ML APPROACHES FOR STELLAR CLASSIFICATION

BCSE209L (MACHINE LEARNING) DIGITAL ASSIGNMENT

Aakaash A
22BAI1113

Jayanth
22BAI1177

Atchudhan Sreekanth
22BAI1256

Table of contents

01 **Objectives**

A brief description of the goals, datasets used and the theory and science involved in it

02 **Modelling**

Coding and execution of different ML models/algorithms to analyse the chosen dataset

03 **Analysis**

Comparing the complexity of the models used, through statistical and mathematical methods

04 **Inferences**

Final remarks after the analysis, and a list of the pros and cons involved in each algorithm

01
OBJECTIVES

A brief description of the goals, datasets used and the theory and science involved in it

STAR (/sta:r/)

- noun

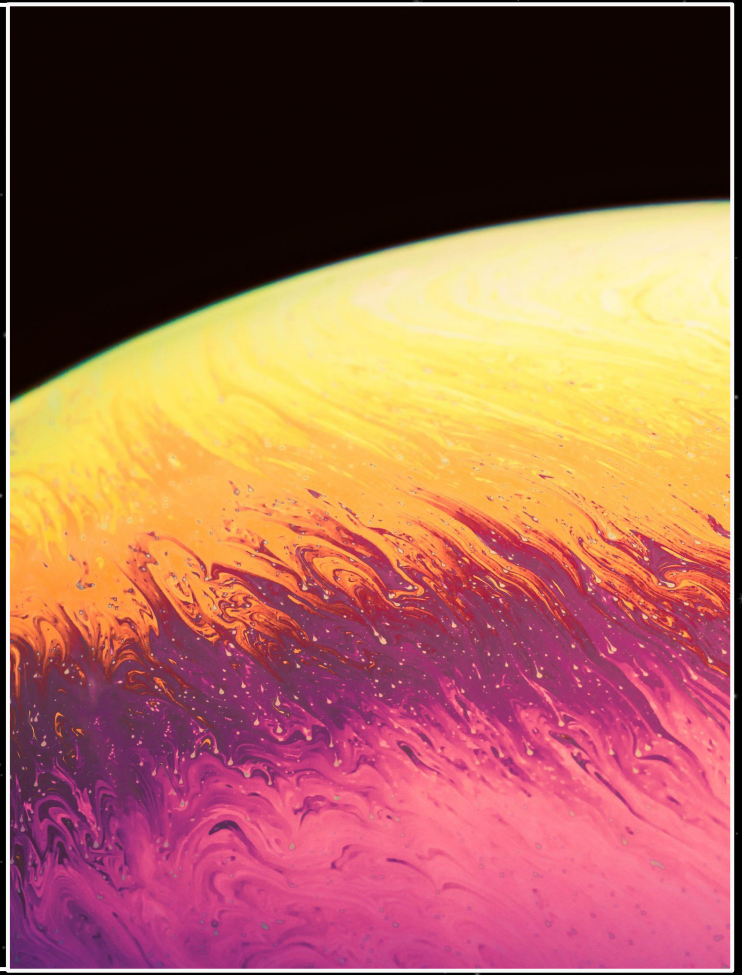
A star is a massive celestial body that emits light and heat through nuclear fusion reactions in its core

Stellar Classification

Stellar classification = Harvard System + Morgan Keenan System
Harvard System: Based on a star's surface temperature
Morgan Keenan System: Based on the luminousity
To classify stars, astronomers plot them on a graph called an HR diagram. The horizontal axis represents temperature (hot to cool from left to right), and the vertical axis represents luminosity (low to high) and these are then clustered into 7 classes

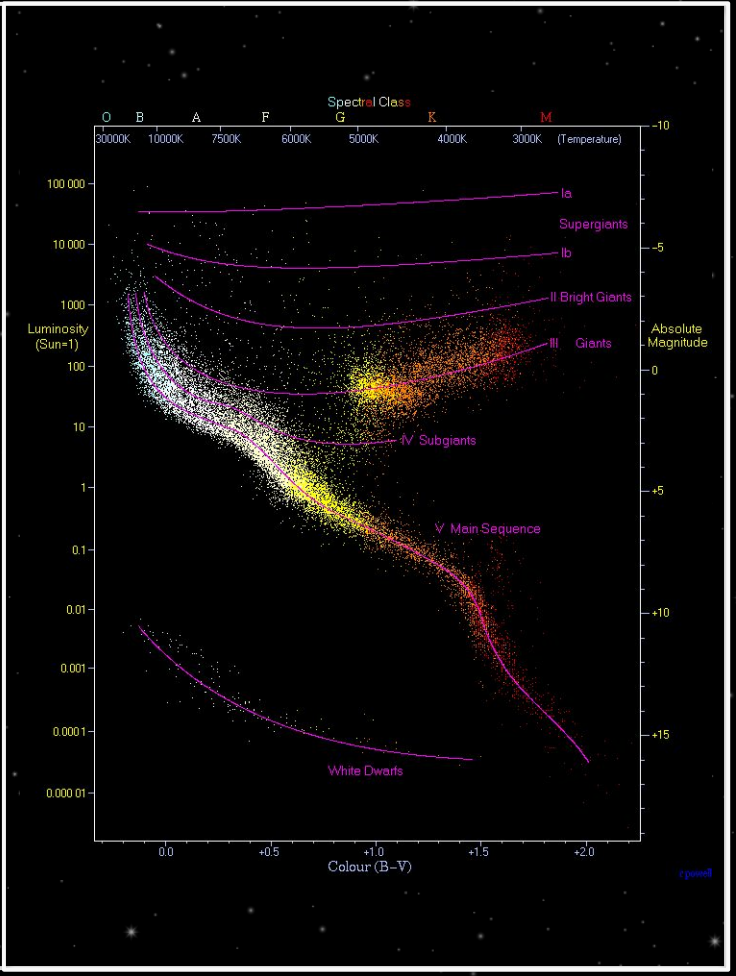
Types of stars:

- O stars (Blue giants): Extremely hot, massive stars, often blue or blue-white in color, with surface temperatures exceeding 30,000 K.
- B stars (Blue-white stars): Hot, blue stars with surface temperatures ranging from 10,000 to 30,000 K, and they are typically larger and more luminous than the Sun.



Stellar Classification

- A stars (White dwarfs): White or bluish-white stars with surface temperatures between 7,500 and 10,000 K, shining with a steady luminosity.
- F stars (Yellow-white dwarfs): Main-sequence stars with surface temperatures ranging from 6,000 to 7,500 K, similar to our Sun but slightly hotter and more luminous.
- G stars (Yellow dwarfs): Yellow main-sequence stars like our Sun, with surface temperatures between 5,200 and 6,000 K and a stable fusion of hydrogen into helium in their cores.
- K stars (Orange dwarfs): Orange main-sequence stars with surface temperatures between 3,500 and 5,200 K, cooler and less luminous than the Sun.
- M stars (Red dwarfs): Cool, red main-sequence stars with surface temperatures below 3,500 K, the most common type of star in the Milky Way



Dataset

	A	B	C	D	E	F	G
1	Temperature (K)	Luminosity(L/Lo)	Radius(R/Ro)	Absolute magnitude(Mv)	Star type	Star color	Spectral Class
2	3068	0.0024	0.17	16.12	0	Red	M
3	3042	0.0005	0.1542	16.6	0	Red	M
4	2600	0.0003	0.102	18.7	0	Red	M
5	2800	0.0002	0.16	16.65	0	Red	M
6	1939	0.000138	0.103	20.06	0	Red	M
7	2840	0.00065	0.11	16.98	0	Red	M
8	2637	0.00073	0.127	17.22	0	Red	M
9	2600	0.0004	0.096	17.4	0	Red	M
10	2650	0.00069	0.11	17.45	0	Red	M
11	2700	0.00018	0.13	16.05	0	Red	M
12	3600	0.0029	0.51	10.69	1	Red	M
13	3129	0.0122	0.3761	11.79	1	Red	M
14	3134	0.0004	0.196	13.21	1	Red	M
15	3628	0.0055	0.393	10.48	1	Red	M
16	2650	0.0006	0.14	11.782	1	Red	M
17	3340	0.0038	0.24	13.07	1	Red	M
18	2799	0.0018	0.16	14.79	1	Red	M
19	3692	0.00367	0.47	10.8	1	Red	M
20	3192	0.00362	0.1967	13.53	1	Red	M
21	3441	0.039	0.351	11.18	1	Red	M
22	25000	0.056	0.0084	10.58	2	Blue White	B
23	7740	0.00049	0.01234	14.02	2	White	A
24	7220	0.00017	0.011	14.23	2	White	F
25	8500	0.0005	0.01	14.5	2	White	A
26	16500	0.013	0.014	11.89	2	Blue White	B
27	12990	0.000085	0.00984	12.23	2	Yellowish White	F

Factors included:

1. Temperature: Defines spectral characteristics, revealing composition and surface conditions, essential for stellar classification.

2. Luminosity: Total energy output, indicating evolutionary stage, determined by surface temperature and radius.

3. Radius: Physical size affects luminosity, surface temperature, and spectral lines, revealing evolutionary phase and mass-loss processes.

4. Absolute Magnitude: Standardized brightness at 10 parsecs, crucial for comparing luminosity, and determining evolutionary stage.

Insights that can be inferred from the project

1. Evolutionary stage and Age

Stellar classification helps us determine the evolutionary stage of a star.
For example, main sequence stars like our Sun are in the prime of their lives, while giant and supergiant stars are in later stages of evolution.

