CUSTOMER CHURN PREDICTION

**TEAM LEADER:** GURU VARSHENI A

**REGISTER NO:** 211521244015

**TEAM MEMBER:** DHANUSHREE.S

**REGISTER NO:** 211521244011

**TEAM MEMBER:** ISHA.K

**REGISTER NO:** 211521244018

**TEAM MEMBER:** POOJA.S

**REGISTER NO:** 211521244039

* ***PREDICTIVE MODELLING:***

Before applying any machine learning algorithm, the data needs to be prepared properly, which involves defining the churn event and the time period, collecting and integrating relevant data sources, cleaning and transforming the data, exploring and analysing it, and selecting and creating features.

Churn prediction can be done using several machine learning algorithms, such as logistic regression, decision tree, random forest, k-means, and hierarchical clustering.

**Logistic regression** models the relationship between features and outcomes using a logistic function and is simple and fast, but can be prone to overfitting or underfitting.

**Decision trees** split data into branches based on features and are intuitive and flexible, but can be sensitive to noise, variance, and bias. Random forests combine multiple decision trees and are robust, accurate, and versatile, but complex and slow.

**K-means** assigns customers to one of k clusters based on their distance to the cluster centre and is simple, scalable, and efficient; however, it requires choosing the number of clusters and can be sensitive to initialization and outliers.

**Random Forest** is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemblelearning, which is a process of combining multiple classifiersto solve a complex problem and to improve the performance of the model.

**Support Vector Machine (SVM)** is a relatively simple Supervised Machine Learning Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data.

**Gradient Boosting** is a powerful boosting algorithm that combines several weak learners into strong learners, in which each new model is trained to minimize the loss function such as mean squared error or cross-entropy of the previous model using gradient descent.

**K-Nearest Neighbours** is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining, and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data

**Naive Bayes** are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality. They find use when the dimensionality of the inputs is high. Complex classification problems can also be implemented by using Naive Bayes Classifier.

After applying the machine learning algorithms, the next step is to evaluate and improve their performance and accuracy. This requires splitting the data into training, validation, and testing sets, choosing an appropriate metric or criteria, comparing and selecting the best algorithm and interpreting and explaining the algorithm.

The algorithm can be optimized by adjusting parameters or selecting features and interpreted by analysing coefficients or importance scores.

The code for the project “**CUSTOMER CHURN PREDICTION**” is given below. We made use of several modules of which some important ones are pandas, NumPy, matplotlib and so on.

## Customer Churn Prediction

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import plotly.express as px  
import plotly.graph\_objects as go  
from plotly.subplots import make\_subplots  
import warnings  
warnings.filterwarnings('ignore')  
from sklearn.preprocessing import LabelEncoder  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.preprocessing import StandardScaler  
from sklearn.preprocessing import LabelEncoder  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.neural\_network import MLPClassifier  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import GradientBoostingClassifier  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from xgboost import XGBClassifier  
from catboost import CatBoostClassifier  
from sklearn import metrics  
from sklearn.metrics import roc\_curve  
from sklearn.metrics import recall\_score, confusion\_matrix, precision\_score, f1\_score, accuracy\_score, classification\_report  
from sklearn.ensemble import VotingClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, fbeta\_score  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
from sklearn.model\_selection import cross\_val\_score  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import ShuffleSplit  
from sklearn.model\_selection import KFold  
from sklearn import feature\_selection  
from sklearn import model\_selection  
from sklearn import metrics  
from sklearn.metrics import classification\_report, precision\_recall\_curve  
from sklearn.metrics import auc, roc\_auc\_score, roc\_curve  
from sklearn.metrics import make\_scorer, recall\_score, log\_loss  
from sklearn.metrics import average\_precision\_score  
#Standard libraries for data visualization:

data = pd.read\_csv("data.csv")  
data.head()

customerID gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 7590-VHVEG Female 0 Yes No 1 No   
1 5575-GNVDE Male 0 No No 34 Yes   
2 3668-QPYBK Male 0 No No 2 Yes   
3 7795-CFOCW Male 0 No No 45 No   
4 9237-HQITU Female 0 No No 2 Yes   
  
 MultipleLines InternetService OnlineSecurity ... DeviceProtection \  
0 No phone service DSL No ... No   
1 No DSL Yes ... Yes   
2 No DSL Yes ... No   
3 No phone service DSL Yes ... Yes   
4 No Fiber optic No ... No   
  
 TechSupport StreamingTV StreamingMovies Contract PaperlessBilling \  
0 No No No Month-to-month Yes   
1 No No No One year No   
2 No No No Month-to-month Yes   
3 Yes No No One year No   
4 No No No Month-to-month Yes   
  
 PaymentMethod MonthlyCharges TotalCharges Churn   
0 Electronic check 29.85 29.85 No   
1 Mailed check 56.95 1889.5 No   
2 Mailed check 53.85 108.15 Yes   
3 Bank transfer (automatic) 42.30 1840.75 No   
4 Electronic check 70.70 151.65 Yes   
  
[5 rows x 21 columns]

data.isnull().any().any()

False

data.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 7043 entries, 0 to 7042  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 customerID 7043 non-null object   
 1 gender 7043 non-null object   
 2 SeniorCitizen 7043 non-null int64   
 3 Partner 7043 non-null object   
 4 Dependents 7043 non-null object   
 5 tenure 7043 non-null int64   
 6 PhoneService 7043 non-null object   
 7 MultipleLines 7043 non-null object   
 8 InternetService 7043 non-null object   
 9 OnlineSecurity 7043 non-null object   
 10 OnlineBackup 7043 non-null object   
 11 DeviceProtection 7043 non-null object   
 12 TechSupport 7043 non-null object   
 13 StreamingTV 7043 non-null object   
 14 StreamingMovies 7043 non-null object   
 15 Contract 7043 non-null object   
 16 PaperlessBilling 7043 non-null object   
 17 PaymentMethod 7043 non-null object   
 18 MonthlyCharges 7043 non-null float64  
 19 TotalCharges 7043 non-null object   
 20 Churn 7043 non-null object   
dtypes: float64(1), int64(2), object(18)  
memory usage: 1.1+ MB

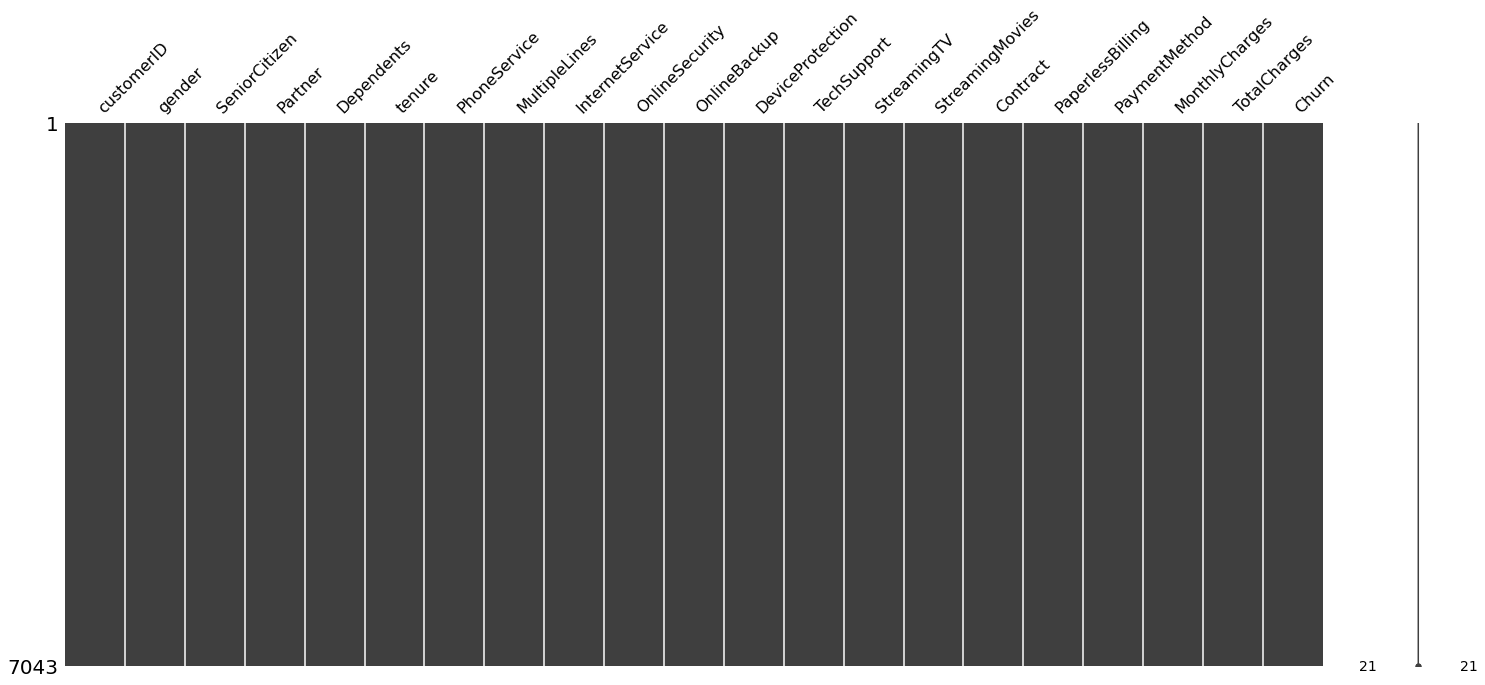
data.shape

(7043, 21)

### Visualize missing values

import missingno as msno  
msno.matrix(data)

<AxesSubplot:>



data = data.drop(["customerID"], axis = 1)  
data.head()

gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 Female 0 Yes No 1 No   
1 Male 0 No No 34 Yes   
2 Male 0 No No 2 Yes   
3 Male 0 No No 45 No   
4 Female 0 No No 2 Yes   
  
 MultipleLines InternetService OnlineSecurity OnlineBackup \  
0 No phone service DSL No Yes   
1 No DSL Yes No   
2 No DSL Yes Yes   
3 No phone service DSL Yes No   
4 No Fiber optic No No   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
0 No No No No Month-to-month   
1 Yes No No No One year   
2 No No No No Month-to-month   
3 Yes Yes No No One year   
4 No No No No Month-to-month   
  
 PaperlessBilling PaymentMethod MonthlyCharges TotalCharges \  
0 Yes Electronic check 29.85 29.85   
1 No Mailed check 56.95 1889.5   
2 Yes Mailed check 53.85 108.15   
3 No Bank transfer (automatic) 42.30 1840.75   
4 Yes Electronic check 70.70 151.65   
  
 Churn   
0 No   
1 No   
2 Yes   
3 No   
4 Yes

data[data["TotalCharges"] == ' ']

gender SeniorCitizen Partner Dependents tenure PhoneService \  
488 Female 0 Yes Yes 0 No   
753 Male 0 No Yes 0 Yes   
936 Female 0 Yes Yes 0 Yes   
1082 Male 0 Yes Yes 0 Yes   
1340 Female 0 Yes Yes 0 No   
3331 Male 0 Yes Yes 0 Yes   
3826 Male 0 Yes Yes 0 Yes   
4380 Female 0 Yes Yes 0 Yes   
5218 Male 0 Yes Yes 0 Yes   
6670 Female 0 Yes Yes 0 Yes   
6754 Male 0 No Yes 0 Yes   
  
 MultipleLines InternetService OnlineSecurity \  
488 No phone service DSL Yes   
753 No No No internet service   
936 No DSL Yes   
1082 Yes No No internet service   
1340 No phone service DSL Yes   
3331 No No No internet service   
3826 Yes No No internet service   
4380 No No No internet service   
5218 No No No internet service   
6670 Yes DSL No   
6754 Yes DSL Yes   
  
