```
# 1.    Load the basic libraries and packages

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
```

```
# 2.    Load the dataset

data = pd.read_csv('/content/naivebayes.csv')
data
```

	Age	Salary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
# 3.    Analyse the dataset

data.describe()
```

	Age	Salary	Purchased
count	400.000000	400.000000	400.000000
mean	37.655000	69742.500000	0.357500
std	10.482877	34096.960282	0.479864
min	18.000000	15000.000000	0.000000
25%	29.750000	43000.000000	0.000000
50%	37.000000	70000.000000	0.000000
75%	46.000000	88000.000000	1.000000
max	60.000000	150000.000000	1.000000

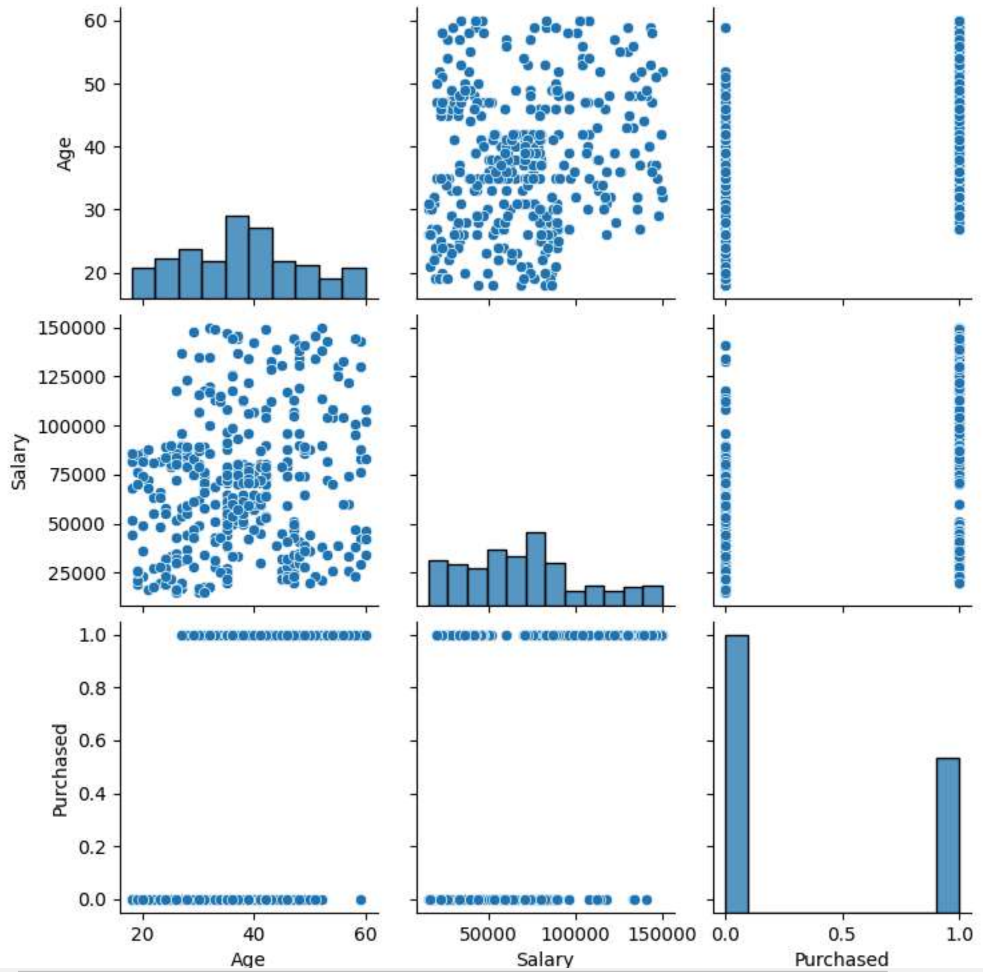
```
# 4.    Normalize the data

def Feature_Normalization(X):
  X = (X - np.mean(X)) / np.std(X)  # Calculate mean and std across the entire 1D array
  return X
```

```
# 5.  Pre-process the data

x = data.iloc[: , :-1].values
y = data.iloc[: , -1].values

# Initialize x_norm as a list to store normalized features
```

```
# Initialize x_norm as a list to store normalized features
x_norm = []

for i in range(x.shape[1]):  # Iterate through columns of x
    norm_feature = Feature_Normalization(x[:, i])
    x_norm.append(norm_feature)  # Append normalized feature to the list

