Take-Home Exam One

Noah Gallego

2024-10-19

Import Libraries

```
library(dplyr)      # For data manipulation
library(ggplot2)    # For plotting
library(ggrepel)    # For labeling plots
library(factoextra) # For Scree Plot and K-means visualization
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WB
a
```

```
library(readxl)     # For reading Excel files
library(lubridate)  # For handling date-related operations
library(maps)       # For map data
```

```
##
## Attaching package: 'maps'
```

```
## The following object is masked from 'package:purrr':
##
##     map
```

```
library(gganimate)  # For animating polygons
library(cluster)    # For clustering analysis
```

```
##
## Attaching package: 'cluster'
```

```
## The following object is masked from 'package:maps':
##
##     votes.repub
```

Read-In Data

```
# Load the datasets required for analysis
student_data = read_excel("student_performance_missing.xlsx")
stocks_data = read.table("stocks2.txt", header = TRUE)
airbnb_data = read.csv("airbnb.csv")
```

Question One: Summary Function

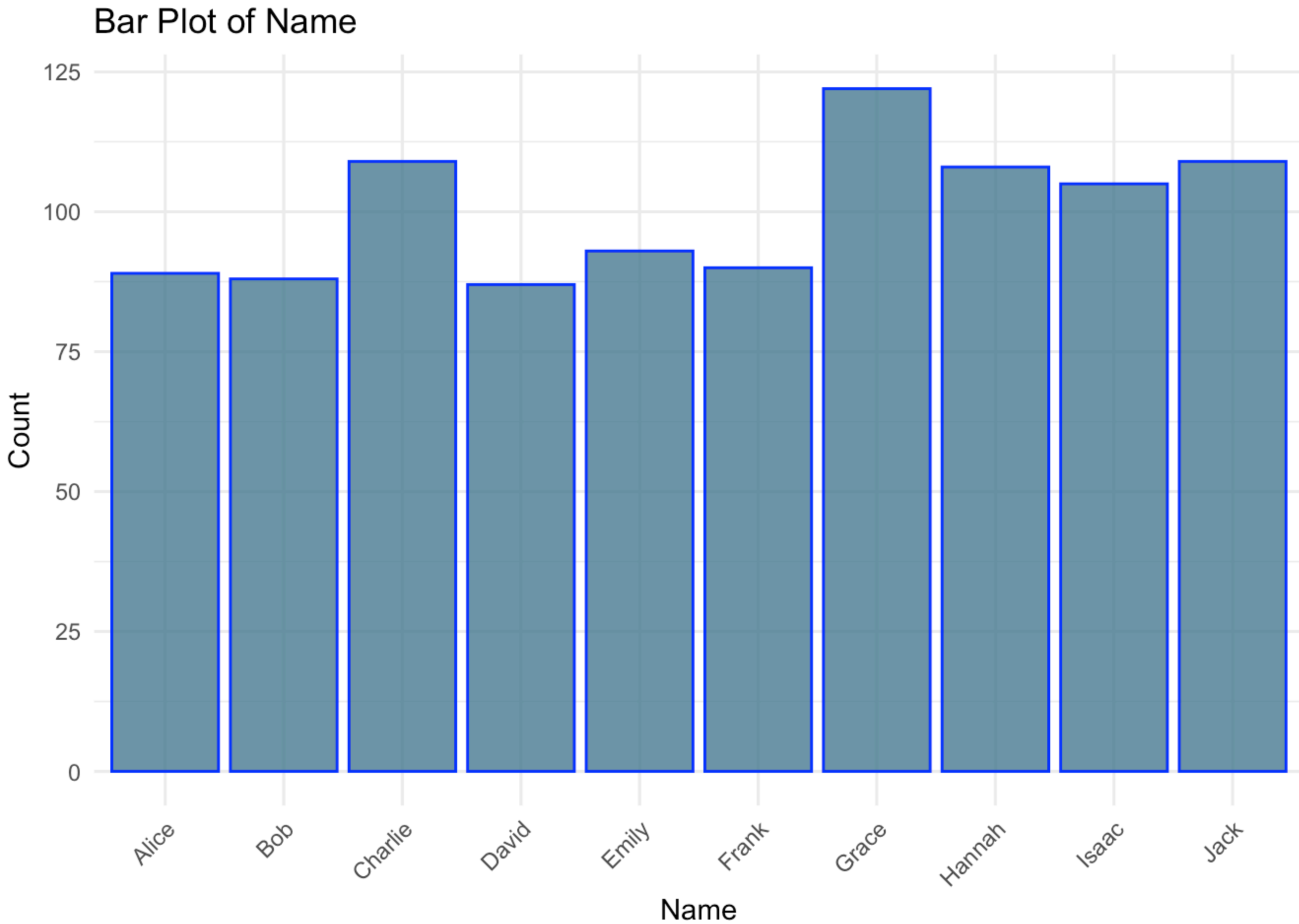
Define Summary Function

```
# Function to summarize data frame
# Inputs: df - data frame to summarize
# Outputs: Class information, summary statistics, and bar plots for categorical variable
s

display_results = function(df) {
  for (col_name in names(df)) {
    col_data = df[[col_name]]  # Get the column data

    # Find class of each variable of the data frame
    cat("The class of", col_name, "is:", class(col_data), "\n")

    if (is.numeric(col_data)) {
      # Impute missing values with the mean if any are detected
      if (any(is.na(col_data))) {
        cat("Missing values detected in", col_name, "- imputing with mean.\n")
        df[[col_name]][is.na(col_data)] = mean(col_data, na.rm = TRUE)
      }

      # Print summary statistics for numeric columns
      cat("\nSummary statistics for", col_name, ":\n")
      cat("Mean:", mean(col_data, na.rm = TRUE), "\n")
      cat("Median:", median(col_data, na.rm = TRUE), "\n")
      cat("Variance:", var(col_data, na.rm = TRUE), "\n")
      cat("IQR:", IQR(col_data, na.rm = TRUE), "\n")
      cat("Standard Deviation:", sd(col_data, na.rm = TRUE), "\n\n")

    } else if (is.factor(col_data) || is.character(col_data)) {
      # Create a bar plot for categorical variables
      plot = ggplot(data = df, aes(x = col_data)) +
        geom_bar(color = "blue", fill = rgb(0.1, 0.4, 0.5, 0.7)) +
        labs(title = paste("Bar Plot of", col_name), x = col_name, y = "Count") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 45, hjust = 1))

      print(plot)
    }
  }
}

# Test the Function with Modified Student Data
# Convert Exam_Score to numeric before running the function
copy = student_data
copy$Exam_Score = as.numeric(copy$Exam_Score)
```

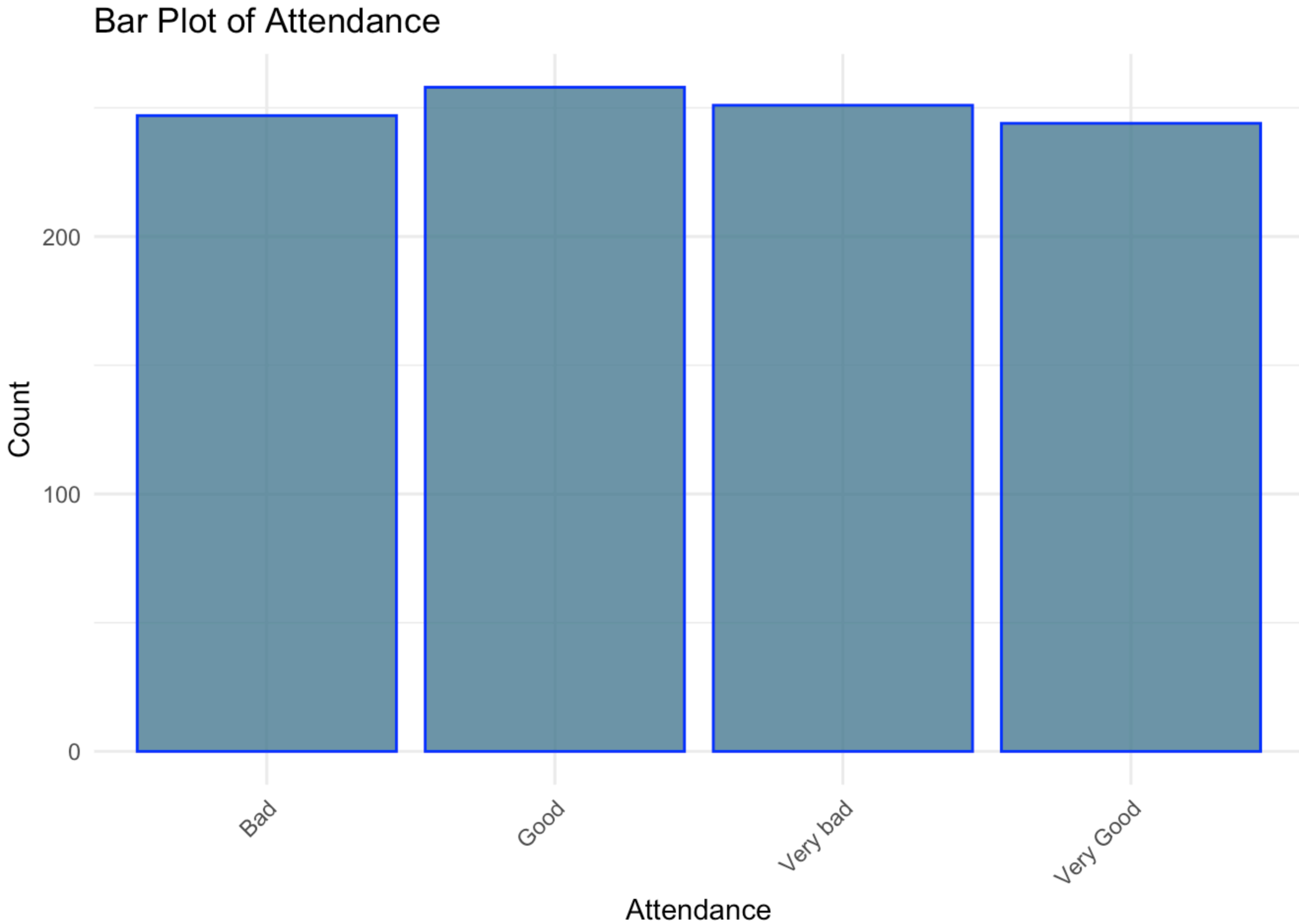
```
## Warning: NAs introduced by coercion
```

```
display_results(copy)
```

