SECTION B

MOUNT GOOGLE DRIVE

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

TASK 1

IMPORT LIBRARIES

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

TASK 2

LOAD A DATASET

```
df = pd.read_csv('/content/drive/MyDrive/DAML/Assignment/staff_dataset.csv')
df
```

	Age	BusinessTravel	MonthlyIncome	JobSatisfaction	Bonus	Department	DistanceFromHome	Education	E
0	41	Travel_Rarely	5993	4	17979	Sales	1	2	
1	49	Travel_Frequently	5130	2	20520	Research & Development	8	1	
2	37	Travel_Rarely	2090	3	6270	Research & Development	2	2	
3	33	Travel_Frequently	2909	3	8727	Research & Development	3	4	
4	27	Travel_Rarely	3468	2	10404	Research & Development	2	1	
...	...	...	...	...	...	...	...	...	
1465	36	Travel_Frequently	2571	4	7713	Research & Development	23	2	
1466	39	Travel_Rarely	9991	1	29973	Research & Development	6	1	
1467	27	Travel_Rarely	6142	2	24568	Research & Development	4	3	
1468	49	Travel_Frequently	5390	2	16170	Sales	2	3	
1469	34	Travel_Rarely	4404	3	13212	Research & Development	8	3	

1470 rows × 24 columns

DATASET DESCRIPTION

```
#DATASET INFORMATION
print("\n DATASET INFORMATION")
df.info()
```

```
 DATASET INFORMATION
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 24 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   BusinessTravel            1470 non-null   object
 2   MonthlyIncome             1470 non-null   int64
 3   JobSatisfaction           1470 non-null   int64
 4   Bonus                     1470 non-null   int64
 5   Department                1470 non-null   object
 6   DistanceFromHome          1470 non-null   int64
 7   Education                 1470 non-null   int64
 8   EducationField            1470 non-null   object
 9   EmployeeCount             1470 non-null   int64
 10  EmployeeNumber            1470 non-null   int64
 11  EnvironmentSatisfaction   1470 non-null   int64
 12  Gender                    1470 non-null   object
 13  JobLevel                  1470 non-null   int64
 14  JobRole                   1470 non-null   object
 15  MaritalStatus             1470 non-null   object
 16  PerformanceRating         1470 non-null   int64
 17  StockOptionLevel          1470 non-null   int64
 18  TrainingTimesLastYear     1470 non-null   int64
 19  WorkLifeBalance           1470 non-null   int64
 20  YearsAtCompany            1470 non-null   int64
```

```
 21  YearsSinceLastPromotion  1470 non-null   int64
 22  OverTime                 1470 non-null   object
 23  Attrition                1470 non-null   object
dtypes: int64(16), object(8)
memory usage: 275.8+ KB
```

Based on the output above, it is shown that all of the variables have two datatypes which are integer and object which suits the value type of the variables, thus it is not needed to change the datatypes

```
#STASTISTICAL SUMMARY
print("\n STATISTICAL SUMMARY")
df.describe()
```

```
 STATISTICAL SUMMARY
```

	Age	MonthlyIncome	JobSatisfaction	Bonus	DistanceFromHome	Education	EmployeeCount
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.000000	1470.0
mean	36.923810	6502.931293	2.728571	20479.501361	9.192517	2.912925	1.0
std	9.135373	4707.956783	1.102846	15066.272964	8.106864	1.024165	0.0
min	18.000000	1009.000000	1.000000	3027.000000	1.000000	1.000000	1.0
25%	30.000000	2911.000000	2.000000	9333.750000	2.000000	2.000000	1.0
50%	36.000000	4919.000000	3.000000	15484.500000	7.000000	3.000000	1.0
75%	43.000000	8379.000000	4.000000	26103.750000	14.000000	4.000000	1.0
max	60.000000	19999.000000	4.000000	79892.000000	29.000000	5.000000	1.0

Based on the output, we could observe the details of the dataset:

- count: the total number of non-empty rows in a feature.
- mean: the average or mean value of each feature.
- std: value of standard deviation of each feature.
- min: minimum value.
- max: maximum value.
- 25%, 50%, 75%: percentile/quartile of each feature. Based on all of this information, it could be use in some cases, for example: if there are missing values in a specific feature, that missing value can be replaced with the mean value.

```
#DATASET SHAPE
print("\n DATASET SHAPE")
df.shape
```

```
 DATASET SHAPE
(1470, 24)
```

Based on the output, it is shown that the dataset consists of 1470 observation units, and 9 variables.

