Foundations of Data Science Group Project
Air Quality Analysis
Task 4: Documentation

Matthias Mifsud and Matthew Micallef

Work division

We both tried to split the workload evenly by putting an even amount of effort in each task. Every single part of each task was done together with the other overlooking the work being done. For the purpose of efficiency, we had to split some sections separately.

Matthias Mifsud:

Mainly worked to create a functioning code for the air quality analysis of the dataset chosen.

Matthew Micallef:

Mainly worked on the documentation part of the project and the designing of the flex dashboard.

Although we worked mainly on different sections, we still dedicated time to review each other's work and assist the other with other sections.

Task 1: Understanding the Data

Extracting and Describing Variables

```
##         Date Time PM2.5 PM10 NO2 SO2  CO O3 Temperature
## 1 2013-03-01    0     4    4   7   4 300 77        -0.7
## 2 2013-03-01    1     8    8   7   4 300 77        -1.1
## 3 2013-03-01    2     7    7  10   5 300 73        -1.1
## 4 2013-03-01    3     6    6  11  11 300 72        -1.4
## 5 2013-03-01    4     3    3  12  12 300 72        -2.0
## 6 2013-03-01    5     5    5  18  18 400 66        -2.2
```

Description

Date: The day, month and year of which the monitoring occurred.

Time: The hour at which the monitoring took place.

PM2.5 (Particular Matter <= 2.5): Fine particular matter in the air that is 2.5 micrometers or smaller.

PM10 (Particular Matter <= 10): Fine particular matter in the air that is 10 micrometers or smaller.

NO2 (Nitrogen Dioxide): A harmful gas pollutant produced from a combustion process.

SO2 (Sulfur Dioxide): A harmful gas produced mainly through the burning of fossil fuels.

CO (Carbon Monoxide): A colorless and odorless gas pollutant produced through incomplete fuel combustion.

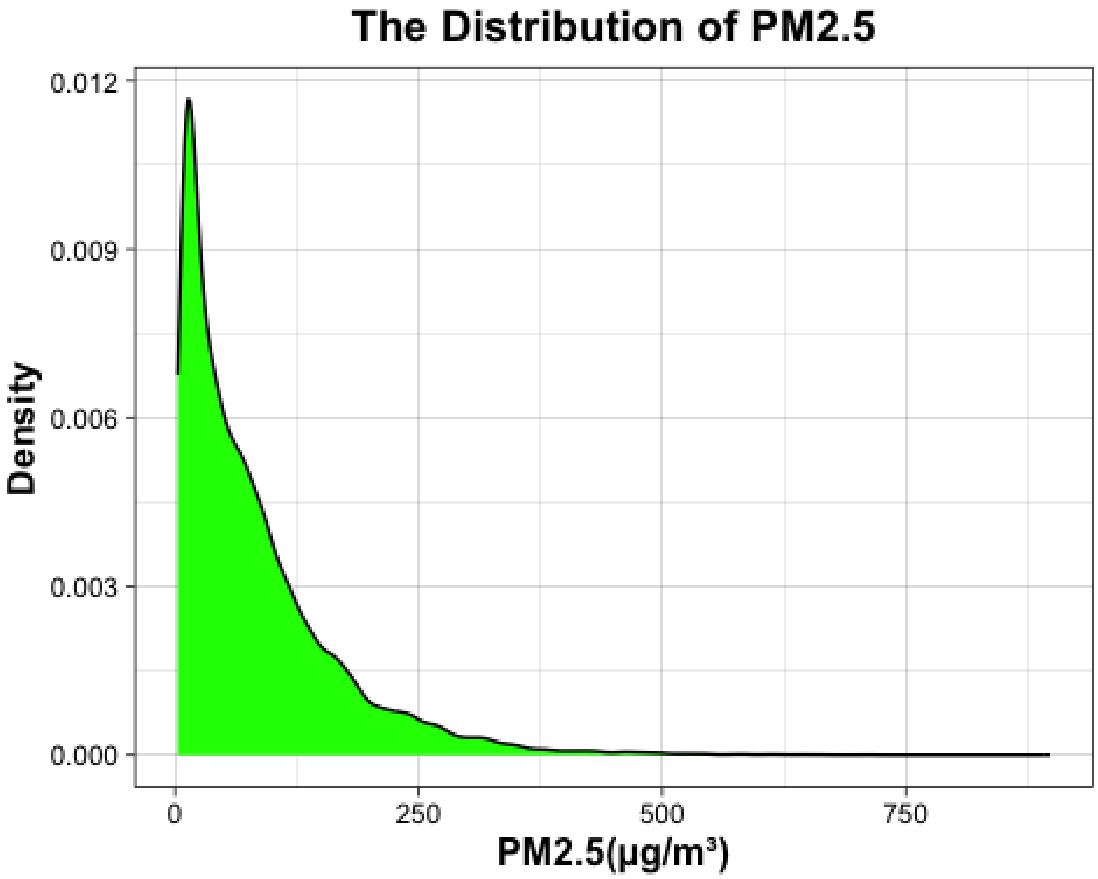
O3 (Ozone): A harmful gas produced due to the reaction of sunlight and other pollutants.

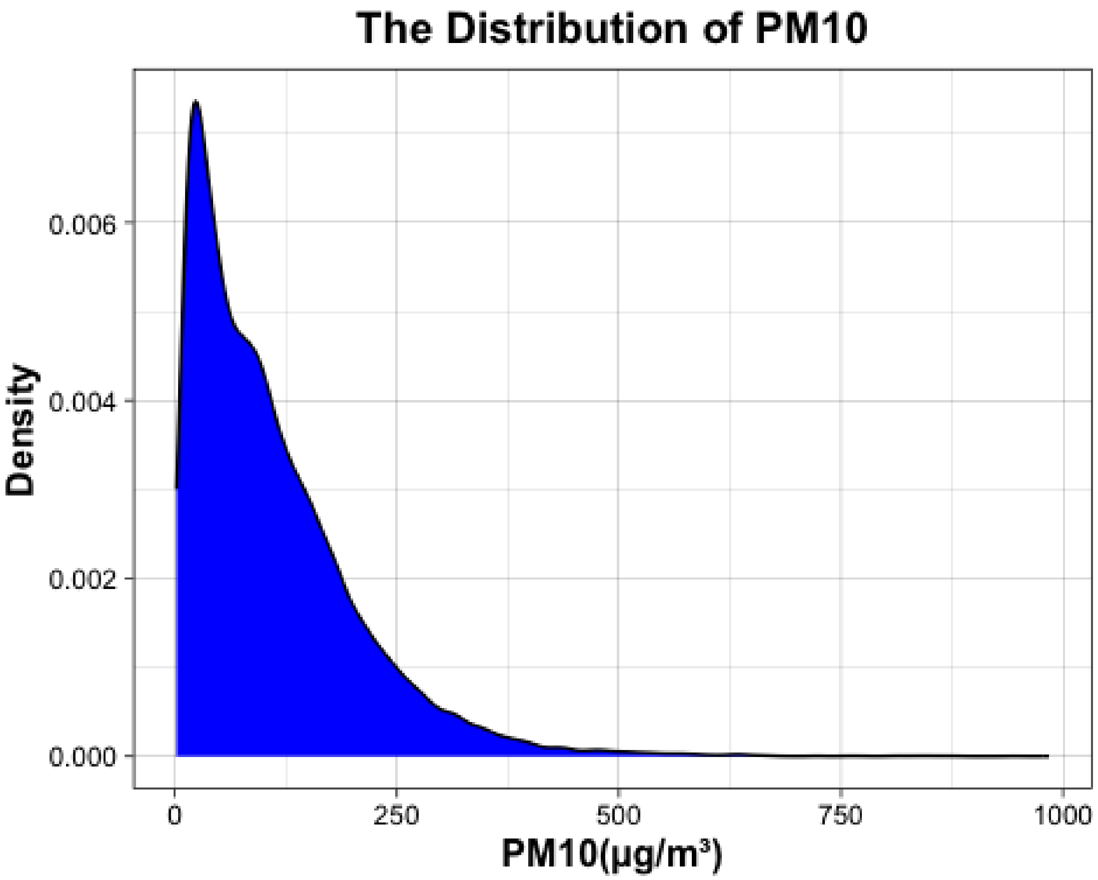
Temperature: The temperature in Celcius at the monitoring site.

