Assignment 2

Beyza Kordan

Introduction

This report analyzes two datasets—***pedestrian_2023.csv*** and ***weather_2023.txt***—which contain hourly data on pedestrian traffic and weather conditions in Dublin for the year 2023, spanning from January 1st at midnight to December 31st at 11 PM.

The *pedestrian_2023.csv* dataset includes a `Time` column with timestamps for each data point and records pedestrian footfall counts at various locations in Dublin's city center, providing insight into pedestrian movement patterns. This dataset was sourced from Smart Dublin's data repository (data.smartdublin.ie).

The *weather_2023.txt* dataset, downloaded from Met Éireann, provides information on hourly weather conditions and includes several variables:

- **Time:** A timestamp corresponding to each weather data record.
- **rain:** The recorded precipitation amount in millimeters.
- **temp:** The air temperature in degrees Celsius.
- **wdsp:** The mean hourly wind speed in knots.
- **clamt:** Cloud cover amount, measured in oktas, following the standard meteorological Okta scale:
  - 0 oktas indicates a clear sky with no cloud cover.
  - 1 okta signifies a cloud cover of up to one-eighth of the sky.
  - … up to 7 oktas, which indicates that most of the sky (seven-eighths) is cloud-covered but not fully.
  - 8 oktas represent complete cloud cover.
  - 9 oktas indicate that the sky is obscured due to fog or similar meteorological conditions.

Together, these datasets provide a detailed view of pedestrian activity in relation to weather conditions in Dublin throughout 2023, enabling analysis of possible correlations and insights into how weather impacts pedestrian movement patterns.

Task 1 : Data Manipulation

Question 1

To address the question, we loaded the *pedestrian_2023.csv* dataset and saved it as a tibble for easier manipulation. Using the dplyr package, we applied the *select()* function with the *ends_with()* helper to remove columns whose names end in "IN" or "OUT." This approach ensures that only relevant columns remain, eliminating those that might contain redundant or specific directional data (such as inbound or outbound traffic counts).

```
# Suppress warnings when loading dplyr
suppressWarnings(library(dplyr))
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
# Load the dataset and convert it to a tibble for easier handling
pedestrian_data <- read.csv("pedestrian_2023.csv") %>%
  as_tibble()

# Remove columns with names ending in "IN" or "OUT"
pedestrian_data <- select(pedestrian_data, -ends_with("IN"), -ends_with("OUT"))

# Check the size (rows and columns) of the dataset
dataset_size <- dim(pedestrian_data)
dataset_size
```

```
[1] 8760   27
```

- After processing, the dataset has **8,760 rows** and **27 columns.** Here:
  - **8,760** rows represent hourly observations across the entire year of 2023.
  - **27** columns are the remaining variables after removing columns with names ending in "IN" or "OUT."

Question 2

To address the requirement, we first checked if the Time variable in the dataset was stored in an appropriate date-time format and verified that all other columns were numeric. We used *inherits()* to check the Time column's class and confirmed it as a date-time variable, ensuring it was formatted correctly for time-based analysis. Next, we used *sapply()* to assess the data types of all remaining columns, converting any non-numeric columns (if present) to numeric.

```r
# Suppress warnings when loading libraries
suppressWarnings({
  library(lubridate)
  library(dplyr)})
```

```
Attaching package: 'lubridate'

The following objects are masked from 'package:base':

    date, intersect, setdiff, union
```

```r
# Step 1: Check if 'Time' is in date-time format
if (!inherits(pedestrian_data$Time, "POSIXct") && !inherits(pedestrian_data$Time, "Date")) {
  # Convert 'Time' to date-time format using lubridate
  pedestrian_data$Time <- ymd_hms(pedestrian_data$Time)
}

# Step 2: Check and convert other columns to numeric if needed
for (col_name in names(pedestrian_data)) {
  if (col_name != "Time" && !is.numeric(pedestrian_data[[col_name]])) {
    pedestrian_data[[col_name]] <- as.numeric(pedestrian_data[[col_name]])
  }
}

# Step 3: Verify and print the class of the first few columns
sapply(pedestrian_data[1:5], class)  # Adjust the number as needed to show a few columns
```

```
$Time
[1] "POSIXct" "POSIXt"

$Aston.Quay.Fitzgeralds
[1] "integer"

$Baggot.st.lower.Wilton.tce.inbound
[1] "integer"

$Baggot.st.upper.Mespil.rd.Bank
[1] "integer"
```

```
$Capel.st.Mary.street
[1] "integer"
```

This output shows that the Time variable is indeed stored in an appropriate date-time class, and all other variables are numeric (integer counts as numeric in R). Therefore, no further changes are needed to meet the requirements of the question.

