Sloan Digital Sky Survey (SDSS) galaxy classification using machine learning

Created By

Nandyala Janaki Lakshmi

228x1a4288@khitguntur.ac.in

College: Kallam HarnadhaReddy Institute Of Technology

INTRODUCTION

Project Description:

The project aims to leverage a comprehensive database sourced from major manufacturing plants across three different countries, including Brazil, to analyse and mitigate workplace accidents. With a focus on promoting industrial safety and health, the database provides insights into the occurrence, severity, and potential risks associated with accidents within these plants. By sharing this valuable dataset with the community, the project seeks to foster collaborative efforts in understanding accident patterns, identifying root causes, and implementing proactive measures to prevent future incidents.

Scenario 1: Galaxy Morphology Classification

Astronomers are interested in studying the morphology of galaxies to understand their formation and evolution processes. By utilizing machine learning techniques, researchers can train a classification model to categorize galaxies into different morphological types such as elliptical, spiral, or irregular. This automated classification process enables astronomers to analyze large datasets of galaxy images efficiently and identify trends or patterns related to galaxy morphology.

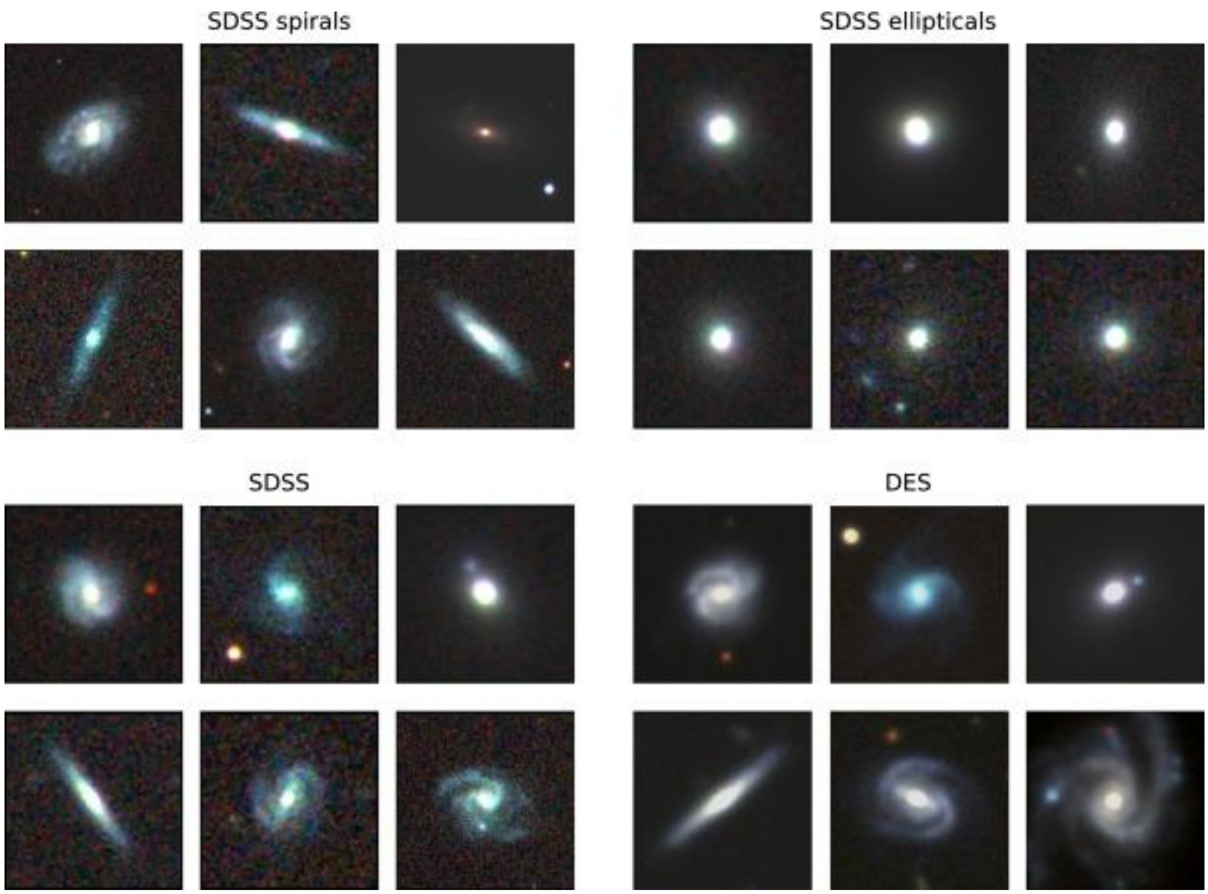
Scenario 2: Galaxy Redshift Estimation

Redshift, which indicates the extent to which light from a galaxy has been shifted towards longer wavelengths due to the expansion of the universe, is a crucial parameter for studying cosmic distances and cosmological phenomena. Machine learning models can be trained to estimate galaxy redshifts based on features extracted from their spectra or photometric properties measured by SDSS. Accurate redshift estimation enables astronomers to map the three-dimensional distribution of galaxies in the universe and investigate large-scale structures such as galaxy clusters and filaments.

Scenario 3: Active Galactic Nuclei (AGN) Identification

Galaxies hosting active galactic nuclei (AGN) exhibit intense emission from a compact region at their centers, powered by accretion onto supermassive black holes. Identifying AGN candidates from SDSS data is essential for studying their properties and understanding their impact on galaxy evolution. Machine learning algorithms can be trained to recognize characteristic signatures of AGN in galaxy spectra or multi-wavelength photometric data, facilitating the automated identification of AGN hosts within large galaxy surveys like SDSS. This enables astronomers to conduct statistical analyses of AGN properties and investigate their role in galaxy formation and evolution processes.