Summary Statistics (mean, median and standard deviation)

```
##      Variable       mean median         sd
## 1       PM2.5   82.77361   58.0   82.13569
## 2        PM10  110.06039   87.0   95.22300
## 3         SO2   17.37590    9.0   22.82302
## 4         NO2   59.30583   53.0   37.11620
## 5          CO 1262.94515  900.0 1221.43624
## 6          O3   56.35336   42.0   57.91633
## 7 Temperature   13.58461   14.5   11.39910
```

Plots of the Distributions of PM2.5 and PM10 Levels





Observations About Air Quality Trends

The above distributions of PM2.5 and PM10 both show to be an example of a right-skew distribution. Being a right skew distribution implies that low amounts of fine particular matter(PM) were most commonly recorded, with occasional higher levels being recorded as well. This indicates that high concentrations of PM are less frequent but still present.

Task 2: Cleaning the Data

Identifying and Counting the Number of Missing Values

```
##      No    year   month     day    hour   PM2.5    PM10     SO2     NO2 
CO 
##       0       0       0       0       0     925     718     935    1023 
1776 
##      O3    TEMP    PRES    DEWP    RAIN      wd    WSPM station    Date 
##    1719      20      20      20      20      81      14       0       0

## [1] "Total number of missing values in 'monitoring_dataset': 7271"
```

List of Variables That Have Missing Values

```
##  [1] "Variable that include missing values: PM2.5"
##  [2] "Variable that include missing values: PM10"
##  [3] "Variable that include missing values: SO2"
##  [4] "Variable that include missing values: NO2"
##  [5] "Variable that include missing values: CO"
##  [6] "Variable that include missing values: O3"
##  [7] "Variable that include missing values: TEMP"
##  [8] "Variable that include missing values: PRES"
##  [9] "Variable that include missing values: DEWP"
## [10] "Variable that include missing values: RAIN"
## [11] "Variable that include missing values: wd"
## [12] "Variable that include missing values: WSPM"
```

Justification and Application of an Imputation Strategy to Handle Missing Values

Justification

A mixture of mean-substitution and forward-fill were chosen as the imputation strategies for handling missing values. Forward-fill was used to hold variables that vary slightly over time: the TEMP, PRES, DEWP, RAIN and wd variables. The mean-substitutions were used for those values that may vary significantly over time: the PM2.5, PM10, SO2, NO2, CO and O3 variables.

Application

```r
cleaned_monitoring_dataset <- monitoring_centre_data

#forward-fill
cleaned_monitoring_dataset$TEMP <- na.locf(cleaned_monitoring_dataset$TEMP,
                                           na.rm = FALSE,
                                           fromLast = FALSE)
cleaned_monitoring_dataset$PRES <- na.locf(cleaned_monitoring_dataset$PRES,
                                           na.rm = FALSE,
                                           fromLast = FALSE)
cleaned_monitoring_dataset$DEWP <- na.locf(cleaned_monitoring_dataset$DEWP,
                                           na.rm = FALSE,
                                           fromLast = FALSE)
cleaned_monitoring_dataset$RAIN <- na.locf(cleaned_monitoring_dataset$RAIN,
                                           na.rm = FALSE,
                                           fromLast = FALSE)
cleaned_monitoring_dataset$wd <- na.locf(cleaned_monitoring_dataset$wd,
                                         na.rm = FALSE,
                                         fromLast = FALSE)
cleaned_monitoring_dataset$WSPM <- na.locf(cleaned_monitoring_dataset$WSPM,
                                           na.rm = FALSE,
                                           fromLast = FALSE)

#mean-substitution
cleaned_monitoring_dataset$PM2.5[is.na(cleaned_monitoring_dataset$PM2.5)] <-
  mean(cleaned_monitoring_dataset$PM2.5, na.rm = TRUE)
cleaned_monitoring_dataset$PM10[is.na(cleaned_monitoring_dataset$PM10)] <-
  mean(cleaned_monitoring_dataset$PM10, na.rm = TRUE)
cleaned_monitoring_dataset$SO2[is.na(cleaned_monitoring_dataset$SO2)] <-
  mean(cleaned_monitoring_dataset$SO2, na.rm = TRUE)
cleaned_monitoring_dataset$NO2[is.na(cleaned_monitoring_dataset$NO2)] <-
  mean(cleaned_monitoring_dataset$NO2, na.rm = TRUE)
cleaned_monitoring_dataset$CO[is.na(cleaned_monitoring_dataset$CO)] <-
  mean(cleaned_monitoring_dataset$CO, na.rm = TRUE)
cleaned_monitoring_dataset$O3[is.na(cleaned_monitoring_dataset$O3)] <-
  mean(cleaned_monitoring_dataset$O3, na.rm = TRUE)
```

Checking That There is No Missing Values in the Cleaned Data-set

```
## [1] "Amount of missing data of original dataset: 7271"

## [1] "Amount of missing data of cleaned dataset: 0"
```

Creating a Cleaned Copy of the Original CSV File

```r
write.csv(cleaned_monitoring_dataset,
          "cleaned_PRSA_Data_Aotizhongxin_20130301-20170228.csv")
```

Task 3: Data Analysis

Correlations between PM2.5, PM10, NO2, TEMP

```
##   Variable      PM2.5       PM10        NO2       TEMP
## 1    PM2.5  1.0000000  0.8741056  0.6768694 -0.1271987
## 2     PM10  0.8741056  1.0000000  0.6469587 -0.1095267
## 3      NO2  0.6768694  0.6469587  1.0000000 -0.2371195
## 4     TEMP -0.1271987 -0.1095267 -0.2371195  1.0000000
```

Interpretation of the Relationships Between Pollutants and Temperature

PM2.5 and PM10:

PM2.5 and PM10 were shown to be highly correlated at 0.8741056. This indicates that they are highly dependent on one another hence, a gain or increase in one variable ensures the same effect for the other.

PM2.5 and NO2:

The correlation of PM2.5 and NO2 show to be moderately correlated at 0.6768694. This correlation is not as strong as that between PM2.5 and PM10 but still significant enough so that a change in one variable is possible to cause a change to the other variable.

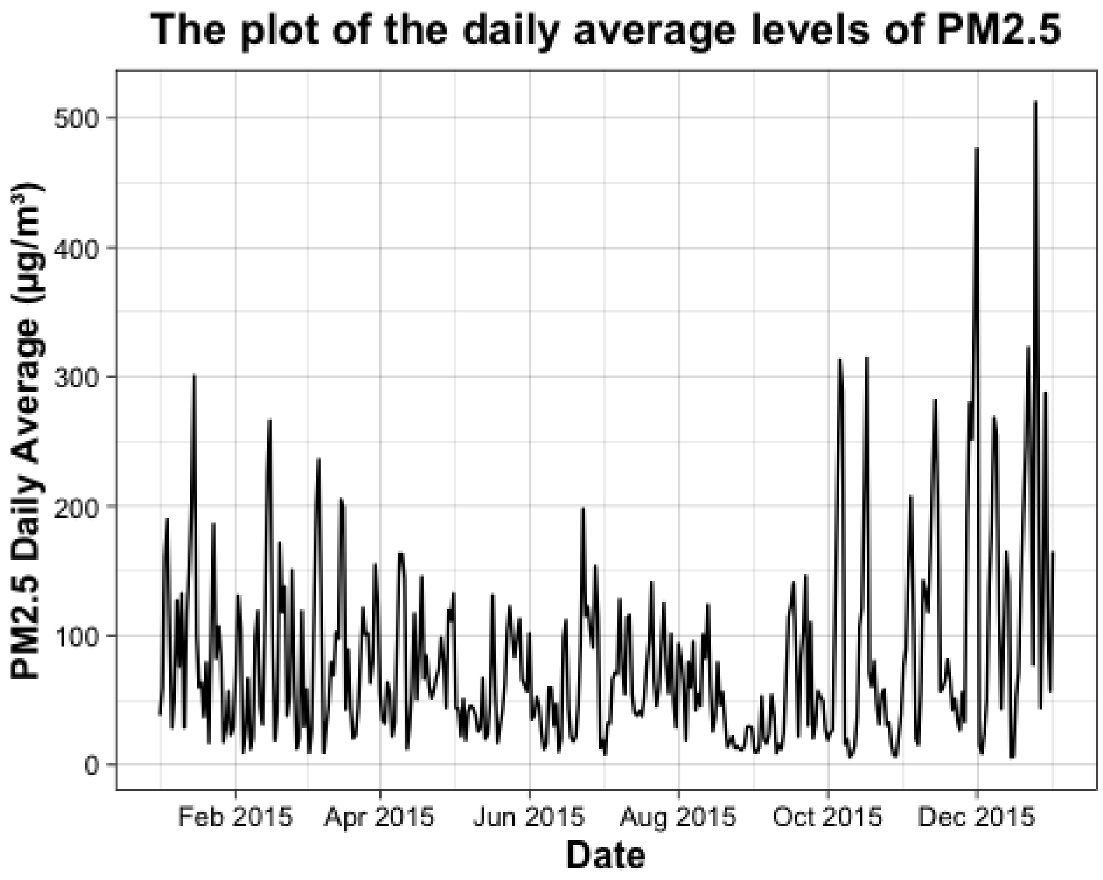
PM10 and NO2:

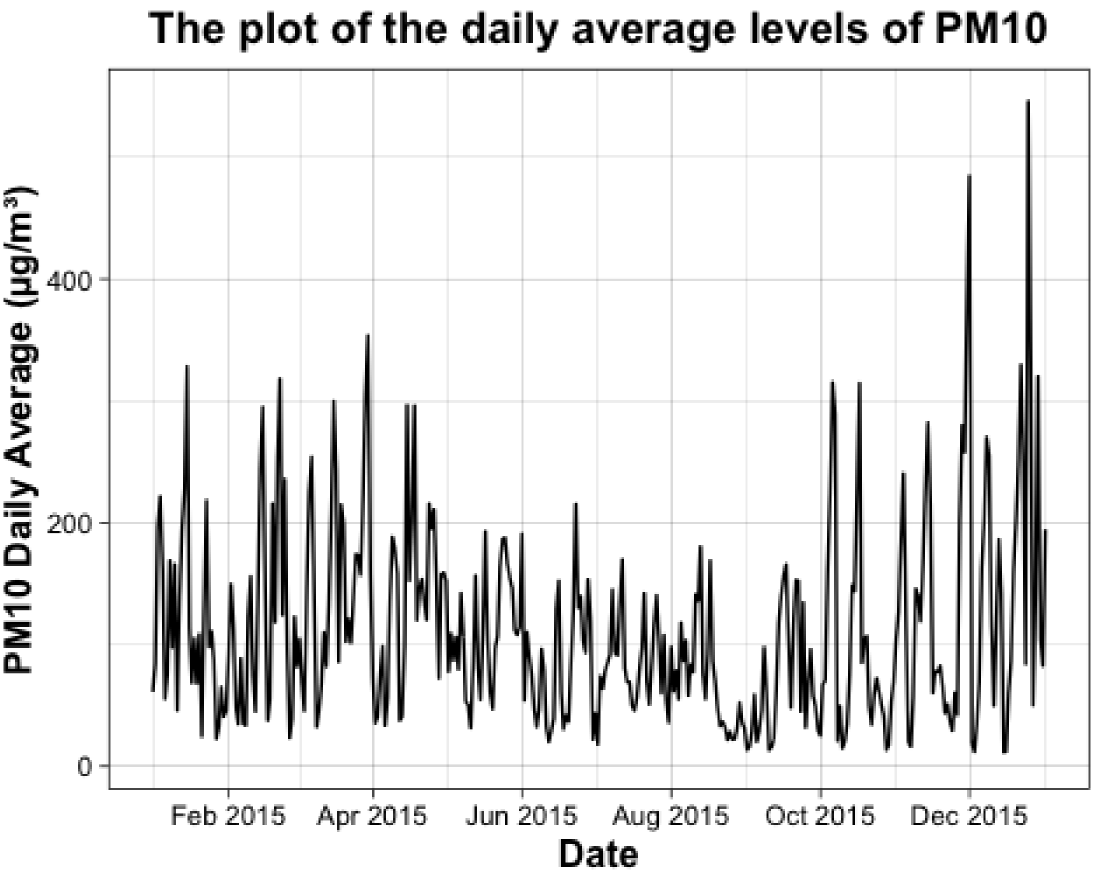
The correlation of PM10 and NO2 show to be moderately correlated at 0.6469587. This correlation is not as strong as that between PM2.5 and PM10 but still significant enough so that a change in one variable may cause a change to the other variable.

Pollutants(PM2.5, PM10, NO2) and Temperature(TEMP):

All pollutants show a slight negative correlation with temperature, implying that a gain in temperature might result in lower pollution values, while a decrease in temperature may indicate a gain in pollution values. The correlation is, however, very small, which means that this result is quite unlikely.

Plots of the Daily Average Levels of PM2.5 and PM10 Over Time





Noticeable Trends and Anomalies

From the two graphs, we can notice that both PM2.5 and PM10 levels suffer in maintaining consistency in their values. They are constantly alternating from maxima to minima throughout the months. From the graphs, we can conclude that during the summer periods, PM2.5 and PM10 levels are the lowest, indicating a lower pollution rate, which may be due to hotter temperatures since PM2.5 and PM10 have a negative correlation with Temperature. On the other hand, PM2.5 and PM10 show higher levels during other seasonal periods, especially during the December period. This indicates that higher pollution rates are expected during this period, which could be due to colder weather due to the negative correlation between PM and Temperature values.

Some very visible anomalies were during the December period, where large spikes of PM2.5 and PM10 values occurred, which exceeded normal values. This could be due to some specific events (e.g. large-scale construction, extreme traffic or forest fires) that cause greater than usual PM values in the air. Such spikes appear constantly throughout the graphs but on smaller levels, indicating a constant change in environmental conditions (e.g. change in weather, traffic conditions and construction patterns).

Comparing Average Levels of PM2.5 During Weekdays and Weekends

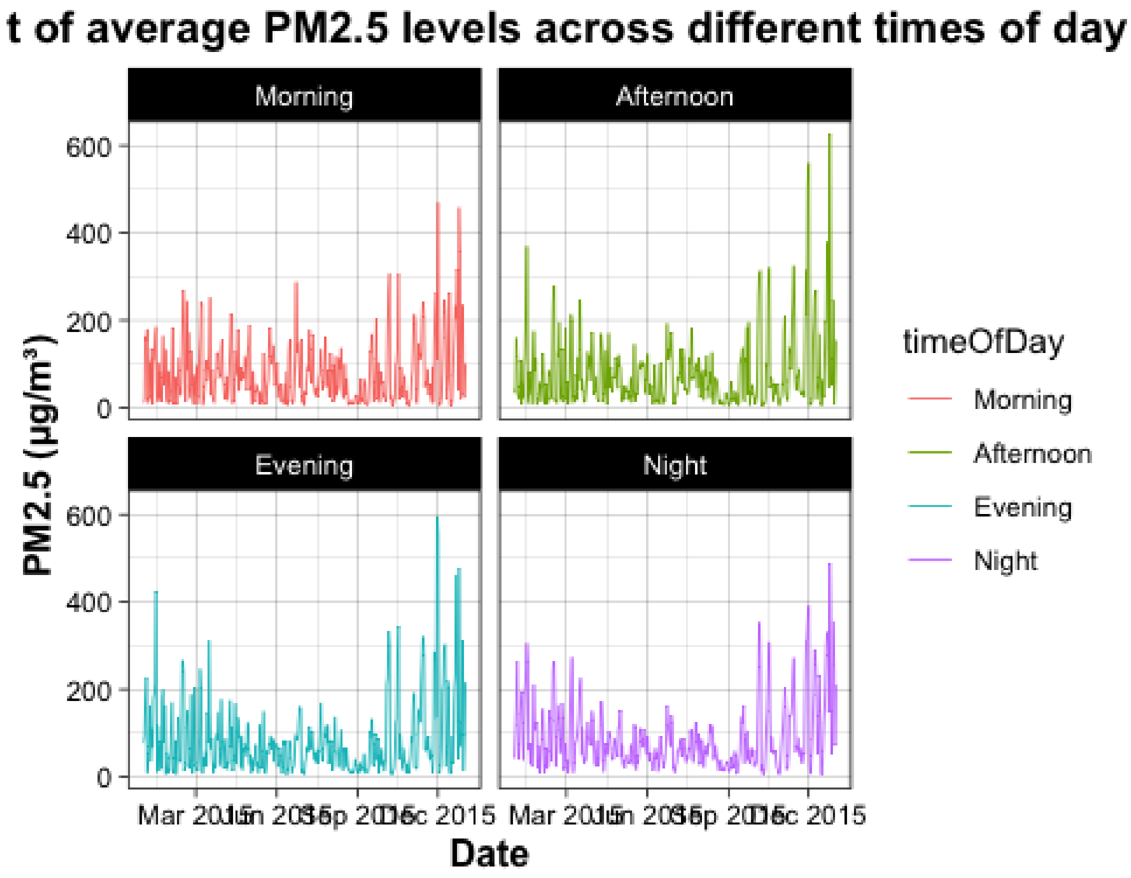
```
## [1] "Weekday average of PM2.5: 81.0258444667551"

## [1] "Weekend average of PM2.5: 87.1346642067919"
```