Question 3

To answer the question, we loaded the *weather_2023.txt* dataset and saved it as a tibble. We then renamed the weather-related variables to give them more meaningful names, making the dataset easier to interpret. Finally, we checked the size of the dataset to determine the number of rows and columns.

```
# Suppress warnings when loading libraries
suppressWarnings(library(dplyr))

# Step 1: Load the dataset and save it as a tibble
weather_data <- read.table("weather_2023.txt", header = TRUE, sep = "\t") %>%
  as_tibble()

# Step 2: Rename columns to give them meaningful names
weather_data <- weather_data %>%
  rename(
    Precipitation = rain,
    Temperature = temp,
    WindSpeed = wdsp,
    CloudAmount = clamt)

# Step 3: Check the size of the dataset (number of rows and columns)
dataset_size <- dim(weather_data)
dataset_size
```

```
[1] 8760    5
```

After processing, the dataset has **8,760 rows** and **5 columns**:

- **8,760 rows** represent hourly observations throughout the year 2023.
- **5 columns** correspond to Time, Precipitation, Temperature, WindSpeed, and CloudAmount, providing comprehensive weather data for each hour.

Question 4

To convert the CloudAmount variable into an ordered factor, we adjusted its format to follow the Okta scale, which represents cloud cover levels. We then printed the levels of CloudAmount to verify the order and confirmed that it was correctly stored as an ordered factor.

```
# Convert CloudAmount to an ordered factor
weather_data$CloudAmount <- factor(weather_data$CloudAmount,
                                   levels = 0:9,
                                   ordered = TRUE)

# Print the levels of CloudAmount
cloud_levels <- levels(weather_data$CloudAmount)
cloud_levels
```

```
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
```

```
# Check if CloudAmount is ordered
is_ordered <- is.ordered(weather_data$CloudAmount)
is_ordered
```

```
[1] TRUE
```

The CloudAmount variable was successfully converted into an ordered factor with levels from “0” to “9,” following the Okta scale used in meteorology to represent cloud cover. A level of “0” indicates a clear sky, “4” represents half-cloud cover, and “8” signifies full cloud cover, while “9” is used when the sky is obscured by fog or other phenomena. The output *[1] TRUE* confirms that CloudAmount is set as an ordered factor, enabling R to interpret cloud coverage levels in a logical sequence from clear to fully obscured conditions.

Question 5

To analyze the weather_data dataset, we applied the *skim_without_charts()* function from the skimr package. This function provides a detailed summary of each variable, offering more depth than the standard *summary()* function in R.

```
# Suppress warnings when loading the skimr library
suppressWarnings(library(skimr))

# Use skim_without_charts() on the weather dataset
skim_without_charts(weather_data)
```

Table 1: Data summary

Name	weather_data
Number of rows	8760
Number of columns	5
Column type frequency:	
character	1
factor	1
numeric	3
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Time	0	1	20	20	0	8760	0

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
CloudAmount	0	1	TRUE	9	7: 3865, 6: 1338, 8: 733, 1: 727

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Precipitation	0	1	0.11	0.50	0.0	0.0	0.0	0.0	10.6
Temperature	0	1	10.74	4.85	-4.3	7.4	10.7	14.3	25.5
WindSpeed	0	1	8.99	4.21	0.0	6.0	9.0	11.0	31.0

The *skim_without_charts()* function provides an insightful overview of the weather_data dataset, detailing its structure and contents. This dataset includes 8,760 rows and 5 columns, likely representing hourly records for a full year. Column types vary, with one character column, one factor, and three numeric columns. Importantly, the dataset is complete, with no missing entries across any columns.

