```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib notebook
import seaborn as sns
import numpy as np
%matplotlib inline
# %load ../standard_import.txt
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_classification

from sklearn.preprocessing import scale
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans

from scipy.cluster import hierarchy

pd.set_option('display.notebook_repr_html', False)

%matplotlib inline
plt.style.use('seaborn-white')
```

```
<ipython-input-297-76b734dff1de>:24: MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are deprecated since 3.6, as they no longer correspond to the styles
  plt.style.use('seaborn-white')
```

```
#Loading data
df = pd.read_csv('/content/churn.csv')
df.head()
```

None

```
df.tail()
```

None

2.2.1 Number of Observations:

```
df.shape
```

```
(10000, 14)
```

2.2.2 Field names

```
df.columns
```

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

2.2.3 Type of fields

```
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
```

```
table = df.describe(include='all')
table
```

None

3.1 Data Processing/Cleaning

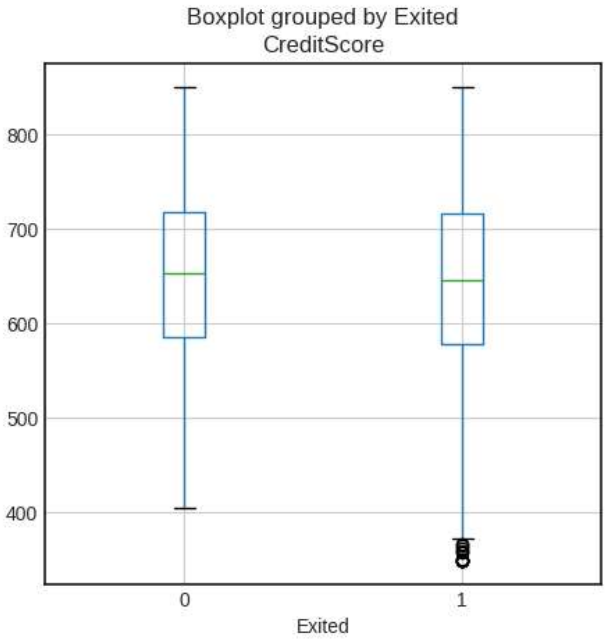
```
df.isna().sum()
```

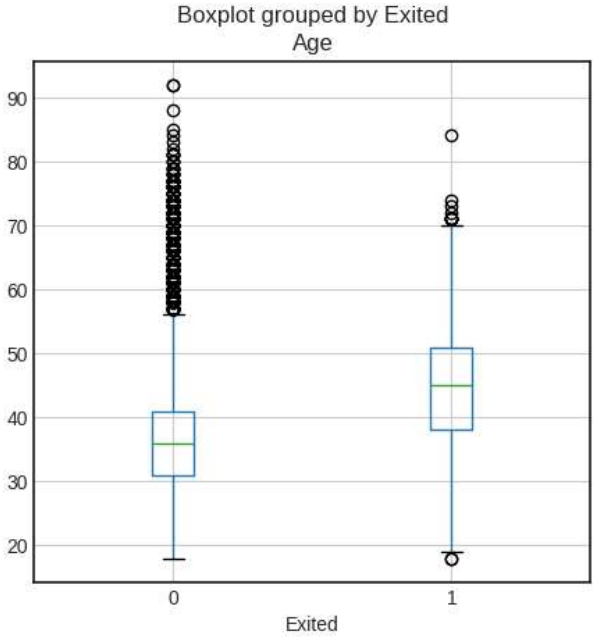
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
```

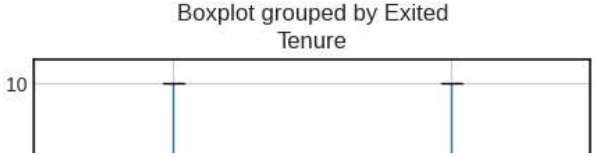
```
Gender            0
Age               0
Tenure            0
Balance           0
NumOfProducts     0
HasCrCard         0
IsActiveMember    0
EstimatedSalary   0
Exited            0
dtype: int64
```

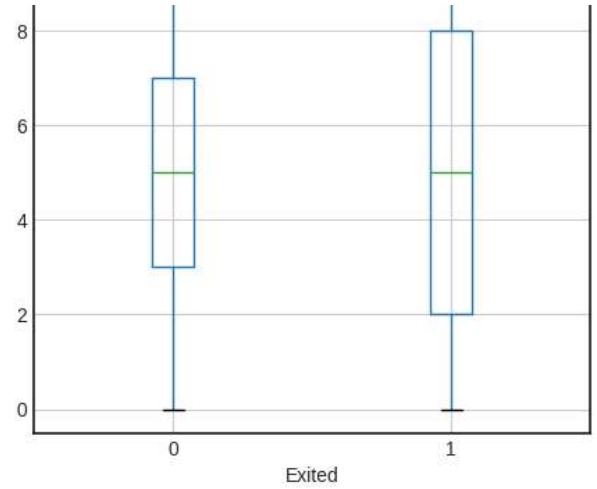
3.1.1 Outlier Detection

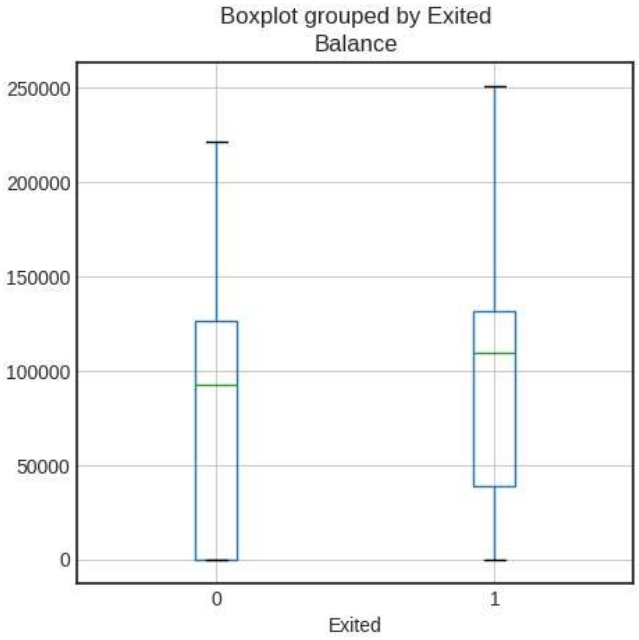
```
for i in df.columns.drop(['RowNumber', 'CustomerId', 'Surname','Geography','Gender','Exited']):
    boxplot = df.boxplot(column=i, by = 'Exited',figsize=(5,5))
```

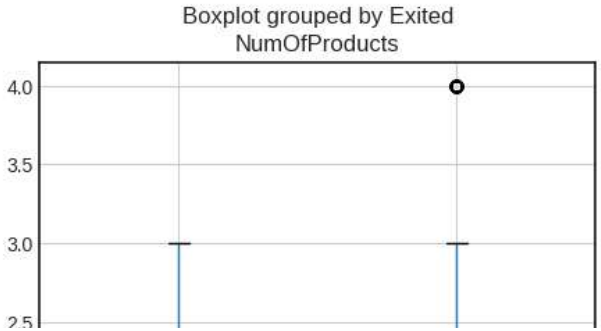

Boxplot grouped by Exited
