**Python Codes: (Telecom Customer Churn Customer)**

```
Import numpy as np# linear algebra
Import pandas as pd# data processing, CSV file I/O (e.g. pd.read_csv)

Import os for dirname, _, filenames in os.walk('/kaggle/input'):
for filename in filenames:
print(os.path.join(dirname, filename))
```

```
pd.set_option('display.max_rows', 500) pd.set_option('display.max_columns', 500)
import warnings warnings.filterwarnings('ignore')
```

Step 1: **Read data from Sources:**

```
# import customer data
customer_df = pd.read_csv('C:/Users/user/Desktop/SMBA PROJECT2/customer_data.csv')
```

```
# import churn_data
churn_df = pd.read_csv('C:/Users/user/Desktop/SMBA PROJECT2/churn_data.csv')
```

```
# import internet data
internet_df = pd.read_csv('C:/Users/user/Desktop/SMBA PROJECT2/internet_data.csv')
```

Lets explore the data

```
customer_df.head()
customer_df.shape
```

```
churn_df.head()
internet_df.head()
```

```
# join all the columns by customer ID
print(len(np.setdiff1d(customer_df.customerID, internet_df.customerID)))telecom_df = pd.merge(internet_df, df1,
how='inner', on='customerID')
#merge customer and churn dataframes into df1
df1 = pd.merge(customer_df, churn_df, how='inner', on='customerID')
```

```
# merge df1 and internet dataframes to telecom df
telecom_df = pd.merge(internet_df, df1, how='inner', on='customerID')
```

```
# explore final telecom dftelecom_df.head()
```

```
telecom_df.columns
```

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

```
# check the data types
telecom_df.info()
TotalCharges is an object and not float!!!
```

```
# We don´t have null values but from error we can see that column 'TotalCharges' contains whitespace = ' '
# telecom_df['TotalCharges'] = pd.to_numeric(telecom_df['TotalCharges'])
# How many whitespace = ' ' we have in column 'TotalCharges'
telecom_df['TotalCharges'].str.isspace().value_counts
```

```
telecom_df['TotalCharges'].isnull().sum()
# Replacing whitespace to NAN values and converting to numeric data (float)
telecom_df['TotalCharges'] = telecom_df['TotalCharges'].replace(' ', np.nan)
telecom_df['TotalCharges'] = pd.to_numeric(telecom_df['TotalCharges'])
```

```
# How many NAN values is in column
telecom_df['TotalCharges'].isnull().sum()
```

```
# Replacing NAN values with mean value from all data in column 'TotalCharges'
#new_value = telecom_df['TotalCharges'].astype('float').mean(axis=0)
new_value = (telecom_df['TotalCharges']/telecom_df['MonthlyCharges']).mean()*telecom_df['MonthlyCharges']
telecom_df['TotalCharges'].replace(np.nan, new_value, inplace=True)
```

```
# How many NAN values is in column 'TotalCharges' after replacing NAN with mean
telecom_df['TotalCharges'].isnull().sum()
# Checking for null valuestelecom_df.isnull().sum()
```

```
telecom_df.TotalCharges.dtype
```

```
# analyze customerID
telecom_df.customerID.nunique()
# analyze MultipleLinestelecom_df.MultipleLines.value_counts()
```

```
# analyze OnlineSecurity
telecom_df.OnlineSecurity.value_counts()
# analyze InternetService
telecom_df.InternetService.value_counts()
```

```
#analyze DeviceProtection
telecom_df.DeviceProtection.value_counts()
```

```
# analye TechSupport
```

```
telecom_df.TechSupport.value_counts()
# analye StreamingTV
telecom_df.StreamingTV.value_counts()
```

```
# analyze StreamingMovies
telecom_df.StreamingMovies.value_counts()
analyze gender
telecom_df.gender.value_counts()
# analyze SeniorCitizen
telecom_df.SeniorCitizen.value_counts()
# analyze partner
telecom_df.Partner.value_counts()
# analyze dependents
telecom_df.Dependents.value_counts()
# analyze tenure
np.sort(telecom_df.tenure.unique())
analyze phone services
telecom_df.PhoneService.value_counts()
telecom_df.Contract.value_counts()
```

```
# analyze paperless billing
telecom_df.PaperlessBilling.value_counts()
# analyze paymentmethod
telecom_df.PaymentMethod.value_counts()
telecom_df.Churn.value_counts()
sns.countplot(x="Churn",data=telecom_df)
```

```
# import required visual libraries
Import matplotlib.pyplot as plt import seaborn as sns
```

```
def category_plot(df_src, df_by,
    h_v='h'):frequency_table(df_src)
    fig, ax = plt.subplots(1,2,figsize=(10,5))
    ax[1] = sns.countplot(x=df_src, hue=df_by, ax=ax[1], palette="Set3")
    ax[1].set(xlabel=df_src.name, ylabel=df_by.name, title = df_src.name +' vs '+ df_by.name +' plot')
    values = df_src.value_counts(normalize=True)*100
    ax[0] = sns.countplot(x=df_src, palette='Set3', ax=ax[0])
    ax[0].set(xlabel=df_src.name, ylabel ='Count', title='Frequency Plot')
if (h_v =='v'):
        ax[0].set_xticklabels(ax[0].get_xticklabels(), rotation=45)
        ax[1].set_xticklabels(ax[1].get_xticklabels(), rotation=45)

    plt.show()
```

```
def get_percent(value, total, round_number =2):
```

```
return round(100* value / total, round_number)
```

```
def frequency_table(df,with_percent=True, with_margins=False):
    freq_df = pd.crosstab(index=df, columns="count", margins=with_margins).reset_index()
if with_percent:
        freq_df['percent(%)'] = get_percent(freq_df['count'] , df.shape[0])
print(freq_df)
```