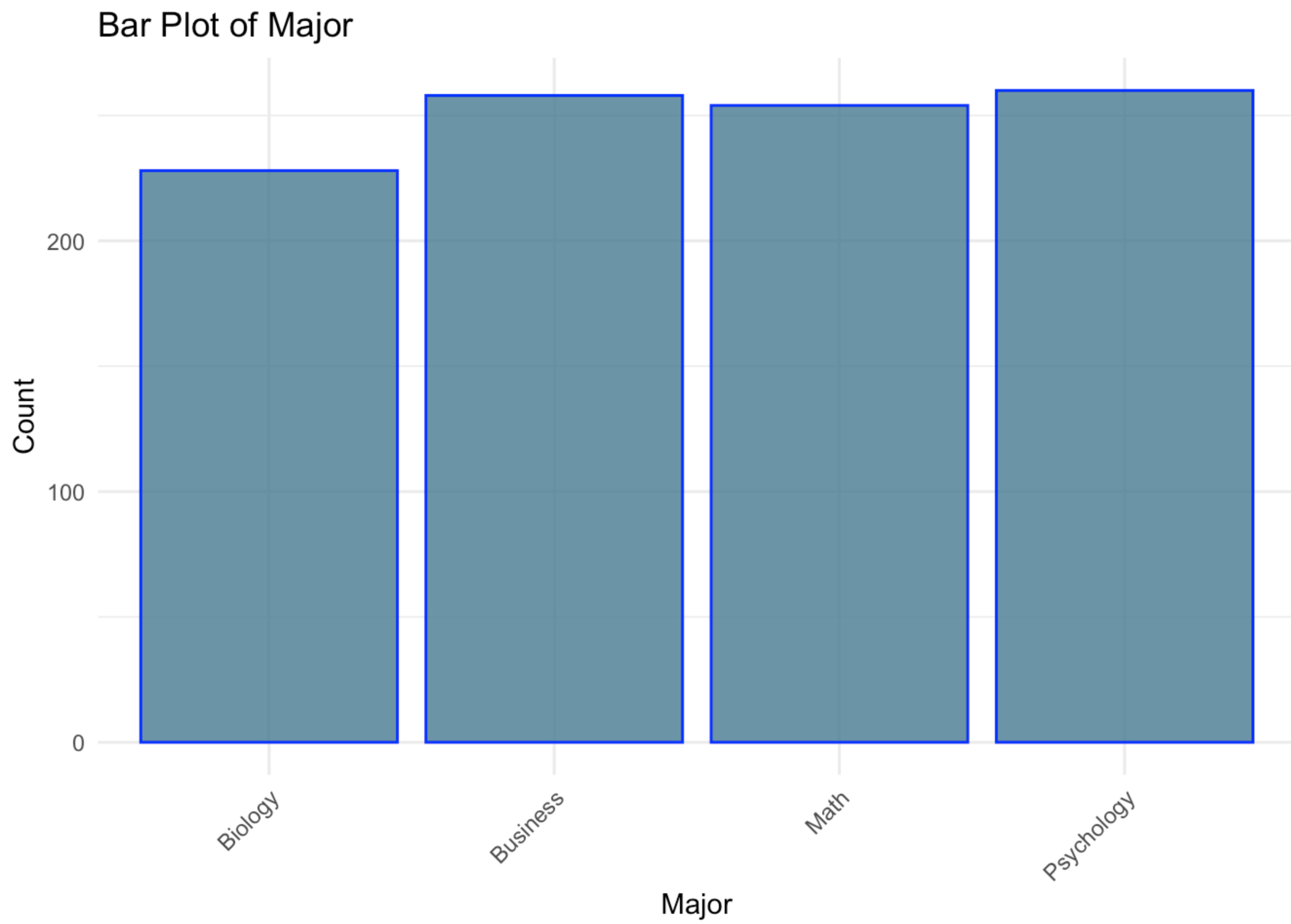
```
## The class of Name is: character
```




```
## The class of Attendance is: character
```




```
## The class of Exam_Score is: numeric
## Missing values detected in Exam_Score - imputing with mean.
##
## Summary statistics for Exam_Score :
## Mean: 75.28557
## Median: 80
## Variance: 278.5613
## IQR: 30
## Standard Deviation: 16.69016
##
## The class of Study_Time is: numeric
##
## Summary statistics for Study_Time :
## Mean: 4.902639
## Median: 5.057341
## Variance: 5.036645
## IQR: 3.965244
## Standard Deviation: 2.244247
##
## The class of Major is: character
```



Question Two: Vector Conversion Function

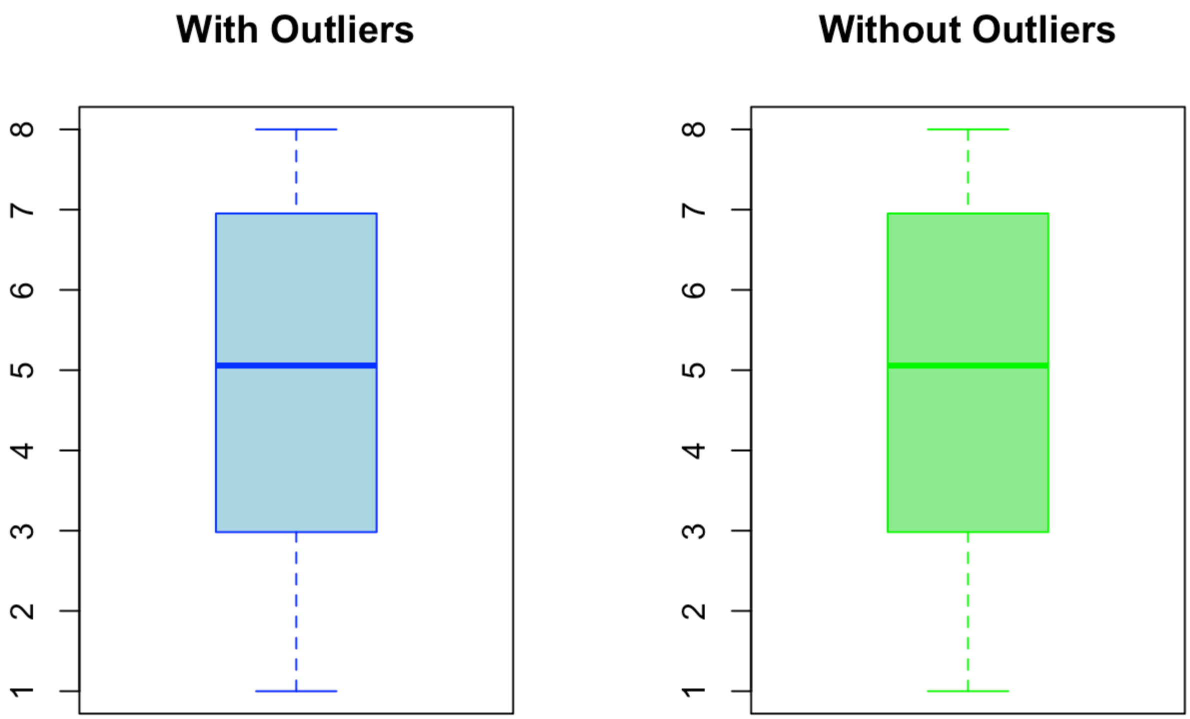
Define Conversion Function

```
# Function to remove outliers and draw boxplots
# Inputs: col - vector to analyze
# Outputs: Boxplots of original data vs. data without outliers

convert_vector = function(col) {
  # Check whether col is numeric or not
  if (!is.numeric(col)) {
    cat("The vector", names(col), "is not numeric.")
    return()
  } else {
    # Remove outliers using the IQR method
    q1 = quantile(col, 0.25, na.rm = TRUE)
    q3 = quantile(col, 0.75, na.rm = TRUE)
    iqr_val = IQR(col, na.rm = TRUE)

    lower_bound = q1 - 1.5 * iqr_val
    upper_bound = q3 + 1.5 * iqr_val

    filtered_col = col[col >= lower_bound & col <= upper_bound]

    # Draw boxplots: with and without outliers
    par(mfrow = c(1, 2))
    boxplot(col, main = "With Outliers", col = "lightblue", border = "blue")
    boxplot(filtered_col, main = "Without Outliers", col = "lightgreen", border = "gree
n")
  }
}

# Test the Function
convert_vector(student_data$Attendance)
```

```
## The vector is not numeric.
```

```
## NULL
```

```
convert_vector(student_data$Study_Time)
```



Question Three: Stock Data Analysis

Data Cleaning and Summary Statistics

```
# Examine Stock Data
head(stocks_data)
```

	Date	AMZN	DUK	KO
	<chr>	<chr>	<chr>	<chr>
1	1/3/2007	38.700	34.971	17.875
2	1/4/2007	38.900	35.044	17.882
3	1/5/2007	38.370	34.240	17.757
4	1/8/2007	miss	miss	miss
5	1/9/2007	37.780	34.131	17.886
6	1/10/2007	37.150	33.984	17.912

6 rows

```
suppressWarnings({
  # Correct data types for stock data
  stocks_data$Date = as.Date(stocks_data$Date, format = "%m/%d/%Y")
  dates = stocks_data$Date

  # Convert all columns (except Date) to numeric
  stocks_data[, 2:ncol(stocks_data)] = stocks_data[, 2:ncol(stocks_data)] %>%
    mutate_all(as.numeric)
  stocks_data$Date = dates
})

# Deal with NA's by replacing with median
stocks_data[, 2:ncol(stocks_data)] = stocks_data[, 2:ncol(stocks_data)] %>%
  mutate_all(~ replace(., is.na(.), median(., na.rm = TRUE)))

# Calculate summary statistics
cat("AMZN Five Number Summary: \n")
```

```
## AMZN Five Number Summary:
```

```
summary(stocks_data$AMZN)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   35.03   89.42  206.78  258.90  332.61  844.36
```

```
cat("\nDUK Five Number Summary: \n")
```

```
##
## DUK Five Number Summary:
```

```
summary(stocks_data$DUK)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   24.09   35.79   50.23   50.64   63.86   84.44
```

```
cat("\nKO Five Number Summary: \n")
```

```
##
## KO Five Number Summary:
```

```
summary(stocks_data$KO)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   14.70   21.77   29.23   29.68   37.26   45.37
```

