```
#Standard libraries for data analysis:

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy.stats import norm, skew
from scipy import stats
import statsmodels.api as sm
# sklearn modules for data preprocessing:
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
#sklearn modules for Model Selection:
from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble, discriminant_analysis, gaussian_process
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
#sklearn modules for Model Evaluation & Improvement:

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:
import seaborn as sn
from matplotlib import pyplot
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import matplotlib
%matplotlib inline
color = sn.color_palette()
import matplotlib.ticker as mtick
from IPython.display import display
pd.options.display.max_columns = None
```

```
pd.options.display.max_columns = None
from pandas.plotting import scatter_matrix
from sklearn.metrics import roc_curve
#Miscellaneous Utilitiy Libraries:

import random
import os
import re
import sys
import timeit
import string
import time
from datetime import datetime
from time import time
from dateutil.parser import parse
import joblib
```

```
/usr/local/lib/python3.7/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnir
  import pandas.util.testing as tm
```

```
from google.colab import drive
drive.mount('/content/drive')
dataset = pd.read_excel('/content/drive/My Drive/dataSet TCC V2.xlsx')
dataset.info()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202436 entries, 0 to 202435
Data columns (total 16 columns):
 #   Column                Non-Null Count   Dtype         
---  ------                --------------   -----         
 0   CDCLIENTE             202436 non-null  int64         
 1   NOME                  202436 non-null  object        
 2   NMCLIENTE             202031 non-null  object        
 3   UF                    202031 non-null  object        
 4   NMMODALIDADE          202436 non-null  object        
 5   NMSEGMENTO            202436 non-null  object        
 6   NMMERCADO             202436 non-null  object        
 7   PRIMEIRAAQUISICAO     202383 non-null  object        
 8   CANCELAMENTO          94036 non-null   datetime64[ns]
 9   NMMOTIVOCANCELAMENTO  93636 non-null   object        
 10  TEMPOANOS             202383 non-null  float64       
 11  TEMPOMESES            202383 non-null  float64       
 12  TEMPODIAS             202383 non-null  float64       
 13  VLMENSAL              201004 non-null  float64       
 14  VLIMPLANTACAO         202417 non-null  float64       
 15  VLAQUISICAO           202386 non-null  float64       
dtypes: datetime64[ns](1), float64(6), int64(1), object(8)
memory usage: 24.7+ MB
```

```
dataset
```

	CDCLIENTE	NOME	NMCLIENTE	UF	NMMODALIDADE	NMSEGMENTO	N
0	1	CONGER - Sistema de Contabilidade	Maracanaú	CE	Locação	INDÚSTRIA	CORF
1	1	Cálculos Financeiros e Comerc - FINANCE	Maracanaú	CE	Venda	INDÚSTRIA	CORF
2	1	Cálculos Financeiros e Comerc - FINANCE	Maracanaú	CE	Assinatura	INDÚSTRIA	CORF
3	1	Ente Pessoal	Maracanaú	CE	Implantação	INDÚSTRIA	CORF
4	1	Ente Pessoal	Maracanaú	CE	Venda	INDÚSTRIA	CORF
...	...	...	...	...	...	...	
202431	292652	Solução Total Contador-PCT Básico	Fortaleza	CE	Implantação	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	(
202432	292653	Fortes Contabil Limite de Usuários	Jaboatão dos Guararapes	PE	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	(
202433	292653	Fortes Fiscal Limite de Usuários	Jaboatão dos Guararapes	PE	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	(
202434	292653	Fortes Pessoal Limite de	Jaboatão dos Guararapes	PE	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E	(

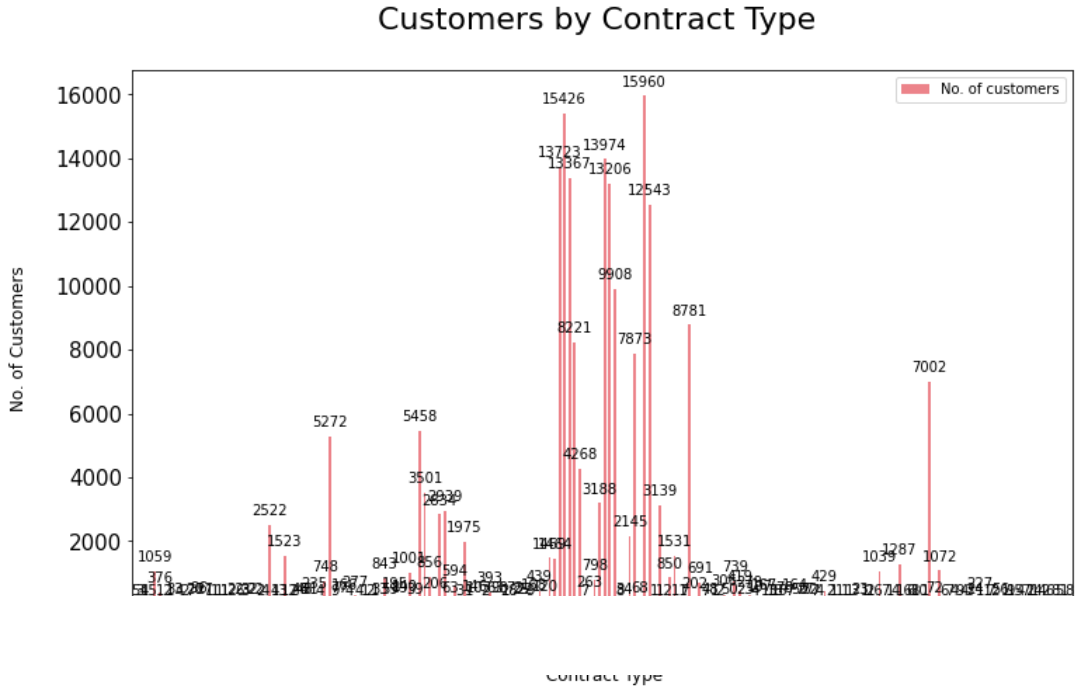
```
churn=[]
for index in range(len(dataset)):
    if (dataset['CANCELAMENTO'][index] is pd.NaT):
        #print('0')
        churn.append(0)
    else:
        #print('1')
        churn.append(1)

dataset['CHURN'] = churn


dataset.head()
```

