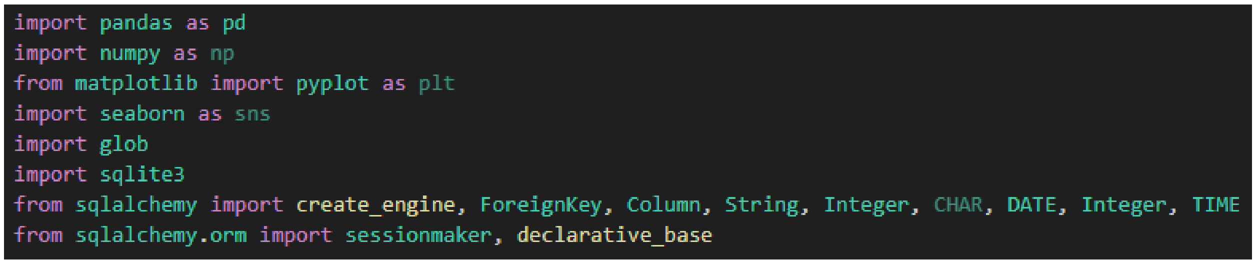
Bike Rental Data Management Overview

Overview and Goals:

- A bike rental company wants to know the relationship that the weather has on bike rentals
    - We have been given the following:
        - A CSV file containing the daily weather for the year 2016
        - A folder containing 12 CSV files for each month containing the data on that month's bike rentals
- The primary goal is to go through these CSV, clean and inspect the data, and aggregate and organize the data so that the company can better understand the relationship between bike rentals and weather.

Project Breakdown:

Part 0: Importing the necessary libraries

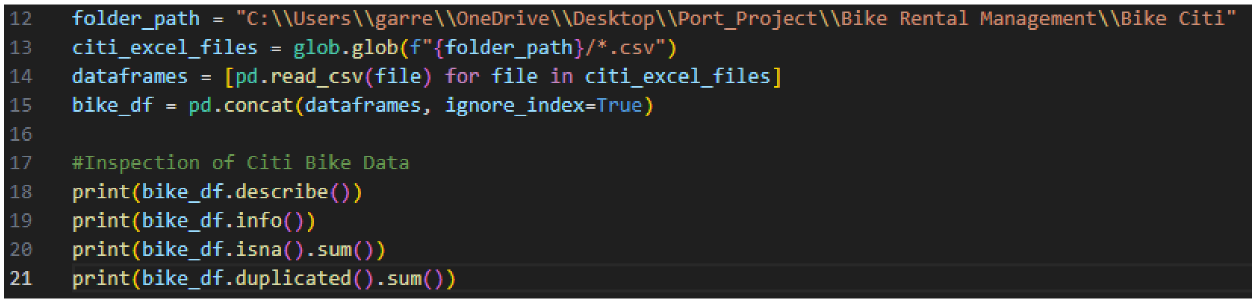


```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import glob
import sqlite3
from sqlalchemy import create_engine, ForeignKey, Column, String, Integer, CHAR, DATE, Integer, TIME
from sqlalchemy.orm import sessionmaker, declarative_base
```

- The above screenshot shows all the libraries I elected to import before starting the project. I will later describe how I used them as I further breakdown my code

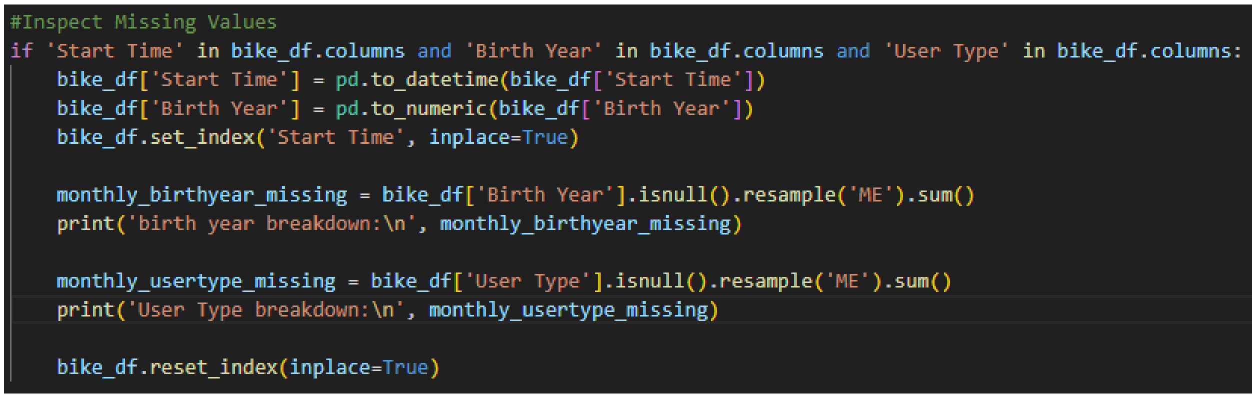
Part 1: Citi Bike Data

- I first needed to import the CSV file containing the Bike Rental Data. I used glob.glob to open all the files at once, then used pd.concat to combine them all into 1 dataframe. I then inspected the data to get a general understanding of what I was working with.

