star prediction using AI

```
import numpy as np
import pandas as pd
```

numpy (np): Used for numerical operations, like working with arrays. (documentaion: https://numpy.org/doc/)

pandas (pd): Used for handling and analyzing data in tables (like Excel sheets). (documentaion: https://pandas.pydata.org/)

Load data

```
data = pd.read_csv('star_data.csv')
```

This line reads a CSV file called 'star_data.csv' and loads it into a pandas DataFrame called data, which is like a table to store and analyze data.

```
data.head()

   Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
0             3068          0.002400        0.1700
16.12
1             3042          0.000500        0.1542
16.60
2             2600          0.000300        0.1020
18.70
3             2800          0.000200        0.1600
16.65
4             1939          0.000138        0.1030
20.06

   Star type Star color Spectral Class
0          0        Red              M
1          0        Red              M
2          0        Red              M
3          0        Red              M
4          0        Red              M
```

This displays the first five rows of the data DataFrame, giving a quick preview of the dataset.

```
data.tail()

     Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
235            38940          374830.0        1356.0
-9.93
```

```
236             30839         834042.0       1194.0
-10.63
237              8829         537493.0       1423.0
-10.73
238              9235         404940.0       1112.0
-11.23
239             37882         294903.0       1783.0
-7.80

     Star type Star color Spectral Class
235          5       Blue              O
236          5       Blue              O
237          5      White              A
238          5      White              A
239          5       Blue              O
```

This shows the last five rows of the data DataFrame, allowing you to see the end of the dataset.

```
data.shape

(240, 7)
```

This returns the dimensions of the data DataFrame, showing how many rows and columns it has (in the format (rows, columns)).

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 240 entries, 0 to 239
Data columns (total 7 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Temperature (K)         240 non-null    int64
 1   Luminosity(L/Lo)        240 non-null    float64
 2   Radius(R/Ro)            240 non-null    float64
 3   Absolute magnitude(Mv)  240 non-null    float64
 4   Star type               240 non-null    int64
 5   Star color              240 non-null    object
 6   Spectral Class          240 non-null    object
dtypes: float64(3), int64(2), object(2)
memory usage: 13.2+ KB
```

This provides a summary of the data DataFrame, including the number of non-null entries, data types of each column, and memory usage

data.isnull().sum()

This calculates and displays the number of missing (null) values in each column of the data DataFrame.

```
data.describe().round(1)
```

```
       Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  \
count            240.0             240.0         240.0
mean           10497.5          107188.4         237.2
std             9552.4          179432.2         517.2
min             1939.0               0.0           0.0
25%             3344.2               0.0           0.1
50%             5776.0               0.1           0.8
75%            15055.5          198050.0          42.8
max            40000.0          849420.0        1948.5

       Absolute magnitude(Mv)  Star type
count                   240.0      240.0
mean                      4.4        2.5
std                      10.5        1.7
min                     -11.9        0.0
25%                      -6.2        1.0
50%                       8.3        2.5
75%                      13.7        4.0
max                      20.1        5.0
```

This generates descriptive statistics for the data DataFrame, such as mean, standard deviation, and quartiles, rounded to one decimal place.

Mean: The average value of the data in a column.

Std (Standard Deviation): Measures how spread out the values are from the mean.

Min: The smallest value in the column.

25% (First Quartile): The value below which 25% of the data falls.

50% (Median or Second Quartile): The middle value of the data, where half the values are above and half are below.

75% (Third Quartile): The value below which 75% of the data falls.

Max: The largest value in the column.

```
data['Star type'].unique()
```

```
array([0, 1, 2, 3, 4, 5])
```

This shows all the unique values in the Star type column of the data DataFrame, helping you see what different types of stars are listed.

```
data["Star color"].value_counts()
```

```
Star color
Red                   112
```

```
Blue                      55
Blue-white                26
Blue White                10
yellow-white               8
White                      7
Yellowish White            3
Blue white                 3
white                      3
Orange                     2
Whitish                    2
yellowish                  2
Pale yellow orange         1
White-Yellow               1
Blue                       1
Yellowish                  1
Orange-Red                 1
Blue white                 1
Blue-White                 1
Name: count, dtype: int64
```

This counts and displays the number of occurrences of each unique value in the Star color column, showing how many stars are of each color.

