AI FOUNDATIONS IN PRACTICE

Coursework-01

ID-2293815

1.1 Prepare the dataset for training (Check for missing values, splitting the dataset in training and testing datasets…).

- First, I imported NumPy and Pandas for numerical computation and data manipulation and analysis. Then I read my Heart Attack dataset.
- The next step is to analyze the data before building a model. The dataset has in total of 129998 rows and 16 columns shown in Table 1.

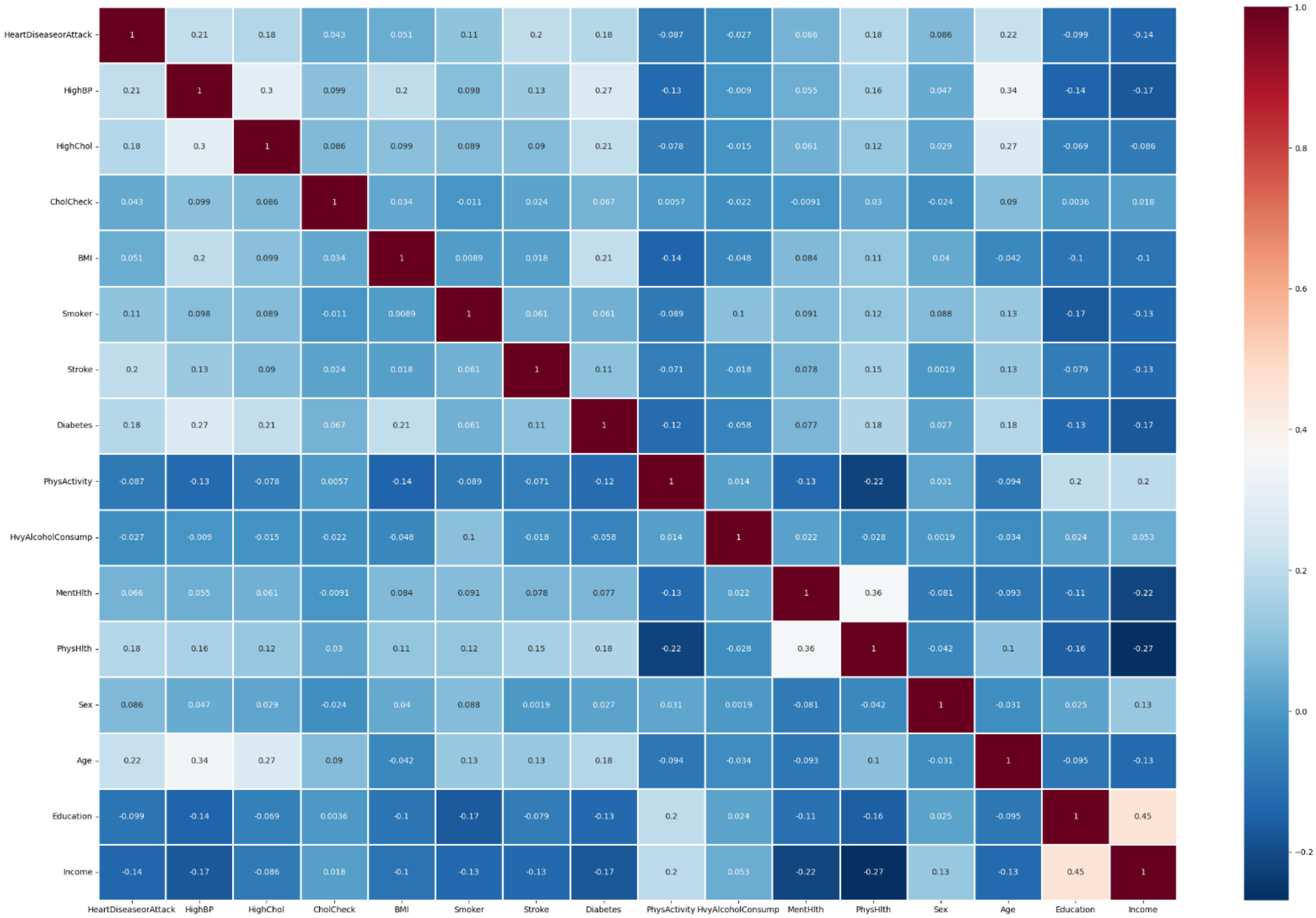
Table 1 All Columns of the dataset

Column name	Data Type
HeartDiseaseorAttack	int64
HighBP	int64
HighChol	int64
CholCheck	int64
BMI	int64
Smoker	int64
Stroke	int64
Diabetes	int64
PhysActivity	int64
HvyAlcoholConsump	int64
MentHlth	int64
PhysHlth	int64
Sex	int64
Age	int64
Education	int64
Income	int64

- All of the columns are numerical. The First column which is ‘HeartDiseaseorAttack’ is our target column. It has only ‘0’ or ‘1’. So, I will do Binary Classification on this dataset.
- Only the BMI column has 10 missing values, and as 10 is a very small number compared to 129998, I decided to drop the null value containing rows, so that it would not affect the model decision making. Now the dataset has 129988 rows.
- Then I checked for outliers by using the describe () function. BMI column has some extremely large values which could be considered as outliers, but as high BMI indicates high chance of getting heart disease and heart attack, I did not remove those extensive values.

- Then I checked for any high correlation between all the numerical attribute, if found so I can remove one column to reduce redundancy, but there was no high correlation as noticed in Figure 1. Values greater than 0.8 or less than -0.8 can be considered as high correlation

Figure 1: Pearsons Correlation between the Numerical Attributes



- After that, I split the dataset into two parts, using scikit-learn library imported train-test-split, training (80%) and testing (20%).
- After that, I applied Min-Max normalization to the features of the dataset for making the values come in a range (0 to 1). I only fit transformed the training data, and only transformed the testing data, because if the testing data is also fit transformed, then the values of training and testing might become same, and we cannot do that. Now my dataset is ready to be trained by Machine Learning models.

1.2 Choose and tailor 4 Models you have learnt during lectures including an ANN based model and train and build the models.

