"Public Concern vs. Emission Targets"

Question

How does public concern about climate change correlate with national greenhouse gas emission reduction targets across different countries?

Description

The aim of the files address the relationship between public concern about climate change and the rigor of national emission reduction policies. The goal is to determine if higher public concern correlates with stricter policies, or if discrepancies suggest areas for increasing public awareness to support more robust climate action.

Data Source

Data Source1: Climate change mitigation policies and measures (greenhouse gas emissions)

- Metadata URL: https://data.europa.eu/data/datasets/data_climate-change-mitigation-policies-and-measures-1?locale=en
- Data URL: https://www.eea.europa.eu/data-and-maps/data/climate-change-mitigation-policies-and-measures-1/pam-table/climate-change-mitigation-policies-and-3/download.csv
- Data Type: CSV
- License: The United Framework Convention on Climate Change (UNFCCC) or the EEA.
- Description: This file likely details national greenhouse gas emission reduction targets, reflecting countries' commitments to international agreements. Analyzing it can reveal insights into governmental responses to climate challenges.

Data Source2: International Climate Change Opinion Survey

- Metadata URL: https://data.europa.eu/data/datasets/https-opendata-edf-fr-explore-dataset-enquete-dopinion-internationale-sur-le-changement-climatique-obscop-?locale=en
- Data URL: https://opendata.edf.fr/api/explore/v2.1/catalog/datasets/enquete-dopinion-internationale-sur-le-changement-climatique-obscop/exports/csv
- Data Type: CSV
- License:https://creativecommons.org/licenses/by-nc-sa/4.0/
- Description: This file presumably contains survey data on public opinions and concerns about climate change across different countries, detailing perceptions of the issue's severity, the urgency for action, and personal impacts.

Data Pipeline

Overview:

The technology which is used in this data pipeline is Python. is involves 2 parts:

- Data Manipulation: In this phase, data is first extracted and then cleaned, normalized, and transformed into a suitable format for analysis. To achieve this, libraries such as Pandas, StringIO, logging, and requests are utilized.

- Storing data: In this phase, the transformed data is loaded into a database. To achieve this, libraries such as sqlite3 and logging are utilized.

Transformation and Cleaning Steps:

- **"download_and_process_data" Function:**

```python
def download_and_process_data(url,sep, encoding='utf-8', low_memory=False):
    """Download and process CSV data from a URL."""
    response = requests.get(url)
    if response.status_code == 200: # The request was successfully received, understood, and processed by the server
        data = StringIO(response.text)
        df = pd.read_csv(data, sep=sep, encoding=encoding, low_memory=low_memory)
        # Handle null values appropriately
        numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
        df[numeric_columns] = df[numeric_columns].fillna(0)  # Or use another placeholder
        df.fillna('Default Value', inplace=True)
        return df
    else:
        logging.error(f"Failed to download data: HTTP {response.status_code}")
        return None
```

Challenges and Solutions:

1. Reading CSV from URL: The CSV file is downloaded using the requests library to ensure the request is successfully received, understood, and processed by the server.
2. Handling Null Values: Replacing null values ensures that operations such as aggregations, comparisons, and visualizations are not affected by missing data, which can lead to errors or misleading results.

- **"process_data" Function:**

```python
def process_data(df, table_name,dbname, column_mapping):
    """Create table, and load data into the specified table within the database.
    Optionally, manipulate the DataFrame columns based on `column_mapping`."""
    conn = create_connection(dbname)
    if conn and df is not None:
        try:

            # Rename columns based on provided mapping
            if column_mapping:
                df = df.rename(columns=column_mapping)

            # Retain only the columns that have been renamed
            columns_to_keep = list(column_mapping.values())
            df = df[columns_to_keep]

            if df.columns.duplicated().any():
                logging.error("Duplicate column names found after renaming: Please check the column mapping.")
                return  # Add this to prevent attempting to load into SQL with duplicate columns

            # Load data
            df.to_sql(table_name, conn, if_exists='replace', index=False)
            conn.commit()
            logging.info(f"Data loaded successfully into {table_name}.")
        except Exception as e:
            logging.error(f"Failed to process data for {table_name}: {e}")
        finally:
            conn.close()
    else:
        logging.error("Failed to setup database or download data.")
```

Challenges and Solutions:

1. Column Renaming: Renaming columns makes the data easier to understand and maintain, especially when integrating with other data sources.
2. Filtering Columns: Reducing the dataset to only necessary columns simplifies the data structure, speeds up processing, and reduces storage requirements.

3. Checking for Duplicate Columns: Loading data with duplicate column names can cause errors or unexpected results in SQL operations. Ensuring unique column names prevents such issues.
4. Data Loading to SQLite: Storing the transformed data in a database allows for more robust data management, querying capabilities, and scalability.

Result and Limitations

- **Output Data of The Data Pipeline:**

  The output data consists of two primary SQLite database tables:
  climate_change_policies: This table stores data related to climate change mitigation policies and measures across various countries.
  climate_change_survey: This table contains data from an international survey on public opinion regarding climate change.

- **The Data Structure and Quality of The Result:**

  The data structure is highly organized within relational database tables, making it suitable for complex queries and analysis. Data quality is enhanced through:

  - Normalization: Handling null values by filling numeric columns with 0 and other columns with 'Default Value' helps maintain data integrity and avoid computational errors during analysis and also avoiding duplicated columns.
  - Standardization: Renaming columns to consistent, understandable names facilitates easier data handling and reporting.

  However, the quality might still be limited by:

  - Error Handling: The pipeline logs errors rather than halting processes, which could lead to incomplete data loading without immediate notification to the user.

- **The Data Format:**

  The output data format chosen is SQLite database tables. This format is ideal due to its persistence, scalability and flexibility.

- **Critically Reflection:**

  While the pipeline effectively transforms and stores data, several potential issues could affect the final analysis:

  - Data Accuracy: Automatic filling of missing data might mask underlying issues such as data quality or entry errors, leading to incorrect conclusions.
  - Duplicate Data: There may be duplicate records within each dataset or across datasets.
  - Error Transparency: The use of logging for error management might not be sufficient for real-time monitoring in more automated environments.