Technical Architecture :



Project Flow

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below:

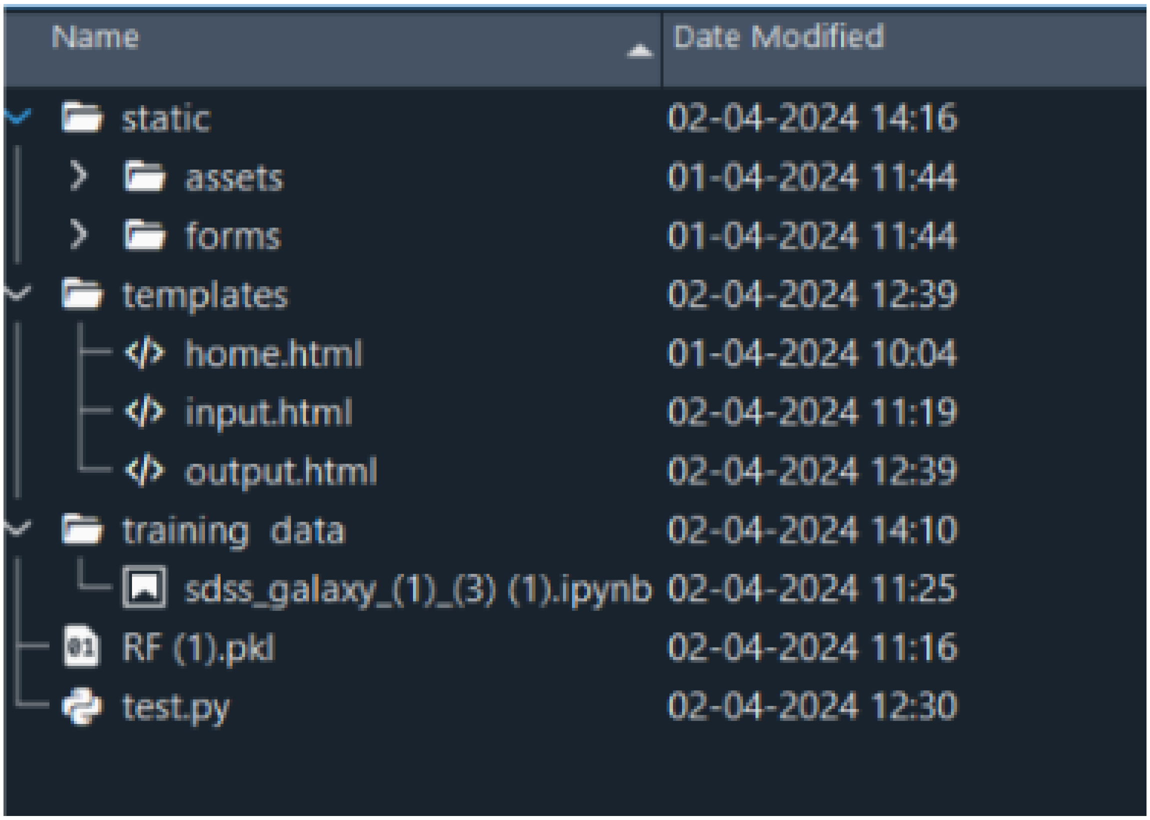
- Data Collection & Preparation
    - Collect the dataset
    - Data Preparation
- Exploratory Data Analysis
    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Training the model in multiple algorithms
    - Testing the model
- Performance Testing
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
    - Save the best model
    - Integrate with Web Framework

Prior Knowledge

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts:
- Supervised learning: https://www.javatpoint.com/supervised-machine-learning
- Decisiontree: https://www.geeksforgeeks.org/python-decision-tree-regression-using-sklearn/
- Logistic Regression :https://www.javatpoint.com/logistic-regression-in-machine-learning
- Random forest: https://www.geeksforgeeks.org/random-forest-regression-in-python/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure



- We are building a flask application which needs HTML pages stored in the Template folder and python script app.py for scripting

- kmodel.pkl is our saved model. Further we will use this model for flask integration.

    - Training folder contains a model training file.

Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc. In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

The Sloan Digital Sky Survey (SDSS) has searched about one-third of the sky and found around 1 billion objects and almost 3 million of those are galaxies. It contains 100,000 rows of photometric image data and the galaxy subclass is limited to two types, 'STARFORMING' or 'STARBURST'

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing the libraries

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings("ignore")
```

Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
#reading the data
df = pd.read_csv('/content/sdss_100k_galaxy_form_burst.csv', header=1)
df.head()
```

	objid	specobjid	ra	dec	u	g	r	i	z	modelFlux_u	...	psfMag_
0	1237646587710669400	8175185722644649984	82.038679	0.847177	21.73818	20.26633	19.32409	18.64037	18.23833	2.007378	...	19.4357
1	1237646588247540577	8175186822156277760	82.138894	1.063072	20.66761	19.32016	18.67888	18.24693	18.04122	5.403369	...	18.8501
2	1237646588247540758	8175187097034184704	82.028510	1.104003	23.63531	21.19671	19.92297	19.31443	18.68396	0.295693	...	19.4223
3	1237648702973083853	332152325571373056	198.544469	-1.097059	20.12374	18.41520	17.47202	17.05297	16.72423	8.920645	...	18.0320
4	1237648702973149350	332154249716721664	198.706864	-1.046217	-9999.00000	-9999.00000	18.37762	18.13383	17.78497	0.000000	...	19.0288

5 rows × 43 columns

Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results.

This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling Outliers

Handling missing values

Let’s find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
#checking shape of dataset
df.shape
```

```
(100000, 43)
```

```
#info about dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 43 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   objid         100000 non-null  int64
 1   specobjid     100000 non-null  uint64
 2   ra            100000 non-null  float64
 3   dec           100000 non-null  float64
 4   u             100000 non-null  float64
 5   g             100000 non-null  float64
 6   r             100000 non-null  float64
 7   i             100000 non-null  float64
 8   z             100000 non-null  float64
 9   modelFlux_u   100000 non-null  float64
 10  modelFlux_g   100000 non-null  float64
 11  modelFlux_r   100000 non-null  float64
 12  modelFlux_i   100000 non-null  float64
 13  modelFlux_z   100000 non-null  float64
 14  petroRad_u    100000 non-null  float64
 15  petroRad_g    100000 non-null  float64
 16  petroRad_i    100000 non-null  float64
 17  petroRad_r    100000 non-null  float64
 18  petroRad_z    100000 non-null  float64
 19  petroFlux_u   100000 non-null  float64
 20  petroFlux_g   100000 non-null  float64
 21  petroFlux_i   100000 non-null  float64
 22  petroFlux_r   100000 non-null  float64
 23  petroFlux_z   100000 non-null  float64
 24  petroR50_u    100000 non-null  float64
 25  petroR50_g    100000 non-null  float64
```

For checking the null values, df.isnull() function is used. To sum those null values, we use .sum() function. From the below image we found that there no null values present in our dataset:

```
[126] # checking for null values
      df.isnull().sum()
```

```
objid           0
specobjid       0
ra              0
dec             0
u               0
g               0
r               0
i               0
z               0
modelFlux_u     0
modelFlux_g     0
modelFlux_r     0
modelFlux_i     0
modelFlux_z     0
petroRad_u      0
petroRad_g      0
petroRad_i      0
petroRad_r      0
petroRad_z      0
petroFlux_u     0
petroFlux_g     0
petroFlux_i     0
petroFlux_r     0
petroFlux_z     0
petroR50_u      0
petroR50_g      0
petroR50_i      0
petroR50_r      0
```

```
petroR50_z      0
psfMag_u        0
psfMag_r        0
psfMag_g        0
psfMag_i        0
psfMag_z        0
expAB_u         0
expAB_g         0
expAB_r         0
expAB_i         0
expAB_z         0
class           0
subclass        0
redshift        0
redshift_err    0
dtype: int64
```

Changing the datatype of subclass from object to int

We transformed the 'subclass' column from object to integer datatype using ordinal encoding to represent object as integers in the dataset.

```
# oridinal encoding - replace subclass with a 0/1 for classification
df['subclass'].replace(['STARFORMING', 'STARBURST'],[0,1], inplace=True)
```

Exploratory Data Analysis.

Descriptive statistical

Descriptive analysis involves examining fundamental characteristics of data using statistical methods. Pandas offers a valuable function known as 'describe' for this purpose. Utilizing the 'describe' function enables us to uncover unique, top, and frequently occurring values within categorical features. Moreover, it provides insights into the mean, standard deviation, minimum, maximum, and percentile values of continuous features.

```
#statistical information about dataset
df.describe()
```

	u	g	r	i	z	modelFlux_u	modelFlux_g	modelFlux_r	modelFlux_i	model
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.
mean	18.518622	17.258221	16.821739	16.362611	15.850865	30.683321	98.845058	175.621855	244.728134	307.
std	105.082004	105.069066	95.035474	100.171155	114.206165	76.552859	229.479215	435.852215	619.825871	809.
min	-9999.000000	-9999.000000	-9999.000000	-9999.000000	-9999.000000	-47.451720	-11.935840	-42.440640	-54.385510	-144.
25%	18.762215	17.505868	16.898845	16.527097	16.281327	9.288132	34.462902	67.453910	91.777325	104.
50%	19.349715	18.072640	17.459080	17.091385	16.861105	18.195690	59.005915	103.828850	145.664550	180.
75%	20.079470	18.656182	17.926918	17.592650	17.453848	31.259628	99.438015	173.929225	244.944825	307.
max	30.960000	30.420980	31.173560	30.562360	28.553240	7915.306000	18668.400000	31755.990000	51923.480000	79058.

8 rows × 37 columns

Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Univariate analysis

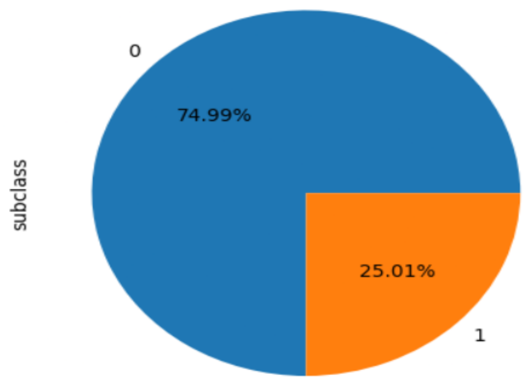
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed a plot that is pie plot.

```
sub=df["subclass"].value_counts()
sub
```

```
0    74993
1    25007
Name: subclass, dtype: int64
```

```
sub.plot(kind="pie",subplots=True,autopct="%1.2f%%")
```

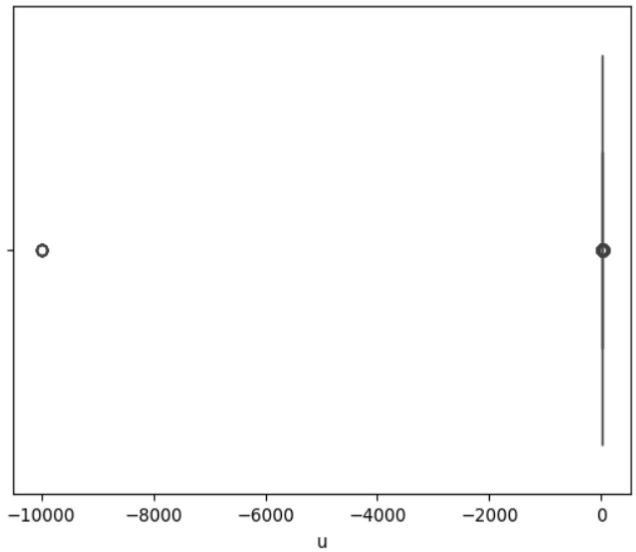
```
array([<Axes: ylabel='subclass'>], dtype=object)
```



From the below boxplot it’s clear that there are outliers in columns.

