```r
# Title: Mercury Case Study --------------------------------
library(readxl)
library(janitor)
library(tidyverse)
library(lubridate)
library(dplyr)
library(zoo)
library(readxl)
library(reshape2)
library(corrplot)
library(broom)

#load Data
file_path <- "mercury_case_study_dataset.xlsx"

if (!file.exists(file_path)) {
  cat("File not found! Please select the correct file.\n")
  file_path <- file.choose()  # Opens file explorer for user selection
}

sheets <- excel_sheets(path = file_path)
# Load sheet names
excel_sheets(file_path)
pointguard_data <- read_excel(file_path, sheet = 1)


# Read data
catapult <- read_excel(file_path, sheet = "Catapult & Wellness")
cmj <- read_excel(file_path, sheet = "CMJ data")
calf_raise <- read_excel(file_path, sheet = "SL Standing Isometric Calf Rais")
sl_cmj <- read_excel(file_path, sheet = "Single leg CMJ")


## Data Cleaning ----------------------------

# Remove empty rows/columns
catapult <- catapult %>% remove_empty("cols") %>% remove_empty("rows")
cmj <- cmj %>% remove_empty("cols") %>% remove_empty("rows")
calf_raise <- calf_raise %>% remove_empty("cols") %>% remove_empty("rows")
sl_cmj <- sl_cmj %>% remove_empty("cols") %>% remove_empty("rows")


# Handle missing values, wellness imputed with forward fill
## unsure if she played in All-Star game
### skip and leave as NA for all star break ---------
# Define and assume All-Star Break Dates: Assume (July 18 - July 26, 2024)
all_star_break <- seq(as.Date("2024-07-18"), as.Date("2024-07-26"), by="days")

# Define wellness variables to forward fill
wellness_vars <- c("Achilles Soreness", "General Soreness", "Sleep Quantity (hr)", "Sleep
Quality (10 best)")

# Apply forward fill only for wellness columns, but NOT during All-Star Break
catapult <- catapult %>%
  arrange(Date) %>%  # Ensure data is sorted
  mutate(across(all_of(wellness_vars),
                ~ ifelse(Date %in% all_star_break, .x, na.locf(.x, na.rm = FALSE)))) %>%
  mutate(across(all_of(wellness_vars),
                ~ na.locf(.x, fromLast = TRUE, na.rm = FALSE)))  # Backward fill for
leading NAs

# Ensure performance metrics remain NA for All-Star, unsure if tracked remote
# and continued to train
performance_vars <- c("Load", "Distance", "intensity_pl_min", "Jumps",
```

```
"explosive_efforts")

catapult[performance_vars] <- lapply(catapult[performance_vars],
                                      function(x) ifelse(catapult$Date %in%
all_star_break, NA, x))
# Read data and clean
catapult <- catapult %>% clean_names()
cmj <- cmj %>% clean_names()
calf_raise <- calf_raise %>% clean_names()
sl_cmj <- sl_cmj %>% clean_names()


### Convert dates, decimals and on_court to minutes ----
catapult$date <- as.Date(catapult$date)
cmj$date <- as.Date(cmj$date)
calf_raise$date <- as.Date(calf_raise$date)
sl_cmj$date <- as.Date(sl_cmj$date)

# convert decimals in my load per distance
catapult <- catapult %>%
  mutate(load_per_distance = round(load / distance, 1))

#convert catapult time on court to minutes
catapult <- catapult %>%
  mutate(on_court_time = ifelse(!is.na(on_court_time),
                                hour(on_court_time) * 60 + minute(on_court_time),
                                NA))  # Convert to minutes while keeping NAs

# Form an only catapult, sleep and wellness dataframe

catapult_wellness_sleep <- catapult %>%
  select(date, on_court_time, intensity_pl_min, load_per_distance, achilles_soreness,