2. Chemical Composition

Analysis of a star's spectrum reveals information about its chemical composition. Spectral lines indicate the presence of specific elements in a star's atmosphere, providing insights into stellar nucleosynthesis and the chemical enrichment of the universe

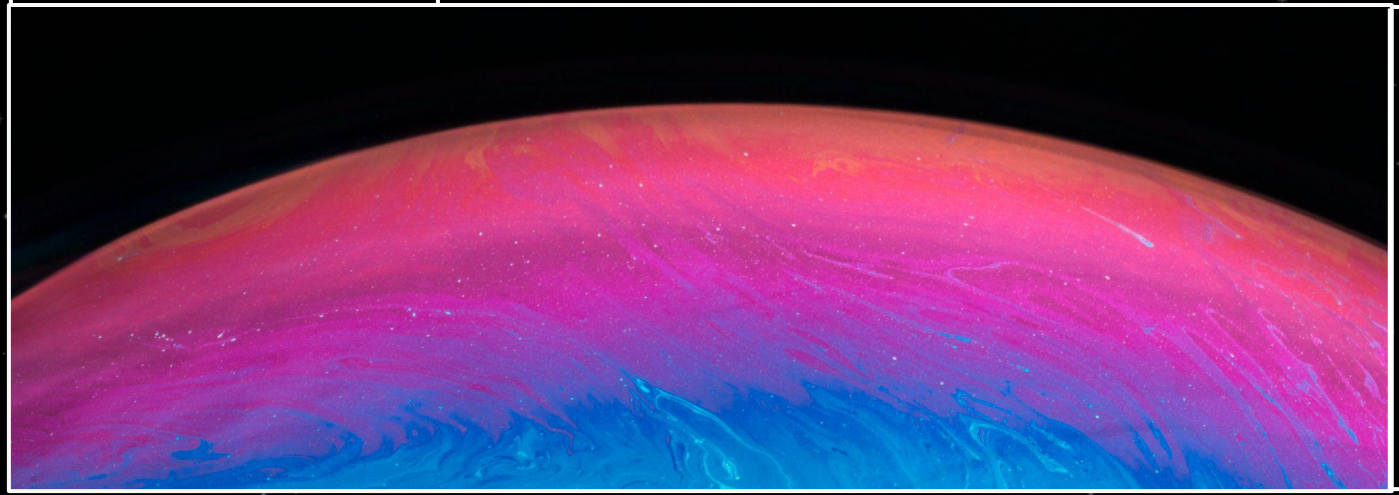
3. Stellar Dynamics and multiplicity

Spectral classification can reveal whether a star is part of a binary or multiple star system. It can also help us determine its distance from Earth and its motion relative to us. This is useful for studying the dynamics of stellar systems, such as star clusters and galaxies.

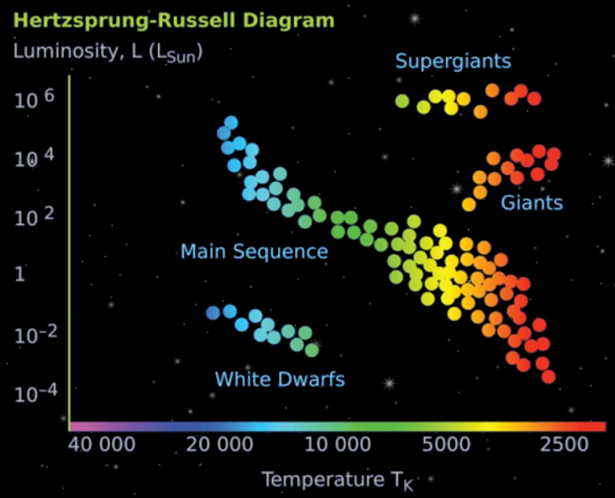
02

Modelling

Exploring optimal machine learning algorithms to classify stars



Stellar Classification



ML models that can be used:

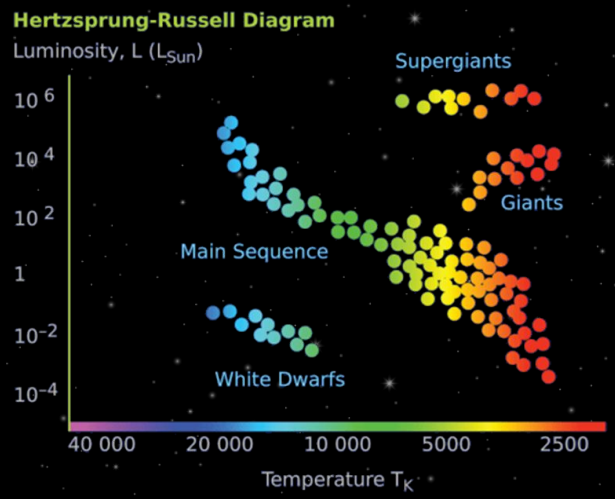
1. **KNN (K Nearest Neighbours):** We can classify stars by measuring their proximity to known stars based on features like luminosity and temperature and assigns a star to the class of its nearest neighbors in the feature space, making it suitable for classifying stars with similar characteristics.

2. **Decision trees:** We can classify stars using factors like radius to split the data based on these features and to create a hierarchical structure that can be used to distinguish between different types of stars

3. **Random forest:** Creating multiple decision trees and making a poll system with them

4. **Support Vector Machine:** Finding the optimal hyperplane that separates data points into different classes while maximizing the margin between classes.

Stellar Classification



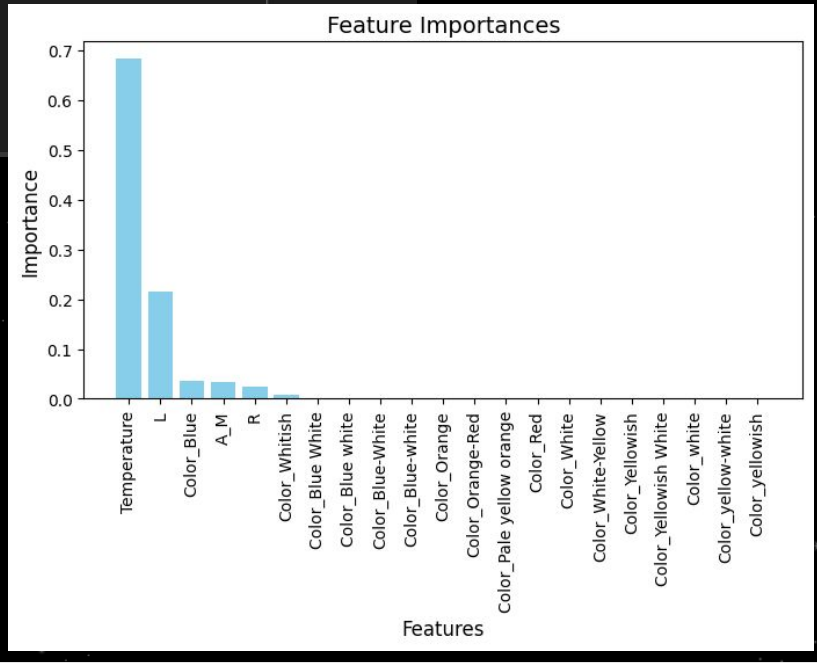
5. **Adaboost (Adaptive Boosting):** We can create multiple SVMs recursively by updating weights of data points to correct previous misclassifications and then combining them into a single region

6. **K-Means**: Clustering data points into a predetermined number of clusters by iteratively updating cluster centroids to minimize the within-cluster variance.

7. **Naive Bayes:** A probabilistic classifier that applies Bayes' theorem with the naive assumption of feature independence to predict the class label of a given data point.

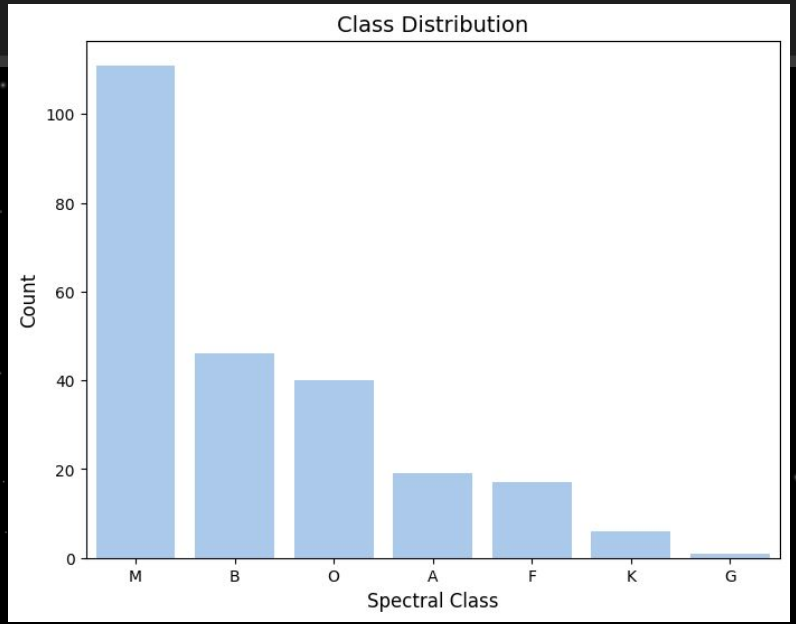

DATASET VISUALISATION

```python
# Plot Feature Importances
plt.figure(figsize=(8, 4))
importances = clf.feature_importances_
features = X.columns
indices = sorted(range(len(importances)), key=lambda i: importances[i], reverse=True)
plt.bar(range(X.shape[1]), importances[indices], align='center', color='skyblue')
plt.xticks(range(X.shape[1]), [features[i] for i in indices], rotation=90, fontsize=10)
plt.xlabel('Features', fontsize=12)
plt.ylabel('Importance', fontsize=12)
plt.title('Feature Importances', fontsize=14)
plt.show()
```



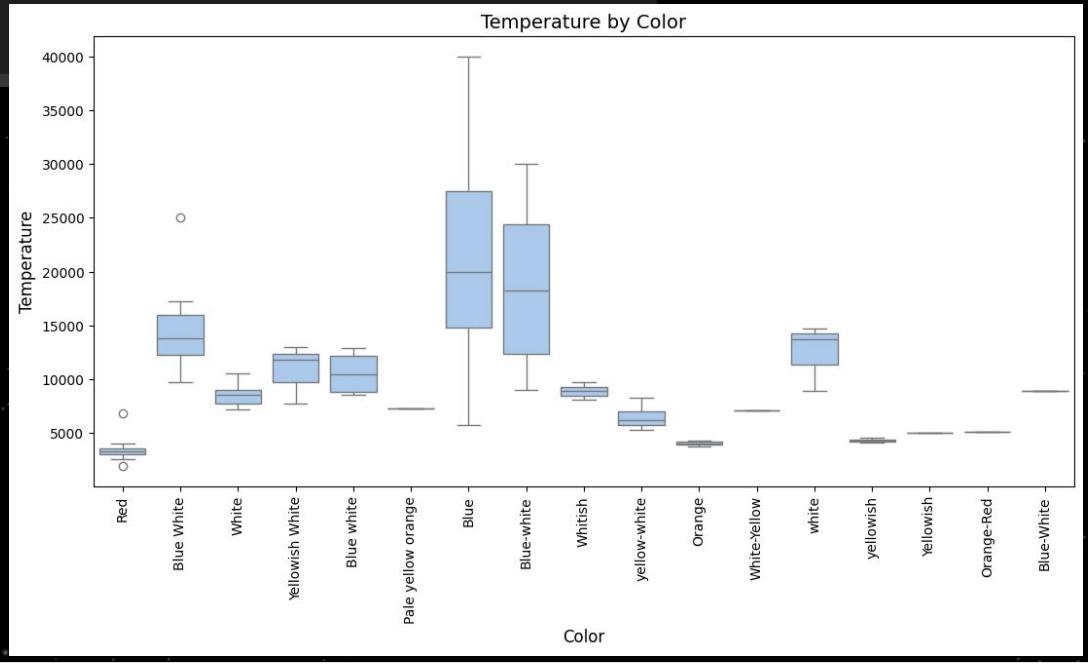
0s

```
# Plot Class Distribution
plt.figure(figsize=(8, 6))
sns.countplot(x=y, order=y.value_counts().index)
plt.xlabel('Spectral Class', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Class Distribution', fontsize=14)
plt.show()
```