**perform univariant analysis to understand the churn**

```
categorial_columns = telecom_df.select_dtypes(['object']).columnscategorial_columns
```

```
# univariant analyis on MultipleLines
category_plot(telecom_df.MultipleLines, telecom_df.Churn)
# univariant analyis on InternetServices
category_plot(telecom_df.InternetService, telecom_df.Churn)
```

```
# univariant analyis on OnlineSecurity
category_plot(telecom_df.OnlineSecurity, telecom_df.Churn)
```

```
# univariant analyis on OnlineBackup
category_plot(telecom_df.OnlineBackup, telecom_df.Churn)
```

```
# univariant analyis on DeviceProtection
category_plot(telecom_df.DeviceProtection, telecom_df.Churn)
# univariant analyis on TechSupport
category_plot(telecom_df.TechSupport, telecom_df.Churn)
```

```
 univariant analyis on StreamingTVc
ategory_plot(telecom_df.StreamingTV, telecom_df.Churn)
```

```
# univariant analyis on Streaming Movies
category_plot(telecom_df.StreamingMovies, telecom_df.Churn)
```

```
# univariant analyis on Partner
category_plot(telecom_df.Partner, telecom_df.Churn)
# univariant analyis on Dependentscategory_plot(telecom_df.Dependents, telecom_df.Churn)
# univariant analyis on PhoneService
category_plot(telecom_df.PhoneService, telecom_df.Churn)
```

```
# univariant analyis on Contract
category_plot(telecom_df.Contract, telecom_df.Churn)
```

```
# univariant analyis on Contract
category_plot(telecom_df.PaperlessBilling, telecom_df.Churn)
```

```
# univariant analyis on Contract
category_plot(telecom_df.PaymentMethod, telecom_df.Churn, h_v='v')
```

```
# check for missing values in the data set
telecom_df.isnull().sum()
```

```
get_boxplot(df,ax):
    ax = sns.boxplot(df, ax=ax, palette="Reds",
                     medianprops =dict(linestyle='-', linewidth=2, color='Yellow'),
                    width =0.4, notch=True,
                    boxprops =dict(linestyle='-', linewidth=2))
return ax
```

```
def dist_plot(df, plots=1):
    fig, ax = plt.subplots(1,2, figsize=(10,4))
    ax[0] = get_boxplot(df, ax[0])
    ax[1] = sns.distplot(df, ax=ax[1], kde_kws={"color": "y"}, hist_kws={"histtype": "step", "color": "k"})
    ax[1].axvline(x = df.mean(), color ='r', linewidth=1.5, linestyle='--', label='mean')
    ax[1].axvline(x = df.median(), color ='g', linewidth=1.5, linestyle='--', label='median')
    ax[1].set(xlabel = df.name, ylabel='frequency', title='Histogram of '+ df.name)
    plt.legend()
    plt.tight_layout()
```

```
dist_plot(telecom_df.TotalCharges)
```

```
dist_plot(telecom_df.MonthlyCharges)
dist_plot(telecom_df.tenure)
```

```
sns.pairplot(telecom_df,vars= ['tenure','MonthlyCharges','TotalCharges'], hue="Churn")
```

```
g = sns.FacetGrid(telecom_df, row='SeniorCitizen', col="gender", hue="Churn", height=3.5)g.map(plt.scatter, "tenure
", "MonthlyCharges", alpha=0.6)g.add_legend();
```