DATA CLEANING

```
#DATA CLEANING
#DROPPING DUPLICATED VALUES
df=df.drop_duplicates()
df
```

	Age	BusinessTravel	MonthlyIncome	JobSatisfaction	Bonus	Department	DistanceFromHome	Education	E
0	41	Travel_Rarely	5993	4	17979	Sales	1	2	
1	49	Travel_Frequently	5130	2	20520	Research & Development	8	1	
2	37	Travel_Rarely	2090	3	6270	Research & Development	2	2	
3	33	Travel_Frequently	2909	3	8727	Research & Development	3	4	
4	27	Travel_Rarely	3468	2	10404	Research & Development	2	1	
...	...	...	...	...	...	...	...	...	
1465	36	Travel_Frequently	2571	4	7713	Research & Development	23	2	
1466	39	Travel_Rarely	9991	1	29973	Research & Development	6	1	
1467	27	Travel_Rarely	6142	2	24568	Research & Development	4	3	
1468	49	Travel_Frequently	5390	2	16170	Sales	2	3	
1469	34	Travel_Rarely	4404	3	13212	Research & Development	8	3	

1470 rows × 24 columns

Based on the output, as there is no duplicated values, there is no dropped values

```
#CHECKING FOR MISSING VALUES
df.isnull().sum()
```

```
Age                         0
BusinessTravel              0
MonthlyIncome               0
JobSatisfaction             0
Bonus                       0
```

```
Department                 0
DistanceFromHome           0
Education                  0
EducationField             0
EmployeeCount              0
EmployeeNumber             0
EnvironmentSatisfaction    0
Gender                     0
JobLevel                   0
JobRole                    0
MaritalStatus              0
PerformanceRating          0
StockOptionLevel           0
TrainingTimesLastYear      0
WorkLifeBalance            0
YearsAtCompany             0
YearsSinceLastPromotion    0
OverTime                   0
Attrition                  0
dtype: int64
```

Based on the output, there is no null values

TASK 4

LABEL ENCODING

```
# LABEL ENCODING
encoder = preprocessing.LabelEncoder()

# Encode labels in column 'Business Travel' and 'Attrition'
df['BusinessTravel'] = encoder.fit_transform(df['BusinessTravel'])
df['Attrition'] = encoder.fit_transform(df['Attrition'])
df
```

	Age	BusinessTravel	MonthlyIncome	JobSatisfaction	Bonus	Department	DistanceFromHome	Education	E
0	41	2	5993	4	17979	Sales	1	2	
1	49	1	5130	2	20520	Research & Development	8	1	
2	37	2	2090	3	6270	Research & Development	2	2	
3	33	1	2909	3	8727	Research & Development	3	4	
4	27	2	3468	2	10404	Research & Development	2	1	
...	...	...	...	...	...	...	...	...	
1465	36	1	2571	4	7713	Research & Development	23	2	
1466	39	2	9991	1	29973	Research & Development	6	1	
1467	27	2	6142	2	24568	Research & Development	4	3	
1468	49	1	5390	2	16170	Sales	2	3	
1469	34	2	4404	3	13212	Research & Development	8	3	

1470 rows × 24 columns

Based on the output, the value of Business Travel and Attrition features have changed into numerical form. For Example:

Business Travel

- 0 – Non-Travel
- 1 - Travel Frequently
- 2 - Travel Rarely

Attrition

- 0 - No
- 1 – Yes

TASK 3

ASSIGN INDEPENDENT AND DEPENDENT VARIABLES

```
# INDEPENDENT VARIABLES
print("INDEPENDENT VARIABLES")
X = df[['Age','BusinessTravel', 'MonthlyIncome', 'JobSatisfaction']].astype(int)
print(X)

# DEPENDENT VARIABLES
print("\nDEPENDENT VARIABLES")
y = df[['Attrition']]
print (y)
```

```
INDEPENDENT VARIABLES
      Age  BusinessTravel  MonthlyIncome  JobSatisfaction
0      41               2           5993                4
1      49               1           5130                2
2      37               2           2090                3
3      33               1           2909                3
4      27               2           3468                2
...   ...             ...            ...              ...
1465   36               1           2571                4
1466   39               2           9991                1
```

```
1467   27              2          6142               2
1468   49              1          5390               2
1469   34              2          4404               3

[1470 rows x 4 columns]

DEPENDENT VARIABLES
      Attrition
0            1
1            0
2            1
3            0
4            0
...        ...
1465         0
1466         0
1467         0
1468         0
1469         0

[1470 rows x 1 columns]
```

The Independent and Dependent variables had been assigned to X and y

TASK 5

SPLIT THE DATASET INTO TRAINING AND TESTING SETS

```
# Training Size 70% and Test Size 30%
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
X_train.shape, y_train.shape
```

```
((1029, 4), (1029, 1))
```

```
X_test.shape, y_test.shape
```

```
((441, 4), (441, 1))
```

Training and Testing Sets have been split into 70% (Train Set) and 30% (Test Set).

