Start coding or generate with AI.

```
pip install CatBoost
```

```
Collecting CatBoost
  Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl.metadata (1.2 kB)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from CatBoost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from CatBoost) (3.8.0)
Requirement already satisfied: numpy<2.0,>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from CatBoost) (1.26.4)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from CatBoost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from CatBoost) (1.13.1)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from CatBoost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from CatBoost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->CatBoost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->CatBoost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->CatBoost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (4.55.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->CatBoost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->CatBoost) (9.0.0)
Downloading catboost-1.2.7-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 98.7/98.7 MB 9.4 MB/s eta 0:00:00
Installing collected packages: CatBoost
Successfully installed CatBoost-1.2.7
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report, f1_score, precision_score, recall_score, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score,recall_score
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, GridSearchCV
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42: FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not installed.

You can install it with `pip install dask[dataframe]` or `conda install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

```
df = pd.read_csv("/content/BankCustomerChurn/churn.csv", index_col=0)
```

```
df.head()
```

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

```
df.shape
```

```
(10000, 13)
```

```
df.info()
```

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

```
df.describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000
5%	1.557882e+07	489.000000	25.000000	1.000000	0.000000	1.000000	0.00000	0.000000	9851.818500
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500
90%	1.579083e+07	778.000000	53.000000	9.000000	149244.792000	2.000000	1.00000	1.000000	179674.704000
95%	1.580303e+07	812.000000	60.000000	9.000000	162711.669000	2.000000	1.00000	1.000000	190155.375500
99%	1.581311e+07	850.000000	72.000000	10.000000	185967.985400	3.000000	1.00000	1.000000	198069.734500
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000

```
categorical_variables = [col for col in df.columns if col in "O"
                      or df[col].nunique() <=11
                      and col not in "Exited"]
```

```
categorical_variables
```

```
['Geography',
 'Gender',
 'Tenure',
 'NumOfProducts',
 'HasCrCard',
 'IsActiveMember']
```

```
numeric_variables = [col for col in df.columns if df[col].dtype != "object"
                        and df[col].nunique() >11
                        and col not in "CustomerId"]
numeric_variables
```

```
['CreditScore', 'Age', 'Balance', 'EstimatedSalary']
```

```
df["Exited"].value_counts()
```

Exited	count
0	7963
1	2037

dtype: int64

```
churn = df.loc[df["Exited"]==1]
```

```
not_churn = df.loc[df["Exited"]==0]
not_churn["Tenure"].value_counts().sort_values()
```

Tenure	count
0	318
10	389
9	771
6	771
4	786
3	796
5	803
1	803
8	828
2	847
7	851

dtype: int64

```
churn["Tenure"].value_counts().sort_values()
```

Tenure	count
0	95
10	101
7	177
6	196
8	197
2	201
4	203
5	209
3	213
9	213
1	232

dtype: int64

```
not_churn["NumOfProducts"].value_counts().sort_values()
```

NumOfProducts	count
3	46
1	3675
2	4242

```
churn["NumOfProducts"].value_counts().sort_values()
```

NumOfProducts	count
4	60
3	220
2	348
1	1409

```
not_churn["HasCrCard"].value_counts()
```

HasCrCard	count
1	5631
0	2332

```
churn["HasCrCard"].value_counts()
```

HasCrCard	count
1	1424
0	613

```
not_churn["IsActiveMember"].value_counts()
```

IsActiveMember	count
1	4416
0	3547

```
churn["IsActiveMember"].value_counts()
```

IsActiveMember	count
0	1302
1	735

```
not_churn.Geography.value_counts().sort_values()
```

Geography	count
Germany	1695
Spain	2064
France	4204

```
churn.Geography.value_counts().sort_values()
```

Geography	count
Spain	413
France	810
Germany	814

```
not_churn.Gender.value_counts()
```