	CDCLIENTE	NOME	NMCLIENTE	UF	NMMODALIDADE	NMSEGMENTO	NMMERCADO	I
0	1	CONGER - Sistema de Contabilidade	Maracanaú	CE	Locação	INDÚSTRIA	CORPORATIVO	
1	1	Cálculos Financeiros e	Maracanaú	CE	Venda	INDÚSTRIA	CORPORATIVO	

```
dataset2 = dataset[['TEMPOANOS',
'TEMPOMESES', 'TEMPODIAS','VLMENSAL',
'VLIMPLANTACAO', 'VLAQUISICAO', 'CHURN']]
#Histogram:
dataset2.hist(bins=30, figsize=(15, 10))
```

```
      array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fafedf392d0>,


    #Step 9.2. Analyze distribution of Key Categorical Variables------------------------------


    #(1) Distribution of Contract Type----------------------------------------------------------

    contract_split = dataset[[ "CDCLIENTE", "NOME"]]
    sectors = contract_split .groupby ("NOME")
    contract_split = pd.DataFrame(sectors["CDCLIENTE"].count())
    contract_split.rename(columns={'CDCLIENTE':'No. of customers'}, inplace=True)


    ax =  contract_split[["No. of customers"]].plot.bar(title = 'Customers by Contract Type',

    plt.ylabel('No. of Customers\n',horizontalalignment="center",fontstyle = "normal", fontsiz
    plt.xlabel('\n Contract Type',horizontalalignment="center",fontstyle = "normal", fontsize
    plt.title('Customers by Contract Type \n',horizontalalignment="center", fontstyle = "norma
    plt.legend(loc='top right', fontsize = "medium")
    plt.xticks(rotation=0, horizontalalignment="center")
    plt.yticks(rotation=0, horizontalalignment="right")

    x_labels = np.array(contract_split[["No. of customers"]])

    def add_value_labels(ax, spacing=5):
        for rect in ax.patches:
            y_value = rect.get_height()
            x_value = rect.get_x() + rect.get_width() / 2
            space = spacing
            va = 'bottom'
            if y_value < 0:
                space *= -1
                va = 'top'
            label = "{:.0f}".format(y_value)
            ax.annotate(
                label,
                (x_value, y_value),
                xytext=(0, space),
                textcoords="offset points",
                ha='center',
                va=va)
    add_value_labels(ax)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: MatplotlibDeprecatic
        best
        upper right
        upper left
        lower left
        lower right
        right
        center left
        center right
        lower center
        upper center
        center
This will raise an exception in 3.3.
```




```
df_final = df.loc[(df['VLMENSAL'] > 0)]
df_final
```

