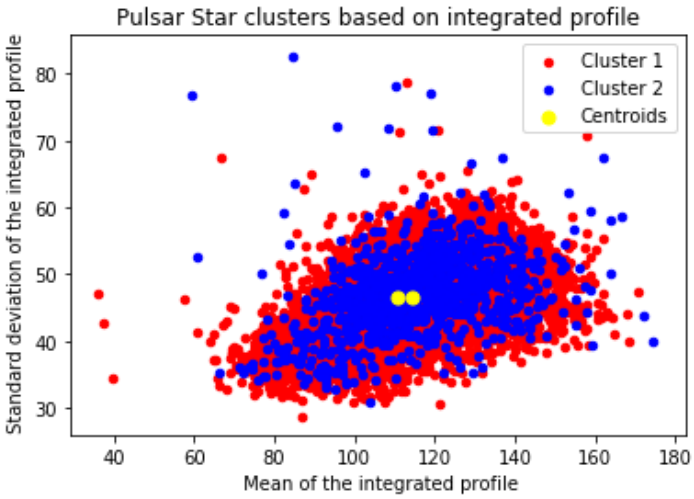
3.3 Code for 1st Iteration

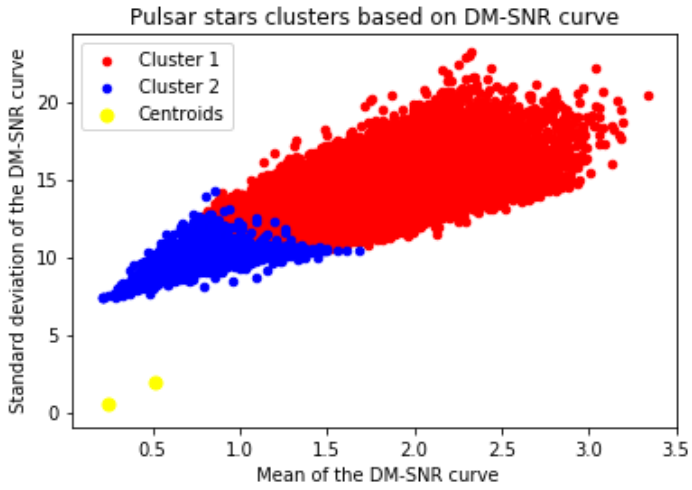
3.3.1 Code for K-means Clustering (with 2 clusters)

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
```

```
dataset = pd.read_csv('pulsar_stars.csv')

ds1_8 = dataset.iloc[:, [0,1,2,3,4,5,6,7]]
print (ds1_8.head())
```

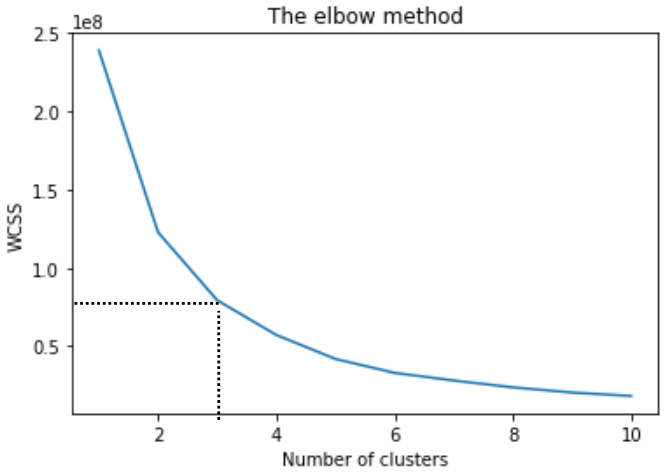
```
#Scaling is important. All distance based methods should scale
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x = scaler.fit_transform(ds1_8)
km2 = KMeans(n_clusters = 2, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)

km2.fit(x)
y_kmeans = km2.fit_predict(x)
#Visualising the clusters by selecting only column 0 and 1
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 20, c = 'red', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 20, c = 'blue', label = 'Cluster 2')

#Plotting the centroids of the clusters
plt.scatter(km2.cluster_centers_[:, 0], km2.cluster_centers_[:,1], s = 50, c = 'yellow', label = 'Centroids')
print (dataset.columns)
plt.title('Pulsar Star clusters based on integrated profile')
plt.xlabel(dataset.columns[0])
plt.ylabel(dataset.columns[1])

plt.legend()
plt.show()
```



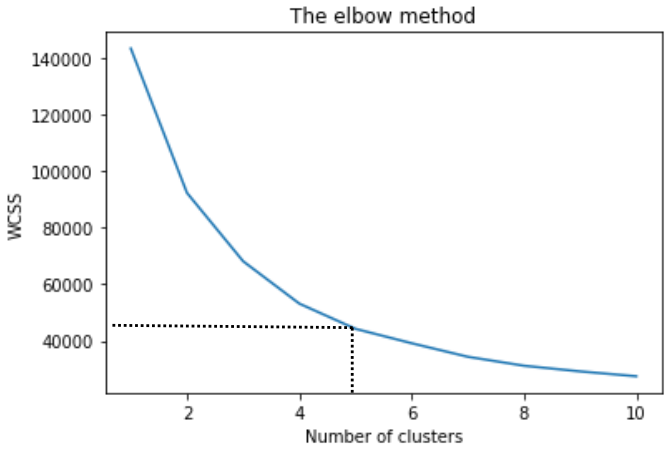


3.3.2 Code for Elbow Method

```
#Finding the optimum number of clusters for k-means classification
from sklearn.cluster import KMeans
wcss = []

# Trying kmeans for k=1 to k=10
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```



Before scaling

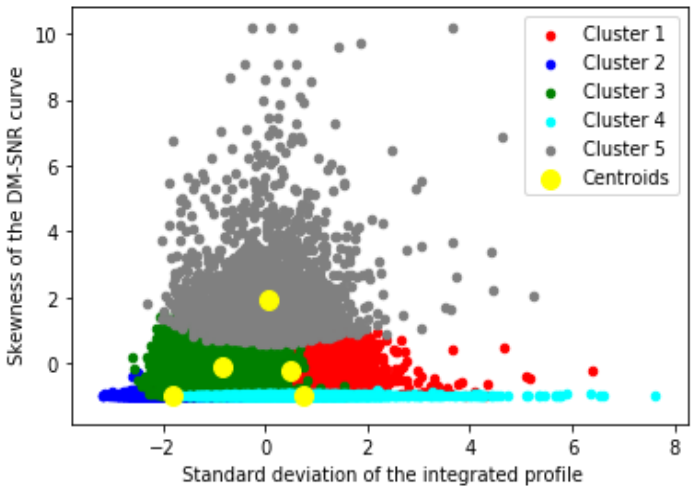


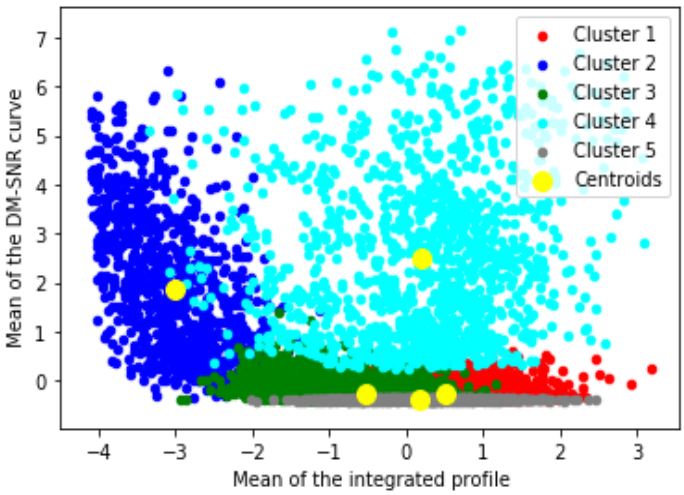
After scaling

As K-Means is a distance-based dataset, we will need to carry out scaling to ensure an accurate prediction of the clusters. From the above figure, we can see that scaling does affect the justification of the optimal WCSS. As observed above, without scaling the suggested number of clusters is ~ 3, whereas after scaling, the suggested number of clusters is ~ 5.

3.3.3 Code for K-means with 5 clusters (as derived from Elbow Method)

```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x)

i = 1
j = 7
#Visualising the clusters
plt.scatter(x[y_kmeans == 0, i], x[y_kmeans == 0, j], s = 20, c = 'red', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, i], x[y_kmeans == 1, j], s = 20, c = 'blue', label = 'Cluster 2')
plt.scatter(x[y_kmeans == 2, i], x[y_kmeans == 2, j], s = 20, c = 'green', label = 'Cluster 3')
plt.scatter(x[y_kmeans == 3, i], x[y_kmeans == 3, j], s = 20, c = 'cyan', label = 'Cluster 4')
plt.scatter(x[y_kmeans == 4, i], x[y_kmeans == 4, j], s = 20, c = 'grey', label = 'Cluster 5')

plt.xlabel(dataset.columns[i])
plt.ylabel(dataset.columns[j])

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, i], kmeans.cluster_centers_[:,j], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```





3.3.4 Code for Hierarchical Clustering (using 5 clusters as derived from Elbow Method)

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
```

```
dataset = pd.read_csv('pulsar_stars.csv')

ds0_7 = dataset.iloc[:, [0,1,2,3,4,5,6,7]]
print (ds0_7.head())
```

