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/ve3WBa

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

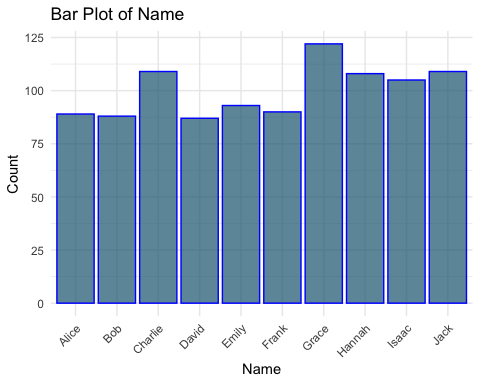
## Define Summary Function

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

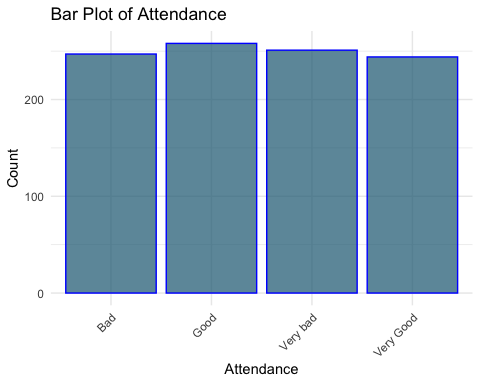
## Warning: NAs introduced by coercion

display\_results(copy)

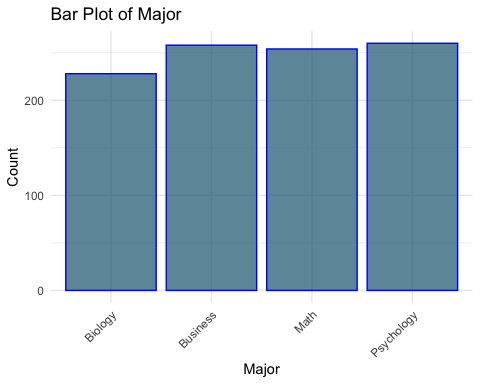
## The class of Name is: character



## 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

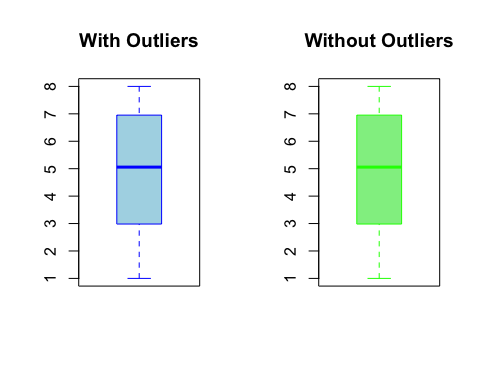
## Define Conversion Function

# 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 = "green")  
 }  
}  
  
# 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  
## 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

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

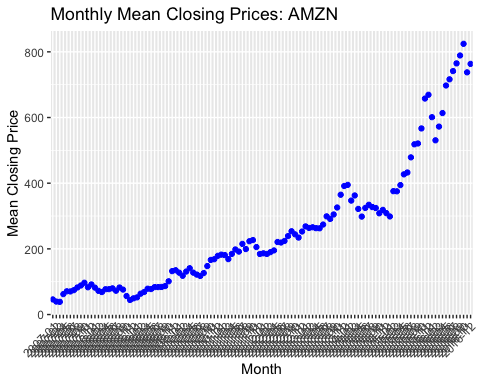
# Calculate daily returns for each company  
returns = list()  
  
for (i in 2:ncol(stocks\_data)) {  
 return\_vector = numeric(nrow(stocks\_data) - 1)  
   
 for (j in 2:nrow(stocks\_data)) {  
 return\_vector[j - 1] = (stocks\_data[j, i] - stocks\_data[j - 1, i]) / stocks\_data[j - 1, i]  
 }  
   
 returns[[colnames(stocks\_data)[i]]] = return\_vector  
}  
  
return\_df = as.data.frame(returns)  
head(return\_df, 10)

