Multi-class classification using Dermatology Data

GROUP 04

S14853 – PRAMUDI RAJAMANTHRI

S15030 – VIDURA CHATHURANGA

S15091 – CHALANI WIJAMUNIGE

ST 4052
Data Analysis Project 2

Abstract

This report outlines the findings of a comprehensive descriptive analysis which was carried on a dataset obtained from Kaggle, which focuses on the challenging field of dermatology. The dataset pertains to "Erythemato-squamous" diseases, which comprises of common clinical features of erythema and scaling, with subtle distinctions, making their differential diagnosis challenging. Evaluation involved 12 clinical features and 22 histopathological features. Furthermore, an advanced analysis was carried out on this dataset aiming to unveil insights into these diseases, potentially assisting dermatologists in refining their differentiation and advancing diagnostic accuracy for better patient care. This report will present the important findings of the descriptive and advanced analysis performed on the dataset, which shows the associations and relationships among variables and special characteristics of certain variables in differentiating Erythemato-squamous diseases, and finally extracts a suitable predictive model.

Contents

List of Figures

List of Tables

1. Introduction

Erythemato-squamous diseases are chronic diseases that negatively affect patients' mental and social quality of life as well as cause economic negativities with high treatment costs with high-cost drugs obtained from foreign countries. The differential diagnosis of Erythemato-squamous diseases presents a significant challenge in dermatology due to their shared clinical features of erythema and scaling, with minimal distinguishing characteristics. This group of disorders includes psoriasis, seborrheic dermatitis, lichen planus, pityriasis rosea, chronic dermatitis, and pityriasis rubra pilaris. Accurate diagnosis often requires a biopsy, but even histopathological analysis obtained by the results of the biopsy faces difficulties as these diseases exhibit many overlapping features such as, a disease may show histopathological features of another disease at the beginning stage and may have the characteristic features at the following stages. In this data analysis project, we aim to address this diagnostic challenge by utilizing a dataset obtained from Kaggle. The dataset comprises 12 clinical and 22 histopathological features, including age and family history with various degrees of presence for each feature along with the response variable *class* which includes the above mentioned 6 erythemato-squamous disease categories. By employing a multi-class classification approach, a predictive model will be developed to aid in the differential diagnosis of these Erythemato-squamous diseases.

2. Description of the problem

In this project, we seek to address the challenging task of differentiating the features between erythemato-squamous disease categories in dermatology. Our main objectives are to analyze the characteristics of the dataset, identifying any patterns or correlations among features, assess the impact of age on the occurrence of these features, narrow down the most influential features in differential diagnosis of these diseases and include those in our classification model assessment. Finally, a predictive model will be developed that can improve the differential diagnostic accuracy of Erythemato-squamous disease and ultimately enhance patient care in dermatology.

3. Description of the data set

The Dermatology Dataset consists of 366 records, each with 35 attributes. Among these attributes, 32 are ordinal, while the *family_history* variable is nominal. The *age* variable is the only continuous variable, where the response variable *class* is a categorical variable with 6 levels, indicating the type of Erythemato-squamous disease. Initially patients have been first examined with 12 clinical features, after which the assessment of 22 histopathological attributes was performed using skin disease samples. Histopathological features have been identified by analyzing the samples under a microscope. If any diseases were found in the family, the family history attribute in the dataset constructed for the domain has a value of 1 (one), and if not, the value is 0 (zero). All other ordinal attributes (clinical and histopathological both) were assigned a value in the range from 0 to 3 (0 = absence of features; 1, 2 = comparative intermediate values; 3 = highest value).

Dataset: Dermatology Dataset (Multi-class classification) | Kaggle

Variable Name	Variable Type	Clinical / Histopathological	Description
erythema	Qualitative	Clinical	skin redness
scaling	Qualitative	Clinical	scaly skin.
definite_borders	Qualitative	Clinical	clear sharp border separating it from its surroundings.
itching	Qualitative	Clinical	unpleasant sensation on the skin that provokes the desire to scratch the area.
koebner_phenomenon	Qualitative	Clinical	refers to when people with a specific dermatological disease manifest disease lesion in other skin lesions
polygonal_papules	Qualitative	Clinical	presence of shiny, flat-topped and firm on palpation circumscribed elevations.
follicular_papules	Qualitative	Clinical	presence of skin lesion, less than one centimeter in diameter, circumscribed, elevated, with well-defined borders and solid content
oral_mucosal_involvement	Qualitative	Clinical	presence of skin lesions inside the mouth.
knee_and_elbow_involvement	Qualitative	Clinical	skin lesions in the knee and/or the elbow
scalp_involvement	Qualitative	Clinical	skin lesions in the scalp
family_history	Qualitative	Clinical	whether there is a family history of similar dermatological conditions
age	Quantitative	Clinical	age of the patient in years
melanin_incontinence	Qualitative	Histopathological	spillage of melanin from basal keratinocytes into underlying connective tissue.
eosinophils_infiltrate	Qualitative	Histopathological	bone marrow-derived cells that infiltrate skin and mucous membrane.
PNL_infiltrate	Qualitative	Histopathological	pure neuritic leprosy, no skin lesions but larger nerve trunks or their branches are enlarged accompanied with a sensory loss in the areas
fibrosis_papillary_dermis	Qualitative	Histopathological	excess development of fibrous connective tissue in the papillary dermis
exocytosis	Qualitative	Histopathological	passage to the epidermis of cells foreign to it
acanthosis	Qualitative	Histopathological	Presence of dark, velvety skin areas in body creases
hyperkeratosis	Qualitative	Histopathological	thickening of the outer layer of the skin
parakeratosis	Qualitative	Histopathological	a mode of keratinization characterized by the retention of nuclei in the stratum corneum
clubbing_rete_ridges	Qualitative	Histopathological	the epithelial extensions that project into the underlying connective tissue in both skin and mucous membranes
elongation_rete_ridges	Qualitative	Histopathological	hyperpigmentation of the basal layer in the papillary dermis
thinning_suprapapillary_epidermis	Qualitative	Histopathological	a thinning of the granular layer at the tips of the papillae
spongiform_pustule	Qualitative	Histopathological	an epidermal pustule formed by infiltration of neutrophils into necrotic epidermis in pustular psoriasis
munro_microabcess	Qualitative	Histopathological	is an abscess in the stratum corneum of the epidermis due to the infiltration of neutrophils from papillary dermis into the epidermal stratum corneum
focal_hypergranulosis	Qualitative	Histopathological	is an increased thickness of the stratum granulosum
disappearance_granular_layer	Qualitative	Histopathological	disappearance of the skin granular layer
vacuolization_damage_basal_layer	Qualitative	Histopathological	presence of vacuolisation and damage of skin basal layer
spongiosis	Qualitative	Histopathological	presence of intercellular edema
saw_tooth_appearance_retes	Qualitative	Histopathological	appearance of saw tooth patterns under the skin tissue
follicular_horn_plug	Qualitative	Histopathological	presence of follicular horn plugs
perifollicular_parakeratosis	Qualitative	Histopathological	keratinization characterized by the retention of nuclei in tissues surrounding skin follicles
inflammatory_monoluclear_inflitrate	Qualitative	Histopathological	increase in the number of infiltrating mononuclear cells in the skin

band_like_infiltrate	Qualitative	Histopathological	basal epidermis in a banded pattern
class	Qualitative	Response	the type of Erythemato-squamous disease (6 different skin diseases)

Table 3.1

Here, class variable contains 6 different categories which are.

(1) **Psoriasis:** Chronic skin condition with red, scaly patches, sometimes affecting joints. May have psychiatric and bowel complications. Auspitz sign on removal of scales. Histopathology shows elongated rete ridges and lymphocytic infiltration.
(2) **Seborrheic Dermatitis:** Chronic inflammatory disease with oily, scaly patches on sebaceous-rich areas. Recurs with stress, depression, and fatigue. Histopathology shows epidermal spongiosis and inflammatory cell infiltration.
(3) **Lichen Planus:** Papulosquamous inflammatory disease affecting skin, nails, and mucous membranes. More common in women. Histopathology shows saw-tooth rete ridges and melanin incontinence.
(4) **Pityriasis Rosea:** Acute, self-limiting inflammatory disease with pink, scaly patches on trunk and extremities. Histopathology shows spongiosis and exocytosis.
(5) **Chronic Dermatitis (Eczema**): Recurrent, chronic inflammatory skin disease, often starting in childhood. Histopathology shows elongated rete ridges and hyperkeratosis.
(6) **Pityriasis Rubra Pilaris (PRP):** Rare inflammatory disease with unknown cause. Affects men and women, divided into five groups based on age and characteristics. Histopathology shows psoriasiform epidermis with parakeratosis and follicular infundibulum plugs.

4. Data pre-processing

- Originally, 33 predictor variables were represented as integers, and these variables were transformed into ordinal scale, allowing for ordered comparisons between their values.
- The "age" variable had 8 values labelled as '?', indicating missing data. The '?' symbols were replaced with NaN and imputation was performed based on the mean age of the training set.
- During our background research an unexpected outlier was detected. The koebner phenomenon, characterized by skin lesions forming at injury sites, exhibited an outlier value of "2" in the data for seborrheic dermatitis cases. Therefore, we filtered the dataset, excluding instances where both the "class" attribute (indicating seborrheic dermatitis) and the *koebner_phenomenon* attribute was equal to 2.
- Finally, the dataset was randomly split into training and test datasets which contains 80% and 20% of the entries from original dataset.

5. Important Results of the Descriptive Analysis

The pie chart (Figure 5.1) shows that class distribution in the training set is slightly unbalanced as the representation of the minority category; pityriasis rubra pilaris (6th category) is considerably lower than that of the other categories.

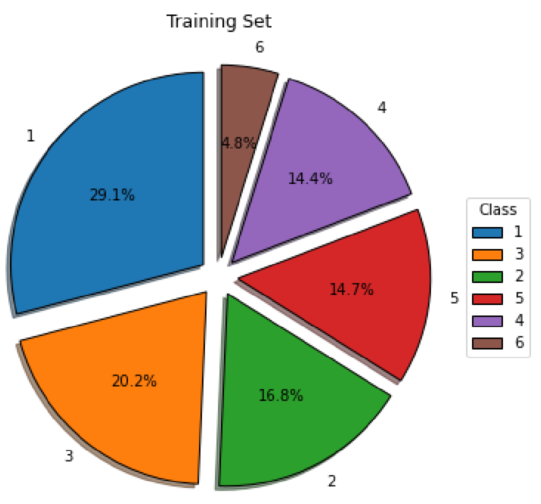


Figure 5.1

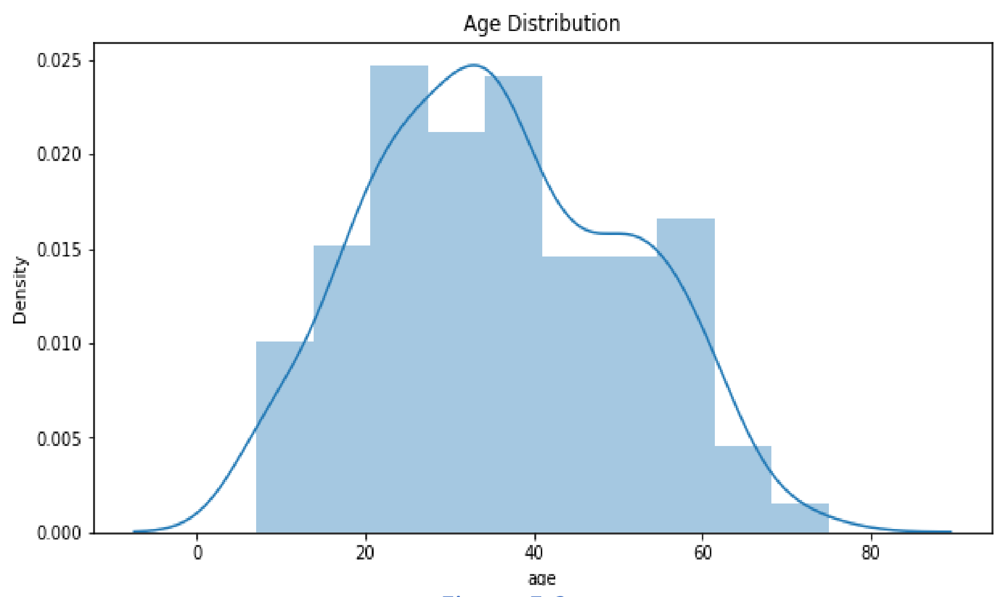


Figure 5.2

As we can observe from the plot in Figure 5.2, which is the *age* distribution, the majority of patients fall within 10-60 years range. It also exhibits an approximately symmetric bi-modal distribution, suggesting that in general 20-40 and 50-60 age groups might be more affected by erythemato-squamous diseases

Upon analyzing the box plots in, Figure 5.3 it is evident that the age distribution varies among the disease classes. For instance, disease class 6 (Pityriasis Rubra Pilaris (PRP)) exhibits a relatively young age profile, with most patients falling in the age range of 7 to 16. In contrast, other disease classes show a broader age range, indicating their potential occurrence across different age groups. Psoriasis disease class shows the broadest distribution in ages from 8 years to 75 years.

Understanding the age distribution allows dermatologists to consider the likelihood of specific diseases based on a patient's age, assisting in narrowing down the potential diagnoses and informing the diagnostic process.

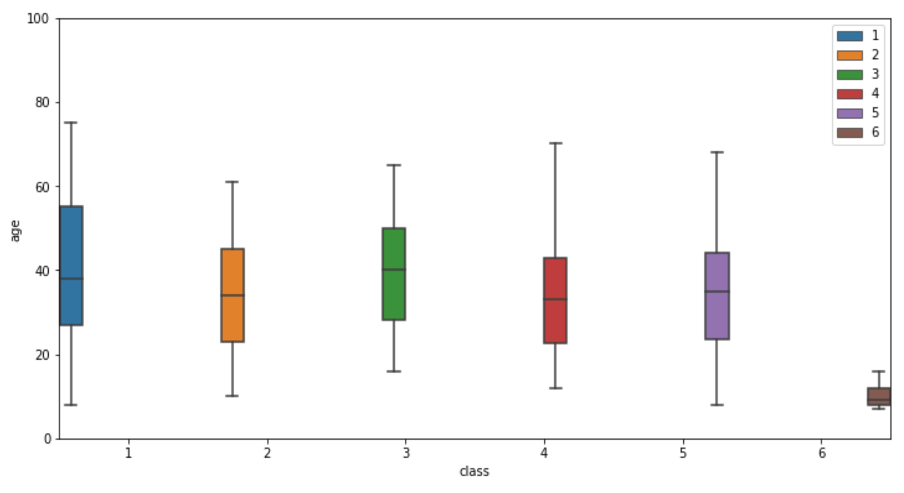


Figure 5.3

Afterwards, stacked bar plots for all clinical and histopathological features were plotted. The bars in each bar plot represents the 6 disease classes and each bar was separated (stacked) according to the observed levels of the considered clinical or histopathological feature.

