Linear Regression

```
# ==========================
# LINEAR REGRESSION FULL CODE
# ==========================

# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score

# --------------------------
# 1. DATA LOADING
# --------------------------
# Example: data.csv should have columns like ['feature1', 'feature2', 'target']
data = pd.read_csv('/content/customers.csv')

print("First 5 rows of data:")
print(data.head())

# --------------------------
# 2. DATA CLEANING
# --------------------------
# Remove duplicate rows
data = data.drop_duplicates()

# Check for missing values
print("\nMissing values:\n", data.isnull().sum())

# Fill missing numeric values with mean
data = data.fillna(data.mean())

# --------------------------
# 3. FEATURE SELECTION
# --------------------------
# Independent (X) and dependent (y) variables
X = data.drop('Spending_Score', axis=1)
y = data['Spending_Score']

# --------------------------
# 4. DATA PREPROCESSING
# --------------------------
# Standardize features (scaling)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# --------------------------
# 5. TRAIN-TEST SPLIT
# --------------------------
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# --------------------------
# 6. MODEL TRAINING
# --------------------------
model = LinearRegression()
model.fit(X_train, y_train)

# --------------------------
# 7. PREDICTION
# --------------------------
y_pred = model.predict(X_test)

# --------------------------
# 8. EVALUATION
# --------------------------
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print("Mean Squared Error (MSE):", mse)
print("R² Score:", r2)

# --------------------------
# 9. SAMPLE OUTPUT
# --------------------------
print("\nActual vs Predicted:")
```

```
result = pd.DataFrame({'Actual': y_test.values, 'Predicted': y_pred})
print(result.head())
```

```
First 5 rows of data:
   CustomerID  Age  Income  Spending_Score
0           1   19      15              39
1           2   21      15              81
2           3   20      16               6
3           4   23      16              77
4           5   31      17              40

Missing values:
 CustomerID       0
Age             0
Income          0
Spending_Score    0
dtype: int64

Model Evaluation:
Mean Squared Error (MSE): 453.8031565430533
R² Score: -0.6167558460674356

Actual vs Predicted:
   Actual  Predicted
0      39  12.269581
1      66  95.747519
2      79  92.777033
3      81  75.903353
```

Logistic Regression

```
# =====================================
# LOGISTIC REGRESSION FULL PRACTICAL CODE
# =====================================

# Import libraries
import pandas as pd
import numpy as np
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

# ---------------------------
# 1. DATA LOADING
# ---------------------------
# Example CSV should contain columns like ['feature1', 'feature2', ..., 'target']
# where 'target' contains binary values (0/1 or Yes/No)
data = pd.read_csv('/content/customers.csv')

print("First 5 rows of dataset:")
print(data.head())

# ---------------------------
# 2. DATA CLEANING
# ---------------------------
# Remove duplicates
data = data.drop_duplicates()

# Handle missing values
print("\nMissing values:\n", data.isnull().sum())
data = data.fillna(data.mean())

# ---------------------------
# 3. FEATURE SELECTION
# ---------------------------
X = data.drop('Spending_Score', axis=1)
y = data['Spending_Score']

# ---------------------------
# 4. DATA PREPROCESSING
# ---------------------------
# Scale features for better model performance
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---------------------------
# 5. TRAIN-TEST SPLIT
# ---------------------------
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

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

```
# 6. MODEL TRAINING
# ---------------------------
model = LogisticRegression()
model.fit(X_train, y_train)

# ---------------------------
# 7. PREDICTION
# ---------------------------
y_pred = model.predict(X_test)

# ---------------------------
# 8. EVALUATION
# ---------------------------
acc = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

print("\nModel Evaluation:")
print("Accuracy:", acc)
print("Confusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

