telecom-churn-prediction

June 30, 2024

1 Enhancing Telecom Customer Retention Through Machine Learning and Deep Learning-Powered Churn Prediction

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

2 Project Objective:

The primary objective of the "Telecom Dynamics: Advancing Customer Retention with Machine Learning-Powered Churn Analysis" project was to identify and analyze the key factors influencing customer churn in the telecommunications sector. By leveraging machine learning models, the project aimed to predict which customers are at high risk of churning and provide actionable insights to enhance customer retention strategies.

PROJECT OUTCOMES Model Performance: • K-Nearest Neighbors (KNN): Among the models tested, KNN with K=31 and the Manhattan distance metric demonstrated superior performance in predicting customer churn. • Other Models: A total of 12 models were evaluated, including Random Forest, KNN with hyperparameter tuning, XGBoost, SVM with linear and polynomial kernels, Random Forest with hyperparameter tuning, Random Forest with feature selection, Random Forest with undersampling, Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN). SVM with a linear kernel achieved the highest accuracy of 80%, although it had a lower precision score compared to other metrics.

Churn Prediction: • The KNN model effectively identified high-risk churn customers, enabling targeted retention efforts. • The SVM model with a linear kernel showed strong overall accuracy, although its precision score was an area needing improvement. Key Indicators of Churn: • Total Charges: A significant predictor of churn, indicating that higher total charges correlate with a higher likelihood of customers leaving. • Tenure: Customers with shorter tenure were more likely to churn, highlighting the importance of early intervention. • Monthly Charges: Higher monthly charges were linked to increased churn risk, suggesting that pricing strategies play a critical role in customer retention. • Internet Service: The type of internet service (e.g., DSL, Fiber optic, No service) also significantly influenced churn. Churn Likelihood: • The combination of Total Charges, Tenure, Monthly Charges, and Internet Service accounted for a significant portion of the likelihood of customer churn, underscoring their importance in predictive modeling. Strategic Insights: • The analysis provided strategic insights into the factors driving customer churn. By focusing on managing Total Charges, optimizing Tenure-based interventions, and adjusting Monthly Charges, telecommunications companies can better retain their customers.

Competitive Advantage: • Utilizing machine learning models for churn analysis offers a competitive advantage in the market by enabling proactive customer retention strategies and personalized interventions.

3 About Data

4 About Data: The data set from a customer churns dataset, likely from a telecommunications company. This dataset includes 7043 entries with 20 columns, detailing various attributes of customers and whether they have churned (left the service) or not. Here's a breakdown of the columns:

gender: Gender of the customer (e.g., male, female). SeniorCitizen: Whether the customer is a senior citizen (likely encoded as binary, e.g., 0 for no, 1 for yes). Partner: Whether the customer has a partner (e.g., yes, no). Dependents: Whether the customer has dependents (e.g., yes, no). tenure: Number of months the customer has stayed with the company. PhoneService: Whether the customer has phone service (e.g., yes, no). MultipleLines: Whether the customer has multiple lines (e.g., yes, no, no phone service). InternetService: Type of internet service the customer has (e.g., DSL, Fiber optic, No). OnlineSecurity: Whether the customer has online security add-on (e.g., yes, no, no internet service). OnlineBackup: Whether the customer has online backup add-on (e.g., yes, no, no internet service). DeviceProtection: Whether the customer has device protection add-on (e.g., yes, no, no internet service). TechSupport: Whether the customer has tech support add-on (e.g., yes, no, no internet service). StreamingTV: Whether the customer has streaming TV service (e.g., yes, no, no internet service). StreamingMovies: Whether the customer has streaming movies service (e.g., yes, no, no internet service). Contract: Type of contract the customer has (e.g., month-to-month, one year, two year). PaperlessBilling: Whether the customer has paperless billing (e.g., yes, no). PaymentMethod: Customer's payment method (e.g., electronic check, mailed check, bank transfer (automatic), credit card (automatic)). MonthlyCharges: Monthly charges the customer incurs. TotalCharges: Total charges the customer has incurred. Churn: Whether the customer has churned (e.g., yes, no).

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

[1]:
```
#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
```

```python
##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)
```

[41]:
```python
import os
os.chdir("C:\DKS\Machine_Learning\KNN_Classification")
data = pd.read_csv('Churn.csv')
data.sample(5)
```

[41]:
```
      customerID  gender  SeniorCitizen Partner Dependents  tenure  \
2530  0722-SVSFK  Female              0      No         No       7
1088  7029-RPUAV    Male              1     Yes         No      17
234   1984-GPTEH  Female              0      No         No      29
3633  3878-AVSOQ  Female              1      No         No       1
4351  6671-NGWON  Female              0      No         No       7

     PhoneService MultipleLines InternetService       OnlineSecurity  \
2530          Yes            No     Fiber optic                   No
1088          Yes           Yes     Fiber optic                   No
234           Yes           Yes              No  No internet service
3633          Yes            No     Fiber optic                   No
4351          Yes            No              No  No internet service

             OnlineBackup     DeviceProtection          TechSupport  \
2530                   No                  Yes                  Yes
1088                   No                  Yes                   No
234   No internet service  No internet service  No internet service
3633                   No                   No                   No
4351  No internet service  No internet service  No internet service

              StreamingTV      StreamingMovies        Contract  \
2530                  Yes                  Yes  Month-to-month
1088                  Yes                  Yes  Month-to-month
234   No internet service  No internet service  Month-to-month
3633                   No                   No  Month-to-month
4351  No internet service  No internet service  Month-to-month

     PaperlessBilling            PaymentMethod  MonthlyCharges TotalCharges  \
2530              Yes         Electronic check          100.40          715
1088              Yes  Credit card (automatic)          100.45      1622.45
234                No         Electronic check           25.15          702
3633              Yes         Electronic check           71.25        71.25
4351               No             Mailed check           20.35        150.6

     Churn
```

```
2530     No
1088    Yes
234      No
3633     No
4351     No
```

[42]:
```
data.drop('customerID', axis=1, inplace=True)
```

[43]:
```
data.dtypes
```

[43]:
```
gender                  object
SeniorCitizen            int64
Partner                 object
Dependents              object
tenure                   int64
PhoneService            object
MultipleLines           object
InternetService         object
OnlineSecurity          object
OnlineBackup            object
DeviceProtection        object
TechSupport             object
StreamingTV             object
StreamingMovies         object
Contract                object
PaperlessBilling        object
PaymentMethod           object
MonthlyCharges         float64
TotalCharges            object
Churn                   object
dtype: object
```

[44]:
```
data['SeniorCitizen'] = data['SeniorCitizen'].astype('object')
```

[45]:
```
data.dtypes
```

[45]:
```
gender                  object
SeniorCitizen           object
Partner                 object
Dependents              object
tenure                   int64
PhoneService            object
MultipleLines           object
InternetService         object
OnlineSecurity          object
OnlineBackup            object
DeviceProtection        object
```

```
TechSupport           object
StreamingTV           object
StreamingMovies       object
Contract              object
PaperlessBilling      object
PaymentMethod         object
MonthlyCharges       float64
TotalCharges          object
Churn                 object
dtype: object
```

[46]:
```
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
#errors='coerce' is a parameter that tells Pandas to handle errors by␣
↪converting problematic entries to NaN (Not a Number)
#instead of raising an error. This is helpful when there are non-numeric values␣
↪in the column.
```

[47]:
```
data.dtypes
```

[47]:
```
gender                object
SeniorCitizen         object
Partner               object
Dependents            object
tenure                 int64
PhoneService          object
MultipleLines         object
InternetService       object
OnlineSecurity        object
OnlineBackup          object
DeviceProtection      object
TechSupport           object
StreamingTV           object
StreamingMovies       object
Contract              object
PaperlessBilling      object
PaymentMethod         object
MonthlyCharges       float64
TotalCharges         float64
Churn                 object
dtype: object
```

[48]:
```
data.shape
```

[48]:
```
(7043, 20)
```

[49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7043 non-null   object
 1   SeniorCitizen     7043 non-null   object
 2   Partner           7043 non-null   object
 3   Dependents        7043 non-null   object
 4   tenure            7043 non-null   int64
 5   PhoneService      7043 non-null   object
 6   MultipleLines     7043 non-null   object
 7   InternetService   7043 non-null   object
 8   OnlineSecurity    7043 non-null   object
 9   OnlineBackup      7043 non-null   object
 10  DeviceProtection  7043 non-null   object
 11  TechSupport       7043 non-null   object
 12  StreamingTV       7043 non-null   object
 13  StreamingMovies   7043 non-null   object
 14  Contract          7043 non-null   object
 15  PaperlessBilling  7043 non-null   object
 16  PaymentMethod     7043 non-null   object
 17  MonthlyCharges    7043 non-null   float64
 18  TotalCharges      7032 non-null   float64
 19  Churn             7043 non-null   object
dtypes: float64(2), int64(1), object(17)
memory usage: 1.1+ MB
```

[50]:
```
data.describe().T
```

[50]:
```
                 count         mean          std    min     25%       50%  \
tenure          7043.0    32.371149    24.559481   0.00    9.00    29.000
MonthlyCharges  7043.0    64.761692    30.090047  18.25   35.50    70.350
TotalCharges    7032.0  2283.300441  2266.771362  18.80  401.45  1397.475

                      75%      max
tenure            55.0000    72.00
MonthlyCharges    89.8500   118.75
TotalCharges    3794.7375  8684.80
```

[51]:
```
Total_missing = data.isnull().sum().sort_values(ascending = False)
Total_missing
```

[51]:
```
TotalCharges        11
gender               0
SeniorCitizen        0
MonthlyCharges       0
```

```
PaymentMethod       0
PaperlessBilling    0
Contract            0
StreamingMovies     0
StreamingTV         0
TechSupport         0
DeviceProtection    0
OnlineBackup        0
OnlineSecurity      0
InternetService     0
MultipleLines       0
PhoneService        0
tenure              0
Dependents          0
Partner             0
Churn               0
dtype: int64
```

[52]:
```
data.dropna(axis=0, inplace=True)
```

[53]:
```
Total_missing = data.isnull().sum().sort_values(ascending = False)
Total_missing
```

[53]:
```
gender              0
SeniorCitizen       0
TotalCharges        0
MonthlyCharges      0
PaymentMethod       0
PaperlessBilling    0
Contract            0
StreamingMovies     0
StreamingTV         0
TechSupport         0
DeviceProtection    0
OnlineBackup        0
OnlineSecurity      0
InternetService     0
MultipleLines       0
PhoneService        0
tenure              0
Dependents          0
Partner             0
Churn               0
dtype: int64
```

[54]:
```
data.shape
```

[54]: (7032, 20)

[55]:
```
data.describe(include='object').T
```

[55]:

	count	unique	top	freq
gender	7032	2	Male	3549
SeniorCitizen	7032	2	0	5890
Partner	7032	2	No	3639
Dependents	7032	2	No	4933
PhoneService	7032	2	Yes	6352
MultipleLines	7032	3	No	3385
InternetService	7032	3	Fiber optic	3096
OnlineSecurity	7032	3	No	3497
OnlineBackup	7032	3	No	3087
DeviceProtection	7032	3	No	3094
TechSupport	7032	3	No	3472
StreamingTV	7032	3	No	2809
StreamingMovies	7032	3	No	2781
Contract	7032	3	Month-to-month	3875
PaperlessBilling	7032	2	Yes	4168
PaymentMethod	7032	4	Electronic check	2365
Churn	7032	2	No	5163

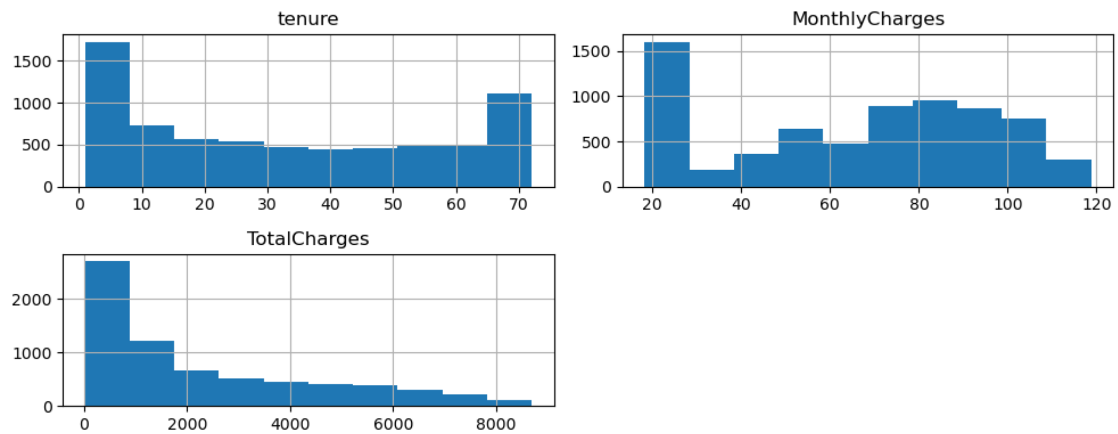
[56]:
```
data.describe().T
```

[56]:

	count	mean	std	min	25%	50%	\
tenure	7032.0	32.421786	24.545260	1.00	9.0000	29.000	
MonthlyCharges	7032.0	64.798208	30.085974	18.25	35.5875	70.350	
TotalCharges	7032.0	2283.300441	2266.771362	18.80	401.4500	1397.475	

	75%	max
tenure	55.0000	72.00
MonthlyCharges	89.8625	118.75
TotalCharges	3794.7375	8684.80

[57]:
```
data.hist()
plt.tight_layout()
plt.show()
```



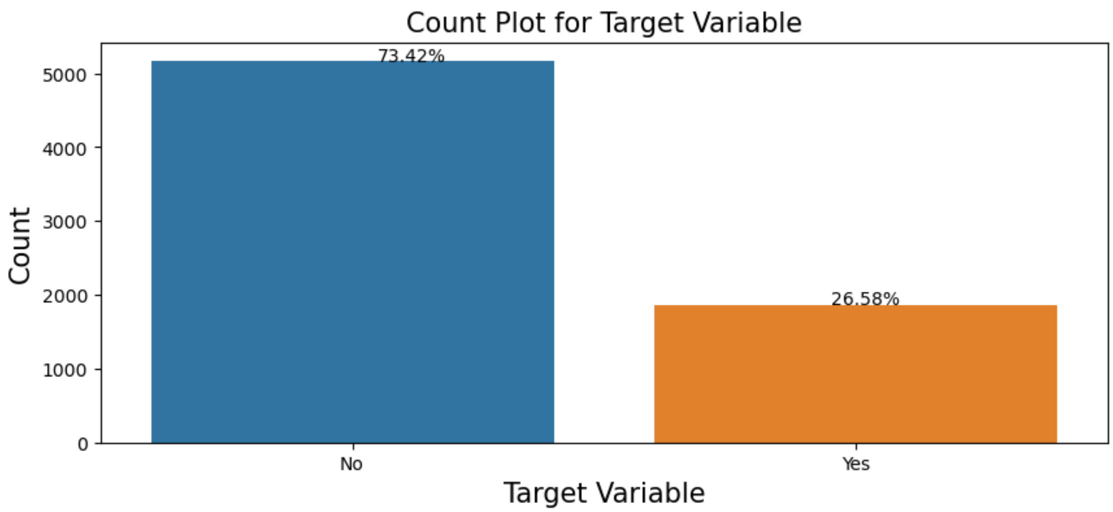
[58]:
```python
data_x = data.iloc[:, data.columns != 'Churn']
data_y = data.iloc[:,data.columns == 'Churn']
print(data_y.head(2))
print(data_x.head(2))
```

```
  Churn
0    No
1    No
   gender SeniorCitizen Partner Dependents  tenure PhoneService  \
0  Female             0     Yes         No       1           No
1    Male             0      No         No      34          Yes

      MultipleLines InternetService OnlineSecurity OnlineBackup  \
0  No phone service             DSL             No          Yes
1                No             DSL            Yes           No

  DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
0               No          No          No              No  Month-to-month
1              Yes          No          No              No        One year

