LoL-Statistics

Andrew Roddy

2025-11-24

MatchStatsTbl.csv

Import code and print off column and row count

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

```
## There are 150505 rows.
```

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

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

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

```
## There are 68676 rows.
```

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

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

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

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

```r
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"))
```

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

names <- colnames(match_stats)
print(names)
```

```
##  [1] "MatchStatsId"         "SummonerMatchFk"      "MinionsKilled"
##  [4] "DmgDealt"             "DmgTaken"             "TurretDmgDealt"
##  [7] "TotalGold"            "Lane"                 "Win"
## [10] "item1"                "item2"                "item3"
## [13] "item4"                "item5"                "item6"
## [16] "kills"                "deaths"               "assists"
## [19] "PrimaryKeyStone"      "CurrentMasteryPoints" "EnemyChampionFk"
## [22] "DragonKills"          "BaronKills"           "visionScore"
## [25] "SummonerFk"           "MatchFk"              "ChampionFk"
## [28] "ChampionName"
```

Multiple regression to test if

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.5.2
```

```r
library(scales)
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

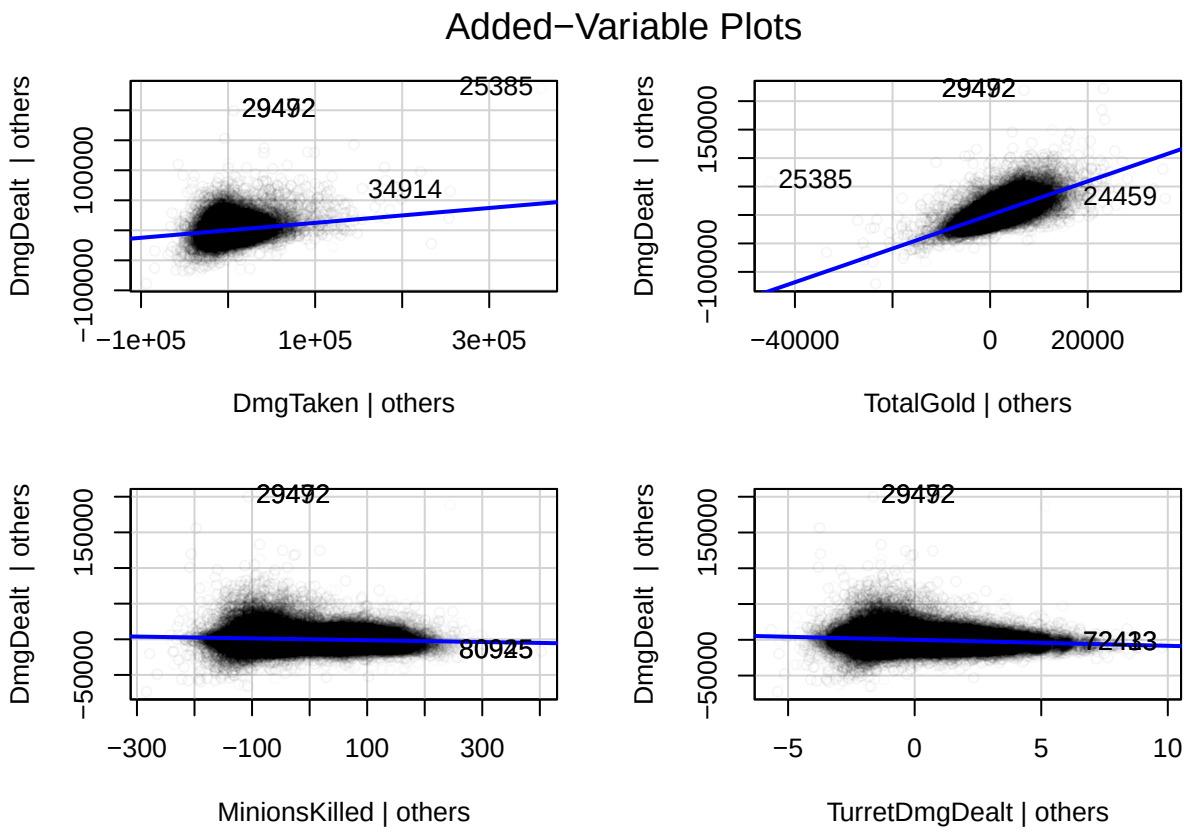
```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```r
# make simple regression model
multiple_regression <- lm(DmgDealt ~
    DmgTaken + TotalGold + MinionsKilled + TurretDmgDealt,
```

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