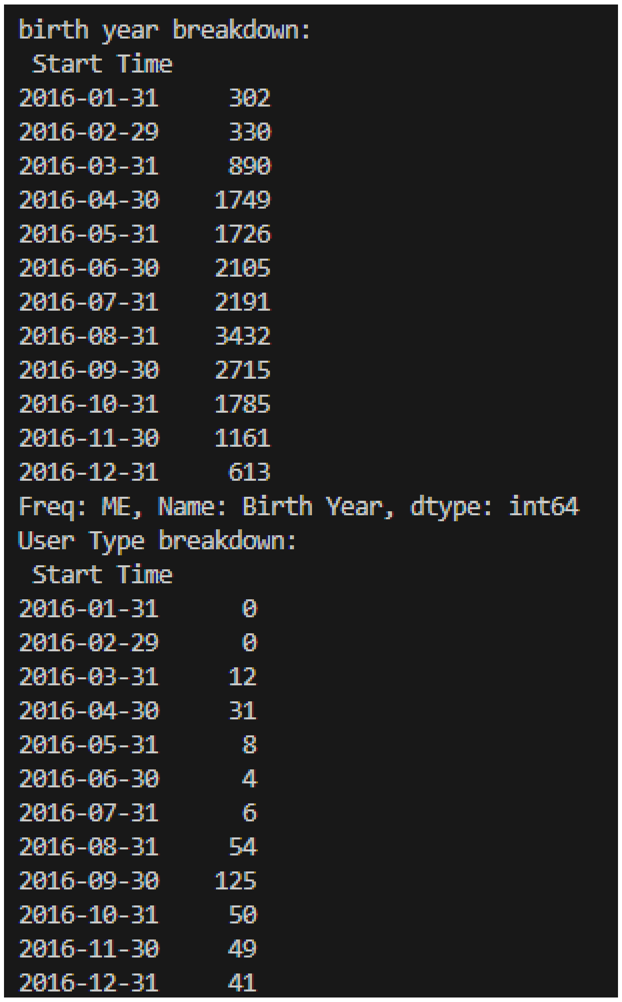


```
folder_path = "C:\\Users\\garre\\OneDrive\\Desktop\\Port_Project\\Bike Rental Management\\Bike Citi"
citi_excel_files = glob.glob(f"{folder_path}/*.csv")
dataframes = [pd.read_csv(file) for file in citi_excel_files]
bike_df = pd.concat(dataframes, ignore_index=True)

#Inspection of Citi Bike Data
print(bike_df.describe())
print(bike_df.info())
print(bike_df.isna().sum())
print(bike_df.duplicated().sum())
```

- Something I immediately noticed was that there was a very large portion of 'Birth Year' missing (7.6% of the total entries) and a rather small but noticeable amount of 'User Type' as well. I wanted to further explore to get some more information on to why this might be the case.

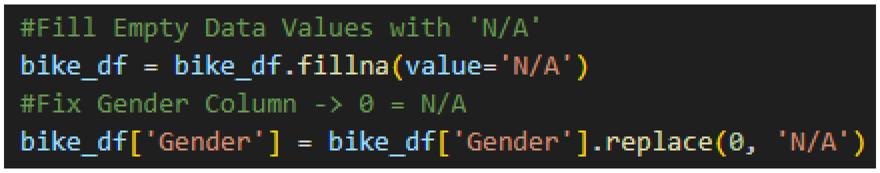


```python
#Inspect Missing Values
if 'Start Time' in bike_df.columns and 'Birth Year' in bike_df.columns and 'User Type' in bike_df.columns:
    bike_df['Start Time'] = pd.to_datetime(bike_df['Start Time'])
    bike_df['Birth Year'] = pd.to_numeric(bike_df['Birth Year'])
    bike_df.set_index('Start Time', inplace=True)

    monthly_birthyear_missing = bike_df['Birth Year'].isnull().resample('ME').sum()
    print('birth year breakdown:\n', monthly_birthyear_missing)

    monthly_usertype_missing = bike_df['User Type'].isnull().resample('ME').sum()
    print('User Type breakdown:\n', monthly_usertype_missing)

    bike_df.reset_index(inplace=True)
```



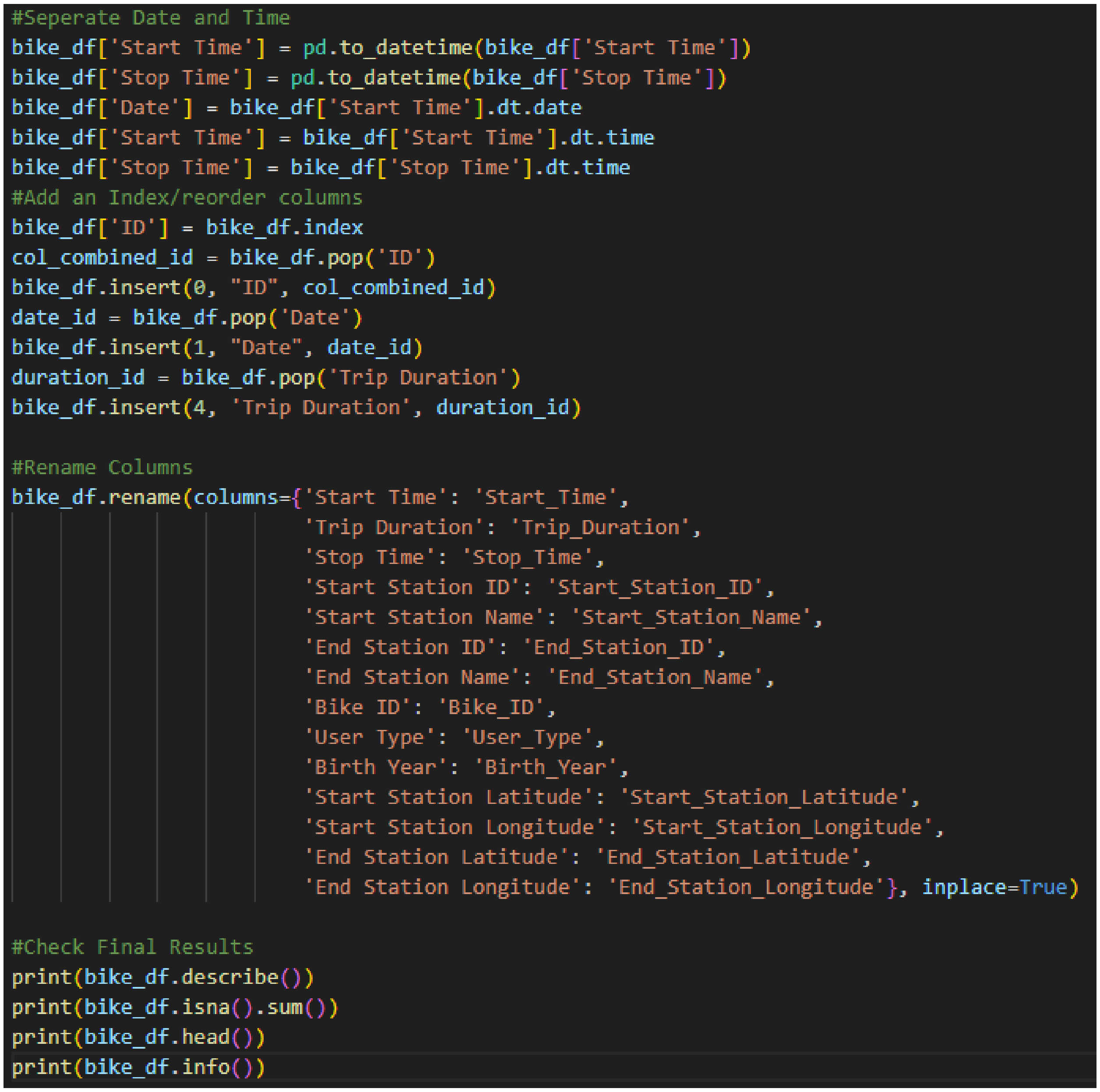
```
birth year breakdown:
 Start Time
2016-01-31     302
2016-02-29     330
2016-03-31     890
2016-04-30    1749
2016-05-31    1726
2016-06-30    2105
2016-07-31    2191
2016-08-31    3432
2016-09-30    2715
2016-10-31    1785
2016-11-30    1161
2016-12-31     613
Freq: ME, Name: Birth Year, dtype: int64
User Type breakdown:
 Start Time
2016-01-31      0
2016-02-29      0
2016-03-31     12
2016-04-30     31
2016-05-31      8
2016-06-30      4
2016-07-31      6
2016-08-31     54
2016-09-30    125
2016-10-31     50
2016-11-30     49
2016-12-31     41
```

