code loads diabetes.csv, treats the column Outcome as the target, trains a **LinearRegression** model and evaluates it. Linear regression predicts a continuous value; since Outcome is usually 0 or 1 (binary), this is **technically a regression-on-binary** approach — it can work but usually **Logistic Regression** (classification) is more appropriate. I’ll explain that as we go and show how to switch to classification.

**Step-by-step explanation of your code**

**1) Imports**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

* pandas for loading/manipulating the CSV.
* matplotlib.pyplot for plotting.
* train\_test\_split to split data into training and test sets.
* LinearRegression — the model class from scikit-learn.
* mean\_squared\_error, r2\_score — regression evaluation metrics.

**2) Load dataset**

df = pd.read\_csv("diabetes.csv")

print("Dataset shape:", df.shape)

print(df.head())

* pd.read\_csv reads the CSV file into a DataFrame.
* df.shape shows rows × columns.
* df.head() displays the first 5 rows so you can inspect column names and example values.

**Tip:** Always inspect for missing values, wrong dtypes, or obviously incorrect zeros (e.g., in medical datasets, some columns may have zeros representing missingness).

**3) Split features and target**

X = df.drop("Outcome", axis=1) # Features

y = df["Outcome"] # Target (we treat it as regression)

* X contains all columns except Outcome.
* y is the target vector (0/1 in typical diabetes datasets).
* **Important note:** You are using Outcome as a regression target. That means the model will predict continuous values (e.g. 0.2, 0.7) even though the real labels are 0 or 1.

**4) Train / test split**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

* test\_size=0.2 puts 20% of data into the test set, 80% in training.
* random\_state=42 seeds the shuffle so results are reproducible.
* The split helps measure generalization (performance on unseen data).

**5) Train the linear regression**

model = LinearRegression()

model.fit(X\_train, y\_train)

* LinearRegression() builds the ordinary least squares (OLS) linear model.
* .fit() computes the intercept and coefficients by minimizing squared error on the training set.

**6) Predict on test set**

y\_pred = model.predict(X\_test)

* y\_pred contains continuous predicted values (not 0/1 necessarily).
* If you want predicted classes, you’d typically threshold (e.g. y\_pred\_class = (y\_pred >= 0.5).astype(int)), but that’s a hack compared to using logistic regression.

**7) Model outputs: intercept & coefficients**

print("Intercept:", model.intercept\_)

print("Coefficients:", model.coef\_)

* **Intercept** = predicted y when all features = 0.
* **Coefficients** = array of weights; each weight = expected change in prediction per unit increase in that feature *holding other features constant*.
* Example: if coefficient for Glucose is 0.02, then a one-unit increase in glucose increases predicted Outcome by 0.02.

**Caveat for binary y:** interpreting these as probabilities is approximate — linear regression can give outputs outside [0,1].

**8) Evaluation: MSE and R²**

print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred))

print("R² Score:", r2\_score(y\_test, y\_pred))

* **MSE** = average squared difference between true values and predictions. Lower is better.
* **R²** (coefficient of determination) measures fraction of variance explained by the model (1 = perfect, 0 = as good as predicting the mean, negative = worse than mean predictor).

**Important:** For a binary target:

* MSE and R² *are defined* but they are not the usual choice for evaluating classification tasks. They tell you how far continuous predictions are from 0/1 but don’t measure classification quality directly.

**9) Plot Actual vs Predicted**

plt.scatter(y\_test, y\_pred, color="blue")

plt.xlabel("Actual Values")

plt.ylabel("Predicted Values")

plt.title("Linear Regression - Actual vs Predicted")

plt.show()

* This scatter puts actual (x-axis) vs predicted (y-axis).
* For binary y\_test values (0 or 1) you’ll see two horizontal rows of points (one at x=0, one at x=1). Predicted values near 0 or 1 cluster near those rows.
* If many predicted points for actual 1 are below 0.5, the model is poor at separating classes.

**Plotting tips:** add some jitter to y\_test if points overlap, or plot a histogram of y\_pred separated by true class to see separation.

**Why Linear Regression for a binary Outcome is not ideal**

* Linear regression can predict values outside [0,1].
* It assumes linear relationship and Gaussian residuals — not appropriate for a binary dependent variable.
* **Logistic Regression** models the log-odds and naturally outputs probabilities in [0,1], and is the standard for binary classification.

