Linear Regression Course

Contents

Introduction

In this course, we discuss the statistical concepts underlying the quantification of relationships between variables in multivariate datasets using linear regression. We will also examine confounding, where extraneous variables affect the relationship between other variables, creating spurious associations.

Regression

Case Study: Heredity

We'll start off with the dataset used by the inventor of the regression method, Francis Galton. He was trying to understand to what degree the heights of fathers predicted the heights of their sons – i.e. to what extend height is hereditary. The dataset is available in the `HistData` package.

```
data("GaltonFamilies")
galton_heights <- GaltonFamilies %>%
  filter(childNum == 1 & gender == "male") %>%
  select(father, childHeight) %>%
  rename(son = childHeight)
```

We'll come back to this dataset to demonstrate the application of regression methods.

Correlation and the Regression Line

Two variables are said to be correlated if the change in one coincides with a commensurate change in the other, either directly or inversely. The correlation coefficient describes how much variability in one variable is

explained by the other.

The correlation coefficient is defined for a list of pairs $(x_1, y_1), \ldots, (x_n, y_n)$ as:

$$\rho = \frac{1}{n} \sum_{i=1}^{n} \left(\frac{x_i - \mu_x}{\sigma_x} \right) \left(\frac{y_i - \mu_y}{\sigma_y} \right)$$

with μ_x, μ_y the averages of $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ respectively and σ_x, σ_y the standard deviations.

Note that each of the terms being summed above represent the number of standard deviations that the *ith* x or y is from the mean. If x and y are unrelated, then the product will be positive as often as negative, and the sum will be about zero. If the variables are related, then the relation will be more often positive or negative, and ρ (Greek for r) will be larger in magnitude.

The correlation coefficient, ρ, is always between -1 and 1.

```
galton_heights %>% summarize(r = cor(father, son))
```

```
##           r
## 1 0.5007248
```

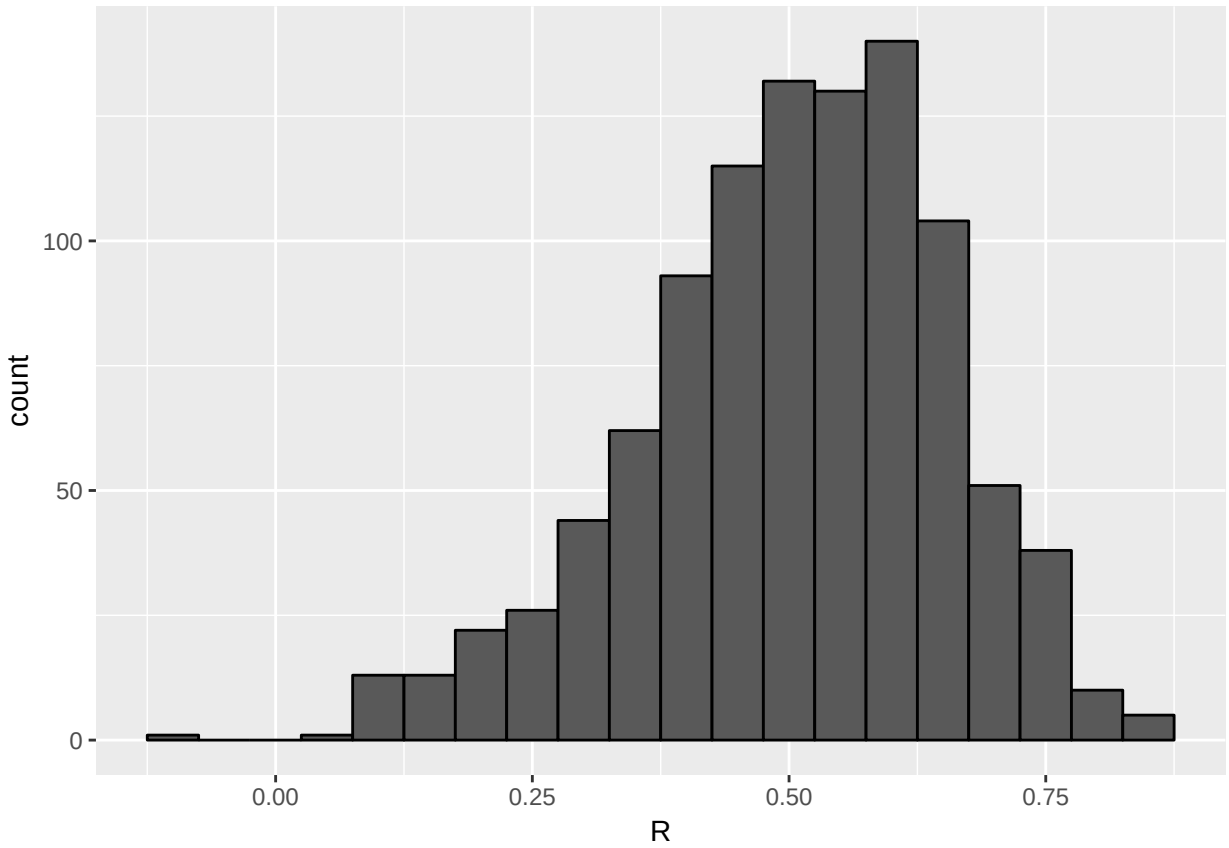
As you can see, the correlation between fathers' heights and that of their sons is about 0.2.

Sample Correlation is a Random Variable

We generally use the sample correlation as an estimate of the population correlation. The correlation coefficient between random samples of a population is a random variable, which we can illustrate with the following example.

Assume that the 26 pairs of fathers and sons represents our entire population, and we sample 25 of them. Let's simulate how the correlation would be distributed.

```
#set number of replications
B <- 1000
#set sample size
N <- 25
#compute correlation coefficient for each sample
R <- replicate(B, {
sample_n(galton_heights, N, replace = TRUE) %>% summarize(r=cor(father, son)) %>% .$r
})
#plot histogram
data.frame(R) %>% ggplot(aes(R)) + geom_histogram(binwidth = 0.05, color = "black")
```



The mean of the sample correlation coefficients is the mean of the population:

```
mean(R)
```

```
## [1] 0.503084
```

And has a relatively high standard error:

```
sd(R)
```

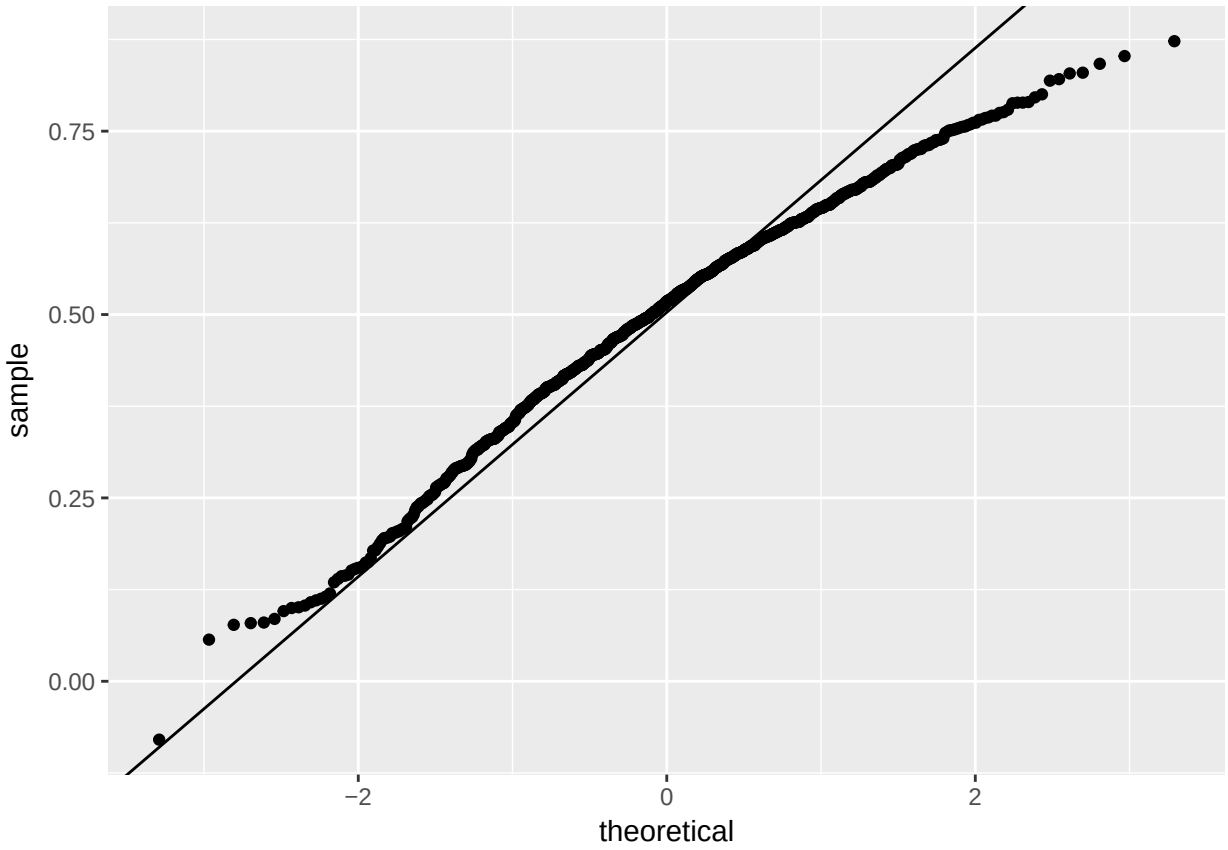
```
## [1] 0.1483063
```

Since R is a random variable, the CLT applies. With a large enough N, R is normally distributed with expected value ρ. The theoretical standard deviation of the (sample) correlation coefficient, r, is:

$$SD(r) = \sqrt{\frac{1-r^2}{N-2}}$$

Plotting the sample coefficients above in a qq-plot indicates that 25 samples is not enough to reliably distribute normally. (Note that in a qq-plot the intercept of the normal line is at the mean and the slope is the standard deviation.)

```
data.frame(R) %>%
  ggplot(aes(sample=R)) +
  stat_qq() +
  geom_abline(intercept = mean(R),
              slope = sqrt((1-mean(R)^2)/(N-2)))
```

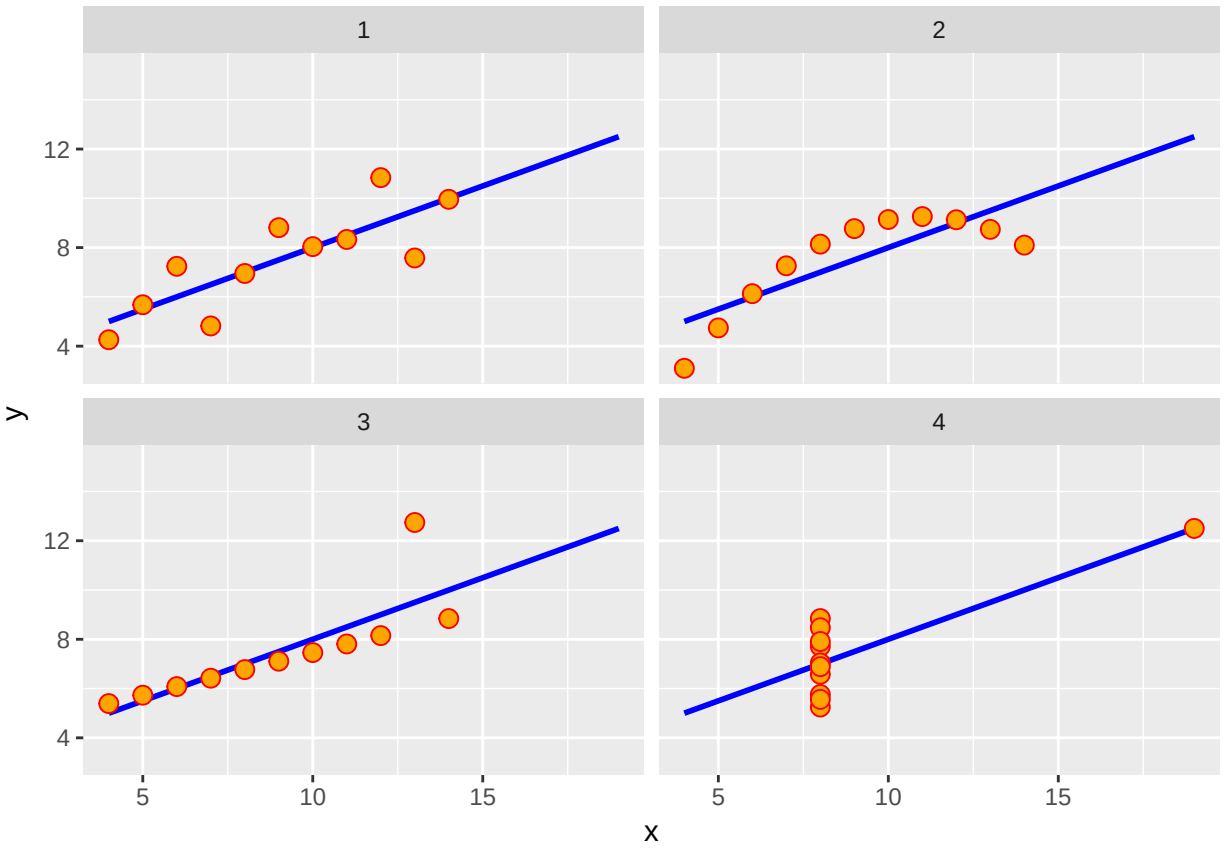


Increasing N would make the above plot converge to the normal line.