CreditScore
800
700
600
500
400
0
1
Exited

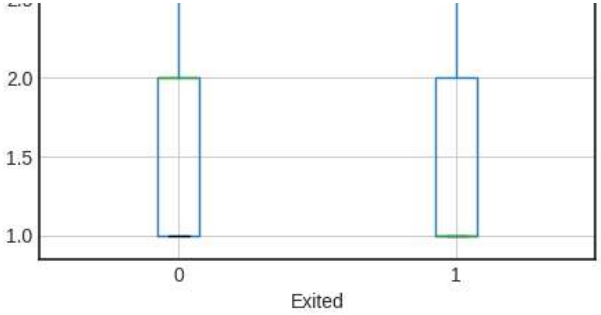

Boxplot grouped by Exited
Age
90
80
70
60
50
40
30
20
0
1
Exited

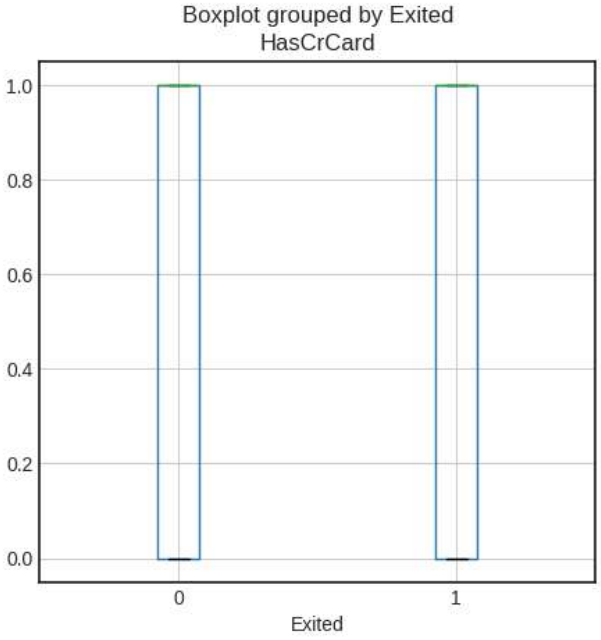

Boxplot grouped by Exited
Tenure
10

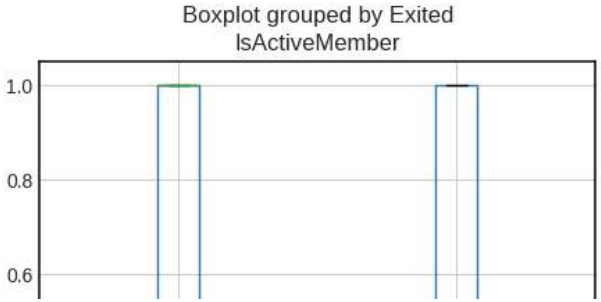

8
6
4
2
0
0
1
Exited


Boxplot grouped by Exited
Balance
250000
200000
150000
100000
50000
0
0
1
Exited


Boxplot grouped by Exited
NumOfProducts
4.0
3.5
3.0
2.5

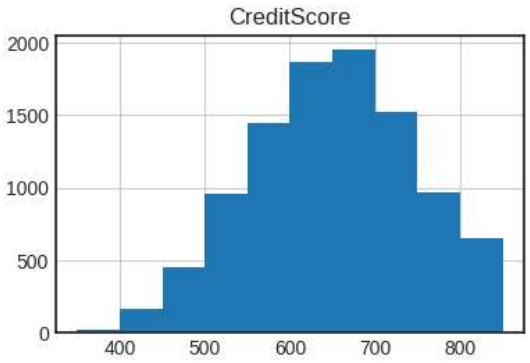


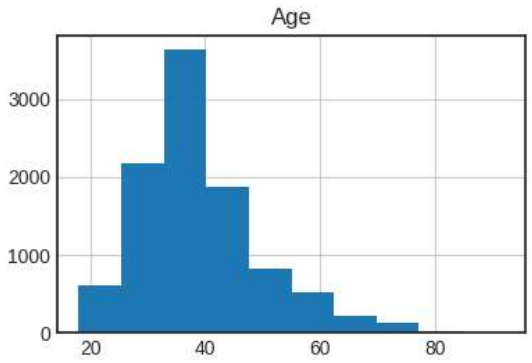


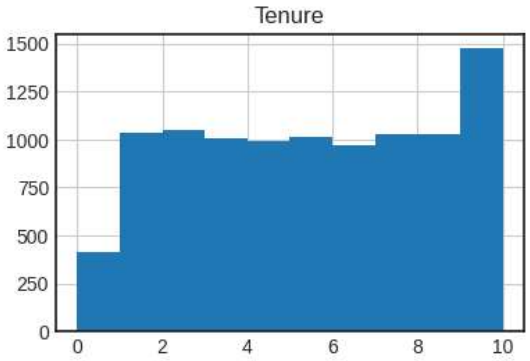


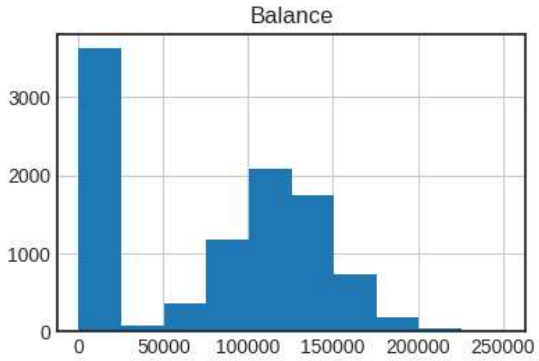
Histogram

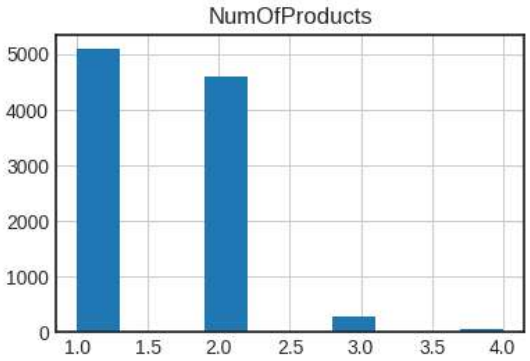
```
df.hist(column=['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary'], figsize=(10,10))
plt.show()
```

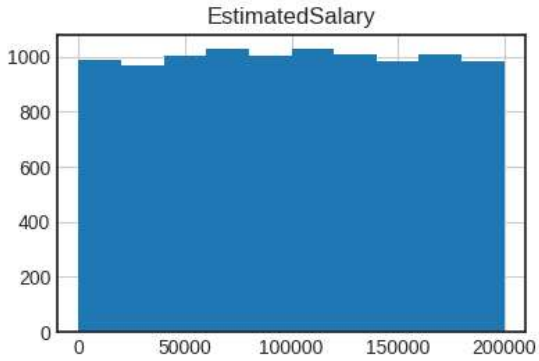






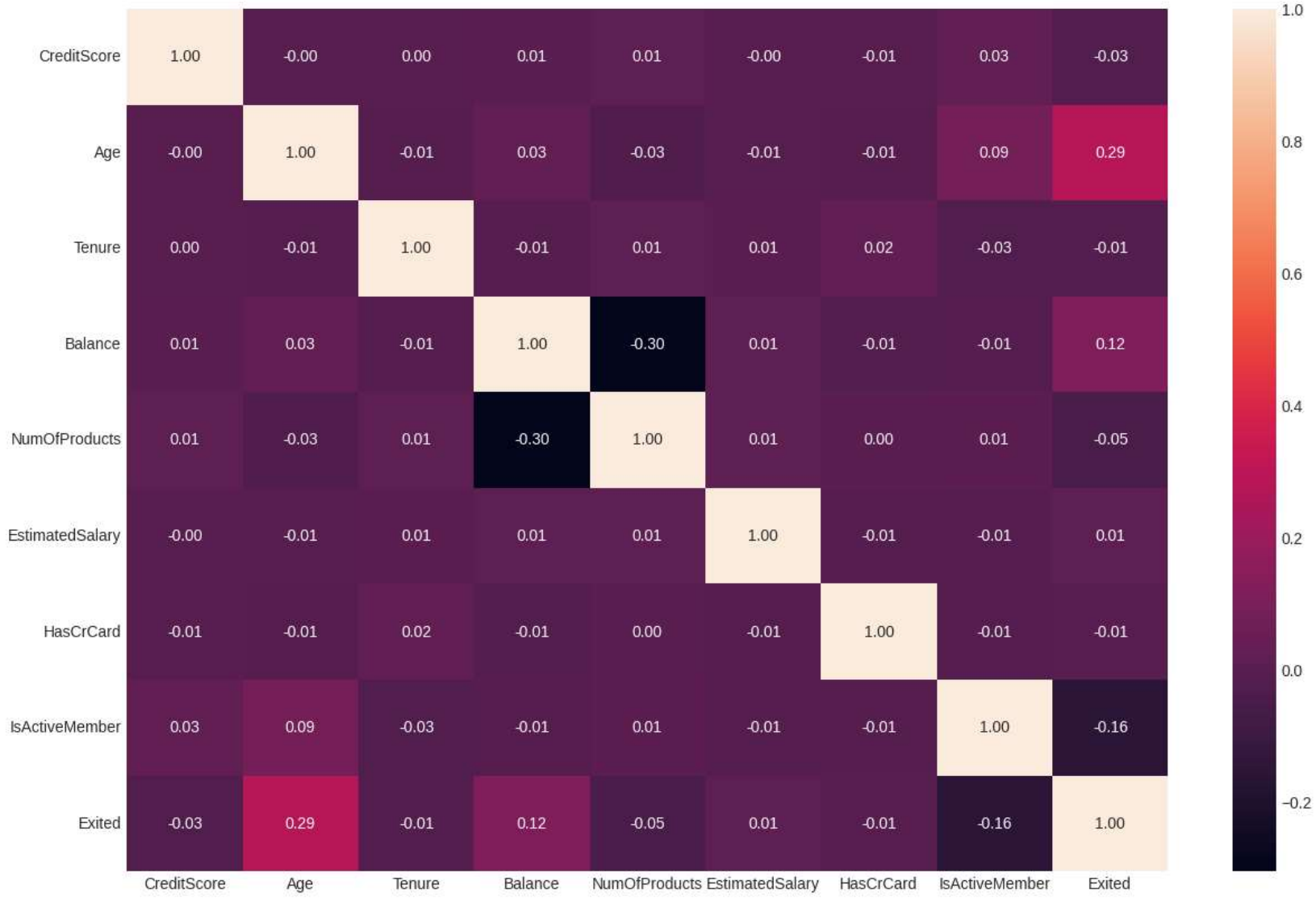




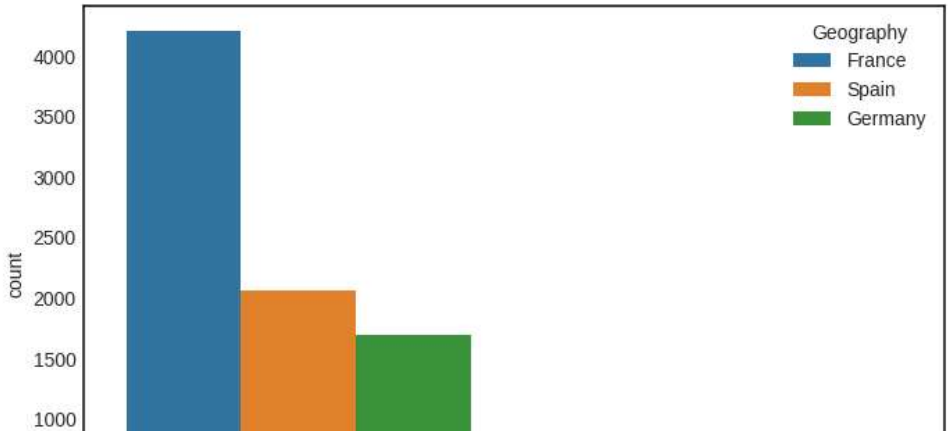


Correlation Matrix heatmap

