learning-and-deep-learning-models

June 28, 2024

1 Titanic Survival Prediction: A Comparative Analysis of Machine Learning and Deep Learning Models

Dr. DILEEP KUMAR SHETTY (Ph.D. ,M.Sc with KSET & Data Science)

2 ABSTRACT:

This project predicted Titanic passenger survival using Logistic Regression, Decision Tree, Naïve Bayes, and Artificial Neural Network (ANN) models. ANN demonstrated superior performance among the models. Exploratory Data Analysis revealed higher survival rates for Class 1 passengers, females, aged individuals, and children. The study successfully applied both machine learning and deep learning models to predict survival outcomes. It also identified significant survival patterns, enhancing understanding of the Titanic dataset.

3 ABOUT DATA:

The data set is a subset of a Titanic passenger dataset. Here's a detailed explanation of each column in the dataset: 1. Name: This column contains the names of the passengers. It is of type object, indicating that it contains string data. 2. survived: This column indicates whether a passenger survived or not. It is of type object, which means it likely contains categorical string data such as "yes" or "no". 3. sex: This column denotes the gender of the passengers. It is of type object, containing string data such as "male" or "female". 4. age: This column contains the ages of the passengers. It is of type float64, indicating that it contains numerical data. There are some missing values in this column, as indicated by the count of non-null values (1046 out of 1309). 5. passengerClass: This column indicates the class in which the passenger was traveling (e.g., First, Second, or Third class). It is of type object, containing string data.

4 PROJECT OBJECTIVES

1. Predict Titanic Passenger Survival: Develop predictive models to estimate the likelihood of survival for passengers aboard the Titanic.
2. Comparative Analysis of Models: Evaluate and compare the performance of various machine learning and deep learning models including Logistic Regression, Decision Tree, Naïve Bayes, and Artificial Neural Network (ANN).
3. Exploratory Data Analysis (EDA): Conduct a thorough EDA to uncover significant patterns and relationships in the Titanic dataset, particularly focusing on factors that influenced survival rates.

4. Model Performance Evaluation: Assess the accuracy and effectiveness of each predictive model using appropriate metrics to determine the best performing model.
5. Identification of Key Survival Factors: Identify and interpret significant factors influencing survival rates, such as passenger class, gender, age, and other relevant features.

5 PROJECT OUTCOMES

1. Successful Survival Prediction Models: Developed and tested multiple models for predicting survival, with the ANN model demonstrating superior performance compared to traditional machine learning models.
2. Performance Metrics: The ANN model outperformed others in key metrics such as accuracy, precision, recall, and F1 score, indicating its effectiveness in predicting survival outcomes.
3. Insights from EDA: The EDA revealed critical insights: o Higher survival rates were observed among Class 1 passengers. o Females had a significantly higher survival rate compared to males. o Children and aged individuals showed higher chances of survival.
4. Enhanced Understanding of the Dataset: The project provided a comprehensive understanding of the factors that affected survival on the Titanic, contributing valuable insights into historical data analysis.
5. Practical Application of ML and DL Models: Successfully applied both machine learning and deep learning techniques to a real-world dataset, showcasing the practical applications and benefits of these methods in predictive analytics.
6. Significant Survival Patterns: Identified and confirmed significant survival patterns which could be useful for historical analysis and for developing similar predictive models in other domains. By achieving these objectives and outcomes, the project not only demonstrated the practical application of predictive models but also contributed to the deeper understanding of the Titanic disaster through data analysis.

6 Libraries and modules commonly used in data analysis and machine learning in Python

[1]:
```python
#Pandas is a powerful data manipulation library for Python.
import pandas as pd

#NumPy is a numerical computing library for Python.
import numpy as np

#Matplotlib is a plotting library for creating static, interactive, and␣
↪animated visualizations in Python.
import matplotlib.pyplot as plt

#ListedColormap is a class in Matplotlib used to create a colormap from a list␣
↪of colors.
from matplotlib.colors import ListedColormap

#Seaborn is a statistical data visualization library based on Matplotlib.
import seaborn as sns
```

```
#is_string_dtype is a function from Pandas used to check if a dtype is of␣
↪object type.
from pandas.api.types import is_string_dtype

#StandardScaler is a preprocessing technique used to standardize features by␣
↪removing the mean and scaling to unit variance.
from sklearn.preprocessing import StandardScaler

#train_test_split is a function in scikit-learn used for splitting a dataset␣
↪into training and testing sets.
from sklearn.model_selection import train_test_split
```

[2]:
```
#The metrics module in scikit-learn provides various metrics for evaluating␣
↪model performance.
from sklearn import metrics

#LogisticRegression is a class in scikit-learn used for logistic regression␣
↪modeling.
from sklearn.linear_model import LogisticRegression

#classification_report is a function in scikit-learn that generates a text␣
↪report showing the main classification metrics.
from sklearn.metrics import classification_report

#cohen_kappa_score is a function in scikit-learn used for calculating the␣
↪Cohen's kappa statistic.
from sklearn.metrics import cohen_kappa_score

#confusion_matrix is a function in scikit-learn that computes the confusion␣
↪matrix to evaluate the accuracy of a classification.
from sklearn.metrics import confusion_matrix

#roc_auc_score is a function in scikit-learn used for computing the area under␣
↪the ROC AUC.
from sklearn.metrics import roc_auc_score

#roc_curve is a function in scikit-learn used for generating receiver operating␣
↪characteristic (ROC) curves.
from sklearn.metrics import roc_curve

#SGDClassifier is a class in scikit-learn implementing linear classifiers with␣
↪Stochastic Gradient Descent training.
from sklearn.linear_model import SGDClassifier
```

```
#DecisionTreeClassifier is a class in scikit-learn for building decision tree␣
 ↪models.
from sklearn.tree import DecisionTreeClassifier

#GridSearchCV is a class in scikit-learn for hyperparameter tuning using grid␣
 ↪search.
from sklearn.model_selection import GridSearchCV

#The tree module in scikit-learn provides tools for working with decision trees.
from sklearn import tree

#export_graphviz is a function in scikit-learn for exporting decision tree␣
 ↪models to Graphviz format.
from sklearn.tree import export_graphviz
```

[3]:
```
#Statsmodels is a library for estimating and testing statistical models.
import statsmodels
import statsmodels.api as sm

#SVC is a class in scikit-learn implementing Support Vector Classification.
from sklearn.svm import SVC

#GaussianNB is a class in scikit-learn implementing Gaussian Naive Bayes␣
 ↪classification.
from sklearn.naive_bayes import GaussianNB

#KNeighborsClassifier is a class in scikit-learn for k-nearest neighbors␣
 ↪classification.
from sklearn.neighbors import KNeighborsClassifier
```

[4]:
```
#Ignore Warnings:
import warnings
from warnings import filterwarnings
filterwarnings('ignore')

#Adjust Figure Size for Matplotlib:
plt.rcParams['figure.figsize'] = [10,4]
```

[5]:
```
#Adjusting some display and print options for Pandas and NumPy
#max_columns to None, Pandas not to truncate the display of columns.
pd.options.display.max_columns = None

##max_rows to None, Pandas not to truncate the display of rows.
pd.options.display.max_rows = None

# To see the full numeric values without exponential notation.
```

```
np.set_printoptions(suppress=True)
```

[6]:
```
import os
os.chdir("C:\DKS\Machine_Learning\Titanic_Project")
data = pd.read_csv('TitanicSurvival1.csv')
data.sample(5)
```

[6]:

	Name	survived	sex	age	passengerClass
508	Moraweck, Dr. Ernest	no	male	54.0	2nd
643	Asplund, Miss. Lillian Gertrud	yes	female	5.0	3rd
249	Ryerson, Master. John Borie	yes	male	13.0	1st
273	Spedden, Master. Robert Douglas	yes	male	6.0	1st
317	Williams, Mr. Richard Norris II	yes	male	21.0	1st

[8]:
```
data.dtypes
```

[8]:
```
Name              object
survived          object
sex               object
age              float64
passengerClass    object
dtype: object
```

[9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Name            1309 non-null   object
 1   survived        1309 non-null   object
 2   sex             1309 non-null   object
 3   age             1046 non-null   float64
 4   passengerClass  1309 non-null   object
dtypes: float64(1), object(4)
memory usage: 51.3+ KB
```

[10]:
```
data.describe().T
```

[10]:

	count	mean	std	min	25%	50%	75%	max
age	1046.0	29.881135	14.4135	0.1667	21.0	28.0	39.0	80.0

[11]:
```
Total_missing = data.isnull().sum().sort_values(ascending = False)
Total_missing
```

[11]:
```
age              263
Name               0
survived           0
sex                0
passengerClass     0
dtype: int64
```

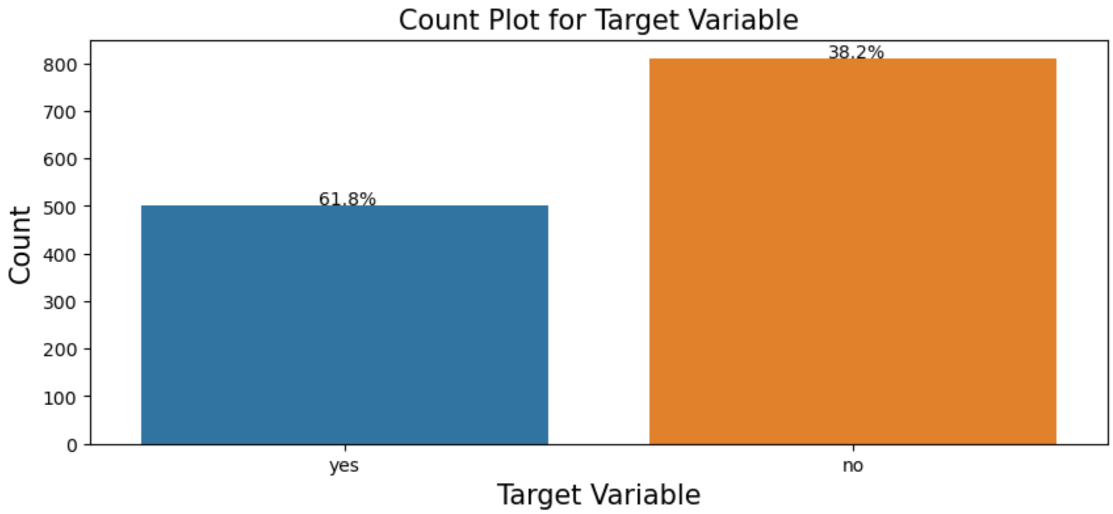
[12]:
```
data_x = data.iloc[:, data.columns != 'survived']
data_y = data.iloc[:,data.columns == 'survived']
print(data_y.head(2))
print(data_x.head(2))
```

