Objective:

The objective of this project is to determine the type of Eryhemato-Squamous Disease using a comprehensive dataset. The primary goals include preprocessing the data to ensure its quality and completeness, handling any missing values, and applying appropriate feature selection and elimination techniques to optimize the dataset. The project aims to evaluate the performance of various classification models, such as Logistic Regression, Naive Bayes, Multiple Linear Regression, and Decision Tree, in accurately predicting the type of Eryhemato-Squamous Disease. The accuracy, precision, recall, and F1-score of each model will be assessed, and the ROC curves will be plotted to analyze the trade-off between true positive rate and false positive rate. Additionally, the project seeks to identify the most influential features contributing to the prediction of Eryhemato-Squamous Disease. The ultimate objective is to develop a reliable classification model that can assist in diagnosing and distinguishing different types of Eryhemato-Squamous Disease, thereby improving patient care and treatment outcomes.

Python code for Implement the given task:

Import necessary Libraries

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

Load the data

```
df=pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/dermatology/dermatology.data",header=None)
```

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	25	26	27	28	29	30	31	32	33	34
0	2	2	0	3	0	0	0	0	1	0	...	0	0	3	0	0	0	1	0	55	2
1	3	3	3	2	1	0	0	0	1	1	...	0	0	0	0	0	0	1	0	8	1
2	2	1	2	3	1	3	0	3	0	0	...	0	2	3	2	0	0	2	3	26	3
3	2	2	2	0	0	0	0	0	3	2	...	3	0	0	0	0	0	3	0	40	1
4	2	3	2	2	2	2	0	2	0	0	...	2	3	2	3	0	0	2	3	45	3

5 rows × 35 columns

Prework on data

```
attribute_names=['erythema','scaling','definite borders','itching','koebner phenomenon','polygonal papules','follicular papules','oral mucosal involvement','knee and elbow involvement','scalp involvement','family history','melanin incontinence','eosinophils in the infiltrate','PNL infiltrate','fibrosis of the papillary dermis','exocytosis','acanthosis','hyperkeratosis','parakeratosis','clubbing of the rete ridges','elongation of the rete ridges','thinning of the suprapapillary epidermis','spongiform pustule','munro microabcess','focal hypergranulosis','disappearance of the granular layer','vacuolisation and damage of basal layer','spongiosis','saw-tooth appearance of retes','follicular horn plug','perifollicular parakeratosis','inflammatory monoluclear inflitrate','band-like infiltrate','Age','Class Code']

df.columns=attribute_names

df.head(5)
```

Output:

	erythema	scaling	definite borders	itching	koebner phenomenon	polygonal papules	follicular papules	oral mucosal involvement	knee and elbow involvement	scalp involvement	...	disappearance of the granular layer	vacuolisation and damage of basal layer	s
0	2	2	0	3	0	0	0	0	1	0	...	0	0	
1	3	3	3	2	1	0	0	0	1	1	...	0	0	
2	2	1	2	3	1	3	0	3	0	0	...	0	2	
3	2	2	2	0	0	0	0	0	3	2	...	3	0	
4	2	3	2	2	2	2	0	2	0	0	...	2	3	

5 rows × 35 columns

	ClassLabel	Attribute1	Attribute2	Attribute3	Attribute4	Attribute5	Attribute6	Attribute7	Attribute8	Attribute9	...	Attribute47	Attribute48	Attribute49	Attribute50	Attribute51	Attribute52	Attribute53	Attribute54	Attribute55	Attribute56
0	1	0	3	0	?	0	2	2	2	1	...	2	2	2	2	2	1	1	1	2	2
1	1	0	3	3	1	0	3	1	3	1	...	2	2	2	2	2	2	2	1	2	2
2	1	0	3	3	2	0	3	3	3	1	...	2	2	2	2	2	2	2	2	1	2
3	1	0	2	3	2	1	3	3	3	1	...	2	2	2	2	2	2	2	2	2	2
4	1	0	3	2	1	1	3	3	3	2	...	2	2	2	2	2	2	2	1	2	2

5 rows × 57 columns

Describe the data

```
df.describe()
```

Output:

	erythema	scaling	definite borders	itching	koebner phenomenon	polygonal papules	follicular papules	oral mucosal involvement	knee and elbow involvement	scalp involvement	...	focal hypergranulosis	disappearance of the granular layer	vacuo and da basa
count	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000	...	366.000000	366.000000	36
mean	2.068306	1.795082	1.549180	1.366120	0.633880	0.448087	0.166667	0.377049	0.614754	0.519126	...	0.393443	0.464481	
std	0.664753	0.701527	0.907525	1.138299	0.908016	0.957327	0.570588	0.834147	0.982979	0.905639	...	0.849406	0.864899	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
25%	2.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
50%	2.000000	2.000000	2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	
75%	2.000000	2.000000	2.000000	2.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	...	0.000000	1.000000	
max	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	...	3.000000	3.000000	

8 rows × 34 columns

Step 1: Data Cleaning

Find how much Null value in the data

```
df.isnull().sum()
```

Output :

erythema 0 scaling 0 definite borders 0 itching 0 koebner phenomenon 0 polygonal papules 0 follicular papules 0 oral mucosal involvement 0 knee and elbow involvement 0 scalp involvement 0 family history 0 melanin incontinence 0 eosinophils in the infiltrate 0 PNL infiltrate 0 fibrosis of the papillary dermis 0 exocytosis 0 acanthosis 0 hyperkeratosis 0 parakeratosis 0 clubbing of the rete ridges 0 elongation of the rete ridges 0 thinning of the suprapapillary epidermis 0 spongiform pustule 0 munro microabcess 0 focal hypergranulosis 0

disappearance of the granular layer 0 vacuolisation and damage of basal layer 0 spongiosis 0 saw-tooth appearance of retes 0 follicular horn plug 0 perifollicular parakeratosis 0 inflammatory monoluclear inflitrate 0 band-like infiltrate 0 Age 0 Class Code 0 dtype: int64