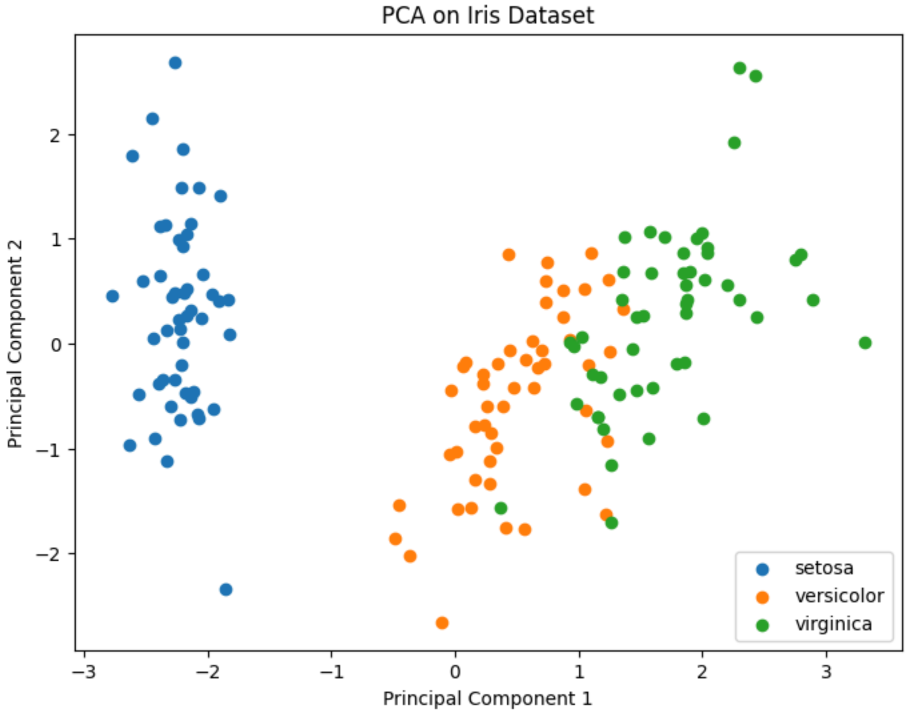
```
First 5 rows of dataset:
   CustomerID  Age  Income  Spending_Score
0           1   19      15              39
1           2   21      15              81
2           3   20      16               6
3           4   23      16              77
4           5   31      17              40

Missing values:
 CustomerID        0
Age               0
Income            0
Spending_Score    0
dtype: int64

Model Evaluation:
Accuracy: 0.0
Confusion Matrix:
 [[0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0]
 [0 0 0 0 0 0 1]
 [0 0 0 0 0 0 0]
 [0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]]

Classification Report:
               precision    recall  f1-score   support

           6       0.00      0.00      0.00       0.0
          39       0.00      0.00      0.00       1.0
          66       0.00      0.00      0.00       1.0
          77       0.00      0.00      0.00       0.0
          79       0.00      0.00      0.00       1.0
          81       0.00      0.00      0.00       1.0
          98       0.00      0.00      0.00       0.0

    accuracy                           0.00       4.0
   macro avg       0.00      0.00      0.00       4.0
weighted avg       0.00      0.00      0.00       4.0

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-de
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Recall is ill-defir
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

PCA with IRIS Dataset

```
# ====================================
# PCA with IRIS DATASET
# ====================================

# Import libraries
import pandas as pd
from sklearn.datasets import load_iris
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# ---------------------------
# 1. LOAD DATA
# ---------------------------
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Convert to DataFrame for easy handling
data = pd.DataFrame(X, columns=iris.feature_names)
data['target'] = y
print("First 5 rows of Iris Data:")
print(data.head())

# ---------------------------
# 2. DATA PREPROCESSING
# ---------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---------------------------
# 3. APPLY PCA
# ---------------------------
# Reduce from 4 features → 2 principal components
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)

# Create new DataFrame for PCA output
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['target'] = y

# ---------------------------
# 4. VARIANCE EXPLAINED
# ---------------------------
print("\nExplained Variance Ratio:", pca.explained_variance_ratio_)
print("Total Variance Captured:", sum(pca.explained_variance_ratio_))

# ---------------------------
# 5. VISUALIZATION
# ---------------------------
plt.figure(figsize=(8,6))
for i, target_name in enumerate(target_names):
    plt.scatter(pca_df.loc[pca_df['target'] == i, 'PC1'],
                pca_df.loc[pca_df['target'] == i, 'PC2'],
                label=target_name)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Iris Dataset')
plt.legend()
plt.show()
```

```
First 5 rows of Iris Data:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0