	CDCLIENTE	NOME	NMCLIENTE	UF	NMMODALIDADE	NMSEGMENTO
0	1	CONGER - Sistema de Contabilidade	Maracanaú	CE	Locação	INDÚSTRIA
2	1	Cálculos Financeiros e Comerc - FINANCE	Maracanaú	CE	Assinatura	INDÚSTRIA
5	1	Ente Pessoal	Maracanaú	CE	Manutenção/Locação com limite de horas	INDÚSTRIA
6	1	Ente Pessoal	Maracanaú	CE	Manutenção/Locação com limite de horas	INDÚSTRIA
8	1	FISCAL - Sistema de Escrita Fiscal	Maracanaú	CE	Locação	INDÚSTRIA
...	...	...	...	...	...	...
202429	292652	Fortes Fiscal Limite de Usuários	Fortaleza	CE	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS
202430	292652	Fortes Pessoal Limite de Usuários	Fortaleza	CE	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS
202432	292653	Fortes Contabil	Jaboatão dos	PE	Assinatura Limite	ATIVIDADES DE CONTABILIDADE,

```
df_final.isnull().sum()
```

```
CDCLIENTE                 0
NOME                      0
NMCLIENTE                84
UF                       84
NMMODALIDADE              0
NMSEGMENTO                0
NMMERCADO                 0
PRIMEIRAAQUISICAO         0
CANCELAMENTO          35415
NMMOTIVOCANCELAMENTO  35499
TEMPOANOS                 0
TEMPOMESES                0
TEMPODIAS                 0
VLMENSAL                  0
VLIMPLANTACAO             0
VLAQUISICAO               0
CHURN                     0
dtype: int64
```

```
df_final = df_final.drop(columns=['NMCLIENTE','UF','CANCELAMENTO'])
df_final.isnull().sum()
```

```
CDCLIENTE                 0
```

```
NOME                        0
NMMODALIDADE                0
NMSEGMENTO                  0
NMMERCADO                   0
PRIMEIRAAQUISICAO           0
NMMOTIVOCANCELAMENTO    35499
TEMPOANOS                   0
TEMPOMESES                  0
TEMPODIAS                   0
VLMENSAL                    0
VLIMPLANTACAO               0
VLAQUISICAO                 0
CHURN                       0
dtype: int64
```

```
df_final['NMMOTIVOCANCELAMENTO'] = df_final['NMMOTIVOCANCELAMENTO'].fillna(0)
```

```
df_final.isnull().sum()
```

```
CDCLIENTE               0
NOME                    0
NMMODALIDADE            0
NMSEGMENTO              0
NMMERCADO               0
PRIMEIRAAQUISICAO       0
NMMOTIVOCANCELAMENTO    0
TEMPOANOS               0
TEMPOMESES              0
TEMPODIAS               0
VLMENSAL                0
VLIMPLANTACAO           0
VLAQUISICAO             0
CHURN                   0
dtype: int64
```

```
#df_final[df_final['NMCLIENTE'].isnull()]
```

```
labelencoder = LabelEncoder()
# Assigning numerical values and storing in another column
df_final['NOME_LE'] = labelencoder.fit_transform(df_final['NOME'])
#df_final['NMMOTIVOCANCELAMENTO_LE'] = labelencoder.fit_transform(df_final['NMMOTIVOCANCEL
df_final['NMMOTIVOCANCELAMENTO_LE'] =le.fit_transform(df_final['NMMOTIVOCANCELAMENTO'].ast
df_final['NMSEGMENTO_LE'] = labelencoder.fit_transform(df_final['NMSEGMENTO'])
df_final['NMMODALIDADE_LE'] = labelencoder.fit_transform(df_final['NMMODALIDADE'])
df_final
```

	CDCLIENTE	NOME	NMMODALIDADE	NMSEGMENTO	NMMERCADO	I
0	1	CONGER - Sistema de Contabilidade	Locação	INDÚSTRIA	CORPORATIVO	
2	1	Cálculos Financeiros e Comerc - FINANCE	Assinatura	INDÚSTRIA	CORPORATIVO	2
5	1	Ente Pessoal	Manutenção/Locação com limite de horas	INDÚSTRIA	CORPORATIVO	2
6	1	Ente Pessoal	Manutenção/Locação com limite de horas	INDÚSTRIA	CORPORATIVO	2
8	1	FISCAL - Sistema de Escrita Fiscal	Locação	INDÚSTRIA	CORPORATIVO	
...	...	...	...	...	...	
202429	292652	Fortes Fiscal Limite de Usuários	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	CONTABIL	2
202430	292652	Fortes Pessoal Limite de Usuários	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	CONTABIL	2
202432	292653	Fortes Contabil Limite de Usuários	Assinatura Limite Usuário	ATIVIDADES DE CONTABILIDADE, AUDITORIA E AFINS	CONTABIL	2
202433	292653	Fortes Fiscal Limite de	Assinatura Limite	ATIVIDADES DE CONTABILIDADE,	CONTABIL	2

```
df_final.isnull().sum()
```

```
CDCLIENTE                 0
NOME                      0
NMMODALIDADE              0
NMSEGMENTO                0
NMMERCADO                 0
PRIMEIRAAQUISICAO         0
NMMOTIVOCANCELAMENTO      0
TEMPOANOS                 0
TEMPOMESES                0
TEMPODIAS                 0
VLMENSAL                  0
VLIMPLANTACAO             0
VLAQUISICAO               0
CHURN                     0
NOME_LE                   0
NMMOTIVOCANCELAMENTO_LE   0
NMSEGMENTO_LE             0
NMMODALIDADE_LE           0
dtype: int64
```

```
#df_final = df_final.drop(columns = "NMMOTIVOCANCELAMENTO")
df_final = df_final.drop(columns=['NMMOTIVOCANCELAMENTO','NOME','NMMODALIDADE','NMMERCADO'
df_final
```

	CDCLIENTE	TEMPOANOS	TEMPOMESES	TEMPODIAS	VLMENSAL	VLIMPLANTACAO	VLAQU
0	1	3.0	43.0	1323.0	60.000000	0.0	
2	1	0.0	0.0	23.0	125.000000	0.0	
5	1	1.0	22.0	654.0	478.265300	0.0	
6	1	2.0	27.0	820.0	533.600595	0.0	
8	1	3.0	43.0	1323.0	60.000000	0.0	
...	...	...	...	...	...	...	
202429	292652	0.0	0.0	2.0	100.000000	0.0	
202430	292652	0.0	0.0	2.0	100.000000	0.0	
202432	292653	0.0	0.0	1.0	86.670000	0.0	
202433	292653	0.0	0.0	1.0	86.660000	0.0	
202434	292653	0.0	0.0	1.0	86.670000	0.0	

