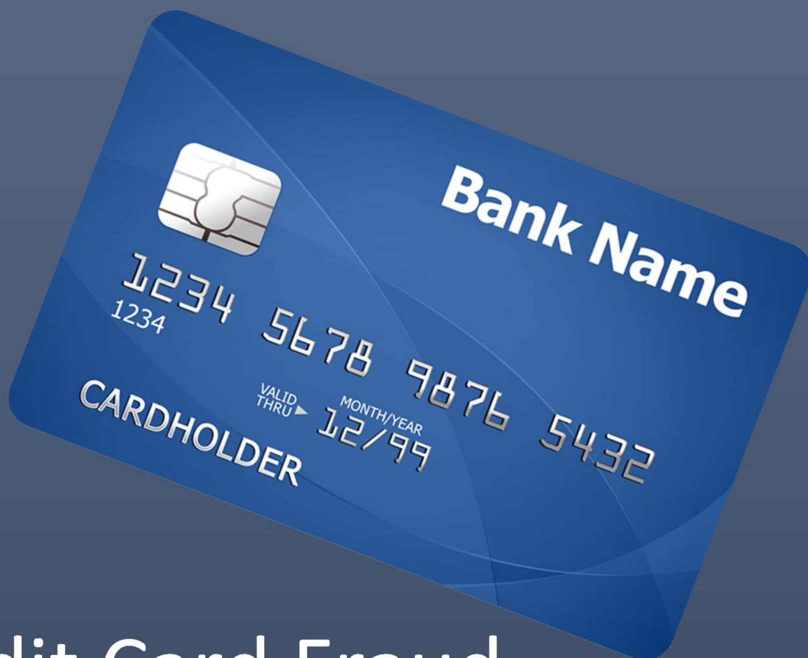
2022



Credit Card Fraud Detection

Under Supervision of: Mr. Ubaid Zaman

INTRODUCTION TO DATA SCIENCE

AMIN M QURAISHI (63855) & BUSHRA MUNEER (63759)

PAF-KIET | Karachi Institute of Economics and Technology

Contents

Importing Libraries

Adding files and loading dataset into data frame

First five data of dataset

Last five data of dataset

Information of dataset

Check for the missing values in each column

Distribution of normal and fraud transactions

Separating the data

Comparing the values of both normal and fraud transactions

Under sampling

K-Neighbors Classifier

Logistic Regression

Decision Tree

Bagging Classifier (base as Logistic Regression)

Random Forest Classifier

Summary

Credit Card Fraud Detection

Importing Libraries

```
[ ] from google.colab import drive
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import accuracy_score, recall_score,precision_score
    from sklearn.metrics import confusion_matrix,classification_report
    from sklearn.metrics import plot_confusion_matrix
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier
    from sklearn import model_selection
    from sklearn import tree
    from sklearn.neighbors import KNeighborsClassifier
```

Adding Files and loading dataset into dataframe

```
[ ] drive.mount('/content/drive')
    df = pd.read_csv('/content/drive/My Drive/creditcard.csv')

    Mounted at /content/drive
```

First five data of dataset

```
[ ] df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

5 rows × 31 columns

last five data of dataset

```
df.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.9
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.5
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.4
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.3
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.4

5 rows × 31 columns

Information of dataset

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Check for the missing values in each column

```
[ ] df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Distribution of normal and fraud transactions

```
[ ] df['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

dataset highly unblanced..

0 normal transanction, 1 fraudulant transanction

Separating the data

```
[ ] non_fraud=df[df.Class == 0]
    fraud=df[df.Class == 1]
```

```
[ ] print(non_fraud.shape)
    print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

Comparing the values of both normal and fraud transactions

```
[ ] df.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-:

2 rows × 30 columns

Under sampling

Building a sample dataset which will be containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions : 492

```
[ ] non_fraud_sample = non_fraud.sample(n=492)
```

```
[ ] new_dataset = pd.concat([non_fraud_sample,fraud],axis=0)
```

```
[ ] new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
102920	68427.0	-0.312981	0.505159	1.603193	0.701377	-0.195373	-0.521946	0.280061	0.038075	0.15
84973	60544.0	1.103947	0.219566	0.155744	1.027188	-0.051178	-0.400656	0.212803	0.000572	-0.35
35168	37998.0	1.046539	-0.407327	0.919160	0.649235	-1.184232	-0.663533	-0.459903	0.030745	0.70
136323	81648.0	-0.173644	-2.953591	-0.672841	0.080156	-1.715645	-0.826130	0.830937	-0.552689	-1.17
242982	151741.0	-0.055903	-0.976752	0.030459	-0.863338	0.548546	-0.533087	-0.619720	0.048248	-0.54

5 rows × 31 columns