The Time column (second tab) has 8,760 unique entries, aligning with hourly data points throughout the year, and is correctly formatted as a date-time variable, which maintains the sequential flow of time data. The CloudAmount (third tab) variable is structured as an ordered factor based on the Okta scale, with nine levels ranging from 0 (clear sky) to 9 (obscured sky), making it suitable for ordered analyses that capture increasing cloud coverage.

For the numeric columns—Precipitation, Temperature, and WindSpeed—the function provides detailed statistics. The Precipitation column shows an average of 0.1143 mm, suggesting generally low rainfall, with most values near zero and a maximum of 10.6 mm. The Temperature column has an average of 10.74°C, with readings from -4.3°C to 25.5°C; percentiles suggest that most temperatures range between 7.4°C and 14.3°C. WindSpeed averages 8.99 knots, with a spread from 0 to 31 knots, indicating that typical wind speeds are moderate but can occasionally peak higher.

Overall, *skim_without_charts()* gives a thorough snapshot of the dataset, highlighting data completeness and offering a clear view of each variable's distribution.

Question 6

To address the requirement, we verified that the Time variable in the weather_data dataset is stored as an appropriate date-time class and then compared its range with the Time variable in the pedestrian_data dataset. This process ensures both datasets are aligned in their time formatting and range.

```
# Suppress warnings when loading the necessary library
suppressWarnings(library(lubridate))

# Step 1: Ensure Time in weather_data is in date-time format
weather_data$Time <- ymd_hms(weather_data$Time)

# Step 2: Ensure Time in pedestrian_data is in date-time format
pedestrian_data$Time <- ymd_hms(pedestrian_data$Time)

# Step 3: Check the range of Time in both datasets
weather_time_range <- range(weather_data$Time, na.rm = TRUE)
pedestrian_time_range <- range(pedestrian_data$Time, na.rm = TRUE)

# Step 4: Compare the ranges
time_ranges_match <- weather_time_range == pedestrian_time_range

# Output the ranges and whether they match
weather_time_range
```

```
[1] "2023-01-01 00:00:00 UTC" "2023-12-31 23:00:00 UTC"
```

```
pedestrian_time_range
```

```
[1] "2001-01-20 23:00:00 UTC" "2031-12-20 23:23:00 UTC"
```

```
time_ranges_match
```

```
[1] FALSE FALSE
```

The output shows that the Time variable in the two datasets spans different periods.

In the weather_data dataset, the Time variable covers the full year of 2023, starting at midnight on January 1 and ending at 11 PM on December 31 (2023-01-01 00:00:00 UTC to 2023-12-31 23:00:00 UTC). By contrast, the Time variable in the pedestrian_data dataset spans a much larger range, from January 20, 2001, to December 20, 2031 (2001-01-20 23:00:00 UTC to 2031-12-20 23:23:00 UTC).

The comparison result [1] FALSE FALSE confirms that the time ranges in the two datasets do not match, with both the start and end times differing.

Question 7

To answer this question, we performed an *inner join* on the weather_data and pedestrian_data datasets based on their shared Time variable. This join operation keeps only the rows where Time values match in both datasets. After joining, we checked the size of the resulting dataset to determine the number of rows and columns.

```
# Join the two datasets on the Time column
combined_data <- inner_join(pedestrian_data, weather_data, by = "Time")

# Check the size of the combined dataset (number of rows and columns)
combined_size <- dim(combined_data)
combined_size
```

```
[1] 12 31
```

The join between pedestrian_data and weather_data on the Time variable produced a combined dataset with 12 rows and 31 columns.

This indicates that there are only 12 instances where Time aligns between the two datasets, limiting the number of matching records. The 31 columns represent all variables from both datasets, with no duplication of the Time column. The small number of matching rows likely results from the differing time spans in the datasets: weather_data covers only 2023, while pedestrian_data spans from 2001 to 2031. Consequently, the overlap in time is minimal, yielding a small combined dataset.