96510 rows × 12 columns

```
#Step 10: Encode Categorical data----------------------------------------------------------

#Incase if user_id is an object:

identity = df_final["CDCLIENTE"]

df_final = df_final.drop(columns="CDCLIENTE")

# convert rest of categorical variable into dummy

df_final= pd.get_dummies(df_final)

#Rejoin userid to dataset (column concatenation)

df_final = pd.concat([df_final, identity], axis = 1)


df_final
```

	TEMPOANOS	TEMPOMESES	TEMPODIAS	VLMENSAL	VLIMPLANTACAO	VLAQUISICAO	CHU
0	3.0	43.0	1323.0	60.000000	0.0	0.0	
2	0.0	0.0	23.0	125.000000	0.0	0.0	
5	1.0	22.0	654.0	478.265300	0.0	0.0	
6	2.0	27.0	820.0	533.600595	0.0	0.0	
8	3.0	43.0	1323.0	60.000000	0.0	0.0	
...	...	...	...	...	...	...	
202429	0.0	0.0	2.0	100.000000	0.0	0.0	

```
#Step 11: Split dataset into dependent and independent variables-------------------------

#identify response variable:

response = df_final["CHURN"]

df_final = df_final.drop(columns="CHURN")


#Step 12: Generate training and test datasets of dependent and independent variables------


X_train, X_test, y_train, y_test = train_test_split(df_final, response,
                                                    stratify=response,
                                                    test_size = 0.2, #use 0.9 if data is h
                                                    random_state = 0)

#to resolve any class imbalance - use stratify parameter.

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

```
Number transactions X_train dataset:  (77208, 11)
Number transactions y_train dataset:  (77208,)
Number transactions X_test dataset:  (19302, 11)
Number transactions y_test dataset:  (19302,)
```

```
# Step 13: Removing Identifiers-----------------------------------------------------------

train_identity = X_train['CDCLIENTE']
X_train = X_train.drop(columns = ['CDCLIENTE'])

test_identity = X_test['CDCLIENTE']
X_test = X_test.drop(columns = ['CDCLIENTE'])



# Step 14: Feature Scaling-----------------------------------------------------------------
```

```
sc_X = StandardScaler()
X_train2 = pd.DataFrame(sc_X.fit_transform(X_train))
X_train2.columns = X_train.columns.values
X_train2.index = X_train.index.values
X_train = X_train2

X_test2 = pd.DataFrame(sc_X.transform(X_test))
X_test2.columns = X_test.columns.values
X_test2.index = X_test.index.values
X_test = X_test2



#Step 15.1: Compare Baseline Classification Algorithms - First Iteration
#Using Accuracy and ROC AUC Mean Metrics


models = []

models.append(('Logistic Regression', LogisticRegression(solver='liblinear', random_state
                                                         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)))



#Evaluating Model Results:


acc_results = []
auc_results = []
names = []
# set table to table to populate with performance results
col = ['Algorithm', 'ROC AUC Mean', 'ROC AUC STD',
       'Accuracy Mean', 'Accuracy STD']