```
##
## Call:
## lm(formula = DmgDealt ~ DmgTaken + TotalGold + MinionsKilled +
##     TurretDmgDealt, data = match_stats)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -77051  -5559   -702   4379 198600
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -1.171e+04  7.372e+01 -158.85   <2e-16 ***
## DmgTaken        1.241e-01  1.828e-03   67.88   <2e-16 ***
## TotalGold       2.954e+00  7.589e-03  389.27   <2e-16 ***
## MinionsKilled  -1.272e+01  3.176e-01  -40.03   <2e-16 ***
## TurretDmgDealt -8.420e+02  1.473e+01  -57.15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 9929 on 150500 degrees of freedom
## Multiple R-squared:  0.6843, Adjusted R-squared:  0.6843
## F-statistic: 8.154e+04 on 4 and 150500 DF,  p-value: < 2.2e-16
```

```
# Creates the variable plots
#000000 point color
#0.01 : 1 percent opacity

# Commented out for now
avPlots(multiple_regression, col = alpha("#000000", 0.02))
```




```r
library(ggplot2)
library(dplyr)


#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"
  )
```

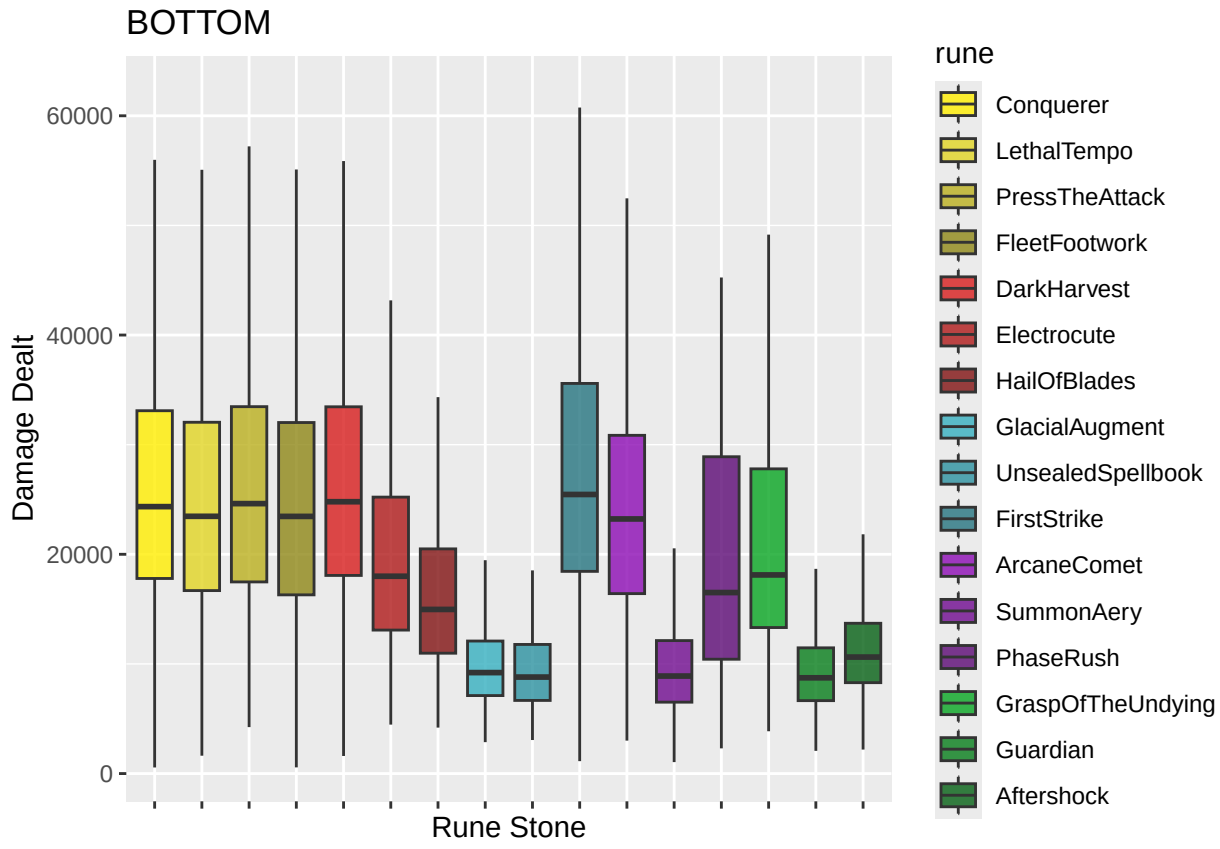
```
)