Days with High Closing Prices

```
# Calculate number of days with closing price > 20% higher than the mean
stocks_data[, 2:ncol(stocks_data)] %>%
  summarise_all(~ sum(. > 1.2 * mean(., na.rm = TRUE))) -> days_higher_than_mean

cat("Days higher than the mean: (AMZN, DUK, KO)\n")
```

```
## Days higher than the mean: (AMZN, DUK, KO)
```

```
print(days_higher_than_mean)
```

```
##   AMZN DUK  KO
## 1  745 790 814
```

Calculate Daily Returns

```
# Calculate daily returns for each company
returns = list()

for (i in 2:ncol(stocks_data)) {
  return_vector = numeric(nrow(stocks_data) - 1)

  for (j in 2:nrow(stocks_data)) {
    return_vector[j - 1] = (stocks_data[j, i] - stocks_data[j - 1, i]) / stocks_data[j - 
1, i]
  }

  returns[[colnames(stocks_data)[i]]] = return_vector
}

return_df = as.data.frame(returns)
head(return_df, 10)
```

	AMZN <dbl>	DUK <dbl>	KO <dbl>
1	0.005167959	0.002087444	0.0003916084
2	-0.013624679	-0.022942586	-0.0069902695
3	4.389106072	0.466910047	0.6463929718
4	-0.817293742	-0.320465088	-0.3881990764
5	-0.016675490	-0.004306935	0.0014536509
6	0.006729475	0.002707156	0.0012282269

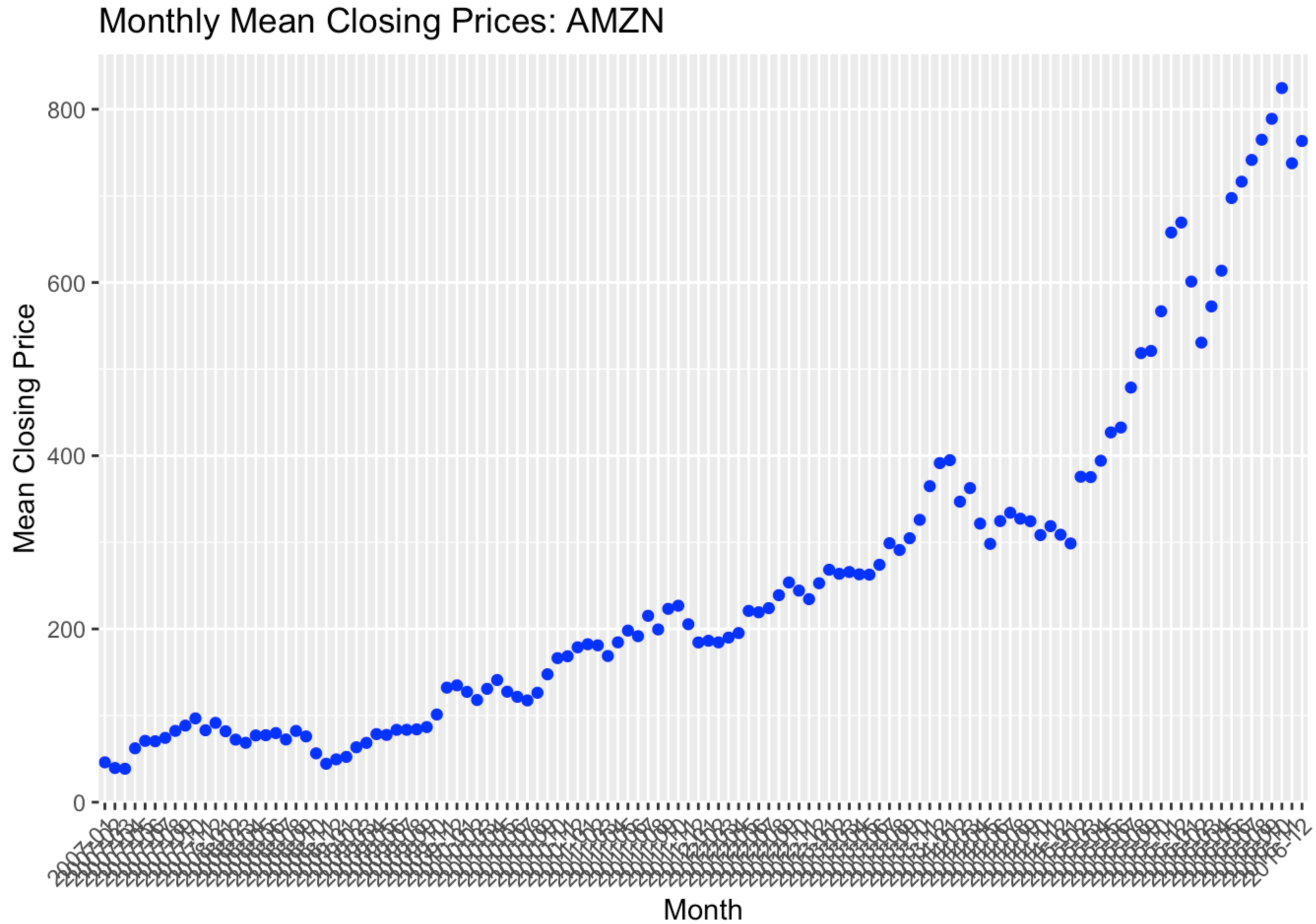
	AMZN <dbl>	DUK <dbl>	KO <dbl>
7	0.021390374	-0.004842118	-0.0039032006
8	0.012041885	0.003243785	-0.0010635916
9	-0.020175892	-0.001616649	0.0020734099
10	-0.023759240	0.004857799	-0.0051448384

1-10 of 10 rows

Monthly Mean Closing Prices

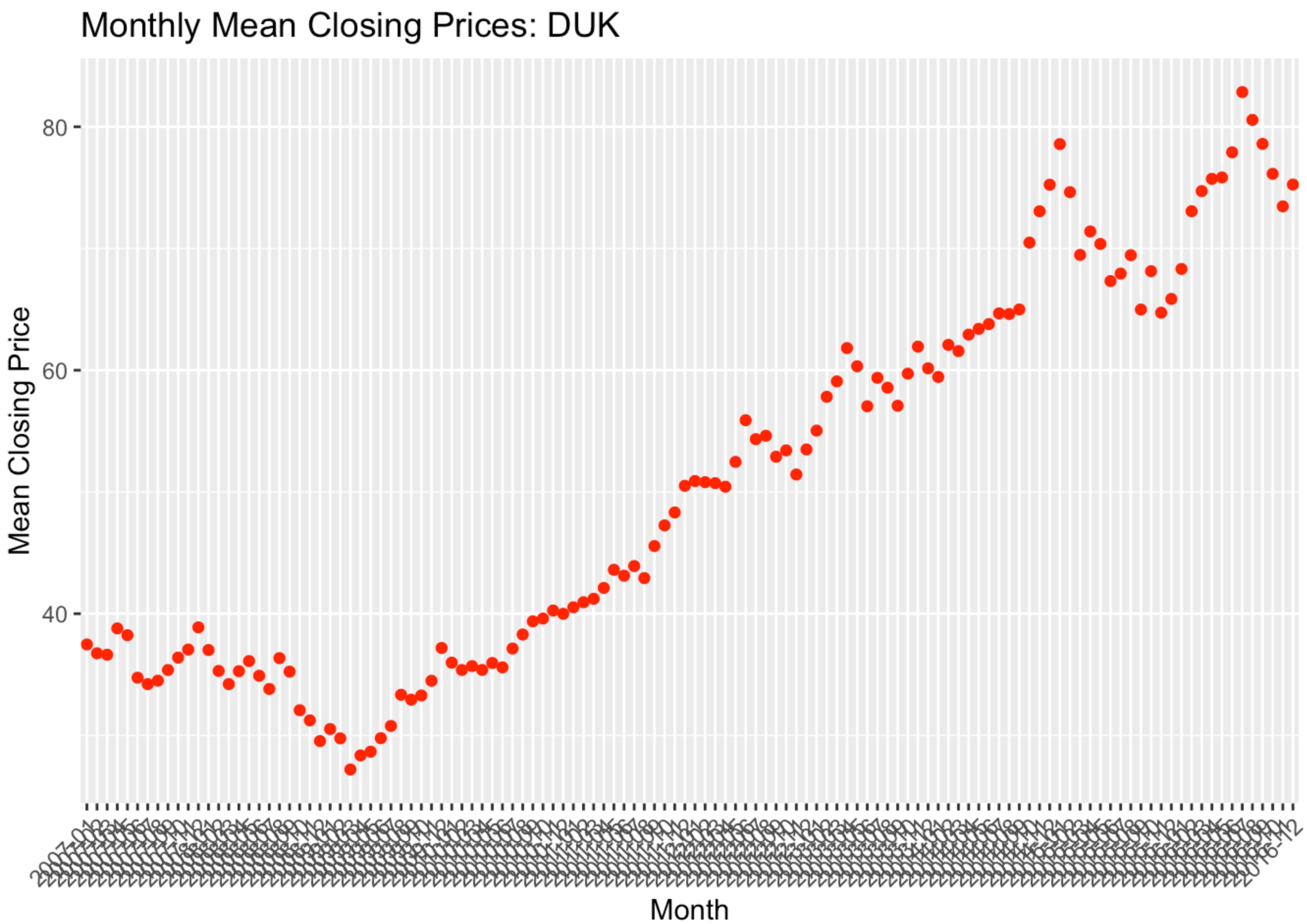
```
# Extract month and calculate monthly averages for each company
stocks_data$Month = format(stocks_data$Date, "%Y-%m")

monthly_means = stocks_data %>%
  group_by(Month) %>%
  summarise(
    monthly_AMZN = mean(AMZN, na.rm = TRUE),
    monthly_DUK = mean(DUK, na.rm = TRUE),
    monthly_KO = mean(KO, na.rm = TRUE)
  )

# Plot monthly mean prices for each company
## Amazon (AMZN)
ggplot(monthly_means, aes(x = Month, y = monthly_AMZN)) +
  geom_line(color = "blue") +
  geom_point(color = "blue") +
  labs(title = "Monthly Mean Closing Prices: AMZN", x = "Month", y = "Mean Closing Pric
e") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
```



