UNIVERSITY OF CANBERRA
AUSTRALIA'S CAPITAL UNIVERSITY

Faculty of Science and Technology

Assignment Coversheet

Student ID number & Student Name	u3145190 Jacinda Zou
Unit name	Software Technology
Unit number	8995
Unit Tutor	Girija
Assignment name	ST1 Capstone Project – Semester 1 2023
Due date	12 May 2023
Date submitted	12 May 2023

You must keep a photocopy or electronic copy of your assignment.

Student declaration

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

Signature of student: Jing Zou **Date:** **12/05/2023**

UNIVERSITY OF CANBERRA
AUSTRALIA'S CAPITAL UNIVERSITY

Table of Contents

UNIVERSITY OF CANBERRA
AUSTRALIA'S CAPITAL UNIVERSITY

Introduction

This document outlines the specifics of the Python Capstone Project for the ST1 unit, adhering to the project requirements stated in the assignment handout. I have undertaken the project using a shop customer dataset accessible in Kaggle's data repositories [1].

For a shop proprietor, comprehending the target customers is crucial in making informed business decisions, such as adjusting product inventory, selecting branch store locations, and introducing new services. However, predicting customer behaviour has always been a challenge for businesses. Therefore, developing a software tool that utilizes existing customer data to forecast customer behaviour would benefit the community.

This report provides a comprehensive overview of the prototype software platform, which includes several Python software tools developed as part of this capstone project. The development process follows a data-driven scientific approach, encompassing exploratory data analysis, predictive analytics, and implementation as a desktop Tkinter application. The methodology employed in this project is elaborated in the subsequent section.

Methodology

The methodology employed in the software platform's development comprises three stages, which are described as follows:

1. Design and development of decision support algorithms: This stage involves creating and refining algorithms that utilize exploratory data analysis and predictive analytics. The goal is to identify the most effective algorithm for addressing a real-world problem.
2. Implementation of the best-performing algorithm: Once the optimal algorithm has been identified, it is implemented as a desktop Tkinter software tool. This involves coding and configuring the algorithm within the Tkinter framework to create a user-friendly interface.
3. Deployment of the tool as a web or cloud-enabled platform: The final stage involves making the software tool accessible as a web or cloud-based platform. This enables users to access the tool remotely through a web browser or cloud service, expanding its reach and usability.

Stage 1: Algorithm Design Stage

The design of algorithms for exploratory data analysis and predictive analytics may vary depending on the complexity of the problem and the dataset utilized. However, the workflow for algorithm development will be as outlined in the Figure 1 schematic shown below:

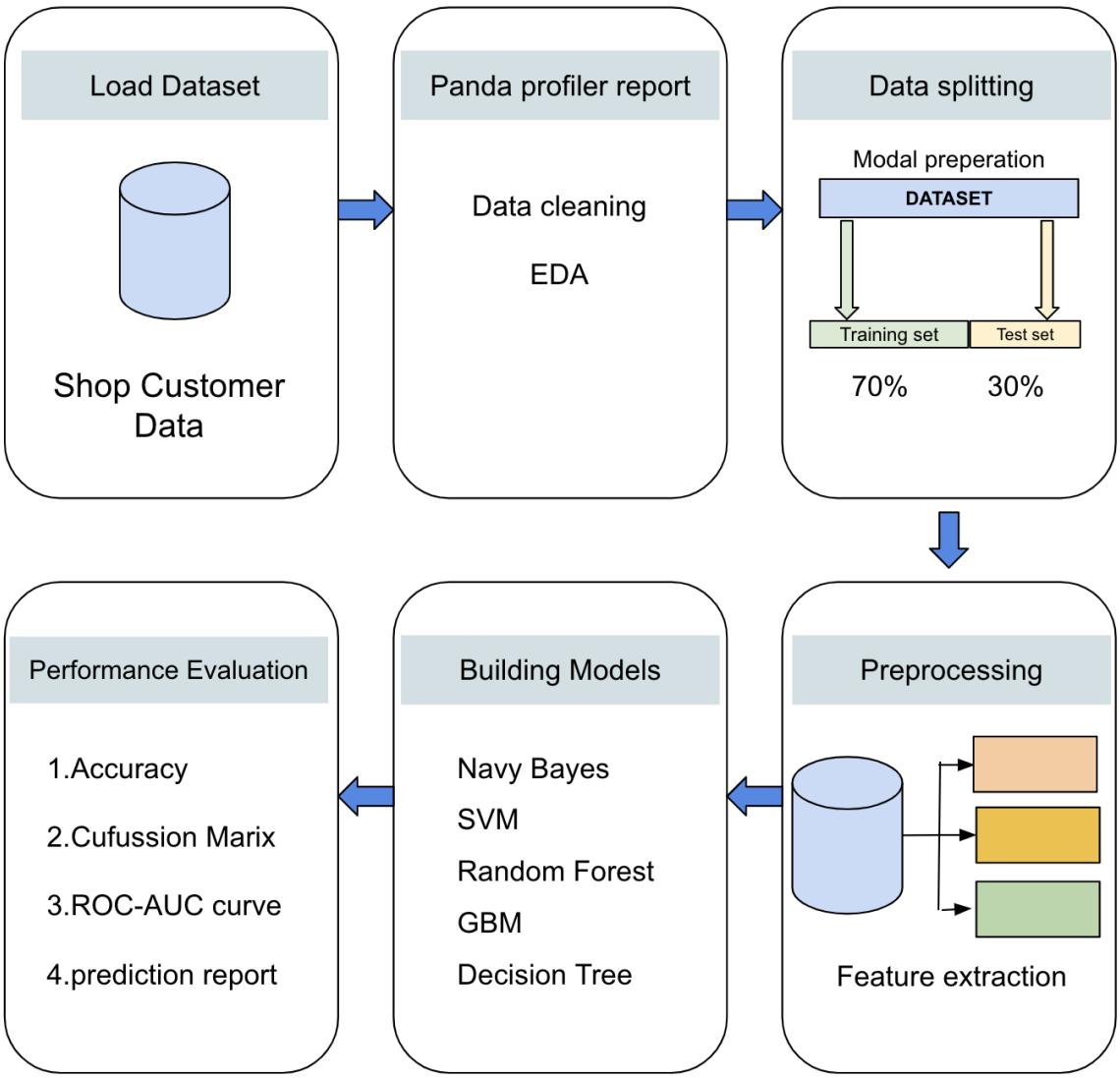


Figure 1: Schematic for Algorithm Design Methodology for Valued Customer Prediction

The details of each building block in Figure 1 schematic for algorithm design is described in the next few sections.

Dataset Description

For this project, only one dataset is used, and it can be found on Kaggle [1]. The dataset consists of 2000 observations, 8 columns. The dataset includes information on various variables such as gender, age, profession, family size, etc. This dataset can be used to gain insights into customer behaviour and preferences in a retail store. The task at hand is to develop a software tool to predict

and determine the potential target customers, so it can help the shop owner make business decisions.

