Suggested code may be subject to a license | 3arii/LogReg-GUI | Abhishek3689/Player_Selection_Prediction | Abhishekjha111/textmassage-classifier | fmakayi/disaster_response_pipeline_project | Abrar

```
# Importing the Libararies

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, precision_score, recall_score, f1_score
```

```
df = pd.read_csv('Churn_Modelling.csv')
```

```
df.head()
```

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

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

```
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
```

```
# Checking Null Values

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

	0
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

```
# Checking Duplicates

df.duplicated().sum()
```

```
0
```

```
# Converting Categorical data into numeric

label_encoder = LabelEncoder()
df['Gender']= label_encoder.fit_transform(df['Gender'])
df = pd.get_dummies(df, columns=['Geography'], drop_first= True)
```

```
# Checking the head again for feature selection as per logical understanding of data
# Gender has been changed to 0 & 1
# Geography coulmn has changed too

df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	1	15634602	Hargrave	619	0	42	2	0.00	1	1	1	1013
1	2	15647311	Hill	608	0	41	1	83807.86	1	0	1	1125
2	3	15619304	Onio	502	0	42	8	159660.80	3	1	0	1139
3	4	15701354	Boni	699	0	39	1	0.00	2	0	0	938
4	5	15737888	Mitchell	850	0	43	2	125510.82	1	1	1	790

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

```
# Feature Selection

features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Gender', 'Ge

X = df[features]
y = df['Exited']
```

```
# Spliting data into testing and training

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Feature Scaling

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Checking Scaled Values

X_train[:5], X_test[:5]
```

```
(array([[ 0.35649971, -0.6557859 ,  0.34567966, -1.21847056,  0.80843615,
          0.64920267,  0.97481699,  1.36766974,  0.91324755, -0.57638802,
         -0.57946723],
        [-0.20389777,  0.29493847, -0.3483691 ,  0.69683765,  0.80843615,
          0.64920267,  0.97481699,  1.6612541 ,  0.91324755, -0.57638802,
          1.72572313],
        [-0.96147213, -1.41636539, -0.69539349,  0.61862909, -0.91668767,
          0.64920267, -1.02583358, -0.25280688,  0.91324755,  1.73494238,
         -0.57946723],
        [-0.94071667, -1.13114808,  1.38675281,  0.95321202, -0.91668767,
          0.64920267, -1.02583358,  0.91539272, -1.09499335, -0.57638802,
         -0.57946723],
        [-1.39733684,  1.62595257,  1.38675281,  1.05744869, -0.91668767,
         -1.54035103, -1.02583358, -1.05960019,  0.91324755, -0.57638802,
         -0.57946723]]),
 array([[-0.57749609, -0.6557859 , -0.69539349,  0.32993735,  0.80843615,
         -1.54035103, -1.02583358, -1.01960511,  0.91324755, -0.57638802,
          1.72572313],
        [-0.29729735,  0.3900109 , -1.38944225, -1.21847056,  0.80843615,
          0.64920267,  0.97481699,  0.79888291,  0.91324755, -0.57638802,
         -0.57946723],
        [-0.52560743,  0.48508334, -0.3483691 , -1.21847056,  0.80843615,
          0.64920267, -1.02583358, -0.72797953, -1.09499335,  1.73494238,
         -0.57946723],
        [-1.51149188,  1.91116988,  1.03972843,  0.68927246,  0.80843615,
          0.64920267,  0.97481699,  1.22138664,  0.91324755, -0.57638802,
          1.72572313],
        [-0.9510944 , -1.13114808,  0.69270405,  0.78283876, -0.91668767,
          0.64920267,  0.97481699,  0.24756011, -1.09499335,  1.73494238,
         -0.57946723]]))
```

```
# Randomn Forest

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
# Predicting

y_pred = model.predict(X_test)
```

```
# Confusion Matrix, Accuracy and Classification Report

conf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)


print("Confusion Matrix:")
print(conf_matrix)
print("\nAccuracy:", accuracy)
print("\nClassification Report:")
print(classification_rep)
```

