Senior Data Engineer – Python assessment

Task

Develop a Python application that visualises the Share of search data for 3 terms: Football, Rugby and Tennis over the past 3 months.

To accomplish this please use Google Trends data as the data source. Additionally, we would like you to present this data using a suitable graph to effectively present this data. In case access to the Google Trends API is not feasible, explore alternative datasets consumable with Python for achieving the same objective.

Pyplot

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. The plot() function is used to draw points (markers) in a diagram.

Solution Options:

To accomplish the solution using Google Trends data, I have configured trends and downloaded the trends from the Google Trends. Please refer the Google Trends URL:

https://trends.google.com/trends/explore?date=today%203-m&q=Football,Rugby,%2Fm%2F07bs0&hl=en-US

We have 2 Options to implement the Visualization.

1. Downloaded the trends from the Google as CSV file and use the CSV file as data source.

   This solution option loads search data from a CSV file into a DataFrame, converts the date column to datetime format, and then plots the share of search for Football, Rugby, and Tennis over the past 3 months using Matplotlib.

Importing Required Libraries:

- **import pandas as pd**: Imports the pandas library, which is used for data manipulation and analysis.
- **import matplotlib.pyplot as plt**: Imports the pyplot module from the matplotlib library, which is used for creating plots.

Loading CSV File into a DataFrame:

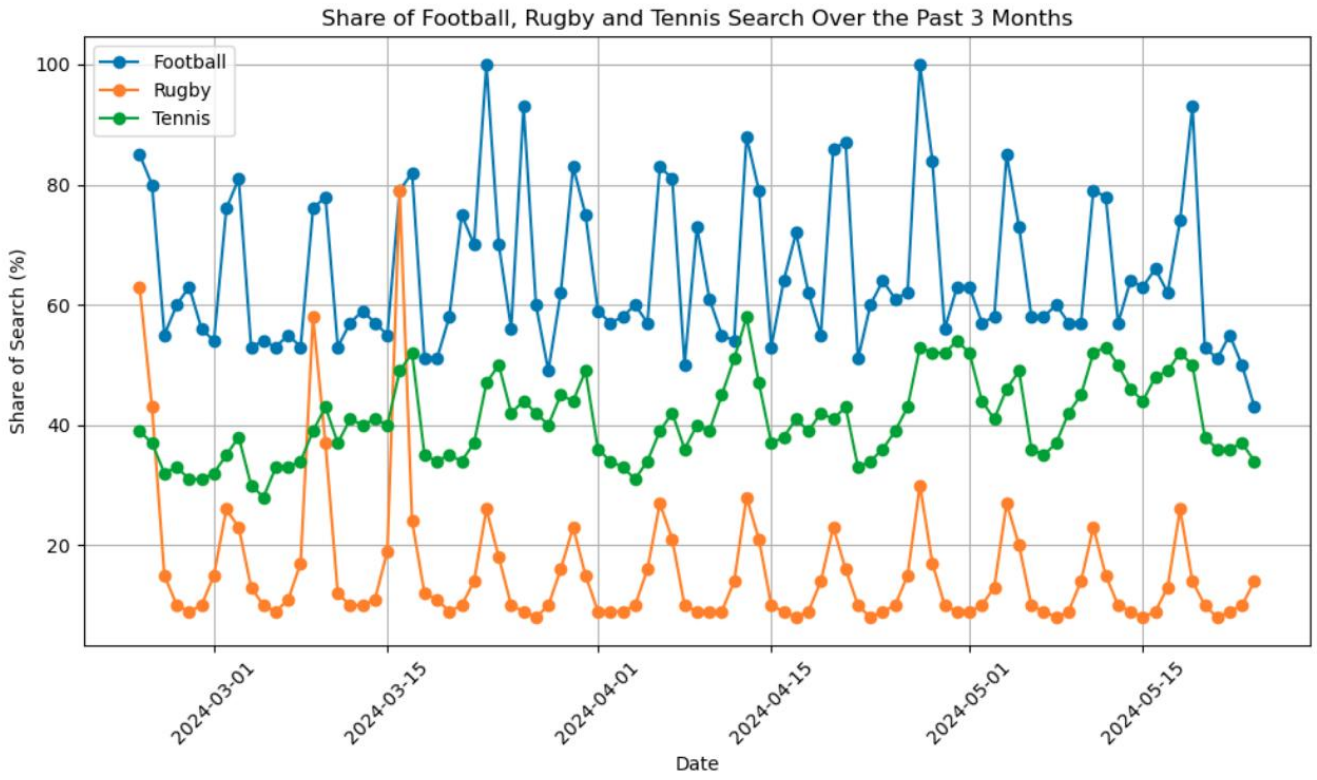
- The code reads the CSV file named "multiTimeline_FRT_Worldwide.csv" into a DataFrame df using the **pd.read_csv()** function. This CSV file contains search data for three terms (Football, Rugby, and Tennis) over the past 3 months.

