Regression in R

Fitting a Linear Regression Model

In these notes, we are going to re-analyze the data presented in Figure 12.4 from the Clark, Golder, and Golder (2013, pp. 477-478) reading. The necessary variables are in the data set `gamson.rds`. You might remember that the data frame `gamson` has two variables: `seat_share` and `portfolio_share`.

```
# load data
gamson <- readRDS("data/gamson.rds")
# note: make sure the file 'gamson.rds' is in the 'data' subdirectory
#   and your working directory is set appropriately.

# quick look at data
tibble::glimpse(gamson)
```

```
## Observations: 826
## Variables: 2
## $ seat_share      <dbl> 0.02424242, 0.46060607, 0.51515150, 0.47204968...
## $ portfolio_share <dbl> 0.09090909, 0.36363637, 0.54545456, 0.45454547...
```

This is a particularly useful data set on which to estimate a linear regression model. Remember that Gamson's Law states that "cabinet portfolios will be distributed among government parties in strict proportion to the number of seats that each party contributes to the government's legislative majority."

Gamson's Law implies that legislative seat shares should (according to the Law) relate to government portfolios linearly, with an intercept of zero and a slope of one. The slope should equal one because a one percentage point increase in seat share ought to lead to a one percentage point increase in portfolio share, at least on average. The intercept should equal zero because we expect a party with 0% of the legislative seats to get 0% of the government portfolios.

To fit a linear regression model, we use the `lm()` function. (The "l" stands for "linear" and the "m" stands for "model.")

The `lm()` function takes two arguments:

- The first argument to `lm()` is called a formula. For simple linear models with only one predictor, the formula has the form `y ~ x`, where `y` is the dependent variable (or outcome variable) and `x` is the independent variable (or explanatory variable or predictor). I usually put the formula first and leave the argument implicit.
- The second argument to `lm()` is the `data` argument. This will be the data frame in which to find `x` and `y`. I usually put this argument second and make it explicit.

In the case of the data frame `gamson`, our `y` is `portfolio_share`, our `x` is `seat_share`, and our `data` is `gamson`.

```
# fit linear model
fit <- lm(portfolio_share ~ seat_share, data = gamson)
```

Now we have used the `lm()` function to fit the model and stored the output in the object `fit`. So what type of object is `fit`? It turns out that `lm()` outputs a new type of object called a `list()` which is like a data frame, but more general. Whereas we think of a data frame as a box of equal-length vectors, we can think of lists as a box of objects. The objects in the list can be scalars, functions, vectors, data frames, or another list. This turns out to not be practically important to us, since we will not work directly with lists in this class.

Instead of working directly with `fit`, which is a list, we'll use other function to compute quantities that interest us.

Model Coefficients

First, we might care about the coefficients. In the case of a model with only one independent variable (explanatory variable or predictor), we'll just have an intercept and a slope. The `coef()` function takes one argument–the output of the `lm()` function, which we stored as the object `fit`–and returns the estimated coefficients.

```
coef(fit)
```

```
## (Intercept)  seat_share
##  0.06913558  0.79158398
```

We can see that Gamson's Law isn't exactly correct, a one-unit percentage point increase in vote share least to about a 0.8 unit increase in the portfolio share. Also, the intercept is slightly above zero, suggesting that smaller parties seem to have a disproportionate advantage.

R.M.S. Error

We might also be interested in the r.m.s. error of the model. The function `residuals()` works just like `coef()`, except it returns the model errors (i.e., "residuals") instead of the model coefficients. Surprisingly, there is no function that automatically calculates the r.m.s. in R (without loading a package), so we'll have to calculate it ourselves.

```
# one step at a time, r.m.s. error
errors <- residuals(fit)
s <- errors^2
m <- mean(s)
r <- sqrt(m)
print(r)  # r.m.s. error
```

```
## [1] 0.06880963
```

```
# all at once
sqrt(mean(residuals(fit)^2))  # r.m.s. error
```