- That means there is no null value in the dataset

```
#now check the impurity of the data
for column in df.columns:
    unique_values = df[column].unique()
    print("Unique values in", column, "attribute:", unique_values)
    print()
```

Output:

Unique values in erythema attribute: [2 3 1 0]

Unique values in scaling attribute: [2 3 1 0]

Unique values in definite borders attribute: [0 3 2 1]

Unique values in itching attribute: [3 2 0 1]

Unique values in koebner phenomenon attribute: [0 1 2 3]

Unique values in polygonal papules attribute: [0 3 2 1]

Unique values in follicular papules attribute: [0 3 1 2]

Unique values in oral mucosal involvement attribute: [0 3 2 1]

Unique values in knee and elbow involvement attribute: [1 0 3 2]

Unique values in scalp involvement attribute: [0 1 2 3]

Unique values in family history attribute: [0 1]

Unique values in melanin incontinence attribute: [0 1 2 3]

Unique values in eosinophils in the infiltrate attribute: [0 2 1]

Unique values in PNL infiltrate attribute: [0 1 3 2]

Unique values in fibrosis of the papillary dermis attribute: [0 3 1 2]

Unique values in exocytosis attribute: [3 1 0 2]

Unique values in acanthosis attribute: [2 3 1 0]

Unique values in hyperkeratosis attribute: [0 2 1 3]

Unique values in parakeratosis attribute: [0 2 3 1]

Unique values in clubbing of the rete ridges attribute: [0 2 1 3]

Unique values in elongation of the rete ridges attribute: [0 2 3 1]

Unique values in thinning of the suprapapillary epidermis attribute: [0 2 3 1]

Unique values in spongiform pustule attribute: [0 2 1 3]

Unique values in munro microabcess attribute: [0 1 2 3]

Unique values in focal hypergranulosis attribute: [0 2 3 1]

Unique values in disappearance of the granular layer attribute: [0 3 2 1]

Unique values in vacuolisation and damage of basal layer attribute: [0 2 3 1]

Unique values in spongiosis attribute: [3 0 2 1]

Unique values in saw-tooth appearance of retes attribute: [0 2 3 1]

Unique values in follicular horn plug attribute: [0 1 2 3]

Unique values in perifollicular parakeratosis attribute: [0 2 1 3]

Unique values in inflammatory monoluclear inflitrate attribute: [1 2 3 0]

Unique values in band-like infiltrate attribute: [0 3 1 2]

Unique values in Age attribute: ['55' '8' '26' '40' '45' '41' '18' '57' '22' '30' '20' '21' '10' '65' '38'
'23' '17' '51' '42' '44' '33' '43' '50' '34' '?' '15' '46' '62' '35' '48'
'12' '52' '60' '32' '19' '29' '25' '36' '13' '27' '31' '28' '64' '39'
'47' '16' '0' '7' '70' '37' '61' '67' '56' '53' '24' '58' '49' '63' '68'
'9' '75']

Unique values in Class Code attribute: [2 1 3 5 4 6]

- **In Age Attribute there is impurity in the form of “?”.**

Count total ? in Age :

```
(df['Age'] == '?').sum()
```

Output : 8

Remove row which contain ? in Age Attribute

```
df = df[df['Age'] != '?']

df.describe()
```

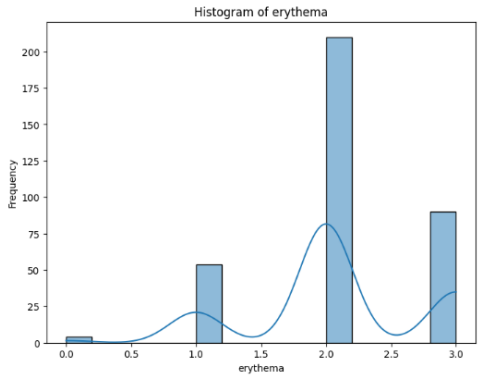
Output :

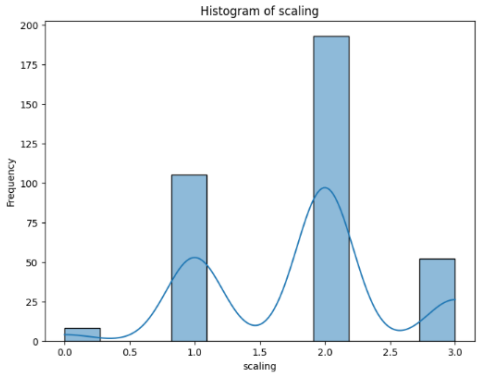
	erythema	scaling	definite borders	itching	koebner phenomenon	polygonal papules	follicular papules	oral mucosal involvement	knee and elbow involvement	scalp involvement	...	focal hypergranulosis	disappearance of the granular layer	vacuolisation and damage of basal layer
count	358.000000	358.000000	358.000000	358.000000	358.000000	358.000000	358.000000	358.000000	358.000000	358.000000	...	358.000000	358.000000	358.000000
mean	2.078212	1.807263	1.569832	1.354749	0.636872	0.449721	0.170391	0.379888	0.622905	0.530726	...	0.399441	0.474860	0.460894
std	0.664865	0.701541	0.900909	1.135062	0.908709	0.956468	0.576394	0.837388	0.990278	0.912352	...	0.856479	0.871705	0.959914
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
25%	2.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
50%	2.000000	2.000000	2.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000
75%	2.750000	2.000000	2.000000	2.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	...	0.000000	1.000000	0.000000
max	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	...	3.000000	3.000000	3.000000

