Experiment No.6
Data Visualization using Hive.
Date of Performance:
Date of Submission:

Aim: Data Visualization using Hive.

Theory:

Hive has a fascinating history related to the world's largest social networking site: Facebook. Facebook adopted the Hadoop framework to manage their big data. If you have read our previous blogs, you would know that big data is nothing but massive amounts of data that cannot be stored, processed, and analyzed by traditional systems.

Architecture of Hive

The architecture of the Hive is as shown below. We start with the Hive client, who could be the programmer who is proficient in SQL, to look up the data that is needed.

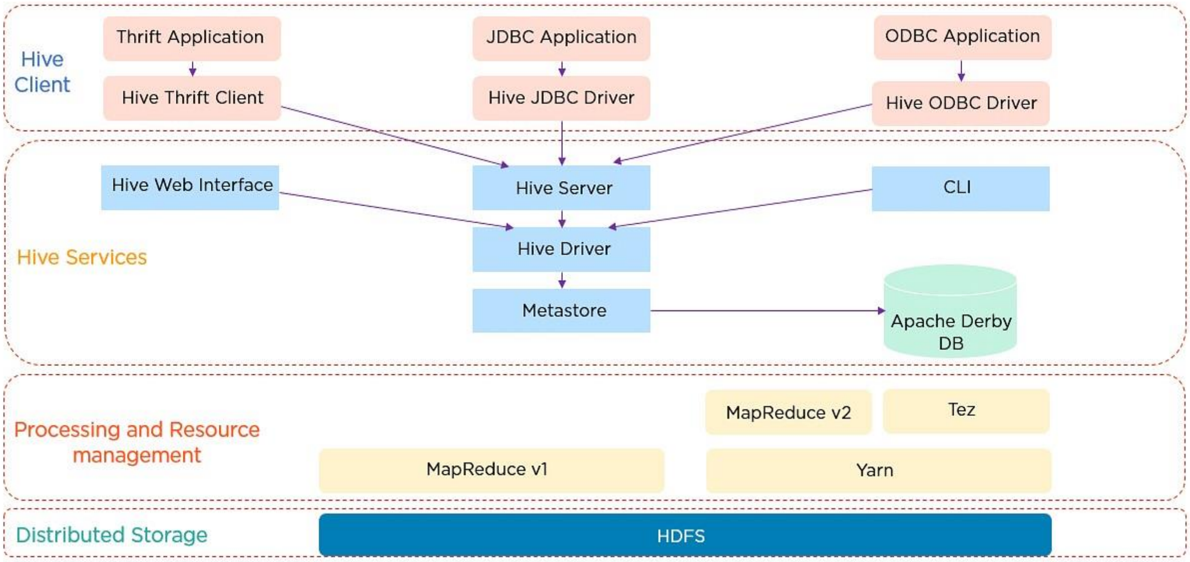


Fig: Architecture of Hive

The Hive client supports different types of client applications in different languages to perform queries. Thrift is a software framework. The Hive Server is based on Thrift, so it can serve requests from all of the programming languages that support Thrift.

The data flow in the following sequence:

1. We execute a query, which goes into the driver
2. Then the driver asks for the plan, which refers to the query execution
3. After this, the compiler gets the metadata from the metastore
4. The metastore responds with the metadata
5. The compiler gathers this information and sends the plan back to the driver
6. Now, the driver sends the execution plan to the execution engine
7. The execution engine acts as a bridge between the Hive and Hadoop to process the query

8. In addition to this, the execution engine also communicates bidirectionally with the metastore to perform various operations, such as create and drop tables
9. Finally, we have a bidirectional communication to fetch and send results back to the client