Converting Date Column to Datetime Format:

The 'Day' column in the DataFrame is converted to datetime format using the **pd.to_datetime()** function. This ensures that the dates are treated as datetime objects, allowing for easier manipulation and plotting.

Plotting:

- A new figure is created with a specified size using **plt.figure(figsize=(25, 9))**.
- Line plots are created for each search term (Football, Rugby, and Tennis) using **plt.plot()** with the date column **(df['Day'])** as the x-axis and the corresponding search data columns **(df['Football:'], df['Rugby:'], df['Tennis:'])** as the y-axis. Marker 'o' is used to indicate data points.
- The plot is customized with a title, x-axis label, y-axis label, legend, gridlines, and rotated x-axis labels for better readability.
- Finally, the plot is displayed using **plt.show()**.



2. Retrieves data from the Google Trends API and plots the share of search Trends

This solution approach retrieves data from the Google Trends API and plots the share of search over the past 3 months for three terms: "Football," "Rugby," and "Tennis" using Matplotlib. Let's break down the code:

Importing Required Libraries:

- from **pytrends.request import TrendReq**: This imports the **TrendReq** class from the **pytrends.request** module, which is used to make requests to the Google Trends API.
- **import matplotlib.pyplot as plt**: This imports the **pyplot** module from the **matplotlib** library, which is used for creating plots.
- **import pandas as pd**: This imports the **pandas** library, which is used for data manipulation and analysis.

Connecting to Google Trends API:

- An instance of the **TrendReq** class is created to connect to the Google Trends API.

Defining Search Terms and Timeframe:

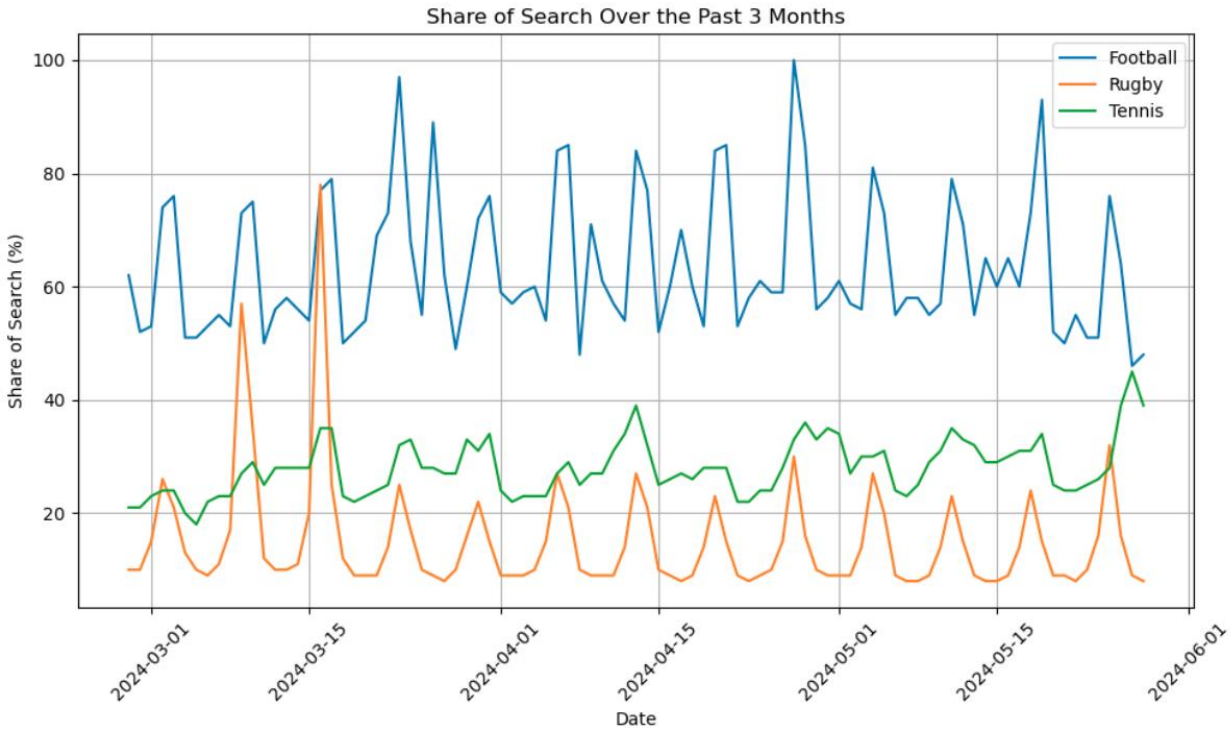
- search_terms = ['Football', 'Rugby', 'Tennis']: This defines a list of search terms for which data will be retrieved.
- timeframe = 'today 3-m': This specifies the timeframe for which data should be retrieved, in this case, the past 3 months.