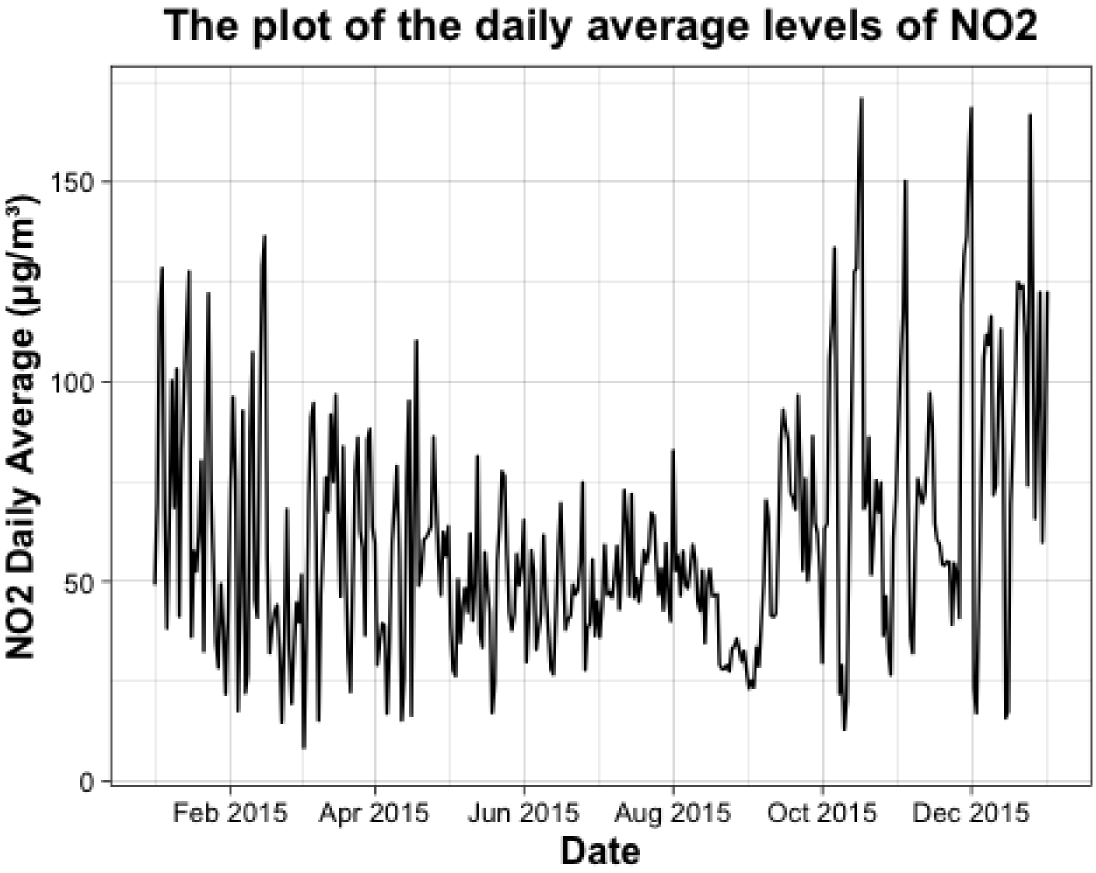
Interpreting the Difference

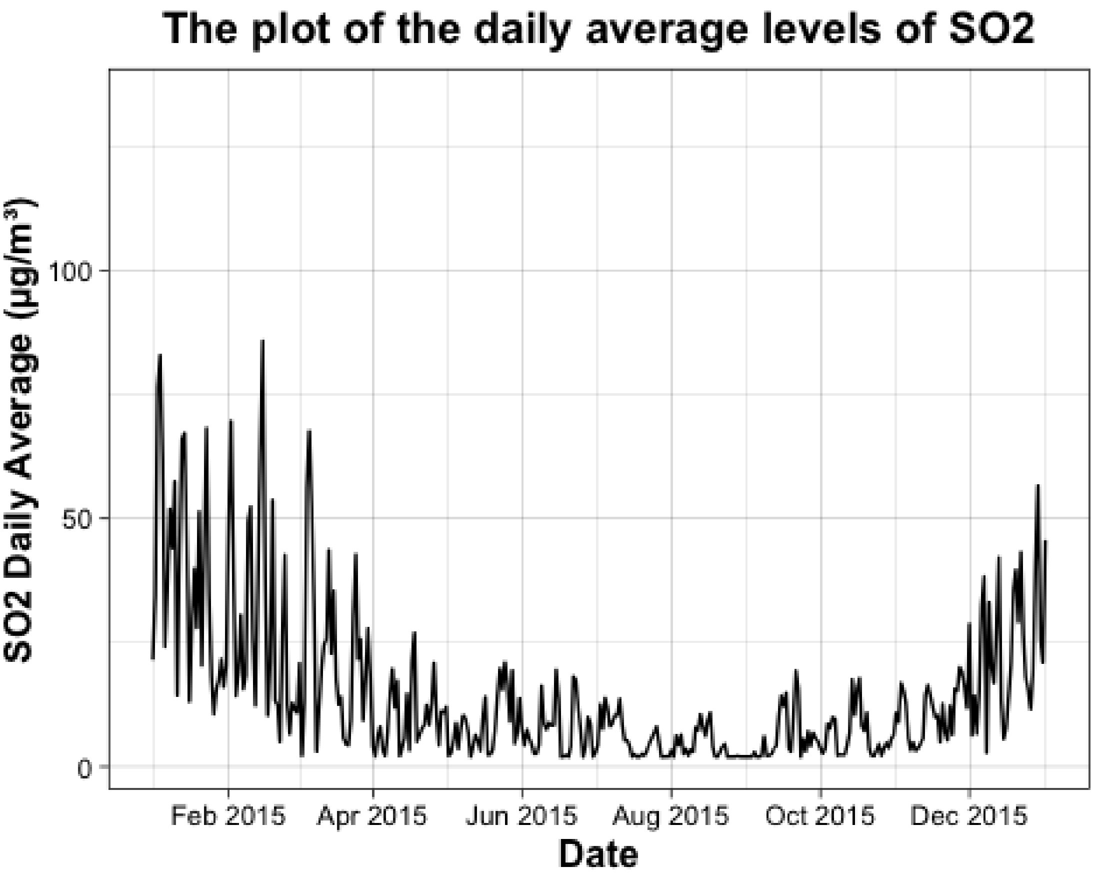
The weekend average PM2.5 levels seem higher than weekday PM2.5 levels by approximately 6 μg/m^3. This possibly shows that an increase in pollution occurs during the weekend. Many factors could cause this increase, one of which is the enhanced traffic conditions due to the fact that many people are going out for leisure.