Exploratory Data Analysis

To commence the software development process, the first stage involved comprehending the data, conducting exploratory data analysis, and producing visualizations. Google Colab has virtual hardware and resources, eliminating the need for extra physical hardware and enabling the programme to be run directly on a web browser. Therefore, it was chosen as the experimentation environment. The development scripts were created using Python and executed on an online Jupyter notebook within Google Colab, which can be saved virtually on Google Drive with only a free Google account and without any additional configurations. Before performing exploratory data analysis, it was necessary to import some Python libraries for EDA and acquire the dataset using the Python script.

```
from google.colab import drive
drive.mount("/content/drive")


#Import Required Packages for EDA
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno
import plotly.graph_objects as go
import plotly.express as px
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')

#Read the dataset/s

df = pd.read_csv('/content/drive/…../Customers.csv')
```

(1) The EDA starts with understanding the basic description of data as described next:

```
#1. Checking description(first 5 and last 5 rows)
df.head()
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1	Male	19	15000	39	Healthcare	1	4
1	2	Male	21	35000	81	Engineer	3	3
2	3	Female	20	86000	6	Engineer	1	1
3	4	Female	23	59000	77	Lawyer	0	2
4	5	Female	31	38000	40	Entertainment	2	6

```
df.tail() #last 5 rows
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
1995	1996	Female	71	184387	40	Artist	8	7
1996	1997	Female	91	73158	32	Doctor	7	7
1997	1998	Male	87	90961	14	Healthcare	9	2
1998	1999	Male	77	182109	4	Executive	7	2
1999	2000	Male	90	110610	52	Entertainment	5	2

```
#rows and columns-data shape(attributes & samples)

df.shape
```

```
(2000, 8)
```

```
# name of the attributes
df.columns
```

```
Index(['CustomerID', 'Gender', 'Age', 'Annual Income ($)',
       'Spending Score (1-100)', 'Profession', 'Work Experience',
       'Family Size'],
      dtype='object')
```

```
#unique values for each attribute
df.nunique()
```

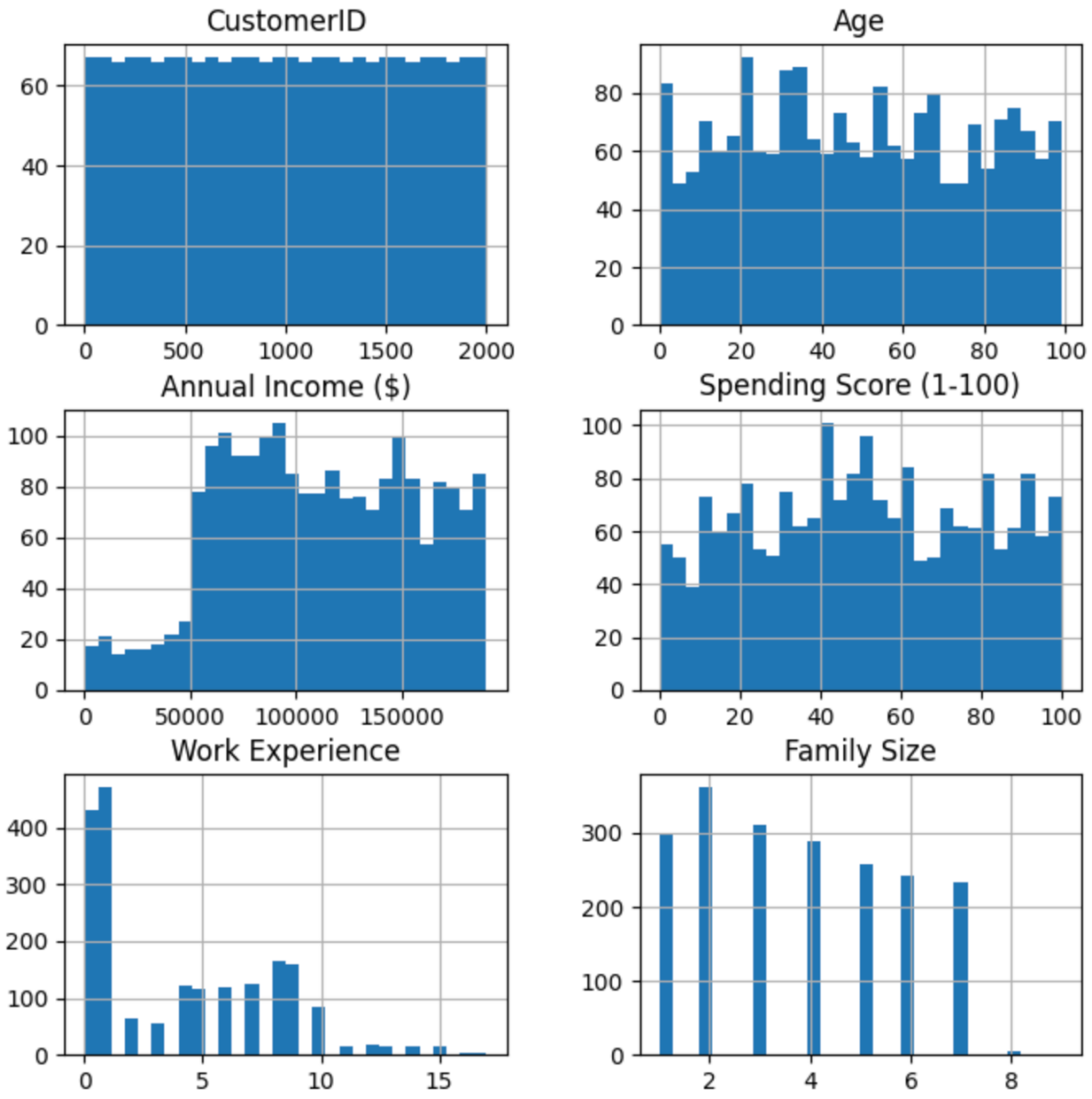
```
CustomerID                2000
Gender                       2
Age                        100
Annual Income ($)         1786
Spending Score (1-100)     101
Profession                   9
Work Experience             18
Family Size                  9
dtype: int64
```

```
#Complete info about data frame
df.info()
```

