Xander

www.xandertalent.com

Certified B Corporation

Chicago On Wheels - Bicycle Renting

Est. 2014

Data Cleaning Walk Through

Salvatore Amaddio
12/04/2024

Contents

1.0. INTRODUCTION

The following is a walk-through of the cleaning process that involved a large dataset. The dataset weighs 4 GB and contains 69 CSV files. Those files contain all services (named Trips) offered from 2014 until 2022, as well as all stations where bicycles were rented. This walk-through is meant for technical stakeholders of Chicago on Wheels—Bicycle Renting.

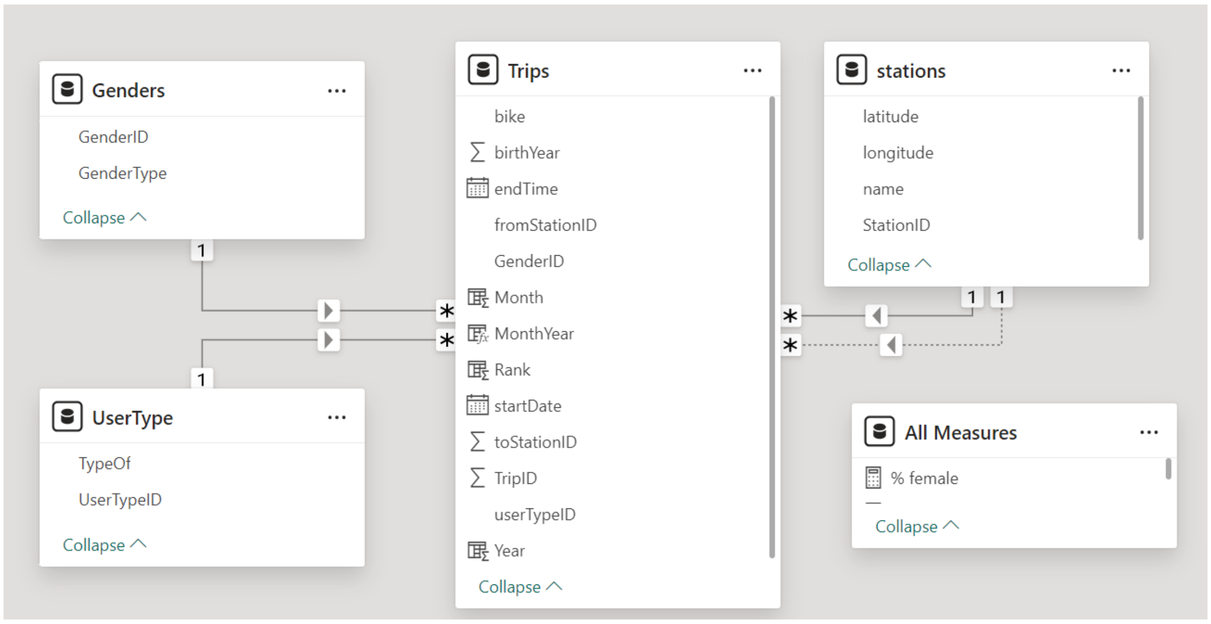
The cleaning process was conducted using Python and PowerBI. Python performed an initial cleaning process by fixing the dates' format and column naming. Finally, it merged the files by producing two final CSV files named ***data_trips*** and ***stations***.

As to PowerBI, it performed the rest of the data cleaning as follows:

- Some additional small tables were created. (***Gender, UserType***)
- Two column values in the data_trips model have been replaced with an appropriate ID. (***GenderID, UserTypeID***)
- both data_trips and stations have removed Potential Duplicates.
- The model ***data_set*** was renamed as ***Trips.***
- Relationships have been set among those four models.
- An additional named ***All Measures*** was created, which keeps all measures created for data management.

1.1. ERD

The following is the Entity Relationship Diagram that was set in the PowerBI:



2.0. Python Walkthrough:

2.1. Project Structure:

The root directory of the project contains a run.py file, which is the program's entry point. However, before running the program, three folders must be downloaded and placed in the ***processed_data*** folder.

