- Name: Bhavesh Waghela
- Student Number: N01639685

In [1]:
```
import pandas as pd

data = pd.read_csv('NBC_DATA.csv')

data
```

Out[1]:

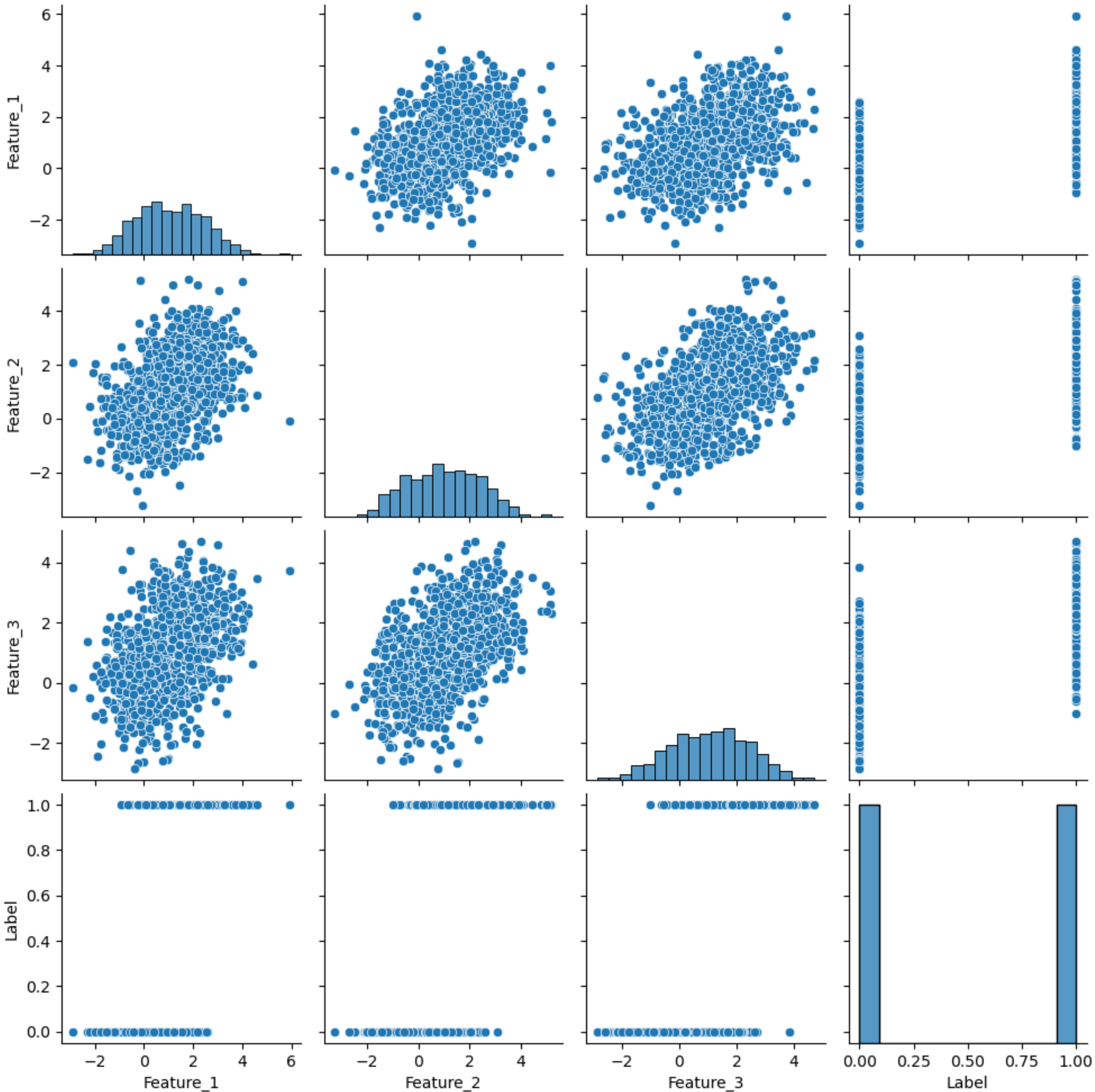
	Feature_1	Feature_2	Feature_3	Label
0	0.4967	-0.1383	0.6477	0
1	1.5230	-0.2342	-0.2341	0
2	1.5792	0.7674	-0.4695	0
3	0.5426	-0.4634	-0.4657	0
4	0.2420	-1.9133	-1.7249	0
...	...	...	...	...
995	1.0400	1.8773	2.0934	1
996	0.8698	4.4117	3.5164	1
997	2.6021	2.0720	1.7878	1
998	1.0481	2.0775	2.2578	1
999	0.7582	2.3342	1.8447	1

