Files

..
sample_data
hr_modified_1.csv

```
# upload file
# have removed spaces e.g "Reasearch & Development" to "RD"
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
```

```
df_org = pd.read_csv("hr_modified_1.csv")
df = df_org
```

```
#replace space with nan
df = df.replace(r'^\s+$', np.nan, regex=True)

#df.info()
df[df.isna().any(axis=1)]
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome
162	27	No	Travel_Rarely	NaN	NaN	2
516	33	No	NaN	1392	RAD	3
798	37	Yes	NaN	1373	RAD	2
1273	49	No	Travel_Frequently	279	RAD	8

4 rows × 35 columns

```
# df.info()
# listed are columns with nans

# BusinessTravel -char - 2
# Department    - char   - 1
# EducationField  -char - 1

# DailyRate    - numeric   - 1
# MonthlyIncome   -numeric  - 1
# MonthlyRate     -numeric  - 1
```

```
# identifying BusinessTravel=='Travel_Rarely' and df.EducationField =='Medical' datas
df_medical_travel_rarely = df[(df.BusinessTravel=='Travel_Rarely') & (df.EducationFie
# replacing DailyRate with mean for dataset where BusinessTravel=='Travel_Rarely' and
df_daily_date = df_medical_travel_rarely['DailyRate']
df_df_daily_date = pd.DataFrame(df_daily_date)
df_df_daily_date['DailyRate'] = pd.to_numeric(df_df_daily_date['DailyRate'],errors =
```

```
# df_df_daily_date.mean()
df['DailyRate'] = df['DailyRate'].fillna(836.299694)
```

```
# df[df.isna().any(axis=1)]
#identifying Department for row 162 , Research And Development looks to be most occur
df_medical_travel_rarely.Department.value_counts(dropna=False)
```

```
RAD                255
Sales               62
Human_Resources     10
NaN                  1
Name: Department, dtype: int64
```

```
df[df.isna().any(axis=1)]
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome
162	27	No	Travel_Rarely	836.299694	NaN	2
516	33	No	NaN	1392	RAD	3
798	37	Yes	NaN	1373	RAD	2
1273	49	No	Travel_Frequently	279	RAD	8

4 rows × 35 columns

```
# df[df.isna().any(axis=1)]
df['Department'] = df['Department'].fillna('RAD')
```

```
df.at[798, 'BusinessTravel'] ='Travel_Rarely'
```

```
df.at[516, 'BusinessTravel'] ='Travel_Rarely'
```

```
# checked for average of Attrition = Yes and Gender = Male and JobRole = Laboratory_T
df.at[798, 'MonthlyIncome'] = 2978
```

```
# checked for average of Attrition = No and Gender = Female and JobRole = Research_Sc
df.at[516, 'MonthlyRate'] = 15533
```

```
df
df.at[1273, 'EducationField'] = 'LS'
```

```
df[df.isna().any(axis=1)]
# df_no_missing_values = df
```

Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educa

0 rows × 35 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   object
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   object
 19  MonthlyRate               1470 non-null   object
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(23), object(12)
memory usage: 402.1+ KB
```

```
# columns listed below do not add any value on model.
# col = ['Attrition','EmployeeNumber','Over18']
col = ['EmployeeNumber','Over18']
X = df
for c in col:
  X = X.loc[:, X.columns != c]
y = df.Attrition
```

```
X['DailyRate'] = df['DailyRate'].astype(float)
X['MonthlyIncome'] = df['MonthlyIncome'].astype(float)
X['MonthlyRate'] = df['MonthlyRate'].astype(float)
```

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 33 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   float64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EnvironmentSatisfaction   1470 non-null   int64
 10  Gender                    1470 non-null   object
 11  HourlyRate                1470 non-null   int64
 12  JobInvolvement            1470 non-null   int64
 13  JobLevel                  1470 non-null   int64
 14  JobRole                   1470 non-null   object
 15  JobSatisfaction           1470 non-null   int64
 16  MaritalStatus             1470 non-null   object
 17  MonthlyIncome             1470 non-null   float64
 18  MonthlyRate               1470 non-null   float64
 19  NumCompaniesWorked        1470 non-null   int64
 20  OverTime                  1470 non-null   object
 21  PercentSalaryHike         1470 non-null   int64
 22  PerformanceRating         1470 non-null   int64
 23  RelationshipSatisfaction  1470 non-null   int64
 24  StandardHours             1470 non-null   int64
 25  StockOptionLevel          1470 non-null   int64
 26  TotalWorkingYears         1470 non-null   int64
 27  TrainingTimesLastYear     1470 non-null   int64
 28  WorkLifeBalance           1470 non-null   int64
 29  YearsAtCompany            1470 non-null   int64
 30  YearsInCurrentRole        1470 non-null   int64
 31  YearsSinceLastPromotion   1470 non-null   int64
 32  YearsWithCurrManager      1470 non-null   int64
dtypes: float64(3), int64(22), object(8)
memory usage: 379.1+ KB
```

```
cat_cols=X.select_dtypes(include="object").columns
```

```
cat_cols_x = cat_cols.to_list()
cat_cols_x.remove('Attrition')
cat_cols_x
```

```
['BusinessTravel',
 'Department',
 'EducationField',
 'Gender',
 'JobRole',
 'MaritalStatus',
 'OverTime']
```

```
num_cols= X.select_dtypes(exclude="object").columns
```

```
num_cols
```

```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StandardHours', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
```

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

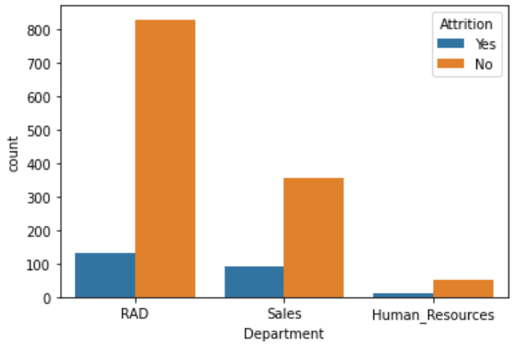
```
num_cols_x = num_cols.to_list()
no_need_to_scale =['Education','EnvironmentSatisfaction','JobInvolvement','JobLevel',
for c in no_need_to_scale:
  num_cols_x.remove(c)
```

