In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
#Loading data data
df= pd.read_csv(r"D:\AI\data\Bank_churn_modelling.csv")
df.shape
```

Out[2]:
```
(10000, 14)
```

Data Exploration

In [3]:
```
df.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	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	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	

In [4]:
```
# unwanted columns - Rownumber, customerid, surname
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
RowNumber          10000 non-null int64
CustomerId         10000 non-null int64
Surname            10000 non-null object
CreditScore        10000 non-null int64
Geography          10000 non-null object
Gender             10000 non-null object
Age                10000 non-null int64
Tenure             10000 non-null int64
Balance            10000 non-null float64
NumOfProducts      10000 non-null int64
HasCrCard          10000 non-null int64
IsActiveMember     10000 non-null int64
EstimatedSalary    10000 non-null float64
Exited             10000 non-null int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

In [6]:
```
df.Geography.unique()
```

Out[6]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [7]:
```
df.Gender.unique()
```

Out[7]:
```
array(['Female', 'Male'], dtype=object)
```

In [8]:
```
df.describe()
```

Out[8]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.00000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.51510
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.49979
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.00000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.00000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.00000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.00000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.00000

Data Cleaning

In [9]:
```
#remove the unwanted columns
print(df.shape)
df.drop(["RowNumber","CustomerId","Surname"],axis=1,inplace=True)
print(df.shape)
```

```
(10000, 14)
(10000, 11)
```

In [10]:
```
#check for missing values
df.isnull().sum()
```

Out[10]:
```
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [11]:
```
#check for duplicates
df.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```
# check for outliers
df.skew()
```

Out[12]:
```
CreditScore       -0.071607
Age                1.011320
Tenure             0.010991
Balance           -0.141109
NumOfProducts      0.745568
HasCrCard         -0.901812
IsActiveMember    -0.060437
EstimatedSalary    0.002085
Exited             1.471611
dtype: float64
```

