Tutorial 9: Scatter Plots and Cleveland Dot Plots

Leah Brooks

March 28, 2022

Contents

This week we are learning about two types of charts you can create with `geom_point()`: scatter plots and Cleveland dot plots, also known as lollipop charts.[1]

Scatter plots are very useful for seeing the relationship between two variables. They are ideal for initial exploratory data analysis. They are usually not the best – without substantial editing – for final data presentation to a non-technical audience.

Cleveland dot plots are a way of summarizing data from a scatter plot. To arrive at data suitable for a Cleveland dot plot, we will revisit some techniques we've learned in previous classes: `summarize()` and `pivot_longer()`, along with `group_by()`.

A. Clear environment and load packages

Sometimes it is helpful to get rid of everything in your R environment – all the past dataframes and plots and commands. This is particularly useful when you've made a lot of dataframes and your memory is getting clogged, or when you've made some odd error and want to start fresh. To clear the R environment, type the below. I usually put it at the beginning of my R programs to be sure that I am working on the most recently loaded dataframe.

[1] This is our ninth tutorial, posted for Lecture 10.

```
rm(list=ls())
```

Let's also load packages. This tutorial has no new packages.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

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

```
library(scales)
```

```
##
## Attaching package: 'scales'
```

```
## The following object is masked from 'package:purrr':
##
##     discard
```

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

B. Load and check out data

For this class we're using new data: property tax records from Arlington County.

B.1. Load data

In principle, you used to be able to find the Arlington data here. You'd use the download button at the bottom left to download the data as a .csv. Then load the data using `read.csv()`. However, this link no longer works, and – see the next paragraph.

A previous year's TA had a lot of trouble with this file. So we can all use the same dataset, please download it from here. These are vintage 2019 data.

Now read these data using 'read.csv'.

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

```
## [1] 65791    46
```

I have 65,791 observations – hopefully you do, too.

B.2. Explore a little bit

Each observation in this data frame is a property in Arlington County, VA. Broadly speaking, this is the information the county collects on each individual property. Counties in the US (in a few cases, other jurisdictions) are in charge of land ownership records and property taxation.

As these data are new to us (and as Arlington doesn't seem to provide a dictionary – though if you find one, let me know!), let's explore these data a bit before moving on.