```
  survived
0      yes
1      yes
                             Name     sex      age passengerClass
0   Allen, Miss. Elisabeth Walton  female  29.0000            1st
1  Allison, Master. Hudson Trevor    male   0.9167            1st
```

[13]:
```
class_frequency =data_y.value_counts()
class_frequency
```

[13]:
```
survived
no          809
yes         500
dtype: int64
```

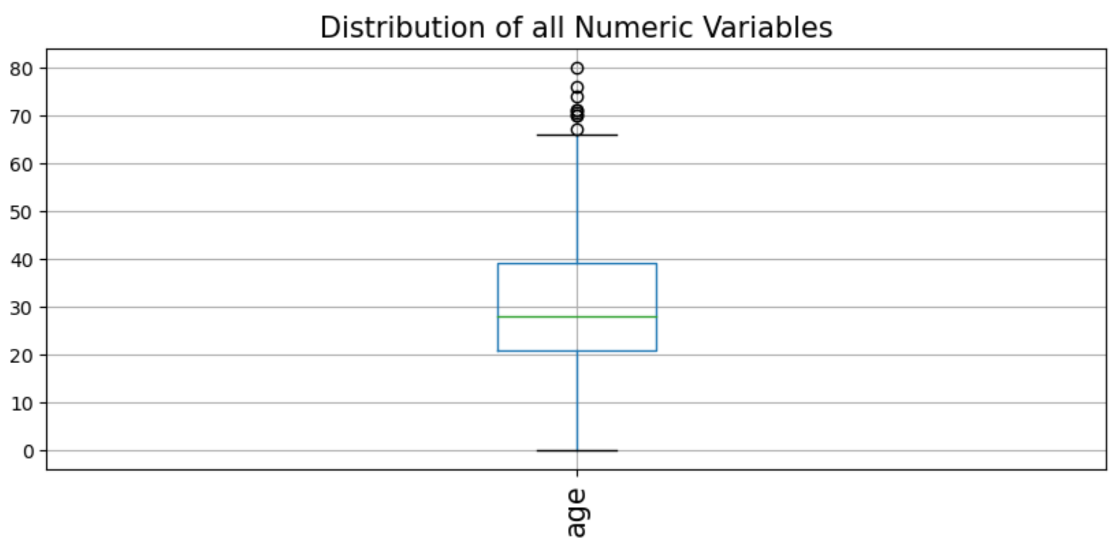
[14]:
```
sns.countplot(data=data_y,x ="survived")
plt.text(x = -0.05, y =data_y.value_counts()[1]+2, s =␣
 ↪str(round((class_frequency[0])*100/len(data_y),2)) + '%')
plt.text(x = 0.95, y =data_y.value_counts()[0]+2, s =␣
 ↪str(round((class_frequency[1])*100/len(data_y),2)) + '%')
plt.title('Count Plot for Target Variable', fontsize = 15)
plt.xlabel('Target Variable', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
```



[15]:
```
data.groupby(['sex', 'survived'])['survived'].count()
```

[15]:
```
sex     survived
female  no          127
        yes         339
male    no          682
        yes         161
Name: survived, dtype: int64
```

[16]:
```
data_x.boxplot()
plt.title('Distribution of all Numeric Variables', fontsize = 15)
plt.xticks(rotation = 'vertical', fontsize = 15)
plt.show()
```



[17]:
```
Q1 = data_x.quantile(0.25)
Q3 = data_x.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
age    18.0
dtype: float64
```

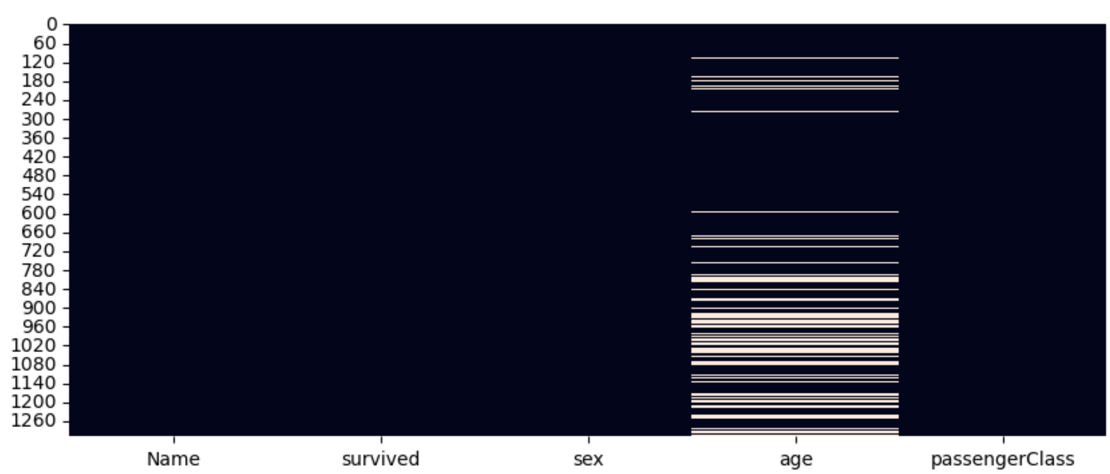
[18]:
```
#data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).
↪any(axis=1)]
#data = data.reset_index(drop = True)
#data
```

[19]:
```
Total = data.isnull().sum().sort_values(ascending = False)
Percentage = (data.isnull().sum()*100/data.isnull().count()).
↪sort_values(ascending = False)
Missing_Values = pd.concat([Total, Percentage], axis = 1, keys = ['Total',␣
↪'Percentage of missing observations'])
Missing_Values
```

[19]:

	Total	Percentage of missing observations
age	263	20.091673
Name	0	0.000000
survived	0	0.000000
sex	0	0.000000
passengerClass	0	0.000000

[20]:
```
sns.heatmap(data.isnull(), cbar=False)
plt.show()
```



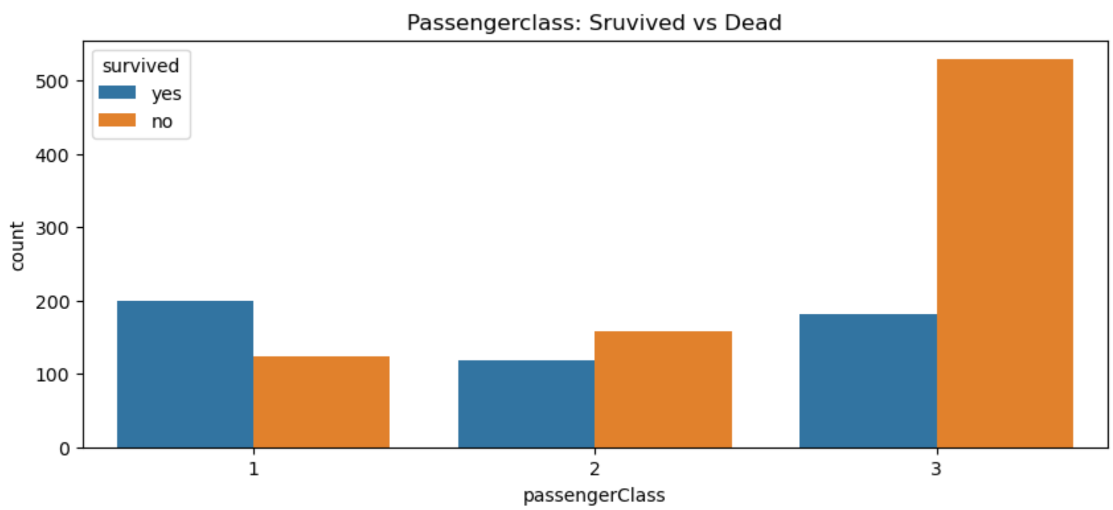
[21]:
```
#Passengenrs categories: coding
data.replace(to_replace='1st',value='1',inplace=True)
data.replace(to_replace='2nd',value='2',inplace=True)
data.replace(to_replace='3rd',value='3',inplace=True)
```

[22]:
```
data.head()
```

[22]:

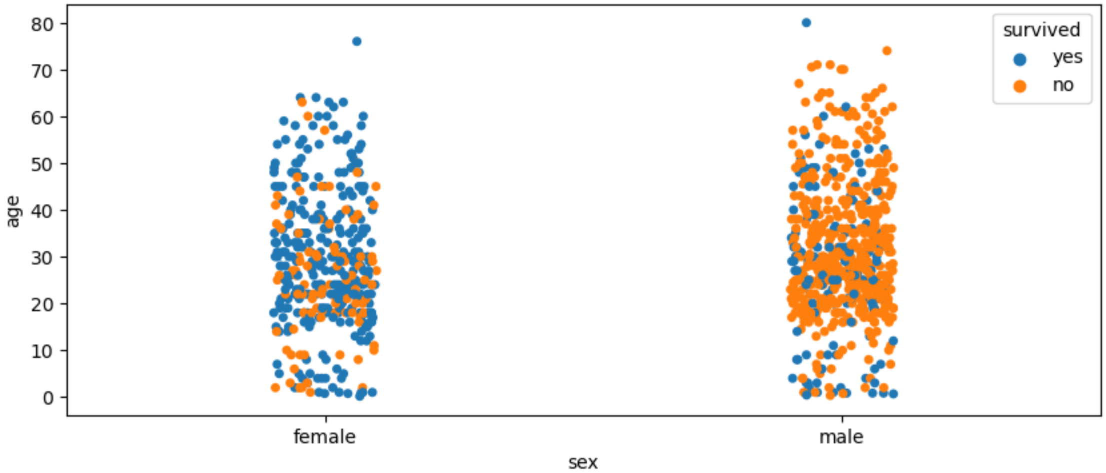
	Name	survived	sex	age	passengerClass
0	Allen, Miss. Elisabeth Walton	yes	female	29.0000	1
1	Allison, Master. Hudson Trevor	yes	male	0.9167	1
2	Allison, Miss. Helen Loraine	no	female	2.0000	1
3	Allison, Mr. Hudson Joshua Crei	no	male	30.0000	1
4	Allison, Mrs. Hudson J C (Bessi	no	female	25.0000	1

[23]:
```
sns.countplot(x='passengerClass', hue='survived', data=data)
plt.title('Passengerclass: Sruvived vs Dead')
plt.show()
```



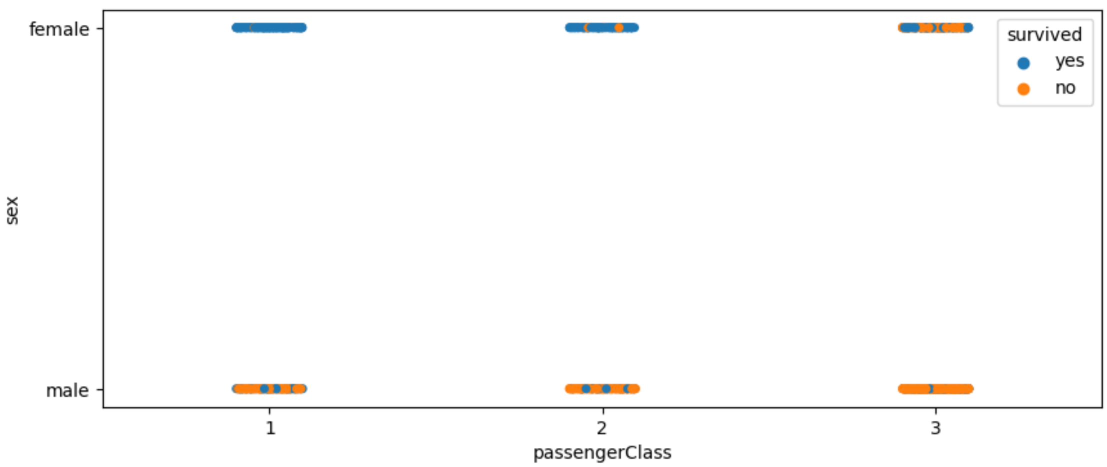
[24]:
```
sns.stripplot(x="sex",y="age",data=data,hue="survived")
```

[24]: <Axes: xlabel='sex', ylabel='age'>



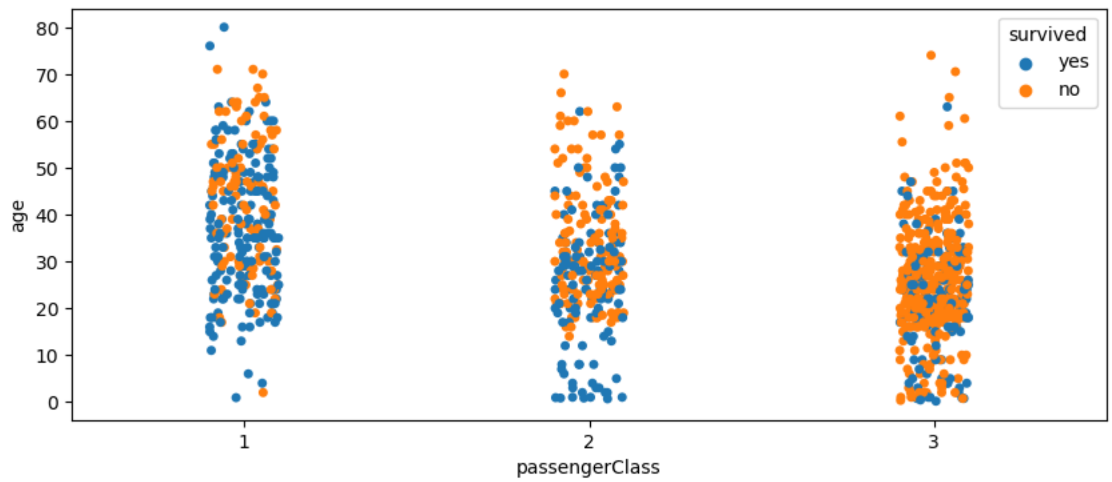
[25]:
```
sns.stripplot(x="passengerClass",y="sex",data=data,hue="survived")
```

[25]: <Axes: xlabel='passengerClass', ylabel='sex'>



[26]:
```
sns.stripplot(x="passengerClass",y="age",data=data,hue="survived")
```

[26]: <Axes: xlabel='passengerClass', ylabel='age'>



[27]:
```
data['age'].fillna(data["age"].median() , inplace = True)
```

[28]:
```
data.isnull().sum()
```

[28]:
```
Name              0
survived          0
sex               0
age               0
passengerClass    0
dtype: int64
```

[29]:
```
data.drop('Name', axis=1, inplace=True)
```

[30]:
```
from sklearn.preprocessing import LabelEncoder
data_y=pd.DataFrame(data["survived"])
target=LabelEncoder().fit_transform(data_y)
target_df=pd.DataFrame(target,columns=["survived"])
data['survived']=target_df["survived"]
data.head()
```

[30]:

	survived	sex	age	passengerClass
0	1	female	29.0000	1
1	1	male	0.9167	1
2	0	female	2.0000	1
3	0	male	30.0000	1
4	0	female	25.0000	1

[31]:
```
data.dtypes
```

[31]:
```
survived            int32
sex                object
age               float64
passengerClass     object
dtype: object
```

[32]:
```
data_numeric = data.select_dtypes(include=np.number)
print(data_numeric.columns)
data_categoric = data.select_dtypes(include = object)
print(data_categoric.columns)
```