Explained Variance Ratio: [0.72962445 0.22850762]
Total Variance Captured: 0.9581320720000166
```

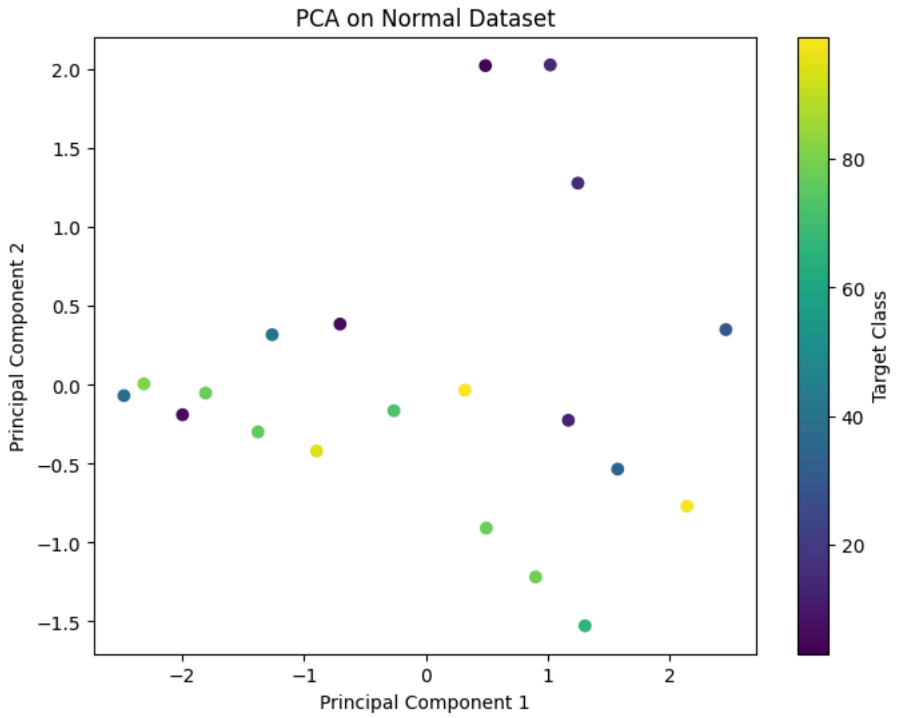


PCA with NORMAL Dataset

```
# ====================================
# PCA on a Normal Dataset (data.csv)
# ====================================

# Import libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# ---------------------------
# 1. LOAD DATA
# ---------------------------
data = pd.read_csv('/content/customers.csv')
print("First 5 rows:")
print(data.head())

# ---------------------------
# 2. DATA CLEANING
# ---------------------------
# Drop duplicates and handle missing values
data = data.drop_duplicates()
data = data.fillna(data.mean())

# ---------------------------
# 3. FEATURE SELECTION
# ---------------------------
# Assuming the last column is the target
X = data.drop('Spending_Score', axis=1)
y = data['Spending_Score']

# ---------------------------
# 4. SCALING (Important!)
```

```
# ---------------------------
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# ---------------------------
# 5. APPLY PCA
# ---------------------------
pca = PCA(n_components=2)  # Reduce to 2 dimensions for visualization
X_pca = pca.fit_transform(X_scaled)

print("\nExplained Variance Ratio:", pca.explained_variance_ratio_)
print("Total Variance Captured:", sum(pca.explained_variance_ratio_))

# ---------------------------
# 6. VISUALIZATION
# ---------------------------
plt.figure(figsize=(8,6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='viridis')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA on Normal Dataset')
plt.colorbar(label='Target Class')
plt.show()
```

```
First 5 rows:
   CustomerID  Age  Income  Spending_Score
0           1   19      15              39
1           2   21      15              81
2           3   20      16               6
3           4   23      16              77
4           5   31      17              40