The **general tasks** which are followed for training the models are-

- Importing the model from scikit-learn;
- Initialize the model;

- Fit the training train data to the model;
- Make predictions on the test set (that part of data which is not seen by the model before)

Model Specifications:

- **KNN:** For training the K-Nearest Neighbors classifier, I trained it with k=3 value, meaning it will take nearest 3 neighbors, and based on voting, it will give predictions of the target class.
- **SVM:** For Support Vector Machine classifier, I used the Polynomial kernel which maps the input data into a higher-dimensional space, and it becomes easier to separate classes.
- **Random Forest:** I used the default parameters for Random Forest classifier.
- **ANN:** For Artificial Neural Network classifier, I implemented a neural network with two hidden layers. For implementing it, I scaled my features of the dataset with Standard Scaler which will bring the values of the dataset to a common scale.

  Then I initialized the classifier as sequential because, the layers of the neural network will be one after another.

  After that, I added my input layer and the first hidden layers with 6 neurons, and as I have 15 features in the dataset, the input dimension is set to 15. I used the Relu activation function. Then I added my second hidden layer with same number of neurons and the same activation function.

  Then I added the output layer with sigmoid function, which is for binary classification as it shoots 0 or 1 as an output.

  Then I compiled my model. I used ‘adam’ as the optimizer which determines how the weights of the neural network gets updated and minimizes the loss during training. Then I used loss function as ‘binary_crossentropy’ which measures the loss of training. The Binary Cross Entropy penalizes wrong predictions more intensely and encourages the model to output probabilities that are closer to 0 or 1.
- **XGBoost (Extra):** I used the XGBoost as the extra classifier to train the model. I used all the default parameters in this model.

1.3 Evaluate your model using metrics you have learnt during the lectures.

For evaluating models, I used confusion metrics, accuracy, precision, recall and F1-score. I am reporting the scores below.

KNN:

Figure 2 Confusion Matrix for KNN

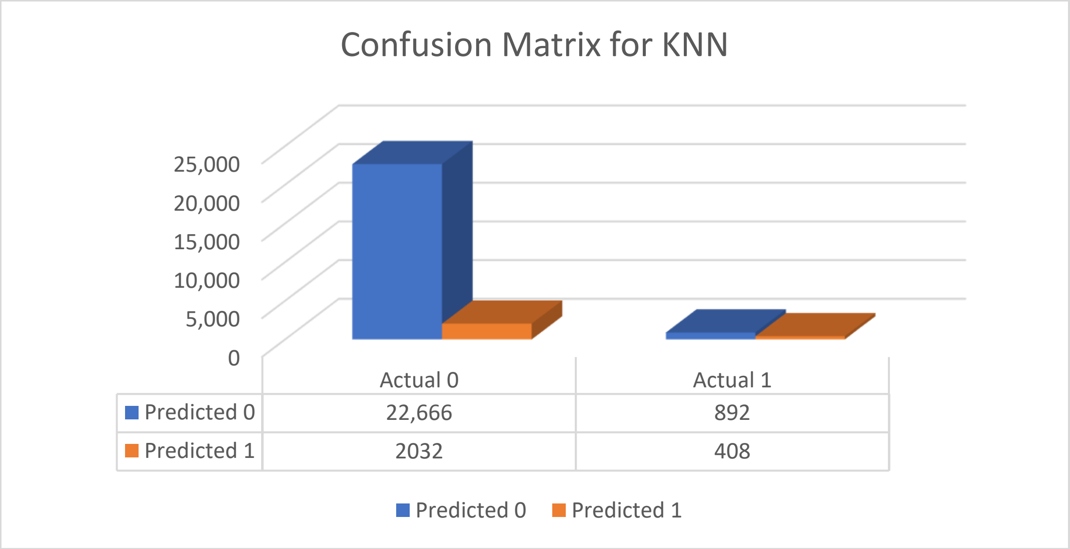
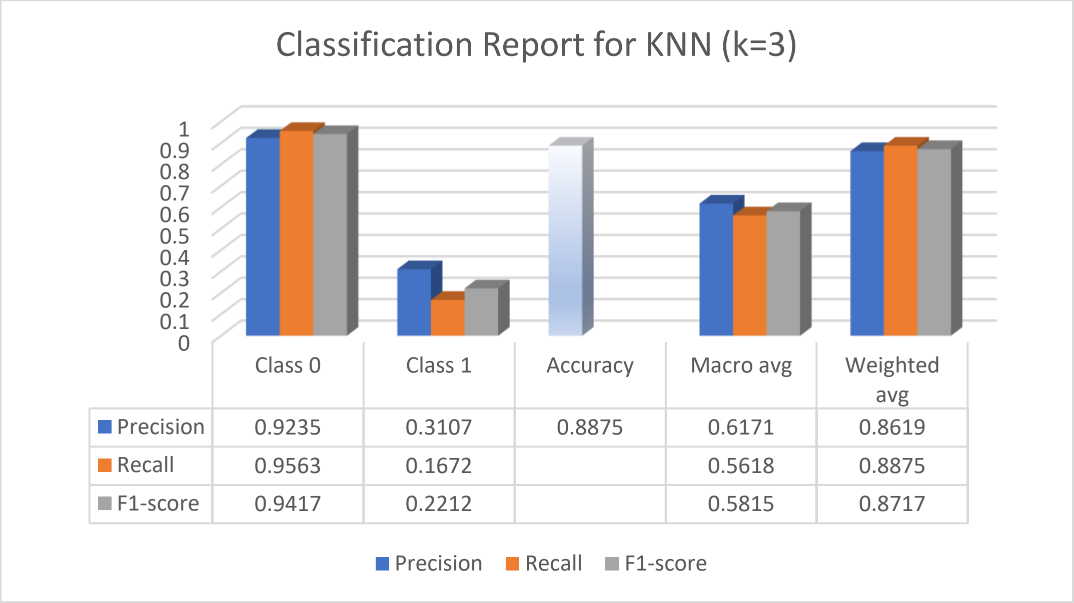


