2.Naive Bayes

```
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, roc_auc_score, roc_curve, accuracy_score, log_loss
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

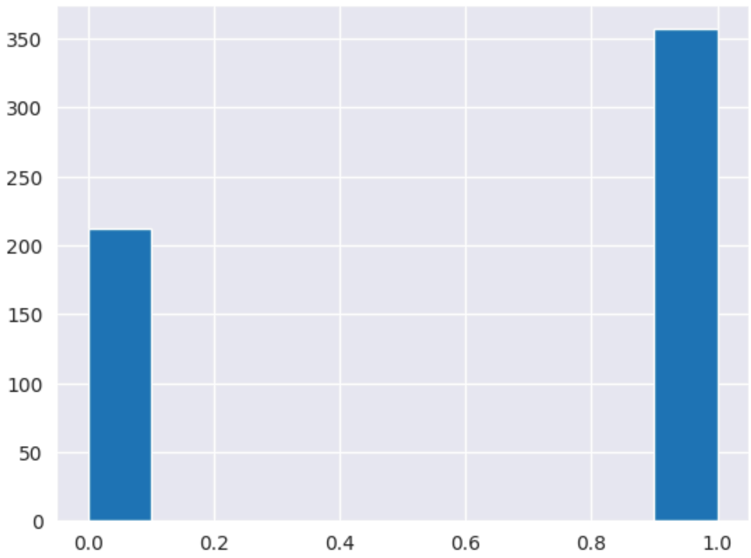
```
data = pd.read_csv("/content/BreastCancerData.csv")
data.head(10)
```

	mean_radius	mean_texture	mean_perimeter	mean_area	mean_smoothness	diagnosis
0	17.99	10.38	122.80	1001.0	0.11840	0
1	20.57	17.77	132.90	1326.0	0.08474	0
2	19.69	21.25	130.00	1203.0	0.10960	0
3	11.42	20.38	77.58	386.1	0.14250	0
4	20.29	14.34	135.10	1297.0	0.10030	0
5	12.45	15.70	82.57	477.1	0.12780	0
6	18.25	19.98	119.60	1040.0	0.09463	0
7	13.71	20.83	90.20	577.9	0.11890	0
8	13.00	21.82	87.50	519.8	0.12730	0
9	12.46	24.04	83.97	475.9	0.11860	0

Next steps: Generate code with data View recommended plots

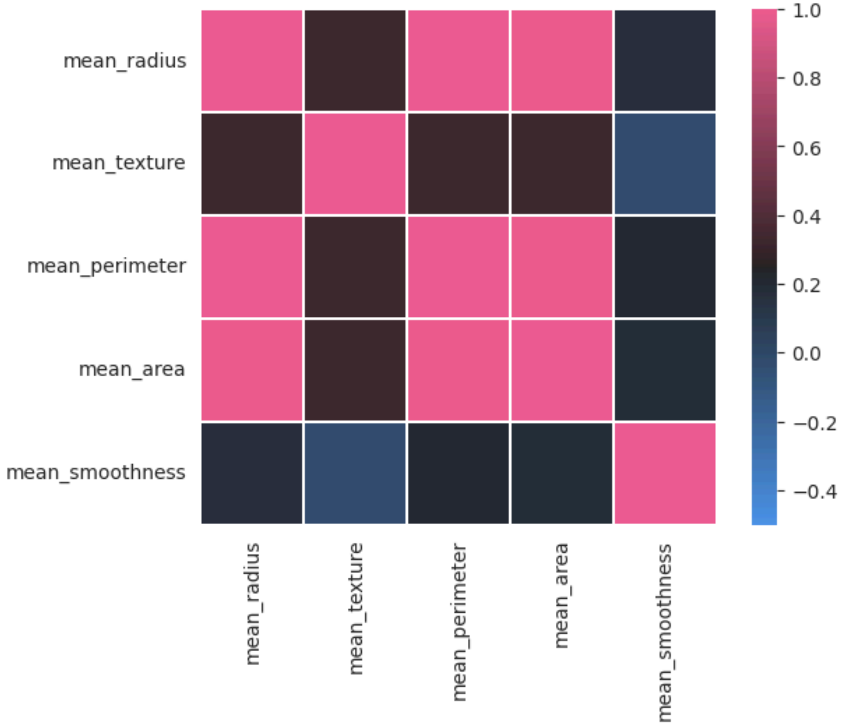
```
data["diagnosis"].hist()
```

<Axes: >




