Exploratory Data Analysis

Jonathan Kropko (jkropko@virginia.edu)
District Data Labs

Descriptive Statistics

A descriptive statistic is a quantity that **summarizes** some feature of data. These statistics are usually quick and easy to calculate.

Descriptive Statistics

A descriptive statistic is a quantity that **summarizes** some feature of data. These statistics are usually quick and easy to calculate.

Examples: mean, median, standard deviation, correlation, quantiles, frequency tables, cross-tabulations

Descriptive Statistics

A descriptive statistic is a quantity that **summarizes** some feature of data. These statistics are usually quick and easy to calculate.

Examples: mean, median, standard deviation, correlation, quantiles, frequency tables, cross-tabulations

Descriptive statistics contrast with inferential statistics, which involve hypothesis tests, *p*-values, etc. **Simple** inferential statistics are great ways to explore data quickly, without spending much time worrying about model design (we'll have plenty of chances for that later!)

Descriptive Statistics

A descriptive statistic is a quantity that **summarizes** some feature of data. These statistics are usually quick and easy to calculate.

Examples: mean, median, standard deviation, correlation, quantiles, frequency tables, cross-tabulations

Descriptive statistics contrast with inferential statistics, which involve hypothesis tests, *p*-values, etc. **Simple** inferential statistics are great ways to explore data quickly, without spending much time worrying about model design (we'll have plenty of chances for that later!)

Examples: t-tests, χ^2 tests of association

The `summary()` function

The `summary()` function is one of the most useful R commands to run after loading and cleaning data.

The `summary()` function

The `summary()` function is one of the most useful R commands to run after loading and cleaning data.

It gives different output depending on the class of the object passed to it. For a data frame, it provides the mean, median, min, and max, and quartiles for **continuous** variables:

```
> summary(anes$fthrc)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00    3.00   44.00   42.99   76.00  100.00       1
```

And it displays frequencies for **factor** variables:

```
> summary(anes$vote12)
 Mitt Romney Barack Obama Someone else         NA's
         371          512           68          249
```

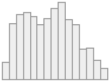
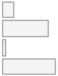
The `dfSummary()` function

The `dfSummary()` from the `summarytools` package provides many descriptive statistics in a neat HTML format.

The `dfSummary()` function

The `dfSummary()` from the `summarytools` package provides many descriptive statistics in a neat HTML format.

```
dfSummary(gss, plain.ascii = FALSE, style = "grid",
          graph.magnif = 0.75, valid.col = FALSE,
          tmp.img.dir = "/tmp", headings = FALSE)
```

No	Variable	Stats / Values	Freqs (% of Valid)	Graph	Missing
1	sex [factor]	1. female 2. male	1586 (55.5%) 1273 (44.5%)		0 (0%)
2	age [numeric]	Mean (sd) : 49.1 (17.7) min < med < max: 18 < 49 < 89 IQR (CV) : 28 (0.4)	72 distinct values		10 (0.35%)
3	artexbt [factor]	1. no 2. yes	987 (67.2%) 481 (32.8%)		1391 (48.65%)
4	class [factor]	1. lower class 2. middle class 3. upper class 4. working class	286 (10.1%) 1143 (40.3%) 80 (2.8%) 1328 (46.8%)		22 (0.77%)

Quick calculations

There are several functions that quickly calculate summary statistics. They have straightforward names:

Quick calculations

There are several functions that quickly calculate summary statistics. They have straightforward names:

- `mean()` — calculates the mean
- `median()` — calculates the median
- `sd()` — calculates the standard deviation
- `var()` — calculates the variance
- `min()` — calculates the minimum value
- `max()` — calculates the maximum value

Quick calculations

There are several functions that quickly calculate summary statistics. They have straightforward names:

- `mean()` — calculates the mean
- `median()` — calculates the median
- `sd()` — calculates the standard deviation
- `var()` — calculates the variance
- `min()` — calculates the minimum value
- `max()` — calculates the maximum value

If there are missing values for an object, these functions will return `NA` unless you also specify the argument `na.rm=TRUE`, in which case it ignores the missing values.

Percentiles

A percentile is a statistic that **ranks** the observations from lowest to highest, counts from the lowest to a specified percent, and reports the value of the observation at that percent.

Percentiles

A percentile is a statistic that **ranks** the observations from lowest to highest, counts from the lowest to a specified percent, and reports the value of the observation at that percent.

The median is the **50th percentile**. 50% of the observations have a value lower than or equal to the median.

Percentiles

A percentile is a statistic that **ranks** the observations from lowest to highest, counts from the lowest to a specified percent, and reports the value of the observation at that percent.

The median is the **50th percentile**. 50% of the observations have a value lower than or equal to the median.

To find percentiles, use the `quantile()` function. Use the `probs` argument to specify the percentiles you want to see.

Percentiles

A percentile is a statistic that **ranks** the observations from lowest to highest, counts from the lowest to a specified percent, and reports the value of the observation at that percent.

The median is the **50th percentile**. 50% of the observations have a value lower than or equal to the median.

To find percentiles, use the `quantile()` function. Use the `probs` argument to specify the percentiles you want to see.

Here are the 10th, 40th, and 85th percentiles for Hillary Clinton's thermometer scores in the 2016 ANES pilot study:

```
> quantile(anes$fthrc, probs=c(.1, .4, .85), na.rm=TRUE)
10% 40% 85%
  0  20  89
```

Frequency tables

A frequency table is the most useful way to describe a categorical (factor) variable. It is simply a count of the number of observations with each category.

Frequency tables

A frequency table is the most useful way to describe a categorical (factor) variable. It is simply a count of the number of observations with each category.

There are three functions that report frequencies:

- `table()`
- `plyr::count()`
- `xtabs()`

Frequency tables

A frequency table is the most useful way to describe a categorical (factor) variable. It is simply a count of the number of observations with each category.