 OnlineBackup DeviceProtection TechSupport \  
488 No Yes Yes   
753 No internet service No internet service No internet service   
936 Yes Yes No   
1082 No internet service No internet service No internet service   
1340 Yes Yes Yes   
3331 No internet service No internet service No internet service   
3826 No internet service No internet service No internet service   
4380 No internet service No internet service No internet service   
5218 No internet service No internet service No internet service   
6670 Yes Yes Yes   
6754 Yes No Yes   
  
 StreamingTV StreamingMovies Contract PaperlessBilling \  
488 Yes No Two year Yes   
753 No internet service No internet service Two year No   
936 Yes Yes Two year No   
1082 No internet service No internet service Two year No   
1340 Yes No Two year No   
3331 No internet service No internet service Two year No   
3826 No internet service No internet service Two year No   
4380 No internet service No internet service Two year No   
5218 No internet service No internet service One year Yes   
6670 Yes No Two year No   
6754 No No Two year Yes   
  
 PaymentMethod MonthlyCharges TotalCharges Churn   
488 Bank transfer (automatic) 52.55 No   
753 Mailed check 20.25 No   
936 Mailed check 80.85 No   
1082 Mailed check 25.75 No   
1340 Credit card (automatic) 56.05 No   
3331 Mailed check 19.85 No   
3826 Mailed check 25.35 No   
4380 Mailed check 20.00 No   
5218 Mailed check 19.70 No   
6670 Mailed check 73.35 No   
6754 Bank transfer (automatic) 61.90 No

data['TotalCharges'] = pd.to\_numeric(data.TotalCharges, errors='coerce')  
data.isnull().sum()

gender 0  
SeniorCitizen 0  
Partner 0  
Dependents 0  
tenure 0  
PhoneService 0  
MultipleLines 0  
InternetService 0  
OnlineSecurity 0  
OnlineBackup 0  
DeviceProtection 0  
TechSupport 0  
StreamingTV 0  
StreamingMovies 0  
Contract 0  
PaperlessBilling 0  
PaymentMethod 0  
MonthlyCharges 0  
TotalCharges 11  
Churn 0  
dtype: int64

### There are 11 records with missing Total charges

data[data["tenure"] == 0]

gender SeniorCitizen Partner Dependents tenure PhoneService \  
488 Female 0 Yes Yes 0 No   
753 Male 0 No Yes 0 Yes   
936 Female 0 Yes Yes 0 Yes   
1082 Male 0 Yes Yes 0 Yes   
1340 Female 0 Yes Yes 0 No   
3331 Male 0 Yes Yes 0 Yes   
3826 Male 0 Yes Yes 0 Yes   
4380 Female 0 Yes Yes 0 Yes   
5218 Male 0 Yes Yes 0 Yes   
6670 Female 0 Yes Yes 0 Yes   
6754 Male 0 No Yes 0 Yes   
  
 MultipleLines InternetService OnlineSecurity \  
488 No phone service DSL Yes   
753 No No No internet service   
936 No DSL Yes   
1082 Yes No No internet service   
1340 No phone service DSL Yes   
3331 No No No internet service   
3826 Yes No No internet service   
4380 No No No internet service   
5218 No No No internet service   
6670 Yes DSL No   
6754 Yes DSL Yes   
  
 OnlineBackup DeviceProtection TechSupport \  
488 No Yes Yes   
753 No internet service No internet service No internet service   
936 Yes Yes No   
1082 No internet service No internet service No internet service   
1340 Yes Yes Yes   
3331 No internet service No internet service No internet service   
3826 No internet service No internet service No internet service   
4380 No internet service No internet service No internet service   
5218 No internet service No internet service No internet service   
6670 Yes Yes Yes   
6754 Yes No Yes   
  
 StreamingTV StreamingMovies Contract PaperlessBilling \  
488 Yes No Two year Yes   
753 No internet service No internet service Two year No   
936 Yes Yes Two year No   
1082 No internet service No internet service Two year No   
1340 Yes No Two year No   
3331 No internet service No internet service Two year No   
3826 No internet service No internet service Two year No   
4380 No internet service No internet service Two year No   
5218 No internet service No internet service One year Yes   
6670 Yes No Two year No   
6754 No No Two year Yes   
  
 PaymentMethod MonthlyCharges TotalCharges Churn   
488 Bank transfer (automatic) 52.55 NaN No   
753 Mailed check 20.25 NaN No   
936 Mailed check 80.85 NaN No   
1082 Mailed check 25.75 NaN No   
1340 Credit card (automatic) 56.05 NaN No   
3331 Mailed check 19.85 NaN No   
3826 Mailed check 25.35 NaN No   
4380 Mailed check 20.00 NaN No   
5218 Mailed check 19.70 NaN No   
6670 Mailed check 73.35 NaN No   
6754 Bank transfer (automatic) 61.90 NaN No

data.drop(labels=data[data["tenure"] == 0].index, axis = 0, inplace = True)

data.fillna(data["TotalCharges"].mean())

gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 Female 0 Yes No 1 No   
1 Male 0 No No 34 Yes   
2 Male 0 No No 2 Yes   
3 Male 0 No No 45 No   
4 Female 0 No No 2 Yes   
... ... ... ... ... ... ...   
7038 Male 0 Yes Yes 24 Yes   
7039 Female 0 Yes Yes 72 Yes   
7040 Female 0 Yes Yes 11 No   
7041 Male 1 Yes No 4 Yes   
7042 Male 0 No No 66 Yes   
  
 MultipleLines InternetService OnlineSecurity OnlineBackup \  
0 No phone service DSL No Yes   
1 No DSL Yes No   
2 No DSL Yes Yes   
3 No phone service DSL Yes No   
4 No Fiber optic No No   
... ... ... ... ...   
7038 Yes DSL Yes No   
7039 Yes Fiber optic No Yes   
7040 No phone service DSL Yes No   
7041 Yes Fiber optic No No   
7042 No Fiber optic Yes No   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
0 No No No No Month-to-month   
1 Yes No No No One year   
2 No No No No Month-to-month   
3 Yes Yes No No One year   
4 No No No No Month-to-month   
... ... ... ... ... ...   
7038 Yes Yes Yes Yes One year   
7039 Yes No Yes Yes One year   
7040 No No No No Month-to-month   
7041 No No No No Month-to-month   
7042 Yes Yes Yes Yes Two year   
  
 PaperlessBilling PaymentMethod MonthlyCharges \  
0 Yes Electronic check 29.85   
1 No Mailed check 56.95   
2 Yes Mailed check 53.85   
3 No Bank transfer (automatic) 42.30   
4 Yes Electronic check 70.70   
... ... ... ...   
7038 Yes Mailed check 84.80   
7039 Yes Credit card (automatic) 103.20   
7040 Yes Electronic check 29.60   
7041 Yes Mailed check 74.40   
7042 Yes Bank transfer (automatic) 105.65   
  
 TotalCharges Churn   
0 29.85 No   
1 1889.50 No   
2 108.15 Yes   
3 1840.75 No   
4 151.65 Yes   
... ... ...   
7038 1990.50 No   
7039 7362.90 No   
7040 346.45 No   
7041 306.60 Yes   
7042 6844.50 No   
  
[7032 rows x 20 columns]

data['TotalCharges'] = pd.to\_numeric(data.TotalCharges, errors='coerce')  
data.isnull().sum()

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

data.SeniorCitizen.unique()

array([0, 1], dtype=int64)

data.SeniorCitizen = data.SeniorCitizen.map({0: "No", 1: "Yes"})  
data.head()

gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 Female No Yes No 1 No   
1 Male No No No 34 Yes   
2 Male No No No 2 Yes   
3 Male No No No 45 No   
4 Female No No No 2 Yes   
  
 MultipleLines InternetService OnlineSecurity OnlineBackup \  
0 No phone service DSL No Yes   
1 No DSL Yes No   
2 No DSL Yes Yes   
3 No phone service DSL Yes No   
4 No Fiber optic No No   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
0 No No No No Month-to-month   
1 Yes No No No One year   
2 No No No No Month-to-month   
3 Yes Yes No No One year   
4 No No No No Month-to-month   
  
 PaperlessBilling PaymentMethod MonthlyCharges TotalCharges \  
0 Yes Electronic check 29.85 29.85   
1 No Mailed check 56.95 1889.50   
2 Yes Mailed check 53.85 108.15   
3 No Bank transfer (automatic) 42.30 1840.75   
4 Yes Electronic check 70.70 151.65   
  
 Churn   
0 No   
1 No   
2 Yes   
3 No   
4 Yes

data.InternetService.describe(include=["object", "bool"])

count 7032  
unique 3  
top Fiber optic  
freq 3096  
Name: InternetService, dtype: object

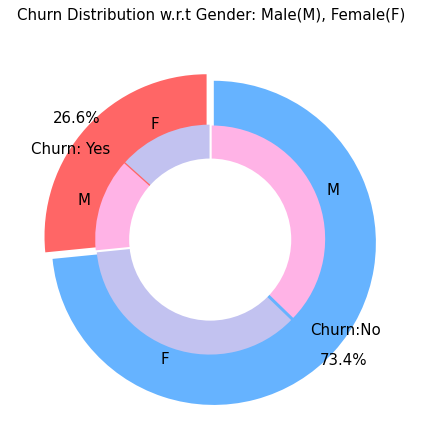
data.Churn[data.Churn == "No"].groupby(by = data.gender).count()

gender  
Female 2544  
Male 2619  
Name: Churn, dtype: int64

data.Churn[data.Churn == "Yes"].groupby(by = data.gender).count()

gender  
Female 939  
Male 930  
Name: Churn, dtype: int64

plt.figure(figsize=(6, 6))  
labels =["Churn: Yes","Churn:No"]  
values = [1869,5163]  
labels\_gender = ["F","M","F","M"]  
sizes\_gender = [939,930 , 2544,2619]  
colors = ['#ff6666', '#66b3ff']  
colors\_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']  
explode = (0.3,0.3)  
explode\_gender = (0.1,0.1,0.1,0.1)  
textprops = {"fontsize":15}  
#Plot  
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangle=90,frame=True, explode=explode,radius=10, textprops =textprops, counterclock = True, )  
plt.pie(sizes\_gender,labels=labels\_gender,colors=colors\_gender,startangle=90, explode=explode\_gender,radius=7, textprops =textprops, counterclock = True, )  
#Draw circle  
centre\_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)  
fig = plt.gcf()  
fig.gca().add\_artist(centre\_circle)  
  
plt.title('Churn Distribution w.r.t Gender: Male(M), Female(F)', fontsize=15, y=1.1)  
  
# show plot  
  
plt.axis('equal')  
plt.tight\_layout()  
plt.show()



#### Customers with monthly contract are more likely to churn

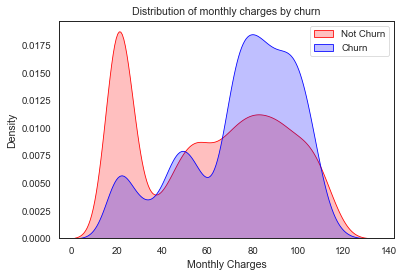
data[data["gender"]=="Male"][["InternetService", "Churn"]].value\_counts()

InternetService Churn  
DSL No 992  
Fiber optic No 910  
No No 717  
Fiber optic Yes 633  
DSL Yes 240  
No Yes 57  
dtype: int64

data[data["gender"]=="Female"][["InternetService", "Churn"]].value\_counts()

