Predicting Marathon Average Finishing Times: *The Impact of Weather and Air Quality on Marathons*

By: Krisha Bugajski-Sharp, Zachary D'Urso, Meghan Holden

Date: 12-19-2025

1 Load All Libraries (KB, ZD, MH)

```
library(here)
```

```
## here() starts at /Users/krishabugajski/git_projects/capstone-project-team-c
```

```
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
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(readr)
library(tibble)
library(knitr)
library(ggplot2)
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
library(tidyr)
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```
library(gt)
library(VIM)
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
library(rpart)
library(rpart.plot)
library(combinat)
```

```
##
## Attaching package: 'combinat'
```

```
## The following object is masked from 'package:utils':
##
##     combn
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
##     slice
```

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(fastDummies)
library(SHAPforxgboost)
```

2 Import, Clean, and Join All Raw Data (KB)

```
weather = read.csv(here("data","raw-data","weather-data.csv"))
airquality = read.csv(here("data","raw-data","air-quality-data.csv"))
boston = read.csv(here("data","raw-data","boston-marathon-data.csv"))
chicago = read.csv(here("data","raw-data","chicago-marathon-data.csv"))
nyc = read.csv(here("data","raw-data","nyc-marathon-data.csv"))
berlin = read.csv(here("data","raw-data","berlin-marathon-data.csv"))
```

Look at the raw datasets (KB)

```
str(boston)
```

```
## 'data.frame':    494326 obs. of  7 variables:
##  $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ age         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ gender      : chr  "M" "M" "M" "M" ...
##  $ official_time: chr  "2:07:15" "2:07:19" "2:08:08" "2:08:09" ...
##  $ overall     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ city        : chr  "Boston" "Boston" "Boston" "Boston" ...
##  $ year        : int  1994 1994 1994 1994 1994 1994 1994 1994 1994 1994 ...
```

```
str(berlin)
```

```
## 'data.frame':    884944 obs. of  5 variables:
##  $ YEAR   : int  1974 1974 1974 1974 1974 1974 1974 1974 1974 1974 ...
##  $ COUNTRY: chr  "" "" "" "" ...
##  $ GENDER : chr  "male" "male" "male" "male" ...
##  $ AGE    : chr  "L1" "L2" "L2" "L" ...
##  $ TIME   : chr  "02:44:53" "02:46:43" "02:48:08" "02:48:40" ...
```

```
str(nyc)
```

```
## 'data.frame':    1460286 obs. of  10 variables:
##  $ Year       : int  2024 2024 2024 2024 2024 2024 2024 2024 2024 2024 ...
##  $ Race       : chr  "NYC Marathon" "NYC Marathon" "NYC Marathon" "NYC Maratho
n" ...
##  $ Name       : chr  "Abdi Nageeye" "Evans Chebet" "Albert Korir" "Tamirat Tol
a" ...
##  $ Gender     : chr  "M" "M" "M" "M" ...
##  $ Age        : int  35 35 30 33 31 27 31 30 35 31 ...
##  $ State      : chr  "" "-0" "" "" ...
##  $ Country    : chr  "NLD" "KEN" "KEN" "ETH" ...
##  $ Overall    : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Finish.Time: chr  "02:07:39" "02:07:45" "02:08:00" "02:08:12" ...
##  $ Finish     : int  7659 7665 7680 7692 7730 7740 7761 7839 7839 7857 ...
```

```
str(chicago)
```

```
## 'data.frame':    886546 obs. of  9 variables:
##  $ Race       : chr  "Chicago" "Chicago" "Chicago" "Chicago" ...
##  $ Year       : int  2017 2017 2017 2017 2017 2017 2017 2017 2017 2017 ...
##  $ Name       : chr  "Greg McCarter" "Marcos De Sa" "Claudio Ravera" "Mitch Au
erbach" ...
##  $ Gender     : chr  "M" "M" "M" "M" ...
##  $ Age        : int  59 44 44 54 39 24 34 24 39 44 ...
##  $ Country    : chr  "USA" "USA" "ITA" "USA" ...
##  $ Overall    : int  3547 3548 3549 3550 3551 3552 3553 3554 3555 3556 ...
##  $ Finish.Time: chr  "03:25:51" "03:25:52" "03:25:52" "03:25:52" ...
##  $ Finish     : int  12351 12352 12352 12352 12353 12353 12353 12353 12354 123
54 ...
```

```
str(weather)
```

```
## 'data.frame':    115 obs. of  11 variables:
##  $ Year             : int  2025 2024 2023 2022 2021 2019 2018 2017 2016 2015
...
##  $ Date             : chr  "10-12-2025" "10-13-2024" "10-8-2023" "10-9-2022"
...
##  $ Marathon         : chr  "Chicago" "Chicago" "Chicago" "Chicago" ...
##  $ High.Temp        : int  68 67 55 71 83 55 63 80 63 79 ...
##  $ Low.Temp         : int  51 51 45 46 72 39 57 56 50 53 ...
##  $ Day.Average.Temp : num  59.9 60.2 50.1 58 75.5 ...
##  $ Precipitation    : num  0 0 0 0 0 0 0.19 0.44 0 0 ...
##  $ Average.Dew.Point: num  48.9 46.5 36 35.1 60.7 ...
##  $ Max.Wind.Speed   : int  12 23 13 14 18 23 13 21 14 18 ...
##  $ Visibility       : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ Sea.Level.Pressure: num  29.4 29.3 29.4 29.5 29.1 ...
```

```
str(airquality)
```

```
## 'data.frame':    115 obs. of  10 variables:
##  $ Year           : int  2025 2024 2023 2022 2021 2019 2018 2017 2016 2015
...
##  $ Date           : chr  "10-12-2025" "10-13-2024" "10-8-2023" "10-9-2022"
...
##  $ Marathon       : chr  "Chicago" "Chicago" "Chicago" "Chicago" ...
##  $ Overall.AQI.Value: int  42 91 43 50 72 44 48 46 67 51 ...
##  $ Main.Pollutant   : chr  "PM2.5" "PM10" "PM2.5" "PM2.5" ...
##  $ Ozone          : int  41 34 27 39 46 36 21 44 37 47 ...
##  $ PM10           : chr  "16" "91" "16" "46" ...
##  $ PM2.5          : int  42 70 43 50 72 44 48 46 67 51 ...
##  $ NO2            : int  NA 25 25 39 28 27 8 36 27 20 ...
##  $ CO             : int  NA 5 3 8 6 3 3 3 5 8 ...
```

We can see that several of the datasets have varying variable names/formats for year, marathon, gender, and chip_time. We can also see there are variables that are not necessary in several of the marathon datasets.

2.1 Standardize All Datasets (KB)

Standardize all the raw marathon datasets; clean variable names and select necessary variables.

```
boston_clean <- boston %>%
  transmute(
    year = year,
    marathon = "Boston",
    gender = gender,
    chip_time = official_time
  )

berlin_clean <- berlin %>%
  transmute(
    year = YEAR,
    marathon = "Berlin",
    gender = GENDER,
    chip_time = TIME
  )

nyc_clean <- nyc %>%
  transmute(
    year = Year,
    marathon = "NYC",
    gender = Gender,
    chip_time = Finish.Time
  )

chicago_clean <- chicago %>%
  transmute(
    year = Year,
    marathon = "Chicago",
    gender = Gender,
    chip_time = Finish.Time
  )
```

Standardize the raw weather data; format the variables names to make them clean (KB)

```
weather_clean <- weather %>%
  transmute(
    year = Year,
    marathon = Marathon,
    high_temp = High.Temp,
    low_temp = Low.Temp,
    avg_temp = Day.Average.Temp,
    precipitation = Precipitation,
    dew_point = Average.Dew.Point,
    wind_speed = Max.Wind.Speed,
    visibility = Visibility,
    sea_level_pressure = Sea.Level.Pressure
  )
```

Standardize the raw airquality data; format the variables names to make them clean (KB)

```
airquality_clean <- airquality %>%
  transmute(
    year = Year,
    marathon = Marathon,
    aqi = Overall.AQI.Value,
    main_pollutant = Main.Pollutant,
    co = as.numeric(CO),
    ozone = as.numeric(Ozone),
    pm10 = suppressWarnings(as.numeric(PM10)),
    pm25 = as.numeric(PM2.5),
    no2 = as.numeric(NO2)
  )
```

2.2 Combine the Marathon Datasets (KB)

```
# using bind_rows so we can row-wise combine and have data for each runner
marathons_all <- bind_rows(
  boston_clean,
  berlin_clean,
  nyc_clean,
  chicago_clean
)
str(marathons_all)
```

```
## 'data.frame':    3726102 obs. of  4 variables:
##  $ year     : int  1994 1994 1994 1994 1994 1994 1994 1994 1994 1994 ...
##  $ marathon : chr  "Boston" "Boston" "Boston" "Boston" ...
##  $ gender   : chr  "M" "M" "M" "M" ...
##  $ chip_time: chr  "2:07:15" "2:07:19" "2:08:08" "2:08:09" ...
```

We can see that marathons_all contains the identifying variables (*year*, *marathon*, *gender*), and the outcome variable *chip_time*.

Select only years we need (1996 to 2025) (KB)

```
marathons_all <- marathons_all %>%
  filter(year >= 1996 & year <= 2025)

unique(marathons_all$year)
```

```
##  [1] 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 201
0
## [16] 2011 2012 2014 2015 2016 2017 2018 2019 2013 2024 2022 2021 2023
```

We can that the merged marathon data set now contains years from 1996 to 2024, as most don't have data up to 2025.

2.3 Chip-Time Cleaning Function (KB)

Need chip_seconds to find the average finishing times (pulled from chatgbt)

```
# Cleans and standardizes chip-time values by converting Excel-style numeric
# times to HH:MM:SS, trimming and formatting text times, replacing invalid
# entries with NA, and padding missing leading zeros.

clean_chip_time <- function(x) {

  # Convert numeric Excel-style times to character HH:MM:SS
  x_numeric <- suppressWarnings(as.numeric(x))
  is_fraction <- !is.na(x_numeric) & x_numeric < 1 & x_numeric > 0

  x[is_fraction] <- format(
    as.POSIXct("1970-01-01", tz = "UTC") + x_numeric[is_fraction] * 86400,
    "%H:%M:%S"
  )

  # Everything else treat as text and clean
  x <- as.character(x)
  x <- trimws(x)

  # Replace known invalid strings with NA
  invalid <- c("", "NA", "N/A", "—", "-", "DNF", "DNS", "DQ", "no time", "No Tim
e", "NO TIME")
  x[x %in% invalid] <- NA

  # Pad missing zeros (H:MM:SS → HH:MM:SS, etc.)
  x <- gsub("^([0-9]):", "0\\1:", x)
  x <- gsub(":([0-9]):", ":0\\1:", x)
  x <- gsub(":([0-9])$", ":0\\1", x)

  return(x)
}
```

Clean chip_time; using `clean_chip_time` function (KB)

```
marathons_all <- marathons_all %>%
  mutate(chip_time_clean = clean_chip_time(chip_time))
```

Convert the cleaned chip_time values (HH:MM:SS) to hms() and period_to_seconds() and gets a new column chip_seconds (KB)

```
marathons_all <- marathons_all %>%
  mutate(
    chip_seconds = suppressWarnings(period_to_seconds(hms(chip_time_clean)))
  )

head(marathons_all)
```

```
##   year marathon gender chip_time chip_time_clean chip_seconds
## 1 1996   Boston      M   2:09:15        02:09:15         7755
## 2 1996   Boston      M   2:09:26        02:09:26         7766
## 3 1996   Boston      M   2:09:51        02:09:51         7791
## 4 1996   Boston      M   2:10:03        02:10:03         7803
## 5 1996   Boston      M   2:10:09        02:10:09         7809
## 6 1996   Boston      M   2:10:21        02:10:21         7821
```

We can see `chip_time` with the original data, `chip_time_clean` with the uniform cleaned data across all marathons, and `chip_seconds` with the total time in seconds.

2.4 Remove Missing Finish Times (KB)

Need to see what we want to do with these missing values and where they are coming from (KB)

```
marathons_all %>%
  summarize(missing_finish_times = sum(is.na(chip_seconds)))
```

```
##   missing_finish_times
## 1                    3
```

```
marathons_all %>%
  filter(is.na(chip_seconds))
```

```
##   year marathon gender chip_time chip_time_clean chip_seconds
## 1 1996   Berlin   male   no time           <NA>           NA
## 2 1996   Berlin   male   no time           <NA>           NA
## 3 1998   Berlin   male   no time           <NA>           NA
```

We can see that there are only 3 rows with missing chip_times.

Remove missing finish times. This is safe because we only use average finishing times in our model, so individual missing times do not matter. Also there are only 3 total

```
marathons_all <- marathons_all %>%
  filter(!is.na(chip_seconds))
```

2.5 Label Gender (KB)

Label as 'male', 'female', or 'unknown', and we decided that nonbinary falls under female.

```
# Check all unique names under gender
unique(marathons_all$gender)
```

```
## [1] "M"      "F"      "U"      "m"      "male"   "female" "W"      "X"
## [9] ""
```

We can see that there are a lot of ways 'male', 'female', 'nonbinary', and 'unknown' are labled.

Make all the genders uniform and standardized (KB)

```
marathons_all <- marathons_all %>%
  mutate(
    gender = tolower(gender),
    gender = case_when(
      gender %in% c("male", "m") ~ "male",
      gender %in% c("female", "f", "w", "x", "nonbinary", "nb") ~ "female",
      TRUE ~ "unknown"
    )
  )

table(marathons_all$gender)
```

```
##
##  female    male unknown
## 1132049 2052858      56
```

We can see that now we only have three genders, `female` , `male` , and `unknown` which consist of 56 rows.

Remove unknowns since we have only 56 unknowns, and they will not help the model and most likely add noise: (KB)

```
marathons_all <- marathons_all %>%
  filter(gender != "unknown")

table(marathons_all$gender)
```

```
##
##  female    male
## 1132049 2052858
```

We successfully removed the unknowns, leaving us with the desired `female` and `male` .

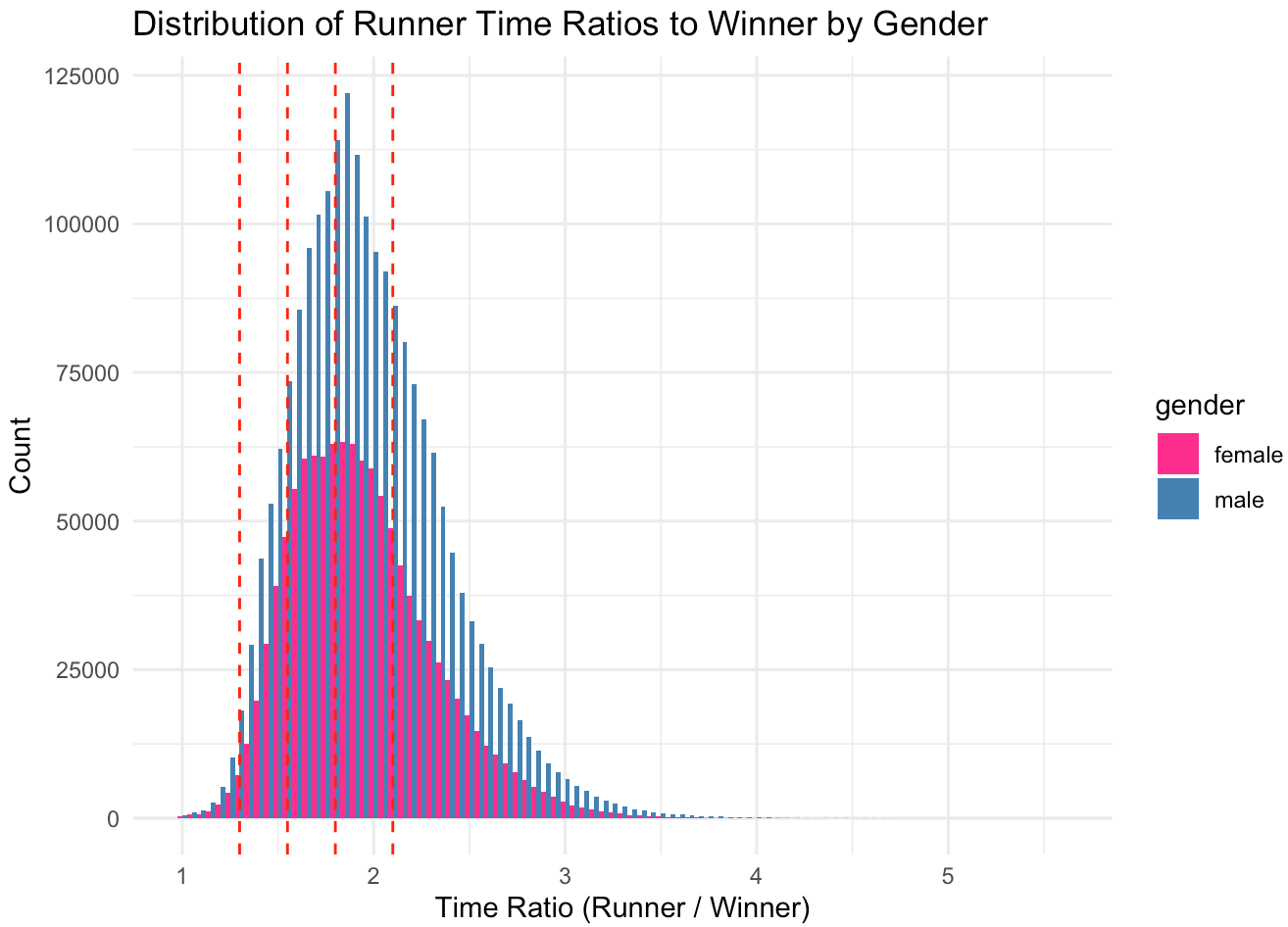
2.6 Create Preformance Subgroups (KB)

Create a variable called winner_time, that consist of the winners or best finishing for that year. Then create a variable called time_ratio (), that consists of chip_seconds / winner_time.

```
# Add winner_time for each marathon-year-gender
runners <- marathons_all %>%
  group_by(marathon, year, gender) %>%
  mutate(winner_time = min(chip_seconds, na.rm = TRUE)) %>%
  ungroup() %>%
  mutate(time_ratio = chip_seconds / winner_time)
```

Create subgroups based on time_ratio. Look at a histogram to adjust the boundaries of the subgroups based on the distribution of the histogram of time_ratio and clear clusters. (KB)

```
runners <- runners %>%
  mutate(
    subgroup = case_when(
      time_ratio <= 1.30 ~ "elite",
      time_ratio > 1.30 & time_ratio <= 1.55 ~ "competitive",
      time_ratio > 1.55 & time_ratio <= 1.80 ~ "average",
      time_ratio > 1.80 & time_ratio <= 2.10 ~ "recreational",
      time_ratio > 2.10 ~ "slow",
      TRUE ~ NA_character_
    ),
    subgroup = factor(subgroup, levels = c("elite", "competitive", "average", "re
creational", "slow"))
  )

# Create histogram of ratio_time amongst male and female runners with boundaries
for subgroups in red
ggplot(runners, aes(x = time_ratio, fill = gender)) +
  geom_histogram(binwidth = 0.05, position = "dodge") +
  scale_fill_manual(values = c("female" = "deeppink", "male" = "steelblue")) +
  geom_vline(xintercept = c(1.3, 1.55, 1.80, 2.10), linetype = "dashed", color =
"red") +
  labs(
    title = "Distribution of Runner Time Ratios to Winner by Gender",
    x = "Time Ratio (Runner / Winner)",
    y = "Count"
  ) +
  theme_minimal()
```



Looking at the histogram of the time_ratio above, we can see that there is a clear clustering in runner-to-winner time ratios and a long right-skewed tail, which helped with the placement of the performance thresholds/ vertical dashed red lines (1.30, 1.55, 1.80, 2.10). We can also see that the distribution for both male and female follow a similar shape, with overall less female relative male runners. Overall, this figure supports the use of ratio-based performance categories by illustrating natural clustering and skewness in marathon finishing times.

Now we can further divide the subgroups created by genders. (KB)

```
runners %>%
  group_by(gender, subgroup) %>%
  summarise(count = n()) %>%
  arrange(gender, subgroup)
```

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

```
## # A tibble: 10 × 3
## # Groups:   gender [2]
##    gender subgroup      count
##    <chr>  <fct>         <int>
##  1 female elite         12412
##  2 female competitive  127521
##  3 female average      293867
##  4 female recreational 356394
##  5 female slow         341855
##  6 male   elite         29047
##  7 male   competitive  233651
##  8 male   average      482666
##  9 male   recreational 624377
## 10 male   slow         683117
```

We can see that the subgroups now have groups for `male` and `female`. We can see that the elite groups both has less runners and male runners have more runners in each group overall. We will leave this as is because these cutoffs are not arbitrary, and they follow the natural shape of the data rather than relying on fixed percentile breaks. Weighting could later be applied; however, it is not initially because subgroup effects are not the primary focus.

2.7 Compute Average Finishing Time per Subgroup (KB)

```
avg_times <- runners %>%
  group_by(marathon, year, gender, subgroup) %>%
  summarize(
    n = n(),  # number of runners in each subgroup
    avg_chip_seconds = mean(chip_seconds, na.rm = TRUE),
    .groups = "drop"
  )

avg_times
```

```
## # A tibble: 1,005 × 6
##    marathon  year gender subgroup         n avg_chip_seconds
##    <chr>    <int> <chr>  <fct>        <int>            <dbl>
##  1 Berlin    1996 female elite          131           10641.
##  2 Berlin    1996 female competitive    541           12733.
##  3 Berlin    1996 female average        901           14707.
##  4 Berlin    1996 female recreational   402           16821.
##  5 Berlin    1996 female slow            60           19054.
##  6 Berlin    1996 male   elite          737            9468.
##  7 Berlin    1996 male   competitive   3416           11173.
##  8 Berlin    1996 male   average       5411           12974.
##  9 Berlin    1996 male   recreational  3439           14841.
## 10 Berlin    1996 male   slow           894           17374.
## # ℹ 995 more rows
```

We now have a dataset that contains 1005 rows and 6 columns/ variables. This means that some of the years must be missing for some of the marathon as we know previously when collecting the raw data. Berlin goes from (1996 to 2019), Boston goes from (1996 to 2019), Chicago goes from (1996 to 2023), and NYC goes from (1996 to 2024). Also, after looking through the dataset more, we can see that Berlin is missing data from all female subgroups in 2019 and NYC is missing all male subgroups for 2024. We will leave the data as is because having missing years and genders is not our primary focus, as our focus is on having as much data for finishing/chip times as possible.

2.8 Left-Join Weather and Air Quality Data with Marathon Datasets (KB)

```
final_data <- avg_times %>%
  left_join(weather_clean, by = c("year", "marathon")) %>%
  left_join(airquality_clean, by = c("year", "marathon"))

# move n (number of individuals representing each group) to the first column for
neatness
final_data <- final_data %>%
  select(n, everything())

final_data
```

```
## # A tibble: 1,005 × 21
##        n marathon  year gender subgroup     avg_chip_seconds high_temp low_tem
p
##    <int> <chr>    <int> <chr>  <fct>                  <dbl>     <int>    <int
>
##  1   131 Berlin    1996 female elite                 10641.        57       4
4
##  2   541 Berlin    1996 female competitive           12733.        57       4
4
##  3   901 Berlin    1996 female average               14707.        57       4
4
##  4   402 Berlin    1996 female recreational          16821.        57       4
4
##  5    60 Berlin    1996 female slow                  19054.        57       4
4
##  6   737 Berlin    1996 male   elite                  9468.        57       4
4
##  7  3416 Berlin    1996 male   competitive           11173.        57       4
4
##  8  5411 Berlin    1996 male   average               12974.        57       4
4
##  9  3439 Berlin    1996 male   recreational          14841.        57       4
4
## 10   894 Berlin    1996 male   slow                  17374.        57       4
4
## # ℹ 995 more rows
## # ℹ 13 more variables: avg_temp <dbl>, precipitation <dbl>, dew_point <dbl>,
## #   wind_speed <int>, visibility <dbl>, sea_level_pressure <dbl>, aqi <int>,
## #   main_pollutant <chr>, co <dbl>, ozone <dbl>, pm10 <dbl>, pm25 <dbl>,
## #   no2 <dbl>
```

```
str(final_data)
```

```
## tibble [1,005 × 21] (S3: tbl_df/tbl/data.frame)
##  $ n                 : int [1:1005] 131 541 901 402 60 737 3416 5411 3439 894
...
##  $ marathon          : chr [1:1005] "Berlin" "Berlin" "Berlin" "Berlin" ...
##  $ year              : int [1:1005] 1996 1996 1996 1996 1996 1996 1996 1996 19
96 1996 ...
##  $ gender            : chr [1:1005] "female" "female" "female" "female" ...
##  $ subgroup          : Factor w/ 5 levels "elite","competitive",..: 1 2 3 4 5
1 2 3 4 5 ...
##  $ avg_chip_seconds  : num [1:1005] 10641 12733 14707 16821 19054 ...
##  $ high_temp         : int [1:1005] 57 57 57 57 57 57 57 57 57 57 ...
##  $ low_temp          : int [1:1005] 44 44 44 44 44 44 44 44 44 44 ...
##  $ avg_temp          : num [1:1005] 52.3 52.3 52.3 52.3 52.3 ...
##  $ precipitation     : num [1:1005] 0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.
23 0.23 ...
##  $ dew_point         : num [1:1005] 48.6 48.6 48.6 48.6 48.6 ...
##  $ wind_speed        : int [1:1005] 10 10 10 10 10 10 10 10 10 10 ...
##  $ visibility        : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ sea_level_pressure: num [1:1005] 30.1 30.1 30.1 30.1 30.1 ...
##  $ aqi               : int [1:1005] 11 11 11 11 11 11 11 11 11 11 ...
##  $ main_pollutant    : chr [1:1005] "NO2" "NO2" "NO2" "NO2" ...
##  $ co                : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ ozone             : num [1:1005] 10 10 10 10 10 10 10 10 10 10 ...
##  $ pm10              : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ pm25              : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ no2               : num [1:1005] 11 11 11 11 11 11 11 11 11 11 ...
```

We can see that all the datasets were merged into one final_data successfully. We now have a new column n that counts the amount of runners in each group for future computations and potential weighing if needed.

3 Exploratory Data Analysis (EDA): Tables and Figures (KB, ZD, MH)

3.1 Table 1: Continuous Variable Summary (ZD)

```
# Select the continuous variables
continuous_summary <- final_data %>%
  select(avg_chip_seconds, high_temp, low_temp, avg_temp, precipitation,
         dew_point, wind_speed, visibility, sea_level_pressure, aqi, pm10, pm25,
no2, ozone, co) %>%
  summarise_all(list(
    mean = ~mean(., na.rm = TRUE),
    median = ~median(., na.rm = TRUE),
    sd = ~sd(., na.rm = TRUE),
    min = ~min(., na.rm = TRUE),
    max = ~max(., na.rm = TRUE)
  ))
continuous_summary
```

```
## # A tibble: 1 × 75
##   avg_chip_seconds_mean high_temp_mean low_temp_mean avg_temp_mean
##                   <dbl>          <dbl>         <dbl>         <dbl>
## 1                14081.           61.6          46.0          53.7
## # ℹ 71 more variables: precipitation_mean <dbl>, dew_point_mean <dbl>,
## #   wind_speed_mean <dbl>, visibility_mean <dbl>,
## #   sea_level_pressure_mean <dbl>, aqi_mean <dbl>, pm10_mean <dbl>,
## #   pm25_mean <dbl>, no2_mean <dbl>, ozone_mean <dbl>, co_mean <dbl>,
## #   avg_chip_seconds_median <dbl>, high_temp_median <int>,
## #   low_temp_median <int>, avg_temp_median <dbl>, precipitation_median <dbl>,
## #   dew_point_median <dbl>, wind_speed_median <int>, visibility_median <dbl>,
…
```

3.2 Table 2: Continuous Variables by Subgroup and

Gender (ZD)

```
# Select continuous variables
continuous_summary_grouped <- final_data %>%
  group_by(gender, subgroup) %>% # group by gender and subgroup
  summarise(
    avg_chip_seconds_mean = mean(avg_chip_seconds, na.rm = TRUE),
    avg_chip_seconds_sd = sd(avg_chip_seconds, na.rm = TRUE),
    high_temp_mean = mean(high_temp, na.rm = TRUE),
    high_temp_sd = sd(high_temp, na.rm = TRUE),
    low_temp_mean = mean(low_temp, na.rm = TRUE),
    low_temp_sd = sd(low_temp, na.rm = TRUE),
    avg_temp_mean = mean(avg_temp, na.rm = TRUE),
    avg_temp_sd = sd(avg_temp, na.rm = TRUE),
    precipitation_mean = mean(precipitation, na.rm = TRUE),
    precipitation_sd = sd(precipitation, na.rm = TRUE),
    dew_point_mean = mean(dew_point, na.rm = TRUE),
    dew_point_sd = sd(dew_point, na.rm = TRUE),
    wind_speed_mean = mean(wind_speed, na.rm = TRUE),
    wind_speed_sd = sd(wind_speed, na.rm = TRUE),
    visibility_mean = mean(visibility, na.rm = TRUE),
    visibility_sd = sd(visibility, na.rm = TRUE),
    sea_level_pressure_mean = mean(sea_level_pressure, na.rm = TRUE),
    sea_level_pressure_sd = sd(sea_level_pressure, na.rm = TRUE),
    aqi_mean = mean(aqi, na.rm = TRUE),
    aqi_sd = sd(aqi, na.rm = TRUE),
    pm10_mean = mean(pm10, na.rm = TRUE),
    pm10_sd = sd(pm10, na.rm = TRUE),
    pm25_mean = mean(pm25, na.rm = TRUE),
    pm25_sd = sd(pm25, na.rm = TRUE),
    no2_mean = mean(no2, na.rm = TRUE),
    no2_sd = sd(no2, na.rm = TRUE),
    ozone_mean = mean(ozone, na.rm = TRUE),
    ozone_sd = sd(ozone, na.rm = TRUE),
    co_mean = mean(co, na.rm = TRUE),
    co_sd = sd(co, na.rm = TRUE)
  ) %>%
  arrange(gender, subgroup)
```

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

```
continuous_summary_grouped
```

```
## # A tibble: 10 × 32
## # Groups:   gender [2]
##    gender subgroup     avg_chip_seconds_mean avg_chip_seconds_sd high_temp_mea
n
##    <chr>  <fct>                        <dbl>               <dbl>         <dbl
>
##  1 female elite                       10426.                334.          61.
6
##  2 female competitive                 12612.                319.          61.
6
##  3 female average                     14479.                386.          61.
6
##  4 female recreational                16718.                449.          61.
6
##  5 female slow                        20177.                810.          61.
6
##  6 male   elite                        9350.                223.          61.
6
##  7 male   competitive                 11101.                225.          61.
6
##  8 male   average                     12870.                263.          61.
6
##  9 male   recreational                14834.                317.          61.
6
## 10 male   slow                        18278.                615.          61.
6
## # ℹ 27 more variables: high_temp_sd <dbl>, low_temp_mean <dbl>,
## #   low_temp_sd <dbl>, avg_temp_mean <dbl>, avg_temp_sd <dbl>,
## #   precipitation_mean <dbl>, precipitation_sd <dbl>, dew_point_mean <dbl>,
## #   dew_point_sd <dbl>, wind_speed_mean <dbl>, wind_speed_sd <dbl>,
## #   visibility_mean <dbl>, visibility_sd <dbl>, sea_level_pressure_mean <dbl>,
## #   sea_level_pressure_sd <dbl>, aqi_mean <dbl>, aqi_sd <dbl>, pm10_mean <dbl
>,
## #   pm10_sd <dbl>, pm25_mean <dbl>, pm25_sd <dbl>, no2_mean <dbl>, …
```

3.3 Table 3: Countinuous Variables Overall (KB)

Select the continuous variables to make a clean table:

```
# Select continuous variables
continuous_vars <- final_data %>%
  select(
    avg_chip_seconds,
    high_temp,
    low_temp,
    avg_temp,
    dew_point,
    wind_speed,
    visibility,
    sea_level_pressure,
    aqi,
    co,
    ozone,
    pm10,
    pm25,
    no2
  )


str(continuous_vars)
```

```
## tibble [1,005 × 14] (S3: tbl_df/tbl/data.frame)
##  $ avg_chip_seconds  : num [1:1005] 10641 12733 14707 16821 19054 ...
##  $ high_temp         : int [1:1005] 57 57 57 57 57 57 57 57 57 57 ...
##  $ low_temp          : int [1:1005] 44 44 44 44 44 44 44 44 44 44 ...
##  $ avg_temp          : num [1:1005] 52.3 52.3 52.3 52.3 52.3 ...
##  $ dew_point         : num [1:1005] 48.6 48.6 48.6 48.6 48.6 ...
##  $ wind_speed        : int [1:1005] 10 10 10 10 10 10 10 10 10 10 ...
##  $ visibility        : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ sea_level_pressure: num [1:1005] 30.1 30.1 30.1 30.1 30.1 ...
##  $ aqi               : int [1:1005] 11 11 11 11 11 11 11 11 11 11 ...
##  $ co                : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ ozone             : num [1:1005] 10 10 10 10 10 10 10 10 10 10 ...
##  $ pm10              : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ pm25              : num [1:1005] NA NA NA NA NA NA NA NA NA NA ...
##  $ no2               : num [1:1005] 11 11 11 11 11 11 11 11 11 11 ...
```

We can see that there are a total of 14 continuous variables.

Find Statistics for only the continuous variables: (KB)

```
continuous_only <- continuous_vars %>%
  select(where(is.numeric))

continuous_summary <- describe(continuous_only) %>%
  as.data.frame() %>%
  select(
    mean,
    sd,
    median,
    min,
    max,
    skew,
    kurtosis,
    n
  )
continuous_summary
```

```
##                            mean           sd   median      min      max
## avg_chip_seconds   14080.542935 3355.8623052 13566.35 8912.405 21729.70
## high_temp             61.621891   10.4803135    61.00   44.000    88.00
## low_temp              46.009950    7.9703072    45.00   28.000    72.00
## avg_temp              53.718209    8.6975811    52.88   29.580    79.35
## dew_point             41.023831   10.7187978    42.40   21.130    65.22
## wind_speed            13.119403    6.7699588    13.00    3.000    39.00
## visibility             9.848072    1.4872173    10.00    6.060    20.00
## sea_level_pressure    29.925970    0.3391476    29.98   29.130    30.54
## aqi                   50.572139   21.7623899    52.00   11.000   119.00
## co                    13.948052   11.7791733     9.00    2.000    56.00
## ozone                 36.179104   18.0530913    34.00    7.000   100.00
## pm10                  23.576000   13.0305755    21.00    5.000    69.00
## pm25                  57.510345   16.7892845    53.00   21.000   119.00
## no2                   31.915423   14.9253725    32.00    7.000    66.00
##                           skew   kurtosis    n
## avg_chip_seconds    0.39027498 -0.8608475 1005
## high_temp           0.57757331 -0.3573111 1005
## low_temp            0.77231953  1.0186755 1005
## avg_temp            0.42260873  0.3116392 1005
## dew_point          -0.08246867 -0.9095537 1005
## wind_speed          0.78517635  1.0872273 1005
## visibility          2.79083334 25.7005389  830
## sea_level_pressure -0.41101449 -0.7527869 1005
## aqi                 0.32111228  0.2198760 1005
## co                  1.58769285  2.1592651  770
## ozone               1.22285653  2.3430336 1005
## pm10                0.89184403  0.8083714  625
## pm25                0.85098957  1.3911927  725
## no2                 0.15839726 -0.7783209 1005
```

Add continuous variables to a clean table: (KB)

Table: Descriptive Statistics for Continuous Variables

variable	mean	sd	median	min	max	skew	kurtosis	n
avg_chip_seconds	15631.62	2938.88	15284.20	10304.77	23679.03	0.51	-0.39	1005
high_temp	61.62	10.48	61.00	44.00	88.00	0.58	-0.36	1005
low_temp	46.01	7.97	45.00	28.00	72.00	0.77	1.02	1005
avg_temp	53.72	8.70	52.88	29.58	79.35	0.42	0.31	1005
dew_point	41.02	10.72	42.40	21.13	65.22	-0.08	-0.91	1005
wind_speed	13.12	6.77	13.00	3.00	39.00	0.79	1.09	1005
visibility	9.85	1.49	10.00	6.06	20.00	2.79	25.70	830
sea_level_pressure	29.93	0.34	29.98	29.13	30.54	-0.41	-0.75	1005
aqi	50.57	21.76	52.00	11.00	119.00	0.32	0.22	1005
co	13.95	11.78	9.00	2.00	56.00	1.59	2.16	770
ozone	36.18	18.05	34.00	7.00	100.00	1.22	2.34	1005
pm10	23.58	13.03	21.00	5.00	69.00	0.89	0.81	625
pm25	57.51	16.79	53.00	21.00	119.00	0.85	1.39	725
no2	31.92	14.93	32.00	7.00	66.00	0.16	-0.78	1005

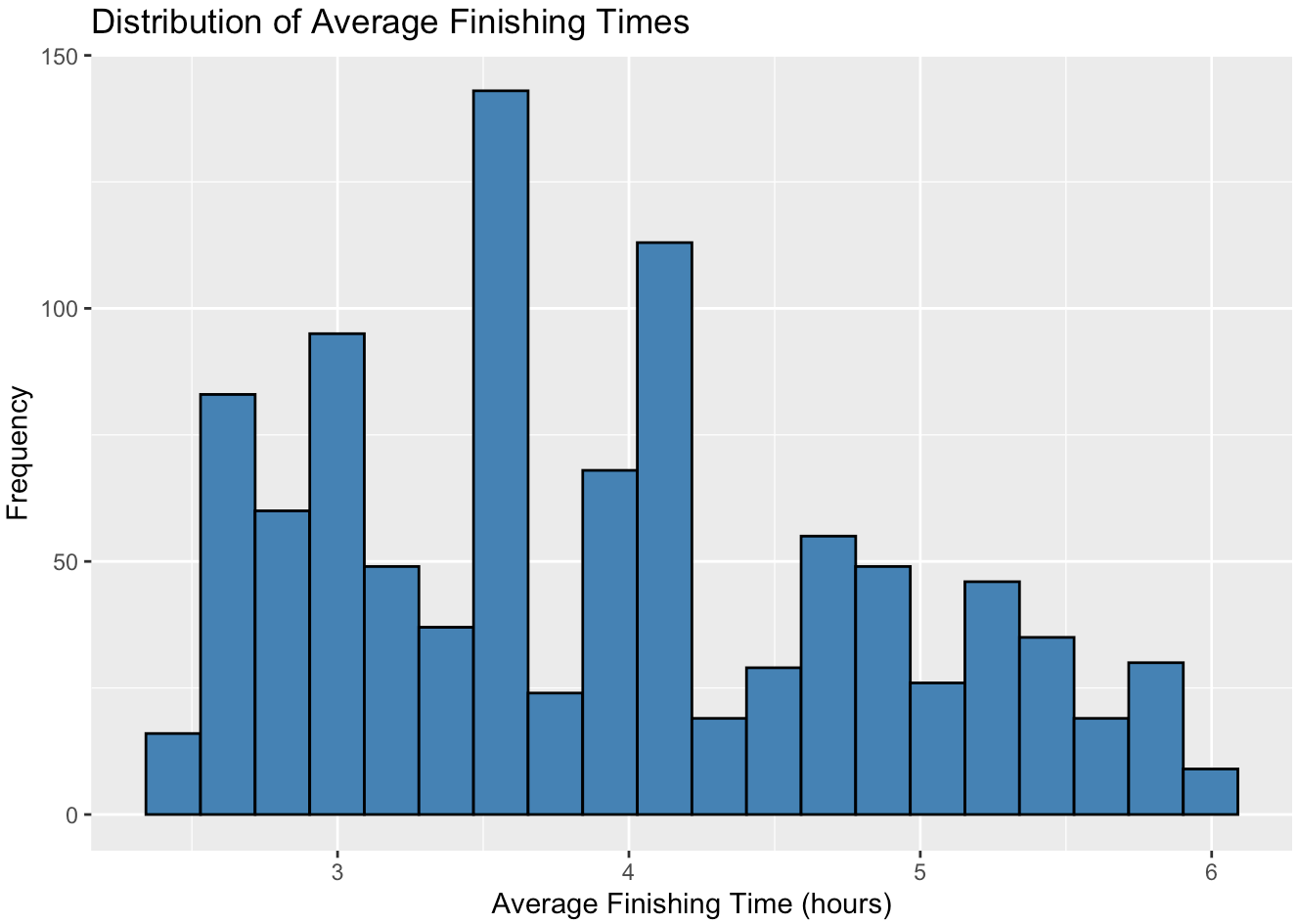
The table above summarizes key descriptive statistics for continuous variables in the dataset, including average marathon chip time (in seconds), temperature measures, humidity-related metrics (dew point), wind speed, visibility, air pressure, and air quality indicators (aqi, co, ozone, pm10, pm2.5, no2). For each variable, the table reports its mean, standard deviation, median, minimum, maximum, skewness, kurtosis, and sample size, giving an overview of central tendency, variability, distribution shape, and data completeness.

All the variables show reasonable ranges and central values for weather and airquality conditions during the marathon events. We can see that chip times cluster around 4 hours (about 15,600 seconds) with moderate variability. Temperature measures (high, low, and average) fall within typical seasonal ranges, while air-quality variables such as ozone, pm10, pm2.5, and co show right-skewed distributions, indicating occasional higher pollution days. Visibility also shows strong right skew and high kurtosis, suggesting a few unusually high values compared with the rest of the data. Overall, most variables are moderately skewed, with some (like visibility and co) showing heavier tails.

3.4 Figure 1: Distribution of Average Finishing Times (ZD)

Create a histogram for average finishing time