model_results = pd.DataFrame(columns=col)
```

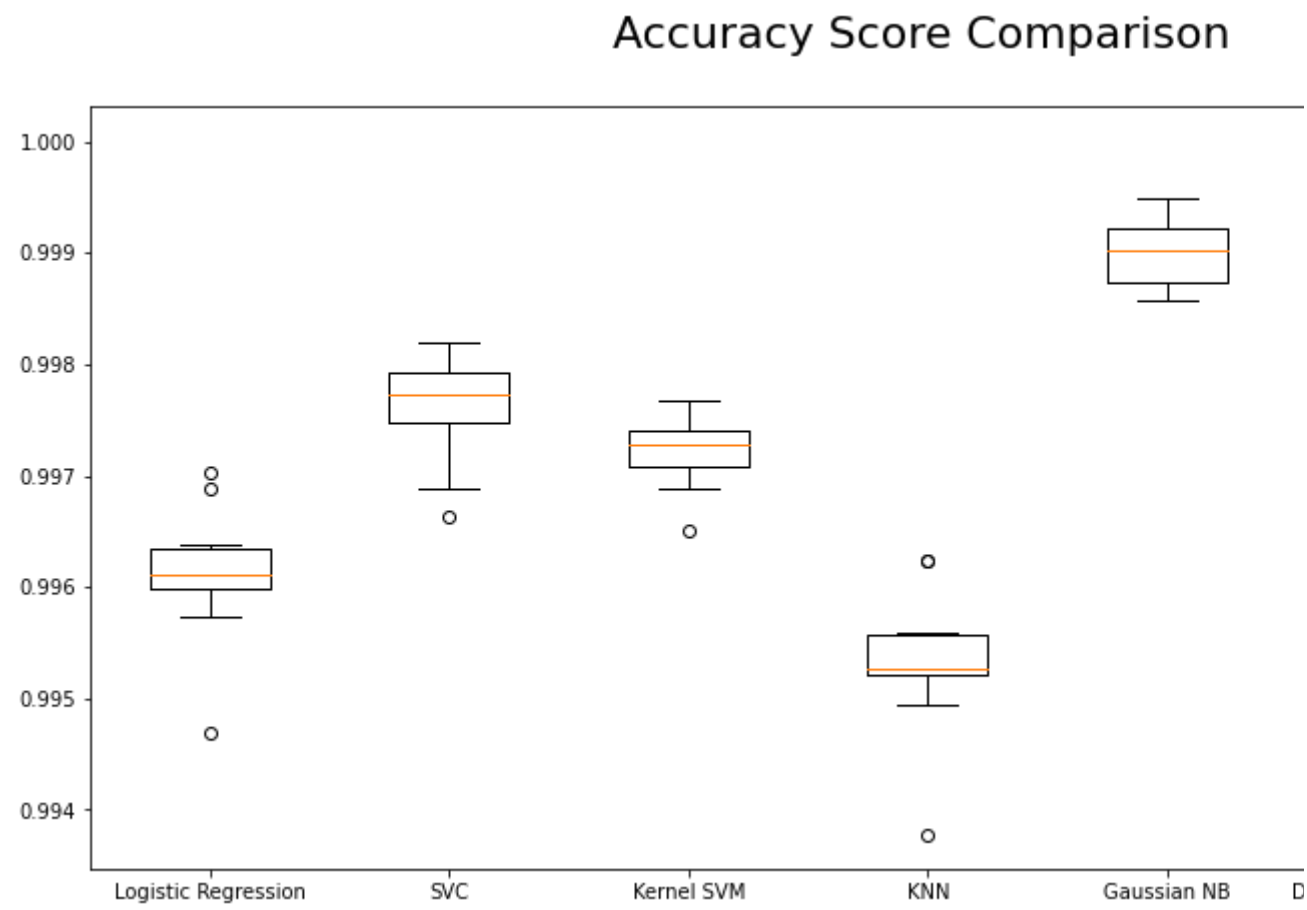
```
i = 0
# evaluate each model using k-fold cross-validation
for name, model in models:
    kfold = model_selection.KFold(
        n_splits=10, random_state=0)  # 10-fold cross-validation

    cv_acc_results = model_selection.cross_val_score(  # accuracy scoring
        model, X_train, y_train, cv=kfold, scoring='accuracy')

    cv_auc_results = model_selection.cross_val_score(  # roc_auc scoring
        model, X_train, y_train, cv=kfold, scoring='roc_auc')

    acc_results.append(cv_acc_results)
    auc_results.append(cv_auc_results)
    names.append(name)
    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)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:296: FutureW
  FutureWarning
```

	Algorithm	ROC AUC Mean	ROC AUC STD	Accuracy Mean	Accuracy STD
6	Random Forest	100.00	0.00	99.98	0.01
5	Decision Tree Classifier	99.99	0.01	99.99	0.01
0	Logistic Regression	99.98	0.01	99.61	0.06
2	Kernel SVM	99.98	0.01	99.72	0.03
4	Gaussian NB	99.97	0.02	99.90	0.03
1	SVC	99.95	0.02	99.76	0.05
3	KNN	99.83	0.03	99.53	0.07

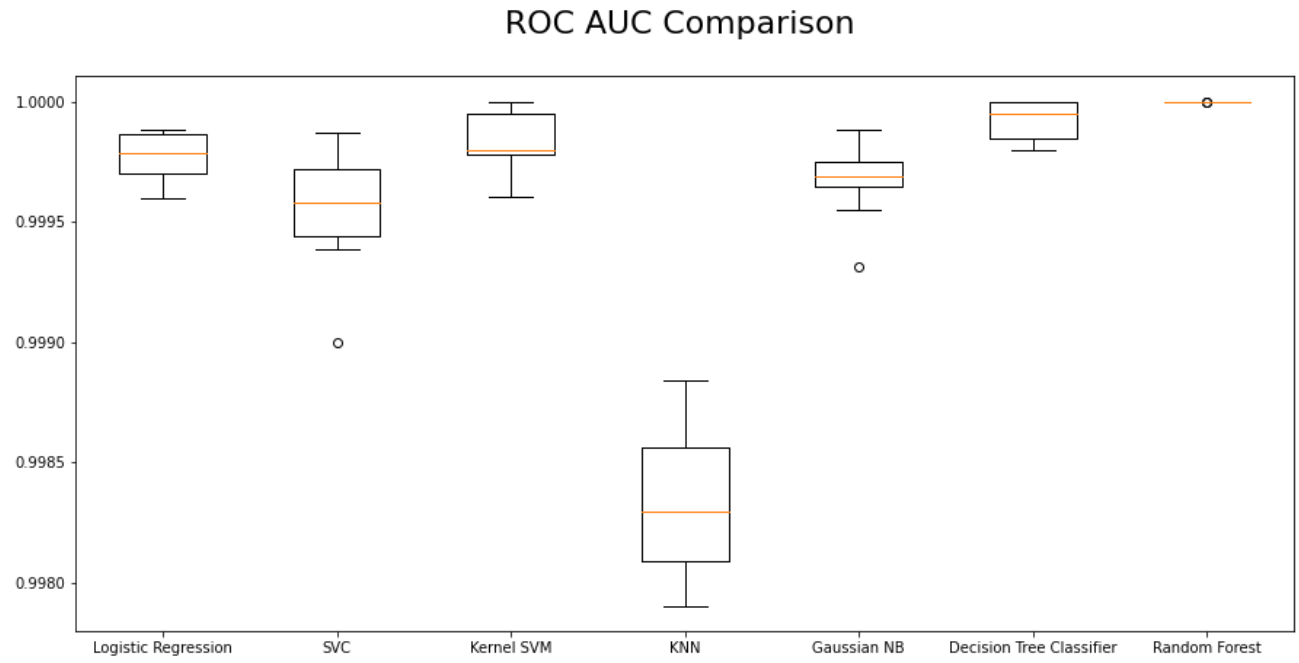
```
#Step 15.2.  Visualize Classification Algorithms Accuracy Comparisons:--------------------
```