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

#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",
]

# 2 Way ANOVA on Keystone and Role
filtered$PrimaryKeyStone <- factor(filtered$PrimaryKeyStone)
```

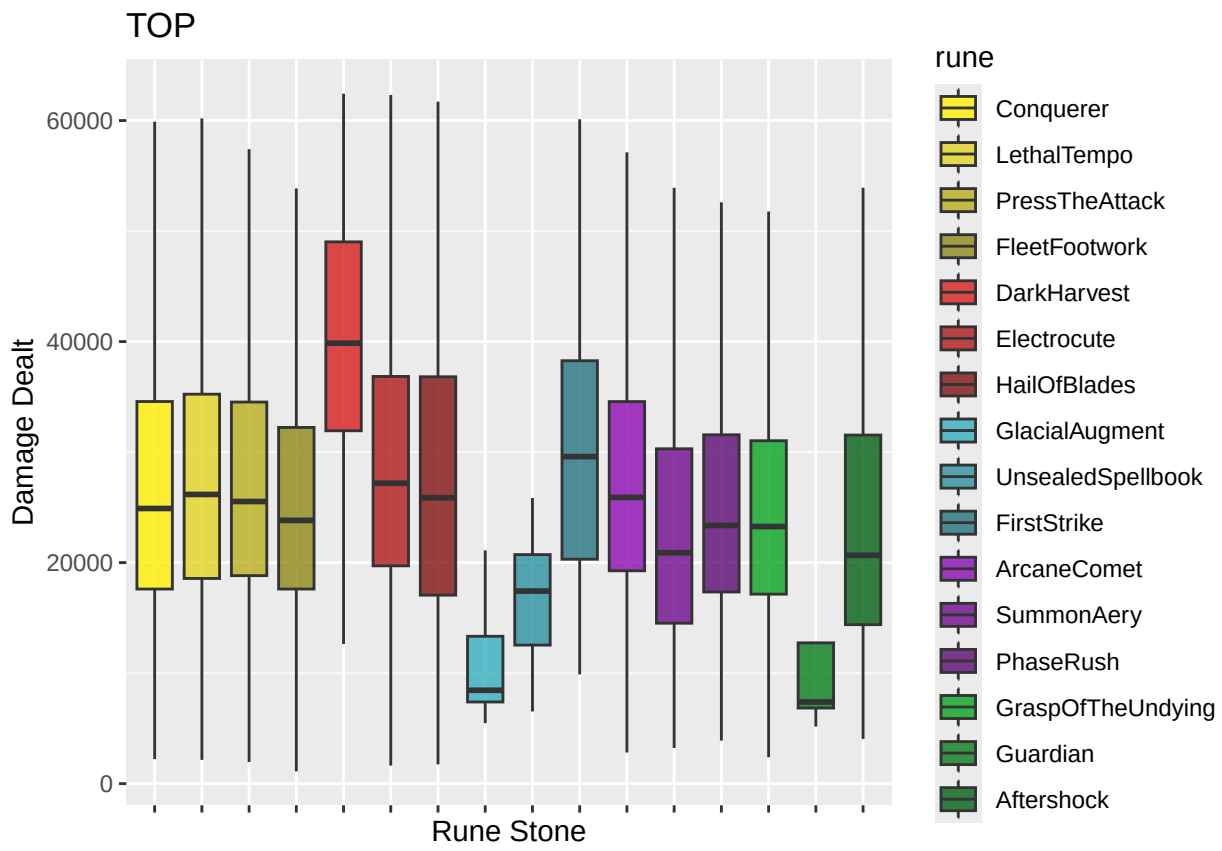
```r
filtered$Lane <- factor(filtered$Lane)
model <- aov(DmgDealt ~ PrimaryKeyStone * Lane, data = filtered)
summary(model)
```

