Informatics Institute of Technology

In affiliated with

Robert Gorden University Aberdeen

Artificial Intelligence & Data Science

Data Engineering

Coursework

Name: Gouri Medhavi Napevithanage

Student ID(IIT): 20210808

Student ID(RGU): 2237943

Module ID: CM2606

Module Coordinator

Mr. Mohomad Ayoob

Submitted partially fulfilling the requirements for BSc (Hons) in Artificial Intelligence and Data Science degree at Robert Gorden University.

April 2024

Contents

Introduction

HCHO is a dangerous air pollutant that can cause respiratory irritation, eye pain, and has the potential to aggravate asthma, other lung illnesses and cancer. Understanding geographical and temporal patterns of HCHO is important for assessing air quality and public health hazards across regions.

This coursework taught practical data engineering skills and contributes to environmental and public health research. Analyzing HCHO data from prominent Sri Lankan cities can help address air quality issues and promote sustainable development methods.

Satellite observations of formaldehyde (HCHO) are important for investigating air quality and climate at different scales. This gas is essential for breaking down contaminants, influencing ozone production and, atmospheric chemistry, which affects climate processes. HCHO is primarily caused by human activity such as grass burning, vehicles, and factories, while some may occur from methane breakdown in distant places. HCHO levels are also influenced by seasonal fluctuations and fires, Due to its short lifespan, it serves as a reliable indicator of hydrocarbon emissions.

Understanding its sources and dynamics is critical for developing climate models and mitigation methods. HCHO exposure data supports public health actions and awareness efforts to protect sensitive groups.

Assessment Objectives

This assessment primarily aided in using data engineering skills to analyze HCHO data in Sri Lanka, as well as identifying the value of HCHO monitoring for air quality and climate research. This assessment provided insight into potential HCHO sources and spatiotemporal trends. It was additionally tasked with developing time-series prediction methods that used machine learning models.

Dataset Selection

The given dataset included daily HCHO observations from the European Space Agency's Sentinel-5P satellite for seven Sri Lankan cities.
Historical data is available from January 1, 2019, to December 31, 2023. The cities include Colombo Proper, Deniyaya Matara, Nuwara Eliya Proper, Bibile Monaragala, Kurunegala Proper, Jaffna Proper, and Kandy Proper.

The dataset link:
https://drive.google.com/drive/folders/1xzQ5pIEnaUN2DOyZTqYSJrxFMC8Unx73?usp=sharing
Weather dataset link:
https://www.kaggle.com/datasets/rasulmah/sri-lanka-weather-dataset

Each row of the dataset contained four columns: HCHO reading, location, current date and next date. Attached below displays a unfiltered sample of the dataset.

HCHO reading	Location	Current Date	Next Date
1.9698344E-4	Colombo Proper	2019-01-01	2019-01-02
2.625522E-4	Colombo Proper	2019-01-02	2019-01-03
9.852119E-5	Colombo Proper	2019-01-03	2019-01-04
2.0993206E-4	Colombo Proper	2019-01-04	2019-01-05
1.7853372E-4	Colombo Proper	2019-01-05	2019-01-06
1.0822967E-4	Colombo Proper	2019-01-06	2019-01-07
3.9268294E-4	Colombo Proper	2019-01-07	2019-01-08
9.153156E-5	Colombo Proper	2019-01-08	2019-01-09
1.205979E-4	Colombo Proper	2019-01-09	2019-01-10
1.2977235E-4	Colombo Proper	2019-01-10	2019-01-11
2.2391882E-4	Colombo Proper	2019-01-11	2019-01-12
1.5694181E-4	Colombo Proper	2019-01-12	2019-01-13
NULL	Colombo Proper	2019-01-13	2019-01-14
1.3362919E-4	Colombo Proper	2019-01-14	2019-01-15
6.374418E-5	Colombo Proper	2019-01-15	2019-01-16
1.18106225E-4	Colombo Proper	2019-01-16	2019-01-17
2.4725552E-4	Colombo Proper	2019-01-17	2019-01-18
3.6675254E-5	Colombo Proper	2019-01-18	2019-01-19
4.057501E-4	Colombo Proper	2019-01-19	2019-01-20
1.6878563E-4	Colombo Proper	2019-01-20	2019-01-21

only showing top 20 rows

As shown in the image above, the dataset contained missing values as well as other data inconsistencies.

The dataset was divided into 3 individual files containing different cities.

The first csv file (. col_mat_nuw_output.csv) contained data about Colombo proper, Nuwara Eliya proper and Deniyaya Matara.

Second csv file(mon_kur_jaf_output.csv) contained data about Bible Monaragala, Kurunegala proper and Jaffna proper.

The third csv file (kan_output.csv) carried data about Kandy proper.

Technical Implementation

Google collab was selected to write and execute python code for the assessment. Open Java Development kit was installed alongside pyspark API.

```
[1]  install JDK
     import os       #importing os to set environment variable
     def install_java():
       !apt-get install -y openjdk-8-jdk-headless -qq > /dev/null      #install openjdk
       os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"     #set environment variable
       !java -version       #check java version

     #Install pyspark
       ! pip install pyspark==3.5.1
       ! pip install pmdarima
     install_java()
     ! pyspark --version #Check the version
```

Necessary libraries were imported. sparksession, matplotlib.pyplot, sklearn and pandas were mainly used to complete the task at hand.

```
import pyspark
import pyspark.sql
from pyspark.sql import SparkSession
from pyspark import SparkContext, SparkConf
from  pyspark.sql.functions import input_file_name
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql.functions import isnan, when, count, col, udf, lower, regexp_replace, unix_timestamp
from pyspark.sql.types import BooleanType, StructType, StructField, StringType, FloatType, DateType
from pyspark.sql import functions as F
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
from functools import reduce

from sklearn.model_selection import train_test_split
from pmdarima import auto_arima
from statsmodels.tsa.arima.model import ARIMA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

The sparksession was created using the builder pattern. It is the entry point to programming spark with the Dataset and DataFrame API. Data Frames were created using the spark session. The spark UI port is set to 4050 avoid conflicts with the default spark UI port number (4040).

```
# create the spark session
conf = SparkConf().set("spark.ui.port", "4050") #Set the UI port to 4050 for the Spark session

#Create a SparkContext with the above configuration
sc = pyspark.SparkContext(conf=conf)
spark = SparkSession.builder.getOrCreate() #To create dataframes and perform SQL operations
```

Data Preprocessing

A sparksession was initialized with the application name ‘HCHO data analyses. A schema was then defined to format the data when reading it.

Three CSV files were read into separate DataFrames using the defined schema. The header was not included in these DataFrames. The three DataFrames were then combined into one DataFrame by performing union operations.

A list of string value columns was defined to remove all white spaces and was converted to lower case to handle inconsistencies.

An image of the code is attached below.

```
spark = SparkSession.builder.appName('HCHO data analysis').getOrCreate()# Initialize SparkSession with the application name

# Define the schema
schema = StructType([
    StructField("HCHO reading", FloatType(), True),
    StructField("Location", StringType(), True),
    StructField("Current Date", DateType(), True),
    StructField("Next Date", DateType(), True)]) #This format the data to a specific structure when reading data

#Loading Data
df1 = spark.read.csv('data/col_mat_nuw_output.csv', header=False, schema=schema) #add the first cv to a df without header row
df2 = spark.read.csv('data/kan_output.csv', header=False, schema=schema) #df of the 2nd csv
df3 = spark.read.csv('data/mon_kur_jaf_output.csv', header=False, schema=schema)

df = df1.union(df2).union(df3)# Join the dataframes into one by performing union operations
columns =["location"] #Define the string value column

#Handling Inconsistencies

for column in columns:
    df = df.withColumn(column, lower(col(column))) #Convert all string values in the DataFrame to lower case

for column in columns:
    df = df.withColumn(column, regexp_replace(col(column), "\s", "_")) #Remove all white spaces from string values in the DataFrame


df.show()# Display the first 20 rows of the DataFrame

num_entries = df.count() # Count the number of rows in the DataFrame

print(f"Number of entries in the DataFrame: {num_entries}")
```

Result of the inconsistencies handling:

```
+-------------+--------------+------------+----------+
| HCHO reading|      location|Current Date| Next Date|
+-------------+--------------+------------+----------+
| 1.9698344E-4|colombo_proper|  2019-01-01|2019-01-02|
|  2.625522E-4|colombo_proper|  2019-01-02|2019-01-03|
|  9.852119E-5|colombo_proper|  2019-01-03|2019-01-04|
| 2.0993206E-4|colombo_proper|  2019-01-04|2019-01-05|
| 1.7853372E-4|colombo_proper|  2019-01-05|2019-01-06|
| 1.0822967E-4|colombo_proper|  2019-01-06|2019-01-07|
| 3.9268294E-4|colombo_proper|  2019-01-07|2019-01-08|
|  9.153156E-5|colombo_proper|  2019-01-08|2019-01-09|
|  1.205979E-4|colombo_proper|  2019-01-09|2019-01-10|
| 1.2977235E-4|colombo_proper|  2019-01-10|2019-01-11|
| 2.2391882E-4|colombo_proper|  2019-01-11|2019-01-12|
| 1.5694181E-4|colombo_proper|  2019-01-12|2019-01-13|
|         NULL|colombo_proper|  2019-01-13|2019-01-14|
| 1.3362919E-4|colombo_proper|  2019-01-14|2019-01-15|
|  6.374418E-5|colombo_proper|  2019-01-15|2019-01-16|
|1.18106225E-4|colombo_proper|  2019-01-16|2019-01-17|
| 2.4725552E-4|colombo_proper|  2019-01-17|2019-01-18|
| 3.6675254E-5|colombo_proper|  2019-01-18|2019-01-19|
|  4.057501E-4|colombo_proper|  2019-01-19|2019-01-20|
| 1.6878563E-4|colombo_proper|  2019-01-20|2019-01-21|
+-------------+--------------+------------+----------+
only showing top 20 rows

Number of entries in the DataFrame: 12782
```

Then, The DataFrame was initially checked for null values in all columns. This was done by selecting and counting the number of null values in each column, displaying, and then dropping the rows with null values. Then the number of rows dropped was calculated and displayed. Finally, the DataFrame was checked again for null values in all columns to confirm that all the null values had been successfully removed. The code snippet and the results are shown below.

```
df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show() # Check for null values

num_rows_before = df.count()# number of rows before dropping null values

df = df.na.drop()# Drop rows with null values

num_rows_after = df.count()# number of rows after dropping null values

num_rows_dropped = num_rows_before - num_rows_after # number of rows dropped

print(f"Number of rows before dropping null values: {num_rows_before}")
print(f"Number of rows after dropping null values: {num_rows_after}")
print(f"Number of rows dropped: {num_rows_dropped}")

df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).show() #Check again for null values
```

Before handling null values:

```
+------------+--------+------------+---------+
|HCHO reading|location|Current Date|Next Date|
+------------+--------+------------+---------+
|        4864|       0|           0|        0|
+------------+--------+------------+---------+
```

After handling null values:

```
Number of rows before dropping null values: 12782
Number of rows after dropping null values: 7918
Number of rows dropped: 4864
+------------+--------+------------+---------+
|HCHO reading|location|Current Date|Next Date|
+------------+--------+------------+---------+
|           0|       0|           0|        0|
+------------+--------+------------+---------+
```

The DataFrame was checked for duplicate rows across all columns. This was done by comparing the total number of rows in the DataFrame with the number of rows in the DataFrame after dropping duplicates. If the total number of rows was greater than the number of rows after dropping duplicates, it was concluded that the DataFrame had duplicates. In this case the result was shown otherwise. The image of the code snippet is attached below.

```
# Check for duplicated rows across all columns
if df.count() > df.dropDuplicates().count():
    print("Data has duplicates")
else:
    print("No duplicates found")
```

The result:

```
No duplicates found
```

Then the column “HCHO reading” was selected to remove the outliers. A dictionary was created to store the first quartile(q1) and the third quartile(q3) and calculated. The interquartile range (IQR) was calculated as the difference between q1 and q3. The lower and the upper bounds for outliers were calculated using the IQR. Any value less than the lower bound or greater than the upper bound was considered an outlier. A new column was added to the DataFrame to indicate whether each row was an outlier (true) or not (false). All rows that were outliers were removed from the DataFrame. Then the number of outliers were counted before and after removing them. This confirmed that all outliers had been successfully removed. The code snippet is attached below.

```
column = "HCHO reading"# Select the column to remove the outliers

bounds = {
    c: dict(zip(["q1", "q3"], df.approxQuantile(c, [0.25, 0.75], 0))) #Create a dictionary to store quantile values
    for c in [column]}

iqr = bounds[column]['q3'] - bounds[column]['q1'] #Calculation of interquartile range (IQR)
lower_bound = bounds[column]['q1'] - (iqr * 1.5) #Calculation of upper bound for outliers
upper_bound = bounds[column]['q3'] + (iqr * 1.5) #calculation of lower bound

def is_outlier(value):# Define a function to check if a value is an outlier based on calculated bounds
    return (value < lower_bound) | (value > upper_bound)
is_outlier_udf = udf(is_outlier, BooleanType())

df = df.withColumn("outlier", is_outlier_udf(col(column)))# Returns True for outliers and False for non-outliers

outliers_count = df.filter(col("outlier") == True).count() # Count outliers before removing

print("Number of outliers before cleaning: ", outliers_count)

df0 = df.filter(col("outlier") == False)# Select the rows where 'outlier' column is false
outliers_count_after = df0.filter(col("outlier") == True).count()# Count outliers again

print("Number of outliers after cleaning: ", outliers_count_after)
```

```
Number of outliers before cleaning:  215
Number of outliers after cleaning:  0
```

Then the DataFrame was grouped by the 'Location" column to calculate the unique mean, median and the standard deviation values for each location.

The results are attached below alongside the code snippet.

```
# Group the df by 'location' and calculate mean, median, and standard deviation
df0_grouped = df0.groupBy("Location").agg(
    F.mean("HCHO reading").alias("mean"),
    F.expr('percentile_approx(`HCHO reading`, 0.5)').alias("median"),
    F.stddev("HCHO reading").alias("stddev")
)
df0_grouped.show()
```

