PRCP-1027-Skin Disorder

Project team's ID : PTID-CDS-JUL-24-1984

PRCP-1027-Skin Disorder

Problem Statement

Task 1:-Prepare a complete data analysis report on the given data.

Task 2:-Create a predictive model using machine learning techniques to predict the various classes of skin disease.

Task3:-Suggestions to the Doctors to identify the skin diseases of the patient at the earliest.

Dataset Information:

This database contains 34 attributes, 33 of which are linear valued and one of them is nominal.The differential diagnosis of erythemato-squamous diseases is a real problem in dermatology. They all share the clinical features of erythema and scaling, with very little differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, cronic dermatitis, and pityriasis rubra pilaris. Usually a biopsy is necessary for the diagnosis but unfortunately these diseases share many histopathological features as well. Another difficulty for the differential diagnosis is that a disease may show the features of another disease at the beginning stage and may have the characteristic features at the following stages. Patients were first evaluated clinically with 12 features. Afterwards, skin samples were taken for the evaluation of 22 histopathological features. The values of the histopathological features are determined by an analysis of the samples under a microscope.In the dataset constructed for this domain, the family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise. The age feature simply represents the age of the patient. Every other feature (clinical and histopathological) was given a degree in the range of 0 to 3. Here, 0 indicates that the feature was not present, 3 indicates the largest amount possible, and 1, 2 indicate the relative intermediate values.The names and id numbers of the patients were recently removed from the database.

Domain: Healthcare

Attribute Information:

Clinical Attributes: (take values 0, 1, 2, 3, unless otherwise indicated)

1: erythema 2: scaling 3: definite borders 4: itching 5: koebner phenomenon 6: polygonal papules 7: follicular papules 8: oral mucosal involvement 9: knee and elbow involvement 10: scalp involvement 11: family history, (0 or 1) Histopathological Attributes: (take values 0, 1, 2, 3) 12: melanin incontinence 13: eosinophils in the infiltrate 14: PNL infiltrate 15: fibrosis of the papillary dermis 16: exocytosis 17: acanthosis 18: hyperkeratosis 19: parakeratosis 20: clubbing of the rete ridges 21: elongation of the rete ridges 22: thinning of the suprapapillary epidermis 23: spongiform pustule 24: munro microabcess 25: focal hypergranulosis 26: disappearance of the granular layer 27: vacuolisation and damage of basal layer 28: spongiosis 29: saw-tooth appearance of retes 30: follicular horn plug 31: perifollicular parakeratosis 32: inflammatory monoluclear inflitrate 33: band-like infiltrate 34: Age (linear)

Model Comparison Report

Create a report stating the performance of multiple models on this data and suggest the best model for production.

Report on Challenges faced

Create a report which should include challenges you faced on data and what technique used with proper reason.

Note:-All above task has been created on single jupyter notebook and share the same while

1. Attribute Information :

1.1 Clinical Attributes :

1. Erythema :

- The presence or absence of redness or inflammation of the skin.

2. Scaling :

- Whether there are visible scales or flakes on the skin's surface.

3. Definite Borders :

- Describes whether the skin lesion has well-defined or clear borders.

4. Itching :

- Indicates whether the patient experiences itching or pruritus associated with the skin condition.

5. Koebner Phenomenon :

- Refers to whether new skin lesions appear at sites of trauma or injury to the skin.

6. Polygonal Papules :

- Presence or absence of polygonal-shaped raised skin lesions.

7. Follicular Papules :

- Whether there are raised lesions involving hair follicles.

8. Oral Mucosal Involvement :

- Indicates if the condition affects the mucous membranes inside the mouth.

9. Knee and Elbow Involvement :

- Describes whether the skin condition is specifically located on the knees and elbows.

10. Scalp Involvement :

- Whether the condition affects the scalp.

11. Family History (0 or 1) :

- Indicates whether there is a family history of similar skin conditions or relevant genetic factors.

12. Age :

13. Summary of all the above Attributes :

- These clinical attributes are important for dermatologists to assess and diagnose various skin conditions accurately. They are used in clinical examinations and research to characterize the presentation and progression of dermatological diseases. Researchers and healthcare providers can analyze these attributes to understand patterns and correlations in different skin conditions and to develop treatment plans tailored to individual patients.

1.2 Histopathological Attributes:

1. Melanin Incontinence :

- Melanin incontinence refers to the presence of melanin pigment within the dermis. It can occur when melanocytes are damaged or when melanin leaks from epidermal cells into the deeper skin layers. Melanin incontinence is often associated with conditions like melanoma and lichen planus.

2. Eosinophils in the Infiltrate :

- Eosinophils are a type of white blood cell involved in allergic and inflammatory responses. The presence of eosinophils in the inflammatory infiltrate can be indicative of certain allergic or eosinophilic skin conditions.

3. PNL Infiltrate :

- PNL (polymorphonuclear leukocyte) infiltrate refers to the presence of polymorphonuclear white blood cells in the skin tissue. PNLs are involved in the early stages of inflammation and can be seen in various inflammatory skin disorders.

4. Fibrosis of the Papillary Dermis :

- Fibrosis in the papillary dermis indicates the accumulation of excessive fibrous tissue in the uppermost layer of the dermis. It can result from chronic inflammation and is often associated with scarring.

5. Exocytosis :

- Exocytosis refers to the migration of inflammatory cells (usually lymphocytes) from the bloodstream into the epidermis. This phenomenon is commonly observed in conditions like psoriasis.

6. Acanthosis :

- Acanthosis is characterized by the thickening of the epidermis, particularly the stratum spinosum layer. It is a common histological finding in conditions like acanthosis nigricans and psoriasis.

7. Hyperkeratosis :

- Hyperkeratosis is the excessive thickening of the stratum corneum, the outermost layer of the epidermis. It leads to the formation of a thickened, keratinized surface and can be seen in various skin disorders.

8. Parakeratosis :

- Parakeratosis is a histological feature where nuclei are retained in the stratum corneum. It is often seen in psoriasis and other hyperproliferative skin conditions.

9. Clubbing of the Rete Ridges :

- Clubbing of the rete ridges refers to the bulbous enlargement of the rete ridges, which are the finger-like projections of the epidermis into the dermis. This can be seen in conditions like lichen planus.

10. Elongation of the Rete Ridges :

- Elongation of the rete ridges indicates the lengthening of these epidermal projections into the dermis. It is a common finding in many skin diseases.

11. Thinning of the Suprapapillary Epidermis :

- This feature involves a reduction in the thickness of the suprapapillary epidermis, which is the epidermal layer above the dermal papillae. It can be observed in some inflammatory skin conditions.

12. Spongiform Pustule :

- A spongiform pustule is a blister-like structure filled with neutrophils and located within the epidermis. It is characteristic of pustular psoriasis.

13. Munro Microabscess :

- Munro microabscesses are small collections of neutrophils in the stratum corneum. They are commonly found in the epidermis of individuals with psoriasis.

14. Focal Hypergranulosis :

- Focal hypergranulosis refers to localized thickening of the granular layer of the epidermis. This can be seen in various skin disorders.

15. Disappearance of the Granular Layer :

- In some skin conditions, the granular layer of the epidermis may be absent or significantly reduced.

16. Vacuolization and Damage of Basal Layer :

- Vacuolization refers to the formation of empty spaces (vacuoles) within the basal layer of the epidermis. It is often associated with autoimmune blistering disorders.

17. Spongiosis :

- Spongiosis is the presence of intercellular edema (fluid accumulation between epidermal cells) in the epidermis. It is a common feature of eczematous conditions.

18. Saw-tooth Appearance of Rete Ridges :

- The saw-tooth appearance is characterized by irregular, jagged projections of the epidermal rete ridges and is often seen in lichen planus.

19. Follicular Horn Plug :

- A follicular horn plug is a collection of keratinous material within a hair follicle. It can be seen in various conditions, including acne.

20. Perifollicular Parakeratosis :

- Perifollicular parakeratosis is the presence of parakeratosis around hair follicles. It can be observed in certain inflammatory skin conditions.

21. Inflammatory Mononuclear Infiltrate :

- This refers to the presence of mononuclear white blood cells (such as lymphocytes) in the dermal or epidermal infiltrate, indicating chronic inflammation.

22. Band-like Infiltrate :

- A band-like infiltrate is characterized by a dense, linear accumulation of inflammatory cells within the skin tissue. It can be seen in conditions like lichen planus.

1.3 Class of Diseases :

The diseases in this group are :

- **1. Psoriasis** :- Psoriasis is a skin disease that causes a rash with itchy, scaly patches, most commonly on the knees, elbows, trunk and scalp.

- **2. Seboreic Dermatitis** :- Seborrheic (seb-o-REE-ik) dermatitis is a common skin condition that mainly affects your scalp. It causes scaly patches, inflamed skin and stubborn dandruff. It usually affects oily areas of the body, such as the face, sides of the nose, eyebrows, ears, eyelids and chest. ,

- **3. Lichen Planus** :- Lichen planus (LIE-kun PLAY-nus) is a condition of the skin, hair, nails, mouth and genitals. On skin, lichen planus often appears as purple, itchy, flat bumps that develop over several weeks. In the mouth and genital mucosa, lichen planus forms lacy white patches, sometimes with painful sores.,

- **4. Pityriasis Rosea** :- Pityriasis rosea is a rash that often begins as an oval spot on the face, chest, abdomen or back. This is called a herald patch and may be up to 4 inches (10 centimeters) across. Then you may get smaller spots that sweep out from the middle of the body in a shape that looks like drooping pine-tree branches. The rash can be itchy. Pityriasis (pit-ih-RIE-uh-sis) rosea can happen at any age but is most common between the ages of 10 and 35. It tends to go away on its own within 10 weeks.,

- **5. Cronic Dermatitis** :- Dermatitis is a common condition that causes swelling and irritation of the skin. It has many causes and forms and often involves itchy, dry skin or a rash. Or it might cause the skin to blister, ooze, crust or flake. Three common types of this condition are atopic dermatitis, contact dermatitis and seborrheic dermatitis. Atopic dermatitis is also known as eczema.,

- **6. Pityriasis Rubra Pilaris** :- Pityriasis rubra pilaris (PRP) is a rare skin disease. It causes constant inflammation and shedding of the skin. PRP can affect parts of your body or your entire body. The disorder may start in childhood or adulthood. PRP affects males and females equally..

2. Business case

Machine learning algorithm to predict the various skin diseases. The main objective of this project is to achieve maximum accuracy of skin disease prediction. Machine learning techniques helps in detection of skin disease at an initial stage. The feature extraction plays a key role in classification of skin diseases.The usage of Machine Learning algorithms reduces the need for human labor, such as manual feature extraction and data reconstruction for classification purpose. Moreover, Explainable ML model is used to interpret the decisions made by our model.

3. Importing some basic libraries and reading the dataset

In [1]:
```python
# It is used for creating plots and charts.
import matplotlib.pyplot as plt


# It is used to display plots inline in jupyter notebook.
%matplotlib inline

from PIL import Image

# It is used for statistical data visualization.
import seaborn as sns

# It is used for numerical operations.
import numpy as np

# It is used for data manipulation and analysis.
import pandas as pd

# Ignore warnings messages to keep the output clean.
import warnings
warnings.filterwarnings("ignore")
```

4. Load Dataset

In [2]:
```python
df = pd.read_csv('dataset_35_dermatology (1).csv')
```

5. Data exploration

5.1.1 To display 5 rows from starting of the dataset.

In [3]:
```python
df.head()
```

Out[3]:

	erythema	scaling	definite_borders	itching	koebner_phenomenon	polygonal_papules	fol
0	2	2	0	3	0	0	
1	3	3	3	2	1	0	
2	2	1	2	3	1	3	
3	2	2	2	0	0	0	
4	2	3	2	2	2	2	

5 rows × 35 columns

5.1.2 To display 5 rows randomly of the dataset.

In [4]:
```
df.sample(5)
```

Out[4]:

	erythema	scaling	definite_borders	itching	koebner_phenomenon	polygonal_papules
199	3	2	1	0	0	0
205	3	2	2	0	0	0
184	2	2	1	0	0	0
226	1	2	1	1	0	0
26	1	1	0	1	0	0

5 rows × 35 columns

5.1.3 To display last 5 rows of the dataset.

In [5]:
```
df.tail(5)
```

Out[5]:

	erythema	scaling	definite_borders	itching	koebner_phenomenon	polygonal_papules
361	2	1	1	0	1	0
362	3	2	1	0	1	0
363	3	2	2	2	3	2
364	2	1	3	1	2	3
365	3	2	2	0	0	0

5 rows × 35 columns

5.2 Chceking size of of the database

In [6]:
```
print(f'\033[1;31m  \033[1m size of given dataset  {df.size} ')
```

```
 size of given dataset  12810
```

5.3 Check no. of rows and columns

In [7]:
```
print(f'\033[1;31m  \033[1m Given dataset contains {df.shape[0]} rows 
```

```
 Given dataset contains 366 rows and 35 columns.
```

