Step 1: Understand the Numerical Method

Study the theory behind the numerical method:

What problem does it solve? (e.g., optimization, root-finding, linear system solving)

What are the steps involved? (e.g., update rule of Gradient Descent, stopping criteria)

Step 2: Select the Dataset

Use the dataset provided in the assignment.

Load the dataset using:

pandas for CSV/Excel

sklearn.datasets for built-in datasets

opencv, numpy, or skimage for image data

python

```
import pandas as pd

df = pd. read_sv("your_dataset.csv")
```

Step 3: Preprocess the Dataset

Handle missing values

Normalize/scale data

Encode categorical variables if needed

python

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler

X_scaled = scaler. fit_transform (X)
```

Step 4: Define the ML Task

Clearly define:

Whether the task is regression or classification

Which features are inputs (X) and which is the output (y)

What model you're trying to optimize (linear, logistic, neural net, etc.)

Step 5: Apply the Numerical Method

Use the assigned method to solve or optimize the model.

Examples by method:

Method

ML Use Case

What to Code

Gradient Descent

Linear Regression, Logistic Regression

Derive cost function, compute gradients, update weights

SGD

Classification (logistic), deep nets

Update weights using one sample at a time

Newton's Method

Logistic Regression, root of gradients

Use second derivative (Hessian), update rule

Conjugate Gradient

Large linear regression

Use CG to solve normal equations

L-BFGS|

Logistic regression

Use scipy.optimize.minimize or

PyTorch/TensorFlow

LU / QR Decomposition

Solve AX = b in regression

Decompose and solve manually or with NumPy

Trapezoidal Rule

Integrate cost function or area under curve

Apply integration formula to f(x) values

Bisection/Newton-Raphson

Find root of cost function or threshold

Use method iteratively to find root

Step 6: Implement from Scratch (Optional)

If assigned, write the algorithm without libraries for better understanding.

E.g., manual Gradient Descent without scikit-learn.

Step 7: Visualize the Process

Plot loss function vs iterations

Plot decision boundary or regression line

Show convergence of error

python

```
import matplotlib.pyplot as plt plt. plot(range(epochs), losses)
plt. title("Loss over Iterations")
plt.xlabel ("Epoch")|
plt.ylabel ("Loss")
plt.show
```

Step 8: Evaluate the Model

Use appropriate metrics:

Regression: RMSE, MAE, R?

Classification: Accuracy, Precision, Recall, F1, AUC

python

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error (y_true, y_pred)
```

Step 9: Compare with Library/Sklearn Result (Optional)

Compare your method's results with built-in functions like:

python

```
from sklearn. linear_model import LinearRegression
```

Step 10: Report Writing

Each student should submit a brief report with the following:

Report Format:

markdown

Title

Objective

Dataset Source & Description

Method Description (Math)

Step-by-step Code Explanation

Plots & Results

Evaluation

Comparison (if any)

Conclusion

References

Optional Tools

Python: Main language (NumPy, Pandas, Scikit-learn, Matplotlib)|

Google Colab or Jupyter Notebook: For reproducibility

Latex (for report) or Word document with equations typed clearly

Example: Gradient Descent for Linear Regression

text

Objective: Use Gradient Descent to fit a line predicting house prices

Load House Price Dataset

Extract features (X) and label (y)

Normalize data

Define cost function and its gradient

Initialize weights, run Gradient Descent loop

Plot cost vs iteration

Evaluate RMSE, plot prediction vs true value

Compare with sklearn's LinearRegression)