LoL-Statistics

Andrew Roddy, Caleb Stanberry, Logan Senol

2025-11-24

```
library(dplyr)
library(ggplot2)
library(scales)
library(car)
library(sandwich)
library(lmtest)
library(plotly)
library(reshape2)
library(tidyr)
```

League of Legends Crash Course

What is League of Legends?

Components

Six main components

- champions
- teams
- towers
- minions
- gold
- nexus

Champions

Before the game each player selects one of the 172 champions. Each champion has unique abilities and stats.

Teams

Two teams compete against eachother with 5 players on each team. Each player chooses an different role.

The Nexus

Destroy the opposing teams nexus to win the game. There are problems along the way.

Towers

Each team has 9 towers on their side of the map. They must be destroyed to access the Nexus but if the opponent gets too close they attack.

Minions

Luckily minions can be used to distract the towers from attacking the players. The minions also drop gold when eliminated.

Gold

Gold can be used to purchase items which increase the stats of the champions.

Data Size

Our analysis uses data from 150,000 matches of player data and 68,000 matches of team data. This is why our P-values may be very small and why there will be a lot of data points in the scatter plots.

```
match_stats <- read.csv("data/MatchStatsTbl.csv")
row_count <- nrow(match_stats)
col_count <- ncol(match_stats)
cat("Player Data:\nThere are", row_count, "rows.\n")
```

```
## Player Data:
## There are 150505 rows.
```

```
cat("There are", col_count, "columns.\n\n")
```

```
## There are 31 columns.
```

```
team_stats <- read.csv("data/TeamMatchTbl.csv")
row_count2 <- nrow(team_stats)
col_count2 <- ncol(team_stats)
cat("Team Data:\nThere are", row_count2, "rows.\n")
```

```
## Team Data:
## There are 68676 rows.
```

```
cat("There are", col_count2, "columns.\n")
```

```
## There are 24 columns.
```

```
summoner_match <- read.csv("data/other/SummonerMatchTbl.csv")
match_stats <- match_stats |>
  left_join(summoner_match, by = c("SummonerMatchFk" = "SummonerMatchId"))
```

```r
champions <- read.csv("data/keys/ChampionTbl.csv")
match_stats <- match_stats |>
left_join(champions, by = c("ChampionFk" = "ChampionId"))

match_data <- read.csv("data/other/MatchTbl.csv")
match_stats <- match_stats |>
left_join(match_data, by = c("MatchFk" = "MatchId"))
```

```r
# Removes unimportant columns of data
match_stats[, c(
  "item1", "item2", "item3", "item4", "item5", "item6",
  "PrimarySlot1", "PrimarySlot2", "PrimarySlot3",
  "SecondarySlot1", "SecondarySlot2",
  "SummonerSpell1", "SummonerSpell2"
)] <- list(NULL)

match_stats <- match_stats |>
  filter(QueueType == "CLASSIC")

names <- colnames(match_stats)
```

Multiple Linear Regression

We use multiple linear regression to find the factors that influence damage dealt by players.

This test finds the relationship between damage dealt and the predictors damage taken, total gold, minions killed, and towers destroyed (turret damage dealt). This test allows us to see the combined and individual effects of the continuous variables.

Multiple Linear Regression Assumptions

The assumptions of multiple linear regression are:

1. independence
2. linearity
3. no multicollinearity
4. normality of residuals
5. homoscedasticity

Before doing multiple linear regression we must check that all of these are true

Independence

All 150,000 matches in out game are done in different matches. These means the outcome of the matches do not influence eachother. Because of this we know each data point is independent.

5 Assumptions

1. ~~independence~~

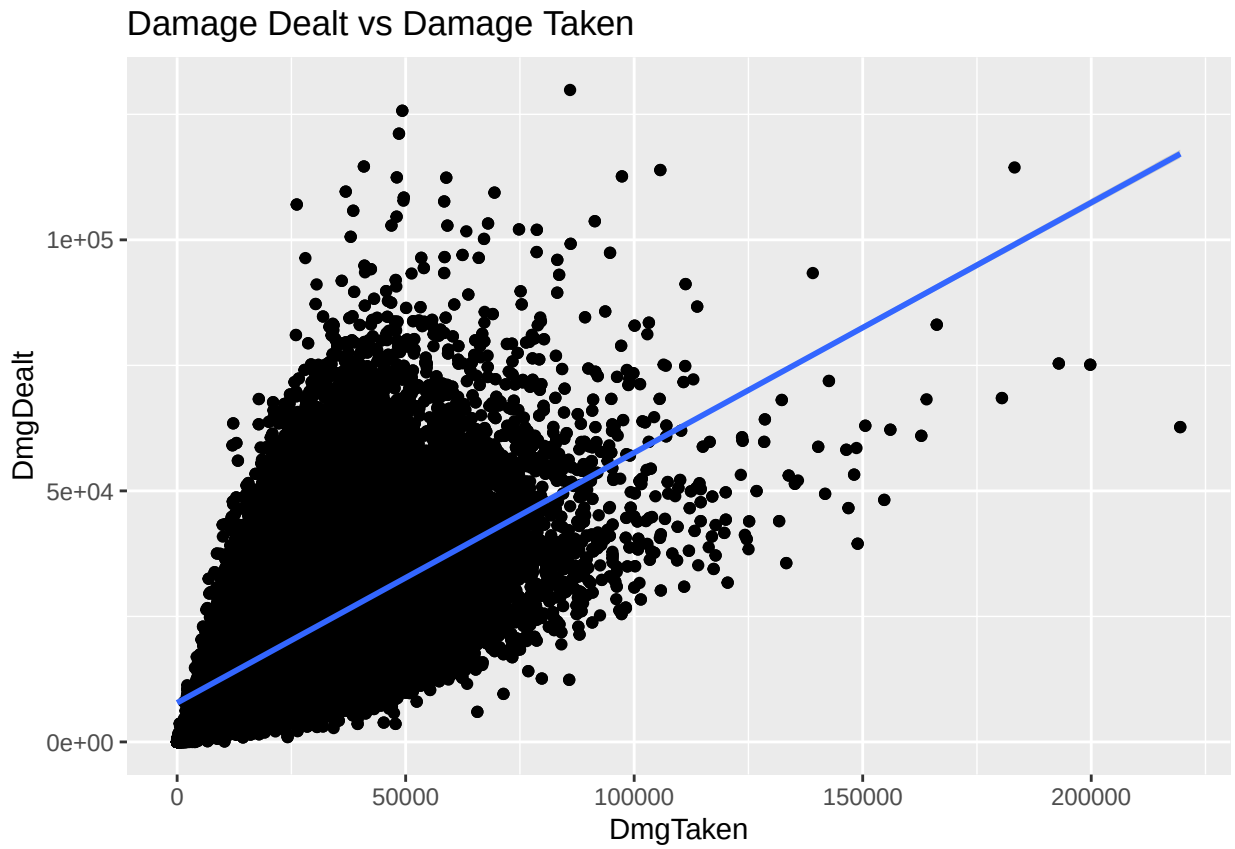
2. **linearity**
3. no multicollinearity
4. normality of residuals
5. homoscedasticity

Linearity

To test linearity we will use `ggplot` to make scatter plots for each type of data. We will be doing a plot of Damage Dealt against : Damage Taken, Total Gold, and Minions Killed, Turret's Destroyed. We will then visually check for linearity.

Damage Dealt vs Damage Taken

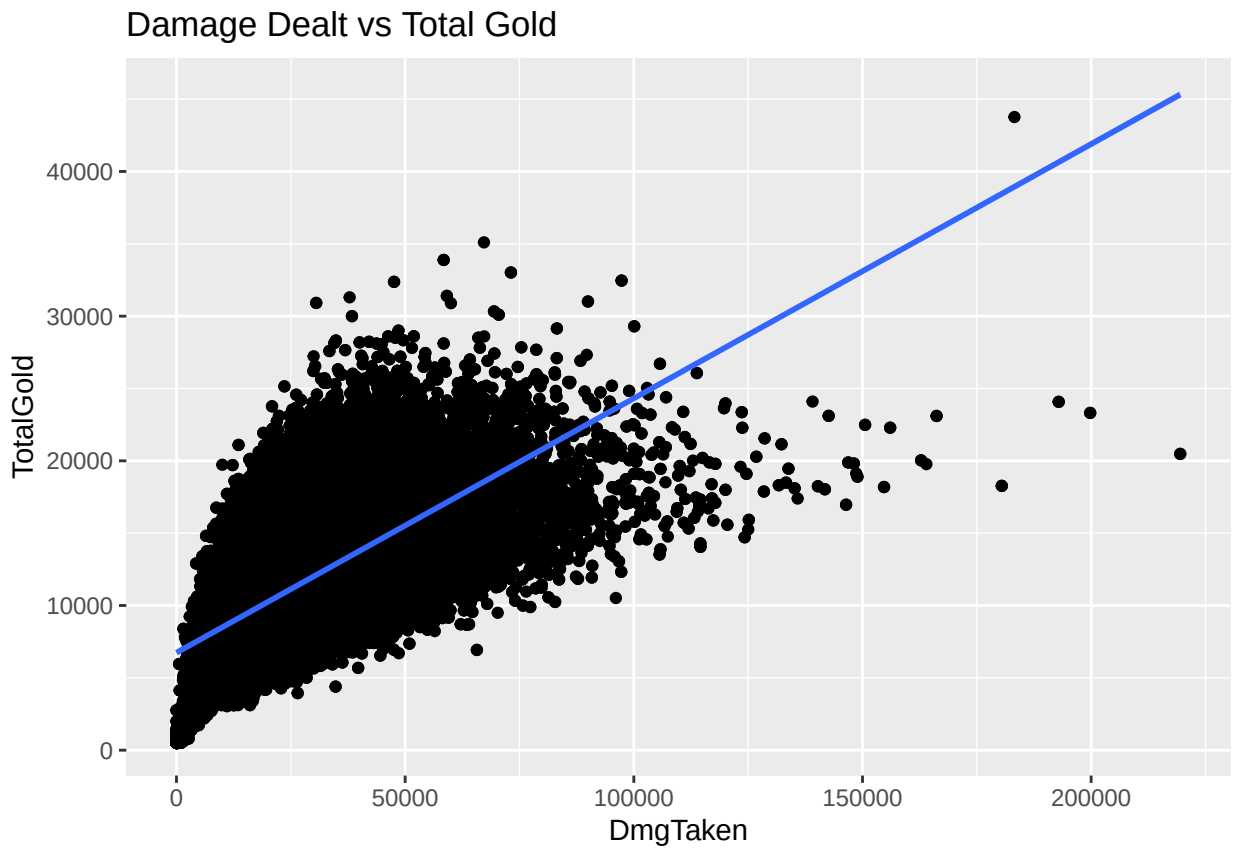
```
ggplot(match_stats, aes(x = DmgTaken, y = DmgDealt)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Damage Dealt vs Damage Taken")
```



Visually linear.

