Tutorial 12: Overall Look: `ggplot` Themes

Leah Brooks

April 25, 2019

We are concluding with a class that helps us work on the overall look of a plot. Apart from the data points you put on the plot – what we've spent most of this course working on – we also care about whether the labels are visible, what colors the axes are (and whether they exist), and the size of the labels, among many other considerations.

In `ggplot` we make these kind of modifications in a theme, which is the section that controls the overall look of the plot.

Happily, you can create a theme that you can use across multiple plots, and you can also tweak this theme for different plots.

A. Load packages and data

We begin by loading packages. Since we need both `ggplot` and `dplyr`, we'll just load the `tidyverse` package that has them both.

You may need to install this package if you haven't already installed it. To check if you've installed a package, you type

```
installed.packages()
```

I omit the output here, since it is very long – and has many columns.

To show just the packages column without all the other information, type

```
installed.packages()[,"Packages"]
```

If you don't have it installed, use `install.packages()` to install it and then use the `library()` command below. If you do have it installed, then just go straight to the `library()` command below.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.4.4
```

```
## -- Attaching packages ---------------------------------------------------------------- tidyverse 1.2
```

```
## v ggplot2 3.1.1       v purrr   0.2.4
## v tibble  2.1.1       v dplyr   0.8.0.1
## v tidyr   0.8.0       v stringr 1.2.0
## v readr   1.1.1       v forcats 0.4.0
```

```
## Warning: package 'ggplot2' was built under R version 3.4.4
```

```
## Warning: package 'tibble' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
## Warning: package 'forcats' was built under R version 3.4.4
```

```
## -- Conflicts ------------------------------------------------------------------- tidyverse_conflicts
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

We'll use the same data from Arlington County that we did last week. Please see tutorial 11 if you don't already have these data.

```
arl.p <- read.csv("H:/pppa_data_viz/2019/tutorial_data/lecture11/2019-04-19_arlington_2019_assessment_da
```

B. Pre-made themes

R has a variety of pre-made themes, and we'll start our exploration of themes by testing them out. However, to test them out, we need plots. So you can see what the themes look like across different types of plots, we make three different types of plots and apply the theme to all of them.

B.1. make three basic different plots

Look at what variables are in the Arlington data.

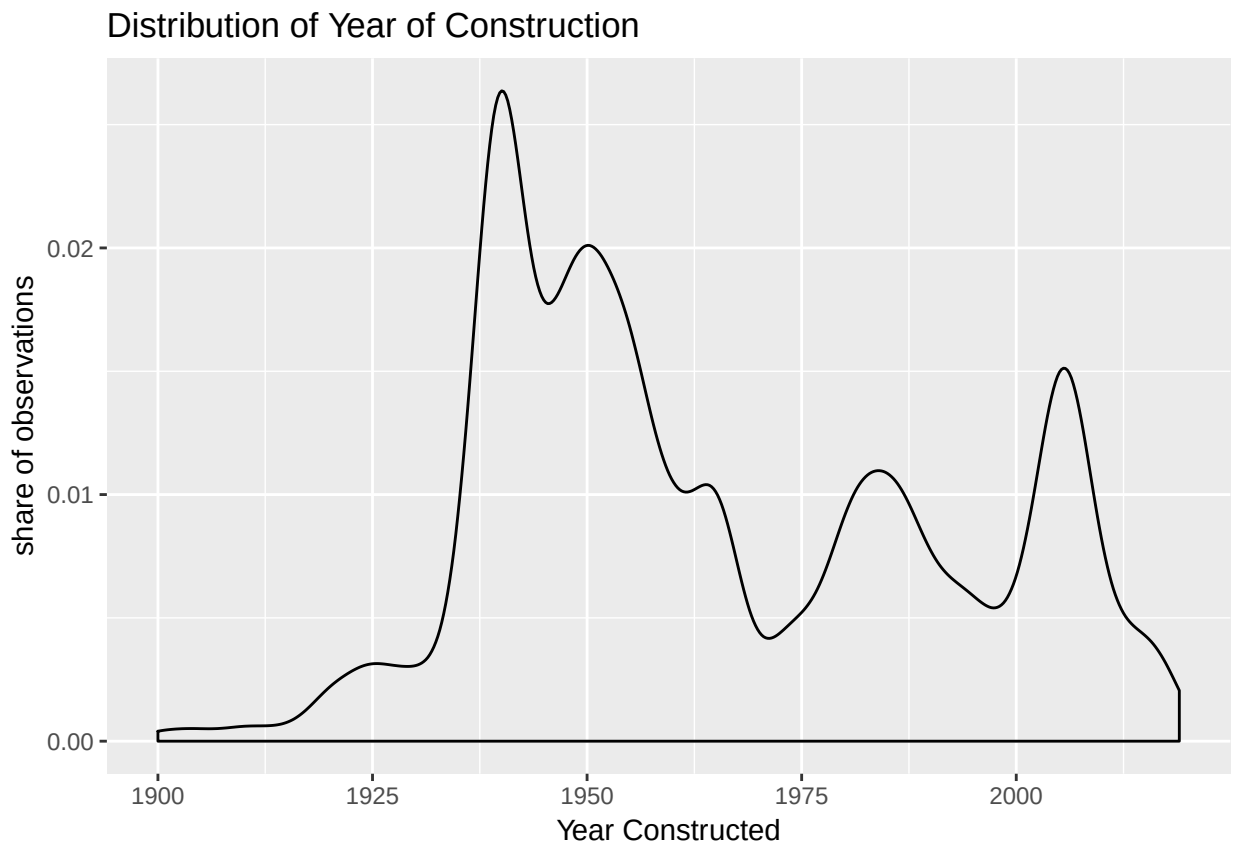
```
str(arl.p)
```

```
## 'data.frame':    65791 obs. of  46 variables:
##  $ IvwPropertyAssessmentHistoryKey  : int  74 111 148 185 222 260 334 371 408 445 ...
##  $ ProvalLrsnId                     : int  134 136 137 139 140 141 143 144 145 147 ...
##  $ RealEstatePropertyCode           : int  1001007 1001009 1001010 1001012 1001013 1001014 1001016 10
##  $ MasterRealEstatePropertyCode     : Factor w/ 37988 levels "","01001001",..: 3 5 6 8 9 10 12 13 14
##  $ ReasPropertyStatusCode           : Factor w/ 2 levels "A","T": 1 1 1 1 1 1 1 1 1 ...
##  $ PropertyClassTypeCode            : int  511 511 511 511 511 511 511 511 511 511 ...
##  $ PropertyClassTypeDsc             : Factor w/ 58 levels "100-Off Bldg-VacLand-no s.plan",..: 37 37
##  $ PropertyStreetNbrNameText        : Factor w/ 63816 levels "","1 N FENWICK ST",..: 31569 57412 573
##  $ PropertyStreetNbr                : int  3007 6547 6541 3518 3526 3530 3538 3544 3550 3562 ...
##  $ PropertyStreetNbrSuffixCode      : Factor w/ 15 levels "","A","B","BK",..: 1 1 1 1 1 1 1 1 1 1 ..
##  $ PropertyStreetDirectionPrefixCode: Factor w/ 3 levels "","N","S": 2 1 1 2 2 2 2 2 2 2 ...
##  $ PropertyStreetName               : Factor w/ 302 levels "","10th","10TH",..: 244 293 293 256 256 1
##  $ PropertyStreetTypeCode           : Factor w/ 14 levels "","AVE","BLVD",..: 13 3 3 13 13 13 13 13 1
##  $ PropertyStreetDirectionSuffixCode: Factor w/ 3 levels "","N","S": 1 1 1 1 1 1 1 1 1 1 ...
##  $ PropertyUnitNbr                  : Factor w/ 6329 levels "","# 1","# 102",..: 1 1 1 1 1 1 1 1 1 1
##  $ PropertyCityName                 : Factor w/ 13 levels "","ARILNGTON",..: 6 6 6 6 6 6 6 6 6 6 ...
##  $ PropertyZipCode                  : int  22213 22213 22213 22213 22213 22213 22213 22213 22213 222
##  $ ZoningDescListText               : Factor w/ 155 levels "","Accessory Dwell",..: 44 44 44 44 44 44
##  $ TradeName                        : Factor w/ 1788 levels "","#129 CRYSTAL GATEWAY 3",..: 1 1 1 1 1
##  $ PropertyYearBuilt                : int  2012 1950 1950 2008 1950 1950 1950 1950 1950 2013 ...
##  $ GrossFloorAreaSquareFeetQty      : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ EffectiveAgeYearDate             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ NumberOfUnitsCnt                 : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ StoryHeightCnt                   : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ ValuationYearDate                : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ CommercialPropertyTypeDsc        : Factor w/ 5 levels "","Apartment",..: 1 1 1 1 1 1 1 1 1 ...
##  $ CondoModelName                   : Factor w/ 2370 levels "","(Penthse A) 2 Bd/2 Bth (1,510)",..: 1
##  $ CondoStyleName                   : Factor w/ 31 levels "","19","20","Co-op",..: 1 1 1 1 1 1 1 1 1
##  $ FinishedStorageAreaSquareFeetQty : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ StorageAreaSquareFeetQty         : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ UnitNbr                          : Factor w/ 6716 levels "","00000","00001",..: 1 1 1 1 1 1 1 1 1
##  $ PropertyKey                      : int  2 3 4 5 6 7 9 10 11 12 ...
##  $ ReasPropertyOwnerKey             : int  20362 55397 57081 26494 36100 40877 11120 37425 17377 619
##  $ ArlingtonStreetKey               : int  179 222 222 185 185 185 185 185 185 185 ...
```

```
##  $ PropertyExpiredInd              : Factor w/ 1 level "False": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CommercialInd                   : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1 1 1 ...
##  $ DistrictNbr                     : Factor w/ 11 levels "0C","0G","0M",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ TaxExemptionTypeDsc             : Factor w/ 16 levels "","0 - WMATA - NVTC",..: 1 1 1 1 1 1 1 1
##  $ CondominiumProjectName          : Factor w/ 199 levels "","1423 RHODES STEET CONDOMINIUM",..: 1
##  $ TaxBalanceAmt                   : num  NA NA NA -6749 NA ...
##  $ AssessedYearDate                : int  2019 2019 2019 2019 2019 2019 2019 2019 2019 2019 ...
##  $ TotalAssessedAmt                : int  1941900 1053200 1039500 1406500 760400 1189400 788400 808
##  $ AssessmentDate                  : Factor w/ 6 levels "1/1/2019 12:00:00 AM",..: 1 1 1 1 1 1 1 1
##  $ AssessmentChangeReasonTypeDsc   : Factor w/ 3 levels "01- Annual","16- Tax to Exempt",..: 1 1 1
##  $ ImprovementValueAmt             : int  1151000 400900 380000 709800 87500 482900 105900 136300 1
##  $ LandValueAmt                    : int  790900 652300 659500 696700 672900 706500 682500 671900 6
```