**If you meant classification → use Logistic Regression (example)**

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

clf = LogisticRegression(max\_iter=1000) # increase max\_iter if needed

clf.fit(X\_train, y\_train)

y\_pred\_class = clf.predict(X\_test)

y\_pred\_prob = clf.predict\_proba(X\_test)[:,1] # probability for class 1

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_class))

print("Confusion matrix:\n", confusion\_matrix(y\_test, y\_pred\_class))

print("Classification report:\n", classification\_report(y\_test, y\_pred\_class))

print("ROC AUC:", roc\_auc\_score(y\_test, y\_pred\_prob))

* predict\_proba gives the predicted probability for the positive class (useful for ROC AUC).
* classification\_report provides precision, recall, f1-score — the right kind of metrics for classification.

**Useful checks & improvements (practical checklist)**

1. **Missing / invalid values**: check df.isnull().sum() and domain-specific invalids (e.g., columns that shouldn’t be zero).
2. **Feature scaling**: many classifiers benefit from scaling:
3. from sklearn.preprocessing import StandardScaler
4. scaler = StandardScaler()
5. X\_train\_scaled = scaler.fit\_transform(X\_train)
6. X\_test\_scaled = scaler.transform(X\_test)
7. **Class imbalance**: if Outcome is imbalanced, consider stratified split (train\_test\_split(..., stratify=y)) and metrics like ROC AUC, precision/recall.
8. **Cross-validation**: use cross\_val\_score or GridSearchCV for hyperparameter tuning and robust estimates.
9. **Regularization**: for logistic regression, consider C parameter or use penalty='l1'/'l2'.
10. **Feature selection / multicollinearity**: check correlations and consider removing/reducing correlated features.
11. **Interpretability**: coefficients in logistic regression are in log-odds scale — exponentiate to get odds ratios.

**Checking linear regression assumptions (if you still use OLS)**

* **Linearity**: residuals should have no pattern vs fitted values (plot residuals).
* **Homoscedasticity**: residual variance constant across predictions.
* **Normality of residuals**: residual histogram or QQ plot.
* **Independence**: samples independent.

Example residual plot:

residuals = y\_test - y\_pred

plt.scatter(y\_pred, residuals)

plt.axhline(0, color='k', linestyle='--')

plt.xlabel("Predicted")

plt.ylabel("Residuals")

plt.show()

**Short summary / recommendation**

* Your script runs fine technically, but because Outcome is binary, **switch to LogisticRegression** if your goal is to **classify** diabetes vs no diabetes.
* If you intentionally want to model probability as a continuous target, you can proceed with LinearRegression but be aware of interpretation problems and that logistic regression is almost always better for binary targets.
* Apply preprocessing (missing values, scaling), consider cross-validation, and use classification metrics if you change to logistic regression.

Let’s take your **Linear Regression diabetes script** and re-write it for **Logistic Regression** (the correct model for binary outcomes).

Here’s the full working code you can run directly in **PyCharm**:

# Logistic Regression on Diabetes Dataset

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

# Step 1: Load dataset

df = pd.read\_csv("diabetes.csv")

print("Dataset shape:", df.shape)

print(df.head())

# Step 2: Split features (X) and target (y)

X = df.drop("Outcome", axis=1) # Features

y = df["Outcome"] # Target (binary: 0 = No Diabetes, 1 = Diabetes)

# Step 3: Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y

)

# Step 4: Train Logistic Regression model

model = LogisticRegression(max\_iter=1000)

model.fit(X\_train, y\_train)

# Step 5: Make predictions

y\_pred\_class = model.predict(X\_test) # Predicted classes (0/1)

y\_pred\_prob = model.predict\_proba(X\_test)[:, 1] # Predicted probability for class 1

# Step 6: Evaluate model

print("Accuracy:", accuracy\_score(y\_test, y\_pred\_class))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_class))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_class))

print("ROC AUC Score:", roc\_auc\_score(y\_test, y\_pred\_prob))

# Step 7: Plot ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

plt.plot(fpr, tpr, label="Logistic Regression")

plt.plot([0, 1], [0, 1], linestyle="--", color="gray")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()

**🔹 What Changed from Linear Regression?**

1. **Model**: switched from LinearRegression() → LogisticRegression().
2. **Predictions**:
   * predict() → class labels (0 or 1).
   * predict\_proba() → probabilities (useful for ROC curve, thresholds).
3. **Evaluation**: used **classification metrics** (accuracy, precision, recall, f1-score, confusion matrix, ROC AUC).
4. **ROC Curve**: plots trade-off between sensitivity (TPR) and specificity (1-FPR).