Damage Dealt vs Total Gold

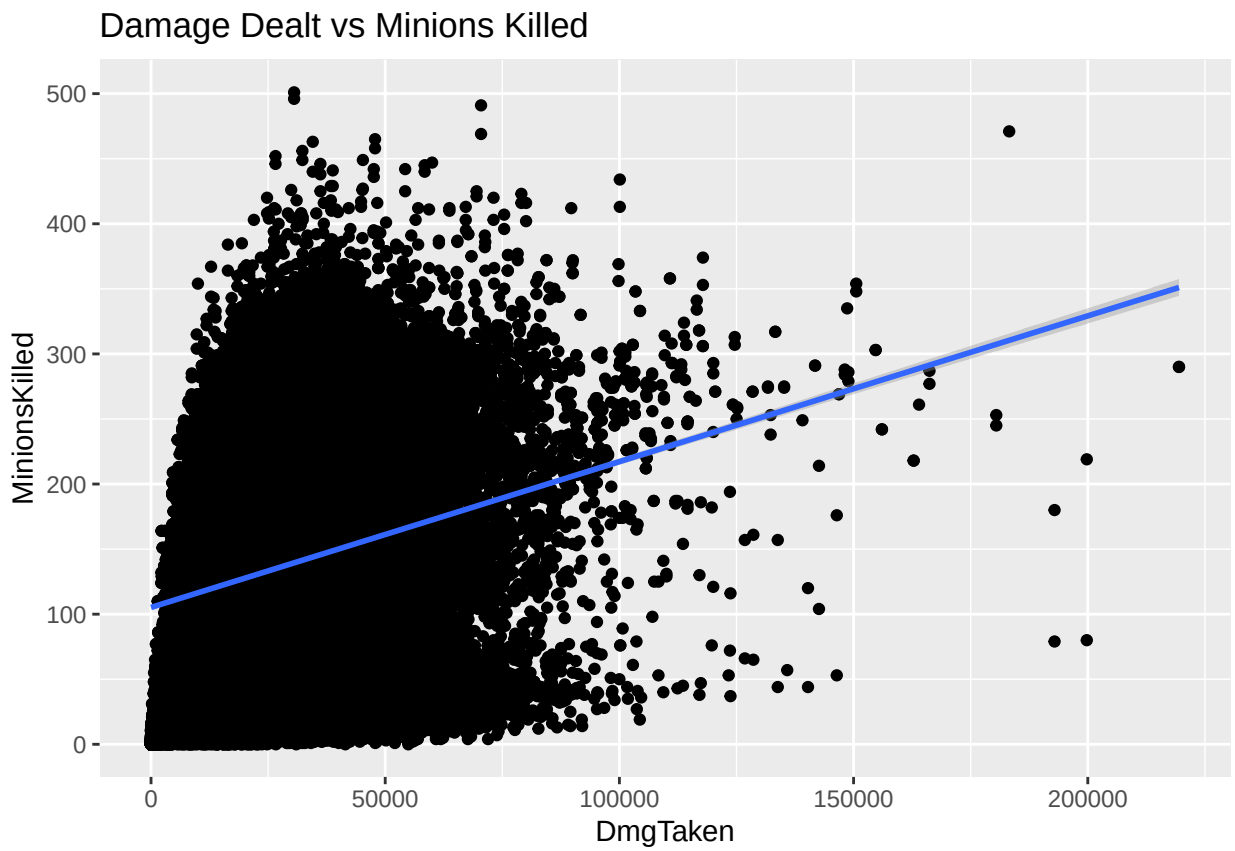
```
ggplot(match_stats, aes(x = DmgTaken, y = TotalGold)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Damage Dealt vs Total Gold")
```



Visually linear.

Damage Dealt vs Minions Killed

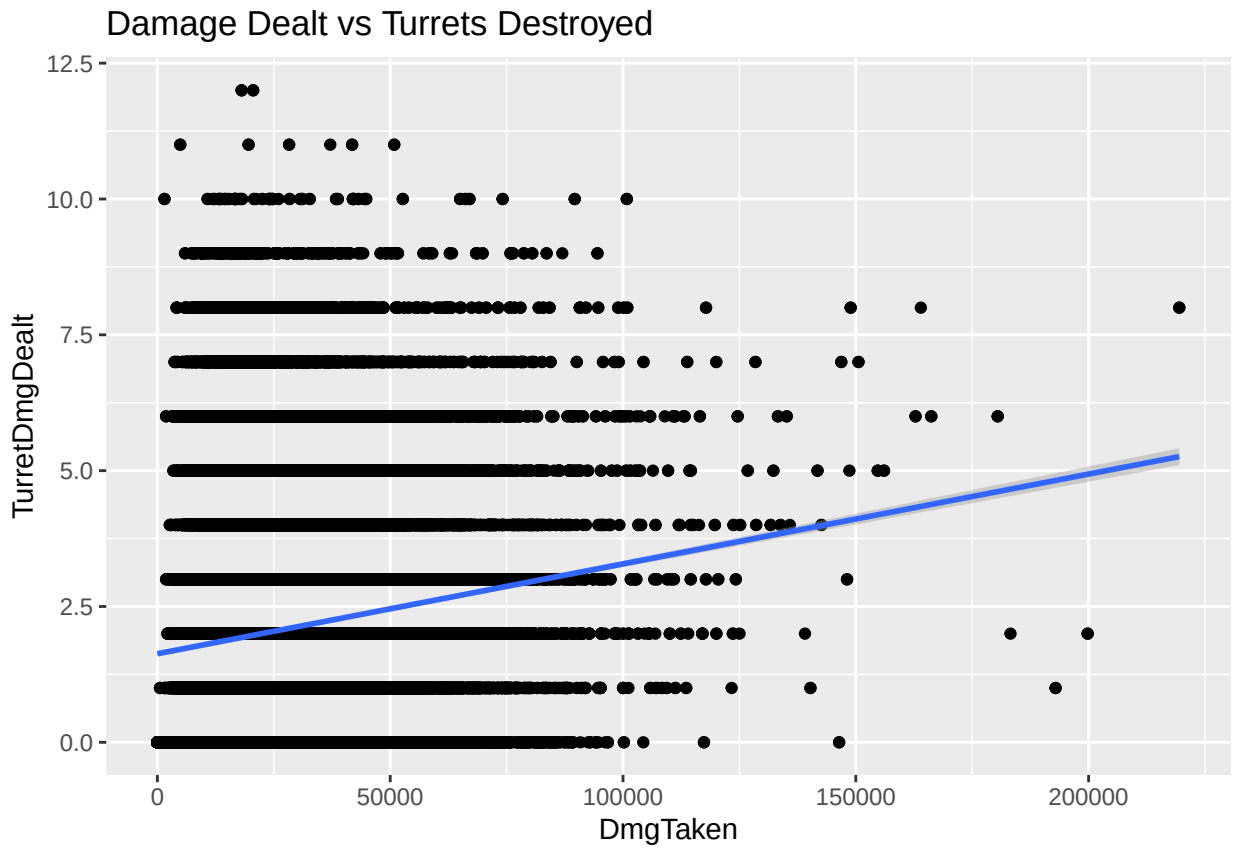
```
ggplot(match_stats, aes(x = DmgTaken, y = MinionsKilled)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Damage Dealt vs Minions Killed")
```



Visually not linear

Damage Dealt vs Turrets Destroyed

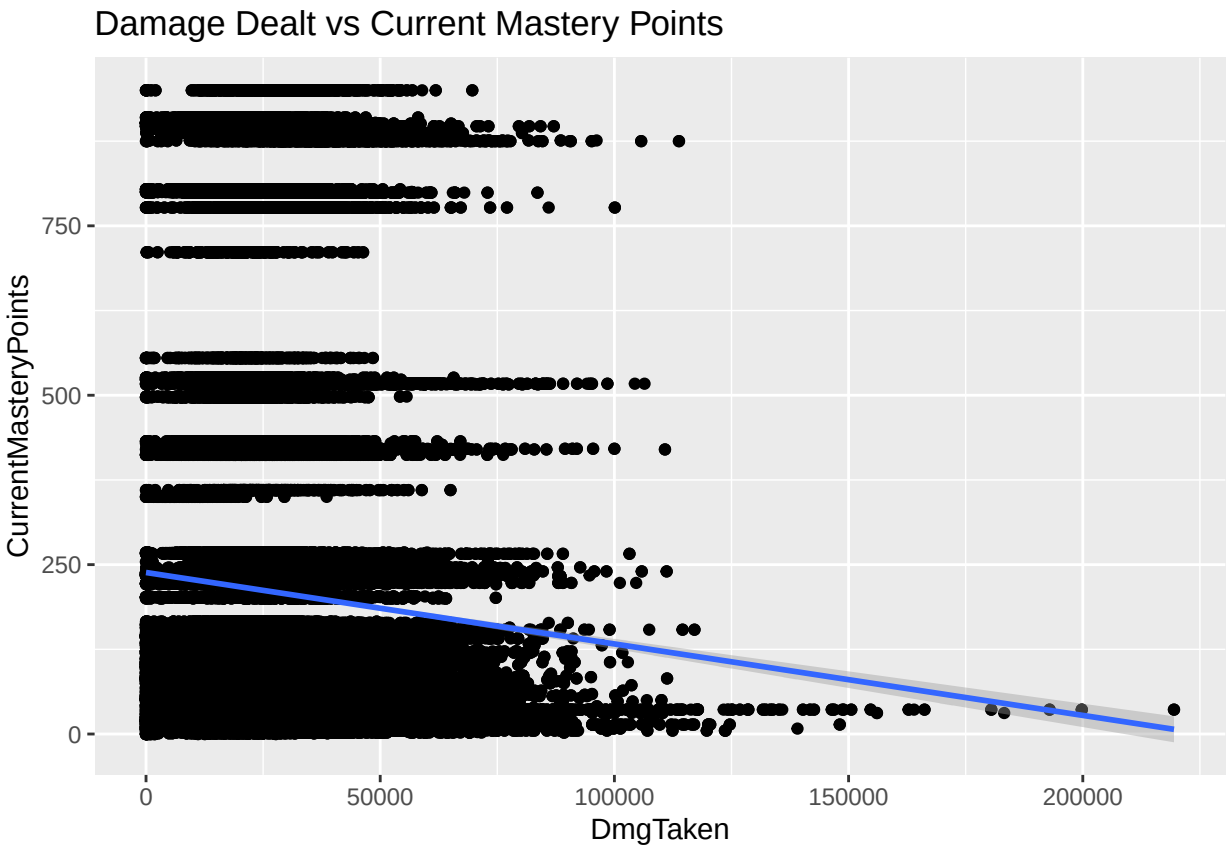
```
ggplot(match_stats, aes(x = DmgTaken, y = TurretDmgDealt)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Damage Dealt vs Turrets Destroyed")
```



Visually not linear

Damage Dealt vs Curret Mastery Points

```
ggplot(match_stats, aes(x = DmgTaken, y = CurrentMasteryPoints)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Damage Dealt vs Current Mastery Points")
```



Visually not linear

Changing the Test

Because we can only use linear relationships between variables we will have to change what we test.

