National College of Ireland

# Configuration Manual

MSc Research Project

Data Analytics


Piyush Ingle

Student ID: 22154779

School of Computing

National College of Ireland


Supervisor: Furqan Rustam

**National College of**

**Ireland MSc Project**

**Submission Sheet**

**School of Computing**

**Student Name:** ..... Piyush Ingle.....

**Student ID:** .....x22154779.....

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

**Module:** .....Research Project.....

**Supervisor:** .....Furqan Rustam.....

**Submission Due Date:** .....14/12/2023.....

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

**Word Count:** .....1340..... **Page Count**.....14.....

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:** .....**Piyush Ingle**.....

**Date:** .....14/12/2023.....

**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 placed into the assignment box located outside the office.

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

# Configuration Manual

Piyush Ingle

Student Id: 22154779

## 1. Introduction

This handbook provides instructions on how to run and set up the implementation code for the ongoing study. This paper offers specific information about the hardware of the computer as well as the applications that need to be used. By following the procedures listed below, users will be able to employ machine learning and deep learning algorithms along with sampling approaches to diagnose diabetes.

## 2. System Specifications:

### 2.1 Hardware Specification

Below are the hardware specifications on which the experiments are performed.
Processor: 12th Gen Intel(R) Core(TM) i7-12700H  2.30 GHz
RAM: 16.0 GB
Storage: 500 GB
Operating System: Windows 11
Graphics card: GeForce RTX 3060

## 3. Software tools

Below are the software that are used to build this project.

### 3.1 Python

Python is a popular high-level interpreted programming language that is easy to read and understand. Python, which was developed by Guido van Rossum and initially published in 1991, is a flexible language that is frequently used for web development, data analysis, artificial intelligence, and automation. It places an emphasis on readability and usability of code. Python is a popular choice for developers of all skill levels due to its big standard library and robust community support.

### 3.2 Anaconda Navigator:

Anaconda software helps to create an environment for many different versions of python and Package versions. Anaconda can also be used to install, remove and upgrade packages in your project environment. Anaconda is an open source and can be downloaded from https://www.anaconda.com/.

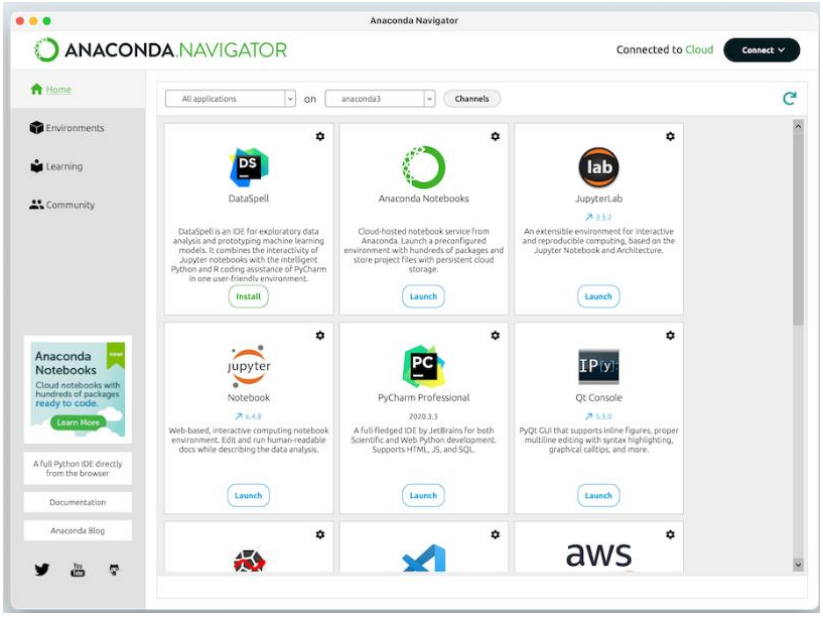


Fig 1. Anaconda Navigator Home page

### 3.3 Jupyter Notebook:

It is a web based, interactive computing notebook environment. It is used to Edit and Run human-readable docs while describing the data analysis.


Fig 2. Home page of Jupyter Notebook

## 4. Project Implementation

Python libraries that are installed and necessary for the project are as follows:

- Numpy
- Pandas
- Matplotlib
- Keras
- Tensorflow
- Seaborn
- SciPy

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
from pandas.plotting import scatter_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve


from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import r2_score
from sklearn.feature_selection import SequentialFeatureSelector
from sklearn import preprocessing
from sklearn.model_selection import cross_validate
from sklearn.metrics import explained_variance_score
from sklearn.metrics import mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier



sns.set_style('white')
sns.set_palette('Set2')
```

Fig 3. Important Libraries and Packages.

- In the below image we can see the file path is given from where the data is taken and read command is used to read the data.