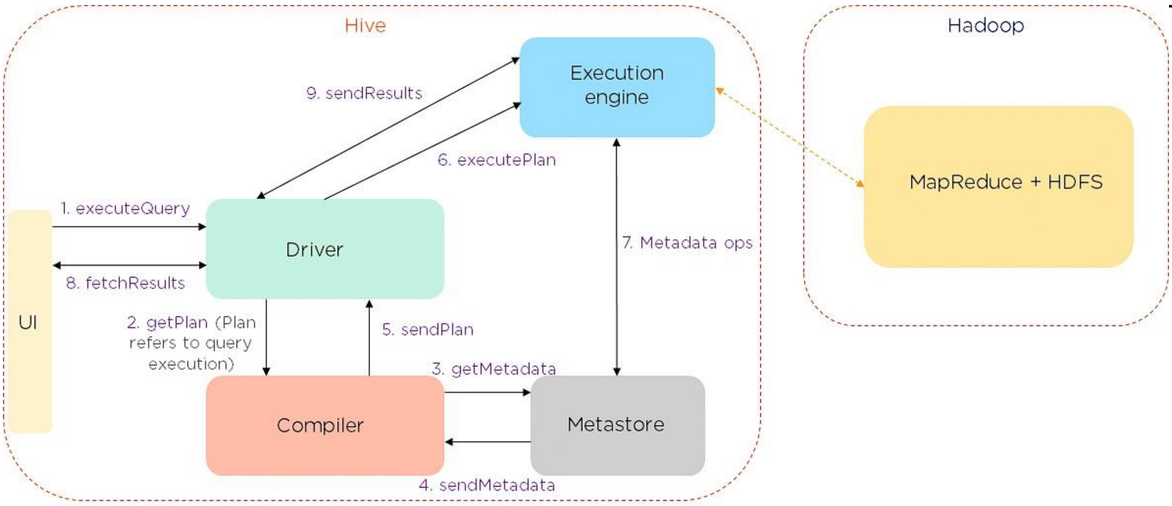


Fig: Data flow in Hive

Consider the following air quality dataset for visualization in R:

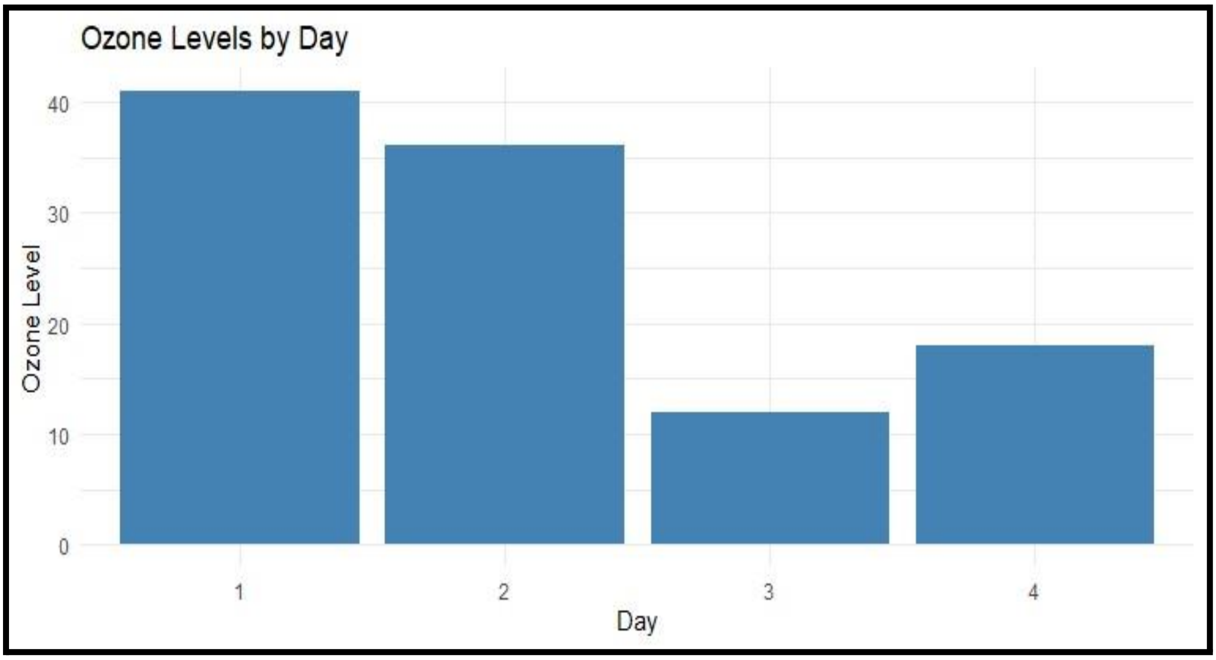
Ozone	Solar R.	Wind	Temp	Month	Day
41	190	7.4	67	5	1
36	118	8.0	72	5	2
12	149	12.6	74	5	3
18	313	11.5	62	5	4
NA	NA	14.3	56	5	5
28	NA	14.9	66	5	6