```
def func(col):
  sns.boxplot(x=col,data=df)
  plt.show()
```

```
for i in df.columns:
  func(i)
```

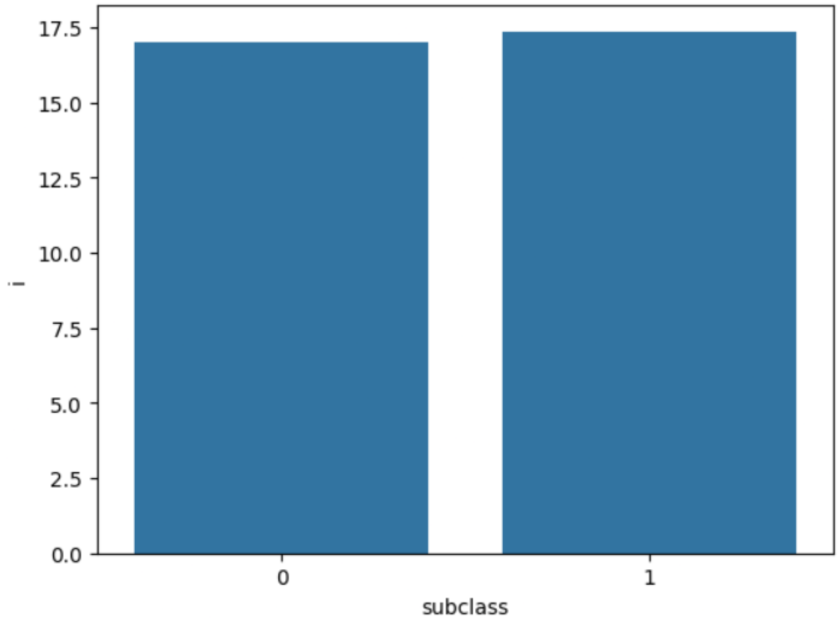


Bivariate analysis

Bivariate analysis is employed to explore the relationship between two features

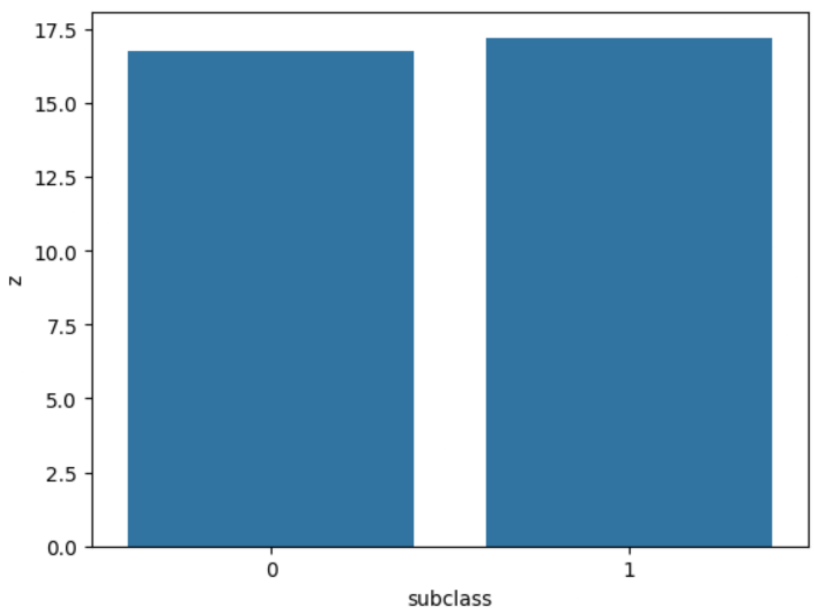
```
sns.barplot(x='subclass', y='i',data=df)
```

<Axes: xlabel='subclass', ylabel='i'>



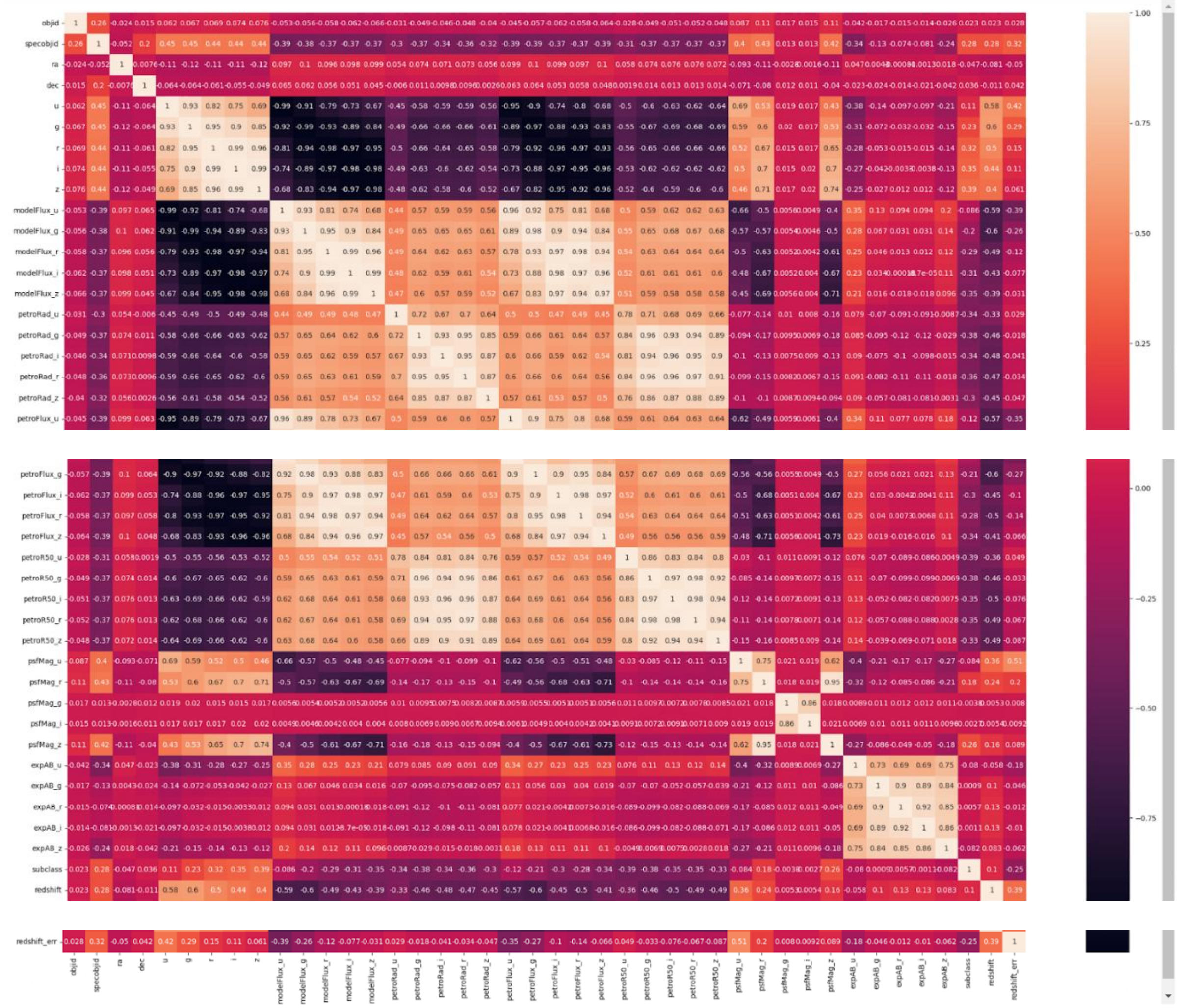