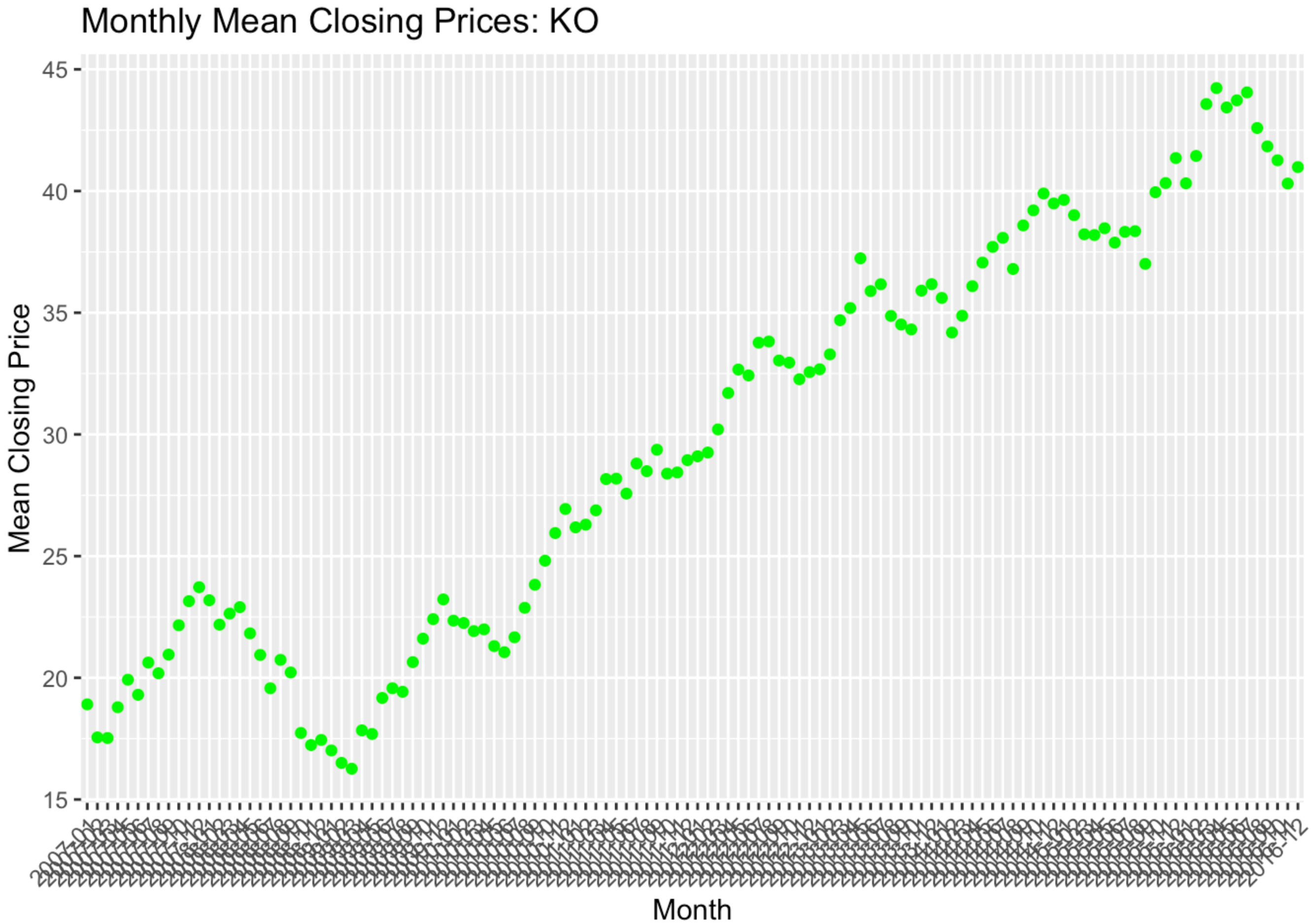

```
## Duke Energy (DUK)
ggplot(monthly_means, aes(x = Month, y = monthly_DUK)) +
  geom_line(color = "red") +
  geom_point(color = "red") +
  labs(title = "Monthly Mean Closing Prices: DUK", x = "Month", y = "Mean Closing Pric
e") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
```




```
## Coca-Cola (KO)
ggplot(monthly_means, aes(x = Month, y = monthly_KO)) +
  geom_line(color = "green") +
  geom_point(color = "green") +
  labs(title = "Monthly Mean Closing Prices: KO", x = "Month", y = "Mean Closing Price")
+
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
```



Question Four: AirBNB Data Analysis

Data Inspection and Cleaning

```
# Inspect AirBNB Data Structure
str(airbnb_data, 2)
```

```
## 'data.frame':    1275 obs. of  13 variables:
##  $ room_id             : int  5453 5506 6695 6976 8789 8792 9273 9765 9824 9827 ...
##  $ host_id             : int  8021 8229 8229 16701 26988 26988 4804 25188 25188 25188
...
##  $ room_type           : chr  "Private room" "Private room" "Entire home/apt" "Privat
e room" ...
##  $ neighborhood        : chr  "Jamaica Plain" "Roxbury" "Roxbury" "Roslindale" ...
##  $ reviews             : int  53 30 39 26 1 11 8 5 18 8 ...
##  $ overall_satisfaction: num  5 4.5 5 5 5 4.5 5 4.5 4 4.5 ...
##  $ accommodates        : int  2 2 4 2 2 3 4 2 2 3 ...
##  $ bedrooms            : int  1 1 1 1 1 1 2 1 1 1 ...
##  $ price               : int  171 165 222 74 165 228 257 319 212 330 ...
##  $ minstay             : int  1 3 3 1 5 5 3 3 2 3 ...
##  $ latitude            : num  42.3 42.3 42.3 42.3 42.4 ...
##  $ longitude           : num  -71.1 -71.1 -71.1 -71.1 -71.1 ...
##  $ last_modified       : chr  "33:58.0" "03:21.1" "14:31.7" "27:09.3" ...
```

```
# Drop unnecessary columns
airbnb_data = subset(airbnb_data, select = -c(latitude, longitude, last_modified))
```

Guest Accommodation Analysis

```
# Filter for 'Entire home/apt' and calculate accommodation statistics
apt_home_df = airbnb_data[airbnb_data$room_type == 'Entire home/apt', ]

# Calculate average and maximum accommodation capacity
guests_mean = mean(apt_home_df$accommodates, na.rm = TRUE)
guests_max = max(apt_home_df$accommodates, na.rm = TRUE)

cat("Average Accommodation for Entire Home/Apt:", guests_mean, "\n")
```

```
## Average Accommodation for Entire Home/Apt: 3.593709
```

```
cat("Maximum Accommodation for Entire Home/Apt:", guests_max, "\n")
```

```
## Maximum Accommodation for Entire Home/Apt: 12
```

Neighborhood Satisfaction

```
# Calculate average satisfaction by neighborhood and display top 10
osat = airbnb_data %>%
  group_by(neighborhood) %>%
  summarise(avg = mean(overall_satisfaction, na.rm = TRUE)) %>%
  arrange(desc(avg))

head(osat, 10)
```

neighborhood <chr>	avg <dbl>
Leather District	4.875000
South Boston Waterfront	4.833333
Chinatown	4.812500
Roslindale	4.788462
Jamaica Plain	4.745968
South End	4.726852
Charlestown	4.700000
Roxbury	4.697917
South Boston	4.695122
North End	4.682692

1-10 of 10 rows

Question 5: EPA Air Data Analysis

Load Pollutant Data

```
folder_path = "Air_Data/"

# List all the CSV files for each pollutant (Ozone, SO2, CO, NO2)
file_list_CO = list.files(path = folder_path, pattern = "daily_42101_[0-9]{4}.csv", ful
l.names = TRUE)   # CO
file_list_SO2 = list.files(path = folder_path, pattern = "daily_42401_[0-9]{4}.csv", ful
l.names = TRUE)  # SO2
file_list_NO2 = list.files(path = folder_path, pattern = "daily_42602_[0-9]{4}.csv", ful
l.names = TRUE)  # NO2
file_list_Ozone = list.files(path = folder_path, pattern = "daily_44201_[0-9]{4}.csv", f
ull.names = TRUE)  # Ozone

# Function to load and extract required columns
load_pollutant_data = function(files, pollutant_name) {
  pollutant_data = lapply(files, function(file) {
    data = read.csv(file, header = TRUE)
    data %>%
      select(Date.Local, State.Name, County.Name, Arithmetic.Mean) %>%  # Extract the re
quired columns
      rename(State = State.Name, County = County.Name, !!pollutant_name := Arithmetic.Me
an)
  })
  bind_rows(pollutant_data)
}

# Load and combine data for each pollutant
combined_data_CO = load_pollutant_data(file_list_CO, "CO")
combined_data_SO2 = load_pollutant_data(file_list_SO2, "SO2")
combined_data_NO2 = load_pollutant_data(file_list_NO2, "NO2")
combined_data_Ozone = load_pollutant_data(file_list_Ozone, "Ozone")

# Combine the pollutants into a single DataFrame by matching on Date, State, and County
combined_data = combined_data_Ozone %>%
  full_join(combined_data_SO2, by = c("Date.Local", "State", "County")) %>%
  full_join(combined_data_CO, by = c("Date.Local", "State", "County")) %>%
  full_join(combined_data_NO2, by = c("Date.Local", "State", "County"))
```

```
## Warning in full_join(., combined_data_SO2, by = c("Date.Local", "State", : Detected a
n unexpected many-to-many relationship between `x` and `y`.
## ℹ Row 1527 of `x` matches multiple rows in `y`.
## ℹ Row 58 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
## Warning in full_join(., combined_data_CO, by = c("Date.Local", "State", : Detected an
unexpected many-to-many relationship between `x` and `y`.
## ℹ Row 1527 of `x` matches multiple rows in `y`.
## ℹ Row 1212 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
## Warning in full_join(., combined_data_NO2, by = c("Date.Local", "State", : Detected a
n unexpected many-to-many relationship between `x` and `y`.
## ℹ Row 1527 of `x` matches multiple rows in `y`.
## ℹ Row 1 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
combined_data = combined_data %>%
  rename(Date = Date.Local)
```

