Configuration Manual

MSc Research Project
Data Analytics

VINITH KUMAR GUDIBANDA PRASANNAKUMAR
Student ID: 22131248

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of

Ireland MSc Project

Submission Sheet

School of Computing

Student Name: …….…VINITH KUMAR GUDIBANDA PRASANNAKUMAR…………………….

Student ID: ………………………………….….22131248………………………..……………………..

Programme: …….…Data Analytics……………………………… **Year:** ………2023……….

Module: …………Research Project…………………………..………………………………….……

Lecturer: …………Furqan Rustam…………………………………………………………………….……

Submission Due Date: ………………31/01/2024……………………………………………………………….………

Project Title: Enhancing Purchase Predictions with Machine Learning: Customer Propensity Modelling through Predictive Analytics

Word Count: ……………1669……… **Page Count:** ………..21……………….…….….……….…….

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………**VINITH KUMAR GUDIBANDA PRASANNAKUMAR**…………….

Date: …….……………31/01/2024…………………………………………….……………….

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placedinto the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

1. Introduction

This manual illustrates how to execute and configure the implementation code for the current research project. This document provides specified details about the machine hardware as well as the programs to run. Following the below steps will enable the users to generate summaries of the research papers using the Logistic Regression, Decision Tree, GBM and RNN models.

2. Dataset Description

Sl.No	Attribute Name	Data Type	Attribute Description
1	Administrative	int64	This is the number of pages of this type (administrative) that the user visited
2	Administrative_Duration	float64	This is the amount of time spent in this category of pages.
3	Informational	int64	This is the number of pages of this type (informational) that the user visited.
4	Informational_Duration	float64	This is the amount of time spent in this category of pages.
5	ProductRelated	int64	This is the number of pages of this type (product related) that the user visited.
6	ProductRelated_Duration	float64	This is the amount of time spent in this category of pages.
7	BounceRates	float64	The percentage of visitors who enter the website through that page and exit without triggering any additional tasks.
8	ExitRates	float64	The percentage of pageviews on the website that end at that specific page.
9	PageValues	float64	The average value of the page averaged over the value of the target page and/or the completion of an eCommerce
10	SpecialDay	float64	This value represents the closeness of the browsing date to special days or holidays (eg Mother's Day or Valentine's day).
11	Month	object	Contains the month the pageview occurred, in string form.

12	OperatingSystems	int64	An integer value representing the operating system that the user was on when viewing the page.
13	Browser	int64	An integer value representing the browser that the user was using to view the page.
14	Region	int64	An integer value representing which region the user is located in.
15	TrafficType	int64	An integer value representing what type of traffic the user is categorized into.
16	VisitorType	object	A string representing whether a visitor is New Visitor, Returning Visitor, or Other.
17	Weekend	bool	A boolean representing whether the session is on a weekend.
18	Revenue	bool	A boolean representing whether or not the user completed the purchase.

Table 1: Description of Dataset

3. System Specification

3.1 Hardware Specification

Following are the hardware specifications of the system that was used to develop the project:

Processor: Apple M1 Chip
RAM: 16GB
Storage: 256GB
Graphics Card: 8-core GPU
Operating System: macOS Sonoma

3.2 Software Specification

The Google Colab a web-based platform was used to train and evaluate the models and its specification was the following:

Processor: Intel Xeon
Graphics Card: A100 40GB
RAM: 80GB
Storage: 160GB

4. Software Tools

Following are the software tools that were used to implement the project:

4.1 Python

Python was chosen for its useful libraries in visualization, dataset preparation, and deep learning models. It was downloaded from the official website.

Fig. 1: Python Program Language

4.2 Google Colab

Google Collaboratory, also known as Google Colab, is a cloud-based platform that offers an interactive environment for programming in Python. It's built on the Jupyter Notebook framework, which allows users to write and execute Python code through their web browsers. One of the best things about Colab is that it requires no setup and is incredibly user-friendly, making it accessible to users of all skill levels.

Colab provides free access to essential computing resources such as GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units). These resources are incredibly useful for intensive computational tasks, making Colab an ideal platform for machine learning, data analysis, and educational purposes. These fields often require significant computational power, and Colab makes it easy to access these resources without any extra hassle.

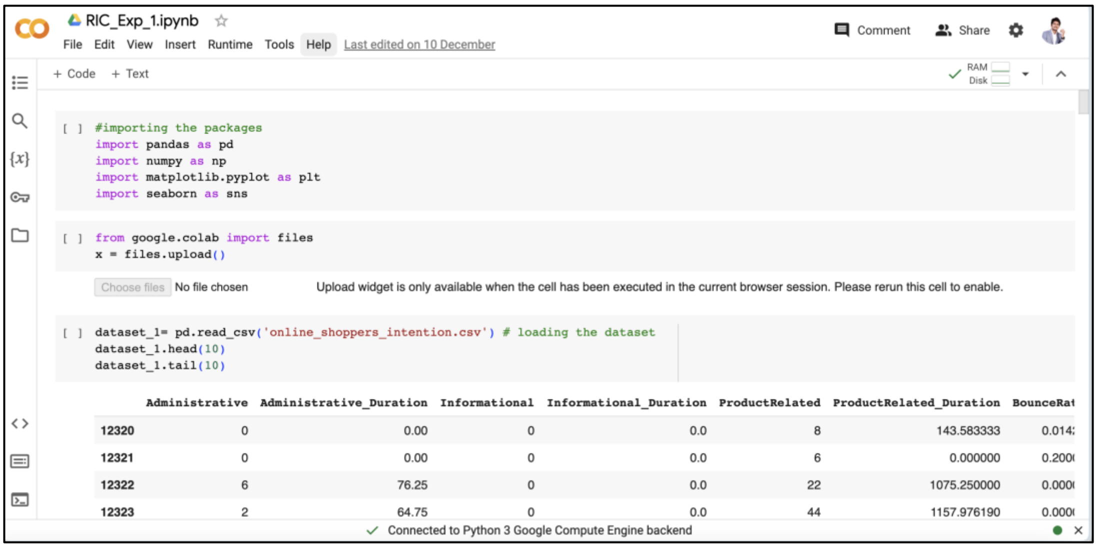


Figure 2: Google Colab Notebook

5. Project Implementation

The following Python packages were installed using pip and used to implement the project:

- Pandas
- Numpy
- Matplotlib

- Seaborn
- Plotly
- Sklearn
- Tensorflow
- Statsmodels
- Imblearn
- Ctgan

These packages were chosen for their usefulness in data analysis, dataset preparation, and deep learning models. These packages were readily available in Colab without the need for installation. Only CT-GAN was explicitly installed for implementation purposes. Fig 3 shows all the library packages used in the code.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense, Dropout

from sklearn.metrics import accuracy_score
import sklearn.metrics as metrics
from sklearn.metrics import classification_report
```

```
#feature selection using stats Model
import statsmodels.api as sm

#Oversampling using SMOTE
from imblearn.over_sampling import SMOTE

!pip install ctgan
from ctgan import CTGAN

#Recursive Feature Elimination
from sklearn.datasets import make_classification
from sklearn.feature_selection import RFECV
from sklearn.neural_network import MLPClassifier
```

Fig. 3: Necessary Libraries and Packages

The panda's library is utilized for loading and analysing datasets shown in Fig. 4.

```
dataset_1= pd.read_csv('online_shoppers_intention.csv') # loading the dataset
dataset_1.head(10)
dataset_1.tail(10)
```

Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0.00	0	0.0	8	143.583333	0.014286	0.050000	0.000000
0	0.00	0	0.0	6	0.000000	0.200000	0.200000	0.000000
6	76.25	0	0.0	22	1075.250000	0.000000	0.004167	0.000000
2	64.75	0	0.0	44	1157.976190	0.000000	0.013953	0.000000
0	0.00	1	0.0	16	503.000000	0.000000	0.037647	0.000000
3	145.00	0	0.0	53	1783.791667	0.007143	0.029031	12.241717
0	0.00	0	0.0	5	465.750000	0.000000	0.021333	0.000000
0	0.00	0	0.0	6	184.250000	0.083333	0.086667	0.000000
4	75.00	0	0.0	15	346.000000	0.000000	0.021053	0.000000
0	0.00	0	0.0	3	21.250000	0.000000	0.066667	0.000000

Fig. 4: Viewing the dataset using pandas

5.1 Preparing of Data

1. Looking for missing or null values in Fig. 5
2. Examining the data in the columns in Fig. 5
3. The basic statistics of the numeric column Fig. 6

```
[ ] print(dataset_1.columns[dataset_1.isna().any()])

    Index([], dtype='object')

dataset_1.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

Fig 5: Looking for Null Value and Information of Data

```python
dataset_1.describe()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration
count	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000	12330.000000
mean	2.315166	80.818611	0.503569	34.472398	31.731468	1194.746220
std	3.321784	176.779107	1.270156	140.749294	44.475503	1913.669288
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	7.000000	184.137500
50%	1.000000	7.500000	0.000000	0.000000	18.000000	598.936905
75%	4.000000	93.256250	0.000000	0.000000	38.000000	1464.157214
max	27.000000	3398.750000	24.000000	2549.375000	705.000000	63973.522230

Fig 6: Statistical Description

5.2 Data Pre-Processing

It's important to note that label encoding is a useful technique in data pre-processing that converts categorical variables into a numerical format. Similarly, Boolean variables can be transformed into binary numeric formats. Fig. 7 and Fig. 8 shows the pre-processing of data.

```python
from sklearn import preprocessing
number = preprocessing.LabelEncoder()
for i in categorical_values:
    dataset_1[i] = number.fit_transform(dataset_1[i])

dataset_1.tail(5)
```

Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
3	145.0	0	0.0	53	1783.791667	0.007143	0.029031	12.241717
0	0.0	0	0.0	5	465.750000	0.000000	0.021333	0.000000
0	0.0	0	0.0	6	184.250000	0.083333	0.086667	0.000000
4	75.0	0	0.0	15	346.000000	0.000000	0.021053	0.000000
0	0.0	0	0.0	3	21.250000	0.000000	0.066667	0.000000

Fig. 7: Data pre-processing using sklearn

```python
for i in bool_values:
    dataset_1[i] = number.fit_transform(dataset_1[i])

dataset_1.tail(5)
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRat
12325	3	145.0	0	0.0	53	1783.791667	0.0071
12326	0	0.0	0	0.0	5	465.750000	0.0000
12327	0	0.0	0	0.0	6	184.250000	0.0833
12328	4	75.0	0	0.0	15	346.000000	0.0000
12329	0	0.0	0	0.0	3	21.250000	0.0000

Fig. 8: Data pre-processing using sklearn

5.3 Class Balancing

A pie chart in Fig.9 demonstrating the distribution of the classes in the dataset is shown below.

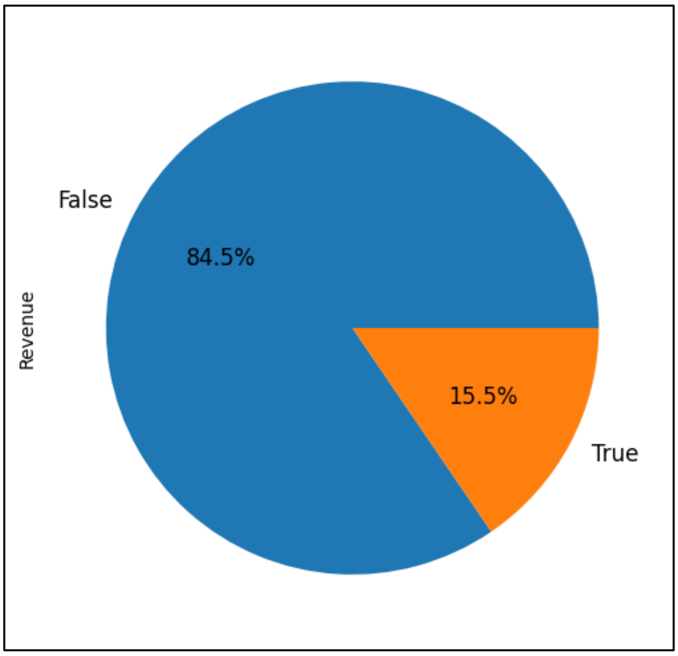


Fig. 9: Original dataset Class distribution

5.3.1 Class Balancing using SMOTE

Fig. 10 implies the data balancing outcome after the implementation of SMOTE

```
[ ] from imblearn.over_sampling import SMOTE
    smote = SMOTE(random_state=42, sampling_strategy=1)
    X_sm, y_sm = smote.fit_resample(X, Y)
    X_sm.head()
```