There are three functions that report frequencies:

- `table()`
- `plyr::count()`
- `xtabs()`

`table()` displays these frequencies side by side:

```
> table(anes$vote12)

 Mitt Romney Barack Obama Someone else
         371          512           68
```

Frequency tables

`plyr::count()` displays these frequencies vertically:

```
> plyr::count(anes$vote12)
             x freq
1  Mitt Romney  371
2 Barack Obama  512
3 Someone else   68
4         <NA>  249
```

Frequency tables

`plyr::count()` displays these frequencies vertically:

```
> plyr::count(anes$vote12)
             x freq
1  Mitt Romney  371
2 Barack Obama  512
3 Someone else   68
4         <NA>  249
```

`xtabs()` also displays these frequencies side by side, but requires a slightly different syntax:

```
> xtabs( ~ vote12, data=anes)
vote12
 Mitt Romney Barack Obama Someone else
         371          512           68
```

Cross tabulations

A cross tabulation is a method for examining the **relationship between two categorical variables**.

Cross tabulations

A cross tabulation is a method for examining the **relationship between two categorical variables**.

You specify two variables, one for the rows and one for the columns. The cells in the tables contain the **number of observations** that have the category represented by the row AND the category represented by the column.

Cross tabulations

A cross tabulation is a method for examining the **relationship between two categorical variables**.

You specify two variables, one for the rows and one for the columns. The cells in the tables contain the **number of observations** that have the category represented by the row AND the category represented by the column.

Both `table()` and `xtabs()` can create cross tabulations. For `table()`, write the rows variable first, the columns variable second:

```
> table(anes$gender, anes$vote12)

         Mitt Romney Barack Obama Someone else
  Male           191          236           46
  Female         180          276           22
```

Cross tabulations

For `xtabs()`, write a

$$\sim$$

first, then rows variable, then $+$, then the columns variable:

```
> xtabs( ~ gender + vote12, data=anes)
        vote12
gender   Mitt Romney Barack Obama Someone else
  Male           191          236           46
  Female         180          276           22
```

Cross tabulations

For `xtabs()`, write a

$$\sim$$

first, then rows variable, then $+$, then the columns variable:

```
> xtabs( ~ gender + vote12, data=anes)
        vote12
gender    Mitt Romney Barack Obama Someone else
  Male            191          236           46
  Female          180          276           22
```

You get a slightly cleaner display by placing this function within `ftable()` (this creates a "flat table"):

```
ftable(xtabs( ~ gender + vote12, data=anes))
       vote12 Mitt Romney Barack Obama Someone else
gender
Male                  191          236           46
Female                180          276           22
```

Cross tabulations

If you write a third variable in the formula, it displays a different cross-tab for each value of the third variable:

```
> xtabs( ~ gender + vote12 + sign, data=anes)
, , sign = Have done this in the past 12 months

        vote12
gender   Mitt Romney Barack Obama Someone else
  Male            50           69           15
  Female          35           75            1

, , sign = Have not done this in the past 12 months

        vote12
gender   Mitt Romney Barack Obama Someone else
  Male           141          167           31
  Female         145          201           21
```

Cross tabulations

Row percents are the percent that each cell comprises of the row total. Column percents are the percent that each cell comprises of the column total. And cell percents are the percent that each cell comprises of the overall total.

Cross tabulations

Row percents are the percent that each cell comprises of the row total. Column percents are the percent that each cell comprises of the column total. And cell percents are the percent that each cell comprises of the overall total.

To calculate these percents, save the table as a separate object

```
my.table <- table(anes$gender, anes$vote12)
```

Cross tabulations

Row percents are the percent that each cell comprises of the row total. Column percents are the percent that each cell comprises of the column total. And cell percents are the percent that each cell comprises of the overall total.

To calculate these percents, save the table as a separate object

```
my.table <- table(anes$gender, anes$vote12)
```

Then use the `prop.table()` function. By default it gives cell proportions (percents/100)

```
> prop.table(my.table)

         Mitt Romney Barack Obama Someone else
  Male    0.20084122   0.24815983   0.04837014
  Female  0.18927445   0.29022082   0.02313354
```

Cross tabulations

For row proportions, use the argument `margin=1`:

```
> prop.table(my.table, margin=1)

         Mitt Romney Barack Obama Someone else
  Male    0.40380550   0.49894292   0.09725159
  Female  0.37656904   0.57740586   0.04602510
```

Cross tabulations

For row proportions, use the argument `margin=1`:

```
> prop.table(my.table, margin=1)

       Mitt Romney Barack Obama Someone else
Male    0.40380550   0.49894292   0.09725159
Female  0.37656904   0.57740586   0.04602510
```

For column proportions, use the argument `margin=2`:

```
> prop.table(my.table, margin=2)

       Mitt Romney Barack Obama Someone else
Male     0.5148248    0.4609375    0.6764706
Female   0.4851752    0.5390625    0.3235294
```

Cross tabulations

For row proportions, use the argument `margin=1`:

```
> prop.table(my.table, margin=1)

       Mitt Romney Barack Obama Someone else
  Male    0.40380550   0.49894292   0.09725159
  Female  0.37656904   0.57740586   0.04602510
```

For column proportions, use the argument `margin=2`:

```
> prop.table(my.table, margin=2)

       Mitt Romney Barack Obama Someone else
  Male     0.5148248    0.4609375    0.6764706
  Female   0.4851752    0.5390625    0.3235294
```

For percents, multiply by 100:

```
> 100*prop.table(my.table, margin=2)

       Mitt Romney Barack Obama Someone else
  Male      51.48248     46.09375     67.64706
  Female    48.51752     53.90625     32.35294
```

Correlation

Correlation is a statistic that measures the extent to which two continuous variables move with one another or opposed to one another.

Correlation

Correlation is a statistic that measures the extent to which two continuous variables move with one another or opposed to one another.

Positive correlation means that as one variable changes, the other changes in the same direction. If the correlation is 1, there's a perfect upward-sloping linear formula that describes this relationship.

Correlation

Correlation is a statistic that measures the extent to which two continuous variables move with one another or opposed to one another.

Positive correlation means that as one variable changes, the other changes in the same direction. If the correlation is 1, there's a perfect upward-sloping linear formula that describes this relationship.

Negative correlation means that as one variable changes, the other changes in the opposite direction. If the correlation is -1, there's a perfect downward-sloping linear formula that describes this relationship.