general_soreness, sleep_quantity_hr, sleep_quality_10_best,
         load, distance, jumps, accelerations, decelerations)  # Add relevant columns
# drop NA's
catapult_wellness_sleep <- catapult_wellness_sleep %>% drop_na()

# convert to just on court minutes
catapult_wellness_sleep <- catapult_wellness_sleep %>%
  mutate(on_court_time = as.POSIXct(on_court_time, origin = "1899-12-30", tz = "UTC")) %>%
# Convert to proper date-time
  mutate(on_court_time = hour(on_court_time) * 60 + minute(on_court_time))  # Extract
total minutes

# merge two bodyweight kg columns, remove old and make just one
cmj <- cmj %>%
  mutate(bodyweight_kg = ifelse(!is.na(athlete_standing_weight_kg),
                                athlete_standing_weight_kg,
                                bodyweight_in_kilograms_kg)) %>%
  select(-bodyweight_in_kilograms_kg, -athlete_standing_weight_kg)

# Ensure date column is formatted correctly
cmj <- cmj %>% arrange(date)
cmj$date <- as.Date(cmj$date, format = "%Y-%m-%d")
# make sure bodyweight is numeric
cmj$bodyweight_kg <- as.numeric(cmj$bodyweight_kg)


## Bodyweight z scores detected that may be input error with losing
# 15 pounds in 3 days etc. Estimated weight based on nearby values in order to
# still use metrics

# z scores for outlier detecttion
#Detect outliers using Z-scores after data is cleaned
detect_outliers_with_zscores <- function(data, df_name, threshold = 1.5) {
```

```
  # Select numeric columns only
  numeric_data <- data %>% select(where(is.numeric))

  # Ensure dataset has numeric columns before proceeding
  if (ncol(numeric_data) == 0) {
    stop(paste0("Error: No numeric columns found in dataset: ", df_name))
  }

  # Compute mean and standard deviation BEFORE filtering or modifying dataset
  mean_values <- colMeans(numeric_data, na.rm = TRUE)
  sd_values <- apply(numeric_data, 2, sd, na.rm = TRUE)

  # Remove columns with near-zero standard deviation
  valid_columns <- sd_values > 1e-5
  numeric_data <- numeric_data[, valid_columns, drop = FALSE]
  mean_values <- mean_values[valid_columns]
  sd_values <- sd_values[valid_columns]

  # Compute Z-scores, keeping original row count
  z_scores <- as.data.frame(scale(numeric_data, center = mean_values, scale = sd_values))

  # Restore original row count (keeping NA values where applicable)
  z_scores <- data.frame(Date = data$date, z_scores)

  # Identify outliers where abs(Z-score) > threshold
  outlier_mask <- abs(z_scores[,-1]) > threshold

  # Convert Z-scores to data frame and keep only outliers
  outlier_values <- z_scores
  outlier_values[,-1] <- outlier_values[,-1] * outlier_mask  # Keep only flagged values

  # Convert to long format
  outlier_long <- melt(outlier_values, id.vars = "Date", variable.name = "Metric",
value.name = "Z_Score")

  # Ensure no filtering removes valid outliers
  outlier_long <- outlier_long %>% filter(!is.na(Z_Score) & Z_Score != 0)

  # If no outliers found, return an empty DataFrame with correct structure
  if (nrow(outlier_long) == 0) {
    print(paste0("No outliers detected for: ", df_name))
    return(data.frame(Date = as.Date(character()), Metric = character(), Z_Score =
numeric(), Dataset = character()))
  }

  # Attach dataset name
  outlier_long$Dataset <- df_name

  return(outlier_long)
}

# Apply the function to each dataset
catapult_outliers <- detect_outliers_with_zscores(catapult, "Catapult & Wellness")
