1. **Perform linear regression algorithm to predict**
    a. **The price of a used car. Dataset: Used_Car_dataset.csv**
    b. **The price of a House. Dataset: Housing_price_dataset.csv**
    c. **CO_2 Emission Dataset: FuelConsumption_dataset.csv**

Consider both univariate and multivariate cases for these problems. Elaborate gradient descent algorithm and hyper-parameter tuning for the best result with predefined convergent criteria. Display the hypothesis function and find the accuracy of the linear regression in predicting the correct price.

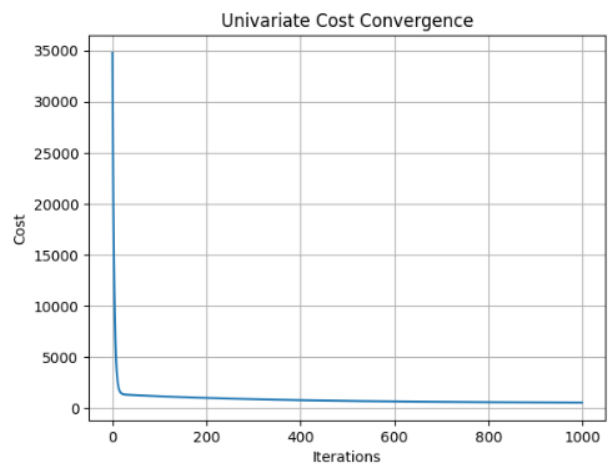
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

df = pd.read_csv('FuelConsumption.csv')
print(df.head())
```

```
   MODELYEAR   MAKE       MODEL VEHICLECLASS  ENGINESIZE  CYLINDERS  \
0       2014  ACURA         ILX      COMPACT         2.0          4
1       2014  ACURA         ILX      COMPACT         2.4          4
2       2014  ACURA  ILX HYBRID      COMPACT         1.5          4
3       2014  ACURA     MDX 4WD  SUV - SMALL         3.5          6
4       2014  ACURA     RDX AWD  SUV - SMALL         3.5          6

  TRANSMISSION FUELTYPE  FUELCONSUMPTION_CITY  FUELCONSUMPTION_HWY  \
0          AS5        Z                   9.9                  6.7
1           M6        Z                  11.2                  7.7
2          AV7        Z                   6.0                  5.8
3          AS6        Z                  12.7                  9.1
4          AS6        Z                  12.1                  8.7

   FUELCONSUMPTION_COMB  FUELCONSUMPTION_COMB_MPG  CO2EMISSIONS
0                   8.5                        33           196
1                   9.6                        29           221
2                   5.9                        48           136
3                  11.1                        25           255
4                  10.6                        27           244
```

```
# ========== STEP 2: LINEAR REGRESSION FUNCTIONS ==========
# Hypothesis: h(x) = X @ theta
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        predictions = X.dot(theta)
        error = predictions - y
        gradient = (1/m) * X.T.dot(error)
        theta -= alpha * gradient
        cost = (1/(2*m)) * np.sum(error ** 2)
        cost_history.append(cost)
        if len(cost_history) > 1 and abs(cost_history[-1] - cost_history[-2]) < 1e-6:
            break
```

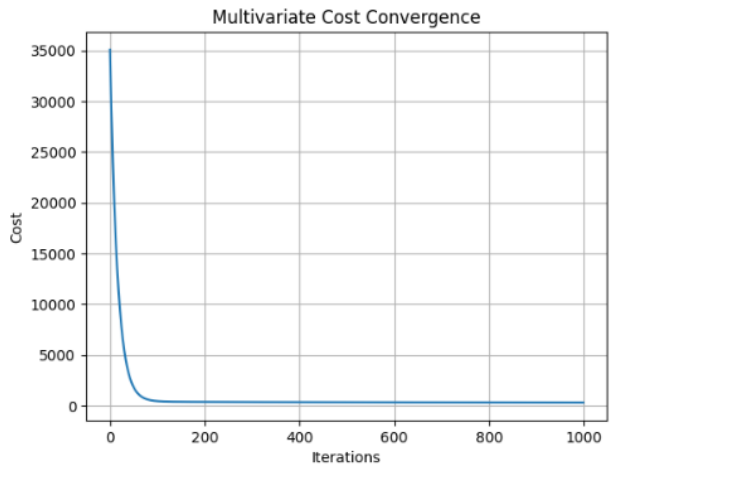
```
    return theta, cost_history


def evaluate(X, y, theta):
    predictions = X.dot(theta)
    mse = mean_squared_error(y, predictions)
    r2 = r2_score(y, predictions)
    return mse, r2


# ========== STEP 3: UNIVARIATE LINEAR REGRESSION ==========


print("\n--- Univariate Linear Regression ---")
X_uni = df[['ENGINESIZE']].values
y_uni = df[['CO2EMISSIONS']].values
X_uni = np.c_[np.ones(X_uni.shape[0]), X_uni]
X_train_uni, X_test_uni, y_train_uni, y_test_uni = train_test_split(X_uni, y_uni, test_size=0.2,
random_state=4)


theta_uni = np.zeros((X_uni.shape[1], 1))
alpha = 0.01
iterations = 1000
theta_uni_final, cost_history_uni = gradient_descent(X_train_uni, y_train_uni, theta_uni, alpha,
iterations)


# Print hypothesis
print(f"Hypothesis: h(x) = {theta_uni_final[0][0]:.2f} + {theta_uni_final[1][0]:.2f} *
EngineSize")
# Accuracy
mse_uni, r2_uni = evaluate(X_test_uni, y_test_uni, theta_uni_final)
print(f"MSE: {mse_uni:.2f}, R²: {r2_uni:.2f}")


# Plot cost
plt.plot(cost_history_uni)
plt.title("Univariate Cost Convergence")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.grid(True)
plt.show()
```

```
--- Univariate Linear Regression ---
Hypothesis: h(x) = 99.40 + 45.72 * EngineSize
MSE: 791.17, R²: 0.79
```



```
# ========== STEP 4: MULTIVARIATE LINEAR REGRESSION ==========

print("\n--- Multivariate Linear Regression ---")

X_multi = df[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB']].values

y_multi = df[['CO2EMISSIONS']].values

X_multi = np.c_[np.ones(X_multi.shape[0]), X_multi]

X_train_multi, X_test_multi, y_train_multi, y_test_multi = train_test_split(X_multi, y_multi,
test_size=0.2, random_state=42)

theta_multi = np.zeros((X_multi.shape[1], 1))

theta_multi_final, cost_history_multi = gradient_descent(X_train_multi, y_train_multi,
theta_multi, alpha, iterations)


# Print hypothesis

print("Hypothesis: h(x) = {:.2f} + {:.2f}*ENGINESIZE + {:.2f}*CYLINDERS +
{:.2f}*FUELCONSUMPTION_COMB"

      .format(*theta_multi_final.flatten()))

# Accuracy

mse_multi, r2_multi = evaluate(X_test_multi, y_test_multi, theta_multi_final)

print(f"MSE: {mse_multi:.2f}, R²: {r2_multi:.2f}")

# Plot cost

plt.plot(cost_history_multi)

plt.title("Multivariate Cost Convergence")

plt.xlabel("Iterations")

plt.ylabel("Cost")

plt.grid(True)

plt.show()
```

```
--- Multivariate Linear Regression ---
Hypothesis: h(x) = 29.46 + -1.00*ENGINESIZE + 15.90*CYLINDERS + 11.79*FUELCONSUMPTION_COMB
MSE: 546.10, R²: 0.87
```



2. **Perform a non-linear regression algorithm to predict China's GDP from year 1960 to 2014 from given features. Elaborate gradient descent algorithm and hyper-parameter tuning for the best result with predefined convergent criteria.**

Dataset: china_gdp.csv

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

df = pd.read_csv('china_gdp.csv')
print(df.head())
```