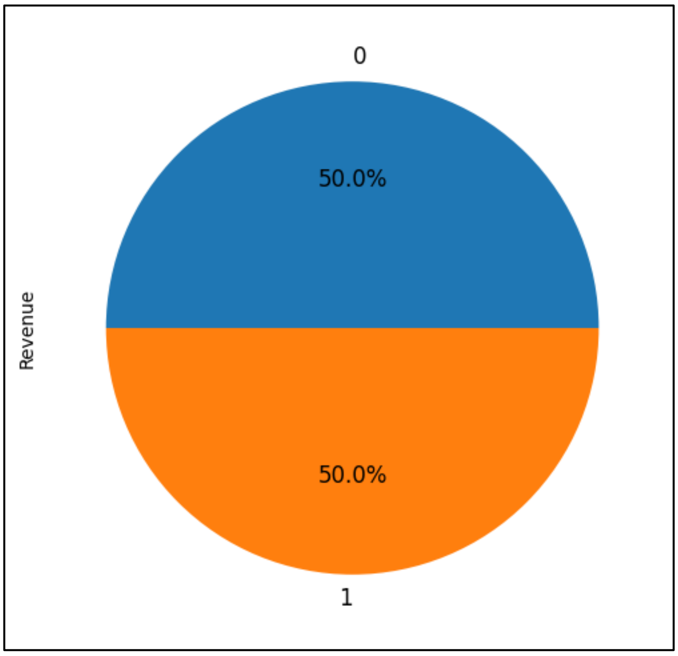


Fig. 10: After Oversampling using SMOTE

5.3.2 Class Balancing using CTGAN

Fig. 11 and Fig.12 implies the data balancing outcome after the implementation of CTGAN

```
[ ] from ctgan import CTGAN

[ ] model = CTGAN(verbose=True)

[ ] model.fit(minority_class)

    Gen. (-3.10) | Discrim. (0.18): 100%|██████████| 300/300 [01:32<00:00,  3.23it/s]

[ ] new_data = model.sample(8514)

[ ] new_data.head()
```

Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	375.323465	0	468.970469	20	910.257271	0.003150	0.049833	25.164783
1	62.267550	4	2.770716	60	1003.491731	0.003054	0.063636	40.842695
0	29.840802	0	97.591839	49	5241.308124	0.001511	0.048557	21.754704
0	276.721462	2	524.863361	76	550.419369	0.001228	0.021374	21.601322
0	20.703616	1	5.564539	82	2780.337751	0.002314	0.046092	13.070126

Fig. 11: Applying CT-GAN

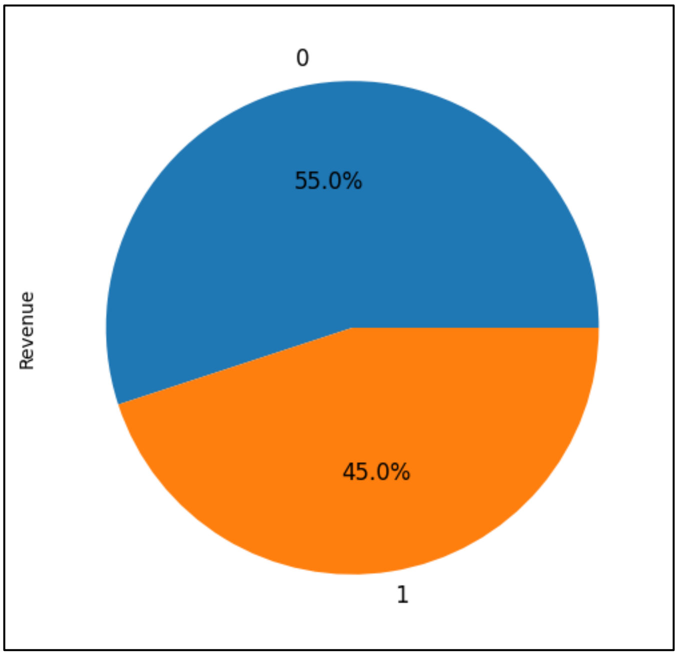


Fig. 12: After Class balancing using CTGAN

Dropping the columns and saving the processed dataset shown in Fig. 13.

```
[ ] X = dataset_1.drop('Revenue',axis='columns')
    #print(X.shape)
    #X
    Y = dataset_1.Revenue
```

Fig. 13: Dropping the target class

The dataset is split into 70% training data and 30% test data for model evaluation shown in Fig. 14.

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, stratify= Y, random_state = 42)

    ## checking for the distribution of traget variable in train test split
    print('Distribution of target variable in training dataset')
    print(Y_train.value_counts())

    print('Distribution of target variable in test dataset')
    print(Y_test.value_counts())
```

Fig. 14: Splitting the dataset into training and testing samples

5.4 Data normalization is carried out using MinMax Scaler shown in Fig. 15

```
[ ] from sklearn.preprocessing import MinMaxScaler
    sc = MinMaxScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```

Fig. 15: Data Normalization using sklearn

I have utilized two feature selection models for various experimental setups.