We make a density curve, a bar graph and a scatter. For the density graph, we'll show the share of structures built in each year using `geom_density()`. Hopefully the code below is familiar to you.

```
b1a <- ggplot() +
       geom_density(data = arl.p[which(arl.p$PropertyYearBuilt >= 1900),],
                    aes(x=PropertyYearBuilt))  +
       labs(title = "Distribution of Year of Construction",
            x = "Year Constructed",
            y = "share of observations")
b1a
```



Now we want to make a bar graph of the number of structures of each Property Class type. Because there are many property types that have very few observations, we limit the bar chart to those with at least 200 structures. To do this, we need to know the total number of structures by type in the `alr.p` dataframe.

We can do this with `group_by()` to tell R that we are interested in statistics by `PropertyClassTypeDsc` and then `mutate()` which can add a summary statistic to a dataframe.

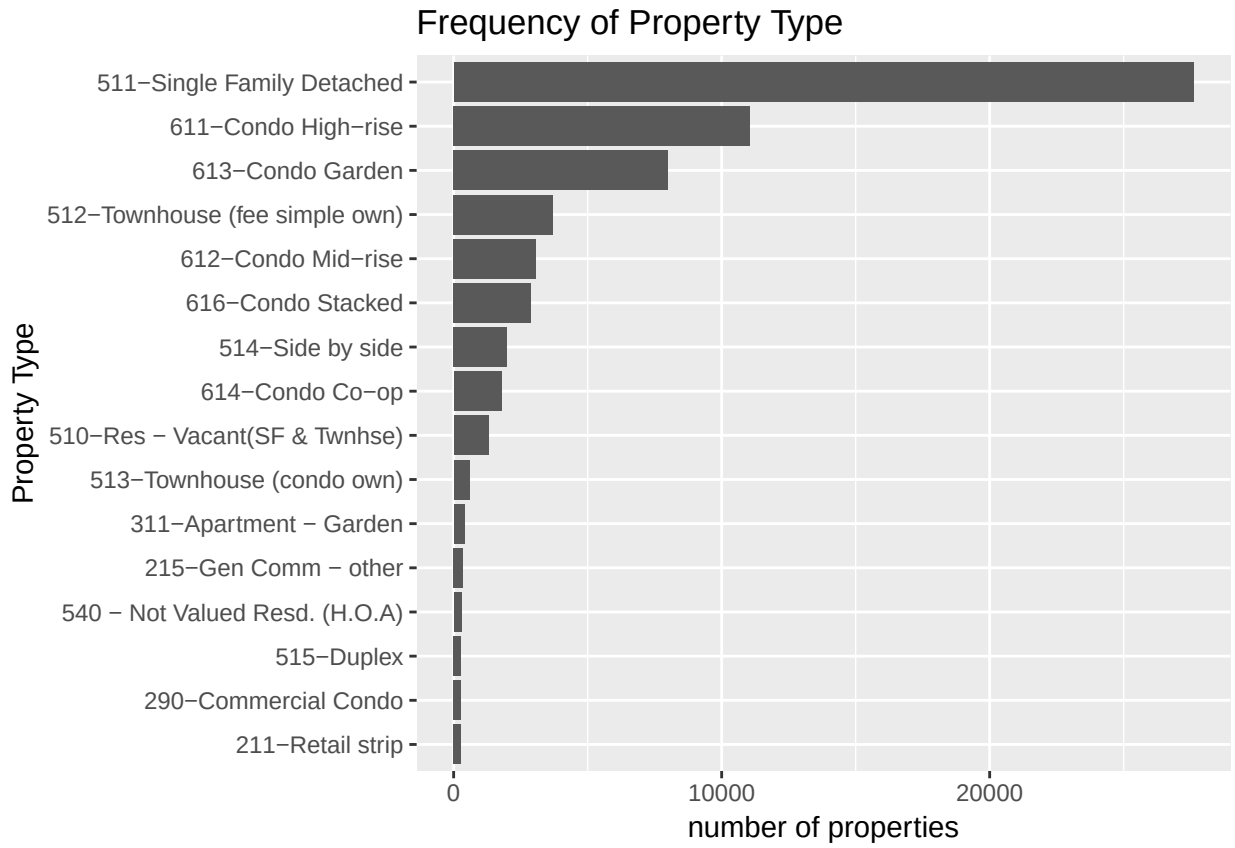
Specifically, the mutate command tells R to add a new variable named `num.pct` to `arl.p` that counts the number of observations (`n()`) by property class group (due to `group_by`). I then use `table()` to make sure that these numbers look reasonable.

```
# find the number of each type so we can restrict the graph
arl.p <- group_by(.data = arl.p, PropertyClassTypeDsc)
arl.p <- mutate(.data = arl.p, num.pct = n())
table(arl.p$num.pct)
```

```
##
##     1     5     6     7     8     9    10    11    16    19    20    24
##     1    15    12     7     8    18    10    11    32    19    40    72
##    27    29    36    37    45    50    51    53    56    58    60    64
##    27    58    36    37    45    50    51    53    56    58    60    64
##    65    68    80   108   115   137   148   159   180   276   280   293
##    65    68    80   108   115   137   148   159   180   552   280   293
##   350   402   595  1296  1784  1987  2894  3084  3715  8006 11051 27602
##   350   402   595  1296  1784  1987  2894  3084  3715  8006 11051 27602
```

