Please note that this book is a work in progress, the equivalent of a pre-alpha release. All the code should work, because if it didn't the site could not be built. But there is still a lot of work to do to explain the historical methods under discussion. Feel free to leave feedback as issues on the GitHub repository, or to e-mail me.

Statistics

Basic statistical techniques of interest to historians.

Yes, we'll start with `mean()` and `sd()` and `hist()`.

TODO: Measures of the center, such as mean and median.

TODO: Measures of variation from the center.

Regression

One important statistical technique of use to historians is regression analysis. Regression analysis tries to quantify the relationship between two variables. For example, one might ask whether there is a relationship between height and weight (there is: taller people tend to be heavier). Not only does regression analysis attempt to tell us whether there is a relationship between variables: it also tries to tell us how accurate and how reliable that estimate is. For a historian, a particular interesting relationship to quantify is change over time: that is, does, a change in time correlate with a with a change in some other variable. For instance, as time goes on does the population of the United State increase? (Yes.) But we need not limit our inquiry just to change over time. We could check if there is a relationship betweena any two variables. For instance, suppose we have a set of data that includes the number of attendees at a Catholic mission and the number of people who converted to Catholicism. As the meeting gets bigger in terms of attendees, does the number of converts

increase? (It doesn't seem to: if anything, missions with fewer Catholics in attendance made more converts, since there were more non-Catholics to convert.)

In regression analysis we distinguish between independent variables and dependent variables. Independent variables are the variables that we think might be the causes; dependent variables are the variables that we think might be the effect. It is also important to point out that regression analysis can really only show correlation, not cause. For example, change over time does not really change population, and it makes even less sense to say that change in population causes a change in time. Historians must think carefully about which variables are independent and which are dependent: such questions must usually be settled by recourse to other kinds of evidence.

There are many kinds of regression analysis. If you've heard of any kind of regression analysis you've probably heard of linear regression. That kind of regression (shown below) assumes that the relationship between variables takes the form of a straight line. Other kinds of regression analysis, which assume some kind of exponential change or cyclical change, might be more appropriate to any given historical question. There are many more details about regression. This section will show you what it looks like to fit a linear regression visually to a small data set, then it will show you how to apply that model to a very large data set.

One question of interest to historians is how naming practices change. We will use a data set collected by the Social Security Administration in the United States, which includes names and birth years for all names used more than five times in a given year. This data can be found in Hadley Wickham's babynames package, which you can install with the command `install.packages(babynames)`. When we load the data set along with dplyr to manipulate the data, we can get a sense of what the data looks like.

```
library(dplyr)
```

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

```
##
##     filter
##
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

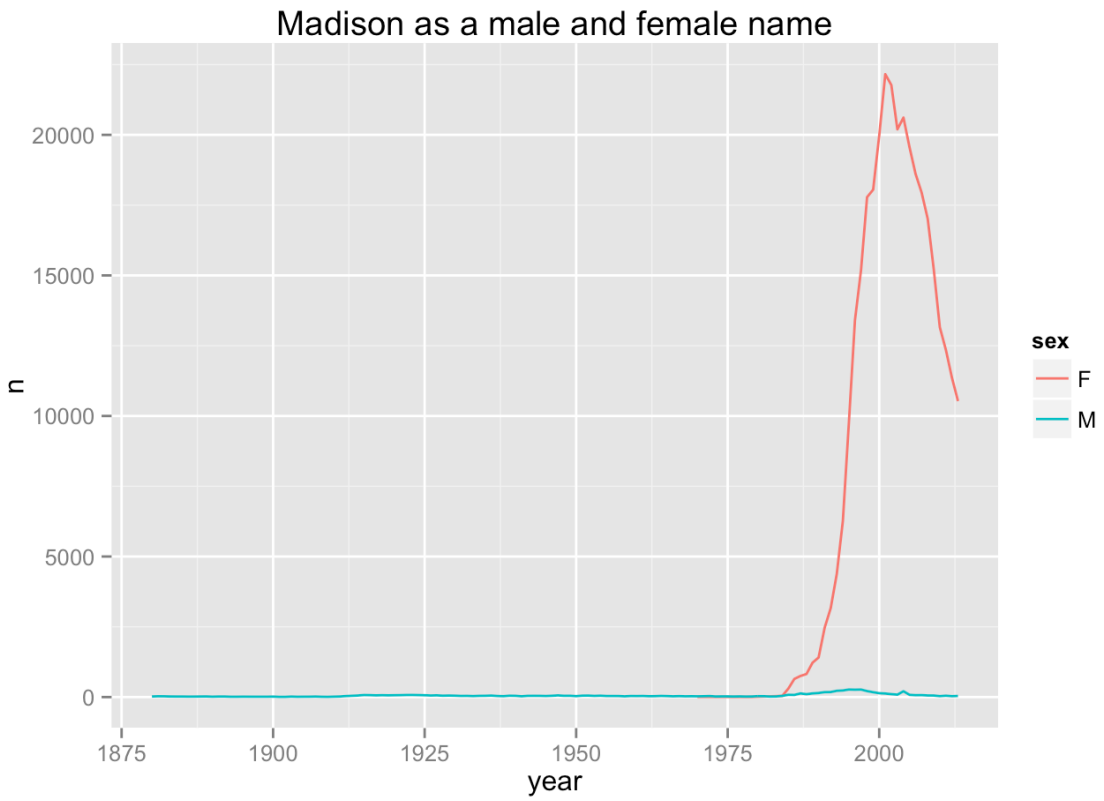
```
library(babynames)

