goJitsu

Cade Michael Lueker

12/29/2021

Goal and Methods

- After taking class in R and basic data analytics I thought that it would be fun to try and find / scrape a data set of my own. Not one that was given or loaded from elsewhere. I decided that I would try to get my hands on some Jiu Jitsu data and turned to **BJJHeroes** to do so.
- as of right now my data is comprised of a
  - name
  - number of wins
  - number of losses
  - number per type of win
  - number per type of loss
- this data is relatively unintersting as it is fairly simple and doesn't allow for much prediction, ie predicting if one fighter would beat another.
- But scraping the data was a foreign task and I will try to get more detailed data in the near future to update and or re write this report.

Data Collection

- to collect the data I decided to try **Go** as it is a language that I am very interested in and want to become more familiar with.
  - I utilized the go colly library and found that it offered a very clean and consise meta for web scraping.
  - I visit a page that lists fighters and every fighter's individual page gets visited for individual data collection
  - if this fighter has wins or losses (some fighters only have descriptions) their unique data gets entered into a CSV engendered by the program. *the go code is at the end of this report*

Possible observations

- The only real observations I can make with the data I collected is to see the distribution of different win & loss types. I can compare the frequencies of each, I can determine what a median and or average competitor (who is well known enough for BJJ heroes) looks like in terms of wins and losses.
- I will rely mainly on histograms as they are utilized to measure frequency which is what I can analyze.
- I will use basic linear models to see if certain win type frequencies can predict more wins overall.

Load Required packages

```
# load libraries
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(ggplot2)
library(vioplot)
```

```
## Loading required package: sm
```

```
## Package 'sm', version 2.2-5.7: type help(sm) for summary information
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

Read The data from the CSV

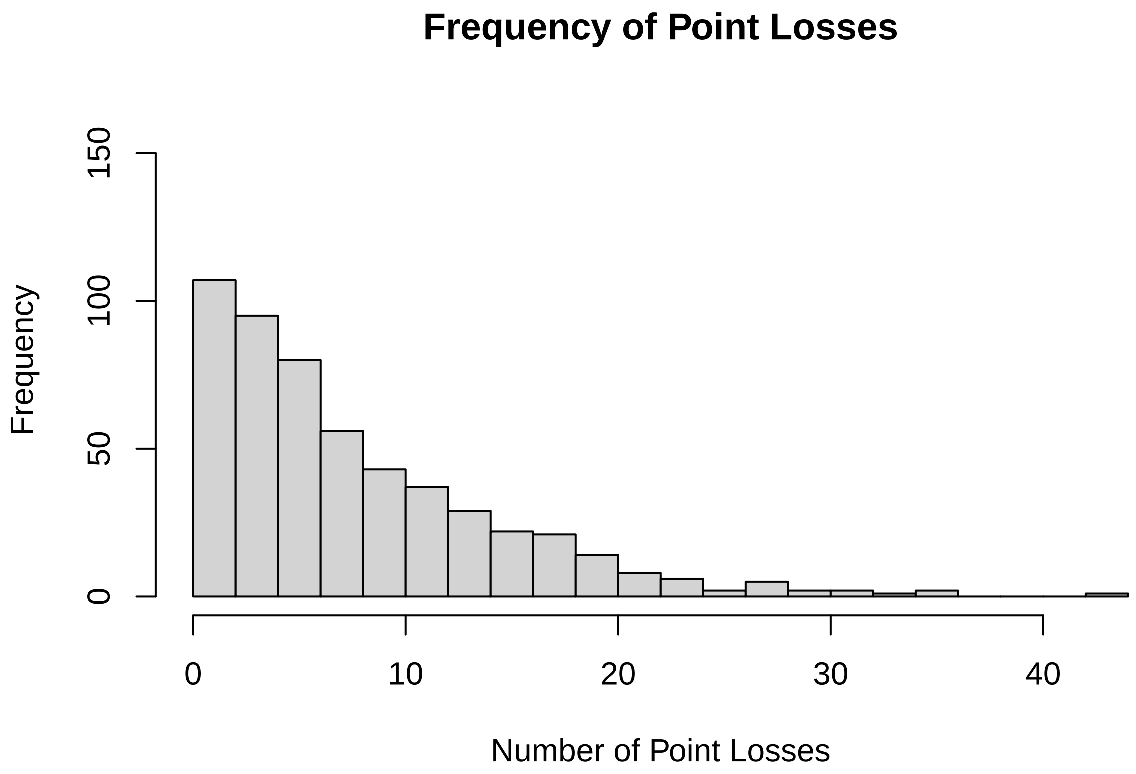
```
goData <- read.csv("./workingFiles/winTypes.csv")
```

Clean data

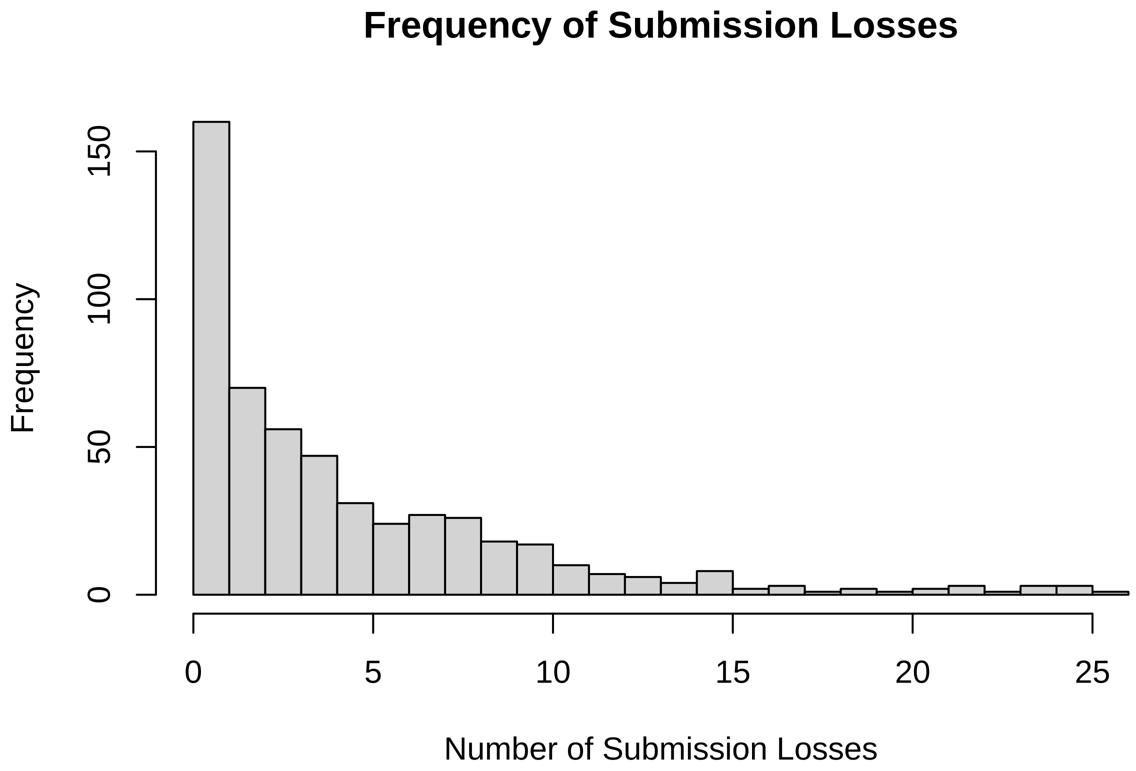
```
goData$wins <- gsub("[A-Z]", "", goData$wins)
goData$losses <- gsub("[A-Z]", "", goData$losses)
nonNumer <- names(goData)[2:length(goData)]
goData[nonNumer] <- lapply(goData[nonNumer], as.numeric)
# add total number of matches
goData <- goData %>%
  mutate(total = losses + wins)
# convert to R df
winLoss_df <- as.data.frame(goData)

# add number of matches won not by submission
winLoss_df <- winLoss_df %>%
  mutate(nonSubWins = wins - Win.Submissions)

# separate  wins
wins_df <- winLoss_df %>% select(-c(colnames(winLoss_df[11:17]),"nonSubWins"))
# calculate submission percentage for wins
wins_df <- wins_df %>%
  mutate(subRate = Win.Submissions / wins) %>% # change between total and wins
  mutate(winRate = wins / total) %>%
  mutate(pWinRate = Win.Points / wins)
wins_df[is.na(wins_df)] <- 0
```

Initial Plots on match data, Histograms

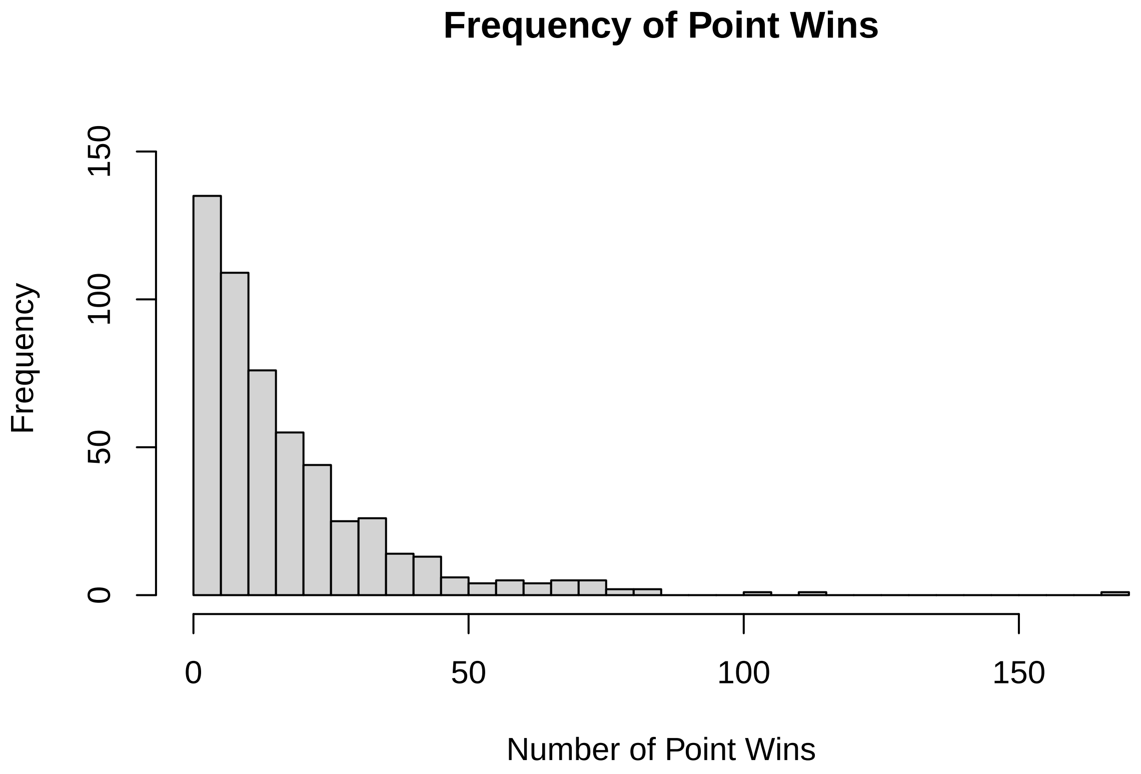
```
# Basic Histograms
hist(winLoss_df$Losses.Points,
     ylim = range(0,160),
     xlab = "Number of Point Losses",
     main = "Frequency of Point Losses",
     breaks = 25)
```



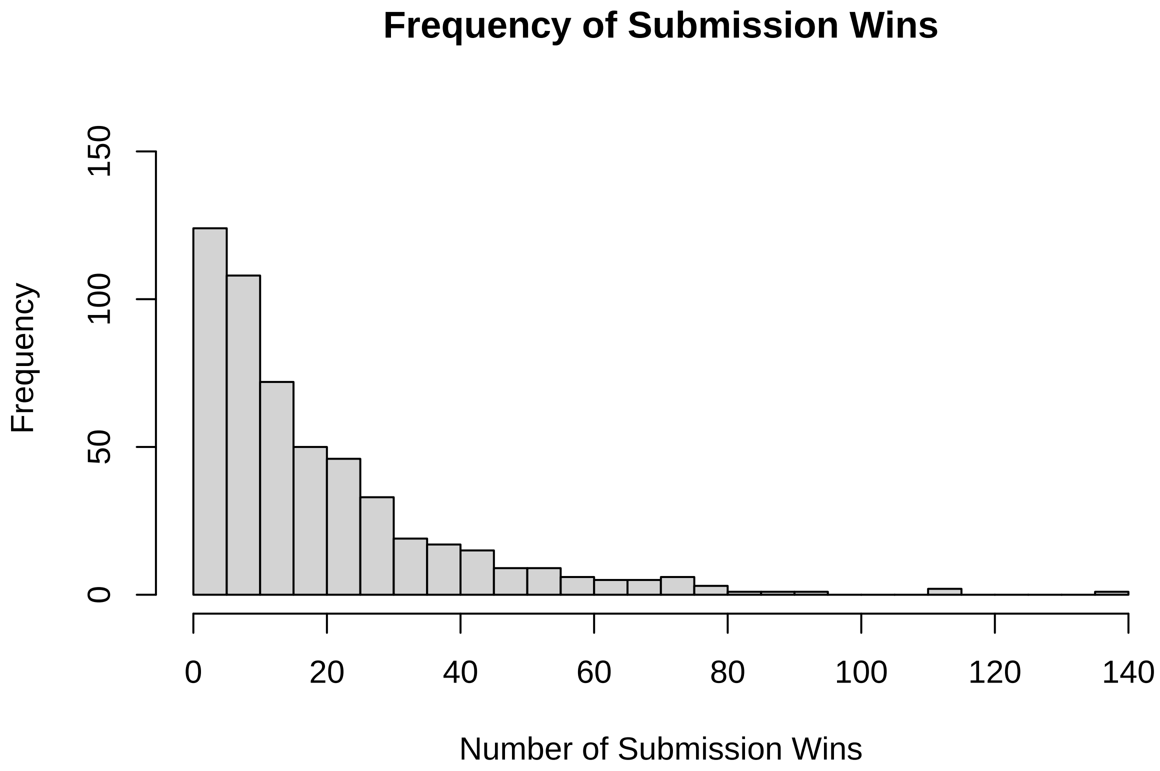
```
hist(winLoss_df$Losses.Submissions,
     ylim = range(0,160),
     xlab = "Number of Submission Losses",
     main = "Frequency of Submission Losses",
     breaks = 25)
```



