Bank Customer Churn Prediction

In this project, we use supervised learning models to identify customers who are likely to churn in the future. Furthermore, we will analyze top factors that influence user retention. [Dataset information](#).

Contents

Part 0: Setup Google Drive Environment / Data Collection

check this [link](#) for more info

```
# install pydrive to load data
!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)
```

```
# the same way we get id from last class
#https://drive.google.com/file/d/1szdCZ98EK59cfJ4jG03g1HOv_OhC1oyN/view?usp=sharing
id = "1szdCZ98EK59cfJ4jG03g1HOv_OhC1oyN"
file = drive.CreateFile({'id':id})
file.GetContentFile('bank_churn.csv')
```

```
import pandas as pd

df = pd.read_csv('bank_churn.csv')
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balan
0	1	15634602	Hargrave	619	France	Female	42	2	0.
1	2	15647311	Hill	608	Spain	Female	41	1	83807.
2	3	15619304	Onio	502	France	Female	42	8	159660.
3	4	15701354	Boni	699	France	Female	39	1	0.
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.

Part 1: Data Exploration

Part 1.1: Understand the Raw Dataset

```
import pandas as pd
import numpy as np

churn_df = pd.read_csv('bank_churn.csv')
```

```
churn_df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balan
0	1	15634602	Hargrave	619	France	Female	42	2	0.
1	2	15647311	Hill	608	Spain	Female	41	1	83807.
2	3	15619304	Onio	502	France	Female	42	8	159660.
3	4	15701354	Boni	699	France	Female	39	1	0.
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.

```
# check data info
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
```

```
# check the unique values for each column
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
```

```
# Get target variable
y = churn_df['Exited']
```

Part 1.2: Understand the features

```
# check missing values
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
```

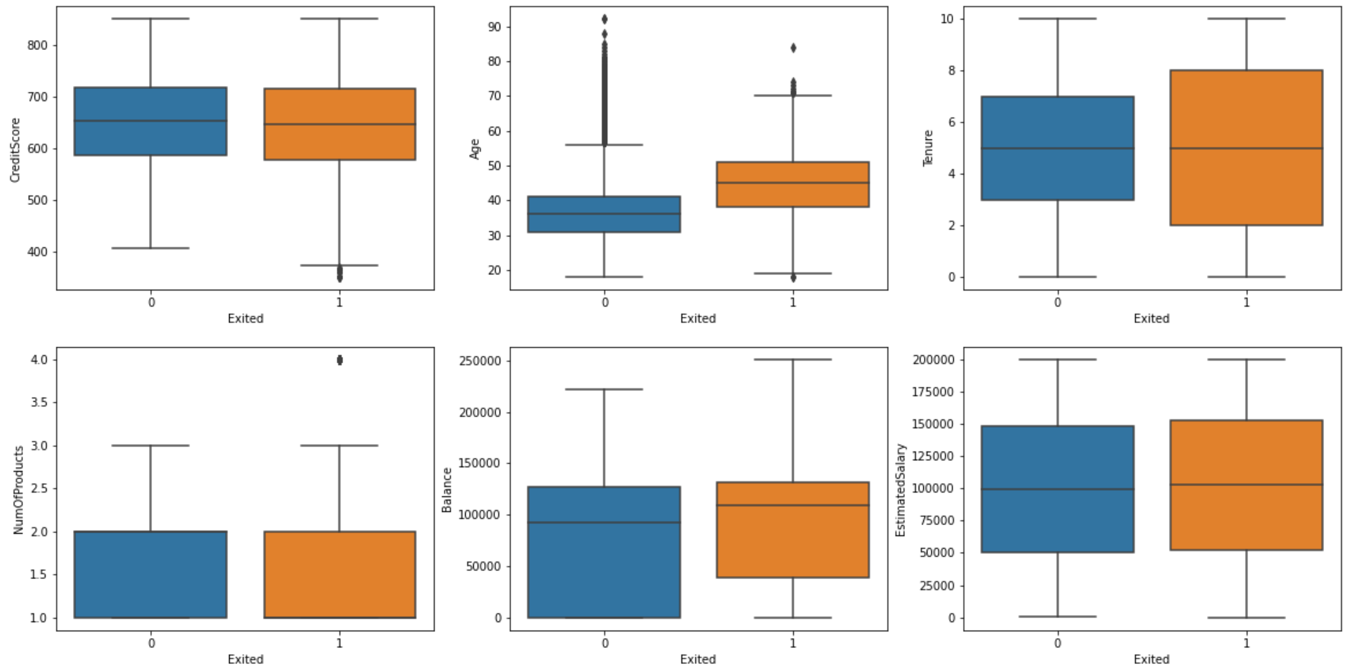
```
# understand Numerical feature
# discrete/continuous
# 'CreditScore', 'Age', 'Tenure', 'NumberOfProducts'
# 'Balance', 'EstimatedSalary'
churn_df[['CreditScore', 'Age', 'Tenure', 'NumOfProducts','Balance', 'EstimatedSalary']].d
```