Now that we've calculated this number of observations in each property type group, we can use this to subset the data as I do below in `arl.p[which(arl.p$num.pct > 200),]`. I also use `reorder()` in `geom_bar()` to order the plot by the number of structures in the county.

```
b1b <- ggplot() +
       geom_bar(data = arl.p[which(arl.p$num.pct > 200),],
                aes(x = reorder(PropertyClassTypeDsc,num.pct))) +
       coord_flip() +
       labs(title = "Frequency of Property Type",
            x = "Property Type",
            y = "number of properties")
b1b
```



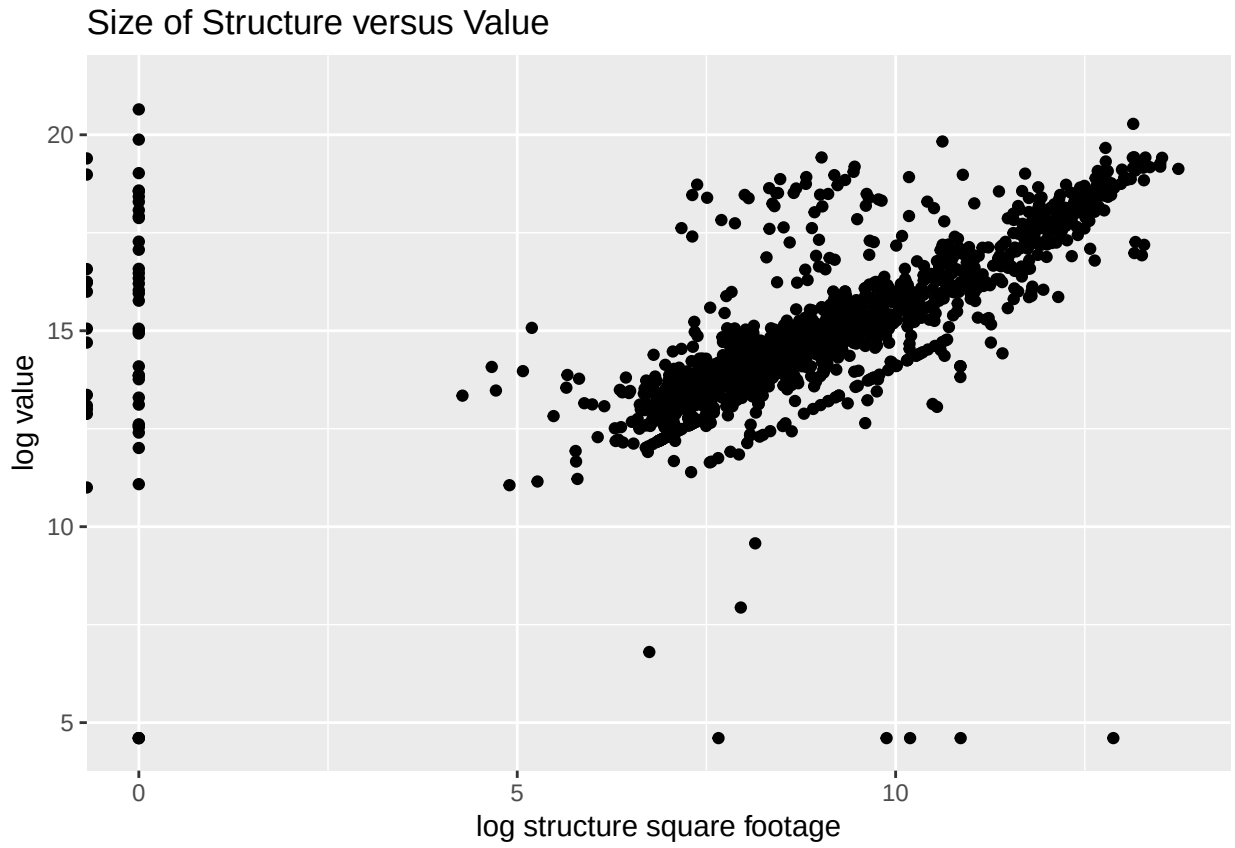
Finally, a scatter. We are looking at the relationship between structure size ("GrossFloorAreaSquareFeetQty") versus value ("TotalAssessedAmt"). Because the distributions of these variables both have very long tails, we use the log to plot them. Below we use the technique as we did last class to take the log of two variables at once.

```
tolog <- c("GrossFloorAreaSquareFeetQty","TotalAssessedAmt")
arl.p[paste0("ln.",tolog)] <- log(arl.p[tolog])
```

Now we use `geom_point()` and the new variables to make the scatter.

```
b1c <- ggplot() +
       geom_point(data = arl.p, aes(x = ln.GrossFloorAreaSquareFeetQty,
                                    y = ln.TotalAssessedAmt)) +
       labs(title = "Size of Structure versus Value",
            x = "log structure square footage",
            y = "log value")
b1c
```

```
## Warning: Removed 64151 rows containing missing values (geom_point).
```



B.2. Add some pre-baked themes

Now that we have our graphs in hand, we can try out these themes. R has 9 or 10 pre-existing themes with settings that you can use without much bother at all (count 9 if you think that grey = gray). You can see the full list of these themes (here)[https://ggplot2.tidyverse.org/reference/ggtheme.html].