```
new_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 984 entries, 102920 to 281674
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Time    984 non-null    float64
 1   V1      984 non-null    float64
 2   V2      984 non-null    float64
 3   V3      984 non-null    float64
 4   V4      984 non-null    float64
 5   V5      984 non-null    float64
 6   V6      984 non-null    float64
 7   V7      984 non-null    float64
 8   V8      984 non-null    float64
 9   V9      984 non-null    float64
 10  V10     984 non-null    float64
 11  V11     984 non-null    float64
 12  V12     984 non-null    float64
 13  V13     984 non-null    float64
 14  V14     984 non-null    float64
 15  V15     984 non-null    float64
 16  V16     984 non-null    float64
 17  V17     984 non-null    float64
 18  V18     984 non-null    float64
 19  V19     984 non-null    float64
 20  V20     984 non-null    float64
 21  V21     984 non-null    float64
 22  V22     984 non-null    float64
 23  V23     984 non-null    float64
 24  V24     984 non-null    float64
 25  V25     984 non-null    float64
 26  V26     984 non-null    float64
 27  V27     984 non-null    float64
 28  V28     984 non-null    float64
 29  Amount  984 non-null    float64
 30  Class   984 non-null    int64  
dtypes: float64(30), int64(1)
memory usage: 246.0 KB
```

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577

5 rows × 31 columns

```
new_dataset['Class'].value_counts()
```

```
0    492
1    492
Name: Class, dtype: int64
```

```
X=new_dataset.drop(columns='Class',axis=1)
Y=new_dataset['Class']
```

```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
102920   68427.0 -0.312981  0.505159  1.603193  0.701377 -0.195373 -0.521946
84973    60544.0  1.103947  0.219566  0.155744  1.027188 -0.051178 -0.400656
35168    37998.0  1.046539 -0.407327  0.919160  0.649235 -1.184232 -0.663533
136323   81648.0 -0.173644 -2.953591 -0.672841  0.080156 -1.715645 -0.826130
242982  151741.0 -0.055903 -0.976752  0.030459 -0.863338  0.548546 -0.533087
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
102920  0.280061  0.038075  0.156139  ... -0.029140  0.037971  0.277495
84973   0.212803  0.000572 -0.352222  ... -0.149483  0.074633  0.177438
35168  -0.459903  0.030745  0.709154  ...  0.010701  0.033611 -0.160247
136323  0.830937 -0.552689 -1.170958  ...  1.246760  0.118773 -0.984116
242982 -0.619720  0.048248 -0.547072  ...  0.473728  0.510498  1.355643
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amount
102920 -0.119164  0.433383 -0.391840  0.435189  0.132972  0.154542    4.99
84973  -0.070577  0.210336  0.574347 -0.359584  0.008746  0.008916   30.76
35168   0.001088  0.386509  0.068980  0.386655 -0.033160  0.039037   95.70
136323 -0.686252  0.495051  0.006298  0.949754 -0.214172  0.149118  825.73
242982  0.349074  0.570515 -1.661312 -0.240226  0.148534  0.131473   55.00
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

```
print(Y)
```

```
102920    0
84973     0
35168     0
136323    0
242982    0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Splitting data to test and train

```
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, stratify=Y, random_state=1)
```

```
[ ] print(X.shape, X_train.shape, X_test.shape)

    (984, 30) (738, 30) (246, 30)
```

KNeighbors Classifier

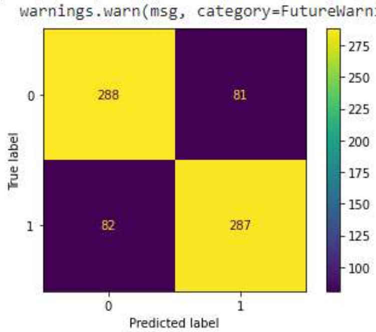
K- NN is a Supervised Learning Technique which assumes the similarity b/w new and avaialble data and put the new data into the category that is most similar to the available categories.

