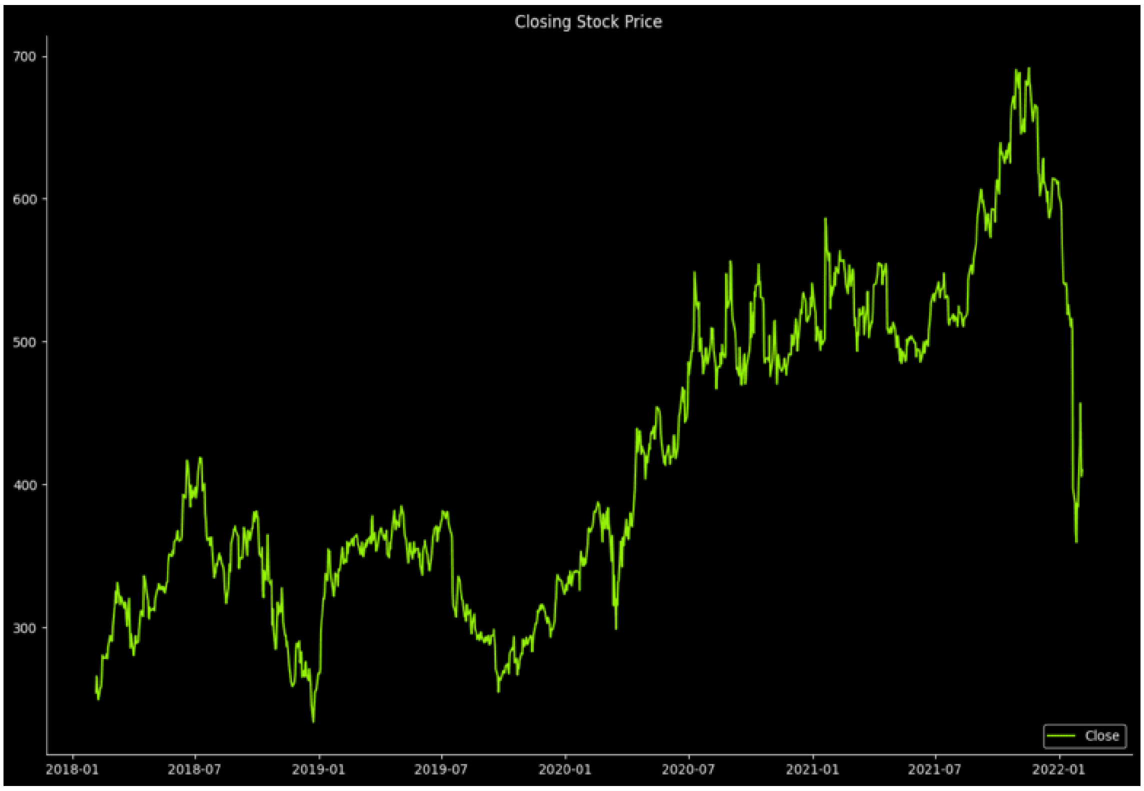
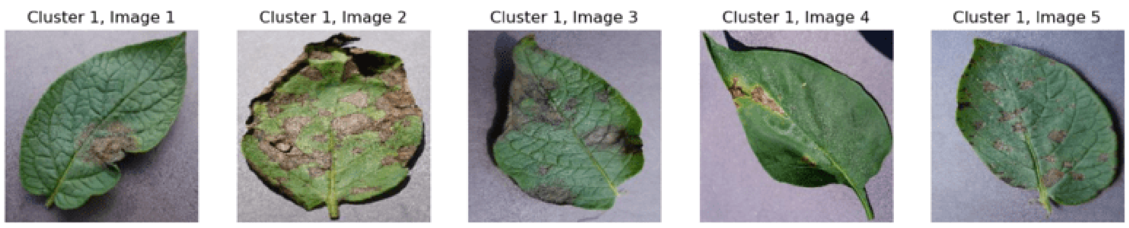
Automatic leaf infection identification & Stock Price Prediction

We Have two project in the repo the first is (Numeric) Netflex Stock Prediction, the Secound is (Image) leaf Disease identification