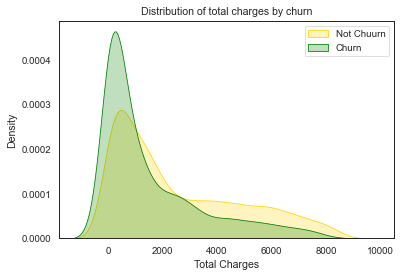
InternetService Churn  
DSL No 965  
Fiber optic No 889  
No No 690  
Fiber optic Yes 664  
DSL Yes 219  
No Yes 56  
dtype: int64

sns.set\_context("paper",font\_scale=1.1)  
ax = sns.kdeplot(data.MonthlyCharges[(data["Churn"] == 'No') ],  
 color="Red", shade = True);  
ax = sns.kdeplot(data.MonthlyCharges[(data["Churn"] == 'Yes') ],  
 ax =ax, color="Blue", shade= True);  
ax.legend(["Not Churn","Churn"],loc='upper right');  
ax.set\_ylabel('Density');  
ax.set\_xlabel('Monthly Charges');  
ax.set\_title('Distribution of monthly charges by churn');



### Customers with higher monthly charges are more likely to churn

ax = sns.kdeplot(data.TotalCharges[(data["Churn"] == 'No') ],  
 color="Gold", shade = True);  
ax = sns.kdeplot(data.TotalCharges[(data["Churn"] == 'Yes') ],  
 ax =ax, color="Green", shade= True);  
ax.legend(["Not Chuurn","Churn"],loc='upper right');  
ax.set\_ylabel('Density');  
ax.set\_xlabel('Total Charges');  
ax.set\_title('Distribution of total charges by churn');



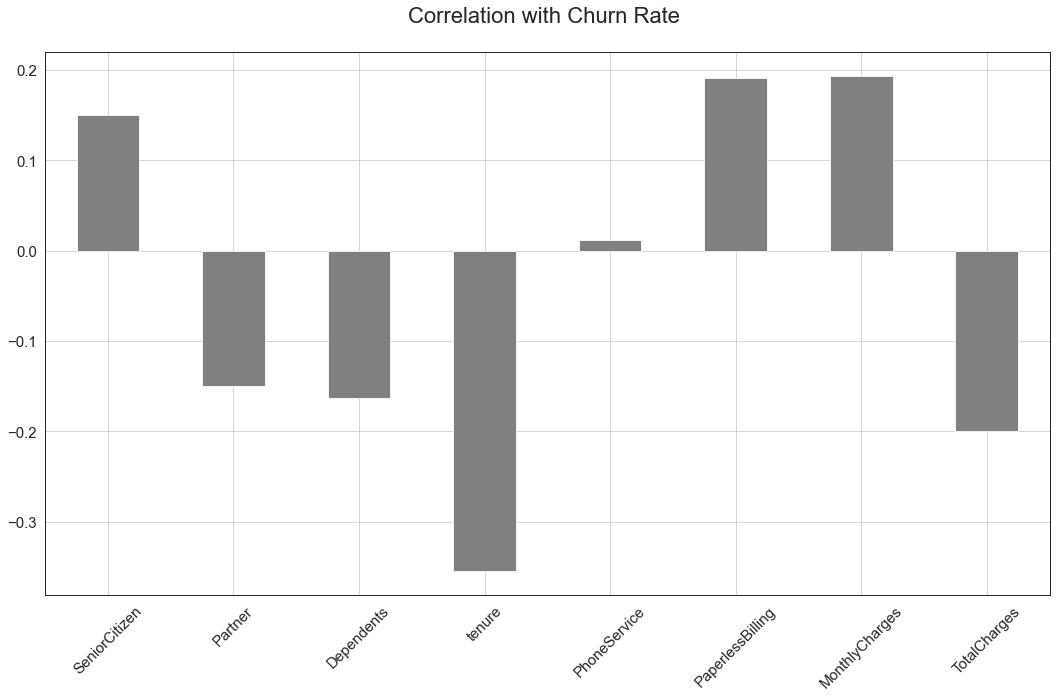
New customers are more likely to churn

#Create a label encoder object  
le = LabelEncoder()  
# Label Encoding will be used for columns with 2 or less unique  
values  
le\_count = 0  
for col in data.columns[1:]:  
 if data[col].dtype == 'object':  
 if len(list(data[col].unique())) <= 2:  
 le.fit(data[col])  
 data[col] = le.transform(data[col])  
 le\_count += 1  
print('{} columns were label encoded.'.format(le\_count))

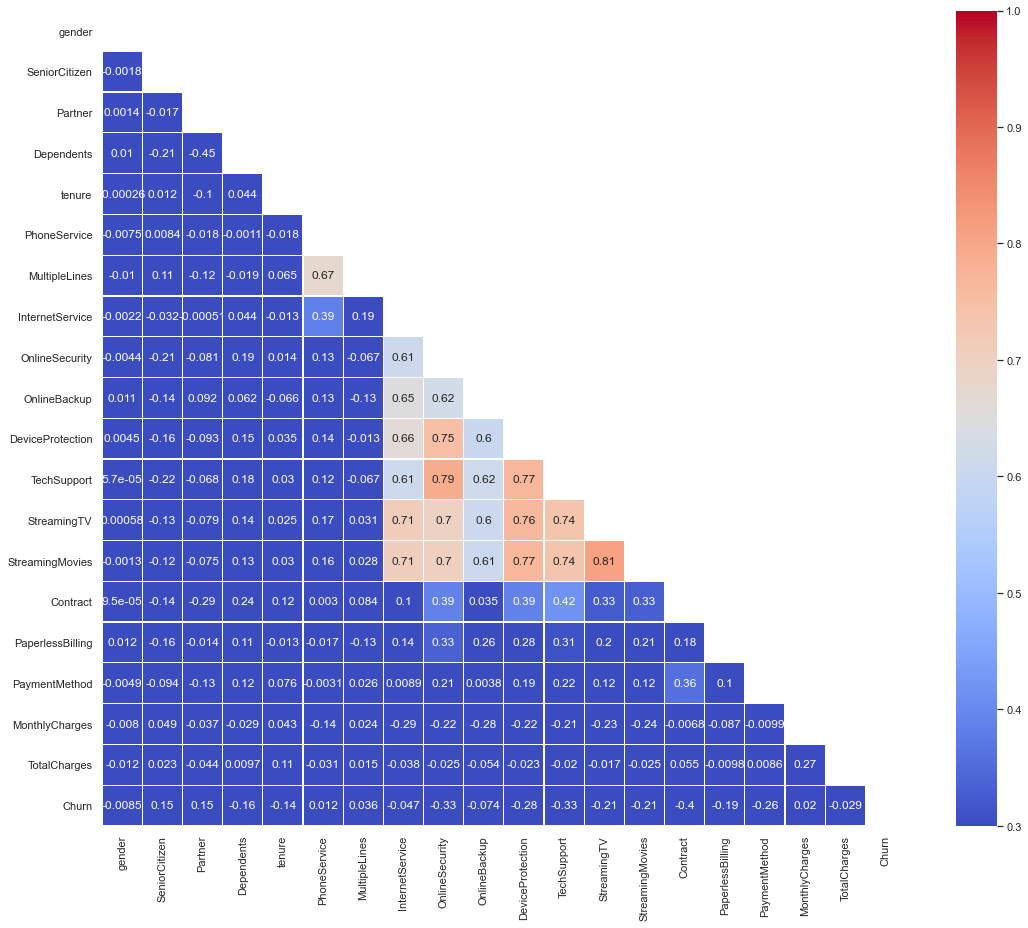
6 columns were label encoded.

data2 = data[['SeniorCitizen', 'Partner', 'Dependents',  
 'tenure', 'PhoneService', 'PaperlessBilling',  
 'MonthlyCharges', 'TotalCharges']]  
  
correlations = data2.corrwith(data.Churn)  
correlations = correlations[correlations!=1]  
positive\_correlations = correlations[correlations >0].sort\_values(ascending = False)  
negative\_correlations =correlations[correlations<0].sort\_values(ascending = False)  
  
correlations.plot.bar(  
 figsize = (18, 10),  
 fontsize = 15,  
 color = 'grey',  
 rot = 45, grid = True)  
plt.title('Correlation with Churn Rate \n',  
horizontalalignment="center", fontstyle = "normal",  
fontsize = "22", fontfamily = "sans-serif")

Text(0.5, 1.0, 'Correlation with Churn Rate \n')

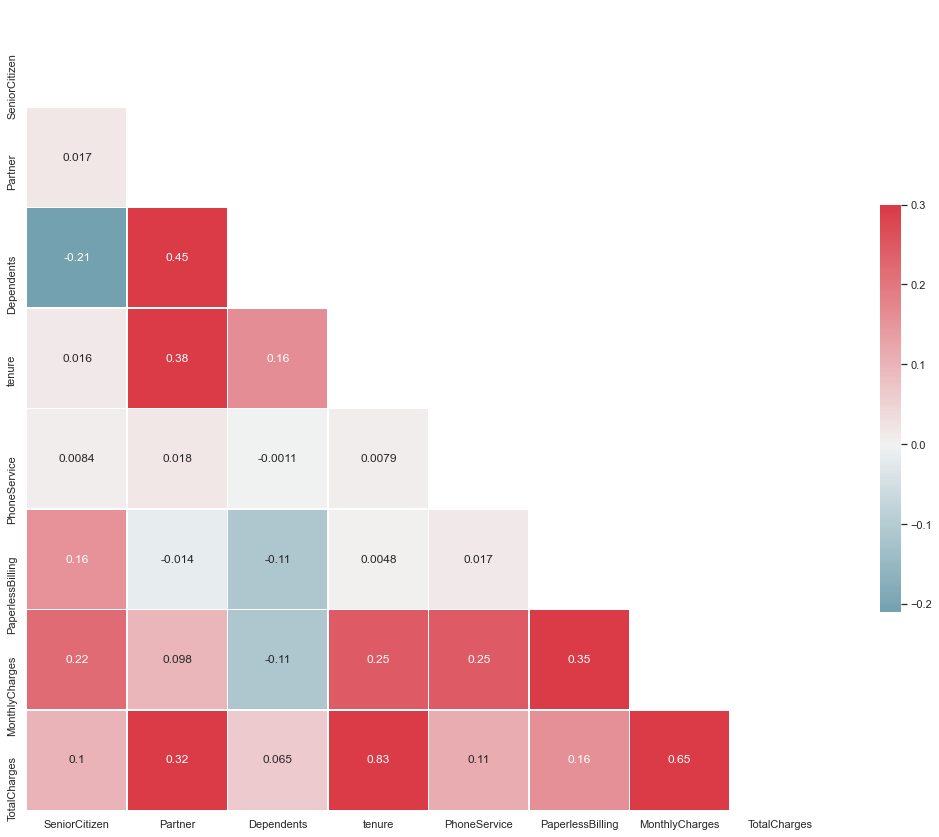


#Set and compute the Correlation Matrix:  
sns.set(style="white")  
plt.figure(figsize=(18, 15))  
  
corr = data.apply(lambda x: pd.factorize(x)[0]).corr()  
  
mask = np.triu(np.ones\_like(corr, dtype=bool))  
  
ax = sns.heatmap(corr, mask=mask, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, linewidths=.2, cmap='coolwarm', vmin=0.3, vmax=1)



#Set and compute the Correlation Matrix:  
sns.set(style="white")  
corr = data2.corr()  
  
#Generate a mask for the upper triangle:  
  
mask = np.zeros\_like(corr, dtype=np.bool)  
mask[np.triu\_indices\_from(mask)] = True  
  
#Set up the matplotlib figure and a diverging colormap:  
f, ax = plt.subplots(figsize=(18, 15))  
cmap = sns.diverging\_palette(220, 10, as\_cmap=True)  
  
#Draw the heatmap with the mask and correct aspect ratio:  
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,square=True,annot = True, linewidths=.5, cbar\_kws={"shrink": .5})