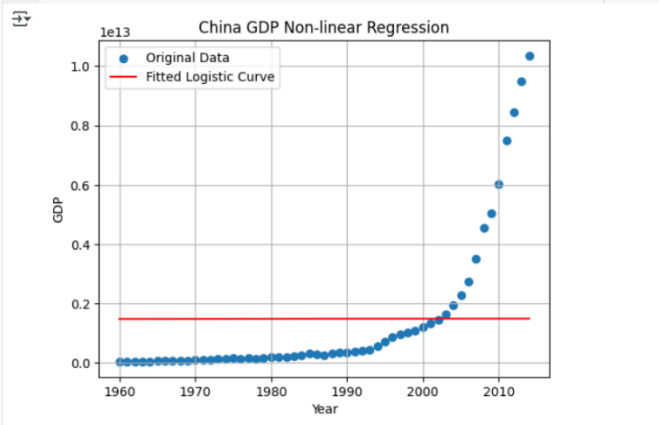
```
   Year         Value
0  1960  5.918412e+10
1  1961  4.955705e+10
2  1962  4.668518e+10
3  1963  5.009730e+10
4  1964  5.906225e+10
```

```
# Extract data
x_data = df["Year"].values
y_data = df["Value"].values

# Normalize
x = x_data / max(x_data)
y = y_data / max(y_data)
# Logistic function
def sigmoid(x, L, k, x0):
    return L / (1 + np.exp(-k * (x - x0)))

# Gradient descent for non-linear model
def gradient_descent(x, y, L, k, x0, alpha, iterations):
    m = len(x)
    cost_history = []
    for i in range(iterations):
        y_pred = sigmoid(x, L, k, x0)
        error = y_pred - y
        cost = np.sum(error ** 2) / (2 * m)
        cost_history.append(cost)

        # Compute gradients numerically (simplified)
        dL = np.sum(error * (1 / (1 + np.exp(-k * (x - x0))))) / m
```

```
        dk = np.sum(error * L * (x - x0) * np.exp(-k * (x - x0)) / ((1 + np.exp(-k * (x - x0)))
** 2)) / m
        dx0 = np.sum(error * L * k * np.exp(-k * (x - x0)) / ((1 + np.exp(-k * (x - x0))) ** 2))
/ m

        # Update parameters
        L -= alpha * dL
        k -= alpha * dk
        x0 -= alpha * dx0
        # Convergence condition
        if i > 1 and abs(cost_history[-1] - cost_history[-2]) < 1e-6:
            break
    return L, k, x0, cost_history

# Initialize parameters
L = 1
k = 1
x0 = 0.5
alpha = 0.1
iterations = 10000
L_final, k_final, x0_final, cost_history = gradient_descent(x, y, L, k, x0, alpha, iterations)
print(f"Optimal parameters: L={L_final}, k={k_final}, x0={x0_final}")
```

```
Optimal parameters: L=0.22335620753293403, k=0.900065934451286, x0=0.33049893727398616
```

```
# Predict using the trained model
x_range = np.linspace(min(x), max(x), 100)
y_pred = sigmoid(x_range, L_final, k_final, x0_final)

# Convert back to original scale
plt.scatter(x_data, y_data, label='Original Data')
plt.plot(x_range * max(x_data), y_pred * max(y_data), color='red', label='Fitted Logistic Curve')
plt.xlabel("Year")
plt.ylabel("GDP")
plt.legend()
plt.title("China GDP Non-linear Regression")
plt.grid(True)
plt.show()
```



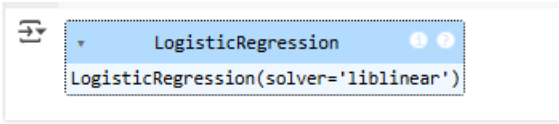

```
from sklearn.metrics import mean_squared_error, r2_score
y_fit = sigmoid(x, L_final, k_final, x0_final)
mse = mean_squared_error(y, y_fit)
r2 = r2_score(y, y_fit)
print(f"MSE: {mse:.6f}")
print(f"R² Score: {r2:.6f}")
```

```
MSE: 0.057133
R² Score: 0.001768
```

3. **Perform logistic regression to classify if a patient has a benign tumor or malignant tumor (cancer) based on the features provided. Utilize gradient descent with regularization for hyper-parameter tuning. Also, generate log-loss curve for this problem.**

   Dataset: samples_cancer.csv

```
# Import necessary libraries
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.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, log_loss
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("samples_cancer.csv")

df.replace("?", np.nan, inplace=True)
df = df.apply(pd.to_numeric, errors="coerce")
df.dropna(inplace=True)

label_encoders = {}
for col in ['Class']:
    encoder = LabelEncoder()
    df[col] = encoder.fit_transform(df[col])  # Convert category labels
    label_encoders[col]=encoder

feature_df=df[df.columns[0:-1]]
X=np.asarray(feature_df)
y=np.asarray(df[df.columns[-1]])



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4)

lr = LogisticRegression(solver='liblinear', penalty='l2', C=1.0)
lr.fit(X_train, y_train)
```



```
df.head(10)
```

	ID	Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
0	1000025	5	1	1	1	2	1.0	3	1	1	0
1	1002945	5	4	4	5	7	10.0	3	2	1	0
2	1015425	3	1	1	1	2	2.0	3	1	1	0
3	1016277	6	8	8	1	3	4.0	3	7	1	0
4	1017023	4	1	1	3	2	1.0	3	1	1	0
5	1017122	8	10	10	8	7	10.0	9	7	1	1
6	1018099	1	1	1	1	2	10.0	3	1	1	0
7	1018561	2	1	2	1	2	1.0	3	1	1	0
8	1033078	2	1	1	1	2	1.0	1	1	5	0
9	1033078	4	2	1	1	2	1.0	2	1	1	0

```
print("Classes after encoding:", encoder.classes_)
```

⇨ Classes after encoding: [2 4]

```
y_pred = lr.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Logistic Regression Accuracy:", np.round(accuracy * 100, 2), "%")
```

⇨ Logistic Regression Accuracy: 65.69 %

```
# Classification Report

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

```
Classification Report:
              precision    recall  f1-score   support

           0       0.66      1.00      0.79        90
           1       0.00      0.00      0.00        47

    accuracy                           0.66       137
   macro avg       0.33      0.50      0.40       137
weighted avg       0.43      0.66      0.52       137
```

```
features = [[1000025, 5,  1,  1,  1,  2,  1,  3, 1, 1]]

predicted_category = lr.predict(features)

print("Predicted tumor for a patient:",
label_encoders["Class"].inverse_transform(predicted_category))
```

⇨ Predicted tumor for a patient: [2]

```
y_train_prob = lr.predict_proba(X_train)
y_test_prob = lr.predict_proba(X_test)


train_log_loss = log_loss(y_train, y_train_prob)

test_log_loss = log_loss(y_test, y_test_prob)

print(f"Training Log Loss: {train_log_loss:.4f}")

print(f"Testing Log Loss: {test_log_loss:.4f}")
```

⇨ Training Log Loss: 0.6432
⇨ Testing Log Loss: 0.6307

```
import numpy as np

import matplotlib.pyplot as plt
```

```
y_train_prob_pos = y_train_prob[:, 1] if y_train_prob.ndim > 1 else y_train_prob
y_test_prob_pos = y_test_prob[:, 1] if y_test_prob.ndim > 1 else y_test_prob

# Function to calculate log-loss
def compute_log_loss(y_true, y_pred):
    return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))

# Store log-loss at each iteration during training
train_log_loss = []
test_log_loss = []

for i in range(1, 1001):  # Replace 1000 with your actual number of iterations
    # Calculate the log-loss for training and test set at each iteration
    train_log_loss.append(compute_log_loss(y_train, y_train_prob_pos))
    test_log_loss.append(compute_log_loss(y_test, y_test_prob_pos))

    # Update your model parameters here for each iteration (this part depends on your
implementation)
# Plotting Log-Loss Curve
epochs = np.arange(1, len(train_log_loss) + 1)
plt.figure(figsize=(7, 5))
plt.plot(epochs, train_log_loss, label="Train Log-Loss", color="blue")
plt.plot(epochs, test_log_loss, label="Test Log-Loss", color="red")
plt.xlabel("Iterations")
plt.ylabel("Log-Loss")
plt.title("Log-Loss Curve for Logistic Regression")
plt.legend()
plt.show()
```

4. Using Decision Tree algorithm, predict which drug among drug X, drug Y and drug C should be given to a patient. Find the accuracy of the decision tree in predicting the correct drug for the patient.

Dataset: drug.csv

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the dataset
file_path = "drug.csv"  # Change this to your file path
df = pd.read_csv(file_path)

# Encode categorical variables
label_encoders = {}
for col in ["Sex", "BP", "Cholesterol", "Drug"]:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# Define features and target variable
X = df.drop(columns=["Drug"])
y = df["Drug"]

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=4)

# Train a Decision Tree classifier with specified parameters
dtree = DecisionTreeClassifier(criterion='entropy', max_depth=None)
dtree_y_pred = dtree.fit(X_train, y_train).predict(X_test)

# Calculate accuracy
dtree_acc = accuracy_score(y_test, dtree_y_pred)
print(f'Accuracy of Decision Tree Classifier: {dtree_acc * 100:.2f}%')
```

```
Accuracy of Decision Tree Classifier: 95.00%
```