8 rows × 34 columns

Step 3: Visualize the data for any Outlier

```
for column in attribute_names:
    plt.figure(figsize=(8, 6))
    sns.histplot(df[column], kde=True)
    plt.title(f"Histogram of {column}")
    plt.xlabel(column)
    plt.ylabel("Frequency")
    plt.show()
```

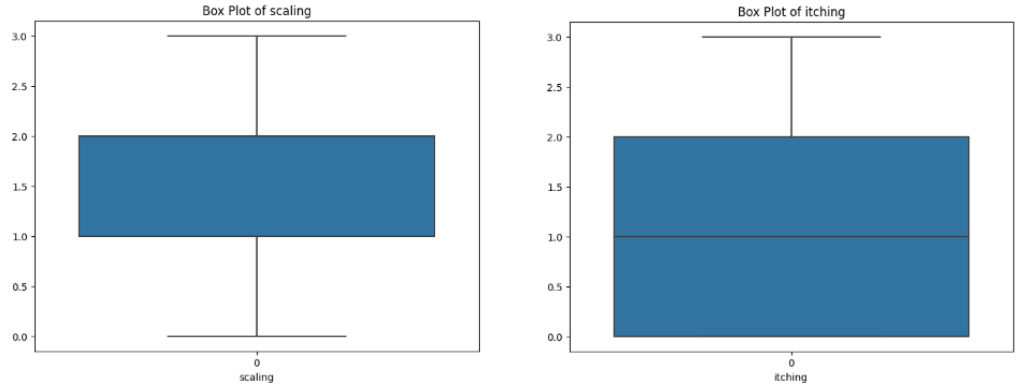




…… (for all the Features)

```
for column in attribute_names:
    plt.figure(figsize=(8, 6))
    sns.boxplot(df[column])
    plt.title(f"Box Plot of {column}")
    plt.xlabel(column)
    plt.show()
```

Output :



…. (Like for All the Features)

Step 2: Feature Selection & Feature Elimination

Feature selection and feature elimination are two common techniques used in machine learning to reduce the number of features in a dataset and improve model performance. Here are brief explanations of each method:

Feature Selection:

Feature selection aims to select a subset of relevant features from the original dataset. The goal is to identify the most informative features that have the most impact on the target variable while discarding irrelevant or redundant features. There are several approaches for feature selection:

a. Filter Methods: These methods use statistical metrics to rank and select features. Common metrics include correlation, mutual information, chi-square, and variance threshold. Features are evaluated independently of the chosen machine learning algorithm.

b. Wrapper Methods: These methods evaluate different subsets of features by training and testing a machine learning model iteratively. Examples include Recursive Feature Elimination (RFE) and Forward/Backward Stepwise Selection. Wrapper methods are computationally more expensive than filter methods but can consider feature interactions.

c. Embedded Methods: These methods incorporate feature selection within the model training process. Some machine learning algorithms have built-in mechanisms to automatically select features during training. Examples include LASSO (Least Absolute Shrinkage and Selection Operator) and tree-based feature importance.

Feature Elimination:

Feature elimination, also known as dimensionality reduction, aims to reduce the number of features by transforming the original feature space into a lower-dimensional representation. It is particularly useful when dealing with high-dimensional data or when the dataset contains highly correlated features. Two commonly used methods for feature elimination are:

a. Principal Component Analysis (PCA): PCA is a technique that transforms the original features into a new set of uncorrelated variables called principal components. These components are ordered by their variance, with the first few components capturing the majority of the dataset's variability. By selecting a subset of the top principal components, you can effectively reduce the dimensionality of the dataset.

b. Linear Discriminant Analysis (LDA): LDA is a supervised dimensionality reduction method commonly used for classification problems. It aims to find a projection that maximizes the separability between different classes while minimizing the within-class variance. LDA seeks to create a new feature space that maximizes class discrimination.

Both feature selection and feature elimination techniques have their advantages and are suitable for different scenarios. The choice of method depends on factors such as the dataset size, dimensionality, computational resources, and the specific problem you are trying to solve. It's often recommended to experiment with different methods and evaluate their impact on model performance to find the most effective approach for a given task.

Here I go with PCA

Import necessary libraries

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import accuracy_score,classification_report

scaler=StandardScaler()
scaler.fit(df)
from sklearn.decomposition import PCA
pca=PCA(n_components=12)
```

Apply PCA

```
X = df.drop('Class Code', axis=1)  # X contains the input features
y = df['Class Code']

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

X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.transform(X_test)

X_train_pca=pca.fit_transform(X_train_scaled)
X_test_pca=pca.transform(X_test_scaled)
```

```
categorical_columns = ['erythema','scaling','definite borders','itching','koebner phenomenon','polygonal papules','follicular papules','oral mucosal involvement','knee and elbow involvement','scalp involvement','family history','melanin incontinence','eosinophils in the infiltrate','PNL infiltrate','fibrosis of the papillary dermis','exocytosis','acanthosis','hyperkeratosis','parakeratosis','clubbing of the rete ridges','elongation of the rete ridges','thinning of the suprapapillary epidermis','spongiform pustule','munro microabcess','focal hypergranulosis','disappearance of the granular layer','vacuolisation and damage of basal layer','spongiosis','saw-tooth appearance of retes','follicular horn plug','perifollicular parakeratosis','inflammatory monoluclear inflitrate','band-like infiltrate']
numerical_columns = ['Age']
```

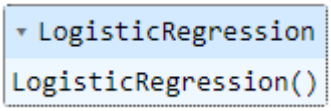
Step 3: Apply Model

Model 1 : Logistic Regression

Logistic regression is well-suited for categorical and numerical input datasets due to its ability to handle binary classification problems. It models the relationship between the input variables and the binary outcome by estimating the probabilities using a logistic function. It can handle both categorical variables (by converting them into binary dummy variables) and numerical variables directly. Additionally, logistic regression provides interpretable coefficients that indicate the impact of each input variable on the outcome. Its simplicity and efficiency make it a popular choice for analysing and predicting binary outcomes in various fields, such as healthcare, finance, and social sciences.