```
import dataProcess
dataProcess.data_adjust(data)
```

```
     Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
0               3068          0.002400        0.1700
16.12
1               3042          0.000500        0.1542
16.60
2               2600          0.000300        0.1020
18.70
3               2800          0.000200        0.1600
16.65
4               1939          0.000138        0.1030
20.06
..               ...               ...           ...
...
235            38940     374830.000000     1356.0000
-9.93
236            30839     834042.000000     1194.0000
-10.63
237             8829     537493.000000     1423.0000
-10.73
238             9235     404940.000000     1112.0000
-11.23
239            37882     294903.000000     1783.0000
```

```
-7.80

     Star type Star color Spectral Class
0            0        red              M
1            0        red              M
2            0        red              M
3            0        red              M
4            0        red              M
..         ...        ...            ...
235          5       blue              O
236          5       blue              O
237          5      white              A
238          5      white              A
239          5       blue              O

[240 rows x 7 columns]
```

the function adjusts the Star color column to:

Standardize Colors: Ensure that similar star colors are represented consistently (e.g., "yellowishwhite" and "whiteyellow" are both changed to "yellowwhite").

Remove Extra Characters: Eliminate spaces and hyphens to make the color names uniform. Simplify Data: Make the data cleaner and easier to work with for analysis.

```
data["Star color"].value_counts()

Star color
red                   112
blue                   56
bluewhite              41
yellowwhite            12
white                  12
yellowish               3
orange                  2
paleyelloworange        1
orangered               1
Name: count, dtype: int64

print(data['Star color'].unique())
data['Star color'].nunique()

['red' 'bluewhite' 'white' 'yellowwhite' 'paleyelloworange' 'blue'
 'orange' 'yellowish' 'orangered']

9
```

print(data['Star color'].unique()): Displays all the unique values in the Star color column, showing the distinct colors after adjustments.

data['Star color'].nunique(): Counts and returns the number of unique colors in the Star color column, giving you the total number of distinct star colors.

Now the Star color feature has meaningful values. All entries have been standardized, orange and orangered are distinct colors, so it's better to keep them as they are.

