# CUSTOMER CHURN ANALYSIS AND CLASSIFICATION

## AIM

With the rapid expansion of the telecom business, specialist co-ops are increasingly focusing on growing its endorser base. It has become a challenge to maintain existing clients in the difficult environment. It is stated that the cost of acquiring a new client is undoubtedly more than the cost of keeping the current one. As a result, telecom companies should use advanced analysis to figure out customer behaviour and, as a result, predict client relationships if they quit the service.

Some probable questions are as follows:

a) What factors contribute to customer churn?

b) Which customers are more likely to churn?

c) What can be done to prevent them from leaving?

## Overview

1. Import data and python packages
   - Import packages
   - Import data
   - Data shape and info
2. Data visualization
   - Count Plots
   - Pie Plots
   - Box Plots
   - Heatmap(Correlation)
   - Pairplot
3. Classification

   3.1 Split data as train and test

   3.2 Functions for models

   3.3 Models

   - Decision Tree Classifier
   - Gradient Booster Classifier
   - KNN Classifier
   - Random Forest Classifier
   - Artificial Nural Network
4. Result

- Cross validation scores

# 1. Import data and python packages

```
#installation of packages and libraries required
!pip install mglearn
```

```
Requirement already satisfied: mglearn in c:\users\user\anaconda3\lib\
site-packages (0.1.9)
Requirement already satisfied: pandas in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (1.4.2)
Requirement already satisfied: pillow in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (9.0.1)
Requirement already satisfied: cycler in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (0.11.0)
Requirement already satisfied: matplotlib in c:\users\user\anaconda3\
lib\site-packages (from mglearn) (3.5.1)
Requirement already satisfied: joblib in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (1.1.0)
Requirement already satisfied: imageio in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (2.9.0)
Requirement already satisfied: scikit-learn in c:\users\user\
anaconda3\lib\site-packages (from mglearn) (1.0.2)
Requirement already satisfied: numpy in c:\users\user\anaconda3\lib\
site-packages (from mglearn) (1.21.5)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\user\
anaconda3\lib\site-packages (from matplotlib->mglearn) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\user\
anaconda3\lib\site-packages (from matplotlib->mglearn) (1.3.2)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\user\
anaconda3\lib\site-packages (from matplotlib->mglearn) (2.8.2)
Requirement already satisfied: packaging>=20.0 in c:\users\user\
anaconda3\lib\site-packages (from matplotlib->mglearn) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\user\
anaconda3\lib\site-packages (from matplotlib->mglearn) (3.0.4)
Requirement already satisfied: six>=1.5 in c:\users\user\anaconda3\
lib\site-packages (from python-dateutil>=2.7->matplotlib->mglearn)
(1.16.0)
Requirement already satisfied: pytz>=2020.1 in c:\users\user\
anaconda3\lib\site-packages (from pandas->mglearn) (2021.3)
Requirement already satisfied: scipy>=1.1.0 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn->mglearn) (1.7.3)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn->mglearn) (2.2.0)
```

```
#module and libraries

import numpy as np
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
import mglearn

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

## Machine Learning Models Diffrent Algorithms
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix,accuracy_score, f1_score,
precision_score, recall_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense , Activation
from tensorflow.keras.utils import to_categorical

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

churn_df = pd.read_csv('telecom_churn.csv')

# Show Data
churn_df.head()
```

```
   Churn  AccountWeeks  ContractRenewal  DataPlan  DataUsage
CustServCalls  \
0      0           128                1         1        2.7
1
1      0           107                1         1        3.7
1
2      0           137                1         0        0.0
0
3      0            84                0         0        0.0
2
4      0            75                0         0        0.0
3

   DayMins  DayCalls  MonthlyCharge  OverageFee  RoamMins
0    265.1       110           89.0        9.87      10.0
1    161.6       123           82.0        9.78      13.7
2    243.4       114           52.0        6.06      12.2
```

```
3    299.4        71            57.0        3.10       6.6
4    166.7       113            41.0        7.42      10.1
```

```
# Data shape
churn_df.shape
```

```
(3333, 11)
```

```
#Statistics based information of data
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Churn            3333 non-null   int64  
 1   AccountWeeks     3333 non-null   int64  
 2   ContractRenewal  3333 non-null   int64  
 3   DataPlan         3333 non-null   int64  
 4   DataUsage        3333 non-null   float64
 5   CustServCalls    3333 non-null   int64  
 6   DayMins          3333 non-null   float64
 7   DayCalls         3333 non-null   int64  
 8   MonthlyCharge    3333 non-null   float64
 9   OverageFee       3333 non-null   float64
 10  RoamMins         3333 non-null   float64
dtypes: float64(5), int64(6)
memory usage: 286.6 KB
```

There are no null or missing data.

The attributes are of numeric type(int or float).

```
#Data Description
churn_df_describe = churn_df.describe().T
churn_df_describe
```

```
                  count        mean        std  min     25%     50%  
75%  \
Churn            3333.0    0.144914   0.352067  0.0    0.00    0.00   
0.00   
AccountWeeks     3333.0  101.064806  39.822106  1.0   74.00  101.00  
127.00   
ContractRenewal  3333.0    0.903090   0.295879  0.0    1.00    1.00   
1.00   
DataPlan         3333.0    0.276628   0.447398  0.0    0.00    0.00   
1.00   
DataUsage        3333.0    0.816475   1.272668  0.0    0.00    0.00   
1.78   
CustServCalls    3333.0    1.562856   1.315491  0.0    1.00    1.00   
2.00   
```

```
DayMins           3333.0  179.775098  54.467389   0.0  143.70  179.40
216.40
DayCalls          3333.0  100.435644  20.069084   0.0   87.00  101.00
114.00
MonthlyCharge     3333.0   56.305161  16.426032  14.0   45.00   53.50
66.20
OverageFee        3333.0   10.051488   2.535712   0.0    8.33   10.07
11.77
RoamMins          3333.0   10.237294   2.791840   0.0    8.50   10.30
12.10

                    max
Churn              1.00
AccountWeeks     243.00
ContractRenewal    1.00
DataPlan           1.00
DataUsage          5.40
CustServCalls      9.00
DayMins          350.80
DayCalls         165.00
MonthlyCharge    111.30
OverageFee        18.19
RoamMins          20.00
```

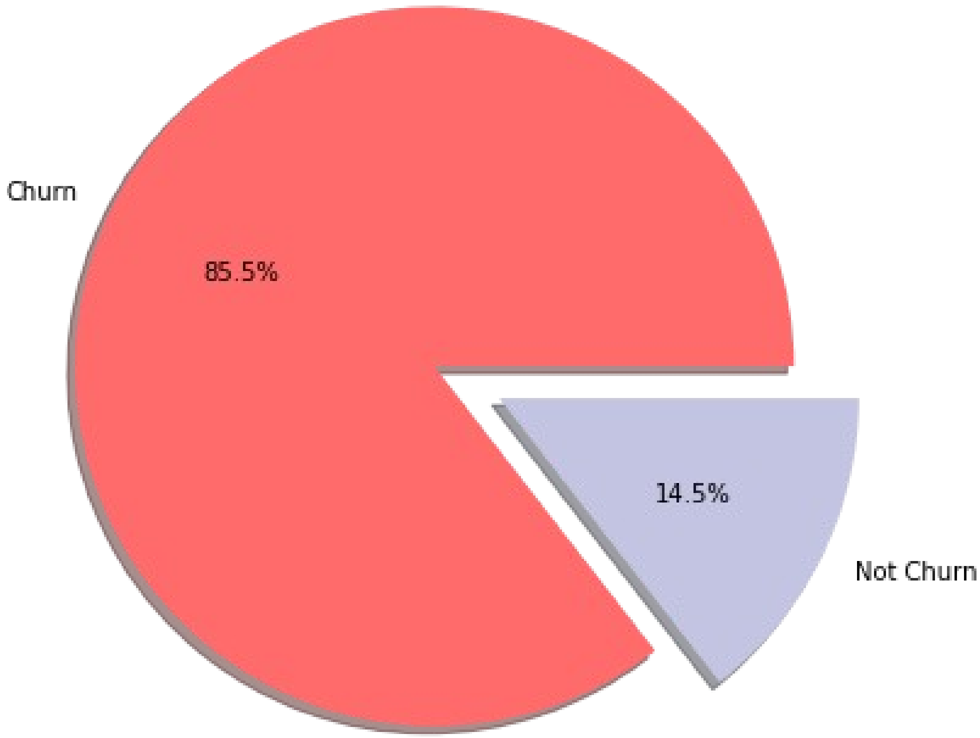
Descriptive statistics for attributes of dataframe.

- The count of each attribute represents no missing data anywhere.
- The min and max values can be observed along with mean to analyze whether the outliers have impacted the mean or not.
- Along with that, the percentiles are also given analyzing the same.

## 2. Data visualization

### *2.1 Analysis of target label*

```
#churn feature as target
labels = ['Churn', 'Not Churn']
sizes = churn_df['Churn'].value_counts(sort = True)


explode = (0.2, 0)

fig1, ax1 = plt.subplots(figsize=(8, 6))
ax1.pie(sizes, explode=explode, colors = ["#FF6A6A","#C2C4E2"],
labels=labels, autopct='%1.1f%%', shadow=True)
ax1.axis('equal')

plt.show()
sizes
```




```
0    2850
1     483
Name: Churn, dtype: int64
```

Target labels interpretation:

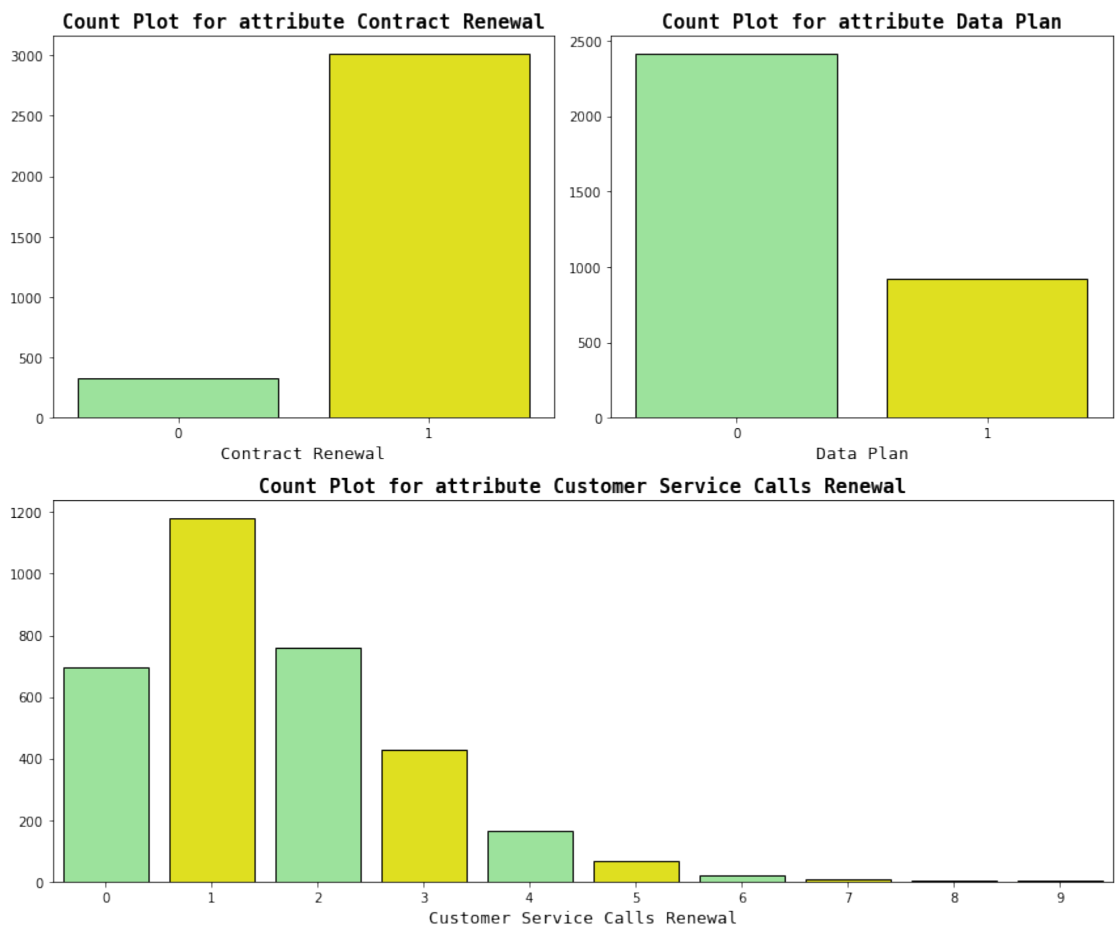
- 0 class: The customer will not leave the company service
- 1 class: The customer will leave the company service

