Bank Customer Churn Prediction

1. Data Exploration

Load data

In [1]:
```
import pandas as pd
import numpy as np
```

In [4]:
```
df=pd.read_csv('Churn_Modelling.csv')
```

In [6]:
```
df.head()
```

Out[6]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

In [8]:
```
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
```

In [15]:
```
# Check numerical features
df.describe()
```

Out[15]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfPro
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.0

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfPro
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.5
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.5
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.0
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.0
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.0
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.0
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.0

In [7]:
```
# check number of nique values for each column
df.nunique()
```

Out[7]:
```
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
```

In [9]:
```
# get target variable
y=df['Exited']
```

In [10]:
```
# get descriptive statistics of y
y.describe()
```

Out[10]:
```
count    10000.000000
mean         0.203700
std          0.402769
min          0.000000
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000
Name: Exited, dtype: float64
```

In [13]:
```
# check proportion of y=1 (same as mean shown above)
print(round(y.sum()/y.shape[0]*100,2), '%')
```

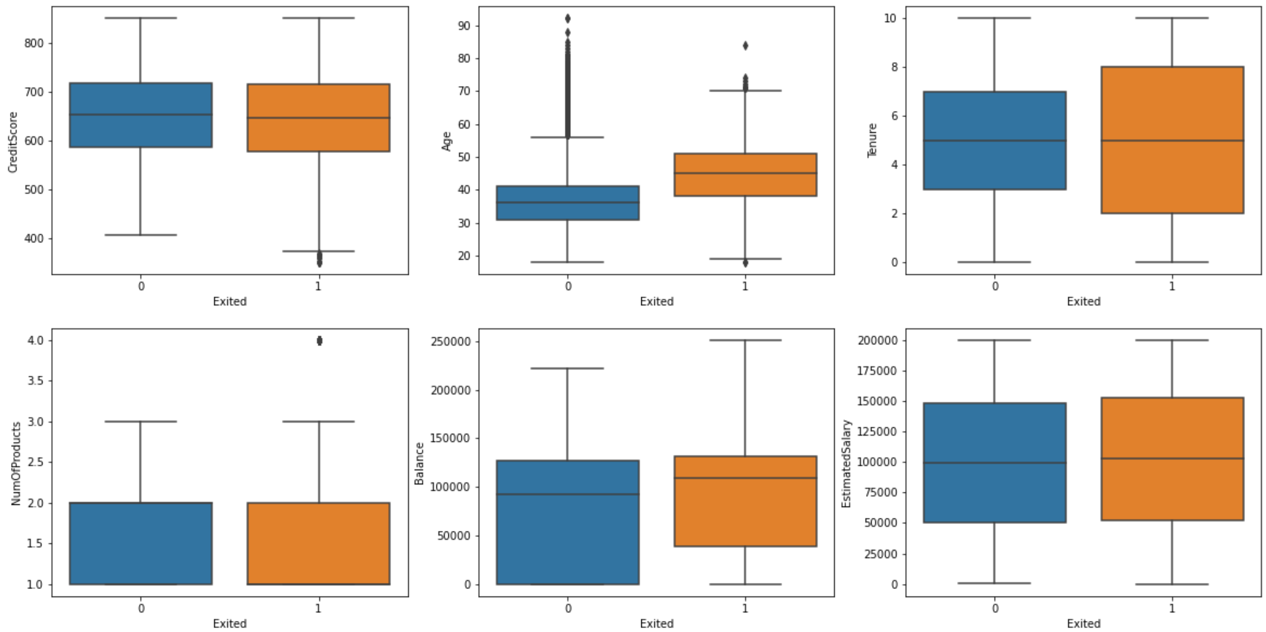
```
20.37 %
```

Explore features

In [16]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

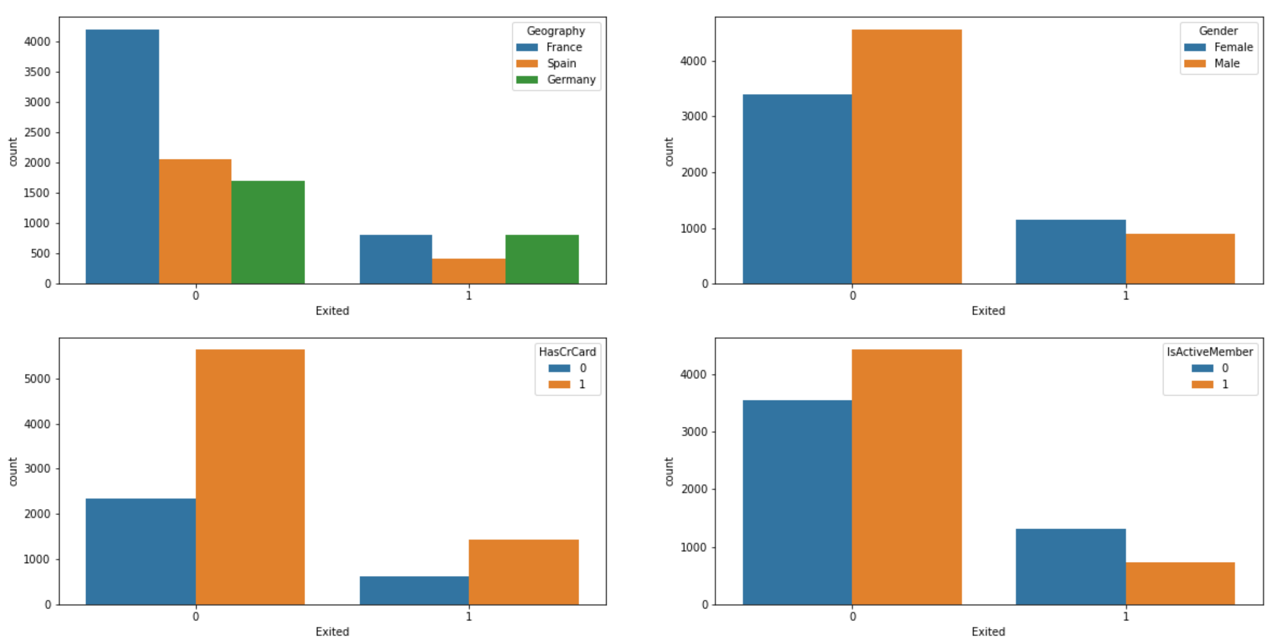
In [18]:
```