I use `str()`, `names()`, `table()` and `summary()` to look at values in this dataframe. I encourage you to use these commands to look at variables other than those I've listed below.

```
names(arl.p)
```

```
##  [1] "IvwPropertyAssessmentHistoryKey"   "ProvalLrsnId"
##  [3] "RealEstatePropertyCode"            "MasterRealEstatePropertyCode"
##  [5] "ReasPropertyStatusCode"            "PropertyClassTypeCode"
##  [7] "PropertyClassTypeDsc"              "PropertyStreetNbrNameText"
##  [9] "PropertyStreetNbr"                 "PropertyStreetNbrSuffixCode"
## [11] "PropertyStreetDirectionPrefixCode" "PropertyStreetName"
## [13] "PropertyStreetTypeCode"            "PropertyStreetDirectionSuffixCode"
## [15] "PropertyUnitNbr"                   "PropertyCityName"
## [17] "PropertyZipCode"                   "ZoningDescListText"
## [19] "TradeName"                         "PropertyYearBuilt"
## [21] "GrossFloorAreaSquareFeetQty"       "EffectiveAgeYearDate"
## [23] "NumberOfUnitsCnt"                  "StoryHeightCnt"
## [25] "ValuationYearDate"                 "CommercialPropertyTypeDsc"
## [27] "CondoModelName"                    "CondoStyleName"
## [29] "FinishedStorageAreaSquareFeetQty"  "StorageAreaSquareFeetQty"
## [31] "UnitNbr"                           "PropertyKey"
## [33] "ReasPropertyOwnerKey"              "ArlingtonStreetKey"
## [35] "PropertyExpiredInd"                "CommercialInd"
## [37] "DistrictNbr"                       "TaxExemptionTypeDsc"
## [39] "CondominiumProjectName"            "TaxBalanceAmt"
## [41] "AssessedYearDate"                  "TotalAssessedAmt"
## [43] "AssessmentDate"                    "AssessmentChangeReasonTypeDsc"
## [45] "ImprovementValueAmt"               "LandValueAmt"
```

```
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     : chr  "01001007" "01001009" "01001010" "01001012" ...
##  $ ReasPropertyStatusCode           : chr  "A" "A" "A" "A" ...
##  $ PropertyClassTypeCode            : int  511 511 511 511 511 511 511 511 511 511 ...
##  $ PropertyClassTypeDsc             : chr  "511-Single Family Detached" "511-Single Family Detached"
##  $ PropertyStreetNbrNameText        : chr  "3007 N ROCHESTER ST" "6547 WILLIAMSBURG BLVD" "6541 WILL
##  $ PropertyStreetNbr                : int  3007 6547 6541 3518 3526 3530 3538 3544 3550 3562 ...
##  $ PropertyStreetNbrSuffixCode      : chr  "" "" "" "" ...
##  $ PropertyStreetDirectionPrefixCode: chr  "N" "" "" "N" ...
##  $ PropertyStreetName               : chr  "ROCHESTER" "WILLIAMSBURG" "WILLIAMSBURG" "SOMERSET" ...
##  $ PropertyStreetTypeCode           : chr  "ST" "BLVD" "BLVD" "ST" ...
##  $ PropertyStreetDirectionSuffixCode: chr  "" "" "" "" ...
##  $ PropertyUnitNbr                  : chr  "" "" "" "" ...
##  $ PropertyCityName                 : chr  "ARLINGTON" "ARLINGTON" "ARLINGTON" "ARLINGTON" ...
##  $ PropertyZipCode                  : int  22213 22213 22213 22213 22213 22213 22213 22213 22213 222
##  $ ZoningDescListText               : chr  "R-10" "R-10" "R-10" "R-10" ...
##  $ TradeName                        : chr  "" "" "" "" ...
##  $ 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        : chr  "" "" "" "" ...
##  $ CondoModelName                   : chr  "" "" "" "" ...
##  $ CondoStyleName                   : chr  "" "" "" "" ...
##  $ 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                          : chr  "" "" "" "" ...
##  $ 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               : chr  "False" "False" "False" "False" ...
##  $ CommercialInd                    : chr  "False" "False" "False" "False" ...
##  $ DistrictNbr                      : chr  "0G" "0G" "0G" "0G" ...
##  $ TaxExemptionTypeDsc              : chr  "" "" "" "" ...
##  $ CondominiumProjectName           : chr  "" "" "" "" ...
##  $ 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                   : chr  "1/1/2019 12:00:00 AM" "1/1/2019 12:00:00 AM" "1/1/2019 1
##  $ AssessmentChangeReasonTypeDsc    : chr  "01- Annual" "01- Annual" "01- Annual" "01- Annual" ...
##  $ ImprovementValueAmt              : int  1151000 400900 380000 709800 87500 482900 105900 136300 1
##  $ LandValueAmt                     : int  790900 652300 659500 696700 672900 706500 682500 671900 6
```

```
table(arl.p$NumberOfUnitsCnt)
```

```