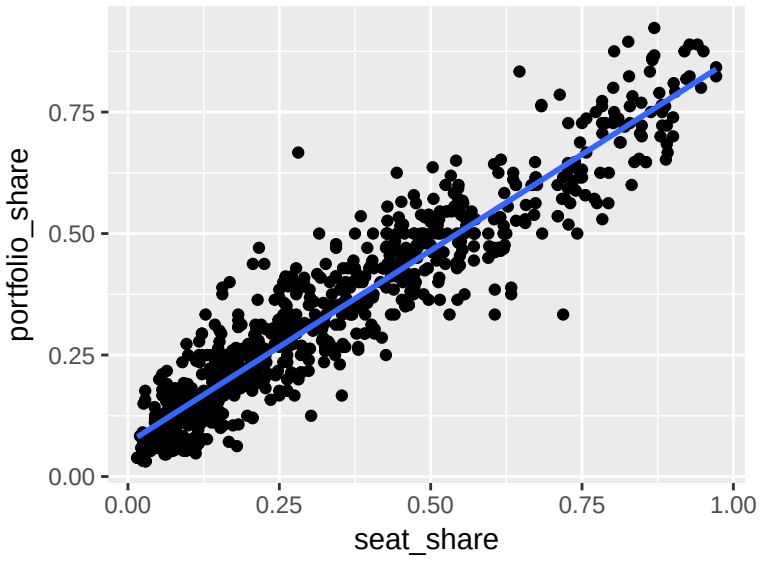
```
## [1] 0.06880963
```

For now, all we are really interested in is the coefficients and the r.m.s. error, so these two are all we need.

Fitting a Line in ggplot

Whenever we draw a scatterplot, we'll usually want to draw the regression line through the data. That is done by adding `geom_smooth()` to the plot. We'll usually want to supply two arguments to `geom_smooth()`. First, we'll want to set `method = "lm"` so that `geom_smooth()` fits a line through the data, as opposed to a smooth curve. Second, we'll want to set `se = FALSE` so that `geom_smooth()` does not include the standard error, which we haven't discussed yet (that's the last third of the class).

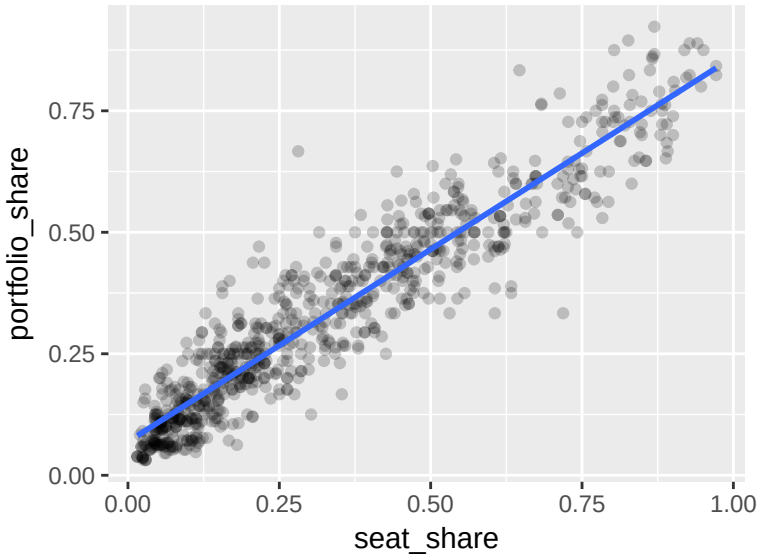
```
# load packages
library(ggplot2)

# create scatterplot with regression line
ggplot(gamson, aes(x = seat_share, y = portfolio_share)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



Alpha Transparency

Sometimes, when points overlap substantially, we need to make the points somewhat transparent. We have seen this idea for density plots that overlap as well. To make the points transparent, we simply supply the `alpha` argument to `geom_point()`. As before, you need to experiment with values of `alpha` between zero and one to find the appropriate level of transparency.