Correlation

To use R to calculate correlation, create a separate data frame with all of the continuous variables you want to calculate correlations between.

Correlation

To use R to calculate correlation, create a separate data frame with all of the continuous variables you want to calculate correlations between.

The use `na.omit()` to remove rows with missing values.

Correlation

To use R to calculate correlation, create a separate data frame with all of the continuous variables you want to calculate correlations between.

The use `na.omit()` to remove rows with missing values.

Then use the `cor()` function on the data.

```
> cor.data <- select(anes, fthrc, ftobama, fttrump, ftcruz)
> cor.data <- na.omit(cor.data)
> cor(cor.data)
             fthrc    ftobama    fttrump     ftcruz
fthrc    1.0000000  0.7936290 -0.4505056 -0.4266794
ftobama  0.7936290  1.0000000 -0.5849056 -0.5280243
fttrump -0.4505056 -0.5849056  1.0000000  0.5071978
ftcruz  -0.4266794 -0.5280243  0.5071978  1.0000000
```

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a t-test. Many of you probably already have! We'll cover just the rudimentary basics here.

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a t-test. Many of you probably already have! We'll cover just the rudimentary basics here.

A t-test takes the following steps:

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a t-test. Many of you probably already have! We'll cover just the rudimentary basics here.

A t-test takes the following steps:

- Calculate the mean of both variables and the difference between these two means

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a t-test. Many of you probably already have! We'll cover just the rudimentary basics here.

A t-test takes the following steps:

- Calculate the mean of both variables and the difference between these two means
- Suppose the means are equal in the population, and that the sample is random

Student's t-test

A t-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a t-test. Many of you probably already have! We'll cover just the rudimentary basics here.

A t-test takes the following steps:

- Calculate the mean of both variables and the difference between these two means
- Suppose the means are equal in the population, and that the sample is random
- Ask: how likely is it that the difference could have been as far from 0 as it is? Calculate a probability called a p-**value**.

Student's *t*-test

A *t*-test is a way to test whether or not two variables **have equal means**.

We could spend a semester talking about the math behind a *t*-test. Many of you probably already have! We'll cover just the rudimentary basics here.

A *t*-test takes the following steps:

- Calculate the mean of both variables and the difference between these two means
- Suppose the means are equal in the population, and that the sample is random
- Ask: how likely is it that the difference could have been as far from 0 as it is? Calculate a probability called a *p*-**value**.
- If $p < .05$ (or another standard), then it's not likely the mean could have been this far from 0. In that case conclude that the means weren't equal in the population after all.

Student's t-test

When working with data frames, we usually test whether the means of variables are equal. In this case, the test should be paired, since the observations correspond.

Student's *t*-test

When working with data frames, we usually test whether the means of variables are equal. In this case, the test should be paired, since the observations correspond.

To run a paired *t*-test, use the `t.test()` function with the `paired=TRUE` argument:

```
> t.test(anes$fthrc, anes$fttrump, paired=TRUE)

        Paired t-test

data:  anes$fthrc and anes$fttrump
t = 2.5485, df = 1195, p-value = 0.01094
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 1.055163 8.113733
sample estimates:
mean of the differences
               4.584448
```

χ^2 test of association

A χ^2 test of association provides an inference about whether two categorical variables have a statistically significant relationship with one another.

χ^2 test of association

A χ^2 test of association provides an inference about whether two categorical variables have a statistically significant relationship with one another.

Consider this table again:

```
> table(anes$gender, anes$vote12)

         Mitt Romney Barack Obama Someone else
  Male           191          236           46
  Female         180          276           22
```

χ^2 test of association

A χ^2 test of association provides an inference about whether two categorical variables have a statistically significant relationship with one another.

Consider this table again:

```
> table(anes$gender, anes$vote12)

         Mitt Romney Barack Obama Someone else
  Male           191          236           46
  Female         180          276           22
```

Does gender appear to have anything to do with voting?

Are men more likely than women to vote for Romney?

Is there enough evidence here to conclude the relationship is real?

χ^2 test of association

To run a χ^2 test of association in R, save the table as a separate object. Then use the `chisq.test()` function to run the test:

```
> my.table <- table(anes$gender, anes$vote12)
> chisq.test(my.table)

        Pearson's Chi-squared test

data:  my.table
X-squared = 11.896, df = 2, p-value = 0.002611
```

Beautiful graphics with ggplot2

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence –

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

(2) Visual Integrity –

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

(2) Visual Integrity – the graphic should neither distort the underlying data nor create a false impression or interpretation of that data.

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

(2) Visual Integrity – the graphic should neither distort the underlying data nor create a false impression or interpretation of that data.

(3) Maximizing the Data-Ink Ratio – remove superfluous elements that do not report or describe the data.

Beautiful graphics with `ggplot2`

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

(2) Visual Integrity – the graphic should neither distort the underlying data nor create a false impression or interpretation of that data.

(3) Maximizing the Data-Ink Ratio – remove superfluous elements that do not report or describe the data.

(4) Aesthetic Elegance –

Beautiful graphics with ggplot2

Edward Tufte, a Professor Emeritus of statistics at Yale, is the world's leading authority on data visualization. In his books he outlines four properties of good visualizations:

(1) Graphical Excellence – the graphic must be usable, and covey information efficiently.

(2) Visual Integrity – the graphic should neither distort the underlying data nor create a false impression or interpretation of that data.

(3) Maximizing the Data-Ink Ratio – remove superfluous elements that do not report or describe the data.

(4) Aesthetic Elegance – use as simple a design as possible to convey a complex data structure.

Beautiful graphics with `ggplot2`

`ggplot()` is a system for making graphics in R easily, and following Tufte's principles as closely as possible.

Beautiful graphics with `ggplot2`

`ggplot()` is a system for making graphics in R easily, and following Tufte's principles as closely as possible.

This is NOT the only way to create graphics in R. Some alternatives:

- The base plotting system, which uses the `plot()` function
- The `lattice` package
- Numerous smaller plotting packages

Beautiful graphics with `ggplot2`

`ggplot()` is a system for making graphics in R easily, and following Tufte's principles as closely as possible.