```
Index(['survived', 'age'], dtype='object')
Index(['sex', 'passengerClass'], dtype='object')
```

[33]:
```
dummy_variables = pd.get_dummies(data_categoric, drop_first = True)
```

[34]:
```
data_dummy = pd.concat([data_numeric, dummy_variables], axis=1)
data_dummy.head()
```

[34]:

	survived	age	sex_male	passengerClass_2	passengerClass_3
0	1	29.0000	0	0	0
1	1	0.9167	1	0	0
2	0	2.0000	0	0	0
3	0	30.0000	1	0	0
4	0	25.0000	0	0	0

[35]:
```
data_dummy.shape
```

[35]: (1309, 5)

[62]:
```
X = data_dummy.drop(['survived'], axis = 1)
#X=sm.add_constant(X)
#data_y =['0' if x < 0.8 else '1' for x in data["CoA"]]
#y=np.array(data_y,dtype=np.float32)
y = pd.DataFrame(data_dummy['survived'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 1)
```

[63]:
```
def get_test_report(model):
    return(classification_report(y_test,y_pred))
```

[64]:
```
def plot_confusion_matrix(model):
    cm = confusion_matrix(y_test, y_pred)
    conf_matrix = pd.DataFrame(data = cm,columns = ['Predicted:0','Predicted:
↪1'], index = ['Actual:0','Actual:1'])
```

```
    sns.heatmap(conf_matrix, annot = True, fmt = 'd', cmap =␣
↪ListedColormap(['lightskyblue']),cbar = False, linewidths = 0.1, annot_kws =␣
↪{'size':25})
    plt.xticks(fontsize = 20)
    plt.yticks(fontsize = 20)
    plt.show()
```

[65]:
```
def plot_roc(model):
    fpr,tpr,_=roc_curve(y_test,y_pred)
    plt.plot(fpr,tpr)
    plt.xlim([0.0,1.0])
    plt.ylim([0.0,1.0])
    plt.plot([0,1],[0,1],"r--")
    plt.title("ROC Curve",fontsize=15)
    plt.xlabel("False positive",fontsize=15)
    plt.ylabel("True positive",fontsize=15)
    plt.text(x=0.02,y=0.9,s=("AUC Score:
↪",round(roc_auc_score(y_test,y_pred),4)))
    plt.grid(True)
```

[66]:
```
score_card=pd.DataFrame(columns=["Model","AUC Score","Precision Score","Recall␣
↪Score","Accuracy Score","Kappa Score","f1-Score"])
def update_score_card(model_name):
    global score_card
    score_card=score_card.append({"Model":model_name,"AUC Score":
↪roc_auc_score(y_test,y_pred),"Precision Score":metrics.
↪precision_score(y_test,y_pred),"Recall Score":metrics.
↪accuracy_score(y_test,y_pred),'Accuracy Score': metrics.
↪accuracy_score(y_test, y_pred),"Kappa Score":
↪cohen_kappa_score(y_test,y_pred),"f1-Score":metrics.
↪f1_score(y_test,y_pred)},ignore_index=True)
    return(score_card)
```

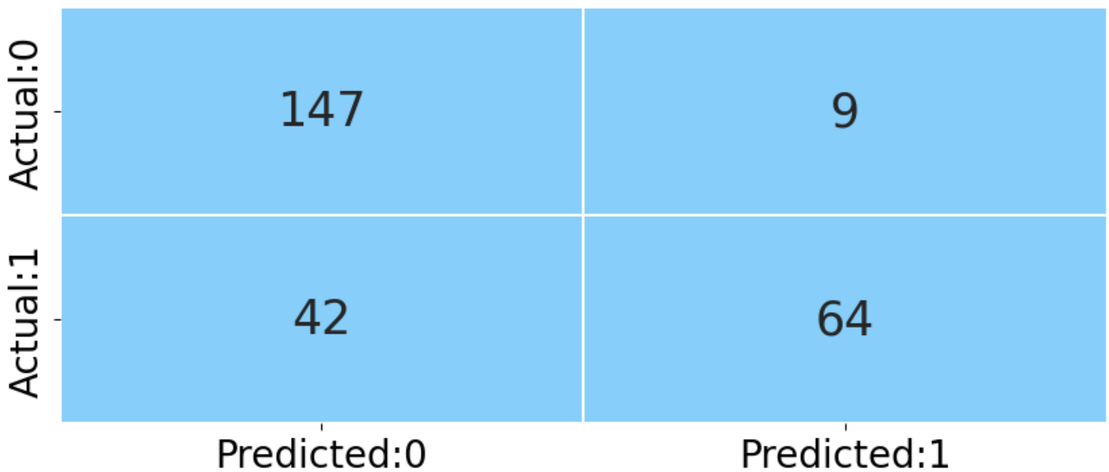
[67]:
```
from sklearn.ensemble import RandomForestClassifier
#intantiate the regressor
rf_cls = RandomForestClassifier(n_estimators=100, random_state=10)

# fit the regressor with training dataset
rf_cls.fit(X_train, y_train)
```

[67]:
```
RandomForestClassifier(random_state=10)
```

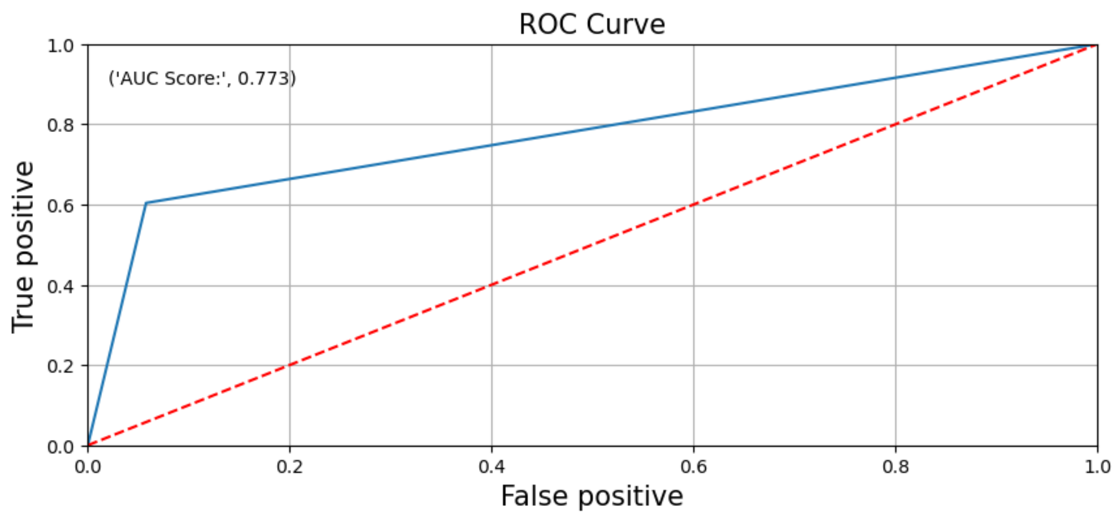
[68]:
```
y_pred = rf_cls.predict(X_test)
y_pred
```

[68]:
```
array([0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1])
```

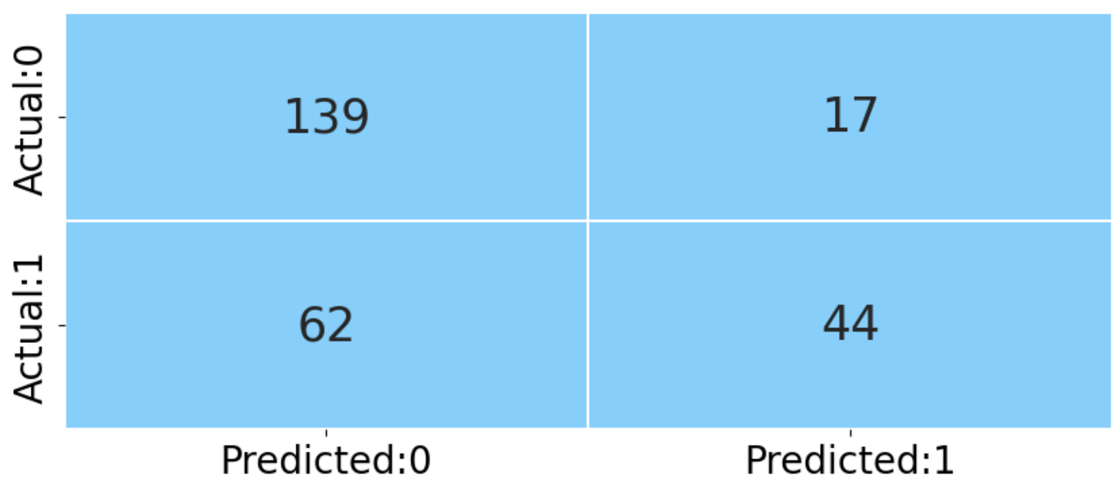
[69]:
```
plot_confusion_matrix(rf_cls)
plot_roc(rf_cls)
update_score_card(model_name="rf_cls")
```



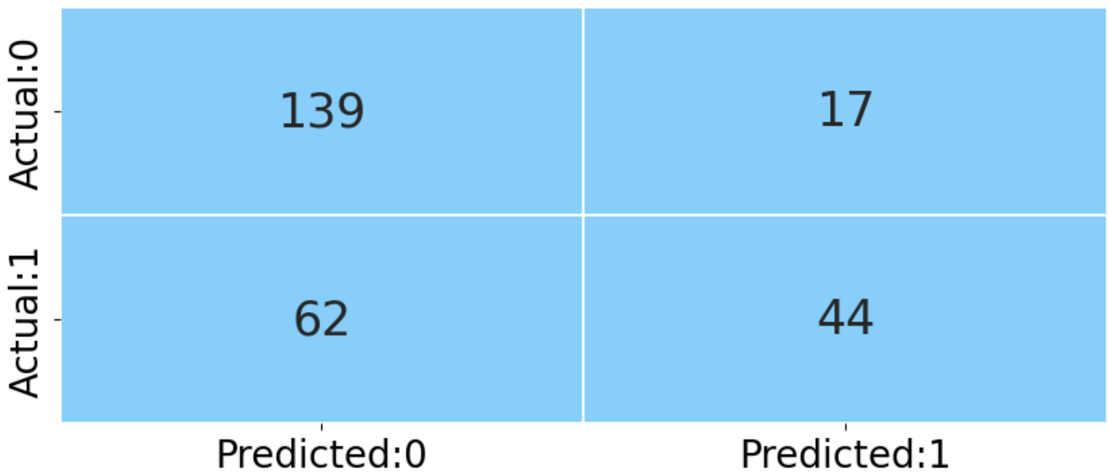
[69]:
```
    Model  AUC Score  Precision Score  Recall Score  Accuracy Score  \
0  rf_cls   0.773041         0.876712      0.805344        0.805344

