```
#Import the library
import pandas as pd
```

1.Choose a binary classification dataset

```
#load the dataset
df=pd.read_csv('/content/data.csv')
```

```
#check the first 5 rows
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_me
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.277
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.078
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.159
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.283
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.132

5 rows × 33 columns

```
#check the row & column no.
df.shape
```

```
(569, 33)
```

```
#check the information of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   id                       569 non-null    int64  
 1   diagnosis                569 non-null    object 
 2   radius_mean              569 non-null    float64
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
 32  Unnamed: 32              0 non-null      float64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

```
#drop unnecessary columns
df=df.drop(columns=['id','Unnamed: 32'])
```

```
#convert diagnosis to binary (M=1, B=0)
df['diagnosis']=df['diagnosis'].map({'M': 1, 'B': 0})
```

2.Train/test split and standardize features.

```
#split features and target
X = df.drop('diagnosis', axis=1)
y = df['diagnosis']
```

```
from sklearn.model_selection import train_test_split
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
```

```
# Standardize features
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

```
# Output the shapes
X_train_scaled.shape, X_test_scaled.shape, y_train.shape, y_test.shape
```

```
((455, 30), (114, 30), (455,), (114,))
```

3.Fit a Logistic Regression model

```
from sklearn.linear_model import LogisticRegression
```

```
# Create the logistic regression model instance
lr= LogisticRegression(random_state=42, max_iter=10000)
```

```
# Fit the model on training data
lr.fit(X_train_scaled, y_train)
```

```
▾               LogisticRegression          i ?
LogisticRegression(max_iter=10000, random_state=42)
```

```
# Predict on the test set
y_pred = lr.predict(X_test_scaled)
```

4.Evaluate with confusion matrix, precision, reca l, ROC-AUC.

```
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, RocCurveDisplay
```

```
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

```
print("Confusion Matrix:\n",cm)
```

```
Confusion Matrix:
 [[70  1]
 [ 2 41]]
```

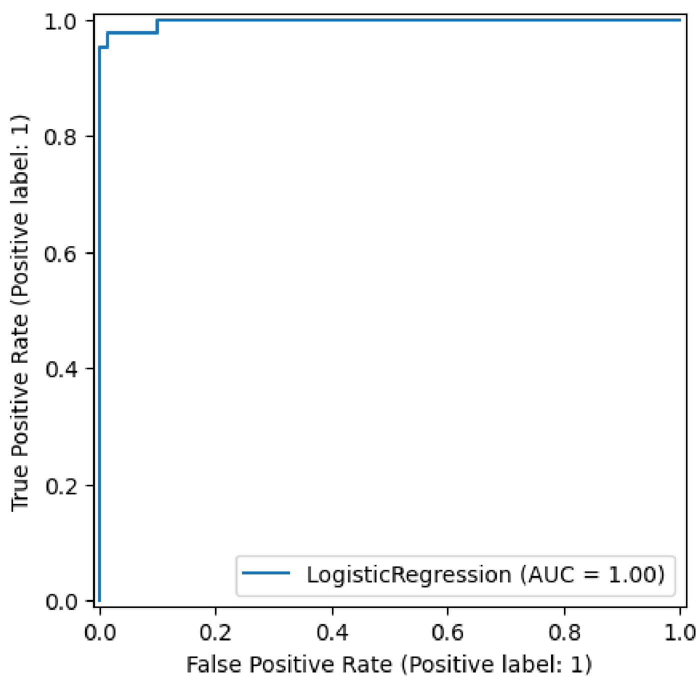
```
# Generate classification report for precision, recall, f1-score
cr= classification_report(y_test, y_pred, output_dict=True)
```

```
print("Classification Report:\n",cr)
```

```
Classification Report:
 {'0': {'precision': 0.9722222222222222, 'recall': 0.9859154929577465, 'f1-score': 0.9790209790209791, 'supp
```

```
# Compute ROC-AUC score
y_prob = lr.predict_proba(X_test_scaled)[:, 1]
roc_auc = roc_auc_score(y_test, y_prob)
```

```
# Display ROC curve
roc_display = RocCurveDisplay.from_estimator(lr, X_test_scaled, y_test)
```



```
ra=roc_auc
print("ROC-AUC Score:", ra)
```

```
ROC-AUC Score: 0.99737962659679
```

5.Tune threshold and explain sigmoid function

```
import numpy as np
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
```

```
# Get predicted probabilities for the positive class
y_prob = lr.predict_proba(X_test_scaled)[:, 1]
```

```
# Define thresholds to test
thresholds = [0.4, 0.5, 0.6]
results = []

# Evaluate performance at each threshold
for thresh in thresholds:
    y_pred_thresh = (y_prob >= thresh).astype(int)
    precision = precision_score(y_test, y_pred_thresh)
    recall = recall_score(y_test, y_pred_thresh)
    f1 = f1_score(y_test, y_pred_thresh)
    results.append((thresh, precision, recall, f1))

# Convert to DataFrame for display
results_df = pd.DataFrame(results, columns=["Threshold", "Precision", "Recall", "F1 Score"])
results_df
```

```
   Threshold  Precision    Recall  F1 Score
0        0.4   0.976744  0.976744  0.976744
1        0.5   0.976190  0.953488  0.964706
2        0.6   1.000000  0.953488  0.976190
```

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

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```