data-report

June 6, 2024

0.1 Project Report:Global Trends in Fossil CO2 Emissions and Air Travel Emission Intensities

0.1.1 Main Question:

How have global fossil CO2 emissions evolved from 1991 to 2016? What are the trends in air travel emission intensities between 2019 to 2022 period?

Introduction In our project, we delve into the exploration of two key datasets that shed light on critical environmental and economic aspects: the Emissions Database for Global Atmospheric Research (EDGAR) and the Urban Data Platform (UDP) Air Travel Emission Intensity dataset. These datasets, provided by the Joint Research Centre, offer comprehensive insights into the dynamics of fossil CO2 emissions and air travel emission intensities, spanning the years from 1970 to 2016. By leveraging these datasets, our project seeks to uncover meaningful correlations between emissions, economic indicators, and tourism activities, providing essential insights for policymakers, researchers, and stakeholders striving for sustainable development and environmental stewardship.

Data Sources Description:

1. Emissions Database for Global Atmospheric Research (EDGAR) URL: https://jeodpp.jrc.ec.europa.eu/ftp/jrc-opendata/EDGAR/datasets/v432_FT2016/EDGARv432_FT2016_CO2_per_GDP_emissions_1970-2016.csv

Reason for Selection: The EDGAR dataset was chosen due to its comprehensive coverage of global atmospheric emissions, providing detailed insights into fossil CO2 and greenhouse gas emissions over an extended period.

Source: The EDGAR dataset is provided by the Joint Research Centre (JRC), a research arm of the European Commission dedicated to providing scientific research and technological development. The dataset is publicly available through the JRC's Open Data Portal.

Content: The dataset contains time-series data on fossil CO2 and greenhouse gas emissions, disaggregated by country and economic sector. It spans the years from 1970 to 2016, offering a comprehensive view of historical emissions trends.

Data Structure and Quality: The EDGAR dataset is structured in a tabular format, typically in CSV (Comma-Separated Values) files. Each row represents a specific country or region, while columns represent different years and emission metrics. The data is of high quality, with rigorous methodologies employed for emissions estimation. However, there may be some limitations, such as missing data for certain years or regions, which may require careful handling during analysis.

2. Urban Data Platform (UDP) - Air Travel Emission Intensity URL: https://urban.jrc.ec.europa.eu/api/udp/v2/en/data/?databrick_id=739&nutslevel=0&ts=TOURISM&nutsversi
1&mpx=1&nutslevel=9&format=csv

Reason for Selection: The UDP dataset was selected for its focus on air travel emission intensities, providing valuable insights into the environmental impact of aviation activities, particularly in the context of tourism.

Source: Similar to the EDGAR dataset, the UDP dataset is also provided by the Joint Research Centre (JRC) and is accessible through their Urban Data Platform.

Content: The UDP dataset includes information on the intensity of emissions from air travel, with a focus on tourism-related activities. It covers a range of metrics related to air travel emissions, allowing for detailed analysis of trends and patterns.

Data Structure and Quality: The UDP dataset is structured in a similar tabular format to the EDGAR dataset, typically in CSV files. Each row represents a specific observation (e.g., country-year), while columns contain different variables and emission metrics. The data quality is generally high, but similar to the EDGAR dataset, there may be limitations such as missing data or inconsistencies across regions.

High-Level Data Pipeline Overview: The data pipeline implemented for this project follows a series of steps to acquire, clean, transform, and store data from two distinct sources: the Emissions Database for Global Atmospheric Research (EDGAR) and the Urban Data Platform (UDP) Air Travel Emission Intensity dataset. Python, along with libraries such as pandas and SQLAlchemy, is used to develop the pipeline.

Data Pipeline Steps:

Data Acquisition & Data Loading: The get_source_data function retrieves CSV data from the provided URLs using the requests library. Data is then read into pandas DataFrames for further processing. The transformed DataFrames are written to an SQLite database using the write_to_target function and SQLAlchemy's create_engine method.