Gender	count
Male	4559
Female	3404

```
churn.Gender.value_counts()
```

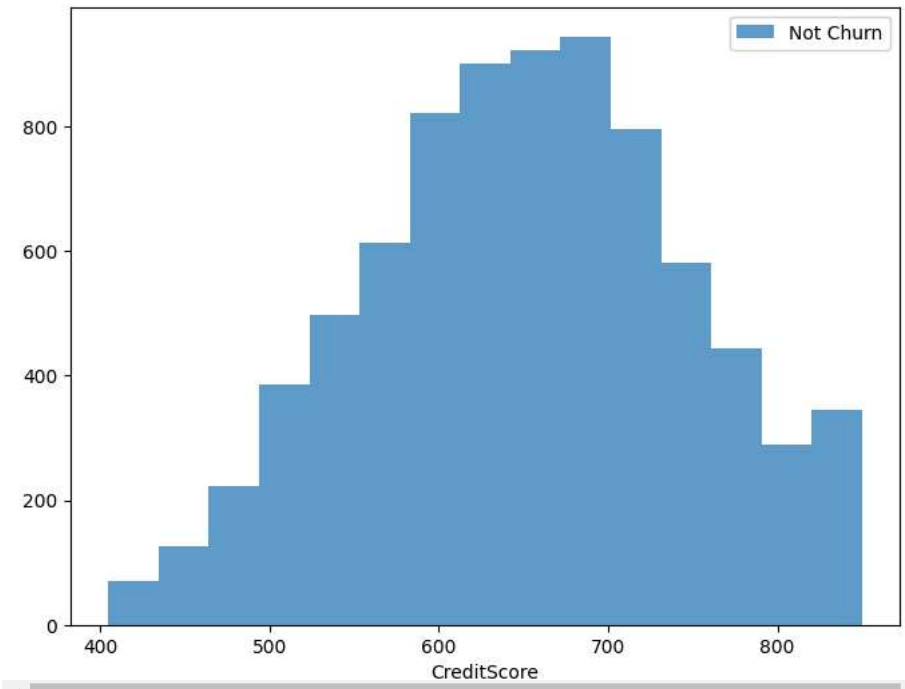
Gender	count
Female	1139
Male	898

Credit Score

```
not_churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	CreditScore
count	7963.000000
mean	651.853196
std	95.653837
min	405.000000
5%	492.000000
25%	585.000000
50%	653.000000
75%	718.000000
90%	778.000000
95%	812.000000
99%	850.000000
max	850.000000

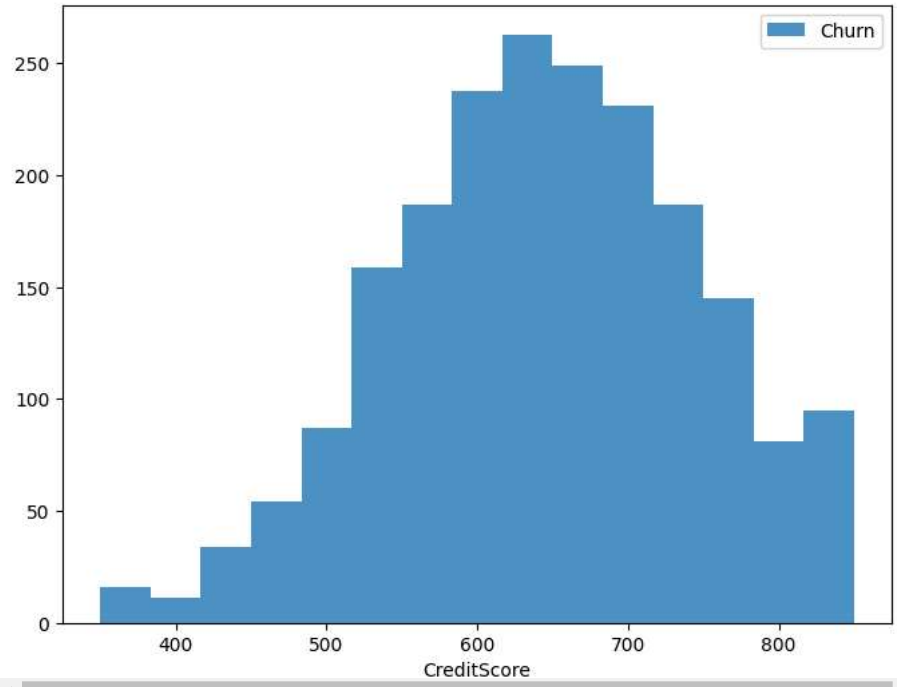
```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(not_churn["CreditScore"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

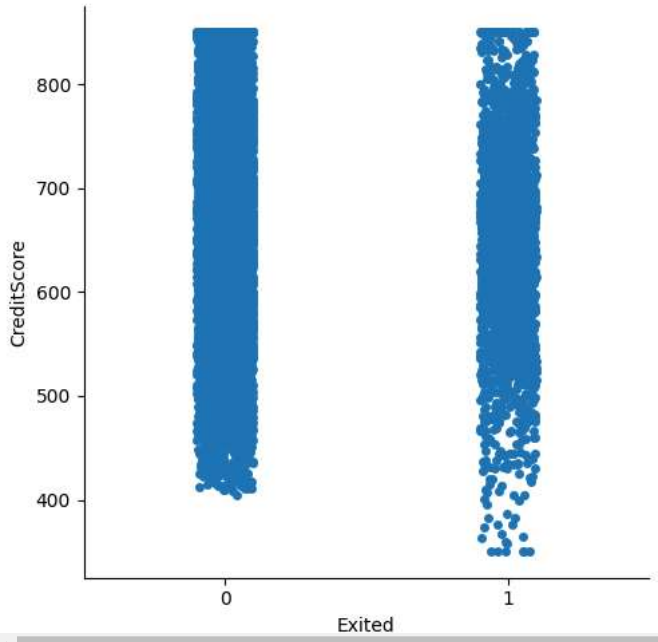
	CreditScore
count	2037.000000
mean	645.351497
std	100.321503
min	350.000000
5%	479.000000
25%	578.000000
50%	646.000000
75%	716.000000
90%	776.400000
95%	812.200000
99%	850.000000
max	850.000000

```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(churn["CreditScore"],bins=15, alpha=0.8, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
sns.catplot(x="Exited", y="CreditScore", data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe26cb7b730>
```

