Call of Duty Examples

Matt Slifko (mds6457@psu.edu (mailto:mds6457@psu.edu))

June 1, 2023

Preliminaries

```
#Delete all objects in Environment
remove(list = ls())

#Load libraries
library(tidyverse)

#Read datasets into Environment
CODGames1 <- read.csv(file = "https://raw.githubusercontent.com/matthewdslifko/CallOfDut
yProject/main/Data/CODGames1SP.csv", header = TRUE)
CODGames2 <- read.csv(file = "https://raw.githubusercontent.com/matthewdslifko/CallOfDut
yProject/main/Data/CODGames2SP.csv", header = TRUE)
GameModes <- read.csv(file = "https://raw.githubusercontent.com/matthewdslifko/CallOfDut
yProject/main/Data/CODGameModes.csv", header = TRUE)
Maps <- read.csv(file = "https://raw.githubusercontent.com/matthewdslifko/CallOfDutyProj
ect/main/Data/CODMaps.csv", header = TRUE)
Weapons <- read.csv(file = "https://raw.githubusercontent.com/matthewdslifko/CallOfDutyP
roject/main/Data/CODWeapons.csv", header = TRUE)
```

Example 1 - Exploratory Data Analysis (EDA) Process and Processing Character Data

Discussion Opportunities

This example provides an opportunity to discuss the following ideas:

- data visualization
- the process of data exploration
- complementary roles of visualization and numerical summaries
- annotating plots with text
- character string processing and new variable creation
- multivariable thinking

Background Information

Begin by providing students with some information about the variable of interest: The variable `Damage` represents the amount of damage issued by the player on the opposing team's players, weapons, and vehicles.

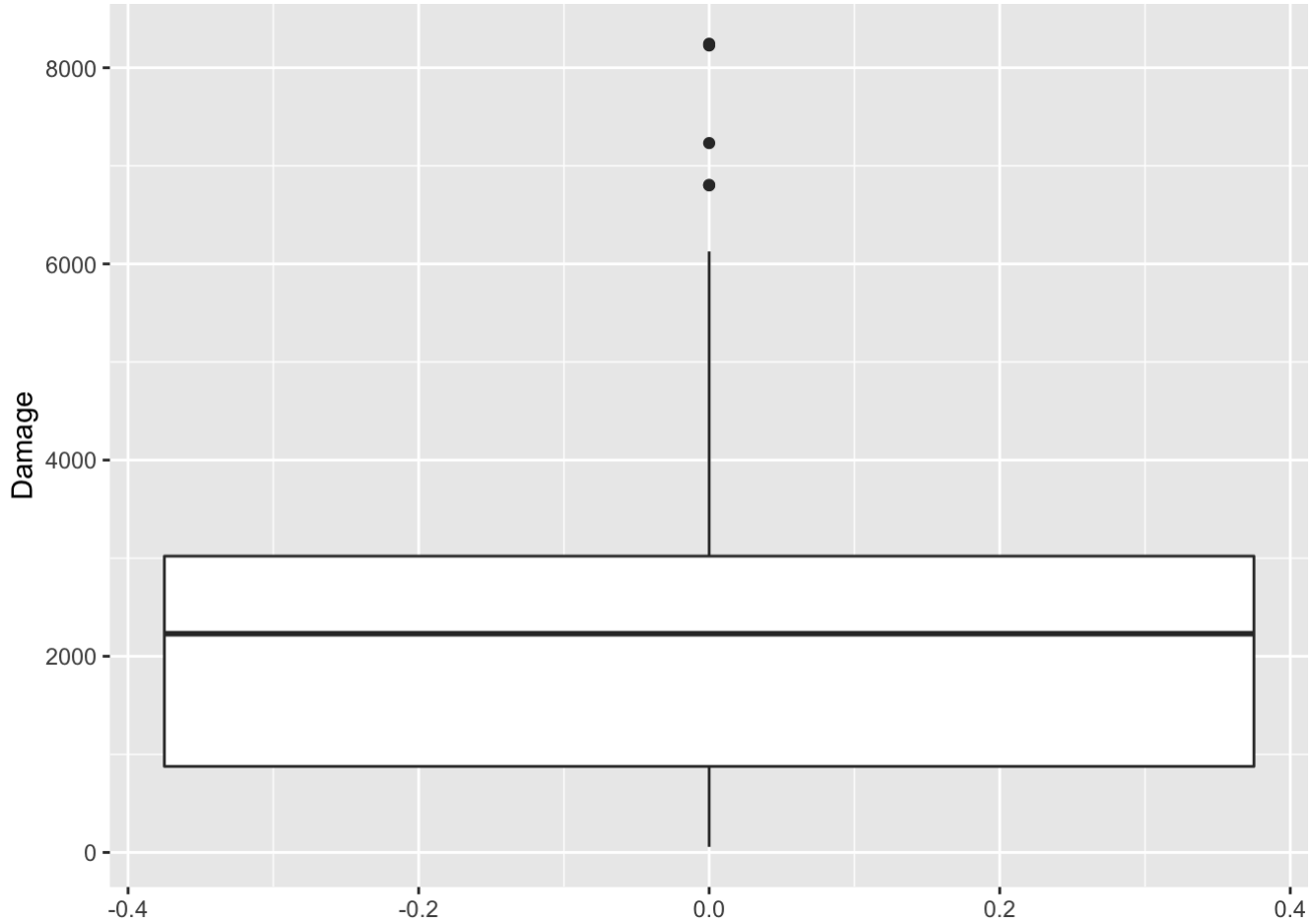
Initial Task and Exploration

Instruction for students: Explore the distribution of damage in `CODGames2` .

Initial Exploration: Students will recognize the variable of interest, `Damage`, as quantitative. This may prompt students to create a boxplot, histogram, or some other type of plot that is appropriate for exploring the distribution of a single quantitative variable. We start with a boxplot.

```
#Create Boxplot
ggplot(data = CODGames2, mapping = aes(y = Damage)) +
  geom_boxplot() +
  labs(y = "Damage")
```

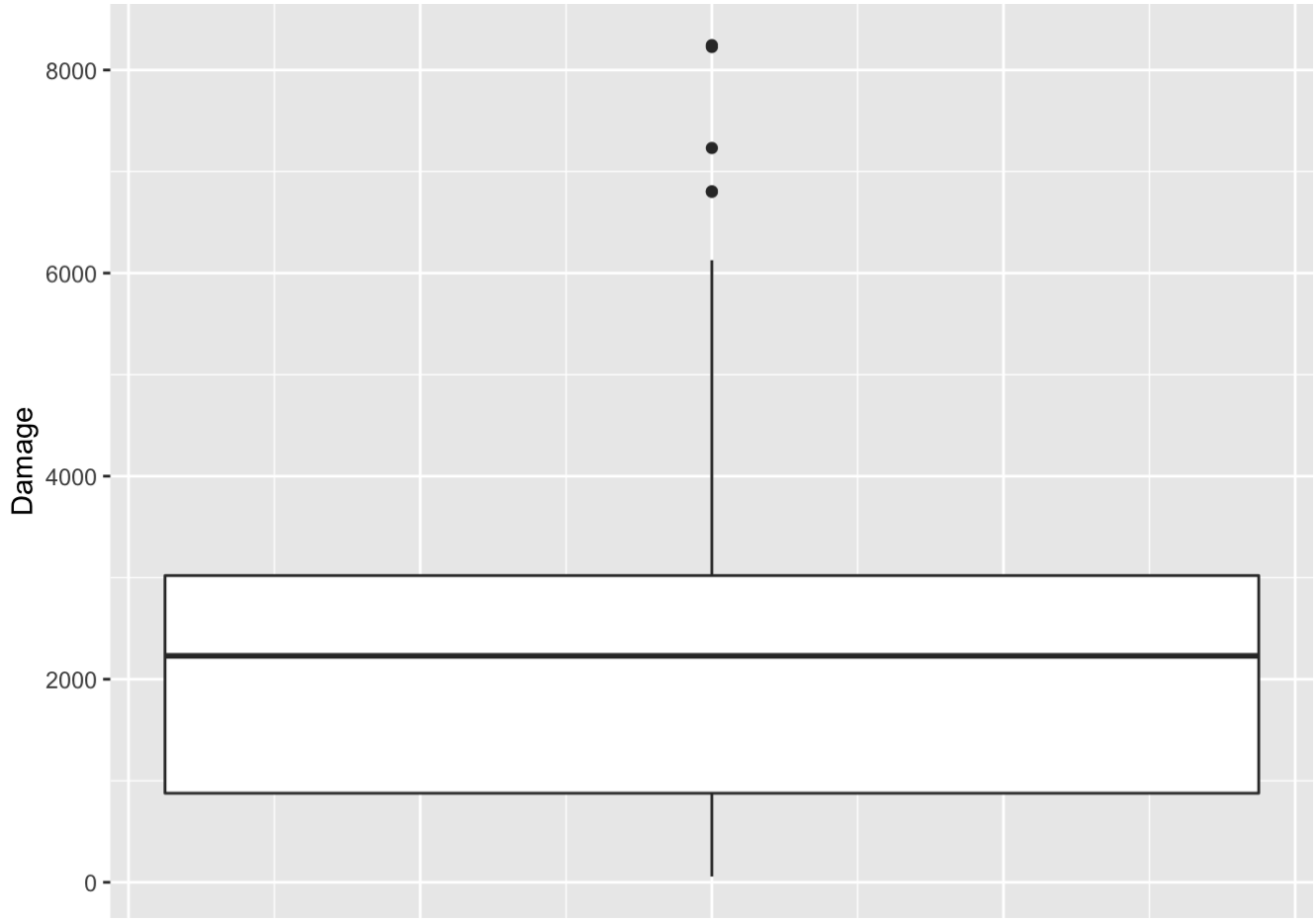
```
## Warning: Removed 164 rows containing non-finite values (stat_boxplot).
```



I like to take this opportunity to remind students that the x-axis labels on this boxplot are not helpful and should be suppressed. As the code required to do this may not have been covered in class, I might suggest that students try a Google search such as “remove x axis boxplot ggplot r”. One solution is shown below.

```
#Create boxplot of Damage with meaningless axis suppressed
ggplot(data = CODGames2, mapping = aes(y = Damage)) +
  geom_boxplot() +
  labs(y = "Damage") +
  theme(axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```