1000 rows × 4 columns

In [2]:
```
import seaborn as sns

sns.pairplot(data)
```

Out[2]: <seaborn.axisgrid.PairGrid at 0x1f2ae388370>



Q. What is Gaussian NBC?

- Gaussian NBC is a variant of the Naive Bayes algorithm, which assumes that the features are conditionally independent given the class label.

Q. Are there any additional assumptions that need to be met for this type of NBCs? If so what are they?

- It be continuous and normally distributed within each class. It assumes that the features are conditionally independent given the class labe.

Q. We have covered histograms before. What are they and what can they provide informationon?

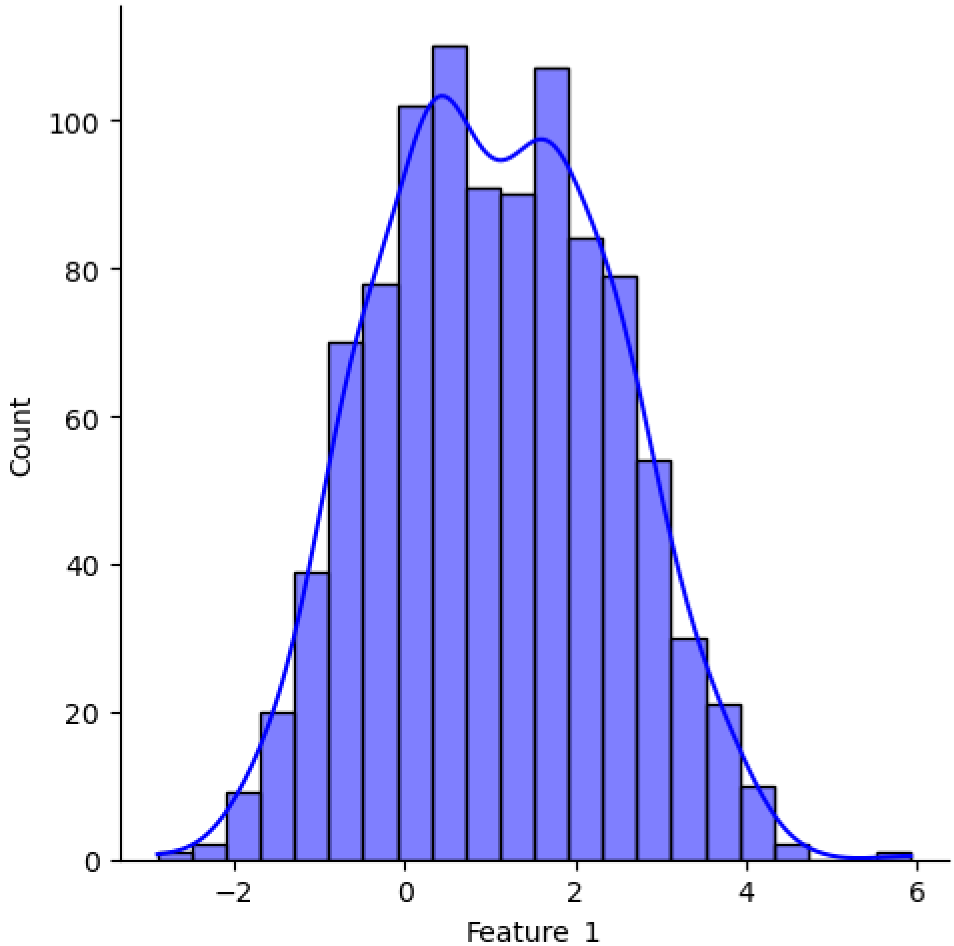
- Histograms are a common chart type used to look at distributions of numeric variables