```
[ ] classifier = KNeighborsClassifier()
    classifier.fit(X_train, Y_train)
    X_test_prediction = classifier.predict(X_test)
    X_train_prediction = classifier.predict(X_train)
```

Train Data

```
[ ] print(confusion_matrix(Y_train, X_train_prediction))
    plot_confusion_matrix(classifier,X_train,Y_train)
    plt.show()
    print(classification_report(Y_train, X_train_prediction))
    print('Accuracy Using KNeighbors classifier on Train Data is ',accuracy_score(X_train_prediction,Y_train)*100)
```

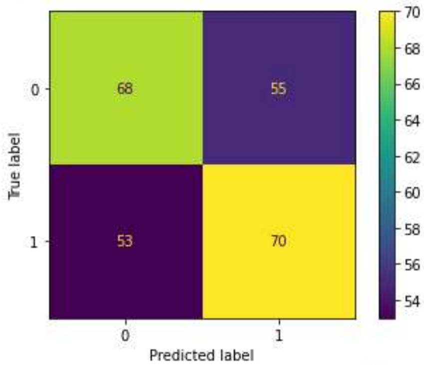
```
[[288  81]
 [ 82 287]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matri
  warnings.warn(msg, category=FutureWarning)
```




```
              precision    recall  f1-score   support

           0       0.78      0.78      0.78       369
           1       0.78      0.78      0.78       369

    accuracy                           0.78       738
   macro avg       0.78      0.78      0.78       738
weighted avg       0.78      0.78      0.78       738

Accuracy Using KNeighbors classifier on Train Data is  77.91327913279133
```

```
[ ] print(confusion_matrix( Y_test,X_test_prediction))
    plot_confusion_matrix(classifier,X_test,Y_test)
    plt.show()
    print(classification_report(Y_test, X_test_prediction))
```

```
[[68 55]
 [53 70]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matri
  warnings.warn(msg, category=FutureWarning)
```




```
              precision    recall  f1-score   support

           0       0.56      0.55      0.56       123
           1       0.56      0.57      0.56       123

    accuracy                           0.56       246
   macro avg       0.56      0.56      0.56       246
weighted avg       0.56      0.56      0.56       246
```

```
print('KNeighbors classifier Accuracy -> ',accuracy_score(X_test_prediction,Y_test)*100)
print("KNeighbors classifier Precision Score -> ",precision_score(X_test_prediction, Y_test,average = 'weighted')*100)
print("KNeighbors classifier Recall Score -> ",recall_score(X_test_prediction, Y_test,average = 'weighted')*100)
```

```
KNeighbors classifier Accuracy ->  60.16260162601627
KNeighbors classifier Precision Score ->  60.16260162601627
KNeighbors classifier Recall Score ->  60.16260162601627
```

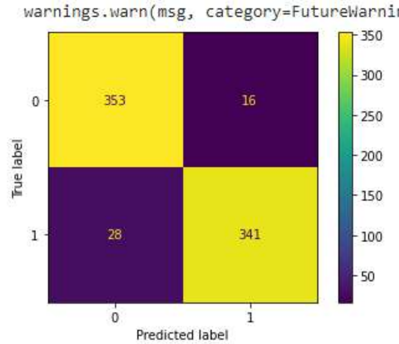
Logistic Regression

Logistic Regression is a Supervised Learning Technique and is used for solving the Classification problems, predict the categorical dependent variable with the help of independent variables.

```
[ ] classifier = LogisticRegression()
    classifier.fit(X_train, Y_train)
```

```
X_train_prediction = classifier.predict(X_train)
print(confusion_matrix( Y_train,X_train_prediction))
plot_confusion_matrix(classifier,X_train,Y_train)
plt.show()
print(classification_report( X_train_prediction,Y_train))
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

```
[[353  16]
 [ 28 341]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is dep
  warnings.warn(msg, category=FutureWarning)
```



