MAP501 Coursework 2023

Last updated: November 30, 2023

Contents


```
library("conflicted")
library("tidyverse")
library("magrittr")
library("here")
library("janitor")
library("lubridate")
library("gridExtra")
library("readxl")
```

```
library("glmnet")
library("Lahman")
library("AER")
library("viridis")
library("lindia")
library("lme4")
library("caret")
library("pROC")
```

1. Simple Linear Regression

a. Creating a 'Mean Salary' Data Frame

Using a groupby performed on the 'Salaries' data frame, the mean salary across all the different teams for each year can be calculated:

```
df_MeanSalaries <- Lahman::Salaries %>%
  dplyr::filter(yearID %in% 1990:2010) %>%
  group_by(teamID, yearID) %>%
  summarise(meanSalary = mean(salary))

df_MeanSalaries
```

```
# A tibble: 608 x 3
# Groups:   teamID [34]
   teamID yearID meanSalary
   <fct>   <int>      <dbl>
 1 ANA      1997   1004370.
 2 ANA      1998   1214147.
 3 ANA      1999   1384704.
 4 ANA      2000   1715472.
 5 ANA      2001   1584506.
 6 ANA      2002   2204345.
 7 ANA      2003   2927099.
 8 ANA      2004   3723506.
 9 ARI      1998    898528.
10 ARI      1999   2020706.
# i 598 more rows
```

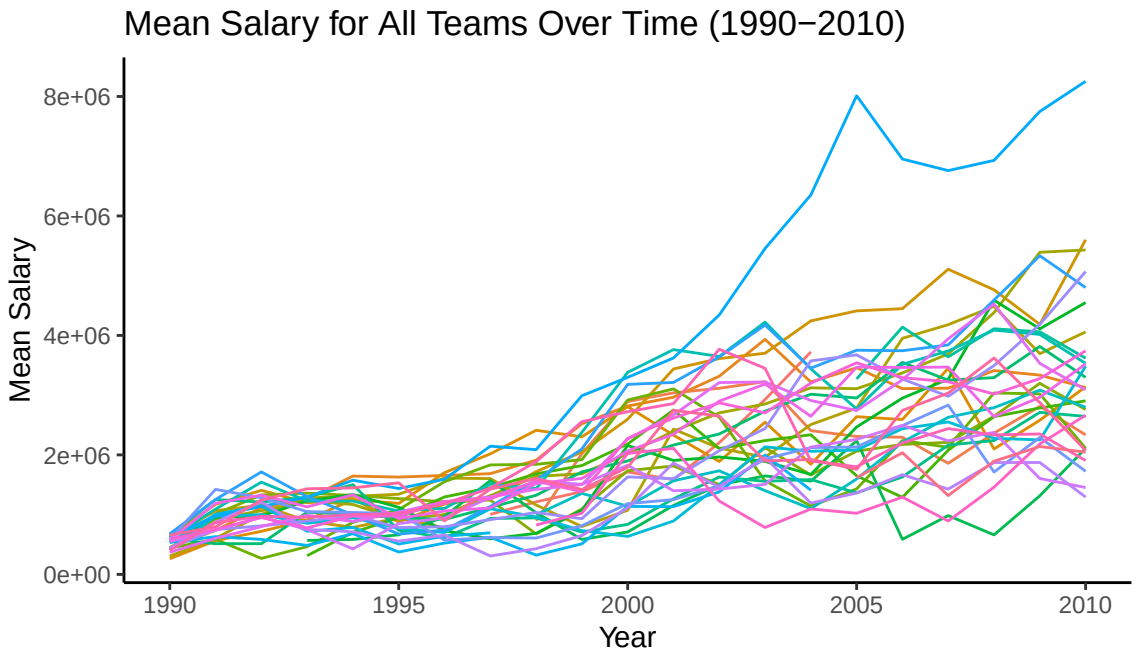
b. Plotting Mean Salary Over Time

Plotting how the mean salary for all the teams has changed over time will demonstrate the relationship between time and mean salary:

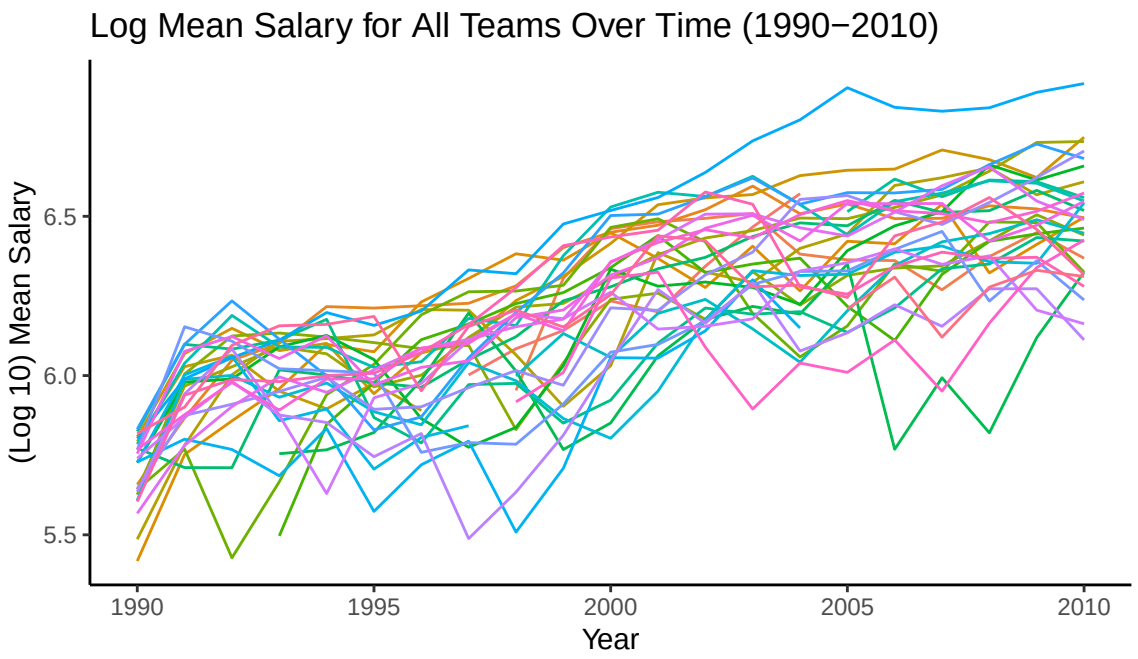
```
library("ggplot2")
```

```
mean_salary_plot <- ggplot(df_MeanSalaries, aes(x = yearID,
    y = meanSalary, group = teamID, color = teamID)) +
    geom_line() + labs(title = "Mean Salary for All Teams Over Time (1990-2010)",
```

```
    x = "Year", y = "Mean Salary") + theme_classic() +
    theme(legend.position = "none")

mean_salary_plot
```




```
log_mean_salary_plot <- ggplot(df_MeanSalaries,
    aes(x = yearID, y = log10(meanSalary),
        group = teamID, color = teamID)) +
    geom_line() + labs(title = "Log Mean Salary for All Teams Over Time (1990-2010)",
    x = "Year", y = "(Log 10) Mean Salary") +
    theme_classic() + theme(legend.position = "none")

log_mean_salary_plot
```



Across 'mean_salary_plot' and 'log_mean_salary_plot', there is a positive relationship between the mean salary of a team and the year. The plots indicate that over time, the mean salary of a team increases. The difference between the two plots is that the 'mean_salary_plot' indicates a non-linear relationship between mean salary and year and the mean salary is increasing more quickly over time. However, the 'log_mean_salary_plot' represents a more linear relationship due to the log transformation applied on the mean salary values where the salary increases more steadily over time.