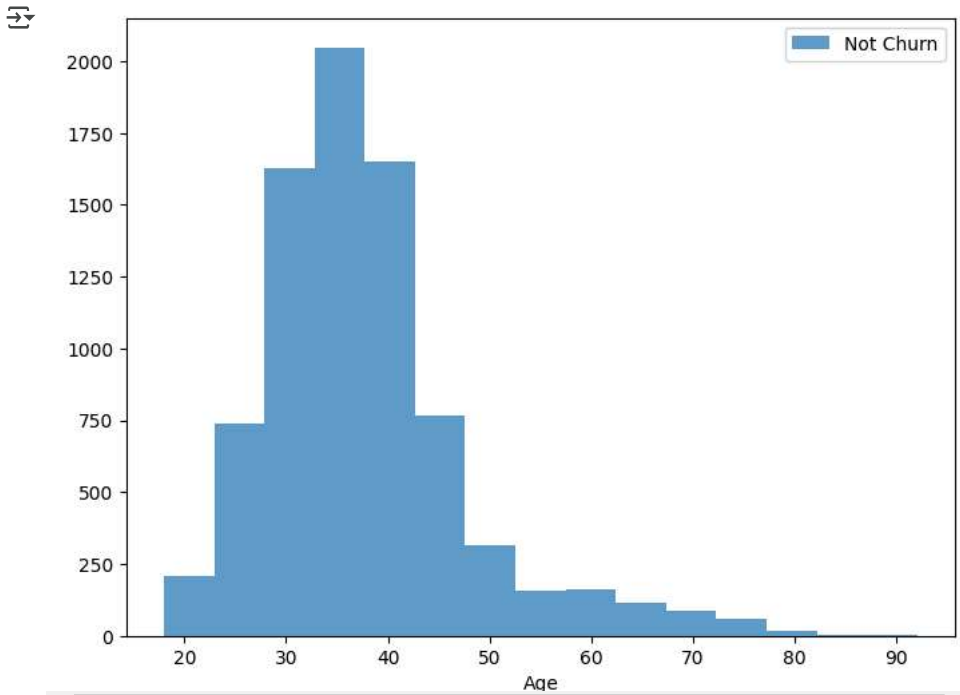


Age

```
not_churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	Age
count	7963.000000
mean	37.408389
std	10.125363
min	18.000000
5%	24.000000
25%	31.000000
50%	36.000000
75%	41.000000
90%	49.000000
95%	59.000000
99%	73.000000
max	92.000000

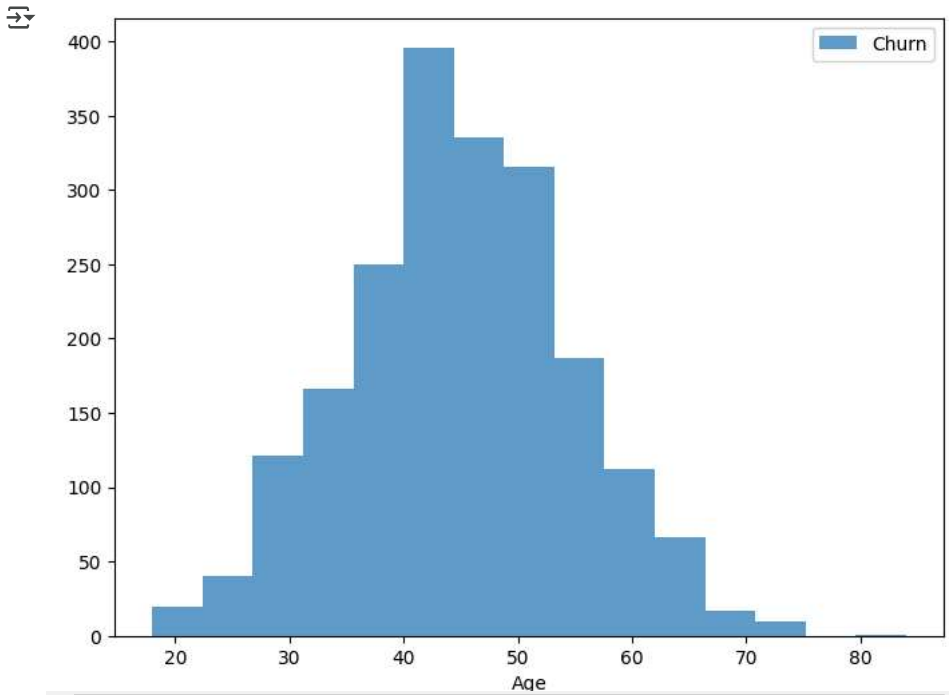
```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(not_churn["Age"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

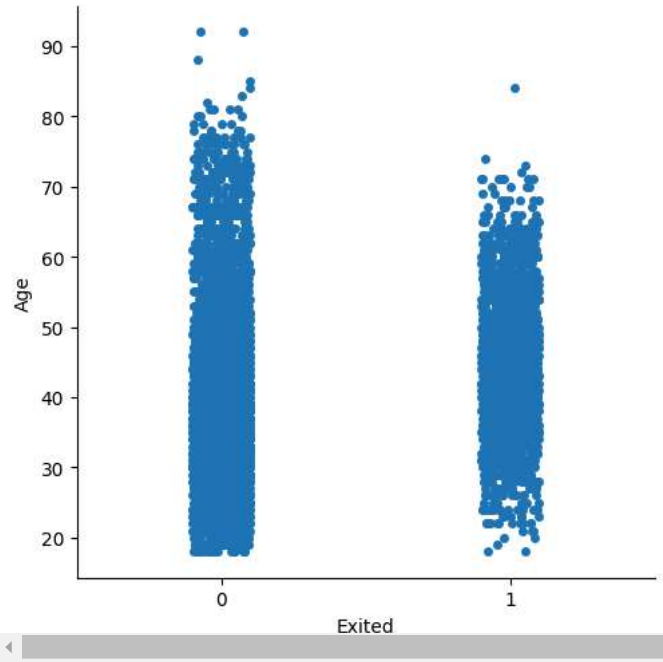
	Age
count	2037.000000
mean	44.837997
std	9.761562
min	18.000000
5%	29.000000
25%	38.000000
50%	45.000000
75%	51.000000
90%	58.000000
95%	61.000000
99%	68.000000
max	84.000000

```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(churn["Age"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
sns.catplot(x="Exited", y="Age", data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe26c3dfa60>
```