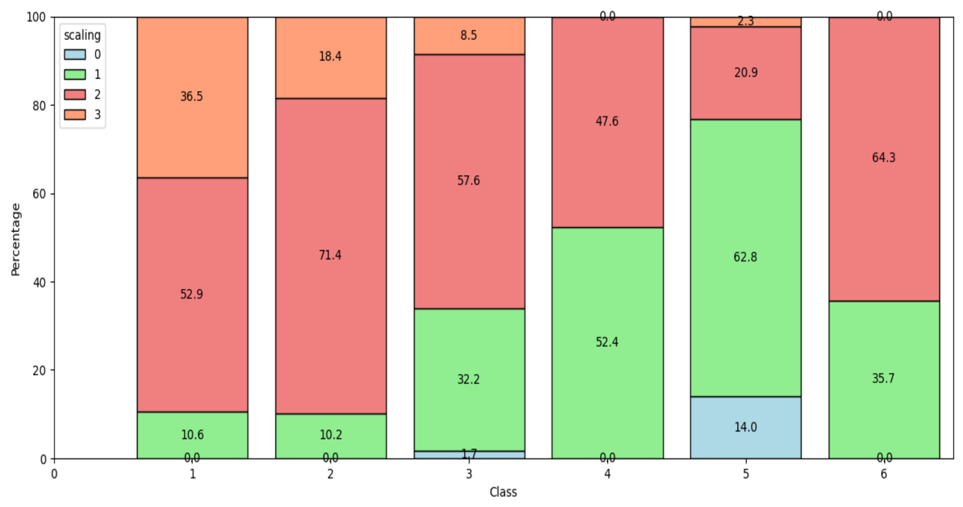


Figure 5.4

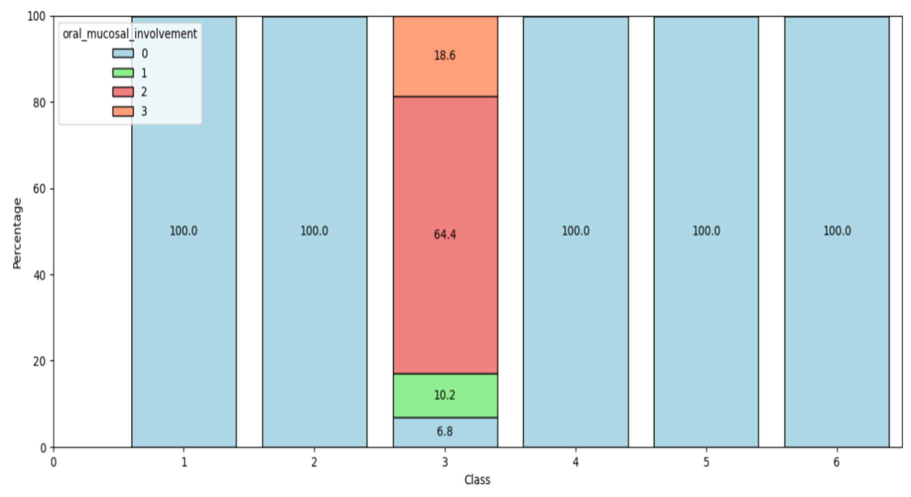


Figure 5.5

Some of the stacked bar plots showed observations on all levels dispersed in all disease classes without any significant association as shown in the Figure 5.4 and such bar plots carry the clinical and histopathological features which are less important in differential diagnosis of Erythmato – Squamous diseases. On the other hand, some stacked bar plots strongly suggested some significant and meaningful associations among clinical and histopathological features with different disease classes. For instance, the plot in Figure 5.5 shows that the clinical feature *oral_mucosal_involvement* is directly associated with the 3rd disease class (Lichen Planus) as the rest of the disease classes predominantly exhibits level 0. Findings obtained by such stacked bar plots are summarized in Table 5.1 given below.

class 1	*clubbing_rete_ridges, thinning_suprapapillary_epidermis, koeber_phenomenon, spongiform_pustule, elongation_rete_ridges, knee_and_elbow_involvement, munro_microabcess*
class 2	*PNL_infiltrate, exocytosis, spongiosis*
class 3	*koeber_phenomenon, polygonal_papules, oral_mucosal_involvement, vacuolisation_damage_basal_layer, saw_tooth_apperance_retes, focal_hypergranulosis, melanin_incontinence, band_like_infiltrate*
class 4	*koeber_phenomenon*
class 5	*fibrosis_papillary_dermis, elongation_rete_ridges, follicular_papules*
class 6	*follicular_horn_plug, follicular_papules, perifollicular_parakeratosis, knee_and_elbow_involvement*

Table 5.1

To get more insights on differential diagnosis, a Multiple Correspondence Analysis was conducted.

According to Figure 5.6 a total variability of 17.46% of the categorical predictor variables is addressed by the main two components of MCA, which further depicts how the encoded variables(columns) and the records(rows) are dispersed within the two components and how they are related to each other based on their relative locations within the plot.

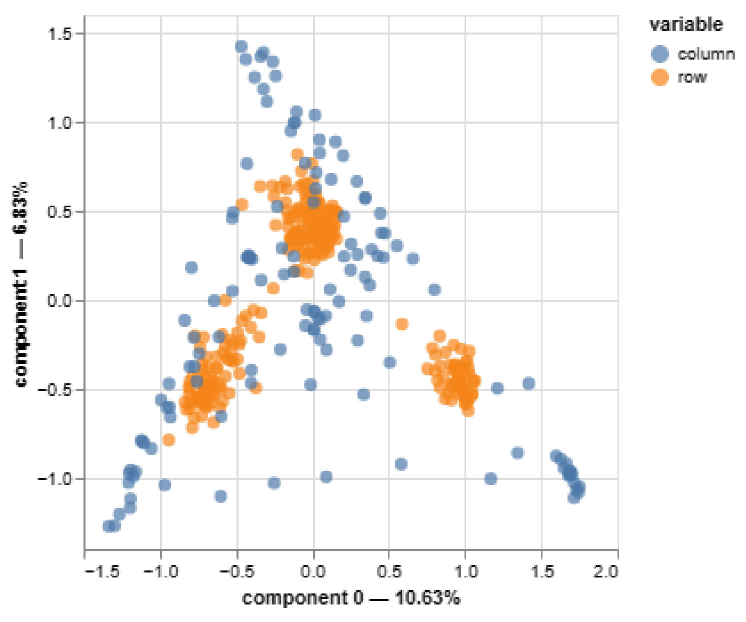


Figure 5.6

With the objective of investigating which clinical and histopathological features are prominent in differentiating the disease categories of the Erythemato-Squamous disease, the findings from MCA in Figure 5.7 provide more insightful facts clarifying the hidden picture of Figure 5.6 as the most critical features in the diagnosis are located closer to each disease class while the less critical ones are scattered around the plot.

Particularly, the clinical features are visible to be scattered around which further approves the results obtained from the bar charts above indicating their less prominence in the differential diagnosis of the disease.

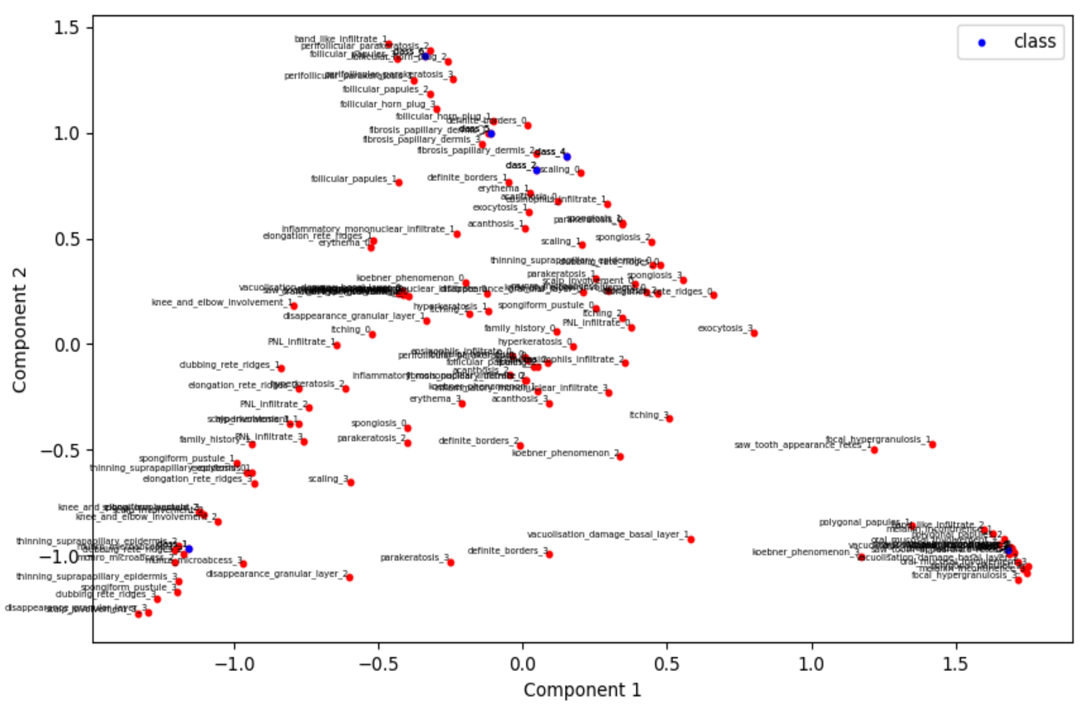


Figure 5.7

The specifically identified associations from the above MCA plot in Figure 5.7 can be summarized as follows.

class 1	*clubbing_rete_ridges, thinning_suprapapillary_epidermis, disappearance_granular_layer, spongiform_pustule*
class 2	*spongiosis, parakeratosis, exocytosis*
class 3	*koeber_phenomen, saw_tooth_apperance_retes, focal hypergranulosis, melanin incontinence*
class 4	*spongiosis, exocytosis, fibrosis_papillary_dermis*
class 5	*fibrosis_papillary_dermis, follicular_horn_plug*
class 6	*follicular_horn_plug, follicular_papules, parakeratosis*

Table 5.2

In order to check for clustering and variable importance, Partial Least Square – Discriminant Analysis was conducted.

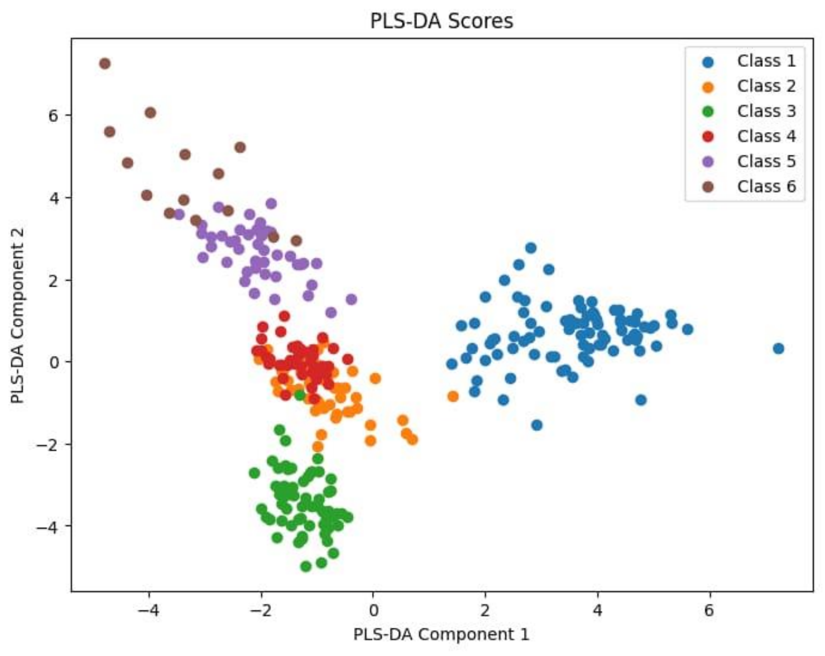


Figure 5.8

According to the score plot obtained in PLS – DA which is shown in Figure 5.8, 4 distinct clusters can be observed. Class 1 and Class 3 forms two separate clusters signifying the prominence of those two disease categories in our dataset. On the other hand, Class 5 and Class 6 collectively forms a single cluster as they overlap with each other, and this can be reasoned as these two disease classes have some significant features in common. Similar to this, Class 2 and Class 4 also collectively forms a single cluster due to its overlapping nature and the results obtained from bar plots and MCA show that those two classes also have many shared features.

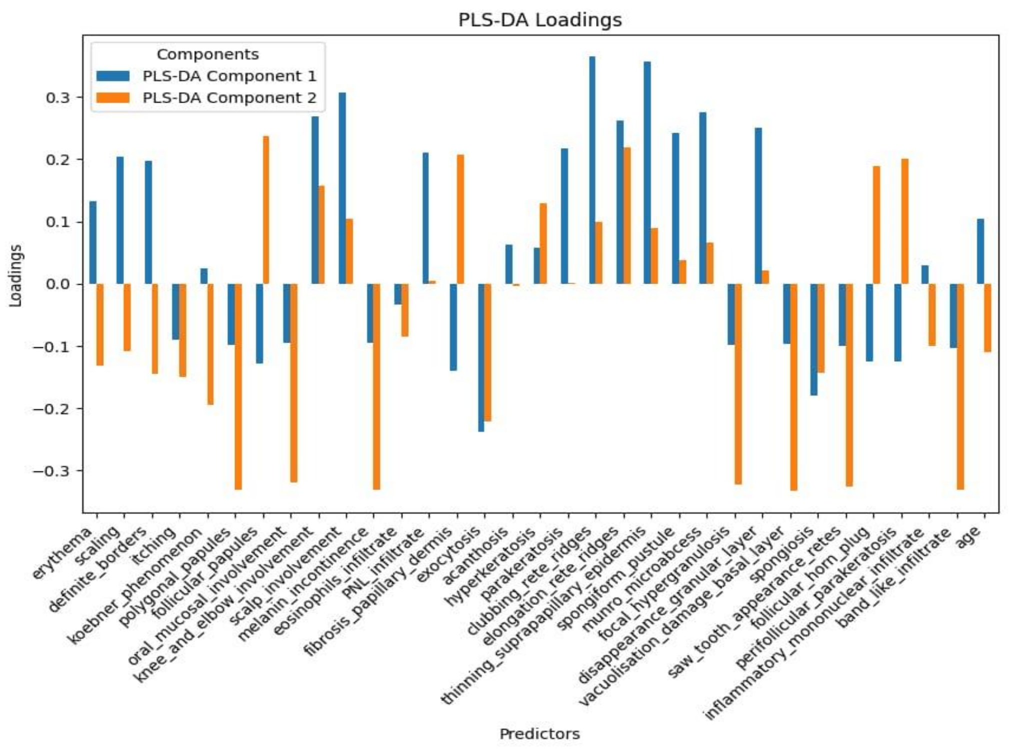


Figure 5.9

As the presence of clusters in the dataset was evident through PLS – DA, in the advanced analysis, predictive models will be fitted for the entire dataset and clusters separately in order to obtain better accuracy

By observing the loadings plot of PLS – DA given in Figure 5.9, the variables with the highest loadings, or in other words, the most important variables are very much the same variables that are mentioned in the Table 5.1 and Table 5.2.

6. Important Results of the Advanced Analysis

In the advanced analysis phase, our primary objective was to delve deeper into the classification problem inherent in dermatology data. Specifically, we aimed to evaluate the performance of a range of machine learning models in their capacity to accurately classify instances based on the “Class” variable. To build the foundation to this, the correlation among the variables were checked and a cluster analysis was conducted.

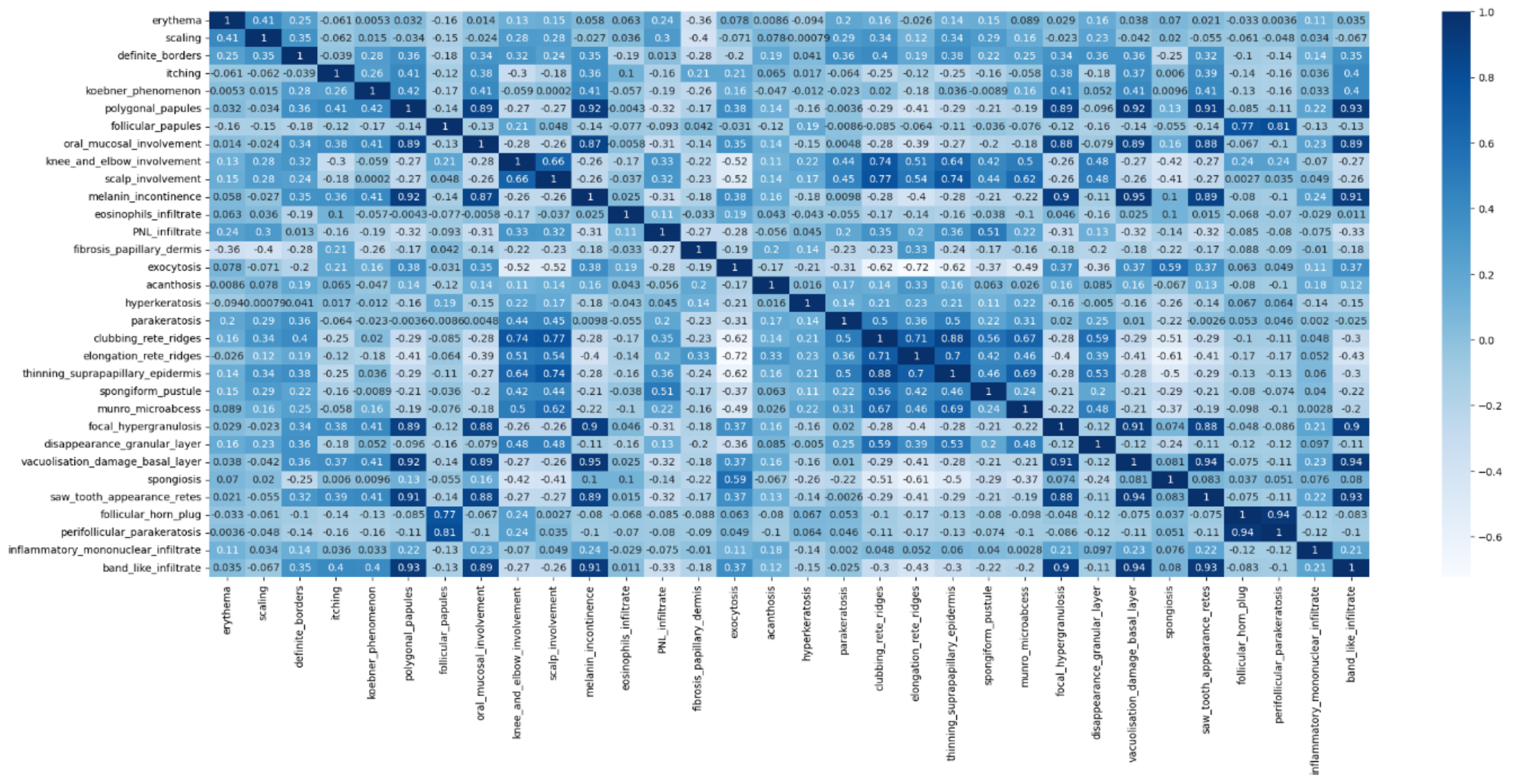


Figure 6.1

The spearman correlation plot depicts that there is considerable amount of significant positive and negative monotonic relationships among the ordinal variables.