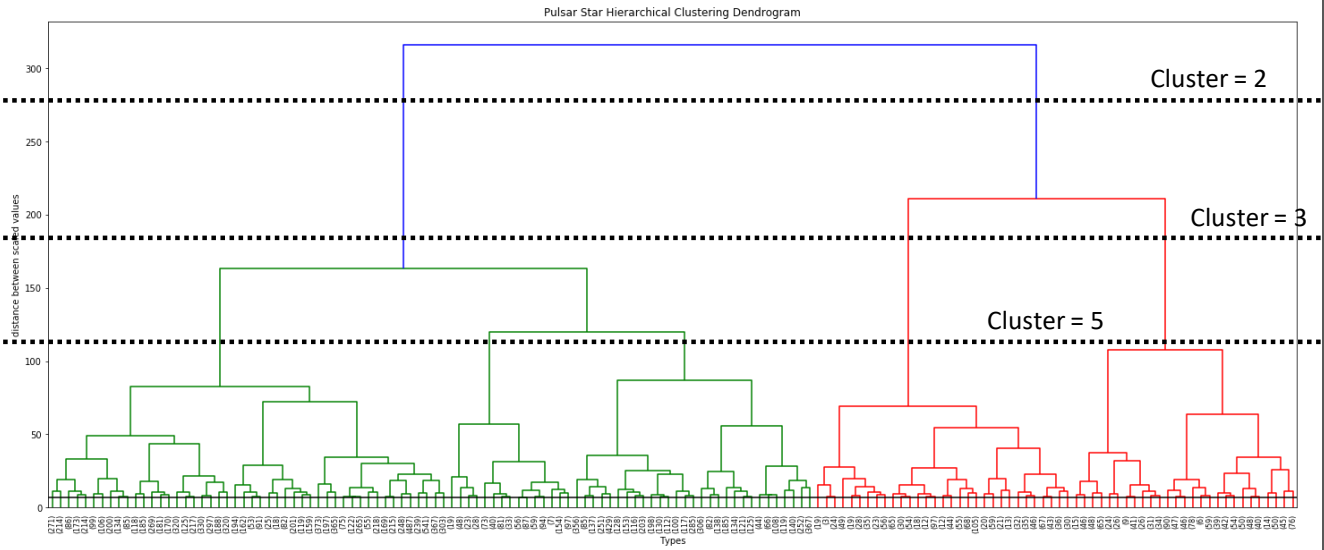
```
#Scaling is important. All distance based methods should scale
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x =preprocessing.MinMaxScaler().fit_transform(ds0_7)
#x = scaler.fit_transform(ds0_7)
```

```
from scipy.cluster.hierarchy import dendrogram, linkage

# generate the linkage matrix
Z = linkage(x, 'ward')

# set cut-off to 150 cluster merges
max_d = 7.08                # max_d as in max_distance

plt.figure(figsize=(25, 10))
plt.title('Pulsar Star Hierarchical Clustering Dendrogram')
plt.xlabel('Types')
plt.ylabel('distance between scaled values')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=150,                  # Try changing values of p
    leaf_rotation=90.,      # rotates the x axis labels
    leaf_font_size=8.,      # font size for the x axis labels
)
plt.axhline(y=max_d, c='k')
plt.show()
```

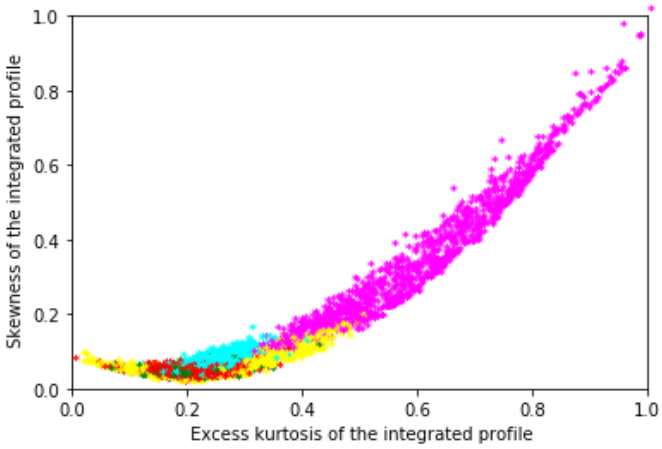
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(linkage="ward", n_clusters=5)
clustering.fit(x);
print(clustering)
```

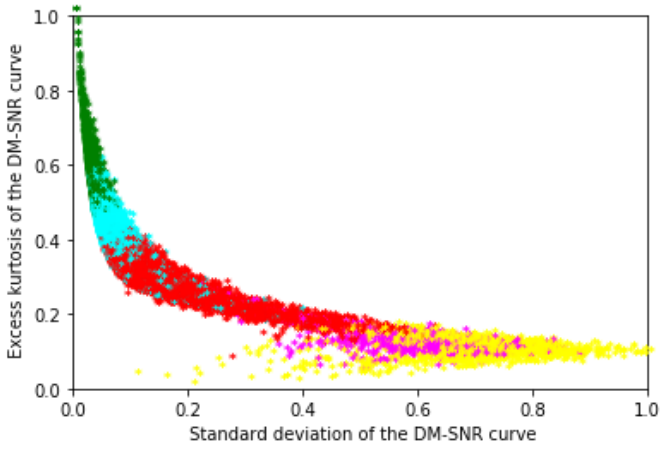
```
from sklearn import preprocessing

# label_encoder object knows how to understand word labels.
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'target_class'.
y = label_encoder.fit_transform(dataset['target_class'])

# MinMax scale the data so that it fits nicely onto the 0.0->1.0 axes of the plot.
from sklearn import preprocessing
X_plot = preprocessing.MinMaxScaler().fit_transform(x)
```

```
colours = ['magenta','cyan','yellow','red','green','blue','black']
for i in range(x.shape[0]):
    plt.text(X_plot[i, 5], X_plot[i, 6], str("*"),
             color=colours[clustering.labels_[i]],
             fontdict={'weight': 'bold', 'size': 9}
        )
plt.xlabel(dataset.columns[5])
plt.ylabel(dataset.columns[6])
#plt.xticks([])
#plt.yticks([])
#plt.axis('off')
plt.show()
```



Cluster 5 is decided to be the most optimum based on the elbow method.





3.3.5 Code for DBSCAN

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
# importing the pulsar star dataset with pandas
dataset = pd.read_csv('pulsar_stars.csv')
X = dataset.drop(['target_class'], axis=1)

X.head()
```

```
y = dataset['target_class']
```

```
X = np.array(X)
```

```
# transform the data to be stretched based on the random state - change the random state to change the shape
y = np.array(y)
y
```

```
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# cluster the data
dbscan = DBSCAN(eps=1, min_samples = 5)#increase min sample
clusters = dbscan.fit_predict(X_scaled)

# plot the cluster assignments
plt.scatter(X[:, 0], X[:, 7], c=clusters,)
plt.xlabel("Feature 1")
plt.ylabel("Feature 8")

plt.show()
```

```
# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(clusters)) - (1 if -1 in clusters else 0)
n_noise_ = list(clusters).count(-1)

print(f'Number of clusters = {n_clusters_}')
print(f'Number of noise sample = {n_noise_}')
```

```
from sklearn import metrics
print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(y, clusters))
print("Completeness: %0.3f" % metrics.completeness_score(y, clusters))
print("V-measure: %0.3f" % metrics.v_measure_score(y, clusters))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(y, clusters))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(y, clusters))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, clusters))
```

DBScan (6 clusters)

Estimated number of clusters: 6

Estimated number of noise points: 17587

Homogeneity: 0.006

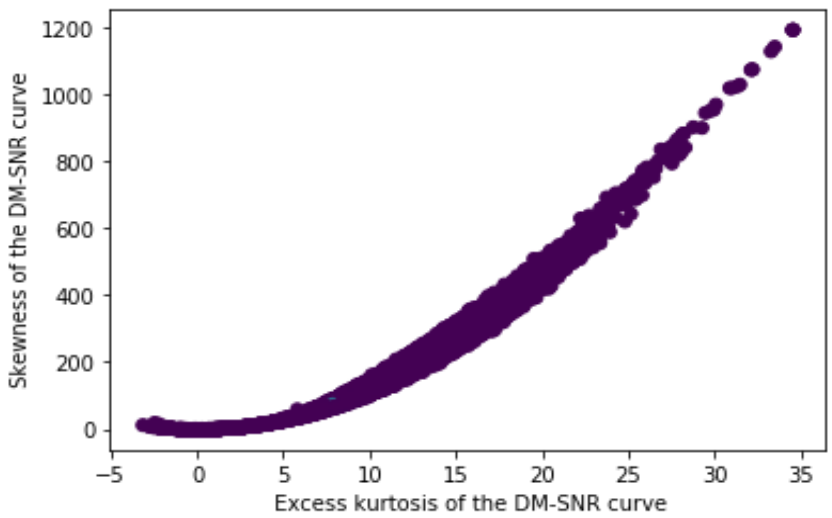
Completeness: 0.015

V-measure: 0.008

Adjusted Rand Index: -0.027

Adjusted Mutual Information: 0.005

Silhouette Coefficient: -0.544



eps=0.2, min_samples = 25

DBScan (2 clusters most optimum)