This is NOT the only way to create graphics in R. Some alternatives:

- The base plotting system, which uses the `plot()` function
- The `lattice` package
- Numerous smaller plotting packages

Like anything else in R, there are many ways to do the same thing. The advantage of `ggplot()` is that it is relatively simple, prettier, and more elegant than the other systems are by default.

Beautiful graphics with `ggplot2`

`ggplot()` is a system for making graphics in R easily, and following Tufte's principles as closely as possible.

This is NOT the only way to create graphics in R. Some alternatives:

- The base plotting system, which uses the `plot()` function
- The `lattice` package
- Numerous smaller plotting packages

Like anything else in R, there are many ways to do the same thing. The advantage of `ggplot()` is that it is relatively simple, prettier, and more elegant than the other systems are by default.

The disadvantage of `ggplot()` is it's **inflexibility** when you want to customize elements of the graphic.

The `ggplot()` function

There are two ways to run a `ggplot()` function. If you call `ggplot()` directly:

```
ggplot(data, aes(x, y)) + . . .
```

then the graphic will appear immediately in the plot window in R Studio, or in the markdown file output.

The `ggplot()` function

There are two ways to run a `ggplot()` function. If you call `ggplot()` directly:

```
ggplot(data, aes(x, y)) + . . .
```

then the graphic will appear immediately in the plot window in R Studio, or in the markdown file output.

If you save the `ggplot()` output to an object, then the graphic won't display until you call the object directly.

```
g <- ggplot(data, aes(x, y)) + . . .
```

Nothing will be displayed until I type `g`. That's useful if I want to create the graphic, but I don't want to slow down compilation by displaying it right away.

The `ggplot()` function

The `ggplot()` function has three parts:

1. The data frame that contains the data to be plotted
2. An aesthetics statement: which variable is x? which is y? which is a grouping variable? etc
3. Additional information about the **type of plot**, labels, grids

The `ggplot()` function

The `ggplot()` function has three parts:

1. The data frame that contains the data to be plotted
2. An aesthetics statement: which variable is x? which is y? which is a grouping variable? etc
3. Additional information about the **type of plot**, labels, grids

Just like all the `tidyverse` commands, the first argument is the **data frame**. To make a plot from the ANES data we've been using, begin the command by typing

```
g <- ggplot(anes,
```

Aesthetics

The second thing to type in the `ggplot()` function is `aes()`, which is a function for the "aesthetics" of the plot.

Aesthetics

The second thing to type in the `ggplot()` function is `aes()`, which is a function for the "aesthetics" of the plot.

This is where you specify the x variable, the y variable (if you are making a plot that needs one), and optionally, variables that define colors, shapes, line styles, etc.

Aesthetics

The second thing to type in the `ggplot()` function is `aes()`, which is a function for the "aesthetics" of the plot.

This is where you specify the x variable, the y variable (if you are making a plot that needs one), and optionally, variables that define colors, shapes, line styles, etc.

To make a plot (such as a scatterplot, more on this later) with Hillary Clinton's thermometer score on the x axis and Donald Trump's thermometer on the y axis, type

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump))
```

Aesthetics

The second thing to type in the `ggplot()` function is `aes()`, which is a function for the "aesthetics" of the plot.

This is where you specify the x variable, the y variable (if you are making a plot that needs one), and optionally, variables that define colors, shapes, line styles, etc.

To make a plot (such as a scatterplot, more on this later) with Hillary Clinton's thermometer score on the x axis and Donald Trump's thermometer on the y axis, type

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump))
```

It's easy to forget about `aes()`, but it's required. Don't forget!

Different kinds of plots with `geom` functions

`ggplot()` works differently from other functions in one important way. Most functions use options, separated by **commas**, within the function itself.

Different kinds of plots with `geom` functions

`ggplot()` works differently from other functions in one important way. Most functions use options, separated by **commas**, within the function itself.

`ggplot()` uses options, but uses different functions for different options, and <u>separates the options with plus signs</u>. As far as I know, `ggplot()` is the only function that works this way.

Different kinds of plots with `geom` functions

`ggplot()` works differently from other functions in one important way. Most functions use options, separated by **commas**, within the function itself.

`ggplot()` uses options, but uses different functions for different options, and separates the options with plus signs. As far as I know, `ggplot()` is the only function that works this way.

There are currently 44 different functions that all begin `geom_...()`, where the dots are replaced by different words. Each `geom` function works with `ggplot()` to create a different graph.

Here's a few of the `geom_...()` functions

`geom_point()` – creates a scatterplot for two **continuous variables**. Draws a point for every observation, with the two variables providing the x and y coordinates.

`geom_smooth()` – draw best-fit lines or curves

`geom_line()` and `geom_path()` – create line plots

`geom_bar()` – create bar charts

`geom_histogram()` – create histograms

`geom_boxplot()` – create box plots

`geom_density()` and `geom_violin()` – draws the density of a continuous variable (shows the relative frequency of values)

Different kinds of plots with `geom` functions

Different `geom_...()` require different options, and there are too many to memorize right away. You will have to look up examples for a while – that's fine.

Different kinds of plots with geom functions

Different `geom_...()` require different options, and there are too many to memorize right away. You will have to look up examples for a while – that's fine.