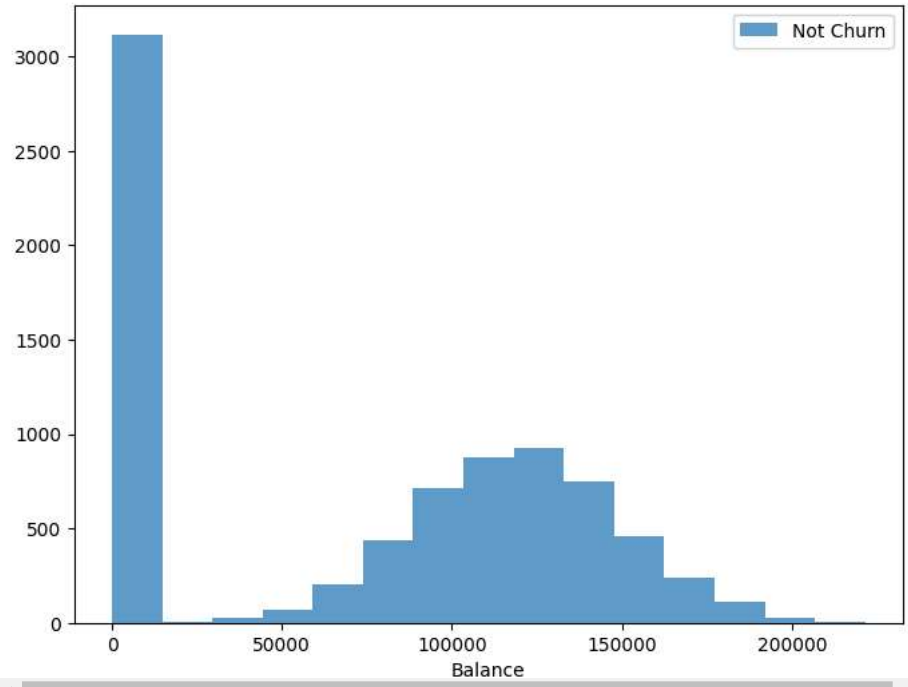


Balance

```
not_churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	Balance
count	7963.000000
mean	72745.296779
std	62848.040701
min	0.000000
5%	0.000000
25%	0.000000
50%	92072.680000
75%	126410.280000
90%	148730.298000
95%	161592.595000
99%	183753.906200
max	221532.800000

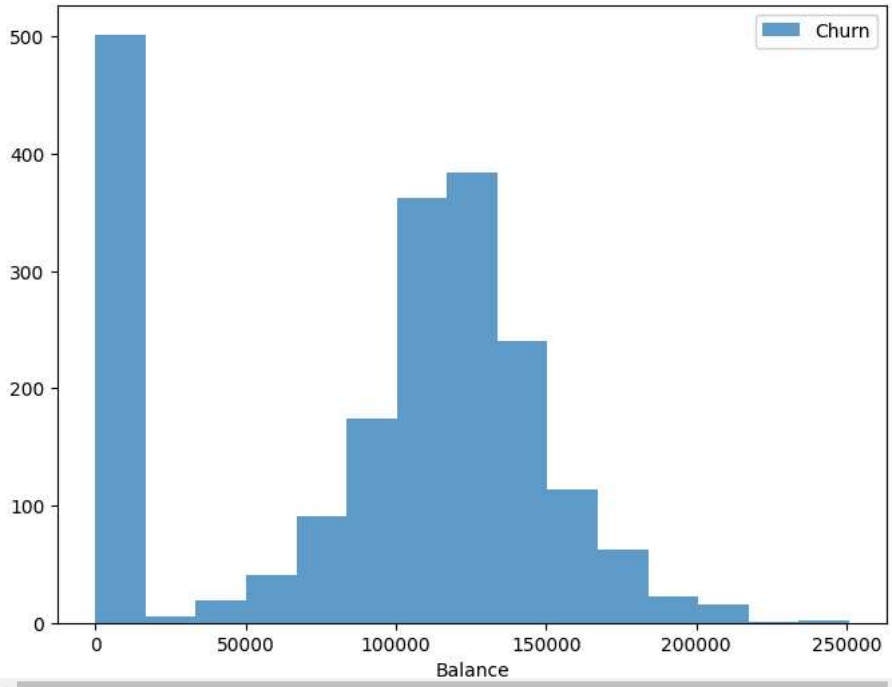
```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(not_churn["Balance"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

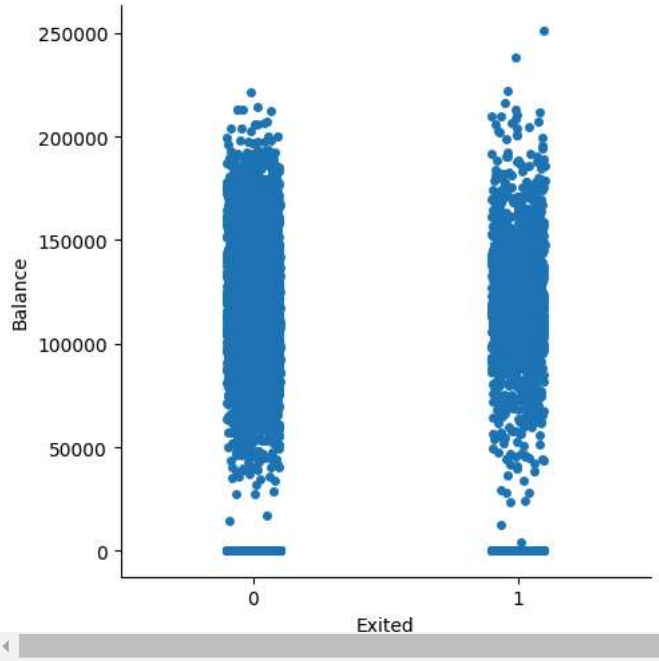
	Balance
count	2037.000000
mean	91108.539337
std	58360.794816
min	0.000000
5%	0.000000
25%	38340.020000
50%	109349.290000
75%	131433.330000
90%	152080.618000
95%	167698.240000
99%	197355.288400
max	250898.090000

```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(churn["Balance"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
sns.catplot(x="Exited", y="Balance", data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe26c74c520>
```