```
## Warning: Removed 164 rows containing non-finite values (stat_boxplot).
```

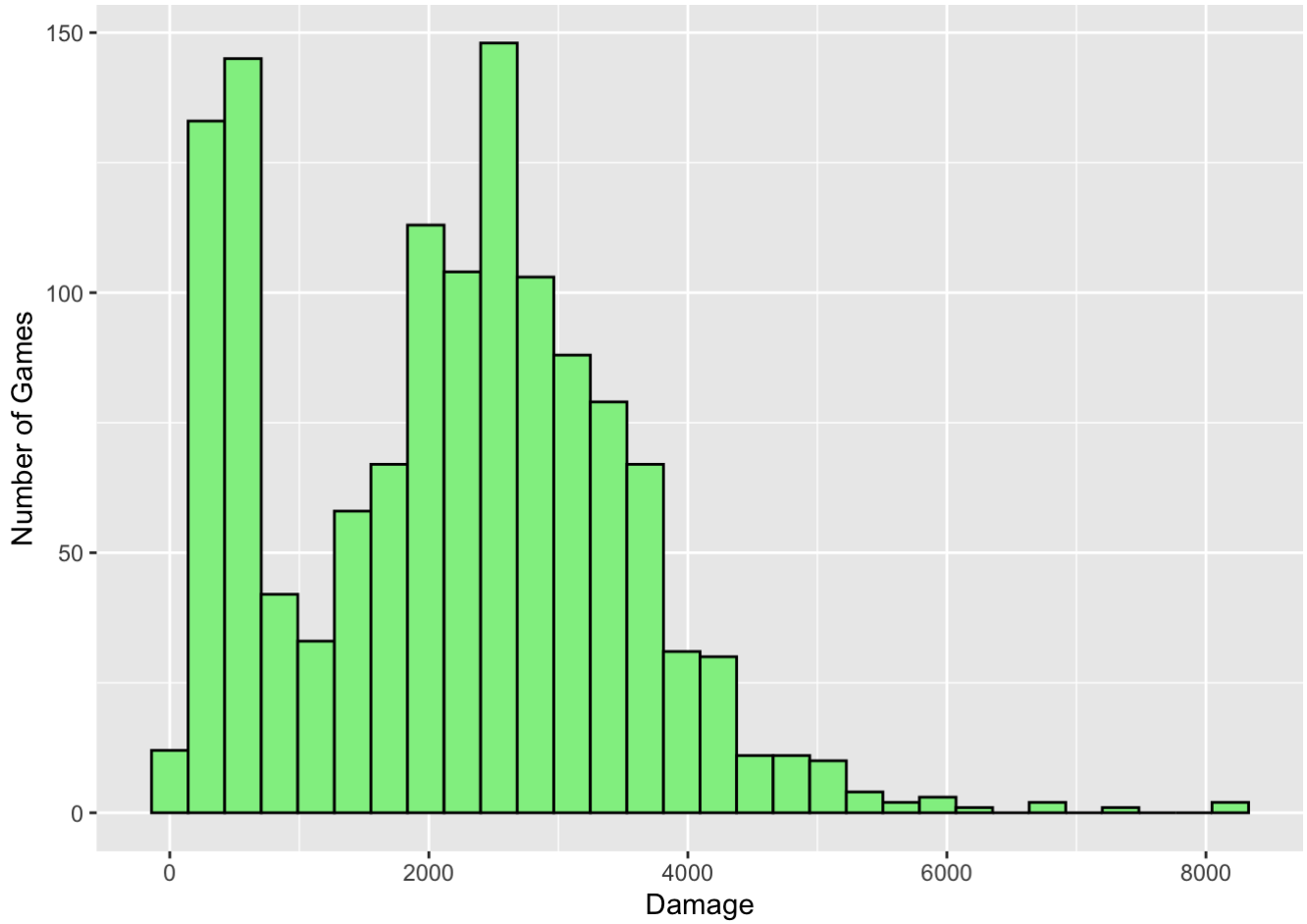


Since a boxplot reduces the distribution to 5 numbers (and outliers), we might not get a full picture of the distribution based on the boxplot. A histogram or violin plot might provide more detail.

```
#Create Histogram
ggplot(data = CODGames2, mapping = aes(x = Damage)) +
  geom_histogram(fill = "lightgreen", color = "black") +
  labs(x = "Damage",
       y = "Number of Games")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 164 rows containing non-finite values (stat_bin).
```



Important Discussion Point: The end goal of EDA is not to create the visualization. Instead we want to learn something about the variable(s). In this case, we can see that the distribution is clearly bimodal with some outliers on the high end of the distribution.

(Optional) For the sake presentations, I like to remind students that it is often helpful to annotate plots. For instance, I might want to call attention to the 2 highest bins. One way to accomplish this is demonstrated below. The annotate commands may require some trial and error when determining coordinates.

```
#Store the histogram as an object
histPlot <- ggplot(data = CODGames2, mapping = aes(x = Damage)) +
  geom_histogram(fill = "lightgreen", color = "black") +
  labs(x = "Damage",
       y = "Number of Games")

#Extract data used to create the histogram
histDat <- layer_data(histPlot)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

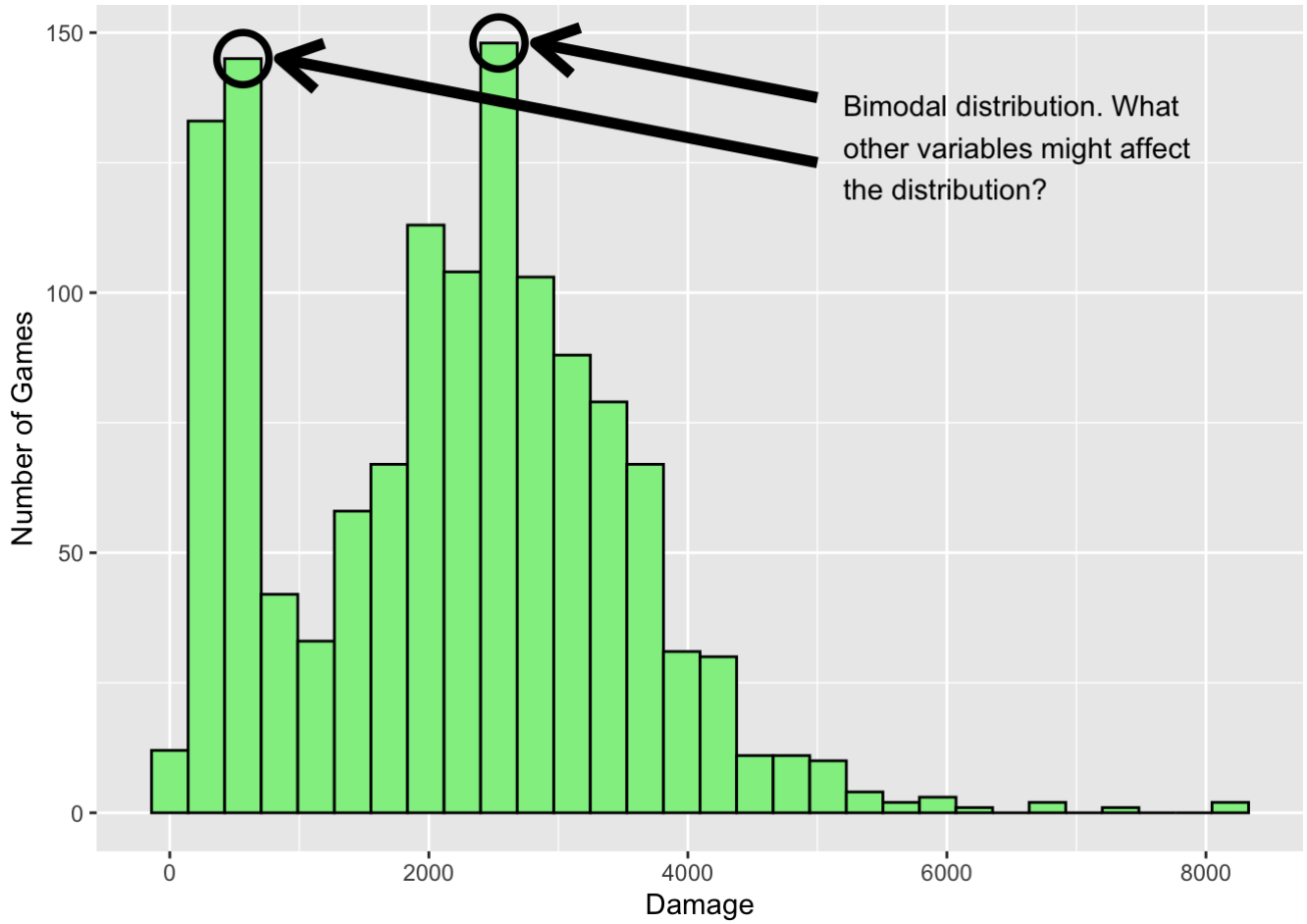
```
## Warning: Removed 164 rows containing non-finite values (stat_bin).
```

```
#Sort the bins by the count variable and select the 2 highest counts
histDatModes <-
  histDat %>%
  arrange(desc(count)) %>%
  head(2)

#Recreate the histogram; add open circles (shape = 1) to callout modes (stroke controls
thickness of of circle borders) and annotate with line segments
ggplot(data = CODGames2, mapping = aes(x = Damage)) +
  geom_histogram(fill = "lightgreen", color = "black") +
  labs(x = "Damage",
       y = "Number of Games") +
  geom_point(data = histDatModes, mapping = aes(x = x, y = count),
             shape = 1, size = 8, stroke = 2) +
  annotate("segment", x = 5000, xend = 2824, y = 137.5, yend = 148,
           color = "black", size = 2, arrow = arrow()) +
  annotate("segment", x = 5000, xend = 847, y = 125, yend = 145,
           color = "black", size = 2, arrow = arrow()) +
  annotate("text", x = 5200, y = 128,
           label = "Bimodal distribution. What \nother variables might affect \nthe dist
ribution?",
           hjust = 0)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 164 rows containing non-finite values (stat_bin).
```



At this time, I might ask students to think about other variables, either included in the dataset or not, might affect the distribution of `Damage`. Students often respond by suggesting that the primary weapon (`PrimaryWeapon`) might have an impact since different weapons do different amounts of damage. This is a good answer, but there is a better answer. (See next sections.)

Further Background

After students have considered other variables that could impact the distribution of `Damage`, I will share the following background information: Some values in the `GameType` variable contain an "HC" designation. In the lingo of the game, this is meant to distinguish "Core" games from "Hardcore" games. Unlike Core games, players start with less health and health does not regenerate. Thus, there is less opportunity to deal damage in "HC" game modes.

Revised Task and Exploration

Instruction for students: Explore the distribution of damage after accounting for whether the game is Core or HardCore.

