Diabetes Prediction Using Decision Tree and Multinomial Naive Bayes Classifiers

June 21, 2024

1 PIMA Indian Diabetes Prediction Analysis Using Decision Tree and Multinomial Naive Bayes Classifiers

2 Import the Required Libraries

[83]:
```
# Data analysis and wrangling
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree
 ↪Classifier
from sklearn.model_selection import train_test_split # Import train_test_split
 ↪function
from sklearn import metrics #Import scikit-learn metrics module for accuracy
 ↪calculation
from sklearn.metrics import confusion_matrix,classification_report

# suppress the warning adding the following lines to the imports of your program
import warnings
#warnings.simplefilter(action='ignore', category=FutureWarning)
# ignore all warnings
warnings.filterwarnings('ignore')

# sns.set(rc={'figure.figsize': [20, 20]}, font_scale=1.4)
sns.set_theme(color_codes=True)
```

3 Load the Dataset

[84]:
```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree',␣
↪'age', 'label']
# load dataset
pima = pd.read_csv("data/pima-indians-diabetes.csv", header=None,␣
↪names=col_names)
```

[85]:
```
# let's look at the datatype of each column
pima.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pregnant  768 non-null    int64
 1   glucose   768 non-null    int64
 2   bp        768 non-null    int64
 3   skin      768 non-null    int64
 4   insulin   768 non-null    int64
 5   bmi       768 non-null    float64
 6   pedigree  768 non-null    float64
 7   age       768 non-null    int64
 8   label     768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

[86]:
```
#let's see if any column has any null values
pima.isna().sum()
```

[86]:
```
pregnant    0
glucose     0
bp          0
skin        0
insulin     0
bmi         0
pedigree    0
age         0
label       0
dtype: int64
```

4 Feature Selection

[87]:
```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
```

```
y = pima.label # Target variable
```

5 Split the Dataset

[88]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=1) # 70% training and 30% test
```

6 Correlation Heatmap for the Dataset

A correlation heatmap is a graphical tool that displays the correlation between multiple variables as a color-coded matrix. It's like a color chart that shows us how closely related different variables are.

In a correlation heatmap, each variable is represented by a row and a column, and the cells show the correlation between them. The color of each cell represents the strength and direction of the correlation, with darker colors indicating stronger correlations.

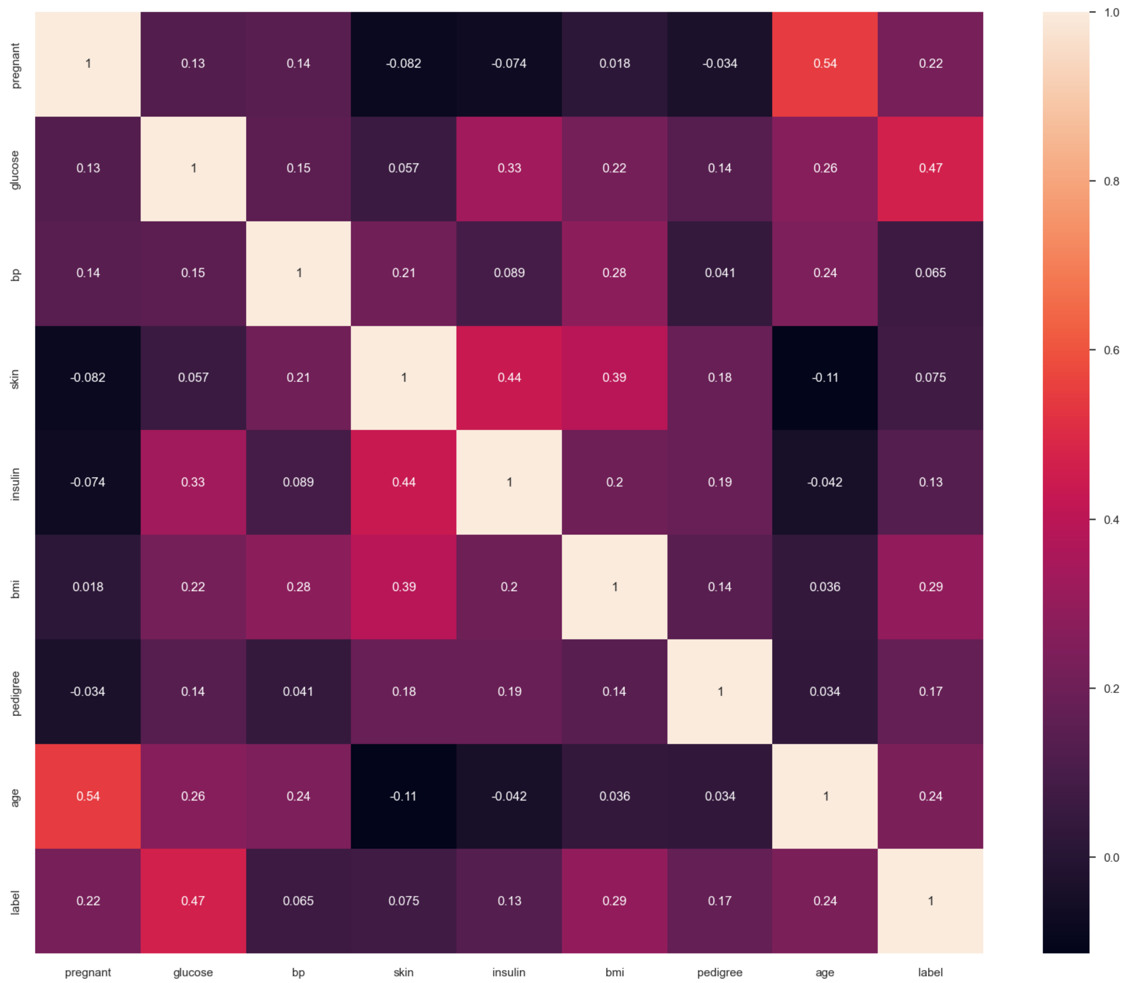
How to Read a Correlation Heatmap? In this section, we will delve into how to read a correlation heatmap, an effective visual tool for discerning the strength and direction of relationships between variables:

- Look at the color of each cell to see the strength and direction of the correlation.
- Darker colors indicate stronger correlations, while lighter colors indicate weaker correlations.
- Positive correlations (when one variable increases, the other variable tends to increase) are usually represented by warm colors, such as red or orange.
- Negative correlations (when one variable increases, the other variable tends to decrease) are usually represented by cool colors, such as blue or green.