babynames %>% head()
```

```
## Source: local data frame [6 x 5]
##
##   year sex      name    n       prop
## 1 1880   F      Mary 7065 0.07238359
## 2 1880   F      Anna 2604 0.02667896
## 3 1880   F      Emma 2003 0.02052149
## 4 1880   F Elizabeth 1939 0.01986579
## 5 1880   F    Minnie 1746 0.01788843
## 6 1880   F  Margaret 1578 0.01616720
```

For each combination of name, year, and sex, we have the value `n`, which is the number of people who received that name, and the values `prop` which is the proportion of people who received that name that year. We can begin with a simple plot of the name `Madison`, which shows us how the name changed over time:

```
library(ggplot2)
babynames %>%
  filter(name == "Madison") %>%
  ggplot(aes(x = year, y = n, color = sex)) + geom_line() + ggtitle("Mad
```



Already we notice something interesting. Before about 1985, Madison seems to have been an fairly unpopular name used exclusively for boys. From 1985 to 2001 Madison became a much more popular name used almost exclusively for girls. Were we to calculate the proportion of male and female uses of the name in a given year, we would find that the proportion of male uses was about `1` until almost the 1980s, and the proportion of female uses of the names was very close to `1` after that date. As historians, we would like to know: what other names changed? Did they tent to change from male to female or female to male? Did any names change in both directions? Did the proportion change because a name became more popular?

As in any data manipulation task, we need to think about what form of the data would be most useful to us. We might decide that we are most interested in the proportion of female uses to total uses of a name in any given year. We might just as well have chosen the proportion of male uses, since `1 = proportion_female + proportion_male`. We will use dplyr to summarize the data for that purpose. Since I happen to know that, because of the reasons people had to apply for Social Security during the New Deal, the gender ratios for the data set are unreliable before about 1940, we will only consider those years even though the data goes back to 1880. (We could correct this skew, but that is another problem.)

```
library(tidyr)

ratios <- babynames %>%
  filter(year >= 1940) %>%           # exclude years we don't want
  select(-prop) %>%                  # we don't need the prop column
  group_by(name, year) %>%           # find combinations of name and yea
  mutate(total_n = sum(n)) %>%       # sum both male and female uses
  group_by(name, sex, year) %>%      # include sex in grouping
  mutate(ratio = n / total_n) %>%    # calculate proportions of male/fem
  select(name, year, sex, ratio) %>% # only keep the columns we need
  ungroup() %>%
  spread(sex, ratio)                 # spread the data into M and F colu
```

Spreading the data with the `spread()` function from the tidyr shows a problem in our data. In years where there were no male or female uses of a given name, we have `NA` values. We can safely assume that in those years the proportion was actually `0`, so will replace all such values.