5.4 Basic information of dataset

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 366 entries, 0 to 365
Data columns (total 35 columns):
 #   Column                                    Non-Null Count  Dtype 
---  ------                                    --------------  ----- 
 0   erythema                                  366 non-null    int64 
 1   scaling                                   366 non-null    int64 
 2   definite_borders                          366 non-null    int64 
 3   itching                                   366 non-null    int64 
 4   koebner_phenomenon                        366 non-null    int64 
 5   polygonal_papules                         366 non-null    int64 
 6   follicular_papules                        366 non-null    int64 
 7   oral_mucosal_involvement                  366 non-null    int64 
 8   knee_and_elbow_involvement                366 non-null    int64 
 9   scalp_involvement                         366 non-null    int64 
 10  family_history                            366 non-null    int64 
 11  melanin_incontinence                      366 non-null    int64 
 12  eosinophils_in_the_infiltrate             366 non-null    int64 
 13  PNL_infiltrate                            366 non-null    int64 
 14  fibrosis_of_the_papillary_dermis          366 non-null    int64 
 15  exocytosis                                366 non-null    int64 
 16  acanthosis                                366 non-null    int64 
 17  hyperkeratosis                            366 non-null    int64 
 18  parakeratosis                             366 non-null    int64 
 19  clubbing_of_the_rete_ridges               366 non-null    int64 
 20  elongation_of_the_rete_ridges             366 non-null    int64 
 21  thinning_of_the_suprapapillary_epidermis  366 non-null    int64 
 22  spongiform_pustule                        366 non-null    int64 
 23  munro_microabcess                         366 non-null    int64 
 24  focal_hypergranulosis                     366 non-null    int64 
 25  disappearance_of_the_granular_layer       366 non-null    int64 
 26  vacuolisation_and_damage_of_basal_layer   366 non-null    int64 
 27  spongiosis                                366 non-null    int64 
 28  saw-tooth_appearance_of_retes             366 non-null    int64 
 29  follicular_horn_plug                      366 non-null    int64 
 30  perifollicular_parakeratosis              366 non-null    int64 
 31  inflammatory_monoluclear_inflitrate       366 non-null    int64 
 32  band-like_infiltrate                      366 non-null    int64 
 33  Age                                       366 non-null    object
 34  class                                     366 non-null    int64 
dtypes: int64(34), object(1)
memory usage: 100.2+ KB
```

5.5 Now let us analyse the statistical description of our data.

In [9]:
```
df.describe()
```

Out[9]:

	erythema	scaling	definite_borders	itching	koebner_phenomenon	polygona
count	366.000000	366.000000	366.000000	366.000000	366.000000	3
mean	2.068306	1.795082	1.549180	1.366120	0.633880	
std	0.664753	0.701527	0.907525	1.138299	0.908016	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	1.000000	1.000000	0.000000	0.000000	
50%	2.000000	2.000000	2.000000	1.000000	0.000000	
75%	2.000000	2.000000	2.000000	2.000000	1.000000	
max	3.000000	3.000000	3.000000	3.000000	3.000000	

8 rows × 34 columns

6. Data Preprocessing , Data Manipulation and Cleaning.

6.1 Check for duplicate data

In [ ]:
```
print("\033[1;31m  \033[1m Total no of duplicate value :-",df.duplicate
```

6.2 To check all the null values from the dataset

In [ ]:
```
df.isnull().sum()
```

In [ ]:
```
null_data = df.isna().sum().sort_values(ascending=False)
null_data = null_data.reset_index(drop = False)
null_data = null_data.rename(columns={"index": "Columns", 0:"Value"})
null_data['proportion'] = (null_data['Value']/len(df))*100
null_data
```

6.3 missing values

In [ ]:
```

missing=pd.DataFrame((df.isnull().sum())*100/df.shape[0]).reset_index(
plt.figure(figsize=(16,5))
ax=plt.stem(missing['index'],missing[0])
plt.xticks(rotation=90,fontsize=7)
plt.title('percent of missing values ')
plt.ylabel('percent')
plt.show()
```

Observations:

Dataset does not have null values. Hence, null value handling will not be done as a part of this assignment.

6.4.0 Unique value in dataset

6.4.1 Unique Columns

In [ ]:
```
num_unique_col=df.nunique()
print(num_unique_col)
print("\033[1;31m  \033[1m Total no of columns contain unique values :'
```

Observations:

```
We can see above each and every single columns contain unique valu
e
```

6.4.2 Uniqueness of columns

In [ ]:
```
unique_values=df.apply(pd.Series.unique)
unique_values
```

6.4.3 Exploration of Numerical Columns: Unique Values and Value Counts

In [ ]:
```python
# Iterate over each numerical column in 'num_col'.
for i in data_col_int :

    # It will print unique values and column name
    print(df[i].unique(), i)

    # It will print value counts for each unique value in the column.
    print(df[i].value_counts())

    # It will print a seperator for clarity.
    print("**********************")
```

6.5.1 Different age within the age columns

In [ ]:
```python
# Iterate over each categorical column in 'cat_col'.
for i in data_col_obj :

    # It will print unique values and column name.
    print(df[i].unique(), i)

```

Observations:

- The presence of '?' in the 'Age' column suggests the existence of missing or unknown values.

- Some values appear to be represented as strings, and there's a mix of numerical and non-numerical data.

- The column may need cleaning and conversion to a numeric format for further analysis.

6.5.2 Total count of different age

In [ ]:
```python
# Iterate over each categorical column in 'cat_col'.
for i in data_col_obj :

    # It will print value counts for each unique value in the column.
    print(df[i].value_counts())


```

6.5.3 Count of Different age group

In [ ]:
```

plt.figure(figsize= (14,8))

ax = sns.countplot(x='Age', data=df )
ax.set_title(label='Count of Different age group', fontsize=20)

ax.set_xlabel(xlabel='Age')
ax.set_ylabel(ylabel='Count')

plt.show()
```

Insights from Age column for missing values:

- 1. 0 As a age present in age columns which is not possible so we will treat it with forward fill or backward fill in upcoming section

- 2. special character " ? " present in column , we will replace it either one of them mean median mode .

6.5.4.1 Identification of Missing Values in 'Age' Column:

In [ ]:
```
# It will filter rows in the 'data' DataFrame where the 'Age' column is

df.loc[df['Age']=='?']
```

6.5.4.2 Handling Missing and Zero Values in 'Age' Column:

In [ ]:
```
# Age doesn't follow normal distribution.
# So, we will replace the missing value with median.

# Here, It will replace the '?' values in the 'Age' column with NaN (No
df.loc[df['Age']=='?', 'Age'] = np.nan

# Here, It will replace the '0' values in the 'Age' column with NaN (No
df.loc[df['Age']=='0', 'Age'] = np.nan
```

Analysis:

Identification of Missing Values:

- '?' values are replaced with NaN, which is a common practice to represent missing or undefined values in data.

Handling Zero Values:

- Zero values in the 'Age' column are also replaced with NaN. This is a

6.5.4.3 Replacing NaN and Zero Age Values with Median:

In [ ]:
```
# Replacing NaN and 0(zero) age value using median.

median = df['Age'].median()

df['Age'].fillna(median, inplace = True)

df['Age'] = df['Age'].replace(0, df['Age'].median())
```

In [ ]:
```
df['Age'].unique()
```

6.5.5 Converting Categrical data into numerical data.

In [ ]:
```
# It will convert the 'Age' column to the 'int64' data type.

df['Age'] = df['Age'].astype('int64')
```

In [ ]:
```
datatypes = df.dtypes['Age']
print(f'\033[1;31m  \033[1m datatype of Age column  {datatypes} ')
```

6.5.6 Oldest patient

In [ ]:
```
print("Oldest Patient: ")
df.loc[df['Age'] == df['Age'].max()]
```

6.5.7 Youngest Patient

In [ ]:
```
print("Oldest Patient: ")
df.loc[df['Age'] == df['Age'].min()]
```

7. EDA (Exploratory Data Analysis)

- We will try to understand our data by plotting different visuals(Graphs).

- To know which particular field is good in order to invest, We do the Exploratory Data Analysis(EDA)

7.1 Creating a deep copy of dataset for exploratory data analysis

In [ ]:
```
df_copy=df.copy(deep=True)
```

In [ ]:
```
df_copy['class'].replace([1,2,3,4,5,6], ['Psoriasis', 'Seboreic_Dermat
```

In [ ]:
```
df_copy.head()
```

7.2 Univariate Analysis

- Analysing single variable/column/feature.

- In Univariate Analysis, we check the distribution of each column.

Histplot

- A 'histplot' is a graphical representation of a histogram.

- It's a way to see the overall pattern or shape of our data and understand how values are distributed across different ranges.

7.2.1 Count and Distribution of clinical_features using Histplot :

In [ ]:
```python
# List of clinical features related to a medical condition.

clinical_features = [
    'erythema',
    'scaling',
    'definite_borders',
    'itching',
    'koebner_phenomenon',
    'polygonal_papules',
    'follicular_papules',
    'oral_mucosal_involvement',
    'knee_and_elbow_involvement',
    'scalp_involvement',
    'family_history'
]
```

In [ ]:
```python
# It will create a new figure with a fixed size and background color.
plt.figure(figsize = (23,30), facecolor = 'white')

# Initializing a counter for subplot placement.
plotnumber = 1

# Looping through each clinical features in the data.
for clinical_features in df_copy :

    # Limiting the number of subplots to 11.
    if plotnumber <= 11:

        # Create a subplot in a 4*3 grid
        ax = plt.subplot(4, 3, plotnumber)

        # Plotting a histogram with kernel density function.
        sns.histplot(x = df_copy[clinical_features], kde = True)

        # We have set the x-axis label to the clinical features.
        plt.xlabel(clinical_features, fontsize = 15)
        plt.ylabel("count", fontsize = 15)

    # Incrementing the plot number for the next iteration.
    plotnumber = plotnumber + 1

# Adjusting the layout of the subplots for better spacing.
plt.tight_layout()

# It will display the plot.
plt.show()
```

Insights from clinical features

- In this each attribute contains 0-3 values except family history.

- family history contains only 0 & 1.

- family history feature has the value 1 if any of these diseases has been observed in the family, and 0 otherwise.

- By using each attribute to find which class of disease distributed in the range of 0 to 3.

- 0 means no disease content.

- 3 means high amount of disease content is there.

- 1 & 2 intermediate amount of disease content is there

7.2.2 Count and Distribution of Histopathological Features using Histplot.

In [ ]:
```python
# List of histopathological features related to a medical condition.

Histopath_Features = [
    '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'
]
```

In [ ]:
```
# It will create a new figure with a fixed size and background color.
plt.figure(figsize=(12,28), facecolor = 'white')

# Initializing a counter for subplot placement.
plotnumber = 1

# Looping through each histopathological features in the data.
for Histopath_Features in df_copy :

    # Limiting the number of subplots to 22.
    if plotnumber <= 22 :

        # creating a subplot in an 8*3 grid.
        ax = plt.subplot(8,3,plotnumber)

        # Plotting a histogram with KDE for the current histopathologic
        sns.histplot(x = df_copy[Histopath_Features], kde = True)

        # Setting labels for the x and y axes.
        plt.xlabel(Histopath_Features, fontsize = 15)
        plt.ylabel('count', fontsize = 15)

    # Incrementing the plot number for the next iteration or
    # Move to the next subplot.
    plotnumber = plotnumber + 1

# Adjust the layout for better spacing.
plt.tight_layout()

# It will display the plot.
plt.show()
```

Insights from histplot of Histopath features

- Attributes have values from 0 to 3, except for the class attribute and eosinophils_in_the_infiltrate.

- eosinophils_in_the_infiltrate has values 0, 1, and 2.

- People with a value of 0 (feature not present) are most likely to belong to all classes.

- Class 1 tends to have the highest values across most attributes (0, 1, 2, 3).

- Hyperkeratosis and parakeratosis show the highest probability of belonging to multiple classes.

- Thinning_of_the_suprapapillary_epidermis and spongiform_pustule have a value of 3 only for class 1.

- Follicular_horn_plug and perifollicular_parakeratosis shows similar patterns.

- The class attribute includes Psoriasis, Lichen_Planus, Seboreic_Dermatitis, Cronic_Dermatitis, Pityriasis_Rosea, and Pityriasis_rubra_pilaris.

- Based on a count plot, Psoriasis is the most common disease, while Pityriasis_rubra_pilaris is the least common.

7.3 distplot for Age of the patient

In [ ]:
```
plt.figure(figsize=(14,5))
plt.title('Age of the patient')
sns.distplot(a=df_copy['Age'], kde=True, bins=20)
plt.axvline(x=np.mean(df_copy['Age']),c='orange',label='Mean Age of the
plt.legend()
```

Insights from Distplot of Age

- A distogram is like a chart that helps us see how ages are distributed in our data. It shows how many times each age appears and what range of ages we have.

- This can help us notice if most ages are similar, spread out evenly, or if there's anything unusaul in age pattern.

7.3.1 Skewness

In [ ]:
```
df['Age'].skew()
```

analysis :-

- A skewness value of 0.10114792694386719 is close to zero, means a slight rightward skewness in the data, but it is generally considered very close to symmetrical.

7.3.2 kurtosis

In [ ]:
```
df['Age'].kurtosis()
```

analysis :-

> - A kurtosis value of -0.6773471369712993 means that the distribution has lighter tails than a normal distribution. It is mildly platykurtic. This means that the distribution has fewer extreme values in the tails compared to a normal distribution.

7.4 Count and Distribution of Classes of Skin Disorders

In [ ]:
```
# It will set the figure size.
plt.figure(figsize=(12,6))

