Prediction And Analysis Of Liver Patient Data Using Machine Learning

Introduction

1.1. Project overviews

1.2. Objectives

2. Project Initialization and Planning Phase

2.1. Define Problem Statement

2.2. Project Proposal (Proposed Solution)

2.3. Initial Project Planning

3. Data Collection and Preprocessing Phase

3.1. Data Collection Plan and Raw Data Sources Identified

3.2. Data Quality Report

3.3. Data Exploration and Preprocessing

4. Model Development Phase

4.1. Feature Selection Report

4.2. Model Selection Report

4.3. Initial Model Training Code, Model Validation and Evaluation Report

5. Model Optimization and Tuning Phase

5.1. Hyperparameter Tuning Documentation

5.2. Performance Metrics Comparison Report

5.3. Final Model Selection Justification

6. Results

6.1. Output Screenshots

7. Advantages & Disadvantages

8. Conclusion

9. Future Scope

10. Appendix

10.1. Source Code

10.2. GitHub & Project Demo Link

1.INTRODUCTION

1.1 Project overview

Liver is the largest organ in the abdomen. This is the primary organ for maintaining the chemicals like glucose, balancing so many nutrients, fat, vitamins, cholesterol and hormones. Liver disease prevents normal liver function. Mainly due to the large amount of alcohol consumption liver disease develops. Early detection of liver disease using classification algorithms is an effective activity that can help doctors diagnose the disease in a short period of time. Early detection of liver disease is a daunting task for doctors. The main purpose of our project is to analyse the parameters of the various classification algorithms and compare their predictive accuracy to determine the best stage for determining liver disease.

1.2 Objectives

The primary objective is to develop a machine learning model that accurately predicts the likelihood of a patient developing a liver disease based on their medical history, symptoms, and other relevant data, enabling early intervention and improved health outcomes

2. Project Initialization and Planning Phase

2.1 Define Problem Statement

The early prediction of liver disease can significantly improve patient outcomes, reduce healthcare costs, and enhance the quality of life. With the advent of big data, machine learning, and advanced analytics, there is a growing potential to predict diseases before they manifest clinically. This predictive capability can aid in proactive treatment, preventive measures, and personalized healthcare. In the realm of healthcare, accurately predicting the onset and progression of disease is crucial for early intervention and improved patient outcomes. Despite advancements in medical technology and data analytics, current methods often lack precision and timeliness, leading to delayed diagnoses and suboptimal treatment plans. This project aims to develop a robust and scalable liver disease prediction model that leverages patient data, including medical history, genetic information, lifestyle factors, and real-time health indicators, to forecast the likelihood of disease occurrence with high accuracy. The goal is to enhance predictive capabilities, reduce healthcare costs, and ultimately improve patient care through timely and personalized interventions.

2.2 Project Proposal (Proposed solution)

This project proposal outlines a solution to address the problem of early liver disease detection through machine learning. With a clear objective to develop a predictive model for assessing disease risk based on symptoms, lifestyle factors, and health data, the proposal defines the scope of the project, including data collection, model development, and deployment. The proposed solution details the approach to be used, key features of the model, and specifies the resource requirements including hardware, software, and personnel. By creating an accurate and user friendly tool, the project aims to enable proactive health management and improve early liver disease detection.

2.3 Initial Project Planning

Develop a machine learning model to predict the likelihood of a liver disease based on patient data. The scope includes data collection, basic preprocessing, model development, and initial evaluation for accuracy. The model will be deployed in a simple application for healthcare use. Ongoing maintenance and updates are not included.

3. Data Collection and Preprocessing Phase

3.1 Data Collection Plan and Raw Data Sources Identified

- **Data Availability**: Determine which sources provide the required data and whether they are accessible.
- **Data Quality**: Evaluate the reliability, completeness, and accuracy of the data from each source.
- **Data Consistency**: Ensure consistency across different sources and data formats.Time: time in format hh:mm:ss
- **Data Extraction**: Design protocols for extracting data from electronic health records (EHRs), databases, or files.

- **Data Integration**: Merge data from multiple sources into a unified dataset suitable for analysis.
- **Data Cleaning**: Preprocess data to handle missing values, outliers, and inconsistencies.
- **Hospital EHRs**: Patient demographics, medical history, medications, and laboratory results.
- **Public Datasets**: Liver disease datasets from repositories like UCI Machine Learning Repository.
- **Research Databases**: Specific datasets from liver disease research studies.
- **Health Insurance Claims**: Historical claims data with diagnostic codes related to liver diseases.
- **3.2 Data Quality Report**
- Data quality is ensured through thorough verification, addressing missing values, and maintaining adherence to ethical guidelines, establishing a reliable foundation for predictive modeling.

3.3 Data Exploration and preprocessing

- Data Exploration involves analyzing the liver disease patient dataset to understand patterns, distributions, and outliers.
- Preprocessing includes handling missing values, scaling, and encoding categorical variables.
- These crucial steps enhance data quality, ensuring the reliability and effectiveness of subsequent analysis.

4. Model Development Phase

4.1 Feature Selection Report

- The Feature Selection Report outlines the rationale behind choosing specific features (e.g., age, food plan, gender) for the liver disease prediction model.
- It evaluates relevance, importance, and impact on predictive accuracy, ensuring the inclusion of key factors influencing the model's ability.

4.2 Model Selection Report

- The Model Selection Report details the rationale behind choosing Linear Regression, Random Forest, Decision Tree, and XGB models for liver disease prediction.

- It considers each model's strengths in handling complex relationships, interpretability, adaptability, and overall predictive performance, ensuring an informed choice aligned with project objectives.

4.3 Initial Model Training Code, Model Validation and Evaluation Report

- The Initial Model Training Code employs selected algorithms on the liver disease dataset, setting the foundation for predictive modeling.
- The subsequent Model Validation and Evaluation Report rigorously assesses model performance, employing metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), R-squared error to ensure reliability and effectiveness in predicting liver disease.

5.Model Optimization and Tuning Phase

Final Model Selection Justification

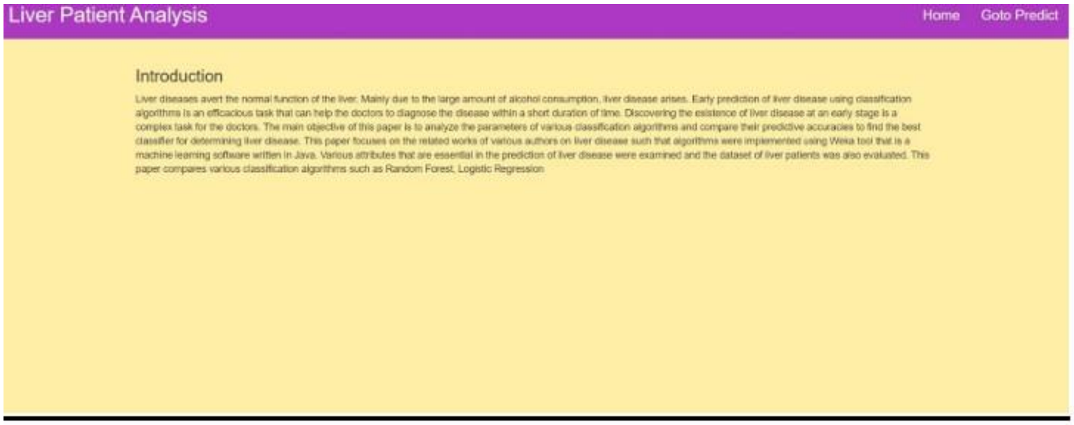
- The XGBoost Regressor is the final model chosen because of its best overall performance compared to the other models.
- It captures the variance in the data very well with minimal prediction error. XGBoost can capture complex non-linear relationships.

6. RESULTS

6.1 Output Screenshots

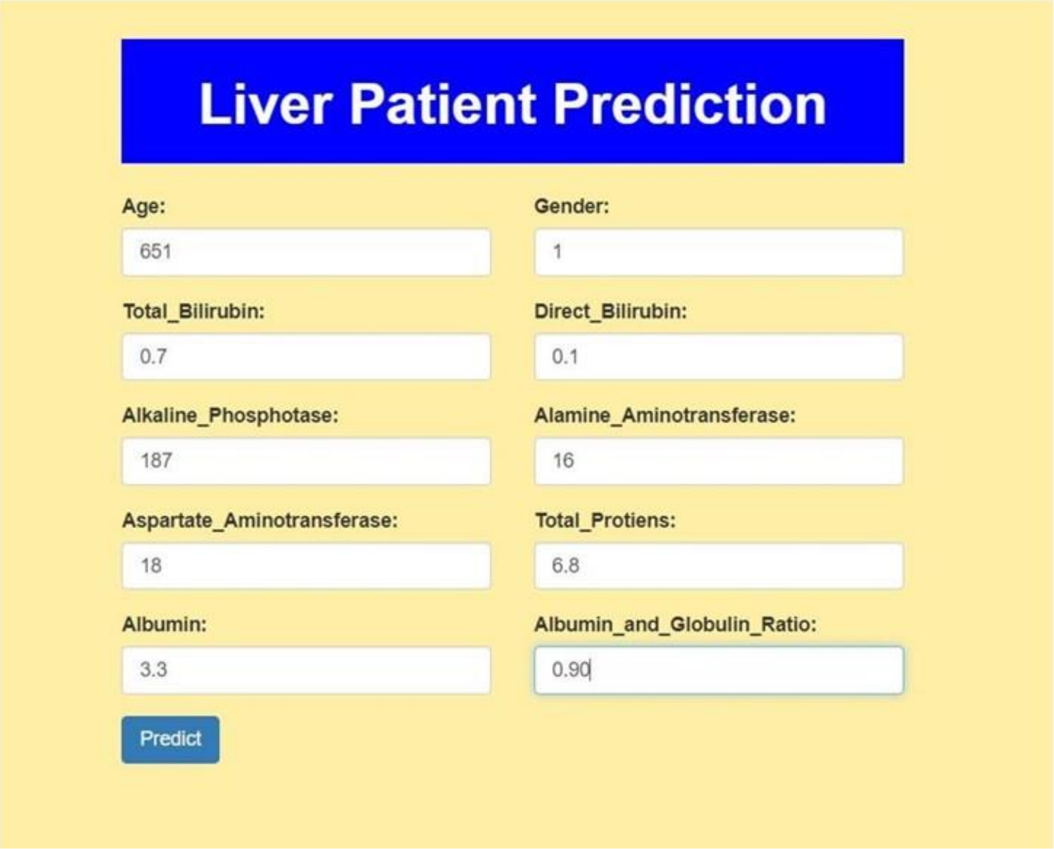
PCA.HTML

HOME PAGE



Introduction

Liver diseases avert the normal function of the liver. Mainly due to the large amount of alcohol consumption, liver disease arises. Early prediction of liver disease using classification algorithms is an efficacious task that can help the doctors to diagnose the disease within a short duration of time. Discovering the existence of liver disease at an early stage is a complex task for the doctors. The main objective of this paper is to analyze the parameters of various classification algorithms and compare their predictive accuracies to find the best classifier for determining liver disease. This paper focuses on the related works of various authors on liver disease such that algorithms were implemented using Weka tool that is a machine learning software written in Java. Various attributes that are essential in the prediction of liver disease were examined and the dataset of liver patients was also evaluated. This paper compares various classification algorithms such as Random Forest, Logistic Regression