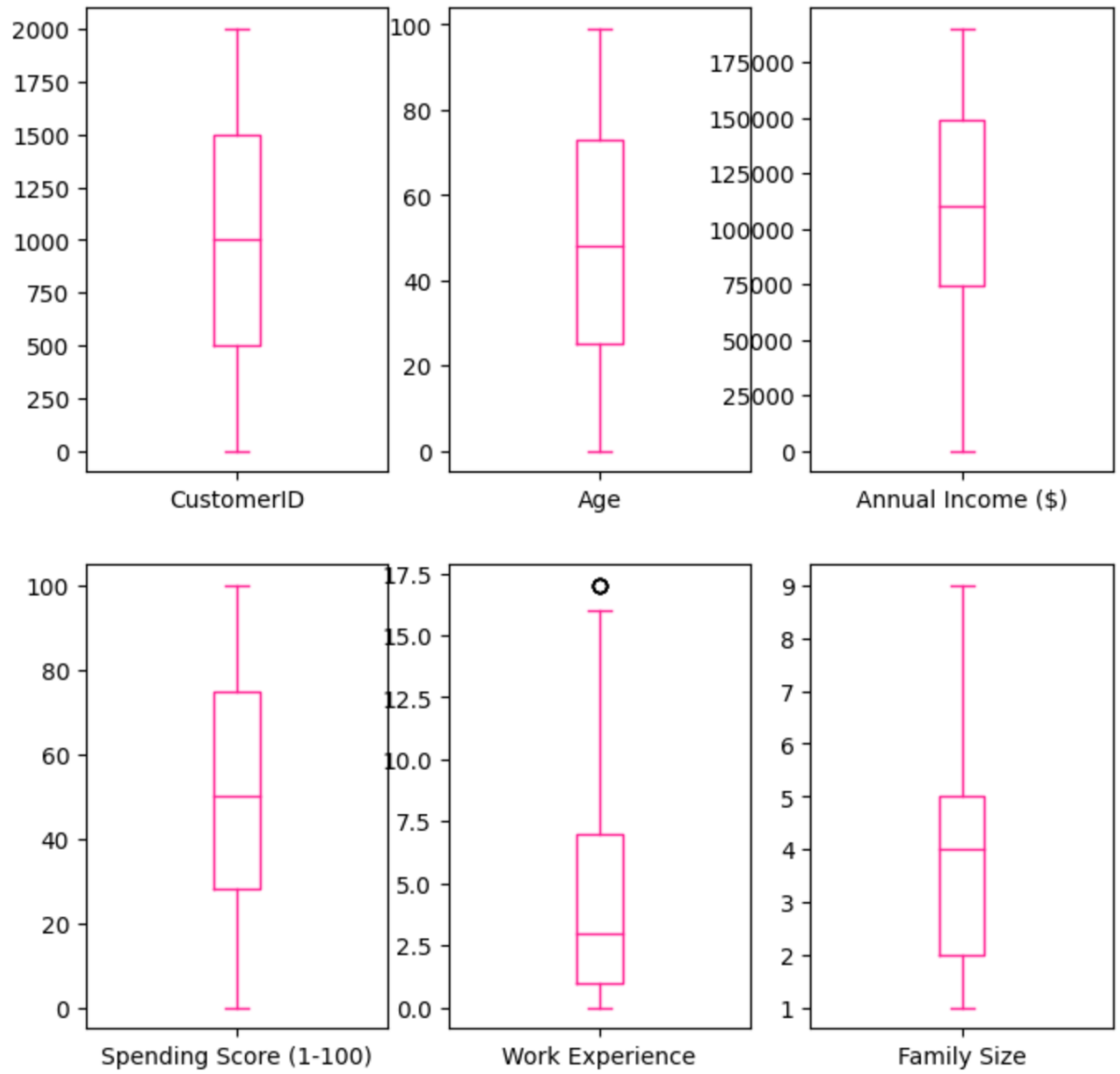
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 8 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              2000 non-null   int64
 1   Gender                  2000 non-null   object
 2   Age                     2000 non-null   int64
 3   Annual Income ($)       2000 non-null   int64
 4   Spending Score (1-100)  2000 non-null   int64
 5   Profession              1965 non-null   object
 6   Work Experience         2000 non-null   int64
 7   Family Size             2000 non-null   int64
dtypes: int64(6), object(2)
memory usage: 125.1+ KB
```