# Creating a countplot for the 'class' column in the data.
ax = sns.countplot(x = df_copy['class'])

# Adding labels to the bars in the countplot.
for label in ax.containers :
    ax.bar_label(label)

# Adjusting the layout for better spacing.
plt.tight_layout()

# It will display the plot.
plt.show()
```

In [ ]:
```
plt.figure(figsize=(14, 7))
sns.countplot(x=df_copy['class'], data=df_copy, palette='bright', order
```

In [ ]:
```
df_copy['class'].value_counts().plot(kind = 'pie', autopct = '%1.1f%%')
```

analysis :-

> - Among all the diseases the highest pecentage is of psoriasis and the lowest is of pityriasis_rubra_pilaris.

8. Bivariate Analysis

In [ ]:
```
# It will set the figure size.
plt.figure(figsize = (25,35))

# Initializing a counter for subplot placement.
plotnumber = 1

# Looping through each column in the data.
for column in df_copy:

    # Limiting the number of subplots to 36.
    if plotnumber <= 36:

        # It will create a subplot in 9*4 grid.
        ax=plt.subplot(9,4,plotnumber)

        # It will create a countplot for the current column, with color
        sns.countplot(x = df_copy[column], data = df_copy, hue = 'class

        # It will set labels for the x and y axes.
        plt.xlabel(column,fontsize = 15)
        plt.ylabel('Count',fontsize = 15)

    # Incrementing the plot number for the next iteration or
    # Move to the next subplot.
    plotnumber = plotnumber + 1

# It will display the plot.
plt.tight_layout()
```

Insights from Bivariate Analysis

- Skin diseases can occur even if there is no family history.

- One type of skin problem shows signs like strange bumps, mouth involvement, pigment changes, and harm to a specific layer.

- Another type, class 1 disease, affects the scalp, causing changes like ridge clubbing, tiny abscesses, skin thining, and pimplr-like blisters.

- Good skin health is indicated by the absence of redness, scaling, thickness of the skin, and inflammation.

9. Age vs class

In [ ]:
```

plt.figure()
plt.figure(figsize=(10,5))
x=df.head(20)['Age']
y=df.head(20)['class']

#Scatter Plot with Regression Line
sns.regplot(x=x,y=y)
plt.title('Age vs class')
plt.xlabel('Age')
plt.ylabel('class')

plt.show()
```

Insights from Age vs class

- Age between 20 to 45 are more prone to skin diseases.

10. Age for each class based on family history

We are going to create swarm plot to visualize the distribution of ages ('Age') for each 'class' in a dataset. Additionally, the points are colored based on the presence or absence of family history ('family_history').

In [ ]:
```
# It will set the figure size.
plt.figure(figsize=(15,8))

# It will create a swarm plot to show the distribution of ages('Age') f
# "x = 'class' " : The x-axis represents different categories based on
# "y = 'Age' " : The y-axis shows the distribution of ages from the 'Ag
# "hue = 'family_history' " : The points are colored based on whether t

sns.swarmplot (x = 'class',
               y = 'Age',
               data = df_copy,
               hue = 'family_history'
              )

# It will display the plot.
plt.show()
```

Insights from swarmplot

> - Pityriasis_Rubra_Pilaris is only found ages below 20. This disease show high relation with family_history. It was observed in family_history of almost half of the patients.

> - Except Pityriasis_Rubra_Pilaris, remaining class of diseases are found in almost all the ages.

> - Psoriasis is also significantly related to the family history

11. Visualizing Class Correlations

We are going to plot bivariate analysis between every single syndrome (clinical features and histopathological features) with different class of diseases . It help us to find how every syndrome effect the different class of diseases .

Every class has 4 different degree.

> - degree 0 - 0 indicates that the feature was not present
> - degree 1 - 1 indicates that the feature was intermediately present
> - degree 2 - 2 indicates that the feature was intermediately present(higher then degree 1)
> - degree 3 - 3 indicates that the feature was highly responsible for the diseases

In [ ]:
```
import seaborn as sns
columns = list(df_copy.columns)
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

> - erythema is intermediately responsible for all diseases. But in some cases it highly responsible for seboreic_dermatitis and psoriasis.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

> - scaling is intermediately responsible for all diseases.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- definite_borders (skin lesion) is intermediately responsible for seboreic_dermatitis , psoriasis diseases.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- itching is responsible for psoriasis , seboreic_dermatitis , lichen_planus , cronic_dermatitis but is not responsible forpityriasis_rosea and pityriasis_rubra_pilaris

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- koebner_phenomenon is not responsible for any severe skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- polygonal_papules is only responsible for lichen_planus skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac

```

analysis:-

- follicular_papules is not responsible for any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- oral_mucosal_involvement is only responsible for lichen_planus skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- knee_and_elbow_involvement is not responsible for any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- scalp_involvement is not responsible for any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- family_history is miner responsible for skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- melanin_incontinence is only responsible for lichen_planus skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- eosinophils_in_the_infiltrate is not responsible for any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="sta
```

analysis:-

- PNL_infiltrate is intermediately responsible for seboreic_dermatitis and psoriasis skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="sta
```

analysis:-

- fibrosis_of_the_papillary_dermis is syndrome of cronic_dermatitis skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="sta
```

analysis:-

- exocytosis is responsible of all skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="sta
```

analysis:-

- acanthosis is intermediately responsible of all skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="sta
```

analysis:-

- hyperkeratosis is miner responsible of all skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- parakeratosis is intermediately responsible of all skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- clubbing_of_the_rete_ridges is only responsible of psoriasis skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- elongation_of_the_rete_ridges is intermediately responsible of psoriasis , seboreic_dermatitis and cronic_dermatitis skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- thinning_of_the_suprapapillary_epidermis is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- spongiform_pustule is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- munro_microabcess is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- focal_hypergranulosis is only responsible of lichen_planus skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- disappearance_of_the_granular_layer is only responsible of psoriasis skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- vacuolisation_and_damage_of_basal_layer is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- spongiosis is responsible of all skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- saw-tooth_appearance_of_retes is miner responsible of lichen_planus skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- follicular_horn_plug is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- perifollicular_parakeratosis is not responsible of any skin disorder.

In [ ]:
```
plt.figure(figsize = (10,5))
sns.kdeplot(data=df_copy, x=columns.pop(0), hue="class", multiple="stac
```

analysis:-

- perifollicular_parakeratosis is intermediately responsible of all skin disorder.

In [ ]:
```

```

12. Feature Engineering

Finding correlation among data

- If it's a Regression, We check correlation between input and output variables.

- If it's a Classification, We check correlation among the input variables.

- For adject correlation we are going to plot heatmap and find variance_inflation_factor and then after analyze both for correlation.

12.1 Creating a deep copy of dataset for exploratory data analysis

- we are here going to creat another copy of dataset for finding correlation.

In [ ]:
```
df_copy_2=df.copy(deep=True)
```

12.2 Heatmap

- Using heatmap plots of all the features to obtain correlation of the Skin Disorder with other features

heatmap depicting intervariable dependency

In [ ]:
```
# Setting plot size
plt.figure(figsize=(30,30))

# Plotting Heatmap
sns.heatmap(df_copy_2.corr(), annot= True, cmap= 'turbo')

# Setting title
plt.title('heatmap depicting intervariable dependency'.upper(), fontsiz

plt.show()
```

Insights from Heatmap

- follicular_papules , fibrosis_of_the_papillary_dermis , follicular_horn_plug , perifollicular_parakeratosis are most correlated with class(target variable)

- The Age attribute is more correlated with focal_hypergranulosis and least correlated with follicular_papules column.

- 20 columns are negetively correlated with target(class) column and 14 columns are positively correlated with target column.

- band-like_infiltrate , polygonal_papules , oral_mucosal_involvement , melanin_incontinence , focal_hypergranulosis , vacuolisation_and_damage_of_basal_layer , saw-tooth_appearance_of_retes are highly correlated with each other.

12.3 variance_inflation_factor for Correlation

- Variance Inflation Factors (VIFs) measure the correlation among independent variables in least squares regression models.Multicollinearity is correlation amongst the independent variables. Consequently, it seems logical to assess the pairwise correlation between all independent variables (IVs) in the model.

In [ ]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_fac
```

In [ ]:
```python
def get_vif(df_copy_2):
    vif=pd.DataFrame()
    vif['variables']=df_copy_2.columns
    vif['VIF']=[variance_inflation_factor(df_copy_2.values,i) for i in

    return vif

```

In [ ]:
```python
get_vif(df_copy_2[[i for i in df_copy_2.describe().columns]])
```

Multicollinearity Analysis Report

High VIF (Above 10):

- Erythema, Scaling, Polygonal Papules, Acanthosis, Clubbing of the Rete Ridges, Elongation of the Rete Ridges, Vacuolisation and Damage of Basal Layer, Saw-Tooth Appearance of Retes, Band-Like Infiltrate, Age, and Class exhibit high VIF, indicating strong correlation with other variables. This may affect the reliability of the regression model.

Moderate VIF (5 to 10):

- Definite Borders, Itching, Follicular Papules, Oral Mucosal Involvement, Knee and Elbow Involvement, Fibrosis of the Papillary Dermis, Exocytosis, Spongiform Pustule, Munro Microabcess, Focal Hypergranulosis, Thinning of the Suprapapillary Epidermis, Disappearance of the Granular Layer, Spongiosis, Follicular Horn Plug, Perifollicular Parakeratosis, Inflammatory Mononuclear Infiltrate, and Class have moderate VIF values, indicating moderate correlation.

Low VIF (Below 5):

- Koebner Phenomenon, Eosinophils in the Infiltrate, PNL Infiltrate, Hyperkeratosis, Parakeratosis, and Age have lower VIF values, indicating relatively lower correlation with other variables.

After analyzing heatmap and variance_inflation_factor some of the columns are identical upto 90 % but in healthcare dataset every syndrome is important to find disease so we are not going to operate anything on columns like drop the columns etc. But we have to be very careful during model creation because it may create baising or overfitting.

In [ ]:
```

```

13. Model Creation

Split data into x and y

In [ ]:
```
# Here, 'x' is all the variables except target.
# Extracting features (independent variables) by dropping the "class" c

x = df_copy_2.drop('class', axis=1)

y = df_copy_2[['class']]
```

- 'x' represents the features (independent variables) of our dataset. excluding the "class" column.
- 'y' represents the target variable (dependent variable) , which is the "class" column.

In [ ]:
```
x.head()
```

In [ ]:
```
y.head()
```

Split Data for Training and Testing

- We will divide our data into training and testing.
- It works on (70:30) or (80:20) rule. It means 70% of the data will be used for training and 30% data will be used for testing.
- Whichever i.e, (70:30) or (80:20) will give best result. We will go with that result.
- In order to test the model, We take a data which is not seen by Model.

In [ ]:
```
# Importing the necessary module for splitting the dataset.
from sklearn.model_selection import train_test_split
```

splitting the dataset into training and testing set.

- x_train : Features for training.
- x_test : Features for testing.
- y_train : Traget variable for training.
- y_test : Target variable for testing.
- test_size : The "test_size" parameter is set to 25% that means 25% of the dataset will be used for testing,
- While the remaining 75% data will be used for training.
- random_state : random_state is set to 42 for reproducibility,
- It means if we run the code with the same random state, we will get the same split each time.

In [ ]:
```
# splitting the dataset into training and testing set.
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,rar
```

In [ ]:
```
# shape of training and testing data.

print(f'\033[1;31m  \033[1m The shape of x_train:',{x_train.shape})
print(f'\033[1;31m  \033[1m The shape of x_test:', {x_test.shape})
print(f'\033[1;31m  \033[1m The shape of y_train', y_train.shape)
print(f'\033[1;31m  \033[1m The shape of y_test:', y_test.shape)
```

In [ ]:
```
# It will count the occurence of each unique value in the 'y_train' var

y_train.value_counts()
```

In [ ]:
```

```

14. Logistic Regression

Model Building

In [ ]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import f1_score
```

14.1 Model creation

For creating a LogisticRegression model , we are going to implement a pipeline for logistic regression with feature scaling using StandardScaler and hyperparameter tuning using GridSearchCV.

1. Pipeline Creation:

- A pipeline (pipe_LR) is created using Pipeline from scikit-learn.
- The pipeline consists of two steps:
  - Step 1 ('scaler'): Standardizing the features using StandardScaler().
  - Step 2 ('classifier'): Applying Logistic Regression using LogisticRegression().

2. Hyperparameter Grid:

- We define a hyperparameter grid (param_grid_LR) for Logistic Regression.
- The grid includes parameters such as regularization strength (C), regularization type (penalty), and the solver for logistic regression (solver).

GridSearchCV:

- We use GridSearchCV to perform a search over the hyperparameter grid within the defined pipeline.
- The estimator is set to our pipeline (pipe_LR), and the hyperparameter grid is set to param_grid_LR.
- Cross-validation with 5 folds (cv=5) is used to evaluate the performance.
- return_train_score=True ensures that training scores are also returned.

Model Training:

- The grid search is fit on the training data (x_train, y_train).

Print Best Results:

- The code prints the best score achieved by the model and the corresponding best hyperparameters.