Understanding correlation heatmaps can help us identify patterns and relationships between multiple variables. So next time you analyze data with many variables, think like an artist and use a correlation heatmap to see the colors of the relationships!

[89]:
```
# Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(pima.corr(), fmt='.2g', annot=True)
```

[89]: <Axes: >



7 1. Decision Tree Classifier in Scikit-learn

8 Bulid the Decision Tree Model

[90]:
```
# Create Decision Tree classifer object
dtree = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
dtree.fit(X_train,y_train)

# Predict the response for test dataset
y_pred = dtree.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy of Decision Tree Classifier : ",metrics.accuracy_score(y_test,␣
↪y_pred)*100)
```

```
Accuracy of Decision Tree Classifier :  77.05627705627705
```

9 Evaluate the Decision Tree Model

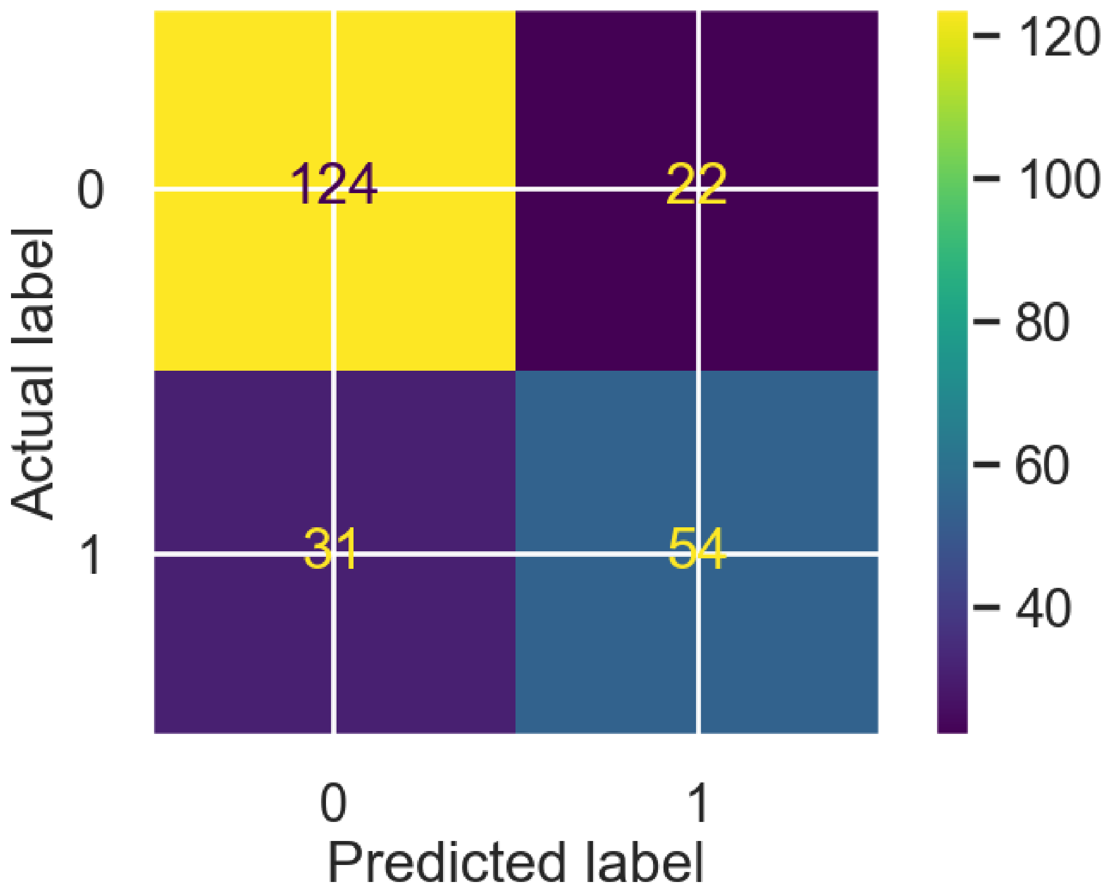
10 Confusion Matrix and computing Sensitivity and Specificity

[91]:
```python
cm_dtree = confusion_matrix(y_test, y_pred)

# Calculating accuracy, sensitivity and specificity from confusion matrix
total_dtree=sum(sum(cm_dtree))
accuracy_dtree=(cm_dtree[0,0]+cm_dtree[1,1])/total_dtree
print('Accuracy : ', accuracy_dtree)
sensitivity_dtree = cm_dtree[0,0]/(cm_dtree[0,0]+cm_dtree[0,1])
print('Sensitivity : ', sensitivity_dtree)
specificity_dtree = cm_dtree[1,1]/(cm_dtree[1,0]+cm_dtree[1,1])
print('Specificity : ', specificity_dtree)


tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

#tn = cm_dtree[0][0]
#fn = cm_dtree[1][0]
#tp = cm_dtree[1][1]
#fp = cm_dtree[0][1]

print("True Negative = ",tn)
print("False Negative = ",fn)
print("True Positive = ",tp)
print("False Positive = ",fp)

print(classification_report(y_test, y_pred, zero_division=1))
sns.set_context ("poster")
# Confusion matrix and derived metrics Display - New Format
plt.figure(figsize=(5,5))
from sklearn.metrics import ConfusionMatrixDisplay
_ = ConfusionMatrixDisplay.from_estimator(dtree, X_test , y_test)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
#all_sample_title = 'Accuracy Score for Decision Tree Classifier is {0}'.
 ↪format(dtree.score(X_test, y_test))
#plt.title(all_sample_title, size = 15)
plt.show()
```

```
Accuracy :  0.7705627705627706
Sensitivity :  0.8493150684931506
Specificity :  0.6352941176470588
True Negative =  124
```

```
False Negative =  31
True Positive =  54
False Positive =  22
              precision    recall  f1-score   support

           0       0.80      0.85      0.82       146
           1       0.71      0.64      0.67        85

    accuracy                           0.77       231
   macro avg       0.76      0.74      0.75       231
weighted avg       0.77      0.77      0.77       231
```

```
<Figure size 500x500 with 0 Axes>
```