More instances i.e almost 85% are of Churn category that means customer will leave the company on the basis of the attributes mentioned.

### *2.2 Analysis of attributes; Counts*

```
ax = plt.figure(figsize=(12,10))

#Contract Renewal
plt.subplot(2,2,1)
sns.countplot(data = churn_df , x =
"ContractRenewal" ,palette=["lightgreen","yellow"], edgecolor='k')
plt.title("Count Plot for attribute Contract Renewal" , size=15,
fontweight='bold', fontfamily='monospace')
plt.xlabel("Contract Renewal", size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel('')
```

```
#DataPlan
plt.subplot(2,2,2)
sns.countplot(data = churn_df , x = "DataPlan" ,
palette=["lightgreen","yellow"], edgecolor='k')
plt.title("Count Plot for attribute Data Plan" , size=15,
fontweight='bold', fontfamily='monospace')
plt.xlabel("Data Plan", size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel('')

#Customer Service Calls
plt.subplot(2,2,(3,4))
sns.countplot(data = churn_df , x = "CustServCalls" ,
palette=["lightgreen","yellow"], edgecolor='k')
plt.title("Count Plot for attribute Customer Service Calls Renewal" ,
size=15, fontweight='bold', fontfamily='monospace')
plt.xlabel("Customer Service Calls Renewal", size=13,
fontweight='light', fontfamily='monospace')
plt.ylabel('')
plt.tight_layout()
plt.show()
```

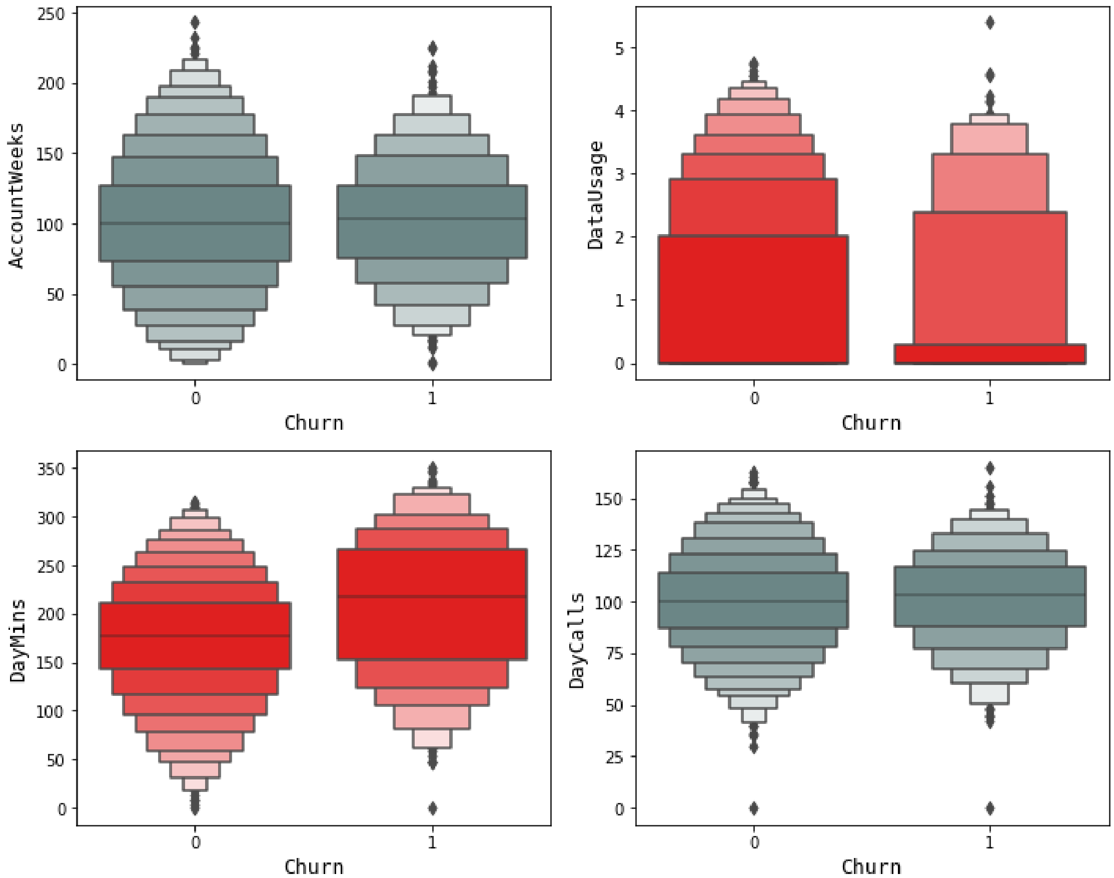


Interpretation of Analysis:

- **Contract Renewal**: Maximum cases are of churn where the factor is contract renewal. The services might have not been satisfiable.
- **Data Plan**: On the basis of data plan service provided by the telecom company, the customers are more satisfied resulting in non-churn cases. Less than a half of customers didn't like the service.

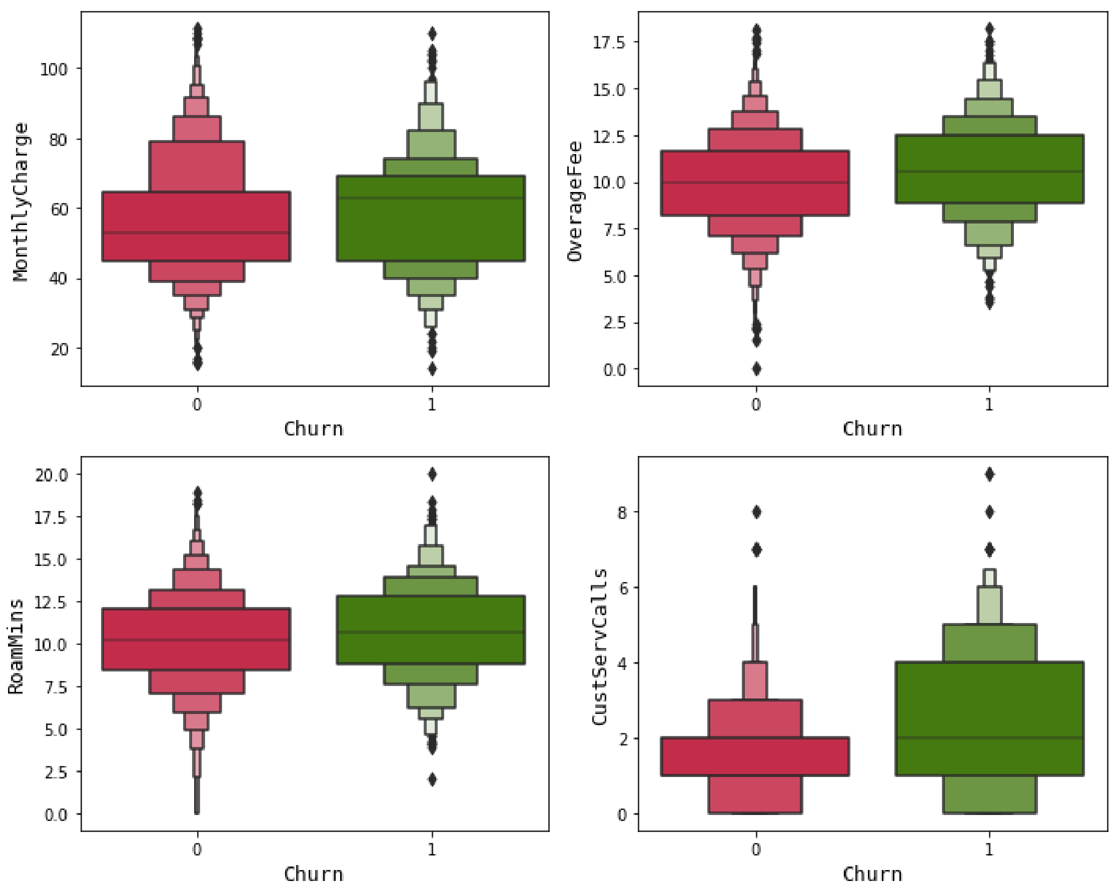
### 2.3 Outliers detection

```
#AccountWeeks
ax = plt.figure(figsize=(10,8))
plt.subplot(2,2,1)
sns.boxenplot(data = churn_df , y = "AccountWeeks" , x = "Churn" ,
color="#668B8B", scale="linear")
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("AccountWeeks",size=13, fontweight='light',
fontfamily='monospace')

#DataUsage
plt.subplot(2,2,2)
sns.boxenplot(data = churn_df , y = "DataUsage" ,  x = "Churn" ,
color="#FF0000", scale="linear")
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("DataUsage",size=13, fontweight='light',
fontfamily='monospace')

#DayMins
plt.subplot(2,2,3)
sns.boxenplot(data = churn_df , y = "DayMins" , x = "Churn" ,
color="#FF0000", scale="linear")
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("DayMins",size=13, fontweight='light',
fontfamily='monospace')

#DayCalls
plt.subplot(2,2,4)
sns.boxenplot(data = churn_df , y = "DayCalls" , x = "Churn" ,
color="#668B8B", scale="linear")
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("DayCalls",size=13, fontweight='light',
fontfamily='monospace')
plt.tight_layout()
plt.show()
```




```
ax = plt.figure(figsize=(10,8))

#MonthlyCharge
plt.subplot(2,2,1)
sns.boxenplot(data = churn_df , y = "MonthlyCharge" , x = "Churn" ,
palette=["#DC143C","#458B00"])
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("MonthlyCharge",size=13, fontweight='light',
fontfamily='monospace')

#OverageFee
plt.subplot(2,2,2)
sns.boxenplot(data = churn_df , y = "OverageFee" , x =
"Churn" ,palette=["#DC143C","#458B00"])
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("OverageFee",size=13, fontweight='light',
fontfamily='monospace')

#RoamMins
plt.subplot(2,2,3)
sns.boxenplot(data = churn_df , y = "RoamMins" , x = "Churn" ,
```

```
palette=["#DC143C","#458B00"])
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("RoamMins",size=13, fontweight='light',
fontfamily='monospace')

#CustServCalls
plt.subplot(2,2,4)
sns.boxenplot(data = churn_df , y = "CustServCalls" , x = "Churn" ,
palette=["#DC143C","#458B00"])
plt.xlabel("Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.ylabel("CustServCalls",size=13, fontweight='light',
fontfamily='monospace')
plt.tight_layout()
plt.show()
```