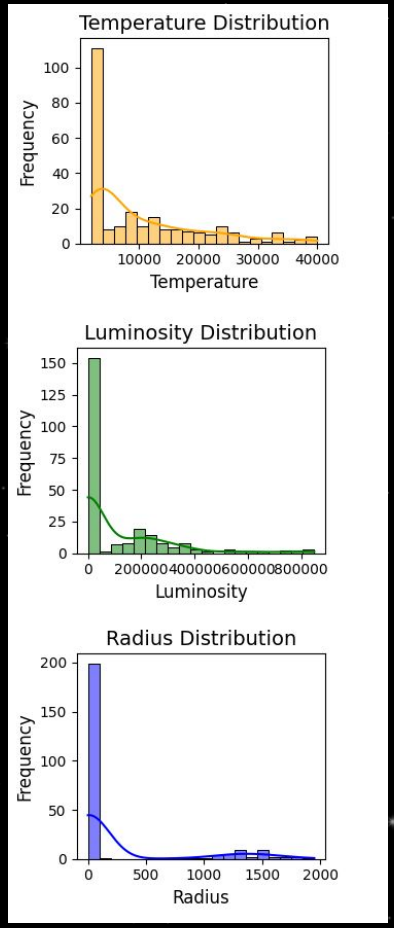


0s

```
# Plot Temperature by Color
plt.figure(figsize=(9, 6))
sns.boxplot(x='Color', y='Temperature', data=data)
plt.xlabel('Color', fontsize=12)
plt.ylabel('Temperature', fontsize=12)
plt.title('Temperature by Color', fontsize=14)
plt.xticks(rotation=90)
plt.show()
```

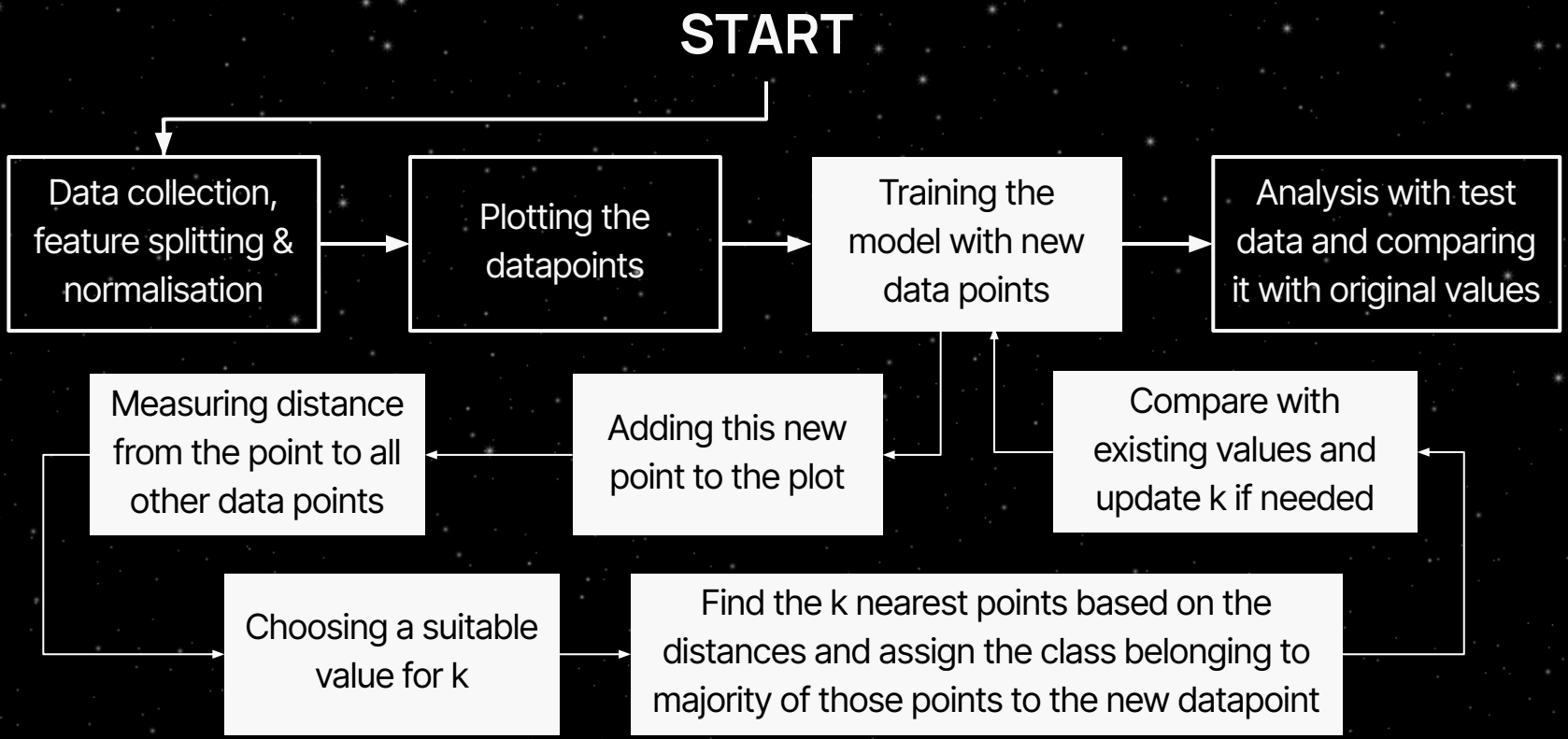


```python
# Plot Temperature, Luminosity, and Radius Distribution
plt.figure(figsize=(10, 3))
plt.subplot(1, 3, 1)
sns.histplot(data['Temperature'], bins=20, kde=True, color='orange')
plt.xlabel('Temperature', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Temperature Distribution', fontsize=14)


plt.subplot(1, 3, 2)
sns.histplot(data['L'], bins=20, kde=True, color='green')
plt.xlabel('Luminosity', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Luminosity Distribution', fontsize=14)

plt.subplot(1, 3, 3)
sns.histplot(data['R'], bins=20, kde=True, color='blue')
plt.xlabel('Radius', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Radius Distribution', fontsize=14)

plt.tight_layout()
plt.show()
```




KNN CLASSIFIER

KNN Classifier



```
[1] #importing packages
    import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.preprocessing import StandardScaler,LabelEncoder
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import accuracy_score
    import matplotlib.pyplot as plt
    from sklearn.model_selection import cross_val_score
```

```
[2] data = pd.read_csv('Stars.csv')
```

```
    print("Dataset shape:", data.shape)
    print("First few rows of the dataset:\n", data.head())
```

```
Dataset shape: (240, 6)
First few rows of the dataset:
   Temperature        L       R    A_M Spectral_Class  Type
0         3068  0.002400  0.1700  16.12              M     0
1         3042  0.000500  0.1542  16.60              M     0
2         2600  0.000300  0.1020  18.70              M     0
3         2800  0.000200  0.1600  16.65              M     0
4         1939  0.000138  0.1030  20.06              M     0
```

```
[4] X = data.drop(columns=['Temperature','L'])
    y = data[['Spectral_Class']]
```

```
[5] label_encoder = LabelEncoder()
    y_encoded = label_encoder.fit_transform(y)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_label.py:116: DataConversionWarning: A column-vector y was passed when a 1d array was exp
  y = column_or_1d(y, warn=True)
```

```
[6] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[7] X_train = X_train.drop(columns=['Spectral_Class'])
    X_test = X_test.drop(columns=['Spectral_Class'])
```

```
[8] scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
```

```
[9] knn_classifier = KNeighborsClassifier(n_neighbors=7)
```

```
[10] y_train.dtypes
```

```
Spectral_Class    object
dtype: object
```

```
knn_classifier.fit(X_train_scaled, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/neighbors/_classification.py:215: DataConversionWarning: A column-vector y was passed when a 1d array was
  return self._fit(X, y)
```

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=7)

```
[12] y_pred = knn_classifier.predict(X_test_scaled)
```

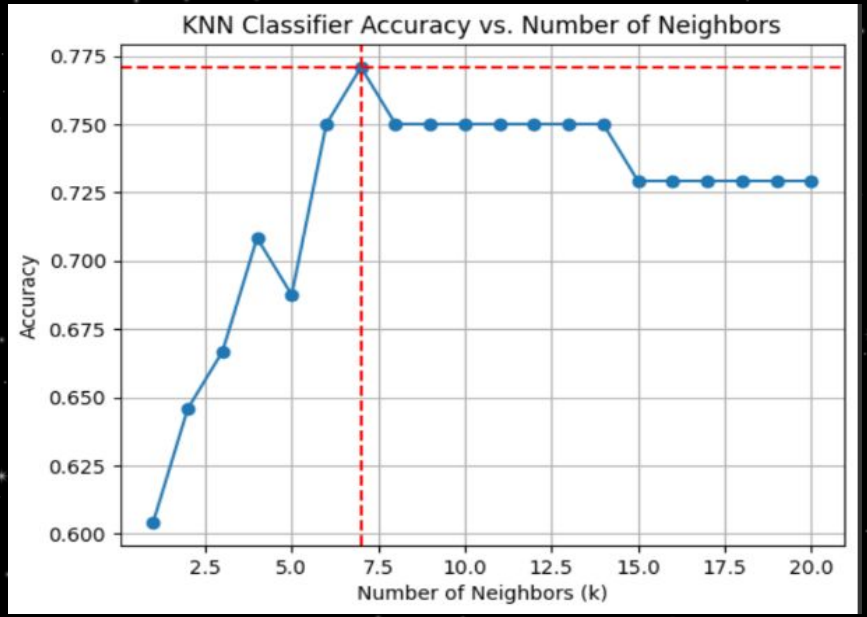
```
[13] accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy)
```

```
Accuracy: 0.7708333333333334
```

```
[14] k_values = range(1, 21)  # Testing k values from 1 to 20
     accuracy_scores = []
```