```
#3. Visualising data  distribution in detail
```

```
fig = plt.figure(figsize =(8,8))
ax=fig.gca()
df.hist(ax=ax,bins =30)
plt.show()
```



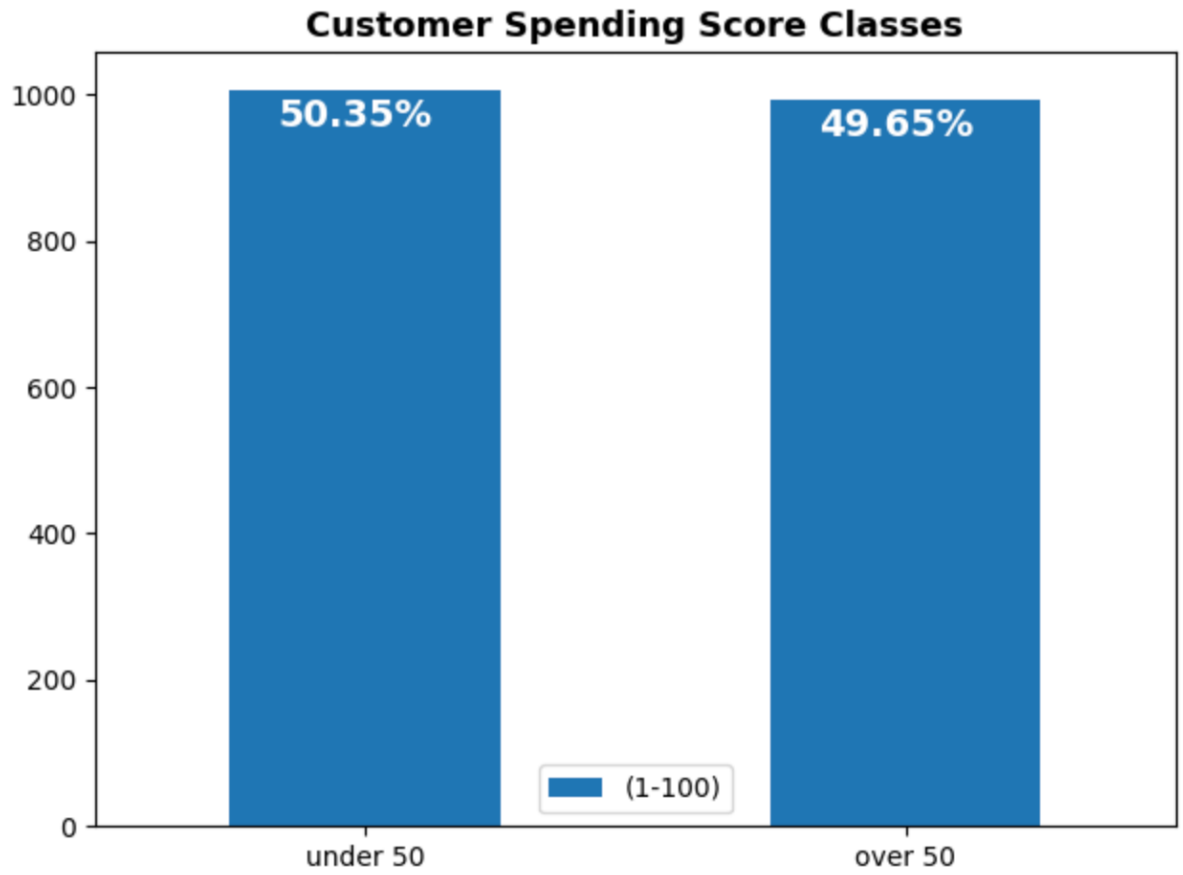
```
#detecting outliers
df.plot(kind='box', subplots=True,
       layout=(2,3),sharex=False,sharey=False, figsize=(8, 8),
color='deeppink');
```



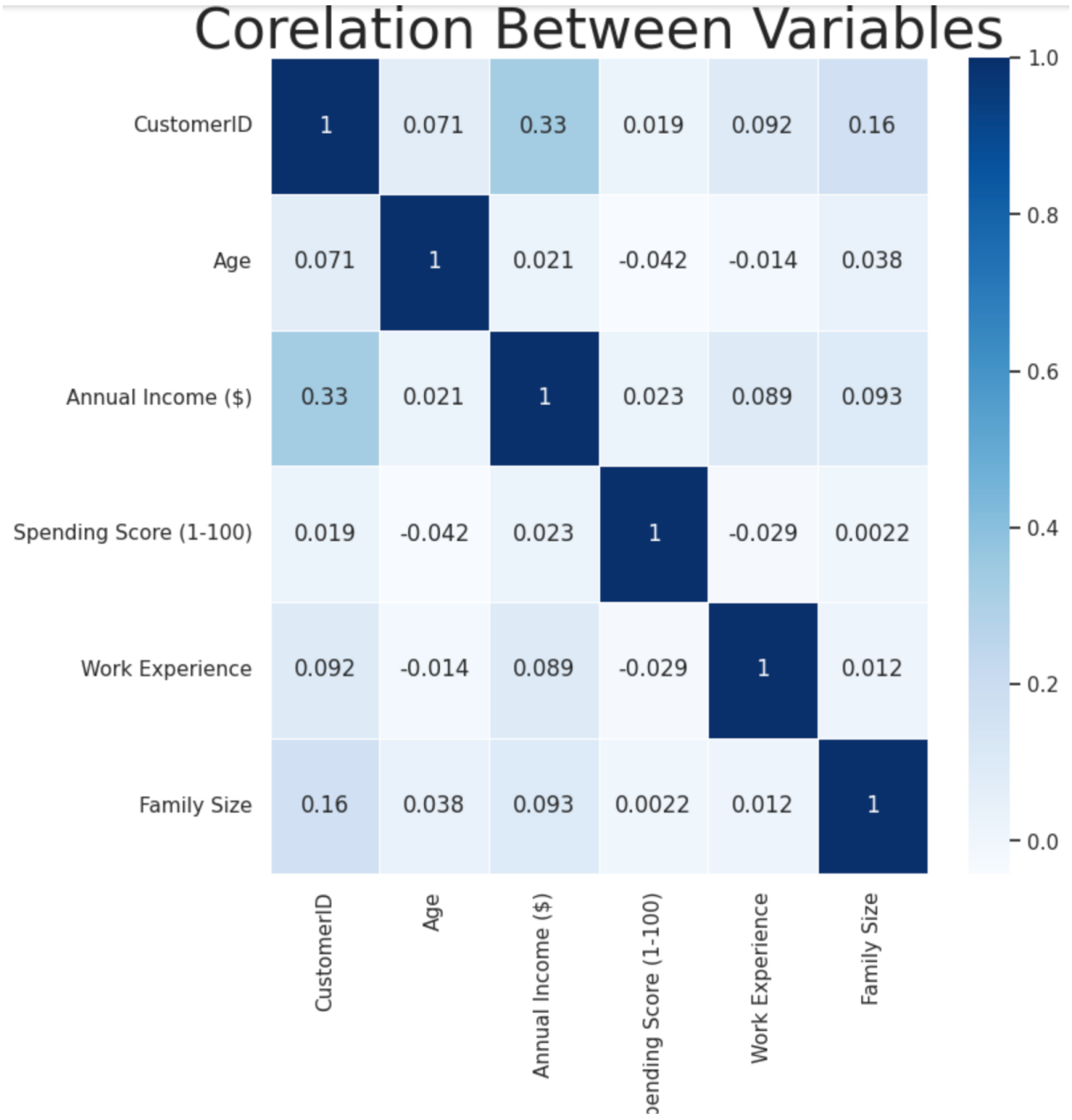

```
#checking target value distribution
under50 = df[df['Spending Score (1-100)'] <=50]
over50 = df[df['Spending Score (1-100)'] >50]
print('under50 ',len(under50['Spending Score (1-100)']))
print('over50 ',len(over50['Spending Score (1-100)']))

data = [len(under50['Spending Score (1-100)']),len(over50['Spending
Score (1-100)'])]

name = ["under 50", "over 50"]
```

```
ax = pd.DataFrame(data,columns=['(1-100)']).plot(kind='bar')
ax.set_title("Customer Spending Score Classes", fontsize = 13, weight =
'bold')
ax.set_xticklabels (name, rotation = 0)


# To calculate the percentage
totals = []
for i in ax.patches:
   totals.append(i.get_height())
total = sum(totals)
for i in ax.patches:
   ax.text(i.get_x()+.09, i.get_height()-50, \
           str(round((i.get_height()/total)*100, 2))+'%', fontsize=14,
               color='white', weight = 'bold')

plt.tight_layout()
```

```
under50  1007
over50  993
```



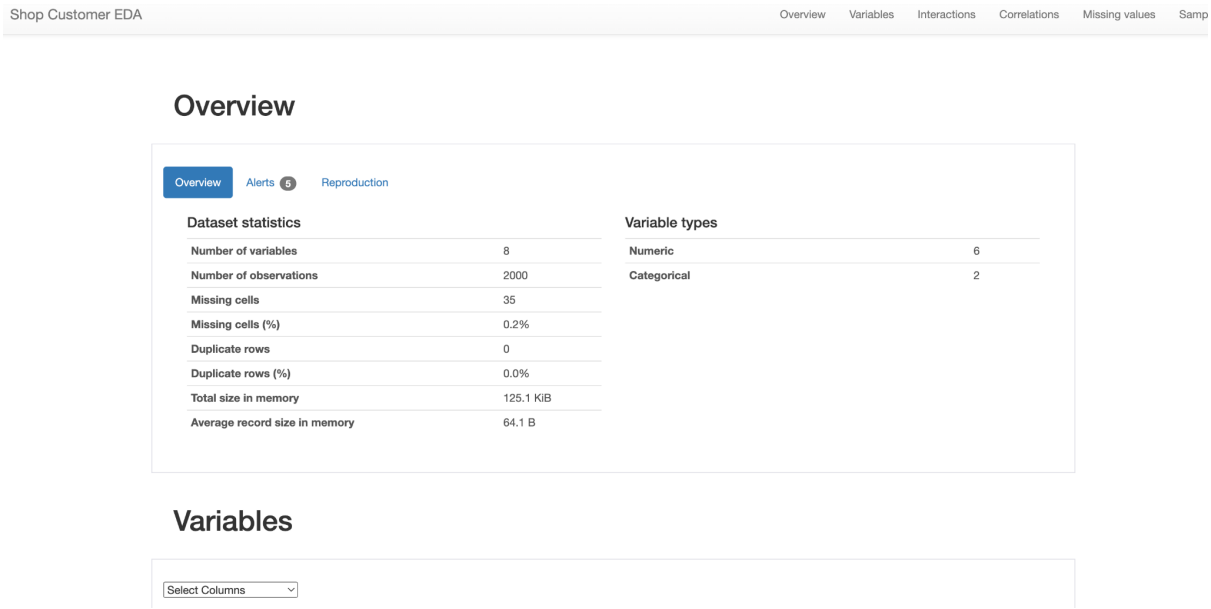

```
#check correlation between variables
sns.set(style="white")
plt.rcParams['figure.figsize'] = (8, 8)
sns.heatmap(df.corr(), annot = True, linewidths=.5, cmap="Blues")
plt.title('Corelation Between Variables', fontsize = 30)
plt.show()
```



!pip install https://github.com/pandas-profiling/pandas-profiling/archive/master.zip