```
+------------------+--------------------+------------+--------------------+
|          Location|                mean|      median|              stddev|
+------------------+--------------------+------------+--------------------+
|nuwara_eliya_proper|8.799096372229397E-5| 8.174215E-5|7.894000945972918E-5|
|     colombo_proper|1.485576533640281...|1.4571268E-4|8.388822521361778E-5|
|   deniyaya,_matara|8.818049407657948E-5| 8.092664E-5|7.584945309894013E-5|
|       kandy_proper|1.066285936246885...| 1.060346E-4|8.238293162656169E-5|
|      jaffna_proper|1.061667994349809...| 1.009731E-4|6.603454815760093E-5|
| bibile,_monaragala|1.249318856210353...|1.2424483E-4| 8.14183935102564E-5|
|  kurunegala_proper|1.306639412104597E-4|1.2470146E-4| 7.60784649941602E-5|
+------------------+--------------------+------------+--------------------+
```

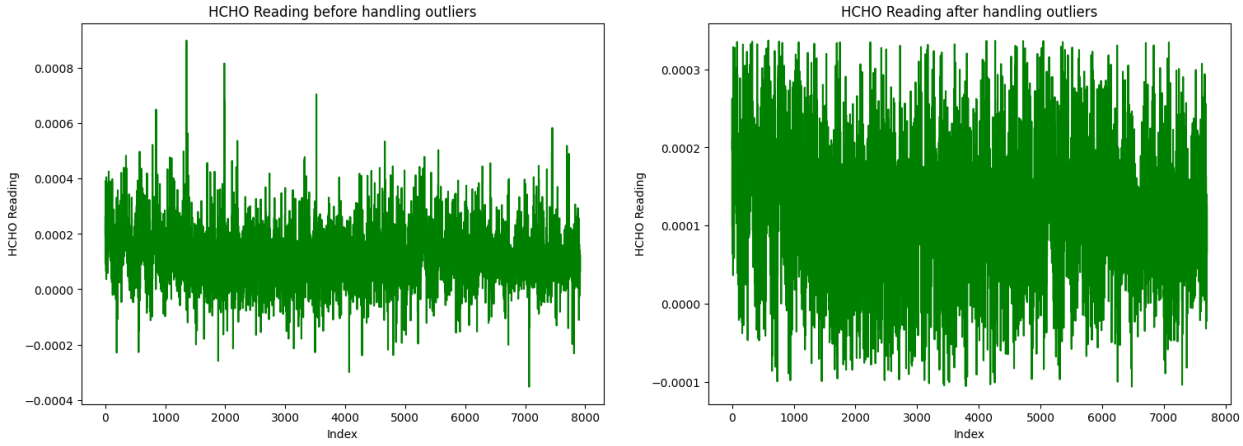
This calculation helps to summarize the central tendency and variability of the data for each location. Based on the results, Nuwara Eliya proper has the highest mean and median HCHO reading among all the locations. The standard deviation is also high indicating a wider spread of values around the mean while Kurunegala proper has the lowest average reading and the least variability. The other locations fall in between these extremes.

Data Visualization

First the DataFrame was converted to a pandas dataframe for easier polloting. HCHO reading before and after handling outlier is plotted using matplotlib.

The graph on the left, titled "HCHO reading before handling outliers" shows a wide range of fluctuations in the data points with extreme spikes indicating that there were some unusually high or low values in the dataset. However, the graph on the right appears more consistent in its

range of data points with fewer extreme spikes. This proves that the outlier data has been addressed, resulting in a more uniformed distribution of readingxs.

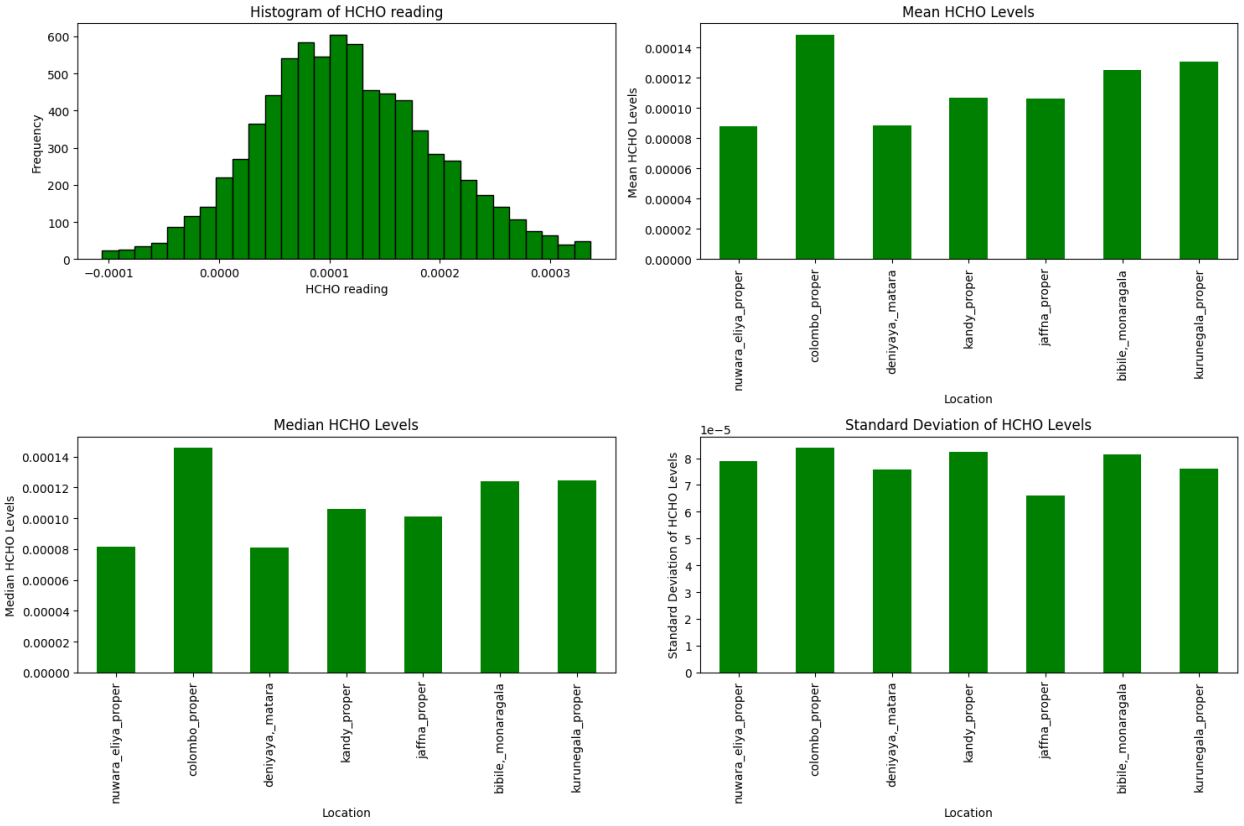


Comparing the two graphs above, it is evident that the process of handling outliers has reduced the variability in the readings which led to more accurate and reliable data analysis.

The mean HCHO level, Median HCHO levels and standard deviation calculations were visualized and displayed alongside the histogram of HCHO readings.

- **Histogram of HCHO Reading:** shows a bell-shaped distribution, indicating normal distribution. This suggests that most of the readings are near to the mean, with fewer values further out.
- **Mean HCHO Levels:** Mean HCHO levels vary by location, as indicated by the heights of the bars. It shows that 'nuwara eliya proper' has the greatest mean HCHO level, whilst 'kurunegala proper' has the lowest.
- **Median HCHO Levels:** • This graph shows the median HCHO values, which are the middle value in each location's dataset. The median is the middle value in a set of data when it is ordered. It measures central tendency and is unaffected by outliers or extreme values. Like the mean HCHO levels, 'nuwara eliya proper' has the greatest median HCHO level, while 'kurunegala proper' has the lowest.
- **Standard Deviation of HCHO:** The final graph depicts the standard deviation of HCHO levels by location, which shows how much individual measurements deviate from the mean value. Taller bars indicate greater variability in readings. 'Colombo proper' has the

highest standard deviation, indicating a wide range of HCHO levels, and 'Jaffna proper' has the lowest standard deviation.



These graphs provide a comprehensive view of the HCHO levels. They are useful for understanding the distribution and consistency of HCHO readings in environmental monitoring and health studies.

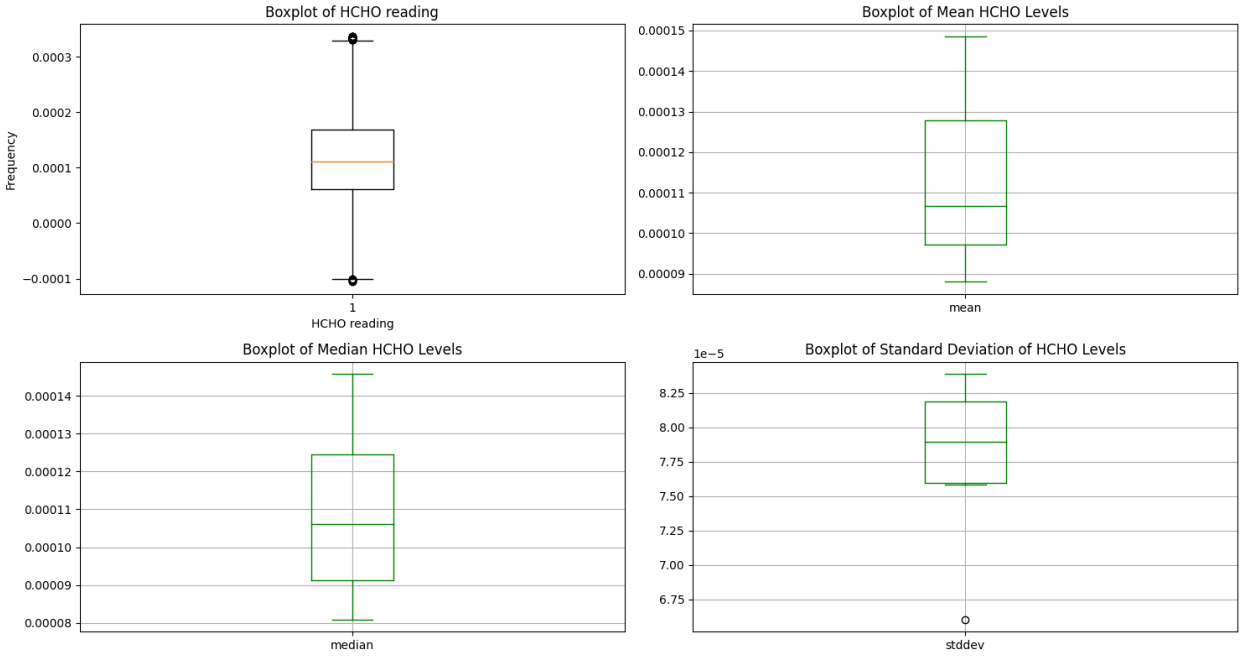
Boxplots of the three aggregate functions were created for display.

Boxplot for HCHO Reading: This graphic depicts the distribution of HCHO readings. The box indicates the interquartile range (IQR), while the line inside it denotes the median. The 'whiskers' stretch to the farthest points that are not regarded as outliers, while the dots denote outlier readings.

Boxplot of mean HCHO levels: This graphic depicts the average HCHO levels at various places. The line inside the box represents the median of these means, while the IQR depicts the range of the mean values. Again, dots are used to symbolize outliers.

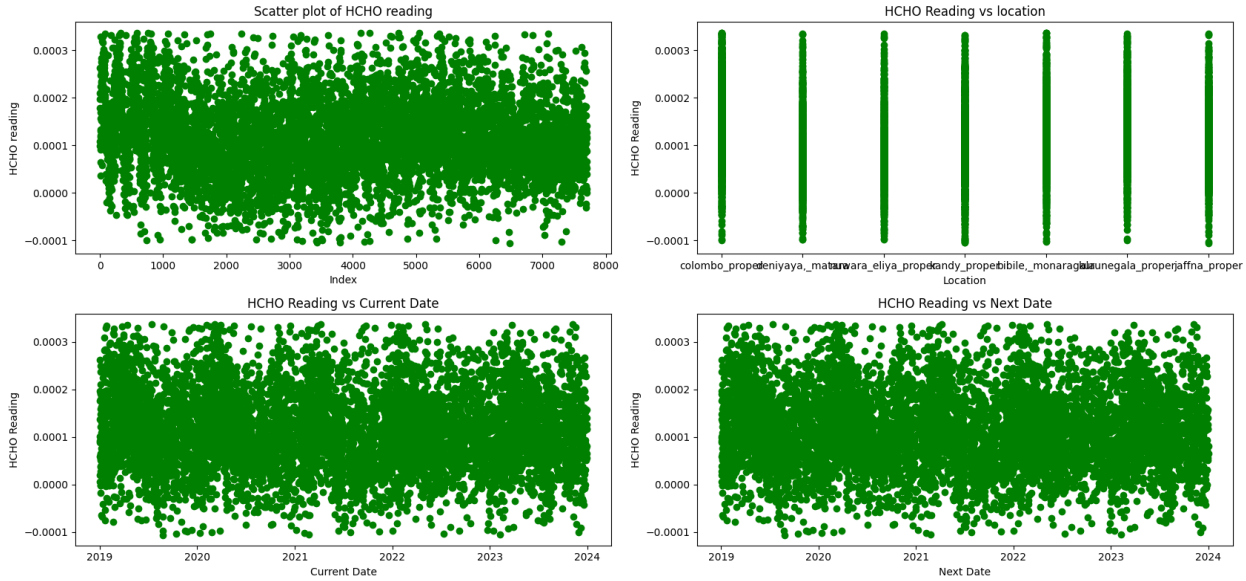
Boxplot of Median HCHO Levels: Like the mean levels, this plot depicts the median HCHO levels. The median of medians is the line inside the box, and the IQR illustrates how much the median values differ between locations.

Boxplot of the standard deviation of HCHO levels: This graphic depicts the variability of HCHO measurements at each site as determined by standard deviation. A higher standard deviation suggests greater variability in HCHO values.



These boxplots are useful for comparing the central tendency and variability of HCHO readings across sites, as well as finding any areas with very high or low readings or variability. They present a visual representation of the data's distribution, central tendency, and dispersion.

Scatter plots for HCHO readings scatter plot, HCHO readings by location, HCHO readings by current date, HCHO readings by next date were created, each providing a visual representation of data points for HCHO readings and their relationship with other factors.



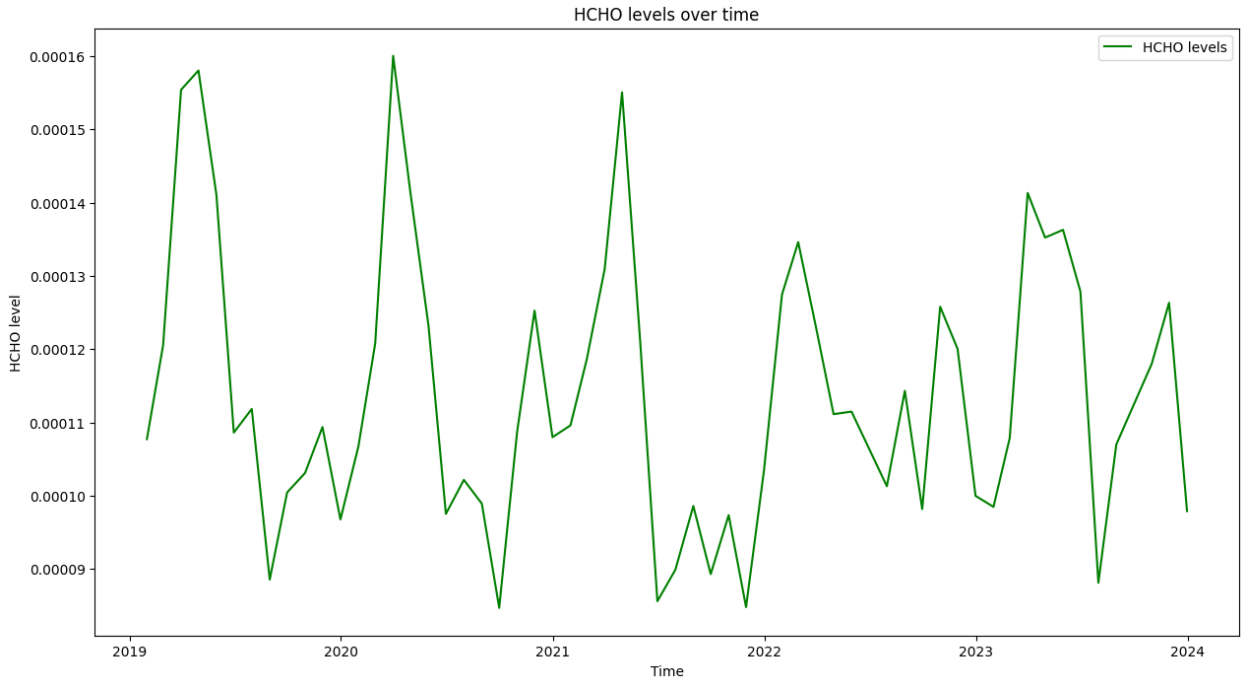
These scatterplots are useful for identifying patterns and trends in the data. They help in understanding how HCHO readings vary by other factors over time, which is crucial for environmental monitoring and analysis.

Spatio-Temporal Analysis

The Spark DataFrame was transformed to a Pandas DataFrame for more convenient use. The 'Current Date' column was converted to datetime format and used as the DataFrame's index. The data was then resampled monthly by computing the average HCHO measurement for each month. This resampled data was graphed as a time series, with HCHO levels on the y-axis and time on the x-axis. The resulting plot depicted how HCHO levels fluctuated over time, which is important for environmental monitoring and analysis. An image of the code and the graph is displayed below.

