CUSTOMER CHURN PREDICTION (LR,KNN,DT,RF,SVR,GB)

Author: Irfan Ullah Khan


```
# @title
!pip install catboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.10/dist-packages (1.2.3)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.25.2
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from catboost) (1.11.4)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catbo
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->cat
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboos
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->ca
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->ca
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catb
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catbo
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->cat
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost
```

```
# loading necessary libraries
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
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.preprocessing import RobustScaler
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
```

```
df = pd.read_csv("BankCustomerData.csv")
```

Exploratory Data Analysis

```
df.head()
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_nun
0	15634602	619	France	Female	42	2	0.00	
1	15647311	608	Spain	Female	41	1	83807.86	
2	15619304	502	France	Female	42	8	159660.80	
3	15701354	699	France	Female	39	1	0.00	
4	15737888	850	Spain	Female	43	2	125510.82	

Next steps: Generate code with df View recommended plots

```
df.shape
```

```
(10000, 12)
```

```
df.info()
```

```
<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
```

```
df.describe()
```

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	activ
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	1000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	

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

categorical_variables
```

```
['country',
 'gender',
 'tenure',
 'products_number',
 'credit_card',
 'active_member']
```

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

```
['credit_score', 'age', 'balance', 'estimated_salary']
```

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

```
0    7963
1    2037
Name: churn, dtype: int64
```

```
exit = df.loc[df["churn"]==1]
not_exit = df.loc[df["churn"]==0]
```

```
not_exit.shape
#exit.shape
```

```
(7963, 12)
```

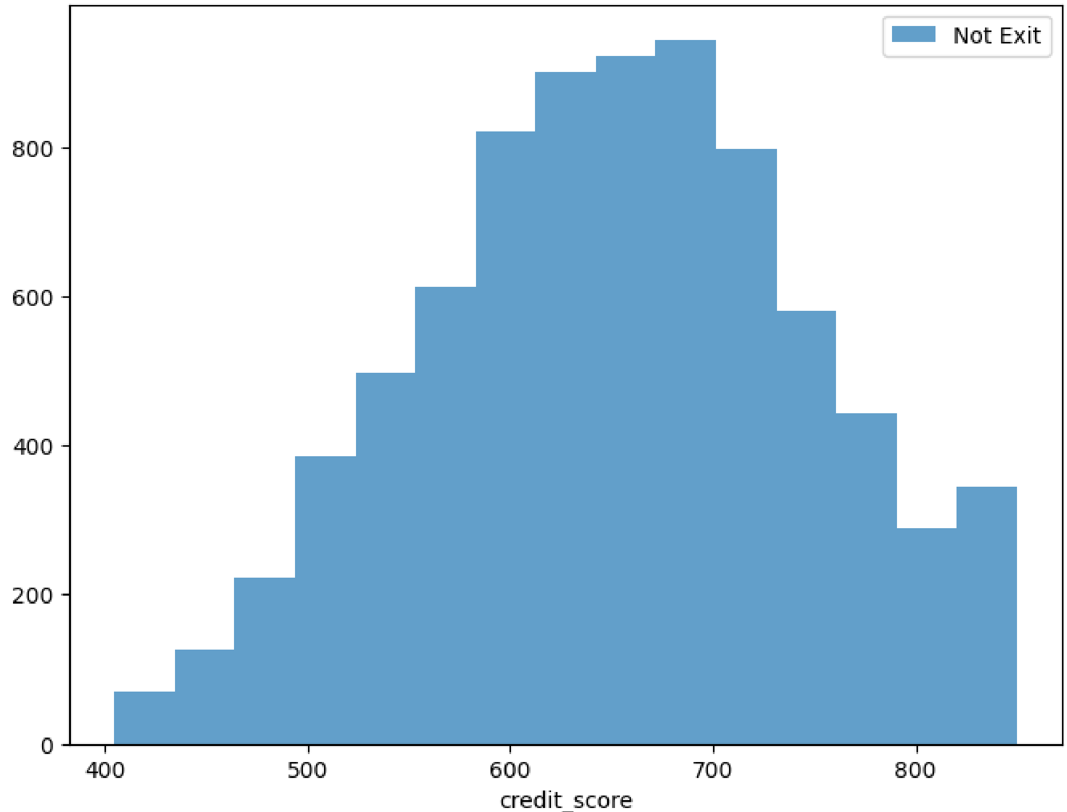
```
def get_sorted_value_counts(df, column_name):
    return df[column_name].value_counts().sort_values()

print('Tenure frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "tenure"))
print(get_sorted_value_counts(exit, "tenure"))

print('Number of products frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "products_number"))
print(get_sorted_value_counts(exit, "products_number"))
```

```
print('credit card frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "credit_card"))
print(get_sorted_value_counts(exit, "credit_card"))

print('If active based frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "active_member"))
print(get_sorted_value_counts(exit, "active_member"))

print('country frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "country"))
print(get_sorted_value_counts(exit, "country"))

print('gender frequency of the churned and not churned groups')
print(get_sorted_value_counts(not_exit, "gender"))
print(get_sorted_value_counts(exit, "gender"))
```