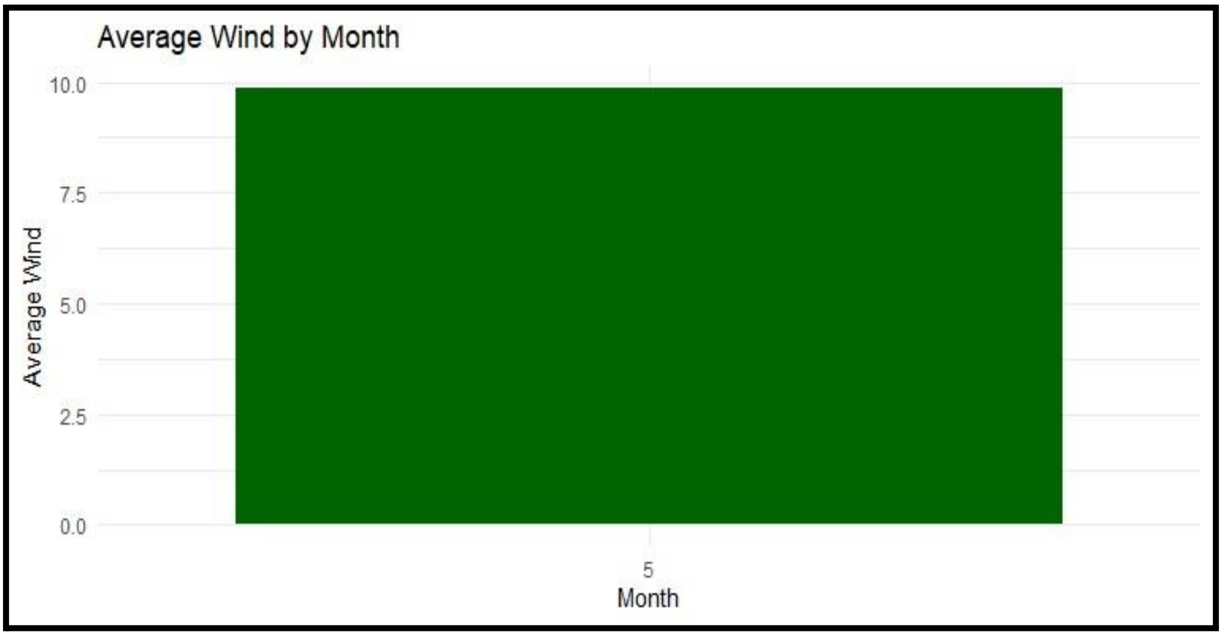
1] Bar Plot

Code:

```
# Install and load the ggplot2 package
if (!require("ggplot2")) install.packages("ggplot2")
library(ggplot2)
# Prepare and clean the data
data <- data.frame(
  Ozone   = c(41, 36, 12, 18, NA, 28),
  Solar_R = c(190, 118, 149, 313, NA, NA),
  Wind    = c(7.4, 8.0, 12.6, 11.5, 14.3, 14.9),
  Temp    = c(67, 72, 74, 62, 56, 66),
  Month   = c(5, 5, 5, 5, 5, 5),
  Day     = c(1, 2, 3, 4, 5, 6)
)
# Remove rows with missing values
data_clean <- na.omit(data)
# Bar Plot of Ozone by Day
ggplot(data_clean, aes(x = factor(Day), y = Ozone)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "Ozone Levels by Day",
       x = "Day", y = "Ozone Level") +
  theme_minimal()
```

Output:





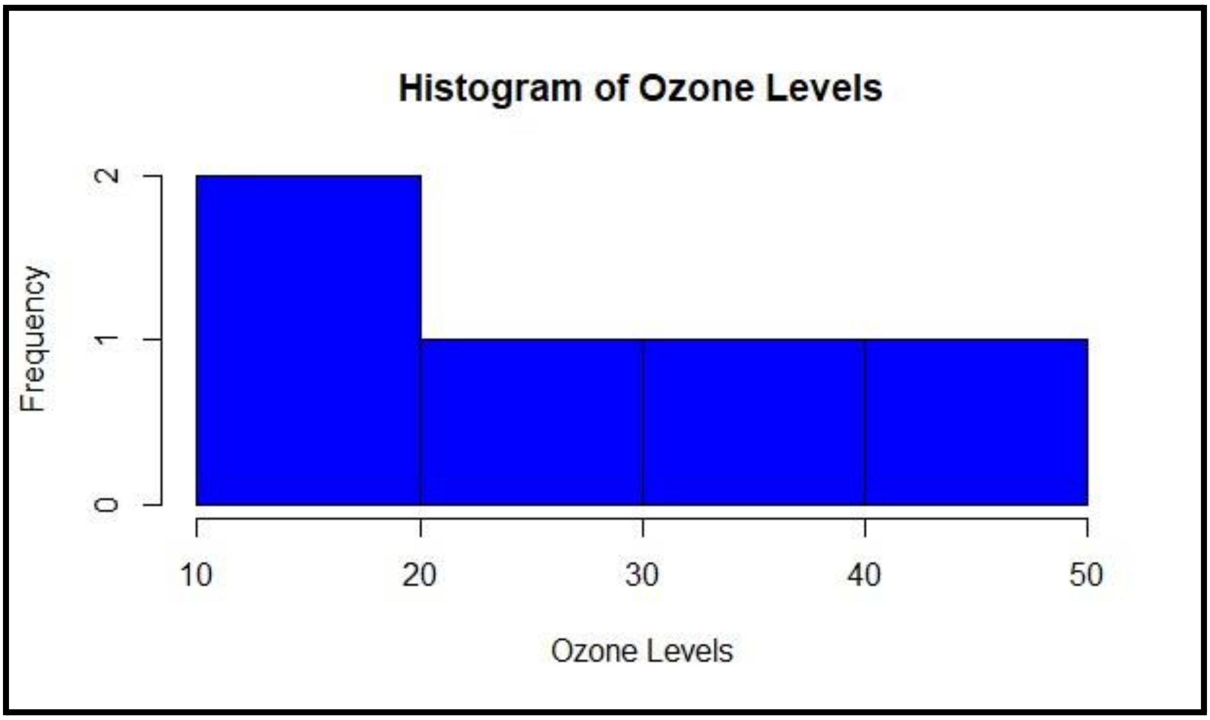
2]Histogram

Code:

```
# Histogram of Ozone levels
hist(na.omit(data$Ozone),
     main = 'Histogram of Ozone Levels',
     xlab = 'Ozone Levels',
```

```
    col = 'blue', border = 'black')
```

Output:

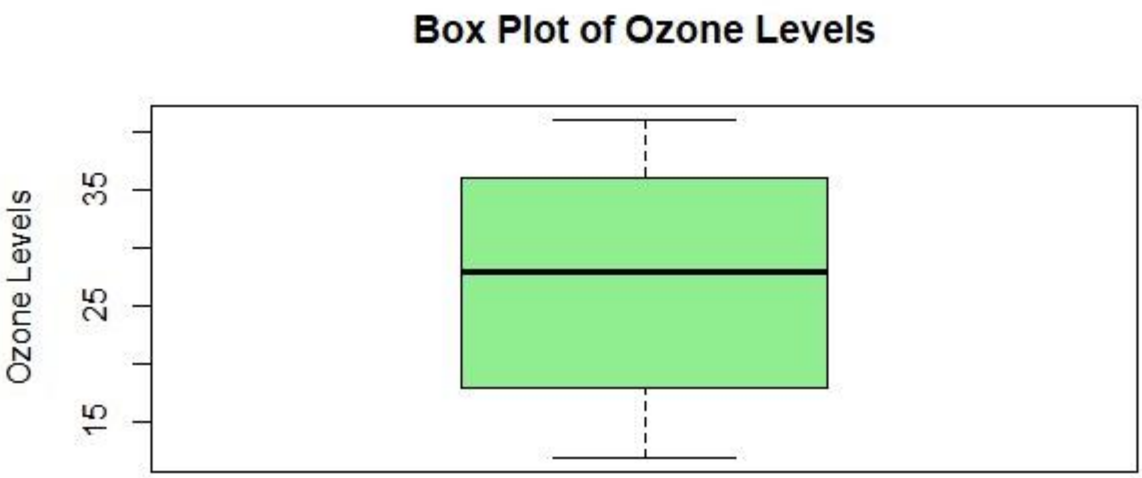


4] Box Plot

Code:

```
# Box plot of Ozone levels
boxplot(na.omit(data$Ozone),
    main = 'Box Plot of Ozone Levels',
    ylab = 'Ozone Levels',
    col = 'lightgreen')
```

Output:

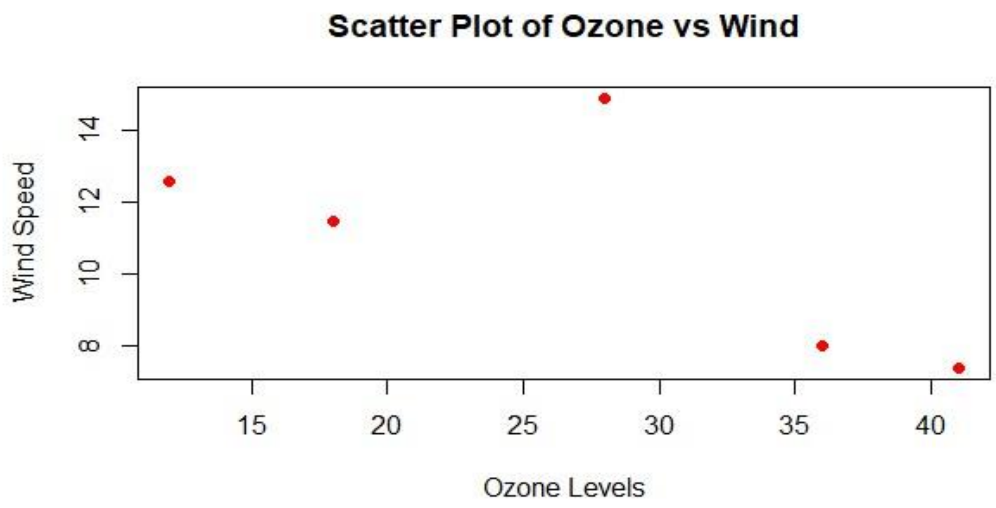


5] Scatter Plot

Code:

```
# Scatter plot of Ozone vs Wind
plot(data$Ozone, data$Wind,
     main = 'Scatter Plot of Ozone vs Wind',
     xlab = 'Ozone Levels',
     ylab = 'Wind Speed',
     pch = 19, col = 'red')
```

Output:

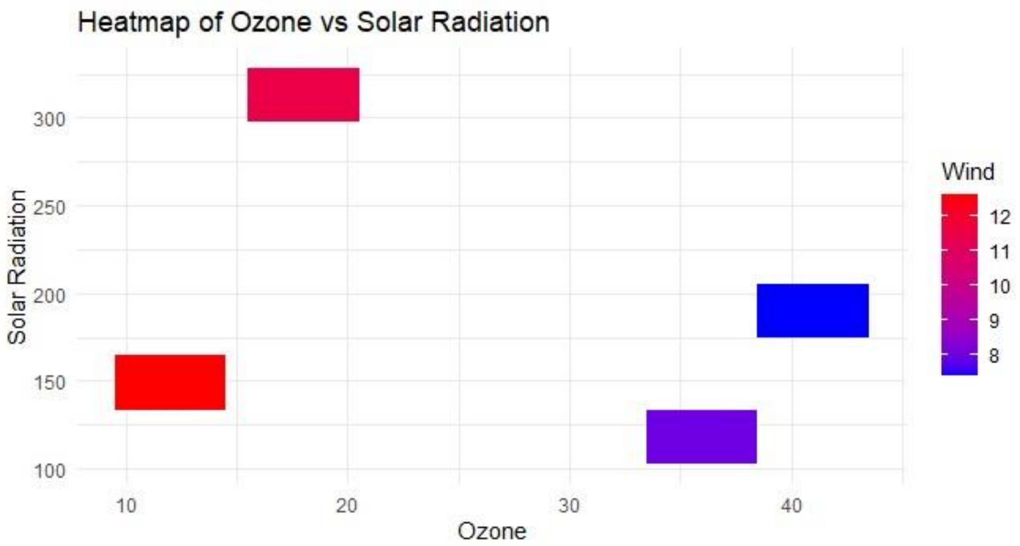


6] Heat Map

Code:

```
# Create the heatmap using ggplot2

ggplot(data_clean, aes(x = Ozone, y = Solar_R, fill = Wind)) +

  geom_tile() +

  scale_fill_gradient(low = "blue", high = "red") +

  labs(title = "Heatmap of Ozone vs Solar Radiation",

       x = "Ozone", y = "Solar Radiation", fill = "Wind") +

  theme_minimal()
```

Output:



7) Map Visualization

Code:

```
# Install and load required packages
if (!require("maps")) install.packages("maps")
library(maps)

# Map visualization of US states
map("state", fill = TRUE, col = rainbow(50), main = "Map of US States")
```

Output:

8. 3D Visualizations

a. Plotly 3D Scatter Plot

Code:

```
# Install and load necessary packages
if (!require("plotly")) install.packages("plotly", dependencies = TRUE)
library(plotly)

# Prepare the data (reuse data_clean)


# Plotly 3D Scatter Plot
plot_ly(data_clean, x = ~Ozone, y = ~Solar_R, z = ~Wind,
        type = "scatter3d", mode = "markers",
        marker = list(size = 5, color = ~Temp, colorscale = "Viridis")) %>%
  layout(scene = list(
    xaxis = list(title = 'Ozone'),
    yaxis = list(title = 'Solar Radiation'),
    zaxis = list(title = 'Wind')
  ))
```

b. Plotly 3D Surface Plot:

Code:

```
# Generate data for the surface plot
x <- seq(-5, 5, length.out = 50)
y <- seq(-5, 5, length.out = 50)
z <- outer(x, y, function(x, y) sin(sqrt(x^2 + y^2)))

# Plotly 3D Surface Plot
plot_ly(x = x, y = y, z = z, type = "surface")
```

c. RGL 3D Scatter Plot:

Code:

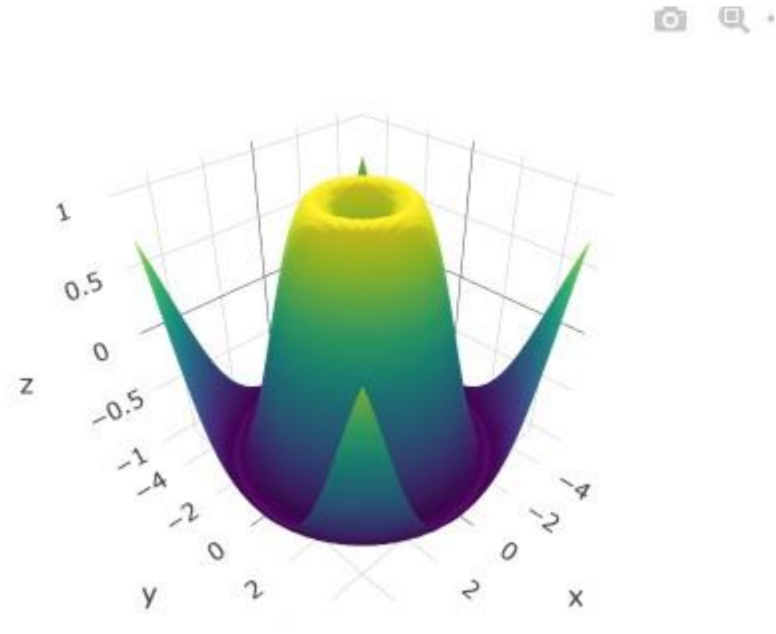
```
# Install and load necessary packages
if (!require("rgl")) install.packages("rgl", dependencies = TRUE)
library(rgl)

# RGL 3D Scatter Plot
open3d()
plot3d(data_clean$Ozone, data_clean$Solar_R, data_clean$Wind,
    col = rainbow(nrow(data_clean)), size = 5, type = "s",
    xlab = "Ozone", ylab = "Solar Radiation", zlab = "Wind")
```

d. RGL 3D Surface Plot

Code:

```
# Generate data for the surface plot
x <- seq(-5, 5, length.out = 50)
y <- seq(-5, 5, length.out = 50)
z <- outer(x, y, function(x, y) sin(sqrt(x^2 + y^2)))

# RGL 3D Surface Plot
open3d()
surface3d(x, y, z, color = "lightblue", back = "lines")
```

Output:



CONCLUSION:

In this experiment, data visualizations using R were created to analyze air quality parameters like ozone levels, wind speed, and solar radiation. Visualizations such as bar plots, scatter plots, histograms, and 3D plots helped in understanding the dataset better.

Datatypes in Hive:

Hive supports various data types for handling structured data:

1. Primitive Data Types:
   - Numeric: TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, DECIMAL.
   - String: STRING, VARCHAR, CHAR.
   - Date/Time: DATE, TIMESTAMP, INTERVAL.
   - Boolean: BOOLEAN.
2. Complex Data Types:
   - ARRAY: Ordered collection of elements (e.g., ARRAY<INT>).
   - MAP: Key-value pairs (e.g., MAP<STRING, INT>).
   - STRUCT: Grouping of fields (e.g., STRUCT<name: STRING, age: INT>).
   - UNIONTYPE: Allows multiple data types (e.g., UNIONTYPE<INT, STRING>).