   Kappa Score  f1-Score
0     0.574757  0.715084
```



[70]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
KN_Classifier=KNeighborsClassifier(n_neighbors=29)
KN_Classifier_st=KN_Classifier.fit(X_train, y_train)
y_pred_prob =KN_Classifier_st.predict_proba(X_test)[:,1]
y_pred =KN_Classifier_st .predict(X_test)
plot_confusion_matrix(KN_Classifier_st)
```

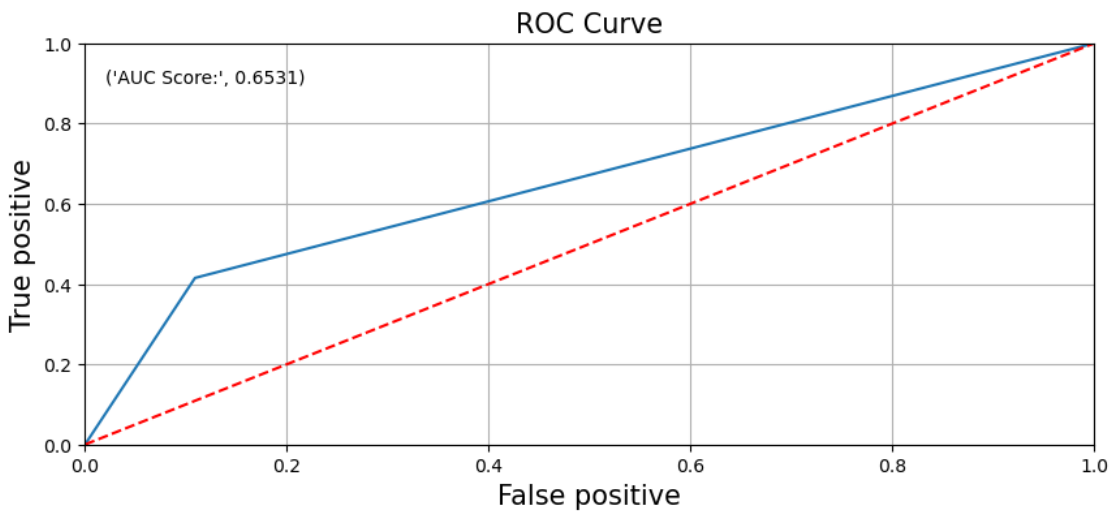


[71]:
```
plot_confusion_matrix(KN_Classifier_st)
plot_roc(KN_Classifier_st)
update_score_card(model_name="KN_Classifier_st")
```



[71]:

```
               Model  AUC Score  Precision Score  Recall Score  Accuracy Score  \
0             rf_cls   0.773041         0.876712      0.805344        0.805344
1  KN_Classifier_st   0.653060         0.721311      0.698473        0.698473

   Kappa Score  f1-Score
0     0.574757  0.715084
1     0.328467  0.526946
```



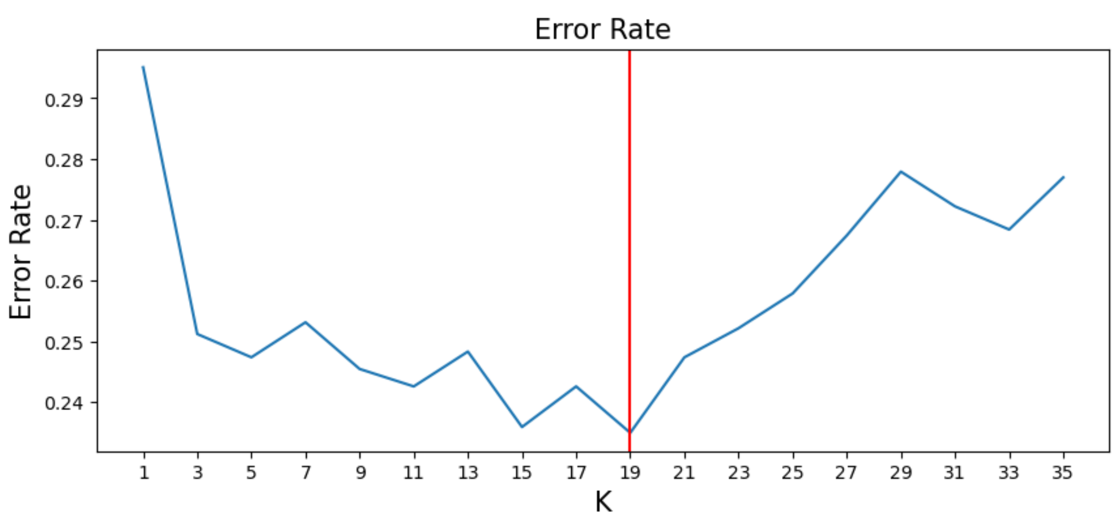
[72]:

```python
tuned_paramaters = {'n_neighbors': np.arange(1, 51, 2),'metric':
 ↪['hamming','euclidean','manhattan','Chebyshev']}
knn_classification = KNeighborsClassifier()
knn_grid = GridSearchCV(estimator = knn_classification,param_grid =
 ↪tuned_paramaters,cv = 5,scoring = 'accuracy')
```

```
knn_grid.fit(X_train, y_train)
print('Best parameters for KNN Classifier: ', knn_grid.best_params_, '\n')
```

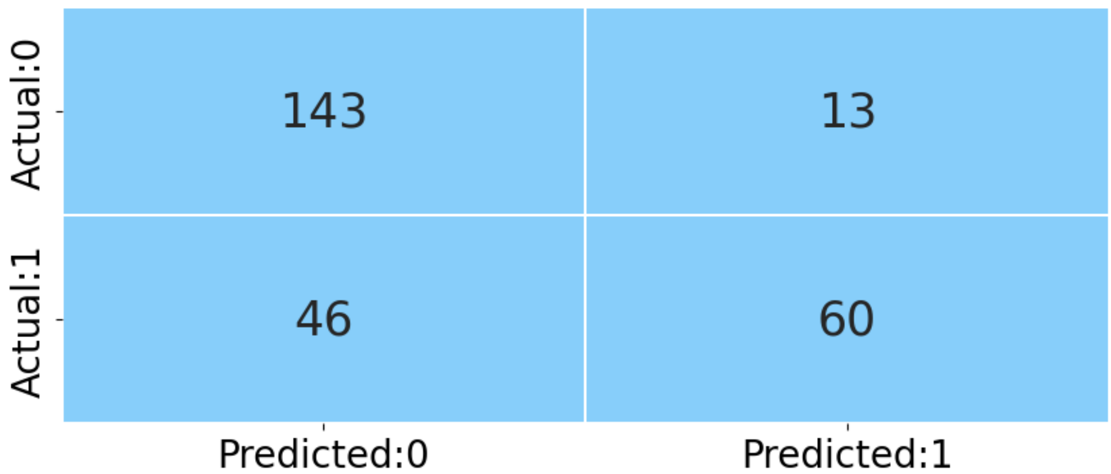
```
Best parameters for KNN Classifier:  {'metric': 'hamming', 'n_neighbors': 43}
```

[73]:
```
error_rate = []
for i in np.arange(1,37,2):
    knn = KNeighborsClassifier(i, metric = 'manhattan')
    score = cross_val_score(knn, X_train, y_train, cv = 5)
    score = score.mean()
    error_rate.append(1 - score)
plt.plot(range(1,37,2), error_rate)
plt.title('Error Rate', fontsize = 15)
plt.xlabel('K', fontsize = 15)
plt.ylabel('Error Rate', fontsize = 15)
plt.xticks(np.arange(1, 37, step = 2))
plt.axvline(x = 19, color = 'red')
plt.show()
```



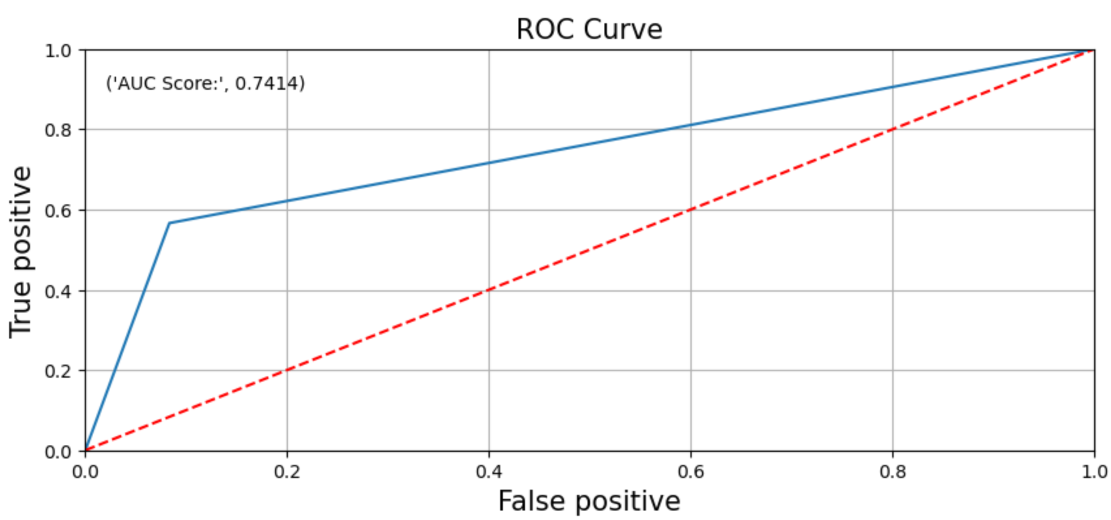
[74]:
```
KN_Classifier=KNeighborsClassifier(n_neighbors=19,metric='manhattan')
KN_Classifier_tunning=KN_Classifier.fit(X_train, y_train)
y_pred_prob =KN_Classifier_tunning.predict_proba(X_test)[:,1]
y_pred =KN_Classifier_tunning .predict(X_test)
```

[75]:
```
plot_confusion_matrix(KN_Classifier_tunning)
plot_roc(KN_Classifier_tunning)
update_score_card(model_name="KN_Classifier_tunning")
```