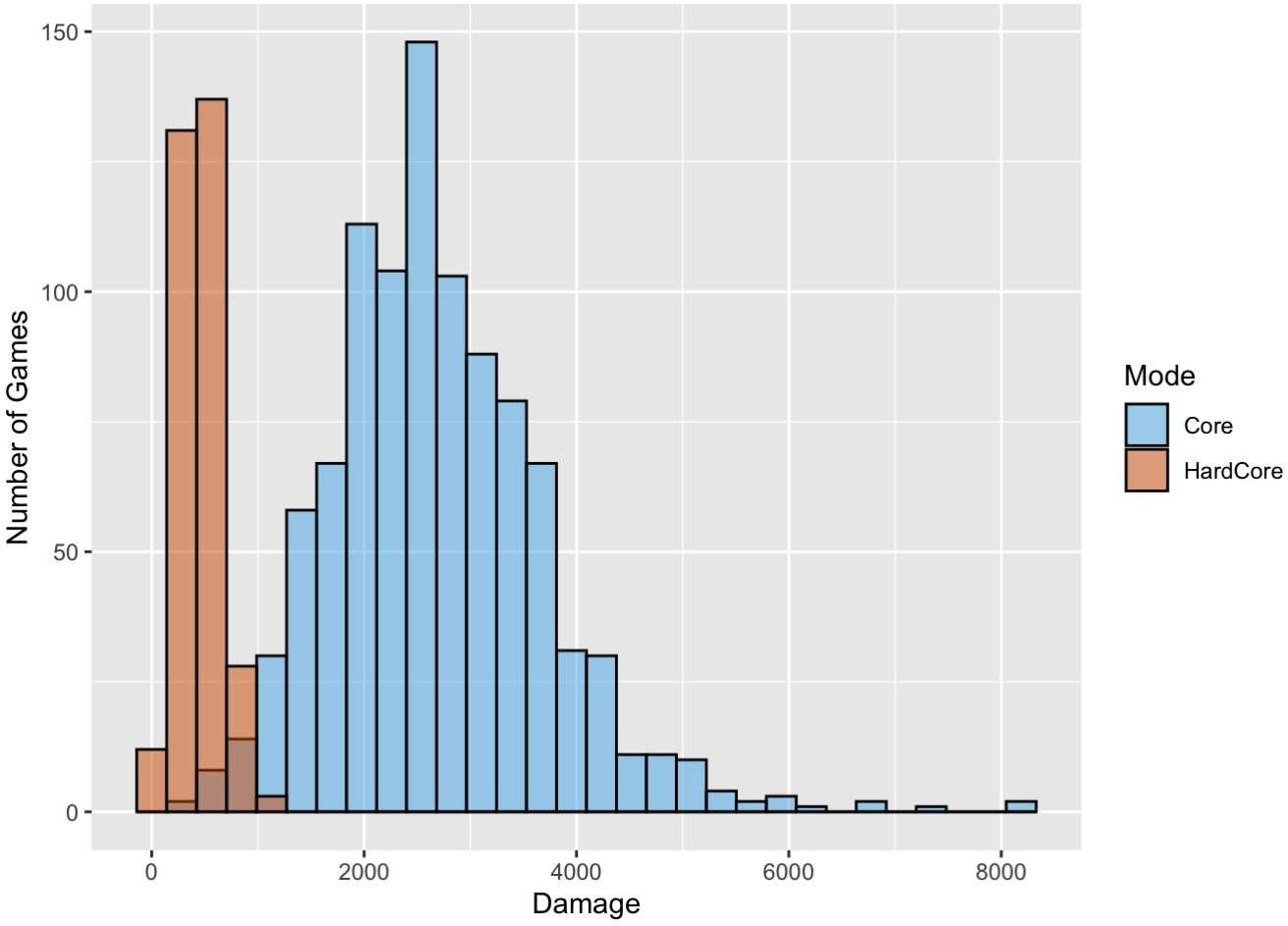
Revised Exploration: One approach for this situation is to create a new variable called `Mode`. The code below handles this by using the `grepl()` function to determine if `GameType` contains "HC" and incorporate in histogram.

```
#Create Variable named Mode that takes on values of Core and Hardcore
CODGames2 <-
  CODGames2 %>%
  mutate(Mode = ifelse(grepl("HC - ", GameType, ignore.case = FALSE), "HardCore", "Cor
e"))

#Create Histogram of Damage Incorporating Core
ggplot(data = CODGames2, mapping = aes(x = Damage, fill = Mode)) +
  geom_histogram(color = "black", alpha = 0.5, position = "identity") +
  scale_fill_manual(values=c("#54B6E9", "#D55E00")) +
  labs(y = "Number of Games",
       fill = "Mode")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 164 rows containing non-finite values (stat_bin).
```



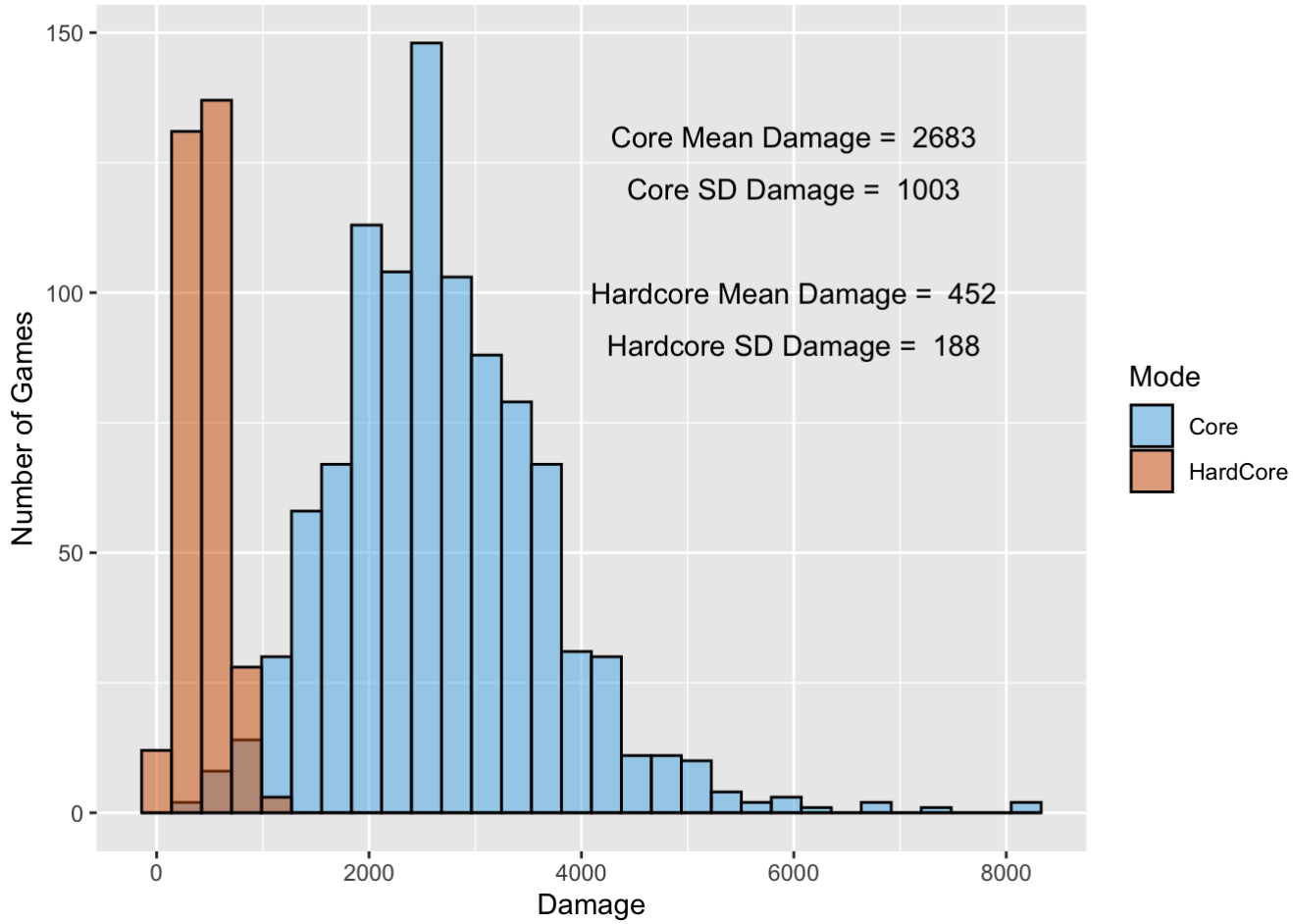
Based on this plot, it is clear that `Damage` is a mixture of 2 distributions. The larger values of `Damage` occur during the `Core` games, while the `HardCore` games correspond to the lower values. To support this, we can calculate some summary statistics and add them to the plot.

```
#Calculate summary stats of Damage for each level of Mode
DamageSummary <-
  CODGames2 %>%
  group_by(Mode) %>%
  summarize(Mean = round(mean(Damage, na.rm = TRUE), digits = 0),
            StdDev = round(sd(Damage, na.rm = TRUE), digits =0))

#Create Histogram of Damage Incorporating Mode and add summary stats
ggplot(data = CODGames2, mapping = aes(x = Damage, fill = Mode)) +
  geom_histogram(color = "black", alpha = 0.5, position = "identity") +
  scale_fill_manual(values=c("#54B6E9", "#D55E00")) +
  labs(y = "Number of Games",
       fill = "Mode") +
  annotate("text", x = 6000, y = 130,
           label= paste("Core Mean Damage = ", DamageSummary$Mean[1])) +
  annotate("text", x = 6000, y = 120,
           label= paste("Core SD Damage = ", DamageSummary$StdDev[1]))+
  annotate("text", x = 6000, y = 100,
           label= paste("Hardcore Mean Damage = ", DamageSummary$Mean[2])) +
  annotate("text", x = 6000, y = 90,
           label= paste("Hardcore SD Damage = ", DamageSummary$StdDev[2]))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 164 rows containing non-finite values (stat_bin).
```



Summary

This example provides an opportunity to discuss a number of useful R functions and ideas, while presenting statistical thinking as a process of exploration, decision making, and multivariable reasoning.

Example 2 - Processing Character Data, Logical Errors/Importance of Checking Calculations, and Investigating Warnings

Discussion Opportunities

This example provides an opportunity to discuss the following ideas:

- character string processing and new variable creation
- if/else with more than 2 conditions
- importance of manually checking calculations
- data visualization and reordering factors for visualization
- exploring WARNING messages
- BONUS: discuss whether there is a difference between proportions of wins and losses OR is the difference just due to random variation?

Background Information

Each online match consists of two competing teams that earn points for various tasks. The `Result` variable shows the team scores at the end of the match. The player's team score is listed first and the team with the higher score is considered the winner. Using `CODGames1`, some examples:

- Row 1: The character string "100-97" means that the player's team won by a score of 100 to 97.
- Row 2: The character string "76-89" means that the player's team lost by a score of 89 to 76.
- Row 36: The character string "200-200" means that the teams played to a draw, or tied 200 to 200.

Initial Task and Exploration

Instruction for students: Answer the following research question using `CODGames1`: Is the player's team more likely to win, lose, or draw?

Initial exploration: If this is task is taught during the exploratory data analysis phase, I would expect students to support their answer both visually and numerically. I also emphasize that when calculating proportions, the code should work for any dataset, not just this specific dataset. (In other words, students should not use "/235" since 235 is the specific number of rows in this dataset.) The challenge is that the character string `Result` is not currently in a usable form. I would then prompt students to brainstorm how we can convert `Result` to something that is more useful to us.