	CreditScore	Age	Tenure	NumOfProducts	Balance	EstimatedS
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0
mean	650.528800	38.921800	5.012800	1.530200	76485.889288	100090.2
std	96.653299	10.487806	2.892174	0.581654	62397.405202	57510.4
min	350.000000	18.000000	0.000000	1.000000	0.000000	11.5
25%	584.000000	32.000000	3.000000	1.000000	0.000000	51002.1
50%	652.000000	37.000000	5.000000	1.000000	97198.540000	100193.9
75%	718.000000	44.000000	7.000000	2.000000	127644.240000	149388.2
max	850.000000	92.000000	10.000000	4.000000	250898.090000	199992.4

```
# check the feature distribution
# pandas.DataFrame.describe()
# boxplot, distplot, countplot
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# boxplot for numerical feature
_,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])
```

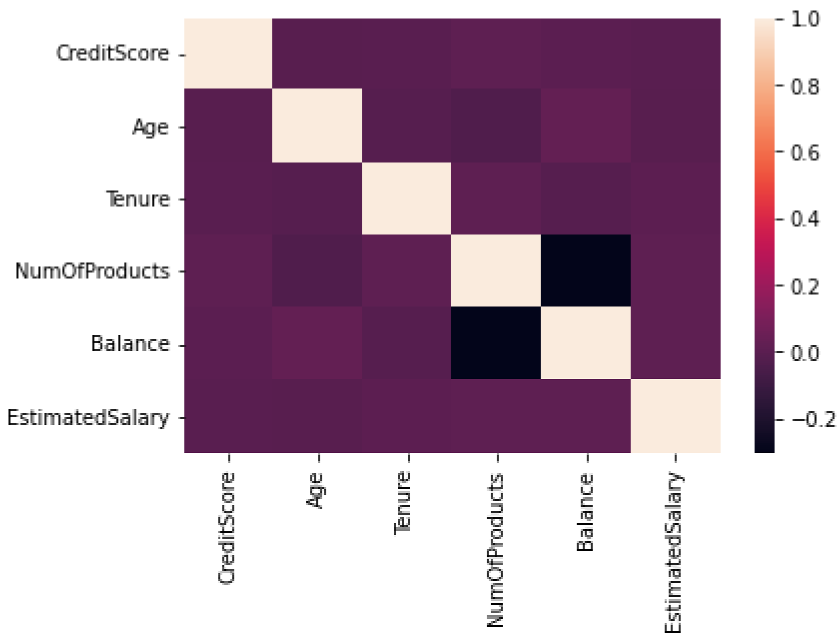
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3ab028c10>




```
# correlations between features
corr_score = churn_df[['CreditScore', 'Age', 'Tenure', 'NumOfProducts','Balance', 'Estimat

# show heapmap of correlations
sns.heatmap(corr_score)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb3aae68ad0>

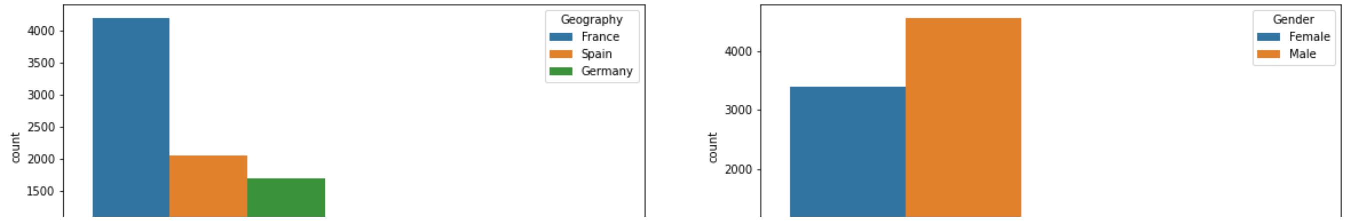