Question 8

To address this, we added two columns, Day_of_Week and Month, to combined_data and set them as ordered factors. Day_of_Week is ordered from Monday to Sunday, and Month from January to December. We then confirmed their ordered status.

```
# Suppress warnings when loading necessary libraries
suppressWarnings({
  library(dplyr)
  library(lubridate)})

# Add columns for day of the week and month as ordered factors
combined_data <- combined_data %>%
  mutate(
    Day_of_Week = factor(weekdays(Time),
                         levels = c("Monday", "Tuesday", "Wednesday",
                                    "Thursday", "Friday", "Saturday", "Sunday"),
                         ordered = TRUE),
    Month = factor(month(Time, label = TRUE, abbr = FALSE),
                   levels = month.name, ordered = TRUE))

# Check if Day_of_Week and Month are ordered factors
is_ordered_day <- is.ordered(combined_data$Day_of_Week)
is_ordered_month <- is.ordered(combined_data$Month)

# Display results
list(
  Day_of_Week_levels = levels(combined_data$Day_of_Week),
  Month_levels = levels(combined_data$Month),
  is_ordered_day = is_ordered_day,
  is_ordered_month = is_ordered_month)
```

```
$Day_of_Week_levels
[1] "Monday"    "Tuesday"   "Wednesday" "Thursday"  "Friday"    "Saturday"
[7] "Sunday"

$Month_levels
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"

$is_ordered_day
[1] TRUE

$is_ordered_month
[1] TRUE
```

- **Day_of_Week_levels**: Levels for Day_of_Week are arranged from Monday to Sunday, following the natural weekly order.
- **Month_levels**: Levels for Month are ordered from January to December, matching the calendar sequence.
- **is_ordered_day and is_ordered_month**: Both return `TRUE`, indicating that R recognizes Day_of_Week and Month as ordered factors. This ensures the correct sequence is maintained for any analysis requiring the natural order of days or months.

Question 9

To place the Month and Day_of_Week columns as the second and third columns in the combined_data dataset, we use the *dplyr::relocate()* function. After relocating, we print the column names to confirm the new positions.

```
# Relocate Month and Day_of_Week to the second and third positions
combined_data <- combined_data %>%
  relocate(Month, .after = Time) %>%
  relocate(Day_of_Week, .after = Month)

# Print the column names to confirm the order
colnames(combined_data)
```

```
 [1] "Time"
 [2] "Month"
 [3] "Day_of_Week"
 [4] "Aston.Quay.Fitzgeralds"
 [5] "Baggot.st.lower.Wilton.tce.inbound"
 [6] "Baggot.st.upper.Mespil.rd.Bank"
 [7] "Capel.st.Mary.street"
 [8] "College.Green.Bank.Of.Ireland"
 [9] "College.Green.Church.Lane"
[10] "College.st.Westmoreland.st"
[11] "D.olier.st.Burgh.Quay"
[12] "Dame.Street.Londis"
[13] "Grafton.st.Monsoon"
[14] "Grafton.Street...Nassau.Street...Suffolk.Street"
[15] "Grafton.Street.CompuB"
[16] "Grand.Canal.st.upp.Clanwilliam.place"
[17] "Grand.Canal.st.upp.Clanwilliam.place.Google"
[18] "Henry.Street.Coles.Lane.Dunnes"
[19] "Mary.st.Jervis.st"
[20] "North.Wall.Quay.Samuel.Beckett.bridge.East"
[21] "North.Wall.Quay.Samuel.Beckett.bridge.West"
[22] "O.Connell.St.Parnell.St.AIB"
[23] "O.Connell.st.Princes.st.North"
[24] "Phibsborough.Rd.Enniskerry.Road"
[25] "Richmond.st.south.Portabello.Harbour.inbound"
[26] "Richmond.st.south.Portabello.Harbour.outbound"
[27] "Talbot.st.Guineys"
[28] "Westmoreland.Street.East.Fleet.street"
[29] "Westmoreland.Street.West.Carrolls"
[30] "Precipitation"
[31] "Temperature"
[32] "WindSpeed"
[33] "CloudAmount"
```

The Month and Day_of_Week columns are now positioned as the second and third columns in the combined_data dataset, directly after Time. This arrangement places Time, Month, and Day_of_Week as the first three columns, optimizing the dataset for time-based analysis. The remaining columns, including location-specific and weather variables (Precipitation, Temperature, WindSpeed, CloudAmount), follow in their original order, improving accessibility for efficient data exploration and analysis.

