Elementary Regression Example

- Downloads a file
- Explores the data
- Fits a linear regression model, creates output tables
- Creates some diagnostic plots
- Demonstrates my Constitutional right to name parameters anything I want (c_j, not β_j)

Get Data For Regression

```
## Either use the existing "StrengthJobData.rds" file, or make one
if (file.exists("StrencthJobData.rds")) {
   dat <- readRDS("StrengthJobData.rds")
 } else {
   dat <- read.table(url("http://pj.freefaculty.org/guides/stat/
        DataSets/StrengthJobData/StrengthJobdata.txt"), header =
        TRUE)
   saveRDS(dat, file = "StrengthJobData.rds")
 }
```

README says...

Data were collected from electricians, construction and maintenance workers, auto mechanics, and linemen. Two measures of strength were gathered from each participant, reflecting grip and arm strength via the Jackson Evaluation System (a piece of strength-testing equipment). Each participant was asked to exert as much force as they could for a period of 2 seconds, equipment recording the maximum force exerted in pounds. Supervisors for each worker were asked to rate the employee's performance in his/her physical tasks on a 60-pt scale. Also, a simulated wrench, used to measure exerted force, was used to obtain an objective measure of practical job performance.

What Do We Have?

```
str(dat)
```

```
'data.frame': 147 obs. of  4 variables:
 $ GRIP    : num  105.5 106.5 94 90.5 104 ...
 $ ARM     : num  80.5 93 81 33.5 47.5 ...
 $ RATINGS: num  31.8 39.8 46.8 52.2 31.2 46.6 29.8 39 50.6 40.1 ...
 $ SIMS    : num  1.18 0.94 0.84 -2.45 1 4.38 -0.38 -0.01 -0.99 -0.04
      ...
```

Grab Some Summary Stats I Want

summary(dat) would be a nice start, but the output hard to manage. So Build own summary:

```
sumdat <- apply(dat, 2, quantile, na.rm = TRUE)
sumdat <- rbind(sumdat, mean= apply(dat, 2, mean, na.rm = TRUE))
sumdat <- rbind(sumdat, sd= apply(dat, 2, sd, na.rm = TRUE))
sumdat <- rbind(sumdat, var= apply(dat, 2, var, na.rm = TRUE))
sumdat
```

	GRIP	ARM	RATINGS	SIMS
0%	29.00000	19.00000	21.600000	−4.1700000
25%	94.00000	64.50000	34.800000	−0.9650000
50%	111.00000	81.50000	41.300000	0.1600000
75%	124.50000	94.00000	47.700000	1.0700000
100%	189.00000	132.00000	57.200000	5.1700000
mean	110.23129	78.75170	41.009878	0.2017687
sd	23.62987	21.10933	8.521865	1.6789742
var	558.37079	445.60402	72.622184	2.8189544

This is, essentially, what rockchalk::summarize does for us.

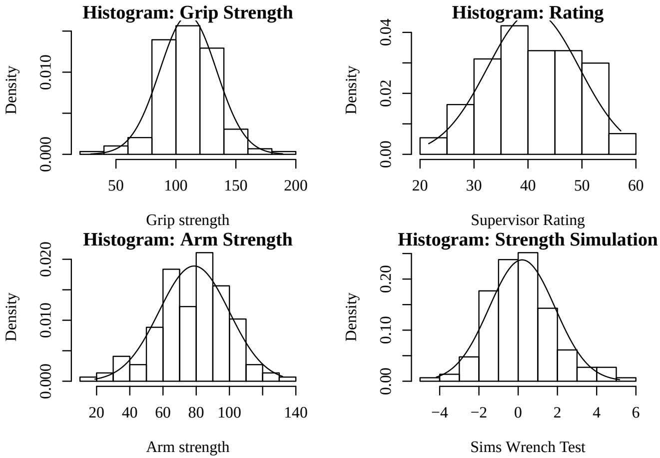
rockchalk::summarize

sumdat2 is a structured object. We could use sumdat2$numerics instead of sumdat in what follows

```
library(rockchalk)
sumdat2 <- summarize(dat)
sumdat2$numerics
```

	ARM	GRIP	RATINGS	SIMS
0%	19.00	29.00	21.600	−4.1700
25%	64.50	94.00	34.800	−0.9650
50%	81.50	111.00	41.300	0.1600
75%	94.00	124.50	47.700	1.0700
100%	132.00	189.00	57.200	5.1700
mean	78.75	110.20	41.010	0.2018
sd	21.11	23.63	8.522	1.6790
var	445.60	558.40	72.620	2.8190
NA's	0.00	0.00	0.000	0.0000
N	147.00	147.00	147.000	147.0000

4 Histograms with Normal PDF superimposed



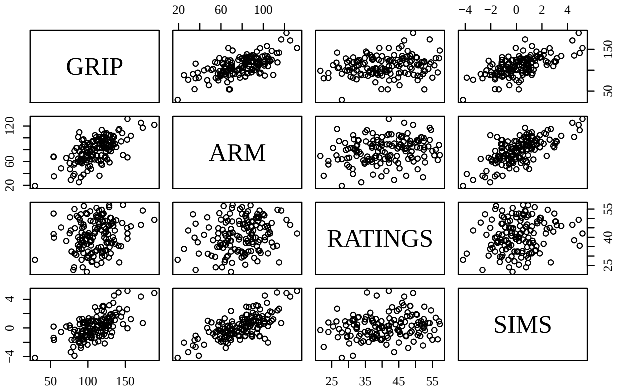
4 Histograms with Normal PDF superimposed ...

```
par(mfcol=c(2,2))
hist(dat$GRIP, prob = TRUE, xlab = "Grip strength", main = "
    Histogram: Grip Strength")
curve(dnorm(x, m = sumdat["mean","GRIP"], s = sumdat["sd", "GRIP"])
    , from = range(dat$GRIP)[1], to = range(dat$GRIP)[2], add =
    TRUE)
hist(dat$ARM, prob = TRUE, xlab = "Arm strength", main = "Histogram
    : Arm Strength")
curve(dnorm(x, m = sumdat["mean","ARM"], s = sumdat["sd", "ARM"]),
    from = range(dat$ARM)[1], to = range(dat$ARM)[2], add = TRUE)
hist(dat$RATINGS, prob = TRUE, xlab = "Supervisor Rating", main = "
    Histogram: Rating")
curve(dnorm(x, m = sumdat["mean","RATINGS"], s = sumdat["sd", "
    RATINGS"]), from = range(dat$RATINGS)[1], to = range(dat$
    RATINGS)[2], add = TRUE)
hist(dat$SIMS, prob = TRUE, xlab="Sims Wrench Test", main="
    Histogram: Strength Simulation")
curve(dnorm(x, m = sumdat["mean","SIMS"], s = sumdat["sd", "SIMS"])
    , from = range(dat$SIMS)[1], to = range(dat$SIMS)[2], add =
    TRUE)
```

Could go further with that distribution analysis

- There's a warning in the help page for R's hist function.
- Previous plot not rigorous proof of Normality or non-Normality, just a visual depiction
- qqplot is suggested method of rigorously comparing a sample to a given probability model.
- A Chi-square or likelihood-based test would be even more rigorous

Scatterplot matrix OK for Small Datasets




```
plot(dat) ## That's same as pairs(dat)
```

Regress Ratings on Grip

```
mod1 <- lm (RATINGS ~ GRIP, data = dat)
summary(mod1)
```

```
Call:
lm(formula = RATINGS ~ GRIP, data = dat)

Residuals:
     Min       1Q   Median       3Q      Max
-18.6346  -6.5850   0.4132   6.0314  16.6298

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 33.72471    3.31870  10.162   <2e-16 ***
GRIP         0.06609    0.02944   2.245   0.0263 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 8.406 on 145 degrees of freedom
Multiple R^2:  0.03358,  Adjusted R^2:  0.02692
F-statistic: 5.039 on 1 and 145 DF,  p-value: 0.0263
```

Regression Table

	M1 Estimate (S.E.)
(Intercept)	33.725***
	(3.319)
GRIP	0.066*
	(0.029)
N	147
RMSE	8.406
R^2	0.034

$*p \leq 0.05 ** p \leq 0.01 *** p \leq 0.001$

- Estimated Intercept
- Estimated Slope
- Standard Error of Intercept Estimate (estimated standard deviation of intercept estimator)
- Standard Error of Slope Estimate (estimated standard deviation of slope estimator)

Hypothesis Test for Slope

- Theory: $RATINGS_i = c_0 + c_1 GRIP_i + u_i$
  c_0 and c_1 are real-valued constants, $E[u_i] = 0$, $Var[u_i] = E[u_i^2] = \sigma_u^2$.
- The Null Hypothesis: $H_0 : c_1 = 0$
- $\hat{c}_1$ is approximately Normal, So create T test:

$$\hat{t} = \frac{\hat{c}_1 - 0}{std.err.(\hat{c}_1)} = \frac{0.66}{0.029} = 2.245$$

- The critical value of t is:

```
> qt( c(0.025, 0.975), df=90)
[1] -1.986675  1.986675
```

- Conclusion: "The estimate $\hat{c}_1$ is statistically significantly different from 0."

Confidence Intervals for Intercept and Slope

```
confint(mod1)
```

```
                  2.5 %      97.5 %
(Intercept) 27.165443886 40.2839844
GRIP         0.007898305  0.1242813
```

Supposing the model's theory is correct, we believe

- the we believe with probability is 0.95 the true slope c_1 is in (0.0079, 0.125).
- the estimated slope $\hat{c}_1$ would be between 0.0079 and 0.125 in 95% of repeated samples from same process

Obtain Predicted Values

- predict returns one predicted value for each input row

```
mod1.predict <- predict(mod1)
head(mod1.predict, 10)
```

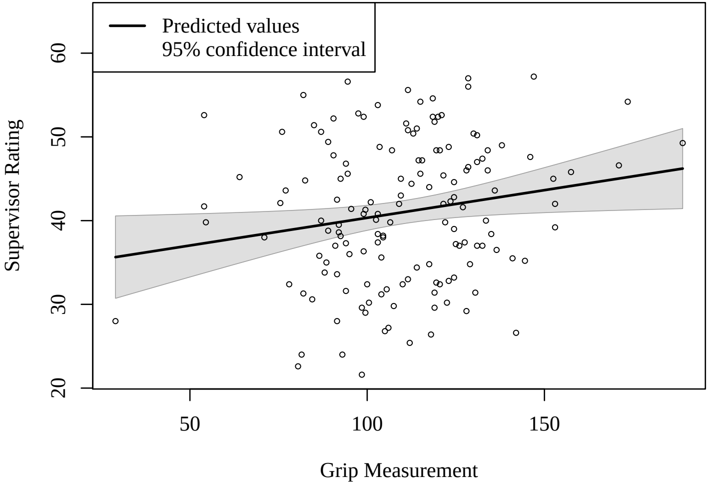
```
       1        2        3        4        5        6        7
              8        9       10
40.69719 40.76328 39.93715 39.70584 40.59805 45.02607 40.82937 41
    .95289 39.47453 40.49892
```

- Or ask for particular values by using a newdata argument

```
ndf2 <- data.frame(GRIP = plotSeq(dat$GRIP, 5))
ndf2$pred2 <- predict(mod1, ndf2)
ndf2
```

```
  GRIP     pred2
1   29  35.64132
2   69  38.28491
3  109  40.92850
4  149  43.57209
5  189  46.21569
```

Predicted value line overlaid on a scatterplot

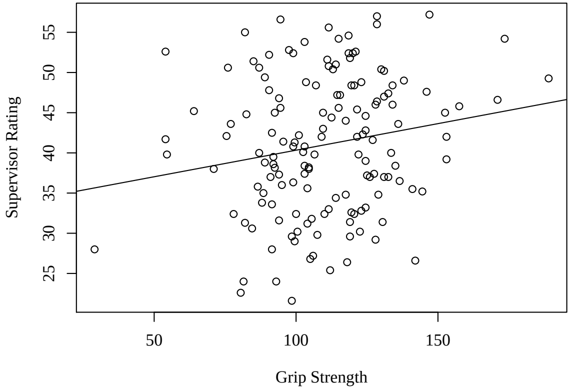



```
plotSlopes(mod1, plotx = "GRIP", xlab = "Grip Measurement", ylab =
    "Supervisor Rating", interval = "confidence")
```

I worry that's too easy for you

- You don't learn so much about the great Wild World of R if I predigest everything for you
- But I'll get nicer looking homeworks if I give you an easy way to make nice looking plots
- But I worry you'll never feel like a grown up in the R community if you only know how to speak baby talk
- So I've written out an explanation of how some of this gets done.

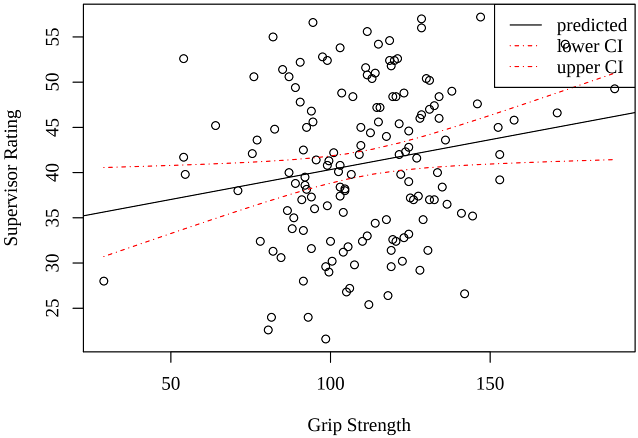
The usual Way in R is like this:




```
plot(RATINGS ~ GRIP, data = dat, xlab = "Grip Strength", ylab = "
    Supervisor Rating")
abline(mod1)
```

That exploits the multi-purpose power of abline to extract predicted values and plot them.

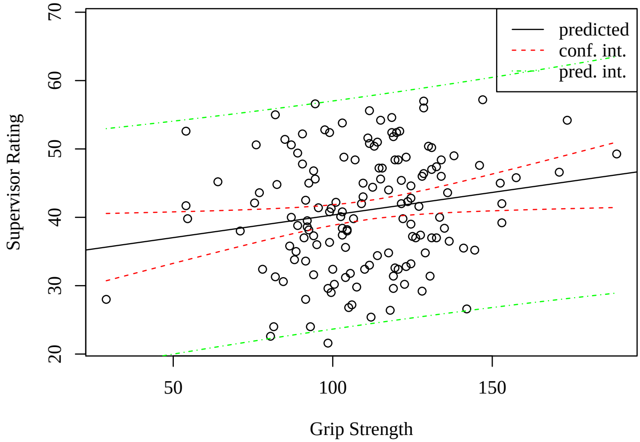
Superimpose Confidence Interval For Predicted Values



Code For Previous

```
plot(RATINGS ~ GRIP, data=dat, xlab="Grip Strength", ylab="
     Supervisor Rating")
abline(mod1)
newdf <- data.frame(GRIP=plotSeq(dat$GRIP, 20))
pconf <- predict(mod1, interval="confidence", newdata=newdf)
lines(newdf$GRIP, pconf[, "lwr"], lty=4, col="red")
lines(newdf$GRIP, pconf[, "upr"], lty=4, col="red")
legend("topright", legend=c("predicted","lower CI","upper CI"), lty
     =c(1,4,4), col=c("black", "red","red"))
```

Superimpose Confidence and Prediction Intervals



Code For Previous

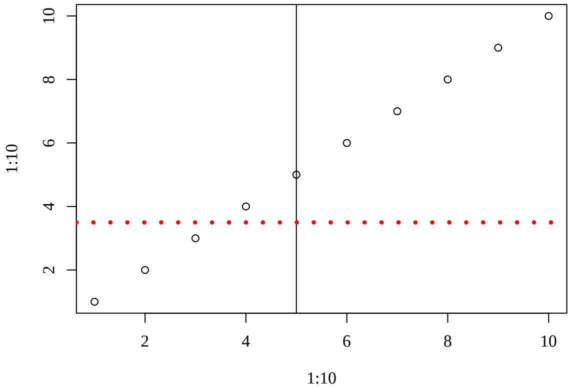
```
plot(RATINGS ~ GRIP, data=dat, xlab="Grip Strength", ylab="
    Supervisor Rating", ylim = c(1,1.2)*range(dat$RATINGS))
abline(mod1)
newdf <- data.frame(GRIP=plotSeq(dat$GRIP, 20))
pconf <- predict(mod1, interval="confidence", newdata=newdf)
lines(newdf$GRIP, pconf[ ,"lwr"], lty=2, col="red")
lines(newdf$GRIP, pconf[ ,"upr"], lty=2, col="red")
ppred <- predict(mod1, interval="prediction", newdata=newdf)
lines(newdf$GRIP, ppred[ ,"lwr"], lty=4, col="green")
lines(newdf$GRIP, ppred[ ,"upr"], lty=4, col="green")
legend("topright", legend=c("predicted","conf. int.","pred. int."),
      lty=c(1,2,4), col=c("black", "red","green"))
```

You learn a lot about R by studying abline()

- abline(v = 5) draws a vertical line where x = 5.
- abline(h = 3.5) draws a horizontal line where y = 3. 5.
- abline allows all of the customizations that lines allows, like col, lty, lwd
- The plot must exist first, however, before you run those functions
- Try

```
plot(1:10, 1:10)
abline(v = 5)
abline(h = 3.5, col = "red", lty = 3, lwd = 4)
```

2 ablines



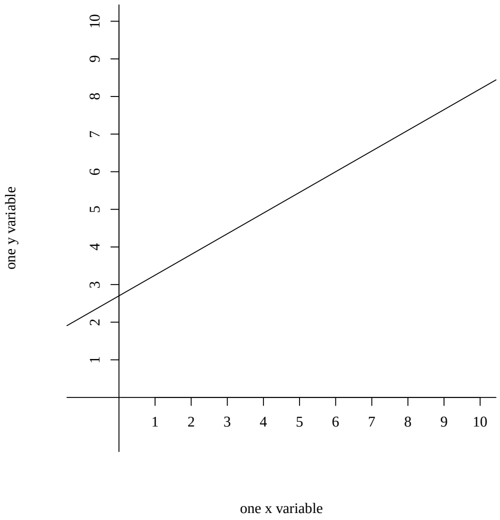
More about abline

- abline will also plot a line with a given intercept and slope, using arguments a and b. Try

```
plot(0:10, 0:10, type = "n")
abline(a = 2.7, b = 0.55)
```

That will be a bit disappointing because axes don't cross at 0, so I hammered on this a while to make a "mathbook style" plot

More about abline ...



More about abline ...

- Which you could draw by the seemingly tedious sequence

```
plot(-1:10, -1:10, type = "n", xlim = c(-1, 10), ylim = c(-1,10),
    axes = FALSE, bty = "n", xlab = "one x variable", ylab = "one y
    variable")
axis(1, pos = 0, at = seq(1, 10, by = 1))
axis(2, pos = 0, at = seq(1, 10, by = 1))
abline(a = 2.7, b = 0.55)
abline(v = 0)
abline(h = 0)
```

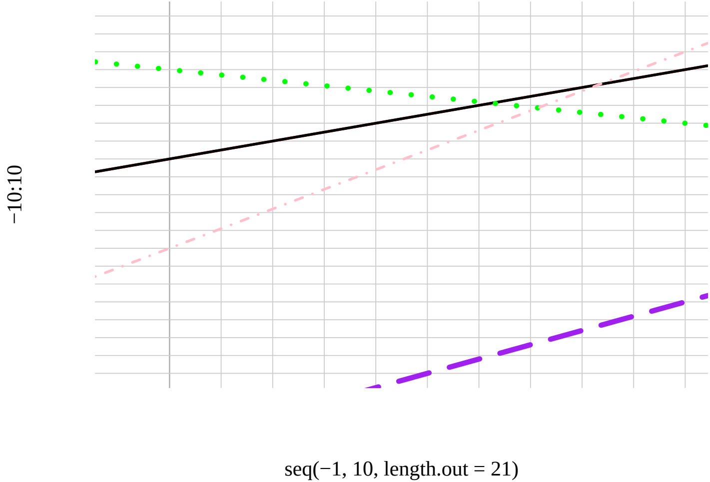
- Run "?abline" to read about it

Step through this, line-by line

This command creates a “blank plot” and fiddles with abline()

```
plot(seq(-1, 10, length.out = 21), -10:10, axes = FALSE, type = "n
    ")
## Make some phony graph paper
abline(v = 0, col= gray(.70)) ##get the idea?
abline(v = seq(1, 10), col = gray(.80), lwd = 0.7)
abline(h = seq(-10, 10), col = gray(.80), lwd = 0.7)
abline(a= 2, b = 0.5, col = "red", lwd = 2)
abline(coef = c(2, 0.5), col = "black", lwd = 2)
abline(coef = c(7, -0.3), col = "green", lwd = 4, lty = 3)
abline(coef = c(-3, 1.1), col = "pink", lwd =2, lty = 4)
abline(coef = c(-14, 0.8), col = "purple", lwd = 4, lty = 5)
```

Step through this, line-by line ...

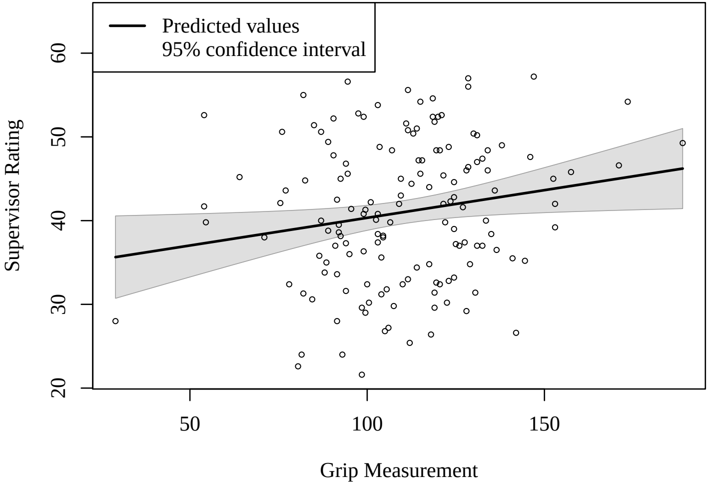


Step through this, line-by line ...

- Run "abline" without any parentheses and you'll see their code

And you realize the difficult part for the abline function is to examine the type of arguments you give it, because it has a lot of "if" "then" conditions to obey.

But, honestly, I'd use plotSlopes




```
plotSlopes(mod1, plotx = "GRIP", xlab = "Grip Measurement", ylab =
    "Supervisor Rating", interval = "confidence")
```