Despite the above identified monotonic relationships, multinomial logistic regression model can be used as a base step in modelling the differential diagnosis of the dataset.

To initiate the analysis several predictive models including Logistic Regression, Logistic Ridge Regression, Logistic Lasso Regression, K–Nearest Neighbors (KNN), Multinomial Naïve Bayes, Random Forest and Support Vector Machine (SVM) with linear kernel were fitted. Also, since we observed our dataset to be imbalanced over the class distribution, the oversampling technique for mixed data 'SMOTE–NC' (Synthetic Minority Oversampling Technique for Nominal and Continuous features) was used to balance out the data distribution and enhance the accuracy. After applying SMOTE–NC, to check whether there are any unusual patterns observed in newly added data, a surface level descriptive analysis was conducted and no such anomalies were detected. The evaluation metrics of the initially fitted models applying the technique SMOTE–NC and without SMOTE–NC are as follows. Here the F1 score, precision and recall were calculated for all classes and the average was taken applying the technique 'Micro' when the dataset was imbalanced and the technique 'Macro' was applied when the dataset was balanced using 'SMOTE–NC'.

	Without SMOTE – NC													
	Training Set							Testing Set						
	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM
Accuracy	0.9863	0.9863	0.9897	0.9726	0.9863	1.0	1.0	0.9863	0.9828	0.9828	0.9863	0.9863	1.0	0.9863
F1-Score	0.9863	0.9863	0.9897	0.9726	0.9863	1.0	1.0	0.9863	0.9828	0.9828	0.9863	0.9863	1.0	0.9863
Precision	0.9863	0.9863	0.9897	0.9726	0.9863	1.0	1.0	0.9863	0.9828	0.9828	0.9863	0.9863	1.0	0.9863
Recall	0.9863	0.9863	0.9897	0.9726	0.9863	1.0	1.0	0.9863	0.9828	0.9828	0.9863	0.9863	1.0	0.9863

Table 6.1

	With SMOTE – NC													
	Training Set							Testing Set						
	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM
Accuracy	0.9980	0.9980	0.9941	0.9922	0.9902	1.0	1.0	0.9863	0.9863	0.9863	0.9726	0.9863	0.9863	0.9863
F1-Score	0.9980	0.9980	0.9941	0.9922	0.9902	1.0	1.0	0.9799	0.9799	0.9799	0.9610	0.9799	0.9809	0.9799
Precision	0.9980	0.9980	0.9941	0.9922	0.9902	1.0	1.0	0.9861	0.9861	0.9861	0.9610	0.9792	0.9861	0.9861
Recall	0.9980	0.9980	0.9941	0.9922	0.9902	1.0	1.0	0.9762	0.9762	0.9762	0.9610	0.9762	0.9848	0.9762

Table 6.2

Since this analysis is conducted with the purpose of medical diagnosis recall and F1-score are the best evaluation metrics as they calculate correctly predicted cases considering the false negative cases, which is predicting the disease to not be present when actually the disease is present. Comparing all the values in the above 2 tables, in models fitted with SMOTE–NC, a noticeable and consistent improvement was examined through multiple evaluation metrics in both training and test sets. This strongly suggest the advantage in continuing with SMOTE–NC technique as it effectively addresses the class imbalanced issue and enhances the overall model performance. Considering the models with and without SMOTE–NC, Random Forest in both cases shows the best accuracies in classifying data highlighting their robustness in generalizing to unseen dermatology data. Therefore, it can be said that the Random Forest model with SMOTE–NC delivers the best results out of all the fitted models.

As Random Forest with SMOTE–NC model shows the best performance, the model can be improved further by applying hyper parameter tuning. Also, since all these above-mentioned models in the presence of SMOTE–NC show considerably high performance, an ensemble model is fitted joining all of the models above. Here, the Random Forest model with tuned parameters is used instead of initial Random Forest model. The evaluation metrics of the Random Forest model after hyper parameter tuning and the ensemble model Voting Classifier are given in Table 6.3.

	Random Forest		Voting Classifier	
	Training Set	Testing Set	Training Set	Testing Set
Accuracy	0.9961	0.9863	0.9980	0.9863
F1-Score	0.9961	0.9799	0.9980	0.9799
Precision	0.9961	0.9861	0.9981	0.9861
Recall	0.9961	0.9762	0.9980	0.9762

Table 6.3

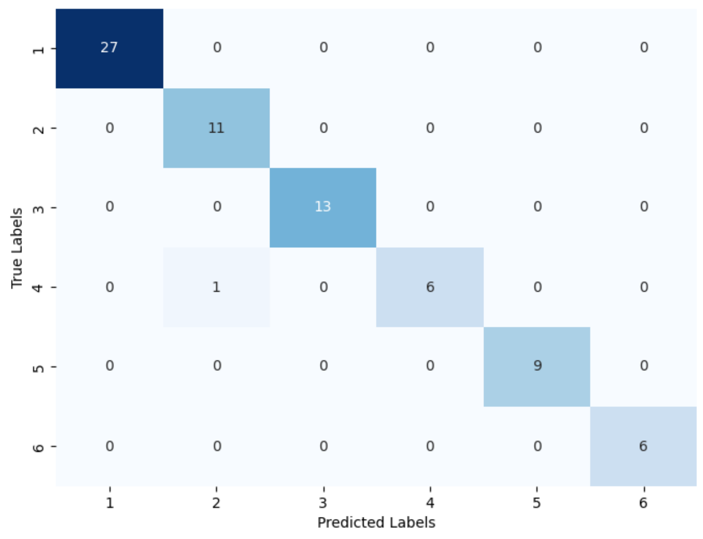


Figure 6.2

The best parameters for the Random Forest model are as follows,
Best Parameters: {'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}

By considering the F1-score, precision and recall values in the table 6.3, the performance of the Random Forest model has improved with the hyper parameter tuning. But when comparing the tuned Random Forest model with the Voting Classifier, it is evident that the ensemble model Voting Classifier shows better performance proving the high performance of the collectively captivated intelligence of multiple individual models and the appropriateness of using such models when dealing with complex and imbalanced datasets. Figure 6.2 gives the Confusion Matrix of the Voting Classifier.

Due to the high dimensionality (34 features) of this dataset the performance of certain predicting models can be inhibited. Therefore, dimensionality reduction is a crucial step to follow in order to obtain more refined and accurate models while reducing the complexity. Referring to the Feature Importance plot obtained from the Random Forest Classifier shown in Figure 6.3, the most important features in predicting the diseases classes can be extracted. Therefore, reduced models were fitted by reducing the features to 20, 18, 13 and 10. It was identified that almost all the model fitted using the top most 13 features in the presence of SMOTE-NC gives similar performance to all respective full models. The evaluation metrics of the optimum reduced models (13 variables) are as follows.

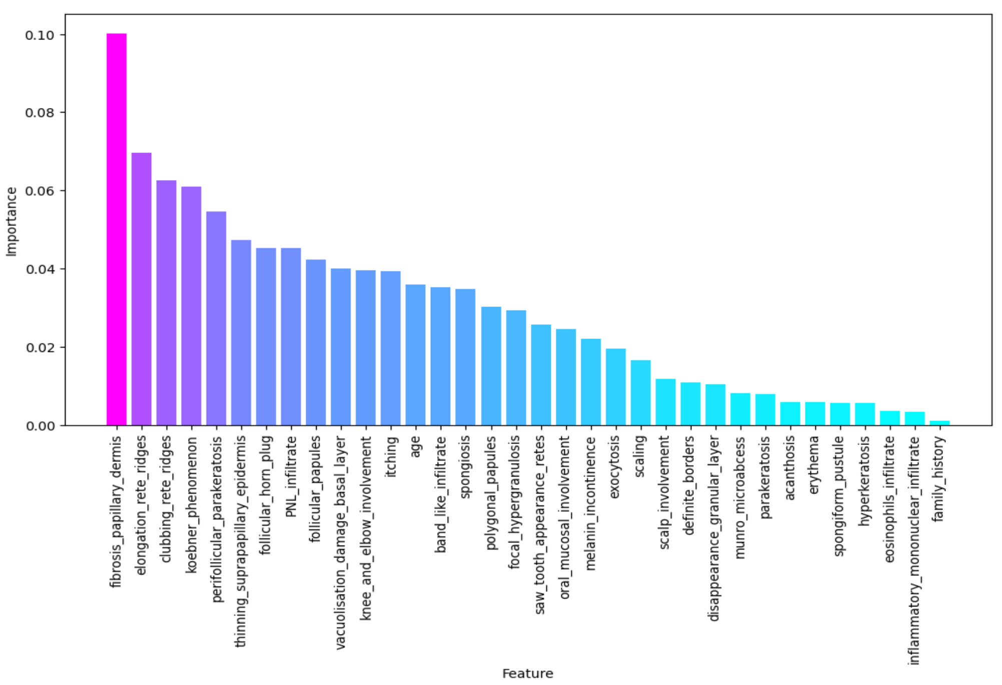


Figure 6.3

	Training Set							Testing Set						
	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM	Logistic Regression	Logistic Ridge	Logistic Lasso	KNN	Multinomial Naïve Bayes	Random Forest	SVM
Accuracy	0.9882	0.9882	0.9863	0.9843	0.9726	0.9980	0.9863	0.9726	0.9726	0.9863	0.9315	0.9725	0.9863	0.9863
F1-Score	0.9882	0.9882	0.9863	0.9843	0.9722	0.9980	0.9863	0.9635	0.9635	0.9799	0.9095	0.9633	0.9799	0.9799
Precision	0.9883	0.9883	0.9863	0.9843	0.9725	0.9980	0.9863	0.9744	0.9744	0.9861	0.9286	0.9682	0.9861	0.9861
Recall	0.9882	0.9882	0.9863	0.9843	0.9725	0.9980	0.9863	0.9577	0.9577	0.9762	0.9055	0.9610	0.9762	0.9762

Table 6.4

Since the overall performance of all models are considerably high and similar to the full model performance as shown in table 6.4, the analysis will be carried down further using the reduced dataset with the 13 selected features. By analyzing the above evaluation metrics, it is observed that the best performance is shown once again by the model Random Forest. Therefore, as done before, the model was further improved by applying hyper parameter tuning. Also since, all the above-mentioned models show high performance, a Voting Classifier, which is an ensemble model was also fitted to capture the overall accuracy of all models. The hyper parameter tuned Random Forest model was included in the ensemble model instead the original model. The obtain outputs are given below.

	Random Forest		Voting Classifier	
	Training Set	Testing Set	Training Set	Testing Set
Accuracy	0.9863	0.9843	0.9863	0.9799
F1-Score	0.9843	0.9799	0.9863	0.9861
Precision	0.9844	0.9861	0.9863	0.9762
Recall	0.9843	0.9762	0.9863	0.9762

Table 6.5

The best parameters for the Random Forest model are as follows,
Best Parameters: {'max_depth': 2, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 200}

As shown in the table 6.5 after carefully analyzing the F1-score, precision and recall values of the Random Forest model with hyper parameter tuning and the ensemble model Voting Classifier, it can be observed that the Voting Classifier gives out the highest accuracies in terms of the considered evaluation metrics and it outperforms all the other models that has been fitted so far. The confusion matrix of the test dataset of the Voting Classifier shown in Figure 6.4 and the classification report of the test dataset of the Voting Classifier shown in table 6.6 further approves the above finding. According to the confusion matrix all the observations except one are correctly classified and by analyzing the classification report, the accuracies in terms of precision, recall and F1-score of in classifying the test observations to 6 class levels are almost 100%.

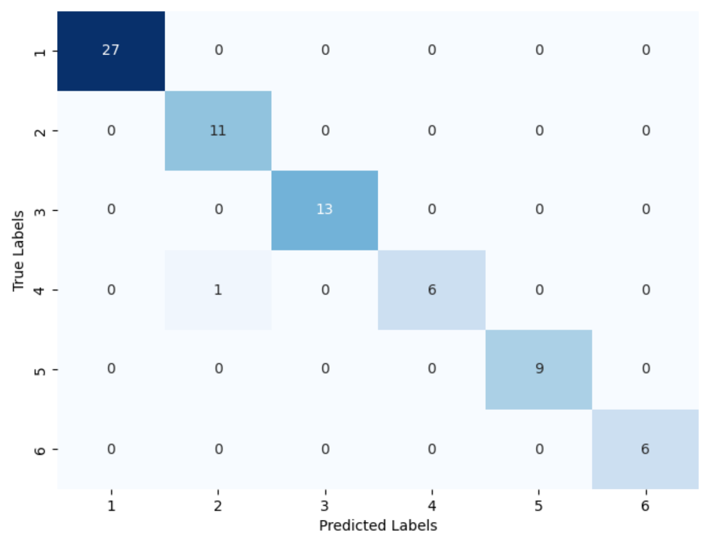


Figure 6.4

	Precision	Recall	F1-Score	Support
1	1.00	1.00	1.00	27
2	0.92	1.00	0.96	11
3	1.00	1.00	1.00	13
4	1.00	0.86	0.92	7
5	1.00	1.00	1.00	9
6	1.00	1.00	1.00	6
Accuracy			0.99	73
Macro average	0.99	0.98	0.98	73
Weighted average	0.99	0.99	0.99	73

Table 6.6

7. Issues Encountered and Proposed Solutions

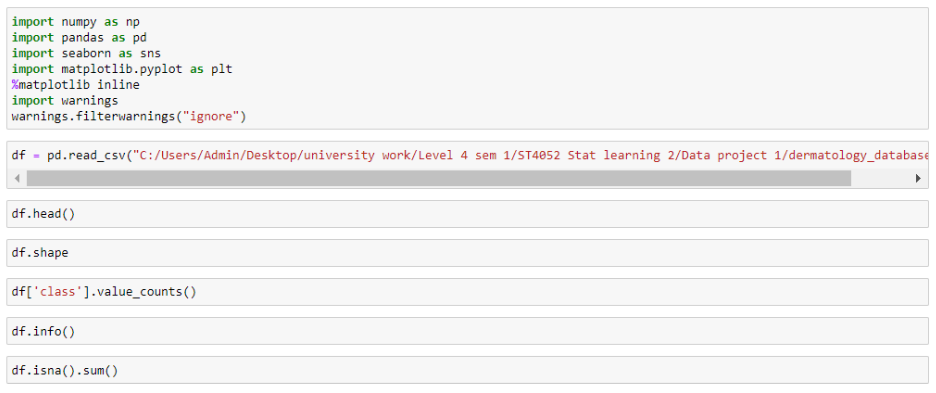
- The data distribution over 6 classes in this multiclass dataset was highly imbalanced and this caused certain classes to have very a less amount of observations and comparatively very few distinct features were able to differently identify the observations falling under these classes. Therefore, to overcome this issue and balance out the dataset, the oversampling technique SMOTE–NC (Synthetic Minority Oversampling Technique for Nominal and Continuous features) was used.
- After oversampling data to make the dataset balanced, unusual patters may be observed where certain features which were not prominent in classifying observations to a certain class may get prominent due to these newly created data using oversampling. To check such unusual observations were present, after applying SMOTE–NC to the dataset, a surface level descriptive analysis was conducted and the results obtained did not show any abnormalities.
- Almost all of the initially fitted predictive models exhibited high performance. Therefore, in classifying certain new observations some of these models may perform better than the others. In order to capture this collective intelligence, an ensemble model was created including all models. The model Voting Classifier was chosen as the ensemble model where it gives votes to the included models when they perform well in classifying given observations.
- The dataset was consisted of 34 predictor variables which is considerably a very high number to have and it causes high dimensionality in the dataset. Due to this reason the model fitting can be very complex and that will cause models to perform poorly. Therefore, to reduce dimensionality, a feature importance plot was obtained and the most important features (13 features) were extracted and models were refitted in a way that the accuracies are improved.