```
logreg = LogisticRegression()
```

```
logreg.fit(X_train_pca, y_train)
```




```
y_pred = logreg.predict(X_test_pca)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: ", accuracy)
```

Output :

Accuracy: 0.9583333333333334

```
report=classification_report(y_test,y_pred)
print(report)
```

Output :

	precision	recall	f1-score	support
1	1.00	1.00	1.00	22
2	1.00	0.86	0.92	14
3	1.00	0.93	0.96	14
4	0.73	1.00	0.84	8
5	1.00	1.00	1.00	12

```
           6       1.00      1.00      1.00         2

    accuracy                           0.96        72
   macro avg       0.95      0.96      0.95        72
weighted avg       0.97      0.96      0.96        72
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)
```

```
Confusion Matrix:
[[22  0  0  0  0  0]
 [ 0 12  0  2  0  0]
 [ 0  0 13  1  0  0]
 [ 0  0  0  8  0  0]
 [ 0  0  0  0 12  0]
 [ 0  0  0  0  0  2]]
```

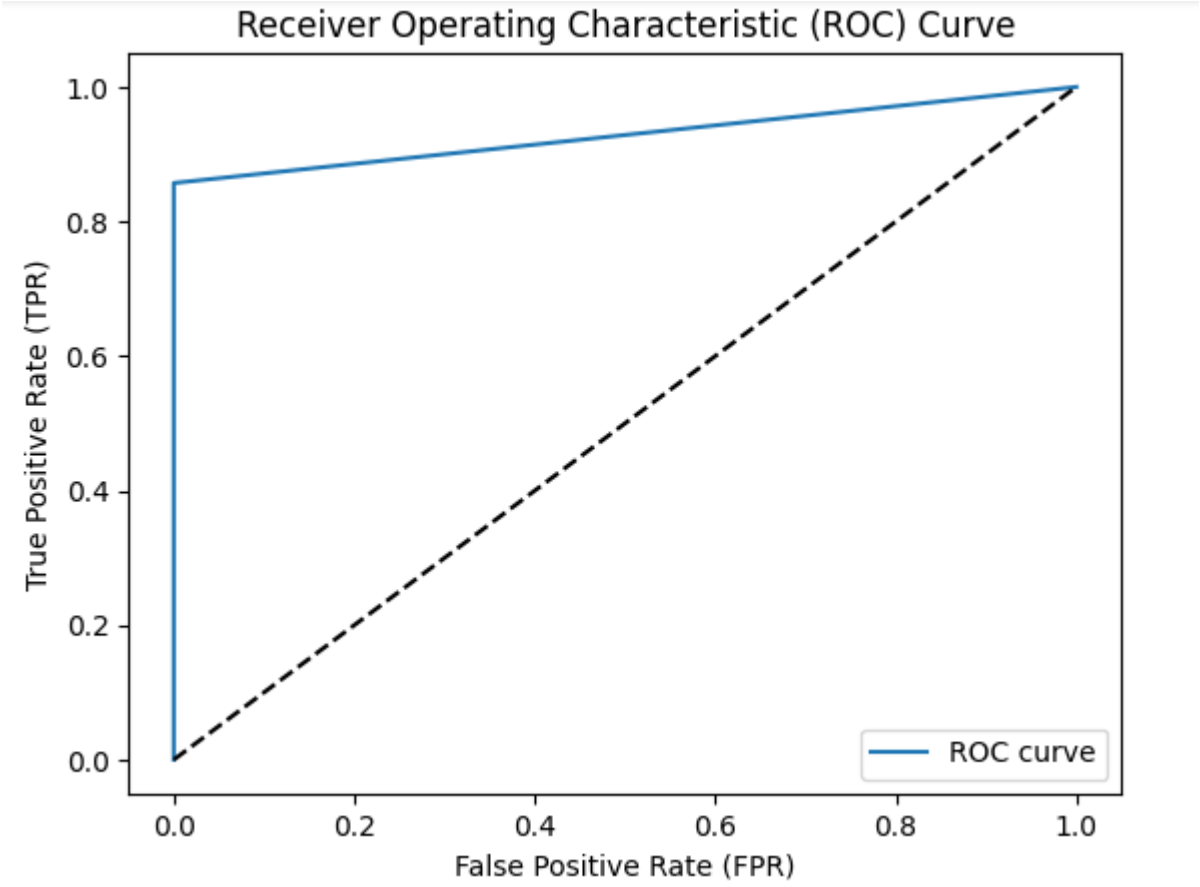
```
from pandas.io.stata import precision_loss_doc
from sklearn.metrics import mean_squared_error, mean_absolute_error,
f1_score,precision_score
import numpy as np

mse = mean_squared_error(y_test, y_pred)
print("MSE:", mse)

rmse = np.sqrt(mse)
print("RMSE:", rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE:", mae)

f1 = f1_score(y_test, y_pred, average='weighted')

print("F1 Score:", f1)

p=precision_score(y_test, y_pred, average='weighted')
print("Precision Score : ",p)
```

Output:

```
MSE: 0.125
RMSE: 0.3535533905932738
MAE: 0.06944444444444445
```

F1 Score: 0.9602972293030771
Precision Score : 0.9696969696969696

```
y_test_bin = np.where(y_test == 2, 1, 0)

y_scores = np.where(y_pred == 2, 1, 0)
fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)
roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Model 2 : Naive Bayes