[75]:

```
                   Model  AUC Score  Precision Score  Recall Score  \
0                 rf_cls   0.773041         0.876712      0.805344
1        KN_Classifier_st   0.653060         0.721311      0.698473
2  KN_Classifier_tunning   0.741352         0.821918      0.774809

   Accuracy Score  Kappa Score  f1-Score
0        0.805344     0.574757  0.715084
1        0.698473     0.328467  0.526946
2        0.774809     0.508052  0.670391
```

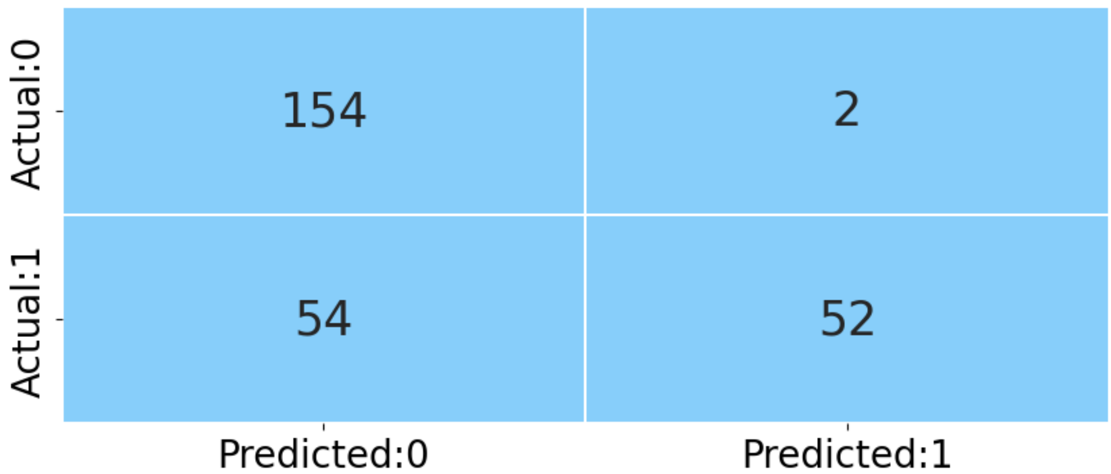


[76]:

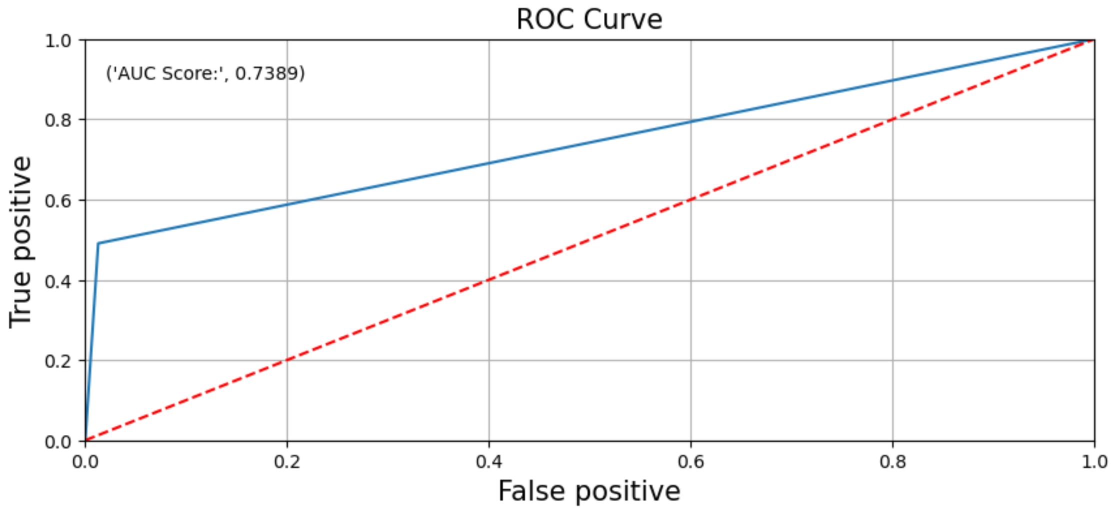
```python
from xgboost.sklearn import XGBClassifier
xgbm=XGBClassifier(random_state=1,learning_rate=0.01)
xgbm.fit(X_train,y_train)
```

```
y_pred =xgbm .predict(X_test)
```

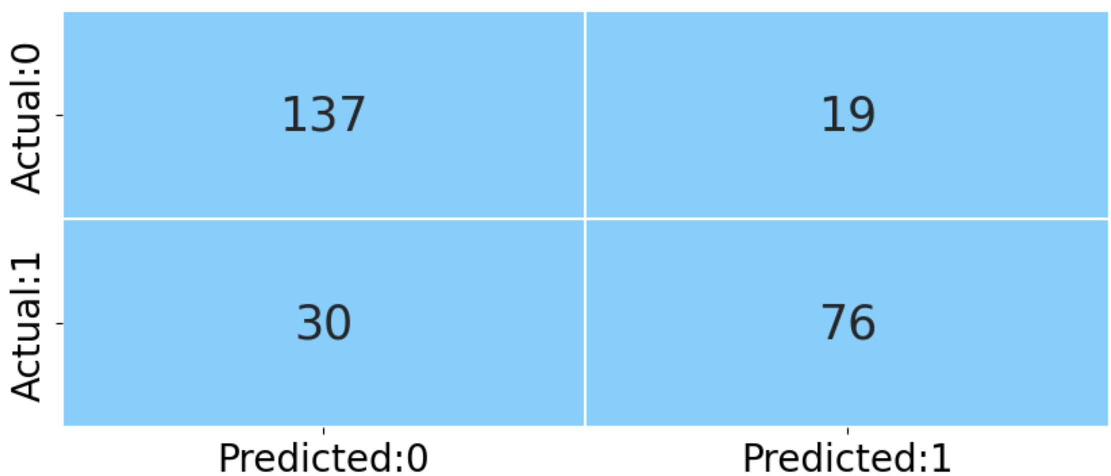
[77]:
```
plot_confusion_matrix(xgbm)
plot_roc(xgbm)
update_score_card(model_name="xgbm")
```



[77]:
```
                   Model  AUC Score  Precision Score  Recall Score  \
0                 rf_cls   0.773041         0.876712      0.805344
1       KN_Classifier_st   0.653060         0.721311      0.698473
2  KN_Classifier_tunning   0.741352         0.821918      0.774809
3                   xgbm   0.738873         0.962963      0.786260

   Accuracy Score  Kappa Score  f1-Score
0        0.805344     0.574757  0.715084
1        0.698473     0.328467  0.526946
2        0.774809     0.508052  0.670391
3        0.786260     0.518509  0.650000
```




```
[78]: svc_linear = SVC(kernel='linear', probability=True)  # Specify␣
       ↪'probability=True' to enable probability estimates
      svm_linear=svc_linear.fit(X_train, y_train)
      y_pred_prob =svm_linear.predict_proba(X_test)[:,1]
      y_pred =svm_linear .predict(X_test)
      plot_confusion_matrix(svm_linear)
      test_report = get_test_report(svm_linear)
      print(test_report)
      plot_roc(svm_linear)
      update_score_card(model_name = 'svm_linear')
```

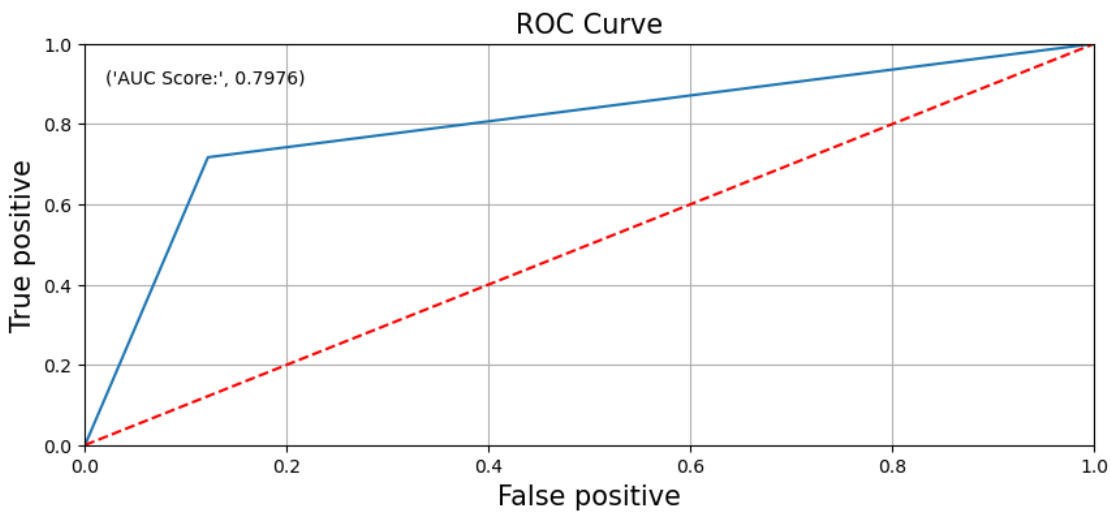



```
              precision    recall  f1-score   support
```

```
              0       0.82      0.88      0.85       156
              1       0.80      0.72      0.76       106

       accuracy                           0.81       262
      macro avg       0.81      0.80      0.80       262
   weighted avg       0.81      0.81      0.81       262
```

[78]:

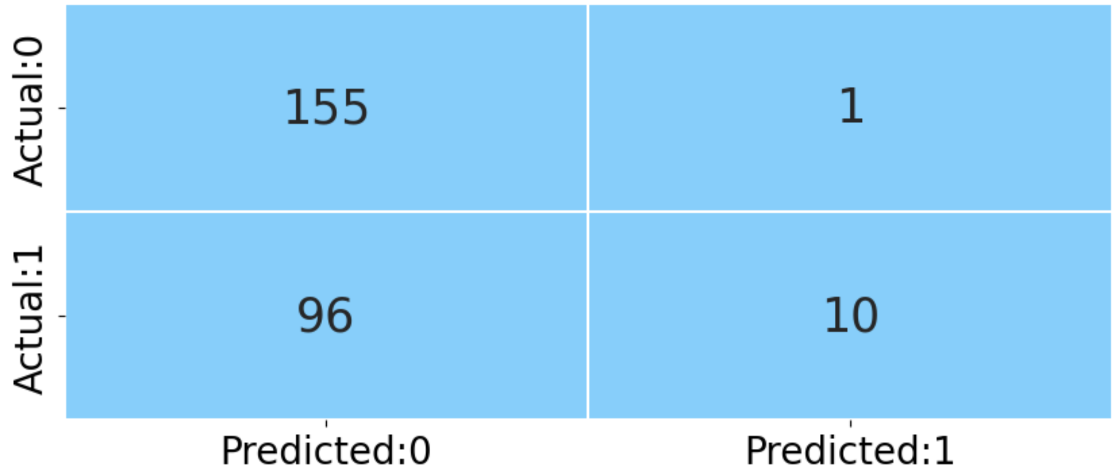
```
                     Model  AUC Score  Precision Score  Recall Score  \
0                   rf_cls   0.773041         0.876712      0.805344
1         KN_Classifier_st   0.653060         0.721311      0.698473
2  KN_Classifier_tunning   0.741352         0.821918      0.774809
3                     xgbm   0.738873         0.962963      0.786260
4               svm_linear   0.797593         0.800000      0.812977

   Accuracy Score  Kappa Score  f1-Score
0        0.805344     0.574757  0.715084
1        0.698473     0.328467  0.526946
2        0.774809     0.508052  0.670391
3        0.786260     0.518509  0.650000
4        0.812977     0.605252  0.756219
```