In [ ]:
```
# Create a pipeline with StandardScaler and LogisticRegression
pipe_LR = Pipeline([
    ('scaler', StandardScaler()), # Step 1: Standardize the features
    ('classifier', LogisticRegression()) # Step 2: Logistic Regression
    ])

# Define the hyperparameter grid for Logistic Regression
param_grid_LR = {
    'classifier__C': [0.001, 0.01, 0.1, 1, 10, 100],  # Regularization
    'classifier__penalty': ['l1', 'l2'],  # Regularization type
    'classifier__solver': ['liblinear']  # Solver for logistic regressi
}

# Create GridSearchCV with the pipeline
grid_search_LR = GridSearchCV(estimator=pipe_LR,
                              param_grid=param_grid_LR,
                              cv =5,
                              return_train_score=True)

# Fit the grid search on the training data
grid_search_LR.fit(x_train, y_train)
print(f" Best score is: {grid_search_LR.best_score_} with parameters: 
```

14.2 Fit the grid search on the training data

In [ ]:
```

# Fit the grid search on the training data
grid_search_LR.fit(x_train, y_train)

```

14.3 best parameters and best estimator

In [ ]:
```
# Get the best parameters and best estimator
best_params = grid_search_LR.best_params_
best_estimator = grid_search_LR.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for LogisticRegressio
print()
print(f'\033[1;31m  \033[1m  best estimator for LogisticRegression:',be


```

14.4 Make Predictions/ Test model

In [ ]:
```
# Make Testing set predictions on the test data using the best estimato
y_pred_LR = best_estimator.predict(x_test)
y_pred_LR
```

In [ ]:
```
# Training set predictions using Logistic Regression.
y_pred_LR_train = best_estimator.predict(x_train)
y_pred_LR_train
```

14.5 Evaluating the Model

14.5.1 Confusion Matrix

- Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels and predicted labels.

In [ ]:
```
# Importing the ConfusionMatrixDisplay class from 'sklearn.metrics' moc
from sklearn.metrics import ConfusionMatrixDisplay

```

In [ ]:
```
# Plot non-normalized confusion matrix
# Define titles and options for confusion matrix display
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
# Loop through each title and normalization option
for title, normalize in titles_options:
    # Fit the grid search on the training data and create a confusion m
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_LR.fit(x_train, y_train),# Fit the grid search on t
        x_test,# Test features
        y_test,# True labels for the test set
        display_labels=['1', '2', '3', '4','5','6'],# Class labels
        cmap=plt.cm.Greens,# Color map for the display
        normalize=normalize,# Normalization option
    )

    disp.ax_.set_title(title)# Set the title for the confusion matrix c

    print(title)# Print title and confusion matrix
    print(disp.confusion_matrix)

plt.show()# Show the plots
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**
   - True Positives (TP): 34
   - No False Positives (FP), False Negatives (FN), or True Negatives (TN).
2. **Seboreic Dermatitis (Class 2):**
   - TP: 12
   - No FP, FN, or TN.

3. **Lichen Planus (Class 3):**
    - TP: 17
    - No FP, FN, or TN.
4. **Pityriasis Rosea (Class 4):**
    - TP: 9
    - Two False Positives (mistakenly classified as Class 2), and no FN or TN.
5. **Cronic Dermatitis (Class 5):**
    - TP: 14
    - No FP, FN, or TN.
6. **Pityriasis Rubra Pilaris (Class 6):**
    - TP: 4
    - No FP, FN, or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:
```
# Plot the comparison between the actual and predicted bike rental cour
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_LR, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], cc
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix of Logistic Regression :

This confusion matrix gives a lot of information about the model’s performance:

> - As usual, the diagonal elements are the correctly predicted samples. A total of 90 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 97.82 %.

> - M_42=2 implies that the model does confuse samples originally belonging to class-4 (Pityriasis Rosea) with class-2 (Seboreic Dermatitis) , i.e., the classification boundary between classes 4 and 2 was not learned well by the classifier.

In [ ]:
```

```

14.5.2 Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

Mean Squared Error

> - The Mean Squared Error (MSE) or Mean Squared Deviation (MSD) of an estimator measures the average of error squares i.e. the average squared difference between the estimated values and true value.

r2_score

> - The coefficient of determination (R^2) is a statistical measure that indicates how well a regression model predicts an outcome.

Mean Absolute Error

> - Mean Absolute Error (MAE) is a metric that measures the average absolute difference between the predicted values and the actual target values in a regression model.

Root Mean Squared Error

> - Root Mean Squared Error (RMSE) is a metric that measures the average difference between the predicted values and the actual values in a regression model.

> - The lower the RMSE, the better a given model is able to “fit” a dataset.

In [ ]:
```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test,y_pred_LR)
r2 = metrics.r2_score(y_test, y_pred_LR)
mae = metrics.mean_absolute_error(y_test, y_pred_LR)
rmse = metrics.mean_squared_error(y_test, y_pred_LR, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

> - MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

> - R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

> - MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

> - RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

14.5.3 Accuracy Score

- It will print the accuracy scores for training and testing sets using Logistic Regression.

In [ ]:
```
# Importing metrics for evaluating models.
from sklearn.metrics import confusion_matrix,accuracy_score,precision_s
```

In [ ]:
```
#accuracy of Logistic Regression for testing set
accuracy_LR = accuracy_score(y_test, y_pred_LR)
#accuracy of Logistic Regression for training set
accuracy_LR_train = accuracy_score(y_train, y_pred_LR_train)
print(f'\033[1;31m  \033[1m  accuracy of Logistic Regression for testir
print(f'\033[1;31m  \033[1m  accuracy of Logistic Regression for traini
```

14.5.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test, y_pred_LR, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of Logistic Regression:',p
```

14.5.5. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test, y_pred_LR, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of Logistic Regression:',f1_score
```

14.5.6. Recall Score

- It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test, y_pred_LR, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of Logistic Regression:',reca
```

14.5.7. Classification Report

In [ ]:
```
print(classification_report(y_test, y_pred_LR, target_names=
                            ['psoriasis','seboreic_dermatitis','lichen_
                             'pityriasis_rubra_pilaris']))
```

Classification Report Analysis

The classification report provides a detailed summary of the performance of a classification model on a per-class basis. Let's analyze the report:

1. **Psoriasis:**

   - Precision: 1.00 (100%) - All instances predicted as Psoriasis are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Psoriasis.
   - F1-Score: 1.00 (100%) - The harmonic mean of precision and recall is excellent.
   - Support: 34 instances.

2. **Seborrheic Dermatitis:**
    - Precision: 0.86 (86%) - Some instances predicted as Seborrheic Dermatitis may be incorrect.
    - Recall: 1.00 (100%) - The model captures all actual instances of Seborrheic Dermatitis.
    - F1-Score: 0.92 (92%) - A good balance between precision and recall.
    - Support: 12 instances.
3. **Lichen Planus:**
    - Precision: 1.00 (100%) - All instances predicted as Lichen Planus are correct.
    - Recall: 1.00 (100%) - The model captures all actual instances of Lichen Planus.
    - F1-Score: 1.00 (100%) - Excellent performance on this class.
    - Support: 17 instances.
4. **Pityriasis Rosea:**
    - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rosea are correct.
    - Recall: 0.82 (82%) - The model may miss some instances of Pityriasis Rosea.
    - F1-Score: 0.90 (90%) - A relatively good balance between precision and recall.
    - Support: 11 instances.
5. **Chronic Dermatitis:**
    - Precision: 1.00 (100%) - All instances predicted as Chronic Dermatitis are correct.
    - Recall: 1.00 (100%) - The model captures all actual instances of Chronic Dermatitis.
    - F1-Score: 1.00 (100%) - Excellent performance on this class.
    - Support: 14 instances.
6. **Pityriasis Rubra Pilaris:**
    - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rubra Pilaris are correct.
    - Recall: 1.00 (100%) - The model captures all actual instances of Pityriasis Rubra Pilaris.
    - F1-Score: 1.00 (100%) - Excellent performance on this class.
    - Support: 4 instances.

The model performs exceptionally well across most classes, with high precision, recall, and F1-Score. However, it shows a slight decrease in recall for 'Pityriasis Rosea,' indicating a potential area for improvement. The overall high accuracy and balanced metrics suggest that the model is effective in classifying skin diseases.

In [ ]:
```

```

15. KNN (K Nearest Neighbors) Classifier

Model Building

In [ ]:
```
# Using K-Nearest Neighbors (KNN) classifier to predict target values f

from sklearn.neighbors import KNeighborsClassifier
```

15.1 model creation

For creating a KNeighborsClassifier model , we are going to implement a pipeline for KNN (K Nearest Neighbors) Classifier with feature scaling using StandardScaler and hyperparameter tuning using GridSearchCV.

1. Pipeline Creation:

> - A pipeline (pipe_knn) is created using Pipeline from scikit-learn.
> - The pipeline consists of two steps:
>   - Step 1 ('sc'): Standardizing the features using StandardScaler().
>   - Step 2 ('knn'): Applying KNN (K Nearest Neighbors) Classifier using KNeighborsClassifier().

2.Hyperparameter Grid:

> - We define a hyperparameter grid (params_knn) for KNN (K Nearest Neighbors) Classifier.
> - The grid includes the number of neighbors (n_neighbors) ranging from 1 to 19.

3. GridSearchCV:

> - We use GridSearchCV to perform a search over the hyperparameter grid within the defined pipeline.
> - The estimator is set to our pipeline (pipe_knn), and the hyperparameter grid is set to params_knn.
> - Cross-validation with 5 folds (cv=5) is used to evaluate the performance.
> - return_train_score=True ensures that training scores are also returned.

4. Model Training:

> - The grid search is fit on the training data (x_train, y_train).

5. Print Best Results:

> - The code prints the best score achieved by the model and the corresponding best hyperparameters.

In [ ]:
```
# Create a pipeline with StandardScaler and KNN (K Nearest Neighbors) C
pipe_knn = Pipeline([
    ('sc', StandardScaler()), # Step 1: Standardize the features
    ('knn', KNeighborsClassifier())# Step 2:KNN (K Nearest Neighbors) C
    ])

# Define the hyperparameter grid for KNN (K Nearest Neighbors) Classifi
params_knn = {
    'knn__n_neighbors': range(1, 20)
    }

# Create GridSearchCV with the pipeline
grid_search_knn = GridSearchCV(estimator=pipe_knn,
                          param_grid=params_knn,
                          cv = 5,
                          return_train_score=True)

grid_search_knn .fit(x_train, y_train)
print(f" Best score is: {grid_search_knn.best_score_} with parameters:
```

15.2 best parameters and best estimator

In [ ]:
```
# Get the best parameters and best estimator
best_params = grid_search_knn.best_params_
best_estimator = grid_search_knn.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for KNeighborsClassifi
print()
print(f'\033[1;31m  \033[1m  best estimator for KNeighborsClassifier:',

```

15.3 Fit the grid search on the training data

In [ ]:
```

# Fit the grid search on the training data
grid_search_knn.fit(x_train, y_train)
```

15.4 Make Predictions/ Test model

In [ ]:
```
# Make predictions on the test data using the best estimator
y_pred_knn = best_estimator.predict(x_test)
y_pred_knn
```

In [ ]:
```
# Making predictions on the training data using the trained KneighborsC
y_pred_knn_train = best_estimator.predict(x_train)
y_pred_knn_train
```

15.5 Evaluating the Model

15.5.1 Confusion Matrix

- Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels and predicted labels .

In [ ]:
```
# Plot non-normalized confusion matrix
# Define titles and options for confusion matrix display
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
# Loop through each title and normalization option
for title, normalize in titles_options:
    # Fit the grid search on the training data and create a confusion m
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_knn .fit(x_train, y_train),# Fit the grid search on
        x_test,# Test features
        y_test,# True labels for the test set
        display_labels=['1', '2', '3', '4','5','6'],# Class labels
        cmap=plt.cm.Greens,# Color map for the display
        normalize=normalize,# Normalization option
    )
    disp.ax_.set_title(title) # Set the title for the confusion matrix


    print(title) # Print title and confusion matrix
    print(disp.confusion_matrix)

plt.show()# Show the plots
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**
   - True Positives (TP): 33
   - One False Positive (misclassified as Class 2), and no False Negatives (FN) or True Negatives (TN).
2. **Seboreic Dermatitis (Class 2):**
   - TP: 12
   - No FP, FN, or TN.
3. **Lichen Planus (Class 3):**
   - TP: 17
   - No FP, FN, or TN.
4. **Pityriasis Rosea (Class 4):**
   - TP: 9
   - Two False Positives (mistakenly classified as Class 2), and no FN or TN.
5. **Cronic Dermatitis (Class 5):**
   - TP: 14

- No FP, FN, or TN.

6. **Pityriasis Rubra Pilaris (Class 6):**
   - TP: 4
   - No FP, FN, or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:
```
# Plot the comparison between the actual and predicted Skin Disorder co
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_knn, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], co
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix

This confusion matrix gives a lot of information about the model's performance:

> - As usual, the diagonal elements are the correctly predicted samples. A total of 89 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 96.73 %.

> - M_42=2 implies that the model does confuse samples originally belonging to class-4 with class-2, i.e., the classification boundary between classes 4 and 2 was not learned well by the classifier.

> - M_12=1 implies that the model does confuse samples originally belonging to class-1 with class-2, i.e., the classification boundary between classes 1 and 2 was not learned well by the classifier.