# Convert the list of normalized features to a NumPy array
x_norm = np.array(x_norm).T  # Transpose to get the desired shape
x_norm
```

```
       [ 0.03295203, -0.13926283],
       [ 0.89257019, -0.55037082],
       [ 0.89257019,  1.03533143],
       [ 0.31949142, -0.19799255],
       [ 1.46564897,  0.06629116],
       [ 1.5611621 ,  1.123426  ],
       [ 0.12846516,  0.21311545],
       [ 0.03295203, -0.25672226],
       [ 0.03295203,  1.27025028],
       [-0.0625611 ,  0.15438573],
       [ 0.41500455,  0.59485858],
       [-0.0625611 , -0.37418169],
       [-0.15807423,  0.85914229],
       [ 2.13424088, -1.04957339],
       [ 1.5611621 ,  0.00756145],
       [ 0.31949142,  0.06629116],
       [ 0.22397829,  0.03692631],
       [ 0.41500455, -0.46227625],
       [ 0.51051768,  1.74008799],
       [ 1.46564897, -1.04957339],
       [ 0.89257019, -0.57973568],
       [ 0.41500455,  0.27184516],
       [ 0.41500455,  1.00596657],
       [ 2.03872775, -1.19639767],
       [ 1.94321462, -0.66783025],
       [ 0.79705706,  0.53612887],
       [ 0.03295203,  0.03692631],
       [ 1.5611621 , -1.28449224],
       [ 2.13424088, -0.69719511],
       [ 2.13424088,  0.38930459],
       [ 0.12846516,  0.09565602],
       [ 2.03872775,  1.76945285],
       [-0.0625611 ,  0.30121002],
       [ 0.79705706, -1.1083031 ],
       [ 0.79705706,  0.12502088],
       [ 0.41500455, -0.49164111],
       [ 0.31949142,  0.50676401],
       [ 1.94321462, -1.37258681],
       [ 0.41500455, -0.16862769],
       [ 0.98808332, -1.07893824],
       [ 0.60603081,  2.03373655],
       [ 1.08359645, -1.22576253],
       [ 1.84770149, -1.07893824],
       [ 1.75218836, -0.28608712],
       [ 1.08359645, -0.9027491 ],
       [ 0.12846516,  0.03692631],
       [ 0.89257019, -1.04957339],
       [ 0.98808332, -1.02020853],
       [ 0.98808332, -1.07893824],
       [ 0.89257019, -1.37258681],
       [ 0.70154394, -0.72655996],
       [ 2.13424088, -0.81465453],
       [ 0.12846516, -0.31545197],
       [ 0.79705706, -0.84401939],
       [ 1.27462271, -1.37258681],
       [ 1.17910958, -1.46068138],
       [-0.15807423, -1.07893824],
       [ 1.08359645, -0.99084367]])
```

```
# 6.    Visualize the Data

sns.pairplot(data)
plt.show()
```




```
# 7.    Separate the training and testing data

x_train , x_test , y_train , y_test = train_test_split(x_norm , y , test_size = 0.2 , random_state = 42)
```