# explore numerical features: 'CreditScore', 'Age', 'Tenure', 'NumberOfProducts', 'Bala
_,axss = plt.subplots(2,3, figsize=[20,10])
sns.boxplot(x='Exited', y ='CreditScore', data=df, ax=axss[0][0])
sns.boxplot(x='Exited', y ='Age', data=df, ax=axss[0][1])
sns.boxplot(x='Exited', y ='Tenure', data=df, ax=axss[0][2])
sns.boxplot(x='Exited', y ='NumOfProducts', data=df, ax=axss[1][0])
sns.boxplot(x='Exited', y ='Balance', data=df, ax=axss[1][1])
sns.boxplot(x='Exited', y ='EstimatedSalary', data=df, ax=axss[1][2])
```

Out[18]: <AxesSubplot:xlabel='Exited', ylabel='EstimatedSalary'>



In [22]:
```
# explore categorical feature: 'Geography', 'Gender', 'HasCrCard', 'IsActiveMember'
_,axss = plt.subplots(2,2, figsize=[20,10])
sns.countplot(x='Exited', hue='Geography', data=df, ax=axss[0][0])
sns.countplot(x='Exited', hue='Gender', data=df, ax=axss[0][1])
sns.countplot(x='Exited', hue='HasCrCard', data=df, ax=axss[1][0])
sns.countplot(x='Exited', hue='IsActiveMember', data=df, ax=axss[1][1])
```

Out[22]: <AxesSubplot:xlabel='Exited', ylabel='count'>



In [24]:
```
# correlations between features
corr_score = df[['CreditScore', 'Age', 'Tenure', 'NumOfProducts','Balance', 'EstimatedS
corr_score
```

Out[24]:

	CreditScore	Age	Tenure	NumOfProducts	Balance	EstimatedSalary
CreditScore	1.000000	-0.003965	0.000842	0.012238	0.006268	-0.001384
Age	-0.003965	1.000000	-0.009997	-0.030680	0.028308	-0.007201
Tenure	0.000842	-0.009997	1.000000	0.013444	-0.012254	0.007784
NumOfProducts	0.012238	-0.030680	0.013444	1.000000	-0.304180	0.014204
Balance	0.006268	0.028308	-0.012254	-0.304180	1.000000	0.012797
EstimatedSalary	-0.001384	-0.007201	0.007784	0.014204	0.012797	1.000000

In [25]:
```
# show heapmap of correlations
sns.heatmap(corr_score)
```

Out[25]: <AxesSubplot:>

No two features are significantly correlated given the plot.