1. Feature Selection using OLS Stats Model: In this analysis, I have selected only those attributes whose p-values are less than 0.05 shown in Fig. 16

```
#feature selection using stats Model
import statsmodels.api as sm
X = sm.add_constant(X)

lr = sm.OLS(Y,X).fit()
print(lr.summary2())
```

```
                     Results: Ordinary least squares
=======================================================================
Model:                  OLS                 Adj. R-squared:      0.277
Dependent Variable:     Revenue             AIC:                 5923.9338
Date:                   2023-12-10 21:23    BIC:                 6057.4901
No. Observations:       12330               Log-Likelihood:      -2944.0
Df Model:               17                  F-statistic:         279.4
Df Residuals:           12312               Prob (F-statistic):  0.00
R-squared:              0.278               Scale:               0.094525
-----------------------------------------------------------------------
                          Coef.  Std.Err.    t     P>|t|   [0.025  0.975]
-----------------------------------------------------------------------
const                     0.1207  0.0133  9.1025 0.0000  0.0947  0.1467
Administrative            0.0016  0.0012  1.3700 0.1707 -0.0007  0.0039
Administrative_Duration  -0.0000  0.0000 -0.9717 0.3312 -0.0001  0.0000
Informational             0.0033  0.0030  1.1139 0.2654 -0.0025  0.0091
Informational_Duration    0.0000  0.0000  0.5139 0.6073 -0.0000  0.0001
ProductRelated            0.0004  0.0001  3.1727 0.0015  0.0002  0.0007
ProductRelated_Duration   0.0000  0.0000  3.3787 0.0007  0.0000  0.0000
BounceRates               0.4787  0.1446  3.3114 0.0009  0.1953  0.7620
ExitRates                -1.0096  0.1523 -6.6302 0.0000 -1.3080 -0.7111
PageValues                0.0090  0.0002 58.4897 0.0000  0.0087  0.0093
SpecialDay               -0.0726  0.0142 -5.1287 0.0000 -0.1004 -0.0449
Month                     0.0095  0.0012  8.0549 0.0000  0.0072  0.0118
OperatingSystems         -0.0091  0.0032 -2.8861 0.0039 -0.0154 -0.0029
Browser                   0.0029  0.0017  1.7433 0.0813 -0.0004  0.0062
Region                   -0.0020  0.0012 -1.7423 0.0815 -0.0043  0.0003
TrafficType               0.0002  0.0007  0.3332 0.7390 -0.0012  0.0016
VisitorType              -0.0259  0.0042 -6.1614 0.0000 -0.0341 -0.0177
Weekend                   0.0104  0.0066  1.5855 0.1129 -0.0025  0.0233
---------------------------------------------------------------------
Omnibus:                3213.137        Durbin-Watson:          1.993
Prob(Omnibus):          0.000           Jarque-Bera (JB):       10794.239
Skew:                   1.305           Prob(JB):               0.000
Kurtosis:               6.768           Condition No.:          167416
=====================================================================
```

```
[ ] p_values = lr.pvalues
    vars = p_values[p_values<=0.05].index.tolist()
    print(vars)
    print(len(vars))

    ['const', 'ProductRelated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay', 'Month', 'OperatingSyst
    10
```

Fig. 16: Feature importance using OLS STATS Model

2. Recursive Feature Elimination with Cross-Validation: Using feature ranking with recursive feature elimination and cross-validated selection to determine the optimal number of features shown in Fig. 17

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Create a neural network classifier
clf = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=1000, random_state=42)
# Wrap the neural network model in a Logistic Regression model for feature ranking
wrapped_clf = LogisticRegression(max_iter=5000)
# Create RFECV model
rfecv = RFECV(estimator=wrapped_clf, step=1, cv=5)  # 5-fold cross-validation
# Fit RFECV
rfecv.fit(X_train, y_train)
# Transform the data to keep only selected features
X_train_selected = rfecv.transform(X_train)
X_test_selected = rfecv.transform(X_test)

# Train the model with selected features
clf.fit(X_train_selected, y_train)
# Make predictions on the test set
y_pred = clf.predict(X_test_selected)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Test Accuracy: {accuracy}')

# Print the selected features
print("Selected Features:", np.where(rfecv.support_)[0])
```

Fig. 17: RFE using Cross-Validation

6. Model Building and Evaluation

Algorithms: The algorithm used in the experiment is outlined below.

1. Decision Tree
2. Logistic Regression
3. Gradient Boosting Machine
4. Recurrent Neural Network

Experiment (EXP) – 1

Experiment 1 was carried out on the original dataset, without using any feature selection or solving the class imbalance issue.

Experiment (EXP) – 2

Experiment 2 was carried out on the dataset, using OLS feature selection and SMOTE to solve for class imbalance issue.

Experiment (EXP) – 3

Experiment 3 was carried out on the dataset, using OLS feature selection and CT-GAN for the class imbalance issue.

Experiment (EXP) – 4

Experiment 4 was carried out on the dataset, using RFECV feature selection and CT-GAN for the class imbalance issue.

Hyperparameters Calculation:

Algorithm 1 – Decision Tree

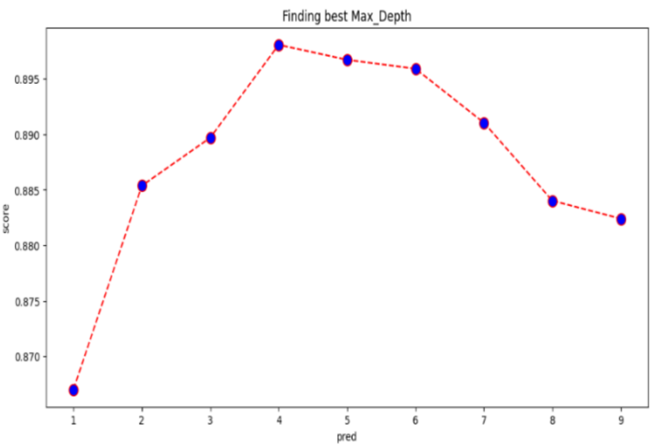
Hyperparameter calculation is only performed for the decision tree model, all other models use using default hyperparameter setting. Finding the best max_depth for the decision tree is shown in Fig. 18:

```
#Finding best max_depth Value

accuracy = []
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

for i in range(1, 10):
    model = DecisionTreeClassifier(max_depth = i, random_state = 0)
    model.fit(X_train, Y_train)
    pred = model.predict(X_test)
    score = accuracy_score(Y_test, pred)
    accuracy.append(score)

plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Finding best Max_Depth')
plt.xlabel('pred')
plt.ylabel('score')
plt.show()
```

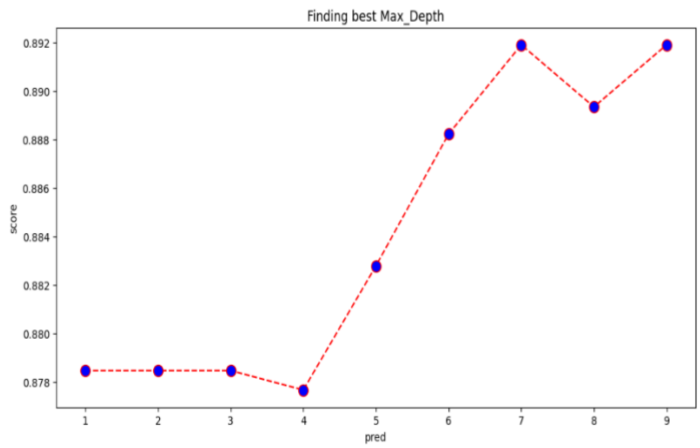
Fig. 18: Finding the max_depth

Results of calculated max_depth for each experiment shown in Fig. 19

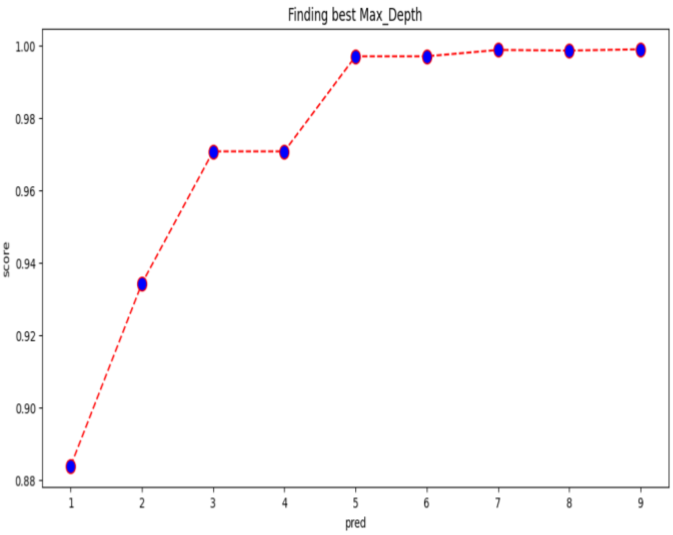
EXP-1:



EXP-2:



EXP-3:



EXP-4

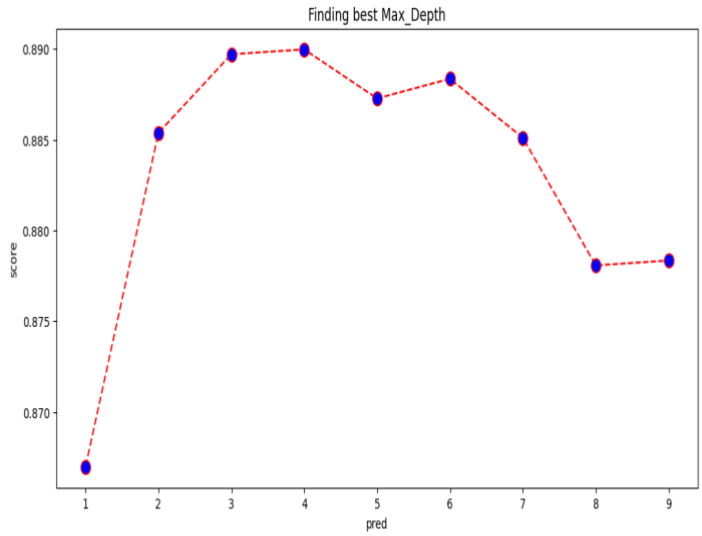


Fig. 19: Max-Depth graph for various dataset

6.1 Implementation of Decision Tree Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import sklearn.metrics as metrics

# create an instance of the DecisionTreeClassifier class
clf = DecisionTreeClassifier(max_depth = 4,random_state = 42)
clf.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=42)
```

Fig. 20: Exp-1 Hyperparameters and Model Building

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import sklearn.metrics as metrics

# create an instance of the DecisionTreeClassifier class
clf = DecisionTreeClassifier(max_depth = 7,random_state = 42)
clf.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, random_state=42)
```

Fig. 21: Exp-2 Hyperparameters and Model Building.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import sklearn.metrics as metrics

# create an instance of the DecisionTreeClassifier class
clf = DecisionTreeClassifier(max_depth = 7,random_state = 42)
clf.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, random_state=42)
```

Fig. 22: Exp-3 Hyperparameters and Model Building.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
import sklearn.metrics as metrics

# create an instance of the DecisionTreeClassifier class
clf = DecisionTreeClassifier(max_depth = 4,random_state = 42)
clf.fit(X_train,Y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=4, random_state=42)
```

Fig. 23: Exp-4 Hyperparameters and Model Building.

6.2 Implementation of Logistic Regression

Fig. 24 shows Logistic Regression model process.

```
[ ] from sklearn.linear_model import LogisticRegression
    logmodel = LogisticRegression(random_state = 42)
    logmodel.fit(X_train,Y_train)

    ▾ LogisticRegression
    LogisticRegression(random_state=42)

[ ] predictions_LR = logmodel.predict(X_test)
    predictions_LR

    final_model_predictions_LR = pd.DataFrame({'Actual':Y_test, 'predictions':predictions_LR})

[ ] from sklearn.metrics import classification_report
    print(classification_report(Y_test, predictions_LR))

                  precision    recall  f1-score   support

               0       0.88      0.99      0.93      3127
               1       0.77      0.25      0.38       572

        accuracy                           0.87      3699
       macro avg       0.82      0.62      0.66      3699
    weighted avg       0.86      0.87      0.84      3699
```

Fig. 24: Logistic Regression Model.

6.3 Implementation of GBM

Fig. 25 shows GBM model process.