Group and Summarize Pollutant Data

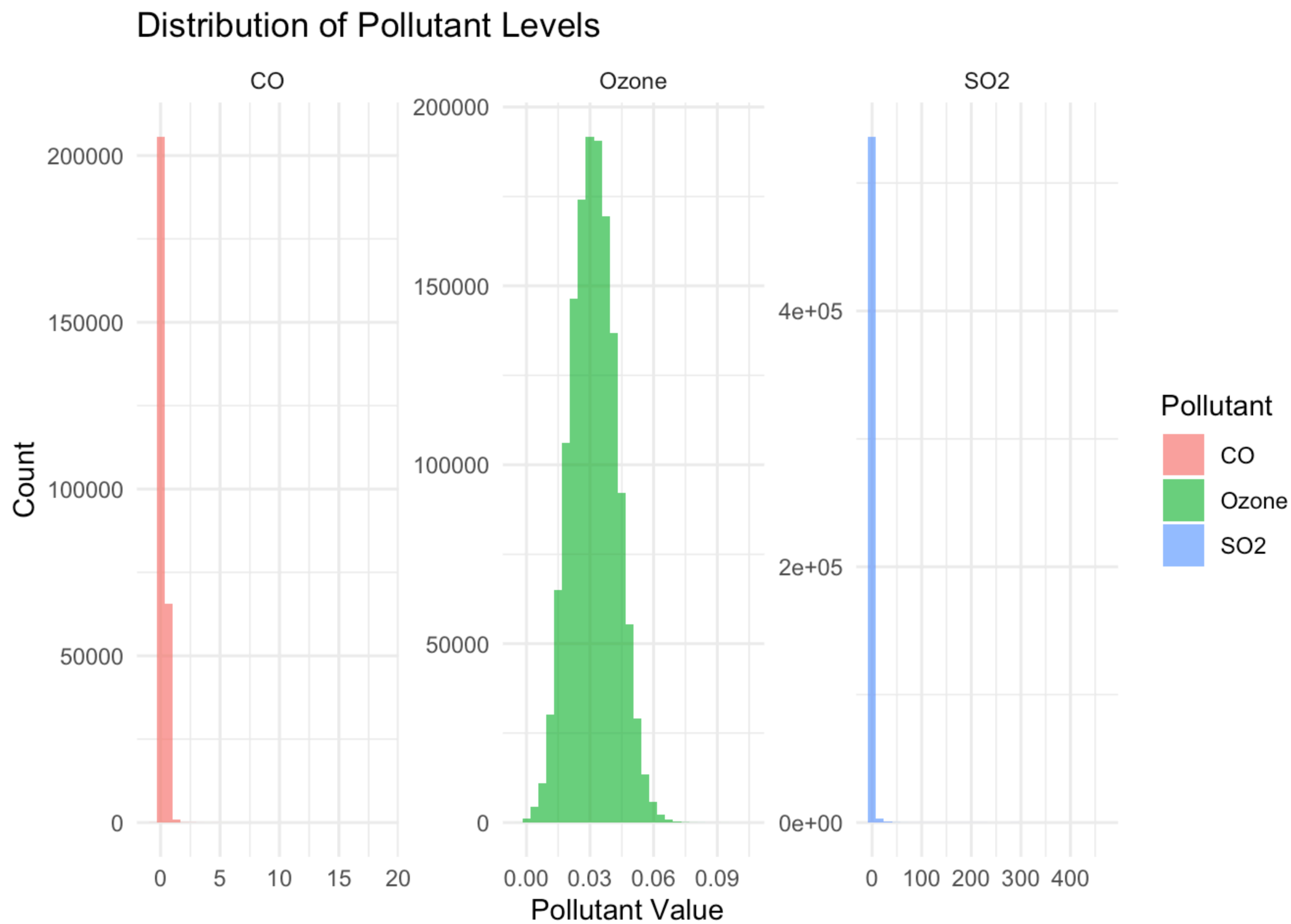
```
combined_data_cleaned = combined_data %>%
  group_by(Date, State, County) %>%
  summarise(
    Ozone = mean(Ozone, na.rm = TRUE),
    SO2 = mean(SO2, na.rm = TRUE),
    CO = mean(CO, na.rm = TRUE),
    NO2 = mean(NO2, na.rm = TRUE)
  )
```

```
## `summarise()` has grouped output by 'Date', 'State'. You can override using the
## `.groups` argument.
```

Visualize Distribution of Pollutant Levels

```
combined_data_cleaned %>%
  gather(Pollutant, Value, Ozone:CO) %>%
  ggplot(aes(x = Value, fill = Pollutant)) +
  geom_histogram(bins = 30, alpha = 0.7, position = "identity") +
  facet_wrap(~ Pollutant, scales = "free") +
  theme_minimal() +
  labs(title = "Distribution of Pollutant Levels", x = "Pollutant Value", y = "Count")
```

```
## Warning: Removed 2571607 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



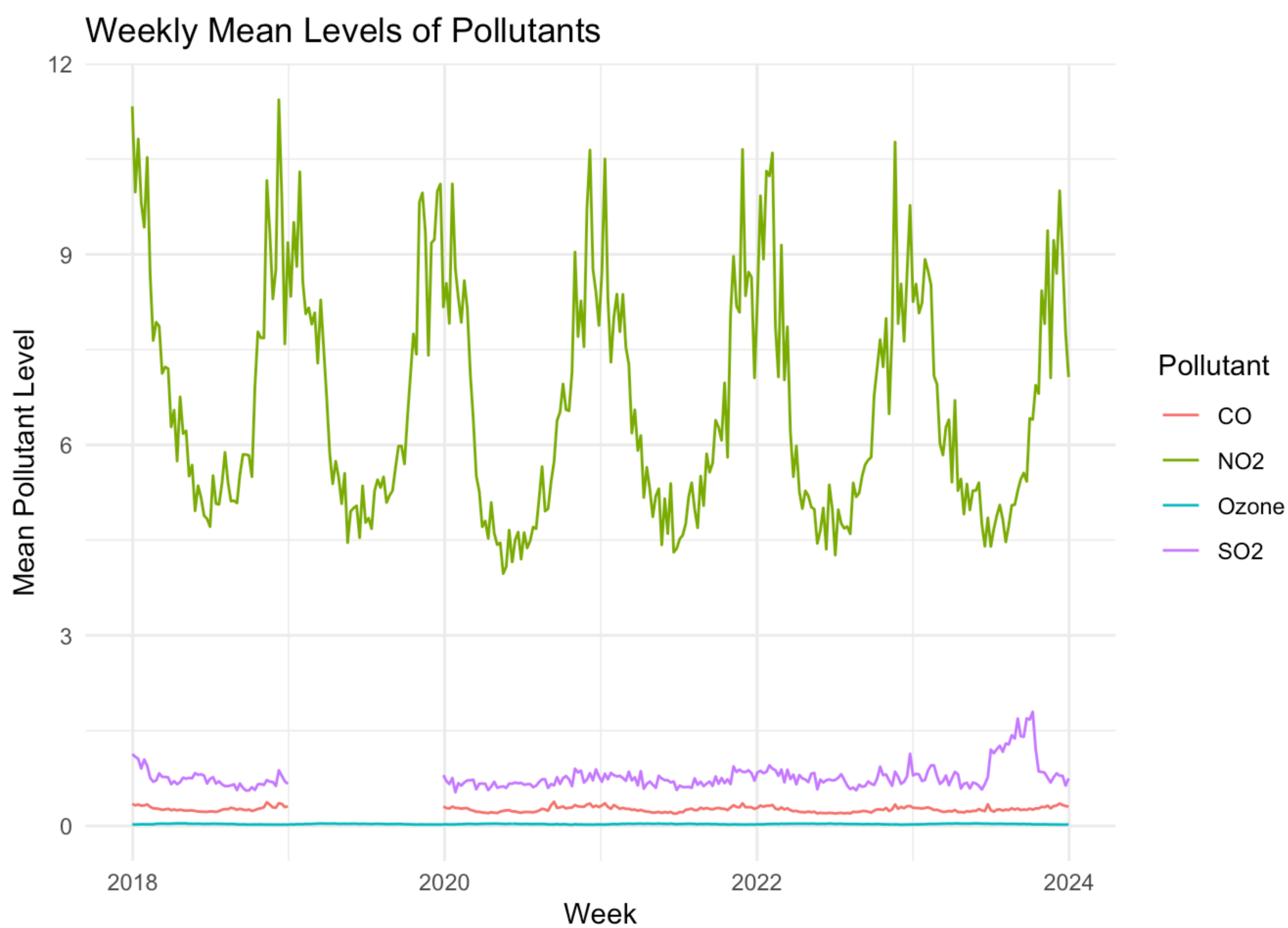
Weekly Means

```
combined_data_cleaned$Date = as.Date(combined_data_cleaned$Date)

weekly_means = combined_data_cleaned %>%
  group_by(Week = floor_date(Date, "week")) %>%
  summarise(across(c(Ozone, SO2, CO, NO2), mean, na.rm = TRUE))
```

```
## Warning: There was 1 warning in `summarise()`.
## ℹ In argument: `across(c(Ozone, SO2, CO, NO2), mean, na.rm = TRUE)`.
## ℹ In group 1: `Week = 2017-12-31`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