- I cycled through all the missing Birth Year and User Types as seen above. I broke it down into a monthly basis for a more easily digestible format. Something I noticed is that it seems that the NULL values are somewhat spread out throughout the year. It seems like the last half of the year for both seems to have a bit more missing data, but it's still relatively spread out enough to conclude that there isn't a correlation between time of year and why the data is missing. Although September was the highest by far for User Type by about 2.5x, so there could be a potential correlation there.



```python
#Fill Empty Data Values with 'N/A'
bike_df = bike_df.fillna(value='N/A')
#Fix Gender Column -> 0 = N/A
bike_df['Gender'] = bike_df['Gender'].replace(0, 'N/A')
```

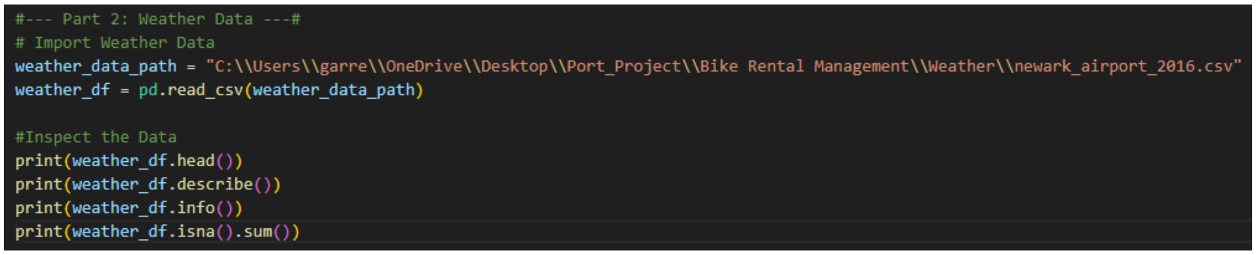
- I then decided to replace all the NULL values with 'N/A'. I also elected to replace '0' w/ 'N/A'. The rational behind this was that having a 0th gender didn't make a whole lot of sense to me, and with no other data or information about this column, I though it was best to put it as 'N/A'. Other viable option could have been 'Other', or in a real-world scenario ask the client further on this.



```
#Seperate Date and Time
bike_df['Start Time'] = pd.to_datetime(bike_df['Start Time'])
bike_df['Stop Time'] = pd.to_datetime(bike_df['Stop Time'])
bike_df['Date'] = bike_df['Start Time'].dt.date
bike_df['Start Time'] = bike_df['Start Time'].dt.time
bike_df['Stop Time'] = bike_df['Stop Time'].dt.time
#Add an Index/reorder columns
bike_df['ID'] = bike_df.index
col_combined_id = bike_df.pop('ID')
bike_df.insert(0, "ID", col_combined_id)
date_id = bike_df.pop('Date')
bike_df.insert(1, "Date", date_id)
duration_id = bike_df.pop('Trip Duration')
bike_df.insert(4, 'Trip Duration', duration_id)

#Rename Columns
bike_df.rename(columns={'Start Time': 'Start_Time',
                        'Trip Duration': 'Trip_Duration',
                        'Stop Time': 'Stop_Time',
                        'Start Station ID': 'Start_Station_ID',
                        'Start Station Name': 'Start_Station_Name',
                        'End Station ID': 'End_Station_ID',
                        'End Station Name': 'End_Station_Name',
                        'Bike ID': 'Bike_ID',
                        'User Type': 'User_Type',
                        'Birth Year': 'Birth_Year',
                        'Start Station Latitude': 'Start_Station_Latitude',
                        'Start Station Longitude': 'Start_Station_Longitude',
                        'End Station Latitude': 'End_Station_Latitude',
                        'End Station Longitude': 'End_Station_Longitude'}, inplace=True)

#Check Final Results
print(bike_df.describe())
print(bike_df.isna().sum())
print(bike_df.head())
print(bike_df.info())
```