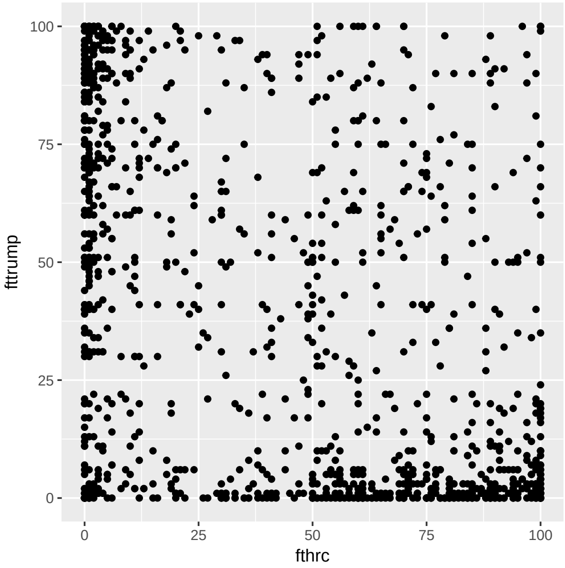
To create a scatterplot of Clinton and Trump's thermometer scores, add `geom_point()` to the code:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
        geom_point()
```

Style point: I like to press enter after every + sign to make the code easier to look at.

Then to display the graphic, type `g` in the script, console, or markdown file.

Different kinds of plots with `geom` functions



Different kinds of plots with `geom` functions

You can use more than one `geom_...()` function within one graphic. This overlays one graph over another.

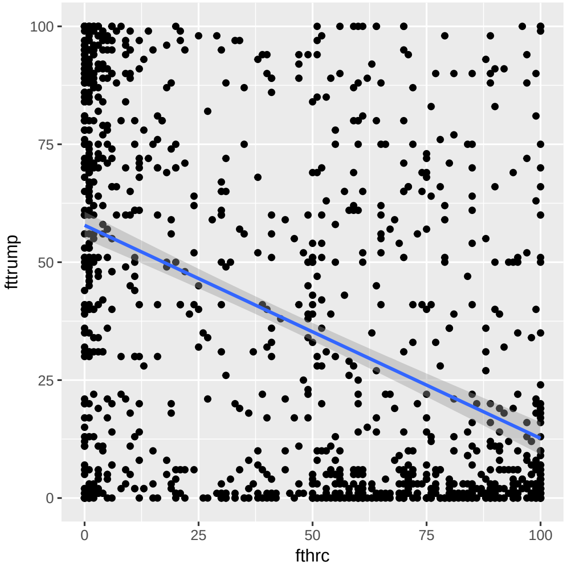
Different kinds of plots with geom functions

You can use more than one `geom_...()` function within one graphic. This overlays one graph over another.

To superimpose a best-fit line over the scatterplot, add `geom_smooth(method="lm")` to the code:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
        geom_point() +
        geom_smooth(method="lm")
```

Different kinds of plots with `geom` functions



Axis labels and titles

The axis labels for our working example right now are the raw variable names. We can make a nicer-looking graph by replacing these labels with better names, and by giving the graph a title.

Axis labels and titles

The axis labels for our working example right now are the raw variable names. We can make a nicer-looking graph by replacing these labels with better names, and by giving the graph a title.

To label the x and y axes, add `xlab("x-axis label")` and `ylab("y-axis label")` to the code.

Axis labels and titles

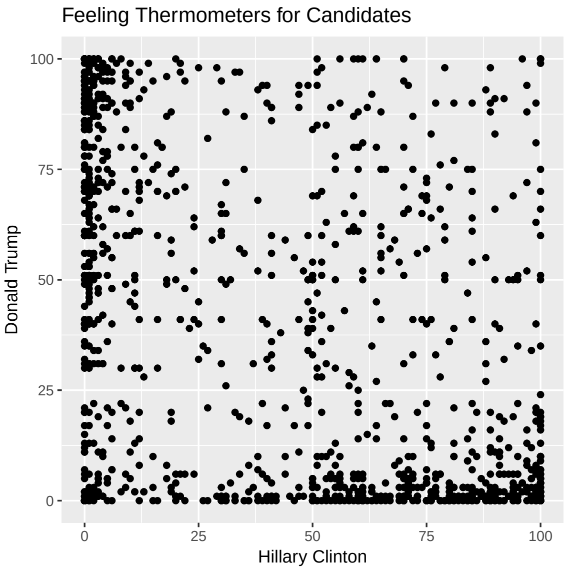
The axis labels for our working example right now are the raw variable names. We can make a nicer-looking graph by replacing these labels with better names, and by giving the graph a title.

To label the x and y axes, add `xlab("x-axis label")` and `ylab("y-axis label")` to the code.

To give the graph a title, add `ggtitle("title")` to the code.

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
```

Axis labels and titles



Using colors, shapes, or line types for groups

Sometimes it makes sense to distinguish between groups in a graphic.

Using colors, shapes, or line types for groups

Sometimes it makes sense to distinguish between groups in a graphic.

In the scatterplot, we can distinguish men and women, Democrats and Republicans, voters from non-voters, and so on. We can use **colors, shapes, or both** to distinguish these groups.

Using colors, shapes, or line types for groups

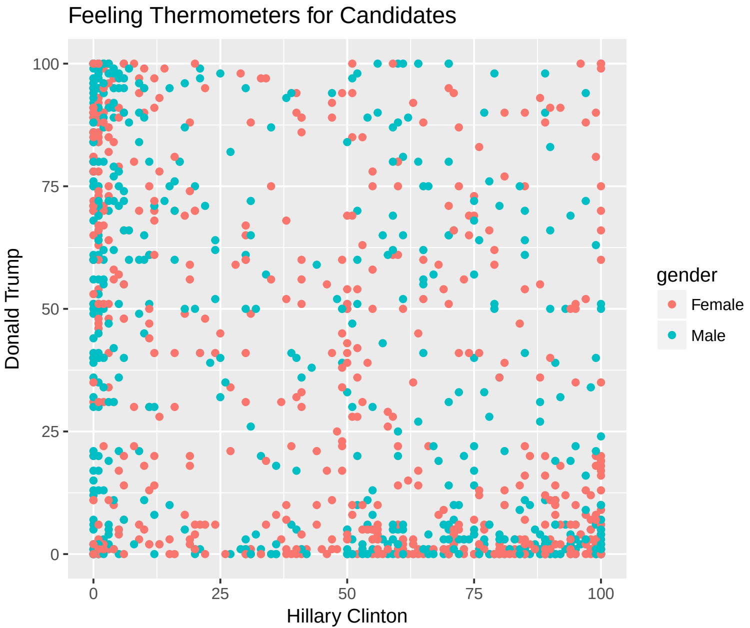
Sometimes it makes sense to distinguish between groups in a graphic.

In the scatterplot, we can distinguish men and women, Democrats and Republicans, voters from non-voters, and so on. We can use **colors, shapes, or both** to distinguish these groups.

Inside the `aes()` function, after specifying `x` and `y`, type `col=gender` to use different colors for the points that refer to men and the points that refer to women:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump, col=gender)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
```