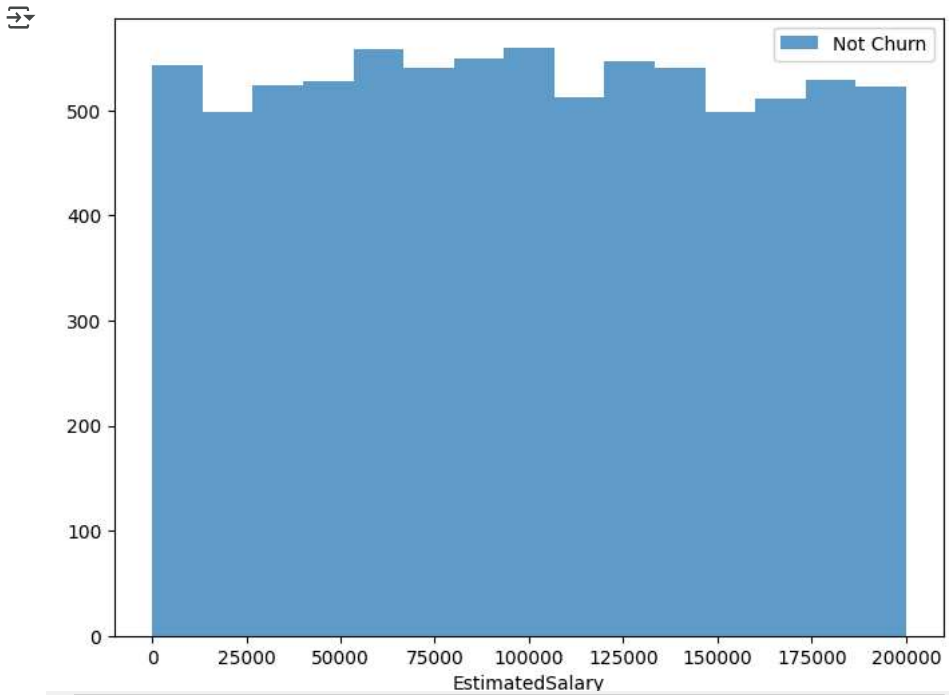


Estimsated Salary

```
not_churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	EstimatedSalary
count	7963.000000
mean	99738.391772
std	57405.586966
min	90.070000
5%	9773.542000
25%	50783.490000
50%	99645.040000
75%	148609.955000
90%	179453.212000
95%	190107.557000
99%	198131.465200
max	199992.480000

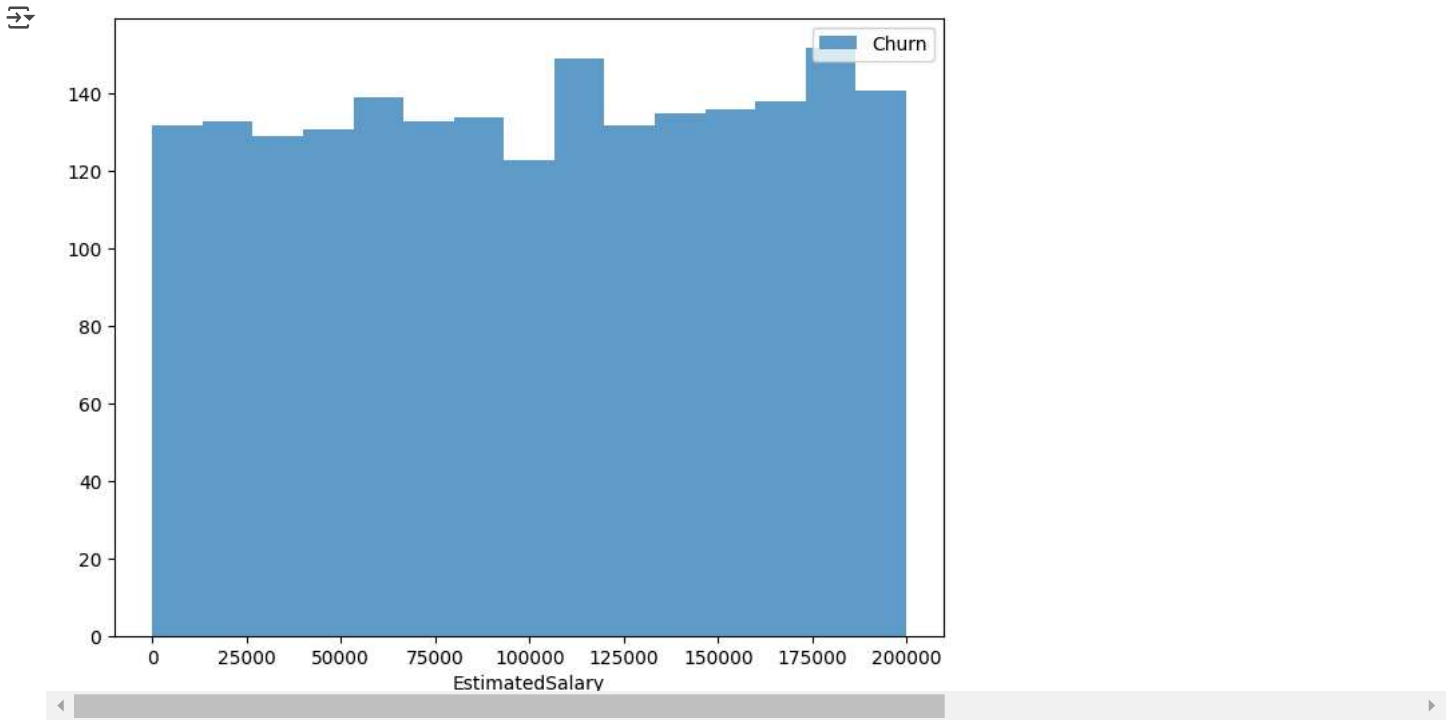
```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
pyplot.hist(not_churn["EstimatedSalary"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