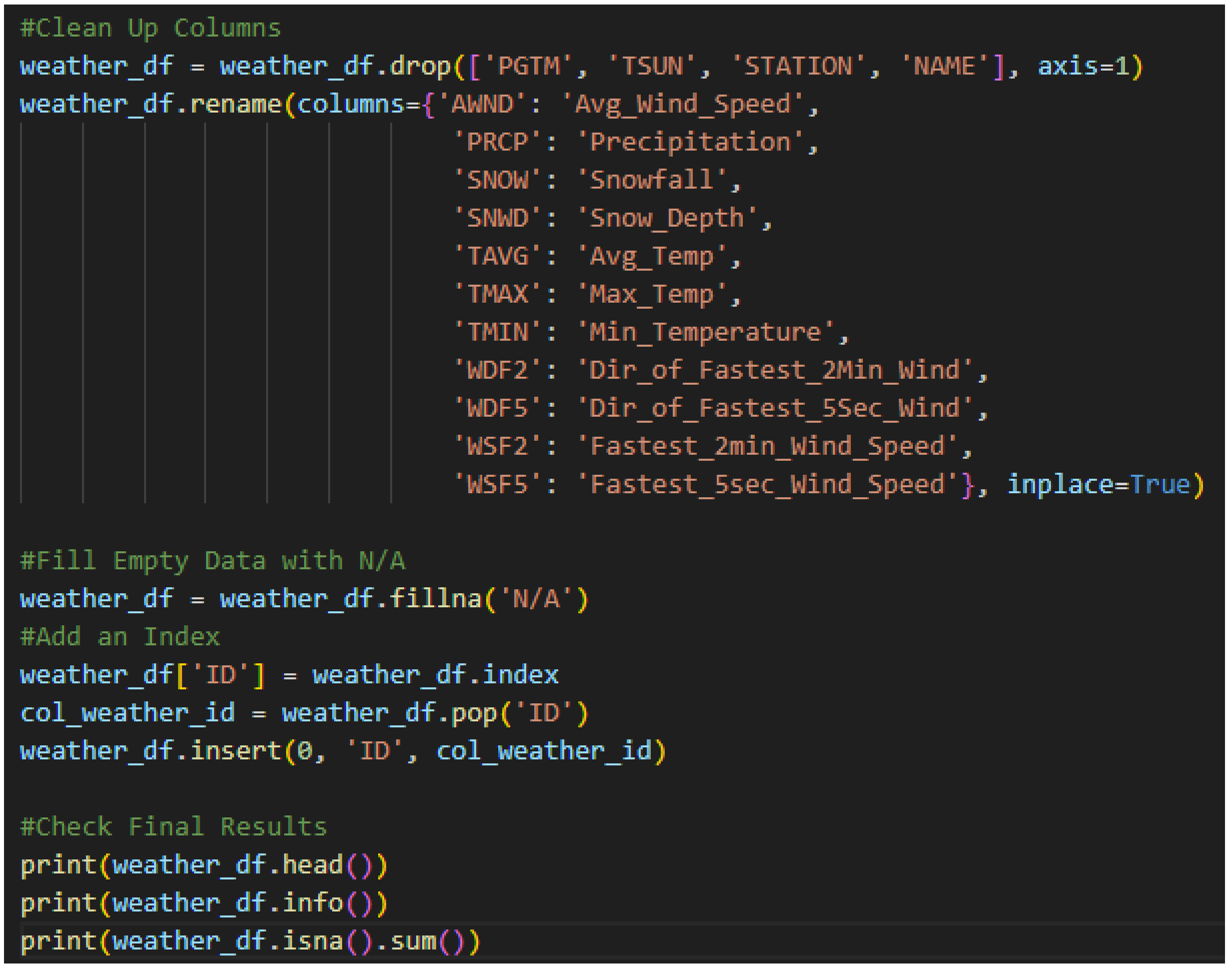
- This next section was bit of re-organization. Something I initially noticed was that in the CSV file the Date and Time were all in 1 column. While this isn't inherently bad, I figured having the Date and Time separate would make for easier data analysis down the road. I then added an index column and reorganized the order of some of the columns. I then printed my results to confirm my changes and that everything came out to the desired result.

Part 2: Importing the Weather Data



```python
#--- Part 2: Weather Data ---#
# Import Weather Data
weather_data_path = "C:\\Users\\garre\\OneDrive\\Desktop\\Port_Project\\Bike Rental Management\\Weather\\newark_airport_2016.csv"
weather_df = pd.read_csv(weather_data_path)

#Inspect the Data
print(weather_df.head())
print(weather_df.describe())
print(weather_df.info())
print(weather_df.isna().sum())
```

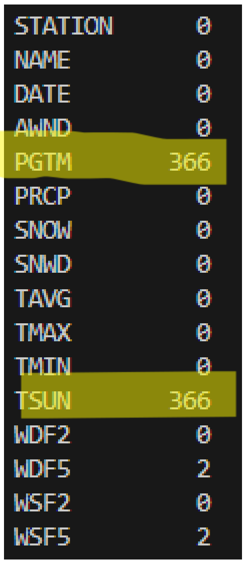
- Similarly to the bike data, I imported the weather CSV file along with doing a general inspection of the data.



```python
#Clean Up Columns
weather_df = weather_df.drop(['PGTM', 'TSUN', 'STATION', 'NAME'], axis=1)
weather_df.rename(columns={'AWND': 'Avg_Wind_Speed',
                           'PRCP': 'Precipitation',
                           'SNOW': 'Snowfall',
                           'SNWD': 'Snow_Depth',
                           'TAVG': 'Avg_Temp',
                           'TMAX': 'Max_Temp',
                           'TMIN': 'Min_Temperature',
                           'WDF2': 'Dir_of_Fastest_2Min_Wind',
                           'WDF5': 'Dir_of_Fastest_5Sec_Wind',
                           'WSF2': 'Fastest_2min_Wind_Speed',
                           'WSF5': 'Fastest_5sec_Wind_Speed'}, inplace=True)

#Fill Empty Data with N/A
weather_df = weather_df.fillna('N/A')
#Add an Index
weather_df['ID'] = weather_df.index
col_weather_id = weather_df.pop('ID')
weather_df.insert(0, 'ID', col_weather_id)

#Check Final Results
print(weather_df.head())
print(weather_df.info())
print(weather_df.isna().sum())
```

- After a quick inspection, I was able to determine the following:
    1. The entire columns for PGTM and TSUN were completely empty, and with this I concluded I should delete these columns entirely because there really isn't anything to work with.

   a. There are also 2 missing data points for WDF2 and WSF5. Having only 2/365 missing is rather small, and this could be simple data logging errors on the weather side or maybe there isn't suppose to be data there at all. Regardless I decided the best course of action was to fill the empty values with 'N/A'.