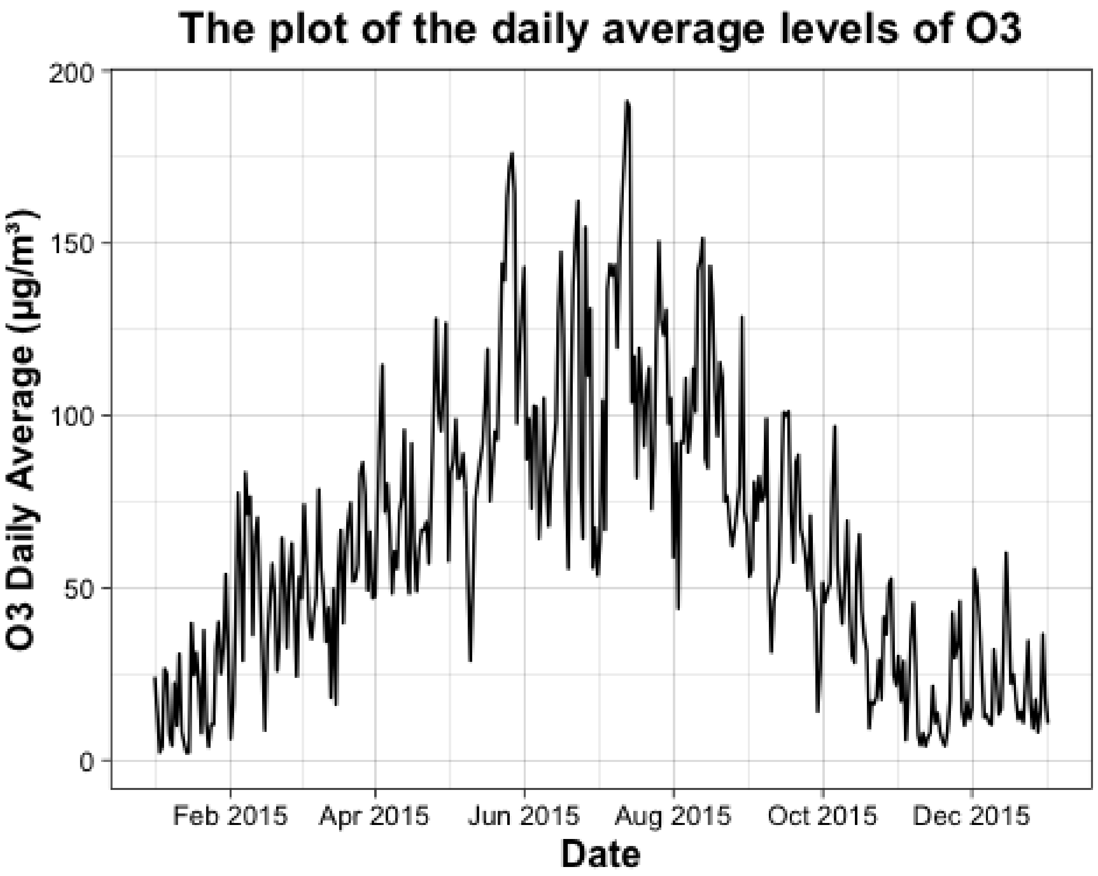
Multi-panel Plots Showing Variations in PM2.5 Across Different Times of the Day (morning, afternoon, evening, and night)