```
#obtain full profiler report
#restart kernel
#re-run import libraries and data
import pandas as pd
import numpy as np
```

```
from pandas_profiling import ProfileReport
profile = ProfileReport(df,title="Shop Customer EDA",
                        html={'style':{'full_width':True}})
profile.to_notebook_iframe()
```



Predictive Data Analytics Stage

Predictive analytics involves several essential processing steps. These steps encompass pre-processing, classifier comparison to determine the most suitable machine learning classifier, and performance evaluation utilizing various objective metrics such as accuracy, classification report, confusion matrix, ROC-AUC curve, and prediction report obtained through the utilization of the Python scikit-learn package. The details of each of these steps are described below.

- Pre-processing: Pre-processing is the initial step in predictive analytics, which involves preparing the data for analysis. This includes handling missing values, data normalization or standardization, feature scaling, and handling categorical variables through encoding techniques such as one-hot encoding or label encoding. Pre-processing ensures that the data is in a suitable format for training and evaluation.

- Classifier Comparison: The next step involves comparing different machine learning classifiers to identify the most effective one for the given problem. This typically includes trying out multiple algorithms such as decision trees, random forests, support vector machines (SVM), logistic regression, and neural networks. Each classifier is trained on the

pre-processed data and evaluated using cross-validation techniques or a separate validation dataset.

- Performance Evaluation: Performance evaluation is a critical step in predictive analytics. Various objective metrics are used to assess the performance of the classifiers. These metrics include accuracy, which measures the overall correctness of the predictions; classification report, which provides precision, recall, F1-score, and support for each class; confusion matrix, which illustrates the predicted and actual class labels; and the ROC-AUC curve, which shows the trade-off between true positive rate and false positive rate for different classification thresholds. Additionally, prediction reports provide detailed insights into the predictions made by the classifiers.

```
#pre-processing
from sklearn.exceptions import DataDimensionalityWarning
#encode object columns to integers
from sklearn import preprocessing
from sklearn.preprocessing import OrdinalEncoder

for col in df:
  if df[col].dtype =='object':
    df[col]=OrdinalEncoder().fit_transform(df[col].values.reshape(-1,1))
df
```

	CustomerID	Gender	Age	Annual Income ($)	Spending Score (1-100)	Profession	Work Experience	Family Size
0	1	1.0	19	15000	39	5.0	1	4
1	2	1.0	21	35000	81	2.0	3	3
2	3	0.0	20	86000	6	2.0	1	1
3	4	0.0	23	59000	77	7.0	0	2
4	5	0.0	31	38000	40	3.0	2	6
...	...	...	...	...	...	...	...	...
1995	1996	0.0	71	184387	40	0.0	8	7
1996	1997	0.0	91	73158	32	1.0	7	7
1997	1998	1.0	87	90961	14	5.0	9	2
1998	1999	1.0	77	182109	4	4.0	7	2
1999	2000	1.0	90	110610	52	3.0	5	2

2000 rows × 8 columns

```
class_label = df['Spending Score (1-100)']
df = df.drop(['Spending Score (1-100)'], axis =1)
df = (df-df.min())/(df.max()-df.min())
df['Spending Score (1-100)']= class_label
df['Value Customer'] = (df['Spending Score (1-100)'] >
50).astype(int)
df
```

	CustomerID	Gender	Age	Annual Income ($)	Profession	Work Experience	Family Size	Spending Score (1-100)	Value Customer
0	0.000000	1.0	0.191919	0.078958	0.625	0.058824	0.375	39	0
1	0.000500	1.0	0.212121	0.184236	0.250	0.176471	0.250	81	1
2	0.001001	0.0	0.202020	0.452694	0.250	0.058824	0.000	6	0
3	0.001501	0.0	0.232323	0.310569	0.875	0.000000	0.125	77	1
4	0.002001	0.0	0.313131	0.200027	0.375	0.117647	0.625	40	0
...	...	...	...	...	...	...	...	...	...
1995	0.997999	0.0	0.717172	0.970591	0.000	0.470588	0.750	40	0
1996	0.998499	0.0	0.919192	0.385095	0.125	0.411765	0.750	32	0
1997	0.998999	1.0	0.878788	0.478808	0.625	0.529412	0.125	14	0
1998	0.999500	1.0	0.777778	0.958600	0.500	0.411765	0.125	4	0
1999	1.000000	1.0	0.909091	0.582238	0.375	0.294118	0.125	52	1

2000 rows × 9 columns

```
#pre-processing
customer_data = df.copy()
le = preprocessing.LabelEncoder()
id = le.fit_transform(list(customer_data["CustomerID"]))
#Identification
gender = le.fit_transform(list(customer_data["Gender"])) # gender (1 =
male; 0 = female)
age = le.fit_transform(list(customer_data["Age"]))
annual_income = le.fit_transform(list(customer_data["Annual Income
($)"]))
profession = le.fit_transform(list(customer_data["Profession"])) #
Proffesion of a customer
spending_score = le.fit_transform(list(customer_data["Spending Score
(1-100)"])) # Score assigned by the shop, based on customer behavior
and spending nature
work_experience = le.fit_transform(list(customer_data["Work
Experience"])) # In years
family_size = le.fit_transform(list(customer_data["Family Size"]))
#Family members of a customer
value_customer = le.fit_transform(list(customer_data["Value
Customer"])) #customer spend more than 50 score
```

Model Preparation and Development

Steps used for machine learning model preparation are described below:

- Convert the dataframe to training and validation/test subsets by taking a random sample of 80% of the data and defining it as train subset. This leaves 20% of the data for validation/testing
- Create the validation/test set by dropping all of the rows that comprise the training set from the dataframe.
- Create y_train by using using the last column of train (target class).
- Create x_train by using all of the columns in train except the last one.
- The validation set of y_val and x_val or (y_test and x_test), can be created using the same methodology that used to create y_train and x_train