```
hist(winLoss_df$Win.Points,
     ylim = range(0,160),
     xlab = "Number of Point Wins",
     main = "Frequency of Point Wins",
     breaks = 25)
```



```
hist(winLoss_df$Win.Submissions,
     ylim = range(0,160),
     xlab = "Number of Submission Wins",
     main = "Frequency of Submission Wins",
     breaks = 25)
```



```r
# number of losses for Subs and Points
sum(winLoss_df$Losses.Submissions)
```

```
## [1] 2517
```

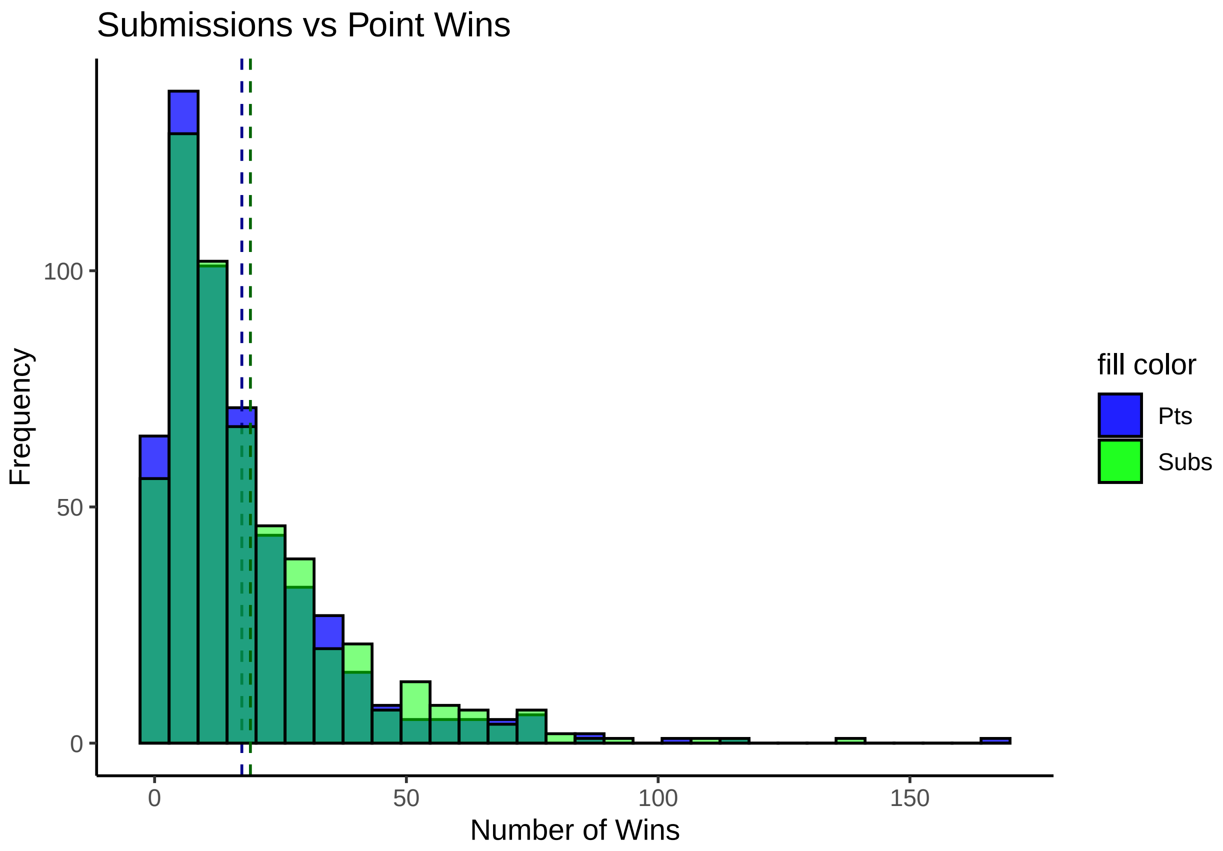
```r
sum(winLoss_df$Losses.Points)
```

```
## [1] 4253
```

Box Plots For wins

```r
# sub wins vs points wins
winBar <- winLoss_df %>%
  ggplot() +
    geom_histogram(aes(x = Win.Points,fill= "blue"),
                   color = "black", alpha = 0.75) +
    geom_vline(aes(xintercept = mean(Win.Points)),
               color = "dark blue", linetype = "dashed") +
    geom_histogram(aes(x = Win.Submissions, fill= "green"),
                   color = "black", alpha = 0.5) +
    geom_vline(aes(xintercept = mean(Win.Submissions)),
               color = "dark green", linetype = "dashed") +
    labs(x = ("Number of Wins"),
         y = ("Frequency"),
         title = "Submissions vs Point Wins") +
    scale_fill_identity(name = "fill color",
                        guide = "legend",
                        labels = c("Pts", "Subs")) +
    theme_classic()
# plot
winBar
```

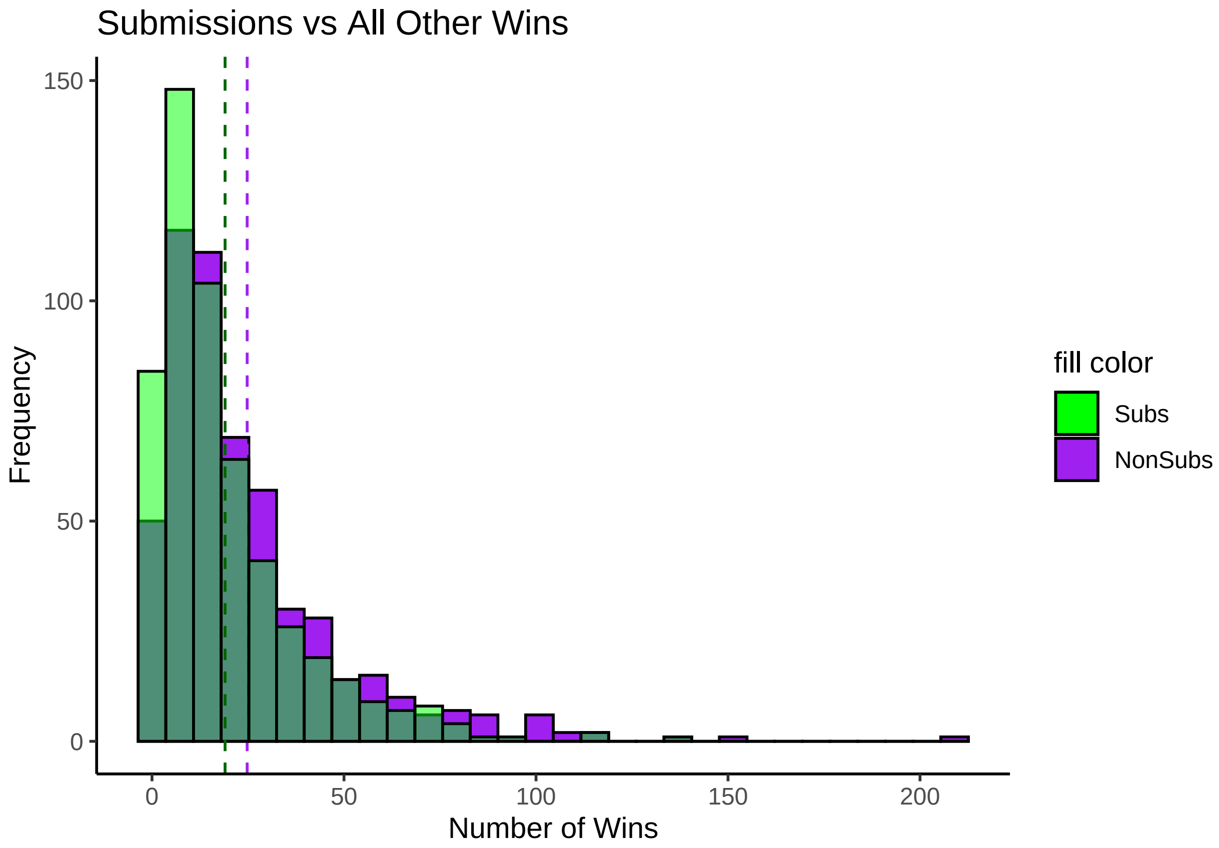
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
# sub wins vs all other types of wins
subVsNon <- winLoss_df %>%
  ggplot() +
    geom_histogram(aes(x = nonSubWins, fill= "purple"),
                   color = "black", alpha = 1) +
    geom_vline(aes(xintercept = mean(nonSubWins)),
               color = "purple", linetype = "dashed") +
    geom_histogram(aes(x = Win.Submissions, fill= "green"),
                   color = "black", alpha = 0.5) +
    geom_vline(aes(xintercept = mean(Win.Submissions)),
               color = "dark green", linetype = "dashed") +
    labs(x = ("Number of Wins"),
         y = ("Frequency"),
         title = "Submissions vs All Other Wins") +
    scale_fill_identity(name = "fill color",
                        guide = "legend",
                        labels = c("Subs", "NonSubs")) +
    theme_classic()
```