**What is a Decision Tree in Machine Learning?**

A Decision Tree is a supervised learning algorithm that is used for both classification (predict categories like “Diabetic / Not Diabetic”) and regression (predict numbers like “House Price”).

It works like a flowchart:

* Each internal node → represents a condition (e.g., *"Glucose > 120?"*).
* Each branch → represents the outcome of that condition (Yes/No).
* Each leaf node → represents a final decision or prediction (e.g., *"Diabetic"*).

⚙️ How it Works

1. The algorithm looks at all features (columns in your dataset).
2. It finds the best question (split) to divide the data into groups that are as “pure” as possible.
   * “Pure” means most samples in a group belong to the same class.
3. It repeats this process recursively, creating branches, until:
   * It reaches a stopping condition (e.g., max depth), or
   * All samples in a group belong to the same class.

📊 Example (Diabetes dataset)

Imagine we want to predict if a person has diabetes (Outcome = 0 or 1):

* Root node: *Is Glucose > 130?*
  + If Yes → next split: *Is BMI > 30?*
    - If Yes → Predict Diabetes (1)
    - If No → Predict No Diabetes (0)
  + If No → Predict No Diabetes (0)

It’s literally like answering a series of Yes/No questions until we reach a decision.

✅ Advantages

* Easy to understand & visualize.
* Can handle both numerical and categorical data.
* No need for feature scaling (like normalization).

❌ Disadvantages

* Prone to overfitting if not controlled (tree grows too deep).
* Small changes in data can produce very different trees.
* Usually not as accurate as ensemble methods (Random Forest, XGBoost).

Let’s simplify everything and focus only on the **Decision Tree Classifier** using your diabetes.csv.

Here’s the clean code you can run directly in **PyCharm**:

# Decision Tree Classifier on Diabetes Dataset

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score, roc\_curve

# Step 1: Load dataset

df = pd.read\_csv("diabetes.csv")

print("Dataset shape:", df.shape)

print(df.head())

# Step 2: Split features (X) and target (y)

X = df.drop("Outcome", axis=1)

y = df["Outcome"]

# Step 3: Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y

)

# Step 4: Train Decision Tree model

clf = DecisionTreeClassifier(criterion="gini", max\_depth=4, random\_state=42)

clf.fit(X\_train, y\_train)

# Step 5: Predictions

y\_pred = clf.predict(X\_test)

y\_pred\_prob = clf.predict\_proba(X\_test)[:, 1]

# Step 6: Evaluation

print("\n=== Decision Tree Results ===")

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred))

print("ROC AUC:", roc\_auc\_score(y\_test, y\_pred\_prob))

# Step 7: Plot ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)

plt.plot(fpr, tpr, label="Decision Tree")

plt.plot([0, 1], [0, 1], linestyle="--", color="gray")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve - Decision Tree")

plt.legend()

plt.show()

# Step 8: Visualize the Decision Tree

plt.figure(figsize=(16,10))

plot\_tree(

clf,

feature\_names=X.columns,

class\_names=["No Diabetes", "Diabetes"],

filled=True,

rounded=True,

fontsize=10

)

plt.show()

**🔹 What this script does**

1. Loads your diabetes.csv.
2. Splits data into train/test sets.
3. Trains a **Decision Tree Classifier** (depth limited to 4 for readability).
4. Prints **Accuracy, Confusion Matrix, Classification Report, ROC AUC**.
5. Plots the **ROC Curve**.
6. Displays the **Decision Tree visualization**.

**Comparison Table**

| **Feature** | **Linear Regression** | **Logistic Regression** | **Decision Tree** |
| --- | --- | --- | --- |
| **Type** | Regression | Classification (mainly binary) | Regression & Classification |
| **Output** | Continuous values | Probability (0–1), then class | Class label or continuous value |
| **Equation-based** | Yes (linear equation) | Yes (sigmoid/logit function) | No (tree-based rules) |
| **Assumptions** | Linear relationship, normal errors | Linear relation in log-odds | No strict assumptions |
| **Handles Non-linearity** | ❌ Poor | ❌ Poor | ✅ Excellent |
| **Interpretability** | ✅ Easy (coefficients) | ✅ Easy (odds ratios) | ✅ Very intuitive (rules) |
| **Overfitting risk** | Low–Moderate | Low–Moderate | High (can be reduced with pruning/ensemble methods) |
| **Examples** | House price, salary | Diabetes, churn, spam detection | Loan approval, fraud detection |