Step 4: **Data preparation**

```
set(telecom_df.dtypes)

# List of variables to map
varlist =  ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']
# Defining the map function
def binary_map(x):
return x.map({'Yes': 1, "No": 0})
# Applying the function to the housing list
telecom_df[varlist] = telecom_df[varlist].apply(binary_map)
```

```
telecom_df.head()
```

```
telecom_df.columns
```

```
# Creating dummy variables for the variable 'MultipleLines'
ml = pd.get_dummies(telecom_df['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'], 1)
#Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,ml1], axis=1)
```

```
# Creating dummy variables for the variable 'Internetservice'
iss = pd.get_dummies(telecom_df.InternetService)
# Dropping InternetService_No column
iss = iss.drop(['No'], axis=1)telecom_df = pd.concat([telecom_df, iss], axis=1)
```

```
# Creating dummy variables for the variable 'OnlineSecurity'
.os = pd.get_dummies(telecom_df['OnlineSecurity'], prefix='OnlineSecurity')os1 = os.drop(['OnlineSecurity_No intern
et service'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,os1], axis=1)
```

```
# Creating dummy variables for the variable 'DeviceProtection'.
```

```
dp = pd.get_dummies(telecom_df['DeviceProtection'], prefix='DeviceProtection')dp1 = dp.drop(['DeviceProtection_N
o internet service'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,dp1], axis=1)
```

```
# Creating dummy variables for the variable 'TechSupport'. ts = pd.get_dummies(telecom_df['TechSupport'], prefix='
TechSupport')ts1 = ts.drop(['TechSupport_No internet service'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,ts1], axis=1)
```

```
# Creating dummy variables for the variable 'StreamingTV'.
st =pd.get_dummies(telecom_df['StreamingTV'], prefix='StreamingTV')st1 = st.drop(['StreamingTV_No internet servi
ce'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,st1], axis=1)
```

```
# Creating dummy variables for the variable 'StreamingMovies'.
sm = pd.get_dummies(telecom_df['StreamingMovies'], prefix='StreamingMovies')sm1 = sm.drop(['StreamingMovies_
No internet service'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,sm1], axis=1)
```

```
# Defining the map function
def gender_map(x):
return x.map({'Female': 1, "Male": 0})
# Applying the function to the housing list
telecom_df['gender'] = telecom_df[['gender']].apply(gender_map)
```

```
cc = pd.get_dummies(telecom_df.Contract)
# Adding the results to the master dataframet
telecom_df = pd.concat([telecom_df,cc], axis=1)
```

```
# Creating dummy variables for the variable 'OnlineBackup'
ob = pd.get_dummies(telecom_df['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,ob1], axis=1)
```

```
pm = pd.get_dummies(telecom_df.PaymentMethod)
# Adding the results to the master dataframe
telecom_df = pd.concat([telecom_df,pm], axis=1)
```