```
x = list(zip(gender, age, annual_income, work_experience, family_size))
y = list(value_customer)
# Predictive analytics model development by comparing different Scikit-
learn classification algorithms
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
# Test options and evaluation metric
num_folds = 5
seed = 7
scoring = 'accuracy'
```

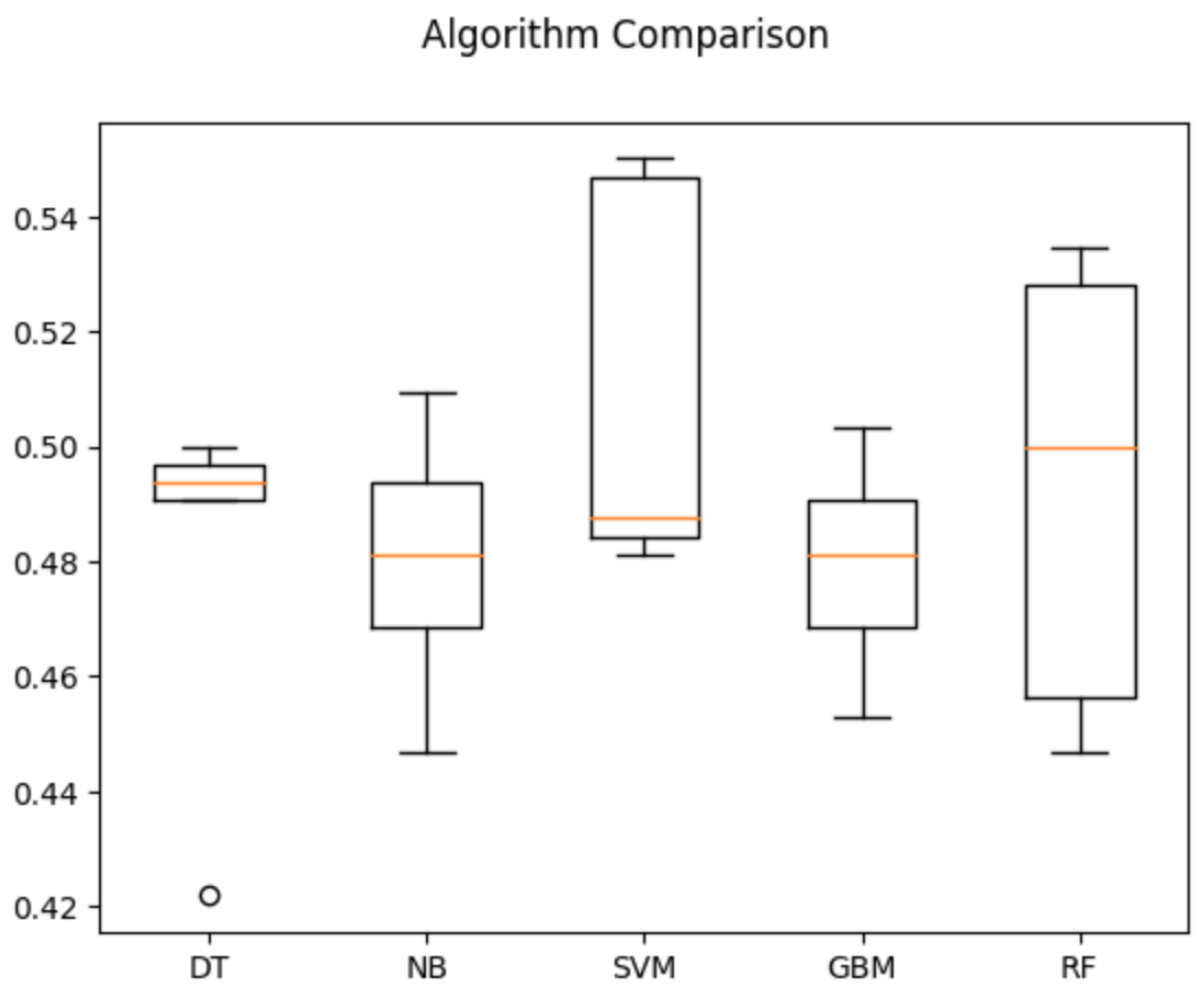
```
# Model Test/Train
# Splitting what we are trying to predict into 4 different arrays -
# X train is a section of the x array(attributes) and vise versa for
Y(features)
# The test data will test the accuracy of the model created
x_train, x_test, y_train, y_test =
sklearn.model_selection.train_test_split(x, y, test_size = 0.20,
random_state=seed)
#splitting 20% of our data into test samples. If we train the model
with higher data it already has seen that information and knows


#size of train and test subsets after splitting
np.shape(x_train), np.shape(x_test)
```

```
((1600, 5), (400, 5))
```

```
models = []
models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RF', RandomForestClassifier()))
# evaluate each model in turn
results = []
names = []
print("Performance on Training set")
for name, model in models:
 kfold = KFold(n_splits=num_folds,shuffle=True,random_state=seed)
 cv_results = cross_val_score(model, x_train, y_train, cv=kfold,
scoring='accuracy')
 results.append(cv_results)
 names.append(name)
 msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
 msg += '\n'
 print(msg)
```

```
Performance on Training set
DT: 0.480625 (0.029541)

NB: 0.480000 (0.021342)

SVM: 0.510000 (0.031462)

GBM: 0.479375 (0.017298)

RF: 0.493125 (0.035980)
```

```
# Compare Algorithms' Performance
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



