



## ⌚ PRACTICAL OVERVIEW

This practical demonstrates how to apply **Gradient Descent** to find the **local minimum** of a function.

We minimize the function:

$$y = (x + 3)^2$$

starting from initial point  $x = 2$ .

**Subject:** Machine Learning (ML)

**Concept:** Optimization using Gradient Descent algorithm

**Goal:** Find the value of  $x$  where the function reaches its minimum.

---



## PROBLEM STATEMENT

Implement Gradient Descent to find the local minimum of the function:

$$y = (x + 3)^2$$

using an initial guess of  $x = 2$ , learning rate = 0.1, and run for 50 iterations.

---

## 🔍 CODE EXPLANATION (Step-by-Step)

```
def func(x):  
    return (x + 3) ** 2
```

This defines the function whose minimum we want to find.

```
def grad(x):
```

```
    return 2 * (x + 3)
```

This is the first-order derivative (gradient) of the function.

```
x = 2
```

```
learning_rate = 0.1
```

```
epochs = 50
```

- Starting value (initial guess) = **2**

- **Learning rate** controls step size
- **epochs** represent iteration count

`x_vals = [x]`

`y_vals = [func(x)]`

These lists store all step values for plotting.

### Gradient Descent Loop

for i in range(epochs):

`dx = grad(x)`

`x = x - learning_rate * dx`

This is the core update rule:

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot \nabla f(x)$$

`x_vals.append(x)`

`y_vals.append(func(x))`

We store each updated value for visualization.

`print(f"Iteration {i+1}: x = {x:.5f}, f(x) = {func(x):.5f}")`

Displays progress at each iteration.

### Final Output

`print(f"\nLocal minimum at x = {x:.5f}, f(x) = {func(x):.5f}")`

### Plotting Section

- Plots the function curve
- Plots red dots → gradient descent steps
- Shows movement toward the minimum

## ⚙️ ALGORITHM & COMPLEXITY

### Gradient Descent Algorithm

1. Choose starting point  $x_0$
2. Compute gradient  $f'(x)$
3. Update:  $x_{k+1} = x_k - \alpha f'(x_k)$

4. Repeat until convergence

## Complexity

Measure	Value
---------	-------

Time Complexity  $O(n)$ (n = number of iterations)

Space Complexity  $O(n)$ (for storing history)

---

## INPUT

No runtime input required.

Values used inside code:

- Initial guess = **2**
  - Learning rate = **0.1**
  - Iterations = **50**
- 

## OUTPUT EXPLANATION

### Example printed output (first few iterations):

Iteration 1:  $x = 1.40000, f(x) = 19.36000$

Iteration 2:  $x = 0.88000, f(x) = 15.05440$

Iteration 3:  $x = 0.50400, f(x) = 12.25442$

...

Iteration 50:  $x \approx -2.99999, f(x) \approx 0$

### Final Results:

Local minimum at  $x \approx -3$

Minimum value  $f(x) \approx 0$

### Graph Interpretation

- Blue curve = function shape (U-shaped parabola)
  - Red dots = gradient descent steps moving down the curve toward minimum
  - Dots get closer together → slope becomes smaller → convergence
-

## EXAM TIPS & VIVA QUESTIONS

Question	Best Answer
<b>Why Gradient Descent?</b>	To minimize cost/loss functions and optimize model parameters.
<b>Why move opposite to gradient?</b>	Because gradient gives direction of increase; moving opposite decreases value.
<b>What is Learning Rate?</b>	A parameter that controls step size. Too high → diverge, too low → slow.
<b>What is Cost Function?</b>	A function that evaluates model performance; gradient descent tries to minimize it.
<b>Global vs Local Minimum?</b>	Local minimum is lowest nearby point; global minimum is absolute lowest point in function.

---

## PRACTICE TASKS (Likely viva modifications)

Try This	Purpose
Change learning rate to 0.01	Observe slower convergence
Change start point to $x = -10$	Demonstrates descent from far point
Replace function with $y = x^2 - 4x + 4$	Practice on general quadratic function

---

If you want, I can now **convert this into your final practical journal format (handwritten/typed PDF)**.

Just say: "Make Journal Ready PDF" 

**You said:**

explain gradient decent and local minima

**ChatGPT said:**

Sure — here is a **clear, exam-ready** explanation of **Gradient Descent** and **Local Minima** in simple but accurate language.

---

 **Gradient Descent (Explanation)**

**Gradient Descent** is an **optimization algorithm** used to **find the minimum (lowest value)** of a function.

In Machine Learning, it is used to **minimize the Cost Function** (error) so that the model learns better.

It works by **starting from any initial value** and then **taking small steps downhill** in the direction where the function decreases.

### Key Idea:

- If the **slope (derivative)** of the function at a point is **positive**, move **left**.
- If the **slope (derivative)** is **negative**, move **right**.
- Repeat until you reach the point where the slope becomes **zero** → that is the **minimum**.

### Update Formula (Very Important for Viva)

$$x_{\text{new}} = x_{\text{old}} - \alpha \cdot \frac{d}{dx} f(x)$$

Where:

- $\alpha$ = **Learning Rate** (step size)
- $\frac{d}{dx} f(x)$ = **Gradient (slope)** of the function

### Intuition

Think of a person walking **down a hill** blindfolded.

They check the slope and walk in the **downward direction** until they reach the valley.

---

### Local Minima (Explanation)

A **local minimum** is a point in the function where the function's value is **lower than nearby points**, but **not necessarily the lowest overall**.

### Example

For the function:

$$y = (x + 3)^2$$

The graph is a **U-shaped parabola**, and its lowest point is at:

$$x = -3, y = 0$$

This is the **minimum point** of the function → also called **local minima**, and in this case, it is also the **global minimum** because no value is lower than 0.