The key for this dataset is to recognize that a hyphen ("-") separates the player's team score from the opposing team score. So, we need to split the `Result` into two scores based on the hyphen. There are many ways to do this, but the `separate()` function from the `tidyr` package (part of `tidyverse`) is one potential solution. In the solution shown below, I also used the `case_when()` function to illustrate how to handle an if/else with more than 2 conditions. (A nested `ifelse()` command is another viable approach.)

```
#Separate Result into 2 new variables (PlayerTeamScore and OpponentScore); then create M
atchResult to determine if player's team won, lost, or played to a draw
Scores <- CODGames1 %>%
  separate(Result, into=c("PlayerTeamScore", "OpponentScore"),
           sep="-", remove=FALSE) %>%
  mutate(MatchResult = case_when(PlayerTeamScore < OpponentScore ~ "Loss",
                                 PlayerTeamScore > OpponentScore ~ "Win",
                                 PlayerTeamScore == OpponentScore ~ "Draw"))
```

Important Discussion Point: Just because your code runs without error does not mean that your code is correct. The code used above has an intentional mistake. The code runs without error, or warnings, but there is a logical error affecting the results. You can see the mistake by exploring the first few rows the `Scores` dataset.

```
Scores %>%
  select(Result, PlayerTeamScore, OpponentScore, MatchResult) %>%
  head()
```

```
##   Result PlayerTeamScore OpponentScore MatchResult
## 1 100-97             100            97        Loss
## 2  76-89              76            89        Loss
## 3 100-92             100            92        Loss
## 4 80-100              80           100         Win
## 5 71-100              71           100         Win
## 6 85-100              85           100         Win
```

Based on the first six rows, we can see that the `MatchResult` values are seemingly incorrect for rows 1, 3, 4, 5, and 6. This prompts an important discussion as to the reason for the incorrect results.

The key is that `PlayerTeamScore` and `OpponentScore` are still character strings. The greater than and less than comparisons are based on the alphabetical orderings. To clarify, consider Row 1. Since the "1" in "100" comes before the "9" in "97", "100" is viewed as alphabetically first (or smaller) than "97". This means that `PlayerTeamScore < OpponentScore`, making this a loss.

One possible solution is to convert these variables to numeric before making the comparison as shown below.

```
#Separate Result into 2 new variables (PlayerTeamScore and OpponentScore) and convert to
numeric; then create MatchResult to determine if player's team won, lost, or played to a
draw
Scores <- CODGames1 %>%
  separate(Result, into=c("PlayerTeamScore", "OpponentScore"), sep="-", remove=FALSE) %
>%
  mutate(PlayerTeamScore = as.numeric(PlayerTeamScore),
         OpponentScore = as.numeric(OpponentScore),
         MatchResult = case_when(PlayerTeamScore < OpponentScore ~ "Loss",
                                 PlayerTeamScore > OpponentScore ~ "Win",
                                 PlayerTeamScore == OpponentScore ~ "Draw"))

Scores %>%
  select(Result, PlayerTeamScore, OpponentScore, MatchResult) %>%
  head()
```

```
##   Result PlayerTeamScore OpponentScore MatchResult
## 1 100-97             100            97         Win
## 2  76-89              76            89        Loss
## 3 100-92             100            92         Win
## 4 80-100              80           100        Loss
## 5 71-100              71           100        Loss
## 6 85-100              85           100        Loss
```

Now that `Result` has been converted to useful information, we can calculate proportions and visualize the results. (NOTE: Although it makes creating the plot more challenging, I would recommend exploring the proportions rather than the counts of Wins, Losses, and Draws. The main reason for this is that I sometimes tell students that the data in `CODGames2` corresponds to a different player who played many more games. Comparing counts between the players is not as fair as comparing proportions.)

First, determine the counts for Win, Loss, and Draw.

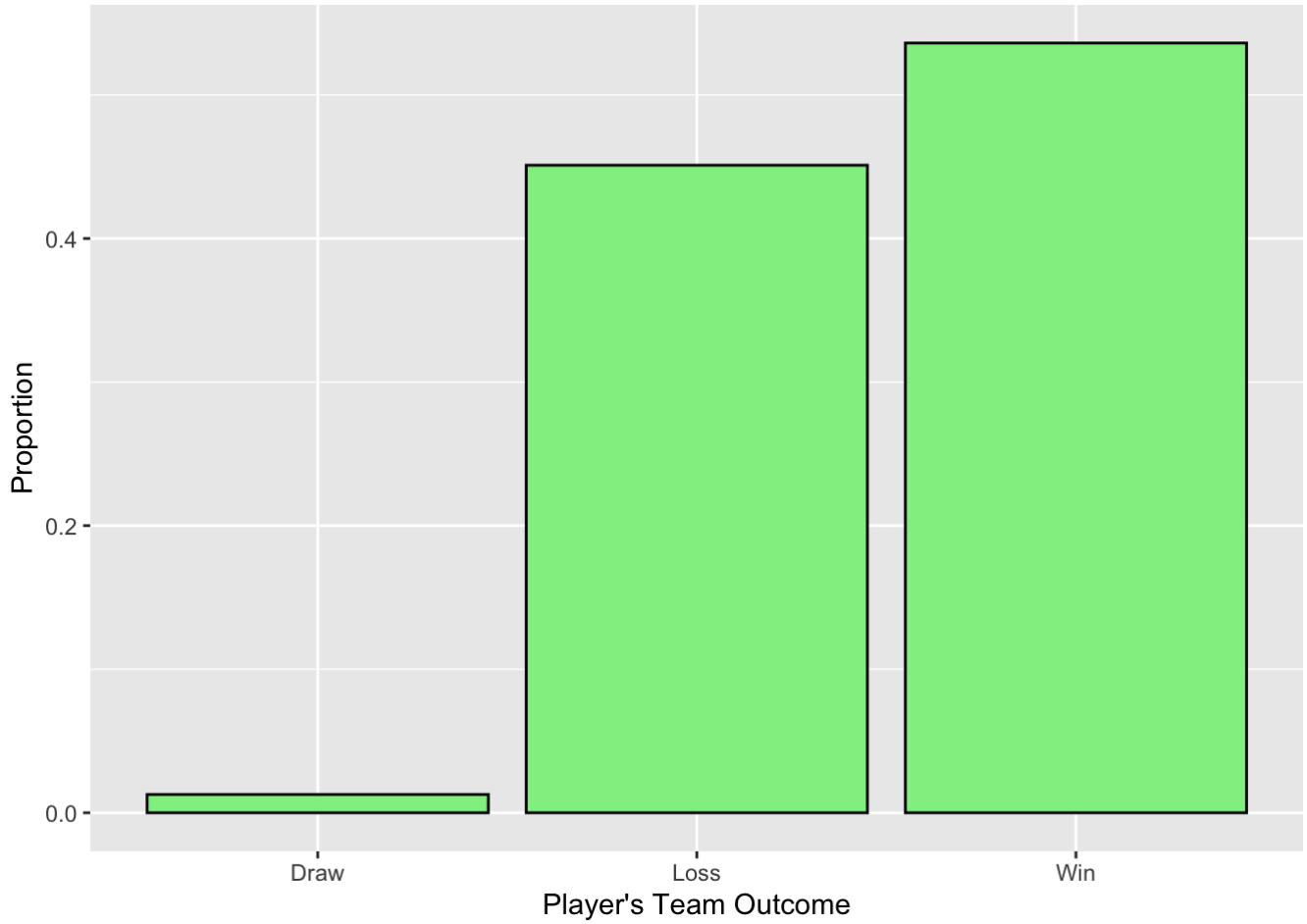
```
#Calculate Counts and Proportions for Win, Loss, Draw
MatchSummary <-
  Scores %>%
  group_by(MatchResult) %>%
  summarize(Count = n()) %>%
  mutate(Proportion = Count/sum(Count))

MatchSummary
```

```
## # A tibble: 3 × 3
##   MatchResult Count Proportion
##   <chr>       <int>      <dbl>
## 1 Draw            3     0.0128
## 2 Loss          106     0.451
## 3 Win           126     0.536
```

We can use the `MatchSummary` data frame to visualize the results as shown below:

```
#Create a plot showing the proportions
ggplot(data = MatchSummary, mapping = aes(x = MatchResult, y = Proportion)) +
  geom_bar(stat = "identity", fill = "lightgreen", color = "black") +
  labs(x = "Player's Team Outcome")
```

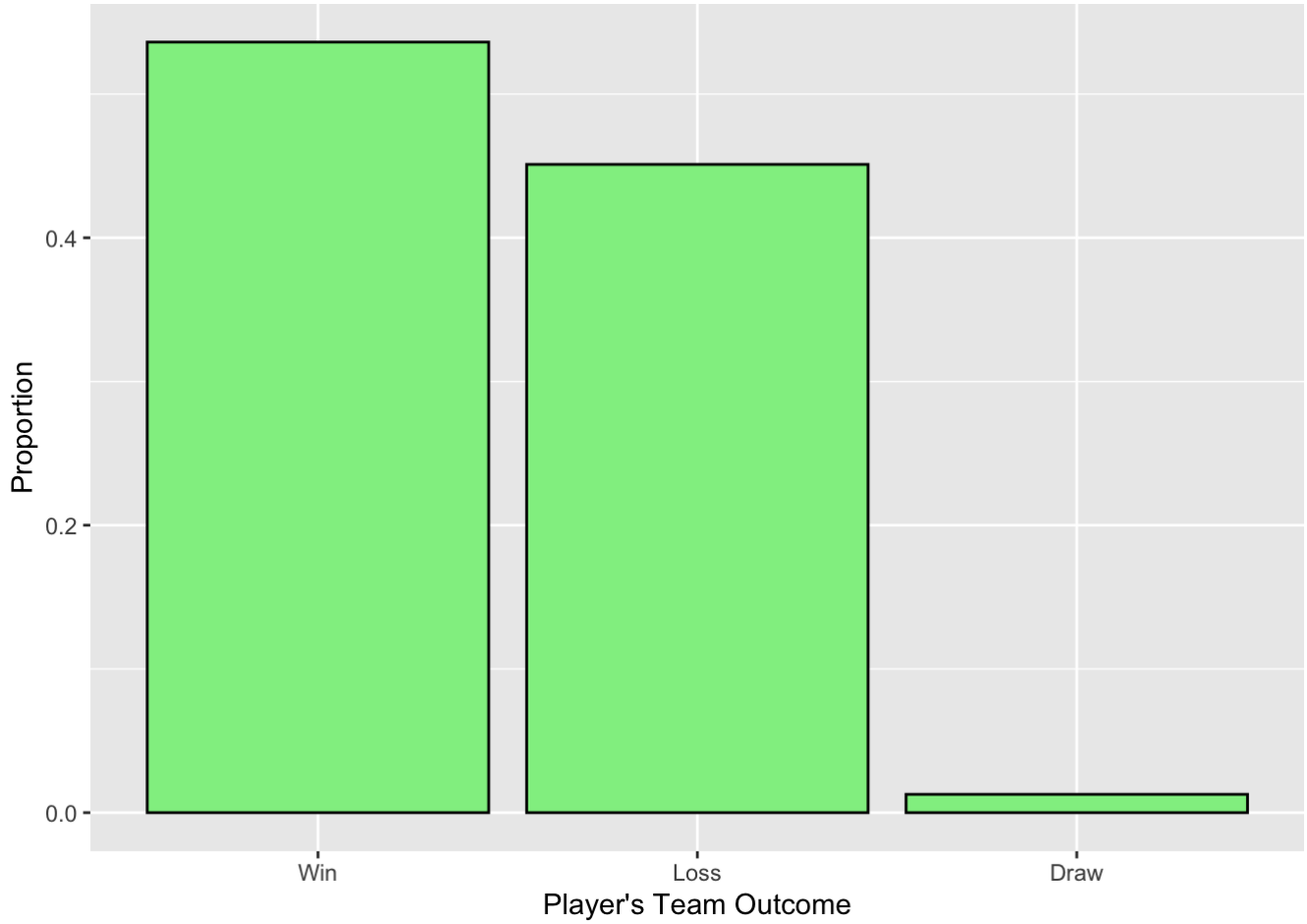


In many instances, we may want to reorder the categories on the x-axis. For instance, in this case, we might want reorder the bars in descending order based on Proportion or Count. I might ask students to spend a few minutes Google searching for how this might be accomplished.