Plot Weekly Means

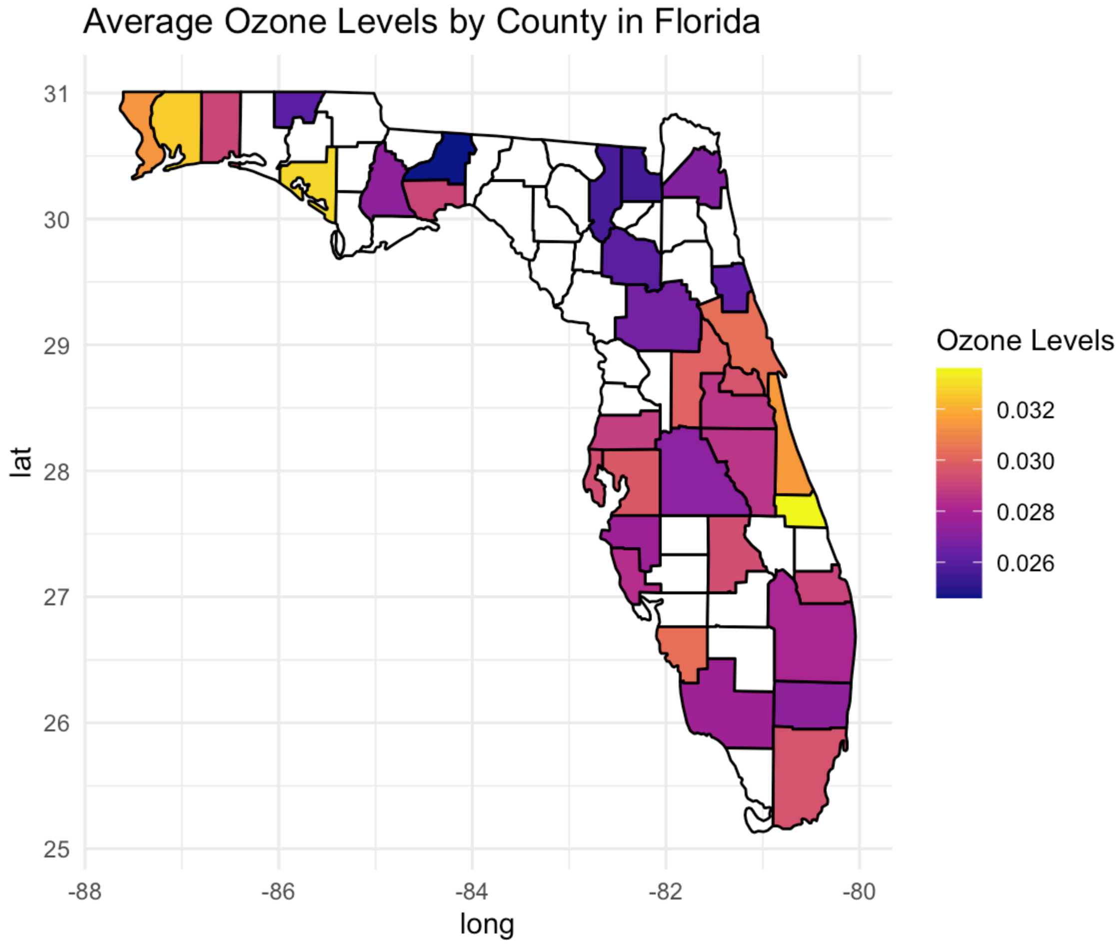
```
weekly_means %>%
  gather(Pollutant, Value, Ozone:NO2) %>%
  ggplot(aes(x = Week, y = Value, color = Pollutant)) +
  geom_line() +
  labs(title = "Weekly Mean Levels of Pollutants", x = "Week", y = "Mean Pollutant Leve
l") +
  theme_minimal()
```




```
combined_data_cleaned$Year = format(as.Date(combined_data_cleaned$Date), "%Y")
```

Function to Plot Pollutant Maps for a Given State

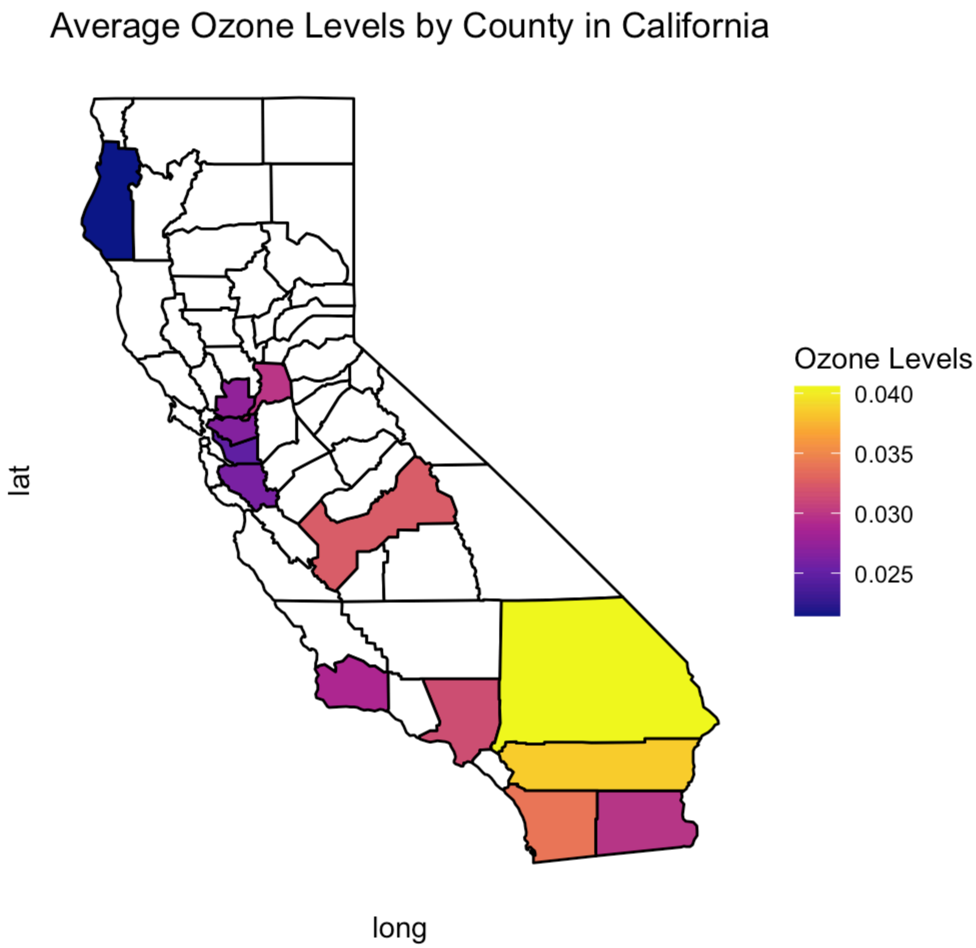
```
plot_pollutant_map = function(data, state_name) {
  data$County = tolower(data$County)
  state_data = data[data$State == state_name, ]

  # Group and summarise the pollutants
  state_data = state_data %>%
    group_by(County) %>%
    summarise(
      Ozone = mean(Ozone, na.rm = TRUE),
      SO2 = mean(SO2, na.rm = TRUE),
      CO = mean(CO, na.rm = TRUE),
      NO2 = mean(NO2, na.rm = TRUE)
    )

  # Load map data for U.S. counties and filter for the specific state
  state_map = map_data("county")
  state_map = state_map[state_map$region == tolower(state_name), ]
  state_map$subregion = tolower(state_map$subregion)

  # Merge map data with state pollutant data
  state_map_data = left_join(state_map, state_data, by = c("subregion" = "County"))

  # Plot Ozone levels
  ggplot(state_map_data, aes(x = long, y = lat, group = group, fill = Ozone)) +
    geom_polygon(color = "black") +
    coord_fixed(1.3) +
    scale_fill_viridis_c(option = "plasma", na.value = "white") +
    theme_minimal() +
    labs(title = paste("Average Ozone Levels by County in", state_name), fill = "Ozone L
evels")
}

plot_pollutant_map(combined_data_cleaned, "Florida")
```



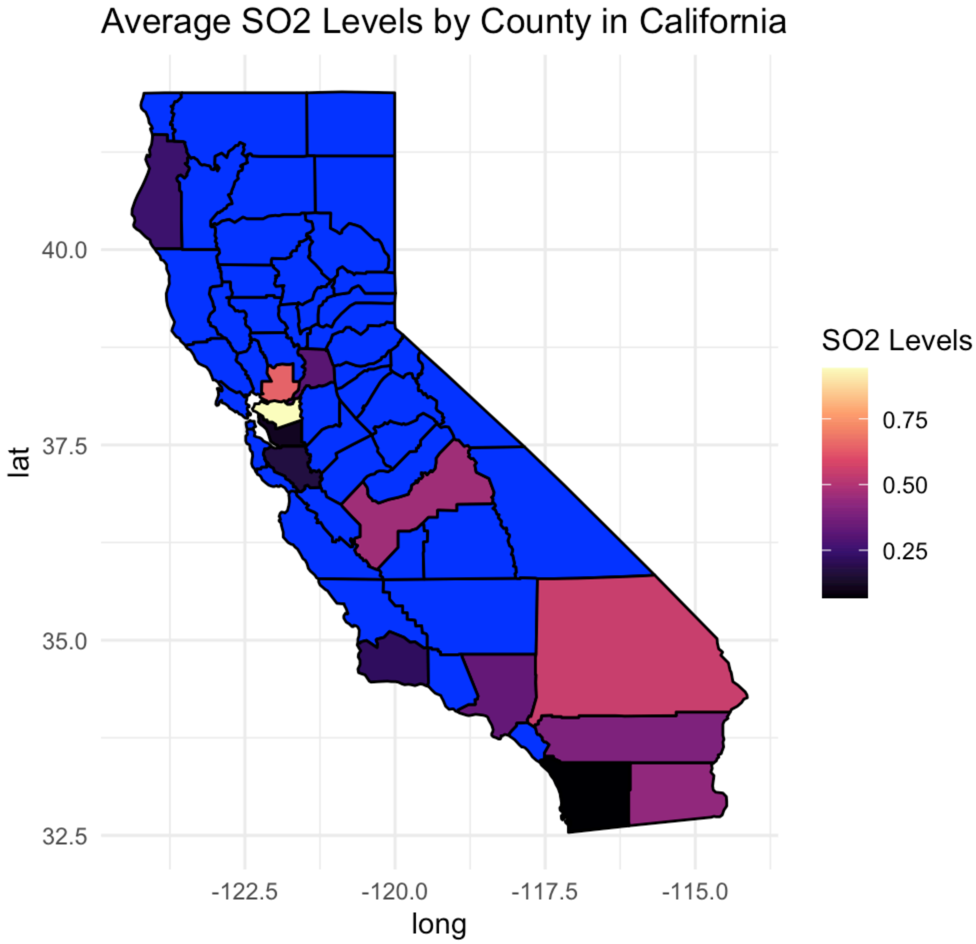
Average Ozone Levels by County in Florida
Ozone Levels
0.032
0.030
0.028
0.026
lat
31
30
29
28
27
26
25
long
-88
-86
-84
-82
-80