```
corr = data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250,354,80,60,center='dark',as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
```

```
<Axes: >
```



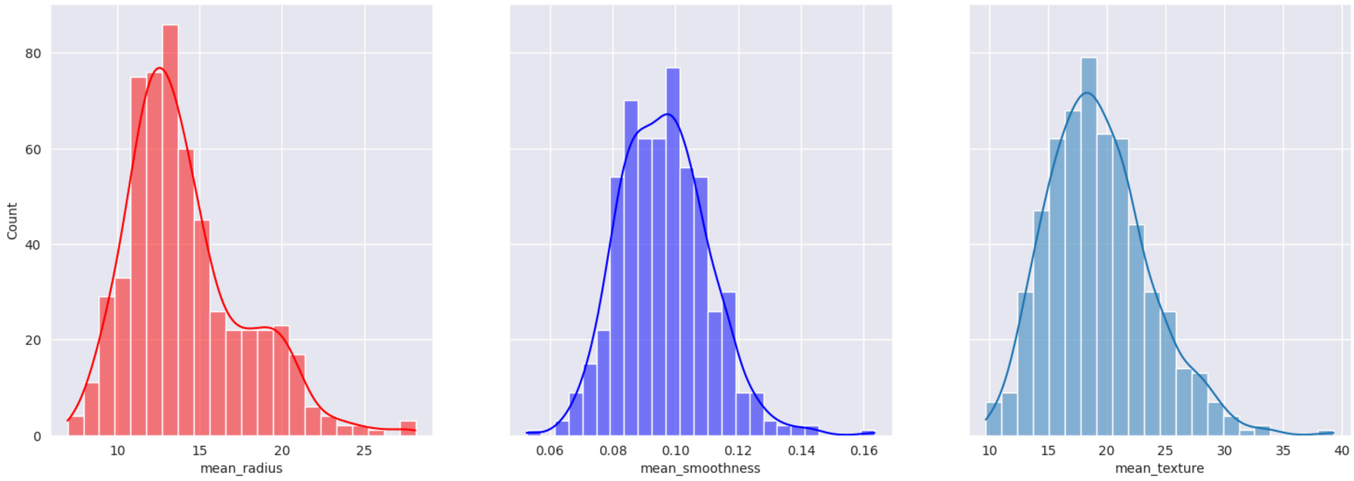

```
data = data[["mean_radius", "mean_texture", "mean_smoothness", "diagnosis"]]
data.head(10)
```

	mean_radius	mean_texture	mean_smoothness	diagnosis
0	17.99	10.38	0.11840	0
1	20.57	17.77	0.08474	0
2	19.69	21.25	0.10960	0
3	11.42	20.38	0.14250	0
4	20.29	14.34	0.10030	0
5	12.45	15.70	0.12780	0
6	18.25	19.98	0.09463	0
7	13.71	20.83	0.11890	0
8	13.00	21.82	0.12730	0
9	12.46	24.04	0.11860	0

Next steps: Generate code with data View recommended plots

```
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(data, ax=axes[0], x="mean_radius", kde=True, color='r')
sns.histplot(data, ax=axes[1], x="mean_smoothness", kde=True, color='b')
sns.histplot(data, ax=axes[2], x="mean_texture", kde=True)
```

```
<Axes: xlabel='mean_texture', ylabel='Count'>
```




```
def calculate_prior(df, Y):
    classes = sorted(list(df[Y].unique()))
    prior = []
    for i in classes:
        prior.append(len(df[df[Y]==i])/len(df))
    return prior
```

```
def calculate_likelihood_gaussian(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    mean, std = df[feat_name].mean(), df[feat_name].std()
    p_x_given_y = (1 / (np.sqrt(2 * np.pi) * std)) *  np.exp(-((feat_val-mean)**2 / (2 * std**2 )))
    return p_x_given_y
```

```
def naive_bayes_gaussian(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_gaussian(df, features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

