```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive
drive.mount('/content/drive')
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from pandas.api import types
from sklearn.preprocessing import MinMaxScaler, normalize
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, KFold
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, ConfusionMatrixDisplay, accuracy_score
from sklearn.tree import DecisionTreeClassifier
import seaborn as sns
import sklearn.tree as tree
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn import svm
# Load employee data
Gdata = pd.read_csv('/content/drive/My Drive/data/general_data.csv')
employee_survey_data = pd.read_csv('/content/drive/My Drive/data/employee_survey_data.csv')
manager_survey_data = pd.read_csv('/content/drive/My Drive/data/manager_survey_data.csv')

display(Gdata.head() , employee_survey_data.head() , manager_survey_data.head())
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	Educat
0	51	No	Travel_Rarely	Sales	6	2	Lif
1	31	Yes	Travel_Frequently	Research & Development	10	1	Lif
2	32	No	Travel_Frequently	Research & Development	17	4	
3	38	No	Non-Travel	Research & Development	2	5	Lif
4	32	No	Travel_Rarely	Research & Development	10	1	

5 rows × 24 columns

	EmployeeID	EnvironmentSatisfaction	JobSatisfaction	WorkLifeBalance
0	1	3.0	4.0	2.0
1	2	3.0	2.0	4.0
2	3	2.0	2.0	1.0
3	4	4.0	4.0	3.0
4	5	4.0	1.0	3.0

	EmployeeID	JobInvolvement	PerformanceRating
0	1	3	3
1	2	2	4
2	3	3	3
3	4	2	3
4	5	3	3

```
# Merging all three data frames
data = pd.merge(Gdata, employee_survey_data, on='EmployeeID')
data = pd.merge(data, manager_survey_data, on='EmployeeID')
# Drop rows with missing values
data.dropna(inplace=True)
data.isnull().sum()
```

```
Age                        0
Attrition                  0
BusinessTravel             0
Department                 0
DistanceFromHome           0
Education                  0
EducationField             0
EmployeeCount              0
EmployeeID                 0
Gender                     0
JobLevel                   0
JobRole                    0
MaritalStatus              0
MonthlyIncome              0
NumCompaniesWorked         0
Over18                     0
PercentSalaryHike          0
StandardHours              0
StockOptionLevel           0
TotalWorkingYears          0
TrainingTimesLastYear      0
YearsAtCompany             0
YearsSinceLastPromotion    0
YearsWithCurrManager       0
EnvironmentSatisfaction    0
JobSatisfaction            0
WorkLifeBalance            0
JobInvolvement             0
PerformanceRating          0
dtype: int64
```

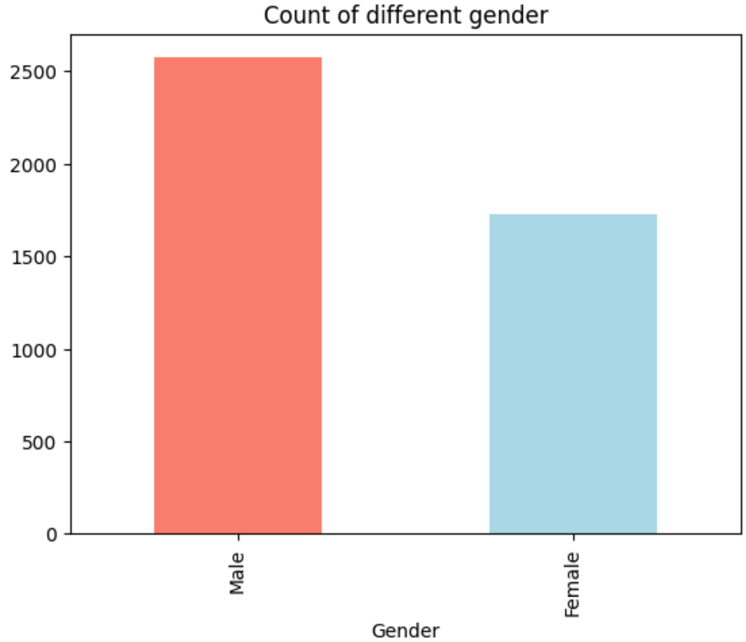
```
# Exploratory data analysis (EDA)
data.describe()
```

	Age	DistanceFromHome	Education	EmployeeCount	EmployeeID	JobLe
count	4300.000000	4300.000000	4300.000000	4300.0	4300.000000	4300.00(
mean	36.926977	9.197907	2.913256	1.0	2211.695116	2.06(
std	9.146517	8.097059	1.024774	0.0	1272.117692	1.10(
min	18.000000	1.000000	1.000000	1.0	1.000000	1.00(
25%	30.000000	2.000000	2.000000	1.0	1110.750000	1.00(
50%	36.000000	7.000000	3.000000	1.0	2215.500000	2.00(
75%	43.000000	14.000000	4.000000	1.0	3314.250000	3.00(
max	60.000000	29.000000	5.000000	1.0	4409.000000	5.00(

8 rows × 21 columns