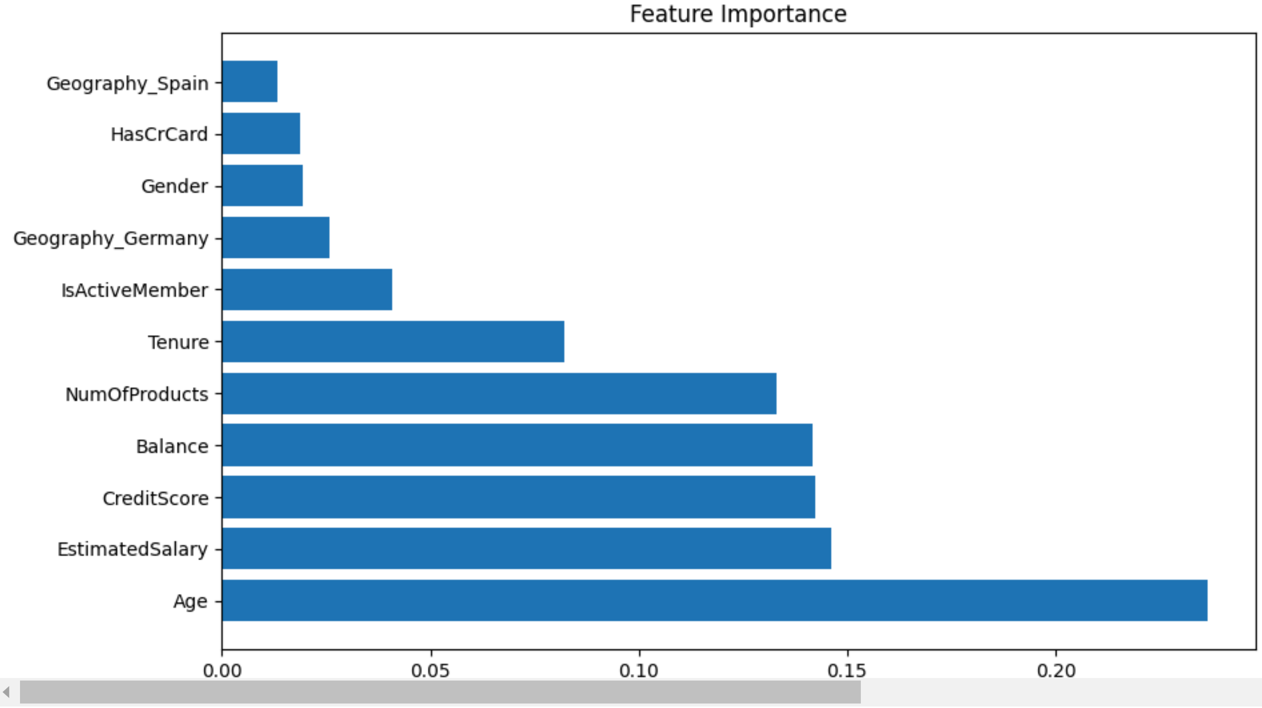
```
Confusion Matrix:
[[1554   53]
 [ 208  185]]

Accuracy: 0.8695

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1607
           1       0.78      0.47      0.59       393

    accuracy                           0.87      2000
   macro avg       0.83      0.72      0.75      2000
weighted avg       0.86      0.87      0.86      2000
```

```
# Feature Importance

importances = model.feature_importances_
indices = np.argsort(importances)[::-1]
names = [features[i] for i in indices]
```

```
plt.figure(figsize=(10, 6))
plt.title("Feature Importance")
plt.barh(range(X.shape[1]), importances[indices])
plt.yticks(range(X.shape[1]), names)
plt.show
```

```
matplotlib.pyplot.show
def show(*args, **kwargs)
```

```
**Auto-show in jupyter notebooks**

The jupyter backends (activated via ``%matplotlib inline``,
``%matplotlib notebook``, or ``%matplotlib widget``), call ``show()`` at
the end of every cell by default. Thus, you usually don't have to call it
explicitly there.
```




```
# Now lets apply Logistic Regression to compare

from sklearn.linear_model import LogisticRegression

# Build and Train the Logistic Regression Model
log_reg = LogisticRegression(random_state=42)
log_reg.fit(X_train, y_train)

# Make Predictions
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate the model
conf_matrix_log_reg = confusion_matrix(y_test, y_pred_log_reg)
accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)
classification_rep_log_reg = classification_report(y_test, y_pred_log_reg)

print("Logistic Regression - Confusion Matrix:")
print(conf_matrix_log_reg)
print("\nLogistic Regression - Accuracy:", accuracy_log_reg)
print("\nLogistic Regression - Classification Report:")
print(classification_rep_log_reg)
```

```
Logistic Regression - Confusion Matrix:
[[1543   64]
 [ 314   79]]

Logistic Regression - Accuracy: 0.811

Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.96      0.89      1607
           1       0.55      0.20      0.29       393

    accuracy                           0.81      2000
   macro avg       0.69      0.58      0.59      2000
weighted avg       0.78      0.81      0.77      2000
```

```
# Now lets apply SVM to compare

from sklearn.svm import SVC

# Build and Train the SVM Model
svm_model = SVC(kernel ='linear' ,random_state=42)
svm_model.fit(X_train, y_train)

# Make Predictions
y_pred_svm = svm_model.predict(X_test)

# Evaluate the SVM Model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
classification_rep_svm = classification_report(y_test, y_pred_svm)
confusion_matrix= confusion_matrix(y_test, y_pred_svm)

print("SVM - Confusion Matrix:")
print(confusion_matrix)
print("\nSVM - Accuracy:", accuracy_svm)
print("\nSVM - Classification Report:")
print(classification_rep_svm)

# As per the warnings this model was not able to predict properly as for some lables it did not predict any samples.
```

```
SVM - Confusion Matrix:
[[1607    0]
 [ 393    0]]

SVM - Accuracy: 0.8035

SVM - Classification Report:
              precision    recall  f1-score   support

           0       0.80      1.00      0.89      1607
           1       0.00      0.00      0.00       393

    accuracy                           0.80      2000
   macro avg       0.40      0.50      0.45      2000
weighted avg       0.65      0.80      0.72      2000

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are i
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# Lets try KNN Model Now

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

# Build a model
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)

