https://www.kaggle.com/datasets/adammaus/predicting-churn-for-bank-customers the derivative of data set

```
#Part0 SETUP GOOGLE DRIVE ENVIRONMENT/DATA COLLECTION
```

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default
drive = GoogleDrive(gauth)
```

```
import numpy as np
import pandas as pd
churn_df = pd.read_csv('/content/Churn_Modelling.csv')
churn_df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	B
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8
2	3	15619304	Onio	502	France	Female	42	8	15
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12

Next steps: View recommended plots

```
#Part1:Data Exploration
```

```
churn_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
churn_df.nunique()
```

```
RowNumber          10000
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
```

```
IsActiveMember        2
EstimatedSalary    9999
Exited                2
dtype: int64
```

```
y = churn_df['Exited']
```

```
churn_df.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
churn_df[['CreditScore','Age','Tenure','NumOfProducts','Balance','EstimatedSalary']]
```

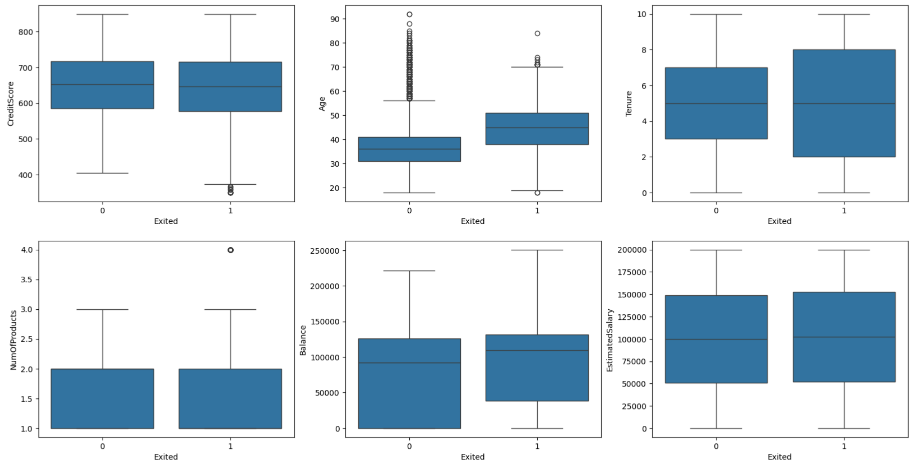
	CreditScore	Age	Tenure	NumOfProducts	Balance	EstimatedSalary
0	619	42	2	1	0.00	101348.88
1	608	41	1	1	83807.86	112542.58
2	502	42	8	3	159660.80	113931.57
3	699	39	1	2	0.00	93826.63
4	850	43	2	1	125510.82	79084.10
...	...	...	...	...	...	...
9995	771	39	5	2	0.00	96270.64
9996	516	35	10	1	57369.61	101699.77
9997	709	36	7	1	0.00	42085.58
9998	772	42	3	2	75075.31	92888.52
9999	792	28	4	1	130142.79	38190.78

10000 rows × 6 columns

```
import matplotlib.pyplot as plt
import seaborn as sns
```