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
X[num_cols_x]= sc.fit_transform(X[num_cols_x])
```

```
y.replace('Yes',1, inplace=True)
y.replace('No',0, inplace=True)
```

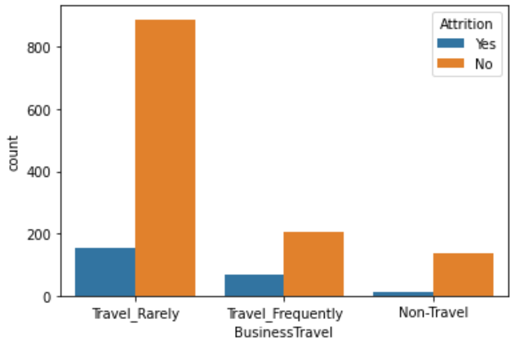
```
import seaborn as sns
sns.countplot (x=X.Department,hue=X.Attrition)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7efda19551f0>



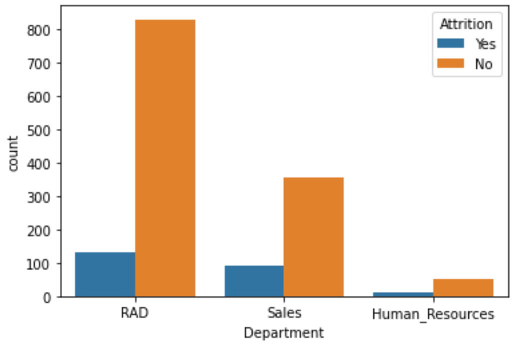

```
sns.countplot (x=X.BusinessTravel,hue=X.Attrition)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7efd94ea5280>




```
sns.countplot (x=X.Department,hue=X.Attrition)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7efd94e7ad60>

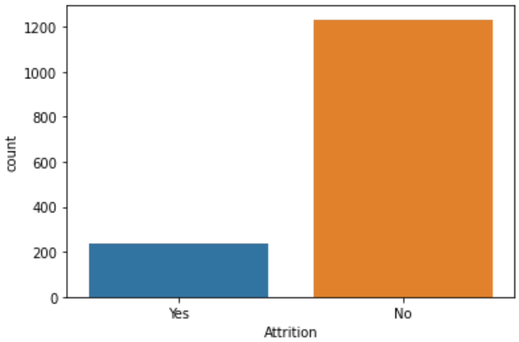



```
sns.countplot (x=X.EducationField,hue=X.Attrition)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd94de79d0>
```


```
sns.countplot (x=X.Attrition )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7efd94f95520>
```



Null accuracy = 1200/1400 = 85.7%

```
# tried One Hot encoding with sklearn however he issue is i could not get "Column Nam
# this causes issue for pairplots.

# from sklearn.preprocessing import OneHotEncoder
# from sklearn.compose import ColumnTransformer

# ct=ColumnTransformer(
# transformers=[('encoder',OneHotEncoder(sparse=False),[0])],remainder="passthrough")
# X_one=ct.fit_transform(X)

# pd.DataFrame(X_one)
# X_one
# ct.get_feature_names_out

# from sklearn.preprocessing import OneHotEncoder
# ohe = OneHotEncoder(sparse=False)
# X_ohe = ohe.fit_transform(X)
# ohe.get_feature_names_out()

X.drop(['Attrition'], axis=1, inplace=True)
dummies_df = pd.get_dummies(X, columns=cat_cols_x)
dummies_df.columns
dummies_df
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	Environm
0	-2.072192	-1.419794	-0.764121	3	0.0	
1	-2.072192	0.023175	0.099639	3	0.0	
2	-2.072192	1.247964	-0.517332	3	0.0	
3	-2.072192	-1.278472	-0.517332	2	0.0	
4	-2.072192	-1.377645	-0.147150	1	0.0	
...	...	...	...	...	...	
1465	2.526886	-0.943763	-0.270544	3	0.0	
1466	2.526886	1.726474	2.320735	3	0.0	
1467	2.526886	0.933089	0.840004	4	0.0	
1468	2.526886	-0.264427	-0.270544	4	0.0	
1469	2.526886	-1.072688	-1.010909	4	0.0	

1470 rows × 53 columns

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(dummies_df,y,test_size=.25)
```

```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
dummies_df.columns
```

```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
       'StandardHours', 'StockOptionLevel', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager',
       'BusinessTravel_Non-Travel', 'BusinessTravel_Travel_Frequently',
       'BusinessTravel_Travel_Rarely', 'Department_Human_Resources',
       'Department_RAD', 'Department_Sales', 'EducationField_Human_Resources',
       'EducationField_LS', 'EducationField_Marketing',
       'EducationField_Medical', 'EducationField_Other',
       'EducationField_Technical_Degree', 'Gender_Female', 'Gender_Male',
       'JobRole_Healthcare_Representative', 'JobRole_Human_Resources',
       'JobRole_Laboratory_Technician', 'JobRole_Manager',
       'JobRole_Manufacturing_Director', 'JobRole_Research_Director',
       'JobRole_Research_Scientist', 'JobRole_Sales_Executive',
       'JobRole_Sales_Representative', 'MaritalStatus_Divorced',
       'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_No',
       'OverTime_Yes'],
      dtype='object')
```

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.7336956521739131
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
cm
```

```
array([[252,  64],
       [ 34,  18]])
```

```
# !pip install sklearn.externals.six
# from sklearn.externals.six import StringIO
# from IPython.display import Image
# from sklearn.tree import export_graphviz
# import pydotplus