Because of this we will only be testing the dependent variable of Damage Dealt against the independent variables of Damage Taken and Total Gold.

5 Assumptions

1. ~~independence~~
2. ~~linearity~~
3. **no multicollinearity**
4. normality of residuals
5. homoscedasticity

Checking for no multicollinearity

We also need to check that there is no multicollinearity. This makes sure that our two independent variables are not too highly coorelated. This would be like if every time we took one damage we got one gold. This would make the individual coefficients unreliable. To test for this we need to calculate the Variance Inflation Factor.

```r
# make simple regression model
multiple_regression <- lm(DmgDealt ~
    DmgTaken + TotalGold,
  data = match_stats
)
```

Variance Inflation Factor

A VIF of 1 means there is no correlation. A VIF of greater than 1 means there is some coorelation. This only really matters if the VIF is greater than 5 as this suggests a high multicollinearity.

```r
# Check for multicollinearity (if they are too dependent)
# Values 1-2 are good
# Our values are good
values <- vif(multiple_regression)
cat(
  "Damage Taken:   ",    as.numeric(values["DmgTaken"]),
  "\nTotal Gold:     ",      as.numeric(values["TotalGold"]), "\n"
)
```

```
## Damage Taken:    1.644358
## Total Gold:      1.644358
```

5 Assumptions

1. ~~independence~~
2. ~~linearity~~
3. ~~no multicollinearity~~
4. **normality of residuals**
5. homoscedasticity

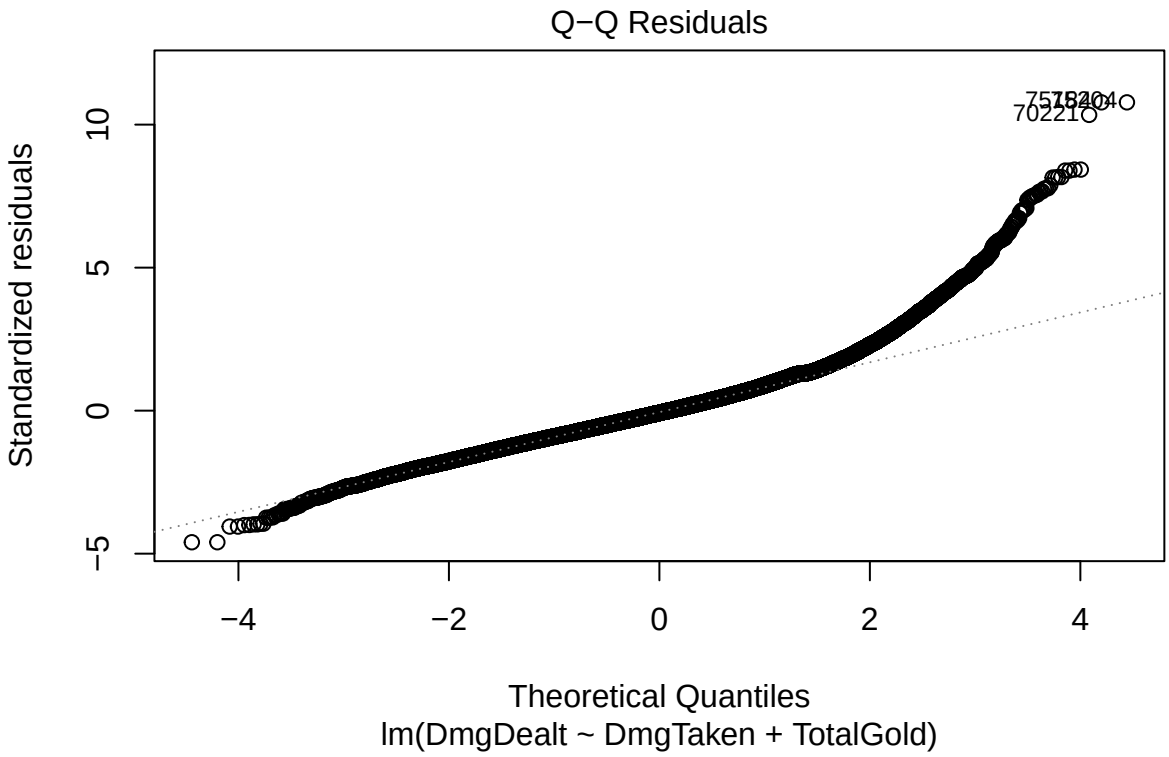
Normailty of Residuals

Based on the test, both of our variables have a pretty low multicollinearity and are safe to use.

To check for normality of residuals we can plot the multiple regression graph. Using `which = 2` prints a Q-Q Residuals graph or Quantile-Quantile plot of residuals. If this is straight diagnol it means our data follows a normal distribution.

Q-Q Residuals Graph

```r
plot(multiple_regression, which = 2)
```



Q-Q Residuals Graph Takeaway

The graph not being straight diagonal is okay for our data through. This is alright because of the Central Limit Theorem. This basically means that if the sample size is greater than 30 the sample mean will be about normal. Luckily we have 150,000 samples which allows us to ignore this requirement.

5 Assumptions

1. ~~independence~~
2. ~~linearity~~
3. ~~no multicollinearity~~
4. ~~normality of residuals~~
5. **homoscedasticity**

Checking for Homoscedasticity

Homoscedasticity is same variance. It means that if the player had a low, medium, or high damage taken, the spread of how far the predicted damage dealt is from the actual damage dealt should be about the same. We want homoscedasticity. To test for this we can use a Breusch-Pagan test.

Breusch-Pagan Test

Because our p value is so low we reject the idea that we have homoscedasticity which is a requirement for this test. This means our confidence intervals may be incorrect but our other visualizations should be fine.

```
breusch_pagan_test <- ncvTest(multiple_regression)
print(breusch_pagan_test)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 24605.11, Df = 1, p = < 2.22e-16
```

Multiple Linear Regression Summary

This is a summary of our multiple linear regression without accounting for homoscedasticity. The coorelation between damage dealt, damage taken, and total gold is statistically significant.

```
# Print off if this matters or not
summary(multiple_regression)
```

```
##
## Call:
## lm(formula = DmgDealt ~ DmgTaken + TotalGold, data = match_stats)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -31089  -4331   -532   3614  72834
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.019e+04  5.947e+01 -171.26   <2e-16 ***
## DmgTaken     2.903e-02  1.739e-03   16.69   <2e-16 ***
## TotalGold    2.669e+00  6.192e-03  430.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6758 on 112583 degrees of freedom
## Multiple R-squared:  0.7401, Adjusted R-squared:  0.7401
## F-statistic: 1.603e+05 on 2 and 112583 DF,  p-value: < 2.2e-16
```

Accounting for Homoscedasticity

Using robust coefficients allows us to account for the lack of homoscedasticity. It ensures the standard errors are valid even when the residual variance is not constant.

```
coeftest(multiple_regression, vcov = vcovHC(multiple_regression, type = "HC1"))
```

```
##
## t test of coefficients:
##
```

```
##                Estimate  Std. Error  t value  Pr(>|t|)
## (Intercept) -1.0186e+04  6.9931e+01 -145.651 < 2.2e-16 ***
## DmgTaken     2.9030e-02  2.2671e-03   12.805 < 2.2e-16 ***
## TotalGold    2.6686e+00  8.1821e-03  326.158 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Coefficient Test Conclusions

Luckily our p values are so small not having homoscedasticity doesn't have a large effect. This very slightly changes our estimates and t values. This does not change our p-values though as they are so small.

5 Assumptions

Now that our assumptions are correct. We can use partial regression to make interesting graphs.

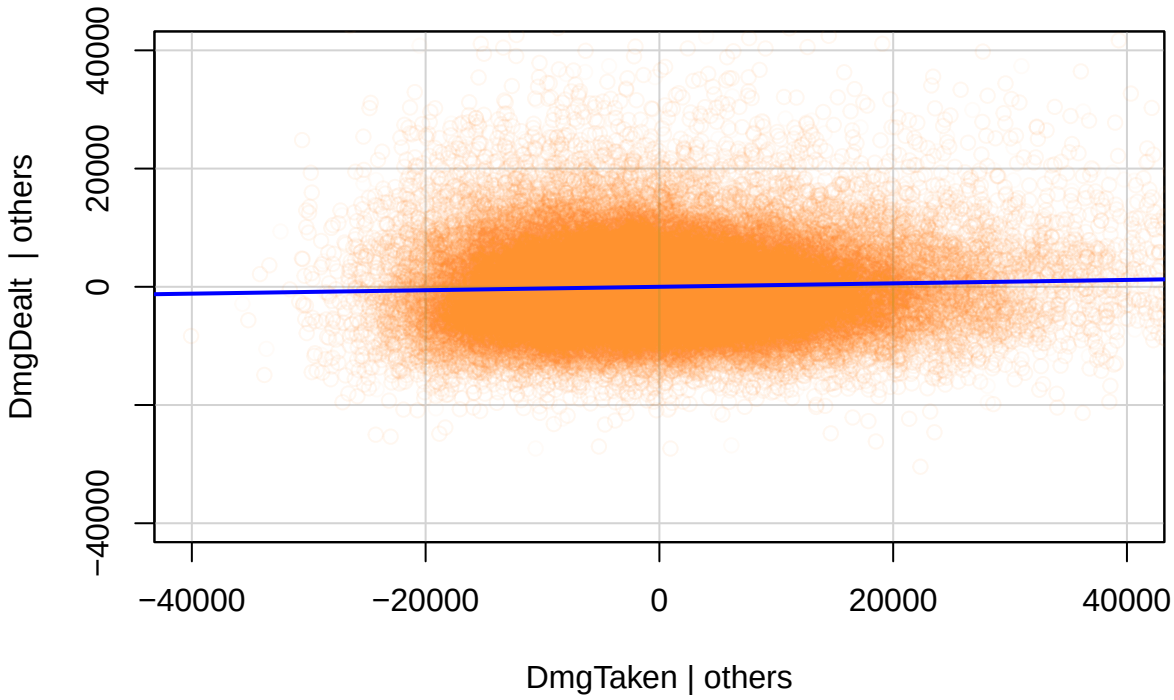
1. ~~independence~~
2. ~~linearity~~
3. ~~no multicollinearity~~
4. ~~normality of residuals~~
5. ~~homoscedasticity~~

Partial Regression

The next graphs will use Partial Regression. This uses the previous multiple linear regression claculations but holds all other variables constant. Holding all other variable's constant allows us to use a 2d graph to compare variables impact on eachother.

Partial Regression

