**Final Report**

**1. Extraction Phase:**

The data sources for the project were sourced from kaggle.com. All of them were in **csv format**. The table below provides all the data sources with their respective links and the names of the downloaded CSV files.

|  |  |  |
| --- | --- | --- |
| **Data** | **Link** | **Name of CSV file** |
| Large dataset of cities with population and latitude, longitude data | <https://www.kaggle.com/i2i2i2/cities-of-the-world> | 15000cities.csv |
| Smaller dataset of cities with temperature data | <https://www.kaggle.com/swapnilbhange/average-temperature-of-cities> | Average Temperature of Cities.csv |
| Dataset of cities and country codes | <https://www.kaggle.com/koki25ando/country-code> | country\_code.csv |

The dataset were imported using the pandas library’s “read\_csv” function and imported into dataframes. The encoding “Latin1” needed to be used instead of the default “UTF-8”. These names of these dataframes have also been provided in the table below.

|  |  |
| --- | --- |
| **CSV File** | **Dataframe Name** |
| 15000cities.csv | all\_cities |
| Average Temperature of Cities.csv | select\_cities |
| country\_code.csv | country\_codes |

2. **Transform Phase:**

2.1. Selection of Columns:

From “all\_cities”, the columns “asciiname”, “latitude”, “longitude”, “country code” and “population” were selected to be the relevant ones. The dataframes with the selected columns were named “select\_df”.

From “select\_cities”, the columns “Country”, “City” and “Year” were considered relevant.

The columns from the “country\_codes” were filtered later, as will be described. The dataframes with the selected columns were named “all\_df”.

2.2. Renaming Columns:

The asciiname was changed to City, which better describes the column.

The Year column was split into “Celsius” and “Fahrenheit” columns, giving us the average yearly temperature of the cities in Celsius and Fahrenheit units for the years (1961-1990). This is described in more detail in the cleaning section.

2.3. Cleaning Data:

The “select\_df” dataframe had the yearly average temperature in the format “{celsius temp}/n({Fahrenheit temp})”.

The Celsius and Fahrenheit temperatures were separated, into separate columns to make them easier to read. The split function was used to extract the Celsius temperature and store it in a new column, using the “/n” prompt, and the original “Year” column dropped. A separate “Fahrenheit” column was then created by converting the Celsius column to a numeric data type and then calculating the new Fahrenheit temperature from the Celsius column.

2.4. Removal of Duplicates:

For “select\_df” : The “City” column was tested for duplicates using the “City” column and the is\_unique() function. As it returned “False”, another test was conducted using the “City” and “Country” columns. This was done using the duplicated() function. When no “True” values were observed for this, it was ascertained that all the combinations of “City” and “Country” in select\_df were unique.

For “all\_df”: A similar process was run for all\_df, but with a subset combination of “City”, “Country\_Code”, “Latitude” and “Longitude”. No duplicates were found.

2.5. Removal of Null Values:

For select\_df: All columns were tested for null values using the is\_null() function. There were no null values observed.

For “all\_df”: All columns were tested for null values using the is\_null() function. 13 rows were found that had missing country codes and 2 were found where everything other than city names were missing. These two cities, namely “Irvine” and “Irving” were dropped as they had no information to contribute to the dataset. The other 13 cities were incorporated manually into an array called “null\_cities”. There were two options for handling these cities. Either the missing country codes could be added, or the cities could be dropped. While the first option made more sense, the cities were searched for in the “select\_df” dataframe, which had temperature information using the isin() function. They were found to be missing in this dataset and therefore for the purposes of correlating population, latitude, longitude, and temperature of cities by connecting the two datasets, were considered irrelevant. Therefore, these were dropped. It must also be noted here that 13 is a very small percentage of the total 23,469 cities that were present in “all\_df”.

2.6. Handling of “country\_codes” dataset:

This dataset was included because the “all\_cities” had country codes in them while “select\_cities” had country names. Therefore, a third dataset connecting these two columns were needed. This is very important as a country can have multiple cities by the same name.

However, this dataset had two sets of country codes, namely, 2-digit and 3-digit codes. Therefore, a check was done to ensure that all country codes in “all\_df” were 2-digit codes. When this was confirmed, only the “Country\_name” and “code\_2digit” columns were selected as relevant, and others were dropped. These were renamed into “Country” and “Code” respectively.

Then, this was checked for duplicate values using duplicated() method using a “Country” column and a combination of “Country” and “Code”.

Finally, the dataframe was checked for null values. Only 1 was observed, namely for Namibia. This maybe however, because the code for Namibia is “NA”. This was therefore, corrected manually. A check was performed again after that and it returned no null values.

**3. Load Phase:**

Before loading the column names had to be converted into lowercase, using the rename() and str.lower() functions, for them to work in SQL. This was done after it was realized that this was the source of error while attempting to load using capitalized column names.

A cities\_db database was created in pgAdmin4 using the columns in “all\_df”, “select\_df” and “country\_codes” as the basis to build the SQL schema. The data in all three dataframes were sufficiently structured to advocate the use of SQL databases. The schema for the SQL tables have been provided in the figure below:

Text

Description automatically generated

It is clear that the “select\_cities”, “all\_cities” and “country\_codes” are the SQL databases for “select\_df”, “all\_df” and “country\_codes” dataframes respectively.

Finally, while loading the data using sqlalchemy from the Jupyter notebook into SQL, an error was incurred owing to the fact that there was a row where the data was offset by one column. This error was not caught during the transform phase as this was neither a duplicate nor a null row. To identify this error, the data source table needed to be revisited. The error was resolved by dropping the row. While there may be an argument to fix the row data, this was avoided considering the scope of this project.