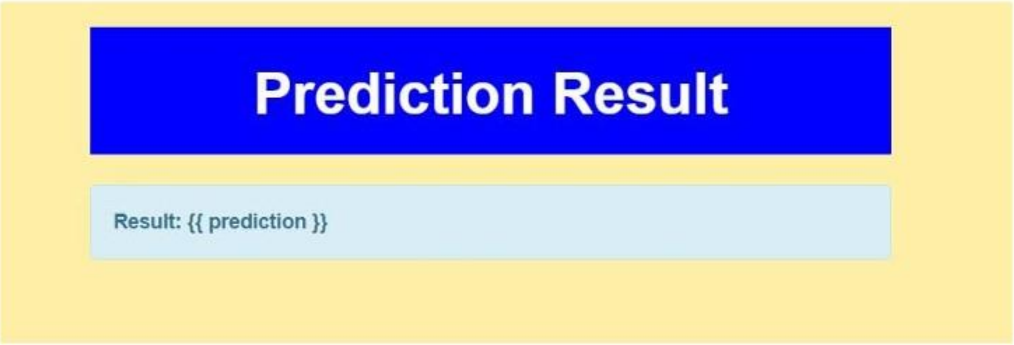


Liver Patient Prediction

Age:	Gender:
651	1
Total_Bilirubin:	Direct_Bilirubin:
0.7	0.1
Alkaline_Phosphotase:	Alamine_Aminotransferase:
187	16
Aspartate_Aminotransferase:	Total_Protiens:
18	6.8
Albumin:	Albumin_and_Globulin_Ratio:
3.3	0.90

OUTPUT PAGE

RESULT.HTML



Prediction Result

Result: {{ prediction }}

7. Advantages and disadvantages

7.ADVANTAGES AND DISADVANTAGES

Advantages:

A. It can be widely used in Medical Field to predict liver disease.

B. It would reduce the burden on doctors and medical staff leading to a more accurate treatment of the patients.

Disadvantages:

A. The cost of misclassification of a single patient can be very high.

B. Even the best performing Machine Learning Algorithm 20

8.Conclusion

The main objective of this study was to provide a summary of the classification algorithms in the field of data driven predictive data for liver disease. In this study, various classification algorithms were analysed to help doctors predict liver disease early. The purpose of this study was achieved by conducting comparative research in various papers. Based on this study, Random Forest has a much higher accuracy than other algorithms and can be used continuously in predicting user-recommended liver disease.

9. FUTURE SCOPE

- Improved Accuracy: As more data becomes available and algorithms become more sophisticated, the accuracy of predicting liver disease will continue to improve. This includes both diagnosing the presence of liver disease and predicting its progression.
- Early Detection: Machine learning models can potentially detect liver disease at earlier stages than traditional methods, allowing for timely interventions and better outcomes for patients.
- Personalized Medicine: ML algorithms can analyze large datasets to identify patterns and correlations that can lead to personalized treatment plans. This could involve tailoring treatment strategies based on individual patient data, genetics, lifestyle factors, etc.
- Integration with Healthcare Systems: There is a growing trend towards integrating machine learning models into electronic health records (EHRs) and clinical decision support systems. This integration can streamline diagnosis and treatment planning processes.
- Feature Importance and Interpretability: Future research can focus on improving the interpretability of machine learning models used for liver disease prediction. Understanding which features (e.g., biomarkers, genetic markers) contribute most to predictions can enhance medical understanding and guide further research.
- Handling Imbalanced Data: Imbalanced datasets (where one class is more prevalent than another) are common in medical data. Future research can explore techniques to handle

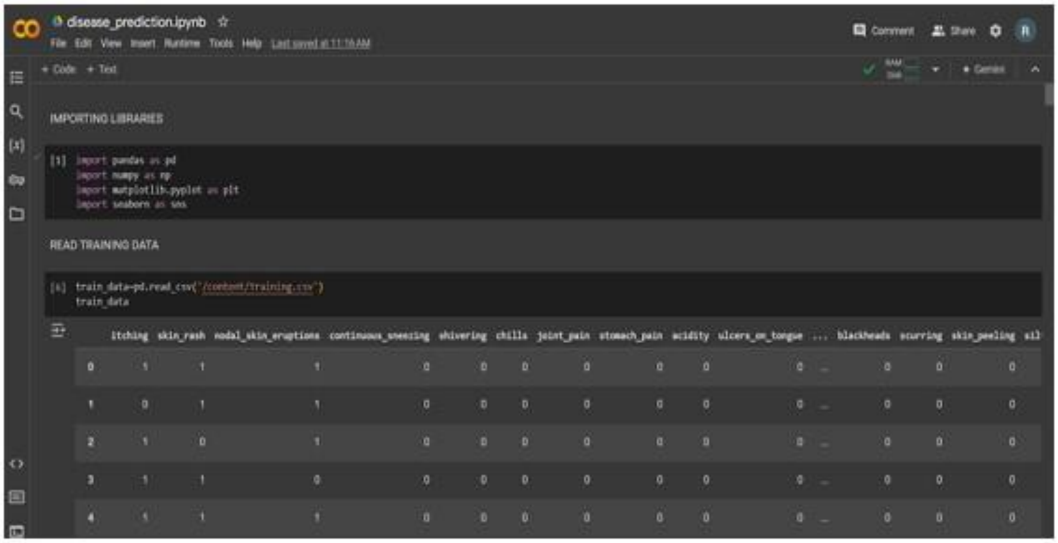
imbalanced data effectively, ensuring that models are robust and reliable.

- Telemedicine and Remote Monitoring: ML models can support telemedicine initiatives by enabling remote monitoring of liver disease patients. This can include predicting exacerbations, monitoring treatment effectiveness, and providing real-time alerts for healthcare providers.
- Drug Discovery and Development: Machine learning can aid in identifying potential drug targets and predicting the efficacy of treatments for liver disease. This can accelerate the drug discovery process and improve outcomes for patients with liver diseases.
- Ethical Considerations: As with any AI application in healthcare, there will be ongoing discussions about ethical considerations such as patient privacy, consent, and the responsible use of AI in medical decision-making.
- Global Health Impact: Machine learning can potentially have a significant impact on global health by improving access to diagnostic tools and personalized treatment options, particularly in underserved areas where access to healthcare is limited.

10.Appendix

10.1. Source Code

Code Snippets



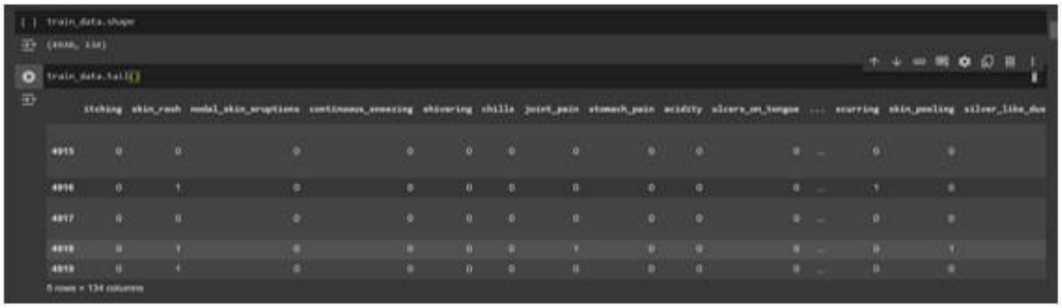
IMPORTING LIBRARIES

```
[1] import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
```

READ TRAINING DATA

```
[2] train_data=pd.read_csv('/content/training.csv')
    train_data
```

```
   itching  skin_rash  nodal_skin_eruptions  continuous_sneezing  shivering  chills  joint_pain  stomach_pain  acidity  ulcers_on_tongue  ...  blackheads  scurring  skin_peeling  sil
0        1          1                     1                    0          0       0           0             0        0                 0  ...           0         0             0
1        0          1                     1                    0          0       0           0             0        0                 0  ...           0         0             0
2        1          0                     1                    0          0       0           0             0        0                 0  ...           0         0             0
3        1          1                     0                    0          0       0           0             0        0                 0  ...           0         0             0
4        1          1                     1                    0          0       0           0             0        0                 0  ...           0         0             0
```

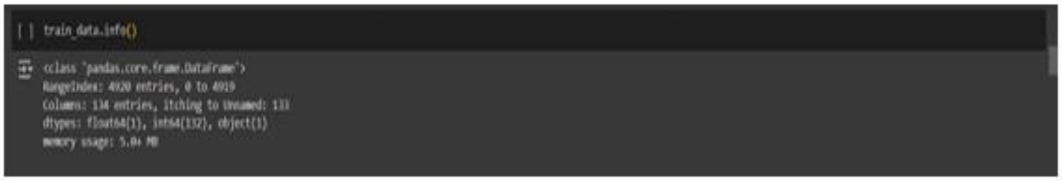


```
[ ] train_data.shape
```

```
(4920, 134)
```

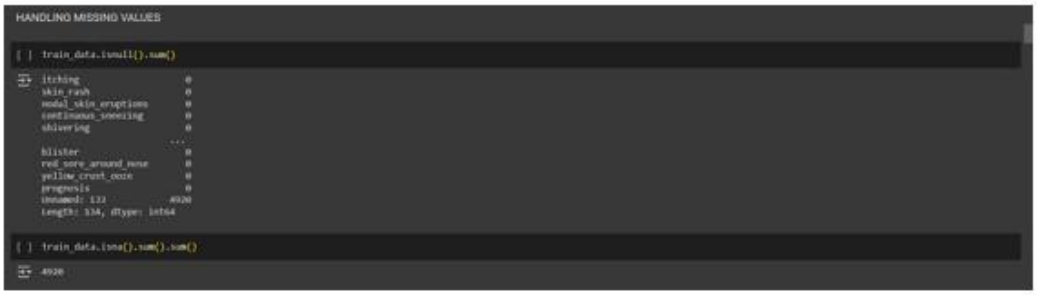
```
train_data.tail()
```

```
      itching  skin_rash  nodal_skin_eruptions  continuous_sneezing  shivering  chills  joint_pain  stomach_pain  acidity  ulcers_on_tongue  ...  scurring  skin_peeling  silver_like_dus
4915        0          0                     0                    0          0       0           0             0        0                 0  ...         0             0
4916        0          1                     0                    0          0       0           0             0        0                 0  ...         1             0
4917        0          0                     0                    0          0       0           0             0        0                 0  ...         0             0
4918        0          1                     0                    0          0       0           1             0        0                 0  ...         0             1
4919        0          1                     0                    0          0       0           0             0        0                 0  ...         0             0
5 rows × 134 columns
```



```
[ ] train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4920 entries, 0 to 4919
Columns: 134 entries, itching to Unnamed: 133
dtypes: float64(1), int64(132), object(1)
memory usage: 5.0+ MB
```



