Supervised Learning: Classification

Section 1. Main objective of the analysis that specifies whether your model will be focused on prediction or interpretation.

Main objective of the analysis is prediction of Bank Customer Churn and finding the most important feature on determining the quality. This analysis will help to producing Customer Churn with fine quality.

Section 2. Brief description of the data set you chose and a summary of its attributes.

Bank Customer Churn Dataset

Dataset is related to ABC Multistate Bank. Dataset link from UCI machine learning repository:

https://www.kaggle.com/datasets/gauravtopre/bank-customer-churn-dataset

Input Variables:

- customer_id
- credit_score
- country
- gender
- age
- tenure
- balance
- products_number
- credit_card
- active_member
- estimated_salary

Output

- churn

Section 3. Data exploration and actions taken for data cleaning and feature engineering.

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
%matplotlib inline
```
✓ 5.6s

```python
data = pd.read_csv("Bank Customer Churn Prediction.csv")
data.head(7)
```
✓ 0.1s

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	15634602	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0
5	15574012	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
6	15592531	822	France	Male	50	7	0.00	2	1	1	10062.80	0

Checking for missing value.

```python
data.info()
```
✓ 0.5s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       10000 non-null  int64
 1   credit_score      10000 non-null  int64
 2   country           10000 non-null  object
 3   gender            10000 non-null  object
 4   age               10000 non-null  int64
 5   tenure            10000 non-null  int64
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64
 8   credit_card       10000 non-null  int64
 9   active_member     10000 non-null  int64
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64
dtypes: float64(2), int64(8), object(2)
memory usage: 937.6+ KB
```

Checking for duplicated column.

```python
data.duplicated().value_counts()
```
✓ 0.4s

```
False    10000
dtype: int64
```

 # Dataset is not including any unique column. So, I don’t delete any column.

Checking for value in column which is an object type.

```
data.country.value_counts()
✓ 0.3s

France     5014
Germany    2509
Spain      2477
Name: country, dtype: int64
```

```
data.gender.value_counts()
✓ 0.4s

Male      5457
Female    4543
Name: gender, dtype: int64
```

```
LE = LabelEncoder()
data['country'] = LE.fit_transform(data['country'])
data.country.value_counts()
✓ 0.5s

0    5014
1    2509
2    2477
Name: country, dtype: int64
```

```
data['gender'] = LE.fit_transform(data['gender'])
data.gender.value_counts()
✓ 0.5s

1    5457
0    4543
Name: gender, dtype: int64
```

 # Encoded the object value with LabelEncoder from sklearn.preprocessing

```
data.info()
✓ 0.5s

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customer_id       10000 non-null  int64
 1   credit_score      10000 non-null  int64
 2   country           10000 non-null  int32
 3   gender            10000 non-null  int32
 4   age               10000 non-null  int64
 5   tenure            10000 non-null  int64
 6   balance           10000 non-null  float64
 7   products_number   10000 non-null  int64
 8   credit_card       10000 non-null  int64
 9   active_member     10000 non-null  int64
 10  estimated_salary  10000 non-null  float64
 11  churn             10000 non-null  int64
dtypes: float64(2), int32(2), int64(8)
memory usage: 859.5 KB
```

Checking for statistics value of dataset.

```
data.describe()
✓ 0.7s
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	0.746300	0.545700	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	7.193619e+04	96.653299	0.827529	0.497932	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.556570e+07	350.000000	0.000000	0.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	1.562853e+07	584.000000	0.000000	0.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	1.569074e+07	652.000000	0.000000	1.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	1.575323e+07	718.000000	1.000000	1.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	1.581569e+07	850.000000	2.000000	1.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

Drop some useless column.

```
data2 = data.drop(columns='customer_id', axis = 1)
data2.head()
```

✓ 0.4s

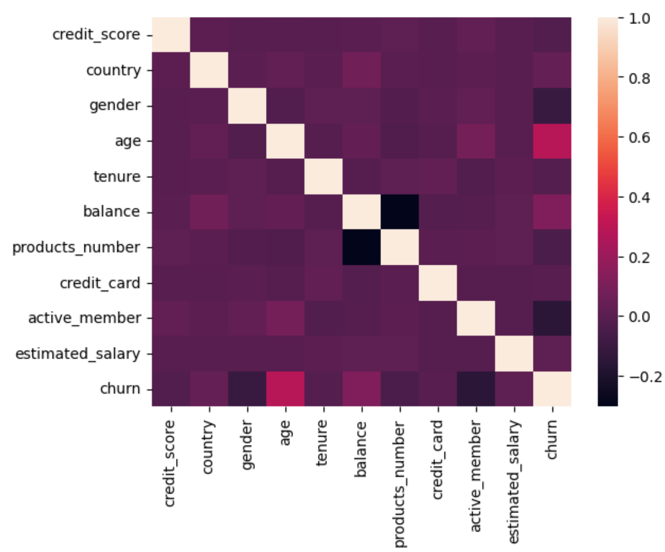
	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	850	2	0	43	2	125510.82	1	1	1	79084.10	0

Analyzing features importance