```
ratios[is.na(ratios)] <- 0
```

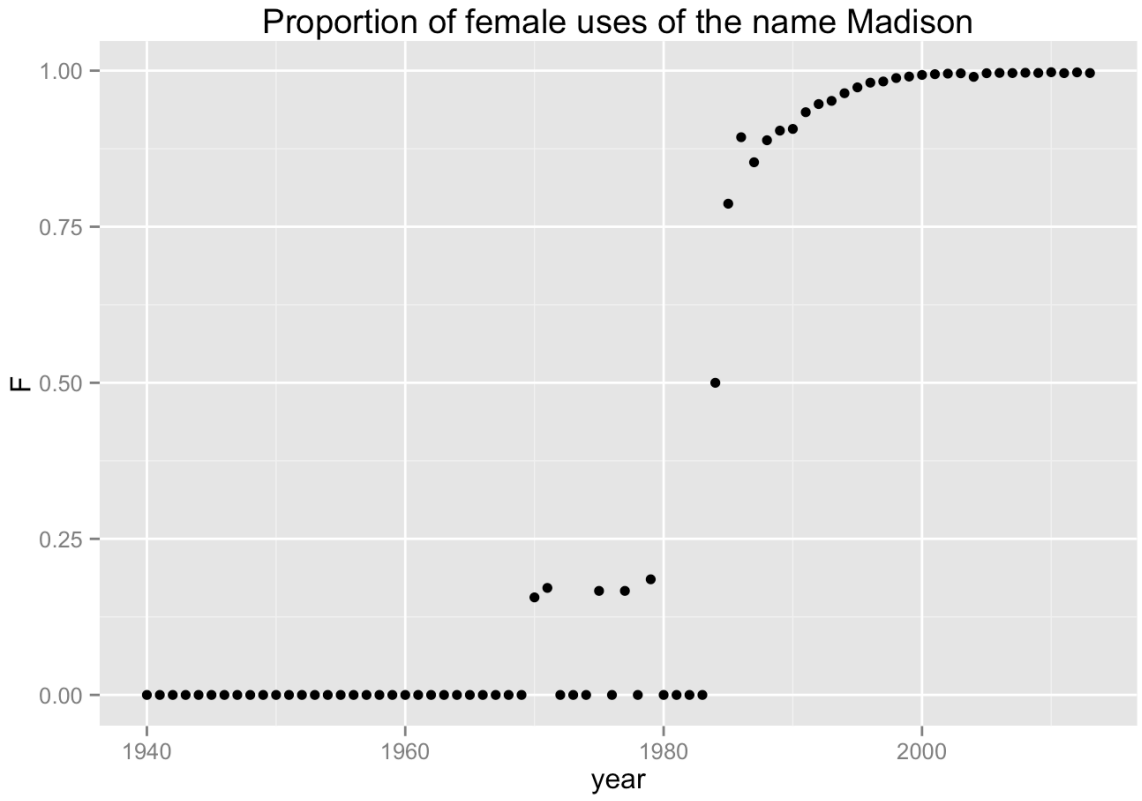
If we look at our results for the period around 1985, we can see the change shift in the proportion of female (`F`) and proportion of male (`M`) uses of the name Madison, and we can also plot that change for the entire dataset.

```
ratios %>% filter(name == "Madison", year >= 1980, year <= 1990)
```

```
## Source: local data frame [11 x 4]
##
##       name year         F          M
## 1  Madison 1980 0.0000000 1.00000000
## 2  Madison 1981 0.0000000 1.00000000
## 3  Madison 1982 0.0000000 1.00000000
## 4  Madison 1983 0.0000000 1.00000000
## 5  Madison 1984 0.5000000 0.50000000
## 6  Madison 1985 0.7868421 0.21315789
## 7  Madison 1986 0.8933518 0.10664820
## 8  Madison 1987 0.8532423 0.14675768
```

```
## 9  Madison 1988 0.8885281 0.11147186
## 10 Madison 1989 0.9039172 0.09608278
## 11 Madison 1990 0.9066323 0.09336768
```

