# Learning Python for Data Science

Tanner J. Bessette     Advisor: Matt Higham

2024-05-09

## Table of contents

## 0.1 SYE Goals

This semester, I am conducting an independent honors project with the main goal of teaching myself data science in Python. I have taken Foundations of Data Science, as well as many other statistics courses in R, and I have also utilized R for a Summer research project in 2023 and an internship this past Summer. I think Python is an important skill to have for a soon-to-be graduated Statistics major looking to make a career out of statistics or data science, and I had little to no experience with data manipulation, data visualization, and statistical modeling in Python previous to this project. After going through a data science textbook with Python coding instructions and tasks, I split the remainder of my SYE semester into two major projects: an Olympics project and an NBA project.

In this write-up, I will first highlight R and Python coding similarities and differences that I found. Then, I will include my Olympics project code in R, then the same project with equivalent Python code, with analysis and code langauge comparisons included. Finally I will include my entire NBA project, where instead of doing it entirely in R then entirely in Python, I utilized R first to read in the data, wrangle the data, and visualize the data, and used Python to create a knn model with the goal of successfully predicting NBA all-stars. For this project, I primarily used the `pandas` package in Python for data manipulation and the `tidyverse` package in R.

In the following section, I complete a short analysis on Olympics data in both R and Python, pointing out comparisons between the two languages along the way. Then, I complete another project, where I wrangle the data in R first, then attempt to predict whether players are selected to the NBA All-Star game by building a k-nearest-neighbors model in Python. I created this document in Quarto, which supports rednering for both R and Python code.

## 0.2 R vs. Python Language Comparison

After conducting this project, one major take-away was that lots of functions have direct translations from R to Python. Here are some of the helpful functions that I found to be almost equivalent from R to Python:

Functions:

```
-filter() in R → .query() in Python
```

```
-group_by() in R → .groupby() in Python
```

```
-read_csv() in R → pd.read_csv() in Python
```

```
-as.factor() in R → .astype("category") in Python
```

```
-head() in R → dataset.head() in Python
```

```
-arrange() in R → .sort_values(by = ‘variable name’, ascending = False)
```

-pivot_longer() and pivot_wider() in R with names_from and values_from as arguments

→ .pivot() in Python with index, columns, values as arguments

-left_join(), right_join(), full_join in R → pd.merge() with on = "variable name" and how as

-as.factor() in R → .astype(‘category’) in Python

-n() in R → .size() in Python

Other Syntax:

-%in% in R → .isin() in Python

-& in R → and in Python

- | in R → or in Python

-FALSE in R → False in Python

-TRUE in R → TRUE in Python

`-c() to access/initialize a list in R → [] in Python`

`-  |>  in R → . in Python`

Additionally, I have found the plotting structure to be overall very similar using `ggplot` in R and `plotnine` in Python. There were slight differences with quotes which I found to be a difference between R and Python in other situations as well, but functions within plots are all named the same from what I've seen, with the same/similar inputs.

Of course, there were differences I encountered as well. A specific difference that I found challenging was that there is not a good Python equivalent to mutate in R, which I find to be one of the most helpful and one of my most-used functions when I am manipulating data in R. Transform in Python works similarly to a mutate in R that is doing a mathematical operation, and assign in Python creates a new column similarly to mutate, but there is not an exact mutate equivalent in Python.

There are also some big downsides to using Python instead of R in this setting, in my opinion. After spending a few months getting familiar using and practicing Python, there are some significant downsides to using Python for a basic data science project, and reasons that I would personally prefer R. The first reason is that code becomes unreadable so quickly in Python. Instead of a simple pipe in R, where you can space out and enter in between each step to clearly show the process and keep the coding tidy, in Python you need to use a . in between each new function you are applying to the dataset, and you cannot neatly space out or enter between operations. Evidence of this difference is how much Python code needs to be broken down into multiple steps as you code, whereas in R it could all be contained within one data piping process easily.

Interpretability is so much more feasible in R. For somebody unfamiliar with either language, it would be easier to learn basic R, and it would be easier for somebody who doesn't code to be able to understand what the coder is doing to the data each step of the data manipulation process. The structure of the code is much cleaner and easier to follow.

The majority of the math that I do in R is using built-in functions, whereas I find it tedious to have to load a library into Python in order to even do math, and then call that package when doing some of the operations. It feels like R was made specifically to perform mathematical and statistical calculations, and Python is good at doing a variety of things, but for math and statistics specifically, it is not as user-friendly. Although I do definitely prefer R overall for this style of project, the two languages are similar, especially with things like reading in data, joining datasets, and basic plotting.

Below is a project that I conducted about the 2024 Paris Olympics, the first of my two projects utilizing both R and Python.

## 0.3 Olympics Project Goals

The main purpose of these projects is to gain familiarity with Python, and to help draw comparisons between Python in R. For this project I am aiming to be able to read in, wrangle, and create visuals all using Python, after first doing it in R.

In this project I wanted to investigate and visualize how the men's and women's teams for all of the countries that participated in the 2024 Olympics performed compared to each other. Specifically, I wanted to visualize and compare the number of athletes and medals for each country's male and female teams.

For my investigation, I utilized a Kaggle dataset called "Paris 2024 Olympic Summer Games" by Petro, who has a PhD and is actively working as a data scientist at CheAI. This dataset was updated daily throughout the Olympics, and includes all of the athletes that competed, all of the events that were hosted, all of the medals that were won, the daily schedules, the sport venues, and so much more. For this project I utilized his athletes and medals datasets. The athletes dataset includes each athlete, their gender, the country they are competing for, and other facts about them. The medals dataset includes information on each gold, silver, and bronze medal that was won in each event throughout the Olympics, and the athlete/team that won each medal.

Here is the R code that I utilized for this project. After I am finished with the R code, I will include the equivalent Python code to do the same process to help compare the languages.

## 0.4 Olympics Project in R

Load in the necessary libraries and read in the athletes and medals datasets:

```r
library(tidyverse)
library(hrbrthemes)
Athletes_df <- read_csv("/Users/tannerbessette/Desktop/SYE/athletes.csv")
Medals_df <- read_csv("/Users/tannerbessette/Desktop/SYE/medals.csv")
```

Here is a preview of what the 2024 Olympic athletes dataset looks like:

```r
library(knitr)

example_olympics_athletes <- Athletes_df |>
  slice_head(n = 5) |>
  select("code", "name", "gender", "country_code", "country")

kable(example_olympics_athletes)
```

| code | name | gender | country_code | country |
|---:|:---|:---|:---|:---|
| 1532872 | ALEKSANYAN Artur | Male | ARM | Armenia |
| 1532873 | AMOYAN Malkhas | Male | ARM | Armenia |
| 1532874 | GALSTYAN Slavik | Male | ARM | Armenia |
| 1532944 | HARUTYUNYAN Arsen | Male | ARM | Armenia |
| 1532945 | TEVANYAN Vazgen | Male | ARM | Armenia |

In total, the athletes dataset has 36 variables, and 11,113 rows of data.

Here is a preview of the 2024 Olympic medals dataset:

```r
library(knitr)

example_olympics_medals <- Medals_df |>
  slice_head(n = 5) |>
  select("name", "gender", "medal_type", "discipline", "country")

kable(example_olympics_medals)
```

| name | gender | medal_type | discipline | country |
|---|---|---|---|---|
| Remco EVENEPOEL | M | Gold Medal | Cycling Road | Belgium |
| Filippo GANNA | M | Silver Medal | Cycling Road | Italy |
| Wout van AERT | M | Bronze Medal | Cycling Road | Belgium |
| Grace BROWN | W | Gold Medal | Cycling Road | Australia |
| Anna HENDERSON | W | Silver Medal | Cycling Road | Great Britain |

This dataset has a similar structure as the athletes, but has 13 variables and 1,044 rows.

Create the male and female athletes datasets:

```r
Male_Athletes_df <- Athletes_df %>%
  filter(gender == "Male")

Female_Athletes_df <- Athletes_df %>%
  filter(gender == "Female")
```

Calculate the total male and female medals for each sport:

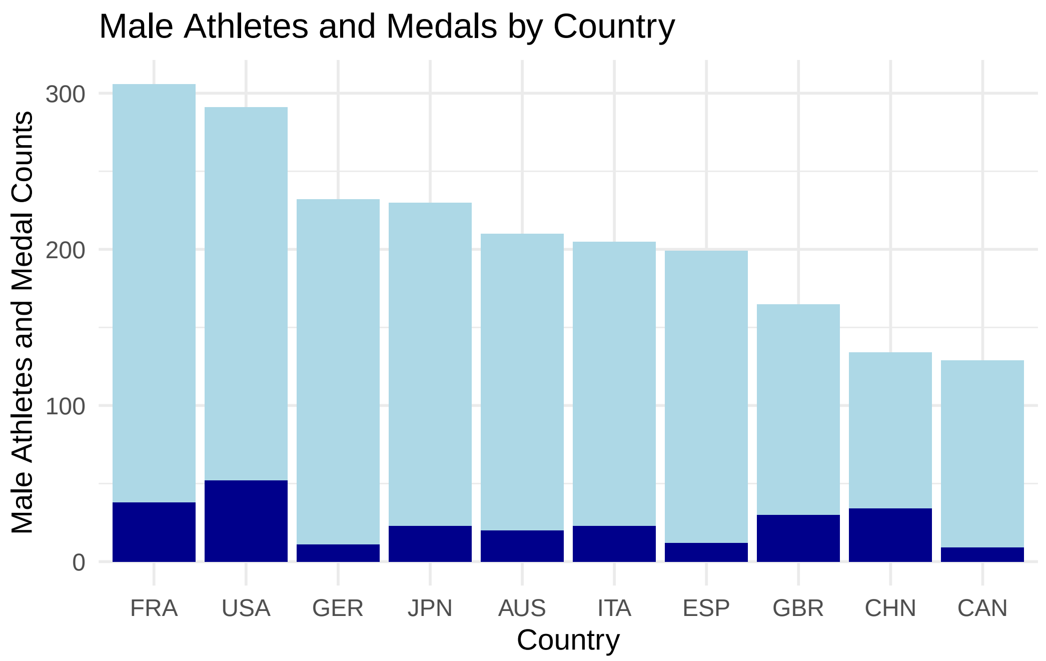
```r
Male_Total_Medal_Counts <- Medals_df %>%
  filter(gender == "M") %>%
  group_by(country_code) %>%
  summarise(medal_totals = n())

Female_Total_Medal_Counts <- Medals_df %>%
  filter(gender == "W") %>%
  group_by(country_code) %>%
  summarise(medal_totals = n())
```

In the bar plots below, each bar represents a country, and the light blue fill represents the number of athletes the country sent, and the dark blue fill represents the number of medals won by each country.

Visualize the total number of male athletes by country, and fill the bars by medals won to indicate the proportion of each country's athletes who medalled:

```r
Male_Athletes_Count <- Male_Athletes_df %>%
  group_by(country_code) %>%
  summarise(athlete_count = n()) %>%
  arrange(desc(athlete_count))

# join the medals counts with the athletes count
Male_Athletes_Count <- left_join(
  Male_Athletes_Count, Male_Total_Medal_Counts, by = "country_code")

# Calculate max country counts and visualize the top 10 countries:
Male_Athletes_Count <- Male_Athletes_Count %>%
  filter(athlete_count > 128)

# Create the plot:
Male_Athletes_Count_barplot <- ggplot(data = Male_Athletes_Count) +
        geom_bar(aes(x = reorder(country_code, -athlete_count), y = athlete_count), fill =
        geom_bar(aes(x = reorder(country_code, -athlete_count), y = medal_totals), fill =
        labs(title = "Male Athletes and Medals by Country",
             x = "Country",
             y = "Male Athletes and Medal Counts") +
        theme_minimal() +
        theme(legend.position = "none")
```

```r
print(Male_Athletes_Count_barplot)
```

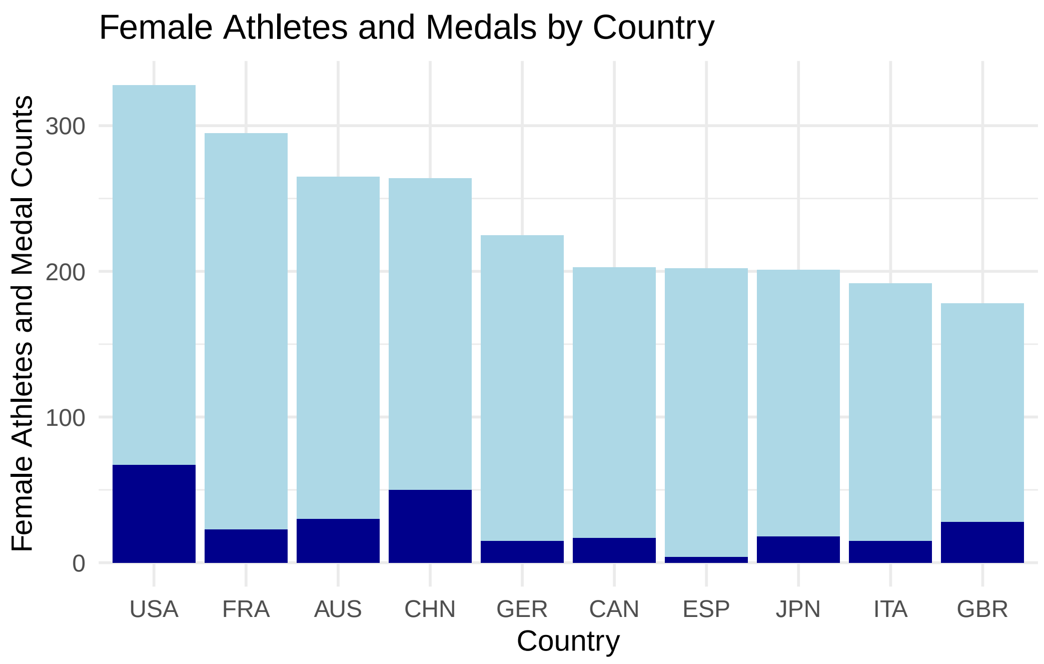


In this plot, the dark blue represents the total number of medals won by men for each country, and the light blue bar indicates the number of male athletes sent by each country.