The **processed_data** folder must contain three directories named ***trips, rides*** and ***stations.*** These can be downloaded from the Google Drive folder. The **processed_data** also includes the data already cleaned by the program named ***data_trips*** and ***stations***. These two files must remain the same as they are the source models for the PowerBI Report. However, the stakeholder can still run the program, which will overwrite the two files without affecting PowerBI's behaviour.

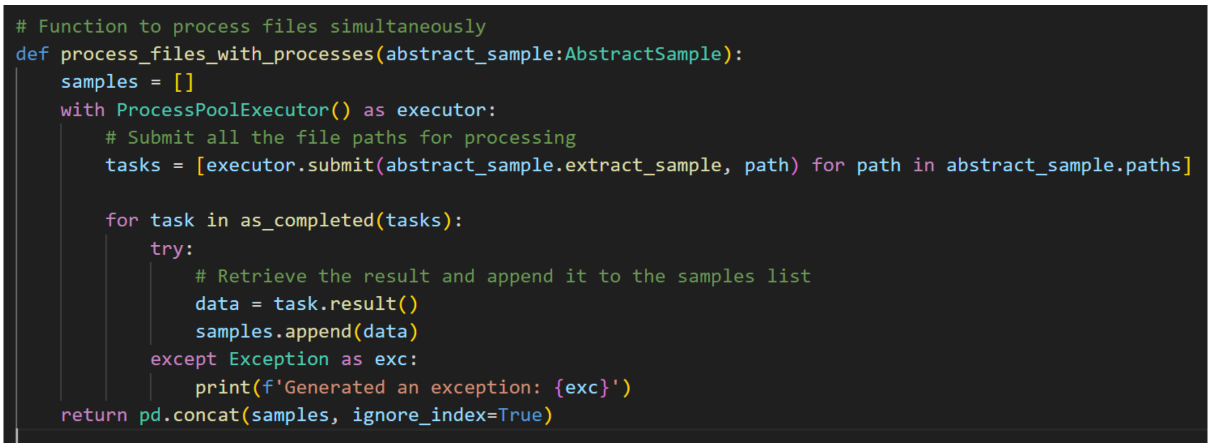
In addition, there is another directory named ***header_check***. This directory contains some CSV files made by the program. The technical Consultant used them to assess the status of the columns in the dataset. This gave a good overview of how to approach potential column naming issues. However, these files do not affect both the program and PowerBI.

Moving onto the ***Scripts*** folder, it contains a set of custom functions as classes that helped in the cleaning process. The following is a breakdown of the scripts:

- abstract_sample.py: This is an abstract class that sets some common methods that the models must follow. Here, the structure of the CSV files is cleaned per module. There is a model for each of the three directories: Trips, Rides, and ***Stations.***
- functions.py: This file contains a few functions that helped make the program.
- header_check.py: This is the file that created the content of the ***header_check*** folder.

2.2. Introduction to Tasks in Python:

Python offers a set of built-in functions and classes to run multi-treading programming. Multi-trading programming allows the execution of blocks of code simultaneously. Whenever an executor object makes a class, the program submits a Task (or **Future**) to another thread. This allows the program to keep running while the Task is completed in the background. Multiple tasks can be submitted and executed concurrently in the background.



```python
# Function to process files simultaneously
def process_files_with_processes(abstract_sample:AbstractSample):
    samples = []
    with ProcessPoolExecutor() as executor:
        # Submit all the file paths for processing
        tasks = [executor.submit(abstract_sample.extract_sample, path) for path in abstract_sample.paths]

        for task in as_completed(tasks):
            try:
                # Retrieve the result and append it to the samples list
                data = task.result()
                samples.append(data)
            except Exception as exc:
                print(f'Generated an exception: {exc}')
    return pd.concat(samples, ignore_index=True)
```