**Final Dataset after Data Preparation**

```
telecom_df.head()
telecom_df.columns
```

```
# drop the original columns
telecom_df = telecom_df.drop(['MultipleLines', 'InternetService', 'OnlineSecurity','OnlineBackup',
'DeviceProtection', 'TechSupport','StreamingTV', 'StreamingMovies'], axis=1)
```

```
telecom_df = telecom_df.drop(['PaymentMethod', 'Contract',], axis=1)
```

**Final Dataset after dropping Original Columns for which dummies are created**

```
telecom_df.head()
```

```
telecom_df.shape
```

```
# import required libraries
from sklearn.model_selection import train_test_split
```

```
# Create dependent and independent data frames
X = telecom_df.drop(['customerID', 'Churn'], axis=1)# drop CustomerID and churn columns from X
Y = telecom_df['Churn']
```

```
X.shape
Y.shape
# perform train-test split
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, random_state =100)
```

Step 6: **Feature Scaling**

```
# import Standard Scaler from preprocesing module
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

**Also, apply normalization to x in order to scale all values**

```
scale_columns = ['MonthlyCharges', 'tenure', 'TotalCharges']X_train[scale_columns] = scaler.fit_transform(X_train[sc
ale_columns])
```

```
X_train.head()
```

```
scale_columns = ['MonthlyCharges', 'tenure', 'TotalCharges']X_test[scale_columns] = scaler.fit_transform(X_test[scal
e_columns])
X_test.columns
X_train.shape
X_test.shape
X_train.index
```

```
def rate(df):
print(df.name,  ' Rate : ', round(100*sum(df) /len(df), 2), '%')
```

```
# check churn rate in the data set.
rate(y_train)
```

```
Churn  Rate :  26.46 %
```

**Co-relation Matrix**

```
# check co-realtion between the variables
plt.figure(figsize=(30,20))sns.heatmap(X_train.corr(), annot=True)plt.show()
```

```
#drop co-realted columns
corelated_cols = ['MultipleLines_No', 'OnlineSecurity_No', 'OnlineBackup_No', 'DeviceProtection_No', 'TechSupport
_No',
'StreamingTV_No', 'StreamingMovies_No']X_train = X_train.drop(corelated_cols, axis=1)X_test = X_test.drop(corela
ted_cols, axis=1)
```

```
X_train.head()
X_train.shape
X_test.shape
```

```
import statsmodels.api as sm
```

```
def get_lrm(y_train, x_train):
    lrm = sm.GLM(y_train, (sm.add_constant(x_train)), family = sm.families.Binomial())
    lrm = lrm.fit()
print(lrm.summary())
return lrm
```

**LOGIT MODEL¶**

```
# running the logistic regression model once
lrm_1 =get_lrm(y_train, X_train)
X_train = sm.add_constant(X_train)
```

```
logit = sm.Logit(endog=y_train, exog = X_train)
result = logit.fit()
result.summary()
```

**PROBIT MODEL**

```
X_train = sm.add_constant(X_train)
probit = sm.Probit(endog=y_train, exog = X_train)
result = probit.fit()
result.summary()
```

**RECURSIVE FEATURE ELIMINATION¶**

```
# using RFE remove some features.# import required libraries
from sklearn.feature_selection import RFE from
sklearn.linear_model import LogisticRegression
```

```
lg_reg = LogisticRegression()rfe = RFE(lg_reg, 15)rfe = rfe.fit(X_train, y_train)
```

```
rfe_df = pd.DataFrame({'columns': list(X_train.columns), 'rank' : rfe.ranking_, 'support' : rfe.support_ }).sort_values(b
y='rank', ascending=True)rfe_df
```

```
# get supported columns
rfe_columns = X_train.columns[rfe.support_]rfe_columns
```

```
# import vif from statsmodel
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
def calculate_vif(df):
    vif = pd.DataFrame()
    vif['Features'] = df.columns
    vif['vif'] = [variance_inflation_factor(df.values, i) for i inrange(df.shape[1])]
    vif['vif'] =round(vif['vif'],2)
    vif = vif.sort_values(by='vif', ascending=False)
print(vif)
```