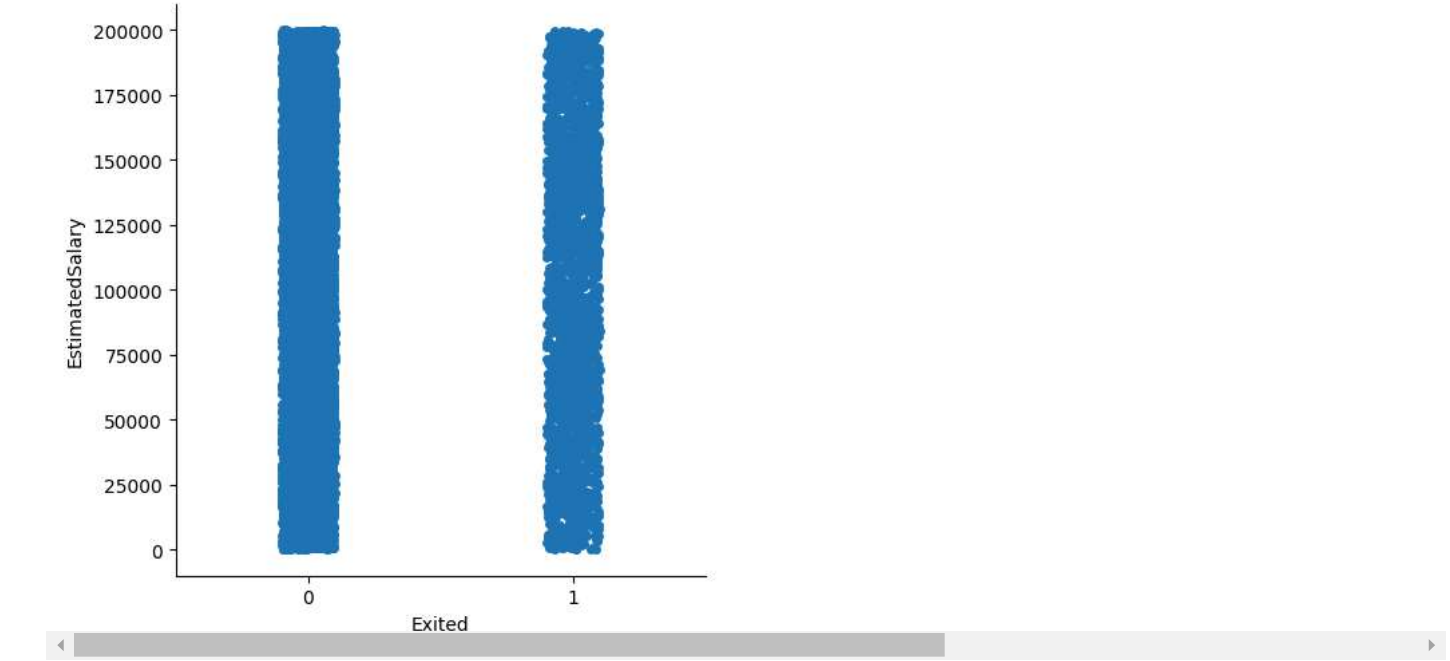
	EstimatedSalary
count	2037.000000
mean	101465.677531
std	57912.418071
min	11.580000
5%	10030.760000
25%	51907.720000
50%	102460.840000
75%	152422.910000
90%	180169.390000
95%	190328.982000
99%	197717.297600
max	199808.100000

```
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
pyplot.hist(churn["EstimatedSalary"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
sns.catplot(x="Exited", y="EstimatedSalary", data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe26cb80af0>
```