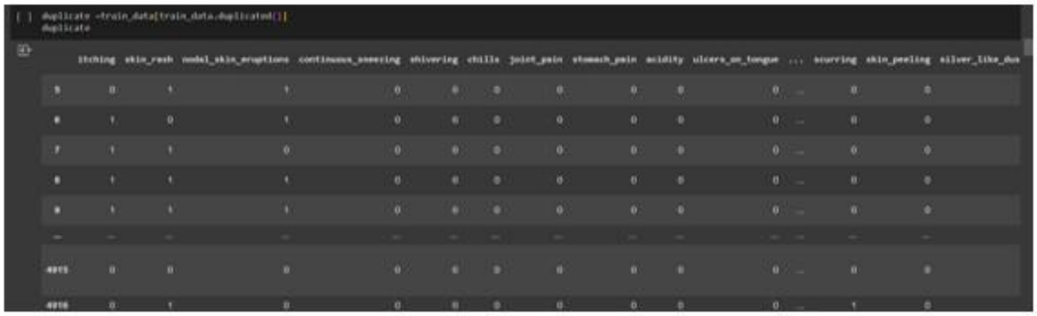
HANDLING MISSING VALUES

```
train_data.isnull().sum()
```

```
itching                   0
skin_rash                 0
nodal_skin_eruptions      0
continuous_sneezing       0
shivering                 0
                        ...
blister                   0
red_sore_around_nose      0
yellow_crust_ooze         0
prognosis                 0
Unnamed: 133           4920
Length: 134, dtype: int64
```

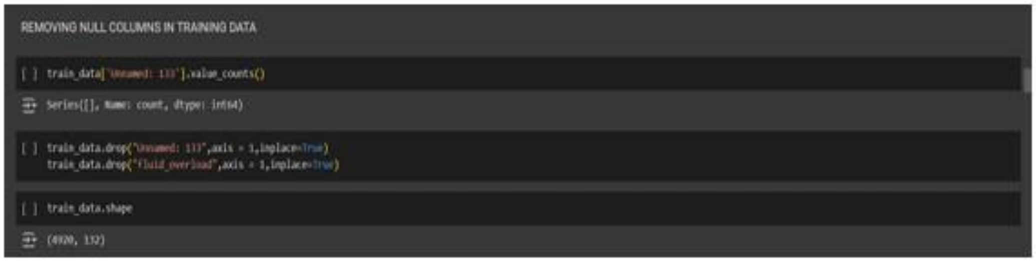
```
train_data.isna().sum().sum()
```

```
4920
```



```
duplicate =train_data[train_data.duplicated()]
duplicate
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	scurring	skin_peeling	silver_like_dus
5	0	1	1	0	0	0	0	0	0	0	...	0	0	
6	1	0	1	0	0	0	0	0	0	0	...	0	0	
7	1	1	0	0	0	0	0	0	0	0	...	0	0	
8	1	1	1	0	0	0	0	0	0	0	...	0	0	
9	1	1	1	0	0	0	0	0	0	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4915	0	0	0	0	0	0	0	0	0	0	...	0	0	
4916	0	1	0	0	0	0	0	0	0	0	...	1	0	



REMOVING NULL COLUMNS IN TRAINING DATA

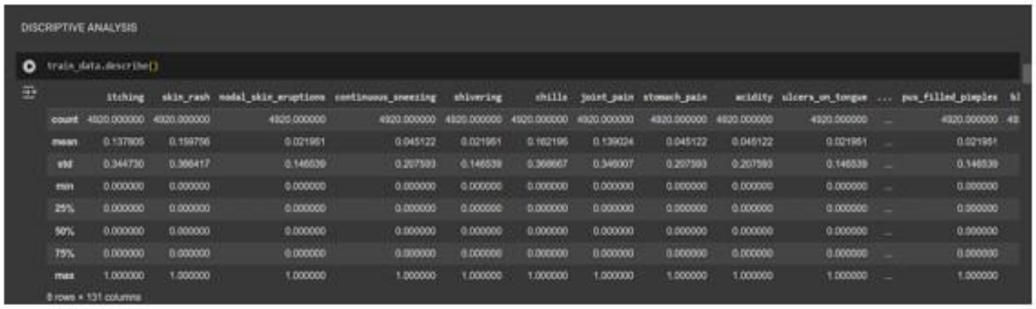
```
train_data['Unnamed: 133'].value_counts()
```

```
Series([], Name: count, dtype: int64)
```

```
train_data.drop("Unnamed: 133",axis = 1,inplace=True)
train_data.drop("fluid_overload",axis = 1,inplace=True)
```

```
train_data.shape
```

```
(4920, 132)
```

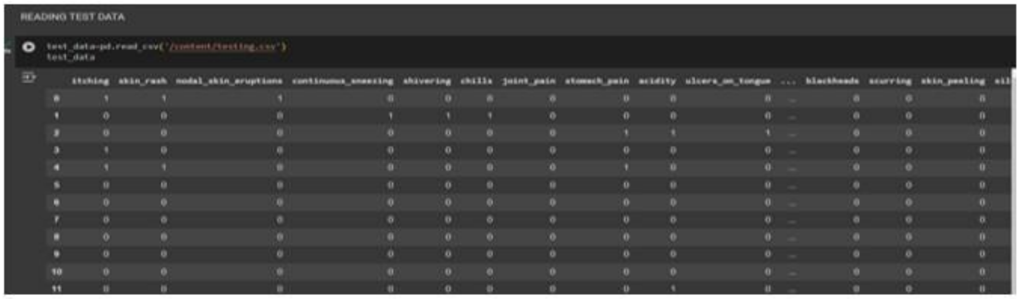


DISCRIPTIVE ANALYSIS

```
train_data.describe()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	pus_filled_pimples	bl
count	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	4920.000000	...	4920.000000	49
mean	0.137805	0.159756	0.021951	0.045122	0.021951	0.162195	0.139024	0.045122	0.045122	0.021951	...	0.021951	
std	0.344730	0.366417	0.146539	0.207593	0.146539	0.368667	0.346007	0.207593	0.207593	0.146539	...	0.146539	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	

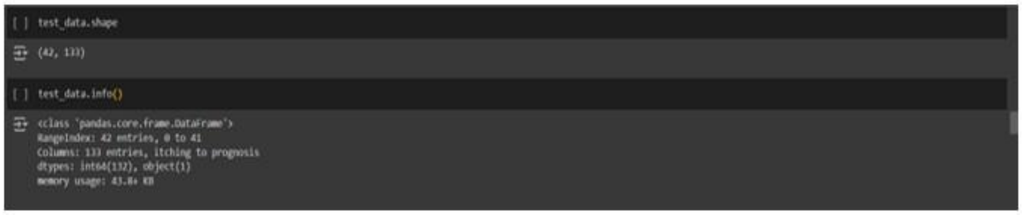
8 rows × 131 columns



READING TEST DATA

```
test_data=pd.read_csv('/content/testing.csv')
test_data
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	blackheads	scurring	skin_peeling	sil
0	1	1	1	0	0	0	0	0	0	0	...	0	0	0	
1	0	0	0	1	1	1	0	0	0	0	...	0	0	0	
2	0	0	0	0	0	0	0	1	1	1	...	0	0	0	
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	
4	1	1	0	0	0	0	0	1	0	0	...	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	...	0	0	0	
11	0	0	0	0	0	0	0	0	1	0	...	0	0	0	

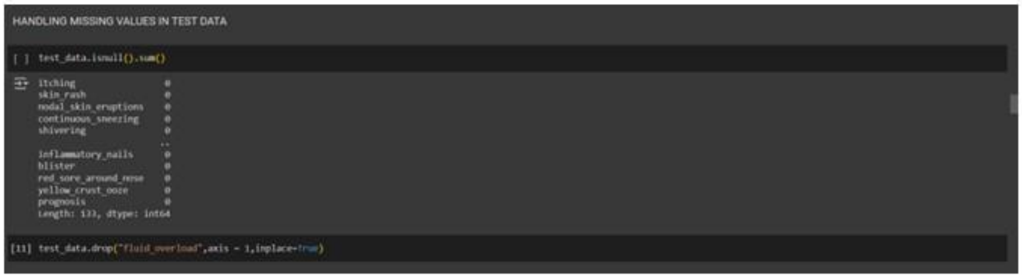


```
[ ] test_data.shape
```

```
(42, 133)
```

```
[ ] test_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42 entries, 0 to 41
Columns: 133 entries, itching to prognosis
dtypes: int64(132), object(1)
memory usage: 43.8+ KB
```

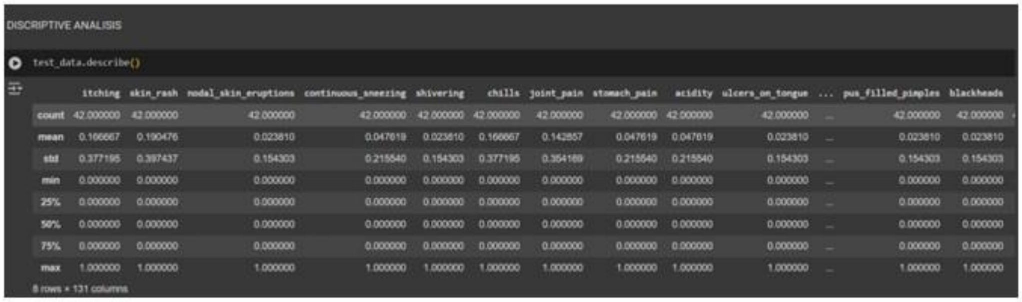


HANDLING MISSING VALUES IN TEST DATA

```
[ ] test_data.isnull().sum()
```

```
itching                 0
skin_rash               0
nodal_skin_eruptions    0
continuous_sneezing     0
shivering               0
                       ..
inflammatory_nails      0
blister                 0
red_sore_around_nose    0
yellow_crust_ooze       0
prognosis               0
Length: 133, dtype: int64
```