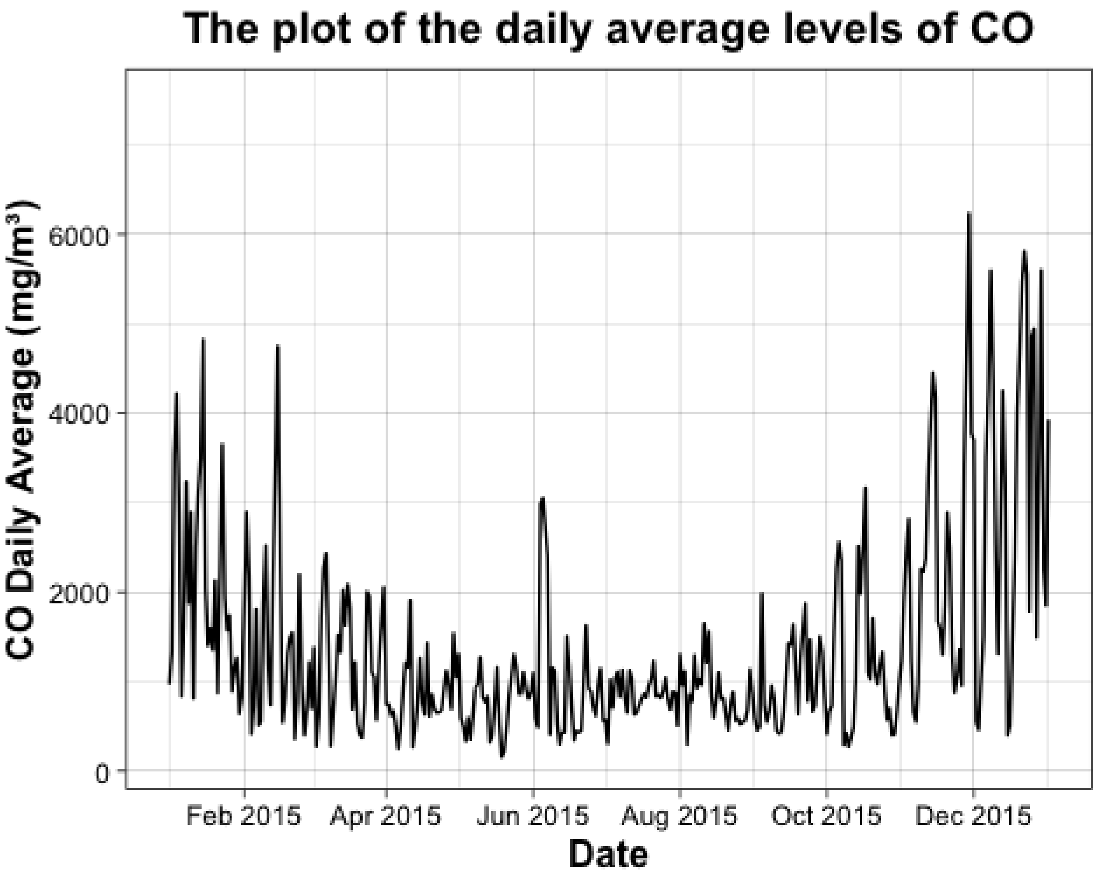


Plots of the Daily Average Levels of NO2, SO2, O3, and CO Over Time

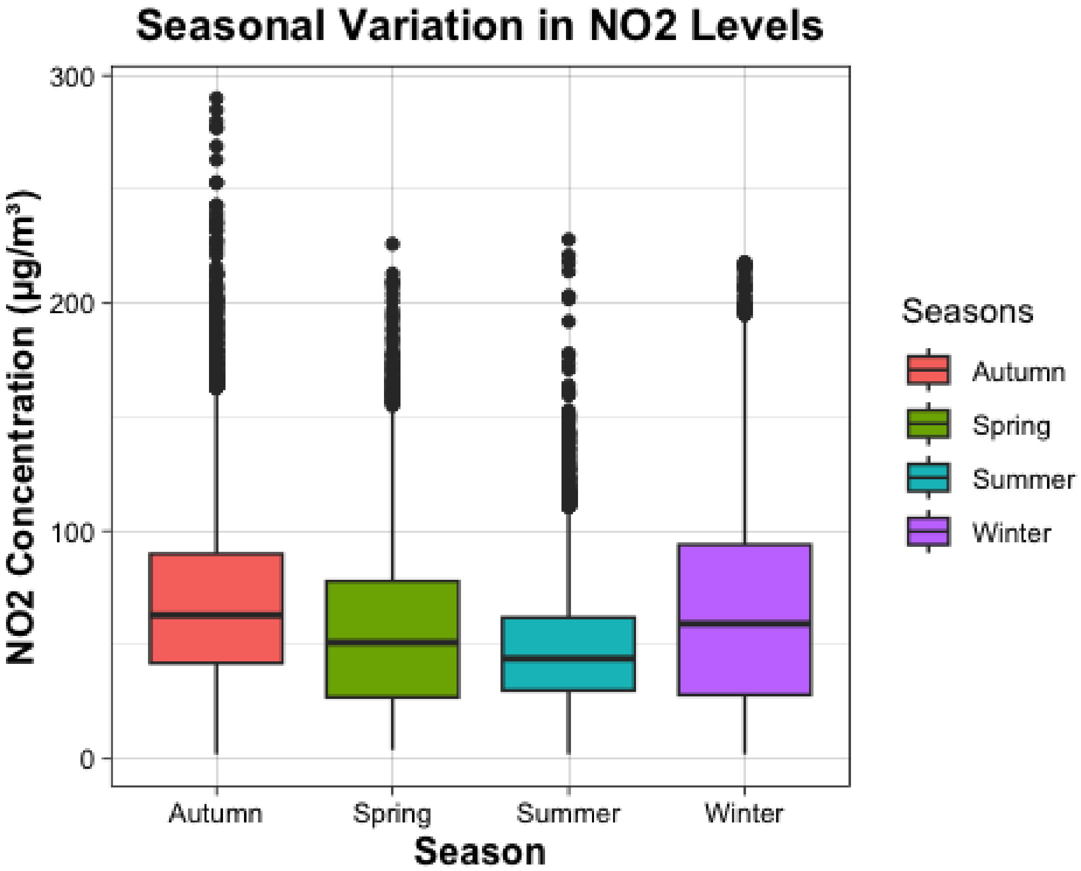


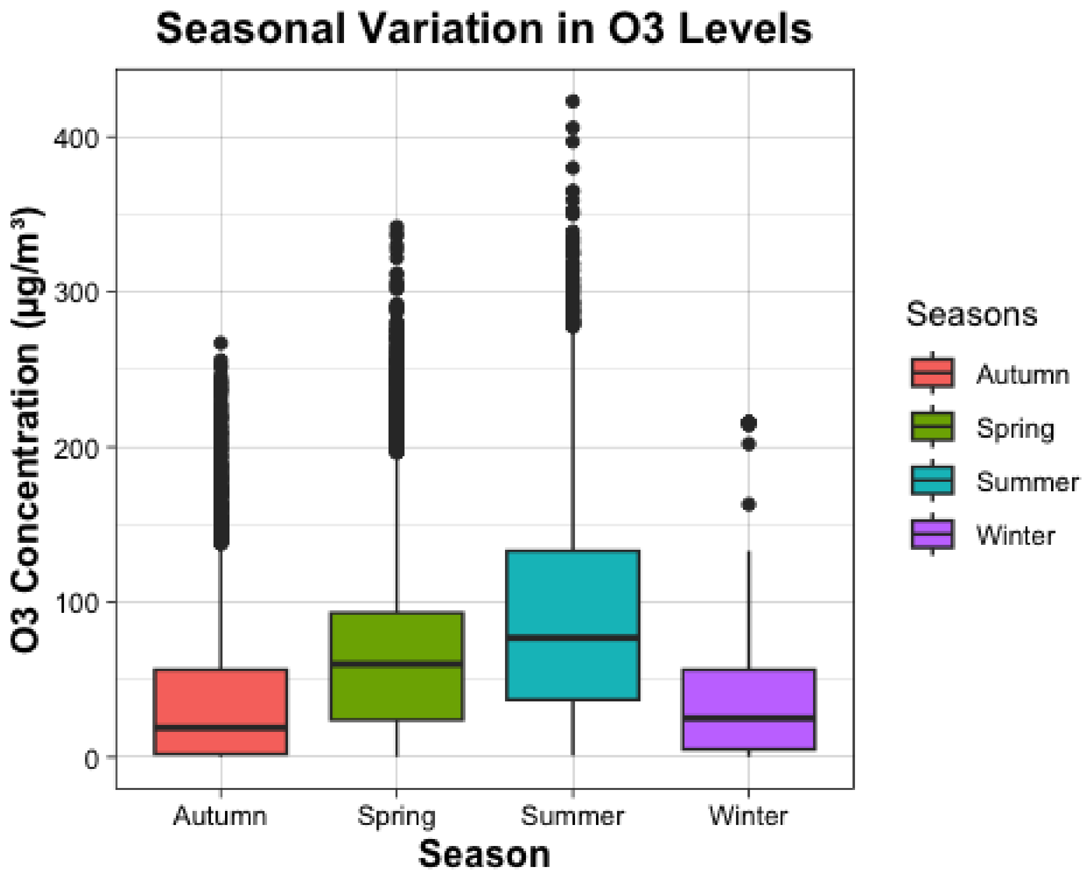

The plot of the daily average levels of SO2
SO2 Daily Average (µg/m³)
100
50
0
Feb 2015
Apr 2015
Jun 2015
Aug 2015
Oct 2015
Dec 2015
Date


The plot of the daily average levels of O3
O3 Daily Average (μg/m³)
200
150
100
50
0
Feb 2015
Apr 2015
Jun 2015
Aug 2015
Oct 2015
Dec 2015
Date


The plot of the daily average levels of CO
CO Daily Average (mg/m³)
6000
4000
2000
0
Feb 2015
Apr 2015
Jun 2015
Aug 2015
Oct 2015
Dec 2015
Date

Plots of the Seasonal Variations in NO2 and O3





Observed patterns:

From the graph of “Seasonal Variation in NO2 Levels”, we can notice that NO2 levels have a higher median in the autumn and winter periods. This could be due to the negative correlation between pollutants and temperature and could also be due to the higher heating demand during cold periods, which raises NO2 levels as a result of the combustion process in the power plants. A higher IQR is also seen during these periods, indicating a larger variability in pollution rates. A significantly lower median during the summer and spring periods can be seen, which may be due to lower heating demands, which reduce NO2 levels. A much lower IQR is also seen, especially in the summer period, indicating lower variability in pollution rates.

A significant rise in median O3 levels is seen during the spring and summer periods. This could be due to the increased intensity of sunlight, which, when combined with pollutants, O3. A very large IQR is also visible, mainly in summer, which indicates a greater variation in O3 levels, which could be due to weather events. Much lower median O3 rates are seen during the autumn and winter periods, which could be due to lower sunlight intensity. A much lower IQR is also seen for these seasons, which indicates a greater stability in O# values during the autumn and winter seasons.

Top Three Most Polluted Days (highest PM2.5 levels) and Analysis of Meteorological Conditions (e.g. temperature, humidity, and wind speed) of Those Days

```
##         Date    PM2.5  Temperature   Humidity Wind_Speed
## 1 25/12/2015 512.2917 -3.175000000 -3.7791667   1.350000
## 2 01/12/2015 476.4489 -0.008333333 -0.4708333   1.191667
## 3 01/01/2017 464.0833 -2.895833333 -5.0291667   1.095833
```

Potential Reasons for the High Levels of Pollution

25/12/2015

On December 25, 2015, Beijing experienced severe smog, among the worst of the year, which cancelled flights, visibility was extremely low, and the city remained engulfed in hazardous pollution prompting another “red alert” status mainly caused by coal burning.

https://www.forbes.com/sites/trevornace/2015/12/09/beijing-declares-red-alert-pollution-haze-visible-space/ (Dec 17, 2015)

https://indianexpress.com/article/world/world-news/years-worst-pollution-engulfs-beijing-on-christmas-day/ (Dec 25, 2015)

https://www.nytimes.com/2015/12/25/world/asia/liu-bolin-air-pollution-beijing-winter-solstice.html (Dec 24, 2015)

https://www.chinadaily.com.cn/china/2015-12/25/content_22806691.htm (Dec 25, 2015)

01/12/2015

On December 1, 2015, Beijing faced extreme air pollution that darkened the sky, creating hazardous conditions for residents and a “red alert”.

https://www.bbc.com/news/world-asia-china-35028483 (Dec 7, 2015)

https://www.bbc.com/news/av/world-asia-china-34972375 (Dec 1, 2015)

https://www.bbc.com/news/world-asia-china-35026363 (Dec 8, 2015)

01/01/2017

On January 1, 2017, Beijing was masked in thick, toxic smog marking one of the longest air pollution episodes in recent years. Health alerts were issued, and the smog caused widespread travel chaos.

https://www.theguardian.com/world/2017/jan/02/china-smog-millions-start-new-year-shrouded-by-health-alerts-and-travel-chaos (Jan 2, 2017)

https://unearthed.greenpeace.org/2017/01/11/beijing-air-pollution-smog-episode-longest-this-century (Jan 1, 2017)

https://mashable.com/article/china-smog-new-year (Jan 1, 2017)

https://hongkongfp.com/2017/01/01/beijing-starts-2017-cloud-thick-toxic-smog/ (Jan 1, 2017)

Task 4: Use Of Generative AI Tools

Lists of AI-Generated Prompts and Corresponding Outputs

1.