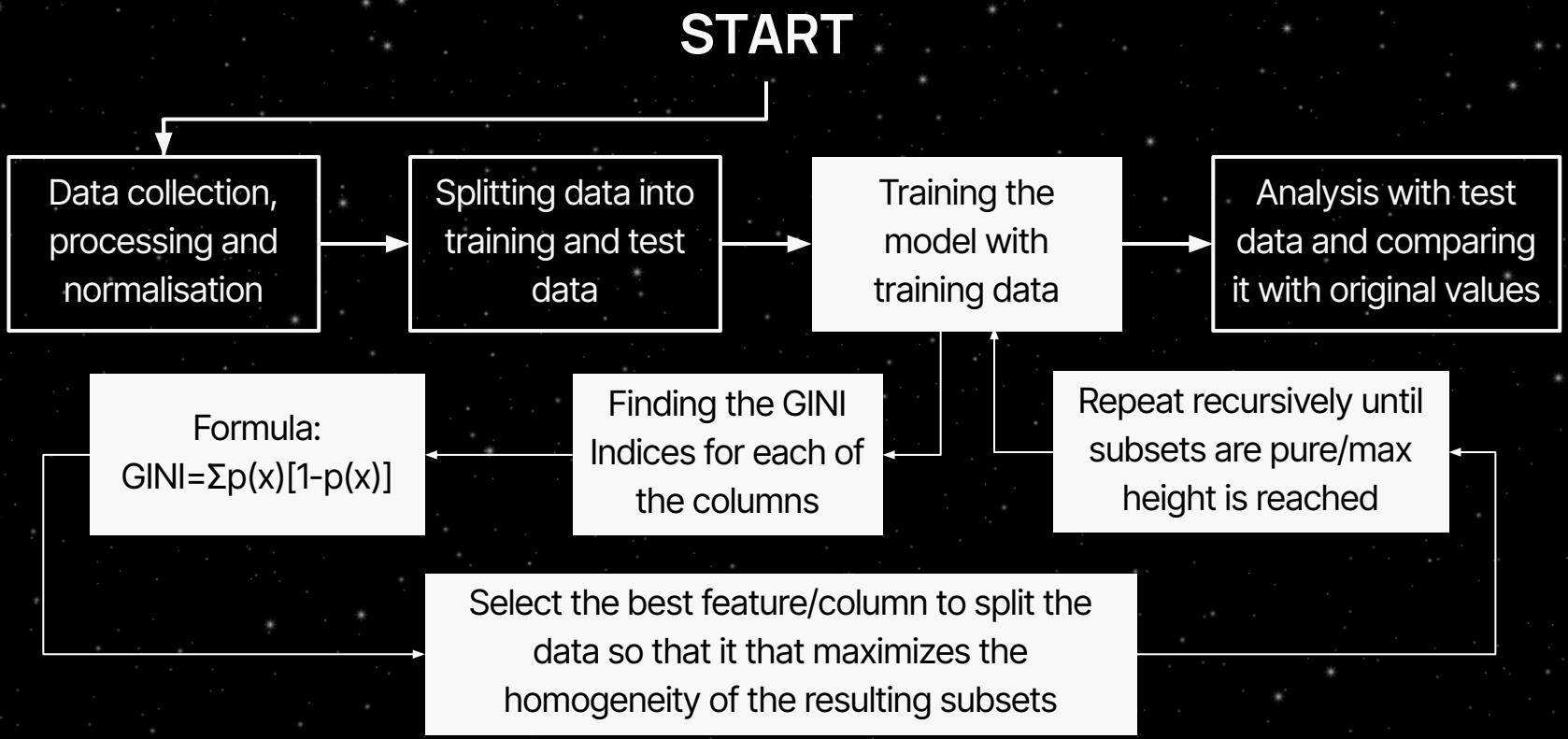
```
[15] for k in k_values:
         knn_classifier = KNeighborsClassifier(n_neighbors=k)
         knn_classifier.fit(X_train_scaled, y_train)
         y_pred = knn_classifier.predict(X_test_scaled)
         accuracy = accuracy_score(y_test, y_pred)
         accuracy_scores.append(accuracy)
```

```
plt.plot(k_values, accuracy_scores, marker='o')
plt.title('KNN Classifier Accuracy vs. Number of Neighbors')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.axvline(x=7, color='r', linestyle='--')
plt.axhline(y=accuracy_scores[6], color='r', linestyle='--')
plt.show()
```




DECISION TREE

Decision trees



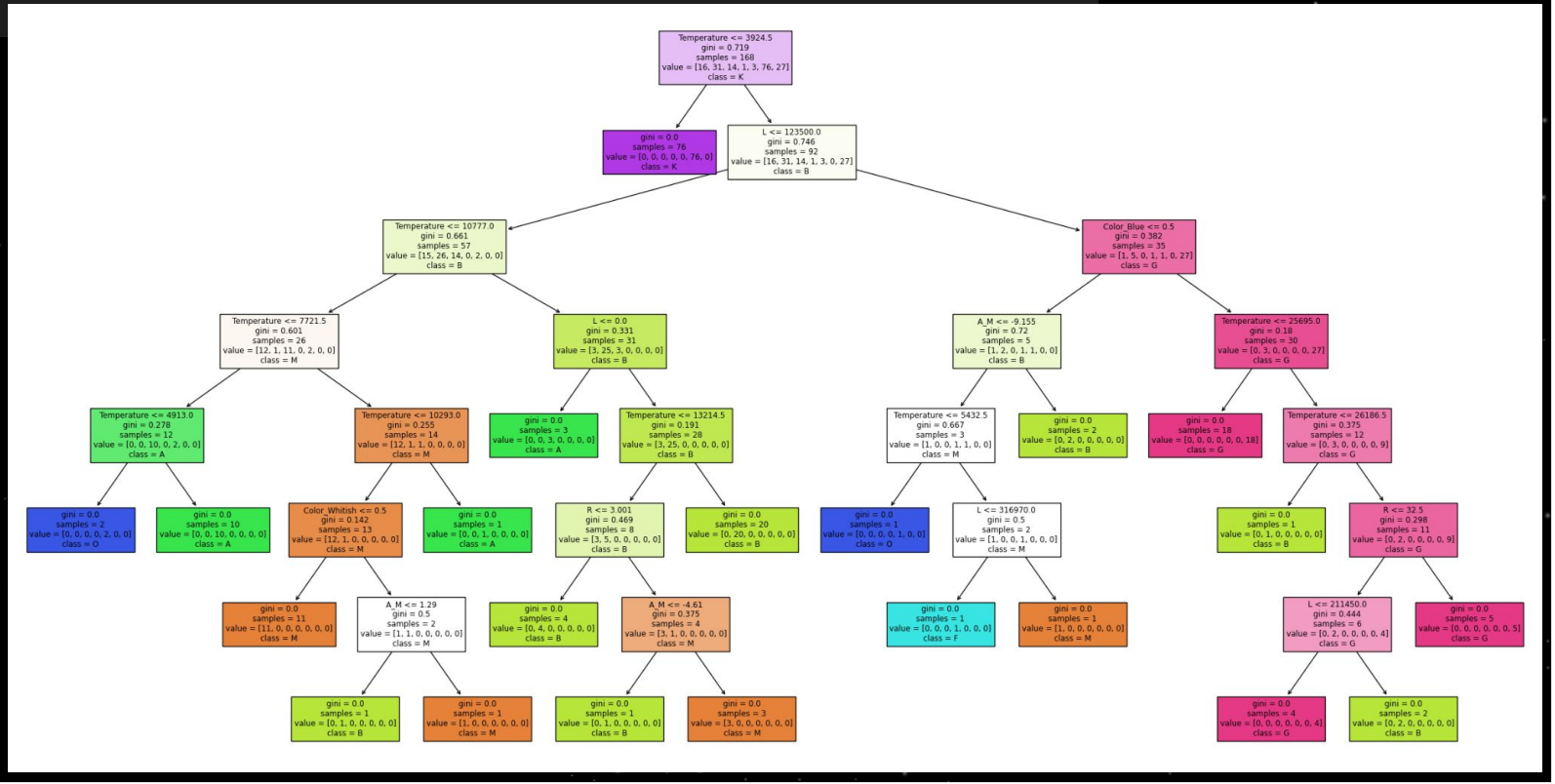
```
[2]  import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.tree import DecisionTreeClassifier, plot_tree
     from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[3]  # Load the dataset
     data = pd.read_csv("Stars.csv")
```

```
[4]  # Split the dataset into features and target variable
     X = data[['Temperature', 'L', 'R', 'A_M', 'Color']]  # Features
     y = data['Spectral_Class']  # Target variable

     # Convert categorical variables into dummy/indicator variables
     X = pd.get_dummies(X)
```

```
[5]  # Split dataset into training set and test set
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

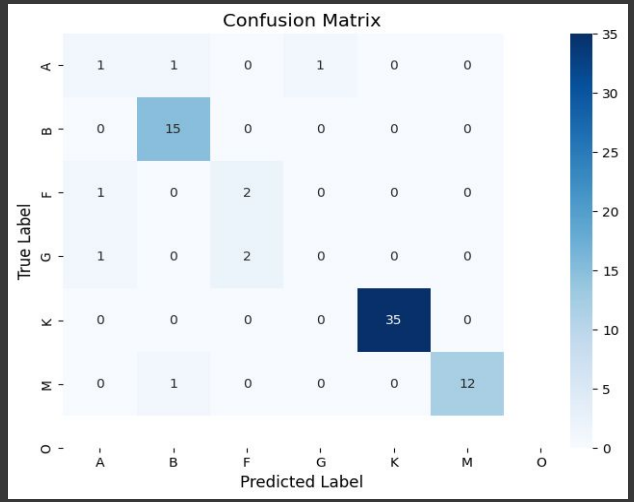
```
# Plot the Decision Tree
plt.figure(figsize=(30, 15))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=y.unique())
plt.show()
```