Fetching Google Trends Data:

- The **build_payload** method is called to build the payload for the Google Trends API request with the specified search terms and timeframe.
- The **interest_over_time method** is called to retrieve the interest over time data from the Google Trends API for the specified search terms and timeframe.

Plotting the Data:

- A new figure is created with a specified size using **plt.figure(figsize=(10, 6)).**
- A loop iterates over each search term in the **search_terms** list.
- For each term, a line plot is created using **plt.plot** with the date index and search interest data.
- The plot is customized with a title, x-axis label, y-axis label, legend, gridlines, and rotated x-axis labels for better readability.
- Finally, the plot is displayed using **plt.show()**



DB Developer – SQL Assessment

Introduction

We process a large amount of web traffic data that is then used when we review the performance of our Direct Response advertising. For most clients we use our own tracking script which collects around 12M daily web page hit records across those clients. For a few of our other large clients we process their own sourced web visit data and some of the tables contain billions of web traffic records.

For the 1st interview stage you will need to solve the SQL assessment questions in this document: a strong, commerciallyapplicable use of T-SQL is important for this role. Your answers should be sent to Client as soon as possible and will be used in a screening process for candidates to pass to the 2nd Interview stage, a video conference call with the Director of Technology.

Be prepared to answer questions about your assessment answers during that conference call interview. Ideally limited use of Google should be made during this 1st stage assessment. You will be asked if you needed to use Google.

Task 1: Find the clashes

For this task you need to write code to find the clashing records:

- *No Googling the specific scenario*
- *Please write the query with "best practice" code structure in mind (how you would do it in production)*
- *Try to not only solve this, but solve it with optimised code*
- *Consider how you would explain why you solved the problem with your specific method*
- *All code should be in T-SQL*

Consider that you want to identify clashes for a set of trainers based on their class schedule. Specifically, the class that starts while another class is still busy. Create a query to pull back all records that have overlapping datetimes for a specific Trainer from the sample rows below. In other words: clashes for each individual trainer, not clashes with another trainer's classes. Your query should identify the clashing records as per the sample, but also for any other rows that can be added into the table in the future). Ensure your solution also work for new data that might be added into the table.

Table A

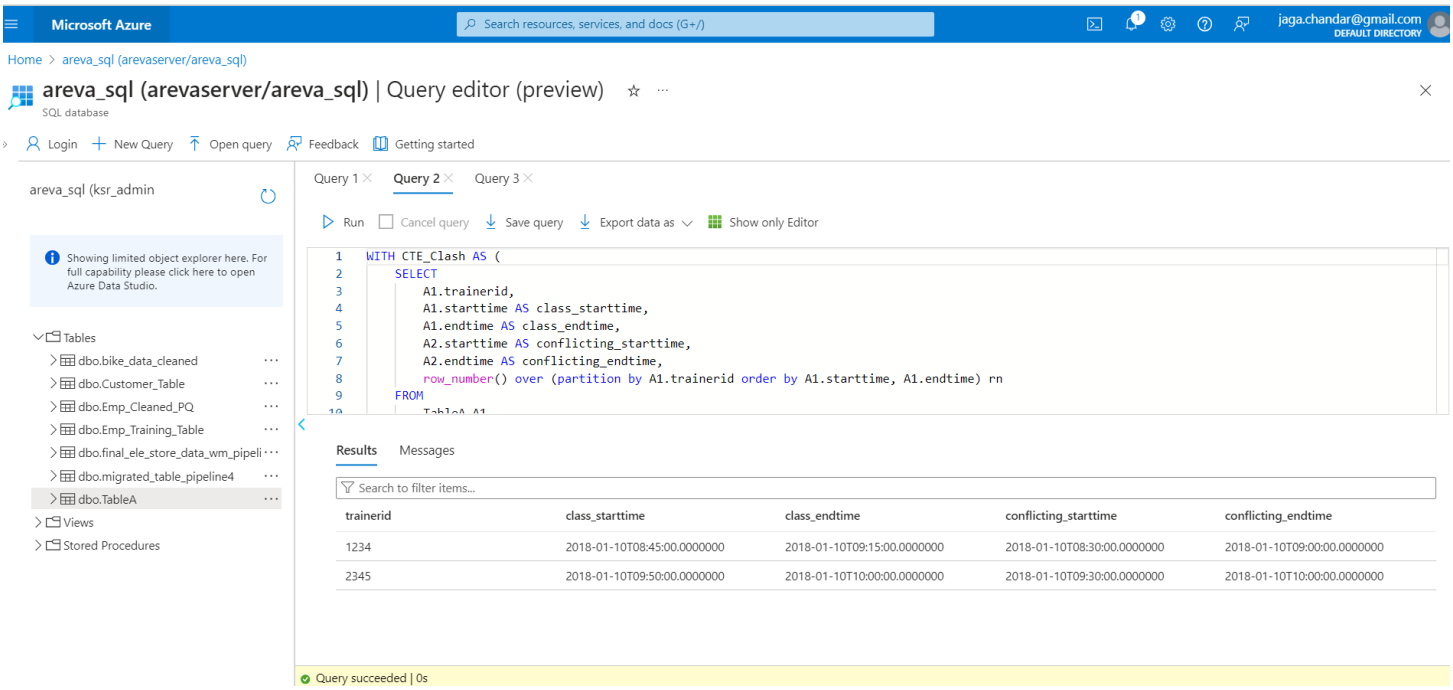
trainerid	starttime	endtime
1234	01/10/2018 08:30	01/10/2018 09:00
1234	01/10/2018 08:45	01/10/2018 09:15
1234	01/10/2018 09:30	01/10/2018 10:00
2345	01/10/2018 08:45	01/10/2018 09:15
2345	01/10/2018 09:30	01/10/2018 10:00
2345	01/10/2018 10:50	01/10/2018 11:00
2345	01/10/2018 09:50	01/10/2018 10:00

For this dataset above your query should identify that there are clashes for the following records only:

trainerid	starttime	endtime
1234	01/10/2018 08:45	01/10/2018 09:15
2345	01/10/2018 09:50	01/10/2018 10:00

Solution:

Created table in AZURE SQL Database as below screens:



T-SQL Query:

```
 -- T-SQL query serves the purpose of identifying clashes for each trainer based on their
class schedules.
WITH CTE_Clash AS (
    SELECT
        A1.trainerid,
        A1.starttime AS class_starttime,
        A1.endtime AS class_endtime,
        A2.starttime AS conflicting_starttime,
        A2.endtime AS conflicting_endtime,
        row_number() over (partition by A1.trainerid order by A1.starttime, A1.endtime) rn
    FROM
        TableA A1,TableA A2
        WHERE A1.trainerid = A2.trainerid
                AND A1.starttime < A2.endtime
                AND A1.endtime > A2.starttime
                AND A1.starttime <> A2.starttime
)
SELECT DISTINCT
    c.trainerid,
    c.class_starttime,
    c.class_endtime,
    c.conflicting_starttime,
    c.conflicting_endtime
FROM
    CTE_Clash c
where rn > 1
```

T-SQL Query using JOIN:

```
WITH CTE_Clash AS (
    SELECT
        A1.trainerid,
        A1.starttime AS class_starttime,
        A1.endtime AS class_endtime,
        A2.starttime AS conflicting_starttime,
        A2.endtime AS conflicting_endtime,
        row_number() over (partition by A1.trainerid order by A1.starttime, A1.endtime) rn
    FROM
        TableA A1
    JOIN
        TableA A2 ON A1.trainerid = A2.trainerid
                 AND A1.starttime < A2.endtime
                 AND A1.endtime > A2.starttime
                 AND A1.starttime <> A2.starttime
)
SELECT DISTINCT
    c.trainerid,
    c.class_starttime,
    c.class_endtime,
    c.conflicting_starttime,
    c.conflicting_endtime
FROM
    CTE_Clash c
where rn > 1
```

Solution - Explanation:

Common Table Expressions (CTEs): The query starts with a CTE named **CTE_Clash**, which computes potential clashes between class schedules for each trainer. This approach helps in breaking down the logic into manageable parts and improves readability.

Meaningful Aliases and Column Names: Using aliases for tables and prefixing column names with them improves readability and reduces ambiguity, making the query easier to understand and maintain.

JOIN Syntax: First query using the implicit join syntax (using comma) and Second query using explicit INNER JOIN syntax for better readability and maintainability.

Window Function (ROW_NUMBER()): The **ROW_NUMBER()** function assigns a unique sequential number to each row within a partition of a result set. In this case, it helps in identifying potential clashes and partitioning them by trainer ID.

SELECT DISTINCT: We can remove the DISTINCT keyword in the final SELECT statement because it's redundant. Since we are filtering rows where **rn > 1**, there's no need for **DISTINCT** as each row in the result set is unique.

By following these practices, the query is not only optimized for performance but also easy to understand, maintain, and troubleshoot in a production environment.