Estimated number of clusters: 2

Estimated number of noise points: 169

Homogeneity: 0.007

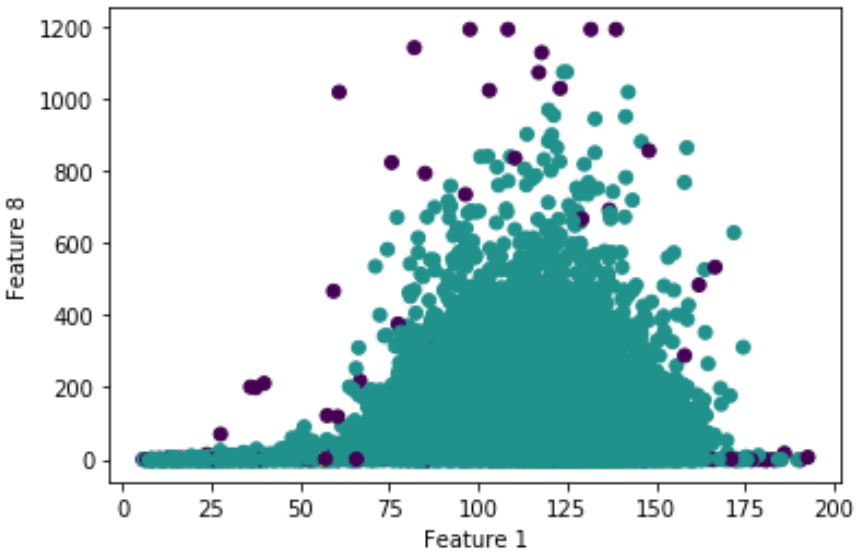
Completeness: 0.040

V-measure: 0.012

Adjusted Rand Index: 0.041

Adjusted Mutual Information: 0.007

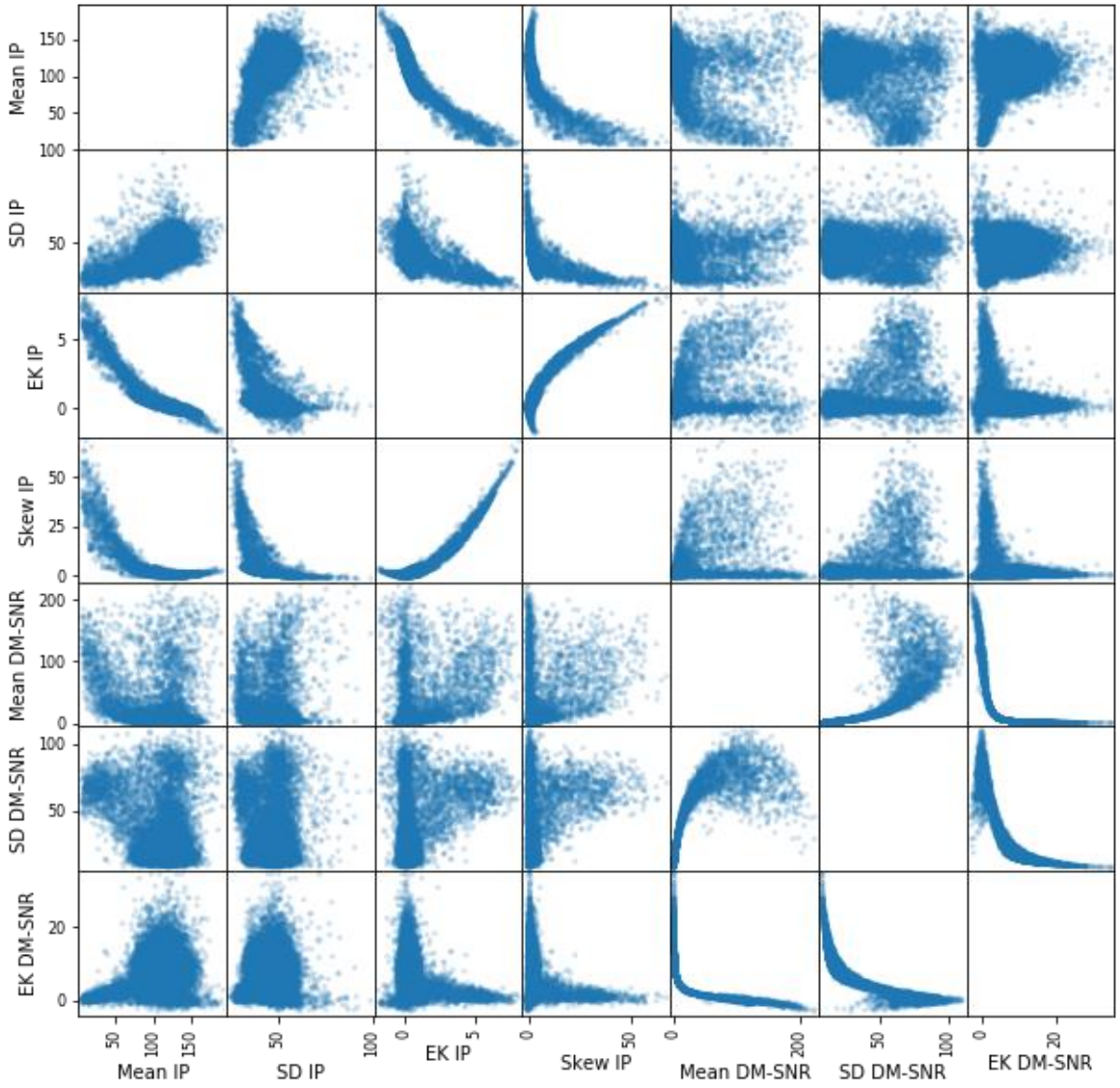
Silhouette Coefficient: 0.414

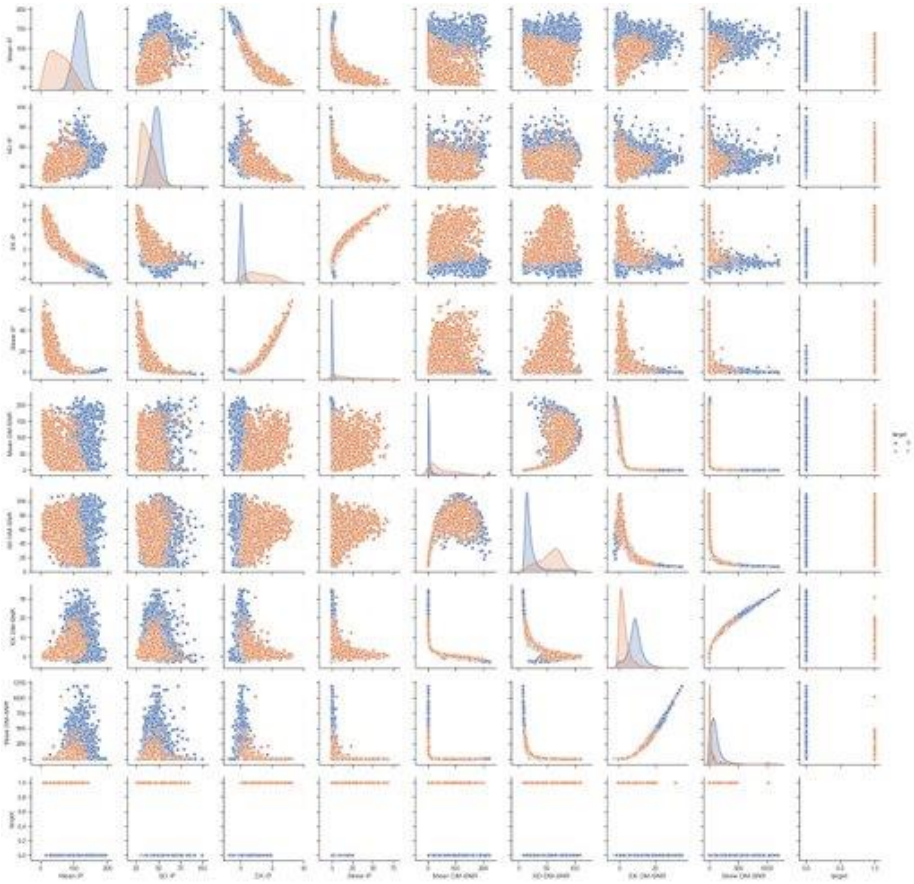


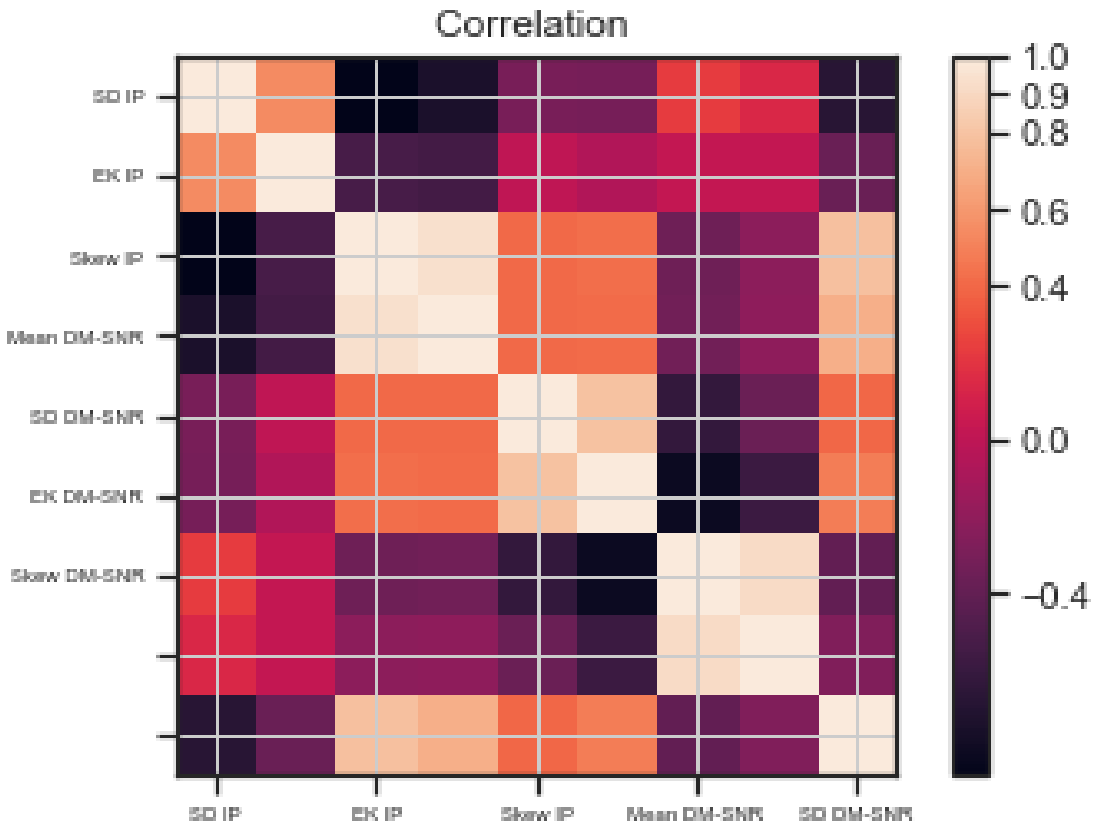
eps=1, min_samples = 5

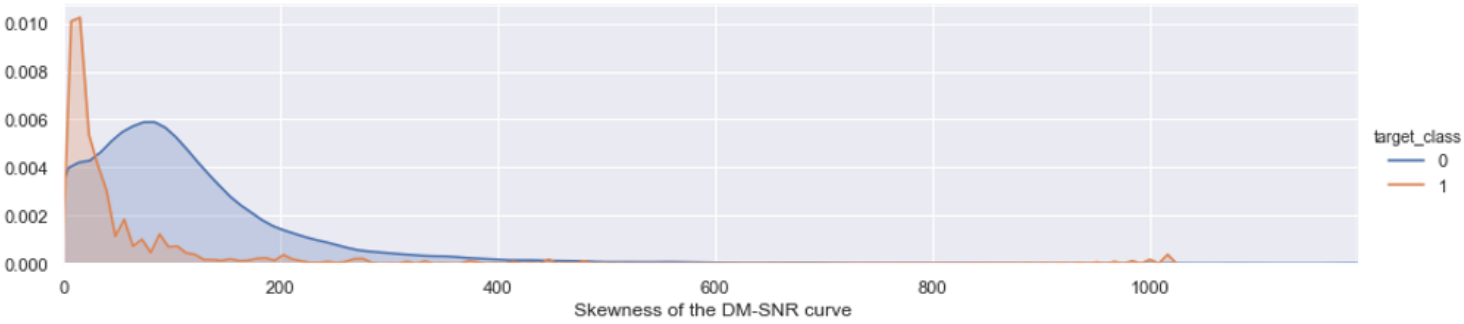
3.4 Feature Engineering

The following workflow was applied as part of feature engineering. From the findings, it was determined that the correlation matrix was the most helpful in visualizing the relationship between the different attributes. **Since all the attributes were found to be important towards the model, no features were dropped.** The findings of the feature engineering are shown below.






Correlation
SD IP
EK IP
Skew IP
Mean DM-SNR
SD DM-SNR
EK DM-SNR
Skew DM-SNR
SD IP
EK IP
Skew IP
Mean DM-SNR
SD DM-SNR
1.0
0.9
0.8
0.6
0.4
0.0
−0.4



	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
Mean of the integrated profile	1	0.547137	-0.873898	-0.738775	-0.298841	-0.307016	0.234331	0.144033	-0.673181
Standard deviation of the integrated profile	0.547137	1	-0.521435	-0.539793	0.00686873	-0.0476316	0.0294294	0.0276915	-0.363708
Excess kurtosis of the integrated profile	-0.873898	-0.521435	1	0.945729	0.414368	0.43288	-0.341209	-0.214491	0.791591
Skewness of the integrated profile	-0.738775	-0.539793	0.945729	1	0.412056	0.41514	-0.328843	-0.204782	0.709528
Mean of the DM-SNR curve	-0.298841	0.00686873	0.414368	0.412056	1	0.796555	-0.615971	-0.354269	0.400876
Standard deviation of the DM-SNR curve	-0.307016	-0.0476316	0.43288	0.41514	0.796555	1	-0.809786	-0.5758	0.491535
Excess kurtosis of the DM-SNR curve	0.234331	0.0294294	-0.341209	-0.328843	-0.615971	-0.809786	1	0.923743	-0.390816