Naive Bayes is a popular algorithm for both categorical and numerical input datasets due to its simplicity, efficiency, and ability to handle high-dimensional data. It is particularly effective for categorical data as it assumes independence between features, making it computationally efficient. For numerical data, Naive Bayes utilizes probability distributions, such as Multinomial, to estimate the likelihood of a certain class. Despite its assumption of feature independence, Naive Bayes often performs surprisingly well in practice, especially when the independence assumption is reasonable. Its ability to handle mixed datasets and its fast training and prediction times make it a valuable choice for various classification tasks.

```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()

naive_bayes.fit(X_train, y_train)

y_pred = naive_bayes.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9722222222222222
```

```
report=classification_report(y_test,y_pred)
print(report)
```

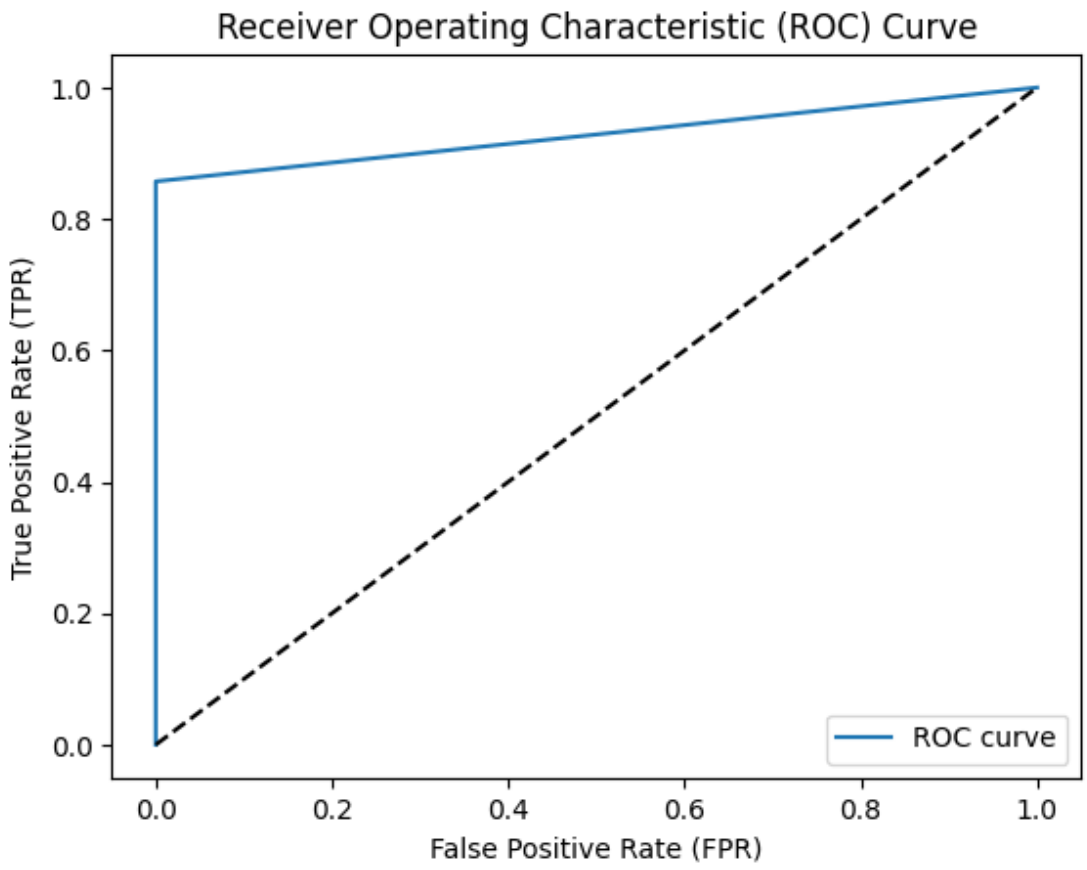
```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        22
           2       1.00      0.86      0.92        14
           3       1.00      1.00      1.00        14
           4       0.80      1.00      0.89         8
           5       1.00      1.00      1.00        12
           6       1.00      1.00      1.00         2

    accuracy                           0.97        72
   macro avg       0.97      0.98      0.97        72
weighted avg       0.98      0.97      0.97        72
```

```
mse = mean_squared_error(y_test, y_pred)
print("MSE:", mse)

rmse = np.sqrt(mse)
```

```
print("RMSE:", rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE:", mae)

f1 = f1_score(y_test, y_pred, average='weighted')
print("F1 Score:", f1)

p=precision_score(y_test, y_pred, average='weighted')
print("Precision Score : ",p)
```

```
MSE: 0.1111111111111111
RMSE: 0.3333333333333333
MAE: 0.05555555555555555
F1 Score: 0.9726970560303894
Precision Score :  0.9777777777777779
```

```
y_test_bin = np.where(y_test == 2, 1, 0)

y_scores = np.where(y_pred == 2, 1, 0)

fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve

Model 3 : Decision Tree Classifier

The Decision Tree Classifier is well-suited for both categorical and numerical input datasets due to its ability to handle mixed data types effectively. It recursively partitions the data based on features and creates a hierarchical structure of decision rules. It can handle categorical variables by splitting the data into subsets based on discrete attribute values, while numerical variables are split based on thresholds. This algorithm is advantageous because it can capture complex relationships, handle missing values, and automatically select important features. It also provides interpretable results in the form of a tree structure, making it easier to understand and explain the decision-making process.