Feature Engineering

- Feature Extraction

- Feature Selection

Feature Selection

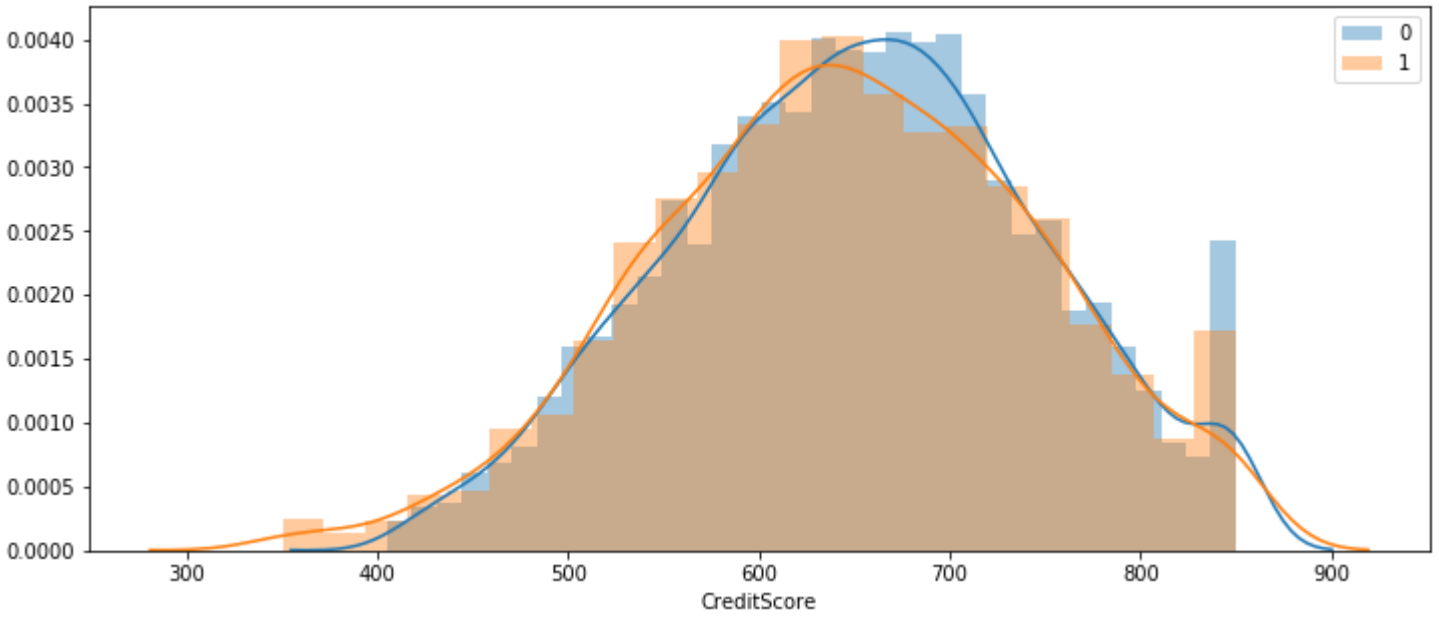
Using Data Visualization

In [13]:
```
df.columns
```

Out[13]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

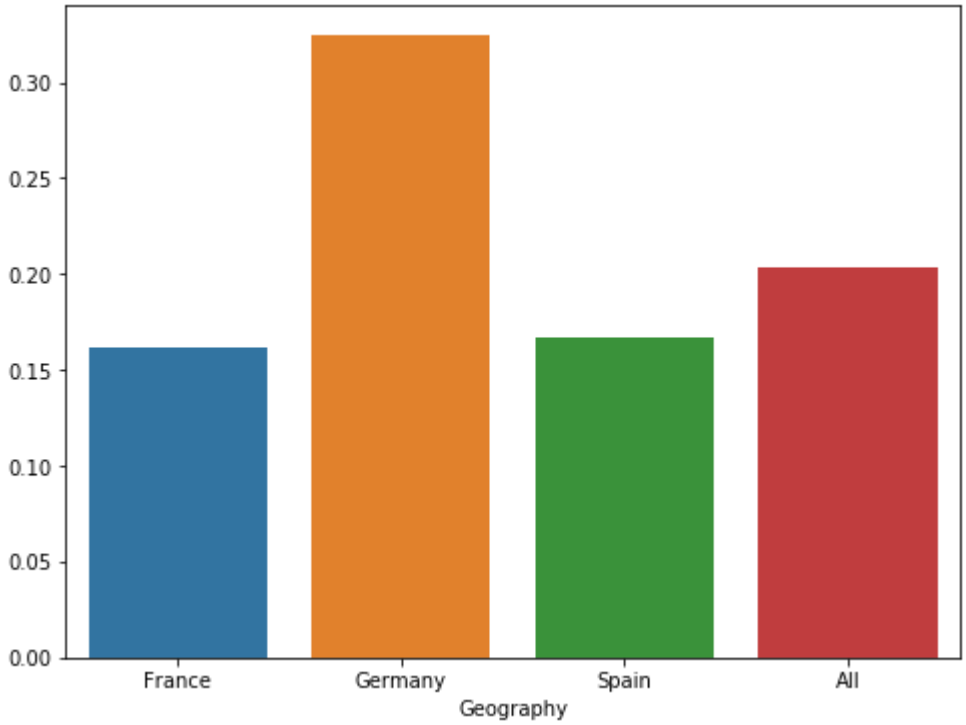
In [14]:
```
numerics = ['CreditScore','Age', 'Tenure', 'Balance','NumOfProducts', 'EstimatedSalary']
categorics = ['Geography', 'Gender','HasCrCard', 'IsActiveMember']
```

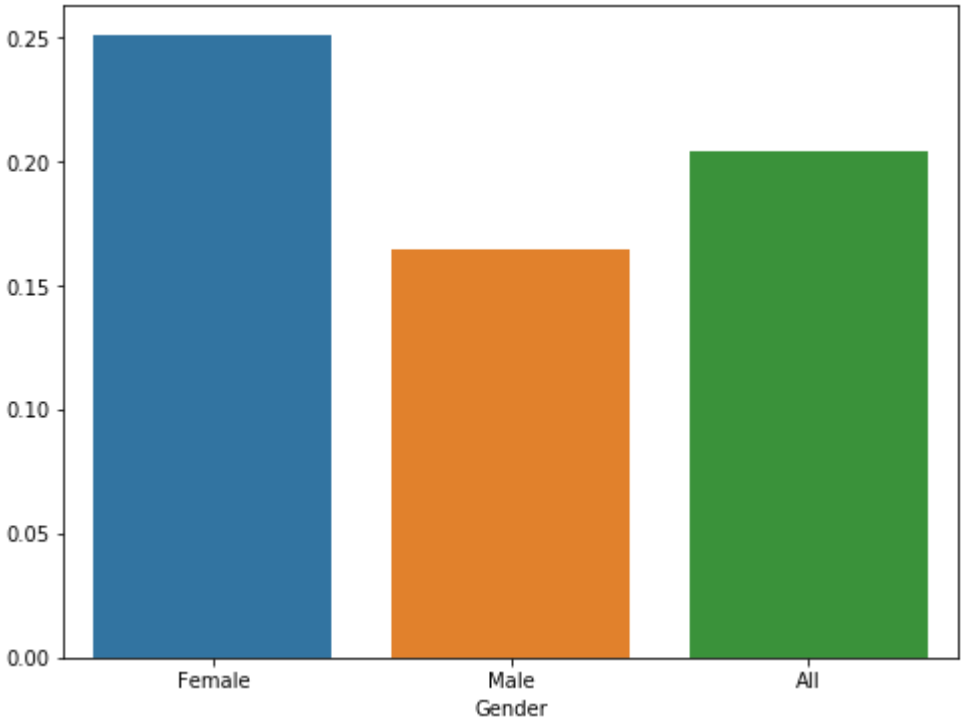
In [15]:
```
# Distplot for numeric v/s categorical(Exited)
for col in numerics:
    plt.figure(figsize=(12,5))
    sns.distplot(df[col][df.Exited==0])
    sns.distplot(df[col][df.Exited==1])
    plt.legend(['0','1'])
    plt.show()
```

