

```
df = pd.read_csv('train.csv')
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S

```
# 1. Imports
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score

# 2. Load dataset
df = pd.read_csv("train.csv")

# 3. Target column
target_col = "Survived"

# 4. Identify feature columns
numeric_features = df.drop(columns=[target_col]).select_dtypes(include=["int64", "float64"]).columns
categorical_features = df.drop(columns=[target_col]).select_dtypes(include=["object"]).columns

# 5. Preprocessing
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="mean"))
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numeric_features),
    ("cat", categorical_transformer, categorical_features)
])

# 6. Train & evaluate function
def train_and_evaluate(test_size):
    train_df, test_df = train_test_split(
        df, test_size=test_size, random_state=42
    )

    model = Pipeline([
        ("preprocessor", preprocessor),
        ("classifier", LogisticRegression(max_iter=1000))
    ])

    model.fit(
        train_df.drop(columns=[target_col]),
        train_df[target_col]
    )

    predictions = model.predict(
        test_df.drop(columns=[target_col])
    )

    print(f"\nTrain/Test Split: {int((1-test_size)*100)}/{int(test_size*100)}")
    print("Accuracy:", accuracy_score(test_df[target_col], predictions))
    print("Confusion Matrix:\n", confusion_matrix(test_df[target_col], predictions))
    print("Precision:", precision_score(test_df[target_col], predictions))
```

```
print("Recall:", recall_score(test_df[target_col], predictions))
print("F1-score:", f1_score(test_df[target_col], predictions))
```

```
# 7. Run experiments
train_and_evaluate(0.30) # 70/30
train_and_evaluate(0.20) # 80/20
train_and_evaluate(0.10) # 90/10
```

/usr/local/lib/python3.12/dist-packages/sklearn/linear\_model/\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta  
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

Train/Test Split: 70/30

Accuracy: 0.8171641791044776

Confusion Matrix:

```
[[139 18]
```

```
[ 31 80]]
```

Precision: 0.8163265306122449

Recall: 0.7207207207207207

F1-score: 0.7655502392344498

/usr/local/lib/python3.12/dist-packages/sklearn/linear\_model/\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta  
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

Train/Test Split: 80/20

Accuracy: 0.8212290502793296

Confusion Matrix:

```
[[92 13]
```

```
[19 55]]
```

Precision: 0.8088235294117647

Recall: 0.7432432432432432

F1-score: 0.7746478873239436

Train/Test Split: 90/10

Accuracy: 0.8777777777777778

Confusion Matrix:

```
[[48  6]
```

```
[ 5 31]]
```

Precision: 0.8378378378378378

Recall: 0.8611111111111112

F1-score: 0.8493150684931506

/usr/local/lib/python3.12/dist-packages/sklearn/linear\_model/\_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (sta  
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

```
n_iter_i = _check_optimize_result(
```

```
df = pd.read_csv('diabetes.csv')
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

```
# 1. Imports
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, roc_curve

# 2. Load dataset
df = pd.read_csv("diabetes.csv") # change filename if needed

# 3. Separate features and target
X = df.drop(columns=["Outcome"])
y = df["Outcome"]

# 4. Function to train & evaluate
def train_and_evaluate(test_size):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42
    )

    # Feature Scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Logistic Regression
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob)
    cm = confusion_matrix(y_test, y_pred)

    print(f"\nTrain/Test Split: {int((1-test_size)*100)}/{int(test_size*100)}")
    print("Accuracy:", acc)
    print("ROC-AUC Score:", roc_auc)
    print("Confusion Matrix:\n", cm)

    return roc_auc, y_test, y_prob

# 5. Run experiments
roc_60, y_60, p_60 = train_and_evaluate(0.40) # 60/40
roc_75, y_75, p_75 = train_and_evaluate(0.25) # 75/25
roc_80, y_80, p_80 = train_and_evaluate(0.20) # 80/20

# 6. Select best split (highest ROC-AUC)
roc_scores = {
    "60/40": (roc_60, y_60, p_60),
    "75/25": (roc_75, y_75, p_75),
    "80/20": (roc_80, y_80, p_80)
}

best_split = max(roc_scores, key=lambda x: roc_scores[x][0])
best_roc, best_y, best_prob = roc_scores[best_split]

print(f"\nBest Split Based on ROC-AUC: {best_split}")

# 7. Plot ROC Curve for best split
fpr, tpr, _ = roc_curve(best_y, best_prob)

plt.figure()
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {best_roc:.2f})")
plt.plot([0, 1], [0, 1], linestyle="--")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f"ROC Curve for Best Split ({best_split})")
plt.legend()
plt.show()

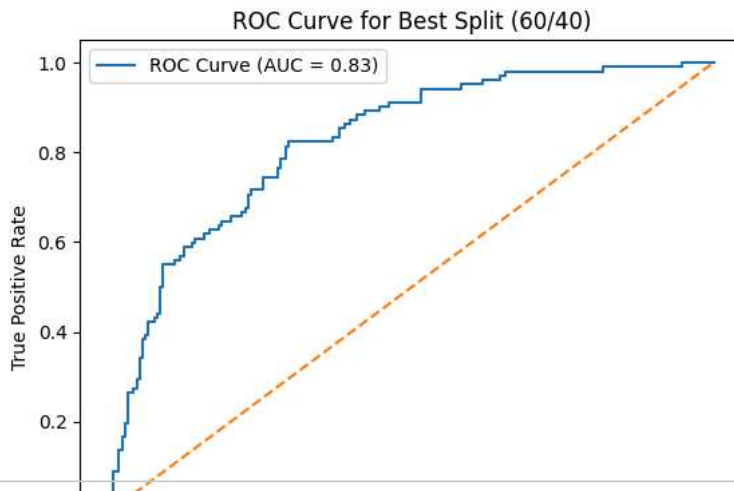
```