### 📌 Characteristics of Local Minima:

- Slope (derivative) = **0** at this point
  - The curve **changes direction** here
  - Cost function stops decreasing → **convergence**
- 

### ⭐ Simple Summary (Say This in Viva)

Term	Meaning
<b>Gradient Descent</b>	A method that reduces the cost function step-by-step by moving in the direction of the <b>negative gradient</b> .
<b>Local Minima</b>	The point where the cost function is <b>minimum (lowest)</b> and the gradient (slope) becomes <b>zero</b> .

---

### 🗣 Viva-Type Short Oral Answer

**Q:** What is gradient descent?

**A:** Gradient Descent is an iterative optimization algorithm used to minimize a cost function by updating parameters in the direction opposite to the gradient.

**Q:** What is local minima?

**A:** Local minima is the point where the function reaches its minimum value, and the slope becomes zero. Gradient descent tries to reach this point.

---

If you'd like, I can now:

- Convert this into **handwritten notebook wording**
- Make a **diagram** like your exam book
- Give **example viva questions with answers**

Just tell me 😊

## 1. Function Definition

```
def func(x):  
    return (x + 3) ** 2
```

- `def func(x):` → This defines a function named **func** that takes one input **x**.
- `return (x + 3) ** 2` → It returns the value of the mathematical expression  $(x + 3)^2$ .

This is the function we want to **minimize**.

---

## 2. Gradient (Derivative) Definition

```
def grad(x):  
    return 2 * (x + 3)
```

- `def grad(x):` → Defines another function named **grad** (short for gradient).
- `return 2 * (x + 3)` → This returns the **derivative** of  $(x + 3)^2$ , which is  $2(x + 3)$ .

The gradient tells us **which direction the function increases**.

---

## 3. Gradient Descent Parameters

`x = 2`

`learning_rate = 0.1`

`epochs = 50`

- `x = 2` → Starting value (initial guess).
  - `learning_rate = 0.1` → Step size (how big our updates are each time).
  - `epochs = 50` → Number of times we update **x**.
- 

## 4. Lists for Storing Values (for plotting later)

`x_vals = [x]`

`y_vals = [func(x)]`

- `x_vals` stores all **x** values.
- `y_vals` stores the function values at those **x** values.  
This helps us show how **x** changes during gradient descent.

---

## 5. Gradient Descent Loop

for i in range(epochs):

- Run the update step 50 times.

$dx = \text{grad}(x)$

- $dx$  = the derivative (slope) at current  $x$ .

$x = x - \text{learning\_rate} * dx$

- Update rule:

$x_{\text{new}} = x_{\text{old}} - \text{learning\_rate} \times \text{slope}$

- We **subtract** because we want to move **downhill** (towards minimum).

$x\_vals.append(x)$

$y\_vals.append(func(x))$

- Save new  $x$  and its  $y$ -value for plotting.

print(f'Iteration {i+1}: x = {x:.5f}, f(x) = {func(x):.5f}')

- Print the iteration number, the current  $x$  value, and function value.

- $:.5f$  means **show 5 decimal places**.

---

## 6. Final Result

print(f"\nLocal minimum at x = {x:.5f}, f(x) = {func(x):.5f}")

After finishing all iterations, we print the final  $x$  and function value.

---

## 7. Plotting

$x\_plot = [i \text{ for } i \text{ in range}(-10, 5)]$

$y\_plot = [func(i) \text{ for } i \text{ in } x\_plot]$

- This creates a range of  $x$  values and their corresponding  $y$  values to draw the curve.

`plt.figure(figsize=(8,5))`

`plt.plot(x_plot, y_plot, label="y = (x + 3)^2")`

- Plots the function.

```
plt.scatter(x_vals, y_vals, color='red', label="Gradient Descent Steps")
plt.plot(x_vals, y_vals, linestyle='--', color='gray', alpha=0.6)
    • Plots the movement of x during gradient descent to show how it approaches the minimum.

plt.title("Gradient Descent to Find Local Minima")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
    • Adds labels, legend, grid, and displays the graph.
```