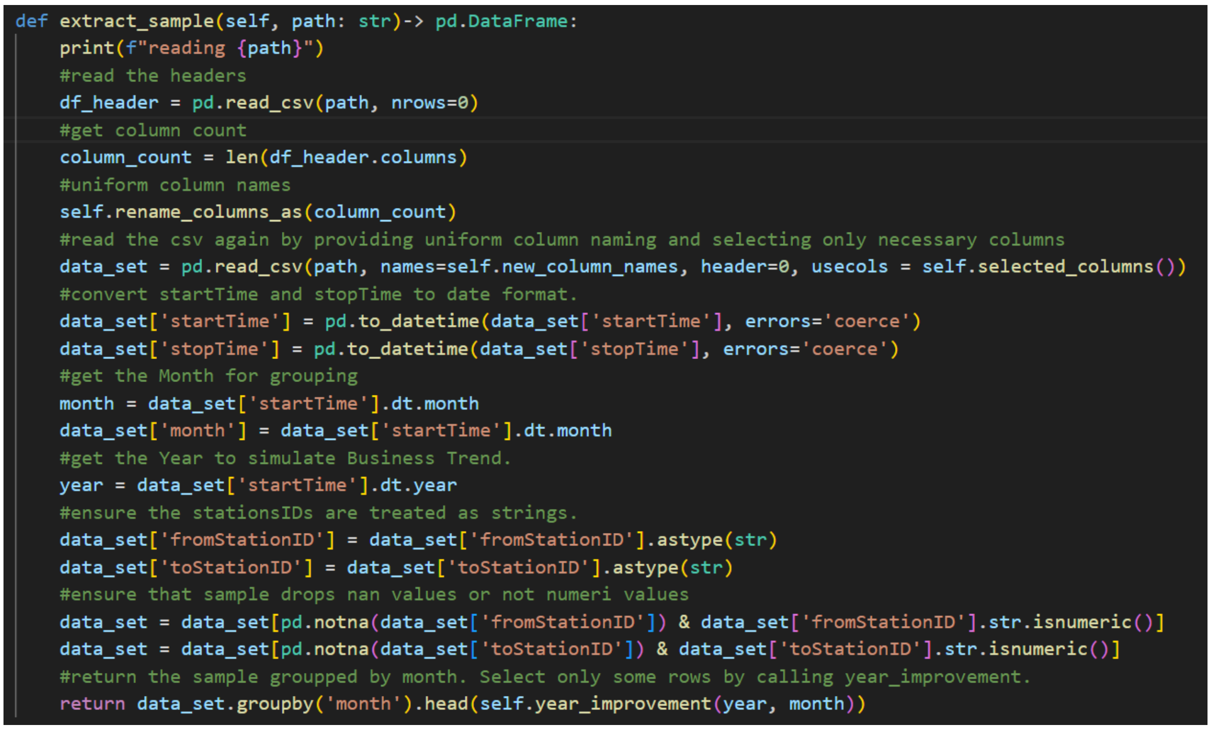
The above snippets show the execution of multiple tasks. The executor submits a list of Abstract Samples (one for each path). In the for loop, as a Task is completed, the result is taken by calling tasks.result() and appended to the Sample list object. This function allows the files in a directory to be read and cleaned concurrently.

The following snippet below submits the tasks to read the directories. It allows Python to loop through each directory and run the snippet previously shown. As a result, we have a concurrent operation that handles other concurrent operations.

```
task_trips = executor.submit(process_files_with_processes, trip)
task_rides = executor.submit(process_files_with_processes, ride)
task_stations = executor.submit(process_files_with_processes, station)
print("Waiting for reading task to complete. Go grab a coffee...")
trips_sample = task_trips.result()
rides_sample = task_rides.result()
station_sample = task_stations.result()
print("All Read")
```