  PaperlessBilling     PaymentMethod  MonthlyCharges  TotalCharges
0              Yes  Electronic check           29.85         29.85
1               No      Mailed check           56.95       1889.50
```

[59]:
```python
class_frequency =data_y.value_counts()
class_frequency
```

[59]:
```
Churn
No       5163
Yes      1869
dtype: int64
```

[60]:
```
sns.countplot(data=data_y,x ="Churn")
plt.text(x =0.05, y =data_y.value_counts()[0]+1, s =␣
 ↪str(round((class_frequency[0])*100/len(data_y),2)) + '%')
plt.text(x =0.95, y =data_y.value_counts()[1]+1, 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()
```

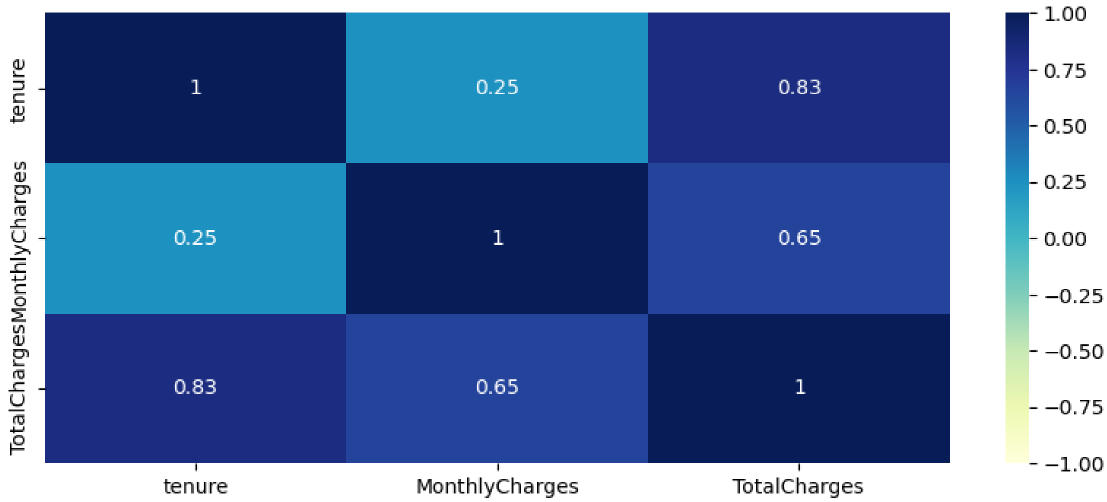


[61]:
```
corr = data_x.corr()
corr
```

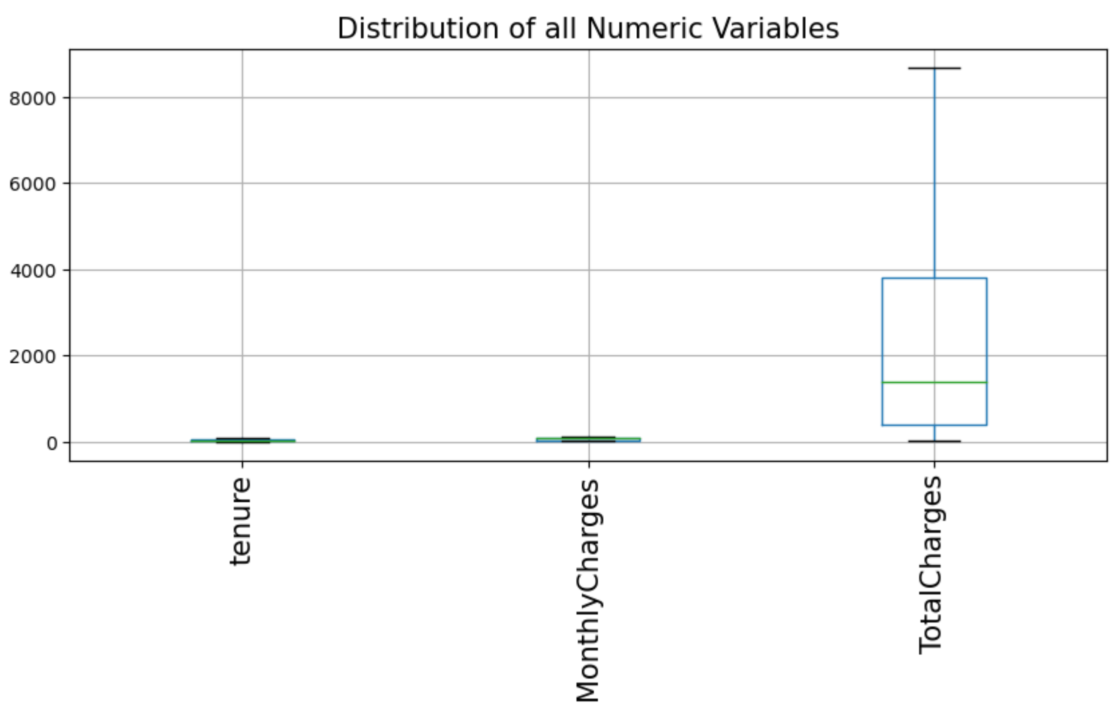
[61]:

	tenure	MonthlyCharges	TotalCharges
tenure	1.000000	0.246862	0.825880
MonthlyCharges	0.246862	1.000000	0.651065
TotalCharges	0.825880	0.651065	1.000000

[62]:
```
sns.heatmap(corr, cmap = 'YlGnBu', vmax = 1.0, vmin = -1.0, annot = True,␣
 ↪annot_kws = {"size": 10})
plt.show()
```

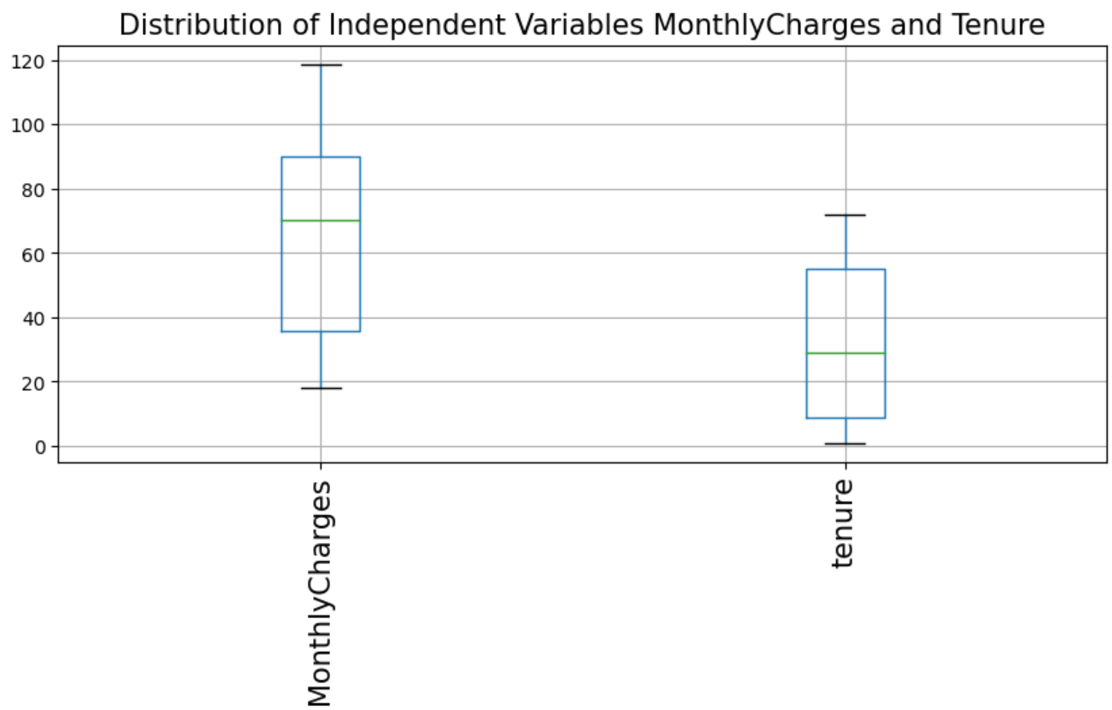


[63]:
```
data_x.boxplot()
plt.title('Distribution of all Numeric Variables', fontsize = 15)
plt.xticks(rotation = 'vertical', fontsize = 15)
plt.show()
```



[64]:
```
variables = [ 'MonthlyCharges','tenure']
data_x[variables].boxplot()
```

```
plt.title('Distribution of Independent Variables MonthlyCharges and Tenure',␣
  ↪fontsize = 15)
plt.xticks(rotation = 'vertical', fontsize = 15)
plt.show()
```



[65]:
```
data.head(2)
```

[65]:
```
   gender SeniorCitizen Partner Dependents  tenure PhoneService  \
0  Female             0     Yes         No       1           No
1    Male             0      No         No      34          Yes

      MultipleLines InternetService OnlineSecurity OnlineBackup  \
0  No phone service             DSL             No          Yes
1                No             DSL            Yes           No

  DeviceProtection TechSupport StreamingTV StreamingMovies        Contract  \
0               No          No          No              No  Month-to-month
1              Yes          No          No              No        One year

  PaperlessBilling     PaymentMethod  MonthlyCharges  TotalCharges Churn
0              Yes  Electronic check           29.85         29.85    No
1               No      Mailed check           56.95       1889.50    No
```

[66]:
```
data.replace(to_replace='No',value='0',inplace=True)
data.replace(to_replace='Yes',value='1',inplace=True)
data['Churn'] = pd.to_numeric(data['Churn'], errors='coerce')  #Target varialbe
data.dtypes
```

[66]:
```
gender                object
SeniorCitizen          int64
Partner               object
Dependents            object
tenure                 int64
PhoneService          object
MultipleLines         object
InternetService       object
OnlineSecurity        object
OnlineBackup          object
DeviceProtection      object
TechSupport           object
StreamingTV           object
StreamingMovies       object
Contract              object
PaperlessBilling      object
PaymentMethod         object
MonthlyCharges       float64
TotalCharges         float64
Churn                  int64
dtype: object
```

[67]:
```
data_numeric = data.select_dtypes(include=np.number)
print(data_numeric.columns)
data_categoric = data.select_dtypes(include = object)
print(data_categoric.columns)
```

```
Index(['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges', 'Churn'],
dtype='object')
Index(['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
       'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod'],
      dtype='object')
```

[68]:
```
from sklearn.preprocessing import LabelEncoder
label_encoders = {}
for column in data_categoric.columns:
    label_encoders[column] = LabelEncoder()
    data_categoric[column] = label_encoders[column].
 ↪fit_transform(data_categoric[column])
data_categoric.head()
```

```
data_categoric.dtypes
```

[68]:
```
gender              int32
Partner             int32
Dependents          int32
PhoneService        int32
MultipleLines       int32
InternetService     int32
OnlineSecurity      int32
OnlineBackup        int32
DeviceProtection    int32
TechSupport         int32
StreamingTV         int32
StreamingMovies     int32
Contract            int32
PaperlessBilling    int32
PaymentMethod       int32
dtype: object
```

[69]:
```
for variable in data_categoric.columns:
    data_categoric[variable] = data_categoric[variable].astype('object')
```

[70]:
```
dummy_variables = pd.get_dummies(data_categoric, drop_first = True)
```

[71]:
```
data_dummy = pd.concat([data_numeric, dummy_variables], axis=1)
data_dummy.head()
```

[71]:
```
   SeniorCitizen  tenure  MonthlyCharges  TotalCharges  Churn  gender_1  \
0              0       1           29.85         29.85      0         0
1              0      34           56.95       1889.50      0         1
2              0       2           53.85        108.15      1         1
3              0      45           42.30       1840.75      0         1
4              0       2           70.70        151.65      1         0

   Partner_1  Dependents_1  PhoneService_1  MultipleLines_1  MultipleLines_2  \
0          1             0               0                0                1
1          0             0               1                0                0
2          0             0               1                0                0
3          0             0               0                0                1
4          0             0               1                0                0

   InternetService_1  InternetService_2  OnlineSecurity_1  OnlineSecurity_2  \
0                  1                  0                 0                 0
1                  1                  0                 1                 0
2                  1                  0                 1                 0
3                  1                  0                 1                 0
4                  0                  1                 0                 0
```

```
   OnlineBackup_1  OnlineBackup_2  DeviceProtection_1  DeviceProtection_2  \
0               1               0                   0                   0
1               0               0                   1                   0
2               1               0                   0                   0
3               0               0                   1                   0
4               0               0                   0                   0

   TechSupport_1  TechSupport_2  StreamingTV_1  StreamingTV_2  \
0              0              0              0              0
1              0              0              0              0
2              0              0              0              0
3              1              0              0              0
4              0              0              0              0

   StreamingMovies_1  StreamingMovies_2  Contract_1  Contract_2  \
0                  0                  0           0           0
1                  0                  0           1           0
2                  0                  0           0           0
3                  0                  0           1           0
4                  0                  0           0           0

   PaperlessBilling_1  PaymentMethod_1  PaymentMethod_2  PaymentMethod_3
0                   1                0                1                0
1                   0                0                0                1
2                   1                0                0                1
3                   0                0                0                0
4                   1                0                1                0
```

[72]:
```
data_dummy.shape
```

[72]:
```
(7032, 31)
```

[109]:
```
X = data_dummy.drop(['Churn'], axis = 1)

y =data_dummy.Churn
```

[110]:
```
X = data_dummy.drop(['Churn'], axis = 1)

y =data_dummy.Churn
#X_Scale=X.apply(lambda x:(x-x.mean())/x.std())
#print(X_Scale.head())
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_Scale = scaler.transform(X)
```

```python
# Retrieve column names from the original DataFrame X
column_names = X.columns

# Create a new DataFrame using the scaled data and column names
X_scaled_df = pd.DataFrame(X_Scale, columns=column_names)
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size =
    0.3, random_state = 1)
print(X_train.head())
```

```
      SeniorCitizen    tenure  MonthlyCharges  TotalCharges  gender_1  \
1579            1.0  0.901408        0.350746      0.403773       0.0
1040            0.0  0.436620        0.512438      0.268763       1.0
1074            0.0  0.563380        0.957711      0.520269       0.0
2473            0.0  0.042254        0.261692      0.023304       1.0
6897            0.0  0.112676        0.369154      0.049729       1.0

      Partner_1  Dependents_1  PhoneService_1  MultipleLines_1  \
1579        0.0           0.0             0.0              0.0
1040        1.0           1.0             1.0              0.0
1074        0.0           0.0             1.0              1.0
2473        0.0           0.0             1.0              0.0
6897        0.0           1.0             1.0              0.0

      MultipleLines_2  InternetService_1  InternetService_2  OnlineSecurity_1  \
1579              1.0                1.0                0.0               0.0
1040              0.0                0.0                1.0               0.0
1074              0.0                0.0                1.0               1.0
2473              0.0                1.0                0.0               0.0
6897              0.0                1.0                0.0               0.0

      OnlineSecurity_2  OnlineBackup_1  OnlineBackup_2  DeviceProtection_1  \
1579               0.0             1.0             0.0                 1.0
1040               0.0             0.0             0.0                 0.0
1074               0.0             1.0             0.0                 1.0
2473               0.0             0.0             0.0                 0.0
6897               0.0             0.0             0.0                 0.0

      DeviceProtection_2  TechSupport_1  TechSupport_2  StreamingTV_1  \
1579                 0.0            0.0            0.0            1.0
1040                 0.0            0.0            0.0            0.0
1074                 0.0            1.0            0.0            1.0
2473                 0.0            0.0            0.0            0.0
6897                 0.0            0.0            0.0            1.0

      StreamingTV_2  StreamingMovies_1  StreamingMovies_2  Contract_1  \
1579            0.0                1.0                0.0         0.0
1040            0.0                0.0                0.0         0.0
```

```
1074           0.0                 1.0                 0.0           0.0
2473           0.0                 0.0                 0.0           0.0
6897           0.0                 0.0                 0.0           0.0

      Contract_2  PaperlessBilling_1  PaymentMethod_1  PaymentMethod_2  \
1579         1.0                 1.0              0.0              0.0
1040         0.0                 1.0              0.0              0.0
1074         0.0                 1.0              0.0              0.0
2473         0.0                 0.0              0.0              0.0
6897         0.0                 0.0              0.0              1.0

