Credit card churn

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df=pd.read_csv('C:\\Users\\hp\\Downloads\\Churn_Modelling.csv')
df
```

```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender
Age  \
0             1    15634602   Hargrave          619    France  Female
42
1             2    15647311       Hill          608     Spain  Female
41
2             3    15619304       Onio          502    France  Female
42
3             4    15701354       Boni          699    France  Female
39
4             5    15737888   Mitchell          850     Spain  Female
43
...         ...         ...        ...          ...       ...     ...
...
9995       9996    15606229   Obijiaku          771    France    Male
39
9996       9997    15569892  Johnstone          516    France    Male
35
9997       9998    15584532        Liu          709    France  Female
36
9998       9999    15682355  Sabbatini          772   Germany    Male
42
9999      10000    15628319     Walker          792    France  Female
28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0          2       0.00              1          1               1
1          1   83807.86              1          0               1
2          8  159660.80              3          1               0
3          1       0.00              2          0               0
4          2  125510.82              1          1               1
...      ...        ...            ...        ...             ...
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
```

```
0           101348.88       1
1           112542.58       0
2           113931.57       1
3            93826.63       0
4            79084.10       0
...               ...     ...
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0

[10000 rows x 14 columns]
```

```
df.head()
```

```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age
\
0          1    15634602  Hargrave          619    France  Female   42

1          2    15647311      Hill          608     Spain  Female   41

2          3    15619304      Onio          502    France  Female   42

3          4    15701354      Boni          699    France  Female   39

4          5    15737888  Mitchell          850     Spain  Female   43


   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0
3       1       0.00              2          0               0
4       2  125510.82              1          1               1

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
3         93826.63       0
4         79084.10       0
```

```
df.tail()
```

```
      RowNumber  CustomerId    Surname  CreditScore Geography  Gender
Age  \
9995       9996    15606229   Obijiaku          771    France    Male
39
9996       9997    15569892  Johnstone          516    France    Male
35
```

```
9997       9998    15584532        Liu           709    France  Female
36
9998       9999    15682355  Sabbatini           772   Germany    Male
42
9999      10000    15628319     Walker           792    France  Female
28

      Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
9995       5       0.00              2          1               0
9996      10   57369.61              1          1               1
9997       7       0.00              1          0               1
9998       3   75075.31              2          1               0
9999       4  130142.79              1          1               0

      EstimatedSalary  Exited
9995         96270.64       0
9996        101699.77       0
9997         42085.58       1
9998         92888.52       1
9999         38190.78       0
```

```
df.shape
```

```
(10000, 14)
```

```
print("number of Rows", df.shape[0])
print("number of Columns", df.shape[1])
```

```
number of Rows 10000
number of Columns 14
```

```
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
```

```
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
```

```
df.describe()
```

```
         RowNumber    CustomerId   CreditScore           Age
Tenure  \
count  10000.00000  1.000000e+04  10000.000000  10000.000000
10000.000000
mean    5000.50000  1.569094e+07    650.528800     38.921800
5.012800
std     2886.89568  7.193619e+04     96.653299     10.487806
2.892174
min        1.00000  1.556570e+07    350.000000     18.000000
0.000000
25%     2500.75000  1.562853e+07    584.000000     32.000000
3.000000
50%     5000.50000  1.569074e+07    652.000000     37.000000
5.000000
75%     7500.25000  1.575323e+07    718.000000     44.000000
7.000000
max    10000.00000  1.581569e+07    850.000000     92.000000
10.000000

             Balance  NumOfProducts    HasCrCard  IsActiveMember  \
count   10000.000000   10000.000000  10000.00000    10000.000000
mean    76485.889288       1.530200      0.70550        0.515100
std     62397.405202       0.581654      0.45584        0.499797
min         0.000000       1.000000      0.00000        0.000000
25%         0.000000       1.000000      0.00000        0.000000
50%     97198.540000       1.000000      1.00000        1.000000
```