<AxesSubplot:>



vif\_data = pd.DataFrame() vif\_data["feature"] = X.columns

#calculating VIF for each feature vif\_data["VIF"] = [variance\_inflation\_factor(X.values, i) for i in range(len(X.columns))]

print(vif\_data)

def encode\_data(dataframe):  
 if dataframe.dtype == "object":  
 dataframe = LabelEncoder().fit\_transform(dataframe)  
 return dataframe  
  
data = data.apply(lambda x: encode\_data(x))  
data.head()

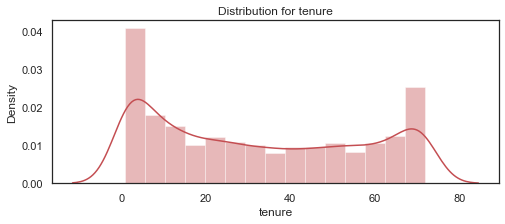
gender SeniorCitizen Partner Dependents tenure PhoneService \  
0 0 0 1 0 1 0   
1 1 0 0 0 34 1   
2 1 0 0 0 2 1   
3 1 0 0 0 45 0   
4 0 0 0 0 2 1   
  
 MultipleLines InternetService OnlineSecurity OnlineBackup \  
0 1 0 0 2   
1 0 0 2 0   
2 0 0 2 2   
3 1 0 2 0   
4 0 1 0 0   
  
 DeviceProtection TechSupport StreamingTV StreamingMovies Contract \  
0 0 0 0 0 0   
1 2 0 0 0 1   
2 0 0 0 0 0   
3 2 2 0 0 1   
4 0 0 0 0 0   
  
 PaperlessBilling PaymentMethod MonthlyCharges TotalCharges Churn   
0 1 2 29.85 29.85 0   
1 0 3 56.95 1889.50 0   
2 1 3 53.85 108.15 1   
3 0 0 42.30 1840.75 0   
4 1 2 70.70 151.65 1

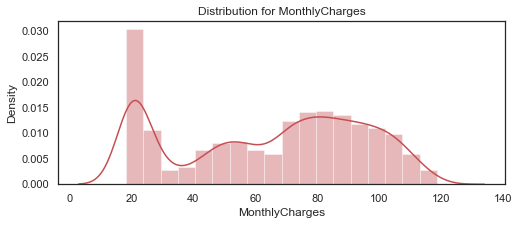
X = data.drop(columns = "Churn")  
y = data["Churn"].values

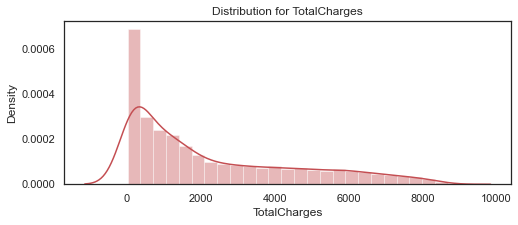
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 4, stratify =y)

def distplot(feature, frame, color='r'):  
 plt.figure(figsize=(8,3))  
 plt.title("Distribution for {}".format(feature))  
 ax = sns.distplot(frame[feature], color= color)

col = ["tenure", 'MonthlyCharges', 'TotalCharges']  
for features in col :distplot(features, data)

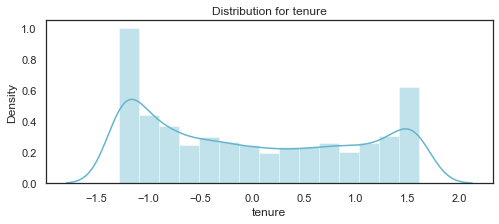


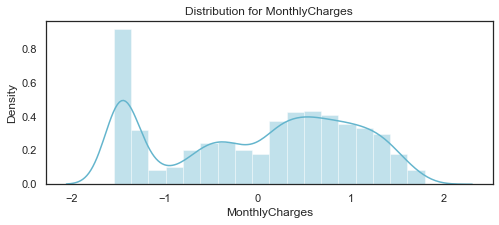


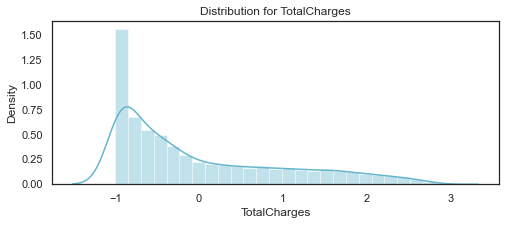


The features need standard scaling as all of them are distributed over different range values

data\_std = pd.DataFrame(StandardScaler().fit\_transform(data[col]).astype('float64'), columns = col)  
for feat in col: distplot(feat, data\_std, color='c')







data.columns

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

for i in data.columns:  
 print(i, ": ", data\_[i].unique())

gender : ['Female' 'Male']  
SeniorCitizen : [0 1]  
Partner : ['Yes' 'No']  
Dependents : ['No' 'Yes']  
tenure : [ 1 34 2 45 8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27  
 5 46 11 70 63 43 15 60 18 66 9 3 31 50 64 56 7 42 35 48 29 65 38 68  
 32 55 37 36 41 6 4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 0  
 39]  
PhoneService : ['No' 'Yes']  
MultipleLines : ['No phone service' 'No' 'Yes']  
InternetService : ['DSL' 'Fiber optic' 'No']  
OnlineSecurity : ['No' 'Yes' 'No internet service']  
OnlineBackup : ['Yes' 'No' 'No internet service']  
DeviceProtection : ['No' 'Yes' 'No internet service']  
TechSupport : ['No' 'Yes' 'No internet service']  
StreamingTV : ['No' 'Yes' 'No internet service']  
StreamingMovies : ['No' 'Yes' 'No internet service']  
Contract : ['Month-to-month' 'One year' 'Two year']  
PaperlessBilling : ['Yes' 'No']  
PaymentMethod : ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'  
 'Credit card (automatic)']  
MonthlyCharges : [29.85 56.95 53.85 ... 63.1 44.2 78.7 ]  
TotalCharges : ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']  
Churn : ['No' 'Yes']

# Divide the columns into 3 categories, one ofor standardisation, one for label encoding and one for one hot encoding  
  
cat\_cols\_ohe =['PaymentMethod', 'Contract', 'InternetService'] # those that need one-hot encoding  
cat\_cols\_le = list(set(X\_train.columns)- set(col) - set(cat\_cols\_ohe)) #those that need label encoding  
  
print(cat\_cols\_le)

['StreamingMovies', 'PaperlessBilling', 'SeniorCitizen', 'OnlineBackup', 'StreamingTV', 'MultipleLines', 'OnlineSecurity', 'Partner', 'PhoneService', 'gender', 'DeviceProtection', 'Dependents', 'TechSupport']

scaler = StandardScaler()  
X\_train[col] = StandardScaler().fit\_transform(X\_train[col])  
X\_test[col] = StandardScaler().fit\_transform(X\_test[col])

models = []  
  
models.append(('Logistic Regression', LogisticRegression(solver='liblinear', random\_state = 0, class\_weight='balanced')))  
models.append(('SVC', SVC(kernel = 'linear', random\_state = 0)))  
models.append(('Kernel SVM', SVC(kernel = 'rbf', random\_state = 0)))  
models.append(('KNN', KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)))  
models.append(('Gaussian NB', GaussianNB()))  
models.append(('Decision Tree Classifier', DecisionTreeClassifier(criterion = 'entropy', random\_state = 0)))  
models.append(('Random Forest', RandomForestClassifier(n\_estimators=100, criterion = 'entropy', random\_state = 0)))  
models.append(("Adaboost", AdaBoostClassifier()))  
models.append(("Gradient boost classifier", GradientBoostingClassifier()))  
models.append(("Voting Classifier", VotingClassifier(estimators=[('gbc', GradientBoostingClassifier()), ('lr', LogisticRegression()), ('abc', AdaBoostClassifier())], voting='soft')))

## Evaluating the model Results

acc\_results =[]  
auc\_results =[]  
names = []  
  
result\_col = ["Algorithm", "ROC AUC Mean", "ROC AUC STD", "Accuracy Mean", "Accuracy STD"]  
model\_results = pd.DataFrame(columns = result\_col)  
  
i=0  
# K- fold cross validation  
  
for name, model in models:  
 names.append(name)  
 kfold = model\_selection.KFold(n\_splits=10, random\_state=0)  
  
 cv\_acc\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train,  
 cv = kfold, scoring="accuracy")  
 cv\_auc\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train,  
 cv = kfold, scoring="roc\_auc")  
 acc\_results.append(cv\_acc\_results)  
 auc\_results.append(cv\_auc\_results)  
  
 model\_results.loc[i] = [name,  
 round(cv\_auc\_results.mean()\*100,2),  
 round(cv\_auc\_results.std()\*100,2),  
 round(cv\_acc\_results.mean()\*100,2),  
 round(cv\_acc\_results.std()\*100,2)]  
 i+=1  
  
model\_results.sort\_values(by = ['ROC AUC Mean'], ascending=False)

Algorithm ROC AUC Mean ROC AUC STD Accuracy Mean \  
9 Voting Classifier 84.93 1.39 80.23   
8 Gradient boost classifier 84.72 1.42 79.72   
7 Adaboost 84.55 1.25 80.09   
0 Logistic Regression 84.39 1.47 74.38   
1 SVC 82.99 2.07 79.11   
6 Random Forest 82.75 2.01 78.67   
4 Gaussian NB 82.32 1.28 75.38   
2 Kernel SVM 79.65 2.12 79.26   
3 KNN 77.14 1.43 75.90   
5 Decision Tree Classifier 66.67 1.07 73.73   
  
 Accuracy STD   
9 1.89   
8 1.95   
7 1.77   
0 1.94   
1 2.01   
6 1.98   
4 1.23   
2 1.67   
3 2.01   
5 1.12

fig = plt.figure(figsize=(25,15))  
ax = fig.add\_subplot(111)  
plt.boxplot(acc\_results)  
ax.set\_xticklabels(names)  
  
plt.ylabel('ROC AUC Score\n',  
horizontalalignment="center",fontstyle = "normal",  
fontsize = "large", fontfamily = "sans-serif")  
  
plt.xlabel('\n Baseline Classification Algorithms\n',  
horizontalalignment="center",fontstyle = "normal",  
fontsize = "large", fontfamily = "sans-serif")  
  
plt.title('Accuracy Score Comparison \n',  
horizontalalignment="center", fontstyle = "normal",  
fontsize = "22", fontfamily = "sans-serif")  
  
plt.xticks(rotation=0, horizontalalignment="center")  
plt.yticks(rotation=0, horizontalalignment="right")  
plt.show()