California Map

```
# Filter the data to include only California counties
ca_data = combined_data_cleaned[combined_data_cleaned$State == "California", ]

# Aggregate pollutant data by County
ca_data_summary = aggregate(cbind(Ozone, SO2, CO, NO2) ~ County,
                            data = ca_data,
                            FUN = function(x) mean(x, na.rm = TRUE))

# Load U.S. county map data
california_map = map_data("county")

# Filter map data for California
california_map = california_map[california_map$region == "california", ]

# Ensure lowercase consistency for merging
california_map$subregion = tolower(california_map$subregion)
ca_data_summary$County = tolower(ca_data_summary$County)

# Merge map data with pollutant data by county
california_map_data = california_map %>%
  left_join(ca_data_summary, by = c("subregion" = "County"))

# Plot Ozone levels by county in California
ggplot(california_map_data, aes(x = long, y = lat, group = group, fill = Ozone)) +
  geom_polygon(color = "black") +
  coord_fixed(1.3) +
  scale_fill_viridis_c(option = "plasma", na.value = "white") +  # Color scale
  theme_minimal() +
  labs(title = "Average Ozone Levels by County in California", fill = "Ozone Levels") +
  theme(
    axis.text = element_blank(),
    axis.ticks = element_blank(),
    panel.grid = element_blank()
  )
```




```r
# Plot SO2 levels by county in California
ggplot(california_map_data, aes(x = long, y = lat, group = group, fill = SO2)) +
  geom_polygon(color = "black") +
  coord_fixed(1.3) +
  scale_fill_viridis_c(option = "magma", na.value = "blue") +
  theme_minimal() +
  labs(title = "Average SO2 Levels by County in California", fill = "SO2 Levels")
```



##

Animated Map

```
yearly_averages = combined_data_cleaned %>%
  group_by(Year, State, County) %>%
  summarise(
    Ozone = mean(Ozone, na.rm = TRUE),
    SO2 = mean(SO2, na.rm = TRUE),
    CO = mean(CO, na.rm = TRUE),
    NO2 = mean(NO2, na.rm = TRUE)
  ) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'Year', 'State'. You can override using the
## `.groups` argument.
```

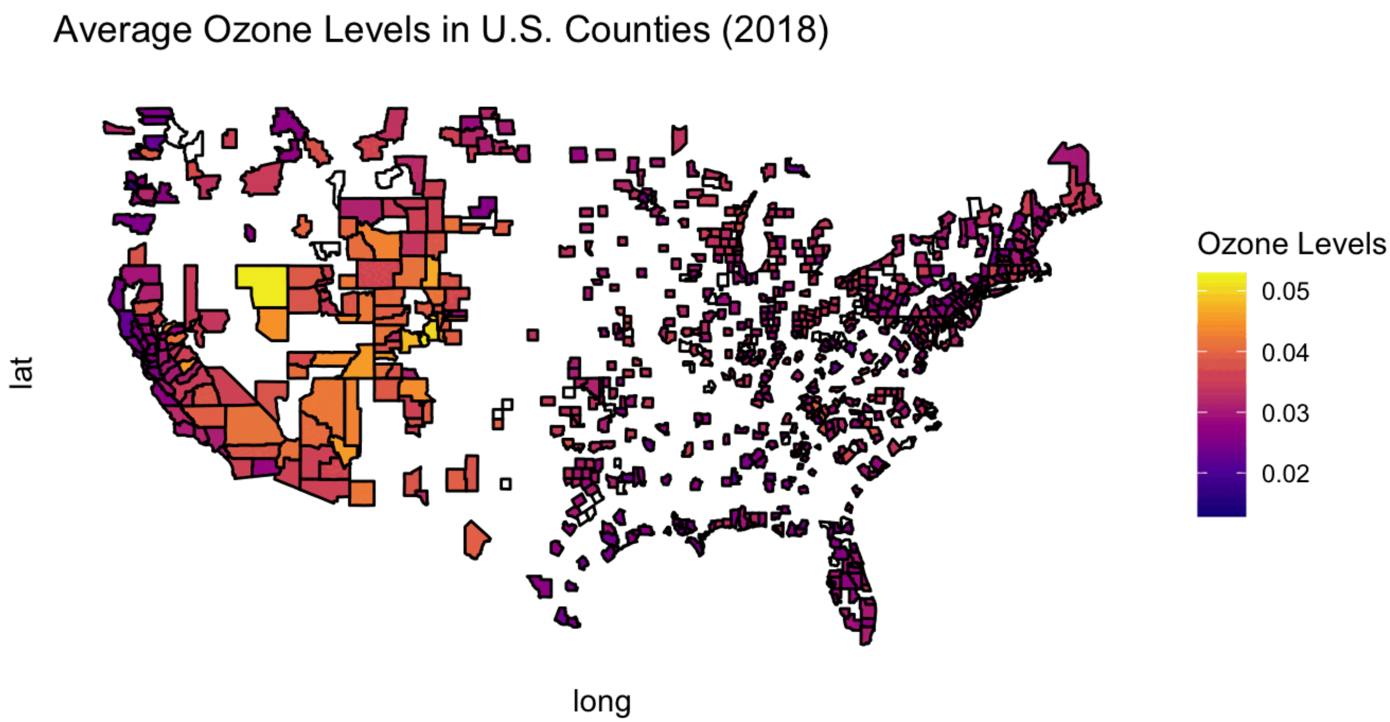
```
# Ensure 'Year' is treated as a character for data manipulation
yearly_averages$Year = as.character(yearly_averages$Year)

# Load map data for U.S. counties
county_map = map_data("county")

# Ensure lowercase county names and state names for consistency in merging
county_map$subregion = tolower(county_map$subregion)
yearly_averages$County = tolower(yearly_averages$County)
yearly_averages$State = tolower(yearly_averages$State)

# Replace missing values in critical columns with "unknown"
yearly_averages[is.na(yearly_averages$State), "State"] = "unknown"
yearly_averages[is.na(yearly_averages$County), "County"] = "unknown"
yearly_averages[is.na(yearly_averages$Year), "Year"] = "unknown"

# Convert Year back to a factor for animation purposes
yearly_averages$Year = as.factor(yearly_averages$Year)

# Merge yearly averages with map data
map_data_yearly = left_join(county_map, yearly_averages, by = c("region" = "State", "sub
region" = "County"))
```

```
## Warning in left_join(county_map, yearly_averages, by = c(region = "State", : Detected
an unexpected many-to-many relationship between `x` and `y`.
## ℹ Row 52 of `x` matches multiple rows in `y`.
## ℹ Row 1 of `y` matches multiple rows in `x`.
## ℹ If a many-to-many relationship is expected, set `relationship =
##   "many-to-many"` to silence this warning.
```

```
# Create an animated map for Ozone levels over time
animated_map = ggplot(map_data_yearly, aes(x = long, y = lat, group = group, fill = Ozon
e)) +
  geom_polygon(color = "black") +
  coord_fixed(1.3) +
  scale_fill_viridis_c(option = "plasma", na.value = "white") +
  theme_minimal() +
  labs(title = "Average Ozone Levels in U.S. Counties ({closest_state})", fill = "Ozone
Levels") +
  theme(
    axis.text = element_blank(),
    axis.ticks = element_blank(),
    panel.grid = element_blank()
  ) +
  transition_states(Year, state_length = 1, transition_length = 1) +
  ease_aes('linear')

# Animate the plot
animate(animated_map)
```

```
## Warning in lapply(row_vars$states, as.integer): NAs introduced by coercion
```

```
## Warning in expand_panel(..., self = self): NAs introduced by coercion
```



Function to Analyze Pollution Data

```
analyze_pollution = function(data, year = NULL, pollutant = NULL) {
  # If a year is provided, filter the data by that year
  if (!is.null(year)) {
    data = data[data$Year == year, ]
  }

  # If a specific pollutant is provided, select only that column
  if (!is.null(pollutant)) {
    if (!(pollutant %in% colnames(data))) {
      stop("Invalid pollutant name provided.")
    }
    data = data %>% select(Year, State, County, all_of(pollutant))
  }

  # Calculate the mean, median, and standard deviation for each pollutant
  analysis = data %>%
    summarise(
      Ozone_Mean = mean(Ozone, na.rm = TRUE),
      Ozone_Median = median(Ozone, na.rm = TRUE),
      Ozone_SD = sd(Ozone, na.rm = TRUE),
      SO2_Mean = mean(SO2, na.rm = TRUE),
      SO2_Median = median(SO2, na.rm = TRUE),
      SO2_SD = sd(SO2, na.rm = TRUE),
      CO_Mean = mean(CO, na.rm = TRUE),
      CO_Median = median(CO, na.rm = TRUE),
      CO_SD = sd(CO, na.rm = TRUE),
      NO2_Mean = mean(NO2, na.rm = TRUE),
      NO2_Median = median(NO2, na.rm = TRUE),
      NO2_SD = sd(NO2, na.rm = TRUE)
    )

  return(analysis)
}

# Ex: analyze data for 2020
analyze_pollution(combined_data_cleaned, year = "2020")
```

```
## `summarise()` has grouped output by 'Date'. You can override using the
## `.groups` argument.
```