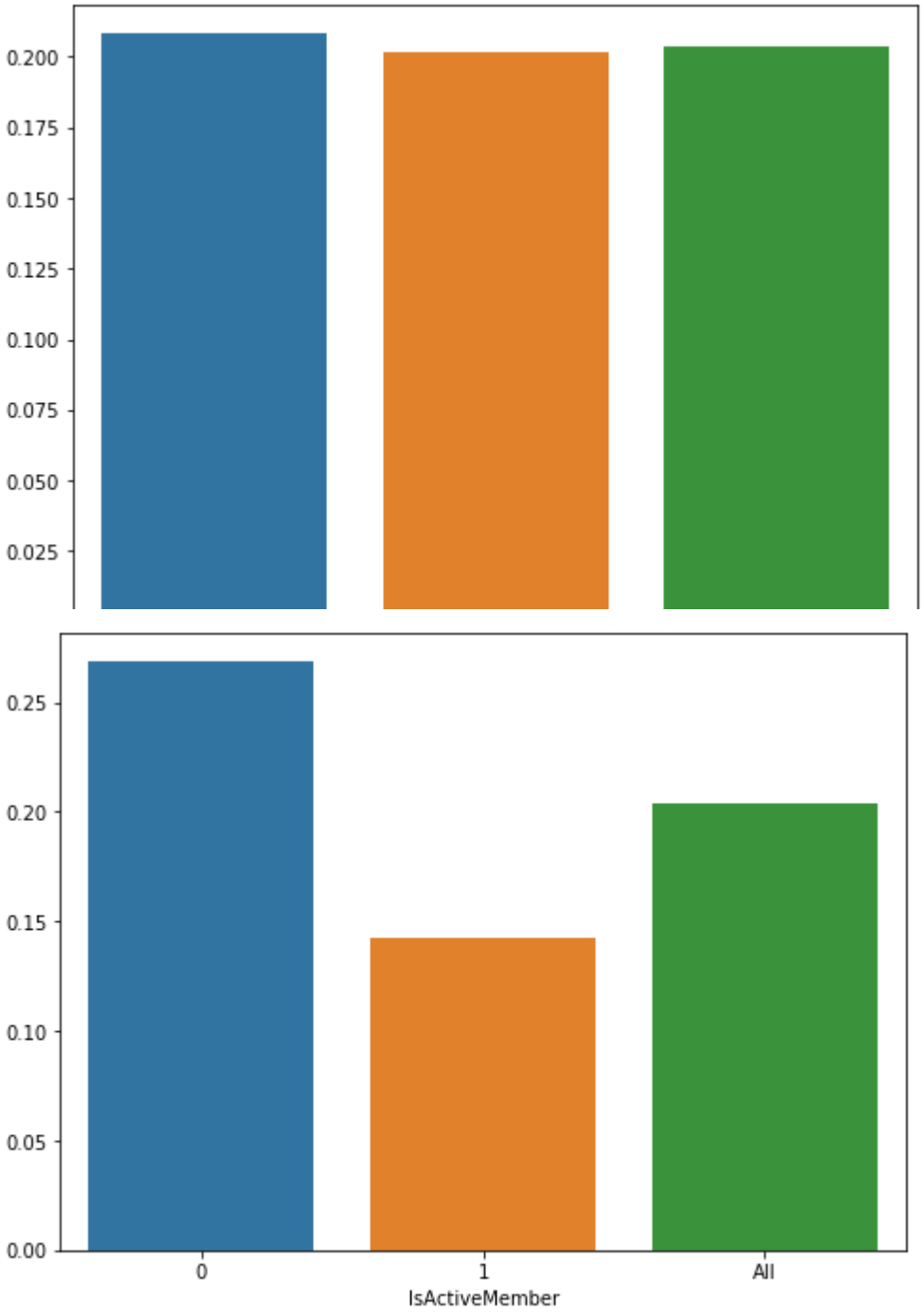


In [16]:
```
# for categorical features

for col in categorics:
    table = pd.crosstab(df[col],df["Exited"],margins=True)
    dropout_rate = table[1]/table["All"]
    plt.figure(figsize=(8,6))
    sns.barplot(dropout_rate.index,dropout_rate.values)
    plt.show()
```