```
#Understanding the balancing of the Gender column visually
data['Gender'].value_counts().plot(kind='bar',color=['salmon','lightblue'],title="Count of different gender")
```

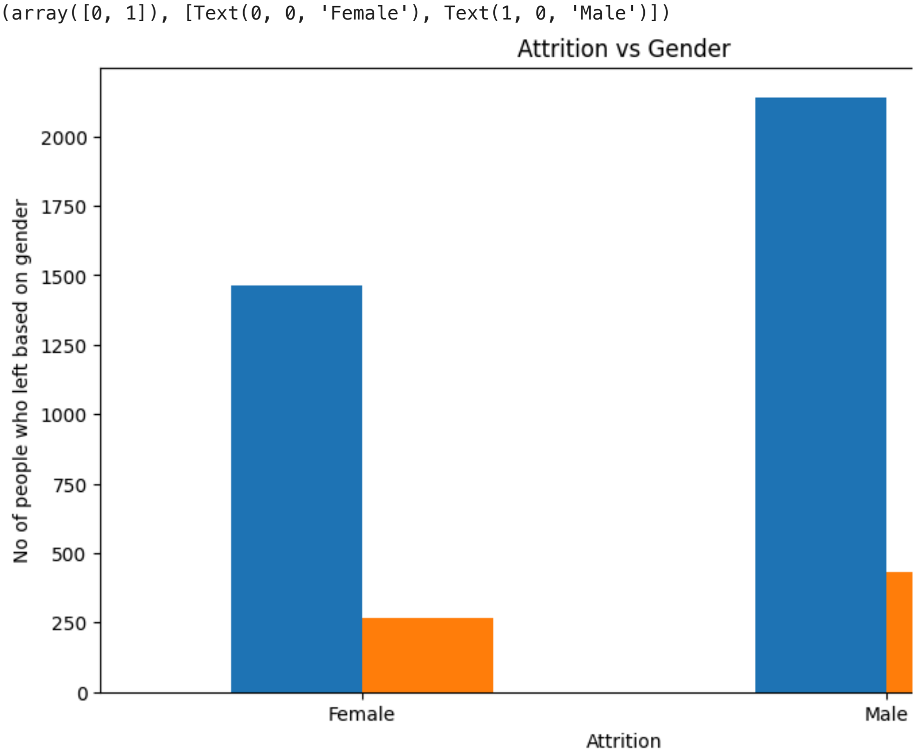
```
<Axes: title={'center': 'Count of different gender'}, xlabel='Gender'>
```



Now, let's figure out that how gender could be the reason for employees to leave the company or to stay in.