This can be done many ways such as using the `reorder()` function from the `stats` package or `fct_reorder()` from the `forcats` library. (`fct_reorder()` is loaded as part of the `tidyverse` and is useful for working with factors.)

```
#Reorder levels of MatchResult variable using descending values of Proportion
MatchSummary <-
  MatchSummary %>%
  mutate(MatchResult = fct_reorder(MatchResult, Proportion, .desc = TRUE))

ggplot(data = MatchSummary, mapping = aes(x = MatchResult, y = Proportion)) +
  geom_bar(stat = "identity", fill = "lightgreen", color = "black") +
  labs(x = "Player's Team Outcome")
```



Challenges on Other Gameplay Dataset (CODGames2)

The second dataset, `CODGames2` , presents some additional challenges that arise when working with data. Suppose we try to separate the `Result` variable as before using the code shown below. A warning message is produced.

```
Scores <- CODGames2 %>%
  separate(Result, into=c("PlayerTeamScore", "OpponentScore"), sep="-", remove=FALSE) %
>%
  mutate(PlayerTeamScore = as.numeric(PlayerTeamScore),
         OpponentScore = as.numeric(OpponentScore),
         MatchResult = case_when(PlayerTeamScore < OpponentScore ~ "Loss",
                                 PlayerTeamScore > OpponentScore ~ "Win",
                                 PlayerTeamScore == OpponentScore ~ "Draw"))
```

```
## Warning: Expected 2 pieces. Missing pieces filled with `NA` in 20 rows [56, 57,
## 58, 325, 326, 327, 768, 769, 770, 771, 772, 892, 893, 894, 895, 896, 897, 898,
## 899, 900].
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

Students often ignore warning messages, but this should be investigated. It is not obvious as to the reason that previously working code now produces a previously unseen warning. Have students investigate a few of the rows mentioned in he warning.

```
Scores %>%
  select(Result, PlayerTeamScore, OpponentScore, MatchResult, GameType) %>%
  filter(row_number() %in% c(56:58, 325:327))
```

```
##   Result PlayerTeamScore OpponentScore MatchResult           GameType
## 1 3 to 0              NA            NA        <NA>            Control
## 2 3 to 0              NA            NA        <NA>            Control
## 3 3 to 2              NA            NA        <NA>            Control
## 4 2 to 6              NA            NA        <NA> Search and Destroy
## 5 0 to 6              NA            NA        <NA> Search and Destroy
## 6 5 to 6              NA            NA        <NA> Search and Destroy
```

Unfortunately, the `Result` column is not always formatted to include a hyphen. We can see that certain game types (Control and Search and Destroy) have the `PlayerTeamScore` and `OpponentScore` separated by the word “to” instead of a hyphen. There are several ways to handle this such as modifying the `separate()` function to look for a hyphen or “to”. Alternatively, we could replace “to” using a `gsub()` command.

Bonus Discussion

The plot shows that the player won a higher proportion of games than they lost. Do you believe the difference is due to random variation or something else?

Summary

This example provides an opportunity to discuss statistical thinking as an investigative process. The example requires students to process character data to create new/usable features, provides an opportunity to explore the critical idea that syntax error free code may still be incorrect, allows for a discussion of many useful R functions and ideas, and presents a situation in which our working code breaks when used on a new dataset.

Example 3 - Combining Data Tables and Data Quality Issues

Discussion Opportunities

This example provides an opportunity to discuss the following ideas:

- Joining tables (with no common variable names)
- Filtering observations according to a condition
- Data visualization
- Adding means to boxplot
- Categories with small n
- Handling typographical errors in data entry

Background Information

The variable `Score` represents points earned by the player for actions completed during a match. (This `Score` variable is different than the team scores explored in the previous question.) For instance, capturing an enemy location in the Domination game type earns 200 points while eliminating an enemy in Team Death Match (TDM) earns 50 points.

The variable `FullPartial` indicates whether the player participated in the complete match (`FullPartial = Full`) or only participated in a portion of the match (`FullPartial = Partial`). There are several reasons that the player may have only participated in a partial match. First, the player may have encountered internet issues and was disconnected from the server. In such cases, the player is likely missing statistics related to the match performance. Second, the more common reason for a partial match is that the player was assigned to the game lobby after the game had already started. When the player joined an in-progress game, the player only completed a partial match.

During each match, the player selects a primary weapon to use for the match. The player's choice is listed in the `PrimaryWeapon` variable. As the game offers many weapon options and some weapons may have been used in limited circumstances, it may be helpful to explore the weapon class (e.g., shotgun, sniper rifle, melee, etc.). Information about the weapon classes are not found in the gameplay datasets. To get this information, you will have to use the `Class` variable from the `Weapons` dataset.

Initial Task and Exploration

Instruction for students: Answer the following research question: For complete matches in `CODGames1` , does the player's score depend on the weapon class used?

Initial exploration: For a question like this, I like to ask students to sketch the visualization they wish to create and outline the steps that must be taken in order to create the visualization. For this question, we must add the weapon class to the gampeplay data, select the observations associated with full/complete matches, and create a visualization.

NOTE: This problem requires students to get the weapon class (named `Class`) from the `Weapons` dataset and add it to the `CODGames1` dataset. First, check to see if the datasets have any common variable names. (They do not.) Then, we will preview the `Weapons` dataset to see how we might establish a match.

```
#Check to see if the datasets have any common variable names - They do not
intersect(colnames(CODGames1), colnames(Weapons))
```

```
## character(0)
```

```
#Preview Weapons
head(Weapons, 5)
```

```
##   Weapon         Class
## 1  AK-47 Assault Rifle
## 2    XM4 Assault Rifle
## 3 Krig 6 Assault Rifle
## 4 QBZ-83 Assault Rifle
## 5 FFAR 1 Assault Rifle
```

Since the datasets have no variable names in common, we must decide how to establish a match between the datasets. In `CODGames1` , the player's primary weapon choice was indicated in the `PrimaryWeapon` variable. In the `Weapons` dataset, the variable containing the name of specific weapons is named `Weapon` . We start by performing an appropriate join and removing the partial matches.

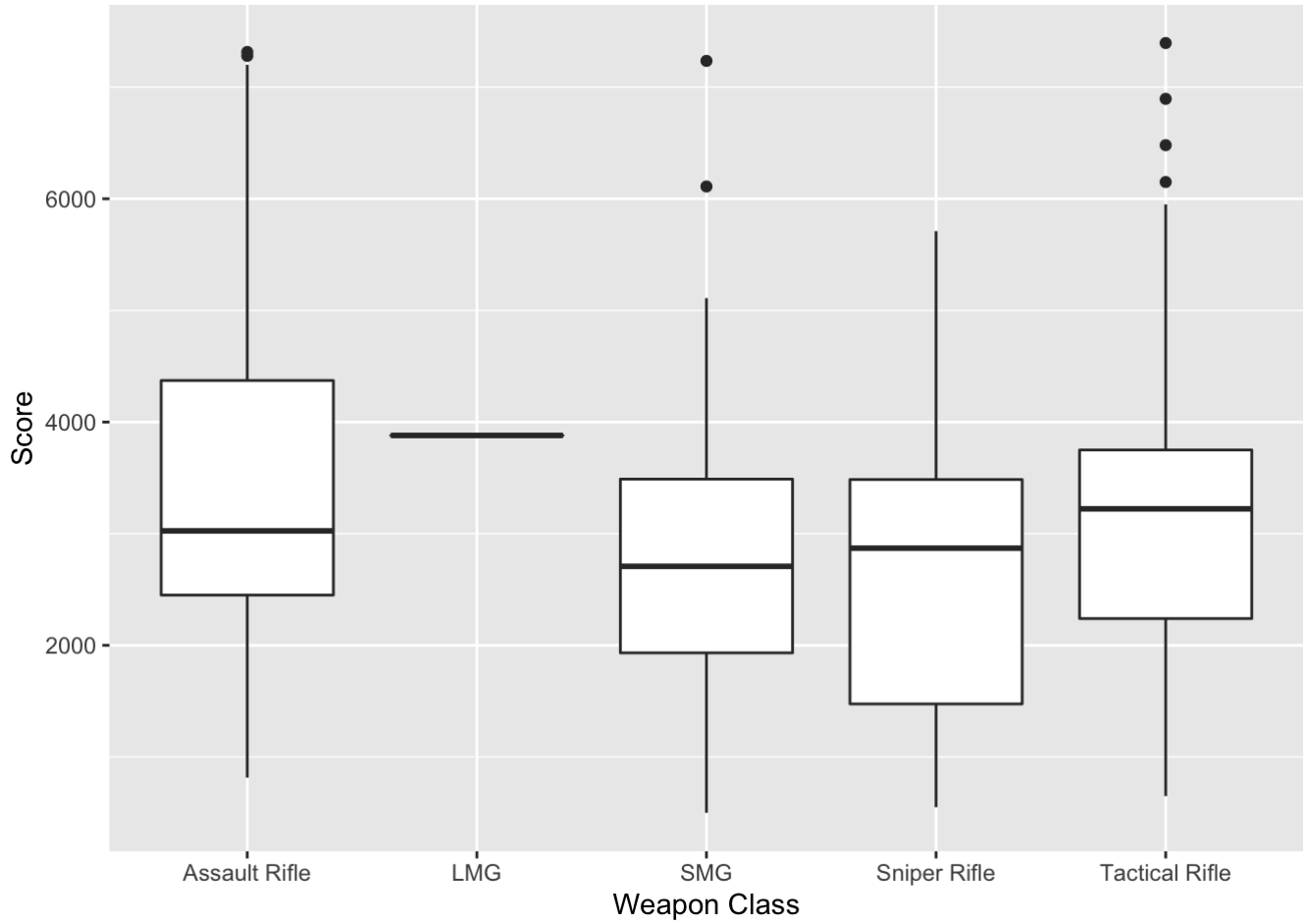
```
GamesWithClass <-
  CODGames1 %>%
  left_join(Weapons, by = c("PrimaryWeapon"="Weapon")) %>%
  filter(FullPartial == "Full") %>%
  select(Score, FullPartial, PrimaryWeapon, Class)

head(GamesWithClass)
```

```
##   Score FullPartial PrimaryWeapon          Class
## 1  4070        Full           M16 Tactical Rifle
## 2  5305        Full           M16 Tactical Rifle
## 3  3335        Full           M16 Tactical Rifle
## 4  2170        Full           M16 Tactical Rifle
## 5  2195        Full           M16 Tactical Rifle
## 6   850        Full           M16 Tactical Rifle
```

We can explore this research question by creating side-by-side boxplots that show the distribution of `Score` for each of the weapon classes as shown below.

```
#Create side-by-side boxplots for Score based on each Weapon class
ggplot(data = GamesWithClass, mapping = aes(x = Class, y = Score)) +
  geom_boxplot() +
  labs(x = "Weapon Class",
       y = "Score")
```

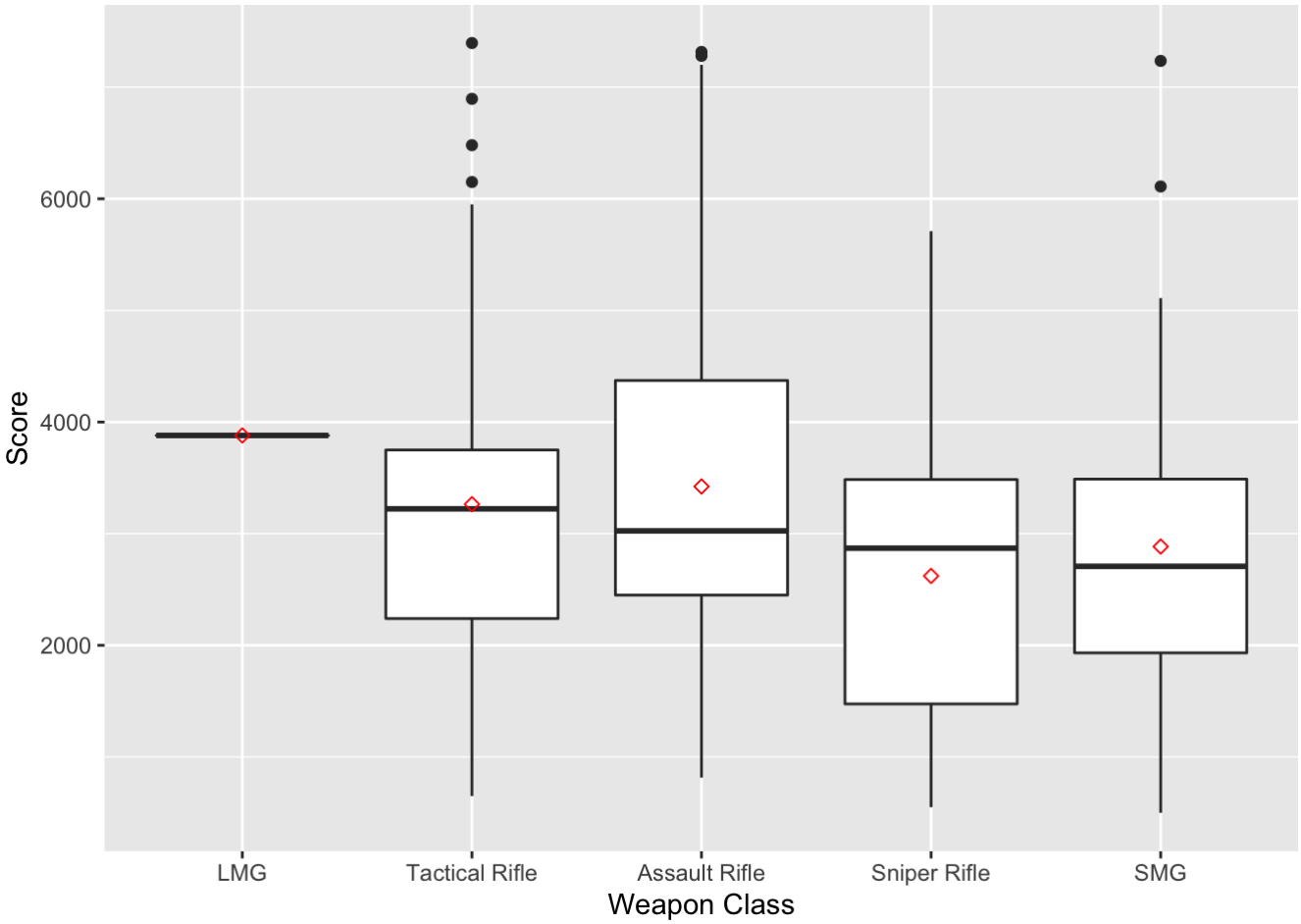


Applying tricks that were previously covered, we might want to reorder the weapon classes in descending order of the medians. Further, since students often look at boxplots and talk about the mean, which is not included by default, I will ask students to add the mean to these boxplots. To do this, create a new dataset called `ClassSummaries` that have the means for each class. Then, recreate the boxplots and add the means.

```
#Calculate summary stats for each class
ClassSummaries <-
  GamesWithClass %>%
  group_by(Class) %>%
  summarize(Games = n(),
            MeanScore = mean(Score, na.rm = TRUE),
            StdDevScore = sd(Score, na.rm = TRUE),
            MedianScore = median(Score, na.rm = TRUE))

#Reorder Weapon classes based on decreasing medians
GamesWithClass <-
  GamesWithClass %>%
  mutate(Class = fct_reorder(Class, Score, .fun = median, .desc = TRUE))

#Create Side by side boxplots with means
ggplot(data = GamesWithClass, mapping = aes(x=Class, y=Score)) +
  geom_boxplot() +
  geom_point(data = ClassSummaries, aes(y = MeanScore), shape = 5, color = "red") +
  labs(x = "Weapon Class",
       y = "Score")
```



Do you think it is fair to conclude that the LMG weapon class is associated with larger values of `Score` than the other classes? Why does the boxplot consist of a single line but lacks a box and whiskers? In the LMG class, there is only one observation.

Challenges on Other Gameplay Dataset (`CODGames2`)

The second dataset, `CODGames2` , presents some additional challenges that arise when working with data. In particular, some of the values in `PrimaryWeapon` do not match the values for `Weapon` found in the `Weapons` dataset. To illustrate this, consider the following which lists the distinct values of `PrimaryWeapon` in `CODGames2` in ascending order.

```
CODGames2 %>%
  distinct(PrimaryWeapon) %>%
  arrange(PrimaryWeapon) %>%
  head(10)
```

```
##    PrimaryWeapon
## 1          AK 47
## 2          AK-47
## 3         AK-47u
## 4            AUG
## 5   Combat Knife
## 6         DMR 14
## 7        FARA-83
## 8         FFAR 1
## 9          Gallo
## 10         Groza
```

Looking at the first few rows reveals some potential issues. There is an “AK 47” and an “AK-47”. Checking the `Weapons` dataset reveals that the proper name is “AK-47”. (“AK-47u” is actually a different type of weapon.) So, the rows with “AK 47” (no hyphen) are likely typographical errors and will have no match in the `Weapons` dataset. Similar mistakes occur with other weapons, such as “Pelington 703”, “Pellington”, and “Pellington 703”. This helps to illustrate that data quality issues present additional challenges.

This opens up several good discussion points. First, ask students how they will go about solving this issue. Second, it brings up an opportunity to review how different types of joins will handle the situation when no match is present. For instance, `left_join()` and `inner_join()` handle situations without a match differently. Finally, these could be used to motivate the idea of probabilistic joins.

Summary

This example provides an opportunity to have students join tables for the purpose of adding new information and to discuss the decision making process for handling low quality data in the form of typographical errors.

Example 4 - Regression

Discussion Opportunities

This example provides an opportunity to discuss the following ideas:

- linear regression and checking assumptions
- residuals
- multivariable thinking and lurking variables
- indicator variables
- interaction
- comparing models

- outliers
- statistics as an investigative process
- implementation issues on new data

Background Information

In the game, players progress through a series of ranks (Private, Corporal, Sergeant, etc.) by earning eXperience Points (XP). The amount of XP the player earned in each match is contained in the `TotalXP` variable.

The `Score` variable (described in Example 3), along with other factors that are not necessarily included in the dataset, are used to calculate the XP earned during each game.

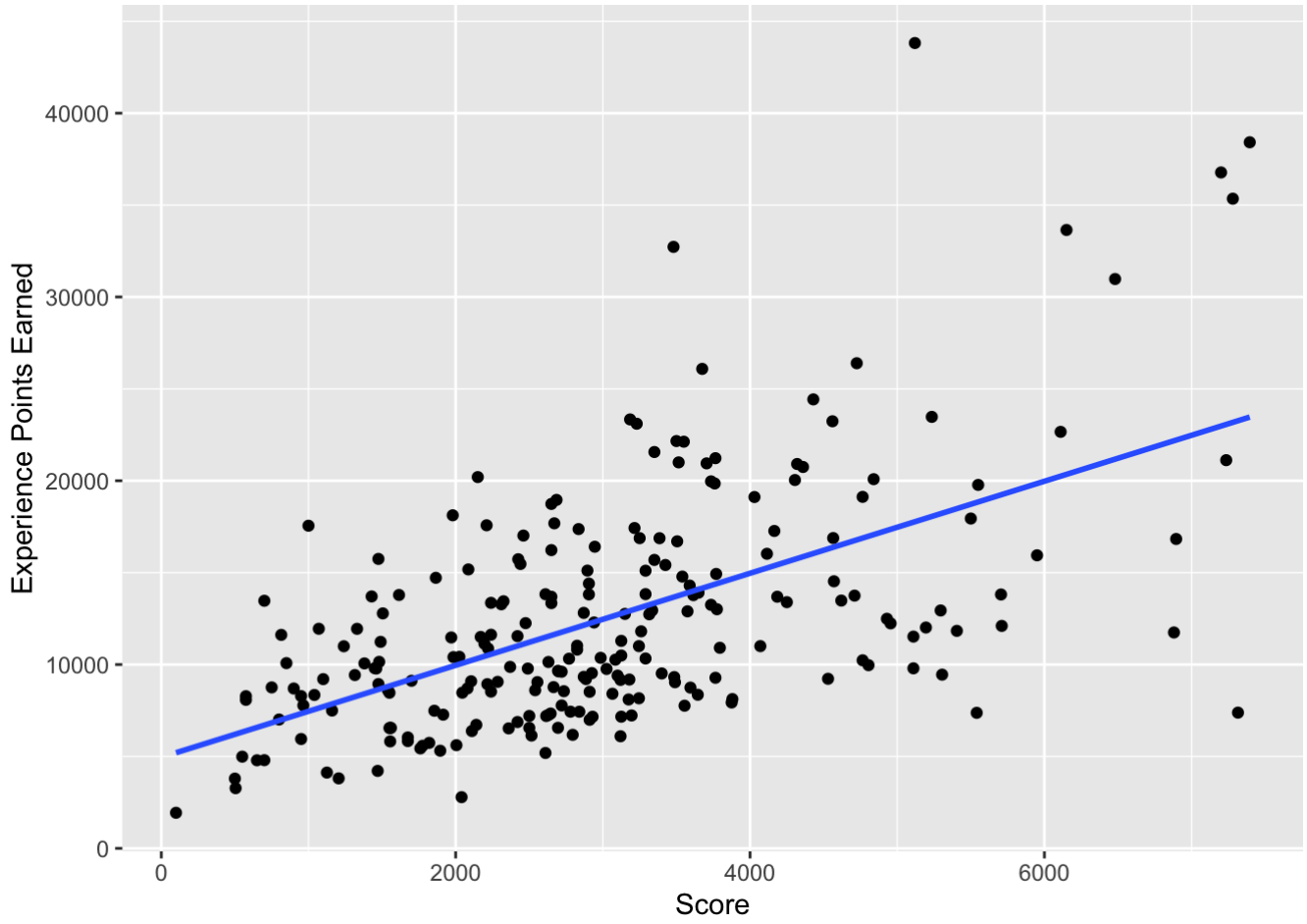
Initial Task and Exploration

Instruction for students: Using the `CODGames1` dataset, answer the following research question: What is the relationship between the player's score and the XP earned?

As a starting point, students may begin to explore the relationship by creating an appropriate visualization such as shown below.