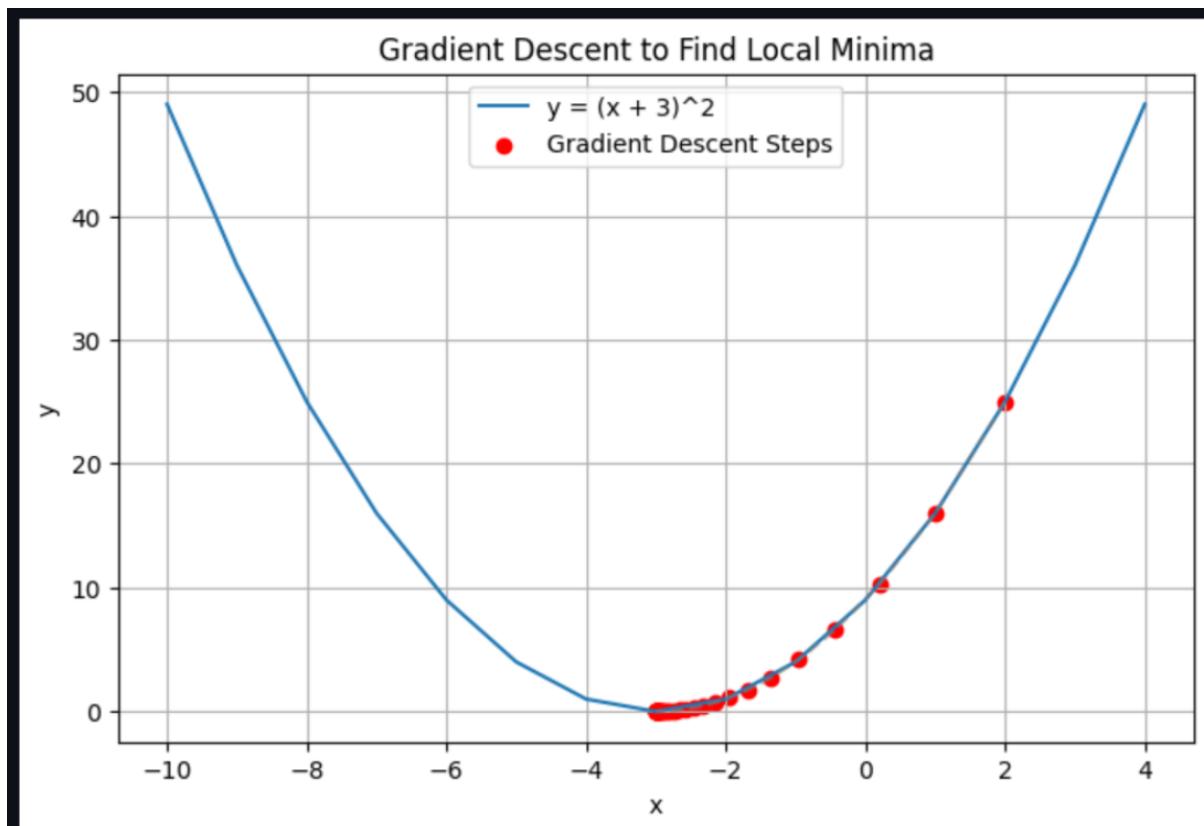
---

## In Simple Words

- The function  $(x + 3)^2$  has a **minimum** at **x = -3**.
  - Starting from **x = 2**, gradient descent moves step-by-step **closer to -3**.
  - The loop updates x using the slope.
  - Finally, x reaches near **-3**.
- 

If you want, I can now also give you a **visual animation** or **handwritten explanation**.

**You said:**



ChatGPT said:

Great — this plot exactly shows **how Gradient Descent moves step-by-step toward the minimum of the function**

$$f(x) = (x + 3)^2$$

Let's break down what you are seeing **in the graph**:

### ✓ Blue Curve

This is the graph of the function:

$$y = (x + 3)^2$$

- It's a **U-shaped parabola**.
- The **lowest point (minimum)** of this parabola is at  $x = -3$ , where  $f(x) = 0$ .

### ● Red Dots (Gradient Descent Steps)

Each red dot shows a **new value of x** calculated during gradient descent.

- You started from  $x = 2$  (which is on the right side of the curve).
  - Gradient descent uses the **slope (gradient)** to move **downhill** toward the minimum.
  - The dots move **left** step-by-step until they get very close to  $x = -3$ .
- 

## What's Happening?

At each step:

$$x_{\text{new}} = x_{\text{old}} - \text{learning\_rate} \cdot \text{gradient}(x)$$

Since  $\text{learning\_rate} = 0.1$ , the steps are **small and controlled**.

The red dots slowly **decrease** and begin **converging near  $x = -3$** , where the curve is flat.

This shows that:

- **When slope is large**, gradient descent moves fast.
  - **When slope is small (near the bottom)**, steps become smaller.
  - Finally, **x stabilizes at -3 — the minimum**.
- 

## Key Understanding

Part	Meaning
Function bottom (minimum) At $x = -3$	
Red points	Path taken by gradient descent
Movement direction	From <b>right to left</b> (towards minimum)
Why it stops near -3	Because slope becomes $\sim 0$ there

---

## In Simple Words

- The blue curve is the hill.
- The red dots are you **walking downhill**.
- You stop when the ground becomes **flat**.



```
# ===== KNN on Pima Indians Diabetes dataset =====  
# Save this as knn_diabetes.py (or run in a notebook cell)
```

```
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.neighbors import KNeighborsClassifier  
from sklearn.metrics import (  
    confusion_matrix,  
    accuracy_score,  
    precision_score,  
    recall_score,  
    f1_score,  
    classification_report  
)
```

```
# -----  
# 1) Load data  
# -----  
# Ensure diabetes.csv is in the working directory  
df = pd.read_csv("diabetes.csv")  
print("First 5 rows:\n", df.head(), "\n")  
print("Shape:", df.shape)
```

```
# -----
```

```
# 2) Features / target  
# -----  
X = df.drop("Outcome", axis=1)  
y = df["Outcome"]  
  
# -----  
# 3) Train-test split  
# -----  
X_train, X_test, y_train, y_test = train_test_split(  
    X, y,  
    test_size=0.20,  
    random_state=42,  
    stratify=y      # keep class balance  
)  
  
# -----  
# 4) Scale features (important for KNN)  
# -----  
scaler = StandardScaler()  
X_train = scaler.fit_transform(X_train) # fit on train, transform train  
X_test = scaler.transform(X_test)     # transform test with train stats  
  
# -----  
# 5) Build & train KNN  
# -----  
knn = KNeighborsClassifier(n_neighbors=5) # try 3,5,7,... later  
knn.fit(X_train, y_train)  
  
# -----
```

```

# 6) Predict
#
y_pred = knn.predict(X_test)

#
# -----
# 7) Metrics
#
cm = confusion_matrix(y_test, y_pred)

accuracy = accuracy_score(y_test, y_pred)
error_rate = 1 - accuracy
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)

print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred,
zero_division=0))
print(f"Accuracy : {accuracy:.4f}")
print(f"Error : {error_rate:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall : {recall:.4f}")
print(f"F1-score : {f1:.4f}")

# for plotting convenience (names you used earlier)
acc, prec, rec, err = accuracy, precision, recall, error_rate

#
# -----
# 8) Plots

```

```
# -----
sns.set(style="whitegrid")

# Confusion matrix heatmap
plt.figure(figsize=(4,3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
            cbar=False, square=True)
plt.title("Confusion Matrix (KNN)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# Bar plot of metrics
plt.figure(figsize=(5,3))
sns.barplot(x=["Accuracy", "Precision", "Recall", "Error"],
            y=[acc, prec, rec, err])
plt.ylim(0, 1)
plt.title("KNN Metrics")
plt.ylabel("Score")
plt.tight_layout()
plt.show()
```

## Confusion Matrix

The confusion matrix compares **Actual Output** vs **Predicted Output**.