```
# check the actual values of correlations
corr_score
```

	CreditScore	Age	Tenure	NumOfProducts	Balance	EstimatedSal
CreditScore	1.000000	-0.003965	0.000842	0.012238	0.006268	-0.001
Age	-0.003965	1.000000	-0.009997	-0.030680	0.028308	-0.007
Tenure	0.000842	-0.009997	1.000000	0.013444	-0.012254	0.007
NumOfProducts	0.012238	-0.030680	0.013444	1.000000	-0.304180	0.014
Balance	0.006268	0.028308	-0.012254	-0.304180	1.000000	0.012
EstimatedSalary	-0.001384	-0.007201	0.007784	0.014204	0.012797	1.000

```
# understand categorical feature
# 'Geography', 'Gender'
# 'HasCrCard', 'IsActiveMember'
_,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])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb3aac7fcd0>
```



Part 2: Feature Preprocessing

```
# Get feature space by dropping useless feature
to_drop = ['RowNumber','CustomerId','Surname','Exited']
X = churn_df.drop(to_drop, axis=1)
```

```
X.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsA
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

```
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 == 'O']
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')
```

Split dataset

```
# Splite data into training and testing
# 100 -> 75:y=1, 25:y=0
# training(80): 60 y=1; 20 y=0
# testing(20):  15 y=1; 5 y=0

from sklearn import model_selection

# 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,

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

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

- 10000 -> 8000 '0' + 2000 '1'
- 25% test 75% training

---

without stratified sampling:

• extreme case:

1. testing: 2000 '1' + 500 '0'
2. training: 7500 '0'

---

with stratified sampling:

1. testing: 2000 '0' + 500 '1'
2. training: 6000 '0' + 1500 '1'

Read more for handling categorical feature, and there is an awesome package for encoding.

```
X_train.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	I
7971	633	Spain	Male	42	10	0.00	1	0	
9152	708	Germany	Female	23	4	71433.08	1	1	
6732	548	France	Female	37	9	0.00	2	0	
902	645	France	Female	48	7	90612.34	1	1	
2996	729	Spain	Female	45	7	91091.06	2	1	

```
# One hot encoding
# another way: get_dummies
from sklearn.preprocessing import OneHotEncoder

def OneHotEncoding(df, enc, categories):
  transformed = pd.DataFrame(enc.transform(df[categories]).toarray(), columns=enc.get_feat
  return pd.concat([df.reset_index(drop=True), transformed], axis=1).drop(categories, axis

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)

```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

```
X_train.head()
```

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

```
# Ordinal encoding
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	IsActiveMember
0	633	1.0	42	10	0.00	1	0	1
1	708	0.0	23	4	71433.08	1	1	0
2	548	0.0	37	9	0.00	2	0	0
3	645	0.0	48	7	90612.34	1	1	1
4	729	0.0	45	7	91091.06	2	1	0

Standardize/Normalize Data

```
# Scale the data, using standardization
# standardization (x-mean)/std
# normalization (x-x_min)/(x_max-x_min) ->[0,1]

# 1. speed up gradient descent
# 2. same scale
# 3. algorithm requirments

# for example, use training data to train the standardscaler to get mean and std
# apply mean and std to both training and testing data.
# fit_transform does the training and applying, transform only does applying.
# Because we can't use any info from test, and we need to do the same modification
# to testing data as well as training data

# https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html#sphx-g
# https://scikit-learn.org/stable/modules/preprocessing.html


# min-max example: (x-x_min)/(x_max-x_min)
# [1,2,3,4,5,6,100] -> fit(min:1, max:6) (scalar.min = 1, scalar.max = 6) -> transform [(1
# scalar.fit(train) -> min:1, max:100
# scalar.transform(apply to x) -> apply min:1, max:100 to X_train
# scalar.transform -> apply min:1, max:100 to X_test

```

```
# scalar.fit -> mean:1, std:100
# scalar.transform -> apply mean:1, std:100 to X_train
# scalar.transform -> apply mean:1, std:100 to X_test

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	IsActiveMember
0	633	1.0	42	10	0.00	1	0	1
1	708	0.0	23	4	71433.08	1	1	0
2	548	0.0	37	9	0.00	2	0	0
3	645	0.0	48	7	90612.34	1	1	1
4	729	0.0	45	7	91091.06	2	1	0

Part 3: Model Training and Result Evaluation

Part 3.1: Model Training

```
#@title build models
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# Logistic Regression
classifier_logistic = LogisticRegression()

# K Nearest Neighbors
classifier_KNN = KNeighborsClassifier()

# Random Forest
classifier_RF = RandomForestClassifier()
```

build models

```
# Train the model
classifier_logistic.fit(X_train, y_train)
```

```
LogisticRegression()
```

```
# Prediction of test data
classifier_logistic.predict(X_test)
```

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

```
# Accuracy of test data
classifier_logistic.score(X_test, y_test)
```

```
0.7936
```

```
# Use 5-fold Cross Validation to get the accuracy for different models
model_names = ['Logistic Regression','KNN','Random Forest']
model_list = [classifier_logistic, classifier_KNN, classifier_RF]
count = 0

for classifier in model_list:
    cv_score = model_selection.cross_val_score(classifier, X_train, y_train, cv=5)
    print(cv_score)
    print('Model accuracy of ' + model_names[count] + ' is ' + str(cv_score.mean()))
    count += 1
```

```
[0.78533333 0.78933333 0.78866667 0.792      0.78933333]
Model accuracy of Logistic Regression is 0.7889333333333333
[0.76333333 0.772      0.75733333 0.76466667 0.758     ]
Model accuracy of KNN is 0.7630666666666668
[0.88133333 0.864      0.85266667 0.85533333 0.86533333]
Model accuracy of Random Forest is 0.8637333333333332
```

(Optional) Part 3.2: Use Grid Search to Find Optimal Hyperparameters

alternative: random search

```
#Loss/cost function --> (wx + b - y) ^2 + ƛ * |w| --> ƛ is a hyperparameter
```

```
from sklearn.model_selection import GridSearchCV

# helper function for printing out grid search results
def print_grid_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]))
```

Part 3.2.1: Find Optimal Hyperparameters - LogisticRegression

```
# Possible hyperparamter options for Logistic Regression Regularization
# Penalty is choosed from L1 or L2
# C is the 1/lambda value(weight) for L1 and L2
# solver: algorithm to find the weights that minimize the cost function

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

```
# the best hyperparameter combination
# C = 1/lambda
print_grid_search_metrics(Grid_LR)
```

```
Best score: 0.8130666666666666
Best parameters set:
C:0.05
penalty:l1
```

```
# best model
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)
```

```
0.8084
```

```
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')
```

Part 3.2.2: Find Optimal Hyperparameters: KNN

```
# Possible hyperparamter options for KNN
# Choose k
parameters = {
    'n_neighbors':[1,3,5,7,9]
}
Grid_KNN = GridSearchCV(KNeighborsClassifier(),parameters, cv=5)
Grid_KNN.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 3, 5, 7, 9]})
```

```
# best k
print_grid_search_metrics(Grid_KNN)
```

```
Best score: 0.7856
Best parameters set:
n_neighbors:9
```

```
best_KNN_model = Grid_KNN.best_estimator_
```

Part 3.2.3: Find Optimal Hyperparameters: Random Forest

```
# Possible hyperparamter options for Random Forest
# Choose the number of trees
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(cv=5, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [1, 5, 10],
                         'n_estimators': [60, 80, 100]})
```

```
# best number of tress
print_grid_search_metrics(Grid_RF)
```

```
Best score: 0.8667999999999999
Best parameters set:
max_depth:10
n_estimators:100
```

```
# best random forest
best_RF_model = Grid_RF.best_estimator_
```

```
best_RF_model
```

```
RandomForestClassifier(max_depth=10)
```

Part 3.3: Model Evaluation - Confusion Matrix (Precision, Recall, Accuracy)

class of interest as positive

TP: correctly labeled real churn

Precision(PPV, positive predictive value): tp / (tp + fp); Total number of true predictive churn divided by the total number of predictive churn; High Precision means low fp, not many return users were predicted as churn users.

Recall(sensitivity, hit rate, true positive rate): tp / (tp + fn) Predict most postive or churn user correctly. High recall means low fn, not many churn users were predicted as return users.

```
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 matrix, accuracy, precison and recall for random forest and logistic regressio
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)))
]

draw_confusion_matrices(confusion_matrices)
```

```
Random Forest
Accuracy is: 0.8604
precision is: 0.8007518796992481
recall is: 0.41846758349705304

Logistic Regression
Accuracy is: 0.8084
precision is: 0.6056338028169014
recall is: 0.16895874263261296

K nearest neighbor
Accuracy is: 0.776
precision is: 0.2
recall is: 0.03339882121807466
```

Part 3.4: Model Evaluation - ROC & AUC

RandomForestClassifier, KNeighborsClassifier and LogisticRegression have predict_prob() function

Part 3.4.1: ROC of RF Model

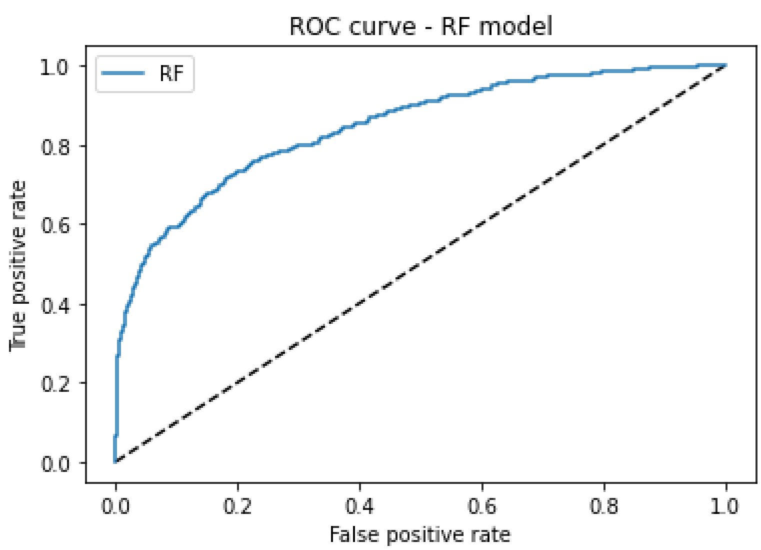
```
from sklearn.metrics import roc_curve
from sklearn import metrics

# 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([[0.69937921, 0.30062079],
       [0.95239486, 0.04760514],
       [0.74105858, 0.25894142],
       ...,
       [0.85967291, 0.14032709],
       [0.95083241, 0.04916759],
       [0.90647722, 0.09352278]])
```

```
# ROC curve of Random Forest result
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 - RF model')
plt.legend(loc='best')
plt.show()
```




```
from sklearn import metrics

# AUC score
metrics.auc(fpr_rf,tpr_rf)
```

```
0.8457044914295074
```

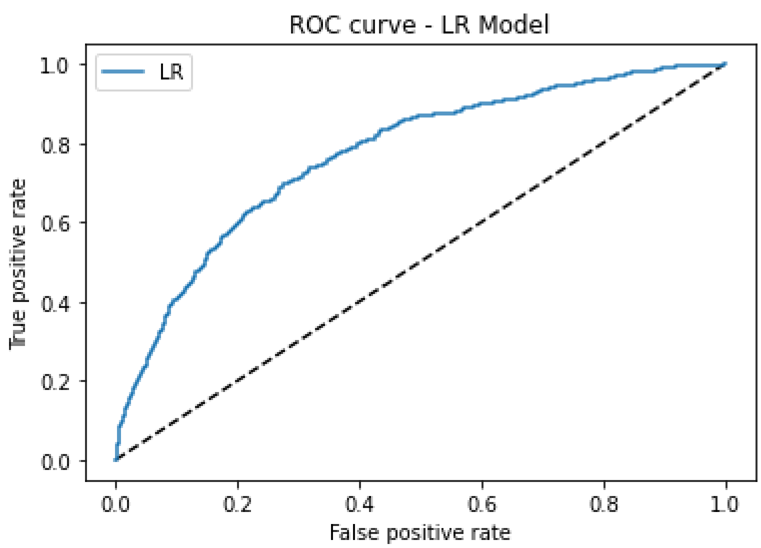
Part 3.4.1: ROC of LR Model

```
# Use predict_proba to get the probability results of Logistic Regression
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([[0.83732252, 0.16267748],
       [0.91667203, 0.08332797],
       [0.85713621, 0.14286379],
       ...,
       [0.72754602, 0.27245398],
       [0.88726134, 0.11273866],
       [0.86330938, 0.13669062]])
```

```
# ROC Curve
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()
```




```
# AUC score
metrics.auc(fpr_lr,tpr_lr)
```

```
0.7703625055381831
```

Part 4: Model Extra Functionality

Part 4.1: Logistic Regression Model

The corelated features that we are interested in

```
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()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
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	

```
# add L1 regularization to logistic regression
# check the coef for feature selection
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)

indices = np.argsort(abs(LRmodel_l1.coef_[0]))[::-1]

print ("Logistic Regression (L1) Coefficients")
for ind in range(X_with_corr.shape[1]):
  print ("{0} : {1}".format(X_with_corr.columns[indices[ind]],round(LRmodel_l1.coef_[0][in
```

```
Logistic Regression (L1) Coefficients
Age : 0.7307
IsActiveMember : -0.5046
Geography_Germany : 0.3121
Gender : -0.2409
Balance : 0.151
CreditScore : -0.0457
NumOfProducts : -0.0438
Tenure : -0.0271
EstimatedSalary : 0.0055
Geography_France : -0.0043
SalaryInRMB : 0.0037
HasCrCard : -0.0022
Geography_Spain : 0.0
```

```
# add L2 regularization to logistic regression
# check the coef for feature selection
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_l2, y)
LRmodel_l2.coef_[0]

indices = np.argsort(abs(LRmodel_l2.coef_[0]))[::-1]

print ("Logistic Regression (L2) Coefficients")
for ind in range(X_with_corr.shape[1]):
  print ("{0} : {1}".format(X_with_corr.columns[indices[ind]],round(LRmodel_l2.coef_[0][in
```

```
Logistic Regression (L2) Coefficients
Age : 0.751
IsActiveMember : -0.5272
Gender : -0.2591
Geography_Germany : 0.2279
Balance : 0.162
Geography_France : -0.1207
Geography_Spain : -0.089
CreditScore : -0.0637
NumOfProducts : -0.0586
Tenure : -0.0452
HasCrCard : -0.0199
SalaryInRMB : 0.0137
EstimatedSalary : 0.0137
```

Part 4.2: 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()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

CreditScore Gender Age Tenure Balance NumOfProducts HasCrCard IsActiveMember

```
# check feature importance of random forest for feature selection
forest = RandomForestClassifier()
forest.fit(X_RF, y)

importances = forest.feature_importances_

indices = np.argsort(importances)[::-1]

# Print the feature ranking
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)
```

```
Feature importance ranking by Random Forest Model:
Age : 0.2372
EstimatedSalary : 0.1483
Balance : 0.1429
CreditScore : 0.1426
NumOfProducts : 0.1302
Tenure : 0.0822
IsActiveMember : 0.0396
Geography_Germany : 0.0217
HasCrCard : 0.0186
Gender : 0.0182
```

✓ 0s completed at 10:30 AM