```r
# Partial Regression
avPlots(
  multiple_regression,
  "DmgTaken",
  col = alpha("#ff922f", 0.04), # 4% opacity
  xlim = c(-40000, 40000),
  ylim = c(-40000, 40000)
)
```



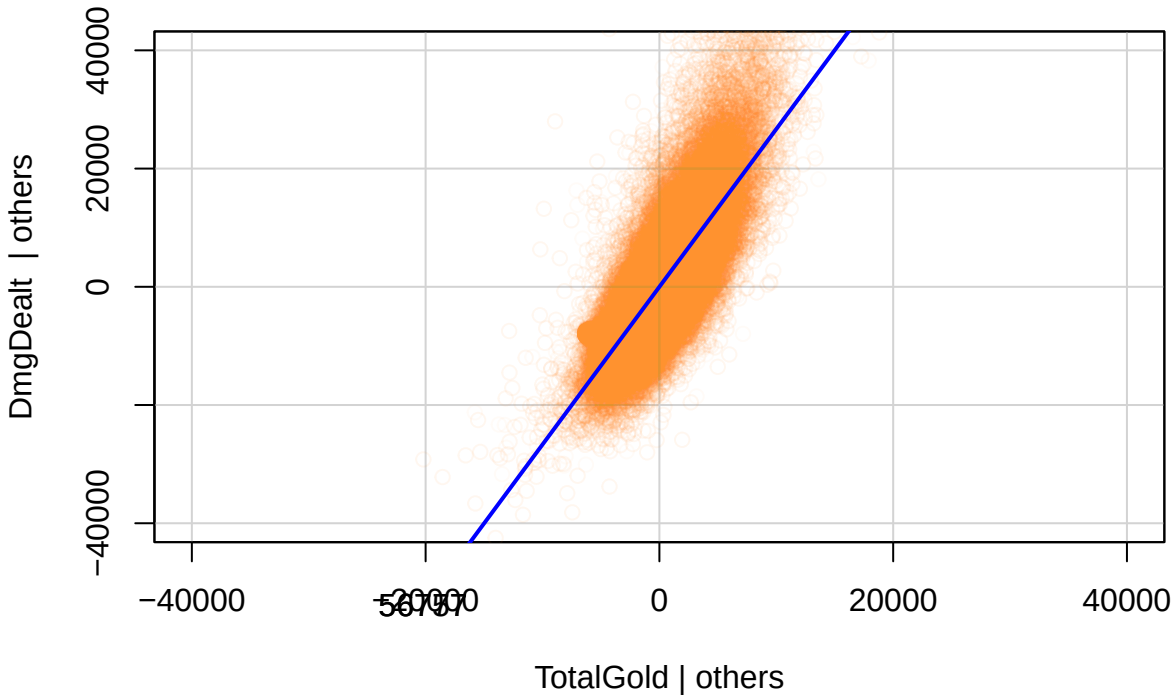
Compares Damage Dealt with Damage Taken.

Partial Regression

Takeaway: Both having a positive relationship makes sense as hitting sombody usually results in getting hit back.

Partial Regression

```
avPlots(
  multiple_regression,
  "TotalGold",
  col = alpha("#ff922f", 0.04),
  xlim = c(-40000, 40000),
  ylim = c(-40000, 40000)
)
```



Compares Damage Dealt with Total Gold aquired.

Partial Regression

Takeaway: Gold allows users to purchase items that increase damage. More gold meaning more damage dealt makes sense.

3D Multiple Linear Regression Graph

One advantage of only having two independent variables is we can make a 3D graph! This graph is essentially a combination of the previous two so no other conclusions will likely be drawn from it.

3D Graph

```
# Makes the regression plane
damage_taken_sequence <- seq(
  min(match_stats$DmgTaken),
  max(match_stats$DmgTaken),
  length.out = 30 # 30 points for plane
)

total_gold_sequence <- seq(
  min(match_stats$TotalGold),
```

```
  max(match_stats$TotalGold),
  length.out = 30
)

# Grid of all combinations of damage taken and damage dealt
grid <- expand.grid(
  DmgTaken = damage_taken_sequence,
  TotalGold = total_gold_sequence
)

# Predicts damage dealt for each point in the grid
grid$PredictedDmg <- predict(multiple_regression, newdata = grid)

# Makes empty plot point
three_dimension_graph <- plot_ly()

# Puts in the dots
three_dimension_graph <- add_markers(
  p = three_dimension_graph, # Adds onto the plot I already made
  data = match_stats,
  x = ~DmgTaken,
  y = ~TotalGold,
  z = ~DmgDealt,
  marker = list(
    color = "#ff922f", # Makes them orange
    opacity = 0.05, # Makes the dots 5% opacity
    size = 5 # Makes them a bit smaller
  )
)

# Puts in the regression plane
three_dimension_graph <- add_surface(
  p = three_dimension_graph, # Adds onto the plot again
  x = ~damage_taken_sequence,
  y = ~total_gold_sequence,
  z = matrix(
    grid$PredictedDmg,
    nrow = length(damage_taken_sequence),
    ncol = length(total_gold_sequence)
  )
)

three_dimension_graph <- layout(
  p = three_dimension_graph,
  margin = list(l = 0, r = 0, b = 0, t = 0)
)

three_dimension_graph
```

3D graphs do not render in pdf form. Run the quarto presentation or Rmd file to view.

Multiple Linear Regression Conclusion

From this test we can confidently conclude that damage taken and total gold can significantly predict the amount of damage a player has dealt.

Pearson Correlation Tests

This test measures the strength and direction of a linear relationship between two variables.

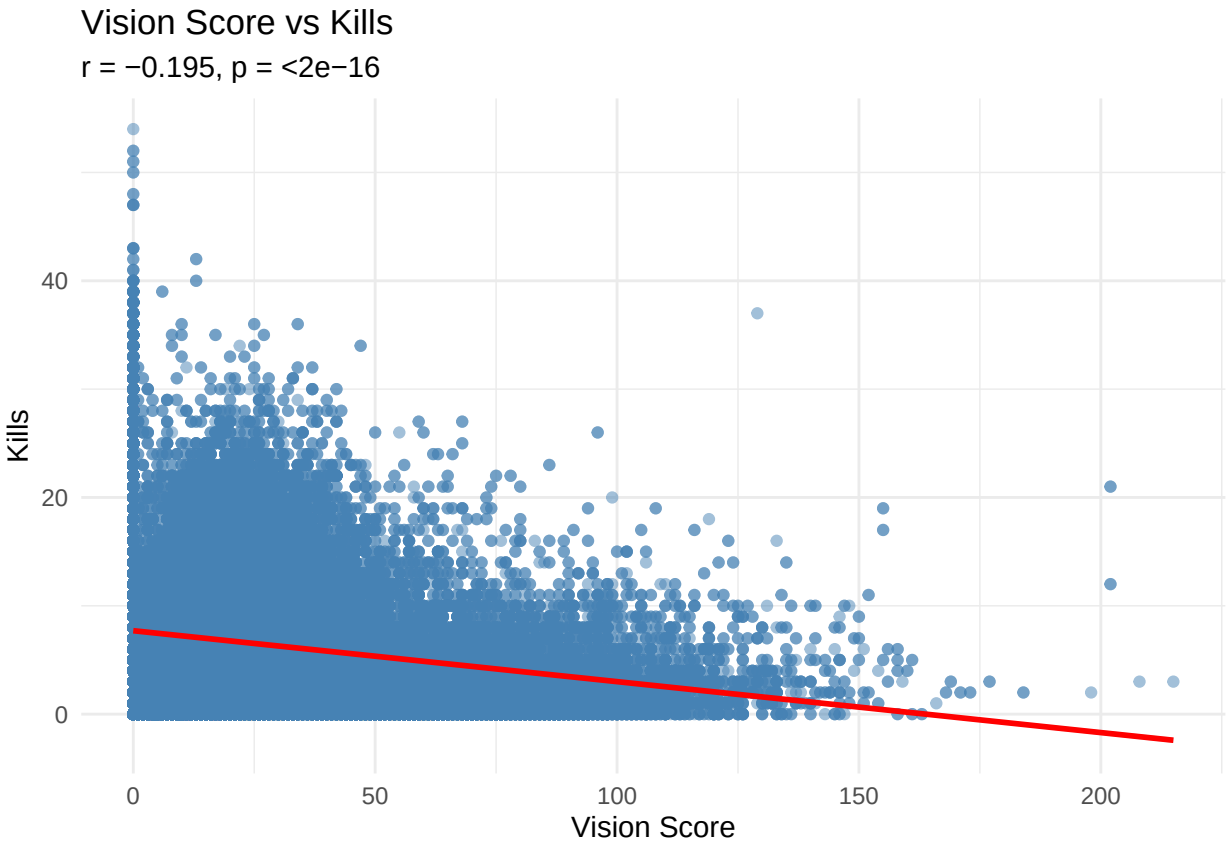
Assumptions:
- Both variables are numerical - Normal distributions for both variables - Approximately linear relationship

```
match_stats <- read.csv("data/MatchStatsTbl.csv")
summoner_match <- read.csv("data/other/SummonerMatchTbl.csv")
match_stats <- match_stats |>
 left_join(summoner_match, by = c("SummonerMatchFk" = "SummonerMatchId"))
champions <- read.csv("data/keys/ChampionTbl.csv")
match_stats <- match_stats |>
 left_join(champions, by = c("ChampionFk" = "ChampionId"))

# Vision Score vs Kills

cor_test_kills <- cor.test(match_stats$visionScore,
                           match_stats$kills,
                           method = "pearson")

# Scatterplot with regression line
vscore_kills_plot <- ggplot(match_stats, aes(x = visionScore, y = kills)) +
  geom_point(alpha = 0.5, color = "steelblue") +
  geom_smooth(method = "lm", se = TRUE, color = "red") +
  labs(
    title = "Vision Score vs Kills",
    subtitle = paste0("r = ", round(cor_test_kills$estimate, 3),
                      ", p = ", format.pval(cor_test_kills$p.value,
                                            digits = 3)),
    x = "Vision Score",
    y = "Kills"
  ) +
  theme_minimal()

# Vision Score vs Deaths
cor_test_deaths <- cor.test(match_stats$visionScore,
                            match_stats$deaths,
                            method = "pearson")

# Scatterplot with regression line
vscore_deaths_plot <- ggplot(match_stats, aes(x = visionScore, y = deaths)) +
  geom_point(alpha = 0.5, color = "coral") +
  geom_smooth(method = "lm", se = TRUE, color = "darkred") +
  labs(
    title = "Vision Score vs Deaths",
    subtitle = paste0("r = ", round(cor_test_deaths$estimate, 3),
                      ", p = ", format.pval(cor_test_deaths$p.value,
                                            digits = 3)),
```