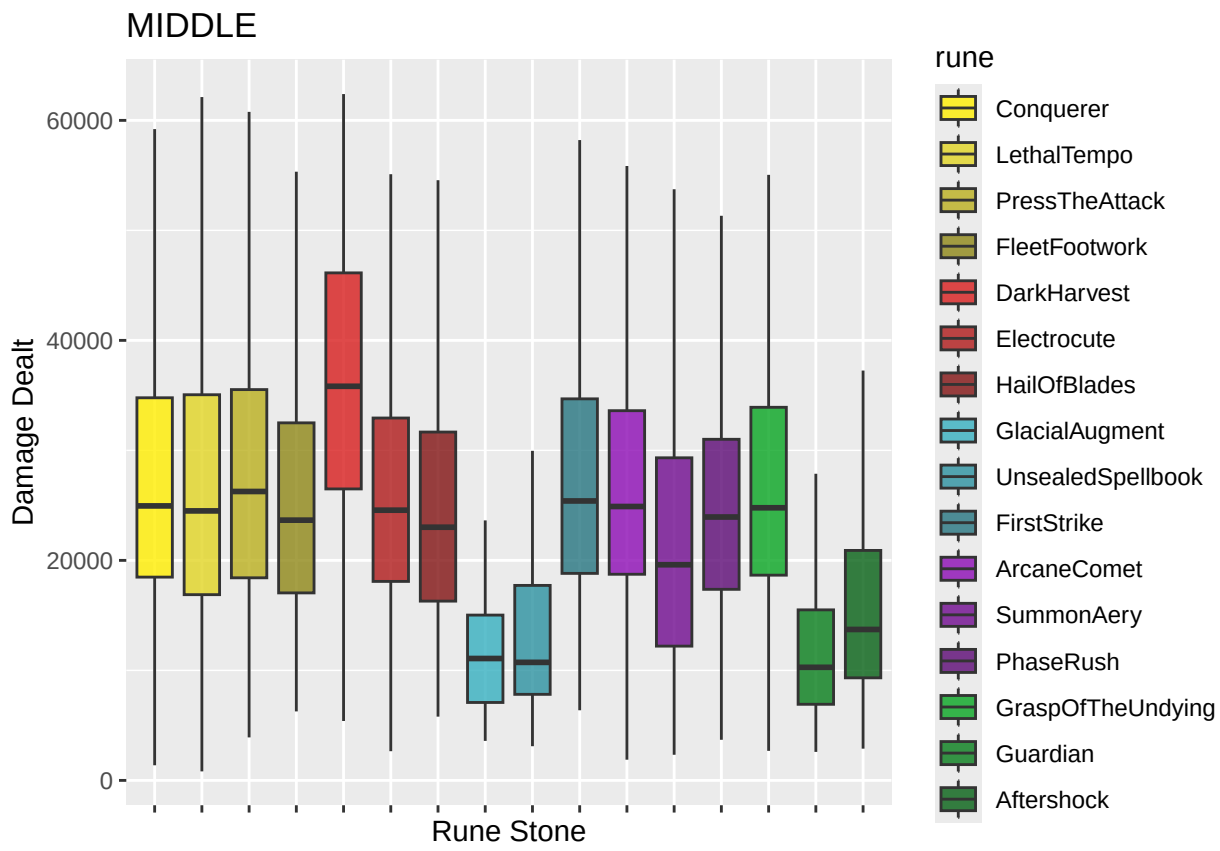
```
##                            Df    Sum Sq   Mean Sq F value Pr(>F)
## PrimaryKeyStone            15 2.299e+12 1.533e+11 1287.77 <2e-16 ***
## Lane                        4 5.753e+11 1.438e+11 1208.36 <2e-16 ***
## PrimaryKeyStone:Lane       60 3.482e+11 5.804e+09   48.76 <2e-16 ***
## Residuals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