c. Creating a Linear Model of Mean Salary as a Function of Year

Using the yearID variable as the sole predictor, a linear model that predicts the log in base 10 of the mean salary can be created:

```
mean_salary_model <- lm(log10(meanSalary) ~ yearID, data = df_MeanSalaries)
summary(mean_salary_model)
```

```
Call:
lm(formula = log10(meanSalary) ~ yearID, data = df_MeanSalaries)

Residuals:
     Min       1Q   Median       3Q      Max
-0.66273 -0.11316  0.01141  0.13542  0.52738

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -64.835665   2.462606  -26.33   <2e-16 ***
yearID        0.035517   0.001231   28.85   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.1818 on 606 degrees of freedom
Multiple R-squared:  0.5787,    Adjusted R-squared:  0.578
F-statistic: 832.3 on 1 and 606 DF,  p-value: < 2.2e-16
```

The model summary indicates that for each unit increase in the year, ie. for every year that passes, the log in base 10 of mean salary will increase by 0.035517 on average. The model intercept is -64.835665, which means that the model will estimate the log in base 10 of mean salary by multiplying the year by 0.035517 and subtracting 64.835665. For example, if the year given was 2000, the model would estimate the log in base 10 of the mean salary to be - 64.835665 + (2000 * 0.035517), which equates to $6.198435. The 'Multiple R-squared' value of 0.5787 indicates the year variable can explain approximately 58% of the variation of the log in base 10 of the mean salary. The p-value of 2.2e-16 can be used to determine whether or not the null hypothesis should be rejected. The null hypothesis between the two variables used to make this model would be that there is no relationship between the log in base 10 of mean salary and the year. The p-value of 2.2e-16 suggests that given the null hypothesis is true, there is a probability of 2.2e-16 that the relationship between the log in base 10 of mean salary and the year is by chance. The p-value given makes it reasonable to reject the null hypotheses, as the typical threshold given to assess whether or not to reject the null hypothesis is a p-value of 0.05 and lower, which is significantly more than the p-value given by the model.

```r
coefficients_2sfs <- round(summary(mean_salary_model)$coefficients, 2)
coefficients_2sfs
```

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -64.84       2.46  -26.33        0
yearID          0.04       0.00   28.85        0
```

The equation for the fitted model is as follows:

$$log10(meanSalary) = -64.84 + (0.04 \times yearID)$$

d. Evaluating the Assumptions of the Linear Model

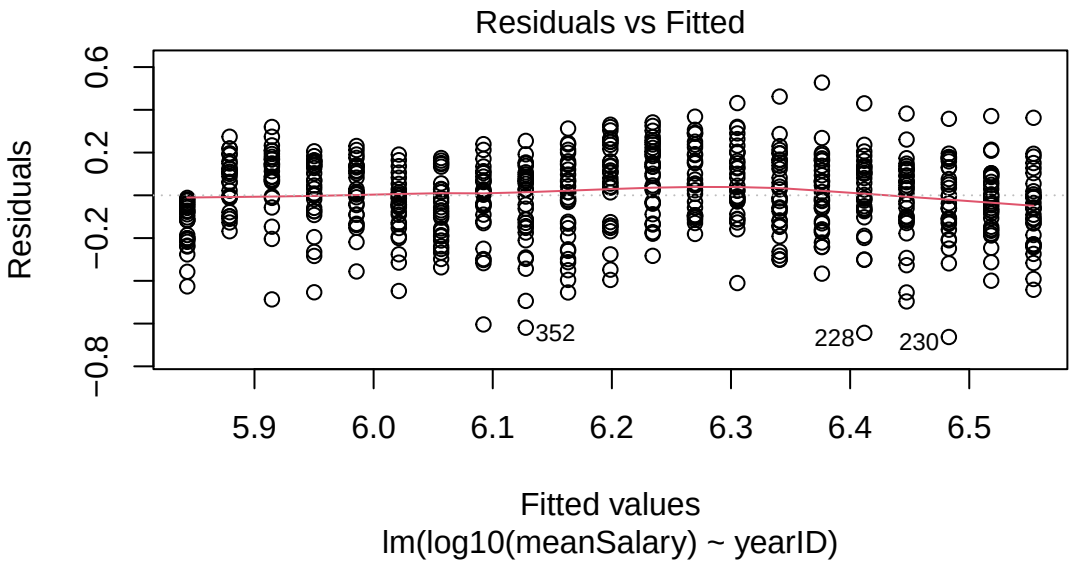
There are a few assumptions that need to be checked for in the fitted linear model:

1. The relationship between the year and the log in base 10 of the mean salary is linear
2. The variance of the residuals are assumed to have a constant variance, which means they show homoskedasticity.
3. The residual errors are normally distributed

The 'log_mean_salary_plot' indicates there is a linear relationship between the log in base 10 of mean salary and year, as the log in base 10 of mean salary does not increase more quickly or less quickly over time. Therefore, the assumption of linearity has not been violated.

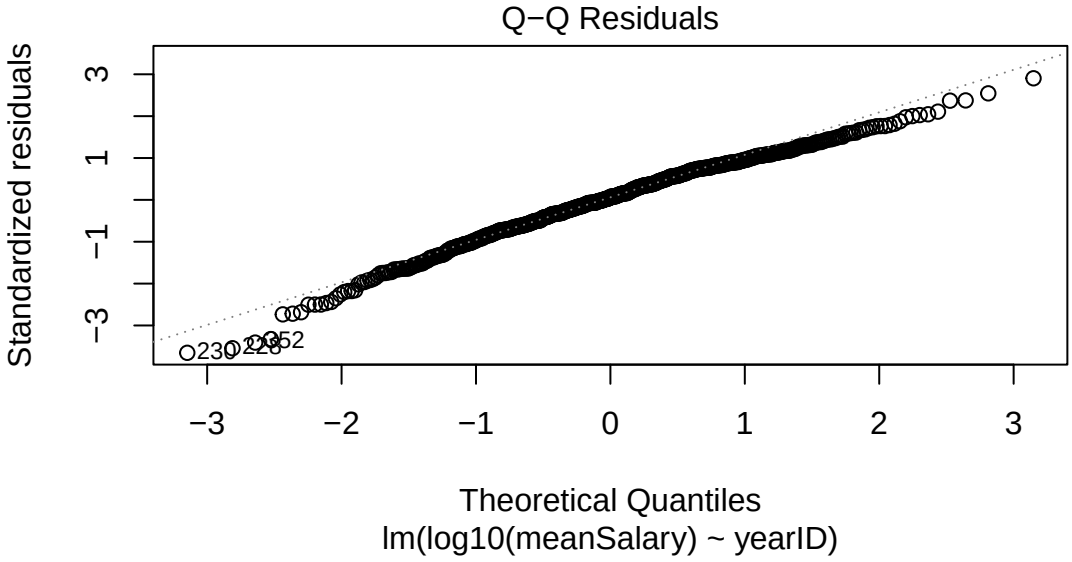
Next, the variance and distribution of the residuals need to be checked:

```r
plot(mean_salary_model, 1)
```



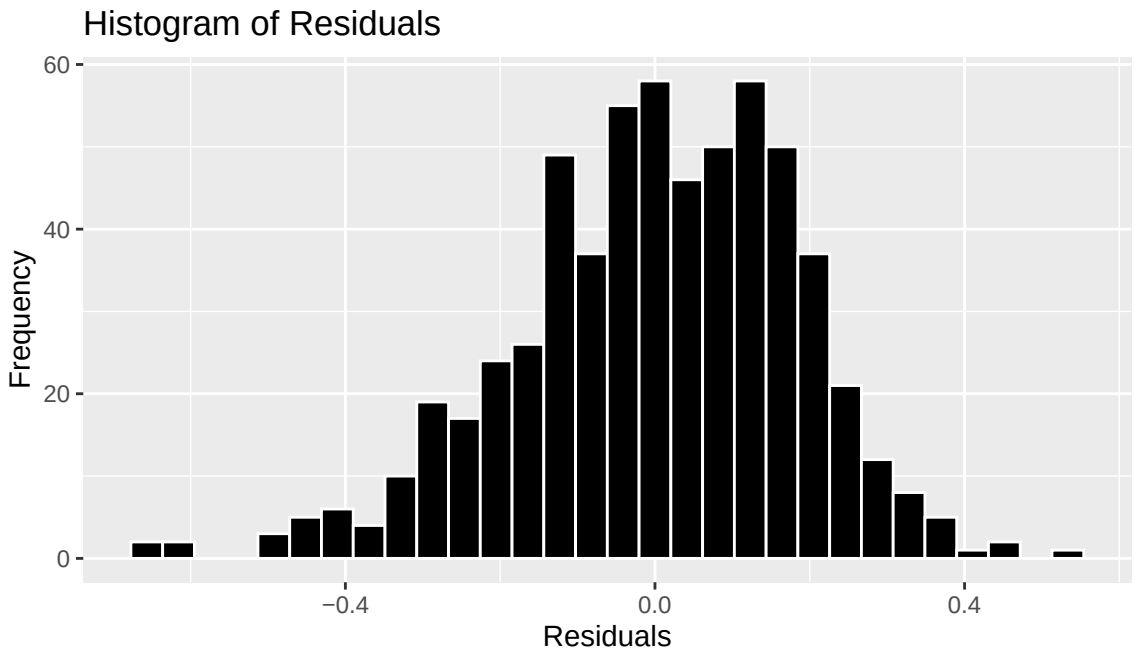
The plot of the residuals against the fitted log in base 10 of mean salary values indicates there is homoskedasticity of the residuals. The roughly horizontal line at 0 shows that there is no pattern between the residuals and the fitted values of the log in base 10 of mean salary. For example, if the residuals appeared to be increasing as the fitted values of the log in base 10 of mean salary increased, the assumption of homoskedasticity would be violated, as this would mean the fitted values are becoming less accurate with higher values of the log in base 10 of mean salary.

