BANK CUSTOMER CHURN MODEL 🏦

OBJECTIVE

The main objective of the Bank Churn Project is to predict customer churn and provide actionable insights to improve customer retention. This can be broken down into the following goals:

1. Predict Customer Churn:

- Develop a predictive model to identify customers who are likely to leave the bank.

2. Understand Key Drivers of Churn:

- Analyze customer data to uncover patterns and factors (e.g., low engagement, high fees, poor service) that contribute to churn.

3. Increase Revenue and Customer Loyalty:

- Minimize revenue loss due to churn and strengthen long-term relationships with customers.

By achieving these objectives, the project enables the bank to proactively address churn, improve customer satisfaction, and enhance overall competitiveness.

DATA SOURCE

The data is taken from *ybi foundation github dataset (Bank Churn Modelling)*

IMPORT LIBRARIES

```
import pandas as pd
import seaborn as sns
```

IMPORT DATA

```
data = pd.read_csv('https://github.com/YBI-Foundation/Dataset/raw/refs/heads/main/Bank%20Churn%20Modelling.csv')
```

DESCRIBE DATA

```
data.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
0	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
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   Num Of Products   10000 non-null  int64  
 9   Has Credit Card   10000 non-null  int64  
 10  Is Active Member  10000 non-null  int64  
 11  Estimated Salary  10000 non-null  float64
 12  Churn             10000 non-null  int64  
dtypes: float64(2), int64(8), object(3)
memory usage: 1015.8+ KB
```

```
data.describe()
```

	CustomerId	CreditScore	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	10000.000000	10000.000000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	100090.239881	0.203700
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	57510.492818	0.402769
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	11.580000	0.000000
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	51002.110000	0.000000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	100193.915000	0.000000
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	149388.247500	0.000000
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	199992.480000	1.000000

```
data.shape
```

```
(10000, 13)
```

```
data.isnull().sum()
```

	0
CustomerId	0
Surname	0
CreditScore	0
Geography	0
Gender	0
Age	0
Tenure	0
Balance	0
Num Of Products	0
Has Credit Card	0
Is Active Member	0
Estimated Salary	0
Churn	0

dtype: int64

DATA PROCESSING

Drop irrelavent features

```
data.columns
```

```
Index(['CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age',
       'Tenure', 'Balance', 'Num Of Products', 'Has Credit Card',
       'Is Active Member', 'Estimated Salary', 'Churn'],
      dtype='object')
```

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

```
data.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Next steps: Generate code with data | View recommended plots | New interactive sheet

Encoding categorical data

```
data['Geography'].unique()
```

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

```
data = pd.get_dummies(data, drop_first = True)
```

```
data.head()
```

	CreditScore	Age	Tenure	Balance	Num Of Products	Has Credit Card	Is Active Member	Estimated Salary	Churn	Geography_Germany	Geography_Spain	Gender_Male
0	619	42	2	0.00	1	1	1	101348.88	1	False	False	False
1	608	41	1	83807.86	1	0	1	112542.58	0	False	True	False
2	502	42	8	159660.80	3	1	0	113931.57	1	False	False	False
3	699	39	1	0.00	2	0	0	93826.63	0	False	False	False
4	850	43	2	125510.82	1	1	1	79084.10	0	False	True	False

Next steps: Generate code with data | View recommended plots | New interactive sheet

DEFINE TARGET VARIABLE (y) AND FEATURE VARIABLE (X)

```
data.columns
```

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'Num Of Products',
       'Has Credit Card', 'Is Active Member', 'Estimated Salary', 'Churn',
       'Geography_Germany', 'Geography_Spain', 'Gender_Male'],
      dtype='object')
```

```
X = data.drop('Churn', axis = 1)
y = data['Churn']
```

```
X.shape,y.shape
```

```
((10000, 11), (10000,))
```

HANDLING IMBALANCE DATASET

- Class imbalance is a common problem in Machine Learning. So, just by running the model with imbalance data, we may get a higher accuracy but will fail to capture the minority class.

To handle this Imbalance data, we have Two technoques:

1. Oversampling :- We can over sample minority class with duplicates.
2. Undersampling :- We can randomly delete rows from majority class to match the minority class. But a Disadvantage of undersampling is that we can loose a lot of valuable data.