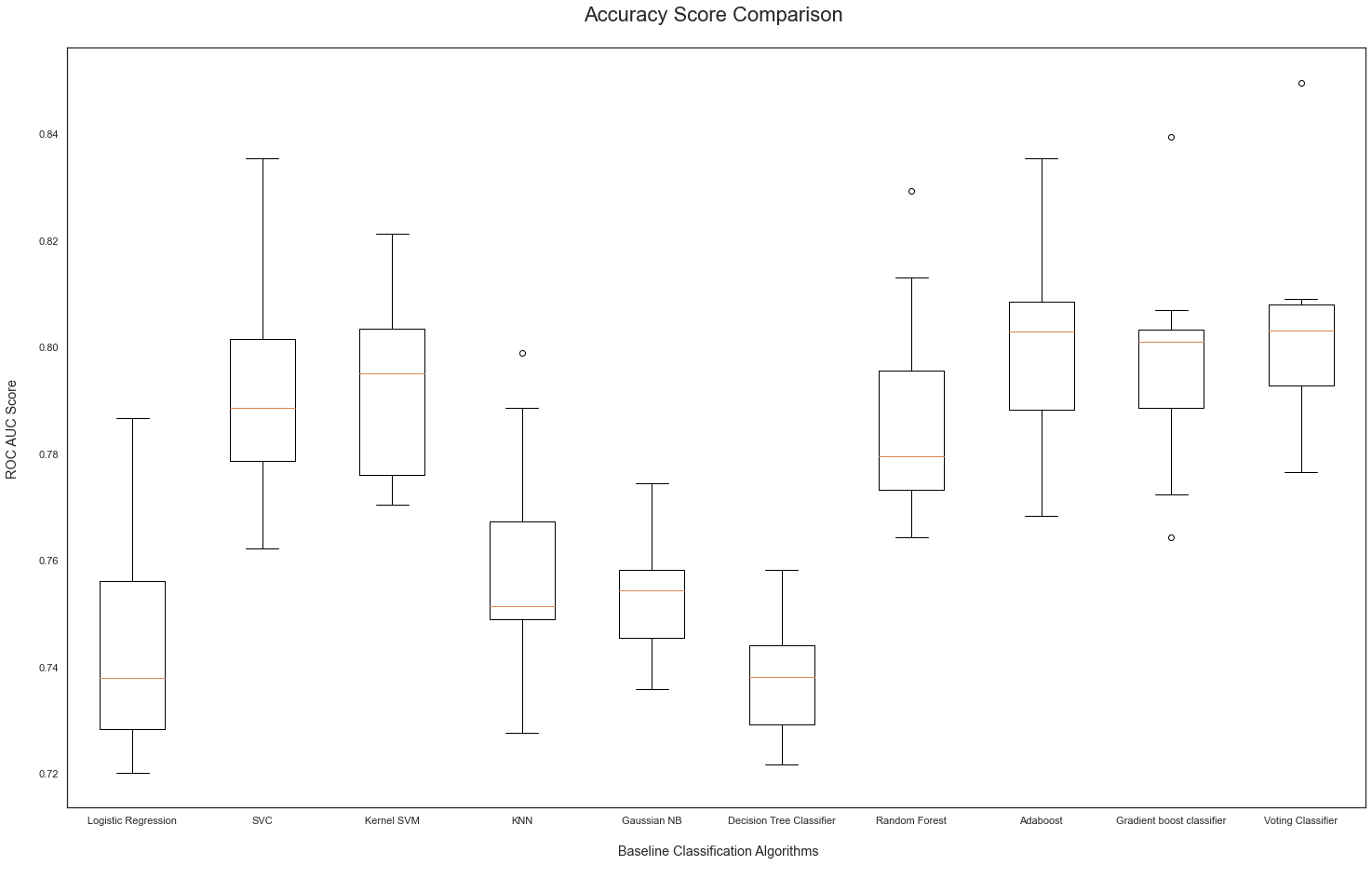
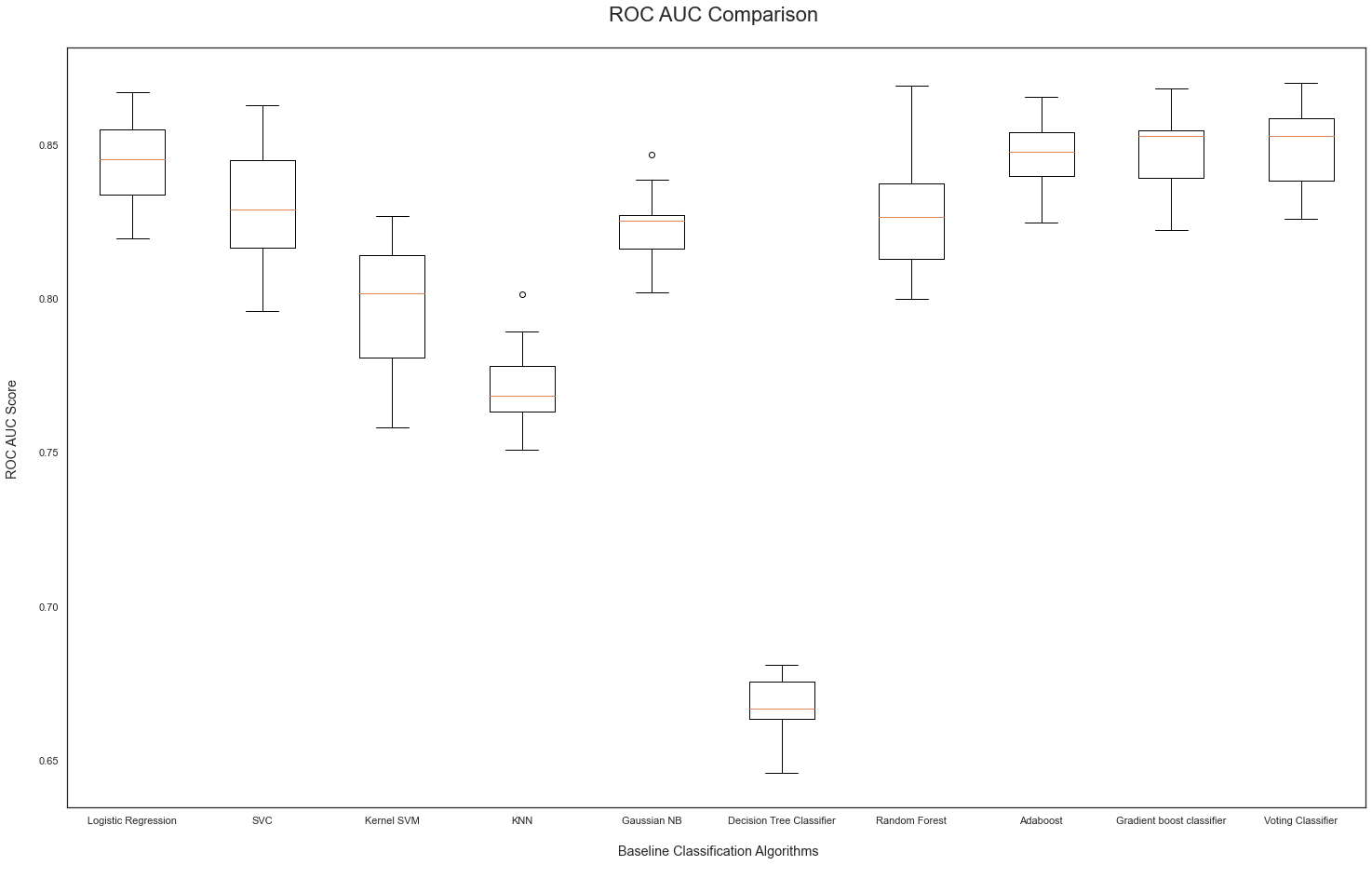


fig = plt.figure(figsize=(25,15))  
ax = fig.add\_subplot(111)  
plt.boxplot(auc\_results)  
ax.set\_xticklabels(names)  
  
plt.ylabel('ROC AUC Score\n',  
horizontalalignment="center",fontstyle = "normal",  
fontsize = "large", fontfamily = "sans-serif")  
  
plt.xlabel('\n Baseline Classification Algorithms\n',  
horizontalalignment="center",fontstyle = "normal",  
fontsize = "large", fontfamily = "sans-serif")  
  
plt.title('ROC AUC Comparison \n',  
horizontalalignment="center", fontstyle = "normal",  
fontsize = "22", fontfamily = "sans-serif")  
  
plt.xticks(rotation=0, horizontalalignment="center")  
plt.yticks(rotation=0, horizontalalignment="right")  
plt.show()

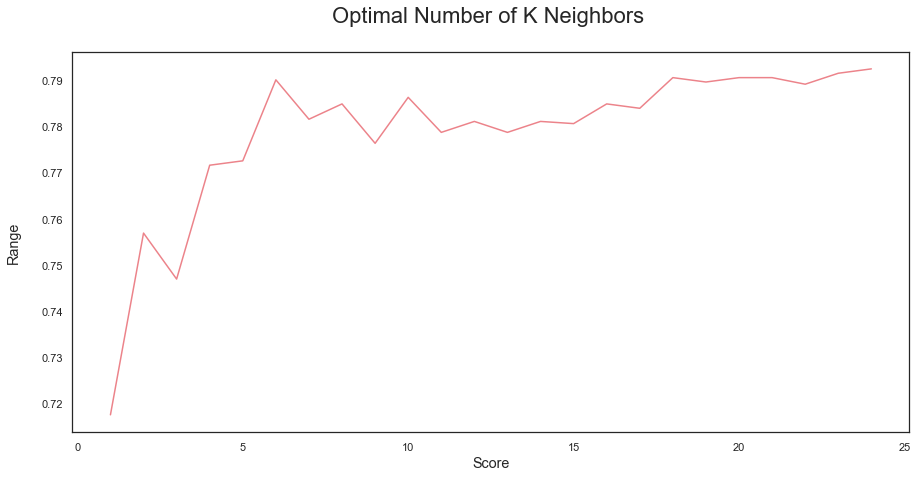


score\_array = []  
  
for each in range(1,25):  
 knn\_loop = KNeighborsClassifier(n\_neighbors = each)  
 knn\_loop.fit(X\_train,y\_train)  
 score\_array.append(knn\_loop.score(X\_test,y\_test))  
  
score\_array

[0.7175355450236967,  
 0.7568720379146919,  
 0.7469194312796209,  
 0.771563981042654,  
 0.7725118483412322,  
 0.7900473933649289,  
 0.7815165876777251,  
 0.7848341232227488,  
 0.776303317535545,  
 0.7862559241706161,  
 0.7786729857819905,  
 0.781042654028436,  
 0.7786729857819905,  
 0.781042654028436,  
 0.7805687203791469,  
 0.7848341232227488,  
 0.7838862559241706,  
 0.790521327014218,  
 0.7895734597156399,  
 0.790521327014218,  
 0.790521327014218,  
 0.7890995260663507,  
 0.7914691943127962,  
 0.7924170616113744]

## KNN

fig = plt.figure(figsize=(15, 7))  
plt.plot(range(1,25),score\_array, color = '#ec838a')  
plt.ylabel('Range\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large", fontfamily = "sans-serif")  
plt.xlabel('Score\n',horizontalalignment="center",fontstyle = "normal", fontsize = "large", fontfamily = "sans-serif")  
  
plt.title('Optimal Number of K Neighbors \n',horizontalalignment="center", fontstyle = "normal",fontsize = "22", fontfamily = "sans-serif")  
#plt.legend(loc='top right', fontsize = "medium")  
  
plt.xticks(rotation=0, horizontalalignment="center")  
plt.yticks(rotation=0, horizontalalignment="right")  
plt.show()



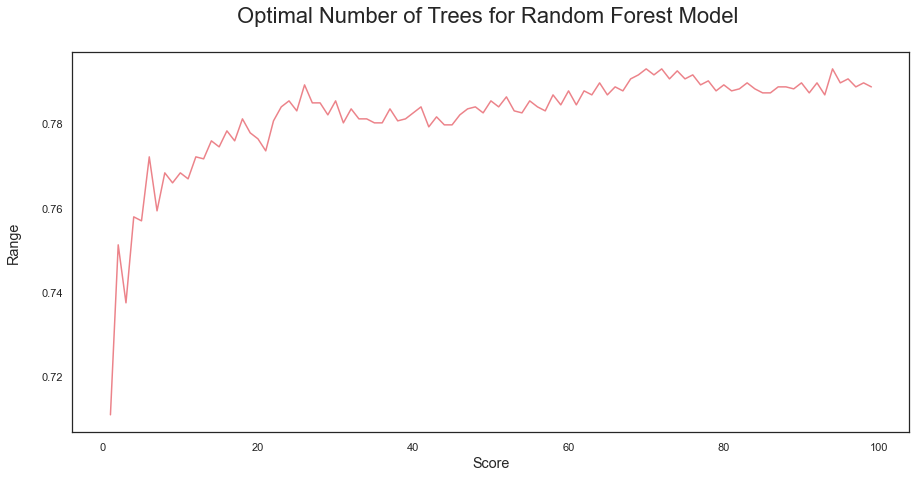
## Random Forest

score\_array = []  
for each in range(1,100):  
 rf\_loop = RandomForestClassifier(n\_estimators = each, random\_state = 1)  
 rf\_loop.fit(X\_train,y\_train)  
 score\_array.append(rf\_loop.score(X\_test,y\_test))

for i,j in enumerate(score\_array):  
 print(i+1,":",j)