```
sns.barplot(x='subclass', y='z',data=df)
```

<Axes: xlabel='subclass', ylabel='z'>



Multivariate analysis

```
plt.figure(figsize=(30,22))
sns.heatmap(df.corr(),annot=True)
plt.show()
```



Handling the outliers

From the boxplot visual we observed that there are few outliers in columns one of them is u.

Here we are handling outliers with iqr method as mentioned below:

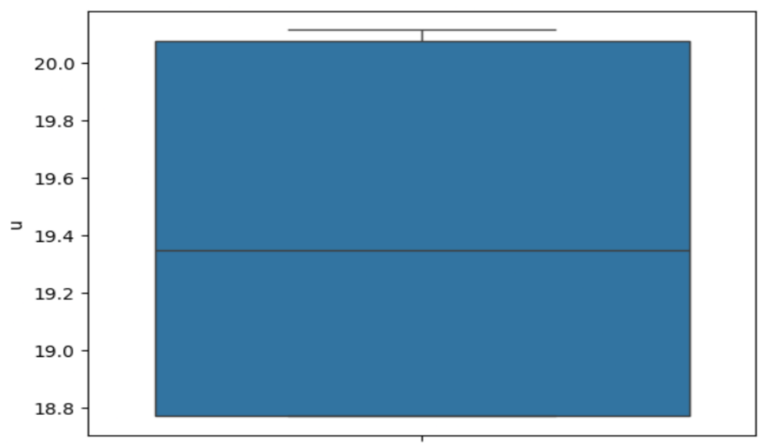
```
quant=df['u'].quantile(q=[0.75,0.25])
print(quant)
Q3=quant.loc[0.75]
print(Q3)
Q1=quant.loc[0.25]
print(Q1)
IQR=Q3-Q1
print(IQR)
maxwhisker=Q3+1.5*IQR
print(maxwhisker)
minwhisker=Q1-1.5*IQR
print(minwhisker)
```

```
0.75    20.079470
0.25    18.762215
Name: u, dtype: float64
20.07947
18.762214999999998
1.317255000000003
22.055352500000005
16.786332499999993
```

```
df['u']=np.where(df['u']> 20.116540, 20.116540,df['u'])
df['u']=np.where(df['u']< 18.772018,18.772018,df['u'])
```

```
sns.boxplot (y='u',data=df)
```

```
<Axes: ylabel='u'>
```



Now we can see that there are no outliers after filling them with iqr using where condition.

Selecting best Features using Select K Best

We selected 10 features out of 43 using Select K Best algorithm to enhance the predictive power and reduce dimensionality in our dataset.

```
x = df.drop(['subclass',], axis=1)
y = df['subclass']
```

```
#i want to know top k best columns in the data frame using SelectkBest k = 10

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# Assuming X and y are your data and target variables
selector = SelectKBest(score_func=f_classif, k=10)  # Select top 10 features
#selector = SelectKBest(score_func=chi2, k=10)  # For classification tasks with non-negative features

# Fit selector to the data
X_selected = selector.fit_transform(X, y)

# Get the names of the selected features
selected_features = X.columns[selector.get_support()]

# Print the selected features
print("Selected features:", selected_features)
```

```
Selected features: Index(['i', 'z', 'modelFlux_z', 'petroRad_g', 'petroRad_r', 'petroFlux_z',
       'petroR50_u', 'petroR50_g', 'petroR50_i', 'petroR50_r'],
      dtype='object')
```

Balancing Value Counts using Smote

We balanced the value counts in our dataset using SMOTE (Synthetic Minority Over-sampling Technique) to address class imbalance and improve the robustness of our machine learning model.

```
# Assuming your target column is 'subclass' in your DataFrame 'df'
X = df.drop(['subclass','class'], axis=1)
y = df['subclass']

# Initialize SMOTE
smote = SMOTE(random_state=42)

# Perform SMOTE oversampling
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check the new value counts
print(pd.Series(y_resampled).value_counts())
```

```
0    74993
1    74993
Name: subclass, dtype: int64
```

Splitting data into train and test

Now let's split the Dataset into train and test sets. First, split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size.

```
df1=df[['i','z','modelFlux_z','petroRad_g','petroRad_r','petroFlux_z','petroR50_u','petroR50_g','petroR50_i','petroR50_r','subclass']]
```

```
from sklearn.model_selection import train_test_split
x = df1[['i', 'z', 'modelFlux_z', 'petroRad_g', 'petroRad_r', 'petroFlux_z',
      'petroR50_u', 'petroR50_g', 'petroR50_i', 'petroR50_r']]
y = df1["subclass"]
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Scaling the feature variables using standardscaler method

From the sklearn pre-processing, we have imported a standard scaler for scaling the feature variables.

```
from sklearn.preprocessing import StandardScaler

# Create a scaler object
sc = StandardScaler()

# Transform your data
scaled_data = sc.fit_transform(x_train)
```

Model Building:

Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Decision Tree Classifier :

A function named "DTC" is created and train and test data are passed as the parameters. Inside the function, Decision Tree Classifier algorithm is initialized and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. For evaluating the model, overfitting and accuracy is calculated.

```python
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
```

```python
# Train the classifier on the training data
clf.fit(x_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(x_test)

# Evaluate the classifier
report = classification_report(y_test, y_pred)
print("Classification Report:\n", report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.83      0.85     14919
           1       0.57      0.65      0.60      5081

    accuracy                           0.79     20000
   macro avg       0.72      0.74      0.73     20000
weighted avg       0.80      0.79      0.79     20000
```

```python
print(accuracy_score( y_pred,y_test))
```

```
0.7851
```

Logistic Regression:

A function named "Logistic_Regression" is implemented, which takes training and testing data as parameters. Within the function, the logistic regression algorithm is initialized and trained using the training data with .fit() function. Subsequently, the model predicts the test data labels using .predict() function, storing the results in a new variable. To assess the model's performance, accuracy is calculated.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,classification_report, recall_score, precision_score, confusion_matrix, f1_score
lg=LogisticRegression()

log=lg.fit(x_train,y_train)

y_pred=lg.predict(x_test)
print("Confusion Matrix: \n",confusion_matrix(y_test,y_pred))
print("-------------------------------------------")
print("Classification report:\n",classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[13332  1587]
 [ 2025  3056]]
-------------------------------------------
Classification report:
               precision    recall  f1-score   support

           0       0.87      0.89      0.88     14919
           1       0.66      0.60      0.63      5081

    accuracy                           0.82     20000
   macro avg       0.76      0.75      0.75     20000
weighted avg       0.81      0.82      0.82     20000
```

```python
print(accuracy_score( y_pred,y_test))
```

```
0.8194
```

Random Forest Classifier:

A function named "RFC" is implemented to train and test data using Random Forest Classifier. Within the function, the Random Forest algorithm is initialized, and the training data is fitted to the model using the .fit() function. Subsequently, predictions are made on the test data using the .predict() function and stored in a new variable. The model's accuracy is then calculated for evaluation.