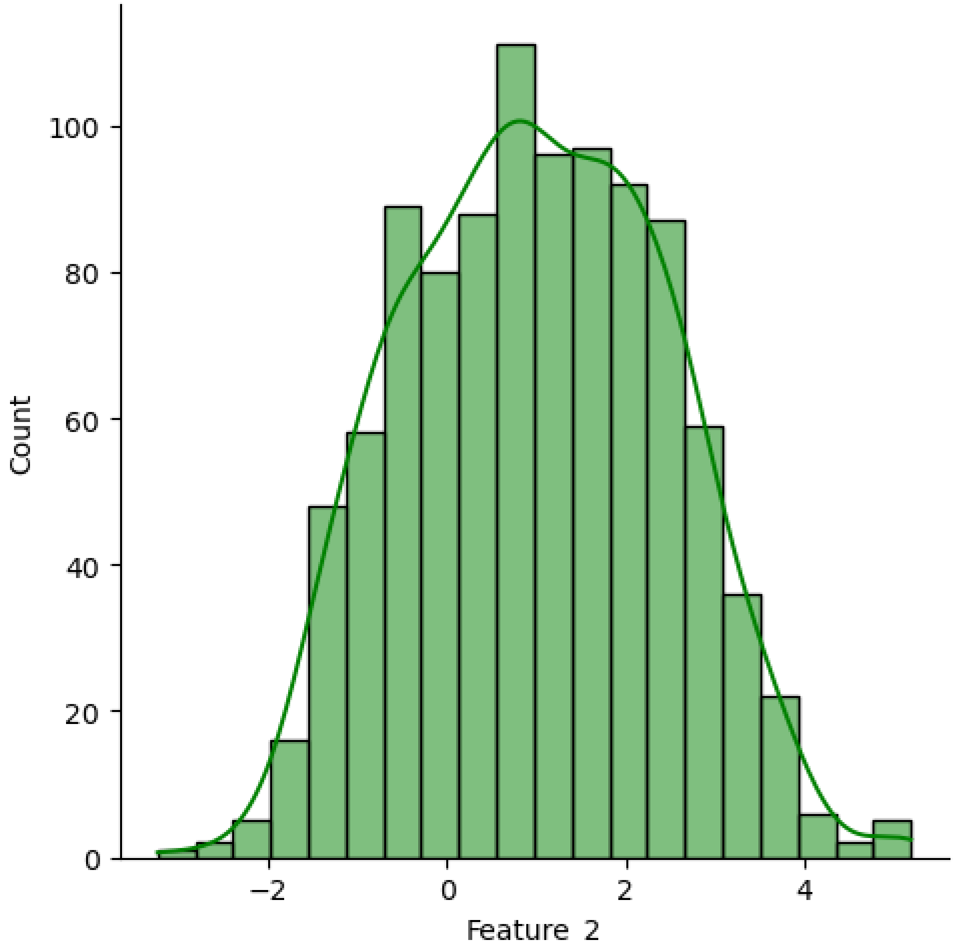
Q. Research KDE (Kernel Density Estimation) plots and understand what information they provide?