**Predicted: Positive (1) Predicted: Negative (0)**

**Actual: Positive (1)** TP (True Positive)      FN (False Negative)

**Actual: Negative (0)** FP (False Positive)      TN (True Negative)

**Meaning of Terms:**

Term	Meaning
<b>TP (True Positive)</b>	Model correctly predicted <b>Diabetes</b>
<b>TN (True Negative)</b>	Model correctly predicted <b>No Diabetes</b>
<b>FP (False Positive)</b>	Model predicted Diabetes, but patient is <b>not</b> diabetic (Type-1 Error)
<b>FN (False Negative)</b>	Model predicted No Diabetes, but patient <b>actually has diabetes</b> (Type-2 Error – more dangerous in medical cases)

---

## Accuracy

Measures **how often the model is correct**.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

**Interpretation:**

Higher accuracy = **better overall performance**.

---

## Error Rate

Measures **how often the model is wrong**.

$$\text{Error Rate} = 1 - \text{Accuracy}$$

**Interpretation:**

Lower error rate = Model predicts less incorrectly.

---

## Precision

Out of all people the model **predicted as diabetic**, how many **are actually diabetic**?

$$\text{Precision} = \frac{TP}{TP + FP}$$

### Interpretation:

High precision = fewer **false alarms**.

Use case: When **false positives are dangerous** (e.g., sending alerts, spam detection).

---

## Recall (Sensitivity / True Positive Rate)

Out of all **actual diabetic patients**, how many were **correctly detected**?

$$\text{Recall} = \frac{TP}{TP + FN}$$

### Interpretation:

High Recall = fewer **missed diabetic patients**.

Use case: **Medical systems must maximize recall**.

---

## Summary Table (Say This in Viva)

Metric	Formula	Meaning
Accuracy	$(TP + TN) / (TP + TN + FP + FN)$	Model correctness
Error Rate	$1 - \text{Accuracy}$	Incorrect predictions
Precision	$TP / (TP + FP)$	Correct positive predictions
Recall	$TP / (TP + FN)$	Ability to detect positive cases

---

## Mini Example to Understand (Very Important for Viva)

Suppose:

TP = 50 (Correct Diabetic)

TN = 40 (Correct Non-Diabetic)

FP = 10 (Wrongly predicted Diabetic)

FN = 5 (Missed Diabetic)

### Accuracy:

$$\frac{50 + 40}{50 + 40 + 10 + 5} = \frac{90}{105} = 0.857 = 85.7\%$$

### Error Rate:

$$1 - 0.857 = 0.143 = 14.3\%$$

### Precision:

$$\frac{50}{50 + 10} = \frac{50}{60} = 0.833 = 83.3\%$$

### Recall:

$$\frac{50}{50 + 5} = \frac{50}{55} = 0.909 = 90.9\%$$

---

## 紫色 One-line Viva Answers

Question	Answer
What is Precision?	Out of predicted positive samples, how many are actually positive.
What is Recall?	Out of actual positive cases, how many were correctly detected.
What is more important in medical diagnosis?	Recall, because missing a patient (FN) is dangerous.

## Imports

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns
```

- import pandas as pd — loads **pandas** and aliases it to pd for dataframes.
- import numpy as np — loads **NumPy** for array/math utilities.
- import matplotlib.pyplot as plt — plotting library, alias plt.
- import seaborn as sns — higher-level plotting on top of matplotlib.

```
from sklearn.model_selection import train_test_split  
    • pulls in a utility to split data into train/test subsets.
```

```
from sklearn.preprocessing import StandardScaler  
    • scaler that standardizes features to mean 0 and std 1 (crucial for KNN).
```

```
from sklearn.neighbors import KNeighborsClassifier  
    • the K-Nearest Neighbors classifier.
```

```
from sklearn.metrics import (  
    confusion_matrix,  
    accuracy_score,  
    precision_score,  
    recall_score,  
    f1_score,  
    classification_report  
)
```

- imports evaluation metrics and a summary report.

---

## Load data

```
df = pd.read_csv("diabetes.csv")  
    • reads the CSV file into a pandas DataFrame named df.  
print("First 5 rows:\n", df.head(), "\n")
```

- prints the first 5 rows to quickly inspect columns/values.

```
print("Shape:", df.shape)
```

- prints the dataset size as (rows, columns).
- 

## Split features and target

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

- creates X by **dropping** the Outcome column (all input features).
- axis=1 means “drop a **column**”.

```
y = df["Outcome"]
```

- selects the Outcome column as the **target** vector y (0/1).
- 

## Train–test split

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y,
```

```
    test_size=0.20,
```

```
    random_state=42,
```

```
    stratify=y
```

```
)
```

- splits X and y into training and testing sets.
  - test\_size=0.20 → 20% test, 80% train.
  - random\_state=42 → deterministic split (reproducible).
  - stratify=y → keeps the **class ratio** similar in train and test.
- 

## Scale features

```
scaler = StandardScaler()
```

- creates a scaler object.

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

- **fit**: compute mean/std on **training** features.
- **transform**: apply standardization to training data.

- returns a NumPy array, assigned back to `X_train`.