cmj_outliers <- detect_outliers_with_zscores(cmj, "CMJ")
calf_outliers <- detect_outliers_with_zscores(calf_raise, "Calf Raise")

# Combine outlier reports
all_outliers <- bind_rows(catapult_outliers, cmj_outliers, calf_outliers) %>%
  mutate(Date = as.Date(Date))  # Ensure Date remains in Date format

# Save the Z-score outlier report
write.csv(all_outliers, file.path("C:/Users/prist/OneDrive/Desktop/mercury",
"zscore_outlier_report.csv"), row.names = FALSE)
```

```
# Ensure date is in Date format
cmj <- cmj %>%
  mutate(date = as.Date(date))

# Check basic statistics for bodyweight
mean_bw <- mean(cmj$bodyweight_kg, na.rm = TRUE)
sd_bw <- sd(cmj$bodyweight_kg, na.rm = TRUE)
max_change <- max(abs(diff(cmj$bodyweight_kg)), na.rm = TRUE)

# use 86.31 as mean imputation for bodybweight_kg outliers
cmj <- cmj %>%
  mutate(bodyweight_kg = if_else(date == as.Date("2024-05-31", "2024-5-13"), 86.31,
bodyweight_kg))


# convert kg to pounds
# Ensure bodyweight_kg is numeric
cmj$bodyweight_kg <- as.numeric(cmj$bodyweight_kg)

# Convert kg to pounds and round to one decimal place
cmj <- cmj %>%
  mutate(bodyweight_in_pounds_lbs = round(bodyweight_kg * 2.20462, 1))



# Recalculate CMJ metrics that depend on bodyweight now that imputed correctly

cmj <- cmj %>%
  mutate(
    # Recalculate Concentric Impulse (overwrite existing values)
    concentric_impulse_n_s = if_else(!is.na(concentric_impulse_n_s) &
!is.na(bodyweight_kg),
                                     concentric_impulse_n_s / bodyweight_kg *
bodyweight_kg,  # Correcting for bodyweight errors
                                     concentric_impulse_n_s),

    # Recalculate RSI (Reactive Strength Index)
    rsi_modified_m_s = if_else(!is.na(jump_height_flight_time_in_inches_in) &
!is.na(eccentric_duration_ms),
                               jump_height_flight_time_in_inches_in /
(eccentric_duration_ms / 1000), # RSI = Jump Height / Eccentric Time
                               rsi_modified_m_s),

    # Recalculate Eccentric Braking Impulse (overwrite existing values)
    eccentric_braking_impulse_n_s = if_else(!is.na(eccentric_braking_impulse_n_s) &
!is.na(bodyweight_kg),
                                            eccentric_braking_impulse_n_s / bodyweight_kg
* bodyweight_kg,  # Correcting for bodyweight changes
                                            eccentric_braking_impulse_n_s)
  )

# Delete cmj date for 8-4 due to missing bodyweight originally, peak power/bm at 3.26 and
#multiple metrics at >1.5 z scores errors.
# Remove CMJ data for 8/4/2024
cmj <- cmj %>%
  filter(date != as.Date("2024-08-04"))



#### Data Visualizations and Stats -------------
# Achilles soreness over time
ggplot(catapult_wellness_sleep, aes(x = date, y = achilles_soreness)) +
  geom_line(color = "red", size = 1) +
```

```
  geom_point(color = "black", size = 2) +
  labs(title = "Achilles Soreness Over Time", x = "Date", y = "Soreness Level") +
  theme_minimal()

# Sleep Quality and Quantity over time
ggplot(catapult_wellness_sleep, aes(x = date)) +
  geom_line(aes(y = sleep_quality_10_best, color = "Sleep Quality"), size = 1) +
  geom_line(aes(y = sleep_quantity_hr, color = "Sleep Quantity"), size = 1) +
  labs(title = "Sleep Trends Over Time", x = "Date", y = "Sleep Score") +
  scale_color_manual(values = c("Sleep Quality" = "blue", "Sleep Quantity" = "green")) +
  theme_minimal()