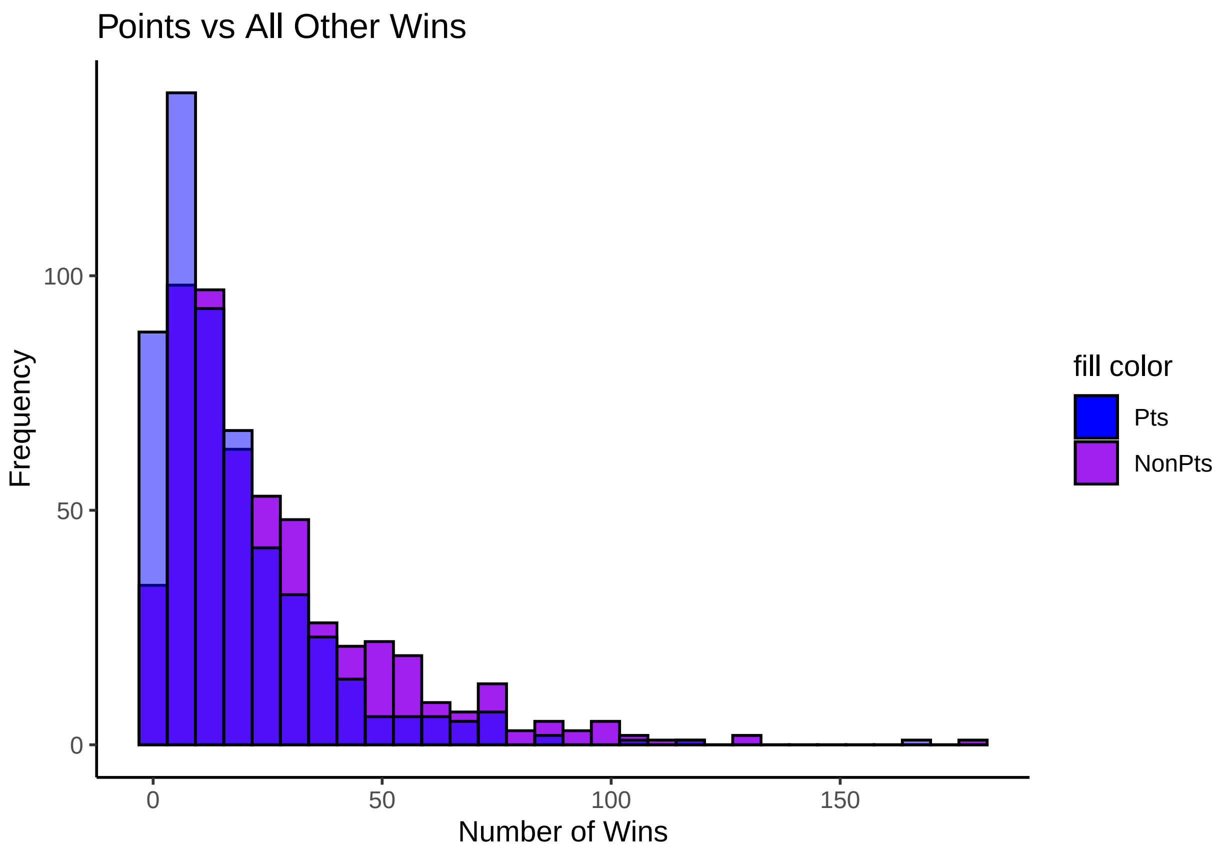
```r
# plot
subVsNon
```

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



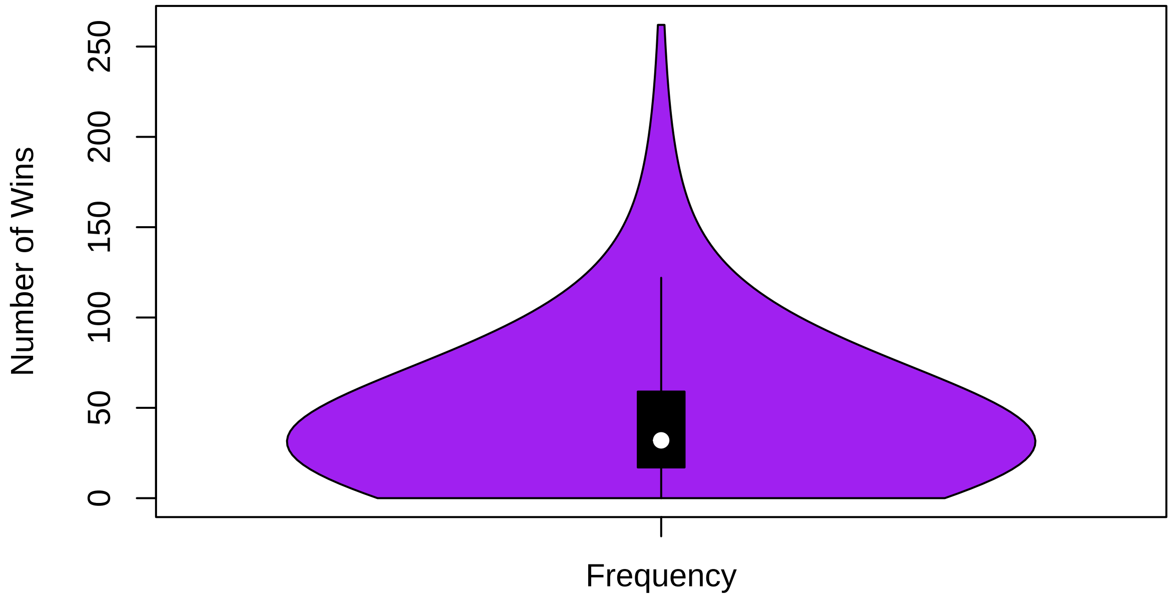
```r
# wins by points vs all other types of wins
PtsVsNon <- winLoss_df %>%
  ggplot() +
    geom_histogram(aes(x = wins - Win.Points, fill= "purple"),
                   color = "black", alpha = 1) +
    geom_histogram(aes(x = Win.Points, fill= "blue"),
                   color = "black", alpha = 0.5) +
    labs(x = ("Number of Wins"), y = ("Frequency"),
         title = "Points vs All Other Wins") +
    scale_fill_identity(name = "fill color",
                        guide = "legend",
                        labels = c("Pts", "NonPts")) +
    theme_classic()

# plot
PtsVsNon
```

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



Violin Plots

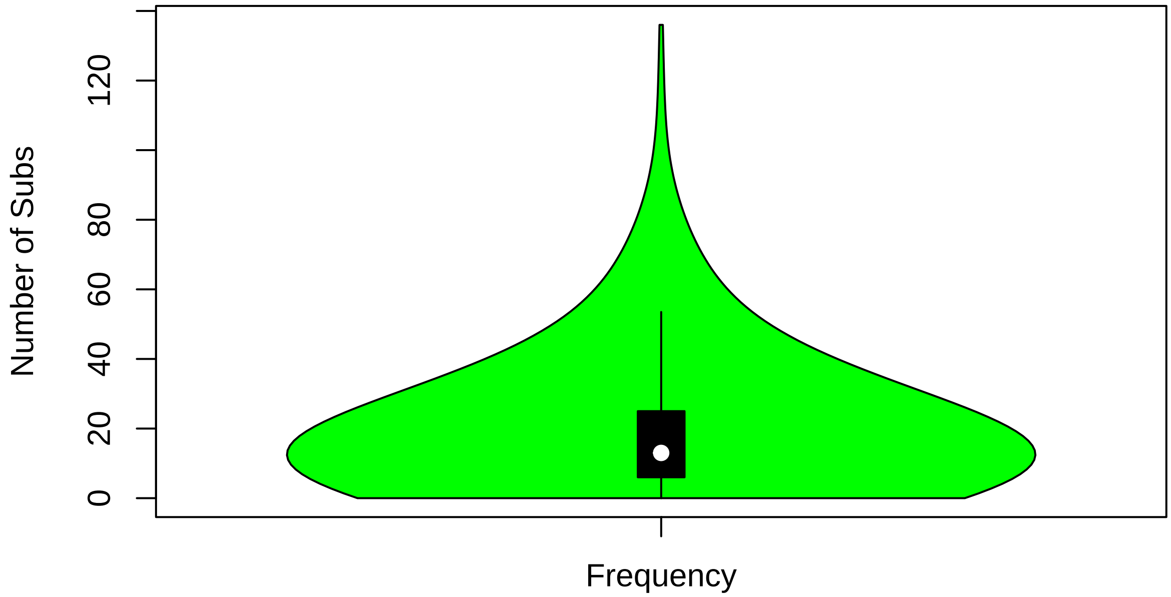
```
winViolin <- vioplot(winLoss_df$wins,
                     col = "purple",
                     names = "Frequency",
                     ylab = "Number of Wins")
```



```
winViolin
```

```
## $upper
## [1] 122
##
## $lower
## [1] 0
##
## $median
## [1] 32
##
## $q1
## [1] 17
##
## $q3
## [1] 59
```

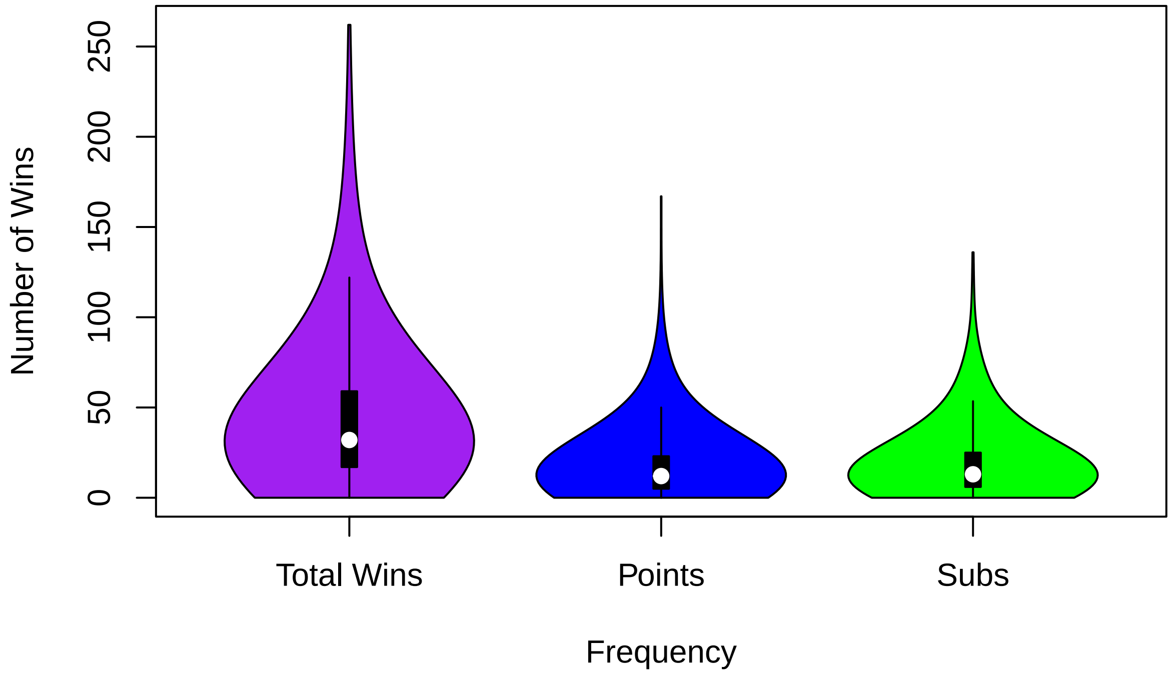
```
subViolin <- vioplot(winLoss_df$Win.Submissions,
                     col = "green",
                     names = "Frequency",
                     ylab = "Number of Subs")
```



```
subViolin
```