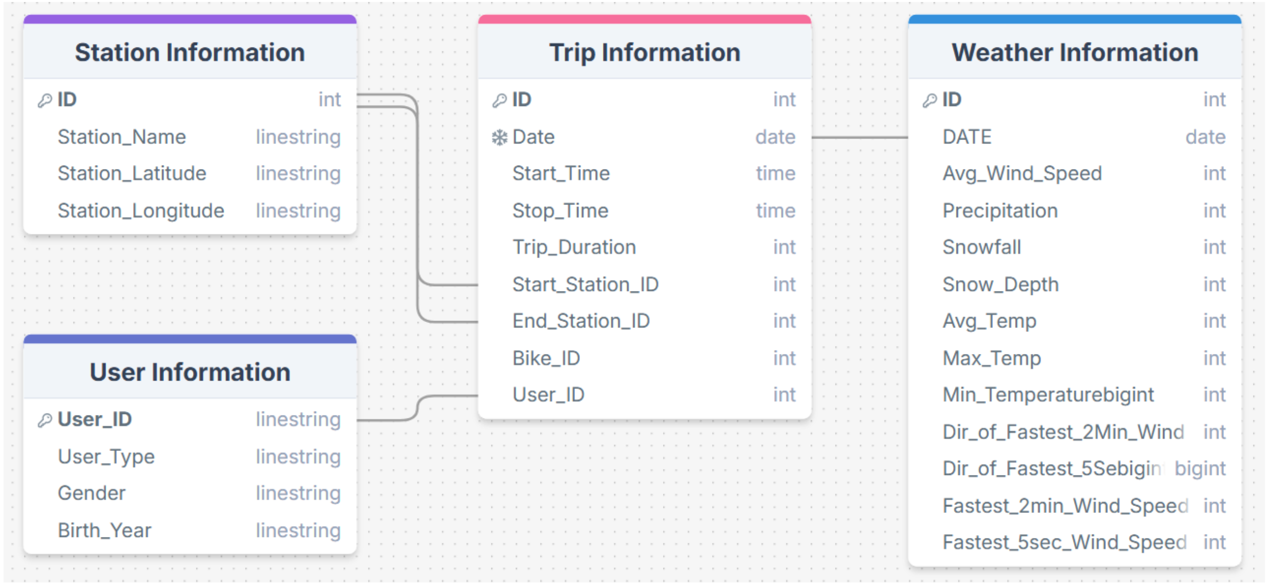


```
STATION      0
NAME         0
DATE         0
AWND         0
PGTM       366
PRCP         0
SNOW         0
SNWD         0
TAVG         0
TMAX         0
TMIN         0
TSUN       366
WDF2         0
WDF5         2
WSF2         0
WSF5         2
```

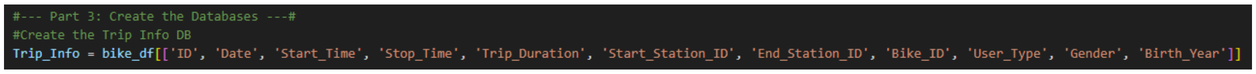
2. I also found the 'Station' and 'Name' columns to be a bit repetitive and unnecessary. Since we are only working with 1 location and this is assumed knowledge, it seems unnecessary to have the same name and station constantly repeated. These two columns were deleted as well.
3. The columns were renamed to be more descriptive, and I added an index as well. I moved around some columns and did a final check to make sure the results were what was desired.

Part 3: Creating the Databases

- Now that all our data is clean and inspected, the next step is to create the databases that will help set us up for creating the SQL DB down the line. In this step, it is crucial that we decide how we want our tables and data to be organized, and how all our tables are going to be linked together. After some thought and consideration, I am up with the following schema:

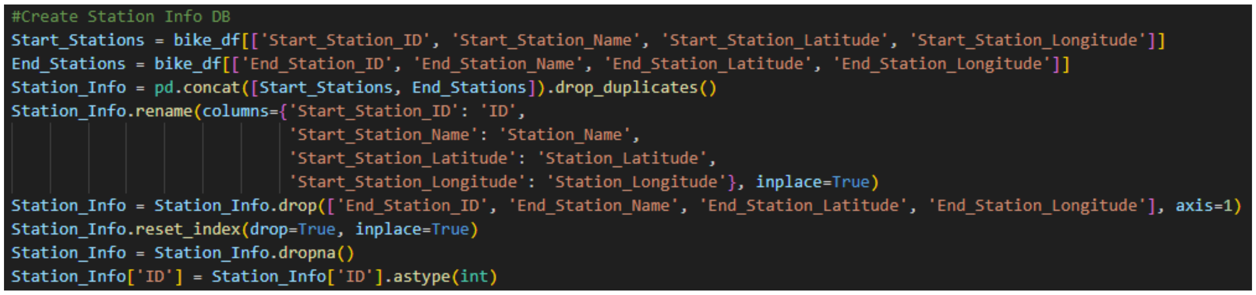


- The following is a breakdown of the thought process behind the creation of each table:



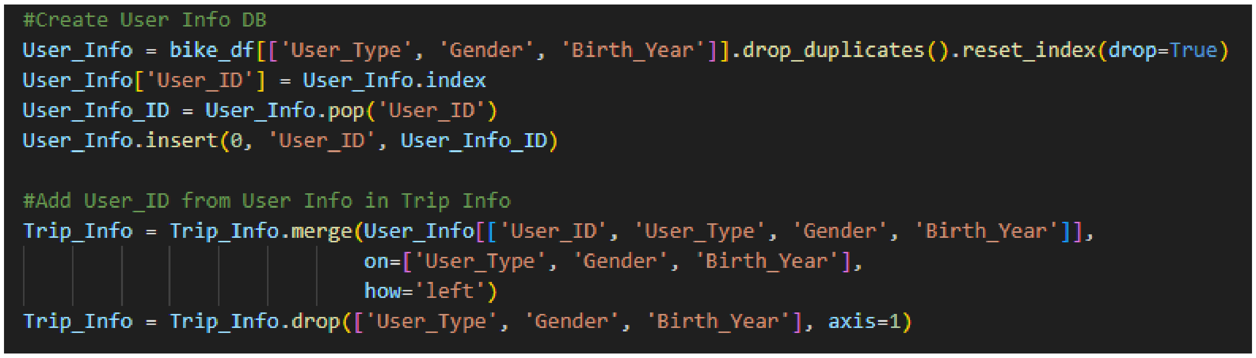
```
#--- Part 3: Create the Databases ---#
#Create the Trip Info DB
Trip_Info = bike_df[['ID', 'Date', 'Start_Time', 'Stop_Time', 'Trip_Duration', 'Start_Station_ID', 'End_Station_ID', 'Bike_ID', 'User_Type', 'Gender', 'Birth_Year']]
```

- Trip_Info was going to be the 'primary table'. Everything above from ID to User_Type was going to be information that was going to be found in this table, as it relates to the information regarding the trip. Even though I added 'Gender' and 'Birth Year' as the last two, they weren't going to be in the final databas, rather a way for us to set up our joins for later (which I will describe in a bit).



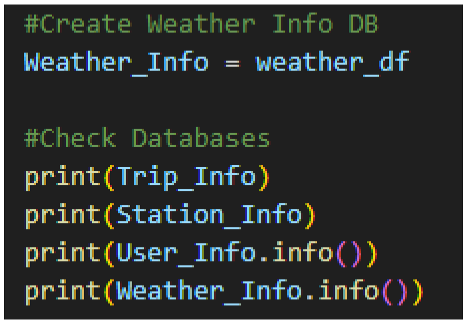
```
#Create Station Info DB
Start_Stations = bike_df[['Start_Station_ID', 'Start_Station_Name', 'Start_Station_Latitude', 'Start_Station_Longitude']]
End_Stations = bike_df[['End_Station_ID', 'End_Station_Name', 'End_Station_Latitude', 'End_Station_Longitude']]
Station_Info = pd.concat([Start_Stations, End_Stations]).drop_duplicates()
Station_Info.rename(columns={'Start_Station_ID': 'ID',
                            'Start_Station_Name': 'Station_Name',
                            'Start_Station_Latitude': 'Station_Latitude',
                            'Start_Station_Longitude': 'Station_Longitude'}, inplace=True)
Station_Info = Station_Info.drop(['End_Station_ID', 'End_Station_Name', 'End_Station_Latitude', 'End_Station_Longitude'], axis=1)
Station_Info.reset_index(drop=True, inplace=True)
Station_Info = Station_Info.dropna()
Station_Info['ID'] = Station_Info['ID'].astype(int)
```

- The general idea behind the Station Information table was to create two DBs: a start and end stations. I then pd.concat to combine these two databases, then renamed and dropped the necessary columns. I then set the Station ID as the index. The result was a clean and easy to read Station Table with the Station ID, Name along with the respective longitudes and latitudes.



```
#Create User Info DB
User_Info = bike_df[['User_Type', 'Gender', 'Birth_Year']].drop_duplicates().reset_index(drop=True)
User_Info['User_ID'] = User_Info.index
User_Info_ID = User_Info.pop('User_ID')
User_Info.insert(0, 'User_ID', User_Info_ID)

#Add User_ID from User Info in Trip Info
Trip_Info = Trip_Info.merge(User_Info[['User_ID', 'User_Type', 'Gender', 'Birth_Year']],
                            on=['User_Type', 'Gender', 'Birth_Year'],
                            how='left')
Trip_Info = Trip_Info.drop(['User_Type', 'Gender', 'Birth_Year'], axis=1)
```

- The next database that was going to be created was the "User Information'.
    1. The first step was to create the User_Info DB, which was going to consist of the User_Type, Gender and Birth_Year data from 'bike_df' from earlier.
    2. I then merged the Trip Info into the User Info table in order to populate it with the necessary info.
    3. I then drop the appropriate columns from Trip Info to avoid redundancy