Thats the reason we mostly use oversampling technoque.

Imbalance in Data

```
data['Churn'].value_counts()
```

	count
Churn	
0	7963
1	2037

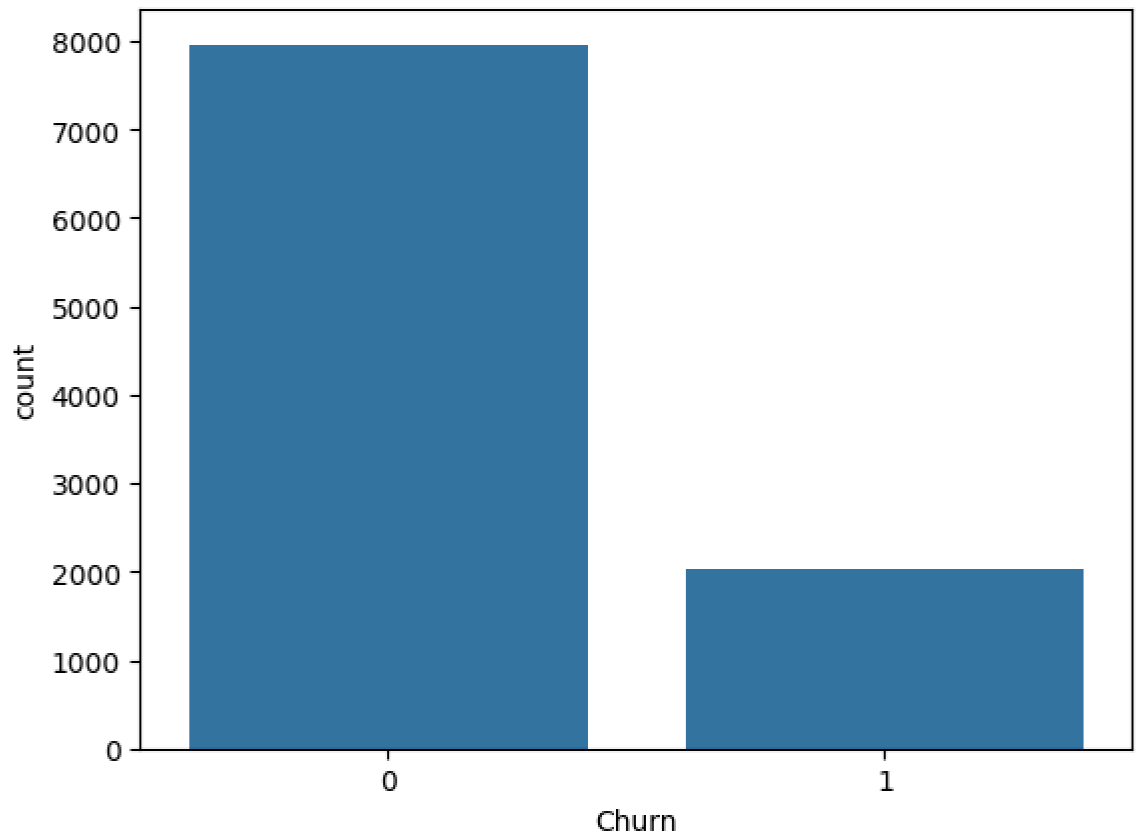
dtype: int64

```
X.shape, y.shape
```

```
((10000, 11), (10000,))
```

```
sns.countplot(x = 'Churn', data = data)
```

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



UNDER SAMPLING (X_rus, y_rus)

We are using a technique called *RandomUnderSampler*.

```
from imblearn.under_sampling import RandomUnderSampler
```

```
rus = RandomUnderSampler(random_state=2529)
```

```
X_rus, y_rus = rus.fit_resample(X,y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:484: FutureWarning: `BaseEstimator._check_n_features` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:493: FutureWarning: `BaseEstimator._check_feature_names` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validat
  warnings.warn(
```

Here we can see that data is reduced as we have removed some majority classes

```
X_rus.shape, y_rus.shape
```

```
((4074, 11), (4074,))
```