# dot_data = StringIO()
# export_graphviz(clf, out_file=dot_data,
#                 filled=True, rounded=True,
#                 special_characters=True)

# graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
# Image(graph.create_png())

# https://machinelearningmastery.com/calculate-feature-importance-with-python/
from matplotlib import pyplot
# get importance
importance = clf.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature: 0, Score: 0.06610
Feature: 1, Score: 0.09201
Feature: 2, Score: 0.06557
Feature: 3, Score: 0.02035
Feature: 4, Score: 0.00000
Feature: 5, Score: 0.03107
Feature: 6, Score: 0.07556
Feature: 7, Score: 0.00879
Feature: 8, Score: 0.00561
Feature: 9, Score: 0.01515
Feature: 10, Score: 0.11230
Feature: 11, Score: 0.03029
Feature: 12, Score: 0.01608
Feature: 13, Score: 0.05648
Feature: 14, Score: 0.00433
Feature: 15, Score: 0.02221
Feature: 16, Score: 0.00000
Feature: 17, Score: 0.05441
Feature: 18, Score: 0.00325
Feature: 19, Score: 0.00709
Feature: 20, Score: 0.02838
Feature: 21, Score: 0.03732
Feature: 22, Score: 0.02209
Feature: 23, Score: 0.03477
Feature: 24, Score: 0.00000
Feature: 25, Score: 0.00000
Feature: 26, Score: 0.01895
Feature: 27, Score: 0.00000
Feature: 28, Score: 0.00839
Feature: 29, Score: 0.00000
Feature: 30, Score: 0.00000
Feature: 31, Score: 0.00000
Feature: 32, Score: 0.00000
Feature: 33, Score: 0.00000
Feature: 34, Score: 0.00780
Feature: 35, Score: 0.00000
Feature: 36, Score: 0.01482
Feature: 37, Score: 0.00487
Feature: 38, Score: 0.00000
Feature: 39, Score: 0.00000
Feature: 40, Score: 0.00000
Feature: 41, Score: 0.00711
Feature: 42, Score: 0.00000
Feature: 43, Score: 0.00000
Feature: 44, Score: 0.00000
Feature: 45, Score: 0.00585
Feature: 46, Score: 0.03017
Feature: 47, Score: 0.00000
Feature: 48, Score: 0.00000
Feature: 49, Score: 0.00000
Feature: 50, Score: 0.00000
Feature: 51, Score: 0.05918
Feature: 52, Score: 0.03365
```

Following come out to be important features from CART

1	0.07703	'DailyRate',
18	0.06661	'TotalWorkingYears',
10	0.06341	'MonthlyIncome',
21	0.0603	'YearsAtCompany',
8	0.06009	'JobLevel',
2	0.05733	'DistanceFromHome',
11	0.05533	'MonthlyRate',
0	0.04628	'Age',
9	0.04415	'JobSatisfaction',
51	0.04333	'OverTime_No',
12	0.04019	'NumCompaniesWorked',
13	0.03464	'PercentSalaryHike',
6	0.03378	'HourlyRate',
19	0.02847	'TrainingTimesLastYear',
20	0.02695	'WorkLifeBalance',
37	0.02613	'Gender_Female',
22	0.02596	'YearsInCurrentRole',
7	0.02153	'JobInvolvement',
52	0.02075	'OverTime_Yes'
23	0.02	'YearsSinceLastPromotion',

```
cat_cols_feat = dummies_df.select_dtypes(include="object").columns
num_cols_feat = dummies_df.select_dtypes(exclude="object").columns

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.compose import ColumnTransformer

# Cat Tranformer
categorical_tranformer= Pipeline(steps=[('ohe',OneHotEncoder())])

# Num tranfoemr
numerical_tranformer = Pipeline(steps=[('sc',StandardScaler())])

col_tranform= ColumnTransformer(transformers=[
                                     ('cat_feat',categorical_tranformer,cat_cols_fea
                                    ('num_feat',numerical_tranformer,num_cols_feat),
                                       ],
                                    remainder='passthrough')

X_trans = col_tranform.fit_transform(dummies_df)
type(X_trans)
```

```
numpy.ndarray
```

```
my_pipeline= Pipeline(steps=[('first_pipe',col_tranform),('model',LogisticRegression(
# X_train
```

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

```
Pipeline(steps=[('first_pipe',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_feat',
                                                  Pipeline(steps=[('ohe',

OneHotEncoder())]),
                                                  Index([], dtype='object')),
                                                 ('num_feat',
                                                  Pipeline(steps=[('sc',

StandardScaler())]),
                                                  Index(['Age', 'DailyRate',
'DistanceFromHome', 'Education', 'EmployeeCount',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'J...
       'JobRole_Laboratory_Technician', 'JobRole_Manager',
       'JobRole_Manufacturing_Director', 'JobRole_Research_Director',
       'JobRole_Research_Scientist', 'JobRole_Sales_Executive',
       'JobRole_Sales_Representative', 'MaritalStatus_Divorced',
       'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_No',
       'OverTime_Yes'],
      dtype='object'))])),
                ('model', LogisticRegression())])
```

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

```
from sklearn.metrics import accuracy_score
pd.Series(accuracy_score(y_test,y_pred))
```

```
0    0.88587
dtype: float64
```

```
from sklearn import metrics
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[301,  15],
       [ 27,  25]])
```

```
metrics.recall_score(y_test,y_pred)
```

```
0.4807692307692308
```

```
from sklearn.ensemble import RandomForestClassifier
my_pipeline= Pipeline(steps=[('first_pipe',col_tranform),('model',RandomForestClassif
my_pipeline.fit(X_train,y_train)
```

```
Pipeline(steps=[('first_pipe',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat_feat',
                                                  Pipeline(steps=[('ohe',

OneHotEncoder())]),
                                                  Index([], dtype='object')),
                                                 ('num_feat',
                                                  Pipeline(steps=[('sc',

StandardScaler())]),
                                                  Index(['Age', 'DailyRate',
```