```
list1 = ['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary','HasCrCard','IsActiveMember','Exited']
plt.figure(figsize=(15,10))
sns.heatmap(df[list1].corr(), annot=True, fmt=".2f")
plt.show()
```




```
plt.figure(figsize=(8, 5))
sns.countplot(x='Exited', hue='Geography', data=df)
plt.show()
```




```
plt.figure(figsize=(8, 5))
crosstab = pd.crosstab(df['Geography'], df['Gender'], values=df['Exited'], aggfunc=np.sum)
sns.heatmap(crosstab, annot=True, fmt='d')
plt.show()
```

Preprocessing & Feature Engineering

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

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
```

```
             'Exited'],
            dtype='object')
```

```
for i in range(len(df)):
    if df['EstimatedSalary'][i]<= 50000:
        df['EstimatedSalary'][i]= 'Poor'
    elif 50000 < df['EstimatedSalary'][i]<= 100000:
        df['EstimatedSalary'][i]= 'Good'
    elif 100000 < df['EstimatedSalary'][i]<= 200000:
        df['EstimatedSalary'][i]= 'Excellent'
    else:
        df['EstimatedSalary'][i]= "Top"
```

```
<ipython-input-311-f91b2aab9968>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['EstimatedSalary'][i]= 'Excellent'
```

```
for i in range(len(df)):
    if df['Balance'][i]<= 25000:
        df['Balance'][i]= 'Poor'
    elif 25000 < df['Balance'][i]<= 100000:
        df['Balance'][i]= 'Good'
    elif 100000 < df['Balance'][i]<= 200000:
        df['Balance'][i]= 'Excellent'
    else:
        df['Balance'][i]= "Top"
```

```
<ipython-input-312-71ee8dca9bc5>:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Balance'][i]= 'Poor'
```

```
df['Gender'],df['Geography'],df['Balance'],df['EstimatedSalary']=df['Gender'].astype('category'),df['Geography'].astype('category'),df['Balance'].astype('category'),df['EstimatedSal
df['Gender']=df.Gender.cat.codes
df['Geography']=df.Geography.cat.codes
df['Balance']=df.Balance.cat.codes
df['EstimatedSalary']=df.EstimatedSalary.cat.codes
```