# Correlations: relationships between soreness, sleep, and performance.
# 10+ variables, at least 100 observations are recommended.
correlation_data <- catapult_wellness_sleep %>%
  select(achilles_soreness, general_soreness, sleep_quantity_hr, sleep_quality_10_best,
         load, distance, jumps, accelerations, decelerations, intensity_pl_min,
load_per_distance)

# Compute correlation matrix (pairwise complete)
cor_matrix <- cor(correlation_data, use = "pairwise.complete.obs")

# Plot heatmap
corrplot(cor_matrix, method = "color", type = "lower", tl.col = "black", tl.srt = 45,
         addCoef.col = "black", number.cex = 0.7)

####Compare Pre- and Post-All-Star Break Performance
#Though this looks significant, Does Achilles Soreness Change After the All-Star
#Break? All tests Wilcoxon Rank-Sum Test, T Test, Regression) were insignificant)
ggplot(catapult_wellness_sleep, aes(x = date, y = load_per_distance, color = date >=
as.Date("2024-07-26"))) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Load Per Distance Before and After All-Star Break",
       x = "Date", y = "Load Per Distance", color = "Post-All-Star Break") +
  theme_minimal()


#### Time-Series Trends: Achilles Soreness vs. Load, Distance, Jumps, and
Intensity############
catapult_wellness_sleep$date <- as.Date(catapult_wellness_sleep$date)

# Select relevant columns
plot_data <- catapult_wellness_sleep %>%
  select(date, achilles_soreness, load, distance, jumps, intensity_pl_min)

# Convert date to Date format
plot_data$date <- as.Date(plot_data$date)

# Plot with two y-axes
ggplot(plot_data, aes(x = date)) +
  geom_line(aes(y = load, color = "Load"), size = 1) +
  geom_line(aes(y = distance, color = "Distance"), size = 1) +
  geom_line(aes(y = jumps, color = "Jumps"), size = 1) +
  geom_line(aes(y = intensity_pl_min, color = "Intensity"), size = 1) +
  geom_line(aes(y = achilles_soreness * 10, color = "Achilles Soreness"), size = 1,
linetype = "dashed") +  # Adjust scale
  scale_y_continuous(sec.axis = sec_axis(~ . / 10, name = "Achilles Soreness")) +
  labs(title = "Time-Series: Soreness vs Performance Metrics",
       x = "Date",
       y = "Primary Metrics",
       color = "Metric") +
  theme_minimal()

## Pre- vs. Post-All-Star Break Comparison, skewed soreness so wilcoxson test
```

```
# Define pre- and post-All-Star Break groups
pre_break <- catapult_wellness_sleep %>% filter(date < as.Date("2024-07-26"))
post_break <- catapult_wellness_sleep %>% filter(date >= as.Date("2024-07-26"))

# Perform Wilcoxon Rank-Sum Test (Non-Parametric)
wilcox_test <- wilcox.test(pre_break$achilles_soreness, post_break$achilles_soreness,
exact = FALSE)

# Print results
print(wilcox_test)



ggplot(catapult_wellness_sleep, aes(x = date, y = achilles_soreness, color = date >=
as.Date("2024-07-26"))) +
  geom_point(size = 3, alpha = 0.6) +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Achilles Soreness Before & After All-Star Break",
       x = "Date", y = "Soreness Level", color = "Post-All-Star Break") +
  theme_minimal()


### Anomaly Detection: Spotting Sudden Workload Spikes
# Compute rolling average and flag spikes
catapult_wellness_sleep <- catapult_wellness_sleep %>%
  mutate(load_rolling_avg = rollmean(load, k = 7, fill = NA, align = "right"),  # 7-day
rolling avg
         load_spike = case_when(
           load > (load_rolling_avg * 1.5) ~ "Severe Spike",
           load > (load_rolling_avg * 1.3) ~ "Moderate Spike",
           TRUE ~ "Normal"
         ))


# Plot workload with anomaly detection
ggplot(catapult_wellness_sleep, aes(x = date, y = load, color = load_spike)) +
  geom_line(size = 1) +
  geom_point(size = 3, alpha = 0.7) +