5. Using KNN algorithm, predict which category a customer belongs to onthe basis ofthe data provided by a telecommunications firm. Find the accuracy of the KNN algorithm in predicting the category of a customer

Dataset: teleCust.csv

```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn import preprocessing

#Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import jaccard_score
from sklearn.metrics import roc_curve

from google.colab import files
uploaded = files.upload()
```

```
Saving teleCust.csv to teleCust.csv
```

```
data=pd.read_csv('teleCust.csv')
raw = data.copy()
data.shape
```

```
(1000, 12)
```

```
data.head()
data.describe()

x=np.asarray(data[data.columns[0:-1]])
y=np.asarray(data[data.columns[-1]])
x0=preprocessing.StandardScaler().fit(x)
x1=x0.transform(x)
```

x1

```
array([[-0.02696767, -1.055125  ,  0.18450456, ..., -0.22207644,
        -1.03459817, -0.23065004],
       [ 1.19883553, -1.14880563, -0.69181243, ..., -0.22207644,
        -1.03459817,  2.55666158],
       [ 1.19883553,  1.52109247,  0.82182601, ..., -0.22207644,
         0.96655883, -0.23065004],
       ...,
       [ 1.19883553,  1.47425216,  1.37948227, ..., -0.22207644,
         0.96655883, -0.92747794],
       [ 1.19883553,  1.61477311,  0.58283046, ..., -0.22207644,
         0.96655883, -0.92747794],
       [ 1.19883553,  0.67796676, -0.45281689, ..., -0.22207644,
         0.96655883,  0.46617787]])
```

```
x_train,x_test,y_train,y_test=train_test_split(x1,y,test_size=0.2,random_state=10)
```

```
knn = KNeighborsClassifier(n_neighbors=5,metric='minkowski')
y_knn = knn.fit(x_train,y_train)
y_knn_pred = y_knn.predict(x_test)
```

```
knn_acc=accuracy_score(y_test,y_knn_pred)
knn_f1=f1_score(y_test,y_knn_pred,average='micro')
```

```
print("accuracy score ", knn_acc*100, "%")
print("f1_score ", knn_f1*100, "%")
```

```
accuracy score  42.5 %
f1_score  42.5 %
```

```
knn_cm=confusion_matrix(y_test,y_knn_pred)
knn_cr=classification_report(y_test,y_knn_pred)
print('confusion matrix\n' , knn_cm)
print('classification report\n', knn_cr)
```

```
confusion matrix
 [[36  3 11 13]
 [ 7 11 12  7]
 [18  7 21  6]
 [15  6 10 17]]
classification report
              precision    recall  f1-score   support

           1       0.47      0.57      0.52        63
           2       0.41      0.30      0.34        37
           3       0.39      0.40      0.40        52
           4       0.40      0.35      0.37        48

    accuracy                           0.42       200
   macro avg       0.42      0.41      0.41       200
weighted avg       0.42      0.42      0.42       200
```

6. Using Gaussian and Multinomial Naive Bayes algorithms, predict if a person is diabetic or not, based on the features provided. Find accuracy and F1-Scores of both algorithms. Dataset: pima-indians-diabetes.data.csv

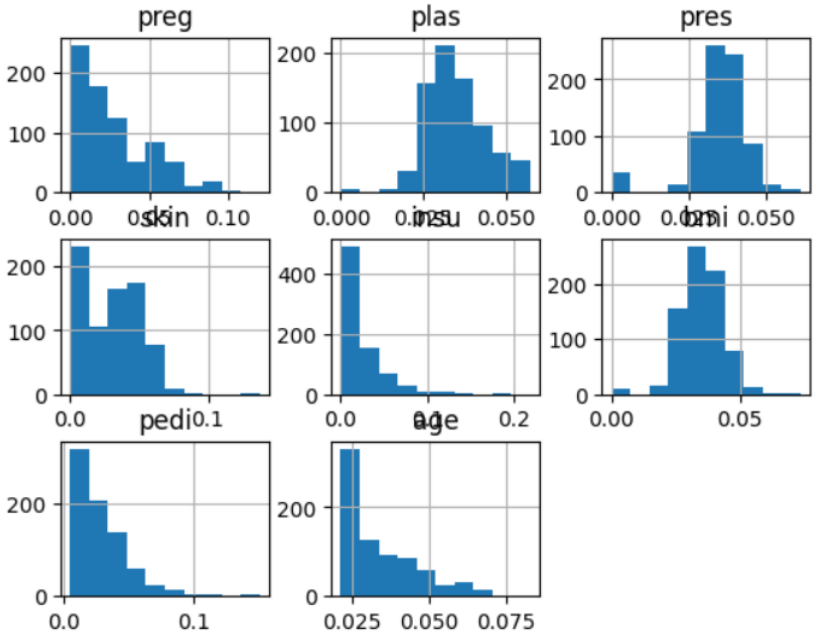
```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier

from google.colab import files
uploaded = files.upload()
```

```
 Saving pima-indians-diabetes.csv to pima-indians-diabetes.csv
```

```
data=pd.read_csv('pima-indians-diabetes.csv')
raw = data.copy()
data.describe()
headers = ['preg', 'plas', 'pres', 'skin', 'insu', 'bmi', 'pedi', 'age', 'class']
data = pd.read_csv('pima-indians-diabetes.csv', names=headers)
data.head()
data.describe()
data=data.drop_duplicates()
feature_df=data[data.columns[0:-1]]
x=np.asarray(feature_df)
y=np.asarray(data[data.columns[-1]])
x1 = preprocessing.normalize(x, axis=0)
x_train,x_test,y_train,y_test=train_test_split(x1,y,test_size=0.2,random_state=16)
```

```
pd.DataFrame(data=x1, columns=headers[:-1]).hist()
```

```
array([[<Axes: title={'center': 'preg'}>,
        <Axes: title={'center': 'plas'}>,
        <Axes: title={'center': 'pres'}>],
       [<Axes: title={'center': 'skin'}>,
        <Axes: title={'center': 'insu'}>,
        <Axes: title={'center': 'bmi'}>],
       [<Axes: title={'center': 'pedi'}>,
        <Axes: title={'center': 'age'}>, <Axes: >]], dtype=object)
```




```
gnb = GaussianNB()

y_pred = gnb.fit(x_train, y_train).predict(x_test)

gnb_accuracy = accuracy_score(y_test, y_pred)

gnb_f1 = f1_score(y_test, y_pred)

print("Gaussian Naive Bayes:")

print("Accuracy:", round(gnb_accuracy*100, 2))

print("F1-score:", round(gnb_f1*100, 2))
```

```
Gaussian Naive Bayes:
Accuracy: 82.47
F1-score: 72.16
```

```
mnb = MultinomialNB()

y_pred_mnb = mnb.fit(x_train, y_train).predict(x_test)

mnb_accuracy = accuracy_score(y_test, y_pred_mnb)

mnb_f1 = f1_score(y_test, y_pred)

print("Multinomial Naive Bayes:")

print("Accuracy:", round(mnb_accuracy*100, 2))

print("F1-score:", round(mnb_f1*100, 2))
```

```
Multinomial Naive Bayes:
Accuracy: 66.23
F1-score: 72.16
```

7. Using SVM algorithm, predict if a patient has a benign tumor or malignant tumor (cancer) based on the features provided. Use the following kernel for the SVM algorithm:

a) Linear b) Polynomial c) RBF d) Sigmoid

Find the following metrics for each of the SVM algorithms:

1) Accuracy 2) Recall 3) Precision 4) F1-Score

5) Jaccard Score 6) Error rates 7) Confusion Matrix

Compare all four SVM models using an ROC curve.