      PaymentMethod_3
1579              0.0
1040              0.0
1074              0.0
2473              1.0
6897              0.0
```

[79]:
```
def get_test_report(model):
    return(classification_report(y_test,y_pred))
```

[80]:
```
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()
```

[81]:
```
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)
```

[82]:
```
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)
```

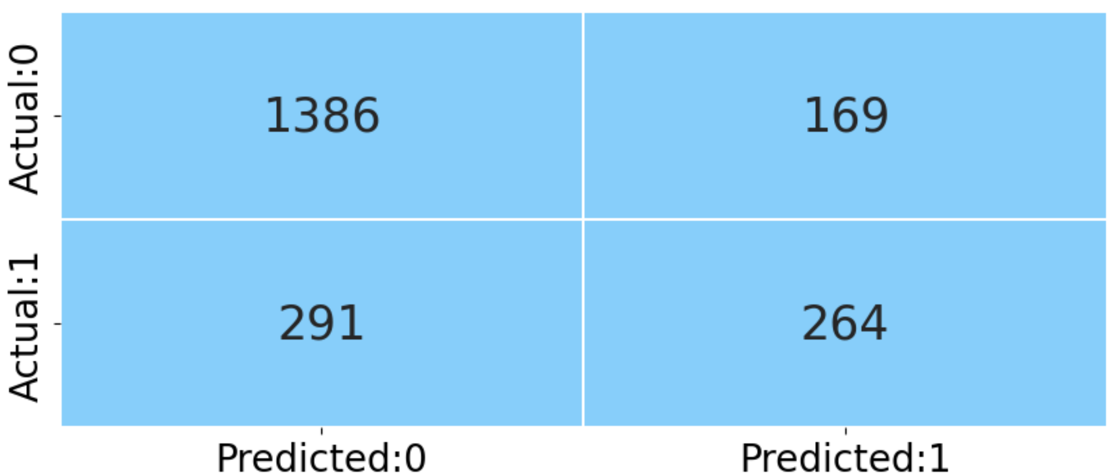
[83]:
```
from sklearn.ensemble import RandomForestClassifier
#intantiate the regressor
rf_cls = RandomForestClassifier(n_estimators=100, random_state=10)

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

[83]: RandomForestClassifier(random_state=10)

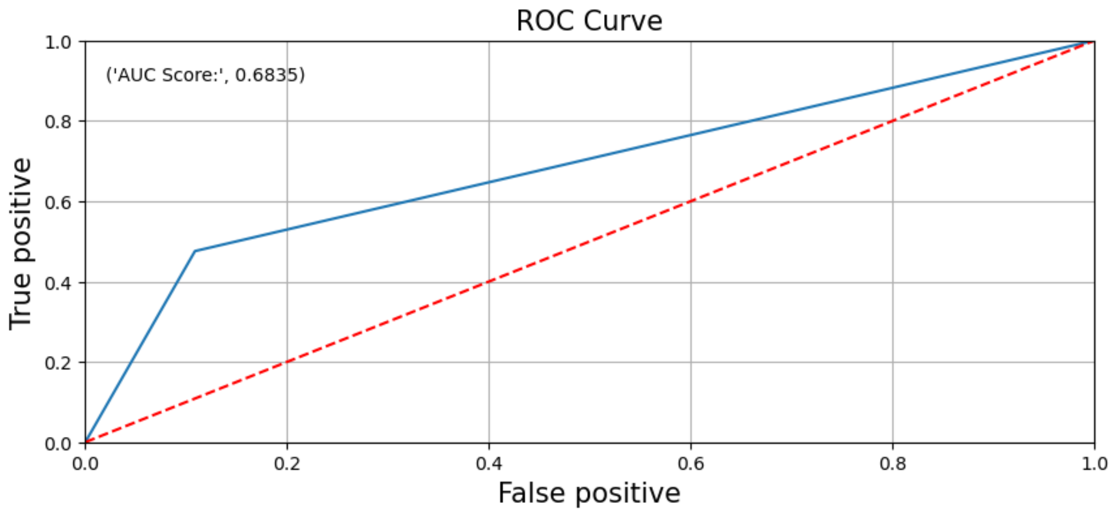
[84]:
```
y_pred = rf_cls.predict(X_test)
y_pred
```

[84]: array([0, 1, 0, …, 0, 1, 0], dtype=int64)

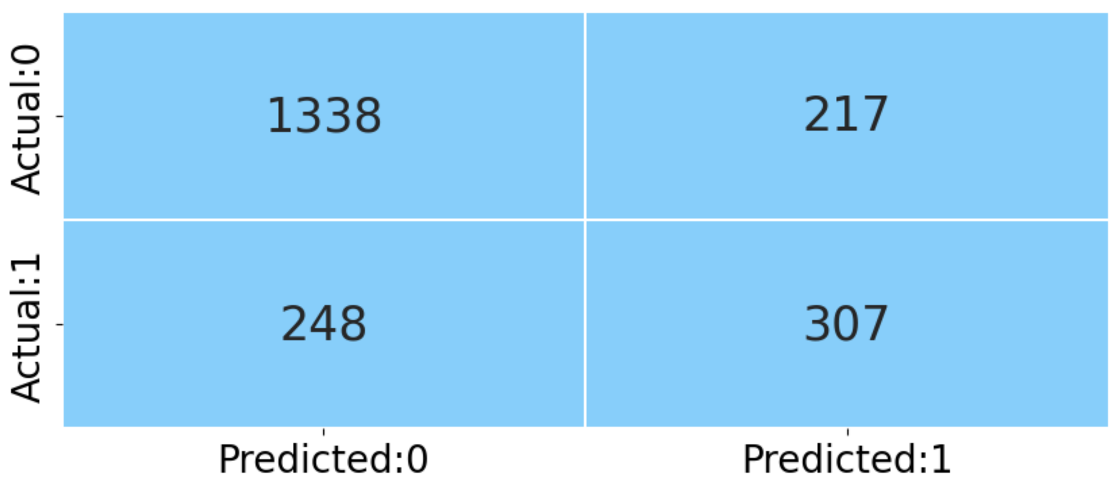
[85]:
```
plot_confusion_matrix(rf_cls)
plot_roc(rf_cls)
update_score_card(model_name="rf_cls")
```



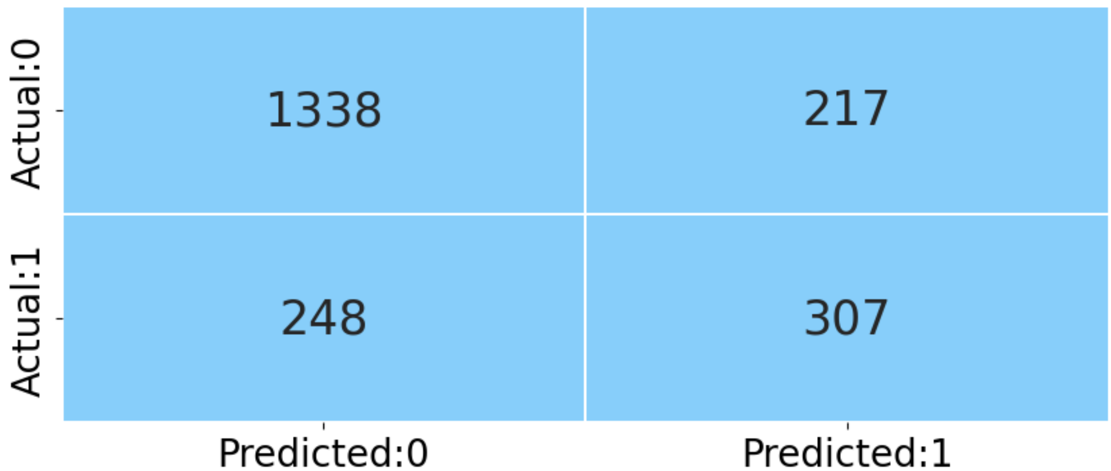
```
[85]:     Model  AUC Score  Precision Score  Recall Score  Accuracy Score  \
      0  rf_cls   0.683497           0.6097      0.781991        0.781991

         Kappa Score  f1-Score
      0     0.394907  0.534413
```



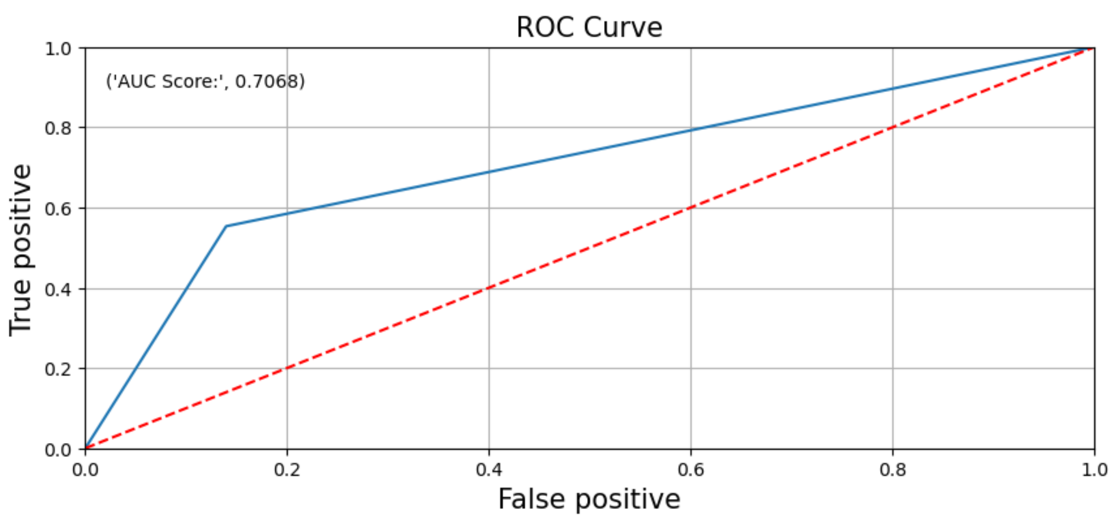

```
[86]: from sklearn.neighbors import KNeighborsClassifier
      from sklearn.model_selection import GridSearchCV
      from sklearn.model_selection import cross_val_score
      KN_Classifier=KNeighborsClassifier(n_neighbors=29)
      KN_Classifier_st=KN_Classifier.fit(X_train, y_train)
      y_pred_prob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



[87]:
```
plot_confusion_matrix(KN_Classifier_st)
plot_roc(KN_Classifier_st)
update_score_card(model_name="KN_Classifier_st")
```



[87]:
```
              Model  AUC Score  Precision Score  Recall Score  Accuracy Score  \
0            rf_cls   0.683497         0.609700      0.781991        0.781991
1  KN_Classifier_st   0.706802         0.585878      0.779621        0.779621

   Kappa Score  f1-Score
0     0.394907  0.534413
1     0.421168  0.569045
```



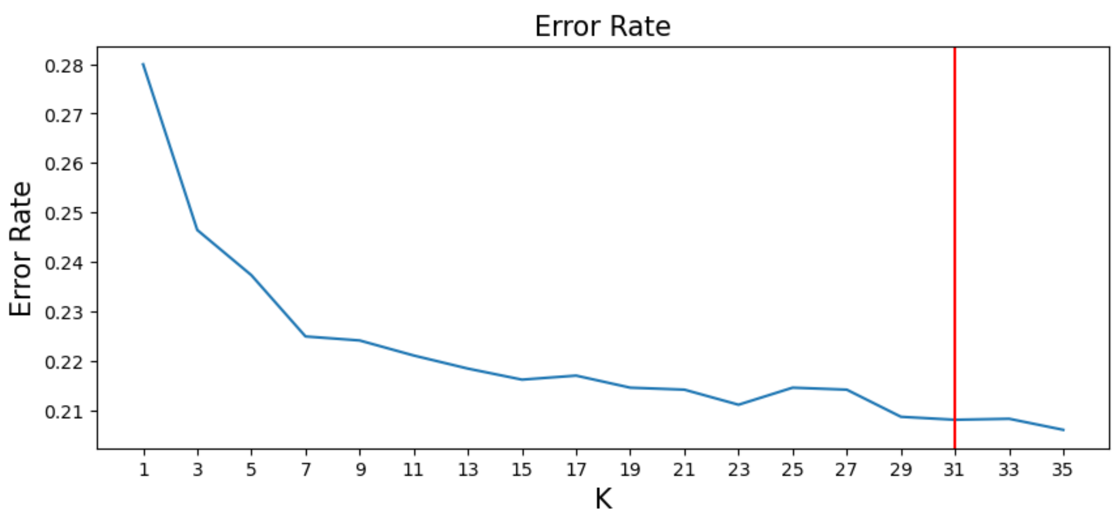
[88]:

```
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': 'manhattan', 'n_neighbors': 35}
```

[89]:

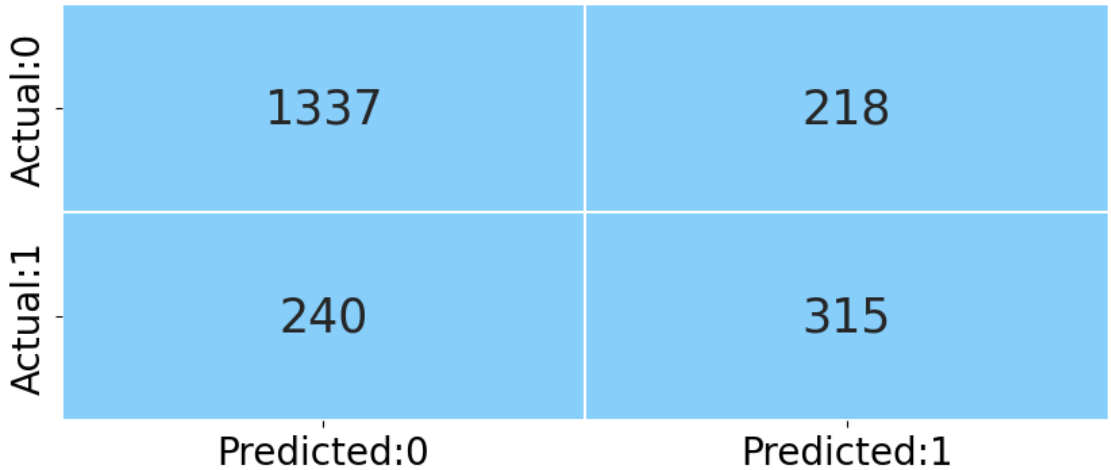
```
error_rate = []
for i in np.arange(1,37,2):
    knn = KNeighborsClassifier(i, metric = 'manhattan')
    score = cross_val_score(knn, X_train, y_train, cv = 5)
    score = score.mean()
    error_rate.append(1 - score)
plt.plot(range(1,37,2), error_rate)
plt.title('Error Rate', fontsize = 15)
plt.xlabel('K', fontsize = 15)
plt.ylabel('Error Rate', fontsize = 15)
plt.xticks(np.arange(1, 37, step = 2))
plt.axvline(x = 31, color = 'red')
plt.show()
```



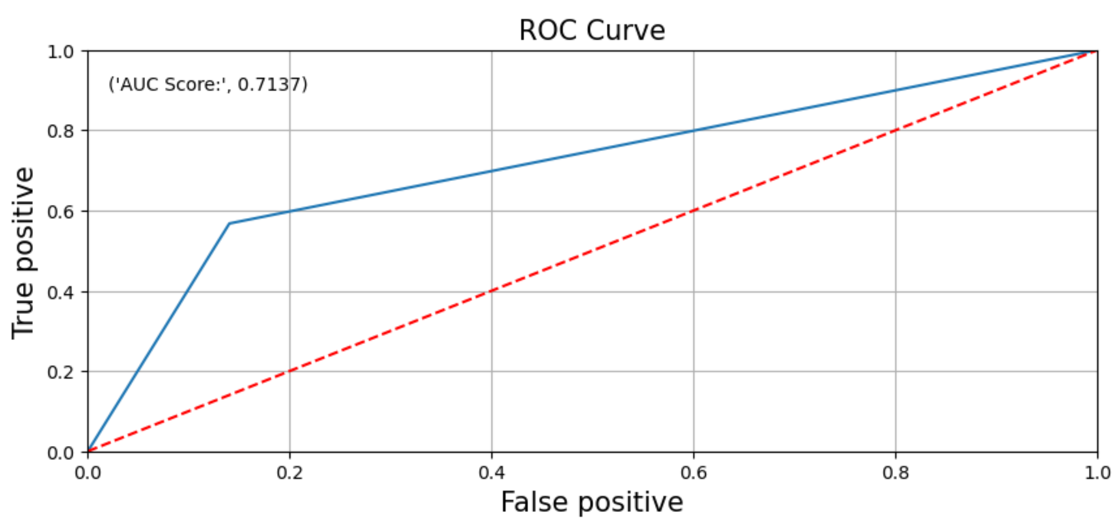
[90]:

```
KN_Classifier=KNeighborsClassifier(n_neighbors=31,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)
```

[91]:
```
plot_confusion_matrix(KN_Classifier_tunning)
plot_roc(KN_Classifier_tunning)
update_score_card(model_name="KN_Classifier_tunning")
```

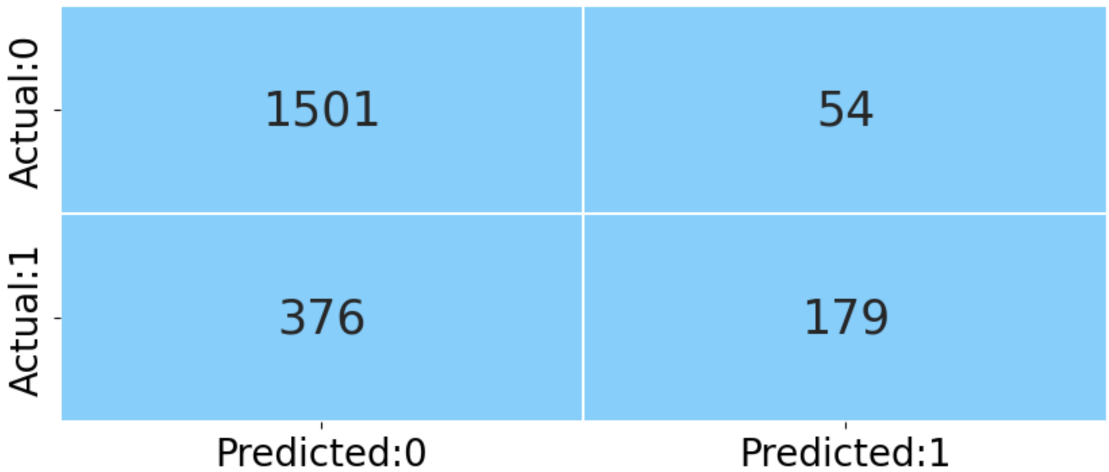


[91]:
```
                    Model  AUC Score  Precision Score  Recall Score  \
0                  rf_cls   0.683497         0.609700      0.781991
1        KN_Classifier_st   0.706802         0.585878      0.779621
2   KN_Classifier_tunning   0.713687         0.590994      0.782938

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
```

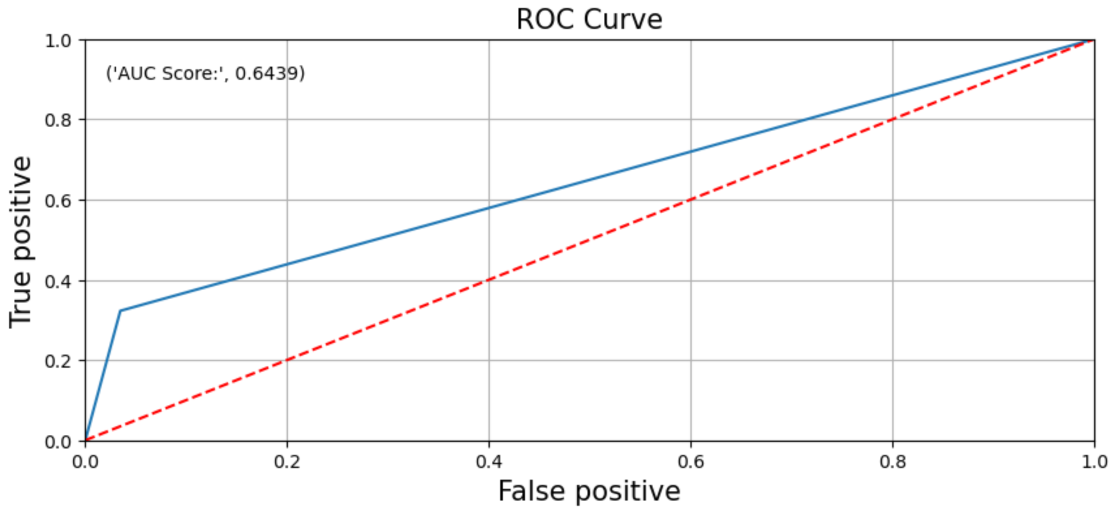


[92]:
```
from xgboost.sklearn import XGBClassifier
xgbm=XGBClassifier(random_state=1,learning_rate=0.01)
xgbm.fit(X_train,y_train)
y_pred =xgbm .predict(X_test)
```

[93]:
```
plot_confusion_matrix(xgbm)
plot_roc(xgbm)
update_score_card(model_name="xgbm")
```



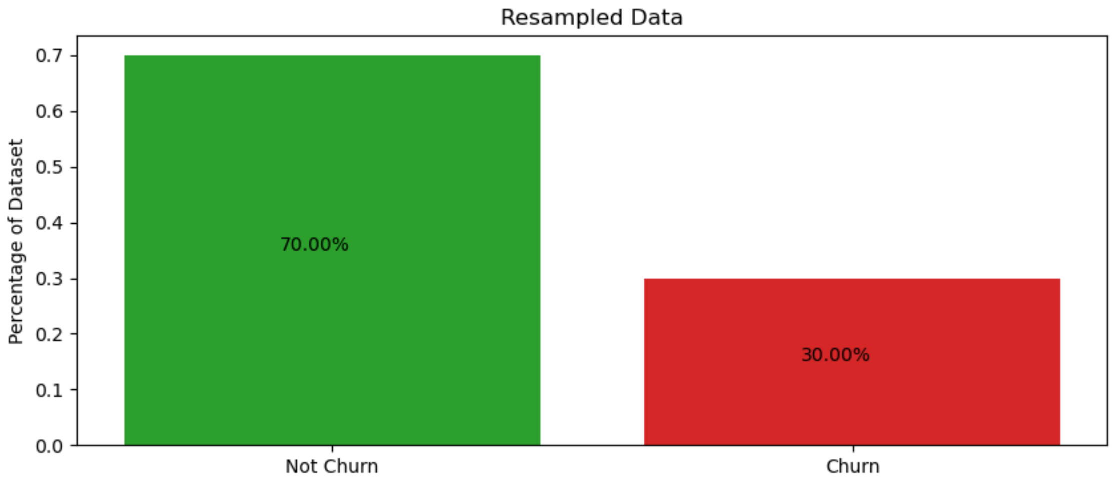
[93]:
```
                    Model  AUC Score  Precision Score  Recall Score  \
0                  rf_cls   0.683497         0.609700      0.781991
1        KN_Classifier_st   0.706802         0.585878      0.779621
2  KN_Classifier_tunning   0.713687         0.590994      0.782938
3                    xgbm   0.643898         0.768240      0.796209

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
3        0.796209     0.353798  0.454315
```



[94]:
```
y = data_dummy['Churn']
X =data_dummy.drop(['Churn'], axis=1)
```

[95]:
```
from imblearn.under_sampling import RandomUnderSampler
```

[97]:
```
rus = RandomUnderSampler(sampling_strategy=(3/7), random_state=0)
rus_X_train, rus_y_train = rus.fit_resample(X_train, y_train)

fig, ax = plt.subplots()
client = ['Not Churn','Churn']
proportions = rus_y_train.value_counts(normalize=True)
bar_colors = ['tab:green', 'tab:red']
ax.bar(client, proportions, color=bar_colors)
ax.set_ylabel('Percentage of Dataset')
ax.set_title('Resampled Data')
ax.text(1-0.1, proportions[1]/2, '{:.2%}'.format(proportions[1]), size=10)
ax.text(0-0.1, proportions[0]/2, '{:.2%}'.format(proportions[0]), size=10)

fig.show()
```

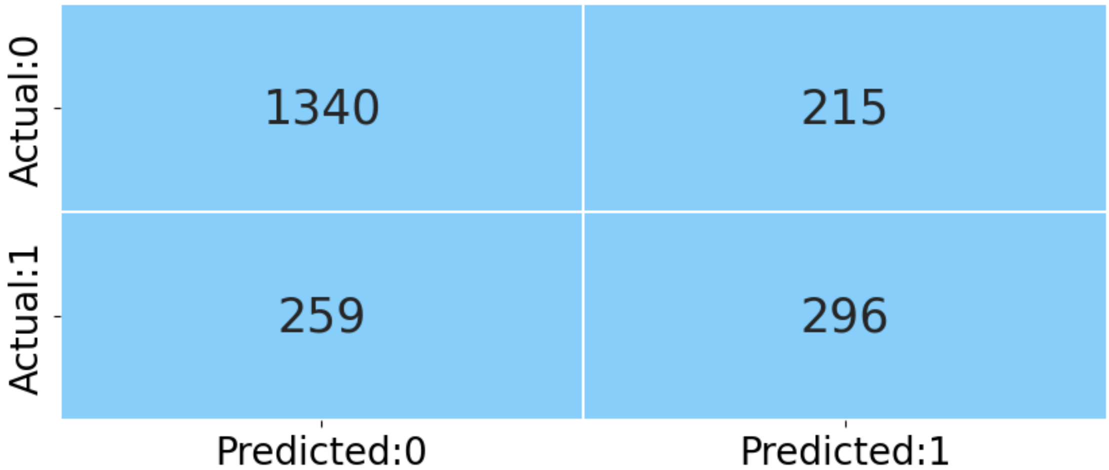


[98]:
```
rus_rf = RandomForestClassifier( n_estimators = 1000, max_features =12,␣
 ↪random_state=0)
rus_rf.fit(rus_X_train, rus_y_train)
```

[98]:
```
RandomForestClassifier(max_features=12, n_estimators=1000, random_state=0)
```

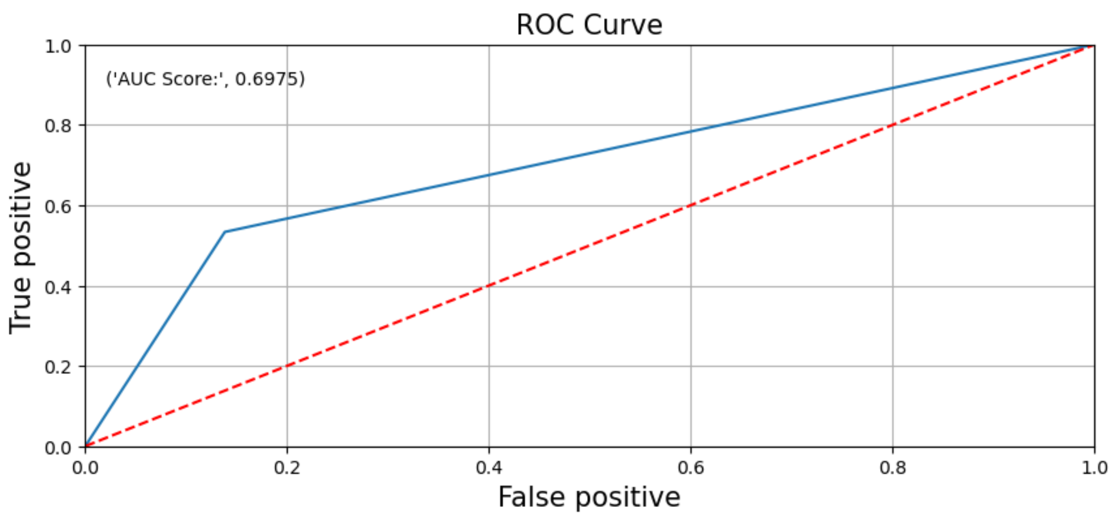
[99]:
```
y_pred = rus_rf.predict(X_test)
```

[100]:
```
plot_confusion_matrix(rus_rf)
plot_roc(rus_rf)
update_score_card(model_name="rus_rf")
```



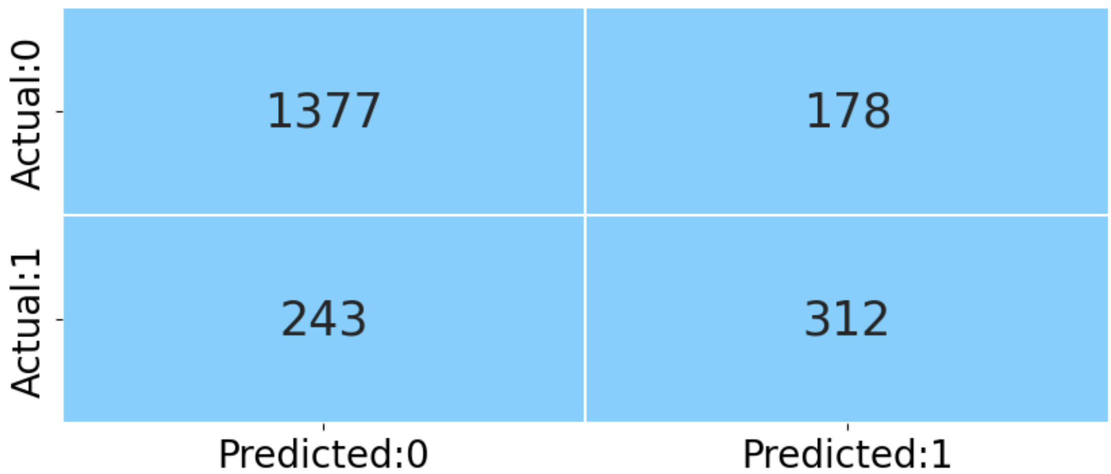
[100]:

```
                   Model  AUC Score  Precision Score  Recall Score  \
0                 rf_cls   0.683497         0.609700      0.781991
1        KN_Classifier_st   0.706802         0.585878      0.779621
2  KN_Classifier_tunning   0.713687         0.590994      0.782938
3                   xgbm   0.643898         0.768240      0.796209
4                 rus_rf   0.697535         0.579256      0.775355

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
3        0.796209     0.353798  0.454315
4        0.775355     0.405404  0.555347
```



[102]:

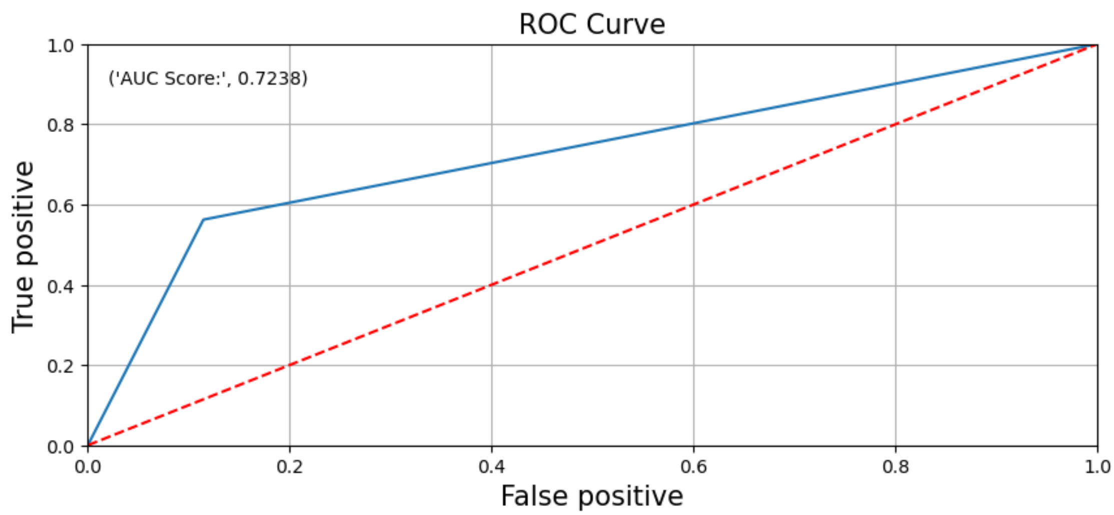
```
svc_linear = SVC(kernel='linear', probability=True)  # Specify␣
↪'probability=True' to enable probability estimates
svm_linear=svc_linear.fit(X_train, y_train)
y_pred_prob =svm_linear.predict_proba(X_test)[:,1]
y_pred =svm_linear .predict(X_test)
plot_confusion_matrix(svm_linear)
test_report = get_test_report(svm_linear)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