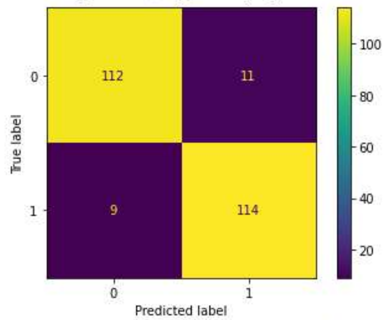

```
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       381
           1       0.92      0.96      0.94       357

    accuracy                           0.94       738
   macro avg       0.94      0.94      0.94       738
weighted avg       0.94      0.94      0.94       738
```

```
[ ] print('Accuracy Using Logistic Regression on Training data : ', (training_data_accuracy*100))
```

```
Accuracy Using Logistic Regression on Training data :  94.03794037940379
```

```
X_test_prediction = classifier.predict(X_test)
print(confusion_matrix( Y_test,X_test_prediction))
plot_confusion_matrix(classifier,X_test,Y_test)
plt.show()
print(classification_report(Y_test, X_test_prediction))
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
```

```
[[112  11]
 [  9 114]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is dep
  warnings.warn(msg, category=FutureWarning)
```




```
              precision    recall  f1-score   support

           0       0.93      0.91      0.92       123
           1       0.91      0.93      0.92       123

    accuracy                           0.92       246
   macro avg       0.92      0.92      0.92       246
weighted avg       0.92      0.92      0.92       246
```

```
print('Logistic Regression Accuracy Score -> : ', test_data_accuracy*100)
print("Logistic Regression Precision Score -> ",precision_score(X_test_prediction, Y_test,average = 'weighted')*100)
print("Logistic Regression Recall Score -> ",recall_score(X_test_prediction, Y_test,average = 'weighted')*100)
```

```
Logistic Regression Accuracy Score -> :  91.869918699187
Logistic Regression Precision Score ->  60.16260162601627
Logistic Regression Recall Score ->  60.16260162601627
```

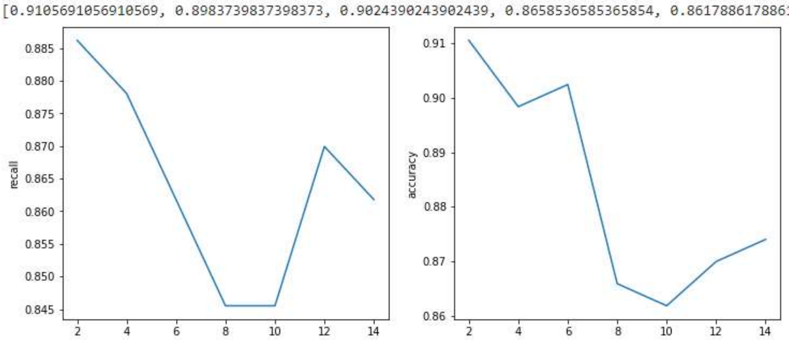
Decision Tree

Decision Tree is a Supervised learning technique based on a a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.There are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

```
accuracy=[]
recall_scores=[]
max_depths=[2,4,6,8,10,12,14]
for n in max_depths:
    clf_u=DecisionTreeClassifier(max_depth=n)
    clf_u.fit(X_train,Y_train)
    predict=clf_u.predict(X_test)
    recall_scores.append(recall_score(Y_test,predict))
    accuracy.append(clf_u.score(X_test,Y_test))
```

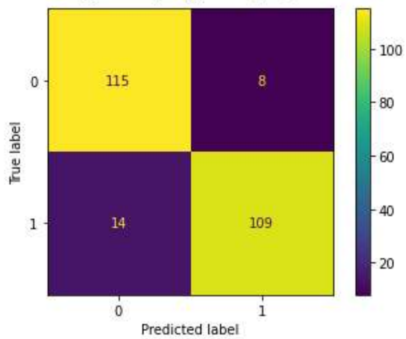
```
[ ] fig,(ax1,ax2)=plt.subplots(1,2,figsize=(12,5))

figs=[ax1,ax2]

def plot_graphs(fig_index,y,y_label,X=max_depths):
    global figs
    figs[fig_index].plot(X,y)
    figs[fig_index].set_ylabel(y_label)
print(accuracy)
plot_graphs(1,accuracy,'accuracy')
plot_graphs(0,recall_scores,'recall')
```

```
[0.9105691056910569, 0.8983739837398373, 0.9024390243902439, 0.8658536585365854, 0.8617886178861789, 0.8699186991869918, 0.8
```



```
#training the decision tree classifier with the max depth that has the maximum recall
max_depth=max_depths[recall_scores.index(max(recall_scores))]
clf_u=DecisionTreeClassifier(max_depth=max_depth)
clf_u.fit(X_train,Y_train)
predict=clf_u.predict(X_test)
recall_score(Y_test,predict)
print(confusion_matrix( Y_test,predict))
plot_confusion_matrix(clf_u,X_test,Y_test)
plt.show()
print(classification_report(Y_test, predict))
test_data_accuracy = accuracy_score(predict, Y_test)
print('Accuracy Using Decision Tree on Test Data : ', test_data_accuracy*100)
print('Max_depth with maximum recall  {}'.format(max_depth))
```

```
[[115   8]
 [ 14 109]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is dep
  warnings.warn(msg, category=FutureWarning)
```