France and USA both sent many male athletes to the Olympics, and received a lot of medals. Germany and Spain, despite sending many male athletes, received relatively few total medals. China and Great Britain sent less athletes than some other countries, but had a high proportion of their male athletes medal.

Visualize the total number of female athletes by country, and fill the bars by medals won to indicate the proportion of each country's athletes who medalled:

```
Female_Athletes_Count <- Female_Athletes_df %>%
  group_by(country_code) %>%
  summarise(athlete_count = n()) %>%
  arrange(desc(athlete_count))

# join the medals counts with the athletes count
Female_Athletes_Count <- left_join(
  Female_Athletes_Count, Female_Total_Medal_Counts, by = "country_code")

# Calculate max country counts and visualize the top 10 countries:
Female_Athletes_Count <- Female_Athletes_Count %>%
  filter(athlete_count > 177)

# Create the plot:
Female_Athletes_Count_barplot <- ggplot(data = Female_Athletes_Count) +
          geom_bar(aes(x = reorder(country_code, -athlete_count), y = athlete_count), fill =
          geom_bar(aes(x = reorder(country_code, -athlete_count), y = medal_totals), fill =
          labs(title = "Female Athletes and Medals by Country",
               x = "Country",
               y = "Female Athletes and Medal Counts") +
          theme_minimal() +
          scale_fill_viridis_c() +
          theme(legend.position = "none")

print(Female_Athletes_Count_barplot)
```



USA and France sent the most female athletes as well. USA and China had the highest number of their female athletes medal. Despite being seventh in the total number of female athletes that participated in the Olympics, Spain had a remarkably low proportion of their athletes receive medals.

**Heatmaps:**

Heatmaps provide a way to visualize where each country's medals are coming from, and how different countries perform in different sports.

I am dividing sports into categories because there are so many sports in the Olympics that the plot would become unreadable and not interpretable.

Create a Sports Category variable that groups similar sports together:

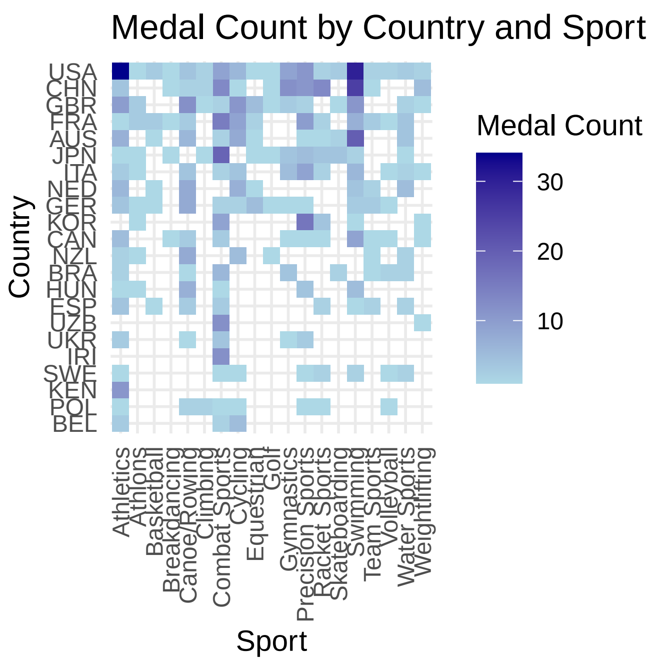
```
# Create new Sport_Category variable for heatmap that reduces
# the amount of categories:
Medal_Counts <- Medals_df %>%
  mutate(Sport_Category = as.factor(case_when(
    discipline %in% c("3x3 Basketball", "Basketball") ~ "Basketball",
    discipline %in% c("Badminton", "Table Tennis", "Tennis") ~ "Racket Sports",
    discipline %in% c("Boxing", "Taekwondo", "Wrestling", "Judo") ~ "Combat Sports",
    discipline %in% c("Swimming", "Marathon Swimming", "Diving", "Artistic Swimming") ~
      "Swimming",
    discipline %in% c("Volleyball", "Beach Volleyball") ~ "Volleyball",
    discipline %in% c("Cycling BMX Freestyle", "Cycling BMX Racing",
                      "Cycling Mountain Bike",
                 "Cycling Road", "Cycling Track") ~ "Cycling",
    discipline %in% c("Water Polo", "Surfing", "Sailing") ~ "Water Sports",
    discipline %in% c("Artistic Gymnastics", "Rhythmic Gymnastics",
                      "Trampoline Gymnastics") ~ "Gymnastics",
    discipline == "Athletics" ~ "Athletics",
    discipline %in% c("Canoe Slalom", "Canoe Sprint", "Rowing") ~ "Canoe/Rowing",
    discipline == "Equestrian" ~ "Equestrian",
    discipline %in% c("Football", "Handball", "Hockey", "Rugby Sevens") ~
      "Team Sports",
    discipline %in% c("Archery", "Shooting", "Fencing") ~ "Precision Sports",
    discipline %in% c("Triathlon", "Modern Pentathlon") ~ "Athlons",
    discipline == "Weightlifting" ~ "Weightlifting",
    discipline == "Skateboarding" ~ "Skateboarding",
    discipline == "Breaking" ~ "Breakdancing",
    discipline == "Sport Climbing" ~ "Climbing",
    discipline == "Golf" ~ "Golf",
    TRUE ~ "Other"  # default if any sport doesn't match
  )))
```

Team Sports is a vague category name, I chose that to keep the name more concise; it consists of team sports besides basketball that are played with more than 4 people, (Rugby, Football, Handball, and Hockey). The "Athlons" level contains the triathlon and Modern Pentathlon.

I decided that a fair cutoff for including countries in the heatmaps was to exclude those with fewer than ten medals. After trying different medal cutoffs, setting any higher than ten made the heatmaps overcrowded.

Create an overall heatmap for medals won by sport and country:**

```
# group by our new Sport_Category variable before counting number of medals
Medal_Counts <- Medal_Counts  %>%
  group_by(country_code, Sport_Category) %>%
  mutate(count = n()) %>%
  arrange(desc(count))

# Only keep countries who got at least 10 total medals:
# (because heatmap was getting sloppy)
Medal_Counts <- Medal_Counts %>%
  group_by(country_code) %>%
  mutate(total_count = n()) %>%
  filter(total_count >= 10)

# Heatmap
ggplot(Medal_Counts, aes(x = Sport_Category,
                         y = fct_reorder(country_code, total_count),
                         fill= count)) +
  geom_tile() +
  labs(title = "Medal Count by Country and Sport",
             x = "Sport",
             y = "Country") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count") +
  coord_fixed()
```

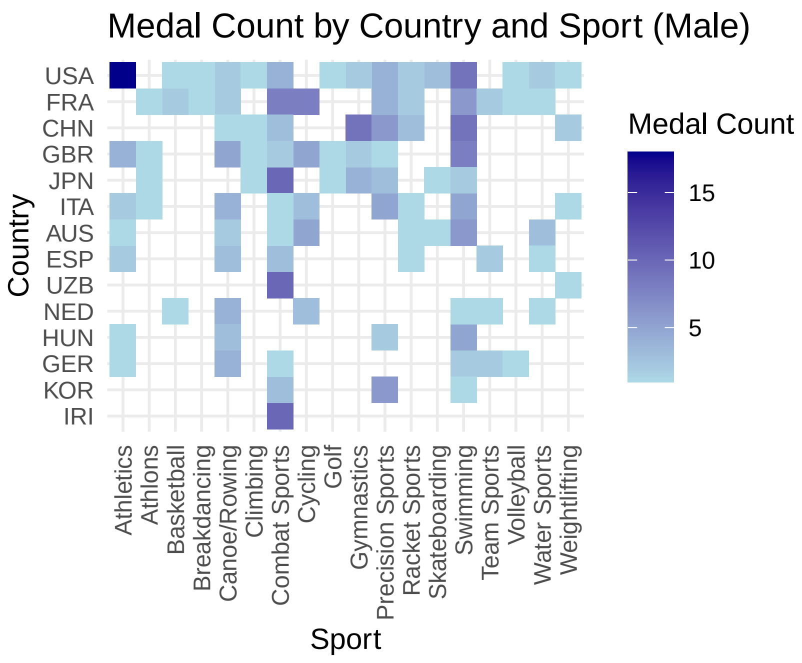


It appears that there is a wide variety of countries who medalled in athletics and in combat sports. All of the countries who had the most overall medals also have a significant number of medals in swimming, likely because there are so many available medals to win in swimming. A country that stands out in the plot is Kenya - they are one of the strongest performers in the whole world in athletics, yet they didn't obtain a single medal in any other sports!

Now, let's split up the heatmap between men and women:

Create the men's medal heatmap:**

```r
# Only keep countries who got at least 10 total medals:
# (because heatmap was getting sloppy)
# group by our new Sport_Category variable before counting number of medals
Male_Medal_Counts <- Medal_Counts  %>%
  filter(gender == "M") %>%
  group_by(country_code, Sport_Category) %>%
  mutate(count = n()) %>%
  arrange(desc(count))

Male_Medal_Counts <- Male_Medal_Counts %>%
  group_by(country_code) %>%
  mutate(total_count = n()) %>%
  filter(total_count >= 10)

# Heatmap
ggplot(Male_Medal_Counts, aes(x = Sport_Category,
                              y = reorder(country_code, total_count),
                              fill= count)) +
  geom_tile() +
  labs(title = "Medal Count by Country and Sport (Male)",
             x = "Sport",
             y = "Country") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count") +
  coord_fixed()
```

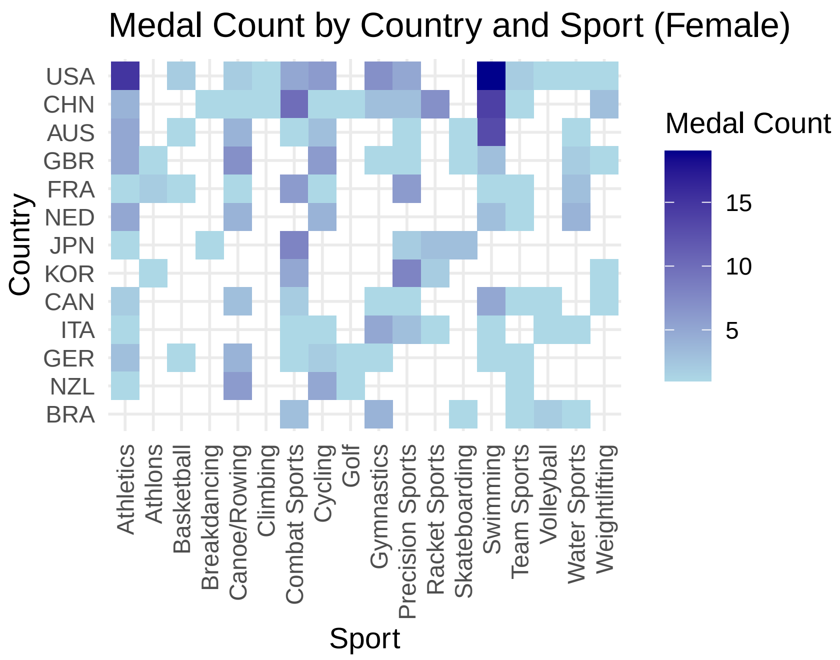


Similarly to Kenya with athletics, Iran received zero medals in all other sports, but they performed very strongly in combat sports, getting some of the most male combat sport medals of any country.

USA received a huge majority of their male medals from athletics. Combat sports, swimming, and canoe/rowing had a wide spread of countries perform well, without just one or two obvious countries dominating a huge majority of those medals.

Create the women's medal heatmap:

```
# Only keep countries who got at least 10 total medals:
# (because heatmap was getting sloppy)
# group by our new Sport_Category variable before counting number of medals
Female_Medal_Counts <- Medal_Counts  %>%
  filter(gender == "W") %>%
  group_by(country_code, Sport_Category) %>%
  mutate(count = n()) %>%
  arrange(desc(count))

Female_Medal_Counts <- Female_Medal_Counts %>%
  group_by(country_code) %>%
  mutate(total_count = n()) %>%
  filter(total_count >= 10)

# Heatmap
ggplot(Female_Medal_Counts, aes(x = Sport_Category,
                                y = reorder(country_code, total_count),
                                fill= count)) +
  geom_tile() +
  labs(title = "Medal Count by Country and Sport (Female)",
             x = "Sport",
             y = "Country") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count") +
  coord_fixed()
```



The USA women clearly received far more medals than any other country's women's teams did. They were especially dominant in swimming, with China and Australia being the only other countries who even came close, and a big drop off after them.

An overall takeaway from the heatmaps is that the countries that tend to get the most overall medals tend to get some medals from almost every sport, and tend to do especially well in swimming and athletics, where there are the most medals up for grabs.