15.5.2 Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

In [ ]:

```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test,y_pred_knn)
r2 = metrics.r2_score(y_test, y_pred_knn)
mae = metrics.mean_absolute_error(y_test,y_pred_knn)
rmse = metrics.mean_squared_error(y_test, y_pred_knn, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

- MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

- R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

- MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

- RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

15.5.3. Accuracy Score

- It will print the accuracy scores for training and testing sets using KNeighborsClassifier.

In [ ]:

```
#accuracy of KNeighborsClassifier for testing set
accuracy_knn = accuracy_score(y_test,y_pred_knn)
#accuracy of KNeighborsClassifier for training set
accuracy_knn_train = accuracy_score(y_train,y_pred_knn_train)
print(f'\033[1;31m  \033[1m  accuracy of KNeighborsClassifier for testi
print(f'\033[1;31m  \033[1m  accuracy of KNeighborsClassifier for trair
```

15.5.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test,y_pred_knn, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of KNeighborsClassifier:',
```

15.5.5. Recall Score

-It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test, y_pred_knn, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of KNeighborsClassifier:',rec
```

15.5.6. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test,y_pred_knn, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of KNeighborsClassifier:',f1_scor
```

15.6 Classification Report

In [ ]:
```
print(classification_report(y_test, y_pred_knn, target_names=
                            ['psoriasis','seboreic_dermatitis','lichen_
                             'pityriasis_rubra_pilaris']))
```

KNN Model Classification Report Analysis

1. **Psoriasis:**
    - Precision: 1.00 (100%) - High precision, indicating few instances predicted as Psoriasis are incorrect.
    - Recall: 0.97 (97%) - The model captures most actual instances of Psoriasis.
    - F1-Score: 0.99 (99%) - A good balance between precision and recall.
    - Support: 34 instances.
2. **Seborrheic Dermatitis:**

- Precision: 0.80 (80%) - Some instances predicted as Seborrheic Dermatitis may be incorrect.
- Recall: 1.00 (100%) - The model captures all actual instances of Seborrheic Dermatitis.
- F1-Score: 0.89 (89%) - A moderate balance between precision and recall.
- Support: 12 instances.

3. **Lichen Planus:**

   - Precision: 1.00 (100%) - All instances predicted as Lichen Planus are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Lichen Planus.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 17 instances.

4. **Pityriasis Rosea:**

   - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rosea are correct.
   - Recall: 0.82 (82%) - The model may miss some instances of Pityriasis Rosea.
   - F1-Score: 0.90 (90%) - A moderate balance between precision and recall.
   - Support: 11 instances.

5. **Chronic Dermatitis:**

   - Precision: 1.00 (100%) - All instances predicted as Chronic Dermatitis are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Chronic Dermatitis.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 14 instances.

6. **Pityriasis Rubra Pilaris:**

   - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rubra Pilaris are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Pityriasis Rubra Pilaris.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 4 instances.

Accuracy and Averaged Metrics:

- **Accuracy:**
  - 97% - Overall, the model is accurate in its predictions.
- **Macro Avg (Macro-Averaging):**
  - Precision: 97% - The average precision across all classes.
  - Recall: 96% - The average recall across all classes.
  - F1-Score: 96% - The average F1-Score across all classes.
- **Weighted Avg (Weighted-Averaging):**
  - Precision: 97% - The weighted average precision considering class imbalance.
  - Recall: 97% - The weighted average recall considering class imbalance.
  - F1-Score: 97% - The weighted average F1-Score considering class imbalance.

Conclusion:

The KNN model demonstrates good performance, especially on classes like 'Lichen Planus' and 'Chronic Dermatitis,' achieving high precision, recall, and F1-Score. However, it shows a decrease in recall for 'Pityriasis Rosea,' suggesting potential challenges in correctly identifying instances of this class. The overall accuracy and balanced metrics suggest that

In [ ]:
```

```

16. Support Vector Classifier

Model Building

In [ ]:
```python
# Importing Support Vector Machine (SVM) classifier from scikit-learn.
from sklearn.svm import SVC
```

16.1 model creation

For creating a Support Vector Machine model, we are going to implement a pipeline for the Support Vector Classifier (SVC) with feature scaling using StandardScaler and hyperparameter tuning using GridSearchCV.

1. Define Hyperparameter Grid:

> - Define the hyperparameter grid for the Support Vector Classifier (SVC) with different kernel options (rbf, poly, sigmoid) and regularization strength (C).

2. Pipeline Creation:

> - Create a pipeline (pipe_svm) using Pipeline from scikit-learn.
> - The pipeline consists of two steps:
>   - Step 1 ('sc'): Standardizing the features using StandardScaler().
>   - Step 2 ('SVM'): Applying Support Vector Classifier (SVC) using SVC().

3. Hyperparameter Grid:

> - Define a hyperparameter grid (params_svm) for the SVC model, including options for regularization strength (C) and kernel type (kernel).

4. GridSearchCV:

- Use GridSearchCV to perform a search over the hyperparameter grid within the defined pipeline.
- The estimator is set to our pipeline (pipe_svm), and the hyperparameter grid is set to params_svm.
- Cross-validation with 5 folds (cv=5) is used to evaluate the performance.
- return_train_score=True ensures that training scores are also returned.

5. Model Training:

In [ ]:
```
kernels = ['rbf', 'poly', 'sigmoid']
C = np.logspace(-2, 10, 13)

# Create a pipeline with StandardScaler and SVC
pipe_svm = Pipeline([
    ('sc', StandardScaler()),# Step 1: Standardize the features
    ('SVM', SVC())# Step 2: SVC model
    ])

# Define the hyperparameter grid for SVC
params_svm = {'SVM__C': C,
              'SVM__kernel': kernels,
             }

# Create GridSearchCV with the pipeline
grid_search_svm = GridSearchCV(estimator=pipe_svm,
                      param_grid=params_svm,
                      cv = 5,
                      return_train_score=True)

# Fit the grid search on the training data
grid_search_svm.fit(x_train, y_train)
print(f" Best score is: {grid_search_svm.best_score_} with parameters:

```

16.2. best parameters and best estimator

In [ ]:
```
# Get the best parameters and best estimator
best_params = grid_search_svm.best_params_
best_estimator = grid_search_svm.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for SVC:',best_params)
print(f'\033[1;31m  \033[1m  best estimator for SVC:',best_estimator)
```

16.3. Fit the grid search on the training data

In [ ]:
```
# Fit the grid search on the training data
grid_search_svm.fit(x_train, y_train)

```

16.4. Make Predictions/ Test model

In [ ]:
```
# Make predictions on the test data using the best estimator
y_pred_svm = best_estimator.predict(x_test)
y_pred_svm
```

In [ ]:
```
# Making predictions on the training data using the trained SVM classi
y_pred_svm_train = best_estimator.predict(x_train)
y_pred_svm_train
```

16.5. Evaluating the Model

16.5.1. Confusion Matrix

- Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels and predicted labels .

In [ ]:
```
# Plot non-normalized confusion matrix
# Define titles and options for confusion matrix display
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
# Loop through each title and normalization option
for title, normalize in titles_options:
    # Fit the grid search on the training data and create a confusion m
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_svm.fit(x_train, y_train),# Fit the grid search on
        x_test,# Test features
        y_test,# True labels for the test set
        display_labels=['1', '2', '3', '4','5','6'],# Class labels
        cmap=plt.cm.Greens, # Color map for the display
        normalize=normalize,# Normalization option
    )
    disp.ax_.set_title(title)# Set the title for the confusion matrix d

    print(title)# Print title and confusion matrix
    print(disp.confusion_matrix)

plt.show()# Show the plots
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**
   - True Positives (TP): 34
   - No False Positives (FP), False Negatives (FN), or True Negatives (TN).
2. **Seboreic Dermatitis (Class 2):**

- TP: 12
- No FP, FN, or TN.

3. **Lichen Planus (Class 3):**
   - TP: 17
   - No FP, FN, or TN.
4. **Pityriasis Rosea (Class 4):**
   - TP: 9
   - Two False Positives (misclassified as Class 2), and no FN or TN.
5. **Cronic Dermatitis (Class 5):**
   - TP: 14
   - No FP, FN, or TN.
6. **Pityriasis Rubra Pilaris (Class 6):**
   - TP: 4
   - No FP, FN, or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:
```python
# Plot the comparison between the actual and predicted Skin Disorder co
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_svm, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], co
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix

This confusion matrix gives a lot of information about the model’s performance:

> - As usual, the diagonal elements are the correctly predicted samples. A total of 90 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 97.82 %.

> - M_42=2 implies that the model does confuse samples originally belonging to class-4 with class-2, i.e., the classification boundary between classes 4 and 2 was not learned well by the classifier.

16.5.2. Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

In [ ]:
```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test,y_pred_svm)
r2 = metrics.r2_score(y_test, y_pred_svm)
mae = metrics.mean_absolute_error(y_test,y_pred_svm)
rmse = metrics.mean_squared_error(y_test, y_pred_svm, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

- MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

- R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

- MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

- RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

16.5.3. Accuracy Score

- It will print the accuracy scores for training and testing sets using SVC.

In [ ]:
```
accuracy_svm = accuracy_score(y_test, y_pred_svm)
accuracy_svm_train = accuracy_score(y_train, y_pred_svm_train)
print(f'\033[1;31m  \033[1m  accuracy of SVC for testing set  :',accura
print(f'\033[1;31m  \033[1m  accuracy of SVC for training set :',accura
```

16.5.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test,y_pred_svm, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of SVC:',precision)
```

16.5.5. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test,y_pred_svm, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of SVC:',f1_score)
```

16.5.6. Recall Score

- It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test, y_pred_svm, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of SVC:',recall)
```

16.6. Classification Report

In [ ]:
```
print(classification_report(y_test, y_pred_svm, target_names=
                            ['psoriasis','seboreic_dermatitis','lichen_
                             'pityriasis_rubra_pilaris']))
```

SVM Model Classification Report Analysis

1. **Psoriasis:**
   - Precision: 1.00 (100%) - High precision, indicating few instances predicted as Psoriasis are incorrect.
   - Recall: 1.00 (100%) - The model captures all actual instances of Psoriasis.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 34 instances.
2. **Seborrheic Dermatitis:**

- Precision: 0.86 (86%) - Some instances predicted as Seborrheic Dermatitis may be incorrect.
- Recall: 1.00 (100%) - The model captures all actual instances of Seborrheic Dermatitis.
- F1-Score: 0.92 (92%) - A good balance between precision and recall.
- Support: 12 instances.

3. **Lichen Planus:**

   - Precision: 1.00 (100%) - All instances predicted as Lichen Planus are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Lichen Planus.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 17 instances.

4. **Pityriasis Rosea:**

   - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rosea are correct.
   - Recall: 0.82 (82%) - The model may miss some instances of Pityriasis Rosea.
   - F1-Score: 0.90 (90%) - A moderate balance between precision and recall.
   - Support: 11 instances.

5. **Chronic Dermatitis:**

   - Precision: 1.00 (100%) - All instances predicted as Chronic Dermatitis are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Chronic Dermatitis.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 14 instances.

6. **Pityriasis Rubra Pilaris:**

   - Precision: 1.00 (100%) - All instances predicted as Pityriasis Rubra Pilaris are correct.
   - Recall: 1.00 (100%) - The model captures all actual instances of Pityriasis Rubra Pilaris.
   - F1-Score: 1.00 (100%) - Excellent performance on this class.
   - Support: 4 instances.

Accuracy and Averaged Metrics:

- **Accuracy:**
  - 98% - Overall, the SVM model is accurate in its predictions.
- **Macro Avg (Macro-Averaging):**
  - Precision: 98% - The average precision across all classes.
  - Recall: 97% - The average recall across all classes.
  - F1-Score: 97% - The average F1-Score across all classes.
- **Weighted Avg (Weighted-Averaging):**
  - Precision: 98% - The weighted average precision considering class imbalance.
  - Recall: 98% - The weighted average recall considering class imbalance.
  - F1-Score: 98% - The weighted average F1-Score considering class imbalance.

Conclusion:

The SVM model demonstrates excellent performance across all classes, achieving high precision, recall, and F1-Score. The model is particularly effective in classifying 'Psoriasis,' 'Lichen Planus,' and 'Chronic Dermatitis.' The overall accuracy and balanced metrics suggest that the SVM model is robust and reliable for skin disease classification

In [ ]:
```

```

17. Decision Tree

Model Building

In [ ]:
```
# Importing DecisionTreeClassifier from the 'sklearn.tree'.
from sklearn.tree import DecisionTreeClassifier

```

17.1. Model creation

For creating a DecisionTreeClassifier model , we are going to implement a pipeline for the Decision Tree Classifier with feature scaling using StandardScaler and hyperparameter tuning using GridSearchCV.

Pipeline Creation:

> - Create a pipeline (pipe_decsT) using Pipeline from scikit-learn.
> - The pipeline consists of two steps:
>   - Step 1 ('sc'): Standardizing the features using StandardScaler().
>   - Step 2 ('decsT'): Applying Decision Tree Classifier using DecisionTreeClassifier().

Hyperparameter Grid:

> - Define a hyperparameter grid (params_decsT) for the Decision Tree Classifier model, including options for the criterion (gini or entropy) and maximum depth of the tree (max_depth).