```
plot(mean_salary_model, 2)
```



The Q-Q plot of the standardized residuals against the theoretical quantiles of the log in base 10 of mean salary indicate the residuals follow a normal distribution. The plot shows how the data is distributed across the different quantiles, where the quantiles of the standardized residuals have been plotted against the quantiles of the log in base 10 of mean salary. Therefore, for the residuals to be normally distributed, the plot of the standardized residuals should follow approximately a straight line, which is visible in the plot.

```
ggplot(data = df_MeanSalaries, aes(x = mean_salary_model$residuals)) +
    geom_histogram(fill = 'black', color = 'white') +
    labs(title = 'Histogram of Residuals', x = 'Residuals', y = 'Frequency')
```



The histogram of the residuals is also used to check whether the variance of the residuals is normally distributed. Here, the residuals follow a roughly normal distribution, which is also evidence that the assumption of normally distributed residuals has not been violated. Overall, the assumptions associated with a linear model have not been violated and the model created is valid regarding the predictions it makes.

e. Confidence Bands of the Linear Model

Plotting the confidence bands of the model will give an indication as to how confident the model is regarding its' prediction of the true mean salary of the population:

```
confint(mean_salary_model)
```

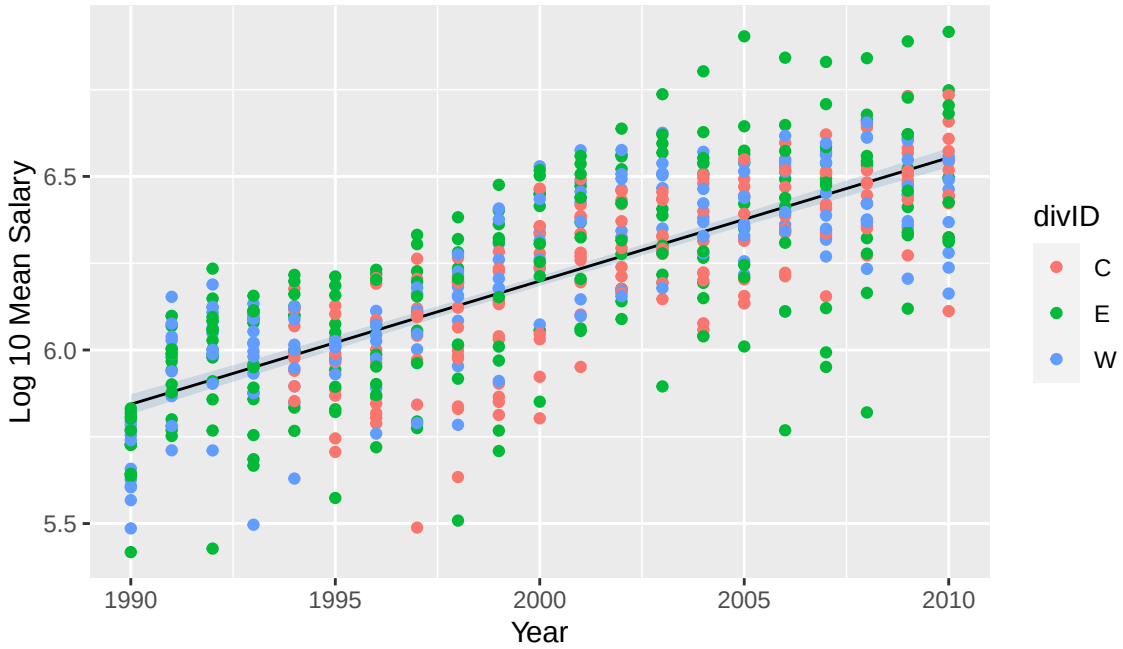
```
                 2.5 %       97.5 %
(Intercept) -69.6719437 -59.99938706
yearID        0.0330994   0.03793503
```

```
years_df <- data.frame(yearID = seq(min(df_MeanSalaries$yearID),
    max(df_MeanSalaries$yearID),
    length = 21))
```

```r
salary_predictions <- predict(mean_salary_model,
    newdata = years_df,
    interval = "confidence",
    level = 0.95)
years_df <- cbind(years_df,
    salary_predictions)
# creating a
# dataframe which
# contains the
# predictions of
# the mean salaries
# for each year, as
# well as the upper
# and lower
# intervals for
# each year
```

```r
mean_sal_and_team <- left_join(df_MeanSalaries, Teams,
    by = c("teamID", "yearID"))
# merging the 'df_MeanSalaries' dataframe with
# the 'Teams' dataframe to get additional
# variables
```

```r
ggplot(mean_sal_and_team, aes(x = yearID, y = log10(meanSalary))) +
  geom_ribbon(data = years_df, aes(ymin = lwr, ymax = upr, x = yearID),
              fill = "steelblue", alpha = 0.2, inherit.aes = FALSE) +
  geom_line(data = years_df, aes(y = fit, x = yearID)) +
  geom_point(aes(color = divID)) +
  labs(y = "Log 10 Mean Salary", x = "Year")
```



The confidence band is extremely tight around the line of fit. Therefore, the model has placed relatively

small upper and lower intervals on either side of its' predictions, which means the 95% probability that the confidence interval contains the true mean of the population is spread over a small range of mean salaries. The teams which lie outside the prediction band can also be attained:

```
mean_sal_and_team$sal_pred <- predict(mean_salary_model,
    newdata = mean_sal_and_team)
mean_sal_and_team$residuals <- mean_sal_and_team$meanSalary -
    mean_sal_and_team$sal_pred  # adding the residuals to the data frame

mean_sal_and_team$lwr_pred <- mean_sal_and_team$sal_pred -
    qt(0.975, mean_salary_model$df.residual) *
        sd(mean_sal_and_team$residuals)
mean_sal_and_team$upr_pred <- mean_sal_and_team$sal_pred +
    qt(0.975, mean_salary_model$df.residual) *
        sd(mean_sal_and_team$residuals)
# finding out the
# prediction band
# using the 97.5th
# percentile of the
# t distribution
# and standard
# deviation of the
# residuals

teams_outside_pred_band <- mean_sal_and_team %>%
    dplyr::filter(meanSalary <
        lwr_pred | meanSalary >
        upr_pred) %>%
    select(teamID) %>%
    distinct(teamID)