Dataset: samples_cancer.csv

```
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.svm import SVC
from sklearn.metrics import (
    accuracy_score, recall_score, precision_score, f1_score,
    jaccard_score, confusion_matrix, roc_curve, auc
)
# Load dataset
df = pd.read_csv('samples_cancer.csv')
print(df.head())
```

```
        ID  Clump  UnifSize  UnifShape  MargAdh  SingEpiSize BareNuc  \
0  1000025      5         1          1        1            2       1
1  1002945      5         4          4        5            7      10
2  1015425      3         1          1        1            2       2
3  1016277      6         8          8        1            3       4
4  1017023      4         1          1        3            2       1

   BlandChrom  NormNucl  Mit  Class
0           3         1    1      2
1           3         2    1      2
2           3         1    1      2
3           3         7    1      2
4           3         1    1      2
```

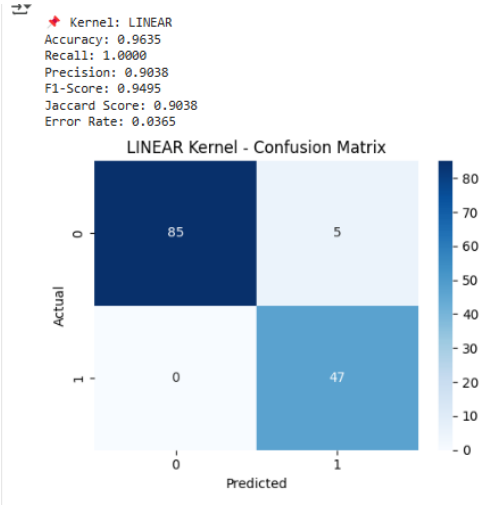
```
# Step 3: Data Preprocessing
# Convert BareNuc to numeric (in case of '?')
df['BareNuc'] = pd.to_numeric(df['BareNuc'], errors='coerce')
# Drop rows with missing values
df.dropna(inplace=True)
# Drop the 'ID' column
df.drop(columns=['ID'], inplace=True)
# Convert target: 2 = Benign (0), 4 = Malignant (1)
df['Class'] = df['Class'].apply(lambda x: 1 if x == 4 else 0)
```

```
# Step 4: Feature Scaling
X = df.drop('Class', axis=1)
y = df['Class']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=4)
import seaborn as sns
# Step 5: Train and Evaluate SVM Models
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
results = {}
fpr_dict, tpr_dict = {}, {}

for kernel in kernels:
    model = SVC(kernel=kernel, probability=True, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    jaccard = jaccard_score(y_test, y_pred)
    error = 1 - acc
    cm = confusion_matrix(y_test, y_pred)

    results[kernel] = {
        'Accuracy': acc,
        'Recall': rec,
        'Precision': prec,
        'F1-Score': f1,
        'Jaccard Score': jaccard,
        'Error Rate': error,
        'Confusion Matrix': cm
    }
    # ROC
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    fpr_dict[kernel] = fpr
```

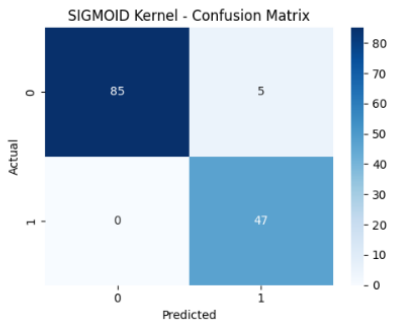
```
    tpr_dict[kernel] = tpr

import seaborn as sns
import matplotlib.pyplot as plt

# Step 6: Display Results with Confusion Matrix Heatmaps
for kernel in results:
    print(f"\n📌 Kernel: {kernel.upper()}")
    for metric, value in results[kernel].items():
        if metric != "Confusion Matrix":
            print(f"{metric}: {value:.4f}")

    # Plot Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(results[kernel]["Confusion Matrix"], annot=True, fmt='d', cmap="Blues")
    plt.title(f"{kernel.upper()} Kernel - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.tight_layout()
    plt.show()
```

[1]

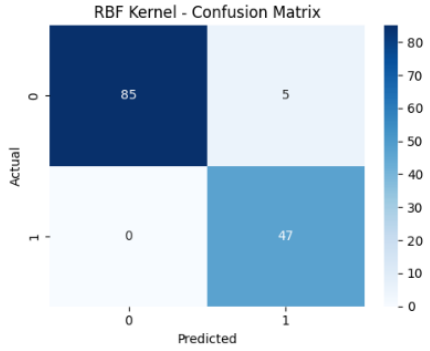
```
📌 Kernel: LINEAR
Accuracy: 0.9635
Recall: 1.0000
Precision: 0.9038
F1-Score: 0.9495
Jaccard Score: 0.9038
Error Rate: 0.0365
```



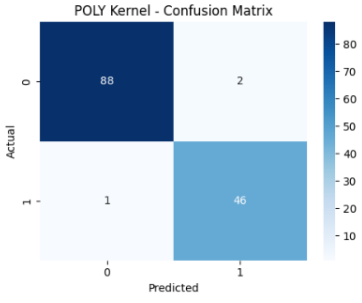

```
📌 Kernel: SIGMOID
Accuracy: 0.9635
Recall: 1.0000
Precision: 0.9038
F1-Score: 0.9495
Jaccard Score: 0.9038
Error Rate: 0.0365
```



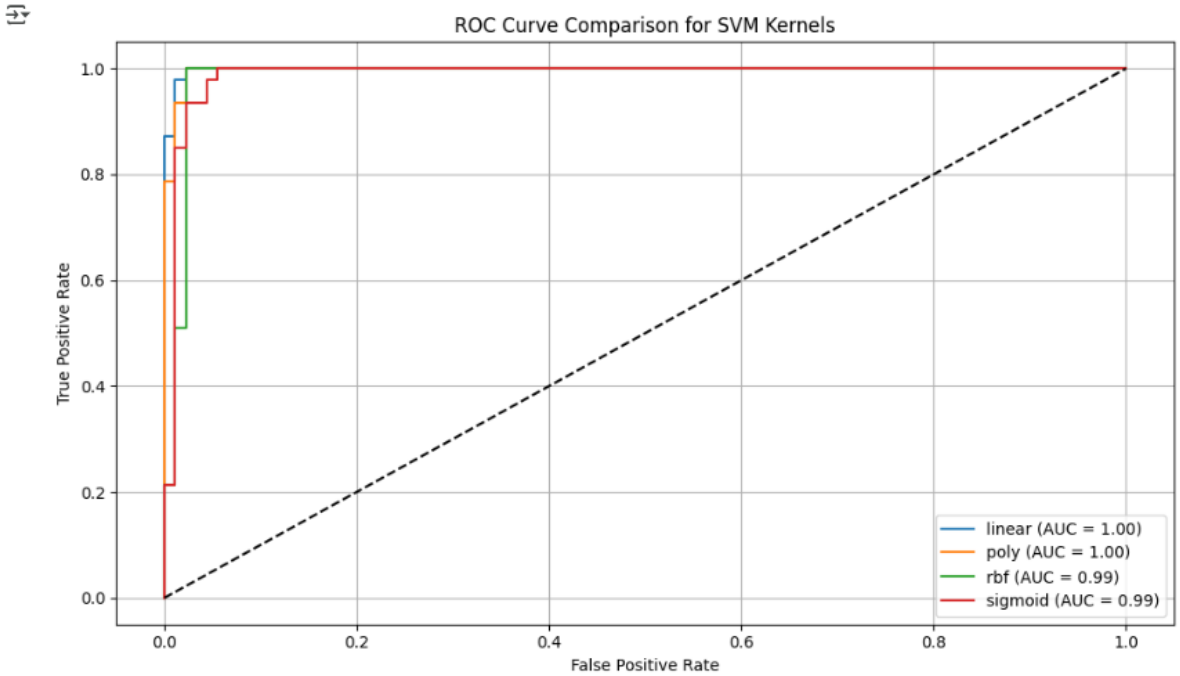

```
📌 Kernel: RBF
Accuracy: 0.9635
Recall: 1.0000
Precision: 0.9038
F1-Score: 0.9495
Jaccard Score: 0.9038
Error Rate: 0.0365
```




```
📌 Kernel: POLY
Accuracy: 0.9781
Recall: 0.9787
Precision: 0.9583
F1-Score: 0.9684
Jaccard Score: 0.9388
Error Rate: 0.0219
```



```
# Step 7: Plot ROC Curve

plt.figure(figsize=(10, 6))

for kernel in kernels:

    plt.plot(fpr_dict[kernel], tpr_dict[kernel], label=f'{kernel} (AUC = {auc(fpr_dict[kernel], tpr_dict[kernel]):.2f})')


plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve Comparison for SVM Kernels')

plt.legend()

plt.grid(True)

plt.tight_layout()

plt.show()
```



8. **Applying SVM, Naive Bayes, Decision tree and KNN predict diabetes based on features set. Compare the four classification algorithms with performance metrics such as accuracy, recall, precision, F1- score. Also design the heat map confusion matrix for above algorithms and construct ROC curve for comparison.**

Dataset: pima-indians-diabetes.data.csv

```
# Step 1: Import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score, recall_score, precision_score, f1_score,
    confusion_matrix, roc_curve, auc
)

# Step 2: Upload the dataset
from google.colab import files
uploaded = files.upload()

# Step 3: Load the dataset
df = pd.read_csv('pima-indians-diabetes.data.csv', header=None)
df.head()
```