```
    x = "Vision Score",
    y = "Deaths"
  ) +
  theme_minimal()
```

Vision Score vs Kills

```
vscore_kills_plot
```

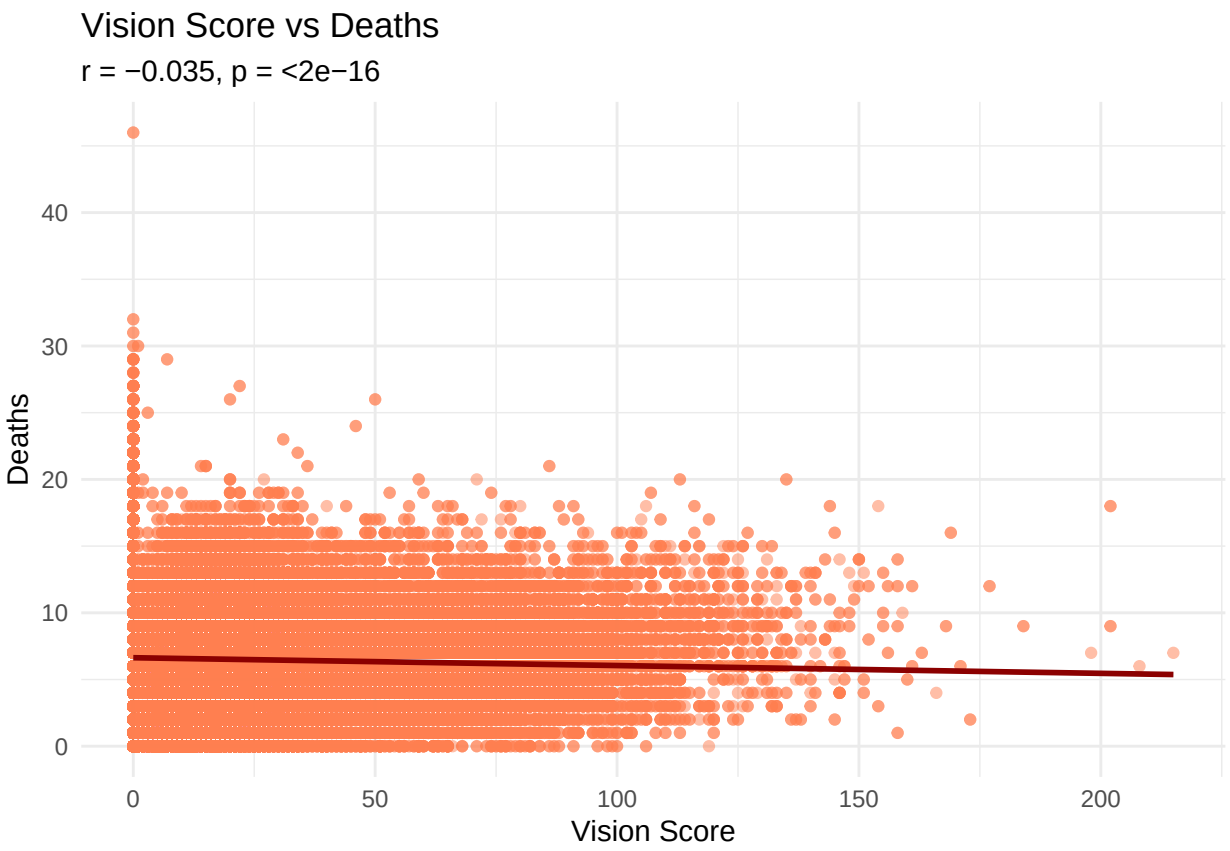


Vision Score vs Kills Conlusion

- The correlation is slightly negative. Players with higher vision scores tend to have fewer kills.
- Heteroscedasticity means the strength of this relationship varies across the range of Vision Score.

Vision Score vs Deaths

```
vscore_deaths_plot
```



Vision Score vs Deaths Conlusion

Although the correlation between Vision Score and Deaths is statistically significant (r = -0.035, p < 2e-16) due to the very large sample size, the correlation coefficient is negligible. Therefore, there is no practically meaningful linear relationship between Vision Score and Deaths.

Two Way Anova

To determine the best runes to take per lane, we run a Two Way ANOVA test. By visualizing the results as a series of boxplots we can see the highest damage runes per lane and see which over and underperform. The high F values from our results show that what runes/keystones you and, and the lane you choose do have a significant effect on you damage dealt. Although it should be noted that the large amount of residuals show that damage is mostly determined by other factors in the game such as gold gained and individual differences in player skill.

Two Way Anova

```
library(car)
#Definition list for each rune name and ID
rune_stones_lookup <- data.frame(
  rune = c(
    "Guardian", "SummonAery", "Electrocute", "DarkHarvest", "Conquerer",
```

```
    "ArcaneComet", "FirstStrike", "LethalTempo", "PressTheAttack",
    "Aftershock", "PhaseRush", "HailOfBlades", "FleetFootwork",
    "GlacialAugment", "GraspOfTheUndying", "UnsealedSpellbook"
  ),

  ID = c(
    "8465", "8214", "8112", "8128", "8010",
    "8229", "8369", "8008", "8005",
    "8439", "8230", "9923", "8021", "8351",
    "8437", "8360"
  ),

  tree = c(
    "Resolve1", "Sorcery1", "Domination1", "Domination2", "Precision1",
    "Sorcery2", "Inspiration1", "Precision2", "Precision3",
    "Resolve2", "Sorcery3", "Domination3", "Precision4", "Inspiration2",
    "Resolve3", "Inspiration3"
  )
)

keystone_order <- c(
  #Precision
  "Conquerer", "LethalTempo", "PressTheAttack", "FleetFootwork",
  #Domination
  "DarkHarvest", "Electrocute", "HailOfBlades",
  #Inspiration
  "GlacialAugment", "UnsealedSpellbook", "FirstStrike",
  #Sorcery
  "ArcaneComet", "SummonAery", "PhaseRush",
  #Resolve
  "GraspOfTheUndying", "Guardian", "Aftershock"
)
#Define Color Pallette for graphing and relations of color -> Rune
color_pallette <- c(
  "Precision1" = "#ffee00",
  "Precision2" = "#e2d524",
  "Precision3" = "#bbb01e",
  "Precision4" = "#8f8717",
  "Domination1" = "#af1919",
  "Domination2" = "#d81d1d",
  "Domination3" = "#811212",
  "Inspiration1" = "#267580",
  "Inspiration2" = "#36b0c0",
  "Inspiration3" = "#2c92a0",
  "Sorcery1" = "#700a8f",
  "Sorcery2" = "#8d0cb4",
  "Sorcery3" = "#5c0975",
  "Resolve1" = "#077e1b",
  "Resolve2" = "#056114",
  "Resolve3" = "#08a522"
)
rune_colors <- color_pallette[rune_stones_lookup$tree]
names(rune_colors) <- rune_stones_lookup$rune
```

```r
#Filter out ourliers in data & missing data entries
#Some games have extremely high damage due to non-realistic playstyles
dmg_dealt <- match_stats$DmgDealt
q1 <- quantile(dmg_dealt, 0.25)
q3 <- quantile(dmg_dealt, 0.75)
iqr <- q3 - q1

lower_bound <- q1 - 1.5 * iqr
upper_bound <- q3 + 1.5 * iqr

filtered <- match_stats[
  dmg_dealt >= lower_bound & dmg_dealt <= upper_bound &
    match_stats$PrimaryKeyStone != "0" & match_stats$Lane != "NONE",
]

filtered$PrimaryKeyStone <- factor(filtered$PrimaryKeyStone)
filtered$Lane <- factor(filtered$Lane)

# Levenes Test
levene_test <- leveneTest(DmgDealt ~ PrimaryKeyStone * Lane, data = filtered)

# 2 Way ANOVA on Keystone and Role
model <- aov(DmgDealt ~ PrimaryKeyStone * Lane, data = filtered)
summary(model)
```

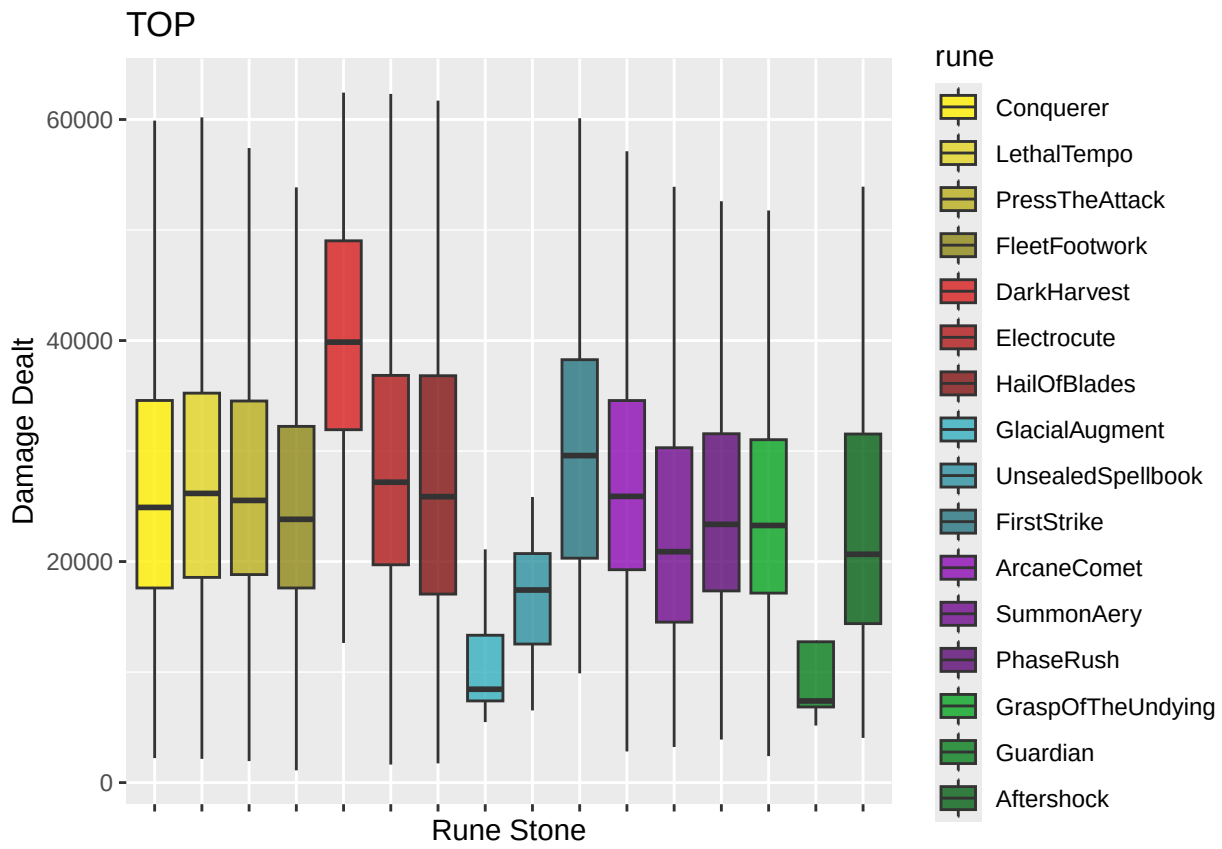
```
##                          Df    Sum Sq   Mean Sq F value Pr(>F)
## PrimaryKeyStone          15 2.299e+12 1.533e+11 1287.77 <2e-16 ***
## Lane                      4 5.753e+11 1.438e+11 1208.36 <2e-16 ***
## PrimaryKeyStone:Lane     60 3.482e+11 5.804e+09   48.76 <2e-16 ***
## Residuals            107096 1.275e+13 1.190e+08
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Results show a high F value meaning that KeyStone has a large
# effect on average damage even with the large amount of noise
# (random player variation still makes most of the difference)