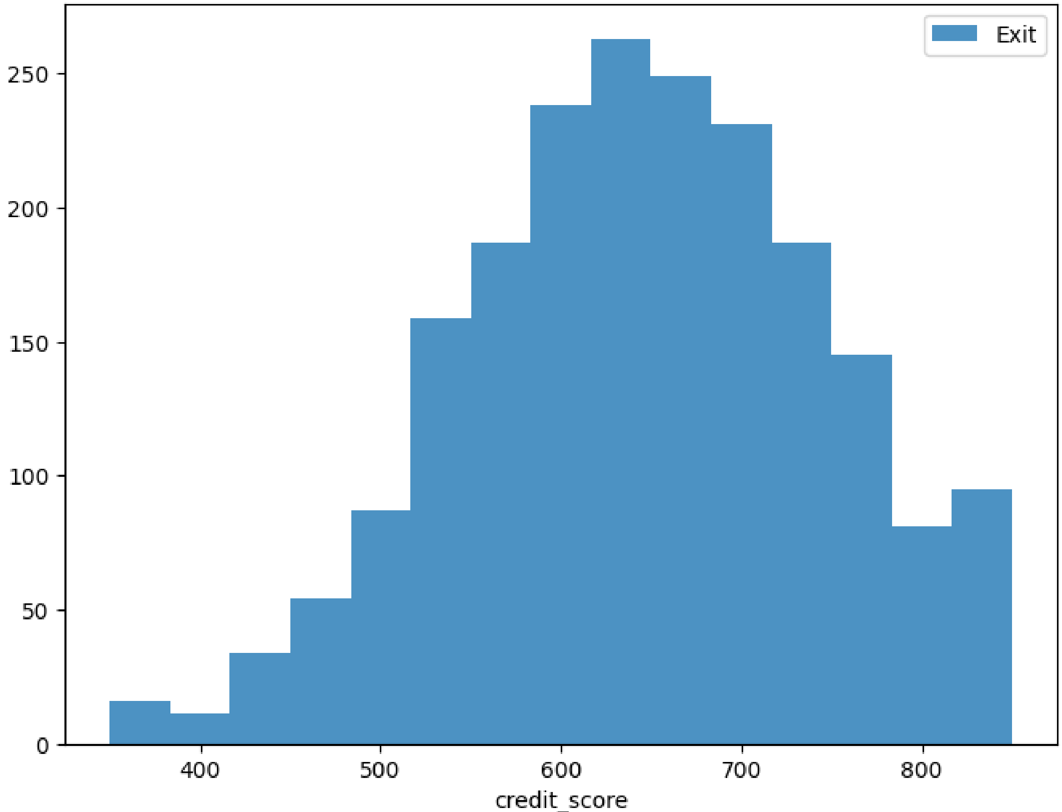
```
1     803
5     803
8     828
2     847
7     851
Name: tenure, dtype: int64
0      95
10    101
7     177
6     196
8     197
2     201
4     203
5     209
3     213
9     213
1     232
Name: tenure, dtype: int64
Number of products frequency of the churned and not churned groups
3      46
1    3675
2    4242
Name: products_number, dtype: int64
4      60
3     220
2     348
1    1409
Name: products_number, dtype: int64
credit card frequency of the churned and not churned groups
0    2332
1    5631
Name: credit_card, dtype: int64
0     613
1    1424
Name: credit_card, dtype: int64
If active based frequency of the churned and not churned groups
0    3547
1    4416
Name: active_member, dtype: int64
1     735
0    1302
Name: active_member, dtype: int64
country frequency of the churned and not churned groups
Germany    1695
Spain      2064
```

```
Female    3404
Male      4559
Name: gender, dtype: int64
Male       898
Female    1139
Name: gender, dtype: int64
```

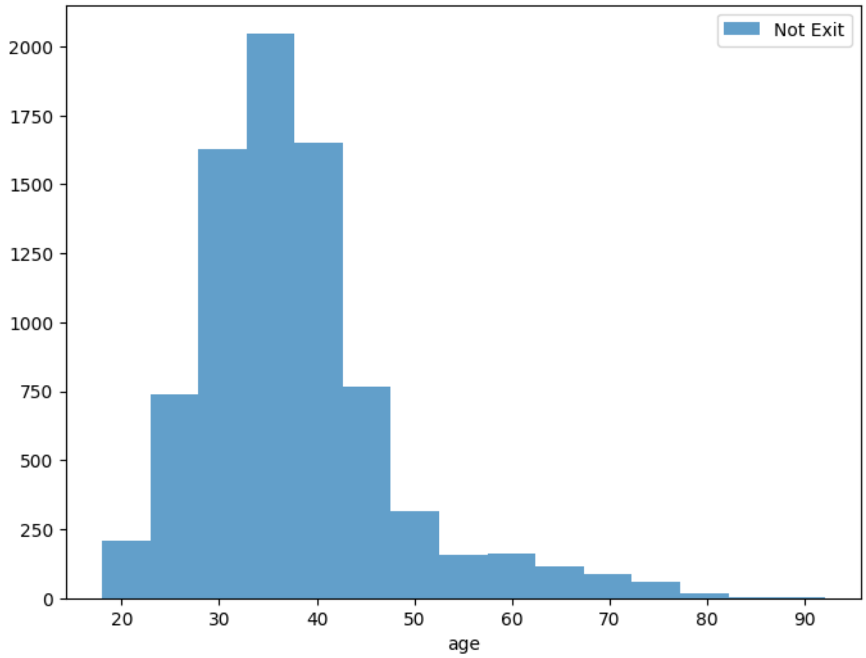
```
# distribution of the Credit Score for not_exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('credit_score')
pyplot.hist(not_exit["credit_score"],bins=15, alpha=0.7, label='Not Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```



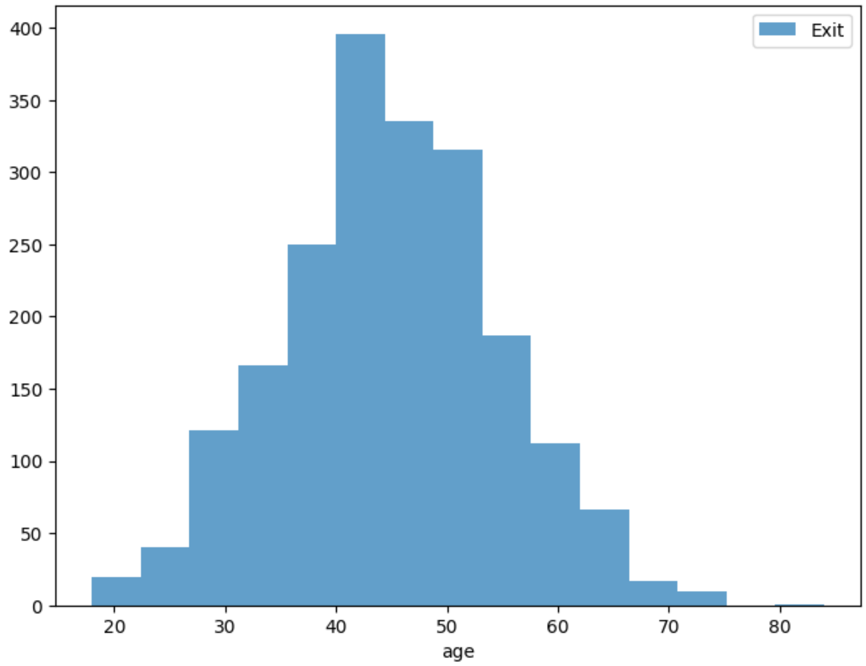