  scale_color_manual(values = c("Severe Spike" = "red", "Moderate Spike" = "orange",
"Normal" = "black")) +
  labs(title = "Workload Spike Detection Over Time",
       x = "Date", y = "Load", color = "Spike Type") +
  theme_minimal()

catapult_wellness_sleep %>%
  filter(load_spike != "Normal") %>%
  select(date, load, load_rolling_avg, load_spike)

#Compare Load Per Distance Before & After Spikes
ggplot(catapult_wellness_sleep, aes(x = date, y = load_per_distance, color = load_spike))
+
  geom_line(size = 1) +
  geom_point(size = 3, alpha = 0.7) +
  scale_color_manual(values = c("Severe Spike" = "red", "Moderate Spike" = "orange",
"Normal" = "black")) +
  labs(title = "Load Per Distance Before & After Spikes",
       x = "Date", y = "Load Per Distance", color = "Spike Type") +
  theme_minimal()

## Visualization load per distance vs decels
# Convert date
catapult_wellness_sleep$date <- as.Date(catapult_wellness_sleep$date)

# Ensure numeric columns
catapult_wellness_sleep <- catapult_wellness_sleep %>%
```

```
  mutate(load_per_distance = as.numeric(load_per_distance),
         decelerations = as.numeric(decelerations))

# Plot Load Per Distance vs Decelerations Over Time
ggplot(catapult_wellness_sleep, aes(x = date)) +
  geom_line(aes(y = load_per_distance, color = "Load Per Distance"), size = 1.2) +
  geom_line(aes(y = decelerations * 10, color = "Decelerations"), linetype = "dashed",
size = 1.2) +  # Adjust scaling if needed
  scale_color_manual(values = c("Load Per Distance" = "blue", "Decelerations" = "red")) +
  labs(title = "Load Per Distance vs. Decelerations Over Time",
       x = "Date",
       y = "Metrics",
       color = "Legend") +
  theme_minimal()


# force plate asymmetry trends visual
# Read the Excel file (No sheet name required, as it's the first sheet)
force_plate_data <- read_excel("C:/Users/prist/OneDrive/Desktop/mercury/cmj.xlsx")

# Ensure the date column is correctly formatted
force_plate_data <- force_plate_data %>%
  mutate(Date = as.Date(date))

# Function to extract numerical values from asymmetry columns (removing "R" or "L" text)
clean_numeric <- function(column) {
  as.numeric(gsub("[^0-9.]", "", as.character(column)))
}

# Extract relevant asymmetry metrics and clean data
force_plate_data <- force_plate_data %>%
  mutate(
    `Jump Height Asymmetry (%)` =
clean_numeric(eccentric_braking_impulse_percent_asym_percent),
    `Peak Power Asymmetry (%)` =
clean_numeric(eccentric_deceleration_impulse_percent_asym_percent),
    `RSI Asymmetry (%)` = clean_numeric(landing_impulse_percent_asym_percent),
    `Eccentric Braking Asymmetry (%)` =
clean_numeric(eccentric_peak_force_percent_asym_percent)
  )

# Create the Force Plate Asymmetry Trend Plot
ggplot(force_plate_data, aes(x = Date)) +
  geom_line(aes(y = `Jump Height Asymmetry (%)`, color = "Jump Height Asymmetry"), size=1)
+
  geom_line(aes(y = `Peak Power Asymmetry (%)`, color = "Peak Power Asymmetry"), size=1) +
  geom_line(aes(y = `RSI Asymmetry (%)`, color = "RSI Asymmetry"), size=1) +
  geom_line(aes(y = `Eccentric Braking Asymmetry (%)`, color = "Eccentric Braking
Asymmetry"), size=1) +
  geom_point(aes(y = `Jump Height Asymmetry (%)`, color = "Jump Height Asymmetry"),
size=3) +
  geom_point(aes(y = `Peak Power Asymmetry (%)`, color = "Peak Power Asymmetry"), size=3)
+
  geom_point(aes(y = `RSI Asymmetry (%)`, color = "RSI Asymmetry"), size=3) +
  geom_point(aes(y = `Eccentric Braking Asymmetry (%)`, color = "Eccentric Braking
Asymmetry"), size=3) +
  scale_color_manual(values = c("Jump Height Asymmetry" = "blue",
                                "Peak Power Asymmetry" = "red",
                                "RSI Asymmetry" = "green",
                                "Eccentric Braking Asymmetry" = "purple")) +