TASK 6

DATA NORMALIZATION

```
# Data Normalization
scaler = StandardScaler()
X_Normalized_Test = scaler.fit_transform(X_test)
X_Normalized_Train = scaler.fit_transform(X_train)

X_Normalized_Test
X_Normalized_Train
```

```
array([[ 1.41369115,  0.59277912,  2.28479911,  1.1744894 ],
       [-0.09834647,  0.59277912, -0.31484463, -0.6497364 ],
       [-1.71838678,  0.59277912, -0.78686214, -0.6497364 ],
       ...,
       [-1.61038409, -0.92079337, -0.7505531 ,  1.1744894 ],
       [-0.85436528,  0.59277912, -0.50598877, -0.6497364 ],
       [ 1.41369115,  0.59277912,  2.68795465,  0.2623765 ]])
```

The independent variable has been normalized, the purpose of normalization is to scale the features into similar range, where it will improve the performance and train the stability of the model.

TASK 7

TRAINING THE MODEL USING THE TRAINING SET

```
# ASSIGN THE MODEL
nb = GaussianNB()
nb.fit(X_Normalized_Train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-
  y = column_or_1d(y, warn=True)
▾ GaussianNB
GaussianNB()
```

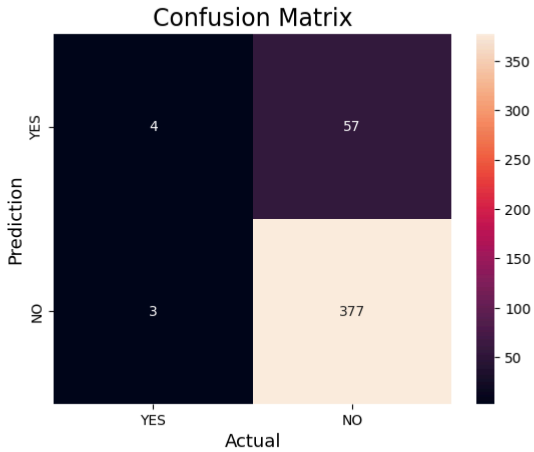
TASK 8

MODEL EVALUATION USING TESTING SET

WITH NORMALIZATION

```
# TEST SET PREDICTION
predict_test = nb.predict(X_Normalized_Test)

# TEST SET ACCURACY SCORE
accuracy_test = accuracy_score(y_test, predict_test)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, predict_test, labels= [1,0])
print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

```
Accuracy on Test Dataset:, 86.39%

Confusion Matrix on Test Dataset
```



The accuracy of the model is 86.39%, moreover the confusion matrix above shown that there are 4 True Positive, 377 True Negative, 3 False Negative (type 1 error), and 57 False Positive (type 2 error). Which the higher the accuracy, true positive, and true negative, the better the prediction is.

TASK 9

ADDITIONAL ANALYSIS

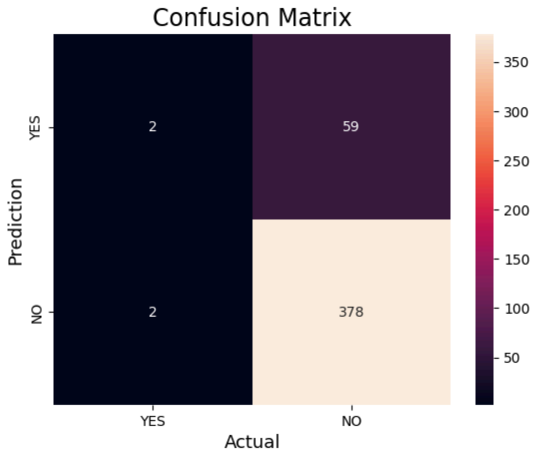
WITHOUT NORMALIZATION

```
# MODEL TRAINING
nb.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-
  y = column_or_1d(y, warn=True)
▾ GaussianNB
GaussianNB()
```

```
# MODEL EVALUATION WITH TEST SET
predict_test = nb.predict(X_test)

# TEST SET ACCURACY PREDICTION
accuracy_test = accuracy_score(y_test, predict_test)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, predict_test, labels= [1,0])

print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

```
Accuracy on Test Dataset:, 86.17%

Confusion Matrix on Test Dataset
```