```
# create scatterplot
ggplot(gamson, aes(x = seat_share, y = portfolio_share)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "lm", se = FALSE)
```



Jittering

While it is not the case with these data, in many cases, data tend to take on only a few values. For example, number of children tends to be either 0, 1, 2, 3, or 4, with very few families having 5 or more children. Even variables like age make the scatterplot appear clumpy at the positive integers. These clumps, which appear

as columns and/or rows of points in the scatterplot, make the scatterplot difficult to interpret, since many points are positioned directly on top of one another.

To see this, let's load a new data set. This data sets contains several feeling thermometer ratings from the 2012 American National Election Study (ANES), which is a large, representative survey of Americans. In this in-person survey, the interviewer asks many questions. For the feeling thermometers, the surveyor asks the respondent to rate individuals and groups on a scale from 0 degrees to 100 degrees, with the following scale in hand.

Figure 1: Scale for ANES feeling thermometer.

```
getwd()
```

```
## [1] "/Users/carlislerainey/Dropbox/classes/pols-209"
```

```
# load data
therms <- read.csv("data/therms.csv")
# note: make sure the file 'therms.rds' is in the 'data' subdirectory
#   and your working directory is set appropriately.

# quick look at data
tibble::glimpse(therms)
```

```
## Observations: 5,914
## Variables: 31
## $ barack_obama              <dbl> NA, 100, 100, 100, 100, 100, 100, 10...
## $ mitt_romney               <dbl> NA, 0, 0, 0, 50, 30, 15, 60, 40, 60,...
## $ ann_romney                <dbl> NA, 15, 0, 0, 50, 30, 15, 70, 50, 50...
## $ michelle_obama            <dbl> NA, 100, 100, 100, 100, 100, 100, 10...
## $ joe_biden                 <dbl> NA, 95, 100, 0, 85, 75, 100, 85, 50,...
## $ paul_ryan                 <dbl> NA, 0, 0, 0, 60, 50, 30, 70, 50, 60,...
## $ hillary_clinton           <dbl> 100, 100, 100, 100, 70, 78, 100, 100...
## $ george_w_bush             <dbl> 0, 0, 15, 0, 30, NA, 15, 0, 30, 40, ...
## $ john_roberts              <dbl> NA, 100, 60, 0, 50, 60, 40, 100, 50,...
## $ christian_fundamentalists <dbl> NA, 60, NA, 0, 40, 45, 70, 85, 50, 5...
```

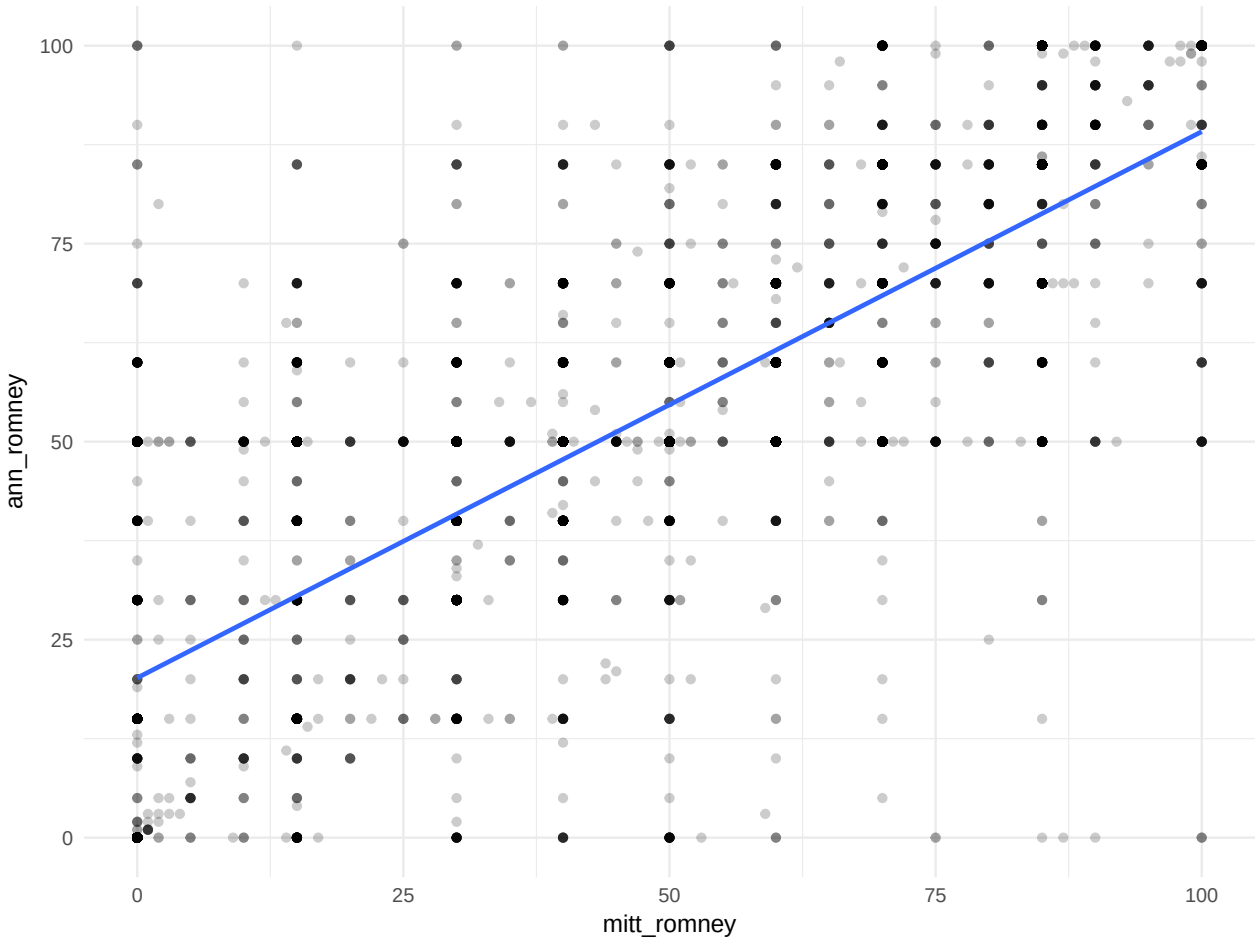
```
## $ catholics                <dbl> NA, 70, 100, 85, 85, 35, 70, 100, 50...
## $ feminists                <dbl> NA, 70, 60, 85, 85, 35, 60, 85, 50, ...
## $ federal_government       <dbl> NA, 85, 100, 60, 70, 25, 85, 100, 40...
## $ liberals                 <dbl> NA, 85, 60, 100, 85, 50, 50, 70, 50,...
## $ middle_class_people      <dbl> NA, 70, 100, 100, 100, 50, 70, 100, ...
## $ unions                   <dbl> NA, 84, 85, 85, 60, 45, 70, 85, 70, ...
## $ poor_people              <dbl> NA, 70, 100, 100, 60, 75, 30, 85, 50...
## $ military                 <dbl> NA, 60, 100, 0, 100, 35, 100, 85, 70...
## $ big_business             <dbl> NA, 40, 70, 100, 60, 45, 85, 100, 40...
## $ welfare_recipients       <dbl> NA, 70, 70, 100, 40, 35, 30, 85, 50,...