1 : 0.7109004739336493  
2 : 0.7511848341232228  
3 : 0.7374407582938388  
4 : 0.7578199052132701  
5 : 0.7568720379146919  
6 : 0.7720379146919432  
7 : 0.7592417061611374  
8 : 0.7682464454976303  
9 : 0.7658767772511849  
10 : 0.7682464454976303  
11 : 0.7668246445497631  
12 : 0.7720379146919432  
13 : 0.771563981042654  
14 : 0.7758293838862559  
15 : 0.7744075829383886  
16 : 0.7781990521327015  
17 : 0.7758293838862559  
18 : 0.781042654028436  
19 : 0.7777251184834123  
20 : 0.776303317535545  
21 : 0.7734597156398104  
22 : 0.7805687203791469  
23 : 0.7838862559241706  
24 : 0.7853080568720379  
25 : 0.7829383886255924  
26 : 0.7890995260663507  
27 : 0.7848341232227488  
28 : 0.7848341232227488  
29 : 0.7819905213270142  
30 : 0.7853080568720379  
31 : 0.7800947867298578  
32 : 0.7834123222748816  
33 : 0.781042654028436  
34 : 0.781042654028436  
35 : 0.7800947867298578  
36 : 0.7800947867298578  
37 : 0.7834123222748816  
38 : 0.7805687203791469  
39 : 0.781042654028436  
40 : 0.7824644549763033  
41 : 0.7838862559241706  
42 : 0.7791469194312797  
43 : 0.7815165876777251  
44 : 0.7796208530805687  
45 : 0.7796208530805687  
46 : 0.7819905213270142  
47 : 0.7834123222748816  
48 : 0.7838862559241706  
49 : 0.7824644549763033  
50 : 0.7853080568720379  
51 : 0.7838862559241706  
52 : 0.7862559241706161  
53 : 0.7829383886255924  
54 : 0.7824644549763033  
55 : 0.7853080568720379  
56 : 0.7838862559241706  
57 : 0.7829383886255924  
58 : 0.7867298578199052  
59 : 0.7843601895734598  
60 : 0.7876777251184834  
61 : 0.7843601895734598  
62 : 0.7876777251184834  
63 : 0.7867298578199052  
64 : 0.7895734597156399  
65 : 0.7867298578199052  
66 : 0.7886255924170616  
67 : 0.7876777251184834  
68 : 0.790521327014218  
69 : 0.7914691943127962  
70 : 0.7928909952606635  
71 : 0.7914691943127962  
72 : 0.7928909952606635  
73 : 0.790521327014218  
74 : 0.7924170616113744  
75 : 0.790521327014218  
76 : 0.7914691943127962  
77 : 0.7890995260663507  
78 : 0.7900473933649289  
79 : 0.7876777251184834  
80 : 0.7890995260663507  
81 : 0.7876777251184834  
82 : 0.7881516587677725  
83 : 0.7895734597156399  
84 : 0.7881516587677725  
85 : 0.7872037914691943  
86 : 0.7872037914691943  
87 : 0.7886255924170616  
88 : 0.7886255924170616  
89 : 0.7881516587677725  
90 : 0.7895734597156399  
91 : 0.7872037914691943  
92 : 0.7895734597156399  
93 : 0.7867298578199052  
94 : 0.7928909952606635  
95 : 0.7895734597156399  
96 : 0.790521327014218  
97 : 0.7886255924170616  
98 : 0.7895734597156399  
99 : 0.7886255924170616

fig = plt.figure(figsize=(15, 7))  
plt.plot(range(1,100),score\_array, color = '#ec838a')  
plt.ylabel('Range\n',horizontalalignment="center",  
fontstyle = "normal", fontsize = "large",  
fontfamily = "sans-serif")  
plt.xlabel('Score\n',horizontalalignment="center",  
fontstyle = "normal", fontsize = "large",  
fontfamily = "sans-serif")  
plt.title('Optimal Number of Trees for Random Forest Model \n',horizontalalignment="center", fontstyle = "normal", fontsize = "22", fontfamily = "sans-serif")  
#plt.legend(loc='top right', fontsize = "medium")  
plt.xticks(rotation=0, horizontalalignment="center")  
plt.yticks(rotation=0, horizontalalignment="right")  
plt.show()



## 2nd Iteration

#evaluation of results  
def model\_evaluation(y\_test, y\_pred, model\_name):  
 acc = accuracy\_score(y\_test, y\_pred)  
 prec = precision\_score(y\_test, y\_pred)  
 rec = recall\_score(y\_test, y\_pred)  
 f1 = f1\_score(y\_test, y\_pred)  
 f2 = fbeta\_score(y\_test, y\_pred, beta = 2.0)  
  
 results = pd.DataFrame([[model\_name, acc, prec, rec, f1, f2]],  
 columns = ["Model", "Accuracy", "Precision", "Recall",  
 "F1 SCore", "F2 Score"])  
 results = results.sort\_values(["Precision", "Recall", "F2 Score"], ascending = False)  
 return results

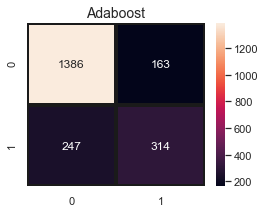
# Logistic regression  
classifier = LogisticRegression(random\_state=0)  
classifier.fit(X\_train, y\_train)  
  
y\_pred = classifier.predict(X\_test)  
  
#SVC  
  
classifier2 = SVC(kernel = 'linear', random\_state = 0)  
classifier2.fit(X\_train, y\_train)  
y\_pred2 = classifier2.predict(X\_test)  
  
#knn  
  
classifier3 = KNeighborsClassifier(n\_neighbors=22, metric="minkowski", p=2)  
classifier3.fit(X\_train, y\_train)  
y\_pred3 = classifier3.predict(X\_test)  
  
  
#Kernel SVM  
classifier4 = SVC(kernel="rbf", random\_state =0)  
classifier4.fit(X\_train, y\_train)  
y\_pred4 = classifier4.predict(X\_test)  
  
  
#Naive Bayes  
classifier5 = GaussianNB()  
classifier5.fit(X\_train, y\_train)  
y\_pred5 = classifier5.predict(X\_test)  
  
#Decision tree  
classifier6 = DecisionTreeClassifier(criterion="entropy", random\_state=0)  
classifier6.fit(X\_train, y\_train)  
y\_pred6 = classifier6.predict(X\_test)  
  
#Random Forest  
  
classifier7 = RandomForestClassifier(n\_estimators=72, criterion="entropy", random\_state=0)  
classifier7.fit(X\_train, y\_train)  
y\_pred7 = classifier7.predict(X\_test)  
  
#Adaboost  
classifier8 = AdaBoostClassifier()  
classifier8.fit(X\_train, y\_train)  
y\_pred8 = classifier8.predict(X\_test)  
  
#Gradient Boost  
classifier9 = GradientBoostingClassifier()  
  
  
classifier9.fit(X\_train, y\_train)  
y\_pred9 = classifier9.predict(X\_test)  
  
  
  
  
  
#Voting Classifier  
  
classifier10 = VotingClassifier(estimators=[('gbc', GradientBoostingClassifier()), ('lr', LogisticRegression()),  
 ('abc', AdaBoostClassifier())], voting='soft')  
  
  
  
classifier10.fit(X\_train, y\_train)  
y\_pred10 = classifier10.predict(X\_test)

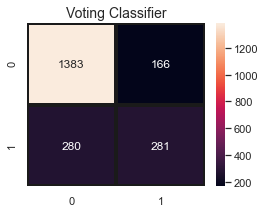
lr = model\_evaluation(y\_test, y\_pred, "Logistic Regression")  
svm = model\_evaluation(y\_test, y\_pred2, "SVM (Linear)")  
knn = model\_evaluation(y\_test, y\_pred3, "K-Nearest Neighbours")  
k\_svm = model\_evaluation(y\_test, y\_pred4, "Kernel SVM")  
nb = model\_evaluation(y\_test, y\_pred5, "Naive Bayes")  
dt = model\_evaluation(y\_test, y\_pred6, "Decision Tree")  
rf = model\_evaluation(y\_test, y\_pred7, "Random Forest")  
ab = model\_evaluation(y\_test, y\_pred8, "Adaboost")  
gb = model\_evaluation(y\_test, y\_pred9, "Gradient Boost")  
vc = model\_evaluation(y\_test, y\_pred10, "Voting Classifier")

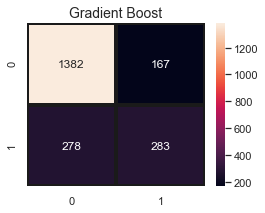
eval\_ =lr.append(svm).append(knn).append(k\_svm).append(nb).append(dt).append(rf).append(ab).append(gb).append(vc).sort\_values(["Precision",  
"Recall", "F2 Score"], ascending = False).reset\_index().drop(columns = "index")  
eval\_

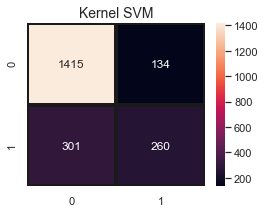
Model Accuracy Precision Recall F1 SCore F2 Score  
0 Adaboost 0.812322 0.687927 0.538324 0.604000 0.562803  
1 Voting Classifier 0.808531 0.675615 0.538324 0.599206 0.561130  
2 Gradient Boost 0.805213 0.672018 0.522282 0.587763 0.546642  
3 Kernel SVM 0.793839 0.659898 0.463458 0.544503 0.492798  
4 Logistic Regression 0.805687 0.658281 0.559715 0.605010 0.576994  
5 Random Forest 0.796682 0.650685 0.508021 0.570571 0.531320  
6 K-Nearest Neighbours 0.789100 0.628889 0.504456 0.559842 0.525241  
7 SVM (Linear) 0.788626 0.628635 0.500891 0.557540 0.522111  
8 Naive Bayes 0.756872 0.531088 0.730838 0.615154 0.679708  
9 Decision Tree 0.732701 0.497364 0.504456 0.500885 0.503022

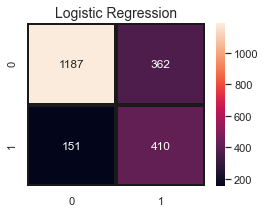
predictions = [y\_pred, y\_pred2 , y\_pred3, y\_pred4, y\_pred5, y\_pred5, y\_pred6, y\_pred7,  
 y\_pred8, y\_pred9, y\_pred10]  
  
for i, j in zip(predictions, eval\_.Model.values):  
 plt.figure(figsize=(4,3))  
 sns.heatmap(confusion\_matrix(y\_test, i),  
 annot=True,fmt = "d",linecolor="k",linewidths=3)  
  
 plt.title(j,fontsize=14)  
 plt.show()

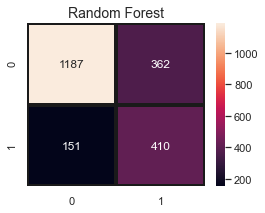


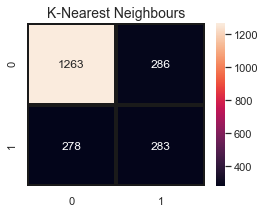


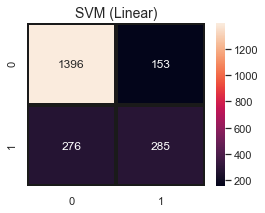


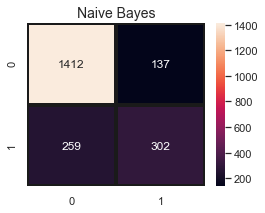


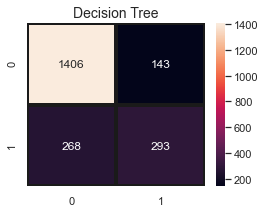












k-Fold Cross-Validation: Model evaluation is most commonly done through ‘K- fold Cross-Validation’ technique that primarily helps us to fix the variance. Variance problem occurs when we get good accuracy while running the model on a training set and a test set but then the accuracy looks different when the model is run on another test set. So, in order to fix the variance problem, k-fold cross-validation basically split the training set into 10 folds and train the model on 9 folds (9 subsets of the training dataset) before testing it on the test fold. This gives us the flexibility to train our model on all ten combinations of 9 folds; giving ample room to finalize the variance.