```
# distribution of the Credit Score for exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('credit_score')
pyplot.hist(exit["credit_score"],bins=15, alpha=0.8, label='Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```



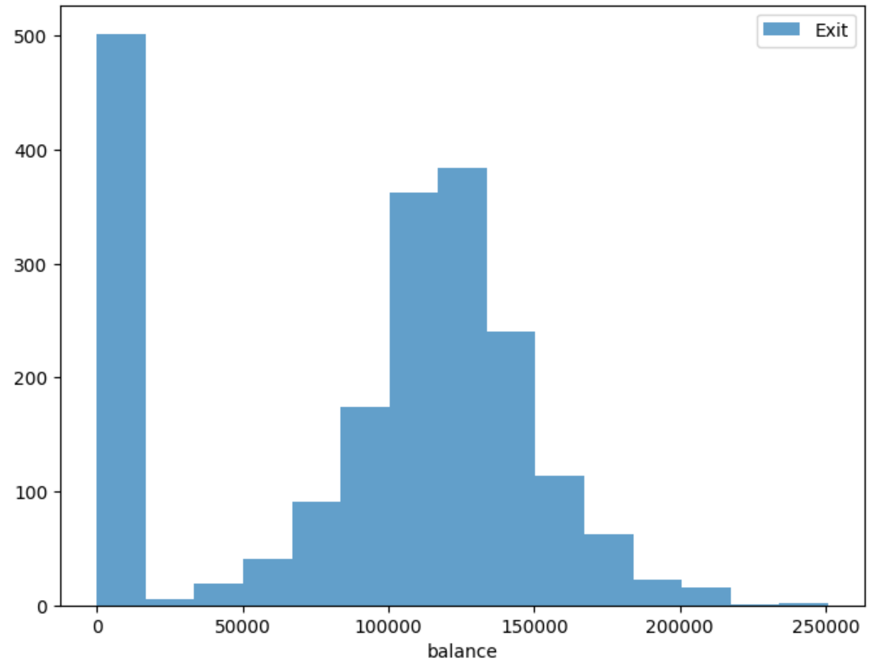

```
# distribution of the Age for not_exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('age')
pyplot.hist(not_exit["age"],bins=15, alpha=0.7, label='Not Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```



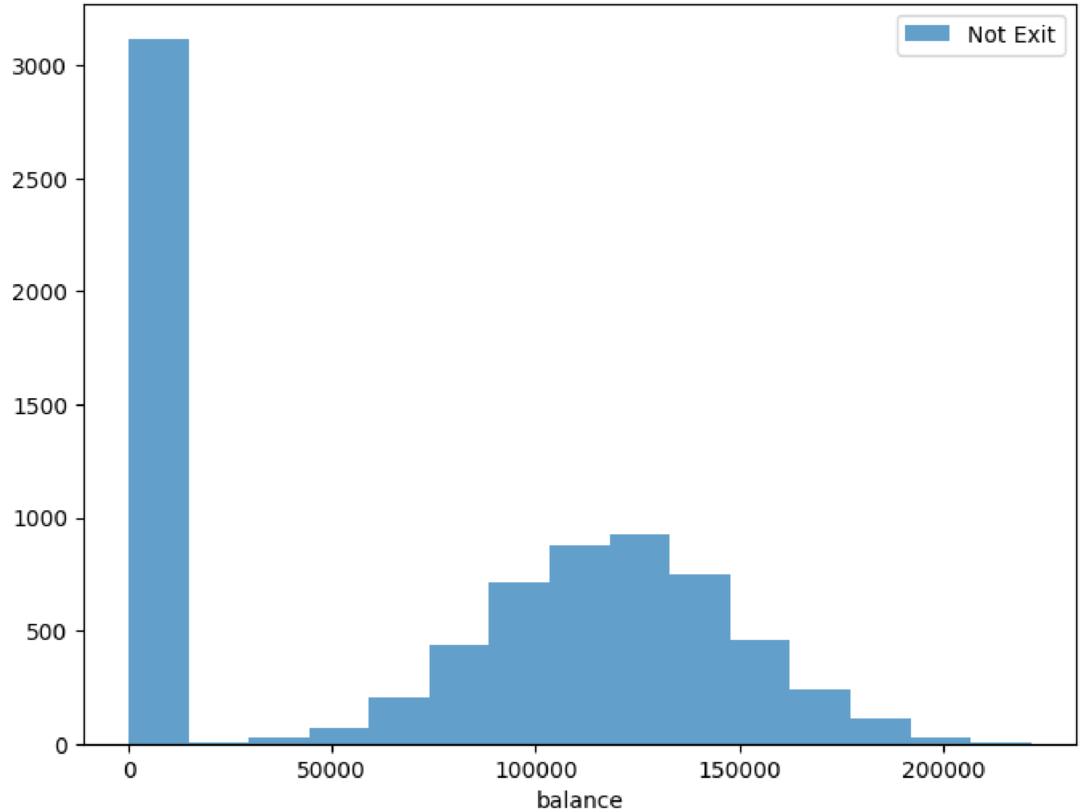