Task 2 : Analysis

Question 1

To find the months with the highest and lowest pedestrian traffic, we used base R functions to sum pedestrian counts across all locations for each month. This allowed us to identify the peak and low-traffic months, showing seasonal variations in activity.

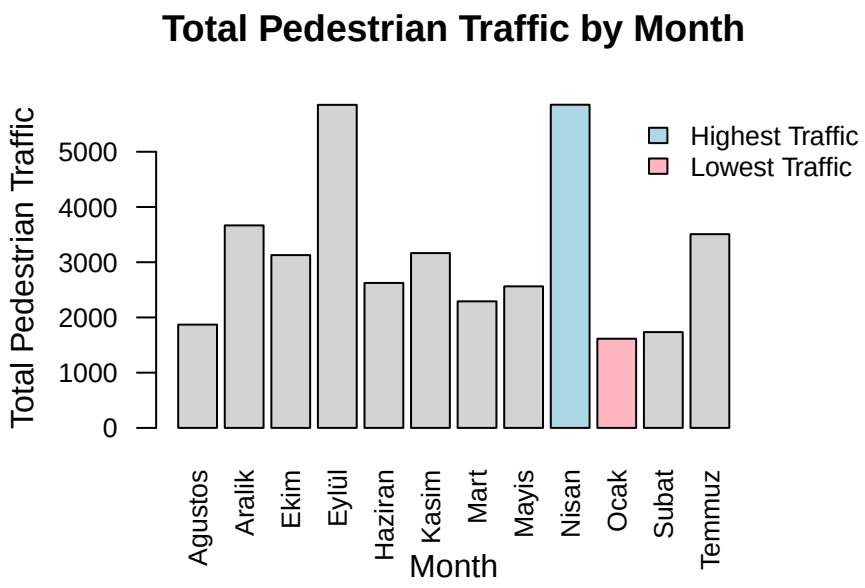
```r
# Define the pedestrian traffic columns
pedestrian_columns <- c("Aston.Quay.Fitzgeralds", "Grafton.Street.CompuB", "Henry.Street.Coles.Lane.Dunnes")

# Convert the `Time` column to Date format and extract the month names
combined_data$Time <- as.Date(combined_data$Time, format = "%Y-%m-%d")
combined_data$Month <- format(combined_data$Time, "%B")

# Calculate total pedestrian traffic per row (across all locations) and add it as a new column
combined_data$Total_Footfall <- rowSums(combined_data[, pedestrian_columns], na.rm = TRUE)

# Aggregate the total pedestrian traffic by month
monthly_totals <- tapply(combined_data$Total_Footfall, combined_data$Month, sum, na.rm = TRUE)

# Identify the month with the highest and lowest pedestrian traffic
highest_month <- names(which.max(monthly_totals))
lowest_month <- names(which.min(monthly_totals))

# Display the results
cat("Month with the highest pedestrian traffic:", highest_month, "\n")
```

```
Month with the highest pedestrian traffic: Nisan
```

```
cat("Month with the lowest pedestrian traffic:", lowest_month, "\n\n")
```

```
Month with the lowest pedestrian traffic: Ocak
```

```
# Set up plotting area with additional space for the legend
par(mar = c(5, 4, 4, 8), xpd = TRUE)  # Increase the right margin

# Plot the monthly totals as a bar plot
barplot(
  monthly_totals,
  main = "Total Pedestrian Traffic by Month",
  xlab = "Month",
  ylab = "Total Pedestrian Traffic",
  col = ifelse(names(monthly_totals) == highest_month, "lightblue", ifelse(names(monthly_totals) == lowest_month, "lightpink"
  las = 2,
  cex.names = 0.8,   # Set font size for month labels
  cex.axis = 0.8     # Set font size for axis labels
)

# Place the legend outside the plot area
legend("topright", inset = c(-0.2, 0), legend = c("Highest Traffic", "Lowest Traffic"),
       fill = c("lightblue", "lightpink"), border = "black", bty = "n", cex = 0.8)
```