# P-Values all very low so all very signifigant
# Residuals are very high, meaning that KeyStone and
# Lane are primarily not the biggest effects and
# most is left up to other factors
# (Likely player randomness and maybe gold earned)

#Find Individual stats
key_stone_stats <- TukeyHSD(model, "PrimaryKeyStone")
#key_stone_stats
lane_stats <- TukeyHSD(model, "Lane")
#lane_stats

# Make sure factors are properly set
filtered$PrimaryKeyStone <- factor(filtered$PrimaryKeyStone)
filtered$Lane <- factor(filtered$Lane)
```

```r
# Relate Keystone ID, Name, and Colors
filtered <- filtered |>
  left_join(rune_stones_lookup, by = c("PrimaryKeyStone" = "ID"))

# Reorganizes data to group by keystone order (Domination, Precision, etc.)
filtered$rune <- factor(filtered$rune, levels = keystone_order)

# Loop over lanes and create graph for each
lanes <- unique(filtered$Lane)
```

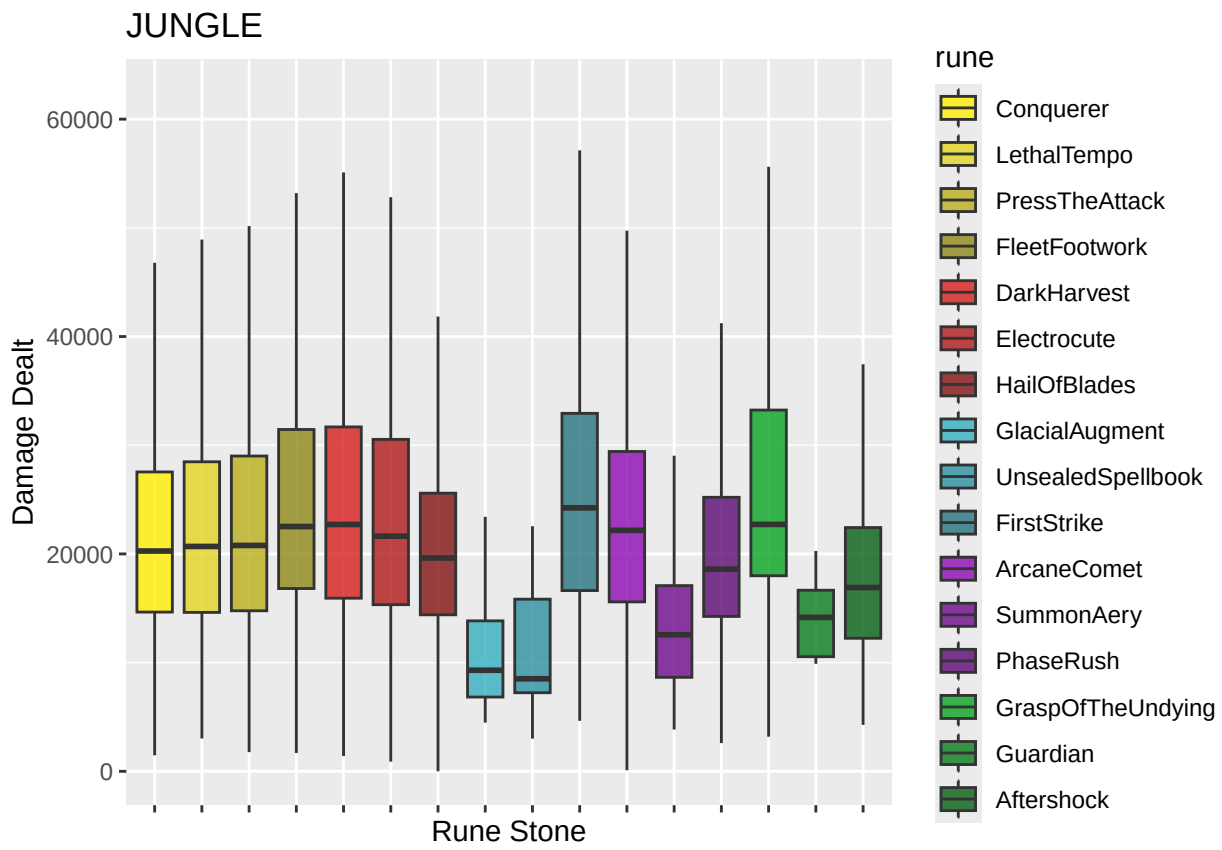
Top - Two Way Anova

```r
df_lane <- filtered[filtered$Lane == "TOP", ]
  plot <- ggplot(df_lane, aes(x = rune, y = DmgDealt, fill = rune)) +
    geom_boxplot(alpha = 0.8, outlier.shape = NA) +
    scale_fill_manual(values = rune_colors) +
    xlab("Rune Stone") +
    ylab("Damage Dealt") +
    ggtitle("TOP") +
    theme(axis.text.x = element_blank())

print(plot)
```



Jungle - Two Way Anova

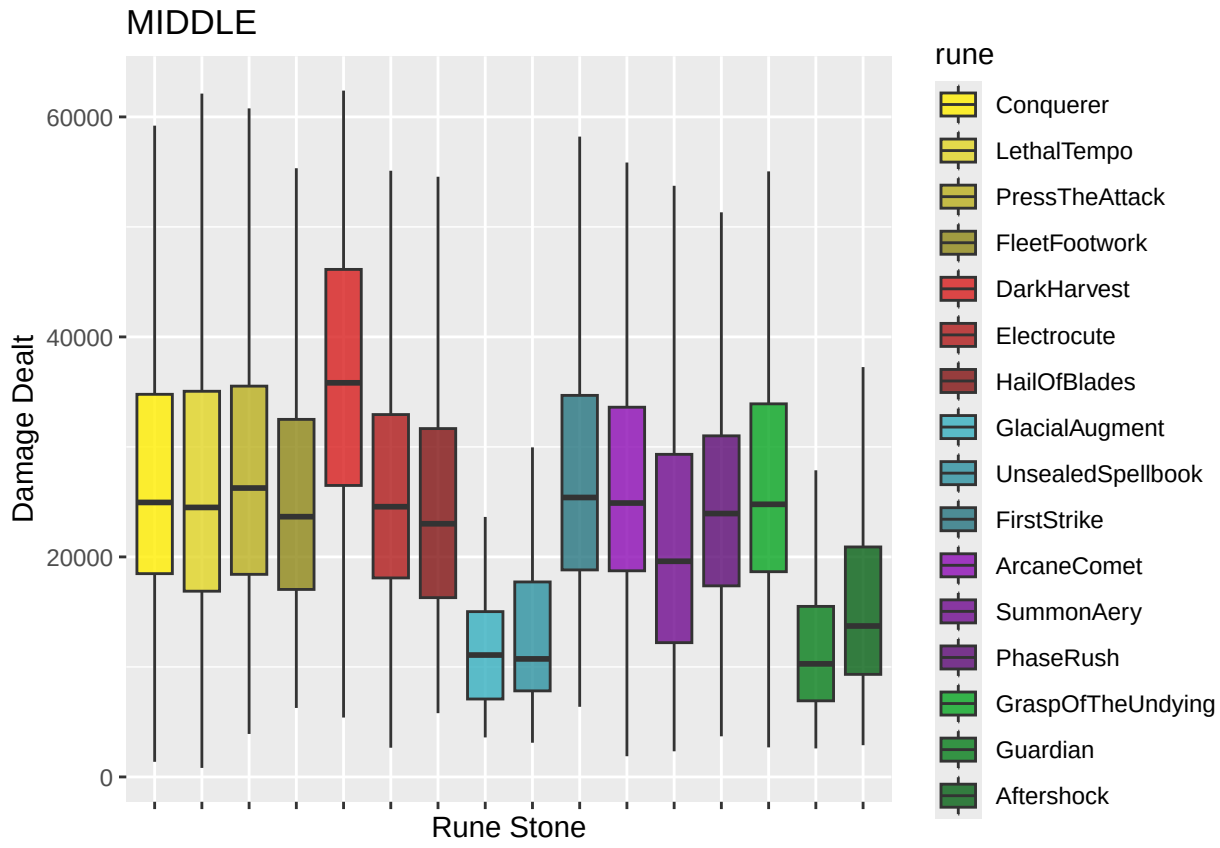
```
df_lane <- filtered[filtered$Lane == "JUNGLE", ]
  plot <- ggplot(df_lane, aes(x = rune, y = DmgDealt, fill = rune)) +
    geom_boxplot(alpha = 0.8, outlier.shape = NA) +
    scale_fill_manual(values = rune_colors) +
    xlab("Rune Stone") +
    ylab("Damage Dealt") +
    ggtitle("JUNGLE") +
    theme(axis.text.x = element_blank())