2. Feature Preprocessing

In [26]:
```
# ordinal encoding
df['Gender'] = (df['Gender'] == 'Female')
```

In [27]:
```
df.shape
```

Out[27]: (10000, 14)

In [29]:
```
df = pd.get_dummies(df, columns=['Geography'])
```

In [30]:
```
df.head(10)
```

Out[30]:

	RowNumber	CustomerId	Surname	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts
0	1	15634602	Hargrave	619	True	42	2	0.00	1
1	2	15647311	Hill	608	True	41	1	83807.86	1
2	3	15619304	Onio	502	True	42	8	159660.80	3
3	4	15701354	Boni	699	True	39	1	0.00	2
4	5	15737888	Mitchell	850	True	43	2	125510.82	1
5	6	15574012	Chu	645	False	44	8	113755.78	2
6	7	15592531	Bartlett	822	False	50	7	0.00	2
7	8	15656148	Obinna	376	True	29	4	115046.74	4
8	9	15792365	He	501	False	44	4	142051.07	2
9	10	15592389	H?	684	False	27	2	134603.88	1

In [32]:

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

In [33]:

```python
x.head()
```

Out[33]:

	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimate
0	619	True	42	2	0.00	1	1	1	10
1	608	True	41	1	83807.86	1	0	1	11
2	502	True	42	8	159660.80	3	1	0	11
3	699	True	39	1	0.00	2	0	0	9
4	850	True	43	2	125510.82	1	1	1	7

In [34]:

```python
y.head()
```

Out[34]:

```
0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64
```

3. Model Training and Result Evaluation

Split dataset

In [35]:

```python
from sklearn import model_selection
```

In [36]:

```python
# Reserve 25% for testing
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, test_size=0.2
```

```
print('training data has ' + str(x_train.shape[0]) + ' observation with ' + str(x_train
print('test data has ' + str(x_test.shape[0]) + ' observation with ' + str(x_test.shape
```

```
training data has 7500 observation with 12 features
test data has 2500 observation with 12 features
```

In [37]:
```
# Scale the data, using standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

Model Training

In [38]:
```
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()
```

In [51]:
```
# Train the model, predict on test data, and get accuracy of test data
model_names = ['Logistic Regression','KNN','Random Forest']
model_list = [classifier_logistic, classifier_KNN, classifier_RF]
count = 0

for classifier in model_list:
    classifier.fit(x_train, y_train)
    classifier.predict(x_test)
    model_score=classifier.score(x_test, y_test)
    print('Accuracy of test data of ' + model_names[count]+':  '+str(model_score))
    count +=1
```

```
Accuracy of test data of Logistic Regression:  0.808
Accuracy of test data of KNN:  0.8268
Accuracy of test data of Random Forest:  0.8596
```

In [52]:
```
# 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
```

```
[0.81933333 0.80666667 0.80666667 0.80933333 0.82      ]
Model accuracy of Logistic Regression is 0.8124
[0.82533333 0.836      0.814      0.824      0.832     ]
Model accuracy of KNN is 0.8262666666666666
```

```
[0.876      0.864      0.856      0.85333333 0.868     ]
Model accuracy of Random Forest is 0.8634666666666668
```

Use Grid Search to Find Optimal Hyperparameters

In [53]:
```
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]))
```

Optimal Hyperparameters - LogisticRegression

In [55]:
```
# Possible hyperparamter options for Logistic Regression Regularization
# Penalty is choosed from L1 or L2
# C is the lambda value(weight) for L1 and L2

# ('l1', 1) ('l1', 5) ('l1', 10) ('l2', 1) ('l2', 5) ('l2', 10)
parameters = {
    'penalty':('l1', 'l2'),
    'C':(0.01, 1, 5, 10, 100)
}
Grid_LR = GridSearchCV(LogisticRegression(solver='liblinear'),parameters, cv=5)
Grid_LR.fit(x_train, y_train)
```

Out[55]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(solver='liblinear'),
             param_grid={'C': (0.01, 1, 5, 10, 100), 'penalty': ('l1', 'l2')})
```

In [56]:
```
# the best hyperparameter combination
# C = 1/lambda
print_grid_search_metrics(Grid_LR)
```

```
Best score: 0.8124
Best parameters set:
C:1
penalty:l2
```