```
ratios %>% filter(name == "Madison") %>%
  ggplot(aes(x = year, y = F)) + geom_point() +
  ggtitle("Proportion of female uses of the name Madison")
```



We can now use ggplot2, which will in turn use base R's built-in `lm()` function, to plot a linear regression line for this change.

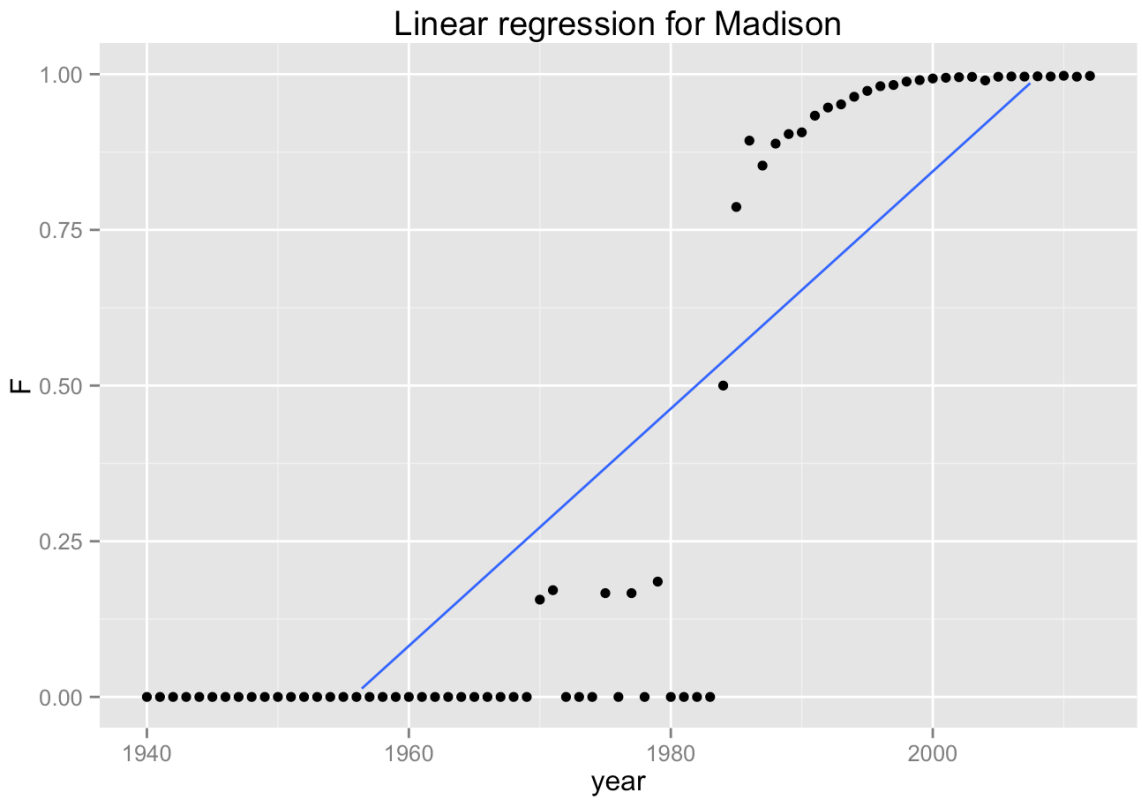
```
ratios %>% filter(name == "Madison") %>%
  ggplot(aes(x = year, y = F)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  xlim(1940, 2012) + ylim(0, 1) +
  ggtitle("Linear regression for Madison")
```

```
## Warning in loop_apply(n, do.ply): Removed 1 rows containing missing va
```

```
## (stat_smooth).
```

```
## Warning in loop_apply(n, do.ply): Removed 1 rows containing missing va
## (geom_point).
```

```
## Warning in loop_apply(n, do.ply): Removed 23 rows containing missing
## values (geom_path).
```



The blue line in that plot is the regression line. You can think of a linear regression line as the single straight line which best connects a series of dots. (I'll explain the mathematical definition later.) In the case of the name Madison, the line can only come close to capturing the change. But the slope of the line does tell us as time goes on the proportion of female uses of the name Madison rose. Figuring out where on the x-axis the regression line passes the `0.5` on the y-axis is a decent indicator of when the name switched from being mostly male to mostly female. Figuring out where on the x-axis the regression line passes `0` and `1` on the y-axis is a less good indicator of when a name is

almost totally male or female.

We can see how regression analysis helps us when we apply it more than one name. First we need to learn a way to find out if any given value is in a list of values that we define. If we define a character vector of names that we want to test, we can use R's `%in%` operator to find if any given string is in that vector. (The kind of operator whose name is enclosed between parentheses is called a binary operator, because it takes a value on both its left-hand and right-hand side. You are already familiar with binary operators like `+` and `%>%`.)