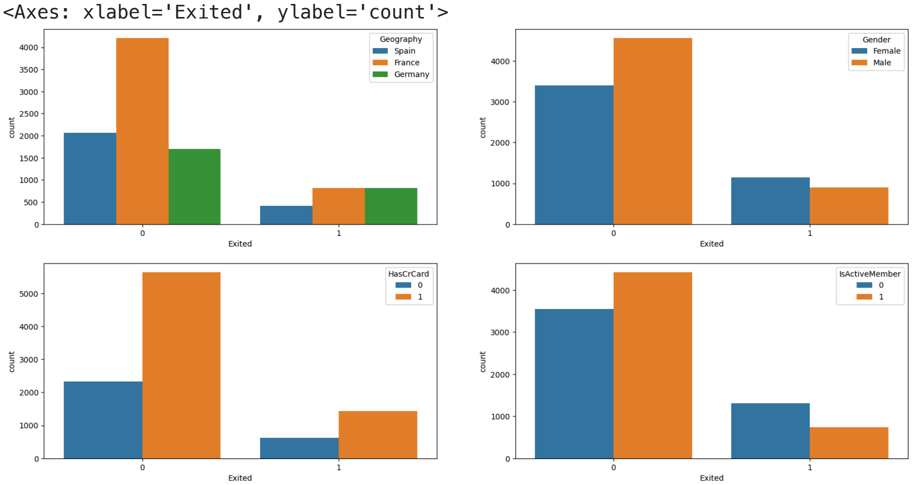
```
_,axss = plt.subplots(2,3, figsize=[20,10])
sns.boxplot(x='Exited', y ='CreditScore', data=churn_df, ax=axss[0][0])
sns.boxplot(x='Exited', y ='Age', data=churn_df, ax=axss[0][1])
sns.boxplot(x='Exited', y ='Tenure', data=churn_df, ax=axss[0][2])
sns.boxplot(x='Exited', y ='NumOfProducts', data=churn_df, ax=axss[1][0])
sns.boxplot(x='Exited', y ='Balance', data=churn_df, ax=axss[1][1])
sns.boxplot(x='Exited', y ='EstimatedSalary', data=churn_df, ax=axss[1][2])
```

```
<Axes: xlabel='Exited', ylabel='EstimatedSalary'>
```




```
_,axss = plt.subplots(2,2, figsize=[20,10])
sns.countplot(x='Exited', hue='Geography', data=churn_df, ax=axss[0][0])
sns.countplot(x='Exited', hue='Gender', data=churn_df, ax=axss[0][1])
sns.countplot(x='Exited', hue='HasCrCard', data=churn_df, ax=axss[1][0])
sns.countplot(x='Exited', hue='IsActiveMember', data=churn_df, ax=axss[1][1])
```

```
<Axes: xlabel='Exited', ylabel='count'>
```




```
#part2:Feature preprocessing
```

```
to_drop = ['RowNumber','CustomerId','Surname','Exited']
X = churn_df.drop(to_drop,axis = 1)
```

```
X.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCar
0	619	France	Female	42	2	0.00	1	
1	608	Spain	Female	41	1	83807.86	1	
2	502	France	Female	42	8	159660.80	3	
3	699	France	Female	39	1	0.00	2	

```
X.dtypes
```

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
dtype: object
```

```
cat_cols = X.columns[X.dtypes == 'object']
num_cols = X.columns[(X.dtypes == 'float64')|(X.dtypes == 'int64')]
```

```
num_cols
```

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary'],
      dtype='object')
```

```
cat_cols
```

```
Index(['Geography', 'Gender'], dtype='object')
```

```
from sklearn import model_selection#try again

# Reserve 25% for testing
# stratify example:
# 100 -> y: 80 '0', 20 '1' -> 4:1
# 80% training 64: '0', 16:'1' -> 4:1
# 20% testing  16:'0', 4: '1' -> 4:1
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.25, stratify = y, random_state = 1) #st

print('training data has ' + str(X_train.shape[0]) + ' observation with ' + str(X_train.shape[1]) + ' features')
print('test data has ' + str(X_test.shape[0]) + ' observation with ' + str(X_test.shape[1]) + ' features')
```

```
training data has 7500 observation with 10 features
test data has 2500 observation with 10 features
```

```
X_train.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCa
7971	633	Spain	Male	42	10	0.00	1	
9152	708	Germany	Female	23	4	71433.08	1	
6732	548	France	Female	37	9	0.00	2	
902	645	France	Female	48	7	90612.34	1	

Next steps: View recommended plots

```
from sklearn.preprocessing import OneHotEncoder
def OneHotEncoding(df,enc,categories):
  transformed = pd.DataFrame(enc.transform(df[categories]).toarray(), columns = enc.get_feature_names_out(categories))
  return pd.concat([df.reset_index(drop=True), transformed], axis=1).drop(categories, axis=1)

categories = ['Geography']
enc_ohe = OneHotEncoder()
enc_ohe.fit(X_train[categories])

X_train = OneHotEncoding(X_train, enc_ohe, categories)
X_test = OneHotEncoding(X_test, enc_ohe, categories)
```

```
X_train.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	633	Male	42	10	0.00	1	0	
1	708	Female	23	4	71433.08	1	1	
2	548	Female	37	9	0.00	2	0	
3	645	Female	48	7	90612.34	1	1	
4	729	Female	45	7	91091.06	2	1	

Next steps: View recommended plots

```
from sklearn.preprocessing import OrdinalEncoder

categories = ['Gender']
enc_oe = OrdinalEncoder()
enc_oe.fit(X_train[categories])

X_train[categories] = enc_oe.transform(X_train[categories])
X_test[categories] = enc_oe.transform(X_test[categories])
```

```
X_train.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	633	Male	42	10	0.00	1	0	
1	708	Female	23	4	71433.08	1	1	
2	548	Female	37	9	0.00	2	0	
3	645	Female	48	7	90612.34	1	1	
4	729	Female	45	7	91091.06	2	1	

Next steps: View recommended plots

```
#standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train[num_cols])

X_train[num_cols] = scaler.transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