```
[11] test_data.drop("fluid_overload",axis = 1,inplace=True)
```

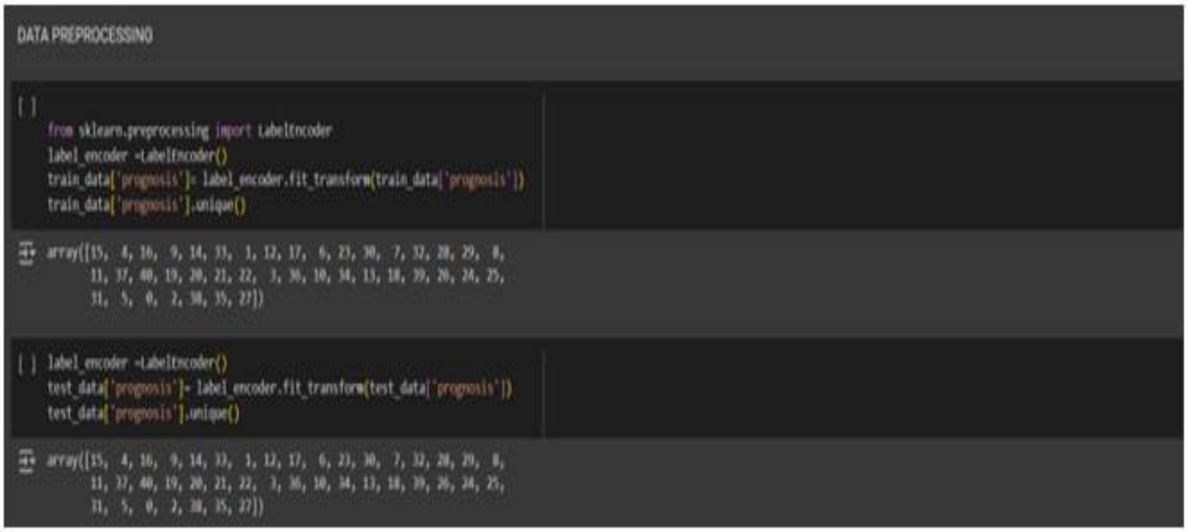


DISCRIPTIVE ANALISIS

```
test_data.describe()
```

	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	...	pus_filled_pimples	blackheads
count	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	42.000000	...	42.000000	42.000000
mean	0.166667	0.190476	0.023810	0.047619	0.023810	0.166667	0.142857	0.047619	0.047619	0.023810	...	0.023810	0.023810
std	0.377195	0.397437	0.154303	0.215540	0.154303	0.377195	0.354169	0.215540	0.215540	0.154303	...	0.154303	0.154303
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000

8 rows × 131 columns



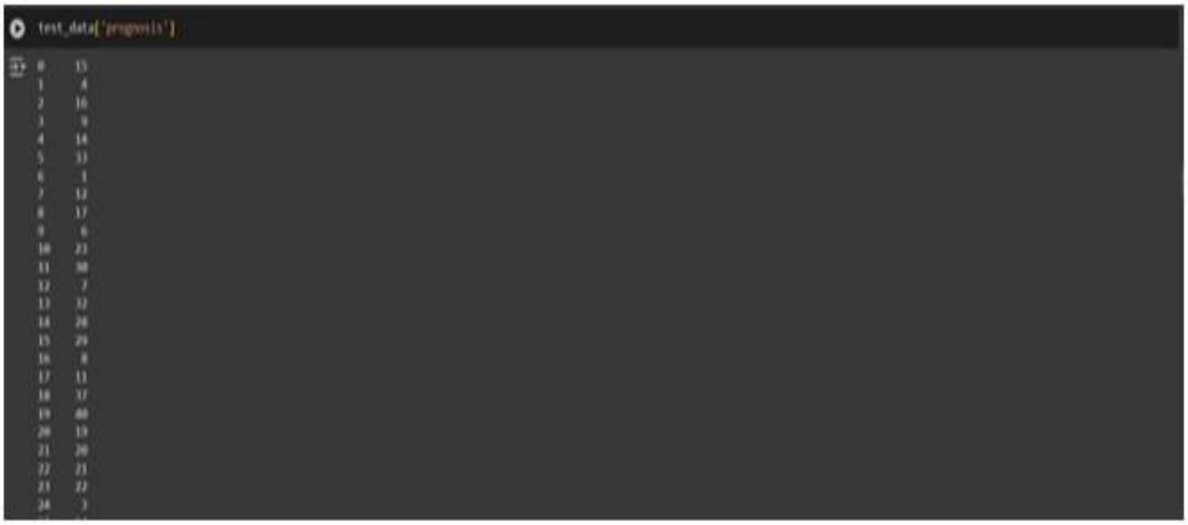
DATA PREPROCESSING

```
[ ]
    from sklearn.preprocessing import LabelEncoder
    label_encoder =LabelEncoder()
    train_data['prognosis']= label_encoder.fit_transform(train_data['prognosis'])
    train_data['prognosis'].unique()
```

```
array([15,  4, 16,  9, 14, 33,  1, 12, 17,  6, 23, 30,  7, 32, 28, 29,  8,
       11, 37, 40, 19, 20, 21, 22,  3, 36, 10, 34, 13, 18, 39, 26, 24, 25,
       31,  5,  0,  2, 38, 35, 27])
```

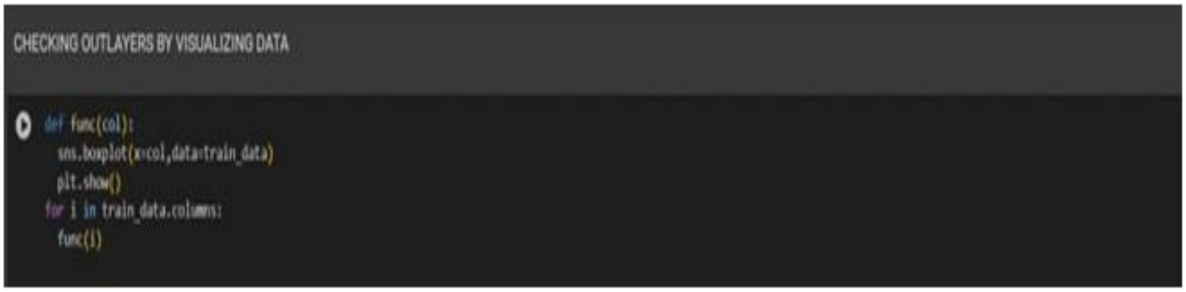
```
[ ] label_encoder =LabelEncoder()
    test_data['prognosis']= label_encoder.fit_transform(test_data['prognosis'])
    test_data['prognosis'].unique()
```

```
array([15,  4, 16,  9, 14, 33,  1, 12, 17,  6, 23, 30,  7, 32, 28, 29,  8,
       11, 37, 40, 19, 20, 21, 22,  3, 36, 10, 34, 13, 18, 39, 26, 24, 25,
       31,  5,  0,  2, 38, 35, 27])
```



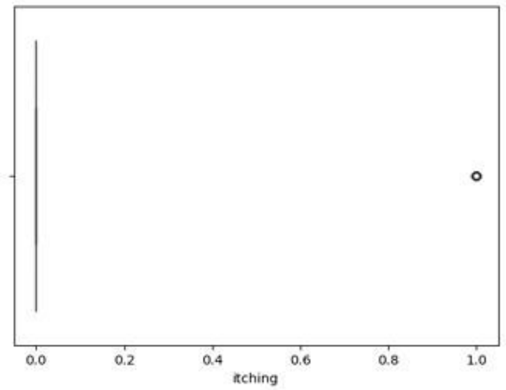
```
test_data['prognosis']
```

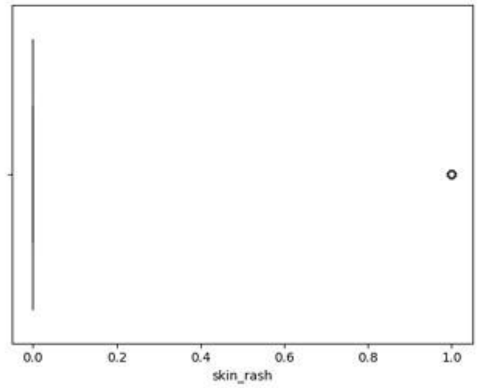
```
0     15
1      4
2     16
3      9
4     14
5     33
6      1
7     12
8     17
9      6
10    23
11    30
12     7
13    32
14    28
15    29
16     8
17    11
18    37
19    40
20    19
21    20
22    21
23    22
24     3
```



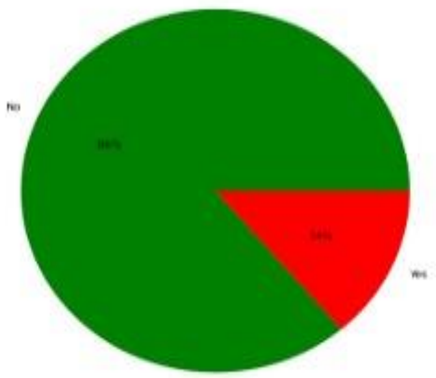
CHECKING OUTLAYERS BY VISUALIZING DATA