```
df = df0.toPandas()#Convert the spark dataframe to pandas dataframe
df['Current Date'] = pd.to_datetime(df['Current Date'])# Convert the date column to datetime
df.set_index('Current Date', inplace=True)# Set the date column as the index
monthly_df = df['HCHO reading'].resample('M').mean()# Resample the data to monthly frequency

plt.figure(figsize=(15,8))
plt.plot(monthly_df, label='HCHO levels',color='green')
plt.title('HCHO levels over time')
plt.xlabel('Time')
plt.ylabel('HCHO level')
plt.legend(loc='best')
plt.show()
```

A line graph is created to represent the HCHO levels overtime, spanning from 2019 to 2024.



The graph depicts changes in HCHO concentration levels, with peaks and troughs representing different amounts of HCHO in the environment. The biggest peaks come in early 2020, late 2021, and mid-2023, indicating periods of elevated HCHO levels. In contrast, there is a major trough in late 2022, where the level lowers considerably.

The locations were initially classified and divided into high and low-altitude categories. A dictionary was then built to associate each place with its altitude categorization. A function was

created to classify a given location using this dictionary, and it was transformed into a user-defined function (UDF) for usage with Spark DataFrames. The UDF was applied to the DataFrame df0's 'location' column to generate a new 'altitude' column, with each entry classified as 'High Altitude', 'Low Altitude', or 'Unknown'. The revised DataFrame was then presented.

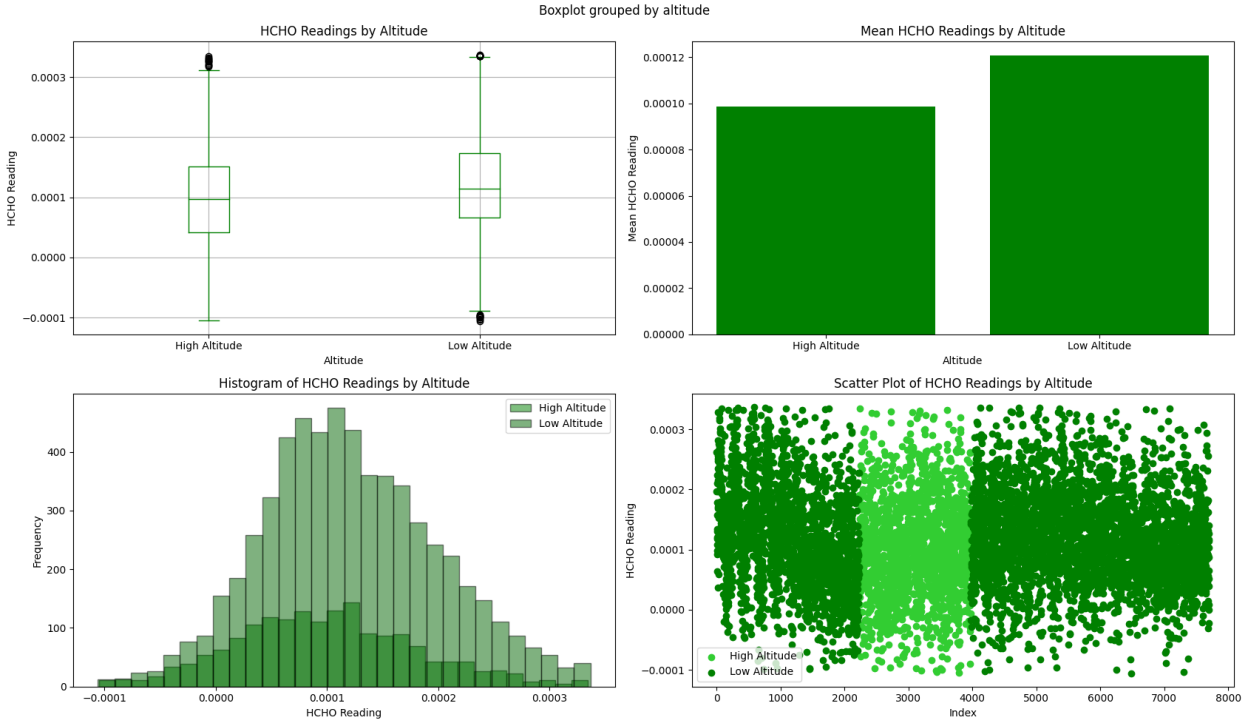
The code snippet:

```
# Define the locations
locations = ['colombo_proper', 'deniyaya,_matara', 'bibile,_monaragala', 'kurunegala_proper', 'jaffna_proper','nuwara_eliya_proper', 'kandy_proper']
high_altitude_locations = ['nuwara_eliya_proper', 'kandy_proper']
low_altitude_locations = ['colombo_proper', 'deniyaya,_matara', 'bibile,_monaragala', 'kurunegala_proper', 'jaffna_proper']

# Define dictionary where the keys are locations and the values are their classifications
location_classification = {location: 'High Altitude' for location in high_altitude_locations}
location_classification.update({location: 'Low Altitude' for location in low_altitude_locations})

# Define the classification function
def classify_location(location):
    return location_classification.get(location, 'Unknown')

# UDF from the classification function
classify_location_udf = udf(classify_location, StringType())

# new 'altitude' column to the DataFrame using the UDF
updated_df = df0.withColumn('altitude', classify_location_udf(df0['location']))

updated_df.show()
```

The result:

```
+-------------+--------------+------------+----------+-------+------------+
| HCHO reading|      location|Current Date| Next Date|outlier|    altitude|
+-------------+--------------+------------+----------+-------+------------+
| 1.9698344E-4|colombo_proper|  2019-01-01|2019-01-02|  false|Low Altitude|
|  2.625522E-4|colombo_proper|  2019-01-02|2019-01-03|  false|Low Altitude|
|  9.852119E-5|colombo_proper|  2019-01-03|2019-01-04|  false|Low Altitude|
| 2.0993206E-4|colombo_proper|  2019-01-04|2019-01-05|  false|Low Altitude|
| 1.7853372E-4|colombo_proper|  2019-01-05|2019-01-06|  false|Low Altitude|
| 1.0822967E-4|colombo_proper|  2019-01-06|2019-01-07|  false|Low Altitude|
|  9.153156E-5|colombo_proper|  2019-01-08|2019-01-09|  false|Low Altitude|
|  1.205979E-4|colombo_proper|  2019-01-09|2019-01-10|  false|Low Altitude|
| 1.2977235E-4|colombo_proper|  2019-01-10|2019-01-11|  false|Low Altitude|
| 2.2391882E-4|colombo_proper|  2019-01-11|2019-01-12|  false|Low Altitude|
| 1.5694181E-4|colombo_proper|  2019-01-12|2019-01-13|  false|Low Altitude|
| 1.3362919E-4|colombo_proper|  2019-01-14|2019-01-15|  false|Low Altitude|
|  6.374418E-5|colombo_proper|  2019-01-15|2019-01-16|  false|Low Altitude|
|1.18106225E-4|colombo_proper|  2019-01-16|2019-01-17|  false|Low Altitude|
| 2.4725552E-4|colombo_proper|  2019-01-17|2019-01-18|  false|Low Altitude|
| 3.6675254E-5|colombo_proper|  2019-01-18|2019-01-19|  false|Low Altitude|
| 1.6878563E-4|colombo_proper|  2019-01-20|2019-01-21|  false|Low Altitude|
| 3.2829228E-4|colombo_proper|  2019-01-22|2019-01-23|  false|Low Altitude|
| 2.1477981E-4|colombo_proper|  2019-01-23|2019-01-24|  false|Low Altitude|
| 2.3472452E-4|colombo_proper|  2019-01-24|2019-01-25|  false|Low Altitude|
+-------------+--------------+------------+----------+-------+------------+
only showing top 20 rows
```

The Spark DataFrame was first transformed into a Pandas DataFrame. A figure containing a grid of subplots was then generated. A boxplot was created to illustrate the distribution of HCHO measurements by altitude. A bar plot was produced to show the average HCHO values by altitude. Histograms were created to depict the distribution of HCHO measurements at high and low elevations. A scatter plot was also produced to visually represent HCHO measurements by altitude. Each plot was well labeled, and the figure was exhibited. This visualization offered a thorough picture of HCHO values at various elevations.



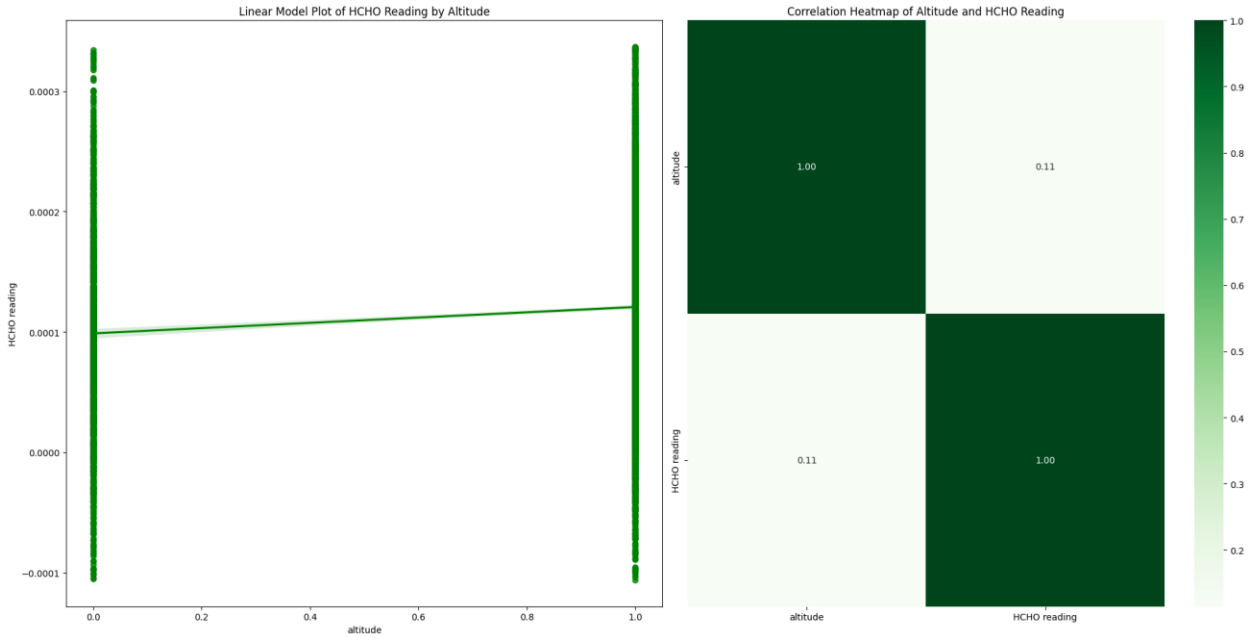
Overall, the graphs indicate that the HCHO readings vary with altitude.

Then the relationship between altitude and HCHO readings were analyzed. The altitude column was converted to a categorical type and codes were assigned to each category. The correlation between altitude and HCHO reading was calculated, printed, and plotted. The code snippet is attached below.