```
# distribution of the Age for exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('age')
pyplot.hist(exit["age"],bins=15, alpha=0.7, label='Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```



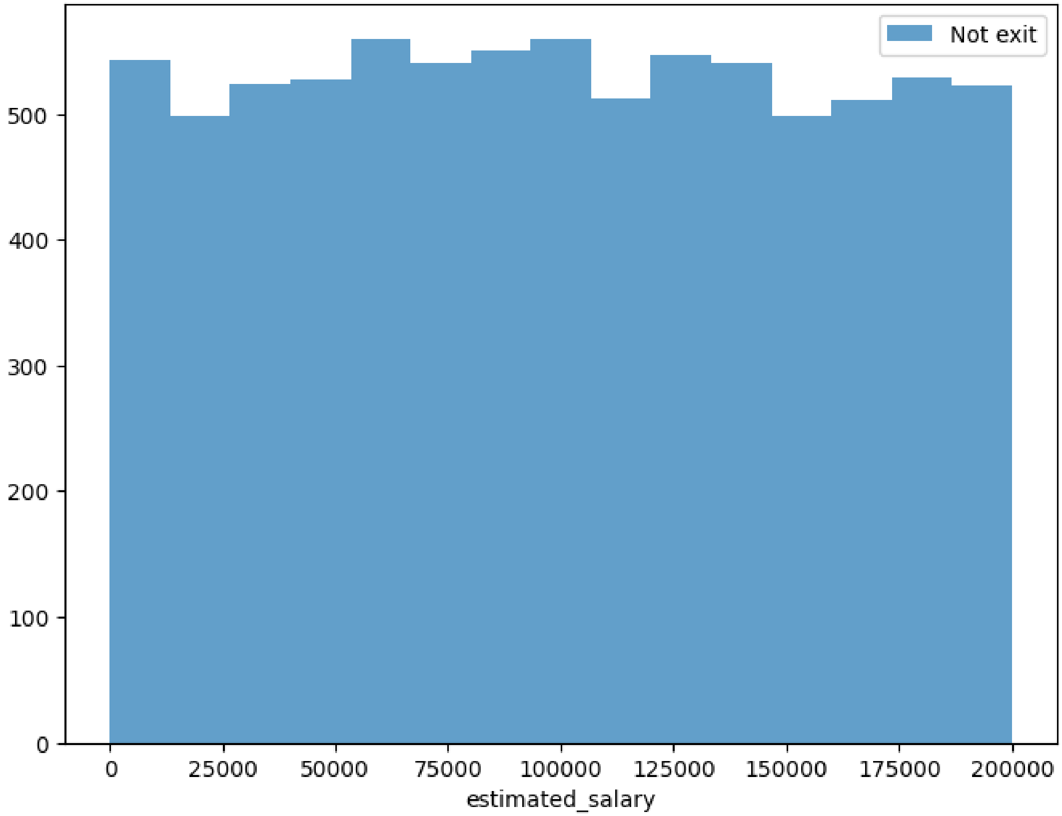

```
# distribution of the Balance for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('balance')
pyplot.hist(exit["balance"],bins=15, alpha=0.7, label='Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```



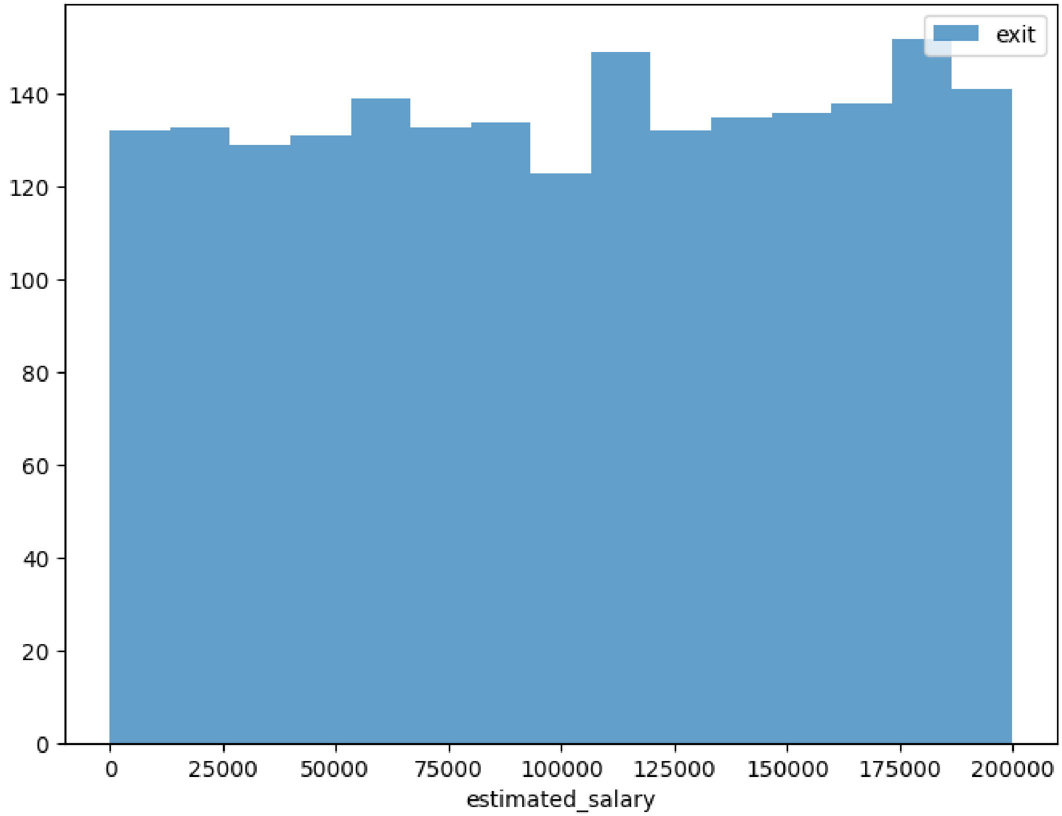