#TODO: Model Evaluation

def k\_fold\_cross\_validation(classifier\_name, name):  
 accuracies = cross\_val\_score(estimator=classifier\_name,  
 X=X\_train, y=y\_train, cv =10)  
 print(name, "accuracy: %0.2f (+/- %0.2f)" % (accuracies.mean(), accuracies.std() \* 2))

k\_fold\_cross\_validation(classifier8, "Adaboost")

Adaboost accuracy: 0.80 (+/- 0.03)

k\_fold\_cross\_validation(classifier10, "Voting classifier")

Voting classifier accuracy: 0.80 (+/- 0.04)

k\_fold\_cross\_validation(classifier9, "Gradient Boost classifier")

Gradient Boost classifier accuracy: 0.80 (+/- 0.04)

k\_fold\_cross\_validation(classifier, "Logistic regression")

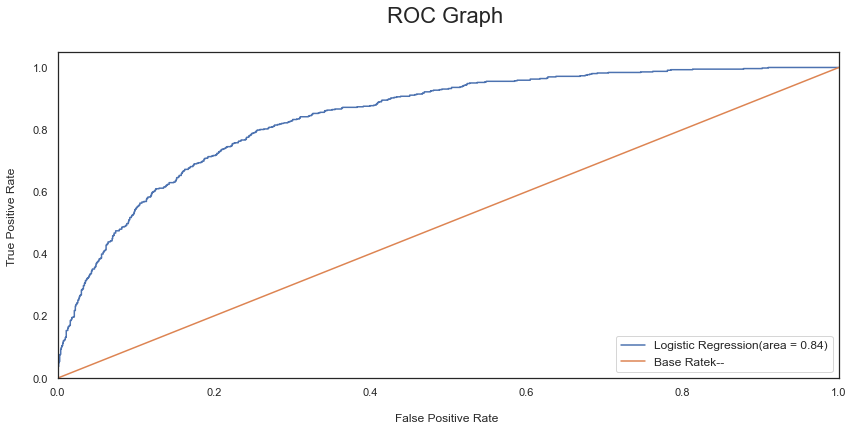
Logistic regression accuracy: 0.80 (+/- 0.04)

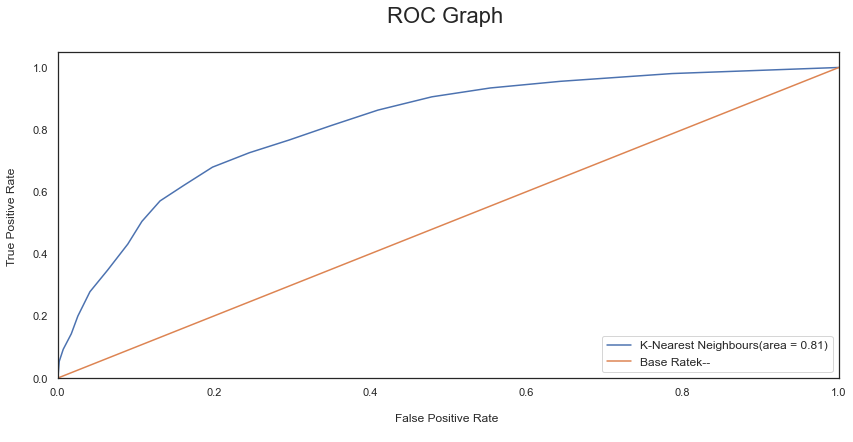
k\_fold\_cross\_validation(classifier4, "Kernel SVM")

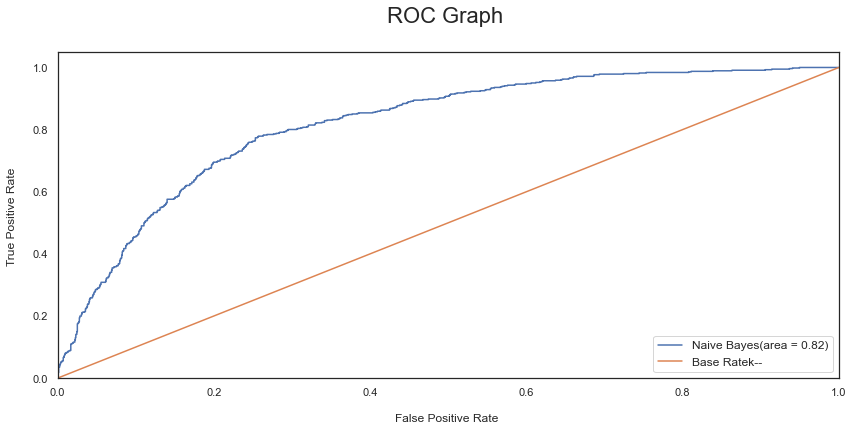
Kernel SVM accuracy: 0.80 (+/- 0.03)

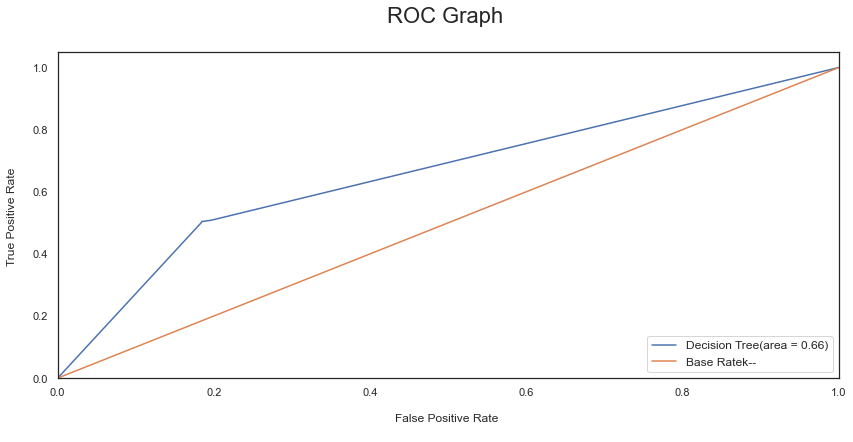
# ROC Curve  
  
def ROC\_curve(classifier\_, name, y\_pred\_):  
 classifier\_.fit(X\_train, y\_train)  
 probs = classifier\_.predict\_proba(X\_test)  
 probs = probs[:, 1]  
 classifier\_roc\_auc = roc\_auc\_score(y\_test, probs )  
 rf\_fpr, rf\_tpr, rf\_thresholds = roc\_curve(y\_test, classifier\_.predict\_proba(X\_test)[:,1])  
 plt.figure(figsize=(14, 6))  
  
 label\_ = name + '(area = %0.2f)' % classifier\_roc\_auc  
 # Plot Adaboost ROC  
 plt.plot(rf\_fpr, rf\_tpr,  
 label=label\_)  
 # Plot Base Rate ROC  
 plt.plot([0,1], [0,1],label='Base Rate' 'k--')  
 plt.xlim([0.0, 1.0])  
 plt.ylim([0.0, 1.05])  
 plt.ylabel('True Positive Rate \n',horizontalalignment="center",  
 fontstyle = "normal", fontsize = "medium",  
 fontfamily = "sans-serif")  
  
 plt.xlabel('\nFalse Positive Rate \n',horizontalalignment="center",  
 fontstyle = "normal", fontsize = "medium",  
 fontfamily = "sans-serif")  
  
 plt.title('ROC Graph \n',horizontalalignment="center",  
 fontstyle = "normal", fontsize = "22",  
 fontfamily = "sans-serif")  
  
 plt.legend(loc="lower right", fontsize = "medium")  
 plt.xticks(rotation=0, horizontalalignment="center")  
 plt.yticks(rotation=0, horizontalalignment="right")  
 plt.show()

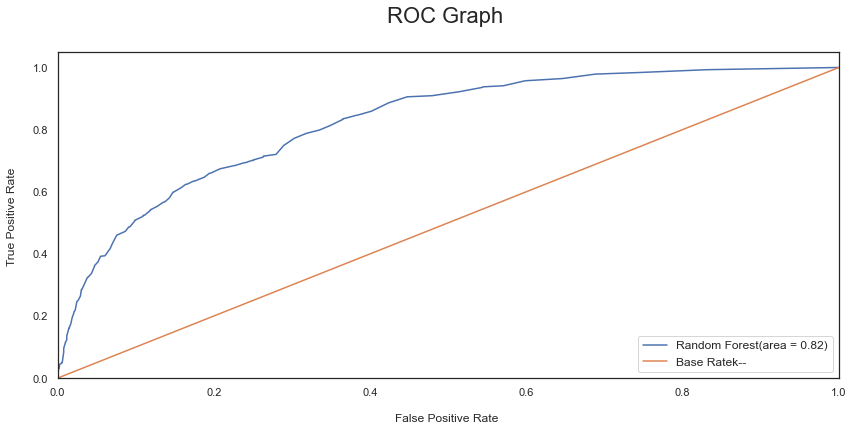
preds = [y\_pred, y\_pred3, y\_pred5, y\_pred6, y\_pred7,  
 y\_pred8, y\_pred9, y\_pred10]  
classifiers = [classifier , classifier3, classifier5, classifier6, classifier7,  
 classifier8, classifier9, classifier10]  
model\_names\_ = ["Logistic Regression", "K-Nearest Neighbours","Naive Bayes",  
 "Decision Tree", "Random Forest", "Adaboost", "Gradient Boost", "Voting Classifier"]  
  
for i, j, k in zip(classifiers, model\_names\_, predictions):  
 ROC\_curve(i, j, k)