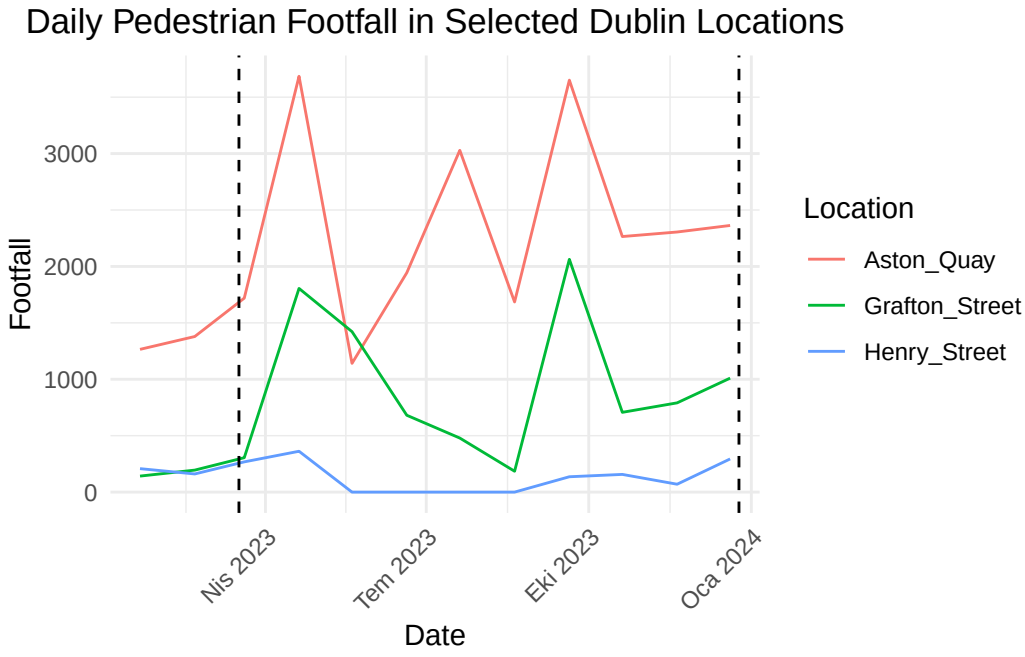


The analysis shows that **April** had the highest pedestrian traffic, indicating peak activity during this month. In contrast, **January** had the lowest total pedestrian traffic.

Question 2

To visualize daily pedestrian footfall for three selected locations and highlight St. Patrick's Day (March 17) and Christmas Day (December 25), we will use `ggplot2` to create a time series plot. First, we'll prepare the data by aggregating footfall by day, select three locations, and then plot the time series in a single plot with vertical lines marking the two holidays.

```r
# Load necessary libraries with warning suppression
suppressWarnings({
  library(ggplot2)
  library(dplyr)
  library(tidyr)})

# Prepare data for daily footfall totals by selecting three locations
daily_data <- combined_data %>%
  mutate(Date = as.Date(Time)) %>%
  group_by(Date) %>%
  summarise(
    Aston_Quay = sum(`Aston.Quay.Fitzgeralds`, na.rm = TRUE),
    Grafton_Street = sum(`Grafton.Street.CompuB`, na.rm = TRUE),
    Henry_Street = sum(`Henry.Street.Coles.Lane.Dunnes`, na.rm = TRUE))

# Define St. Patrick's Day and Christmas Day as special dates
special_dates <- as.Date(c("2023-03-17", "2023-12-25"))

# Convert data to long format for plotting
daily_data_long <- daily_data %>%
  pivot_longer(cols = c(Aston_Quay, Grafton_Street, Henry_Street),
               names_to = "Location", values_to = "Footfall")

# Create the plot
ggplot(daily_data_long, aes(x = Date, y = Footfall, color = Location)) +
  geom_line() +
  geom_vline(xintercept = as.numeric(special_dates), linetype = "dashed", color = "black") +
  labs(
    title = "Daily Pedestrian Footfall in Selected Dublin Locations",
    x = "Date",
    y = "Footfall",
    color = "Location"
  ) +
  theme_minimal() +
  theme(
    plot.title = element_text(hjust = 0.5),
    axis.text.x = element_text(angle = 45, hjust = 1))
```



The graph shows daily pedestrian footfall trends in three Dublin locations during 2023: Aston Quay (blue), Grafton Street (red), and Henry Street (purple). Aston Quay consistently has the highest footfall, with noticeable peaks around April and October. Grafton Street follows similar trends but at lower levels, while Henry Street has the lowest and most stable footfall.