```
      'DistanceFromHome', 'Education', 'EmployeeCount',
             'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'J...
             'JobRole_Laboratory_Technician', 'JobRole_Manager',
             'JobRole_Manufacturing_Director', 'JobRole_Research_Director',
             'JobRole_Research_Scientist', 'JobRole_Sales_Executive',
             'JobRole_Sales_Representative', 'MaritalStatus_Divorced',
             'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_No',
             'OverTime_Yes'],
            dtype='object'))])),
                      ('model', RandomForestClassifier())])
```

```
my_pipeline.fit(X_train,y_train)
y_pred= my_pipeline.predict(X_test)
```

```
from sklearn.metrics import accuracy_score
pd.Series(accuracy_score(y_test,y_pred))
```

```
0    0.877717
dtype: float64
```

```
from sklearn import metrics
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[313,   3],
       [ 42,  10]])
```

```
metrics.precision_score(y_test,y_pred)
```

```
0.7692307692307693
```

```
metrics.recall_score(y_test,y_pred)
```

```
0.19230769230769232
```

```
metrics.f1_score(y_test,y_pred)
```

```
0.3076923076923077
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# # Define Parameters
# max_depth=[2, 8, 16]
# n_estimators = [64, 128, 256]
# param_grid = dict(max_depth=max_depth, n_estimators=n_estimators)

# # Build the grid search
# dfrst = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth)
# grid = GridSearchCV(estimator=dfrst, param_grid=param_grid, cv = 5, scoring ='accur
# grid_results = grid.fit(X_train, y_train)


# # Summarize the results in a readable format
# print("Best: {0}, using {1}".format(grid_results.cv_results_['accuracy'], grid_resu
# results_df = pd.DataFrame(grid_results.cv_results_)
# results_df


# rf = RandomForestClassifier()

# # grid search cv
# grid_space={'max_depth':[3,5,10,None],
#               'n_estimators':[10,100,200],
#               'max_features':[1,3,5,7],
#               'min_samples_leaf':[1,2,3],
#               'min_samples_split':[1,2,3]
#            }

# grid = GridSearchCV(rf,param_grid=grid_space,cv=3,scoring='accuracy')
# model_grid = grid.fit(X_train,y_train)

# # grid search results
# print('Best grid search hyperparameters are: '+str(model_grid.best_params_))
# print('Best grid search score is: '+str(model_grid.best_score_))
```

```
!pip install sweetviz
import sweetviz
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wh
Requirement already satisfied: sweetviz in /usr/local/lib/python3.8/dist-packag
Requirement already satisfied: importlib-resources>=1.2.0 in /usr/local/lib/pyt
Requirement already satisfied: matplotlib>=3.1.3 in /usr/local/lib/python3.8/di
Requirement already satisfied: jinja2>=2.11.1 in /usr/local/lib/python3.8/dist-
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: tqdm>=4.43.0 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: pandas!=1.0.0,!=1.0.1,!=1.0.2,>=0.25.3 in /usr/l
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.8/dist-p
Requirement already satisfied: zipp>=3.1.0 in /usr/local/lib/python3.8/dist-pac
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.8/dis
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.8
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.8/di
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packag
```

```
# my_report = sweetviz.compare([X, "Train"] "Attrition")
# my_report =sweetviz.analyze(source=df_org,target_feat='Attrition')
```

```
# my_report.show_html("Report.html")
```

** Gaussian naive bayes ** **this is the best fit of all**

```
# X_train,X_test,y_train,y_test
from sklearn.naive_bayes import GaussianNB
GNBclf = GaussianNB()
model = GNBclf.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
metrics.recall_score(y_test,y_pred)
```

```
0.7115384615384616
```

```
metrics.precision_score(y_test,y_pred)
```

```
0.25170068027210885
```

```
metrics.f1_score(y_test,y_pred)
```

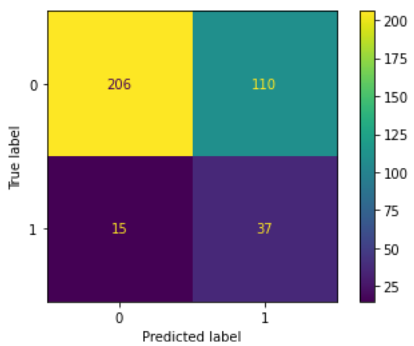
```
0.37185929648241206
```

```
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[206, 110],
       [ 15,  37]])
```

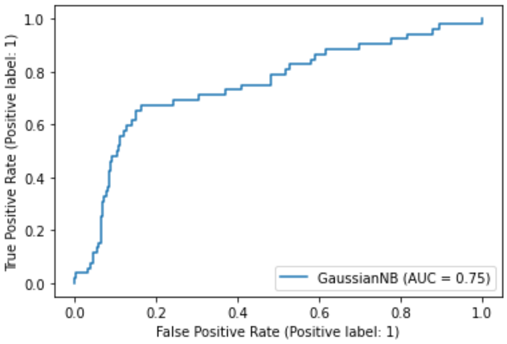
```
from sklearn.metrics  import confusion_matrix, ConfusionMatrixDisplay
metrics.confusion_matrix(y_test,y_pred)
# metrics.recall_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7efd9c4b1e50>
```




```
from numpy import argmax
from sklearn.metrics import plot_roc_curve
roc = plot_roc_curve(model, X_test, y_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureW
  warnings.warn(msg, category=FutureWarning)
```