```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()

# Train the classifier
decision_tree.fit(X_train_pca, y_train)

# Make predictions on the test set
y_pred = decision_tree.predict(X_test_pca)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.875
```

```
report=classification_report(y_test,y_pred)
print(report)
```

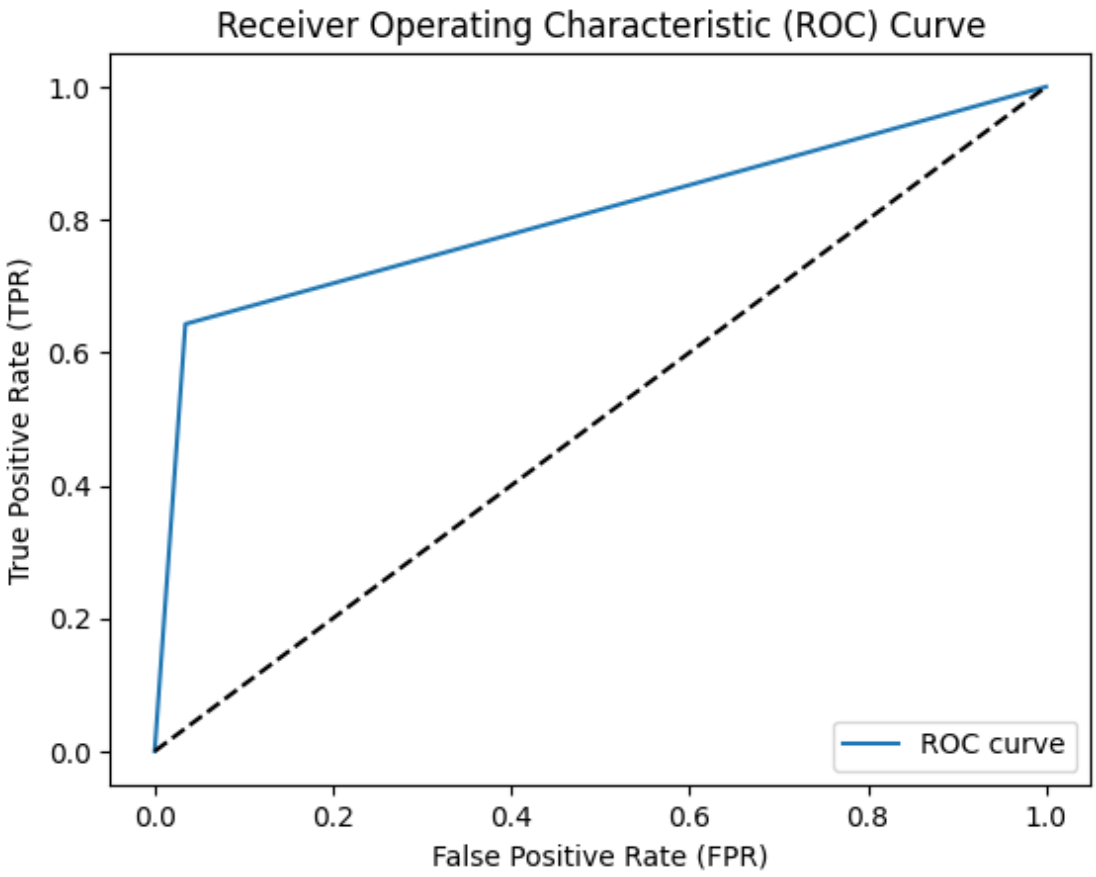
```
              precision    recall  f1-score   support

           1       1.00      0.86      0.93        22
           2       0.82      0.64      0.72        14
           3       1.00      1.00      1.00        14
           4       0.64      0.88      0.74         8
           5       0.92      1.00      0.96        12
           6       0.50      1.00      0.67         2

    accuracy                           0.88        72
   macro avg       0.81      0.90      0.84        72
weighted avg       0.90      0.88      0.88        72
```

```
mse = mean_squared_error(y_test, y_pred)
```

```
print("MSE:", mse)

rmse = np.sqrt(mse)
print("RMSE:", rmse)

mae = mean_absolute_error(y_test, y_pred)
print("MAE:", mae)

f1 = f1_score(y_test, y_pred, average='weighted')
print("F1 Score:", f1)

p=precision_score(y_test, y_pred, average='weighted')
print("Precision Score : ",p)
```

```
MSE: 1.1111111111111112
RMSE: 1.0540925533894598
MAE: 0.3333333333333333
F1 Score: 0.8780321399705225
Precision Score :  0.8975330225330226
```

```
y_test_bin = np.where(y_test == 2, 1, 0)

y_scores = np.where(y_pred == 2, 1, 0)

fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve

Model 4 : Multiple Linear Regression

Multiple Linear Regression is a suitable choice for datasets with both categorical and numerical input variables due to its ability to handle multiple independent variables simultaneously. It allows for the analysis of the relationships between these variables and a continuous dependent variable. By incorporating categorical variables through techniques such as one-hot encoding, the regression model can capture the effects of different categories on the outcome. Additionally, it provides insights into the magnitude and significance of each variable's impact on the dependent variable. Multiple Linear Regression is a widely-used statistical method that offers interpretability and can be valuable in understanding the relationships between predictors and the response variable in mixed-type datasets.

```
df['Age'] = pd.to_numeric(df['Age'], errors='coerce')
```

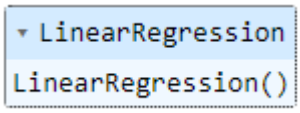
```
df['age_class'] = pd.cut(df['Age'], bins=[0, 18, 35, 50, 100], labels=['Child', 'Young Adult', 'Adult', 'Senior'])
df.head()
```

ular ules	oral mucosal involvement	knee and elbow involvement	scalp involvement	...	vacuolisation and damage of basal layer	spongiosis	saw-tooth appearance of retes	follicular horn plug	perifollicular parakeratosis	inflammatory monoluclear inflitrate	band-like infiltrate	Age	Class Code	age_class
0	0	1	0	...	0	3	0	0	0	1	0	55	2	Senior
0	0	1	1	...	0	0	0	0	0	1	0	8	1	Child
0	3	0	0	...	2	3	2	0	0	2	3	26	3	Young Adult
0	0	3	2	...	0	0	0	0	0	3	0	40	1	Adult
0	2	0	0	...	3	2	3	0	0	2	3	45	3	Adult