```
[ ] from sklearn.ensemble import GradientBoostingClassifier
    gbc = GradientBoostingClassifier(n_estimators=300,learning_rate=0.05,random_state=42,max_features=6 )

    # Fit to training set
    gbc.fit(X_train,Y_train)

    ▾ GradientBoostingClassifier
    GradientBoostingClassifier(learning_rate=0.05, max_features=6, n_estimators=300,
                               random_state=42)

[ ] y_pred_GBC = gbc.predict(X_test)

    final_model_predictions_GBC = pd.DataFrame({'Actual':Y_test, 'predictions':y_pred_GBC})

    accuracy_GBC=np.round(metrics.accuracy_score( Y_test, y_pred_GBC ),2 )*100
    accuracy_GBC='{:.2f}'.format(accuracy_GBC)
    print( 'Total Accuracy : ',accuracy_GBC)

    Total Accuracy :  90.00
```

Fig. 25: Gradient Boosting Model.

6.4 Implementation of RNN

Fig. 26 shows RNN model process.

```
# Define your model
model = Sequential()
# Add the first RNN layer with a specified number of units and input shape
model.add(SimpleRNN(units=64, activation='relu', return_sequences=True, input_shape=(X.shape[1], 1)))
# Add the first dropout layer to prevent overfitting
model.add(Dropout(0.5))
# Add the second RNN layer
model.add(SimpleRNN(units=64, activation='relu', return_sequences=True))
# Add the second dropout layer
model.add(Dropout(0.5))
# Add the third RNN layer
model.add(SimpleRNN(units=64, activation='relu'))
# Add the third dropout layer
model.add(Dropout(0.5))
# Add the output dense layer for classification or regression
model.add(Dense(units=1, activation='sigmoid'))  # For binary classification, change activation function accordingly
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])  # Adjust the loss function as needed
# Summary of the model architecture
model.summary()

# Reshape the input data to match RNN input shape
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
```

Fig. 26: RNN Model

Fig. 27 shows RNN model training for 25 epochs

```
[ ] # Train the model
    epochs = 25
    batch_size = 16
    H = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, Y_test))

    # Evaluate the model
    loss, accuracy = model.evaluate(X_test, Y_test)
    print(f'Accuracy: {accuracy * 100:.2f}%')

Epoch 1/25
540/540 [==============================] - 18s 27ms/step - loss: 0.4403 - accuracy: 0.8439 - val_loss: 0.4028 - val_accuracy: 0.8454
Epoch 2/25
540/540 [==============================] - 10s 18ms/step - loss: 0.3460 - accuracy: 0.8686 - val_loss: 0.3616 - val_accuracy: 0.8724
Epoch 3/25
540/540 [==============================] - 10s 18ms/step - loss: 0.3012 - accuracy: 0.8815 - val_loss: 0.3026 - val_accuracy: 0.8851
Epoch 4/25
540/540 [==============================] - 8s 15ms/step - loss: 0.2918 - accuracy: 0.8865 - val_loss: 0.2908 - val_accuracy: 0.8867
Epoch 5/25
540/540 [==============================] - 10s 18ms/step - loss: 0.2812 - accuracy: 0.8902 - val_loss: 0.2765 - val_accuracy: 0.8927
Epoch 6/25
540/540 [==============================] - 10s 18ms/step - loss: 0.2769 - accuracy: 0.8965 - val_loss: 0.2821 - val_accuracy: 0.8913
Epoch 7/25
540/540 [==============================] - 8s 16ms/step - loss: 0.2805 - accuracy: 0.8919 - val_loss: 0.2884 - val_accuracy: 0.8938
Epoch 8/25
540/540 [==============================] - 10s 18ms/step - loss: 0.2732 - accuracy: 0.8903 - val_loss: 0.2738 - val_accuracy: 0.8927
Epoch 9/25
540/540 [==============================] - 10s 18ms/step - loss: 0.2714 - accuracy: 0.8926 - val_loss: 0.2765 - val_accuracy: 0.8883
```

Fig. 27: Training of RNN Model

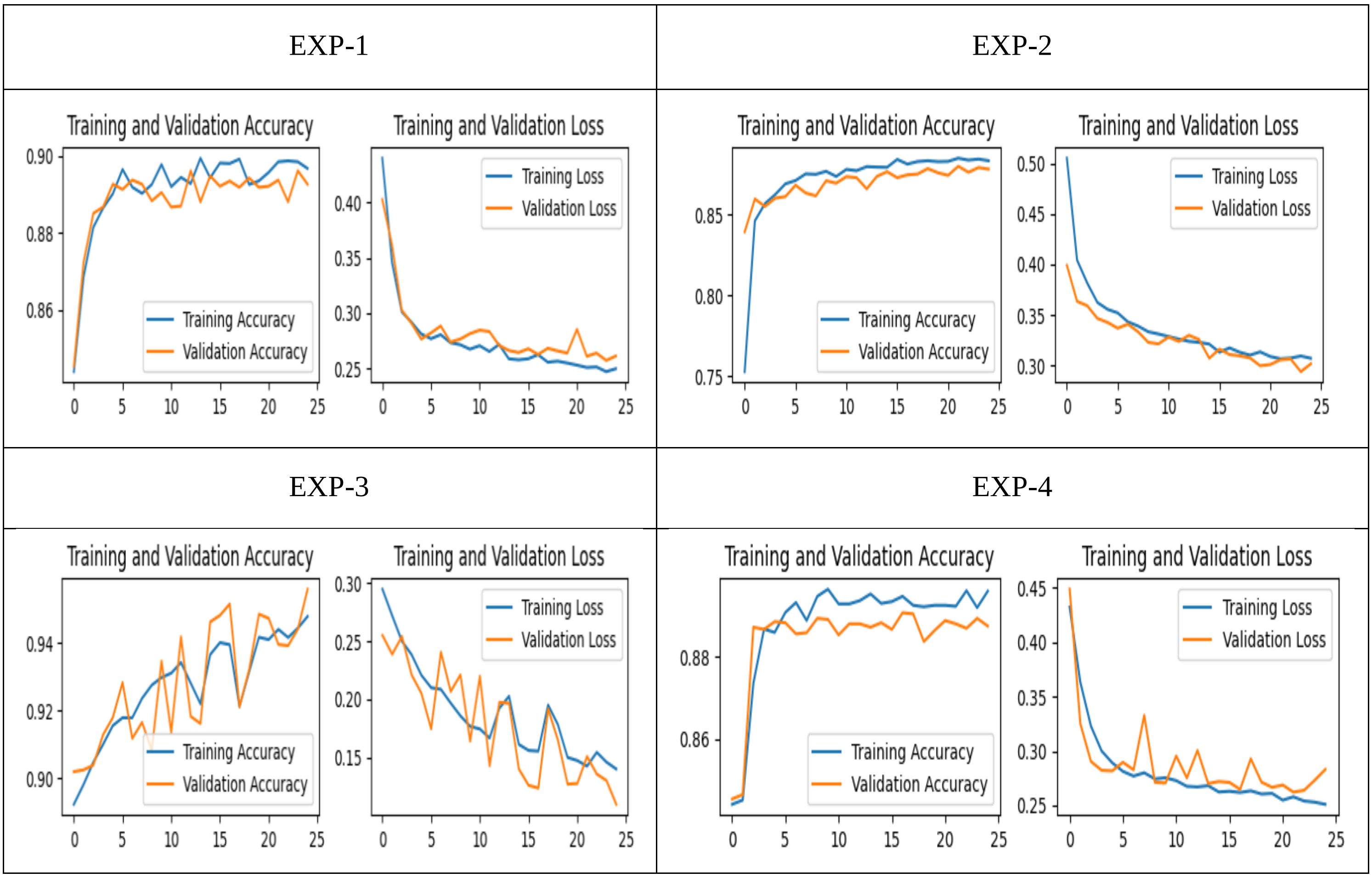


Fig. 28: Accuracy and Loss graphs of RNN

6.5. Performing K-fold validation for GBM in Experiment-3