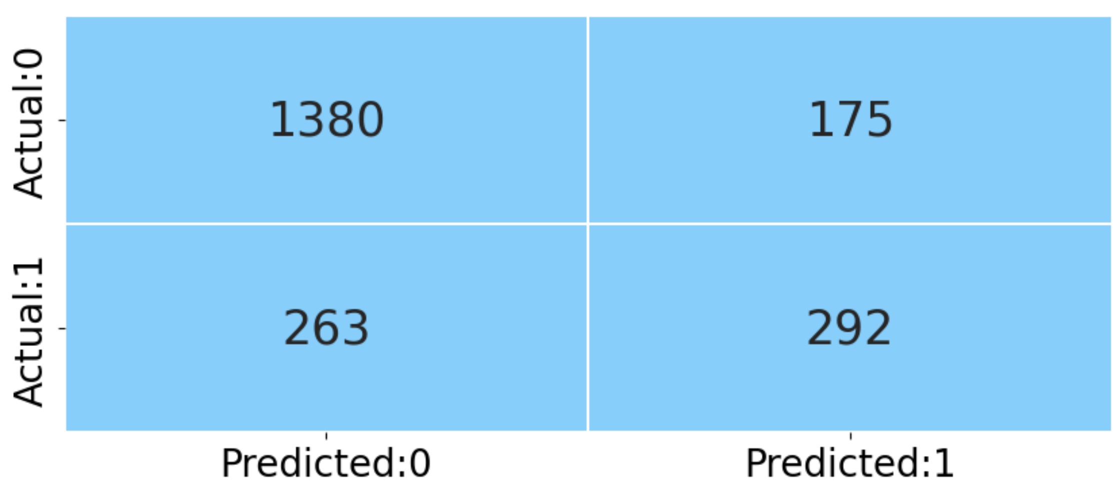

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1555
           1       0.64      0.56      0.60       555

    accuracy                           0.80      2110
   macro avg       0.74      0.72      0.73      2110
weighted avg       0.79      0.80      0.80      2110
```

[102]:

```
                  Model  AUC Score  Precision Score  Recall Score  \
0                rf_cls   0.683497         0.609700      0.781991
1      KN_Classifier_st   0.706802         0.585878      0.779621
2  KN_Classifier_tunning  0.713687         0.590994      0.782938
3                  xgbm   0.643898         0.768240      0.796209
4                rus_rf   0.697535         0.579256      0.775355
5            svm_linear   0.723846         0.636735      0.800474

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
3        0.796209     0.353798  0.454315
4        0.775355     0.405404  0.555347
5        0.800474     0.465212  0.597129
```



[103]:
```
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')
```

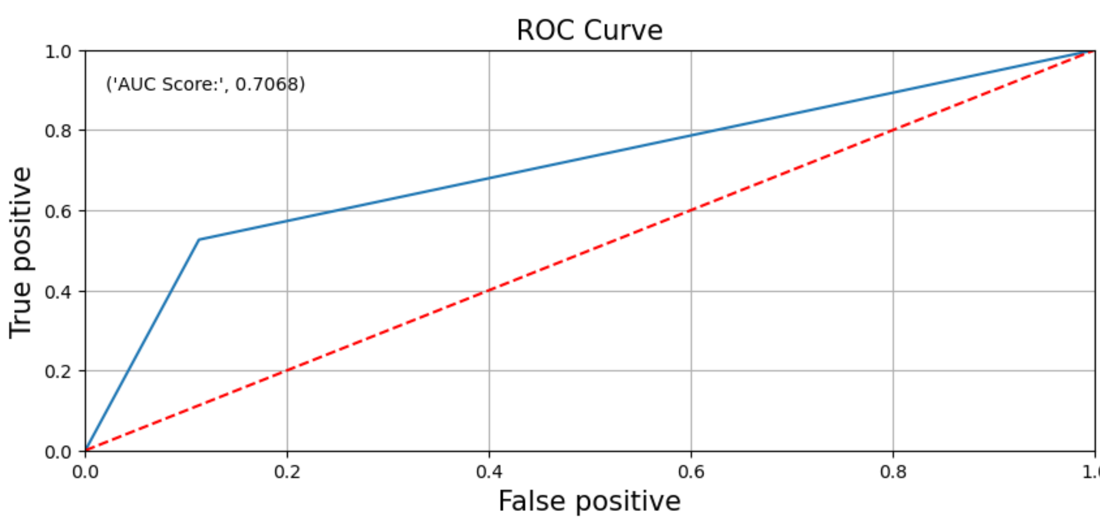



```
              precision    recall  f1-score   support
```

```
           0       0.84      0.89      0.86      1555
           1       0.63      0.53      0.57       555

    accuracy                           0.79      2110
   macro avg       0.73      0.71      0.72      2110
weighted avg       0.78      0.79      0.79      2110
```

[103]:

```
                    Model  AUC Score  Precision Score  Recall Score  \
0                  rf_cls   0.683497         0.609700      0.781991
1        KN_Classifier_st   0.706802         0.585878      0.779621
2  KN_Classifier_tunning   0.713687         0.590994      0.782938
3                    xgbm   0.643898         0.768240      0.796209
4                  rus_rf   0.697535         0.579256      0.775355
5              svm_linear   0.723846         0.636735      0.800474
6                svm_poly   0.706793         0.625268      0.792417

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
3        0.796209     0.353798  0.454315
4        0.775355     0.405404  0.555347
5        0.800474     0.465212  0.597129
6        0.792417     0.435805  0.571429
```

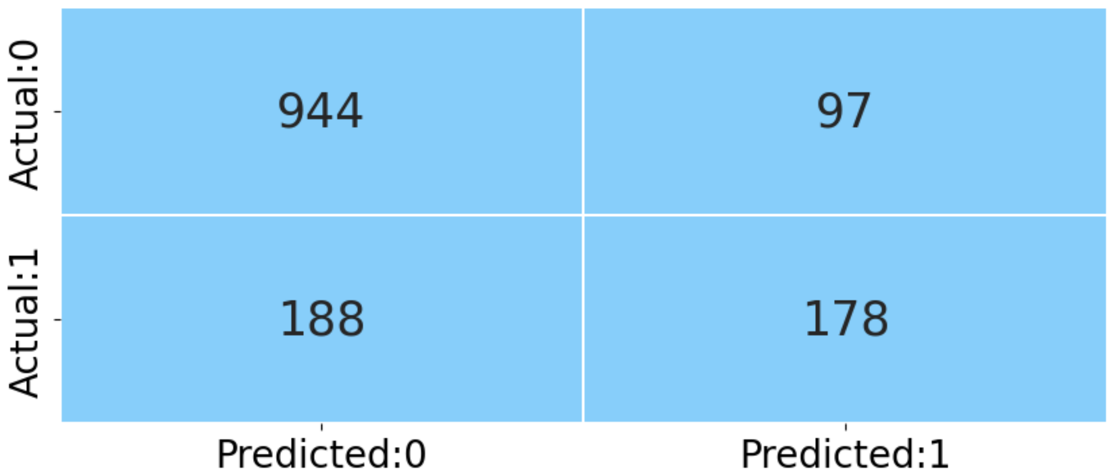


[111]:

```python
X = data_dummy.drop(['Churn'], axis = 1)
y = pd.DataFrame(data_dummy['Churn'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,␣
↪random_state = 1)
```

```python
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],
    'bootstrap': [True, False]
}

# 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")
```

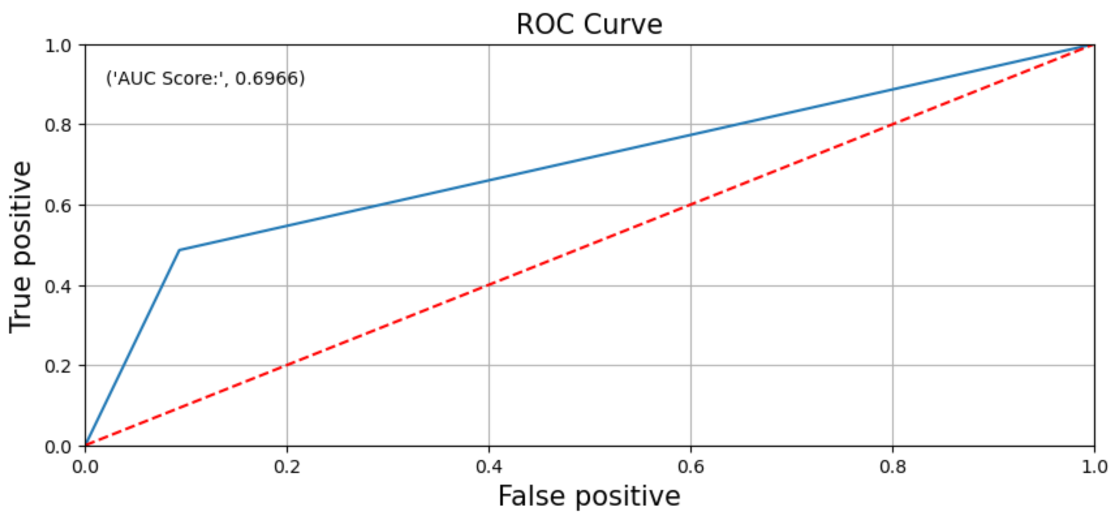
```
Best Parameters:  {'bootstrap': True, 'max_depth': 30, 'max_features': 'auto',
'min_samples_leaf': 4, 'min_samples_split': 2, 'n_estimators': 300}
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1041
           1       0.65      0.49      0.56       366

    accuracy                           0.80      1407
   macro avg       0.74      0.70      0.71      1407
weighted avg       0.79      0.80      0.79      1407
```



[111]:

	Model	AUC Score	Precision Score	Recall Score	\
0	rf_cls	0.683497	0.609700	0.781991	
1	KN_Classifier_st	0.706802	0.585878	0.779621	
2	KN_Classifier_tunning	0.713687	0.590994	0.782938	
3	xgbm	0.643898	0.768240	0.796209	
4	rus_rf	0.697535	0.579256	0.775355	
5	svm_linear	0.723846	0.636735	0.800474	
6	svm_poly	0.706793	0.625268	0.792417	
7	Hyper_Parameter_RF	0.696580	0.647273	0.797441	

	Accuracy Score	Kappa Score	f1-Score
0	0.781991	0.394907	0.534413
1	0.779621	0.421168	0.569045
2	0.782938	0.432892	0.579044
3	0.796209	0.353798	0.454315
4	0.775355	0.405404	0.555347
5	0.800474	0.465212	0.597129
6	0.792417	0.435805	0.571429
7	0.797441	0.427630	0.555382



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

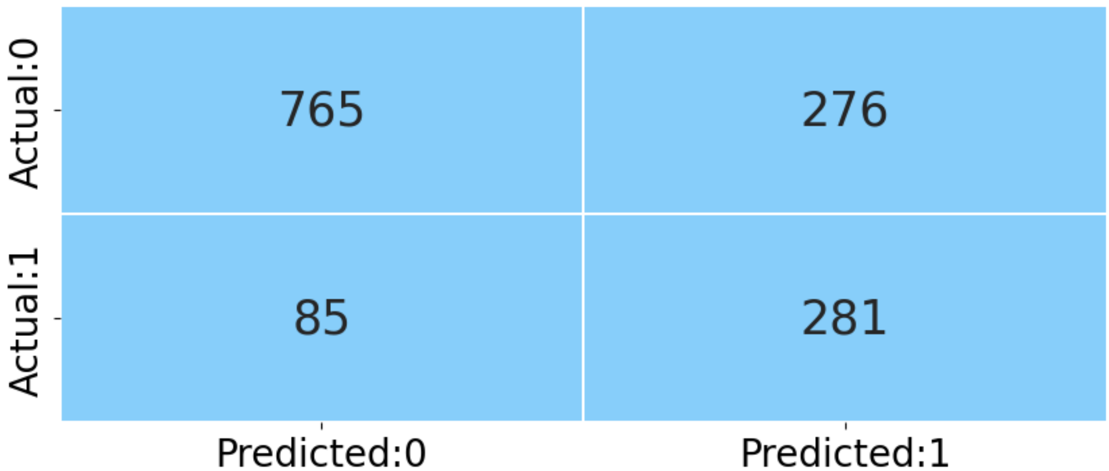
[112]:
```
from imblearn.under_sampling import RandomUnderSampler

# 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       0.90      0.73      0.81      1041
           1       0.50      0.77      0.61       366
```

```
    accuracy                           0.74      1407
   macro avg       0.70      0.75      0.71      1407
weighted avg       0.80      0.74      0.76      1407
```

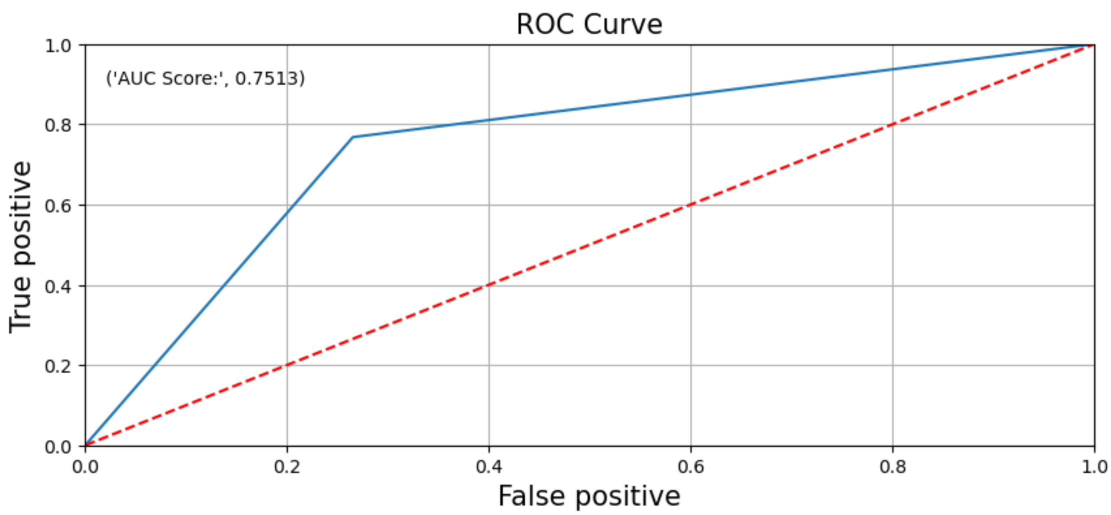
[113]:

```
plot_confusion_matrix(model_random_forest_undersample)
plot_roc(model_random_forest_undersample)
update_score_card(model_name="Random_forest_undersample")
```



[113]:

```
                       Model  AUC Score  Precision Score  Recall Score  \
0                     rf_cls   0.683497         0.609700      0.781991
1          KN_Classifier_st   0.706802         0.585878      0.779621
2      KN_Classifier_tunning   0.713687         0.590994      0.782938
3                       xgbm   0.643898         0.768240      0.796209
4                     rus_rf   0.697535         0.579256      0.775355
5                 svm_linear   0.723846         0.636735      0.800474
6                   svm_poly   0.706793         0.625268      0.792417
7         Hyper_Parameter_RF   0.696580         0.647273      0.797441
8  Random_forest_undersample   0.751315         0.504488      0.743426

   Accuracy Score  Kappa Score  f1-Score
0        0.781991     0.394907  0.534413
1        0.779621     0.421168  0.569045
2        0.782938     0.432892  0.579044
3        0.796209     0.353798  0.454315
4        0.775355     0.405404  0.555347
5        0.800474     0.465212  0.597129
6        0.792417     0.435805  0.571429
7        0.797441     0.427630  0.555382
8        0.743426     0.429896  0.608884
```



7 Feature Selection Using Random Forest Technique

[114]:
```
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
importances = rf_model.feature_importances_
importances
```

[114]:
```
array([0.02168385, 0.16662233, 0.16722656, 0.19605775, 0.02882377,
       0.02367736, 0.02117196, 0.0040623 , 0.02001794, 0.00403584,
       0.01484613, 0.02828419, 0.02474572, 0.00710689, 0.02186342,
       0.00563107, 0.01965557, 0.00483872, 0.02399968, 0.00702627,
       0.01665279, 0.00497943, 0.01823195, 0.00775594, 0.02093646,
       0.02776971, 0.02648933, 0.01386343, 0.03908962, 0.01285404])
```

[115]:
```
feature_names = X_train.columns.tolist()
print(feature_names)
```

