In [ ]:
```python
import pandas as pd


df = pd.read_excel('Time Series.xlsx')

# Rename columns for consistency
df.columns = ['bot_id', 'start_time', 'end_time', 'activity']

# Convert 'start_time' and 'end_time' columns to datetime objects
df['start_time'] = pd.to_datetime(df['start_time'])
df['end_time'] = pd.to_datetime(df['end_time'])

# Sort the dataframe by 'bot_id' and 'start_time'
df.sort_values(by=['bot_id', 'start_time'], inplace=True)

# Initialize a list to hold the results
continuous_periods = []

# Iterate through each bot's data
for bot_id, bot_data in df.groupby('bot_id'):
    bot_data = bot_data.reset_index(drop=True)
    current_period = {
        'bot_id': bot_id,
        'start_time': bot_data.loc[0, 'start_time'],
        'end_time': bot_data.loc[0, 'end_time'],
        'activities': [bot_data.loc[0, 'activity']]
    }

    for i in range(1, len(bot_data)):
        row = bot_data.loc[i]
        if row['start_time'] <= current_period['end_time']:
            # Extend the current period
            current_period['end_time'] = max(current_period['end_time'], row['end_time'])
            current_period['activities'].append(row['activity'])
        else:
            # Save the current period and start a new one
            continuous_periods.append(current_period)
            current_period = {
                'bot_id': bot_id,
```

```
                'start_time': row['start_time'],
                'end_time': row['end_time'],
                'activities': [row['activity']]
            }

    # Add the last continuous period for this bot
    continuous_periods.append(current_period)

# Convert the list of continuous periods to a DataFrame
result_df = pd.DataFrame(continuous_periods)

# Display the result
result_df
```

Out[ ]:

	bot_id	start_time	end_time	activities
0	Deepti	2023-03-29 15:31:52.620	2023-10-15 15:31:52.620	[Business Development, Reply to Customers, Sen...
1	Jyoti	2023-03-29 15:31:52.620	2023-10-15 13:07:52.620	[Business Development, Inspection, Fund raisin...
2	Priyanka	2023-03-29 15:31:52.620	2023-10-15 15:31:52.620	[Business Development, Remote Inspection, Podc...
3	Ravi	2023-03-29 15:31:52.620	2023-10-15 08:19:52.620	[Call, Call, Fund raising, Call, Fund raising,...
4	Ravi	2023-10-15 10:43:52.620	2023-06-10 17:55:52.620	[Send Email]
5	Ravi	2023-10-15 10:43:52.620	2023-06-28 13:07:52.620	[Reporting]
6	Ravi	2023-10-15 10:43:52.620	2023-07-26 01:07:52.620	[Inspection]
7	Ravi	2023-10-15 10:43:52.620	2023-07-06 13:07:52.620	[Inspection]
8	Ravi	2023-10-15 13:07:52.620	2023-04-18 10:43:52.620	[Send Email]
9	Ravi	2023-10-15 13:07:52.620	2023-08-27 08:19:52.620	[Podcast]
10	Sharan	2023-03-29 15:31:52.620	2023-10-15 05:55:52.620	[Inspection, Send Email, Fund raising, Updates...
11	Sharan	2023-10-15 08:19:52.620	2023-07-20 08:19:52.620	[Fund raising]
12	Sharan	2023-10-15 08:19:52.620	2023-08-14 17:55:52.620	[Reporting]
13	Sharan	2023-10-15 08:19:52.620	2023-04-08 05:55:52.620	[Podcast]
14	Sharan	2023-10-15 15:31:52.620	2023-07-23 17:55:52.620	[Fund raising]

This Python script processes a dataset stored in an Excel file (`Time Series.xlsx`) to identify continuous periods of activities for each bot

Step-by-Step Explanation:

1. **Read the Excel File:**
   - `df = pd.read_excel('Time Series.xlsx')` : Reads the Excel file into a Pandas DataFrame `df` .
2. **Column Renaming:**

- `df.columns = ['bot_id', 'start_time', 'end_time', 'activity']` : Renames the columns for consistency and clarity. Assumes the original column names might have been different.