Dashed lines mark St. Patrick's Day (March) and Christmas Day (December) to highlight potential shifts, though there are no visible spikes on these dates. The chart provides a clear view of seasonal footfall patterns across these locations.

Question 3

To answer this question, we'll create a table that summarizes key weather statistics by season. This table will include the minimum and maximum temperature, mean daily precipitation, and mean daily wind speed for each season.

```
# Define seasons in combined_data
combined_data <- combined_data %>%
  mutate(
    Season = case_when(
      month(Time) %in% c(12, 1, 2) ~ "Winter",
      month(Time) %in% c(3, 4, 5) ~ "Spring",
      month(Time) %in% c(6, 7, 8) ~ "Summer",
      month(Time) %in% c(9, 10, 11) ~ "Autumn"))

# Calculate seasonal statistics
seasonal_summary <- combined_data %>%
  group_by(Season) %>%
  summarise(
    Min_Temperature = min(Temperature, na.rm = TRUE),
    Max_Temperature = max(Temperature, na.rm = TRUE),
    Mean_Daily_Precipitation = mean(Precipitation, na.rm = TRUE),
    Mean_Daily_Wind_Speed = mean(WindSpeed, na.rm = TRUE))

# Display the summary table
seasonal_summary
```

```
# A tibble: 4 x 5
  Season Min_Temperature Max_Temperature Mean_Daily_Precipitation
  <chr>            <dbl>           <dbl>                    <dbl>
1 Autumn             7.4             9.5                        0
2 Spring             8.3            10.8                        0
3 Summer            11.6            16.1                        0
4 Winter             5.9            10.6                        0
# i 1 more variable: Mean_Daily_Wind_Speed <dbl>
```

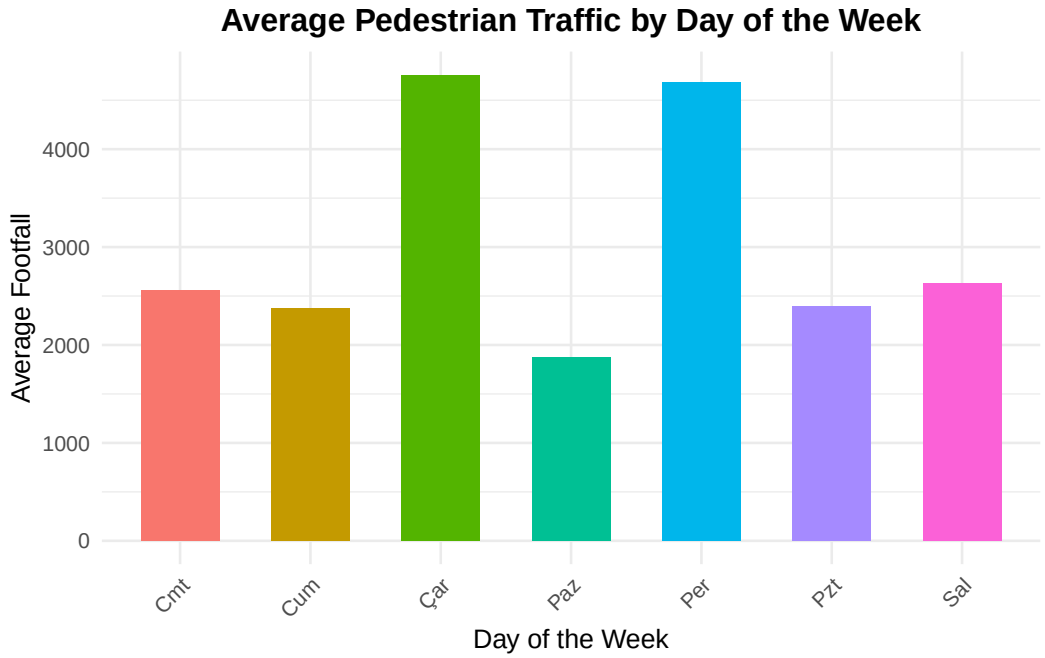
The seasonal summary shows that **Summer** has the highest temperatures, reaching up to 16.1°C, while **Winter** is the coldest with temperatures as low as 5.9°C. **Autumn** experiences the highest average wind speed at 13.67 knots, compared to the calmest season, **Summer**, with 6.33 knots. Interestingly, all seasons report 0 mm for mean daily precipitation, suggesting a dry period in the recorded data or potential gaps in precipitation measurements. Overall, the table highlights warmer temperatures in Summer, colder and windier conditions in Autumn and Winter, and no significant rainfall across all seasons.