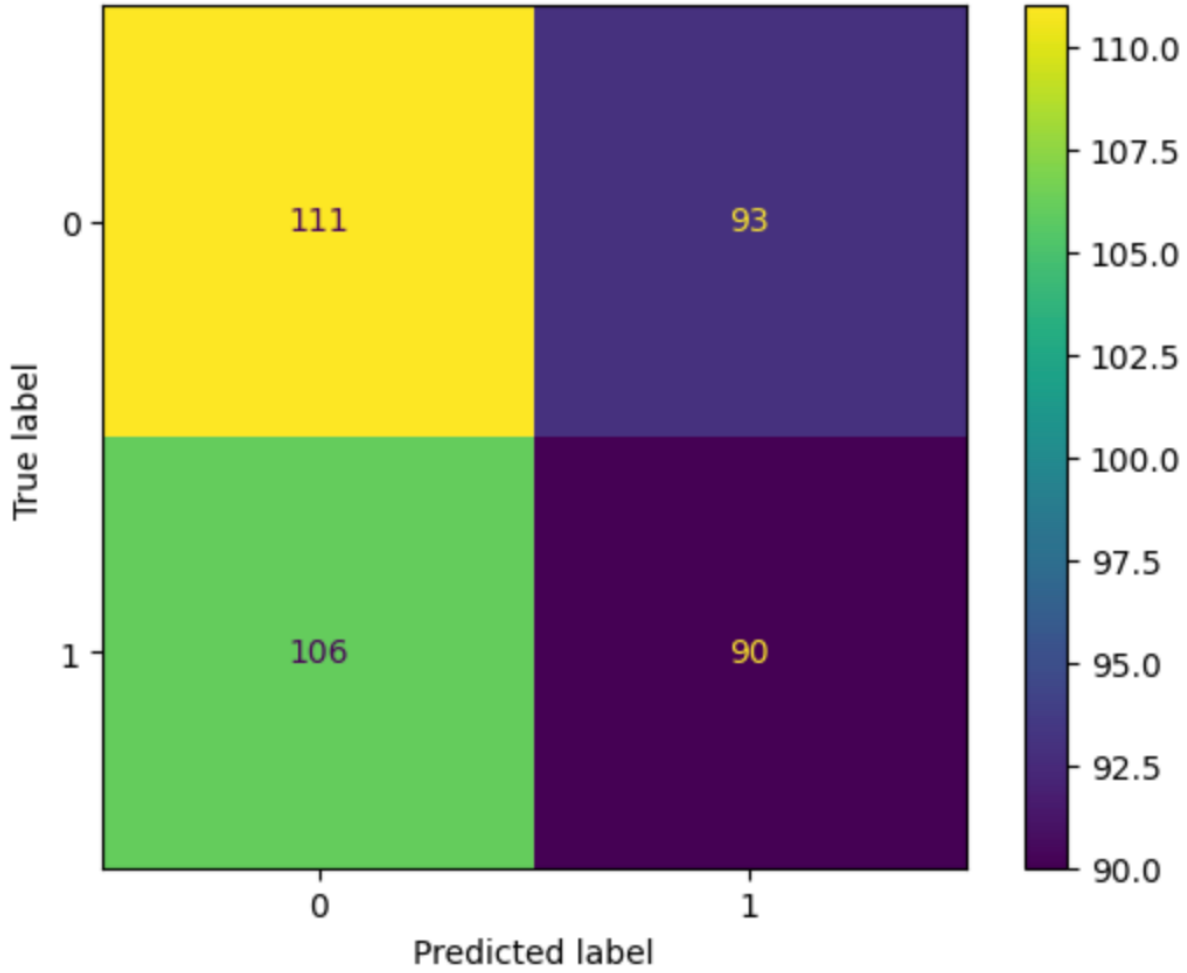
```
#Model Evaluation by testing with independent/external test data set.
# Make predictions on validation/test dataset


models.append(('DT', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('GBM', GradientBoostingClassifier()))
models.append(('RF', RandomForestClassifier()))
dt = DecisionTreeClassifier()
nb = GaussianNB()
svm = SVC()
gb = GradientBoostingClassifier()
rf = RandomForestClassifier()


best_model = rf
best_model.fit(x_train, y_train)
y_pred = best_model.predict(x_test)
print("Best Model Accuracy Score on Test Set:", accuracy_score(y_test,
y_pred))
```

```
Best Model Accuracy Score on Test Set: 0.5025
```

```
#Model Performance Evaluation Metric 1 - Classific
ation Report
print(classification_report(y_test, y_pred))
```

