```
# Uses the local database and a stored procedure that accepts 'timeframe'  (hours/ week/month) as input

import pyodbc
import pandas as pd
from tabulate import tabulate
from termcolor import colored
from colored import fg
from flask import Flask, render_template, request
from datetime import datetime, timedelta

app = Flask(__name__)
# server = 'tp-dev-sql.database.windows.net'  # Replace with your server name or IP
# Define connection parameters
server = r'localhost\SQLEXPRESS'  # Change to your SQL Server instance name
database = 'Staging_Web_Interactions_18Mar25_Bkp'  # Your database name
# stored_procedure = 'GetAgencyPerformanceCounts'
stored_procedure ='GetAllCountsPerAgency'

# timeframe='month'
# Windows Authentication (Trusted Connection)
# connection_string = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database} " #;UID={username};PWD={password}"

@app.route('/')
def display_data():
    try:
        # Create connection string
        connection_string = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;"

        # Establish the connection
        connection = pyodbc.connect(connection_string)

        # Create a cursor to execute SQL queries
        print("Connection to SQL Server database established successfully.")
        cursor = connection.cursor()
        timeframe =  input ("Enter the timeframe (hours/week/prev_week/month) : ") #timeframe
        # query = """  """
        # cursor.execute(query, (timeframe, ))
        cursor.execute(f"EXEC {stored_procedure} @timeframe =?", timeframe)
        rows = cursor.fetchall()
#
```

```
        # Get current date
        current_date = datetime.now()

        # Calculate date ranges based on timeframe
        if timeframe == 'hours':
            start_date = (current_date - timedelta(hours=24)).strftime('%Y-%m-%d
 %H:%M:%S')
            end_date = current_date.strftime('%Y-%m-%d %H:%M:%S')
        elif timeframe == 'week':
            start_date = (current_date - timedelta(days=7)).strftime('%Y-%m-%d')
            end_date = current_date.strftime('%Y-%m-%d')
        elif timeframe == 'prev_week':
            start_date = (current_date - timedelta(days=14)).strftime('%Y-%m-%d')
            end_date = (current_date - timedelta(days=7)).strftime('%Y-%m-%d')
        elif timeframe == 'month':
            start_date = (current_date - timedelta(days=30)).strftime('%Y-%m-%d')
            end_date = current_date.strftime('%Y-%m-%d')
        else:
            start_date = "N/A"
            end_date = "N/A"
        # Get column names
        columns = [column[0] for column in cursor.description]
        # percentage_columns = ['UWBlockPercentage', 'NonUWErrorPercentage', '
SuccessfulQuotePercentage']

        # Clean the rows to remove newline characters
        cleaned_rows = [
            tuple(str(value).replace("\n", " ").strip() if isinstance(value, str) else value
for value in row)
            for row in rows
        ]

        # Create a DataFrame
        df = pd.DataFrame.from_records(cleaned_rows, columns=columns)
        df = df[df["AgencyName"] != "Agency not mapped"]

        # Replace \n in the DataFrame for clean display
        df.replace(r'\n', ' ', regex=True, inplace=True)
        percentage_columns = ['UWBlockPercentage', 'NonUWErrorPercentage',
'SuccessfulQuotePercentage' ,'DeclinedQuotePercentage']

        if not df.empty:
            for col in percentage_columns:
                df[col] = df[col].apply(lambda x: f"{x:.2f}%")

            # Function to color the text based on PerformanceStatus
        def colorize_text(df, text_column):
```

```
            """
            Applies color to text based on color names in a dataframe column.
            """
            def apply_color(row):
                color = row[text_column]
                return [f"color: {color}" if pd.notna(color) else "" for _ in row]

            # Apply styling
            styled_df = df.style.apply(apply_color, axis=1)
            styled_df = styled_df.hide(axis="columns", subset=[text_column])
            return styled_df

            # styled_df = df.style.apply (lambda row: [f"color: {row[text_column]}" if
pd.notna(row[text_column]) else "" for _ in row], axis=1).hide_columns([
text_column])
        styled_df = colorize_text(df, 'PerformanceStatus')
        # except pyodbc.Error as e:
        #     print("Error while connecting to SQL Server:", e)
        #
        # finally:
        #     # Clean up and close the connection
        #     if 'connection' in locals() and connection:
        connection.close()
        print("Connection closed.")
        return render_template('AgencyAnalysisTable.html', tables=[styled_df.
to_html(classes='data', header="False")],
                                                    timeframe=timeframe, start_date=
start_date, end_date=end_date)

    except pyodbc.Error as e:
        print(colored(f"Error while connecting to SQL Server: {e}", "red"))

    finally:
        # Clean up and close the connection
        # if 'connection' in locals() and connection:
        #     connection.close()
            print(colored("Connection closed.", "blue"))
if __name__ == '__main__':
    app.run(debug=True)
```

```
# import pandas as pd
# from functools import reduce
# def thanos_blip(age, blip_time = 5):
#  return age + blip_time
#
# data = {
# 'Name': ['Spider-Man', 'Black Panther', 'Doctor Strange', 'Scarlet Witch'],
#  'Age': [16, 35, 45, 30] }
#
# df = pd.DataFrame(data)
# filtered_ages = df[(df['Age'] > 30)]['Age'].map(lambda x: thanos_blip(x))
#
# result = reduce(lambda x, y: x + y, filtered_ages)
# print(result)
```