```
X_train_lg_1 = X_train[rfe_columns]log_reg_1 = get_lrm(y_train, X_train_lg_1)
```

```
#Pseudo R-squared value of MODEL 1
X_train = sm.add_constant(X_train)
```

```
logit = sm.Logit(endog=y_train, exog = X_train_lg_1)
result = logit.fit()
result.summary()
```

```
 #drop Mailed check column due to high p-value
X_train_lg_2 = X_train_lg_1.drop(['Mailed check'], axis=1)
```

MODEL 2

```
log_reg_2 = get_lrm(y_train, X_train_lg_2)
```

```
calculate_vif(X_train_lg_2)
```

```
# drop Month-to-Month as it it highly co-realted
X_train_lg_3 = X_train_lg_2.drop(['Month-to-month'], axis=1)
```

```
#Pseudo R-squared value of MODEL 2
X_train = sm.add_constant(X_train)
logit = sm.Logit(endog=y_train, exog = X_train_lg_2)
result = logit.fit()
result.summary()
```

MODEL 3

```
log_reg_3 = get_lrm(y_train, X_train_lg_3)
```

```
calculate_vif(X_train_lg_3)
```

```
# drop 'TotalCharges' as it is hightly co-realted with other features
X_train_lg_4 = X_train_lg_3.drop(['TotalCharges'], axis=1)
```

```
#Pseudo R-squared value of MODEL 3
X_train = sm.add_constant(X_train)
logit = sm.Logit(endog=y_train, exog = X_train_lg_3)
result = logit.fit()
result.summary()
```

MODEL 4

```
log_reg_4 = get_lrm(y_train, X_train_lg_4)
```

```
# drop DSL, as it is insignificant
X_train_lg_5 = X_train_lg_4.drop(['DSL'], axis=1)
#Pseudo R-squared value of MODEL 4
X_train = sm.add_constant(X_train)
logit = sm.Logit(endog=y_train, exog = X_train_lg_4)
result = logit.fit()
result.summary()
```

MODEL 5

```
log_reg_5 = get_lrm(y_train, X_train_lg_5)
```

```
# drop fiber optic as it is insignificant
X_train_lg_6 = X_train_lg_5.drop(['Fiber optic'], axis=1)
```

```
#Pseudo R-squared value of MODEL 5
X_train = sm.add_constant(X_train)
logit = sm.Logit(endog=y_train, exog = X_train_lg_5)
result = logit.fit()
result.summary()
```

MODEL 6

```
log_reg_6 = get_lrm(y_train, X_train_lg_6)
```

```
# looks all features are significant, lets check VIF
calculate_vif(X_train_lg_6)
```

```
#Pseudo R-squared value of MODEL 6
X_train = sm.add_constant(X_train)
logit = sm.Logit(endog=y_train, exog = X_train_lg_6)
result = logit.fit()
result.summary()
```

```
# predict the values from the model
y_train_pred = log_reg_6.predict(sm.add_constant(X_train_lg_6))y_train_pred[:10]
```

```
y_train_pred_values = y_train_pred.values.reshape(-1)y_train_pred_values[:10]
```

```
X_train_lg_6.columns
```

```
# create a data frame having actual, customerID and predicted
churn_df = pd.DataFrame({'Churn_actual': y_train.values, 'Churn_prob' : y_train_pred_values})
churn_df['Cust_ID'] = y_train.indexchurn_df.head()
```

```
from sklearn import metrics

from sklearn.ensemble import
RandomForestClassifierfromsklearn.metrics
import confusion_matrix, classification_report, accuracy_score, roc_curve, auc
# All the features are significant and there is no co-realtion between the variables.
# calucate the confusion matrix.
```