```
75%    127644.240000       2.000000      1.00000         1.000000
max    250898.090000       4.000000      1.00000         1.000000
```

```
       EstimatedSalary        Exited
count     10000.000000  10000.000000
mean     100090.239881      0.203700
std       57510.492818      0.402769
min          11.580000      0.000000
25%       51002.110000      0.000000
50%      100193.915000      0.000000
75%      149388.247500      0.000000
max      199992.480000      1.000000
```

```
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore',
'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

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

```
df.head()
```

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

   HasCrCard  IsActiveMember  EstimatedSalary  Exited
0          1               1        101348.88       1
1          0               1        112542.58       0
2          1               0        113931.57       1
3          0               0         93826.63       0
4          1               1         79084.10       0
```

```
# encoding categorical data
df['Geography'].unique()
```

```
array(['France', 'Spain', 'Germany'], dtype=object)
```

```
# from sklearn.preprocessing import OneHotEncoder

df = pd.get_dummies(df, drop_first=True)   # one-hot encode
df = df.astype(int)

df.head()
```

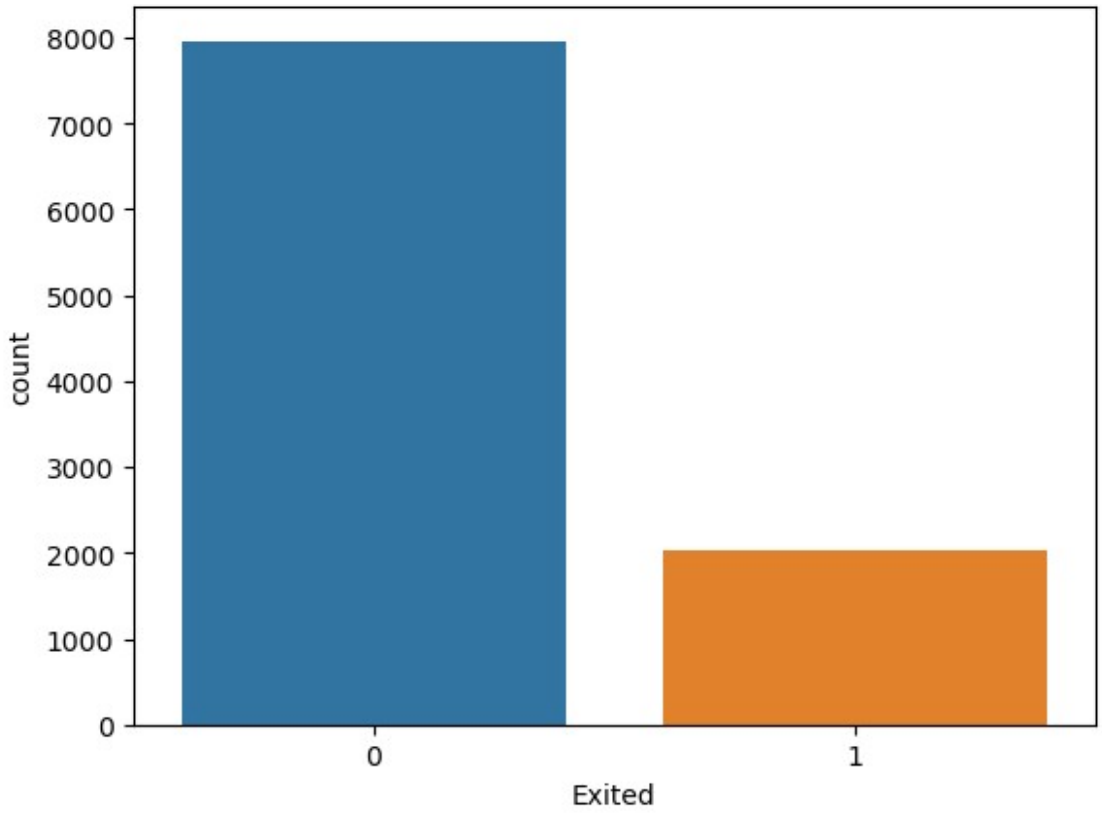
```
   CreditScore  Age  Tenure  Balance  NumOfProducts  HasCrCard  \
0          619   42       2        0              1          1
1          608   41       1    83807              1          0
2          502   42       8   159660              3          1
3          699   39       1        0              2          0
4          850   43       2   125510              1          1