## $ conservatives            <dbl> NA, 30, 70, 85, 40, 20, 70, 85, 40, ...
## $ working_class            <dbl> NA, 70, 100, 100, 100, 65, 85, 85, 6...
## $ supreme_court            <dbl> NA, 70, 85, 60, 70, 25, 60, 100, 50,...
## $ gays_and_lesbians        <dbl> NA, 85, 50, 0, 70, 50, 85, 70, 50, 5...
## $ congress                 <dbl> NA, 15, 85, 60, 40, 25, 50, 85, 50, ...
## $ rich_people              <dbl> NA, 0, 40, 60, 70, 35, 15, 100, 50, ...
## $ muslims                  <dbl> NA, 85, 60, 60, 60, 35, 70, 70, 50, ...
## $ christians               <dbl> NA, 70, 100, 85, 85, 25, 40, 85, 100...
## $ atheists                 <dbl> NA, 70, 0, 0, 40, 50, 30, 70, 40, 50...
## $ mormons                  <dbl> NA, 70, 40, 85, 50, 30, 15, 85, 50, ...
## $ tea_party                <dbl> NA, 0, 0, 100, 30, 45, 30, 100, 50, ...
```

As you can see, the variable names in these data refer to the individual or group the surveyor asks the respondent to rate.

One of the most interesting patterns in these data is the relationship between the feeling thermometers for Republican presidential candidate Mitt Romney and his wife Ann Romney.

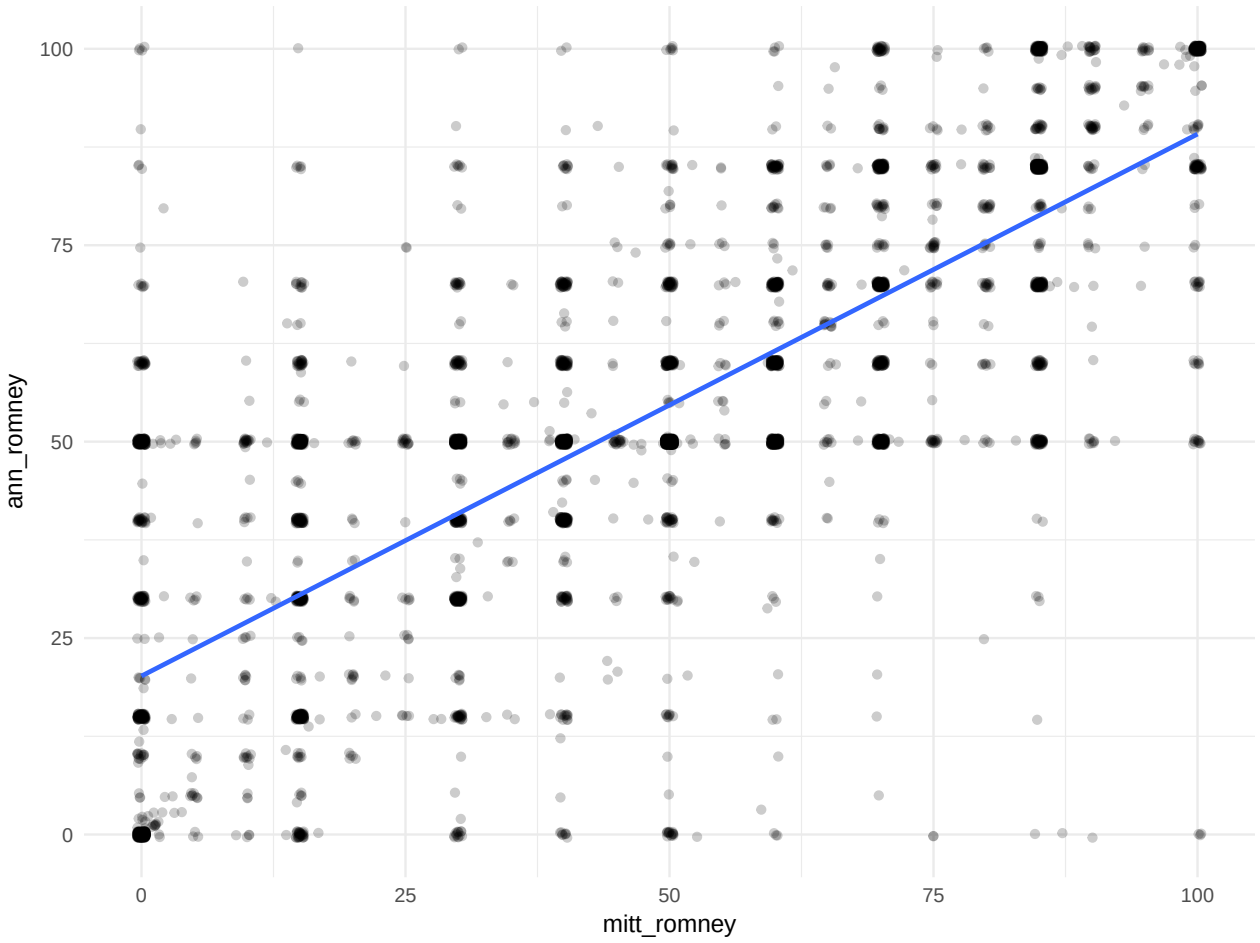
In these data, you can see a clear clumping at points like 50-50. Notice that I've used a small value of `alpha = 0.2` and a theme with a white background to make the transparency as effective as possible, yet so many points fall on top of each other in some locations that the points show up as completely black.

```
# create scatterplot
ggplot(therms, aes(x = mitt_romney, y = ann_romney)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal()
```



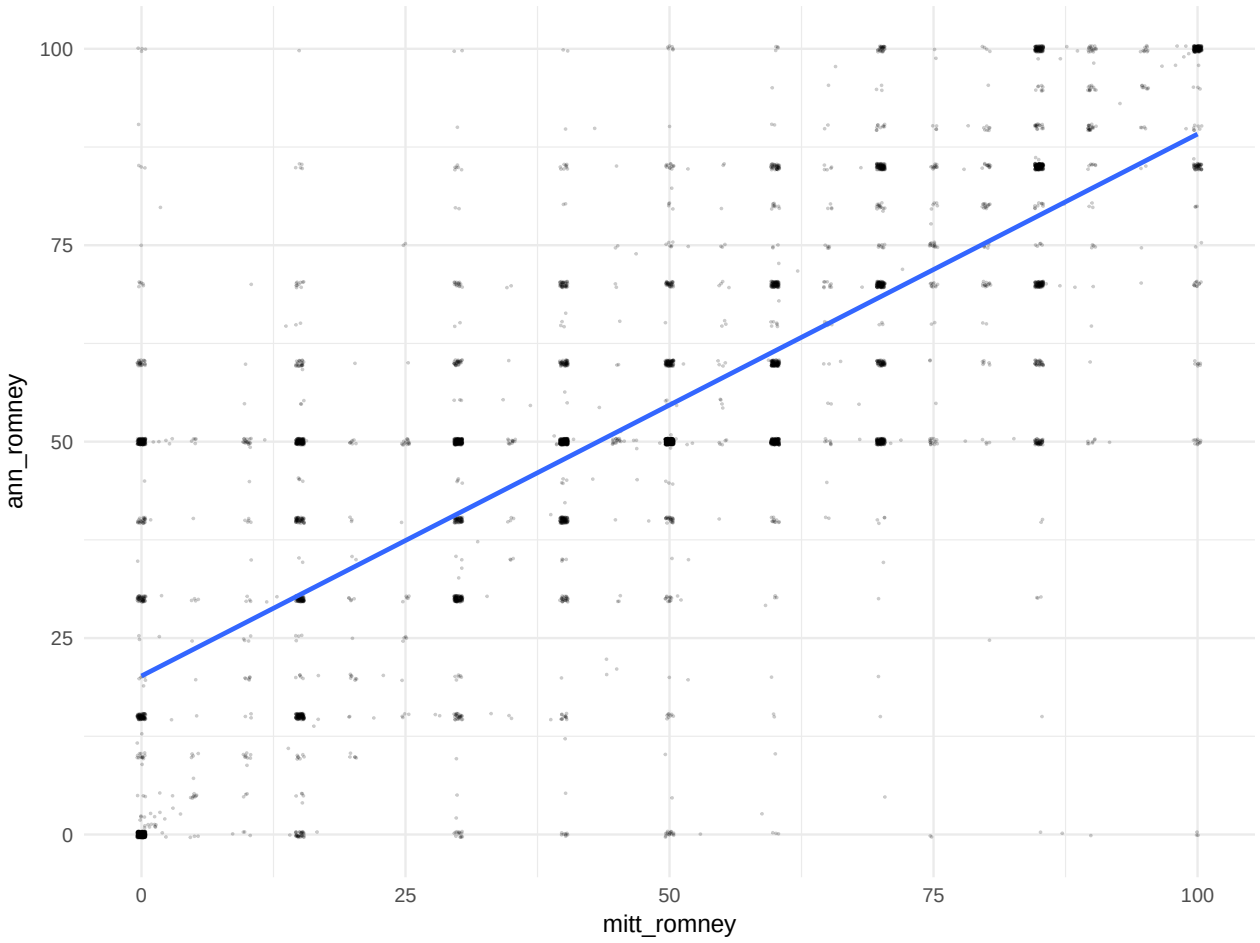
Because we use alpha transparency in coloring the points, we can see the substantial overlap in the points. We can a tiny bit of random noise to the vertical and horizontal location to the points to so that each point can be seen more easily. We do this be supplying the argument `position = "jitter"` to the function `geom_point()`. Note that we can use alpha transparency and jittering in combination.