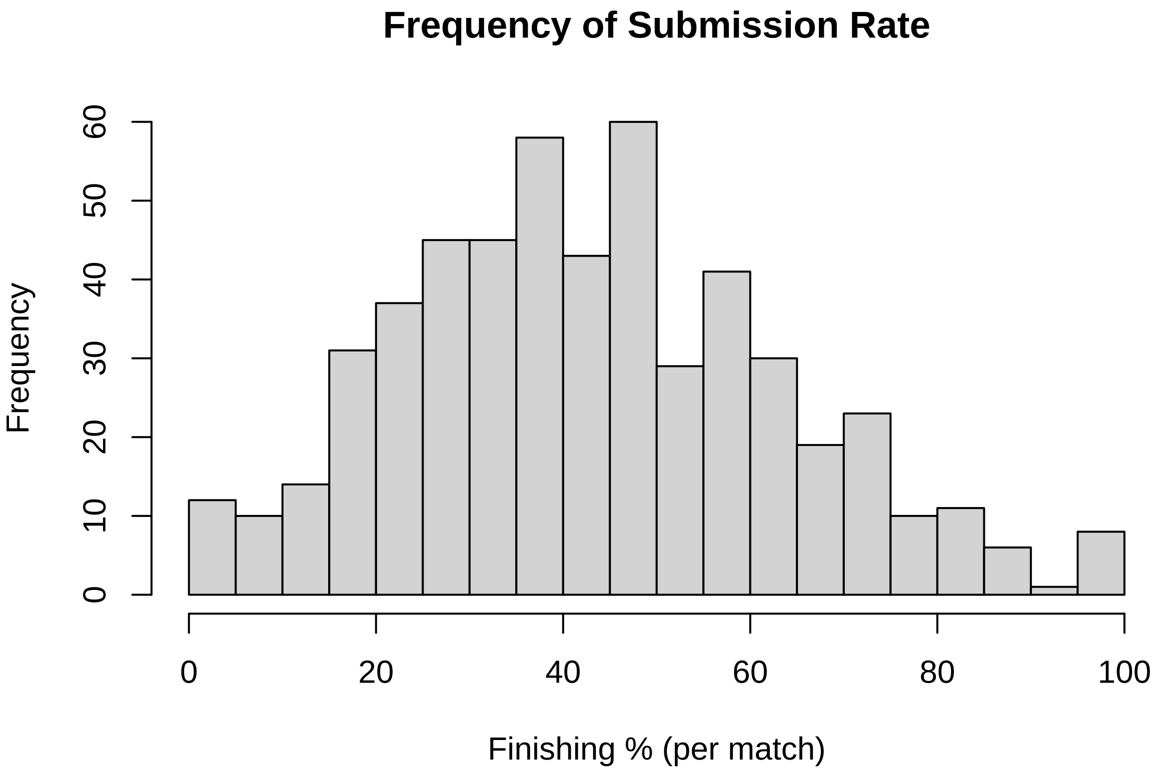
```
## $upper
## [1] 53.5
##
## $lower
## [1] 0
##
## $median
## [1] 13
##
## $q1
## [1] 6
##
## $q3
## [1] 25
```

```
allTogether <- vioplot(
  winLoss_df$wins,
  winLoss_df$Win.Points,
  winLoss_df$Win.Submissions,
  col = c("purple", "blue", "green"),
  names = c("Total Wins", "Points", "Subs"),
  ylab = "Number of Wins",
  xlab = "Frequency"
)
```

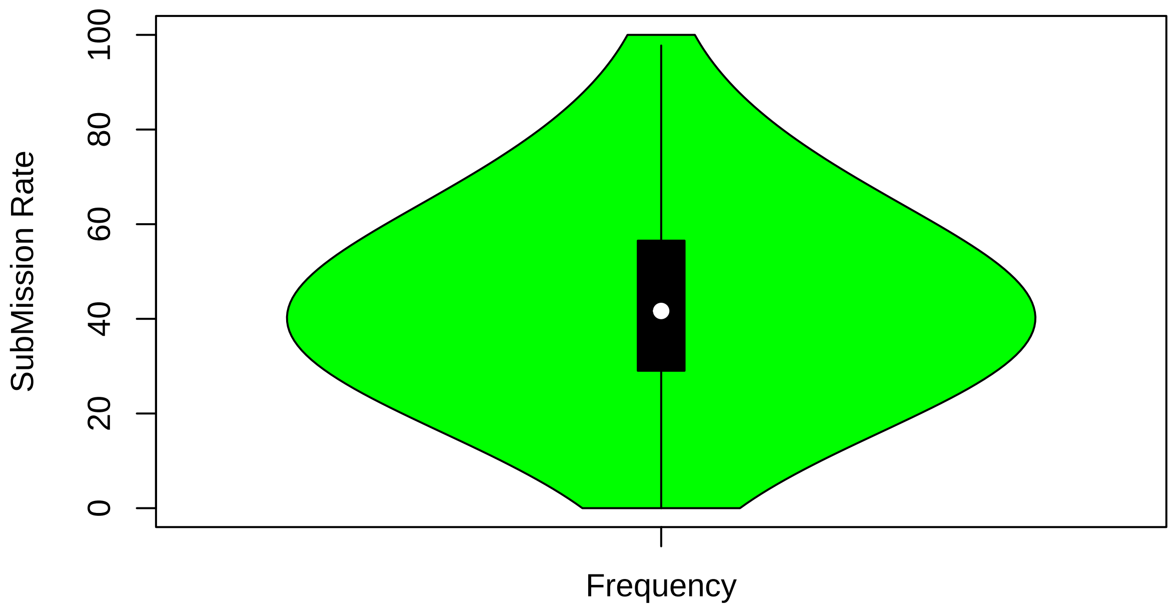


Finishing rate (of Wins) & Win Rate (of total matches)

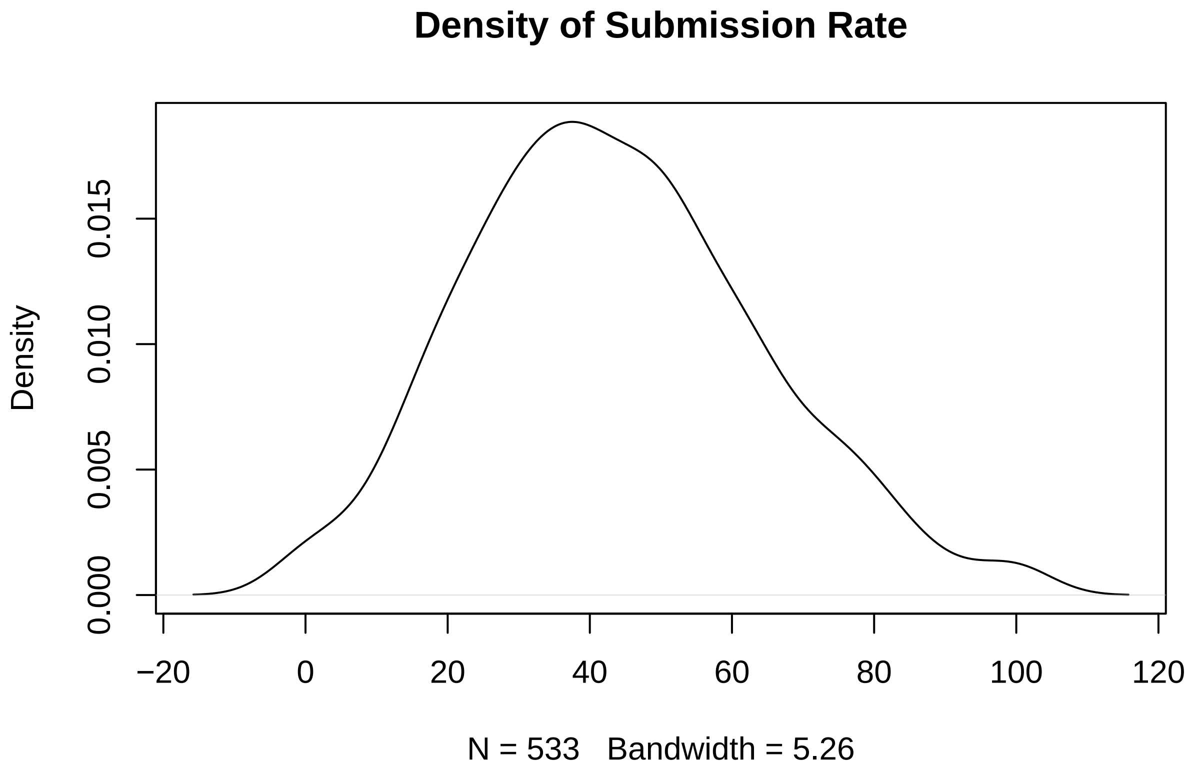
```
# keep in mind this is submissions / total wins
hist(wins_df$subRate * 100,
     xlab = "Finishing % (per match)",
     main = "Frequency of Submission Rate",
     breaks = 25)
```



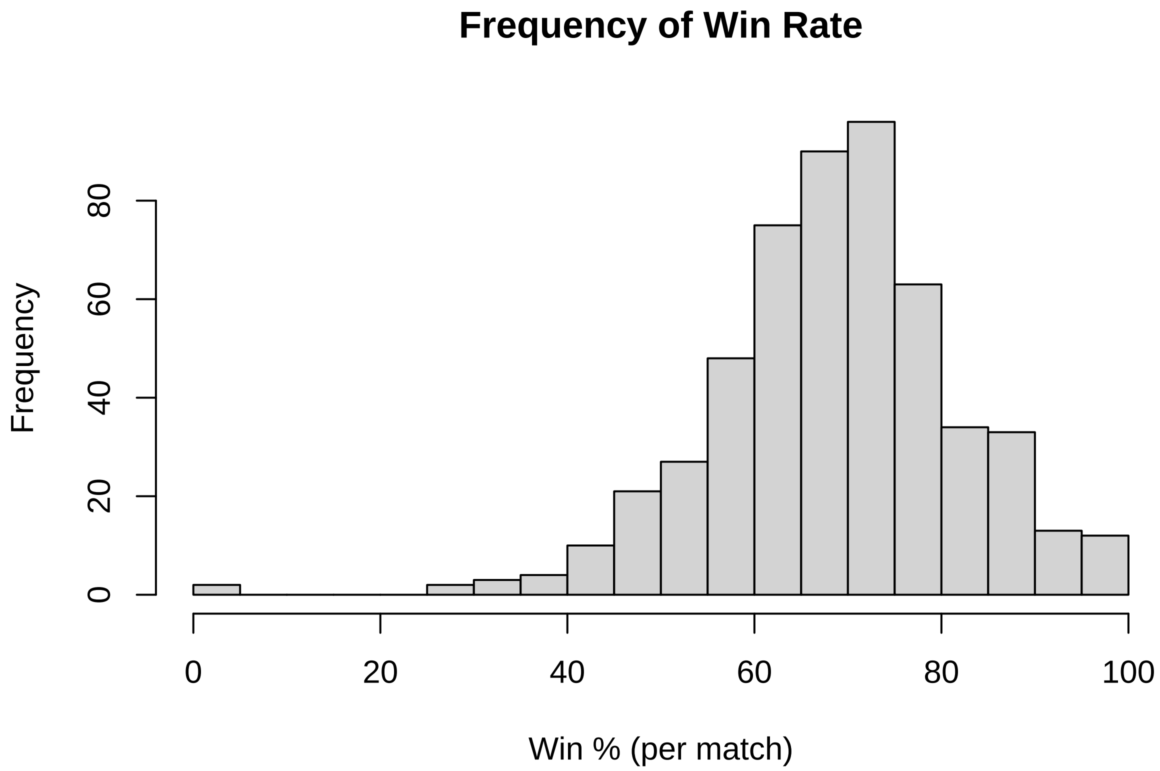
```
# violin plot of subrate
subRateViolin <- vioplot(wins_df$subRate * 100,
                         col = "green",
                         names = "Frequency",
                         ylab = "SubMission Rate")
```