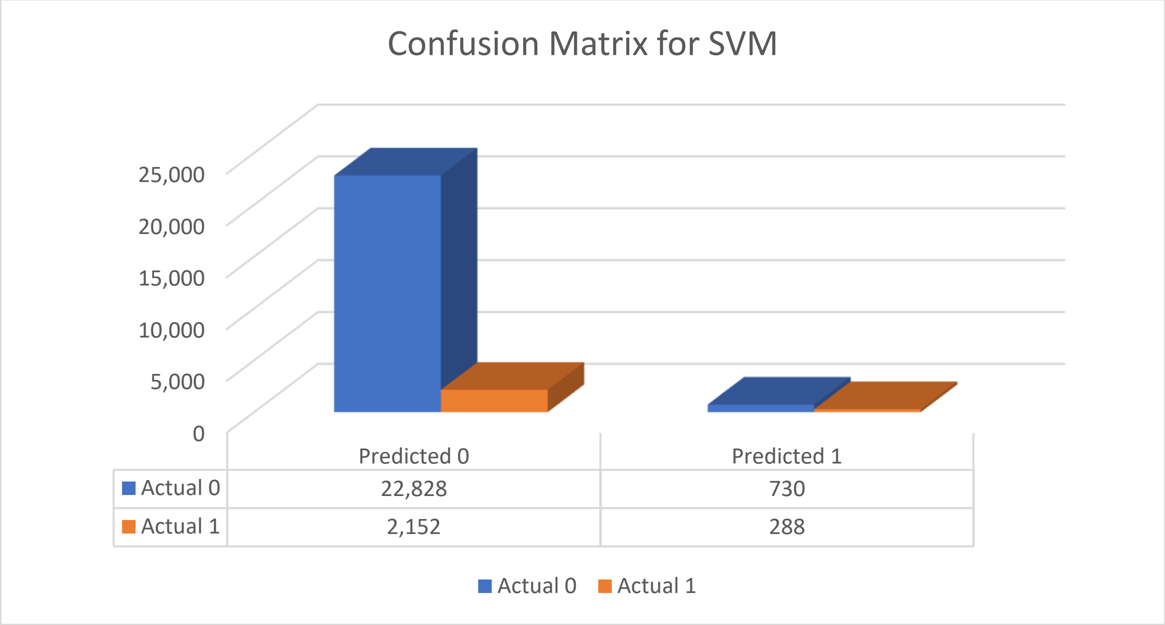
Figure 3 Classification Report for KNN



From the confusion matrix in Figure 2 and Classification Report in Figure 3, it can be said that the KNN model performs very well on Class 0 with high precision (92%) and recall (96%) and F1-score (94%), which is the majority class. With the minority class, all the metrics give poor performance, almost under 30% due to class imbalance. The accuracy which is 88.75% reflects strong performance, however, as the class imbalance is present, accuracy is not reliable.

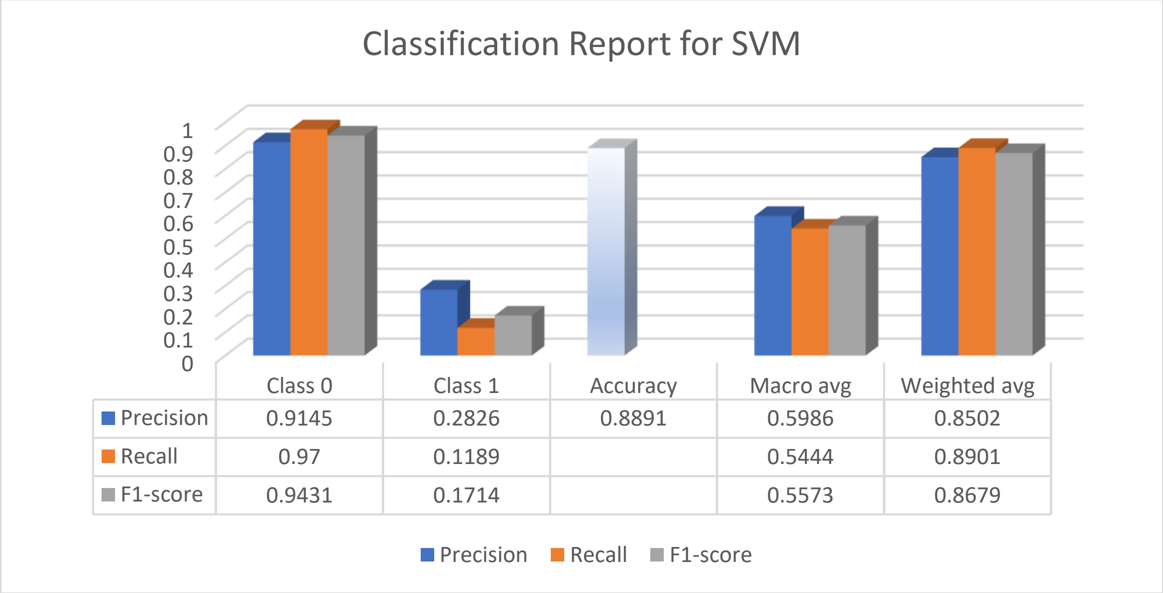
SVM:

Figure 4 Confusion Matrix for SVM



	Predicted 0	Predicted 1
🟦 Actual 0	22,828	730
🟧 Actual 1	2,152	288

Figure 5 Classification Report for SVM

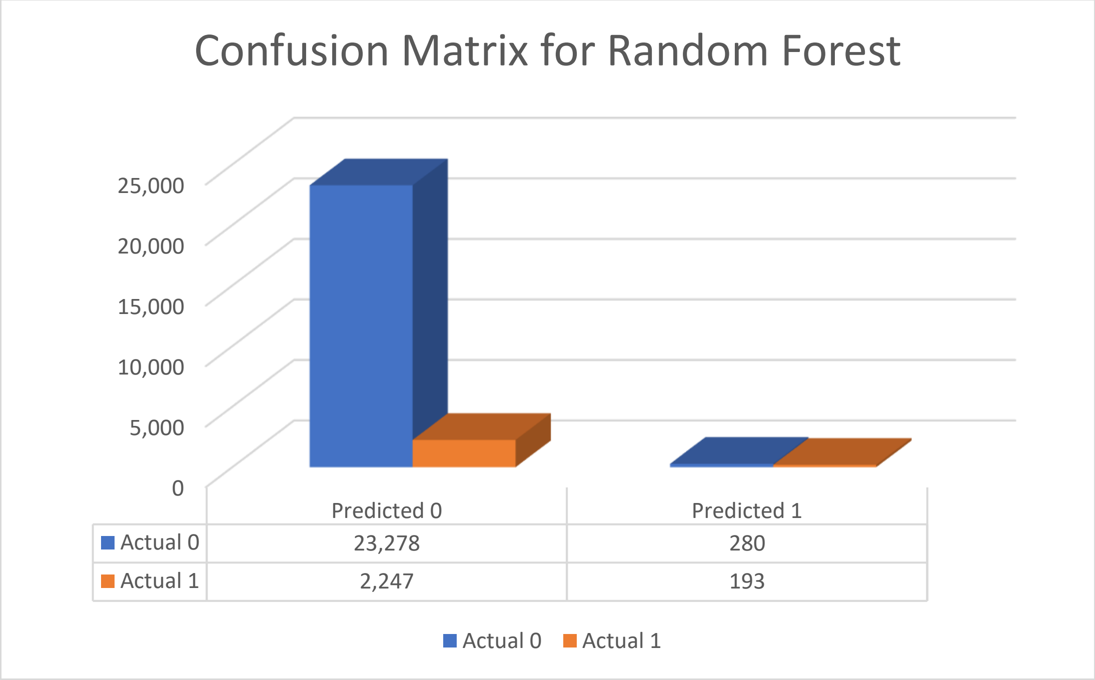


	Class 0	Class 1	Accuracy	Macro avg	Weighted avg
🟦 Precision	0.9145	0.2826	0.8891	0.5986	0.8502
🟧 Recall	0.97	0.1189		0.5444	0.8901
⬜ F1-score	0.9431	0.1714		0.5573	0.8679

From Figure 4 and Figure 5, it is seen that, the SVM model also performs very well on the majority class (class 0), but poor on the minority class (class 1).

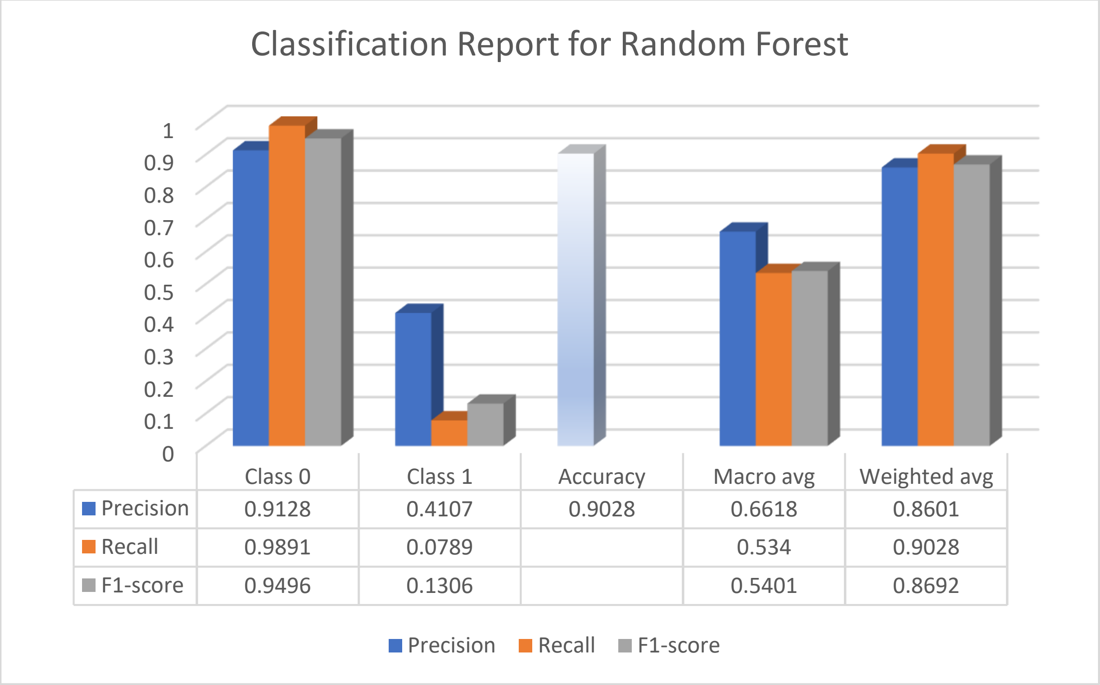
Random Forest:

Figure 6 Confusion Matrix for Random Forest



	Predicted 0	Predicted 1
Actual 0	23,278	280
Actual 1	2,247	193

Figure 7 Classification Report for Random Forest

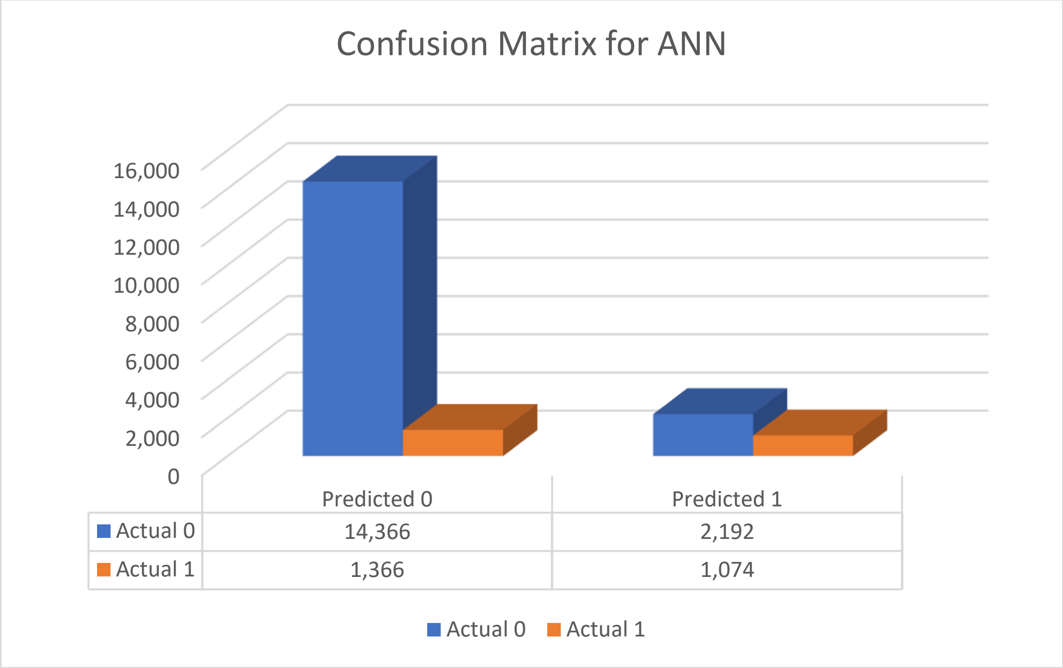


	Class 0	Class 1	Accuracy	Macro avg	Weighted avg
Precision	0.9128	0.4107	0.9028	0.6618	0.8601
Recall	0.9891	0.0789		0.534	0.9028
F1-score	0.9496	0.1306		0.5401	0.8692

From Figure 6 and Figure 7, seeing the confusion matrix and classification report, also in the Random Forest model evaluation, the dominance of Class 0 is clearly visible.