8. Discussion and Conclusion

- Due to the imbalanced nature of the dataset, the models fitted on the dataset which was balanced using the oversampling technique SMOTE – NC performed better and out of those models which includes Logistic Regression, Logistic Ridge Regression, Logistic Lasso Regression, K–Nearest Neighbors (KNN), Multinomial Naïve Bayes, Random Forest and Support Vector Machine (SVM) with linear kernel, the Random Forest model performed the best.
- The accuracy of the best performing Random Forest model was improved using hyper parameter tuning due to the possibility of overfitting.
- This tuned Random Forest Model and the rest of the previously fitted models were combined in and ensemble model as all of those had noticeably better performance and the created ensembled model using the Voting Classifier outperformed all models fitted so far.
- The accuracies of all those models fitted above was improved by reducing the dimensionality where the dataset with 13 features with the highest feature importance was extracted and models were fitted.
- Out of all the models obtained after dimensionality reduction, once again the ensemble model Voting Classifier outperforms all the predictive models fitted so far showing the best accuracies in term of F1-score, precision and recall.
- Therefore, the Voting Classifier fitted on the reduced dataset with balanced data (applying SMOTE–NC) was identified as the best model with the most accurate predictions and less information loss.

9. Appendix of the code

01.



```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```python
df = pd.read_csv("C:/Users/Admin/Desktop/university work/Level 4 sem 1/ST4052 Stat learning 2/Data project 1/dermatology_database
```

```python
df.head()
```

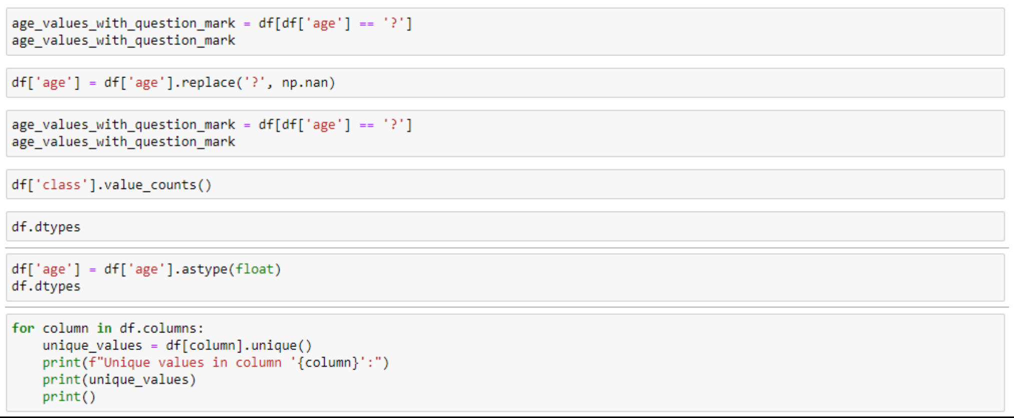
```python
df.shape
```

```python
df['class'].value_counts()
```

```python
df.info()
```

```python
df.isna().sum()
```

02.



```python
age_values_with_question_mark = df[df['age'] == '?']
age_values_with_question_mark
```

```python
df['age'] = df['age'].replace('?', np.nan)
```

```python
age_values_with_question_mark = df[df['age'] == '?']
age_values_with_question_mark
```

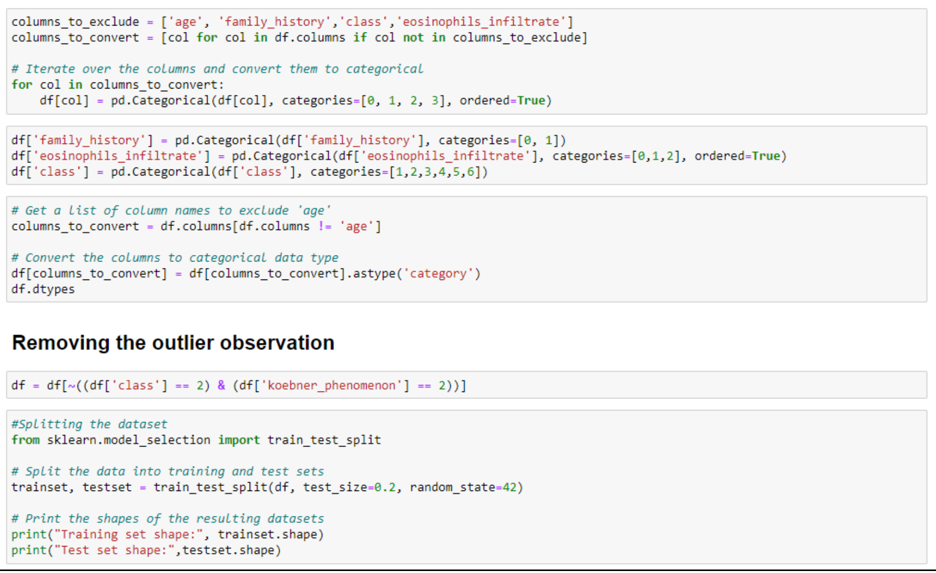
```python
df['class'].value_counts()
```

```python
df.dtypes
```

```python
df['age'] = df['age'].astype(float)
df.dtypes
```

```python
for column in df.columns:
    unique_values = df[column].unique()
    print(f"Unique values in column '{column}':")
    print(unique_values)
    print()
```

03.



```python
columns_to_exclude = ['age', 'family_history','class','eosinophils_infiltrate']
columns_to_convert = [col for col in df.columns if col not in columns_to_exclude]

# Iterate over the columns and convert them to categorical
for col in columns_to_convert:
    df[col] = pd.Categorical(df[col], categories=[0, 1, 2, 3], ordered=True)
```

```python
df['family_history'] = pd.Categorical(df['family_history'], categories=[0, 1])
df['eosinophils_infiltrate'] = pd.Categorical(df['eosinophils_infiltrate'], categories=[0,1,2], ordered=True)
df['class'] = pd.Categorical(df['class'], categories=[1,2,3,4,5,6])
```

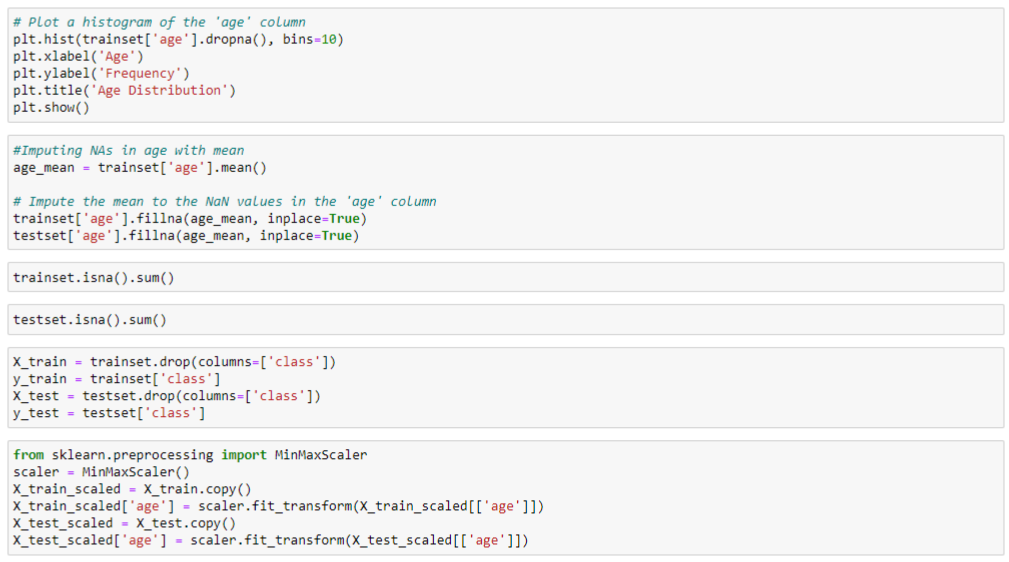
```python
# Get a list of column names to exclude 'age'
columns_to_convert = df.columns[df.columns != 'age']

# Convert the columns to categorical data type
df[columns_to_convert] = df[columns_to_convert].astype('category')
df.dtypes
```

Removing the outlier observation

```python
df = df[~((df['class'] == 2) & (df['koebner_phenomenon'] == 2))]
```

```python
#Splitting the dataset
from sklearn.model_selection import train_test_split

# Split the data into training and test sets
trainset, testset = train_test_split(df, test_size=0.2, random_state=42)

# Print the shapes of the resulting datasets
print("Training set shape:", trainset.shape)
print("Test set shape:",testset.shape)
```

04.



```python
# Plot a histogram of the 'age' column
plt.hist(trainset['age'].dropna(), bins=10)
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.title('Age Distribution')
plt.show()
```

```python
#Imputing NAs in age with mean
age_mean = trainset['age'].mean()

# Impute the mean to the NaN values in the 'age' column
trainset['age'].fillna(age_mean, inplace=True)
testset['age'].fillna(age_mean, inplace=True)
```

```python
trainset.isna().sum()
```

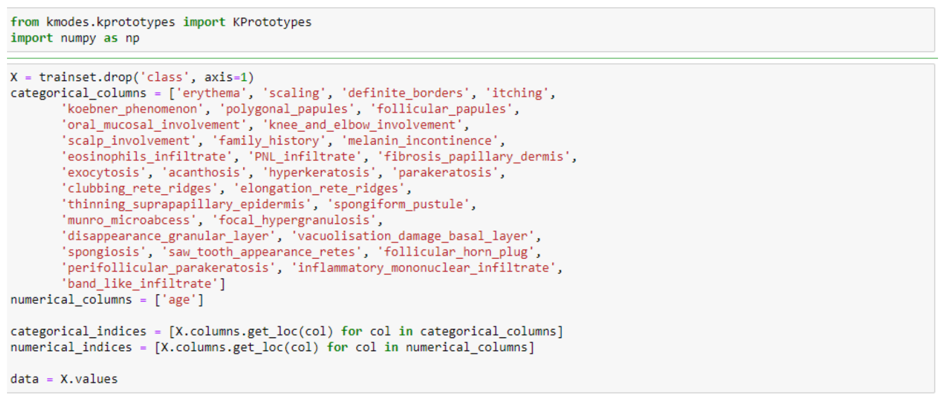
```python
testset.isna().sum()
```

```python
X_train = trainset.drop(columns=['class'])
y_train = trainset['class']
X_test = testset.drop(columns=['class'])
y_test = testset['class']
```

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = X_train.copy()
X_train_scaled['age'] = scaler.fit_transform(X_train_scaled[['age']])
X_test_scaled = X_test.copy()
X_test_scaled['age'] = scaler.fit_transform(X_test_scaled[['age']])
```

05.

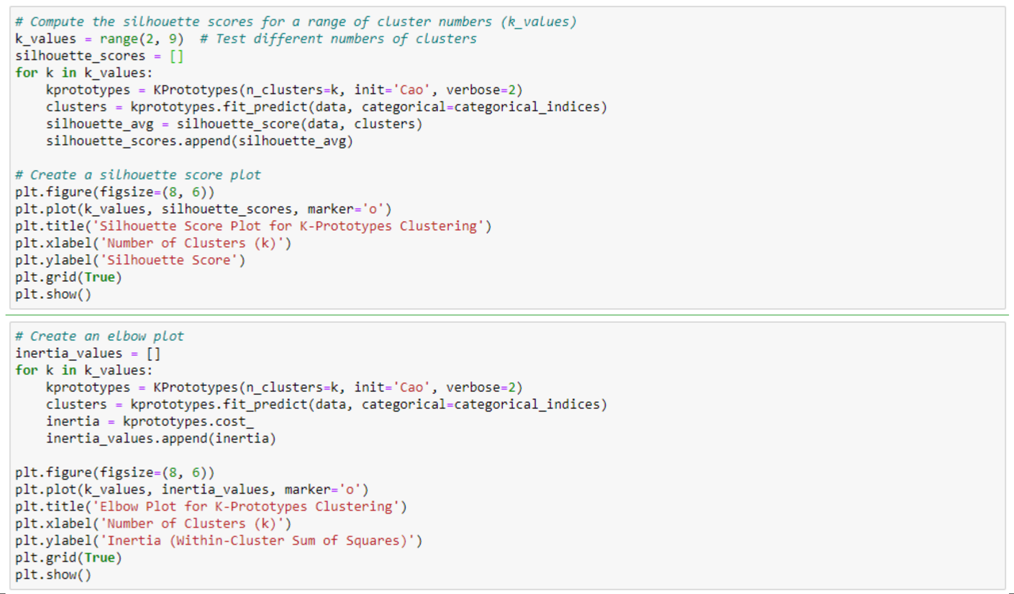
K-prototype Cluster Analysis



```
from kmodes.kprototypes import KPrototypes
import numpy as np
```

```
X = trainset.drop('class', axis=1)
categorical_columns = ['erythema', 'scaling', 'definite_borders', 'itching',
       'koebner_phenomenon', 'polygonal_papules', 'follicular_papules',
       'oral_mucosal_involvement', 'knee_and_elbow_involvement',
       'scalp_involvement', 'family_history', 'melanin_incontinence',
       'eosinophils_infiltrate', 'PNL_infiltrate', 'fibrosis_papillary_dermis',
       'exocytosis', 'acanthosis', 'hyperkeratosis', 'parakeratosis',
       'clubbing_rete_ridges', 'elongation_rete_ridges',
       'thinning_suprapapillary_epidermis', 'spongiform_pustule',
       'munro_microabcess', 'focal_hypergranulosis',
       'disappearance_granular_layer', 'vacuolisation_damage_basal_layer',
       'spongiosis', 'saw_tooth_appearance_retes', 'follicular_horn_plug',
       'perifollicular_parakeratosis', 'inflammatory_mononuclear_infiltrate',
       'band_like_infiltrate']
numerical_columns = ['age']

categorical_indices = [X.columns.get_loc(col) for col in categorical_columns]
numerical_indices = [X.columns.get_loc(col) for col in numerical_columns]

data = X.values
```

06.

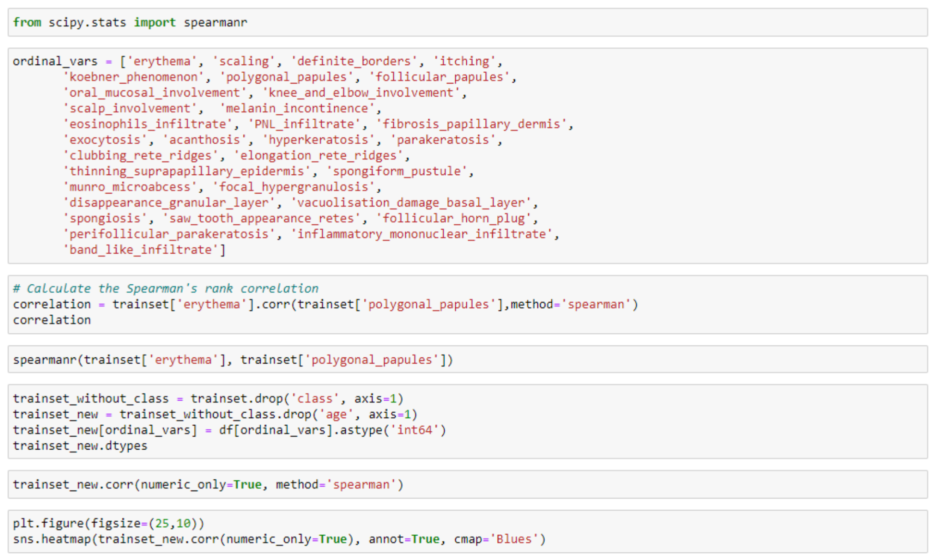


```
# Compute the silhouette scores for a range of cluster numbers (k_values)
k_values = range(2, 9)  # Test different numbers of clusters
silhouette_scores = []
for k in k_values:
    kprototypes = KPrototypes(n_clusters=k, init='Cao', verbose=2)
    clusters = kprototypes.fit_predict(data, categorical=categorical_indices)
    silhouette_avg = silhouette_score(data, clusters)
    silhouette_scores.append(silhouette_avg)

# Create a silhouette score plot
plt.figure(figsize=(8, 6))
plt.plot(k_values, silhouette_scores, marker='o')
plt.title('Silhouette Score Plot for K-Prototypes Clustering')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.grid(True)
plt.show()
```