GridSearchCV:

> - Use GridSearchCV to perform a search over the hyperparameter grid within the defined pipeline.
> - The estimator is set to our pipeline (pipe_decsT), and the hyperparameter grid is set to params_decsT.
> - Cross-validation with 5 folds (cv=5) is used to evaluate the performance.
> - return_train_score=True ensures that training scores are also returned.

Model Training:

> - The grid search is fit on the training data (x_train, y_train).

Print Best Results:

- The code prints the best score achieved by the model and the corresponding best hyperparameters.

In [ ]:
```python
# Create a pipeline with StandardScaler and  DecisionTreeClassifier
pipe_decsT = Pipeline([
    ('sc', StandardScaler()),# Step 1: Standardize the features
    ('decsT', DecisionTreeClassifier())# Step 2: DecisionTreeClassifier
    ])

# Define the hyperparameter grid for DecisionTreeClassifier model
params_decsT = {
    'decsT__criterion' : ['gini', 'entropy'],
    'decsT__max_depth' : np.arange(3, 15)
    }

# Create GridSearchCV with the pipeline
grid_search_decsT = GridSearchCV(estimator=pipe_decsT,
                          param_grid=params_decsT,
                          cv = 5,
                          return_train_score=True)

# Fit the grid search on the training data
grid_search_decsT.fit(x_train, y_train)
print(f" Best score is: {grid_search_decsT.best_score_} with parameters
```

17.2. best parameters and best estimator

In [ ]:
```python
# Get the best parameters and best estimator
best_params = grid_search_decsT.best_params_
best_estimator = grid_search_decsT.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for DecisionTreeClassi
print()
print(f'\033[1;31m  \033[1m  best estimator for DecisionTreeClassifier:

```

17.3. Fit the grid search on the training data

In [ ]:
```python

# Fit the grid search on the training data
grid_search_decsT.fit(x_train, y_train)
```

17.4. Make Predictions/ Test model

In [ ]:
```python
# Make predictions on the test data using the best estimator
y_pred_decsT = best_estimator.predict(x_test)
# The predicted labels are stored in the variable 'y_pred'.
y_pred_decsT
```

In [ ]:
```
# Making predictions on the training data using the trained DecisionTre
y_pred_decsT_train = best_estimator.predict(x_train)
y_pred_decsT_train
```

17.5. Decision Tree Regressor Visualization

we are going to used the DecisionTreeRegressor from scikit-learn to fit a decision tree regression model and visualize the resulting tree structure.

In [ ]:
```
# Import necessary libraries
from sklearn import tree

# Specify the target column and feature columns
target_col = "class"
use_cols = list(set(df_copy_2.columns) - set([target_col]))
len(use_cols)

# Create a Decision Tree Regressor with max depth of 3
tree_reg = tree.DecisionTreeRegressor(max_depth=3)
tree_reg.fit(x_train, y_train)

# Plot the Decision Tree
fig = plt.figure(figsize=(20, 5))
tree.plot_tree(tree_reg, feature_names=use_cols, filled=True)
plt.show()
```

Decision Tree Regressor Analysis

> - The Decision Tree Regressor, with a depth of 3, provides a simple and interpretable model for regression tasks. It allows for an understanding of how different features contribute to predictions. Further tuning and exploration of different tree depths may be considered to optimize the model's performance.

17.6. Evaluating the Model

17.6.1. Confusion Matrix

> - Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels and predicted labels

In [ ]:
```python
# Plot non-normalized confusion matrix
# Define titles and options for confusion matrix display
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]

# Loop through each title and normalization option
for title, normalize in titles_options:
    # Fit the grid search on the training data and create a confusion m
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_decsT.fit(x_train, y_train),# Fit the grid search o
        x_test,# Test features
        y_test,# True labels for the test set
        display_labels=['1', '2', '3', '4','5','6'],# Class labels
        cmap=plt.cm.Greens, # Color map for the display
        normalize=normalize, # Normalization option
    )
    disp.ax_.set_title(title) # Set the title for the confusion matrix

    print(title)# Print title and confusion matrix
    print(disp.confusion_matrix)

plt.show() # Show the plots
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**
   - True Positives (TP): 34
   - No False Positives (FP), False Negatives (FN), or True Negatives (TN).
2. **Seboreic Dermatitis (Class 2):**
   - TP: 12
   - No FP, FN, or TN.
3. **Lichen Planus (Class 3):**
   - TP: 16
   - One False Positive (misclassified as Class 2), and no FN or TN.
4. **Pityriasis Rosea (Class 4):**
   - TP: 11
   - No FP, FN, or TN.
5. **Cronic Dermatitis (Class 5):**
   - TP: 14
   - No FP, FN, or TN.
6. **Pityriasis Rubra Pilaris (Class 6):**
   - TP: 3
   - One False Positive (misclassified as Class 5), and no FN or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:
```
# Plot the comparison between the actual and predicted Skin Disorder co
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_decsT, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], co
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix

This confusion matrix gives a lot of information about the model's performance:

- As usual, the diagonal elements are the correctly predicted samples. A total of 89 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 96.73 %.

- M_32=2 implies that the model does confuse samples originally belonging to class-3 with class-2, i.e., the classification boundary between classes 3 and 2 was not learned well by the classifier.

- M_62=1 implies that the model does confuse samples originally belonging to class-6 with class-2, i.e., the classification boundary between classes 6 and 2 was not learned well by the classifier.

17.6.2. Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

In [ ]:
```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test,y_pred_decsT)
r2 = metrics.r2_score(y_test, y_pred_decsT)
mae = metrics.mean_absolute_error(y_test, y_pred_decsT)
rmse = metrics.mean_squared_error(y_test, y_pred_decsT, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

- MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

- R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

- MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

- RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

17.6.3. Accuracy Score

- It will print the accuracy scores for training and testing sets using KNeighborsClassifier.

In [ ]:
```
#accuracy of DecisionTreeClassifier for testing set
accuracy_decsT = accuracy_score(y_test, y_pred_decsT)
#accuracy of DecisionTreeClassifier for training set
accuracy_decsT_train = accuracy_score(y_train, y_pred_decsT_train)
print(f'\033[1;31m  \033[1m  accuracy of DecisionTreeClassifier for tes
print(f'\033[1;31m  \033[1m  accuracy of DecisionTreeClassifier for tra
```

17.6.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test,y_pred_decsT, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of DecisionTreeClassifier:
```

17.6.5. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test, y_pred_decsT, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of DecisionTreeClassifier:',f1_sc
```

17.6.6. Recall Score

- It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test,y_pred_decsT, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of DecisionTreeClassifier:',r
```

17.7. Classification Report

In [ ]:
```
print(classification_report(y_test,y_pred_decsT, target_names=
                               ['psoriasis','seboreic_dermatitis','lichen_
                                'pityriasis_rubra_pilaris']))
```

Decision Tree Classifier Report

1. **Psoriasis:**
   - Perfect precision, recall, and F1-Score (all 1.00), indicating accurate and reliable predictions for psoriasis.
2. **Seboreic Dermatitis:**
   - High precision (0.92) and perfect recall and F1-Score (both 1.00), showcasing strong performance.
3. **Lichen Planus:**
   - Perfect precision, recall, and F1-Score (all 1.00), demonstrating accurate predictions for lichen planus.
4. **Pityriasis Rosea:**
   - Perfect precision, recall, and F1-Score (all 1.00), reflecting precise predictions for pityriasis rosea.
5. **Cronic Dermatitis:**
   - Perfect precision, recall, and F1-Score (all 1.00), indicating excellent performance for cronic dermatitis.
6. **Pityriasis Rubra Pilaris:**
   - High precision (1.00) but lower recall (0.75) and F1-Score (0.86), suggesting some room for improvement.

Model Evaluation Metrics:

1. **Accuracy:**

- Overall accuracy of 99%, indicating a highly reliable model.

2. **Macro Average:**
    - Macro-average precision, recall, and F1-Score are 0.99, 0.96, and 0.97, respectively, showcasing consistent performance across classes.
3. **Weighted Average:**
    - Weighted-average precision, recall, and F1-Score are 0.99, 0.99, and 0.99, respectively, reflecting robust performance considering class imbalances.

Conclusion:

The Decision Tree Classifier, with its high accuracy and balanced precision-recall performance, proves to be a valuable model for predicting skin diseases. It provides detailed insights into each class's predictive capabilities and can be considered a reliable tool in a dermatological diagnostic setting.

In [ ]:
```

```

18. Random Forest Classifier

Model Building

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

```

18.1. Model creation

For creating a RandomForestClassifier model, we have performed hyperparameter tuning for a Random Forest Classifier using Grid Search.

Random Forest Classifier:

> - Create an instance of the Random Forest Classifier (rf_classifier) with a specified random state.

Hyperparameter Grid:

> - Define a hyperparameter grid (param_rf) that includes options for the number of estimators, maximum features, and whether to bootstrap.

GridSearchCV:

> - Create an instance of the GridSearchCV class (grid_search_rf) to perform a search over the hyperparameter grid.
> - Set the estimator to the Random Forest Classifier, the hyperparameter grid to param_rf, and use 5-fold cross-validation (cv=5).
> - return_train_score=True ensures that training scores are also returned.

Model Training:

> - Fit the grid search on the training data (x_train, y_train).

Print Best Results:

In [ ]:
```python
# Random Forest Classifier Hyperparameter Tuning and Grid Search
# Create a Random Forest Classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Define the parameter grid for hyperparameter tuning
param_rf = {
    "n_estimators": [10, 50, 100],
    "max_features": ["auto", "log2", "sqrt"],
    "bootstrap": [True, False]
}


# Create an instance of the GridSearchCV class
grid_search_rf = GridSearchCV(estimator=rf_classifier,
                    param_grid=param_rf,
                    return_train_score=True,
                    cv=5)

# Fit the grid search on the training data
grid_search_rf.fit(x_train, y_train)
print(f" Best score is: {grid_search_rf.best_score_} with parameters: 

```

18.2. best parameters and best estimator

In [ ]:
```python
# Get the best parameters and best estimator
best_params =grid_search_rf.best_params_
best_estimator = grid_search_rf.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for RandomForestClassi
print()
print(f'\033[1;31m  \033[1m  best estimator for RandomForestClassifier:

```

18.3. Fit the grid search on the training data

In [ ]:
```python
# Fit the grid search on the training data
grid_search_rf.fit(x_train, y_train)
```

18.4. Make Predictions/ Test model

In [ ]:
```
# Make predictions on the test data using the best estimator
y_pred_rf = best_estimator.predict(x_test)
# The predicted labels are stored in the variable 'y_pred'.
y_pred_rf
```

In [ ]:
```
# Making predictions on the training data using the trained RandomFores
y_pred_rf_train = best_estimator.predict(x_train)
y_pred_rf_train
```

18.5. Evaluating the Model

18.5.1 Confusion Matrix

- Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels (y_test) and predicted labels (y_test_pred).

In [ ]:
```
# Plot non-normalized confusion matrix
# Define titles and options for confusion matrix display
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
# Loop through each title and normalization option
for title, normalize in titles_options:
    # Fit the grid search on the training data and create a confusion m
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_rf.fit(x_train, y_train),# Fit the grid search on t
        x_test,# Test features
        y_test,# True labels for the test set
        display_labels=['1', '2', '3', '4','5','6'],# Class labels
        cmap=plt.cm.Greens,# Color map for the display
        normalize=normalize,# Normalization option
    )
    disp.ax_.set_title(title)# Set the title for the confusion matrix d

    print(title)# Print title and confusion matrix
    print(disp.confusion_matrix)

plt.show()# Show the plots
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**

   - True Positives (TP): 34

- No False Positives (FP), False Negatives (FN), or True Negatives (TN).

2. **Seboreic Dermatitis (Class 2):**
    - TP: 12
    - No FP, FN, or TN.
3. **Lichen Planus (Class 3):**
    - TP: 17
    - No FP, FN, or TN.
4. **Pityriasis Rosea (Class 4):**
    - TP: 10
    - One False Positive (misclassified as Class 3), and no FN or TN.
5. **Cronic Dermatitis (Class 5):**
    - TP: 14
    - No FP, FN, or TN.
6. **Pityriasis Rubra Pilaris (Class 6):**
    - TP: 3
    - No FP, FN, or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:
```python
# Plot the comparison between the actual and predicted Skin Disorder

plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_rf, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], co
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix

This confusion matrix gives a lot of information about the model’s performance:

> - As usual, the diagonal elements are the correctly predicted samples. A total of 90 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 97.82 %.

> - M_42=1 implies that the model does confuse samples originally belonging to class-4 with class-2, i.e., the classification boundary between classes 4 and 2 was not learned well by the classifier.

> - M_61=1 implies that the model does confuse samples originally belonging to class-6 with class-1, i.e., the classification boundary between classes 6 and 1 was not learned well by the classifier.

18.5.2. Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

In [ ]:
```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test,y_pred_rf)
r2 = metrics.r2_score(y_test, y_pred_rf)
mae = metrics.mean_absolute_error(y_test,y_pred_rf)
rmse = metrics.mean_squared_error(y_test,y_pred_rf, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

- MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

- R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

- MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

- RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

18.5.3.Accuracy Score

- It will print the accuracy scores for training and testing sets using RandomForestClassifier.