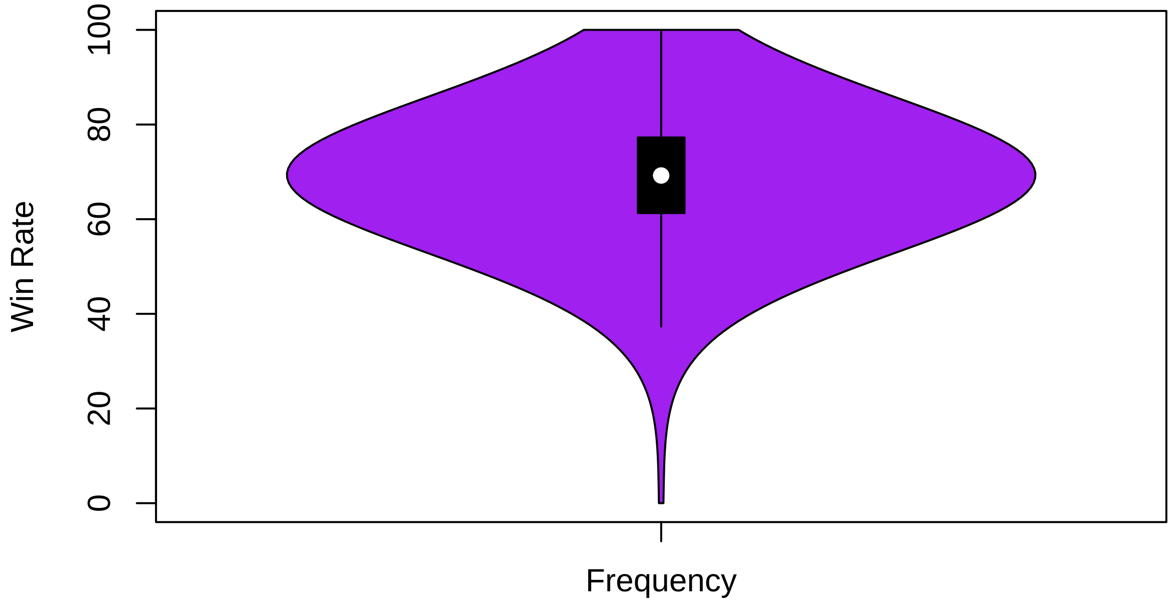
```
# plot the density of the submission rate
plot(density(wins_df$subRate * 100), main = "Density of Submission Rate")
```



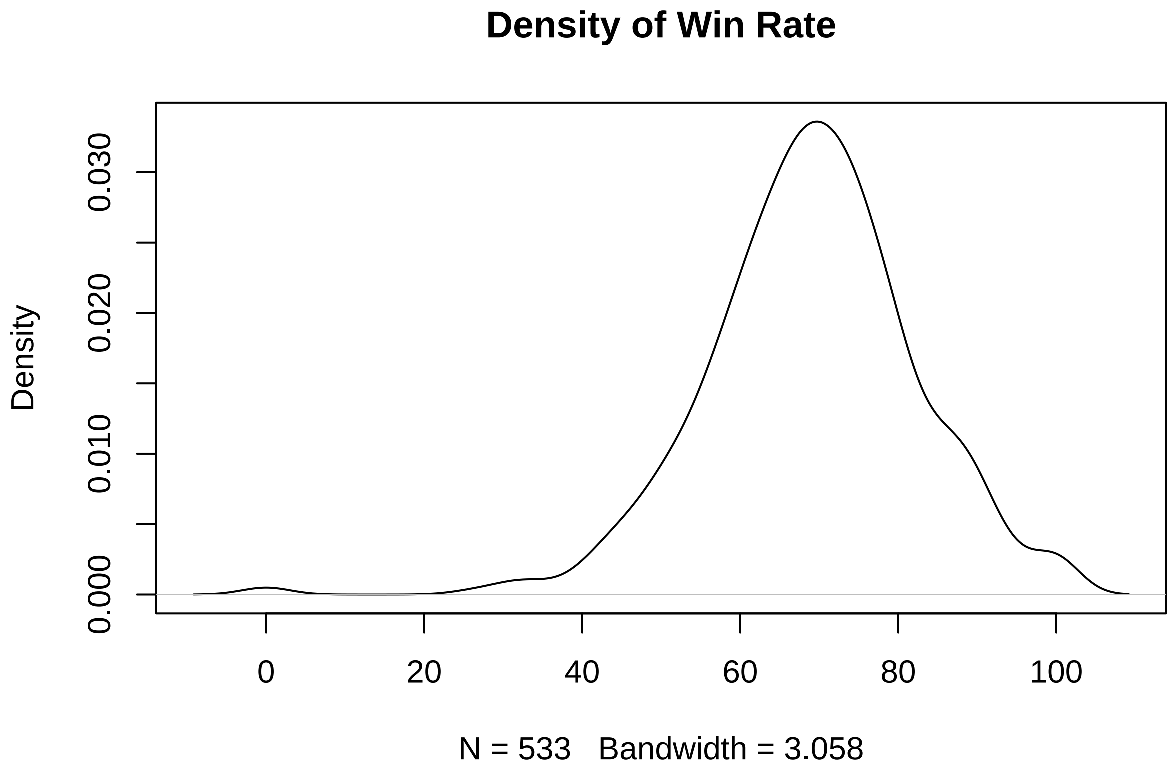
```r
# Win Rate
hist(wins_df$winRate * 100,
     xlab = "Win % (per match)",
     main = "Frequency of Win Rate",
     breaks = 25)
```



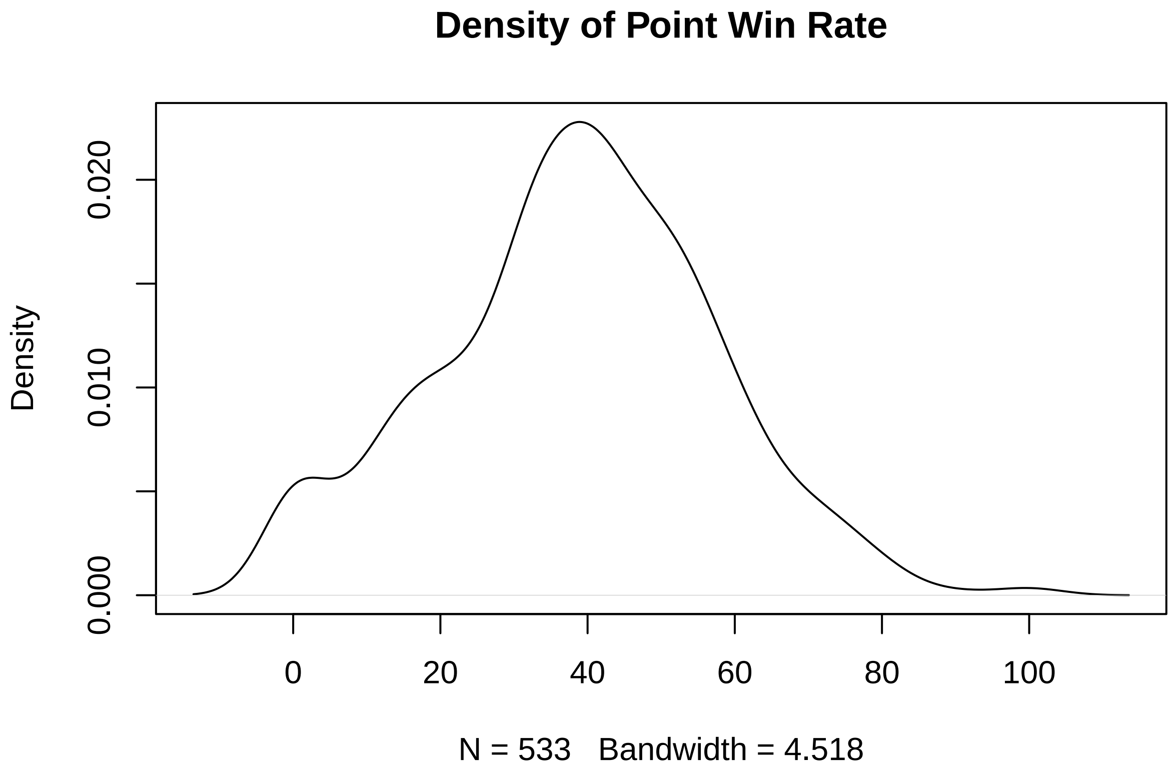
```
# violin plot of winRate
subRateViolin <- vioplot(wins_df$winRate * 100,
                         col = "purple",
                         names = "Frequency",
                         ylab = "Win Rate")
```



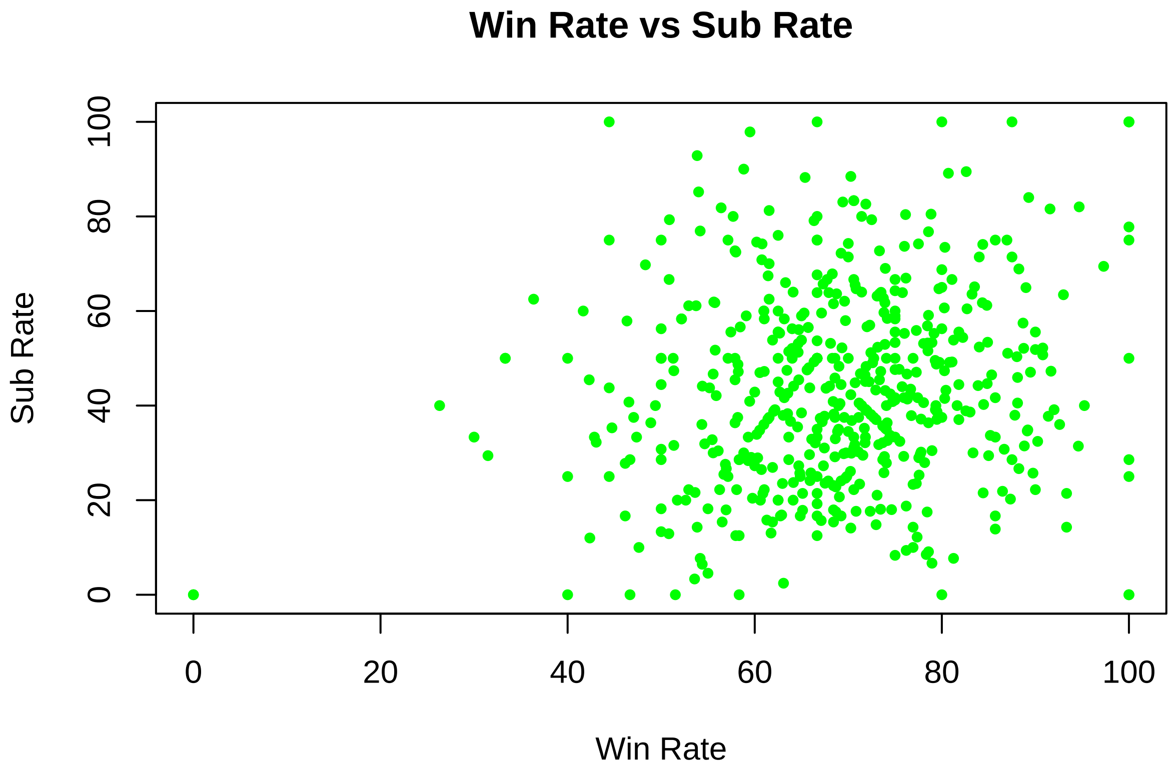
```
# plot the density of the Win rate
plot(density(wins_df$winRate * 100), main = "Density of Win Rate")
```



```
# quick look at point win rate

plot(density(wins_df$pWinRate * 100), main = "Density of Point Win Rate")
```



```
# comparing win rate and submission rate on a graph with a quick graph
plot(wins_df$winRate * 100, wins_df$subRate * 100,
     xlab = "Win Rate",
     ylab = "Sub Rate",
     main = "Win Rate vs Sub Rate",
     col = "green",
     pch = 16,
     cex = 0.75)
```