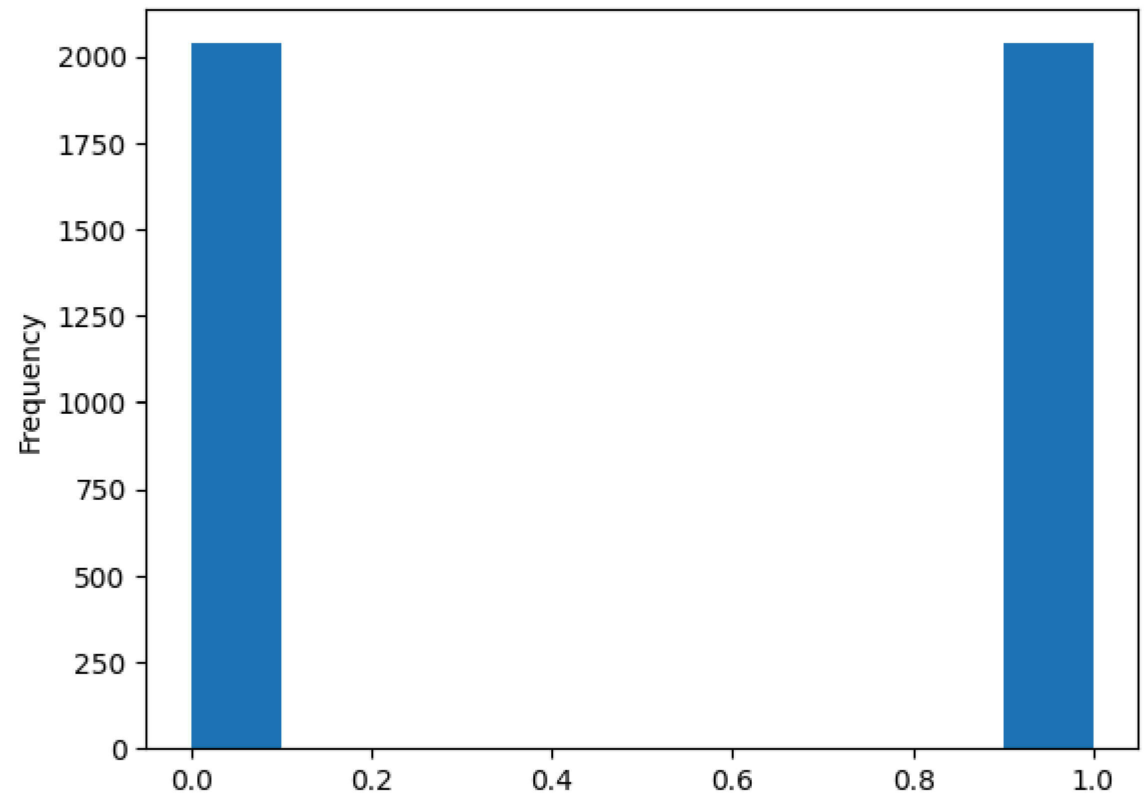
```
y_rus.value_counts()
```

	count
Churn	
0	2037
1	2037

dtype: int64

```
y_rus.plot(kind = 'hist')
```

<Axes: ylabel='Frequency'>



OVER SAMPLING (X_ros, y_ros)

Here also we have a technique called *RandomOverSampler*

But we are going to look at another very famous technique for over sampling that is ***SMOTE***

handling inbalanced data using *SMOTE (Synthetic Minority Oversampling Technique)*

```
from imblearn.over_sampling import SMOTE
```

```
ros = SMOTE(random_state=2529)
```

```
X_ros, y_ros = SMOTE().fit_resample(X, y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator._validate_data` is deprecated in 1.6 and will be removed in 1.7. Use `sklearn.utils.validation.va
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/utils/_tags.py:354: FutureWarning: The SMOTE or classes from which it inherits use `_get_tags` and `_more_tags`. Please define the `__sklear
  warnings.warn(
```

Here we can see the samples have been increased as the duplicate values have been added to the data set

```
X_ros.shape, y_ros.shape
```

```
((15926, 11), (15926,))
```