In [ ]:
```
accuracy_rf = accuracy_score(y_test, y_pred_rf)
accuracy_rf_train = accuracy_score(y_train, y_pred_rf_train)
print(f'\033[1;31m  \033[1m  accuracy of RandomForestClassifier for tes
print(f'\033[1;31m  \033[1m  accuracy of RandomForestClassifier for  tr
```

18.5.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test, y_pred_rf, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of RandomForestClassifier:
```

18.5.5. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test, y_pred_rf, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of RandomForestClassifier:',f1_sc
```

18.5.6. Recall Score

- It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test, y_pred_rf, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of RandomForestClassifier:',r
```

18.6 Classification Report

In [ ]:
```
print(classification_report(y_test, y_pred_rf, target_names=
                            ['psoriasis','seboreic_dermatitis','lichen_
                             'pityriasis_rubra_pilaris']))
```

Random Forest Classifier Report

Class-specific Analysis:

1. **Psoriasis:**
   - High precision (0.97) and perfect recall and F1-Score (both 1.00), indicating accurate and reliable predictions for psoriasis.
2. **Seboreic Dermatitis:**

- High precision (0.92) and perfect recall and F1-Score (both 1.00), showcasing strong performance.

3. **Lichen Planus:**
   - Perfect precision, recall, and F1-Score (all 1.00), demonstrating accurate predictions for lichen planus.
4. **Pityriasis Rosea:**
   - High precision (1.00) and recall (0.91), resulting in a balanced F1-Score of 0.95.
5. **Cronic Dermatitis:**
   - Perfect precision, recall, and F1-Score (all 1.00), indicating excellent performance for cronic dermatitis.
6. **Pityriasis Rubra Pilaris:**
   - High precision (1.00) but lower recall (0.75) and F1-Score (0.86), suggesting some room for improvement.

Model Evaluation Metrics:

1. **Accuracy:**
   - Overall accuracy of 98%, indicating a highly reliable model.
2. **Macro Average:**
   - Macro-average precision, recall, and F1-Score are 0.98, 0.94, and 0.96, respectively, showcasing consistent performance across classes.
3. **Weighted Average:**
   - Weighted-average precision, recall, and F1-Score are 0.98, 0.98, and 0.98, respectively, reflecting robust performance considering class imbalances.

Conclusion:

The Random Forest Classifier proves to be a highly effective and accurate model for predicting skin diseases. With balanced precision and recall scores across various classes, it can serve as a reliable tool in a dermatological diagnostic setting. Further investigation into feature importance can provide insights into the key factors influencing the model's predictions.

In [ ]:
```
 
```

In [ ]:
```
 
```

19 Gradient Boosting Classifier

Model Building

In [ ]:
```python
# Importing GradientBoostingClassifier from 'sklearn.ensemble'.
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
```

19.1. Model creation

For creating a Gradient Boosting Classifier model, we have performed hyperparameter tuning for a Gradient Boosting Classifier using Grid Search within a pipeline.

1. Import Libraries:

> - Import necessary libraries, including GradientBoostingClassifier, GridSearchCV, StandardScaler, and Pipeline from scikit-learn.

2. Pipeline Creation:

> - Create a pipeline (pipeline_GBC) with two steps:
>   - Step 1 ('scaler'): Standardize the features using StandardScaler.
>   - Step 2 ('classifier'): Apply the Gradient Boosting Classifier using GradientBoostingClassifier().

3. Hyperparameter Grid:

> - Define a hyperparameter grid (param_GBC) for the Gradient Boosting Classifier, including options for the number of estimators, learning rate, and maximum depth.

4.GridSearchCV:

> - Create an instance of the GridSearchCV class (grid_search_GBC) to perform a search over the hyperparameter grid within the defined pipeline.
> - Set the estimator to the pipeline (pipeline_GBC), the hyperparameter grid to param_GBC, use 5-fold cross-validation (cv=5), and choose accuracy as the scoring metric.

5. Model Training:

> - Fit the grid search on the training data (x_train, y_train).

6. Print Best Results:

- Print the best score achieved by the model and the corresponding best hyperparameters.

In [ ]:
```
# Create a pipeline with StandardScaler and GradientBoostingClassifier
pipeline_GBC = Pipeline([
    ('scaler', StandardScaler()),  # You can customize preprocessing st
    ('classifier', GradientBoostingClassifier())  # Step 2: GradientBoo
])

# Define the parameter grid for GridSearchCV
param_GBC = {
    'classifier__n_estimators': [50, 100, 200],  # Number of boosting s
    'classifier__learning_rate': [0.01, 0.1, 0.2],  # Step size shrinka
    'classifier__max_depth': [3, 4, 5]  # Maximum depth of the individu
}

# Create GridSearchCV object
grid_search_GBC = GridSearchCV(estimator=pipeline_GBC,
                               param_grid=param_GBC,
                               cv=5,
                               scoring='accuracy')
grid_search_GBC.fit(x_train, y_train)
print(f" Best score is: {grid_search_GBC.best_score_} with parameters:

```

19.2. best parameters and best estimator

In [ ]:
```
# Get the best parameters and best estimator
best_params =grid_search_GBC.best_params_
best_estimator = grid_search_GBC.best_estimator_
print(f'\033[1;31m  \033[1m  best hyperparameter for GradientBoostingCl
print()
print(f'\033[1;31m  \033[1m  best estimator for GradientBoostingClassif

```

19.3 Fit the grid search on the training data

In [ ]:
```

# Fit the grid search on the training data
grid_search_GBC.fit(x_train, y_train)
```

19.4. Make Predictions/ Test model

In [ ]:
```
# Make predictions on the test data using the best estimator
y_pred_GBC = best_estimator.predict(x_test)
# The predicted labels are stored in the variable 'y_pred'.
y_pred_GBC
```

In [ ]:
```
# Making predictions on the training data using the trained GradientBoo
y_pred_GBC_train = best_estimator.predict(x_train)
y_pred_GBC_train
```

19.5 Evaluating the Model

19.5.1. Confusion Matrix

- Creating a ConfusionMatrixDisplay object and visualize the confusion matrix by providing the true labels and predicted labels .

In [ ]:
```
# Plot non-normalized confusion matrix
titles_options = [
    ("Confusion matrix, without normalization", None),
    ("Normalized confusion matrix", "true"),
]
for title, normalize in titles_options:
    disp = ConfusionMatrixDisplay.from_estimator(
        grid_search_GBC.fit(x_train, y_train),
        x_test,
        y_test,
        display_labels=['1', '2', '3', '4','5','6'],
        cmap=plt.cm.Greens,
        normalize=normalize,
    )
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

plt.show()
```

Confusion Matrix Report Interpretation:

1. **Psoriasis (Class 1):**
   - True Positives (TP): 34
   - No False Positives (FP), False Negatives (FN), or True Negatives (TN).
2. **Seboreic Dermatitis (Class 2):**
   - TP: 12
   - No FP, FN, or TN.
3. **Lichen Planus (Class 3):**
   - TP: 17
   - No FP, FN, or TN.
4. **Pityriasis Rosea (Class 4):**
   - TP: 10

- One False Positive (misclassified as Class 3), and no FN or TN.

5. **Cronic Dermatitis (Class 5):**

   - TP: 14
   - No FP, FN, or TN.

6. **Pityriasis Rubra Pilaris (Class 6):**

   - TP: 4
   - No FP, FN, or TN.

Plot the comparison report to show where the datapoint misclassified

In [ ]:

```python
# Plot the comparison between the actual and predicted bike rental cour
plt.figure(figsize=(10,6))
plt.scatter(y_test, y_pred_GBC, color='blue', label='Data points')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], co
plt.xlabel('Actual Skin Disorder')
plt.ylabel('Predicted Skin Disorder')
plt.title('Comparison between actual and predicted Skin Disorder')
plt.legend()
plt.show()
```

Insights of confusion_matrix

This confusion matrix gives a lot of information about the model's performance:

- As usual, the diagonal elements are the correctly predicted samples. A total of 91 samples were correctly predicted out of the total 92 samples. Thus, the overall accuracy is 98.91 %.

- M_42=1 implies that the model does confuse samples originally belonging to class-4 with class-2, i.e., the classification boundary between classes 4 and 2 was not learned well by the classifier.

19.5.2. Mean Squared Error , r2_score , Mean Absolute Error , Root Mean Squared Error

In [ ]:
```
# Evaluate the performance of the model
mse = metrics.mean_squared_error(y_test, y_pred_GBC)
r2 = metrics.r2_score(y_test,  y_pred_GBC)
mae = metrics.mean_absolute_error(y_test,  y_pred_GBC)
rmse = metrics.mean_squared_error(y_test,  y_pred_GBC, squared=False)



print(f'\033[1;31m  \033[1m  Mean Squared Error: {mse:.2f}')
print(f'\033[1;31m  \033[1m  R-squared Score: {r2:.2f}')
print(f'\033[1;31m  \033[1m  Mean Absolute Error: {mae:.2f}')
print("\033[1;31m  \033[1m  Root Mean Squared Error:", rmse)
```

observation :-

- MSE: The closer the MSE is to zero, the better the model's predictions align with the actual values.

- R-squared Score (R2): A higher R2 indicates that a larger proportion of the variance in the target variable is explained by the model.

- MAE: Similar to MSE, a lower MAE indicates better performance. It provides a measure of average prediction error.

- RMSE: Like MSE, lower RMSE values indicate better model performance. It is in the same scale as the target variable.

19.5.3. Accuracy Score

- It will print the accuracy scores for training and testing sets using GradientBoostingClassifier.

In [ ]:
```
accuracy_GBC = accuracy_score(y_test, y_pred_GBC)
accuracy_GBC_train = accuracy_score(y_train, y_pred_GBC_train)
print(f'\033[1;31m  \033[1m  accuracy of GradientBoostingClassifier for
print(f'\033[1;31m  \033[1m  accuracy of GradientBoostingClassifier for
```

19.5.4. Precision Score

- It will calculate the weighted precision score for the model(testing set).

In [ ]:
```
precision=precision_score(y_test, y_pred_GBC, average='weighted')
print(f'\033[1;31m  \033[1m  precision_score of GradientBoostingClassif
```

19.5.5. Recall Score

- It will calculate the weighted recall score for the model (testing set).

In [ ]:
```
recall=recall_score(y_test, y_pred_GBC, average = 'weighted')
print(f'\033[1;31m  \033[1m  recall_score of GradientBoostingClassifier
```

19.5.6. f1 score

- It will calculate the weighted f1 score for the testing set.

In [ ]:
```
from sklearn.metrics import f1_score
f1_score=f1_score(y_test, y_pred_GBC, average = 'weighted')
print(f'\033[1;31m  \033[1m  f1_score of GradientBoostingClassifier:',f
```

19.6. Classification Report

In [ ]:
```
print(classification_report(y_test, y_pred_GBC, target_names=
                            ['psoriasis','seboreic_dermatitis','lichen_
                             'pityriasis_rubra_pilaris']))
```

Gradient Boosting Classifier Report

Class-specific Analysis:

1. **Psoriasis:**
   - High precision (1.00) and perfect recall and F1-Score (both 1.00), indicating accurate and reliable predictions for psoriasis.
2. **Seboreic Dermatitis:**

    - High precision (0.86) and perfect recall and F1-Score (both 1.00), showcasing strong performance.
3. **Lichen Planus:**
    - Perfect precision, recall, and F1-Score (all 1.00), demonstrating accurate predictions for lichen planus.
4. **Pityriasis Rosea:**
    - High precision (1.00) and recall (0.82), resulting in a balanced F1-Score of 0.90.
5. **Cronic Dermatitis:**
    - Perfect precision, recall, and F1-Score (all 1.00), indicating excellent performance for cronic dermatitis.
6. **Pityriasis Rubra Pilaris:**
    - Perfect precision, recall, and F1-Score (all 1.00), demonstrating accurate predictions.

Model Evaluation Metrics:

1. **Accuracy:**
    - Overall accuracy of 98%, indicating a highly reliable model.
2. **Macro Average:**
    - Macro-average precision, recall, and F1-Score are 0.98, 0.97, and 0.97, respectively, showcasing consistent performance across classes.
3. **Weighted Average:**
    - Weighted-average precision, recall, and F1-Score are 0.98, 0.98, and 0.98, respectively, reflecting robust performance considering class imbalances.

Conclusion:

The Gradient Boosting Classifier proves to be a highly effective and accurate model for predicting skin diseases. With balanced precision and recall scores across various classes, it can serve as a reliable tool in a dermatological diagnostic setting. Further investigation into feature importance can provide insights into the key factors influencing the model's predictions.