```
corr_value = data2.corr()
print(corr_value.abs()["churn"].nlargest(10))
sns.heatmap(corr_value)
```

✓ 0.3s

```
churn               1.000000
age                 0.285323
active_member       0.156128
balance             0.118533
gender              0.106512
products_number     0.047820
country             0.035943
credit_score        0.027094
tenure              0.014001
estimated_salary    0.012097
Name: churn, dtype: float64
```



Summary of data exploration and actions taken for data cleaning and feature engineering.

After the data exploration there's no need to any action for data cleaning. The dataset has no missing value. Besides that, there are 2 columns which are object value, so I use a Label Encoder to encode this value to continuous value. I will do scaling the values.

Section 4. Scaling, Train-Test Split and Classification

Train-Test Split and Scaling

```
X_train,X_test,y_train,y_test = train_test_split(data2[feature_cols],data2['churn'], test_size= 0.2)
```
✓ 0.3s

```
X_train = StandardScaler().fit_transform(X_train)
X_test =StandardScaler().fit_transform(X_test)
```
✓ 0.6s

Classification

```
LR = LogisticRegression()
LR.fit(X_train,y_train)
y_pred = LR.predict(X_test)

DTC = DecisionTreeClassifier(criterion='entropy', max_features= 3 , max_depth=2 )
DTC.fit(X_train, y_train)
y_pred2 = DTC.predict(X_test)

KNN = KNeighborsClassifier(n_neighbors=20)
KNN.fit(X_train, y_train)
y_pred3 = KNN.predict(X_test)

RC = RandomForestClassifier(n_estimators=400)
RC.fit(X_train, y_train)
y_pred4 = RC.predict(X_test)

GBC = GradientBoostingClassifier(learning_rate=0.1, max_features=4, subsample= 0.5, n_estimators= 200)
GBC.fit(X_train, y_train)
y_pred5 = GBC.predict(X_test)
```

The Accuracy and F1-score for each model

- Logistic Regression

```
print('Accuracy score:', accuracy_score(y_test, y_pred))
print('F1_score:', f1_score(y_test, y_pred))
✓ 0.3s
Accuracy score: 0.817
F1_score: 0.27380952380952384
```

- Decision Tree

```
print('Accuracy score:', accuracy_score(y_test, y_pred2))
print('F1_score:', f1_score(y_test, y_pred2))
✓ 0.3s
Accuracy score: 0.8335
F1_score: 0.2309468822170901
```

- K Nearest Neighbors