```
#Using Accuracy Mean:

fig = plt.figure(figsize=(15, 7))
ax = fig.add_subplot(111)
plt.boxplot(acc_results)
ax.set_xticklabels(names)



#plt.ylabel('ROC AUC Score\n',horizontalalignment="center",fontstyle = "normal", fontsize
#plt.xlabel('\n Baseline Classification Algorithms\n',horizontalalignment="center",fontsty
plt.title('Accuracy Score Comparison \n',horizontalalignment="center", fontstyle = "normal
#plt.legend(loc='top right', fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")


plt.show()
```




```
#using Area under ROC Curve:

fig = plt.figure(figsize=(15, 7))
ax = fig.add_subplot(111)
plt.boxplot(auc_results)
ax.set_xticklabels(names)
```

```
#plt.ylabel('ROC AUC Score\n',horizontalalignment="center",fontstyle = "normal", fontsize
#plt.xlabel('\n Baseline Classification Algorithms\n',horizontalalignment="center",fontsty
plt.title('ROC AUC Comparison \n',horizontalalignment="center", fontstyle = "normal", font
#plt.legend(loc='top right', fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")


plt.show()
```




```
#--Step 15.4.1. Logistic Regression-----------------

# Fitting Logistic Regression to the Training set
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results

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([['Logistic Regression', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor



#Step 15.4.2. . Support Vector Machine (linear classifier)-----------------------


# Fitting SVM (SVC class) to the Training set:

classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results

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)

model_results = pd.DataFrame([['SVM (Linear)', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

results = results.append(model_results, ignore_index = True)


#Step 15.4.3. K-Nearest Neighbours-----------------------


# Fitting KNN to the Training set:

classifier = KNeighborsClassifier(n_neighbors = 22, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred  = classifier.predict(X_test)

#Evaluate results
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)

model_results = pd.DataFrame([['K-Nearest Neighbours', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

results = results.append(model_results, ignore_index = True)
```

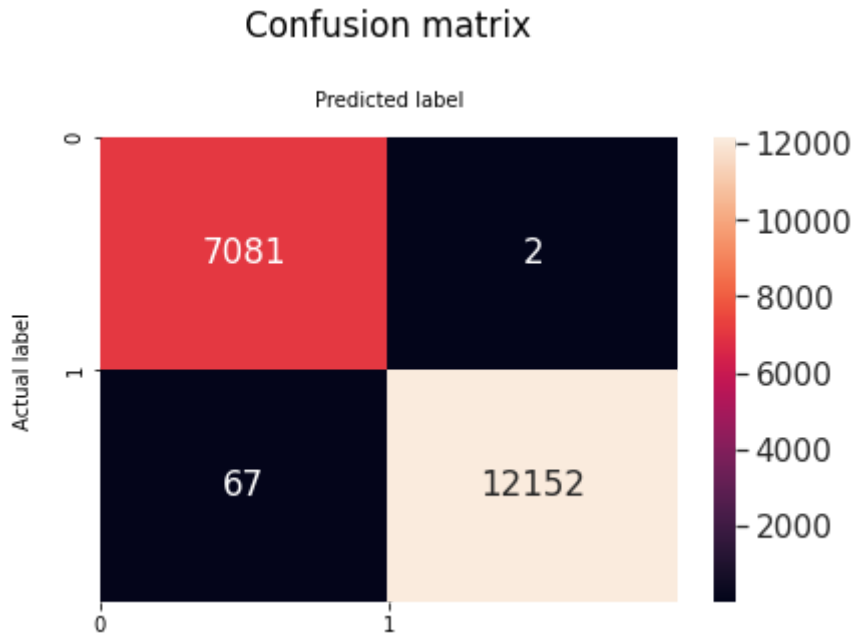
```
#Step 15.4.4.  Kernel SVM-----------------------

# Fitting Kernel SVM to the Training set:

classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results

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)

model_results = pd.DataFrame([['Kernel SVM', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

results = results.append(model_results, ignore_index = True)