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
df['CreditScore'] = scaler.fit_transform(df[['CreditScore']])
df['Age'] = scaler.fit_transform(df[['Age']])
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
```

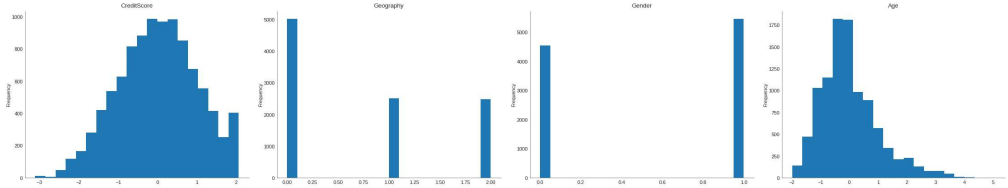
```
 0   CreditScore      10000 non-null  float64
 1   Geography        10000 non-null  int8
 2   Gender           10000 non-null  int8
 3   Age              10000 non-null  float64
 4   Tenure           10000 non-null  int64
 5   Balance          10000 non-null  int8
 6   NumOfProducts    10000 non-null  int64
 7   HasCrCard        10000 non-null  int64
 8   IsActiveMember   10000 non-null  int64
 9   EstimatedSalary  10000 non-null  int8
 10  Exited           10000 non-null  int64
dtypes: float64(2), int64(5), int8(4)
memory usage: 586.1 KB
```

```
df
```

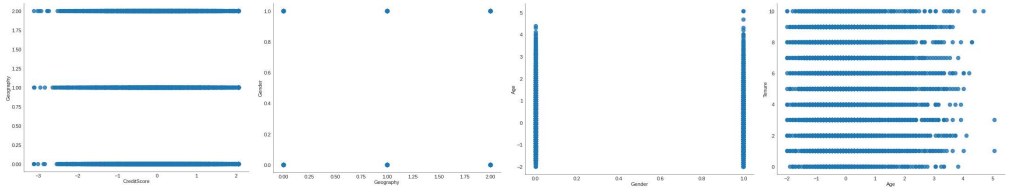
None

Like what you see? Visit the data table notebook to learn more about interactive tables.

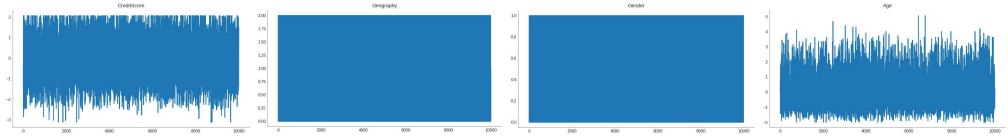
Distributions



2-d distributions



Values




```
Error: Runtime no longer has a reference to this dataframe, please re-run this cell and try again.
Error: Runtime no longer has a reference to this dataframe, please re-run this cell and try again.
Error: Runtime no longer has a reference to this dataframe, please re-run this cell and try again.
```

```
predictors = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
dependent_variable_name = ['Exited']
```

```
X = df[['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']]
X = scale(X)
```

```
X = pd.DataFrame(X)
X.head()
```

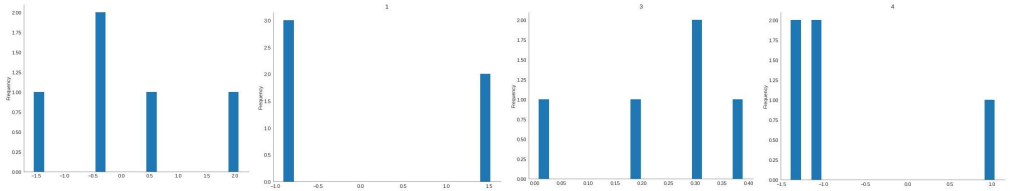
1 to 5 of 5 entries Filter

index	0	1	2	3	4	5	
0	-0.32622142203674587	-0.9018862432746051	-1.0959875190286648	0.29351742289674687	-1.0417596792253108	1.2076506494805654	-0.
1	-0.4400359548576653	1.5150673766493	-1.0959875190286648	0.19816383219544517	-1.3875375865624426	0.11712270160214265	-0.
2	-1.5367941802228888	-0.9018862432746051	-1.0959875190286648	0.29351742289674687	1.03290776479748	-0.97340524627628	
3	0.5015206348426682	-0.9018862432746051	-1.0959875190286648	0.007456650792841798	-1.3875375865624426	1.2076506494805654	0.
4	2.063883767202562	1.5150673766493	-1.0959875190286648	0.38887101359804854	-1.0417596792253108	-0.97340524627628	-0.

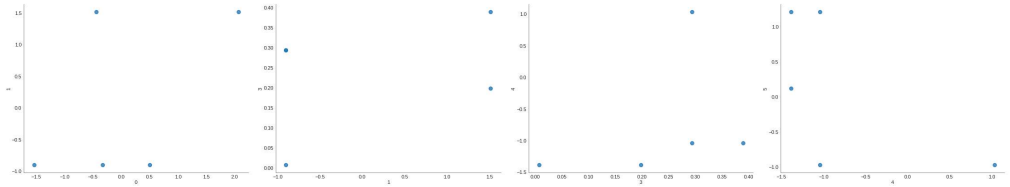
Show 25 per page

Like what you see? Visit the data table notebook to learn more about interactive tables.

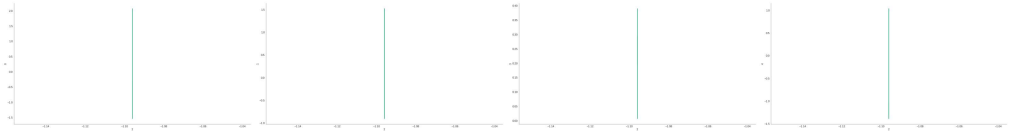
Distributions



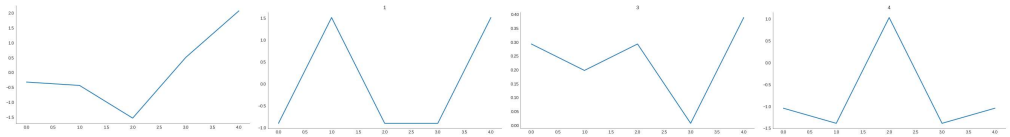
2-d distributions



Time series



Values