```
print(data['Spectral Class'].unique())
data['Spectral Class'].nunique()
```

```
['M' 'B' 'A' 'F' 'O' 'K' 'G']
```

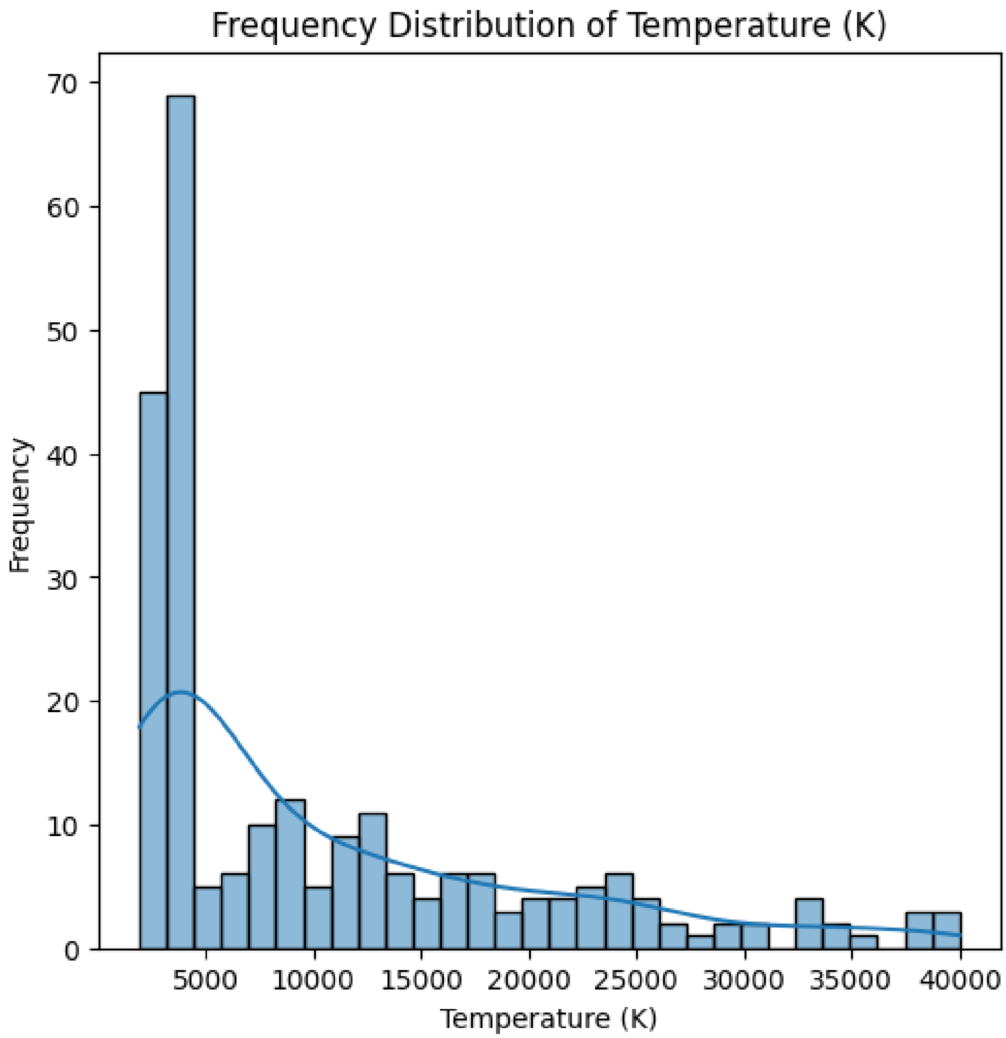
```
7
```

print(data['Spectral Class'].unique()): Displays all the unique values in the Spectral Class column, showing the distinct spectral classes present in the data.

data['Spectral Class'].nunique(): Counts and returns the number of unique spectral classes in the Spectral Class column, giving you the total number of distinct spectral classes.

Data visualisation

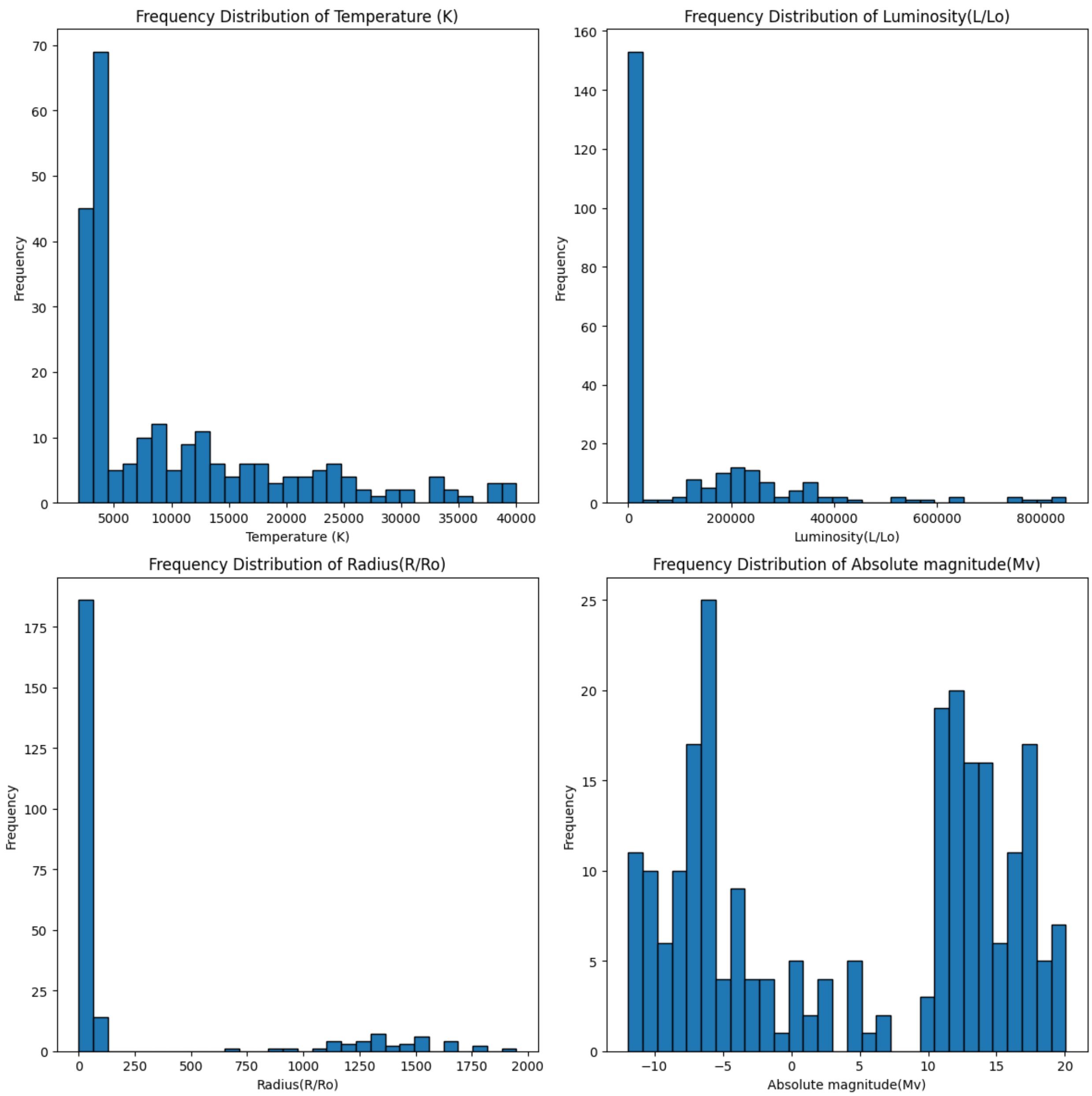
```
import visual
visual.plot(data)
```



we created this plot to visualize the distribution of temperatures in your dataset. By showing a histogram with a kernel density estimate, you can:

- Understand Distribution: See how temperatures are distributed across different values.
- Identify Patterns: Recognize any patterns or trends in the temperature data.
- Check for Skewness: Observe if the temperature data is skewed or if there are any anomalies.