```
# Perform 10-fold cross-validation
scores = cross_val_score(gbc, X, Y, cv=10)

print("Accuracy scores for each fold:", scores)
print("Average accuracy:", scores.mean())

Accuracy scores for each fold: [1.         1.         1.         1.         1.         0.99947202
 1.         0.99947174 1.         0.99947174]
Average accuracy: 0.9998415492859538
```

Fig. 29: K-fold average accuracy for GBM in Experiment-3

7. Evaluation Metrics Graphs for all the experiment models

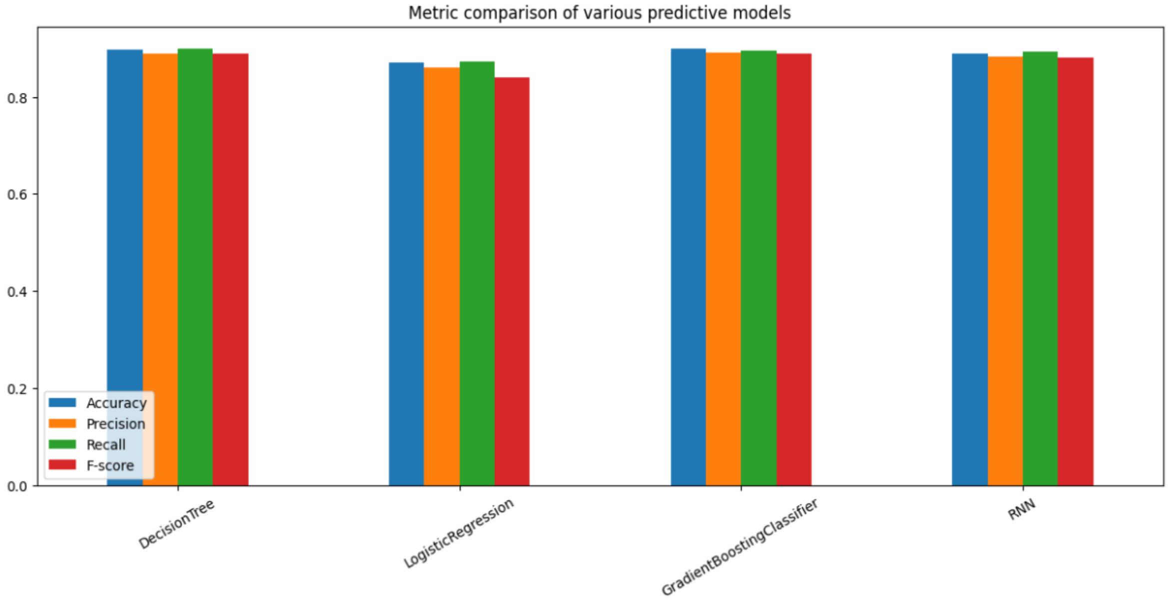


Fig. 30: Evaluation Result of EXP-1

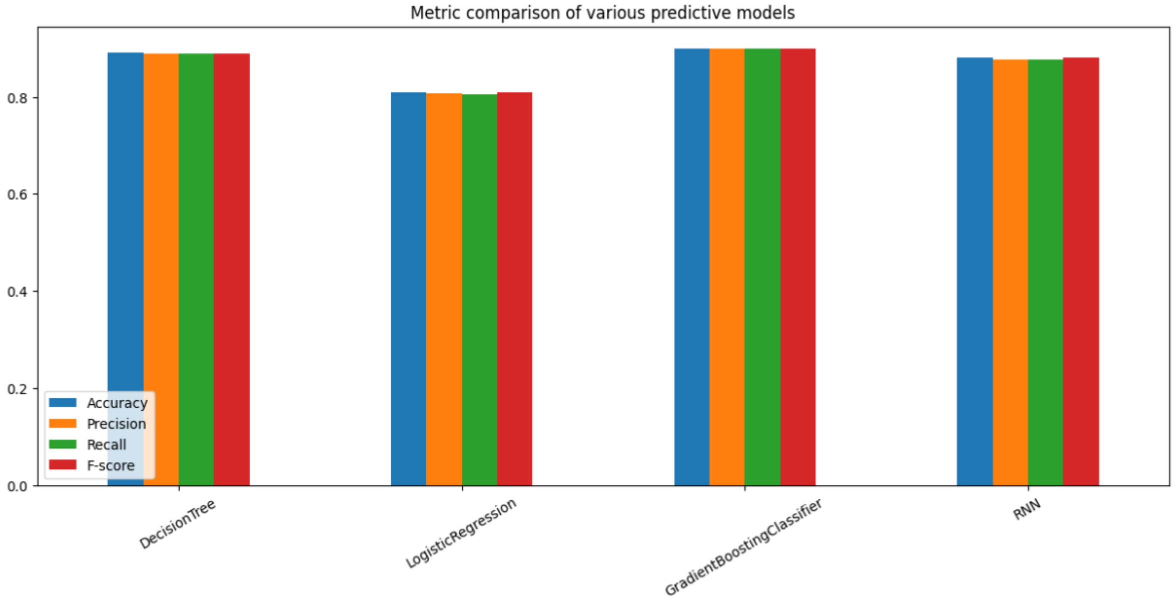


Fig. 31: Evaluation Result of EXP-2

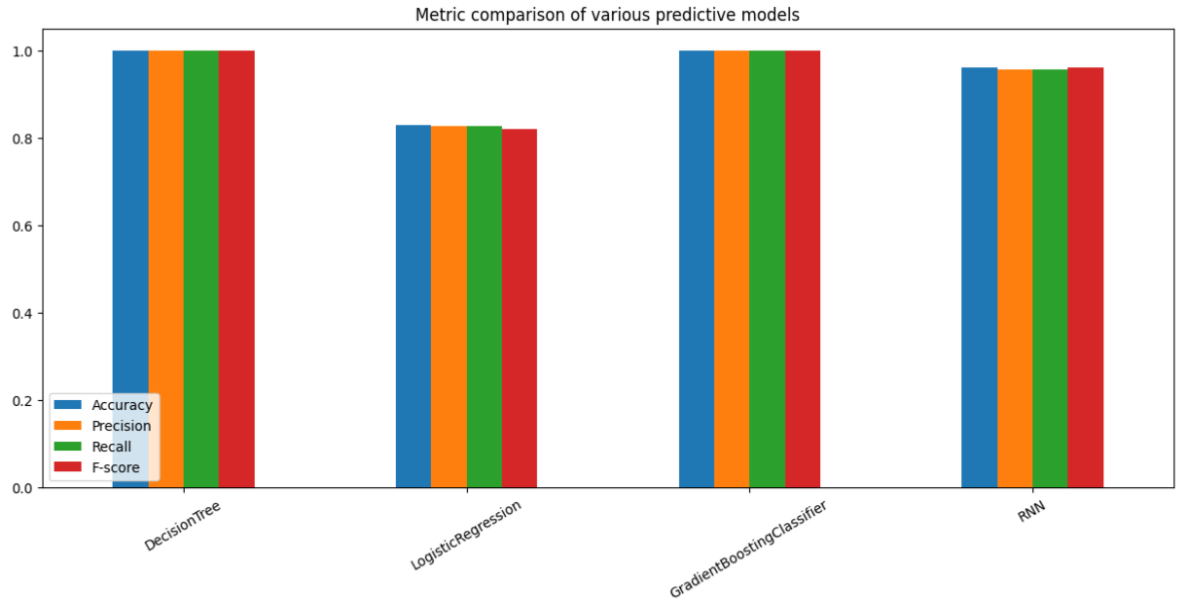


Fig. 32: Evaluation Result of EXP-3

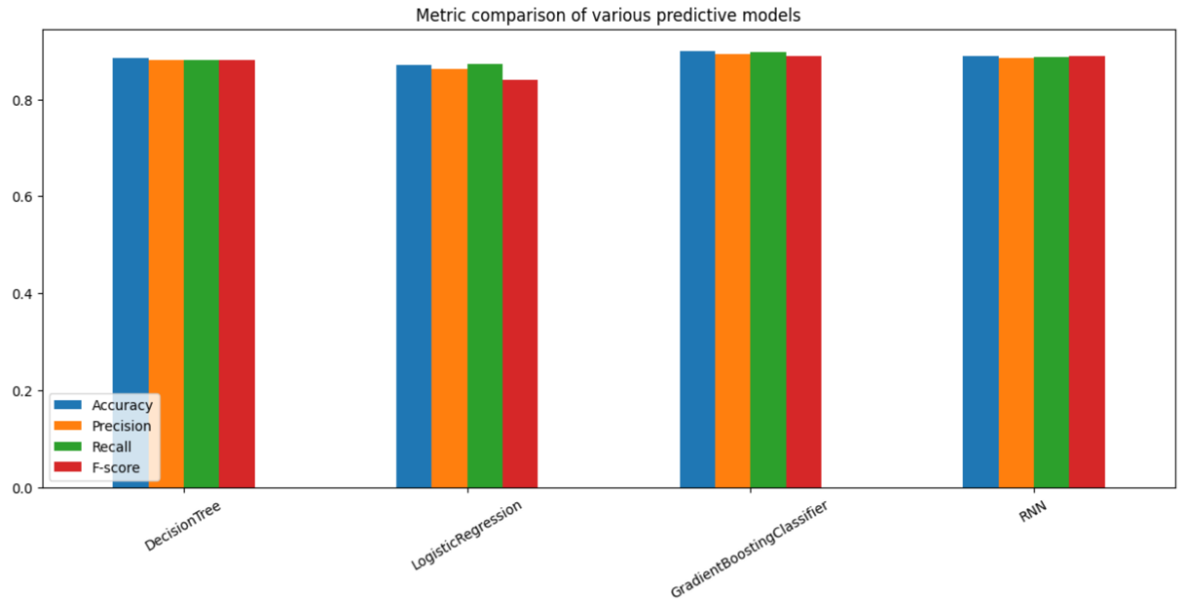


Fig. 33: Evaluation Result of EXP-4