```
#Create a plot for crosstab

pd.crosstab(data['Gender'],data['Attrition']).plot(kind="bar",figsize=(10,6))
plt.title("Attrition vs Gender")
plt.xlabel("Attrition")
plt.ylabel("No of people who left based on gender")
plt.legend(["No","Yes"])
plt.xticks(rotation=0)
```

```
(array([0, 1]), [Text(0, 0, 'Female'), Text(1, 0, 'Male')])
```

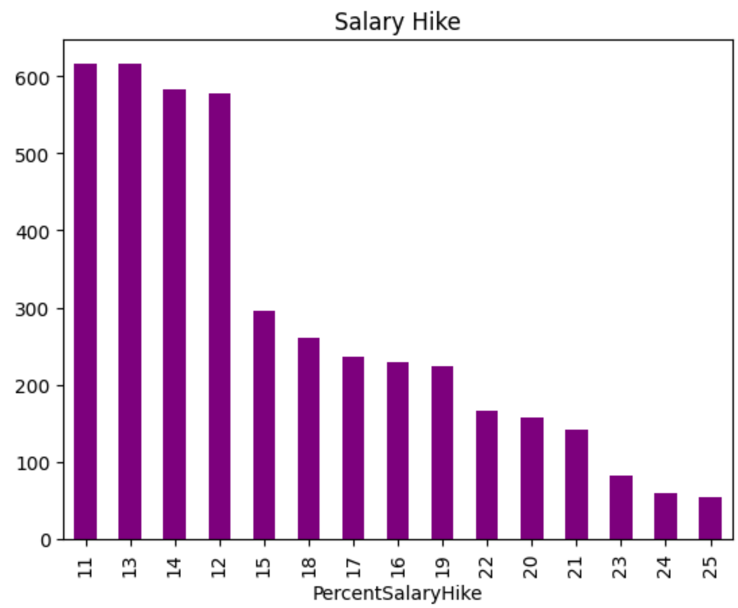


```
#PercentSalaryHike
promoted_dict = data["PercentSalaryHike"].value_counts()
promoted_dict
```

```
PercentSalaryHike
11    616
13    616
14    583
12    577
15    296
18    260
17    236
16    230
19    224
22    167
20    158
21    142
23     82
24     59
25     54
Name: count, dtype: int64
```

```
data["PercentSalaryHike"].value_counts().plot(kind='bar',color=['purple'],title= "Salary Hike")
```

```
<Axes: title={'center': 'Salary Hike'}, xlabel='PercentSalaryHike'>
```




```
# convert the non-numeric columns to numeric
for column in data.columns:
  if not types.is_numeric_dtype(data[column]):
    data[column] = pd.Categorical(data[column])
    data[column] = data[column].cat.codes
data.head()
```

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	Educat
0	51	0	2	2	6	2	
1	31	1	1	1	10	1	
2	32	0	1	1	17	4	
3	38	0	0	1	2	5	
4	32	0	2	1	10	1	

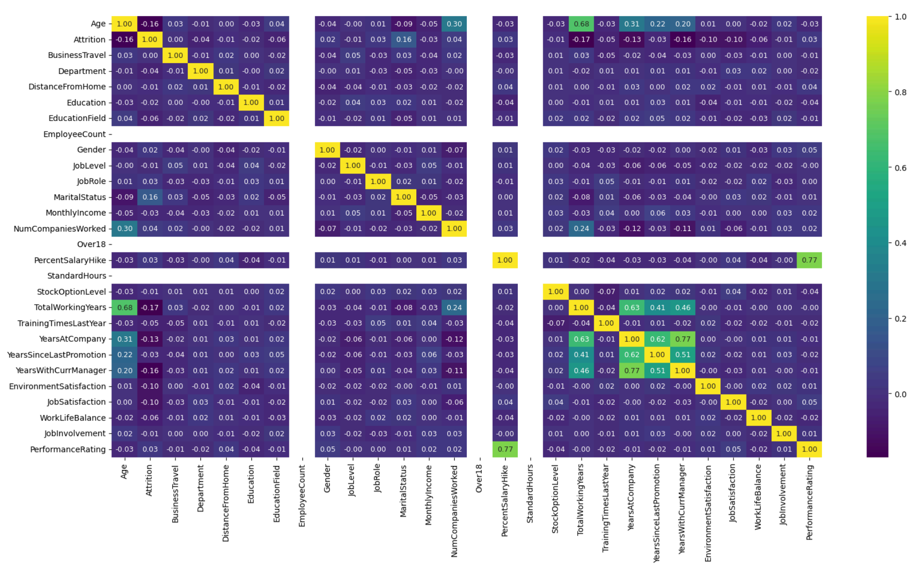
5 rows × 29 columns