   IsActiveMember  EstimatedSalary  Exited  Geography_Germany  \
0               1           101348       1                  0
1               1           112542       0                  0
2               0           113931       1                  0
3               0            93826       0                  0
4               1            79084       0                  0

   Geography_Spain  Gender_Male
0                0            0
1                1            0
2                0            0
3                0            0
4                1            0
```

```
# Not Handling Imbalanced
df['Exited'].value_counts()
```

```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

```
# data visualizaton
sns.countplot(x=df['Exited'])
# sns.countplot(x=df['Exited'])
```

```
<Axes: xlabel='Exited', ylabel='count'>
```




```
# data split
X=df.drop('Exited',axis=1)
y=df['Exited']
```

```
pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\programdata\
anaconda3\lib\site-packages (0.11.0)
Requirement already satisfied: numpy>=1.17.3 in c:\programdata\
anaconda3\lib\site-packages (from imbalanced-learn) (1.26.4)
Requirement already satisfied: scipy>=1.5.0 in c:\programdata\
anaconda3\lib\site-packages (from imbalanced-learn) (1.11.4)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\programdata\
anaconda3\lib\site-packages (from imbalanced-learn) (1.7.2)
Requirement already satisfied: joblib>=1.1.1 in c:\programdata\
anaconda3\lib\site-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\
anaconda3\lib\site-packages (from imbalanced-learn) (3.6.0)
Note: you may need to restart the kernel to use updated packages.
```

```
import sklearn
import imblearn

print("scikit-learn version:", sklearn.__version__)   # Should be >=
```

```
1.2
print("imblearn version:", imblearn.__version__)
```

```
scikit-learn version: 1.7.2
imblearn version: 0.14.0
```

```
from imblearn.over_sampling import SMOTE

X_res,y_res=SMOTE().fit_resample(X,y)

y_res.value_counts()
```

```
Exited
1    7963
0    7963
Name: count, dtype: int64
```

```
# splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_res, y_res, test_size=0.20, random_state=42
)

# X_train, X_test, y_train, y_test = train_test_split(
    # X, y, test_size=0.20, random_state=42, stratify=y
# )   # stratify use for data balance

# feature calling  data scalling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

# Fit on training data and transform both
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

X_train
```

```
array([[ 1.058568  ,  1.71508648,  0.68472287, ..., -0.57831252,
        -0.57773517,  0.90750738],
       [ 0.91362605, -0.65993547, -0.6962018 , ...,  1.72916886,
        -0.57773517,  0.90750738],
       [ 1.07927399, -0.18493108, -1.73189531, ...,  1.72916886,
        -0.57773517, -1.10191942],
       ...,
       [ 0.16821031, -0.18493108,  1.3751852 , ..., -0.57831252,
        -0.57773517, -1.10191942],
       [ 0.37527024, -0.37493284,  1.02995403, ..., -0.57831252,
         1.73089688,  0.90750738],
       [ 1.56586482,  1.14508121,  0.68472287, ..., -0.57831252,
         1.73089688,  0.90750738]])
```

```
# Logistic Regression

from sklearn.linear_model import LogisticRegression

lr=LogisticRegression()

lr = LogisticRegression(max_iter=1000)   # increase iterations if
needed
lr.fit(X_train, y_train)

LogisticRegression(max_iter=1000)

y_pred1 = lr.predict(X_test)

from sklearn.metrics import accuracy_score

accuracy = accuracy_score(y_test, y_pred1)   #use smote
print("Accuracy:", accuracy)

Accuracy: 0.7827997489014438

# accuracy = accuracy_score(y_test, y_pred)    #use stratify
# print("Accuracy:", accuracy)

Accuracy: 0.808

# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import StandardScaler
# from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import accuracy_score

# # Split the dataset
# X_train, X_test, y_train, y_test = train_test_split(
#     X, y, test_size=0.20, random_state=42, stratify=y
# )

# # Initialize StandardScaler
# sc = StandardScaler()

# # Fit on training data and transform both
# X_train = sc.fit_transform(X_train)
# X_test = sc.transform(X_test)

# # Train Logistic Regression
# model = LogisticRegression(max_iter=1000)   # increase iterations if
needed
# model.fit(X_train, y_train)

# # Predictions
# y_pred = model.predict(X_test)

# # Accuracy
```

```
# accuracy = accuracy_score(y_test, y_pred)
# print("Accuracy:", accuracy)
```

```
Accuracy: 0.808
```

```
from sklearn.metrics import precision_score,recall_score,f1_score
```

```
# precision_score(y_test,y_pred)
```

```
precision_score(y_test,y_pred)
```

```
0.8462515883100381
```

```
# recall_score(y_test,y_pred)
```

```
0.18673218673218672
```

```
recall_score(y_test,y_pred)
```

```
0.8576947842884739
```

```
# f1_score(y_test,y_pred)
```

```
0.2835820895522388
```

```
f1_score(y_test,y_pred)
```

```
0.8519347617524784
```