0.25
0.20
0.15
0.10
0.05
0.00
Female
Male
All
Gender


0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.25
0.20
0.15
0.10
0.05
0.00
0
1
All
IsActiveMember

In [17]:
```
table = pd.crosstab(df.Geography,df.Exited,margins=True)
table
```

Out[17]:

Exited	0	1	All
Geography			
France	4204	810	5014
Germany	1695	814	2509
Spain	2064	413	2477
All	7963	2037	10000

In [18]:
```
table[1]/table["All"]
```

Out[18]:
```
Geography
France     0.161548
Germany    0.324432
Spain      0.166734
All        0.203700
dtype: float64
```

In [19]:
```
df.columns
```

Out[19]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

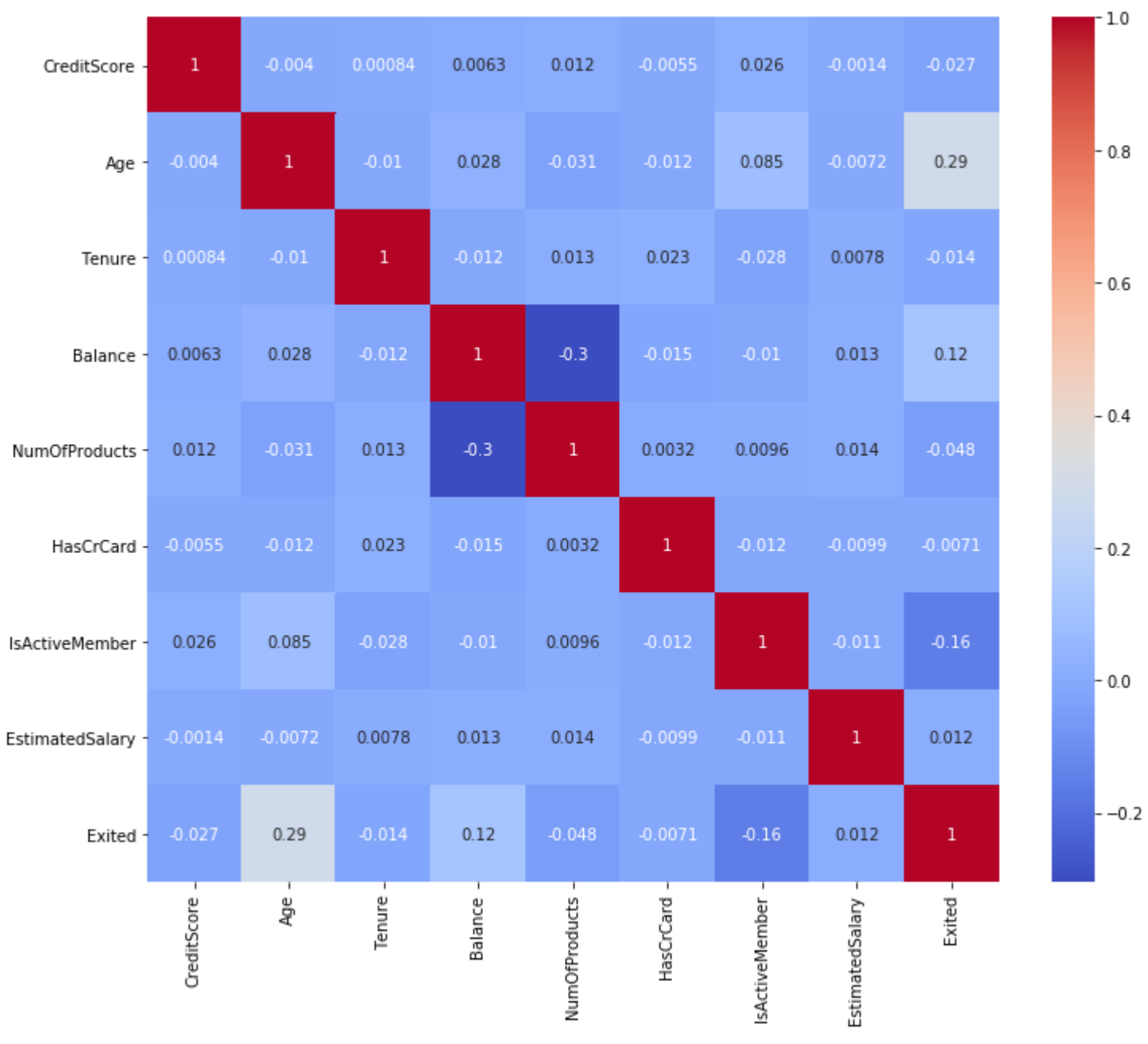
Important features - Geography, Gender, Age, Balance, Numofproducts, IsActiveMember