```
def func(col):
  sns.boxplot(x=col,data=train_data)
  plt.show()
for i in train_data.columns:
  func(i)
```

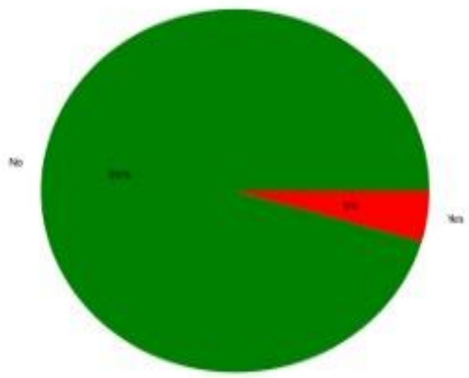


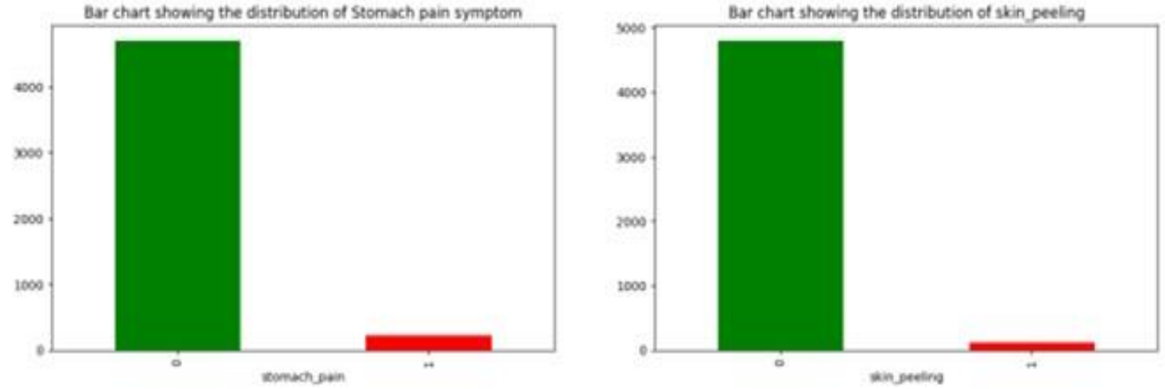


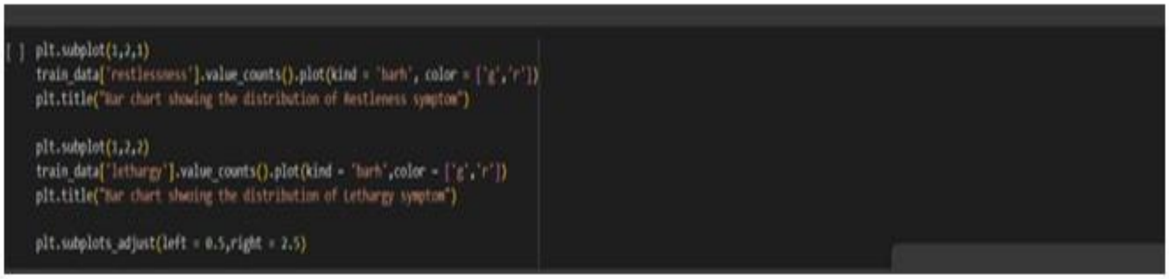
Pie chart showing the distribution of Itching symptom into number of Yes/No



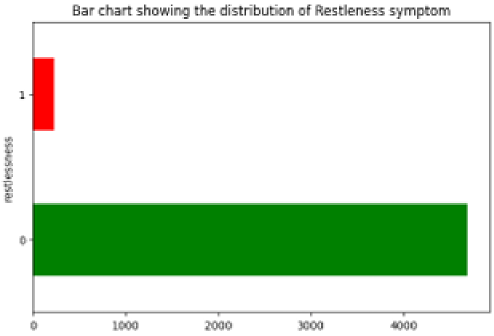
Pie Chart showing the distribution of Continuous Sneezing symptom into number of Yes/No

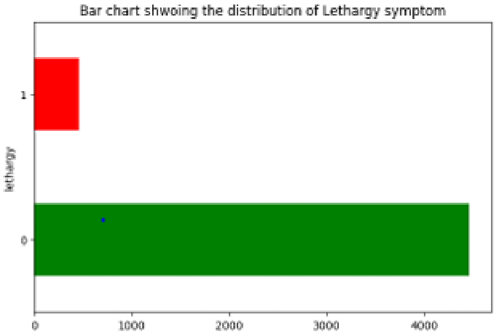


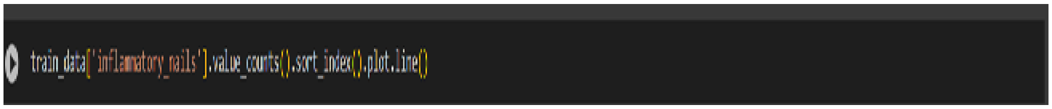




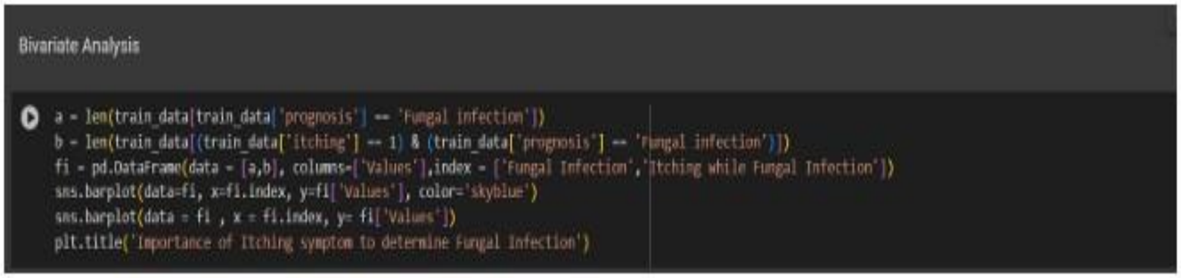
```
[ ] plt.subplot(1,2,1)
    train_data['restlessness'].value_counts().plot(kind = 'barh', color = ['g','r'])
    plt.title("Bar chart showing the distribution of Restleness symptom")

    plt.subplot(1,2,2)
    train_data['lethargy'].value_counts().plot(kind = 'barh',color = ['g','r'])
    plt.title("Bar chart showing the distribution of Lethargy symptom")

    plt.subplots_adjust(left = 0.5,right = 2.5)
```





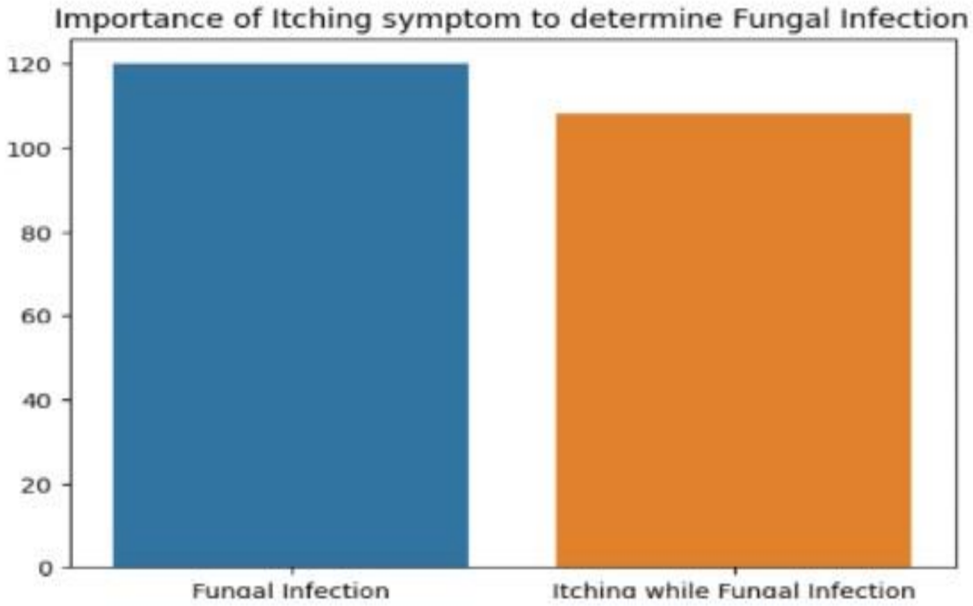


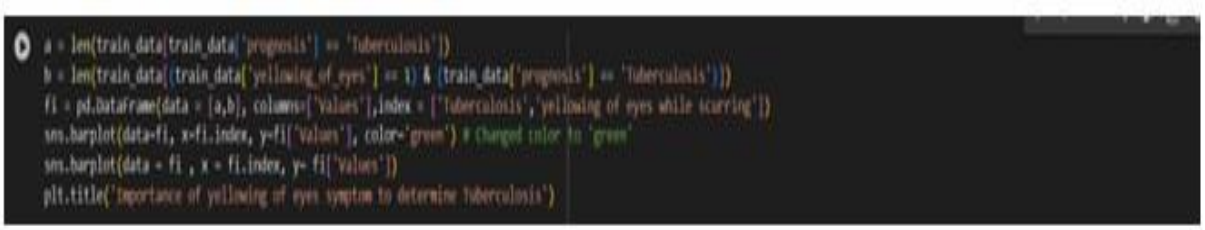
```
train_data['inflamatory_nails'].value_counts().sort_index().plot.line()
```



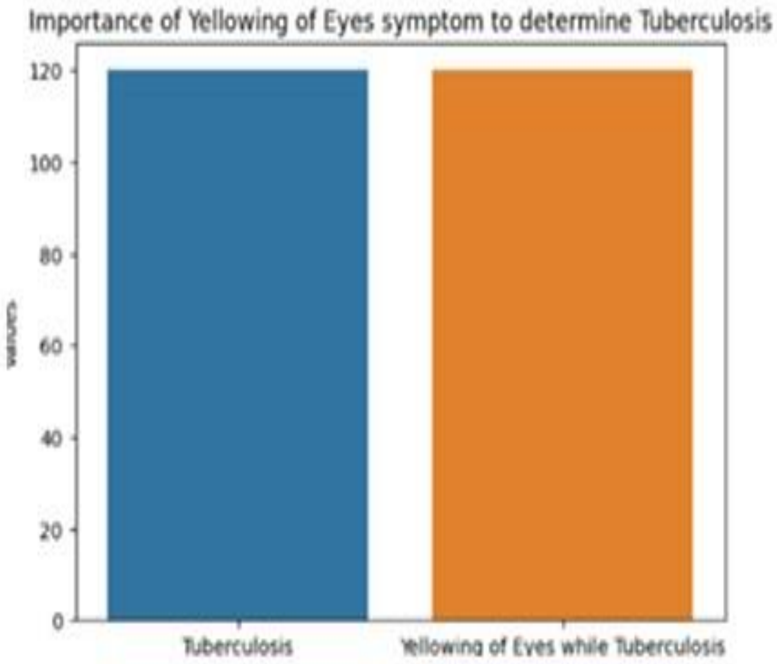
Bivariate Analysis

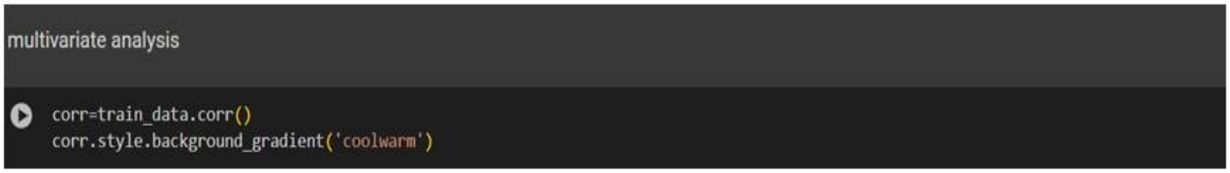
```
a = len(train_data[train_data['prognosis'] == 'Fungal infection'])
b = len(train_data[(train_data['itching'] == 1) & (train_data['prognosis'] == 'Fungal infection')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Fungal Infection','Itching while Fungal Infection'])
sns.barplot(data=fi, x=fi.index, y=fi['Values'], color='skyblue')
sns.barplot(data = fi , x = fi.index, y= fi['Values'])
plt.title('Importance of Itching symptom to determine Fungal Infection')
```





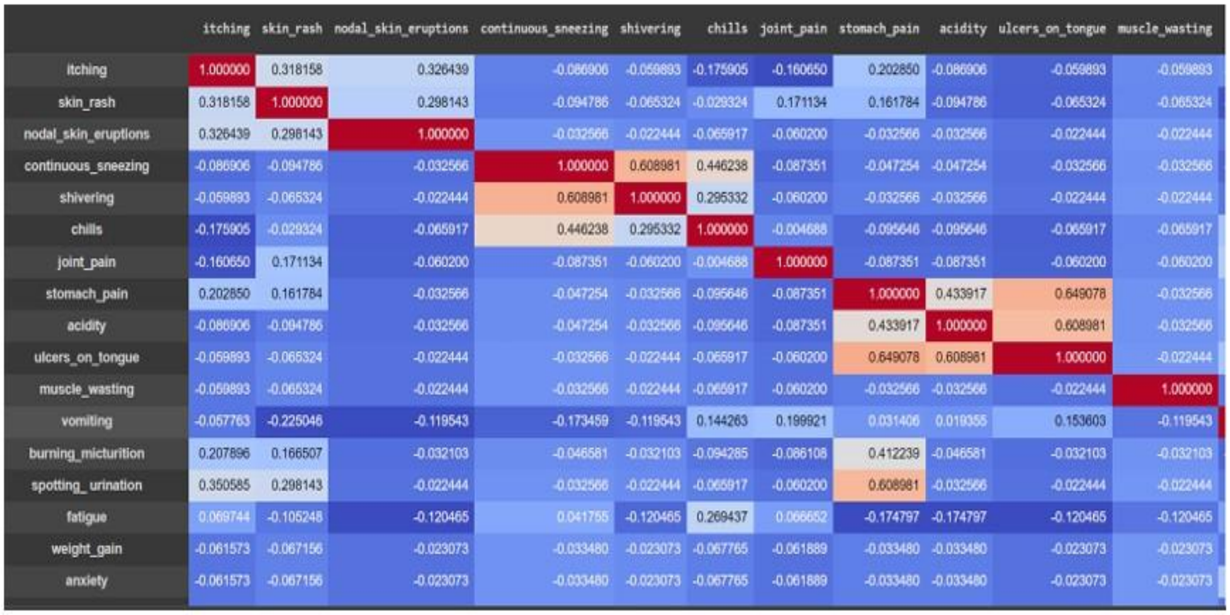
```
a = len(train_data[train_data['prognosis'] == 'Tuberculosis'])
b = len(train_data[(train_data['yellowing_of_eyes'] == 1) & (train_data['prognosis'] == 'Tuberculosis')])
fi = pd.DataFrame(data = [a,b], columns=['Values'],index = ['Tuberculosis','yellowing of eyes while scurring'])
sns.barplot(data=fi, x=fi.index, y=fi['Values'], color='green') # Changed color to 'green'
sns.barplot(data = fi , x = fi.index, y= fi['Values'])
plt.title('Importance of yellowing of eyes symptom to determine Tuberculosis')
```



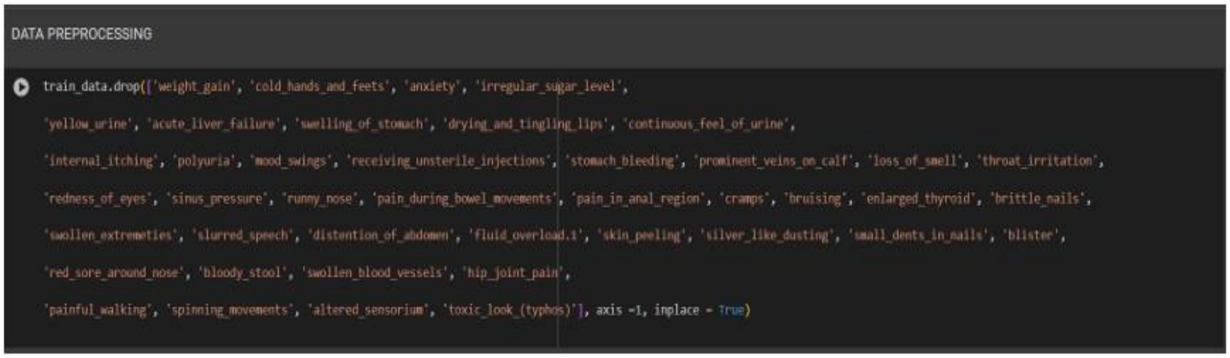


multivariate analysis