11 Compute the other measures or metrics for the Decision Tree Model

[92]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_percentage_error,␣
↪mean_absolute_error, mean_squared_error, r2_score
import math
y_pred = dtree.predict(X_test)
dtree_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score of Decision Tree Classifier is ",␣
↪round(dtree_accuracy*100 ,2), "%")
print("Accuracy Score of Decision Tree Classifier is ", dtree_accuracy)

#Print the training score and test score
dtree_training_score=dtree.score(X_train,y_train)
print(f"Training Score of Decision Tree Classifier:{dtree_training_score*100}%")
dtree_test_score=dtree.score(X_test,y_test)
print(f"Test score of Decision Tree Classifier:{dtree_test_score*100}%")
```

```
Accuracy Score of Decision Tree Classifier is  77.06 %
Accuracy Score of Decision Tree Classifier is  0.7705627705627706
Training Score of Decision Tree Classifier:76.35009310986965%
Test score of Decision Tree Classifier:77.05627705627705%
```

[93]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score, jaccard_score, log_loss

dtree_f1_score = f1_score(y_test, y_pred, average='micro')
dtree_precision_score = precision_score(y_test, y_pred, average='micro')
dtree_recall_score = recall_score(y_test, y_pred, average='micro')
dtree_jaccard_score = jaccard_score(y_test, y_pred, average='micro')
dtree_log_loss = log_loss(y_test, y_pred)
print('Scores Calculation using average: micro')
print('========================================')
print('F-1 Score of Decision Tree Classifier is ', dtree_f1_score)
print('Precision Score of Decision Tree Classifier is ', dtree_precision_score)
print('Recall Score of Decision Tree Classifier is ', dtree_recall_score)
print('Jaccard Score of Decision Tree Classifier is ', dtree_jaccard_score)
print('Log Loss of Decision Tree Classifier is ', dtree_log_loss)

# Manual Caluclation
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
dtree_accuracy_calculated = (tp + tn) / (tp + fp + fn + tn)
dtree_precision_calculated = tp / (tp + fp)
dtree_recall_calculated = tp / (tp + fn)
```

```
dtree_f1_score_calculated = 2 * ( (dtree_precision_calculated *␣
 ↪dtree_recall_calculated) / (dtree_precision_calculated +␣
 ↪dtree_recall_calculated) )
print('Scores Calculation using Manual Method')
print('========================================')
print('F-1 Score of Decision Tree Classifier is ', dtree_f1_score_calculated)
print('Precision Score of Decision Tree Classifier is ',␣
 ↪dtree_precision_calculated)
print('Recall Score of Decision Tree Classifier is ', dtree_recall_calculated)
print("Accuracy Score of Decision Tree Classifier is ",␣
 ↪dtree_accuracy_calculated)
```

```
Scores Calculation using average: micro
========================================
F-1 Score of Decision Tree Classifier is  0.7705627705627706
Precision Score of Decision Tree Classifier is  0.7705627705627706
Recall Score of Decision Tree Classifier is  0.7705627705627706
Jaccard Score of Decision Tree Classifier is  0.6267605633802817
Log Loss of Decision Tree Classifier is  8.269755972394846
Scores Calculation using Manual Method
========================================
F-1 Score of Decision Tree Classifier is  0.6708074534161491
Precision Score of Decision Tree Classifier is  0.7105263157894737
Recall Score of Decision Tree Classifier is  0.6352941176470588
Accuracy Score of Decision Tree Classifier is  0.7705627705627706
```

12 Visualize the Decision Tree

[94]:
```
#! pip install graphviz
```

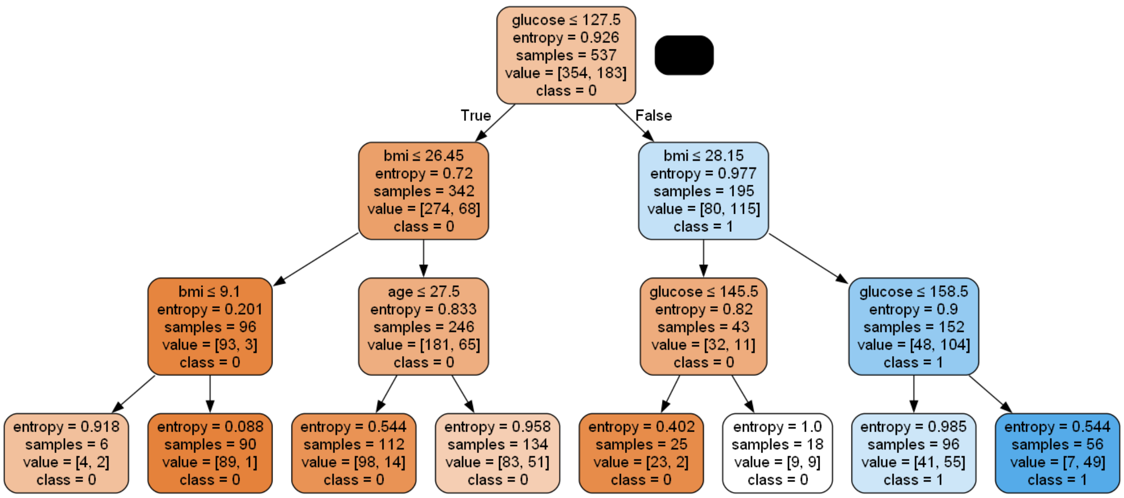
[95]:
```
#! pip install pydotplus
```

12.1 export_graphviz function converts decision tree classifier into dot file and pydotplus convert this dot file to png or displayable form on Jupyter.