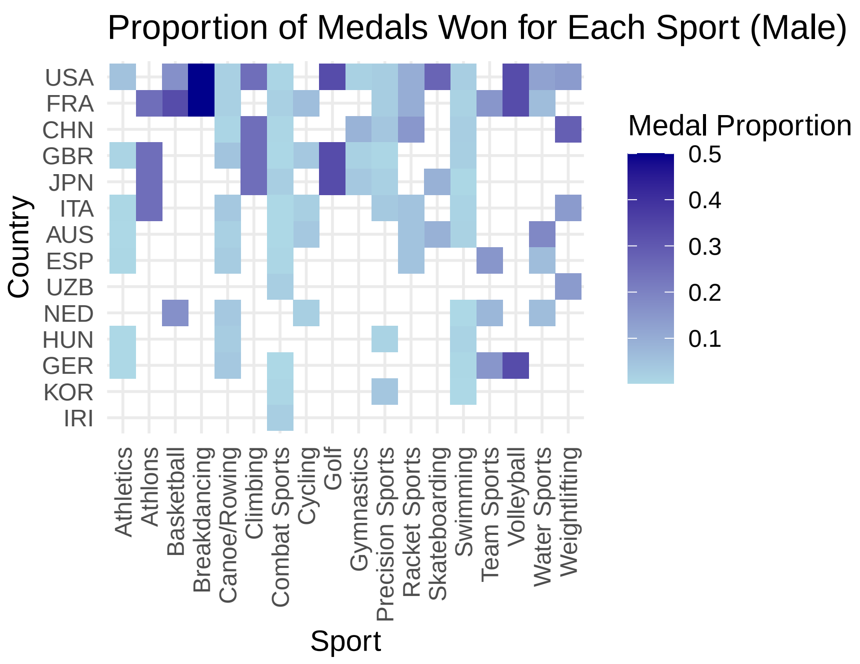
**Create Heatmaps that color based on proportion of medals won in each sport:**

Aside from coloring heatmaps based solely on medal totals, I thought basing heatmaps on proportion of medals could provide additional insights that are harder to see in the previous plots. Specifically, athletics, swimming, and other sports with more medals up for grabs overshadow the other sports by having darker colors. By coloring based on proportion, it shows whcih countries were competitive in the different sports more clearly.

For example, let's say there are 12 total medals available in basketball and the United States won 4 of them, and there are 24 medals available in volleyball and the United States also won 4 of them. Then, their proportion of medals won in basketball was 1/3, and in volleyball was 1/6. These heatmaps would color based on these proportions, rather than showing both squares as 4 medals, as the previous heatmaps did.

Create the men's medal heatmap for proportion of each sport's medals each country won:

```r
Male_Medal_Prop <- Male_Medal_Counts %>%
  group_by(Sport_Category) %>%
  mutate(total_sport_medals = sum(count)) %>%
  mutate(prop_sport_medals = count / total_sport_medals)

# Heatmap
ggplot(Male_Medal_Prop, aes(x = Sport_Category,
                             y = reorder(country_code, total_count),
                             fill= prop_sport_medals)) +
  geom_tile() +
  labs(title = "Proportion of Medals Won for Each Sport (Male)",
             x = "Sport",
             y = "Country") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Proportion") +
  coord_fixed()
```

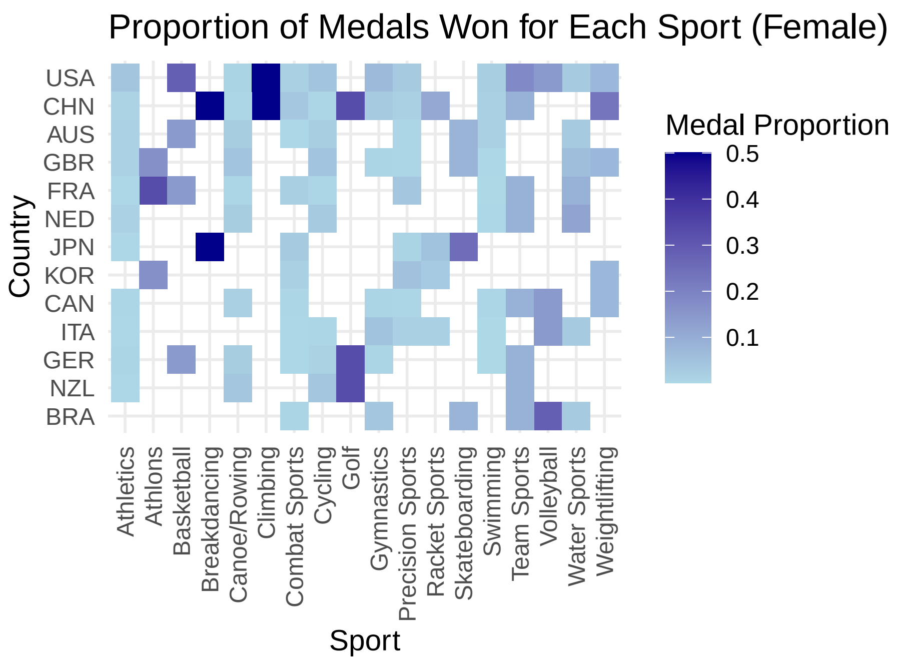


The male proportion heatmap helps to highlight how dominant countries were in certain sports. Specifically, USA and France being the only countries to medal in breakdancing, and USA, France, and Germany being the only countries to medal in volleyball. Basketball, skateboarding, and golf also only had three countries receive a medal in the men's competitions.

It also demonstrates that although certain countries had many medals in certain sports (i.e. USA in athletics), it does not mean that they had as high of a porportion of total medals in that sport.

Create the men's medal heatmap for proportion of each sport's medals each country won:

```r
Female_Prop <- Female_Medal_Counts %>%
  group_by(Sport_Category) %>%
  mutate(total_sport_medals = sum(count)) %>%
  mutate(prop_sport_medals = count / total_sport_medals)

# Heatmap
ggplot(Female_Prop, aes(x = Sport_Category,
                              y = reorder(country_code, total_count),
                              fill= prop_sport_medals)) +
  geom_tile() +
  labs(title = "Proportion of Medals Won for Each Sport (Female)",
             x = "Sport",
             y = "Country") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Proportion") +
  coord_fixed()
```



In the women's competitions, climbing, breakdancing, and golf seem to be the sports that were most dominated by just a couple of countries. Many of the sports in the women's proportions plot have a wide range of countries with roughly the same proportion of medals. Some of these sports include athletics, combat sports, and swimming. Swimming proportions being close was especially surprising to me in this plot, given how many medals we could see that USA, China, and Australia won in total. It just goes to show how many total swimming medals are up for grabs.

Now, I will be going through the same project, this time in Python.

Specify the correct Python path before switching to Python:

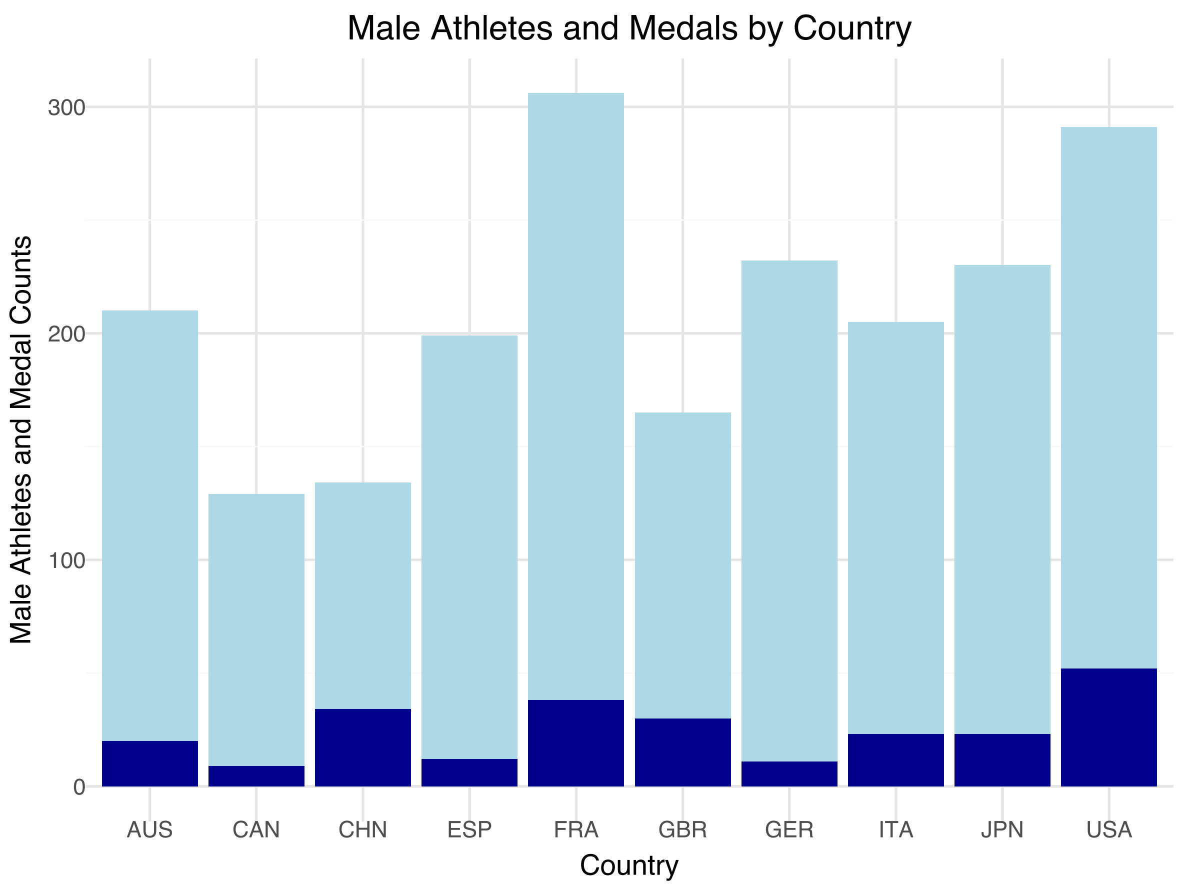
```
library(reticulate)
use_python("/Library/Frameworks/Python.framework/Versions/3.12/bin/python3", required = TRUE
```

## 0.5 Olympics Project in Python

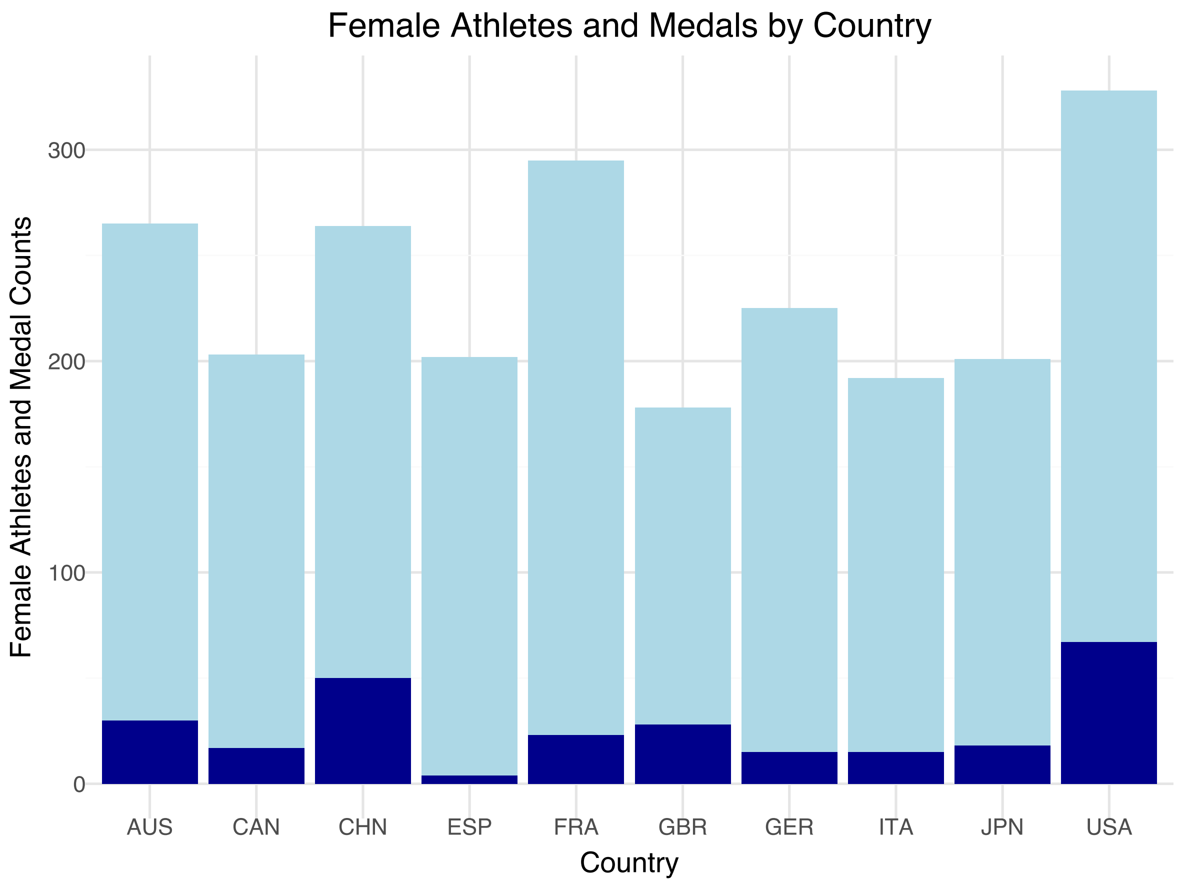
Reading in datasets is very similar, as is filtering (`.query()` in Python), and plotting. Mutations/summarizing are a bit different between the two languages, it was a two step process to create the athlete count variable in Python (`.size()` then `.reset_index()`) instead of one in R.

```
import pandas as pd
import numpy as np
from palmerpenguins import load_penguins
from plotnine import *

# Read in data:
```

```
Athletes_df = pd.read_csv("/Users/tannerbessette/Desktop/SYE/athletes.csv")
Medals_df = pd.read_csv("/Users/tannerbessette/Desktop/SYE/medals.csv")


# Create Male and Female Datasets:
Male_Athletes_df = Athletes_df.query("gender == 'Male'")
Female_Athletes_df = Athletes_df.query("gender == 'Female'")


# Total Male and Female Medals for each sport:
# (learned reset_index from ChatGPT - helps to mimick summarise in R)
Male_Total_Medal_Counts = Medals_df.query("gender == 'M'").groupby("country_code").size().res
Female_Total_Medal_Counts = Medals_df.query("gender == 'W'").groupby("country_code").size().r


# MALE BAR PLOT:
# data manipulation (group_by, summarise):
Male_Athletes_Count = Male_Athletes_df.groupby("country_code").size().reset_index(name = "atl
# join data (looked ahead in textbook to ch. 22):
# (pd.merge is the join function, on is like by, how = join method(L/R))
Male_Athletes_Count_joined = pd.merge(Male_Athletes_Count, Male_Total_Medal_Counts, on="coun
# Only keep top 10 countries with most athletes:
Male_Athletes_Count_joined = Male_Athletes_Count_joined.query("athlete_count > 128")
# Create the Male bar plot with country's athletes and medals:
#Male_Athletes_Count.sort_values (essentially arrange -> do instead of reorder)
Male_Athletes_Count_joined = Male_Athletes_Count_joined.sort_values("athlete_count", ascendir
plot_1 = (
    ggplot() +
          geom_bar(aes(x = 'country_code', y = 'athlete_count'), fill = "lightblue", stat = '
          geom_bar(aes(x = 'country_code', y = 'medal_totals'), fill = "darkblue", stat = "ic
          labs(title = "Male Athletes and Medals by Country",
               x = "Country",
               y = "Male Athletes and Medal Counts") +
          theme_minimal()
)
plot_1.show()
```



```
# FEMALE BAR PLOT:
# data manipulation (group_by, summarise):
Female_Athletes_Count = Female_Athletes_df.groupby("country_code").size().reset_index(name =
# join data (looked ahead in textbook to ch. 22):
# (pd.merge is the join function, on is like by, how = join method(L/R))
Female_Athletes_Count_joined = pd.merge(Female_Athletes_Count, Female_Total_Medal_Counts, on=
# Only keep top 10 countries with most athletes:
Female_Athletes_Count_joined = Female_Athletes_Count_joined.query("athlete_count > 177")
# Create the Male bar plot with country's athletes and medals:
# (arrange first):
Female_Athletes_Count_joined = Female_Athletes_Count_joined.sort_values("athlete_count", asce
plot_2 = (
    ggplot() +
          geom_bar(aes(x = 'country_code', y = 'athlete_count'), fill = "lightblue", stat = '
          geom_bar(aes(x = 'country_code', y = 'medal_totals'), fill = "darkblue", stat = "id
          labs(title = "Female Athletes and Medals by Country",
               x = "Country",
               y = "Female Athletes and Medal Counts") +
            theme_minimal()
)
plot_2.show()
```