#Step 15.4.5.  Naive Byes-----------------------------------------------

# Fitting Naive Byes to the Training set:

classifier = GaussianNB()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results
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)

model_results = pd.DataFrame([['Naive Byes', acc, prec, rec, f1, f2]],
                columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Sco

results = results.append(model_results, ignore_index = True)



#Step 15.4.6. Decision Tree-----------------------------------------------


# Fitting Decision Tree to the Training set:

classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results
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)

model_results = pd.DataFrame([['Decision Tree', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

results = results.append(model_results, ignore_index = True)


#Step 15.4.7. Random Forest-----------------------------------------


# Fitting Random Forest to the Training set:

classifier = RandomForestClassifier(n_estimators = 72, criterion = 'entropy', random_state
classifier.fit(X_train, y_train)



# Predicting the Test set results
y_pred = classifier.predict(X_test)

#Evaluate results

from sklearn.metrics import confusion_matrix, accuracy_score, f1_score, precision_score, r
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)

model_results = pd.DataFrame([['Random Forest', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

results = results.append(model_results, ignore_index = True)



#Step 15.5. Visualize the results and compare the baseline algorithms--------------------

# ============================================================================================
#Sort results based on the right classification metric:
#(Accuracy/ROC_AUC / Precision/Recall/F1/F2 scores)

#Since we have class imbalance. When we look into the business challenge,
# our false negatives will be costly and hence we need to Keep an eye onto the Precision,
```

```
# ============================================================================================

results = results.sort_values(["Precision", "Recall", "F2 Score"], ascending = False)


print (results)

                      Model  Accuracy  Precision    Recall  F1 Score  F2 Score
     5         Decision Tree  0.999896   0.999836  1.000000  0.999918  0.999967
     6         Random Forest  0.999845   0.999836  0.999918  0.999877  0.999902
     4            Naive Byes  0.998808   0.999836  0.998281  0.999058  0.998592
     1          SVM (Linear)  0.997306   0.999836  0.995908  0.997868  0.996691
     0   Logistic Regression  0.996425   0.999835  0.994517  0.997169  0.995576
     2  K-Nearest Neighbours  0.992436   0.999669  0.988379  0.993992  0.990616
     3            Kernel SVM  0.996736   0.999589  0.995253  0.997416  0.996117


#Step 16: Train & evaluate Chosen Model-----------------------------------------


# Fit Logistic Regression on the Training dataset:

classifier = LogisticRegression(random_state = 0, penalty = 'l2')
classifier.fit(X_train, y_train)


# Predict the Test set results

y_pred = classifier.predict(X_test)


#Evaluate Model Results on Test Set:

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([['Logistic Regression', acc, prec, rec, f1, f2]],
               columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'F2 Scor

print (results)

                     Model  Accuracy  Precision    Recall  F1 Score  F2 Score
     0  Logistic Regression  0.996425   0.999835  0.994517  0.997169  0.995576


# Re-check k-Fold Cross Validation:

accuracies = cross_val_score(estimator = classifier, X = X_train, y = y_train, cv = 10)
print("Logistic Regression Classifier Accuracy: %0.2f (+/- %0.2f)"  % (accuracies.mean(),

     Logistic Regression Classifier Accuracy: 1.00 (+/- 0.00)


#Visualize results on a Confusion Matrix:
```

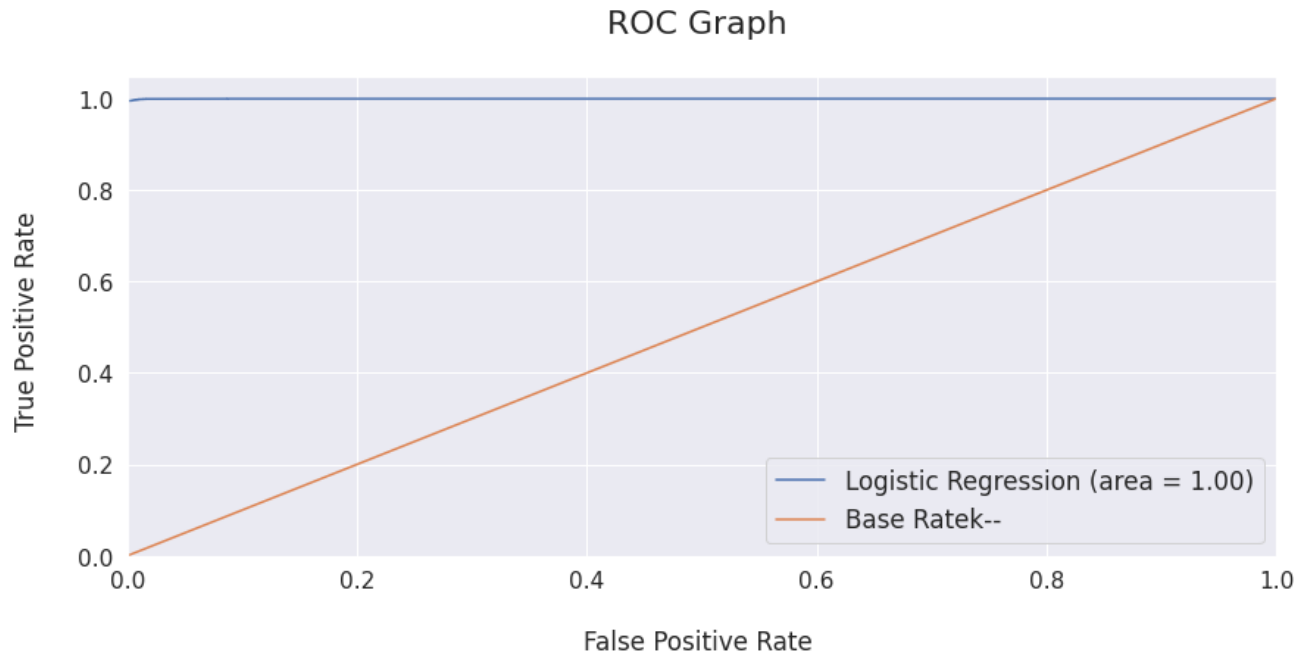
```
cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index = (0, 1), columns = (0, 1))
plt.figure(figsize = (28,20))