Using statistics

Correlation Analysis

In [20]:

```
cor = df.corr()
plt.figure(figsize=(12,10))
ax = sns.heatmap(cor,annot=True,cmap="coolwarm")
top,bot = ax.get_ylim()
ax.set_ylim(top+0.5,bot-0.5)
plt.show()
```



important features - Age, Balance, Numofproducts

ANOVA

ANOVA = Analysis of variance – compare means / variation in two or more samples

Sample1 – an array having data of age of customers who left the bank

Sample2 – an array having data of age of customers who didn't leave the bank

---

For ANOVA -

Null Hypothesis – H0 = means of two samples are similar to each other

Alternate Hypothesis – Ha = means of two samples are different than each other

- If the business confidence is = 95% = 0.95
- Significance level = 5% = 0.05

If pvalue > 0.05 = we accept the H0 (Null Hypothesis) = feature is not important

If pvalue <0.05 = we reject the H0 (Null Hypothesis) = feature is important

In [21]:
```
xnum = df[numerics]
y = df['Exited']

from sklearn.feature_selection import f_classif
fval,pval = f_classif(xnum,y)
for i in range(len(xnum.columns)): print(numerics[i]," ",pval[i])
```

```
CreditScore   0.006738213892258643
Age   1.2399313093415039e-186
Tenure   0.1615268494952801
Balance   1.275563319153163e-32
NumOfProducts   1.7173330048040421e-06
EstimatedSalary   0.22644042802376574
```

Chi Square Test

Chi Square Test - comparision of distribution of categories among multiple groups

Sample1 – distribution of geography for customers who left

Sample2 – distrbution of geography for customers who didn't leave

---

For Chiq Square -

Null Hypothesis – H0 = two groups have similar distribution

Alternate Hypothesis – Ha = two groups have different distribution.

- If the business confidence is = 95% = 0.95
- Significance level = 5% = 0.05

If pvalue > 0.05 = we accept the H0 (Null Hypothesis) = feature is not important

If pvalue <0.05 = we reject the H0 (Null Hypothesis) = feature is important

In [22]:
```
xcat = df[categorics]
from sklearn.preprocessing import LabelEncoder
xcat.Geography = LabelEncoder().fit_transform(xcat.Geography)
xcat.Gender = LabelEncoder().fit_transform(xcat.Gender)

from sklearn.feature_selection import chi2
cval,pval = chi2(xcat,y)
for i in range(len(categorics)): print(categorics[i]," ",pval[i])
```

```
Geography   0.0005756078382573235
Gender   7.015574513879596e-13
HasCrCard   0.6984962089530451
IsActiveMember   1.5680362405434552e-27

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:5208: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  self[name] = value
```

important features - Creditscore, Age, Balance, Numofproducts, Geography, Gender, IsActiveMember

In [23]:
```
df.columns
```

Out[23]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [24]:
```
x = df[['CreditScore', 'Geography', 'Gender', 'Age','Balance',
       'NumOfProducts', 'IsActiveMember']]
y = df["Exited"]
```

Preprocessing

In [25]:
```
x.head()
```

Out[25]:

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

Encoding of Geography and Gender

In [26]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.compose import ColumnTransformer
```

In [27]:
```
# for Gender
le1 = LabelEncoder()
le1.fit(x.Gender)
print(le1.classes_)
```

```
['Female' 'Male']
```

In [28]:
```
# encoding of gender
x.Gender = le1.transform(x.Gender)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:5208: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  self[name] = value
```

In [29]:
```
from sklearn.preprocessing import StandardScaler
# Onehot encoding for geography
encoder = ColumnTransformer([("ohe",OneHotEncoder(),[1]),
                             ("sc",StandardScaler(),[0,3,4,5]) ],remainder='passthrough')
encoder.fit(x)
x2 = encoder.transform(x)
x2.shape
```

Out[29]:
```
(10000, 9)
```

In [30]:
```
x2 = pd.DataFrame(x2)
x2.head()
```

Out[30]:

	0	1	2	3	4	5	6	7	8
0	1.0	0.0	0.0	-0.326221	0.293517	-1.225848	-0.911583	0.0	1.0
1	0.0	0.0	1.0	-0.440036	0.198164	0.117350	-0.911583	0.0	1.0
2	1.0	0.0	0.0	-1.536794	0.293517	1.333053	2.527057	0.0	0.0
3	1.0	0.0	0.0	0.501521	0.007457	-1.225848	0.807737	0.0	0.0
4	0.0	0.0	1.0	2.063884	0.388871	0.785728	-0.911583	0.0	1.0

In [31]:
```
x.head(6)
```

Out[31]:

	CreditScore	Geography	Gender	Age	Balance	NumOfProducts	IsActiveMember
0	619	France	0	42	0.00	1	1
1	608	Spain	0	41	83807.86	1	1
2	502	France	0	42	159660.80	3	0
3	699	France	0	39	0.00	2	0
4	850	Spain	0	43	125510.82	1	1
5	645	Spain	1	44	113755.78	2	0

In [32]:
```
df.head(6)
```

Out[32]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
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
5	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1

splitting data into train and test set

In [33]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x2,y,test_size=0.2,random_state=5)
print(x2.shape)
print(xtrain.shape)
print(xtest.shape)
print(y.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(10000, 9)
(8000, 9)
(2000, 9)
(10000,)
(8000,)
(2000,)
```

