```
import os, math
import numpy as np, pandas as pd
import matplotlib.pyplot as plt, seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (confusion_matrix, classification_report,
                             accuracy_score, precision_score, recall_score, f1_score,
                             roc_curve, roc_auc_score, mean_squared_error)

sns.set()
```

```
fname = "samples_cancer.csv"
if not os.path.exists(fname):
    try:
        from google.colab import files
        print("Upload 'samples_cancer.csv' now (choose file from your PC).")
        uploaded = files.upload()
        if uploaded:
            uploaded_name = list(uploaded.keys())[0]
            if uploaded_name != fname:
                os.rename(uploaded_name, fname)
    except Exception:
        raise FileNotFoundError(f"'{fname}' not found. Upload it to Colab Files.")

df = pd.read_csv(fname)
print("Loaded:", df.shape)
display(df.head())
```

```
Loaded: (699, 11)
```

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2

```
candidates = [c for c in df.columns if c.lower() in ('diagnosis','target','label','class','outcome','status')]
if not candidates:
    candidates = [c for c in df.columns if any(k in c.lower() for k in ('diagnos','benign','malign','class','target'))]
if not candidates:
    raise ValueError("Could not auto-detect target. Set target = '<your_target_column>' manually in the code.")
target = candidates[0]
print("Using target column:", target)
```

```
Using target column: Class
```

```
df = df.dropna(subset=[target]).copy()
```

```
y = df[target]
if y.dtype == object or y.dtype.name == 'category':
    y = y.astype(str).str.lower()
    # common mappings
    y = y.replace({'benign':'0','b':'0','malignant':'1','m':'1','pos':'1','neg':'0'})
    uniq = pd.Series(y).unique().tolist()
    if set(uniq) <= set(['0','1']):
        y = y.astype(int)
    else:
        # fallback: map first->0 second->1
        vals = uniq[:2]
        y = y.map({vals[0]:0, vals[1]:1}).astype(int)
else:
    # if numeric but not 0/1, binarize by median if necessary
    if sorted(y.unique()) not in ([0,1],):
        y = (y != y.min()).astype(int)
```

```
X = df.drop(columns=[target])
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()

if len(cat_cols) > 0:
    X = pd.get_dummies(X, columns=cat_cols, drop_first=True)
else:
```

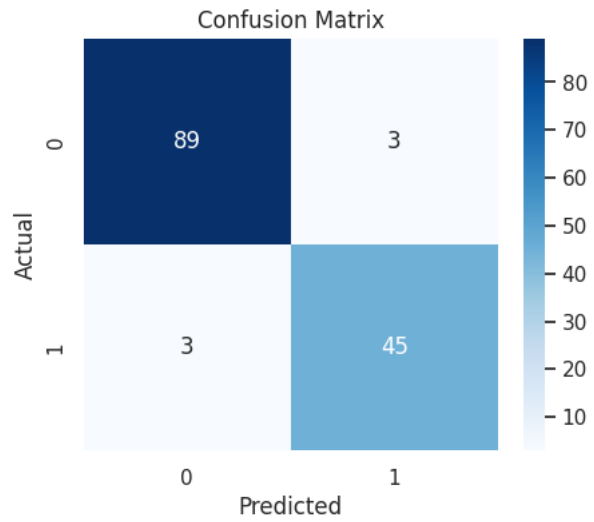
```
    X = X[num_cols]

# fill NA and ensure numeric
X = X.fillna(X.median())

print("Feature count:", X.shape[1], "Positive rate:", y.mean().round(3))

# ---- split & scale ----
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
scaler = StandardScaler()
X_train_s = scaler.fit_transform(X_train)
X_test_s  = scaler.transform(X_test)
```

```
Feature count: 19 Positive rate: 0.345
```

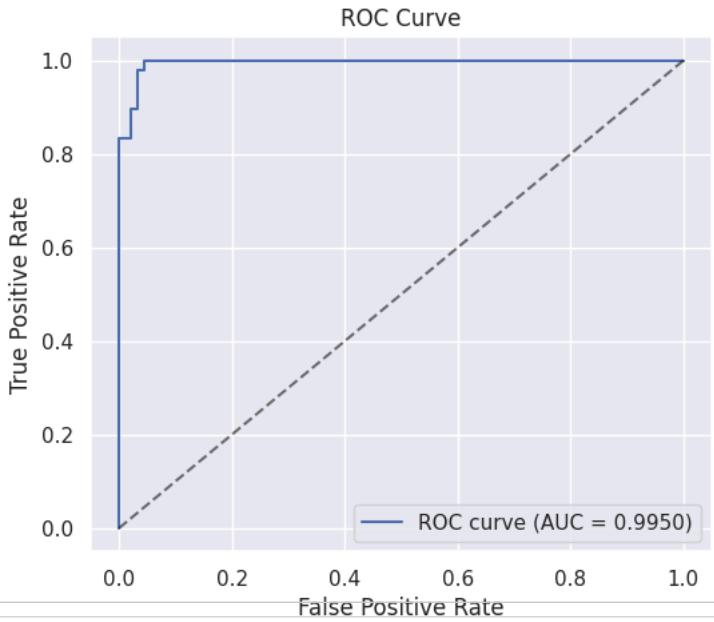
```
model = LogisticRegression(max_iter=10000, solver='liblinear')
model.fit(X_train_s, y_train)
y_pred = model.predict(X_test_s)
y_proba = model.predict_proba(X_test_s)[:,1]
```

```
acc  = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, zero_division=0)
rec  = recall_score(y_test, y_pred, zero_division=0)
f1   = f1_score(y_test, y_pred, zero_division=0)
print(f"\nAccuracy: {acc:.4f}  Precision: {prec:.4f}  Recall: {rec:.4f}  F1: {f1:.4f}\n")
print("Classification report:\n", classification_report(y_test, y_pred, digits=4))

# ---- confusion matrix ----
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted'); plt.ylabel('Actual'); plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.9571  Precision: 0.9375  Recall: 0.9375  F1: 0.9375

Classification report:
              precision    recall  f1-score   support

           0     0.9674    0.9674    0.9674        92
           1     0.9375    0.9375    0.9375        48

    accuracy                         0.9571       140
   macro avg     0.9524    0.9524    0.9524       140
weighted avg     0.9571    0.9571    0.9571       140
```




```
fpr, tpr, thr = roc_curve(y_test, y_proba)
auc = roc_auc_score(y_test, y_proba)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc:.4f})')
plt.plot([0,1],[0,1],'k--', alpha=0.6)
plt.xlabel('False Positive Rate'); plt.ylabel('True Positive Rate'); plt.title('ROC Curve')
plt.legend(); plt.grid(True); plt.show()
```




```
plt.figure(figsize=(6,4))
plt.scatter(y_test, y_proba, alpha=0.7)
plt.xlabel('Actual (0/1)'); plt.ylabel('Predicted probability (positive)')
plt.title('Actual vs Predicted probability'); plt.grid(True); plt.show()
```