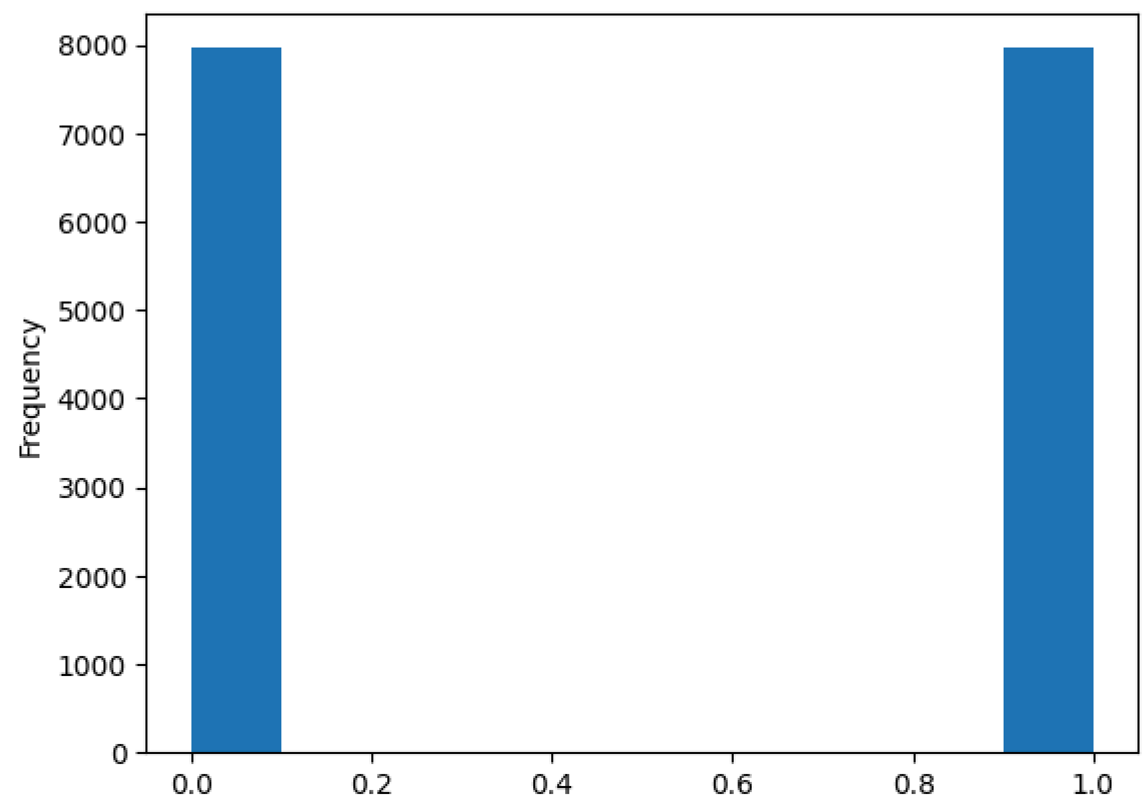
```
y_ros.value_counts()
```

	count
Churn	
1	7963
0	7963

dtype: int64

```
y_ros.plot(kind = 'hist')
```

<Axes: ylabel='Frequency'>



TRAIN TEST SPLIT

```
from sklearn.model_selection import train_test_split
```

Now we are going to use the Over Sampled data *(X_ros, y_ros)* as it is much more balanced and it also contains all the important majority and minority classes with duplicated

```
X_train, X_test, y_train, y_test = train_test_split(X_ros, y_ros, test_size = 0.2, random_state = 42)
```

DATA PROCESSING

FEATURE SCALING

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

MODELING AND MODEL EVALUATION

```
from sklearn.metrics import classification_report, confusion_matrix
```

Now we have to find the perfect model by using different models like *Logistic Regression*, *SVM*, *KNeighbors classifier*, *Decision Tree Classifier*, *Random Forest Classifier* and *Gradient Boosting Classifier*.

1. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
log = LogisticRegression()
```

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

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred_log = log.predict(X_test)
```

```
confusion_matrix(y_test, y_pred_log)
```

```
array([[1265,  368],
       [ 364, 1189]])
```

```
print(classification_report(y_test, y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.78      0.77      0.78      1633
           1       0.76      0.77      0.76      1553

    accuracy                           0.77      3186
   macro avg       0.77      0.77      0.77      3186
weighted avg       0.77      0.77      0.77      3186
```

2. SVM

```
from sklearn import svm
```

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

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

```
▾ SVC
SVC()
```

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

```
confusion_matrix(y_test, y_pred_svc)
```

```
array([[1396,  237],
       [ 281, 1272]])
```

```
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      1633
           1       0.84      0.82      0.83      1553

    accuracy                           0.84      3186
   macro avg       0.84      0.84      0.84      3186
weighted avg       0.84      0.84      0.84      3186
```

3. KNeighbors Classifier

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

```
knn = KNeighborsClassifier()
```

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

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred_knn = knn.predict(X_test)
```

```
confusion_matrix(y_test, y_pred_knn)
```

```
array([[1315,  318],
       [ 280, 1273]])
```

```
print(classification_report(y_test, y_pred_knn))
```

```
              precision    recall  f1-score   support

           0       0.82      0.81      0.81      1633
           1       0.80      0.82      0.81      1553

    accuracy                           0.81      3186
   macro avg       0.81      0.81      0.81      3186
weighted avg       0.81      0.81      0.81      3186
```

4. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()
```

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred_dt = dt.predict(X_test)
```

```
confusion_matrix(y_test, y_pred_dt)
```

```
array([[1280,  353],
       [ 299, 1254]])
```

```
print(classification_report(y_test, y_pred_dt))
```

```
              precision    recall  f1-score   support

           0       0.81      0.78      0.80      1633
           1       0.78      0.81      0.79      1553

    accuracy                           0.80      3186
   macro avg       0.80      0.80      0.80      3186
weighted avg       0.80      0.80      0.80      3186
```

5. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier()
```

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

```
▾ RandomForestClassifier
RandomForestClassifier()
```

```
y_pred_rf = rf.predict(X_test)
```

```
confusion_matrix(y_test, y_pred_rf)
```

```
array([[1410,  223],
       [ 223, 1330]])
```



```
print(classification_report(y_test, y_pred_rf))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      1633
           1       0.86      0.86      0.86      1553

    accuracy                           0.86      3186
   macro avg       0.86      0.86      0.86      3186
weighted avg       0.86      0.86      0.86      3186
```

6. Gradient boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
gbc = GradientBoostingClassifier()
```

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

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

```
y_pred_gbc = gbc.predict(X_test)
```

```
confusion_matrix(y_test, y_pred_gbc)
```

```
array([[1384,  249],
       [ 276, 1277]])
```

```
print(classification_report(y_test, y_pred_gbc))
```

```
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      1633
           1       0.84      0.82      0.83      1553
```

```
    accuracy                           0.84      3186
   macro avg       0.84      0.83      0.84      3186
weighted avg       0.84      0.84      0.84      3186
```

VISUALIZATION

Visualizing the model performance to select the Best model

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

ACCURACY

```
final_data = pd.DataFrame({'Models':['LR','SVM','KNN','DT','RF','GBC'],
                           'Accuracy':[accuracy_score(y_test,y_pred_log),
                                       accuracy_score(y_test,y_pred_svc),
                                       accuracy_score(y_test,y_pred_knn),
                                       accuracy_score(y_test,y_pred_dt),
                                       accuracy_score(y_test,y_pred_rf),
                                       accuracy_score(y_test,y_pred_gbc)]})
```