```
# Create an elbow plot
inertia_values = []
for k in k_values:
    kprototypes = KPrototypes(n_clusters=k, init='Cao', verbose=2)
    clusters = kprototypes.fit_predict(data, categorical=categorical_indices)
    inertia = kprototypes.cost_
    inertia_values.append(inertia)

plt.figure(figsize=(8, 6))
plt.plot(k_values, inertia_values, marker='o')
plt.title('Elbow Plot for K-Prototypes Clustering')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-Cluster Sum of Squares)')
plt.grid(True)
plt.show()
```

07.

Spearman Rank Correlation



```
from scipy.stats import spearmanr
```

```
ordinal_vars = ['erythema', 'scaling', 'definite_borders', 'itching',
       'koebner_phenomenon', 'polygonal_papules', 'follicular_papules',
       'oral_mucosal_involvement', 'knee_and_elbow_involvement',
       'scalp_involvement',  'melanin_incontinence',
       'eosinophils_infiltrate', 'PNL_infiltrate', 'fibrosis_papillary_dermis',
       'exocytosis', 'acanthosis', 'hyperkeratosis', 'parakeratosis',
       'clubbing_rete_ridges', 'elongation_rete_ridges',
       'thinning_suprapapillary_epidermis', 'spongiform_pustule',
       'munro_microabcess', 'focal_hypergranulosis',
       'disappearance_granular_layer', 'vacuolisation_damage_basal_layer',
       'spongiosis', 'saw_tooth_appearance_retes', 'follicular_horn_plug',
       'perifollicular_parakeratosis', 'inflammatory_mononuclear_infiltrate',
       'band_like_infiltrate']
```

```
# Calculate the Spearman's rank correlation
correlation = trainset['erythema'].corr(trainset['polygonal_papules'],method='spearman')
correlation
```

```
spearmanr(trainset['erythema'], trainset['polygonal_papules'])
```

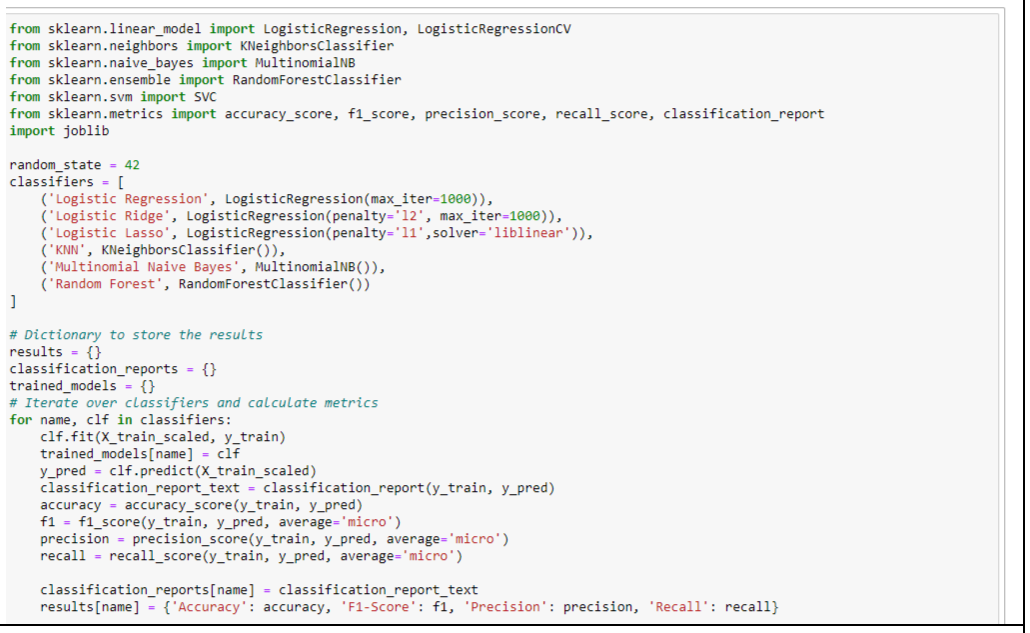
```
trainset_without_class = trainset.drop('class', axis=1)
trainset_new = trainset_without_class.drop('age', axis=1)
trainset_new[ordinal_vars] = df[ordinal_vars].astype('int64')
trainset_new.dtypes
```

```
trainset_new.corr(numeric_only=True, method='spearman')
```

```
plt.figure(figsize=(25,10))
sns.heatmap(trainset_new.corr(numeric_only=True), annot=True, cmap='Blues')
```

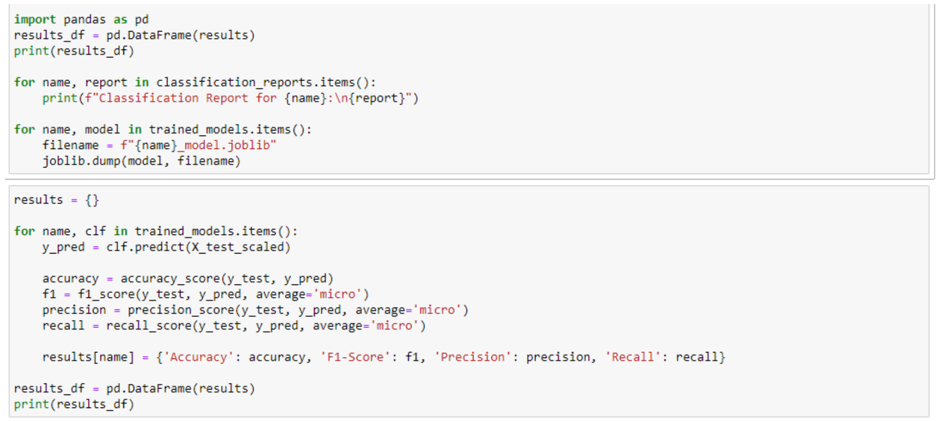
08.

Advanced Analysis



```
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
import joblib

random_state = 42
classifiers = [
    ('Logistic Regression', LogisticRegression(max_iter=1000)),
    ('Logistic Ridge', LogisticRegression(penalty='l2', max_iter=1000)),
    ('Logistic Lasso', LogisticRegression(penalty='l1',solver='liblinear')),
    ('KNN', KNeighborsClassifier()),
    ('Multinomial Naive Bayes', MultinomialNB()),
    ('Random Forest', RandomForestClassifier())
]

# Dictionary to store the results
results = {}
classification_reports = {}
trained_models = {}
# Iterate over classifiers and calculate metrics
for name, clf in classifiers:
    clf.fit(X_train_scaled, y_train)
    trained_models[name] = clf
    y_pred = clf.predict(X_train_scaled)
    classification_report_text = classification_report(y_train, y_pred)
    accuracy = accuracy_score(y_train, y_pred)
    f1 = f1_score(y_train, y_pred, average='micro')
    precision = precision_score(y_train, y_pred, average='micro')
    recall = recall_score(y_train, y_pred, average='micro')

    classification_reports[name] = classification_report_text
    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}
```

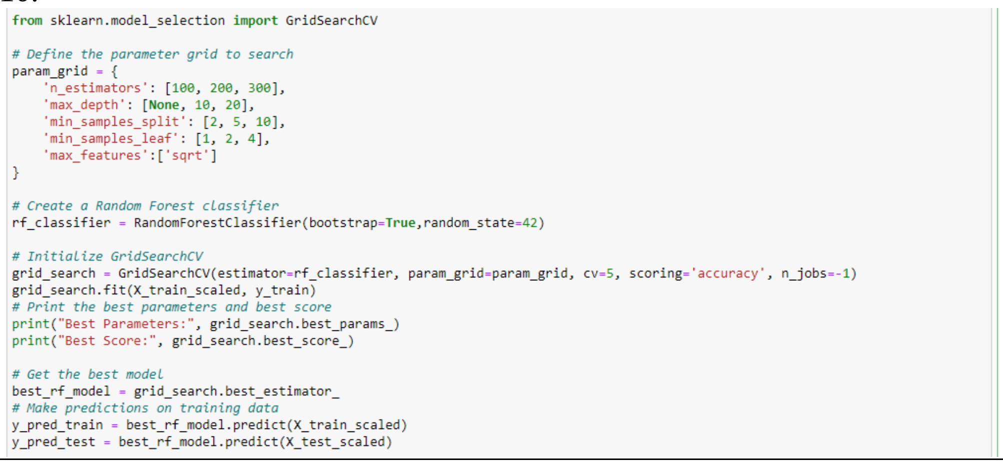
09.



```
import pandas as pd
results_df = pd.DataFrame(results)
print(results_df)

for name, report in classification_reports.items():
    print(f"Classification Report for {name}:\n{report}")

for name, model in trained_models.items():
    filename = f"{name}_model.joblib"
    joblib.dump(model, filename)
```

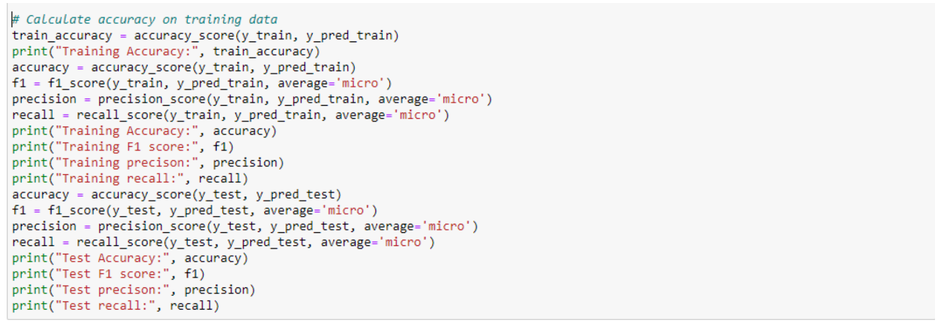
```
results = {}

for name, clf in trained_models.items():
    y_pred = clf.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='micro')
    precision = precision_score(y_test, y_pred, average='micro')
    recall = recall_score(y_test, y_pred, average='micro')

    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}

results_df = pd.DataFrame(results)
print(results_df)
```

10.



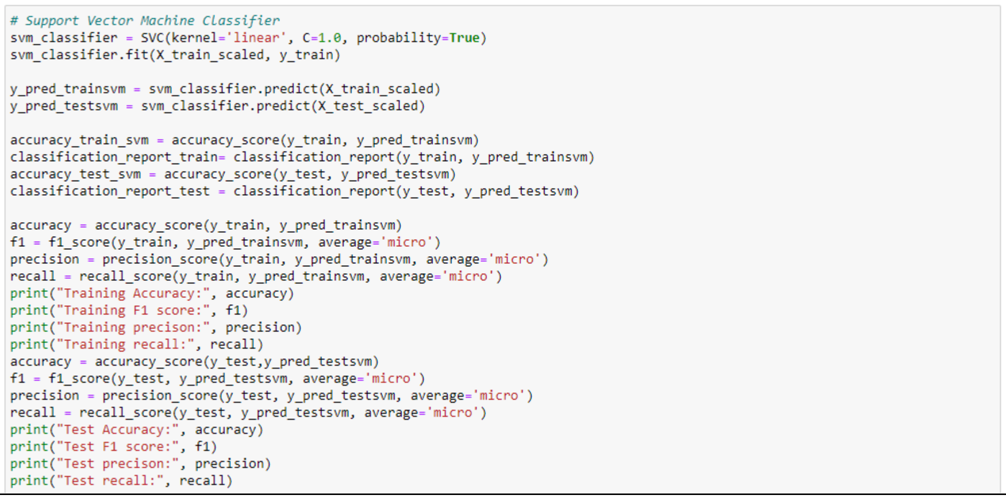
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features':['sqrt']
}

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(bootstrap=True,random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)
# Print the best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Get the best model
best_rf_model = grid_search.best_estimator_
# Make predictions on training data
y_pred_train = best_rf_model.predict(X_train_scaled)
y_pred_test = best_rf_model.predict(X_test_scaled)
```

11.



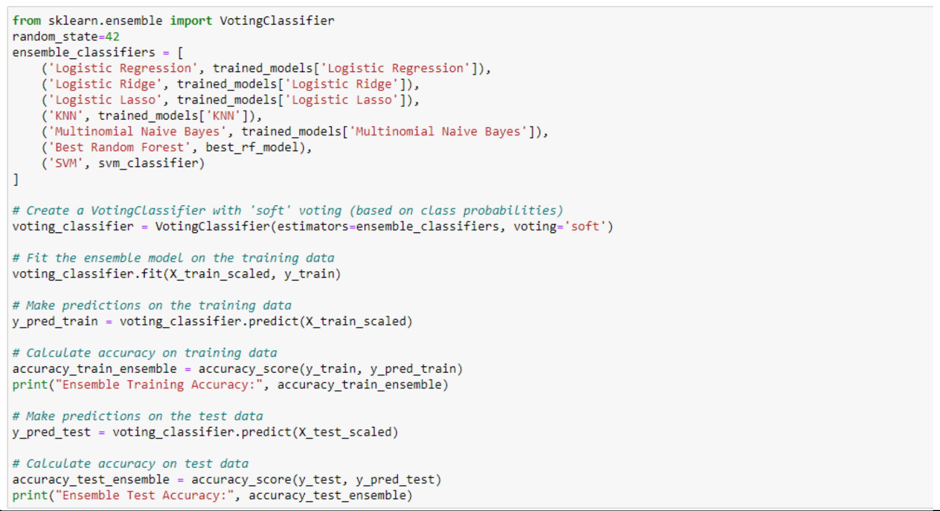
```
# Calculate accuracy on training data
train_accuracy = accuracy_score(y_train, y_pred_train)
print("Training Accuracy:", train_accuracy)
accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='micro')
precision = precision_score(y_train, y_pred_train, average='micro')
recall = recall_score(y_train, y_pred_train, average='micro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='micro')
precision = precision_score(y_test, y_pred_test, average='micro')
recall = recall_score(y_test, y_pred_test, average='micro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)
```

12.



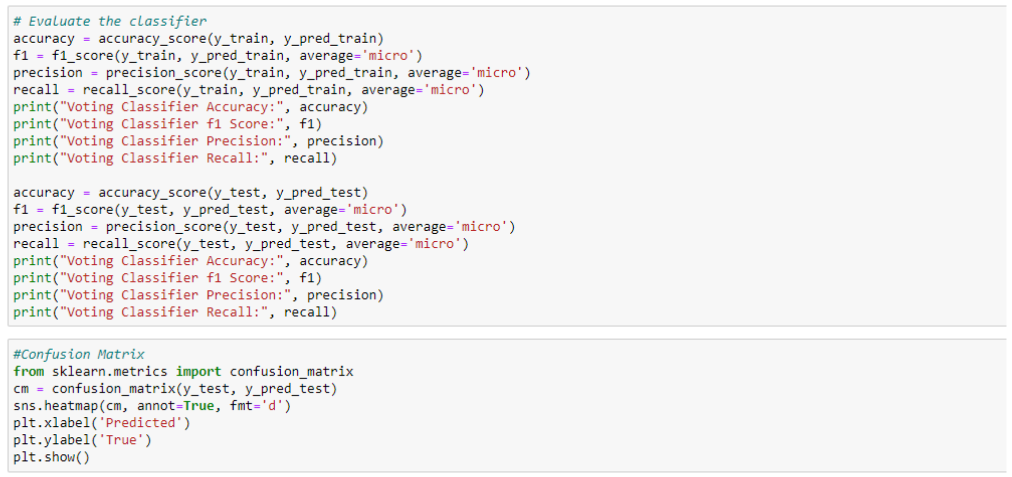
```
# Support Vector Machine Classifier
svm_classifier = SVC(kernel='linear', C=1.0, probability=True)
svm_classifier.fit(X_train_scaled, y_train)

y_pred_trainsvm = svm_classifier.predict(X_train_scaled)
y_pred_testsvm = svm_classifier.predict(X_test_scaled)

accuracy_train_svm = accuracy_score(y_train, y_pred_trainsvm)
classification_report_train= classification_report(y_train, y_pred_trainsvm)
accuracy_test_svm = accuracy_score(y_test, y_pred_testsvm)
classification_report_test = classification_report(y_test, y_pred_testsvm)

accuracy = accuracy_score(y_train, y_pred_trainsvm)
f1 = f1_score(y_train, y_pred_trainsvm, average='micro')
precision = precision_score(y_train, y_pred_trainsvm, average='micro')
recall = recall_score(y_train, y_pred_trainsvm, average='micro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test,y_pred_testsvm)
f1 = f1_score(y_test, y_pred_testsvm, average='micro')
precision = precision_score(y_test, y_pred_testsvm, average='micro')
recall = recall_score(y_test, y_pred_testsvm, average='micro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)
```

13.



```python
from sklearn.ensemble import VotingClassifier
random_state=42
ensemble_classifiers = [
    ('Logistic Regression', trained_models['Logistic Regression']),
    ('Logistic Ridge', trained_models['Logistic Ridge']),
    ('Logistic Lasso', trained_models['Logistic Lasso']),
    ('KNN', trained_models['KNN']),
    ('Multinomial Naive Bayes', trained_models['Multinomial Naive Bayes']),
    ('Best Random Forest', best_rf_model),
    ('SVM', svm_classifier)
]

# Create a VotingClassifier with 'soft' voting (based on class probabilities)
voting_classifier = VotingClassifier(estimators=ensemble_classifiers, voting='soft')

# Fit the ensemble model on the training data
voting_classifier.fit(X_train_scaled, y_train)

# Make predictions on the training data
y_pred_train = voting_classifier.predict(X_train_scaled)

# Calculate accuracy on training data
accuracy_train_ensemble = accuracy_score(y_train, y_pred_train)
print("Ensemble Training Accuracy:", accuracy_train_ensemble)

# Make predictions on the test data
y_pred_test = voting_classifier.predict(X_test_scaled)

# Calculate accuracy on test data
accuracy_test_ensemble = accuracy_score(y_test, y_pred_test)
print("Ensemble Test Accuracy:", accuracy_test_ensemble)
```

14.