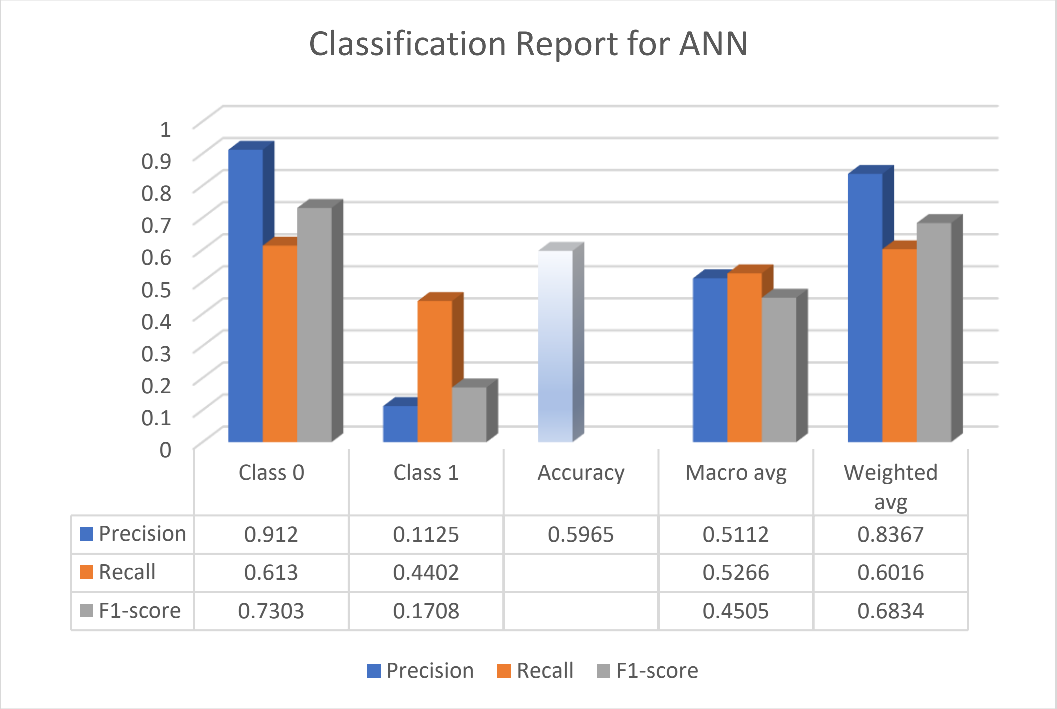
ANN:

Figure 8 Confusion Matrix for ANN



	Predicted 0	Predicted 1
🟦 Actual 0	14,366	2,192
🟧 Actual 1	1,366	1,074

Figure 9 Classification Report for ANN



	Class 0	Class 1	Accuracy	Macro avg	Weighted avg
🟦 Precision	0.912	0.1125	0.5965	0.5112	0.8367
🟧 Recall	0.613	0.4402		0.5266	0.6016
⬜ F1-score	0.7303	0.1708		0.4505	0.6834

In Figure 9 and Figure 10, the confusion matrix and the classification report suggest the accuracy in this ANN model is moderately low (59.65%). It performs poor overall, but in class 1, its performance is poorer. This ANN model needs to be adjusted through hyperparameter tuning and handling class imbalance.

<u>XGBoost (Extra):</u>

Figure 10 Confusion Matrix for XGBoost

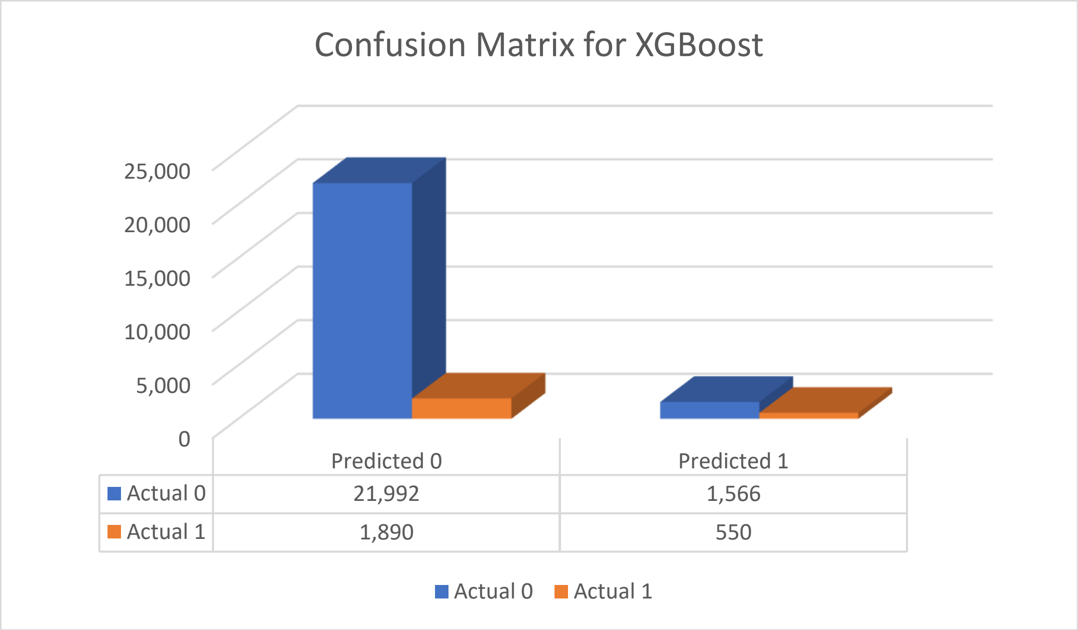
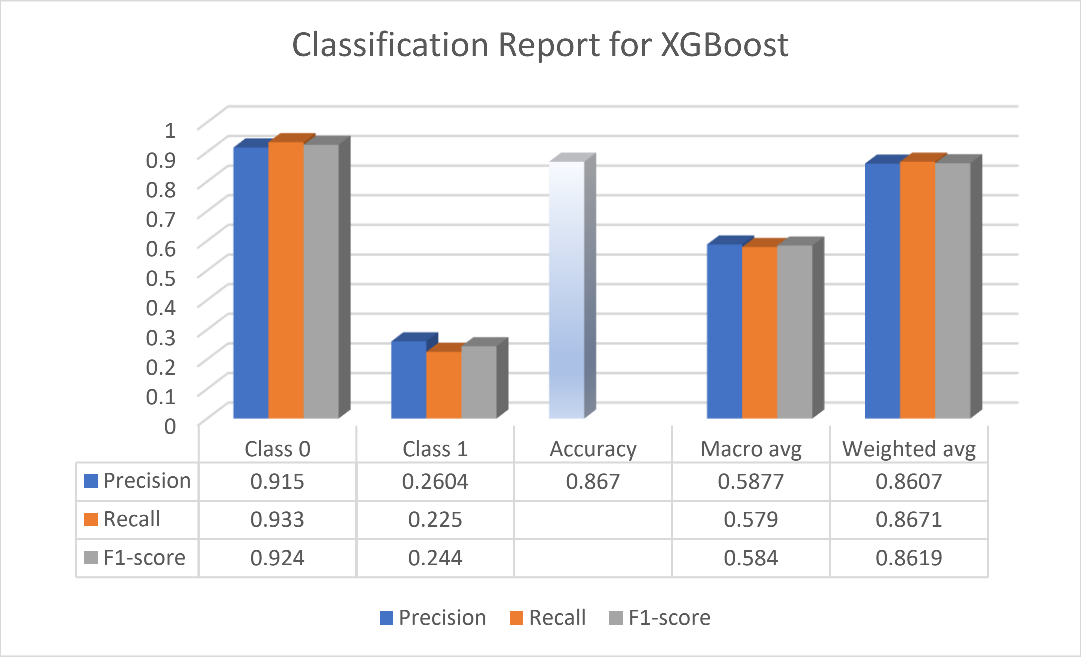