```
visual.plot_all(data)
```



Multiple Histograms: The function generates histograms for several features (Temperature, Luminosity, Radius, Absolute magnitude, and spectral class) to show their distributions.

In this dataset, it's clear that some classes, like 'M', have many more instances than others. This imbalance is important to note because it can affect the performance of machine learning models, potentially resulting in biased predictions that favor the majority class.

```
visual.MaxScale(data)

     Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
0           0.151602          0.147329      0.243442
0.876798
```

```
1          0.148790         0.079381      0.235546
0.891807
2          0.096917         0.057254      0.202094
0.957473
3          0.121402         0.039691      0.238535
0.893371
4          0.000000         0.023617      0.202884
1.000000
..              ...              ...           ...
...
235        0.991127         0.964563      0.970656
0.062226
236        0.914065         0.999209      0.960358
0.040338
237        0.500831         0.980177      0.974560
0.037211
238        0.515685         0.967910      0.954599
0.021576
239        0.982026         0.954175      0.992815
0.128831

     Star type Star color Spectral Class
0            0        red              M
1            0        red              M
2            0        red              M
3            0        red              M
4            0        red              M
..         ...        ...            ...
235          5       blue              O
236          5       blue              O
237          5      white              A
238          5      white              A
239          5       blue              O

[240 rows x 7 columns]
```

we will apply log Transformation and MinMax Scaling

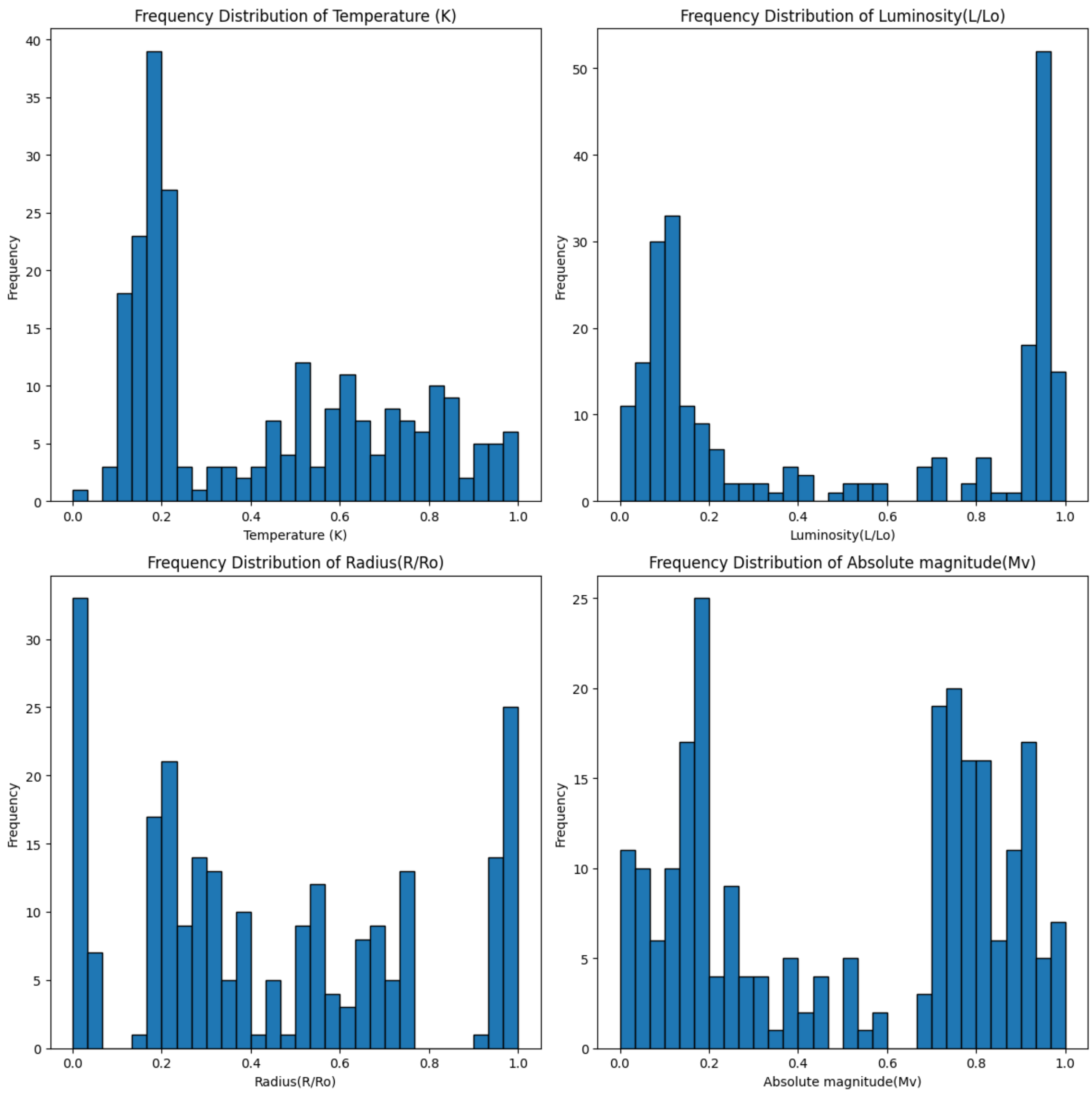
Log transformation

Purpose: To correct skewed distributions. Many features in datasets, like temperature and luminosity, often have skewed distributions, meaning they have a long tail on one side. Log transformation helps to compress the range and make the data more normally distributed, which improves the performance of many machine learning algorithms.

MinMax Scaling

Purpose: To normalize the data. After applying the log transformation, the values might still vary widely. MinMax scaling adjusts all features to a uniform scale (between 0 and 1), which ensures that no feature dominates others due to its scale. This helps algorithms that are sensitive to the scale of data, like gradient-based methods, perform better.