```
[23] clf = DecisionTreeClassifier()
     clf = clf.fit(X_train, y_train)                # Train Decision Tree Classifier
     y_pred = clf.predict(X_test)                   # Predict the response for test dataset
     accuracy = accuracy_score(y_test, y_pred)      # Model Accuracy
     conf_matrix = confusion_matrix(y_test, y_pred) # Confusion Matrix
```

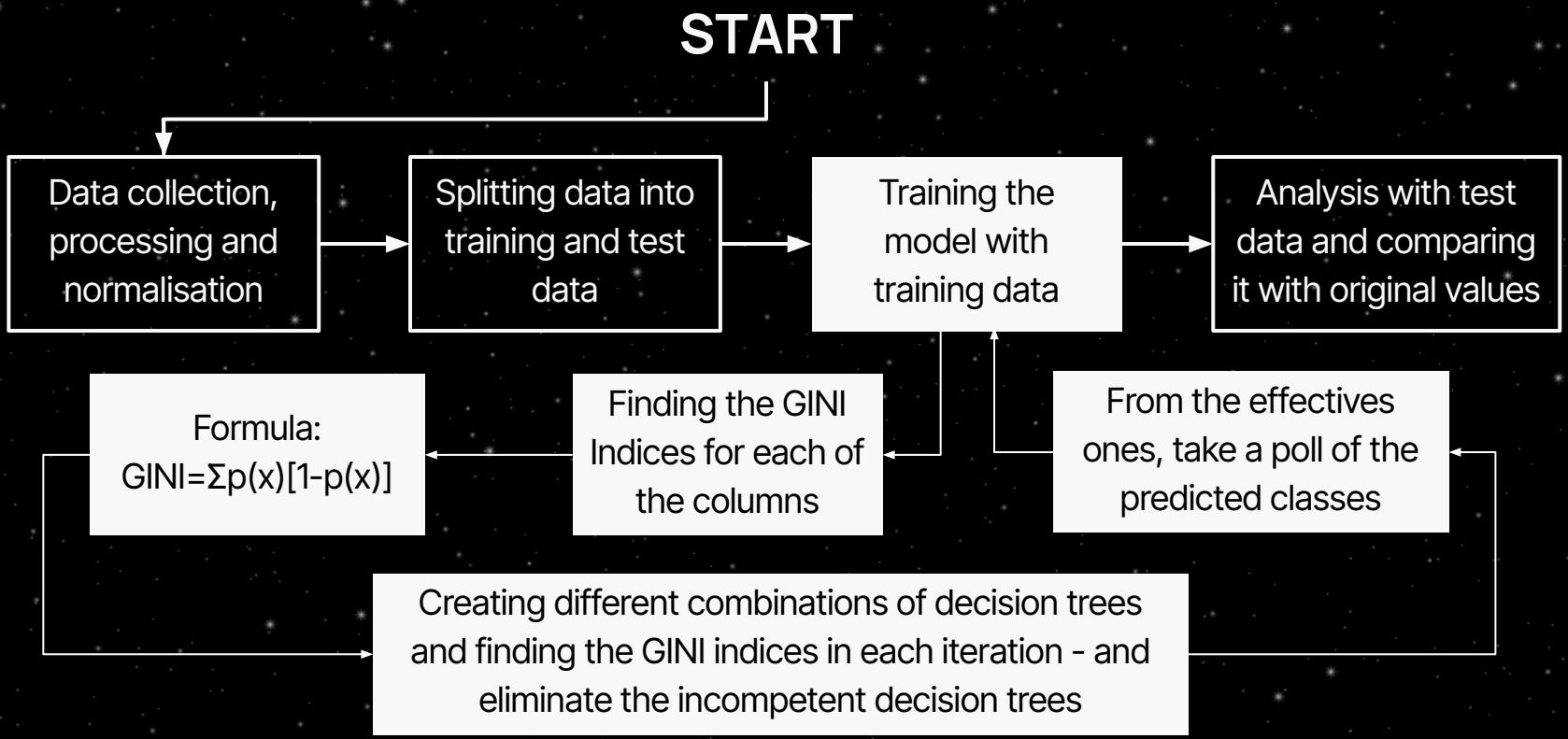
```
# Print Accuracy and Classification Report
print("Accuracy:", accuracy)
print("\nClassification Report:")
print(class_report)
```

```
Accuracy: 0.9027777777777778

Classification Report:
              precision    recall  f1-score   support

           A       0.67      0.67      0.67         3
           B       0.82      0.93      0.87        15
           F       0.50      0.67      0.57         3
           G       0.00      0.00      0.00         0
           K       1.00      0.33      0.50         3
           M       1.00      1.00      1.00        35
           O       1.00      0.85      0.92        13

    accuracy                           0.90        72
   macro avg       0.71      0.64      0.65        72
weighted avg       0.93      0.90      0.91        72
```




RANDOM FOREST

Random forest



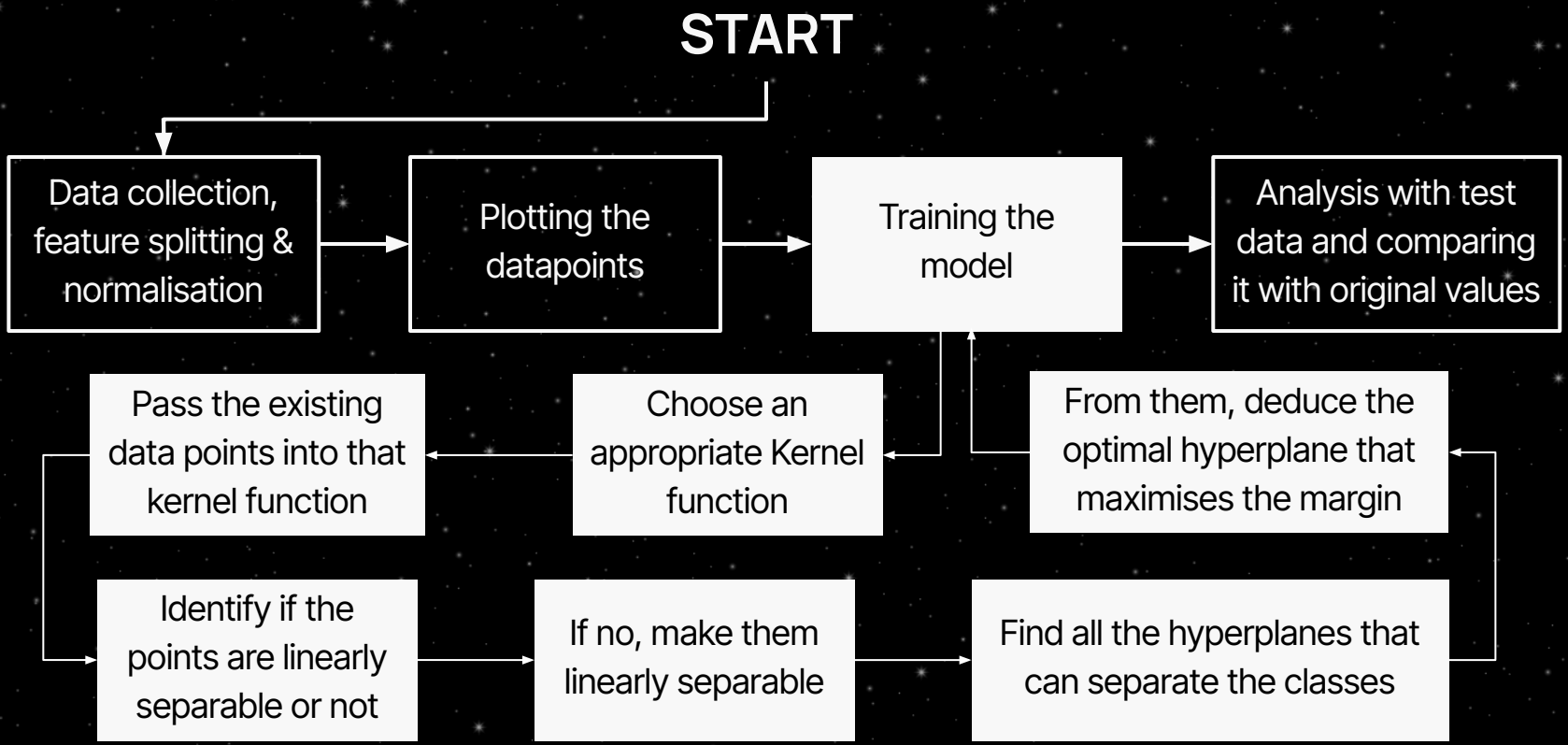
```python
# Import necessary libraries
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load the dataset (replace with your data path)
data = pd.read_csv("/content/Stars1.csv")

# Separate features and target variables with informative names
features = data.drop(columns=['Spectral_Class', 'Type'])
target_spectral_class = data['Spectral_Class']  # Target variable for Spectral_Class prediction
target_type = data['Type']  # Target variable for Type prediction

# Perform one-hot encoding on the 'Color' column
encoded_data = pd.get_dummies(features['Color'])

# Concatenate the original DataFrame (features) with the encoded columns
result = pd.concat([features, encoded_data], axis=1)

# Drop the original 'Color' column
result.drop('Color', axis=1, inplace=True)

# Encode the target variable 'Type' using LabelEncoder
label_encoder_type = LabelEncoder()
target_type_encoded = label_encoder_type.fit_transform(target_type)

# Initialize and fit the random forest classifiers (one for Spectral_Class and another for Type)
rf_classifier_spectral = RandomForestClassifier()
rf_classifier_spectral.fit(result, target_spectral_class)
rf_classifier_type = RandomForestClassifier()
rf_classifier_type.fit(result, target_type_encoded)
```

```
# Calculate accuracy of the models
accuracy_spectral_class = accuracy_score(target_spectral_class, rf_classifier_spectral.predict(result))
accuracy_type = accuracy_score(target_type_encoded, rf_classifier_type.predict(result))

# Print the accuracies in terms of percentile
print("Accuracy of Spectral Class prediction: {:.2f}%".format(accuracy_spectral_class * 100))
print("Accuracy of Type prediction: {:.2f}%".format(accuracy_type * 100))

# Prompt the user for input
temperature = float(input("Enter temperature: "))
l = float(input("Enter L: "))
r = float(input("Enter R: "))
a_m = float(input("Enter A_M: "))
color = input("Enter Color: ")

# Create a DataFrame with the user input
new_data = pd.DataFrame([[temperature, l, r, a_m, color]], columns=['Temperature', 'L', 'R', 'A_M', 'Color'])

# Perform one-hot encoding on the 'Color' column if needed
new_encoded_data = pd.get_dummies(new_data['Color'])

# Concatenate the original DataFrame (new_data) with the encoded columns
new_result = pd.concat([new_data, new_encoded_data], axis=1)

# Drop the original 'Color' column if needed
new_result.drop('Color', axis=1, inplace=True)

# Ensure all the required columns are present in the new_result DataFrame
# Add any missing columns with all zero values
missing_cols = set(result.columns) - set(new_result.columns)
for col in missing_cols:
    new_result[col] = 0