```
y_pred_prob = model.predict_proba(X_test)[:,1]
```

```
y_pred_prob
```

```
       2.92002670e-05, 2.76470963e-03, 1.31719269e-17, 6.73637676e-01,
       9.96560115e-01, 8.29874625e-13, 8.70571987e-01, 9.96415717e-01,
       6.96922333e-02, 8.03420852e-01, 1.62506069e-04, 4.95565050e-01,
       2.87049549e-04, 8.78774642e-05, 1.27389193e-01, 3.05996456e-02,
       7.92601619e-01, 1.55333892e-01, 3.60842166e-04, 9.88196492e-01,
       9.81841915e-04, 2.34407597e-45, 1.11041725e-04, 1.24308234e-01,
       7.38001888e-07, 3.00622466e-14, 4.68746585e-01, 9.69055437e-01,
       9.99703617e-01, 9.24536918e-01, 4.59105778e-08, 4.52572990e-39,
       2.36668103e-06, 9.97039016e-01, 9.29786616e-03, 9.31145361e-10,
       2.09595470e-43, 8.62356416e-01, 1.64683281e-07, 1.10359321e-40,
       7.58234011e-04, 9.38404343e-01, 7.67031960e-01, 9.99998784e-01,
       9.61046482e-01, 1.10745361e-42, 1.42515397e-02, 3.54653565e-03,
       8.28224104e-01, 4.63347459e-19, 7.61644690e-01, 8.57443457e-01,
       9.92789120e-01, 9.99997591e-01, 6.24181506e-02, 9.99974950e-01,
       9.99999610e-01, 4.65120980e-01, 2.06160444e-12, 5.35875698e-01,
       2.15661537e-02, 6.54087677e-03, 1.63745887e-02, 3.63879109e-10,
       4.01967434e-04, 3.98726152e-08, 1.01332510e-01, 3.32597937e-41,
       9.95906340e-01, 6.17867233e-01, 4.76788213e-01, 1.57245312e-16,
       1.39261951e-39, 6.53084330e-08, 3.66512600e-02, 9.99999962e-01,
       5.47572038e-01, 3.14351175e-07, 1.10615369e-01, 9.97903631e-01,
       2.30076083e-01, 1.47723295e-01, 9.77505617e-01, 9.99345438e-01,
       9.83873092e-01, 3.42122825e-07, 1.38338784e-43, 9.99745825e-01,
       1.36977564e-03, 3.70122251e-16, 1.40451114e-41, 4.78597450e-06,
       6.85867918e-04, 9.13594565e-02, 1.19694528e-01, 2.29196568e-42,
       2.78150005e-02, 1.72588120e-04, 9.82773558e-01, 9.65937060e-01,
       9.91786794e-01, 1.80912073e-06, 8.26930003e-01, 1.98495155e-15,
       1.80894966e-01, 9.43889204e-01, 9.39557314e-02, 6.64074663e-06,
       9.90204099e-01, 6.54753124e-01, 4.24314004e-02, 9.07241235e-01,
       9.58881698e-07, 1.61409296e-03, 2.55808527e-19, 2.93798674e-05,
       5.85966764e-01, 1.60004222e-04, 9.98478941e-01, 1.51521245e-02,
       7.15443293e-03, 7.32402621e-07, 1.87017944e-03, 8.70875696e-01,
       1.68789278e-01, 9.76159501e-01, 2.02915232e-40, 2.83695382e-07,
       9.48667882e-01, 1.40773580e-19, 1.07823205e-03, 8.81682064e-01,
       4.46441865e-44, 2.04105354e-05, 9.60613367e-01, 6.11504540e-04,
       2.02543374e-02, 1.16420107e-01, 1.67542057e-03, 3.59313004e-01,
       3.32365970e-01, 6.92834219e-04, 9.98087855e-01, 1.76200596e-01,
       6.43208447e-02, 3.42387211e-06, 2.45052921e-01, 3.02991933e-05,
       8.05800237e-01, 3.82873051e-16, 9.46490825e-01, 6.66973730e-01,
       6.25620561e-07, 4.43688577e-01, 4.27236924e-44, 6.75321779e-20,
       6.73530241e-01, 9.99418741e-01, 7.64761553e-01, 6.97478310e-01,
       2.90798735e-03, 4.02793650e-42, 4.55574016e-10, 4.72018588e-01,
       1.90632459e-01, 8.76387166e-01, 7.63871285e-01, 2.68990462e-01,
       9.63846200e-01, 6.40349178e-01, 9.87829345e-01, 1.11274476e-03,
       9.98560038e-01, 3.20039453e-12, 3.62807567e-02, 4.58669892e-01,
       1.09112950e-01, 1.58309372e-07, 5.04427204e-07, 9.63841756e-01,
       6.82147313e-01, 5.99393793e-17, 3.01688974e-01, 9.99999995e-01,
       6.63483724e-01, 6.63301734e-13, 1.52142015e-06, 1.39567167e-01,
       8.43503483e-01, 5.12323044e-01, 9.99901402e-01, 7.54094866e-01,
       6.47545256e-07, 2.15976502e-01, 3.04738419e-01, 9.89597685e-01,
       9.53320231e-01, 1.11113802e-40, 4.51665067e-01, 9.99998614e-01,
       8.69879335e-01, 9.99442139e-01, 1.12021546e-02, 1.53756696e-01,
       6.26364445e-01, 5.20946366e-01, 5.92528415e-02, 5.63400550e-07,
       8.09149272e-01, 9.97225615e-01, 9.93946040e-01, 8.62359644e-02,
       6.53541422e-01, 2.94899158e-01, 9.76651182e-01, 8.99417544e-01,
       1.14268322e-06, 7.64728586e-08, 1.85814569e-01, 3.21990546e-07,
       2.66540017e-02, 8.96057027e-01, 2.62364618e-01, 6.65928683e-01,
       6.28303843e-45, 5.70863676e-01, 1.23020917e-03, 5.36148760e-01,
       6.82893115e-03, 6.99930787e-07, 8.37429588e-01, 7.61274252e-20])
```