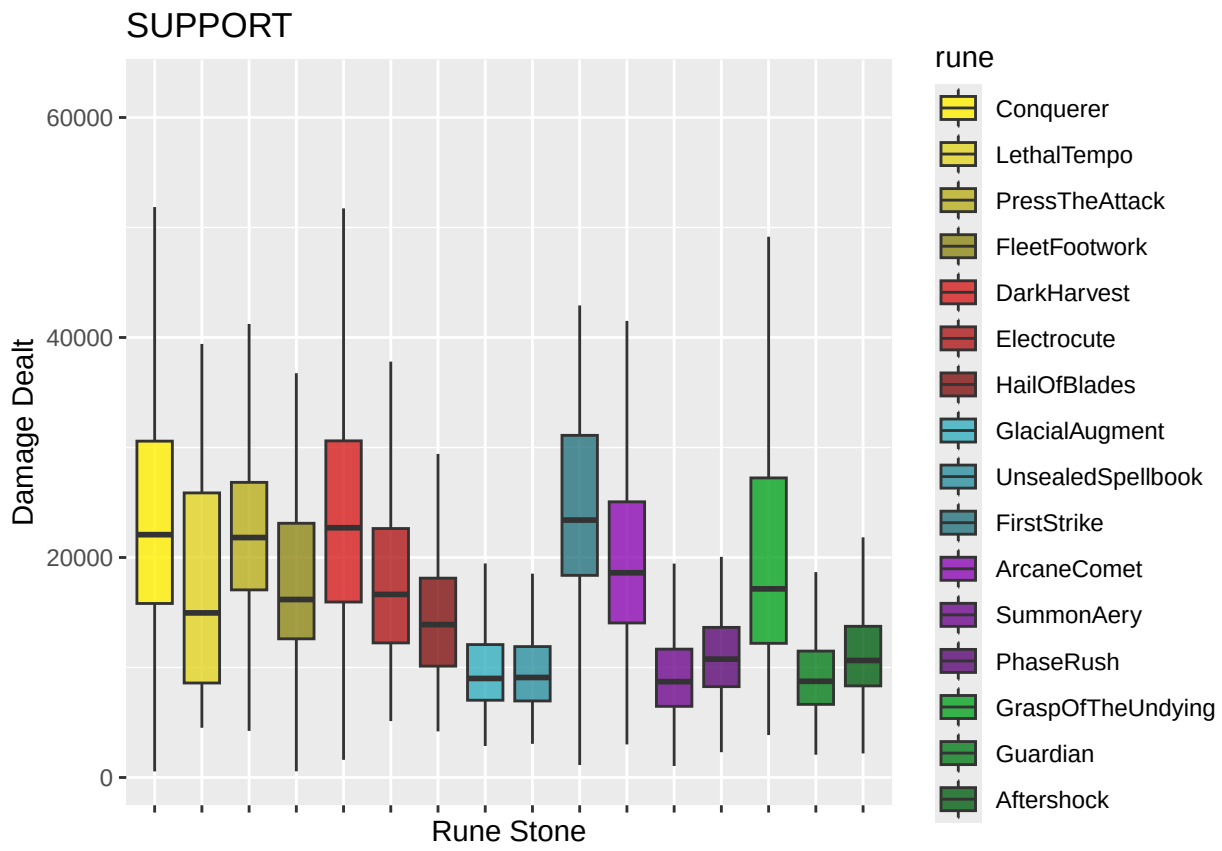
```r
# Results show a high F value meaning that KeyStone has a large
# effect on average damage even with the large amount of noise
# (random player variation still makes most of the difference)

# 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)

#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)
for (lane in lanes){

  df_lane <- filtered[filtered$Lane == lane, ]
  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(lane) +
    theme(axis.text.x = element_blank())

  print(plot)
```