# Reorder the columns to match the order used during training
new_result = new_result[result.columns]

# Make predictions for Spectral_Class and Type separately
predicted_spectral_class = rf_classifier_spectral.predict(new_result)[0]
predicted_type_encoded = rf_classifier_type.predict(new_result)[0]

# Inverse transform the predicted Type back to original labels
predicted_type = label_encoder_type.inverse_transform([predicted_type_encoded])[0]

# Print the predicted Spectral Class and Type
print("Predicted Spectral Class:", predicted_spectral_class)
print("Predicted Type:", predicted_type)
```

```
Accuracy of Spectral Class prediction: 100.00%
Accuracy of Type prediction: 100.00%
Enter temperature: 1831
Enter L: 0.03118
Enter R: 0.31
Enter A_M: 18.31
Enter Color: Blue
Predicted Spectral Class: M
Predicted Type: 0
```


SUPPORT VECTOR
MACHINES

Support Vector Machines



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import classification_report, confusion_matrix
```

```
data = pd.read_csv('/content/Stars_data.csv')
```

```
label_encoder = LabelEncoder()
data['Spectral_Class'] = label_encoder.fit_transform(data['Spectral_Class'])
data['Type'] = label_encoder.fit_transform(data['Type'])
```

```
X = data[['Temperature','A_M']]
y = data['Type']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)
```

```
svm_classifier.fit(X_train_scaled, y_train)
```

```
SVC
SVC(kernel='linear', random_state=42)
```

```
y_pred_train = svm_classifier.predict(X_train_scaled)
y_pred_test = svm_classifier.predict(X_test_scaled)
```

```
train_accuracy = accuracy_score(y_train, y_pred_train)
test_accuracy = accuracy_score(y_test, y_pred_test)
```

```
print("Training Accuracy:", train_accuracy)
print("Testing Accuracy:", test_accuracy)

print("\nClassification Report:")
print(classification_report(y_test, y_pred_test))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_test))
```

```
Training Accuracy: 0.9270833333333334
Testing Accuracy: 0.8958333333333334

Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94         8
           1       1.00      0.86      0.92         7
           2       1.00      1.00      1.00         6
           3       0.89      1.00      0.94         8
           4       0.70      0.88      0.78         8
           5       1.00      0.73      0.84        11

    accuracy                           0.90        48
   macro avg       0.91      0.91      0.90        48
weighted avg       0.91      0.90      0.90        48

Confusion Matrix:
[[8 0 0 0 0 0]
 [1 6 0 0 0 0]
 [0 0 6 0 0 0]
 [0 0 0 8 0 0]
 [0 0 0 1 7 0]
 [0 0 0 0 3 8]]
```

```
[12] import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.datasets import make_classification
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
```

```
[13] x_min, x_max = X_train_scaled[:, 0].min() - 1, X_train_scaled[:, 0].max() + 1
     y_min, y_max = X_train_scaled[:, 1].min() - 1, X_train_scaled[:, 1].max() + 1
     xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                          np.arange(y_min, y_max, 0.1))
```

```
[ ]  import numpy as np
     import matplotlib.pyplot as plt
     from sklearn.datasets import make_classification
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
```

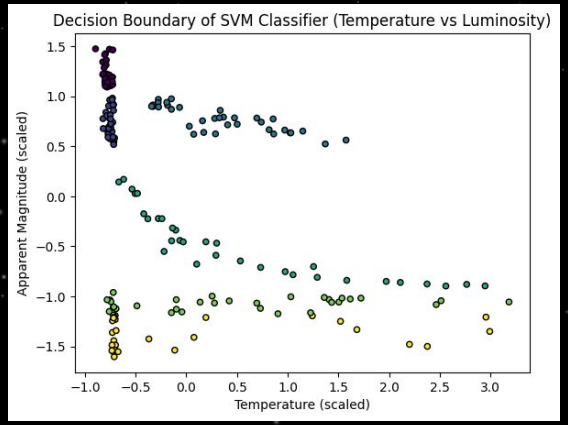
```
[ ]  x_min, x_max = X_train_scaled[:, 0].min() - 1, X_train_scaled[:, 0].max() + 1
     y_min, y_max = X_train_scaled[:, 1].min() - 1, X_train_scaled[:, 1].max() + 1
     xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                          np.arange(y_min, y_max, 0.1))
```

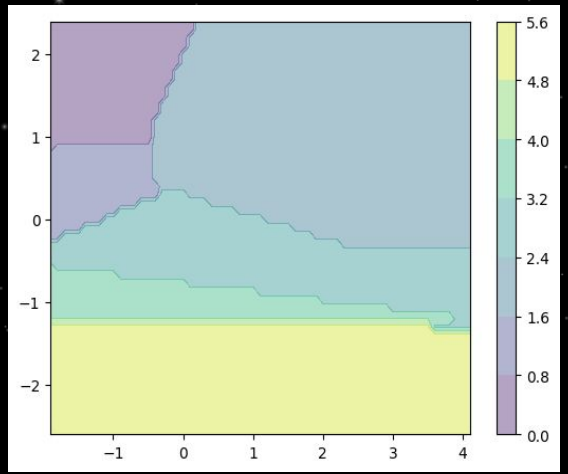
```
[ ]  svm_classifier.fit(X_train_scaled[:, :2], y_train)
```

```
[ ]  Z = svm_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
     Z = Z.reshape(xx.shape)
```

```
[ ]  plt.contourf(xx, yy, Z, alpha=0.4)
     plt.colorbar()
```

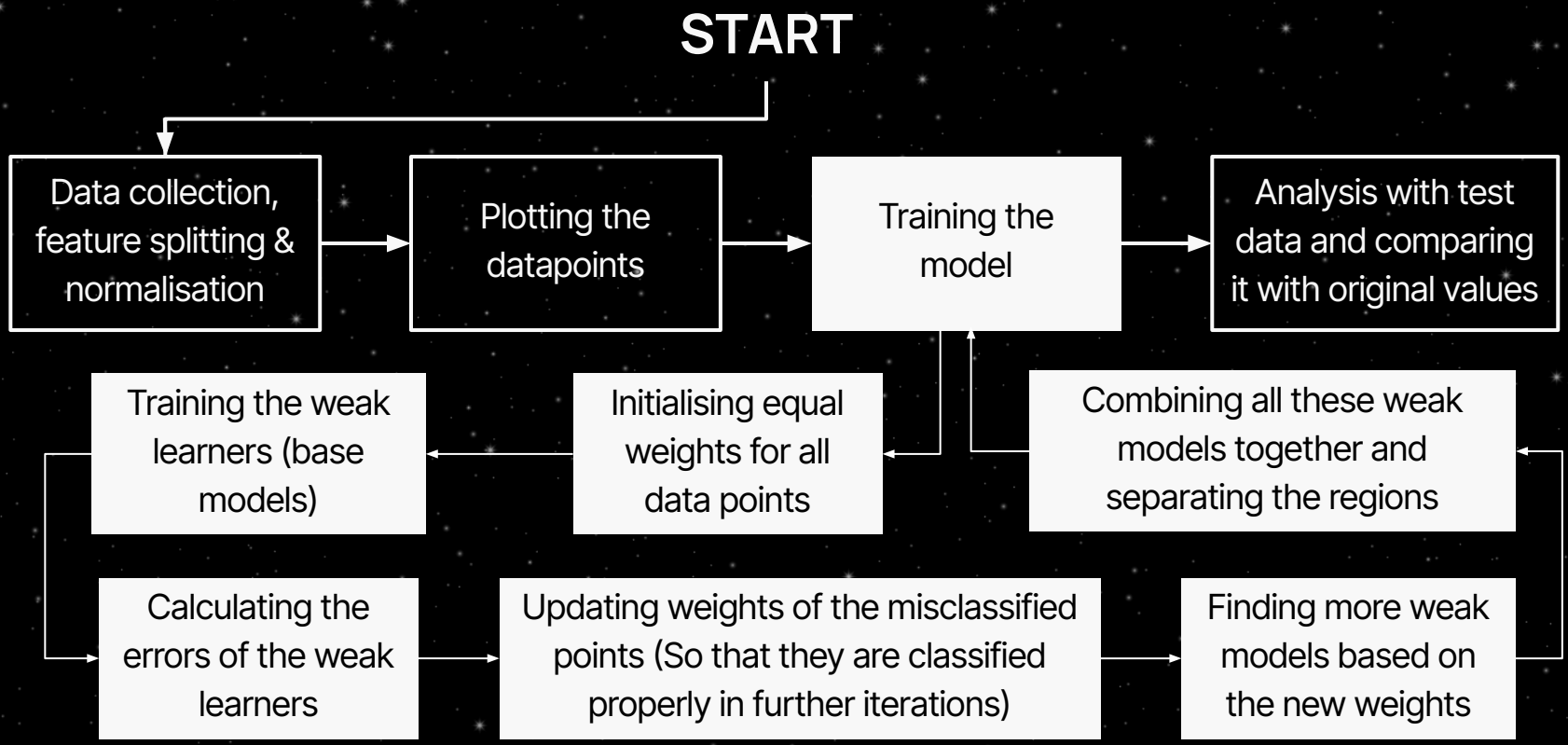
```
     plt.scatter(X_train_scaled[:, 0], X_train_scaled[:, 1], c=y_train, s=20, edgecolor='k')
     plt.xlabel('Temperature (scaled)')
     plt.ylabel('Apparent Magnitude (scaled)')
     plt.title('Decision Boundary of SVM Classifier (Temperature vs Luminosity)')
     plt.show()
```






ADABOOST

Adaboost


START
Data collection, feature splitting & normalisation
Plotting the datapoints
Training the model
Analysis with test data and comparing it with original values
Training the weak learners (base models)
Initialising equal weights for all data points
Combining all these weak models together and separating the regions
Calculating the errors of the weak learners
Updating weights of the misclassified points (So that they are classified properly in further iterations)
Finding more weak models based on the new weights

```
[3] import pandas as pd
    from sklearn.model_selection import train_test_split
    from sklearn.ensemble import AdaBoostClassifier
    from sklearn.metrics import accuracy_score, confusion_matrix
    import matplotlib.pyplot as plt
    import seaborn as sns
    from sklearn.decomposition import PCA
    from sklearn.preprocessing import MinMaxScaler
```

```
[4] #Loading the dataset
    data = pd.read_csv('Stars.csv')
```

```
[5] #Assigning input and output variables
    X = data[['Temperature', 'A_M', 'R']]
    y = data['Spectral_Class']
```

