Discussion of Initial Findings

Task 1: Understanding the Dataset - The GAIA Dataset

Introduction to the Dataset

This project utilizes the Gaia dataset, a vast astrometric data release from the European Space Agency's (ESA) Gaia mission. The Gaia spacecraft is charting a three-dimensional map of our galaxy, the Milky Way, in the process revealing the composition, formation, and evolution of the Galaxy. The dataset contains high-precision measurements for over a billion stars, including their positions, parallaxes (distances), proper motions (movements across the sky), brightness, and temperature.

For this analysis, we will be using a subset of the Gaia data, focusing on the columns relevant to our research questions.

Data Schema and Key Columns

The following columns from the Gaia dataset will be used in our analysis:

- `source_id`: Unique identifier for each star.
- `ra`: Right Ascension (celestial longitude).
- `dec`: Declination (celestial latitude).
- `parallax`: Parallax in milliarcseconds, used to calculate distance ($d = 1/p$).
- `parallax_error`: The uncertainty in the parallax measurement.
- `pmra`: Proper motion in the direction of Right Ascension.
- `pmdec`: Proper motion in the direction of Declination.
- `phot_g_mean_mag`: Mean apparent magnitude in the G-band (a measure of brightness as seen from Earth).
- `bp_rp`: The blue-red color index, a proxy for the star's surface temperature.
- `teff_gspphot`: Effective temperature of the star's photosphere, derived from photometry.

Initial Data Loading and Exploration

As per the coursework requirements, our first step is to load and understand the dataset. Since a specific subset of the Gaia data was not provided, we will programmatically download a representative sample using the `astroquery` library. This ensures our analysis is reproducible.

The following Python script performs these actions:

1. It defines a query in Astronomical Data Query Language (ADQL) to fetch the top 50,000 stars from the Gaia DR3 catalog that have complete data for our columns of interest.
2. It uses `astroquery` to execute this query against the official Gaia archive.
3. To avoid re-downloading the data on every run, it saves the resulting dataset as a `gaia_subset.csv` file in the local `data/` directory.
4. Finally, it loads this local CSV file into a PySpark DataFrame for initial inspection.

```
import os
from pyspark.sql import SparkSession
from astroquery.gaia import Gaia

# --- Part 1: Download data using Astroquery ---

# Define the ADQL query to select a subset of data from Gaia DR3.
# We select 50,000 stars with non-null values for key columns.
adql_query = """
SELECT TOP 5000
  source_id, ra, dec, parallax, parallax_error, pmra, pmdec,
  phot_g_mean_mag, bp_rp, teff_gspphot
FROM
  gaiadr3.gaia_source
WHERE
  parallax IS NOT NULL AND
  parallax_error IS NOT NULL AND
  pmra IS NOT NULL AND
  pmdec IS NOT NULL AND
  phot_g_mean_mag IS NOT NULL AND
  bp_rp IS NOT NULL AND
  teff_gspphot IS NOT NULL
"""

# Define the output path for the cached data
output_dir = "data"
output_file = os.path.join(output_dir, "gaia_subset.csv")
```

```
# Create the data directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Check if the file already exists to avoid re-downloading
if not os.path.exists(output_file):
    print("Downloading Gaia data subset via astroquery...")
    # Launch the synchronous query
    job = Gaia.launch_job(adql_query)
    # Get the results as an astropy Table
    results_table = job.get_results()
    # Convert to a pandas DataFrame
    results_df = results_table.to_pandas()
    # Save the DataFrame to a CSV file
    results_df.to_csv(output_file, index=False)
    print(f"Data saved to {output_file}")
else:
    print(f"Data already exists at {output_file}, skipping download.")


# --- Part 2: Load data into Spark ---

# Initialize Spark Session
spark = SparkSession.builder \
    .appName("Gaia Dataset Exploration") \
    .getOrCreate()

# Load the data from our local CSV file
gaia_df = spark.read.csv(output_file, header=True, inferSchema=True)

# Print the schema to understand the data types
print("\nDataFrame Schema:")
gaia_df.printSchema()

# Show the first 5 records from the dataset
print("\nFirst 5 records:")
gaia_df.show(5)
```

The script's output confirms the successful download and caching of our data subset. The `printSchema()` output verifies that Spark has correctly inferred the data types (e.g., `Double` for numeric columns, `Long` for IDs), which is essential for accurate calculations in subsequent SQL queries. The `show(5)` command displays a sample of the records, confirming that the

data is loaded correctly and matches the structure we requested from the Gaia archive.

This reproducible data acquisition and initial exploration step fulfills the requirements of Task 1 and prepares us for the in-depth analysis in Task 2.