Figure 11 Classification Report for XGBoost

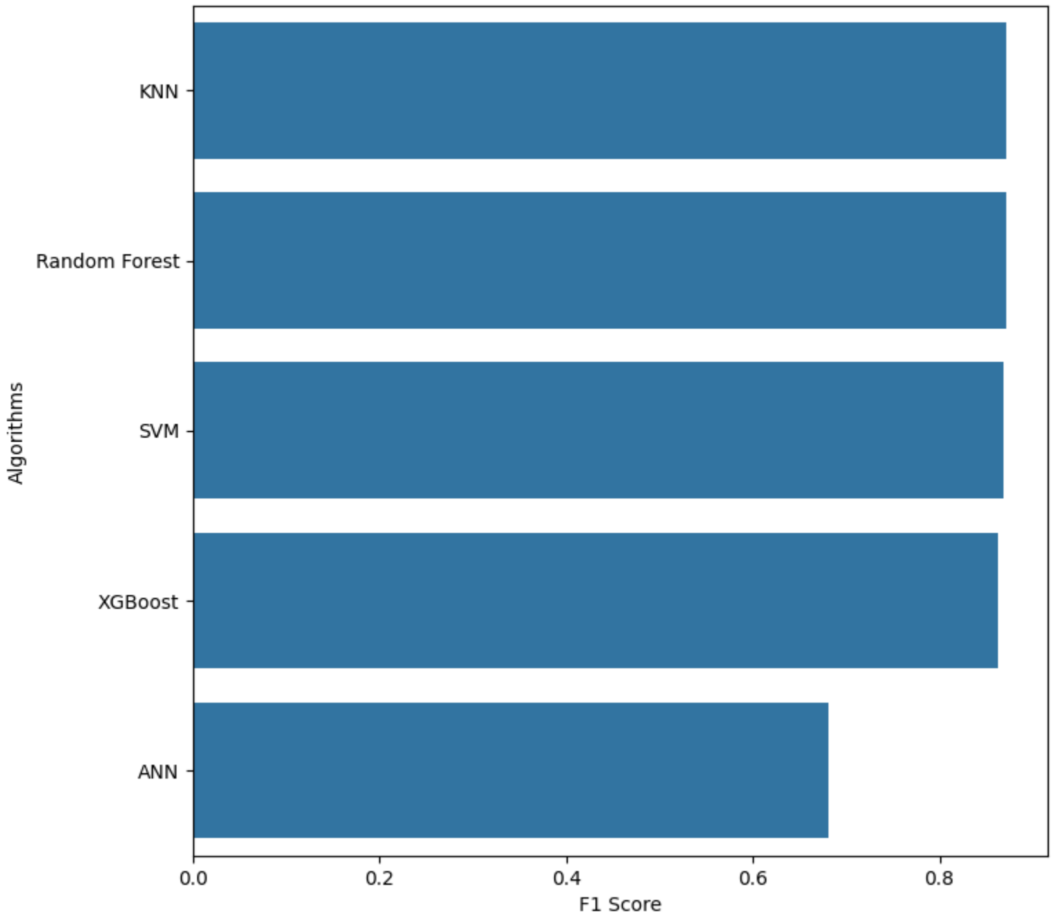


From Figure 10 and Figure 11, it can be seen that the XGBoost model also struggles with class 1 performing poor on that class and excels at Class 0.