Explained Variance Ratio: [0.73013048 0.26907424]
Total Variance Captured: 0.9992047211642271
```



Installation of minisom

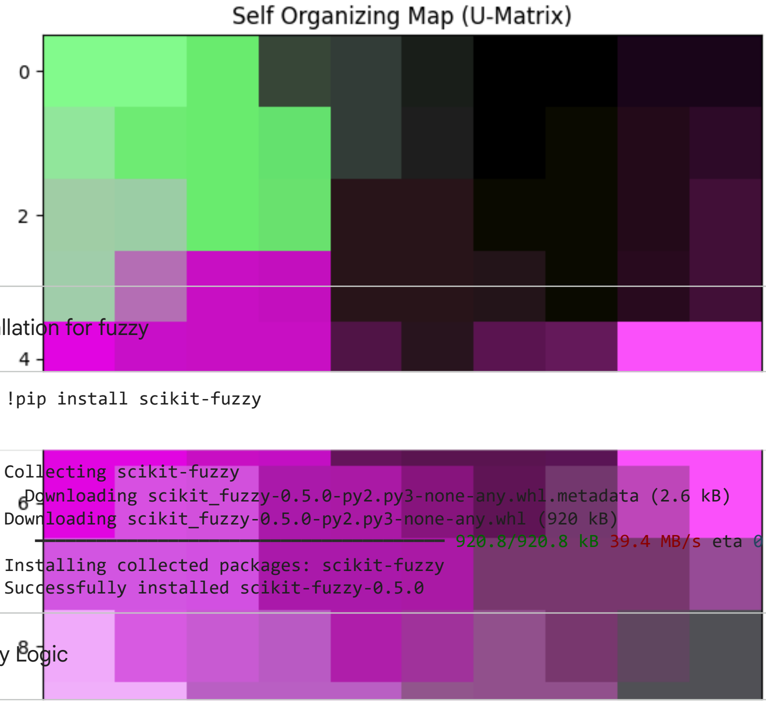
```
!pip install minisom
```

```
Collecting minisom
  Downloading minisom-2.3.5.tar.gz (12 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py) ... done
  Created wheel for minisom: filename=MiniSom-2.3.5-py3-none-any.whl size=12031 sha256=c1ea658ec87b120326b8ed133290ab4755311
  Stored in directory: /root/.cache/pip/wheels/0f/8c/a4/5b7aa56fa6ef11d536d45da775bcc5a2a1c163ff0f8f11990b
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.5
```

SELF ORGANIZING MAP (SOM) - Unsupervised

```
# =====================================
# SELF ORGANIZING MAP (SOM)
# =====================================

# Import libraries
import pandas as pd
import numpy as np
from minisom import MiniSom
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

# ---------------------------
# 1. LOAD DATA
# ---------------------------
# Example dataset (replace with your CSV)
data = pd.read_csv('/content/customers.csv')
print("First 5 rows of dataset:")
print(data.head())

# ---------------------------
# 2. DATA CLEANING
# ---------------------------
data = data.drop_duplicates()
data = data.fillna(data.mean())

# ---------------------------
# 3. FEATURE SELECTION
# ---------------------------
X = data.drop('Spending_Score', axis=1, errors='ignore')  # If target exists
scaler = MinMaxScaler(feature_range=(0, 1))
X_scaled = scaler.fit_transform(X)

# ---------------------------
# 4. INITIALIZE SOM
# ---------------------------
som = MiniSom(x=10, y=10, input_len=X_scaled.shape[1], sigma=1.0, learning_rate=0.5)
som.random_weights_init(X_scaled)
print("\nTraining the SOM...")

# ---------------------------
# 5. TRAINING
# ---------------------------
som.train_random(data=X_scaled, num_iteration=10000)
print("SOM training complete!")

# ---------------------------
# 6. VISUALIZATION
# ---------------------------
plt.figure(figsize=(7,7))
from pylab import bone, pcolor, colorbar, plot, show, legend

bone()  # background
plt.imshow(som.get_weights(), interpolation='nearest')


plt.title('Self Organizing Map (U-Matrix)')
plt.show()
```

```
First 5 rows of dataset:
   CustomerID  Age  Income  Spending_Score
0           1   19      15              39
1           2   21      15              81
2           3   20      16               6
3           4   23      16              77
4           5   31      17              40