```
# create scatterplot
ggplot(therms, aes(x = mitt_romney, y = ann_romney)) +
  geom_point(alpha = 0.2, position = "jitter") +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal()
```



To make the jittering more effective, we can make the points smaller by supplying a `size = 0.1` argument to `geom_point()`.

```
# create scatterplot
ggplot(therms, aes(x = mitt_romney, y = ann_romney)) +
  geom_point(alpha = 0.2, position = "jitter", size = 0.1) +
  geom_smooth(method = "lm", se = FALSE) +
  theme_minimal()
```



Prediction

One important use of the linear regression model is prediction. For prediction problems, there are typically three data sets.

1. A **training set**. The researcher uses this data set to fit the model. In our case, this will be a linear regression model.
2. A **prediction set**. This data set contains the predictors, but not the outcome, for several new cases. This might be new people, states, or years. Based on the variables in this data set, the researcher produces a prediction.
3. A **test set**. The same as the prediction set, but with the outcome variable included. The researcher uses the test set to evaluate her modeling approach.

The taxes data have a training set and a prediction set. For these data, the outcome we're trying to predict is `tax_change` which is the legislated changes in tax revenue (in billions) for the 50 U.S. states from 1988 to 2009. The prediction set contains the 50 U.S. states for 2010 and 2011.

For our purposes, a good predictive model minimizes the r.m.s. error of the predictions in the prediction set (although we can't know the r.m.s. in the prediction set–there's no outcome to compare our predictions to. After all, it wouldn't be a prediction otherwise.)

```
# load data
training_set <- readRDS("data/taxes-training.rds")
prediction_set <- readRDS("data/taxes-prediction.rds")
```

```
# quick look at data
tibble::glimpse(training_set)
```

```
## Observations: 1,055
## Variables: 21
## $ state                           <chr> "Alabama", "Alabama", "Alabama"...
## $ state_abbr                      <fctr> AL, AL, AL, AL, AL, AL, AL, AL...
## $ year                            <int> 1988, 1989, 1990, 1991, 1992, 1...
## $ tax_change                      <dbl> 0, 0, 47, 22, 100, 0, 0, 0, 0, ...
## $ personal_income                 <dbl> 0.060, 0.063, 0.067, 0.072, 0.0...
## $ percent_change_personal_income <dbl> 7.84, 9.09, 5.00, 6.35, 7.46, 4...
## $ estimated_imbalance             <int> 8, 67, -3, 0, 32, 119, 0, 43, 4...
## $ taxes_last_year                 <dbl> 3.4, 3.7, 3.8, 3.9, 4.2, 4.6, 4...
## $ gov_request                     <dbl> 0, 0, 55, 0, 520, 0, 0, 0, 0, 0...
## $ lag_tax_change                  <dbl> 0.0, 0.0, -4.8, 0.0, 0.0, 0.0, ...
## $ population                      <dbl> 4.000000, 4.100000, 4.100000, 4...
## $ percent_change_population       <dbl> 0.00, 0.00, 2.50, 0.00, 2.44, 0...
## $ gov_party                       <fctr> Republican, Republican, Republ...
## $ house_dem_share                 <dbl> 0.8476190, 0.8333333, 0.8333333...
## $ senate_dem_share                <dbl> 0.8571429, 0.8235294, 0.8235294...
## $ citizen_ideology                <dbl> 45.02993, 37.47248, 33.83535, 3...
## $ year_of_biennium                <fctr> Second Year of Biennium, First...
## $ change_in_gov_party             <fctr> No Change, No Change, No Chang...
## $ change_in_house_dem_share       <dbl> 0.000000000, -0.014285716, 0.00...
## $ change_in_senate_dem_share      <dbl> 0.00000000, -0.03361351, 0.0000...
## $ change_in_citizen_ideology      <dbl> 6.51300, -7.55745, -3.63713, 5....
```

Two variables stand out as especially important. The first is `gov_request`, which is the amount of new taxes requested by the governor. The second is `estimated_imbalance`, which is the estimated budget imbalance for that year. Let's use each of these variables and fit two linear models.