```
# distribution of the Balance for not_exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('balance')
pyplot.hist(not_exit["balance"],bins=15, alpha=0.7, label='Not Exit')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# distribution of the estimated_salary for exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('estimated_salary')
pyplot.hist(not_exit["estimated_salary"],bins=15, alpha=0.7, label='Not exit')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# distribution of the estimated_salary for exit
pyplot.figure(figsize=(8,6))
pyplot.xlabel('estimated_salary')
pyplot.hist(exit["estimated_salary"],bins=15, alpha=0.7, label='exit')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
df.head()
```

	customer_id	credit_score	country	gender	age	tenure	balance	products_number	credit_card	active_membe
0	15634602	619	France	Female	42	2	0.00	1	1	
1	15647311	608	Spain	Female	41	1	83807.86	1	0	
2	15619304	502	France	Female	42	8	159660.80	3	1	
3	15701354	699	France	Female	39	1	0.00	2	0	
4	15737888	850	Spain	Female	43	2	125510.82	1	1	

Next steps: Generate code with df View recommended plots

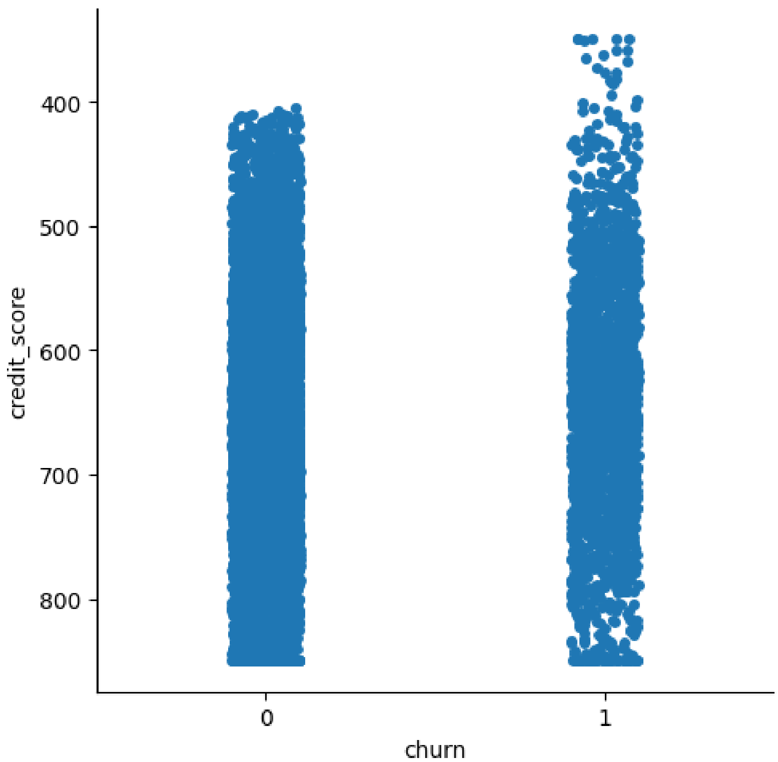
```
df.dtypes
```

```
customer_id            int64
credit_score        category
country               object
gender                object
age                    int64
tenure                 int64
balance              float64
products_number        int64
credit_card            int64
active_member          int64
estimated_salary     float64
churn               category
dtype: object
```