```
corr=train_data.corr()
corr.style.background_gradient('coolwarm')
```

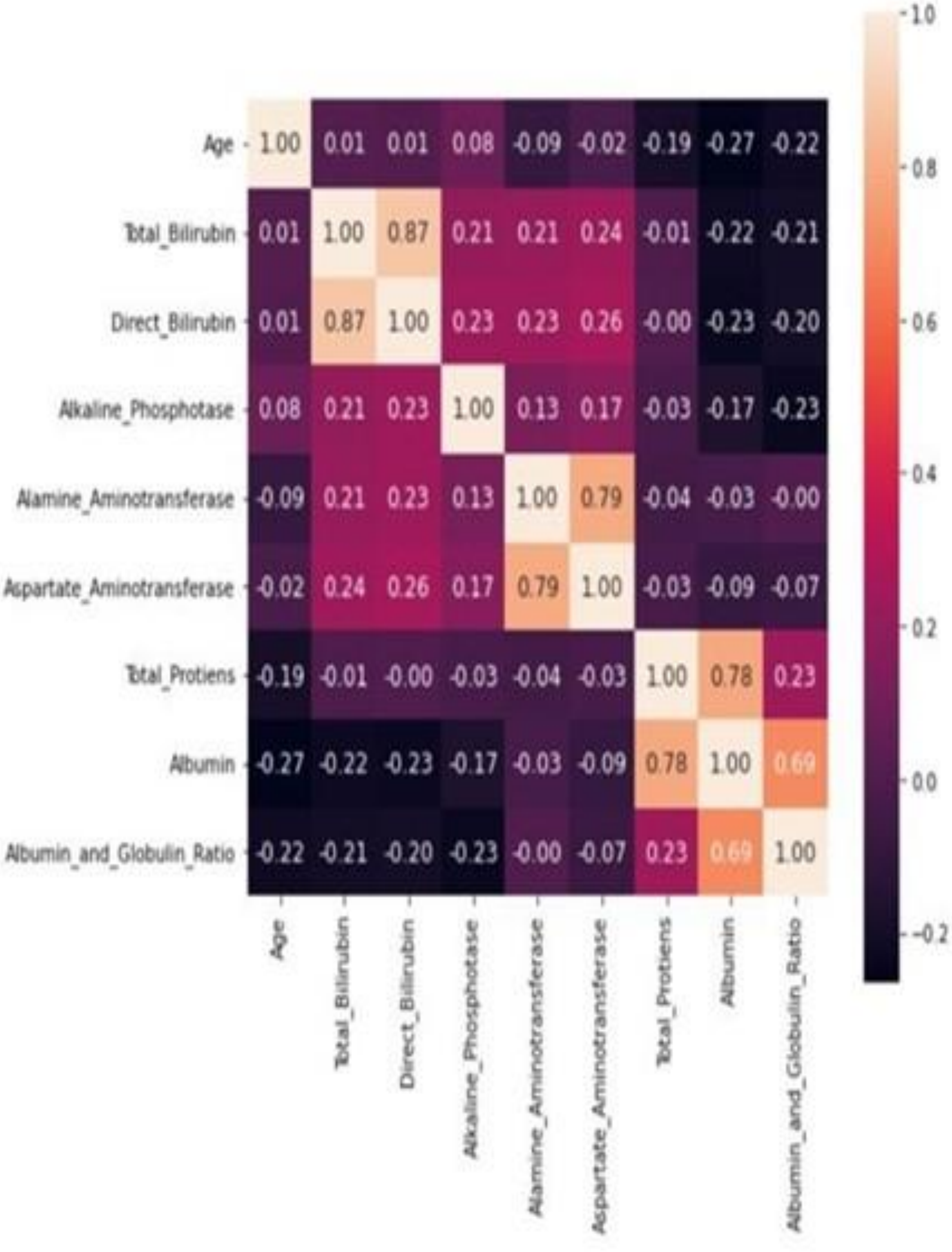


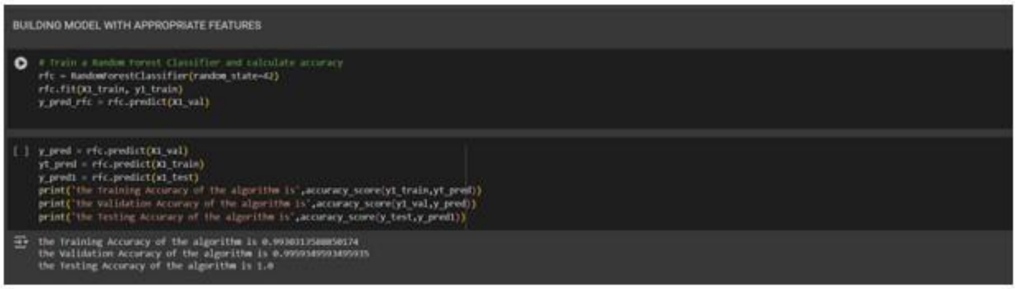
	itching	skin_rash	nodal_skin_eruptions	continuous_sneezing	shivering	chills	joint_pain	stomach_pain	acidity	ulcers_on_tongue	muscle_wasting
itching	1.000000	0.318158	0.326439	-0.086906	-0.059893	-0.175905	-0.160650	0.202850	-0.086906	-0.059893	-0.059893
skin_rash	0.318158	1.000000	0.298143	-0.094786	-0.065324	-0.029324	0.171134	0.161784	-0.094786	-0.065324	-0.065324
nodal_skin_eruptions	0.326439	0.298143	1.000000	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	-0.022444	-0.022444
continuous_sneezing	-0.086906	-0.094786	-0.032566	1.000000	0.608981	0.446238	-0.087351	-0.047254	-0.047254	-0.032566	-0.032566
shivering	-0.059893	-0.065324	-0.022444	0.608981	1.000000	0.295332	-0.060200	-0.032566	-0.032566	-0.022444	-0.022444
chills	-0.175905	-0.029324	-0.065917	0.446238	0.295332	1.000000	-0.004688	-0.095646	-0.095646	-0.065917	-0.065917
joint_pain	-0.160650	0.171134	-0.060200	-0.087351	-0.060200	-0.004688	1.000000	-0.087351	-0.087351	-0.060200	-0.060200
stomach_pain	0.202850	0.161784	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	1.000000	0.433917	0.649078	-0.032566
acidity	-0.086906	-0.094786	-0.032566	-0.047254	-0.032566	-0.095646	-0.087351	0.433917	1.000000	0.608981	-0.032566
ulcers_on_tongue	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.649078	0.608981	1.000000	-0.022444
muscle_wasting	-0.059893	-0.065324	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	-0.032566	-0.032566	-0.022444	1.000000
vomiting	-0.057763	-0.225046	-0.119543	-0.173459	-0.119543	0.144263	0.199921	0.031406	0.019355	0.153603	-0.119543
burning_micturition	0.207896	0.166507	-0.032103	-0.046581	-0.032103	-0.094285	-0.086108	0.412239	-0.046581	-0.032103	-0.032103
spotting_ urination	0.350585	0.298143	-0.022444	-0.032566	-0.022444	-0.065917	-0.060200	0.608981	-0.032566	-0.022444	-0.022444
fatigue	0.069744	-0.105248	-0.120465	0.041755	-0.120465	0.269437	0.066652	-0.174797	-0.174797	-0.120465	-0.120465
weight_gain	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	-0.023073	-0.023073
anxiety	-0.061573	-0.067156	-0.023073	-0.033480	-0.023073	-0.067765	-0.061889	-0.033480	-0.033480	-0.023073	-0.023073



DATA PREPROCESSING