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

- standardizes test features using **training** mean/std (prevents leakage).
- 

## Build & train KNN

```
knn = KNeighborsClassifier(n_neighbors=5)
```

- creates a KNN classifier that looks at the **5 nearest** neighbors.

```
knn.fit(X_train, y_train)
```

- “trains” the model: KNN stores the training data internally.
- 

## Predict

```
y_pred = knn.predict(X_test)
```

- predicts class labels (0/1) for the test set using neighbor votes.
- 

## Metrics

```
cm = confusion_matrix(y_test, y_pred)
```

- builds a  $2 \times 2$  matrix:
  - $[0,0]=\text{TN}$ ,  $[0,1]=\text{FP}$ ,  $[1,0]=\text{FN}$ ,  $[1,1]=\text{TP}$ .

```
accuracy = accuracy_score(y_test, y_pred)
```

- overall correctness:  $(\text{TP}+\text{TN}) / \text{total}$ .

```
error_rate = 1 - accuracy
```

- fraction wrong.

```
precision = precision_score(y_test, y_pred, zero_division=0)
```

- of predicted **positives**, how many are truly positive:  $\text{TP}/(\text{TP}+\text{FP})$ .
- `zero_division=0` avoids errors if  $\text{TP}+\text{FP}$  is zero (returns 0).

```
recall = recall_score(y_test, y_pred, zero_division=0)
```

- of actual **positives**, how many were found:  $\text{TP}/(\text{TP}+\text{FN})$ .
- `zero_division=0` similarly guards divide-by-zero.

```
f1 = f1_score(y_test, y_pred, zero_division=0)
```

- harmonic mean of precision & recall:  $2PR/(P+R)$ .
- 

## Print results

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

- prints the matrix values.

```
print("\nClassification Report:\n", classification_report(y_test, y_pred,  
zero_division=0))
```

- per-class precision/recall/F1 and support (counts).

```
print(f'Accuracy : {accuracy:.4f}')
```

```
print(f'Error   : {error_rate:.4f}')
```

```
print(f'Precision: {precision:.4f}')
```

```
print(f'Recall  : {recall:.4f}')
```

```
print(f'F1-score : {f1:.4f}')
```

- formatted to **4 decimal places**.

```
acc, prec, rec, err = accuracy, precision, recall, error_rate
```

- convenience aliases for plotting labels later.
- 

## Plot settings

```
sns.set(style="whitegrid")
```

- sets Seaborn's visual style to a clean white grid.
- 

## Confusion matrix heatmap

```
plt.figure(figsize=(4,3))
```

- starts a new figure  $4 \times 3$  inches.

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
```

```
    cbar=False, square=True)
```

- heatmap visualizes cm.
- annot=True writes the numbers in cells.
- fmt='d' formats them as integers.

- `cmap='Greens'` green color palette.
- `cbar=False` hides color bar.
- `square=True` makes cells square.

```
plt.title("Confusion Matrix (KNN)")

plt.xlabel("Predicted")
plt.ylabel("Actual")
    • axis labels and title.

plt.tight_layout()
    • adjusts spacing so labels don't overlap.

plt.show()
    • renders the heatmap.
```

---

## Bar plot of metrics

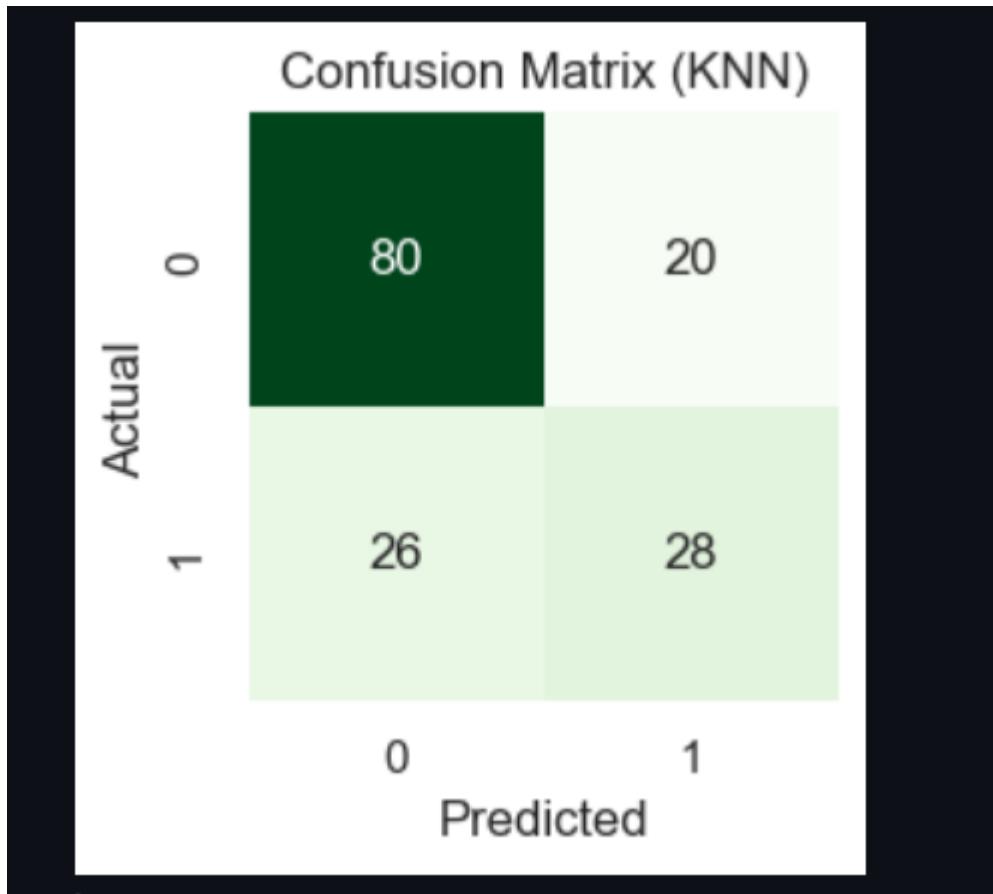
```
plt.figure(figsize=(5,3))
    • new figure 5×3 inches.