Using Correlation Appropriately

Correlation coefficients can be misleading. Check out these canonical examples, known as Anscombe's quartet, which illustrate this point. All of the following plots have correlation coefficient $r = 0.82$.

```
anscombe %>% mutate(row = seq_len(n())) %>%
  gather(name, value, -row) %>%
  separate(name, c("axis", "group"), sep=1) %>%
  spread(axis, value) %>% select(-row) %>%
  ggplot(aes(x,y)) +
  facet_wrap(~group)  +
  geom_smooth(method="lm", fill=NA, fullrange=TRUE, color="blue") +
  geom_point(bg="orange",color="red",cex=3,pch=21)
```



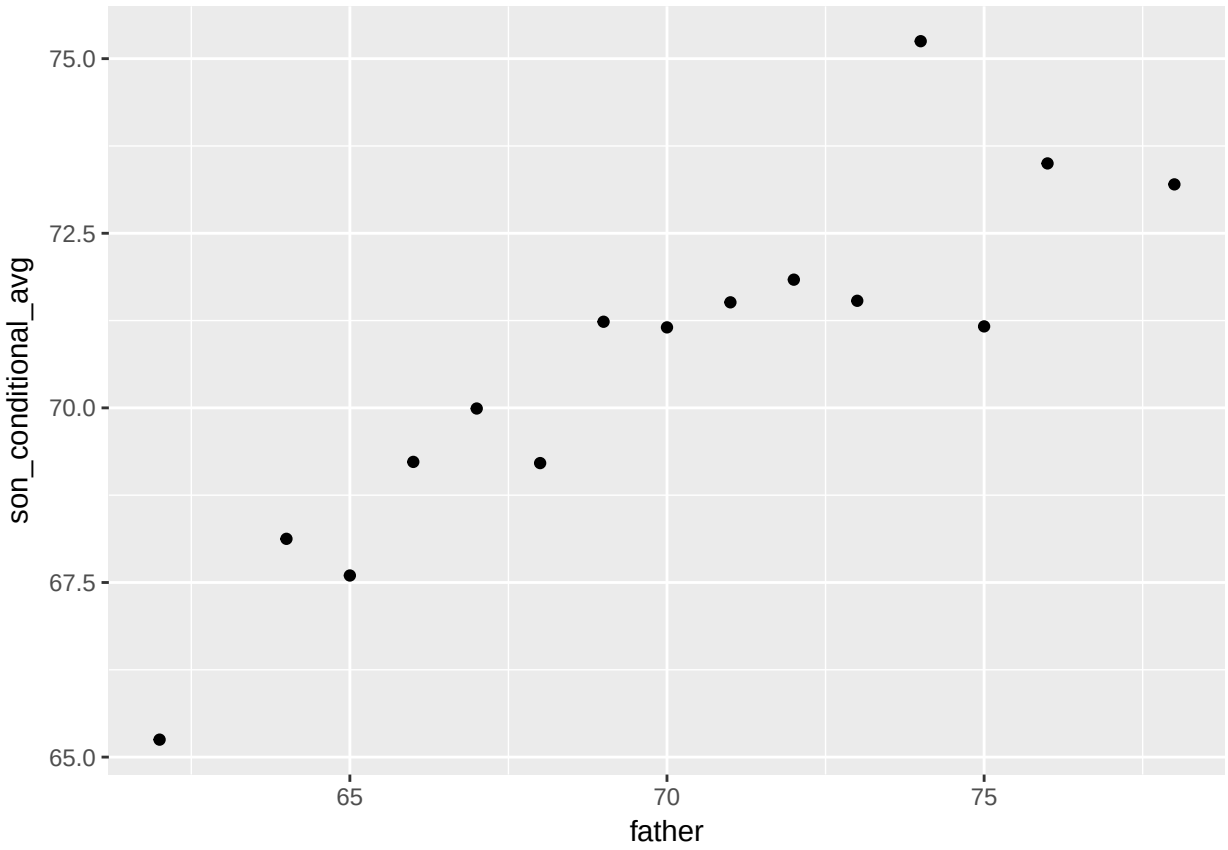
Correlation is only meaningful in a particular context. To see how and when correlation is useful, let's return to predicting the sons' heights based on their fathers'.

Prediction based on Stratification

If we wanted to predict the height of one of the sons given that his father's height is 72 inches, would we guess the average of all of the sons heights? Probably not. Ideally, we'd find out the heights of the sons of many fathers who were 72 inches tall and guess the average of their sons' heights. That sample would be normally distributed, and is known as a *conditional distribution*.

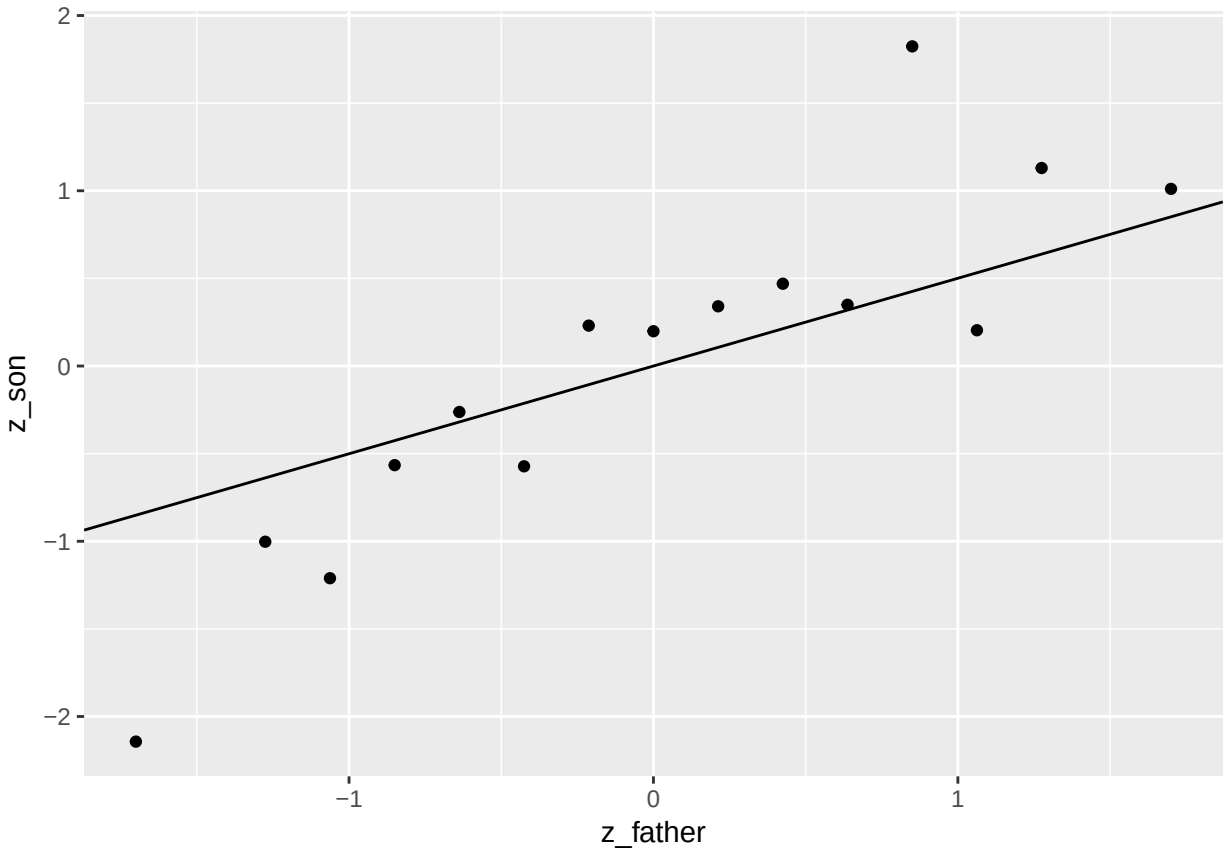
If we stratify the sons' heights based on the fathers' heights (rounding to the nearest inch), we see that the average of sons' heights increases as the father's height increases.

```
galton_heights %>% mutate(father = round(father)) %>% group_by(father) %>% summarize(son_conditional_av
```



Taking into account that these averages are random variables with standard errors, we can see that they follow a straight line once scaled.

```
r <- galton_heights %>% summarize(r = cor(father, son)) %>% .$r
galton_heights %>%
mutate(father = round(father)) %>%
group_by(father) %>%
summarize(son = mean(son)) %>%
mutate(z_father = scale(father), z_son = scale(son)) %>% ggplot(aes(z_father, z_son)) +
geom_point() +
geom_abline(intercept = 0, slope = r)
```



The line these averages follow is known as the *regression line*.

The Regression Line

If we are predicting a random variable Y knowing the value of another $X = x$ using a regression line, then we predict that for every standard deviation, σ_X, that x increases above the average mu_X, Y increase ρ standard deviations σ_Y above the average μ_Y with ρ the correlation between X and Y. The formula for the regression is therefore:

$$\left(\frac{Y - \mu_Y}{\sigma_Y}\right) = \rho\left(\frac{x - \mu_X}{\sigma_X}\right)$$

Rewritten into a $y = mx + b$ form:

$$Y = \mu_Y + \rho\left(\frac{x - \mu_X}{\sigma_X}\right)\sigma_Y$$

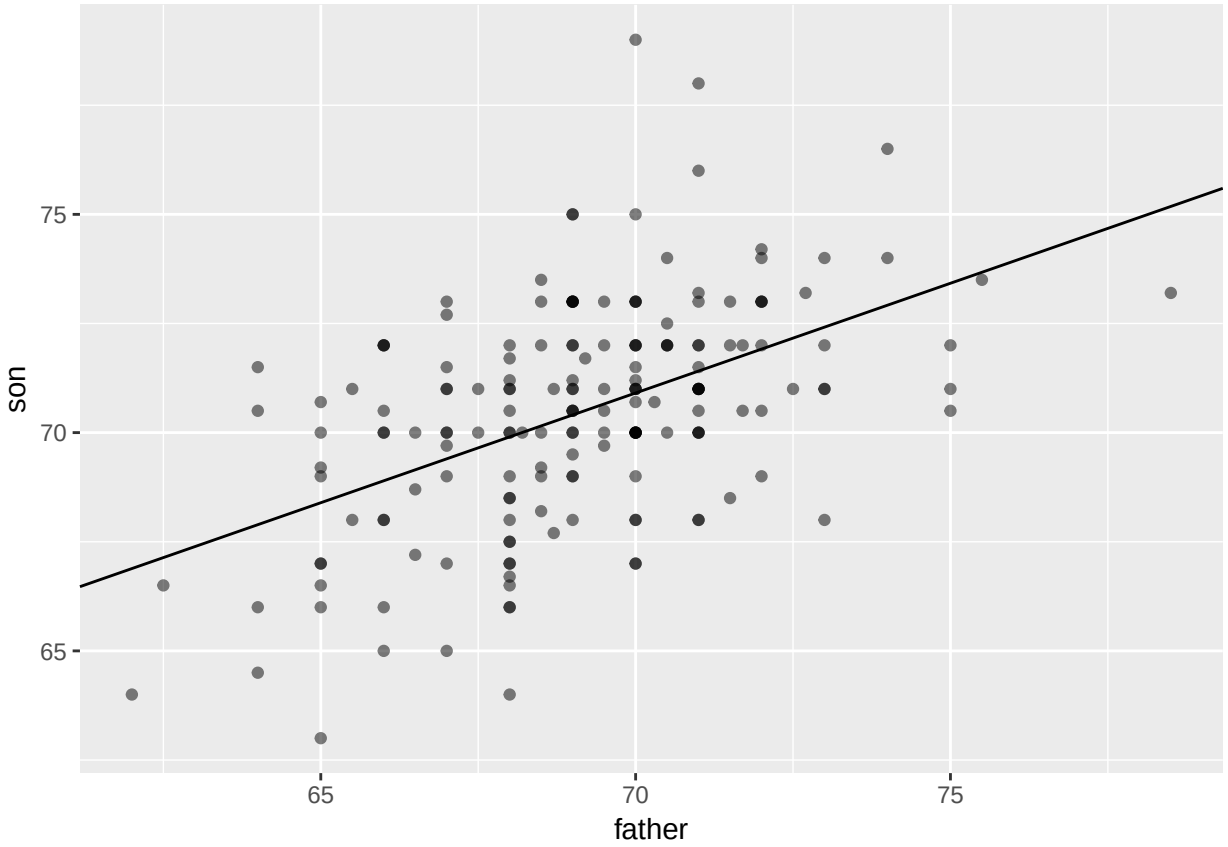
If there is a perfect correlation, we predict that an increase of 1 SD in x will be matched with an incrase of 1 SD in Y. If there is no correlation, ρ is 0 and we just use the mean to predict Y.

In general regression lines are constructed as follows:

$$y = b + mx \text{ with slope } m = \rho\frac{\sigma_y}{\sigma_x} \text{ and intercept } b = \mu_y - m\mu_x$$

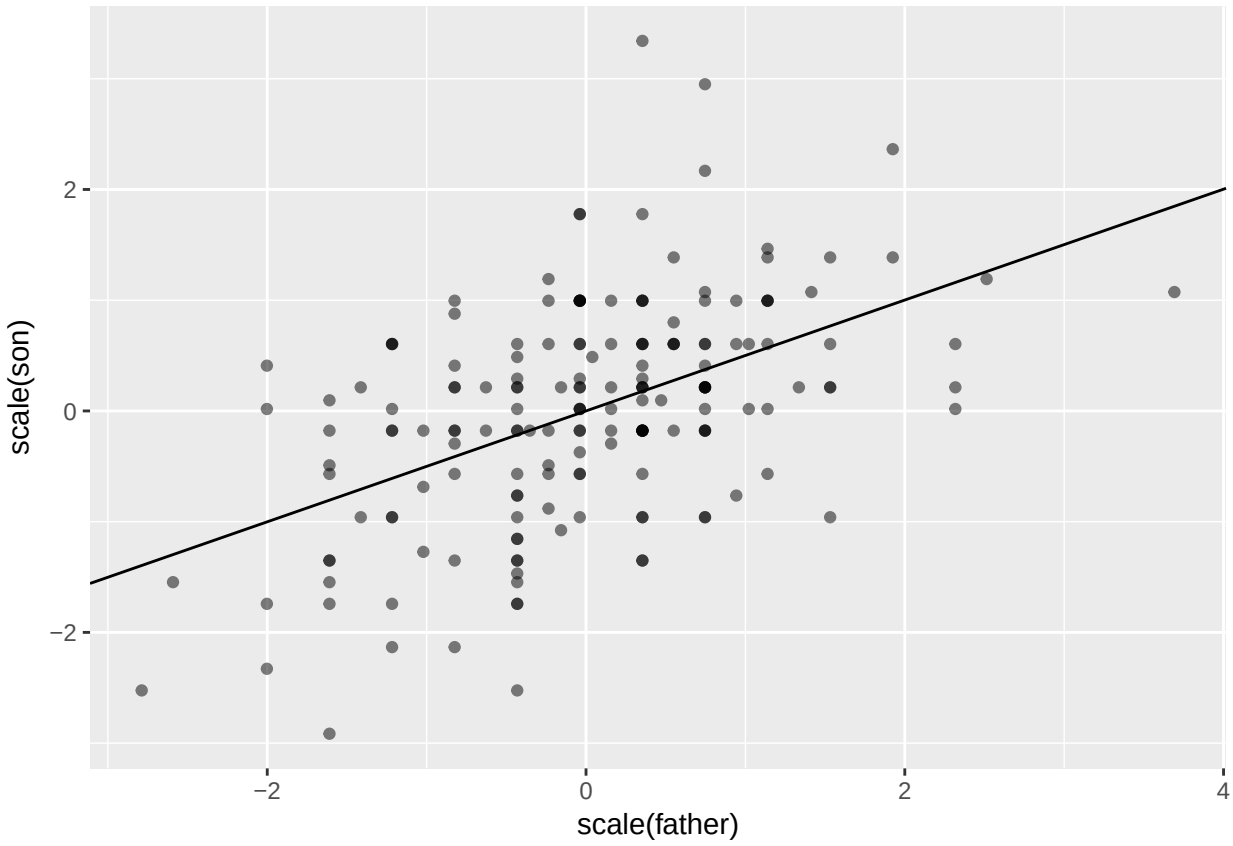
You can add the regression line to the original data as follows:

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m<- r*s_y/s_x #plug into the above equation for slope
b <- mu_y - m*mu_x #plug into the above equation for intercept
galton_heights %>%
ggplot(aes(father, son)) + geom_point(alpha = 0.5) + geom_abline(intercept = b, slope = m )
```



A simpler method is to first standardize the two variables, making the intercept 0 and the slope r.

```
galton_heights %>%
  ggplot(aes(scale(father), scale(son))) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = 0, slope = r)
```



The Regression Line Improves Precision

Let's compare the stratification and regression approaches:

1. Round father's inches to closest inch, stratify, and take the average
2. Compute the regression line and use it to predict

```
B <- 1000
N <- 50

set.seed(1)
conditional_avg <- replicate(B, {
  dat <- sample_n(galton_heights, N, replace = TRUE)
  dat %>% filter(round(father) == 72) %>%
    summarize(avg = mean(son)) %>%
    .$avg
  })

regression_prediction <- replicate(B, {
  dat <- sample_n(galton_heights, N, replace = TRUE)
  mu_x <- mean(dat$father)
  mu_y <- mean(dat$son)
  s_x <- sd(dat$father)
  s_y <- sd(dat$son)
  r <- cor(dat$father, dat$son)

  mu_y + r*(72 - mu_x)/s_x*s_y
```

```
})
```

The mean of each method's output is about the same, but the regression method produces a lower standard error, and is thus more precise.

```
data.frame(Mean_Conditional_Average = mean(conditional_avg, na.rm = TRUE),
           Mean_Regression = mean(regression_prediction))
```

```
##   Mean_Conditional_Average Mean_Regression
## 1                 71.87454        71.90665
```

```
data.frame(SD_Conditional_Average = sd(conditional_avg, na.rm = TRUE),
      SD_Regression = sd(regression_prediction))
```

```
##   SD_Conditional_Average SD_Regression
## 1              0.9713105     0.4706101
```

The regression line is more precise because it uses all of the data for every prediction, rather than a small subset. However, it is not always appropriate, since two variables may have a relationship that is nonlinear.

Galton showed that heights are in fact linearly correlated, and that the correlation coefficient is thus an appropriate measure of their relationship.

The Bivariate Normal Distribution

The *bivariate normal distribution* describes normal behavior between two variables: if X and Y are normally distributed random variables, and for any stratum of X, say $X = x$, Y is approximately normal in that stratum, then the pair is approximately bivariate normal.

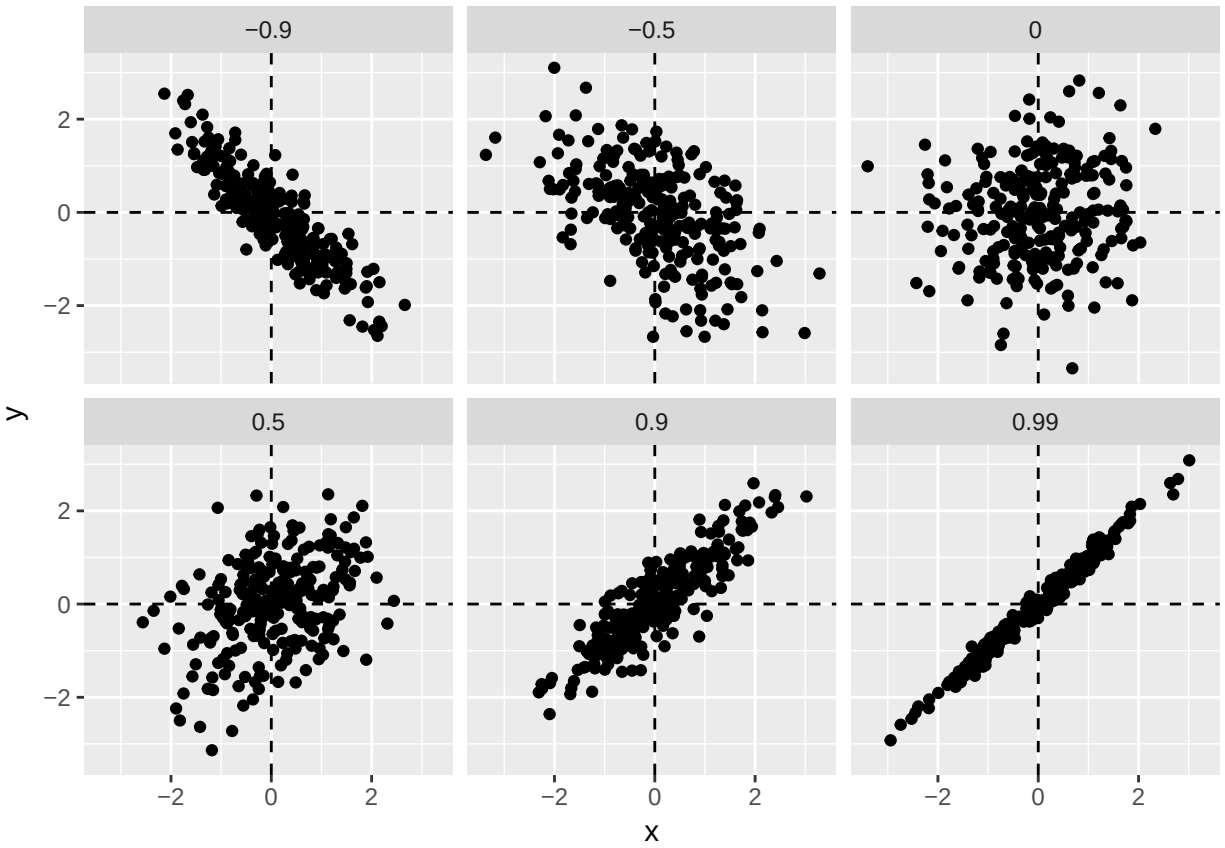
When we fix $X = x$ in this way, we then refer to the resulting distribution of the Ys in the strata as the *conditional distribution* of Y given $X = x$. We can write it using mathematical notation like this:

$$\text{conditional distribution: } f_{Y|X=x}$$

$$\text{conditional expected value: } E(Y \mid X = x)$$

In graphical form, the correlation coefficient corresponds to how slim the oval is:

```
n <- 250
cors <- c(-0.9,-0.5,0,0.5,0.9,0.99)
sim_data <- lapply(cors,function(r) MASS::mvrnorm(n,c(0,0), matrix(c(1,r,r,1),2,2)))
sim_data <- Reduce(rbind, sim_data)
sim_data <- cbind( rep(cors, each=n), sim_data)
colnames(sim_data) <- c("r","x","y")
as.data.frame(sim_data) %>% ggplot(aes(x,y)) +
  facet_wrap(~r) + geom_point() +
  geom_vline(xintercept = 0,lty=2) +
  geom_hline(yintercept = 0,lty=2)
```



If a dataset is normally bivariate, we should expect to see a normal distribution in Y for all strata of X. Below we can see that this seems to hold for the `galton_heights` dataset.

```
galton_heights %>%
  mutate(z_father = round((father - mean(father))/sd(father))) %>%
  filter(z_father %in% -2:2) %>%
  ggplot() +
  stat_qq(aes(sample=son)) +
  facet_wrap(~z_father)
```

When two variables follow the bivariate normal distribution, computing the regression line is equivalent to computing conditional expectations.

$$\mathrm{E}(Y|X=x) = \mu_Y + \rho\frac{X-\mu_X}{\sigma_X}\sigma_Y$$

In summary, if our data is approximately bivariate, then the conditional expectation, the best prediction of Y given we know the value of X, is given by the regression line.

Variance Explained

The bivariate normal theory also tells us that the standard deviation of the *conditional* distribution described above is:

$$\mathrm{SD}(Y \mid X=x) = \sigma_Y\sqrt{1-\rho^2}$$

Specifically, it is reduced to $\sqrt{1-\rho^2} = \sqrt{1-0.25} = 0.86$ of what it was originally. We could say that the fathers height "explains" 14% of the sons height variability.

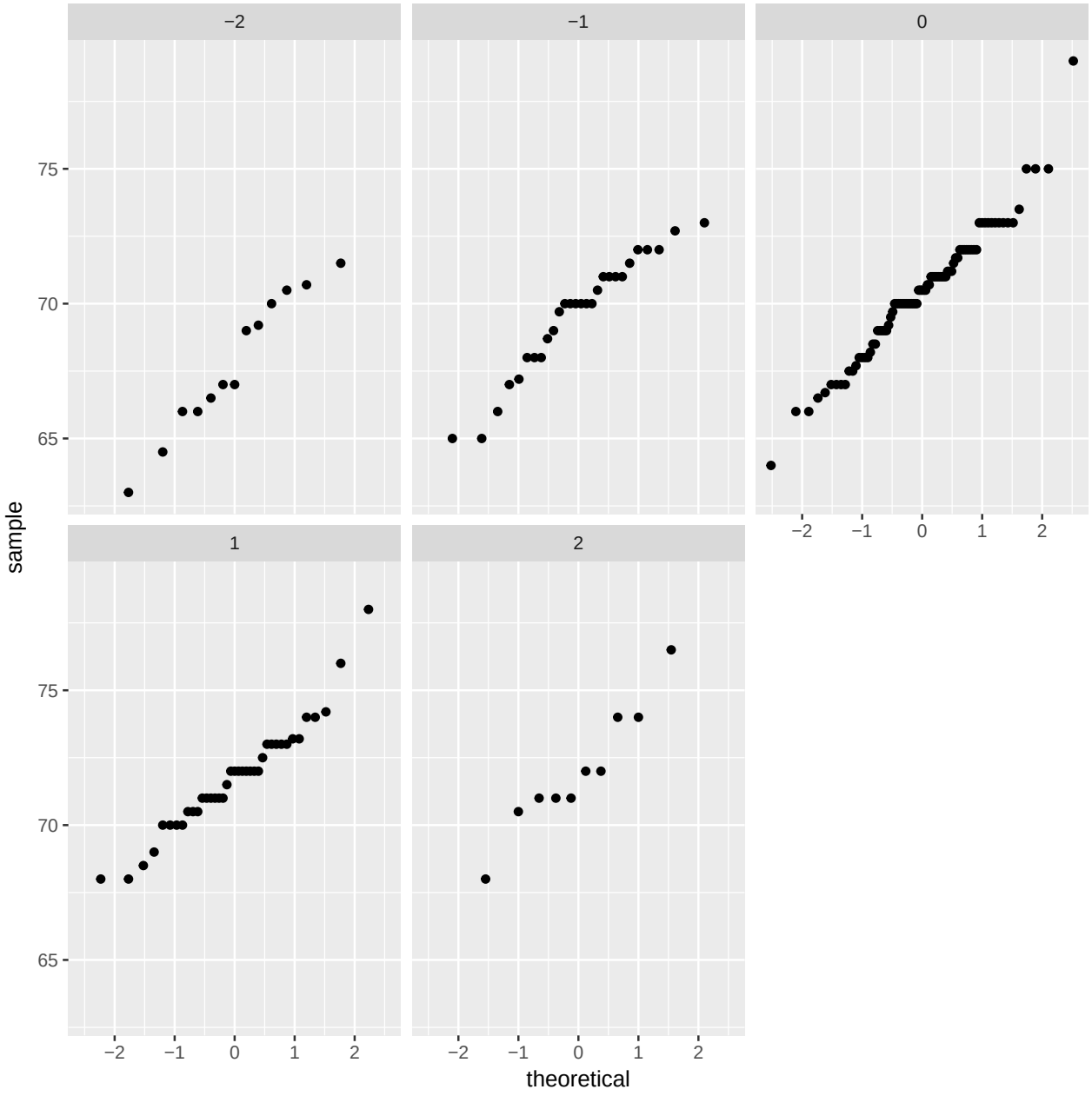


Figure 1: qqplots of son heights for four strata defined by father heights.

The statement X explains such and such percent of the variability is commonly used in academic papers. In this case, this percent actually refers to the variance (the SD squared). So if the data is bivariate normal, the variance is reduced by $1 - \rho^2$ so we say that X explains $1 - (1 - \rho^2) = \rho^2$ (the correlation squared) of the variance.

Note: the "variance explained" statement only makes sense when the data is approximated by a *bivariate normal distribution*.

The Other Regression Line

We could also compute the regression line to predict the father's height based on their sons'.

```
mu_x <- mean(galton_heights$father)
mu_y <- mean(galton_heights$son)
s_x <- sd(galton_heights$father)
s_y <- sd(galton_heights$son)
r <- cor(galton_heights$father, galton_heights$son)
m_1 <-  r * s_y / s_x
b_1 <- mu_y - m_1*mu_x
m_2 <-  r * s_x / s_y
b_2 <- mu_x - m_2*mu_y
```

Note that the second regression line is not the inverse of the first. Rather it is a function of recomputed conditional probabilities.

```
galton_heights %>%
  ggplot(aes(father, son)) +
  geom_point(alpha = 0.5) +
  geom_abline(intercept = b_1, slope = m_1, col = "blue") +
  geom_abline(intercept = -b_2/m_2, slope = 1/m_2, col = "red")
```

Introduction to Linear Models

Linear Models

Confounding

> A guiding principle for all linear regression analysis: *correlation is not causation*.

As an illustrative example, the classical statistic measuring professional baseball players' offensive ability is the batting average, or BA. The BA captures how often a player gets hit per at-bat. As famously discussed in Michael Lewis' *Moneyball*, it does not include the frequency with which a player gets on base after being walked, aka "bases on balls", or BBs.

We can see from the regression slope below that BBs are correlated with more runs per game.

```
bb_slope <- Teams %>%
  filter(yearID %in% 1961:2001 ) %>%
  mutate(BB_per_game = BB/G, R_per_game = R/G) %>%
  lm(R_per_game ~ BB_per_game, data = .) %>%
  .$coef  %>%
  .[2]
bb_slope
```

```
## BB_per_game
##    0.735269
```

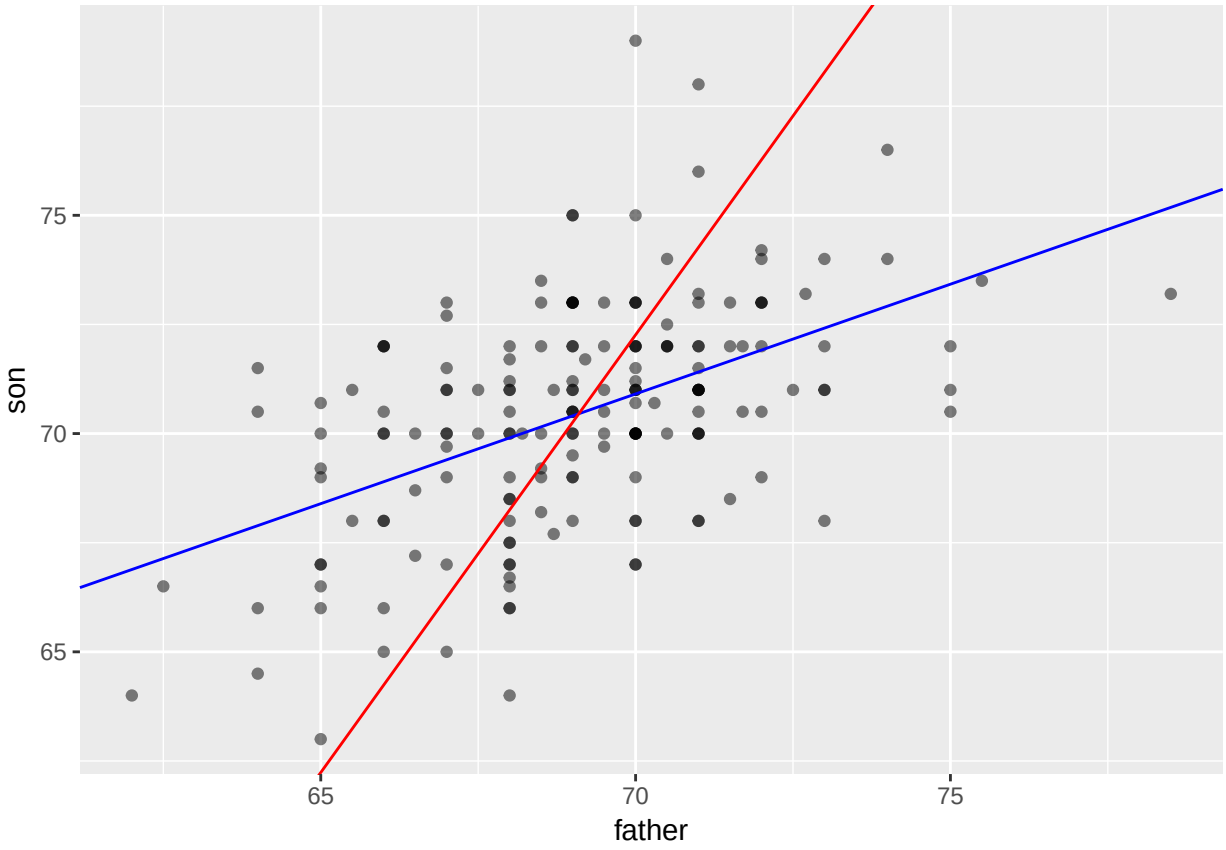


Figure 2: Regression Lines for Heights of Fathers to Predict Son's and Vice Versa

However, as we can see below, the correlation between BB and runs is stronger than the one between singles and runs, even though they have the same outcome (a player on first base). What explains this?

```
Teams %>%
  filter(yearID %in% 1961:2001 ) %>%
  mutate(Singles = (H-HR-X2B-X3B)/G, BB = BB/G, HR = HR/G,
         Runs = R/G) %>%
  summarize(cor(BB, Runs), cor(Singles, Runs), cor(BB, HR), cor(Singles, HR), cor(BB,Singles))
```

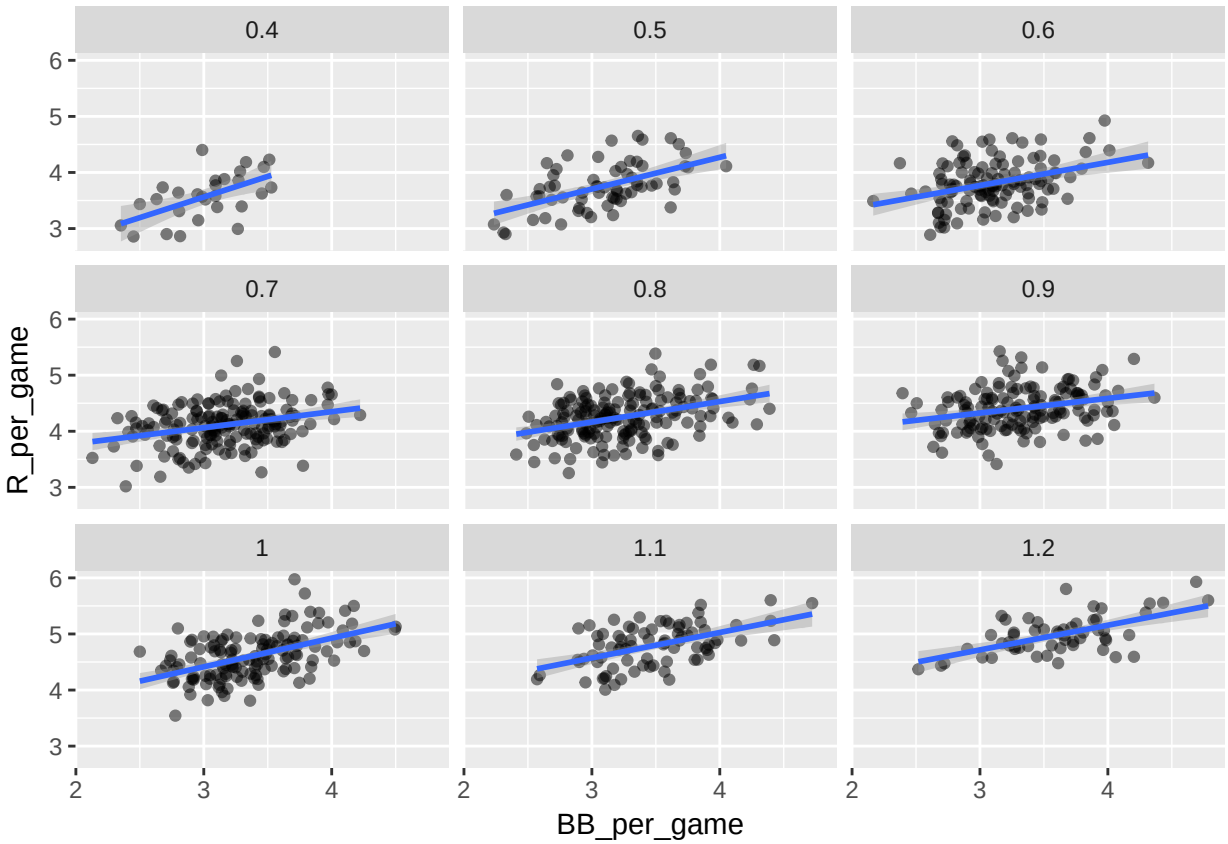
```
##   cor(BB, Runs) cor(Singles, Runs) cor(BB, HR) cor(Singles, HR)
## 1     0.5501986          0.2982577   0.4039125       -0.1738404
##   cor(BB, Singles)
## 1      -0.05605071
```

It turns out that BBs are also strongly correlated with HRs, since pitchers will sometimes avoid throwing strikes to known HR hitters. Unsurprisingly, HRs also strongly predict overall runs, so we can say that the correlation between BBs and runs is confounded by the third variable, HRs, which partially explains both.

Stratification and Multivariate Regression

We can control for the effect of HRs by comparing BBs to Runs within strata where HRs are fixed.

```
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR_strata = round(HR/G,1), #create strata by rounding to one decimal point
         BB_per_game = BB/G,
         R_per_game = R/G) %>%
  filter(HR_strata >= 0.4 & HR_strata <=1.2) #filter outliers
dat %>%
  ggplot(aes(BB_per_game, R_per_game)) +  #create scatterplot with regression line
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm") +
  facet_wrap(~HR_strata) #create scatterplot for each stratum
```



We can see that the correlations (and thus the slopes) of the lines are substantially reduced once HRs are fixed. The table below shows the slopes of the above graph.

```
dat %>%
  group_by(HR_strata) %>%
  summarize(slope = cor(BB_per_game, R_per_game)*sd(R_per_game)/sd(BB_per_game))
```

```
## # A tibble: 9 x 2
##   HR_strata slope
##       <dbl> <dbl>
## 1       0.4 0.734
## 2       0.5 0.566
## 3       0.6 0.412
## 4       0.7 0.285
## 5       0.8 0.365
## 6       0.9 0.261
## 7       1   0.511
## 8       1.1 0.454
## 9       1.2 0.440
```

We see the slopes are about the same as for singles (~0.5), which makes sense given that they both leave a runner on first base. If we were to switch the variables in the graphs above such that BBs were fixed and HRs were on the x-axis, the correlation between HRs and Runs would be about the same.

Incidentally, each of the variable pairs follows an approximately normal distribution, which means that we can use regression to make predictions.

Multiple Regression

Finding regression lines for each stratum can be expressed as follows:

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1(x_2)x_1 + \beta_2(x_1)x_2$$

Here, the slope of β_1 and β_2 changing depending on the value of BB and HR. However, when we take random variability into account, the slopes in each of the graphs don't change much. If they are in fact the same, we can create a much simpler model where we treat β_1 and β_2 as constants:

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

This model implies that if the number of home runs is fixed, we observe a *linear relationship* between runs and BBs, and that the slope of the regression line does *not* depend on the number of home runs. In this multivariate analysis, the BB slope is said to be *adjusted* for the HR effect.

But how do we estimate β_1 and β_2 from the data? For that, we need least squares estimates.

Linear Models

Before we go into a least squares estimates, a quick primer on linear models: It is common to presume as we have above that the variable pairs are bivariate normal and that the strength of thecorrelation between two variables does not depend a third. A linear model refers to a linear combination of variables, so $\beta_0+\beta_1 x_1+\beta_2 x_2$ is a linear combination of x_1 and x_2.

Key to remember: if your data is bivariate normal, the linear model holds.

For Galton's data, we can write the model for predicting sons' heights from their fathers' as follows:

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,\ i = 1, \ldots, N$$

Here, Y_i represents the predicted son's height, $\beta_0 + \beta_1 x_1$ represents the intercept and effect of the father's height, and ε_i represents the error (which we assume is normally distributed with expected value zero). We

To make the intercept more interpretable, we can center the fathers' heights so that the intercept represents the predicted height of the son of father of mean height (i.e. β_0 would be the height when $x_i = \bar{x}$).

$$Y_i = \beta_0 + \beta_1(x_i - \bar{x}) + \varepsilon_i,\ i = 1, \ldots, N$$

Least Squares Estimate

To make our model useful, we estimate the unknown βs with the residual sum of squares (RSS) equation:

$$RSS = \sum_{i=1}^{n} \{Y_i - (\beta_0 + \beta_1 x_i)\}^2$$

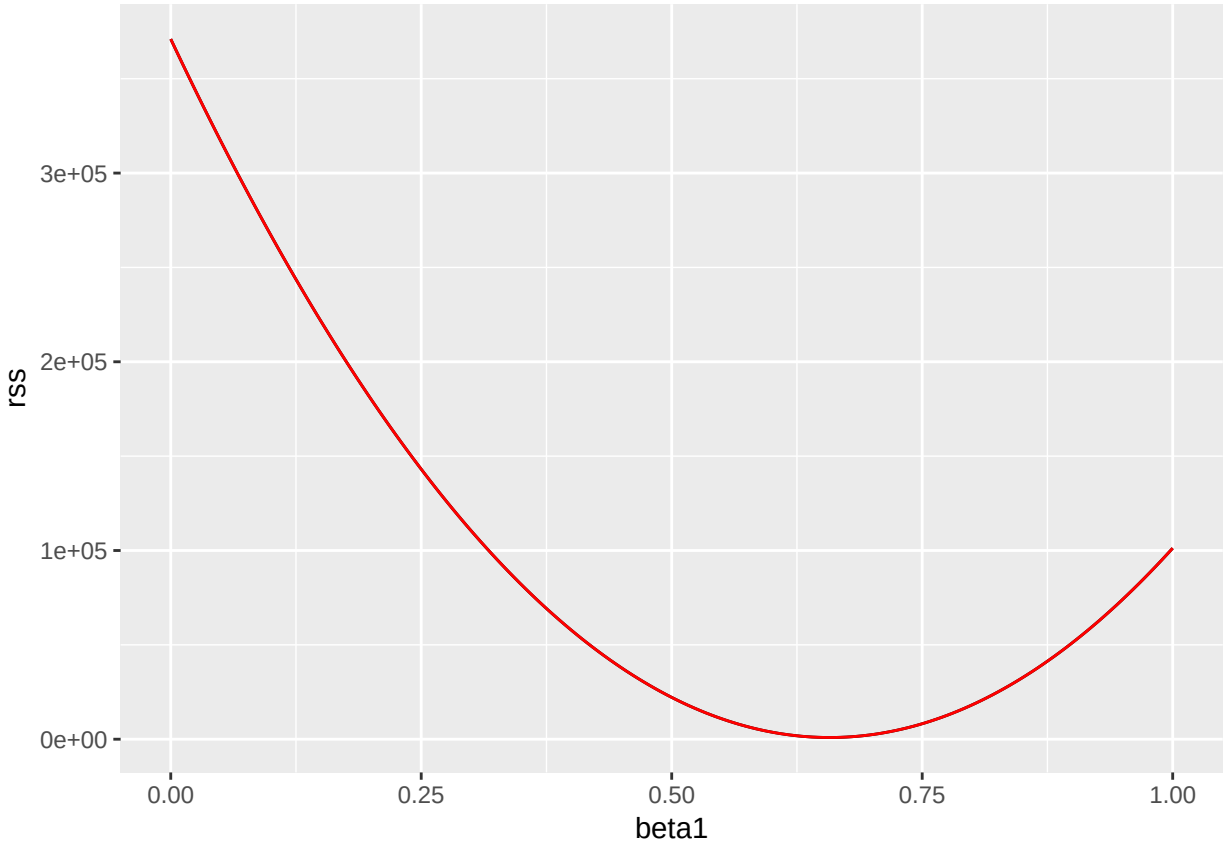
This function gives us the sum of the squared error of the prediction for a given set of βs. The set of βs that minimizes the RSS is the least squares estimate, which we then use to make predictions with our model.

Let's write a function to find the residual least squares for the Galton data given a set of βs:

```
rss <- function(beta0, beta1, data){
  resid <- galton_heights$son - (beta0+beta1*galton_heights$father)
  return(sum(resid^2))
}
```

To find the minimum (LSE), we could create a 3D plot with the betas on the x and y axis and RSS on the z. More simply, the following shows what the plot looks like in 2 dimensions when β_0 is set to 25.

```
beta1 = seq(0, 1, len=nrow(galton_heights))
results <- data.frame(beta1 = beta1,
                      rss = sapply(beta1, rss, beta0 = 25))
results %>% ggplot(aes(beta1, rss)) + geom_line() +
  geom_line(aes(beta1, rss), col=2)
```



Of course, the minimum (around 0.65) doesn't account for any of the other β_0s. To find the true LSE, we would need to take the partial derivatives of each β and solve. This is outside the scope of this series, but we will learn the functions that calculate this in R.

The Lm Function

To find the LSEs for our galton data model, we use the `lm` function.

To fit the following model,

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

```
fit <- lm(son ~ father, data = galton_heights)
fit
```

```
##
## Call:
## lm(formula = son ~ father, data = galton_heights)
##
## Coefficients:
## (Intercept)       father
##     35.7125       0.5028
```

The `lm` function predicts the variable to the left of the tilda based on the one(s) to the right. With `summary`, we can extract more information about the fitted model.

```
summary(fit)
```

```
##
## Call:
## lm(formula = son ~ father, data = galton_heights)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.9022 -1.4050  0.0922  1.3422  8.0922
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 35.71249    4.51737   7.906 2.75e-13 ***
## father       0.50279    0.06533   7.696 9.47e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.22 on 177 degrees of freedom
## Multiple R-squared:  0.2507, Adjusted R-squared:  0.2465
## F-statistic: 59.23 on 1 and 177 DF,  p-value: 9.473e-13
```

```
library(Lahman)
data("LahmanData")
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR_per_game = HR/G,
         BB_per_game = BB/G,
         Runs_per_game = R/G)
fit2 <- lm(Runs_per_game ~ HR_per_game + BB_per_game, data = dat)
summary(fit2)
```

```
##
## Call:
## lm(formula = Runs_per_game ~ HR_per_game + BB_per_game, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.87322 -0.24503 -0.01446  0.23870  1.24222
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.74440    0.08235   21.18   <2e-16 ***
## HR_per_game  1.56112    0.04896   31.89   <2e-16 ***
## BB_per_game  0.38740    0.02701   14.34   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3484 on 1023 degrees of freedom
## Multiple R-squared:  0.6503, Adjusted R-squared:  0.6496
## F-statistic: 951.2 on 2 and 1023 DF,  p-value: < 2.2e-16
```

LSEs are random variables

The LSE are derived from the data $Y_1, \ldots, Y_N$, which are random. Thus, our estimates are random variables.

Ee can run a Monte Carlo simulation in which we assume the son and father height data defines a population, take a random sample of size $N = 50$ and compute the regression slope coefficient for each one:

```
B <- 1000
N <- 50
lse <- replicate(B, {
  sample_n(galton_heights, N, replace = TRUE) %>%
   lm(son ~ father, data = .) %>% .$coef
})
lse <- data.frame(beta_0 = lse[1,], beta_1 = lse[2,])
```

We can see the variability of the estimates by plotting their distributions:

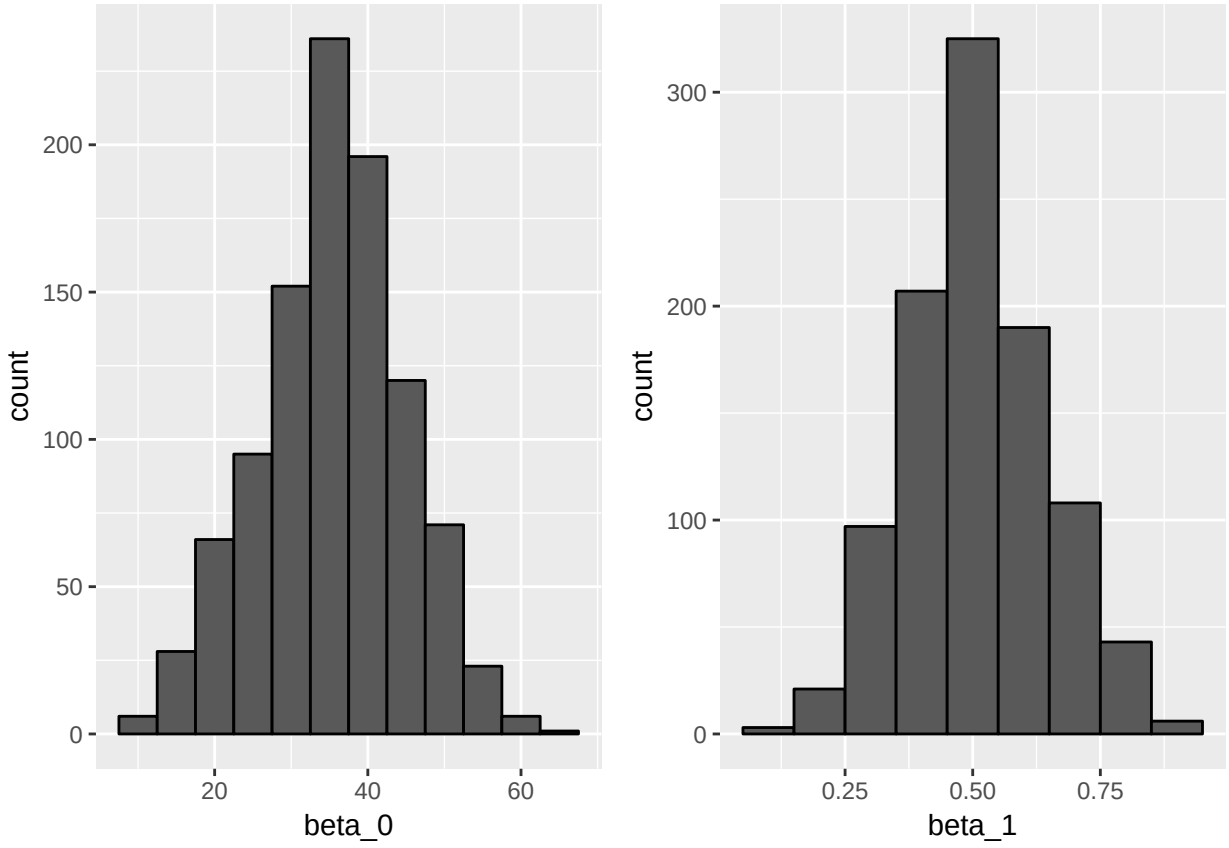
```
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.2.5
```

```
##
## Attaching package: 'gridExtra'
```

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

```
p1 <- lse %>% ggplot(aes(beta_0)) + geom_histogram(binwidth = 5, color = "black")
p2 <- lse %>% ggplot(aes(beta_1)) + geom_histogram(binwidth = 0.1, color = "black")
grid.arrange(p1, p2, ncol = 2)
```



Because the CLT applies, the least squares estimates will be approximately normal with expected value β_0 and β_1 respectively. The standard error of these results is provided as well.

```
 sample_n(galton_heights, N, replace = TRUE) %>%
   lm(son ~ father, data = .) %>% summary
```

```
##
## Call:
## lm(formula = son ~ father, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.5902 -1.5610 -0.1086  1.7041  7.3022
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  14.1914    11.5424   1.230    0.225
## father        0.8215     0.1661   4.946  9.7e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.635 on 48 degrees of freedom
## Multiple R-squared:  0.3376, Adjusted R-squared:  0.3238
## F-statistic: 24.46 on 1 and 48 DF,  p-value: 9.699e-06
```

We can see that the standard error from the summary is similar to the simulated error for both βs.

```
lse %>% summarize(se_0 = sd(beta_0), se_1 = sd(beta_1))
```

```
##       se_0    se_1
## 1 9.395117 0.13572
```

If you assume that N is large enough to use the CLT or that the errors are normal (and use the t-distribution), you can construct confidence intervals. The t-statistic in the `lm` summary uses the latter assumption. Namely, the LSE divided by their standard error, $\hat{\beta}_0/\hat{\text{SE}}(\hat{\beta}_0)$ and $\hat{\beta}_1/\hat{\text{SE}}(\hat{\beta}_1)$ follow a t-distribution with $N-p$ degrees of freedom and with p the number of parameters in our model. In the case of height $p=2$, the two p-value are testing the null hypothesis that $\beta_0 = 0$ and $\beta_1 = 0$ respectively.

LSEs are correlated

```
lse %>% summarize(cor(beta_0, beta_1))
```

```
##   cor(beta_0, beta_1)
## 1          -0.9994646
```

However, the correlation depends on how the predictors are defined or transformed.

Here we standardize the father heights, which changes x_i to $x_i - \bar{x}$.

```
B <- 1000
N <- 50
lse <- replicate(B, {
      sample_n(galton_heights, N, replace = TRUE) %>%
      mutate(father = father - mean(father)) %>%
      lm(son ~ father, data = .) %>% .$coef
})
```

Observe what happens to the correlation in this case:

```
cor(lse[1,], lse[2,])
```

```
## [1] -0.1697038
```

Standardizing the father's height dramatically reduces the correlation between the simulated βs.

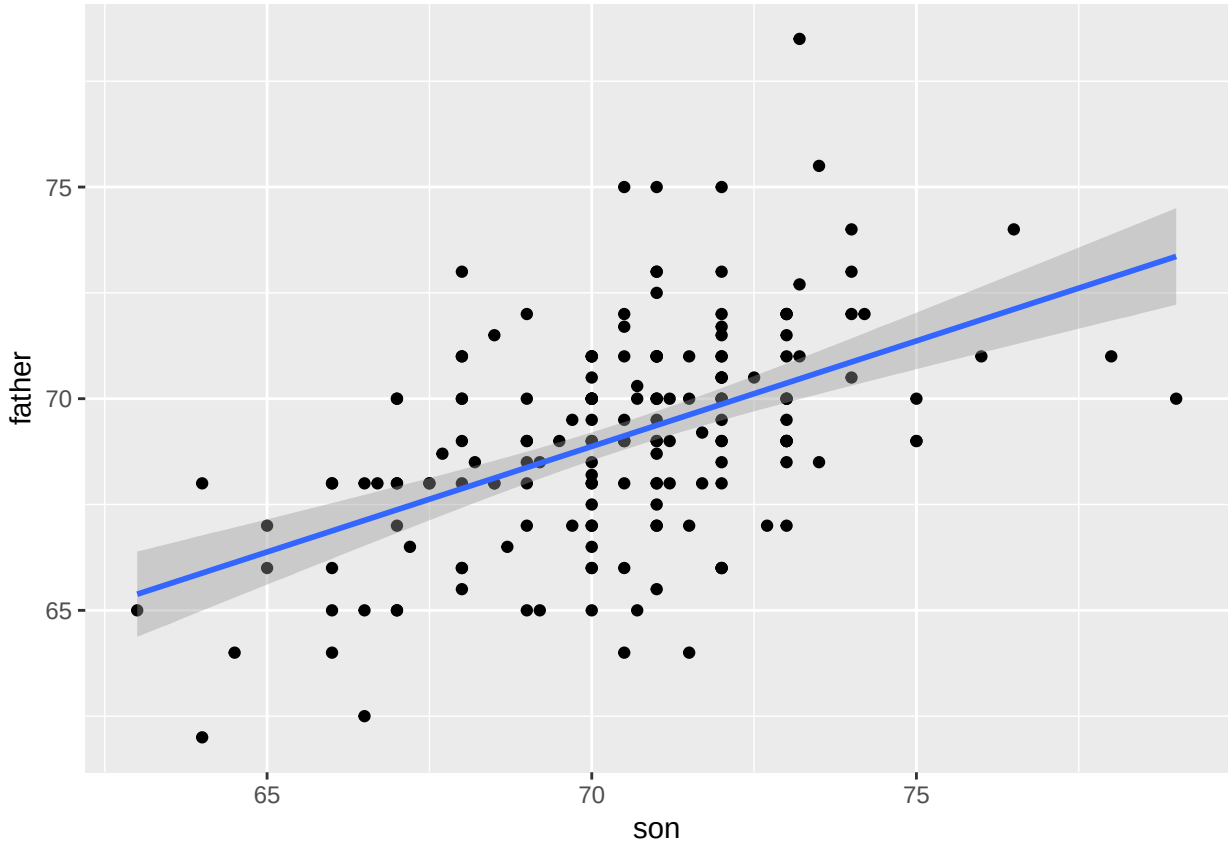
Predicted Variables are Random Variables

The βs represent the factor by which the predicted variable changes based on the predictor. Similar to the true proportion of a population, we don't know the "true" βs, but we can estimate them from our sample data. These estimates are denoted with a hat: $\hat{\beta}$. For the Galton data, we have the following equation:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

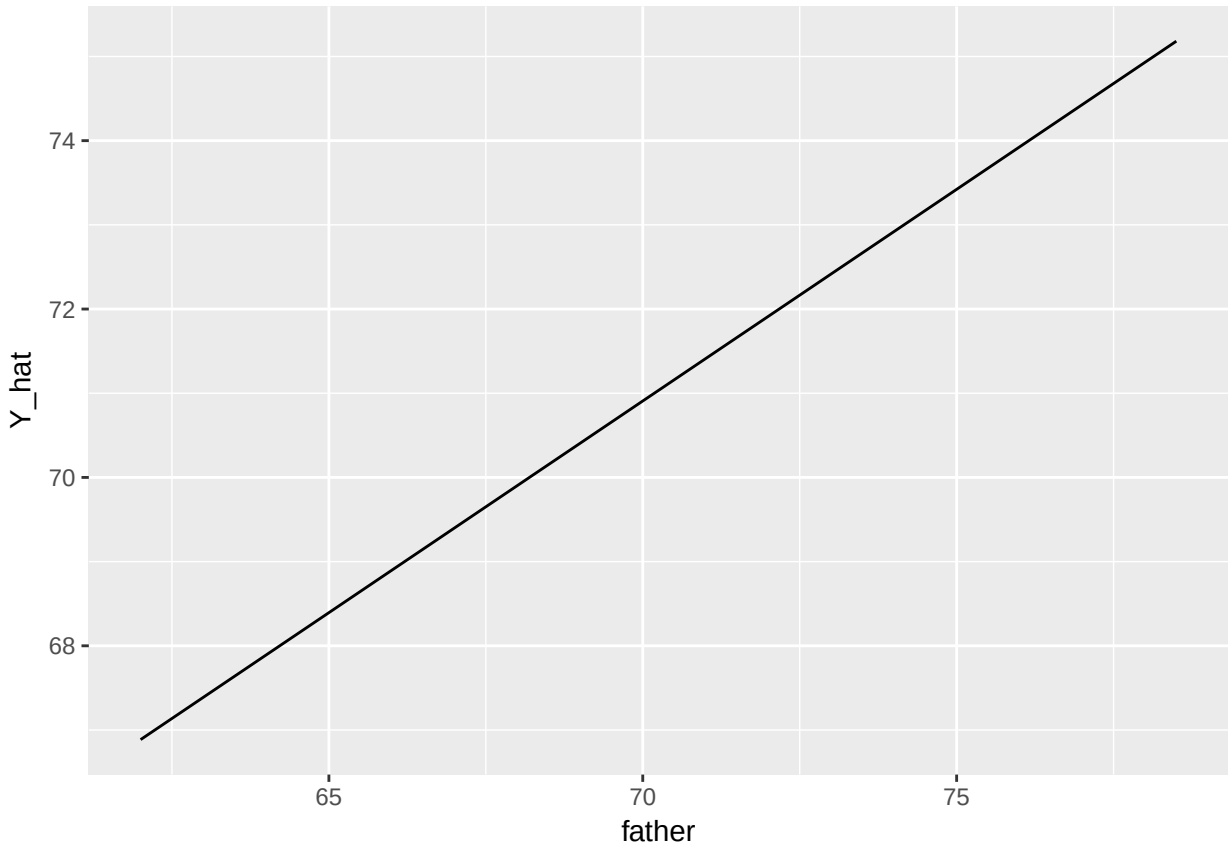
If we assume that the errors are normal or N is large enough, we can also use theory to construct the confidence intervals. `ggplot` has a function within `geom_smooth(method = "lm")` that plots $\hat{Y}$ surrounded by the confidence interval.

```
galton_heights %>% ggplot(aes(son, father)) +
  geom_point() +
  geom_smooth(method = "lm")
```



The `predict` function returns the prediction from an `lm` object input. The confidence intervals can also be extracted.

```
galton_heights %>%
  mutate(Y_hat = predict(lm(son ~ father, data = .))) %>%
  ggplot(aes(father, Y_hat)) +
  geom_line()
```




```
fit <- galton_heights %>% lm(son ~ father, data = .)
Y_hat <- predict(fit, se.fit = TRUE)
names(Y_hat)
```

```
## [1] "fit"            "se.fit"         "df"             "residual.scale"
```

Dataframes in the tidyverse

Tibbles

Tibbles are a specialized type of dataframe that are always the output of functions such as `group_by` and `summarize` from the `tidyverse`. Base functions such as `lm` are not capable of reading tibbles grouped with `group_by` and thus provide the same output as would result from an ungrouped dataframe.

Tibbles are the default dataframe of the tidyverse.

Notably, `select`, `filter`, `mutate`, and `arrange` don't necessarily return tibbles. Rather, they preserve the class of the input.

Tibbles have important differences from dataframes: 1. Tibbles handle printing datasets with many columns and rows better than dataframes, showing a subset of the data and a description of the remaining rows and columns. 2. Unlike dataframes, tibbles are always preserved when they are subset. Since most `tidyverse` functions require dataframes as input, this can eliminate the need to reclass subsetted dataframes. 3. Tibbles will provide a warning if a column or row is called that does not exist, whereas dataframes will return a NULL. The warning is useful for debugging. 4. While dataframes are limited to integers, Booleans, and strings, tibbles can contain more complex objects, such as functions. 5. Tibbles can be grouped. The function `group_by` returns a grouped tibble, which tidyverse functions can understand and incorporate into their outputs (e.g. grouped outputs from `summarize`).

Bridging Base R to the Tidyverse with Do

The `do` function serves as a bridge between base R functions such as `lm` and the `tidyverse`. `do` understands grouped tibbles and will always return a dataframe that can be piped into the `tidyverse` function.

If we return to our stratification of runs (R) and bases on balls (BB) by HRs for a moment, we can recall that the `lm` function does not understand grouped tibbles, and so does not return our desired output when we give it Rs and BBs grouped by HR strata. We can bridge the gap with the `do` function:

```
dat <- Teams %>% filter(yearID %in% 1961:2001) %>%
  mutate(HR = round(HR/G, 1),
         BB = BB/G,
         R = R/G) %>%
  select(HR, BB, R) %>%
  filter(HR >= 0.4 & HR<=1.2)
```

```
dat %>%
  group_by(HR) %>%
  do(fit = lm(R~BB, data = .))
```

```
## Source: local data frame [9 x 2]
## Groups: <by row>
##
## # A tibble: 9 x 2
##      HR fit
## * <dbl> <list>
## 1   0.4 <S3: lm>
## 2   0.5 <S3: lm>
## 3   0.6 <S3: lm>
## 4   0.7 <S3: lm>
## 5   0.8 <S3: lm>
## 6   0.9 <S3: lm>
## 7   1   <S3: lm>
## 8   1.1 <S3: lm>
## 9   1.2 <S3: lm>
```

Note that the output above has the HR strata in one column and the `lm` object outputs in another, which isn't very useful. Also, the column must be defined within `do` (i.e. `fit` above) else the output will not be a dataframe and `do` will return an error.

To get a useful output, we need the output of `do` to be a dataframe. One way to accomplish this is to write a function that extracts the variables we want and puts them into a dataframe:

```
get_slope <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(slope = fit$coefficients[2],
             se = summary(fit)$coefficient[2,2])
}#function fits a linear model to the input and returns a dataframe with the slope and standard error
```

Then extracting the slope and standard error is as simple as grouping the data and plugging our function into `do`.

```
dat %>%
  group_by(HR) %>%
  do(get_slope(.))
```

```
## # A tibble: 9 x 3
## # Groups:   HR [9]
```

```
##      HR slope     se
##   <dbl> <dbl>  <dbl>
## 1   0.4 0.734 0.208
## 2   0.5 0.566 0.110
## 3   0.6 0.412 0.0974
## 4   0.7 0.285 0.0705
## 5   0.8 0.365 0.0652
## 6   0.9 0.261 0.0754
## 7   1   0.511 0.0749
## 8   1.1 0.454 0.0855
## 9   1.2 0.440 0.0801
```

Note that we do not name the output of do. This would cause slope and se to be put into dataframes within the tibble.

Conveniently, `do` concatenates dataframe outputs that have multiple rows. If we wanted the Intercept and BB from our fit above, we could write the following code.

```
get_lse <- function(data){
  fit <- lm(R ~ BB, data = data)
  data.frame(term = names(fit$coefficients),
    slope = fit$coefficients,
    se = summary(fit)$coefficient[,2])
}
dat %>%
  group_by(HR) %>%
  do(get_lse(.))
```

Extracting from objects with Broom

`Broom` has three main functions, each of which extracts information from an `lm` object and returns a dataframe. These functions are `tidy`, `glance`, and `augment`.

- `tidy` returns estimates and related information as a dataframe
  - setting `conf.int` to TRUE adds columns with confidence intervals to the output
- `augment` adds the results of the model to the original data. This includes predictions, residuals, and cluster assignments
- `glance` creates a concise one-row summary of the model R^2 and adjusted R^2

For example, stringing together `tidy` and `do` efficiently constructs a dataframe from `lm` object outputs.

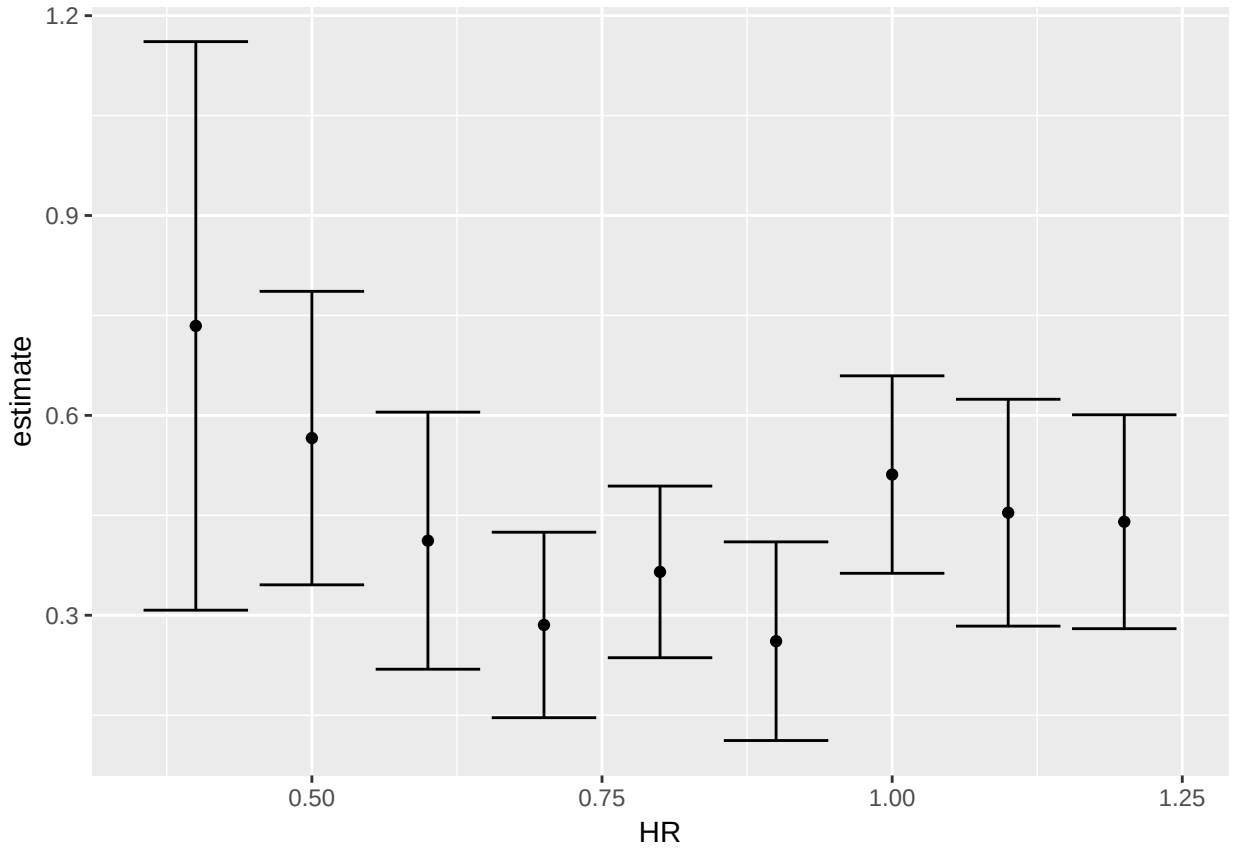
```
library(broom)
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE))
```

```
## # A tibble: 18 x 8
## # Groups:   HR [9]
##       HR term     estimate std.error statistic  p.value conf.low conf.high
##    <dbl> <chr>       <dbl>     <dbl>     <dbl>    <dbl>    <dbl>     <dbl>
##  1   0.4 (Interc~    1.36     0.631       2.16 4.05e- 2   0.0631     2.66
##  2   0.4 BB          0.734    0.208       3.54 1.54e- 3   0.308      1.16
##  3   0.5 (Interc~    2.01     0.344       5.84 2.07e- 7   1.32       2.69
##  4   0.5 BB          0.566    0.110       5.14 3.02e- 6   0.346      0.786
##  5   0.6 (Interc~    2.53     0.305       8.32 2.43e-13   1.93       3.14
##  6   0.6 BB          0.412    0.0974      4.23 4.80e- 5   0.219      0.605
```

```
##  7   0.7 (Interc~    3.21     0.225      14.3  1.49e-30   2.76       3.65
##  8   0.7 BB          0.285    0.0705      4.05 7.93e- 5   0.146      0.425
##  9   0.8 (Interc~    3.07     0.213      14.4  5.06e-31   2.65       3.49
## 10   0.8 BB          0.365    0.0652      5.59 8.94e- 8   0.236      0.494
## 11   0.9 (Interc~    3.54     0.252      14.1  1.37e-28   3.04       4.04
## 12   0.9 BB          0.261    0.0754      3.46 7.12e- 4   0.112      0.410
## 13   1   (Interc~    2.88     0.255      11.3  5.25e-21   2.38       3.39
## 14   1   BB          0.511    0.0749      6.82 3.06e-10   0.363      0.659
## 15   1.1 (Interc~    3.21     0.300      10.7  6.46e-17   2.61       3.81
## 16   1.1 BB          0.454    0.0855      5.31 1.03e- 6   0.284      0.624
## 17   1.2 (Interc~    3.40     0.291      11.7  2.33e-16   2.81       3.98
## 18   1.2 BB          0.440    0.0801      5.50 1.07e- 6   0.280      0.601
```

The above can be plugged straight into `ggplot` to make a useful graph that confirms there are no significant differences between the slopes (corroborating the assumption for the linear model that the strength of the relationship between R and BB is independent of HR).

```
dat %>%
  group_by(HR) %>%
  do(tidy(lm(R ~ BB, data = .), conf.int = TRUE)) %>%
  filter(term == "BB") %>%
  select(HR, estimate, conf.low, conf.high) %>%
  ggplot(aes(HR, y = estimate, ymin = conf.low, ymax = conf.high)) +
  geom_errorbar() +
  geom_point()
```



Building a Better Offensive Metric for Baseball

Our earlier exploration of the data on led us to this model to predict runs:

$$\mathrm{E}[R \mid BB = x_1, HR = x_2] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

We showed that the the data and each of the conditional distributions are approximately normal, justifying a linear model:

$$Y_i = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon_i$$

with Y_i runs per game, x_1 walks per game, and x_2 HRs per game. We plug both predictors into `lm` using the `+` operator.

```
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>%
  mutate(BB = BB/G, HR = HR/G,  R = R/G) %>%
  lm(R ~ BB + HR, data = .)
```

`tidy` returns a readable summary.

```
tidy(fit, conf.int = TRUE)
```

```
##          term  estimate  std.error statistic       p.value  conf.low
## 1 (Intercept) 1.7444018 0.08234949  21.18291  7.300077e-83 1.5828085
## 2          BB 0.3873978 0.02700913  14.34322  1.195862e-42 0.3343982
## 3          HR 1.5611228 0.04895718  31.88751 1.751870e-155 1.4650548
##   conf.high
## 1 1.9059950
## 2 0.4403974
## 3 1.6571907
```

If we want to construct a metric to pick players, we need to consider singles, doubles, and triples as well. Can we build a model that predicts runs based on all these outcomes?

Let's assume for the moment that all of these additional variables are jointly normal. This means that if we pick any one of them, and hold the other four fixed, the relationship with the outcome is linear and the slope does not depend on the four values held constant. If this is true, then a linear model for our data is:

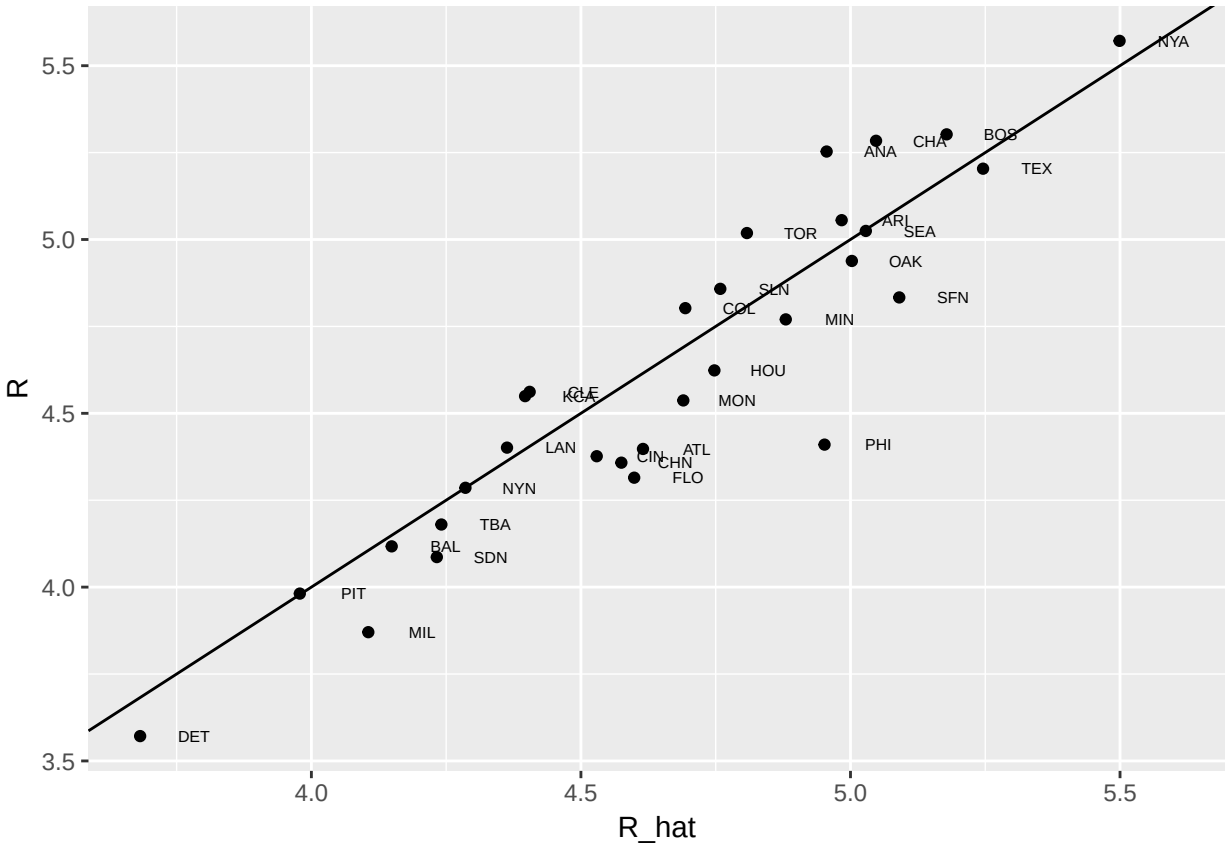
$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \beta_5 x_{i,5} + \varepsilon_i$$

with x_1, x_2, x_3, x_4, x_5 representing BB, singles, doubles, triples, and HR respectively.

To predict how many runs each team will score in the 2002 season, we use the data up through 2001. We fit the model based on the equation above and compare our predicted runs to the actual scores:

```
fit <- Teams %>%
  filter(yearID %in% 1961:2001) %>% #filter to data before 2002
  mutate(BB = BB/G, #create per game variables
         singles = (H-X2B-X3B-HR)/G,
         doubles = X2B/G,
         triples =X3B/G,
         HR=HR/G,
         R=R/G) %>%
  lm(R ~ BB + singles + doubles + triples + HR, data = .) #fit model based on selected variables
```

```
Teams %>%
  filter(yearID %in% 2002) %>% #using the data from 2002 to find the actual runs
  mutate(BB = BB/G, #create our variables
         singles = (H-X2B-X3B-HR)/G,
         doubles = X2B/G,
         triples =X3B/G,
         HR=HR/G,
         R=R/G)  %>%
  mutate(R_hat = predict(fit, newdata = .)) %>% #predict runs based on the other scores from 2002
  ggplot(aes(R_hat, R, label = teamID)) + #plot
  geom_point() + #plotting predicted and real values to assess accuracy
  geom_text(nudge_x=0.1, cex = 2) + #add team names
  geom_abline() #identity line
```



Clearly, our model does a pretty good job predicting the real results.

To create a player-specific model, we need to adjust our metrics so that they approximate individuals rather than teams (since we've been using team-level statistics). We need to adjust by taking into account the "per plate appearance" rate, which controls for the different frequencies with which players get a chance to hit.

To make the per-game team rate comparable to the per-plate-appearance player rate, we compute the average number of team plate appearances per game:

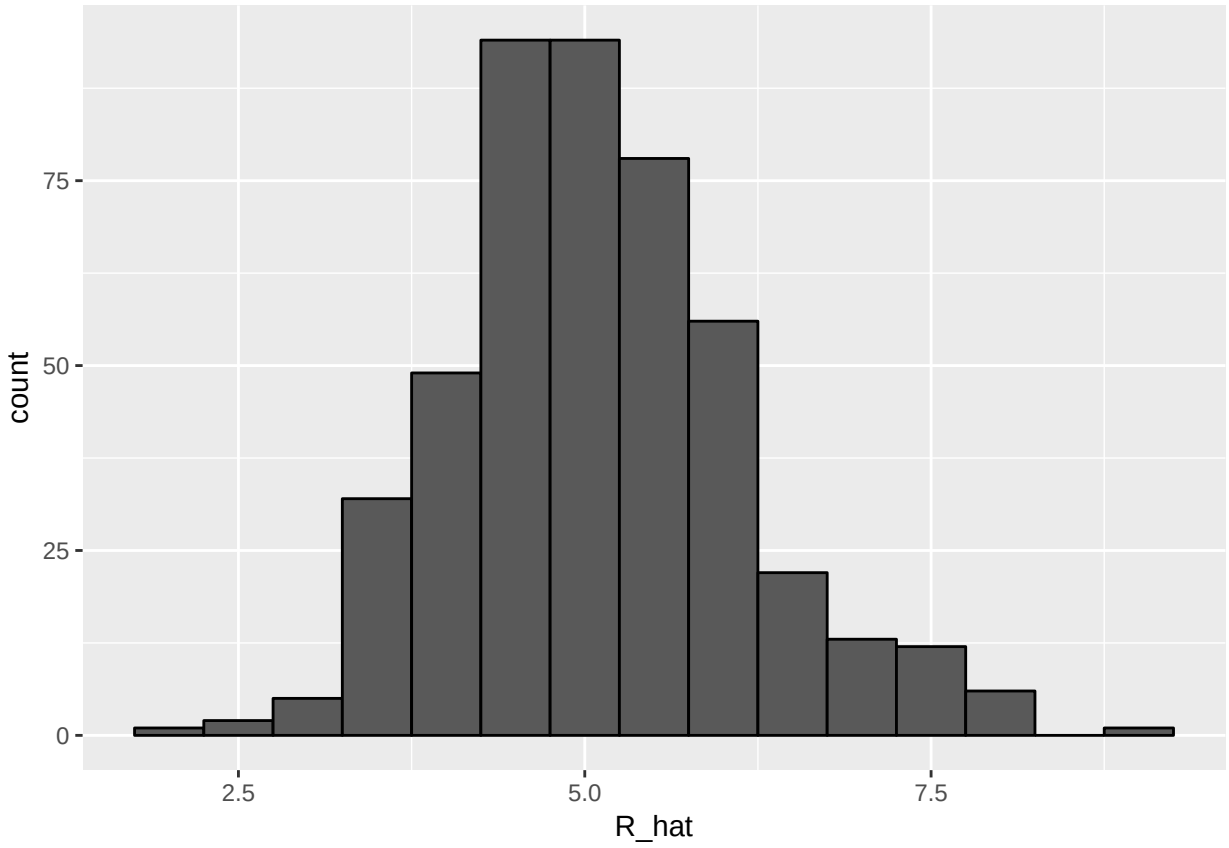
```
pa_per_game <- Batting %>% filter(yearID == 2002) %>%
  group_by(teamID) %>%
  summarize(pa_per_game = sum(AB+BB)/max(G)) %>%
  .$pa_per_game %>%
  mean
```

We compute the per-plate-appearance rates for players available in 2002 on data from 1999-2001. To avoid small sample artifacts, we filter players with few plate appearances. Here is the entire calculation in one line:

```
players <- Batting %>% filter(yearID %in% 1999:2001) %>%
  group_by(playerID) %>%
  mutate(PA = BB + AB) %>%
  summarize(G = sum(PA)/pa_per_game,
    BB = sum(BB)/G,
    singles = sum(H-X2B-X3B-HR)/G,
    doubles = sum(X2B)/G,
    triples = sum(X3B)/G,
    HR = sum(HR)/G,
    AVG = sum(H)/sum(AB),
    PA = sum(PA)) %>%
  filter(PA >= 300) %>%
  select(-G) %>%
  mutate(R_hat = predict(fit, newdata = .))
```

The player specific predicted runs computed here can be interpreted as the number of runs we predict a team will score if all batters are exactly like that player. A histogram shows that there is wide variability across players:

```
players %>% ggplot(aes(R_hat)) +
  geom_histogram(binwidth = 0.5, color = "black")
```



To actually construct the team, we'll need to know the salary of each player and their defensive position (since we can only have 1 of each). We also remove the position "OF", which denotes multiple positions in the outfield, and pitchers, who don't hit in the American League.

```r
players <- Salaries %>% #join players and salary datasets
  filter(yearID == 2002) %>%
  select(playerID, salary) %>%
  right_join(players, by="playerID")
players <- Fielding %>% filter(yearID == 2002) %>% #remove OF and P
  filter(!POS %in% c("OF","P")) %>%
  group_by(playerID) %>%
  top_n(1, G) %>% #select most common position
  filter(row_number(G) == 1) %>% #choose first row for ties
  ungroup() %>%
  select(playerID, POS) %>%
    right_join(players, by="playerID") %>% #join player stats to player IDs and position
  filter(!is.na(POS)  & !is.na(salary)) #remove players with missing info
players <- Master %>%
  select(playerID, nameFirst, nameLast, debut) %>% # select
  right_join(players, by="playerID")
```

```r
players %>% select(nameFirst, nameLast, POS, salary, R_hat) %>%
  arrange(desc(R_hat)) %>%
  top_n(10)
```

```
## Selecting by R_hat
```

```
##    nameFirst    nameLast POS   salary    R_hat
## 1       Todd      Helton  1B  5000000 8.228149
## 2      Jason      Giambi  1B 10428571 7.993738
## 3     Albert      Pujols  3B   600000 7.536519
## 4      Nomar Garciaparra  SS  9000000 7.505192
## 5       Jeff     Bagwell  1B 11000000 7.475011
## 6       Alex   Rodriguez  SS 22000000 7.439996
## 7     Carlos     Delgado  1B 19400000 7.368549
## 8     Rafael    Palmeiro  1B  8712986 7.257184
## 9       Mike      Piazza   C 10571429 7.161069
## 10       Jim       Thome  1B  8000000 7.156321
```