```
Y = df[['Exited']]
Y
```

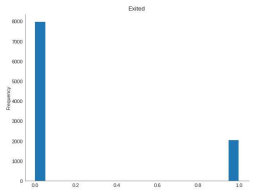
1 to 25 of 10000 entries Filter

index	Exited
0	1
1	0
2	1
3	0
4	0
5	1
6	0
7	1
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	1
17	0
18	0
19	0
20	0
21	0
22	1
23	0
24	0

Show 25 per page 1 2 10 100 300 390 400

Like what you see? Visit the data table notebook to learn more about interactive tables.

Distributions



Values

```
df['Exited'].value_counts()
```

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

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

```
rus = RandomUnderSampler(random_state=42)
X_train_rus, y_train_rus= rus.fit_resample(X, Y)
```

```
y_train_rus.value_counts()
```

```
Exited
0         2037
1         2037
dtype: int64
```

Decision Tree

```
from sklearn.model_selection import train_test_split
np.random.seed(42)
```

```
X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size=0.2, random_state=100)

print('The number of records in the training dataset is', X_train.shape[0])
print('The number of records in the test dataset is', X_test.shape[0])
```

```
The number of records in the training dataset is 8000
The number of records in the test dataset is 2000
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from collections import Counter
```

```
clf = DecisionTreeClassifier(criterion = 'entropy', random_state = 10, max_depth =8)
clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=8, random_state=10)
```

```
y_predic = clf.predict(X_test)
y_predic
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
print("Accuracy of this model is:", accuracy_score(y_test, y_predic))
```

```
Accuracy of this model is: 0.8543333333333333
```

```
sco = accuracy_score(y_test, y_predic)*100
```

```
print("This model is", sco, "accurate, that is in acceptable " )
```

```
This model is 85.43333333333332 accurate, that is in acceptable
```

```
print(classification_report(y_test, y_predic))
```

```
              precision    recall  f1-score   support

           0       0.86      0.97      0.91      2416
           1       0.76      0.37      0.49       584

    accuracy                           0.85      3000
   macro avg       0.81      0.67      0.70      3000
weighted avg       0.84      0.85      0.83      3000
```

```
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz
import graphviz
import pydot
from IPython.display import Image, display
```

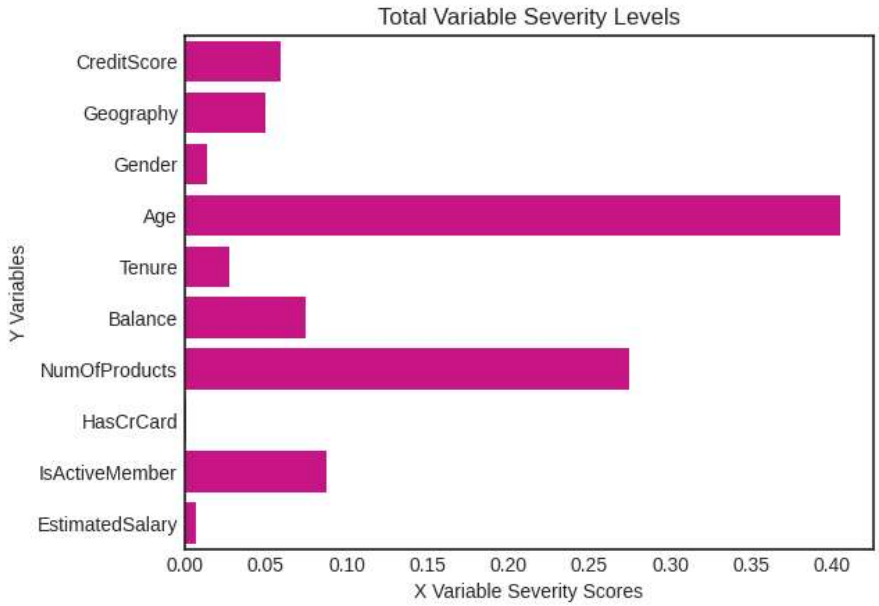
```
feature_importances = clf.feature_importances_
pd.DataFrame(data=feature_importances, index=predictors, columns = ['Importance']).sort_values(by=['Importance'], ascending=False)
```

1 to 10 of 10 entries Filter

index	Importance
Age	0.40521188612984704
NumOfProducts	0.27426100459028024
CreditScore	0.059541987744209685