Data Retrieval: The pipeline begins by fetching data from the specified sources using the get_source_data function. This function utilizes the requests library to retrieve data from URLs provided as inputs. The data is then read into Pandas DataFrames.

Data Transformation: After retrieving the data, various transformation functions are applied to clean and preprocess it. These functions include dropping unnecessary columns, renaming columns, fixing date formats, and transposing the data to a suitable format for analysis. This step ensures that the data is in a standardized format and ready for further processing.

Data Storage: Once the data has been transformed, it is written to a target database using the write_to_target function. The pipeline creates a SQLite database using SQLAlchemy, a Python SQL toolkit, to store the processed data. DataFrames are converted to SQL tables and written to the database for later retrieval and analysis.

Visualization: Additionally, the pipeline includes code for visualizing the processed data. Matplotlib, a popular plotting library in Python, is used to create bar charts and line plots to visualize emissions data for European countries and global regions over multiple years.

This data pipeline facilitates the automated extraction, transformation, and loading of data from the specified sources into a structured format in an SQLite database, ensuring that the data is readily available for analysis and visualization.

The data pipeline described implements several transformation and cleaning steps to prepare the raw data for analysis. Here's a summary of these steps and their rationale:

Dropping Unnecessary Columns: **Purpose & Implementation**: Certain columns in the raw datasets may not be relevant to the analysis or visualization tasks at hand. Removing these columns helps streamline the datasets, reducing complexity and improving computational efficiency. The `drop_columns` function selectively removes specified columns from the DataFrame, enhancing clarity and focus.

Renaming Columns: **Purpose & Implementation**: Column names in the raw datasets may not always be intuitive or descriptive. Renaming columns using meaningful labels improves readability and facilitates easier interpretation of the data. The `rename_columns` function employs dictionaries to map existing column names to new, more informative labels, enhancing the overall clarity of the datasets.

Fixing Date Formats: **Purpose & Implementation**: Date columns in the raw datasets may be stored in inconsistent formats or data types. Standardizing date formats ensures uniformity and compatibility across different datasets, facilitating easier manipulation and analysis. The `fix_date_format` function converts date columns to a consistent format (`'%Y-%m-%d %H:%M'`), simplifying subsequent date-related operations and visualizations.

Transposing Data: **Purpose & Implementation**: In certain cases, transposing the data can facilitate better visualization or analysis of specific trends or patterns. For example, converting rows to columns can be useful for creating time-series plots or comparing multiple variables across different categories. The `transpose_data` function pivots the data, interchanging rows and columns as needed based on specific criteria such as years and values, enabling more intuitive visualizations and insights.

Overall, these transformation and cleaning steps are essential for preparing the raw datasets for downstream analysis and visualization. By removing noise, standardizing formats, and enhancing clarity, these steps help uncover meaningful insights and facilitate informed decision-making.

Problems encountered and how I solved them: The major issue was finding a dataset with the web downloadbale link as the initial dataset I found didnt had the web link and was directly getting saved on my local computer. Initially I added the dataset to github and used the raw file in my code however after attending the lecture realised that this approach was incorrect and had to change the entire dataset and redo the datapipeline according to the new dataset.

The pipeline incorporates error handling mechanisms to address potential issues with input data or processing. These include: Before processing, the pipeline validates input data integrity, ensuring it meets expected formats and standards. This helps identify and mitigate errors early in the process.Exception handling techniques are employed to gracefully handle errors encountered during data retrieval, cleaning, or transformation. This prevents pipeline failures and allows for continued processing with minimal disruption. The pipeline logs errors encountered during execution, providing valuable insights into potential issues. This facilitates troubleshooting and debugging, enabling prompt resolution of errors to maintain pipeline integrity. By incorporating these error-handling strategies, the pipeline maintains resilience in the face of uncertainties or fluctuations in input data, ensuring consistent and reliable performance.