Training the SOM...
SOM training complete!
```



Installation for fuzzy

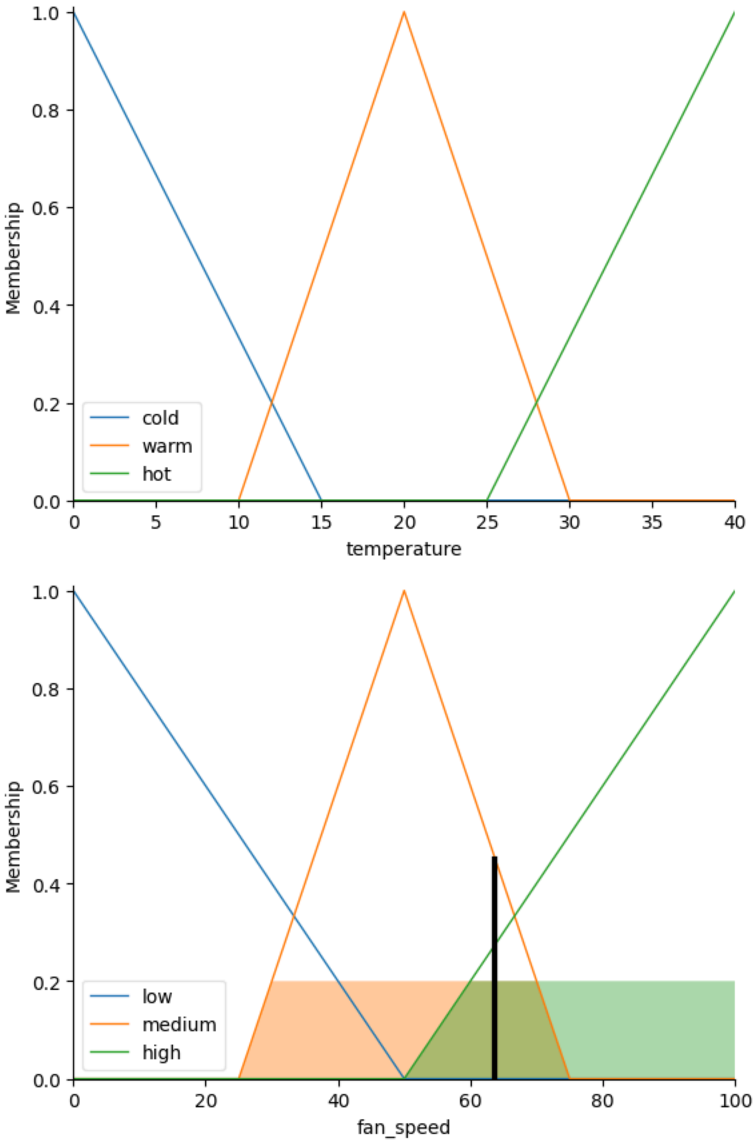
```
!pip install scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl.metadata (2.6 kB)
Downloading scikit_fuzzy-0.5.0-py2.py3-none-any.whl (920 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 920.8/920.8 kB 39.4 MB/s eta 0:00:00
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.5.0
```

Fuzzy Logic

```
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Define fuzzy variables
temperature = ctrl.Antecedent(np.arange(0, 41, 1), 'temperature')
fan_speed = ctrl.Consequent(np.arange(0, 101, 1), 'fan_speed')

# Define membership functions for Temperature
temperature['cold'] = fuzz.trimf(temperature.universe, [0, 0, 15])
temperature['warm'] = fuzz.trimf(temperature.universe, [10, 20, 30])
temperature['hot'] = fuzz.trimf(temperature.universe, [25, 40, 40])

# Define membership functions for Fan Speed
fan_speed['low'] = fuzz.trimf(fan_speed.universe, [0, 0, 50])
fan_speed['medium'] = fuzz.trimf(fan_speed.universe, [25, 50, 75])
fan_speed['high'] = fuzz.trimf(fan_speed.universe, [50, 100, 100])

# Define fuzzy rules
rule1 = ctrl.Rule(temperature['cold'], fan_speed['low'])
rule2 = ctrl.Rule(temperature['warm'], fan_speed['medium'])
rule3 = ctrl.Rule(temperature['hot'], fan_speed['high'])

# Create control system and simulation
fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan_sim = ctrl.ControlSystemSimulation(fan_ctrl)

# Input a sample value
fan_sim.input['temperature'] = 28

# Compute result
fan_sim.compute()
print("Predicted Fan Speed:", round(fan_sim.output['fan_speed'], 2))

# Visualize
temperature.view()
fan_speed.view(sim=fan_sim)
```

```
Predicted Fan Speed: 63.74
```



General Code for fuzzy logic

```
# --------- FUZZY LOGIC GENERAL CODE TEMPLATE ---------
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl

# Step 1: Define Fuzzy Variables (Inputs and Outputs)
# Change the names and ranges as per your problem
input1 = ctrl.Antecedent(np.arange(0, 11, 1), 'input1')
input2 = ctrl.Antecedent(np.arange(0, 11, 1), 'input2')  # Optional second input
output = ctrl.Consequent(np.arange(0, 26, 1), 'output')