[96]:
```
from sklearn.tree import export_graphviz
#from sklearn.externals.six import StringIO
from six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(dtree, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names =␣
 ↪feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
```

```
graph.write_png('diabetes.png')
Image(graph.create_png())
```

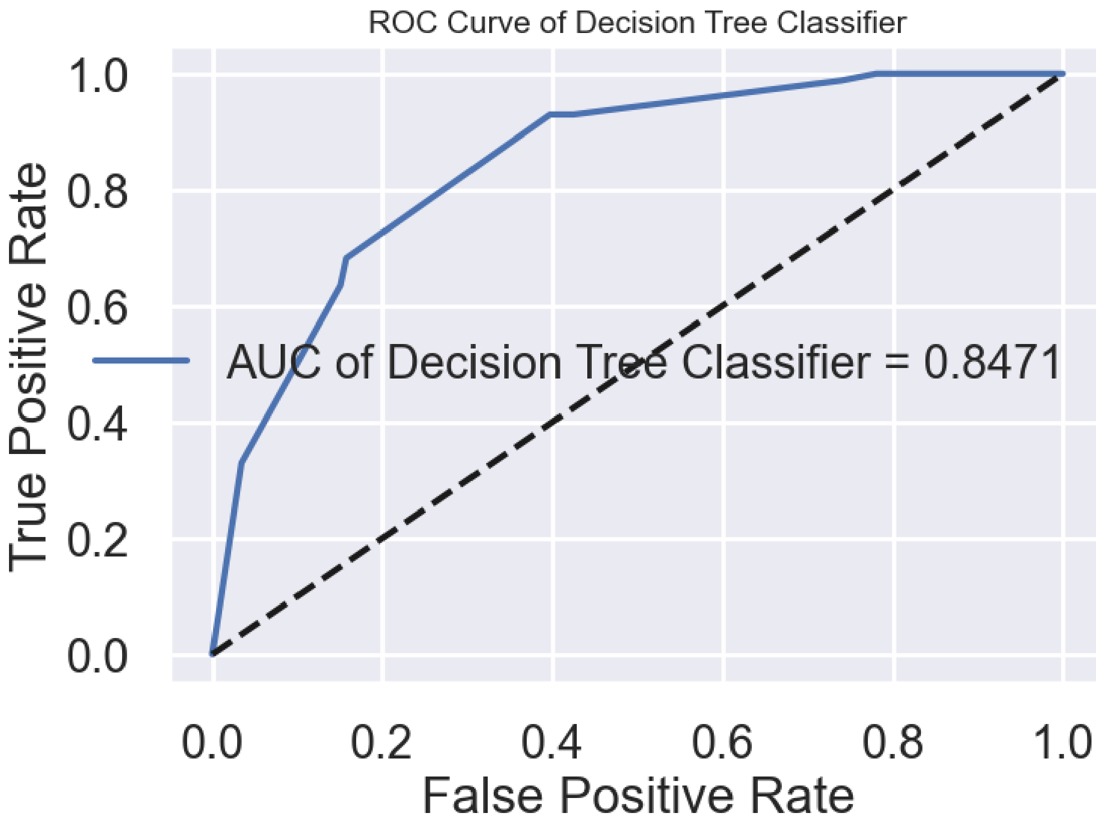
[96]:



13 Plot ROC Cure of the Decision Tree Model

[97]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test),␣
  ↪columns=['y_actual']), pd.DataFrame(y_pred_proba,␣
  ↪columns=['y_pred_proba'])], axis=1)
df_actual_predicted.index = y_test.index

# Calculate and print the ROC (Receiver Operating Characteristic) curve and AUC␣
  ↪(Area under the ROC Curve)
fpr_dtree, tpr_dtree, tr_dtree = roc_curve(df_actual_predicted['y_actual'],␣
  ↪df_actual_predicted['y_pred_proba'])
auc_dtree = roc_auc_score(df_actual_predicted['y_actual'],␣
  ↪df_actual_predicted['y_pred_proba'])

plt.plot(fpr_dtree, tpr_dtree, label='AUC of Decision Tree Classifier = %0.4f'␣
  ↪%auc_dtree)
plt.plot(fpr_dtree, fpr_dtree, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve of Decision Tree Classifier', size = 15)
plt.legend()
```

[97]: <matplotlib.legend.Legend at 0x19ef8551510>



14 2. Multinomial Naive Bayes Classifier in Scikit-learn

15 Bulid the Multinomial Naive Bayes Model

[98]:
```
from sklearn.naive_bayes import MultinomialNB
# Create Multinomial Naive Bayes classifer object
mnb = MultinomialNB()

# Train Multinomial Naive Bayes Classifer
mnb.fit(X_train,y_train)

# Predict the response for test dataset
y_pred = mnb.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy of Multinomial Naive Bayes Classifier : ",metrics.
 ↪accuracy_score(y_test, y_pred)*100)
```

```
Accuracy of Multinomial Naive Bayes Classifier :  55.84415584415584
```

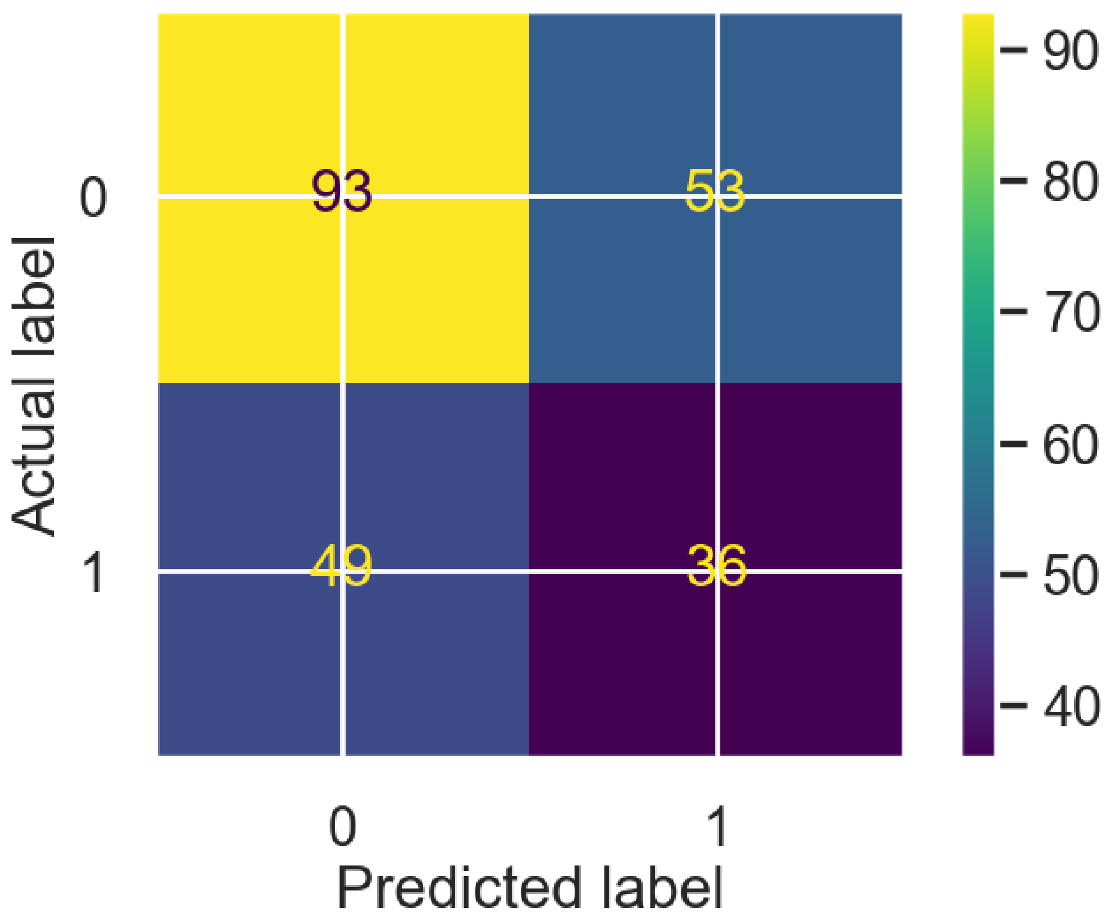
16 Evaluate the Multinomial Naive Bayes Model

17 Confusion Matrix and computing Sensitivity and Specificity

[99]:
```
cm_mnb = confusion_matrix(y_test, y_pred)