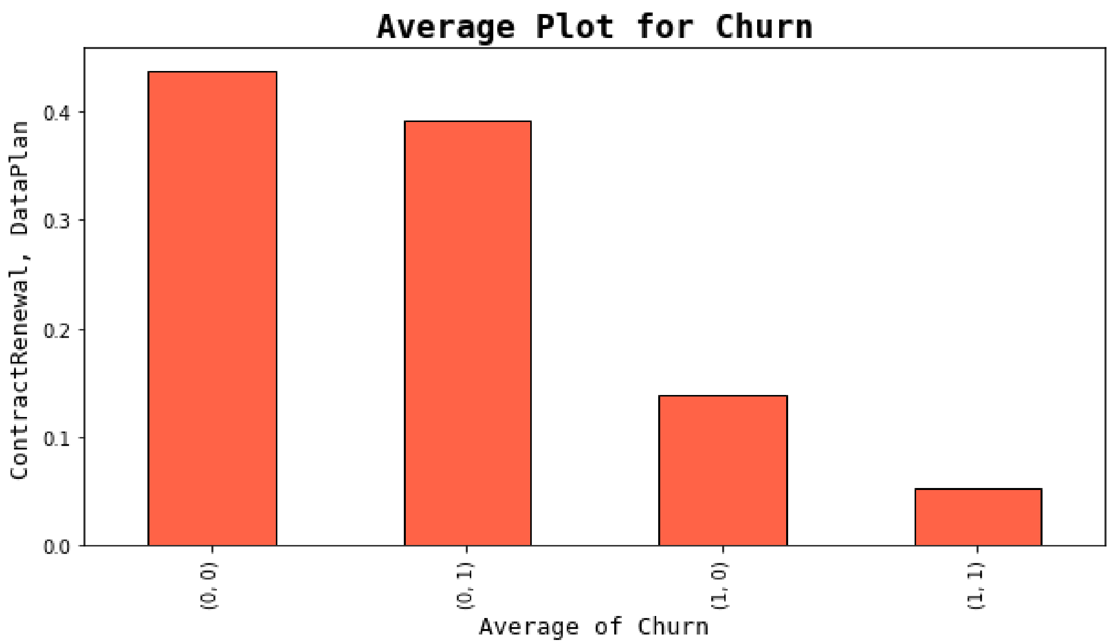



Analysis Interpretation:

- The boxen plots of these four attributes show some outliers that are nearer to the min and max quartiles therefore, can be left untreated.

### 2.4 Paired Analysis

```
ax = plt.figure(figsize=(10,5))
churn_df.groupby(['ContractRenewal',"DataPlan"])
['Churn'].mean().plot(figsize=(10,5),kind="bar",color="#FF6347",

edgecolor='k')
plt.title("Average Plot for Churn" , size=18, fontweight='bold',
fontfamily='monospace')
plt.ylabel("ContractRenewal, DataPlan",size=13, fontweight='light',
fontfamily='monospace')
plt.xlabel("Average of Churn",size=13, fontweight='light',
fontfamily='monospace')
plt.show()
```




**Interpretation of Analysis:**

Customer churn analysis requires the attributes ContractRenewal and DataPlan. The probability of customer churn is low if these two attributes are "1". DataPlan has a greater impact than ContractRenewal.

By increasing DataUsage, customers are less likely to churn, and by decreasing other attributes, customers are also less likely to churn. This work has been done by classifying and averaging Churn.

### 2.5 Basic Statistics about data

```
#mean corresponding to each attribute with churn(0 and 1)
print("Churn:0 and Churn:1")
mean_df = churn_df.mean().reset_index()
mean_df.columns = ['Feature', 'Mean']
mean_df.set_index('Feature')
```

```
Churn:0 and Churn:1

                       Mean
Feature
Churn              0.144914
AccountWeeks     101.064806
ContractRenewal    0.903090
DataPlan           0.276628
DataUsage          0.816475
CustServCalls      1.562856
DayMins          179.775098
DayCalls         100.435644
MonthlyCharge     56.305161
OverageFee        10.051488
RoamMins          10.237294
```

```
#mean corresponding to each attriibute with churn(0)
print("Churn:0")
mean_df = churn_df.loc[churn_df["Churn"]==0].mean().reset_index()
mean_df.columns = ['Feature', 'Mean']
mean_df.set_index('Feature')
```

```
Churn:0

                       Mean
Feature
Churn              0.000000
AccountWeeks     100.793684
ContractRenewal    0.934737
DataPlan           0.295439
DataUsage          0.862151
CustServCalls      1.449825
DayMins          175.175754
DayCalls         100.283158
MonthlyCharge     55.816246
OverageFee         9.954618
RoamMins          10.158877
```

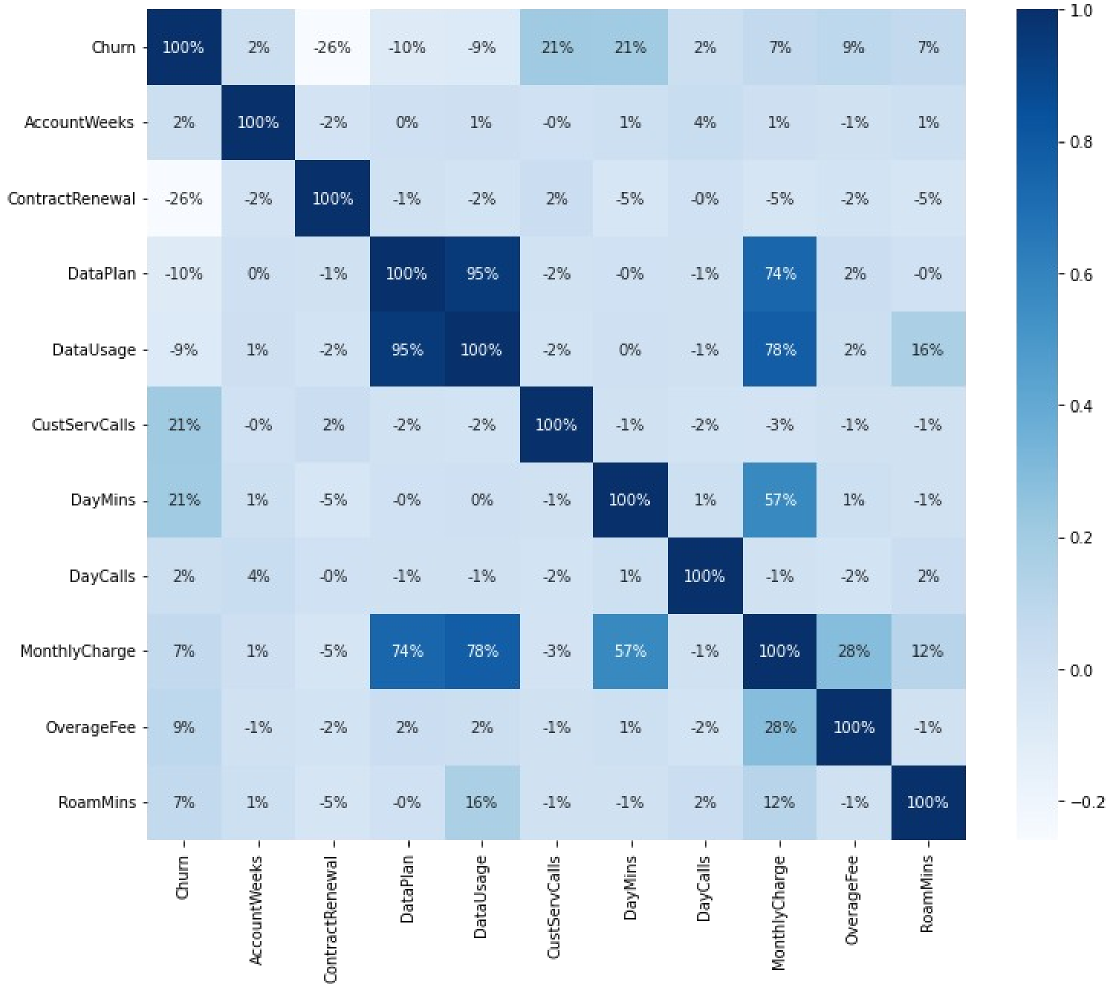
```
#mean corresponding to each attribute with churn(1)
print("Churn:1")
mean_df = churn_df.loc[churn_df["Churn"]==1].mean().reset_index()
mean_df.columns = ['Feature', 'Mean']
mean_df.set_index('Feature')
```

```
Churn:1

                       Mean
Feature
Churn              1.000000
AccountWeeks     102.664596
ContractRenewal    0.716356
DataPlan           0.165631
```

```
DataUsage          0.546957
CustServCalls      2.229814
DayMins          206.914079
DayCalls         101.335404
MonthlyCharge     59.190062
OverageFee        10.623085
RoamMins          10.700000
```

### 2.6 Attributes correlation

```
#correlation matrix
ax = plt.figure(figsize=(12,10))
sns.heatmap(churn_df.corr(),annot=True,cmap="Blues", fmt='.0%')
plt.show()
```