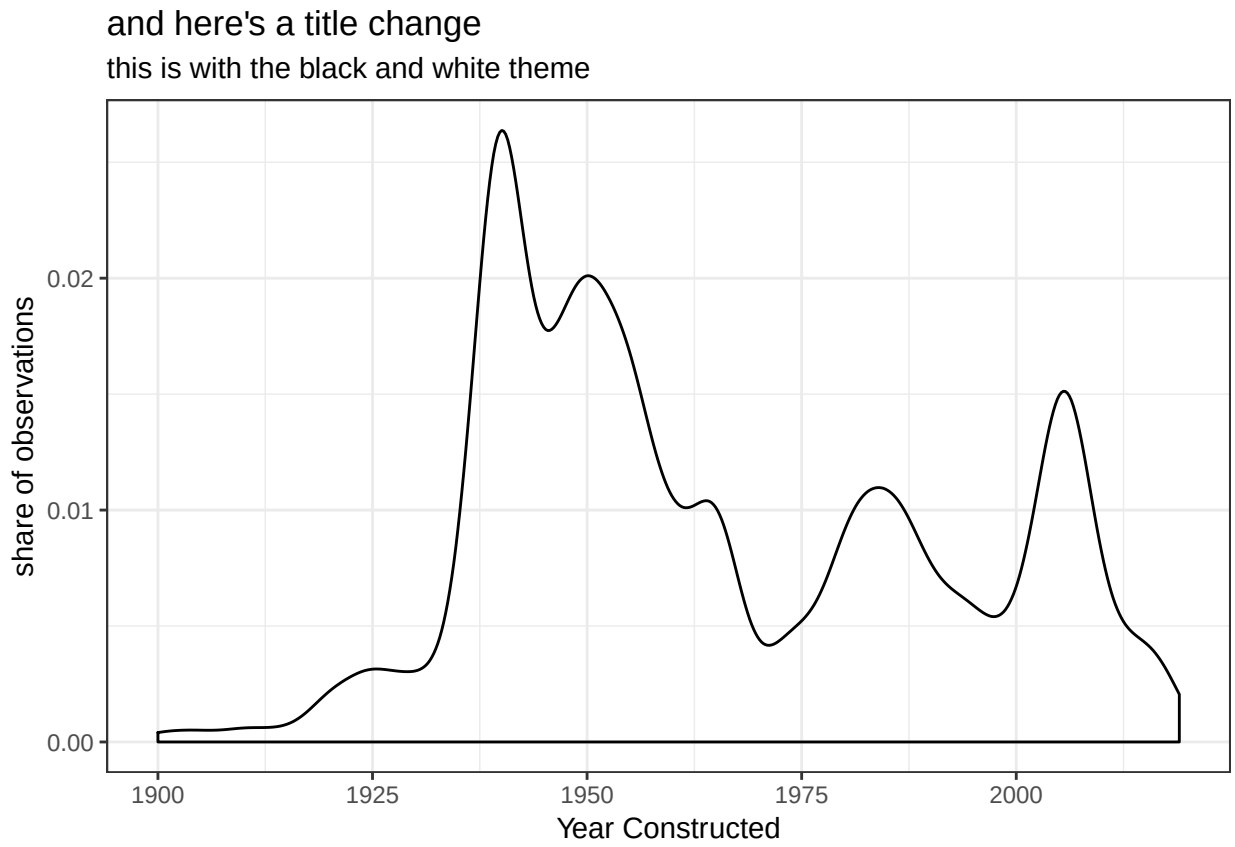
To test one out, let's add it to the first graph, and put a subtitle that names the theme so you can pick it out.

We start with the `theme_grey()`. Notice that we can just add the theme to the graph using a plus. However, notice that we are not overwriting the original plot – we're creating a new one so we can keep modifying the original one.

```
# just add to one
b1a_grey <- b1a +
            theme_grey() +
            labs(subtitle = "this is with the grey theme")
b1a_grey
```

What happens if you add something to a plot that you've already created that contradicts the previous instructions you gave? Here's an example. In the original `b1a` we made a title. Here we define a new title. The general rule is that the last command dominates.

```
b1a_bw   <- b1a + theme_bw() +
            labs(subtitle = "this is with the black and white theme",
                 title = "and here's a title change")
b1a_bw
```



And you can use the power of functions to try out a lot of themes on a lot of plots! Below we create a function called `trythemes` to put the theme and title on all three of our plots.

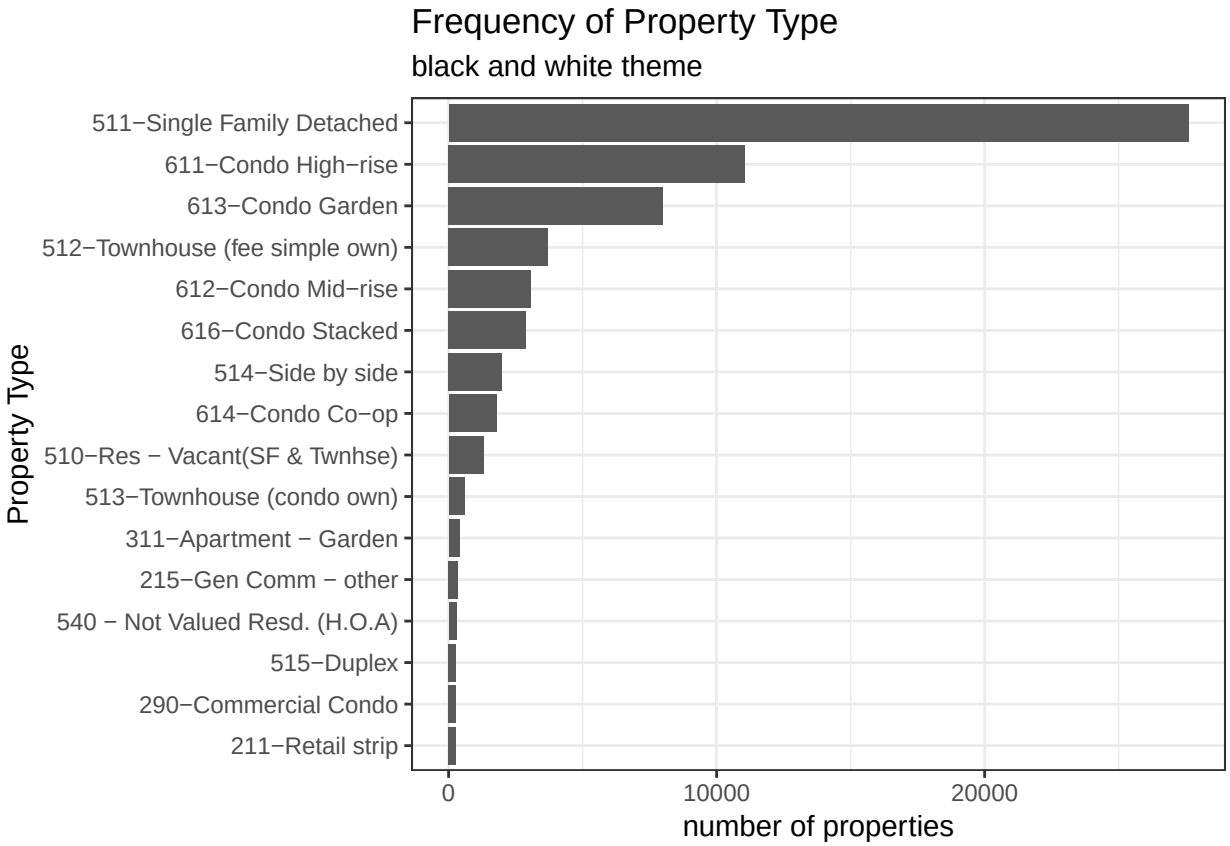
Here we build the function, naming it and telling R that it will have inputs named themer and et.

```
# you can also add it to all of them in a function
trythemes <- function(themer,et){
  b1an <- b1a + themer + et
  print(b1an)
  b1bn <- b1b + themer + et
  print(b1bn)
  b1cn <- b1c + themer + et
  print(b1cn)
}
```