[79]:

```
svc_poly = SVC(kernel='poly', probability=True)  # Specify 'probability=True'␣
 ↪to enable probability estimates
svm_poly=svc_poly.fit(X_train, y_train)
y_pred_prob =svm_poly.predict_proba(X_test)[:,1]
y_pred =svm_poly .predict(X_test)
plot_confusion_matrix(svm_poly)
test_report = get_test_report(svm_poly)
print(test_report)
```

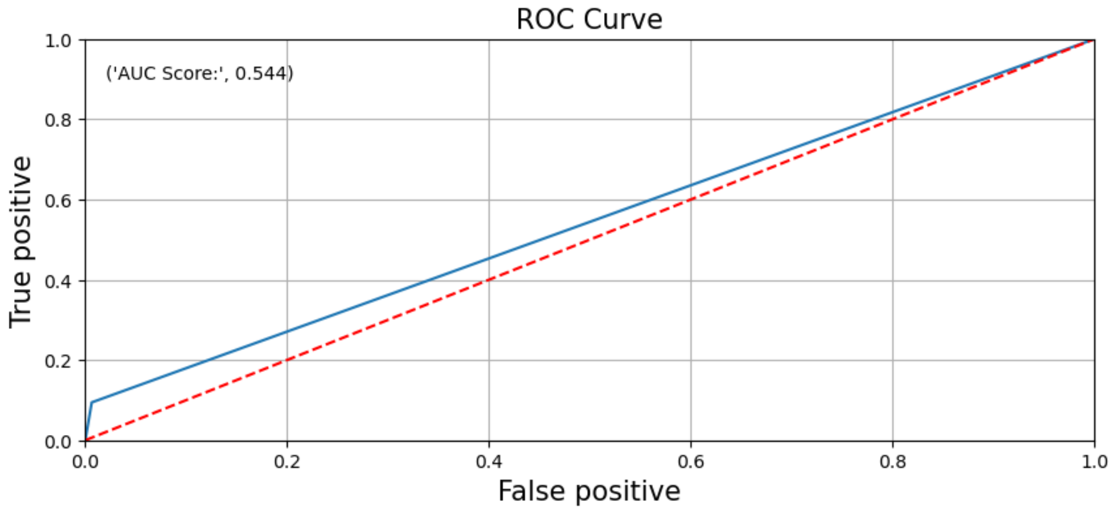
```
plot_roc(svm_poly)
update_score_card(model_name = 'svm_poly')
```




```
              precision    recall  f1-score   support

           0       0.62      0.99      0.76       156
           1       0.91      0.09      0.17       106

    accuracy                           0.63       262
   macro avg       0.76      0.54      0.47       262
weighted avg       0.74      0.63      0.52       262
```

[79]:

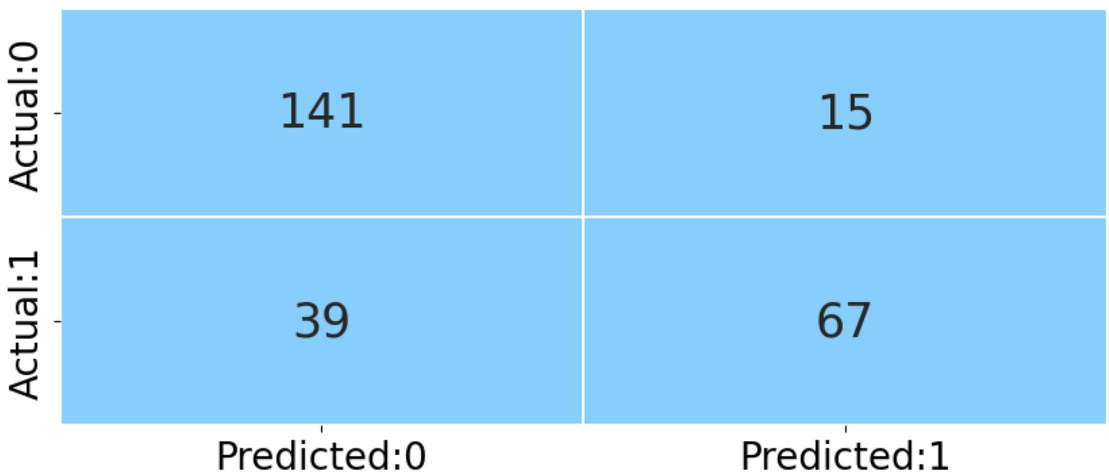
```
                    Model  AUC Score  Precision Score  Recall Score  \
0                  rf_cls   0.773041         0.876712      0.805344
1        KN_Classifier_st   0.653060         0.721311      0.698473
2  KN_Classifier_tunning   0.741352         0.821918      0.774809
3                    xgbm   0.738873         0.962963      0.786260
4              svm_linear   0.797593         0.800000      0.812977
5                svm_poly   0.543965         0.909091      0.629771

   Accuracy Score  Kappa Score  f1-Score
0        0.805344     0.574757  0.715084
1        0.698473     0.328467  0.526946
2        0.774809     0.508052  0.670391
3        0.786260     0.518509  0.650000
4        0.812977     0.605252  0.756219
5        0.629771     0.102676  0.170940
```



[80]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    }

# Initialize the GridSearchCV with RandomForestClassifier
grid_search = GridSearchCV(estimator=RandomForestClassifier(random_state=42),
                           param_grid=param_grid, cv=5)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train, y_train)

# Retrieve the best parameters and the best estimator
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
print("Best Parameters: ", best_params)

# Predict the test set using the best model
y_pred = best_model.predict(X_test)

# Evaluate the model
```

```
print(classification_report(y_test, y_pred))
plot_confusion_matrix(best_model)
plot_roc(best_model)
update_score_card(model_name="Hyper_Parameter_RF")
```

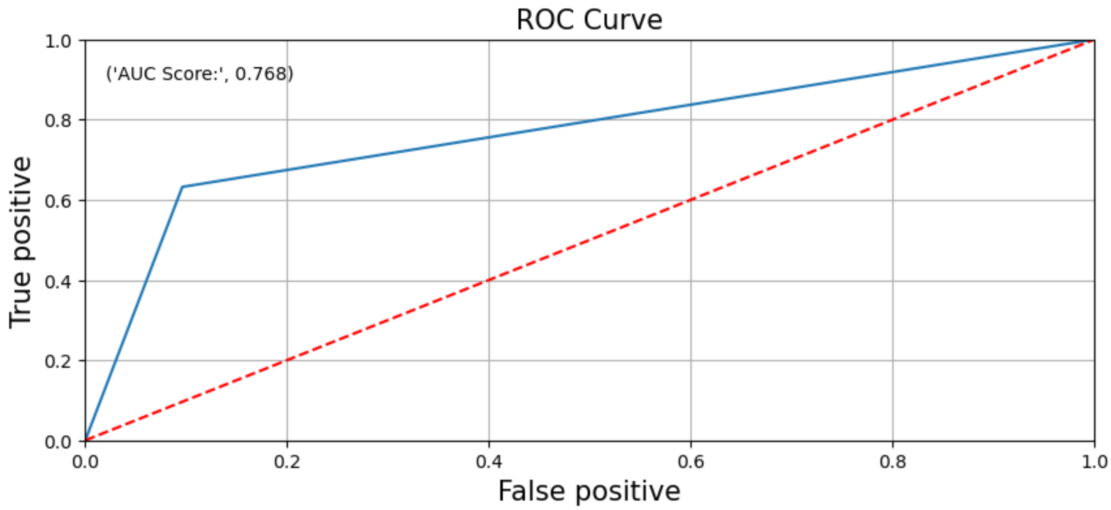
```
Best Parameters:  {'max_depth': 20, 'max_features': 'auto', 'min_samples_leaf':
4, 'min_samples_split': 2, 'n_estimators': 200}
              precision    recall  f1-score   support

           0       0.78      0.90      0.84       156
           1       0.82      0.63      0.71       106

    accuracy                           0.79       262
   macro avg       0.80      0.77      0.78       262
weighted avg       0.80      0.79      0.79       262
```



[80]:

```
                 Model  AUC Score  Precision Score  Recall Score  \
0               rf_cls   0.773041         0.876712      0.805344
1     KN_Classifier_st   0.653060         0.721311      0.698473
2  KN_Classifier_tunning  0.741352        0.821918      0.774809
3                 xgbm   0.738873         0.962963      0.786260
4           svm_linear   0.797593         0.800000      0.812977
5             svm_poly   0.543965         0.909091      0.629771
6   Hyper_Parameter_RF   0.767961         0.817073      0.793893

   Accuracy Score  Kappa Score  f1-Score
0        0.805344     0.574757  0.715084
1        0.698473     0.328467  0.526946
2        0.774809     0.508052  0.670391
```

3	0.786260	0.518509	0.650000
4	0.812977	0.605252	0.756219
5	0.629771	0.102676	0.170940
6	0.793893	0.556099	0.712766

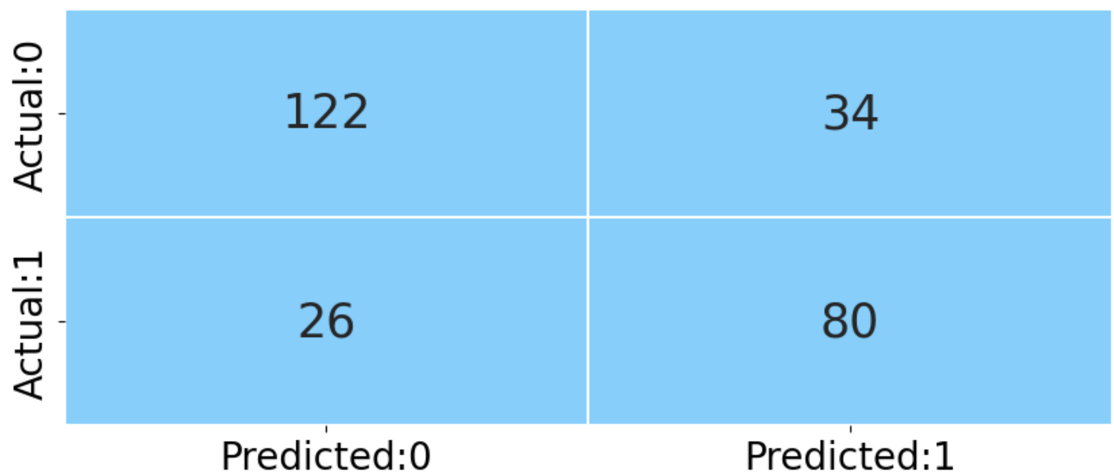


7 Random Undersampling randomly removes samples from the majority class to balance the dataset. This can be easily implemented using the RandomUnderSampler from imbalanced-learn.

[81]:
```
from imblearn.under_sampling import RandomUnderSampler

# Define the undersampling method
undersample = RandomUnderSampler(sampling_strategy='auto', random_state=42)

# Fit and transform the training data
X_train_res, y_train_res = undersample.fit_resample(X_train, y_train)

# Train the model
model_random_forest_undersample = RandomForestClassifier(random_state=42)
model_random_forest_undersample.fit(X_train_res, y_train_res)

# Predict the test set
y_pred =model_random_forest_undersample.predict(X_test)

# Evaluate the model
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.82	0.78	0.80	156
1	0.70	0.75	0.73	106
accuracy			0.77	262
macro avg	0.76	0.77	0.76	262
weighted avg	0.77	0.77	0.77	262

[82]:
```python
plot_confusion_matrix(model_random_forest_undersample)
plot_roc(model_random_forest_undersample)
update_score_card(model_name="Random_forest_undersample")
```