```
train_data.drop(['weight_gain', 'cold_hands_and_feets', 'anxiety', 'irregular_sugar_level',

'yellow_urine', 'acute_liver_failure', 'swelling_of_stomach', 'drying_and_tingling_lips', 'continuous_feel_of_urine',

'internal_itching', 'polyuria', 'mood_swings', 'receiving_unsterile_injections', 'stomach_bleeding', 'prominent_veins_on_calf', 'loss_of_smell', 'throat_irritation',

'redness_of_eyes', 'sinus_pressure', 'runny_nose', 'pain_during_bowel_movements', 'pain_in_anal_region', 'cramps', 'bruising', 'enlarged_thyroid', 'brittle_nails',

'swollen_extremeties', 'slurred_speech', 'distention_of_abdomen', 'fluid_overload.1', 'skin_peeling', 'silver_like_dusting', 'small_dents_in_nails', 'blister',

'red_sore_around_nose', 'bloody_stool', 'swollen_blood_vessels', 'hip_joint_pain',

'painful_walking', 'spinning_movements', 'altered_sensorium', 'toxic_look_(typhos)'], axis =1, inplace = True)
```


Age 1.00 0.01 0.01 0.08 -0.09 -0.02 -0.19 -0.27 -0.22
Total_Bilirubin 0.01 1.00 0.87 0.21 0.21 0.24 -0.01 -0.22 -0.21
Direct_Bilirubin 0.01 0.87 1.00 0.23 0.23 0.26 -0.00 -0.23 -0.20
Alkaline_Phosphotase 0.08 0.21 0.23 1.00 0.13 0.17 -0.03 -0.17 -0.23
Alamine_Aminotransferase -0.09 0.21 0.23 0.13 1.00 0.79 -0.04 -0.03 -0.00
Aspartate_Aminotransferase -0.02 0.24 0.26 0.17 0.79 1.00 -0.03 -0.09 -0.07
Total_Protiens -0.19 -0.01 -0.00 -0.03 -0.04 -0.03 1.00 0.78 0.23
Albumin -0.27 -0.22 -0.23 -0.17 -0.03 -0.09 0.78 1.00 0.69
Albumin_and_Globulin_Ratio -0.22 -0.21 -0.20 -0.23 -0.00 -0.07 0.23 0.69 1.00
Age
Total_Bilirubin
Direct_Bilirubin
Alkaline_Phosphotase
Alamine_Aminotransferase
Aspartate_Aminotransferase
Total_Protiens
Albumin
Albumin_and_Globulin_Ratio
1.0
0.8
0.6
0.4
0.2
0.0
-0.2

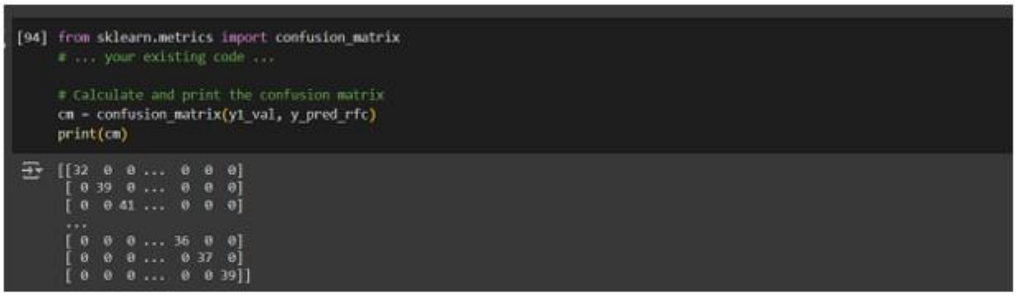


BUILDING MODEL WITH APPROPRIATE FEATURES

```
# Train a Random Forest Classifier and calculate accuracy
rfc = RandomForestClassifier(random_state=42)
rfc.fit(X1_train, y1_train)
y_pred_rfc = rfc.predict(X1_val)
```

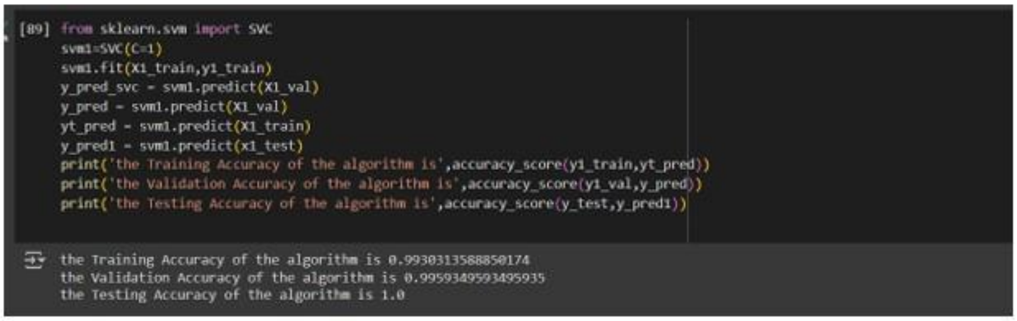
```
[ ] y_pred = rfc.predict(X1_val)
    yt_pred = rfc.predict(X1_train)
    y_pred1 = rfc.predict(x1_test)
    print('the Training Accuracy of the algorithm is',accuracy_score(y1_train,yt_pred))
    print('the Validation Accuracy of the algorithm is',accuracy_score(y1_val,y_pred))
    print('the Testing Accuracy of the algorithm is',accuracy_score(y_test,y_pred1))
```

```
the Training Accuracy of the algorithm is 0.9930313588850174
the Validation Accuracy of the algorithm is 0.9959349593495935
the Testing Accuracy of the algorithm is 1.0
```



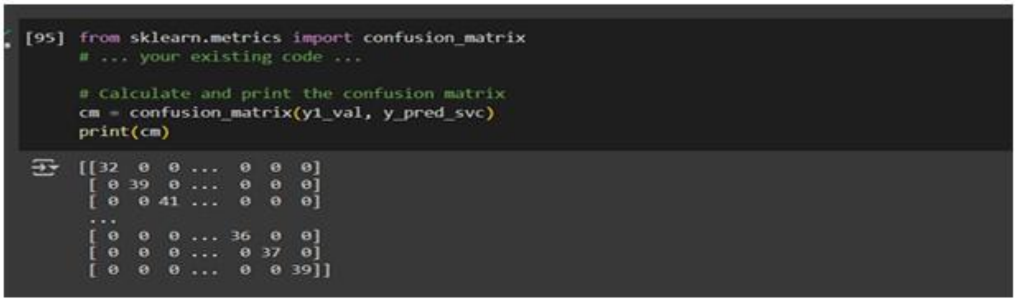
```
[94] from sklearn.metrics import confusion_matrix
     # ... your existing code ...

     # Calculate and print the confusion matrix
     cm = confusion_matrix(y1_val, y_pred_rfc)
     print(cm)
```

```
[[32  0  0 ...  0  0  0]
 [ 0 39  0 ...  0  0  0]
 [ 0  0 41 ...  0  0  0]
 ...
 [ 0  0  0 ... 36  0  0]
 [ 0  0  0 ...  0 37  0]
 [ 0  0  0 ...  0  0 39]]
```



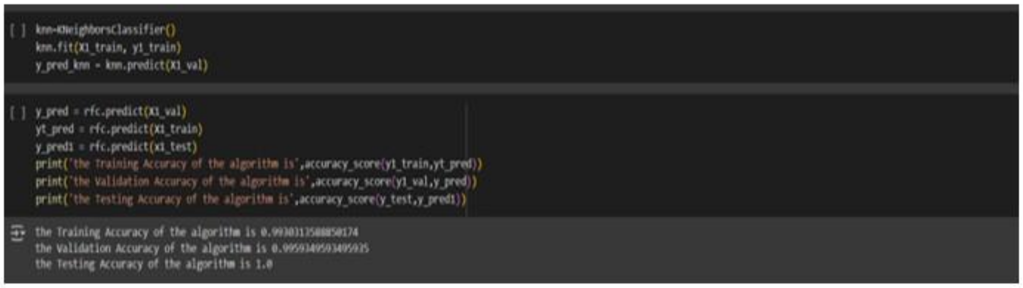
```
[89] from sklearn.svm import SVC
     svm1=SVC(C=1)
     svm1.fit(X1_train,y1_train)
     y_pred_svc = svm1.predict(X1_val)
     y_pred = svm1.predict(X1_val)
     yt_pred = svm1.predict(X1_train)
     y_pred1 = svm1.predict(x1_test)
     print('the Training Accuracy of the algorithm is',accuracy_score(y1_train,yt_pred))
     print('the Validation Accuracy of the algorithm is',accuracy_score(y1_val,y_pred))
     print('the Testing Accuracy of the algorithm is',accuracy_score(y_test,y_pred1))
```

```
the Training Accuracy of the algorithm is 0.9930313588850174
the Validation Accuracy of the algorithm is 0.9959349593495935
the Testing Accuracy of the algorithm is 1.0
```



```
[95] from sklearn.metrics import confusion_matrix
     # ... your existing code ...