print(plot)
```



Middle - Two Way Anova

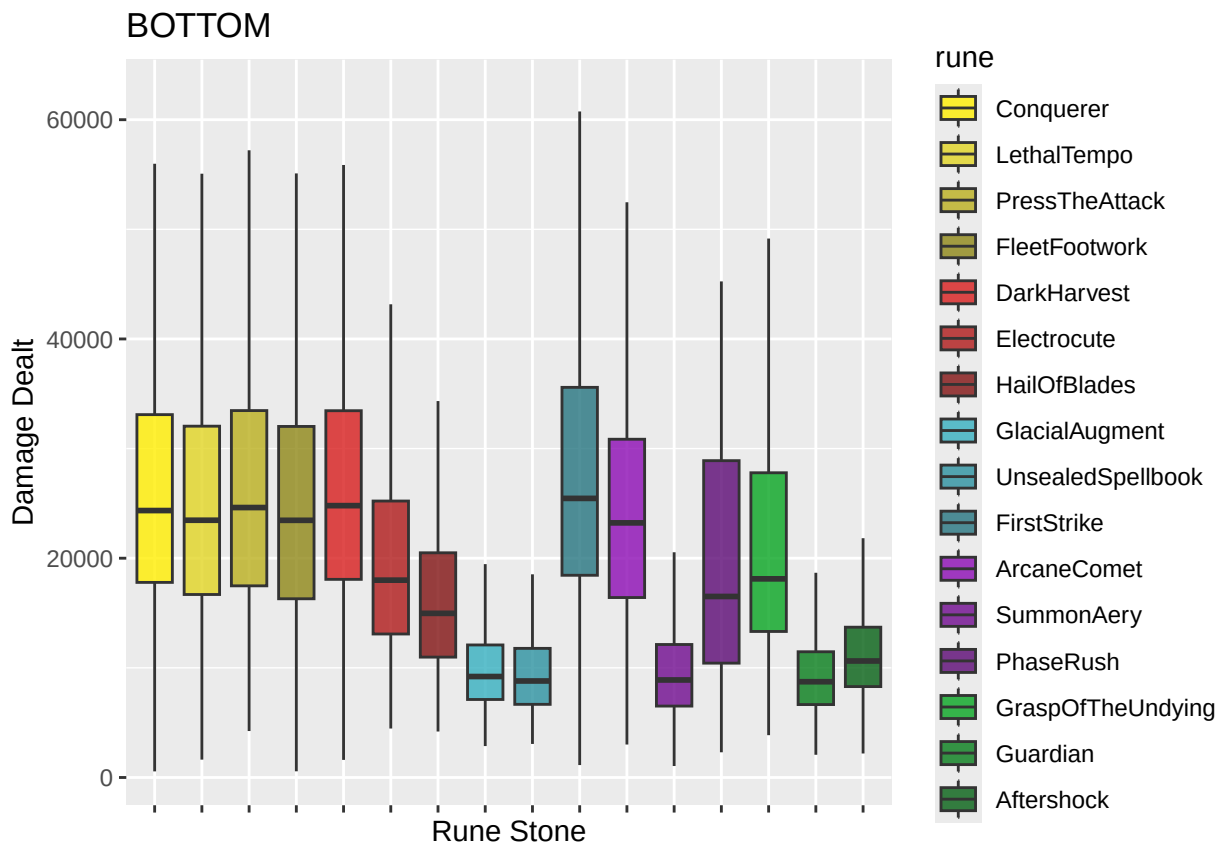
```
df_lane <- filtered[filtered$Lane == "MIDDLE", ]
  plot <- ggplot(df_lane, aes(x = rune, y = DmgDealt, fill = rune)) +
    geom_boxplot(alpha = 0.8, outlier.shape = NA) +
    scale_fill_manual(values = rune_colors) +
    xlab("Rune Stone") +
    ylab("Damage Dealt") +
    ggtitle("MIDDLE") +
```

```
    theme(axis.text.x = element_blank())

print(plot)
```



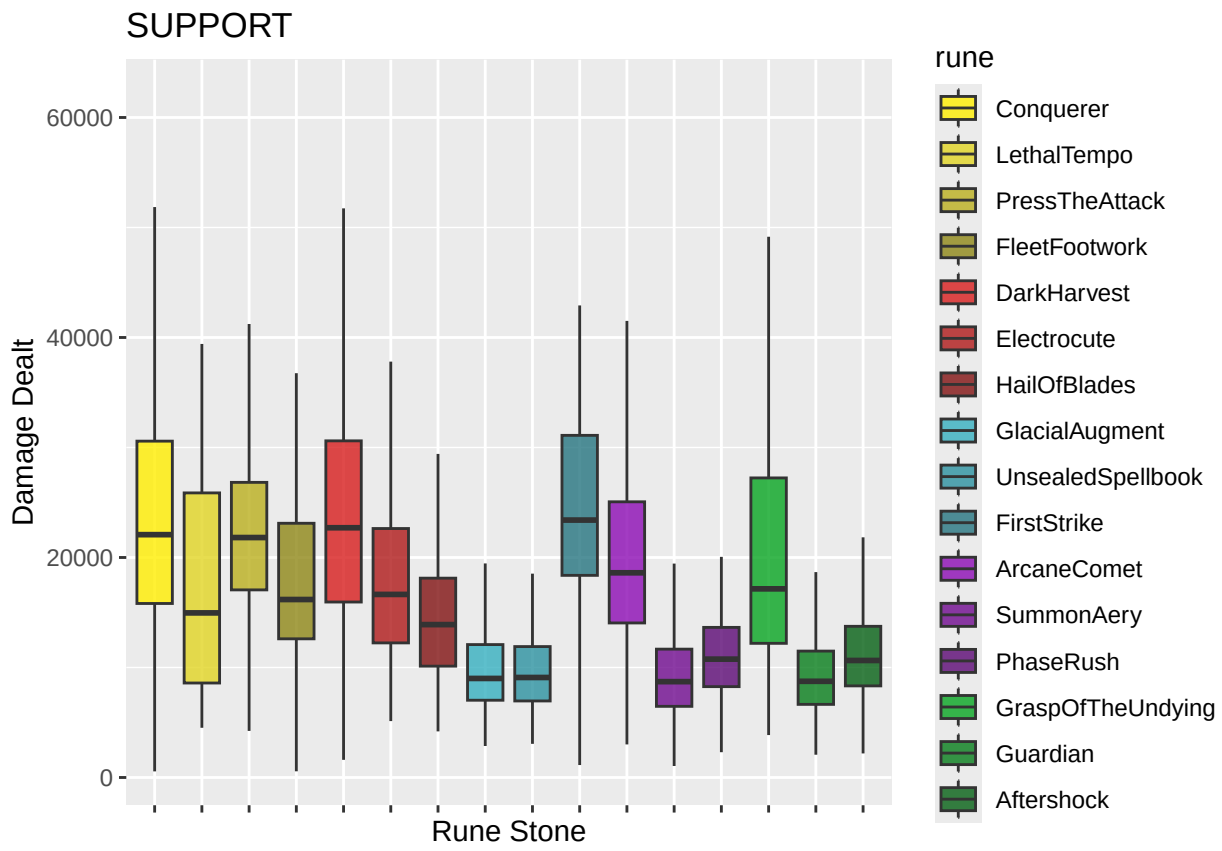
Bottom - Two Way Anova

```
df_lane <- filtered[filtered$Lane == "BOTTOM", ]
  plot <- ggplot(df_lane, aes(x = rune, y = DmgDealt, fill = rune)) +
    geom_boxplot(alpha = 0.8, outlier.shape = NA) +
    scale_fill_manual(values = rune_colors) +
    xlab("Rune Stone") +
    ylab("Damage Dealt") +
    ggtitle("BOTTOM") +
    theme(axis.text.x = element_blank())

print(plot)
```



Support - Two Way Anova

```
df_lane <- filtered[filtered$Lane == "SUPPORT", ]
  plot <- ggplot(df_lane, aes(x = rune, y = DmgDealt, fill = rune)) +
    geom_boxplot(alpha = 0.8, outlier.shape = NA) +
    scale_fill_manual(values = rune_colors) +
    xlab("Rune Stone") +
    ylab("Damage Dealt") +
    ggtitle("SUPPORT") +
    theme(axis.text.x = element_blank())

print(plot)
```



Play Style Clustering

There are 5 roles in LoL (Top, Middle, Bottom, Jungle, Support), which can be combined into 3 distinct playstyle groups (Laner, Jungle, Support).

Can K-means clustering be used to cluster player data into these three playstyle groups?

Features: Damage Dealt, Damage Taken, Total Gold, Kills, Deaths, Assists, Vision Score, Dragon Kills, and Baron Kills.

```
cluster_match_stats <- match_stats |>
  filter(Lane != "NONE")

features <- cluster_match_stats[, c("DmgDealt", "DmgTaken",
                            "TotalGold", "kills", "deaths", "assists",
                            "visionScore", "DragonKills", "BaronKills")]


#Scale features for clustering (Necessary when measuring distance)
scaled_features <- scale(features)

km <- kmeans(scaled_features, centers = 3, nstart = 25)


cluster_match_stats$cluster <- km$cluster
```

```r
centers <- as.data.frame(km$centers)
centers$cluster <- rownames(centers)


centers_melt <- melt(centers, id = "cluster")

play_style_bar_graph <- ggplot(
  centers_melt,
  aes(x = variable, y = value, fill = cluster)
) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Cluster Centers (Scaled Values)",
       x = "Feature", y = "Scaled Value") +
  theme(axis.text.x = element_text(size = 8, angle = 45, hjust = 1))

# Prepare table
cluster_role_percent <- cluster_match_stats |>
  mutate(
    LaneGroup = case_when(
      Lane %in% c("TOP", "MIDDLE", "BOTTOM") ~ "Laner",
      Lane == "JUNGLE" ~ "Jungle",
      Lane == "SUPPORT" ~ "Support",
      TRUE ~ Lane
    )
  ) |>
  group_by(cluster, LaneGroup) |>
  summarise(count = n(), .groups = "drop") |>
  group_by(cluster) |>  # Regroup by cluster only
  mutate(percent = 100 * count / sum(count)) |>
  ungroup() |>
  select(cluster, LaneGroup, percent)

# Make stacked bar chart
cluster_comp_stacked_bar_graph <- ggplot(cluster_role_percent,
                            aes(x = factor(cluster), y = percent,
                                fill = LaneGroup)) +
  geom_bar(stat = "identity") +
  scale_fill_manual(values = c(
    "Laner" = "#E69F00",      # Orange
    "Jungle" = "#009E73",    # Green
    "Support" = "#9E4CBF"     # Purple
  )) +
  scale_y_continuous(labels = percent_format(scale = 1)) +
  labs(
    title = "Lane Composition per Cluster",
    x = "Cluster",
    y = "Percentage",
    fill = "Lane Group"
  ) +
  theme_minimal()
```