```
library(ggplot2) # load plotting library
ggplot(final_data, aes(x = avg_chip_seconds / 3600)) + # convert from seconds to
hours
  geom_histogram(bins = 20, fill = "steelblue", color = "black") + # create a his
togram
  labs(title = "Distribution of Average Finishing Times", # generate labels
    x = "Average Finishing Time (hours)",
    y = "Frequency"
  )
```



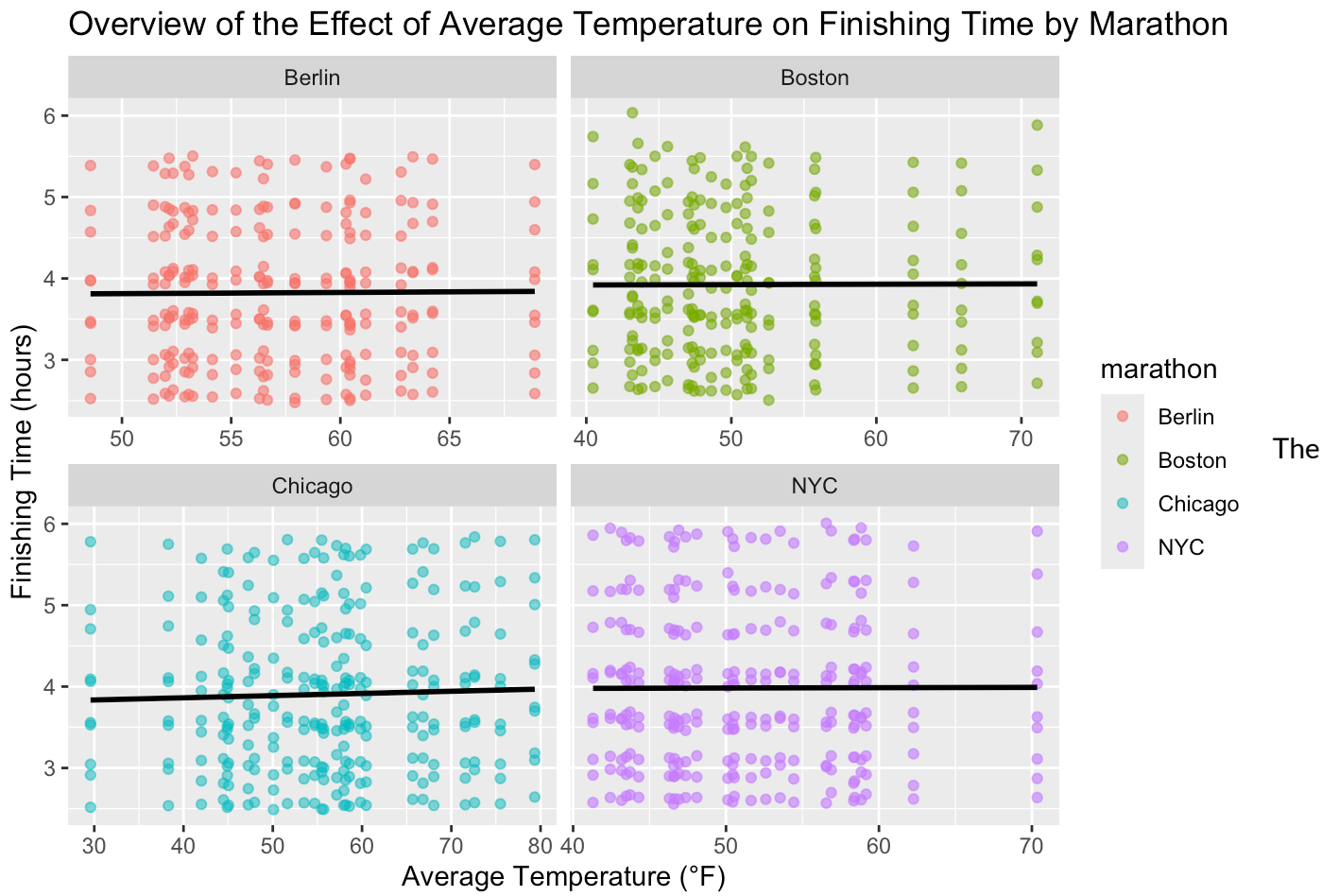
The figure above displays the overall distribution of average finishing times among all runners in all races via a histogram. The distribution is right-skewed, where a large portion of runners are clustered between the 3 and 4-hour mark. We can see that there are fewer runners with longer times (slow/recreational runners), extending the tail of the distribution. A strong majority of the runners have finishing times between the 3 and 4 hour mark.

3.5 Figure 2: Effect of Average Temperature on

Finishing Time by Marathon (ZD)

```
ggplot(final_data, aes(x = avg_temp, y = avg_chip_seconds / 3600, color = maratho
n)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  facet_wrap(~ marathon, scales = "free_x") + # each marathon has own x-axis scal
e
  labs(title = "Overview of the Effect of Average Temperature on Finishing Time b
y Marathon",
    x = "Average Temperature (°F)",
    y = "Finishing Time (hours)"
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



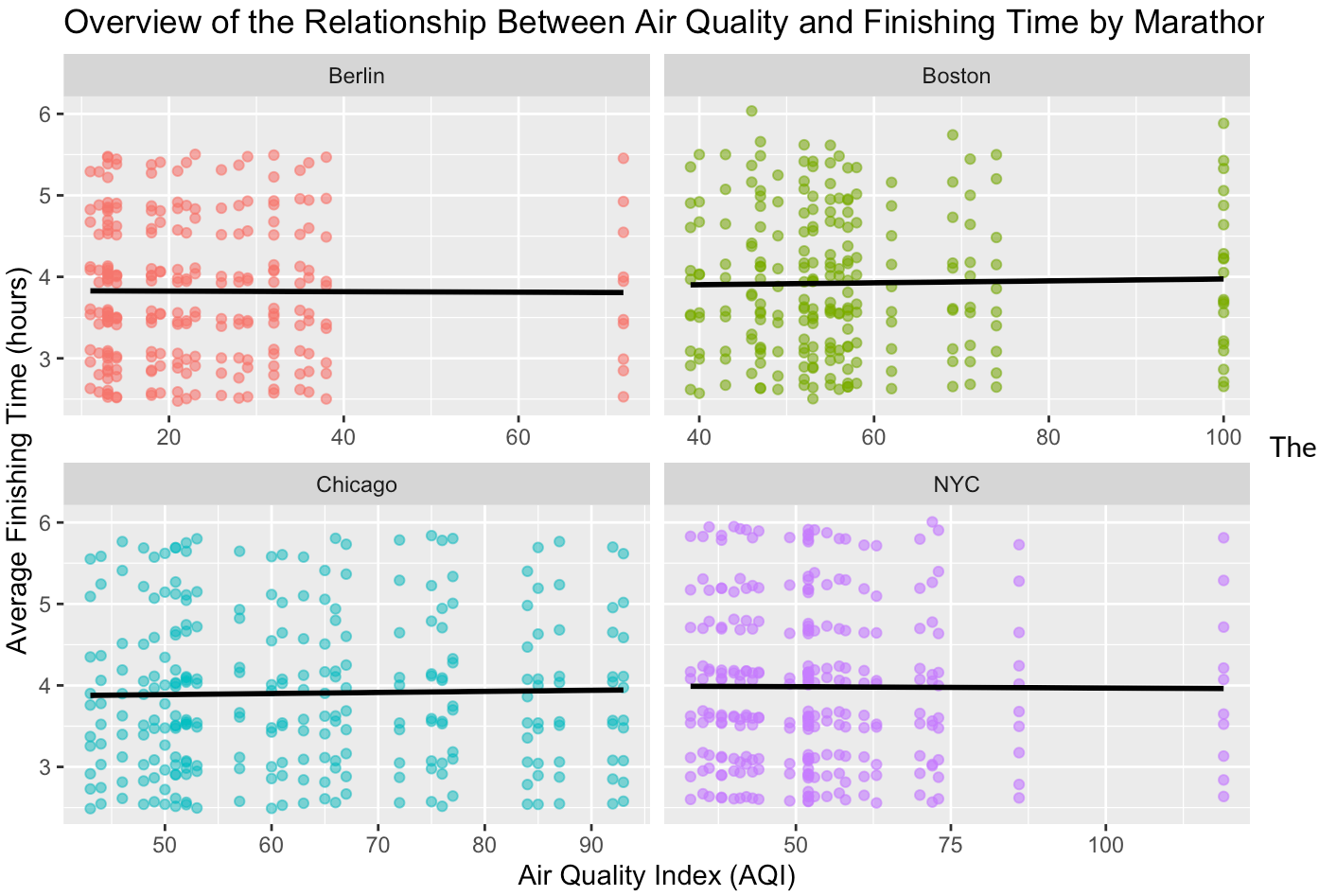
The scatterplots above show the effect of average temperature on the average finishing time by each individual marathon. Each marathon has its own scatterplot, with its own temperature scale on the x-axis and finishing time on the y-axis. The black lines represent fitted regression trend lines that show how the average finishing time changes alongside temperature. Runners are represented by the points on the scatterplot. All four of the trend lines have a slight upward slope, which indicates that as the temperature increases, the average

finishing times tend to become slower as runners take longer to complete the race. From this, we can gather that higher temperatures likely increase the risk of dehydration and fatigue amongst runners, causing runners to slow down.

3.6 Figure 3: Relationship Between Air Quality and Finishing Time by Marathon (ZD)

```
ggplot(final_data, aes(x = aqi, y = avg_chip_seconds / 3600, color = marathon)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "lm", se = FALSE, color = "black") +
  facet_wrap(~ marathon, scales = "free_x") +
  labs(title = "Overview of the Relationship Between Air Quality and Finishing Ti
me by Marathon",
    x = "Air Quality Index (AQI)",
    y = "Average Finishing Time (hours)"
  ) +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



The scatterplots above show the relationship between air quality and the average finishing time by marathon. For Berlin, the AQI values are low, with the majority of data points within an AQI value of 40. The AQI values are

low most likely because of the missing PM2.5 which is a driver for higher AQI. There appears a slight positive relationship between finishing time increasing and increased AQI for Boston and Chicago. NYC and Berlin appear slightly more flat suggesting limited or not relationship.

3.7 Figure 4: Correlation Matrix (MH)

```
# select only the numeric variables
numeric_vars <- final_data %>%
  select(avg_chip_seconds, avg_temp, precipitation, dew_point,
         wind_speed, visibility, sea_level_pressure,
         aqi, pm10, pm25, no2, ozone, co)

# create correlation matrix
cor_matrix <- cor(numeric_vars, use = "complete.obs")
```

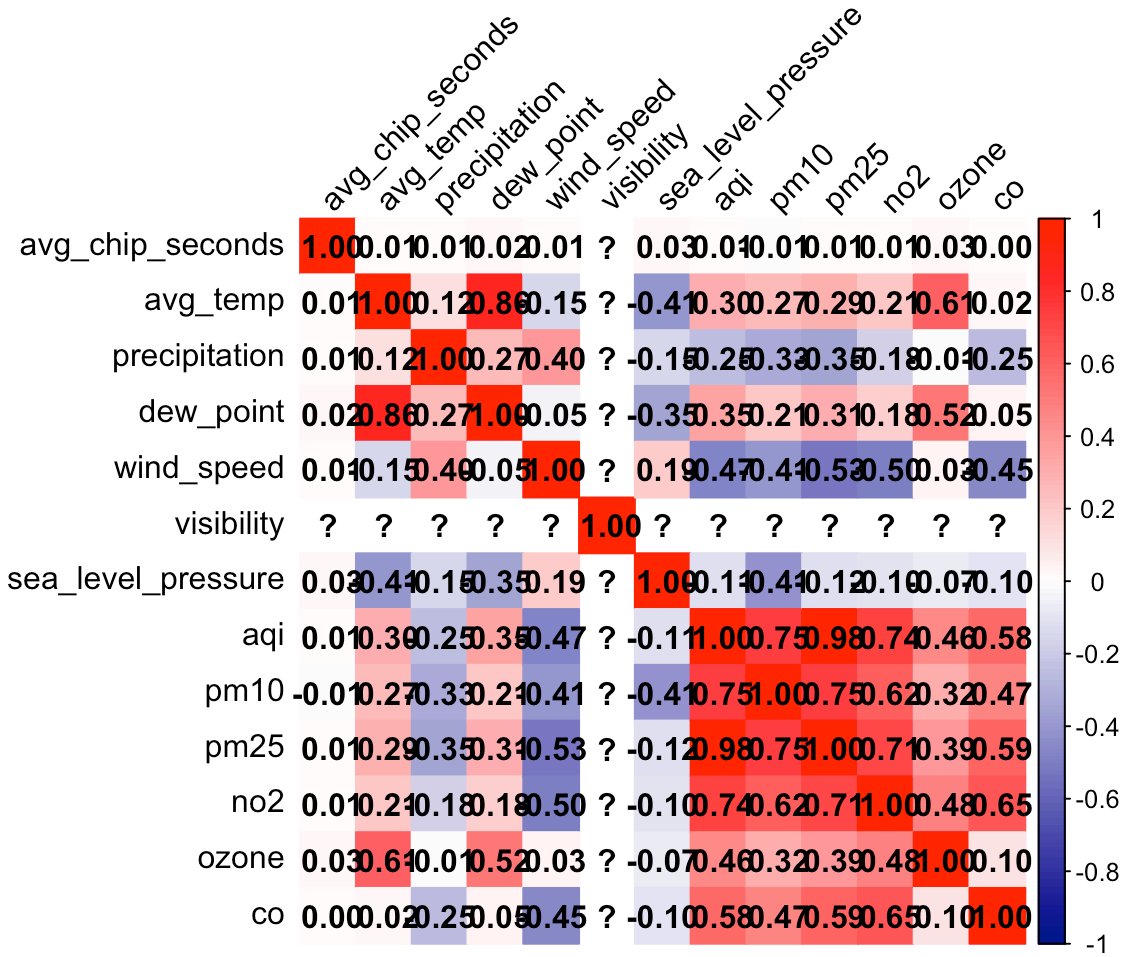
```
## Warning in cor(numeric_vars, use = "complete.obs"): the standard deviation is
## zero
```

```
# round the correlations
round(cor_matrix, 2)
```

```
##                   avg_chip_seconds avg_temp precipitation dew_point wind_spee
d
## avg_chip_seconds             1.00     0.01          0.01      0.02       0.0
1
## avg_temp                     0.01     1.00          0.12      0.86      -0.1
5
## precipitation                0.01     0.12          1.00      0.27       0.4
0
## dew_point                    0.02     0.86          0.27      1.00      -0.0
5
## wind_speed                   0.01    -0.15          0.40     -0.05       1.0
0
## visibility                     NA       NA            NA        NA         N
A
## sea_level_pressure           0.03    -0.41         -0.15     -0.35       0.1
9
## aqi                          0.01     0.30         -0.25      0.35      -0.4
7
## pm10                        -0.01     0.27         -0.33      0.21      -0.4
1
## pm25                         0.01     0.29         -0.35      0.31      -0.5
3
## no2                          0.01     0.21         -0.18      0.18      -0.5
0
## ozone                        0.03     0.61         -0.01      0.52       0.0
3
## co                           0.00     0.02         -0.25      0.05      -0.4
5
##                   visibility sea_level_pressure   aqi  pm10  pm25   no2 ozone
## avg_chip_seconds          NA               0.03  0.01 -0.01  0.01  0.01  0.03
## avg_temp                  NA              -0.41  0.30  0.27  0.29  0.21  0.61
## precipitation             NA              -0.15 -0.25 -0.33 -0.35 -0.18 -0.01
## dew_point                 NA              -0.35  0.35  0.21  0.31  0.18  0.52
## wind_speed                NA               0.19 -0.47 -0.41 -0.53 -0.50  0.03
## visibility                 1                 NA    NA    NA    NA    NA    NA
## sea_level_pressure        NA               1.00 -0.11 -0.41 -0.12 -0.10 -0.07
## aqi                       NA              -0.11  1.00  0.75  0.98  0.74  0.46
## pm10                      NA              -0.41  0.75  1.00  0.75  0.62  0.32
## pm25                      NA              -0.12  0.98  0.75  1.00  0.71  0.39
## no2                       NA              -0.10  0.74  0.62  0.71  1.00  0.48
## ozone                     NA              -0.07  0.46  0.32  0.39  0.48  1.00
## co                        NA              -0.10  0.58  0.47  0.59  0.65  0.10
##                      co
## avg_chip_seconds   0.00
## avg_temp           0.02
## precipitation     -0.25
## dew_point          0.05
## wind_speed        -0.45
```

```
## visibility             NA
## sea_level_pressure -0.10
## aqi                 0.58
## pm10                0.47
## pm25                0.59
## no2                 0.65
## ozone               0.10
## co                  1.00
```

```
#Correlation Heat map
corrplot(
  cor_matrix,
  method="color",col=colorRampPalette(c("darkblue", "white", "red"))(200),
  addCoef.col="black", tl.col="black", tl.srt=45
)
```



From the initial correlation matrix, we can see the highest correlation (>= 0.75) is between pm2.5 and AQI, dew point and average temp, and pm2.5 and pm10.

3.8 Figure 5: Scatter Plots for Pollutants (KB)

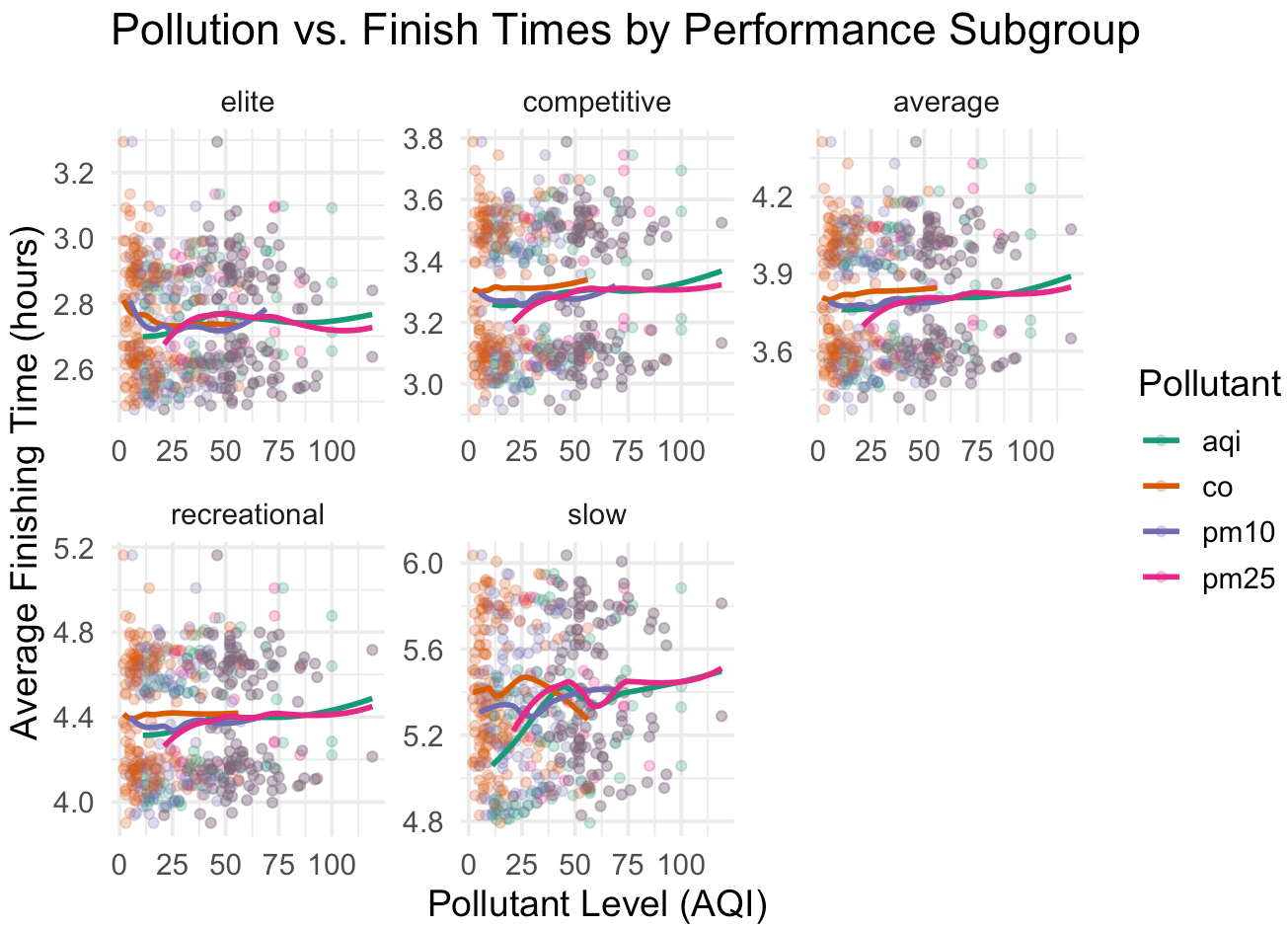
Looking at all the pollutants we have concerns with (co, pm10, pm2.5, and overall aqi).

```
# Make pollutant data into long format for graphing
pollutant_long <- final_data %>%
  mutate(
    finish_hours = avg_chip_seconds / 3600   # convert seconds to hours
  ) %>%
  pivot_longer(
    cols = c(co, pm25, pm10, aqi),
    names_to = "pollutant",
    values_to = "value"
  )

# plotting scatter plots
ggplot(pollutant_long,
       aes(x = value, y = finish_hours, color = pollutant)) +
  geom_point(alpha = 0.25) +
  geom_smooth(se = FALSE, method = "loess") +
  facet_wrap(~ subgroup, scales = "free") +
  scale_color_brewer(palette = "Dark2") +
  labs(
    title = "Pollution vs. Finish Times by Performance Subgroup",
    x = "Pollutant Level (AQI)",
    y = "Average Finishing Time (hours)",
    color = "Pollutant"
  ) +
  theme_minimal(base_size = 14)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 895 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 895 rows containing missing values or values outside the scal
e range
## (`geom_point()`).
```

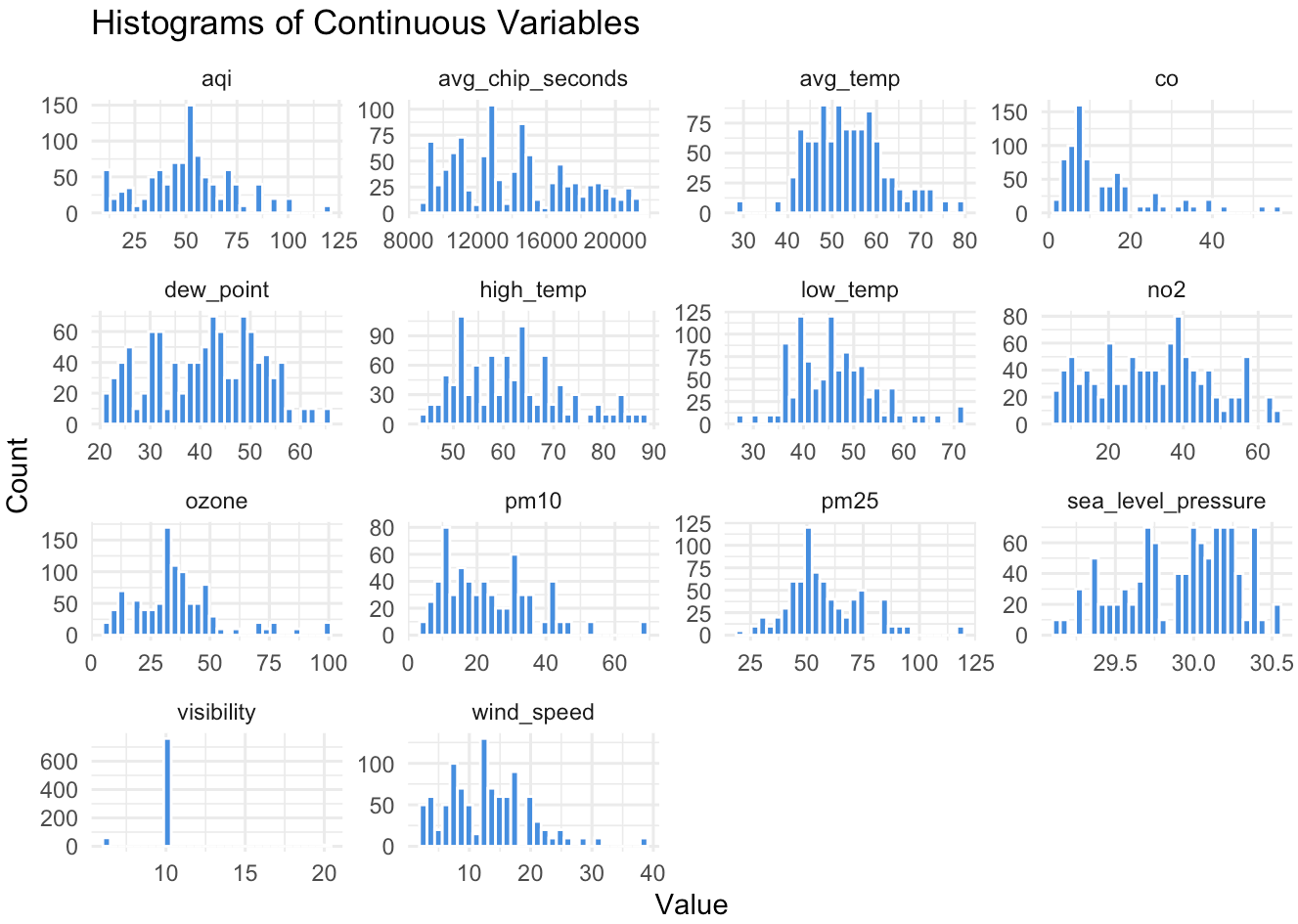


We can see that most of the subgroups follow similar patterns with airquality. As airquality increase, we see an increase in most average finishing times, with some starting to level out at a certain pollutant level among faster subgroups, such as `pm2.5` for elite and competitive subgroups. We can also see that `co` behaves oddly, specifically for slow runners, increasing as fishing time get faster with higher pollutant levels. However this is most likey due to missing `co` data for Berlin.

3.9 Figure 6: Histograms of Continous Variables (KB)

```
continuous_vars <- final_data %>%
  select(avg_chip_seconds, high_temp, low_temp, avg_temp,
         dew_point, wind_speed, visibility, sea_level_pressure,
         aqi, co, ozone, pm10, pm25, no2) %>%
  pivot_longer(everything(), names_to = "variable", values_to = "value")

ggplot(continuous_vars, aes(x = value)) +
  geom_histogram(bins = 30, fill = "#4A90E2", color = "white") +
  facet_wrap(~ variable, scales = "free") +
  theme_minimal() +
  labs(title = "Histograms of Continuous Variables",
       x = "Value",
       y = "Count")
```

```
## Warning: Removed 1070 rows containing non-finite outside the scale range
## (`stat_bin()`).
```



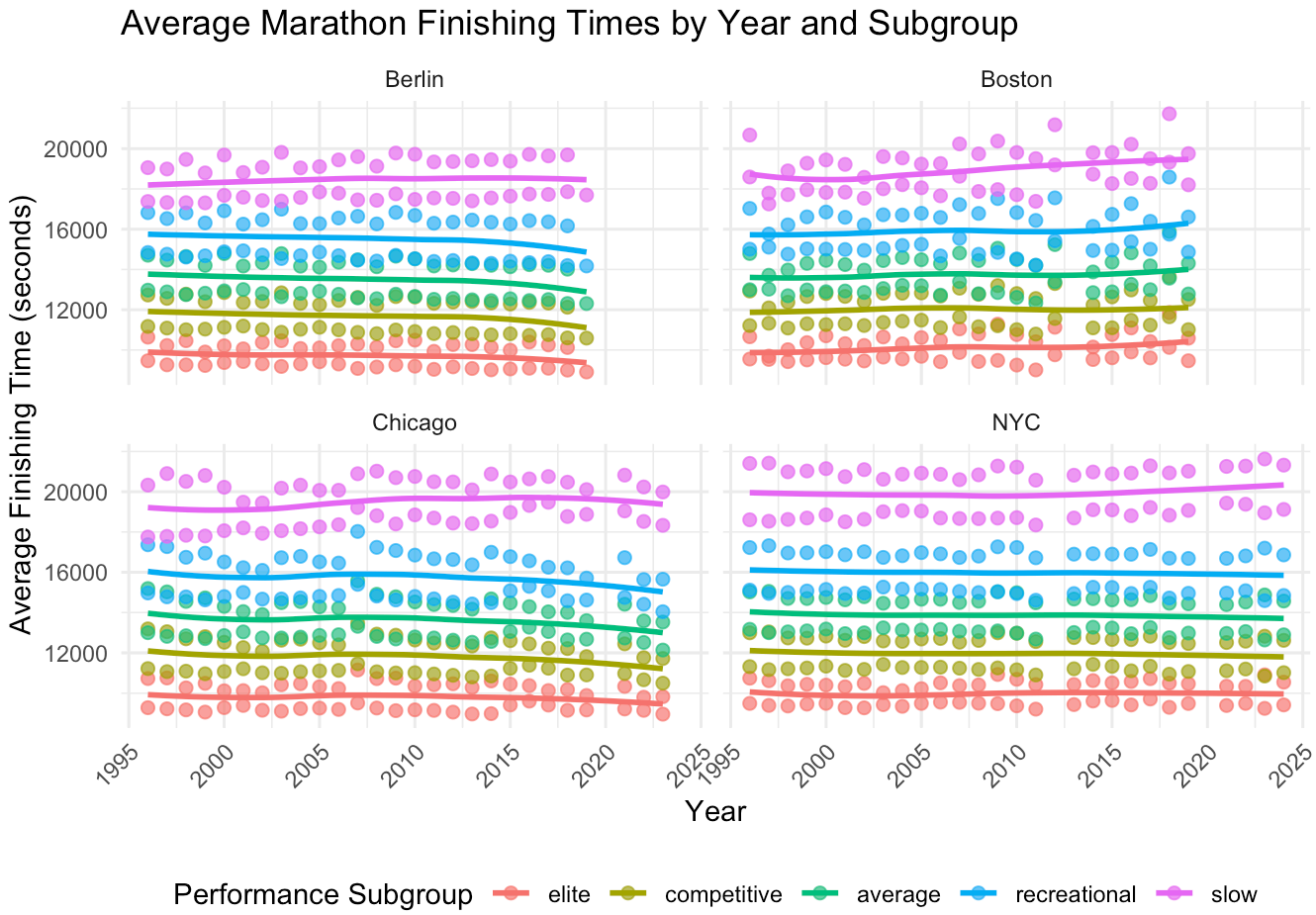
Looking at histograms for all the continuous variables, we can see the distribution shapes. Several of the environmental variables did display expected right skews (such as PM2.5 and wind speed). There is no transformation currently planned for our modeling goals. We can also see that finishing times were right skewed, which is typical for marathon data.

3.10 Figure 7: Scatter Plots by Preformance Subgroups (KB)

We want to see how subgroups behave over the years.

```
# Scatter plots by marathon and subgroup
ggplot(final_data, aes(x = year, y = avg_chip_seconds)) +
  geom_point(aes(color = subgroup), alpha = 0.7, size = 2) +  # points colored by subgroup
  geom_smooth(aes(color = subgroup), method = "loess", se = FALSE) + # optional trend lines
  facet_wrap(~ marathon) +  # one plot per marathon
  labs(
    title = "Average Marathon Finishing Times by Year and Subgroup",
    x = "Year",
    y = "Average Finishing Time (seconds)",
    color = "Performance Subgroup"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    legend.position = "bottom"
  )
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



We cans see that over time, most finishing times become faster. We can also see potential shifts in faster finishing times past 2018, where there was the introduction of supershoes. We can really see this in the Chicago marathon, as it has data till 2023. We can potentially see the trend starting with Berlin, however it is hard to tell since the data is only available till 2019. NYC seems to have a pretty constant finishing times over the years. Boston on the other hand, seems to have increasing finishing times, however it is also hard to interpret the affect of supershoes due to data only being available till 2019, only one year after supershoes have been widely introduced.

3.11 Figure 8. Association Between Visibility and Finishing Time by Performance Subgroup (MH)

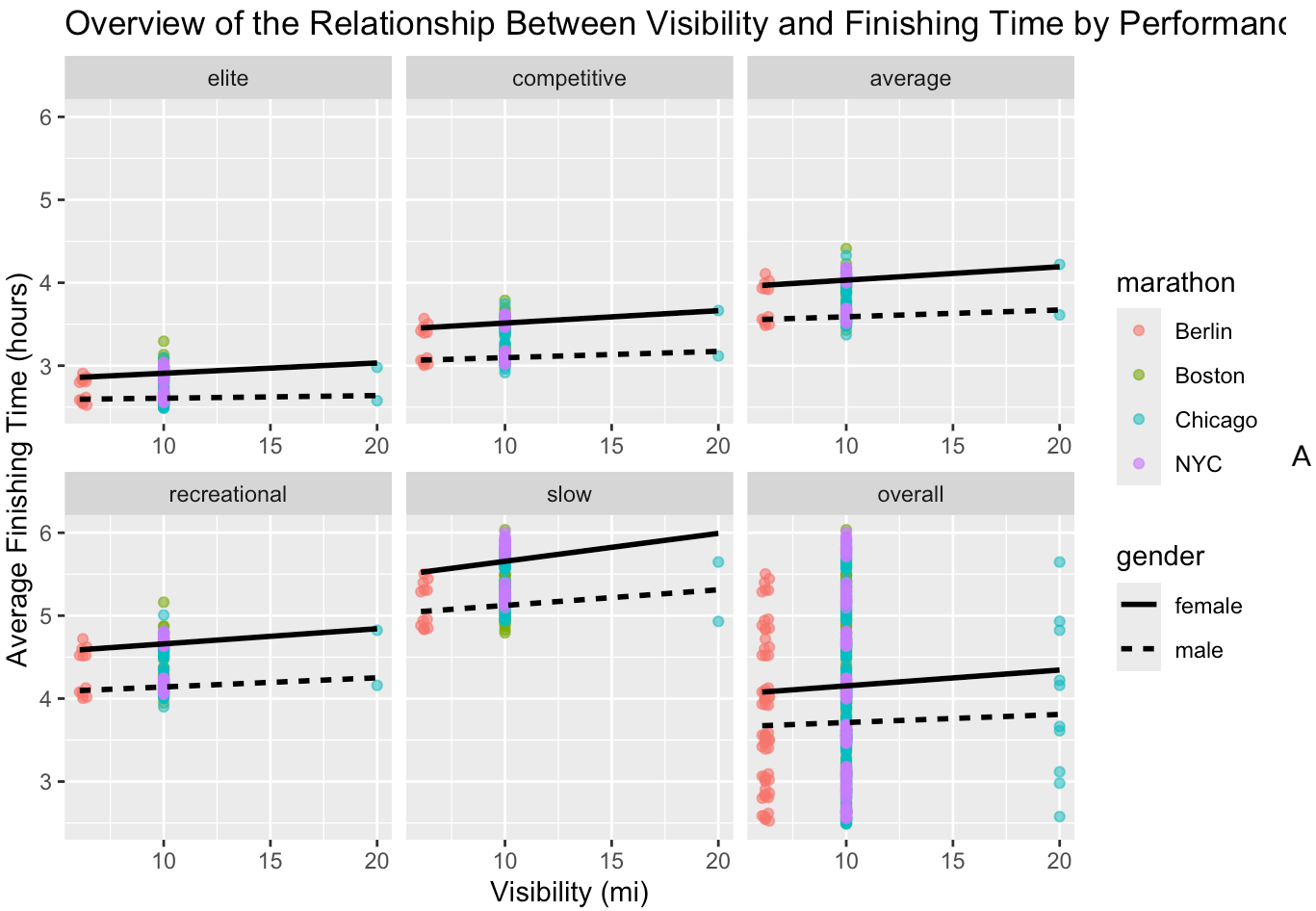
There are gender-specific linear trends seen.

```
#Add 'Overall' graph, making df 'final_data2' for purpose of figure 8 and  9
final_data2 <- final_data %>% mutate(subgroup = tolower(as.character(subgroup)))
%>% bind_rows(final_data %>%
mutate(subgroup = "overall")) %>%
  mutate(subgroup = factor(subgroup,levels = c("elite", "competitive", "average",
"recreational", "slow", "overall")))
final_data2$year <- as.numeric(as.character(final_data2$year))

#Visibility and Finishing time
ggplot(final_data2, aes(x = visibility, y = avg_chip_seconds / 3600, color = mara
thon)) +
  geom_point(alpha = 0.6) +
  geom_smooth(aes(linetype=gender),method = "lm", se = FALSE, color = "black") +
  facet_wrap(~ subgroup, scales = "free_x") +
  labs(title = "Overview of the Relationship Between Visibility and Finishing Tim
e by Performance Subgroup",
       x = "Visibility (mi)",
       y = "Average Finishing Time (hours)"
  ) +
  theme(legend.position = "right")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

```
## Warning: Removed 350 rows containing non-finite outside the scale range
## (`stat_smooth()`).
```

```
## Warning: Removed 350 rows containing missing values or values outside the scal
e range
## (`geom_point()`).
```



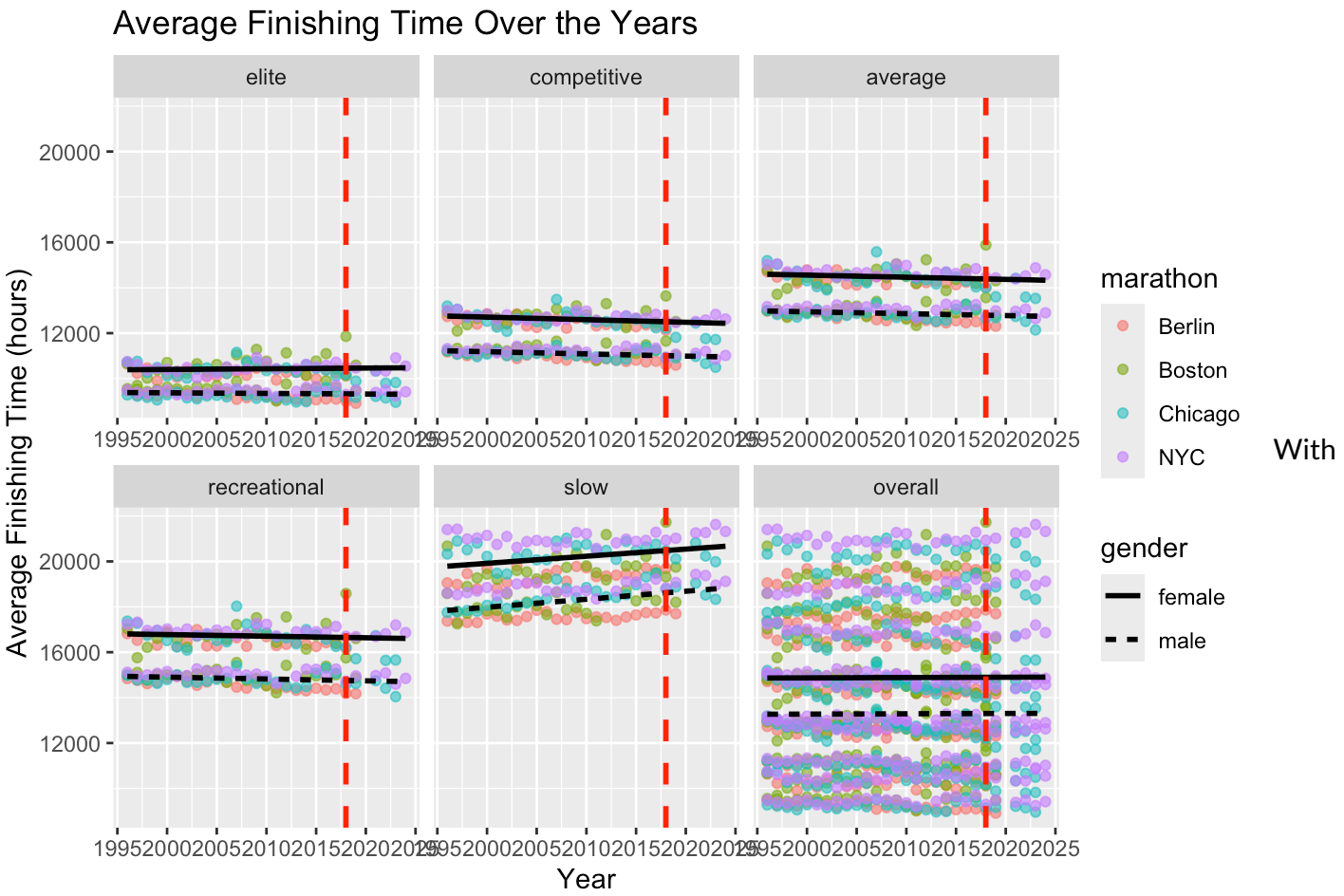
decrease in visibility appears to slow down runners across all subgroups.

3.12 Figure 9: Average Finishing Time Over Time by Subgroup and Gender (MH)

This highlights the post-2018 supershoe period.

```
# Average Finishing Time by Year, Subgroup, Gender (Supershoes)
ggplot(final_data2, aes(x = year, y = avg_chip_seconds, color = marathon)) +
  geom_point(alpha = 0.6) +
  geom_smooth(aes(linetype=gender),method = "lm", se = FALSE, color = "black") +
  facet_wrap(~ subgroup, scales = "free_x") +
  geom_vline(xintercept = 2018, linetype = "dashed", color = "red", linewidth =
1) +  # vertical line
  labs(title = "Average Finishing Time Over the Years",
       x = "Year",
       y = "Average Finishing Time (hours)"
  ) +
  theme(legend.position = "right")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



With the introduction of supershoes around 2018 (red vertical dashed line), we expect to see in improvement in finishing time. This is true for all subgroups except for slow runners. The slow runners make up the largest number of runners in each marathon. The increase could be due to the larger number of runners in the subgroup, it could also be due to an increase in charity bibs at races - where runners do not have to explicitly qualify with a qualifying time in order to run.

4 Preprocessing and Feature Engineering (KB, ZD, MH)

Look at the full dataset for missing values (ZD, KB)

```
# Summarize missing values
missing_summary <- final_data %>%
  summarise(across(everything(), ~sum(is.na(.))))

# Keep only columns with any missing values
missing_summary <- missing_summary %>%
  select(where(~any(. > 0)))

# Display nicely in HTML
knitr::kable(missing_summary, caption = "Missing Values per Variable")
```

Missing Values per Variable

visibility	co	pm10	pm25
175	235	380	280

We can see that `visability`, `co`, `pm10`, and `pm25` all have alot of missing values. This is due to the Berlin dataset.

4.1 Berlin as a Second Case Study (KB)

Spit the data so Berlin is used as a second case study to show how the method performs with missing data (whether successful or not).

```
main_data <- final_data %>% filter(marathon != "Berlin")
berlin_data <- final_data %>% filter(marathon == "Berlin")

str(main_data)
```

```
## tibble [770 × 21] (S3: tbl_df/tbl/data.frame)
##  $ n                : int [1:770] 159 1057 3481 2393 1389 767 5095 8091 6697
6677 ...
##  $ marathon         : chr [1:770] "Boston" "Boston" "Boston" "Boston" ...
##  $ year             : int [1:770] 1996 1996 1996 1996 1996 1996 1996 1996 199
6 1996 ...
##  $ gender           : chr [1:770] "female" "female" "female" "female" ...
##  $ subgroup         : Factor w/ 5 levels "elite","competitive",..: 1 2 3 4 5
1 2 3 4 5 ...
##  $ avg_chip_seconds  : num [1:770] 10661 12931 14792 17030 20673 ...
##  $ high_temp        : int [1:770] 45 45 45 45 45 45 45 45 45 45 ...
##  $ low_temp         : int [1:770] 36 36 36 36 36 36 36 36 36 36 ...
##  $ avg_temp         : num [1:770] 40.5 40.5 40.5 40.5 40.5 ...
##  $ precipitation    : num [1:770] 0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point        : num [1:770] 24.2 24.2 24.2 24.2 24.2 ...
##  $ wind_speed       : int [1:770] 20 20 20 20 20 20 20 20 20 20 ...
##  $ visibility       : num [1:770] 10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure: num [1:770] 30.2 30.2 30.2 30.2 30.2 ...
##  $ aqi              : int [1:770] 69 69 69 69 69 69 69 69 69 69 ...
##  $ main_pollutant   : chr [1:770] "PM10" "PM10" "PM10" "PM10" ...
##  $ co               : num [1:770] 13 13 13 13 13 13 13 13 13 13 ...
##  $ ozone            : num [1:770] 41 41 41 41 41 41 41 41 41 41 ...
##  $ pm10             : num [1:770] 69 69 69 69 69 69 69 69 69 69 ...
##  $ pm25             : num [1:770] NA NA NA NA NA NA NA NA NA NA ...
##  $ no2              : num [1:770] 34 34 34 34 34 34 34 34 34 34 ...
```

We can see the main_data with out berlin now has 770 obs. and 21 variables.

4.2 Handling Missing Values (ZD, KB)

```
# Summarize missing values
missing_summary <- main_data %>%
  summarise(across(everything(), ~sum(is.na(.))))

# Keep only columns with at least one missing value
missing_summary <- missing_summary %>%
  select(where(~any(. > 0)))

# Display nicely in HTML
knitr::kable(missing_summary, caption = "Missing Values per Variable")
```

Missing Values per Variable

pm10	pm25
320	70

In the main_data, we can see that there are still 320 missing values for pm10 and 70 missing values for pm2.5.

4.2.1 Remove PM10 (KB)

Since PM10 has a lot of missing values still, we will drop that column completely.

```
main_data <- main_data %>%
  select(-pm10)
```

4.2.2 KNN Imputation on PM2.5 (KB)

Since PM2.5 is often the main pollutant, we decided to use KNN imputation to fill missing PM2.5 values because it predicts missing data using similar rows without assuming a specific parametric relationship, preserves variance, and works well for our relatively small dataset while using correlations with other environmental variables.

```
# Impute missing PM2.5 values using 5 nearest neighbors
main_data <- kNN(main_data, variable = "pm25", k = 5) # can change later to see w
hich K gives best model performance
```

```
##                   n                year   avg_chip_seconds           high_temp
##               37.00             1996.00            8957.67               44.00
##            low_temp            avg_temp      precipitation           dew_point
##               28.00               29.58               0.00               21.13
##          wind_speed          visibility sea_level_pressure                 aqi
##                6.00               10.00               29.13               33.00
##                  co               ozone                 no2                   n
##                2.00               21.00                8.00            14462.00
##                year   avg_chip_seconds          high_temp           low_temp
##             2024.00            21729.70               88.00               72.00
##            avg_temp      precipitation          dew_point          wind_speed
##               79.35                0.61               65.22               39.00
##          visibility sea_level_pressure                 aqi                  co
##               20.00               30.52              119.00               56.00
##               ozone                 no2
##              100.00               66.00
```

```
# remove pm25_imp
cols_to_remove <- c(
  "pm25_imp"
)

main_data <- main_data[, !(names(main_data) %in% cols_to_remove)]

# Check that missing values are filled
summary(main_data$pm25)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   26.00   47.00   53.00   56.83   65.00  119.00
```

We can see that there are no missing values now and we can get a clean summary of the data.

Convert categorical variables to factors: (KB)

```
main_data  <- main_data  %>%
  mutate(subgroup = factor(subgroup),
         gender = factor(gender),
         marathon = factor(marathon),
         main_pollutant = factor(main_pollutant))

str(main_data)
```

```
## 'data.frame':    770 obs. of  20 variables:
##  $ n                 : int  159 1057 3481 2393 1389 767 5095 8091 6697 6677
...
##  $ marathon          : Factor w/ 3 levels "Boston","Chicago",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ year              : int  1996 1996 1996 1996 1996 1996 1996 1996 1996 1996
...
##  $ gender            : Factor w/ 2 levels "female","male": 1 1 1 1 1 2 2 2 2 2
...
##  $ subgroup          : Factor w/ 5 levels "elite","competitive",..: 1 2 3 4 5
1 2 3 4 5 ...
##  $ avg_chip_seconds  : num  10661 12931 14792 17030 20673 ...
##  $ high_temp         : int  45 45 45 45 45 45 45 45 45 45 ...
##  $ low_temp          : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ avg_temp          : num  40.5 40.5 40.5 40.5 40.5 ...
##  $ precipitation     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point         : num  24.2 24.2 24.2 24.2 24.2 ...
##  $ wind_speed        : int  20 20 20 20 20 20 20 20 20 20 ...
##  $ visibility        : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure: num  30.2 30.2 30.2 30.2 30.2 ...
##  $ aqi               : int  69 69 69 69 69 69 69 69 69 69 ...
##  $ main_pollutant    : Factor w/ 4 levels "NO2","Ozone",..: 3 3 3 3 3 3 3 3 3
3 ...
##  $ co                : num  13 13 13 13 13 13 13 13 13 13 ...
##  $ ozone             : num  41 41 41 41 41 41 41 41 41 41 ...
##  $ pm25              : num  56 55 56 55 56 56 55 56 55 55 ...
##  $ no2               : num  34 34 34 34 34 34 34 34 34 34 ...
```

We can see that the identifiers (subgroup, gender, marathon) and predictor (main_pollutant) were all converted to factors so that our models can properly recognized them.

4.3 Feature Engineering (ZD, KB)

We next created several interaction terms to try and capture potential combined effects of environmental and demographic factors on marathon performance. In addition, we added a control for the introduction of “supershoes,” coded as 1 for marathons in 2018 and later, when these performance-enhancing shoes became widely used, and 0 otherwise.

```
# create interaction terms and convert supershoe to factor
main_data_fe <- main_data %>%
  mutate(
    supershoe = factor(ifelse(year >= 2018, 1, 0), levels = c(0, 1)),
    temp_dew_interaction       = avg_temp * dew_point,
    temp_aqi_interaction       = avg_temp * aqi,
    temp_precip_interaction    = avg_temp * precipitation,
    temp_wind_interaction      = avg_temp * wind_speed,
    pm25_temp_interaction      = pm25 * avg_temp,
    dew_wind_interaction       = dew_point * wind_speed,
    pressure_temp_interaction  = sea_level_pressure * avg_temp,
    avg_temp_gender_interaction = avg_temp * as.numeric(gender == "male")
  )

str(main_data_fe)
```

```
## 'data.frame':    770 obs. of  29 variables:
##  $ n                           : int  159 1057 3481 2393 1389 767 5095 8091 669
7 6677 ...
##  $ marathon                    : Factor w/ 3 levels "Boston","Chicago",..: 1 1
1 1 1 1 1 1 1 1 ...
##  $ year                        : int  1996 1996 1996 1996 1996 1996 1996 1996 1
996 1996 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 1
2 2 2 2 2 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
2 3 4 5 1 2 3 4 5 ...
##  $ avg_chip_seconds            : num  10661 12931 14792 17030 20673 ...
##  $ high_temp                   : int  45 45 45 45 45 45 45 45 45 45 ...
##  $ low_temp                    : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ avg_temp                    : num  40.5 40.5 40.5 40.5 40.5 ...
##  $ precipitation               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point                   : num  24.2 24.2 24.2 24.2 24.2 ...
##  $ wind_speed                  : int  20 20 20 20 20 20 20 20 20 20 ...
##  $ visibility                  : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure          : num  30.2 30.2 30.2 30.2 30.2 ...
##  $ aqi                         : int  69 69 69 69 69 69 69 69 69 69 ...
##  $ main_pollutant              : Factor w/ 4 levels "NO2","Ozone",..: 3 3 3 3 3
3 3 3 3 3 ...
##  $ co                          : num  13 13 13 13 13 13 13 13 13 13 ...
##  $ ozone                       : num  41 41 41 41 41 41 41 41 41 41 ...
##  $ pm25                        : num  56 55 56 55 56 56 55 56 55 55 ...
##  $ no2                         : num  34 34 34 34 34 34 34 34 34 34 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ temp_dew_interaction        : num  978 978 978 978 978 ...
##  $ temp_aqi_interaction        : num  2792 2792 2792 2792 2792 ...
##  $ temp_precip_interaction     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ temp_wind_interaction       : num  809 809 809 809 809 ...
##  $ pm25_temp_interaction       : num  2266 2225 2266 2225 2266 ...
##  $ dew_wind_interaction        : num  483 483 483 483 483 ...
##  $ pressure_temp_interaction   : num  1222 1222 1222 1222 1222 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 0 ...
```

4.4 Correlation Checks (KB)

Need to do a Quick correlation check for numeric variables and see what features introduce multicollinearity (includinf the new engineered features).

```
# Find numeric columns after feature engineering, excluding 'year' and 'n' since
they are identifiers
numeric_vars <- main_data_fe %>%
  select(where(is.numeric)) %>%
  select(-year, -n) %>%
  names()

numeric_vars
```

```
##  [1] "avg_chip_seconds"           "high_temp"
##  [3] "low_temp"                   "avg_temp"
##  [5] "precipitation"              "dew_point"
##  [7] "wind_speed"                 "visibility"
##  [9] "sea_level_pressure"         "aqi"
## [11] "co"                         "ozone"
## [13] "pm25"                       "no2"
## [15] "temp_dew_interaction"       "temp_aqi_interaction"
## [17] "temp_precip_interaction"    "temp_wind_interaction"
## [19] "pm25_temp_interaction"      "dew_wind_interaction"
## [21] "pressure_temp_interaction"  "avg_temp_gender_interaction"
```

We can see that there are a total 14 continuous variables, disincluding the identifiers `year` and `n`.

Creat Correlation Matrix and make a visual: (KB)

```
cor_matrix <- cor(main_data_fe[numeric_vars], use = "pairwise.complete.obs")

# rounding
round(cor_matrix, 2)
```

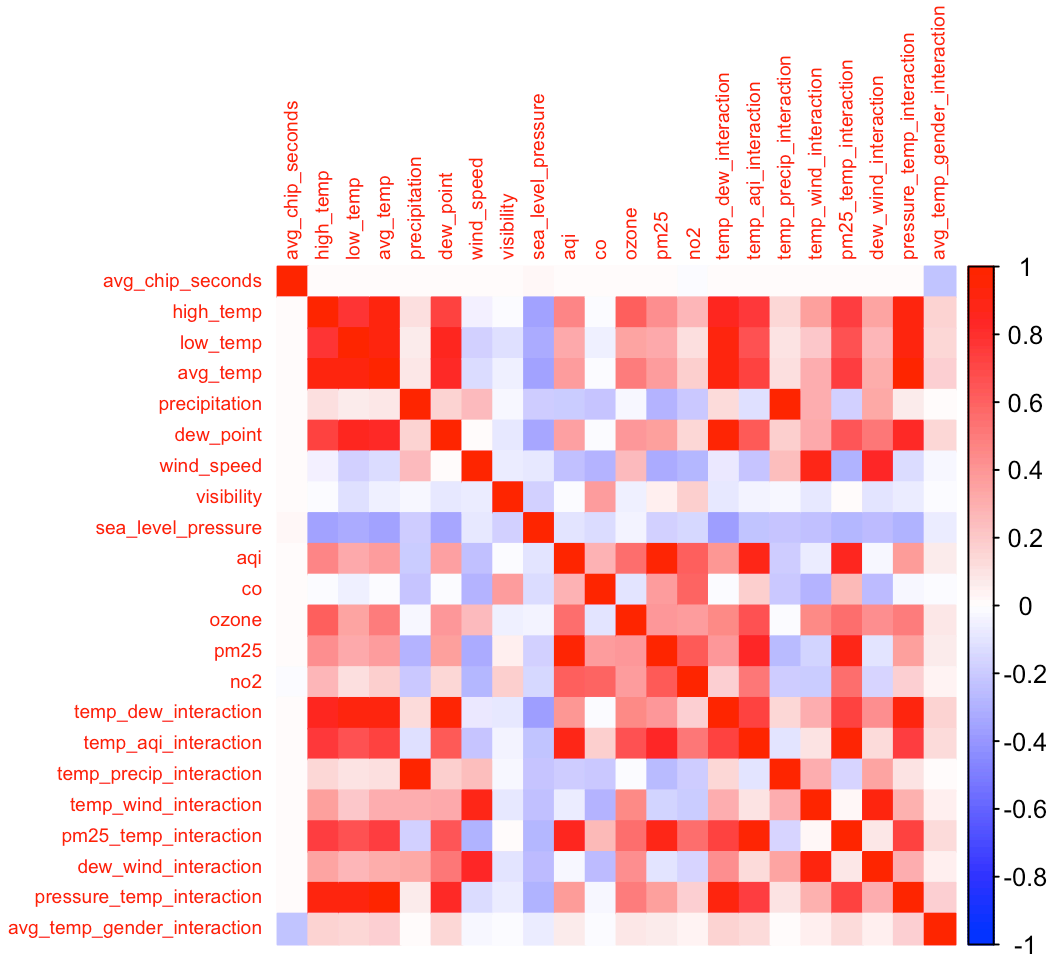
```
##                              avg_chip_seconds high_temp low_temp avg_temp
## avg_chip_seconds                         1.00      0.01     0.01     0.01
## high_temp                                0.01      1.00     0.78     0.93
## low_temp                                 0.01      0.78     1.00     0.92
## avg_temp                                 0.01      0.93     0.92     1.00
## precipitation                            0.00      0.12     0.07     0.08
## dew_point                                0.01      0.74     0.86     0.83
## wind_speed                               0.00     -0.05    -0.17    -0.13
## visibility                               0.00     -0.01    -0.11    -0.06
## sea_level_pressure                       0.02     -0.35    -0.32    -0.35
## aqi                                      0.00      0.47     0.32     0.38
## co                                       0.00     -0.02    -0.06    -0.02
## ozone                                    0.01      0.62     0.35     0.50
## pm25                                     0.00      0.43     0.33     0.37
## no2                                     -0.01      0.27     0.12     0.18
## temp_dew_interaction                     0.01      0.85     0.93     0.94
## temp_aqi_interaction                     0.01      0.77     0.66     0.74
## temp_precip_interaction                  0.00      0.15     0.09     0.11
## temp_wind_interaction                    0.01      0.37     0.21     0.30
## pm25_temp_interaction                    0.01      0.75     0.68     0.74
## dew_wind_interaction                     0.01      0.34     0.28     0.31
## pressure_temp_interaction                0.01      0.93     0.92     1.00
## avg_temp_gender_interaction             -0.22      0.16     0.16     0.17
##                              precipitation dew_point wind_speed visibility
## avg_chip_seconds                      0.00      0.01       0.00       0.00
## high_temp                             0.12      0.74      -0.05      -0.01
## low_temp                              0.07      0.86      -0.17      -0.11
## avg_temp                              0.08      0.83      -0.13      -0.06
## precipitation                         1.00      0.16       0.25      -0.03
## dew_point                             0.16      1.00       0.01      -0.09
## wind_speed                            0.25      0.01       1.00      -0.07
## visibility                           -0.03     -0.09      -0.07       1.00
## sea_level_pressure                   -0.19     -0.33      -0.08      -0.17
## aqi                                  -0.20      0.36      -0.24      -0.01
## co                                   -0.21     -0.01      -0.28       0.37
## ozone                                -0.02      0.39       0.25      -0.06
## pm25                                 -0.29      0.36      -0.33       0.06
## no2                                  -0.21      0.15      -0.28       0.18
## temp_dew_interaction                  0.13      0.96      -0.07      -0.08
## temp_aqi_interaction                 -0.11      0.64      -0.22      -0.04
## temp_precip_interaction               0.99      0.18       0.24      -0.03
## temp_wind_interaction                 0.30      0.33       0.89      -0.08
## pm25_temp_interaction                -0.18      0.65      -0.29       0.01
## dew_wind_interaction                  0.34      0.52       0.84      -0.09
## pressure_temp_interaction             0.07      0.83      -0.14      -0.07
## avg_temp_gender_interaction           0.01      0.14      -0.02      -0.01
##                              sea_level_pressure   aqi    co ozone  pm25   no2
## avg_chip_seconds                           0.02  0.00  0.00  0.01  0.00 -0.01
```

```
## high_temp                          -0.35  0.47 -0.02  0.62  0.43  0.27
## low_temp                           -0.32  0.32 -0.06  0.35  0.33  0.12
## avg_temp                           -0.35  0.38 -0.02  0.50  0.37  0.18
## precipitation                      -0.19 -0.20 -0.21 -0.02 -0.29 -0.21
## dew_point                          -0.33  0.36 -0.01  0.39  0.36  0.15
## wind_speed                         -0.08 -0.24 -0.28  0.25 -0.33 -0.28
## visibility                         -0.17 -0.01  0.37 -0.06  0.06  0.18
## sea_level_pressure                  1.00 -0.10 -0.14 -0.04 -0.18 -0.14
## aqi                                -0.10  1.00  0.28  0.57  0.94  0.62
## co                                 -0.14  0.28  1.00 -0.10  0.37  0.59
## ozone                              -0.04  0.57 -0.10  1.00  0.40  0.38
## pm25                               -0.18  0.94  0.37  0.40  1.00  0.63
## no2                                -0.14  0.62  0.59  0.38  0.63  1.00
## temp_dew_interaction               -0.36  0.40 -0.01  0.46  0.40  0.18
## temp_aqi_interaction               -0.22  0.89  0.19  0.68  0.84  0.53
## temp_precip_interaction            -0.21 -0.18 -0.21 -0.02 -0.27 -0.18
## temp_wind_interaction              -0.23 -0.07 -0.28  0.46 -0.16 -0.19
## pm25_temp_interaction              -0.28  0.86  0.26  0.56  0.89  0.55
## dew_wind_interaction               -0.26 -0.02 -0.25  0.42 -0.09 -0.15
## pressure_temp_interaction          -0.29  0.38 -0.03  0.51  0.37  0.18
## avg_temp_gender_interaction        -0.06  0.06  0.00  0.09  0.06  0.03
##                            temp_dew_interaction temp_aqi_interaction
## avg_chip_seconds                           0.01                 0.01
## high_temp                                  0.85                 0.77
## low_temp                                   0.93                 0.66
## avg_temp                                   0.94                 0.74
## precipitation                              0.13                -0.11
## dew_point                                  0.96                 0.64
## wind_speed                                -0.07                -0.22
## visibility                                -0.08                -0.04
## sea_level_pressure                        -0.36                -0.22
## aqi                                        0.40                 0.89
## co                                        -0.01                 0.19
## ozone                                      0.46                 0.68
## pm25                                       0.40                 0.84
## no2                                        0.18                 0.53
## temp_dew_interaction                       1.00                 0.72
## temp_aqi_interaction                       0.72                 1.00
## temp_precip_interaction                    0.16                -0.09
## temp_wind_interaction                      0.31                 0.10
## pm25_temp_interaction                      0.73                 0.97
## dew_wind_interaction                       0.43                 0.14
## pressure_temp_interaction                  0.93                 0.74
## avg_temp_gender_interaction                0.16                 0.13
##                            temp_precip_interaction temp_wind_interaction
## avg_chip_seconds                              0.00                 0.01
## high_temp                                     0.15                 0.37
## low_temp                                      0.09                 0.21
## avg_temp                                      0.11                 0.30
```

```
## precipitation                          0.99                0.30
## dew_point                              0.18                0.33
## wind_speed                             0.24                0.89
## visibility                            -0.03               -0.08
## sea_level_pressure                    -0.21               -0.23
## aqi                                   -0.18               -0.07
## co                                    -0.21               -0.28
## ozone                                 -0.02                0.46
## pm25                                  -0.27               -0.16
## no2                                   -0.18               -0.19
## temp_dew_interaction                   0.16                0.31
## temp_aqi_interaction                  -0.09                0.10
## temp_precip_interaction                1.00                0.31
## temp_wind_interaction                  0.31                1.00
## pm25_temp_interaction                 -0.16                0.03
## dew_wind_interaction                   0.34                0.93
## pressure_temp_interaction              0.10                0.29
## avg_temp_gender_interaction            0.02                0.05
##                           pm25_temp_interaction dew_wind_interaction
## avg_chip_seconds                           0.01                0.01
## high_temp                                  0.75                0.34
## low_temp                                   0.68                0.28
## avg_temp                                   0.74                0.31
## precipitation                             -0.18                0.34
## dew_point                                  0.65                0.52
## wind_speed                                -0.29                0.84
## visibility                                 0.01               -0.09
## sea_level_pressure                        -0.28               -0.26
## aqi                                        0.86               -0.02
## co                                         0.26               -0.25
## ozone                                      0.56                0.42
## pm25                                       0.89               -0.09
## no2                                        0.55               -0.15
## temp_dew_interaction                       0.73                0.43
## temp_aqi_interaction                       0.97                0.14
## temp_precip_interaction                   -0.16                0.34
## temp_wind_interaction                      0.03                0.93
## pm25_temp_interaction                      1.00                0.08
## dew_wind_interaction                       0.08                1.00
## pressure_temp_interaction                  0.74                0.30
## avg_temp_gender_interaction                0.13                0.05
##                           pressure_temp_interaction
## avg_chip_seconds                               0.01
## high_temp                                      0.93
## low_temp                                       0.92
## avg_temp                                       1.00
## precipitation                                  0.07
## dew_point                                      0.83
## wind_speed                                    -0.14
```

```
## visibility                                      -0.07
## sea_level_pressure                              -0.29
## aqi                                              0.38
## co                                              -0.03
## ozone                                            0.51
## pm25                                             0.37
## no2                                              0.18
## temp_dew_interaction                             0.93
## temp_aqi_interaction                             0.74
## temp_precip_interaction                          0.10
## temp_wind_interaction                            0.29
## pm25_temp_interaction                            0.74
## dew_wind_interaction                             0.30
## pressure_temp_interaction                        1.00
## avg_temp_gender_interaction                      0.17
##                            avg_temp_gender_interaction
## avg_chip_seconds                                -0.22
## high_temp                                        0.16
## low_temp                                         0.16
## avg_temp                                         0.17
## precipitation                                    0.01
## dew_point                                        0.14
## wind_speed                                      -0.02
## visibility                                      -0.01
## sea_level_pressure                              -0.06
## aqi                                              0.06
## co                                               0.00
## ozone                                            0.09
## pm25                                             0.06
## no2                                              0.03
## temp_dew_interaction                             0.16
## temp_aqi_interaction                             0.13
## temp_precip_interaction                          0.02
## temp_wind_interaction                            0.05
## pm25_temp_interaction                            0.13
## dew_wind_interaction                             0.05
## pressure_temp_interaction                        0.17
## avg_temp_gender_interaction                      1.00
```

```
#ploting
corrplot(
  cor_matrix,
  method = "color",
  col = colorRampPalette(c("blue", "white", "red"))(200),
  tl.cex = 0.6
)
```



Looking at the visual, we can see that a lot of these features were highly correlated and will have to be removed. For example, high_temp and low_temp are strongly correlated with avg_temp (0.93 and 0.92), and aqi overlaps with pm25 (0.94). Interaction terms like temp_dew_interaction, temp_precip_interaction, and pressure_temp_interaction show very high correlations with their constituent variables (up to 1.00), making them redundant. Similarly, main_pollutant is just a categorical version of aqi. Overall, these features provide little additional information and can be removed to reduce multicollinearity.

4.5 Examination of Categorical Variables (KB)

Looking at their distributions to see if we should keep or remove them.

```
table(main_data_fe$main_pollutant)
```

```
##
##   NO2 Ozone  PM10 PM2.5
##    80   110    10   570
```

```
table(main_data_fe$supershoe)
```

```
##
##   0   1
## 640 130
```

We can see that main_pollutant still contains data from pm10 which was removed. Also we know that main_pollutant is the categorical variable for aqi, and since aqi is highly correlated with pm2.5 (often the main_pollutant), we will drop aqi and main_pollutant all together.

Looking at the supershoes variable, there is an expected imbalance, with more observations of years without supershoes (0 = 640) and fewer with supershoes (1 = 130) since they were introduced more recently in 2018. We will keep this as a control variable, and if needed, techniques like weighting can be applied to help the imbalance.

Removing correlated variables as seen above: (KB)

```
cols_to_remove <- c(
  "high_temp",
  "low_temp",
  "aqi",
  "temp_dew_interaction",
  "temp_precip_interaction",
  "temp_wind_interaction",
  "pm25_temp_interaction",
  "dew_wind_interaction",
  "pressure_temp_interaction",
  "main_pollutant"
)

main_data_fe <- main_data_fe[, !(names(main_data_fe) %in% cols_to_remove)]
str(main_data_fe)
```

```
## 'data.frame':    770 obs. of  19 variables:
##  $ n                           : int  159 1057 3481 2393 1389 767 5095 8091 669
7 6677 ...
##  $ marathon                    : Factor w/ 3 levels "Boston","Chicago",..: 1 1
1 1 1 1 1 1 1 1 ...
##  $ year                        : int  1996 1996 1996 1996 1996 1996 1996 1996 1
996 1996 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 1
2 2 2 2 2 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
2 3 4 5 1 2 3 4 5 ...
##  $ avg_chip_seconds            : num  10661 12931 14792 17030 20673 ...
##  $ avg_temp                    : num  40.5 40.5 40.5 40.5 40.5 ...
##  $ precipitation               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point                   : num  24.2 24.2 24.2 24.2 24.2 ...
##  $ wind_speed                  : int  20 20 20 20 20 20 20 20 20 20 ...
##  $ visibility                  : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure          : num  30.2 30.2 30.2 30.2 30.2 ...
##  $ co                          : num  13 13 13 13 13 13 13 13 13 13 ...
##  $ ozone                       : num  41 41 41 41 41 41 41 41 41 41 ...
##  $ pm25                        : num  56 55 56 55 56 56 55 56 55 55 ...
##  $ no2                         : num  34 34 34 34 34 34 34 34 34 34 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ temp_aqi_interaction        : num  2792 2792 2792 2792 2792 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 0 ...
```

Now we have a dataset that has 770 obs. and only 19 variables after the addition of newly engineered features and removal if highly correlated features.

5 Feature Selection Using Supervised Models (KB, MH, ZD)

5.1 Decision Trees (KB, MH)

The following uses the final merged dataset before any feature engineering. There is no gender interaction for the sake of feature engineering (binning) investigation.

```
#(KB from merging and pre-feat-eng)

#Compute average finishing time per mararthon/year (no gender) (KB, MH)
avg_times_tree <- marathons_all %>% # (KB) switch to marathons_all dataset instea
d of runners dataset
  group_by(marathon, year) %>%
  summarize(
    n = n(),  # number of runners in each subgroup
    avg_chip_seconds = mean(chip_seconds, na.rm = TRUE),
    .groups = "drop"
  )


# Left join weather_clean and airquailty_clean onto the cleaned marathon datasets
(KB) (MH)
final_data <- avg_times_tree %>%
  left_join(weather_clean, by = c("year", "marathon")) %>%
  left_join(airquality_clean, by = c("year", "marathon"))
# Impute missing PM2.5 values using 5 nearest neighbors
final_data_tree <- kNN(final_data, variable = "pm25", k = 5) # can change later t
o see which K gives best model performance
```

```
##              year                 n   avg_chip_seconds         high_temp
##           1996.00           8194.00           13266.03             44.00
##          low_temp          avg_temp      precipitation         dew_point
##             28.00             29.58               0.00             21.13
##        wind_speed        visibility sea_level_pressure               aqi
##              3.00              6.06             29.13             11.00
##                co             ozone              pm10               no2
##              2.00              7.00              5.00              7.00
##              year                 n   avg_chip_seconds         high_temp
##           2024.00          55515.00           17604.39             88.00
##          low_temp          avg_temp      precipitation         dew_point
##             72.00             79.35               0.85             65.22
##        wind_speed        visibility sea_level_pressure               aqi
##              39.00             20.00             30.54            119.00
##                co             ozone              pm10               no2
##             56.00            100.00             69.00             66.00
```

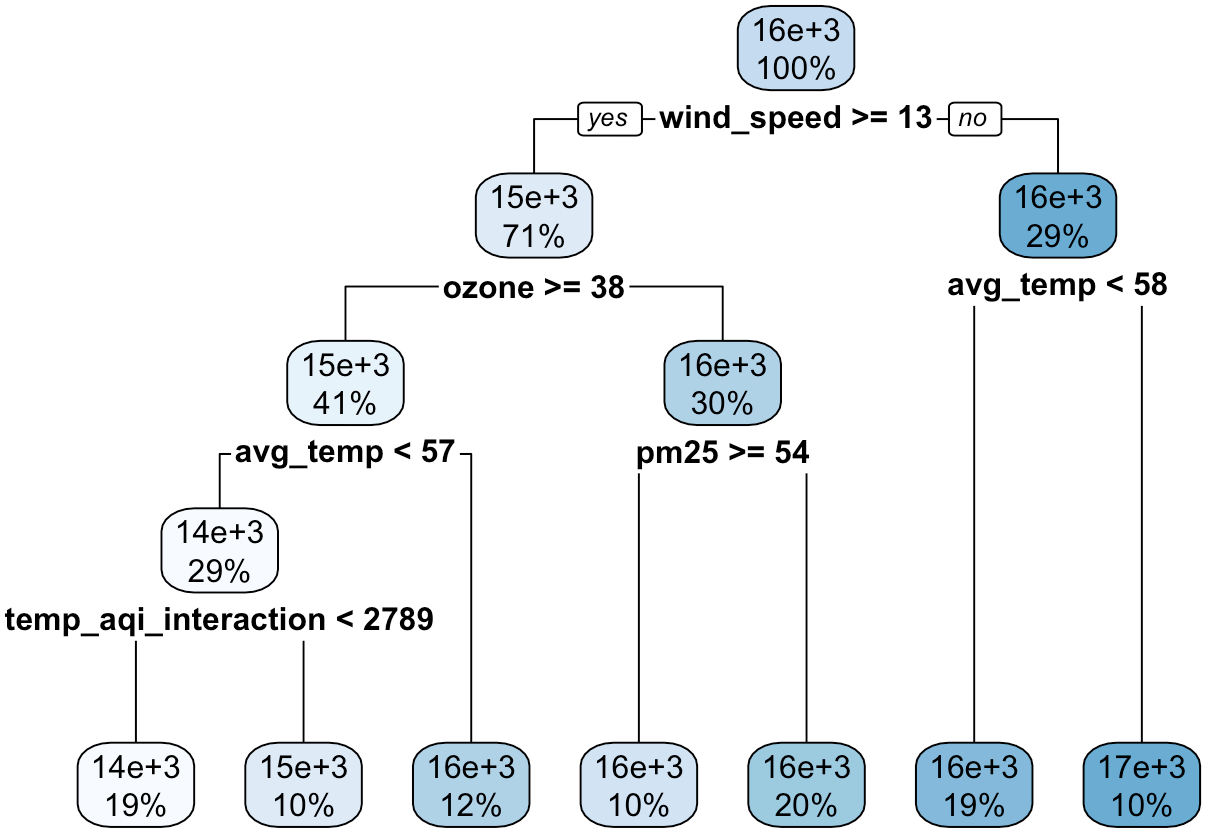
```
# remove pm25_imp
cols_to_remove <- c(
  "pm25_imp"
)

final_data_tree <- final_data_tree[, !(names(final_data_tree) %in% cols_to_remov
e)]


# create interaction terms and convert supershoe to factor (KB, MH)
final_data_tree <- final_data_tree %>%
  mutate(
    supershoe = factor(ifelse(year >= 2018, 1, 0), levels = c(0, 1)),
    temp_dew_interaction       = avg_temp * dew_point,
    temp_aqi_interaction       = avg_temp * aqi,
    temp_precip_interaction    = avg_temp * precipitation,
    temp_wind_interaction      = avg_temp * wind_speed,
    pm25_temp_interaction      = pm25 * avg_temp,
    dew_wind_interaction       = dew_point * wind_speed,
    pressure_temp_interaction  = sea_level_pressure * avg_temp
  )

cols_to_remove <- c(
  "high_temp",
  "low_temp",
  "aqi",
  "temp_dew_interaction",
  "temp_precip_interaction",
  "temp_wind_interaction",
  "pm25_temp_interaction",
  "dew_wind_interaction",
  "pressure_temp_interaction",
  "main_pollutant",
  "pm10"
)
final_data_tree <- final_data_tree[, !(names(final_data_tree) %in% cols_to_remov
e)]

# indicate and define year of covid (ZD)
final_data_tree <- final_data_tree %>%
  mutate(
    covid_era = ifelse(year == 2020, 1, 0)
  )

#removed Berlin
final_data_tree <- final_data_tree %>% filter(marathon != "Berlin")
```

Add our 90/10 training/test split for the decision tree (ZD) (KB)

```
# split data into train and test
set.seed(123)

train_index_tree <- sample(1:nrow(final_data_tree), size = 0.9 * nrow(final_data_
tree)) # use a 90/10 split

train_data_tree <- final_data_tree[train_index_tree, ]
test_data_tree  <- final_data_tree[-train_index_tree, ]
```

Decision tree looking at weather and airquality features to consider breaking down into bins. reference: https://bradleyboehmke.github.io/HOML/DT.html (https://bradleyboehmke.github.io/HOML/DT.html) (MH) (KB)

```
feature_check <- rpart(
  avg_chip_seconds ~ avg_temp + precipitation + dew_point + wind_speed + visibili
ty +
    sea_level_pressure + co + ozone + pm25 + no2 + supershoe + temp_aqi_interacti
on + covid_era,
  data = train_data_tree,
  method = "anova"
)

# Plot showing important features and potential bins
rpart.plot(feature_check)
```



We can see that the decision tree shows that wind speed is the most important factor, splitting at 13 units. For higher wind speeds, ozone and temperature further influence the outcome. For lower wind speeds, temperature is the main driver.

Categorical bins from tree (TRAIN ONLY) (MH) (KB)

```
# wind speed bin
train_data_tree$wind_bin <- ifelse(train_data_tree$wind_speed >= 13,
                                   "high_wind",
                                   "low_wind")

# ozone bin
train_data_tree$ozone_bin <- ifelse(train_data_tree$ozone >= 38,
                                    "high_ozone",
                                    "low_ozone")

# temp bins (merging 'cool' and 'moderate', since moderate is a tiny bin)
train_data_tree$temp_bin <- cut(train_data_tree$avg_temp,
                                breaks = c(-Inf, 58, Inf),  # merge 'cool' and 'modera
te'
                                labels = c("cool_moderate", "warm"),
                                right = FALSE)

# pm2.5 bin
train_data_tree$pm25_bin <- ifelse(train_data_tree$pm25 >= 54,
                                   "high_pm25",
                                   "low_pm25")

# temp × aqi interaction bin
train_data_tree$temp_aqi_bin <- ifelse(train_data_tree$temp_aqi_interaction >= 27
89,
                                       "high_interaction",
                                       "low_interaction")
```

Comparing models with continuous and binned data to determine which featires should be binned or left continuous (TRAIN ONLY) (KB, MH)

```
# continuous-only model
model_cont <- lm(avg_chip_seconds ~ avg_temp + ozone + pm25 + wind_speed + temp_a
qi_interaction,
                 data = train_data_tree)

# full binned model
model_bin <- lm(avg_chip_seconds ~ temp_bin + ozone_bin + pm25_bin + wind_bin + t
emp_aqi_bin,
                data = train_data_tree)

# temperature binned with all others continuous
model_temp <- lm(avg_chip_seconds ~ temp_bin + ozone + pm25 + wind_speed + temp_a
qi_interaction,
                 data = train_data_tree)

# temperature and ozone binned with all others continuous
model_temp_ozone <- lm(avg_chip_seconds ~ temp_bin + ozone_bin + pm25 + wind_spee
d + temp_aqi_interaction,
                       data = train_data_tree)

# air quality binned (ozone + pm25) with temperature continuous
model_airbin <- lm(avg_chip_seconds ~ avg_temp + ozone_bin + pm25_bin + wind_spee
d + temp_aqi_interaction,
                   data = train_data_tree)

# ozone binned only
model_ozone <- lm(avg_chip_seconds ~ avg_temp + ozone_bin + pm25 + wind_speed + t
emp_aqi_interaction,
                  data = train_data_tree)

# wind binned only
model_wind <- lm(avg_chip_seconds ~ avg_temp + ozone + pm25 + wind_bin + temp_aqi
_interaction,
                 data = train_data_tree)

# wind + ozone + pm25 binned
model_wind_air <- lm(avg_chip_seconds ~ avg_temp + ozone_bin + pm25_bin + wind_bi
n + temp_aqi_interaction,
                     data = train_data_tree)
```

Comparing R2 and comparing AIC (KB, MH)

```
#Comparing R2
summary(model_cont)$r.squared
```

```
## [1] 0.4770668
```

```
summary(model_bin)$r.squared
```

```
## [1] 0.5411029
```

```
summary(model_temp)$r.squared
```

```
## [1] 0.4690846
```

```
summary(model_temp_ozone)$r.squared
```

```
## [1] 0.5248978
```

```
summary(model_airbin)$r.squared
```

```
## [1] 0.6100282
```

```
summary(model_ozone)$r.squared
```

```
## [1] 0.5774904
```

```
summary(model_wind)$r.squared
```

```
## [1] 0.4676609
```

```
summary(model_wind_air)$r.squared
```

```
## [1] 0.5919024
```

```
#Comparing AIC
AIC(model_cont)
```

```
## [1] 1140.996
```

```
AIC(model_bin)
```

```
## [1] 1131.983
```

```
AIC(model_temp)
```

```
## [1] 1142.041
```

```
AIC(model_temp_ozone)
```

```
## [1] 1134.377
```

```
AIC(model_airbin)
```

```
## [1] 1120.753
```

```
AIC(model_ozone)
```

```
## [1] 1126.282
```

```
AIC(model_wind)
```

```
## [1] 1142.226
```

```
AIC(model_wind_air)
```

```
## [1] 1123.888
```

Caret cross-validation results (TRAINING DATA) (KB, MH)

```
set.seed(123)

train_control <- trainControl(method = "cv", number = 10)

cv_cont <- train(avg_chip_seconds ~ avg_temp + ozone + pm25 + wind_speed + temp_a
qi_interaction,
                 data = train_data_tree,
                 method = "lm",
                 trControl = train_control)

cv_bin <- train(avg_chip_seconds ~ temp_bin + ozone_bin + pm25_bin + wind_bin + t
emp_aqi_bin,
                data = train_data_tree,
                method = "lm",
                trControl = train_control)

cv_temp <- train(avg_chip_seconds ~ temp_bin + ozone + pm25 + wind_speed + temp_a
qi_interaction,
                 data = train_data_tree,
                 method = "lm",
                 trControl = train_control)

cv_temp_ozone <- train(avg_chip_seconds ~ temp_bin + ozone_bin + pm25 + wind_spee
d + temp_aqi_interaction,
                       data = train_data_tree,
                       method = "lm",
                       trControl = train_control)

cv_airbin <- train(avg_chip_seconds ~ avg_temp + ozone_bin + pm25_bin + wind_spee
d + temp_aqi_interaction,
                   data = train_data_tree,
                   method = "lm",
                   trControl = train_control)

cv_ozone <- train(avg_chip_seconds ~ avg_temp + ozone_bin + pm25 + wind_speed + t
emp_aqi_interaction,
                  data = train_data_tree,
                  method = "lm",
                  trControl = train_control)

cv_wind <- train(avg_chip_seconds ~ avg_temp + ozone + pm25 + wind_bin + temp_aqi
_interaction,
                 data = train_data_tree,
                 method = "lm",
                 trControl = train_control)

cv_wind_air <- train(avg_chip_seconds ~ avg_temp + ozone_bin + pm25_bin + wind_bi
n + temp_aqi_interaction,
```

```
                      data = train_data_tree,
                      method = "lm",
                      trControl = train_control)
```

Getting the results (KB)

```
combined_results <- bind_rows(
  cv_cont$results    %>% mutate(Model = "cv_cont"),
  cv_bin$results     %>% mutate(Model = "cv_bin"),
  cv_temp$results    %>% mutate(Model = "cv_temp"),
  cv_temp_ozone$results %>% mutate(Model = "cv_temp_ozone"),
  cv_airbin$results  %>% mutate(Model = "cv_airbin"),
  cv_ozone$results   %>% mutate(Model = "cv_ozone"),
  cv_wind$results    %>% mutate(Model = "cv_wind"),
  cv_wind_air$results %>% mutate(Model = "cv_wind_air")
) %>%
  select(Model, everything())

print(combined_results)
```

```
##           Model intercept     RMSE  Rsquared      MAE   RMSESD RsquaredSD
## 1       cv_cont      TRUE 895.8262 0.4873768 751.7258 237.5673  0.2541559
## 2        cv_bin      TRUE 868.4068 0.5071339 709.1803 186.7728  0.1941312
## 3       cv_temp      TRUE 896.0868 0.4642291 751.9601 182.3937  0.2025344
## 4 cv_temp_ozone      TRUE 857.2706 0.5004747 706.8899 176.2001  0.2333458
## 5     cv_airbin      TRUE 763.2415 0.5920959 635.5167 229.9983  0.2195542
## 6      cv_ozone      TRUE 802.1434 0.6226780 672.7320 184.3297  0.2071750
## 7       cv_wind      TRUE 930.0377 0.4669259 777.1032 222.1436  0.3023181
## 8   cv_wind_air      TRUE 805.9406 0.6037733 671.5632 243.3696  0.1899028
##      MAESD
## 1 214.7159
## 2 106.4979
## 3 180.0086
## 4 127.0137
## 5 171.8409
## 6 136.5862
## 7 209.5921
## 8 185.8482
```

We can see that the model with ozone and PM2.5 binned has the lowest RMSE and MAE, suggesting higher accuracy, and one of the highest R2 to explain variability. The model with just ozone binned has the highest R2, but also slightly higher RMSE and MAE.

Based on these results, we will incorporate the binned ozone and PM2.5 features into our final dataset for modeling and test a couple of initial models with the continuous version of these features, as well as the binned version.

5.2 LASSO Regression (KB)

LASSO is being used to answer the following question: Which version of the air-quality and temperature features should we keep for the final model?

Add our 90/10 training/test split for LASSO (KB, ZD)

```
# split data into train and test
set.seed(123)

train_index <- sample(1:nrow(main_data_fe), size = 0.9 * nrow(main_data_fe)) # us
e a 90/10 split

train_data_scaled <- main_data_fe[train_index, ]
test_data_scaled <- main_data_fe[-train_index, ]
```

Data Scaling for LASSO and later linear regression model (ZD, KB)

```
# Identify numeric predictors to scale (exclude outcome and identifiers)
numeric_vars <- train_data_scaled %>%
  select(where(is.numeric)) %>%
  select(-avg_chip_seconds, -year, -n) %>%
  names()

# scale training data and save scaling parameters
train_scaled <- scale(train_data_scaled[numeric_vars])
train_data_scaled[paste0("scaled_", numeric_vars)] <- train_scaled

train_center <- attr(train_scaled, "scaled:center")
train_scale  <- attr(train_scaled, "scaled:scale")

# Scale test data (only numeric vars that exist in test)
numeric_vars_test <- intersect(numeric_vars, names(test_data_scaled))
test_scaled <- scale(test_data_scaled[numeric_vars_test],
                     center = train_center[numeric_vars_test],
                     scale  = train_scale[numeric_vars_test])
test_data_scaled[paste0("scaled_", numeric_vars_test)] <- test_scaled


summary(train_data_scaled[paste0("scaled_", numeric_vars)])
```

```
##  scaled_avg_temp   scaled_precipitation scaled_dew_point  scaled_wind_speed
##  Min.   :-2.4665   Min.   :-0.2376      Min.   :-1.6105   Min.   :-1.57367
##  1st Qu.:-0.6886   1st Qu.:-0.2376      1st Qu.:-0.7537   1st Qu.:-0.57955
##  Median :-0.1931   Median :-0.2376      Median :-0.0240   Median :-0.08249
##  Mean   : 0.0000   Mean   : 0.0000      Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.5735   3rd Qu.:-0.2376      3rd Qu.: 0.6771   3rd Qu.: 0.41458
##  Max.   : 2.8256   Max.   : 6.1116      Max.   : 2.4515   Max.   : 3.89402
##  scaled_visibility scaled_sea_level_pressure   scaled_co
##  Min.   :-0.1146   Min.   :-2.06242          Min.   :-1.0106
##  1st Qu.:-0.1146   1st Qu.:-0.80425          1st Qu.:-0.6693
##  Median :-0.1146   Median : 0.09045          Median :-0.4134
##  Mean   : 0.0000   Mean   : 0.00000          Mean   : 0.0000
##  3rd Qu.:-0.1146   3rd Qu.: 0.92924          3rd Qu.: 0.2691
##  Max.   : 8.7115   Max.   : 1.82394          Max.   : 3.5964
##   scaled_ozone      scaled_pm25        scaled_no2
##  Min.   :-1.3319   Min.   :-1.8827   Min.   :-2.41721
##  1st Qu.:-0.7161   1st Qu.:-0.6027   1st Qu.:-0.78842
##  Median :-0.2849   Median :-0.2370   Median : 0.02597
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 0.2694   3rd Qu.: 0.4944   3rd Qu.: 0.59605
##  Max.   : 3.5336   Max.   : 3.7858   Max.   : 2.30628
##  scaled_temp_aqi_interaction scaled_avg_temp_gender_interaction
##  Min.   :-1.2552             Min.   :-0.9842
##  1st Qu.:-0.6619             1st Qu.:-0.9842
##  Median :-0.2796             Median : 0.4232
##  Mean   : 0.0000             Mean   : 0.0000
##  3rd Qu.: 0.1835             3rd Qu.: 0.8932
##  Max.   : 3.0146             Max.   : 1.9318
```

We can see that all the continuous variables for the train and test data, not including the identifying variables `year` and `n`, were successfully scaled with means at 0 and standard deviations of 1, showing that the standardization was correctly applied.

Add Binned features based off Decision Tree (ozone_bin and pm25_bin) (KB)

```
# Create bins on original numeric columns
train_data_scaled$ozone_bin <- factor(ifelse(train_data_scaled$ozone >= 38, 1,
0), levels = c(0, 1))
train_data_scaled$pm25_bin  <- factor(ifelse(train_data_scaled$pm25 >= 54, 1, 0),
levels = c(0, 1))

test_data_scaled$ozone_bin <- factor(ifelse(test_data_scaled$ozone >= 38, 1, 0),
levels = c(0, 1))
test_data_scaled$pm25_bin  <- factor(ifelse(test_data_scaled$pm25 >= 54, 1, 0), l
evels = c(0, 1))

# Remove original numeric columns
train_data_scaled <- train_data_scaled %>%
  select(-all_of(numeric_vars))

test_data_scaled <- test_data_scaled %>%
  select(-all_of(numeric_vars_test))
```

We made the factor levels:
- ozone bin: 0 = low, 1 = high - pm25 bin: 0 = low, 1 = high

Prepare the predictor matrix (x) and outcome vector (y) (KB)

```
# Only using ozone and PM2.5 (continuous and binned)
x_train <- model.matrix(avg_chip_seconds ~ scaled_ozone + scaled_pm25 +
                         ozone_bin + pm25_bin, data = train_data_scaled)[,-1]  # r
emove intercept
y_train <- train_data_scaled$avg_chip_seconds

x_test <- model.matrix(avg_chip_seconds ~ scaled_ozone + scaled_pm25 +
                        ozone_bin + pm25_bin, data = test_data_scaled)[,-1]
y_test <- test_data_scaled$avg_chip_seconds
```

Run LASSO with cross-validation to select lambda (KB)

```
set.seed(123)

lasso_cv <- cv.glmnet(x_train, y_train, alpha = 1, standardize = FALSE)  # alread
y scaled

# Find the best lambda
best_lambda <- lasso_cv$lambda.min
best_lambda
```

```
## [1] 24.4284
```

```
# Fit final LASSO model at best lambda
lasso_model <- glmnet(x_train, y_train, alpha = 1, lambda = best_lambda, standard
ize = FALSE)

# Check coefficients
coef(lasso_model)
```

```
## 5 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  1.422713e+04
## scaled_ozone 8.896372e-15
## scaled_pm25  .
## ozone_bin1   .
## pm25_bin1    .
```

```
# Predict on test set
preds <- predict(lasso_model, newx = x_test)

# Evaluate performance
rmse <- sqrt(mean((y_test - preds)^2))
mae  <- mean(abs(y_test - preds))
rmse
```

```
## [1] 3383.16
```

```
mae
```

```
## [1] 2861.152
```

We can see that both binned features and scaled_pm25 were retained showing (.). We can also see that scaled_ozone shrank to nearly 0, which means it did not contribute to the predicting average chip time once the binned values were also included. The model achieved an RMSE of 3383 seconds and a MAE of 2861 seconds, suggesting that good predictive performance.

For a simpler and more interpretable model, we will drop the original numeric features and keep only the binned features based on LASSO results (KB)

```
# Remove scaled_ozone and scaled_pm25 from training and test
train_data_scaled <- train_data_scaled[, !(names(train_data_scaled) %in% c("scale
d_ozone"))]
test_data_scaled  <- test_data_scaled[, !(names(test_data_scaled) %in% c("scaled_
ozone"))]

train_data_scaled <- train_data_scaled[, !(names(train_data_scaled) %in% c("scale
d_pm25"))]
test_data_scaled  <- test_data_scaled[, !(names(test_data_scaled) %in% c("scaled_
pm25"))]

str(train_data_scaled)
```

```
## 'data.frame':    693 obs. of  19 variables:
##  $ n                                : int  6244 3922 3423 290 883 4643 6002 2
815 1587 692 ...
##  $ marathon                         : Factor w/ 3 levels "Boston","Chicag
o",..: 2 2 1 3 1 1 2 1 2 1 ...
##  $ year                             : int  2014 2019 2014 1998 2016 2007 2002
2019 1997 1997 ...
##  $ gender                           : Factor w/ 2 levels "female","male": 1 1
2 2 1 2 2 2 1 1 ...
##  $ subgroup                         : Factor w/ 5 levels "elite","competitiv
e",..: 5 3 4 1 5 3 4 4 4 4 ...
##  $ avg_chip_seconds                 : num  20881 13601 14941 9364 20215 ...
##  $ supershoe                        : Factor w/ 2 levels "0","1": 1 2 1 1 1 1
1 2 1 1 ...
##  $ scaled_avg_temp                  : num  0.287 -0.588 -0.148 -0.121 -0.193
...
##  $ scaled_precipitation             : num  -0.238 -0.238 -0.238 -0.238 -0.238
...
##  $ scaled_dew_point                 : num  0.474 -0.574 -0.59 -0.127 -0.405
...
##  $ scaled_wind_speed                : num  0.4146 1.243 -0.0825 0.746 0.746
...
##  $ scaled_visibility                : num  -0.115 -0.115 -0.115 -0.115 -0.115
...
##  $ scaled_sea_level_pressure        : num  -1.084 -1.643 1.237 0.314 0.957
...
##  $ scaled_co                        : num  -0.413 -0.925 -0.584 1.208 -0.755
...
##  $ scaled_no2                       : num  0.026 -0.87 -0.3 -0.381 0.189 ...
##  $ scaled_temp_aqi_interaction      : num  -0.174 -0.833 0.485 -1.12 -0.28
...
##  $ scaled_avg_temp_gender_interaction: num  -0.984 -0.984 0.904 0.913 -0.984
...
##  $ ozone_bin                        : Factor w/ 2 levels "0","1": 2 1 2 1 2 2
1 2 2 2 ...
##  $ pm25_bin                         : Factor w/ 2 levels "0","1": 1 1 2 1 2 1
2 1 2 1 ...
```

```
str(test_data_scaled)
```

```
## 'data.frame':    77 obs. of  19 variables:
##  $ n                                 : int  159 3481 720 2236 2550 712 1386 25
66 645 2817 ...
##  $ marathon                          : Factor w/ 3 levels "Boston","Chicag
o",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ year                              : int  1996 1996 1998 1998 1998 2000 2002
2002 2004 2006 ...
##  $ gender                            : Factor w/ 2 levels "female","male": 1 1
1 2 2 1 1 1 1 2 ...
##  $ subgroup                          : Factor w/ 5 levels "elite","competitiv
e",..: 1 3 2 2 3 4 2 3 2 2 ...
##  $ avg_chip_seconds                  : num  10661 14792 12376 11093 12697 ...
##  $ supershoe                         : Factor w/ 2 levels "0","1": 1 1 1 1 1 1
1 1 1 1 ...
##  $ scaled_avg_temp                   : num  -1.31 -1.31 -0.442 -0.442 -0.442
...
##  $ scaled_precipitation              : num  -0.238 -0.238 -0.238 -0.238 -0.238
...
##  $ scaled_dew_point                  : num  -1.33 -1.33 0.439 0.439 0.439 ...
##  $ scaled_wind_speed                 : num  0.746 0.746 0.746 0.746 0.746 ...
##  $ scaled_visibility                 : num  -0.115 -0.115 -0.115 -0.115 -0.115
...
##  $ scaled_sea_level_pressure         : num  0.957 0.957 0.342 0.342 0.342 ...
##  $ scaled_co                         : num  -0.0721 -0.0721 0.2691 0.2691 0.26
91 ...
##  $ scaled_no2                        : num  -0.3 -0.3 0.27 0.27 0.27 ...
##  $ scaled_temp_aqi_interaction       : num  -0.288 -0.288 -0.601 -0.601 -0.601
...
##  $ scaled_avg_temp_gender_interaction: num  -0.984 -0.984 -0.984 0.802 0.802
...
##  $ ozone_bin                         : Factor w/ 2 levels "0","1": 2 2 2 2 2 2
1 1 2 2 ...
##  $ pm25_bin                          : Factor w/ 2 levels "0","1": 2 2 1 1 1 2
2 2 2 1 ...
```

We can see that scaled ozone was successfully removed from both train and test data. Now this dataset is ready for our initial linear regression model.

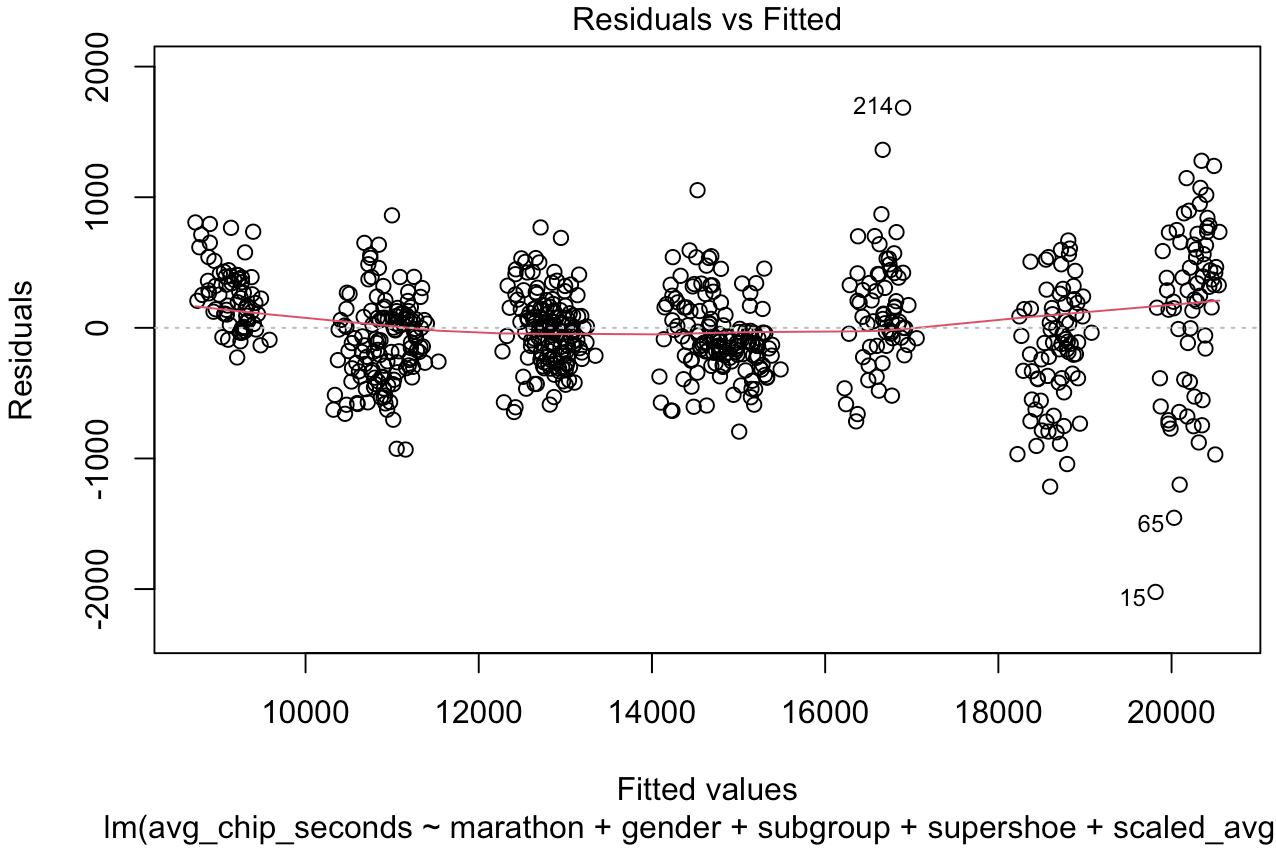
6 Initial Modeling: Linear Regression Model (ZD, MH)

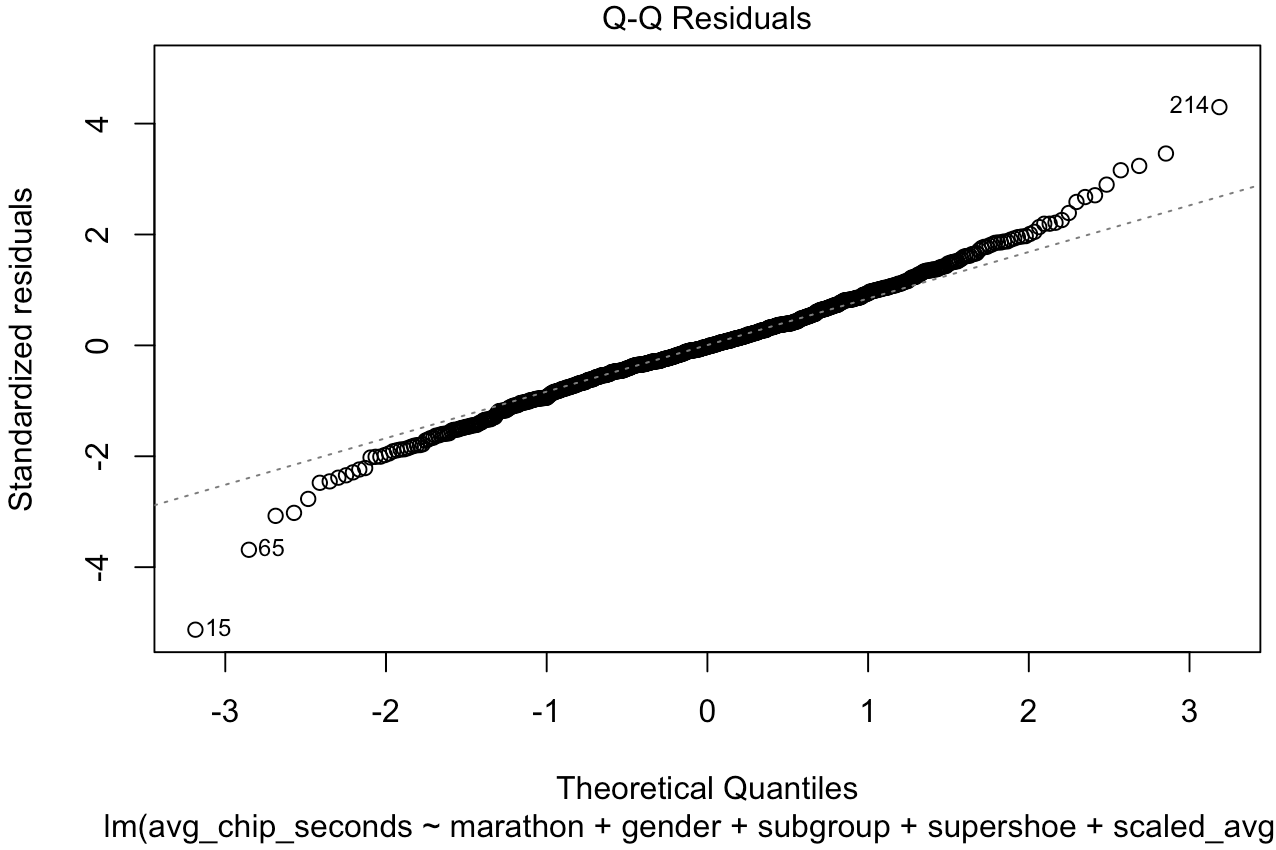
Creating Initial Linear Regression Model (MH)

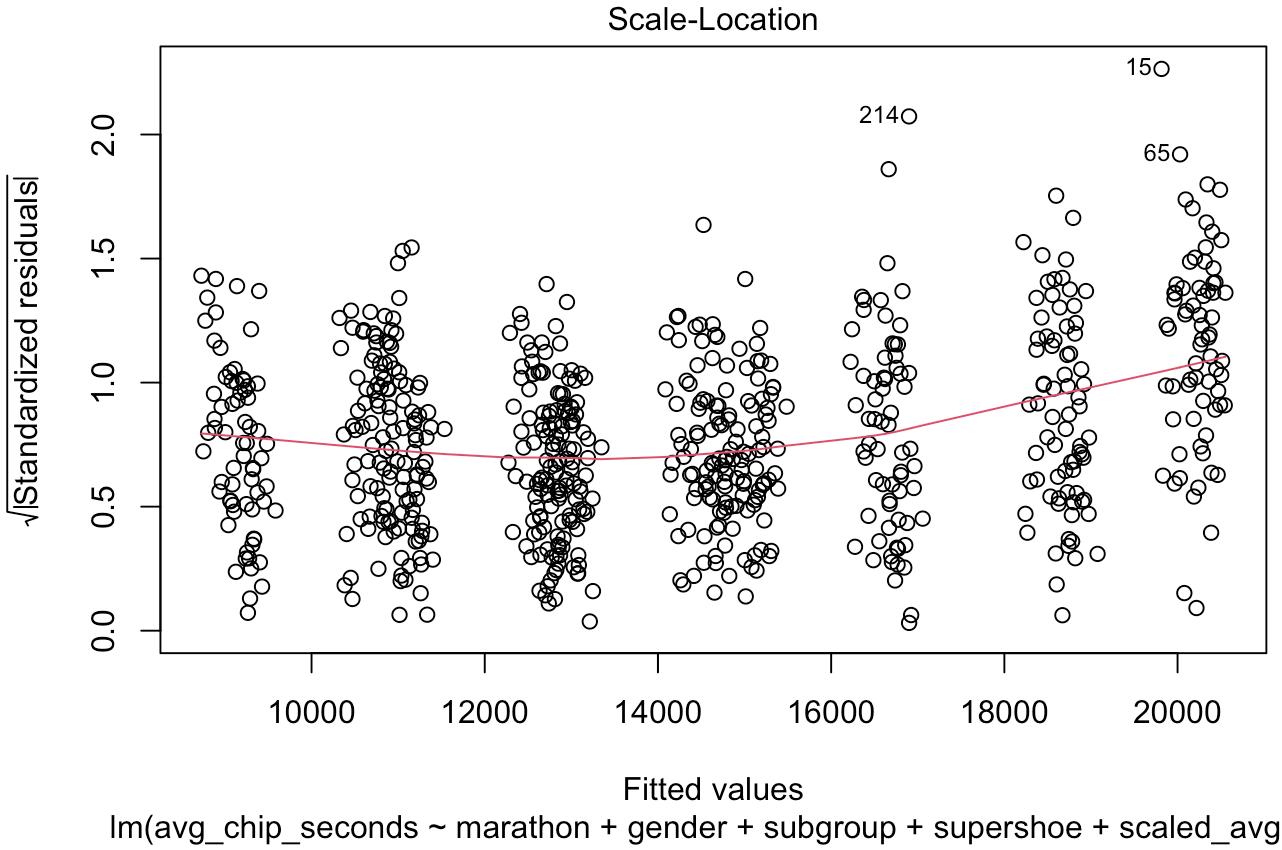
```
lm_initial_PM25bin <- lm(avg_chip_seconds ~ marathon + gender + subgroup + supers
hoe + scaled_avg_temp + scaled_precipitation + scaled_dew_point + scaled_wind_spe
ed + scaled_visibility + scaled_sea_level_pressure + scaled_co + pm25_bin + ozone
_bin + scaled_no2 + scaled_temp_aqi_interaction + scaled_avg_temp_gender_interact
ion, data = train_data_scaled)


summary(lm_initial_PM25bin)
```

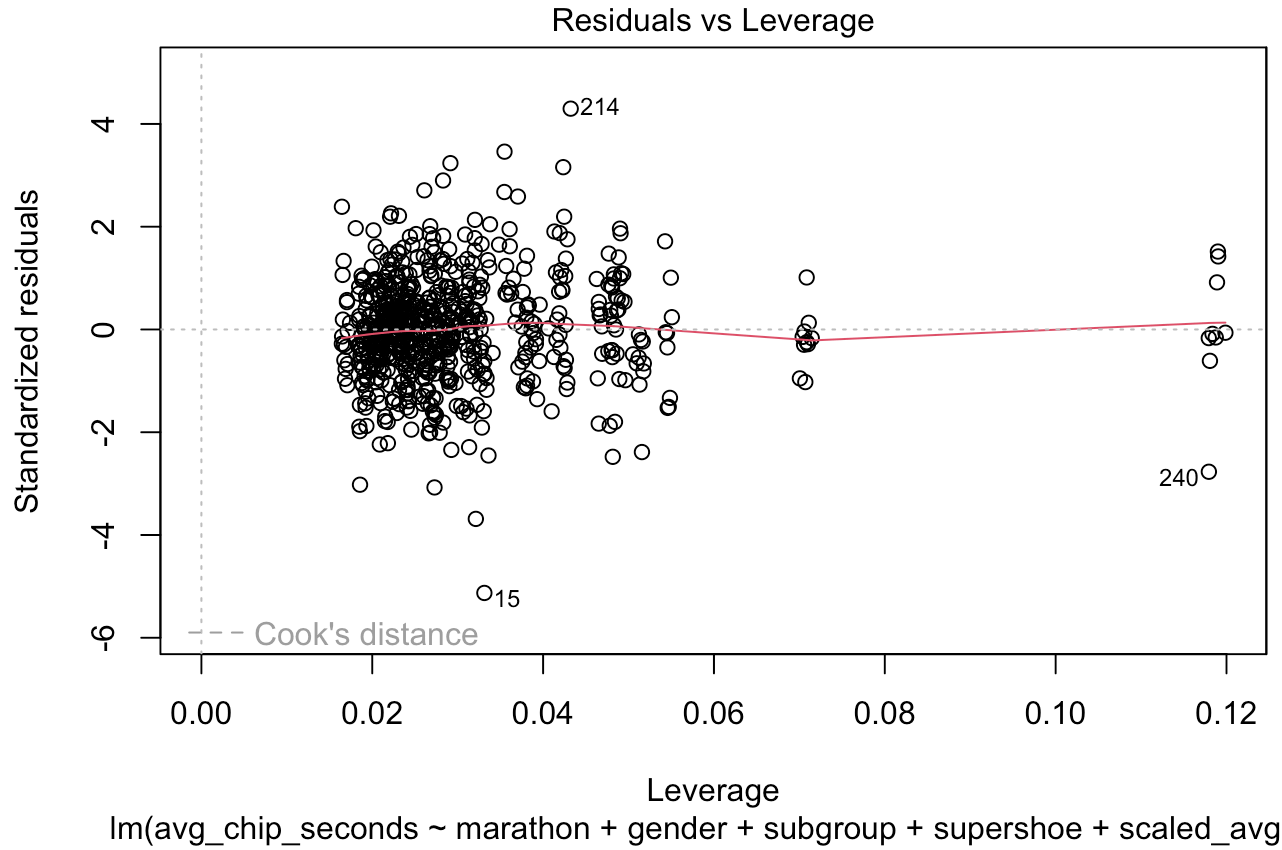
```
##
## Call:
## lm(formula = avg_chip_seconds ~ marathon + gender + subgroup +
##     supershoe + scaled_avg_temp + scaled_precipitation + scaled_dew_point +
##     scaled_wind_speed + scaled_visibility + scaled_sea_level_pressure +
##     scaled_co + pm25_bin + ozone_bin + scaled_no2 + scaled_temp_aqi_interactio
n +
##     scaled_avg_temp_gender_interaction, data = train_data_scaled)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2021.98  -222.10    -9.13   227.41  1685.71
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       10348.22     116.65  88.711  < 2e-16 ***
## marathonChicago                     374.67      79.99   4.684 3.41e-06 ***
## marathonNYC                         609.27      66.09   9.219  < 2e-16 ***
## gendermale                        -1690.84     174.11  -9.711  < 2e-16 ***
## subgroupcompetitive                1951.30      48.80  39.989  < 2e-16 ***
## subgroupaverage                    3763.43      48.28  77.957  < 2e-16 ***
## subgrouprecreational               5901.72      48.43 121.872  < 2e-16 ***
## subgroupslow                       9492.33      47.89 198.218  < 2e-16 ***
## supershoe1                         -155.72      45.82  -3.399 0.000716 ***
## scaled_avg_temp                     -82.71      40.71  -2.032 0.042586 *
## scaled_precipitation                 22.97      17.25   1.332 0.183383
## scaled_dew_point                    107.82      30.38   3.550 0.000413 ***
## scaled_wind_speed                   112.33      21.86   5.138 3.65e-07 ***
## scaled_visibility                    79.96      17.39   4.599 5.08e-06 ***
## scaled_sea_level_pressure           119.25      28.93   4.122 4.22e-05 ***
## scaled_co                           -45.37      23.65  -1.918 0.055521 .
## pm25_bin1                            89.66      45.56   1.968 0.049464 *
## ozone_bin1                          184.35      50.40   3.657 0.000275 ***
## scaled_no2                          -67.35      24.87  -2.708 0.006939 **
## scaled_temp_aqi_interaction          66.48      36.86   1.804 0.071749 .
## scaled_avg_temp_gender_interaction   55.64      88.36   0.630 0.529134
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 401 on 672 degrees of freedom
## Multiple R-squared:  0.9866, Adjusted R-squared:  0.9862
## F-statistic:  2478 on 20 and 672 DF,  p-value: < 2.2e-16
```

```
plot(lm_initial_PM25bin)
```


Residuals vs Fitted
Residuals
Fitted values
lm(avg_chip_seconds ~ marathon + gender + subgroup + supershoe + scaled_avg ...
214
65
15
2000
1000
0
-1000
-2000
10000
12000
14000
16000
18000
20000


Q-Q Residuals
Standardized residuals
Theoretical Quantiles
lm(avg_chip_seconds ~ marathon + gender + subgroup + supershoe + scaled_avg ...
214
65
15
4
2
0
-2
-4
-3
-2
-1
0
1
2
3





The plots mostly satisfy the requirements for linear regression. Next we will test model performance

Adding RMSE and MAE for looking at overfitting with test results below. (ZD,MH)

```
#Train Data
residuals_lm_initial_PM25bin <- lm_initial_PM25bin$residuals
residuals_lm_initial_PM25bin_rmse <- sqrt(mean(residuals_lm_initial_PM25bin^2))
residuals_lm_initial_PM25bin_mae <- mean(abs(residuals_lm_initial_PM25bin))
```

5-fold CV (KB)

```
set.seed(123)
train_control <- trainControl(method = "cv", number = 5)

lm_cv <- train(
  avg_chip_seconds ~ marathon + gender + subgroup + supershoe +
    scaled_avg_temp + scaled_precipitation + scaled_dew_point +
    scaled_wind_speed + scaled_visibility + scaled_sea_level_pressure +
    scaled_co + pm25_bin + ozone_bin + scaled_no2 +
    scaled_temp_aqi_interaction + scaled_avg_temp_gender_interaction,
  data = train_data_scaled,
  method = "lm",
  trControl = train_control
)

cv_rmse <- lm_cv$results$RMSE
cv_r2   <- lm_cv$results$Rsquared
cv_mae  <- lm_cv$results$MAE
```

Checking how well the model works on test data (ZD)

```
# build a function
evaluate <- function(model, test_data_scaled) {
  preds <- predict(model, newdata = test_data_scaled)
  actual <- test_data_scaled$avg_chip_seconds

  rmse <- sqrt(mean((preds - actual)^2))
  mae  <- mean(abs(preds - actual))
  r2   <- cor(preds, actual)^2

  return(list(RMSE = rmse, MAE = mae, R2 = r2, preds  = preds, actual = actual))
}

# Evaluate Model 2 with PM2.5 bin
results_PM25bin <- evaluate(lm_initial_PM25bin, test_data_scaled)
```

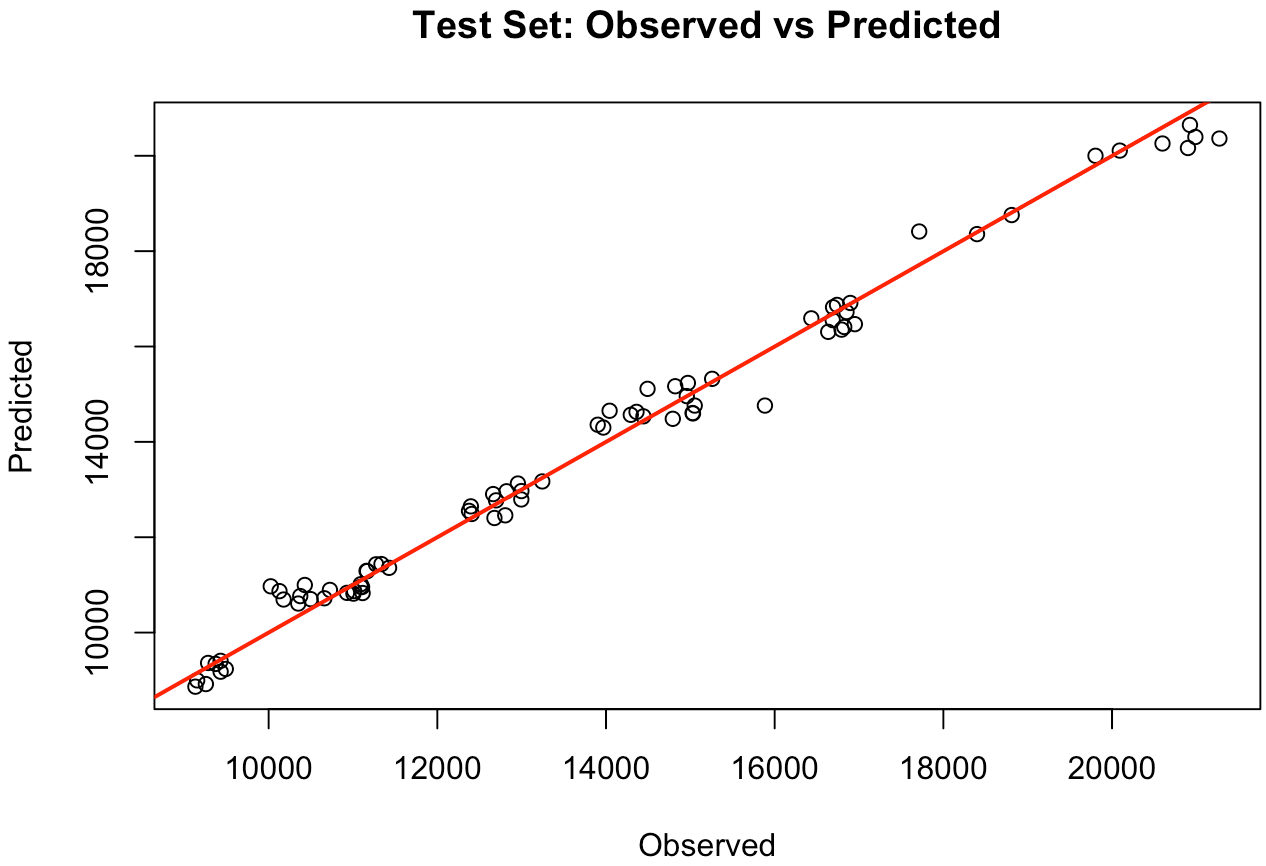
Compare results (ZD/MH)

```
set.seed(123)

# Train metrics
model_compare_train <- data.frame(
  Model = c("PM2.5 Bin Train"),
  RMSE  = c(residuals_lm_initial_PM25bin_rmse),
  MAE   = c(residuals_lm_initial_PM25bin_mae),
  R2    = c(summary(lm_initial_PM25bin)$r.squared)
)

# CV metrics row for the baseline model
model_compare_cv <- data.frame(
  Model = "PM2.5 Bin CV",
  RMSE  = cv_rmse,
  MAE   = cv_mae,
  R2    = cv_r2
)

# Test metrics
model_compare_test <- data.frame(
  Model = "PM2.5 Bin Test",
  RMSE  = results_PM25bin$RMSE,
  MAE   = results_PM25bin$MAE,
  R2    = results_PM25bin$R2
)

# Combine all rows
model_compare_all <- rbind(
  model_compare_train,
  model_compare_cv,
  model_compare_test
)

knitr::kable(model_compare_all, digits = 4,
             caption = "Comparison of Initial Model")
```

Comparison of Initial Model

Model	RMSE	MAE	R2
PM2.5 Bin Train	394.8929	297.1224	0.9866
PM2.5 Bin CV	407.5062	307.7709	0.9861
PM2.5 Bin Test	358.0687	274.0892	0.9892

```
#for plotting
plot(results_PM25bin$actual, results_PM25bin$preds,
     xlab = "Observed",
     ylab = "Predicted",
     main = "Test Set: Observed vs Predicted")
abline(0, 1, col = "red", lwd = 2)
```



The results of the Regression model shows the model is a good fit. The figure of observed v. predicted shows a strong linear relationship, and as the R2 for both train and test data (0.9866, 0.9892 respectively) suggests the model is able to account for almost 99% of the variation the model. This is likely due to subgroups finishing times being manually generated, and being a heavy driver for predicted finshing times which may skew interpretation of the effects of environmental predictors on subgroups and gender. Due to the small size of the dataset, we were unable to have subgroup and or gender as an interaction term for the whole model, therefore we will now investigate XGBoost models to help us develop a model to understand these interactions

7 XGBoost Models (KB, ZD, MH)

7.1 XGBoost Model with Engineered Features (KB,

ZD)

Add our 90/10 training/test split with no scaling (ZD)

```
# split data into train and test
set.seed(42)

train_index <- sample(1:nrow(main_data_fe), size = 0.9 * nrow(main_data_fe)) # us
e a 90/10 split

train_data <- main_data_fe[train_index, ]
test_data  <- main_data_fe[-train_index, ]
```

Add Binned features based off Decision Tree (ozone_bin and pm25_bin) (KB)

```
# Training data
train_data$ozone_bin <- factor(ifelse(train_data$ozone >= 38, 1, 0), levels = c
(0, 1))
train_data$pm25_bin  <- factor(ifelse(train_data$pm25 >= 54, 1, 0), levels = c(0,
1))

# Test data using same thresholds
test_data$ozone_bin <- factor(ifelse(test_data$ozone >= 38, 1, 0), levels = c(0,
1))
test_data$pm25_bin  <- factor(ifelse(test_data$pm25 >= 54, 1, 0), levels = c(0,
1))
```

We made the factor levels:
- ozone bin: 0 = low, 1 = high - pm25 bin: 0 = low, 1 = high

Remove ozone and pm25 based on LASSO results and keeping the ozone and opm25 bins based on decision tree (KB)

```
# Remove ozone and pm25 from training and test data
train_data <- train_data[, !(names(train_data) %in% c("ozone", "pm25"))]
test_data  <- test_data[, !(names(test_data) %in% c("ozone", "pm25"))]

str(train_data)
```

```
## 'data.frame':    693 obs. of  19 variables:
##  $ n                           : int  77 62 4657 1318 4199 269 5155 1603 4888 3
163 ...
##  $ marathon                    : Factor w/ 3 levels "Boston","Chicago",..: 3 2
1 1 1 1 3 1 1 2 ...
##  $ year                        : int  2002 2005 2011 2003 2019 2010 2009 2000 2
008 2003 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 2
2 1 2 2 1 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
1 3 4 3 1 4 4 3 3 ...
##  $ avg_chip_seconds            : num  10485 10318 14201 16716 12785 ...
##  $ avg_temp                    : num  41.3 53.5 52.6 47.3 55.8 ...
##  $ precipitation               : num  0 0 0 0 0.61 0 0 0 0 0 ...
##  $ dew_point                   : num  21.1 42.5 27.7 37.7 49.3 ...
##  $ wind_speed                  : int  13 18 25 25 26 18 9 21 13 16 ...
##  $ visibility                  : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure          : num  30 29.4 30 30.1 29.6 ...
##  $ co                          : num  17 8 5 10 3 3 10 7 7 18 ...
##  $ no2                         : num  44 38 20 43 35 29 30 30 36 45 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1
1 ...
##  $ temp_aqi_interaction        : num  2148 2622 2787 3358 2624 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 55.8 ...
##  $ ozone_bin                   : Factor w/ 2 levels "0","1": 1 1 2 2 2 2 1 2 2
1 ...
##  $ pm25_bin                    : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 2 2
2 ...
```

```
str(test_data)
```

```
## 'data.frame':    77 obs. of  19 variables:
##  $ n                          : int  767 261 2100 712 721 1561 293 798 3096 18
6 ...
##  $ marathon                   : Factor w/ 3 levels "Boston","Chicago",..: 1 1
1 1 1 1 1 1 1 1 ...
##  $ year                       : int  1996 1999 2000 2000 2000 2001 2001 2002 2
005 2007 ...
##  $ gender                     : Factor w/ 2 levels "female","male": 2 2 1 1 2
1 2 2 1 1 ...
##  $ subgroup                   : Factor w/ 5 levels "elite","competitive",..: 1
1 3 4 5 2 1 5 3 1 ...
##  $ avg_chip_seconds           : num  9553 9510 14452 16852 17823 ...
##  $ avg_temp                   : num  40.5 51.1 43 43 43 ...
##  $ precipitation              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point                  : num  24.2 38.4 30.4 30.4 30.4 ...
##  $ wind_speed                 : int  20 15 21 21 21 15 15 15 13 39 ...
##  $ visibility                 : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure         : num  30.2 29.9 30.3 30.3 30.3 ...
##  $ co                         : num  13 14 7 7 7 9 9 13 9 5 ...
##  $ no2                        : num  34 53 30 30 30 42 42 38 39 25 ...
##  $ supershoe                  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ temp_aqi_interaction       : num  2792 2708 2365 2365 2365 ...
##  $ avg_temp_gender_interaction: num  40.5 51.1 0 0 43 ...
##  $ ozone_bin                  : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 1 2
2 ...
##  $ pm25_bin                   : Factor w/ 2 levels "0","1": 2 1 2 2 2 2 2 2 2
1 ...
```

We can see that ozone and pm25 were successfully removed from both datasets and we now have the binned features.

One Hot Encoding and Model Matrix Construction (ZD, KB, MH)

```
# Create numeric training and test matrices with one-hot encoded categorical vari
ables
train_matrix <- train_data %>%
  select(-year, -n, -avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("marathon","subgroup","gender","supershoe","ozone_bin","pm
25_bin"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
 mutate(across(everything(), ~ as.numeric(.))) %>%  # convert all columns to nume
ric
  as.matrix()


test_matrix <- test_data %>%
  select(-year, -n, -avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("marathon","subgroup","gender","supershoe","ozone_bin","pm
25_bin"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
  mutate(across(everything(), ~ as.numeric(.))) %>%  # convert all columns to num
eric
  as.matrix()



# Align columns
test_matrix <- test_matrix[, colnames(train_matrix), drop = FALSE]

dtrain <- xgb.DMatrix(train_matrix, label = train_data$avg_chip_seconds)
dtest  <- xgb.DMatrix(test_matrix,  label = test_data$avg_chip_seconds)
```

7.1.1 Inital Untuned Basline XGBoost Model (ZD, KB)

```
set.seed(123)

xgb_base <- xgboost(
  data = dtrain,
  nrounds = 200,
  objective = "reg:squarederror",
  verbose = FALSE # mute iterations
)

# Predictions on training and testing data
pred_train_xgb <- predict(xgb_base, dtrain)
pred_test_xgb <- predict(xgb_base, dtest)

# Training metrics
rmse_train <- rmse(train_data$avg_chip_seconds, pred_train_xgb)
mae_train  <- mae(train_data$avg_chip_seconds, pred_train_xgb)

# Testing metrics
rmse_test <- rmse(test_data$avg_chip_seconds, pred_test_xgb)
mae_test  <- mae(test_data$avg_chip_seconds, pred_test_xgb)

# Calculate R-squared for train data
rss_train <- sum((train_data$avg_chip_seconds - pred_train_xgb)^2)
tss_train <- sum((train_data$avg_chip_seconds - mean(train_data$avg_chip_second
s))^2)
r2_train <- 1 - (rss_train / tss_train)

# Calculate R-squared for test data
rss_test <- sum((test_data$avg_chip_seconds - pred_test_xgb)^2)  # Residual sum o
f squares
tss_test <- sum((test_data$avg_chip_seconds - mean(test_data$avg_chip_seconds))^
2)  # Total sum of squares
r2_test <- 1 - (rss_test / tss_test)


# Combine metrics into a single table
model_eval <- data.frame(
  Metric = c("Train RMSE", "Train MAE", "Train R2",
             "Test RMSE", "Test MAE", "Test R2"),
  Value  = c(rmse_train, mae_train, r2_train,
             rmse_test, mae_test, r2_test)
)

# Display table nicely
knitr::kable(model_eval, digits = 4,
             caption = "Basline XGBoost Model")
```

Basline XGBoost Model

Metric	Value
Train RMSE	0.9947
Train MAE	0.6970
Train R2	1.0000
Test RMSE	155.7438
Test MAE	98.8799
Test R2	0.9980

We can see that the baseline XGBoost model fits the training data almost perfectly, with a very low Train RMSE of 0.9947 and an R2 of 1.0. However, its performance drops sharply on cross validation and test data, which shows the model is overfitting and memorizing the training data rather than learning patterns that generalize. While the test R2 is still high, the large absolute errors mean predictions on new data would likely be unreliable. Overall, the model works great on the data it has seen but struggles when faced with unseen examples.

7.1.2 Cross-Validation Hyperparameter Tuning (KB)

Do this by running a tuning loop, and used ChatGPT for assitance.

To make sure our XGBoost model is as accurate and reliable as possible, we built a hyperparameter-tuning loop. We can use this instead of guessing which settings would give us the best performance. This loop is able to automatically test several combinations using 5-fold cross-validation. It then checks how well the model predicts on unseen data and records the best RMSE and number of boosting rounds. Then it selects the combination with the lowest cross-validated error. The tuning loop is a balanced model that is flexible enough to capture real patterns and its regularized enough to avoid memorizing all the noise.

Need to undo the #’s if want to run the long loop.

```
#set.seed(42)

#params_grid <- expand.grid(
#  max_depth = c(2, 3, 4),
#  min_child_weight = c(3, 5, 10),
#  subsample = c(0.7, 0.8),
#  colsample_bytree = c(0.6, 0.8),
#  eta = c(0.05, 0.1),
#  gamma = c(0, 0.1),
#  lambda = c(0.5, 1),
#  alpha = c(0, 1)
#)

#results <- list()

#for (i in 1:nrow(params_grid)) {

#  params <- list(
#    objective = "reg:squarederror",
#    max_depth = params_grid$max_depth[i],
#    min_child_weight = params_grid$min_child_weight[i],
#    subsample = params_grid$subsample[i],
#    colsample_bytree = params_grid$colsample_bytree[i],
#    eta = params_grid$eta[i],
#    gamma = params_grid$gamma[i],
#    lambda = params_grid$lambda[i],
#    alpha = params_grid$alpha[i]
#  )

#  cv <- xgb.cv(
#    params = params,
#    data = dtrain,
#    nrounds = 1500,
#    nfold = 5,
#    early_stopping_rounds = 30,
#    verbose = 0
#  )

#  results[[i]] <- list(
#    params = params,
#    best_rmse = min(cv$evaluation_log$test_rmse_mean),
#    best_iter = cv$best_iteration
#  )
#}
```

Next extract the best hyperparameters (KB)

Need to undo the #'s if want to run the long loop and see the best parameters and nrounds.

```
# Find the index of the combination with the lowest CV RMSE
#best_index <- which.min(sapply(results, function(x) x$best_rmse))

# Get the best CV RMSE
#cv_rmse <- results[[best_index]]$best_rmse
#cv_rmse

# Get the best parameters
#best_params <- results[[best_index]]$params
#best_params

# Get the best number of boosting rounds
#best_nrounds <- results[[best_index]]$best_iter
#best_nrounds
```

Output: (set.seed(42))

cv rmse [1] 145.6836

$objective [1] “reg:squarederror”

$max_depth [1] 4

$min_child_weight [1] 5

$subsample [1] 0.8

$colsample_bytree [1] 0.8

$eta [1] 0.1

$gamma [1] 0.1

$lambda [1] 1

$alpha [1] 0

[1] 1248

7.1.3 Final XGBoost Model with Optimal Hyperparameters (KB)

Running XGBoost model with the best hyperparameters. Need to undo the #’s if want to run the long loop and see the model with the best hyperparameters.

```
#set.seed(42)

#watchlist for overfitting test
#watchlist <- list(
#  train = dtrain,
#  eval  = dtest
#)


# Set best hyperparameters
#xgb_model <- xgb.train(
#  params  = best_params,
#  data    = dtrain,
#  nrounds = best_nrounds,
#  watchlist = watchlist,
#  verbose = 0
#)

#extract eval log
#eval_log <- xgb_model$evaluation_log

#Visual overfitting test
#ggplot(eval_log, aes(x = iter)) +
#  geom_line(aes(y = train_rmse, color = "Train")) +
#  geom_line(aes(y = eval_rmse,  color = "Test")) +
#  labs(
#    title = "XGBoost Overfitting Check –Feature Engineered Model",
#    x = "Boosting Round",
#    y = "RMSE"
#  ) +
#  scale_color_manual(values = c("Train" = "blue", "Test" = "red")) +
#  theme_minimal()


# Predictions on train and test sets
#pred_train <- predict(xgb_model, dtrain)
#pred_test  <- predict(xgb_model, dtest)

# Train metrics
#rmse_train <- rmse(train_data$avg_chip_seconds, pred_train)
#mae_train <- mae(train_data$avg_chip_seconds, pred_train)
#rss_train <- sum((train_data$avg_chip_seconds - pred_train)^2)
#tss_train <- sum((train_data$avg_chip_seconds - mean(train_data$avg_chip_second
s))^2)
#r2_train  <- 1 - rss_train / tss_train

# Test metrics
#rmse_test <- rmse(test_data$avg_chip_seconds, pred_test)
```

```
#mae_test  <- mae(test_data$avg_chip_seconds, pred_test)
#rss_test  <- sum((test_data$avg_chip_seconds - pred_test)^2)
#tss_test  <- sum((test_data$avg_chip_seconds - mean(test_data$avg_chip_seconds))
^2)
#r2_test   <- 1 - rss_test / tss_test


#Combine into a single table
#model_eval <- data.frame(
#  Metric = c("Train RMSE", "Train MAE", "Train R2",
#             "CV RMSE",
#             "Test RMSE", "Test MAE", "Test R2"),
#  Value  = c(rmse_train, mae_train, r2_train,
#             cv_rmse,
#             rmse_test, mae_test, r2_test)
#)

#Display table
#knitr::kable(model_eval, digits = 4, caption = "XGBoost (Engineered Features) Mo
del Best Hyperparameters - #Overfitting Check")
```

Output: set.seed(42)

XGBoost (Engineered Features) Model Best Hyperparameters - #Overfitting Check Metric Value Train RMSE 31.2820 Train MAE 23.6533 Train R2 0.9999 CV RMSE 145.6836 Test RMSE 127.4087 Test MAE 82.1464 Test R2 0.9987

Running XGBoost raw feature model with the best hyperparameters explicitly added, so you no longer need any tuning loop since it takes up a lot of time (KB)

```
set.seed(42)

# Set best hyperparameters directly
best_params <- list(
  objective = "reg:squarederror",
  max_depth = 4,
  min_child_weight = 5,
  subsample = 0.8,
  colsample_bytree = 0.8,
  eta = 0.1,
  gamma = 0.1,
  lambda = 1,
  alpha = 0
)

best_nrounds <- 1248   # from updated tuning

# Train final XGBoost model
xgb_model <- xgboost(
  data   = dtrain,
  params = best_params,
  nrounds = best_nrounds,
  verbose = FALSE
)

# Predictions on train and test sets
pred_train <- predict(xgb_model, dtrain)
pred_test  <- predict(xgb_model, dtest)

# Train metrics
rmse_train <- rmse(train_data$avg_chip_seconds, pred_train)
mae_train <- mae(train_data$avg_chip_seconds, pred_train)
rss_train <- sum((train_data$avg_chip_seconds - pred_train)^2)
tss_train <- sum((train_data$avg_chip_seconds - mean(train_data$avg_chip_second
s))^2)
r2_train  <- 1 - rss_train / tss_train

# CV RMSE from tuning
cv_rmse <- 145.6836  # manually use the CV RMSE from latest tuning

# Test metrics
rmse_test <- rmse(test_data$avg_chip_seconds, pred_test)
mae_test  <- mae(test_data$avg_chip_seconds, pred_test)
rss_test  <- sum((test_data$avg_chip_seconds - pred_test)^2)
tss_test  <- sum((test_data$avg_chip_seconds - mean(test_data$avg_chip_seconds))^
2)
r2_test   <- 1 - rss_test / tss_test
```

```
# Combine into a single table
model_eval <- data.frame(
  Metric = c("Train RMSE", "Train MAE", "Train R2",
             "CV RMSE",
             "Test RMSE", "Test MAE", "Test R2"),
  Value  = c(rmse_train, mae_train, r2_train,
             cv_rmse,
             rmse_test, mae_test, r2_test)
)

# Display table
knitr::kable(model_eval, digits = 4, caption = "XGBoost (Engineered Features) Mod
el Best Hyperparameters – Overfitting Check")
```

XGBoost (Engineered Features) Model Best Hyperparameters - Overfitting Check

Metric	Value
Train RMSE	31.2820
Train MAE	23.6533
Train R2	0.9999
CV RMSE	145.6836
Test RMSE	127.4087
Test MAE	82.1464
Test R2	0.9987

NOTE Models may be slightly different with small fluctuations that are likely due to sampling variability.

After tuning, our XGBoost model with engineered features shows substantial improvement over the baseline, though care should still be taken when interpreting training performance metrics.

Although hyperparameter tuning substantially reduced test RMSE, the gap between training and test performance still remains large. This suggests that tuning improved overall predictive accuracy but did not fully eliminate overfitting, which is likely due to the small dataset.

The 5-fold CV RMSE (145.68) is slightly higher than the test RMSE (127.41), indicating the model generalizes reasonably well, though the test set may be somewhat easier to predict.

Both R2 values indicate excellent fit (Train R2 ≈ 0.9999, Test R2 = 0.9987). The nearly perfect training R2 is likely a result of engineered subgroup features capturing much of the variability.

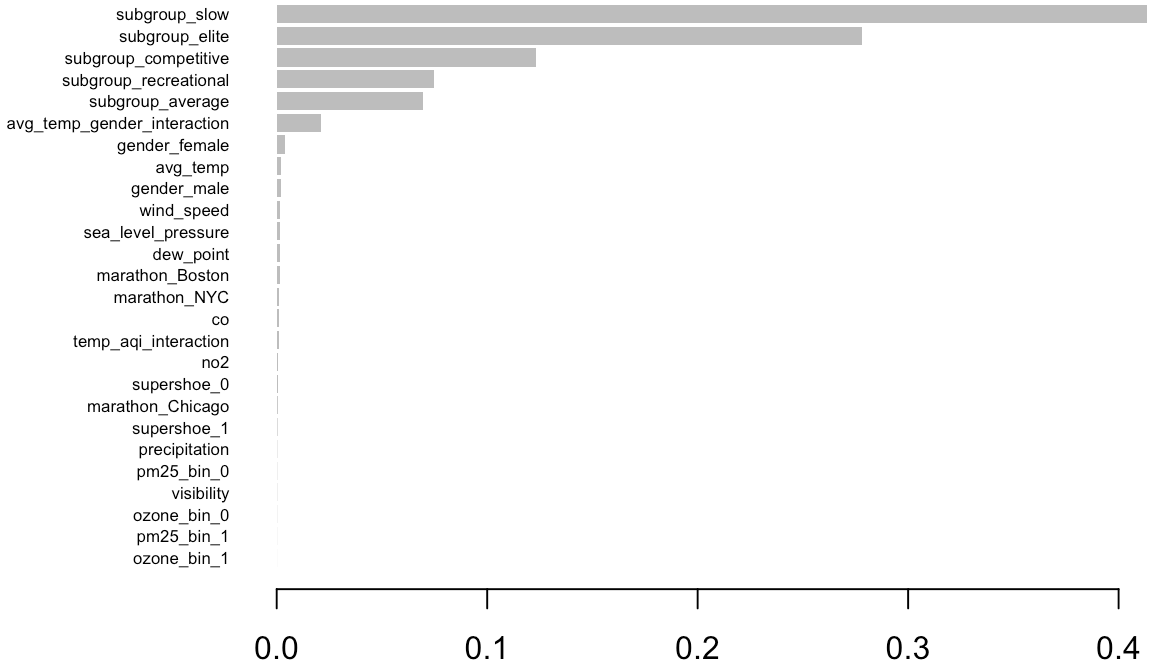
Overall, the tuned model outperforms the baseline, reducing test RMSE from ~155.7 to ~127.4 (~18% reduction), improving predictive accuracy and stability. While training performance remains very strong, the model still shows some overfitting, likely due to the small dataset.

7.1.4 Variable Importance Analysis (ZD, KB)

```
# Variable importance amongst final XGBoosted model
importance <- xgb.importance(model = xgb_model)
print(importance)
```

```
##                        Feature         Gain        Cover   Frequency
##                         <char>        <num>        <num>       <num>
##  1:               subgroup_slow 4.136600e-01 0.0526645228 0.059614488
##  2:              subgroup_elite 2.780016e-01 0.0266750955 0.034043215
##  3:        subgroup_competitive 1.231657e-01 0.0178218765 0.029768382
##  4:       subgroup_recreational 7.496974e-02 0.0233779512 0.029224312
##  5:            subgroup_average 6.933823e-02 0.0140922959 0.021762786
##  6: avg_temp_gender_interaction 2.093127e-02 0.0995486903 0.137338722
##  7:               gender_female 3.769675e-03 0.0087879871 0.021607337
##  8:                    avg_temp 2.206859e-03 0.1479170675 0.131043059
##  9:                 gender_male 2.191460e-03 0.0016205371 0.004274833
## 10:                  wind_speed 1.554213e-03 0.0815322174 0.066842842
## 11:          sea_level_pressure 1.536383e-03 0.1001013294 0.090548733
## 12:                   dew_point 1.532896e-03 0.0912620363 0.089149697
## 13:             marathon_Boston 1.382226e-03 0.0102710973 0.010337323
## 14:                marathon_NYC 1.187030e-03 0.0081528919 0.009093735
## 15:                          co 1.098968e-03 0.0965293315 0.073216229
## 16:        temp_aqi_interaction 1.052730e-03 0.0980073111 0.081299549
## 17:                         no2 7.864654e-04 0.0706725669 0.062023939
## 18:                supershoe_0 5.077717e-04 0.0088319636 0.009171460
## 19:            marathon_Chicago 3.624321e-04 0.0039326059 0.005052075
## 20:                 supershoe_1 2.465485e-04 0.0022955776 0.002564900
## 21:               precipitation 1.297883e-04 0.0117439463 0.009404632
## 22:                  pm25_bin_0 1.202588e-04 0.0066162768 0.008083320
## 23:                  visibility 1.131104e-04 0.0064927759 0.003108969
## 24:                 ozone_bin_0 6.341551e-05 0.0078197693 0.008005596
## 25:                  pm25_bin_1 6.099197e-05 0.0007937773 0.001010415
## 26:                 ozone_bin_1 3.020700e-05 0.0024385015 0.002409451
##                        Feature         Gain        Cover   Frequency
```

```
xgb.plot.importance(importance)
```



We can see that subgroupslow and subgroupelite are the features the model relied on most to reduce prediction error. Variables like avg_temp or sea_level_pressure are less influential but still contribute a little. Features with extremely low Frequency are ozone_bin and pm25_bin both high and low (0 & 1), which are barely used by the model.

7.2 XGBoost Model with Raw Features (KB)

No feature engineering or binning occurred on the main_data.

```
str(main_data)
```

```
## 'data.frame':    770 obs. of  20 variables:
##  $ n                 : int  159 1057 3481 2393 1389 767 5095 8091 6697 6677
...
##  $ marathon          : Factor w/ 3 levels "Boston","Chicago",..: 1 1 1 1 1 1 1
1 1 1 ...
##  $ year              : int  1996 1996 1996 1996 1996 1996 1996 1996 1996 1996
...
##  $ gender            : Factor w/ 2 levels "female","male": 1 1 1 1 1 2 2 2 2 2
...
##  $ subgroup          : Factor w/ 5 levels "elite","competitive",..: 1 2 3 4 5
1 2 3 4 5 ...
##  $ avg_chip_seconds  : num  10661 12931 14792 17030 20673 ...
##  $ high_temp         : int  45 45 45 45 45 45 45 45 45 45 ...
##  $ low_temp          : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ avg_temp          : num  40.5 40.5 40.5 40.5 40.5 ...
##  $ precipitation     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ dew_point         : num  24.2 24.2 24.2 24.2 24.2 ...
##  $ wind_speed        : int  20 20 20 20 20 20 20 20 20 20 ...
##  $ visibility        : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure: num  30.2 30.2 30.2 30.2 30.2 ...
##  $ aqi               : int  69 69 69 69 69 69 69 69 69 69 ...
##  $ main_pollutant    : Factor w/ 4 levels "NO2","Ozone",..: 3 3 3 3 3 3 3 3 3
3 ...
##  $ co                : num  13 13 13 13 13 13 13 13 13 13 ...
##  $ ozone             : num  41 41 41 41 41 41 41 41 41 41 ...
##  $ pm25              : num  56 55 56 55 56 56 55 56 55 55 ...
##  $ no2               : num  34 34 34 34 34 34 34 34 34 34 ...
```

Add our 90/10 training/test split (ZD, KB)

```
# split data into train and test
set.seed(123)

train_index <- sample(1:nrow(main_data), size = 0.9 * nrow(main_data)) # use a 9
0/10 split

train_data_raw <- as.data.frame(main_data[train_index, ]) # making sure they are
data frames
test_data_raw  <- as.data.frame(main_data[-train_index, ])
```

One-Hot Encoding and Model Matrix Construction (ZD, KB, MH)

```
# Create numeric training and test matrices with one-hot encoded categorical vari
ables
train_matrix_raw <- train_data_raw %>%
  select(-year, -n, -avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("marathon","subgroup","gender"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
 mutate(across(everything(), ~ as.numeric(.))) %>%  # convert all columns to nume
ric
  as.matrix()


test_matrix_raw <- test_data_raw %>%
  select(-year, -n, -avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("marathon","subgroup","gender"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
  mutate(across(everything(), ~ as.numeric(.))) %>%  # convert all columns to num
eric
  as.matrix()




# Convert to DMatrix format for tuning, needed for cross validation
dtrain_raw <- xgb.DMatrix(data = train_matrix_raw, label = train_data_raw$avg_chi
p_seconds)
dtest_raw  <- xgb.DMatrix(data = test_matrix_raw,  label = test_data_raw$avg_chip
_seconds)
```

7.2.1 Inital Untuned Basline Raw Features XGBoost Model (ZD, KB)

```
set.seed(42)

xgb_base_raw <- xgboost(
  data = dtrain_raw,
  nrounds = 200,
  objective = "reg:squarederror",
  verbose = FALSE # mute iterations
)

# Predictions on training and testing data
pred_train_xgb_raw <- predict(xgb_base_raw, dtrain_raw)
pred_test_xgb_raw  <- predict(xgb_base_raw, dtest_raw)

# Training metrics
rmse_train_raw <- rmse(train_data_raw$avg_chip_seconds, pred_train_xgb_raw)
mae_train_raw  <- mae(train_data_raw$avg_chip_seconds, pred_train_xgb_raw)

# Testing metrics
rmse_test_raw <- rmse(test_data_raw$avg_chip_seconds, pred_test_xgb_raw)
mae_test_raw  <- mae(test_data_raw$avg_chip_seconds, pred_test_xgb_raw)

# Calculate R-squared for train data
rss_train_raw <- sum((train_data_raw$avg_chip_seconds - pred_train_xgb_raw)^2)
tss_train_raw <- sum((train_data_raw$avg_chip_seconds - mean(train_data_raw$avg_c
hip_seconds))^2)
r2_train_raw <- 1 - (rss_train_raw / tss_train_raw)

# Calculate R-squared for test data
rss_test_raw <- sum((test_data_raw$avg_chip_seconds - pred_test_xgb_raw)^2)
tss_test_raw <- sum((test_data_raw$avg_chip_seconds - mean(test_data_raw$avg_chip
_seconds))^2)
r2_test_raw <- 1 - (rss_test_raw / tss_test_raw)

# Combine metrics into a single table
model_eval_raw <- data.frame(
  Metric = c("Train RMSE", "Train MAE", "Train R2",
             "Test RMSE", "Test MAE", "Test R2"),
  Value  = c(rmse_train_raw, mae_train_raw, r2_train_raw,
             rmse_test_raw, mae_test_raw, r2_test_raw)
)

# Display table nicely
knitr::kable(model_eval_raw, digits = 4,
             caption = "Untuned Baseline XGBoost Model (Raw Features)")
```

Untuned Baseline XGBoost Model (Raw Features)

Metric	Value
Train RMSE	1.3546
Train MAE	0.9630
Train R2	1.0000
Test RMSE	154.7583
Test MAE	112.0259
Test R2	0.9979

We can see the untuned XGBoost model using raw features fits the training data almost perfectly (Train RMSE = 1.35, Train R2 = 1.0), but its performance drops substantially on the test set (Test RMSE = 154.76, Test R2 satay = 0.9979). This large gap between training and test errors indicates that the model is strongly overfitting: it memorizes the training data rather than learning patterns that generalize. While the test R2 is still very high, the absolute errors are large, suggesting predictions on new data would likely be unreliable.

7.2.2 Cross-Validation Hyperparameter Tuning (KB)

Do this by first running a tuning loop on the raw features as done previously (KB)

To make sure our XGBoost raw features model is as accurate and reliable as possible, we built a hyperparameter-tuning loop. We can use this instead of guessing which settings would give us the best performance. This loop is able to automatically test several combinations using 5-fold cross-validation. It then checks how well the model predicts on unseen data and records the best RMSE and number of boosting rounds. Then it selects the combination with the lowest cross-validated error. The tuning loop is a balanced model that is flexible enough to capture real patterns and its regularized enough to avoid memorizing all the noise.

Need to undo the #'s if want to run the long loop.

```
#params_grid_raw <- expand.grid(
#  max_depth = c(2, 3, 4),
#  min_child_weight = c(3, 5, 10),
#  subsample = c(0.7, 0.8),
#  colsample_bytree = c(0.6, 0.8),
#  eta = c(0.05, 0.1),
#  gamma = c(0, 0.1),
#  lambda = c(0.5, 1),
#  alpha = c(0, 1)
#)

#results_raw <- list()

#for (i in 1:nrow(params_grid_raw)) {

#  params <- list(
#    objective = "reg:squarederror",
#    max_depth = params_grid_raw$max_depth[i],
#     min_child_weight = params_grid_raw$min_child_weight[i],
#    subsample = params_grid_raw$subsample[i],
#    colsample_bytree = params_grid_raw$colsample_bytree[i],
#    eta = params_grid_raw$eta[i],
#    gamma = params_grid_raw$gamma[i],
#    lambda = params_grid_raw$lambda[i],
#    alpha = params_grid_raw$alpha[i]
#  )

#  cv_raw <- xgb.cv(
#    params = params,
#    data = dtrain_raw,
#    nrounds = 1500,
#    nfold = 5,
#    early_stopping_rounds = 30,
#    verbose = 0
#  )

#  results_raw[[i]] <- list(
#    params = params,
#    best_rmse = min(cv_raw$evaluation_log$test_rmse_mean),
#    best_iter = cv_raw$best_iteration
#  )
#}
```

Next extract the best hyperparameters (KB)

Need to undo the #'s if want to run the long loop and see the best parameters and nrounds.

```
# Find the index of the combination with the lowest CV RMSE
#best_index_raw <- which.min(sapply(results_raw, function(x) x$best_rmse))

# Get the best CV RMSE
#cv_rmse_raw <- results[[best_index_raw]]$best_rmse
#cv_rmse_raw

# Get the best parameters
#best_params_raw <- results_raw[[best_index_raw]]$params
#best_params_raw

# Get the best number of boosting rounds
#best_nrounds_raw <- results_raw[[best_index_raw]]$best_iter
#best_nrounds_raw
```

Output: set.seed(42)

cv rmse [1] 177.889

$objective [1] “reg:squarederror”

$max_depth [1] 4

$min_child_weight [1] 3

$subsample [1] 0.8

$colsample_bytree [1] 0.8

$eta [1] 0.1

$gamma [1] 0.1

$lambda [1] 0.5

$alpha [1] 0

[1] 1499

7.2.3 Final XGBoost Model with Optimal Hyperparameters (KB)

Running XGBoost model with the best hyperparameters. Need to undo the #’s if want to run the long loop and see the model with the best hyperparameters.

```
#set.seed(42)

# Set best hyperparameters for raw features
#xgb_raw_model <- xgb.train(
#  params  = best_params_raw,
#  data    = dtrain_raw,
#  nrounds = best_nrounds_raw,
#  verbose = 0
#)

# Predictions on train and test sets
#pred_train_raw <- predict(xgb_raw_model, dtrain_raw)
#pred_test_raw  <- predict(xgb_raw_model, dtest_raw)

# Train metrics
#rmse_train <- rmse(train_data_raw$avg_chip_seconds, pred_train_raw)
#mae_train  <- mae(train_data_raw$avg_chip_seconds, pred_train_raw)
#rss_train  <- sum((train_data_raw$avg_chip_seconds - pred_train_raw)^2)
#tss_train  <- sum((train_data_raw$avg_chip_seconds - mean(train_data_raw$avg_chi
p_seconds))^2)
#r2_train   <- 1 - rss_train / tss_train

# Test metrics
#rmse_test <- rmse(test_data_raw$avg_chip_seconds, pred_test_raw)
#mae_test  <- mae(test_data_raw$avg_chip_seconds, pred_test_raw)
#rss_test  <- sum((test_data_raw$avg_chip_seconds - pred_test_raw)^2)
#tss_test  <- sum((test_data_raw$avg_chip_seconds - mean(test_data_raw$avg_chip_s
econds))^2)
#r2_test   <- 1 - rss_test / tss_test

# Combine into a single table
#model_eval <- data.frame(
#  Metric = c("Train RMSE", "Train MAE", "Train R2",
#             "CV RMSE",
#             "Test RMSE", "Test MAE", "Test R2"),
#  Value  = c(rmse_train, mae_train, r2_train,
#             cv_rmse_raw,
#             rmse_test, mae_test, r2_test)
#)

# Display table
#knitr::kable(model_eval, digits = 4, caption = "XGBoost Raw-Feature Model Best H
yperparameters - Overfitting Check")
```

Output: set.seed(42)

XGBoost Raw-Feature Model Best Hyperparameters - Overfitting Check Metric Value Train RMSE 30.6632 Train MAE 23.7209 Train R2 0.9999 CV RMSE 177.8890 Test RMSE 120.8608 Test MAE 90.1244 Test R2 0.9987

Running XGBoost raw feature model with the best hyperparameters explicitly added, so you no longer need any tuning loop since it takes up a lot of time (KB)

```
set.seed(42)

# Set best hyperparameters directly
best_params_raw <- list(
  objective = "reg:squarederror",
  max_depth = 4,
  min_child_weight = 3,
  subsample = 0.8,
  colsample_bytree = 0.8,
  eta = 0.1,
  gamma = 0.1,
  lambda = 0.5,
  alpha = 0
)

best_nrounds_raw <- 1499  # optimal rounds from CV

# Train final XGBoost model
xgb_raw_model <- xgboost(
  data = dtrain_raw,
  params = best_params_raw,
  nrounds = best_nrounds_raw,
  verbose = FALSE
)

# Predictions on train and test sets
pred_train_raw <- predict(xgb_raw_model, dtrain_raw)
pred_test_raw  <- predict(xgb_raw_model, dtest_raw)

# Train metrics (use train_data_raw)
rmse_train <- rmse(train_data_raw$avg_chip_seconds, pred_train_raw)
mae_train  <- mae(train_data_raw$avg_chip_seconds, pred_train_raw)
rss_train  <- sum((train_data_raw$avg_chip_seconds - pred_train_raw)^2)
tss_train  <- sum((train_data_raw$avg_chip_seconds - mean(train_data_raw$avg_chip
_seconds))^2)
r2_train   <- 1 - rss_train / tss_train

# CV RMSE from previous tuning
cv_rmse_raw <- 177.8890  # manually use CV RMSE from previous results

# Test metrics (use test_data_raw)
rmse_test <- rmse(test_data_raw$avg_chip_seconds, pred_test_raw)
mae_test  <- mae(test_data_raw$avg_chip_seconds, pred_test_raw)
rss_test  <- sum((test_data_raw$avg_chip_seconds - pred_test_raw)^2)
tss_test  <- sum((test_data_raw$avg_chip_seconds - mean(test_data_raw$avg_chip_se
conds))^2)
r2_test   <- 1 - rss_test / tss_test
```

```
# Combine into a single table
model_eval <- data.frame(
  Metric = c("Train RMSE", "Train MAE", "Train R2",
             "CV RMSE",
             "Test RMSE", "Test MAE", "Test R2"),
  Value  = c(rmse_train, mae_train, r2_train,
             cv_rmse_raw,
             rmse_test, mae_test, r2_test)
)

# Display table
knitr::kable(
  model_eval,
  digits = 4,
  caption = "XGBoost Raw-Feature Model Best Hyperparameters - Overfitting Check"
)
```

XGBoost Raw-Feature Model Best Hyperparameters - Overfitting Check

Metric	Value
Train RMSE	30.6632
Train MAE	23.7209
Train R2	0.9999
CV RMSE	177.8890
Test RMSE	120.8608
Test MAE	90.1244
Test R2	0.9987

NOTE Models may be slightly different with small fluctuations that are likely due to sampling variability.

After tuning, our XGBoost model shows substantial improvement over the baseline, but some caution is still warranted regarding overfitting, so care should be taken when interpreting training performance metrics.

Although hyperparameter tuning substantially reduced test RMSE, the gap between training and test performance still remains large. This suggests that tuning improved overall predictive accuracy but did not fully eliminate overfitting, which is likely due to the small dataset. (Train RMSE ≈ 30.66, Test RMSE ≈ 120.86).

The 5-fold CV RMSE (177.89) is higher than the test RMSE (120.86), indicating the model generalizes reasonably well, though the test set may be somewhat easier to predict.

Both R2 values indicate excellent fit (Train R2 ≈ 0.9999, Test R2 ≈ 0.9987). The nearly perfect training R2 is likely a result of the raw features capturing much of the variability rather than overfitting.

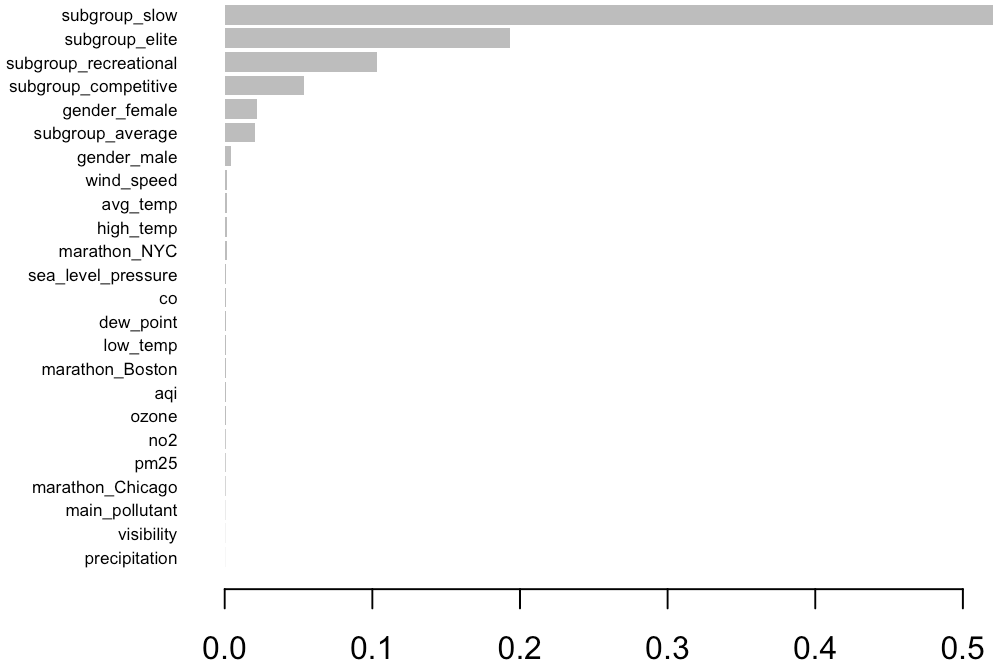
Overall, the tuned raw-feature XGBoost model outperforms the untuned baseline, reducing test RMSE from ~154.76 to ~120.86 (~22% reduction), improving predictive accuracy and stability. While training performance remains very strong, the model still shows some overfitting, likely due to the small dataset.

7.2.4 Variable Importance Analysis (ZD, KB)

```
# Variable importance amongst final XGBoosted model
importance <- xgb.importance(model = xgb_raw_model)
print(importance)
```

```
##                   Feature         Gain       Cover   Frequency
##                    <char>        <num>       <num>       <num>
##  1:          subgroup_slow 0.5897253023 0.051026661 0.068238213
##  2:         subgroup_elite 0.1932764202 0.023126808 0.039347749
##  3: subgroup_recreational 0.1030450579 0.023871904 0.036925440
##  4:  subgroup_competitive 0.0537390835 0.013354396 0.026881720
##  5:          gender_female 0.0216921400 0.037633130 0.108176769
##  6:       subgroup_average 0.0204849275 0.012521320 0.027354366
##  7:            gender_male 0.0045361145 0.009813975 0.024164008
##  8:             wind_speed 0.0018630750 0.065515994 0.052581827
##  9:               avg_temp 0.0013470787 0.087601609 0.058371736
## 10:              high_temp 0.0012630902 0.073179929 0.080586081
## 11:           marathon_NYC 0.0012572520 0.007577169 0.005376344
## 12:    sea_level_pressure 0.0011613556 0.083518444 0.067647406
## 13:                    co 0.0010532179 0.092037905 0.061207610
## 14:             dew_point 0.0009731345 0.065856081 0.057131041
## 15:              low_temp 0.0008401456 0.071266524 0.064397968
## 16:       marathon_Boston 0.0007060872 0.007192789 0.005553586
## 17:                   aqi 0.0006247914 0.063220030 0.055004136
## 18:                 ozone 0.0006208230 0.066666100 0.050336760
## 19:                   no2 0.0005284951 0.057176133 0.045196739
## 20:                  pm25 0.0004836165 0.058310464 0.043247076
## 21:      marathon_Chicago 0.0004153880 0.003659345 0.003367600
## 22:        main_pollutant 0.0001845341 0.012037735 0.010280043
## 23:            visibility 0.0001182190 0.003175457 0.001477018
## 24:         precipitation 0.0000606503 0.010660096 0.007148765
##                   Feature         Gain       Cover   Frequency
```

```
xgb.plot.importance(importance)
```



We can see that subgroupslow and subgroupelite are the features the model relied on most to reduce prediction error. Variables like wind_speed or avg_temp are less influential but still contribute a little. Features with extremely low Frequency are main_pollutant, visibility, and precipitation, which are barely used by the model.

8 Comparison of Models (KB)

```
# Comparison table of Raw vs Engineered XGBoost models + Linear Regression baseli
ne
comparison_table <- data.frame(
  Metric = c(
    "Train RMSE", "Train MAE", "Train R²",
    "CV RMSE",
    "Test RMSE", "Test MAE", "Test R²"
  ),
  Raw_Feature_Model = c(
    30.6632, 23.7209, 0.9999,
    177.8890,
    120.8608, 90.1244, 0.9987
  ),
  Engineered_Feature_Model = c(
    31.2820, 23.6533, 0.9999,
    145.6836,
    127.4087, 82.1464, 0.9987
  ),
  Linear_Regression_Baseline = c(
    394.8929, 297.1224, 0.9866,
    407.5062,
    358.0687, 274.0892, 0.9892
  )
)

# Display table
knitr::kable(
  comparison_table,
  digits = 4,
  caption = "Comparison of XGBoost Raw-Feature, Engineered-Feature Models and Lin
ear Regression Baseline"
)
```

Comparison of XGBoost Raw-Feature, Engineered-Feature Models and Linear Regression Baseline

Metric	Raw_Feature_Model	Engineered_Feature_Model	Linear_Regression_Baseline
Train RMSE	30.6632	31.2820	394.8929
Train MAE	23.7209	23.6533	297.1224
Train R²	0.9999	0.9999	0.9866
CV RMSE	177.8890	145.6836	407.5062
Test RMSE	120.8608	127.4087	358.0687
Test MAE	90.1244	82.1464	274.0892
Test R²	0.9987	0.9987	0.9892

After evaluating both the raw-feature and engineered-feature XGBoost models, clear differences in performance and stability can be seen.

The engineered-feature model has slightly higher training error (Train RMSE ≈ 31.28, Train MAE ≈ 23.65) compared to the raw-feature model (Train RMSE ≈ 30.66, Train MAE ≈ 23.72), but it achieves better cross-validation performance (CV RMSE ≈ 145.68 vs. 177.89). This indicates that the engineered features help the model generalize more consistently across folds, even if training error is similar.

The engineered-feature model also achieves slightly higher test RMSE (127.41 vs. 120.86) but lower test MAE (82.15 vs. 90.12) than the raw-feature model, demonstrating more consistent predictive behavior on unseen data. Both XGBoost models achieve extremely high R^2 values on training and test sets (≈0.998–0.999), capturing nearly all variability in the target variable.

For comparison, the linear regression baseline shows substantially higher errors (Train RMSE ≈ 394.89, Train MAE ≈ 297.12, CV RMSE ≈ 407.51, Test RMSE ≈ 358.07, Test MAE ≈ 274.09) and slightly lower R^2 (≈0.986–0.989), highlighting that while it captures general trends, it cannot match the accuracy and stability of the XGBoost models.

Overall, the engineered-feature XGBoost model is recommended, as it balances low errors, better generalization, and stable performance, whereas the raw-feature model performs slightly worse on CV and test metrics, and the linear regression baseline performs poorly by comparison.

9 Model Interpretation and Feature Analysis (MH / ZD)

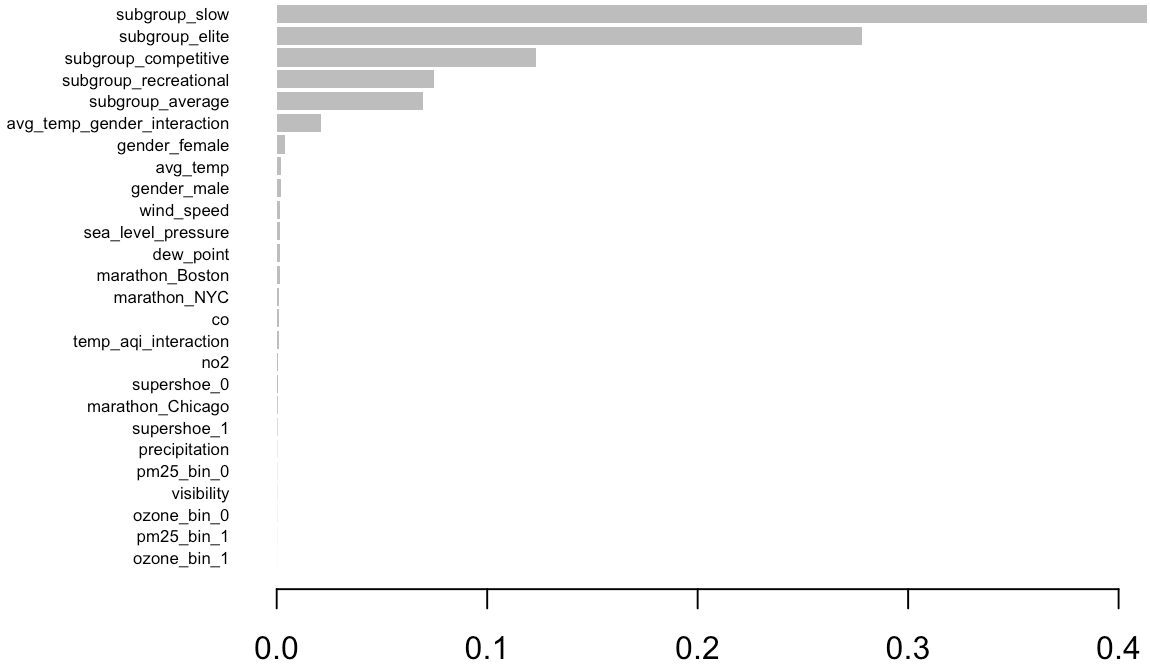
9.1 Feature Importance: Gain, Cover, and Frequency (MH)

Using xgb_model:

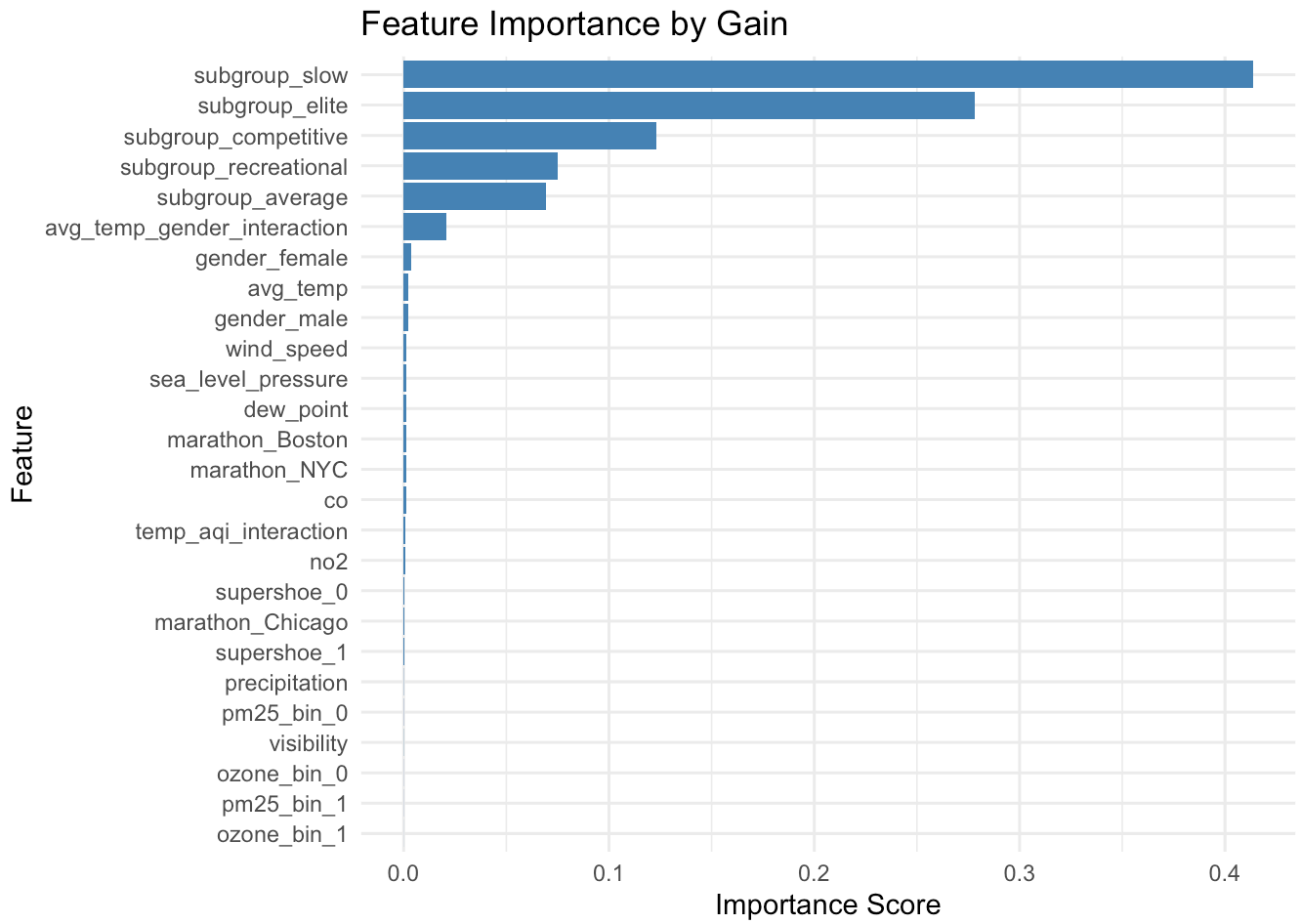
```
importance_Model <- xgb.importance(model = xgb_model)
importance_Model
```

```
##                        Feature         Gain        Cover   Frequency
##                         <char>        <num>        <num>       <num>
##  1:               subgroup_slow 4.136600e-01 0.0526645228 0.059614488
##  2:             subgroup_elite 2.780016e-01 0.0266750955 0.034043215
##  3:       subgroup_competitive 1.231657e-01 0.0178218765 0.029768382
##  4:      subgroup_recreational 7.496974e-02 0.0233779512 0.029224312
##  5:           subgroup_average 6.933823e-02 0.0140922959 0.021762786
##  6: avg_temp_gender_interaction 2.093127e-02 0.0995486903 0.137338722
##  7:              gender_female 3.769675e-03 0.0087879871 0.021607337
##  8:                   avg_temp 2.206859e-03 0.1479170675 0.131043059
##  9:                gender_male 2.191460e-03 0.0016205371 0.004274833
## 10:                 wind_speed 1.554213e-03 0.0815322174 0.066842842
## 11:         sea_level_pressure 1.536383e-03 0.1001013294 0.090548733
## 12:                  dew_point 1.532896e-03 0.0912620363 0.089149697
## 13:            marathon_Boston 1.382226e-03 0.0102710973 0.010337323
## 14:               marathon_NYC 1.187030e-03 0.0081528919 0.009093735
## 15:                         co 1.098968e-03 0.0965293315 0.073216229
## 16:       temp_aqi_interaction 1.052730e-03 0.0980073111 0.081299549
## 17:                        no2 7.864654e-04 0.0706725669 0.062023939
## 18:                supershoe_0 5.077717e-04 0.0088319636 0.009171460
## 19:           marathon_Chicago 3.624321e-04 0.0039326059 0.005052075
## 20:                supershoe_1 2.465485e-04 0.0022955776 0.002564900
## 21:              precipitation 1.297883e-04 0.0117439463 0.009404632
## 22:                 pm25_bin_0 1.202588e-04 0.0066162768 0.008083320
## 23:                 visibility 1.131104e-04 0.0064927759 0.003108969
## 24:                ozone_bin_0 6.341551e-05 0.0078197693 0.008005596
## 25:                 pm25_bin_1 6.099197e-05 0.0007937773 0.001010415
## 26:                ozone_bin_1 3.020700e-05 0.0024385015 0.002409451
##                        Feature         Gain        Cover   Frequency
```

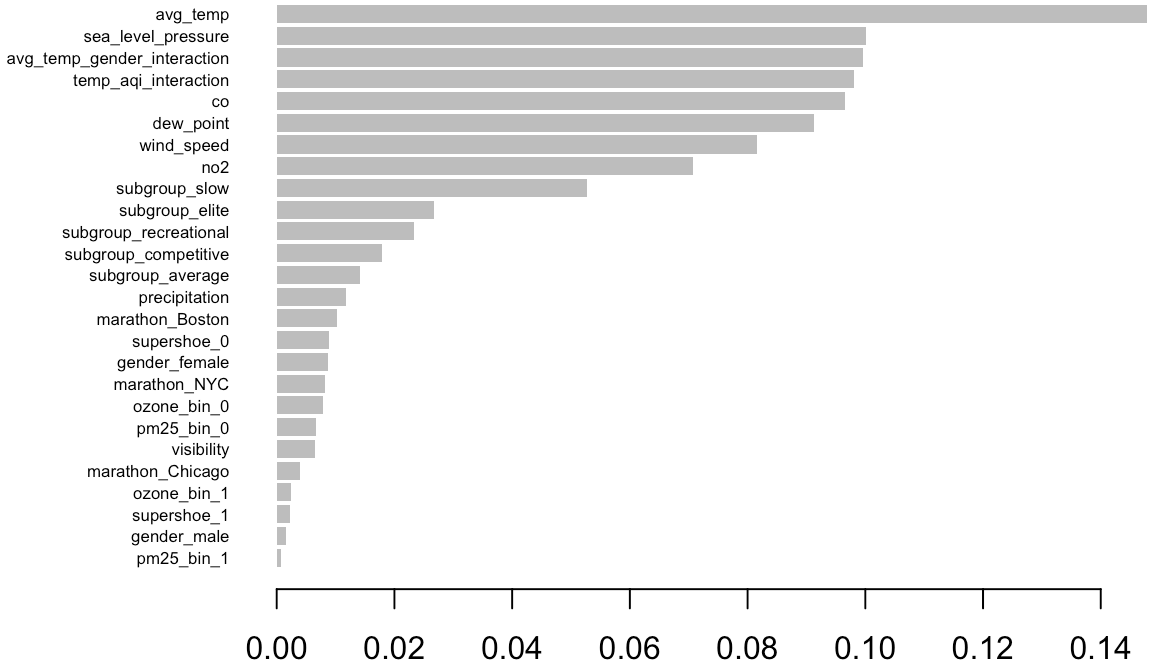
```
#Gain
gain_fig_xgb <- xgb.plot.importance(importance_Model, measure ="Gain")
```



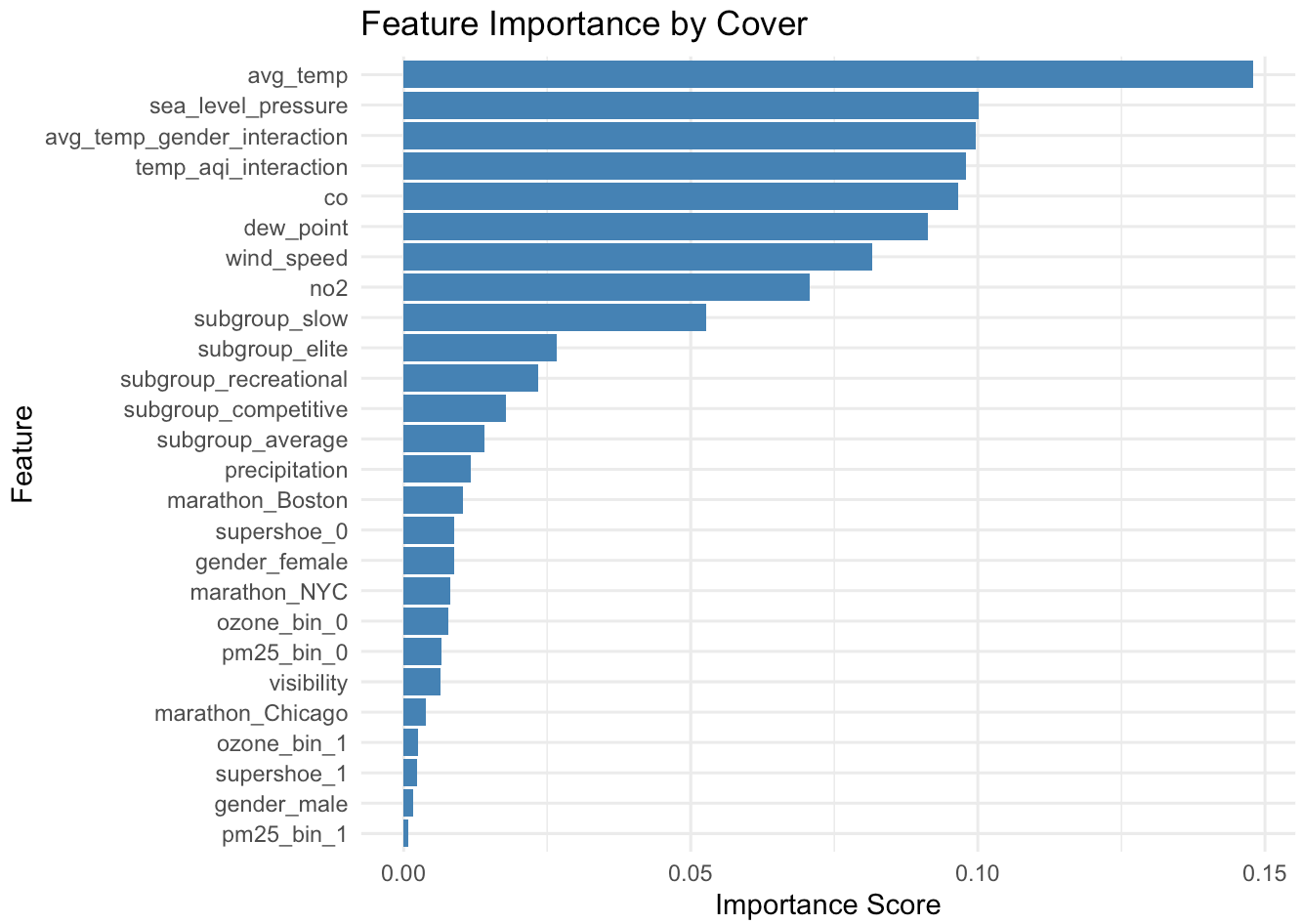

```
#Used ChatGPT since I couldn't figure out how to add labels to the xgb.plot.impor
tance() function
gain_fig <- ggplot(importance_Model, aes(x = reorder(Feature, Gain), y = Gain)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Feature Importance by Gain",
    x = "Feature",
    y = "Importance Score"
  ) +
  theme_minimal()

print(gain_fig)
```



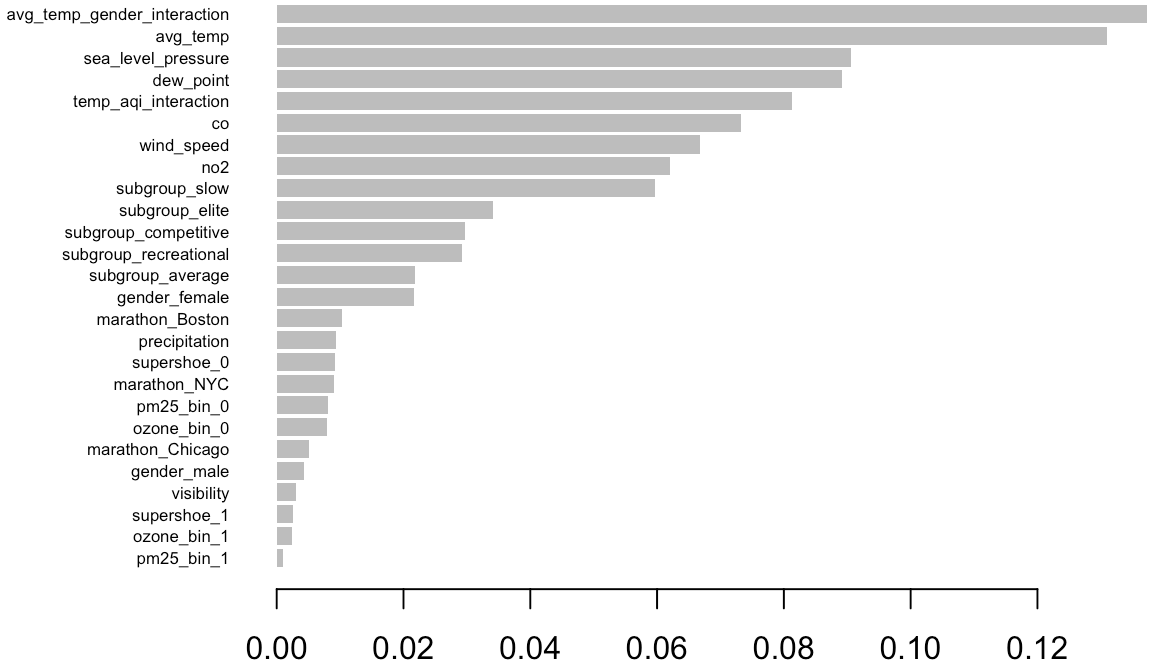

```
#Cover
cover_fig_xgb <- xgb.plot.importance(importance_Model, measure ="Cover")
```



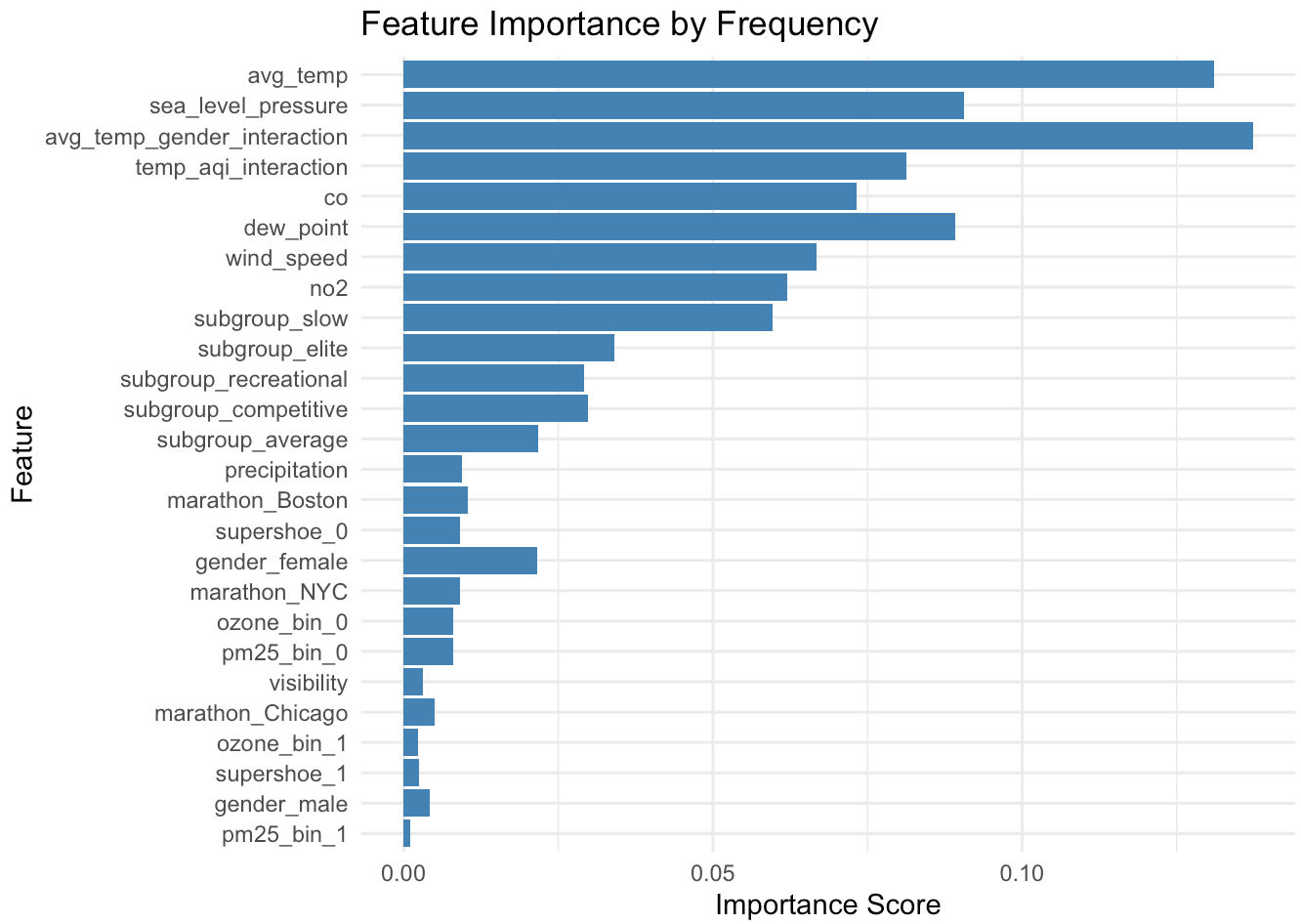

```
cover_fig <- ggplot(importance_Model, aes(x = reorder(Feature, Cover), y = Cove
r)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Feature Importance by Cover",
    x = "Feature",
    y = "Importance Score"
  ) +
  theme_minimal()

print(cover_fig)
```




```
#Frequency (aka weight)
freq_fig_xgb <- xgb.plot.importance(importance_Model, measure ="Frequency")
```




```
freq_fig <- ggplot(importance_Model, aes(x = reorder(Feature, Cover), y = Frequen
cy)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(
    title = "Feature Importance by Frequency",
    x = "Feature",
    y = "Importance Score"
  ) +
  theme_minimal()

print(freq_fig)
```

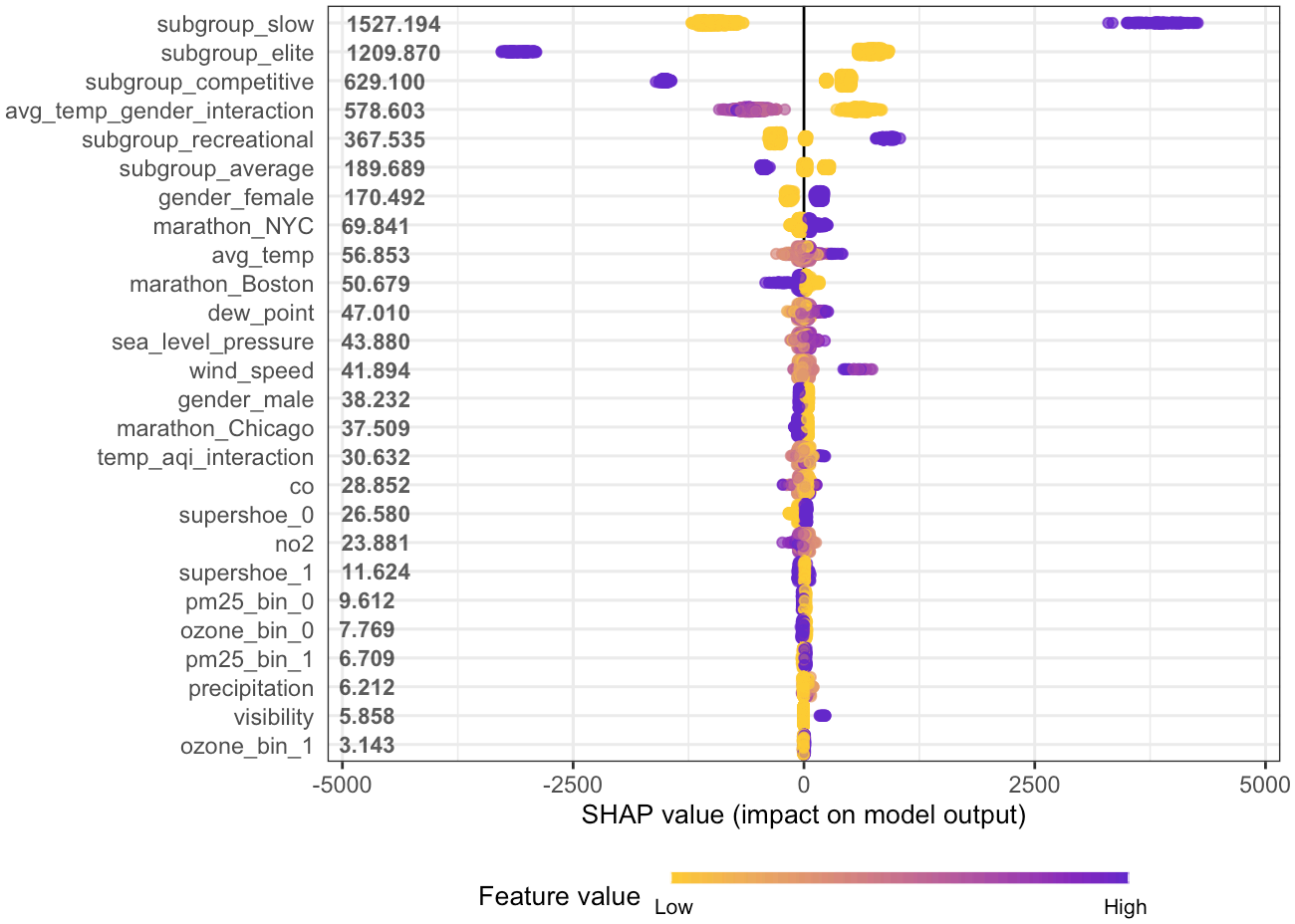


The table shows the breakdown of the gain, cover and frequency (also known as weight). Gain shows the average improvement in the model performance when the feature is used, frequency shows the number of times the feature has been used in a split, and coverage shows the average number of instances affected by splits that use the feature. Based on these results the most important features for prediction are the running performance groups. The frequency and cover figures are helpful for seeing which environmental features are driving the tree once subgroups are split out. Temperature/Gender interaction, Average temperature, dewpoint are consistently high in all three figures.

9.2 SHAP Analysis (ZD)

Implement SHAP:

```
# SHAP
# Compute SHAP values for the raw XGB model
shap_values <- shap.values(
  xgb_model = xgb_model,
  X_train = train_matrix
)

shap_long <- shap.prep( # convert SHAP values to long format
  xgb_model = xgb_model,
  X_train = train_matrix
)
shap.plot.summary(shap_long)
```



SHAP values to left of 0 (negative) represent decreases in predicted finishing time, while SHAP values to the right of 0 (positive) represent increases in predicted finishing time.

Marathon location proves to be important in determining race results. Elite runners strongly decreased in predicted finishing time, while slow runners strongly increased in predicted finishing time. Temperature, ozone, dew point, and PM2.5 influence finishing time but their effects are limited compared to the subgrouping impacts.

9.3 Feature Interactions by Subgroup and Gender (MH

/ ZD)

```
# PM2.5 effect by subgroup
#Because we want to color by subgroup and our subgroups are one-hot encoded, the
best way we found to do this is extrapolate for each one-hot encoded group and pl
ot all together with ggplot

#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "pm25_bin_1",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "pm25_bin_1",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "pm25_bin_1",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "pm25_bin_1",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)
```

```
#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, linewidth = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "PM2.5", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

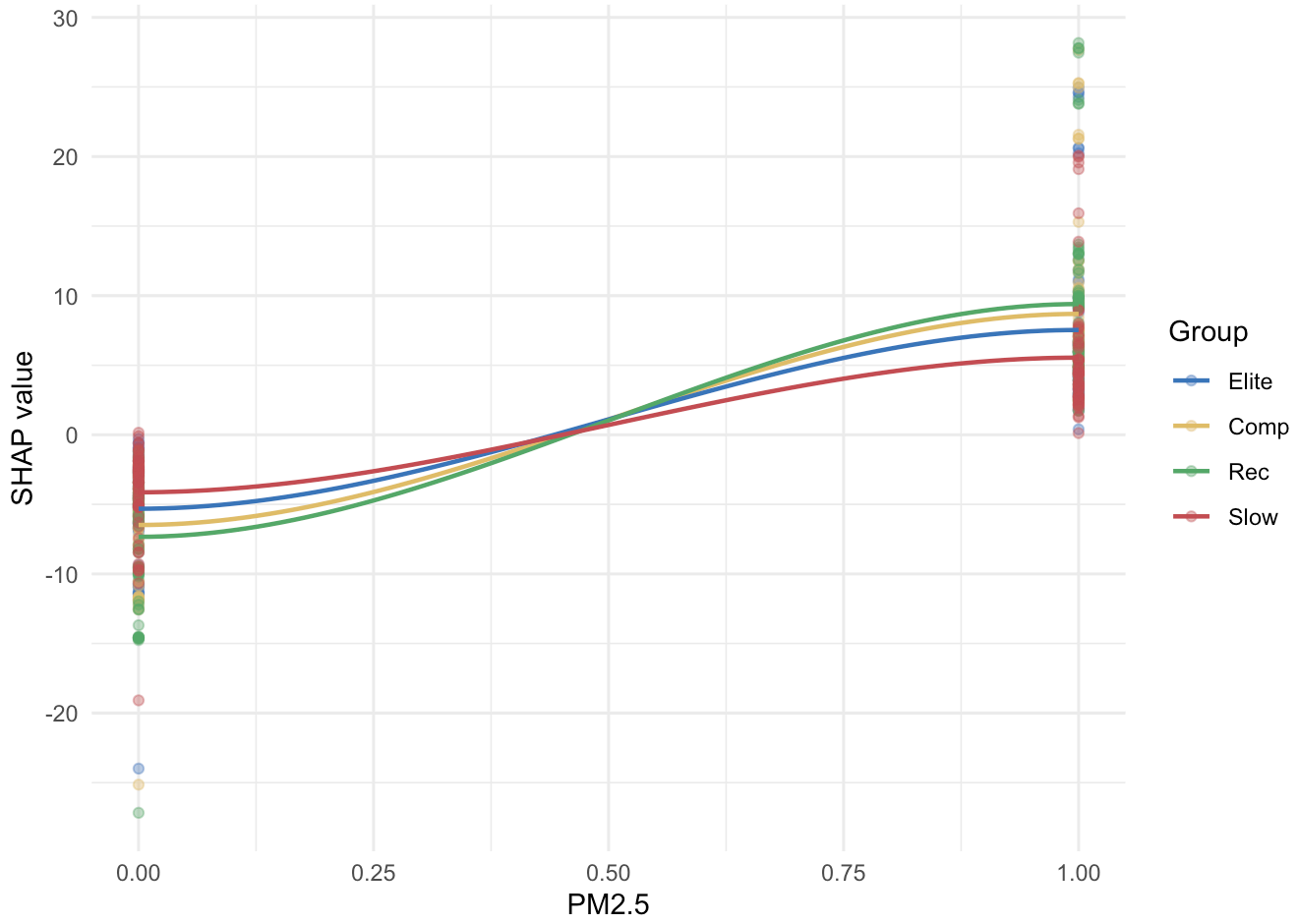
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

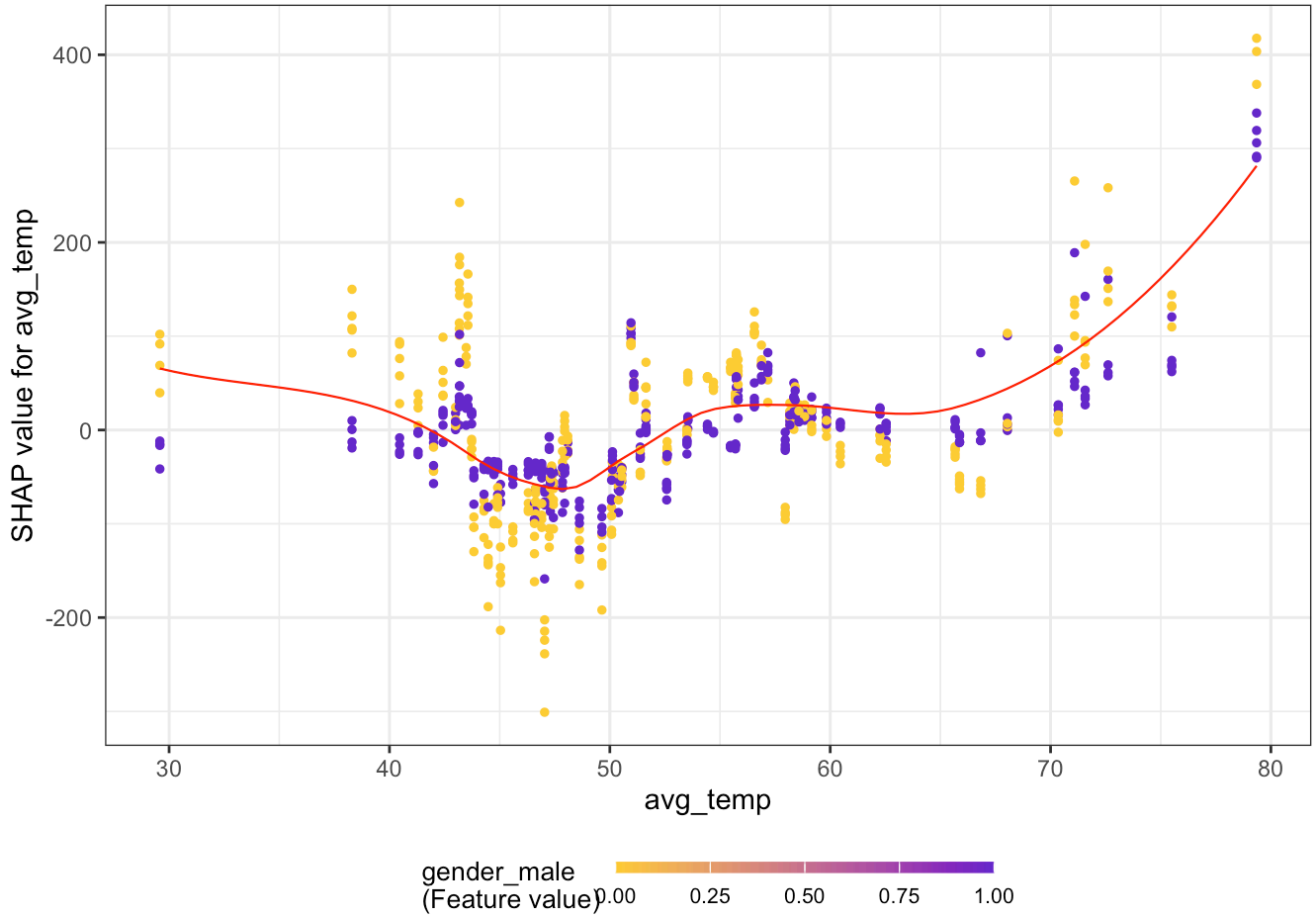


Recreational and Competitive runners are fairly consistent in performance between high and low PM2.5 bins. For slow and elite runners there appears to be a slight improvement between low and high bins which is counter-intuitive.

(ZD/MH)

```
# Temperature effect by gender -
shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "gender_male"
)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```


400
200
0
-200
SHAP value for avg_temp
30
40
50
60
70
80
avg_temp
gender_male
(Feature value)
0.00
0.25
0.50
0.75
1.00

```
#plot for gender:male
gender <- shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "gender_male"
)
#SHAP values broken down by 1 (male) or 0 (female) and relevant color.
gender_points <- gender$data

gender_points$color_value <- factor(
  gender_points$color_value,
  levels = c(0, 1),
  labels = c("Female", "Male")
)



#plot each subgroup and relevant SHAP values
ggplot(gender_points, aes(x = x_feature, y = value, color = color_value)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "loess", se = FALSE, size = 1.2) +
  scale_color_manual(
    values = c("Female" = "#E69F00", "Male" = "#0072B2")
  ) +
  labs(
    x = "Average Temp",
    y = "SHAP value",
    color = "Gender"
  ) +
  theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

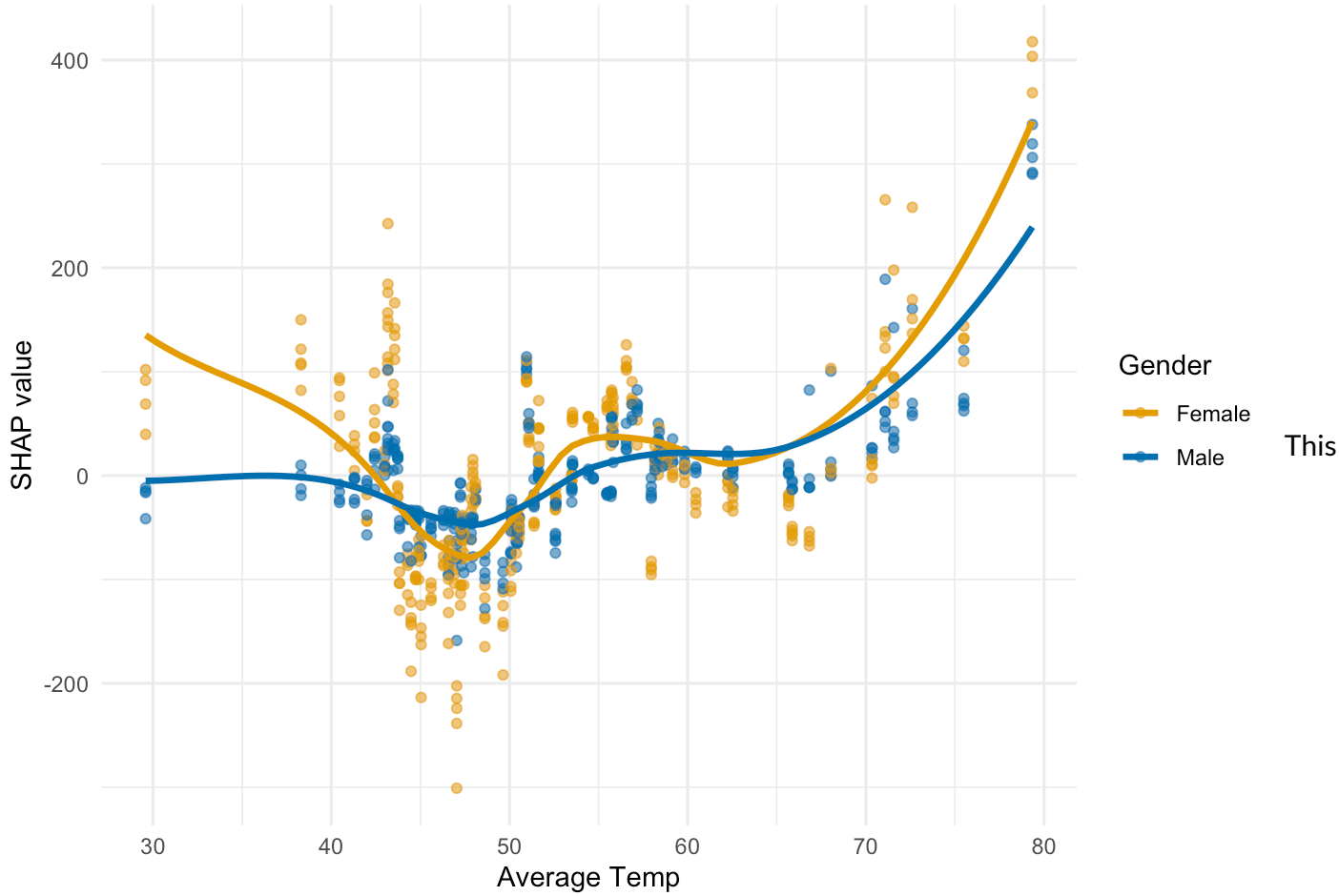


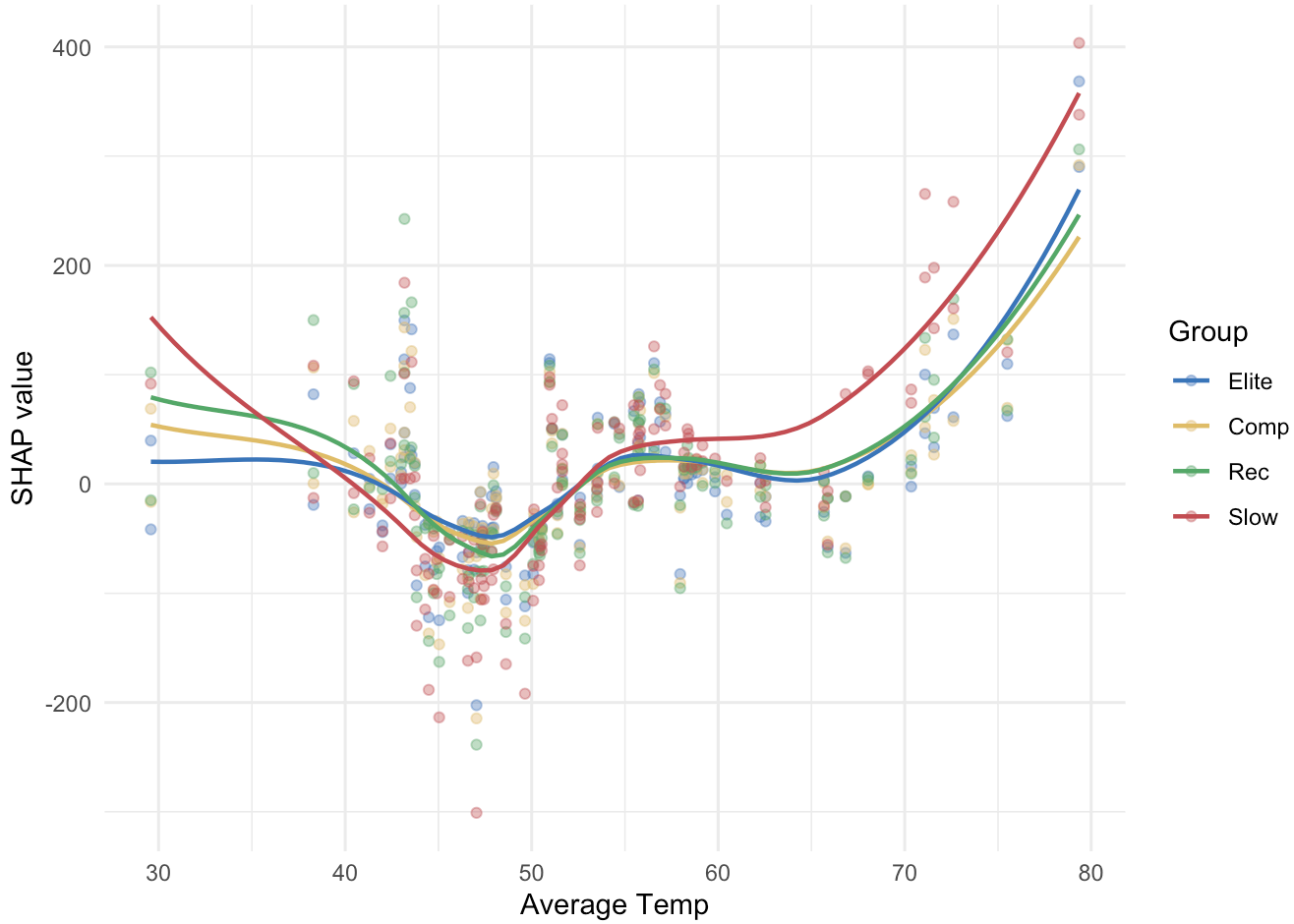
figure shows the comparison of avg temp by gender. Females see greater improvement as temperatures warm up to between 45 and 50 degrees and then begin to slow down, where as men are more consistent until about 65 degrees before they begin to slow down. This suggests that when we're looking at temperature alone. Men are slightly more resistent to performance declines.

(ZD/MH)

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "avg_temp",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "Average Temp", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Temperature displays a much clearer impact than the PM2.5 results. At lower temperatures, the SHAP values are negative and ultimately predict a faster finishing time. At moderate temperatures, the effects are rather neutral. At higher temperatures, the SHAP values begin to rise, indicating a slower predicted finish. While extremely warm temperatures the SHAP values sharply increase the predicted finishing time by a rough average of 400 seconds. Generally speaking, colder temperatures appear to improve race time while warmer

temperatures hinder race time. Slow runners appear to be the least resistent to average temperature increasing, but experience the greatest improvement as temperatures warm up from freezing to about 45 degrees.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "ozone_bin_1",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "ozone_bin_1",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "ozone_bin_1",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "ozone_bin_1",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset has color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "Ozone", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

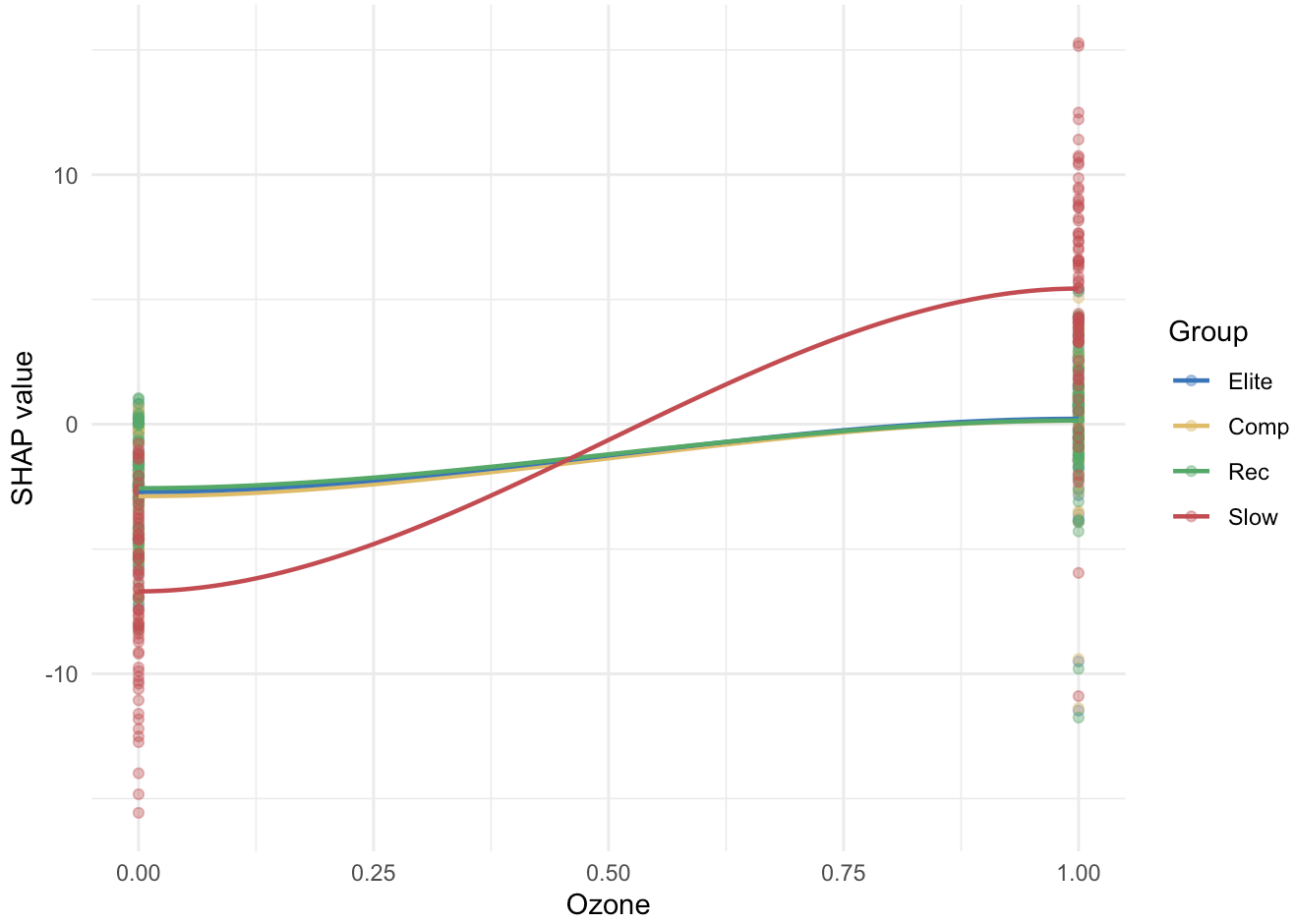
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : pseudoinverse used at -0.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : neighborhood radius 1.005
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : reciprocal condition number 0
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric = parametri
c,
## : There are other near singularities as well. 1.01
```

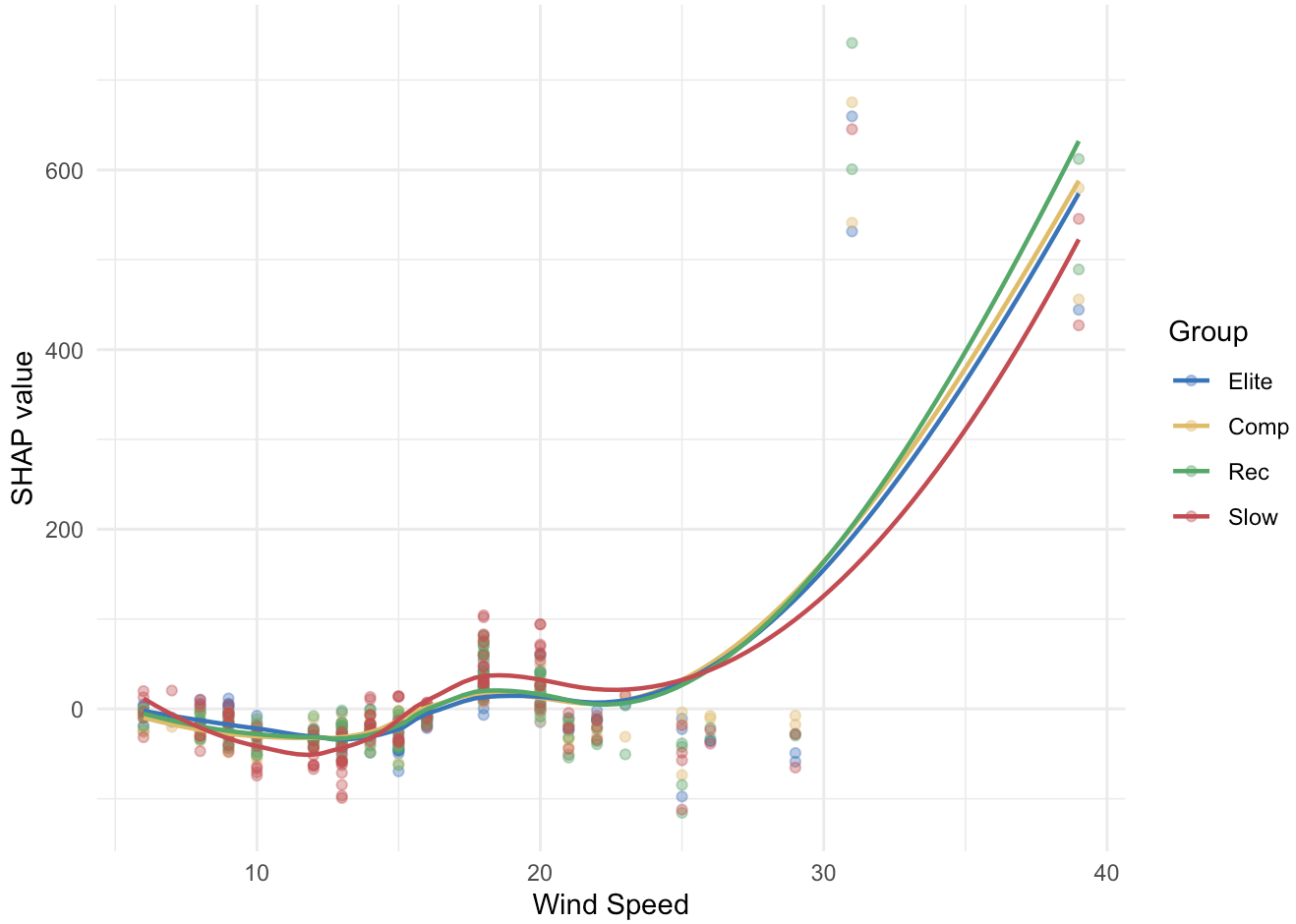


Ozone does not appear to have significant effect on elite, competitive or recreation runners between low ozone bin and high ozone bin. Slow runners however do experience a performance decline with high ozone.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "wind_speed",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "wind_speed",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "wind_speed",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "wind_speed",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "Wind Speed", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

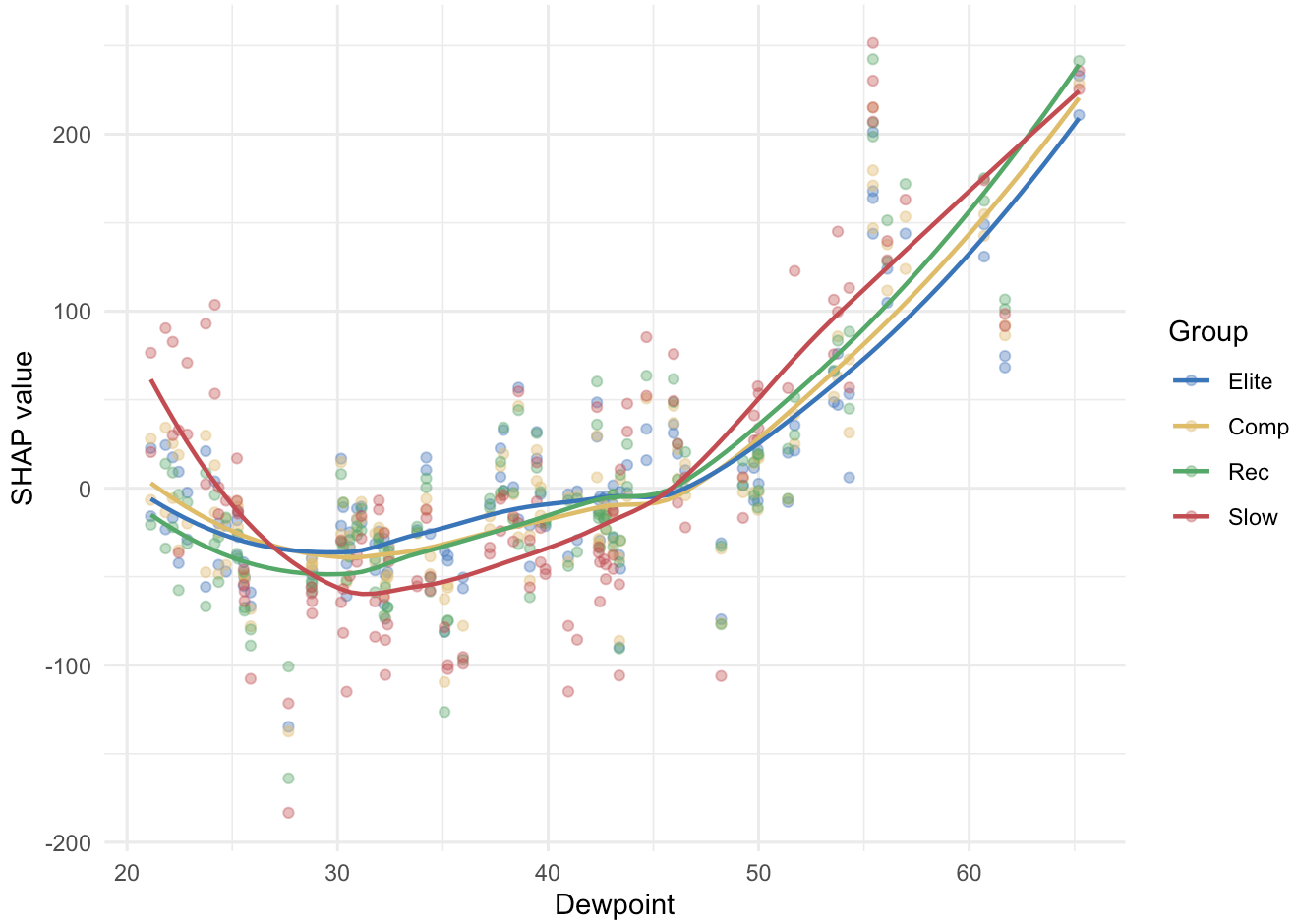


Wind speeds above 30 miles per hour are relatively rare in the dataset, but when they do occur, they're associated with substantial slowdowns across all runner subgroups. Slow runners appear to be the most resilient to increased wind speeds. One possible explanation is that higher running speeds experience greater aerodynamic drag, making wind more impactful for faster runners.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "dew_point",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "dew_point",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "dew_point",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "dew_point",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset has color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "Dewpoint", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

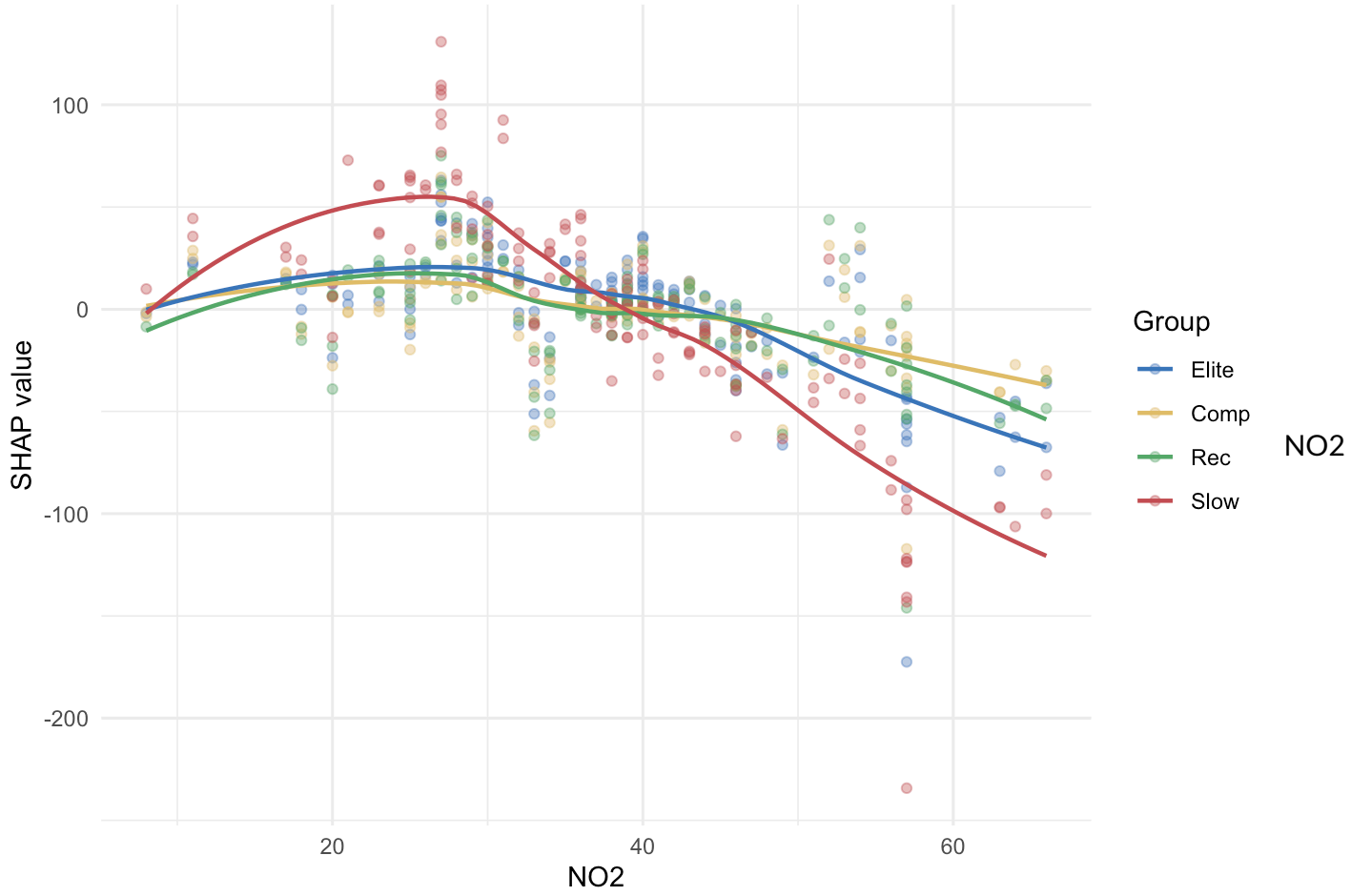


Dewpoint is related to a performance decline across all subgroups. Slow runners appear to be the least resistant, whereas elite runners are the most resistent.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "no2",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "no2",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "no2",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "no2",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "NO2", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

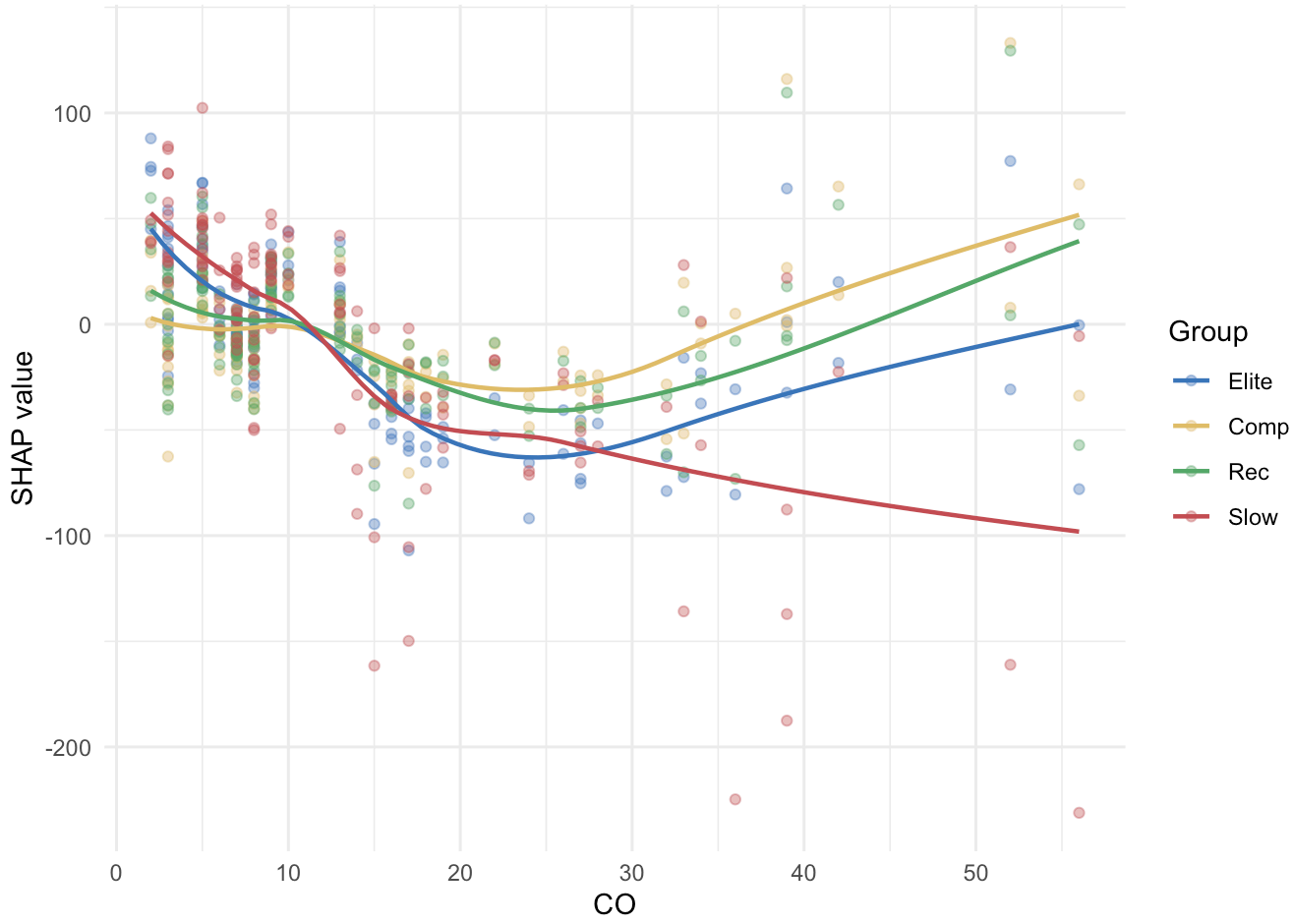


is associated with a performance increase at higher levels based on this model. This is counter intuitive to research suggesting ozone, NO2 and PM2.5 are all related to slower marathons.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "co",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "co",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "co",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "co",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "CO", y = "SHAP value", color = "Group")
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

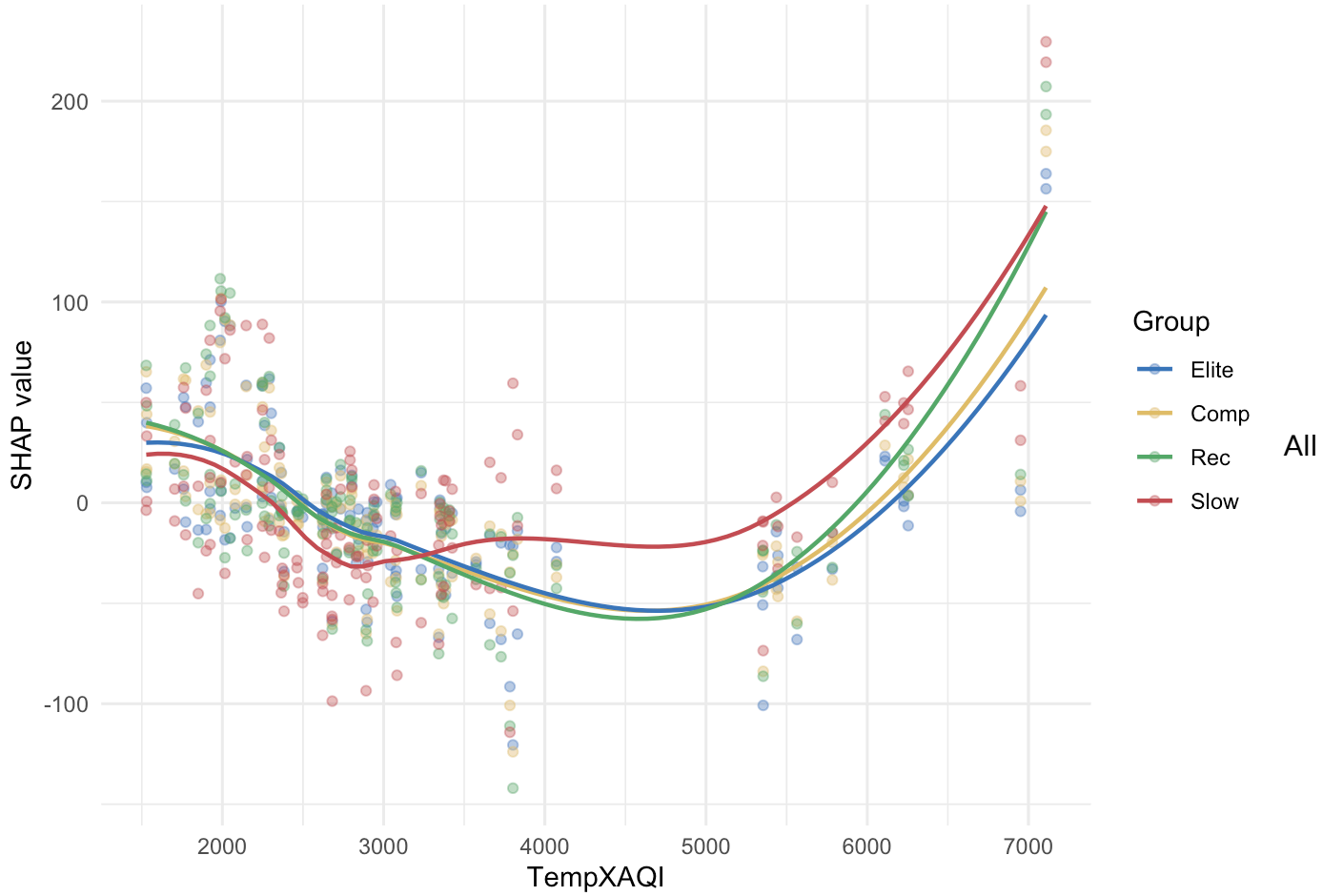


With the exception of slow runners, subgroups experience a slow down as CO increases in the atmosphere. Interestingly, slow runners appear the most resilient.

```
#plot for elite subgroup
elite <- shap.plot.dependence(
  shap_long,
  x = "temp_aqi_interaction",
  color_feature = "subgroup_elite"
)
#SHAP values broken down by 1 (elite) or 0 (not-elite) and relevant color.
elite_points <- elite$data

#repeated for competitive subgroup
comp <- shap.plot.dependence(
  shap_long,
  x = "temp_aqi_interaction",
  color_feature = "subgroup_competitive"
)
comp_points <- comp$data

#repeated for recreational subgroup
rec <- shap.plot.dependence(
  shap_long,
  x = "temp_aqi_interaction",
  color_feature = "subgroup_recreational"
)
rec_points <- rec$data

#repeated for slow subgroup
slow <- shap.plot.dependence(
  shap_long,
  x = "temp_aqi_interaction",
  color_feature = "subgroup_slow"
)
slow_points <- slow$data



#subset so each dataset hase color_value = 1 (the dataframe's positive subroup)
elite_points <- subset(elite_points, color_value == 1)
comp_points <- subset(comp_points, color_value == 1)
rec_points <- subset(rec_points, color_value == 1)
slow_points <- subset(slow_points, color_value == 1)

#add group to each dataframe for merge
elite_points$group <- "Elite"
comp_points$group  <- "Comp"
rec_points$group   <- "Rec"
slow_points$group  <- "Slow"

#merge
```

```
all_points <- rbind(elite_points, comp_points, rec_points, slow_points)

#plot each subgroup and relevant SHAP values
ggplot(all_points, aes(x = x_feature, y = value, color = group)) +
  geom_point(alpha = 0.4) +
  geom_smooth(se = FALSE, size = 0.8) +   # one smooth line per group
  scale_color_manual(values = c(
    "Elite" = "#3B77BC",
    "Comp"  = "#E1BE6A",
    "Rec"   = "#55A868",
    "Slow"  = "#C44E52"
  ),
  breaks = c("Elite", "Comp", "Rec", "Slow")
  ) +
  theme_minimal() +
  labs(x = "TempXAQI", y = "SHAP value", color = "Group")
```

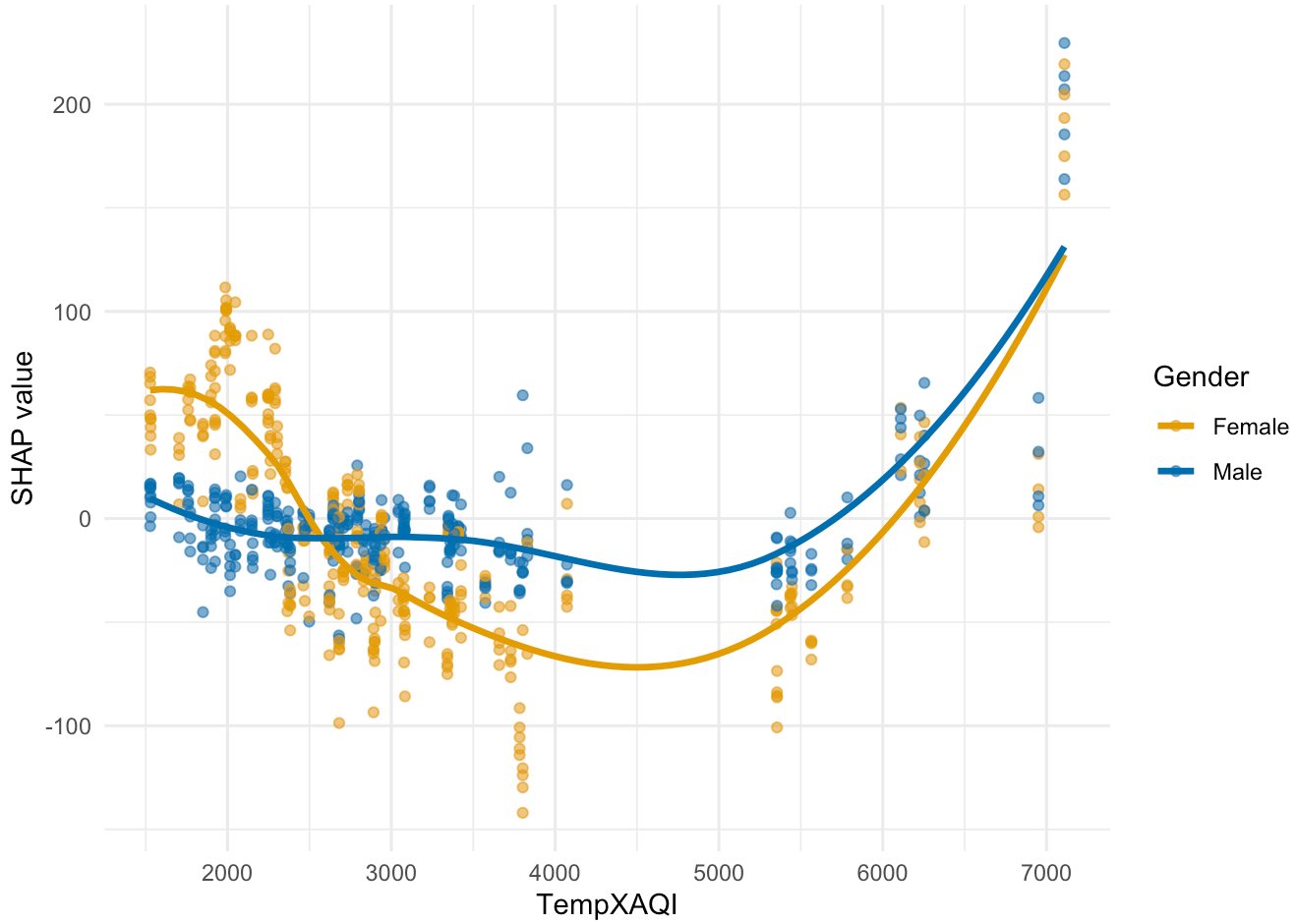
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



subgroups are affected by the interaction to temperature and AQI. Is the interaction term increase, performance decreases. Slow runners experience the greatest slow down due to this interatio term. Elite runners appear the most resilient.

```
#plot for gender:male
gender <- shap.plot.dependence(
  shap_long,
  x = "temp_aqi_interaction",
  color_feature = "gender_male"
)
#SHAP values broken down by 1 (male) or 0 (female) and relevant color.
gender_points <- gender$data

gender_points$color_value <- factor(
  gender_points$color_value,
  levels = c(0, 1),
  labels = c("Female", "Male")
)


#plot each subgroup and relevant SHAP values
ggplot(gender_points, aes(x = x_feature, y = value, color = color_value)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "loess", se = FALSE, size = 1.2) +
  scale_color_manual(
    values = c("Female" = "#E69F00", "Male" = "#0072B2")
  ) +
  labs(
    x = "TempXAQI",
    y = "SHAP value",
    color = "Gender"
  ) +
  theme_minimal()
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



The same interaction term is tested by gender. While the avg temperature showed that male runners experienced less slow downs, when we look at the temperature and AQI interaction term, we see that women are more resilient overall, and actually experience a much greater performance boost at lower interaction terms than male runners.

10 Testing Berlin Data on Best Model (KB, MH)

We found that the best model was the engineered-feature XGBoost model.

10.1 Apply Identical Preprocessing and Feature Engineering to Berlin Data (KB, MH)

Make the Berlin data clean and feature engineered the same way as the main_data for the engineered-feature XGBoost model.

```
# Handling Missing Values (ZD, KB)
missing_summary <- berlin_data %>%
  summarise(across(everything(), ~sum(is.na(.))))

# Keep only columns with at least one missing value
missing_summary <- missing_summary %>%
  select(where(~any(. > 0)))

# Remove PM10 (KB)
berlin_data  <- berlin_data  %>%
  select(-pm10)

# KNN Imputation on PM2.5 using 5 nearest neighbors (KB)
berlin_data <- kNN(berlin_data, variable = "pm25", k = 5) # can change later to s
ee which K gives best model performance
```

```
##                   n                year   avg_chip_seconds           high_temp
##              29.000           1996.000           8912.405              52.000
##            low_temp            avg_temp      precipitation           dew_point
##              40.000              48.560              0.000              40.420
##          wind_speed          visibility sea_level_pressure                 aqi
##               3.000               6.060             29.650              11.000
##                  co               ozone                no2                  n
##               0.000               7.000              7.000           11730.000
##                year   avg_chip_seconds          high_temp           low_temp
##            2019.000           19812.724             82.000              58.000
##            avg_temp      precipitation          dew_point          wind_speed
##              68.900               0.850             57.350              11.000
##          visibility sea_level_pressure                 aqi                  co
##               6.390             30.540             72.000               1.000
##               ozone                no2
##              36.000             22.000
```

```
# remove pm25_imp
cols_to_remove <- c(
  "pm25_imp"
)

berlin_data <- berlin_data[, !(names(berlin_data) %in% cols_to_remove)]

#Convert categorical variables to factors (KB)
berlin_data  <- berlin_data  %>%
  mutate(subgroup = factor(subgroup),
         gender = factor(gender),
         marathon = factor(marathon),
         main_pollutant = factor(main_pollutant))

# Feature engineering (ZD, KB)
berlin_data <- berlin_data %>%
  mutate(
    supershoe = factor(ifelse(year >= 2018, 1, 0), levels = c(0, 1)),
    temp_aqi_interaction       = avg_temp * aqi,
    avg_temp_gender_interaction = avg_temp * as.numeric(gender == "male")
  )

# Removing correlated variables  (KB)
cols_to_remove <- c(
  "high_temp",
  "low_temp",
  "aqi",
  "main_pollutant"
)

berlin_data <- berlin_data[, !(names(berlin_data) %in% cols_to_remove)]

# Add Binned features based off Decision Tree (ozone_bin and pm25_bin) (KB)
berlin_data$ozone_bin <- factor(ifelse(berlin_data$ozone >= 38, 1, 0), levels = c
(0, 1))
berlin_data$pm25_bin  <- factor(ifelse(berlin_data$pm25 >= 54, 1, 0), levels = c
(0, 1))

# Remove original ozone and pm25 from Berlin data so there isn't redundancy (KB)
berlin_data <- berlin_data[, !(names(berlin_data) %in% c("ozone"))]
berlin_data  <- berlin_data[, !(names(berlin_data) %in% c("pm25"))]

str(berlin_data)
```

```
## 'data.frame':    235 obs. of  19 variables:
##  $ n                           : int  131 541 901 402 60 737 3416 5411 3439 894
...
##  $ marathon                    : Factor w/ 1 level "Berlin": 1 1 1 1 1 1 1 1 1
1 ...
##  $ year                        : int  1996 1996 1996 1996 1996 1996 1996 1996 1
996 1996 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 1
2 2 2 2 2 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
2 3 4 5 1 2 3 4 5 ...
##  $ avg_chip_seconds            : num  10641 12733 14707 16821 19054 ...
##  $ avg_temp                    : num  52.3 52.3 52.3 52.3 52.3 ...
##  $ precipitation               : num  0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.23
0.23 0.23 ...
##  $ dew_point                   : num  48.6 48.6 48.6 48.6 48.6 ...
##  $ wind_speed                  : int  10 10 10 10 10 10 10 10 10 10 ...
##  $ visibility                  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ sea_level_pressure          : num  30.1 30.1 30.1 30.1 30.1 ...
##  $ co                          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ no2                         : num  11 11 11 11 11 11 11 11 11 11 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ temp_aqi_interaction        : num  576 576 576 576 576 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 0 ...
##  $ ozone_bin                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ pm25_bin                    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
```

Drop unavailable predictors (KB) The Berlin marathon lacked complete data for some predictors (marathon identifier and CO concentration), these variables were removed from both the training and Berlin datasets to ensure consistent feature spaces and valid outo f sample prediction.

```
bad_cols <- c("marathon", "co")

train_data_clean <- train_data %>%
  select(-all_of(bad_cols))

berlin_data_clean <- berlin_data %>%
  select(-all_of(bad_cols))

str(berlin_data_clean)
```

```
## 'data.frame':    235 obs. of  17 variables:
##  $ n                           : int  131 541 901 402 60 737 3416 5411 3439 894
...
##  $ year                        : int  1996 1996 1996 1996 1996 1996 1996 1996 1
996 1996 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 1
2 2 2 2 2 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
2 3 4 5 1 2 3 4 5 ...
##  $ avg_chip_seconds            : num  10641 12733 14707 16821 19054 ...
##  $ avg_temp                    : num  52.3 52.3 52.3 52.3 52.3 ...
##  $ precipitation               : num  0.23 0.23 0.23 0.23 0.23 0.23 0.23 0.23
0.23 0.23 ...
##  $ dew_point                   : num  48.6 48.6 48.6 48.6 48.6 ...
##  $ wind_speed                  : int  10 10 10 10 10 10 10 10 10 10 ...
##  $ visibility                  : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ sea_level_pressure          : num  30.1 30.1 30.1 30.1 30.1 ...
##  $ no2                         : num  11 11 11 11 11 11 11 11 11 11 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ temp_aqi_interaction        : num  576 576 576 576 576 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 0 ...
##  $ ozone_bin                   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
##  $ pm25_bin                    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1
1 ...
```

```
str(train_data_clean)
```

```
## 'data.frame':    693 obs. of  17 variables:
##  $ n                           : int  77 62 4657 1318 4199 269 5155 1603 4888 3
163 ...
##  $ year                        : int  2002 2005 2011 2003 2019 2010 2009 2000 2
008 2003 ...
##  $ gender                      : Factor w/ 2 levels "female","male": 1 1 1 1 2
2 1 2 2 1 ...
##  $ subgroup                    : Factor w/ 5 levels "elite","competitive",..: 1
1 3 4 3 1 4 4 3 3 ...
##  $ avg_chip_seconds            : num  10485 10318 14201 16716 12785 ...
##  $ avg_temp                    : num  41.3 53.5 52.6 47.3 55.8 ...
##  $ precipitation               : num  0 0 0 0 0.61 0 0 0 0 0 ...
##  $ dew_point                   : num  21.1 42.5 27.7 37.7 49.3 ...
##  $ wind_speed                  : int  13 18 25 25 26 18 9 21 13 16 ...
##  $ visibility                  : num  10 10 10 10 10 10 10 10 10 10 ...
##  $ sea_level_pressure          : num  30 29.4 30 30.1 29.6 ...
##  $ no2                         : num  44 38 20 43 35 29 30 30 36 45 ...
##  $ supershoe                   : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 1 1 1
1 ...
##  $ temp_aqi_interaction        : num  2148 2622 2787 3358 2624 ...
##  $ avg_temp_gender_interaction: num  0 0 0 0 55.8 ...
##  $ ozone_bin                   : Factor w/ 2 levels "0","1": 1 1 2 2 2 2 1 2 2
1 ...
##  $ pm25_bin                    : Factor w/ 2 levels "0","1": 1 1 1 2 1 1 1 2 2
2 ...
```

Rebuild the training model matrix and retrain model (KB)

Factor levels for categorical predictors were explicitly aligned between the training and Berlin datasets to ensure consistent feature encoding and valid out-of-sample prediction.

NOTE: Using train_matrix_clean as a template.

```
# Rebuild training model matrix
#train_matrix_clean <- model.matrix(
#  avg_chip_seconds ~ .,
#  data = train_data_clean %>% select(-year, -n)
#)[, -1]


train_matrix_clean <- train_data_clean %>%
  select(-year, -n, -avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("subgroup","gender","supershoe","ozone_bin","pm25_bin"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
  mutate(across(everything(), ~ as.numeric(.))) %>%  # convert all columns to num
eric
  as.matrix()


# Ensure colnames exist
stopifnot(!is.null(colnames(train_matrix_clean)))

# Create DMatrix
dtrain_clean <- xgb.DMatrix(
  data = train_matrix_clean,
  label = train_data_clean$avg_chip_seconds
)

# Set seed for reproducibility
set.seed(123)

# Train model
xgb_model_berlin <- xgb.train(
  params = best_params, # from previous feature engineered XGBoost model
  data = dtrain_clean,
  nrounds = best_nrounds,
  verbose = 0
)
```

Align all factor variables in Berlin that were using in the train data so that the XGBoost model can process the data. (KB)

```
# Align all categorical predictors
berlin_data_clean$gender <- factor(
  berlin_data_clean$gender,
  levels = levels(train_data_clean$gender)
)

berlin_data_clean$subgroup <- factor(
  berlin_data_clean$subgroup,
  levels = levels(train_data_clean$subgroup)
)

berlin_data_clean$supershoe <- factor(
  berlin_data_clean$supershoe,
  levels = levels(train_data_clean$supershoe)
)

berlin_data_clean$ozone_bin <- factor(
  berlin_data_clean$ozone_bin,
  levels = levels(train_data_clean$ozone_bin)
)

berlin_data_clean$pm25_bin <- factor(
  berlin_data_clean$pm25_bin,
  levels = levels(train_data_clean$pm25_bin)
)
```

10.2 Build and Align the Berlin Feature Matrix (KB)

Filter Berlin data to complete cases (KB)

```
berlin_mm <- model.matrix(
  avg_chip_seconds ~ .,
  data = berlin_data_clean %>% select(-year, -n)
)

# Identify rows used by model.matrix
rows_used <- attr(berlin_mm, "assign") != 0  # not reliable

# Restrict Berlin data to complete cases
berlin_cc <- berlin_data_clean %>%
  select(-year, -n) %>%
  filter(complete.cases(.))
```

Build Berlin model matrix (KB)

```
#berlin_matrix <- model.matrix(
#  avg_chip_seconds ~ .,
#  data = berlin_cc
#)[, -1]


berlin_matrix_clean <- berlin_cc %>%
  select(-avg_chip_seconds) %>%
  fastDummies::dummy_cols(
    select_columns = c("subgroup","gender","supershoe","ozone_bin","pm25_bin"),
    remove_first_dummy = FALSE,
    remove_selected_columns = TRUE
  ) %>%
  mutate(across(everything(), as.numeric)) %>%
  as.matrix()
```

Align Berlin to training feature space (KB)

```
# Align Berlin columns to the training matrix
berlin_matrix_clean <- berlin_matrix_clean[, colnames(train_matrix_clean), drop =
FALSE]

# Sanity check
stopifnot(identical(colnames(berlin_matrix_clean), colnames(train_matrix_clean)))
```

Predict and evaluate on Berlin data (KB)

```
dberlin <- xgb.DMatrix(data = berlin_matrix_clean)
pred_berlin <- predict(xgb_model_berlin, dberlin)

# Only use rows actually in berlin_matrix
actual_berlin <- berlin_cc$avg_chip_seconds
stopifnot(length(actual_berlin) == length(pred_berlin))

rmse_berlin <- rmse(actual_berlin, pred_berlin)
mae_berlin  <- mae(actual_berlin, pred_berlin)

rss_berlin <- sum((actual_berlin - pred_berlin)^2)
tss_berlin <- sum((actual_berlin - mean(actual_berlin))^2)
r2_berlin  <- 1 - rss_berlin / tss_berlin

# Results table
berlin_eval <- data.frame(
  Metric = c("RMSE", "MAE", "R²"),
  Value  = c(rmse_berlin, mae_berlin, r2_berlin)
)

knitr::kable(
  berlin_eval,
  digits = 4,
  caption = "XGBoost Engineered-Feature Model Performance on Berlin Test Data"
)
```

XGBoost Engineered-Feature Model Performance on Berlin Test Data

Metric	Value
RMSE	889.6358
MAE	694.0109
R^2	0.9196

```
# Show how many rows were dropped
cat("Total Berlin rows:", nrow(berlin_data_clean), "\n")
```

```
## Total Berlin rows: 235
```

```
cat("Rows used for prediction (complete cases):", nrow(berlin_cc), "\n")
```

```
## Rows used for prediction (complete cases): 60
```

```
cat("Rows dropped due to missing predictors:", nrow(berlin_data_clean) - nrow(ber
lin_cc), "\n")
```

```
## Rows dropped due to missing predictors: 175
```

We can see that the results indicate that the XGBoost engineered-feature model predicts average chip times for Berlin marathon runners with high accuracy on the subset of data with complete predictor information. The R2 of 0.9174 shows that over 91% of the variance in actual chip times is explained by the model.

Out of 235 total Berlin observations, only 60 rows (or around 26%) were actually used for prediction. 175 rows (or around 74%) were dropped due to missing values in predictors. This is a substantial portion of the dataset, which means the evaluation metrics reflect model performance only on the complete-case subset.

Note: The reported performance may not fully represent how the model would perform on the entire Berlin dataset, so caution should be used when generalizing these results.

Summary:

In preparing the Berlin marathon data for out of sample prediction with the XGBoost model, we first removed unavailable predictors (marathon identifier and CO concentration) from both the training and Berlin datasets to ensure a consistent feature space. The training model matrix was rebuilt and the XGBoost model retrained using only the remaining predictors.

Categorical predictors in the Berlin dataset were explicitly aligned to match the levels in the training data, ensuring correct encoding. Since model.matrix() automatically drops rows with missing predictor values, we restricted the Berlin dataset to complete cases before building the design matrix and making predictions. The Berlin feature matrix was then aligned to the training feature space to prevent feature mismatches. Predictions were made only on the complete case rows, and model performance was evaluated using RMSE, MAE, and R2.

It is important to note that we also tracked how many Berlin observations were dropped due to missing predictors to maintain transparency.

---

11 Inactive / Archived Code

`Contains code ideas that were not used.`

indicate and define year of covid (ZD) final_data <- final_data %>% mutate(covid_era = ifelse(year == 2020, 1, 0))

create custom cutoff times for performance groups (ZD) final_data <- final_data %>% mutate(finish_hours = avg_chip_seconds / 3600, # think we are having all time in seconds, need fixing subgroup2 = case_when(finish_hours < 3 ~ “elite”, finish_hours < 3.75 ~ “competitive”, finish_hours < 4.75 ~ “average”, finish_hours < 5.75 ~ “recreational”, TRUE ~ “slow”))

#The following chunk was used to understand the breakdown between marathons, but was not discussed or actively used for data cleaning or model generation. (MH)

```
#Boston boston <- final_data %>% filter(marathon==“Boston”)

#Summarize summary_long <- boston %>% group_by(year, gender) %>% summarise( N=n(),
mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE),
min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE),
.groups=“drop”) %>% mutate(mean_sd= sprintf(“%.1f (%.1f)”, mean_time, sd_time), min_max= sprintf(“(%.1f,
%.1f)”, min_time, max_time))

#Pivot wider — ensure one row per year summary_wide <- summary_long %>% pivot_wider(id_cols=year,
names_from=gender, values_from=c(N, mean_sd, min_max), names_glue=“{gender}_{.value}“) %>%
arrange(year)

#Combine into one column per gender for GT summary_combined <- summary_wide %>%
mutate(Female=paste0(“N =”, female_N, “
”, female_mean_sd, “
”, female_min_max), Male=paste0(“N =”, male_N, “
”, male_mean_sd, “
”, male_min_max)) %>% select(year, Female, Male)

#GT table summary_combined %>% gt(rowname_col = “year”) %>% cols_label(Female = “Female”, Male =
“Male”) %>% fmt_markdown(columns = c(Female, Male)) %>% tab_options( data_row.padding = px(3),
table.font.size = px(14))

#Berlin berlin<- final_data %>% filter(marathon==“Berlin”) #Summarize summary_long <- berlin %>%
group_by(year, gender) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE),
sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE),
max_time=max(avg_chip_seconds, na.rm = TRUE), .groups=“drop”) %>% mutate(mean_sd= sprintf(“%.1f
(%.1f)”, mean_time, sd_time), min_max= sprintf(“(%.1f, %.1f)”, min_time, max_time))

#Pivot wider — ensure one row per year summary_wide <- summary_long %>% pivot_wider(id_cols=year,
names_from=gender, values_from=c(N, mean_sd, min_max), names_glue=“{gender}_{.value}“) %>%
arrange(year)

#Combine into one column per gender for GT summary_combined <- summary_wide %>%
mutate(Female=paste0(“N =”, female_N, “
”, female_mean_sd, “
”, female_min_max), Male=paste0(“N =”, male_N, “
”, male_mean_sd, “
”, male_min_max)) %>% select(year, Female, Male)

#GT table summary_combined %>% gt(rowname_col = “year”) %>% cols_label(Female = “Female”, Male =
“Male”) %>% fmt_markdown(columns = c(Female, Male)) %>% tab_options( data_row.padding = px(3),
table.font.size = px(14))

#Chicago chicago<- final_data %>% filter(marathon==“Chicago”) #Summarize summary_long <- chicago %>%
group_by(year, gender) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE),
sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE),
max_time=max(avg_chip_seconds, na.rm = TRUE), .groups=“drop”) %>% mutate(mean_sd= sprintf(“%.1f
(%.1f)”, mean_time, sd_time), min_max= sprintf(“(%.1f, %.1f)”, min_time, max_time))
```

```
#Pivot wider — ensure one row per year summary_wide <- summary_long %>% pivot_wider(id_cols=year, names_from=gender, values_from=c(N, mean_sd, min_max), names_glue="{gender}_{.value}") %>% arrange(year)

#Combine into one column per gender for GT summary_combined <- summary_wide %>% mutate(Female=paste0("N =", female_N, "
", female_mean_sd, "
", female_min_max), Male=paste0("N =", male_N, "
", male_mean_sd, "
", male_min_max)) %>% select(year, Female, Male)

#GT table summary_combined %>% gt(rowname_col = "year") %>% cols_label(Female = "Female", Male = "Male") %>% fmt_markdown(columns = c(Female, Male)) %>% tab_options( data_row.padding = px(3), table.font.size = px(14))

#NYC NYC<- final_data %>% filter(marathon=="NYC") #Summarize summary_long <- NYC %>% group_by(year, gender) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE), .groups="drop") %>% mutate(mean_sd= sprintf("%.1f (%.1f)", mean_time, sd_time), min_max= sprintf("(%.1f, %.1f)", min_time, max_time))

#Pivot wider — ensure one row per year summary_wide <- summary_long %>% pivot_wider(id_cols=year, names_from=gender, values_from=c(N, mean_sd, min_max), names_glue="{gender}_{.value}") %>% arrange(year)

#Combine into one column per gender for GT summary_combined <- summary_wide %>% mutate(Female=paste0("N =", female_N, "
", female_mean_sd, "
", female_min_max), Male=paste0("N =", male_N, "
", male_mean_sd, "
", male_min_max)) %>% select(year, Female, Male)

#GT table summary_combined %>% gt(rowname_col = "year") %>% cols_label(Female = "Female", Male = "Male") %>% fmt_markdown(columns = c(Female, Male)) %>% tab_options( data_row.padding = px(3), table.font.size = px(14))

#All runners

#Boston boston<- final_data %>% filter(marathon=="Boston") summary_all <- boston %>% group_by(year) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE), .groups="drop") %>% mutate( combined=paste0( "N =", N, "
", sprintf("%.1f (%.1f)", mean_time, sd_time), "
", sprintf("(%.1f, %.1f)", min_time, max_time))) %>% select(year, combined) summary_all %>% gt(rowname_col = "year") %>% cols_label(combined="All Runners") %>% fmt_markdown(columns="combined") %>% tab_options(data_row.padding = px(3),table.font.size = px(14))

#Berlin berlin<- final_data %>% filter(marathon=="Berlin") summary_all <- berlin %>% group_by(year) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE),
```

```
.groups="drop") %>% mutate( combined=paste0( "N =", N, "
", sprintf("%.1f (%.1f)", mean_time, sd_time), "
", sprintf("(%.1f, %.1f)", min_time, max_time))) %>% select(year, combined) summary_all %>% gt(rowname_col = "year") %>% cols_label(combined="All Runners") %>% fmt_markdown(columns="combined") %>% tab_options(data_row.padding = px(3),table.font.size = px(14))

#Chicago chicago<- final_data %>% filter(marathon=="Chicago") summary_all <- chicago %>% group_by(year) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE), .groups="drop") %>% mutate( combined=paste0( "N =", N, "
", sprintf("%.1f (%.1f)", mean_time, sd_time), "
", sprintf("(%.1f, %.1f)", min_time, max_time))) %>% select(year, combined) summary_all %>% gt(rowname_col = "year") %>% cols_label(combined="All Runners") %>% fmt_markdown(columns="combined") %>% tab_options(data_row.padding = px(3),table.font.size = px(14))

#NYC NYC<- final_data %>% filter(marathon=="NYC") summary_all <- NYC %>% group_by(year) %>% summarise( N=n(), mean_time=mean(avg_chip_seconds, na.rm = TRUE), sd_time=sd(avg_chip_seconds, na.rm = TRUE), min_time=min(avg_chip_seconds, na.rm = TRUE), max_time=max(avg_chip_seconds, na.rm = TRUE), .groups="drop") %>% mutate( combined=paste0( "N =", N, "
", sprintf("%.1f (%.1f)", mean_time, sd_time), "
", sprintf("(%.1f, %.1f)", min_time, max_time))) %>% select(year, combined) summary_all %>% gt(rowname_col = "year") %>% cols_label(combined="All Runners") %>% fmt_markdown(columns="combined") %>% tab_options(data_row.padding = px(3),table.font.size = px(14))

#BOX AND WHISKER #BOSTON ggplot(boston, aes(x = gender, y =avg_chip_seconds/ 3600, fill = gender)) + geom_boxplot(outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Boston Marathon Chip Times by Gender") + theme_minimal() + theme(legend.position = "none")

ggplot(boston, aes(x = "", y =avg_chip_seconds / 3600)) + geom_boxplot(fill ="steelblue", outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Boston Marathon Chip Times - All Runners") + theme_minimal() + theme(axis.title.x = element_blank(), axis.text.x = element_blank(), axis.ticks.x = element_blank())

#BERLIN ggplot(berlin, aes(x = gender, y =avg_chip_seconds/ 3600, fill = gender)) + geom_boxplot(outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Berlin Marathon Chip Times by Gender") + theme_minimal() + theme(legend.position = "none")

ggplot(berlin, aes(x = "", y =avg_chip_seconds / 3600)) + geom_boxplot(fill ="steelblue", outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Berlin Marathon Chip Times - All Runners") + theme_minimal() + theme(axis.title.x = element_blank(), axis.text.x = element_blank(), axis.ticks.x = element_blank())

#CHICAGO ggplot(chicago, aes(x = gender, y =avg_chip_seconds/ 3600, fill = gender)) + geom_boxplot(outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Chicago Marathon Chip Times by Gender") + theme_minimal() + theme(legend.position = "none")

ggplot(chicago, aes(x = "", y =avg_chip_seconds / 3600)) + geom_boxplot(fill ="steelblue", outlier.alpha = 0.3) + scale_y_continuous(name = "Chip Time") + labs(title = "Chicago Marathon Chip Times - All Runners") + theme_minimal() + theme(axis.title.x = element_blank(), axis.text.x = element_blank(), axis.ticks.x =
```

element_blank())

#NYC ggplot(NYC, aes(x = gender, y =avg_chip_seconds/ 3600, fill = gender)) + geom_boxplot(outlier.alpha = 0.3) + scale_y_continuous(name = “Chip Time”) + labs(title = “NYC Marathon Chip Times by Gender”) + theme_minimal() + theme(legend.position = “none”)

ggplot(NYC, aes(x = ““, y =avg_chip_seconds / 3600)) + geom_boxplot(fill =”steelblue”, outlier.alpha = 0.3) + scale_y_continuous(name = “Chip Time”) + labs(title = “NYC Marathon Chip Times - All Runners”) + theme_minimal() + theme(axis.title.x = element_blank(), axis.text.x = element_blank(), axis.ticks.x = element_blank())

#packageVersion(“xgboost”) #[1] ‘1.7.10.1’