Lectures 3 + 4 - Basic Linear Modeling

Will Calandra

Netflix Clip - The Great Hack

- We will watch a clip from the documentary “The Great Hack,” which gives us a look behind the scenes of Cambridge Analytica’s role in the 2016 election.
- Their strategy was to gather data from prospective voters’ Facebook profiles (their likes, comments, friends, groups, etc.) and find out what would persuade them to vote for their candidate. They were quite brilliant in how they targeted a select portion (“the persuadables”) of the population to flip for their candidate. However, their methods were a huge breach of data privacy and ethical standards. More on this below.
- People had no idea that they were a part of Cambridge Analytica’s experiment, as the company psychologically profiled people and sent back targeted ads to generate a reaction that they hoped would translate into a vote for their candidate. These ads didn’t have to be truthful, just engaging, which can be dangerous and manipulative. The big issue is that they tapped into all of people’s data and experimented on them without their consent.
- It doesn’t matter if Cambridge Analytica was working for the left or right - this is a landmark breach of ethics that is worth studying. There are entire fields around ethics and “human-centered” technology that are fascinating - I highly recommend you do some research to learn more. People like Tristan Harris, Jaron Lanier, Cathy O’Neil, Joy Buolamwini (founder of Algorithmic Justice League - that sticker on my computer) are leading the charge to police the usage of algorithms and raise awareness as to how they are impacting society. Look ’em up!
- Why am I showing you this? Well, as you learn more about R and data, it’s important to recognize that these tools are powerful (remember the TED talk), and algorithms in practice can be very influential. As such, with great power comes great responsibility. Down the line, remember the goals and assumptions you make as you build models and algorithms, and understand the potential effects your algorithm can have in practice, both good and bad. Often, we find intention and consequence don’t match up (and I think the social media giants are realizing this now)!

Basic Linear Modeling

In this lecture, we will bring together our learning of basic R and data structures to evaluate the performance of ESPN Fantasy Football projections. We will import a real-world dataset of 2019 ESPN projection data from the web, manipulate the dataset to analyze projections of interest, and generate some basic plots and summary statistics to interpret our results. We will then build our own model to replicate the fantasy football scoring “equation” and compare our results with the ESPN projections.

Import Data

First, we will import the dataset from the fantasy football pros website: https://www.fantasyfootballdatapros.com/csv_files (https://www.fantasyfootballdatapros.com/csv_files)

The .csv download is also in our shared Google Drive. In order to import this dataset into R, in the top right of the RStudio IDE, click on the “import dataset” dropdown and select “from Excel” to load the file into R with ease.

Notice that a dialogue box comes up and gives you an error. This is because the file is not an Excel file but rather a .csv file. No need to panic, we just want to use the code given in the bottom right as a “skeleton” for importing our dataset. Copy the code below to your clipboard - it should look something like this:

```
#Library(readxl)
#X2019FantasyESPN <- read_excel("Your computer path/2019FantasyESPN.csv")
#View(X2019FantasyESPN)
```

Notice that the code imports the library(readxl), saves the file as a “X2019FantasyESPN” object by calling the read_excel() function, and then calls the view() function so you can see how the data is loaded in R. In our case, we will use this structure to import the data, but we will change the functions. We will also get rid of the view() function, because it is not useful in scripting code (it will coerce you to view the file every time you run it - we only care about seeing the data the first time after import).

To read .csv files, we use this code:

```
# Load data
library(utils)
#ESPN_Proj <- read.csv("Your computer path/2019FantasyESPN.csv")
```

Notice how we used the library(utils) and the read.csv() function to load this dataset into R. Also notice the full file path of the .csv file is in my function; this is actually bad practice. An improved version of the code would not have the full file path. In order to achieve this, we would move the file to the same folder as this .R file, and run the code below:

```
# Load data
ESPN_Proj <- read.csv("2019FantasyESPN.csv")
```

The hashtag in front of the ESPN_Proj above is removed so things compile. This is good code! The file should have shown up in your global environment panel in the top right - click on it and see the dataframe appear! Neat, right?

Visualize data

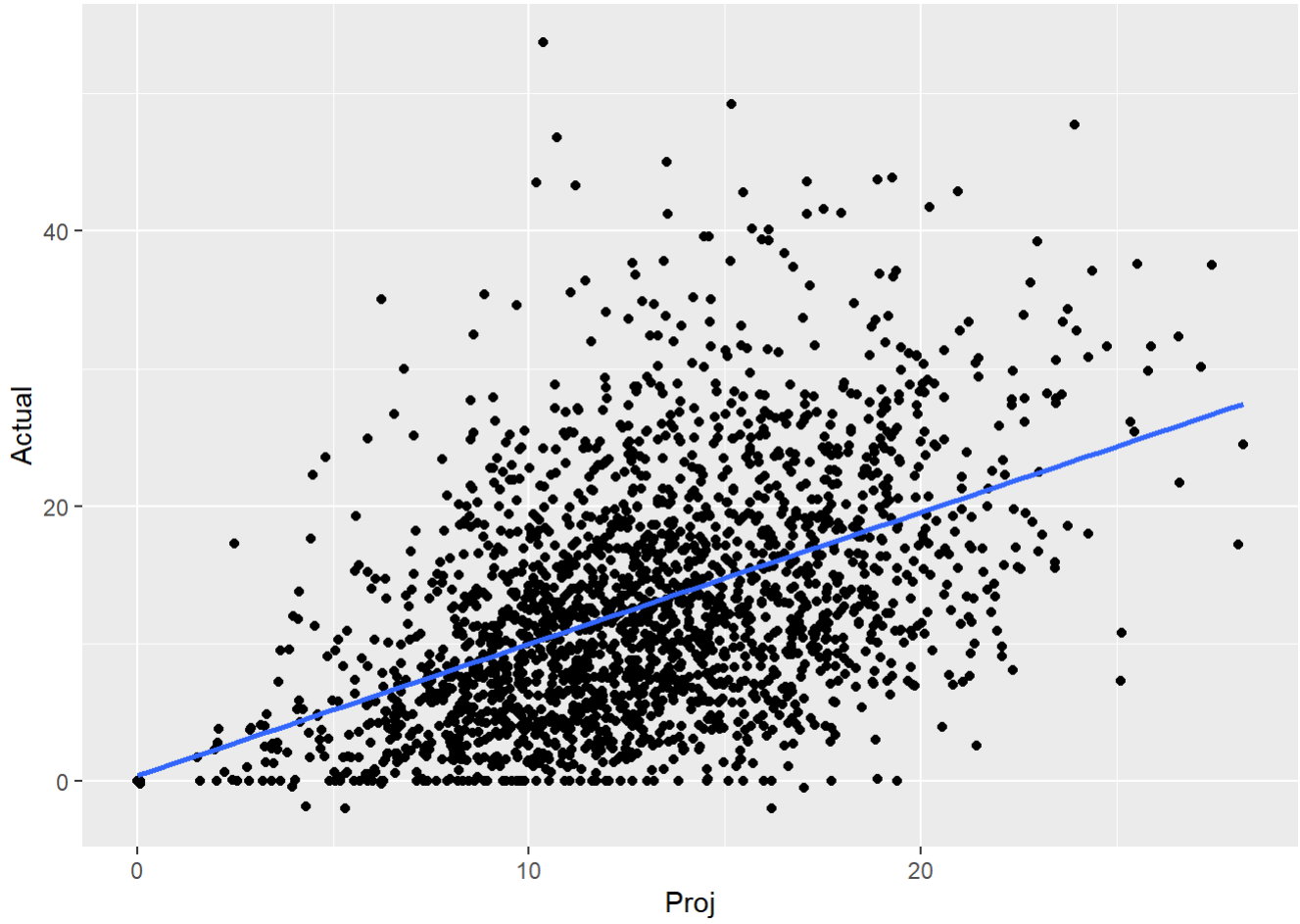
We will now conduct an exploratory analysis of the data we have. Let’s use the library(ggplot2) package to visualize the dataset. If you click on the ESPN_Proj you’ll notice we have a “Proj” and “Actual” column. This is ESPN’s weekly fantasy projections versus the actual output. We will now build a scatter plot to see the relationship between these two variables. Is ESPN good at projecting players each week? If this is the case, what would we expect to see? Code below:

```
# Visualize performance
library(ggplot2)
ggplot(ESPN_Proj, aes(x=Proj,y=Actual)) + geom_point() + geom_smooth(method = lm, se=FALSE)
```

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

```
## Warning: Removed 154 rows containing non-finite values (`stat_smooth()`).
```

```
## Warning: Removed 154 rows containing missing values (`geom_point()`).
```



If ESPN is good at projecting players, we would expect to see a one-to-one relationship between these variables (a line with slope at 1, following the x=y line).

Let's walk through the code. Notice that we call the ggplot() function. It accepts many parameters (which can be read in documentation), but for this analysis, we have the dataset (ESPN_Proj), a comma, and then what is called the aes(), or aesthetic. This is where you tell ggplot which variables you want to visualize. For us, we put the Proj variable on the x axis, and the Actual variable on the y axis. We then add more capability to ggplot() by adding the "+" operator, and then call the geom_point() function to tell ggplot() to plot the points of our data (no need to put anything in the parentheses, we already told ggplot what to plot in the aes()). To then draw the line representing the scatter plot's linear relationship, we add the geom_smooth() capability, the method is a lm (linear model), and the se is false (we don't display standard error here). Boom! Visual done.