```
visual.plot_all(data)
```



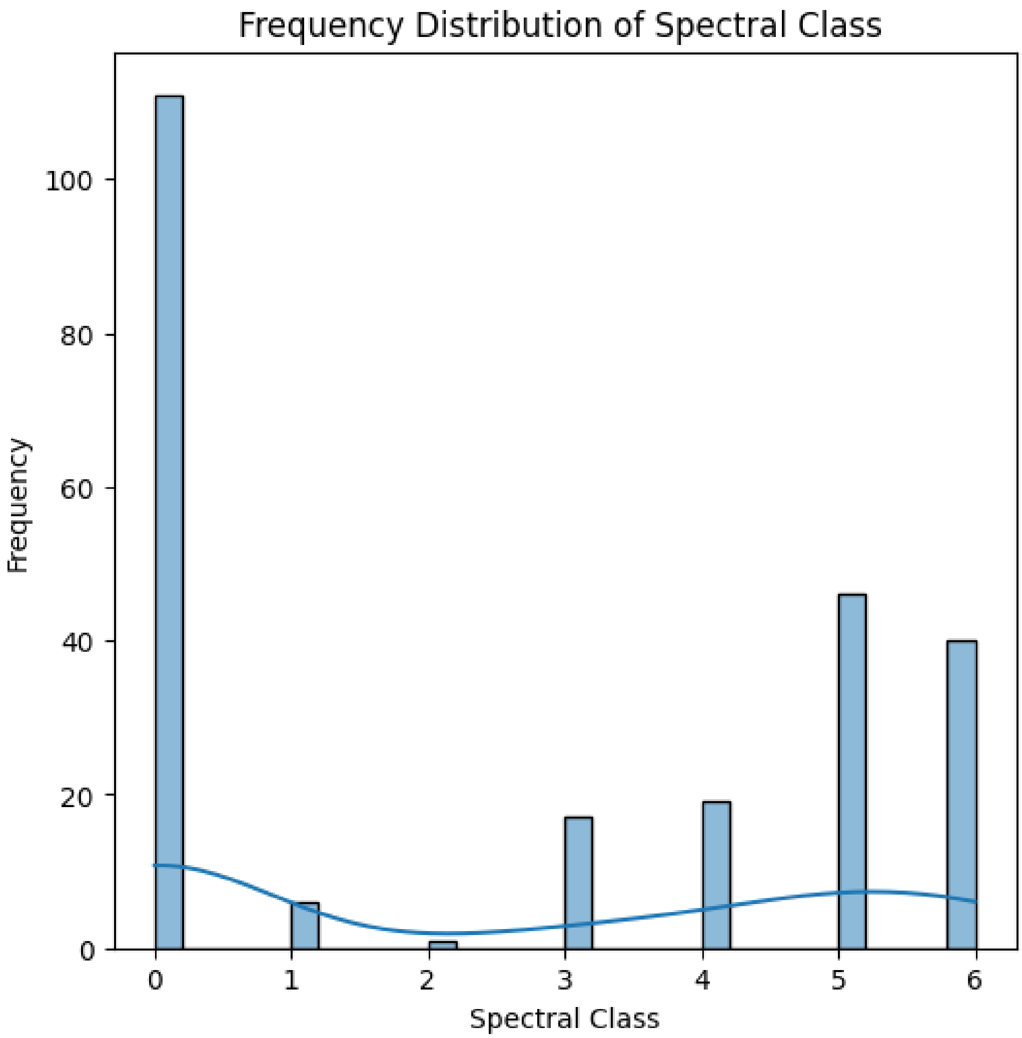
Now the distribution of the data looks better

Now we need to change the output classes from (letters) into numbers, we will Convert categorical data into numerical format,

which is often required for machine learning algorithms that work with numerical inputs.

This transformation simplifies the data, making it easier for algorithms to process and analyze.