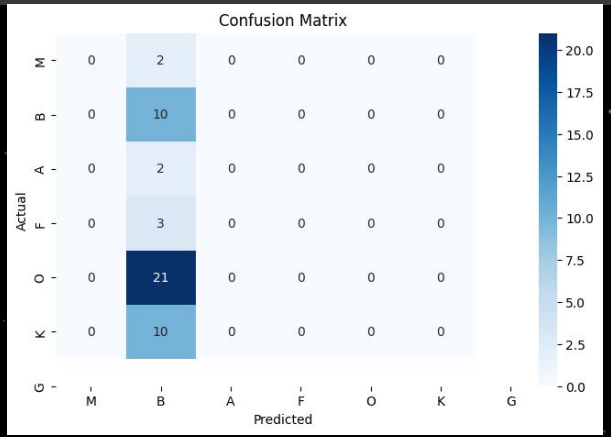
```
[6] #Splitting the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

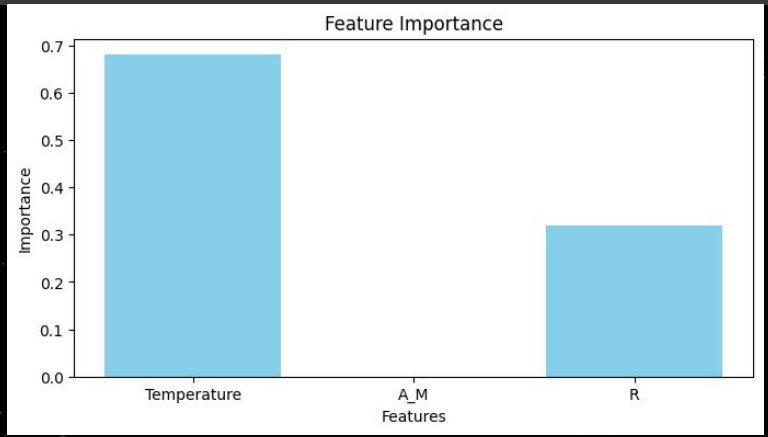
```
    #Adjusting weightage of features after normalizing the features
    scaler = MinMaxScaler()
    X_scaled = scaler.fit_transform(X)

    # Assign weights to features
    weightage = {'Temperature': 0.5, 'A_M': 0.3, 'R': 0.2}
    weighted_X = X_scaled.copy()
    for feature, weight in weightage.items():
        index = X.columns.get_loc(feature)
        weighted_X[:, index] *= weight
```

```
[ ] #AdaBoost classifier
    adaboost_clf = AdaBoostClassifier(n_estimators=50, random_state=42)
    adaboost_clf.fit(weighted_X, y)
```

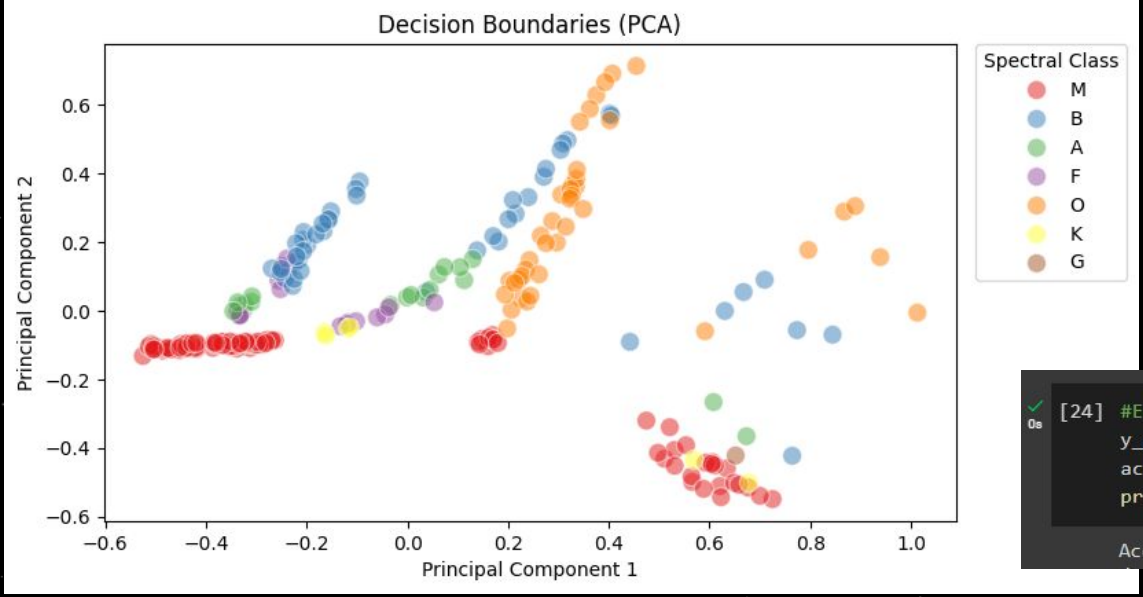
```
[21] # Confusion Matrix
     conf_matrix = confusion_matrix(y_test, y_pred)
     plt.figure(figsize=(8, 5))
     sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=data['Spectral_Class'].unique(), yticklabels=data['Spectral_Class'].unique())
     plt.xlabel('Predicted')
     plt.ylabel('Actual')
     plt.title('Confusion Matrix')
     plt.show()
```

```
     # Feature Importance Plot
     feature_importance = adaboost_clf.feature_importances_
     plt.figure(figsize=(8, 4))
     plt.bar(X.columns, feature_importance, color='skyblue')
     plt.xlabel('Features')
     plt.ylabel('Importance')
     plt.title('Feature Importance')
     plt.show()
```





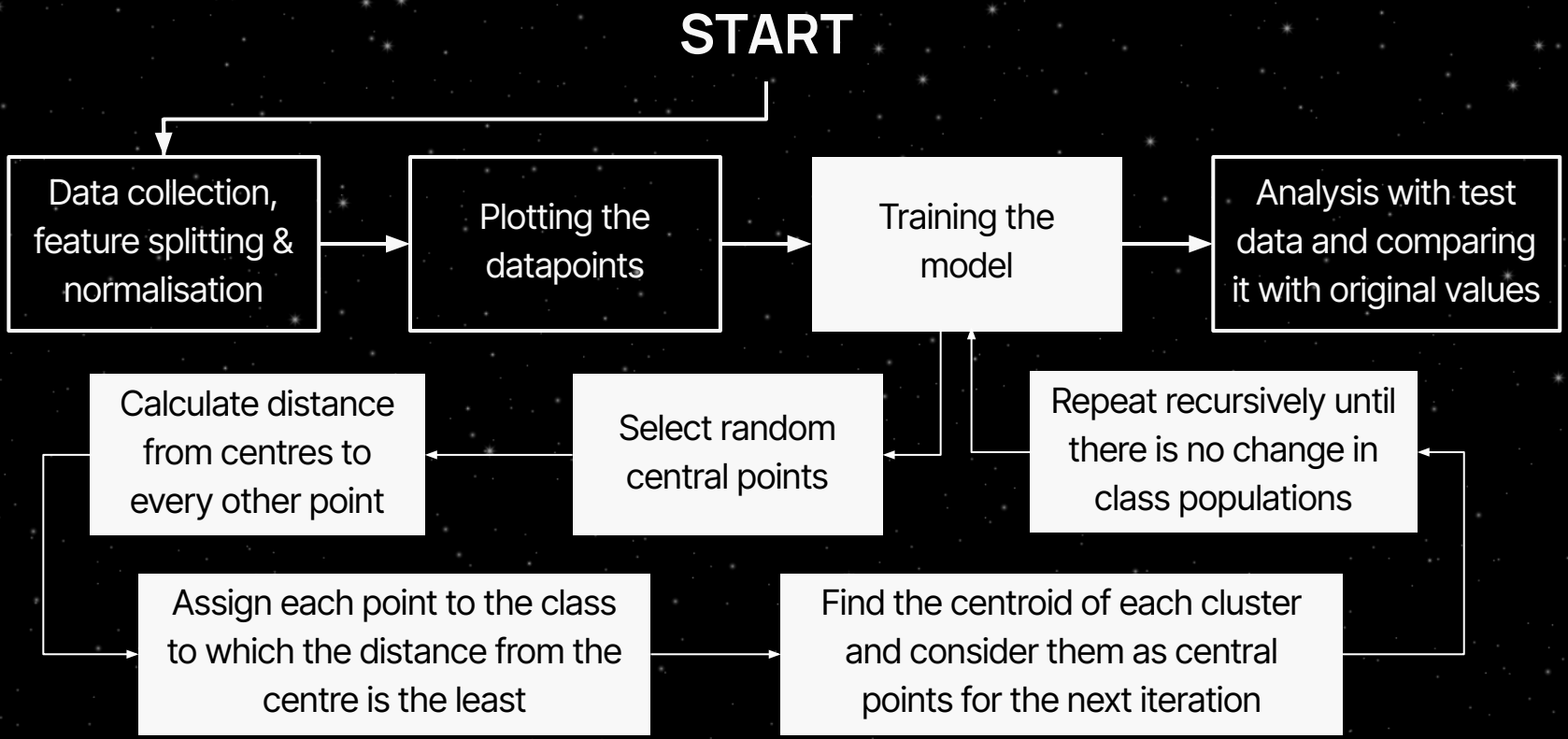
```
[23] # Visualize Decision Boundaries using PCA
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X_scaled)

     plt.figure(figsize=(8, 4.5))
     sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=y, palette='Set1', s=100, alpha=0.5)
     plt.xlabel('Principal Component 1')
     plt.ylabel('Principal Component 2')
     plt.title('Decision Boundaries (PCA)')
     plt.legend(title='Spectral Class', loc='upper right', bbox_to_anchor=(1.2, 1), borderaxespad=0.)
     plt.show()
```




```
[24] #Evaluation
     y_pred = adaboost_clf.predict(X_test)
     accuracy = accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy)

     Accuracy: 0.20833333333333334
```


K-MEANS
CLASSIFIER

K Means Classifier



```
[ ] #importing packages
    import pandas as pd
    from sklearn.cluster import KMeans
    from sklearn.preprocessing import LabelEncoder
    from sklearn.preprocessing import StandardScaler
    import matplotlib.pyplot as plt
```

```
[ ] data = pd.read_csv('/content/Stars_data.csv')
```

```
[ ] data.head()
```

	Temperature	L	R	A_M	Spectral_Class	Type
0	3068	0.002400	0.1700	16.12	M	0
1	3042	0.000500	0.1542	16.60	M	0
2	2600	0.000300	0.1020	18.70	M	0
3	2800	0.000200	0.1600	16.65	M	0
4	1939	0.000138	0.1030	20.06	M	0