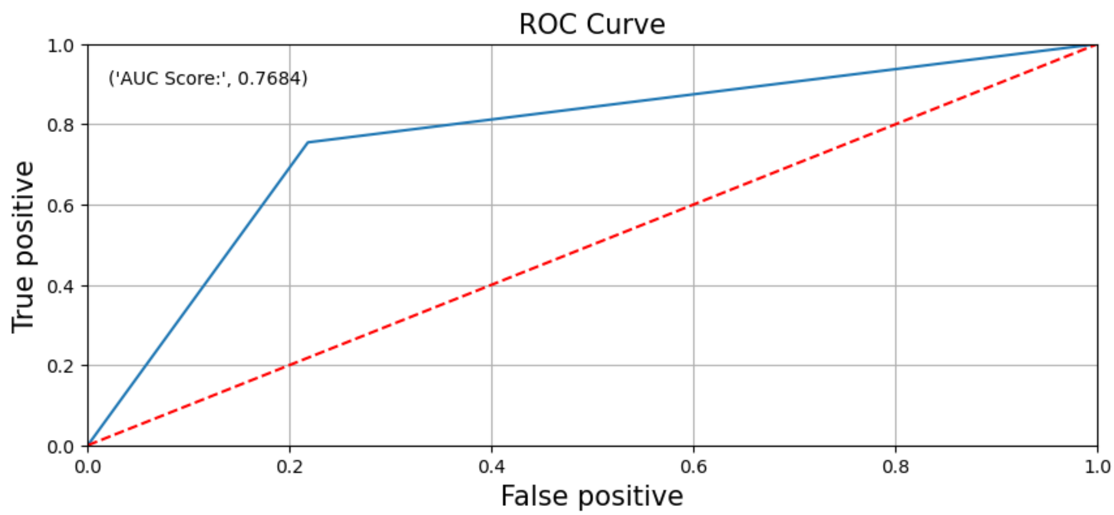


[82]:

	Model	AUC Score	Precision Score	Recall Score	\
0	rf_cls	0.773041	0.876712	0.805344	
1	KN_Classifier_st	0.653060	0.721311	0.698473	
2	KN_Classifier_tunning	0.741352	0.821918	0.774809	
3	xgbm	0.738873	0.962963	0.786260	
4	svm_linear	0.797593	0.800000	0.812977	
5	svm_poly	0.543965	0.909091	0.629771	
6	Hyper_Parameter_RF	0.767961	0.817073	0.793893	
7	Random_forest_undersample	0.768384	0.701754	0.770992	

	Accuracy Score	Kappa Score	f1-Score
0	0.805344	0.574757	0.715084
1	0.698473	0.328467	0.526946
2	0.774809	0.508052	0.670391
3	0.786260	0.518509	0.650000
4	0.812977	0.605252	0.756219
5	0.629771	0.102676	0.170940

6	0.793893	0.556099	0.712766
7	0.770992	0.530354	0.727273



[83]:
```
#!pip install tensorflow
#!pip install keras
```

[84]:
```
#Build Artificial Neural Network
#Import the Keras libraries and packages

import keras
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense
```

[85]:
```
from keras.models import Sequential
from keras.layers import Dense

# Initialize the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units=4, activation='relu', input_shape=(4,)))

# Adding the second hidden layer
classifier.add(Dense(units=4, activation='relu'))

# Adding the output layer
classifier.add(Dense(units=1, activation='sigmoid'))
```

```python
# Compiling the ANN
classifier.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Fit the ANN to the Training set
classifier.fit(X_train, y_train, batch_size=10, epochs=100)
```

```
WARNING:tensorflow:From C:\Users\acer\anaconda3\lib\site-
packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated.
Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From C:\Users\acer\anaconda3\lib\site-
packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is
deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/100
WARNING:tensorflow:From C:\Users\acer\anaconda3\lib\site-
packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue
is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\acer\anaconda3\lib\site-
packages\keras\src\engine\base_layer_utils.py:384: The name
tf.executing_eagerly_outside_functions is deprecated. Please use
tf.compat.v1.executing_eagerly_outside_functions instead.

105/105 [==============================] - 2s 2ms/step - loss: 0.6819 -
accuracy: 0.6275
Epoch 2/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6706 -
accuracy: 0.6256
Epoch 3/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6658 -
accuracy: 0.6218
Epoch 4/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6580 -
accuracy: 0.6237
Epoch 5/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6547 -
accuracy: 0.6237
Epoch 6/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6471 -
accuracy: 0.6237
Epoch 7/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6401 -
accuracy: 0.6237
Epoch 8/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6330 -
```

```
accuracy: 0.6227
Epoch 9/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6227 -
accuracy: 0.6237
Epoch 10/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6156 -
accuracy: 0.6256
Epoch 11/100
105/105 [==============================] - 0s 2ms/step - loss: 0.6043 -
accuracy: 0.6657
Epoch 12/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5965 -
accuracy: 0.6877
Epoch 13/100
105/105 [==============================] - 0s 3ms/step - loss: 0.5878 -
accuracy: 0.6944
Epoch 14/100
105/105 [==============================] - 0s 3ms/step - loss: 0.5773 -
accuracy: 0.7154
Epoch 15/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5668 -
accuracy: 0.7593
Epoch 16/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5616 -
accuracy: 0.7736
Epoch 17/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5513 -
accuracy: 0.8004
Epoch 18/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5432 -
accuracy: 0.7966
Epoch 19/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5377 -
accuracy: 0.7841
Epoch 20/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5314 -
accuracy: 0.7870
Epoch 21/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5246 -
accuracy: 0.7908
Epoch 22/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5201 -
accuracy: 0.7851
Epoch 23/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5151 -
accuracy: 0.7861
Epoch 24/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5118 -
```

```
accuracy: 0.7861
Epoch 25/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5074 -
accuracy: 0.7880
Epoch 26/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5048 -
accuracy: 0.7794
Epoch 27/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5031 -
accuracy: 0.7784
Epoch 28/100
105/105 [==============================] - 0s 2ms/step - loss: 0.5019 -
accuracy: 0.7822
Epoch 29/100
105/105 [==============================] - 0s 3ms/step - loss: 0.4993 -
accuracy: 0.7813
Epoch 30/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4983 -
accuracy: 0.7784
Epoch 31/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4925 -
accuracy: 0.7870
Epoch 32/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4960 -
accuracy: 0.7861
Epoch 33/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4930 -
accuracy: 0.7803
Epoch 34/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4899 -
accuracy: 0.7784
Epoch 35/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4884 -
accuracy: 0.7841
Epoch 36/100
105/105 [==============================] - 0s 3ms/step - loss: 0.4865 -
accuracy: 0.7803
Epoch 37/100
105/105 [==============================] - 0s 3ms/step - loss: 0.4877 -
accuracy: 0.7765
Epoch 38/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4882 -
accuracy: 0.7775
Epoch 39/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4840 -
accuracy: 0.7813
Epoch 40/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4845 -
```

```
accuracy: 0.7784
Epoch 41/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4837 -
accuracy: 0.7813
Epoch 42/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4859 -
accuracy: 0.7927
Epoch 43/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4833 -
accuracy: 0.7861
Epoch 44/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4816 -
accuracy: 0.7832
Epoch 45/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4809 -
accuracy: 0.7813
Epoch 46/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4803 -
accuracy: 0.7832
Epoch 47/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4805 -
accuracy: 0.7813
Epoch 48/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4800 -
accuracy: 0.7803
Epoch 49/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4798 -
accuracy: 0.7803
Epoch 50/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4820 -
accuracy: 0.7841
Epoch 51/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4807 -
accuracy: 0.7822
Epoch 52/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4782 -
accuracy: 0.7803
Epoch 53/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4772 -
accuracy: 0.7803
Epoch 54/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4778 -
accuracy: 0.7813
Epoch 55/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4808 -
accuracy: 0.7765
Epoch 56/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4761 -
```

```
accuracy: 0.7841
Epoch 57/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4763 -
accuracy: 0.7794
Epoch 58/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4746 -
accuracy: 0.7803
Epoch 59/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4740 -
accuracy: 0.7822
Epoch 60/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4735 -
accuracy: 0.7794
Epoch 61/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4731 -
accuracy: 0.7822
Epoch 62/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4683 -
accuracy: 0.7899
Epoch 63/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4743 -
accuracy: 0.7784
Epoch 64/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4726 -
accuracy: 0.7899
Epoch 65/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4738 -
accuracy: 0.7755
Epoch 66/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4690 -
accuracy: 0.7927
Epoch 67/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4690 -
accuracy: 0.7841
Epoch 68/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4689 -
accuracy: 0.7775
Epoch 69/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4663 -
accuracy: 0.7908
Epoch 70/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4665 -
accuracy: 0.7803
Epoch 71/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4633 -
accuracy: 0.7841
Epoch 72/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4660 -
```

```
accuracy: 0.7851
Epoch 73/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4689 -
accuracy: 0.7880
Epoch 74/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4650 -
accuracy: 0.7755
Epoch 75/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4620 -
accuracy: 0.7918
Epoch 76/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4643 -
accuracy: 0.7794
Epoch 77/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4630 -
accuracy: 0.7851
Epoch 78/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4640 -
accuracy: 0.7851
Epoch 79/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4618 -
accuracy: 0.7794
Epoch 80/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4643 -
accuracy: 0.7755
Epoch 81/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4628 -
accuracy: 0.7899
Epoch 82/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4624 -
accuracy: 0.7880
Epoch 83/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4612 -
accuracy: 0.7908
Epoch 84/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4646 -
accuracy: 0.7784
Epoch 85/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4611 -
accuracy: 0.7880
Epoch 86/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4619 -
accuracy: 0.7746
Epoch 87/100
105/105 [==============================] - 0s 3ms/step - loss: 0.4625 -
accuracy: 0.7937
Epoch 88/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4612 -
```

```
accuracy: 0.7966
Epoch 89/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4588 -
accuracy: 0.7908
Epoch 90/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4596 -
accuracy: 0.7851
Epoch 91/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4612 -
accuracy: 0.7841
Epoch 92/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4601 -
accuracy: 0.7813
Epoch 93/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4611 -
accuracy: 0.7937
Epoch 94/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4603 -
accuracy: 0.7822
Epoch 95/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4606 -
accuracy: 0.7889
Epoch 96/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4583 -
accuracy: 0.7956
Epoch 97/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4605 -
accuracy: 0.7765
Epoch 98/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4610 -
accuracy: 0.7908
Epoch 99/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4586 -
accuracy: 0.7803
Epoch 100/100
105/105 [==============================] - 0s 2ms/step - loss: 0.4592 -
accuracy: 0.7880
```

[85]:
```
<keras.src.callbacks.History at 0x22ad6c24850>
```

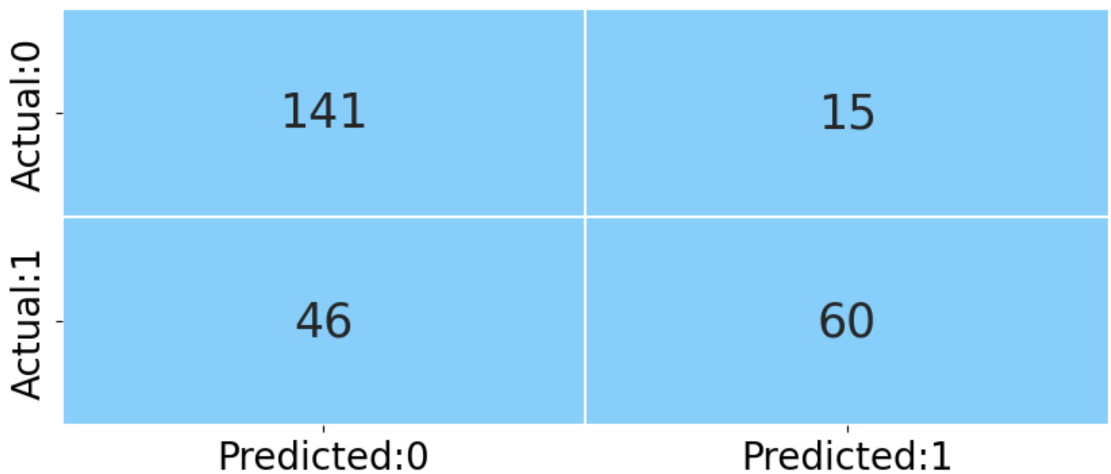
[86]:
```
#Predict the Test Set Results
y_pred = classifier.predict(X_test)
y_pred = (y_pred > 0.5)

#y_pred > 0.5 means if y-pred is in between 0 to 0.5, then this new y_pred will
 ↪become 0(False). And if y_pred is larger than
#0.5, then the new y_pred will become 1(True)
```

```
9/9 [==============================] - 0s 2ms/step
```

