```
#✅ Step 1 – Import the required libraries

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score
from sklearn.preprocessing import StandardScaler
```

```
#✅ Step 2 – Load the dataset

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

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	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
4	0	137	40	35	168	43.1	2.288	33	1

Next steps: Generate code with df New interactive sheet

```
# ✅ Step 3 – Separate the data into input (X) and output (y)

X = df.drop('Outcome', axis=1)   # X has all columns except Outcome → input
y = df['Outcome']                # y has only Outcome column → output (label)
```

💡 What this means (in very simple words):

Part	Meaning
`Outcome` column	This tells if the person is diabetic → 1 = Yes, 0 = No
`X`	All the health-related values (Glucose, BMI, Age, etc.) used to **predict diabetes**
`y`	The actual answer (diabetic or not) we want the model to learn and predict

```
# ✅ Step 4 – Split data into Training and Testing sets

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.2, random_state = 42
)
```

💡 What this means (in very simple words):

Term	Meaning
Training data (X_train, y_train)	Used to **teach** the model.
Testing data (X_test, y_test)	Used to **check if the model learned correctly**.
test_size=0.2	20% of data = for testing, 80% = for training.
random_state=42	Keeps the result same every time (important for repeatable output).

```
#✅ Step 5 – Scale (Normalize) the Data – Important for KNN

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

💡 What this means in super simple words:

KNN works by measuring distance between points.

If one column has values like Glucose = 150 and another has BMI = 30, then Glucose will dominate because it has bigger numbers → bad for accuracy.

So we scale all features to the same range, usually around -1 to +1.

fit_transform() → learns scaling from training data and applies it.

transform() → applies the same scaling to test data (never “fit” on test data!).

```
#✅ Step 6 – Train the K-Nearest Neighbors (KNN) Model

model = KNeighborsClassifier(n_neighbors = 5)
model.fit(X_train, y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

💡 What this means in simple words:

Code	Meaning
`KNeighborsClassifier(n_neighbors=5)`	We tell the algorithm: "Look at the 5 closest data points to make a prediction."
`model.fit(X_train, y_train)`	This **trains** the model using the training data.

🧠 KNN Logic: If a new patient comes... The model looks at 5 similar past patients. If 3 or more out of 5 were diabetic ⟶ predicts diabetic.

```
#✅ Step 7 — Make Predictions using the Trained KNN Model

y_pred = model.predict(X_test)
```

💡 What this means (super easy words):

X_test ⟶ data that model has never seen before.

model.predict() ⟶ tells whether each person in test data is diabetic (1) or not diabetic (0).

Result is stored in y_pred.

Now we have:

y_test ⟶ actual answers (real diabetes status)

y_pred ⟶ model's predicted answers

```
#✅ Step 8 — Evaluate the KNN Model (Confusion Matrix, Accuracy, Precision, Recall, Error Rate)

cm = confusion_matrix(y_test, y_pred)

acc = accuracy_score(y_test, y_pred)

prec = precision_score(y_test, y_pred)

rec = recall_score(y_test, y_pred)

# Error Rate (wrong predictions)
error_rate = 1 - acc

# print all values...
print("Confusion Matrix:\n", cm)
print("\nAccuracy:", round(acc, 3))
print("Error Rate:", round(error_rate, 3))
print("Precision:", round(prec, 3))
print("Recall:", round(rec, 3))
```

```
Confusion Matrix:
 [[79 20]
 [27 28]]

Accuracy: 0.695
Error Rate: 0.305
Precision: 0.583
Recall: 0.509
```

💡 What each metric means (very simple):

Metric	Meaning
Confusion Matrix	Shows **correct vs wrong** predictions.
Accuracy	% of total correct predictions.
Error Rate	% of incorrect predictions = 1 - Accuracy.
Precision	Of all predicted **diabetic (1)**, how many were really diabetic?
Recall	Out of actual diabetic people, how many did the model correctly identify?