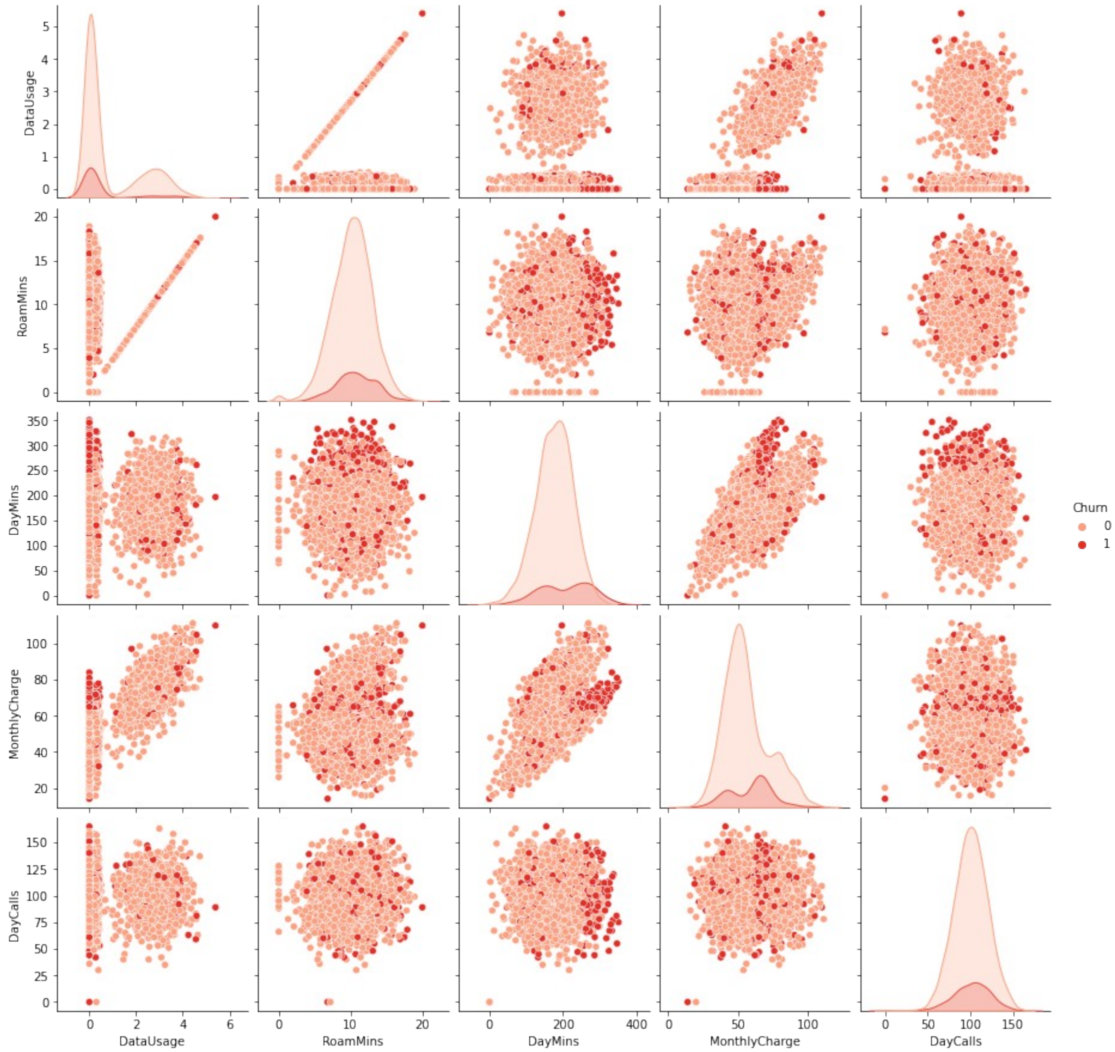


**Interpretation of Analysis:**

Some features can be seen highly correlated with each other where the correlation present is greater than 50%. The dimensionality reduction can be performed to overcome this.

```
sns.pairplot(data =
churn_df[["DataUsage","RoamMins","DayMins","MonthlyCharge","Churn","Da
```

```
yCalls"]],
              hue="Churn", palette='Reds')
plt.show()
```

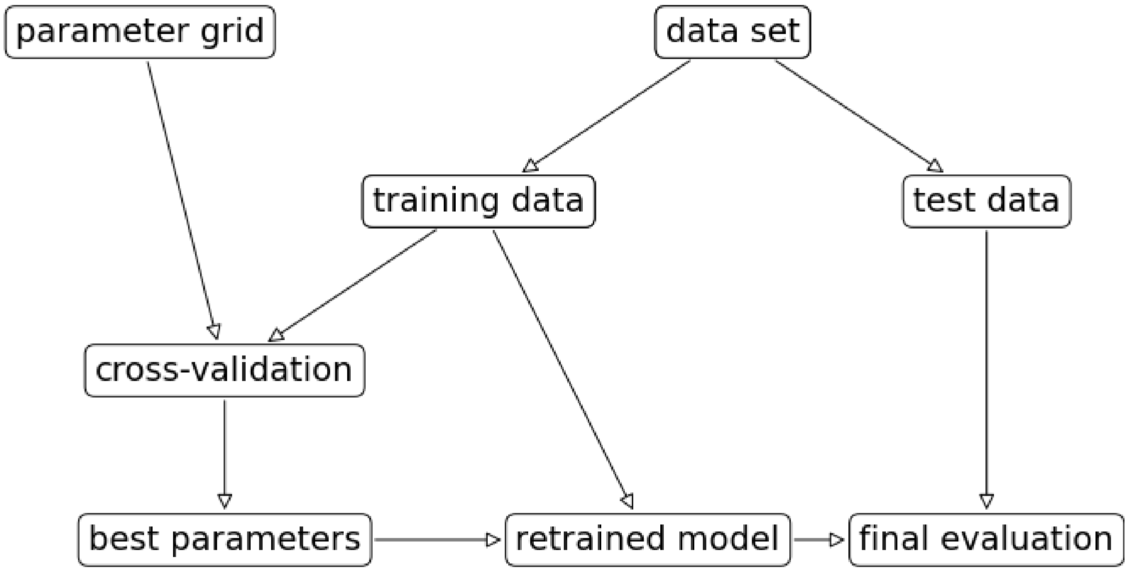



**Interpretation of Analysis:**

The leptokurtic curve can be seen between identical features; observing the peak at curve where the class label is 0 i.e, the churn 0(not churn) category. The dataset/column values are spread evenly without much skewness. A dense cluster can be seen in each attribute.

## 3. Classification

```
#Classification process overview
mglearn.plots.plot_grid_search_overview();
plt.show()
```

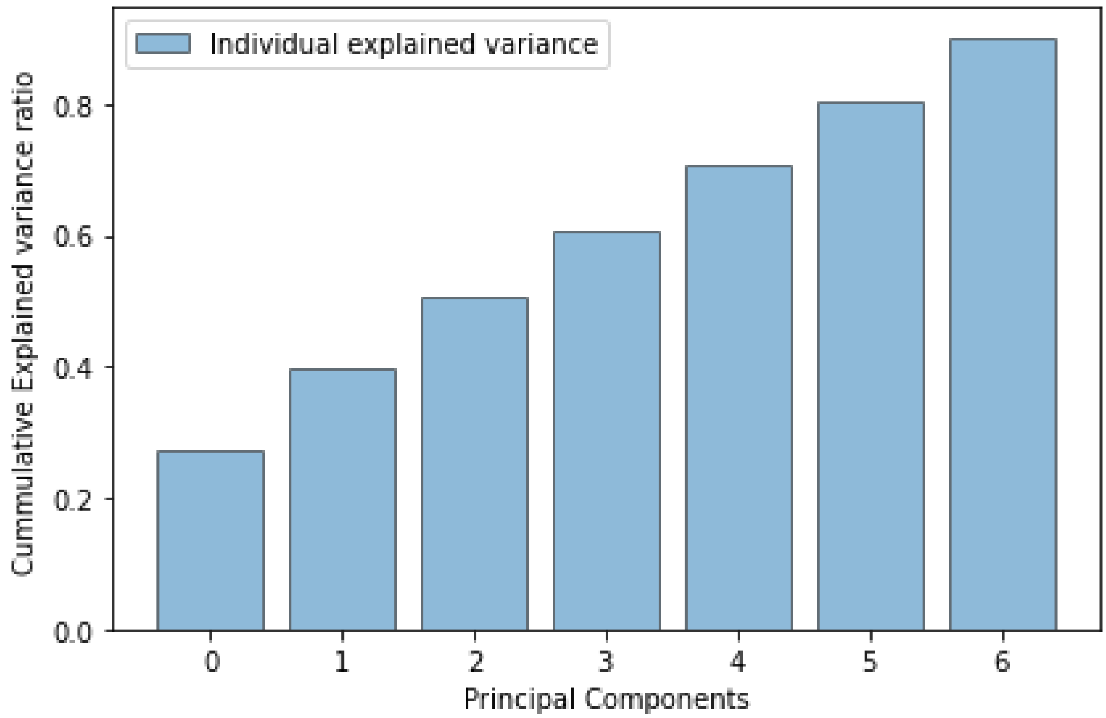



### Data Scaling: Standardization

```
#scaling down values; standardizing
scaler = StandardScaler().fit(churn_df.drop("Churn",axis=1))

#separation of dependent and independent variables
X = scaler.transform(churn_df.drop("Churn",axis=1))
y = churn_df["Churn"]
```

### Dimensionality Reduction: PCA

```
#Reducing dimensions using principal component analysis
pca = PCA(n_components=7)
principalComponents = pca.fit(X)
cumm_expainedvariance =
np.cumsum(principalComponents.explained_variance_ratio_)
principalComponents = pca.transform(X)
principalComponents_df = pd.DataFrame(data = principalComponents
             , columns = ['PC'+str(i+1) for i in range(7)])

#cummulative explained score corresponding to each principal component
plt.bar(range(0,len(cumm_expainedvariance.tolist()[::-1])),
cumm_expainedvariance,
        alpha=0.5, align='center', edgecolor='black',label='Individual
explained variance')

plt.ylabel('Cummulative Explained variance ratio')
plt.xlabel('Principal Components')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```




**Interpretation of Analysis:**

Plot representing the cummulative explained variance score that is the importance or important information each principal component carries.

```
#pca reduced dimension data
principalComponents_df.head()
```

```
        PC1       PC2       PC3       PC4       PC5       PC6
PC7
0  3.138211  0.760811  0.345391 -1.109618 -0.340502 -0.318721
0.253722
1  3.087086 -1.141227  1.137834 -0.165963 -0.238358 -0.356613 -
0.960449
2 -0.711939  0.656365  1.661774 -0.991329 -0.088882 -1.289898
0.525129
3 -0.574675  1.818482  0.349734  0.039606  2.112833 -1.474470
3.707213
4 -1.330330  0.159803  1.325245  0.760565  1.610813 -0.079925
0.826423
```

### 3.1. Split data for train and test

```
#Storing dimensionally reduced data in variable denoted as independent
feature variable
X = principalComponents_df

#splitting data into training and validation sets
X_train, X_test, y_train, y_test =
```

```
train_test_split(principalComponents_df, y, test_size
=0.20,random_state=0, stratify = y)

#Shapes
print("----Shapes of splitted training and test sets----")
print("Shape of Train X: ", X_train.shape)
print("Shape of Train y: ", y_train.shape)
print("Shape of Test X: ", X_test.shape)
print("Shape of Test y: ", y_test.shape)
```