1) Numeric DataSet

- ***Name:*** Netflix Stock Price Prediction
- ***Link:*** https://www.kaggle.com/datasets/jainilcoder/netflix-stock-price-prediction/data
- ***No. of classes:*** 7
- ***Total no. of samples:*** 1009
- ***No. of samples in training\validation:*** 807

- ***No. of samples in testing:*** 202

  ### two Algorithm are used

  - Linear Regression
  - KNN
  - **1-Linear Regression***: Aims to find the optimal line or hyperplane in multiple dimensions that minimizes the difference between predicted and actual values.
  - **Library Used**

    ```
    import numpy as numpy
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_squared_error, mean_absolute_error,
    mean_absolute_percentage_error,r2_score
    ```

  - **Feature Extraction**

    In feature extraction, three columns were dropped:

    - Target Column (Close)
    - Adj Close
    - Date

  - **Create Linear Regression Model**

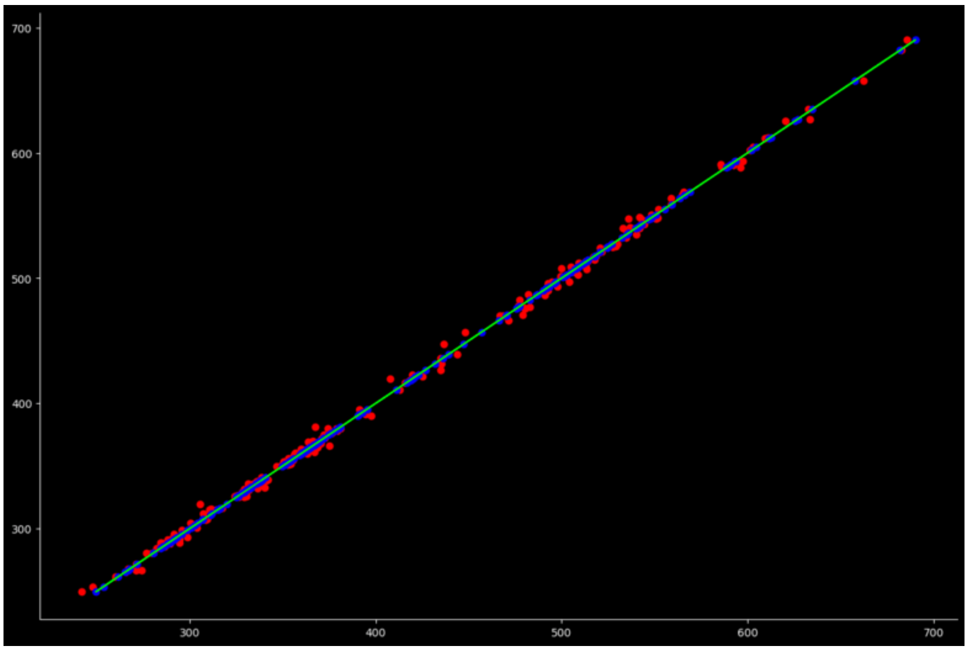
    ```
    # Set Linear Regression model with name (model_lnr)
    model_lnr = LinearRegression()
    # Fit Training data
    model_lnr.fit(x_train, y_train)
    ```

  - **Model Evalution**

    ```
    # Get accuracy of model
    print("MSE",round(mean_squared_error(y_test,y_pred), 3))
    print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
    print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
    print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
    print("R2 Score : ", round(r2_score(y_test,y_pred), 3) * 100)
    ```

    - ***MSE*** => 15.587
    - ***RMSE*** => 3.948
    - ***MAE*** => 2.998
    - ***MAPE*** => 0.007
    - ***R2 Score*** => 99.9

  - **Model Line2D Ploting**



- **Save**
    - Prediction Result is save in a new csv

    ```
    output.to_csv('Dataset/Close_Prediction.csv', index=True)
    ```

- **2-KNN***: Predicts based on the majority class or average value of K-nearest data points in the feature space for classification or regression purposes.
- **Loading and Preprocessing Data**

```
dataset = pd.read_csv('NFLX.csv')
viz = dataset.copy()

# Drop 2 column Data, Adj Close
dataset = dataset.drop(["Date","Adj Close"],axis=1)
x =
dataset.drop("Close",axis=1).values.reshape(dataset.shape[0],dataset.shape

y = dataset["Close"].values.reshape(dataset.shape[0],1)
x, y = shuffle(x,y,random_state=42)
scaler = MinMaxScaler()
x = scaler.fit_transform(x)
train, test = train_test_split(dataset,test_size = 0.2)
x_train,x_test,y_train,y_test =
train_test_split(x,y,test_size=0.2,random_state=42)
```

The data is loaded from a CSV file, analyzed, and normalized using MinMaxScaler, then split into training and testing sets for visualization.

- **Create K-Nearest Neighbors Model**

  ```
  # Using KFold for split data into 5 sample
  kf = KFold(n_splits=5, shuffle=True, random_state=42)
  param_grid = {"n_neighbors": range(1, 20)}
  knn = KNeighborsRegressor()
  knn_cv = GridSearchCV(knn, param_grid, cv=kf)
  knn_cv.fit(x_train,y_train)
  y_pred_train_knn=knn_cv.predict(x_train)
  y_pred_test_knn=knn_cv.predict(x_test)

  print("Train Score:", round(knn_cv.score(x_train,y_train)*100,2))
  print("Test Score :", round(knn_cv.score(x_test,y_test)*100, 2))
  ```

  K-Fold cross-validation is set up, hyperparameter grid for K-Nearest Neighbors defined, grid search tuned, best hyperparameters printed, model trained, predictions made, and training and testing scores printed.

- **Cross-validation Scores and Model Evalution**

  ```
  # Get accuracy of model
  cv_scores_knn = cross_val_score(knn_cv, x, y)
  print("knn's score:" ,cv)
  print("MSE",round(mean_squared_error(y_test,y_pred), 3))
  print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
  print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
  print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
  print("R2 Score : ", round(r2_score(y_test,y_pred), 3) * 100)
  ```

  - ***knn's score*** => [0.9966189 0.99526833 0.99609932 0.9958496 0.99622485]
  - ***MSE*** => 43.871
  - ***RMSE*** => 6.624
  - ***MAE*** => 4.801
  - ***MAPE*** => 0.012
  - ***R2 Score*** => 99.6

    The K-Nearest Neighbors model's cross-validation scores are calculated, and various regression metrics like Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, Mean Absolute Percentage Error, and R2 Score are calculated and printed.

- **User Input and Prediction Function**

  ```
  def predict_stock_price(open_price, high_price, low_price, volume):
    input_data = np.array([open_price, high_price, low_price, 
  volume]).reshape(1, -1)
    predicted_price = knn_cv.predict(input_data)
    return predicted_price[
  ```

- **Taking User Input and Displaying Prediction**

  ```
  # User input
  user_open = float(input("Enter the Open price: "))
  user_high = float(input("Enter the High price: "))
  user_low = float(input("Enter the Low price: "))
  user_volume = float(input("Enter the Volume: "))

  # Predict using user input
  predicted_stock_price = predict_stock_price(user_open, user_high,
  user_low, user_volume)
  print(f"Predicted Close Price: {predicted_stock_price}")
  ```

  The 'predict_stock_price' function uses user input for open, high, low, and volume prices, predicting the close price using the trained K-Nearest Neighbors model.

2) Image DataSet

- ***Name:*** PlantVillage
- ***Link:*** https://www.kaggle.com/datasets/emmarex/plantdisease
- ***No. of classes:*** 3
- ***Class Lables:***
  - Pepper__bell___Bacterial_spot
  - Potato___Early_blight
  - Potato___Late_blight
- ***Total Number of Samples Used in DataSet:*** 2997 aprox => 1000 per class
- ***Total Number of Samples Used in Training:*** 2097
- ***Total Number of Samples Used in Testing:*** 900

two Algorithm are used

- Regression
- K_Means
- **1-Logistic Regression***: aims to create a machine learning model to classify plant leaves into three disease-related categories using a dataset of bacterial spots, early and late blight.
- **Model Architecture**

  The machine learning model classifies plant diseases using a Logistic Regression classifier, extracting features from leaf images using Histogram of Oriented Gradients.
- **Model Evaluation**

  The model's performance is evaluated using standard metrics like accuracy, precision, recall, and confusion matrix, with cross-validation for robust evaluation and ROC curves for visualization.

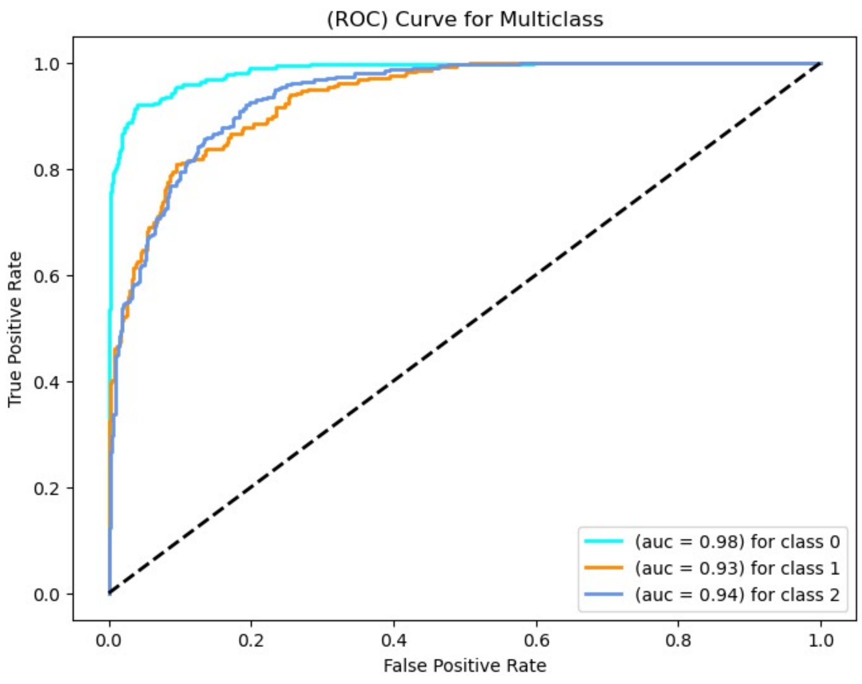
- **Steps Taken**
    - ***Data Collection:*** The task involves obtaining a dataset containing images of plant leaves with disease labels.
    - ***Data Preprocessing:*** The process involves resizing images, converting them to grayscale, and normalizing the pixel values.
    - ***Feature Extraction:*** The use of Histogram of Oriented Gradients (HOG) is employed to identify features from images.
    - ***Image Data Transformation:*** The image list is converted into a NumPy array and reshaped into a 2D array using image_data = np.array(images).reshape(len(images), -1).
    - ***Model Training:*** The task involves training a Logistic Regression model on preprocessed and feature-extracted data.
    - ***Model Evaluation:*** The model's performance is evaluated through metrics like accuracy, precision, recall, and ROC curves.
    - ***Visualization:*** The process involves analyzing ROC curves for each class and plotting the confusion matrix for multiclass classification.
    - ***Cross-Validation:*** The model performance will be evaluated through cross-validation using a Stratified K-Fold with 5 splits.
- **Feature Extraction:** The feature extraction phase involved extracting Histogram of Oriented Gradients (HOG) features from plant leaf images, identifying texture and shape details.
    - ***Number of Features Extracted:*** The HOG algorithm used 8 orientations, resulting in 8 bins per cell in the image grid, resulting in 2048 features per image.
    - ***Names of Extracted Features:*** The HOG algorithm extracts numerical representations of an image's texture and shape, collectively contributing to a descriptive description for each image.
    - ***Dimension of Resulted Features:*** The extracted features were transformed into a one-dimensional array for each image, resulting in a feature matrix with dimensions of (Number of Images, 2048) and a shape of (2997, 2048).
- **Predictions on the test set:**

  ```
  y_pred = model.predict(X_test)
  ```
- **Calculate accuracy:**

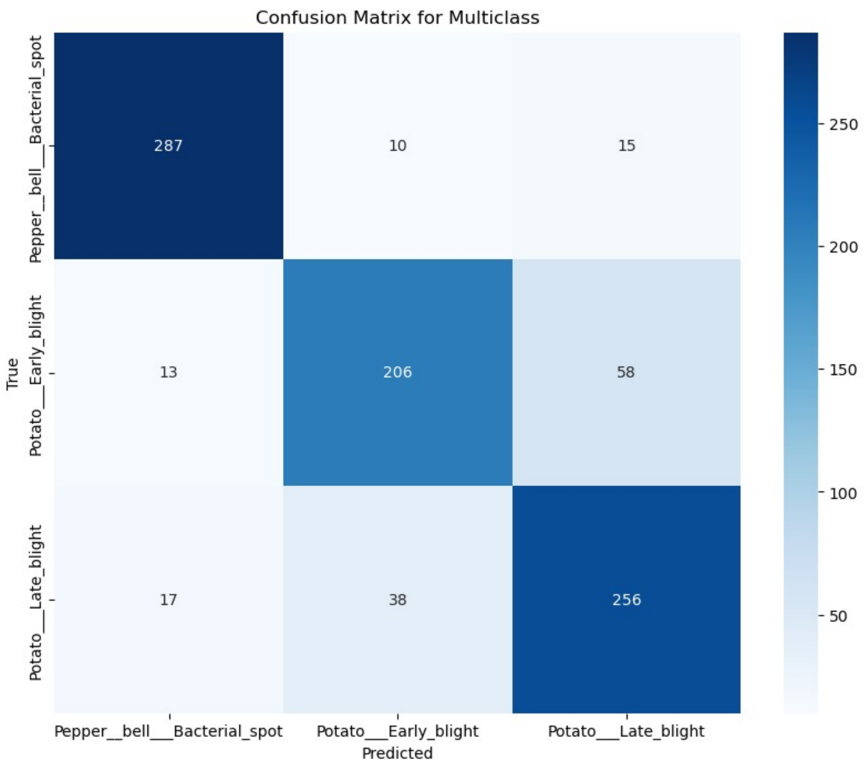
  ```
  accuracy = accuracy_score(y_test, y_pred)
  ```
- **Print the accuracy:**

  ```
  print(f"Accuracy: {accuracy:.2f}")=0.83
  {:.2f}%'.format(accuracy * 100))= 83.44 %
  ```
- **result details:**

- Roc_curve



- Confusion Matrix



- Cross-Validation Scores

```
[0.82833333, 0.85166667, 0.84140234, 0.83806344, 0.83472454]
```

    - Mean Accuracy

      ```
      Mean Accuracy: 0.8388380634390652
      ```

- **2-K-Means***: aims to create a machine learning model to cluster plant leaves into three disease-related categories using a dataset of bacterial spots, early and late blight.
- **Model Architecture**

  The machine learning model for plant disease clustering uses a K_means classifier, extracting features from Histogram color from leaf images, with 2997 samples used for training.
- **Model Evaluation**

  K-means is an unsupervised learning algorithm that can be evaluated through metrics like silhouette score, Davies-Bouldin index, and visual inspection of the clustering.
- **Steps Taken**
    - ***Data Collection:*** The task involves obtaining a dataset containing images of plant leaves.
    - ***Data Preprocessing:*** The task involves Convert to HSV color space, Compute Histograms, Concatenate Histograms, Data Type Conversion, Normalization.
    - ***Feature Extraction:*** the use of extract_histogram_features is employed to identify features from images.
    - ***Image Data Transformation:*** The image list is converted into a NumPy array and reshaped into a 2D array using image_data = np.array(images).reshape(len(images), -1).
    - ***Model Training:*** The task involves training a K-Means model on preprocessed and feature-extracted data.
    - ***Model Evaluation:*** K-means is an unsupervised learning algorithm that can be evaluated through metrics like silhouette score, Davies-Bouldin index, and visual inspection of the clustering.
    - ***Visualization:*** The process involves analyzing Elbow Method curve, plotting the clusters with centroids.
    - ***Number of Features Extracted:*** The default value for image features is 3x5=15, but the bins parameter can be adjusted to control color information granularity and feature extraction.
    - ***Dimension of Resulted Features:*** The extracted features were transformed into a one-dimensional array for each image, resulting in a feature matrix with dimensions of (Number of Images, 2997) and a shape of (2997, 15)
    - ***Data Processing:***

        - *Standardization*: Standardizes the extracted image features.
        - *Dimensionality Reduction*: Reduces feature dimensions using PCA.

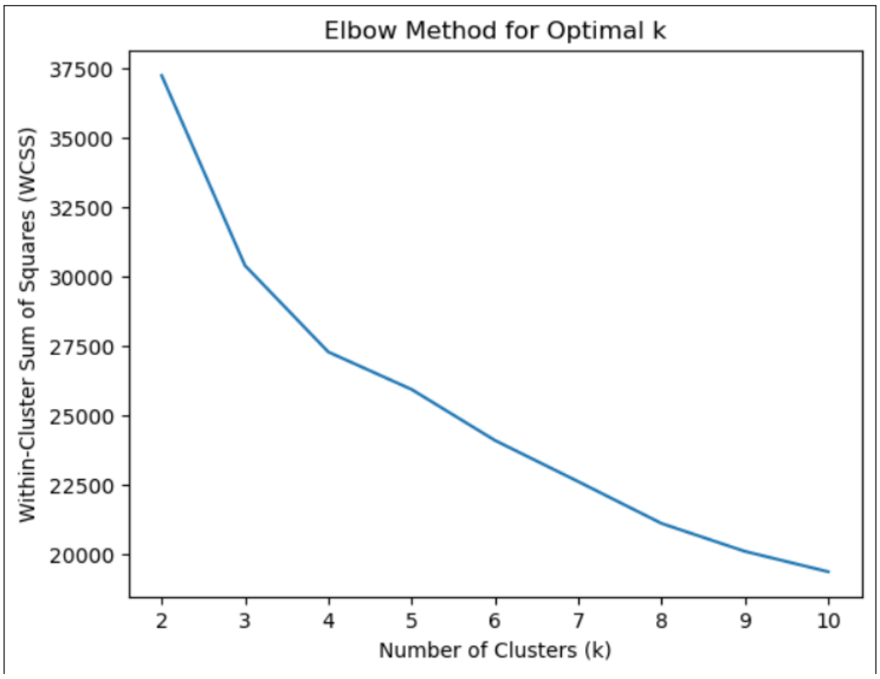
    - **Train Model:**

```
kmeans_model.fit(features_data)
```

    - **plotting the elbow graph:**

```
wcss = []

for k in range(2, 11):  # Try different values of k
    kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
    kmeans.fit(image_data_pca)
    wcss.append(kmeans.inertia_)

    plt.plot(range(2, 11), wcss)
    plt.title('Elbow Method for Optimal k')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
    plt.show()
```

        - elpow plot

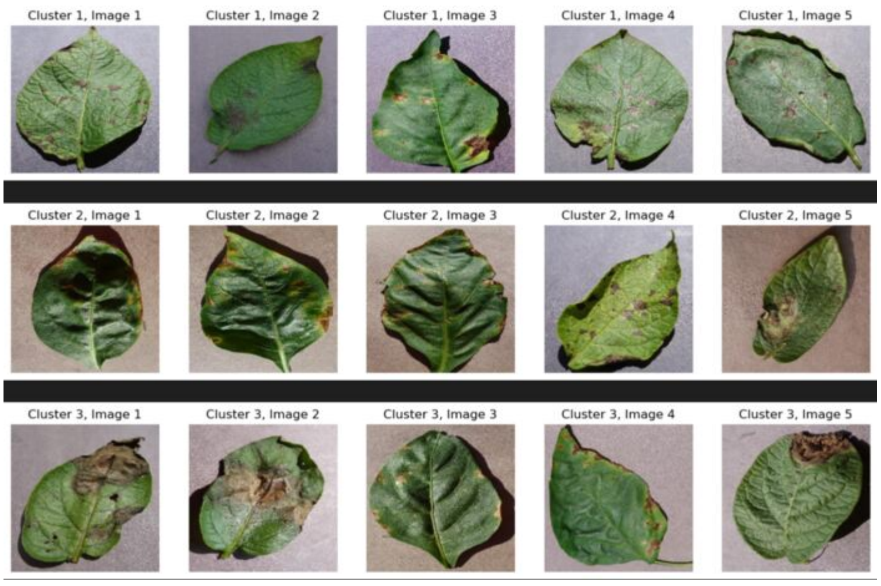


    - **plot first five images from each cluster:**

```
def load_image(image_path):
  return io.imread(image_path)

#Iterate through each cluster
```

```
for cluster_id in range(num_clusters):
    # Select the first five images in the cluster
    cluster_images = image_cluster[image_cluster['clusterid'] ==
cluster_id]['image'][:5]

    #Plot the first five images in the cluster
    plt.figure(figsize=(15, 3))
    for i, image_path in enumerate(cluster_images):
        plt.subplot(1, 5, i + 1)
        image = load_image(image_path)
        plt.imshow(image)
        plt.axis('off')
        plt.title(f'Cluster {cluster_id + 1}, Image {i + 1}')

    plt.show()
```

- Clusters Sample

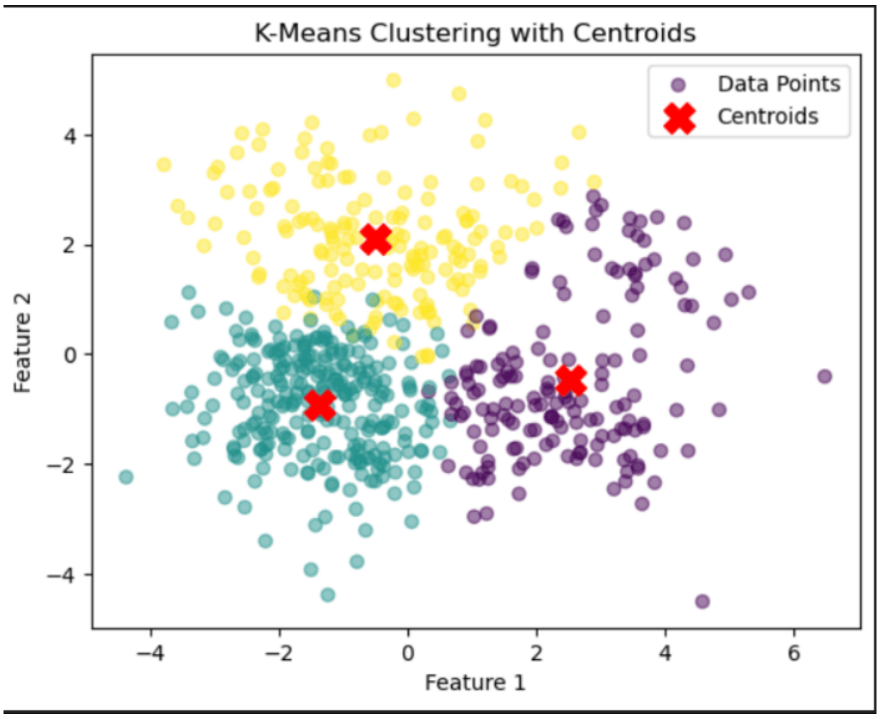


- **Calculate Average Silhouette Score & plot Clusters with Centroid:**

```
  silhouette_scores = []

  for train_index, test_index in kf.split(features_data):
      X_train, X_test = features_data[train_index],
features_data[test_index]

      # You may need to choose an appropriate number of clusters
(n_clusters) for your specific case
      kmeans = KMeans(n_clusters=3, random_state=42,n_init='auto')
      cluster_labels = kmeans.fit_predict(X_test)

      # Plotting the data points
      plt.scatter(X_test[:, 0], X_test[:, 1], c=cluster_labels,
cmap='viridis', alpha=0.5, label='Data Points')
```

```
      # Plotting the centroids
      plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], c='red', marker='X', s=200,
label='Centroids')

      plt.title('K-Means Clustering with Centroids')
      plt.xlabel('Feature 1')
      plt.ylabel('Feature 2')
      plt.legend()
      plt.show()

      # Step 5: Compute silhouette score for the current fold
      silhouette_avg = silhouette_score(X_test, cluster_labels)
      silhouette_scores.append(silhouette_avg)

  # Step 6: Average silhouette scores over all folds
  average_silhouette_score = np.mean(silhouette_scores)

  print(f"Average Silhouette Score across {k_folds} folds:
{average_silhouette_score}")
```

- Plot Sample


K-Means Clustering with Centroids
Data Points
Centroids
Feature 1
Feature 2