```
test_names <- c("Madison", "John", "Mary", "Leslie", "Jan")
"Madison" %in% test_names
```

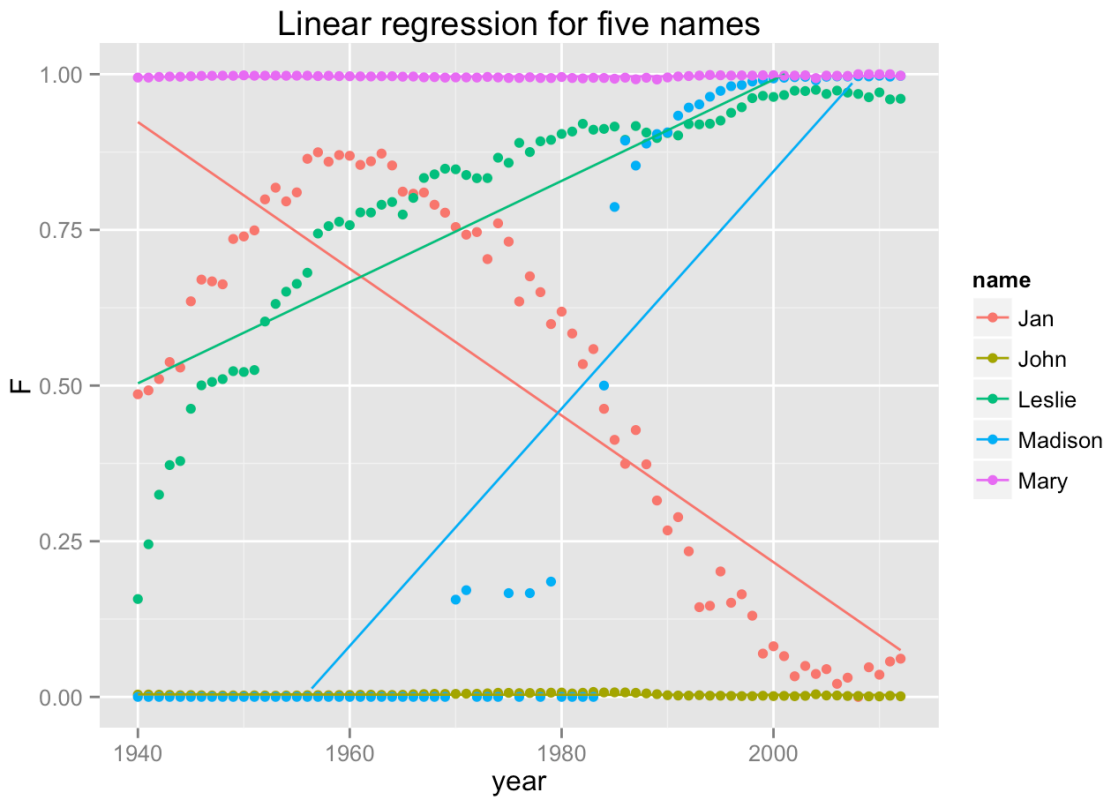
```
## [1] TRUE
```

```
"Henry" %in% test_names
```

```
## [1] FALSE
```

Now we can create another plot using the same technique of creating a regression line for the five names we defined above.