teams_outside_pred_band
```

```
# A tibble: 28 x 1
# Groups:   teamID [28]
   teamID
   <fct>
 1 ANA
 2 ARI
 3 ATL
 4 BAL
 5 BOS
 6 CHA
 7 CHN
 8 CIN
 9 CLE
10 COL
# i 18 more rows
```

There are 28 teams who appeared outside of the prediction band. However, there are 34 distinct teams in the original data, which means the majority of teams appeared outside the prediction band.

2. Multiple Regression for Count Data

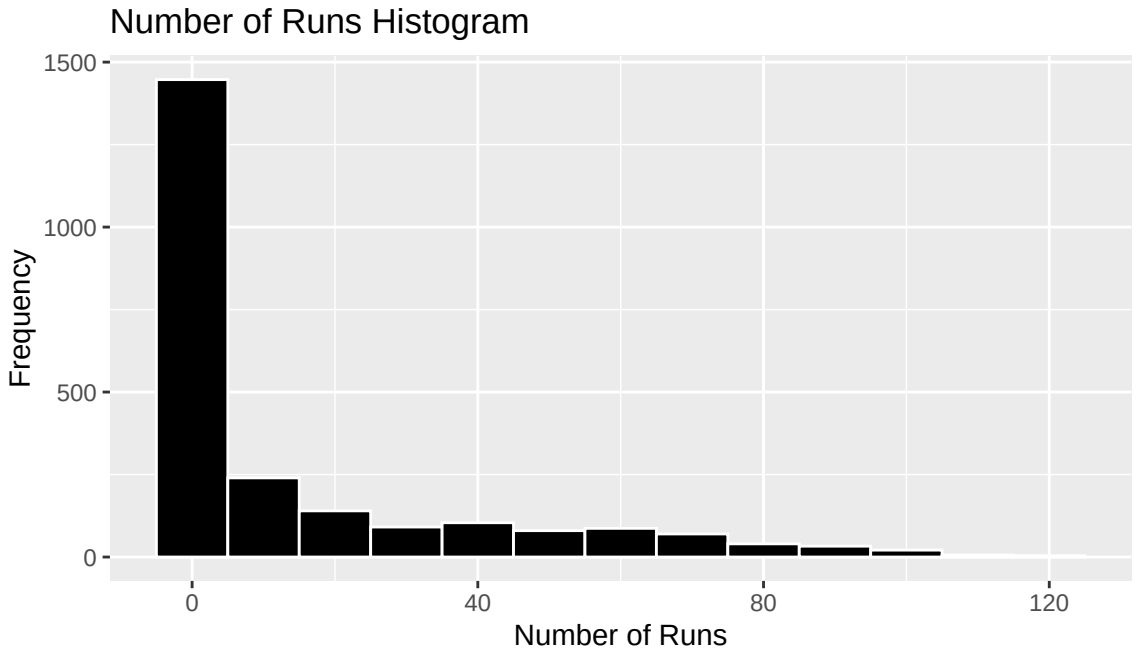
a. Creating datasets

```
df_FieldingData <- Fielding %>%
  dplyr::filter(yearID==1990 | yearID==2015) %>%
  select(playerID, yearID, POS)


df_BattingData <- Batting %>%
  dplyr::filter(yearID==1990 | yearID==2015) %>%
  merge(select(People, playerID, height, weight, birthYear), by = "playerID") %>%
  merge(select(df_FieldingData, playerID, POS), by = "playerID") %>%
  mutate(age = yearID - birthYear) %>%
  na.omit() %>%
  mutate_if(is.factor, droplevels) %>%
  group_by(playerID) %>%
  summarise(across(everything(), ~first(.))) %>%
  ungroup()
```

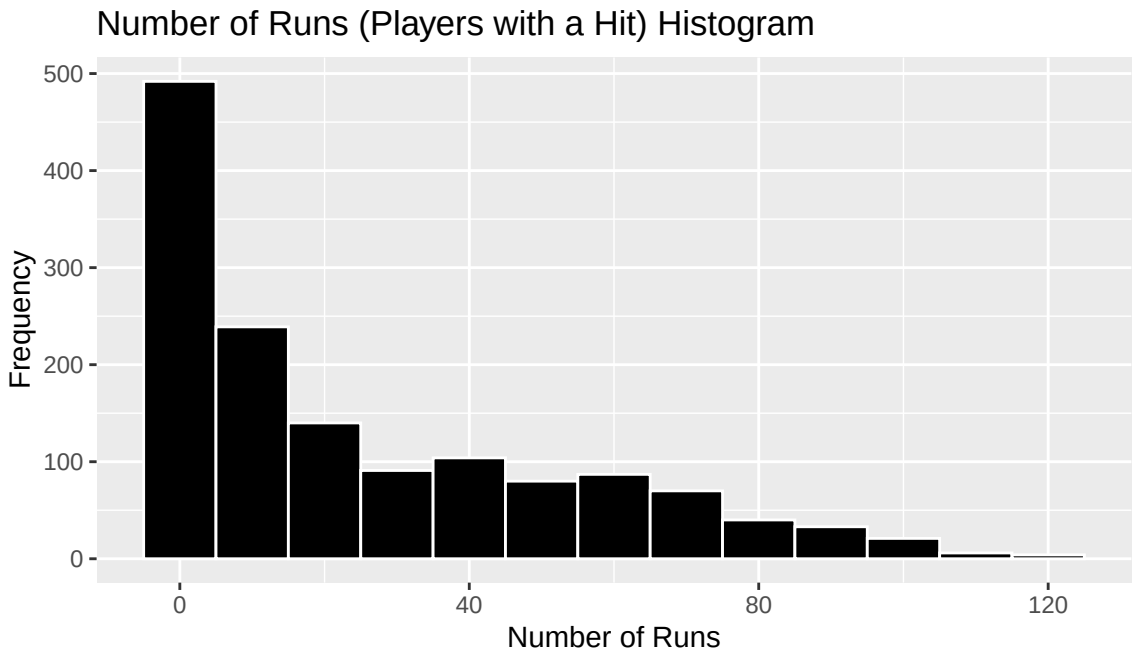
b. Number of Runs Histograms

The number of runs made by each player can be summarized using a histogram. The difference between the frequency of the number of runs made between only players who had a hit and all of the players can also be compared:

```
ggplot(df_BattingData, aes(x = R)) +
  geom_histogram(binwidth = 10, fill = "black", color = "white") +
  labs(title = "Number of Runs Histogram", x = "Number of Runs", y = "Frequency")
```



```r
ggplot(dplyr::filter(df_BattingData, H >= 1), aes(x = R)) +
    geom_histogram(binwidth = 10, fill = "black", color = "white") +
    labs(title = "Number of Runs (Players with a Hit) Histogram",
        x = "Number of Runs", y = "Frequency")
```



Creating a Poisson model of the number of runs scored using only players who have had a hit is more reasonable than creating the same model for all players, as players can only score runs if they have had a hit. Therefore, there will be a much greater number of players who did not score any runs when comparing all players versus only those who had a hit, as shown by the two histograms, where the first one has a much greater number of '0' runs scored. Focusing only on players who have had a hit is more relevant as the analysis of this data will be limited to players who have actually had the opportunity to produce runs.

c. Poisson model of Number of Runs

```r
num_runs_model <- glm(R ~ H + as.factor(yearID) +
    as.factor(POS) + height + age, data = dplyr::filter(df_BattingData,
    H >= 1), family = "poisson")
summary(num_runs_model)
```

```
Call:
glm(formula = R ~ H + as.factor(yearID) + as.factor(POS) + height +
    age, family = "poisson", data = dplyr::filter(df_BattingData,
    H >= 1))

Coefficients:
                      Estimate Std. Error z value Pr(>|z|)
(Intercept)           1.733e+00  2.002e-01   8.658  < 2e-16 ***
```

```
H                      1.414e-02  9.676e-05 146.170  < 2e-16 ***
as.factor(yearID)2015  2.583e-03  1.053e-02   0.245 0.806219
as.factor(POS)2B      -1.430e-02  1.986e-02  -0.720 0.471488
as.factor(POS)3B      -5.770e-02  2.004e-02  -2.879 0.003992 **
as.factor(POS)C       -1.461e-01  2.315e-02  -6.313 2.74e-10 ***
as.factor(POS)OF       7.714e-02  1.504e-02   5.128 2.94e-07 ***
as.factor(POS)P       -1.519e+00  4.349e-02 -34.933  < 2e-16 ***
as.factor(POS)SS      -1.769e-02  2.114e-02  -0.837 0.402738
height                 4.369e-03  2.623e-03   1.666 0.095748 .
age                    5.295e-03  1.389e-03   3.813 0.000137 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 42545.7  on 1406  degrees of freedom
Residual deviance:  6017.7  on 1396  degrees of freedom
AIC: 11771