Skewness of the DM-SNR curve	0.144033	0.0276915	-0.214491	-0.204782	-0.354269	-0.5758	0.923743	1	-0.259117
target_class	-0.673181	-0.363708	0.791591	0.709528	0.400876	0.491535	-0.390816	-0.259117	1

3.5 Dimension Reduction Using PCA and LDA

3.5.1 Principal Component Analysis (PCA)

3.5.1.1 Code

```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
```

```
df = pd.read_csv('pulsar_stars.csv')
print(df.columns)
df.describe()
```

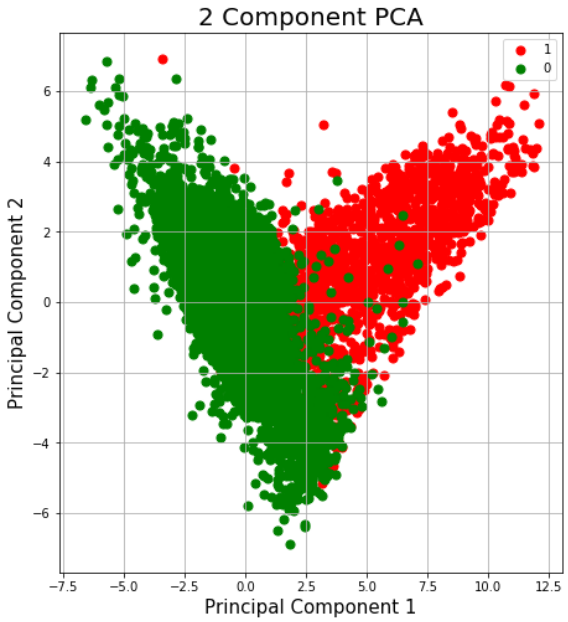
```
from sklearn.preprocessing import StandardScaler
features = [' Mean of the integrated profile', ' Standard deviation of the integrated profile', ' Excess kurtosis of the inte
           ,' Mean of the DM-SNR curve',' Standard deviation of the DM-SNR curve',' Excess kurtosis of the DM-SNR curve',' Sk

# Separating out the features
x = df.loc[:, features].values
# Separating out the target
y = df.loc[:,['target_class']].values
# Standardizing the features
x = StandardScaler().fit_transform(x)

pd.DataFrame(data=x, columns = features).head()
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head()
```

```
finalDf = pd.concat([principalDf, df[['target_class']]], axis = 1)
finalDf.head(5)
```

```
fig = plt.figure(figsize = (16,8))
ax1 = fig.add_subplot(1,2,1)
ax1.set_xlabel('Principal Component 1', fontsize = 15)
ax1.set_ylabel('Principal Component 2', fontsize = 15)
ax1.set_title('2 Component PCA', fontsize = 20)


targets = [1, 0]
colors = 'rgb'
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target_class'] == target
    ax1.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax1.legend(targets)
ax1.grid()
```

	principal component 1	principal component 2
0	-1.278849	-1.273133
1	-1.020553	-0.201162
2	0.188289	0.432114
3	-1.015466	-1.469881
4	-0.822626	2.123651



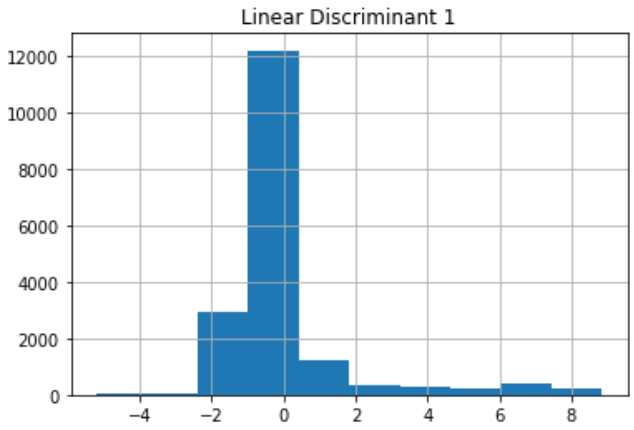
3.5.2 Linear Discriminant Analysis (LDA)

LDA was also performed in comparison to PCA.