3. **Convert Time Columns to Datetime:**

- `df['start_time'] = pd.to_datetime(df['start_time'])`
- `df['end_time'] = pd.to_datetime(df['end_time'])` : Converts the `start_time` and `end_time` columns to Pandas datetime objects, ensuring they are in a format that allows for easy comparison and manipulation.

4. **Sort the DataFrame:**

- `df.sort_values(by=['bot_id', 'start_time'], inplace=True)` : Sorts the DataFrame primarily by `bot_id` and then by `start_time` . This sorting is crucial for correctly identifying continuous periods of activities for each bot.

5. **Identify Continuous Periods:**

- **Iterate through each bot's data (`bot_id` grouping):**
  - `for bot_id, bot_data in df.groupby('bot_id'):` : Groups the DataFrame `df` by `bot_id` , allowing iteration through each bot's data separately.
  - **Initialize a list (`continuous_periods`) to hold the results of continuous periods for each bot.**
  - **Loop through each row (`bot_data`):**
    - `current_period` **initialization:** Starts a new period with the first row of `bot_data` .
    - **Comparison and grouping:** Checks each subsequent row to see if it falls within the current period or starts a new period if it doesn't.
    - **Append continuous periods:** Appends each identified continuous period (when a new period begins) to the `continuous_periods` list.

6. **Convert Results to DataFrame:**

- `result_df = pd.DataFrame(continuous_periods)` : Converts the list of continuous periods (`continuous_periods`) into a new Pandas DataFrame `result_df` .

7. **Display the Result:**

- **`result_df`**: Outputs the DataFrame `result_df`, which now contains columns for `bot_id`, `start_time`, `end_time`, and `activities` (list of activities during each continuous period).

Output:

The `result_df` DataFrame will display each bot's continuous periods of activities, where each row represents a unique continuous period with its start time, end time, and a list of activities performed during that period.

In [ ]:
```
# Save the result to an Excel file
result_df.to_excel('Continuous_Periods.xlsx', index=False)
```

In [ ]:
```
import pandas as pd
import sqlite3

# Load the data from the Excel file
file_path = 'Time Series.xlsx'
df = pd.read_excel(file_path)

# Rename columns to match the SQL script
df.columns = ['bot_id', 'start_time', 'end_time', 'activity']

# Convert date columns to datetime format
df['start_time'] = pd.to_datetime(df['start_time'])
df['end_time'] = pd.to_datetime(df['end_time'])

# Create a SQLite database and load the data into a table
conn = sqlite3.connect(':memory:')
df.to_sql('bot_activities', conn, index=False, if_exists='replace')

# Define the SQL query
query = '''
WITH ordered_activities AS (
    SELECT
        bot_id,
        start_time,
        end_time,
        activity,
        LAG(end_time) OVER (PARTITION BY bot_id ORDER BY start_time) AS prev_end_time
```

```
    FROM bot_activities
),
merged_periods AS (
    SELECT
        bot_id,
        start_time,
        end_time,
        activity,
        SUM(CASE WHEN start_time <= prev_end_time THEN 0 ELSE 1 END) OVER (PARTITION BY bot_id ORDER BY start_time) AS period_
    FROM ordered_activities
)
SELECT
    bot_id,
    MIN(start_time) AS start_time,
    MAX(end_time) AS end_time,
    GROUP_CONCAT(activity) AS activities
FROM merged_periods
GROUP BY bot_id, period_id;
'''

# Execute the query and fetch the results
result = pd.read_sql_query(query, conn)

# Close the connection
conn.close()

# Display the result
result
```