```
path = 'D:/Thesis/diabetes_health/'
```

```
data = pd.read_csv(path + 'diabetes_012.csv', encoding = "ISO-8859-1")
data.head()
```

| | Diabetes_012 | HighBP | HighChol | CholCheck | BMI | Smoker | Stroke | HeartDiseaseorAttack | PhysA |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 1 | 1 | 40 | 1 | 0 | 0 | |
| **1** | 0 | 0 | 0 | 0 | 25 | 1 | 0 | 0 | |
| **2** | 0 | 1 | 1 | 1 | 28 | 0 | 0 | 0 | |
| **3** | 0 | 1 | 0 | 1 | 27 | 0 | 0 | 0 | |
| **4** | 0 | 1 | 1 | 1 | 24 | 0 | 0 | 0 | |

5 rows × 22 columns

Fig 4. File Path and Data head display.

- Below is the image showing code to display unique values in each column.

```
for col in df.columns:
    print(f"{col}: {np.sort(df[col].unique())}")
```

Fig 5. Code to display unique values from dataset.

- In the below image, code to display data types of all columns is shown.

```
df.dtypes
```

```
Diabetes                int64
HighBP                  int64
HighChol                int64
CholCheck               int64
BMI                     int64
Smoker                  int64
Stroke                  int64
HeartDiseaseorAttack    int64
```

Fig. 6 code to display data types.

- Below image shows the code to display the number of missing values in each column.

```
# Check missing values
df.isna().sum()
```

```
Diabetes                0
HighBP                  0
HighChol                0
CholCheck               0
BMI                     0
Smoker                  0
Stroke                  0
```

Fig 7. Displaying missing values.

- The below code shows descriptive statistics of the complete dataset. In which count, mean, standard deviation, and quarterly data is also shown.

```
# Look at the descriptive staistics
df.describe().T
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **Diabetes** | 253680.0 | 0.296921 | 0.698160 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 |
| **HighBP** | 253680.0 | 0.429001 | 0.494934 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |
| **HighChol** | 253680.0 | 0.424121 | 0.494210 | 0.0 | 0.0 | 0.0 | 1.0 | 1.0 |

Fig 8. Descriptive statistics.

- Code showing histogram of High BP and Income columns.

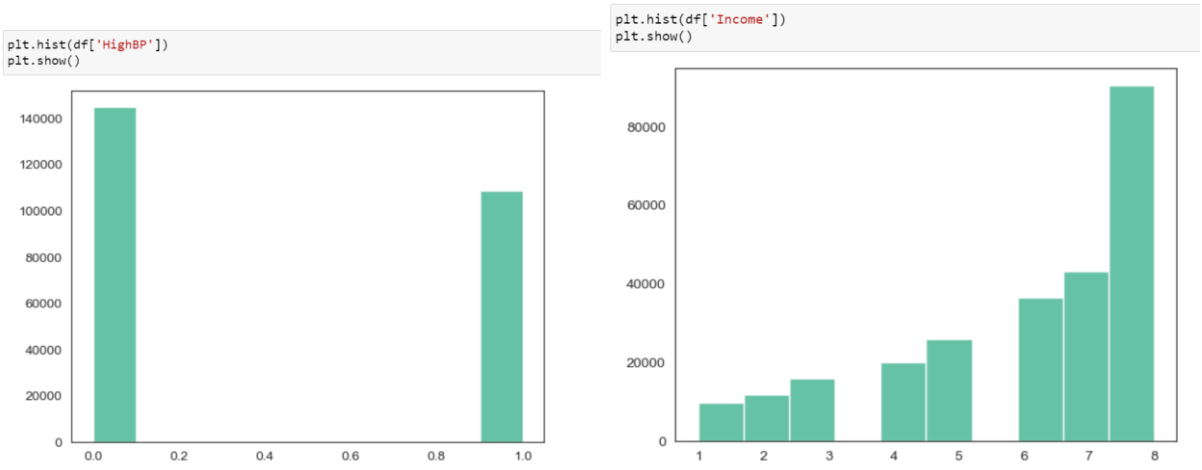


Fig 9. Histogram of High BP and Income.

- The below code shows how to display an sns bar plot of age vs physical health column. In the similar fashion you can displey any columns according to the need.

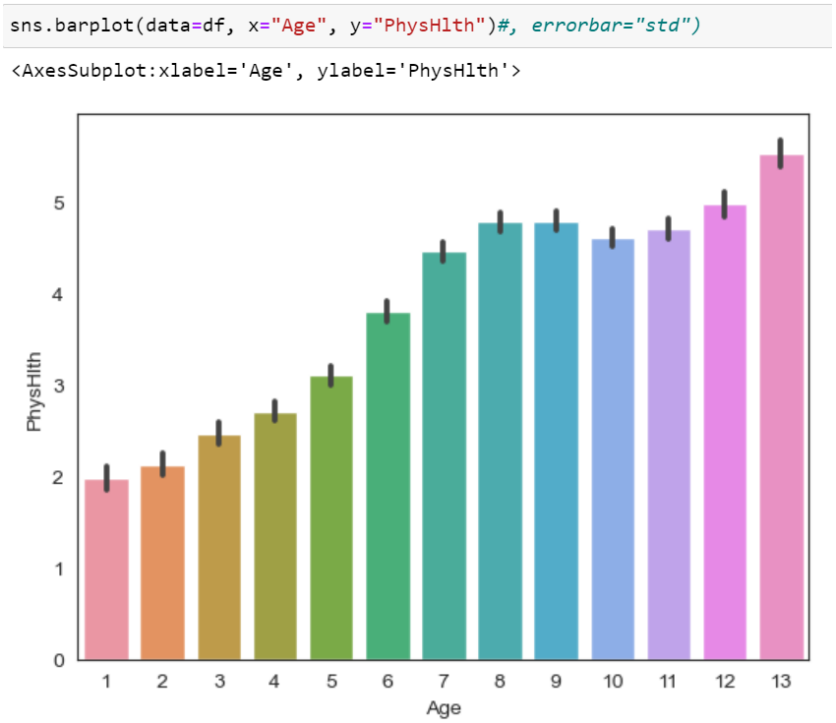


Fig 10. Bar plot Age vs Physical health

- The below image shows code of correlation between Diabetes and other columns in the dataset.

```
df.corr()['Diabetes']
```

```
Diabetes                1.000000
HighBP                  0.271596
HighChol                0.209085
CholCheck               0.067546
BMI                     0.224379
Smoker                  0.062914
Stroke                  0.107179
HeartDiseaseorAttack    0.180272
PhysActivity           -0.121947
Fruits                 -0.042192
Veggies                -0.058972
HvyAlcoholConsump      -0.057882
AnyHealthcare           0.015410
NoDocbcCost             0.035436
GenHlth                 0.302587
MentHlth                0.073507
PhysHlth                0.176287
DiffWalk                0.224239
Sex                     0.031040
Age                     0.185026
Education              -0.130517
Income                 -0.171483
Name: Diabetes, dtype: float64
```

Fig 11. Correlation

- Below image showing heatmap of correlation between all the columns of the dataset.

```
cor_matrix = df.corr()

# Create the heatmap
plt.figure(figsize=(15, 12))
sns.heatmap(cor_matrix, cmap="coolwarm", annot=True)
plt.title("Correlation Heatmap")
plt.show()
```

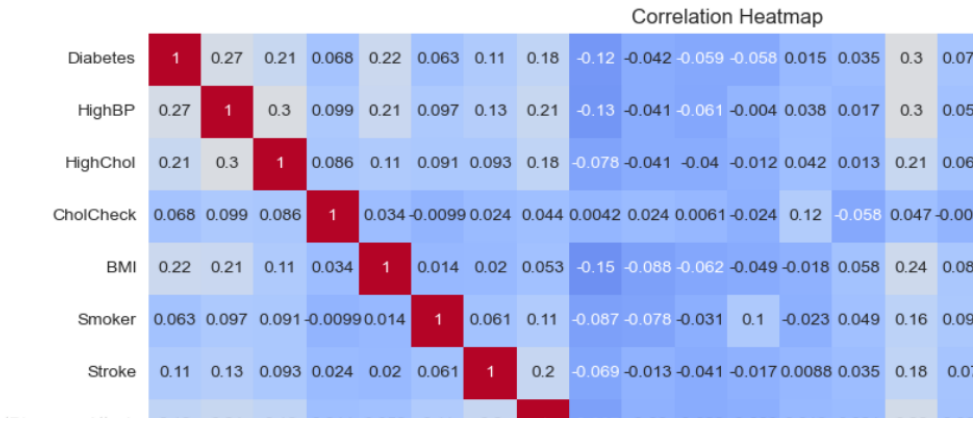


Fig 12. Heatmap pf correlation

- Image showing the code used to plot pie charts of different variables

```
imp_variables = ['Education', 'DiffWalk', 'GenHlth','HighBP']
fig, ax = plt.subplots(2, 2, figsize=(12, 12))
for i, col in enumerate(imp_variables):
    df[col].value_counts().plot.pie(ax=ax[i//2, i%2], autopct='%.2f%%', title=col)
plt.suptitle('Variables affecting the most', fontsize=14)
plt.show()
```

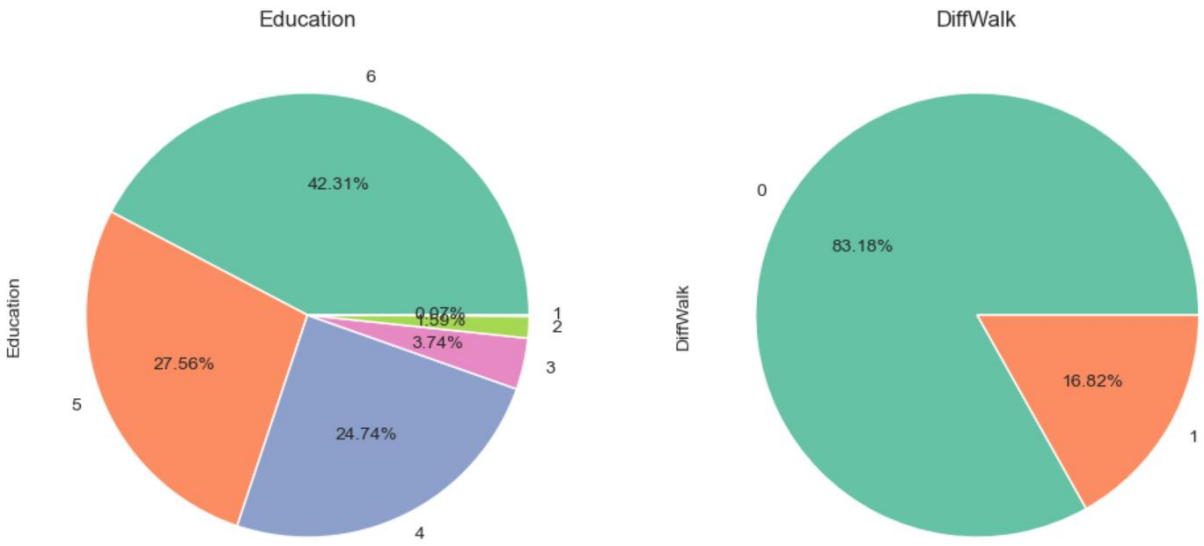


Fig 13. Pie chart of different variables

### 4.1 Data Balancing and Scaling

- The Below image shows code used to divide data into independent and dependent variables.

```
X = df.drop(['Diabetes'], axis=1)
y = df['Diabetes']

print(X.shape, y.shape)
```

```
(253680, 21) (253680,)
```

Fig 14. Data split into independent variables and dependent variable.

- Data balancing using ADASYN

```
adasyn = ADASYN(sampling_strategy='auto', random_state=42)
X, y = adasyn.fit_resample(X, y)
```

Fig 15. Data balancing using ADASYN.

- In the below code it is shown how the data is divided in test and train dataset. Also once the data is balanced you can see the difference that the number of samples has been increased drastically.

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0, shuffle=True)

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(449554, 21) (192666, 21) (449554,) (192666,)
```

Fig 16. Data divided into train and test after using ADASYN.

- Data balancing using SMOTE

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)
X, y = smote.fit_resample(X, y)
```

Fig17. Data balancing using Smote.

- In the below code it is shown how the data is divided in test and train dataset. Also once the data is balanced you can see the difference that the number of samples has been increased drastically.

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=0, shuffle=True)

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(449554, 21) (192666, 21) (449554,) (192666,)
```

Fig 18. Data divided into train and test after using SMOTE.

- Scaling of data Using standard scalar method.

**Scaling of data**

```
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_val = scaler.transform(x_val)
x_test = scaler.transform(x_test)
```

Fig 19. Scaling of data

### 4.2 Model Implementation

- **Machine learning models**
- Below code demonstrates how decision tree classifier is used to prediction and the results below are displayed in the form of Accuracy, classification report and confusion matrix.

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dtPre=dt.fit(x_train, y_train).predict(x_test)
print(accuracy_score(y_test,dtPre))
print(classification_report(y_test,dtPre))
print(confusion_matrix(y_test,dtPre))
```

```
0.8196516250921283
              precision    recall  f1-score   support

           0       0.80      0.69      0.74     64233
           1       0.89      0.97      0.93     64180
           2       0.76      0.80      0.78     64253

    accuracy                           0.82    192666
   macro avg       0.82      0.82      0.82    192666
weighted avg       0.82      0.82      0.82    192666

[[44053  4833 15347]
 [ 1006 62381   793]
 [ 9715  3053 51485]]
```

Fig 20. Decision tree classifier with results.

- Below code demonstrates how AdaBoost is implemented, and the results below are displayed in the form of Accuracy, classification report and confusion matrix.

## Adaboost

```
Ad = AdaBoostClassifier(n_estimators=100, random_state=0)
AdPre=Ad.fit(x_train, y_train).predict(x_test)
print(accuracy_score(y_test,AdPre))
print(classification_report(y_test,AdPre))
print(confusion_matrix(y_test,AdPre))
```

```
0.549853113678594
              precision    recall  f1-score   support

           0       0.63      0.65      0.64     64233
           1       0.52      0.51      0.51     64180
           2       0.50      0.49      0.50     64253

    accuracy                           0.55    192666
   macro avg       0.55      0.55      0.55    192666
weighted avg       0.55      0.55      0.55    192666

[[41655 10385 12193]
 [12429 32877 18874]
 [12476 20371 31406]]
```

Fig 21. AdaBoost classifier with results.

- Below code demonstrates how GaussianNB is implemented, and the results below are displayed in the form of Accuracy, classification report and confusion matrix.

## GaussianNB

```
GNB = GaussianNB()
GNBPre=GNB.fit(x_train, y_train).predict(x_test)
print(accuracy_score(y_test,GNBPre))
print(classification_report(y_test,GNBPre))
print(confusion_matrix(y_test,GNBPre))
```

```
0.4886435593202745
              precision    recall  f1-score   support

           0       0.70      0.41      0.51     64233
           1       0.42      0.74      0.54     64180
           2       0.49      0.32      0.38     64253

    accuracy                           0.49    192666
   macro avg       0.54      0.49      0.48    192666
weighted avg       0.54      0.49      0.48    192666

[[26136 27507 10590]
 [ 5994 47691 10495]
 [ 5365 38570 20318]]
```

Fig 22. GaussianNB Classifier with results.

- Below code demonstrates how KNN is implemented, and the results below are displayed in the form of Accuracy, classification report and confusion matrix.

## KNN

```
KNN = KNeighborsClassifier()
KNNPre = KNN.fit(x_train, y_train).predict(x_test)
print(accuracy_score(y_test,KNNPre))
print(classification_report(y_test,KNNPre))
print(confusion_matrix(y_test,KNNPre))
```

```
C:\Users\piyus\anaconda3\lib\site-packages\sklearn\neighbors\
tions (e.g. `skew`, `kurtosis`), the default behavior of `mod
is behavior will change: the default value of `keepdims` will
e eliminated, and the value None will no longer be accepted.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

```
0.7548711241215368
              precision    recall  f1-score   support

           0       0.78      0.61      0.68     64233
           1       0.77      0.96      0.85     64180
           2       0.72      0.70      0.71     64253

    accuracy                           0.75    192666
   macro avg       0.76      0.75      0.75    192666
weighted avg       0.76      0.75      0.75    192666

[[39155  9003 16075]
 [ 1311 61538  1331]
 [ 9917  9591 44745]]
```

Fig 23. KNN classifier with results.

- **Deep Learning Models**
- This code defines and compiles a simple Recurrent Neural Network (RNN) model using TensorFlow's Keras API for a classification task with three classes. The RNN layer is followed by a dense layer with a softmax activation function for multi-class classification, and the

model is compiled with the Adam optimizer and categorical crossentropy loss.

## RNN

```
from tensorflow.keras.layers import SimpleRNN, Dense
```

```
# Reshape data into 3D array (samples, time steps, features)
X_train_reshaped = x_train.reshape(x_train.shape[0], x_train.shape[1], 1)
X_test_reshaped = x_test.reshape(x_test.shape[0], x_test.shape[1], 1)
```