##
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##    8   35    1    3    4    5    2    2   10    4    6    8    3    3    5    5
##   16   17   18   19   20   21   22   23   25   26   27   28   29   30   31   32
##    4    2    8    3    5    5    5    3    1    3    2    2    1    4    3    4
##   33   34   36   37   38   40   41   42   44   45   46   47   48   49   51   56
##    4    1    2    4    3    4    5    3    1    1    1    1    2    2    2    1
##   57   62   63   64   65   66   67   68   70   74   75   77   78   79   81   82
##    1    1    2    3    1    1    1    1    1    3    2    2    1    1    1    2
##   83   84   85   87   88   90   91   92   93   94   95   97   99  100  101  102
##    1    2    1    1    2    3    1    1    3    1    1    1    1    5    1    1
##  104  105  108  109  110  111  112  114  115  116  119  120  122  125  126  128
##    2    1    2    3    3    1    1    2    1    2    2    1    1    1    1    3
##  132  134  135  138  142  143  144  146  147  148  151  152  153  154  155  161
##    1    4    1    1    1    1    1    1    1    1    1    3    1    1    1    1
##  162  163  168  170  172  173  178  181  183  184  186  187  188  189  191  193
##    2    1    1    1    1    1    1    3    2    2    1    1    3    1    1    1
##  194  197  198  199  200  204  205  208  210  212  214  217  218  219  220  221
##    1    1    2    1    1    2    2    1    1    2    2    1    2    1    2    2
##  224  225  227  228  229  231  234  235  237  238  241  244  247  249  250  252
##    1    1    1    1    1    1    1    1    2    1    1    3    1    3    2    1
##  253  254  255  257  258  261  262  265  266  267  269  270  272  273  274  280
##    1    1    1    2    1    2    2    2    1    2    1    1    1    1    1    2
##  299  300  302  303  308  314  317  318  321  325  326  330  336  337  342  344
##    2    2    1    1    1    1    1    1    1    2    1    2    1    1    1    1
##  345  348  350  360  361  363  365  366  369  377  378  383  385  400  406  407
##    1    1    1    2    1    1    1    2    1    1    2    1    1    1    2    1
##  410  411  412  435  437  440  442  452  453  454  458  464  474  476  483  491
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
```

```
##  499  500  504  509  534  539  564  571  575  577  580  597  647  686  699  714
##    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
##  717  834  900 1075 1295
##    1    1    3    1    1
```

```
table(arl.p$PropertyYearBuilt)
```

```
##
##    0 1000 1742 1750 1832 1836 1840 1845 1848 1850 1860 1862 1865 1866 1867 1870
##    9  129    1    1    1    2    1    1    1    3    4    1    1    1    1    2
## 1875 1876 1880 1881 1887 1888 1889 1890 1891 1892 1893 1894 1895 1896 1898 1899
##    1    1   14    1    1    2    1    6    1    1    1    2    2    2    2    1
## 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915
##  117    5    5    1   99   42    6   11   12   21  131   13   19   30   21   90
## 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931
##   13   17   37   43  483   37   82   97  270  446  109   93  124  189  444   52
## 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945 1946 1947
##  103   80  152  508  354  728 1093 1933 4789 1449  743  208 1764  481  626 1790
## 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959 1960 1961 1962 1963
## 1182  638 2223 1329 1111  731  745 2493  662  370  691  754  637  664  359  268
## 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975 1976 1977 1978 1979
##  644 1643  801  205  154  279  101  103  132  456  580   96  415  128  213  729
## 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
##  697  707  503 1060  546  418 1033  852  242  838  163  271  727  186  350  610
## 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011
##  352  129  324  173  464  270  378  848  362 2038 1533  915  722  431  321  203
## 2012 2013 2014 2015 2016 2017 2018 2019
##  238  365  216  316  277  315  308    4
```

```
summary(arl.p$PropertyYearBuilt)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##       0    1944    1956    1963    1987    2019    3392
```

```
summary(arl.p$TaxBalanceAmt)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
## -40509.1  -3917.3  -2227.3  -2691.3   -556.8   1797.8    64390
```