```
print(classification_report(churn_df.Churn_actual, churn_df.Churn_Pred))
```

```
# confusion matrix visualization
f, ax = plt.subplots(figsize = (5,5))sns.heatmap(cnf_matrix, annot =True, linewidths =0.5, color ="red", fmt =".0f", ax
=None)plt.xlabel("Predicted value")plt.ylabel("True value")plt.title("Confusion Matrix of Logistic Regression")plt.sho
w()
```

```
# calculate the accuracy
print('Accuracy of the model : ', metrics.accuracy_score(churn_df.Churn_actual, churn_df.Churn_Pred))
```

```
Accuracy of the model :  0.8058429701765064
```

```
print('Recall : ', metrics.recall_score(churn_df.Churn_actual, churn_df.Churn_Pred))
```

```
Recall :  0.5276073619631901
```

```
print('Precision : ', metrics.precision_score(churn_df.Churn_actual, churn_df.Churn_Pred))
```

```
Precision :  0.6686103012633625
```

```
tn = cnf_matrix[0,0]fn = cnf_matrix[1,0]fp = cnf_matrix[0,1]tp = cnf_matrix[1,1]
```

```
# Sensistivity , True Positive rateprint('Sensitivity (True Positive Rate) TP / TP + FN : ', tp / (tp + fn))
```

```
Sensitivity (True Positive Rate) TP / TP + FN :  0.5276073619631901
```

```
# specificity, print('Specificity TN / (TN + FP) : ', tn / (tn + fp))
```

```
Specificity TN / (TN + FP) :  0.9059310344827586
```

```
# False positive rateprint('False positive rate FP / (TN + FP) : ', fp / (tn+fp))
```

```
False positive rate FP / (TN + FP) :  0.09406896551724138
```

Step 8 : **ROC Curve**

```
defdraw_roc_curve(actual, probs):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs, drop_intermediate =False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)'% auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.legend(loc="lower right")
    plt.show()
```

```
draw_roc_curve(churn_df.Churn_actual, churn_df.Churn_prob)
```

```
# to the predict for different thresholds
tresholds = [float(x)/10for x inrange(10)]tresholds.append(0.45)tresholds.append(0.55)tresholds =sorted(tresholds)
for i insorted(tresholds):
    churn_df[i] = churn_df.Churn_prob.map(lambda row: 1if row > i else0)
churn_df.head()
```

```
optimal_df = pd.DataFrame(columns=['prob', 'accuracy', 'sensitivity', 'specificity'])for i in tresholds:
    cm = metrics.confusion_matrix(churn_df.Churn_actual, churn_df[i])
    tn = cm[0,0]
    fn = cm[1,0]
    fp = cm[0,1]
    tp = cm[1,1]
    accuracy = (tn + tp) / (tn + tp + fp + fn)
    specificity = tn / (tn + fp)
    sensitivity = tp / (tp + fn)
    optimal_df.loc[i] = [i, accuracy, sensitivity, specificity]
```

```
optimal_df
```

```
# plot the curve
optimal_df.plot(x ='prob', y=['accuracy', 'sensitivity', 'specificity'])plt.show()
```

```
# from the above curve, optimal value
optimal_value =0.3
```

```
churn_df['final_pred'] = churn_df.Churn_prob.map(lambda x: 1if x >0.3else0)
churn_df.head()
```

```
# calcualte the accuracyfinal_accuracy = metrics.accuracy_score(churn_df.Churn_actual, churn_df.final_pred)print('F
inal Accuracy : ', final_accuracy)
```

```
Final Accuracy :  0.7715560965713126
```

```
# calcualte the other parameters
final_cm = metrics.confusion_matrix(churn_df.Churn_actual, churn_df.final_pred)
print('Confusion matrix \n', final_cm)
# confusion matrix visualization
f, ax = plt.subplots(figsize = (5,5))sns.heatmap(final_cm, annot =True, linewidths =0.5, color ="red", fmt =".0f", ax=
None)
plt.xlabel("Predicted value")
plt.ylabel("True value")
plt.title("Confusion Matrix of Logistic Regression final model when threshold=0.3")
plt.show()
```