## AMZN DUK KO  
## 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  
## 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

## Monthly Mean Closing Prices

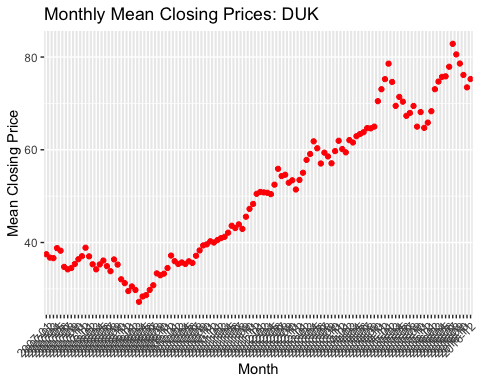
# Extract month and calculate monthly averages for each company  
stocks\_data$Month = format(stocks\_data$Date, "%Y-%m")  
  
monthly\_means = stocks\_data %>%  
 group\_by(Month) %>%  
 summarise(  
 monthly\_AMZN = mean(AMZN, na.rm = TRUE),   
 monthly\_DUK = mean(DUK, na.rm = TRUE),   
 monthly\_KO = 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 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?



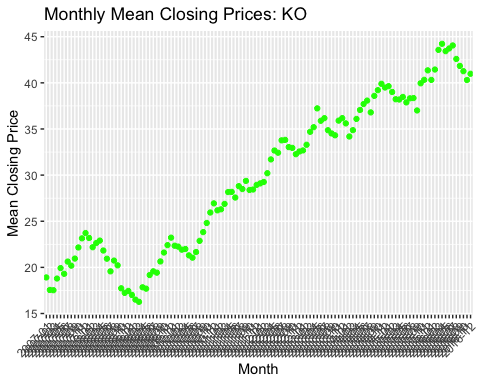
## Duke Energy (DUK)  
ggplot(monthly\_means, aes(x = Month, y = monthly\_DUK)) +  
 geom\_line(color = "red") +  
 geom\_point(color = "red") +  
 labs(title = "Monthly Mean Closing Prices: DUK", x = "Month", y = "Mean Closing Price") +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))

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



## 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" "Private 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)

## # A tibble: 10 × 2  
## neighborhood avg  
## <chr> <dbl>  
## 1 Leather District 4.88  
## 2 South Boston Waterfront 4.83  
## 3 Chinatown 4.81  
## 4 Roslindale 4.79  
## 5 Jamaica Plain 4.75  
## 6 South End 4.73  
## 7 Charlestown 4.7   
## 8 Roxbury 4.70  
## 9 South Boston 4.70  
## 10 North End 4.68

# 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", full.names = TRUE) # CO  
file\_list\_SO2 = list.files(path = folder\_path, pattern = "daily\_42401\_[0-9]{4}.csv", full.names = TRUE) # SO2  
file\_list\_NO2 = list.files(path = folder\_path, pattern = "daily\_42602\_[0-9]{4}.csv", full.names = TRUE) # NO2  
file\_list\_Ozone = list.files(path = folder\_path, pattern = "daily\_44201\_[0-9]{4}.csv", full.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 required columns  
 rename(State = State.Name, County = County.Name, !!pollutant\_name := Arithmetic.Mean)  
 })  
 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 an 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 an 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

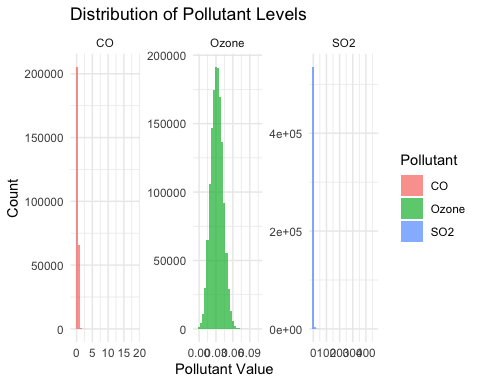
combined\_data\_cleaned = combined\_data %>%  
 group\_by(Date, State, County) %>%  
 summarise(  
 Ozone = mean(Ozone, na.rm = TRUE),  
 SO2 = mean(SO2, na.rm = TRUE),  
 CO = mean(CO, na.rm = TRUE),  
 NO2 = mean(NO2, na.rm = TRUE)  
 )