Visually, ESPN's performance doesn't look bad - the slope is positive, and it seems like the data follows an upward trend in actual results v projected points. Notice how there is an error message here, as 154 rows were removed from our visualization. This is because there are some NA values in these columns; we can scroll through the ESPN_Proj to see these values in the "Actual" column. This represents some data loss in our dataset. A good test for missing values in your data can be done by typing this in the console and hitting enter:

```
# Count NA values
sum(is.na(ESPN_Proj$Actual))
```

```
## [1] 154
```

Notice that “154” displays in the console (a good refresher from our data structures lecture)! We will continue with this analysis knowing that there will be some data points that were not captured. Since there are only 154 points missing and we have 2577 observations, we will continue the analysis.

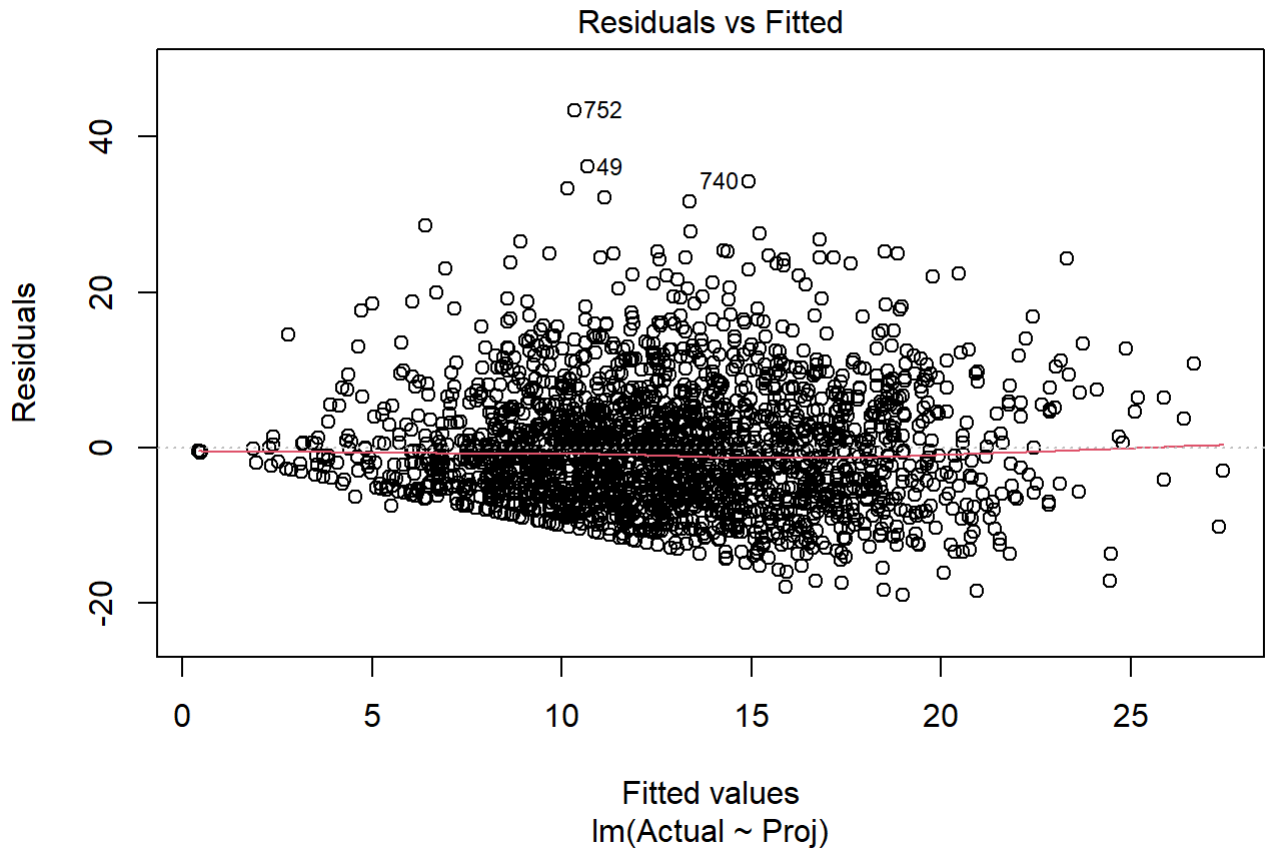
Linear model

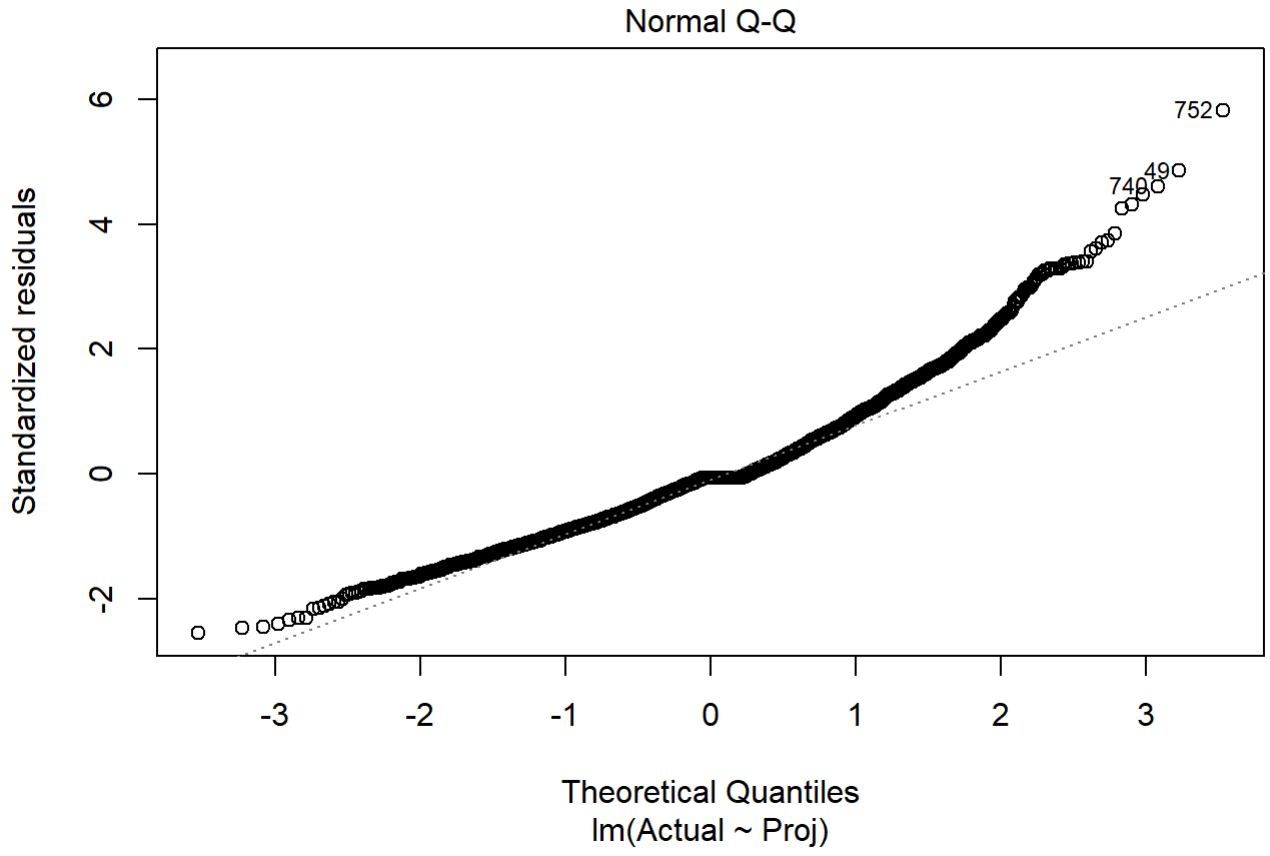
We will now fit a linear model on the Actual v Projected results to quantify what we see from this visualization. Let’s run these lines of code to see what we get!