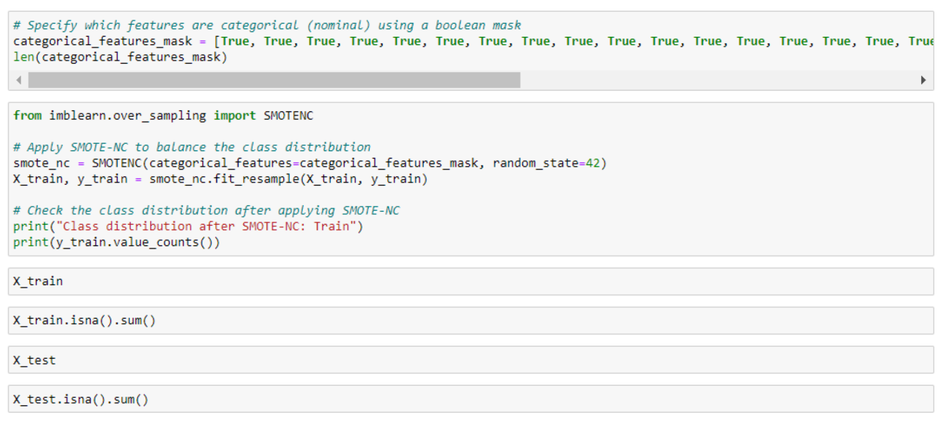


```python
# Evaluate the classifier
accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='micro')
precision = precision_score(y_train, y_pred_train, average='micro')
recall = recall_score(y_train, y_pred_train, average='micro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)

accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='micro')
precision = precision_score(y_test, y_pred_test, average='micro')
recall = recall_score(y_test, y_pred_test, average='micro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)
```

```python
#Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

15.

SMOTE - NC



```python
# Specify which features are categorical (nominal) using a boolean mask
categorical_features_mask = [True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True, True
len(categorical_features_mask)
```

```python
from imblearn.over_sampling import SMOTENC

# Apply SMOTE-NC to balance the class distribution
smote_nc = SMOTENC(categorical_features=categorical_features_mask, random_state=42)
X_train, y_train = smote_nc.fit_resample(X_train, y_train)

# Check the class distribution after applying SMOTE-NC
print("Class distribution after SMOTE-NC: Train")
print(y_train.value_counts())
```

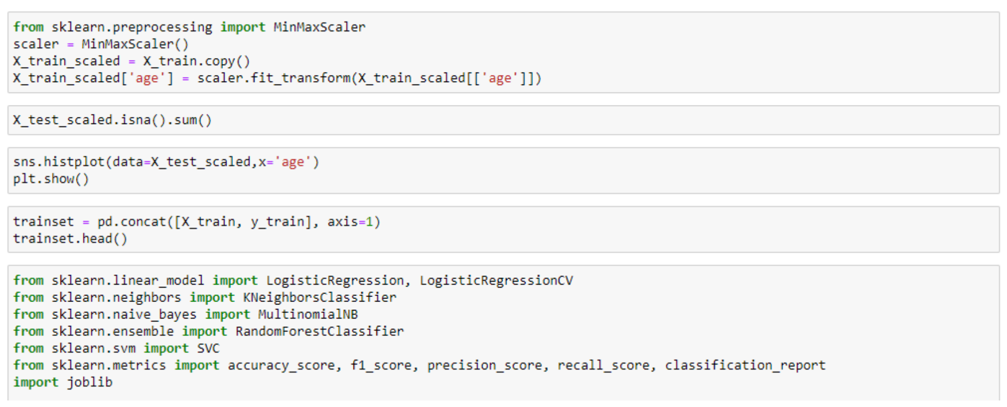
```python
X_train
```

```python
X_train.isna().sum()
```

```python
X_test
```

```python
X_test.isna().sum()
```

16.



```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = X_train.copy()
X_train_scaled['age'] = scaler.fit_transform(X_train_scaled[['age']])
```

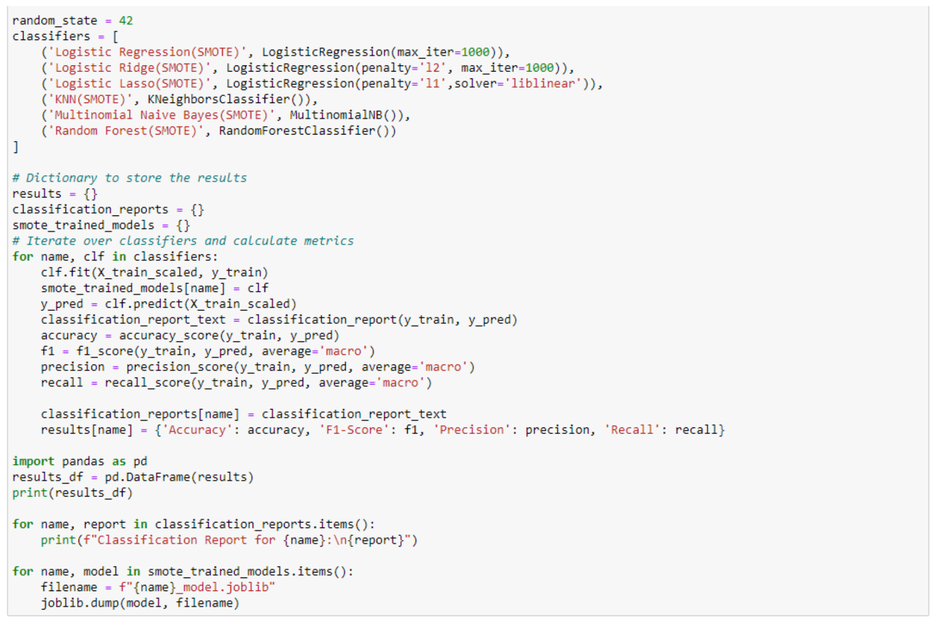
```python
X_test_scaled.isna().sum()
```

```python
sns.histplot(data=X_test_scaled,x='age')
plt.show()
```

```python
trainset = pd.concat([X_train, y_train], axis=1)
trainset.head()
```

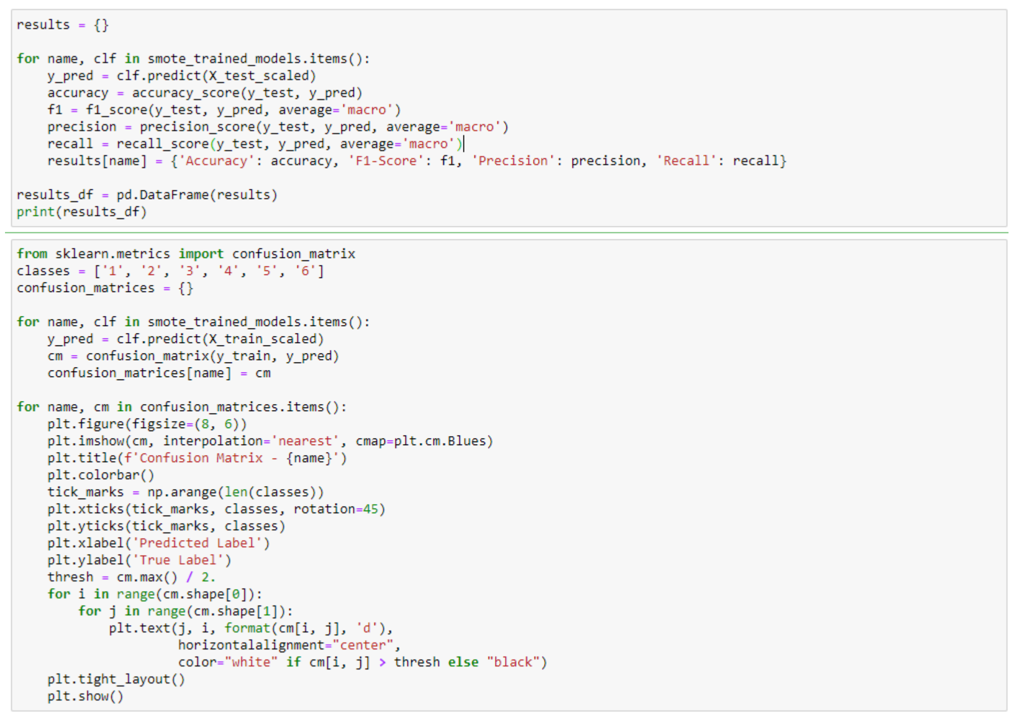
```python
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, classification_report
import joblib
```

17.



```python
random_state = 42
classifiers = [
    ('Logistic Regression(SMOTE)', LogisticRegression(max_iter=1000)),
    ('Logistic Ridge(SMOTE)', LogisticRegression(penalty='l2', max_iter=1000)),
    ('Logistic Lasso(SMOTE)', LogisticRegression(penalty='l1',solver='liblinear')),
    ('KNN(SMOTE)', KNeighborsClassifier()),
    ('Multinomial Naive Bayes(SMOTE)', MultinomialNB()),
    ('Random Forest(SMOTE)', RandomForestClassifier())
]

# Dictionary to store the results
results = {}
classification_reports = {}
smote_trained_models = {}
# Iterate over classifiers and calculate metrics
for name, clf in classifiers:
    clf.fit(X_train_scaled, y_train)
    smote_trained_models[name] = clf
    y_pred = clf.predict(X_train_scaled)
    classification_report_text = classification_report(y_train, y_pred)
    accuracy = accuracy_score(y_train, y_pred)
    f1 = f1_score(y_train, y_pred, average='macro')
    precision = precision_score(y_train, y_pred, average='macro')
    recall = recall_score(y_train, y_pred, average='macro')

    classification_reports[name] = classification_report_text
    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}

import pandas as pd
results_df = pd.DataFrame(results)
print(results_df)

for name, report in classification_reports.items():
    print(f"Classification Report for {name}:\n{report}")

for name, model in smote_trained_models.items():
    filename = f"{name}_model.joblib"
    joblib.dump(model, filename)
```

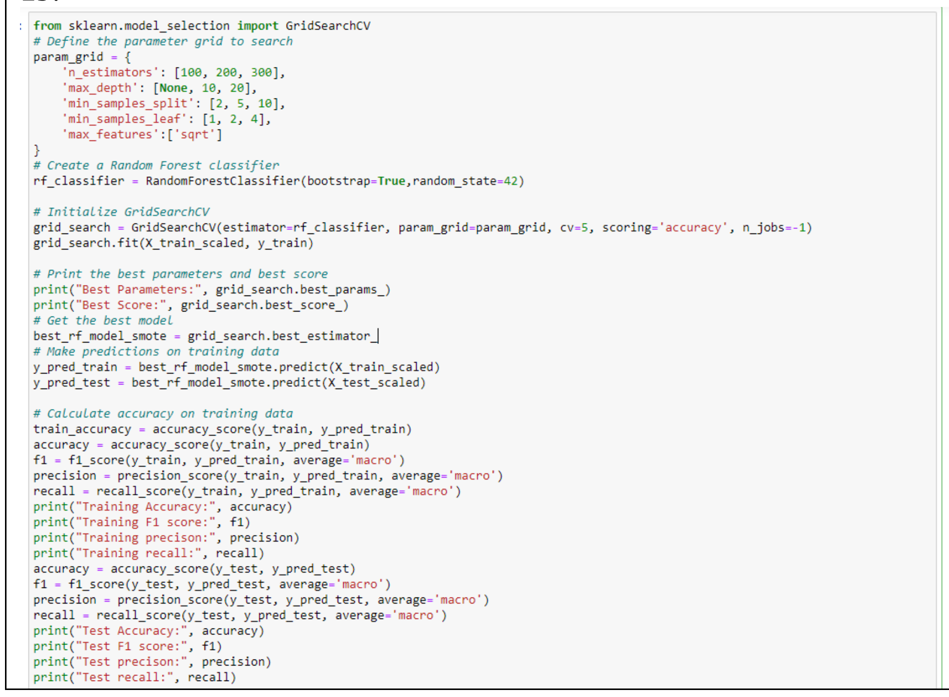
18.



```python
results = {}

for name, clf in smote_trained_models.items():
    y_pred = clf.predict(X_test_scaled)
    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='macro')
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}

results_df = pd.DataFrame(results)
print(results_df)
```

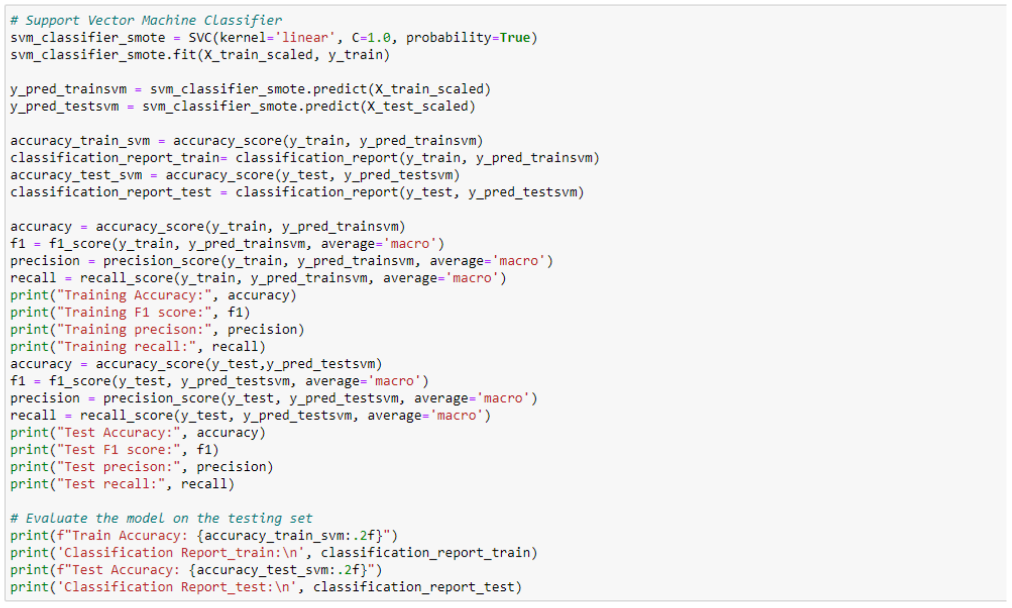
```python
from sklearn.metrics import confusion_matrix
classes = ['1', '2', '3', '4', '5', '6']
confusion_matrices = {}

for name, clf in smote_trained_models.items():
    y_pred = clf.predict(X_train_scaled)
    cm = confusion_matrix(y_train, y_pred)
    confusion_matrices[name] = cm

for name, cm in confusion_matrices.items():
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
    plt.title(f'Confusion Matrix - {name}')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    plt.xlabel('Predicted Label')
    plt.ylabel('True Label')
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, format(cm[i, j], 'd'),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.show()
```

19.



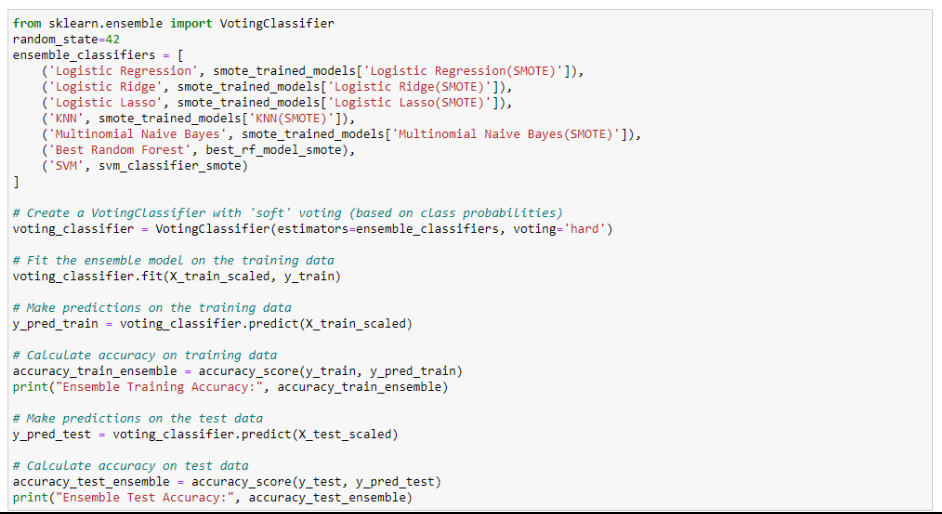
```python
from sklearn.model_selection import GridSearchCV
# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features':['sqrt']
}
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(bootstrap=True,random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Print the best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
# Get the best model
best_rf_model_smote = grid_search.best_estimator_
# Make predictions on training data
y_pred_train = best_rf_model_smote.predict(X_train_scaled)
y_pred_test = best_rf_model_smote.predict(X_test_scaled)