```
----Shapes of splitted training and test sets----
Shape of Train X:  (2666, 7)
Shape of Train y:  (2666,)
Shape of Test X:  (667, 7)
Shape of Test y:  (667,)
```

## 3.2 Functions for models and metrics

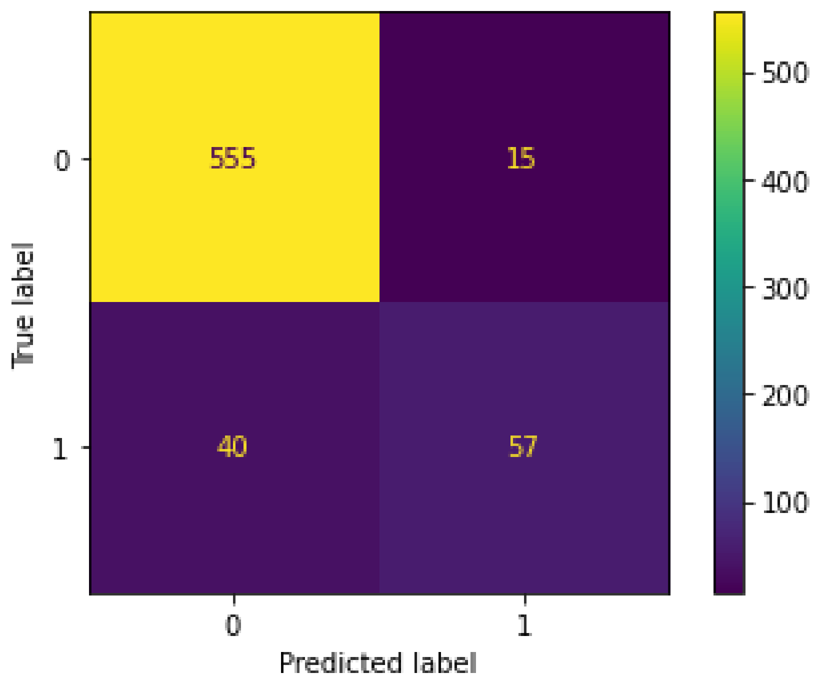
```
from sklearn.metrics import
plot_roc_curve,plot_confusion_matrix,accuracy_score,confusion_matrix

#function for estimator
def Model(model):
    global X,y,X_train, X_test, y_train, y_test
    print(type(model).__name__)
    pred = model.predict(X_test)
    acs = accuracy_score(y_test,pred)
    print("Accuracy Score             :",acs)

    plot_confusion_matrix(model,X,y,cmap="Reds")
    plt.title("Confusion Matrix")
    plt.show()

#function to plot ROC-AUC
def Check(list_of_disp):
    ax = plt.gca()
    for i in list_of_disp:
        i.plot(ax=ax)
    plt.plot([0,1],[0,1],"--",color="k",alpha=0.7)
    plt.title("ROC Curve of Classifiers")
    plt.show()

from sklearn.model_selection import cross_val_score

#function to provide cross-validation score of estimator
def CrossValidationScore(model_list):
    global X,y

    mean_cross_val_score = []
    model_name           = []

    for model in model_list:
```

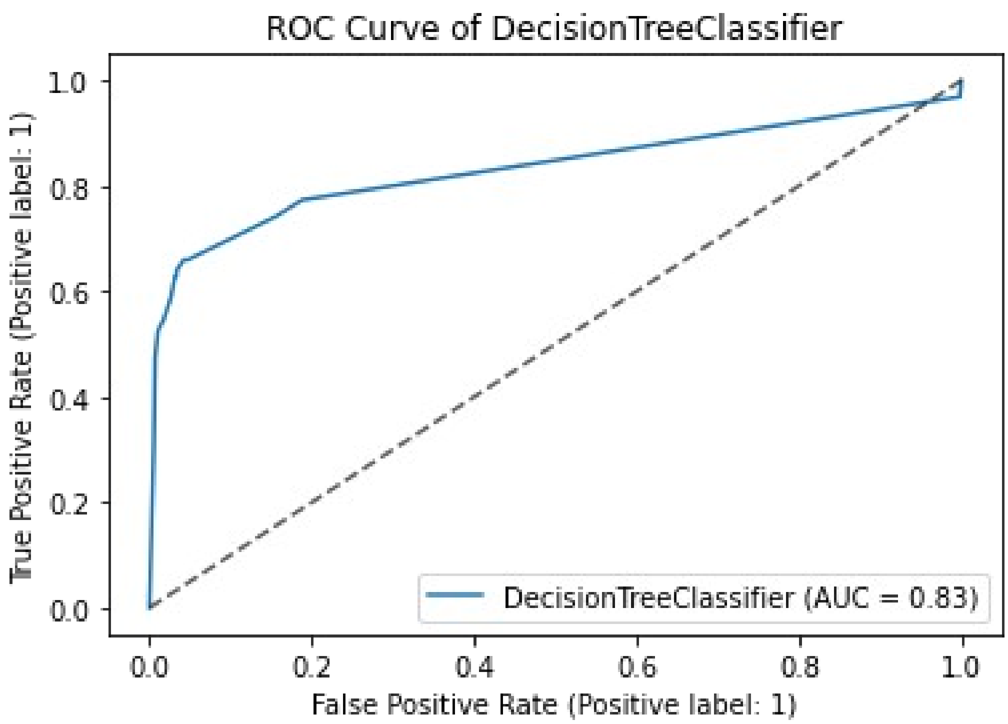
```
        model_name.append(type(model).__name__)

    for i in model_list:
        scores = cross_val_score(i, X, y, cv=5)
        mean_cross_val_score.append(scores.mean())

    cvs = pd.DataFrame({"Model 
Name":model_name,"CVS":mean_cross_val_score})
    return cvs.style.background_gradient("Greens")
```

## 3.3 Models

### *3.3.1. DECISION TREE CLASSIFIER*

```
#DecisionTreeClassifier
dt = DecisionTreeClassifier(random_state=0, max_depth=4, 
min_samples_split=10)
dt.fit(X_train, y_train)
pd = dt.predict(X_test)
plot_confusion_matrix(dt, X_test, y_test)

#cm and roc
dt_disp = plot_roc_curve(dt, X_test, y_test)
plt.title("ROC Curve of {}".format(type(dt).__name__))
plt.plot([0,1],[0,1],"--",color="k",alpha=0.7)
plt.show()

#reults of DecisionTreeClassifier
print("Accuracy Score: ", accuracy_score(y_test, dt.predict(X_test)))
print("Precision Score: ", precision_score(y_test, 
dt.predict(X_test)))
print("Recall Score: ", recall_score(y_test, dt.predict(X_test)))
print("F1 Score: ",f1_score(y_test, dt.predict(X_test)))
```

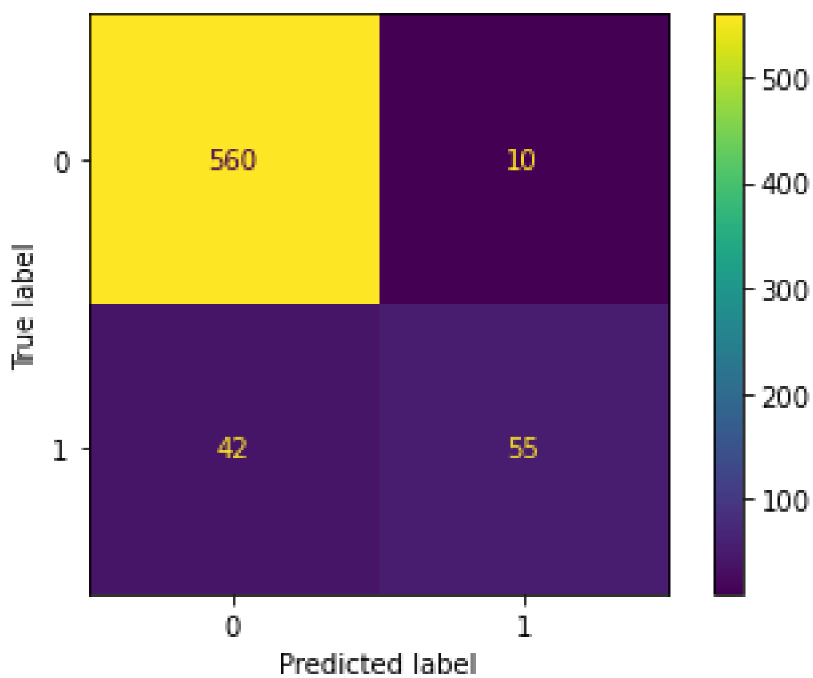


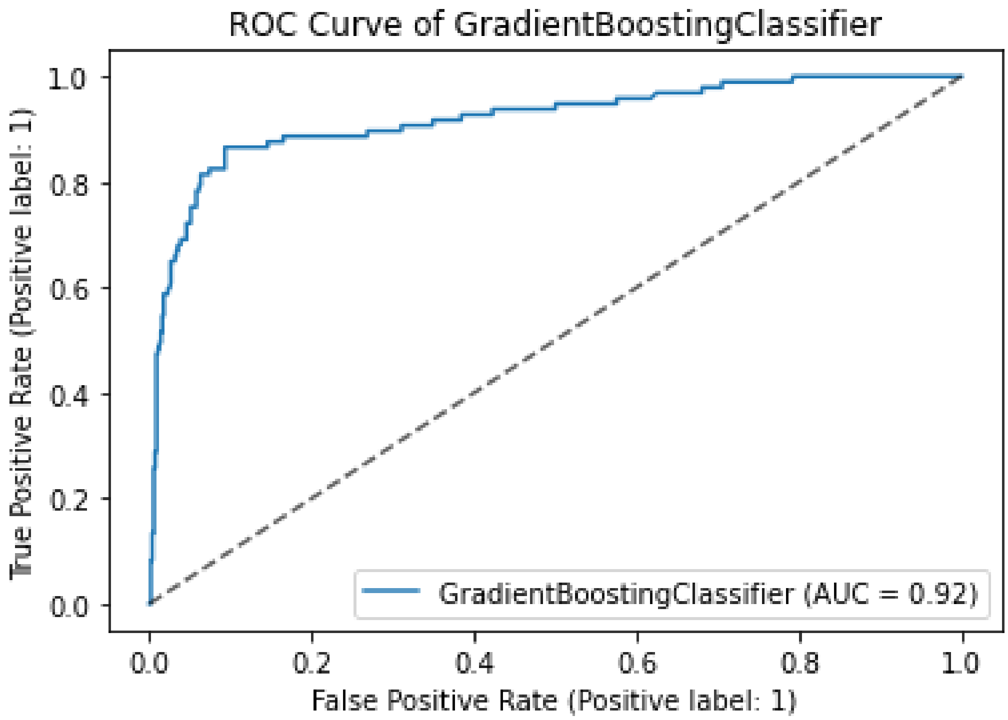




```
Accuracy Score:  0.9175412293853074
Precision Score:  0.7916666666666666
Recall Score:  0.5876288659793815
F1 Score:  0.6745562130177515
```

### 3.3.2. GRADIENT BOOSTING CLASSIFIER

```
#GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators=200, max_depth=2,
random_state=0)
gb.fit(X_train, y_train)
pg = gb.predict(X_test)
plot_confusion_matrix(gb, X_test, y_test)

#cm and roc
gb_disp = plot_roc_curve(gb, X_test, y_test)
plt.title("ROC Curve of {}".format(type(gb).__name__))
plt.plot([0,1],[0,1],"--",color="k",alpha=0.7)
plt.show()