# Calculating accuracy, sensitivity and specificity from confusion matrix
total_mnb=sum(sum(cm_mnb))
accuracy_mnb=(cm_mnb[0,0]+cm_mnb[1,1])/total_mnb
print('Accuracy : ', accuracy_mnb)
sensitivity_mnb = cm_mnb[0,0]/(cm_mnb[0,0]+cm_mnb[0,1])
print('Sensitivity : ', sensitivity_mnb)
specificity_mnb = cm_mnb[1,1]/(cm_mnb[1,0]+cm_mnb[1,1])
print('Specificity : ', specificity_mnb)


tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()

#tn = cm_mnb[0][0]
#fn = cm_mnb[1][0]
#tp = cm_mnb[1][1]
#fp = cm_mnb[0][1]

print("True Negative = ",tn)
print("False Negative = ",fn)
print("True Positive = ",tp)
print("False Positive = ",fp)

print(classification_report(y_test, y_pred, zero_division=1))
sns.set_context ("poster")
# Confusion matrix and derived metrics Display - New Format
plt.figure(figsize=(5,5))
from sklearn.metrics import ConfusionMatrixDisplay
_ = ConfusionMatrixDisplay.from_estimator(mnb, X_test , y_test)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
#all_sample_title = 'Accuracy Score for Multinomial Naive Bayes Classifier is␣
↪{0}'.format(mnb.score(X_test, y_test))
#plt.title(all_sample_title, size = 15)
plt.show()
```

```
Accuracy :  0.5584415584415584
Sensitivity :  0.636986301369863
Specificity :  0.4235294117647059
True Negative =  93
False Negative =  49
True Positive =  36
False Positive =  53
```

```
              precision    recall  f1-score   support

           0       0.65      0.64      0.65       146
           1       0.40      0.42      0.41        85

    accuracy                           0.56       231
   macro avg       0.53      0.53      0.53       231
weighted avg       0.56      0.56      0.56       231

<Figure size 500x500 with 0 Axes>
```



18 Compute the other measures for the Multinomial Naive Bayes Model

[100]:
```
from sklearn.metrics import mean_absolute_percentage_error,␣
↪mean_absolute_error, mean_squared_error, r2_score
import math
y_pred = mnb.predict(X_test)
mnb_accuracy = accuracy_score(y_test, y_pred)
print("Accuracy Score of Multinomial Naive Bayes Classifier is ",␣
↪round(mnb_accuracy*100 ,2), "%")
print("Accuracy Score of Multinomial Naive Bayes Classifier is ", mnb_accuracy)

#Print the training score and test score
mnb_training_score=mnb.score(X_train,y_train)
print(f"Training Score of Multinomial Naive Bayes Classifier:
↪{mnb_training_score*100}%")
mnb_test_score=mnb.score(X_test,y_test)
print(f"Test score of Multinomial Naive Bayes Classifier:{mnb_test_score*100}%")
```

```
Accuracy Score of Multinomial Naive Bayes Classifier is  55.84 %
Accuracy Score of Multinomial Naive Bayes Classifier is  0.5584415584415584
Training Score of Multinomial Naive Bayes Classifier:60.33519553072626%
Test score of Multinomial Naive Bayes Classifier:55.84415584415584%
```

[101]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score, jaccard_score, log_loss
mnb_f1_score = f1_score(y_test, y_pred, average='macro')
mnb_precision_score = precision_score(y_test, y_pred, average='macro')
mnb_recall_score = recall_score(y_test, y_pred, average='macro')
mnb_jaccard_score = jaccard_score(y_test, y_pred, average='macro')
mnb_log_loss = log_loss(y_test, y_pred)

mnb_f1_score = f1_score(y_test, y_pred, average='micro')
mnb_precision_score = precision_score(y_test, y_pred, average='micro')
mnb_recall_score = recall_score(y_test, y_pred, average='micro')
mnb_jaccard_score = jaccard_score(y_test, y_pred, average='micro')
mnb_log_loss = log_loss(y_test, y_pred)
print('Scores Calculation using average: micro')
print('========================================')
print('F-1 Score of Multinomial Naive Bayes Classifier is ', mnb_f1_score)
print('Precision Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_precision_score)
print('Recall Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_recall_score)
print('Jaccard Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_jaccard_score)
print('Log Loss of Multinomial Naive Bayes Classifier is ', mnb_log_loss)
```

```
# Manual Caluclation
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
mnb_accuracy_calculated = (tp + tn) / (tp + fp + fn + tn)
mnb_precision_calculated = tp / (tp + fp)
mnb_recall_calculated = tp / (tp + fn)
mnb_f1_score_calculated = 2 * ( (mnb_precision_calculated *␣
↪mnb_recall_calculated) / (mnb_precision_calculated + mnb_recall_calculated) )
print('Scores Calculation using Manual Method')
print('========================================')
print('F-1 Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_f1_score_calculated)
print('Precision Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_precision_calculated)
print('Recall Score of Multinomial Naive Bayes Classifier is ',␣
↪mnb_recall_calculated)
print("Accuracy Score of Multinomial Naive Bayes Classifier is ",␣
↪mnb_accuracy_calculated)
```