# Calculate accuracy on training data
train_accuracy = accuracy_score(y_train, y_pred_train)
accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='macro')
precision = precision_score(y_train, y_pred_train, average='macro')
recall = recall_score(y_train, y_pred_train, average='macro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='macro')
precision = precision_score(y_test, y_pred_test, average='macro')
recall = recall_score(y_test, y_pred_test, average='macro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)
```

20.



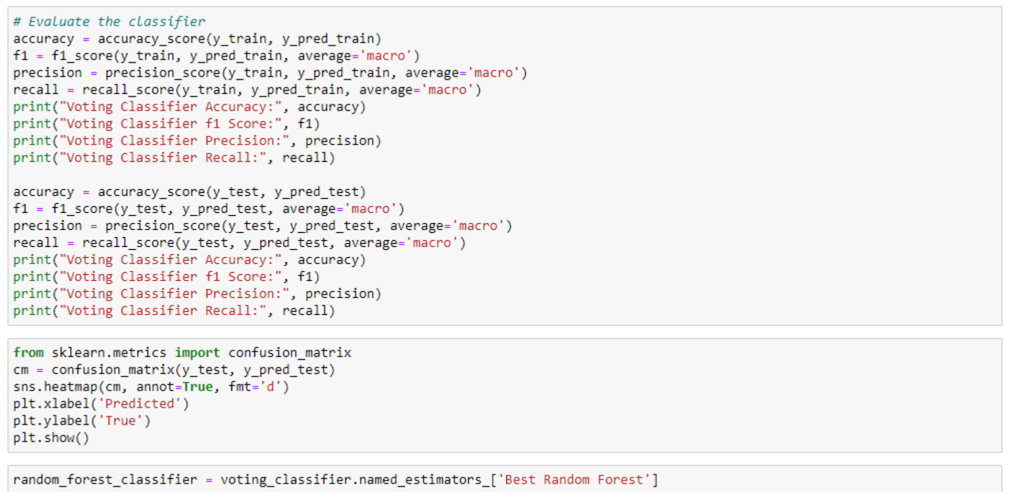
```python
# Support Vector Machine Classifier
svm_classifier_smote = SVC(kernel='linear', C=1.0, probability=True)
svm_classifier_smote.fit(X_train_scaled, y_train)

y_pred_trainsvm = svm_classifier_smote.predict(X_train_scaled)
y_pred_testsvm = svm_classifier_smote.predict(X_test_scaled)

accuracy_train_svm = accuracy_score(y_train, y_pred_trainsvm)
classification_report_train= classification_report(y_train, y_pred_trainsvm)
accuracy_test_svm = accuracy_score(y_test, y_pred_testsvm)
classification_report_test = classification_report(y_test, y_pred_testsvm)

accuracy = accuracy_score(y_train, y_pred_trainsvm)
f1 = f1_score(y_train, y_pred_trainsvm, average='macro')
precision = precision_score(y_train, y_pred_trainsvm, average='macro')
recall = recall_score(y_train, y_pred_trainsvm, average='macro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test,y_pred_testsvm)
f1 = f1_score(y_test, y_pred_testsvm, average='macro')
precision = precision_score(y_test, y_pred_testsvm, average='macro')
recall = recall_score(y_test, y_pred_testsvm, average='macro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)

# Evaluate the model on the testing set
print(f"Train Accuracy: {accuracy_train_svm:.2f}")
print('Classification Report_train:\n', classification_report_train)
print(f"Test Accuracy: {accuracy_test_svm:.2f}")
print('Classification Report_test:\n', classification_report_test)
```

21.



```
from sklearn.ensemble import VotingClassifier
random_state=42
ensemble_classifiers = [
    ('Logistic Regression', smote_trained_models['Logistic Regression(SMOTE)']),
    ('Logistic Ridge', smote_trained_models['Logistic Ridge(SMOTE)']),
    ('Logistic Lasso', smote_trained_models['Logistic Lasso(SMOTE)']),
    ('KNN', smote_trained_models['KNN(SMOTE)']),
    ('Multinomial Naive Bayes', smote_trained_models['Multinomial Naive Bayes(SMOTE)']),
    ('Best Random Forest', best_rf_model_smote),
    ('SVM', svm_classifier_smote)
]

# Create a VotingClassifier with 'soft' voting (based on class probabilities)
voting_classifier = VotingClassifier(estimators=ensemble_classifiers, voting='hard')

# Fit the ensemble model on the training data
voting_classifier.fit(X_train_scaled, y_train)

# Make predictions on the training data
y_pred_train = voting_classifier.predict(X_train_scaled)

# Calculate accuracy on training data
accuracy_train_ensemble = accuracy_score(y_train, y_pred_train)
print("Ensemble Training Accuracy:", accuracy_train_ensemble)

# Make predictions on the test data
y_pred_test = voting_classifier.predict(X_test_scaled)

# Calculate accuracy on test data
accuracy_test_ensemble = accuracy_score(y_test, y_pred_test)
print("Ensemble Test Accuracy:", accuracy_test_ensemble)
```

22.

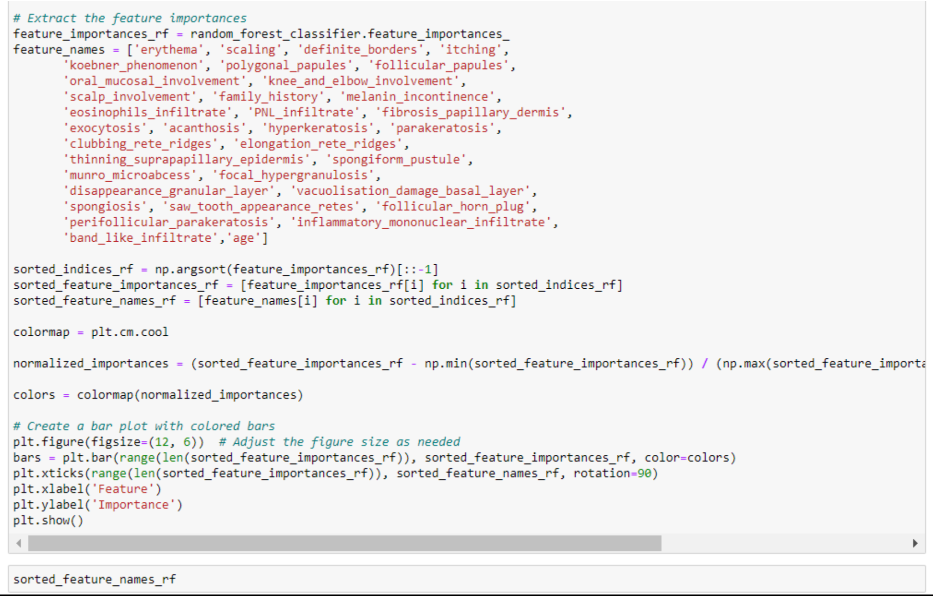


```
# Evaluate the classifier
accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='macro')
precision = precision_score(y_train, y_pred_train, average='macro')
recall = recall_score(y_train, y_pred_train, average='macro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)

accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='macro')
precision = precision_score(y_test, y_pred_test, average='macro')
recall = recall_score(y_test, y_pred_test, average='macro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
random_forest_classifier = voting_classifier.named_estimators_['Best Random Forest']
```

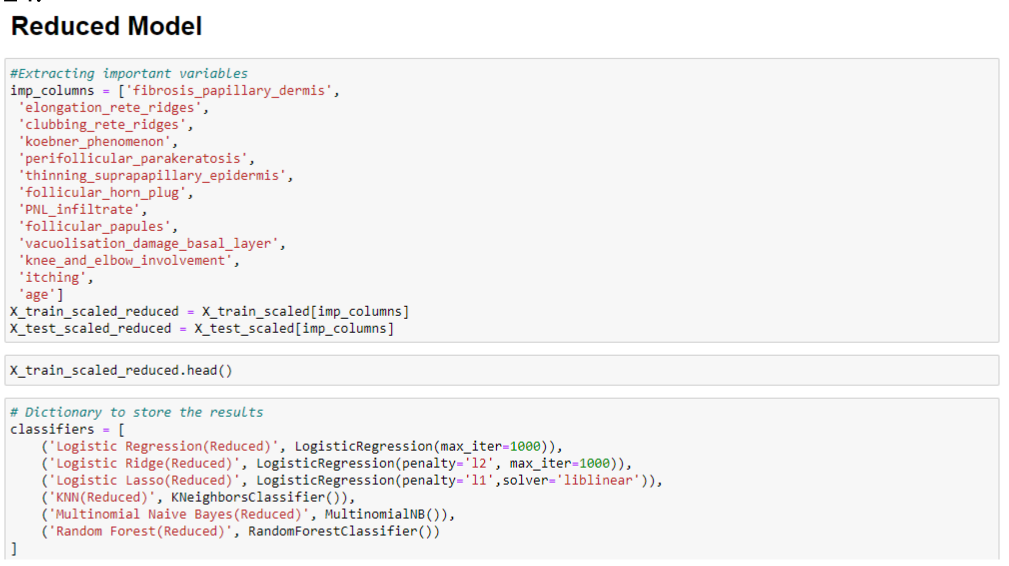
23.



```
# Extract the feature importances
feature_importances_rf = random_forest_classifier.feature_importances_
feature_names = ['erythema', 'scaling', 'definite_borders', 'itching',
       'koebner_phenomenon', 'polygonal_papules', 'follicular_papules',
       'oral_mucosal_involvement', 'knee_and_elbow_involvement',
       'scalp_involvement', 'family_history', 'melanin_incontinence',
       'eosinophils_infiltrate', 'PNL_infiltrate', 'fibrosis_papillary_dermis',
       'exocytosis', 'acanthosis', 'hyperkeratosis', 'parakeratosis',
       'clubbing_rete_ridges', 'elongation_rete_ridges',
       'thinning_suprapapillary_epidermis', 'spongiform_pustule',
       'munro_microabcess', 'focal_hypergranulosis',
       'disappearance_granular_layer', 'vacuolisation_damage_basal_layer',
       'spongiosis', 'saw_tooth_appearance_retes', 'follicular_horn_plug',
       'perifollicular_parakeratosis', 'inflammatory_mononuclear_infiltrate',
       'band_like_infiltrate','age']

sorted_indices_rf = np.argsort(feature_importances_rf)[::-1]
sorted_feature_importances_rf = [feature_importances_rf[i] for i in sorted_indices_rf]
sorted_feature_names_rf = [feature_names[i] for i in sorted_indices_rf]

colormap = plt.cm.cool

normalized_importances = (sorted_feature_importances_rf - np.min(sorted_feature_importances_rf)) / (np.max(sorted_feature_importa

colors = colormap(normalized_importances)

# Create a bar plot with colored bars
plt.figure(figsize=(12, 6))  # Adjust the figure size as needed
bars = plt.bar(range(len(sorted_feature_importances_rf)), sorted_feature_importances_rf, color=colors)
plt.xticks(range(len(sorted_feature_importances_rf)), sorted_feature_names_rf, rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()
```

```
sorted_feature_names_rf
```

24.



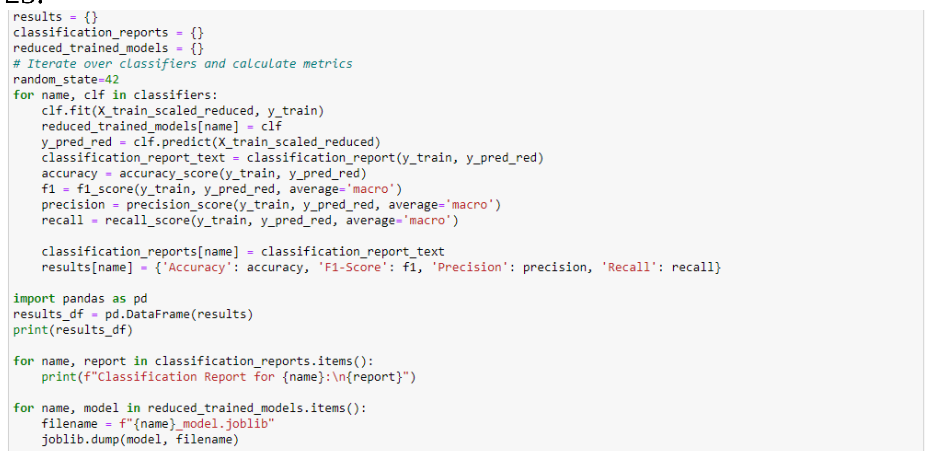
Reduced Model

```
#Extracting important variables
imp_columns = ['fibrosis_papillary_dermis',
 'elongation_rete_ridges',
 'clubbing_rete_ridges',
 'koebner_phenomenon',
 'perifollicular_parakeratosis',
 'thinning_suprapapillary_epidermis',
 'follicular_horn_plug',
 'PNL_infiltrate',
 'follicular_papules',
 'vacuolisation_damage_basal_layer',
 'knee_and_elbow_involvement',
 'itching',
 'age']
X_train_scaled_reduced = X_train_scaled[imp_columns]
X_test_scaled_reduced = X_test_scaled[imp_columns]
```

```
X_train_scaled_reduced.head()
```

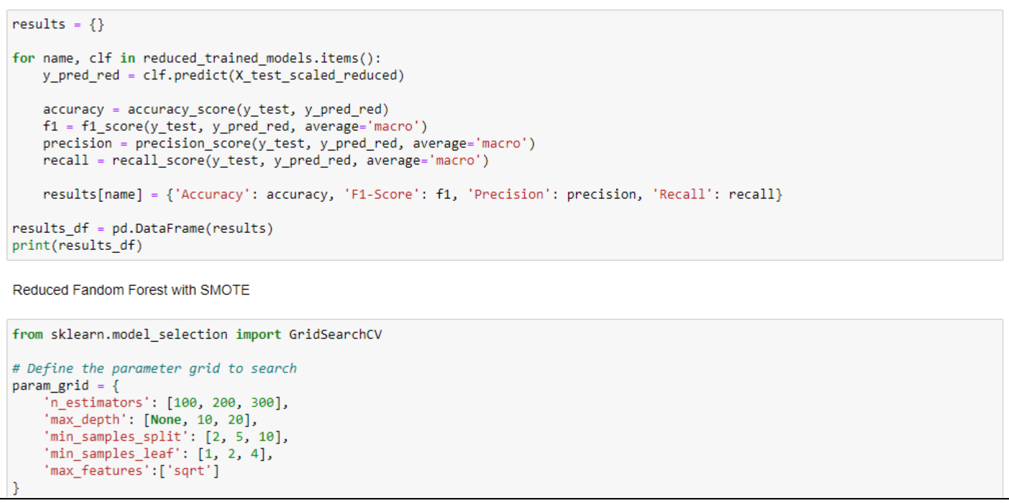
```
# Dictionary to store the results
classifiers = [
    ('Logistic Regression(Reduced)', LogisticRegression(max_iter=1000)),
    ('Logistic Ridge(Reduced)', LogisticRegression(penalty='l2', max_iter=1000)),
    ('Logistic Lasso(Reduced)', LogisticRegression(penalty='l1',solver='liblinear')),
    ('KNN(Reduced)', KNeighborsClassifier()),
    ('Multinomial Naive Bayes(Reduced)', MultinomialNB()),
    ('Random Forest(Reduced)', RandomForestClassifier())
]
```

25.