```
import process
process.map(data)

     Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
0           0.151602          0.147329      0.243442
0.876798
1           0.148790          0.079381      0.235546
0.891807
2           0.096917          0.057254      0.202094
0.957473
3           0.121402          0.039691      0.238535
0.893371
4           0.000000          0.023617      0.202884
1.000000
..               ...               ...           ...
...
235         0.991127          0.964563      0.970656
0.062226
236         0.914065          0.999209      0.960358
0.040338
237         0.500831          0.980177      0.974560
0.037211
238         0.515685          0.967910      0.954599
0.021576
239         0.982026          0.954175      0.992815
0.128831

     Star type Star color  Spectral Class
0            0        red               0
1            0        red               0
2            0        red               0
3            0        red               0
4            0        red               0
..         ...        ...             ...
235          5       blue               6
236          5       blue               6
237          5      white               4
238          5      white               4
239          5       blue               6

[240 rows x 7 columns]

visual.plot_output(data)
```



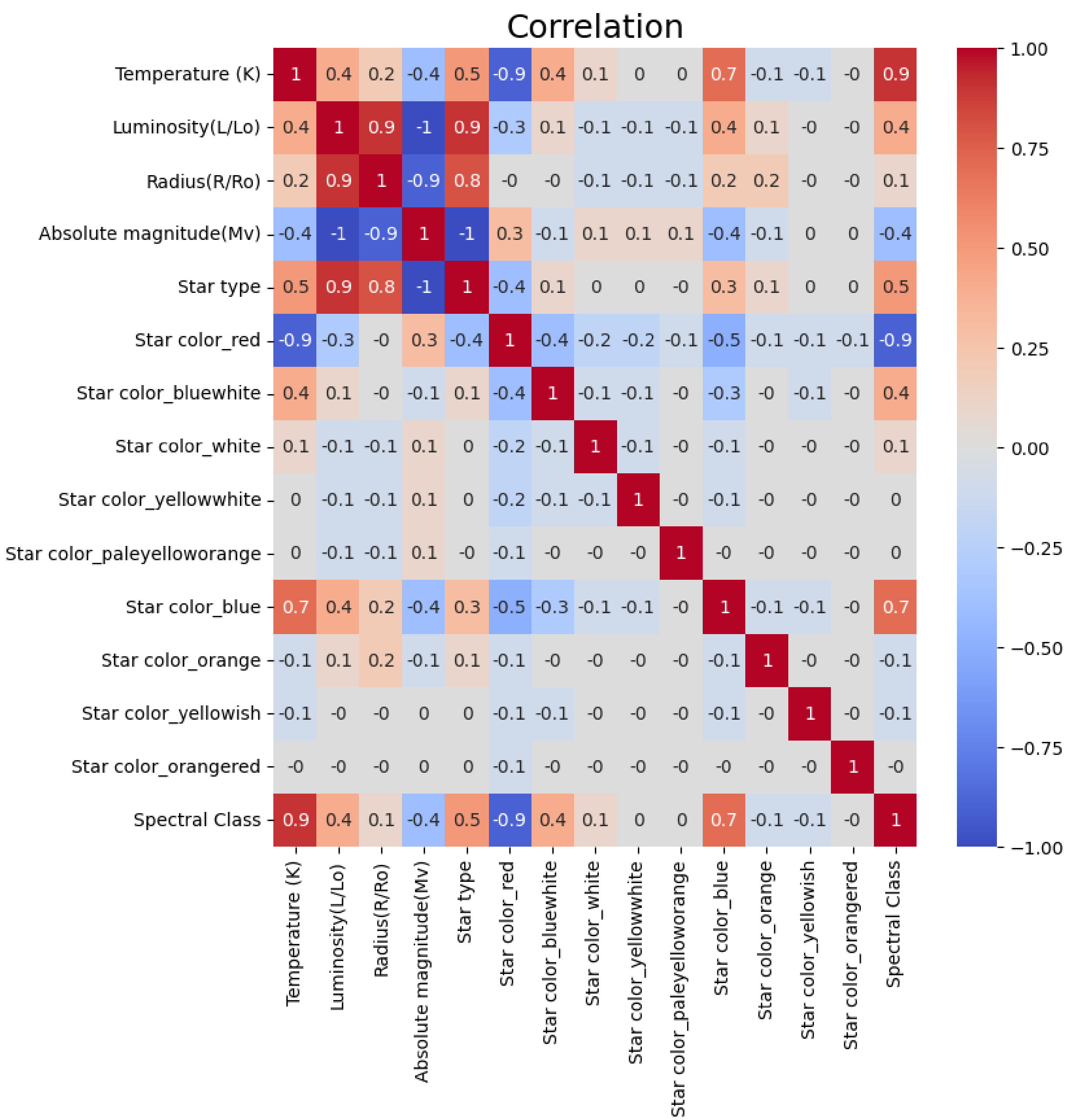
Now as you can see, the star spectral class is as numbers instead of letters, now what we will do next is to make separate columns for each category, we will do this because simply if we map categories to numbers (e.g., 0, 1, 2), the model might infer an unintended ordinal relationship (e.g., 0 < 1 < 2 implies some sort of ranking). One-hot encoding prevents this by treating each category independently