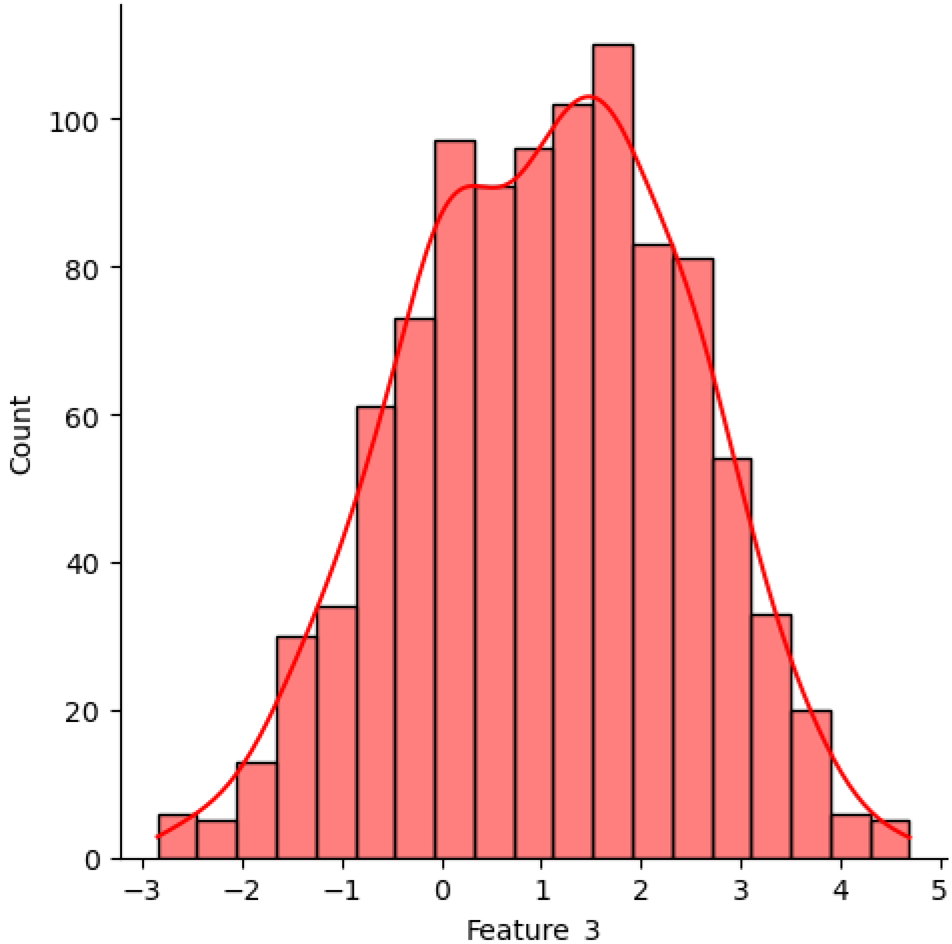
- A kernel density estimate (KDE) plot, which is similar to a histogram, method for visualizing the distribution of observations in a dataset.
- It is a statistical technique used for estimating the probability density function of a random variable.
- It helps in visualizing the shape, spread, and modes of the data distribution.

In [28]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.displot(data['Feature_1'], kde=True, color='blue')
sns.displot(data['Feature_2'], kde=True, color='green')
sns.displot(data['Feature_3'], kde=True, color='red')
plt.show()
```




Count
0
20
40
60
80
100
−2
0
2
4
Feature_2



Q. Verify that your features are normally distributed or use only features that are normally distributed to design and test a Gaussian NBC

- All the features in the dataset are Normally Distributed and hence Gaussuin Destribution can be performed on the dataset.

In [15]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X = data.drop('Label', axis=1)
y = data['Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando

gaus_nb = GaussianNB()

gaus_nb.fit(X_train, y_train)
```

Out[15]:
```
GaussianNB()
```

In [16]:
```
y_pred = gaus_nb.predict(X_test)

y_pred
```

Out[16]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0], dtype=int64)
```

Q. Compute the confusion matrix and accuracy on the test data.

In [29]:
```
from sklearn.metrics import accuracy_score, confusion_matrix

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Model accuracy score: 0.9733
Confusion matrix

 [[146   4]
 [  4 146]]

True Positives(TP) =  146

True Negatives(TN) =  146

False Positives(FP) =  4

False Negatives(FN) =  4
```

From the above confusion matrix it can be infered that the model is performing really well as we only have 4 False Negatives predections and 146 of True Positives predections, also the Model accuracy is 97.33%.