3.5.2.1 Code

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=2)
lda_result = lda.fit_transform(x, np.ravel(y))

ldaDf = pd.DataFrame(data = lda_result
             , columns = ['Linear Discriminant 1'])
```

```
ldaDf.hist()
```



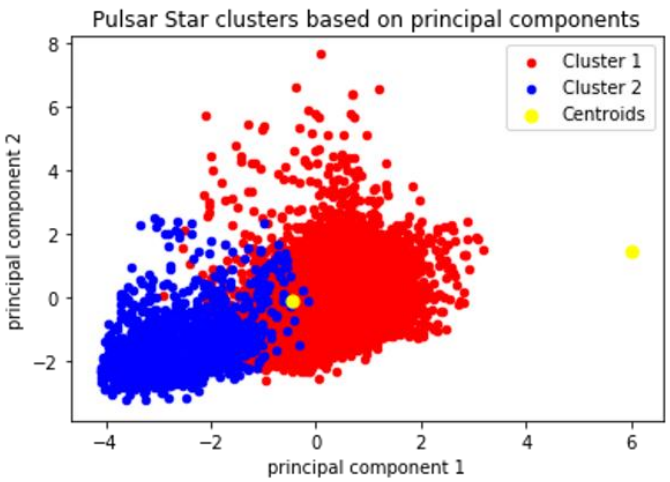
3.5.3 Resulting change in K-means and Hierarchical clustering after PCA

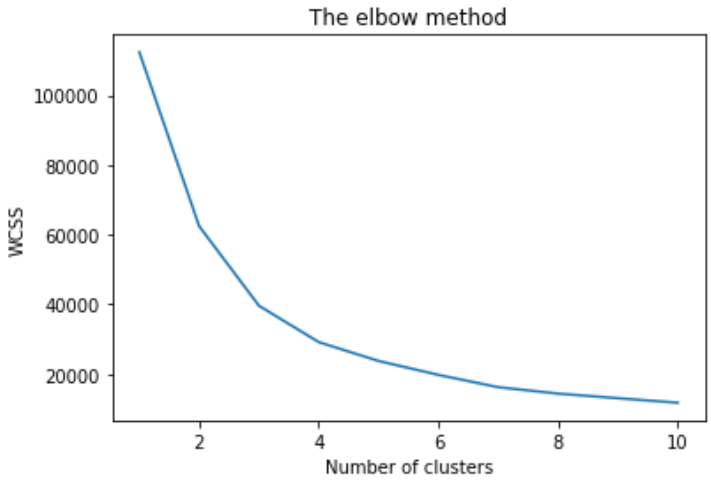
3.5.3.1 Code

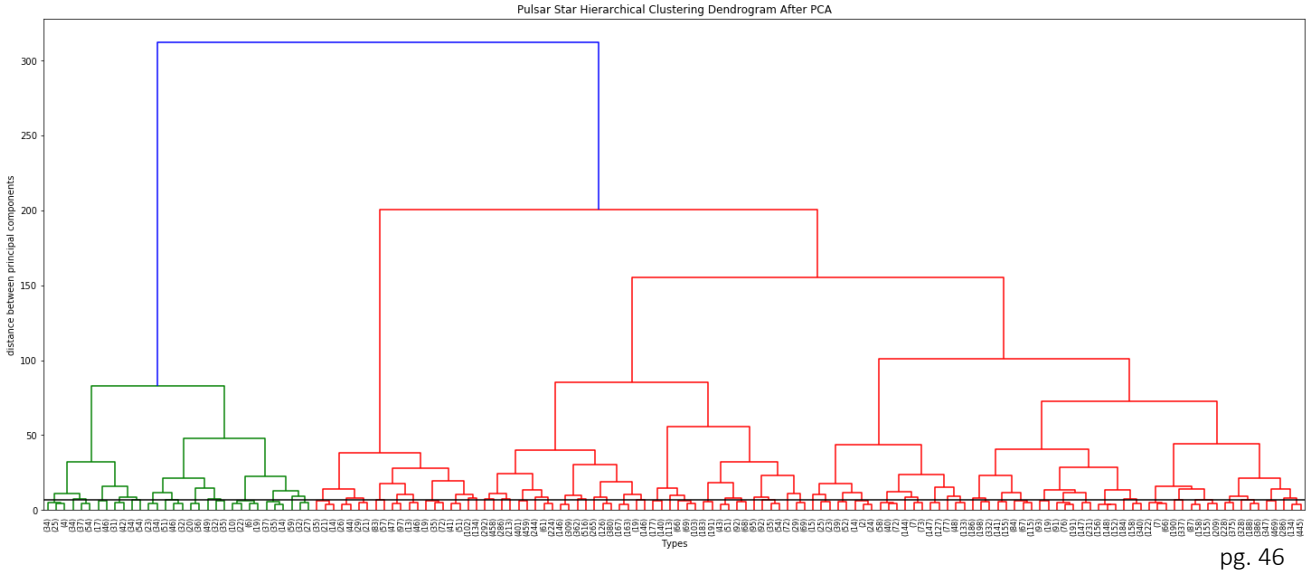
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head()
```

```
km2 = KMeans(n_clusters = 2, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)

km2.fit(principalDf)
y_kmeans = km2.fit_predict(principalDf)
```

```
#Visualising the clusters by selecting only column 0 and 1
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 20, c = 'red', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 20, c = 'blue', label = 'Cluster 2')

#Plotting the centroids of the clusters
plt.scatter(km2.cluster_centers_[:, 0], km2.cluster_centers_[:,1], s = 50, c = 'yellow', label = 'Centroids')
print (principalDf.columns)
plt.title('Pulsar Star clusters based on principal components')
plt.xlabel(principalDf.columns[0])
plt.ylabel(principalDf.columns[1])

plt.legend()
plt.show()
```







3.6 Findings Using Alternative Machine Learning Methods

Method	Accuracy
kNN	97.25
Decision Tree	96.86
Random Forest	97.99
Naive Bayes	94.4
SVM	97.23

3.7 Summary of findings and observations

a) Other clustering models (e.g. Elbow Method, Hierarchical Clustering) rely on own intuition to decide on the optimum number of clusters.

b) DBScan is the most effective in deciding the number of clusters. The ***silhouette coefficient*** (of typical value between -1 and 1) indicates to us if the clusters are optimum (i.e. a negative silhouette coefficient indicates that a sample has been assigned to the wrong cluster). As can be observed in the results from the Elbow Method, the silhouette coefficient of -0.544 suggests that a sample has been assigned to the wrong cluster. Ideally, we should have a positive silhouette coefficient that is as close to unity as possible. The DBScan shows it is clearly more accurate when compared with the Elbow Method and gives an optimum number of clusters of 2 (instead of 6 as determined by the Elbow Method)

c) LDA is not very relevant for the dataset because there were only 2 target classes. PCA is more efficient for this dataset because the samples per class is less. LDA would be more effective if separability is of a higher concern and there are multiple classes.

d) Together, the first two principal components contain **78.48%** of the information. The first principal component contains **51.67%** of the variance and the second principal component contains **0.26%** of the variance.

e) K-Means was also tested out after PCA analysis. After using PCA, we do not need to decide on the 8-dimensionality view for visualisation. K-Means after PCA proves to be accurate and fast in deciding the categories.

4. Comparisons

4.1 Supervised learning

4.1.1 Impact of data cleaning

a) When the number of null data entries are negligible compared to the population, removing null data does not affect the accuracy result significantly.

b) Scaling will not affect the accuracy result for unsupervised learning.

c) Binning is an effective way of improving accuracy.

- The accuracy of equal width binning is slightly better in K-NN, whereas the accuracy of equal frequency binning is better in other models. In general, the difference in accuracy is not significant between two binning methods.

- We can simply use equal width binning when is distribution of the feature is not skewed. The advantage of equal width binning is that it is more straightforward.

- If the distribution of the feature is significantly skewed or if there are outliers from the cluster, we must use equal frequency binning instead.

- The findings pertaining to binning are below (For details of the binning, please refer to the appendix)

Equal Frequency Binning	
Model	**Accuracy**
kNN	92.95
Decision Tree	94.21
Random Forest	95.78
Naïve Bayes	86.98
SVM	76.15
LogReg	87.06

Table 4.1.1.1

Equal Width Binning	
Model	**Accuracy**
kNN	93.3
Decision Tree	94.15
Random Forest	95.66
Naïve Bayes	86.88
SVM	78.68
LogReg	85.66

Table 4.1.1.2

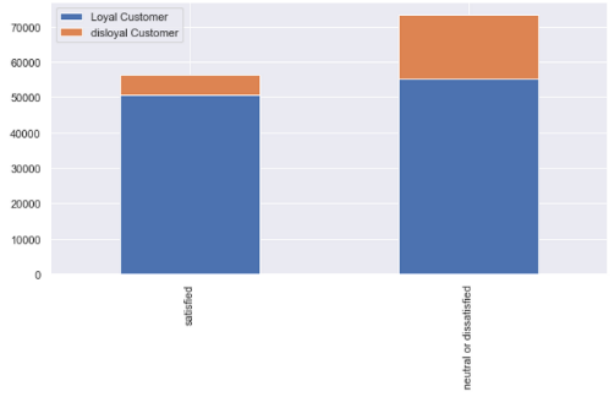
d) Using reducing dimensions itself will increase accuracy. However, if this is used together with other techniques, accuracy results will remain almost the same. Selected features were dropped based on the correlation matrix. (The full matrix may be found in the appendix) Highlighted here are some of the pertinent findings relating to the choice of dropped features which are:

- **Gender : -0.011**
- **Departure/Arrival time convenient : -0.054**
- **Gate location : -0.0029**
- **Departure delay in minutes : -0.071**
- **Arrival delay in minutes : (as departure delay in minutes)**

e) Some other features were also selectively assessed using different visual aids to determine if they should be included or dropped as part of the feature engineering

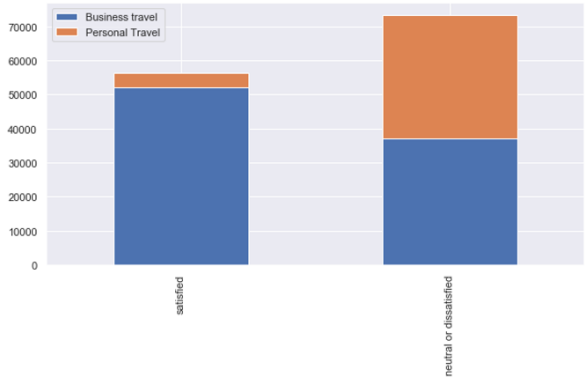
Customer Loyalty

- The Chart confirms **loyal customers** more likely to be satisfied than **disloyal customers**.
- Hence, we will be keeping the customer loyalty feature.