SVM

```
from sklearn import svm
```

```
svm=svm.SVC()
```

```
svm.fit(X_train,y_train)
```

```
SVC()
```

```
y_pred2=svm.predict(X_test)
```

```
accuracy_score(y_test,y_pred2)
```

```
0.8361581920903954
```

```
precision_score(y_test,y_pred2)
```

```
0.8323662153449387
```

KNeighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn=KNeighborsClassifier()
```

```
knn.fit(X_train,y_train)
KNeighborsClassifier()
y_pred3=knn.predict(X_test)
accuracy_score(y_test,y_pred3)
0.8157564344005022
precision_score(y_test,y_pred3)
0.7966830466830467
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier()
dt.fit(X_train,y_train)
DecisionTreeClassifier()
y_pred4=dt.predict(X_test)
accuracy_score(y_test,y_pred4)
0.7884494664155681
precision_score(y_test,y_pred4)
0.7691365584813227
```

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
RandomForestClassifier()
y_pred5=model.predict(X_test)
accuracy_score(y_test,y_pred5)
0.8562460765850597
precision_score(y_test,y_pred5)
0.8480610298792117
```

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(
    n_estimators=200,      # number of trees
    learning_rate=0.1,     # shrinkage step
    max_depth=3,           # depth of each tree
    random_state=42
)

gb.fit(X_train, y_train)
```

```
GradientBoostingClassifier(n_estimators=200, random_state=42)
```

```
y_pred6 = gb.predict(X_test)

precision_score(y_test,y_pred6)
```

```
0.8454545454545455
```

```
print("Accuracy:", accuracy_score(y_test, y_pred6))
```

```
Accuracy: 0.8465160075329566
```

```
final_Data=pd.DataFrame({'Modals':['lr','svm','knn','dt','rf','gb']
                         ,
                         'PRE':[precision_score(y_test,y_pred1),
                              precision_score(y_test,y_pred2),
                              precision_score(y_test,y_pred3),
                              precision_score(y_test,y_pred4),
                              precision_score(y_test,y_pred5),
                              precision_score(y_test,y_pred6)
                              ]})