Number of Fisher Scoring iterations: 5
```

The model summary shows that for every additional hit a player gets, the log of the number of runs they make on average is expected to be 1.414e-02 higher, with all other variables remaining constant. For every additional inch of height and year of age a player has, the model expects the log of the number of runs they make to be 4.369e-03 and 5.295e-03 higher, with other variables remaining constant, respectively. Compared to the reference year of 1990, if a player is playing in 2015, the model expects the log of the number of runs they make to be 2.583e-03 greater. Regarding the 'POS' variable, the reference level is 1B, which is first baseman. The model expects that for all other levels (positions) apart from 'OF', a player will make fewer log(runs) if they are playing in that position, compared to 1B, whereas if the player is an 'OF' the model expects the log of number of runs made to be 7.714e-02 higher compared to if they were a 1B. The model also has an intercept of 1.733e+00, which means it adds 1.733e+00 to the log of the expected number of runs when making a prediction. The null deviance figure of 42545.7 on 1406 degrees of freedom compared to the residual deviance figure of 6017.7 on 1396 degrees of freedom indicates the use of the predictors included in the model accounts for more variance in the number of runs scored than using a model without any predictor variables.

The form of the fitted model is as follows:

$$\begin{aligned} log(E(R)) = 1.73 &+ (0.014 \times H) + (0.0026 \times yearID2015) \\ &- (0.014 \times POS2B) - (0.058 \times POS3B) \\ &- (0.15 \times POSC) + (0.077 \times POSOF) \\ &- (1.5 \times POSP) - (0.018 \times POSSS) \\ &+ (0.0044 \times height) + (0.0053 \times age) \end{aligned}$$

d. Interpreting p-values

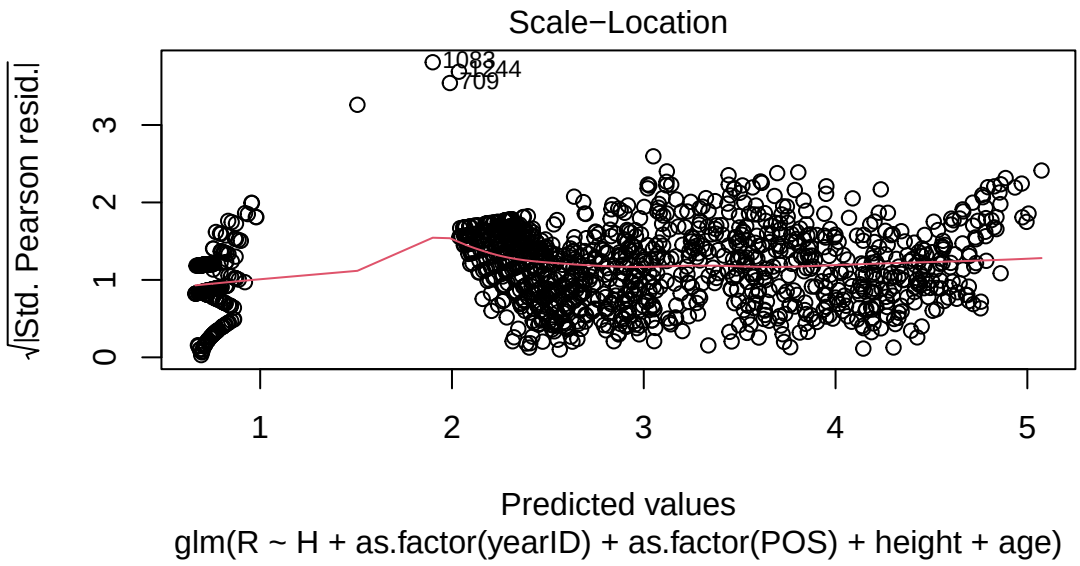
The majority of p-values associated with the model indicate the variables selected as predictors account for a significant amount of variance in the number of runs scored. The number of hits, whether a player is in the position of '3B', 'C', 'OF', or 'P' and the age of a player all have p-values associated with them of less than 0.05. Therefore, for these variables, the null hypothesis can be rejected, as there is sufficient evidence to suggest there is a relationship between these variables and the number of runs scored. For

the variables 'yearID(2015)', '(POS)2B', '(POS)SS' and 'height' the p-values are greater than 0.05. These p-values suggest there is not sufficient evidence to reject the null hypothesis that there is no relationship between these variables and the number of runs scored.

e. Poisson Model Assumptions

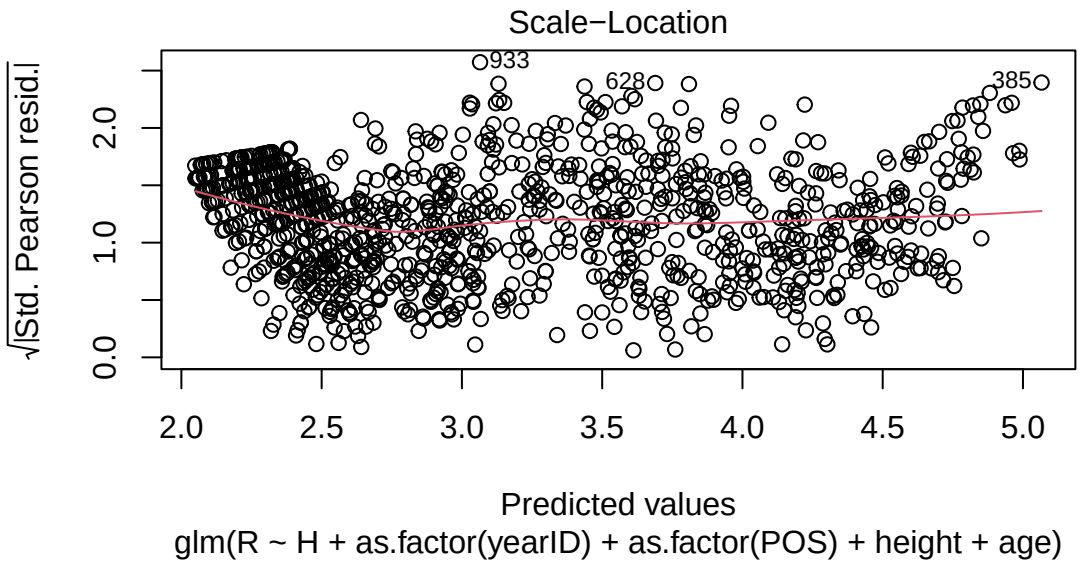
The first poisson model assumption is that the variance is equal to the mean. The assumption is based on the variance of a count variable increasing as the mean increasing.

```
plot(num_runs_model, which=3)
```



The plot of standardized residual values versus predicted values shows the line is not flat, as the standardized residuals shoot up and down around the prediction value of 2, which means the standardized residual variability is not constant and there is some overdispersion. Overdispersion is where the observed variance is greater than the mean and can cause the standard error to be underestimated, which will reduce the p-value and cause a false rejection of the null hypothesis. The overdispersion may have been cause by missing key predictor variables associated with the number of runs made, or potentially by a correlation between two predictor variables. There is also a cluster of points around 1 for the predicted values, which are reducing the straightness of the line.