sns.barplot(x=["Accuracy", "Precision", "Recall", "Error"],
            y=[acc, prec, rec, err])
    • draws bars for the four metric values with the given labels.

plt.ylim(0, 1)
    • y-axis from 0 to 1 since metrics are proportions.

plt.title("KNN Metrics")
plt.ylabel("Score")
    • title and y-axis label.

plt.tight_layout()
plt.show()
    • tidy spacing and display the bar chart.
```



**Confusion Matrix (KNN)**

		Predicted	
		0	1
Actual	0	80	20
	1	26	28

### How to Explain:

Sir/Madam, this is the **confusion matrix** for my **K-Nearest Neighbors (KNN)** classification model.

The dataset contains two classes:

- **0 → No Diabetes**
- **1 → Diabetes**

The confusion matrix compares the **actual outcomes** with the **model's predictions**.

---

## Cell-wise Explanation

	<b>Value Meaning</b>	<b>Explanation</b>
<b>80</b>	True Negative (TN)	The model correctly predicted <b>No Diabetes</b> 80 times.
<b>20</b>	False Positive (FP)	The model predicted <b>Diabetes</b> , but the person actually <b>did not</b> have diabetes <b>20 times</b> .
<b>26</b>	False Negative (FN)	The model predicted <b>No Diabetes</b> , but the person actually <b>had diabetes 26 times</b> .
<b>28</b>	True Positive (TP)	The model correctly predicted <b>Diabetes</b> 28 times.

---

## How to Interpret Performance

- **TN (80)** is high → The model is good at identifying patients **without** diabetes.
  - **TP (28)** is moderate → The model can detect diabetic patients, but **not perfectly**.
  - **FN (26)** is a concern → Some diabetic patients were predicted as non-diabetic. In medical applications, this is risky.
  - **FP (20)** also exists → Patients may be incorrectly alarmed.
- 

## Important Metrics (If Examiner Asks)

<b>Metric</b>	<b>Formula</b>	<b>Value Explanation</b>
<b>Accuracy</b>	$(TP + TN) / \text{Total}$	Correct predictions out of all predictions.
<b>Precision</b>	$TP / (TP + FP)$	When model predicts diabetes, how many are actually diabetic.
<b>Recall (Sensitivity)</b>	$TP / (TP + FN)$	How well the model identifies actual diabetic patients. Important in medical diagnosis.

---

## How to Explain Why Recall Matters Here

*(Say this if examiner asks importance)*

In medical diagnosis, **Recall** is very important because we want to **catch as many actual diabetic patients as possible**.

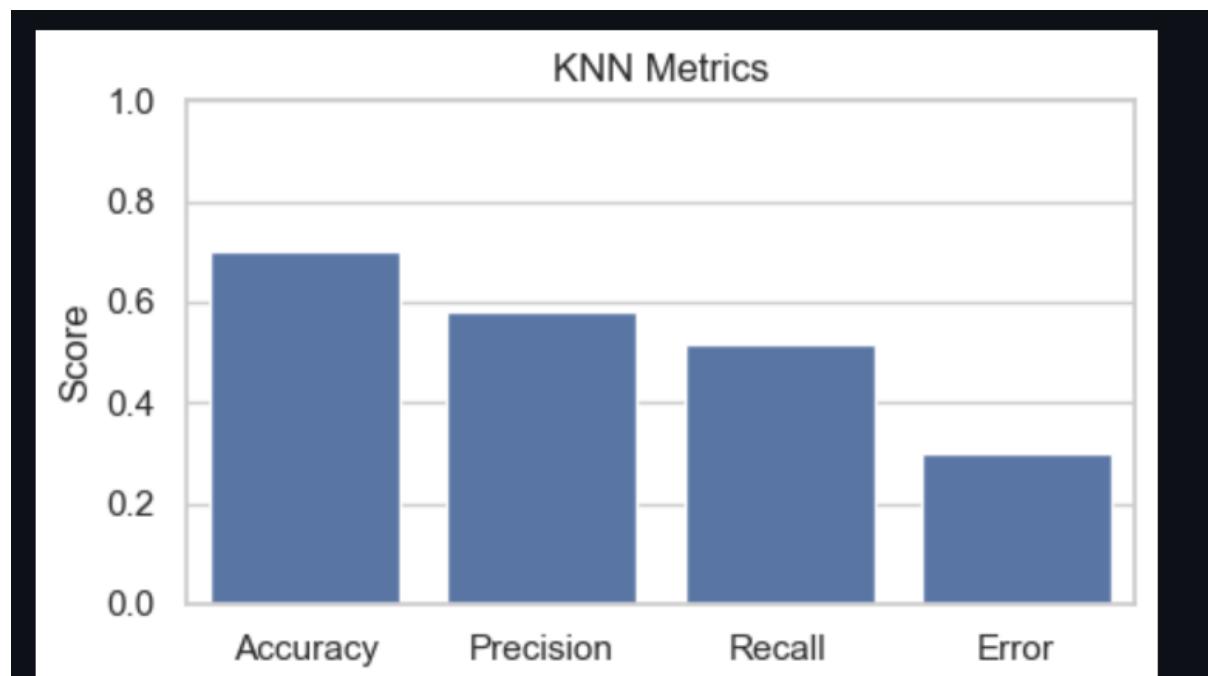
A **False Negative (FN)** means a diabetic person is predicted as healthy, which can be **dangerous**.

So we try to **minimize FN** and increase recall.

---

### One-Line Summary to Say at the End

Overall, the model performs reasonably well, especially for non-diabetic cases. However, the recall value suggests that the model sometimes **misses diabetic patients**, so in real medical scenarios, the model should be improved or combined with other methods.



### How to Explain the Graph (Simple + Professional)

This bar chart shows the **performance metrics** of the K-Nearest Neighbors (KNN) classification model that I applied to the diabetes dataset.

The **four metrics** shown are:

Metric	What it Means	Value Interpretation
<b>Accuracy</b>	Overall correctness of the model	About <b>70%</b> , meaning the model predicts correctly 70 out of 100 cases.

Metric	What it Means	Value Interpretation
Precision	Reliability of positive predictions	Around <b>58%</b> , meaning when the model predicts diabetes, it is correct <b>58% of the time</b> .
Recall	Ability to detect actual diabetic patients	Around <b>51%</b> , meaning it correctly identifies only <b>half</b> of actual diabetes cases.
Error Rate	How often the model makes mistakes	About <b>30%</b> , meaning 3 out of every 10 predictions are <b>incorrect</b> .

---

### Key Interpretation (Important for Viva)

- The **accuracy** is moderate → the model is performing **fairly well overall**.
  - **Precision is lower** → sometimes the model predicts diabetes even when the person does **not** actually have diabetes (false alarms).
  - **Recall is even lower** → this means the model is **missing several diabetic patients**, which is **risky** in real medical diagnosis.
  - The **error rate is not zero**, showing improvement is possible.
- 

### One Strong Conclusion Sentence (Say This at the End)

So, while the KNN model shows reasonable accuracy, its **recall value suggests that there is a chance of missing actual diabetic cases**, therefore in medical contexts, this model should be improved further or combined with other diagnostic approaches.