```
results = {}
classification_reports = {}
reduced_trained_models = {}
# Iterate over classifiers and calculate metrics
random_state=42
for name, clf in classifiers:
    clf.fit(X_train_scaled_reduced, y_train)
    reduced_trained_models[name] = clf
    y_pred_red = clf.predict(X_train_scaled_reduced)
    classification_report_text = classification_report(y_train, y_pred_red)
    accuracy = accuracy_score(y_train, y_pred_red)
    f1 = f1_score(y_train, y_pred_red, average='macro')
    precision = precision_score(y_train, y_pred_red, average='macro')
    recall = recall_score(y_train, y_pred_red, average='macro')

    classification_reports[name] = classification_report_text
    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}

import pandas as pd
results_df = pd.DataFrame(results)
print(results_df)

for name, report in classification_reports.items():
    print(f"Classification Report for {name}:\n{report}")

for name, model in reduced_trained_models.items():
    filename = f"{name}_model.joblib"
    joblib.dump(model, filename)
```

26.

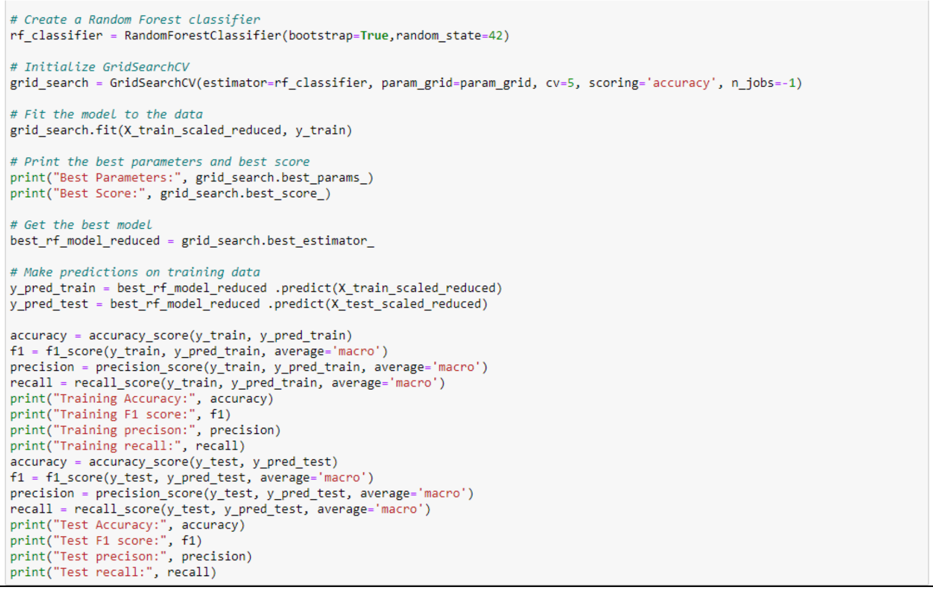


```
results = {}

for name, clf in reduced_trained_models.items():
    y_pred_red = clf.predict(X_test_scaled_reduced)

    accuracy = accuracy_score(y_test, y_pred_red)
    f1 = f1_score(y_test, y_pred_red, average='macro')
    precision = precision_score(y_test, y_pred_red, average='macro')
    recall = recall_score(y_test, y_pred_red, average='macro')

    results[name] = {'Accuracy': accuracy, 'F1-Score': f1, 'Precision': precision, 'Recall': recall}

results_df = pd.DataFrame(results)
print(results_df)
```

Reduced Fandom Forest with SMOTE

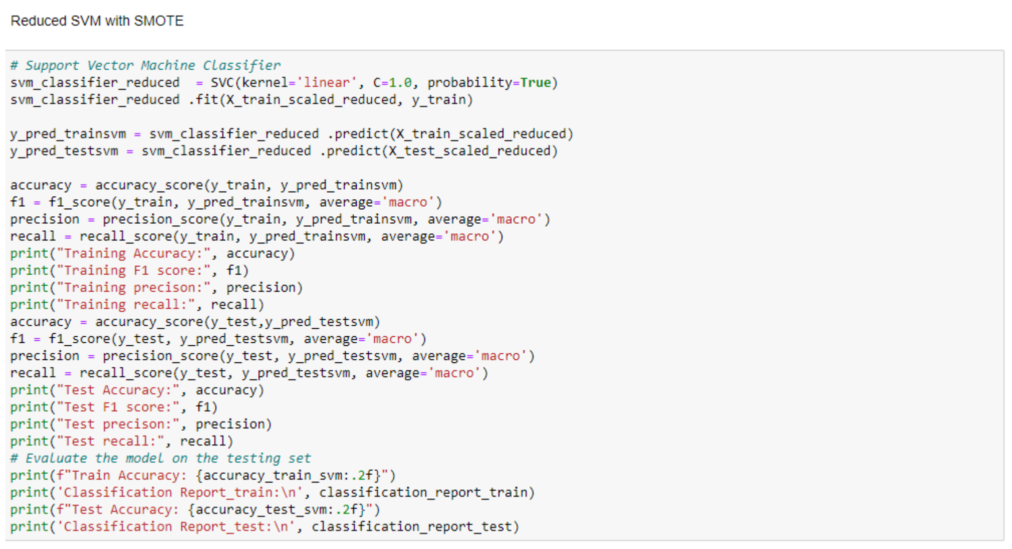
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid to search
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features':['sqrt']
}
```

27.



```
# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(bootstrap=True,random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the model to the data
grid_search.fit(X_train_scaled_reduced, y_train)

# Print the best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

# Get the best model
best_rf_model_reduced = grid_search.best_estimator_

# Make predictions on training data
y_pred_train = best_rf_model_reduced .predict(X_train_scaled_reduced)
y_pred_test = best_rf_model_reduced .predict(X_test_scaled_reduced)

accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='macro')
precision = precision_score(y_train, y_pred_train, average='macro')
recall = recall_score(y_train, y_pred_train, average='macro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='macro')
precision = precision_score(y_test, y_pred_test, average='macro')
recall = recall_score(y_test, y_pred_test, average='macro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)
```

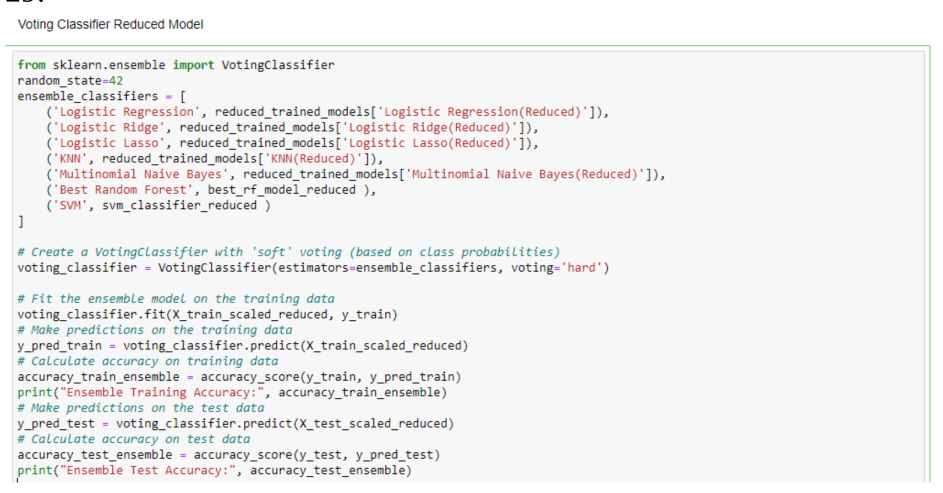
28.



Reduced SVM with SMOTE

```
# Support Vector Machine Classifier
svm_classifier_reduced  = SVC(kernel='linear', C=1.0, probability=True)
svm_classifier_reduced .fit(X_train_scaled_reduced, y_train)

y_pred_trainsvm = svm_classifier_reduced .predict(X_train_scaled_reduced)
y_pred_testsvm = svm_classifier_reduced .predict(X_test_scaled_reduced)

accuracy = accuracy_score(y_train, y_pred_trainsvm)
f1 = f1_score(y_train, y_pred_trainsvm, average='macro')
precision = precision_score(y_train, y_pred_trainsvm, average='macro')
recall = recall_score(y_train, y_pred_trainsvm, average='macro')
print("Training Accuracy:", accuracy)
print("Training F1 score:", f1)
print("Training precison:", precision)
print("Training recall:", recall)
accuracy = accuracy_score(y_test,y_pred_testsvm)
f1 = f1_score(y_test, y_pred_testsvm, average='macro')
precision = precision_score(y_test, y_pred_testsvm, average='macro')
recall = recall_score(y_test, y_pred_testsvm, average='macro')
print("Test Accuracy:", accuracy)
print("Test F1 score:", f1)
print("Test precison:", precision)
print("Test recall:", recall)
# Evaluate the model on the testing set
print(f"Train Accuracy: {accuracy_train_svm:.2f}")
print('Classification Report_train:\n', classification_report_train)
print(f"Test Accuracy: {accuracy_test_svm:.2f}")
print('Classification Report_test:\n', classification_report_test)
```

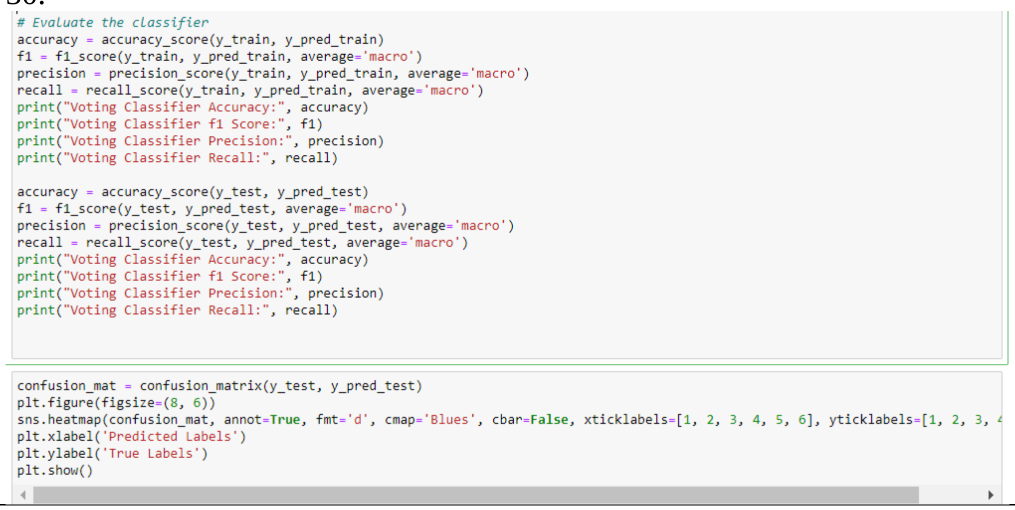
29.



Voting Classifier Reduced Model

```python
from sklearn.ensemble import VotingClassifier
random_state=42
ensemble_classifiers = [
    ('Logistic Regression', reduced_trained_models['Logistic Regression(Reduced)']),
    ('Logistic Ridge', reduced_trained_models['Logistic Ridge(Reduced)']),
    ('Logistic Lasso', reduced_trained_models['Logistic Lasso(Reduced)']),
    ('KNN', reduced_trained_models['KNN(Reduced)']),
    ('Multinomial Naive Bayes', reduced_trained_models['Multinomial Naive Bayes(Reduced)']),
    ('Best Random Forest', best_rf_model_reduced ),
    ('SVM', svm_classifier_reduced )
]

# Create a VotingClassifier with 'soft' voting (based on class probabilities)
voting_classifier = VotingClassifier(estimators=ensemble_classifiers, voting='hard')

# Fit the ensemble model on the training data
voting_classifier.fit(X_train_scaled_reduced, y_train)
# Make predictions on the training data
y_pred_train = voting_classifier.predict(X_train_scaled_reduced)
# Calculate accuracy on training data
accuracy_train_ensemble = accuracy_score(y_train, y_pred_train)
print("Ensemble Training Accuracy:", accuracy_train_ensemble)
# Make predictions on the test data
y_pred_test = voting_classifier.predict(X_test_scaled_reduced)
# Calculate accuracy on test data
accuracy_test_ensemble = accuracy_score(y_test, y_pred_test)
print("Ensemble Test Accuracy:", accuracy_test_ensemble)
```

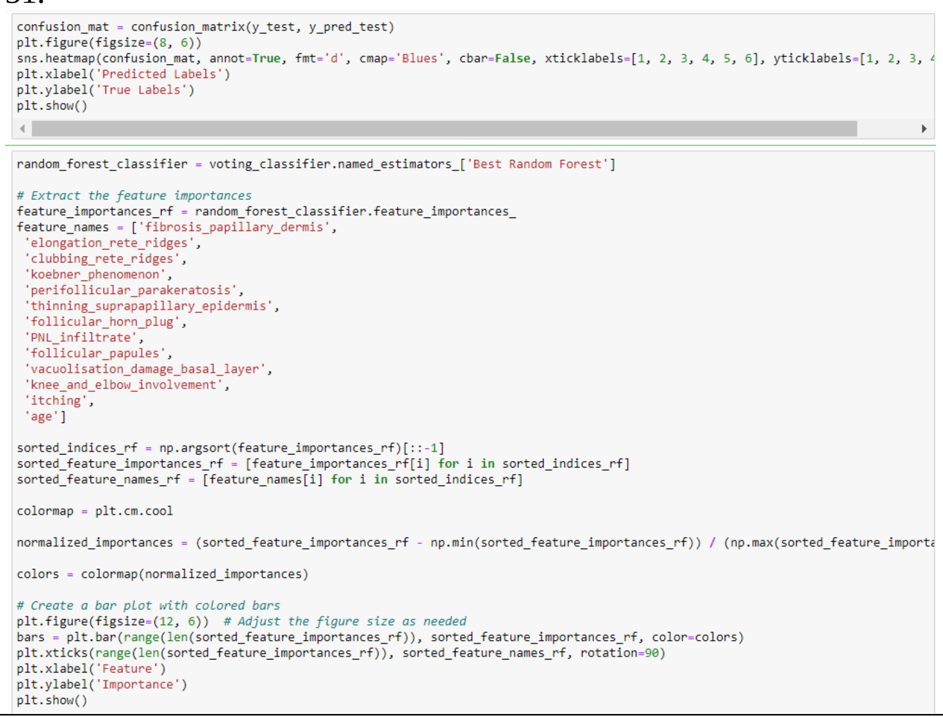
30.



```python
# Evaluate the classifier
accuracy = accuracy_score(y_train, y_pred_train)
f1 = f1_score(y_train, y_pred_train, average='macro')
precision = precision_score(y_train, y_pred_train, average='macro')
recall = recall_score(y_train, y_pred_train, average='macro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)

accuracy = accuracy_score(y_test, y_pred_test)
f1 = f1_score(y_test, y_pred_test, average='macro')
precision = precision_score(y_test, y_pred_test, average='macro')
recall = recall_score(y_test, y_pred_test, average='macro')
print("Voting Classifier Accuracy:", accuracy)
print("Voting Classifier f1 Score:", f1)
print("Voting Classifier Precision:", precision)
print("Voting Classifier Recall:", recall)
```

```python
confusion_mat = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=[1, 2, 3, 4, 5, 6], yticklabels=[1, 2, 3, 4
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

31.



```python
confusion_mat = confusion_matrix(y_test, y_pred_test)
plt.figure(figsize=(8, 6))
sns.heatmap(confusion_mat, annot=True, fmt='d', cmap='Blues', cbar=False, xticklabels=[1, 2, 3, 4, 5, 6], yticklabels=[1, 2, 3, 4
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
```

```python
random_forest_classifier = voting_classifier.named_estimators_['Best Random Forest']

# Extract the feature importances
feature_importances_rf = random_forest_classifier.feature_importances_
feature_names = ['fibrosis_papillary_dermis',
 'elongation_rete_ridges',
 'clubbing_rete_ridges',
 'koebner_phenomenon',
 'perifollicular_parakeratosis',
 'thinning_suprapapillary_epidermis',
 'follicular_horn_plug',
 'PNL_infiltrate',
 'follicular_papules',
 'vacuolisation_damage_basal_layer',
 'knee_and_elbow_involvement',
 'itching',
 'age']

sorted_indices_rf = np.argsort(feature_importances_rf)[::-1]
sorted_feature_importances_rf = [feature_importances_rf[i] for i in sorted_indices_rf]
sorted_feature_names_rf = [feature_names[i] for i in sorted_indices_rf]

colormap = plt.cm.cool

normalized_importances = (sorted_feature_importances_rf - np.min(sorted_feature_importances_rf)) / (np.max(sorted_feature_importa

colors = colormap(normalized_importances)

# Create a bar plot with colored bars
plt.figure(figsize=(12, 6))  # Adjust the figure size as needed
bars = plt.bar(range(len(sorted_feature_importances_rf)), sorted_feature_importances_rf, color=colors)
plt.xticks(range(len(sorted_feature_importances_rf)), sorted_feature_names_rf, rotation=90)
plt.xlabel('Feature')
plt.ylabel('Importance')
plt.show()
```