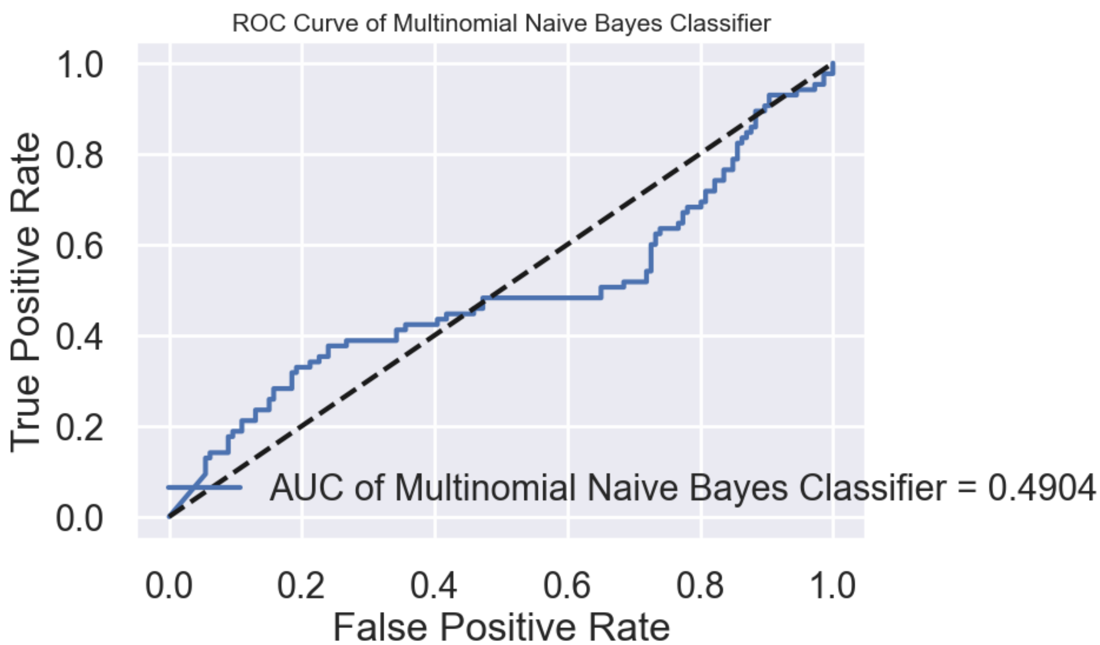
```
Scores Calculation using average: micro
========================================
F-1 Score of Multinomial Naive Bayes Classifier is  0.5584415584415584
Precision Score of Multinomial Naive Bayes Classifier is  0.5584415584415584
Recall Score of Multinomial Naive Bayes Classifier is  0.5584415584415584
Jaccard Score of Multinomial Naive Bayes Classifier is  0.38738738738738737
Log Loss of Multinomial Naive Bayes Classifier is  15.915379418571211
Scores Calculation using Manual Method
========================================
F-1 Score of Multinomial Naive Bayes Classifier is  0.41379310344827586
Precision Score of Multinomial Naive Bayes Classifier is  0.4044943820224719
Recall Score of Multinomial Naive Bayes Classifier is  0.4235294117647059
Accuracy Score of Multinomial Naive Bayes Classifier is  0.5584415584415584
```

19 Plot ROC Cure of the Multinomial Naive Bayes Model

[102]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = mnb.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test),␣
↪columns=['y_actual']), pd.DataFrame(y_pred_proba,␣
↪columns=['y_pred_proba'])], axis=1)
df_actual_predicted.index = y_test.index

# Calculate and print the ROC (Receiver Operating Characteristic) curve and AUC␣
↪(Area under the ROC Curve)
```

```
fpr_mnb, tpr_mnb, tr_mnb = roc_curve(df_actual_predicted['y_actual'],␣
↪df_actual_predicted['y_pred_proba'])
auc_mnb = roc_auc_score(df_actual_predicted['y_actual'],␣
↪df_actual_predicted['y_pred_proba'])