---

### If You Want a Short 10-Second Answer

The accuracy is around 70%, precision is 58%, and recall is 51%. This means the model works okay but is **not perfect**, especially in detecting diabetic patients. The error rate of 30% shows that the model needs improvement.



## Assignment 3 — Gradient Descent (40 Q&A)

### 1. What is Gradient Descent (GD)?

An iterative optimization method that updates parameters in the **negative gradient** direction to minimize a function.

### 2. Why is GD used?

To minimize loss/cost functions and learn model parameters (e.g., in regression, neural nets).

### 3. GD update rule?

$$x_{t+1} = x_t - \alpha \nabla f(x_t), \text{ where } \alpha \text{ is learning rate.}$$

### 4. Learning rate ( $\alpha$ ) role?

Controls step size—too big: diverge; too small: slow.

### 5. Too high vs too low $\alpha$ ?

High → oscillation/divergence; Low → very slow convergence.

### 6. Stopping criteria?

Max iterations,  $\| \nabla f(x) \| < \epsilon$ , or  $| x_{t+1} - x_t | < \text{tol}$ , or small loss change.

### 7. Why negative gradient?

Because gradient points to **steepest ascent**; we want steepest **descent**.

### 8. For $f(x) = (x + 3)^2$ , derivative?

$$f'(x) = 2(x + 3).$$

### 9. Where is the minimum for $(x + 3)^2$ ?

At  $x = -3$  since  $f'(x) = 0 \Rightarrow x = -3$ ; convex quadratic → global minimum.

### 10. First 3 steps from $x_0 = 2$ , $\alpha$ = 0.1?

$$x_1 = 1.0,$$

$$x_2 =$$

$$0.2, \quad x_3 = -0.44 (\text{using } x_{t+1} = x_t - 0.2(x_t + 3)).$$

### 11. Geometric intuition on a parabola?

You take steps downhill; steps shrink near the bottom as slope → 0.

### 12. What is convexity?

Function with a single basin; any local minimum is global—GD converges reliably.

### 13. Variants of GD?

Batch GD, Stochastic GD (SGD), Mini-batch GD.

### 14. Batch vs SGD?

Batch: stable but slower per epoch; SGD: noisy but fast updates and can escape shallow minima.

## 15. What is a cost/loss function?

A scalar measure of model error (e.g., MSE, cross-entropy) minimized by training.

## 16. Why derivatives?

They give the direction/rate of change to reduce the function fastest.

## 17. Step-size scheduling?

Decrease  $\alpha$  over time (e.g.,  $\alpha_t = \alpha_0/(1 + kt)$ ) to stabilize convergence.

## 18. Vanishing gradients?

Gradients become tiny → learning stalls (common in deep nets with certain activations).

## 19. GD vs Newton's Method?

Newton uses Hessian (second derivative) for quadratic convergence but costly; GD uses only first derivatives.

## 20. When can GD get stuck?

In local minima or plateaus/saddle points for non-convex functions.

## 21. Saddle point?

Point where gradient is 0 but not a min/max; GD may stall or pass slowly.

## 22. Momentum methods?

Add velocity term to smooth/accelerate (e.g., Momentum, Nesterov).

## 23. Adaptive methods?

AdaGrad/RMSProp/Adam scale learning rates per parameter for faster, robust training.

## 24. Why feature scaling helps GD?

Reduces condition number; contours less elongated → faster, more stable descent.

## 25. Convergence tolerance (tol)?

Threshold to stop when updates/gradients are sufficiently small.

## 26. Why gradient $\approx 0$ near minimum?

First-order optimality condition:  $\nabla f(x^*) = 0$ .

## 27. Does initial point matter here?

For convex  $(x + 3)^2$ : no (only speed). For non-convex: may affect the minimum found.

## 28. Error recursion for this quadratic with $\alpha=0.1$ ?

Let  $e_t = x_t + 3$ . Then  $e_{t+1} = (1 - 2\alpha)e_t = 0.8e_t \rightarrow$  geometric decay.

## 29. Lipschitz constant of gradient for $(x + 3)^2$ ?

$L = |f''(x)| = 2$ . Converges if  $0 < \alpha < 2/L = 1$ .

### 30. Condition for GD convergence on quadratics?

$0 < \alpha < 2/L$ , where  $L$  is gradient Lipschitz constant.

### 31. Detect divergence programmatically?

Monitor if loss increases repeatedly or  $|x|$  explodes; reduce  $\alpha$ /restart.

### 32. Noisy gradients effect?

Slower, jittery convergence; may help escape shallow minima (SGD effect).

### 33. Line search?

Choose  $\alpha$  per step to sufficiently decrease loss (e.g., Armijo/Wolfe).

### 34. Log (x, grad, f(x)) each epoch?

Print or append to lists inside loop; e.g., `hist.append((t,x,grad(x),f(x)))`.

### 35. Visualize steps?

Plot function curve plus scatter of successive  $x_t$  points; optionally arrows.