Creating the sport categories was different in Python and harder to follow, you need to specify the group order beforehand then the list of categories, instead of using the ~ at each row when I did it in R.

Perform the necessary data manipulations before generating the heatmaps:

```
# Used ChatGPT for this, first step is sorting which sport will be the levels
# for the factor variable, second is assigning names for each level
Medals_df['Sport_Category'] = np.select(
    [
        Medals_df['discipline'].isin(["3x3 Basketball", "Basketball"]),
        Medals_df['discipline'].isin(["Badminton", "Table Tennis", "Tennis"]),
        Medals_df['discipline'].isin(["Boxing", "Taekwondo", "Wrestling", "Judo"]),
        Medals_df['discipline'].isin(["Swimming", "Marathon Swimming", "Diving", "Artistic Sw
        Medals_df['discipline'].isin(["Volleyball", "Beach Volleyball"]),
        Medals_df['discipline'].isin(["Cycling BMX Freestyle", "Cycling BMX Racing",
                                      "Cycling Mountain Bike", "Cycling Road", "Cycling Track
        Medals_df['discipline'].isin(["Water Polo", "Surfing", "Sailing"]),
        Medals_df['discipline'].isin(["Artistic Gymnastics", "Rhythmic Gymnastics", "Trampoli
        Medals_df['discipline'] == "Athletics",
        Medals_df['discipline'].isin(["Canoe Slalom", "Canoe Sprint", "Rowing"]),
        Medals_df['discipline'] == "Equestrian",
        Medals_df['discipline'].isin(["Football", "Handball", "Hockey", "Rugby Sevens"]),
        Medals_df['discipline'].isin(["Archery", "Shooting", "Fencing"]),
        Medals_df['discipline'].isin(["Triathlon", "Modern Pentathlon"]),
        Medals_df['discipline'] == "Weightlifting",
        Medals_df['discipline'] == "Skateboarding",
        Medals_df['discipline'] == "Breaking",
        Medals_df['discipline'] == "Sport Climbing",
        Medals_df['discipline'] == "Golf"
    ],
    [
        "Basketball",
        "Racket Sports",
        "Combat Sports",
        "Swimming",
        "Volleyball",
        "Cycling",
        "Water Sports",
        "Gymnastics",
        "Athletics",
        "Canoe/Rowing",
        "Equestrian",
        "Team Sports",
        "Precision Sports",
        "Athlons",
        "Weightlifting",
        "Skateboarding",
        "Breakdancing",
        "Climbing",
        "Golf"
    ],
    default="Other"  # default if any sport doesn't match
)

# Convert 'Sport_Category' to a categorical type
Medals_df['Sport_Category'] = Medals_df['Sport_Category'].astype('category')
```

Create a medal count variable that counts each country's medals by sport:

```
Medal_Counts = Medals_df.groupby(['country_code', 'Sport_Category']).size().reset_index(name=
```

```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to
```

Perform more manipulations for the combined men's and women's heatmap:

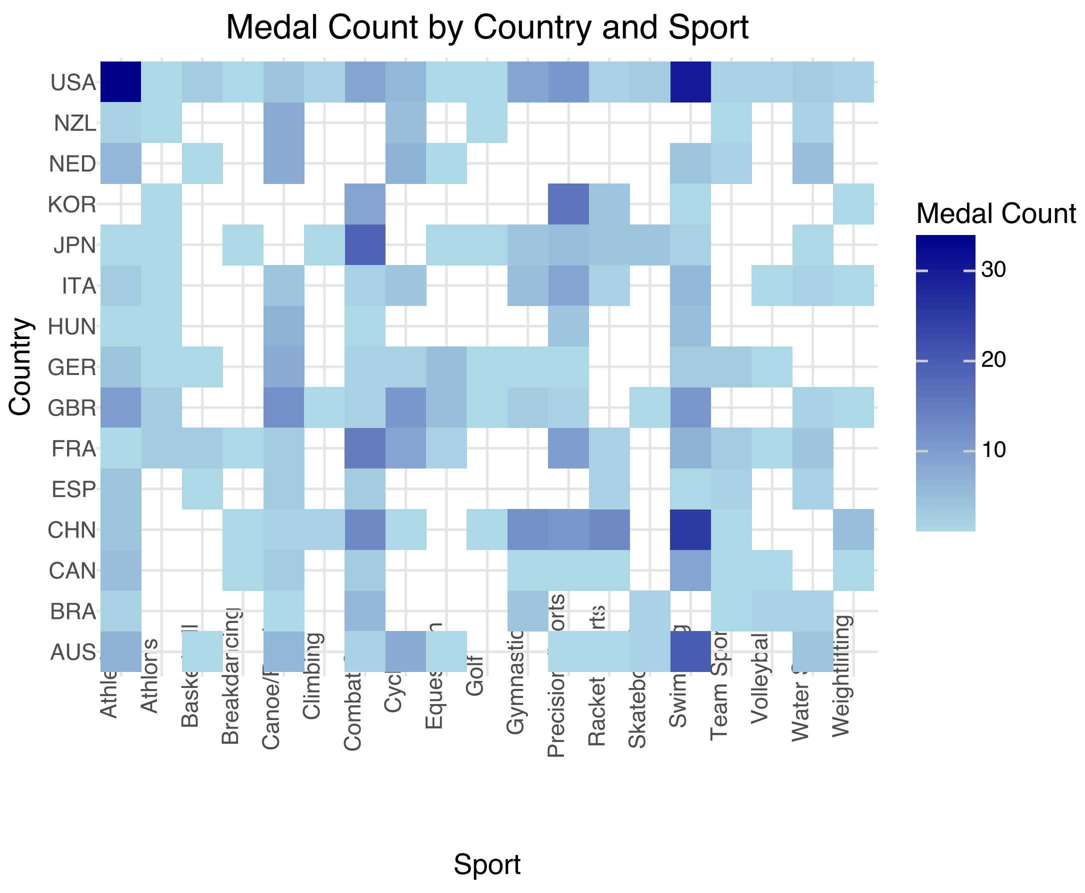
```
# Create a total medal count variable, and only keep countries
# with medals in at least 1 sport
total_counts = Medal_Counts.groupby('country_code').size().reset_index(name='total_count')
Medal_Counts = pd.merge(Medal_Counts, total_counts, on = "country_code",
how = "left")
Medal_Counts = Medal_Counts.query("total_count > 1")
```

In the code chunk below I specified a list of countries to keep, based on the same requirements that I had in my plots in R. The `.query("country_code in @overall_countries_to_keep")` is a good example of some differences in syntax than we would have in doing this in R, specifically with the quotes around the whole filtering statement.

Something to note: unfortunately, for the heatmap plots in Python, the x-axis labels overlap with the plot itself, instead of being below as they were in the R plots - this is a minor weakness of plotting in Python. It could likely be fixed with a little bit of additional code.

Create the combined men's and women's heatmap:

```
overall_countries_to_keep = ['AUS', 'BRA', 'CAN', 'CHN', 'ESP', 'FRA', 'GBR', 'GER', 'HUN',

Overall_Medal_Counts = Medal_Counts.query("country_code in @overall_countries_to_keep").quer

plot_3 = (
    ggplot(Overall_Medal_Counts, aes(x='Sport_Category',
                                     y='country_code',
                                     fill='count')) +
    geom_tile() +
    labs(title="Medal Count by Country and Sport",
         x="Sport",
         y="Country") +
    theme_minimal() +
    theme(axis_text_x=element_text(angle=90, vjust=0.5, hjust=1)) +
    coord_fixed() +
    scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count")
)
plot_3.show()
```



Wrangle the men's data before creating the men's heatmap:

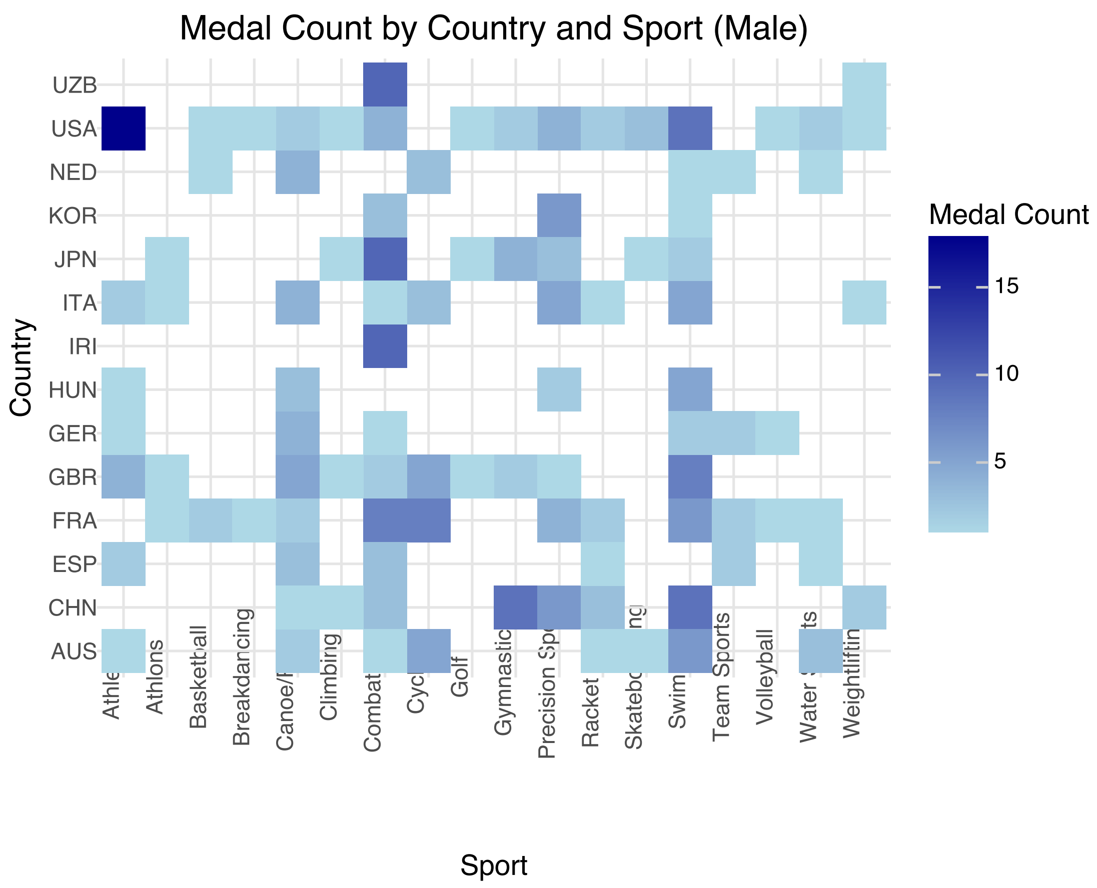
```
Male_Medal_Counts = Medals_df.query("gender == 'M'").groupby(['country_code', 'Sport_Category
```

```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to
```

```
total_male_counts = Male_Medal_Counts.groupby('country_code').size().reset_index(name='total_

Male_Medal_Counts = pd.merge(Male_Medal_Counts, total_male_counts, on = "country_code", how =

Male_Medal_Counts = Male_Medal_Counts.query("total_count > 1")
```