```
X_train.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	-0.172985	1.0	0.289202	1.731199	-1.218916	-0.912769	-1.542199	
1	0.602407	0.0	-1.509319	-0.341156	-0.076977	-0.912769	0.648425	
2	-1.051762	0.0	-0.184093	1.385806	-1.218916	0.796109	-1.542199	
3	-0.048922	0.0	0.857156	0.695022	0.229625	-0.912769	0.648425	
4	0.819517	0.0	0.573179	0.695022	0.237278	0.796109	0.648425	

```
#Part3:Model Training and Result Evaluation
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
classifier_logistic = LogisticRegression()
classifier_KNN = KNeighborsClassifier()
classifier_RF = RandomForestClassifier()
```

```
classifier_logistic.fit(X_train,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
classifier_logistic.predict(X_test)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
classifier_logistic.score(X_test,y_test)
```

```
0.8088
```

```
#use gridsearch to find optimal hyperparameters
from sklearn.model_selection import GridSearchCV
def print_gird_search_metrics(gs):
  print('Best score:'+ str(gs.best_score_))
  print('Best parameters set:')
  best_parameters = gs.best_params_
  for param_name in sorted(best_parameters.keys()):
    print(param_name +':'+str(best_parameters[param_name]))
```

```
parameters = {
    'penalty':('l2','l1'),
    'C':(0.01, 0.05, 0.1, 0.2, 1)
}

Grid_LR = GridSearchCV(LogisticRegression(solver='liblinear'),parameters, cv = 5)
Grid_LR.fit(X_train, y_train)
```

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

```
print(Grid_LR)
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'),
             param_grid={'C': (0.01, 0.05, 0.1, 0.2, 1),
                         'penalty': ('l2', 'l1')})
```

```
best_LR_model = Grid_LR.best_estimator_
```

```
best_LR_model.predict(X_test)
```

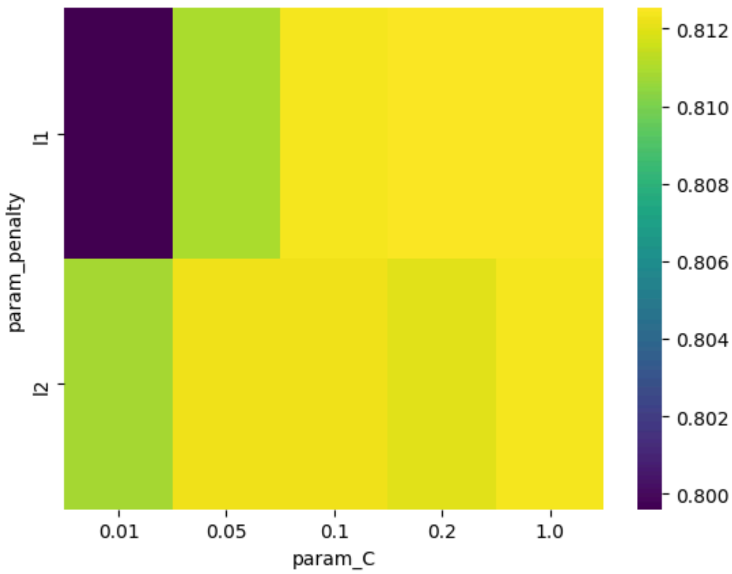
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
best_LR_model.score(X_test,y_test)
```

```
0.81
```

```
LR_models = pd.DataFrame(Grid_LR.cv_results_)
res = (LR_models.pivot(index='param_penalty', columns='param_C', values='mean_test_score'))
_ = sns.heatmap(res, cmap='viridis')
```

```
<ipython-input-52-324dd6f2eb90>:2: FutureWarning: In a future version, the Index
  res = (LR_models.pivot(index='param_penalty', columns='param_C', values='mean_
```




```
#find the optimal hyper parameters: KNN
```

```
parameters ={'n_neighbors':[1,3,5,7,9]}
Grid_KNN = GridSearchCV(KNeighborsClassifier(),parameters,cv=5)
Grid_KNN.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

```
best_KNN_model = Grid_KNN.best_estimator_
```

```
best_KNN_model.predict(X_test)
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
best_KNN_model.score(X_test,y_test)
```

```
0.8428
```

```
#find the optimal hyperparameters:Random Forest
```

```
parameters = {'n_estimators':[60,80,100],
              'max_depth':[1,5,10]}
Grid_RF = GridSearchCV(RandomForestClassifier(),parameters,cv=5)
Grid_RF.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
best_RF_model = Grid_RF.best_estimator_
```

```
best_RF_model
```