```
# 8.    Apply the Bernoulli Naïve Bayes algorithm

model = BernoulliNB()
model.fit(x_train , y_train)
```

```
BernoulliNB
BernoulliNB()
```

```
# 9.  Predict the testing dataset

y_pred_Bernoulli = model.predict(x_test)
```

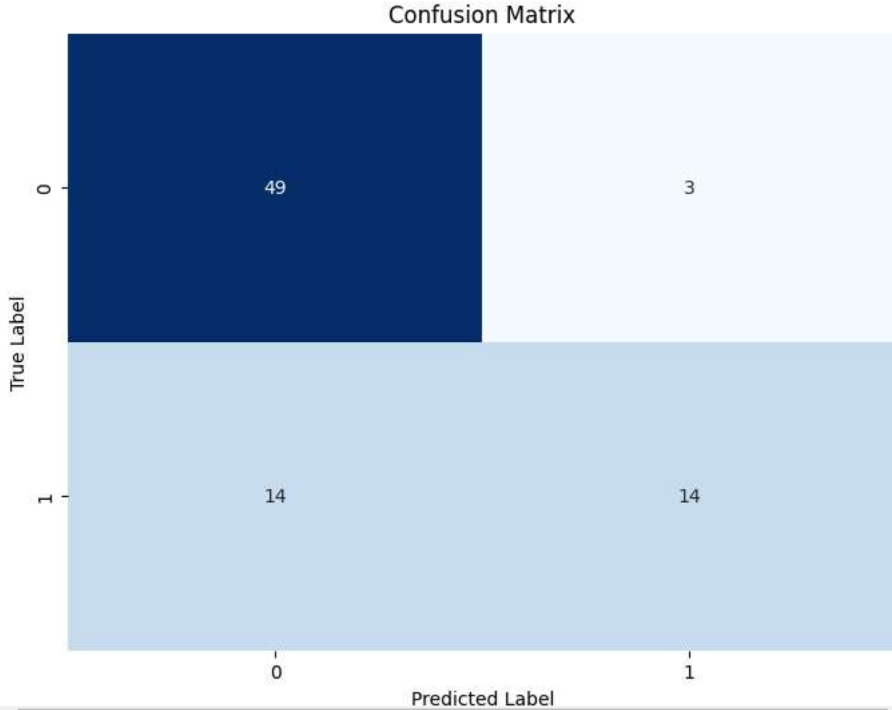
```
# 10.   Obtain the confusion matrix

cm = confusion_matrix(y_test , y_pred_Bernoulli)
cm
```

```
array([[50,  2],
       [ 3, 25]])
```

```
# 11.   Obtain the accuracy score

accuracy_score(y_test , y_pred_Bernoulli)
```

```
0.9375
```

```
# 12.   Visualize the classified dataset

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```




```
# 13.   Apply the Gaussian Naïve Bayes algorithm

model = GaussianNB()
model.fit(x_train , y_train)
```

```
GaussianNB
GaussianNB()
```

```
# 14. Predict the testing dataset

y_pred_GaussianNB = model.predict(x_test)
```

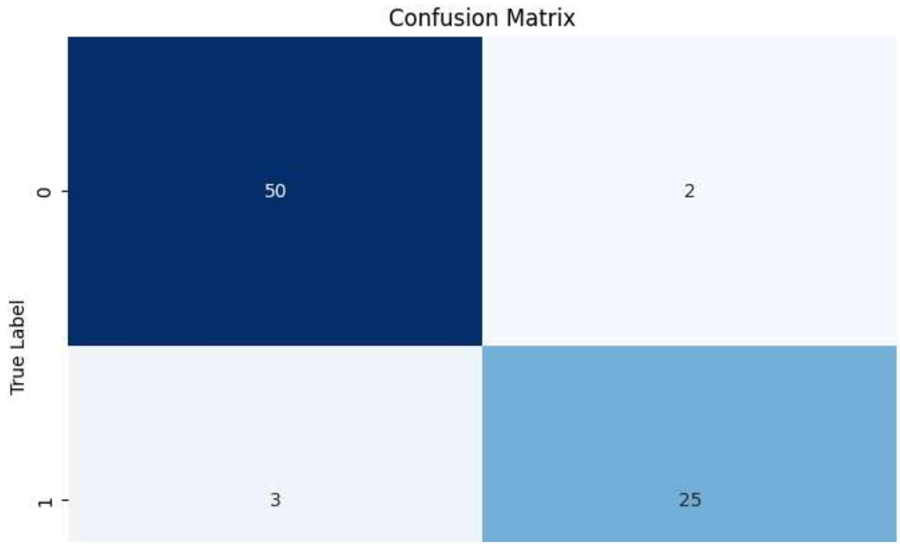
```
# 15.   Obtain the confusion matrix

cm = confusion_matrix(y_test , y_pred_GaussianNB)
cm
```

```
array([[50,  2],
       [ 3, 25]])
```

```
# 16.   Obtain the accuracy score

accuracy_score(y_test , y_pred_GaussianNB)
```

```
0.9375
```

```
# 17.   Visualize the classified dataset

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```




```
# a.  Classified dataset using Bernoulli Naïve Bayes

y_pred_Bernoulli
```

```
array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0])
```

```
# b.  Classified dataset using Gaussian Naïve Bayes

y_pred_GaussianNB
```

```
array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0])
```