```
updated_df_pd = updated_df.toPandas()

updated_df_pd['altitude'] = updated_df_pd['altitude'].astype('category')# Convert the 'altitude' column to a categorical type
updated_df_pd['altitude'] = updated_df_pd['altitude'].cat.codes# Assign codes to the categories of the 'altitude' column
correlation = updated_df_pd['altitude'].corr(updated_df_pd['HCHO reading'])# Calculate the correlation between 'altitude' and 'HCHO reading'
print(f"The correlation between altitude and HCHO reading is {correlation}")
#plot
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
#Ragplot (lmplot doesn't support ax parameter directly)
sns.regplot(x='altitude', y='HCHO reading', data=updated_df_pd, color='green', ax=ax[0])
ax[0].set_title('Linear Model Plot of HCHO Reading by Altitude')
#Correlation matrix
corr_matrix = updated_df_pd[['altitude', 'HCHO reading']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='Greens', fmt=".2f", ax=ax[1])
ax[1].set_title('Correlation Heatmap of Altitude and HCHO Reading')

plt.tight_layout()
plt.show()
```

The first plot displays a linear model plot of HCHO Reading by Altitude using seaborn's regplot function. It suggests that the relationship between altitude and HCHO readings is not linear.

The second plot presented a correlation heatmap of Altitude and HCHO Reading, which was created using a correlation matrix of altitude and HCHO reading. It displays a strong negative correlation between altitude and HCHO reading. Indicating that as the altitude increases, HCHO reading tends to decrease.



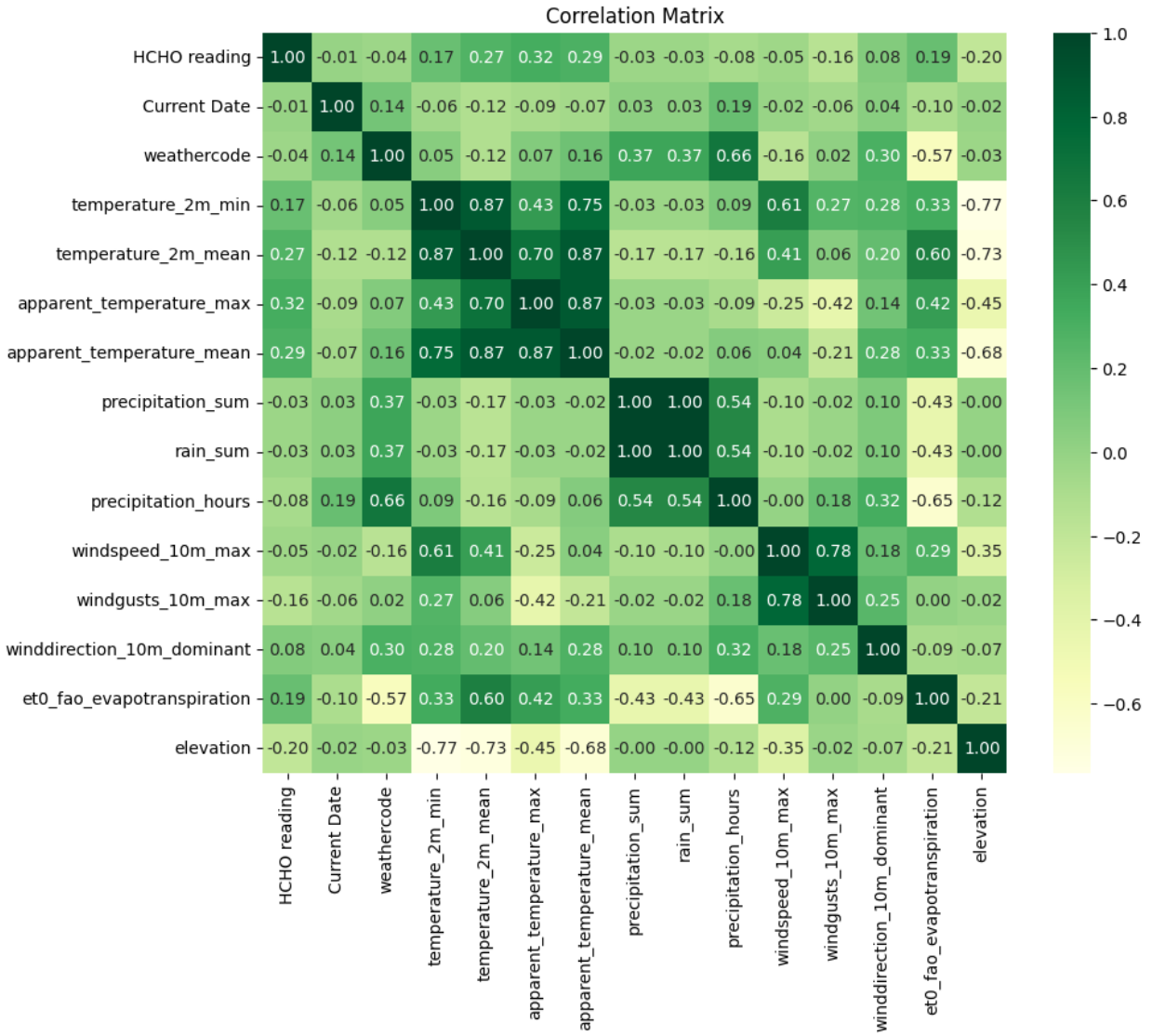
Then the weather data was analyzed from various locations in Sri lanka. First, the data CSV file was loaded, and the unnecessary columns were dropped. For each location, the relevant data was filtered from the main DataFrame and combined with the corresponding weather data. String

columns were dropped to make visualization of correlation matrix heatmap easier. Current date column was converted to Unix timestamp for easier data manipulation. The combined data was then to a pandas Dataframe and the correlation matrix was calculated and visualized.

The code snippet is attached below.

```
# Define the locations
locations = ['colombo_proper', 'deniyaya,_matara', 'bibile,_monaragala', 'kurunegala_proper', 'jaffna_proper','nuwara_eliya_proper', 'kandy_proper']
whether_locations=['Colombo', 'Kandy', 'Jaffna', 'Matara', 'Kurunegala']

# Read the weather data
whether_df = spark.read.csv('data/SriLanka_Weather_Dataset.csv', header=True, inferSchema=True)
whether_df = whether_df.drop('whethercode', 'sunrise', 'sunset', 'snowfall_sum', 'country', 'temperature_2m_max','apparent_temperature_min', 'shortwave_radiation_sum', 'precipitation_sum''et0_

# Initialize an empty list to store the joined DataFrames
joined_dfs = []

for loc, wloc in zip(locations, whether_locations):
    # Filter the DataFrames based on the location
    df_loc = updated_df.filter(updated_df.location == loc)
    whether_df_loc = whether_df.filter(whether_df.city == wloc)

    # Join the DataFrames and drop missing values
    joined_df = df_loc.join(whether_df_loc, df_loc['Current Date'] == whether_df_loc.time, 'outer').dropna()

    # Add the joined DataFrame to the list
    joined_dfs.append(joined_df)

# Combine all the joined DataFrames
combined_df = reduce(lambda a, b: a.union(b), joined_dfs)
ml_df = combined_df.alias('ml_df')


# Convert 'Current Date' to Unix timestamp (number of seconds since 1970-01-01 00:00:00)
combined_df = combined_df.withColumn('Current Date', unix_timestamp('Current Date'))

# Drop the string column
dropped_df = combined_df.drop('location','Next Date','outlier','time','city','altitude')

# Convert the combined DataFrame to pandas and calculate the correlation matrix
correlation_matrix = dropped_df.toPandas().corr()