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=10, n_estimators=80)
```

```
best_RF_model.score(X_test,y_test)
```

```
0.8592
```

```
#model evaluation
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

# calculate accuracy, precision and recall, [[tn, fp],[]]
def cal_evaluation(classifier, cm):
    tn = cm[0][0]
    fp = cm[0][1]
    fn = cm[1][0]
    tp = cm[1][1]
    accuracy  = (tp + tn) / (tp + fp + fn + tn + 0.0)
    precision = tp / (tp + fp + 0.0)
    recall = tp / (tp + fn + 0.0)
    print (classifier)
    print ("Accuracy is: " + str(accuracy))
    print ("precision is: " + str(precision))
    print ("recall is: " + str(recall))
    print ()

# print out confusion matrices
def draw_confusion_matrices(confusion_matricies):
    class_names = ['Not','Churn']
    for cm in confusion_matrices:
        classifier, cm = cm[0], cm[1]
        cal_evaluation(classifier, cm)


confusion_matrices = [
    ("Random Forest", confusion_matrix(y_test,best_RF_model.predict(X_test))),
    ("Logistic Regression", confusion_matrix(y_test,best_LR_model.predict(X_test))),
    ("K nearest neighbor", confusion_matrix(y_test, best_KNN_model.predict(X_test)))
]
print(confusion_matrices)
draw_confusion_matrices(confusion_matrices)
```

```
[('Random Forest', array([[1941,   50],
       [ 302,  207]])), ('Logistic Regression', array([[1927,   64],
       [ 411,   98]])), ('K nearest neighbor', array([[1922,   69],
       [ 324,  185]]))]
Random Forest
Accuracy is: 0.8592
precision is: 0.8054474708171206
recall is: 0.4066797642436149

Logistic Regression
Accuracy is: 0.81
precision is: 0.6049382716049383
recall is: 0.1925343811394892

K nearest neighbor
Accuracy is: 0.8428
precision is: 0.7283464566929134
```

```
recall is: 0.36345776031434185
```

```
#model evaluation ROC/AUC
```

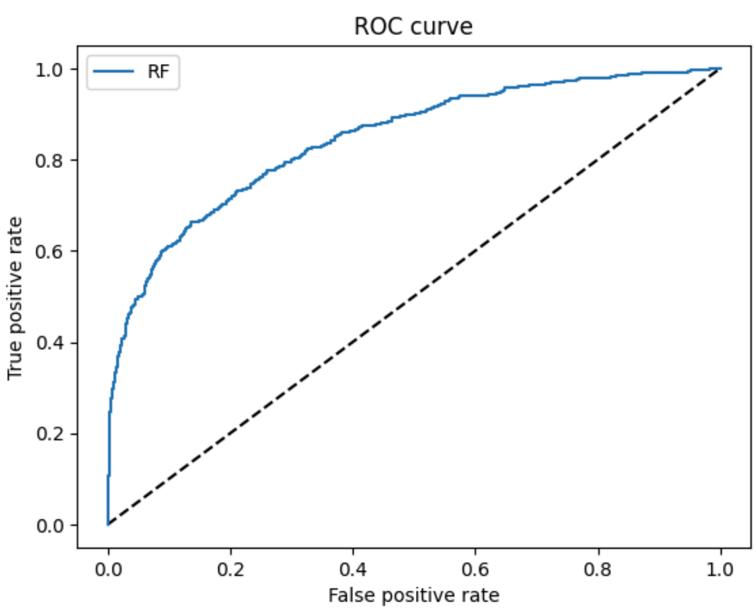
```
from sklearn.metrics import roc_curve
from sklearn import metrics

# Use predict_proba to get the probability results of Random Forest
y_pred_rf = best_RF_model.predict_proba(X_test)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)
```

```
best_RF_model.predict_proba(X_test)
```

```
array([[0.71332423, 0.28667577],
       [0.92374843, 0.07625157],
       [0.69229321, 0.30770679],
       ...,
       [0.86358856, 0.13641144],
       [0.92736437, 0.07263563],
       [0.87915577, 0.12084423]])
```

```
import matplotlib.pyplot as plt
plt.figure(1)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_rf,tpr_rf,label = 'RF')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc = 'best')
plt.show()
```