```
dummies_df.columns
```

```
Index(['Age', 'DailyRate', 'DistanceFromHome', 'Education', 'EmployeeCount',
       'EnvironmentSatisfaction', 'HourlyRate', 'JobInvolvement', 'JobLevel',
       'JobSatisfaction', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'PerformanceRating', 'RelationshipSatisfaction',
```

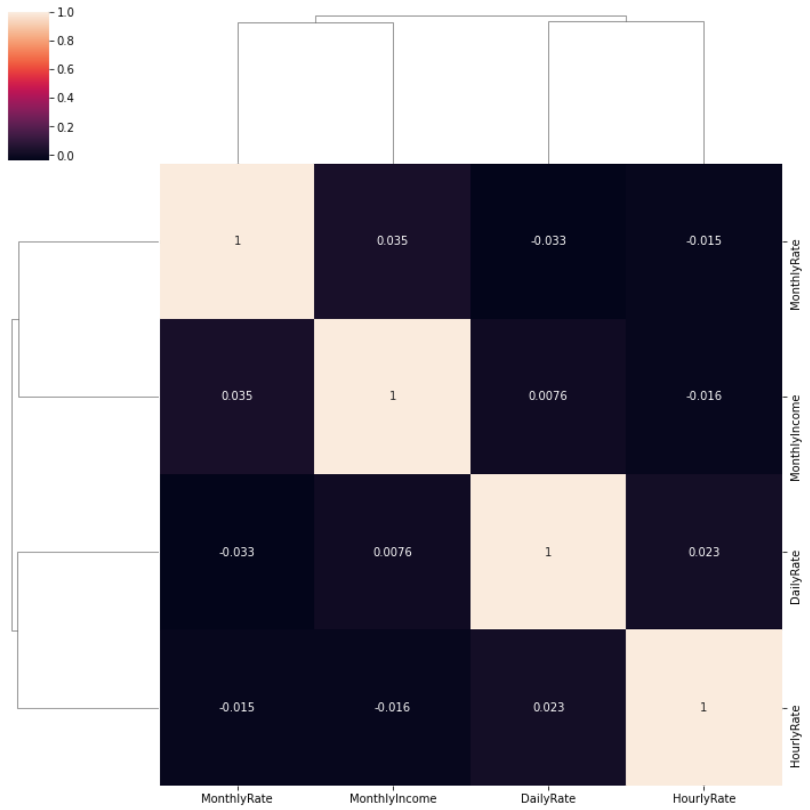
```
            'StandardHours', 'StockOptionLevel', 'TotalWorkingYears',
            'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
            'YearsInCurrentRole', 'YearsSinceLastPromotion', 'YearsWithCurrManager',
            'BusinessTravel_Non-Travel', 'BusinessTravel_Travel_Frequently',
            'BusinessTravel_Travel_Rarely', 'Department_Human_Resources',
            'Department_RAD', 'Department_Sales', 'EducationField_Human_Resources',
            'EducationField_LS', 'EducationField_Marketing',
            'EducationField_Medical', 'EducationField_Other',
            'EducationField_Technical_Degree', 'Gender_Female', 'Gender_Male',
            'JobRole_Healthcare_Representative', 'JobRole_Human_Resources',
            'JobRole_Laboratory_Technician', 'JobRole_Manager',
            'JobRole_Manufacturing_Director', 'JobRole_Research_Director',
            'JobRole_Research_Scientist', 'JobRole_Sales_Executive',
            'JobRole_Sales_Representative', 'MaritalStatus_Divorced',
            'MaritalStatus_Married', 'MaritalStatus_Single', 'OverTime_No',
            'OverTime_Yes'],
           dtype='object')
```

Trying to identify if 'DailyRate','HourlyRate','MonthlyRate','MonthlyIncome' are co-related and any of these can be eliminated. Checking this since Naive bayes treats each dimension independently and would work better if they are not related.

```
dummies_df_corr = dummies_df[['DailyRate','HourlyRate','MonthlyRate','MonthlyIncome']
```

```
correlations = dummies_df_corr.corr()
#sns.heatmap(correlations, xticklabels=correlations.columns, yticklabels=correlations
sns.clustermap(correlations, xticklabels=correlations.columns, yticklabels=correlatio
```

```
<seaborn.matrix.ClusterGrid at 0x7efd94c813d0>
```



Do not find strong corelation between these columns - 'DailyRate','HourlyRate','MonthlyRate','MonthlyIncome'. no Pearson's correlation value is above 0.5 Trying to eliminate corelated columns for Naive bayes.

```
dummies_df.info()
```

```
Data columns (total 53 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
```

```
 2   DistanceFromHome                  1470 non-null   float64
 3   Education                         1470 non-null   int64
 4   EmployeeCount                     1470 non-null   float64
 5   EnvironmentSatisfaction           1470 non-null   int64
 6   HourlyRate                        1470 non-null   float64
 7   JobInvolvement                    1470 non-null   int64
 8   JobLevel                          1470 non-null   int64
 9   JobSatisfaction                   1470 non-null   int64
 10  MonthlyIncome                     1470 non-null   float64
 11  MonthlyRate                       1470 non-null   float64
 12  NumCompaniesWorked                1470 non-null   float64
 13  PercentSalaryHike                 1470 non-null   int64
 14  PerformanceRating                 1470 non-null   int64
 15  RelationshipSatisfaction          1470 non-null   int64
 16  StandardHours                     1470 non-null   float64
 17  StockOptionLevel                  1470 non-null   int64
 18  TotalWorkingYears                 1470 non-null   float64
 19  TrainingTimesLastYear             1470 non-null   float64
 20  WorkLifeBalance                   1470 non-null   int64
 21  YearsAtCompany                    1470 non-null   float64
 22  YearsInCurrentRole                1470 non-null   float64
 23  YearsSinceLastPromotion           1470 non-null   float64
 24  YearsWithCurrManager              1470 non-null   float64
 25  BusinessTravel_Non-Travel         1470 non-null   uint8
 26  BusinessTravel_Travel_Frequently  1470 non-null   uint8
 27  BusinessTravel_Travel_Rarely      1470 non-null   uint8
 28  Department_Human_Resources        1470 non-null   uint8
 29  Department_RAD                    1470 non-null   uint8
 30  Department_Sales                  1470 non-null   uint8
 31  EducationField_Human_Resources    1470 non-null   uint8
 32  EducationField_LS                 1470 non-null   uint8
 33  EducationField_Marketing          1470 non-null   uint8
 34  EducationField_Medical            1470 non-null   uint8
 35  EducationField_Other              1470 non-null   uint8
 36  EducationField_Technical_Degree   1470 non-null   uint8
 37  Gender_Female                     1470 non-null   uint8
 38  Gender_Male                       1470 non-null   uint8
 39  JobRole_Healthcare_Representative 1470 non-null   uint8
 40  JobRole_Human_Resources           1470 non-null   uint8
 41  JobRole_Laboratory_Technician     1470 non-null   uint8
 42  JobRole_Manager                   1470 non-null   uint8
 43  JobRole_Manufacturing_Director    1470 non-null   uint8
 44  JobRole_Research_Director         1470 non-null   uint8
 45  JobRole_Research_Scientist        1470 non-null   uint8
 46  JobRole_Sales_Executive           1470 non-null   uint8
 47  JobRole_Sales_Representative      1470 non-null   uint8
 48  MaritalStatus_Divorced            1470 non-null   uint8
 49  MaritalStatus_Married             1470 non-null   uint8
 50  MaritalStatus_Single              1470 non-null   uint8
 51  OverTime_No                       1470 non-null   uint8
 52  OverTime_Yes                      1470 non-null   uint8
dtypes: float64(15), int64(10), uint8(28)
memory usage: 327.4 KB
```