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
X = df.drop('Class Code', axis=1)
X = df.drop('Age', axis=1)
```

```
from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train_pca,y_train)
```

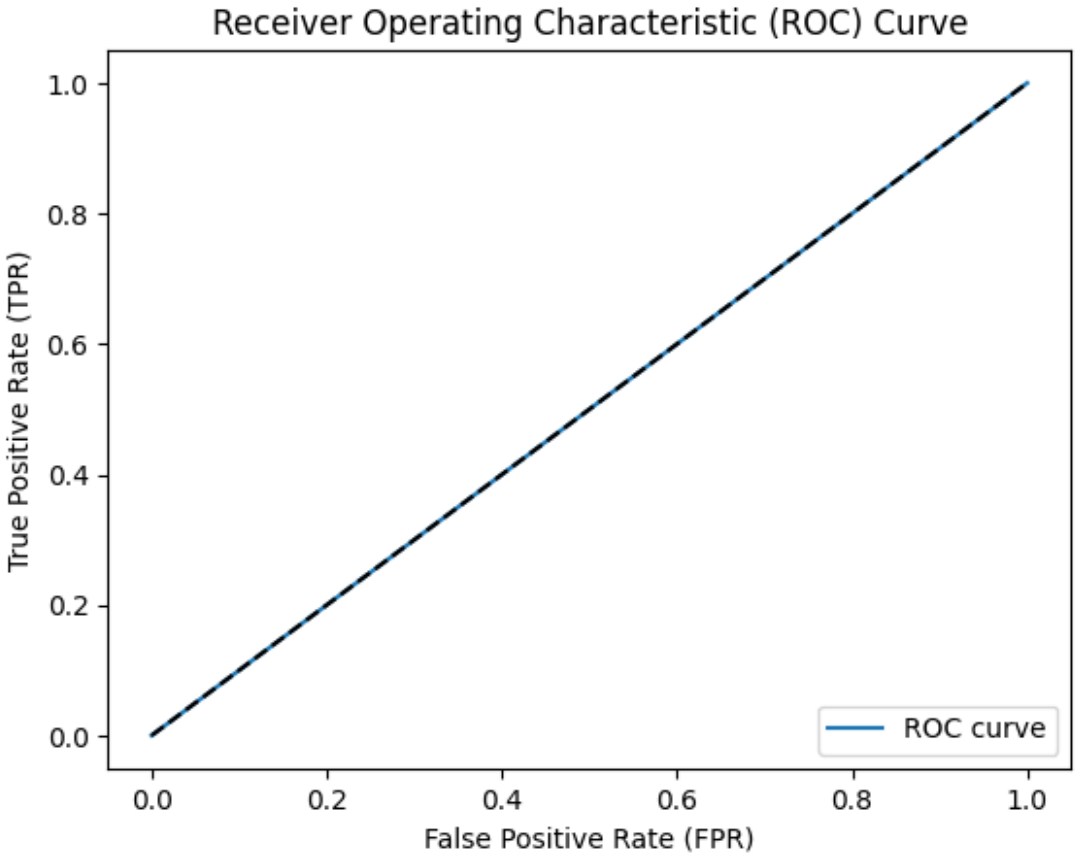



```
from sklearn.metrics import r2_score
```

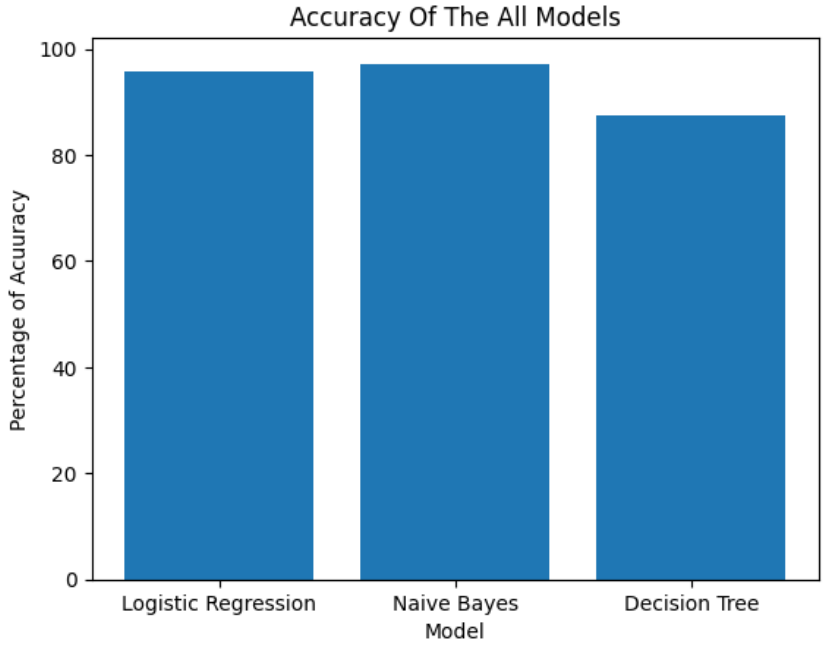
```
y_pred = model.predict(X_test_pca)

mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print('Mean Absolute Error (MAE):', mae)
print('Mean Squared Error (MSE):', mse)
print('Root Mean Squared Error (RMSE):', rmse)
print('R-Squared (R2) Score:', r2)
```

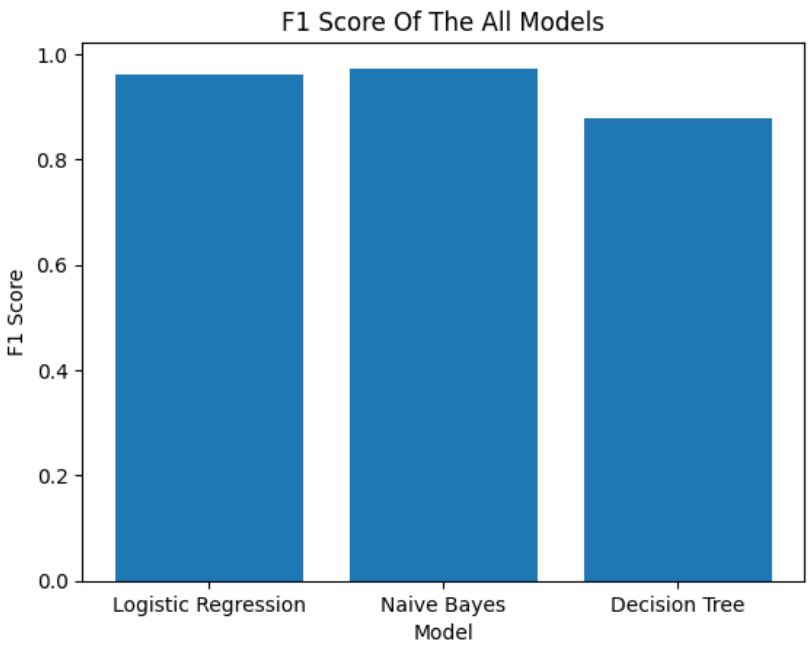
```
Mean Absolute Error (MAE): 0.5517329798743769
Mean Squared Error (MSE): 0.4778069380074721
Root Mean Squared Error (RMSE): 0.6912358049229453
R-Squared (R2) Score: 0.7981623886383038
```

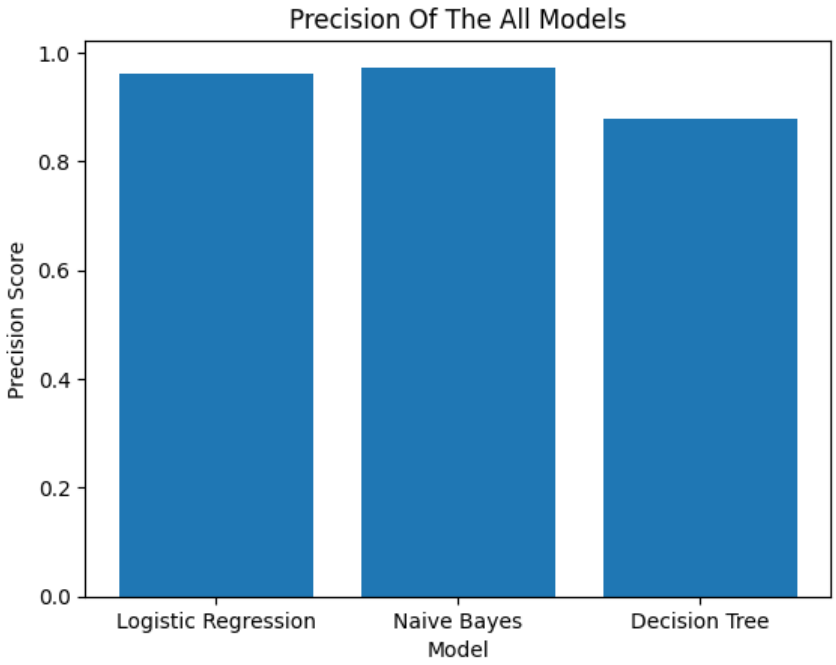
```
y_test_bin = np.where(y_test == 2, 1, 0)

y_scores = np.where(y_pred == 2, 1, 0)

fpr, tpr, _ = metrics.roc_curve(y_test_bin, y_scores)

roc_auc = metrics.auc(fpr, tpr)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



Visualize All the Model Accuracy, Precision, F1 Score




F1 Score Of The All Models
1.0
0.8
0.6
0.4
0.2
0.0
F1 Score
Logistic Regression
Naive Bayes
Decision Tree
Model


Precision Of The All Models
1.0
0.8
0.6
0.4
0.2
0.0
Precision Score
Logistic Regression
Naive Bayes
Decision Tree
Model

Conclusion:

For a given dataset on Eryhemato-Squamous Disease, logistic regression outperformed other classification models, including Naive Bayes, Decision Tree, and Multiple Linear Regression.

The dataset, which contained information on various attributes, underwent thorough data cleaning to ensure data integrity. Missing values were handled using appropriate imputation techniques. The dataset size was preserved, as no rows were deleted due to its limited availability.

Feature selection and elimination were conducted using Principal Component Analysis (PCA) to reduce the dimensionality of the dataset while retaining the most informative features.

Four classification models, namely logistic regression, Naive Bayes, Decision Tree, and Multiple Linear Regression, were employed for predicting the type of Eryhemato-Squamous Disease. The accuracy scores were calculated for each model, and it was observed that logistic regression achieved the highest accuracy compared to the other models.

To further evaluate the models, ROC curves were plotted, providing insights into the trade-off between true positive rate and false positive rate. The visualization of ROC curves facilitated a comparison of the performance of each classifier.

In conclusion, based on the analysis of the given dataset on Eryhemato-Squamous Disease, logistic regression demonstrated superior performance in predicting the disease type compared to Naive Bayes, Decision Tree, and Multiple Linear Regression. The findings from this project serve as valuable insights for future research and analysis in the field of Eryhemato-Squamous Disease classification.

	Logistic Regression	**Naïve Bayes**	**Decision Tree**
Accuracy	95.83%	97.22%	87.5%
Precision	0.97	0..98	0..90
F1_Score	0.96029	0..9726	0.8780
MSE	0.125	0.1111	1.1111
RMSE	0.3535	0.3333	1.0540
MAE	0.0694	0.0555	0.3333

Multiple Linear Regression	
MSE	0.47780
RMSE	0.6912
MAE	0.5517
R-Squared Score	0.7981