  labs(title = "Force Plate Asymmetry Trends Over Time",
       x = "Date",
       y = "Asymmetry Percentage",
       color = "Metrics") +
  theme_minimal() +
```

```r
  theme(axis.text.x = element_text(angle = 45, hjust = 1))


# Statistics: eploratory model
# Shift soreness forward by 1-3 days to see post-spike impact
catapult_wellness_sleep <- catapult_wellness_sleep %>%
  arrange(date) %>%
  mutate(achilles_soreness_tomorrow = lead(achilles_soreness, 1),
         achilles_soreness_2days = lead(achilles_soreness, 2),
         achilles_soreness_3days = lead(achilles_soreness, 3))

# Fit a regression model: Does a spike predict higher soreness?  No
soreness_model <- lm(achilles_soreness_tomorrow ~ load_spike, data =
catapult_wellness_sleep)

summary(soreness_model)  # Check model coefficients


#Regression Load Per Distance) RSI

# Convert date columns to Date format for proper merging
catapult$date <- as.Date(catapult$date)
cmj$date <- as.Date(cmj$date)

# Merge datasets on date to align Load Per Distance and RSI
merged_data <- inner_join(catapult, cmj, by = "date")

# Ensure numeric columns are in the correct format
merged_data <- merged_data %>%
  mutate(load_per_distance = as.numeric(load_per_distance),
         rsi = as.numeric(rsi_modified_m_s)) %>%  # Rename RSI for simplicity
  drop_na()

# Perform linear regression
model <- lm(rsi ~ load_per_distance, data = merged_data)

# Summary of the regression model
summary(model)

# Extract model statistics
model_stats <- glance(model)
print(model_stats)

# Visualization: Load Per Distance vs RSI with Regression Line
ggplot(merged_data, aes(x = load_per_distance, y = rsi)) +
  geom_point(alpha = 0.7, color = "blue") +  # Scatter plot
  geom_smooth(method = "lm", se = TRUE, color = "red") +  # Regression line
  labs(title = "Load Per Distance vs. RSI Regression",
       x = "Load Per Distance",
       y = "Reactive Strength Index (RSI)") +
  theme_minimal()


# regression load vs soreness
# Ensure date
catapult_wellness_sleep$date <- as.Date(catapult_wellness_sleep$date)

# Create a "workload spike" variable (change in load from previous day)
catapult_wellness_sleep <- catapult_wellness_sleep %>%
  arrange(date) %>%
  mutate(workload_spike = load - lag(load))

# Perform linear regression
model <- lm(achilles_soreness ~ workload_spike, data = catapult_wellness_sleep)
```

```r
# Show regression summary
summary(model)

# Extract model statistics
model_stats <- glance(model)
print(model_stats)

# Regression model for soreness 2-3 days after spikes
soreness_model_2day <- lm(achilles_soreness_2days ~ load_spike, data =
catapult_wellness_sleep)
soreness_model_3day <- lm(achilles_soreness_3days ~ load_spike, data =
catapult_wellness_sleep)

summary(soreness_model_2day)  # Check 2-day delay impact
summary(soreness_model_3day)  # Check 3-day delay impact

# Full model including sleep & recovery metrics
soreness_model_full <- lm(achilles_soreness_tomorrow ~ load_spike + sleep_quality_10_best
+ sleep_quantity_hr, data = catapult_wellness_sleep)

summary(soreness_model_full)  # Check if sleep amplifies soreness post-spike


# Compute rolling 7-day workload
catapult_wellness_sleep <- catapult_wellness_sleep %>%
  mutate(rolling_load_7day = rollmean(load, k = 7, fill = NA, align = "right"))

# Regression: Does cumulative load predict soreness better than spikes?
soreness_model_cumulative <- lm(achilles_soreness_tomorrow ~ rolling_load_7day, data =
catapult_wellness_sleep)

summary(soreness_model_cumulative)

# Regression model: Does soreness predict lower intensity?
performance_model <- lm(intensity_pl_min ~ achilles_soreness, data =
catapult_wellness_sleep)

summary(performance_model)
```