#reults of knn
print("Accuracy Score: ", accuracy_score(y_test, gb.predict(X_test)))
print("Precision Score: ", precision_score(y_test,
gb.predict(X_test)))
print("Recall Score: ", recall_score(y_test, gb.predict(X_test)))
print("F1 Score: ",f1_score(y_test, gb.predict(X_test)))
```



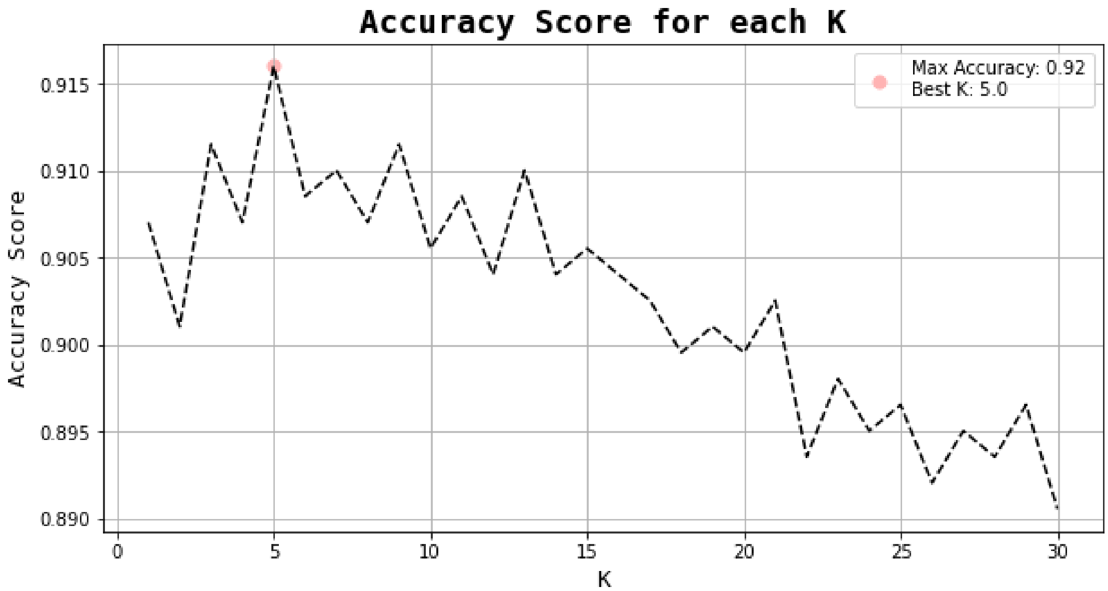



```
Accuracy Score:  0.9220389805097451
Precision Score:  0.8461538461538461
Recall Score:  0.5670103092783505
F1 Score:  0.6790123456790124
```

### *3.3.3. K NEIGHBORS CLASSIFIER*

```
#searching for optimal k value by observing results/score at each k;
k=1 to 29
k_max = 30
accuracy = [[],[]]
for k in range(1,k_max+1):
    mdl = KNeighborsClassifier(n_neighbors=k).fit(X_train,y_train)
    pred = mdl.predict(X_test)
    accuracy[0].append(k)
    accuracy[1].append(accuracy_score(y_test, pred))
accuracy = np.array(accuracy)
max_acc_k = accuracy[1].argmax()
plt.figure(figsize=(10,5))
plt.plot(accuracy[0],accuracy[1], color='k', ls="--")
plt.scatter(x=accuracy[0][max_acc_k], y=accuracy[1][max_acc_k],s=50,
label="Max Accuracy: {}\nBest K: {}".format(round(accuracy[1]
[max_acc_k],2),

accuracy[0][max_acc_k]), color='#ffb3b3')
plt.legend()
plt.grid(True)
plt.title("Accuracy Score for each K" , size=18, fontweight='bold',
fontfamily='monospace')
```

```
plt.xlabel("K", size=13, fontweight='light', fontfamily='monospace')
plt.ylabel('Accuracy Score', size=13, fontweight='light',
fontfamily='monospace')
plt.show()
```



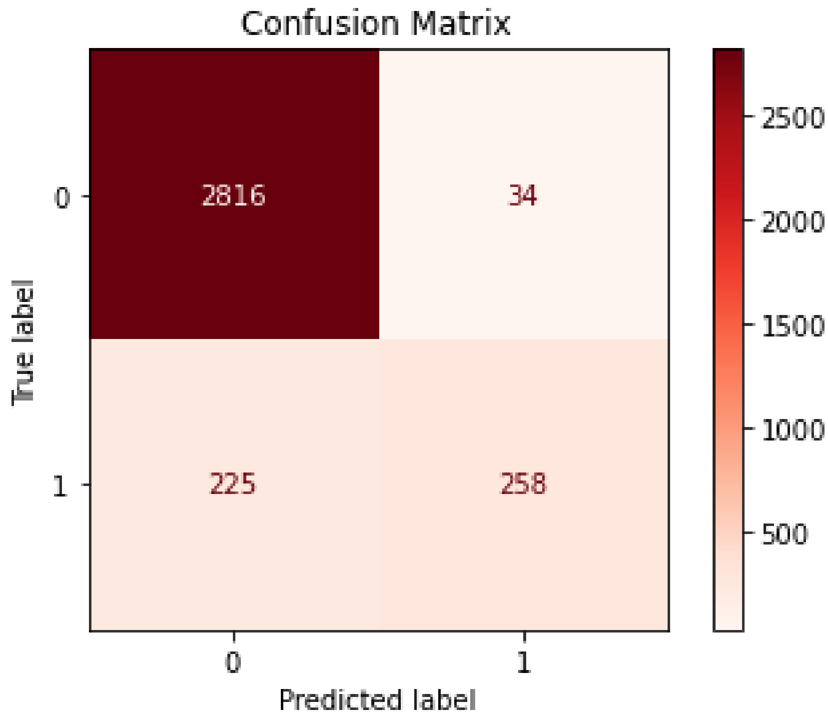

**Interpretation of Analysis:**

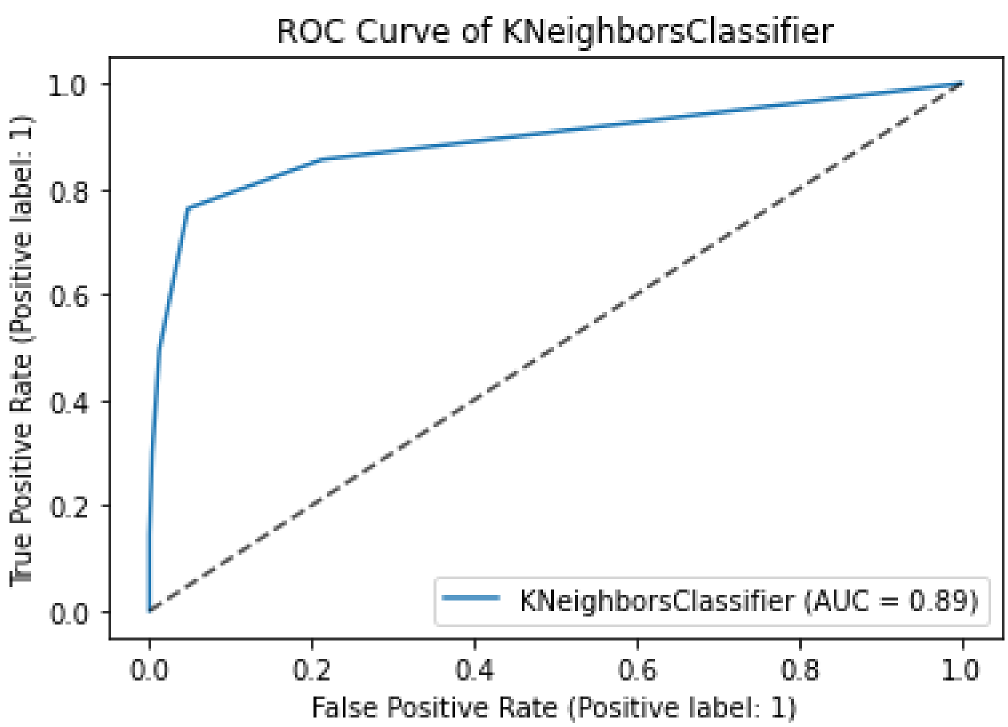
The best K value has come out to be 5 with maximum accuracy score achieved as 92.0 at this particular K. Increase in value of K, further, leads to decrease in accuracy score reflecting not to choose higher K value to prevent a huge degradation in performance of model.

```
#KNN
knn = KNeighborsClassifier(n_neighbors=5).fit(X_train,y_train)
print("Model Installed!")
print("Please Wait for Results..")
Model(knn)

#cm and roc
knn_disp = plot_roc_curve(knn, X_test, y_test)
plt.title("ROC Curve of {}".format(type(knn).__name__))
plt.plot([0,1],[0,1],"--",color="k",alpha=0.7)
plt.show()

#reults of knn
print("Precision Score: ", precision_score(y_test,
knn.predict(X_test)))
print("Recall Score: ", recall_score(y_test, knn.predict(X_test)))
print("F1 Score: ",f1_score(y_test, knn.predict(X_test)))
```