[87]:

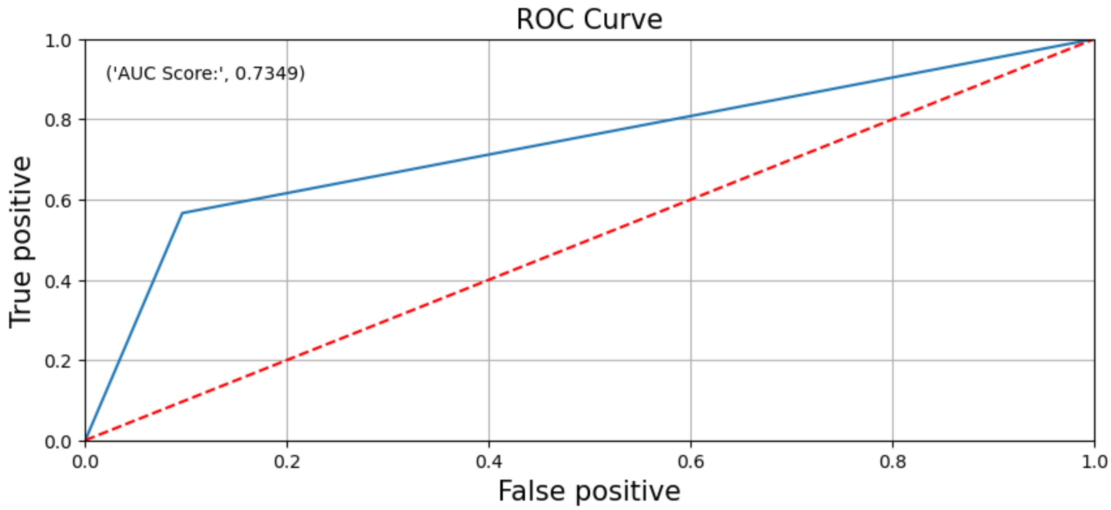
```python
test_report = get_test_report(classifier)
print(classifier)
plot_confusion_matrix(classifier)
plot_roc(classifier)
update_score_card(model_name = 'ANN_classifier')
```

```
<keras.src.engine.sequential.Sequential object at 0x0000022AD6BFF0A0>
```



[87]:

	Model	AUC Score	Precision Score	Recall Score	Accuracy Score	Kappa Score	f1-Score
0	rf_cls	0.773041	0.876712	0.805344	0.805344	0.574757	0.715084
1	KN_Classifier_st	0.653060	0.721311	0.698473	0.698473	0.328467	0.526946
2	KN_Classifier_tunning	0.741352	0.821918	0.774809	0.774809	0.508052	0.670391
3	xgbm	0.738873	0.962963	0.786260	0.786260	0.518509	0.650000
4	svm_linear	0.797593	0.800000	0.812977	0.812977	0.605252	0.756219
5	svm_poly	0.543965	0.909091	0.629771	0.629771	0.102676	0.170940
6	Hyper_Parameter_RF	0.767961	0.817073	0.793893	0.793893	0.556099	0.712766
7	Random_forest_undersample	0.768384	0.701754	0.770992	0.770992	0.530354	0.727273
8	ANN_classifier	0.734942	0.800000	0.767176	0.767176	0.492989	0.662983




```
[101]: X = data_dummy.drop(['survived'], axis = 1)
       X=sm.add_constant(X)
       #data_y =['0' if x < 0.8 else '1' for x in data["CoA"]]
       #y=np.array(data_y,dtype=np.float32)
       y = pd.DataFrame(data_dummy['survived'])
       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
        ↪random_state = 1)
```

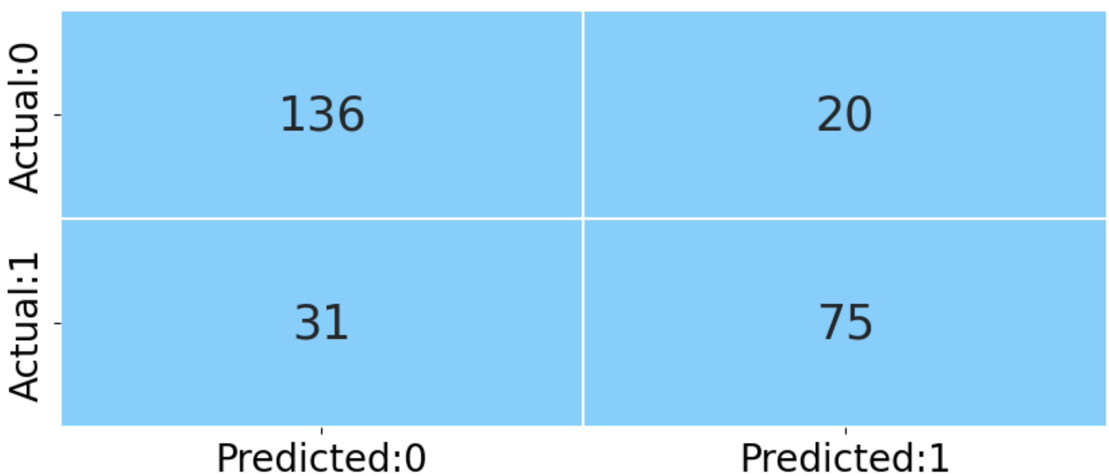
```
[102]: log_reg_model=sm.Logit(y_train,X_train).fit()
       print(log_reg_model.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.473698
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:               survived   No. Observations:                 1047
Model:                          Logit   Df Residuals:                     1042
Method:                           MLE   Df Model:                            4
Date:                Fri, 28 Jun 2024   Pseudo R-squ.:                  0.2847
Time:                        12:40:52   Log-Likelihood:                -495.96
converged:                       True   LL-Null:                       -693.36
Covariance Type:            nonrobust   LLR p-value:                 3.721e-84
================================================================================
====
                       coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
----
const                3.3443      0.332     10.081      0.000       2.694
```

```
3.995
age                   -0.0349      0.007     -5.177      0.000      -0.048
-0.022
sex_male              -2.3973      0.165    -14.567      0.000      -2.720
-2.075
passengerClass_2      -1.1509      0.231     -4.986      0.000      -1.603
-0.698
passengerClass_3      -2.1914      0.216    -10.152      0.000      -2.615
-1.768
================================================================================
====
```

[103]:

```
y_pred_prob=log_reg_model.predict(X_test)
y_pred=["0" if x<0.5 else "1" for x in y_pred_prob]
y_pred=np.array(y_pred,dtype=np.float32)
y_pred[0:5]
plot_confusion_matrix(log_reg_model)
plot_roc(log_reg_model)
update_score_card(model_name="log_reg_model")
```

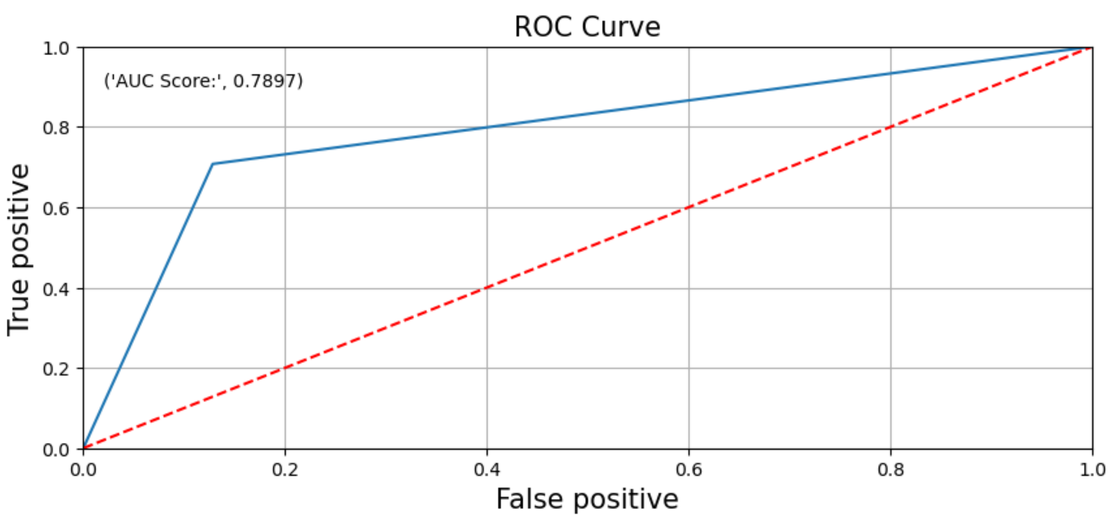


[103]:

	Model	AUC Score	Precision Score	Recall Score	\
0	rf_cls	0.773041	0.876712	0.805344	
1	KN_Classifier_st	0.653060	0.721311	0.698473	
2	KN_Classifier_tunning	0.741352	0.821918	0.774809	
3	xgbm	0.738873	0.962963	0.786260	
4	svm_linear	0.797593	0.800000	0.812977	
5	svm_poly	0.543965	0.909091	0.629771	
6	Hyper_Parameter_RF	0.767961	0.817073	0.793893	
7	Random_forest_undersample	0.768384	0.701754	0.770992	
8	ANN_classifier	0.734942	0.800000	0.767176	
9	log_reg_model	0.766625	0.741497	0.781170	

10	log_reg_model	0.789671	0.789474	0.805344
11	log_reg_model	0.789671	0.789474	0.805344

	Accuracy Score	Kappa Score	f1-Score
0	0.805344	0.574757	0.715084
1	0.698473	0.328467	0.526946
2	0.774809	0.508052	0.670391
3	0.786260	0.518509	0.650000
4	0.812977	0.605252	0.756219
5	0.629771	0.102676	0.170940
6	0.793893	0.556099	0.712766
7	0.770992	0.530354	0.727273
8	0.767176	0.492989	0.662983
9	0.781170	0.538997	0.717105
10	0.805344	0.589140	0.746269
11	0.805344	0.589140	0.746269



8 Conclusion:

This project aimed to predict Titanic passenger survival using Logistic Regression, Random Forest, Support Vector Machine (SVM) with linear and polynomial kernels, and Artificial Neural Network (ANN) models. The SVM with a linear kernel demonstrated superior performance with an accuracy score of 81%. Exploratory Data Analysis (EDA) revealed higher survival rates for Class 1 passengers, females, aged individuals, and children. The study successfully applied both machine learning and deep learning models to predict survival outcomes. It also identified significant survival patterns, enhancing the understanding of the Titanic dataset. Key factors influencing survival included passenger class, gender, age, fare, and embarkation location. The findings highlight the importance of EDA and the effective application of various modeling techniques to derive meaningful insights from historical data.

```
[ ]:
```

```
[ ]:
```