# PySpark Script Functionality Explanation

This document provides an explanation of the key components and functionality of the provided PySpark script. The script uses SQL queries generated from a YAML file to load data into a BigQuery table, with options for loading data directly from a file or applying transformations before loading. It includes error handling, logging, and session management.

## 1. create\_spark\_session()

This function initializes and returns a SparkSession, which is necessary to run PySpark operations. The function uses an environment variable (SPARK\_APP\_NAME) to set the application name dynamically, defaulting to 'Load SQL to BigQuery' if not specified. This session serves as the entry point for all subsequent PySpark operations, such as reading data and executing SQL queries.

## 2. load\_sql\_to\_bigquery(sql\_query, staging\_table\_name, load\_type, logger)

This function loads the results of a SQL query into a specified BigQuery table, with the following steps:

- \*\*Initialize Spark Session\*\*: Uses `create\_spark\_session()` to obtain a Spark session.

- \*\*Execute SQL Query\*\*: Reads the SQL query results into a DataFrame by interacting with BigQuery. If an error occurs, it logs the error and exits.

- \*\*Cache the DataFrame\*\*: Caches the DataFrame to improve performance on subsequent operations.

- \*\*Add Timestamp Column\*\*: Adds a `load\_ts` column containing the current timestamp to record the load time.

- \*\*Check Table Existence\*\*: Uses the BigQuery API to verify if the destination table exists. If it does not, the function creates the table with a schema matching the DataFrame.

- \*\*Truncate Table if Required\*\*: If `load\_type` is set to 'truncate', deletes existing records from the table.

- \*\*Append Data to BigQuery\*\*: Writes the DataFrame to BigQuery in 'append' mode to add new records.

- \*\*Unpersist DataFrame\*\*: Releases DataFrame memory to optimize resources.

## 3. main(yaml\_file\_path, output\_dir, loginput\_path)

The main function serves as the primary workflow for the script, handling the loading of data from a YAML file into BigQuery based on configurations in the YAML file. It includes:

- \*\*Logging Setup\*\*: Initializes logging to track script execution details.

- \*\*Generate SQL from YAML\*\*: Calls `generate\_sql\_from\_yaml\_file()` to convert YAML contents into a SQL query.

- \*\*Load SQL to File\*\*: If specified, writes the SQL query to an output file in the specified directory.

- \*\*Load SQL to BigQuery\*\*: If specified, loads the SQL query results to BigQuery by calling `load\_sql\_to\_bigquery`.

## 4. Argument Parsing and Execution

The script expects three arguments:

- \*\*yaml\_file\_path\*\*: Path to the input YAML file containing the SQL configuration.

- \*\*output\_dir\*\*: Directory where the SQL file will be saved, if `load\_file` is enabled.

- \*\*log\_file\_path\*\*: Directory path where log files will be stored.

The script processes these arguments using the `argparse` library, allowing users to pass values at runtime. After parsing, it calls the `main` function with these arguments to execute the full workflow.