Create the men's heatmap in Python:

```
male_countries_to_keep = ['AUS', 'CHN', 'ESP', 'FRA', 'GBR', 'GER', 'HUN', 'IRI', 'ITA', 'JPN

Updated_Medal_Counts = Male_Medal_Counts.query("country_code in @male_countries_to_keep").que

plot_4 = (
    ggplot(Updated_Medal_Counts, aes(x='Sport_Category',
                             y='country_code',
                             fill='count')) +
    geom_tile() +
    labs(title="Medal Count by Country and Sport (Male)",
         x="Sport",
         y="Country") +
    theme_minimal() +
    theme(axis_text_x=element_text(angle=90, vjust=0.5, hjust=1)) +
    coord_fixed() +
    scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count")
)
plot_4.show()
```



Wrangle the women's data before creating the women's heatmap:

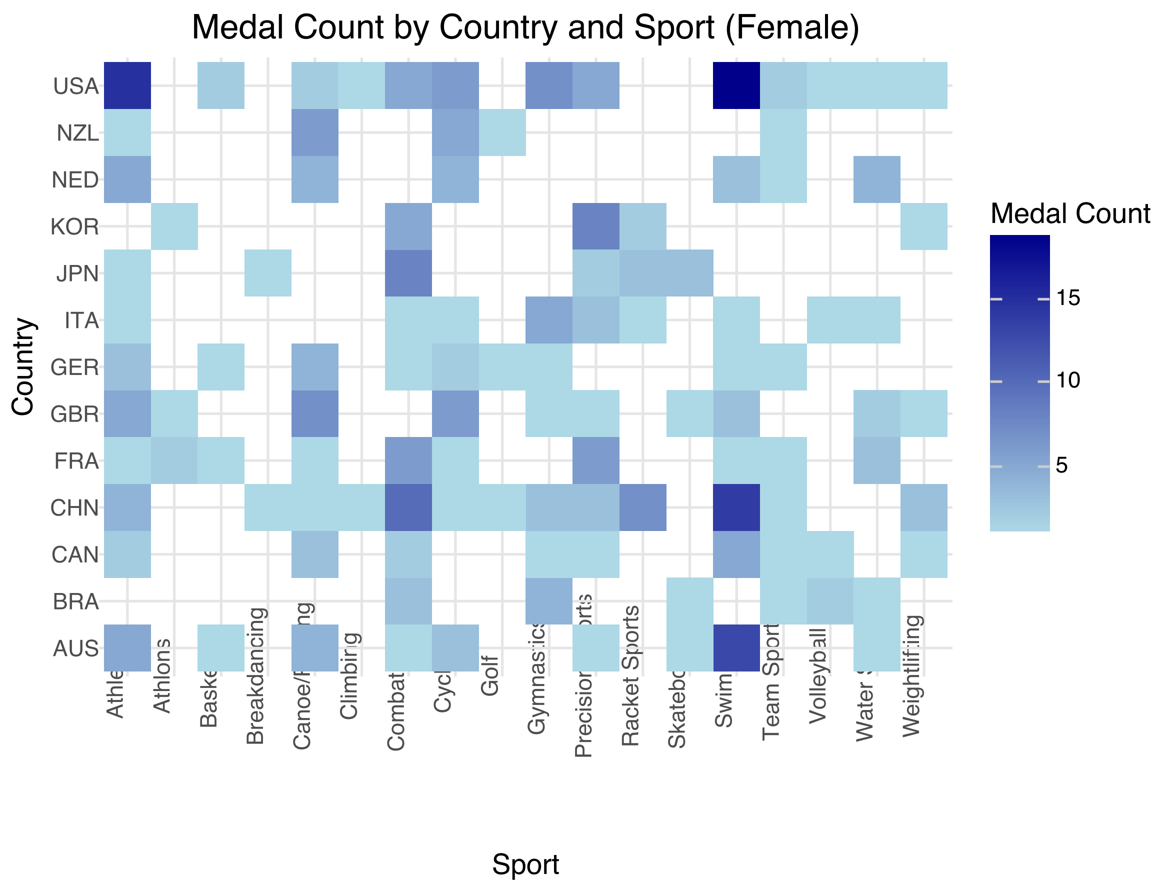
```
Female_Medal_Counts = Medals_df.query("gender == 'W'").groupby(['country_code', 'Sport_Catego
```

```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to
```

```
total_female_counts = Female_Medal_Counts.groupby('country_code').size().reset_index(name='to
Female_Medal_Counts = pd.merge(Female_Medal_Counts, total_female_counts, on = "country_code"
Female_Medal_Counts = Female_Medal_Counts.query("total_count > 1")
```

Create the women's heatmap in Python:

```python
female_countries_to_keep = ['AUS', 'BRA', 'CAN', 'CHN', 'FRA', 'GBR', 'GER',  'ITA', 'JPN',

Updated_Female_Medal_Counts = Female_Medal_Counts.query("country_code in @female_countries_to

plot_5 = (
    ggplot(Updated_Female_Medal_Counts, aes(x='Sport_Category',
                                 y='country_code',
                                 fill='count')) +
    geom_tile() +
    labs(title="Medal Count by Country and Sport (Female)",
         x="Sport",
         y="Country") +
    theme_minimal() +
    theme(axis_text_x=element_text(angle=90, vjust=0.5, hjust=1)) +
    coord_fixed() +
    scale_fill_gradient(low="lightblue", high="darkblue", name = "Medal Count")
)
plot_5.show()
```



Now create heatmaps based on proportion of sport's medals instead of totals:

Create the dataset for men's medal proportions:

```
Male_prop = Male_Medal_Counts
Male_prop['total_sport_medals'] = Male_prop.groupby('Sport_Category')['count'].transform('su
```

```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to
```

```
Male_prop['prop_sport_medals'] = Male_prop['count'] / Male_prop['total_sport_medals']
```

Instead of using a $ in R to access a dataset's variable, you use [] in Python. Instead of a mutate function that I would have used in R, here I am telling Python what variable to save as prop_sport_medals to my male and female datasets.

Create the dataset for women's medal proportions:

```
Female_prop = Female_Medal_Counts
Female_prop['total_sport_medals'] = Female_prop.groupby('Sport_Category')['count'].transform
```

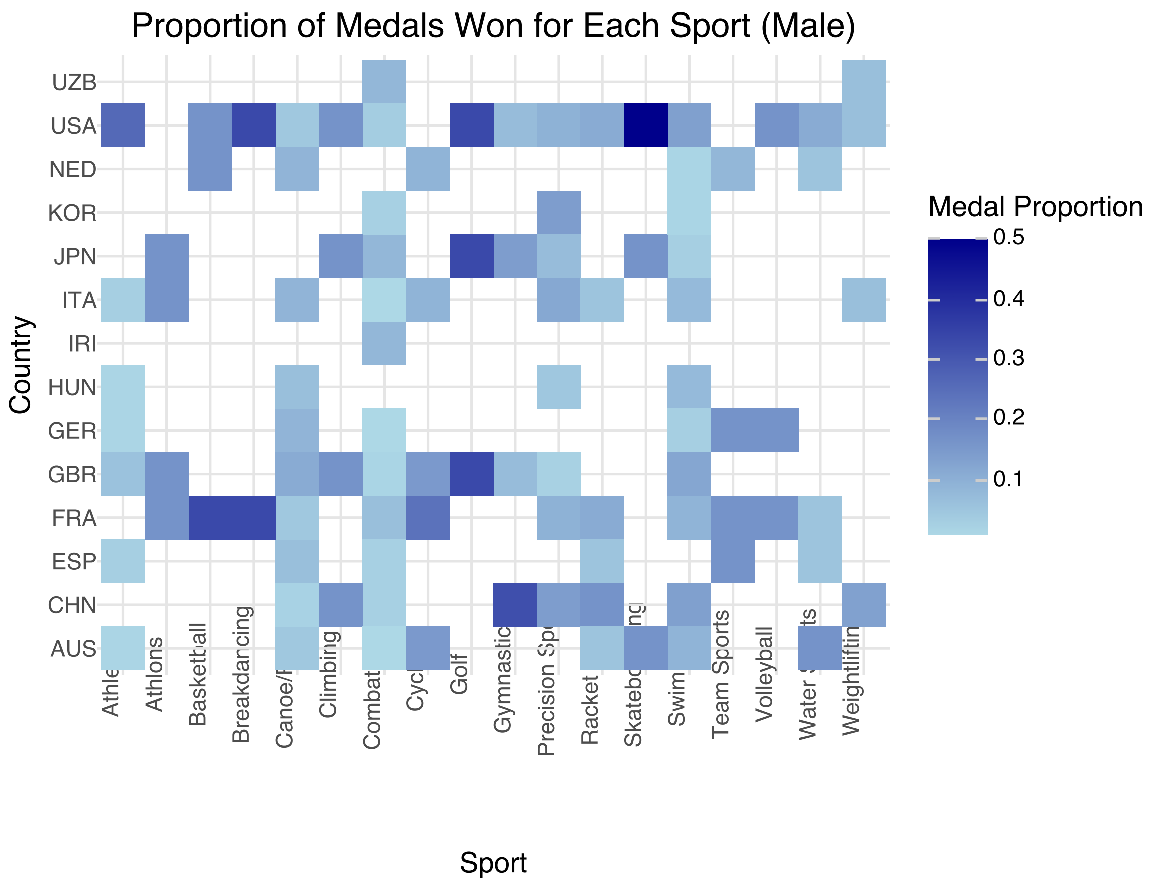
```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to
```

```
Female_prop['prop_sport_medals'] = Female_prop['count'] / Female_prop['total_sport_medals']
```

Now create the proportions heatmap plots:

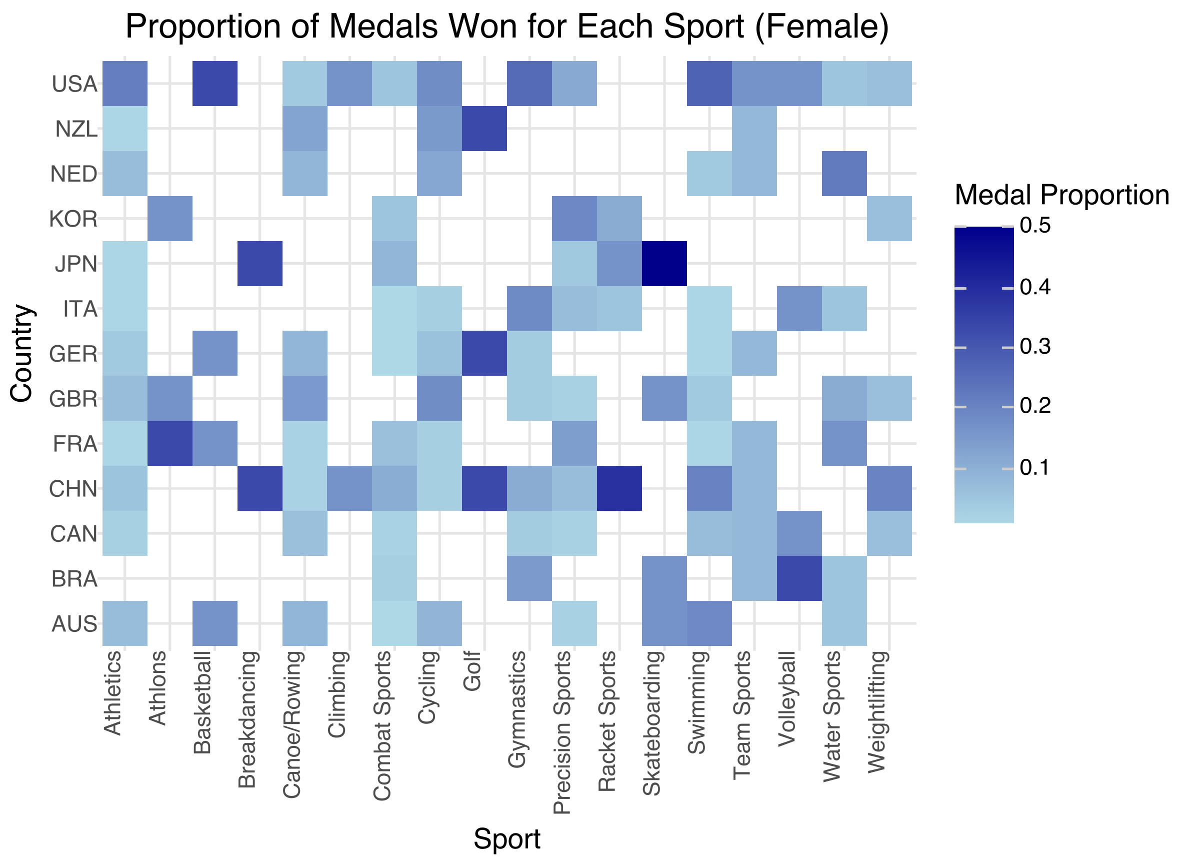
Create the men's proportion heatmap:

```
Updated_Male_Prop = Male_prop.query("country_code in @male_countries_to_keep").query("count >

plot_6 = (
    ggplot(Updated_Male_Prop, aes(x='Sport_Category',
                                  y='country_code',
                                  fill='prop_sport_medals')) +
    geom_tile() +
    labs(title="Proportion of Medals Won for Each Sport (Male)",
        x="Sport",
        y="Country") +
    theme_minimal() +
    theme(axis_text_x=element_text(angle=90, vjust=0.5, hjust=1)) +
    scale_fill_gradient(low="lightblue", high="darkblue", name="Medal Proportion") +
    coord_fixed()
)
plot_6.show()
```



Create the women's proportion heatmap:

```
Updated_Female_Prop = Female_prop.query("country_code in @female_countries_to_keep").query("

plot_7 = (
    ggplot(Updated_Female_Prop, aes(x='Sport_Category',
                                  y='country_code',
                                  fill='prop_sport_medals')) +
    geom_tile() +
    labs(title="Proportion of Medals Won for Each Sport (Female)",
         x="Sport",
         y="Country") +
    theme_minimal() +
    scale_fill_gradient(low="lightblue", high="darkblue", name="Medal Proportion") +
    theme(axis_text_x=element_text(angle=90, vjust=1, hjust=1)) +
    coord_fixed()
)
plot_7.show()
```



Overall, I was able to almost identically replicate each step of my R wrangling and plotting process in my Python analysis.