Train/Test Split: 60/40  
 Accuracy: 0.7564935064935064  
 ROC-AUC Score: 0.8250523510375023  
 Confusion Matrix:  
 [[168 38]  
 [ 37 65]]

Train/Test Split: 75/25  
 Accuracy: 0.7291666666666666  
 ROC-AUC Score: 0.7942735949098622  
 Confusion Matrix:  
 [[95 28]  
 [24 45]]

Train/Test Split: 80/20  
 Accuracy: 0.7532467532467533  
 ROC-AUC Score: 0.8146923783287419  
 Confusion Matrix:  
 [[79 20]  
 [18 37]]

Best Split Based on ROC-AUC: 60/40



```
df = pd.read_csv('heart_disease_uci.csv')
df.head()
```

	id	age	sex	dataset	False Positive Rate		cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	r
0	1	63	Male	Cleveland	typical	angina		145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	
1	2	67	Male	Cleveland	asymptomatic			160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	
2	3	67	Male	Cleveland	asymptomatic			120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversible defect	

```
# 1. Imports
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.impute import SimpleImputer # Import SimpleImputer

# 2. Load dataset
df = pd.read_csv("heart_disease_uci.csv") # use your exact filename

# 3. Convert target to binary (num > 0 → 1)
df["num"] = df["num"].apply(lambda x: 1 if x > 0 else 0)

target_col = "num"

# 4. Identify feature types
numeric_features = df.drop(columns=[target_col]).select_dtypes(include=["int64", "float64"]).columns
```

```

categorical_features = df.drop(columns=[target_col]).select_dtypes(include=["object", "bool"]).columns

# 5. Preprocessing
numeric_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="mean")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline([
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer([
    ("num", numeric_transformer, numeric_features),
    ("cat", categorical_transformer, categorical_features)
])

# 6. Train & evaluate function
def train_and_evaluate(test_size):
    train_df, test_df = train_test_split(
        df, test_size=test_size, random_state=42
    )

    model = Pipeline([
        ("preprocessor", preprocessor),
        ("classifier", LogisticRegression(max_iter=1000))
    ])

    model.fit(
        train_df.drop(columns=[target_col]),
        train_df[target_col]
    )

    predictions = model.predict(
        test_df.drop(columns=[target_col])
    )

    # Metrics
    acc = accuracy_score(test_df[target_col], predictions)
    prec = precision_score(test_df[target_col], predictions)
    rec = recall_score(test_df[target_col], predictions)
    f1 = f1_score(test_df[target_col], predictions)
    cm = confusion_matrix(test_df[target_col], predictions)

    print(f"\nTrain/Test Split: {int((1-test_size)*100)}/{int(test_size*100)}")
    print("Accuracy:", acc)
    print("Precision:", prec)
    print("Recall:", rec)
    print("F1-score:", f1)

    # Confusion Matrix Plot
    plt.figure()
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title(f"Confusion Matrix ({int((1-test_size)*100)}/{int(test_size*100)})")
    plt.show()

    return acc, prec, rec, f1

# 7. Run experiments
results_70 = train_and_evaluate(0.30) # 70/30
results_80 = train_and_evaluate(0.20) # 80/20
results_85 = train_and_evaluate(0.15) # 85/15