From this I learn that there are many variables in this dataframe, that most structures have few units (from `NumberOfUnitsCnt`), but a few have quite a few, that the median structure in Arlington was built in 1956 (`PropertyYearBuilt`), and that most properties are current on their taxes (I take `TaxBalanceAmt` to be the amount outstanding on property taxes; `NA` means the property is current).

C. Basic scatters and drawing attention

We now turn to some basic scatter plots to show the relationship between two variables. We'll rely on `geom_point()`, which is similar to the `ggplot` commands we've used so far. We then move to two methods for calling out particular areas or conditions of interest.

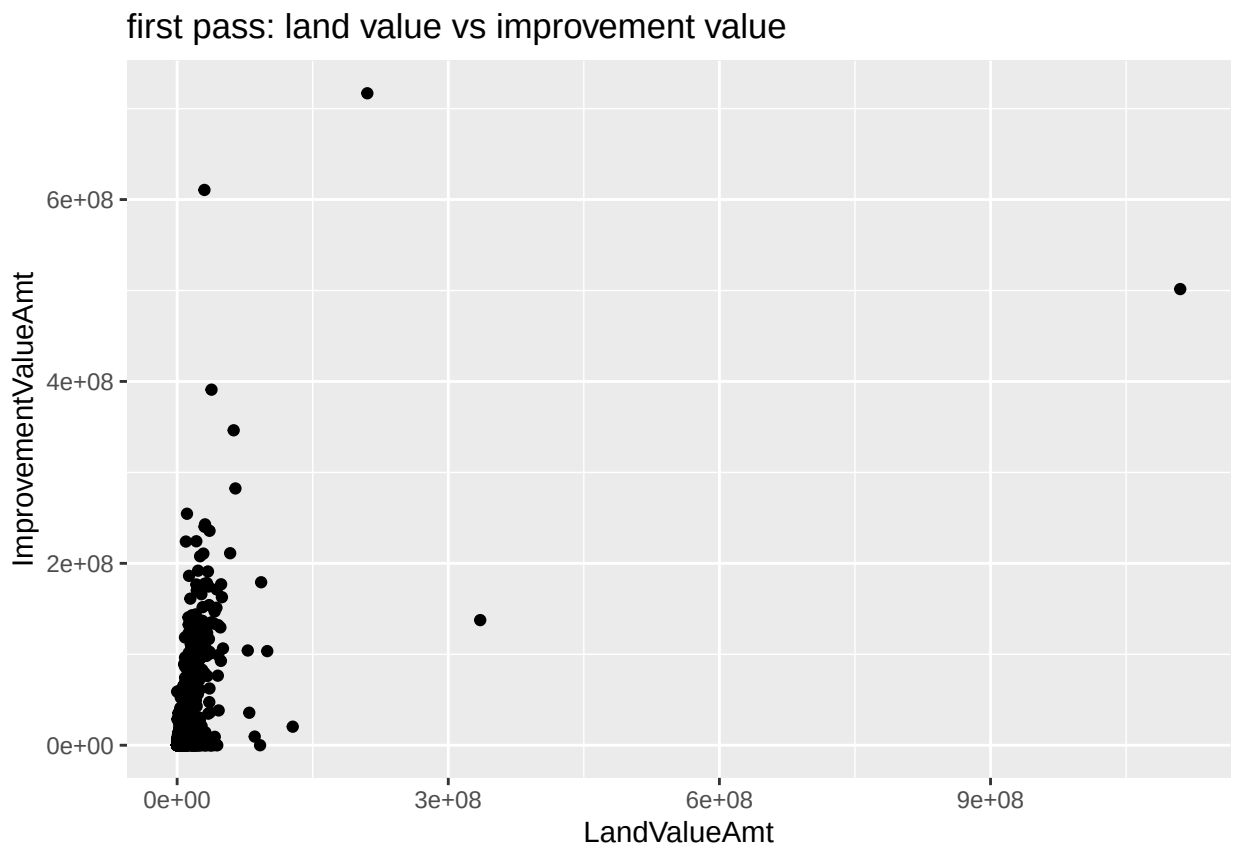
C.1. Basic scatters

We begin by looking at the correlation between the assessed value of land and the assessed value of the improvements on that land (the structure(s)). The assessed value is the value that the county assessor gives to the property for purposes of taxation. It is distinct from the market value, which is the value for which a property sells on the open market.

Generally, economists anticipate that higher value land should have higher valued structures. We can see whether the data are consistent with this hypothesis using a scatter plot.

We start by simply plotting these two measures, specifying the dataframe and the `x` (land value) and `y` (improvement value) amounts.

```
c0 <-
  ggplot() +
  geom_point(data = arl.p,
             mapping = aes(x = LandValueAmt, y = ImprovementValueAmt)) +
  labs(title = "first pass: land value vs improvement value")
c0
```