	0	1	2	3	4	5	6	7	8
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

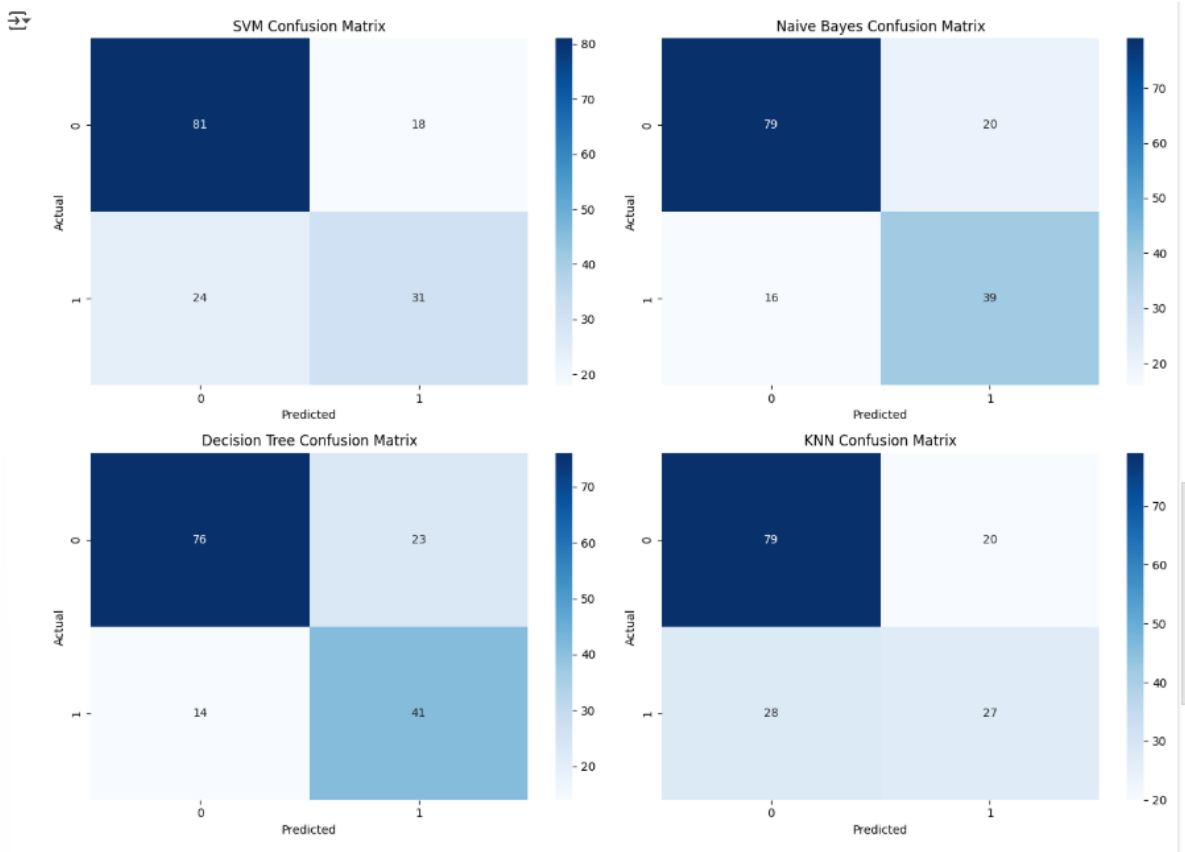
```
# S
X =
y =

# Step 5: Normalize and split
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Step 6: Import models
```

```
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

models = {
    "SVM": SVC(probability=True),
    "Naive Bayes": GaussianNB(),
    "Decision Tree": DecisionTreeClassifier(),
    "KNN": KNeighborsClassifier()
}

metrics = {}

# Step 7: Train and Evaluate
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    acc = accuracy_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    cm = confusion_matrix(y_test, y_pred)

    metrics[name] = {
        "Accuracy": acc,
        "Recall": rec,
        "Precision": prec,
        "F1 Score": f1,
        "Confusion Matrix": cm,
        "y_prob": y_prob
    }

# Step 8: Print metrics
for name in metrics:
    print(f"\n🔍 {name}")
    print(f"Accuracy: {metrics[name]['Accuracy']:.4f}")
```

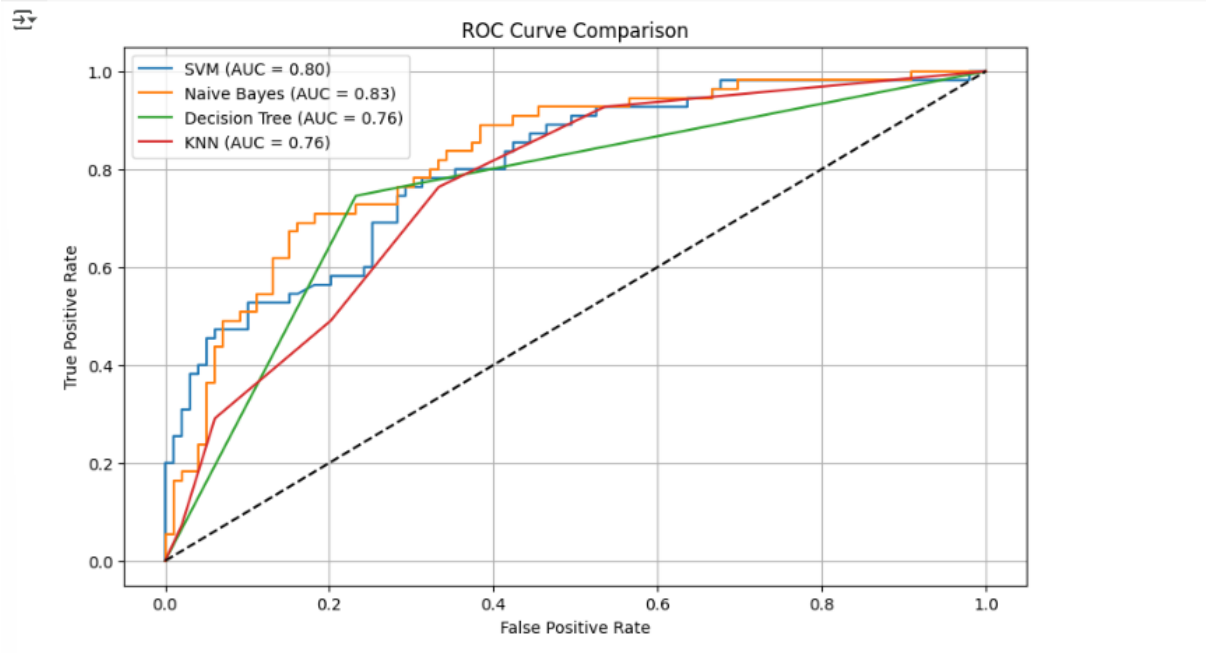
```
    print(f"Recall: {metrics[name]['Recall']:.4f}")
    print(f"Precision: {metrics[name]['Precision']:.4f}")
    print(f"F1 Score: {metrics[name]['F1 Score']:.4f}")
```

```
🔍 SVM
Accuracy: 0.7273
Recall: 0.5636
Precision: 0.6327
F1 Score: 0.5962

🔍 Naive Bayes
Accuracy: 0.7662
Recall: 0.7091
Precision: 0.6610
F1 Score: 0.6842

🔍 Decision Tree
Accuracy: 0.7597
Recall: 0.7455
Precision: 0.6406
F1 Score: 0.6891

🔍 KNN
Accuracy: 0.6883
Recall: 0.4909
Precision: 0.5745
F1 Score: 0.5294
```

```
for i, name in enumerate(metrics):
    plt.subplot(2, 2, i+1)
    sns.heatmap(metrics[name]["Confusion Matrix"], annot=True, fmt='d', cmap="Blues")
    plt.title(f"{name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```



```
# ROC Curve comparison
plt.figure(figsize=(10, 6))
for name in models:
    fpr, tpr, _ = roc_curve(y_test, metrics[name]["y_prob"])
    plt.plot(fpr, tpr, label=f'{name} (AUC = {auc(fpr, tpr):.2f})')

plt.plot([0, 1], [0, 1], 'k--')
plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.show()
```



9. Design and train the neural network that computes the functionality of XOR function.