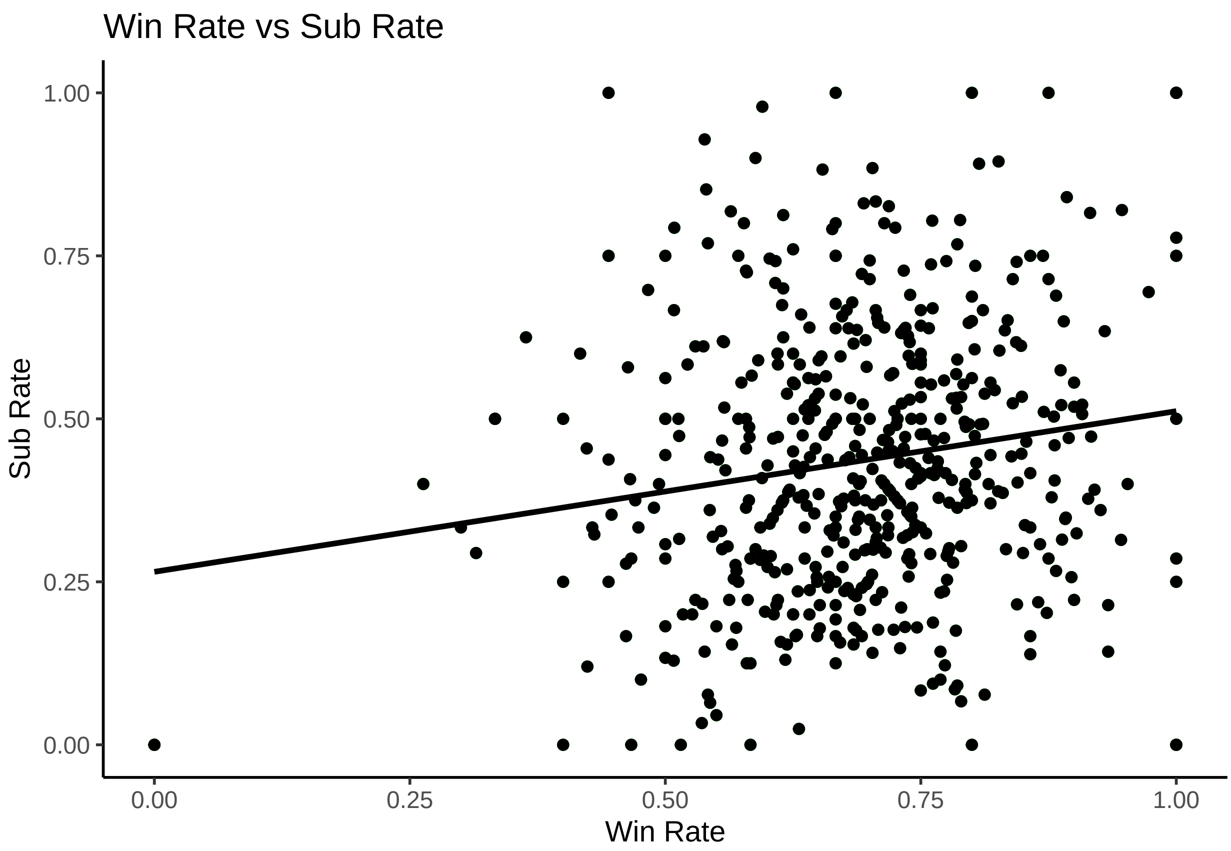


basic modeling to predict Win Rate

Subs Rate (of wins) vs Win Rate (of total matches)

```
winVsSub <- wins_df %>%
  ggplot(aes(y= subRate, x = winRate)) +
  geom_point(color = "green") +
  labs(title = "Win Rate vs Sub Rate", y = "Sub Rate", x = "Win Rate") +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE , color = "black") +
  theme(legend.position="none") +
  theme_classic()

winVsSub
```

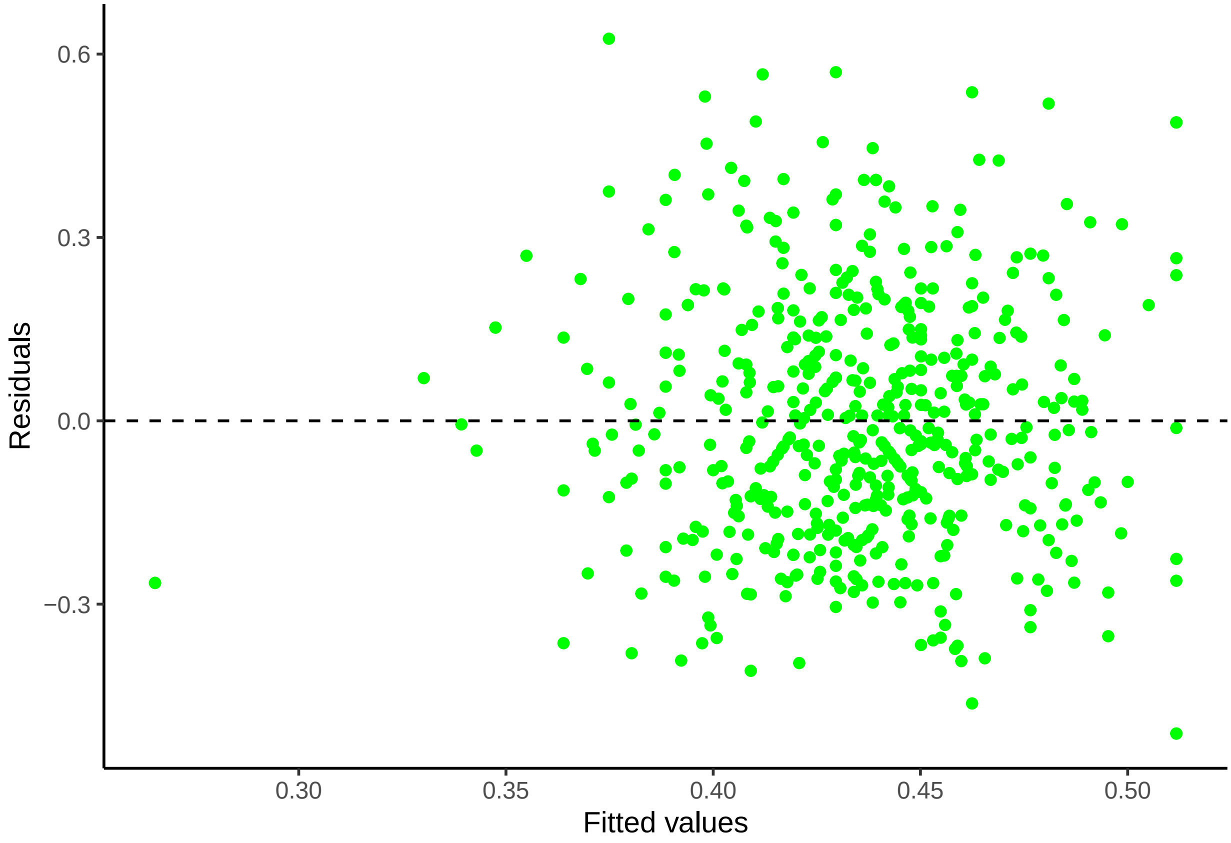
```
## `geom_smooth()` using formula 'y ~ x'
```



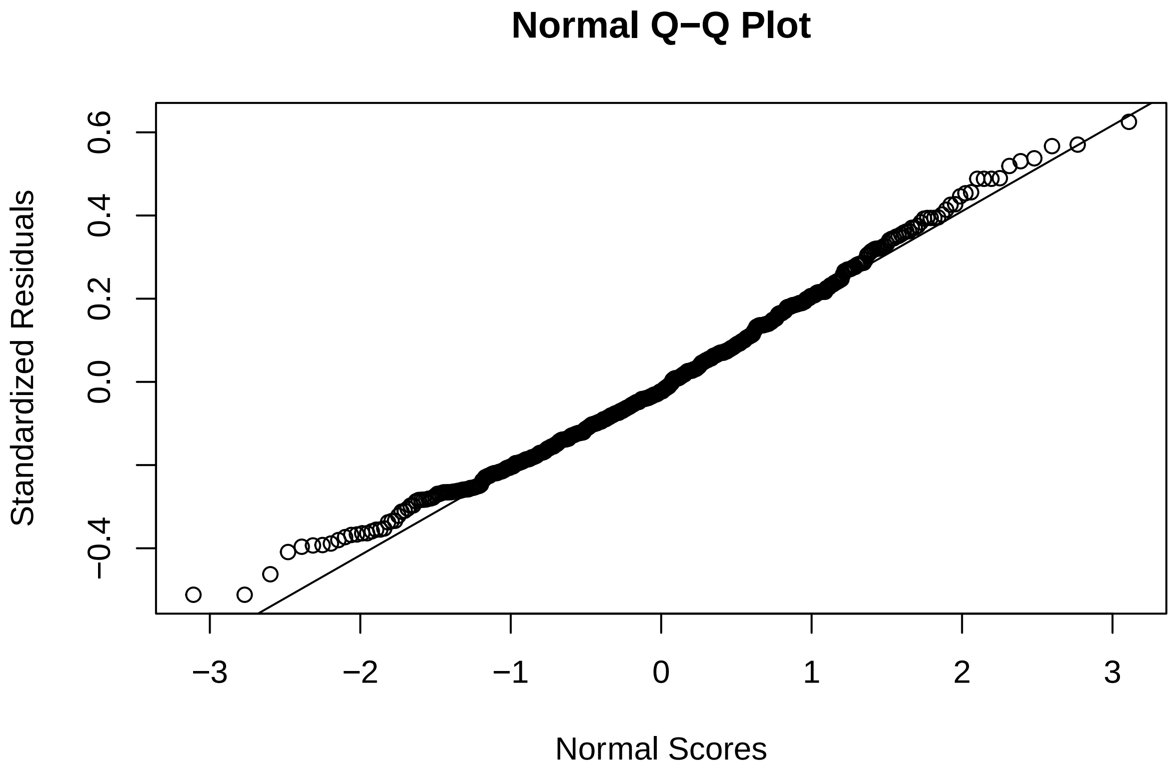
```
# turn the line from the above graph into a model
simpleModel <- lm(wins_df$subRate ~ wins_df$winRate, wins_df)
# show the summary stats of the model
summary(simpleModel)
```

```
##
## Call:
## lm(formula = wins_df$subRate ~ wins_df$winRate, data = wins_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.51173 -0.14262 -0.02262  0.13622  0.62516
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.26533    0.04624   5.738 1.61e-08 ***
## wins_df$winRate  0.24641    0.06585   3.742 0.000202 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2034 on 531 degrees of freedom
## Multiple R-squared:  0.02569,    Adjusted R-squared:  0.02386
## F-statistic:    14 on 1 and 531 DF,  p-value: 0.0002024
```