```
#Create a scatterplot showing relationship between TotalXP and Score
ggplot(data = CODGames1, mapping = aes(x = Score, y = TotalXP)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(y = "Experience Points Earned")
```

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




```
#Fit a simple linear regression model
model1 <- lm(TotalXP ~ Score, data = CODGames1)
```

The plot suggests a reasonably linear pattern and produces an $R^2 = 0.33$. However, there might be an issue with increasing variance and/or some outliers in which the player earned more than 30,000 XP in a single match. (More on these outliers can be find near the end of the Revised Task and Exploration section.)

This might serve as a good time to review the assumptions of linear regression and methods for investigating the assumptions.

Important discussion point (Multivariable thinking): I like to use this opportunity to remind students to think about other variables, included in the dataset or otherwise, that might also impact this relationship.

Further Background

Players can earn and use tokens that grant double the amount of XP for short periods of time (e.g., 30 minutes). The variable `XPType` provides information as to the nature of the XP granted. In the `CODGames1` dataset, `XPType` takes on values of `10% Boost` and `Double XP + 10%`, the latter of which suggests that a double XP token has been used.

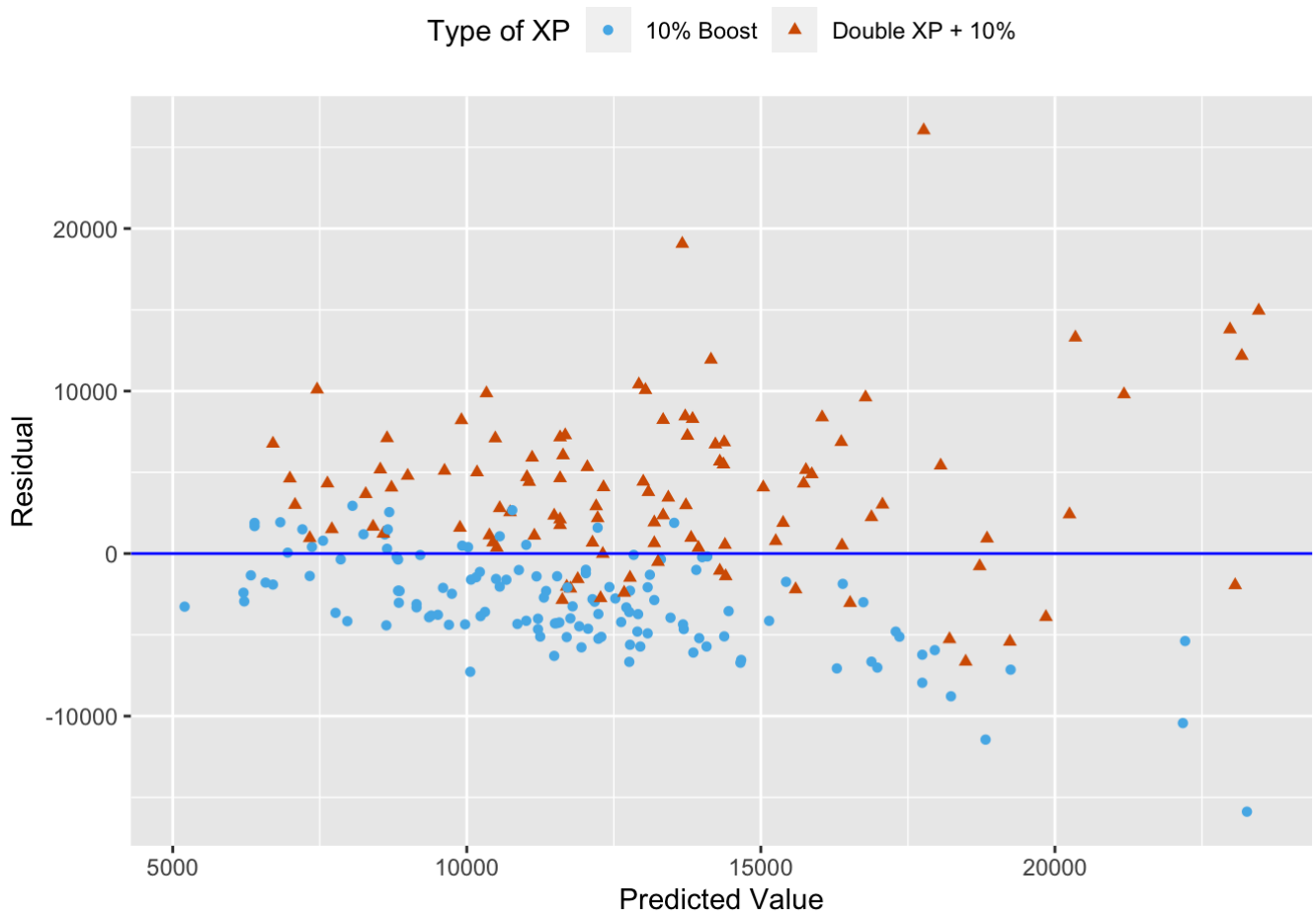
Revised Task and Exploration

Instruction for students: For the regression model for XP based on Score, create a plot showing the residuals as a function of the predicted values. Code the points (use color/shape) to incorporate the `XPType` and look for patterns.

Continued exploration: The plot is shown below. Some discussion points, include adding a horizontal line (through the `geom_hline()`, specifying the colors to be used through `scale_color_manual()`, moving the legend location, and how color might cause problems for people with types of color-blindness (so we added shapes).

```
#Create a temporary data frame containing the residuals, fitted values, and XPType for u
se with ggplot
temp_df <- data.frame(Residuals = model1$residuals, yhat = model1$fitted.values,
                      XPType = CODGames1$XPType)

#Create the visualization
ggplot(data = temp_df, mapping = aes(x = yhat, y = Residuals,
                                     color = XPType, shape = XPType)) +
  geom_point() +
  geom_hline(yintercept = 0, color = "blue") +
  scale_color_manual(values=c("#54B6E9", "#D55E00")) +
  labs(x = "Predicted Value",
       y = "Residual",
       color = "Type of XP",
       shape = "Type of XP") +
  theme(legend.position = "top")
```

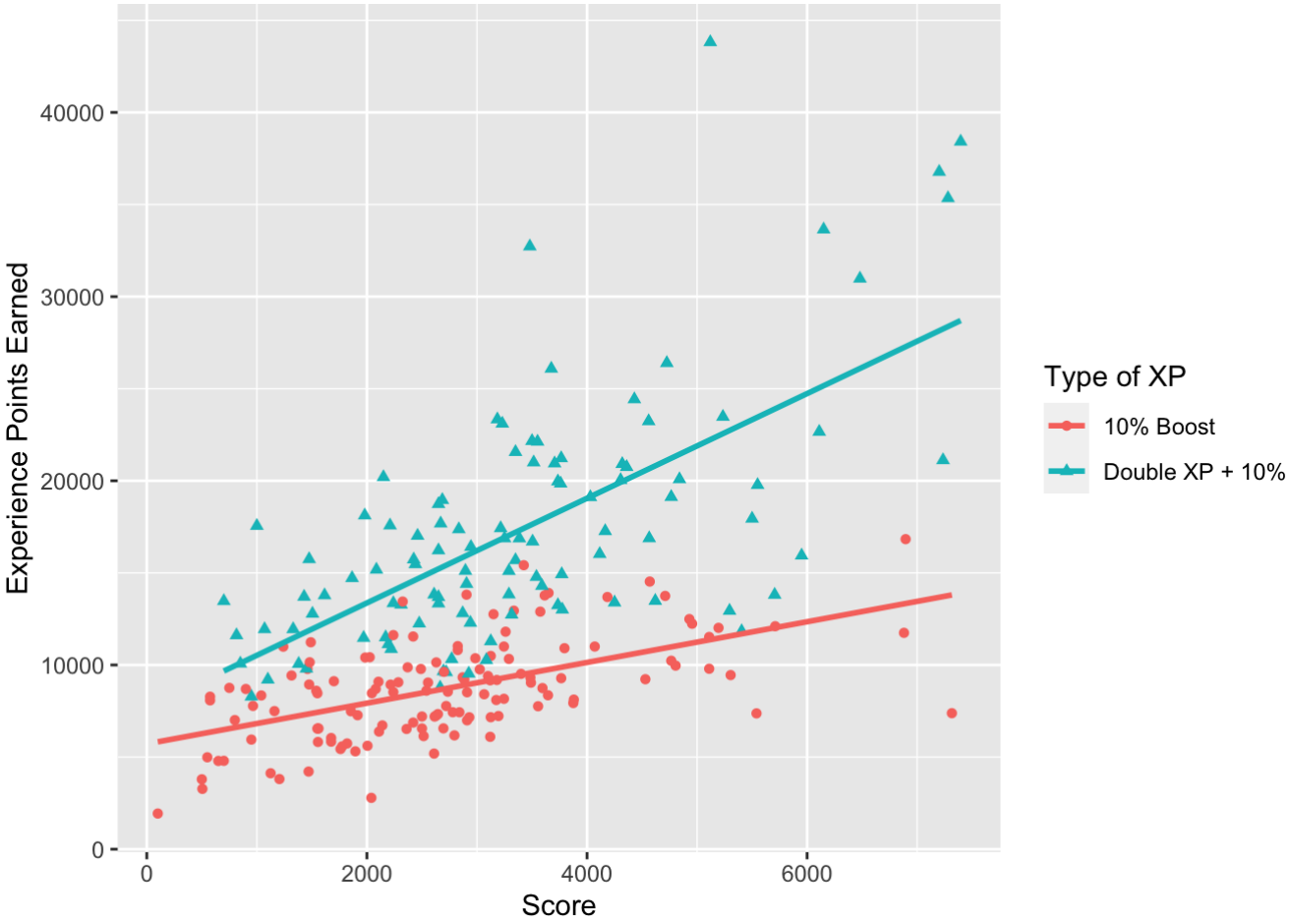


The plot suggests that most of the positive residuals correspond to the situation in which the player has used a double XP token (triangles), while most of the negative residuals correspond to the case when double XP is not used (circles). This means that we are systematically overestimating the XP for cases in which double XP is not

used, while underestimating the XP when double XP tokens are used. Updating the scatterplot of XP as a function of `Score`, but coding by `XPType` helps to visually illustrate the nature of the relationship.