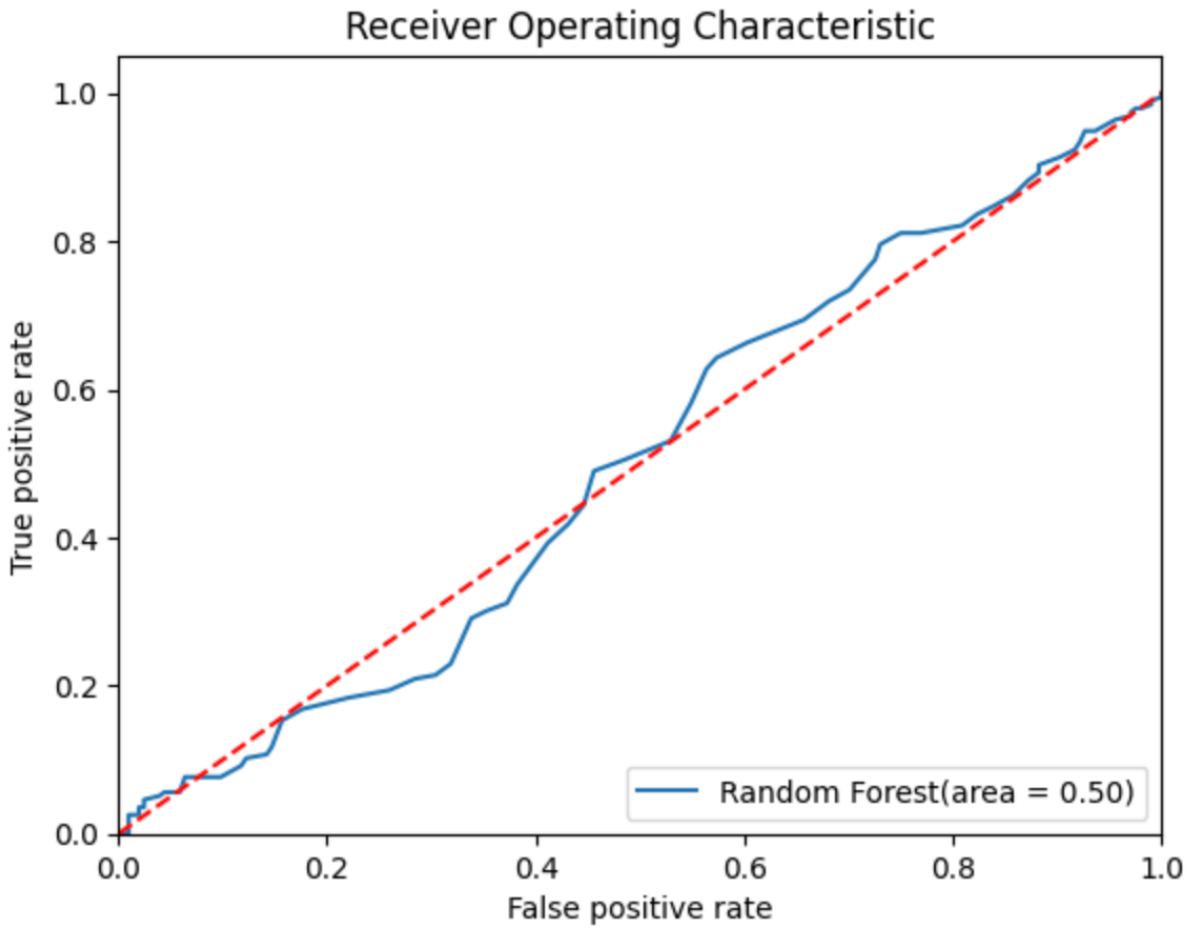
	precision	recall	f1-score	support
0	0.51	0.54	0.53	204
1	0.49	0.46	0.47	196
accuracy			0.50	400
macro avg	0.50	0.50	0.50	400
weighted avg	0.50	0.50	0.50	400

```
#Model Performance Evaluation Metric 2
#Confusion matrix
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
plt.show()
```




```
#Model Evaluation Metric 3- ROC-AUC curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

best_model = rf
best_model.fit(x_train, y_train)
rf_roc_auc = roc_auc_score(y_test,best_model.predict(x_test))
fpr,tpr,thresholds = roc_curve(y_test, best_model.predict_proba(x_test)[:,1
])

plt.figure()
plt.plot(fpr,tpr,label = 'Random Forest(area = %0.2f)'% rf_roc_auc)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False positive rate')
```

```
plt.ylabel('True positive rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.savefig('LOC_ROC')
plt.show()
```




```
#Model Evaluation Metric 4-prediction report
for x in range(len(y_pred)):
  print("Predicted: ", y_pred[x], "Actual: ", y_test[x], "Data: ", x_test[x
],)
```

```
Predicted:  0 Actual:  1 Data:  (0, 22, 1349, 10, 6)
Predicted:  0 Actual:  1 Data:  (0, 82, 1411, 1, 0)
Predicted:  1 Actual:  1 Data:  (1, 70, 469, 0, 1)
Predicted:  1 Actual:  1 Data:  (0, 37, 1277, 1, 1)
Predicted:  1 Actual:  0 Data:  (0, 62, 342, 0, 0)
Predicted:  1 Actual:  0 Data:  (0, 4, 1697, 1, 0)
Predicted:  1 Actual:  1 Data:  (0, 1, 129, 1, 1)
Predicted:  1 Actual:  1 Data:  (0, 43, 482, 1, 1)
Predicted:  0 Actual:  1 Data:  (1, 93, 940, 0, 5)
Predicted:  0 Actual:  1 Data:  (0, 26, 839, 0, 4)
Predicted:  0 Actual:  1 Data:  (0, 78, 169, 14, 6)
Predicted:  1 Actual:  1 Data:  (0, 47, 1765, 6, 1)
Predicted:  1 Actual:  0 Data:  (1, 44, 1395, 9, 4)
Predicted:  1 Actual:  1 Data:  (1, 8, 1327, 8, 5)
Predicted:  1 Actual:  1 Data:  (0, 95, 1421, 6, 1)
Predicted:  0 Actual:  0 Data:  (1, 96, 647, 8, 1)
Predicted:  0 Actual:  0 Data:  (0, 91, 713, 4, 0)
Predicted:  0 Actual:  0 Data:  (0, 3, 1703, 1, 1)
```

```
Predicted:  0 Actual:  1 Data:  (1, 81, 690, 0, 6)
Predicted:  0 Actual:  0 Data:  (0, 40, 18, 0, 0)
Predicted:  0 Actual:  0 Data:  (1, 81, 323, 1, 3)
Predicted:  0 Actual:  0 Data:  (0, 53, 506, 16, 2)
Predicted:  1 Actual:  1 Data:  (1, 84, 1572, 6, 1)
Predicted:  1 Actual:  1 Data:  (1, 75, 1230, 1, 1)
Predicted:  1 Actual:  1 Data:  (1, 34, 1104, 10, 2)
Predicted:  1 Actual:  0 Data:  (0, 40, 460, 4, 0)
Predicted:  0 Actual:  1 Data:  (0, 84, 125, 1, 1)
Predicted:  0 Actual:  1 Data:  (0, 32, 243, 6, 2)
Predicted:  0 Actual:  0 Data:  (0, 78, 421, 10, 6)
Predicted:  1 Actual:  0 Data:  (1, 85, 1191, 0, 5)
Predicted:  0 Actual:  0 Data:  (0, 21, 1437, 1, 6)
Predicted:  0 Actual:  1 Data:  (0, 5, 973, 10, 2)
Predicted:  1 Actual:  1 Data:  (0, 64, 587, 5, 6)
Predicted:  0 Actual:  1 Data:  (1, 52, 1283, 4, 3)
Predicted:  0 Actual:  1 Data:  (1, 76, 450, 6, 0)
Predicted:  0 Actual:  1 Data:  (0, 64, 1612, 9, 4)
Predicted:  0 Actual:  0 Data:  (0, 83, 1598, 4, 3)
Predicted:  0 Actual:  1 Data:  (1, 39, 40, 0, 0)
Predicted:  0 Actual:  1 Data:  (0, 46, 241, 3, 2)
Predicted:  1 Actual:  0 Data:  (1, 48, 144, 1, 1)
Predicted:  0 Actual:  0 Data:  (1, 90, 216, 4, 1)
Predicted:  1 Actual:  0 Data:  (0, 38, 82, 1, 0)
Predicted:  1 Actual:  0 Data:  (1, 88, 1062, 7, 2)
Predicted:  1 Actual:  0 Data:  (0, 18, 199, 0, 1)
Predicted:  1 Actual:  1 Data:  (1, 25, 1682, 7, 4)
Predicted:  0 Actual:  0 Data:  (0, 58, 427, 0, 4)
Predicted:  1 Actual:  1 Data:  (0, 3, 993, 0, 0)
Predicted:  1 Actual:  0 Data:  (0, 48, 356, 1, 1)
Predicted:  0 Actual:  1 Data:  (0, 33, 460, 0, 4)
Predicted:  1 Actual:  0 Data:  (1, 13, 1031, 6, 5)
Predicted:  1 Actual:  0 Data:  (0, 73, 1659, 1, 4)
Predicted:  0 Actual:  0 Data:  (0, 41, 379, 8, 1)
Predicted:  0 Actual:  0 Data:  (0, 18, 190, 8, 1)
Predicted:  1 Actual:  1 Data:  (0, 31, 32, 1, 1)
Predicted:  0 Actual:  1 Data:  (0, 7, 1580, 3, 3)
Predicted:  1 Actual:  1 Data:  (0, 30, 652, 1, 3)
Predicted:  0 Actual:  1 Data:  (1, 16, 418, 6, 1)
Predicted:  0 Actual:  1 Data:  (0, 32, 37, 1, 0)
```

Stage 2 & 3: Algorithm Implementation and Deployment Stage

1. The implementation of the highest performing algorithm as a desktop software tool using the Tkinter package

2. The tool was deployed as a web or cloud-enabled platform tool.

After identifying the best-performing algorithm and machine learning model for predicting potential valued customers in stage 1, the algorithm was implemented as a desktop software tool using the Python Tkinter package and has been Deployed to Github.

The Pycharm project for the implementation can be accessed via this GitHub link:

https://drive.google.com/drive/folders/1b2LyirWXzKWBCBn6m0eyTz_VNjI_Bwy4?usp=share_link

Conclusions

This report showcases the progress made on the ST1 capstone project, which focuses on designing, developing, implementing, and deploying a Python-based data-driven software tool for predicting shop customers. The objective of this tool is to utilize the data gathered from shop members and create a customer classifier platform driven by data analysis. This platform aims to give shop owners a better understanding of their customers and bring numerous benefits to the community.

The project outcomes indicate the need for additional data to train further and enhance the accuracy of the predictive model and improve the platform's capabilities. The tool enables business owners to make more informed decisions by leveraging data analysis and prediction. It is expected that the tool will empower a greater number of business owners to utilize data-driven insights for better decision-making.

References

1. Open Database, Shop Customer Data, [Online]. Available: https://www.kaggle.com/datasets/datascientistanna/customers-dataset

   (accessed: May 2, 2023)