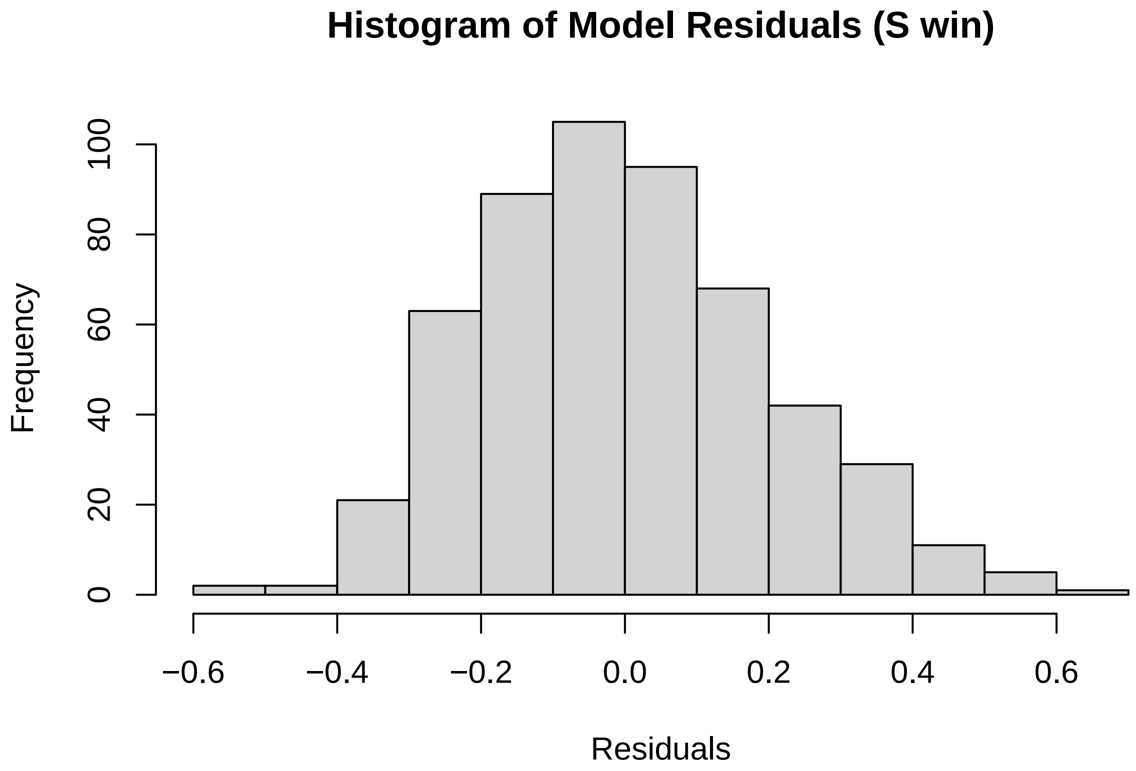
```
# plot the residuals of the model (want to see symmetry)
residualsPlot <- simpleModel %>%
  ggplot(aes(x = .fitted, y = .resid)) +
  geom_point(color = "green") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  xlab("Fitted values") +
  ylab("Residuals") +
  theme_classic()
# show the plot
residualsPlot
```



```
# graphing the residuals
qqnorm(simpleModel$residuals,
  ylab="Standardized Residuals",
  xlab="Normal Scores")
qqline(simpleModel$residuals)
```



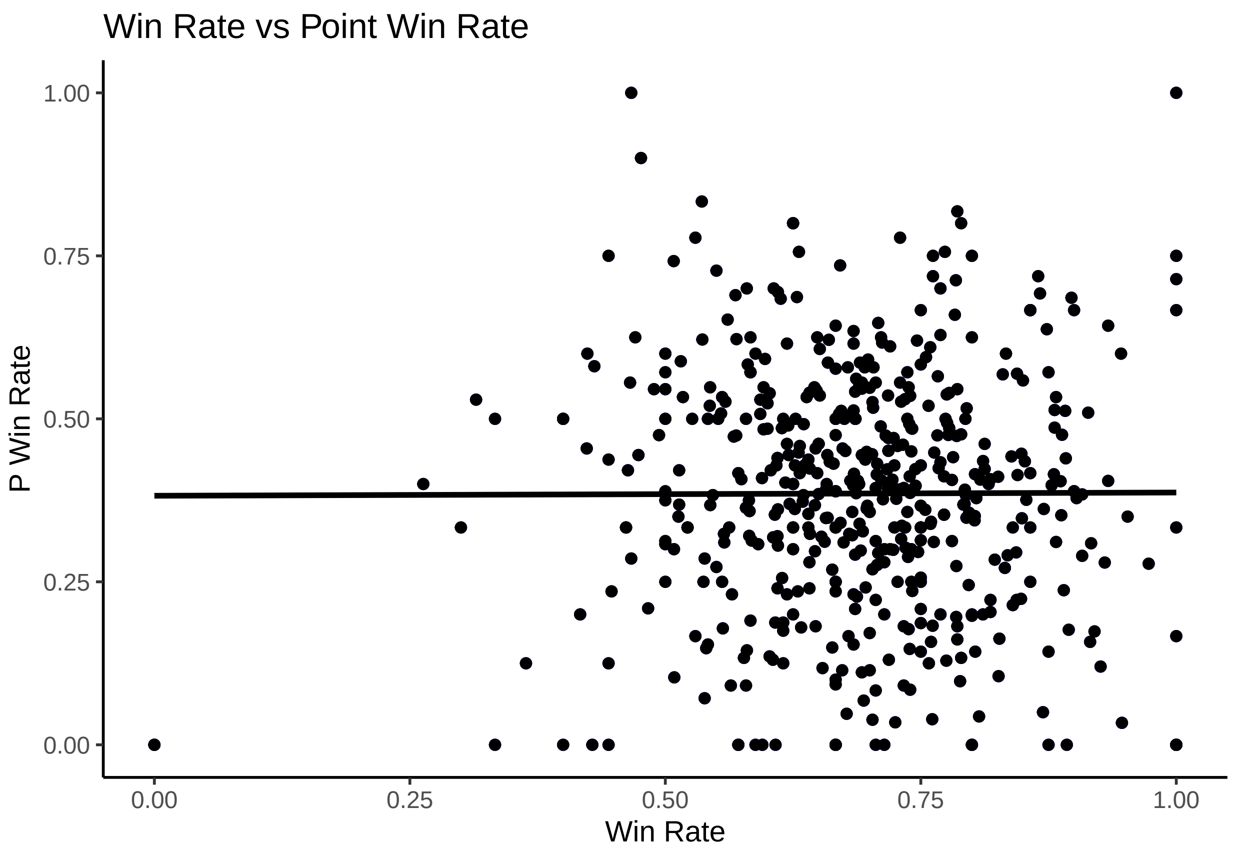
```
# check the distribution of the residuals (want normality)
hist(simpleModel$residuals, main = "Histogram of Model Residuals (S win)", xlab = "Residuals")
```



Point Rate (of wins) vs Win Rate (of total matches)

```
winVsSub <- wins_df %>%
  ggplot(aes(y= pWinRate, x = winRate)) +
  geom_point(color = "blue") +
  labs(title = "Win Rate vs Point Win Rate", y = "P Win Rate", x = "Win Rate") +
  geom_point() +
  stat_smooth(method = "lm", se = FALSE , color = "black") +
  theme(legend.position="none") +
  theme_classic()

winVsSub
```

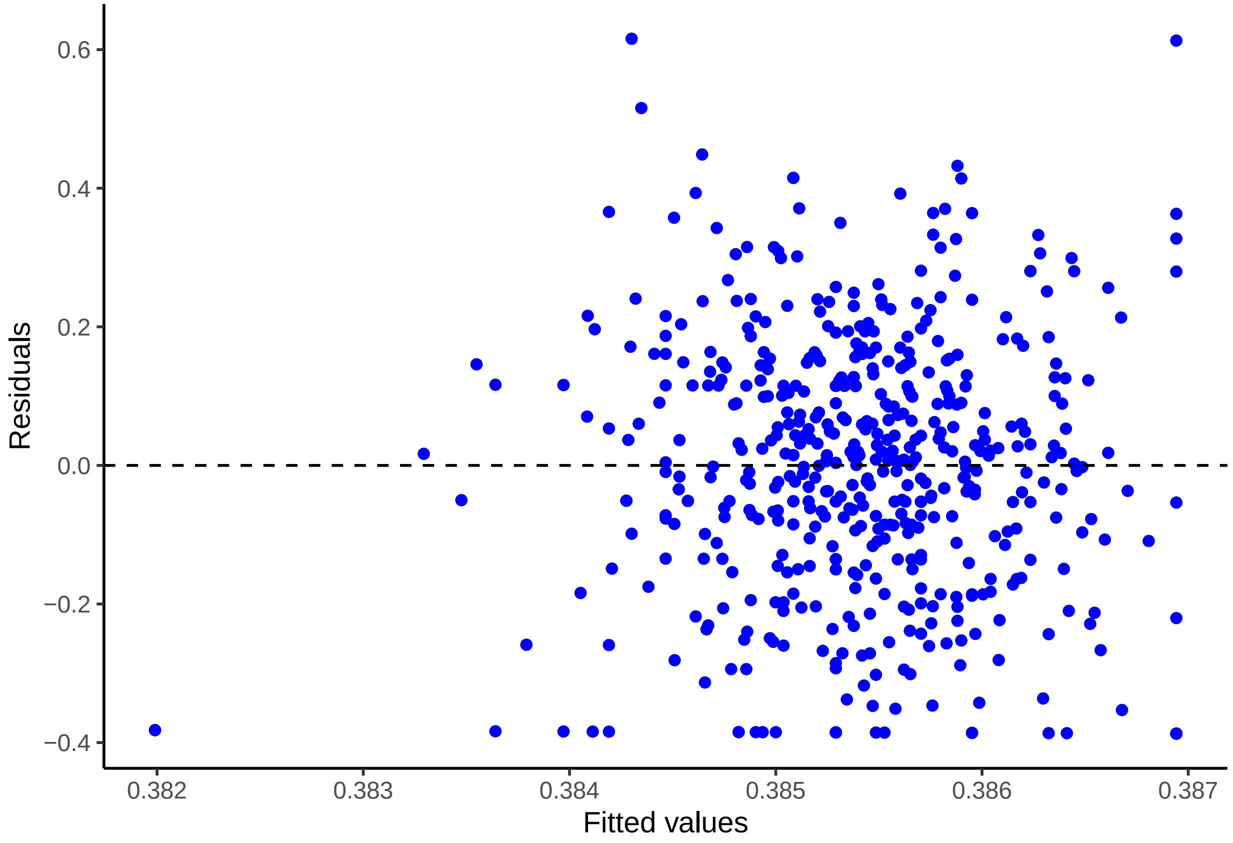
```
## `geom_smooth()` using formula 'y ~ x'
```



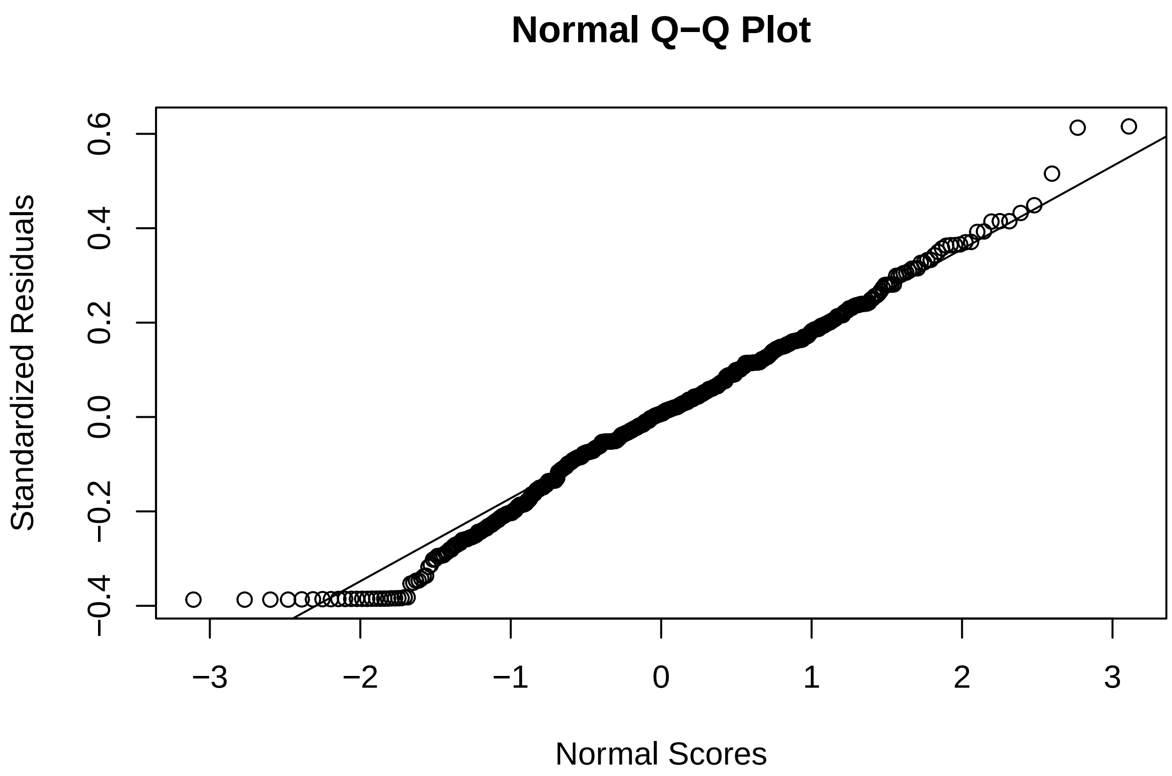
```
# turn the line from the above graph into a model
simpleModel2 <- lm(wins_df$pWinRate ~ wins_df$winRate, wins_df)
# show the summary stats of the model
summary(simpleModel2)
```

```
##
## Call:
## lm(formula = wins_df$pWinRate ~ wins_df$winRate, data = wins_df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.38694 -0.11479  0.00728  0.12254  0.61570
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.381990   0.042816   8.922   <2e-16 ***
## wins_df$winRate 0.004952   0.060969   0.081    0.935
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1883 on 531 degrees of freedom
## Multiple R-squared:  1.242e-05,  Adjusted R-squared:  -0.001871
## F-statistic: 0.006596 on 1 and 531 DF,  p-value: 0.9353
```