```python
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
# Train the Random Forest classifier
RF = RandomForestClassifier()
```

```python
RF.fit(x_train, y_train)
RFtrain =RF.predict(x_train)
RFtest =RF.predict(x_test)
```

```
from sklearn.ensemble import RandomForestClassifier
RF = RandomForestClassifier()
# Train the Random Forest classifier
RF = RandomForestClassifier()
```

```
RF.fit(x_train, y_train)
RFtrain =RF.predict(x_train)
RFtest =RF.predict(x_test)
```

```
# Print classification report , confusion matrix
print(confusion_matrix(RFtrain,y_train))
print(confusion_matrix(RFtest,y_test))
print(classification_report(RFtrain,y_train))
print(classification_report(RFtest,y_test))
```

```
[[56574   805]
 [ 3500 19121]]
[[13451  2079]
 [ 1468  3002]]
              precision    recall  f1-score   support

           0       0.94      0.99      0.96     57379
           1       0.96      0.85      0.90     22621

    accuracy                           0.95     80000
   macro avg       0.95      0.92      0.93     80000
weighted avg       0.95      0.95      0.95     80000

              precision    recall  f1-score   support

           0       0.90      0.87      0.88     15530
           1       0.59      0.67      0.63      4470

    accuracy                           0.82     20000
   macro avg       0.75      0.77      0.76     20000
weighted avg       0.83      0.82      0.83     20000
```

```
print( accuracy_score(RFtrain,y_train))
print(accuracy_score( RFtest,y_test))
```

```
0.9461875
0.82265
```

Model Deployment:

Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle
```

```
pickle.dump(RF,open("RF.pkl","wb"))
```

Test the model

Let’s test the model first in python notebook itself. As we have 10 features in this model, let’s check the output by giving all the inputs.

```
RF.predict([[16.946170,16.708910,207.218700, 4.180779, 4.060687,194.731000,2.141953,2.149080,2.056686,2.055798]])
```

```
array([0])
```

```
RF.predict([[17.675285,17.52775,104.25655,3.397512,3.424717,90.717547,1.613005,1.632243,1.548225,1.596137 ]])
```

```
array([1])
```

Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Building Html Pages:

For this project create two HTML files namely

- index.html
- inner-page.html

and save them in the templates folder.

Build Python code

Import the libraries

```
from flask import Flask, request, render_template
import pickle
import numpy as np
import json
import requests
import pandas as pd
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (__name__) as argument.

```
app = Flask(__name__)

# Load the trained model
with open('RF (1).pkl', 'rb') as file:
    model = pickle.load(file)
```

Render HTML page:

```
@app.route("/")
def home():
    return render_template("home.html")
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered.

Whenever you enter the values from the html page the values can be retrieved using POST and GET Methods.

Retrieving the value from UI:

```
@app.route('/submit', methods=["POST"])  # Specify POST method
def submit():
    # Reading input values from the form
    input_feature = [float(x) for x in request.form.values()]
    names = [ 'i', 'z',  'modelFlux_z', 'petroRad_g',
             'petroRad_r',  'petroFlux_z', 'petroR50_u', 'petroR50_g', 'petroR50_i',
             'petroR50_r']

    print("Number of columns in names:", len(names))
    print("Number of columns in input_feature:", len(input_feature))
    print("Column names:", names)

    data = pd.DataFrame([input_feature], columns=names)

    # Make prediction
    prediction = model.predict(data)


    # Render the output template with the prediction result
    if prediction == 0:
        print(prediction)
        return render_template('output.html',prediction='starforming')
    else:
       return render_template('output.html',prediction='starbursting')
```

Here we are routing our app to conditional statement. This will retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function.

This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

```
if __name__ == "__main__":
    app.run(debug=True, port=2222)
```

Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type "app.py" command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

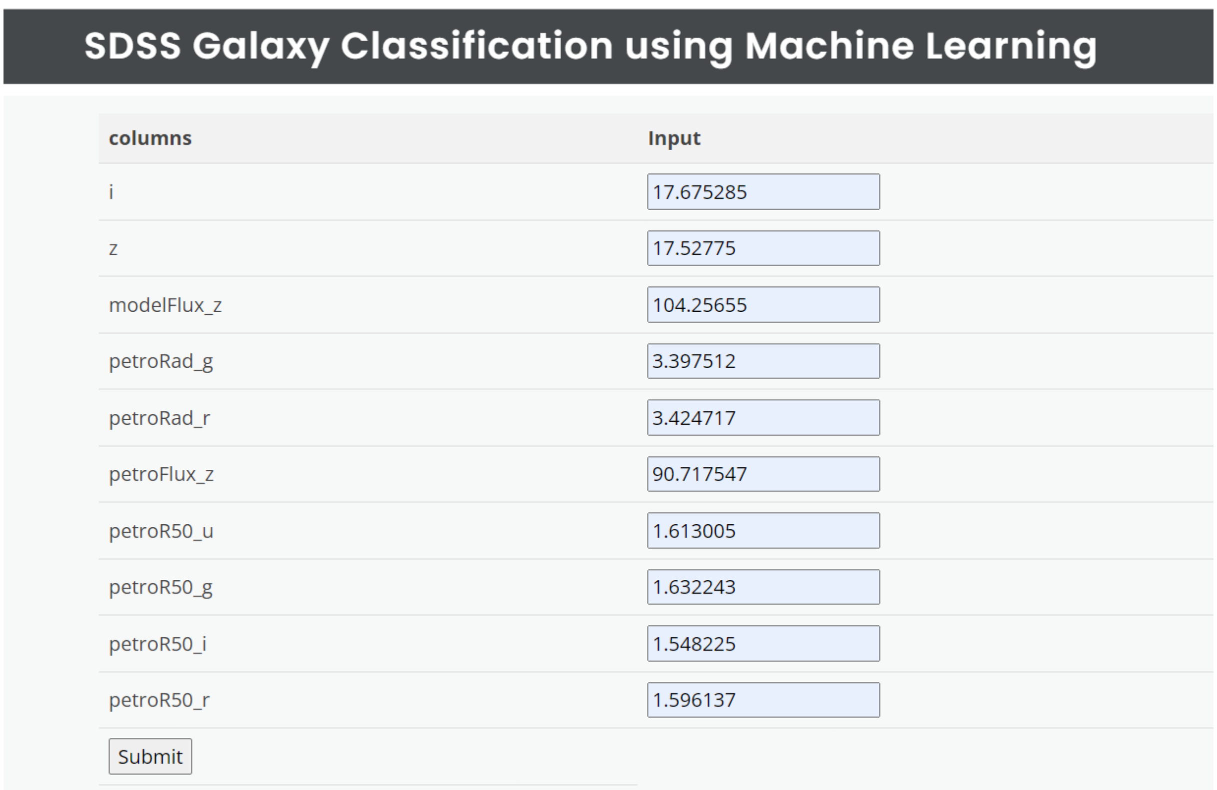
```
IPython 8.22.2 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/apurva/Downloads/project/test.py', wdir='C:/Users/apurva/Downloads/project')
 * Serving Flask app 'test'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:2222
Press CTRL+C to quit
```