In [57]:
```
# best model
best_LR_model = Grid_LR.best_estimator_
```

In [66]:
```
best_LR_model
```

Out[66]:
```
LogisticRegression(C=1, solver='liblinear')
```

Optimal Hyperparameters - KNN

In [58]:
```
# 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)
```

Out[58]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 3, 5, 7, 9]})
```

In [59]:
```
# best k
print_grid_search_metrics(Grid_KNN)
```

```
Best score: 0.8322666666666667
Best parameters set:
n_neighbors:9
```

In [60]:
```
best_KNN_model = Grid_KNN.best_estimator_
```

In [65]:
```
best_KNN_model
```

Out[65]:
```
KNeighborsClassifier(n_neighbors=9)
```

Optimal Hyperparameters - Random Forest

In [61]:
```
# Possible hyperparamter options for Random Forest
# Choose the number of trees
parameters = {
    'n_estimators' : [40,60,80]
}
Grid_RF = GridSearchCV(RandomForestClassifier(),parameters, cv=5)
Grid_RF.fit(x_train, y_train)
```

Out[61]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(),
             param_grid={'n_estimators': [40, 60, 80]})
```

In [62]:
```
# best number of tress
print_grid_search_metrics(Grid_RF)
```

```
Best score: 0.8634666666666666
Best parameters set:
n_estimators:80
```

In [63]:
```
# best random forest
best_RF_model = Grid_RF.best_estimator_
```

In [64]:
```
best_RF_model
```

Out[64]:
```
RandomForestClassifier(n_estimators=80)
```

Model Evaluation - Confusion Matrix

In [74]:
```
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
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):
    for cm in confusion_matrices:
        classifier, cm = cm[0], cm[1]
        cal_evaluation(classifier, cm)
```

In [75]:

```
confusion_matrices = [
    ("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))),
    ("Random Forest", confusion_matrix(y_test, best_RF_model.predict(x_test)))
]

draw_confusion_matrices(confusion_matrices)
```

```
Logistic Regression
Accuracy is: 0.808
precision is: 0.5857988165680473
recall is: 0.1944990176817289

K nearest neighbor
Accuracy is: 0.8336
precision is: 0.6837944664031621
recall is: 0.33988212180746563

Random Forest
Accuracy is: 0.8544
precision is: 0.7508650519031141
recall is: 0.4263261296660118
```

Feature Selection

Feature Selection - Logistic Regression

In [84]:

```
# add L1 regularization to logistic regression
# check the coef for feature selection
scaler = StandardScaler()
X_l1 = scaler.fit_transform(x)
LRmodel_l1 = LogisticRegression(penalty="l1", C = 0.07, 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.shape[1]):
    print("{0} : {1}".format(x.columns[indices[ind]],round(LRmodel_l1.coef_[0][indices[
```

```
Logistic Regression (L1) Coefficients
Age : 0.744
IsActiveMember : -0.5184
Geography_Germany : 0.3157
Gender : 0.2503
Balance : 0.1566
CreditScore : -0.0537
NumOfProducts : -0.0503
Tenure : -0.0351
EstimatedSalary : 0.017
Geography_France : -0.0099
```

```
HasCrCard : -0.0099
Geography_Spain : 0.0
```

In [86]:

```
# add L2 regularization to logistic regression
# check the coef for feature selection
np.random.seed()
scaler = StandardScaler()
X_l2 = scaler.fit_transform(x)
LRmodel_l2 = LogisticRegression(penalty="l2", C = 0.1, solver='liblinear', random_state
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.shape[1]):
    print ("{0} : {1}".format(x.columns[indices[ind]],round(LRmodel_l2.coef_[0][indices
```

```
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
EstimatedSalary : 0.0272
HasCrCard : -0.0199
```

Feature Selection - Random Forest

In [87]:

```
# check feature importance of random forest for feature selection
forest = RandomForestClassifier()
forest.fit(x, 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.columns[indices[ind]],round(importances[indices[ind]],
```

```
Feature importance ranking by Random Forest Model:
Age : 0.2411
EstimatedSalary : 0.1464
CreditScore : 0.1433
Balance : 0.1428
NumOfProducts : 0.128
Tenure : 0.0818
IsActiveMember : 0.0395
Geography_Germany : 0.0224
Gender : 0.0186
HasCrCard : 0.0184
Geography_France : 0.0097
Geography_Spain : 0.008
```

In [ ]: