Assignment - 2 - 24215882

```
library(ggplot2)
library(dplyr)
library(tidyverse)
library(lubridate)
library(skimr)
```

Task 1: Maniputation

1. Loading the pedestrian dataset.

```
pedestrian_data = read_csv("pedestrian_2023.csv")
pedestrian_data = pedestrian_data |>
  select(-ends_with("IN"),-ends_with("OUT"))
pedestrian_data
```

```
# A tibble: 8,760 x 27
   Time     Aston Quay/Fitzgeral~1 Baggot st lower/Wilt~2 Baggot st upper/Mesp~3
   <chr>                     <dbl>                  <dbl>                  <dbl>
 1 01/01/2~                   3700                     74                   4296
 2 01/01/2~                   4369                    100                   1466
 3 01/01/2~                   3807                     52                   1642
 4 01/01/2~                   2477                     42                    967
 5 01/01/2~                   1534                     11                    328
 6 01/01/2~                    491                      5                    291
 7 01/01/2~                    391                      4                    178
 8 01/01/2~                    864                     14                   1364
 9 01/01/2~                    708                      3                   2177
10 01/01/2~                    764                     32                   3549
# i 8,750 more rows
```

```
# i abbreviated names: 1: `Aston Quay/Fitzgeralds`,
#   2: `Baggot st lower/Wilton tce inbound`,
#   3: `Baggot st upper/Mespil rd/Bank`
# i 23 more variables: `Capel st/Mary street` <dbl>,
#   `College Green/Bank Of Ireland` <dbl>, `College Green/Church Lane` <dbl>,
#   `College st/Westmoreland st` <dbl>, `D'olier st/Burgh Quay` <dbl>, ...
```

The **read_csv()** function is used to read the csv data as a tibble.

Used the **select()** method from the **dplyr** library to filter out columns which ends with **IN and OUT**.

The size of the tibble after filtering : **8760 rows and 27 columns**.

2. Checking types for variables

```
pedestrian_data = pedestrian_data |>
  mutate(Time = dmy_hm(Time))
class(pedestrian_data$Time)
```

```
[1] "POSIXct" "POSIXt"
```

Only the **Time** variable was string instead of an appropritate data-time variable.

Using a combination of **mutate()** and **lubridate's dmy_hm()** on the data's time column, the appropriate class is set.

3. Loading the weather dataset.

```
weather_data = read_tsv("weather_2023.txt")
weather_data = weather_data |>
  rename(Rain = rain,
         Temperature = temp,
         Windspeed = wdsp,
         `Cloud Amount` = clamt)
weather_data
```

```
# A tibble: 8,760 x 5
   Time                 Rain Temperature Windspeed `Cloud Amount`
   <dttm>              <dbl>       <dbl>     <dbl>          <dbl>
 1 2023-01-01 00:00:00   0.3         6.7         6              7
 2 2023-01-01 01:00:00   0           6.2         8              6
 3 2023-01-01 02:00:00   0           4.9         7              3
 4 2023-01-01 03:00:00   0           4.4         5              6
 5 2023-01-01 04:00:00   0           4.1         3              6
 6 2023-01-01 05:00:00   0           4.4         3              3
 7 2023-01-01 06:00:00   0           4           3              6
 8 2023-01-01 07:00:00   0           4.3         4              7
 9 2023-01-01 08:00:00   0.2         5.2         3              6
10 2023-01-01 09:00:00   0           5.5         3              7
# i 8,750 more rows
```

The size of the tibble is : **8760 rows and 5 columns**

4. Factor for the cloud variable.

```
weather_data$`Cloud Amount` = factor(weather_data$`Cloud Amount`
                                     ,c(1,2,3,4,5,6,7,8,9),ordered = TRUE)
levels(weather_data$`Cloud Amount`)
```

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

```
is.ordered(weather_data$`Cloud Amount`)
```

```
[1] TRUE
```

We use the **factor()** function here to convert the Cloud Amount into a factor. We use the **ordered=True** to make it an ordered factor with levels 1 to 9.

5. skim_without_charts()

```
skim_without_charts(weather_data)
```

Table 1: Data summary

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

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Cloud Amount	34	1	TRUE	8	7: 3865, 6: 1338, 8: 733, 1: 727

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100
Rain	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

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max	median	n_unique
Time	0	1	2023-01-01	2023-12-31 23:00:00	2023-07-02 11:30:00	8760

skim_without_charts() basically provides a summary table for the different variables. The first pane provides the overall summary with the number of rows and column and the column type frequency which has the frequency for the different column types. For example, we have 1 factor, 3 numeric and 1 POSIXct (data-time) variable.

We also have separate panes for each of the different column types we have. So, we have one pane for the factor variable, one for the numeric and one for the POSIXct. Each pane provides

information relavant to the appropriate column type. For example, the numeric pane gives information about the mean, standard deviation and the quartile ranges.

6. Range for the time variables.

```
class(weather_data$Time)
```

```
[1] "POSIXct" "POSIXt"
```

The correct lubridate class is already set to for the time variable in the weather dataset.

```
all(weather_data$Time == pedestrian_data$Time)
```

```
[1] FALSE
```

We see that the timeframes dont match between the two dataframes.

```
df = weather_data |>
  filter(weather_data$Time != pedestrian_data$Time) |>
  select(Time)
indexes = which(weather_data$Time == df$Time)
indexes
```

```
[1] 2018
```

```
pedestrian_data[indexes,]
```

```
# A tibble: 1 x 27
  Time                `Aston Quay/Fitzgeralds` Baggot st lower/Wilton tce inbo~1
  <dttm>                                  <dbl>                              <dbl>
1 2023-03-26 02:00:00                        NA                                 NA
# i abbreviated name: 1: `Baggot st lower/Wilton tce inbound`
# i 24 more variables: `Baggot st upper/Mespil rd/Bank` <dbl>,
#   `Capel st/Mary street` <dbl>, `College Green/Bank Of Ireland` <dbl>,
#   `College Green/Church Lane` <dbl>, `College st/Westmoreland st` <dbl>,
#   `D'olier st/Burgh Quay` <dbl>, `Dame Street/Londis` <dbl>,
#   `Grafton st/Monsoon` <dbl>,
#   `Grafton Street / Nassau Street / Suffolk Street` <dbl>, ...
```

```
weather_data[indexes,]
```

```
# A tibble: 1 x 5
  Time                 Rain Temperature Windspeed `Cloud Amount`
  <dttm>              <dbl>       <dbl>     <dbl> <ord>
1 2023-03-26 01:00:00     0         7.4         5 5
```

We have to do deeper analysis on the dataframes to figure out the mismatch. We filter out the weather data with the filter() function to see where the mismatch is. Then we get the index of that row by passing it to the which() function. The index here came out to be 2018.

Comparing the row 2018 of weather and the pedestrian data, we see that there is indeed a mismatch in the time column and that the entire row of the pedestrian data is NA. It is safe to remove the entire column from the pedestrian dataframe.

```
pedestrian_data = pedestrian_data[-indexes[1],]
```

After removing this row, the pedestrian dataframe has : **8759 rows** and **27 columns.**

7. Join

```
data = full_join(weather_data,pedestrian_data,by="Time")
data
```

```
# A tibble: 8,760 x 31
   Time                 Rain Temperature Windspeed `Cloud Amount`
   <dttm>              <dbl>       <dbl>     <dbl> <ord>
 1 2023-01-01 00:00:00   0.3         6.7         6 7
 2 2023-01-01 01:00:00   0           6.2         8 6
 3 2023-01-01 02:00:00   0           4.9         7 3
 4 2023-01-01 03:00:00   0           4.4         5 6
 5 2023-01-01 04:00:00   0           4.1         3 6
 6 2023-01-01 05:00:00   0           4.4         3 3
 7 2023-01-01 06:00:00   0           4           3 6
 8 2023-01-01 07:00:00   0           4.3         4 7
 9 2023-01-01 08:00:00   0.2         5.2         3 6
10 2023-01-01 09:00:00   0           5.5         3 7
# i 8,750 more rows
# i 26 more variables: `Aston Quay/Fitzgeralds` <dbl>,
```

```
#   `Baggot st lower/Wilton tce inbound` <dbl>,
#   `Baggot st upper/Mespil rd/Bank` <dbl>, `Capel st/Mary street` <dbl>,
#   `College Green/Bank Of Ireland` <dbl>, `College Green/Church Lane` <dbl>,
#   `College st/Westmoreland st` <dbl>, `D'olier st/Burgh Quay` <dbl>,
#   `Dame Street/Londis` <dbl>, `Grafton st/Monsoon` <dbl>, ...
```

Since we are doing a full join, the entire 8760 rows from the weather dataframe and the 8759 rows from the pedestrian dataframe are joined. Since the row the with time value = 2023-03-26 01:00:00 is missing from the pedestrian dataframe, the columns from the pedestrian dataframe will be populated with NA values after merge.

The size of the dataset after joining is: **8760 rows and 31 columns.**

8. Adding columns

```
data = data |>
  mutate(Day = factor(day(data$Time),seq(1,31,1),ordered = TRUE),
         Month = factor(month(data$Time),seq(1,12,1),ordered = TRUE))
c(class(data$Day),class(data$Month))
```

```
[1] "ordered" "factor"  "ordered" "factor"
```

Using the **mutate()** function from dplyr, we add two new columns day and month as ordered factors. The class is set to **Ordered factor** for both the variables.

9. Relocate

```
data = data |>
  relocate(Day,Month,.after = Time)
knitr::kable(colnames(data),col.names = "Column names")
```

Column names
Time
Day
Month
Rain
Temperature

Column names
Windspeed
Cloud Amount
Aston Quay/Fitzgeralds
Baggot st lower/Wilton tce inbound
Baggot st upper/Mespil rd/Bank
Capel st/Mary street
College Green/Bank Of Ireland
College Green/Church Lane
College st/Westmoreland st
D'olier st/Burgh Quay
Dame Street/Londis
Grafton st/Monsoon
Grafton Street / Nassau Street / Suffolk Street
Grafton Street/CompuB
Grand Canal st upp/Clanwilliam place
Grand Canal st upp/Clanwilliam place/Google
Henry Street/Coles Lane/Dunnes
Mary st/Jervis st
North Wall Quay/Samuel Beckett bridge East
North Wall Quay/Samuel Beckett bridge West
O'Connell St/Parnell St/AIB
O'Connell st/Princes st North
Phibsborough Rd/Enniskerry Road
Richmond st south/Portabello Harbour inbound
Richmond st south/Portabello Harbour outbound
Talbot st/Guineys
Westmoreland Street East/Fleet street
Westmoreland Street West/Carrolls

Using dplyr **relocate()**, we switch the positions of the day and the month columns. Since Time is the first column, we use **.after = Time** to make day and month appear right after Time as the second and third columns. We display the column names as a knitr table.

Task 2: Analysis

1. Highest and lowest pedestrian traffic.

```
by_split = by(as.data.frame(data),
              data$Month,
              function(x) sum(rowSums(x[,8:ncol(x)],na.rm = TRUE)))
months = c("Jan","Feb","Mar","Apr","May","Jun",
           "Jul","Aug","Sep","Oct","Nov","Dec")
months[which.max(by_split)]
```

```
[1] "Mar"
```

```
months[which.min(by_split)]
```

```
[1] "Jun"
```

Using the by function, we pass in a factor i.e the month here to split up the data into the separate months and run the below function. The function returns **sum(rowSums(x[,8:ncol(x)], na.rm = TRUE)).** This basically takes every row and uses **rowSums()** to sum up the row from the **8 column onwards** while avoiding any **NA** values. And sums up the overall value at the end with the **sum()** function. This will happen for every month separately.

We see **which.max(by_split)** return **3** and **which_min(by_split)** return **6** here. That means we see the most overall pedestrian traffic across all the locations in the month of **March** and the least pedestrian traffic in the month of **June**.

2. Timeseries Plot

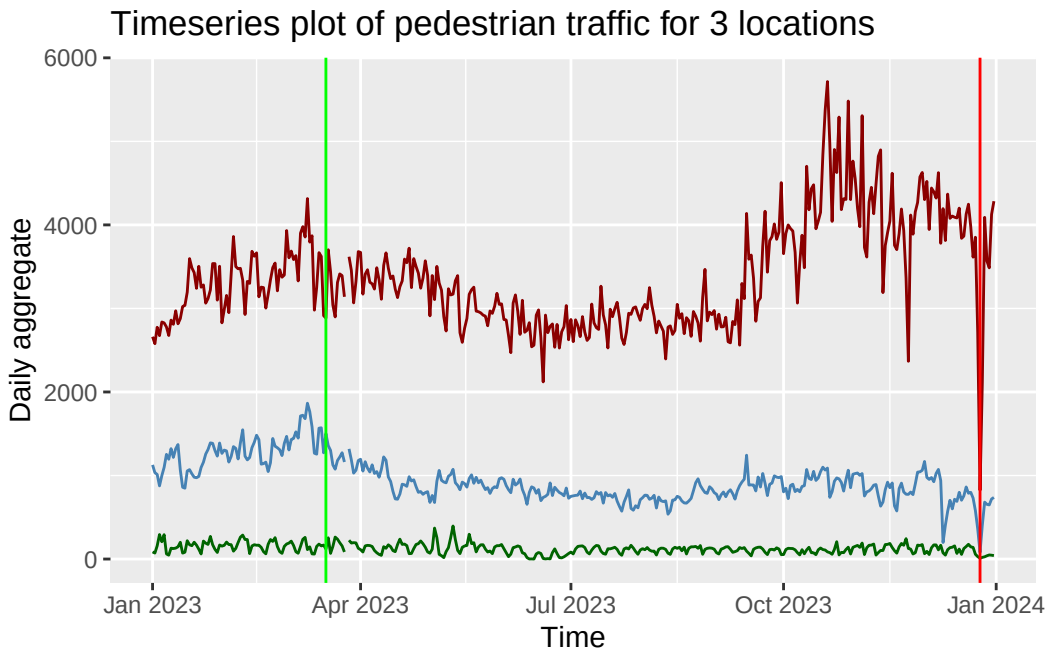
```
data_filtered = data |>
  select(`Aston Quay/Fitzgeralds`,
         `D'olier st/Burgh Quay`,
         `Baggot st lower/Wilton tce inbound`) |>
  mutate(day = as.Date(data$Time)) |>
  group_by(day) |>
  summarise(
    location1_sum = mean(`Aston Quay/Fitzgeralds`),
    location2_sum = mean(`D'olier st/Burgh Quay`),
    location3_sum = mean(`Baggot st lower/Wilton tce inbound`)
  )
data_filtered
```

```
# A tibble: 365 x 4
   day        location1_sum location2_sum location3_sum
   <date>             <dbl>         <dbl>         <dbl>
 1 2023-01-01         2661.         1127.          86
 2 2023-01-02         2578.         1034.          71.5
 3 2023-01-03         2774.         1013.         159.
 4 2023-01-04         2674.          876.         294.
 5 2023-01-05         2838.          998.         207.
 6 2023-01-06         2827.         1097.         292.
 7 2023-01-07         2778.         1253.          61.5
 8 2023-01-08         2676.         1195.          46.2
 9 2023-01-09         2860.         1322.         130.
10 2023-01-10         2810          1217.         126.
# i 355 more rows
```

We first get our data ready. We use **dplyr** chaining to select the columns for **three random locations**. We now add this new column to our selected dataframe to make it easier to group by. We need this because the original Day column is not appropriate for use here for grouping. We then group by that new column and summarise the three column's mean value instead of summing. The data is now ready.

We now plot the time series plots. We use **ggplot** on the filtered data while setting **aes(x=day)**. We then use three **geom_line()** and give **aes(y=location_sum)** for each location while also giving a different color. This will plot the three columns with respect to the dates on the graph. We also use **vline()** to add two vertical lines on 17/03/2023 and 25/12/2023 for Saint Patrick's Day and Chirstmas.

```
ggplot(data_filtered,aes(x=day)) +
  geom_line(aes(y=location1_sum),color="darkred",linewidth = 0.5) +
  geom_line(aes(y=location2_sum),color="steelblue",linewidth = 0.5) +
  geom_line(aes(y=location3_sum),color="darkgreen",linewidth = 0.5) +
  geom_vline(xintercept = dmy("17/03/2023"),color="green",linewidth = 0.5) +
  geom_vline(xintercept = dmy("25/12/2023"),color="red", linewidth = 0.5) +
  labs(title="Timeseries plot of pedestrian traffic for 3 locations",
       x="Time",
       y="Daily aggregate"
       )
```



3. Season summary

```
data["Season"] = cut(as.numeric(data$Month),c(0,2,5,8,11,12),
                     labels = c("Winter",
                                "Spring",
                                "Summer",
                                "Autumn",
                                "Winter")
                     )
summary = by(as.data.frame(data),data$Season,
  function(x) {
    temp1 = by(as.data.frame(x),x$Month,function(y) {
         day_wise_rain = by(as.data.frame(y),y$Day,function(z) {
              return(sum(z$Rain,na.rm = TRUE))
         })
         day_wise_rain = as.data.frame(day_wise_rain)

         return(sum(day_wise_rain,na.rm = TRUE))
      }
    )
    temp2 = by(as.data.frame(x),x$Month,function(y) {
```

```
        day_wise_windspeed_avg = by(as.data.frame(y),y$Day,function(z) {
             return(sum(z$Windspeed,na.rm = TRUE)/24)
        })
        day_wise_windspeed_avg = as.data.frame(day_wise_windspeed_avg)

        return(sum(day_wise_windspeed_avg,na.rm = TRUE))
      }
    )
    number_of_days = max(by(as.data.frame(x),x$Day,function(y){
      return(nrow(x))
    }))/24
    mean_daily_precipitation = sum(as.data.frame(temp1),
                                   na.rm = TRUE)/number_of_days
    mean_daily_windspeed = sum(as.data.frame(temp2),
                               na.rm = TRUE)/number_of_days
    return(
        data.frame(c(
          "Max Temperature" = max(x["Temperature"]),
          "Min Temperature" = min(x["Temperature"]),
          "Mean daily precipitation" = mean_daily_precipitation,
          "Mean daily windspeed" = mean_daily_windspeed
        ))
    )
  }
)
```

- **Adding the `Season` Column**:
  - We categorize months into seasons using the `cut()` function. The breaks (0, 2, 5, 8, 11, 12) divide months into ranges corresponding to "Winter", "Spring", "Summer", "Autumn", and "Winter".
  - The `labels` argument assigns a season to each range.
- **Using the `by()` Function for Grouped Summaries**:
  - We apply the `by()` function to split the data by season (`data$Season`) and compute summaries for each season.
- **Calculating Metrics for Each Season**:
  - **Maximum and Minimum Temperatures**: These are straightforward to calculate using the `max()` and `min()` functions within the `by()` loop for each season.
  - **Mean Daily Precipitation**:

    - The data is further grouped by month (`x$Month`) and then by day (`y$Day`) using nested `by()` functions.
    - For each day, the total rainfall is summed using `sum(z$Rain, na.rm = TRUE)`.
    - The daily totals are then aggregated to calculate the monthly total rainfall, which is finally divided by the number of days in the season to compute the mean daily precipitation.
  - **Mean Daily Windspeed**:
    - Similar to precipitation, we group the data by month and then by day.
    - For each day, the daily average windspeed is calculated as `sum(z$Windspeed, na.rm = TRUE) / 24` (assuming data is hourly).
    - These daily averages are then summed and divided by the number of days in the season to compute the mean daily windspeed.

```
summary = rbind(as.vector(summary$Winter),
                  as.vector(summary$Spring),
                  as.vector(summary$Summer),
                  as.vector(summary$Autumn)
                  )
colnames(summary) = c("Max Temperature",
                        "Min Temperature",
                        "Mean daily precipitation",
                        "Mean daily windspeed"
                        )
rownames(summary) = c("Winter","Spring","Summer","Autumn")
knitr::kable(as.data.frame(summary))
```

	Max Temperature	Min Temperature	Mean daily precipitation	Mean daily windspeed
Winter	13.8	-4.3	1.402222	10.495833
Spring	19.4	-4.3	2.527174	8.634511
Summer	24.0	3.8	3.456522	8.501812
Autumn	25.5	-2.0	3.569231	8.345696

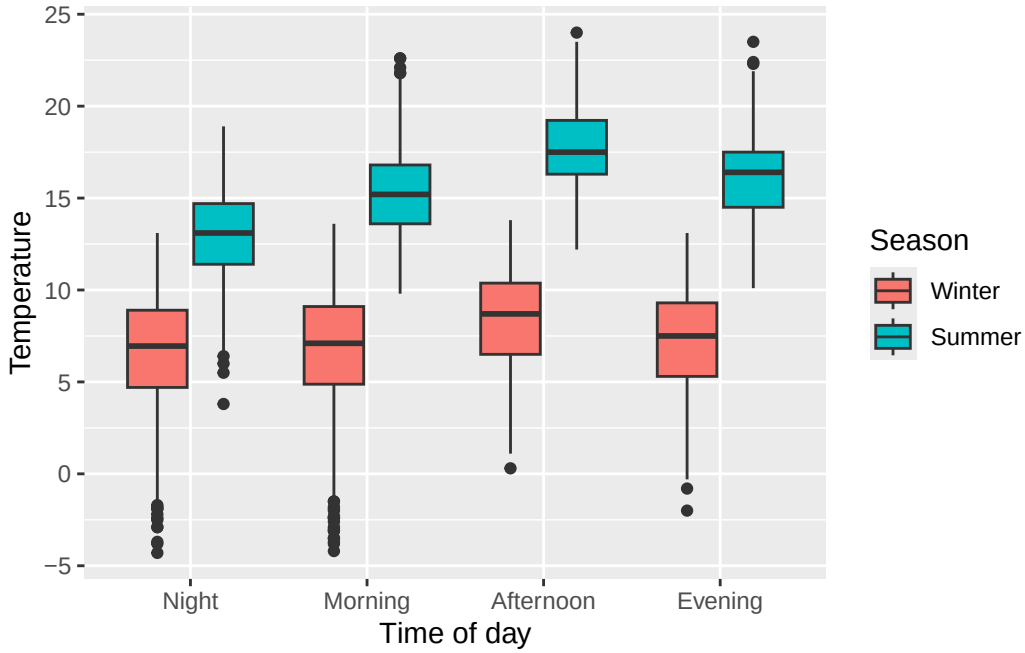
Task 3: Creativity

1. Box plots of temperature and windspeed according to Time of day for summer and winter.

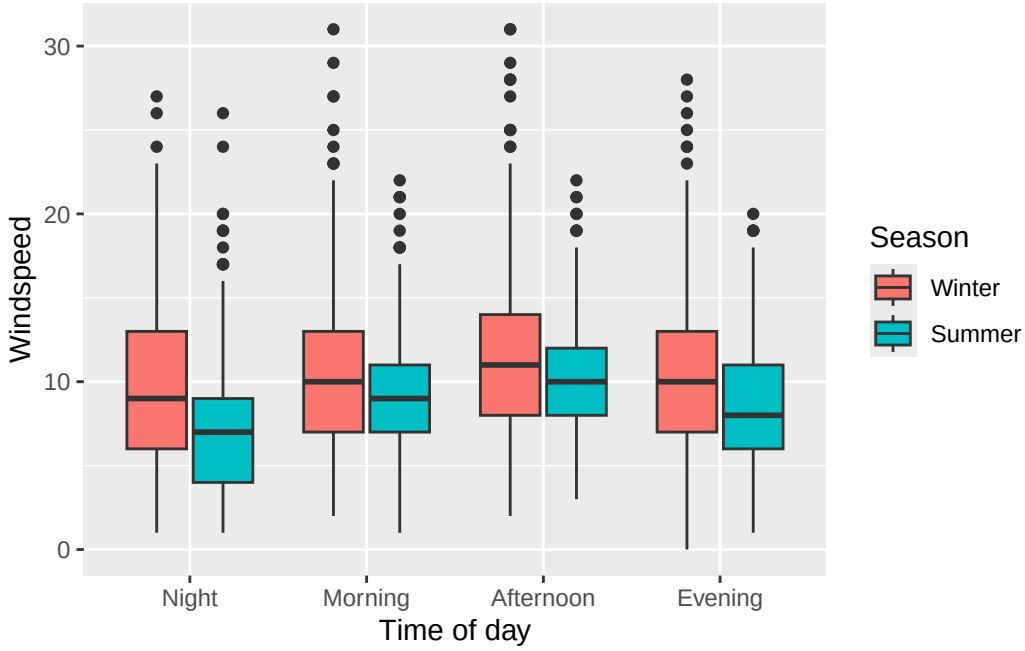
```
data["Time of day"] = cut(as.numeric(hour(data$Time)),c(-1,5,11,16,21,24),
                      labels = c("Night",
                                 "Morning",
                                 "Afternoon",
                                 "Evening",
                                 "Night"
                                 )
                      )
```

We create a new column for Time of day using the **cut()** function. We split based on the hour and decide which hours are Mornings, Afternoons, Evenings and Nights. After we have the column added, we filter the data to keep only winter and summer season records. We then plot the boxplot for the temperature data with the x axis being the time of day. We use `fill = Season` to plot separate boxplots for each season.

```
data_summer_winter = data |>
  filter(Season=="Winter" | Season == "Summer")
ggplot(data_summer_winter,aes(y=Temperature,x=`Time of day`,fill=`Season`)) +
  geom_boxplot()
```




```
ggplot(data_summer_winter,aes(y=Windspeed,x=`Time of day`,fill=`Season`)) +
  geom_boxplot()
```



We can see a clear pattern here. The temperature of winter months are consistently lesser than the summer months. Comparing the time of day, the temperatures reach a low value during nights and early mornings. For the winter months, the highest temperature is almost around the same value at any time of day. But for summer months, the highest temperature is reached during the afternoon times. We see outliers in the temperature data for the winter months. The temperatures seem to dropped way below the lower whisker level, especially during Nights and the mornings.

We can see that the windspeeds are higher in the winter months as compared to the summer months. For the winter months, we see that the upper whiskers are almost at the same point at every time of day meaning that it is more windier during the winter season regardless of the time of day. As for summer months, although not much of a difference, the windspeeds are slightly higher during the morning, afternoon and evening times compared to the night times. We can also see a lot of outliers in the windspeed data. Winter months show more amount of higher windspeeds outside the upper whiskers during mornings and afternoons.

2. Comparison of pedestrian traffic between weekdays and weekends split by time of day.

```
data = data |>
  mutate(is_weekend = wday(Time) %in% c(1,7))

data_weekend = data |>
  filter(is_weekend == TRUE) |>
  select(`D'olier st/Burgh Quay`,
         `Aston Quay/Fitzgeralds`,
         `Baggot st upper/Mespil rd/Bank`,
         `Time of day`) |>
  group_by(`Time of day`) |>
  summarise(
    "Location1 Weekend Avg" = mean(`D'olier st/Burgh Quay`,na.rm = TRUE),
    "Location2 Weekend Avg" = mean(`Aston Quay/Fitzgeralds`,na.rm = TRUE),
    "Location3 Weekend Avg" = mean(`Baggot st upper/Mespil rd/Bank`,
                                   na.rm = TRUE)
  )

data_weekday = data |>
  filter(is_weekend == FALSE) |>
  select(`D'olier st/Burgh Quay`,
         `Aston Quay/Fitzgeralds`,
         `Baggot st upper/Mespil rd/Bank`,
         `Time of day`) |>
```

```
  group_by(`Time of day`) |>
  summarise(
    "Location1 Weekday Avg" = mean(`D'olier st/Burgh Quay`,na.rm = TRUE),
    "Location2 Weekday Avg" = mean(`Aston Quay/Fitzgeralds`,na.rm = TRUE),
    "Location3 Weekday Avg" = mean(`Baggot st upper/Mespil rd/Bank`,
                                   na.rm = TRUE)
  )

data_join = full_join(data_weekend,data_weekday,join_by(`Time of day`))
```

We first use the **mutate()** function to create a new column which have **TRUE** for **weekdays** and FALSE for weekends. We then chain **dplyr** methods one by one to create our dataset for the weekend and the weekdays by **filtering** and **selecting** columns. Also we **group** by the **Time of day** and **summarise** the **mean pedestrian data** for three locations. The locations are chosen randomly here. Finally, we join the dataframe into one.

After that, using the **select()** function, we select specific columns for the locations and display the data as a table using the **knitr::kable()** function. Below are the three tables.

```
data_location1 = data_join |>
  select(`Time of day`,starts_with("location1"))
knitr::kable(data_location1)
```

Time of day	Location1 Weekend Avg	Location1 Weekday Avg
Night	591.8462	400.5087
Morning	689.9683	1193.9808
Afternoon	1290.9390	1311.2077
Evening	1222.7048	1227.8162

```
data_location2 = data_join |>
  select(`Time of day`,starts_with("location2"))
knitr::kable(data_location2)
```

Time of day	Location2 Weekend Avg	Location2 Weekday Avg
Night	2308.505	1518.963
Morning	2585.727	3776.850
Afternoon	4810.276	4610.130
Evening	4654.581	4577.087

```r
data_location3 = data_join |>
  select(`Time of day`,starts_with("location3"))
knitr::kable(data_location3)
```

Time of day	Location3 Weekend Avg	Location3 Weekday Avg
Night	2052.440	1546.808
Morning	2481.484	4490.815
Afternoon	5421.787	7124.888
Evening	5654.526	6038.679

We see a pattern here. On all three locations, the **average pedestrian footfall** during **Nights** is **higher** on the **weekends** than on the weekdays. This means more people are active during the night on weekends than on weekdays. We see an opposite behavior during Mornings, Evenings and Afternoons. The **pedestrian football** is **lesser** on the **weekends** during those times compared to the weekdays. This is justified due to the fact that more people have to go to work, schools and colleges during the weekdays.