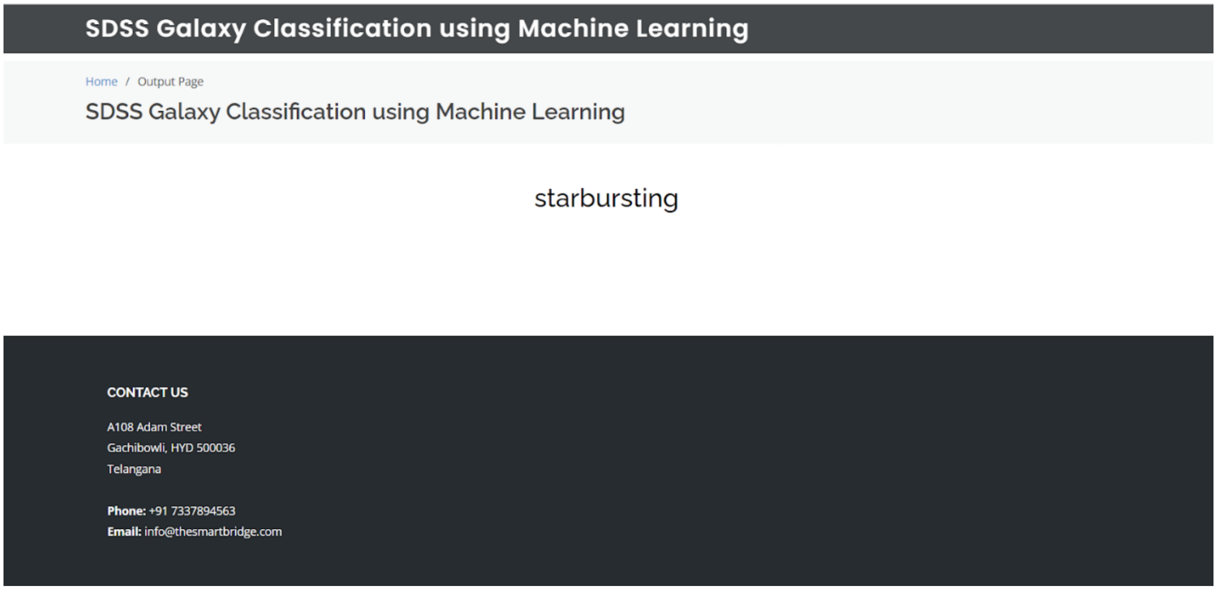
Now, Go the web browser and write the localhost url (http://127.0.0.1:2222)to get the below results


SDSS Galaxy Classification using Machine Learning
HOME
ABOUT
HOW IT WORKS
CONTACT
GET STARTED
Welcome to SDSS Galaxy Classification using Machine Learning
SDSS classifies galaxies into 'STARFORMING' or 'STARBURST' types.

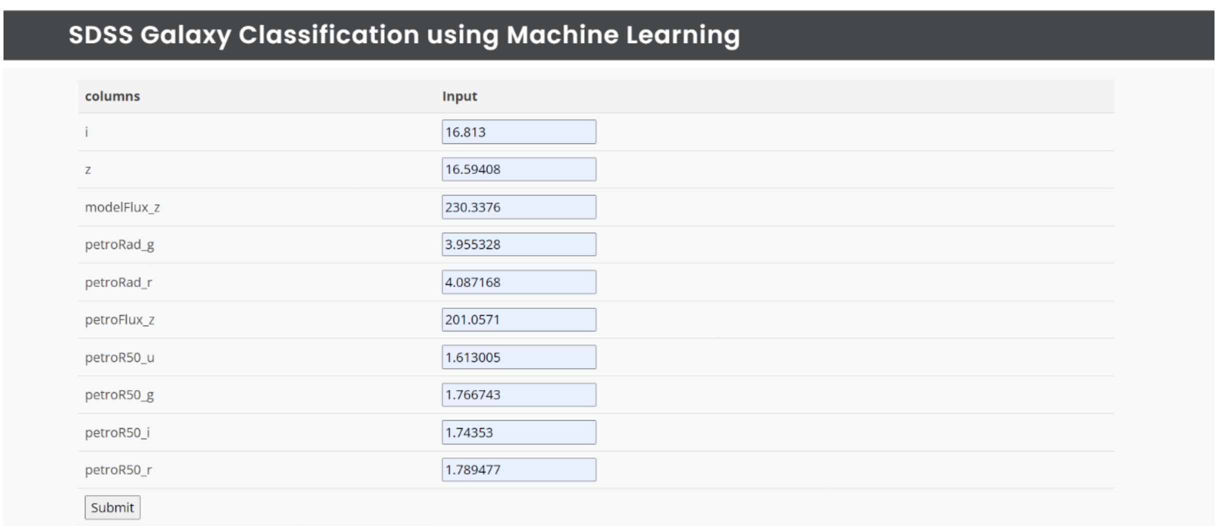
Input Page :