```
feature_importances_list = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
```

```
sns.barplot(x = feature_importances, y = feature_importances_list, color='mediumvioletred', saturation=1)
plt.xlabel('X Variable Severity Scores')
plt.ylabel('Y Variables')
plt.title('Total Variable Severity Levels')
plt.show()
```



Random Forest

```
from sklearn.ensemble import RandomForestClassifier
RF=RandomForestClassifier(n_estimators=20)
RF.fit(X_train,y_train)
```

```
<ipython-input-371-55805171086b>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example
  RF.fit(X_train,y_train)
```

▾ RandomForestClassifier

RandomForestClassifier(n_estimators=20)

```
y_pred=RF.predict(X_test)
```

```
from sklearn import metrics
# Model Accuracy, how often is the classifier correct?
print("Accuracy of model:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy of model: 0.8596666666666667
```

```
from sklearn.metrics import RocCurveDisplay
```

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

```
confusion_matrix(y_test, y_pred)
```

```
array([[1535,   53],
       [ 235,  177]])
```

```
print(classification_report(y_test, y_pred))
```

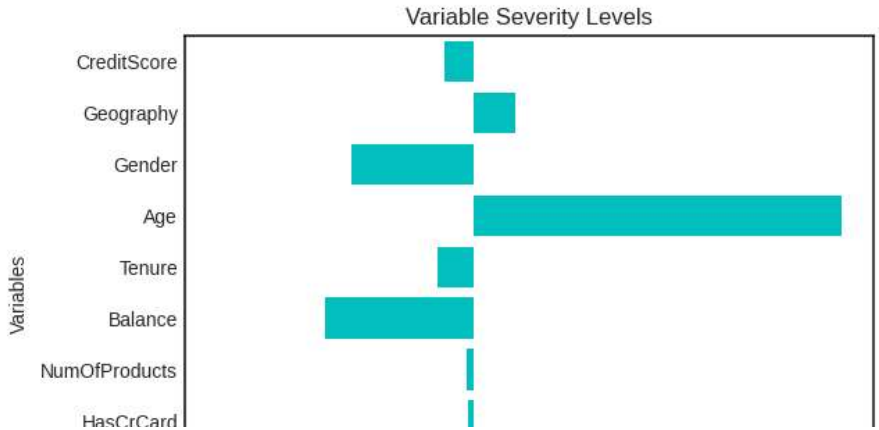
```
              precision    recall  f1-score   support

           0       0.87      0.97      0.91      1588
           1       0.77      0.43      0.55       412

    accuracy                           0.86      2000
   macro avg       0.82      0.70      0.73      2000
weighted avg       0.85      0.86      0.84      2000
```

```
feature_importances = RF.feature_importances_
pd.DataFrame(data=feature_importances, index=predictors, columns = ['Importance']).sort_values(by=['Importance'], ascending=False)
```

None

```
feature_importances_list = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
```

```
sns.barplot(x = feature_importances, y = feature_importances_list, color='c', saturation=1)
plt.xlabel('Variable Severity Scores')
plt.ylabel('Variables')
plt.title('Variable Severity Levels')
plt.show()
```



logistic Regression

```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression(solver='liblinear', max_iter=50)
log_model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the sh
  y = column_or_1d(y, warn=True)
```

LogisticRegression

```
LogisticRegression(max_iter=50, solver='liblinear')
```

```
y_log_pred = log_model.predict(X_test)
y_log_pred.shape[0]
```

```
3000
```

```
print(y_log_pred)
```

```
[0 0 0 ... 0 0 0]
```

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

```
f1_score(y_test,y_log_pred)
```

```
0.27792207792207796
```

```
accuracy_score(y_test,y_log_pred)
```

```
0.8146666666666667
```

```
precision_score(y_test,y_log_pred)
```

```
0.5752688172043011
```

```
recall_score(y_test,y_log_pred)
```

```
0.1832191780821918
```

```
print(classification_report(y_test,y_log_pred))
```

```
                   precision    recall  f1-score   support

               0       0.83      0.97      0.89      2416
               1       0.58      0.18      0.28       584