## 0.6 Specific Code Comparisons

I thought it may be interesting to compare some of the specific data manipulations side by side now that we have seen the project in both R and Python.

**Example 1:**

Before making the male bar plot, I had to group by country and add up all the athletes for each country.

Here is how I did it in R:

```r
Male_Athletes_Count <- Male_Athletes_df %>%
  group_by(country_code) %>%
  summarise(athlete_count = n())
head(Male_Athletes_Count)
```

```
# A tibble: 6 x 2
  country_code athlete_count
  <chr>                <int>
1 AFG                      3
2 AIN                     15
3 ALB                      5
4 ALG                     27
5 AND                      1
6 ANG                      8
```

And here is how I did it in Python:

```python
# data manipulation (group_by, summarise):
Male_Athletes_Count = Male_Athletes_df.groupby("country_code").size().reset_index(name="athle

Male_Athletes_Count.head()
```

```
  country_code  athlete_count
0          AFG              3
1          AIN             15
2          ALB              5
3          ALG             27
4          AND              1
```

Obviously the `group_by()` and `groupby()` functions are extremely similar, as I highlighted in the function comparison. The `.size()` function in Python essentially does the same thing as the combination of a summarise and `n()` in R. In Python, to keep country_code and athlete_count as two different columns and keep the Male_Athletes_Counts as a dataframe, we need to use the `reset_index()` function.

**Example 2:**

Another example of side by side code comparisons of R and Python that I want to show is with joining datasets.

Here is how I performed the joining of the male athletes and male medals datasets before creating the bar plot using R:

```
Male_Athletes_Count <- left_join(Male_Athletes_Count, Male_Total_Medal_Counts,
                                 by = "country_code")
```

And here is how I did that in Python:

```
Male_Athletes_Count_joined = pd.merge(Male_Athletes_Count, Male_Total_Medal_Counts, on="coun
```

Both languages are using a left join, and joining by matching the country_code in the two datasets they are joining. R has a built-in `left_join()` function to specify that the join type is left, while Python's `pd.merge()` function can be used for all types of joins, and you use the how = "left" argument to specify that you want the join to be a left join. The on argument in this R example and the by argument in the Python example are used in the same exact way, to specify the variable to join on.

**Example 3:**

Before creating the heatmaps based on countries' proportions of medals won in each sports category, I had to actually calculate the proportions for each, and save it to a dataset.

Here is how I did that in R:

```
Female_Medal_Prop <- Female_Medal_Counts %>%
  group_by(Sport_Category) %>%
  mutate(total_sport_medals = sum(count)) %>%
  mutate(prop_sport_medals = count / total_sport_medals)
```

And here is how I did it in Python:

```
Female_prop = Female_Medal_Counts
Female_prop['total_sport_medals'] = Female_prop.groupby('Sport_Category', observed = False)[
Female_prop['prop_sport_medals'] = Female_prop['count'] / Female_prop['total_sport_medals']
```

Using R, I used the simple `dplyr` functions of `group_by()` and `mutate()`, and saved an updated version of Female_Medal_Counts that would have two new columns (total medals and proportion of medals for each sport) as Female_Medal_Prop. In Python, I created these same two columns, this time in separate coding commands, and saved these columns to the Female_prop dataset. I edited the two columns using brackets [], which is equivalent to if I had edited the individual columns in R using a dollar sign $, instead of the `dplyr` method.

## 0.7 NBA Project Introduction

Now, onto a new project. This time, I will be looking at datasets that contain every player's NBA statistics, and whether or not that player made it to the NBA all-star game for every season from the year 2000 and on. The purpose for this project is to use both languages to perform a single analysis. I will be importing, wrangling, and visualizing variable correlations using boxplots in R. Then, I will create a knn model in Python with the intent of accurately predicting whether or not each player made the all-star game each season.

## 0.8 NBA Data Wrangling and Plotting in R

Load the necessary libraries and import NBA datasets:

```r
library(tidyverse)
library(dplyr)
player_averages <-
  read_csv("/Users/tannerbessette/Desktop/Python_Textbook/NBA_Project/Player_Per_Game.csv")

all_stars <- read_csv("/Users/tannerbessette/Desktop/Python_Textbook/NBA_Project/All_Star_Se

player_totals <-
  read_csv("/Users/tannerbessette/Desktop/Python_Textbook/NBA_Project/Player_Totals.csv")
```

Join the all_stars and player_averages datasets and create an all-star variable with two levels:

```r
# create an all-star variable with 1 = made all-star and 0 = didn't make it:
all_stars <- all_stars |>
  mutate(all_star = 1)

# Join player averages data with the all_stars data:
joined_all_star_data <- left_join(player_averages, all_stars,
                                  by = c("season", "player"))

# remove variables that we definitely won't need:
joined_all_star_data <- joined_all_star_data |>
  dplyr::select(-c("birth_year", "lg.x", "lg.y", "team"))
```

```r
# fix the all-star variable to be 0s and 1s not 1s and NAs
joined_all_star_data <- joined_all_star_data |>
  mutate(all_star = case_when(
    is.na(all_star) ~ 0,
    all_star == 1 ~ 1))
```

Here is what the basic structure of the dataset looks like:

```r
library(knitr)

slides_nba_data <- joined_all_star_data |>
  slice_head(n = 5) |>
  select("season", "player", "pos", "gs", "pts_per_game",
         "trb_per_game")

kable(slides_nba_data)
```

| season | player | pos | gs | pts_per_game | trb_per_game |
|---:|:---|:---|---:|---:|---:|
| 2025 | A.J. Green | SG | 0 | 3.4 | 0.4 |
| 2025 | AJ Johnson | SG | 0 | 0.7 | 0.3 |
| 2025 | Aaron Gordon | PF | 4 | 15.5 | 7.5 |
| 2025 | Aaron Holiday | PG | 0 | 0.0 | 0.0 |
| 2025 | Aaron Nesmith | SF | 5 | 8.6 | 4.6 |

This is just a snapshot of a few rows and a few variables. In total, the dataset consists of 35 different variables, and 3555 rows of data after the year 2000.

I decided that a fair indication of whether somebody was an "NBA starter" and should remain in the dataset was whether somebody started at least half of their team's games in a given

NBA season (42+ games started). I also only kept seasons in the 21st century, since the game has evolved so much since before the year 2000. Also, I excluded the year 2025, because the season is active, and there has not yet been an all-star game this season.

Filter down the data to keep only the recent years and players who are NBA starters:

```
# specify the seasons:
joined_all_star_data <- joined_all_star_data |>
  filter(season > 1999) |>
  filter(season != 2025)

# specify the number of starts each player must have (at least half of the season):
joined_all_star_data <- joined_all_star_data |>
  filter(gs > 41)
```

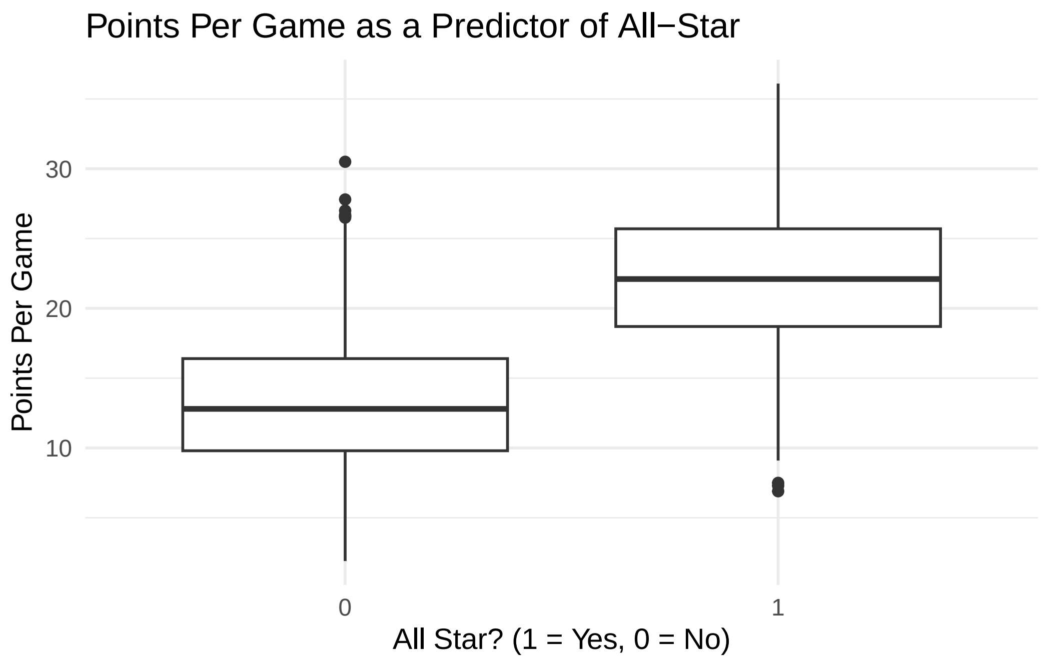
Export the now manipulated data so we can model with Python later:

```
# export updated dataset so we can import again with Python:
write_csv(joined_all_star_data, "/Users/tannerbessette/Desktop/Python_Textbook/NBA_Project/jo
```

**Visualize NBA data in R:**

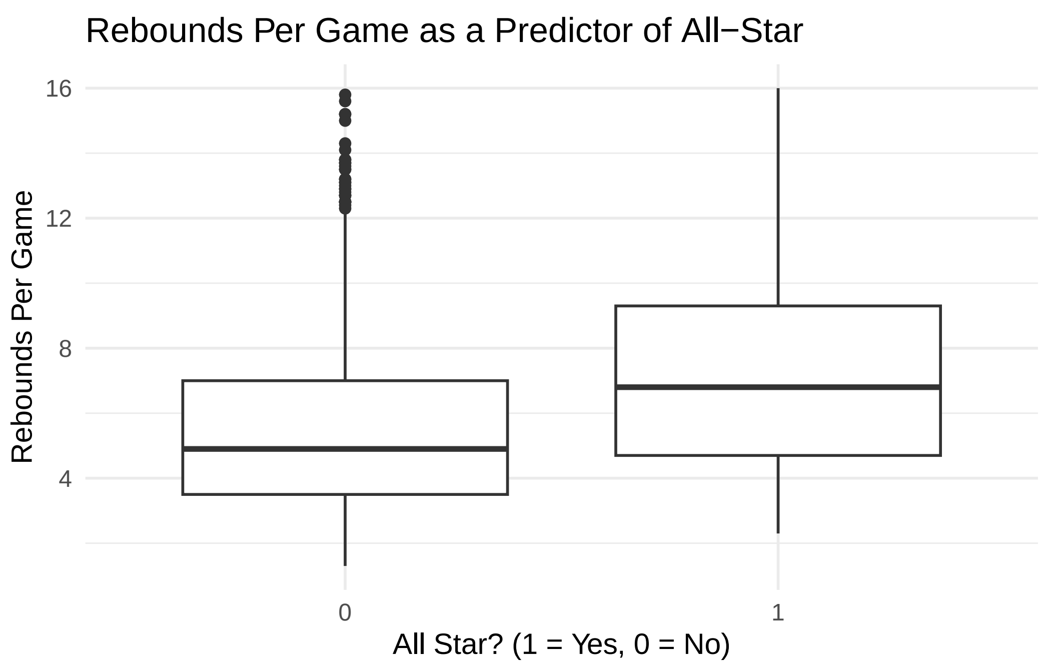
First let's investigate some of the most common statistics, and create box plots to see how points, rebounds, and assists correlate to making the NBA all-star game.

```
library(ggplot2)

# Points:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = pts_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Points Per Game",
         title = "Points Per Game as a Predictor of All-Star") +
    theme_minimal()
```