```
final_data
```

	Models	Accuracy
0	LR	0.770245
1	SVM	0.837414
2	KNN	0.812304
3	DT	0.795355
4	RF	0.860013
5	GBC	0.835217

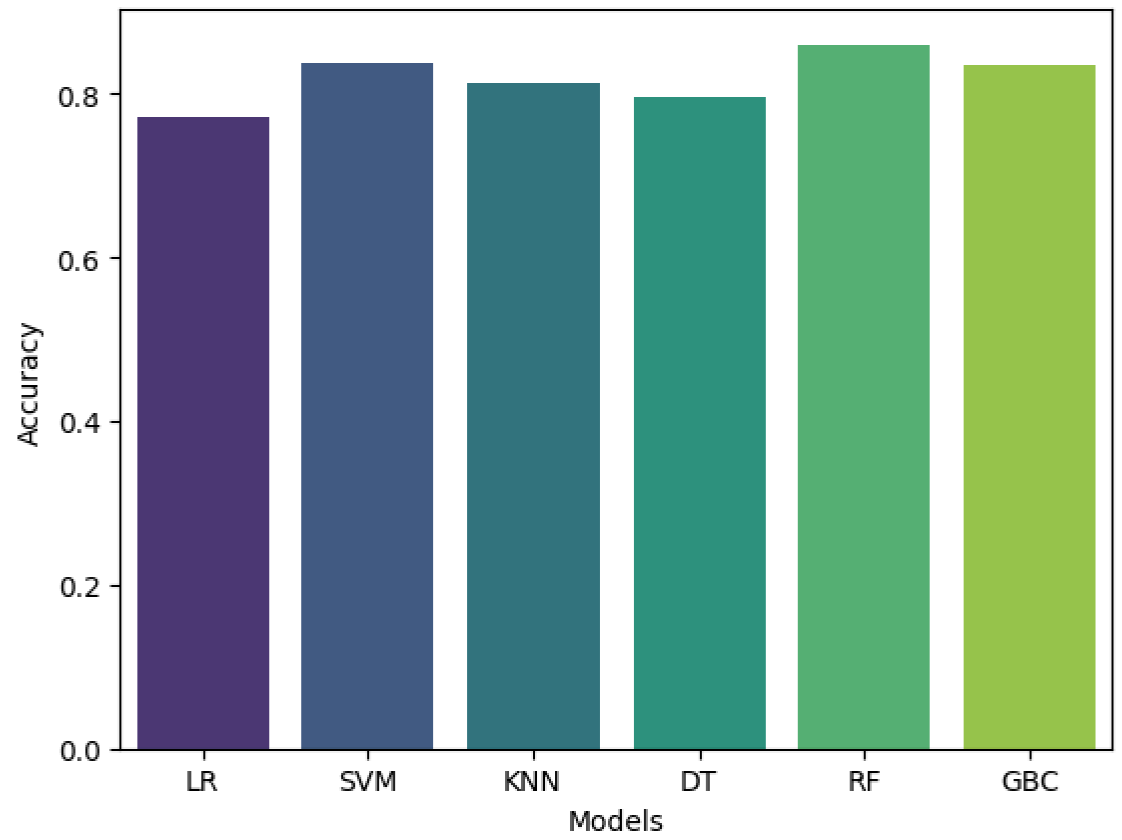
Next steps: Generate code with final_data | View recommended plots | New interactive sheet

```
palette = sns.color_palette("viridis", n_colors=len(final_data))
sns.barplot(x='Models', y='Accuracy', data=final_data, palette=palette)
```

```
<ipython-input-304-14d36ebc846c>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Models', y='Accuracy', data=final_data, palette=palette)
<Axes: xlabel='Models', ylabel='Accuracy'>
```



RECALL

```
final_data = pd.DataFrame({'Models':['LR','SVM','KNN','DT','RF','GBC'],
                           'Recall':[recall_score(y_test,y_pred_log),
                                     recall_score(y_test,y_pred_svc),
                                     recall_score(y_test,y_pred_knn),
                                     recall_score(y_test,y_pred_dt),
                                     recall_score(y_test,y_pred_rf),
                                     recall_score(y_test,y_pred_gbc)]})
```