```
print(classification_report(churn_df.Churn_actual, churn_df.final_pred))
```

```
tn = final_cm[0,0]fn = final_cm[1,0]fp = final_cm[0,1]tp = final_cm[1,1]
sensitivity = tp / (tp + fn)
specificity = tn / (tn + fp)
false_positive_rate =1– specificity
positive_predictive_rate = tp / (tp + fp)
negative_predictive_rate = tn / (tn + fn)
```

```
print('optimal threshold : ', optimal_value)
print('sensitivity : ', sensitivity)
print('specificity : ', specificity)
print('false_positive_rate : ', false_positive_rate)
print('positive_predictive_rate : ', positive_predictive_rate)
print('negative_predictive_rate : ', negative_predictive_rate)
```

```
# precision and recall trade off
```

```
fromsklearn.metricsimport precision_recall_curve
```

```
p, r, tresholds = precision_recall_curve(churn_df.Churn_actual, churn_df.Churn_prob)
```

```
plt.plot(tresholds, p[:-1], 'g-')plt.plot(tresholds, r[:-1], 'r-')plt.show()
```

```
X_test = sm.add_constant(X_test)
```

```
fromsklearn.linear_modelimport LogisticRegression
lr_model = LogisticRegression()lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy on test dataset is :",accuracy_lr)
```

```
Logistic Regression accuracy on test dataset is : 0.7884524372929484
```

```
X_test = X_test[X_train_lg_6.columns]X_test.head()
```

```
# predict the X_test
y_test_pred = log_reg_6.predict(sm.add_constant(X_test))
```

```
test_pred_df = pd.DataFrame(y_test)
test_pred_df.head(5)
```

```
y_test_df = pd.DataFrame(y_test_pred)y_test_df['CustID'] = y_test_df.index
y_test_df.head()
```

```
y_test_df.reset_index(drop=True, inplace=True)
test_pred_df.reset_index(drop=True, inplace=True)
```

```
test_pred_final_df = pd.concat([ test_pred_df, y_test_df], axis=1)
test_pred_final_df.head()
```

```
test_pred_final_df= test_pred_final_df.rename(columns={0 : 'Churn_Prob', 'Churn': 'Churn_Actual'})
test_pred_final_df.head()
```

```
test_pred_final_df['Churn_final_pred'] = test_pred_final_df.Churn_Prob.map(lambda x : 1if x >0.42else0)
```

```
test_pred_final_df.head()
```

```
test_cm = metrics.confusion_matrix(test_pred_final_df.Churn_Actual, test_pred_final_df.Churn_final_pred)
test_cm
print('Test Sensitivity : ', test_cm[1,1] / (test_cm[1,1] + test_cm[1,0]))
print('Test Specificity : ', test_cm[0,0] / (test_cm[0,0] + test_cm[0,1]))
```

```
Test Sensitivity : 0.49911504424778763
Test Specificity :  0.8921188630490956
```

```
print(classification_report(test_pred_final_df.Churn_Actual, test_pred_final_df.Churn_final_pred))
# confusion matrix visualization
f, ax = plt.subplots(figsize = (5,5))sns.heatmap(cnf_matrix, annot =True, linewidths =0.5, color ="red", fmt =".0f", ax
=None)
plt.xlabel("Predicted value")
plt.ylabel("True value")
plt.title("Confusion Matrix of Logistic Regression on Test Dataset")
plt.show()
#plot ROC
draw_roc_curve(test_pred_final_df.Churn_Actual, test_pred_final_df.Churn_final_pred)
```

```
fromsklearn.ensembleimport RandomForestClassifier
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.7, test_size=0.3, random_state =100)model_rf = R
andomForestClassifier(n_estimators=1000 , oob_score =True, n_jobs =-1,
                              random_state =50, max_features ="auto",
                              max_leaf_nodes =30)model_rf.fit(X_train, y_train)
# Make predictionsy_
predicted = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, y_predicted))
importances = model_rf.feature_importances_weights = pd.Series(importances, index=X.columns.values)weights.sort
_values()[-10:].plot(kind ='barh')
```