```
num_runs_model_2 <- glm(R ~ H + as.factor(yearID) +
    as.factor(POS) + height + age, data = dplyr::filter(df_BattingData,
    H >= 1 & POS != "P"), family = "poisson")  # remove 'P' position
plot(num_runs_model_2, which = 3)
```



When removing rows where the POS variable is 'P' (Pitchers), the standardized residuals against predicted values plot gives a straighter line, indicating there is less overdispersion. Less overdispersion may be because pitchers who are batting are not expected to make as many runs, as making runs is not their specialty. Therefore, removing pitchers removes a cluster of low run scorers. However, overdispersion is still present, as the level of the standardized residulas still drops as the predicted values move from 2 to 2.5, before rising again slowly.

Overdispersion can also be checked using a dispersiontest amd calculating the alpha value:

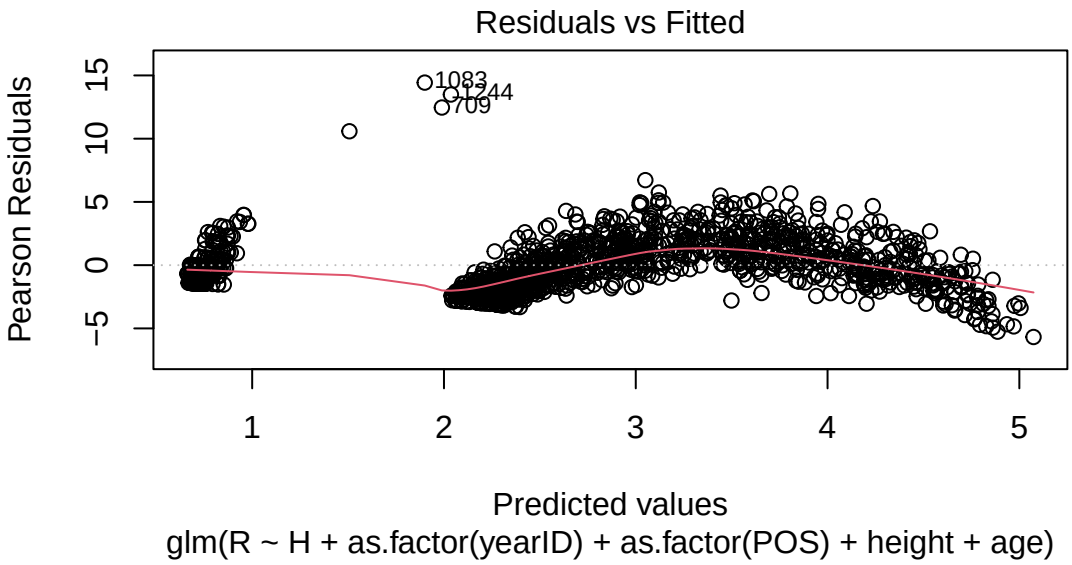
```
dispersiontest(num_runs_model, trafo=1)
```

```
    Overdispersion test

data:  num_runs_model
z = 11.981, p-value < 2.2e-16
alternative hypothesis: true alpha is greater than 0
sample estimates:
   alpha
3.118336
```

The alpha value of 3.118336 is larger than 0, which also suggests there is overdispersion and the assumption of the variance being equal to the mean being violated. However, the level of overdispersion does not necessarily mean a different model is the immediate solution.

The second assumption of a poisson model is linearity, which can be assessed by examining a plot of the residuals versus the predicted values:

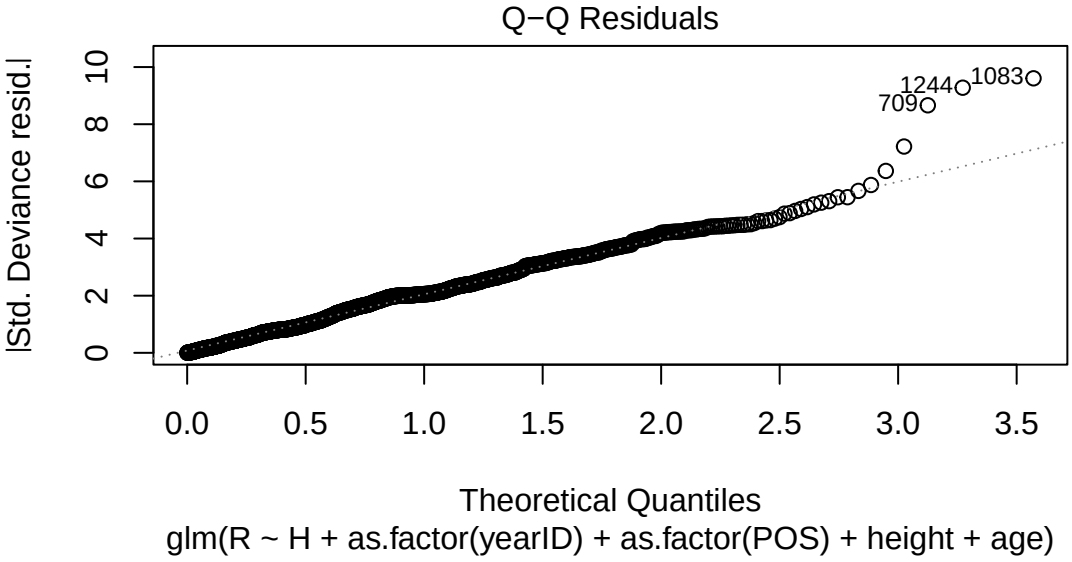
```
plot(num_runs_model,which=1)
```



The residuals are also not constant, as the plot shows a line which drops below and rises above 0 several times. The dropping below and rising above 0 indicates there is a pattern in the value of the residuals, so the assumption of constant residuals has also been violated.

Finally the distribution of the residuals can be examine via a QQ plot:

```
plot(num_runs_model,which=2)
```



The plot of the standard deviance of residuals should follow a straight line for the residuals to represent

normal distribution. However, the plot shows the presence of extreme points, which means a straight line is not followed and the assumption of normally distributed residuals has been violated.

f. Including teamID as a random effect

Creating a model that uses the 'teamID' variable as a random effect can show how the team a player plays for affects the number of runs they make, but without having to list all of the different teams as a separate categorical variable:

```
num_runs_model_random_effect <- glmer(R ~
    H + as.factor(yearID) + as.factor(POS) +
        height + age + (1 | teamID), data = dplyr::filter(df_BattingData,
    H >= 1), family = "poisson", nAGQ = 0)
num_runs_model_random_effect
```

```
Generalized linear mixed model fit by maximum likelihood (Adaptive
  Gauss-Hermite Quadrature, nAGQ = 0) [glmerMod]
 Family: poisson  ( log )
Formula: R ~ H + as.factor(yearID) + as.factor(POS) + height + age + (1 |
    teamID)
   Data: dplyr::filter(df_BattingData, H >= 1)
      AIC       BIC    logLik  deviance  df.resid
11693.165 11756.156 -5834.583 11669.165      1395
Random effects:
 Groups Name        Std.Dev.
 teamID (Intercept) 0.06442
Number of obs: 1407, groups:  teamID, 33
Fixed Effects:
          (Intercept)                     H  as.factor(yearID)2015
              1.871150              0.014277                0.016388
     as.factor(POS)2B       as.factor(POS)3B        as.factor(POS)C
             -0.015695             -0.066862               -0.141220
     as.factor(POS)OF        as.factor(POS)P       as.factor(POS)SS
              0.077603             -1.494296               -0.019047
               height                    age
              0.002473              0.004601
```

The model indicates the estimated standard deviation of the 'teamID' variable was 0.06442. Therefore, the model estimates there is a variability of 0.06442 runs made on average, across all the different teams, which means teams that are significantly above or below this average would significantly improve or impair the number of runs a player makes, indicating the team a player plays for to be an important factor in determining the number of runs a player makes.

g. Making a Prediction

Finally, the model can be used to make a prediction on the number of runs a player makes, using the player's values for the variables used to create the model:

```
num_runs_pred <- predict(num_runs_model_random_effect, newdata =  data.frame(H = 50,
  yearID = factor(2015),
  POS = factor("OF"),
```

```
  height = 85,
  age = 27,
  teamID = factor("CLE")), type = "response")

print(round(num_runs_pred))
```

```
 1
21
```

The model predicts that a player who is 27, is 85 inches tall, is an outfielder for Cleveland Indians, played in 2015 and had 50 hits to score 21 runs.