```
ratios %>% filter(name %in% test_names) %>%
  ggplot(aes(x = year, y = F, color = name)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  xlim(1940, 2012) + ylim(0, 1) +
  ggtitle("Linear regression for five names")
```



This is a busy plot, but we can learn interesting things about these five names. The name Mary never changes from being entirely female, and the name John never changes from being entirely male. The lines for both of those names are thus flat; the line for John is at `0` on the chart and the line for Mary is flat at `1` on the chart. The name Jan has an opposite trajectory from Madison and Leslie; its slope is negative. (We also notice that the linear regression loses the detail that from 1940 to 1960 Jan became more popular as a female name, before becoming mostly a male name since 2000.) Both Madison and Leslie have positive slopes, indicating that they became female names, Leslie has a shallower slope, indicating that it became a female name earlier and less quickly than Madison.

What we have done is create a mathematical model of changes in naming practices. The linear regression line tells us roughly whether a name changed genders, and if so in which direction, when, and how quickly it changed. Unfortunately we are a sticking point. Our previous plot was busy enough with five names. We could only squeeze a few more names on the chart before it is too confusing. But there are 92,600 unique names in the Social Security names data set. We could never meaningfully look through that many plots to locate the names of interest. What we need is a way of moving past visualization to modeling.

Modeling

The visualization cannot be extended to those tens of thousands of names, but it doesn't need to be. The visualization is based on a mathematical model. Specfically, it is was calculated using the `lm()` linear-model fitting function. Although we cannot make ninety thousand plus visualizations, we can fit the linear model ninety thousand times. To borrow a point made by Hadley Wickham, a visualization "surprises, but doesn't scale", while a model "scales, but doesn't (fundamentally) surprise." In other words, once we have figured out a model using visualizations, we can then apply the model, and so on in an iterative cycle.

First let's use the `lm()` function to fit a model just for the name Madison, so that we can understand how it works. Then we will explain the mathematics behind the modeling. Then we can apply the model to our list of names.

The `lm()` function takes two main arguments. The second argument is the data frame with the set of observations to be modeled. Let's filter our `ratios` data frame to just the name Madison.

```
madison <- ratios %>% filter(name == "Madison")
```

The first argument is the formula which expresses what we think is the relationship between the variables in the data. R has a built in class of objects called formulas which use the `~` operator. In our case, our formula will be `F ~ year`. Both `F` and `year` are variables in our data frame. The dependent variable (the variable that we think is the effect) goes on the left, and the independent variable (the variable that we think is the "cause") goes on the right. If we had multiple independent or dependent variables, we would join them with `+`, for example, `F ~ year + population`. But you shouldn't think of the `+` operator as indicating arithmetical addition; in this case it merely shows that both variables are independent.

Putting those two arguments together, we can make a fit a linear model to our Madison data.

```
madison_model <- lm(F ~ year, madison)
madison_model
```

```
##
## Call:
## lm(formula = F ~ year, data = madison)
##
## Coefficients:
## (Intercept)         year
##   -37.05580      0.01895
```

The basic information printed by `madison_model` tells us the equation for the regression line. Recall from the last algebra class that you took that the equation for a straight line is $y = mx + b$, where y and x are the dependendent and independent variables respectively, and m is the slope and b is the y-intercept (the value of y when $x = 0$. In other words, our model for Madison yield a regression line with the equation $F = 0.0189(year) - 37.0558$.

There is much more information available in our model. We can use the `str()` function to find out everything that is available and the base R `plot()` function will show some useful plots, but the `summary()` function will give us a good overview.

```
summary(madison_model)
```

```
##
## Call:
## lm(formula = F ~ year, data = madison)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.51811 -0.13442  0.02734  0.19591  0.31839
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -37.055803   2.409095  -15.38   <2e-16 ***
## year          0.018948   0.001219   15.55   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2239 on 72 degrees of freedom
```

```
## Multiple R-squared:  0.7705, Adjusted R-squared:  0.7673
## F-statistic: 241.7 on 1 and 72 DF,  p-value: < 2.2e-16
```

TODO: description of mathematics of linear models.

So far we have not mentioned certainty. In calculating a regression line, we can also calculate how well that line fits the data. Not only can we figure out what our model tells us about our data, we can also get an indication of how well the model works for our data.

Now that we understand the model and its possibilities, we need to apply this model to all the names in the data set. Actually, for the sake of this book, we're going to apply it just to 1000 names (plus the five we used above) chosen at random, because on my relatively fast laptop applying the model to all ninety-two thousand names takes about ten minutes. But instead of creating the sample data frame as below, you could apply the same code to the original `ratios` data frame. (See also `?sample_n` and `?sample_frac` in the dplyr package for alternative ways of sampling a data set.)

```
set.seed(3453)
unique_names <- ratios$name %>% unique
sample_names <- sample(unique_names, 1000)
sample_names <- c(sample_names, test_names)

names_to_fit <- ratios %>% filter(name %in% sample_names)
```

Now we use the `do()` function from dplyr to apply the `lm()` function to our smaller data frame.