```
# Create a Min/ Max Scaler object
matrix_data = data.copy()
scaler = MinMaxScaler()
# Select the columns to normalize except 'Attrition'
columns_to_normalize = data.columns[data.columns != 'Attrition']
# Normalize the selected columns
matrix_data[columns_to_normalize] = scaler.fit_transform(data[columns_to_normalize])
matrix_data.head()
```

	Age	Attrition	BusinessTravel	Department	DistanceFromHome	Education	Ed
0	0.785714	0	1.0	1.0	0.178571	0.25	
1	0.309524	1	0.5	0.5	0.321429	0.00	
2	0.333333	0	0.5	0.5	0.571429	0.75	
3	0.476190	0	0.0	0.5	0.035714	1.00	
4	0.333333	0	1.0	0.5	0.321429	0.00	

5 rows × 29 columns

```
corr_matrix = matrix_data.drop(columns=['EmployeeID']).corr()
plt.figure(figsize=(20, 10))

# Plot correlation matrix
sns.heatmap(corr_matrix, annot=True, cmap='viridis', fmt='.2f', annot_kws={'size': 9})

plt.subplots_adjust(hspace=0.5)
plt.show()
```



```
# Feature Selection and Engineering

drop_columns = ['EmployeeID', 'EmployeeCount', 'StandardHours', 'Age', 'Over18', 'YearsAtCompany', 'YearsWithCurrManager']
trainin_data = data.drop(columns=drop_columns)
trainin_data.head()
```

	Attrition	BusinessTravel	Department	DistanceFromHome	Education	EducationF:
0	0	2	2	6	2	
1	1	1	1	10	1	
2	0	1	1	17	4	
3	0	0	1	2	5	
4	0	2	1	10	1	

5 rows × 22 columns

```
y = trainin_data["Attrition"]
X = trainin_data.drop("Attrition",axis=1)
#Splitting data — Train test split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=3)
X_train.head()
```

	BusinessTravel	Department	DistanceFromHome	Education	EducationField	Ger
1333	0	2	7	3	1	
2881	2	0	25	1	0	
3540	2	1	6	4	1	
206	2	1	8	1	1	
1052	1	1	10	4	5	

5 rows × 21 columns

```
# Initialize the models:
lr=LogisticRegression(C = 0.1, random_state = 42, solver = 'liblinear')
dt=DecisionTreeClassifier()
rm=RandomForestClassifier()
gnb=GaussianNB()
knn = KNeighborsClassifier(n_neighbors=3)
svm = svm.SVC(kernel='linear')
```

```
#from one block of code, we will check the accuracy of all the model
for a,b in zip([lr,dt,knn,svm,rm,gnb],["Logistic Regression","Decision Tree","KNN","SVM","Random Forest","Naive Bayes"]):
    a.fit(X_train,y_train)
    prediction=a.predict(X_train)
    y_pred=a.predict(X_test)
    score1=accuracy_score(y_train,prediction)
    score=accuracy_score(y_test,y_pred)
    msg1="[%s] training data accuracy is : %f" % (b,score1)
    msg2="[%s] test data accuracy is : %f" % (b,score)
    print(msg1)
    print(msg2)
```

```
[Logistic Regression] training data accuracy is : 0.838663
[Logistic Regression] test data accuracy is : 0.837209
[Decision Tree] training data accuracy is : 1.000000
[Decision Tree] test data accuracy is : 0.984884
[KNN] training data accuracy is : 0.989535
[KNN] test data accuracy is : 0.906977
[SVM] training data accuracy is : 0.833140
[SVM] test data accuracy is : 0.834884
[Random Forest] training data accuracy is : 1.000000
[Random Forest] test data accuracy is : 0.988372
[Naive Bayes] training data accuracy is : 0.838663
[Naive Bayes] test data accuracy is : 0.837209
```

```
#model score
model_scores={'Logistic Regression':lr.score(X_test,y_test),
             'KNN classifier':knn.score(X_test,y_test),
             'Support Vector Machine':svm.score(X_test,y_test),
             'Random forest':rm.score(X_test,y_test),
              'Decision tree':dt.score(X_test,y_test),
              'Naive Bayes':gnb.score(X_test,y_test)
             }
model_scores
```