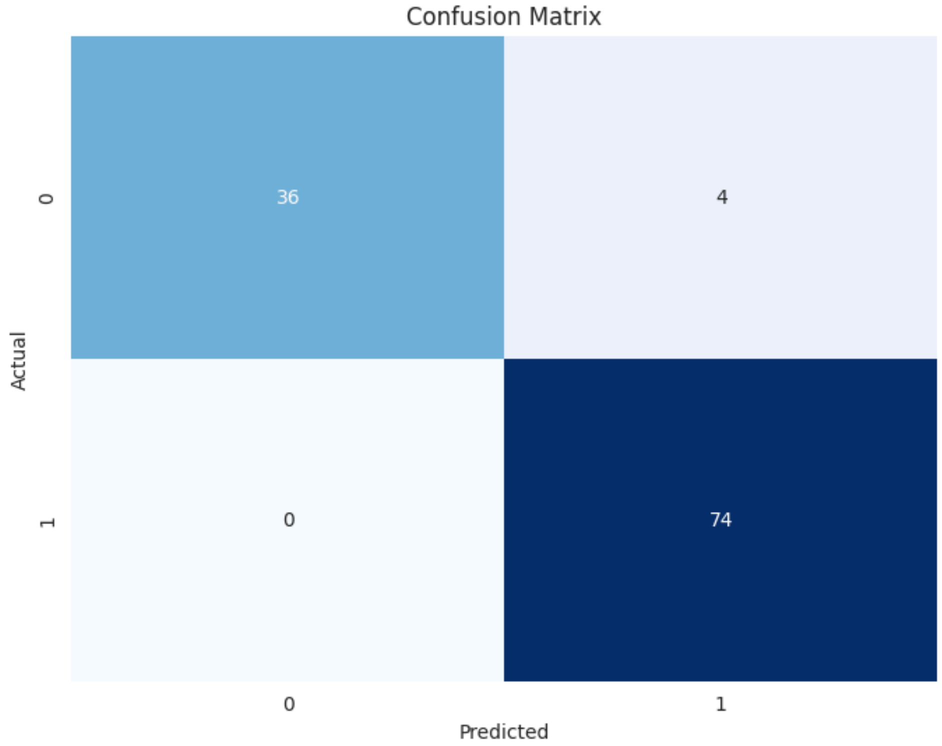
e.2 What are: Accuracy, Confusion Matrix, Precision, Recall & F1 Score, ROC & AUC. Log Loss?

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_gaussian(train, X=X_test, Y="diagnosis")

from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
print("precision_score", precision_score(Y_test, Y_pred))
print("confusion_matrix", confusion_matrix(Y_test, Y_pred))
print("f1_score", f1_score(Y_test, Y_pred))
print("recall_score", recall_score(Y_test, Y_pred))
```

```
precision_score 0.9487179487179487
confusion_matrix [[36  4]
 [ 0 74]]
f1_score 0.9736842105263158
recall_score 1.0
```

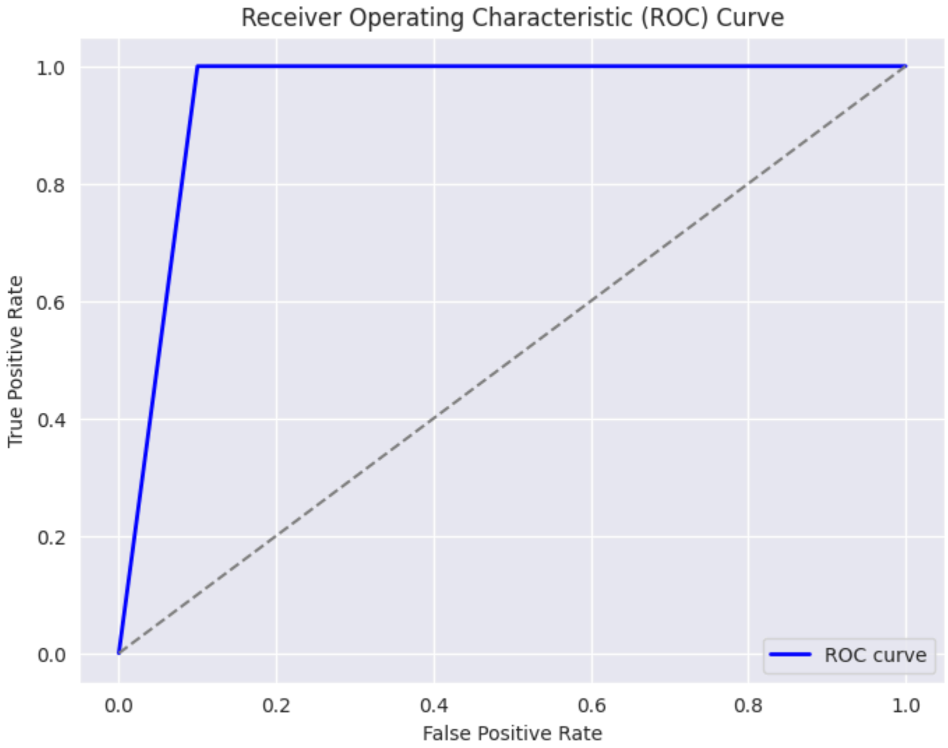
```
# Calculate confusion matrix
cm = confusion_matrix(Y_test, Y_pred)
print(cm)
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

```
[[36  4]
 [ 0 74]]
```



```
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, log_loss

# Accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)

# ROC curve
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# AUC (Area Under the Curve)
auc = roc_auc_score(Y_test, Y_pred)
print("AUC:", auc)

# Log Loss
logloss = log_loss(Y_test, Y_pred)
print("Log Loss:", logloss)
```

```
Accuracy: 0.9649122807017544
```




```
AUC: 0.9500000000000001
Log Loss: 1.2646895926006019
```

```
data["cat_mean_radius"] = pd.cut(data["mean_radius"].values, bins = 3, labels = [0,1,2])
data["cat_mean_texture"] = pd.cut(data["mean_texture"].values, bins = 3, labels = [0,1,2])
data["cat_mean_smoothness"] = pd.cut(data["mean_smoothness"].values, bins = 3, labels = [0,1,2])

data = data.drop(columns=["mean_radius", "mean_texture", "mean_smoothness"])
data = data[["cat_mean_radius", "cat_mean_texture", "cat_mean_smoothness", "diagnosis"]]
data.head(10)
```