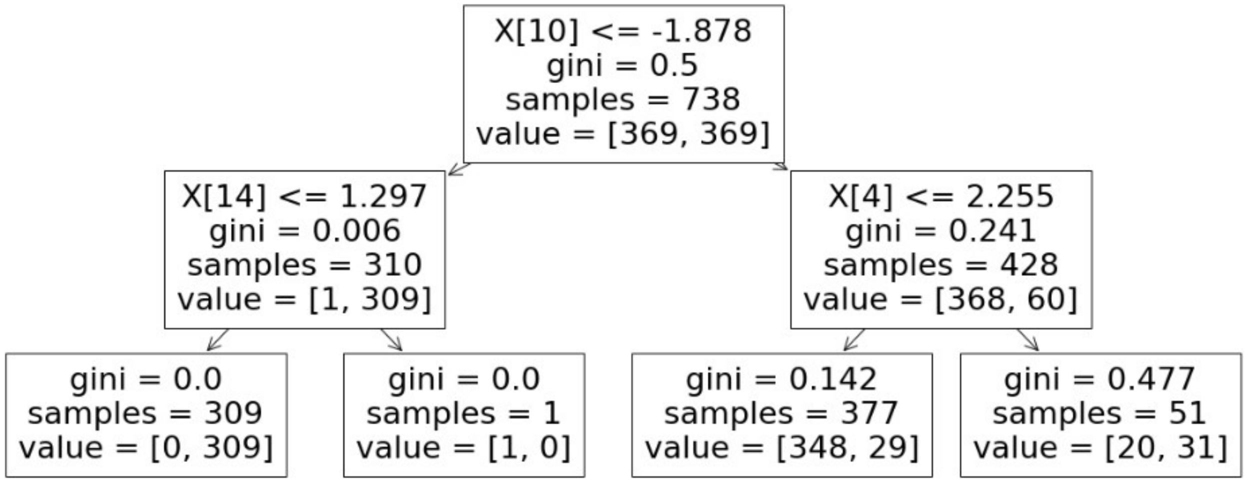



```
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       123
           1       0.93      0.89      0.91       123

    accuracy                           0.91       246
   macro avg       0.91      0.91      0.91       246
weighted avg       0.91      0.91      0.91       246

Accuracy Using Decision Tree on Test Data :  91.05691056910568
Max_depth with maximum recall  2
```

```
# Use accuracy_score function to get the accuracy
print('Decision Tree Accuracy Score : ', test_data_accuracy*100)
print("Decision Tree Precision Score -> ",precision_score(predict, Y_test,average = 'weighted')*100)
print("Decision Tree Recall Score -> ",recall_score(predict, Y_test,average = 'weighted')*100)
```

```
Decision Tree Accuracy Score :  91.05691056910568
Decision Tree Precision Score ->  91.17588736862979
Decision Tree Recall Score ->  91.05691056910568
```

```
[ ] import sklearn
    plt.figure(figsize=(20,8))
    sklearn.tree.plot_tree(clf_u,max_depth=3);
```

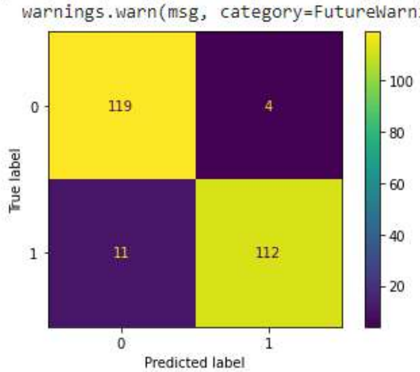


Bagging Classifier (base as Logistic Regression)