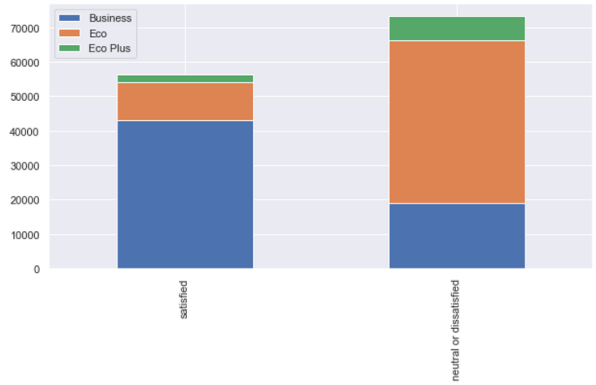
Type of Travel

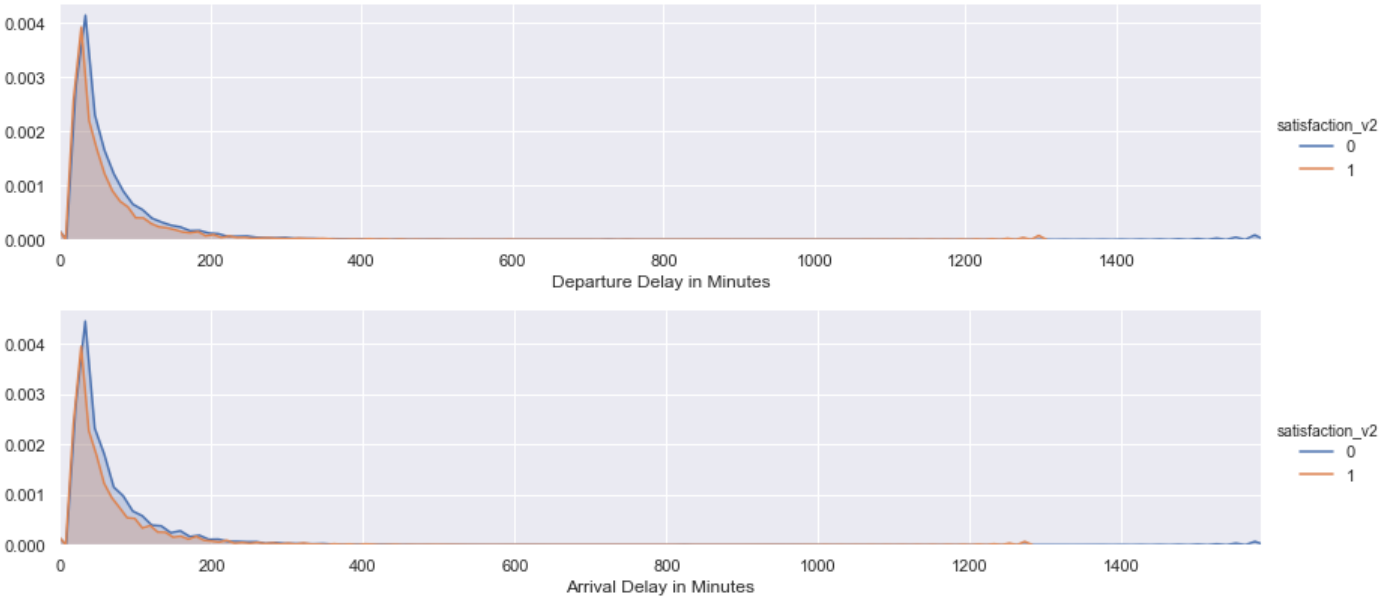
- The Chart confirms **a person aboard in Business Travel** more likely to be satisfied.
- Hence we will be keeping the Type of Travel attribute.



Class

- The Chart confirms **a person in Business class** slightly more likely to be satisfied.
- Hence we will be keeping the Travel Class attribute.





These two charts show that the Departure delay in minutes and the Arrival delay in minutes are approximately the same.

4.1.2 Between different models

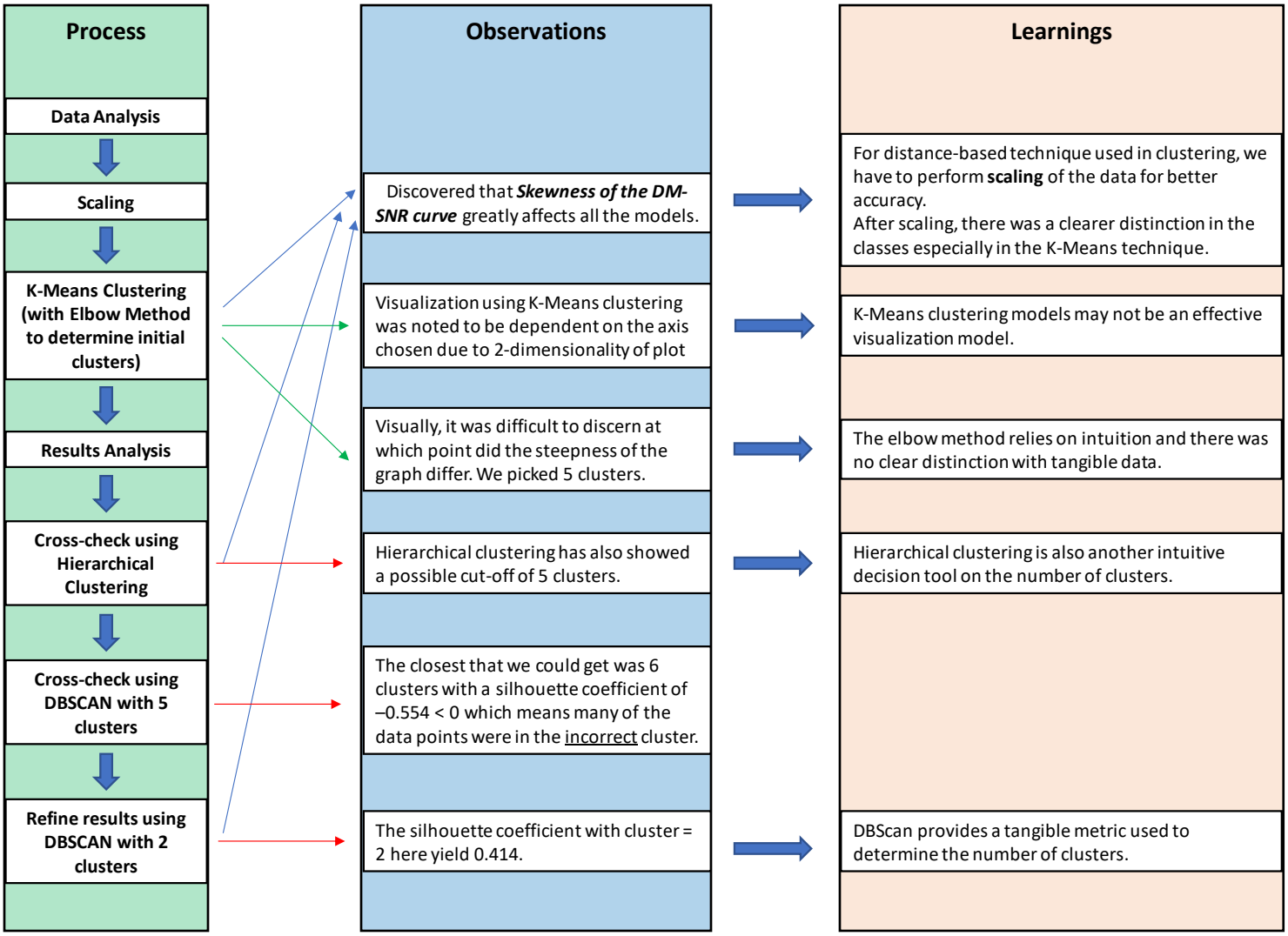
a) Decision tree has a higher accuracy compared to other models because the nature of the problem fits the decision tree model better.

b) Logistic regression does not have a high accuracy result because it might be hard to categorize data using a single hyperplane.

c) SVM runs much slower compared to other models, especially with large datasets.

d) After data cleaning, the accuracy of the decision tree is not really affected (I.e. it still remains at approximately 94%). However, other models (e.g. KNN is affected rom 80% to 92%). This could be because of the structure of the decision tree.

4.2 Unsupervised learning



a) According to the fact that K-means model uses Euclidean distance, it works well if:

i. The clusters have a spherical shape
ii. The clusters are well separated in terms of distance
iii. From visualization, our dataset naturally has two spherical clusters, well separated. Therefore, our K-means model works in helping us group the data into clusters.