plt.plot(fpr_mnb, tpr_mnb, label='AUC of Multinomial Naive Bayes Classifier =␣
↪%0.4f' %auc_mnb)
plt.plot(fpr_mnb, fpr_mnb, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve of Multinomial Naive Bayes Classifier', size = 15)
plt.legend()
```

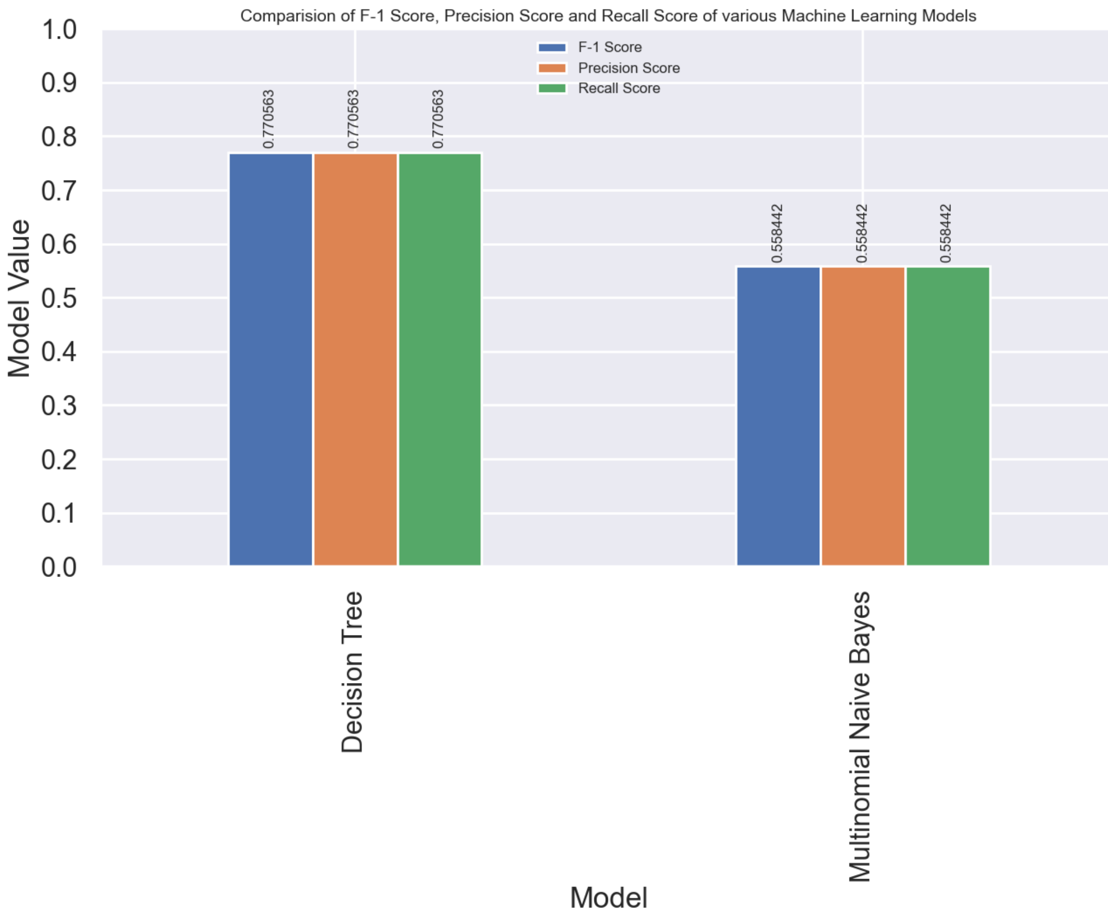
[102]: <matplotlib.legend.Legend at 0x19ef8afa350>



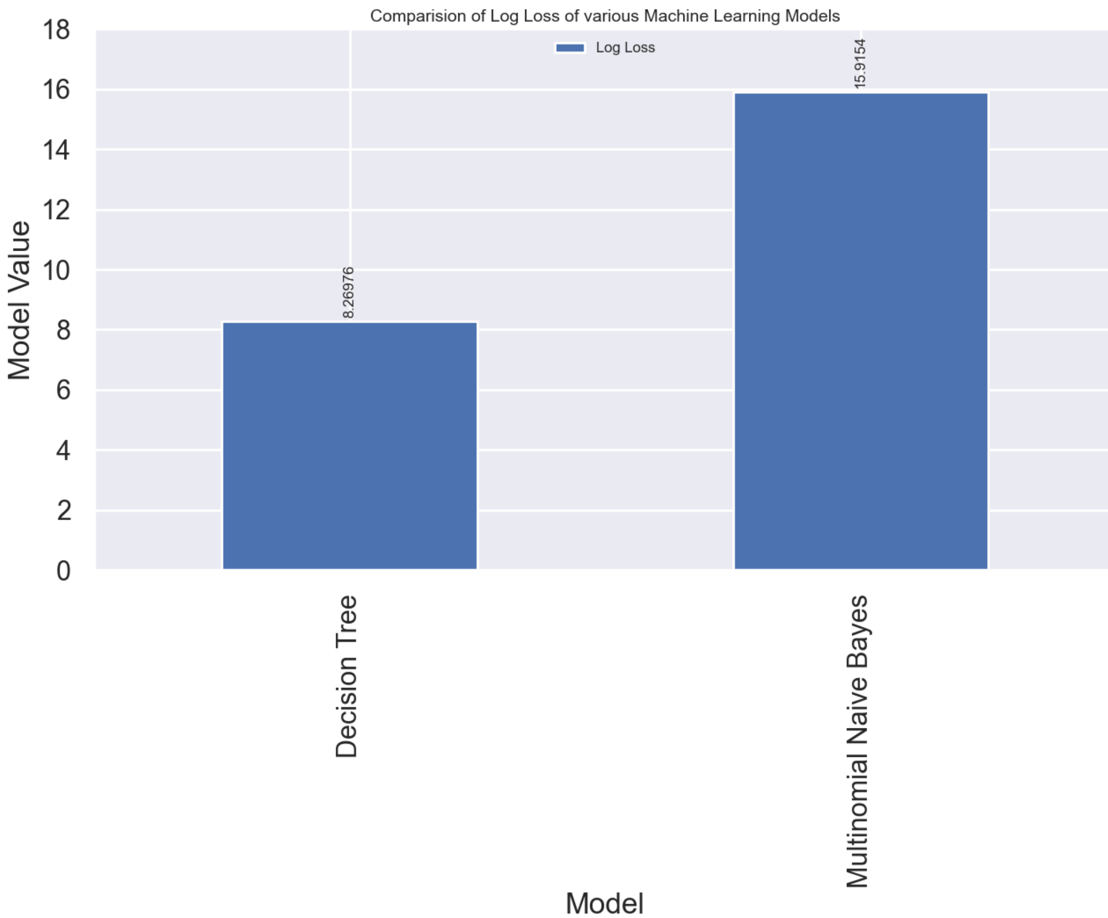
[103]:
```
models = pd.DataFrame({
    'Model': ["Decision Tree","Multinomial Naive Bayes"],
    'F-1 Score': [dtree_f1_score, mnb_f1_score],
    'Precision Score': [dtree_precision_score, mnb_precision_score],
    'Recall Score': [dtree_recall_score, mnb_recall_score],
    'Log Loss': [dtree_log_loss, mnb_log_loss]})
sorted_models=models.sort_values(by=['F-1 Score'], ascending=False)
sorted_models
```

[103]:
```
                   Model  F-1 Score  Precision Score  Recall Score  \
0          Decision Tree   0.770563         0.770563      0.770563
1  Multinomial Naive Bayes   0.558442         0.558442      0.558442

