1 Database Connection

The connection to the MySQL database is established using the following parameters:

- **Host:** `195.201.104.116`
- **User:** `eta14802_astrobeam`
- **Database:** `eta14802_astrobeam`

The connection is made via:

```
db = mysql.connector.connect(
    host="195.201.104.116",
    user="eta14802_astrobeam",
    password="*****",  % Omit for security
    database="eta14802_astrobeam"
)
```

2 Tables and Schemas

2.1 gbt_data

Column Name	Data Type
id	int(11)
vHI	double
fHI	double
fBHI	double
src_file	varchar(100)
telescope	varchar(100)
beam_size	double
object	varchar(100)
ra_dec	varchar(100)
rest_frequency	double
central_velocity	double
integration_time	double
observation_date	varchar(100)
details	varchar(100)

2.2 gbt_observations

Column Name	Data Type
src_file	varchar(100)
telescope	varchar(100)
beam_size	double
object	varchar(100)
ra_dec	varchar(100)

rest_frequency	double
central_velocity	double
integration_time	double
observation_date	varchar(100)
details	varchar(100)

2.3 gbt_values

Column Name	Data Type
id	int(11)
vHI	double
fHI	double
fBHI	double
object	varchar(100)

3 Function: `db_info()`

Purpose

The `db_info()` function connects to the `astrobeam` MySQL database and provides a complete overview of the schema.

Output

- Names of all tables in the database
- Column names and types for each table
- List of database views
- Foreign key relationships between tables
- Primary and foreign keys per table
- Any `UNIQUE` or `CHECK` constraints applied to columns

Connection Details

The connection is established using `mysql.connector` with the following parameters:

```
host="195.201.104.116"
user="eta14802_astrobeam"
database="eta14802_astrobeam"
```

4 Function: `open_con()`

Purpose

The `open_con()` function establishes a connection to the `astrobeam` MySQL database and initializes a cursor for executing queries.

Behavior

- Connects to the database using credentials and host parameters.
- A cursor is created and can be used to execute SQL commands.

Connection Details

- **Host:** `195.201.104.116`
- **User:** `eta14802_astrobeam`
- **Database:** `eta14802_astrobeam`

5 Function: `close_con()`

Purpose

The `close_con()` function safely terminates the MySQL database connection and closes the cursor.

Behavior

- Closes the database cursor.
- Closes the database connection.

6 Function: `tables_info()`

Purpose

The `tables_info()` function connects to the `astrobeam` MySQL database and retrieves the structure of each table, including column names and data types.

Behavior

- Establishes a connection to the database using MySQL Connector.
- Iterates through each table name from a predefined list of tables.
- Executes a `DESCRIBE` query for each table.
- Prints the column names and data types for each table to the console.

Output

- A formatted list printed to the console showing:
  - Table name
  - Column names
  - Data types for each column

7 Function: `databy_condition()`

Purpose

The `databy_condition()` function gets records from a specified table based on multiple dynamic filtering conditions. It returns the result as a Pandas DataFrame.

Inputs

- `table` – The name of the table to query (string).
- `conditions` – A dictionary that defines how to filter the data.
- `logical_operator` – Optional. The word used between multiple conditions: `AND` (default) or `OR`.

Behavior

- Constructs a dynamic SQL `WHERE` clause based on user-defined conditions.
- Supports various operators: `=, >, <, BETWEEN, IN, LIKE`, and compound conditions.
- Executes a `SELECT *` query against the given table.
- Converts the result into a Pandas DataFrame for analysis.

Output

- A `pandas.DataFrame` containing all rows that match the given conditions.

8 Function: `df()`

Purpose

The `df()` function retrieves all records from a specified database table and returns the results as a Pandas DataFrame.

Inputs

- `table` – Name of the table to fetch data from (string).

Behavior

- Connects to the `astrobeam` MySQL database.
- Executes a `SELECT *` query on the specified table.
- Fetches all rows from the table and turns them into a Pandas DataFrame with the same column names as in the database.

Output

- A `pandas.DataFrame` containing all rows and columns from the selected table.