```
#Create a scatterplot showing relationship between TotalXP and Score
ggplot(data = CODGames1, mapping = aes(x = Score, y = TotalXP,
                                       color = XPType, shape = XPType)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(y = "Experience Points Earned",
       color = "Type of XP",
       shape = "Type of XP")
```

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



The plot suggests potentially different intercepts and different slopes for the different levels of `XPType`. At this point, I like to ask students how we can account for this in our model.

We can incorporate this information into our model by introducing an indicator variable for `XPType` and an interaction term between the `XPType` indicator and `Score`. While the `lm()` function can automatically create indicators and the interaction term, I prefer to create my own so that I have control over the use of 0's and 1's.

```
#Create indicator for Double XP and interaction term
CODGames1 <-
  CODGames1 %>%
  mutate(Double = ifelse(XPType == "Double XP + 10%", 1, 0),
         ScoreDoubleInt = Score*Double)

#Build multiple linear regression model
model2 <- lm(TotalXP ~ Score + Double + ScoreDoubleInt, data = CODGames1)
summary(model2)
```

```
##
## Call:
## lm(formula = TotalXP ~ Score + Double + ScoreDoubleInt, data = CODGames1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11212.6  -1962.9    -45.6   1713.4  21580.5
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    5716.2131   724.8749   7.886 1.24e-13 ***
## Score             1.1052     0.2391   4.622 6.33e-06 ***
## Double         1965.5461  1157.0974   1.699   0.0907 .
## ScoreDoubleInt    1.7374     0.3419   5.082 7.70e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3746 on 231 degrees of freedom
## Multiple R-squared:  0.6647, Adjusted R-squared:  0.6604
## F-statistic: 152.7 on 3 and 231 DF,  p-value: < 2.2e-16
```

The analysis of this model can promote a variety of discussions such as comparing models, significance testing, interpreting coefficients, generating confidence intervals, explaining interactions, further investigation of assumptions, etc.

Outliers: The scatterplots of `TotalXP` as a function of `Score` reveal a few outliers on the high end (XP more than 30,000). The outliers can be explained with some knowledge of the data collection. Basically, as a player uses a weapon, the player will complete something called a weapon challenge for completing certain feats. Upon completion of all weapon challenges for a given weapon, the player is awarded 10,000 XP points. If the player was also using a double XP token, this results in the player earning an extra 20,000 points. This is the reason for the extremely high values of `TotalXP` in a few matches.

Important discussion point: I usually take this opportunity to discuss how subject matter expertise/domain knowledge gives certain students an advantage working with this dataset. Although I provide students with the necessary background to guide the exploration, students who have knowledge of the game would likely outperform peers lacking this knowledge. This translates into real world situations as well.

Challenges on Other Gameplay Dataset (`CODGames2`)

Even when students understand the methodology covered in this example, new challenges arise when implementing the methods on different datasets. Applying the same code and methods to the `CODGames2` dataset presents several challenges.

First, the code to create the temporary data frame used for the residuals vs. fitted values plot, no longer works.

```
#Build SLR model on CODGames2
model1_dat2 <- lm(TotalXP ~ Score, data = CODGames2)

#Create temporary dataframe for plotting as we did before
temp_df <- data.frame(Residuals = model1_dat2$residuals, yhat = model1_dat2$fitted.value
s,
                      XPType = CODGames2$XPType)
```

```
## Error in data.frame(Residuals = model1_dat2$residuals, yhat = model1_dat2$fitted.valu
es, : arguments imply differing number of rows: 1309, 1464
```

The error messages mentions that arguments imply differing number of rows. The 1464 number corresponds to the number of rows in `CODGames2` . The 1309 number is a little more mysterious. It corresponds to the number of residuals (or predicted values) from the model. Why are there 1309 residuals when the dataset had 1464 rows? This is because the dataset has some missing values in the `TotalXP` and/or `Score` columns.

Second, suppose that we wish to recreate the scatterplot of `TotalXP` as a function `Score` coded by `XPType` using the `CODGames2` dataset.

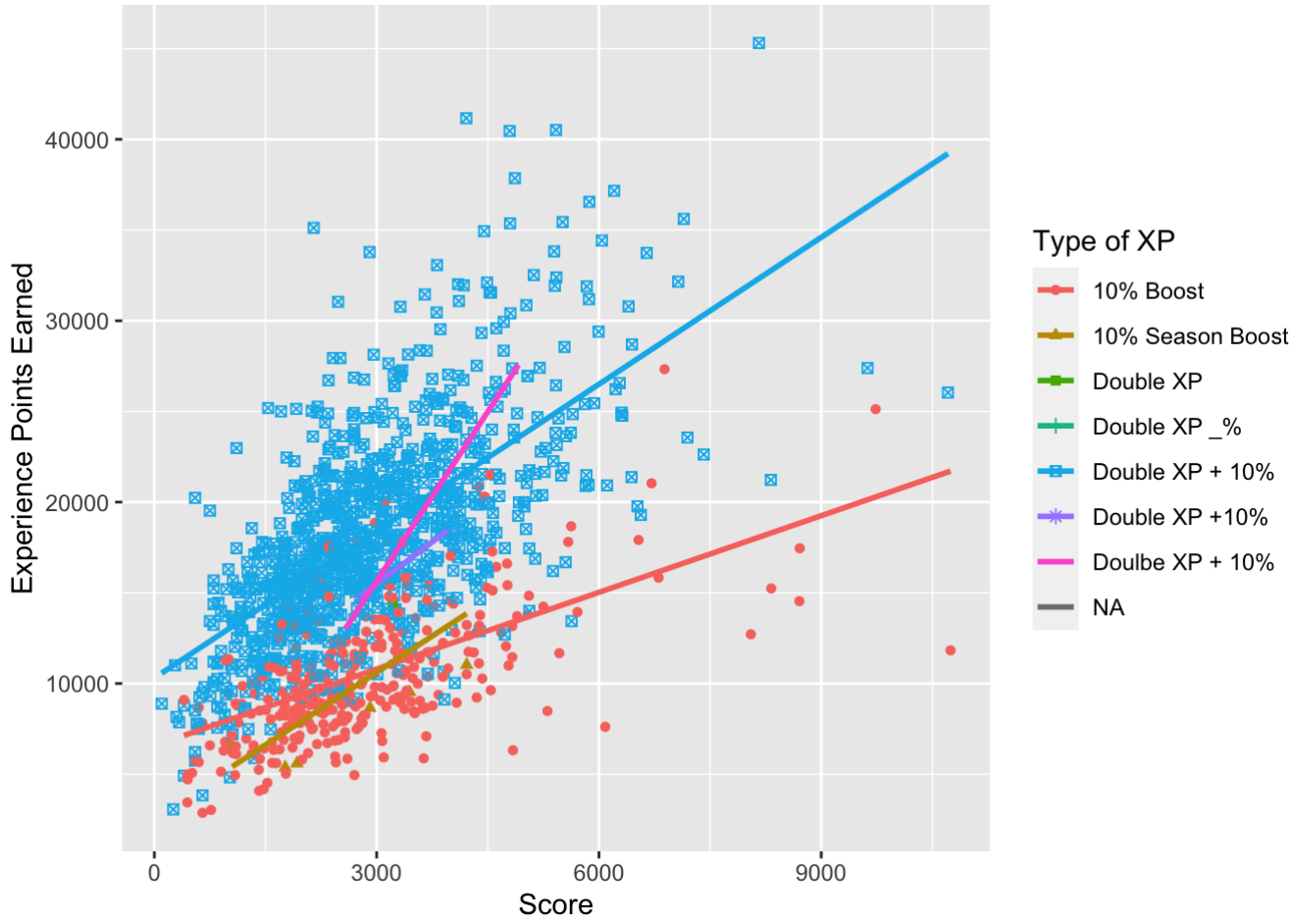
```
#Create a scatterplot showing relationship between TotalXP and Score
ggplot(data = CODGames2, mapping = aes(x = Score, y = TotalXP,
                                       color = XPType, shape = XPType)) +
  geom_point() +
  geom_smooth(method = lm, se = FALSE) +
  labs(y = "Experience Points Earned",
       color = "Type of XP",
       shape = "Type of XP")
```

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

```
## Warning: Removed 155 rows containing non-finite values (stat_smooth).
```

```
## Warning: The shape palette can deal with a maximum of 6 discrete values because
## more than 6 becomes difficult to discriminate; you have 7. Consider
## specifying shapes manually if you must have them.
```

```
## Warning: Removed 157 rows containing missing values (geom_point).
```



The plot reveals that `XPType` has more than the 2 levels (`Double XP + 10%` and `10% Boost`) we expected. Most of the additional levels are typos, spacing issues such as no space after `+` sign, or inconsistent naming (in the case of `10% Boost` vs `10% Season Boost`). This provides a nice opportunity to discuss how students will handle the extra levels.

Third, we often compare models by assessing performance on a test set. Suppose we use `model2` , which was trained used `CODGames1` and included the `XPType` indicator and interaction terms. Suppose we want to see how will this model will generalize to new data in the form of `CODGames2` . Allow R to create its own indicators (something students prefer to do but should do with caution) and add the interaction term through the model statement as shown below. Then, try to obtain predictions for `CODGames2` .

```
#Recreate model2 using CODGames1 but let R create indicators
model2 <- lm(TotalXP ~ Score + XPType + Score:XPType, data = CODGames1)

test_pred <- predict(model2, newdata = CODGames2)
```

```
## Error in model.frame.default(Terms, newdata, na.action = na.action, xlev = object$xle
vels): factor XPType has new levels 10% Season Boost, Double XP, Double XP _%, Double XP
+10%, Doulbe XP + 10%
```

The throws an error since the `XPType` variable in `CODGames2` has new levels that did not show up in the training data (`CODGames1`).

Summary

This example provides an opportunity to discuss statistical thinking as an investigative process, explore multivariable thinking, and covers some common implementation challenges caused by data quality issues.

Final Comment

Thanks for making it this far. If you find any errors, have any feedback, or would like to share additional ideas, please contact Matt Slifko (mds6457@psu.edu (mailto:mds6457@psu.edu)).