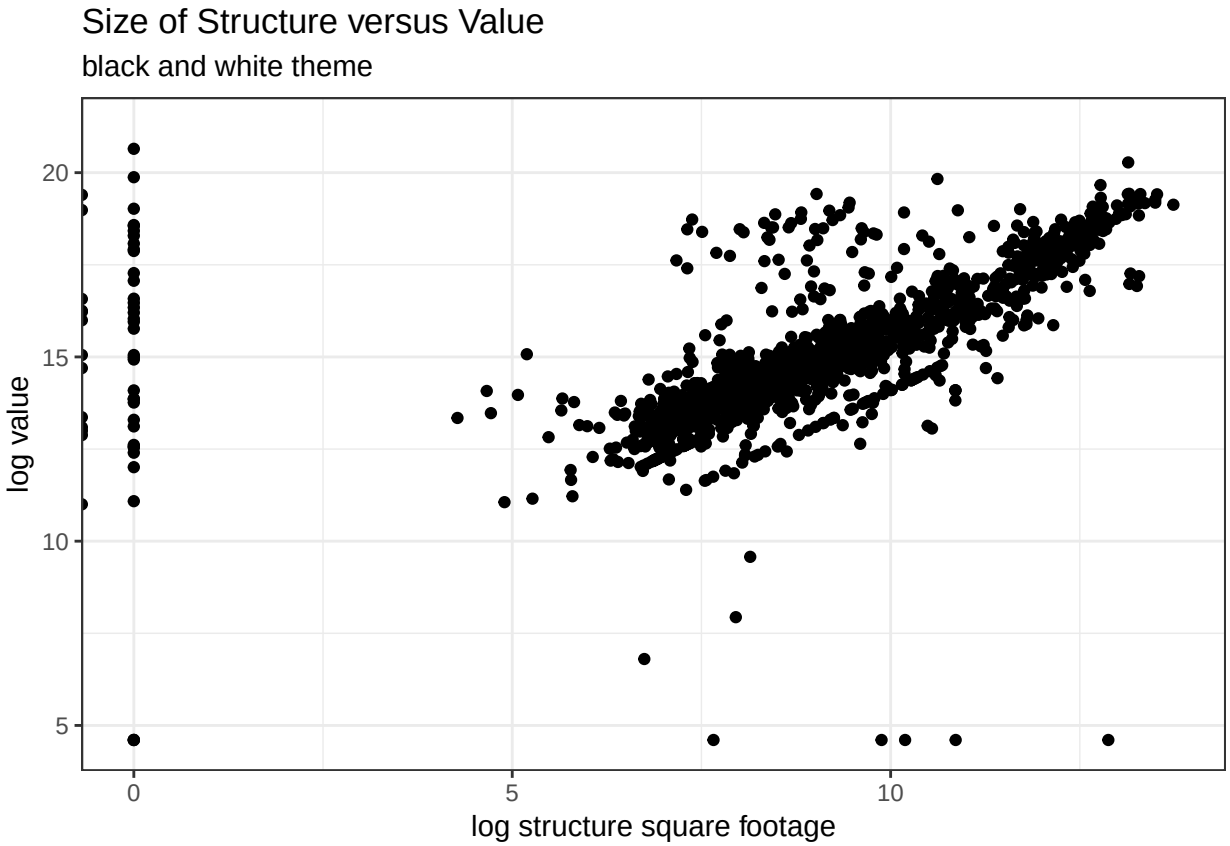
Now we run the function for the black and white theme, and add a subtitle that writes the name of the theme.

```
# run the function for black and white
tester <- trythemes(themer = theme_bw(),
                    et = labs(subtitle = 'black and white theme'))
```


Distribution of Year of Construction
black and white theme
share of observations
0.02
0.01
0.00
1900
1925
1950
1975
2000
Year Constructed




```
## Warning: Removed 64151 rows containing missing values (geom_point).
```

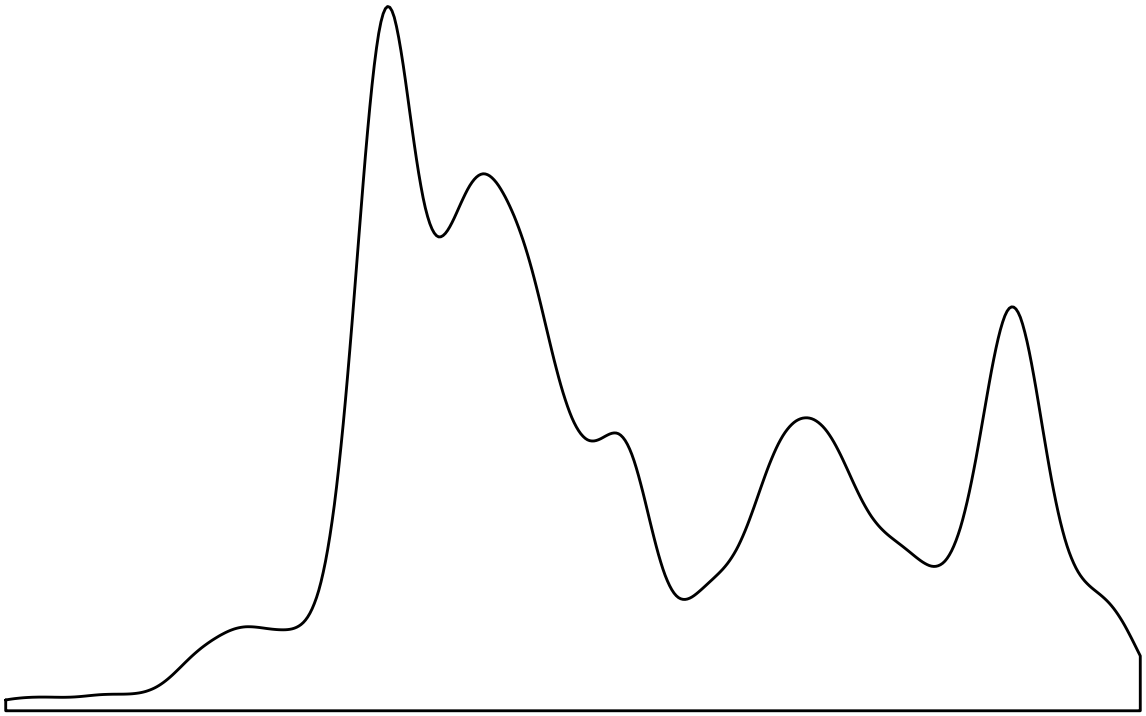


And I can run the function again for the `void` theme, again with a changed subtitle.

```
# run the function for void
tester <- trythemes(themer = theme_void(),
                    et = labs(subtitle = 'void theme'))
```