```r
# Pivot to wide format table
cluster_role_percent <- cluster_role_percent |>
  pivot_wider(
    names_from = LaneGroup,
    values_from = percent,
    values_fill = 0
  )
```

Important Considerations for K-Means

- Features are numerical
- Scale of features matter
- K should be meaningful within the context of the underlying structure of data

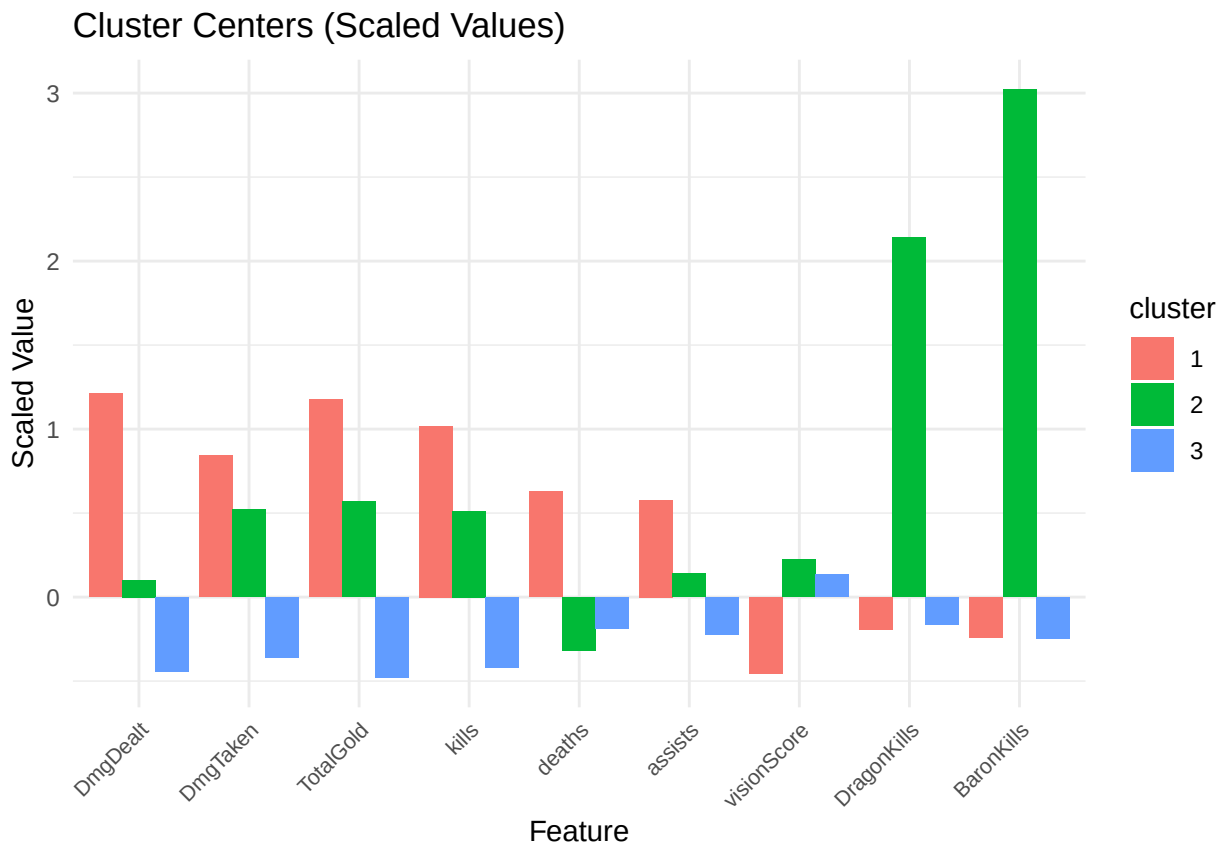
Play Style Bar Graph Explanation

The bar graph on the following slide shows where the center of each cluster converged.

- One cluster converges on higher overall damage dealt/taken, total gold, kills, deaths, and assists (aligns with Laner's playstyle)
- Another cluster converges on a relatively higher vision score and very low total gold (aligns with Support's playstyle)
- The other cluster converges on very high dragon kills and baron kills (aligns with Jungle's playstyle)

Play Style Bar Graph

```r
play_style_bar_graph
```

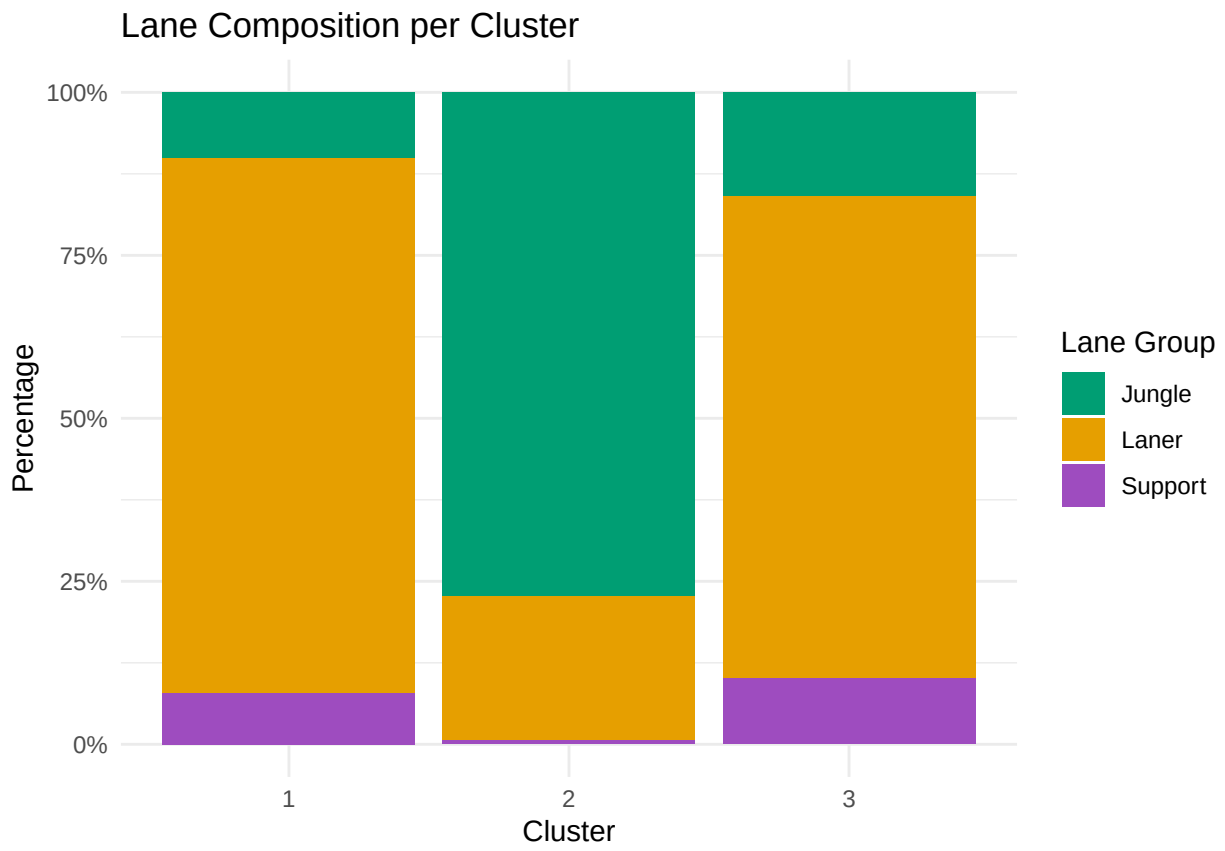


Cluster Composition Stacked Bar Graph Explanation

The stacked bar graph on the following slide shows the composition of each cluster. One cluster has a high percentage of Jungle players. The other two clusters are very similar in composition with a majority of Laner players.

Cluster Composition Stacked Bar Graph

```
cluster_comp_stacked_bar_graph
```



K-Means Cluster Conclusion

In conclusion, k-means was not able to separate Laner and Support players into distinct clusters well based on these set of features and implementation.

Areas for improvement:
- The biggest distinction between support and other roles is vision score per total gold. Adding this ratio as a feature may improve clustering results.
- Laner is composed of three less distinct roles (Top, Middle, Bottom) that appear in each match. Therefore, Laner may have a higher variance or density, which may explain why the Laners were split into 2 clusters. Increasing K to 4 or 5 may resolve this issue.

Team Playstyle Logistical Regression

Using a Logistical Regression test we can determine the optimal playstyle for a team based on a variety of map objectives and team kills

Team Playstyle Logistical Regression

```
#Logistic Regression on a bunch of team Data
blue_barons <- team_stats$BlueBaronKills
blue_dragons <- team_stats$BlueDragonKills
blue_heralds <- team_stats$BlueRiftHeraldKills
```

```r
blue_kills <- team_stats$BlueKills
blue_towers <- team_stats$BlueTowerKills

model <- glm(
  BlueWin ~ blue_barons + blue_dragons + blue_heralds + blue_kills + blue_towers,
  data = team_stats,
  family = binomial
)

summary(model)
```

```
##
## Call:
## glm(formula = BlueWin ~ blue_barons + blue_dragons + blue_heralds +
##     blue_kills + blue_towers, family = binomial, data = team_stats)
##
## Coefficients:
##                Estimate Std. Error  z value Pr(>|z|)
## (Intercept)  -3.2029282  0.0307501 -104.160  < 2e-16 ***
## blue_barons  -0.1237097  0.0244357   -5.063 4.13e-07 ***
## blue_dragons  0.0790852  0.0105454    7.499 6.41e-14 ***
## blue_heralds -0.1706887  0.0241291   -7.074 1.51e-12 ***
## blue_kills    0.0681225  0.0006756  100.839  < 2e-16 ***
## blue_towers   0.2426111  0.0043327   55.995  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 93662  on 68675  degrees of freedom
## Residual deviance: 68016  on 68670  degrees of freedom
## AIC: 68028
##
## Number of Fisher Scoring iterations: 4
```

Team Playstyle Logistical Regression

Looking at this summary we can see the individual effects each factor has on your chance of winning. This shows that towers are by far the most important thing to be playing for as a team with dragons closly behind. Notice the negative coeffiecnt for Barons and Rift Herald takes. This is a side effect of including Tower kills in the model. As the model holds the other factors constant and tests these two it sees a lowering of winrate (Very low data size). This makes sense as these objectives are highly related to taking a tower, and thus taking this objective and failing to secure a tower signify a larger issue with the game state. Using this data we can create a basic win-rate predictor by feeding stats in and getting a value out.

Team Playstyle Logistical Regression

```r
# Results give us a legit function,
# need to convert to individual winrate multipliers
odd_mults <- exp(coef(model)[-1])
odd_mults
```

```
##  blue_barons blue_dragons blue_heralds   blue_kills  blue_towers
##    0.8836364    1.0822965    0.8430840    1.0704965    1.2745729
```

```
#Probability Conversion = Mults/1+mult Can build predictor model off of this
winrate <- 0.5 #50% chance of winning a game
barons <- 1
dragons <- 0
heralds <- 0
kills <- 0
towers <- 1
odds <- winrate / (1 - winrate) *
  odd_mults["blue_barons"]^barons *
  odd_mults["blue_dragons"]^dragons *
  odd_mults["blue_heralds"]^heralds *
  odd_mults["blue_kills"]^kills *
  odd_mults["blue_towers"]^towers

probability <- odds / (1 + odds)
probability
```

```
## blue_barons
##   0.5296904
```

Thank You!