	cat_mean_radius	cat_mean_texture	cat_mean_smoothness	diagnosis
0	1	0	1	0
1	1	0	0	0
2	1	1	1	0
3	0	1	2	0
4	1	0	1	0
5	0	0	2	0
6	1	1	1	0
7	0	1	1	0
8	0	1	2	0
9	0	1	1	0

Next steps: Generate code with `data` View recommended plots

```
def calculate_likelihood_categorical(df, feat_name, feat_val, Y, label):
    feat = list(df.columns)
    df = df[df[Y]==label]
    p_x_given_y = len(df[df[feat_name]==feat_val]) / len(df)
    return p_x_given_y


def naive_bayes_categorical(df, X, Y):
    # get feature names
    features = list(df.columns)[:-1]

    # calculate prior
    prior = calculate_prior(df, Y)

    Y_pred = []
    # loop over every data sample
    for x in X:
        # calculate likelihood
        labels = sorted(list(df[Y].unique()))
        likelihood = [1]*len(labels)
        for j in range(len(labels)):
            for i in range(len(features)):
                likelihood[j] *= calculate_likelihood_categorical(df, features[i], x[i], Y, labels[j])

        # calculate posterior probability (numerator only)
        post_prob = [1]*len(labels)
        for j in range(len(labels)):
            post_prob[j] = likelihood[j] * prior[j]

        Y_pred.append(np.argmax(post_prob))

    return np.array(Y_pred)
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(data, test_size=.2, random_state=41)

X_test = test.iloc[:,:-1].values
Y_test = test.iloc[:,-1].values
Y_pred = naive_bayes_categorical(train, X=X_test, Y="diagnosis")

from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score
print("precision_score", precision_score(Y_test, Y_pred))
print("confusion_matrix", confusion_matrix(Y_test, Y_pred))
print("f1_score", f1_score(Y_test, Y_pred))
print("recall_score", recall_score(Y_test, Y_pred))
```

```
precision_score 0.971830985915493
confusion_matrix [[38  2]
 [ 5 69]]
f1_score 0.9517241379310345
recall_score 0.9324324324324325
```

```
from sklearn.metrics import roc_curve, roc_auc_score, accuracy_score, log_loss

# Accuracy
accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)

# ROC curve
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

# AUC (Area Under the Curve)
auc = roc_auc_score(Y_test, Y_pred)
print("AUC:", auc)

# Log Loss
logloss = log_loss(Y_test, Y_pred)
print("Log Loss:", logloss)
```

```
Accuracy: 0.9385964912280702
```

Receiver Operating Characteristic (ROC) Curve