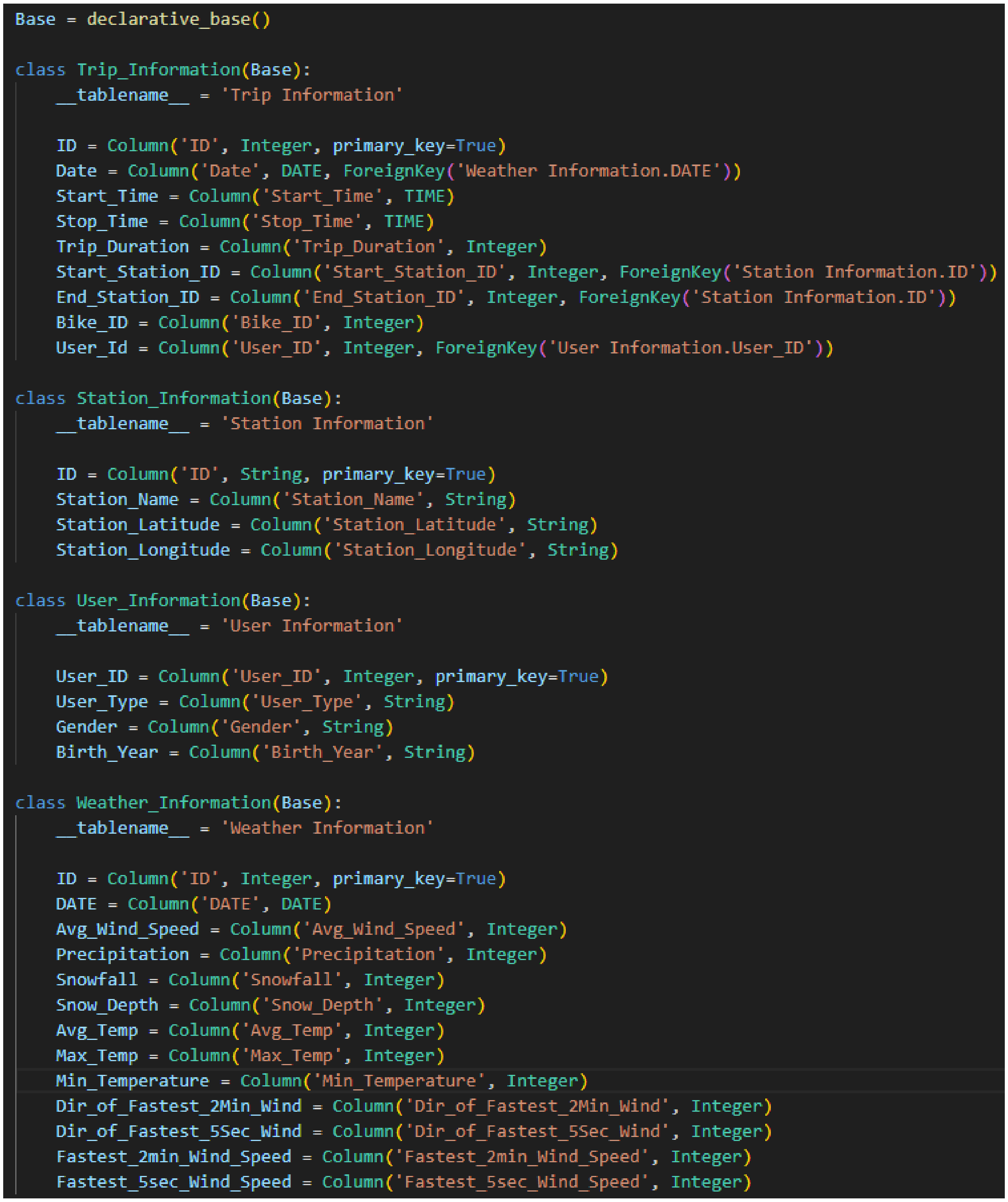


```
#Create Weather Info DB
Weather_Info = weather_df

#Check Databases
print(Trip_Info)
print(Station_Info)
print(User_Info.info())
print(Weather_Info.info())
```

- Since the weather_df was already consistent and didn't need many changes, I left it alone in the new DB
- I think printed my results to make sure I got the desired results

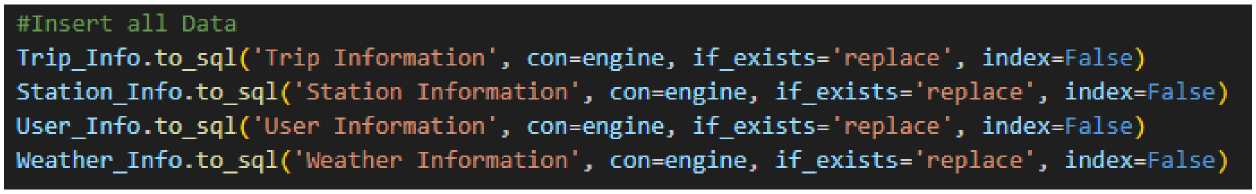
Part 4: Creating the SQL Tables

- Now that we have created our new databases based upon our schema, it is now time to create the tables. The following is code is how I created each SQL table:



```python
Base = declarative_base()

class Trip_Information(Base):
    __tablename__ = 'Trip Information'

    ID = Column('ID', Integer, primary_key=True)
    Date = Column('Date', DATE, ForeignKey('Weather Information.DATE'))
    Start_Time = Column('Start_Time', TIME)
    Stop_Time = Column('Stop_Time', TIME)
    Trip_Duration = Column('Trip_Duration', Integer)
    Start_Station_ID = Column('Start_Station_ID', Integer, ForeignKey('Station Information.ID'))
    End_Station_ID = Column('End_Station_ID', Integer, ForeignKey('Station Information.ID'))
    Bike_ID = Column('Bike_ID', Integer)
    User_Id = Column('User_ID', Integer, ForeignKey('User Information.User_ID'))

class Station_Information(Base):
    __tablename__ = 'Station Information'

    ID = Column('ID', String, primary_key=True)
    Station_Name = Column('Station_Name', String)
    Station_Latitude = Column('Station_Latitude', String)
    Station_Longitude = Column('Station_Longitude', String)

class User_Information(Base):
    __tablename__ = 'User Information'

    User_ID = Column('User_ID', Integer, primary_key=True)
    User_Type = Column('User_Type', String)
    Gender = Column('Gender', String)
    Birth_Year = Column('Birth_Year', String)

class Weather_Information(Base):
    __tablename__ = 'Weather Information'

    ID = Column('ID', Integer, primary_key=True)
    DATE = Column('DATE', DATE)
    Avg_Wind_Speed = Column('Avg_Wind_Speed', Integer)
    Precipitation = Column('Precipitation', Integer)
    Snowfall = Column('Snowfall', Integer)
    Snow_Depth = Column('Snow_Depth', Integer)
    Avg_Temp = Column('Avg_Temp', Integer)
    Max_Temp = Column('Max_Temp', Integer)
    Min_Temperature = Column('Min_Temperature', Integer)
    Dir_of_Fastest_2Min_Wind = Column('Dir_of_Fastest_2Min_Wind', Integer)
    Dir_of_Fastest_5Sec_Wind = Column('Dir_of_Fastest_5Sec_Wind', Integer)
    Fastest_2min_Wind_Speed = Column('Fastest_2min_Wind_Speed', Integer)
    Fastest_5sec_Wind_Speed = Column('Fastest_5sec_Wind_Speed', Integer)
```