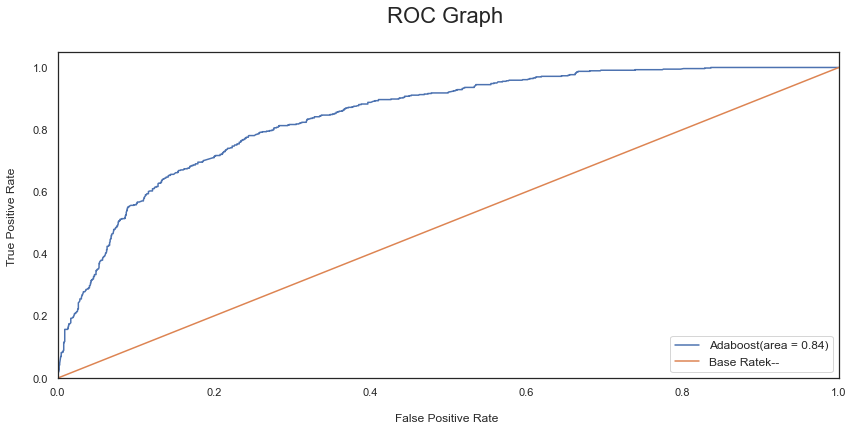


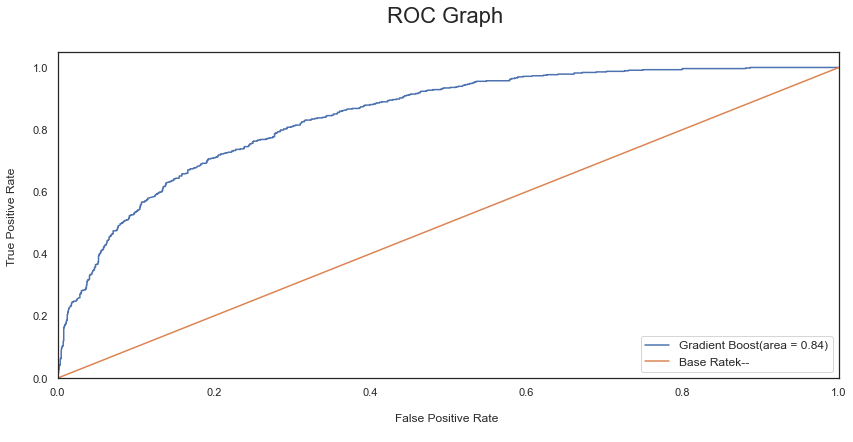


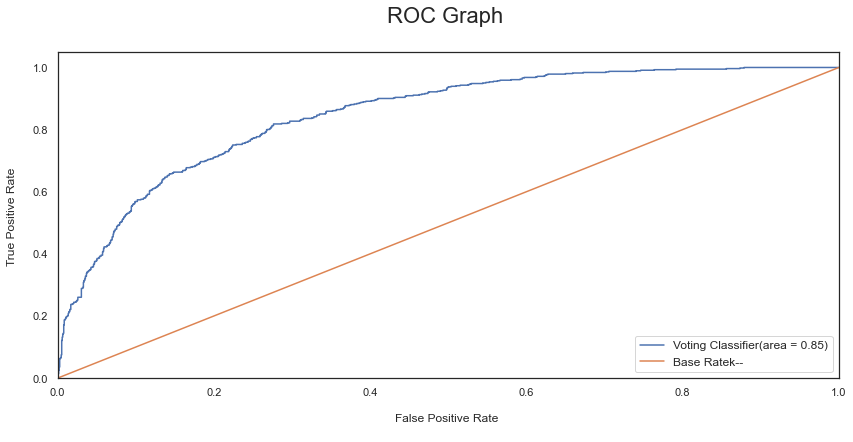












# Cross validation  
  
from sklearn.model\_selection import cross\_val\_score  
  
# Function that will track the mean value and the standard deviation of the accuracy  
def cvDictGen(functions, scr, X\_train = X, y\_train = y, cv = 5):  
 cvDict = {}  
 for func in functions:  
 cvScore = cross\_val\_score(func, X\_train, y\_train, cv = cv, scoring = scr)  
 cvDict[str(func).split('(')[0]] = [cvScore.mean(), cvScore.std()]  
  
 return cvDict

cvD = cvDictGen(classifiers, scr = 'roc\_auc')  
cvD

{'LogisticRegression': [0.841331397558646, 0.010495252078550477],  
 'KNeighborsClassifier': [0.7913242024807321, 0.008198993337848612],  
 'GaussianNB': [0.8232386881685605, 0.00741678015498337],  
 'DecisionTreeClassifier': [0.6470213137060805, 0.02196953973039052],  
 'RandomForestClassifier': [0.8197874155380965, 0.011556155864106703],  
 'AdaBoostClassifier': [0.8445838813774079, 0.01125665302188384],  
 'GradientBoostingClassifier': [0.844630629931458, 0.010723107447558198],  
 'VotingClassifier': [0.8468096379573085, 0.010887508320460332]}

## Predicting feature importance

# Gradient Boost  
feature\_importances = pd.concat([pd.DataFrame(data.columns, columns = ["features"]),  
 pd.DataFrame(np.transpose(classifier9.feature\_importances\_), columns = ["coef"])],axis = 1)  
feature\_importances.sort\_values(by = "coef", ascending = False)

features coef  
14 Contract 0.403837  
4 tenure 0.142425  
17 MonthlyCharges 0.134165  
18 TotalCharges 0.108308  
8 OnlineSecurity 0.064292  
11 TechSupport 0.055295  
7 InternetService 0.024634  
16 PaymentMethod 0.012317  
1 SeniorCitizen 0.011853  
15 PaperlessBilling 0.009874  
9 OnlineBackup 0.008643  
6 MultipleLines 0.006893  
10 DeviceProtection 0.004704  
13 StreamingMovies 0.003343  
2 Partner 0.002785  
0 gender 0.002249  
5 PhoneService 0.001479  
3 Dependents 0.001476  
12 StreamingTV 0.001428  
19 Churn NaN

# Ada boost classifier  
feature\_importances = pd.concat([pd.DataFrame(data.columns, columns = ["features"]),  
 pd.DataFrame(np.transpose(classifier8.feature\_importances\_), columns = ["coef"])],axis = 1)  
feature\_importances.sort\_values(by = "coef", ascending = False)

features coef  
18 TotalCharges 0.34  
17 MonthlyCharges 0.20  
4 tenure 0.14  
14 Contract 0.12  
16 PaymentMethod 0.04  
8 OnlineSecurity 0.04  
10 DeviceProtection 0.02  
15 PaperlessBilling 0.02  
11 TechSupport 0.02  
9 OnlineBackup 0.02  
1 SeniorCitizen 0.02  
6 MultipleLines 0.02  
7 InternetService 0.00  
12 StreamingTV 0.00  
13 StreamingMovies 0.00  
5 PhoneService 0.00  
3 Dependents 0.00  
2 Partner 0.00  
0 gender 0.00  
19 Churn NaN

## Hyper Parameter tuning

## Using Randomized search CV

#Ada boost  
  
from sklearn.model\_selection import RandomizedSearchCV  
from scipy.stats import randint  
adaHyperParams = {'n\_estimators': [10,50,100,200,420], "learning\_rate": [0.001, 0.01, 0.1, 0.3]}  
gridSearchAda = RandomizedSearchCV(estimator = classifier8, param\_distributions = adaHyperParams, n\_iter = 5,  
 scoring = 'roc\_auc') # other option accuracy  
gridSearchAda.fit(X\_train, y\_train)

RandomizedSearchCV(estimator=AdaBoostClassifier(), n\_iter=5,  
 param\_distributions={'learning\_rate': [0.001, 0.01, 0.1,  
 0.3],  
 'n\_estimators': [10, 50, 100, 200,  
 420]},  
 scoring='roc\_auc')

gridSearchAda.best\_params\_, gridSearchAda.best\_score\_

({'n\_estimators': 420, 'learning\_rate': 0.1}, 0.8450951703673771)

bestAdaModFitted = gridSearchAda.best\_estimator\_.fit(X\_train, y\_train)

# Getting the score AdaBoost  
test\_labels = bestAdaModFitted.predict\_proba(np.array(X\_test.values))[:,1]  
roc\_auc\_score(y\_test,test\_labels , average = 'macro', sample\_weight = None)

0.8342096390172947

### Gradient Boost

gbHyperParams = {'loss' : ['deviance', 'exponential'],  
 'n\_estimators': randint(10, 500),  
 'max\_depth': randint(1,10)}  
# Initialization  
gridSearchGB = RandomizedSearchCV(estimator = classifier9, param\_distributions = gbHyperParams, n\_iter = 10,  
 scoring = 'roc\_auc')  
# Fitting the model  
gridSearchGB.fit(X\_train, y\_train)

RandomizedSearchCV(estimator=GradientBoostingClassifier(),  
 param\_distributions={'loss': ['deviance', 'exponential'],  
 'max\_depth': <scipy.stats.\_distn\_infrastructure.rv\_frozen object at 0x000001EF9D363580>,  
 'n\_estimators': <scipy.stats.\_distn\_infrastructure.rv\_frozen object at 0x000001EFA8D079A0>},  
 scoring='roc\_auc')

gridSearchGB.best\_params\_, gridSearchGB.best\_score\_

({'loss': 'deviance', 'max\_depth': 2, 'n\_estimators': 218}, 0.8439536584504147)

bestGBModFitted = gridSearchGB.best\_estimator\_.fit(X\_train, y\_train)

# Getting the score AdaBoost  
test\_labels\_GB = bestGBModFitted.predict\_proba(np.array(X\_test.values))[:,1]  
roc\_auc\_score(y\_test,test\_labels\_GB , average = 'macro', sample\_weight = None)

0.8372850519396677

## Using Grid SearchCV

ABC = AdaBoostClassifier()  
  
ABC\_param\_grid = {"n\_estimators" :[10,50,100,200,420],  
 "learning\_rate": [0.001, 0.01, 0.1, 0.3]}  
  
gsABC = GridSearchCV(ABC, param\_grid = ABC\_param\_grid, cv = 10, scoring = "roc\_auc", n\_jobs = 6, verbose = 1)  
  
gsABC.fit(X\_train,y\_train)  
  
ada\_best = gsABC.best\_estimator\_  
print(ada\_best)  
print(gsABC.best\_score\_)

Fitting 10 folds for each of 20 candidates, totalling 200 fits

[Parallel(n\_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.  
[Parallel(n\_jobs=6)]: Done 38 tasks | elapsed: 8.3s  
[Parallel(n\_jobs=6)]: Done 188 tasks | elapsed: 38.9s  
[Parallel(n\_jobs=6)]: Done 200 out of 200 | elapsed: 45.0s finished

AdaBoostClassifier(learning\_rate=0.1, n\_estimators=200)  
0.8470662551660079

bestAdaModFitted2 = gsABC.best\_estimator\_.fit(X\_train, y\_train)

test\_labels = bestAdaModFitted2.predict\_proba(np.array(X\_test.values))[:,1]  
roc\_auc\_score(y\_test,test\_labels , average = 'macro', sample\_weight = None)

0.8451764061455324

### Gradient Boost

gb\_param\_grid = {'loss' : ['deviance'],  
 'n\_estimators': [10,100,200,300],  
 'max\_depth': [1,2,4,6,8]}  
  
gsGB = GridSearchCV(classifier9, param\_grid = gb\_param\_grid, cv = 10, scoring = "roc\_auc", n\_jobs = 6, verbose = 1)  
  
gsGB.fit(X\_train,y\_train)  
  
gb\_best = gsGB.best\_estimator\_  
print(gb\_best)  
print(gsGB.best\_score\_)

Fitting 10 folds for each of 20 candidates, totalling 200 fits

[Parallel(n\_jobs=6)]: Using backend LokyBackend with 6 concurrent workers.  
[Parallel(n\_jobs=6)]: Done 38 tasks | elapsed: 7.3s  
[Parallel(n\_jobs=6)]: Done 188 tasks | elapsed: 1.0min  
[Parallel(n\_jobs=6)]: Done 200 out of 200 | elapsed: 1.3min finished

GradientBoostingClassifier(max\_depth=1, n\_estimators=300)  
0.8481591522663035

bestGBModFitted2 = gsGB.best\_estimator\_.fit(X\_train, y\_train)  
  
test\_labels\_gb2 = bestGBModFitted2.predict\_proba(np.array(X\_test.values))[:,1]  
roc\_auc\_score(y\_test,test\_labels\_gb2 , average = 'macro', sample\_weight = None)

0.8403339973233261