```
data = process.oneHot(data)
```

Now we have a separete column for each Spectral class

```
data.head()

   Temperature (K)  Luminosity(L/Lo)  Radius(R/Ro)  Absolute
magnitude(Mv)  \
0         0.151602          0.147329      0.243442
0.876798
1         0.148790          0.079381      0.235546
0.891807
```

```
2          0.096917          0.057254      0.202094
0.957473
3          0.121402          0.039691      0.238535
0.893371
4          0.000000          0.023617      0.202884
1.000000

   Star type  Star color_red  Star color_bluewhite  Star
color_white  \
0          0               1                     0                  0

1          0               1                     0                  0

2          0               1                     0                  0

3          0               1                     0                  0

4          0               1                     0                  0


   Star color_yellowwhite  Star color_paleyelloworange  Star
color_blue  \
0                       0                            0
0
1                       0                            0
0
2                       0                            0
0
3                       0                            0
0
4                       0                            0
0

   Star color_orange  Star color_yellowish  Star color_orangered  \
0                  0                     0                     0
1                  0                     0                     0
2                  0                     0                     0
3                  0                     0                     0
4                  0                     0                     0

   Spectral Class
0               0
1               0
2               0
3               0
4               0

process.correlation(data)
```



Visualize Relationships: The heatmap helps visualize the correlation between different features in your dataset, making it easier to understand which features are positively or negatively correlated.

Identify Patterns: By showing these relationships, you can identify patterns and potential multicollinearity issues, which can be useful for feature selection and understanding the data better.

```
data.shape
```

```
(240, 15)
```

Use AI Models

```
import models
models.split(data)
```

This is a standard step in machine learning to prepare the data. By splitting the dataset into training and testing sets, you can train your model on one part of the data and test its performance on another (unseen) part, ensuring the model generalizes well to new data.

```
models.svm_model()

Fitting 3 folds for each of 18 candidates, totalling 54 fits

D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\
_split.py:776: UserWarning: The least populated class in y has only 1
members, which is less than n_splits=3.
  warnings.warn(

SVM Performance:
               precision    recall  f1-score   support

           0       0.95      0.86      0.90        21
           1       0.40      0.67      0.50         3
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         2
           5       1.00      1.00      1.00        10
           6       1.00      1.00      1.00        10

    accuracy                           0.92        48
   macro avg       0.89      0.92      0.90        48
weighted avg       0.94      0.92      0.93        48

Best Parameters for SVM:  {'svc__C': 10, 'svc__gamma': 1,
'svc__kernel': 'linear'}

(Pipeline(steps=[('scaler', StandardScaler()),
                 ('svc',
                  SVC(C=10, class_weight='balanced', gamma=1,
kernel='linear'))]),
 0.9166666666666666)
```

Support Vector Machine (SVM) is a powerful and versatile machine learning algorithm used for classification and regression tasks.

How it works: SVM works by finding the hyperplane that creates the largest distance (margin) between the closest points of different classes, called support vectors. For non-linearly separable data, SVM can use kernels (e.g., linear, radial basis function (RBF)) to project the data into higher dimensions, making it easier to find a separating hyperplane.

Documentation: https://scikit-learn.org/stable/modules/svm.html

```
models.decision_tree_model()