```
In [ ]:
```

20. Model Comparison Report

We are going to creat a DataFrame for Model_Comparison to compare the train and test scores of different machine learning models. The train and test scores represent the accuracy of each model on the training and test datasets.

In [ ]:
```
# Creating a DataFrame with model names, train scores, and test scores.
Model_Comparison = pd.DataFrame({'Model':['Logistic Regressor ()',
                                          'KNN Classifier ()',
                                          'Support Vector Classifier(Tur
                                          'Decision Tree Classifier()',
                                          'Random Forest Classifier',
                                          'Gradient Boosting Classifier(

                                          ],

                                 'Train Score':[accuracy_LR_train,
                                                accuracy_knn_train,
                                                accuracy_svm_train,
                                                accuracy_decsT_train,
                                                accuracy_rf_train,
                                                accuracy_GBC_train,
                                                ],

                                 'Test Score':[accuracy_LR,
                                               accuracy_knn,
                                               accuracy_svm,
                                               accuracy_decsT,
                                               accuracy_rf,
                                               accuracy_GBC,
                                               ]})
# Adjusting the index to start from 1.
Model_Comparison.index = Model_Comparison.index + 1

# It will display the DataFrame.
Model_Comparison
```

This comparison table provides a quick overview of the relative performance of different machine learning models, helping to identify best model for further analysis or deployment based on their accuracy scores on both training and test data.

Analysis:

- Models with a train score close to 1.0 indicate strong performance on the training data.

- Decision Tree Classifier achieved a high accuracy on both the training and test sets, suggesting good generalization.

- Random Forest Classifier, Support Vector Classifier (Tuned), and Gradient Boosting Classifier also show high train and test scores.

21. Model Evaluation Metrics: Jaccard Score and F1-Score

Jaccard Score and F1-Score are calculated for the evaluation of various machine learning models

Jaccard Score:

> - The Jaccard Score is a measure of the similarity between the predicted and true labels, with a range from 0 to 1. A higher score indicates better similarity.

F1-Score:

> - The F1-Score is the harmonic mean of precision and recall, ranging from 0 to 1. A higher F1-Score indicates better balance between precision and recall.

In [ ]:

```
from sklearn.metrics import jaccard_score
from sklearn.metrics import f1_score
from sklearn.metrics import log_loss

# Evaluate Logistic Regression
jc1 = (jaccard_score(y_test, y_pred_LR, average='weighted'))
fs1 = (f1_score(y_test, y_pred_LR, average='weighted'))



# Evaluate KNN
jc2 = (jaccard_score(y_test, y_pred_knn, average='weighted'))
fs2 = (f1_score(y_test, y_pred_knn,average='weighted'))

# Evaluate SVM
jc3 = (jaccard_score(y_test, y_pred_svm, average='weighted'))
fs3 = (f1_score(y_test, y_pred_svm, average='weighted'))

# Evaluate Decision Trees
jc4 = (jaccard_score(y_test, y_pred_decsT, average='weighted'))
fs4 = (f1_score(y_test, y_pred_decsT, average='weighted'))


# Evaluate Random Forest Classifier
jc5 = (jaccard_score(y_test, y_pred_rf, average='weighted'))
fs5 = (f1_score(y_test, y_pred_rf, average='weighted'))

# Evaluate Gradient Boosting Classifier
jc6 = (jaccard_score(y_test, y_pred_GBC, average='weighted'))
fs6 = (f1_score(y_test, y_pred_GBC, average='weighted'))



list_jc = [jc1, jc2, jc3, jc4 ,jc5 ,jc6]
list_fs = [fs1, fs2, fs3, fs4 ,fs5 ,fs6]

```

In [ ]:
```
# fomulate the report format
report = pd.DataFrame( list_jc,index=['Logistic Regression','KNN','SVM
report.columns = ['Jaccard']
report.insert(loc=1, column='F1-score', value=list_fs)
#report.insert(loc=2, column='LogLoss', value=list_ll)
report.columns.name = 'Algorithm'
report
```

Analysis:

- Decision Tree achieved the highest Jaccard Score among the models.

- Decision Tree also achieved the highest F1-Score among the models.

- Decision Tree outperformed other models in terms of both Jaccard Score and F1-Score.

- KNN showed slightly lower performance compared to other models.

- SVM, Logistic Regression, Random Forest Classifier, and Gradient Boosting Classifier had similar Jaccard and F1-Score values.

Note (what we can do next) : -

- The Decision Tree model appears to be the most effective based on the provided evaluation metrics. Consider exploring its interpretability and feature importance for further insights.

- Check for potential overfitting or underfitting in models with perfect Jaccard Scores, especially Decision Tree, Random Forest, and Gradient Boosting.

- Experiment with adjusting hyperparameters or exploring other algorithms to improve model performance, especially if there is a need for better generalization.

In [ ]:
```

```

22. Grid Search Time Analysis

The time analysis of the grid search process for different machine learning models

Time for Fit:

> - Represents the total time taken for fitting the model during grid search.

Time for Score:

> - Represents the total time taken for scoring the model during grid search.

Time per k-fold:

> - Represents the average time taken per k-fold during grid search.

Total Time:

> - Represents the total time for both fit and score over the entire grid search process.

In [ ]:
```python
# Calculate the total time for fitting the models during the grid searc
# it sums up the mean fit times obtained from the cross-validation resu

grid_search_LR.fit = grid_search_LR.cv_results_['mean_fit_time'].sum()
grid_search_knn.fit = grid_search_knn.cv_results_['mean_fit_time'].sum(
grid_svm_fit = grid_search_svm.cv_results_['mean_fit_time'].sum()
grid_decsT_fit = grid_search_decsT.cv_results_['mean_fit_time'].sum()
grid_rf_fit = grid_search_rf.cv_results_['mean_fit_time'].sum()
grid_GBC_fit = grid_search_GBC.cv_results_['mean_fit_time'].sum()

#Calculate the total time for scoring the models during the grid search
# It sums up the mean score times obtained from the cross-validation re

lr_score = grid_search_LR.cv_results_['mean_score_time'].sum()
knn_score = grid_search_knn.cv_results_['mean_score_time'].sum()
svm_score = grid_search_svm.cv_results_['mean_score_time'].sum()
decsT_score = grid_search_decsT.cv_results_['mean_score_time'].sum()
rf_score = grid_search_rf.cv_results_['mean_score_time'].sum()
GBC_score = grid_search_GBC.cv_results_['mean_score_time'].sum()

#Calculate the total time for both fit and score for each model.

time_lr = grid_search_LR.fit + lr_score
time_knn = grid_search_knn.fit + knn_score
time_svm = grid_svm_fit + svm_score
time_decsT = grid_decsT_fit + decsT_score
time_rf =grid_rf_fit + rf_score
time_GBC = grid_GBC_fit + GBC_score

#time_fit: List containing the total time for fitting each model.
#time_score: List containing the total time for scoring each model.
#total_time: List containing the total time for both fit and score mult

time_fit = [grid_search_LR.fit, grid_search_knn.fit , grid_svm_fit ,  g
time_score = [lr_score, knn_score, svm_score, decsT_score, rf_score, GB
time_fit = [time_lr, time_knn, time_svm, time_decsT, time_rf, time_GBC]
total_time = [5*x for x in time_fit]

#time_report: DataFrame summarizing the fitting and scoring times for e
time_report = pd.DataFrame( time_fit,index=['Logistic Regression','KNN'
time_report.columns = ['Time for fit']
time_report.insert(loc=1, column='Time for score', value=time_score)
time_report.insert(loc=2, column='Time per k-fold', value=time_fit)
time_report.insert(loc=3, column='Total time', value=total_time)
time_report.columns.name = 'Algorithm'
time_report.style.format("{:.2f}")#The code uses style.format("{:.2f}")


```

Analysis:

- Logistic Regression and Decision Tree models show relatively lower times, indicating faster grid search.

- Gradient Boosting Classifier takes significantly more time compared to other models, suggesting higher computational requirements.

- Time per k-fold is proportional to the complexity of the model, with more complex models taking longer.

Note (what we can do next) : -

- Consider the trade-off between model complexity and computational resources. If the time is a critical factor, simpler models like Logistic Regression or Decision Trees might be preferred.

- Evaluate the importance of hyperparameter tuning. If the grid search time is significantly impacting overall workflow efficiency, explore strategies such as reducing the hyperparameter search space or optimizing the code for parallel processing.

- For models with long grid search times, consider using distributed computing or parallel processing to speed up the tuning process.

In [ ]:
```

```

23. plot comparing the train and test scores of different classification models.

In [ ]:
```
# Extracting model names, train scores, and test scores from the DataFr
model_names = Model_Comparison['Model'].tolist()
train_scores = Model_Comparison['Train Score'].tolist()
test_scores = Model_Comparison['Test Score'].tolist()

# Here, We are setting up the size of the plot.
plt.figure(figsize=(12,6))

# Plotting the train scores for each model with blue color.
plt.plot(model_names, train_scores, label='Train Score', color='#0590DA

# Plotting the test scores for each model with orange color.
plt.plot(model_names, test_scores, label='Test Score', color='#FF6600')

# Adding a legend to the plot.
plt.legend()

# Rotating x-axis labels for better readability.
plt.xticks(rotation = 90)

# Adding labels to the x and y axes.
plt.xlabel('Model')
plt.ylabel('Score')

# Setting y-axis limits for better visualization.
plt.ylim(0.900, 1.05)


# Adding a title to the plot.
plt.title('Comparison between Train and Test Scores of each Classificat

# Displaying the plot.
plt.show()
```

Analysis:

- The plot allows visualizing and comparing the performance of different models on both training and test datasets.

- The height of each bar indicates the corresponding score for each model.

- Models with similar train and test scores are likely to generalize well to unseen data.

- Discrepancies between train and test scores can provide insights into potential overfitting or underfitting.

24. Creating a bar plot to visualize Accuracy scores for different algorithms

In [ ]:
```python
# Creating a list of Accuracy score for different algorithms
scores = [accuracy_LR, accuracy_knn, accuracy_svm, accuracy_decsT, accu

# List of algorithm names.
algorithms = ["Logistic Regression", "KNN Classifier", "Support Vector

# Looping through each algorithm and printing its Accuracy Score.
for i in range(len(algorithms)) :

    print("The accuracy score achieved using "+ algorithms[i]+" is: "+s

```

In [ ]:
```python
# Here, We are setting the figure size of the plot.
plt.figure(figsize=(10,6))

# Adding labels to the axes.
plt.xlabel("Algorithms", fontsize = 15)
plt.ylabel("Accuracy Score", fontsize = 15)

# Using seaborn to creat a bar plot.
ax = sns.barplot(x = algorithms, y = scores)

# Adding labels to the each bar in the plot.
for label in ax.containers:
    ax.bar_label(label)

# Ensuring tight layout for a clean presentation.
plt.tight_layout()

# it will rotate the x-tick labels by 90 degree for better readability.
plt.xticks(rotation = 90)

# Adjusting tick label font size for better readability.
plt.tick_params(labelsize = 14)

# It will display the plot.
plt.show()
```

In [ ]:
```python

```

25. Summary Report on Skin Disease Classification Models:

1. Logistic Regression:

- **Overall Performance:**
  - Accuracy: 98%

- **Class-wise Evaluation:**
  - High precision, recall, and F1-score for all classes.
  - Particularly effective for Psoriasis, Lichen Planus, and Chronic Dermatitis.

2. KNN (K Nearest Neighbors) Classifier:

- **Overall Performance:**
  - Accuracy: 97%

- **Class-wise Evaluation:**
  - High precision, recall, and F1-score for most classes.
  - Slightly lower performance for Seborrheic Dermatitis and Pityriasis Rosea.

- **Doctor's Insight:**
  - Robust performance but potential improvements in specific conditions.
  - Considerable accuracy for diverse cases.

3. Support Vector Classifier:

- **Overall Performance:**
  - Accuracy: 98%

- **Class-wise Evaluation:**
  - Similar to Logistic Regression, high scores for all classes.

- **Doctor's Insight:**
  - Consistent with Logistic Regression.
  - Suitable for diverse skin disease identification.

4. Decision Tree:

- **Overall Performance:**
  - Accuracy: 99%

- **Class-wise Evaluation:**
  - Outstanding precision, recall, and F1-score for most classes.
  - Slightly lower recall for Pityriasis Rubra Pilaris.

- **Doctor's Insight:**
  - Excellent overall performance, especially for common skin conditions.
  - Potential improvement in rare conditions.

5. Random Forest Classifier:

- **Overall Performance:**
  - Accuracy: 98%

- **Class-wise Evaluation:**
  - High precision, recall, and F1-score, similar to Logistic Regression.

- **Doctor's Insight:**
  - Comparable to Logistic Regression.
  - Reliable choice for skin disease classification.

6. Gradient Boosting Classifier:

- **Overall Performance:**
  - Accuracy: 98%

- **Class-wise Evaluation:**
  - Similar to Logistic Regression and Support Vector Classifier.

- **Doctor's Insight:**
  - Consistent and reliable, similar to other models.
  - Effective for a diverse range of skin conditions.

26. Conclusion:

- All models demonstrate high accuracy, indicating their effectiveness in skin disease classification.

- Decision Tree stands out with exceptional overall performance.

- Logistic Regression, Support Vector Classifier, and Random Forest Classifier show consistent and reliable results.

> - KNN and Gradient Boosting Classifier exhibit slightly lower accuracy but remain suitable for diverse cases.

27. Challenges:

Class Imbalance:

> - The dataset might have imbalanced class distributions, impacting the model's performance on minority classes.
> - Continuous monitoring and model updates are essential for evolving skin disease patterns.

Specificity for Rare Classes:

> - Achieving high performance on rare classes like 'Pityriasis Rubra Pilaris' may be challenging due to limited data.
> - Potential improvements for specific skin conditions, especially in rare cases.

Interpretability:

> - While metrics provide insights, understanding the reasons behind model predictions, especially for misclassifications, is crucial for practical application.
> - Interpretability of Decision Tree might be challenging for doctors.

These models collectively offer doctors valuable tools for early identification and classification of various skin diseases, enhancing the efficiency of diagnosis and treatment planning. The choice of a specific model can be tailored based on the healthcare setting and the specific skin conditions prevalent in the population.

In [ ]:
```

```

In [ ]:
```

```