2.3. Python Cleaning

The snippet below illustrates the step-by-step initial cleaning process.



```
def extract_sample(self, path: str)-> pd.DataFrame:
    print(f"reading {path}")
    #read the headers
    df_header = pd.read_csv(path, nrows=0)
    #get column count
    column_count = len(df_header.columns)
    #uniform column names
    self.rename_columns_as(column_count)
    #read the csv again by providing uniform column naming and selecting only necessary columns
    data_set = pd.read_csv(path, names=self.new_column_names, header=0, usecols = self.selected_columns())
    #convert startTime and stopTime to date format.
    data_set['startTime'] = pd.to_datetime(data_set['startTime'], errors='coerce')
    data_set['stopTime'] = pd.to_datetime(data_set['stopTime'], errors='coerce')
    #get the Month for grouping
    month = data_set['startTime'].dt.month
    data_set['month'] = data_set['startTime'].dt.month
    #get the Year to simulate Business Trend.
    year = data_set['startTime'].dt.year
    #ensure the stationsIDs are treated as strings.
    data_set['fromStationID'] = data_set['fromStationID'].astype(str)
    data_set['toStationID'] = data_set['toStationID'].astype(str)
    #ensure that sample drops nan values or not numeri values
    data_set = data_set[pd.notna(data_set['fromStationID']) & data_set['fromStationID'].str.isnumeric()]
    data_set = data_set[pd.notna(data_set['toStationID']) & data_set['toStationID'].str.isnumeric()]
    #return the sample groupped by month. Select only some rows by calling year_improvement.
    return data_set.groupby('month').head(self.year_improvement(year, month))
```

Due to the large dataset and PowerBI limitations, it was taken as just a sample from the Dataset. The program takes some records each month and simulates a Business Trend. The simulation is made by taking a different number of rows each month.

PowerBI Walkthrough:

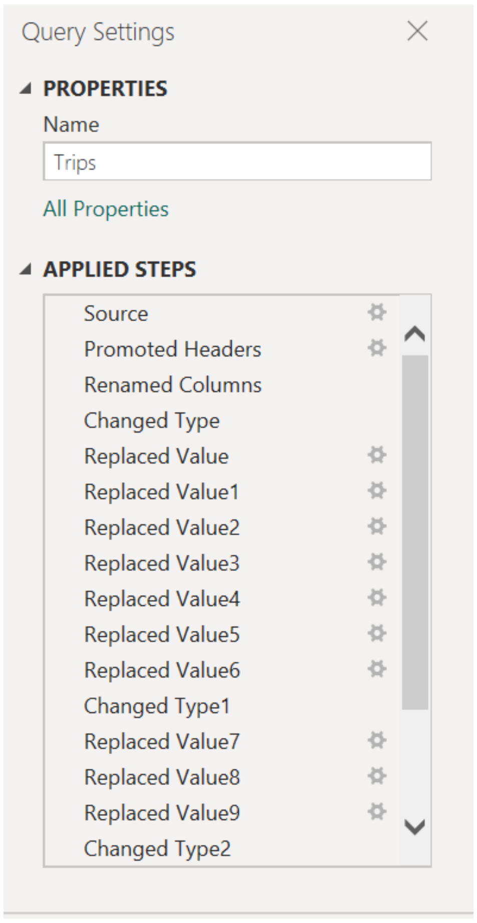
The trip model has some values that need to be added. For ***Bike,*** the dataset does not provide any information apart from an ID. However, Some Rows do have a Bike's name. Therefore, this column was set as Text.

The Dax Columns ***Year, Rank, MonthYear*** and ***Month*** were created to enforce data sorting by StartDate as much as possible. This was important for the design of the Line Chart on the report page ***YearBreakDown***.

The columns Gender and UserType were renamed as **GenderID** and **UserTypeID**. The initial text values were replaced with appropriate ID values. Finally, both columns were converted into Whole Number. This is important to set the relationship between Trips, Gender, and UserType models.

The birthYear column was set as a whole number, importing the data as a decimal number.

You can track each operation by looking at the Query Setting in the Query Editor.



Query Settings

PROPERTIES

Name

Trips

All Properties

APPLIED STEPS

Source
Promoted Headers
Renamed Columns
Changed Type
Replaced Value
Replaced Value1
Replaced Value2
Replaced Value3
Replaced Value4
Replaced Value5
Replaced Value6
Changed Type1
Replaced Value7
Replaced Value8
Replaced Value9
Changed Type2