```
dummies_df_corr = dummies_df[['DailyRate','HourlyRate','MonthlyRate','MonthlyIncome']
```

```
from scipy.stats import pearsonr
# calculate Pearson's correlation
# corr, _ = pearsonr(dummies_df_corr['DailyRate'], dummies_df_corr['HourlyRate'])
corr, _ = pearsonr(dummies_df_corr['DailyRate'], dummies_df_corr['HourlyRate'])
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 0.023
```

```
param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose
nbModel_grid.fit(X_test, y_test)
print(nbModel_grid.best_estimator_)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
GaussianNB(var_smoothing=0.0657933224657568)
```

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

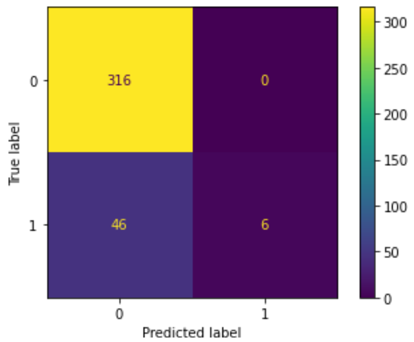
```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred), ": is the confusion matrix")

from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")
```

```
from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred), ": is the recall score")
```

```
[[316   0]
 [ 46   6]] : is the confusion matrix
0.20689655172413793 : is the f1 score
0.11538461538461539 : is the recall score
```

```
from sklearn.metrics  import confusion_matrix, ConfusionMatrixDisplay
metrics.confusion_matrix(y_test,y_pred)
# metrics.recall_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7efd94bc2c40>
```

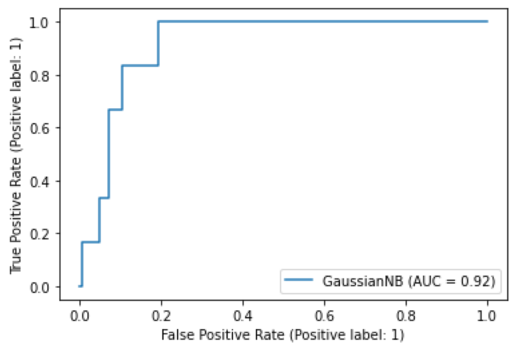



```
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")
```

```
0.20689655172413793 : is the f1 score
```

```
roc = plot_roc_curve(model, X_test, y_pred)
```

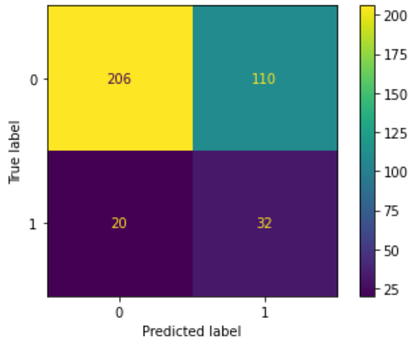
```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureW
  warnings.warn(msg, category=FutureWarning)
```




```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE

# define pipeline
steps = [('over', SMOTE()), ('model', GaussianNB())]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# scores = cross_val_score(pipeline, X_train, y_train, scoring='roc_auc', cv=cv, n_jo
# print('Mean ROC AUC: %.3f' % mean(scores))
# scores = cross_val_score(pipeline, X_train, y_train, scoring='recall', cv=cv, n_job
# print('Mean Recall: %.3f' % mean(scores))
my_pipeline= Pipeline(steps=[('over', SMOTE()), ('model', GaussianNB())])
my_pipeline.fit(X_train,y_train)
y_pred= my_pipeline.predict(X_test)
from sklearn.metrics import accuracy_score
# pd.Series(accuracy_score(y_test,y_pred))
from sklearn.metrics  import confusion_matrix, ConfusionMatrixDisplay
metrics.confusion_matrix(y_test,y_pred)
```

```
# metrics.recall_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7efd9513e790>
```




```
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")
```

```
0.3298969072164949 : is the f1 score
```

```
from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred), ": is the recall score")
```

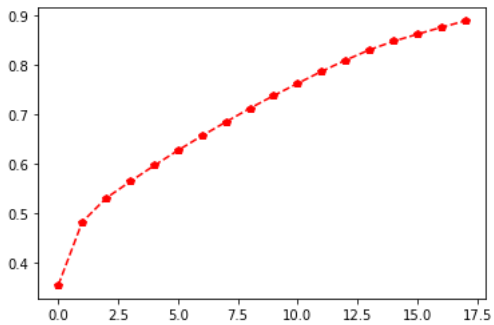
```
0.6153846153846154 : is the recall score
```

Double-click (or enter) to edit

Applying PCA - this is expected to give better results for Naive Bayes