Prompt:

In R commander, how can I convert the rownames to another column with a column name?

Response:

Steps:

1. Load the tibble package (if not already installed):

Make sure the tibble package is installed. If it's not, install it using:

```
install.packages("tibble")
```

2. Use the Script Window in R Commander:

In R Commander, you can execute R code by opening the script window. To do so:

- Click on "Script" in the R Commander toolbar.

- Enter your R code into the script editor.

3. Convert Row Names to a Column:

Assuming your dataset is named mydata in R Commander:

```
library(tibble)
mydata <- rownames_to_column(mydata, var = "new_column_name")
```

Replace new_column_name with the desired name for the new column.

2.

Prompt:

```
summary_ui <- fluidPage(
  titlePanel("Summary Statistics of Date, Time, PM2.5, PM10, NO2, SO2, CO, O3
and Temperature"),
  fluidRow(
    column(12,
           tableOutput("summaryTable")
           )
    )
  )
```

In the above code the summaryTable is not expanding to all 12 columns. What can I add to the code to fix this issue?

Response:

The issue might be related to the default styling applied to the tableOutput. By default, the tableOutput might not expand to the full width of its parent container (the 12-column column in your layout). To fix this, you can explicitly define CSS styles or use the DT package's dataTableOutput for better control and responsiveness.

Here's how to fix the issue:

Add Custom CSS

If you want to apply custom styling more globally, you can add a CSS block to your app's tags$head to set the tableOutput style:

```
summary_ui <- fluidPage(
  titlePanel("Summary Statistics of Date, Time, PM2.5, PM10, NO2, SO2, CO, O3
and Temperature"),
  tags$head(
    tags$style(HTML("
      #summaryTable {
        width: 100%;
        overflow-x: auto;
      }
    "))
  ),
  fluidRow(
    column(12,
           tableOutput("summaryTable")
    )
  )
)
```

3.

Prompt:

Help me filter by date using shiny on a ggplot2 graph.

Response:

To filter data by date and display a filtered ggplot2 graph using Shiny, you can create an interactive app that allows users to select a date range. Here’s a step-by-step guide to help you filter the data based on dates and then display a ggplot2 graph:

Step 1: Setup the Shiny App Structure

A basic Shiny app consists of two main parts:

```
•   UI (User Interface): This defines the layout and input controls for
users.

•   Server: This contains the logic for rendering plots and processing data.
```

Example Code

Let’s assume you have a dataset with a Date column and you want to create a plot that filters data by date.

```r
# Load necessary libraries
library(shiny)
library(ggplot2)
library(dplyr)

# Example dataset
data <- data.frame(
  Date = as.Date('2020-01-01') + 0:100,
  Value = rnorm(101, mean = 50, sd = 10)
)

# Define UI
ui <- fluidPage(
  titlePanel("Filter Data by Date"),

  # Date range input
  sidebarLayout(
    sidebarPanel(
      dateRangeInput("date_range",
                     label = "Select Date Range:",
                     start = min(data$Date),
                     end = max(data$Date),
                     min = min(data$Date),
                     max = max(data$Date))
    ),

    # Plot output
    mainPanel(
      plotOutput("date_plot")
    )
  )
)

# Define server
server <- function(input, output) {

  # Reactive expression to filter the data based on selected date range
  filtered_data <- reactive({
    data %>%
      filter(Date >= input$date_range[1] & Date <= input$date_range[2])
  })

  # Render the ggplot based on filtered data
  output$date_plot <- renderPlot({
    ggplot(filtered_data(), aes(x = Date, y = Value)) +
      geom_line() +
      labs(title = "Filtered Data by Date",
           x = "Date", y = "Value") +
      theme_minimal()
  })
}

# Run the application
shinyApp(ui = ui, server = server)
```

Explanation of the Code:

```
1.  UI (User Interface):

•   dateRangeInput: This creates an interactive date range input where the
user can select a start and end date.

•   plotOutput: This is where the plot will be rendered based on the filtered
data.

2.  Server:

•   filtered_data (Reactive expression): This filters the data based on the
selected date range. It updates whenever the user changes the date range.

•   renderPlot: This function generates the ggplot using the filtered data.
```

```
It will automatically update whenever the filtered data changes.

3.  Plot: The ggplot2 plot is displayed with Date on the x-axis and Value on
the y-axis. You can replace this with your own dataset and columns.
```

Step 2: Customize the App for Your Data

If you want to use your own dataset:

```
1.  Replace the data dataframe with your own data (ensure it has a column of
class Date for the date filtering to work).

2.  Adjust the aes() mapping in the ggplot to fit your actual data columns
(e.g., aes(x = YourDateColumn, y = YourYColumn)).
```

Step 3: Running the App

To run this app:

```
1.  Copy the code into an R script.

2.  Run the script using shinyApp(ui = ui, server = server).

3.  The app will open in your default web browser where you can interact with
it.
```

Now you have an interactive Shiny app that allows you to filter your data by date and display a ggplot2 graph accordingly.

4.

Prompt:

I am using Shiny implementations and I am attempting to filter a graph by different pollutants which are on the x-axis. To display the graph I am using ggplot to but the following code doesn't seem to work properly.

```
ggplot(data = filtered_data(), aes(x = input$PM)) +
  geom_density(fill = "blue", na.rm = TRUE) +
  labs(x = paste(input$PM, "(µg/m³)"), y = "Density") +
  theme_linedraw()
```

How can I fix this?

Response:

The issue you're encountering is likely because input$PM is being passed as a string (the name of the pollutant) rather than the actual variable from your data. In ggplot2, the aes() function expects the actual data column, not the name of the column as a string.

To fix this, you need to programmatically map the variable to the x-axis in ggplot. You can achieve this using aes_string() or aes() with !! (pronounced "bang-bang") and sym() from the rlang package.

Here's a modified version of your code that should work:

Option 1: Using aes_string() (this works if you're using a character string directly):

```
ggplot(data = filtered_data(), aes_string(x = input$PM)) +
  geom_density(fill = "blue", na.rm = TRUE) +
  labs(x = paste(input$PM, "(µg/m³)"), y = "Density") +
  theme_linedraw()
```

Option 2: Using aes() with !! (more modern approach with rlang):

```
library(rlang)

ggplot(data = filtered_data(), aes(x = !!sym(input$PM))) +
  geom_density(fill = "blue", na.rm = TRUE) +
  labs(x = paste(input$PM, "(µg/m³)"), y = "Density") +
  theme_linedraw()
```

Explanation:

```
1.  aes_string(): This is an older function that allows you to pass column
names as strings. If you use input$PM as a string with this function, it will
correctly map to the corresponding column in filtered_data().
```

```
2.  aes() with !!sym(): This approach is more modern and uses the rlang
package to evaluate the string as a variable in ggplot2. sym(input$PM)
creates a symbol from the string, and !! unquotes it so it is interpreted as
a column name.
```

Try either of these solutions, depending on your preference, and it should correctly map the pollutant variable to the x-axis.

5.

Prompt:

Generate some brief descriptions about the pollutants: PM2.5, PM10, SO2, NO2, CO, O3 and Temperature.

Response:

Here are brief descriptions for the pollutants and temperature you mentioned:

`1.  PM2.5 (Particulate Matter ≤ 2.5 micrometers):`

Fine particles in the air that are 2.5 micrometers or smaller. These particles can penetrate deep into the lungs and bloodstream, causing respiratory and cardiovascular problems. Common sources include vehicle emissions, industrial processes, and wildfires.