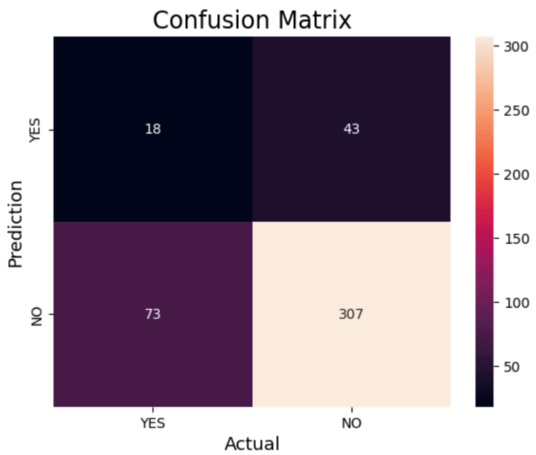
The accuracy of the model is 86.17%, moreover the confusion matrix above shown that there are 2 True Positive, 378 True Negative, 2 False Negative (type 1 error), and 59 False Positive (type 2 error). Which the higher the accuracy, true positive, and true negative, the better the prediction is.

ADDITIONAL ANALYSIS (DECISION TREE CLASSIFIER)

WITH NORMALIZATION

```
print("DECISION TREE CLASS")

# Creating a linear regression model
model = DecisionTreeClassifier()

# Training the model using the training data
model.fit(X_Normalized_Train, y_train)

# Making predictions on the testing data
y_pred = model.predict(X_Normalized_Test)

# TEST SET ACCURACY PREDICTION
accuracy_test = accuracy_score(y_test, y_pred)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, y_pred, labels= [1,0])

print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

```
DECISION TREE CLASS
Accuracy on Test Dataset:, 73.70%

Confusion Matrix on Test Dataset
```



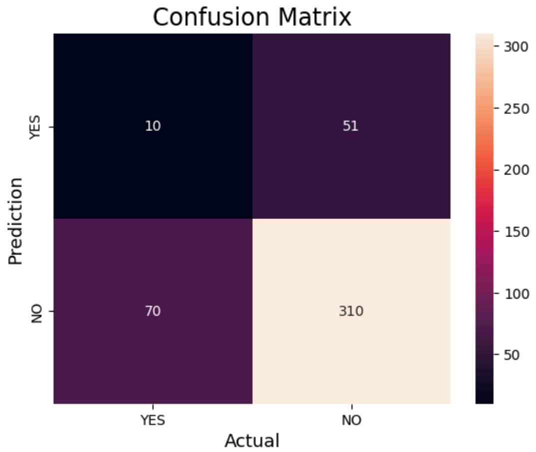
The accuracy of the model is 73.70%, moreover the confusion matrix above shown that there are 18 True Positive, 307 True Negative, 73 False Negative (type 1 error), and 43 False Positive (type 2 error). Which the higher the accuracy, true positive, and true negative, the better the prediction is.

WITHOUT NORMALIZATION

```
print("DECISION TREE CLASS")

# Creating a linear regression model
```

```
# Creating a linear regression model
model = DecisionTreeClassifier()

# Training the model using the training data
model.fit(X_train, y_train)

# Making predictions on the testing data
y_pred = model.predict(X_test)

# TEST SET ACCURACY PREDICTION
accuracy_test = accuracy_score(y_test, y_pred)
print(f'Accuracy on Test Dataset:, {accuracy_test * 100:.2f}% \n')

# TEST SET CONFUSION MATRIX
matrix_test = confusion_matrix(y_test, y_pred, labels= [1,0])

print("Confusion Matrix on Test Dataset")
sns.heatmap(matrix_test,
            annot=True,
            fmt='g',
            xticklabels=['YES','NO'],
            yticklabels=['YES','NO'])
plt.ylabel('Prediction',fontsize=13)
plt.xlabel('Actual',fontsize=13)
plt.title('Confusion Matrix',fontsize=17)
plt.show()
```

```
DECISION TREE CLASS
Accuracy on Test Dataset:, 72.56%

Confusion Matrix on Test Dataset
```



The accuracy of the model is 72.56%, moreover the confusion matrix above shown that there are 10 True Positive, 310 True Negative, 70 False Negative (type 1 error), and 51 False Positive (type 2 error). Which the higher the accuracy, true positive, and true negative, the better the prediction is.

Conclusion

In conclusion, the process of assigned independent and dependent variables run well, as well as the data splitting and normalization. This is because based on our findings, the normalization process in the machine learning model resulted in improved in accuracy by 0.22%, which the total diffrence is not that much. Moreoever, the confusion matrix show positive results whereas the true positive and negative are higher than the false ones, and the accuracy of the model is 86.39%, which it is considered as optimal accurcy results.

Apart form Naïve Bayes, an additional machine learning model (Decision Tree Classifier) were carried out as well. The results of the Decision Tree Classiffier model have a slight improvement when the dataset was normalized, whereas the Accuracy of normalized dataset improved by 1.14% when compared to the one without normalization.