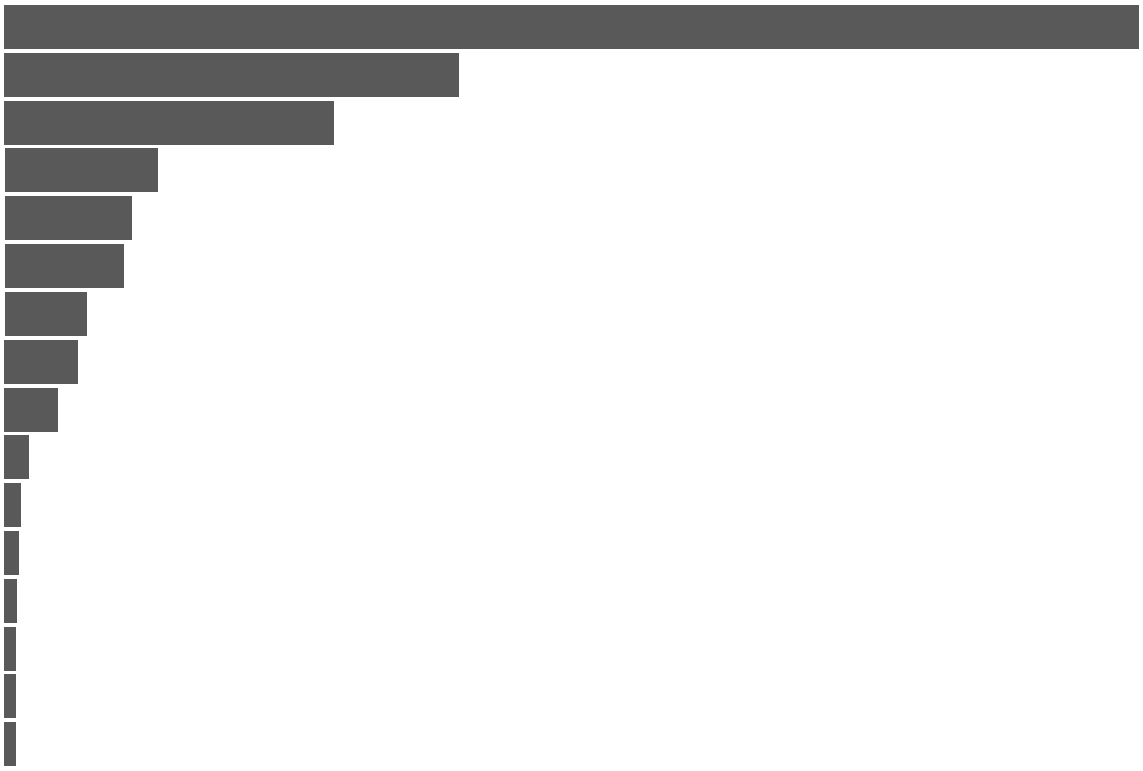
Distribution of Year of Construction

void theme

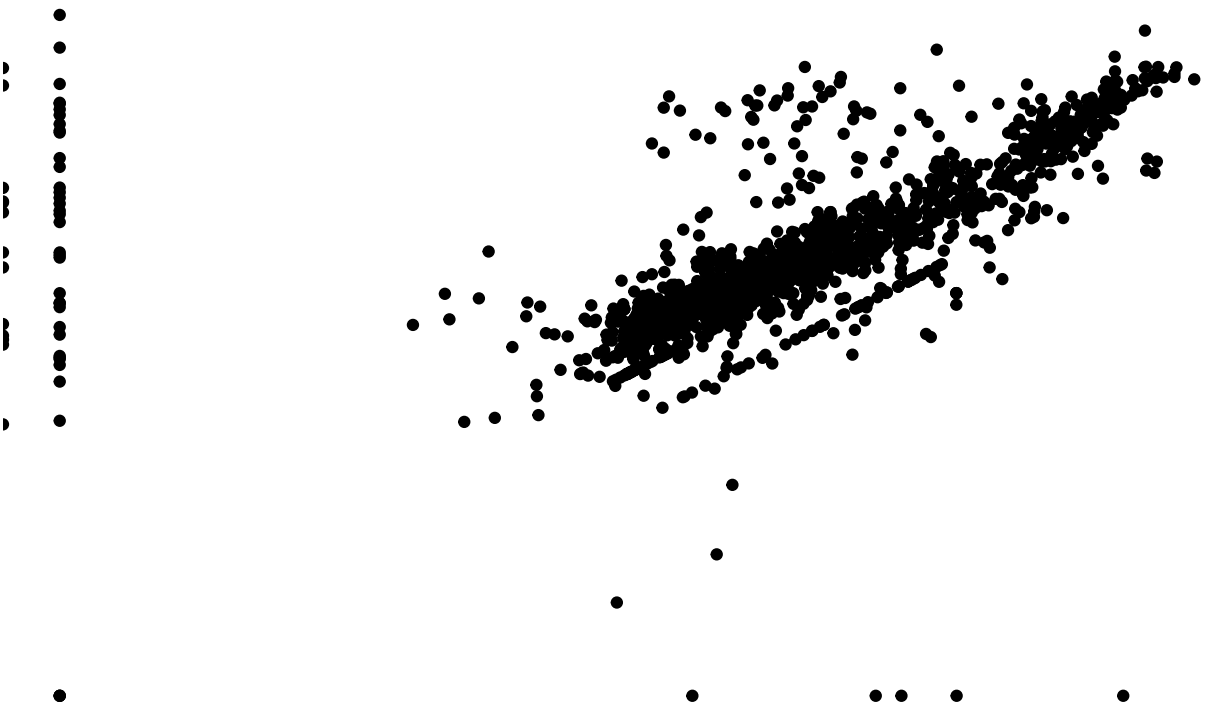


Frequency of Property Type

void theme



```
## Warning: Removed 64151 rows containing missing values (geom_point).
```



C. Theme components

Now we turn to the specific comoponents that make up a theme. Two tutorials that I leaned on in making this one are here and here.