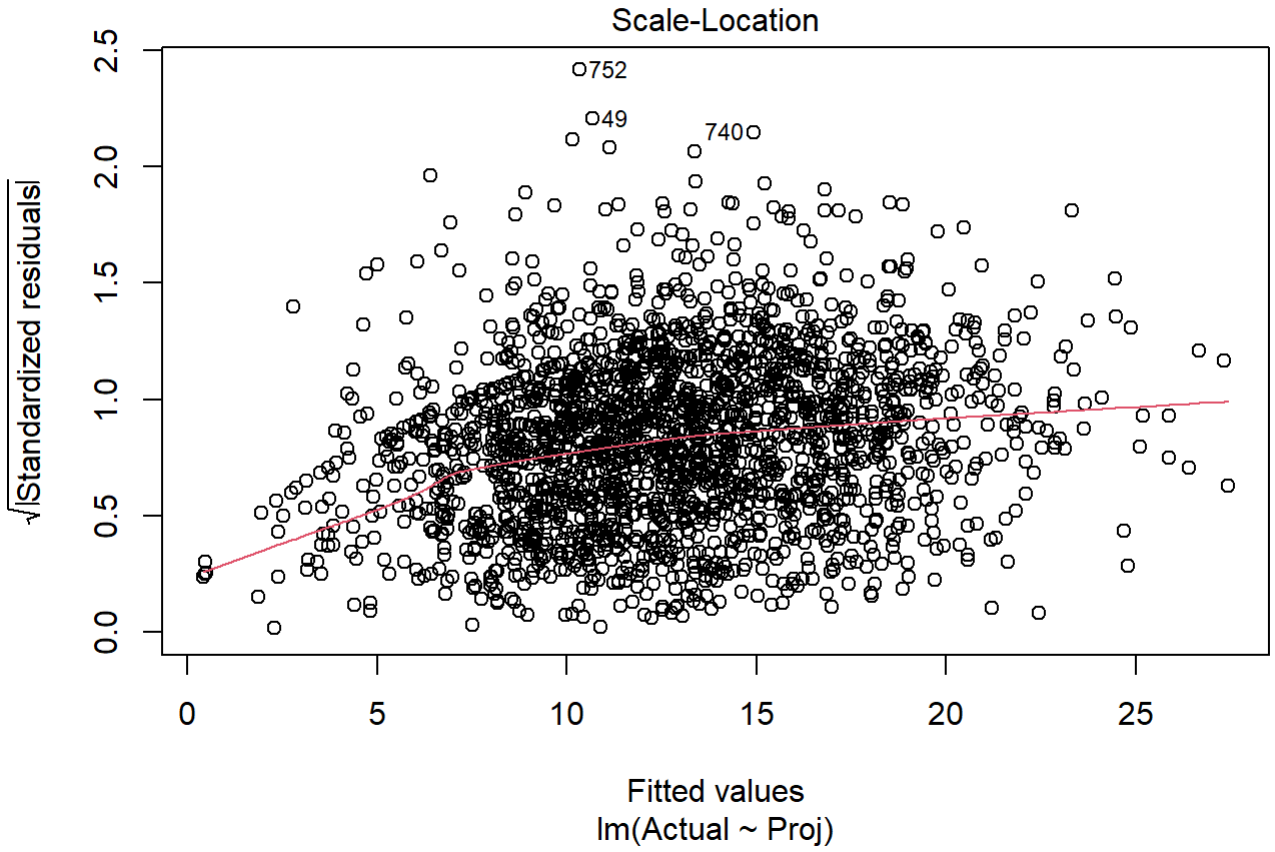
```
# Linear model on performance
Lm <- lm(data = ESPN_Proj, Actual ~ Proj)
summary(Lm)
```

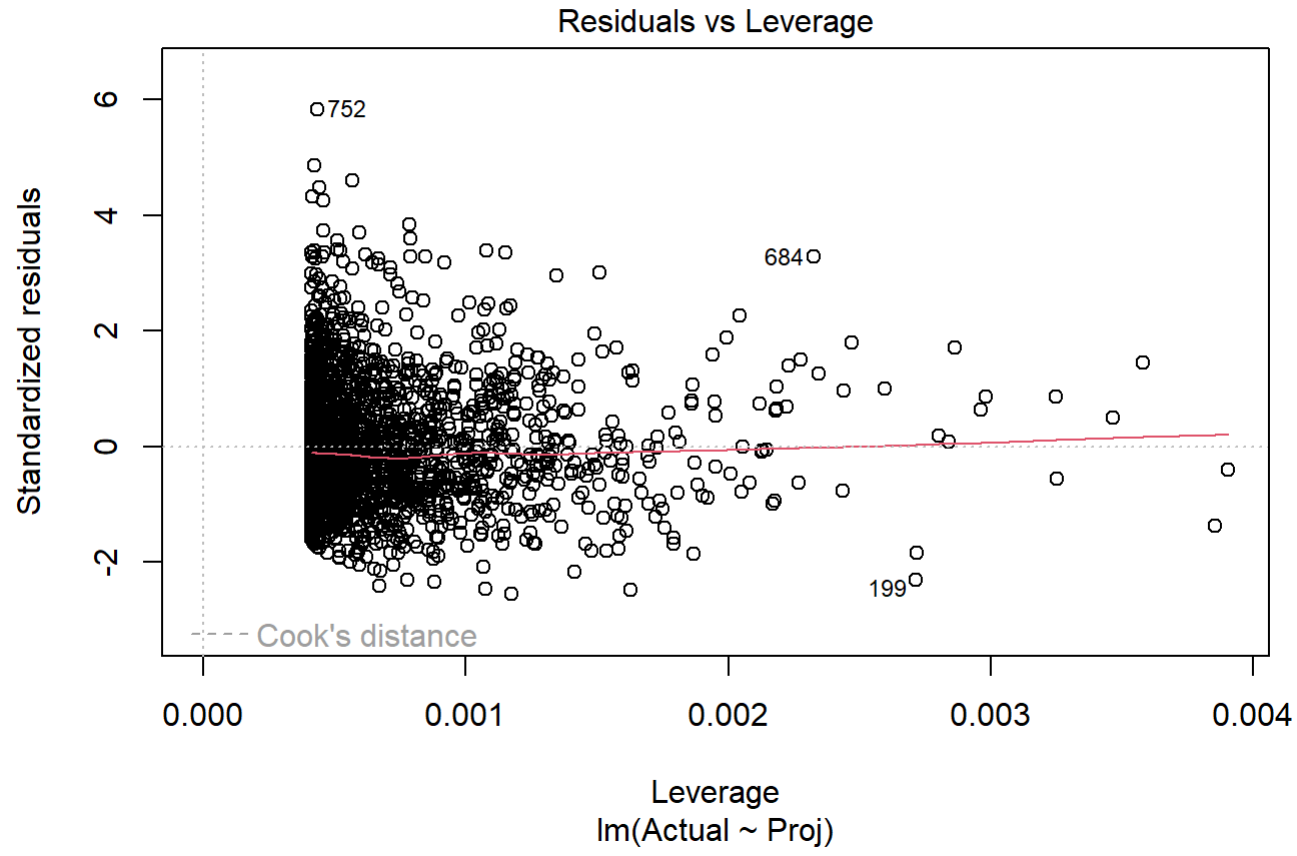
```
##
## Call:
## lm(formula = Actual ~ Proj, data = ESPN_Proj)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.983  -5.074  -0.416   3.673  43.375
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.41616    0.34476   1.207    0.228
## Proj         0.95687    0.02655  36.043   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.439 on 2421 degrees of freedom
##   (154 observations deleted due to missingness)
## Multiple R-squared:  0.3492, Adjusted R-squared:  0.3489
## F-statistic:  1299 on 1 and 2421 DF,  p-value: < 2.2e-16
```

```
plot(Lm)
```


Residuals vs Fitted
752
49
740
Residuals
40
20
0
-20
0
5
10
15
20
25
Fitted values
lm(Actual ~ Proj)


Normal Q-Q
752
49
740
Standardized residuals
6
4
2
0
-2
-3
-2
-1
0
1
2
3
Theoretical Quantiles
lm(Actual ~ Proj)


Scale-Location
752
49
740
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
2.5
0
5
10
15
20
25
Fitted values
lm(Actual ~ Proj)


Residuals vs Leverage
752
684
199
Cook's distance
Standardized residuals
-2
0
2
4
6
0.000
0.001
0.002
0.003
0.004
Leverage
lm(Actual ~ Proj)

The code here is pretty simple - we write a linear model with the lm() function (the actual variable as our y variable and the projected variable as our x variable, separated by the ~). We then call the summary() and plot() function to see what's going on with our model.

Notice below how R gives us a nice summary of the results from the linear model that we fit around this data! Let's first look at the residuals and interpret what they mean. We have a 5 number summary at the top (min, median, max, quartiles). For this project, I like to look at the middle 50% of the data (between 1Q and 3Q). Here, since residuals are actual - expected, we can say that ESPN will typically "overproject" by 5.074 pts or "underproject" by 3.673 points (50% of the time). Now let's look at the coefficients. From the estimate column we see that on average, a 1 point increase in ESPN projections corresponds to a 0.95687 point increase in actual output. Pretty good right? The t-value is 36.043, and the p-value is incredibly low. This means ESPN has a "statistically significant," or useful predictor for actual fantasy output! Looking at the bottom piece, we see the headline R^2 value of 0.3492, in that ~35% of the variation (spread) in actual points can be explained by the variation in ESPN's projections.

If you look at the plots below from the plot() function, R gives us a residual plot, a normal probability plot, a standardized residual plot, and a residual v leverage plot. Notice how the residual plot has a bit of a pattern; the residuals seem to fan out a bit for larger values of x. This is further shown with the normal probability plot as it starts to curve upward at the 2nd quintile; the standardized residual plot also shows this as it bends upwards for larger values of x. The residual v leverage plot does not appear to show any strong issues, but overall, these plots indicate a few issues, which signal that our linear model might not be the best choice for this data.

Filter for >10 pts

Next, we will filter for players of interest. It seems like our evaluation of ESPN's model is benefiting from the fact that we include all values of actual points and projected points (for instance, they predict a 0 point performance very well). However, we are not very interested in projecting players to score under 10 points - we would never play them in our lineups! As such, we will use the mutate() function from library(tidyverse) to add a binary column representing players that score over 10 points in a week. We will call this column "Over10" and use an ifelse() function to create it in our dataframe.

```
# Filter for > 10 pts
library(tidyverse)
```

```
## Warning: package 'tibble' was built under R version 4.2.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✓ dplyr     1.1.0     ✓ readr     2.1.4
## ✓ forcats   1.0.0     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.1     ✓ tidyr     1.3.0
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the ]8;;http://conflicted.r-lib.org/conflicted package]8;; to force all conflicts t
o become errors
```

```
ESPN_Proj %>%
  mutate(Over10 = ifelse(Actual > 10, 1, 0)) -> ESPN_Proj
```

We will then spawn a new dataframe by filtering for players who have an Over10 value of 1.

```
ESPN_Proj10 <- filter(ESPN_Proj, Over10 == 1)
```

We chose this method to show you a pretty complex way to select players who scored over 10 points (using concepts from prior lectures). Other options that are easier to implement are below:

```
#ESPN_Proj2 <- subset(ESPN_Proj, Actual > 10)
#ESPN_Proj3 <- filter(ESPN_Proj, Actual > 10)
```

Filter for WRs >10 pts

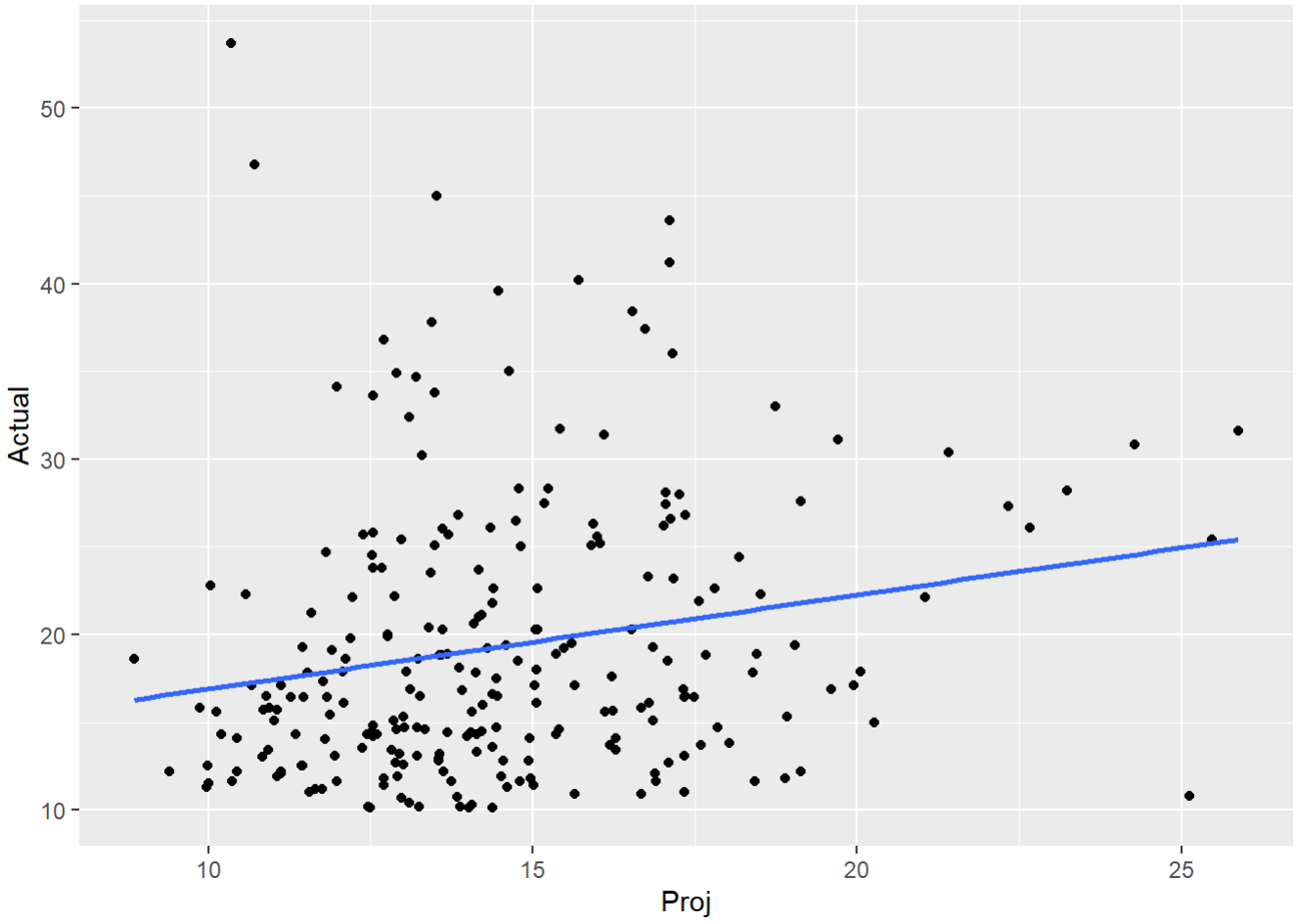
For our project, we will analyze ESPN's projections for WRs of interest (those who score over 10 points). We will use the filter() function to get our dataframe we can analyze:

```
# Filter for WRs > 10 pts
ESPN_WRs <- filter(ESPN_Proj10, Pos == "WR")
```

Again, we will visualize the data to analyze ESPN's projection performance for WRs of interest. We can reuse our code from our earlier visualization and fit a linear model with our new variables and dataframe. Code below:

```
# WR model for performance
ggplot(ESPN_WRs, aes(x=Proj,y=Actual))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
Lm <- lm(data = ESPN_WRs, Actual ~ Proj)
summary(Lm)
```

```
##
## Call:
## lm(formula = Actual ~ Proj, data = ESPN_WRs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -14.210  -5.294  -2.032   3.784  36.613
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.5344     2.4646   4.680 4.76e-06 ***
## Proj          0.5362     0.1662   3.226  0.00143 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.742 on 244 degrees of freedom
## Multiple R-squared:  0.04091,    Adjusted R-squared:  0.03698
## F-statistic: 10.41 on 1 and 244 DF,  p-value: 0.001427
```

To keep this document concise, we decided not to show the entire plot(Lm) output for this linear fit. However, feel free to run it if you'd like as it would provide more insight into our analysis. Looking at the summary values, ESPN remains about the same for the middle 50% of the data, either overprojecting by 5.294 points or underprojecting by 3.784 points. We can see the median move to the left, meaning ESPN is overprojecting more often for these receivers. With our estimate slope of 0.5362, the linear fit is more flat, but be careful, because our y axis starts at 10 now (we filtered out 0-10 point scorers but kept the projections the same). Our t-value decreased by a magnitude of 12 from our prior model(!), but the p-value is still statistically significant. Looking at our R^2 value, it decreases all the way to .04091 (from ~.35). The story of ESPN's performance is a little different here, right?

2019 Week 16 Analysis

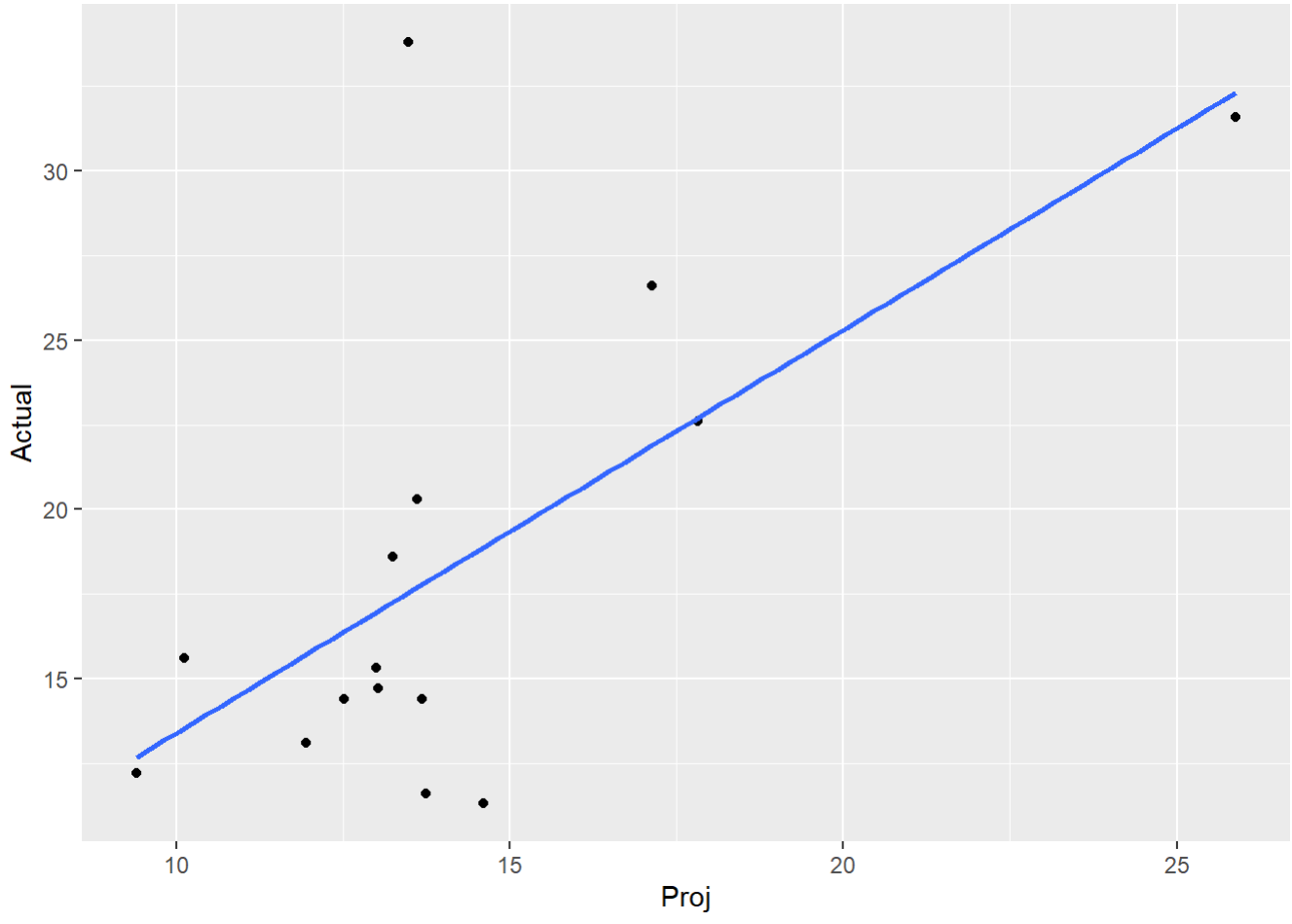
All of our prior analyses were conducted on data for the entire year, but let's zoom in to Week 16 of 2019. We can get this data simply by filtering our last dataframe:

```
# Filter for Wk 16
ESPN_WRs_Wk16 <- filter(ESPN_WRs, Week == 16)
```