final_Data
```

```
  Modals       PRE
0     lr  0.772641
1    svm  0.832366
2    knn  0.796683
3     dt  0.769137
4     rf  0.848061
5     gb  0.845455
```

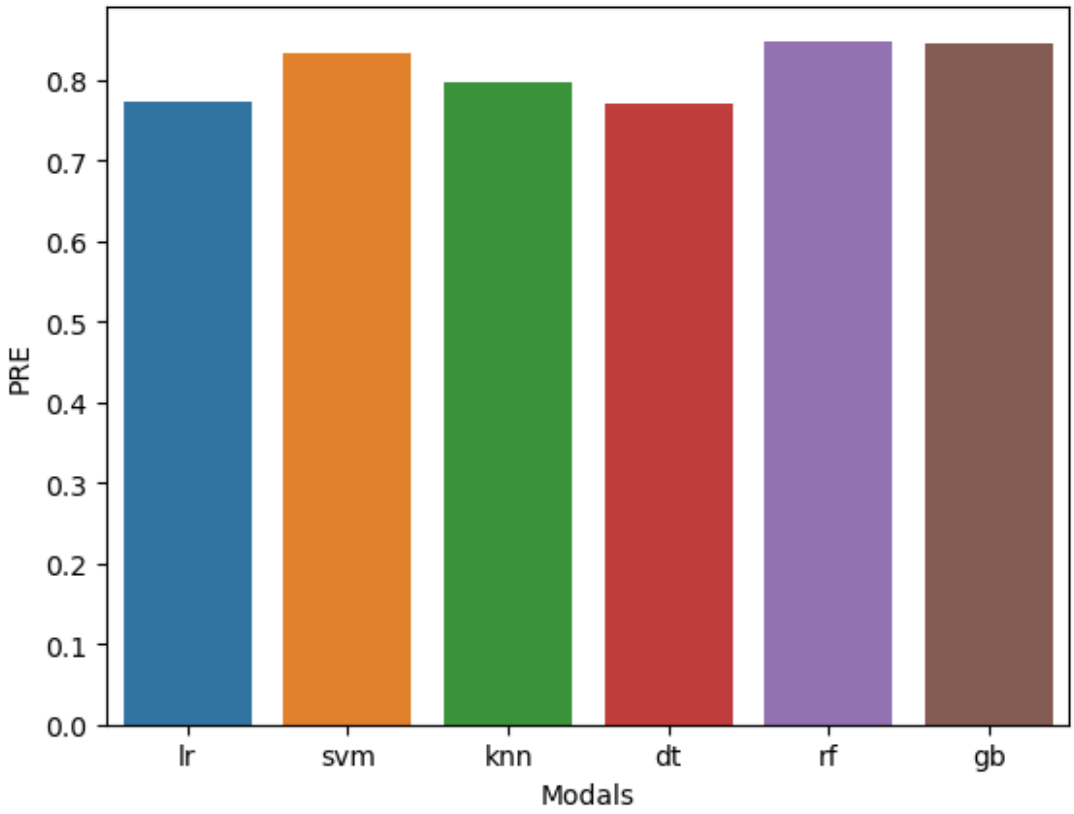
```
import seaborn as sns

final_Data['Modals']
```

```
0     lr
1    svm
2    knn
3     dt
```

```
4     rf
5     gb
Name: Modals, dtype: object
```

```
sns.barplot(x=final_Data['Modals'],y=final_Data['PRE'])
```

```
<Axes: xlabel='Modals', ylabel='PRE'>
```



Save The Model

```
X_res=sc.fit_transform(X_res)
```

```
rf.fit(X_res,y_res)
```

```
RandomForestClassifier()
```

```
import joblib
```

```
joblib.dump(rf,'churn_predict_model')
```

```
['churn_predict_model']
```

```
model=joblib.load('churn_predict_model')
```

```
df.columns
```

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts',
'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited',
'Geography_Germany',
       'Geography_Spain', 'Gender_Male'],
      dtype='object')
```

```
model.predict([[619,42,2,0.0,0,0,0,101348.88,0,0,0]])   #new data
```

```
array([1])
```