```
['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges', 'gender_1',
'Partner_1', 'Dependents_1', 'PhoneService_1', 'MultipleLines_1',
'MultipleLines_2', 'InternetService_1', 'InternetService_2', 'OnlineSecurity_1',
'OnlineSecurity_2', 'OnlineBackup_1', 'OnlineBackup_2', 'DeviceProtection_1',
'DeviceProtection_2', 'TechSupport_1', 'TechSupport_2', 'StreamingTV_1',
'StreamingTV_2', 'StreamingMovies_1', 'StreamingMovies_2', 'Contract_1',
'Contract_2', 'PaperlessBilling_1', 'PaymentMethod_1', 'PaymentMethod_2',
'PaymentMethod_3']
```

[117]:
```
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance':␣
↪importances})
```

```
feature_importance_df = feature_importance_df.sort_values(by='Importance',␣
↪ascending=False)
feature_importance_df.head(15)
```

[117]:

	Feature	Importance
3	TotalCharges	0.196058
2	MonthlyCharges	0.167227
1	tenure	0.166622
28	PaymentMethod_2	0.039090
4	gender_1	0.028824
11	InternetService_2	0.028284
25	Contract_2	0.027770
26	PaperlessBilling_1	0.026489
12	OnlineSecurity_1	0.024746
18	TechSupport_1	0.024000
5	Partner_1	0.023677
14	OnlineBackup_1	0.021863
0	SeniorCitizen	0.021684
6	Dependents_1	0.021172
24	Contract_1	0.020936

[118]:

```
# Select top 'n' features or based on a threshold
selected_features = feature_importance_df[feature_importance_df['Importance']␣
↪>= 0.025]['Feature'].tolist()
selected_features =list(selected_features)
selected_features
```

[118]:

```
['TotalCharges',
 'MonthlyCharges',
 'tenure',
 'PaymentMethod_2',
 'gender_1',
 'InternetService_2',
 'Contract_2',
 'PaperlessBilling_1']
```

[119]:

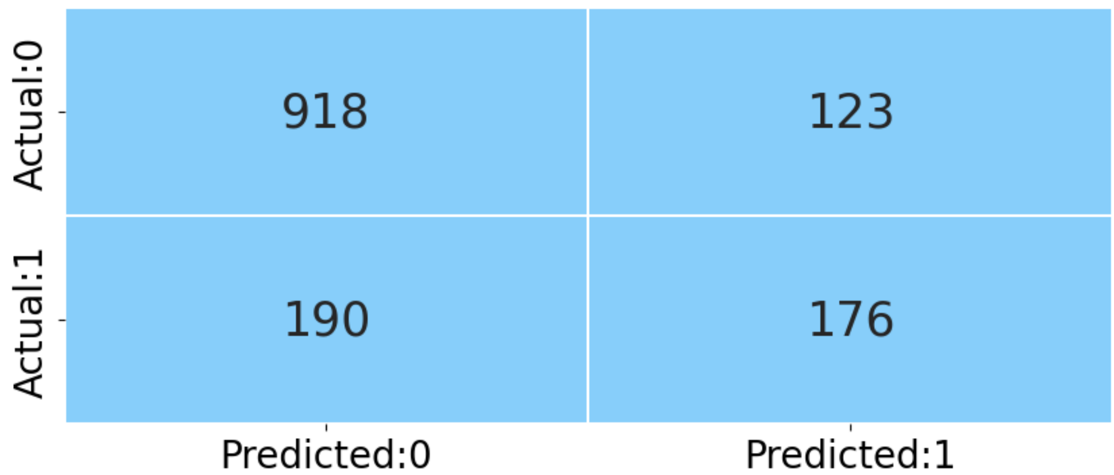
```
X_train_n=X_train[selected_features]
X_test_n=X_test[selected_features]
```

[121]:

```
#intantiate the regressor
Random_Forest_Features_Selection = RandomForestClassifier(n_estimators=100,␣
↪random_state=10)

# fit the regressor with training dataset
Random_Forest_Features_Selection.fit(X_train_n, y_train)
# Predict the test set
y_pred =Random_Forest_Features_Selection.predict(X_test_n)
```

[122]:
```
test_report = get_test_report(Random_Forest_Features_Selection)
print(Random_Forest_Features_Selection)
plot_confusion_matrix(model_random_forest_undersample)
plot_roc(Random_Forest_Features_Selection)
update_score_card(model_name = 'Random_Forest_Features_Selection')
```

```
RandomForestClassifier(random_state=10)
```



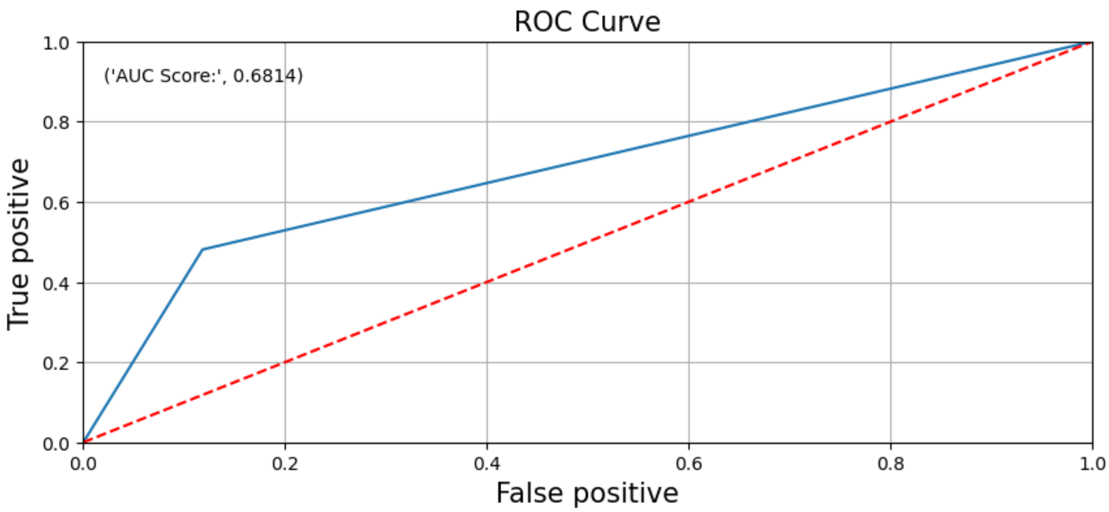
[122]:

	Model	AUC Score	Precision Score	Recall Score
0	rf_cls	0.683497	0.609700	0.781991
1	KN_Classifier_st	0.706802	0.585878	0.779621
2	KN_Classifier_tunning	0.713687	0.590994	0.782938
3	xgbm	0.643898	0.768240	0.796209
4	rus_rf	0.697535	0.579256	0.775355
5	svm_linear	0.723846	0.636735	0.800474
6	svm_poly	0.706793	0.625268	0.792417
7	Hyper_Parameter_RF	0.696580	0.647273	0.797441
8	Random_forest_undersample	0.751315	0.504488	0.743426
9	Random_Forest_Features_Selection	0.681359	0.588629	0.777541

\

	Accuracy Score	Kappa Score	f1-Score
0	0.781991	0.394907	0.534413
1	0.779621	0.421168	0.569045
2	0.782938	0.432892	0.579044
3	0.796209	0.353798	0.454315
4	0.775355	0.405404	0.555347
5	0.800474	0.465212	0.597129
6	0.792417	0.435805	0.571429
7	0.797441	0.427630	0.555382
8	0.743426	0.429896	0.608884

```
9          0.777541      0.385604   0.529323
```



[154]:

```
X = data_dummy.drop(['Churn'], axis = 1)

y =data_dummy.Churn
#X_Scale=X.apply(lambda x:(x-x.mean())/x.std())
#print(X_Scale.head())
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_Scale = scaler.transform(X)

# Retrieve column names from the original DataFrame X
column_names = X.columns

# Create a new DataFrame using the scaled data and column names
X_scaled_df = pd.DataFrame(X_Scale, columns=column_names)
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y, test_size =
 ↪0.3, random_state = 1)
print(X_train.head())
```

```
      SeniorCitizen    tenure  MonthlyCharges  TotalCharges  gender_1  \
1579            1.0  0.901408        0.350746      0.403773       0.0
1040            0.0  0.436620        0.512438      0.268763       1.0
1074            0.0  0.563380        0.957711      0.520269       0.0
2473            0.0  0.042254        0.261692      0.023304       1.0
6897            0.0  0.112676        0.369154      0.049729       1.0

      Partner_1  Dependents_1  PhoneService_1  MultipleLines_1  \
1579        0.0           0.0             0.0              0.0
```

```
1040        1.0            1.0              1.0               0.0
1074        0.0            0.0              1.0               1.0
2473        0.0            0.0              1.0               0.0
6897        0.0            1.0              1.0               0.0

      MultipleLines_2  InternetService_1  InternetService_2  OnlineSecurity_1  \
1579              1.0                1.0                0.0               0.0
1040              0.0                0.0                1.0               0.0
1074              0.0                0.0                1.0               1.0
2473              0.0                1.0                0.0               0.0
6897              0.0                1.0                0.0               0.0

      OnlineSecurity_2  OnlineBackup_1  OnlineBackup_2  DeviceProtection_1  \
1579               0.0             1.0             0.0                 1.0
1040               0.0             0.0             0.0                 0.0
1074               0.0             1.0             0.0                 1.0
2473               0.0             0.0             0.0                 0.0
6897               0.0             0.0             0.0                 0.0

      DeviceProtection_2  TechSupport_1  TechSupport_2  StreamingTV_1  \
1579                 0.0            0.0            0.0            1.0
1040                 0.0            0.0            0.0            0.0
1074                 0.0            1.0            0.0            1.0
2473                 0.0            0.0            0.0            0.0
6897                 0.0            0.0            0.0            1.0

      StreamingTV_2  StreamingMovies_1  StreamingMovies_2  Contract_1  \
1579            0.0                1.0                0.0         0.0
1040            0.0                0.0                0.0         0.0
1074            0.0                1.0                0.0         0.0
2473            0.0                0.0                0.0         0.0
6897            0.0                0.0                0.0         0.0

      Contract_2  PaperlessBilling_1  PaymentMethod_1  PaymentMethod_2  \
1579         1.0                 1.0              0.0              0.0
1040         0.0                 1.0              0.0              0.0
1074         0.0                 1.0              0.0              0.0
2473         0.0                 0.0              0.0              0.0
6897         0.0                 0.0              0.0              1.0

      PaymentMethod_3
1579              0.0
1040              0.0
1074              0.0
2473              1.0
6897              0.0
```

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

[155]:
```
#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
```

[156]:
```
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=30, activation='relu', input_shape=(30,)))

# Adding the second hidden layer
classifier.add(Dense(units=30, activation='relu'))

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

# 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)
```

```
Epoch 1/100
493/493 [==============================] - 2s 2ms/step - loss: 0.4649 -
accuracy: 0.7712
Epoch 2/100
493/493 [==============================] - 1s 2ms/step - loss: 0.4259 -
accuracy: 0.7993
Epoch 3/100
493/493 [==============================] - 1s 2ms/step - loss: 0.4214 -
accuracy: 0.8009
Epoch 4/100
493/493 [==============================] - 1s 2ms/step - loss: 0.4152 -
accuracy: 0.8068
Epoch 5/100
493/493 [==============================] - 1s 2ms/step - loss: 0.4118 -
```

```
accuracy: 0.8102
Epoch 6/100
493/493 [==============================] - 1s 2ms/step - loss: 0.4097 -
accuracy: 0.8080
Epoch 7/100
493/493 [==============================] - 1s 3ms/step - loss: 0.4064 -
accuracy: 0.8111
Epoch 8/100
493/493 [==============================] - 2s 3ms/step - loss: 0.4039 -
accuracy: 0.8113
Epoch 9/100
493/493 [==============================] - 2s 3ms/step - loss: 0.4015 -
accuracy: 0.8135
Epoch 10/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3992 -
accuracy: 0.8131
Epoch 11/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3974 -
accuracy: 0.8155
Epoch 12/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3948 -
accuracy: 0.8196
Epoch 13/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3929 -
accuracy: 0.8174
Epoch 14/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3920 -
accuracy: 0.8206
Epoch 15/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3902 -
accuracy: 0.8184
Epoch 16/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3879 -
accuracy: 0.8216
Epoch 17/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3859 -
accuracy: 0.8222
Epoch 18/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3846 -
accuracy: 0.8255
Epoch 19/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3835 -
accuracy: 0.8226
Epoch 20/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3820 -
accuracy: 0.8241
Epoch 21/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3787 -
```

```
accuracy: 0.8304
Epoch 22/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3789 -
accuracy: 0.8263
Epoch 23/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3775 -
accuracy: 0.8253
Epoch 24/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3752 -
accuracy: 0.8322
Epoch 25/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3739 -
accuracy: 0.8299
Epoch 26/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3725 -
accuracy: 0.8249
Epoch 27/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3694 -
accuracy: 0.8328
Epoch 28/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3700 -
accuracy: 0.8287
Epoch 29/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3665 -
accuracy: 0.8344
Epoch 30/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3652 -
accuracy: 0.8328
Epoch 31/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3634 -
accuracy: 0.8326
Epoch 32/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3632 -
accuracy: 0.8332
Epoch 33/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3614 -
accuracy: 0.8318
Epoch 34/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3584 -
accuracy: 0.8352
Epoch 35/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3583 -
accuracy: 0.8383
Epoch 36/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3575 -
accuracy: 0.8350
Epoch 37/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3542 -
```

```
accuracy: 0.8379
Epoch 38/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3536 -
accuracy: 0.8379
Epoch 39/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3517 -
accuracy: 0.8399
Epoch 40/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3516 -
accuracy: 0.8413
Epoch 41/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3481 -
accuracy: 0.8407
Epoch 42/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3474 -
accuracy: 0.8440
Epoch 43/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3439 -
accuracy: 0.8425
Epoch 44/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3448 -
accuracy: 0.8438
Epoch 45/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3424 -
accuracy: 0.8456
Epoch 46/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3412 -
accuracy: 0.8448
Epoch 47/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3395 -
accuracy: 0.8444
Epoch 48/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3381 -
accuracy: 0.8468
Epoch 49/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3370 -
accuracy: 0.8450
Epoch 50/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3366 -
accuracy: 0.8480
Epoch 51/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3336 -
accuracy: 0.8476
Epoch 52/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3334 -
accuracy: 0.8531
Epoch 53/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3327 -
```

```
accuracy: 0.8490
Epoch 54/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3303 -
accuracy: 0.8511
Epoch 55/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3291 -
accuracy: 0.8501
Epoch 56/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3253 -
accuracy: 0.8545
Epoch 57/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3286 -
accuracy: 0.8527
Epoch 58/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3246 -
accuracy: 0.8543
Epoch 59/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3252 -
accuracy: 0.8533
Epoch 60/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3243 -
accuracy: 0.8555
Epoch 61/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3205 -
accuracy: 0.8557
Epoch 62/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3209 -
accuracy: 0.8551
Epoch 63/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3191 -
accuracy: 0.8582
Epoch 64/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3200 -
accuracy: 0.8549
Epoch 65/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3162 -
accuracy: 0.8598
Epoch 66/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3157 -
accuracy: 0.8578
Epoch 67/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3140 -
accuracy: 0.8570
Epoch 68/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3117 -
accuracy: 0.8639
Epoch 69/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3123 -
```