```
# Step 1: Import libraries
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
# Step 2: Define XOR data
X = np.array([[0, 0],
              [0, 1],
              [1, 0],
              [1, 1]])
y = np.array([[0],
              [1],
              [1],
              [0]])
# Step 3: Build the model
model = Sequential()
model.add(Dense(4, input_dim=2, activation='relu'))  # Hidden layer with 4 neurons
model.add(Dense(1, activation='sigmoid'))            # Output layer
# Step 4: Compile the model
model.compile(optimizer=Adam(learning_rate=0.1), loss='binary_crossentropy',
metrics=['accuracy'])
# Step 5: Train the model
model.fit(X, y, epochs=500, verbose=0)
# Step 6: Evaluate and Predict
predictions = model.predict(X).round()
print("\n🧠 XOR Predictions:")
for i in range(len(X)):
    print(f"Input: {X[i]} -> Predicted Output: {int(predictions[i][0])}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

🧠 XOR Predictions:
Input: [0 0] -> Predicted Output: 0
Input: [0 1] -> Predicted Output: 0
Input: [1 0] -> Predicted Output: 1
Input: [1 1] -> Predicted Output: 1
```

10. Perform Multilayer perception neural network to classify flower type. Utilize number of hidden layer 5 and 200 to 400 iteration with learning rate. Try with different loss functions/ activation functions such as MSE, Cross entropy, sigmoid, tanh, ReLU along with different optimizer GD, SGD, Adam. Illustrate the result with performance metrics and observe Weight, Loss curve and accuracy curve.

Dataset: Iris dataset

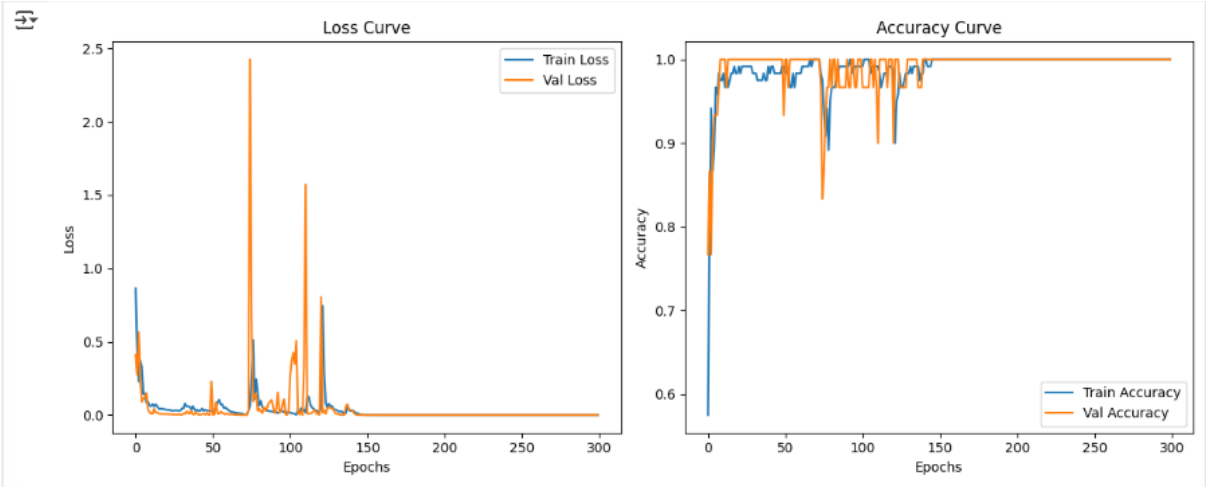
```
# Step 1: Import required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelBinarizer
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.losses import MeanSquaredError, CategoricalCrossentropy

iris = pd.read_csv('iris.csv', header=None)
iris.head()
```

	0	1	2	3	4
0	sepal_length	sepal_width	petal_length	petal_width	species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa

```
# Step 2: Load and prepare Iris data
iris = load_iris()
X = iris.data
y = iris.target

# One-hot encode the target
lb = LabelBinarizer()
y = lb.fit_transform(y)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Standardize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 3: Build MLP model (with 5 hidden layers)
def build_model(activation='relu', loss_fn='mse', optimizer='adam', learning_rate=0.01):
    model = Sequential()
    model.add(Dense(64, input_dim=4, activation=activation))
    for _ in range(4):  # 5 layers in total
        model.add(Dense(64, activation=activation))
    model.add(Dense(3, activation='softmax'))  # Output layer

    # Choose optimizer
    if optimizer == 'sgd':
        opt = SGD(learning_rate=learning_rate)
    elif optimizer == 'adam':
        opt = Adam(learning_rate=learning_rate)
    else:
        opt = SGD(learning_rate=learning_rate)  # Default fallback

    # Choose loss function
    if loss_fn == 'mse':
        loss = MeanSquaredError()
    elif loss_fn == 'crossentropy':
        loss = CategoricalCrossentropy()
    else:
        loss = MeanSquaredError()
    model.compile(optimizer=opt, loss=loss, metrics=['accuracy'])
    return model

# Step 4: Train and evaluate the model
activation = 'relu'              # Try: 'sigmoid', 'tanh', 'relu'
loss_fn = 'crossentropy'         # Try: 'mse', 'crossentropy'
optimizer = 'adam'               # Try: 'sgd', 'adam'
learning_rate = 0.01
epochs = 300

model = build_model(activation, loss_fn, optimizer, learning_rate)
```

```
history = model.fit(X_train, y_train, epochs=epochs, validation_data=(X_test, y_test), verbose=0)

# Step 5: Plot Loss and Accuracy Curves
plt.figure(figsize=(12, 5))
# Loss curve
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title("Loss Curve")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

# Accuracy curve
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title("Accuracy Curve")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.tight_layout()
plt.show()
```




```
# Step 6: Final Evaluation on Test Set
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"\n✅ Final Test Accuracy: {accuracy*100:.2f}% | Loss: {loss:.4f}")
```

⇨ ✅ Final Test Accuracy: 100.00% | Loss: 0.0000

11. Perform k- means clustering algorithm for customer segmentation from given features. Utilize Euclidean distance and Manhattan distance for this problem. Also, plot in terms of 2D and 3D clusters this problem.

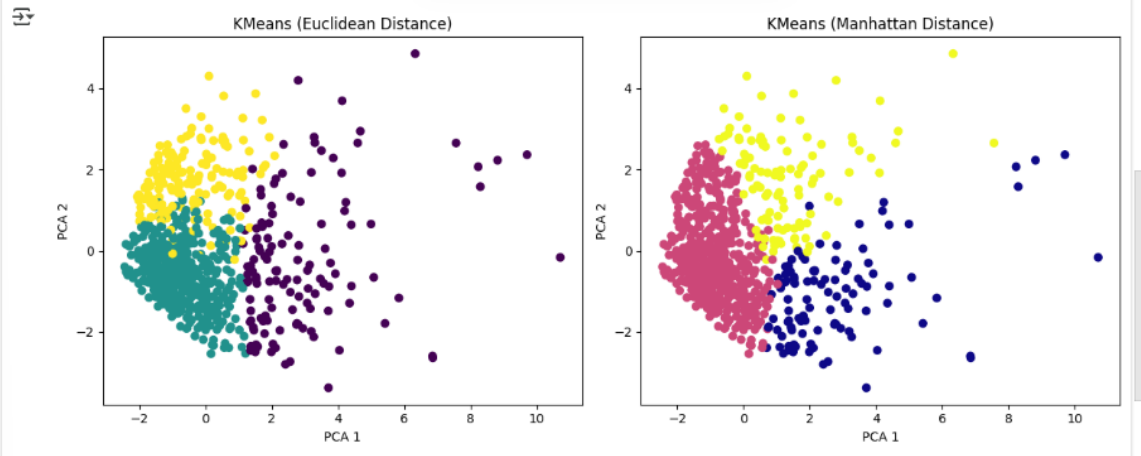
Dataset: Customer segmentation dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from scipy.spatial.distance import cdist

# Load the dataset (update name if different)
df = pd.read_csv("customer_segmentation.csv")
df.head()
```