```
similar_players <- df_BattingData %>%
  dplyr::filter(yearID == 2015 & POS == "OF" & teamID == "CLE" & H>= 1) %>%
  select(R) # finding statistics for runs of cleveland players

summary(similar_players)
```

```
       R
 Min.   : 2.0
 1st Qu.:22.0
 Median :30.0
 Mean   :31.8
 3rd Qu.:37.0
 Max.   :68.0
```

The median number of runs for 'OF' players in 2015 who played for Cleveland is 30 and the mean was 32. Therefore, the prediction is smaller than would be expected given the player's attributes, which indicates the model's accuracy is not optimal, but is still a reasonable prediction.

3. Lasso Regression for Logistic Regression

a. Creating a Div Winners Data Frame

A Data Frame that contains none of the team identification variables will be created, which will then be used to create training and test data and create a logistic regression model:

```
df_DivWinners <- Teams %>%
    dplyr::filter(yearID %in% 1990:2010) %>%
    select(-c("teamID", "teamIDBR", "teamIDlahman45",
        "teamIDretro", "lgID", "Rank", "franchID",
        "divID", "WCWin", "LgWin", "WSWin",
        "name", "park")) %>%
    na.omit()
```

```
set.seed(123)
df_DivWinners$DivWin <- as.factor(df_DivWinners$DivWin)
# converting the DivWin variable into a
# factor
training_rows <- createDataPartition(df_DivWinners$DivWin,
```

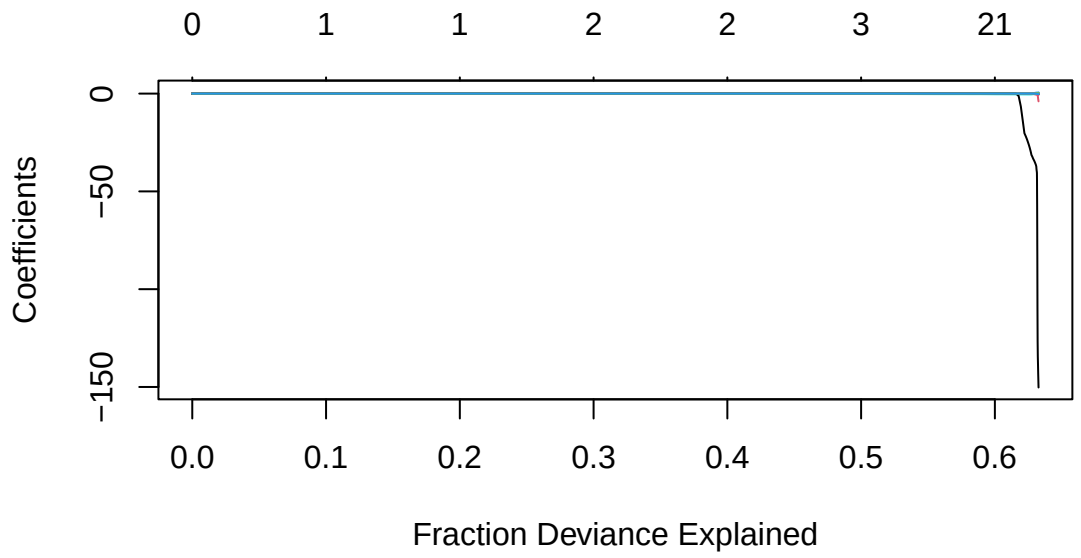
```
    p = 0.8, list = FALSE)
training_data <- df_DivWinners[training_rows,
    ]
test_data <- df_DivWinners[-training_rows,
    ]  # creating training and testing data



train_predictors <- model.matrix(~. - 1,
    subset(training_data, select = -c(DivWin)))
train_response <- training_data$DivWin
```

b. Creating a GLMNet and Plotting Residual Deviance

Using GLMNet, the 'optimal' number of predictors can be ascertained:

```
div_win_gmnet <- glmnet(train_predictors, train_response, alpha = 1, family="binomial")
```

```
plot(div_win_gmnet, xvar = "dev")
```

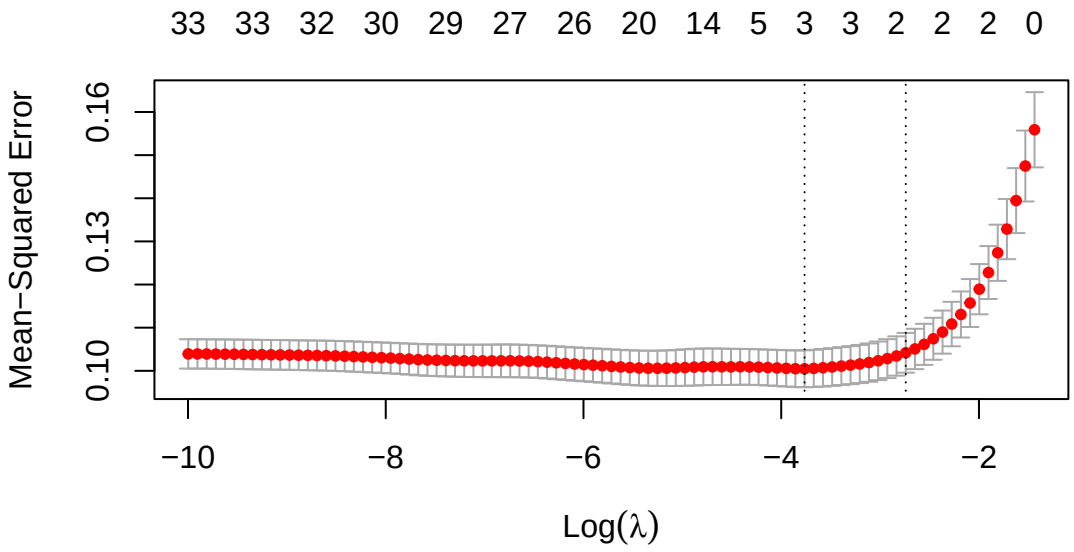


The deviance against the number of predictors shows that over 20 predictors are necessary to account for 60% of the deviance.

c. Using Cross Validation

Using cross validation, a range of optimal values for lambda can be calculated:

```
train_response_2 <- as.numeric(train_response) -
    1  # changing the response to numeric so it can be used in cv.glmnet
div_win_cross_val <- cv.glmnet(train_predictors,
    train_response_2, type.measure = "auc")
plot(div_win_cross_val)
```



The plot shows log(lambda) on the x axis and the error measured on the y axis. If a log(lambda) between -2.5 and -3.8 is chosen model performance will be similar. This information can be used to find out which predictors lead to a log lambda value of -3, which would be conservative, as it would not give as many predictors as choosing a log lambda value of -3.8, but is still in the 'optimal' range.

```
optimal_coeffs <- coef(div_win_gmnet,
    s = exp(-3), exact = TRUE,
    x = train_predictors,
    y = train_response)  # find the coefficients at which log lambda = -3
coefficients_not_zero <- which(optimal_coeffs !=
    0)  # select non-zero coefficients
optimal_predictors <- rownames(optimal_coeffs)[coefficients_not_zero]
optimal_predictors  # select the names of the predictors
```

```
[1] "(Intercept)" "W"           "L"
```

The 'W' and 'L' have been chosen through cross validation as the predictors which explain the 'DivWin' response variable most, other than the intercept. This result makes sense, as the number of games a team wins and loses is the greatest factor in them winning the division, so these variables will be used to create the logistic regression model.

d. Creating a Logistic Regression Model

Next, the logistic regression model can be created, where 'DivWin' is used as the response variable and 'W' and 'L' are the predictors