Apply Machine Learning - Logistic Regression ¶

In [34]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
```

In [35]:
```
#train the model using training data
model.fit(xtrain,ytrain)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[35]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

Performance Analysis

In [36]:
```
new_customer = [[650,"Germany",1,51,456125,3,1]]
new_customer = encoder.transform(new_customer)
```

In [37]:
```
model.predict(new_customer)
```

Out[37]:
```
array([0], dtype=int64)
```

In [38]:
```
# we will assess the performance of model using test data - xtest and ytest
ypred = model.predict(xtest)
from sklearn import metrics
metrics.accuracy_score(ytest,ypred)
```

Out[38]:
```
0.8095
```

In [39]:
```
metrics.recall_score(ytest,ypred)
```

Out[39]:
```
0.19012345679012346
```

In [40]:
```
metrics.confusion_matrix(ytest,ypred)
```

Out[40]:
```
array([[1542,   53],
       [ 328,   77]], dtype=int64)
```

In [41]:
```
metrics.f1_score(ytest,ypred)
```

Out[41]:
```
0.28785046728971964
```

Decision Tree

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
model2 = DecisionTreeClassifier(criterion="gini")
```

In [43]:
```
# train the model using train data - xtrain, ytrain
model2.fit(xtrain,ytrain)
```

Out[43]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [44]:
```
ypred2 = model2.predict(xtest)
metrics.accuracy_score(ytest,ypred2)
```

Out[44]:
```
0.784
```

In [45]:
```
metrics.recall_score(ytest,ypred2)
```

Out[45]:
```
0.5135802469135803
```

visualizing the tree

In [46]:
```
from sklearn.preprocessing import StandardScaler
# Onehot encoding for geography
encoder = ColumnTransformer([("ohe",OneHotEncoder(),[1]) ],remainder='passthrough')
encoder.fit(x)
x2 = encoder.transform(x)
x2.shape
```

Out[46]:
```
(10000, 9)
```

In [47]:
```
pd.DataFrame(x2).head()
```

Out[47]:

	0	1	2	3	4	5	6	7	8
0	1.0	0.0	0.0	619.0	0.0	42.0	0.00	1.0	1.0
1	0.0	0.0	1.0	608.0	0.0	41.0	83807.86	1.0	1.0
2	1.0	0.0	0.0	502.0	0.0	42.0	159660.80	3.0	0.0
3	1.0	0.0	0.0	699.0	0.0	39.0	0.00	2.0	0.0
4	0.0	0.0	1.0	850.0	0.0	43.0	125510.82	1.0	1.0

In [48]:
```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x2,y,test_size=0.2,random_state=5)
print(x2.shape)
print(xtrain.shape)
print(xtest.shape)
print(y.shape)
print(ytrain.shape)
print(ytest.shape)
```

```
(10000, 9)
(8000, 9)
(2000, 9)
(10000,)
(8000,)
(2000,)
```

In [49]:
```
df.columns
```

Out[49]:
```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [50]:
```
from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(criterion="entropy",max_depth=8,max_leaf_nodes=30)
# train the model using train data - xtrain, ytrain
model3.fit(xtrain,ytrain)
```

Out[50]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=8,
                       max_features=None, max_leaf_nodes=30,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [51]:
```
# pip install graphviz
import graphviz
from sklearn import tree
features_names = ["France","Germany","Spain",'CreditScore','Gender','Age','Balance',
       'NumOfProducts','IsActiveMember']
class_names = ['Not Exited','Exited']
graph_data = tree.export_graphviz(model3,feature_names=features_names,class_names=class_names,
                                  rounded=True, filled=True)
graph = graphviz.Source(graph_data)
graph
```

Out[51]:
```
<graphviz.files.Source at 0x263829e3ec8>
```

In [52]:
```
# feature importance - can be used for feature selection
model3.feature_importances_
```