fig, ax = plt.subplots()
sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True, fmt='g'#,cmap="YlGnBu"
           )
class_names=[0,1]
tick_marks = np.arange(len(class_names))
plt.tight_layout()
plt.title('Confusion matrix\n', y=1.1)
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
ax.xaxis.set_label_position("top")
plt.ylabel('Actual label\n')
plt.xlabel('Predicted label\n')
```

```
Text(0.5, 15.0, 'Predicted label\n')
<Figure size 2016x1440 with 0 Axes>
```




```
# Evaluate the model using ROC Graph

classifier.fit(X_train, y_train)
probs = classifier.predict_proba(X_test)
probs = probs[:, 1]
classifier_roc_auc = accuracy_score(y_test, y_pred )


rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, classifier.predict_proba(X_test)[:,1])
plt.figure(figsize=(14, 6))

# Plot Logistic Regression ROC
plt.plot(rf_fpr, rf_tpr, label='Logistic Regression (area = %0.2f)' % classifier_roc_auc)
# 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", font
plt.xlabel('\nFalse Positive Rate \n',horizontalalignment="center",fontstyle = "normal", f
plt.title('ROC Graph \n',horizontalalignment="center", fontstyle = "normal", fontsize = "2
plt.legend(loc="lower right", fontsize = "medium")
plt.xticks(rotation=0, horizontalalignment="center")
plt.yticks(rotation=0, horizontalalignment="right")



plt.show()
```




```
#Step 17:Predict Feature Importance----------------------------------------------------------


# Analyzing Coefficients
feature_importances = pd.concat([pd.DataFrame(df_final.drop(columns = 'CDCLIENTE').columns
           pd.DataFrame(np.transpose(classifier.coef_), columns = ["coef"])
           ],axis = 1)

feature_importances.sort_values("coef", ascending = False)
```

	NOME	coef
7	NMMOTIVOCANCELAMENTO_LE	20.950412
6	NOME_LE	0.486890
9	NMMODALIDADE_LE	0.417257
1	TEMPOMESES	0.414136
8	NMSEGMENTO_LE	0.271470
4	VLIMPLANTACAO	0.000000
5	VLAQUISICAO	0.000000
3	VLMENSAL	-0.367224
0	TEMPOANOS	-0.405401
2	TEMPODIAS	-0.570000

```
#Step 18.3:Final Hyper parameter tuning and selection ----------------------------------


lr_classifier = LogisticRegression(random_state = 0, penalty = 'l2')
lr_classifier.fit(X_train, y_train)


# Predict the Test set results

y_pred = lr_classifier.predict(X_test)

#probability score
y_pred_probs = lr_classifier.predict_proba(X_test)
y_pred_probs  = y_pred_probs [:, 1]


# Step 20: Format Final Results:-------------------------------------------------------


final_results = pd.concat([test_identity, y_test], axis = 1).dropna()

final_results['predictions'] = y_pred

final_results["propensity_to_convert(%)"] = y_pred_probs

final_results["propensity_to_convert(%)"] = final_results["propensity_to_convert(%)"]*100

final_results["propensity_to_convert(%)"]=final_results["propensity_to_convert(%)"].round(

final_results = final_results[['CDCLIENTE', 'CHURN', 'predictions', 'propensity_to_convert

final_results ['Ranking'] = pd.qcut(final_results['propensity_to_convert(%)'].rank(method

print (final_results)

            CDCLIENTE  CHURN  predictions  propensity_to_convert(%) Ranking
     71541      272424      1            1                   100.00       7
```

```
        135829     281967      1            1                    100.00       7
        186414     289479      0            0                      1.30       7
        91013      275149      1            1                    100.00       7
        178042     288140      0            0                      2.06       7
        ...           ...    ...          ...                       ...     ...
        145263     283164      1            1                    100.00       1
        37899        6197      0            0                      0.90       8
        78014      273288      0            0                      0.63       9
        130681     281247      1            1                    100.00       1
        5905          705      1            1                    100.00       1

        [19302 rows x 5 columns]
```

```
#Step 21: Save the model-----------------------------------------------------------------

filename = 'final_model.model'
i = [lr_classifier]
joblib.dump(i,filename)
```

✓ 0s conclusão: 20:33