5. Other types of Machine Learning

5.1 An Attempt at Neural Network

5.1.1 Code

```
[3]  import seaborn as sns
     import numpy as np

     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegressionCV

     from keras.models import Sequential
     from keras.layers.core import Dense, Activation
     from keras.utils import np_utils

     import numpy as np
     import matplotlib.pyplot as plt
     import pandas as pd
     from sklearn import datasets

[4]  dataset = pd.read_csv('pulsar_stars.csv')
     dataset.head()

 X = dataset.values[:, :8]
 y = dataset.values[:, 8]

 train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.5, test_size=0.5, random_state=0)

[11] def one_hot_encode_object_array(arr):
         '''One hot encode a numpy array of objects (e.g. strings)'''
         uniques, ids = np.unique(arr, return_inverse=True)
         return np_utils.to_categorical(ids, len(uniques))

     train_y_ohe = one_hot_encode_object_array(train_y)
     test_y_ohe = one_hot_encode_object_array(test_y)

[13] model = Sequential()

[14] model.add(Dense(16, input_shape=(8,)))
     model.add(Activation('sigmoid'))

[15] model.add(Dense(2))
     model.add(Activation('softmax'))

[16] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=["accuracy"])

[18] from IPython.display import SVG
     from keras.utils.vis_utils import model_to_dot

     SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))

[19] model.fit(train_X, train_y_ohe, epochs=100, batch_size=1, verbose=1);

[20] loss, accuracy = model.evaluate(test_X, test_y_ohe, verbose=0)
     print("Accuracy = {:.2f}".format(accuracy))
```

```
Accuracy = 0.98
```

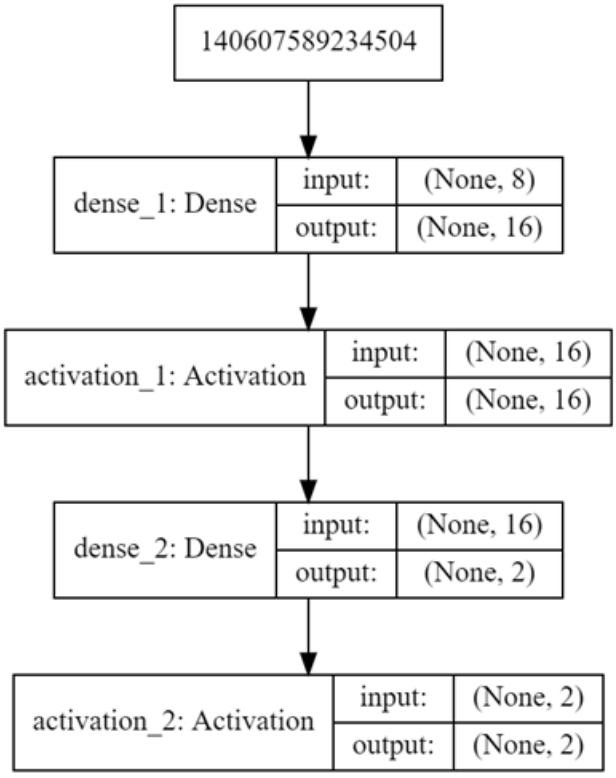
```
model_json = model.to_json()
with open("pulsarstar_nn.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("pulsarstar_nn.h5")
print("Saved model to disk")

[29] from keras.models import model_from_json

     # load json and create model
     json_file = open('pulsarstar_nn.json', 'r')
     loaded_model_json = json_file.read()
     json_file.close()
     loaded_model = model_from_json(loaded_model_json)
     # load weights into new model
     loaded_model.load_weights("pulsarstar_nn.h5")
     print("Loaded model from disk")
```

Saving the model

5.1.2 Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 16)                144
_________________________________________________________________
activation_1 (Activation)    (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 34
_________________________________________________________________
activation_2 (Activation)    (None, 2)                 0
=================================================================
Total params: 178
Trainable params: 178
Non-trainable params: 0
_________________________________________________________________
None
```

5.1.3 Neural Network



5.1.4 Observations

Neural Network Classifier was performed on the Pulsar Stars dataset. There are 8 nodes and 16 perceptrons in the input layer and the output shows 2 nodes. This is a 8-16-2 neural network model. The model could be saved and attached to a web application for usage.

The accuracy from the neural network shown here is 98%. Without the neural network classifer, the accuracy was 97.99%.

The difference between the traditional learning models and the neural network classifier would be more obvious in a larger dataset.

5.2. Processing Images

5.2.1 Code

```
[8]  from keras.applications import VGG16

     vgg16 = VGG16()
     print(vgg16.summary())
```

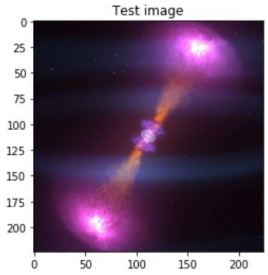
```
[16] # update this path to your own image
     path = 'pulsarstar.jpg'
```

```
[17] def resize_and_crop_image(image_path, width, height):
         """Resizes and crops an image to the desired size
         Args:
             image_path: path to the image
             width: image width
             height: image height
         Returns:
             the resulting image
         """
         from PIL import Image, ImageOps

         img = Image.open(image_path)
         img = ImageOps.fit(img, (width, height))
         return img
```

```
[18] # Preprocess image input
     import numpy as np
     from keras.preprocessing.image import img_to_array
     from keras.applications.vgg16 import preprocess_input, decode_predictions

     img = resize_and_crop_image(path, 224, 224)
     x = img_to_array(img)
     x = preprocess_input(x)
     print('Original image shape:', x.shape)

     # The model expect images in a batch, because it's trained that way
     # Add an extra first axis
     x = np.expand_dims(x, axis=0)
     print('Expected input shape', x.shape)
```

```
[19] y = vgg16.predict(x)

     # display the image
     plt.imshow(img, interpolation='nearest')
     plt.title('Test image')
     plt.show()

     # display the predictions
     #print('Raw predictions:')
     #print(y)

     print('Decoded predictions:')
     print(decode_predictions(y, top=5))
```

5.2.2 Output

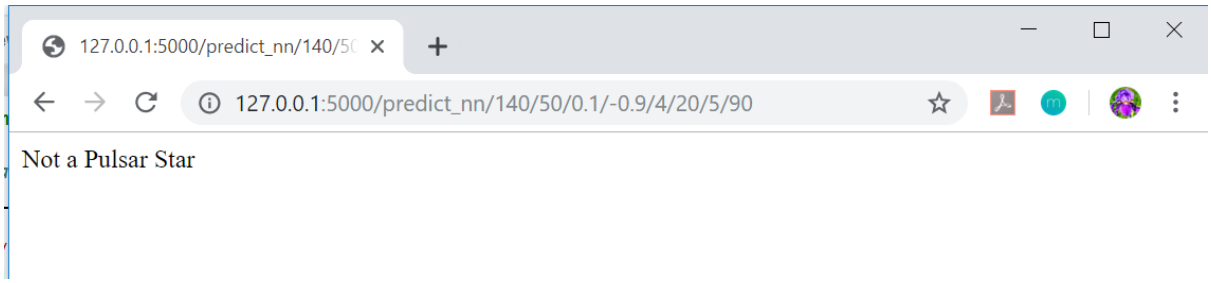



```
Decoded predictions:
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 0us/step
[[('n04286575', 'spotlight', 0.57565236), ('n06874185', 'traffic_light', 0.15130483), ('n04296562', 'stage', 0.0771083), ('n03388043', 'fountain', 0
```

5.2.3 Web Implementation Code