Fitting 3 folds for each of 32 candidates, totalling 96 fits
Decision Tree Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      0.33      0.50         3
           3       1.00      1.00      1.00         2
           4       0.50      1.00      0.67         2
           5       0.83      1.00      0.91        10
           6       1.00      0.80      0.89        10

    accuracy                           0.92        48
   macro avg       0.89      0.86      0.83        48
weighted avg       0.94      0.92      0.91        48

Best Parameters for Decision Tree:  {'criterion': 'gini', 'max_depth': 
10, 'min_samples_leaf': 1, 'min_samples_split': 2}

D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\
_split.py:776: UserWarning: The least populated class in y has only 1 
members, which is less than n_splits=3.
  warnings.warn(

(DecisionTreeClassifier(class_weight='balanced', max_depth=10),
 0.9166666666666666)
```

A decision tree is a valuable tool for classification tasks, offering interpretability and flexibility. By tuning hyperparameters and visualizing the tree, you can build an effective model that provides insights into how predictions are made.

Documentation: https://scikit-learn.org/stable/modules/tree.html

```
models.random_forest_model()

Fitting 3 folds for each of 72 candidates, totalling 216 fits

D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\
_split.py:776: UserWarning: The least populated class in y has only 1 
members, which is less than n_splits=3.
  warnings.warn(

Random Forest Performance:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        21
           1       1.00      0.33      0.50         3
           3       0.67      1.00      0.80         2
           4       1.00      0.50      0.67         2
           5       0.91      1.00      0.95        10
```

```
           6       1.00      1.00      1.00        10

    accuracy                           0.94        48
   macro avg       0.92      0.81      0.82        48
weighted avg       0.95      0.94      0.93        48

Best Parameters for Random Forest:  {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimators': 50}

(RandomForestClassifier(class_weight='balanced', min_samples_leaf=2,
                        min_samples_split=5, n_estimators=50),
 0.9375)
```

A Random Forest is an ensemble learning method that operates by constructing multiple decision trees during training. It combines the outputs of these decision trees to improve classification or regression results. The idea is to "average out" the noise from individual trees and make a more accurate and robust model.

Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

The run_all_models function automates the process of training and evaluating three different machine learning models (SVM, Decision Tree, and Random Forest) on the given dataset. It then selects the best model based on performance scores.

from above we can see thats the best model is Random Forest Classifier

```
models.run_all_models(data)

Fitting 3 folds for each of 18 candidates, totalling 54 fits

D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\_split.py:776: UserWarning: The least populated class in y has only 1 members, which is less than n_splits=3.
  warnings.warn(

SVM Performance:
              precision    recall  f1-score   support

           0       0.95      0.86      0.90        21
           1       0.40      0.67      0.50         3
           3       1.00      1.00      1.00         2
           4       1.00      1.00      1.00         2
           5       1.00      1.00      1.00        10
           6       1.00      1.00      1.00        10

    accuracy                           0.92        48
   macro avg       0.89      0.92      0.90        48
weighted avg       0.94      0.92      0.93        48
```

```
Best Parameters for SVM:  {'svc__C': 10, 'svc__gamma': 1,
'svc__kernel': 'linear'}
Fitting 3 folds for each of 32 candidates, totalling 96 fits
Decision Tree Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      0.33      0.50         3
           3       0.50      1.00      0.67         2
           4       0.33      0.50      0.40         2
           5       0.82      0.90      0.86        10
           6       1.00      0.80      0.89        10

    accuracy                           0.88        48
   macro avg       0.78      0.76      0.72        48
weighted avg       0.91      0.88      0.88        48

Best Parameters for Decision Tree:  {'criterion': 'gini', 'max_depth':
10, 'min_samples_leaf': 1, 'min_samples_split': 2}
Fitting 3 folds for each of 72 candidates, totalling 216 fits

D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\
_split.py:776: UserWarning: The least populated class in y has only 1
members, which is less than n_splits=3.
  warnings.warn(
D:\Yazan\jupyter2\venv\lib\site-packages\sklearn\model_selection\
_split.py:776: UserWarning: The least populated class in y has only 1
members, which is less than n_splits=3.
  warnings.warn(

Random Forest Performance:
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        21
           1       1.00      0.33      0.50         3
           3       0.67      1.00      0.80         2
           4       1.00      0.50      0.67         2
           5       0.91      1.00      0.95        10
           6       1.00      1.00      1.00        10

    accuracy                           0.94        48
   macro avg       0.92      0.81      0.82        48
weighted avg       0.95      0.94      0.93        48

Best Parameters for Random Forest:  {'criterion': 'gini', 'max_depth':
20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Best model is: RandomForestClassifier

RandomForestClassifier(class_weight='balanced', max_depth=20,
n_estimators=50)
```