```
fit1 <- lm(tax_change ~ gov_request, data = training_set)
fit2 <- lm(tax_change ~ estimated_imbalance, data = training_set)
```

Now here is the important question: How do we know which of these models will best predict the years 2010 and 2011, which are not in this data set, but in the prediction set. Here is the answer: we don't. We have to rely on a good theory or model of the budget making process. However, we can try to guess how well out model will predict out-of-sample cases (in this situation, 2010 and 2011) based on how well it predicts in-sample cases (in this situation, 1988-2009).

R.M.S. Error

First, we can look at the r.m.s. error for each regression. Of course, a lower r.m.s. error is prefered. However, a mdel that becomes too complex might fit the in-sample data extremely well, but out-of-sample data quite poorly. So bias yourself toward simpler models and models that make theoretical sense.

```
sqrt(mean(residuals(fit1)^2))  # r.m.s. error for fit1
```

```
## [1] 383.471
```

```
sqrt(mean(residuals(fit2)^2))  # r.m.s. error for fit2
```

```
## [1] 508.0815
```

BIC

We can also use a model summary called the BIC (Bayesian Information Criterion) to guess which model will predict out-of-sample data best. We can calculate the BIC for multiple models using the `BIC()` function and supplying multiple `lm()` outputs as unnamed arguments. Lower values of the BIC indicate a better model.

```
BIC(fit1, fit2)
```

```
##      df      BIC
## fit1  3 15567.79
## fit2  3 16161.50
```

Prediction

Because both the r.m.s. error and BIC suggest that `fit1` is the better model, we might want to use that model to make predictions for the prediction set. For this, we can use the `predict()` function.

The predict function takes two arguments:

1. The first argument is the output of `lm()` for the model you want to use to make predictions. This argument is usually unnamed.
2. The second argument `newdata`, which is the data frame for which you'd like to make predictions. This data frame should include the predictors used the the model (the variable(s) on the right-hand side of the `~`), but it does not need to include the outcome variable. If this argument is not included, then `predict()` makes predictions for the data set used to fit the model.

By default, we store the predictions a new variable in the prediction data set.

```
prediction_set$prediction <- predict(fit1, newdata = prediction_set)
```

Are these predictions any good? We don't know. In order to find out, we'd have to compare the predictions to the actual values, which I have safely hidden away on my computer.

Review Exercises

1. Explain what the `lm()` function does and how it works. In particular, what is the first argument to `lm()`? The second? What does `lm()` output (or return)?
2. What does the function `coef()` do? In particular, what is the first (and only) argument to `coef()`? What does it output?
3. What does the function `residuals()` do? In particular, what is the first (and only) argument to `residuals()`? What does it output? How can you use the output to calculate the r.m.s. error of the regression?
4. What function do you use to add a linear regression line to a scatterplot? What arguments do we typically supply and why?
5. Explain how to adjust the alpha transparency of the points in a scatterplot. Explain how to jitter the points.
6. What two model summaries can we use to guess the predictive ability of the model? How do we calculate each in R?
7. What does the function `predict()` do? In particular, what is the first argument to `predict()`? The second? What does it output? How do you store the output of `predict()` as a variable in the prediction set?