        accuracy                           0.81      3000
       macro avg       0.70      0.58      0.59      3000
    weighted avg       0.78      0.81      0.77      3000
```

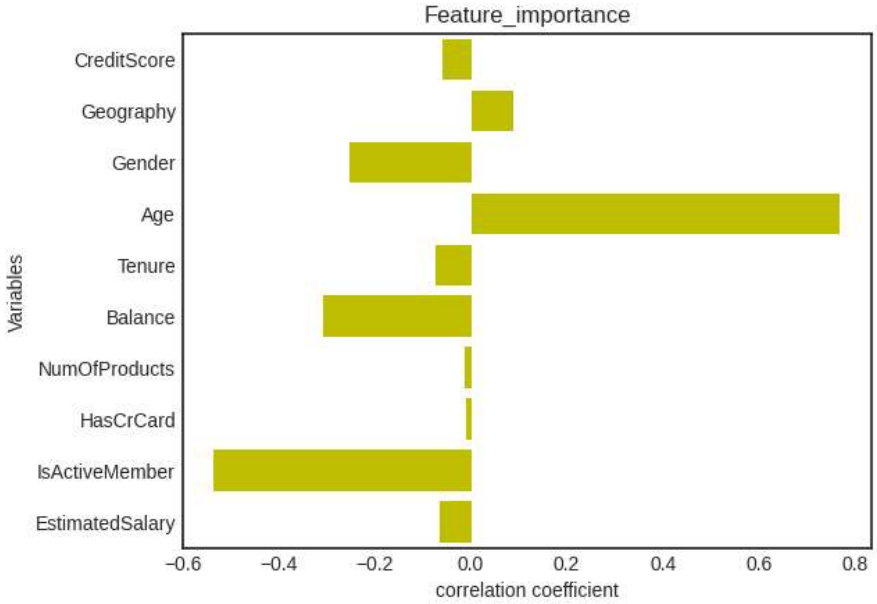
```
feature_importances = log_model.coef_[0]
pd.DataFrame(data=feature_importances, index=predictors,
             columns = ['Importance']).sort_values(by=['Importance'], ascending=False)
```

1 to 10 of 10 entries Filter

index	Importance

```
feature_importances_list = ['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
```

```
sns.barplot(x = feature_importances, y = feature_importances_list, color='y', saturation=10)
plt.xlabel('correlation coefficient')
plt.ylabel('Variables')
plt.title('Feature_importance')
plt.show()
```



KNN

```
from sklearn.neighbors import KNeighborsClassifier
KNN = KNeighborsClassifier(n_neighbors=8)
```

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

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please chang
  return self._fit(X, y)
```

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=8)

```
y_pred=KNN.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy of model:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy of model: 0.838
```

```
confusion_matrix(y_test, y_pred)
```

```
array([[2345,   71],
       [ 415,  169]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.97      0.91      2416
               1       0.70      0.29      0.41       584

        accuracy                           0.84      3000
       macro avg       0.78      0.63      0.66      3000
    weighted avg       0.82      0.84      0.81      3000
```

```
### Voting


from sklearn.ensemble import  VotingClassifier
Classifier = VotingClassifier(estimators=[
('DT', clf), ('RF', RF),('LogReg',log_model),('KNN',KNN)],voting='soft',flatten_transform=True)
eclf1 = Classifier.fit(X_train , y_train)
print(Classifier.score(X_test,y_test))
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:99: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
0.8623333333333333
```

```
y_pred=Classifier.predict(X_test)
```

Hard and soft voting score

```
# importing libraries
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split



# train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                         Y,
                         test_size = 0.30,
                         random_state = 4)

# group / ensemble of models
estimator = []
estimator.append(('LR',
        LogisticRegression(solver ='lbfgs',
                  multi_class ='multinomial',
                  max_iter = 200)))
estimator.append(('SVC', SVC(gamma ='auto', probability = True)))
estimator.append(('DTC', DecisionTreeClassifier()))

# Voting Classifier with hard voting
vot_hard = VotingClassifier(estimators = estimator, voting ='hard')
vot_hard.fit(X_train, y_train)
y_pred = vot_hard.predict(X_test)

# using accuracy_score metric to predict accuracy
score = accuracy_score(y_test, y_pred)
print("Hard Voting Score % d" % score)

# Voting Classifier with soft voting
vot_soft = VotingClassifier(estimators = estimator, voting ='soft')
vot_soft.fit(X_train, y_train)
y_pred = vot_soft.predict(X_test)

# using accuracy_score
score = accuracy_score(y_test, y_pred)
print("Soft Voting Score % d" % score)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:99: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
Hard Voting Score  0
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:99: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
Soft Voting Score  0
```

```
print(classification_report(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

                0       0.88      0.95      0.91      2416
                1       0.68      0.46      0.55       584

         accuracy                           0.85      3000
        macro avg       0.78      0.71      0.73      3000
     weighted avg       0.84      0.85      0.84      3000
```

```
np.random.seed(10)

#set up plotting area
plt.figure(0).clf()

#fit logistic regression model and plot ROC curve
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Logistic Regression, AUC="+str(auc))

#fit KNN model and plot ROC curve
model = KNeighborsClassifier(n_neighbors=15)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="KNN, AUC="+str(auc))

#fit Decision Tree model and plot ROC curve
model = DecisionTreeClassifier(criterion = 'entropy', random_state = 200, max_depth =8)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Decision Tree, AUC="+str(auc))

#fit Random Forest model and plot ROC curve
model = RandomForestClassifier(n_estimators=400)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Random Forest, AUC="+str(auc))

#fit Voting Classifier model and plot ROC curve
model = VotingClassifier(estimators=[
('DT', clf), ('RF', RF),('LogReg',log_model),('KNN',KNN)],voting='soft',flatten_transform=True)
model.fit(X_train, y_train)
y_pred = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred)
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
```

```
auc = round(metrics.roc_auc_score(y_test, y_pred), 4)
plt.plot(fpr,tpr,label="Voting Classifier, AUC="+str(auc))

#add legend
plt.legend()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the sh
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please chang
  return self._fit(X, y)
<ipython-input-406-cc7550c66c2b>:32: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example
  model.fit(X_train, y_train)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:99: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:134: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the
  y = column_or_1d(y, dtype=self.classes_.dtype, warn=True)
<matplotlib.legend.Legend at 0x7cf60da34f10>
```