Out[52]:
```
array([0.        , 0.04896237, 0.        , 0.        , 0.        ,
       0.39736761, 0.12855251, 0.33453138, 0.09058612])
```

In [53]:
```
for i in range(len(features_names)): print(features_names[i],model3.feature_importances_[i])
```

```
France 0.0
Germany 0.04896237277119826
Spain 0.0
CreditScore 0.0
Gender 0.0
Age 0.3973676136731552
Balance 0.12855250832819834
NumOfProducts 0.3345313828739313
IsActiveMember 0.09058612235351693
```

Overfitting

- model has very good performance on train data
- model has not so good performance on test data

Underfitting

- model has not so good performance on test data
- model has similarly not so good performance on train data

In [54]:
```
model2 = DecisionTreeClassifier()
model2.fit(xtrain,ytrain)
```

Out[54]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [55]:
```
#recall on train data
metrics.recall_score(ytrain,model2.predict(xtrain))
```

Out[55]:
```
0.9987745098039216
```

In [56]:
```
# recall on test data
metrics.recall_score(ytest,model2.predict(xtest))
```

Out[56]:
```
0.5012345679012346
```

In [57]:
```
# pip install graphviz
import graphviz
from sklearn import tree
features_names = ["France","Germany","Spain",'CreditScore','Gender','Age','Balance',
       'NumOfProducts','IsActiveMember']
class_names = ['Not Exited','Exited']
graph_data = tree.export_graphviz(model2,feature_names=features_names,class_names=class_names,
                                  rounded=True, filled=True)
graph = graphviz.Source(graph_data)
graph
```

Out[57]:
```
<graphviz.files.Source at 0x26383a90148>
```

In [ ]:
```

```

Handle overfitting

In [58]:
```
model2 = DecisionTreeClassifier(max_depth=9,min_samples_leaf=12,min_samples_split=20,random_state=5)
model2.fit(xtrain,ytrain)
```

Out[58]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=9,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=12, min_samples_split=20,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=5, splitter='best')
```

In [59]:
```
#recall on train data
metrics.recall_score(ytrain,model2.predict(xtrain))
```

Out[59]:
```
0.5575980392156863
```

In [60]:
```
# recall on test data
metrics.recall_score(ytest,model2.predict(xtest))
```

Out[60]:
```
0.47160493827160493
```

Hyperparameter tuning using gridsearch

In [61]:
```
grid = {"max_depth":np.arange(5,31,2),
        "min_samples_leaf":np.arange(10,51,5),
        "min_samples_split":np.arange(10,101,10)}
```

In [62]:
```
np.arange(5,31,2)
```

Out[62]:
```
array([ 5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29])
```

In [63]:
```
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(DecisionTreeClassifier(),param_grid=grid,verbose=True,
                           scoring="recall",cv=5,n_jobs=-1)
```

In [64]:
```
grid_search.fit(x2,y)
```

```
Fitting 5 folds for each of 1170 candidates, totalling 5850 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    6.6s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:   10.7s
[Parallel(n_jobs=-1)]: Done 2292 tasks      | elapsed:   19.8s
[Parallel(n_jobs=-1)]: Done 4392 tasks      | elapsed:   33.0s
[Parallel(n_jobs=-1)]: Done 5850 out of 5850 | elapsed:   42.1s finished
```

Out[64]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort=False, random_state=None,
```

In [65]:
```
grid_search.best_params_
```

Out[65]:
```
{'max_depth': 21, 'min_samples_leaf': 10, 'min_samples_split': 30}
```

In [66]:
```
grid_search.best_score_
```

Out[66]:
```
0.5017148594450066
```

Random Forest

In [67]:
```
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier(n_estimators=50,max_depth=20,min_samples_leaf=15,min_samples_split=40,
                                oob_score=True)
model4.fit(xtrain,ytrain)
```