```
final_data
```

	Models	Recall
0	LR	0.765615
1	SVM	0.819060
2	KNN	0.819704
3	DT	0.807469
4	RF	0.856407
5	GBC	0.822279

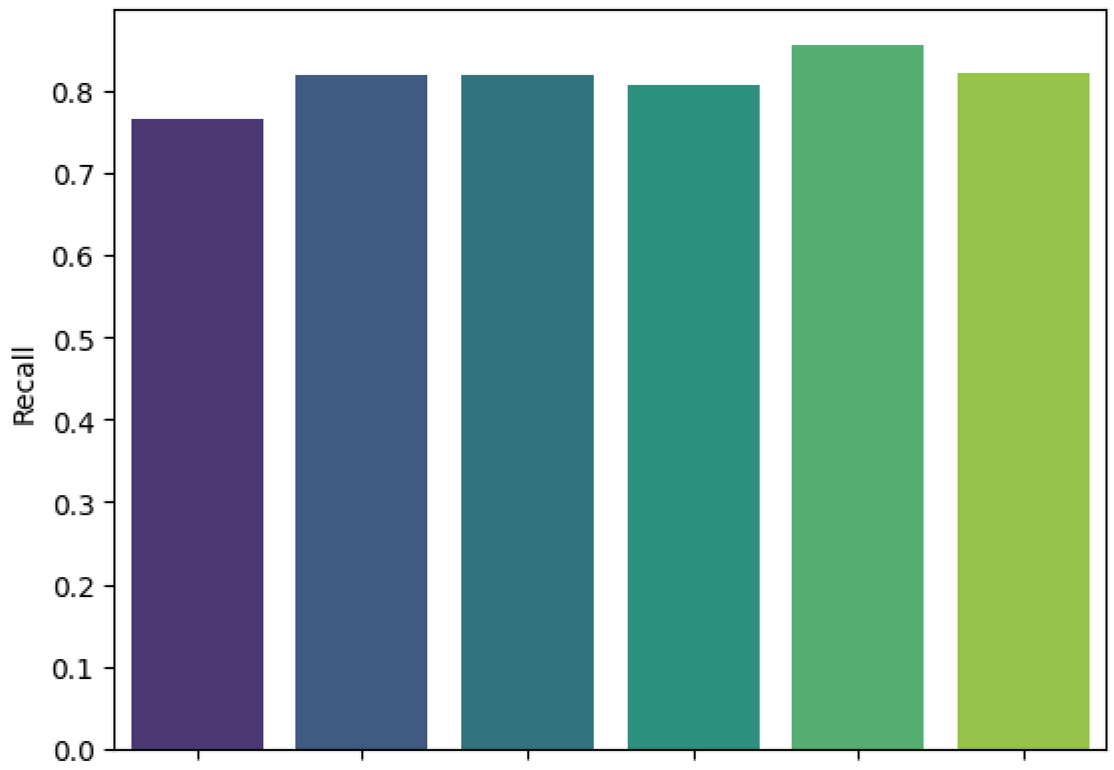
Next steps: Generate code with final_data | View recommended plots | New interactive sheet

```
palette = sns.color_palette("viridis", n_colors=len(final_data))
sns.barplot(x='Models', y='Recall', data=final_data, palette=palette)
```

```
<ipython-input-310-970a4b5b86e5>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x='Models', y='Recall', data=final_data, palette=palette)
<Axes: xlabel='Models', ylabel='Recall'>
```



From the above visualizations, we can say that the *RANDOM FOREST CLASSIFIER* have performed best in all categories. As we cannot only look at the accuracy of the model, but have to look at other parameters as well.

Now lets Save the model as we have to make predictions with the best model that is *Random Forest Classifier*

```
X_ros = sc.fit_transform(X_ros)
```

```
rf.fit(X_ros, y_ros)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

joblib

- It is a Python library used for saving and loading large Python objects, such as machine learning models or datasets, efficiently. It is optimized for serialization and allows for compression of large files, making it ideal for saving trained models and large arrays.

Key Functions:

- dump(): Save an object to a file.
- load(): Load an object from a file.

```
import joblib
```

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

```
['Churn_predict_model']
```

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

PREDICTION

```
data.columns
```

```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'Num Of Products',
       'Has Credit Card', 'Is Active Member', 'Estimated Salary', 'Churn',
       'Geography_Germany', 'Geography_Spain', 'Gender_Male'],
      dtype='object')
```

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

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

We have successfully predicted the given values.

EXPLANATION

The above model is for *Bank Customer Churn Model* with *Random Forest Classifier.*

We had to use the Over Sampling data because if we use the normal inbalance data we will get a good ACCURACY (82%, 77%, ...) but we will not get a proper RECALL (20%, 25%, ...) which implies that our model is good at predict the churning. Eventhough accuracy is good but the recall of the interested category i.e; churn is not good.

So by doing that we have got the best prediction model with Random Forest Classifie