- I then filled out these newly created SQL Tables with our previous created Databases:

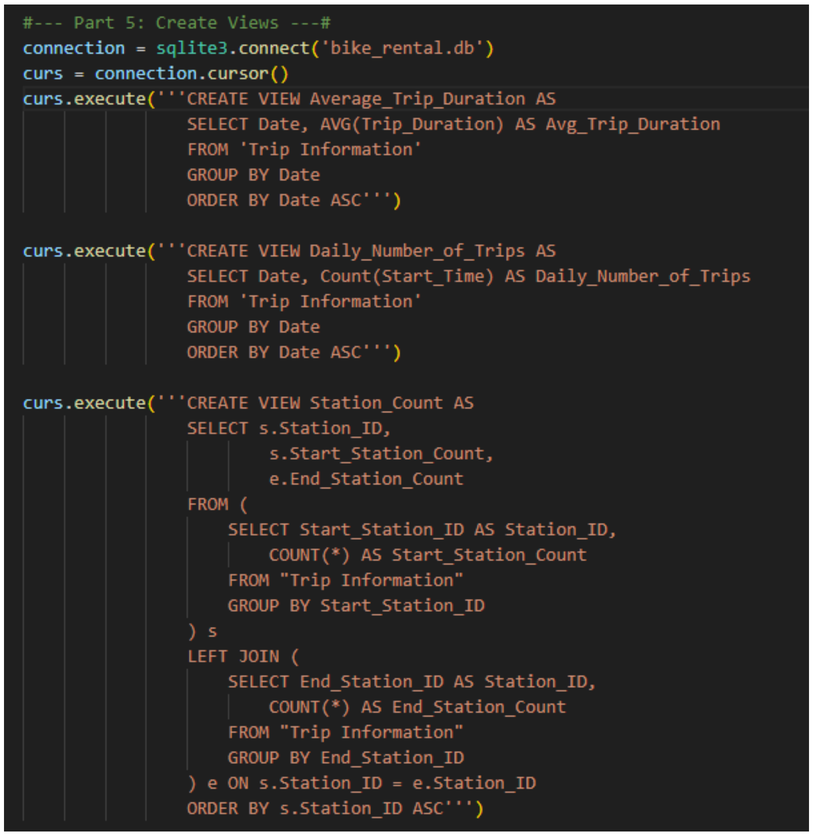


```python
#Insert all Data
Trip_Info.to_sql('Trip Information', con=engine, if_exists='replace', index=False)
Station_Info.to_sql('Station Information', con=engine, if_exists='replace', index=False)
User_Info.to_sql('User Information', con=engine, if_exists='replace', index=False)
Weather_Info.to_sql('Weather Information', con=engine, if_exists='replace', index=False)
```

- Once the program was run, our tables were successfully created.

<u>Part 5: Creating Views</u>

- I wanted to create additional views that would make it easy to visualize important data. While there are additional potential other views, I decided to go with the following three:
    1. Average Trip Duration: Calculates the average trip duration by day, which could be useful to see if the weather affects how long people decide to rent bike (such as does nicer weather really mean longer rental times)
    2. Daily Number of Trip: Calculates how many trips were taken per day. This could be useful to visual to see if there is a relation between weather and the amount of people electing to rent bikes
    3. Station Count: Takes each station and counts how many time that station was the starting location or the destination over the entire year. This is crucial information to see what locations are hotspots, or location that might not be worth keeping due to little travel.
- The following is the code that I used to achieve this:



```python
#--- Part 5: Create Views ---#
connection = sqlite3.connect('bike_rental.db')
curs = connection.cursor()
curs.execute('''CREATE VIEW Average_Trip_Duration AS
                SELECT Date, AVG(Trip_Duration) AS Avg_Trip_Duration
                FROM 'Trip Information'
                GROUP BY Date
                ORDER BY Date ASC''')

curs.execute('''CREATE VIEW Daily_Number_of_Trips AS
                SELECT Date, Count(Start_Time) AS Daily_Number_of_Trips
                FROM 'Trip Information'
                GROUP BY Date
                ORDER BY Date ASC''')

curs.execute('''CREATE VIEW Station_Count AS
                SELECT s.Station_ID,
                        s.Start_Station_Count,
                        e.End_Station_Count
                FROM (
                    SELECT Start_Station_ID AS Station_ID,
                        COUNT(*) AS Start_Station_Count
                    FROM "Trip Information"
                    GROUP BY Start_Station_ID
                ) s
                LEFT JOIN (
                    SELECT End_Station_ID AS Station_ID,
                        COUNT(*) AS End_Station_Count
                    FROM "Trip Information"
                    GROUP BY End_Station_ID
                ) e ON s.Station_ID = e.Station_ID
                ORDER BY s.Station_ID ASC''')
```

Potential Changes:

- While the primary goal of this project was achieved, that doesn't there aren't any potential improvements.

Cleaning Up the Tables

- While I successfully achieved the goal of the projects, there are a handful of things I am going to go back and clean up in relation to the format of the tables. They go as follows:
    - Trip Information: the start and stop time have a lot of unnecessary 0s, as well as the trip duration columns units are not specified
    - User Information: The Gender columns (N/A, 1, 2) are not descriptive of to what the genders are
    - Weather Information: While this table is relatively in good shape, having units specified in the columns headers would be good practice

Creating Additional Graphs

- While the project never stated this, I easily could have created easy-to-read graphs based on a desired visualization of data. The steps I would go to achieve this are:
    - Do any necessary data aggregation needed to create such a graph
    - By using searborn and matplotlib, creat faceted graphs to easily display what data need to be visualized

Done Additional Data Aggregation

- Additional data aggregation such a daily, weekly, monthly averages, totals etc. could have been created to get a more in-depth breakdown. This could be achieved by creating additional SQL views or using functions within python (which could later be exported to a CSV as well)