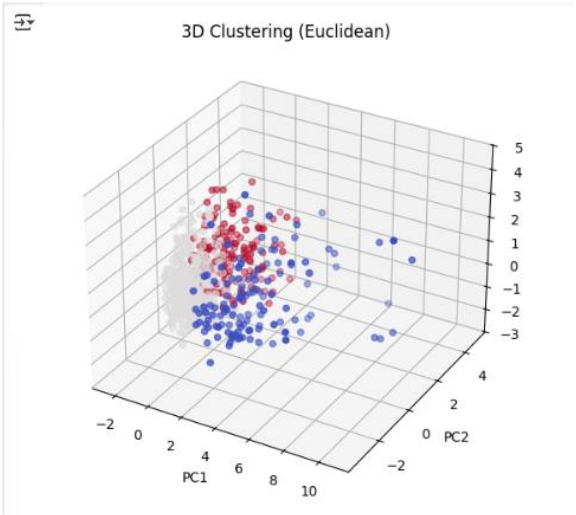
	Customer Id	Age	Edu	Years Employed	Income	Card Debt	Other Debt	Defaulted	Address	DebtIncomeRatio
0	1	41	2	6	19	0.124	1.073	0.0	NBA001	6.3
1	2	47	1	26	100	4.582	8.218	0.0	NBA021	12.8
2	3	33	2	10	57	6.111	5.802	1.0	NBA013	20.9
3	4	29	2	4	19	0.681	0.516	0.0	NBA009	6.3
4	5	47	1	31	253	9.308	8.908	0.0	NBA008	7.2

```
df = df.dropna()
X = df.select_dtypes(include=[np.number])
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
kmeans_euc = KMeans(n_clusters=3, random_state=42)
labels_euc = kmeans_euc.fit_predict(X_scaled)
df['Cluster_Euclidean'] = labels_euc

def kmeans_manhattan(X, k, max_iters=100):
    np.random.seed(42)
    centroids = X[np.random.choice(range(len(X)), k, replace=False)]

    for _ in range(max_iters):
```

```
        labels = np.argmin(cdist(X, centroids, metric='cityblock'), axis=1)
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)])
        if np.allclose(centroids, new_centroids):
            break
        centroids = new_centroids

    return labels, centroids

labels_man, _ = kmeans_manhattan(X_scaled, k=3)
df['Cluster_Manhattan'] = labels_man

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.figure(figsize=(12,5))

# Euclidean Clustering
plt.subplot(1,2,1)
plt.scatter(X_pca[:,0], X_pca[:,1], c=df['Cluster_Euclidean'], cmap='viridis')
plt.title('KMeans (Euclidean Distance)')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
# Manhattan Clustering
plt.subplot(1,2,2)
plt.scatter(X_pca[:,0], X_pca[:,1], c=df['Cluster_Manhattan'], cmap='plasma')
plt.title('KMeans (Manhattan Distance)')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.tight_layout()
plt.show()
```




```
pca_3d = PCA(n_components=3)

X_3d = pca_3d.fit_transform(X_scaled)


fig = plt.figure(figsize=(10, 6))

ax = fig.add_subplot(111, projection='3d')

ax.scatter(X_3d[:, 0], X_3d[:, 1], X_3d[:, 2], c=df['Cluster_Euclidean'],
cmap='coolwarm')

ax.set_title('3D Clustering (Euclidean)')

ax.set_xlabel("PC1")

ax.set_ylabel("PC2")

ax.set_zlabel("PC3")

plt.show()
```

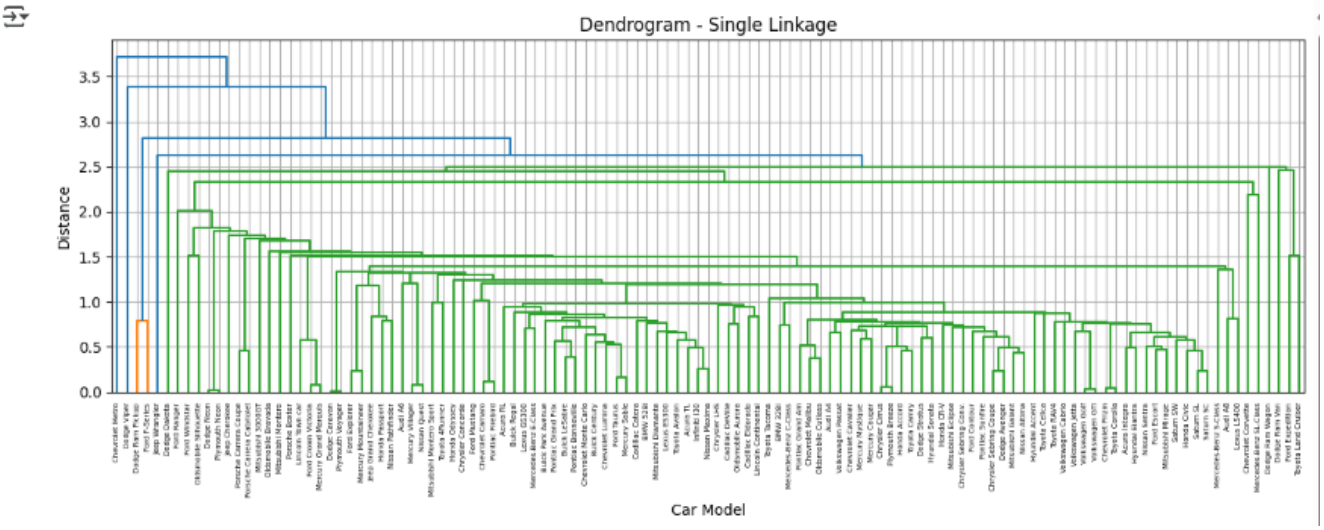


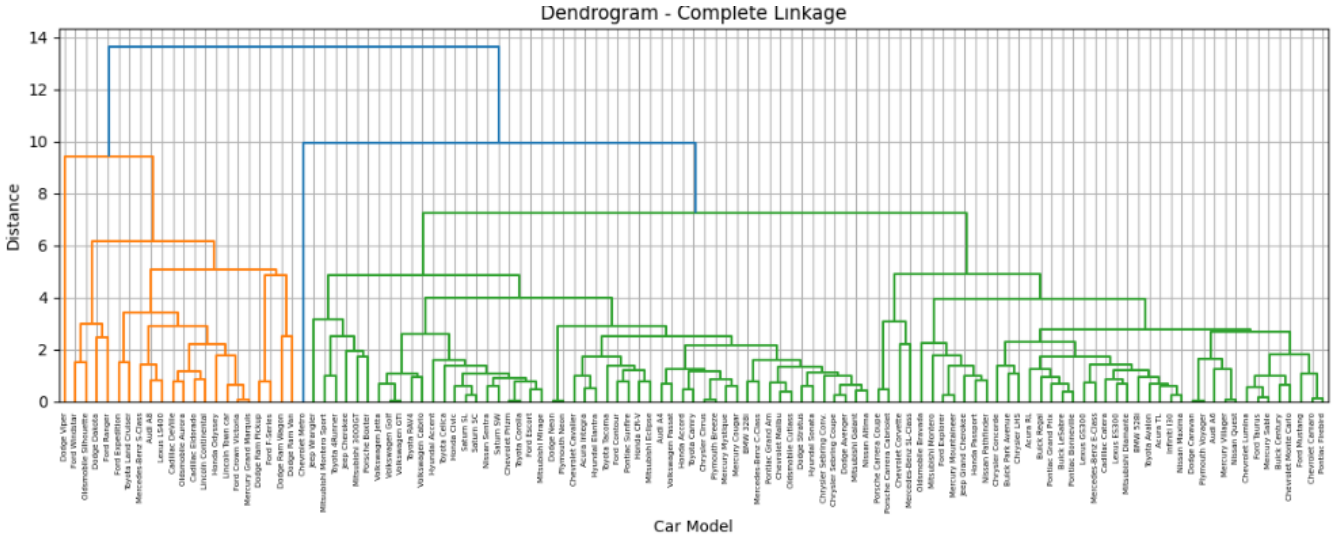
12. Perform hierarchical clustering such as Agglomerative algorithm and Divisive algorithm to group several vehicles. Utilize single, complete and average linkage to define the cluster. Also draw the dendogram for this problem.