```
df["churn"] = df["churn"].astype("category")
df["credit_score"] = df["credit_score"].astype("category")
```

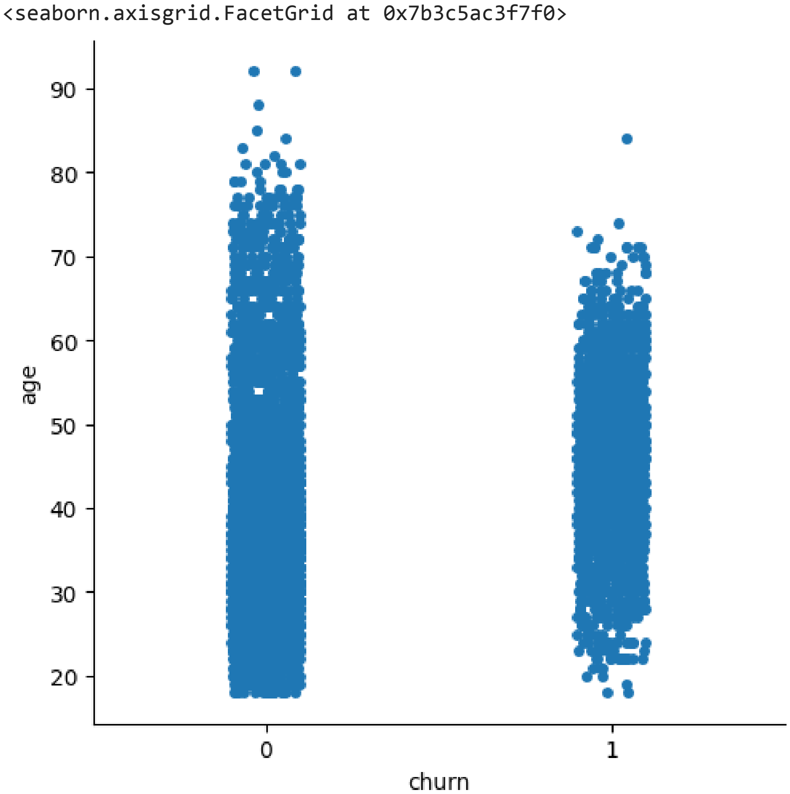
```
sns.catplot(x="churn", y="credit_score", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7b3c5ab07580>
```



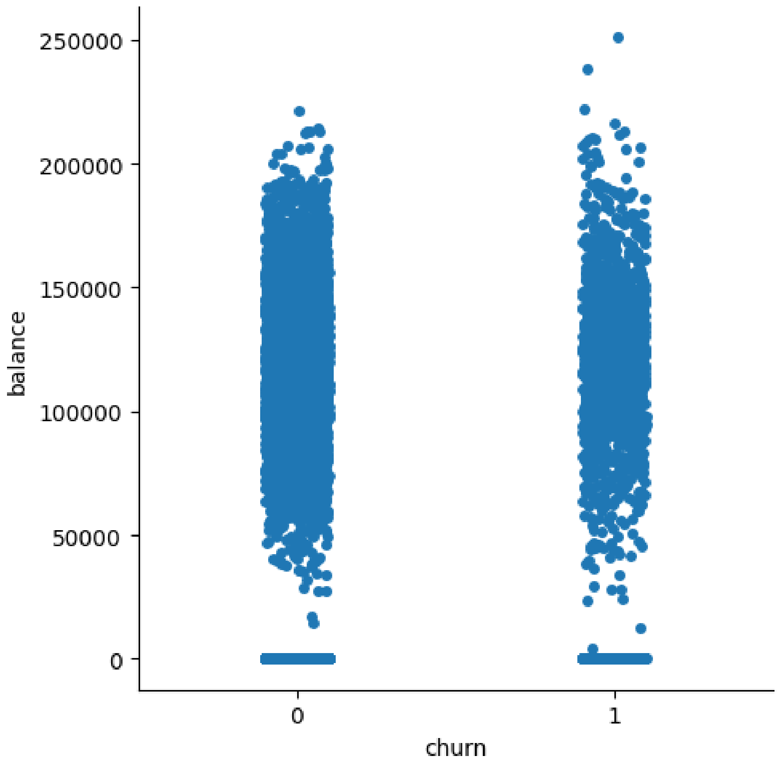

```
sns.catplot(x="churn", y="age", data = df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7b3c5ac3f7f0>
```



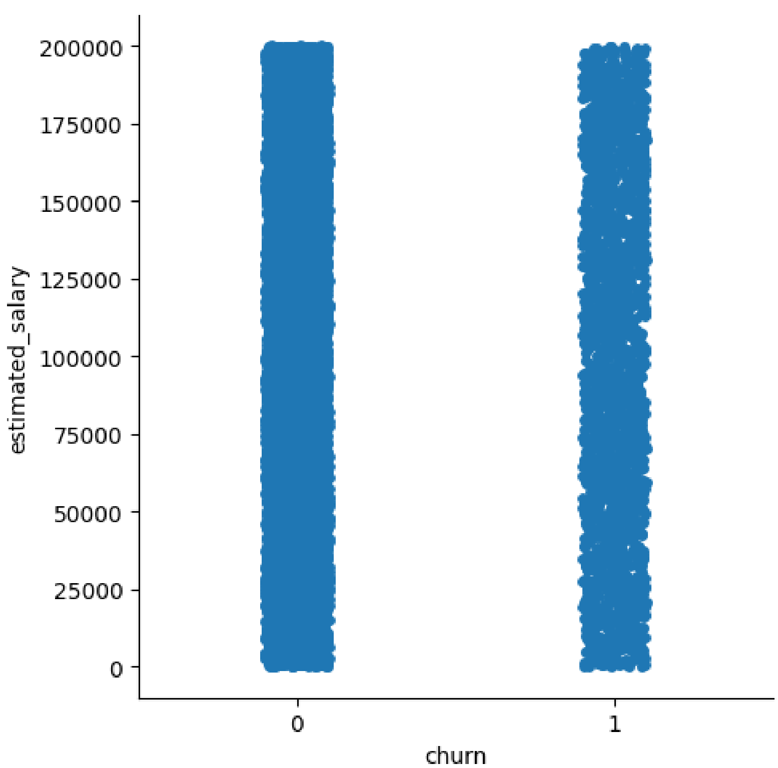
```
sns.catplot(x="churn", y="balance", data = df)
```

<seaborn.axisgrid.FacetGrid at 0x7b3c5a7a4370>




```
sns.catplot(x="churn", y="estimated_salary", data = df)
```

<seaborn.axisgrid.FacetGrid at 0x7b3c5ac0a0e0>



Data Preprocessing

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

```
customer_id         0
credit_score        0
country             0
gender              0
age                 0
tenure              0
balance             0
products_number     0
credit_card         0
active_member       0
estimated_salary    0
churn               0
dtype: int64
```

There are no missing values in this dataset

There are no outliers in this dataset

```
# Variables to apply one hot encoding
list = ["gender", "country"]
df = pd.get_dummies(df, columns =list, drop_first = True)
```

```
df.head()
```

	customer_id	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_sal
0	15634602	619	42	2	0.00	1	1	1	101348
1	15647311	608	41	1	83807.86	1	0	1	112542
2	15619304	502	42	8	159660.80	3	1	0	113931
3	15701354	699	39	1	0.00	2	0	0	93826
4	15737888	850	43	2	125510.82	1	1	1	79084

Next steps: Generate code with df | View recommended plots

Scalling

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