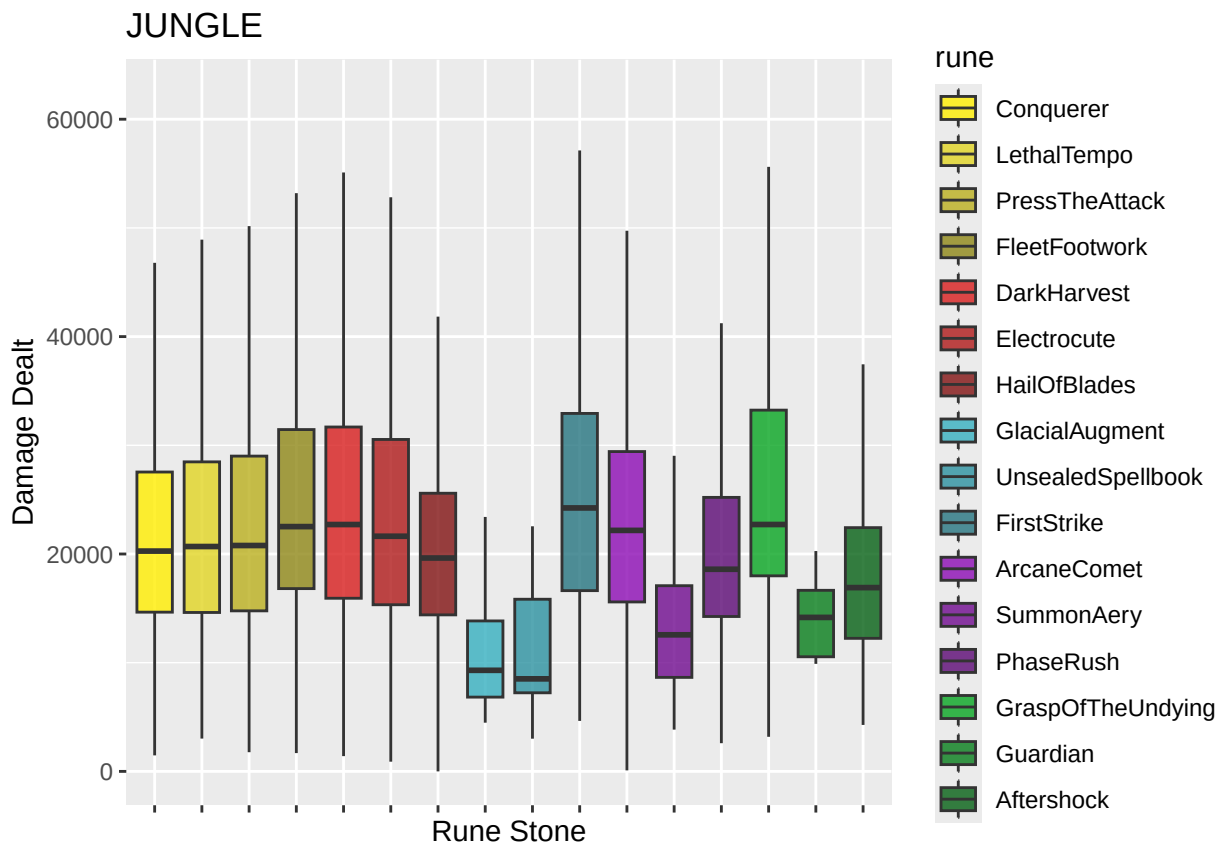
```
}
```


BOTTOM
Damage Dealt
60000
40000
20000
0
Rune Stone
rune
Conquerer
LethalTempo
PressTheAttack
FleetFootwork
DarkHarvest
Electrocute
HailOfBlades
GlacialAugment
UnsealedSpellbook
FirstStrike
ArcaneComet
SummonAery
PhaseRush
GraspOfTheUndying
Guardian
Aftershock


TOP
Damage Dealt
60000
40000
20000
0
Rune Stone
rune
Conquerer
LethalTempo
PressTheAttack
FleetFootwork
DarkHarvest
Electrocute
HailOfBlades
GlacialAugment
UnsealedSpellbook
FirstStrike
ArcaneComet
SummonAery
PhaseRush
GraspOfTheUndying
Guardian
Aftershock


MIDDLE
Damage Dealt
60000
40000
20000
0
Rune Stone
rune
Conquerer
LethalTempo
PressTheAttack
FleetFootwork
DarkHarvest
Electrocute
HailOfBlades
GlacialAugment
UnsealedSpellbook
FirstStrike
ArcaneComet
SummonAery
PhaseRush
GraspOfTheUndying
Guardian
Aftershock


SUPPORT
Damage Dealt
60000
40000
20000
0
Rune Stone
rune
Conquerer
LethalTempo
PressTheAttack
FleetFootwork
DarkHarvest
Electrocute
HailOfBlades
GlacialAugment
UnsealedSpellbook
FirstStrike
ArcaneComet
SummonAery
PhaseRush
GraspOfTheUndying
Guardian
Aftershock




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

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

summary(model)
```

```
##
## Call:
## glm(formula = BlueWin ~ blue_barons + blue_dragons + blue_heralds +
##     blue_kills, family = binomial, data = team_stats)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.8943909  0.0288116 -100.46   <2e-16 ***
## blue_barons   0.3734663  0.0225364   16.57   <2e-16 ***
## blue_dragons  0.3968396  0.0087527   45.34   <2e-16 ***
## blue_heralds  0.3410325  0.0210757   16.18   <2e-16 ***
## blue_kills    0.0694563  0.0006547  106.09   <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: 71484  on 68671  degrees of freedom
## AIC: 71494
##
## Number of Fisher Scoring iterations: 4
```

```
# 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
##     1.452762     1.487117     1.406399     1.071925
```

```
#Probability Conversion = Mults/1+mult Can build predictor model off of this
winrate <- 0.5 #50% chance of winning a game
barons <- -2
dragons <- -1
heralds <- 1
kills <- 5
odds <- winrate / (1 - winrate) *
  odd_mults["blue_barons"]^barons *
  odd_mults["blue_dragons"]^dragons *
  odd_mults["blue_heralds"]^heralds *
  odd_mults["blue_kills"]^kills