```
accuracy: 0.8633
Epoch 70/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3105 -
accuracy: 0.8620
Epoch 71/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3077 -
accuracy: 0.8649
Epoch 72/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3086 -
accuracy: 0.8625
Epoch 73/100
493/493 [==============================] - 2s 4ms/step - loss: 0.3075 -
accuracy: 0.8635
Epoch 74/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3094 -
accuracy: 0.8639
Epoch 75/100
493/493 [==============================] - 2s 4ms/step - loss: 0.3022 -
accuracy: 0.8647
Epoch 76/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3044 -
accuracy: 0.8655
Epoch 77/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3048 -
accuracy: 0.8649
Epoch 78/100
493/493 [==============================] - 2s 4ms/step - loss: 0.3035 -
accuracy: 0.8661
Epoch 79/100
493/493 [==============================] - 2s 3ms/step - loss: 0.3015 -
accuracy: 0.8663
Epoch 80/100
493/493 [==============================] - 1s 3ms/step - loss: 0.3021 -
accuracy: 0.8614
Epoch 81/100
493/493 [==============================] - 1s 2ms/step - loss: 0.3001 -
accuracy: 0.8657
Epoch 82/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2991 -
accuracy: 0.8663
Epoch 83/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2990 -
accuracy: 0.8643
Epoch 84/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2979 -
accuracy: 0.8623
Epoch 85/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2982 -
```

```
accuracy: 0.8700
Epoch 86/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2957 -
accuracy: 0.8681
Epoch 87/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2957 -
accuracy: 0.8726
Epoch 88/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2988 -
accuracy: 0.8690
Epoch 89/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2936 -
accuracy: 0.8706
Epoch 90/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2911 -
accuracy: 0.8732
Epoch 91/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2914 -
accuracy: 0.8740
Epoch 92/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2927 -
accuracy: 0.8688
Epoch 93/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2899 -
accuracy: 0.8710
Epoch 94/100
493/493 [==============================] - 1s 2ms/step - loss: 0.2884 -
accuracy: 0.8736
Epoch 95/100
493/493 [==============================] - 1s 3ms/step - loss: 0.2886 -
accuracy: 0.8720
Epoch 96/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2863 -
accuracy: 0.8755
Epoch 97/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2884 -
accuracy: 0.8720
Epoch 98/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2856 -
accuracy: 0.8744
Epoch 99/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2848 -
accuracy: 0.8753
Epoch 100/100
493/493 [==============================] - 2s 3ms/step - loss: 0.2850 -
accuracy: 0.8791
```

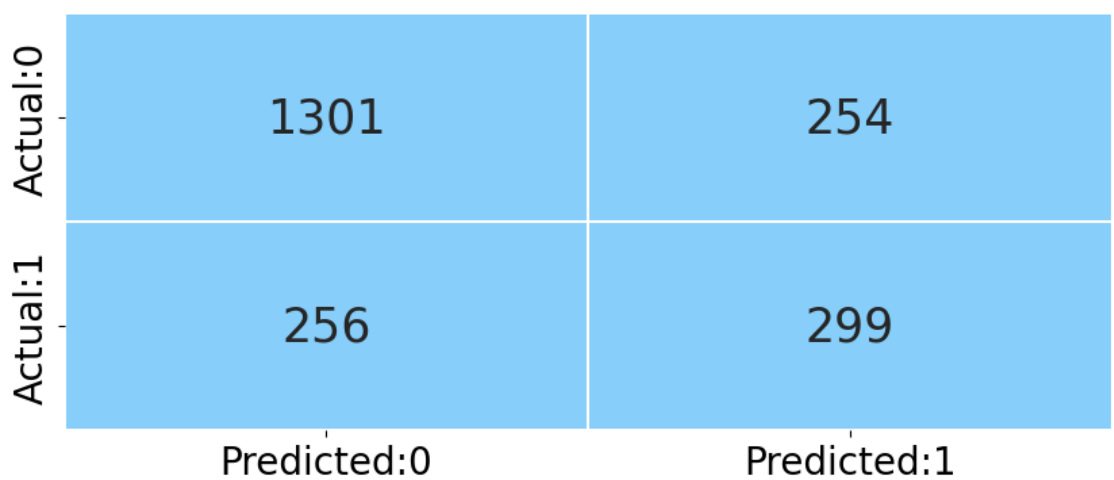
[156]:
```
<keras.src.callbacks.History at 0x1e572cbae00>
```

[157]:
```
#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)
```

```
66/66 [==============================] - 0s 2ms/step
```

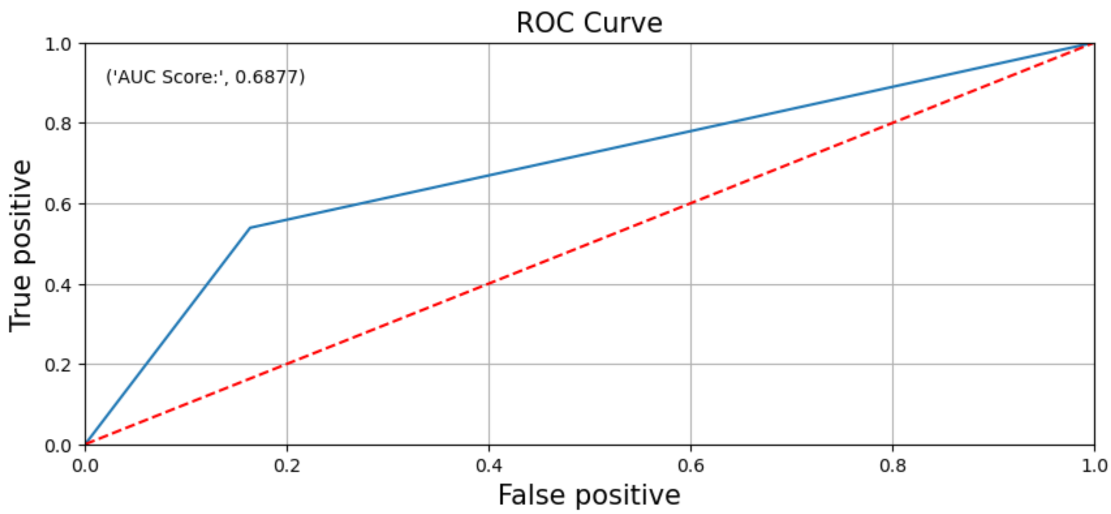
[158]:
```
test_report = get_test_report(classifier)
print(classifier)
plot_confusion_matrix(classifier)
plot_roc(classifier)
update_score_card(model_name = 'ANN_classifier')
```

```
<keras.src.engine.sequential.Sequential object at 0x000001E56F9507C0>
```



[158]:
```
                              Model  AUC Score  Precision Score  \
0                            rf_cls   0.683497         0.609700
1                  KN_Classifier_st   0.706802         0.585878
2              KN_Classifier_tunning   0.713687         0.590994
3                              xgbm   0.643898         0.768240
4                            rus_rf   0.697535         0.579256
5                        svm_linear   0.723846         0.636735
6                          svm_poly   0.706793         0.625268
7                 Hyper_Parameter_RF   0.696580         0.647273
8           Random_forest_undersample   0.751315         0.504488
9      Random_Forest_Features_Selection   0.681359         0.588629
```

```
10                  ANN_classifier   0.669155         0.557447
11                       CNN_model   0.669155         0.557447
12                       CNN_model   0.665047         0.514925
13                  ANN_classifier   0.687697         0.540687

    Recall Score  Accuracy Score  Kappa Score  f1-Score
0       0.781991        0.781991     0.394907  0.534413
1       0.779621        0.779621     0.421168  0.569045
2       0.782938        0.782938     0.432892  0.579044
3       0.796209        0.796209     0.353798  0.454315
4       0.775355        0.775355     0.405404  0.555347
5       0.800474        0.800474     0.465212  0.597129
6       0.792417        0.792417     0.435805  0.571429
7       0.797441        0.797441     0.427630  0.555382
8       0.743426        0.743426     0.429896  0.608884
9       0.777541        0.777541     0.385604  0.529323
10      0.762559        0.762559     0.355833  0.511220
11      0.762559        0.762559     0.355833  0.511220
12      0.744550        0.744550     0.333769  0.505958
13      0.758294        0.758294     0.375830  0.539711
```



[159]:
```python
from keras.models import Sequential
from keras.layers import Dense, Conv1D, Flatten, MaxPooling1D
from sklearn.model_selection import train_test_split

# Define the CNN model
CNN_model = Sequential()
```

```
# Add convolutional layer with 32 filters, kernel size of 3, and ReLU␣
 ↪activation function
CNN_model.add(Conv1D(filters=32, kernel_size=3, activation='relu',␣
 ↪input_shape=(30, 1)))

# Add a max pooling layer
CNN_model.add(MaxPooling1D(pool_size=2))

# Add another convolutional layer
CNN_model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))

# Add another max pooling layer
CNN_model.add(MaxPooling1D(pool_size=2))

# Flatten the output from convolutional layers
CNN_model.add(Flatten())

# Add a fully connected layer
CNN_model.add(Dense(units=100, activation='relu'))

# Add the output layer
CNN_model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
CNN_model.compile(optimizer='adam', loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the model
CNN_model.fit(X_train, y_train, epochs=100, batch_size=10,␣
 ↪validation_data=(X_test, y_test))
```

```
Epoch 1/100
493/493 [==============================] - 7s 12ms/step - loss: 0.4649 -
accuracy: 0.7729 - val_loss: 0.4322 - val_accuracy: 0.7896
Epoch 2/100
493/493 [==============================] - 3s 6ms/step - loss: 0.4411 -
accuracy: 0.7865 - val_loss: 0.4267 - val_accuracy: 0.7919
Epoch 3/100
493/493 [==============================] - 3s 7ms/step - loss: 0.4323 -
accuracy: 0.7911 - val_loss: 0.4315 - val_accuracy: 0.8005
Epoch 4/100
493/493 [==============================] - 4s 7ms/step - loss: 0.4281 -
accuracy: 0.8041 - val_loss: 0.4238 - val_accuracy: 0.7976
Epoch 5/100
493/493 [==============================] - 4s 7ms/step - loss: 0.4217 -
accuracy: 0.7995 - val_loss: 0.4225 - val_accuracy: 0.7986
Epoch 6/100
```

```
493/493 [==============================] - 3s 7ms/step - loss: 0.4189 -
accuracy: 0.8031 - val_loss: 0.4219 - val_accuracy: 0.7938
Epoch 7/100
493/493 [==============================] - 3s 6ms/step - loss: 0.4156 -
accuracy: 0.8033 - val_loss: 0.4200 - val_accuracy: 0.8024
Epoch 8/100
493/493 [==============================] - 3s 7ms/step - loss: 0.4139 -
accuracy: 0.8031 - val_loss: 0.4265 - val_accuracy: 0.7905
Epoch 9/100
493/493 [==============================] - 3s 7ms/step - loss: 0.4107 -
accuracy: 0.8039 - val_loss: 0.4140 - val_accuracy: 0.8057
Epoch 10/100
493/493 [==============================] - 3s 6ms/step - loss: 0.4067 -
accuracy: 0.8064 - val_loss: 0.4245 - val_accuracy: 0.7972
Epoch 11/100
493/493 [==============================] - 3s 6ms/step - loss: 0.4048 -
accuracy: 0.8092 - val_loss: 0.4210 - val_accuracy: 0.7967
Epoch 12/100
493/493 [==============================] - 3s 6ms/step - loss: 0.3994 -
accuracy: 0.8121 - val_loss: 0.4350 - val_accuracy: 0.7919
Epoch 13/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3999 -
accuracy: 0.8119 - val_loss: 0.4278 - val_accuracy: 0.7929
Epoch 14/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3954 -
accuracy: 0.8139 - val_loss: 0.4247 - val_accuracy: 0.7929
Epoch 15/100
493/493 [==============================] - 4s 8ms/step - loss: 0.3911 -
accuracy: 0.8143 - val_loss: 0.4350 - val_accuracy: 0.7915
Epoch 16/100
493/493 [==============================] - 4s 7ms/step - loss: 0.3890 -
accuracy: 0.8161 - val_loss: 0.4321 - val_accuracy: 0.7877
Epoch 17/100
493/493 [==============================] - 4s 9ms/step - loss: 0.3860 -
accuracy: 0.8180 - val_loss: 0.4615 - val_accuracy: 0.7910
Epoch 18/100
493/493 [==============================] - 4s 9ms/step - loss: 0.3809 -
accuracy: 0.8167 - val_loss: 0.4358 - val_accuracy: 0.7900
Epoch 19/100
493/493 [==============================] - 4s 9ms/step - loss: 0.3780 -
accuracy: 0.8239 - val_loss: 0.4447 - val_accuracy: 0.7720
Epoch 20/100
493/493 [==============================] - 4s 8ms/step - loss: 0.3712 -
accuracy: 0.8232 - val_loss: 0.4587 - val_accuracy: 0.7886
Epoch 21/100
493/493 [==============================] - 4s 7ms/step - loss: 0.3695 -
accuracy: 0.8243 - val_loss: 0.4705 - val_accuracy: 0.7754
Epoch 22/100
```

```
493/493 [==============================] - 3s 7ms/step - loss: 0.3665 -
accuracy: 0.8289 - val_loss: 0.4541 - val_accuracy: 0.7749
Epoch 23/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3605 -
accuracy: 0.8348 - val_loss: 0.4605 - val_accuracy: 0.7863
Epoch 24/100
493/493 [==============================] - 3s 6ms/step - loss: 0.3571 -
accuracy: 0.8362 - val_loss: 0.4690 - val_accuracy: 0.7711
Epoch 25/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3521 -
accuracy: 0.8381 - val_loss: 0.4644 - val_accuracy: 0.7749
Epoch 26/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3441 -
accuracy: 0.8405 - val_loss: 0.4860 - val_accuracy: 0.7725
Epoch 27/100
493/493 [==============================] - 4s 7ms/step - loss: 0.3438 -
accuracy: 0.8421 - val_loss: 0.4853 - val_accuracy: 0.7796
Epoch 28/100
493/493 [==============================] - 3s 6ms/step - loss: 0.3397 -
accuracy: 0.8417 - val_loss: 0.4954 - val_accuracy: 0.7739
Epoch 29/100
493/493 [==============================] - 3s 6ms/step - loss: 0.3357 -
accuracy: 0.8440 - val_loss: 0.5053 - val_accuracy: 0.7749
Epoch 30/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3291 -
accuracy: 0.8456 - val_loss: 0.5032 - val_accuracy: 0.7773
Epoch 31/100
493/493 [==============================] - 4s 7ms/step - loss: 0.3255 -
accuracy: 0.8472 - val_loss: 0.5147 - val_accuracy: 0.7777
Epoch 32/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3208 -
accuracy: 0.8525 - val_loss: 0.5284 - val_accuracy: 0.7806
Epoch 33/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3157 -
accuracy: 0.8551 - val_loss: 0.5286 - val_accuracy: 0.7692
Epoch 34/100
493/493 [==============================] - 3s 6ms/step - loss: 0.3131 -
accuracy: 0.8557 - val_loss: 0.5616 - val_accuracy: 0.7706
Epoch 35/100
493/493 [==============================] - 3s 7ms/step - loss: 0.3050 -
accuracy: 0.8602 - val_loss: 0.5405 - val_accuracy: 0.7630
Epoch 36/100
493/493 [==============================] - 4s 7ms/step - loss: 0.3012 -
accuracy: 0.8608 - val_loss: 0.5583 - val_accuracy: 0.7597
Epoch 37/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2962 -
accuracy: 0.8631 - val_loss: 0.5662 - val_accuracy: 0.7635
Epoch 38/100
```

```
493/493 [==============================] - 3s 6ms/step - loss: 0.2901 -
accuracy: 0.8653 - val_loss: 0.5732 - val_accuracy: 0.7621
Epoch 39/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2852 -
accuracy: 0.8683 - val_loss: 0.5850 - val_accuracy: 0.7545
Epoch 40/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2805 -
accuracy: 0.8714 - val_loss: 0.6046 - val_accuracy: 0.7635
Epoch 41/100
493/493 [==============================] - 4s 8ms/step - loss: 0.2774 -
accuracy: 0.8732 - val_loss: 0.6147 - val_accuracy: 0.7654
Epoch 42/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2721 -
accuracy: 0.8759 - val_loss: 0.6166 - val_accuracy: 0.7526
Epoch 43/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2693 -
accuracy: 0.8773 - val_loss: 0.6533 - val_accuracy: 0.7626
Epoch 44/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2627 -
accuracy: 0.8799 - val_loss: 0.6260 - val_accuracy: 0.7517
Epoch 45/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2591 -
accuracy: 0.8818 - val_loss: 0.6334 - val_accuracy: 0.7573
Epoch 46/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2559 -
accuracy: 0.8848 - val_loss: 0.6808 - val_accuracy: 0.7597
Epoch 47/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2477 -
accuracy: 0.8921 - val_loss: 0.6796 - val_accuracy: 0.7536
Epoch 48/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2472 -
accuracy: 0.8939 - val_loss: 0.6695 - val_accuracy: 0.7578
Epoch 49/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2420 -
accuracy: 0.8978 - val_loss: 0.7108 - val_accuracy: 0.7521
Epoch 50/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2406 -
accuracy: 0.8937 - val_loss: 0.7198 - val_accuracy: 0.7602
Epoch 51/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2316 -
accuracy: 0.9004 - val_loss: 0.7297 - val_accuracy: 0.7678
Epoch 52/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2316 -
accuracy: 0.8988 - val_loss: 0.7225 - val_accuracy: 0.7436
Epoch 53/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2272 -
accuracy: 0.8992 - val_loss: 0.7414 - val_accuracy: 0.7592
Epoch 54/100
```