```
df
```

	credit_score	age	tenure	balance	products_number	credit_card	active_member	estimated_salary	churn
0	619	42	2	0.00	1	1	1	101348.88	1
1	608	41	1	83807.86	1	0	1	112542.58	0
2	502	42	8	159660.80	3	1	0	113931.57	1
3	699	39	1	0.00	2	0	0	93826.63	0
4	850	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...
9995	771	39	5	0.00	2	1	0	96270.64	0
9996	516	35	10	57369.61	1	1	1	101699.77	0
9997	709	36	7	0.00	1	0	1	42085.58	1
9998	772	42	3	75075.31	2	1	0	92888.52	1
9999	792	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 12 columns

Next steps: Generate code with df View recommended plots

```
scaler = RobustScaler()
scaled_data = scaler.fit_transform(df)
```

Modeling

```
X = df.drop("churn",axis=1)
y = df["churn"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

```
models = [('LR', LogisticRegression(random_state=42)),
          ('KNN', KNeighborsClassifier()),
          ('DT', DecisionTreeClassifier(random_state=42)),
          ('RF', RandomForestClassifier(random_state=42)),
          ('SVR', SVC(gamma='auto',random_state=42)),
          ('GB', GradientBoostingClassifier(random_state = 42)),
          ("LightGBM", LGBMClassifier(random_state=42))]
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10)
    cv_results = cross_val_score(model, X, y, cv=kfold)
    results.append(cv_results)
    names.append(name)
    output = "%s: %f " % (name, cv_results.mean())
    print(output)
```

```
LR: 0.789800
KNN: 0.765000
DT: 0.790100
RF: 0.861300
SVR: 0.796300
GB: 0.864500
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
```

```
[LightGBM] [Info] Number of positive: 1833, number of negative: 7167
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000538 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 1013
[LightGBM] [Info] Number of data points in the train set: 9000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.203667 -> initscore=-1.363533
[LightGBM] [Info] Start training from score -1.363533
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Info] Number of positive: 1825, number of negative: 7175
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001856 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1014
[LightGBM] [Info] Number of data points in the train set: 9000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.202778 -> initscore=-1.369023
[LightGBM] [Info] Start training from score -1.369023
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Info] Number of positive: 1821, number of negative: 7179
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001862 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1015
[LightGBM] [Info] Number of data points in the train set: 9000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.202333 -> initscore=-1.371774
[LightGBM] [Info] Start training from score -1.371774
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Info] Number of positive: 1823, number of negative: 7177
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001870 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1014
[LightGBM] [Info] Number of data points in the train set: 9000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.202556 -> initscore=-1.370398
[LightGBM] [Info] Start training from score -1.370398
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Info] Number of positive: 1837, number of negative: 7163
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001950 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1015
[LightGBM] [Info] Number of data points in the train set: 9000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.204111 -> initscore=-1.360795
[LightGBM] [Info] Start training from score -1.360795
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Info] Number of positive: 1834, number of negative: 7166
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001911 seconds.
```

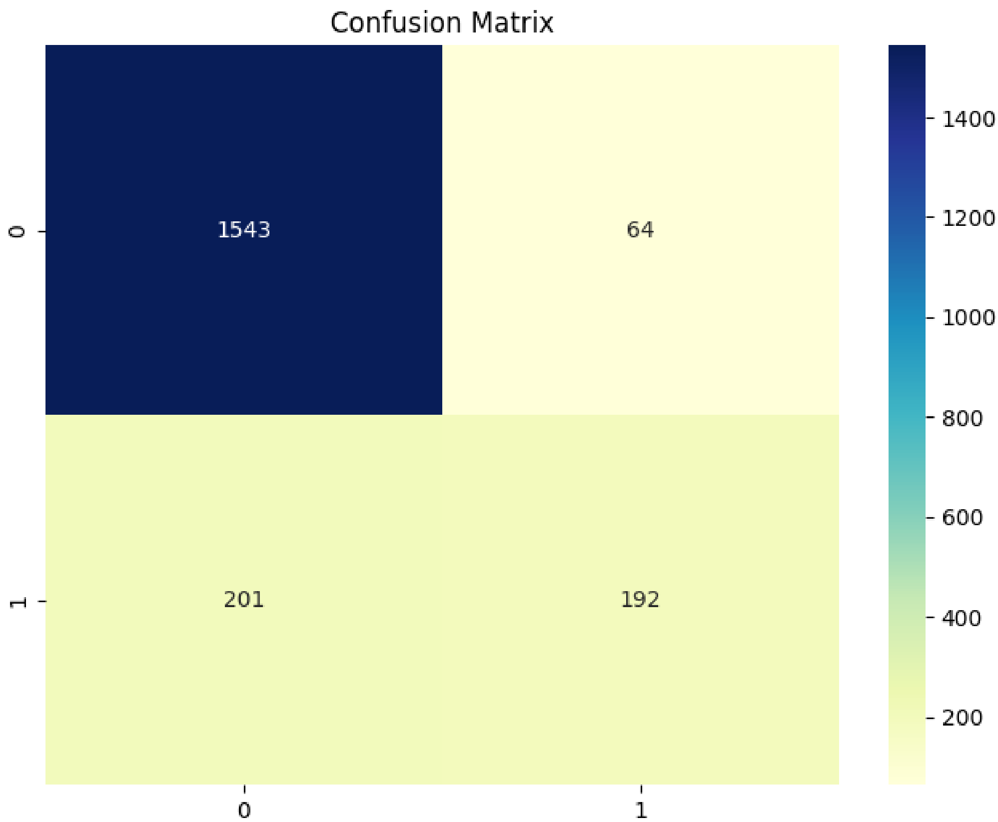
```
def classifier_results(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test, pred)
    precision = precision_score(y_test, pred,zero_division=0)
    recall = recall_score(y_test, pred)
    f1 = f1_score(y_test, pred)
    roc_auc = roc_auc_score(y_test, pred_proba)
    print("Accuracy: {:.4f} Precision: {:.4f} Recall: {:.4f} F1: {:.4f} ROC-AUC: {:.4f}".format(accuracy,precision,r
    plt.figure(figsize=(8, 6))
    ax = sns.heatmap(confusion, cmap = 'YlGnBu',annot = True, fmt='d')
    ax.set_title('Confusion Matrix')

    return confusion
```

```
def generate_auc_roc_curve(y_test,  y_pred_proba):
    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)
```