    Log Loss
0   8.269756
1  15.915379
```

[105]:
```
# Plotting the graph
axs = pd.concat([models['F-1 Score'],models['Precision Score'],models['Recall␣
↪Score']], axis=1).plot.bar(figsize=(15, 8))

axs.set_title('Comparision of F-1 Score, Precision Score and Recall Score of␣
↪various Machine Learning Models', fontsize=14)
axs.set_xlabel('Model')
axs.set_ylabel('Model Value')

axs.set_xticklabels(models['Model'])

for container in axs.containers:
    axs.bar_label(container, padding=3, rotation=90, fontsize=12)

plt.yticks(np.arange(0, 1.1, step=0.1))
#axs.legend(loc='upper center', ncols=3)
axs.legend(loc='upper center',fontsize=12)

plt.show()
```



[106]:
```python
# Plotting the graph
axs = pd.concat([models['Log Loss']], axis=1).plot.bar(figsize=(15, 8))

axs.set_title('Comparision of Log Loss of various Machine Learning␣
↪Models',fontsize=14)
axs.set_xlabel('Model')
axs.set_ylabel('Model Value')

axs.set_xticklabels(models['Model'])

for container in axs.containers:
    axs.bar_label(container, padding=3, rotation=90, fontsize=12)

plt.yticks(np.arange(0, 20, step=2))
#axs.legend(loc='upper center', ncols=3)
axs.legend(loc='upper center',fontsize=12)

plt.show()
```



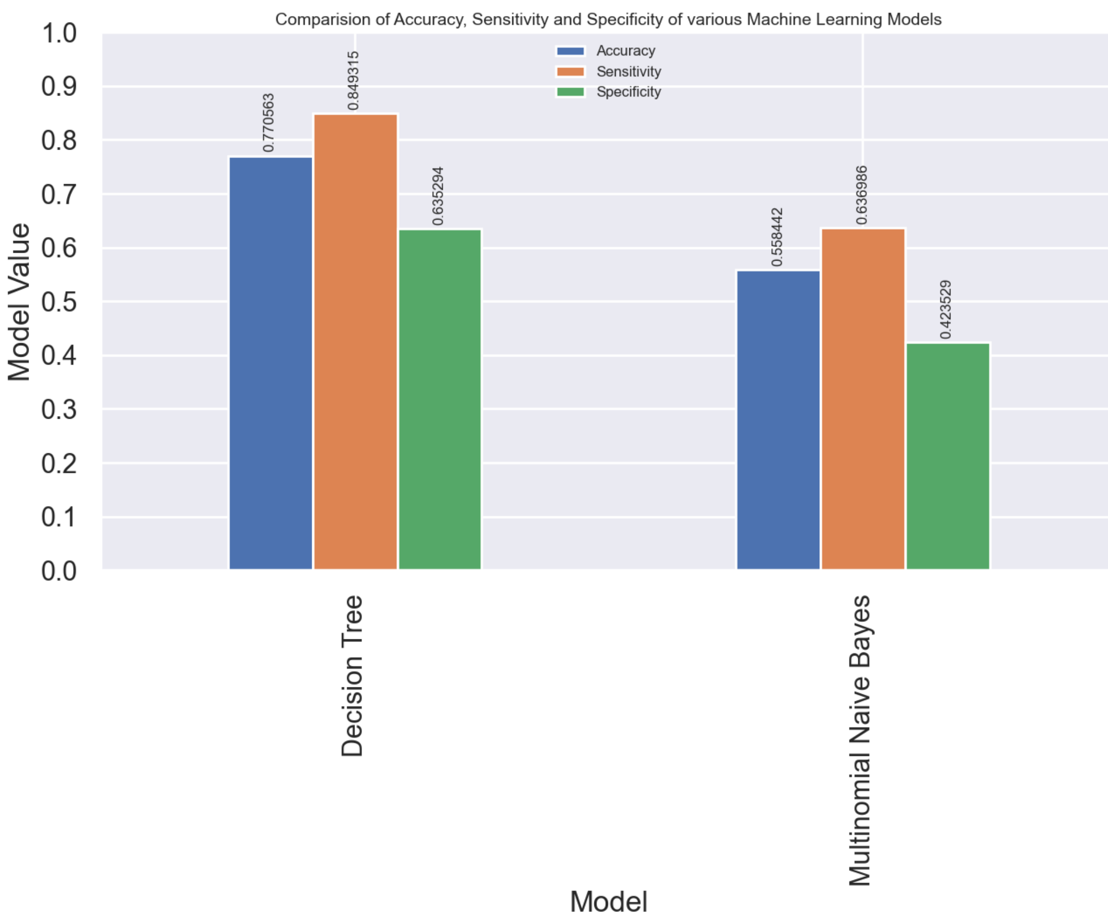
[107]:
```
models = pd.DataFrame({
    'Model': ["Decision Tree", "Multinomial Naive Bayes"],
    'Accuracy': [accuracy_dtree, accuracy_mnb],
    'Sensitivity': [sensitivity_dtree, sensitivity_mnb],
    'Specificity': [specificity_dtree, specificity_mnb]})
sorted_models=models.sort_values(by=['Accuracy'], ascending=False)
sorted_models
```

[107]:

	Model	Accuracy	Sensitivity	Specificity
0	Decision Tree	0.770563	0.849315	0.635294
1	Multinomial Naive Bayes	0.558442	0.636986	0.423529

[109]:
```
# Plotting the graph
axs = pd.
 ↪concat([models['Accuracy'],models['Sensitivity'],models['Specificity']],␣
 ↪axis=1).plot.bar(figsize=(15, 8))
```

```
axs.set_title('Comparision of Accuracy, Sensitivity and Specificity of various Machine Learning Models', fontsize=14)
axs.set_xlabel('Model')
axs.set_ylabel('Model Value')

axs.set_xticklabels(models['Model'])

for container in axs.containers:
    axs.bar_label(container, padding=3, rotation=90, fontsize=12)

plt.yticks(np.arange(0, 1.1, step=0.1))
#axs.legend(loc='upper center', ncols=3)
axs.legend(loc='upper center',fontsize=12)

plt.show()
```



20 Results and Discussion

- The Decision Tree is predicting more accurately the diabetes than the Multinomial Naive Bayes and its accuracy is 77.05%
- The AUC value of Decision Tree is 0.847 which is higher than the AUC value of Multinomial Naive Bayes.