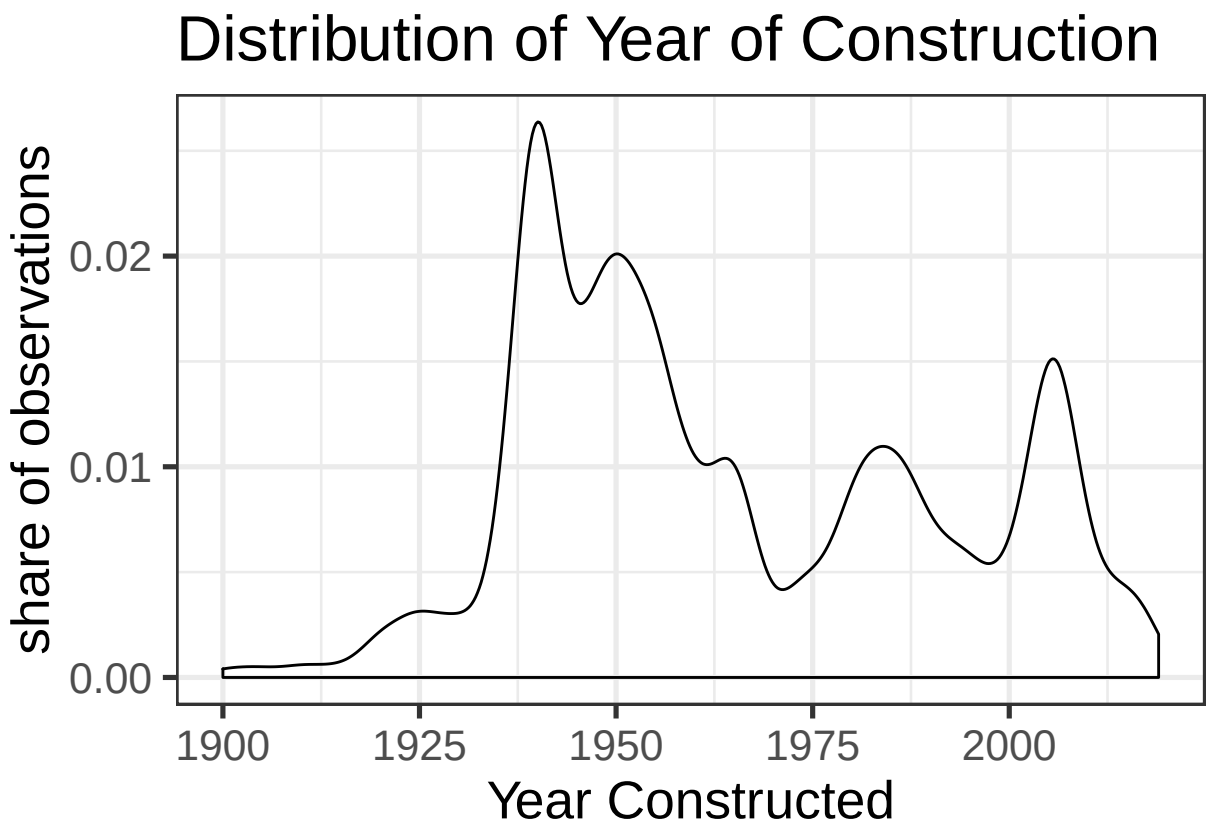
C.1. Modify overall theme elements

We begin with some simpler commands that you can use inside a theme that will change all parts of a theme. These are

- `base_size`: re-scales the font size
- `base_family`: changes font for all elements
- `base_line_size`: re-scales size for all line elements
- `base_rect_size`: re-scales size for all rectangular elements

These are best understood by example, so let's use the `base_size` command with the `theme_bw()`.

```
b1a + theme_bw(base_size = 20)
```



Notice that this command scales everything up – it doesn't just make all font elements the same larger size. You can play with the other commands to see their effect.

C.2. Modify specific elements

Of course, sometimes you want to modify a specific set of elements. In `ggplot` you can modify almost any element of the overall look of the plot. The R page with the (somewhat overwhelming) list of all commands is here.

The basic syntax for modifying any element in a R theme is to tell R that you want to do something in the theme with `theme()`. Then, inside the parentheses, you can modify any of the theme elements.

For example, you can modify the background grid by doing

```
b1a + theme(
  panel.grid = element_blank()
)
```

if you want to get rid of all the grid lines. But R has both major and minor (!) gridlines, and you could modify them separately, as in

```
b1a + theme(
  panel.grid.major = element_line(color = "grey"),
  panel.grid.minor = element_blank()
)
```

Note the comma between theme elements.

C.3. Syntax for modifying theme elements

We modify theme elements with

- `element_text()`
  - for any text, such as titles, axes, etc.
- `element_line()`
  - for axis lines, major and minor grid lines, etc.
- `element_rect()`
  - for rectangular components including plot and panel background
- `element_blank()`
  - to omit an element entirely

So `element_line()` takes options for color, size, linetype, lineend. For example, you could write

```
element_line(size = 0.5, linetype = 'solid', color = "white"),
```

C.4. What you can modify

Almost everything is modifiable

- panel color
- grid lines: major, minor x and y
- plot margins: plot.margin()
- axis lines and ticks
- legend options

You can adjust horizontal and vertical alignment with `hjust` and `vjust`.

D. Make your own theme and use it

Now we get to the best part of themes: making your own personal theme that you can use across multiple charts. In this section, we see how R does it, and then we make our own re-usable theme.

D.1. How does Hadley Wickham do it?

You can find the code for a theme by typing the theme's name without parentheses at the end.

For example, you can see that `theme_bw` is a function and that it is based on `theme_gray`, which is the default R theme.

```
# find the code for theme_bw by typing
theme_bw
```

```
## function (base_size = 11, base_family = "", base_line_size = base_size/22,
##     base_rect_size = base_size/22)
## {
##     theme_grey(base_size = base_size, base_family = base_family,
##         base_line_size = base_line_size, base_rect_size = base_rect_size) %+replace%
##         theme(panel.background = element_rect(fill = "white",
##             colour = NA), panel.border = element_rect(fill = NA,
##             colour = "grey20"), panel.grid = element_line(colour = "grey92"),
##             panel.grid.minor = element_line(size = rel(0.5)),
```