Using colors, shapes, or line types for groups

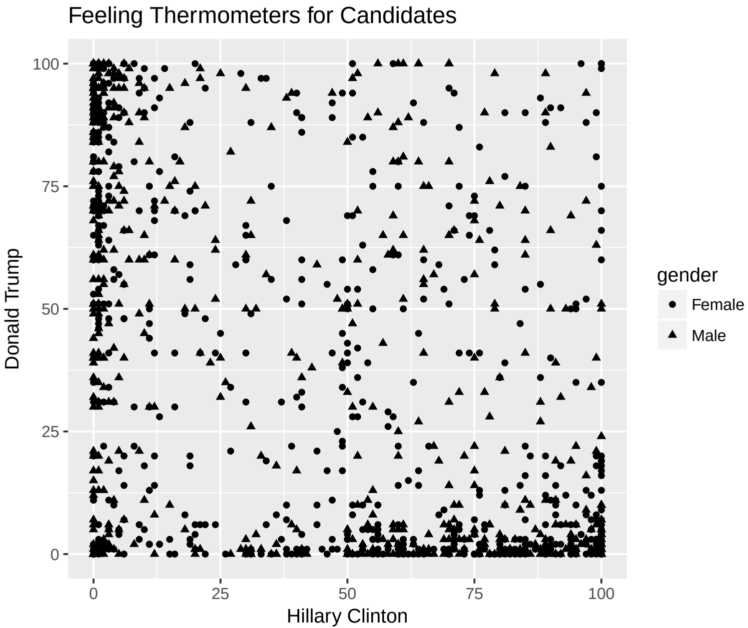


Using colors, shapes, or line types for groups

To use different shapes, type `pch=gender`.

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump, pch=gender)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
```

Using colors, shapes, or line types for groups

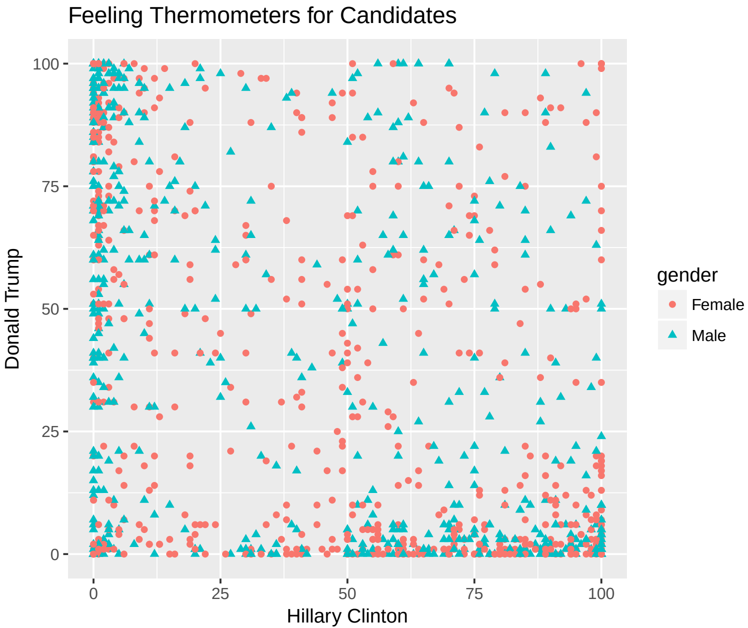


Using colors, shapes, or line types for groups

You can set both colors and shapes to vary based on the same variable:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump, col=gender, pch=gender)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
```