```r
div_win_logistic <- glm(DivWin ~ W + L, family = "binomial", data = training_data)
summary(div_win_logistic)
```

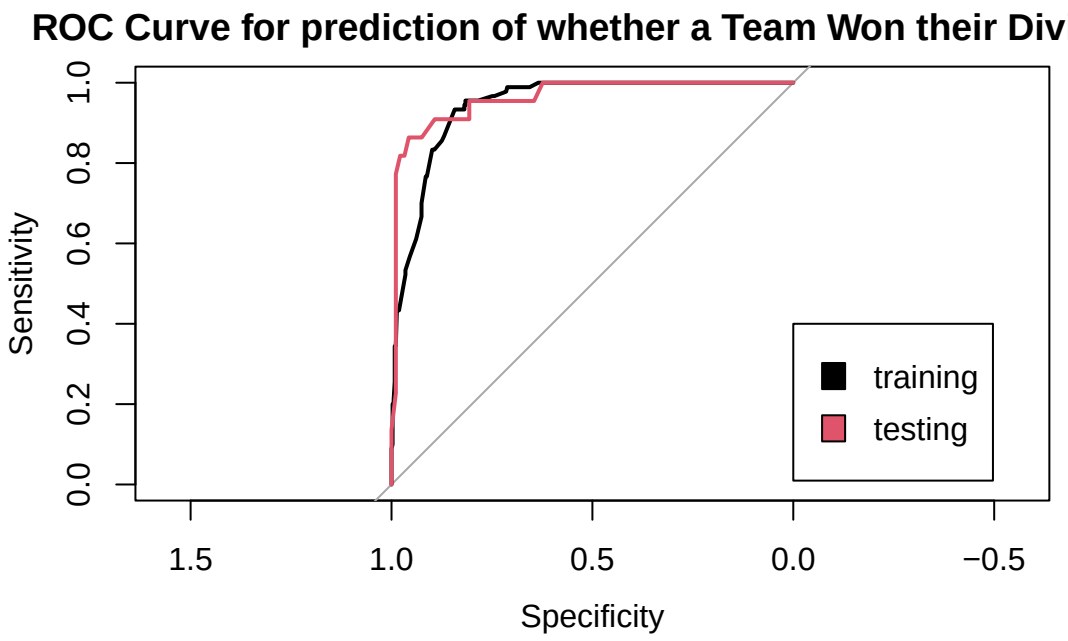
```
Call:
glm(formula = DivWin ~ W + L, family = "binomial", data = training_data)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  9.62134    6.97382   1.380   0.1677
W            0.07297    0.04295   1.699   0.0894 .
L           -0.23478    0.05060  -4.640 3.48e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 456.93  on 464  degrees of freedom
Residual deviance: 220.23  on 462  degrees of freedom
AIC: 226.23

Number of Fisher Scoring iterations: 7
```

The following is the form of the fitted model:

$$logit(p(Y)) = 9.62 + (0.07 \times W) + (-0.23 \times L)$$

```r
div_win_train_preds <- predict(div_win_logistic, type = "response")
div_win_test_preds <- predict(div_win_logistic, newdata = test_data, type = "response")
```

```r
roctrain <- roc(response = training_data$DivWin,
    predictor = div_win_train_preds,
    plot = TRUE, main = "ROC Curve for prediction of whether a Team Won their Division",
    auc = TRUE)
roc(response = test_data$DivWin,
    predictor = div_win_test_preds,
    plot = TRUE, auc = TRUE,
    add = TRUE, col = 2)
legend(0, 0.4, legend = c("training",
    "testing"), fill = 1:2)
```




```
Call:
roc.default(response = test_data$DivWin, predictor = div_win_test_preds,     auc = TRUE, plot = TRUE, a

Data: div_win_test_preds in 93 controls (test_data$DivWin N) < 22 cases (test_data$DivWin Y).
Area under the curve: 0.9616
```

The ROC curve shows the model is neither underfit or overfit on the training data. Comparing the testing and training specificty/sensitivity scores, the training data scores are higher than the testing data scores for a smaller range of values than the testing data scores are higher than the training data scores. Therefore, the model is not significantly outperforming its predictions on the training data than the testing data, so is not overfit. The scores also remain above the random classification predictor line, indicating the model is not underfit, as it is producing better results than would be attained via randomly guessing whether or not a team won its division.

e. Youden's index and Confusion Matrices

Youden's index can be used to find out the optimal threshold at which the area under the curve is maximised, which essentially gives the maximum sum of sensitivity and specificity.

```
youden_divwin <- coords(roctrain, "b", best.method="youden", transpose=TRUE)
youden_divwin
youden_divwin[2] + youden_divwin[3]
```

```
  threshold specificity sensitivity
  0.2023967   0.8426667   0.9333333
specificity
      1.776
```

The best threshold is at 0.2, which achieves a sum of sensitivity and specificity of 1.776, which is a significant improvement on the 1 which would be achieved through random classification. The 0.2 threshold can be used to create confusion matrices for both the training and test data:

```
training_data$winpred <- ifelse(predict(div_win_logistic,
    newdata = training_data, type = "response") >=
    0.2, "Y", "N")
table(training_data$winpred, as.factor(training_data$DivWin))
```

```
     N   Y
  N 316   6
  Y  59  84
```

```
test_data$winpred <- ifelse(predict(div_win_logistic,
    newdata = test_data, type = "response") >= 0.2,
    "Y", "N")
table(test_data$winpred, as.factor(test_data$DivWin))
```

```
    N  Y
  N 75  2
  Y 18 20
```

The confusion matrices for the testing and train datasets show the model to have predicted 316 / 376 non division winners and 84 / 90 division winners correctly on the training data. On the test data, the model predicted 75 / 93 non division winners correctly and 20 / 22 division winners correctly. These figures show the threshold of 0.2 to be very good regarding improving model performance to appropriately classify division winners.

f. Sensitivity and Specificity as a function of divID

The sensitivity and specificity of the test data can also be calculated as a function of 'divID'. First, the 'divID' variable needs to be added back to the test data:

```
df_DivWinners_two <- Teams %>%
    dplyr::filter(yearID %in%
        1990:2010) %>%
    select(-c("teamID",
        "teamIDBR", "teamIDlahman45",
        "teamIDretro",
        "lgID", "Rank",
        "franchID", "WCWin",
        "LgWin", "WSWin",
        "name", "park")) %>%
    na.omit()  # creating initial dataset


set.seed(123)
df_DivWinners_two$DivWin <- as.factor(df_DivWinners_two$DivWin)
training_rows_two <- createDataPartition(df_DivWinners_two$DivWin,
```

```
    p = 0.8, list = FALSE)
test_data_two <- df_DivWinners_two[-training_rows_two,
    ]
# creating test
# data with the
# same random seed
```

Then, the predictions need to be added to the test data:

```
test_data_two$winpred <- ifelse(predict(div_win_logistic,
    newdata = test_data_two, type = "response") >=
    0.2, "Y", "N")
```

Grouping the data by each different division will allow the sensitivity and specificity to be calculated for each one. Using a function to calculate the sensitivity and specificity will also help:

```
sens_spec_func <- function(real, pred) {
  true_pos <- sum(real == "Y" & pred == "Y")
  true_neg <- sum(real == "N" & pred == "N")
  false_pos <- sum(real == "N" & pred == "Y")
  false_neg <- sum(real == "Y" & pred == "N")

  sensitivity <- true_pos / (true_pos + false_neg)
  specificity <- true_neg / (true_neg + false_pos)

  return(list(sensitivity = sensitivity, specificity = specificity))
}
```

```
div_id_sens_spec <- test_data_two %>%
  group_by(divID) %>%
  summarise(
    sensitivity = sens_spec_func(DivWin, winpred)$sensitivity,
    specificity = sens_spec_func(DivWin, winpred)$specificity,
    sens_spec_sum = sensitivity + specificity
  )

div_id_sens_spec
```

```
# A tibble: 3 x 4
  divID sensitivity specificity sens_spec_sum
  <chr>       <dbl>       <dbl>         <dbl>
1 C           1           0.938          1.94
2 E           0.889       0.788          1.68
3 W           0.857       0.679          1.54
```

The 'div_id_sens_spec' data shows the sensitivity and specificity is greatest in the 'C' division with a sum of 1.94 division and smallest in the 'W' division with a sum of 1.54, indicating the model is most accurate at predicting whether a team won its' division if they play in the 'C' division and least accurate if they play in the 'W' division. The results can also be shown in a bar chart:

```
ggplot(div_id_sens_spec, aes(x = divID, y = sens_spec_sum,
    fill = divID)) + geom_bar(stat = "identity",
    position = "dodge") + labs(title = "Sensitivity and Specificity Sum For Each Division",
    x = "Division", y = "Sensitivity and Specificity Sum")
```