```
[ ] label_encoder = LabelEncoder()
    data['Spectral_Class'] = label_encoder.fit_transform(data['Spectral_Class'])
```

```
[ ] X = data[['Temperature', 'A_M']]
```

```
[ ] scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

```
[ ] km = KMeans(init='random', n_clusters=7, n_init=10 , random_state=42)
```

```
[ ] km.fit(X_scaled)
```

```
KMeans(init='random', n_clusters=7, n_init=10, random_state=42)
```

```
[ ] print(km.inertia_)  #lowest SSE value
```

```
26.202999076912434
```

```
[ ] print(km.cluster_centers_)  #cluster centroids
```

```
[[ 0.59151371  0.71908238]
 [-0.70860632  1.00618032]
 [-0.36990589 -0.12928879]
 [ 2.63345997 -1.06731937]
 [ 0.27698679 -0.91234415]
 [-0.66906977 -1.27268416]
 [ 1.42476503 -0.95902108]]
```

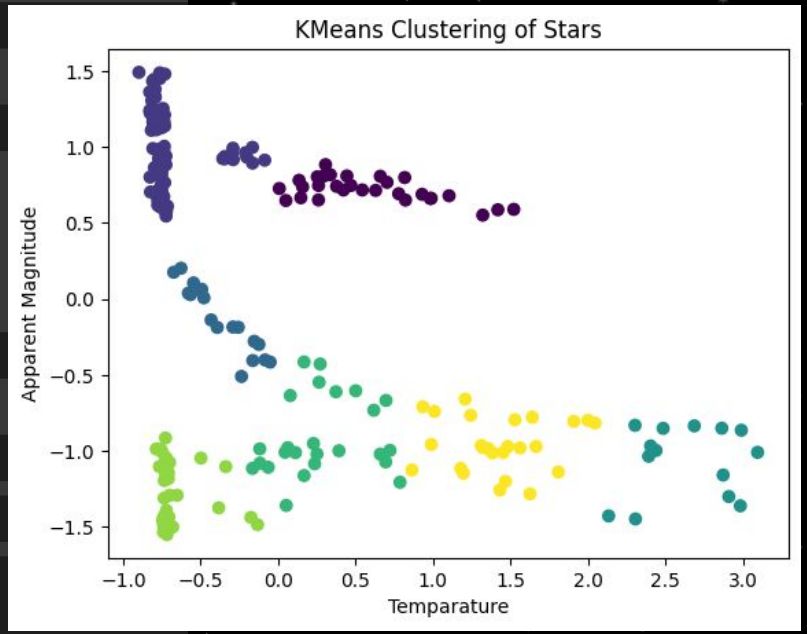
```
[ ] print(km.n_iter_)
```

```
5
```

```
[ ] labels = km.labels_
```

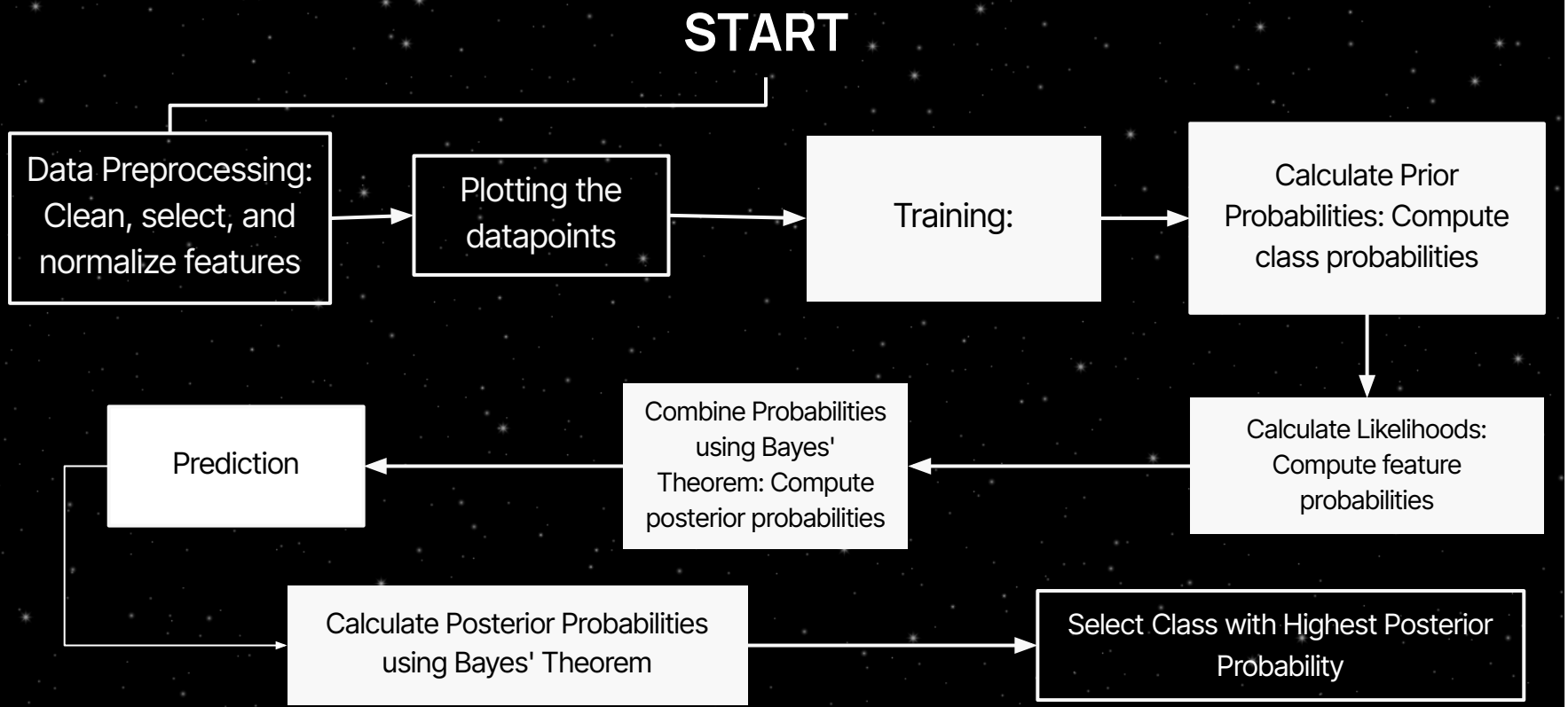
```
data['cluster'] = labels
```

```
[ ] plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis')
    plt.xlabel('Temparature')
    plt.ylabel('Apparent Magnitude') #Apparent Magnitude = Actual Luminosity / Luminosity Of Sun
    plt.title('KMeans Clustering of Stars')
    plt.show()
```




NAIVE BAYES' CLASSIFIER

Naive Bayes Classifier



```python
# Import necessary libraries
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

# Load the dataset (replace with your data path)
data = pd.read_csv("/content/Stars.csv")

# Separate features and target variables with informative names
features = data.drop(columns=['Spectral_Class', 'Type'])
target_spectral_class = data['Spectral_Class']  # Target variable for Spectral_Class prediction
target_type = data['Type']  # Target variable for Type prediction

# Perform one-hot encoding on the 'Color' column
encoded_data = pd.get_dummies(features['Color'])

# Concatenate the original DataFrame (features) with the encoded columns
result = pd.concat([features, encoded_data], axis=1)

# Drop the original 'Color' column
result.drop('Color', axis=1, inplace=True)

# Encode the target variable 'Type' using LabelEncoder
label_encoder_type = LabelEncoder()
target_type_encoded = label_encoder_type.fit_transform(target_type)

# Initialize and fit the Naive Bayes classifiers (one for Spectral_Class and another for Type)
nb_classifier_spectral = GaussianNB()
nb_classifier_spectral.fit(result, target_spectral_class)
nb_classifier_type = GaussianNB()
```

```
nb_classifier_spectral = GaussianNB()
nb_classifier_spectral.fit(result, target_spectral_class)
nb_classifier_type = GaussianNB()
nb_classifier_type.fit(result, target_type_encoded)

# Calculate accuracy of the models
accuracy_spectral_class = accuracy_score(target_spectral_class, nb_classifier_spectral.predict(result))
accuracy_type = accuracy_score(target_type_encoded, nb_classifier_type.predict(result))

# Print the accuracies in terms of percentile
print("Accuracy of Spectral Class prediction: {:.2f}%".format(accuracy_spectral_class * 100))
print("Accuracy of Type prediction: {:.2f}%".format(accuracy_type * 100))

# Prompt the user for input
temperature = float(input("Enter temperature: "))
l = float(input("Enter L: "))
r = float(input("Enter R: "))
a_m = float(input("Enter A_M: "))
color = input("Enter Color: ")

# Create a DataFrame with the user input
new_data = pd.DataFrame([[temperature, l, r, a_m, color]], columns=['Temperature', 'L', 'R', 'A_M', 'Color'])

# Perform one-hot encoding on the 'Color' column if needed
new_encoded_data = pd.get_dummies(new_data['Color'])

# Concatenate the original DataFrame (new_data) with the encoded columns
new_result = pd.concat([new_data, new_encoded_data], axis=1)
```

```python
# Ensure all the required columns are present in the new_result DataFrame
# Add any missing columns with all zero values
missing_cols = set(result.columns) - set(new_result.columns)
for col in missing_cols:
    new_result[col] = 0

# Reorder the columns to match the order used during training
new_result = new_result[result.columns]

# Make predictions for Spectral_Class and Type separately
predicted_spectral_class = nb_classifier_spectral.predict(new_result)[0]
predicted_type_encoded = nb_classifier_type.predict(new_result)[0]

# Inverse transform the predicted Type back to original labels
predicted_type = label_encoder_type.inverse_transform([predicted_type_encoded])[0]

# Print the predicted Spectral Class and Type
print("Predicted Spectral Class:", predicted_spectral_class)
print("Predicted Type:", predicted_type)
```

```
Accuracy of Spectral Class prediction: 42.08%
Accuracy of Type prediction: 87.08%
Enter temperature: 3068
Enter L: 0.0024
Enter R: 0.17
Enter A_M: 16.12
Enter Color: Red
Predicted Spectral Class: F
Predicted Type: 0
```

Thanks!

BY:
AAKAASH A (22BAI1113)
S JAYANTH (22BAI1177)
ATCHUDHAN S G (22BAI1256)