Out[67]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=20, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=15, min_samples_split=40,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=True, random_state=None,
                       verbose=0, warm_start=False)
```

In [68]:
```
#recall on train data
metrics.recall_score(ytrain,model4.predict(xtrain))
```

Out[68]:
```
0.4957107843137255
```

In [69]:
```
# recall on test data
metrics.recall_score(ytest,model4.predict(xtest))
```

Out[69]:
```
0.4444444444444444
```

In [70]:
```
model4.oob_score_
```

Out[70]:
```
0.86675
```

In [71]:
```
model4.feature_importances_
```

Out[71]:
```
array([0.01142849, 0.0495987 , 0.00823252, 0.06034521, 0.0173924 ,
       0.38719286, 0.10683264, 0.27490786, 0.08406932])
```

In [72]:
```
features_names
```

Out[72]:
```
['France',
 'Germany',
 'Spain',
 'CreditScore',
 'Gender',
 'Age',
 'Balance',
 'NumOfProducts',
 'IsActiveMember']
```

Adaboost

In [73]:
```
from sklearn.ensemble import AdaBoostClassifier
model5 = AdaBoostClassifier(n_estimators=50,random_state=5)
model5.fit(xtrain,ytrain)
```

Out[73]:
```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=5)
```

In [74]:
```
#recall on train data
metrics.recall_score(ytrain,model5.predict(xtrain))
```

Out[74]:
```
0.48590686274509803
```

In [75]:
```
# recall on test data
metrics.recall_score(ytest,model5.predict(xtest))
```

Out[75]:
```
0.4444444444444444
```

Gradient Boosting

In [76]:
```
from sklearn.ensemble import GradientBoostingClassifier
model6 = GradientBoostingClassifier(n_estimators=150,learning_rate=0.01)
```

In [77]:
```
model6.fit(xtrain,ytrain)
```

Out[77]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
                           learning_rate=0.01, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=150,
                           n_iter_no_change=None, presort='auto',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [78]:
```
# recall on test data
metrics.recall_score(ytest,model6.predict(xtest))
```

Out[78]:
```
0.32839506172839505
```

In [79]:
```
#recall on train data
metrics.recall_score(ytrain,model6.predict(xtrain))
```

Out[79]:
```
0.3382352941176471
```

XGBoost

In [80]:
```
from xgboost import XGBClassifier
# import xgboost
# model7 = xgboost.XGBClassifier()
```

In [81]:
```
model7 = XGBClassifier(max_depth=20,learning_rate=0.01,n_estimators=50,random_state=5)
model7.fit(xtrain,ytrain)
```

Out[81]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.01, max_delta_step=0, max_depth=20,
              min_child_weight=1, missing=None, n_estimators=50, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=5,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [82]:
```
#recall on train data
metrics.recall_score(ytrain,model7.predict(xtrain))
```

Out[82]:
```
0.6924019607843137
```

In [83]:
```
# recall on test data
metrics.recall_score(ytest,model7.predict(xtest))
```

Out[83]:
```
0.4666666666666667
```

In [84]:
```
def get_recall(thresh,features,label,model):
    ypred = model.predict_proba(features)[:,1]
    ypred = np.where(ypred>thresh,1,0)
    return metrics.f1_score(label,ypred),metrics.recall_score(label,ypred)
```

In [85]:
```
get_recall(0.35,xtest,ytest,model7)
```

Out[85]:
```
(0.5469659185369908, 0.8123456790123457)
```

Stacking

In [86]:
```
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [87]:
```
rf = RandomForestClassifier(n_estimators=100,random_state=5)
gb = GradientBoostingClassifier(n_estimators=100,random_state=5)
lr = LogisticRegression()
dt = DecisionTreeClassifier(max_depth=20)
```

In [88]:
```
model = StackingClassifier(classifiers=[rf,gb,dt],meta_classifier=lr)
```

In [89]:
```
model.fit(xtrain,ytrain)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[89]:
```
StackingClassifier(average_probas=False,
                   classifiers=[RandomForestClassifier(bootstrap=True,
                                                       class_weight=None,
                                                       criterion='gini',
                                                       max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=0.0,
                                                       min_impurity_split=None,
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_leaf=0.0,
                                                       n_estimators=100,
                                                       n_jobs=None,
                                                       oob_score=False,
                                                       random...
                   meta_classifier=LogisticRegression(C=1.0, class_weight=None,
                                                      dual=False,
                                                      fit_intercept=True,
                                                      intercept_scaling=1,
                                                      l1_ratio=None,
                                                      max_iter=100,
                                                      multi_class='warn',
                                                      n_jobs=None, penalty='l2',
                                                      random_state=None,
                                                      solver='warn', tol=0.0001,
                                                      verbose=0,
                                                      warm_start=False),
                   store_train_meta_features=False, use_clones=True,
                   use_features_in_secondary=False, use_probas=False,
                   verbose=0)
```

In [90]:
```
from sklearn import metrics
# recall on test data
metrics.recall_score(ytest,model.predict(xtest))
```

Out[90]:
```
0.4765432098765432
```

In [91]:
```
#recall on train data
metrics.recall_score(ytrain,model.predict(xtrain))
```

Out[91]:
```
0.9987745098039216
```

In [92]:
```
import joblib
joblib.dump(model,"myapp/churn_prediction.pkl")
joblib.dump(le1,"myapp/gender_encoder.pkl")
joblib.dump(encoder,"myapp/encoder.pkl")
```

Out[92]:
```
['myapp/encoder.pkl']
```

In [98]:
```
le1.transform(['Male'])
```

Out[98]:
```
array([1], dtype=int64)
```

In [ ]: