Numerical data set life expectancy

Key Features of the Dataset

- **Number of Columns:** 22
- **Number of Rows:** 2938
- **Target Variable (Label):**
  - `Life expectancy` (a numerical variable measuring average life expectancy in years).

Columns in the Dataset

The dataset includes:

1. **Country** (String): Name of the country.
2. **Year** (Numerical): Year of data collection.
3. **Status** (String): Whether the country is 'Developed' or 'Developing'.
4. **Life expectancy** (Numerical): Average life expectancy at birth.
5. **Adult Mortality** (Numerical): Mortality rate of adults aged 15–60 per 1000 population.
6. **Infant Deaths** (Numerical): Number of infant deaths per 1000 live births.
7. **Alcohol** (Numerical): Per capita alcohol consumption (liters per capita).
8. **Percentage expenditure** (Numerical): Healthcare expenditure as a percentage of GDP.

9. **Hepatitis B** (Numerical): Percentage of immunization coverage for Hepatitis B.
10. **Measles** (Numerical): Reported cases per 1000 population.
11. **BMI** (Numerical): Average body mass index.
12. **Under-five deaths** (Numerical): Number of deaths under age five per 1000 live births.
13. **Polio** (Numerical): Percentage of immunization coverage for Polio.
14. **Total expenditure** (Numerical): Total healthcare expenditure as % of GDP.
15. **Diphtheria** (Numerical): Percentage of immunization coverage for Diphtheria.
16. **HIV/AIDS** (Numerical): Death rate per 1000 population due to HIV/AIDS.
17. **GDP** (Numerical): Gross Domestic Product per capita.
18. **Population** (Numerical): Population size.
19. **Thinness 1-19 years** (Numerical): Percentage of thinness for 1-19 age group.
20. **Thinness 5-9 years** (Numerical): Percentage of thinness for 5-9 age group.
21. **Income composition of resources** (Numerical): Human Development Index (HDI) income component.
22. **Schooling** (Numerical): Average number of years of schooling.

Data Summary

1. **Missing Values:**
    a. This dataset has missing values in multiple columns. Key variables like GDP, `Life expectancy`, and `Schooling` may contain missing data. A detailed analysis shows thousands of missing values across the dataset.
2. **String Columns:**
    a. **2 string columns**: `Country` and `Status`.
3. **Classes/Labels:**
    a. The `Status` column can be considered a categorical variable with **2 classes**:
        i. **Developed**
        ii. **Developing**
4. **Target Variable:**
    a. **Life expectancy** is the continuous variable that the dataset aims to analyze or predict.

Summary

- Total Columns: **22**
- Missing Values: **Several missing values** across numeric variables.
- String Columns: **2** (`Country`, `Status`)
- Classes/Labels: **2** (`Developed` and `Developing`)
- Target Column: **Life expectancy**

This dataset is suitable for **predictive modeling, correlation analysis**, and exploring factors affecting **life expectancy** globally.

Here are the **steps we performed on the data**

1. Data Preparation

- **Target and Features**:
  - You defined the **target** variable (`Life expectancy `) and identified the **features** (all columns except the target).

```
target = 'Life expectancy '
features = [col for col in data.columns if col != target]
```

- **Drop Missing Target Values**:
  - Rows where the **target** value (`Life expectancy`) is missing were removed.
  - This ensures the model has valid target values for training.

```
data_cleaned = data.dropna(subset=[target])
```

- **Separate Features and Target**:
  - You split the cleaned data into:
    - X (features)
    - y (target variable).

```
X = data_cleaned[features]
y = data_cleaned[target]
```

2. Identify Column Types

- You separated the columns into **numerical** and **categorical** features:
  - **Numerical**: Columns with `int64` or `float64` data types.
  - **Categorical**: Columns with `object` (strings).

```
numerical_cols = X.select_dtypes(include=['float64',
'int64']).columns.tolist()
categorical_cols =
X.select_dtypes(include=['object']).columns.tolist()
```

3. Data Preprocessing

- You created **two pipelines** for preprocessing the **numerical** and **categorical** features.

a) Numerical Preprocessing

- Steps:
  - Replace missing values with the **mean** (`SimpleImputer`).
  - Scale numerical values using **StandardScaler** for normalization (mean = 0, std = 1).

```
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())
])
```

b) Categorical Preprocessing

- Steps:
  - Replace missing values with the **most frequent value** (`SimpleImputer`).
  - Convert categorical values to binary columns using **OneHotEncoder**.

```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

c) Combine Both Preprocessors

- You combined the numerical and categorical preprocessing pipelines using a **ColumnTransformer**.
  - This applies different transformations to different columns.

```
preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, numerical_cols),
    ('cat', categorical_transformer, categorical_cols)
])
```

4. Split Data into Training and Testing Sets

- You split the data into **training** (80%) and **testing** (20%) sets using `train_test_split`.

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
```

5. Train and Evaluate Models

a) Linear Regression

- **Model Pipeline**: Combined the **preprocessor** and the **Linear Regression** model.
- **Training**: You fit the model to the training data.
- **Prediction**: Predicted values on the test set.
- **Evaluation**: Calculated metrics:
    - Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score.

```
linear_model = Pipeline(steps=[('preprocessor', preprocessor),
                               ('regressor', LinearRegression())])
linear_model.fit(X_train, y_train)
y_pred_lr = linear_model.predict(X_test)
```

b) K-Nearest Neighbors (KNN) Regressor

- **Model Pipeline**: Combined the **preprocessor** and the **KNN Regressor** model.
- **Training**: You fit the KNN model to the training data.
- **Prediction**: Predicted values on the test set.
- **Evaluation**: Used the same metrics (MAE, MSE, RMSE, R^2 score).

```
knn_model = Pipeline(steps=[('preprocessor', preprocessor),
                            ('regressor',
KNeighborsRegressor(n_neighbors=5))])
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

6. Model Comparison

- You compared both models (Linear Regression and KNN Regressor) using:
    - **RMSE** (Root Mean Squared Error): Lower is better.
    - **R^2 Score**: Higher is better.

```
print(f"Linear Regression - RMSE: {rmse_lr}, R²: {r2_lr}")
print(f"KNN Regressor - RMSE: {rmse_knn}, R²: {r2_knn}")
```

- Linear Regression performed better (higher R^2 and lower RMSE).

7. Visualization of Results

- You plotted the predicted values against the actual values for both models to visually assess their performance.

Summary of Steps

1. **Data Cleaning**: Removed rows with missing target values.
2. **Feature Identification**: Identified numerical and categorical columns.
3. **Data Preprocessing**: Applied transformations: imputation, scaling, and encoding.
4. **Train-Test Split**: Split the data into training and testing sets.
5. **Model Training**:
    a. Trained **Linear Regression** and **KNN Regressor** models.
6. **Model Evaluation**: Compared performance using RMSE and R^2.
7. **Visualization**: Plotted predicted vs. actual values.

This workflow is a **standard pipeline** for working with any dataset for regression problems.

The code you provided uses two machine learning algorithms to analyze and predict **Life Expectancy** based on the given dataset:

1. Linear Regression

Type: Supervised Machine Learning Algorithm for **Regression**

- **Goal**: Predict a continuous target variable (e.g., Life Expectancy) based on input features.
- **Assumption**: It assumes a **linear relationship** between the independent variables (features) and the dependent variable (target).

How It Works:

The Linear Regression model fits a line that minimizes the sum of squared errors (residuals) between the actual and predicted target values.

The line is defined as:

Y=w0+w1X1+w2X2+⋯+wnXnY = w_0 + w_1X_1 + w_2X_2 + \dots + w_nX_n

Where:

- YY is the target (predicted value).
- X1,X2,...,XnX_1, X_2, ..., X_n are the input features.
- w0w_0 is the intercept, and w1,w2,...,wnw_1, w_2, ..., w_n are the coefficients/weights.

Advantages:

- Simple and easy to interpret.
- Works well if the relationship between variables is linear.

Limitations:

- Assumes a linear relationship, which may not hold for complex datasets.

- Sensitive to outliers.

2. K-Nearest Neighbors (KNN) Regressor

Type: Supervised Machine Learning Algorithm for **Regression**

- **Goal**: Predict a continuous target variable by finding the average of the **K-nearest neighbors** in the feature space.
- It is a **non-parametric algorithm** and does not assume any specific relationship between features and the target.

How It Works:

- **Distance-Based Approach**:
    - For each test data point, the algorithm calculates the distance (e.g., Euclidean distance) to all training points.
    - It selects the **K closest points** (neighbors) based on this distance.
- **Prediction**:
    - The predicted value for the test point is the **average** (mean) of the target values of its neighbors.

Steps in KNN:

1. Choose the number of neighbors (KK). In your code, $K=5K = 5$.
2. Calculate the distance between the test point and all training points.
3. Find the KK-nearest neighbors.
4. Compute the average of the target values of those neighbors as the prediction.

Advantages:

- Simple and intuitive.
- Can model non-linear relationships because it does not assume linearity.

Limitations:

- **Computationally expensive** for large datasets since distances are calculated for every point.
- Sensitive to irrelevant features and feature scaling.
- Choosing the right value of KK is critical.

Comparison Between the Two Algorithms

Feature	Linear Regression	KNN Regressor
Assumption	Linear relationship.	No assumption (non-parametric).
Performance	Fast and efficient.	Slower for large datasets.
Complexity	Simple, interpretable.	Can model complex relationships.
Outliers	Sensitive to outliers.	Less sensitive to outliers.
Training Time	Very fast.	Slower (distance computation).
Prediction Time	Fast.	Slow for large datasets.

Why Use Both Algorithms?

- You evaluate both **Linear Regression** and **KNN Regressor** to see which performs better on this dataset.
- Linear Regression assumes a **linear relationship**, while KNN can capture **non-linear patterns**.
- By comparing metrics (e.g., RMSE, R^2), you determine which model fits the data better.

For your dataset:

- **Linear Regression** achieved an R^2 score of **95.16%** (better performance).
- **KNN Regressor** achieved an R^2 score of **89.31%**.

This suggests that a linear model works well for predicting **Life Expectancy** based on the features provided.

Evaluation metrics :

Linear:

Mean Absolute Error: 1.2863764459290652

Mean Squared Error: 4.3227455186545

Root Mean Squared Error: 2.079121333317154

R Squared: 0.9516170600683226

Knn:

Mean Absolute Error For KNN: 1.9182935153583625

Mean Squared Error For KNN: 9.554748122866897

Root Mean Squared Error For KNN: 3.0910755608472105

R Squared For KNN: 0.893057131747402

Comparison Between Linear Regression and KNN Regressor:

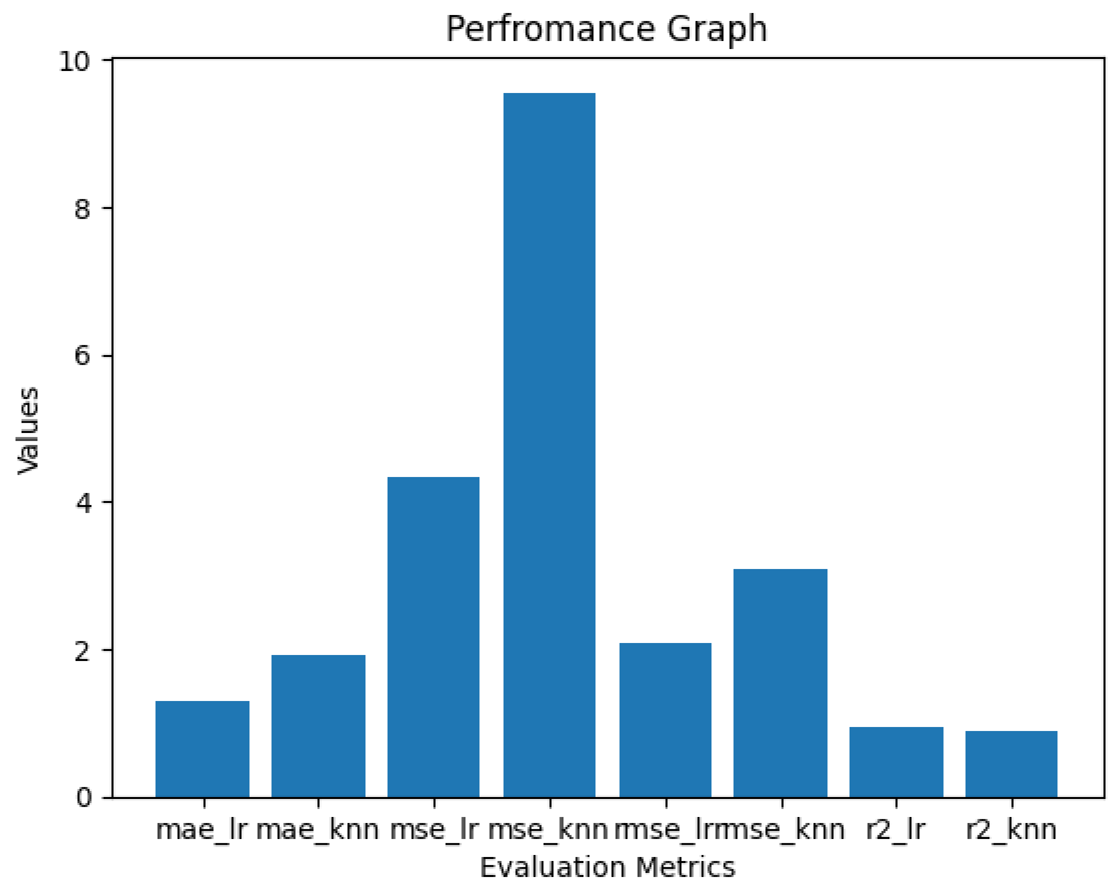
Linear Regression - RMSE: 2.079121333317154,

R^2: 0.9516170600683226

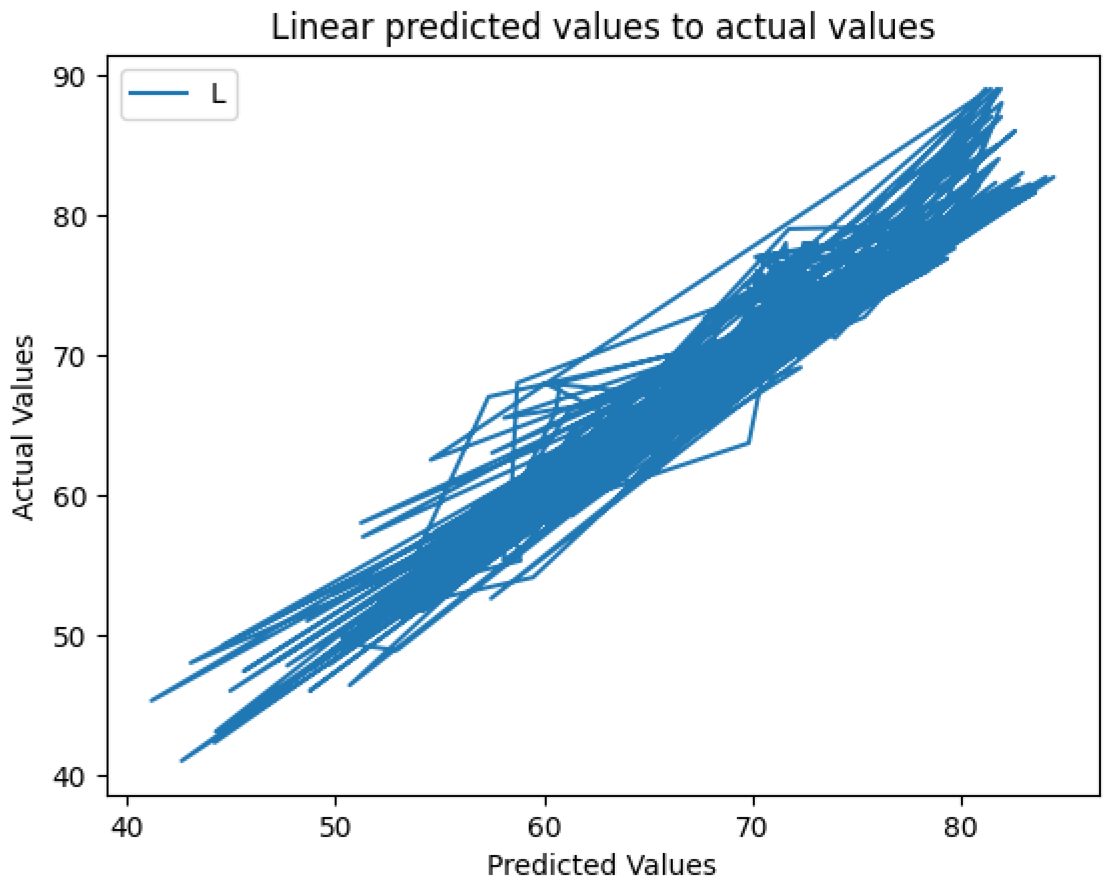
KNN Regressor - RMSE: 3.0910755608472105,

R^2: 0.893057131747402

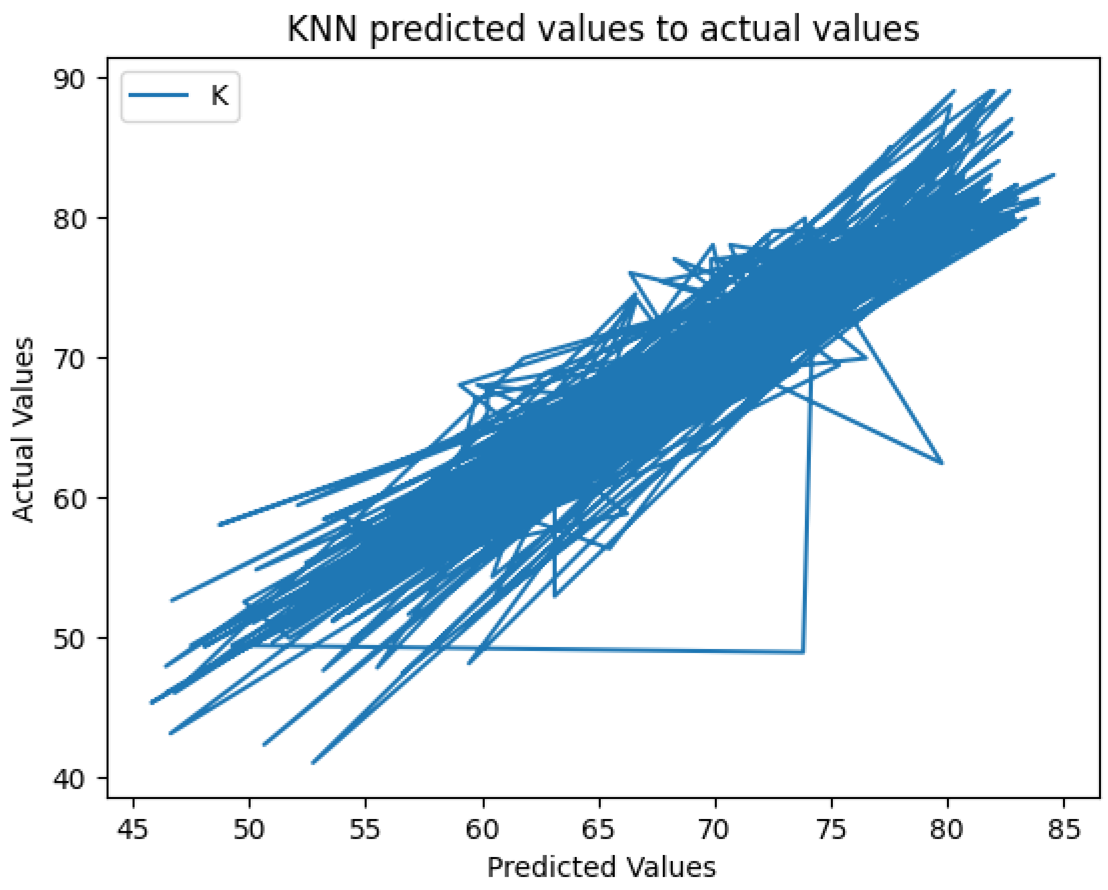
Linear Regression performs better based on lower RMSE and higher R^2.



Linear predicted values to actual ones:



Knn predicted to actual :



Then image dataset:plant village

The **PlantVillage Dataset** consists of **38 classes**, representing various plant diseases and healthy plant images. It includes a variety of crops, such as tomatoes, potatoes, and corn. The dataset has **87,000+ labeled images**, making it suitable for training models to detect plant diseases automatically. The labels distinguish between healthy and diseased plants, each class representing a specific disease or condition. This dataset is widely used for research and machine learning applications in agriculture.

The code provided implements various machine learning algorithms and data preprocessing techniques for classifying plant diseases based on images. Below is a detailed explanation of each part of the code, the algorithms used, and the classes involved in the task.

Classes Worked On:

The dataset involves multiple plant diseases and healthy plant categories. The selected classes for this task are:

1. **Tomato__Tomato_mosaic_virus**
2. **Potato___Early_blight**
3. **Tomato_Early_blight**
4. **Tomato_healthy**
5. **Pepper__bell___healthy**

These classes represent the different conditions (diseased or healthy) for each plant type.

Steps Involved in the Code:

1. Loading and Preprocessing the Data:

- **Function `load_selected_classes()`**: This function loads images from folders corresponding to the selected plant disease classes.
  - **Inputs**:
    - `folder_path`: Path to the root directory where plant disease folders are located.
    - `selected_classes`: A list of classes to include in the dataset.
  - **Output**: Returns two arrays:
    - `images`: Flattened grayscale images of size 64x64 pixels.
    - `labels`: Corresponding labels for each image indicating which plant disease class it belongs to.

The function converts each image to grayscale, resizes it, and flattens it into a 1D array for machine learning processing.

2. Data Splitting:

- The data is split into training and testing sets using **`train_test_split`**. The test size is 20%, and the split is random (with a fixed `random_state` for reproducibility).

3. ***Standardization (Scaling):***

- **`StandardScaler()`**: This is used to standardize the features. Standardization scales the features to have zero mean and unit variance. This is important because machine learning models perform better when the data is standardized.
    - The scaler is fit on the training set and applied to both the training and test sets.

4. ***Label Encoding:***

- **`LabelEncoder()`**: This converts categorical labels (non-numeric class names like "Tomato_healthy", "Pepper_bell_healthy") into numeric values, which can be fed into machine learning models.

5. ***Dimensionality Reduction (PCA):***

- **Principal Component Analysis (PCA)**: PCA is applied to reduce the number of features in the dataset while retaining the most important ones.
    - **n_components=100**: The number of principal components to retain is set to 100. This helps in reducing the dimensionality and computation time while still keeping the essential information.

6. ***Machine Learning Algorithms:***

A. K-Nearest Neighbors (KNN) Classifier:

- **KNeighborsClassifier(n_neighbors=3)**: KNN is a simple and effective machine learning algorithm used for classification.
    - **How it works**: It classifies a data point by majority voting from its k nearest neighbors in the feature space.
    - **Steps**:
        - The model is trained on the training set (`X_train`, `y_train`).
        - The trained model predicts labels for the test set (`X_test`).
        - Various evaluation metrics (accuracy, precision, recall, F1-score) are printed, and a confusion matrix is displayed.

        - A ROC curve is plotted for visualizing model performance for multi-class classification.

B. Logistic Regression:

- **LogisticRegression(max_iter=2000, multi_class='ovr')**: Logistic regression is a linear model used for classification tasks.
    - **How it works**: It models the probability of the target variable belonging to each class using the logistic function.
    - **Steps**:
        - The model is trained on the training set (`X_train`, `y_train`).
        - Predictions are made on the test set (`X_test`).
        - Evaluation metrics such as accuracy, precision, recall, and confusion matrix are displayed.
        - A ROC curve is plotted to evaluate the performance of the logistic regression model.

7. *Evaluation Metrics:*

- **Accuracy**: Measures the percentage of correct predictions. Formula: *Accuracy=Correct PredictionsTotal Predictions\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}*
- **Precision**: Measures the proportion of true positive predictions out of all positive predictions.
- **Recall**: Measures the proportion of true positive predictions out of all actual positive instances.
- **F1-Score**: Harmonic mean of precision and recall.
- **Confusion Matrix**: Displays the performance of the classifier, showing true positives, true negatives, false positives, and false negatives.

8. *ROC Curve and AUC:*

- **ROC Curve**: A graphical representation of the classifier's ability to distinguish between classes at different thresholds.
- **AUC (Area Under the Curve)**: Measures the classifier’s performance; the higher the AUC, the better the model.

9. ***Loss Calculation:***

- **Log Loss**: Measures the performance of the classification model where the output is a probability value between 0 and 1. It penalizes wrong predictions with a higher penalty for confident wrong predictions.
    - The log loss is calculated for both KNN and Logistic Regression models.

10. ***Hyperparameter Tuning and Loss Curves:***

- **Tuning KNN (k values)**:
    - The loss curve is generated by testing KNN classifiers with different values of k (number of neighbors). The model's misclassification rate (1 - accuracy) and log loss are plotted for each k value.
- **Tuning Logistic Regression (`max_iter` values)**:
    - The loss curve is generated by testing Logistic Regression with different values of `max_iter` (maximum number of iterations). The training and validation accuracy are plotted for each iteration value.

Algorithms Explained:

1. **K-Nearest Neighbors (KNN)**:
    a. **Type**: Instance-based learning (non-parametric).
    b. **How it works**: For a given test point, KNN finds the k closest points from the training data, and assigns the majority class label. It does not require a model to be built beforehand, hence is considered "lazy learning".
    c. **Evaluation**: The evaluation includes accuracy, precision, recall, F1-score, confusion matrix, and ROC curve.
2. **Logistic Regression**:
    a. **Type**: Linear model for binary or multi-class classification.
    b. **How it works**: Logistic regression uses the logistic function (sigmoid) to output a probability that the input data belongs to a particular class. It is commonly used for classification tasks.

c. **Evaluation**: The model is evaluated using accuracy, precision, recall, F1-score, confusion matrix, and ROC curve. Additionally, log loss is used to assess the quality of the predicted probabilities.

Summary of Key Concepts and Algorithms:

1. **Data Preprocessing**: Loading, resizing images, converting to grayscale, flattening, and scaling features.
2. **Model Training and Evaluation**: KNN and Logistic Regression are used to classify plant diseases, evaluated using metrics like accuracy, precision, recall, and F1-score.
3. **Dimensionality Reduction**: PCA is used to reduce the dimensionality of the feature space.
4. **Loss and Hyperparameter Tuning**: Log loss is used to measure model performance, and hyperparameter tuning is performed for KNN (k values) and Logistic Regression (`max_iter`).

This approach aims to classify plant diseases efficiently while evaluating the model's performance using different metrics and visualizations.

Evaluation metrics :

Class	Precision	Recall	F1-Score	Support
Tomato__Tomato_mosaic_virus	0.82	0.93	0.87	284
Potato___Early_blight	0.96	0.87	0.91	201
Tomato_Early_blight	0.91	0.64	0.75	196
Tomato_healthy	0.57	0.76	0.65	68
Pepper__bell___healthy	0.89	0.93	0.91	340
Accuracy	**0.86**			1089
Macro avg	0.83	0.83	0.82	1089
Weighted avg	0.87	0.86	0.86	1089

This table summarizes the model's performance, and the metrics you mentioned are part of the **classification report**.

Explanation of Metrics:

- **Precision**: The proportion of true positive predictions among all positive predictions.
- **Recall**: The proportion of true positives among all actual positive instances.
- **F1-Score**: The harmonic mean of precision and recall.
- **Accuracy**: The proportion of correct predictions over the total predictions.
- **Support**: The number of true instances for each class.

These metrics help evaluate the model's effectiveness across various classes and are crucial for understanding its overall performance.

Knn:

KNN Classifier Accuracy: 0.8567493112947658
Precision For KNN 0.8307257950021227

Recall For KNN 0.8270330145932663

loss of KNN: 2.8633534654991872

loss of KNN: 0.14325068870523416

Logistic:

Logistic Regression Accuracy: 0.8384

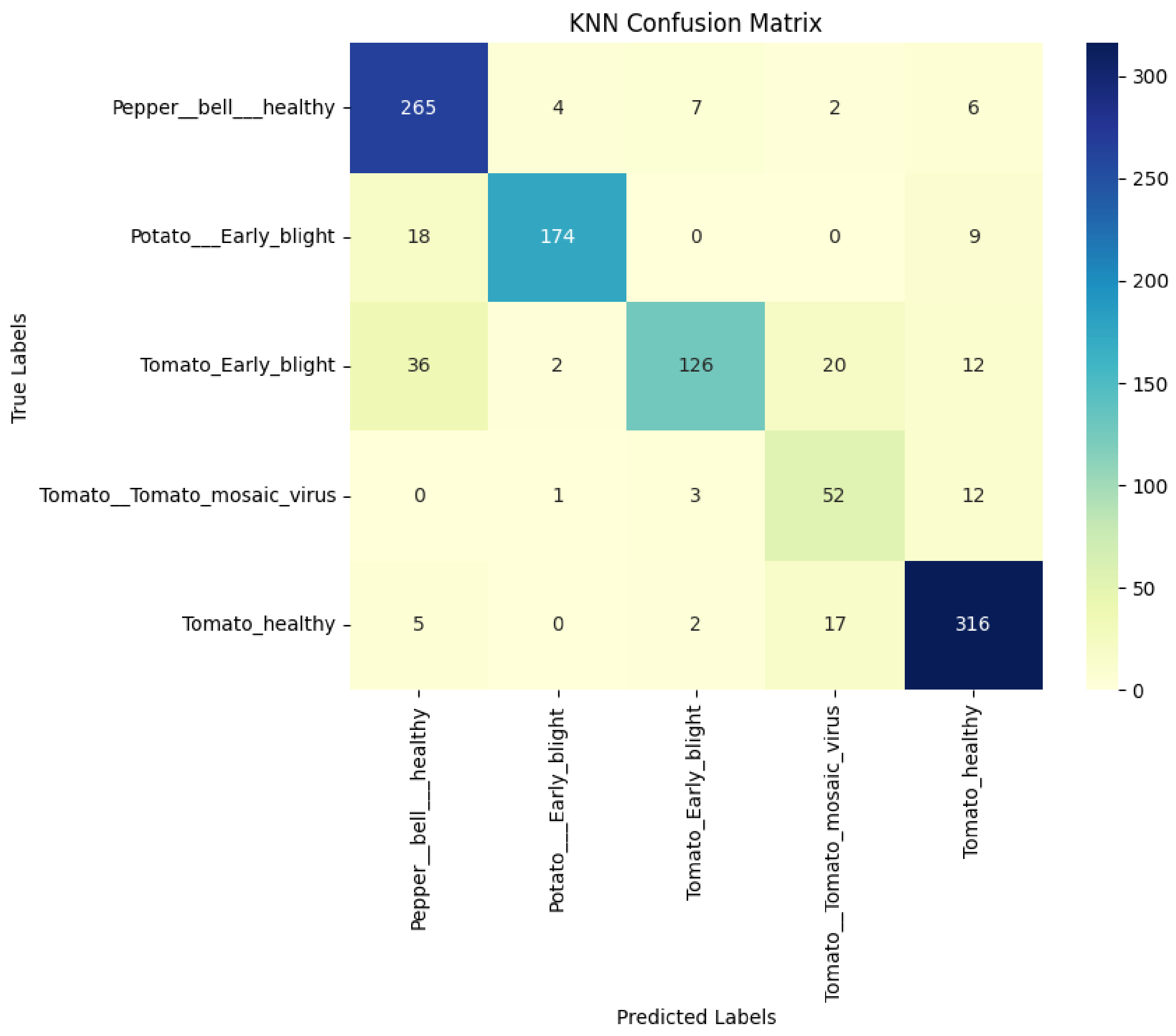
Precision For Logistic Regression 0.837664515375323

Recall For Logistic Regression 0.8080921004096405

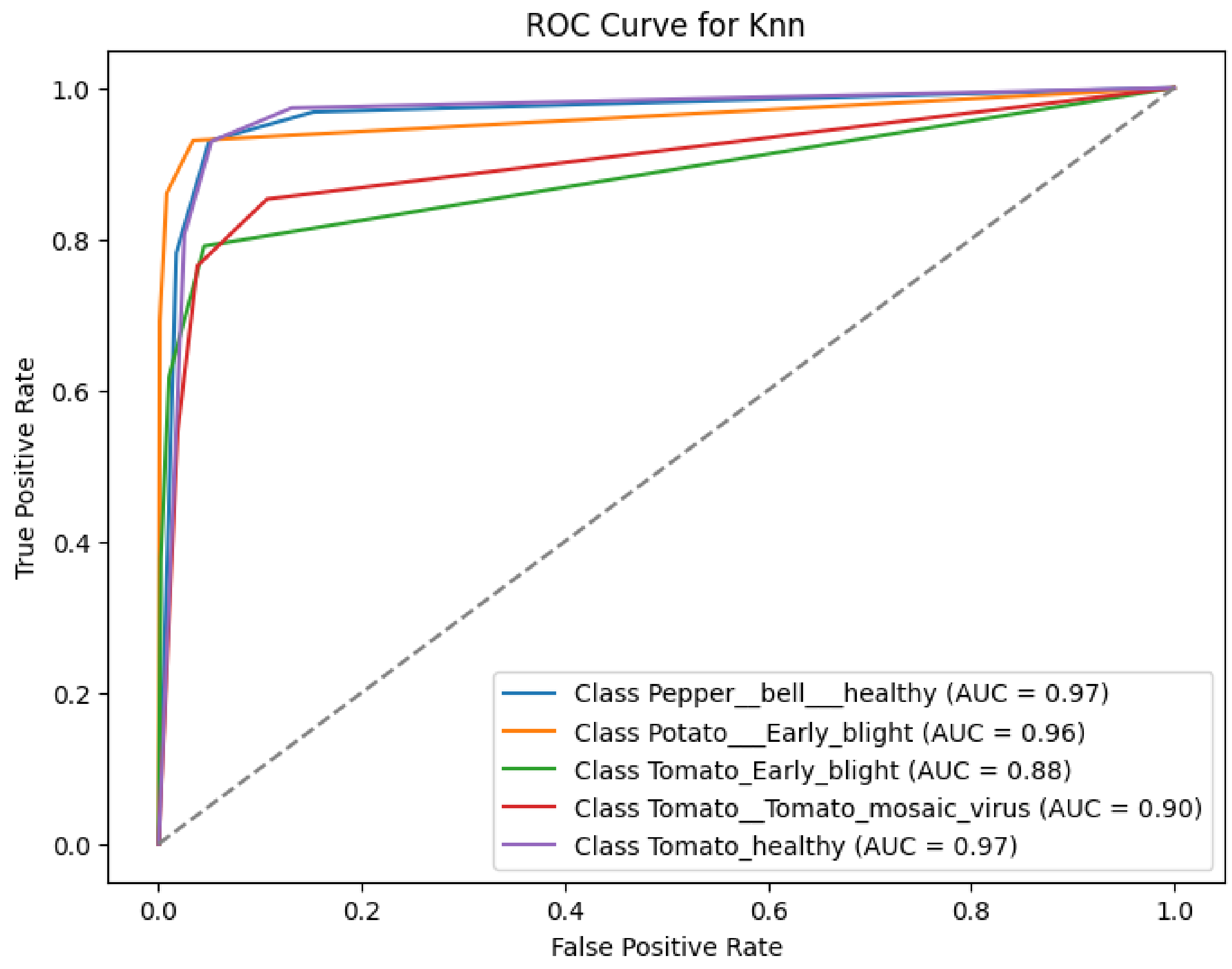
log loss of Logistic Regression: 0.5322898520503058

log loss of Logistic Regression: 0.16161616161616166

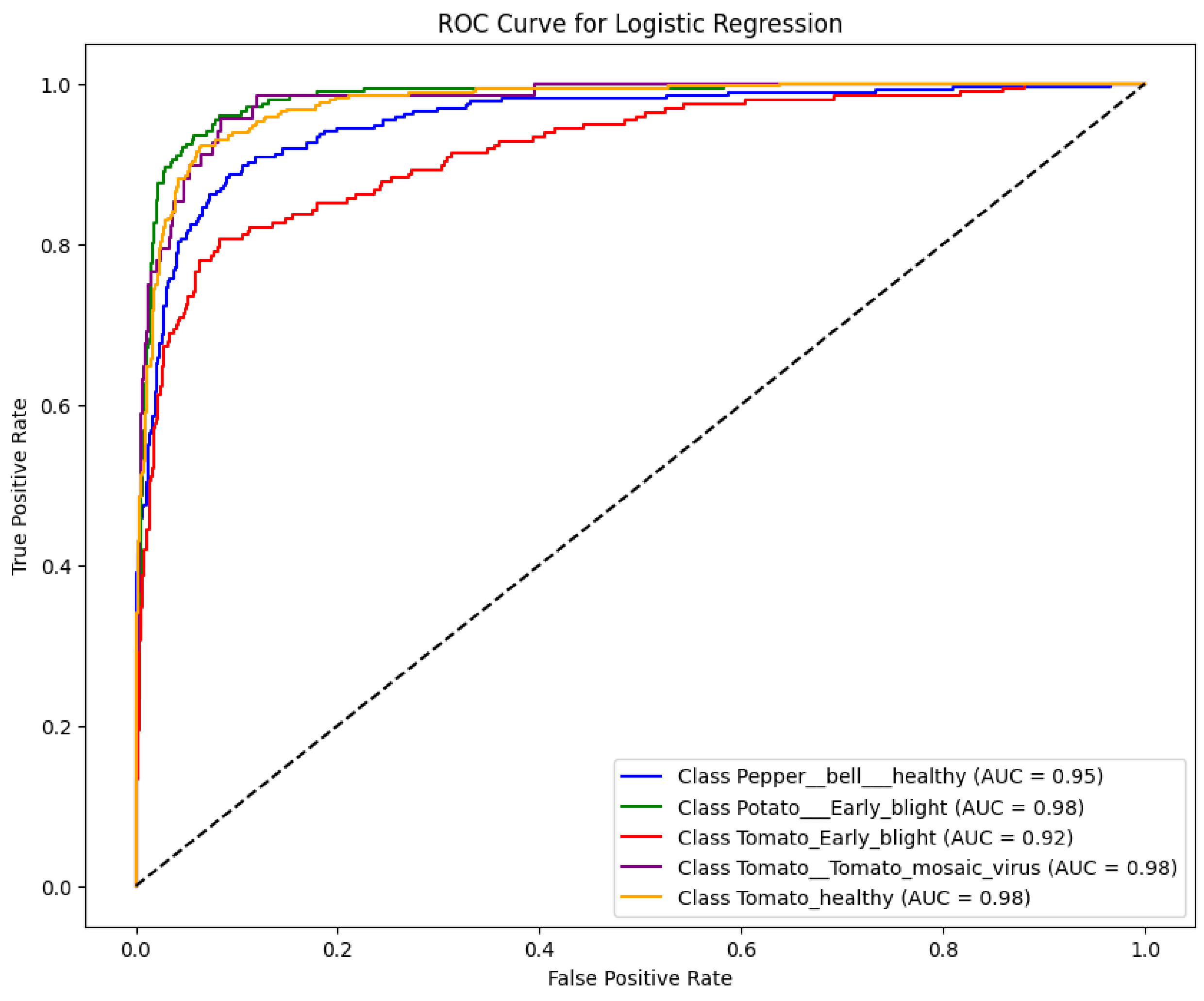
Graphs :

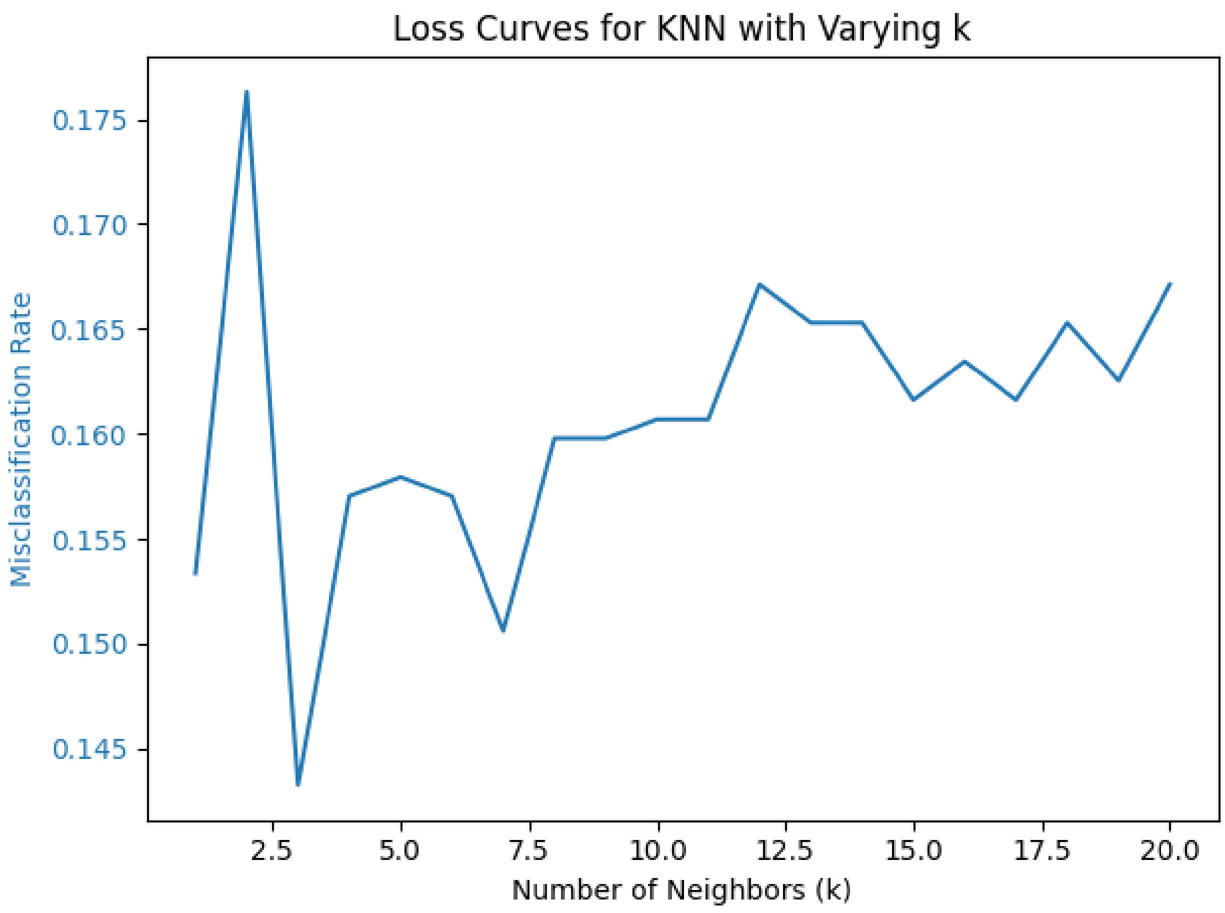


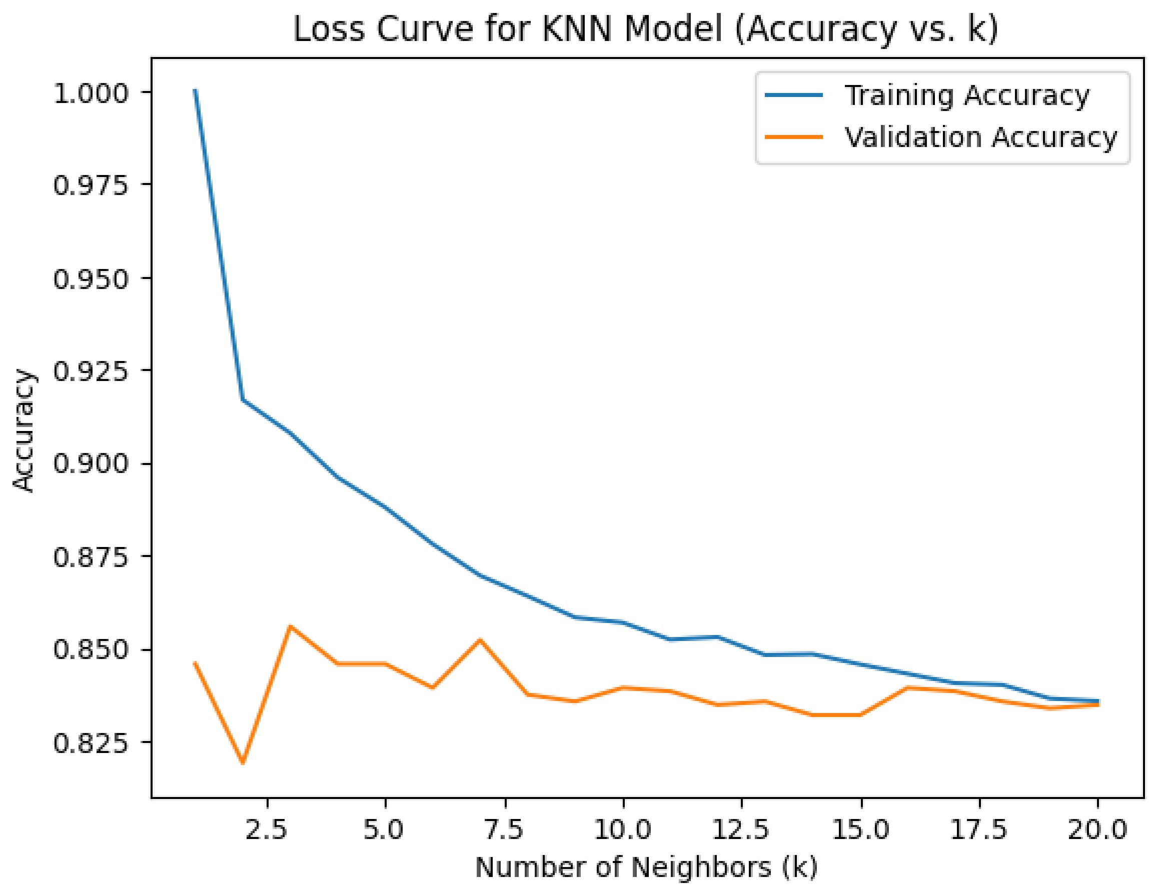
	Pepper__bell___healthy	Potato___Early_blight	Tomato_Early_blight	Tomato__Tomato_mosaic_virus	Tomato_healthy
Pepper__bell___healthy	265	4	7	2	6
Potato___Early_blight	18	174	0	0	9
Tomato_Early_blight	36	2	126	20	12
Tomato__Tomato_mosaic_virus	0	1	3	52	12
Tomato_healthy	5	0	2	17	316

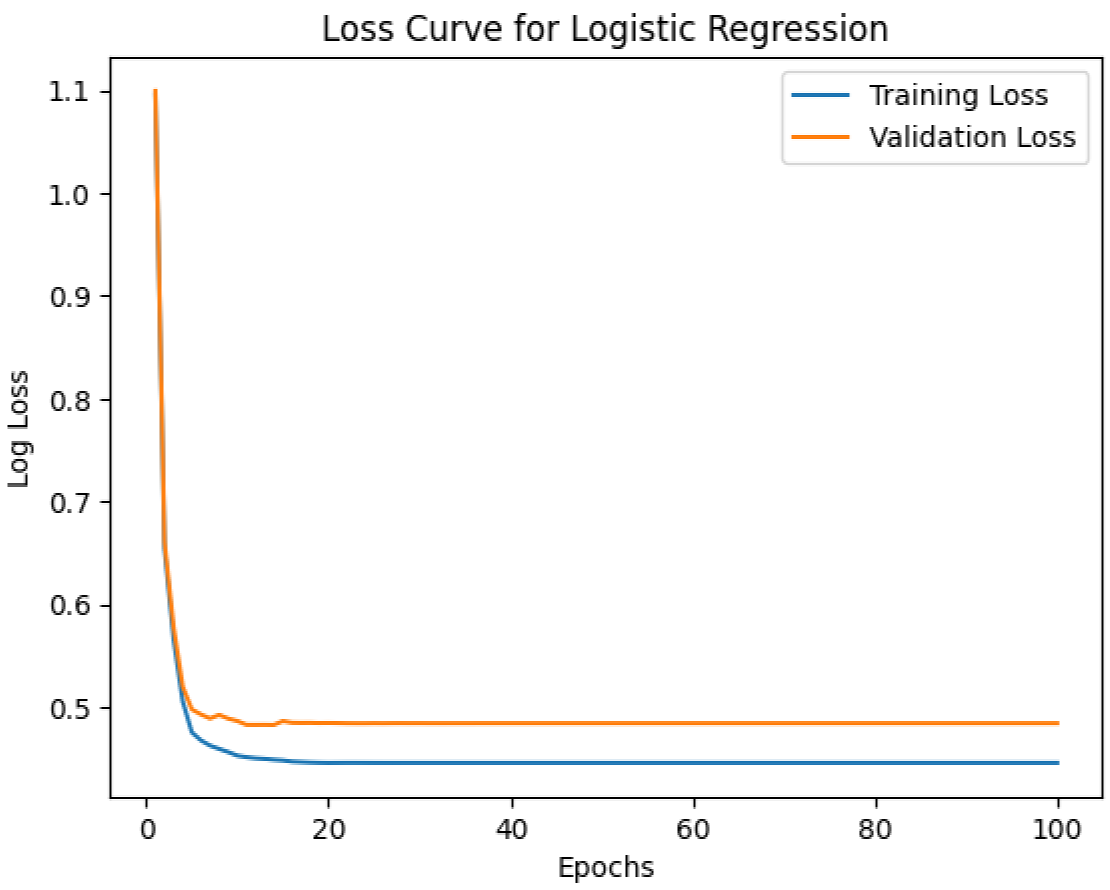


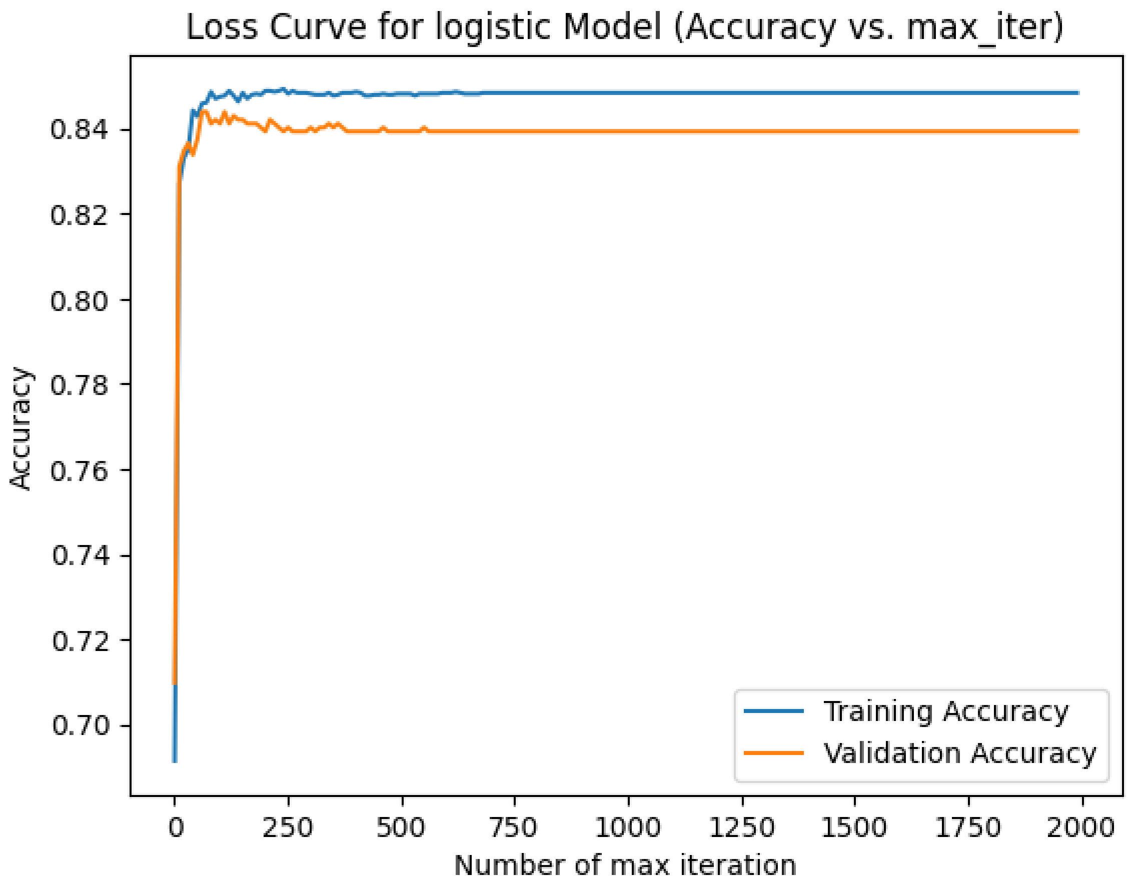
ROC Curve for Knn
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class Pepper__bell___healthy (AUC = 0.97)
Class Potato___Early_blight (AUC = 0.96)
Class Tomato_Early_blight (AUC = 0.88)
Class Tomato__Tomato_mosaic_virus (AUC = 0.90)
Class Tomato_healthy (AUC = 0.97)

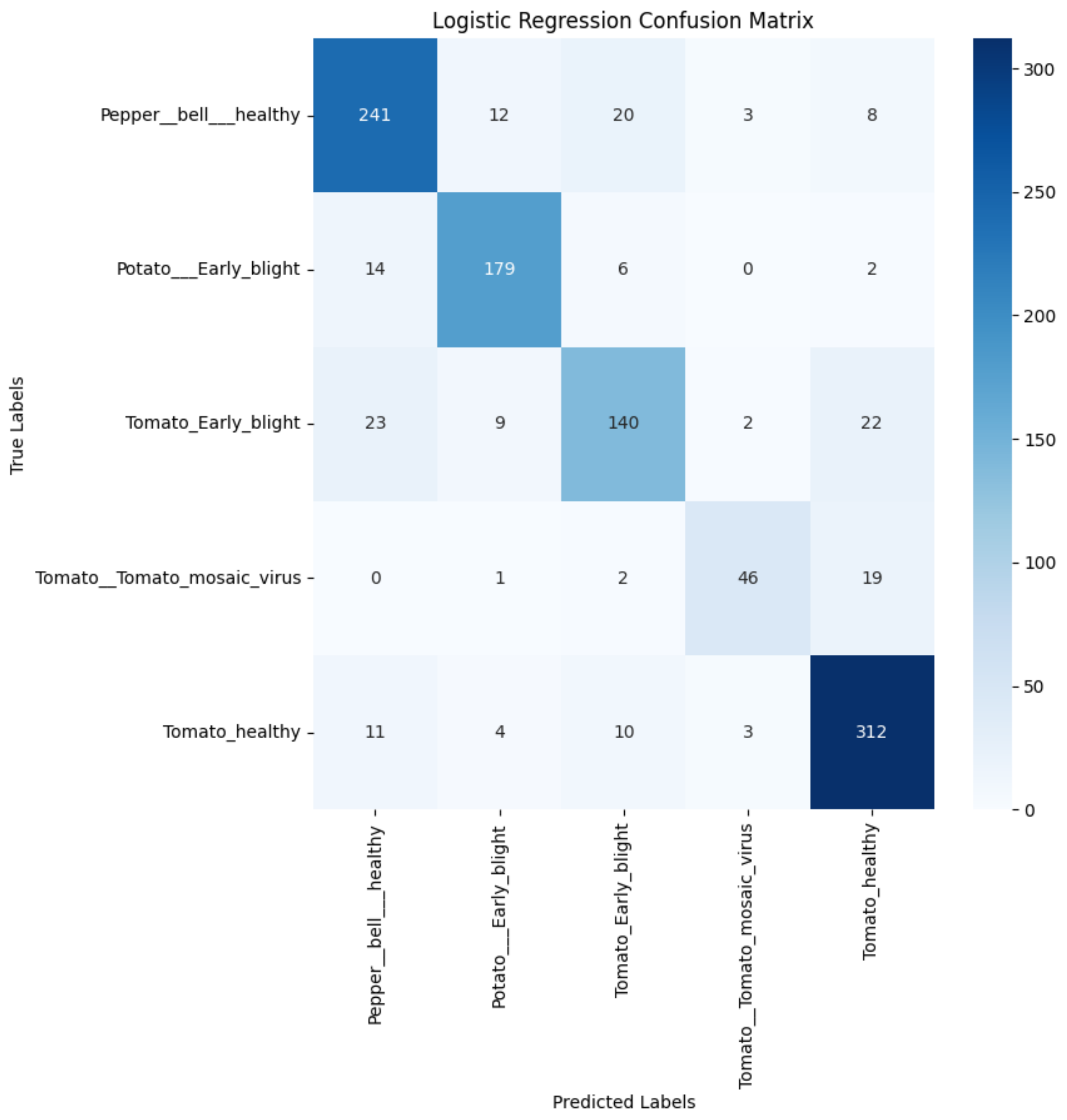

ROC Curve for Logistic Regression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Class Pepper__bell___healthy (AUC = 0.95)
Class Potato___Early_blight (AUC = 0.98)
Class Tomato_Early_blight (AUC = 0.92)
Class Tomato__Tomato_mosaic_virus (AUC = 0.98)
Class Tomato_healthy (AUC = 0.98)


Loss Curves for KNN with Varying k
Misclassification Rate
0.145
0.150
0.155
0.160
0.165
0.170
0.175
Number of Neighbors (k)
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Loss Curve for KNN Model (Accuracy vs. k)
Training Accuracy
Validation Accuracy
Accuracy
Number of Neighbors (k)
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Loss Curve for Logistic Regression
Training Loss
Validation Loss
Log Loss
Epochs
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0
20
40
60
80
100


Loss Curve for logistic Model (Accuracy vs. max_iter)
Accuracy
0.84
0.82
0.80
0.78
0.76
0.74
0.72
0.70
0
250
500
750
1000
1250
1500
1750
2000
Number of max iteration
Training Accuracy
Validation Accuracy



	Pepper__bell___healthy	Potato___Early_blight	Tomato_Early_blight	Tomato__Tomato_mosaic_virus	Tomato_healthy
Pepper__bell___healthy	241	12	20	3	8
Potato___Early_blight	14	179	6	0	2
Tomato_Early_blight	23	9	140	2	22
Tomato__Tomato_mosaic_virus	0	1	2	46	19
Tomato_healthy	11	4	10	3	312