Date <date>	State <chr>	Ozone_Mean <dbl>	Ozone_Median <dbl>	Ozone_SD <dbl>
2020-01-01	Alabama	0.033676500	0.033676500	4.034044e-03
2020-01-01	Alaska	0.031676500	0.031676500	3.119048e-03
2020-01-01	Arizona	0.027776507	0.026941000	7.999536e-03

Date <date>	State <chr>	Ozone_Mean <dbl>	Ozone_Median <dbl>	Ozone_SD <dbl>
2020-01-01	Arkansas	0.031019583	0.032323500	5.289493e-03
2020-01-01	California	0.020686001	0.021000000	6.477521e-03
2020-01-01	Colorado	0.039398218	0.039941500	6.639402e-03
2020-01-01	Connecticut	0.024058500	0.024029000	4.483700e-03
2020-01-01	Country Of Mexico	0.017471000	0.017471000	NA
2020-01-01	Delaware	0.020955750	0.021706000	2.391335e-03
2020-01-01	District Of Columbia	0.019078333	0.019078333	NA

1-10 of 10,000 rows | 1-5 of 14 columns Previous **1** 2 3 4 5 6 ... 1000 Next

Question 6: Wine Analysis

Load and Inspect Wine Dataset

```
wine_data = read.table("wine.txt", sep = ",", header = TRUE)
colnames(wine_data) = c(paste0("Feature_", 1:11), "Class_1", "Class_2", "Class_3")
head(wine_data)
```

Feature_7 <dbl>	Feature_8 <dbl>	Feature_9 <dbl>	Feature_10 <dbl>	Feature_11 <dbl>	Class_1 <dbl>	Class_2 <dbl>	Class_3 <int>
2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290

6 rows | 8-15 of 15 columns

```
# Inspect Data Structure
str(wine_data)
```

```
## 'data.frame':    177 obs. of  14 variables:
##  $ Feature_1 : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Feature_2 : num  13.2 13.2 14.4 13.2 14.2 ...
##  $ Feature_3 : num  1.78 2.36 1.95 2.59 1.76 1.87 2.15 1.64 1.35 2.16 ...
##  $ Feature_4 : num  2.14 2.67 2.5 2.87 2.45 2.45 2.61 2.17 2.27 2.3 ...
##  $ Feature_5 : num  11.2 18.6 16.8 21 15.2 14.6 17.6 14 16 18 ...
##  $ Feature_6 : int  100 101 113 118 112 96 121 97 98 105 ...
##  $ Feature_7 : num  2.65 2.8 3.85 2.8 3.27 2.5 2.6 2.8 2.98 2.95 ...
##  $ Feature_8 : num  2.76 3.24 3.49 2.69 3.39 2.52 2.51 2.98 3.15 3.32 ...
##  $ Feature_9 : num  0.26 0.3 0.24 0.39 0.34 0.3 0.31 0.29 0.22 0.22 ...
##  $ Feature_10: num  1.28 2.81 2.18 1.82 1.97 1.98 1.25 1.98 1.85 2.38 ...
##  $ Feature_11: num  4.38 5.68 7.8 4.32 6.75 5.25 5.05 5.2 7.22 5.75 ...
##  $ Class_1   : num  1.05 1.03 0.86 1.04 1.05 1.02 1.06 1.08 1.01 1.25 ...
##  $ Class_2   : num  3.4 3.17 3.45 2.93 2.85 3.58 3.58 2.85 3.55 3.17 ...
##  $ Class_3   : int  1050 1185 1480 735 1450 1290 1295 1045 1045 1510 ...
```

Split Data into Training and Test Sets

```
set.seed(123)
sample_index = sample(seq_len(nrow(wine_data)), size = 0.8 * nrow(wine_data))
train_data = wine_data[sample_index, ]
test_data = wine_data[-sample_index, ]
```

Standardize Variables

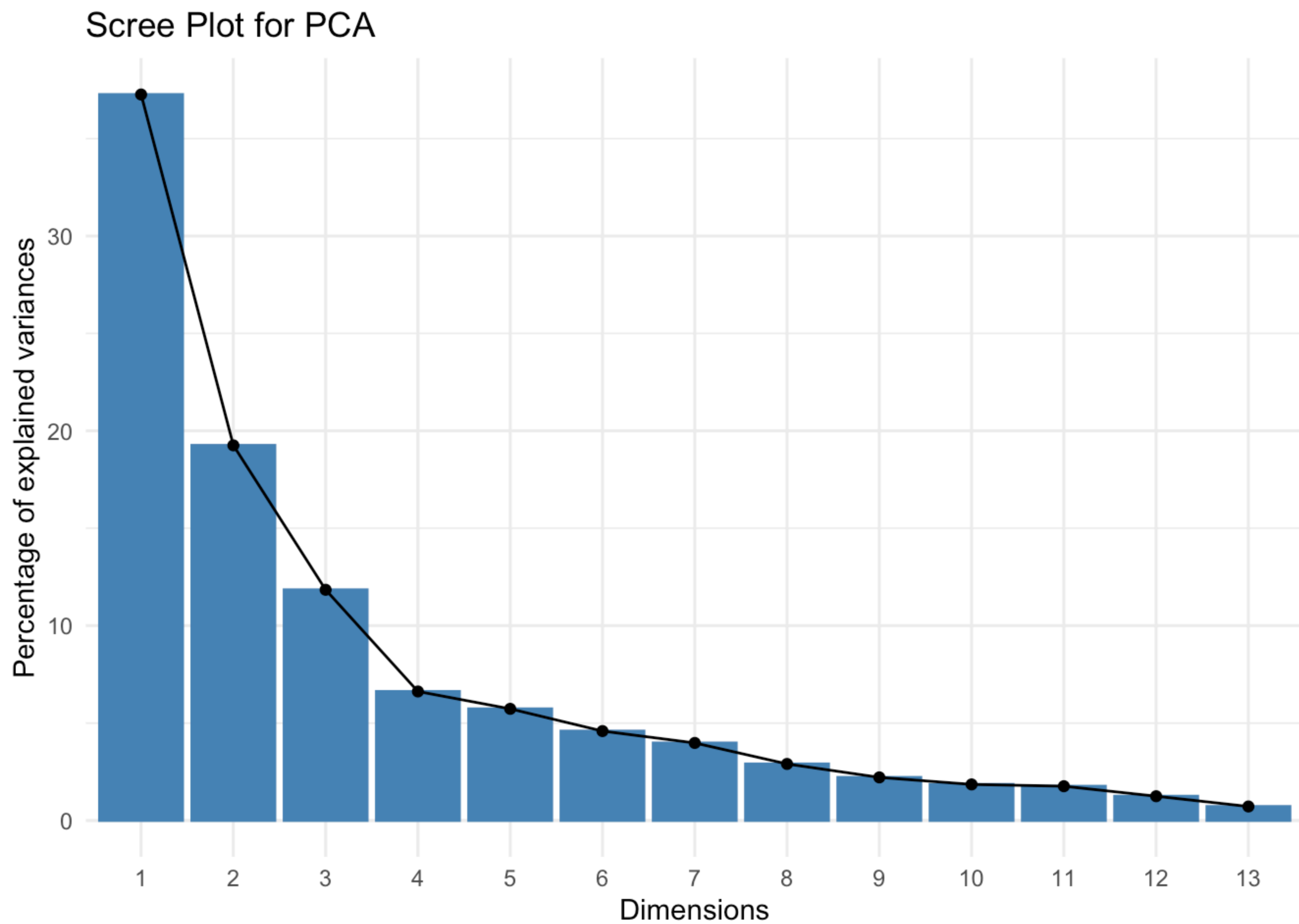
```
train_data_scaled = scale(train_data)
test_data_scaled = scale(test_data)
```

Perform PCA on Training Set

```
pca_train = prcomp(train_data_scaled[, -1], center = TRUE, scale. = TRUE)

# Determine Number of Components to Explain at Least 90% Variance
explained_variance = summary(pca_train)$importance[3, ]
num_components = which(cumsum(explained_variance) >= 0.9)[1]
cat("Number of components explaining at least 90% variance:", num_components, "\n")
```

```
## Number of components explaining at least 90% variance: 2
```

```
# Create Scree Plot
fviz_screeplot(pca_train, ncp = ncol(train_data_scaled) - 1) +
  ggtitle("Scree Plot for PCA") +
  theme_minimal()
```



K-means Clustering on Original Variables

```
set.seed(123)
kmeans_original = kmeans(train_data_scaled[, -1], centers = 3, nstart = 25)

# Assign Clusters to Training Data
train_data_scaled$cluster = kmeans_original$cluster
```

```
## Warning in train_data_scaled$cluster = kmeans_original$cluster: Coercing LHS to
## a list
```

```
# Predict Clusters for the Test Set Manually
get_nearest_cluster = function(point, centroids) {
  distances = apply(centroids, 1, function(centroid) sum((point - centroid) ^ 2))
  return(which.min(distances))
}

test_data_scaled$cluster = apply(test_data_scaled[, -1], 1, get_nearest_cluster, centroi
ds = kmeans_original$centers)
```

```
## Warning in test_data_scaled$cluster = apply(test_data_scaled[, -1], 1,
## get_nearest_cluster, : Coercing LHS to a list
```

K-means Clustering on PCA-transformed Data

```
kmeans_pca = kmeans(as.data.frame(pca_train$x[, 1:num_components]), centers = 3, nstart
= 25)

# Assign Clusters to PCA-transformed Training Data
train_pca_data = as.data.frame(pca_train$x[, 1:num_components])
train_pca_data$cluster = kmeans_pca$cluster
```

Create Scatter Plot for First Two Principal Components

```
fviz_cluster(list(data = train_pca_data, cluster = kmeans_pca$cluster), geom = "point",
stand = FALSE) +
  ggtitle("Clusters on PCA-transformed Data (First Two Components)") +
  theme_minimal()
```