We will now conduct our exploratory analysis again to see the big picture:

```
# Wk 16 model for WR performance
ggplot(ESPN_WRs_Wk16, aes(x=Proj,y=Actual))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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



```
Lm <- lm(data = ESPN_WRs_Wk16, Actual ~ Proj)
summary(Lm)
```

```
##
## Call:
## lm(formula = Actual ~ Proj, data = ESPN_WRs_Wk16)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5799 -2.4432 -0.7012  1.7104 16.2642
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.4709     5.8032   0.253  0.80388
## Proj          1.1915     0.3948   3.018  0.00989 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.731 on 13 degrees of freedom
## Multiple R-squared:  0.412,  Adjusted R-squared:  0.3668
## F-statistic: 9.109 on 1 and 13 DF,  p-value: 0.009888
```

Our sample size decreases to 15 WRs, so we will have to be a bit careful, but looking at the plot and our summary values, ESPN does pretty well here in Week 16. The middle 50% of projections either overproject by 2.4432 points or underproject by 1.7104 points, and the median is slightly below 0, which means a slight overprojection at the center of the data. We find that our t-value remains at 3 and the p-value remains statistically significant. Lastly, the R^2 value increases to .412! Not too bad.

Our projection

Now that we have a pretty good idea of ESPN's performance, we will try to build a basic multiple linear regression model to create our own projections for Week 16 of 2019. We will now load a dataset of yearly data where we will build our projections from, and filter for WRs:

```
# Grab season WR data
Yearly_Data <- read.csv("2019FantasyYearlyData.csv")
Yearly_Data <- filter(Yearly_Data, Yearly_Data$Pos == "WR")
```

Our dataset contains full-season statistics. While we can build projections off of these, we'd like to get more context by scaling these stats to weekly averages. We can do this using the mutate() function.

```
# Add weekly data
Yearly_Data %>%
  mutate(RecYdsWk = ReceivingYds / G) %>%
  mutate(TgtWk = Tgt / G) %>%
  mutate(PPW = FantasyPoints / G) %>%
  mutate(RecTDsWk = ReceivingTD / G) %>%
  mutate(RecWk = Rec / G) %>%
  mutate(FumWk = FumblesLost / G) -> Yearly_Data
```

Notice how that code runs as one block: this is by clever use of the ” %>% ” operator.

Now, how will we project a player’s fantasy output? The fantasy scoring equation for WRs is given below: Score = 1*Rec* + *0.1*Rec Yds + 6*Rec TD* -*2*Fumble

We will try to fit a linear model to “project” each of these variables, and then apply this equation on each expected value to achieve our overall score projection.

Receiving Yards

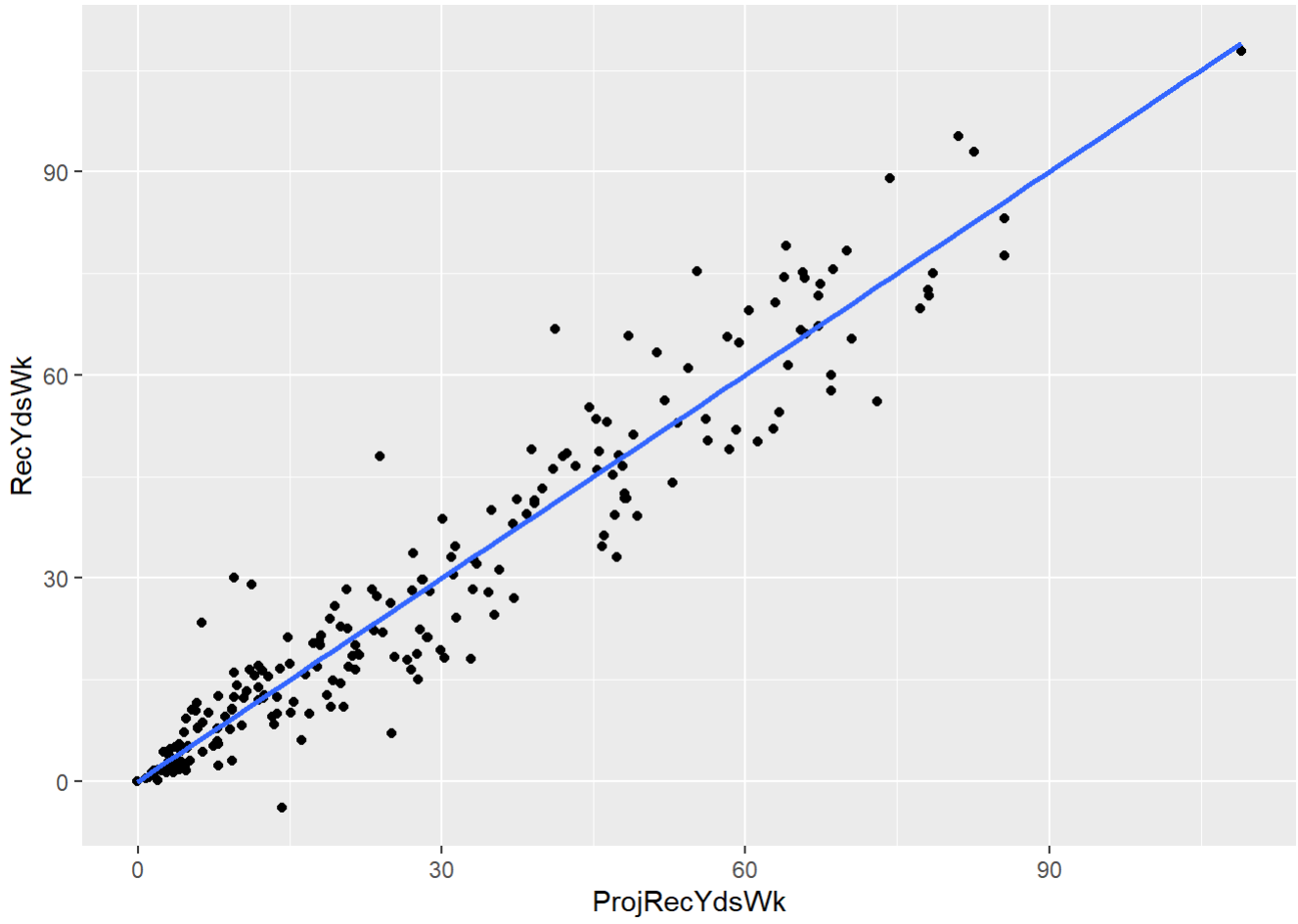
We will now start by fitting a linear model on receiving yards:

```
# Linear model for receiving yards
ProjRecYds <- lm(data = Yearly_Data, RecYdsWk ~ TgtWk+RecTDsWk+RecWk)

# Get projection
Yearly_Data$ProjRecYdsWk <- predict(ProjRecYds)

# Analyze projection performance
ggplot(Yearly_Data, aes(x=ProjRecYdsWk,y=RecYdsWk))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
summary(ProjRecYds)
```

```
##
## Call:
## lm(formula = RecYdsWk ~ TgtWk + RecTDsWk + RecWk, data = Yearly_Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -18.2472  -3.8393  -0.0515   3.2700  25.5848
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.1211     0.8161  -0.148   0.8821
## TgtWk         2.3731     0.8635   2.748   0.0065 **
## RecTDsWk     25.1320     3.6104   6.961 4.12e-11 ***
## RecWk         7.2490     1.3312   5.445 1.42e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.086 on 213 degrees of freedom
## Multiple R-squared:  0.9179, Adjusted R-squared:  0.9168
## F-statistic: 794.3 on 3 and 213 DF,  p-value: < 2.2e-16
```

Notice how the syntax for this model is a little different than the past. This is because we are using a multiple linear regression model, where we try to predict a WR's receiving yards per week from a combination of their weekly targets, receiving TDs, and receptions. We create a new column in our dataframe called "ProjRecWk" to apply the linear model equation so we can build individualized player projections later from the predict() function.

Look at the visual and the summary stats on this model - it's really strong! We can predict a player's average receiving yards per week really well when given these variables. Our residuals indicate that we generally predict within ~3-4 receiving yards, all of our variables are statistically significant, and our R^2 value is through the roof. Done, right?

Well, there's actually a statistical check we have to do in order to make sure we aren't cheating. This check we will do is for multicollinearity. Multicollinearity is present when variables in our regression model are highly correlated with each other, which violates the independence assumption and artificially inflates our statistical significance. Let's check if this is the case by calling the vif() function.