```
# Set the n_components=3
from sklearn.decomposition import PCA

principal=PCA(n_components=18)
principal.fit(dummies_df)
Principal_components =principal.transform(dummies_df)

df_pca = pd.DataFrame(data = Principal_components, columns = ['PC 1', 'PC 2', 'PC 3',

# Check the dimensions of data after PCA
print(df_pca.shape)

# check how much variance is explained by each principal component
print(principal.explained_variance_ratio_)

plt.plot(principal.explained_variance_ratio_.cumsum(), marker='p', color ='r', ls ='-
```

```
(1470, 18)
[0.35372911 0.12762878 0.04834421 0.03329    0.03232717 0.03083077
 0.0293474  0.02798081 0.02747513 0.02583569 0.02461236 0.02439993
 0.02255323 0.02089638 0.01732752 0.01426953 0.01393021 0.01299408]
[<matplotlib.lines.Line2D at 0x7efd945f77f0>]
```

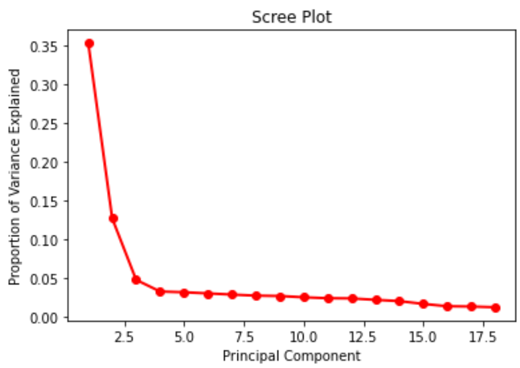



```
df_pca
```

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	
0	-2.161642	-3.509407	-0.454804	-0.737208	-0.597898	-0.233970	1.257237	1.
1	-3.118361	-3.531994	-0.678476	-0.733179	-0.239743	1.783552	-0.037913	0.
2	-1.160822	-3.413908	-0.335248	0.723923	-0.057230	-1.569209	0.188438	-0.
3	-0.144435	-3.588790	-0.691427	-0.782406	0.447589	-1.857597	0.718676	0.
4	-3.146963	-3.642499	-0.682007	-0.330343	-0.916150	-1.331816	0.636305	0.
...	...	...	...	...	...	...	...	
1465	-4.411483	7.093741	0.426234	1.759403	0.451193	-1.670444	1.624432	-0.
1466	3.647571	5.159915	-1.202104	2.207383	-1.096076	-1.221637	-1.281392	-1.
1467	-1.221912	0.007600	2.969123	2.692633	0.451296	-1.371839	-1.137869	0.
1468	2.751522	1.539085	-0.495795	-0.702721	1.532693	0.166326	-0.883045	0.
1469	4.813056	0.765020	3.242149	-0.959570	0.318024	-0.015955	0.068253	0.

1470 rows × 18 columns

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

PC_values = np.arange(principal.n_components_) + 1
plt.plot(PC_values, principal.explained_variance_ratio_, 'ro-', linewidth=2)
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Proportion of Variance Explained')
plt.show()
```




```
X_train,X_test,y_train,y_test = train_test_split(df_pca,y,test_size=.25)

param_grid_nb = {
    'var_smoothing': np.logspace(0,-9, num=100)
}

from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV
nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb, verbose
nbModel_grid.fit(X_train, y_train)
print(nbModel_grid.best_estimator_)
```

```
Fitting 10 folds for each of 100 candidates, totalling 1000 fits
GaussianNB(var_smoothing=0.01519911082952933)
```

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

```
from sklearn.naive_bayes import GaussianNB
GNBclf = GaussianNB()
model = GNBclf.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

```
y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

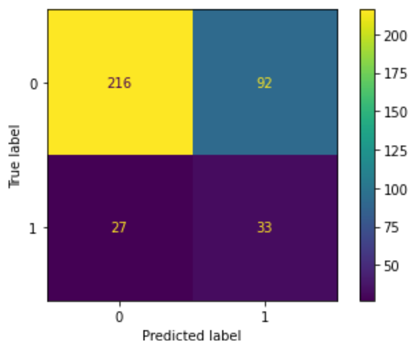
```
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred), ": is the confusion matrix")

from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")

from sklearn.metrics import recall_score
print(recall_score(y_test, y_pred), ": is the recall score")
```

```
[[305   3]
 [ 51   9]] : is the confusion matrix
0.24999999999999997 : is the f1 score
0.15 : is the recall score
```

```
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE

# define pipeline
steps = [('over', SMOTE()), ('model', GaussianNB())]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# scores = cross_val_score(pipeline, X_train, y_train, scoring='roc_auc', cv=cv, n_jo
# print('Mean ROC AUC: %.3f' % mean(scores))
# scores = cross_val_score(pipeline, X_train, y_train, scoring='recall', cv=cv, n_job
# print('Mean Recall: %.3f' % mean(scores))
my_pipeline= Pipeline(steps=[('over', SMOTE()), ('model', GaussianNB())])
my_pipeline.fit(X_train,y_train)
y_pred= my_pipeline.predict(X_test)
from sklearn.metrics import accuracy_score
# pd.Series(accuracy_score(y_test,y_pred))
from sklearn.metrics  import confusion_matrix, ConfusionMatrixDisplay
metrics.confusion_matrix(y_test,y_pred)
# metrics.recall_score(y_test,y_pred)
cm = confusion_matrix(y_test, y_pred, labels=clf.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=clf.classes_)
disp.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7efd945248b0>
```




```
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred), ": is the f1 score")
```

```
0.3567567567567568 : is the f1 score
```

Disk 83.31 GB available

0s completed at 12:47 PM