1.4 Compare the models and report the most accurate model for this purpose.

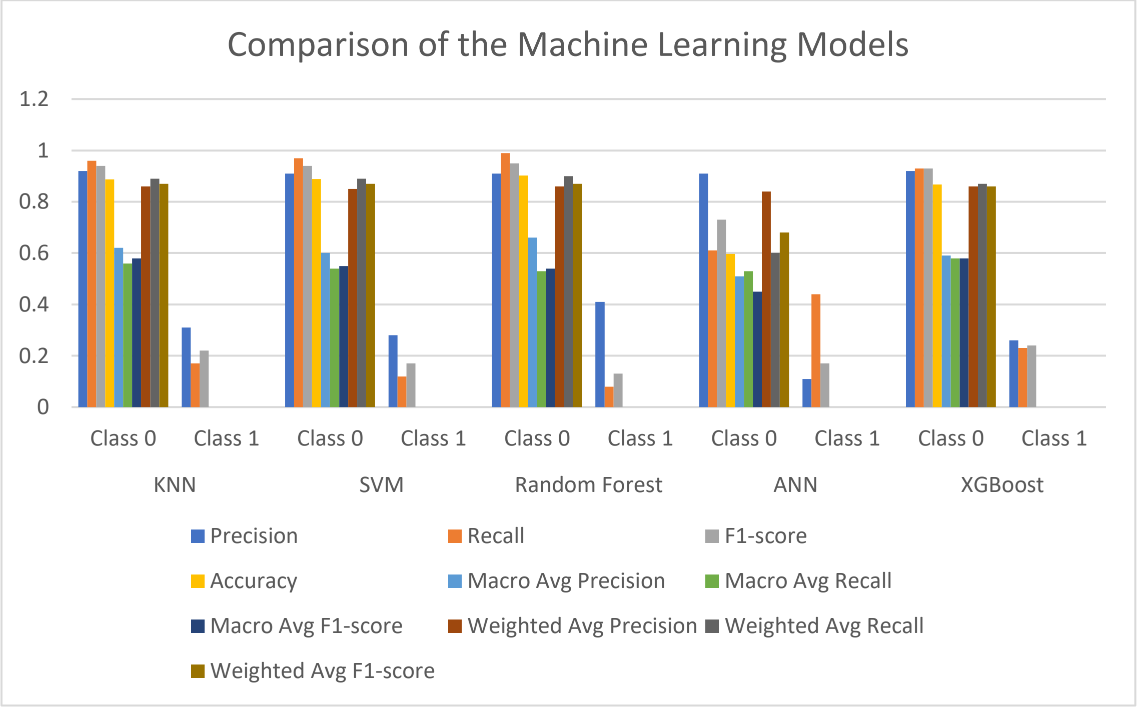
For comparing the models, I used F1-score as a reliable measure.

Figure 12 Model Comparison using F1-score



From Figure 12, on the comparison of F1-score, it can be seen that KNN performed the best among all the models.

Figure 13 Comparison of the Machine Learning Models



In Figure 13, an overall comparison of all five models is shown across 2 two classes where the supremacy of KNN is visible in accuracy as well as F1-score.

1.5 Please provide 2 ways (with evidence) to further improve the accuracy of your best model.

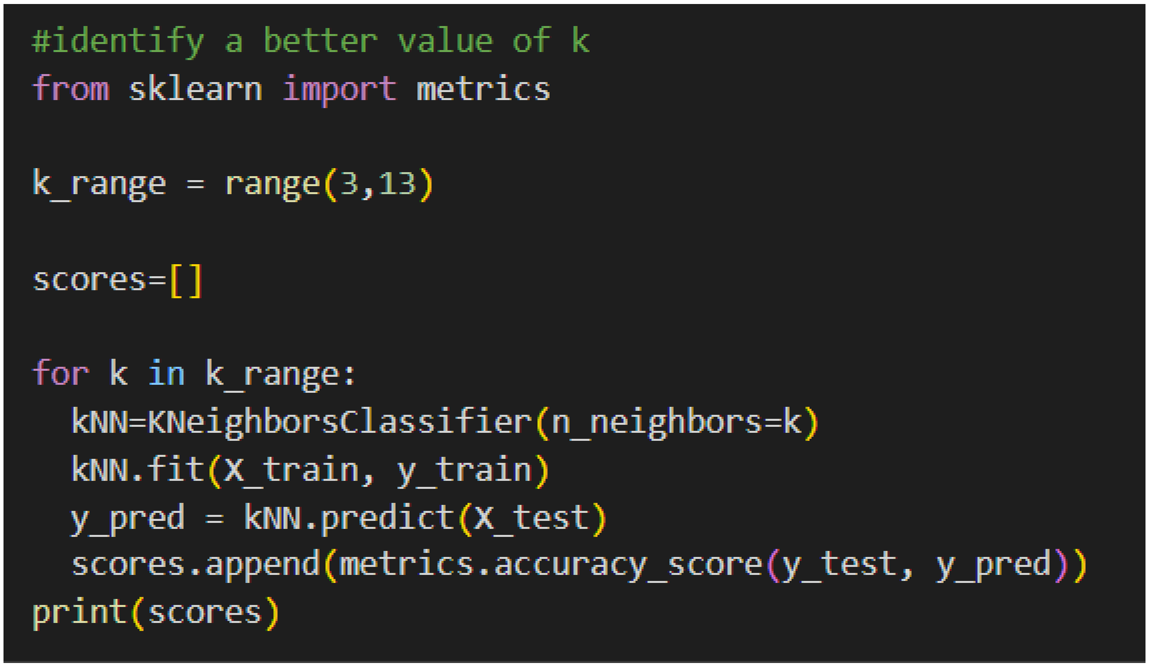
As my best Model was KNN, the two methos I applied was-

1. To check for the best K value and run the model again with that k;
2. Doing 10-fold Cross validation across a range of K, finding the best k, and then predicting the test dataset on that best k valued model.

The two ways are shown below with the achieved improvement-

1. **Finding better K value-**

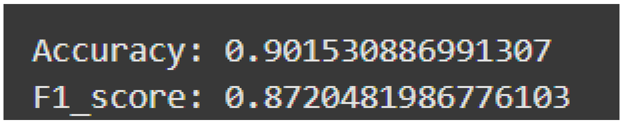
Figure 14 Finding better K value



```
#identify a better value of k
from sklearn import metrics

k_range = range(3,13)

scores=[]

for k in k_range:
  kNN=KNeighborsClassifier(n_neighbors=k)
  kNN.fit(X_train, y_train)
  y_pred = kNN.predict(X_test)
  scores.append(metrics.accuracy_score(y_test, y_pred))
print(scores)
```

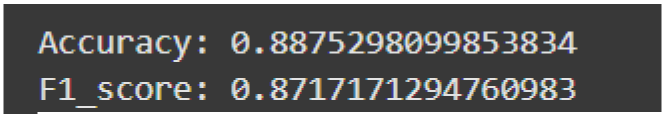
In Figure 14, I searched for better K values in the range of 3 to 12(including). And I got the best optimum accuracy at K=6, even the F1-score also increased.