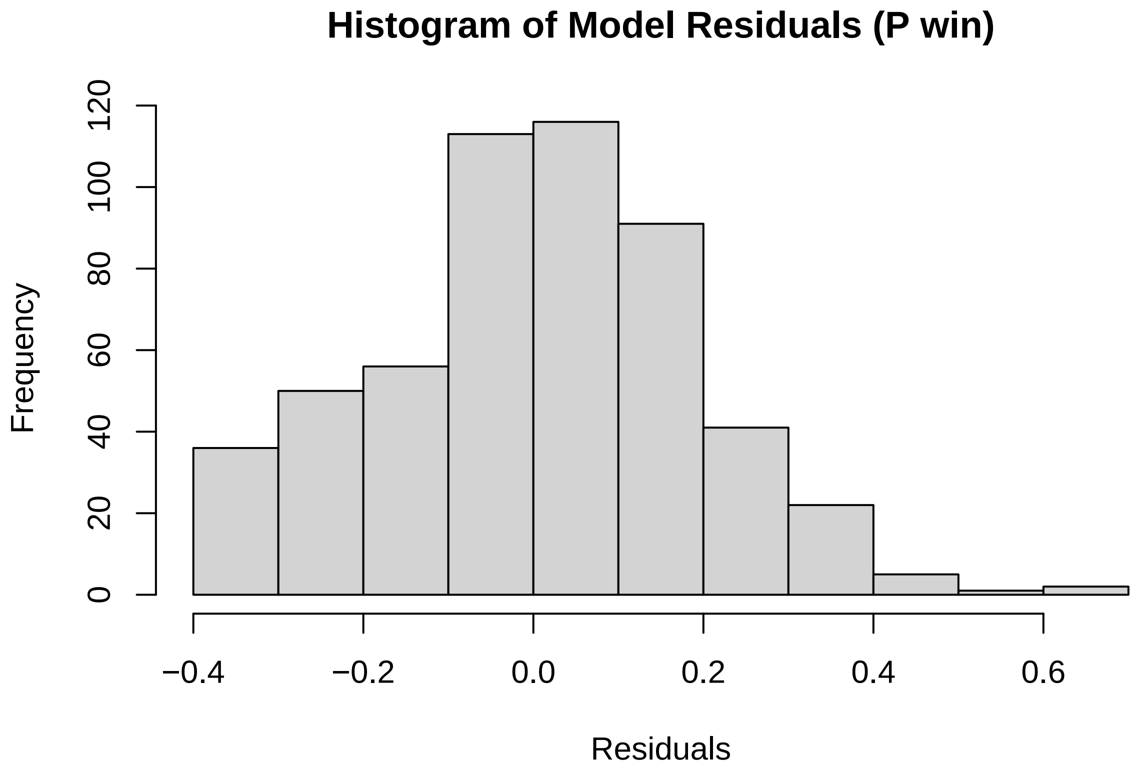
```
# plot the residuals of the model (want to see symmetry)
residualsPlot2 <- simpleModel2 %>%
  ggplot(aes(x = .fitted, y = .resid)) +
  geom_point(color = "blue") +
  geom_hline(yintercept = 0, linetype = "dashed", color = "black") +
  xlab("Fitted values") +
  ylab("Residuals") +
  theme_classic()
# show the plot
residualsPlot2
```



```
# graphing the residuals
qqnorm(simpleModel2$residuals,
  ylab="Standardized Residuals",
  xlab="Normal Scores")
qqline(simpleModel2$residuals)
```



```
# check the distribution of the residuals (want normality)
hist(simpleModel2$residuals, main = "Histogram of Model Residuals (P win)", xlab = "Residuals")
```



Conclusion

- the Medians for submissions and total wins are 13, and 32 respectively. This means that to be in the upper 50% of competitors you need to have 32 wins or more and to be in the upper range of submission artists you need 13 or more submissions.
- initially when running the models I had submission rate and point win rate of all the matches, but this was almost a tautology. Of course someone more likely to win a match by submission or points than not win will have a high win rate overall. So I changed the models to look at sub / point win rate of total wins. This way someone could have an extremely high submission rate but a very low win rate with less variable dependence.
- Based on the models R^2 values we can see a very slight **positive correlation** between the likelihood that a match won by a competitor was won by a submission and the likelihood that that same competitor would win a match via any method. Inversely we can see a very slight **negative correlation** between a competitors likelihood to have won a match by points and their overall likelihood to win a match.
- because the p value of the model looking at submission rate is 0.0002024 we can say that winning more matches by submission than by not submission does signify a greater chance to win a match.
- however, because the p value of the model looking at is above 0.05 we cannot say that having a high likelihood of wins coming from points makes you more likely to lose a match.
- in summary this shows that **more submissions == better Jiu Jitsu Competitor!!!** (but we already knew that)

Go code for scraping

```
package main
```

```
import (
    "encoding/csv"
    "fmt"
    "log"
    "os"

    "github.com/gocolly/colly"
)

// main function
func main() {
    createList()
    visitAll()
    indiv("https://www.bjjheroes.com/?p=7556")
}

// visitAll function
func visitAll() {
    /* instatiate colly */
    c := colly.NewCollector(
        colly.AllowedDomains(
            "www.bjjheroes.com/",
            "bjjheroes.com/",
            "https://bjjheroes.com/",
            "www.bjjheroes.com",
            "bjjheroes.com",
            "https://bjjheroes.com",
        ),
    )
    // if a link is found in a table
    c.OnHTML("td.column-1 a[href]", func(e *colly.HTMLElement) {
        url := e.Request.AbsoluteURL(e.Attr("href")) // get the link as an absolute
        indiv(url)                                    // run indiv on the absolute link
    })
    // start scraping
    c.Visit("https://www.bjjheroes.com/a-z-bjj-fighters-list")
}

// createList function
func createList() {
    // instantiate csv
    file, err := os.OpenFile("list.csv", os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil { // deal with errors
        log.Fatalf("cannot create file due to %s", err)
        return
    }
    defer file.Close()
    writer := csv.NewWriter(file) // create a writer to modify csv
    // write column names to csv
    writer.Write([]string{
        "Name",
        "wins",
        "losses",
        "Win Points",
        "Win Advantages",
        "Win Submissions",
        "Win Decision",
        "Win Penalties",
        "Win EBI / OT",
        "Win DQ",
        "Losses Points",
        "Losses Advantages",
        "Losses Submissions",
        "Losses Decision",
        "Losses Penalties",
        "Losses EBI / OT",
        "Losses DQ",
    })
    writer.Flush() // ensure data is written
}

// indiv function
func indiv(url string) {
    // instantiate or open csv
    file, err := os.OpenFile("list.csv", os.O_APPEND|os.O_CREATE|os.O_WRONLY, 0644)
    if err != nil {
        log.Fatalf("cannot create file due to %s", err)
        return
    }
    defer file.Close()
    writer := csv.NewWriter(file) // create a writer to modify csv
    defer writer.Flush()          // defer pushes to end of function
    /* instatiate colly */
    c := colly.NewCollector(
        colly.AllowedDomains(
            "www.bjjheroes.com/",
            "bjjheroes.com/",
            "https://bjjheroes.com/",
            "www.bjjheroes.com",
            "bjjheroes.com",
            "https://bjjheroes.com",
        ),
    )
    // create empty column variables
    // using standard vs short declaration as short is a bit messy in this case
    fName := "0"
    wins := "0"
    lose := "0"
    wPoints := "0"
    wAdvantage := "0"
    wSubs := "0"
    wDec := "0"
    wPen := "0"
    wEBI := "0"
    wDQ := "0"
    lPoints := "0"
```

```
  lAdvantage := "0"
  lSubs := "0"
  lDec := "0"
  lPen := "0"
  lEBI := "0"
  lDQ := "0"

// find name
  c.OnHTML("h1", func(a *colly.HTMLElement) {
      fName = a.Text
  })
//find wins
  c.OnHTML("div.Win_title", func(b *colly.HTMLElement) {
      wins = b.ChildText("em")
  })
// find win types
  c.OnHTML("div.wrapper_canvas li", func(d *colly.HTMLElement) {
      switch d.ChildText("span.by_points") {
      case "BY POINTS":
          wPoints = d.ChildText("span.per_no")
      case "BY ADVANTAGES":
          wAdvantage = d.ChildText("span.per_no")
      case "BY SUBMISSION":
          wSubs = d.ChildText("span.per_no")
      case "BY DECISION":
          wDec = d.ChildText("span.per_no")
      case "BY PENALTIES":
          wPen = d.ChildText("span.per_no")
      case "BY EBI/OT":
          wEBI = d.ChildText("span.per_no")
      case "BY DQ":
          wDQ = d.ChildText("span.per_no")
      }
  })
// find losses
c.OnHTML("div.Win_title_lose", func(t *colly.HTMLElement) {
  lose = t.ChildText("em")
})
// find loss types
  c.OnHTML("div.wrapper_canvas_lose li", func(d *colly.HTMLElement) {
      switch d.ChildText("span.by_points") {
      case "BY POINTS":
          lPoints = d.ChildText("span.per_no_lose")
      case "BY ADVANTAGES":
          lAdvantage = d.ChildText("span.per_no_lose")
      case "BY SUBMISSION":
          lSubs = d.ChildText("span.per_no_lose")
      case "BY DECISION":
          lDec = d.ChildText("span.per_no_lose")
      case "BY PENALTIES":
          lPen = d.ChildText("span.per_no_lose")
      case "BY EBI/OT":
          lEBI = d.ChildText("span.per_no_lose")
```

```
        case "BY DQ":
            lDQ = d.ChildText("span.per_no_lose")
        }
    })

  // start the scraper
    c.Visit(url)

  // write to csv
    if wins != "0" || lose != "0" {
        writer.Write([]string{
            fName,
            wins,
            lose,
            wPoints,
            wAdvantage,
            wSubs,
            wDec,
            wPen,
            wEBI,
            wDQ,
            lPoints,
            lAdvantage,
            lSubs,
            lDec,
            lPen,
            lEBI,
            lDQ,
        })
    }
  // for every individual fighter print done
    fmt.Printf("!!!Done!!!")
}
```