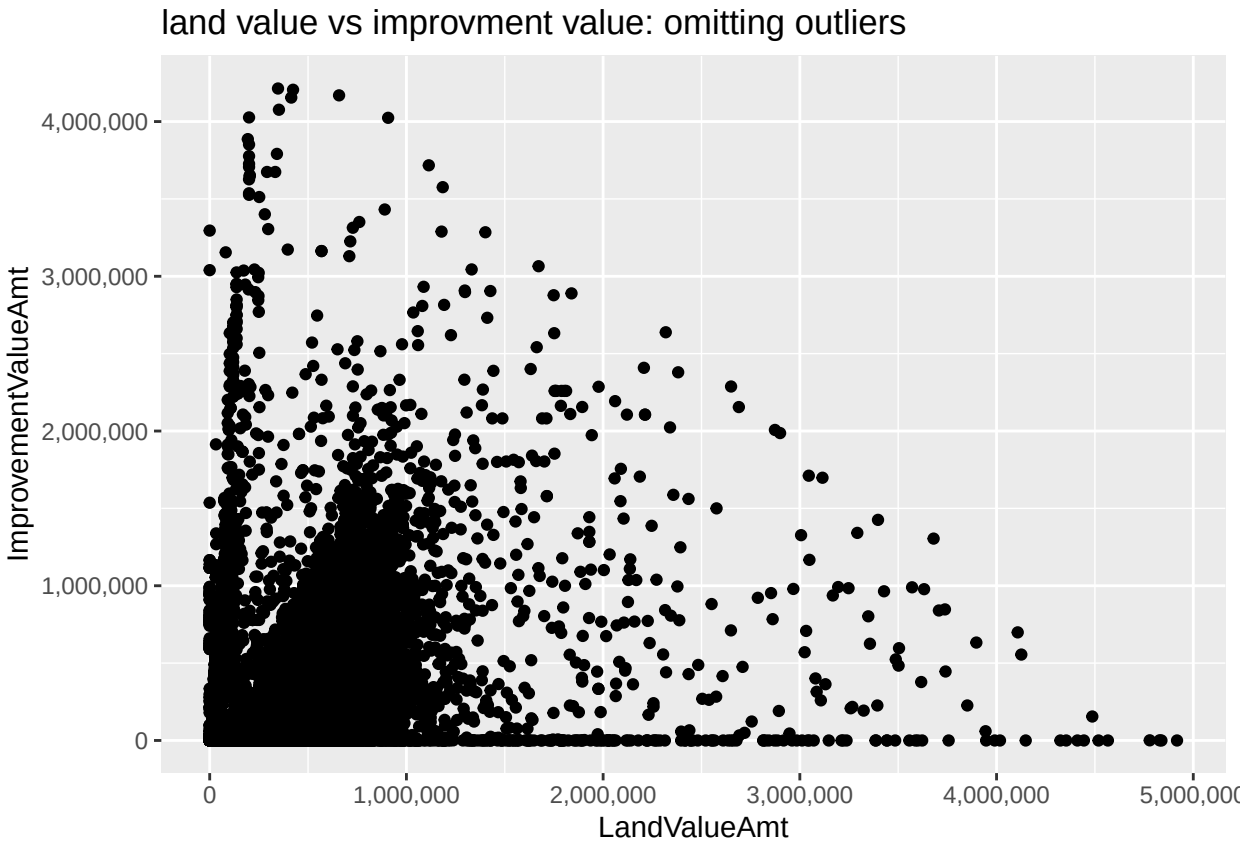


This plot is illegible! The large values at the high end of the distribution make the part of the distribution where the majority of observations lie invisible. In addition, the numbers on the plot are in scientific notation, which is hard to read. We can fix the problem of outliers in two ways.

We'll begin by just omitting the outliers. This isn't good statistical practice if the outliers are true data, but if we are predominantly concerned about the relationship for most observations (and we're clear about what we're doing) this is ok.

You can modify the sample multiple different ways. You can create a new dataframe that is limited by some conditions, and just call that dataframe. You can subset the dataframe directly in the `ggplot` command, or you can use `limits = c(min(xxx), max(xxx))` inside of the `scale_x_continuous()` command. Here I subset the dataframe inside the `ggplot` command.

```
# (1) omit outliers, add commas
c0 <-
  ggplot() +
  geom_point(data = arl.p[which(arl.p$TotalAssessedAmt < 5000000),],
             mapping = aes(x = LandValueAmt, y = ImprovementValueAmt)) +
  scale_x_continuous(label=comma) +
  scale_y_continuous(label=comma) +
  labs(title = "land value vs improvment value: omitting outliers")
c0
```



Another way to "squish" a distribution is by taking logs. The log function spreads out the small numbers and relatively shrinks the big numbers. (If this makes no sense, google "graph of log function'' and take a look at how the log function transforms data.)

We want to take the log of two variables. You could write a log command twice, but you can also be a more efficient programmer in case you decide to come back and make more logged variables. To program efficiently, don't use a loop – and in fact, I'm reasonably confident you cannot use a loop for this problem because of difficulties getting R to recognize a variable name made from a loop variable.