```
{'Logistic Regression': 0.8372093023255814,
 'KNN classifier': 0.9069767441860465,
 'Support Vector Machine': 0.8348837209302326,
 'Random forest': 0.9883720930232558,
 'Decision tree': 0.9848837209302326,
 'Naive Bayes': 0.8372093023255814}
```

```
#Classification Report of Random forest
from sklearn.metrics import classification_report

rm_y_preds = rm.predict(X_test)

print(classification_report(y_test,rm_y_preds))
```

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       715
           1       1.00      0.92      0.96       145

    accuracy                           0.99       860
   macro avg       0.99      0.96      0.97       860
weighted avg       0.99      0.99      0.99       860
```

```
#Classification Report of Logistic Regression
from sklearn.metrics import classification_report

lr_y_preds = lr.predict(X_test)

print(classification_report(y_test,lr_y_preds))
```

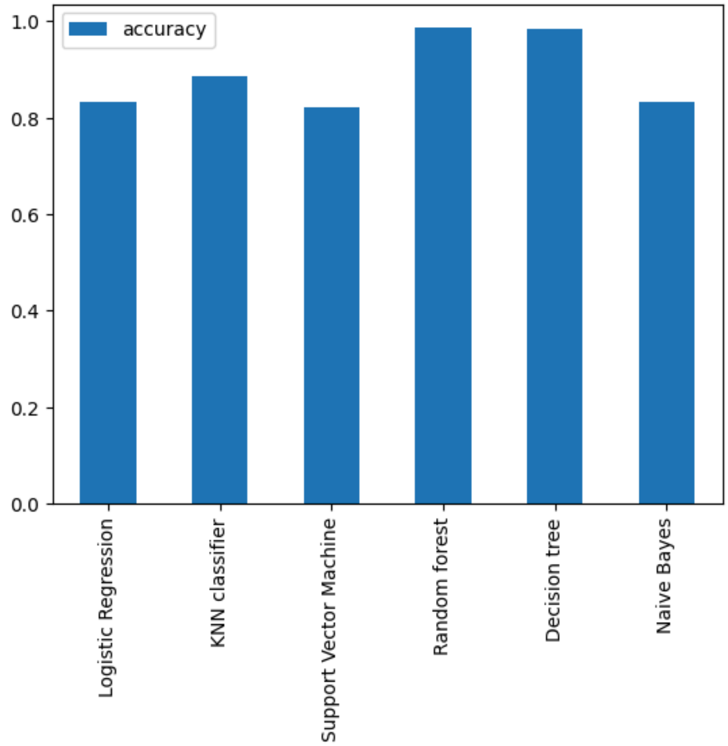
```
              precision    recall  f1-score   support

           0       0.83      1.00      0.91       715
           1       0.00      0.00      0.00       145

    accuracy                           0.83       860
   macro avg       0.42      0.50      0.45       860
weighted avg       0.69      0.83      0.75       860

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-sc
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-sc
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-sc
  _warn_prf(average, modifier, msg_start, len(result))
```

```
#Model Comparison Based on the accuracy

model_compare=pd.DataFrame(model_scores,index=['accuracy'])
model_compare
```

	Logistic Regression	KNN classifier	Support Vector Machine	Random forest	Decision tree	Naive Bayes

```
#Visualize the accuracy of each model
model_compare.T.plot(kind='bar') # (T is here for transpose)
```

```
<Axes: >
```



