Astronomical Data Processing:

#CODE

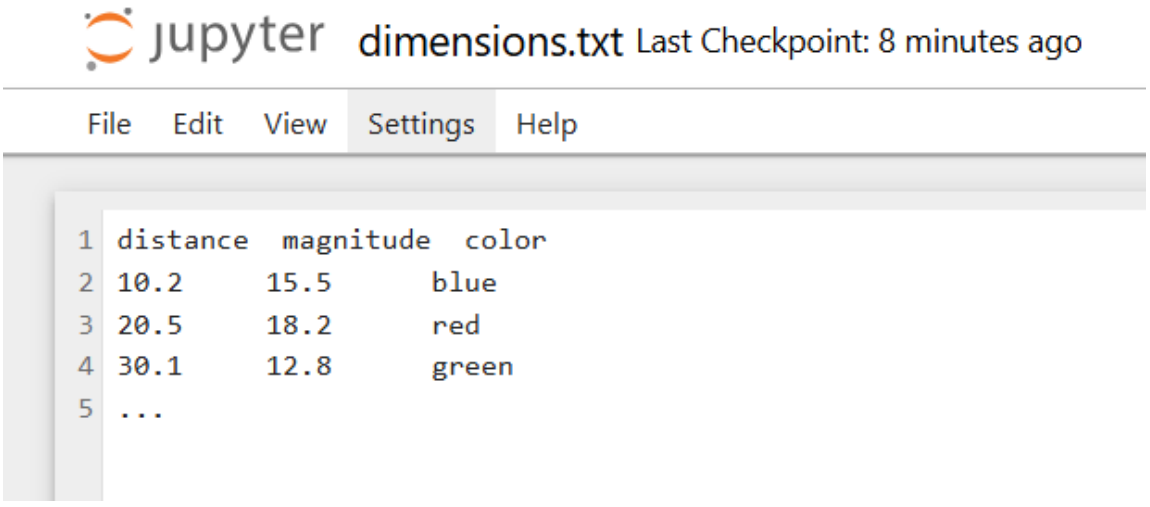
```
import numpy as np

import matplotlib.pyplot as plt

from astropy.io import ascii


def read_astronomical_data(file_path):

  """

  Read astronomical data from the input file into a NumPy array or structured array.


  Args:

    file_path (str): Path to the input file containing astronomical data.


  Returns:

    data (numpy.ndarray or numpy.recarray): NumPy array or structured array representing the astronomical data.

  """

  data = ascii.read(file_path)

  return data


def clean_and_preprocess_data(data):

  """

  Perform data cleaning and preprocessing steps as necessary.


  Args:

    data (numpy.ndarray or numpy.recarray): Astronomical data to be cleaned and preprocessed.


  Returns:

    cleaned_data (numpy.ndarray or numpy.recarray): Cleaned and preprocessed astronomical data.

  """
```

```
    # Handle missing or invalid data values
    data = data[np.isfinite(data['magnitude'])]  # Remove rows with invalid magnitude values

    # Normalize or scale the data if required
    data['magnitude'] /= np.max(data['magnitude'])  # Normalize magnitude values to [0, 1]

    return data

def compute_descriptive_statistics(data):
    """
    Compute descriptive statistics (e.g., mean, median, standard deviation) for relevant attributes.

    Args:
        data (numpy.ndarray or numpy.recarray): Cleaned and preprocessed astronomical data.

    Returns:
        stats (dict): Dictionary containing descriptive statistics for relevant attributes.
    """
    stats = {
        'mean_magnitude': np.mean(data['magnitude']),
        'median_magnitude': np.median(data['magnitude']),
        'std_magnitude': np.std(data['magnitude']),
        'mean_distance': np.mean(data['distance']),
        'median_distance': np.median(data['distance']),
        'std_distance': np.std(data['distance']),
    }
    return stats

def identify_outliers(data):
    """
    Identify outliers or anomalies in the data using statistical methods.
```

```
    Args:
        data (numpy.ndarray or numpy.recarray): Cleaned and preprocessed astronomical data.

    Returns:
        outliers (numpy.ndarray or numpy.recarray): Outliers or anomalies in the data.
    """
    # Use Z-score method to identify outliers
    z_scores = np.abs((data['magnitude'] - np.mean(data['magnitude'])) / np.std(data['magnitude']))
    outliers = data[z_scores > 3]  # Identify outliers with Z-score > 3

    return outliers

def visualize_data(data):
    """
    Utilize Matplotlib to create plots and visualizations of the data.

    Args:
        data (numpy.ndarray or numpy.recarray): Cleaned and preprocessed astronomical data.
    """
    # Scatter plot of magnitude vs. distance
    plt.scatter(data['distance'], data['magnitude'])
    plt.xlabel('Distance (kpc)')
    plt.ylabel('Magnitude')
    plt.title('Magnitude vs. Distance')
    plt.show()

    # Histogram of magnitude values
    plt.hist(data['magnitude'], bins=50)
    plt.xlabel('Magnitude')
    plt.ylabel('Frequency')
```

```
    plt.title('Magnitude Distribution')
    plt.show()

def main():
    file_path = input("Enter the path to the input file: ")
    data = read_astronomical_data(file_path)
    cleaned_data = clean_and_preprocess_data(data)
    stats = compute_descriptive_statistics(cleaned_data)
    outliers = identify_outliers(cleaned_data)
    visualize_data(cleaned_data)

if __name__ == "__main__":
    main()
```

1. Save the script as **astronomical_data_processing.py**.
2. Create a text file containing astronomical data, with each row representing a data point and each column representing a different attribute or feature



3. Run the script using Python: **python astronomical_data_processing.py**.
4. Enter the path to the input file when prompted.

5. The script will process the data, compute descriptive statistics, identify outliers, and visualize the results using Matplotlib.

Optional Enhancements

To implement advanced data analysis techniques, you can modify the script to incorporate clustering, classification, or regression algorithms.

```
from sklearn.cluster import KMeans


def cluster_data(data):
  """
  Perform K-means clustering on the data.

  Args:
    data (numpy.ndarray or numpy.recarray): Cleaned and preprocessed astronomical data.

  Returns:
    clusters (numpy.ndarray): Cluster assignments for each data point.
  """
  kmeans = KMeans(n_clusters=3)
  clusters = kmeans.fit_predict(data[['distance', 'magnitude']])
  return clusters
```

To explore additional datasets or sources ofastronomical data, you can modify the **read_astronomical_data** function to handle different file formats or data sources.

```
import pandas as pd


def read_astronomical_data(file_path):
  """
  Read astronomical data from the input file into a NumPy array or structured array.

  Args:
    file_path (str): Path to the input file containing astronomical data.

  Returns:
```

```
    data (numpy.ndarray or numpy.recarray): NumPy array or structured array representing the astronomical data.
    """
    if file_path.endswith('.csv'):
        data = pd.read_csv(file_path)
    elif file_path.endswith('.txt'):
        data = ascii.read(file_path)
    else:
        raise ValueError("Unsupported file format")

    return data.to_records(index=False)
```

This modification allows the script to handle both CSV and text files containing astronomical data.