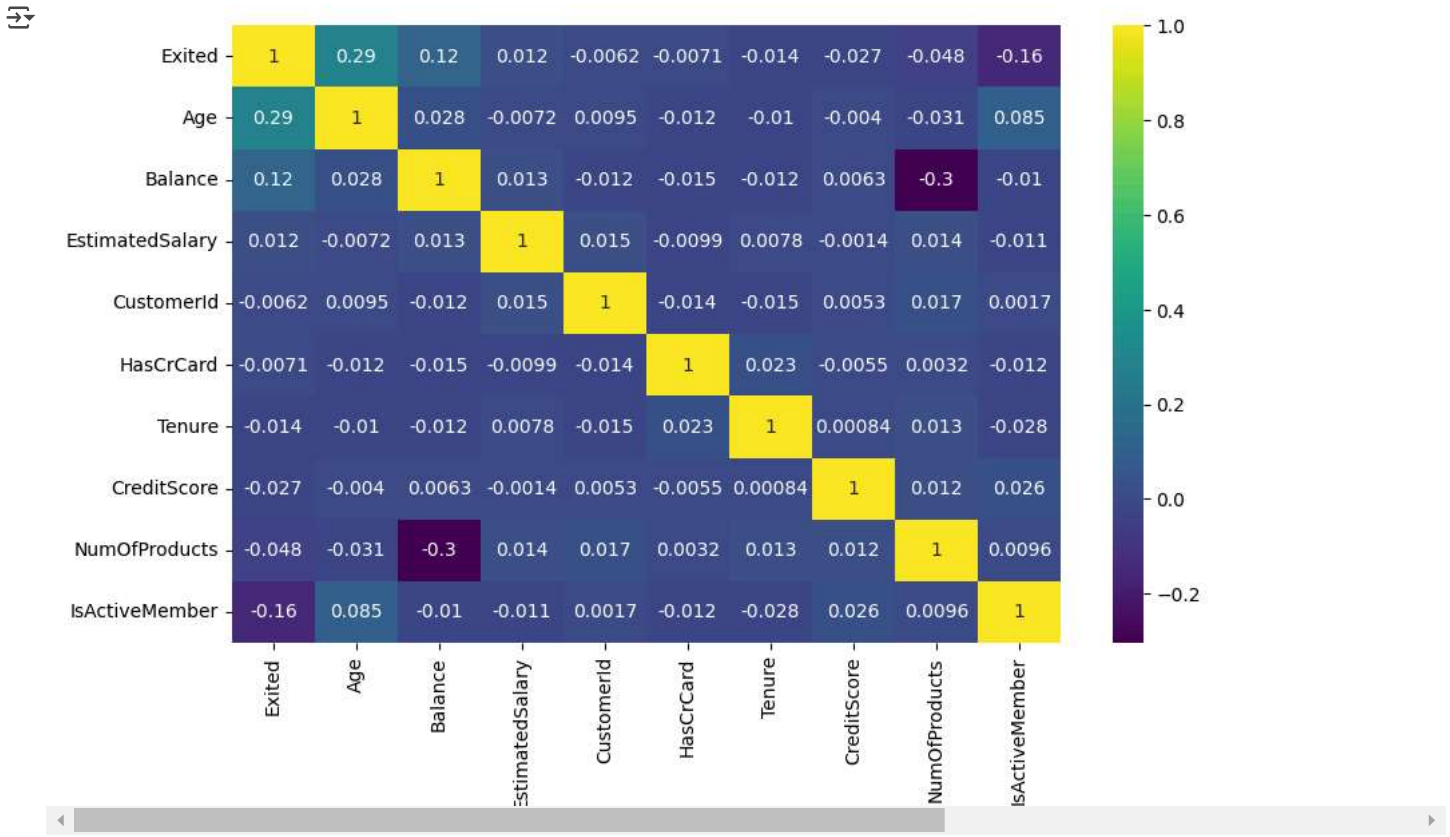


Correlation Matrix

```
numeric_df = df.select_dtypes(include=[float, int])

# Compute the correlation matrix for numeric columns only
k = 10 # number of variables for heatmap
cols = numeric_df.corr().nlargest(k, 'Exited')['Exited'].index
cm = numeric_df[cols].corr()

# Plot the heatmap
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True, cmap='viridis')
plt.show()
```




```
df.isnull().sum()
```

	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

Outliers

```
def outlier_thresholds(dataframe, variable, low_quantile=0.05, up_quantile=0.95):
    quantile_one = dataframe[variable].quantile(low_quantile)
    quantile_three = dataframe[variable].quantile(up_quantile)
    interquantile_range = quantile_three - quantile_one
    up_limit = quantile_three + 1.5 * interquantile_range
    low_limit = quantile_one - 1.5 * interquantile_range
    return low_limit, up_limit


def has_outliers(dataframe, numeric_columns, plot=False):
   # variable_names = []
    for col in numeric_columns:
```

```
        low_limit, up_limit = outlier_thresholds(dataframe, col)
        if dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].any(axis=None):
            number_of_outliers = dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].shape[0]
            print(col, " : ", number_of_outliers, "outliers")
            #variable_names.append(col)
            if plot:
                sns.boxplot(x=dataframe[col])
                plt.show()


for var in numeric_variables:
    print(var, "has " , has_outliers(df, [var]),  "Outliers")
```

```
CreditScore has  None Outliers
Age has  None Outliers
Balance has  None Outliers
EstimatedSalary has  None Outliers
```

Work on Important Features

```
df["NewTenure"] = df["Tenure"]/df["Age"]
df["NewCreditsScore"] = pd.qcut(df['CreditScore'], 6, labels = [1, 2, 3, 4, 5, 6])
df["NewAgeScore"] = pd.qcut(df['Age'], 8, labels = [1, 2, 3, 4, 5, 6, 7, 8])
df["NewBalanceScore"] = pd.qcut(df['Balance'].rank(method="first"), 5, labels = [1, 2, 3, 4, 5])
df["NewEstSalaryScore"] = pd.qcut(df['EstimatedSalary'], 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])


df.head()
```

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

```
list = ["Gender", "Geography"]
df = pd.get_dummies(df, columns =list, drop_first = True)


df.head()
```