# Plot the correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='YlGn')
plt.title("Correlation Matrix")
plt.show()
```

The correlation matrix is a visual representation of the strength and direction of relationships between different variables in the DataFrame.Each cell shows the correlation coefficient between two variables. Yellow represents the weaker correlation and green represents the stronger correlation. The diagonal cells compare a variable to itself always show a correlation of 1.



Machine Learning Model

Initially, the dataframe removes extraneous columns from the DataFrame before converting it to a Pandas DataFrame. It then finds unique places and creates lists to contain Mean Squared Error (MSE) and R-squared (R2) values. The code filters the DataFrame by location, specifies the dependent and independent variables, and finds external features. It divides the data into training and testing sets and applies an ARIMA model to the historical data and a Random Forest model to the external features. The code then makes predictions using both models, calculating the

MSE and R2 values for the combined forecasts. Finally, it compares the actual and projected HCHO measurements, as well as the MSE and R2 scores for each location. This approach aids in understanding HCHO levels' temporal and spatial trends, measuring air quality, identifying emission sources, and monitoring the efficacy of air quality restrictions.

- The ARIMA Model is a common statistical tool for time series forecasting. ARIMA models use previous values to predict future values. They specifically account for trends, seasonality, and noise in the data. In this code, the ARIMA model is used to represent historical HCHO readings, which are time-series data.
- Random Forest is a machine learning method that creates numerous decision trees and predicts the mode or mean of each tree for classification or regression purposes. Random Forest can handle multiple features and is less prone to overfitting than decision trees. In this code, the Random Forest model is used to simulate the link between HCHO values and different environmental factors such as weather and elevation.

The interactive dashboard.

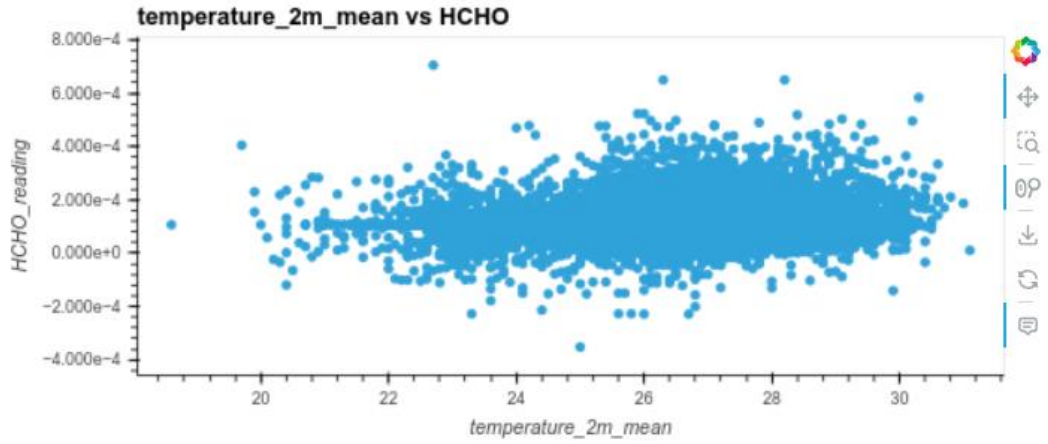
An interactive dashboard is constructed to visualize formaldehyde (HCHO) readings from various locations and features. The dashboard has two dropdown widgets where users can choose a feature (temperature, precipitation, wind speed, or date) and a location (Colombo Proper, Kandy Proper, Jaffna Proper, or Kurunegala Proper). Depending on the selected feature and location, the dashboard shows scatter plots of HCHO measurements versus the specified feature for all or a specific place. This visualization aids in understanding the relationship between HCHO values and other environmental parameters, as well as spotting patterns and trends in air quality over time and across locales. An image of the dashboard is attached below.

HCHO Readings Dashboard

This dashboard shows how HCHO readings vary across different features.

Select Feature

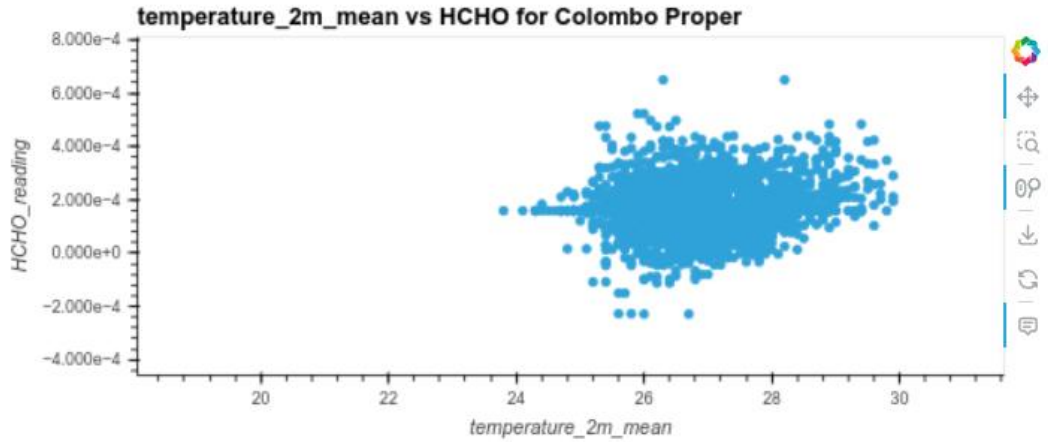
temperature_2m_mean



Location specific data

Select Location

Colombo Proper



Potential Limitations

Data quality: The quality of original data impacts the analysis. handling inconsistencies and missing values have an impact on the results.

Sampling bias: If the data is not representative of the population is collected from, it can lead to biased results.

Model Assumptions: The analysis assumes a linear relationship between altitude and HCHO readings as shown in the scatter plot with a regression line. However, the relationship could be non-linear.

Outlier handling: The method used for detecting and handling outliers may have impact on the results.

Categorical coding: The 'altitude' variable was coded as '0' for low altitude and '1' for high altitude. This binary coding might oversimplify the relationship between altitude and HCHO readings, as altitude is a continuous variable.

The correlation observed between variables does not imply causation and there could be cofounding factors that were not accounted for in the analysis and the conversion of data types, The statistical methods used to calculate correlations data have their own assumptions and limitations.

Therefore, while the analysis provides valuable information, the result should be interpreted with these potential limitations.

Influence on further research and policymaking.

These findings from analysis can be instrumental in both policy making and future research.

If high HCHO levels are persistently connected with specific areas, notably high altitudes, regulators may investigate the origins of these emissions. This could result in stronger limitations on activities that contribute to elevated HCHO levels in these locations.

Understanding how HCHO is distributed can help to establish public health guidelines. For

example, if some places have high HCHO levels, those with respiratory disorders may be warned to avoid or minimize their exposure in those areas.
The findings could influence urban planning decisions, such as the location of industrial districts, parks, and residential zones, to reduce human exposure to high HCHO levels.

While this analysis provides useful insights into the association between altitude and HCHO levels, more research is required to determine the underlying causes of this relationship. Future research could investigate the function of temperature, pressure, and other altitude-related variables in HCHO production.
Longitudinal research could shed light on how HCHO levels fluctuate over time and the potential long-term consequences of exposure to different levels of HCHO.
Researchers could also look at the health effects of HCHO exposure, particularly in places with high HCHO levels. This could include medical research or public health surveys.

Comparison of the findings

The findings of this analysis, which show a link between altitude and HCHO levels, are consistent with several other studies conducted in other countries. For example, research done in the Three Rivers' Source region above the Tibetan Plateau in China discovered that tropospheric NO2 and HCHO vertical column densities dropped as altitude increased1. This implies that height may influence the concentration of several gases in the atmosphere, not simply HCHO. However, the link between altitude and HCHO levels varies by region. For example, research conducted in eastern China discovered that main air pollutant emissions have decreased, and overall air quality in China has significantly improved in recent years2. Despite this, China's volatile organic compounds (VOC) levels continue to rise due to a lack of VOC mitigation measures2. This suggests that, while altitude can influence HCHO levels, other factors such as local emissions and air quality policies can also have a substantial impact.

In contrast, a study conducted in a rural area did not particularly address the effect of altitude on HCHO levels3. This shows that in some places, other factors may have a greater influence on HCHO levels.

Reference

Spark:

- www.youtube.com. (n.d.). *Getting started with Apache Spark / PySpark setup | ETL with Pyspark*. [online] Available at: https://www.youtube.com/watch?v=5S8fEqTfT2I&ab_channel=BIInsightsInc [Accessed 21 Apr. 2024].

- www.youtube.com. (n.d.). *An Introduction to Big Data Processing using Apache Spark | DataHour by Akshay Chauhan*. [online] Available at: https://www.youtube.com/watch?v=trpyELoJ1sk&ab_channel=AnalyticsVidhya [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Using pyspark in Google Colab*. [online] Available at: https://stackoverflow.com/questions/63323344/using-pyspark-in-google-colab [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *How can I merge these many csv files (around 130,000) using PySpark into one large dataset efficiently?* [online] Available at: https://stackoverflow.com/questions/60261615/how-can-i-merge-these-many-csv-files-around-130-000-using-pyspark-into-one-lar [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *How to read csv without header and name them with names while reading in pyspark?* [online] Available at:

https://stackoverflow.com/questions/44558139/how-to-read-csv-without-header-and-name-them-with-names-while-reading-in-pyspark [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Pyspark: display a spark data frame in a table format*. [online] Available at: https://stackoverflow.com/questions/39067505/pyspark-display-a-spark-data-frame-in-a-table-format [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *How to fix inconsistent schemas in parquet file partition using Spark*. [online] Available at: https://stackoverflow.com/questions/53603400/how-to-fix-inconsistent-schemas-in-parquet-file-partition-using-spark [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Remove blank space from data frame column values in Spark*. [online] Available at: https://stackoverflow.com/questions/35540974/remove-blank-space-from-data-frame-column-values-in-spark [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *count rows in Dataframe Pyspark*. [online] Available at: https://stackoverflow.com/questions/65478043/count-rows-in-dataframe-pyspark [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Show number of rows and columns when all rows are displayed*. [online] Available at: https://stackoverflow.com/questions/54696940/show-number-of-rows-and-columns-when-all-rows-are-displayed [Accessed 21 Apr. 2024]