### 36. Per-iteration cost (1D vs nD)?

1D:  $O(1)$  for grad; nD:  $O(d)$  to compute gradient over  $d$  features.

### 37. Why use tol and max\_iter both?

tol for convergence; max\_iter as safety cap if tol not reached.

### 38. Extend to multi-dimensional GD?

Use vector  $\mathbf{x}$  and gradient  $\nabla f(\mathbf{x})$ ; same update rule.

### 39. Gradient ascent change?

Use  $x_{t+1} = x_t + \alpha \nabla f(x_t)$  to maximize.

### 40. Two ML uses of GD?

Training linear/logistic regression; training neural networks.



## Assignment 4 — k-NN on Diabetes Dataset (40 Q&A)

### 1. What is k-NN?

A **lazy, non-parametric, instance-based** classifier using proximity to neighbors.

### 2. Why “lazy learner”?

No explicit training; stores data and computes at prediction time.

### 3. Instance-based meaning?

Decisions are based on comparisons with stored instances (neighbors).

### 4. Effect of k=1 vs k=15?

$k=1$ : low bias, high variance (noisy).  $k=15$ : smoother, higher bias, lower variance.

**5. Why odd k in binary classes?**

To reduce ties in majority voting.

**6. Why scale features?**

Distance-based method; unscaled features with larger ranges dominate.

**7. Common distance metrics?**

Euclidean, Manhattan (L1), Minkowski, (sometimes cosine for vectors).

**8. Euclidean distance formula?**

$$\sqrt{\sum_i (x_i - y_i)^2}.$$

**9. Prediction time complexity (naïve)?**

$O(n \cdot d)$  per query (n points, d features).

**10. How to pick optimal k?**

Use validation/cross-validation; plot accuracy vs k.

**11. Confusion matrix terms?**

TP, TN, FP, FN mapping actual vs predicted classes.

**12. Accuracy formula?**

$$(TP + TN) / (TP + TN + FP + FN).$$

**13. Error rate formula?**

1 – Accuracy.

**14. Precision formula?**

$$TP / (TP + FP).$$

**15. Recall (Sensitivity) formula?**

$$TP / (TP + FN).$$

**16. In medical diagnosis, prioritize?**

**Recall**, to minimize missed positive cases (FN).

**17. Class imbalance effect on accuracy?**

Can inflate accuracy while minority class suffers (poor recall/precision).

**18. Effect of StandardScaler on k-NN?**

Balances feature scales → typically improves performance.

**19. Why train/test split?**

Unbiased performance estimate; prevents overfitting to training data.

**20. random\_state purpose?**

Reproducibility of split.

**21. Handling missing values?**

Impute (mean/median for numeric), or domain-specific strategies.

## 22. Zero-variance features?

Remove—they add no information and distort distance.

## 23. Outliers handling?

Robust scaling, capping, or using distance weights to reduce their influence.

## 24. Weighted k-NN?

Neighbors weighted by inverse distance: closer neighbors have more say.

## 25. Change to Manhattan in sklearn?

KNeighborsClassifier(metric='manhattan').

## 26. Beyond precision/recall?

F1-score, ROC-AUC, PR-AUC, specificity, confusion heatmap.

## 27. F1-score formula?

$$2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

## 28. ROC and AUC?

ROC: TPR vs FPR curve across thresholds; AUC: area under ROC—higher is better.

## 29. Multi-class with k-NN?

Vote among neighbors' labels; majority class wins.

## 30. Pros of k-NN?

Simple, no training, works for non-linear boundaries, decent baseline.

## 31. Cons of k-NN?

Slow prediction, high memory, sensitive to scale/outliers/irrelevant features.

## 32. KD-Tree/Ball Tree?

Spatial indexes to speed neighbor search (good for lower dimensions).

## 33. Cross-validation for k?

Grid over k; use K-fold CV to pick k with best mean score.

## 34. Overfitting/underfitting diagnosis?

k small → overfit; k large → underfit; compare train vs test metrics.

## 35. Standardization vs Min-Max?

Standardization (z-score) robust to outliers' scale; Min-Max keeps [0,1] range—choose per data.

## 36. Most informative diabetes features (intuition)?

Glucose, BMI, Age, Insulin, DiabetesPedigreeFunction often correlate with Outcome.

**37. Improve recall if low?**

Lower k, use weights='distance', adjust class weights/thresholds, resample minority.

**38. Print classification report in sklearn?**

```
from sklearn.metrics import classification_report  
print(classification_report(y_test,y_pred))
```

**39. Plot labeled confusion matrix?**

Use ConfusionMatrixDisplay.from\_predictions(y\_test, y\_pred).

**40. Real-world uses of k-NN?**

Recommenders, OCR, anomaly detection, medical triage, credit risk, image search.