```
from sklearn import metrics
metrics.auc(fpr_rf,tpr_rf)
```

```
0.8442125122974801
```

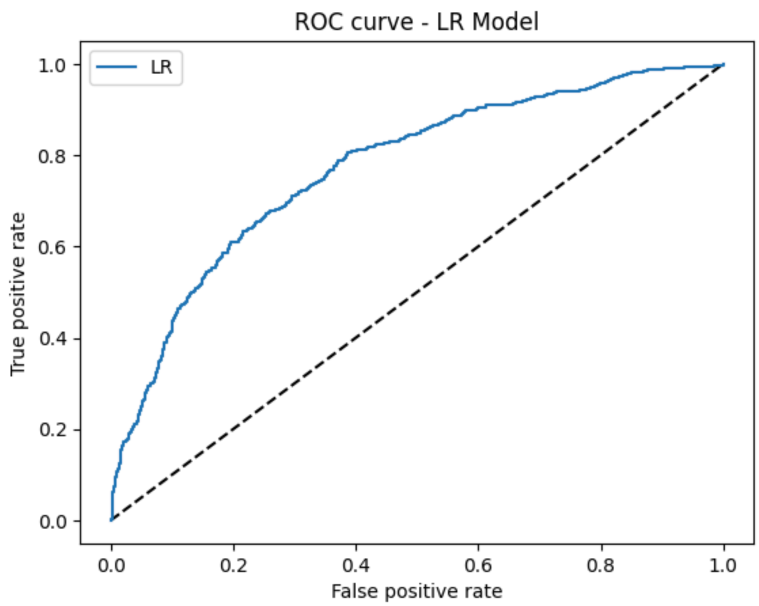
```
y_pred_lr = best_LR_model.predict_proba(X_test)[:, 1]
fpr_lr, tpr_lr, thresh = roc_curve(y_test, y_pred_lr)
```

```
best_LR_model.predict_proba(X_test)
```

```
array([[0.82265126, 0.17734874],
       [0.9292507 , 0.0707493 ],
       [0.85487963, 0.14512037],
       ...,
       [0.71584485, 0.28415515],
```

```
       [0.89060187, 0.10939813],
       [0.85476114, 0.14523886]])
```

```
plt.figure(1)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr_lr,tpr_lr,label ='LR')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve - LR Model')
plt.legend(loc='best')
plt.show()
```




```
metrics.auc(fpr_lr,tpr_lr)
```

```
0.7720784788917515
```

```
#model extra functionality
```

```
X_with_corr = X.copy()

X_with_corr = OneHotEncoding(X_with_corr,enc_ohe,['Geography'])
X_with_corr['Gender'] = enc_oe.transform(X_with_corr[['Gender']])
X_with_corr['SalaryInRMB'] = X_with_corr['EstimatedSalary']*6.4
X_with_corr.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
0	619	0.0	42	2	0.00	1	1	
1	608	0.0	41	1	83807.86	1	0	
2	502	0.0	42	8	159660.80	3	1	
3	699	0.0	39	1	0.00	2	0	
4	850	0.0	43	2	125510.82	1	1	

Next steps: View recommended plots

```
scaler = StandardScaler()
X_l1 = scaler.fit_transform(X_with_corr)
LRmodel_l1 = LogisticRegression(penalty = 'l1',C=0.04,solver = 'liblinear')
LRmodel_l1.fit(X_l1,y)
```

```
▾ LogisticRegression
LogisticRegression(C=0.04, penalty='l1', solver='liblinear')
```

```
np.random.seed()
scaler = StandardScaler()
X_l2 = scaler.fit_transform(X_with_corr)
LRmodel_l2 = LogisticRegression(penalty='l2',C=0.1,solver='liblinear',random_state=42 )
LRmodel_l2.fit(X_l1,y)
LRmodel_l2.coef_[0]
```

```
array([-0.06367794, -0.25913053,  0.75099022, -0.04520424,  0.16199493,
       -0.0585934 , -0.01993898, -0.52716425,  0.01367085, -0.12070832,
        0.22788348, -0.08904964,  0.01367085])
```

```
#random forest model-feature importance discussion
```

```
X_RF=X.copy()
X_RF = OneHotEncoding(X_RF,enc_ohe,['Geography'])
X_RF['Gender'] = enc_oe.transform(X_RF[['Gender']])
X_RF.head()
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Geography_France
0	619	0.0	42	2	0.00	1	1	1	101348.88	1.0
1	608	0.0	41	1	83807.86	1	0	1	112542.58	0.0
2	502	0.0	42	8	159660.80	3	1	0	113931.57	1.0
3	699	0.0	39	1	0.00	2	0	0	93826.63	1.0
4	850	0.0	43	2	125510.82	1	1	1	79084.10	0.0

Next steps: View recommended plots

```
forest =RandomForestClassifier()
forest.fit(X_RF,y)
importances = forest.feature_importances_
indices = np.argsort(importances)[::-1]
print('Feature importance ranking by Random Forest Model:')
for ind in range(X.shape[1]):
    print ("{0} : {1}".format(X_RF.columns[indices[ind]],round(importances[indices[ind]], 4)))
```