Using colors, shapes, or line types for groups

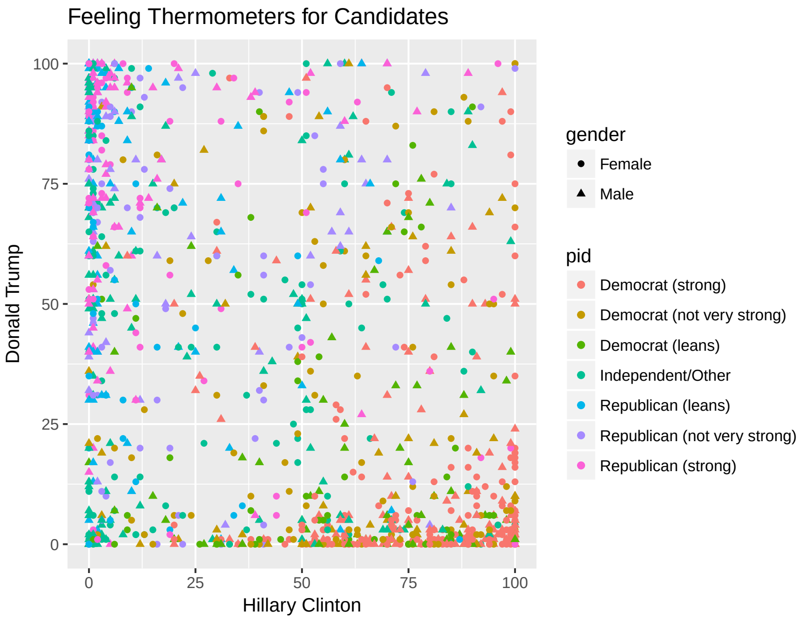


Using colors, shapes, or line types for groups

You can set both colors and shapes to vary based on the two different variables:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump, col=pid, pch=gender)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
```

Using colors, shapes, or line types for groups



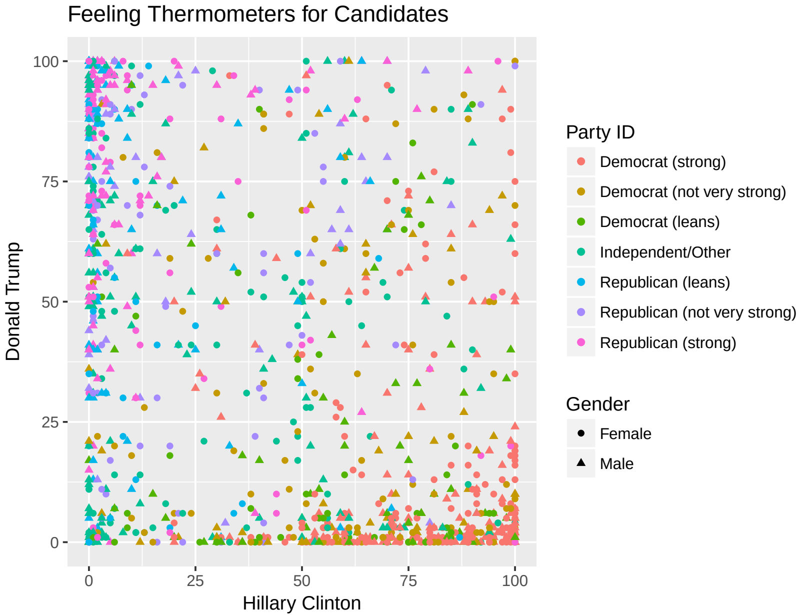
Using colors, shapes, or line types for groups

Notice that when you use colors or shapes to denote groups, `ggplot()` automatically creates a legend. You might want to change the legend title.

To do that, type `labs(color = "Party ID")` to change the legend title for the colors, and type `labs(pch = "Gender")` to change the legend title for the shapes:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump, col=pid, pch=gender)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates") +
      labs(color = "Party ID") +
      labs(pch = "Gender")
```

Using colors, shapes, or line types for groups



Creating a grid of plots

Colors and shapes are good ways to distinguish groups, but they aren't the only ways to do it. Another option is to create a grid of corresponding graphs, one for each category of a group.

Creating a grid of plots

Colors and shapes are good ways to distinguish groups, but they aren't the only ways to do it. Another option is to create a grid of corresponding graphs, one for each category of a group.

`ggplot()` will automatically format the graphs to fit perfectly next to each other, and will put the category name at the top of each graph.

Creating a grid of plots

Colors and shapes are good ways to distinguish groups, but they aren't the only ways to do it. Another option is to create a grid of corresponding graphs, one for each category of a group.

`ggplot()` will automatically format the graphs to fit perfectly next to each other, and will put the category name at the top of each graph.

To create a grid, use a `facet_...()` function. There are four ways to use these functions:

Creating a grid of plots

Colors and shapes are good ways to distinguish groups, but they aren't the only ways to do it. Another option is to create a grid of corresponding graphs, one for each category of a group.

`ggplot()` will automatically format the graphs to fit perfectly next to each other, and will put the category name at the top of each graph.

To create a grid, use a `facet_...()` function. There are <u>four</u> ways to use these functions:

(1) `facet_grid(. ~ gender)` will place the plots for men and women in different columns

Creating a grid of plots

Colors and shapes are good ways to distinguish groups, but they aren't the only ways to do it. Another option is to create a grid of corresponding graphs, one for each category of a group.

`ggplot()` will automatically format the graphs to fit perfectly next to each other, and will put the category name at the top of each graph.

To create a grid, use a `facet_...()` function. There are four ways to use these functions:

(1) `facet_grid(. ~ gender)` will place the plots for men and women in different columns

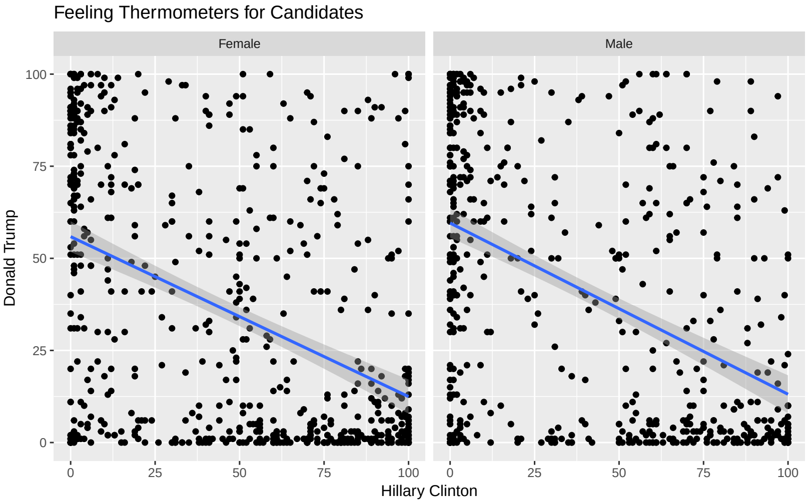
(2) `facet_grid(gender ~ .)` will place the plots for men and women in different rows

Creating a grid of plots

Here's an example breaking up gender into columns:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
     geom_point() +
     geom_smooth(method="lm") +
     xlab("Hillary Clinton") +
     ylab("Donald Trump") +
     ggtitle("Feeling Thermometers for Candidates") +
     facet_grid(. ~ gender)
```

Creating a grid of plots

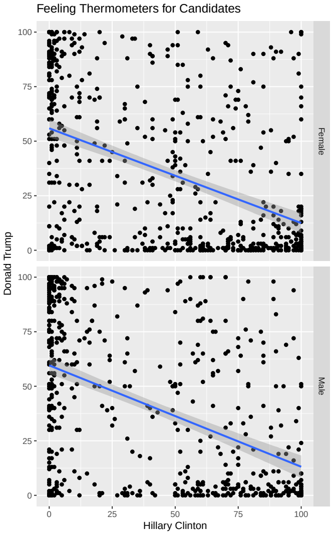


Creating a grid of plots

Here's an example breaking up gender into rows:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
     geom_point() +
     geom_smooth(method="lm") +
     xlab("Hillary Clinton") +
     ylab("Donald Trump") +
     ggtitle("Feeling Thermometers for Candidates") +
     facet_grid(gender ~ .)
```

Creating a grid of plots



Creating a grid of plots

(3) `facet_grid(gender ~ pid)` will create a plot for **each combination of party ID and gender**, and will place them in a grid where the rows represent genders and the columns represent party IDs.