## `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()`).



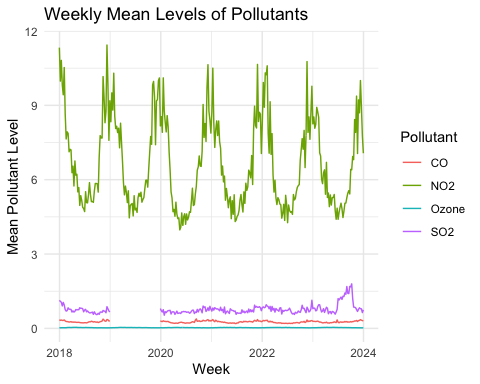
## Weekly Means

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

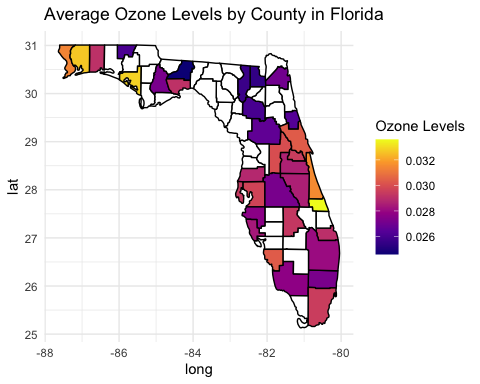
weekly\_means %>%  
 gather(Pollutant, Value, Ozone:NO2) %>%  
 ggplot(aes(x = Week, y = Value, color = Pollutant)) +  
 geom\_line() +  
 labs(title = "Weekly Mean Levels of Pollutants", x = "Week", y = "Mean Pollutant Level") +  
 theme\_minimal()



combined\_data\_cleaned$Year = format(as.Date(combined\_data\_cleaned$Date), "%Y")

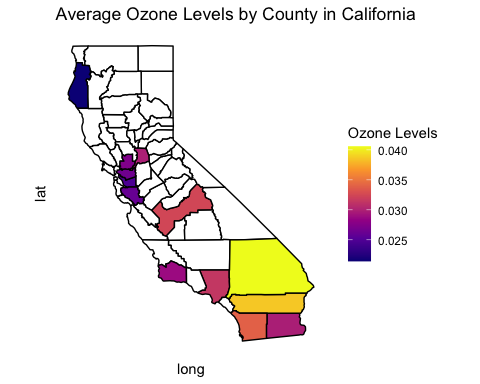
## Function to Plot Pollutant Maps for a Given State

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 Levels")  
}  
  
plot\_pollutant\_map(combined\_data\_cleaned, "Florida")

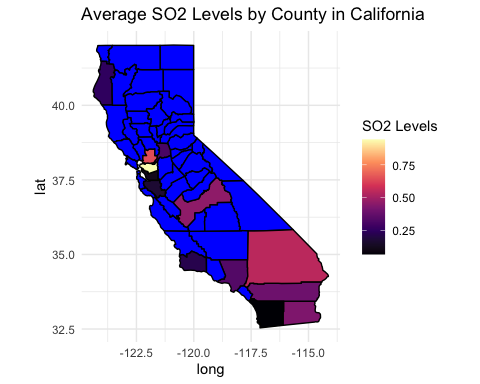


## California Map

# 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()  
 )



# 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.

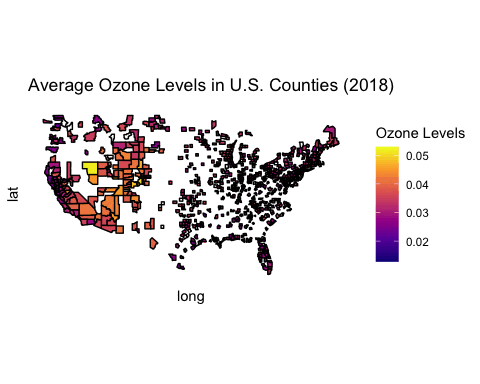
# Ensure 'Year' is treated as a character for data manipulation  
yearly\_averages$Year = as.character(yearly\_averages$Year)  
  
# Load map data for U.S. counties  
county\_map = map\_data("county")  
  
# Ensure lowercase county names and state names for consistency in merging  
county\_map$subregion = tolower(county\_map$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", "subregion" = "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 = Ozone)) +  
 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.

