Task 5: Core Analysis and Visualization

Objective: To calculate the Earth Similarity Index (ESI) for each exoplanet and to create an interactive scatter plot to visualize the results using Chart.js.
Estimated Time: 4-6 hours

Step 1: Implement the Earth Similarity Index (ESI) Formula

File to Modify: app/analysis/core.py
Action:
The ESI is a measure of how physically similar a planetary-mass object is to Earth. It's calculated based on properties like radius, density, escape velocity, and surface temperature.

1. Create a new function calculate_esi(planet_data). This function will take a single planet's data (as a Pandas Series or a dictionary) and return its ESI score.
2. The general formula for the ESI is a weighted geometric mean. For a property x, the similarity S_x is calculated as: S_x = (1 - |(x - x_earth) / (x + x_earth)|) ^ (weight / n)
3. The final ESI is the geometric mean of the individual similarities: ESI = (S_radius * S_density * S_escape_velocity * S_surface_temp) ^ (1/4)
4. Implement this logic in your function. You will need to define the Earth's reference values and the weights for each property inside your function. You can find these values in the documentation below.
5. Modify your existing load_exoplanet_data function. After loading and cleaning the data, use the .apply() method on the DataFrame to run your calculate_esi function for each planet (each row), creating a new 'esi' column in the DataFrame.

Why: The .apply() method is a highly efficient way to perform a complex operation on every row of a DataFrame. Keeping the ESI calculation in its own function makes your code clean, testable, and easy to understand.
Documentation:

- Planetary Habitability Laboratory: Earth Similarity Index (This page contains the formula, reference values for Earth, and weights you'll need.)
- Pandas .apply()

Step 2: Prepare Data for Visualization

File to Modify: app/main/routes.py
Action:
Your dashboard route now needs to prepare the data not just for the HTML table, but also for the JavaScript chart.

1. In your /dashboard route, after you've loaded the data and calculated the ESI, create a separate data structure specifically for the chart.
2. Chart.js scatter plots expect data as a list of objects, where each object has x and y properties. Create a list of these objects. For example, you could plot Orbital Period (pl_orbper) on the x-axis and Planet Mass (pl_masse) on the y-axis.
3. You'll also want to pass the ESI scores to the chart to color-code the points. Create a

separate list containing just the ESI values for each planet.
4. Pass both the main table data (from Task 4) and this new chart-specific data to your render_template function. You'll need to convert the chart data to a JSON string using json.dumps() so it can be safely embedded in the HTML template.

Why: Preparing the data on the backend (in Python) is much more efficient than trying to manipulate complex data structures in frontend JavaScript. Sending clean, ready-to-use data to the template simplifies your JavaScript code significantly.
Documentation:

- Python's json module

Step 3: Add a Chart Canvas to the Dashboard

File to Modify: app/templates/main/dashboard.html
Action:

1. Find a suitable place on your dashboard page (e.g., above the data table) to add the chart.
2. Add an HTML <canvas> element. Give it a unique id, for example, <canvas id="exoplanetChart"></canvas>. This canvas is where Chart.js will draw your visualization.

Why: The <canvas> element is a standard HTML5 feature that provides a drawing surface for JavaScript. Libraries like Chart.js use this surface to render complex graphics.

Step 4: Install and Configure Chart.js

Action:

1. **Install Chart.js:** If you haven't already, add the Chart.js library to your project. The easiest way is to link to its CDN in your base.html template, just like you did for Simple-DataTables.
2. **Initialize the Chart:** In a <script> tag at the bottom of your dashboard.html template, you will now write the JavaScript to create the chart.
    - Get the chart data you passed from Flask. You'll need to use the |tojson filter in Jinja2 to safely render the JSON string into a JavaScript variable.
    - Get the canvas element using document.getElementById('exoplanetChart').
    - Create a new Chart() instance. You will need to configure it:
        - Set the type to 'scatter'.
        - Pass your prepared x and y data to the datasets configuration.
        - Use the ESI data to dynamically set the color of each point in the scatter plot. You can write a small JavaScript function that maps an ESI score (0 to 1) to a color gradient (e.g., blue to green).
        - Configure the chart's options, such as titles for the chart and the axes.

Why: This JavaScript code is the final step that connects your backend data to the frontend visualization. Using the |tojson filter is a secure way to pass data from your server-side Jinja2 template to your client-side JavaScript.
Documentation:

- Chart.js Official Website

- Chart.js Scatter Plots
- Jinja2 tojson filter

Step 5: Display the ESI Score in the Table

File to Modify: app/templates/main/dashboard.html
Action:
This is a simple but important final touch.

1. Add a new column header (<th>) to your data table for "ESI".
2. In your for loop that creates the table rows, add a new table data cell (<td>) that displays the planet's ESI score. You'll likely want to format it to show only 2 or 3 decimal places. You can do this using the .round() method in Pandas before sending it to the template, or using the round filter in Jinja2.

Why: This makes the raw ESI value easily visible to the user, complementing the visual representation in the chart and providing precise data for analysis.
You're now on the verge of having a true data analysis application. This task combines backend data processing with frontend visualization—a core skill set for any data scientist or engineer.