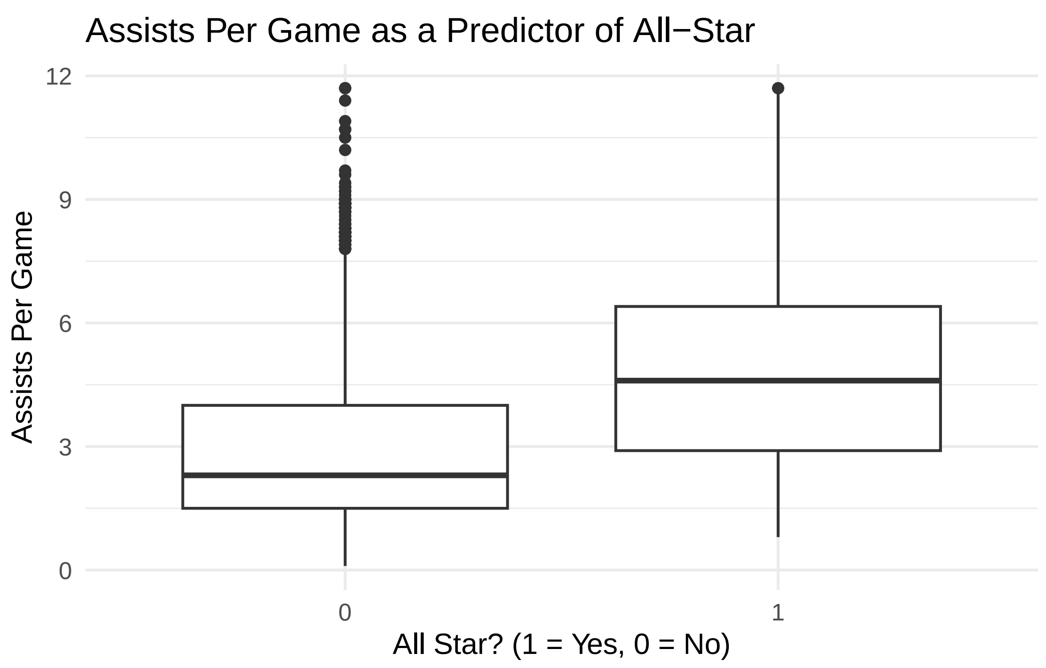
Point per game appears to have a strong positive correlation with making the all-star game. It looks like just one player who scored more than 30 points per game did not make the all-star game that season.

```r
# Rebounds:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = trb_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Rebounds Per Game",
         title = "Rebounds Per Game as a Predictor of All-Star") +
    theme_minimal()
```



Rebounds per game does not have as strong of a correlation as I would have expected, but there still does appear to be a general trend of more rebounds per game means increased likelihood of being an all-star selection.

```
# Assists:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = ast_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Assists Per Game",
         title = "Assists Per Game as a Predictor of All-Star") +
    theme_minimal()
```

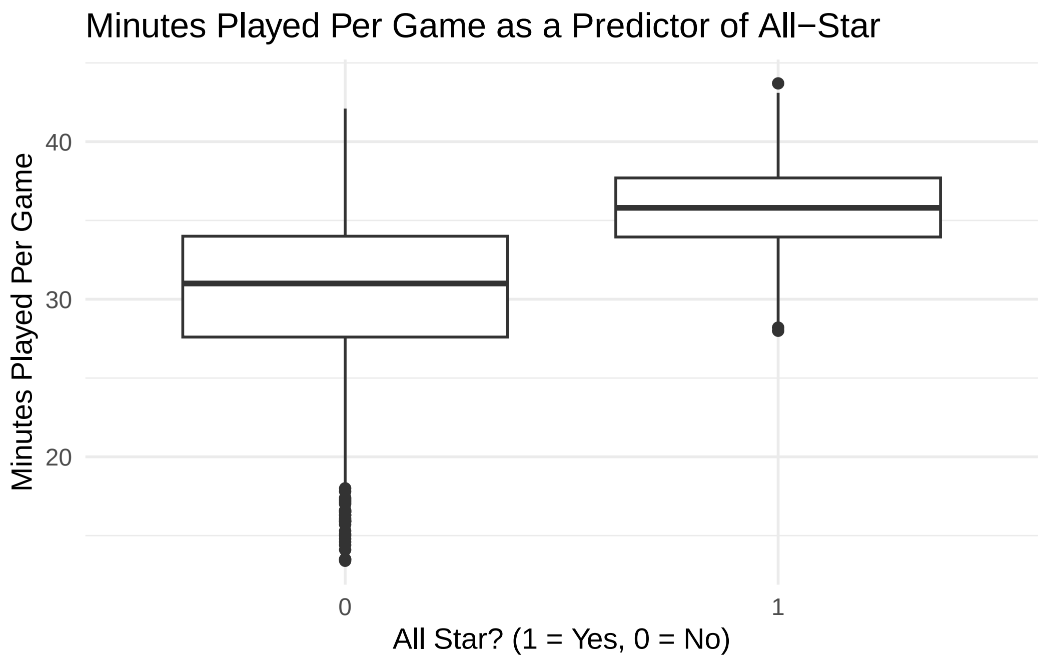


Assists appears to have a stronger correlation with making the all-star game than rebounds does, but a weaker correlation with making the all-star game than points does.

All three seem to have a difference in means between making and missing all-star game as we would expect, but of the three, points seems to be the most significant predictor, then assists, then rebounds seem to be a bit less significant.

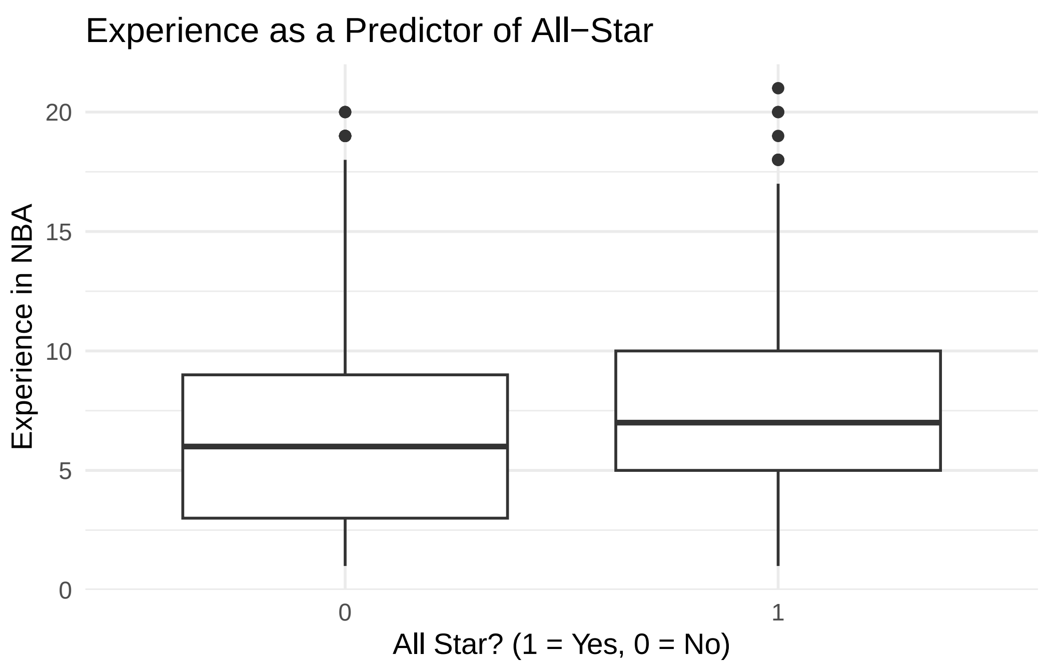
Let's try the same plots for minutes played, experience, 3-pointers per game, blocks and steals per game.

```r
# Minutes Played:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = mp_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Minutes Played Per Game",
          title = "Minutes Played Per Game as a Predictor of All-Star") +
    theme_minimal()
```



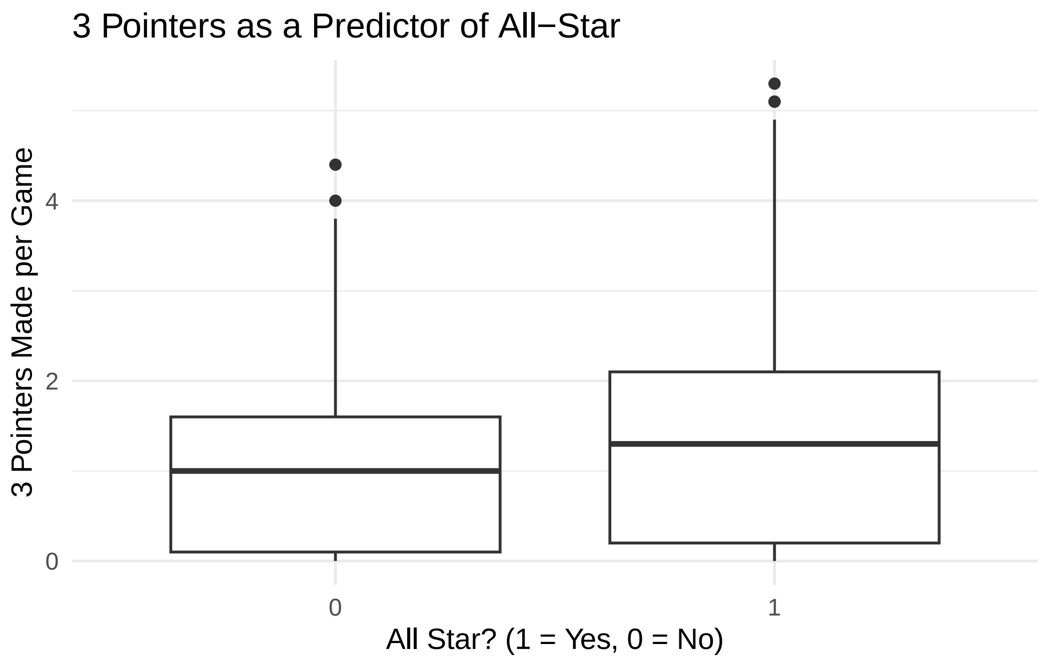
An interesting takeaway from this plot is out of players playing less than 26/27 minutes per game, not a single one made the all-star game.

```
# Experience:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = experience)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Experience in NBA",
         title = "Experience as a Predictor of All-Star") +
    theme_minimal()
```



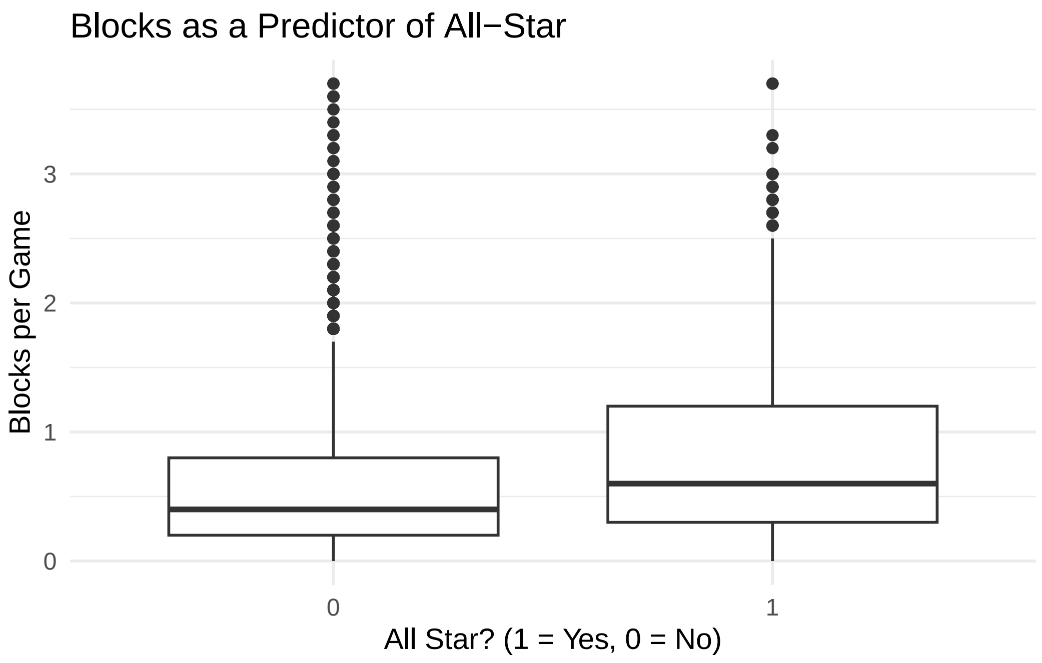
Experience does not appear to be a significant predictor of whether somebody makes the all-star game.

```r
# 3 pointers made per game:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = x3p_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "3 Pointers Made per Game",
         title = "3 Pointers as a Predictor of All-Star") +
    theme_minimal()
```



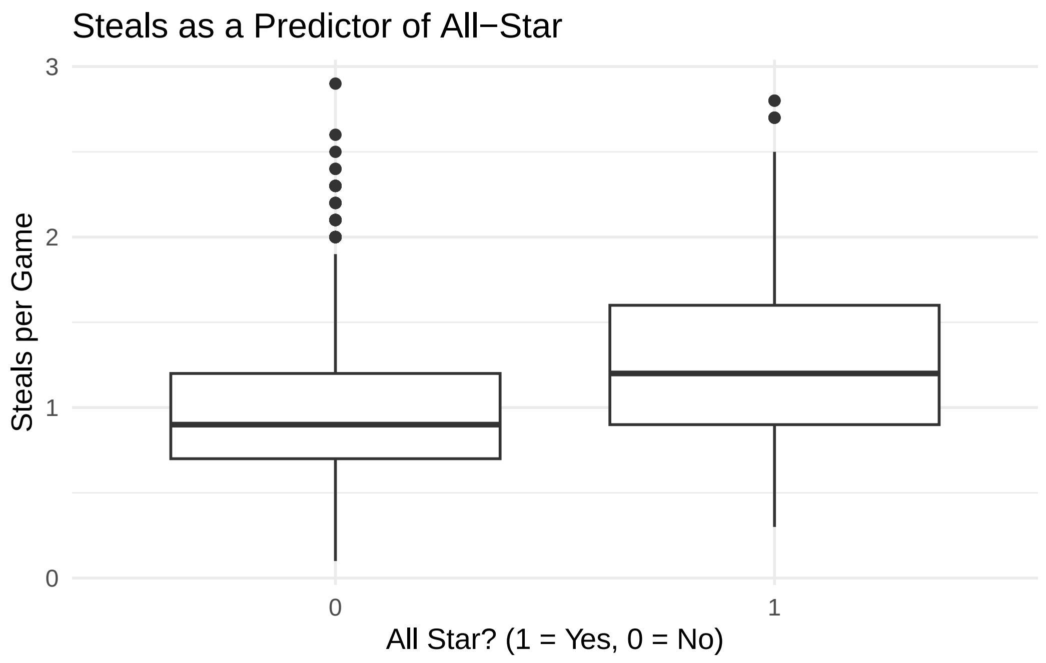
3 pointers does not seem to be very significant either, but one takeaway is that if you are making a very high amount of threes (more than 4 per game), you are almost guaranteed to be an all-star that season.

```r
# Blocks per game:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = blk_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Blocks per Game",
         title = "Blocks as a Predictor of All-Star") +
    theme_minimal()
```



Blocks appears to have a very slight positive correlation to whether or not a player was an all-star.