9 Function: `databy_sort_group_having()`

Purpose

The `databy_sort_group_having()` function applies optional SQL-style operations such as `GROUP BY`, aggregation, `HAVING`-like filters, and `ORDER BY` to a Pandas DataFrame.

Inputs

- `df` – A Pandas DataFrame to process.
- `group_by` – Column name or list of columns to group by (optional).
- `aggregates` – A dictionary where keys are column names and values are aggregation functions (`SUM`, `AVG`, `COUNT`, `MIN`, `MAX`).
- `having` – A dictionary specifying conditions to filter the aggregated result. Format: {`column: (operator, value)`}.
- `order_by` – Column name to sort by (optional).
- `order_type` – `ASC` (default) or `DESC` for sort direction.

Behavior

- If `group_by` and `aggregates` are provided, the function groups the DataFrame by the specified column(s), then applies the selected aggregation functions (like `SUM`, `AVG`, etc.) to the grouped data.
- If a `having` dictionary is provided, it filters the grouped/aggregated results based on the given conditions — similar to SQL's `HAVING` clause.
- If `order_by` is specified, the final DataFrame is sorted by that column. The sort direction depends on the value of `order_type` (`ASC` or `DESC`).

Output

- A transformed `pandas.DataFrame` with applied grouping, filtering, and sorting.

10 Function: `line_plot()`

Purpose

The `line_plot()` function retrieves two columns of numerical data from a database table and visualizes them using a line plot. It also returns the data as a Pandas DataFrame for further analysis.

Inputs

- `table` – Name of the table to query (string).
- `x_column` – Column to use as the x-axis (string).
- `y_column` – Column to use as the y-axis (string).

Behavior

- Connects to the `astrobeam` MySQL database.
- Executes a `SELECT` query for the specified columns.
- Converts the result into a Pandas DataFrame.
- Plots the data using `matplotlib.pyplot.plot()` with:
  - X-axis labeled as "Frequency (MHz)"
  - Y-axis labeled as "Power"
  - Title: "Radio Spectrum"
- Displays the plot.

Output

- A `pandas.DataFrame` containing the queried data.
- A displayed line plot of the selected columns.

11 Function: `heatmap_plot()`

Purpose

The `heatmap_plot()` function retrieves time-series spectral data from the database and generates a heatmap showing power variations across frequency and time. The function returns the corresponding DataFrame.

Inputs

- `table` – Name of the table to query (string).
- `freq_column` – Column representing frequency values (string).
- `time_column` – Column representing time or observation sequence (string).
- `power_column` – Column representing power or intensity (string).

Behavior

- Connects to the `astrobeam` MySQL database.
- Executes a `SELECT` query for the specified columns.
- Constructs a DataFrame and reshapes it into a pivot table using:
  - Index: `time_column`
  - Columns: `freq_column`
  - Values: `power_column` (averaged if duplicates exist)
- Uses `seaborn.heatmap()` to visualize the data.
- If 1420 MHz is present in the frequency data, a vertical line is drawn to mark the hydrogen line.

Output

- A `pandas.DataFrame` containing the raw frequency, time, and power data.
- A heatmap plot showing power distribution over frequency and time.

12 Function: `plot_moving_average()`

Purpose

The `plot_moving_average()` function retrieves frequency and power data from a table, applies a moving average filter to smooth the signal, and plots both raw and smoothed data.

Inputs

- `table` – Table name to query from (string).
- `freq_column` – Name of the frequency column (string).
- `power_column` – Name of the power/intensity column (string).
- `window` – Optional. Size of the moving average window. Default is 5.

Behavior

- Retrieves and sorts frequency-power data.
- Applies a centered moving average using NumPy's convolution.
- Plots both raw and smoothed curves.
- Highlights 1420 MHz (hydrogen line) if present.

Output

- A `pandas.DataFrame` of the raw query results.
- A line plot showing both raw and smoothed spectra.

13 Function: `heatmap_moving_average()`

Purpose

The `heatmap_moving_average()` function visualizes smoothed radio spectrum data over time using a heatmap after applying a moving average to the power column.

Inputs

- `table` – Table name to query from (string).
- `freq_column` – Name of the frequency column (string).
- `time_column` – Name of the time column (string).
- `power_column` – Name of the power column (string).
- `window` – Optional. Size of the moving average window. Default is 5.

Behavior

- Retrieves frequency-time-power data from the table.
- Applies a moving average to smooth the power signal.
- Uses a pivot table to reshape data for heatmap plotting.
- Draws a heatmap with Seaborn and optionally highlights the 1420 MHz hydrogen line.

Output

- A `pandas.DataFrame` containing the smoothed data.
- A heatmap showing power distribution across time and frequency.

14 Function: `detect_spectral_lines()`

Purpose

The `detect_spectral_lines()` function identifies peaks (emission lines) and absorption lines in the power spectrum and visualizes them alongside the raw data.

Inputs

- `table` – Table name to query from (string).
- `freq_column` – Frequency column name (string).
- `power_column` – Power column name (string).
- `prominence` – Optional. Peak prominence threshold for detection. Default is 0.1.

Behavior

- Retrieves spectral data from the database.
- Uses SciPy's `find_peaks()` to detect spectral lines.
- Plots the spectrum with emission and absorption features marked.
- Highlights the hydrogen line (1420 MHz) if present.

Output

- A dictionary with:
  - `"emission"` – Frequencies of detected peaks.
  - `"absorption"` – Frequencies of detected troughs.
- The original `pandas.DataFrame` used for analysis.
- A plot showing detected features.

15 Function: `search_by_rest_frequency()`

Purpose

The `search by rest frequency()` function retrieves rows from a specified table where the `rest_frequency` value falls within an optional range.

Inputs

- `table` – Name of the table to query (string).
- `min_freq` – Optional. Lower frequency bound (float).
- `max_freq` – Optional. Upper frequency bound (float).

Behavior

- Connects to the AstroBeam MySQL database.
- Constructs a dynamic WHERE clause:
  - If `min_freq` is provided, applies `rest_frequency >= min_freq`.
  - If `max_freq` is provided, applies `rest_frequency <= max_freq`.

Output

A `pandas.DataFrame` of rows within the selected frequency range.

16 Function: `search_by_observation_date()`

Purpose
The `search_by_observation_date()` function fetches rows from a table where `observation_date` falls within an optional date range.

Inputs

- `table` – Name of the table to query (string).
- `start_date` – Optional. Start date in `YYYY-MM-DD` format (string).
- `end_date` – Optional. End date in `YYYY-MM-DD` format (string).

Behavior

- Connects to the database and constructs a dynamic WHERE clause:
  - If `start_date` is provided, applies `observation_date >= start_date`.
  - If `end_date` is provided, applies `observation_date <= end_date`.

Output
A `pandas.DataFrame` of date-filtered observations.

17 Function: `search_by_integration_time()`

Purpose
The `search_by_integration_time()` function filters observations based on how long the integration lasted.

Inputs

- `table` – Name of the table to query (string).
- `min_time` – Optional. Minimum integration time (float).
- `max_time` – Optional. Maximum integration time (float).

Behavior

- Dynamically builds a WHERE clause to filter `integration_time` using the provided min/max values.

Output
A `pandas.DataFrame` with observations matching the time range.

18 Function: `search_by_ra_dec()`

Purpose
The `search_by_ra_dec()` function matches rows with a specific coordinate value, constructed from separate RA and DEC inputs.

Inputs

- `table` – Name of the table to query (string).
- `ra` – Right Ascension component (string), e.g., "08 53 29.1".
- `dec` – Declination component (string), e.g., "+57 35 54".

Behavior

- Combines `ra` and `dec` into a full coordinate string.
- Filters rows where the `ra_dec` column exactly matches the combined coordinate.
- Converts the result into a pandas DataFrame.

Output
A `pandas.DataFrame` of matching sky coordinates.

19 Function: `search_by_telescope_name()`

Purpose
The `search_by_telescope_name()` function searches for rows where the telescope name contains a specified keyword.

Inputs

- `table` – Table name to query (string).
- `keyword` – Substring to match in `telescope_name` (string).

Behavior

- Constructs a SQL `LIKE` query using wildcards (`%keyword%`) to perform partial matching.

Output
A `pandas.DataFrame` with all telescope names that contain the specified keyword.