# Step 2: Define Membership Functions for Inputs
# You can change the ranges and shapes as per your problem
input1['low'] = fuzz.trimf(input1.universe, [0, 0, 5])
input1['medium'] = fuzz.trimf(input1.universe, [0, 5, 10])
input1['high'] = fuzz.trimf(input1.universe, [5, 10, 10])

input2['low'] = fuzz.trimf(input2.universe, [0, 0, 5])
input2['medium'] = fuzz.trimf(input2.universe, [0, 5, 10])
input2['high'] = fuzz.trimf(input2.universe, [5, 10, 10])

# Step 3: Define Membership Functions for Output
output['low'] = fuzz.trimf(output.universe, [0, 0, 10])
output['medium'] = fuzz.trimf(output.universe, [5, 10, 20])
output['high'] = fuzz.trimf(output.universe, [10, 25, 25])

# Step 4: Define Fuzzy Rules
# Modify according to your logic
rule1 = ctrl.Rule(input1['low'] & input2['low'], output['low'])
rule2 = ctrl.Rule(input1['medium'] | input2['medium'], output['medium'])
rule3 = ctrl.Rule(input1['high'] & input2['high'], output['high'])
```

```
# Step 5: Create Control System and Simulation
fuzzy_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fuzzy_sim = ctrl.ControlSystemSimulation(fuzzy_ctrl)

# Step 6: Give Inputs and Compute
fuzzy_sim.input['input1'] = 6.5
fuzzy_sim.input['input2'] = 7.0

fuzzy_sim.compute()
print("Fuzzy Output:", round(fuzzy_sim.output['output'], 2))

# Step 7: Visualize Membership Functions
input1.view()
input2.view()
output.view(sim=fuzzy_sim)
```

```
# -------- GENETIC ALGORITHM (COLAB READY CODE) --------
import numpy as np
import random
import matplotlib.pyplot as plt

# Step 1: Define the fitness function
# You can change this function as per your problem
def fitness_function(x):
    # Example: maximize f(x) = x * sin(10πx) + 1
    return x * np.sin(10 * np.pi * x) + 1

# Step 2: Initialize GA parameters
pop_size = 10         # number of individuals
generations = 50      # number of generations
mutation_rate = 0.1   # mutation probability
crossover_rate = 0.8  # crossover probability
x_bounds = [0, 1]     # variable range

# Step 3: Initialize population randomly within bounds
population = np.random.uniform(low=x_bounds[0], high=x_bounds[1], size=pop_size)

# For visualization
best_fitness_per_gen = []

# Step 4: Main GA loop
for gen in range(generations):
    # 4.1 Evaluate fitness
    fitness = fitness_function(population)

    # 4.2 Selection (Roulette Wheel)
    probabilities = fitness / np.sum(fitness)
    parents = np.random.choice(population, size=pop_size, p=probabilities)

    # 4.3 Crossover (Blend Crossover)
    offspring = []
    for i in range(0, pop_size, 2):
        parent1 = parents[i]
        parent2 = parents[i+1]
        if random.random() < crossover_rate:
            alpha = random.random()
            child1 = alpha * parent1 + (1 - alpha) * parent2
            child2 = alpha * parent2 + (1 - alpha) * parent1
        else:
            child1, child2 = parent1, parent2
        offspring.append(child1)
        offspring.append(child2)
    offspring = np.array(offspring)

    # 4.4 Mutation
    for i in range(pop_size):
        if random.random() < mutation_rate:
            mutation_value = np.random.uniform(-0.1, 0.1)
            offspring[i] += mutation_value
            offspring[i] = np.clip(offspring[i], x_bounds[0], x_bounds[1])

    # Replace old population
    population = offspring

    # Track best fitness
    best_fitness_per_gen.append(np.max(fitness_function(population)))

# Step 5: Get the best solution
best_solution = population[np.argmax(fitness_function(population))]
best_fitness = np.max(fitness_function(population))
```

```
# Step 6: Print and plot results
print("✅ Best solution found:", round(best_solution, 4))
print("🏆 Best fitness value:", round(best_fitness, 4))

plt.plot(best_fitness_per_gen)
plt.title("Genetic Algorithm Progress")
plt.xlabel("Generation")
plt.ylabel("Best Fitness")
plt.grid(True)
plt.show()
```

```
✅ Best solution found: 0.4299
🏆 Best fitness value: 1.347
```