`2.  PM10 (Particulate Matter ≤ 10 micrometers):`

Larger particulate matter (10 micrometers or smaller) that can be inhaled into the respiratory system. Though not as harmful as PM2.5, it still poses health risks, especially to the lungs and throat. Sources include dust, pollen, and emissions from vehicles and industries.

`3.  SO2 (Sulfur Dioxide):`

A colorless gas produced by the combustion of fossil fuels, especially coal and oil, and during industrial processes like smelting metal ores. It can irritate the eyes, nose, and throat, and contributes to acid rain and the formation of particulate matter.

`4.  NO2 (Nitrogen Dioxide):`

A reddish-brown gas formed from the combustion of fossil fuels in vehicles and power plants. NO2 can irritate the respiratory system, aggravating asthma and lung diseases. It also contributes to the formation of ground-level ozone and acid rain.

`5.  CO (Carbon Monoxide):`

A colorless, odorless gas produced by incomplete combustion of carbon-containing fuels (e.g., from vehicles, industrial processes, and wood burning). High CO levels can be toxic, especially in confined spaces, leading to headaches, dizziness, and even death in extreme cases.

`6.  O3 (Ozone):`

Ozone is a gas that occurs naturally in the upper atmosphere (stratosphere) but can also form at ground level due to the reaction of sunlight with pollutants like NOx and volatile organic compounds. Ground-level ozone is a major component of smog and can cause respiratory issues and exacerbate lung conditions.

```
7.  Temperature:
```

Temperature is the measure of heat in the environment and can influence air quality. High temperatures can enhance the formation of ozone and other pollutants, leading to more significant air quality problems, particularly in urban areas. Temperature fluctuations can also affect the dispersal of pollutants in the atmosphere.

These pollutants, when present at high concentrations, pose significant risks to human health and the environment.

6.

Prompt:

In R commander, how can I use foward fill to remove missing/NA values?

Response:

In R Commander, forward filling (replacing missing or NA values with the last non-missing value) can be done using R functions, but R Commander itself doesn't directly provide a built-in menu option for this. However, you can accomplish forward filling by using R scripts within R Commander.

Here's a step-by-step guide on how to apply forward fill to remove missing values:

Steps:

```
1.  Open R Commander and load your dataset.

2.  Create an R Script:

•   Go to the Tools menu and choose R Script (or just open the R Script tab).

•   In the R script window, you will write the following code to apply
forward filling.

3.  Forward Fill Using the zoo Package:
```

The zoo package has a function called na.locf() that is commonly used for forward filling.

First, you need to make sure the zoo package is installed. You can install it by running:

```
install.packages("zoo")
```

Once installed, you can load the package and apply forward fill to your dataset. Here's how to do it:

```
# Load the zoo package
library(zoo)

# Forward fill missing values in a specific column (replace "your_column"
with the actual column name)
your_data$your_column <- na.locf(your_data$your_column, na.rm = FALSE)

# If you want to forward fill all columns
your_data[] <- lapply(your_data, function(x) na.locf(x, na.rm = FALSE))
```

```
# View the result
head(your_data)

•   Explanation: The na.locf() function fills the NA values with the last
non-missing value in the data.

•   The lapply() function is used if you want to apply forward filling to all
columns in the dataset.

4.  Run the Script:

•   After writing the script, click Submit or Run to execute the code in the
R Console.

5.  Check the Data:

•   You can view the updated dataset in R Commander by going to the Data menu
and selecting Active data set.
```

This approach uses the zoo package and the na.locf() function to perform forward filling, which should work as expected.

7.

Prompt:

How can I implement a multi-panel plot on ggplot 2

Response:

To create a multi-panel plot in ggplot2, you can use the facet_wrap() or facet_grid() functions. These functions allow you to create multiple smaller plots (facets) based on one or more variables in your data.

Here's how you can use them:

1. Using facet_wrap()

facet_wrap() creates a set of subplots (panels) based on a single factor. The facets will be arranged in a wrapping grid.

Example:

```r
library(ggplot2)

# Example data
data(mpg)

# Multi-panel plot using facet_wrap()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~ class) +  # Facet by the 'class' variable
  labs(title = "Engine Displacement vs Highway MPG by Car Class",
       x = "Engine Displacement (L)",
       y = "Highway MPG")
```

Explanation:

```
•   facet_wrap(~ class) tells ggplot to create a separate plot for each level
of the class variable (such as compact, suv, midsize, etc.).

•   The plots will automatically wrap into multiple rows or columns depending
on the size of the plot.
```

2. Using facet_grid()

facet_grid() allows you to create a grid of plots based on two variables, one for rows and the other for columns.

Example:

```r
library(ggplot2)

# Example data
data(mpg)

# Multi-panel plot using facet_grid()
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(cyl ~ class) +  # Facet by 'cyl' (rows) and 'class' (columns)
  labs(title = "Engine Displacement vs Highway MPG by Cylinders and Car Class",
       x = "Engine Displacement (L)",
       y = "Highway MPG")
```

Explanation:

- facet_grid(cyl ~ class) will arrange the facets in a grid where rows correspond to the cyl variable (number of cylinders) and columns correspond to the class variable (car class).

- You can swap the order of cyl ~ class to place cyl in columns and class in rows.

Summary:

- Use facet_wrap() for a single variable with wrapping panels.

- Use facet_grid() for two variables, creating a grid of rows and columns.

8.

Prompt:

In brief, what does a high IQR of NO2 levels indicate during autumn and winter periods? Also, what does a very high IQR of O3 indicate for summer periods?

Response:

High IQR of NO2 Levels During Autumn and Winter:

A high IQR for NO2 during autumn and winter indicates significant variability in NO2 concentrations, likely due to fluctuating emissions from heating systems (more common in colder months), vehicular traffic, and weather patterns (e.g., temperature inversions). This suggests that NO2 levels can vary greatly within these seasons, with occasional peaks during high pollution events and lower levels on milder days.

Very High IQR of O3 Levels During Summer:

A very high IQR for O3 in the summer indicates significant variability in ozone levels, likely caused by sunlight-driven reactions between pollutants like NOx and volatile organic compounds (VOCs). During the summer, factors such as temperature and sunlight intensity can lead to large fluctuations in ozone formation, with higher concentrations on hotter, sunnier days and lower levels during cooler or cloudier days. This variability can signal inconsistent air quality, with spikes in ozone potentially affecting public health.

9.

Prompt:

In R commander, how do extract the top 3 rows based on one column’s values in a table?

Response:

In R Commander, extracting the top 3 rows based on one column’s values requires using R code since R Commander doesn’t have a direct menu option for this specific task. Here’s how you can do it:

Step-by-Step Instructions:

```
1.  Open R Commander: Ensure R Commander is open with your dataset loaded.

2.  Open the R Script Tab:

•   Navigate to the R Script tab at the top of the R Commander window. This is where you can enter custom R code.

3.  Enter R Code:
```

In the R Script tab, use the following code to extract the top 3 rows based on a specific column’s values. For this example, let’s assume the dataset is named your_data and the column of interest is your_column.

```r
top3_rows <- your_data[order(your_data$your_column, decreasing = TRUE),][1:3,]
```

Replace your_data with the actual name of your dataset and your_column with the actual column name you want to sort by.

This approach allows you to sort the data and extract the top 3 rows based on any numeric column in your dataset.

Description of How GenAI outputs were Integrated Into Our Analysis or Dashboard

Generative AI has helped us perform more in-depth analyses of our datasets with patterns or trends that might go unnoticed by many. It also assisted in resolving display and alignment issues, mainly when using Shiny. We used Gen AI to get further insight into analysis or descriptions, and it helped us debug different parts of our code. This integration also helped us to get a good insight into how certain coding tasks can be implemented.

Reflection on Effectiveness of using GenAI in our workflow

Generative AI has been great for solving specific challenges quickly, especially the display issues mentioned earlier. However, we encountered some problems along the way. The GenAI sometimes fell into hallucinations, where the model occasionally generated inaccurate or fabricated information. Also, on some occasions, when attempting to input a different prompt, the previous result was repeated instead of generating a new one. Another difficulty was inputting various keywords in our prompts to hint at what result we wished for. Without including these, the GenAI often went off on tangents and would output incomplete or irrelevant results.