Task 3 : Creativity

To add a new perspective to the dataset, I analyzed average pedestrian footfall by day of the week to identify peak traffic days and uncover patterns in weekly pedestrian movement. This method provides insights into how footfall varies between weekdays and weekends, offering valuable information for planning in high-footfall areas, such as optimal scheduling, resource allocation, and service operations.

```
# Load necessary libraries with warning suppression
suppressWarnings({
  library(dplyr)
  library(ggplot2)
  library(lubridate)
})

# Define the pedestrian columns
pedestrian_columns <- c("Aston.Quay.Fitzgeralds", "Grafton.Street.CompuB", "Henry.Street.Coles.Lane.Dunnes")

# Step 1: Convert `Time` to POSIXct and extract `Day_of_Week`
```

```r
combined_data <- combined_data %>%
  mutate(
    Time = as.POSIXct(Time, format = "%Y-%m-%d %H:%M:%S"),
    Day_of_Week = weekdays(Time, abbreviate = TRUE),
    Total_Footfall = rowSums(select(., all_of(pedestrian_columns)), na.rm = TRUE)
  )

# Step 2: Calculate average pedestrian footfall by day of the week
daily_footfall_by_day <- combined_data %>%
  group_by(Day_of_Week) %>%
  summarise(Average_Footfall = mean(Total_Footfall, na.rm = TRUE)) %>%
  arrange(match(Day_of_Week, c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun")))

# Step 3: Create the bar plot for average footfall by day of the week
ggplot(daily_footfall_by_day, aes(x = Day_of_Week, y = Average_Footfall, fill = Day_of_Week)) +
  geom_bar(stat = "identity", show.legend = FALSE, width = 0.6) +
  labs(
    title = "Average Pedestrian Traffic by Day of the Week",
    x = "Day of the Week",
    y = "Average Footfall"
  ) +
  theme_minimal(base_size = 10) +  # Reduce base font size for a smaller plot
  theme(
    plot.title = element_text(hjust = 0.5, face = "bold", size = 12),
    axis.title = element_text(size = 10),
    axis.text = element_text(size = 8),  # Smaller axis text
    axis.text.x = element_text(angle = 45, hjust = 1)
  )
```




```r
# Step 4: Display the table for verification
daily_footfall_by_day
```

```
# A tibble: 7 x 2
  Day_of_Week Average_Footfall
  <chr>                  <dbl>
1 Cmt                    2563
```

```
2 Cum                   2372
3 Paz                   1870
4 Per                   4680
5 Pzt                   2398.
6 Sal                   2625
7 Çar                   4758.
```

The analysis of average pedestrian footfall by day of the week highlights distinct traffic patterns. **Wednesday** and **Thursday** see the highest footfall, with averages of 4758.5 and 4680, likely due to increased midweek activity. Tuesday and Saturday show moderate traffic, while **Sunday** has the lowest average of 1870, consistent with reduced weekend commuting.

These patterns suggest that midweek is the busiest period, with traffic tapering off toward the weekend. This information is valuable for businesses and city planners for scheduling, resource allocation, and event planning to align with peak pedestrian activity.