A Bagging classifier is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier
pipeline = make_pipeline(StandardScaler(),LogisticRegression())
bgclassifier = BaggingClassifier(base_estimator=pipeline)
predictions=bgclassifier.fit(X_train, Y_train)
predictions =  bgclassifier.predict(X_test)
print(confusion_matrix(predictions,Y_test))
plot_confusion_matrix(bgclassifier,X_test,Y_test)
plt.show()
print('Model test Score: %.3f, ' %bgclassifier.score(X_test, Y_test),
      'Model training Score: %.3f' %bgclassifier.score(X_train, Y_train))
```

```
[[119  11]
 [  4 112]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is dep
  warnings.warn(msg, category=FutureWarning)
```




```
Model test Score: 0.939,  Model training Score: 0.950
```

```
print('Accuracy Using Bagging Classifier with Logistic Regression on Training data : ', (bgclassifier.score(X_train, Y_train)*100))
print('Accuracy Using Bagging Classifier with Logistic Regression on Test Data : ', bgclassifier.score(X_test, Y_test)*100)
```

```
Accuracy Using Bagging Classifier with Logistic Regression on Training data :  94.98644986449864
Accuracy Using Bagging Classifier with Logistic Regression on Test Data :  93.90243902439023
```

```
[ ]
print("Bagging Classifier with Logistic Regression Accuracy : ", accuracy_score(predictions,Y_test)*100)
print("Bagging Classifier with Logistic Regression Precision : ", precision_score(predictions,Y_test,average = 'weighted')*100)
print("Bagging Classifier with Logistic Regression Recall : ", recall_score(predictions,Y_test,average = 'weighted')*100)
```

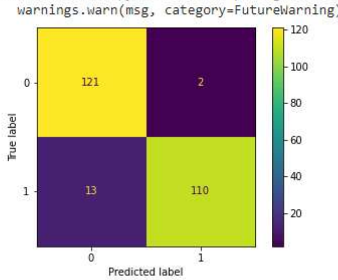
```
Bagging Classifier with Logistic Regression Accuracy :  93.90243902439023
Bagging Classifier with Logistic Regression Precision :  94.06437966818693
Bagging Classifier with Logistic Regression Recall :  93.90243902439023
```

Random Forest Classifier

The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.

```
from sklearn import metrics
classifier = RandomForestClassifier(n_estimators = 100)
classifier.fit(X_train, Y_train)
y_pred = classifier.predict(X_test)
recall_score(Y_test,y_pred)
print(confusion_matrix(y_pred,Y_test))
plot_confusion_matrix(classifier,X_test,Y_test)
plt.show()
print(classification_report(Y_test, y_pred))
print('Accuracy Using Random Forest Classifier', metrics.accuracy_score(Y_test, y_pred)*100)
```

```
[[121  13]
 [  2 110]]
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confus
  warnings.warn(msg, category=FutureWarning)
```




```
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       123
           1       0.98      0.89      0.94       123

    accuracy                           0.94       246
   macro avg       0.94      0.94      0.94       246
weighted avg       0.94      0.94      0.94       246

Accuracy Using Random Forest Classifier 93.90243902439023
```

```
print("Random forest classifier Accuracy : ", accuracy_score(y_pred,Y_test)*100)
print("Random forest classifier Precision : ", precision_score(y_pred,Y_test,average = 'weighted')*100)
print("Random forest classifier Recall : ", recall_score(y_pred,Y_test,average = 'weighted')*100)
```

```
Random forest classifier Accuracy :  93.90243902439023
Random forest classifier Precision :  94.3023332672351
Random forest classifier Recall :  93.90243902439023
```

Summary

Comparison Of Algorithms

S.No	Algorithms	Accuracy in %	Precision in %	Recall in %
1	K Nearest Neighbors	60	60	60
2	Logistic Regression	92	60	60
3	Decision Tree	91	91	91
4	Bagging Classifier	93	94	93
5	Random Forest Classifier	93	94	93

So in order to Conclude, It is identified that bagging classifier and Random forest performs well as it was able to tell that how many times the ML model was correct overall, how good the model is at predicting a specific category and how many times the model was able to detect a specific category where as in Logistic regression the model wasnot performed very well at predicting and detecting number of times the specific category.

Random Forest Classifier/Bagging Classifier > Decision Tree > Logistic Regression > K Nearest Neighbor