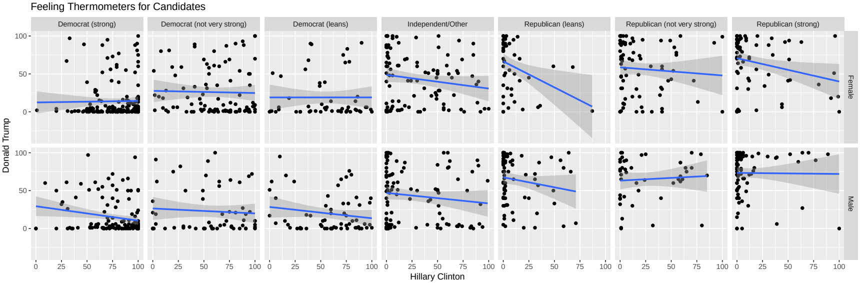
Creating a grid of plots

(3) `facet_grid(gender ~ pid)` will create a plot for **each combination of party ID and gender**, and will place them in a grid where the rows represent genders and the columns represent party IDs.

Here's an example:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
     geom_point() +
     geom_smooth(method="lm") +
     xlab("Hillary Clinton") +
     ylab("Donald Trump") +
     ggtitle("Feeling Thermometers for Candidates") +
     facet_grid(gender ~ pid)
```

Creating a grid of plots



Creating a grid of plots

(4) `facet_wrap( ~ pid)` will make a graph for every party ID, but will automatically fill several rows to make it look good.

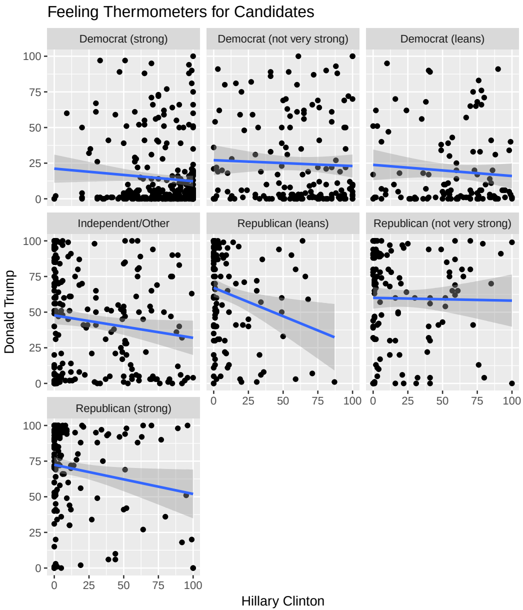
Creating a grid of plots

(4) `facet_wrap( ~ pid)` will make a graph for every party ID, but will automatically fill several rows to make it look good.

Here's an example:

```
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
     geom_point() +
     geom_smooth(method="lm") +
     xlab("Hillary Clinton") +
     ylab("Donald Trump") +
     ggtitle("Feeling Thermometers for Candidates") +
     facet_wrap(~ pid)
```

Creating a grid of plots



Saving graphics as PDF, JPEG, BMP, or PNG files

If you've saved your plot an an object (named g or something else), you can use R code to save the graphic as a PDF, JPEG, BMP, or PNG file.

Saving graphics as PDF, JPEG, BMP, or PNG files

If you've saved your plot an an object (named g or something else), you can use R code to save the graphic as a PDF, JPEG, BMP, or PNG file.

For a PDF, the code is

```
pdf("filename.pdf", width=5, height=5)
g
dev.off()
```

Saving graphics as PDF, JPEG, BMP, or PNG files

If you've saved your plot an an object (named g or something else), you can use R code to save the graphic as a PDF, JPEG, BMP, or PNG file.

For a PDF, the code is

```
pdf("filename.pdf", width=5, height=5)
g
dev.off()
```

The `pdf()` function has three arguments. First, write the name of the file you want to create. Then specify the width and height you want this image to be, in inches.

Saving graphics as PDF, JPEG, BMP, or PNG files

If you've saved your plot an an object (named g or something else), you can use R code to save the graphic as a PDF, JPEG, BMP, or PNG file.

For a PDF, the code is

```
pdf("filename.pdf", width=5, height=5)
g
dev.off()
```

The `pdf()` function has three arguments. First, write the name of the file you want to create. Then specify the width and height you want this image to be, in inches.

g is just the name of your ggplot graphic object.

Saving graphics as PDF, JPEG, BMP, or PNG files

If you've saved your plot an an object (named g or something else), you can use R code to save the graphic as a PDF, JPEG, BMP, or PNG file.

For a PDF, the code is

```
pdf("filename.pdf", width=5, height=5)
g
dev.off()
```

The `pdf()` function has three arguments. First, write the name of the file you want to create. Then specify the width and height you want this image to be, in inches.

g is just the name of your ggplot graphic object.

`dev.off()` turns off the PDF recording device that R uses. If you don't write this function, R will continue to print all visual output to the same PDF as new pages.

Saving graphics as PDF, JPEG, BMP, or PNG files

To save as a JPEG, use this code:

```
jpeg("filename.jpg", width=5, height=5, units="in")
g
dev.off()
```

Saving graphics as PDF, JPEG, BMP, or PNG files

To save as a JPEG, use this code:

```
jpeg("filename.jpg", width=5, height=5, units="in")
g
dev.off()
```

This command works similarly to `pdf()`, but you have to specify `units="in"` for R to understand that the width and heights are in inches.

Saving graphics as PDF, JPEG, BMP, or PNG files

To save as a JPEG, use this code:

```
jpeg("filename.jpg", width=5, height=5, units="in")
g
dev.off()
```

This command works similarly to `pdf()`, but you have to specify `units="in"` for R to understand that the width and heights are in inches.

To save a BMP or PNG file, use the same code as above, but replace `jpeg` with `bmp` or `png`.

Using `ggplot()` within an R markdown document

`ggplot()` works really well within code chunks in an R markdown file.

Using ggplot() within an R markdown document

ggplot() works really well within code chunks in an R markdown file.

You can control the width, height, and position of the graphic directly in the **code chunk options**. For example, to set a figure to be 4 inches wide, 6 inches tall, and centered, type this code chunk:

````
```{r chunkname, fig.width=4, fig.height=6, fig.align="center"}
g <- ggplot(anes, aes(x=fthrc, y=fttrump)) +
      geom_point() +
      xlab("Hillary Clinton") +
      ylab("Donald Trump") +
      ggtitle("Feeling Thermometers for Candidates")
g
```
````