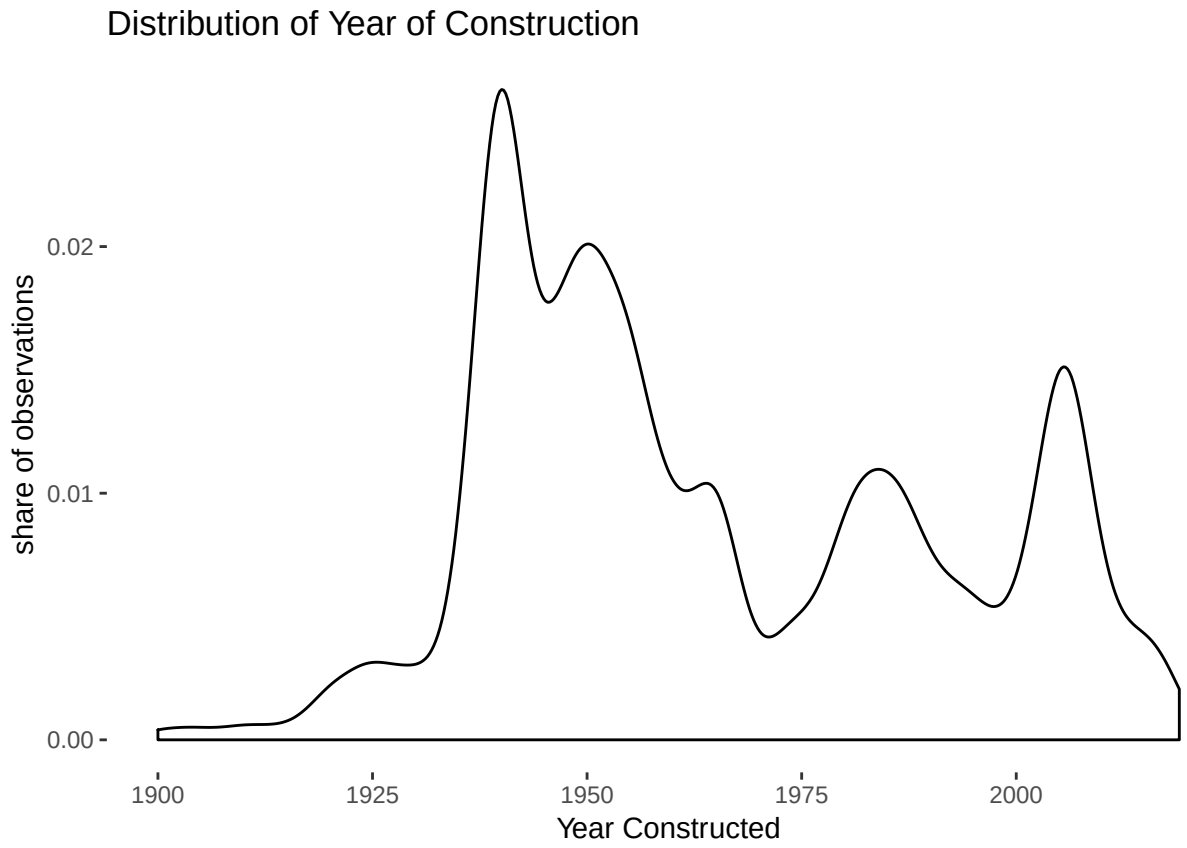
```
##             strip.background = element_rect(fill = "grey85",
##                 colour = "grey20"), legend.key = element_rect(fill = "white",
##                 colour = NA), complete = TRUE)
## }
## <environment: namespace:ggplot2>
```

D.2. Try it yourself

Here is an example of how to create a theme; here I call it `theme_me` and it's created by a function. We base this theme on `theme_grey` and add to that a transparent panel background.

```
theme_me <- function(){
  theme_grey() +
  theme(panel.background = element_rect(fill = "transparent"))
}

b1a + theme_me()
```



Alternatively, I could have used `element_rect()` to modify the panel background. Options here are fill, color, size and linetype.

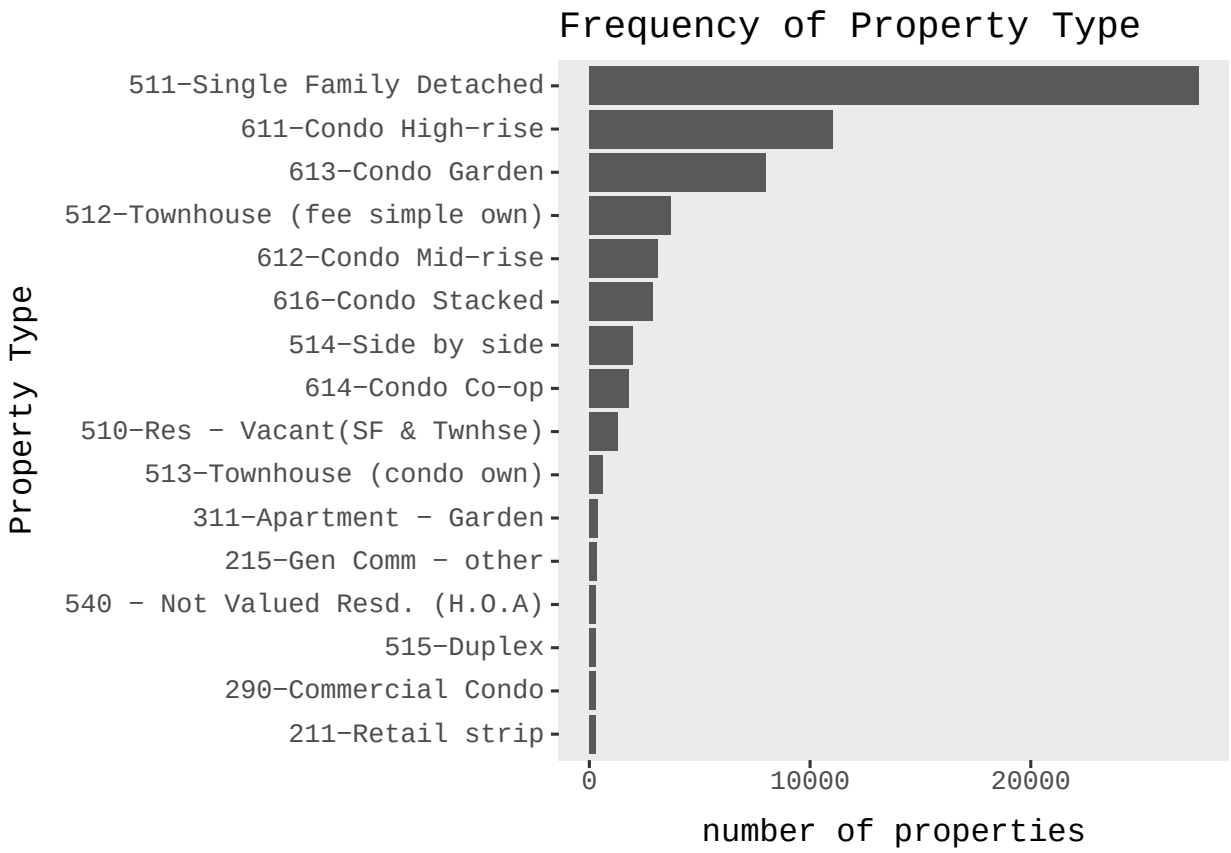
Here's another example of modifications you can make. We change the overall font size (`base_size = 12`) and font type (`base_family = "mono"`), and then get rid of minor gridlines. I also use `margin()` inside `element_text()` to change the spacing between the axis and the axis title. I give the y axis title a larger right margin (`r=10`), and the x axis title a larger top margin (`t=10`).

```
theme_me <- function(){
  theme_gray(base_size = 12, base_family = "mono") +
  theme(
    panel.grid.minor = element_blank(),
    axis.title.y = element_text(margin = margin(t = 0, r = 10, b = 0, l = 0)),
    axis.title.x = element_text(margin = margin(t = 10, r = 0, b = 0, l = 0))
  )
}

b1a + theme_me()
```

Suppose that you'd like to use this new theme for all your charts, but some need special adjustments. This is still possible. In the example below, we use the new theme, but get rid of the major grid lines (`panel.grid.major = element_blank()`).

```
b1b + theme_me() +
  theme(panel.grid.major = element_blank())
```



E. Homework

Make two of your own themes, using some elements we didn't manipulate in this tutorial.

Demostrate both themese on two charts.

Please feel free to make themes that are useful for your presentation or paper.