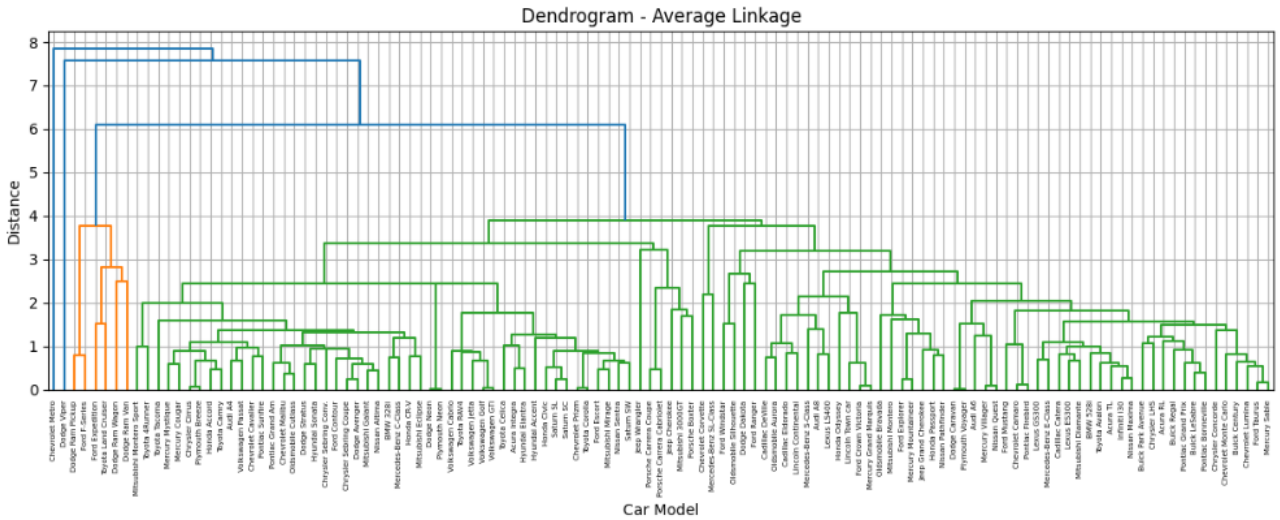
Dataset: Vehicle dataset

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

from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering
# Load dataset
df = pd.read_csv('Vehicle_dataset.csv')

# Step 1: Load & Clean Data
df.replace('$null$', np.nan, inplace=True)
df.dropna(inplace=True)

# Convert necessary columns to numeric
cols_to_convert = ['engine_s', 'horsepow', 'wheelbas', 'width', 'length',
                   'curb_wgt', 'fuel_cap', 'mpg']
df[cols_to_convert] = df[cols_to_convert].astype(float)

# Step 2: Select Features for Clustering
features = df[cols_to_convert]
# Step 3: Standardize Features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Step 4 & 5: Dendrograms with 3 Linkage Types
linkage_methods = ['single', 'complete', 'average']
labels = df['manufact'] + ' ' + df['model']

for method in linkage_methods:
    plt.figure(figsize=(12, 5))
    Z = linkage(scaled_features, method=method)
    dendrogram(Z, labels=labels.values, leaf_rotation=90)
    plt.title(f'Dendrogram - {method.capitalize()} Linkage')
    plt.xlabel('Car Model')
```

```
plt.ylabel('Distance')
plt.tight_layout()
plt.grid(True)
plt.show()
```







13. Perform DBSCAN algorithm for weather station clustering. Utilize proper data cleaning and feature selection. Also, plot all outlier of the cluster label.

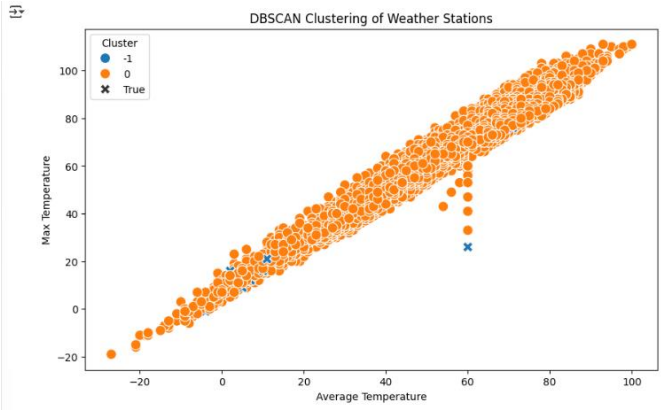
Dataset: Weather Station dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

df = pd.read_csv('weather_stations.csv')
df.head()
```

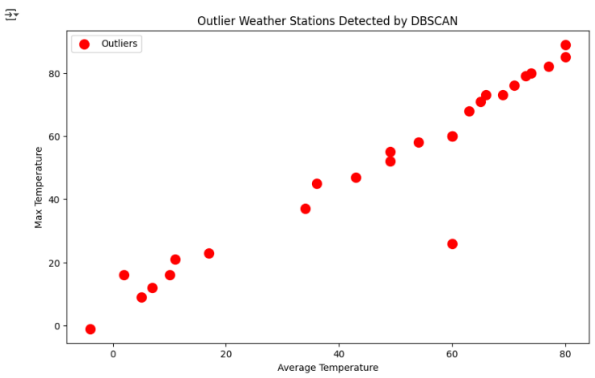
	Data.Precipitation	Date.Full	Date.Month	Date.Week of	Date.Year	Station.City	Station.Code	Station.Location	Station.State	Data.Temperature.Avg Temp	Data.Temperature.Max Temp	Data.Temperature.Min Temp	Data.Wind.Direction	Data.Wind.Speed
0	0.00	2016-01-03	1	3	2016	Birmingham	BHM	Birmingham, AL	Alabama	39	46	32	33	4.33
1	0.00	2016-01-03	1	3	2016	Huntsville	HSV	Huntsville, AL	Alabama	39	47	31	32	3.86
2	0.16	2016-01-03	1	3	2016	Mobile	MOB	Mobile, AL	Alabama	46	51	41	35	9.73
3	0.00	2016-01-03	1	3	2016	Montgomery	MGM	Montgomery, AL	Alabama	45	52	38	32	6.86
4	0.01	2016-01-03	1	3	2016	Anchorage	ANC	Anchorage, AK	Alaska	34	38	29	19	7.80

```
# Drop rows with missing values
df.dropna(inplace=True)
# Display basic info
df.info()

features = df[[
    'Data.Temperature.Avg Temp',
    'Data.Temperature.Max Temp',
    'Data.Temperature.Min Temp',
    'Data.Wind.Speed',
    'Data.Precipitation'
]]
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
# Apply DBSCAN
dbscan = DBSCAN(eps=1.3, min_samples=5)  # You can tweak eps/min_samples
clusters = dbscan.fit_predict(scaled_features)

# Add cluster labels to original dataframe
df = df.loc[features.index]  # Keep only rows with complete features
df['Cluster'] = clusters

# Plot Clusters (Using first 2 temp features for visualization)
plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(
    x=df['Data.Temperature.Avg Temp'],
    y=df['Data.Temperature.Max Temp'],
    hue=df['Cluster'],
    palette='tab10',
    style=(df['Cluster'] == -1),
    s=100
)
plt.title('DBSCAN Clustering of Weather Stations')
plt.xlabel('Average Temperature')
plt.ylabel('Max Temperature')
plt.legend(title='Cluster')
plt.show()
```




```
# Plot Only Outliers
outliers = df[df['Cluster'] == -1]
plt.figure(figsize=(10, 6))
plt.scatter(
    outliers['Data.Temperature.Avg Temp'],
    outliers['Data.Temperature.Max Temp'],
    c='red',
    label='Outliers',
    s=100
)
plt.xlabel('Average Temperature')
plt.ylabel('Max Temperature')
plt.title('Outlier Weather Stations Detected by DBSCAN')
plt.legend()
plt.show()
```



14. Design a recommender system (content based or item-item recommendation) for movie data.

Dataset: Movies dataset

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


movies=pd.read_csv('movies.csv')

movies.dropna(inplace=True)

movies = movies.head(1000)  # we can adjust this number if RAM allows


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


# Clean genre strings

movies['genres'] = movies['genres'].str.replace('|', ' ', regex=False)

# TF-IDF transformation

tfidf = TfidfVectorizer()

tfidf_matrix = tfidf.fit_transform(movies['genres'])


cosine_sim = cosine_similarity(tfidf_matrix)

indices = pd.Series(movies.index, index=movies['title']).drop_duplicates()


def recommend_movie(title, num_recommendations=5):

    if title not in indices:

        return "Movie not found in dataset."

    idx = indices[title]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    sim_scores = sim_scores[1:num_recommendations+1]

    movie_indices = [i[0] for i in sim_scores]

    return movies['title'].iloc[movie_indices].tolist()

print("🎥 Recommendations for 'Toy Story (1995)':")

recommendations = recommend_movie('Toy Story (1995)', 5)

print(recommendations)

for i, movie in enumerate(recommendations, 1):

    print(f"{i}. {movie}")
```

```
🎥 Recommendations for 'Toy Story (1995)':
['Pagemaster, The (1994)', 'James and the Giant Peach (1996)', 'Balto (1995)', 'Space Jam (1996)', 'Kids of the Round Table (1995)']
1. Pagemaster, The (1994)
2. James and the Giant Peach (1996)
3. Balto (1995)
4. Space Jam (1996)
5. Kids of the Round Table (1995)
```