```



Train/Test Split: 70/30  
 Accuracy: 0.855072463768116  
 Precision: 0.8766233766233766  
 Recall: 0.8653846153846154  
 F1-score: 0.8709677419354839

Confusion Matrix (70/30)

```
df = pd.read_csv('data.csv')
df.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974
3	84348301	M	11.42	20.36	77.58	386.1	0.14250	0.28390	0.2414
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980

```
# 1. Imports
import pandas as pd
import matplotlib.pyplot as plt
import 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 accuracy_score, confusion_matrix, roc_auc_score

# 2. Load dataset
df = pd.read_csv("data.csv") # use your filename

# 3. Remove unnecessary columns
df = df.drop(columns=["id", "Unnamed: 32"], errors="ignore")

# 4. Encode target (M = 1, B = 0)
df["diagnosis"] = df["diagnosis"].map({"M": 1, "B": 0})

# 5. Separate features and target
X = df.drop(columns=["diagnosis"])
y = df["diagnosis"]

# 6. Function to train & evaluate
def train_and_evaluate(test_size):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42
    )

    # Scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Logistic Regression
    model = LogisticRegression(max_iter=1000)
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_prob)
    cm = confusion_matrix(y_test, y_pred)

    print(f"\nTrain/Test Split: {int((1-test_size)*100)}/{int(test_size*100)}")
    print("Accuracy:", acc)
    print("ROC-AUC:", roc_auc)

    # Confusion Matrix plot
    plt.figure()
```

```
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title(f"Confusion Matrix ({int((1-test_size)*100)}/{int(test_size*100)})")
plt.show()

return acc, roc_auc

# 7. Run experiments
res_50 = train_and_evaluate(0.50) # 50/50
res_70 = train_and_evaluate(0.30) # 70/30
res_80 = train_and_evaluate(0.20) # 80/20
```



Train/Test Split: 50/50  
Accuracy: 0.9859649122807017

```
df = pd.read_csv('Social_Network_Ads.csv')
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	19	20000	0
2	15668575	Female	26	43000	0
3	15613246	Female	27	57000	0
4	15904002	Male	19	76000	0

```
# 1. Imports
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.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# 2. Load dataset
df = pd.read_csv("Social_Network_Ads.csv") # use your file name

# 3. Select features and target
X = df[["Age", "EstimatedSalary"]]
y = df["Purchased"]

# 4. Function to train & evaluate
def train_and_evaluate(test_size):
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42
    )

    # Scaling
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Logistic Regression
    model = LogisticRegression()
    model.fit(X_train_scaled, y_train)

    # Predictions
    y_pred = model.predict(X_test_scaled)

    print(f"\nTrain/Test Split: {int((1-test_size)*100)}/{int(test_size*100)}")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))

    return model, scaler, accuracy_score(y_test, y_pred)

# 5. Run experiments
model_65, scaler_65, acc_65 = train_and_evaluate(0.35) # 65/35
model_75, scaler_75, acc_75 = train_and_evaluate(0.25) # 75/25
model_80, scaler_80, acc_80 = train_and_evaluate(0.20) # 80/20

# 6. Choose best split (highest accuracy)
accuracies = {
    "65/35": (acc_65, model_65, scaler_65),
    "75/25": (acc_75, model_75, scaler_75),
    "80/20": (acc_80, model_80, scaler_80)
}

best_split = max(accuracies, key=lambda x: accuracies[x][0])
best_model, best_scaler = accuracies[best_split][1], accuracies[best_split][2]

print(f"\nBest Split Based on Accuracy: {best_split}")

# 7. Plot Decision Boundary (2D)
```

```
X_scaled = best_scaler.transform(X)

x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() + 1
y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() + 1

xx, yy = np.meshgrid(
    np.arange(x_min, x_max, 0.01),
    np.arange(y_min, y_max, 0.01)
)

Z = best_model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.figure()
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=y, edgecolor="k")
plt.xlabel("Age (Scaled)")
plt.ylabel("Estimated Salary (Scaled)")
```