We can see that Random Forest has the best accuracy

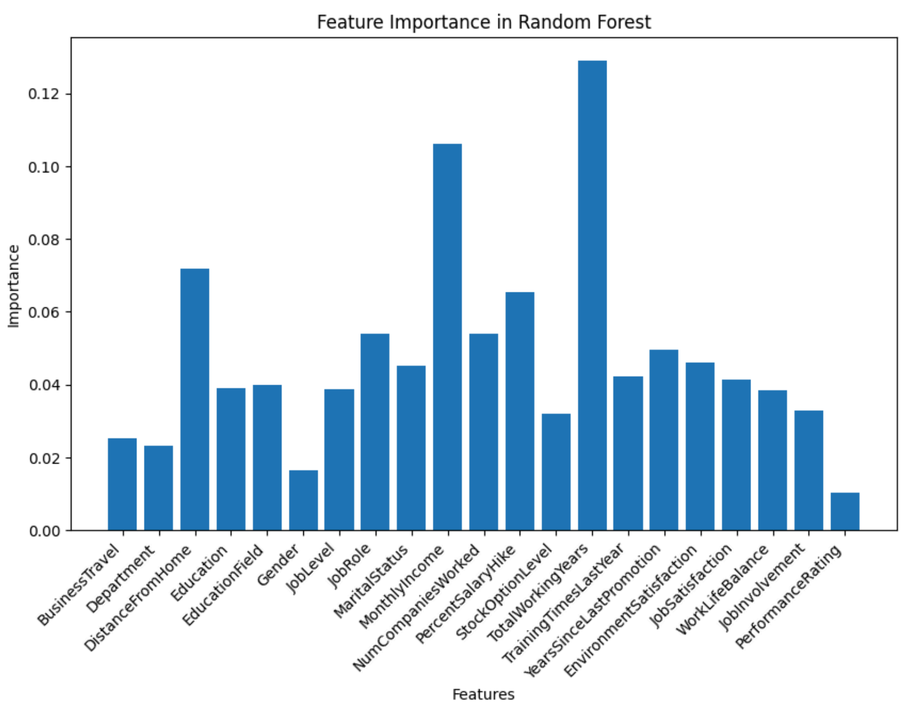
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4300 entries, 0 to 4408
Data columns (total 29 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Age                      4300 non-null   int64  
 1   Attrition                4300 non-null   int8   
 2   BusinessTravel           4300 non-null   int8   
 3   Department               4300 non-null   int8   
 4   DistanceFromHome         4300 non-null   int64  
 5   Education                4300 non-null   int64  
 6   EducationField           4300 non-null   int8   
 7   EmployeeCount            4300 non-null   int64  
 8   EmployeeID               4300 non-null   int64  
 9   Gender                   4300 non-null   int8   
 10  JobLevel                 4300 non-null   int64  
 11  JobRole                  4300 non-null   int8   
 12  MaritalStatus            4300 non-null   int8   
 13  MonthlyIncome            4300 non-null   int64  
 14  NumCompaniesWorked       4300 non-null   float64
 15  Over18                   4300 non-null   int8   
 16  PercentSalaryHike        4300 non-null   int64  
 17  StandardHours            4300 non-null   int64  
 18  StockOptionLevel         4300 non-null   int64  
 19  TotalWorkingYears        4300 non-null   float64
 20  TrainingTimesLastYear    4300 non-null   int64  
 21  YearsAtCompany           4300 non-null   int64  
 22  YearsSinceLastPromotion  4300 non-null   int64  
 23  YearsWithCurrManager     4300 non-null   int64  
 24  EnvironmentSatisfaction  4300 non-null   float64
 25  JobSatisfaction          4300 non-null   float64
 26  WorkLifeBalance          4300 non-null   float64
 27  JobInvolvement           4300 non-null   int64  
 28  PerformanceRating        4300 non-null   int64  
dtypes: float64(5), int64(16), int8(8)
memory usage: 772.7 KB
```

```
# Set a random seed for reproducibility
np.random.seed(42)

# Split the data
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


#random forest
# Instantiate the Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42)

# Train the model
rf_model.fit(X_train, y_train)
```

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

```
# Retrieve the feature importance from the model
feature_importance = rf_model.feature_importances_

# Plot the feature importance
features = X.columns
plt.figure(figsize=(10, 6))
plt.bar(features, feature_importance)
plt.xlabel('Features')
plt.ylabel('Importance')
plt.title('Feature Importance in Random Forest')
plt.xticks(rotation=45, ha='right')
plt.show()
```




```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as  plt

# Predict on the test set
y_pred = rf_model.predict(x_test)

# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_model.classes_)
disp.plot(cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
plt.show()
```