```
print('Accuracy score:', accuracy_score(y_test, y_pred3))
print('F1_score:', f1_score(y_test, y_pred3))
✓ 0.3s
Accuracy score: 0.8515
F1_score: 0.40954274353876746
```

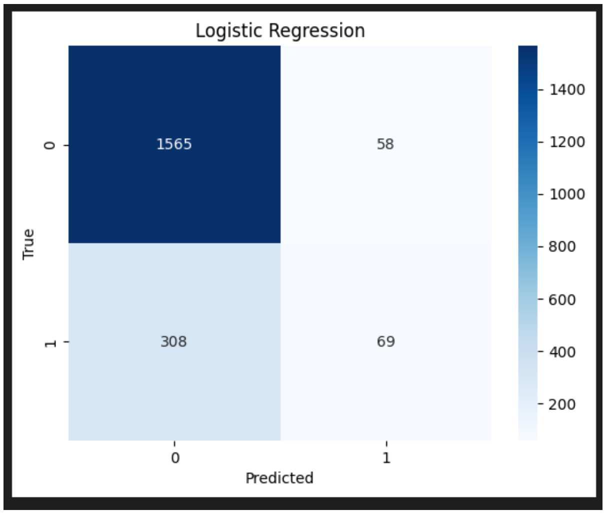
- Random Forest

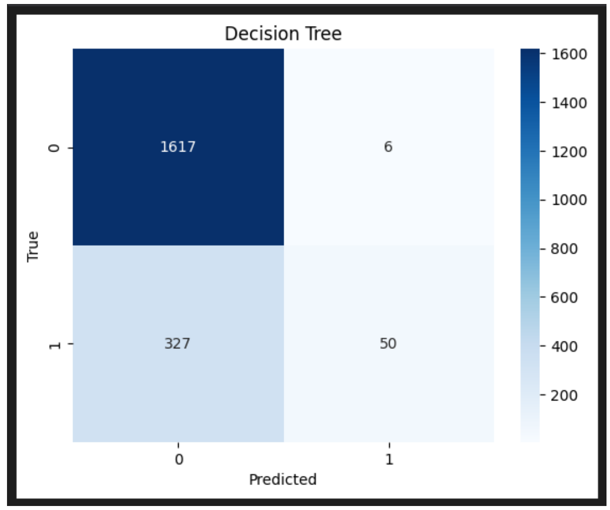
```
print('Accuracy score:', accuracy_score(y_test, y_pred4))
print('F1_score:', f1_score(y_test, y_pred4))
✓ 0.3s
Accuracy score: 0.8785
F1_score: 0.608695652173913
```

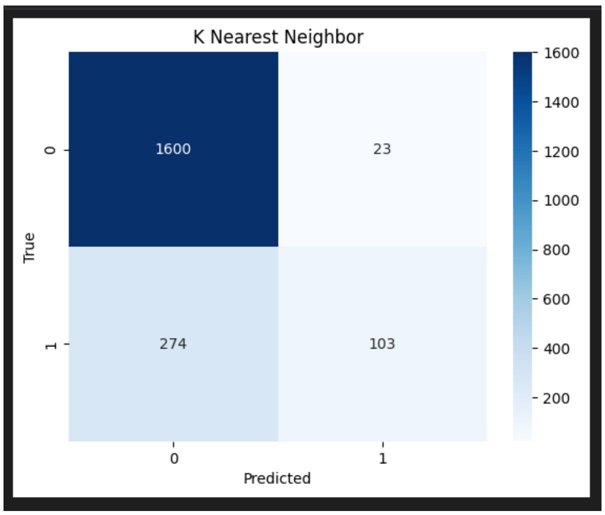
- Gradient Boosting

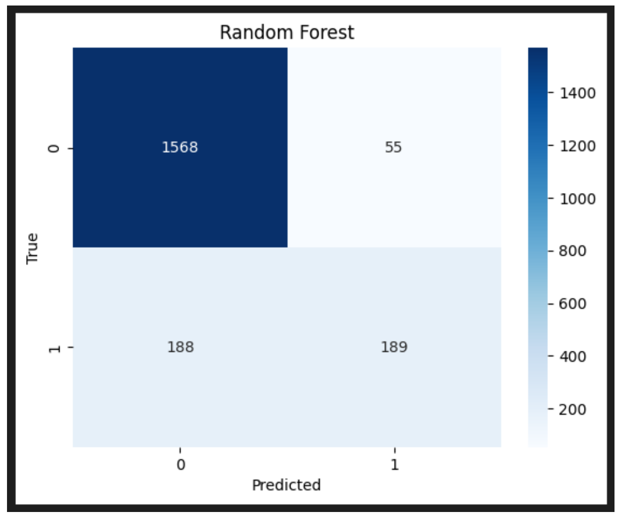
```
print('Accuracy score:', accuracy_score(y_test, y_pred5))
print('F1_score:', f1_score(y_test, y_pred5))
✓ 0.3s
Accuracy score: 0.8815
F1_score: 0.6183574879227053
```

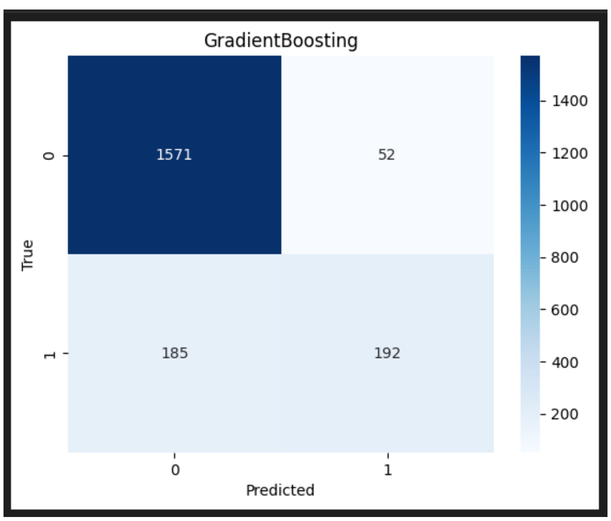
The Confusion Matrix











Section 5. Final Model According to the results Random Forest Classifier is the best model for this dataset.

According to the results Gradient Boosting Classifier is the best model for this dataset.

Section 6. Results

Considering correlations between features, age is the key features to determine customer churn. Accuracy is %88 and F1-Score is %62 for this classification. For better results, hyperparameter tuning to the models will increase accuracy.