Figure 15 Improved Accuracy and F1-score



```
Accuracy: 0.901530886991307
F1_score: 0.8720481986776103
```

Figure 16 Previous Accuracy and F1-score

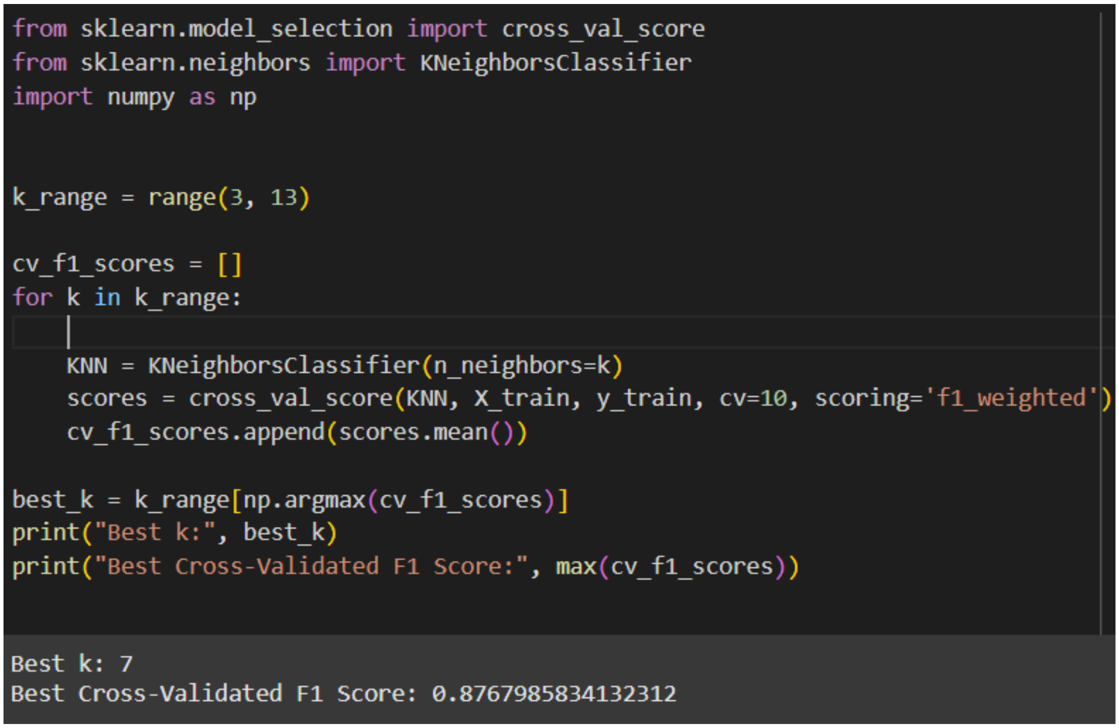


```
Accuracy: 0.8875298099853834
F1_score: 0.8717171294760983
```

From Figure 15 and Figure 16, It is evident that, after changing the k value the model improvised.

2. **Using Cross validation-**

Figure 17 Cross Validation to Improve model

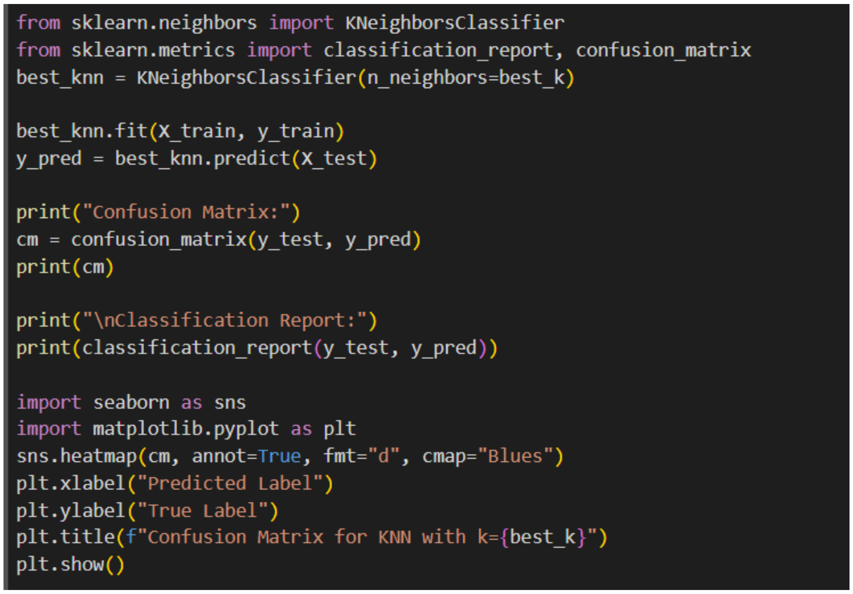


```
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
import numpy as np


k_range = range(3, 13)

cv_f1_scores = []
for k in k_range:

    KNN = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(KNN, X_train, y_train, cv=10, scoring='f1_weighted')
    cv_f1_scores.append(scores.mean())

best_k = k_range[np.argmax(cv_f1_scores)]
print("Best k:", best_k)
print("Best Cross-Validated F1 Score:", max(cv_f1_scores))
```

```
Best k: 7
Best Cross-Validated F1 Score: 0.8767985834132312
```

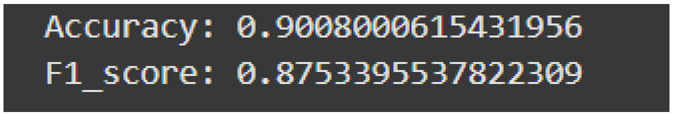
Here in Figure 17, I looped through 3 to 13 (excluding) and implemented 10-fold cross validation. I stored the model which has best F1-score in the best_k, then I predicted my test dataset on it, as shown in Figure 18.

Figure 18 Implementing with Cross validated Best K



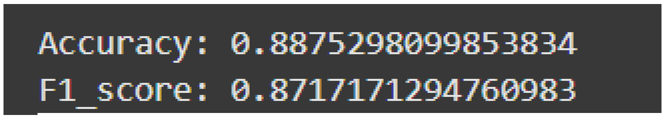
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
best_knn = KNeighborsClassifier(n_neighbors=best_k)

best_knn.fit(X_train, y_train)
y_pred = best_knn.predict(X_test)

print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title(f"Confusion Matrix for KNN with k={best_k}")
plt.show()
```

Figure 19 Accuracy and F1-score after cross validation



```
Accuracy: 0.9008000615431956
F1_score: 0.8753395537822309
```

Figure 20 Accuracy and F1-score after cross validation



```
Accuracy: 0.8875298099853834
F1_score: 0.8717171294760983
```

From Figure 19 and Figure 20, it can be seen the accuracy and the F1-score improved a little bit after doing the cross validation and with K value=7.