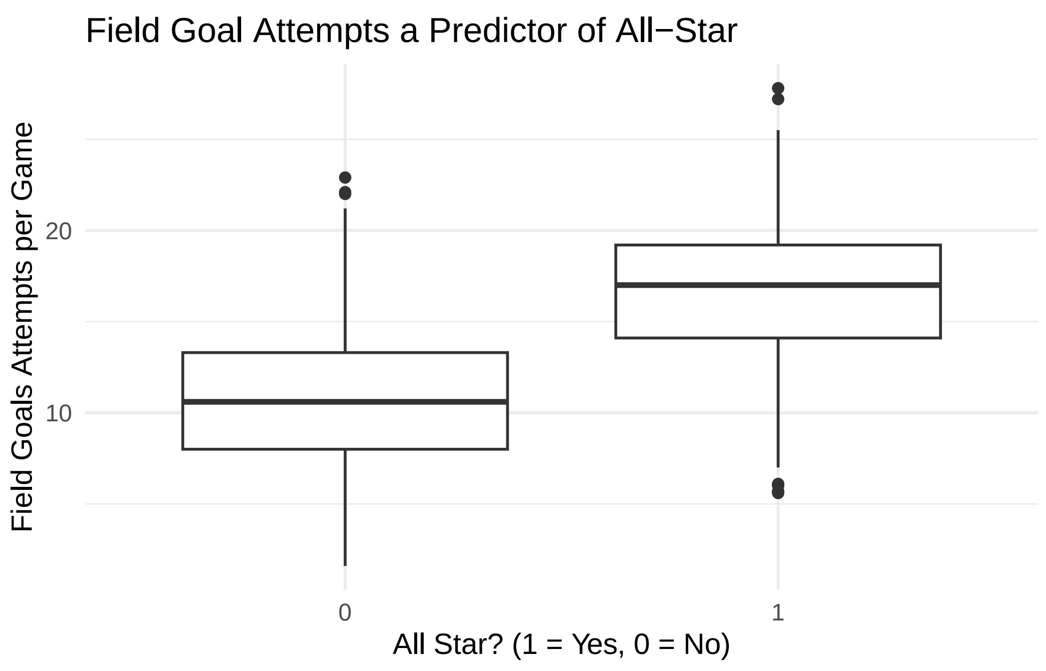
```r
# Steals per game:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = stl_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Steals per Game",
         title = "Steals as a Predictor of All-Star") +
    theme_minimal()
```



Steals appears to have the same correlation as blocks but a little more obvious. (Steals appear more significant than blocks)

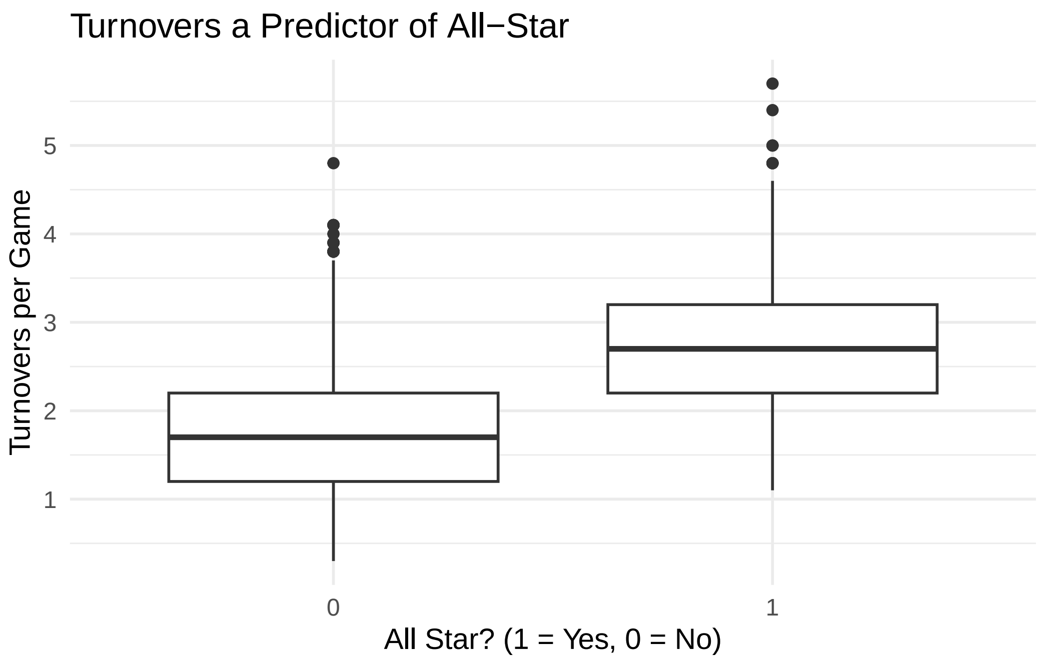
The last few variables I want to look at are attempted field goals per game - because if you are attempting a lot of shots, you are likely one of your team's trusted scorers, and turnovers per game - my thought process is that the better players have the ball more, and if you have the ball a lot of the time you are more likely to turn the ball over.

```r
# Field Goals Made per game:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = fga_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Field Goals Attempts per Game",
         title = "Field Goal Attempts a Predictor of All-Star") +
    theme_minimal()
```



Field goal attempts appears to be one of the most significant predictors I have looked at yet. (Field goals made also appears to be simiarly significicant, but would probably have to include only one field goal statistic when running the knn model).

```
# Turnovers per game:
ggplot(joined_all_star_data, aes(x = as.factor(all_star),
                                 y = tov_per_game)) +
    geom_boxplot() +
    labs(x = "All Star? (1 = Yes, 0 = No)",
         y = "Turnovers per Game",
         title = "Turnovers a Predictor of All-Star") +
    theme_minimal()
```



Turnovers also appears to be significant, just as I had hypothesized, but probably in a different way than the average fan would expect. Typically, we associate more turnovers with a player performing poorly, but clearly this is not something the voters weigh heavily when casting their all-star ballots each season. As it turns out, turnovers may be a telling statistic as to which players have the ball more often, since it appears so strongly positively correlated with making the all-star game.

## 0.9 NBA KNN Modeling in Python

Now that I have explored and visualized the data in R, I am going to perform the stat learning algorithm in Python. As I mentioned above, the statistical learning method I will be implementing is called k-nearest-neighbors, or knn. A knn algorithm predicts the value of a variable (made/missed all-star game) based on whether the k most similar data points made/missed the all-star game. It measures similar data points to be the data points that have the closest distance in each scaled predictor (closest Euclidean distance).

```
import math
from sklearn.neighbors import KNeighborsClassifier
import sklearn.metrics

# read in the data we wrangled previously
joined_all_star_data = pd.read_csv("/Users/tannerbessette/Desktop/Python_Textbook/NBA_Projec

# set the seed
np.random.seed(42)
```

Before performing the knn analysis, I am going to split the dataset into train and test data. It is important to do this so that the model does not overfit and that our classification rate at the end of the analysis is more honest/accurate. If the model was trained on the data it will be tested on, it could perform unrealistically well, because it has already seen those data points.

The rule of thumb that I used for splitting up training and test data was to include 80% of the rows in the training data, and 20% of the rows in the test data (which is a pretty typical split).

Create train and test data by splitting up the joined_all_star_data:

```
# .sample in Python appears to be the same as slice_sample in R
train_sample = joined_all_star_data.sample(n = 2852)

# below is what we are using instead of an anti-join
# basically does a left join but creates a merge variable that
# indicates whether a row was in both original datasets, the left,
# or the right. Only keep in left, and remove the variable afterwards

columns_to_join_on = joined_all_star_data.columns.tolist()
test_sample = joined_all_star_data.merge(train_sample, on=columns_to_join_on, how='left', inc

test_sample = test_sample[test_sample['_merge'] == 'left_only'].drop(columns=['_merge'])
```

I chose predictors based on whichever ones looked most correlated with being selected for the all-star game from the exploratory plots. These predictors were points per game, rebounds per game, assists per game, steals per game, field goal attempts per game, and turnovers per game.

Scaling is important, because we don't want to give unequal weight to our different predictors.

Create a data frame that only has the predictors we will use, and scale all predictors:

```
# Package to scale the variables:
from sklearn.preprocessing import StandardScaler

# to specify the columns to use for knn model, create a list, then
# select just those columns from train and test datasets
columns_to_keep = ['pts_per_game', 'trb_per_game', 'ast_per_game',
                     'stl_per_game', 'fga_per_game', 'tov_per_game']

train_small = train_sample[columns_to_keep]
test_small = test_sample[columns_to_keep]

# Scale the predictors using StandardScaler()

# Initialize the scaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform both train and test data
train_small = scaler.fit_transform(train_small)
test_small = scaler.transform(test_small)
```

Put the all-star response into a vector:

```
# Create vectors:
train_cat = train_sample['all_star']
test_cat = test_sample['all_star']
```

Below, I calculate a number of nearest neighbors that I will use in my knn model by finding the square root of the size of my training dataset. This is a general good starting point, but not the definitive best k to choose. However, for this model any other value of k that I chose was either very similar or worse, so I decided to stick with the square root of the training dataset size for k.

Calculate a reasonable number of nearest neighbors ($\sqrt{2852}$):

```
# Find a good number of nearest neighbors (sqrt of train size):
math.sqrt(2852) # use 53 nearest neighbors
```

```
53.40411969127476
```

The code chunk below utilizes the scikit-learn library to fit a knn model. The KNeighborsClassifier specifies how many neighbors we want to utilize in our model. Then, I use `.fit()` and `.predict()` to fit and predict, respectively.

Fit the KNN model:

```
# Specify 53 neighbors:
knn = KNeighborsClassifier(n_neighbors = 53)

# Fit the knn model
knn.fit(train_small, train_cat)
```

```
KNeighborsClassifier(n_neighbors=53)
```

```
# Make predictions for test data
knn_mod = knn.predict(test_small)
```

A confusion matrix is often used to evaluate a knn model's performance. It shows the true positives (top left), true negatives (bottom right), false positives (bottom left), and false negatives (top right). The classification rate gives us the percentage of observations from the test data that our model correctly predicted as either an all-star or a non-all-star.

Here, I create a confusion matrix to analyze the predictive accuracy of the knn model. The `confusion_matrix()` function creates the confusion matrix, and the `accuracy_score()` function calculates our classification accuracy. These functions are a part of the sklearns.metrics package.

Evaluate the knn model:

```
# Create the confusion matrix
conf_matrix = sklearn.metrics.confusion_matrix(test_cat, knn_mod)

# Display the confusion matrix:
print(conf_matrix)
```

```
[[561  12]
 [ 78  62]]
```

The rows of the table give us our model's predicted all_star choice (made it or didn't make it), and the column's give the actual result.

So, in the above confusion matrix, the top left number, 559, gives us the number of non-all-stars that our knn model correctly predicted as non-all-stars, while the bottom right number, 70, gives us the number of all-stars that our model correctly predicted as all-stars. The bottom left number, 70, gives the number of non-all-stars that our model predicted to actually be all-stars, while the top right number, 14, gives us the number of players that our model predicted to not be all-stars but were actually all-stars.

Obtain the predictive accuracy of the knn model:

```
# Obtain the classification rate from the confusion matrix:
accuracy = sklearn.metrics.accuracy_score(test_cat, knn_mod)

# Print the classification rate:
print(round(accuracy, 3))
```

```
0.874
```

Our classification rate of 87.4% tells us that our knn model predicted whether or not players made the all-star game with an accuracy of 87.4%!

**Potential Future Directions:** With more time, it may be of interest to investigate which players the model predicted to be an all-star but actually weren't selected to be an all-star and why. It will also be cool to test out this model on this year's NBA season at the conclusion of the season, to see how accurately it predicts. We can also use this model to see if the same players that fans believe to be “snubbed” from the all-star game were players that the model predicted to be NBA all-stars but did not get selected.

## 0.10 Conclusion

From these two projects, in my opinion, R wrangling code is a bit easier to learn and is a bit more intuitive. It is worth noting that most wrangling functions share an analogue in each language, such as `filter()` in R $\rightarrow$ `.query()` in Python, `group_by()` in R $\rightarrow$ `.groupby()` in Python, `%in%` in R $\rightarrow$ `.isin()` in Python. Despite my personal preference of R, both languages are more than capable of wrangling and visualizing data.

There are a few caveats to the project that are worth noting. Firstly, the fact that I learned R first, and used it in my research fellowship, internship, and in many courses throughout my time at St. Lawrence University likely has an impact on my favorism of the langauge. Additionally, I did not look at machine learning and did not look at big data throughout this project, both of which are considered “strengths” of Python. Finally, this project was not exactly comparing R and Python, it was essentially comparing the `tidyverse` R package to the `pandas`/`plotnine`/`numpy` packages in Python.