```
Model Installed!
Please Wait for Results..
```

```
KNeighborsClassifier
Accuracy Score             : 0.9160419790104948
```



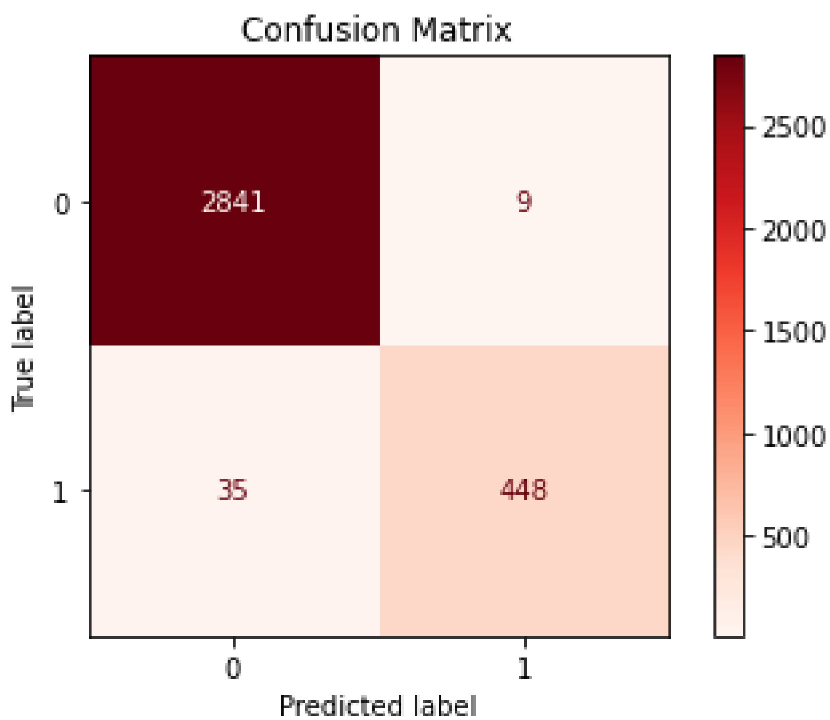



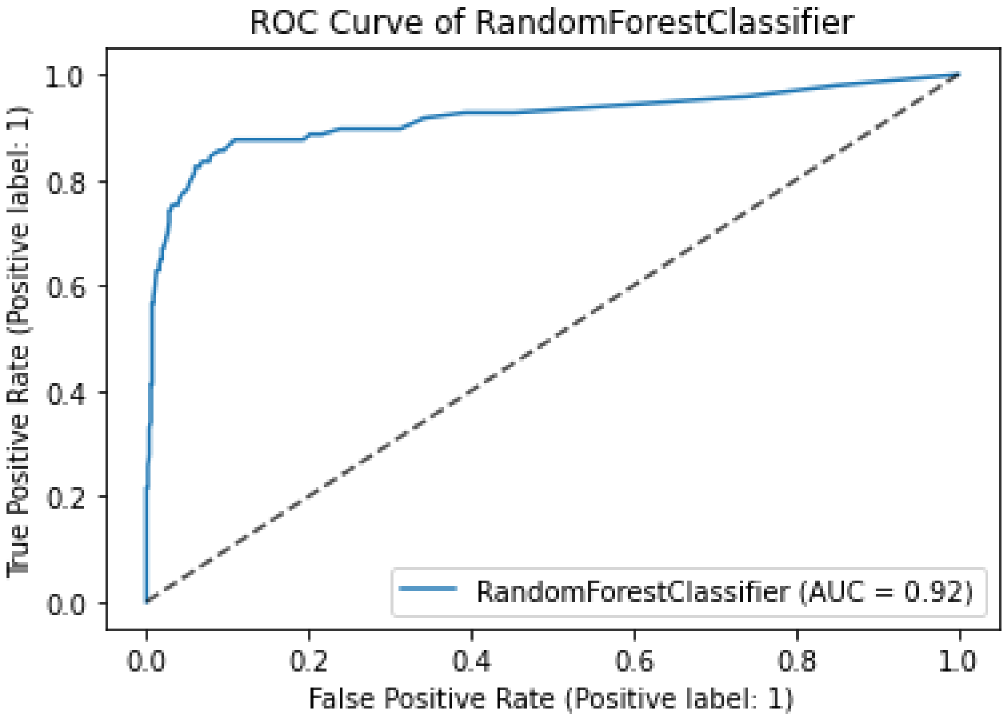

```
Precision Score:  0.8727272727272727
Recall Score:  0.4948453608247423
F1 Score:  0.6315789473684211
```

### 3.3.4. RANDOM FOREST CLASSIFIER

```
#RF
rf = RandomForestClassifier(n_estimators=100).fit(X_train,y_train)
print("Model Installed!")
print("Please Wait for Results..")
Model(rf)

rf_disp = plot_roc_curve(rf, X_test, y_test)
plt.title("ROC Curve of {}".format(type(rf).__name__))
plt.plot([0,1],[0,1],"--",color="k",alpha=0.7)
plt.show()

#results of RF
print("Precision Score: ", precision_score(y_test,
rf.predict(X_test)))
print("Recall Score: ", recall_score(y_test, rf.predict(X_test)))
print("F1 Score: ",f1_score(y_test, rf.predict(X_test)))
```

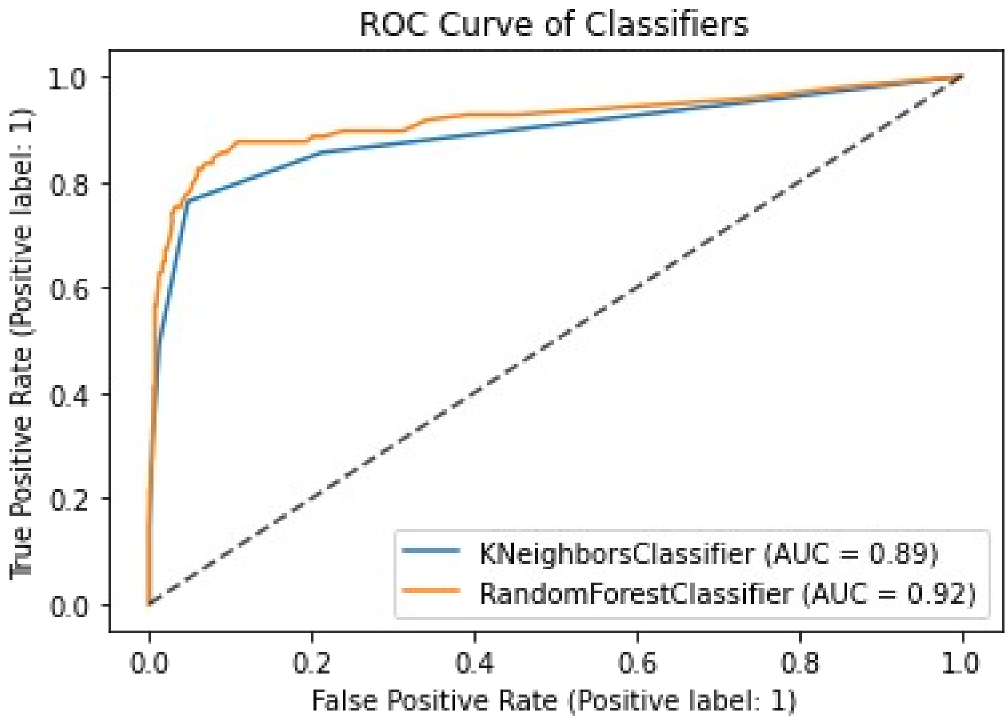
```
Model Installed!
Please Wait for Results..
RandomForestClassifier
Accuracy Score             : 0.9340329835082459
```






```
Precision Score:  0.8732394366197183
Recall Score:  0.6391752577319587
F1 Score:  0.7380952380952381
```

```
#ROC-AUC of knn and RF
list_of_disp = [knn_disp,rf_disp]
Check(list_of_disp)
```




**Interpretation of Analysis:**

The model with the highest Accuracy Score is Random Forest Classifier. At the same time, when looking at the ROC curve of Random Forest Classifier, it is seen that it learns the classes better than other model.

### *3.3.5. ARTIFICIAL NEURAL NETWORK*

#### Multi-layer Perceptron classifier

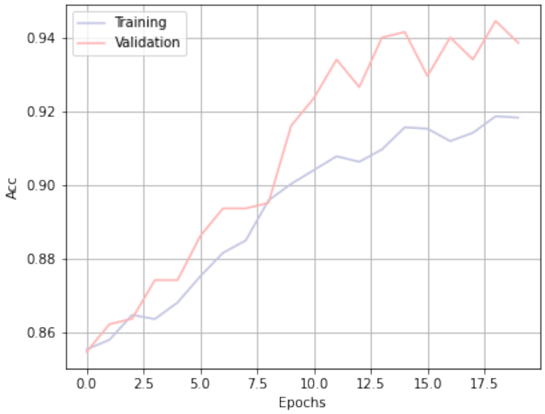
```
#y data to categorical form
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)

X_train = np.array(X_train)
X_test = np.array(X_test)

# ANN model
model = Sequential()

model.add(Dense(64,input_shape=X_train[0].shape,activation="sigmoid"))

model.add(Dense(128,activation="relu"))

model.add(Dense(64,activation="relu"))

model.add(Dense(2,activation="softmax"))
```

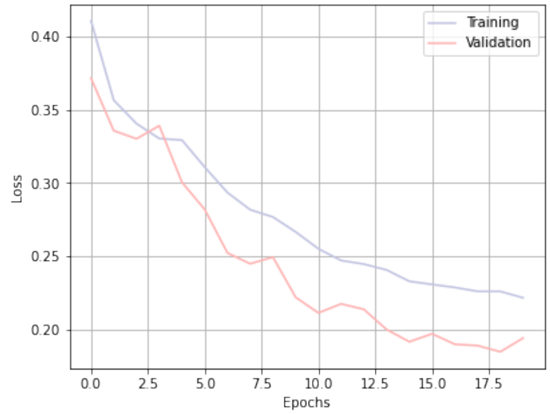
```
model.compile(loss="binary_crossentropy",optimizer="adam",metrics=["ac
c"])

#fitting the network
history =
model.fit(X_train,y_train_cat,batch_size=32,epochs=20,validation_data=
(X_test, y_test_cat))
```

```
Epoch 1/20
84/84 [==============================] - 1s 7ms/step - loss: 0.4104 -
acc: 0.8552 - val_loss: 0.3714 - val_acc: 0.8546
Epoch 2/20
84/84 [==============================] - 0s 4ms/step - loss: 0.3563 -
acc: 0.8578 - val_loss: 0.3356 - val_acc: 0.8621
Epoch 3/20
84/84 [==============================] - 0s 3ms/step - loss: 0.3404 -
acc: 0.8646 - val_loss: 0.3301 - val_acc: 0.8636
Epoch 4/20
84/84 [==============================] - 0s 3ms/step - loss: 0.3303 -
acc: 0.8635 - val_loss: 0.3390 - val_acc: 0.8741
Epoch 5/20
84/84 [==============================] - 0s 3ms/step - loss: 0.3292 -
acc: 0.8680 - val_loss: 0.3004 - val_acc: 0.8741
Epoch 6/20
84/84 [==============================] - 0s 3ms/step - loss: 0.3108 -
acc: 0.8751 - val_loss: 0.2819 - val_acc: 0.8861
Epoch 7/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2934 -
acc: 0.8815 - val_loss: 0.2523 - val_acc: 0.8936
Epoch 8/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2818 -
acc: 0.8848 - val_loss: 0.2449 - val_acc: 0.8936
Epoch 9/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2768 -
acc: 0.8957 - val_loss: 0.2495 - val_acc: 0.8951
Epoch 10/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2666 -
acc: 0.9002 - val_loss: 0.2220 - val_acc: 0.9160
Epoch 11/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2551 -
acc: 0.9040 - val_loss: 0.2114 - val_acc: 0.9235
Epoch 12/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2472 -
acc: 0.9077 - val_loss: 0.2176 - val_acc: 0.9340
Epoch 13/20
84/84 [==============================] - 0s 4ms/step - loss: 0.2447 -
acc: 0.9062 - val_loss: 0.2138 - val_acc: 0.9265
Epoch 14/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2407 -
acc: 0.9096 - val_loss: 0.2001 - val_acc: 0.9400
Epoch 15/20
```

```
84/84 [==============================] - 0s 3ms/step - loss: 0.2330 -
acc: 0.9156 - val_loss: 0.1918 - val_acc: 0.9415
Epoch 16/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2308 -
acc: 0.9152 - val_loss: 0.1972 - val_acc: 0.9295
Epoch 17/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2288 -
acc: 0.9119 - val_loss: 0.1900 - val_acc: 0.9400
Epoch 18/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2261 -
acc: 0.9141 - val_loss: 0.1892 - val_acc: 0.9340
Epoch 19/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2261 -
acc: 0.9186 - val_loss: 0.1849 - val_acc: 0.9445
Epoch 20/20
84/84 [==============================] - 0s 3ms/step - loss: 0.2217 -
acc: 0.9182 - val_loss: 0.1943 - val_acc: 0.9385
```

```
#performance visuals of ANN

#Accuracy
plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.plot(history.history["acc"],color="#C2C4E2")
plt.plot(history.history["val_acc"],color="#ffb3b3")
plt.xlabel("Epochs")
plt.ylabel("Acc")
plt.legend(["Training","Validation"])
plt.grid()

#loss
plt.subplot(1,2,2)
plt.plot(history.history["loss"],color="#C2C4E2")
plt.plot(history.history["val_loss"],color="#ffb3b3")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Training","Validation"])
plt.grid()
plt.show()
```




**Interpretation of Analysis:**

The plots plotted above has shown the accuracy and loss at each epoch compiled during the fitting of neural network; both during training and validation.

- The accuracy during training and validation chasing each other with a similar pace, resulting in average accuracy as 91 and 93 on training and validation respectively.
- The loss during training and validation is moving towards the deepest value, resulting in least average loss.