```
493/493 [==============================] - 3s 7ms/step - loss: 0.2212 -
accuracy: 0.9037 - val_loss: 0.7322 - val_accuracy: 0.7517
Epoch 55/100
493/493 [==============================] - 4s 7ms/step - loss: 0.2203 -
accuracy: 0.8996 - val_loss: 0.7837 - val_accuracy: 0.7474
Epoch 56/100
493/493 [==============================] - 4s 8ms/step - loss: 0.2156 -
accuracy: 0.9120 - val_loss: 0.7994 - val_accuracy: 0.7564
Epoch 57/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2131 -
accuracy: 0.9096 - val_loss: 0.7809 - val_accuracy: 0.7384
Epoch 58/100
493/493 [==============================] - 3s 6ms/step - loss: 0.2097 -
accuracy: 0.9120 - val_loss: 0.8130 - val_accuracy: 0.7493
Epoch 59/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2040 -
accuracy: 0.9130 - val_loss: 0.8171 - val_accuracy: 0.7502
Epoch 60/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2095 -
accuracy: 0.9122 - val_loss: 0.8555 - val_accuracy: 0.7536
Epoch 61/100
493/493 [==============================] - 3s 7ms/step - loss: 0.2020 -
accuracy: 0.9137 - val_loss: 0.8704 - val_accuracy: 0.7555
Epoch 62/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1963 -
accuracy: 0.9159 - val_loss: 0.8919 - val_accuracy: 0.7507
Epoch 63/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1940 -
accuracy: 0.9173 - val_loss: 0.8595 - val_accuracy: 0.7521
Epoch 64/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1923 -
accuracy: 0.9171 - val_loss: 0.8768 - val_accuracy: 0.7498
Epoch 65/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1887 -
accuracy: 0.9206 - val_loss: 0.8960 - val_accuracy: 0.7483
Epoch 66/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1876 -
accuracy: 0.9222 - val_loss: 0.9390 - val_accuracy: 0.7555
Epoch 67/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1815 -
accuracy: 0.9210 - val_loss: 0.9151 - val_accuracy: 0.7479
Epoch 68/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1816 -
accuracy: 0.9252 - val_loss: 0.9093 - val_accuracy: 0.7517
Epoch 69/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1805 -
accuracy: 0.9234 - val_loss: 0.8947 - val_accuracy: 0.7483
Epoch 70/100
```

```
493/493 [==============================] - 3s 7ms/step - loss: 0.1800 -
accuracy: 0.9236 - val_loss: 0.9710 - val_accuracy: 0.7493
Epoch 71/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1759 -
accuracy: 0.9258 - val_loss: 0.9883 - val_accuracy: 0.7474
Epoch 72/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1727 -
accuracy: 0.9283 - val_loss: 1.0195 - val_accuracy: 0.7536
Epoch 73/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1716 -
accuracy: 0.9267 - val_loss: 0.9737 - val_accuracy: 0.7559
Epoch 74/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1658 -
accuracy: 0.9275 - val_loss: 0.9934 - val_accuracy: 0.7531
Epoch 75/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1680 -
accuracy: 0.9319 - val_loss: 1.0825 - val_accuracy: 0.7536
Epoch 76/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1704 -
accuracy: 0.9271 - val_loss: 1.0421 - val_accuracy: 0.7450
Epoch 77/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1687 -
accuracy: 0.9269 - val_loss: 1.0011 - val_accuracy: 0.7469
Epoch 78/100
493/493 [==============================] - 4s 8ms/step - loss: 0.1588 -
accuracy: 0.9336 - val_loss: 1.0333 - val_accuracy: 0.7502
Epoch 79/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1644 -
accuracy: 0.9291 - val_loss: 1.0159 - val_accuracy: 0.7483
Epoch 80/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1576 -
accuracy: 0.9309 - val_loss: 1.0924 - val_accuracy: 0.7540
Epoch 81/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1605 -
accuracy: 0.9311 - val_loss: 1.0393 - val_accuracy: 0.7360
Epoch 82/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1577 -
accuracy: 0.9344 - val_loss: 1.0903 - val_accuracy: 0.7512
Epoch 83/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1545 -
accuracy: 0.9319 - val_loss: 1.0845 - val_accuracy: 0.7431
Epoch 84/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1588 -
accuracy: 0.9340 - val_loss: 1.0725 - val_accuracy: 0.7536
Epoch 85/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1516 -
accuracy: 0.9328 - val_loss: 1.1358 - val_accuracy: 0.7526
Epoch 86/100
```

```
493/493 [==============================] - 3s 6ms/step - loss: 0.1585 -
accuracy: 0.9307 - val_loss: 1.0962 - val_accuracy: 0.7536
Epoch 87/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1537 -
accuracy: 0.9330 - val_loss: 1.1109 - val_accuracy: 0.7531
Epoch 88/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1504 -
accuracy: 0.9354 - val_loss: 1.1081 - val_accuracy: 0.7436
Epoch 89/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1504 -
accuracy: 0.9348 - val_loss: 1.1655 - val_accuracy: 0.7498
Epoch 90/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1476 -
accuracy: 0.9368 - val_loss: 1.1312 - val_accuracy: 0.7403
Epoch 91/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1476 -
accuracy: 0.9372 - val_loss: 1.1749 - val_accuracy: 0.7455
Epoch 92/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1434 -
accuracy: 0.9382 - val_loss: 1.1998 - val_accuracy: 0.7493
Epoch 93/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1476 -
accuracy: 0.9376 - val_loss: 1.1824 - val_accuracy: 0.7517
Epoch 94/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1452 -
accuracy: 0.9388 - val_loss: 1.2091 - val_accuracy: 0.7630
Epoch 95/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1421 -
accuracy: 0.9372 - val_loss: 1.1871 - val_accuracy: 0.7550
Epoch 96/100
493/493 [==============================] - 3s 6ms/step - loss: 0.1434 -
accuracy: 0.9413 - val_loss: 1.1930 - val_accuracy: 0.7521
Epoch 97/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1434 -
accuracy: 0.9382 - val_loss: 1.2252 - val_accuracy: 0.7445
Epoch 98/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1410 -
accuracy: 0.9437 - val_loss: 1.2655 - val_accuracy: 0.7474
Epoch 99/100
493/493 [==============================] - 4s 7ms/step - loss: 0.1394 -
accuracy: 0.9401 - val_loss: 1.2700 - val_accuracy: 0.7464
Epoch 100/100
493/493 [==============================] - 3s 7ms/step - loss: 0.1443 -
accuracy: 0.9393 - val_loss: 1.1977 - val_accuracy: 0.7427
```

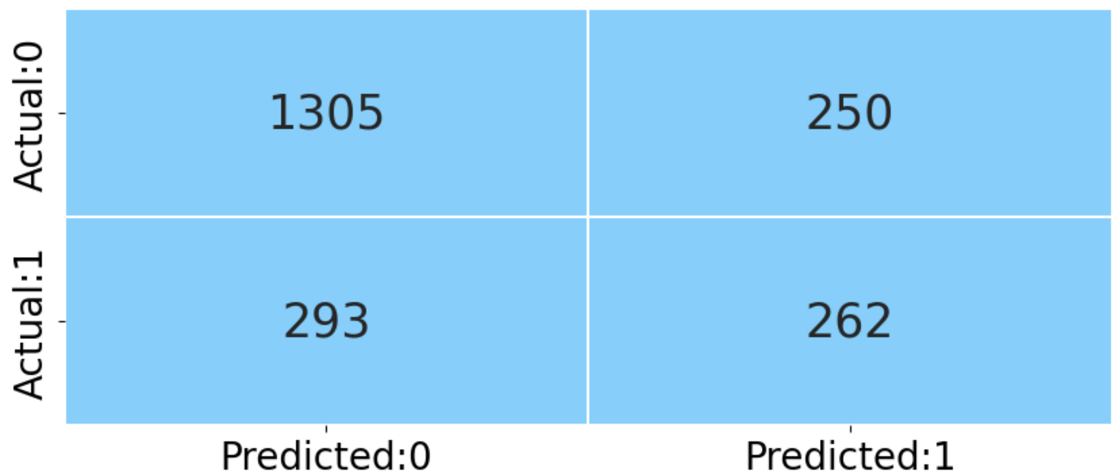
[159]: <keras.src.callbacks.History at 0x1e56fba9300>

[160]:
```
#Predict the Test Set Results
y_pred = CNN_model.predict(X_test)
y_pred = (y_pred > 0.5)
```

```
66/66 [==============================] - 0s 4ms/step
```

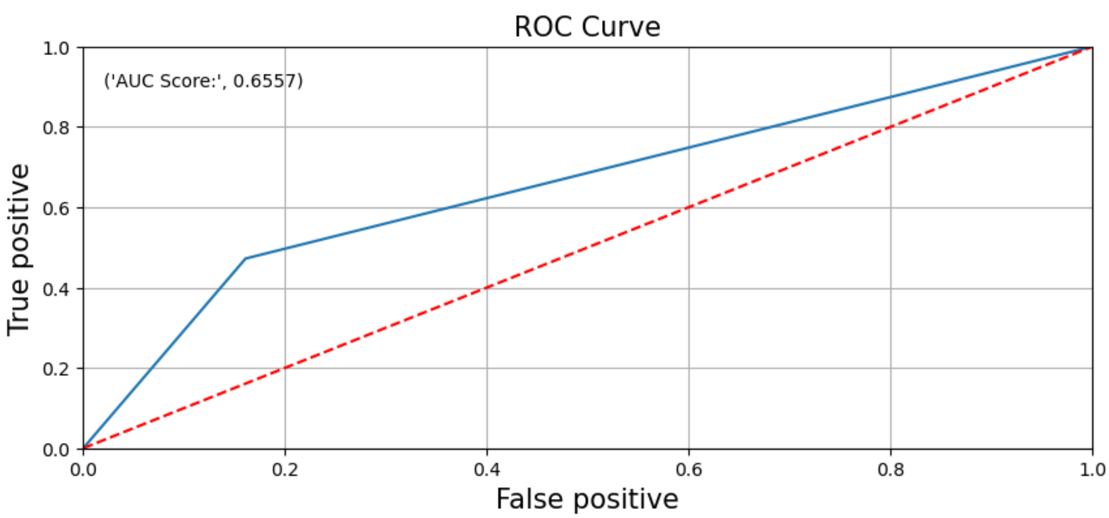
[161]:
```
test_report = get_test_report(CNN_model)
print(CNN_model)
plot_confusion_matrix(CNN_model)
plot_roc(CNN_model)
update_score_card(model_name = 'CNN_model')
```

```
<keras.src.engine.sequential.Sequential object at 0x000001E572C1DCF0>
```



[161]:
```
                               Model  AUC Score  Precision Score  \
0                             rf_cls   0.683497         0.609700
1                   KN_Classifier_st   0.706802         0.585878
2               KN_Classifier_tunning  0.713687         0.590994
3                                xgbm   0.643898         0.768240
4                              rus_rf   0.697535         0.579256
5                          svm_linear   0.723846         0.636735
6                            svm_poly   0.706793         0.625268
7                  Hyper_Parameter_RF   0.696580         0.647273
8            Random_forest_undersample  0.751315         0.504488
9   Random_Forest_Features_Selection   0.681359         0.588629
10                     ANN_classifier   0.669155         0.557447
11                          CNN_model   0.669155         0.557447
12                          CNN_model   0.665047         0.514925
13                     ANN_classifier   0.687697         0.540687
14                          CNN_model   0.655650         0.511719
```

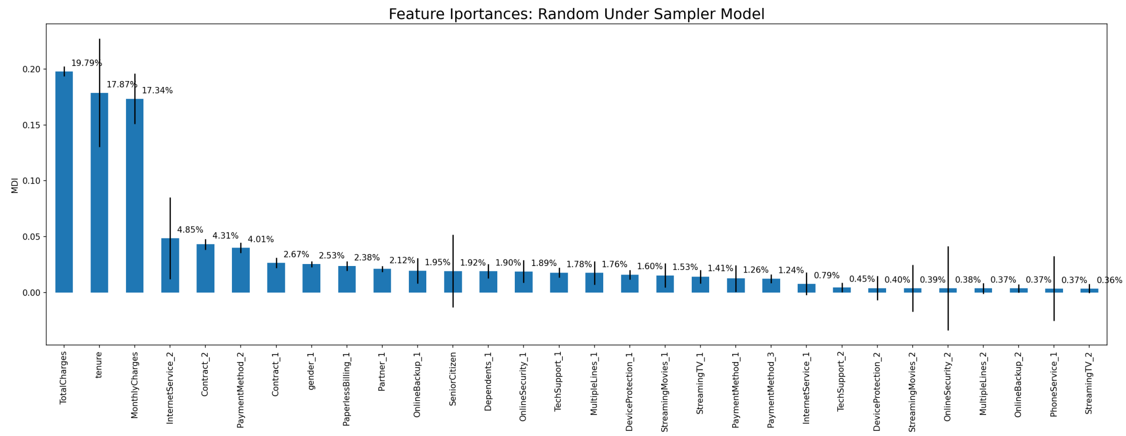
```
    Recall Score  Accuracy Score  Kappa Score  f1-Score
0       0.781991        0.781991     0.394907  0.534413
1       0.779621        0.779621     0.421168  0.569045
2       0.782938        0.782938     0.432892  0.579044
3       0.796209        0.796209     0.353798  0.454315
4       0.775355        0.775355     0.405404  0.555347
5       0.800474        0.800474     0.465212  0.597129
6       0.792417        0.792417     0.435805  0.571429
7       0.797441        0.797441     0.427630  0.555382
8       0.743426        0.743426     0.429896  0.608884
9       0.777541        0.777541     0.385604  0.529323
10      0.762559        0.762559     0.355833  0.511220
11      0.762559        0.762559     0.355833  0.511220
12      0.744550        0.744550     0.333769  0.505958
13      0.758294        0.758294     0.375830  0.539711
14      0.742654        0.742654     0.319254  0.491097
```



8 Features Importance

[163]:
```
std = np.std([tree.feature_importances_ for tree in rus_rf.estimators_], axis=0)
importances = pd.Series(rus_rf.feature_importances_, index=X.columns).
 ↪sort_values(ascending=False)

fig, ax = plt.subplots()
importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature Iportances: Random Under Sampler Model", size=18)
ax.set_ylabel("MDI")
for i in range(len(X.columns)):
```

```
  ax.text(i+0.2, importances[i]+0.005, '{:.2%}'.format(importances[i]), size=10)
fig.set_size_inches(23, 7)
fig.set_dpi(150)
```



9 CONCLUSION:

This project predicts whether customers will churn from a telecom service using various machine learning (ML) and deep learning (DL) models. We applied 12 different types of models, including Random Forest, KNN, KNN with hyperparameter tuning, XGBoost, SVM with linear and polynomial kernels, Random Forest with hyperparameter tuning, Random Forest with feature selection, Random Forest with undersampling, ANN, and CNN models. The SVM model with a linear kernel achieved the highest accuracy at 80% compared to other models. Except for the precision score, all other accuracy metrics were better for the SVM with a linear kernel compared to the other 11 models. Feature importance analysis using the Random Forest with undersampling technique revealed that Total Charges, Tenure, Monthly Charges, and Internet Service are the most significant factors contributing to customer churn. These findings suggest that telecom companies should focus on these factors to retain their customers effectively.