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

```
## blue_barons
##   0.3880638
```

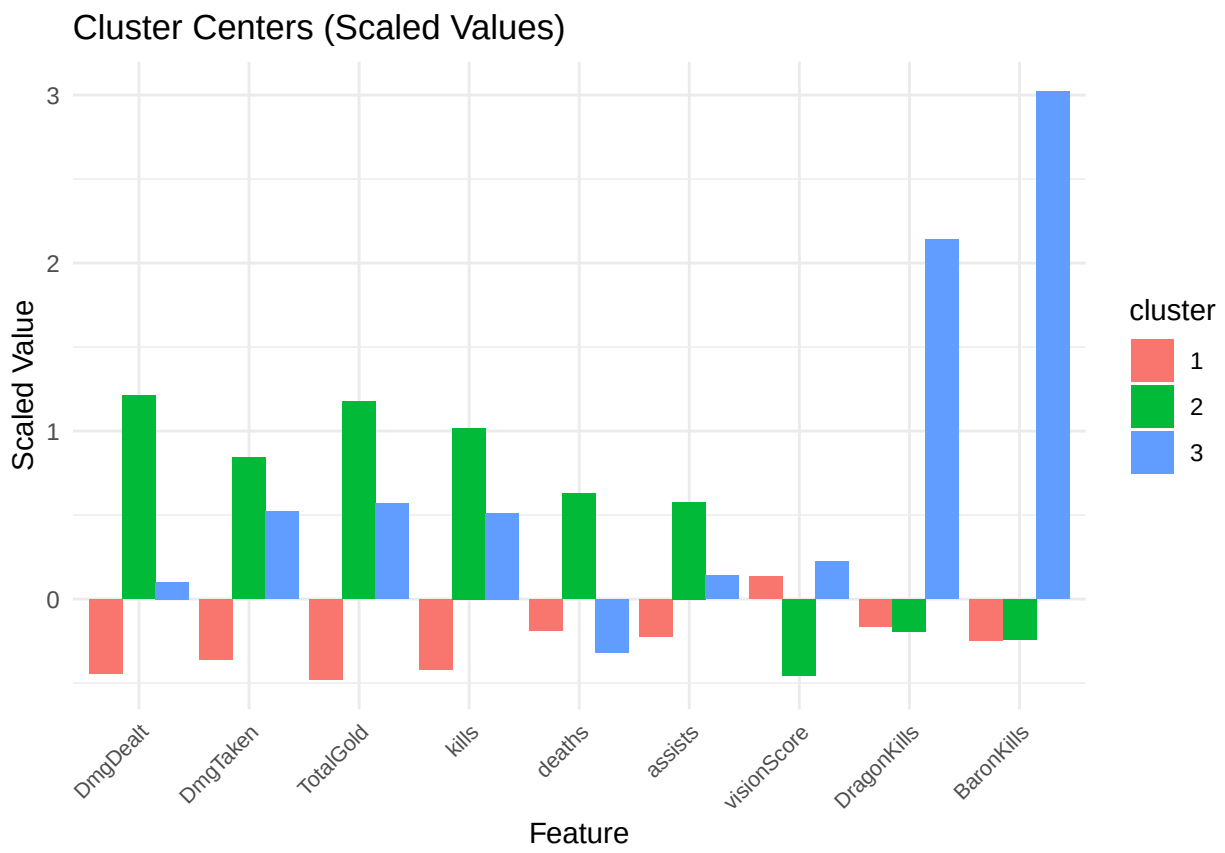
```
tab <- table(match_stats$ChampionName, match_stats$Win)
chi_square_test <- chisq.test(tab)
chi_square_test
```

```
##
##  Pearson's Chi-squared test
##
## data:  tab
## X-squared = 444.22, df = 170, p-value < 2.2e-16
```

```
library(ggplot2)
library(reshape2)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':
##
##     smiths
```

```
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



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


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

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))

print(bar_graph)
```




```
cluster_role_percent <- cluster_match_stats |>
  # Remove rows where Lane = "None"
  filter(Lane != "NONE") |>

  # Rename and group lane roles
  mutate(
    LaneGroup = case_when(
      Lane %in% c("TOP", "MIDDLE", "BOTTOM") ~ "Laner",
      Lane == "JUNGLE" ~ "Jungle",
      Lane == "SUPPORT" ~ "Support",
      TRUE ~ Lane   # fallback if unexpected values appear
    )
  ) |>

  # Count by cluster and lane group
  group_by(cluster, LaneGroup) |>
  summarise(count = n()) |>

  # Convert to percent inside each cluster
  mutate(percent = 100 * count / sum(count)) |>

  # Keep important information
  select(cluster, LaneGroup, percent) |>

  # Pivot to wide format
  pivot_wider(
```

```
    names_from = LaneGroup,
    values_from = percent,
    values_fill = 0
  )
```