```
#evaluation score on test set
model.evaluate(X_test, y_test_cat)

21/21 [==============================] - 0s 2ms/step - loss: 0.1943 -
acc: 0.9385

[0.19426468014717102, 0.9385307431221008]

#probabilistic output to class output conversion
pred_class = model.predict(X_test)

def toClass(pred):
    class_ = np.zeros(len(pred))
    for i in range(len(pred)):
        index = pred[i].argmax()
        class_[i] = index

    return class_

from sklearn.metrics import classification_report
print(classification_report(toClass(y_test_cat),toClass(pred_class)))

              precision    recall  f1-score   support

         0.0       0.97      0.96      0.96       570
         1.0       0.78      0.80      0.79        97

    accuracy                           0.94       667
```

```
   macro avg       0.87      0.88      0.88       667
weighted avg       0.94      0.94      0.94       667
```

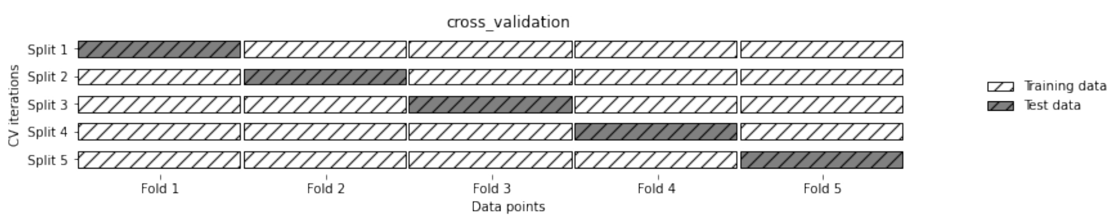
```
#reults of ANN
print("Accuracy Score: ",
accuracy_score(toClass(y_test_cat),toClass(pred_class)))
print("Precision Score: ",
precision_score(toClass(y_test_cat),toClass(pred_class)))
print("Recall Score: ",
recall_score(toClass(y_test_cat),toClass(pred_class)))
print("F1 Score: ",f1_score(toClass(y_test_cat),toClass(pred_class)))
print()
```

```
Accuracy Score:  0.9385307346326837
Precision Score:  0.78
Recall Score:  0.8041237113402062
F1 Score:  0.7918781725888325
```

## 4. Result

*CROSS VALIDATION SCORE*

```
#Overview of cross validation score structure
mglearn.plots.plot_cross_validation();
plt.show();
```




```
#cross validation score achieved by model
from sklearn import svm
model = svm.SVC()
accuracy = cross_val_score(model, X, y, scoring='accuracy', cv = 10)
print(accuracy)
#get the mean of each fold
print("Accuracy of Model with Cross Validation is:",accuracy.mean() *
100)
```

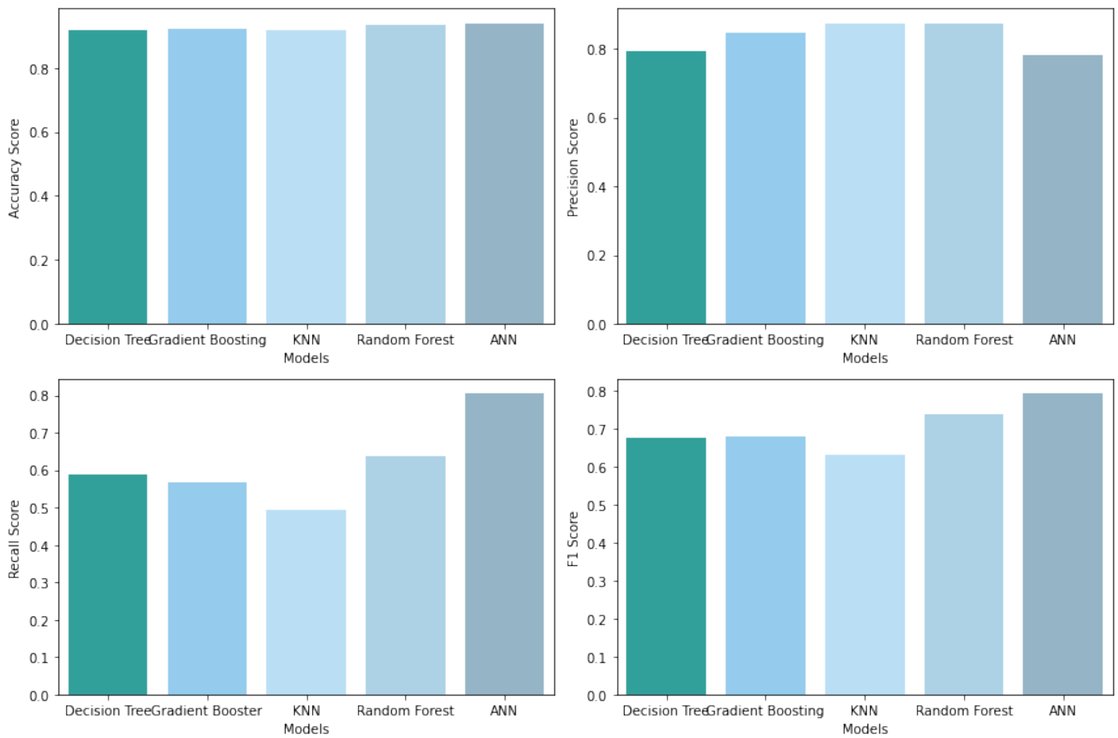
```
[0.92215569 0.90718563 0.91616766 0.89489489 0.91291291 0.92792793
 0.91891892 0.90990991 0.91591592 0.91591592]
Accuracy of Model with Cross Validation is: 91.41905378432324
```

```
#performance plots
ax = plt.figure(figsize=(12,8))
```

```
#accuracy
plt.subplot(2,2,1)
sns.set_color_codes('pastel')
sns.barplot(['Decision Tree', 'Gradient Boosting', 'KNN', 'Random
Forest', 'ANN'],
[accuracy_score(y_test, dt.predict(X_test)),
accuracy_score(y_test, gb.predict(X_test)),
accuracy_score(y_test, knn.predict(X_test)),
accuracy_score(y_test, rf.predict(X_test)),
accuracy_score(toClass(y_test_cat),toClass(pred_class))],
palette=["#20B2AA","#87CEFA","#B0E2FF", "#A4D3EE", "#8DB6CD"])
plt.xlabel("Models")
plt.ylabel("Accuracy Score")

#precision
plt.subplot(2,2,2)
sns.barplot(['Decision Tree', 'Gradient Boosting', 'KNN', 'Random
Forest', 'ANN'],
[precision_score(y_test, dt.predict(X_test)),
precision_score(y_test, gb.predict(X_test)),
precision_score(y_test, knn.predict(X_test)),
precision_score(y_test, rf.predict(X_test)),
precision_score(toClass(y_test_cat),toClass(pred_class))],
palette=["#20B2AA","#87CEFA","#B0E2FF", "#A4D3EE", "#8DB6CD"])
plt.xlabel("Models")
plt.ylabel("Precision Score")

#recall
plt.subplot(2,2,3)
sns.barplot(['Decision Tree', 'Gradient Booster', 'KNN', 'Random
Forest', 'ANN'],
[recall_score(y_test, dt.predict(X_test)),
recall_score(y_test, gb.predict(X_test)),
recall_score(y_test, knn.predict(X_test)),
recall_score(y_test, rf.predict(X_test)),
recall_score(toClass(y_test_cat),toClass(pred_class))],
palette=["#20B2AA","#87CEFA","#B0E2FF", "#A4D3EE", "#8DB6CD"])
plt.xlabel("Models")
plt.ylabel("Recall Score")

#f1score
plt.subplot(2,2,4)
sns.barplot(['Decision Tree', 'Gradient Boosting', 'KNN', 'Random
Forest', 'ANN'],
[f1_score(y_test, dt.predict(X_test)),
f1_score(y_test, gb.predict(X_test)),
f1_score(y_test, knn.predict(X_test)),
f1_score(y_test, rf.predict(X_test)),
f1_score(toClass(y_test_cat),toClass(pred_class))],
```

```
palette=["#20B2AA","#87CEFA","#B0E2FF", "#A4D3EE", "#8DB6CD"])
plt.xlabel("Models")
plt.ylabel("F1 Score")

plt.tight_layout()
plt.show()
```

```
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
KNeighborsClassifier was fitted with feature names
  warnings.warn(
C:\Users\USER\anaconda3\lib\site-packages\sklearn\base.py:450:
UserWarning: X does not have valid feature names, but
RandomForestClassifier was fitted with feature names
  warnings.warn(
```




### Interpretation of Results:

After all the experimentation and analysis, the conclusion that comes out is ANN and Random Forest have worked so well from the beginning to the last and have classified the churn category with least error.

## Overall report of all models

**Decision Tree:**

- Accuracy Score: 0.9175412293853074
- Precision Score: 0.7916666666666666
- Recall Score: 0.5876288659793815
- F1 Score: 0.6745562130177515

**Gradient Boosting:**

- Accuracy Score: 0.9220389805097451
- Precision Score: 0.8461538461538461
- Recall Score: 0.5670103092783505
- F1 Score: 0.6790123456790124

**KNN:**

- Cross-validation-Score: 0.900690
- Accuracy Score: 0.9160419790104948
- Precision Score: 0.8727272727272727

- Recall Score: 0.4948453608247423
- F1 Score: 0.6315789473684211

**Random Forest:**

- Cross-validation-Score: 0.917195
- Accuracy Score: 0.9310344827586207
- Precision Score: 0.8695652173913043
- Recall Score: 0.6185567010309279
- F1 Score: 0.7228915662650603

**ANN:**

- Accuracy Score: 0.9355322338830585
- Precision Score: 0.9090909090909091
- Recall Score: 0.6185567010309279
- F1 Score: 0.7361963190184049

## CONCLUSION

The score of **Artificial Neural Network and Random Forest Classifier** in classifying the customer churn has been observed more active and accurate resulting in best estimators for such cases. Also, the KNN has been seen chasing both other classifiers with competitive scores but lagged behind with few percent declining the accuracy, but cannot be ignored for future improvements. Comparison between both shows, the **Artificial Neural Network(ANN)** is having greater precision score as well as F1-score which reflects its fine behavior in identifying the classes and predicting them positively with any fail.