```
# One-hot encode your labels
y_train_categorical = tf.keras.utils.to_categorical(y_train, num_classes=3)
y_test_categorical = tf.keras.utils.to_categorical(y_test, num_classes=3)
```

```
# Define the model
model = Sequential()
model.add(SimpleRNN(units=50, activation='relu', input_shape=(X_train_reshaped.shape[1], 1)))
model.add(Dense(units=3, activation='softmax'))  # Assuming 3 classes for classification
```

```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Fig 24. Implementation of RNN with results.

- The below code trains the previously defined neural network model (‘model’) on the training data (‘x_train_reshaped’ and ‘y_train_categorical’) for 100 epochs with a batch size of 128. The validation data (‘x_test_reshaped ‘ and ‘y_test_categorical’) is used to assess the model’s performance during training, allowing for monitoring for training and validation accuracy and loss over the specified number of epochs.
- With the same method we have trained and evaluated all the deep learning model that we have implemented i.e. LSTM and GRU.

```
# Train the model
model.fit(X_train_reshaped, y_train_categorical, epochs=100, batch_size=128, validation_data=(X_test_reshaped, y_test_categorical
```

```
Epoch 1/100
2459/2459 [==============================] - 17s 6ms/step - loss: 0.9353 - accuracy: 0.5337 - val_loss: 0.9192 - val_accurac
y: 0.5492
Epoch 2/100
2459/2459 [==============================] - 14s 6ms/step - loss: 0.9141 - accuracy: 0.5509 - val_loss: 0.9098 - val_accurac
y: 0.5561
Epoch 3/100
2459/2459 [==============================] - 15s 6ms/step - loss: 0.9081 - accuracy: 0.5554 - val_loss: 0.9053 - val_accurac
y: 0.5588
Epoch 4/100
2459/2459 [==============================] - 15s 6ms/step - loss: 0.9037 - accuracy: 0.5590 - val_loss: 0.8989 - val_accurac
y: 0.5617
```

Fig 25. Model Training

- Below Image shows the evaluation of models with accuracy, classification matrix.

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test_reshaped, y_test_categorical)
print(f'Accuracy: {accuracy*100:.2f}%')
# Make predictions
y_pred_prob = model.predict(X_test_reshaped)
y_pred = np.argmax(y_pred_prob, axis=1)

# Convert one-hot encoded labels to class labels
y_true = np.argmax(y_test_categorical, axis=1)

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_true, y_pred)
cr= classification_report(y_true, y_pred)

print(cr)
```

```
6021/6021 [==============================] - 12s 2ms/step - loss: 0.8618 - accuracy: 0.5931
Accuracy: 59.31%
6021/6021 [==============================] - 11s 2ms/step
              precision    recall  f1-score   support

           0       0.66      0.63      0.64     64233
           1       0.57      0.65      0.61     64180
           2       0.55      0.50      0.52     64253

    accuracy                           0.59    192666
   macro avg       0.59      0.59      0.59    192666
weighted avg       0.59      0.59      0.59    192666
```

Fig 26. Evaluation of model.

- Cross Validation
- This code uses Scikit-learn to perform 10-fold cross validation with a decision tree classifier ('clf') on input features ('X') and corresponding labels ('y'). it prints the accuracy score for each fold, calculates mean accuracy and standard deviation, and provides a detailed classification report for the entire dataset based on the cross-validation predictions. We have implemented cross-validation method with all 5 machine learning algorithms we have used.

```
clf = DecisionTreeClassifier(random_state=42)
y_pred = cross_val_predict(clf, X, y, cv=10)
scores = cross_val_score(clf, X, y, cv=10)
print(scores)

print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))

cr= classification_report(y, y_pred)

print(cr)
```

```
[0.78000655 0.8388888  0.87038106 0.87056823 0.87671382 0.87559077
 0.87496685 0.87219042 0.87548159 0.87125253]
0.86 accuracy with a standard deviation of 0.03
              precision    recall  f1-score   support

           0       0.85      0.74      0.79    213703
           1       0.92      0.98      0.95    213703
           2       0.80      0.86      0.83    213703

    accuracy                           0.86    641109
   macro avg       0.86      0.86      0.86    641109
weighted avg       0.86      0.86      0.86    641109
```

Fig 27. Cross validation.