```
names_models <- names_to_fit %>%
  group_by(name) %>%
  do(model = lm(F ~ year, data = .))
names_models
```

```
## Source: local data frame [1,005 x 2]
## Groups: <by row>
##
##        name   model
```

```
## 1     Aamna <S3:lm>
## 2    Aaryah <S3:lm>
## 3  Abdiwali <S3:lm>
## 4   Abegail <S3:lm>
## 5      Abia <S3:lm>
## 6  Abrianne <S3:lm>
## 7    Adalyn <S3:lm>
## 8     Adaya <S3:lm>
## 9   Addalie <S3:lm>
## 10     Adji <S3:lm>
## ..      ...     ...
```

Now we have a model fitted to each name. We can test that the model for Madison is the same as what we calculated early.

```
filter(names_models, name == "Madison")$model
```

```
## [[1]]
##
## Call:
## lm(formula = F ~ year, data = .)
##
## Coefficients:
## (Intercept)         year
##   -37.05580      0.01895
```

There is another summary statistic we can calculate, Pearson's r.

```
names_pearson <- names_to_fit %>%
  group_by(name) %>%
  do(pearson = with(., cor(year, F)))
names_pearson
```

```
## Source: local data frame [1,005 x 2]
## Groups: <by row>
##
##        name  pearson
```

```
## 1     Aamna <dbl[1]>
## 2    Aaryah <dbl[1]>
## 3  Abdiwali <dbl[1]>
## 4   Abegail <dbl[1]>
## 5      Abia <dbl[1]>
## 6  Abrianne <dbl[1]>
## 7    Adalyn <dbl[1]>
## 8     Adaya <dbl[1]>
## 9   Addalie <dbl[1]>
## 10     Adji <dbl[1]>
## ..      ...      ...
```

Now we can filter this to find names with a change.

```
interesting_names_pearson <- names_pearson %>% filter(!is.na(pearson))

to_male <- interesting_names_pearson %>%
  filter(pearson < 0) %>%
  arrange(as.numeric(pearson))

to_female <- interesting_names_pearson %>%
  filter(pearson > 0) %>%
  arrange(desc(as.numeric(pearson)))

print("Names that became female")
```

```
## [1] "Names that became female"
```

```
head(to_female, 10)
```

```
## Source: local data frame [10 x 2]
##
##       name  pearson
## 1    Hager <dbl[1]>
## 2  Blakely <dbl[1]>
## 3     Riko <dbl[1]>
## 4   Marlin <dbl[1]>
```

```
## 5   Leslie <dbl[1]>
## 6  Madison <dbl[1]>
## 7      Ivy <dbl[1]>
## 8    River <dbl[1]>
## 9   Morley <dbl[1]>
## 10    Fate <dbl[1]>
```

```
print("Names that became female")
```

```
## [1] "Names that became female"
```

```
head(to_male, 10)
```

```
## Source: local data frame [10 x 2]
##
##       name  pearson
## 1  Romelle <dbl[1]>
## 2  Brecken <dbl[1]>
## 3    Tamar <dbl[1]>
## 4     Eddi <dbl[1]>
## 5      Jan <dbl[1]>
## 6   Indigo <dbl[1]>
## 7     Andi <dbl[1]>
## 8    Keanu <dbl[1]>
## 9  Lindell <dbl[1]>
## 10    Yuki <dbl[1]>
```

The name Keanu is a good example of how we need to refine our model. Keanu was not used hardly at all until the 1990s (Keanu Reeves?). Presumably before that time it appeared less than five times per year. When it appears in the data, it is definitely male but there were at least few instances where it was female at the height of the popularity for the name. And then it became even more definitively male for more recent births. Our model should take into account that in years we are studying Keanu was never a female name.

TODO: Write a function to plot any given name.

TODO: Don't cut out `n` for each year so that we can look it up.

TODO: filtering data frame to get names which switched in particular directions.

TODO: Loess fitting

Next Steps

I recommend that you get a good book on statistics if you intend to do any serious work with it. Any book will do, but one that is particularly useful because it introduces statistics with R is Andy Field, Jeremy Miles, and Zoe Field, *Discovering Statistics Using R* (SAGE, 2012).