     # Calculate and print the confusion matrix
     cm = confusion_matrix(y1_val, y_pred_svc)
     print(cm)
```

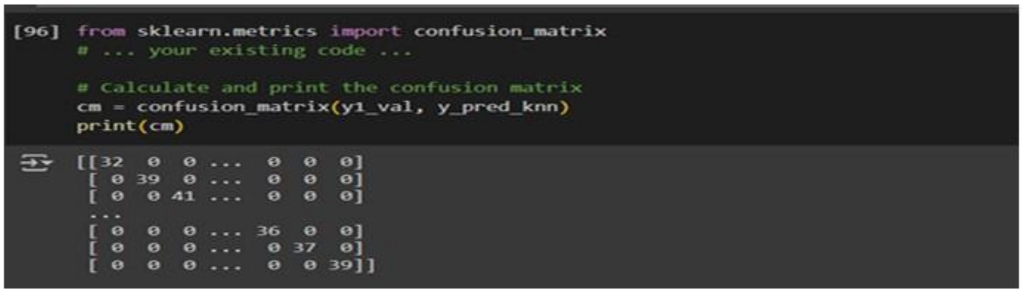
```
[[32  0  0 ...  0  0  0]
 [ 0 39  0 ...  0  0  0]
 [ 0  0 41 ...  0  0  0]
 ...
 [ 0  0  0 ... 36  0  0]
 [ 0  0  0 ...  0 37  0]
 [ 0  0  0 ...  0  0 39]]
```



```
[ ] knn=KNeighborsClassifier()
    knn.fit(X1_train, y1_train)
    y_pred_knn = knn.predict(X1_val)
```

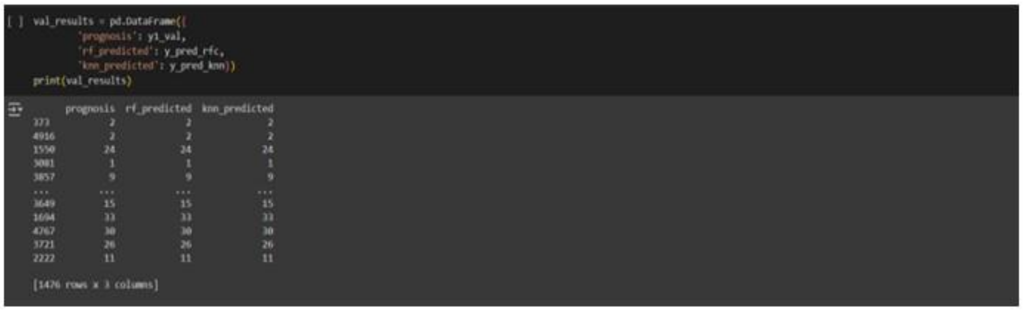
```
[ ] y_pred = rfc.predict(X1_val)
    yt_pred = rfc.predict(X1_train)
    y_pred1 = rfc.predict(x1_test)
    print('the Training Accuracy of the algorithm is',accuracy_score(y1_train,yt_pred))
    print('the Validation Accuracy of the algorithm is',accuracy_score(y1_val,y_pred))
    print('the Testing Accuracy of the algorithm is',accuracy_score(y_test,y_pred1))
```

```
the Training Accuracy of the algorithm is 0.9930311358850174
the Validation Accuracy of the algorithm is 0.9959349593495935
the Testing Accuracy of the algorithm is 1.0
```



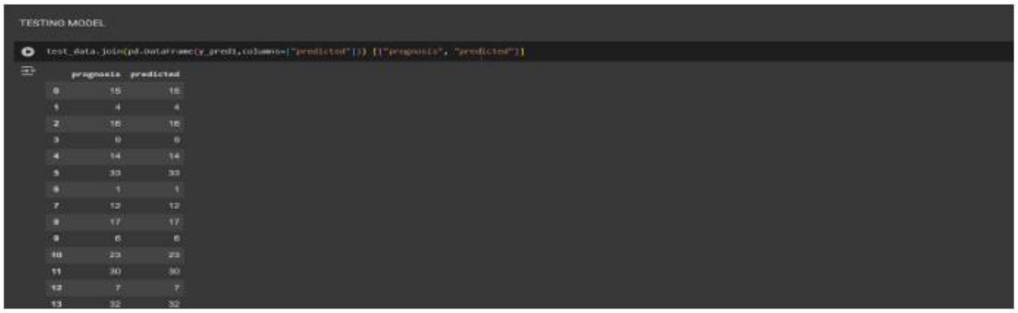
```
[96] from sklearn.metrics import confusion_matrix
     # ... your existing code ...

     # Calculate and print the confusion matrix
     cm = confusion_matrix(y1_val, y_pred_knn)
     print(cm)
```

```
[[32  0  0 ...  0  0  0]
 [ 0 39  0 ...  0  0  0]
 [ 0  0 41 ...  0  0  0]
 ...
 [ 0  0  0 ... 36  0  0]
 [ 0  0  0 ...  0 37  0]
 [ 0  0  0 ...  0  0 39]]
```



```
[ ] val_results = pd.DataFrame({
            'prognosis': y1_val,
            'rf_predicted': y_pred_rfc,
            'knn_predicted': y_pred_knn})
    print(val_results)
```

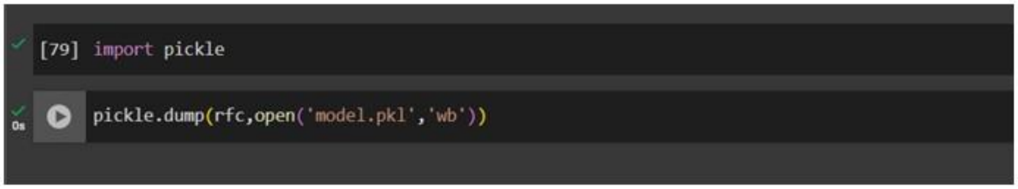
```
      prognosis  rf_predicted  knn_predicted
373           2             2              2
4916          2             2              2
1550         24            24             24
5081          1             1              1
3857          9             9              9
...         ...           ...            ...
3649         15            15             15
1694         33            33             33
4767         30            30             30
3721         26            26             26
2222         11            11             11

[1476 rows x 3 columns]
```



TESTING MODEL

```
test_data.join(pd.DataFrame(y_pred1,columns=["predicted"])) [["prognosis", "predicted"]]
```

```
    prognosis  predicted
0          15         15
1           4          4
2          16         16
3           0          0
4          14         14
5          33         33
6           1          1
7          12         12
8          17         17
9           6          6
10         23         23
11         30         30
12          7          7
13         32         32
```



```
[79] import pickle
```

```
pickle.dump(rfc,open('model.pkl','wb'))
```

PCA.HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Liver Patient Analysis</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">
  <style type="text/css">
    body {
      background-color: #ffcf0059;
    }
    nav {
      background-color: #ad38c2;
      height: 60px;
    }
    .navbar-brand {
      color: white;
      font-size: 30px;
    }
    nav ul li a {
      color: white;
      font-size: 20px;
    }
  </style>
</head>
<body>

<nav class="navbar">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand">Liver Patient Analysis</a>
    </div>
    <ul class="nav navbar-nav navbar-right">
      <li><a href="#">Home</a></li>
      <li><a href="/predict">Goto Predict</a></li>
    </ul>
  </div>
</nav>

<div class="container">
  <h3>Introduction</h3>
  <p>Liver diseases avert the normal function of the liver. Mainly due
to the large amount of alcohol consumption, liver disease arises. Early
prediction of liver disease using classification algorithms is an
efficacious task that can help the doctors to diagnose the disease
within a short duration of time. Discovering the existence of liver
disease at an early stage is a complex task for the doctors. The main
objective of this paper is to analyze the parameters of various
classification algorithms and compare their predictive accuracies to
find the best classifier for determining liver disease. This paper
focuses on the related works of various authors on liver disease such
that algorithms were implemented using Weka tool that is a machine
```

```
learning software written in Java. Various attributes that are essential
in the prediction of liver disease were examined and the dataset of
liver patients was also evaluated. This paper compares various
classification algorithms such as Random Forest, Logistic
Regression</p>

</div>


</body>

</html>
```

Result.html:

```
<!DOCTYPE html>

<html>

<head>

  <title>Prediction Result</title>

  <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.
min.css">

  <style type="text/css">

    body {

      background-color: #ffcf0059;

    }

    .page-header {

      background-color: blue;

      width: 100%;

      height: auto;

      text-align: center;
      padding-top: 5px;
      color: #fff;
    }
    h1 {
      font-size: 40px;
      font-weight: bold;
    }
  </style>
</head>
<body>

  <div class="container">
    <div class="row">
      <div class="col-md-3"></div>
      <div class="col-md-6">
         <div class="page-header">
          <h1>Prediction Result</h1>
        </div>
      </div>
    </div>
  </div>

  <div class="container">
    <div class="row">
      <div class="col-md-3"></div>
      <div class="col-md-6">
```

```
        <div class="alert alert-info">
          <strong>Result: {{ prediction }}</strong>
        </div>
      </div>
    </div>
  </div>

</body>
</html>
```

App.py:

```
# import numpy as np
# from flask import Flask, request, jsonify, render_template
# import pickle

# app = Flask(__name__)
# model = pickle.load(open('indian_liver_patient.pkl', 'rb'))

# @app.route('/')
# def home():
#     return render_template('index.html')

# @app.route('/predict', methods=['POST'])
# def predict():
#     '''
#     For rendering results on HTML GUI
#     '''
#     try:
#         int_features = [int(x) for x in request.form.values()]
#         final_features = [np.array(int_features)]
#         prediction = model.predict(final_features)
#         output = round(prediction[0], 2)
#         return render_template('index.html', prediction_text='Predicted Value: $ {}'.format(output))
#     except Exception as e:
#         return str(e)


# @app.route('/predict_api', methods=['POST'])
# def predict_api():
#     '''
#     For direct API calls through request
#     '''
#     try:
#         data = request.get_json(force=True)
#         prediction = model.predict([np.array(list(data.values()))])
#         output = prediction[0]
#         return jsonify(output)
#     except Exception as e:
#         return jsonify({'error': str(e)})
# if __name__ == "__main__":
#     app.run(debug=True)


"""from flask import Flask, request, render_template
import pandas as pd
import pickle

app = Flask(__name__)

# Load the trained model
try:
    with open('indian_liver_patient.pkl', 'rb') as model_file:
        model = pickle.load(model_file)
except (EOFError, FileNotFoundError) as e:
    model = None
    print(f"Error loading model: {e}")

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        try:
            if model is None:
                raise ValueError("Model not loaded properly")

            # Get input data from form
            data = request.form.to_dict()
            data = {k: float(v) for k, v in data.items()}

            # Convert to DataFrame
            df = pd.DataFrame([data])

            # Make prediction
            prediction = model.predict(df)

            # Return result
            result = 'Liver disease' if prediction[0] == 1 else 'No liver disease'
            return render_template('result.html', result=result)

        except Exception as e:
            return str(e)
    return render_template('predict.html')

if __name__ == "__main__":
    app.run(debug=True)"""
```

```
from flask import Flask, request, render_template
import pandas as pd
import pickle

app = Flask(__name__)

# Load the model once during the app initialization
model = pickle.load(open('liver_analysis.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict')
def index():
    return render_template('predict.html')

@app.route('/data_predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        age = request.form['age']
        gender = request.form['gender']
        tb = request.form['tb']
        db = request.form['db']
        ap = request.form['ap']
        aa1 = request.form['aa1']
        aa2 = request.form['aa2']
        tp = request.form['tp']
        a = request.form['a']
        agr = request.form['agr']

        data = [[float(age), float(gender), float(tb), float(db), float(ap), float(aa1), float(aa2), float(tp), float(a), float(agr)]]

        prediction = model.predict(data)[0]
        if prediction == 1:
            result = "You have a liver disease"
        else:
            result = "You don't have a liver disease"

        return render_template('result.html', prediction=result)

if __name__ == '__main__':
    app.run(debug=True)
```

10.2 GitHub and project Demo link:

Github link: **Click Here**

Project Demo link: **Click Here**