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

```
cluster_role_percent
```

```
## # A tibble: 3 x 4
## # Groups:   cluster [3]
##   cluster Jungle Laner Support
##     <int>  <dbl> <dbl>   <dbl>
## 1       1   15.9  74.0  10.1
## 2       2   10.1  82.0   7.84
## 3       3   77.2  22.2   0.562
```

```
library(dplyr)
library(tidyr)
# Pull all data from matchstats table and find the optimum
#   build and rune setup for each character.
# Where the build appears more than once.
# First Aggregate all builds that appear more than once
# Using Ahri as a proof baseline ChampID:103
library(dplyr)

# Filter to only Ahri games where the
#   Mode was not some alternate gamemode
#   (Normal Summoners Rift Only)
ahri_data <- match_stats |> filter(ChampionName == "Ahri")
ahri_data <- ahri_data |> filter(Lane != "None" & PrimaryKeyStone != 0)
ahri_wins <- sum(ahri_data$Win == 1)
total_games <- length(ahri_data$Win)

# First find if Ahris winrate is statistically signifigant
prop.test(ahri_wins, total_games)
```

```
##
##  1-sample proportions test with continuity correction
##
## data:  ahri_wins out of total_games, null probability 0.5
## X-squared = 27.856, df = 1, p-value = 1.307e-07
## alternative hypothesis: true p is not equal to 0.5
## 95 percent confidence interval:
##  0.5549201 0.6195286
## sample estimates:
##         p
## 0.5875952
```

```r
# Prop Test shows us that Ahri's winrate is statistically different
# from the baseline of 50% by a large margin over time

test_table <- table(as.factor(ahri_data$PrimaryKeyStone), ahri_data$Win)
chisq.test(test_table)
```

```
## Warning in chisq.test(test_table): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  test_table
## X-squared = 10.806, df = 7, p-value = 0.1473
```

```r
# Ahris Winrate is signifigant between Keystones

# Find highest winrate keystone with more than 5 games played
ahri_keystone_summary <- ahri_data |>
  group_by(PrimaryKeyStone) |>
  summarise(
    games = n(),
    winrate = mean(Win)
  ) |>
  filter(games >= 5) |>
  arrange(desc(winrate))

print(ahri_keystone_summary)
```

```
## # A tibble: 4 x 3
##   PrimaryKeyStone games winrate
##             <int> <int>   <dbl>
## 1            8214    81   0.691
## 2            8128    62   0.613
## 3            8112   761   0.577
## 4            8010     6   0.333
```

```r
# Is difference between them signifigant
test_table <- table(ahri_data$PrimaryKeyStone, ahri_data$Win)
fisher.test(test_table) #Alternative chi^2 because of some runes low sample size
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  test_table
## p-value = 0.1193
## alternative hypothesis: two.sided
```

```r
# May be innaccurate due to nature of game

# Find best items on Ahri
# First need to aggregate item counts
```

```r
unique_items <- unique(
  unlist(
    ahri_data[c(
      "item1", "item2", "item3", "item4", "item5", "item6"
    )]
  )
)

ahri_item_data <- ahri_data |>
  pivot_longer(
    cols = c(
      "item1", "item2", "item3", "item4", "item5", "item6"
    ),
    names_to = "slots",
    values_to = "item"
  )

ahri_item_summary <- ahri_item_data |>
  group_by(item) |>
  summarise(
    games = n(),
    winrate = mean(Win)
  ) |>
  filter(games >= 25) |>
  arrange(desc(winrate))

# p rint(ahri_item_summary, n = nrow(ahri_item_summary))
print(ahri_item_summary)
```

```
## # A tibble: 35 x 3
##     item games winrate
##    <int> <int>   <dbl>
##  1  3171    74   0.797
##  2  3041    61   0.787
##  3  2055    27   0.778
##  4  3175   112   0.741
##  5  2421    35   0.714
##  6  3137    36   0.694
##  7  3135    63   0.683
##  8  3111    53   0.679
##  9  4630    56   0.679
## 10  3108    51   0.667
## # i 25 more rows
```