RowNumber	CustomerId	Surname	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	N
1	15634602	Hargrave	619	42	2	0.00	1	1	1	101348.88	1	
2	15647311	Hill	608	41	1	83807.86	1	0	1	112542.58	0	
3	15619304	Onio	502	42	8	159660.80	3	1	0	113931.57	1	
4	15701354	Boni	699	39	1	0.00	2	0	0	93826.63	0	
5	15737888	Mitchell	850	43	2	125510.82	1	1	1	79084.10	0	

```
df = df.drop(["CustomerId","Surname"], axis = 1)


def robust_scaler(variable):
    var_median = variable.median()
    quartile1 = variable.quantile(0.25)
    quartile3 = variable.quantile(0.75)
    interquantile_range = quartile3 - quartile1
    if int(interquantile_range) == 0:
        quartile1 = variable.quantile(0.05)
        quartile3 = variable.quantile(0.95)
        interquantile_range = quartile3 - quartile1
        if int(interquantile_range) == 0:
            quartile1 = variable.quantile(0.01)
```

```
            quartile3 = variable.quantile(0.99)
            interquantile_range = quartile3 - quartile1
            z = (variable - var_median) / interquantile_range
            return round(z, 3)

        z = (variable - var_median) / interquantile_range
        return round(z, 3)
    else:
        z = (variable - var_median) / interquantile_range
    return round(z, 3)


new_cols_ohe = ["Gender_Male","Geography_Germany","Geography_Spain"]
like_num = [col for col in df.columns if df[col].dtypes != 'O' and len(df[col].value_counts()) <= 10]
cols_need_scale = [col for col in df.columns if col not in new_cols_ohe
                   and col not in "Exited"
                   and col not in like_num]

for col in cols_need_scale:
    df[col] = robust_scaler(df[col])


df.head()
```

RowNumber	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	NewTenure	NewCreditsS
1	-0.246	0.417	-0.75	-0.761	1	1	1	0.012	1	-0.217	
2	-0.328	0.333	-1.00	-0.105	1	0	1	0.126	0	-0.279	
3	-1.119	0.417	0.75	0.489	3	1	0	0.140	1	0.164	
4	0.351	0.167	-1.00	-0.761	2	0	0	-0.065	0	-0.276	
5	1.478	0.500	-0.75	0.222	1	1	1	-0.215	0	-0.220	

```
models = [('LR', LogisticRegression(random_state=123456)),
          ('KNN', KNeighborsClassifier()),
          ('CART', DecisionTreeClassifier(random_state=123456)),
          ('RF', RandomForestClassifier(random_state=123456)),
          ('SVR', SVC(gamma='auto',random_state=123456)),
          ('GB', GradientBoostingClassifier(random_state=12345)),
          ("LightGBM", LGBMClassifier(random_state=123456))]

results = []
names = []

# Set shuffle=True for KFold
for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=123456)  # shuffle is True
    cv_results = cross_val_score(model, X, y, cv=kfold)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
model_GB = GradientBoostingClassifier(random_state=12345)
model_GB.fit(X_train, y_train)
y_pred = model_GB.predict(X_test)
conf_mat = confusion_matrix(y_pred,y_test)
conf_mat
```

```
array([[1520,  230],
       [  53,  197]])
```

```
print("True Positive : ", conf_mat[1, 1])
print("True Negative : ", conf_mat[0, 0])
print("False Positive: ", conf_mat[0, 1])
print("False Negative: ", conf_mat[1, 0])
```

```
True Positive :  197
True Negative :  1520
False Positive:  230
False Negative:  53
```

```
print(classification_report(model_GB.predict(X_test),y_test))
```

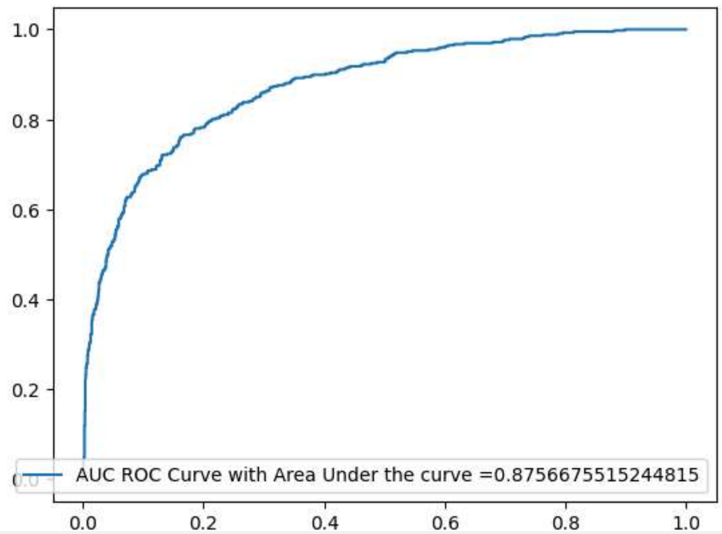
```
              precision    recall  f1-score   support

           0       0.97      0.87      0.91      1750
           1       0.46      0.79      0.58       250

    accuracy                           0.86      2000
   macro avg       0.71      0.83      0.75      2000
weighted avg       0.90      0.86      0.87      2000
```

```
def generate_auc_roc_curve(clf, X_test):
    y_pred_proba = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test,  y_pred_proba)
    auc = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="AUC ROC Curve with Area Under the curve ="+str(auc))
    plt.legend(loc=4)
    plt.show()
    pass
```

```
generate_auc_roc_curve(model_GB, X_test)
```




```
lgb_model = LGBMClassifier()
# Model Tuning
lgbm_params = {'colsample_bytree': 0.5,
 'learning_rate': 0.01,
 'max_depth': 6,
 'n_estimators': 500}

lgbm_tuned = LGBMClassifier(**lgbm_params).fit(X, y)
```

```
[LightGBM] [Info] Number of positive: 2037, number of negative: 7963
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000488 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1137
[LightGBM] [Info] Number of data points in the train set: 10000, number of used features: 16
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.203700 -> initscore=-1.363328
[LightGBM] [Info] Start training from score -1.363328
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
gbm_model = GradientBoostingClassifier()
# Model Tuning
gbm_params = {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 1}
gbm_tuned = GradientBoostingClassifier(**gbm_params).fit(X,y)
```

```
# Define your models
```