```
# Multicollinearity check
library(car)
```

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

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

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

```
## The following object is masked from 'package:purrr':
##
##     some
```

```
vif(ProjRecYds)
```

```
##     TgtWk  RecTDsWk     RecWk
## 24.537232  2.203048 25.052111
```

A vif value >10 is something that we flag by convention, and here, the ~25 values for targets and receptions indicate that we have to get rid of one of these variables. This makes intuitive sense as the variables are strongly correlated. As such, you can make an argument for getting rid of either variable, but we will pick targets over receptions to keep in our model. We will now re-run the projection below:

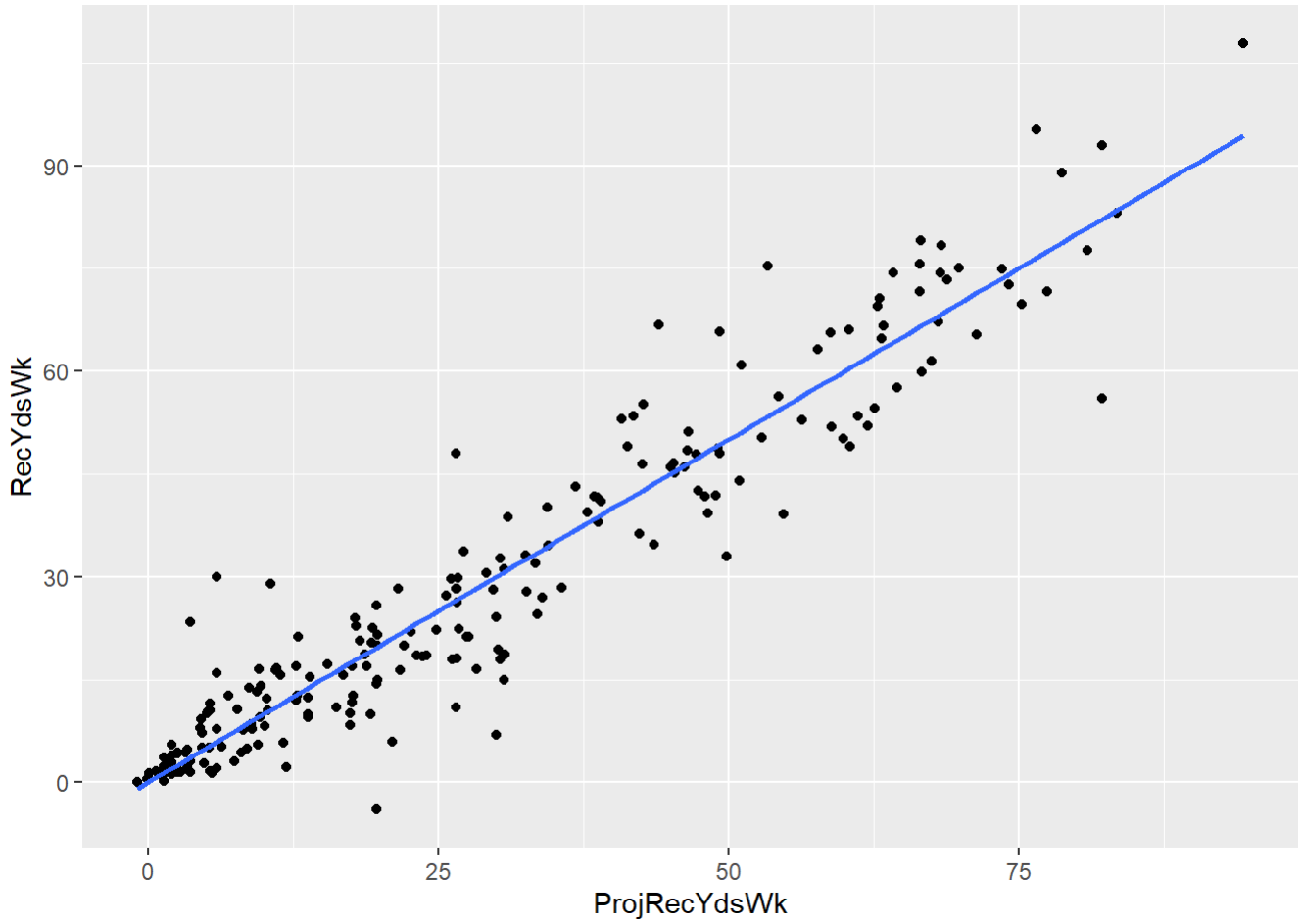
Receiving Yards Correction

```
# Rerun projections without receptions
# Linear model for receiving yards
ProjRecYds <- lm(data = Yearly_Data, RecYdsWk ~ TgtWk+RecTDsWk)

# Get projection
Yearly_Data$ProjRecYdsWk <- predict(ProjRecYds)

# Analyze projection
ggplot(Yearly_Data, aes(x=ProjRecYdsWk,y=RecYdsWk))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
summary(ProjRecYds)
```

```
##
## Call:
## lm(formula = RecYdsWk ~ TgtWk + RecTDsWk, data = Yearly_Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.2199  -4.4802   0.0065   3.8630  24.0453
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.9098     0.8552  -1.064    0.289
## TgtWk         6.8645     0.2721  25.228  < 2e-16 ***
## RecTDsWk     28.2134     3.7970   7.430 2.55e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.546 on 214 degrees of freedom
## Multiple R-squared:  0.9065, Adjusted R-squared:  0.9057
## F-statistic:  1038 on 2 and 214 DF,  p-value: < 2.2e-16
```

```
# Check for multicollinearity
vif(ProjRecYds)
```

```
##    TgtWk RecTDsWk
## 2.148931 2.148931
```

We can see that the model still performs well, and our vif values are within convention. We can now continue to project receptions (code below is finalized after checking vif values… feel free to include whichever variables you'd like in your model, but make sure to check the vif before continuing).

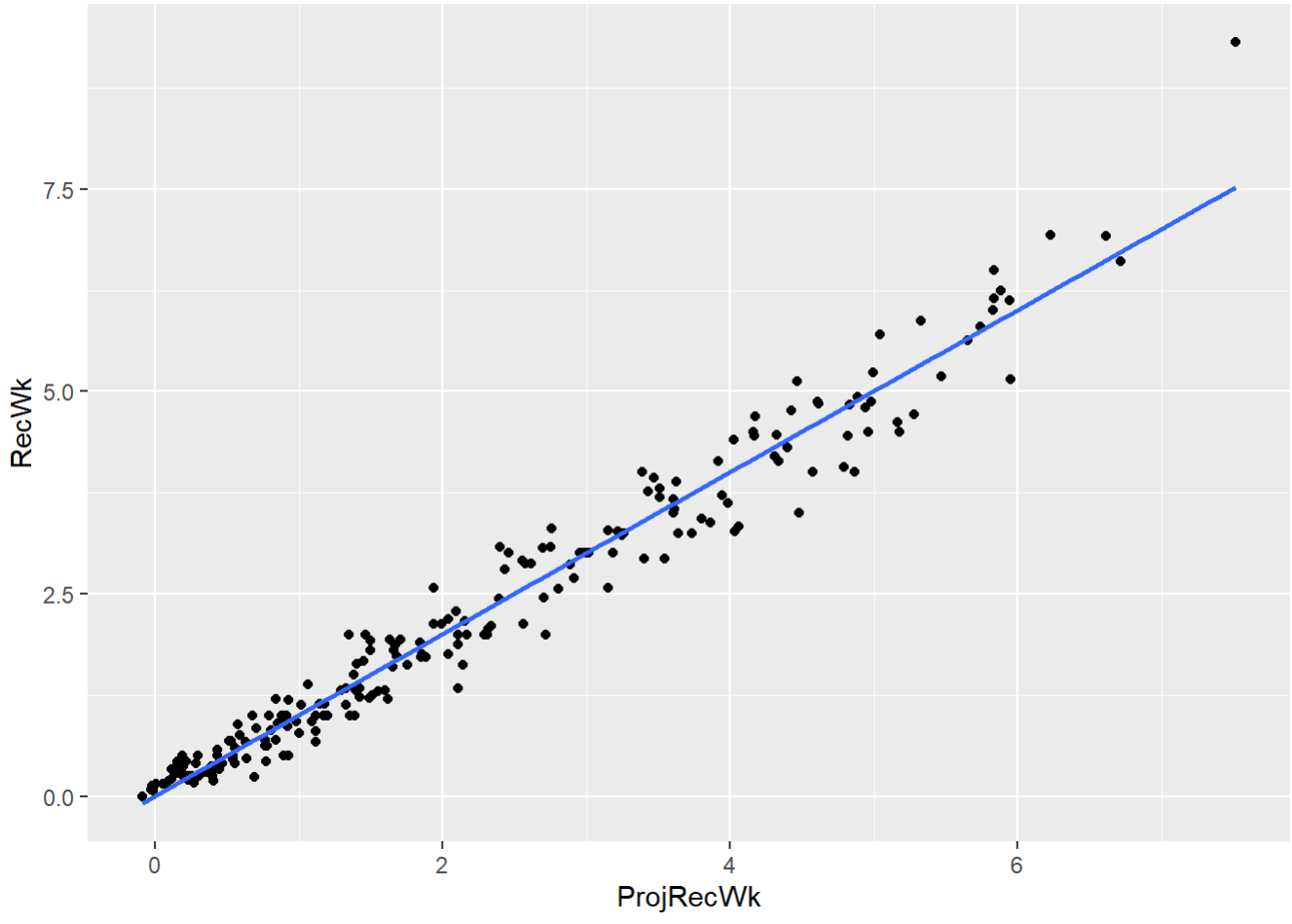
Receptions

```
# Linear model for receptions
ProjRec <- lm(data = Yearly_Data, RecWk ~ RecYdsWk+TgtWk)

# Get projection
Yearly_Data$ProjRecWk <- predict(ProjRec)

# Analyze projection
ggplot(Yearly_Data, aes(x=ProjRecWk,y=RecWk))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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



```
summary(ProjRec)
```

```
##
## Call:
## lm(formula = RecWk ~ RecYdsWk + TgtWk, data = Yearly_Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.98006 -0.18721  0.01326  0.20501  1.79677
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.092992   0.038707  -2.402   0.0171 *
## RecYdsWk     0.016491   0.002754   5.988 8.89e-09 ***
## TgtWk        0.504287   0.024461  20.616  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.341 on 214 degrees of freedom
## Multiple R-squared:  0.9649, Adjusted R-squared:  0.9646
## F-statistic:  2946 on 2 and 214 DF,  p-value: < 2.2e-16
```

```
# Multicollinearity check
vif(ProjRec)
```

```
## RecYdsWk    TgtWk
## 8.504026 8.504026
```

We are able to project a wideout's average receptions per week very well given their average yards and targets per week. We should be a little cautious about this one though… reception yards and targets seem to overfit the data. But for now, we will bypass this (this model won't be very useful) and look at the remaining variable, receiving TDs:

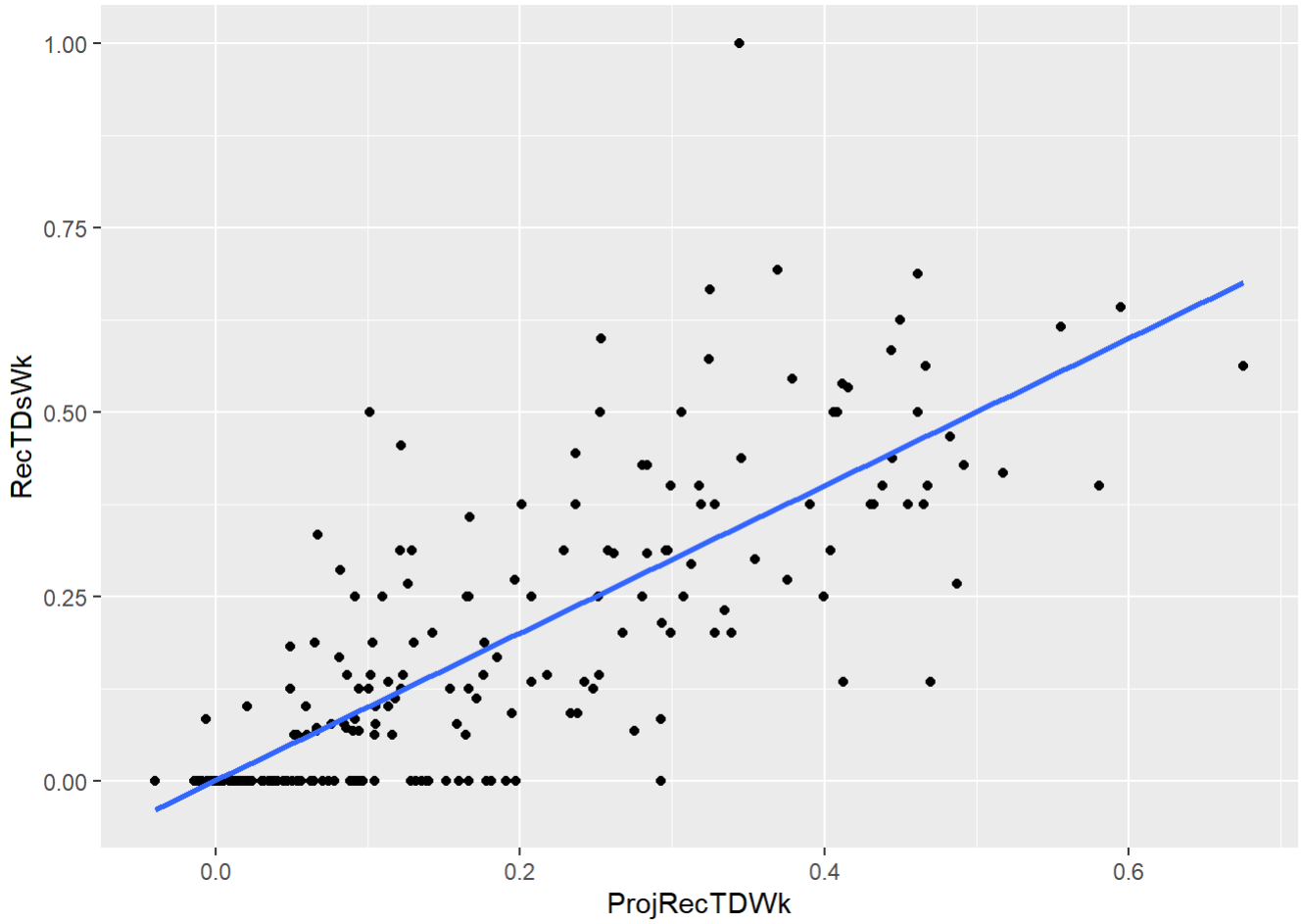
Receiving TDs

```
# Linear model for receiving touchdowns
ProjRecTD <- lm(data = Yearly_Data, RecTDsWk ~ RecYdsWk)

# Get projection
Yearly_Data$ProjRecTDWk <- predict(ProjRecTD)

# Analyze projection
ggplot(Yearly_Data, aes(x=ProjRecTDWk,y=RecTDsWk))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
summary(ProjRecTD)
```

```
##
## Call:
## lm(formula = RecTDsWk ~ RecYdsWk, data = Yearly_Data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33616 -0.06384 -0.00934  0.04576  0.65589
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.0141204  0.0127649  -1.106     0.27
## RecYdsWk     0.0063970  0.0003354  19.073   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1211 on 215 degrees of freedom
## Multiple R-squared:  0.6285, Adjusted R-squared:  0.6268
## F-statistic: 363.8 on 1 and 215 DF,  p-value: < 2.2e-16
```

Final projection

Notice how we left out fumbles in our projection for WR weekly stats. This is a decision made with the assumption that fumbles are a rare, highly variable random event; we see no use in projecting a player's fumble total for each week. You can check this empirically, but to save time, we will bypass fumbles and just impute the average for each player. Therefore, we now have every variable needed to write our projection equation (for PPR scoring being our scoring method):

```
# Generate final projection
Yearly_Data$Our_Proj_PPW <- Yearly_Data$ProjRecWk + 6*Yearly_Data$ProjRecTDWk + 0.1*Yearly_Data
$ProjRecYdsWk - 2*Yearly_Data$FumWk
```

Now, to compare this model to ESPN's model, we are going to use the merge() function to combine both dataframes of interest. In the ESPN_WRs_Wk16 dataframe, we have the WRs of interest already filtered (>10 pts and results for Week 16 rather than yearly), and in the Yearly_Data dataframe, we have our projections for each player. We will now combine these dataframes using the code below:

```
# Combine both dataframes - join on player
Merge <- merge(ESPN_WRs_Wk16, Yearly_Data, by = "Player")
```

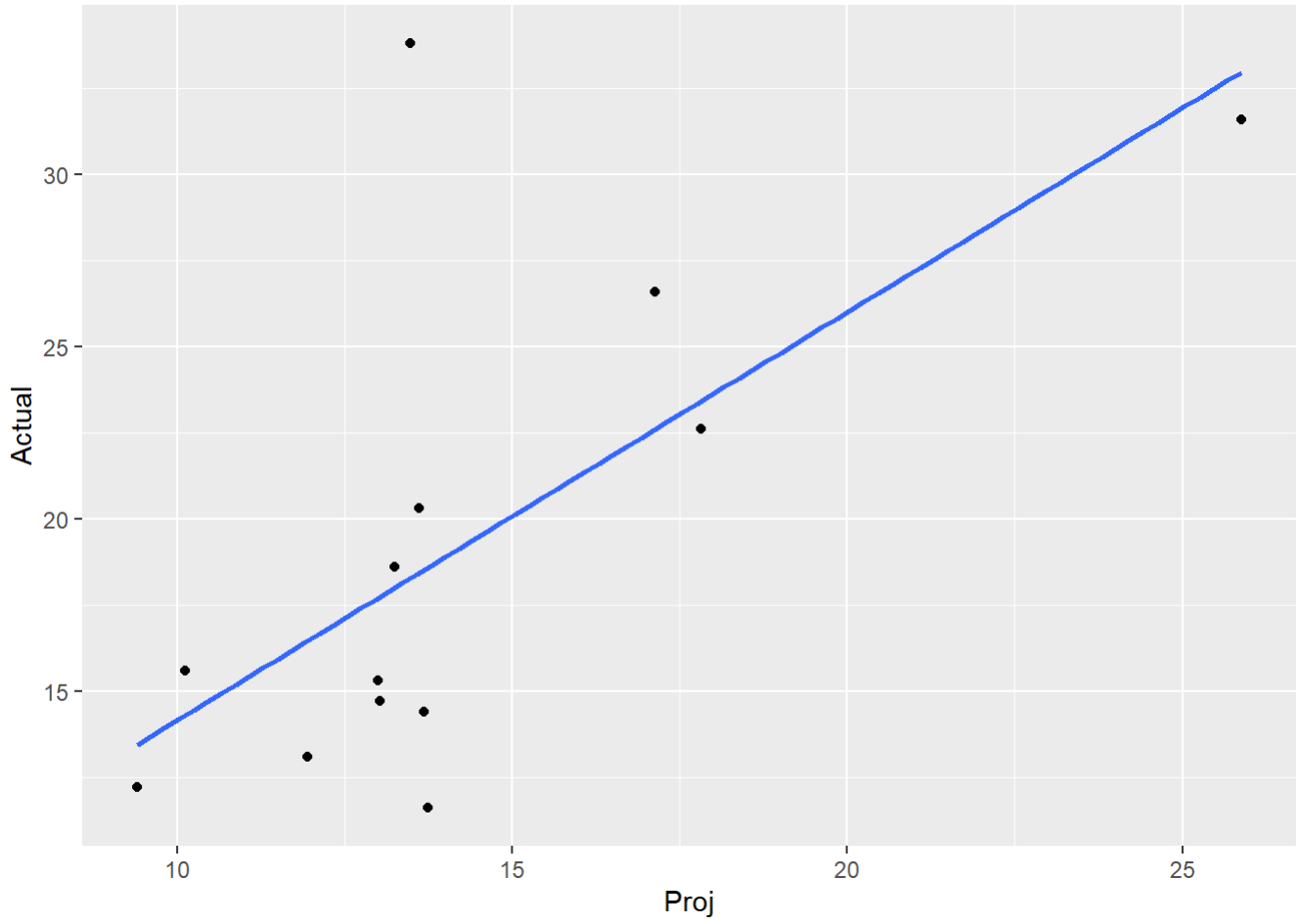
Notice how we merged by the column "Player." The merge function matches the two dataframes by their shared column titled "Player." It is imperative that these columns have the same name, or else the merge() function will return an error message. We can now compare the performance of our model v ESPN's!

ESPN Model

Here is a reminder of the performance of the ESPN model:

```
# ESPN model performance
ggplot(Merge, aes(x=Proj,y=Actual))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
Lm <- lm(data = Merge, Actual ~ Proj)
summary(Lm)
```

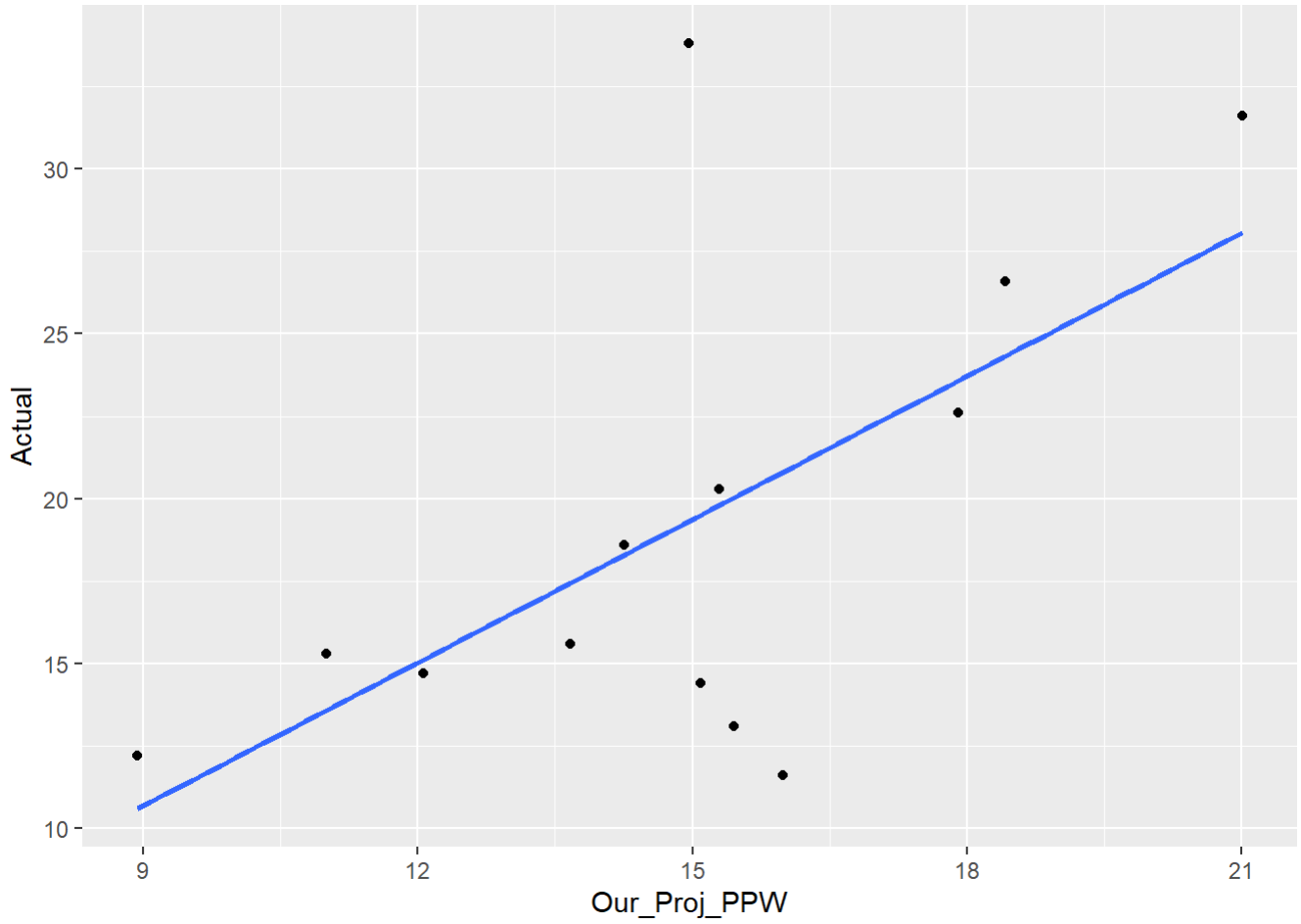
```
##
## Call:
## lm(formula = Actual ~ Proj, data = Merge)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -6.992 -3.029 -1.242  1.305 15.524
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.2969     5.8898    0.39   0.7040
## Proj          1.1852     0.3963    2.99   0.0123 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.71 on 11 degrees of freedom
## Multiple R-squared:  0.4484, Adjusted R-squared:  0.3983
## F-statistic: 8.943 on 1 and 11 DF,  p-value: 0.01229
```

Our model

And now our model!

```
# Our model performance
ggplot(Merge, aes(x=Our_Proj_PPW,y=Actual))+geom_point()+geom_smooth(method=lm,se=FALSE)
```

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




```
Lm <- lm(data = Merge, Actual ~ Our_Proj_PPW)
summary(Lm)
```

```
##
## Call:
## lm(formula = Actual ~ Our_Proj_PPW, data = Merge)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -9.204 -1.834  0.300  1.722 14.480
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.3411     8.3070  -0.282   0.7833
## Our_Proj_PPW   1.4475     0.5453   2.654   0.0224 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.003 on 11 degrees of freedom
## Multiple R-squared:  0.3904, Adjusted R-squared:  0.335
## F-statistic: 7.046 on 1 and 11 DF,  p-value: 0.02241
```

Summary

Comparing the summary stats of the two models, ESPN's model is a slightly more accurate and statistically significant predictor for WRs based on the Week 16 dataset. However, looking at the middle 50% of the data, our projections typically fall within 2 points of the actual result, while ESPN's usually overproject by 3 or underproject by 1. Comparing the mins and maxes, ESPN better captures the lower point values than our model, while we do a better job with the higher point values. More analysis across multiple weeks (with more sample) could be conducted to dig deeper!

A Cautionary Tale and Considerations

Notice how in our yearly dataset we actually had a full year of data in which we built our projection system; therefore, we did factor in actual Week 16 and 17 data in our analysis. At the time of the Week 16 projection, the ESPN model did not have this data (our case would be called a data leakage). Furthermore, while it is impressive that we could predict the average statistics for each WR, predicting with an R^2 value of close to 1 is not really useful in a modeling sense. We are taught that a high R^2 value is good, but if it is close to perfect, why wouldn't we just extrapolate the average values rather than use the model? Therefore, it is important to balance model accuracy with usefulness, because if you overfit the data, your model will not fit well on new data. Additionally, ESPN surely doesn't use linear models for this data, and as evidenced by our residual plots, a linear model is likely not the best fit for this data. More methods would need to be explored. Additionally, there are a lot of variables we did not account for, like pace of play, weather, injuries, quarterback play, etc. Surely inclusion of this data would help make our model more robust (and may make us less prone to outliers). There are also many different criteria to select variables for your model (not just vif), which you can research. However, I hope you enjoyed this exercise and remain curious at looking at real-world datasets to see what we can discover and learn :)