Null values:

- Stack Overflow. (n.d.). *apache spark - How to find count of Null and Nan values for each column in a PySpark dataframe efficiently?* [online] Available at: https://stackoverflow.com/questions/44627386/how-to-find-count-of-null-and-nan-values-for-each-column-in-a-pyspark-dataframe.

- Stack Overflow. (n.d.). *Concatenate two PySpark dataframes*. [online] Available at: https://stackoverflow.com/questions/37332434/concatenate-two-pyspark-dataframes [Accessed 21 Apr. 2024].

Duplicate values:

- Stack Overflow. (n.d.). *check for duplicates in Pyspark Dataframe*. [online] Available at: https://stackoverflow.com/questions/50122955/check-for-duplicates-in-pyspark-dataframe [Accessed 21 Apr. 2024].

- Zach (2023). *How to Find Duplicates in PySpark DataFrame*. [online] Statology. Available at: https://www.statology.org/pyspark-find-duplicates/ [Accessed 21 Apr. 2024].

- GeeksforGeeks. (2021). *PySpark DataFrame - Drop Rows with NULL or None Values*. [online] Available at: https://www.geeksforgeeks.org/pyspark-dataframe-drop-rows-with-null-or-none-values/.

Pandas dataframe:

- Stack Overflow. (n.d.). *How to scatter pyspark.sql.DataFrame?* [online] Available at: https://stackoverflow.com/questions/76083960/how-to-scatter-pyspark-sql-dataframe [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Best way to count the number of rows with missing values in a pandas DataFrame*. [online] Available at:

https://stackoverflow.com/questions/28199524/best-way-to-count-the-number-of-rows-with-missing-values-in-a-pandas-dataframe [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *python - How do I get the row count of a Pandas DataFrame?* [online] Available at: https://stackoverflow.com/questions/15943769/how-do-i-get-the-row-count-of-a-pandas-dataframe.

- Stack Overflow. (n.d.). *How to count number of rows dropped in a pandas dataframe*. [online] Available at: https://stackoverflow.com/questions/74359778/how-to-count-number-of-rows-dropped-in-a-pandas-dataframe [Accessed 21 Apr. 2024].

Outliers:

- Jagdeesh (2023). *PySpark Outlier Detection and Treatment - A Comprehensive Guide How to handle Outlier in PySpark*. [online] Machine Learning Plus. Available at: https://www.machinelearningplus.com/pyspark/pyspark-outlier-detection-and-treatment/.

Calculations:

- Stack Overflow. (n.d.). *How to calculate mean and standard deviation given a PySpark DataFrame?* [online] Available at: https://stackoverflow.com/questions/47995188/how-to-calculate-mean-and-standard-deviation-given-a-pyspark-dataframe [Accessed 21 Apr. 2024].

- Zach (2023). *How to Calculate Mean of Multiple Columns in PySpark*. [online] Statology. Available at: https://www.statology.org/pyspark-mean-multiple-columns/ [Accessed 21 Apr. 2024].

- spark.apache.org. (n.d.). *pyspark.sql.functions.median — PySpark master documentation*. [online] Available at: https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.median.html [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *How to aggregate median and standard deviation in PySpark?* [online] Available at: https://stackoverflow.com/questions/57154280/how-to-aggregate-median-and-standard-deviation-in-pyspark [Accessed 21 Apr. 2024].

SQL functions:

- spark.apache.org. (n.d.). *pyspark.sql.functions — PySpark 2.1.3 documentation*. [online] Available at: https://spark.apache.org/docs/2.1.3/api/python/_modules/pyspark/sql/functions.html [Accessed 21 Apr. 2024].

ML model:

- Overload, D. (2023). *Time Series Forecasting using ARIMA Models: A Step-by-Step Guide*. [online] Medium. Available at: https://medium.com/@data-overload/time-series-forecasting-using-arima-models-a-step-by-step-guide-90940d61337c.

- www.youtube.com. (n.d.). *How to build ARIMA models in Python for time series forecasting*. [online] Available at: https://www.youtube.com/watch?v=-aCF0_wfVwY&t=760s.

- www.youtube.com. (n.d.). Random Forest Algorithm - Random Forest Explained | Random Forest in Machine Learning | Simplilearn. [online] Available

at:https://www.youtube.com/watch?v=eM4uJ6XGnSM&ab_channel=Simplilearn [Accessed 29 Mar. 2024].

Plotting:

- matplotlib.org. (n.d.). List of named colors — Matplotlib 3.4.2 documentation. [online] Available at: https://matplotlib.org/stable/gallery/color/named_colors.html.
- Stack Overflow. (n.d.). Make the size of a heatmap bigger with seaborn. [online] Available at:https://stackoverflow.com/questions/38913965/make-the-size-of-a-heatmap-bigger-with-seaborn [Accessed 29 Mar. 2024].

- www.w3schools.com. (n.d.). Matplotlib Subplot. [online] Available at: https://www.w3schools.com/python/matplotlib_subplot.asp.

- Stack Overflow. (n.d.). *Plotting a scatter plot in python3 where x axis is latitude/longitude in km and y axis is depth*. [online] Available at: https://stackoverflow.com/questions/65797219/plotting-a-scatter-plot-in-python3-where-x-axis-is-latitude-longitude-in-km-and [Accessed 21 Apr. 2024].

Correlation matrix:

- GeeksforGeeks. (2020). *Create a correlation Matrix using Python*. [online] Available at: https://www.geeksforgeeks.org/create-a-correlation-matrix-using-python/.

Dashboard:

- builtin.com. (n.d.). *How to Create Interactive Dashboards With Panel and Python | Built In*. [online] Available at: https://builtin.com/data-science/create-dashboards-panel-python.

Spatio-temporal analysis:

- GitHub. (n.d.). *Build software better, together*. [online] Available at: https://github.com/topics/spatio-temporal-analysis?l=python [Accessed 21 Apr. 2024].

- slat.readthedocs.io. (n.d.). *Spatial-temporal analysis — Python documentation*. [online] Available at: https://slat.readthedocs.io/en/latest/tutorials/times_series.html [Accessed 21 Apr. 2024].
- Stack Overflow. (n.d.). *How to handle data with missing timesteps to create Time series model?* [online] Available at: https://stackoverflow.com/questions/71367650/how-to-handle-data-with-missing-timesteps-to-create-time-series-model [Accessed 21 Apr. 2024].

- mrciolino (2018). *Kaggle-Competitions/Wikipedia Web Traffic/notebooks/notebook_tools/trends_explorer.py at 08af707c5a9d285ec62041294dee88f8bf3fa2d7 · mrciolino/Kaggle-Competitions*. [online] GitHub. Available at: https://github.com/mrciolino/Kaggle-Competitions/blob/08af707c5a9d285ec62041294dee88f8bf3fa2d7/Wikipedia%20Web%20Traffic/notebooks/notebook_tools/trends_explorer.py [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *python - How do I count the occurrences of a list item?* [online] Available at: https://stackoverflow.com/questions/2600191/how-do-i-count-the-occurrences-of-a-list-item.

- GitHub. (n.d.). *pyAnalytics/45-stats1/40G5_quantiles.py at 19c5074781c198c6c29c56d5d9f3896ec6a2d61c · DUanalytics/pyAnalytics*. [online] Available at: https://github.com/DUanalytics/pyAnalytics/blob/19c5074781c198c6c29c56d5d9f3896ec6a2d61c/45-stats1/40G5_quantiles.py [Accessed 21 Apr. 2024].

- Stack Overflow. (n.d.). *Superimpose heatmap on a base image OpenCV Python*. [online] Available at: https://stackoverflow.com/questions/46020894/superimpose-heatmap-on-a-base-image-opencv-python [Accessed 21 Apr. 2024].

Comparison of the findings:

- Cheng, S., Cheng, X., Ma, J., Xu, X., Zhang, W., Jinguang Lv, Bai, G., Chen, B., Ma, S., Steffen Dörner, Donner, S. and Wagner, T. (2023). Mobile MAX-DOAS observations of tropospheric NO_2 and HCHO during summer over the Three Rivers' Source region in China. *Atmospheric Chemistry and Physics*, 23(6), pp.3655–3677. doi:https://doi.org/10.5194/acp-23-3655-2023.
- Sun, Y., Yin, H., Liu, C., Zhang, L., Cheng, Y., Palm, M., Justus Notholt, Lu, X., Vigouroux, C., Zheng, B., Wang, W., Jones, N.B., Shan, C., Qin, M., Tian, Y., Hu, Q., Meng, F. and Liu, J. (2021). Mapping the drivers of formaldehyde (HCHO) variability from 2015 to 2019 over eastern China: insights from Fourier transform infrared observation and GEOS-Chem model simulation. 21(8), pp.6365–6387. doi:https://doi.org/10.5194/acp-21-6365-2021.

- Ou, J., Hu, Q., Xing, C., Zhu, Y., Feng, J., Ji, X., Zhang, M., Wang, X., Li, L., Liu, T., Chang, B., Li, Q., Yin, H. and Liu, C. (2023). Analysis of the Vertical Distribution and Driving Factors of Aerosol and Ozone Precursors in Huaniao Island, China, Based on Ground-Based MAX-DOAS. *Remote Sensing*, [online] 15(21), p.5103. doi:https://doi.org/10.3390/rs15215103.

- Chong, K., Wang, Y., Liu, C., Gao, Y., K. Folkert Boersma, Tang, J. and Wang, X. (2024). Remote Sensing Measurements at a Rural Site in China: Implications for Satellite

NO_2 and HCHO Measurement Uncertainty and Emissions From Fires. *Journal of geophysical research. Atmospheres*, 129(2). doi:https://doi.org/10.1029/2023jd039310.

- Cheng, S., Cheng, X., Ma, J., Xu, X., Zhang, W., Jinguang Lv, Bai, G., Chen, B., Ma, S., Steffen Dörner, Donner, S. and Wagner, T. (2023). Mobile MAX-DOAS observations of tropospheric NO_2 and HCHO during summer over the Three Rivers' Source region in China. *Atmospheric Chemistry and Physics*, 23(6), pp.3655–3677. doi:https://doi.org/10.5194/acp-23-3655-2023.