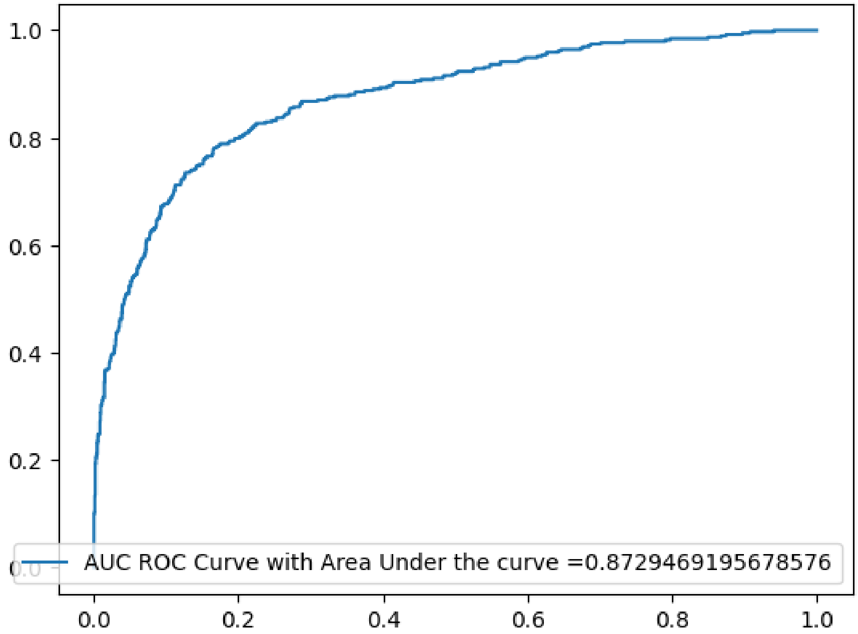
```
    plt.legend(loc=4)
    plt.show()
```

```
model = GradientBoostingClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:, 1]
classifier_results(y_test, pred=y_pred, pred_proba=y_pred_proba)
```

```
Accuracy: 0.8675 Precision: 0.7500 Recall: 0.4885 F1: 0.5917 ROC-AUC: 0.8729
array([[1543,   64],
       [ 201,  192]])
```




```
generate_auc_roc_curve(y_test,  y_pred_proba)
```



Hyperparameter Tuning

```
def grid_search_cv(estimator, param_grid, cv=5, scoring='roc_auc'):
    grid_search = GridSearchCV(estimator=estimator, param_grid=param_grid, cv=cv, scoring=scoring)
    grid_search.fit(X_train, y_train)
    return grid_search.best_params_
```

Lets tune the LGBMClassifier,GradientBoostingClassifier and RandomForestClassifier

```
lgbm_param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': [3, 5, 7]
}

gb_param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.1, 0.05, 0.01],
    'max_depth': [3, 5, 7]
}

rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 5, 7]
}
```

```
lgbm = LGBMClassifier(random_state=42)
lgbm_best_params = grid_search_cv(lgbm, lgbm_param_grid)
print("Best parameters for LGBMClassifier:", lgbm_best_params)
```

```
[LightGBM] [Info] Total Bins 1009
[LightGBM] [Info] Number of data points in the train set: 6400, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.205469 -> initscore=-1.352458
[LightGBM] [Info] Start training from score -1.352458
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Info] Number of positive: 1315, number of negative: 5085
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001371 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1011
[LightGBM] [Info] Number of data points in the train set: 6400, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.205469 -> initscore=-1.352458
[LightGBM] [Info] Start training from score -1.352458
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Info] Number of positive: 1315, number of negative: 5085
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001446 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1009
[LightGBM] [Info] Number of data points in the train set: 6400, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.205469 -> initscore=-1.352458
[LightGBM] [Info] Start training from score -1.352458
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Info] Number of positive: 1315, number of negative: 5085
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000293 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 1007
[LightGBM] [Info] Number of data points in the train set: 6400, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.205469 -> initscore=-1.352458
[LightGBM] [Info] Start training from score -1.352458
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Info] Number of positive: 1644, number of negative: 6356
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001647 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1013
[LightGBM] [Info] Number of data points in the train set: 8000, number of used features: 11
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.205500 -> initscore=-1.352267
```

```
gb = GradientBoostingClassifier(random_state=42)
gb_best_params = grid_search_cv(gb, gb_param_grid)
print("Best parameters for GradientBoostingClassifier:", gb_best_params)
```

```
Best parameters for GradientBoostingClassifier: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 200}
```

```
rf = RandomForestClassifier(random_state=42)
rf_best_params = grid_search_cv(rf, rf_param_grid)
print("Best parameters for RandomForestClassifier:", rf_best_params)
```

```
Best parameters for RandomForestClassifier: {'max_depth': 7, 'n_estimators': 200}
```

```
model1 = LGBMClassifier(learning_rate= 0.05, max_depth= 5, n_estimators= 100,random_state=42)
model1.fit(X_train, y_train)
y_pred = model1.predict(X_test)
y_pred_proba = model1.predict_proba(X_test)[:, 1]
classifier_results(y_test, pred=y_pred, pred_proba=y_pred_proba)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Warning] Categorical features with more bins than the configured maximum bin number found.
[LightGBM] [Warning] For categorical features, max_bin and max_bin_by_feature may be ignored with a large num
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^max_depth > num_leav
[LightGBM] [Info] Number of positive: 1644, number of negative: 6356
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000436 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`.
```