## # A tibble: 19,178 × 14  
## # Groups: Date [366]  
## Date State Ozone\_Mean Ozone\_Median Ozone\_SD SO2\_Mean SO2\_Median SO2\_SD  
## <date> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 2020-01-01 Alab… 0.0337 0.0337 0.00403 0.833 0.680 0.514   
## 2 2020-01-01 Alas… 0.0317 0.0317 0.00312 1.35 1.35 NA   
## 3 2020-01-01 Ariz… 0.0278 0.0269 0.00800 2.47 1.18 3.16   
## 4 2020-01-01 Arka… 0.0310 0.0323 0.00529 0.838 0.838 NA   
## 5 2020-01-01 Cali… 0.0207 0.021 0.00648 0.415 0.129 0.482   
## 6 2020-01-01 Colo… 0.0394 0.0399 0.00664 1.22 0.761 0.809   
## 7 2020-01-01 Conn… 0.0241 0.0240 0.00448 0.106 0.115 0.101   
## 8 2020-01-01 Coun… 0.0175 0.0175 NA NaN NA NA   
## 9 2020-01-01 Dela… 0.0210 0.0217 0.00239 0.163 0.163 0.0950  
## 10 2020-01-01 Dist… 0.0191 0.0191 NA 1.72 1.72 NA   
## # ℹ 19,168 more rows  
## # ℹ 6 more variables: CO\_Mean <dbl>, CO\_Median <dbl>, CO\_SD <dbl>,  
## # NO2\_Mean <dbl>, NO2\_Median <dbl>, NO2\_SD <dbl>

# 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\_1 Feature\_2 Feature\_3 Feature\_4 Feature\_5 Feature\_6 Feature\_7  
## 1 1 13.20 1.78 2.14 11.2 100 2.65  
## 2 1 13.16 2.36 2.67 18.6 101 2.80  
## 3 1 14.37 1.95 2.50 16.8 113 3.85  
## 4 1 13.24 2.59 2.87 21.0 118 2.80  
## 5 1 14.20 1.76 2.45 15.2 112 3.27  
## 6 1 14.39 1.87 2.45 14.6 96 2.50  
## Feature\_8 Feature\_9 Feature\_10 Feature\_11 Class\_1 Class\_2 Class\_3  
## 1 2.76 0.26 1.28 4.38 1.05 3.40 1050  
## 2 3.24 0.30 2.81 5.68 1.03 3.17 1185  
## 3 3.49 0.24 2.18 7.80 0.86 3.45 1480  
## 4 2.69 0.39 1.82 4.32 1.04 2.93 735  
## 5 3.39 0.34 1.97 6.75 1.05 2.85 1450  
## 6 2.52 0.30 1.98 5.25 1.02 3.58 1290

# 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

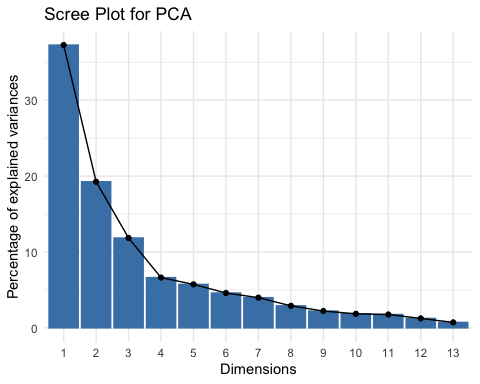
train\_data\_scaled = scale(train\_data)  
test\_data\_scaled = scale(test\_data)

## 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$cluster = apply(test\_data\_scaled[, -1], 1, get\_nearest\_cluster, centroids = kmeans\_original$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()