# Make Predictions
y_pred_knn = knn_model.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
class_rep_knn = classification_report(y_test, y_pred_knn)
conf_matrix_knn= confusion_matrix(y_test, y_pred_knn)

print("KNN - Confusion Matrix:")
print(conf_matrix_knn)
print("\nKNN - Accuracy:", accuracy_knn)
print("\nKNN - Classification Report:")
print(class_rep_knn)
```

```
KNN - Confusion Matrix:
[[1514   93]
 [ 247  146]]

KNN - Accuracy: 0.83

KNN - Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      1607
           1       0.61      0.37      0.46       393

    accuracy                           0.83      2000
   macro avg       0.74      0.66      0.68      2000
```

```
weighted avg       0.81      0.83      0.81      2000
```

```
# Lets apply Gradient Boosting Classifier now

from sklearn.ensemble import GradientBoostingClassifier

# Build and Train the Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb_model.fit(X_train, y_train)

# Make Predictions
y_pred_gb = gb_model.predict(X_test)

# Evaluate the model
accuracy_gb = accuracy_score(y_test, y_pred_gb)
classification_rep_gb = classification_report(y_test, y_pred_gb)
confusion_matrix= confusion_matrix(y_test, y_pred_gb)

print("Gradient Boosting - Confusion Matrix:")
print(confusion_matrix)
print("\nGradient Boosting - Accuracy:", accuracy_gb)
print("\nGradient Boosting - Classification Report:")
print(classification_rep_gb)
```

```
Gradient Boosting - Confusion Matrix:
[[1543   64]
 [ 201  192]]

Gradient Boosting - Accuracy: 0.8675

Gradient Boosting - Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      1607
           1       0.75      0.49      0.59       393

    accuracy                           0.87      2000
   macro avg       0.82      0.72      0.76      2000
weighted avg       0.86      0.87      0.86      2000
```

```
# Feature Engineering

df = pd.read_csv('Churn_Modelling.csv')


# Binary feature for balance
df['Zero_Balance']=(df['Balance']==0).astype(int)


# Age Groups

df['Age_Group'] = pd.cut(df['Age'], bins=[18, 25, 35, 45, 55, 65, 75, 85, 95], labels=['18-25', '26-35', '36-45', '46-55', '56-65', '66-



# Balance to Salary Ratio

df['BSRatio']= df['Balance']/df['EstimatedSalary']


# Interaction Feature between Numofproducts and Isactivmember

df['ProductUsage']= df['NumOfProducts']*df['IsActiveMember']


# Tenure Grouping

df['Tenure_Group']=pd.cut(df['Tenure'], bins=[0, 2,5,7,10], labels=['0-2', '3-5', '6-7', '8-10'])


label_encoder = LabelEncoder()
df['Gender']= label_encoder.fit_transform(df['Gender'])
df = pd.get_dummies(df, columns=['Geography'], drop_first= True)
df['Male_Germany']= df['Gender']*df['Geography_Germany']
df['Male_Spain']= df['Gender']*df['Geography_Spain']


df = pd.get_dummies(df, columns=['Age_Group','Tenure_Group'], drop_first= True)


features = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Gender', 'Ge
```

```
X = df[features]
y = df['Exited']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
from sklearn.metrics import confusion_matrix

cnf_matrix = confusion_matrix(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
classification_rep = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(cnf_matrix)
print("\nAccuracy:", accuracy)
print("\nClassification Report:")
print(classification_rep)
```

```
Confusion Matrix:
[[1554   53]
 [ 208  185]]

Accuracy: 0.8695

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1607
           1       0.78      0.47      0.59       393

    accuracy                           0.87      2000
   macro avg       0.83      0.72      0.75      2000
weighted avg       0.86      0.87      0.86      2000
```