SDSS Galaxy Classification using Machine Learning
columns
Input
i
17.675285
z
17.52775
modelFlux_z
104.25655
petroRad_g
3.397512
petroRad_r
3.424717
petroFlux_z
90.717547
petroR50_u
1.613005
petroR50_g
1.632243
petroR50_i
1.548225
petroR50_r
1.596137
Submit

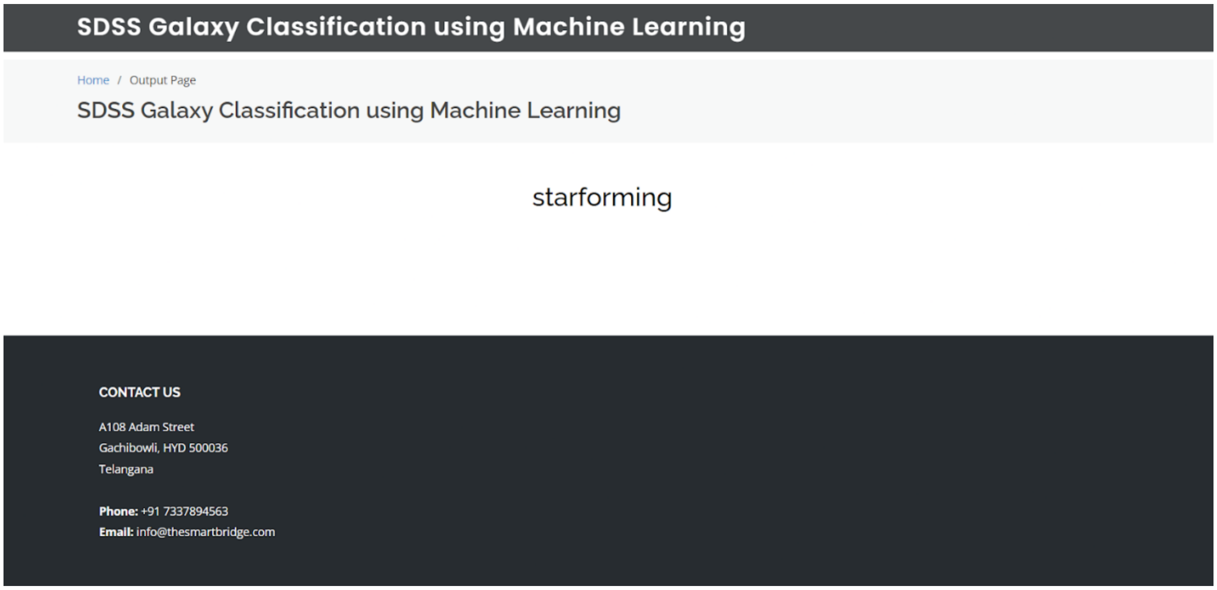
Output Page:

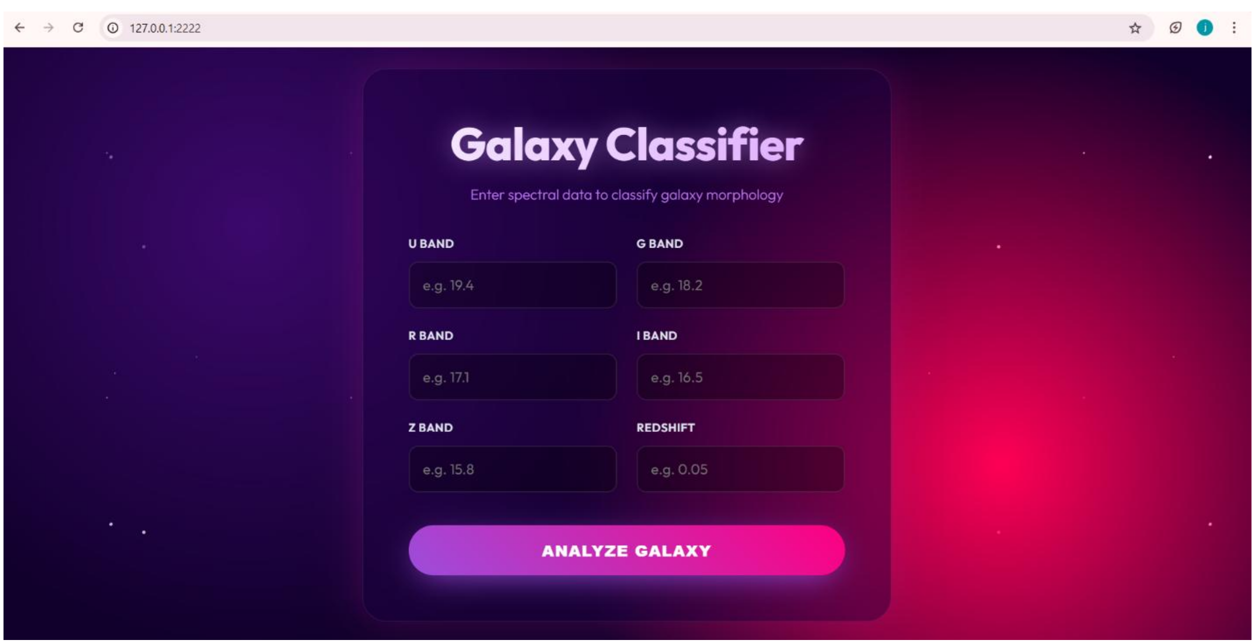


Input Page:



Output Page:


SDSS Galaxy Classification using Machine Learning
Home / Output Page
SDSS Galaxy Classification using Machine Learning
starforming
CONTACT US
A108 Adam Street
Gachibowli, HYD 500036
Telangana
Phone: +91 7337894563
Email: info@thesmartbridge.com


127.0.0.1:2222
Galaxy Classifier
Enter spectral data to classify galaxy morphology
U BAND
e.g. 19.4
G BAND
e.g. 18.2
R BAND
e.g. 17.1
I BAND
e.g. 16.5
Z BAND
e.g. 15.8
REDSHIFT
e.g. 0.05
ANALYZE GALAXY