```
from flask import Flask,request
from sklearn.linear_model import LogisticRegressionCV
from keras.models import Sequential
import numpy as np
import tensorflow as tf
# we are creating a variable which is the Flask application
app = Flask(__name__)
from joblib import dump, load
loaded_lr = clf = load('lr_pulsarstar.joblib')
from keras.models import model_from_json
# load json and create model
json_file = open('pulsarstar_nn.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# load weights into new model
loaded_model.load_weights("pulsarstar_nn.h5")
loaded_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=["accuracy"])
graph = tf.get_default_graph()
@app.route('/predict_lr/<v1>/<v2>/<v3>/<v4>/<v5>/<v6>/<v7>/<v8>')
def predict_lr(v1,v2,v3,v4,v5,v6,v7,v8):
    return
loaded_lr.predict([[float(v1),float(v2),float(v3),float(v4),float(v5),float(v6),float(v7),float(v8)]])[0]
@app.route('/predict_nn/<v1>/<v2>/<v3>/<v4>/<v5>/<v6>/<v7>/<v8>')
def predict_nn(v1,v2,v3,v4,v5,v6,v7,v8):
    global graph
    with graph.as_default():
        loaded_model.predict(np.array([[140,50,0.1,-0.9,4,20,5,90]]))
        nn_y =
loaded_model.predict(np.array([[float(v1),float(v2),float(v3),float(v4),float(v5),float(v6),float(v7),floa
t(v8)]]))[0]
        if nn_y[0] == 1:
            return 'Pulsar Star'
        else:
            return 'Not a Pulsar Star'
@app.route('/hello')
def hello():
    return 'Hello'
if __name__ =='__main__':
    app.run(debug=True)
```

5.2.4 Web Implementation Output



6. Appendix

6.1. Data Cleaning - Binning

Method 1: Equal frequency

Binning is done based on the 0th, 25th and the 75th Percentile of the continuous variable data:

- Age:
  - 0: <27
  - 1: 27-40
  - 2: 40-51
  - 3: >51
- Flight Distance:
  - 0: <414
  - 1: 414-844
  - 2: 844-1744
  - 3: >1744
- Departure delay in minutes
  - 0: <12
  - 1: >12

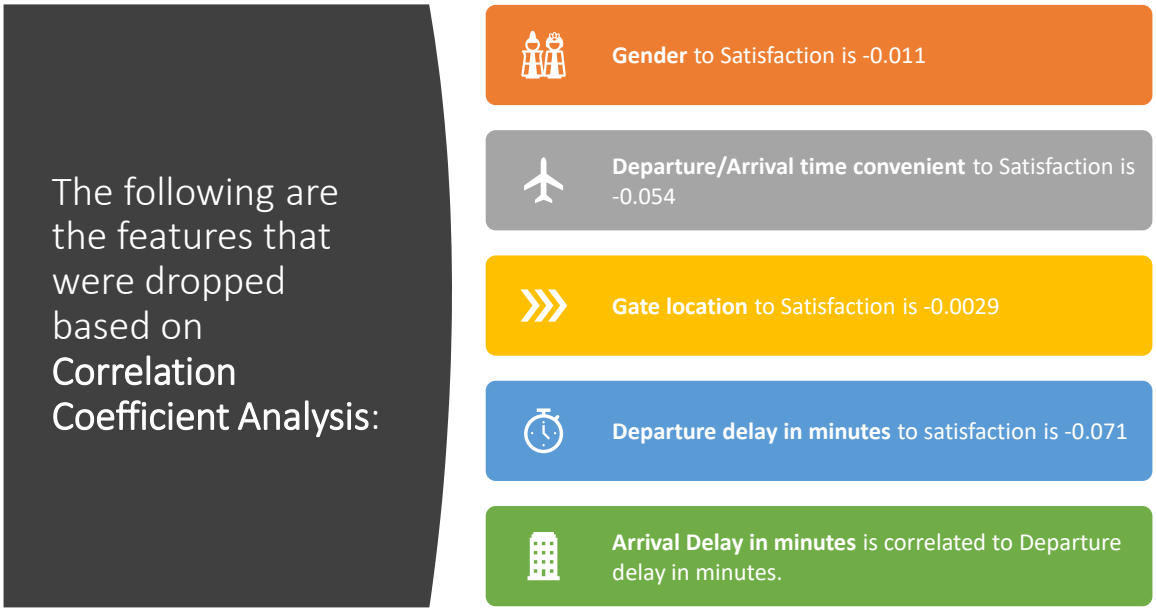
	Age	Flight Distance	Departure/Arrival time convenient	Departure Delay in Minutes	Arrival Delay inMinutes
count	129487.000000	129487.000000	129487.000000	129487.000000	129487.000000
unique	NaN	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN	NaN
mean	39.428761	1190.210662	3.057349	14.643385	15.091129
std	15.117597	997.560954	1.526787	37.932867	38.465650
min	7.000000	31.000000	0.000000	0.000000	0.000000
25%	27.000000	414.000000	2.000000	0.000000	0.000000
50%	40.000000	844.000000	3.000000	0.000000	0.000000
75%	51.000000	1744.000000	4.000000	12.000000	13.000000
max	85.000000	4983.000000	5.000000	1592.000000	1584.000000

Method 2: Equal width

- Categorize age, by an interval of 10 years
- Categorize flight distance, by an interval of 500
- Categorize Departure Delay in Minutes, by an interval of 7.5
- Categorize Arrival Delay in Minutes, by an interval of 7.5

6.2 Data Cleaning – Feature Engineering

The following features were dropped based on the Correlation Coffeficient Analyses



Refer to the analyses on the next page with the respective values circled in dashed orange lines.

	satisfaction_v2	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflightwifiservice	Departure/Arrivaltime convenient	Ease of Onlinebooking	Gate location	Food and drink	Online boarding	Seat comfort	Inflightentertainment	On-board service	Leg room service	Baggage handling	Checkinservice	Inflight service	Cleanliness	Departure Delay inMinutes
satisfaction_v2	1.0000	-0.0115	-0.1859	0.1557	-0.4498	0.4931	0.2470	0.2833	-0.0545	0.1687	-0.0029	0.2112	0.5016	0.3486	0.3983	0.3223	0.3126	0.2487	0.2371	0.2450	0.3069	-0.0705
Gender	-0.0115	1.0000	0.0308	-0.0107	-0.0092	-0.0057	-0.0038	-0.0060	-0.0088	-0.0061	0.0009	-0.0016	0.0449	0.0308	-0.0038	-0.0064	-0.0310	-0.0364	-0.0084	-0.0382	-0.0028	-0.0018
Customer Type	-0.1859	0.0308	1.0000	-0.3337	-0.3082	-0.1056	-0.1921	-0.0059	-0.2069	-0.0182	0.0046	-0.0571	-0.1893	-0.1564	-0.1062	-0.0540	-0.0469	0.0250	-0.0313	0.0236	-0.0814	0.0050
Age	0.1557	-0.0107	-0.3337	1.0000	-0.0290	0.1523	0.0852	0.0175	0.0488	0.0234	0.0006	0.0277	0.2209	0.1734	0.0887	0.0704	0.0555	-0.0360	0.0442	-0.0390	0.0631	-0.0071
Type of Travel	-0.4498	-0.0092	-0.3082	-0.0290	1.0000	-0.5454	-0.2142	-0.1056	0.2572	-0.1339	-0.0299	-0.0687	-0.2238	-0.1274	-0.1527	-0.0597	-0.1395	-0.0329	0.0165	-0.0234	-0.0843	-0.0067
Class	0.4931	-0.0057	-0.1056	0.1523	-0.5454	1.0000	0.3809	0.0368	-0.0901	0.1075	0.0058	0.0882	0.3230	0.2273	0.1963	0.2110	0.2062	0.1614	0.1512	0.1555	0.1386	-0.0064
Flight Distance	0.2470	-0.0038	-0.1921	0.0852	-0.2142	0.3809	1.0000	0.0097	-0.0019	0.0667	0.0033	0.0470	0.1849	0.1340	0.1046	0.0967	0.1167	0.0578	0.0704	0.0559	0.0818	0.0058
Inflight wifiservice	0.2833	-0.0060	-0.0059	0.0175	-0.1056	0.0368	0.0097	1.0000	0.3448	0.7149	0.3385	0.1321	0.4574	0.1214	0.2079	0.1200	0.1604	0.1205	0.0438	0.1103	0.1312	-0.0247
Departure/Arrivaltime convenient	-0.0545	-0.0088	-0.2069	0.0488	0.2572	-0.0901	-0.0019	0.3448	1.0000	0.4377	0.4474	0.0011	0.0722	0.0087	-0.0082	0.0670	0.0106	0.0706	0.0912	0.0722	0.0100	-0.0045
Ease of Onlinebooking	0.1687	-0.0061	-0.0182	0.0234	-0.1339	0.1075	0.0667	0.7149	0.4377	1.0000	0.4602	0.0306	0.4049	0.0286	0.0467	0.0390	0.1093	0.0392	0.0088	0.0354	0.0151	-0.0073
Gate location	-0.0029	0.0009	0.0046	0.0006	-0.0299	0.0058	0.0033	0.3385	0.4474	0.4602	1.0000	-0.0028	0.0026	0.0025	0.0028	-0.0291	-0.0051	0.0011	-0.0393	0.0003	-0.0061	0.0050
Food and drink	0.2112	-0.0016	-0.0571	0.0277	-0.0687	0.0882	0.0470	0.1321	0.0011	0.0306	-0.0028	1.0000	0.2335	0.5760	0.6234	0.0575	0.0332	0.0354	0.0851	0.0354	0.6580	-0.0218
Online boarding	0.5016	0.0449	-0.1893	0.2209	-0.2238	0.3230	0.1849	0.4574	0.0722	0.4049	0.0026	0.2335	1.0000	0.4192	0.2840	0.1543	0.1231	0.0836	0.2042	0.0740	0.3293	-0.0286
Seat comfort	0.3486	0.0308	-0.1564	0.1734	-0.1274	0.2273	0.1340	0.1214	0.0087	0.0286	0.0025	0.5760	0.4192	1.0000	0.6119	0.1307	0.1042	0.0746	0.1898	0.0689	0.6797	-0.0228
Inflightentertainment	0.3983	-0.0038	-0.1062	0.0887	-0.1527	0.1963	0.1046	0.2079	-0.0082	0.0467	0.0028	0.6234	0.2840	0.6119	1.0000	0.4189	0.3006	0.3793	0.1197	0.4066	0.6925	-0.0283
On-board service	0.3223	-0.0064	-0.0540	0.0704	-0.0597	0.2110	0.0967	0.1200	0.0670	0.0390	-0.0291	0.0575	0.1543	0.1307	0.4189	1.0000	0.3579	0.5204	0.2446	0.5515	0.1222	-0.0315
Leg room service	0.3126	-0.0310	-0.0469	0.0555	-0.1395	0.2062	0.1167	0.1604	0.0106	0.1093	-0.0051	0.0332	0.1231	0.1042	0.3006	0.3579	1.0000	0.3716	0.1527	0.3698	0.0968	-0.0080
Baggage handling	0.2487	-0.0364	0.0250	-0.0360	-0.0329	0.1614	0.0578	0.1205	0.0706	0.0392	0.0011	0.0354	0.0836	0.0746	0.3793	0.5204	0.3716	1.0000	0.2347	0.6295	0.0971	-0.0164
Checkin service	0.2371	-0.0084	-0.0313	0.0442	0.0165	0.1512	0.0704	0.0438	0.0912	0.0088	-0.0393	0.0851	0.2042	0.1898	0.1197	0.2446	0.1527	0.2347	1.0000	0.2377	0.1766	-0.0252
Inflight service	0.2450	-0.0382	0.0236	-0.0390	-0.0234	0.1555	0.0559	0.1103	0.0722	0.0354	0.0003	0.0354	0.0740	0.0689	0.4066	0.5515	0.3698	0.6295	0.2377	1.0000	0.0906	-0.0350
Cleanliness	0.3069	-0.0028	-0.0814	0.0631	-0.0843	0.1386	0.0818	0.1312	0.0100	0.0151	-0.0061	0.6580	0.3293	0.6797	0.6925	0.1222	0.0968	0.0971	0.1766	0.0906	1.0000	-0.0196
Departure Delay in Minutes	-0.0705	-0.0018	0.0050	-0.0071	-0.0067	-0.0064	0.0058	-0.0247	-0.0045	-0.0073	0.0050	-0.0218	-0.0286	-0.0228	-0.0283	-0.0315	-0.0080	-0.0164	-0.0252	-0.0350	-0.0196	1.0000