Out[ ]:

	bot_id	start_time	end_time	activities
0	Deepti	2023-03-29 15:31:52.620000	2023-09-14 10:43:52.620000	Business Development,Reply to Customers,Send E...
1	Deepti	2023-03-30 17:55:52.620000	2023-10-09 13:07:52.620000	Podcast,Podcast,Remote Inspection,Reply to Cus...
2	Deepti	2023-04-02 17:55:52.620000	2023-05-06 10:43:52.620000	Updates
3	Deepti	2023-04-02 22:43:52.620000	2023-08-25 05:55:52.620000	Remote Inspection,Reply to Customers,Business ...
4	Deepti	2023-04-03 13:07:52.620000	2023-08-30 15:31:52.620000	Call,Business Development,Inspection,Call,Podc...
...	...	...	...	...
3946	Sharan	2023-10-14 17:55:52.620000	2023-09-03 15:31:52.620000	Reporting,Podcast
3947	Sharan	2023-10-14 22:43:52.620000	2023-09-10 22:43:52.620000	Updates,Updates
3948	Sharan	2023-10-15 03:31:52.620000	2023-08-11 03:31:52.620000	Call,Send Email,Reply to Customers
3949	Sharan	2023-10-15 08:19:52.620000	2023-08-14 17:55:52.620000	Fund raising,Reporting,Podcast
3950	Sharan	2023-10-15 15:31:52.620000	2023-07-23 17:55:52.620000	Fund raising

3951 rows × 4 columns

This Python script performs several operations using Pandas and SQLite to analyze and transform time-series data representing bot activities.

Step-by-Step Explanation:

1. **Load Data from Excel:**
   - `df = pd.read_excel(file_path)`: Reads data from an Excel file (`Time Series.xlsx`) into a Pandas DataFrame `df`.
2. **Column Renaming:**
   - `df.columns = ['bot_id', 'start_time', 'end_time', 'activity']`: Renames columns for clarity and consistency, assuming they might have had different names in the original Excel file.
3. **Convert Date Columns to Datetime:**

- `df['start_time'] = pd.to_datetime(df['start_time'])`
- `df['end_time'] = pd.to_datetime(df['end_time'])` : Converts the `start_time` and `end_time` columns to Pandas datetime objects. This conversion ensures that these columns are in a format suitable for chronological comparisons and operations.

4. **Create SQLite Database and Table:**

- `conn = sqlite3.connect(':memory:')` : Establishes an in-memory SQLite database connection.
- `df.to_sql('bot_activities', conn, index=False, if_exists='replace')` : Loads the Pandas DataFrame `df` into an SQLite table named `bot_activities`. The `if_exists='replace'` parameter ensures that if the table already exists, it will be replaced with the new data.

5. **Define SQL Query for Period Identification:**

- `query = ''' ... '''` : Defines a complex SQL query using Common Table Expressions (CTEs) to identify continuous periods of bot activities. Here's a breakdown of the query:
  - `ordered_activities` **:** Uses the `LAG` function to fetch the end time of the previous activity, partitioned by `bot_id` and ordered by `start_time` **.**
  - `merged_periods` **:** Uses a window function and conditional aggregation (`SUM` with `CASE WHEN`) to assign a period ID to each row based on whether the start time is after the previous activity's end time.
  - `SELECT` **statement:** Aggregates the results to find the minimum `start_time` , maximum `end_time` , and concatenates all activities (`GROUP_CONCAT`) within each identified period for each `bot_id` .

6. **Execute SQL Query and Fetch Results:**

- `result = pd.read_sql_query(query, conn)` : Executes the SQL query defined in `query` using the SQLite connection `conn` and retrieves the results into a Pandas DataFrame `result` .

7. **Close the SQLite Connection:**

- `conn.close()` : Closes the SQLite connection to free up resources.

8. **Display the Result:**

- `result` : Outputs the `result` DataFrame, which contains the aggregated periods of bot activities with their start time, end time, and concatenated activities.

Output:

The `result` DataFrame will display each bot's continuous periods of activities, where each row represents a unique period with its start time, end time, and a concatenated list of activities performed during that period.

Purpose:

This script is useful for analyzing time-series data from bot activities, identifying and summarizing continuous periods of activity for each bot. The combination of Pandas for data manipulation and SQLite for efficient querying and aggregation provides a powerful workflow for data analysis tasks involving time-series data. Adjustments can be made to the SQL query or data processing steps based on specific requirements or additional data exploration needs.