Thankfully, there is a quick, elegant solution (that took me over an hour to figure out). Create two new columns based on the log of two existing columns, using `paste0` to create the new names.

```
# (2) take logs
tolog <- c("LandValueAmt","ImprovementValueAmt")
arl.p[paste0("ln.",tolog)] <- log(arl.p[tolog])
```

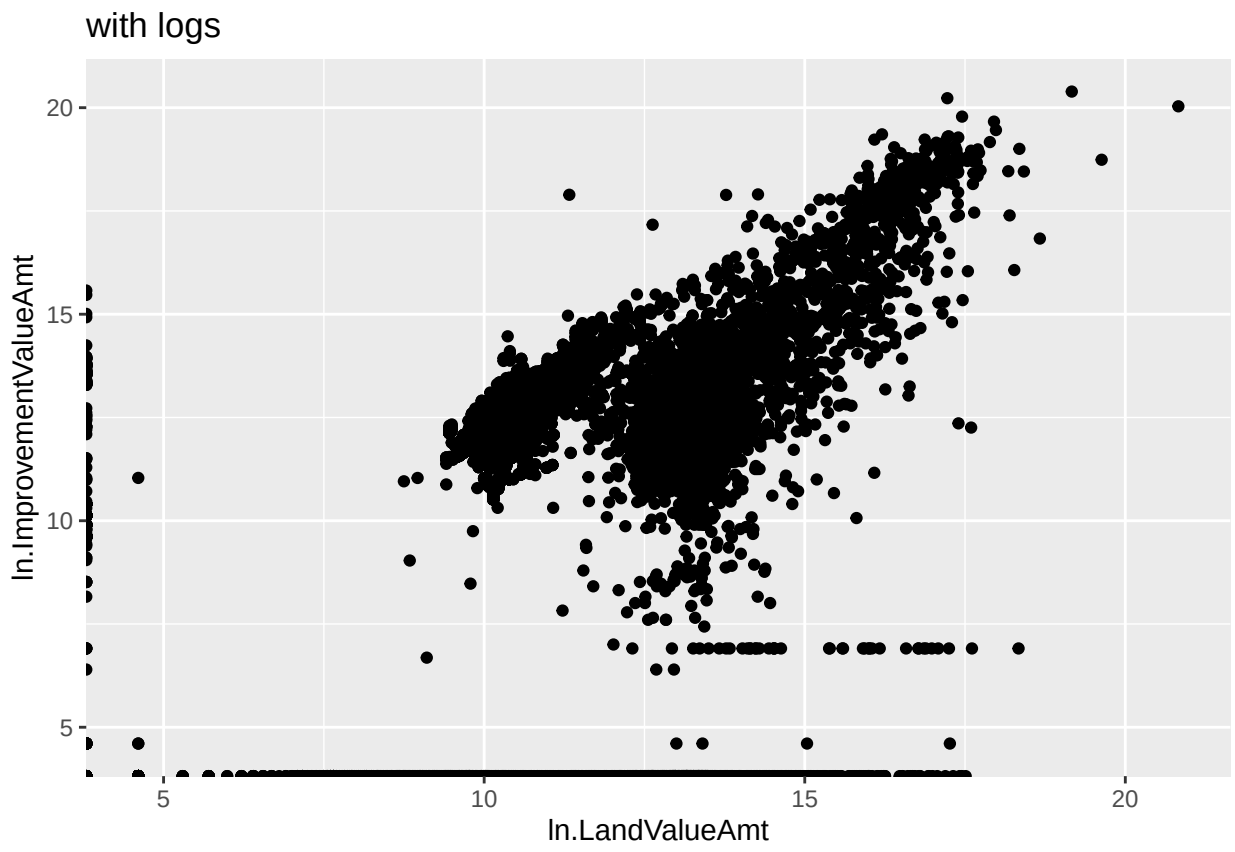
What this is actually doing is saying

```
arl.p["ln.LandValueAmt","ln.ImprovementValueAmt"] <-
  log(arl.p["LandValueAmt","ImprovementValueAmt"])
```

For large dataframes, this type of processing is also faster than loops.

Now create the logged version of the graph using these new variables (alternatively, you could have logged them in the `ggplot` command).

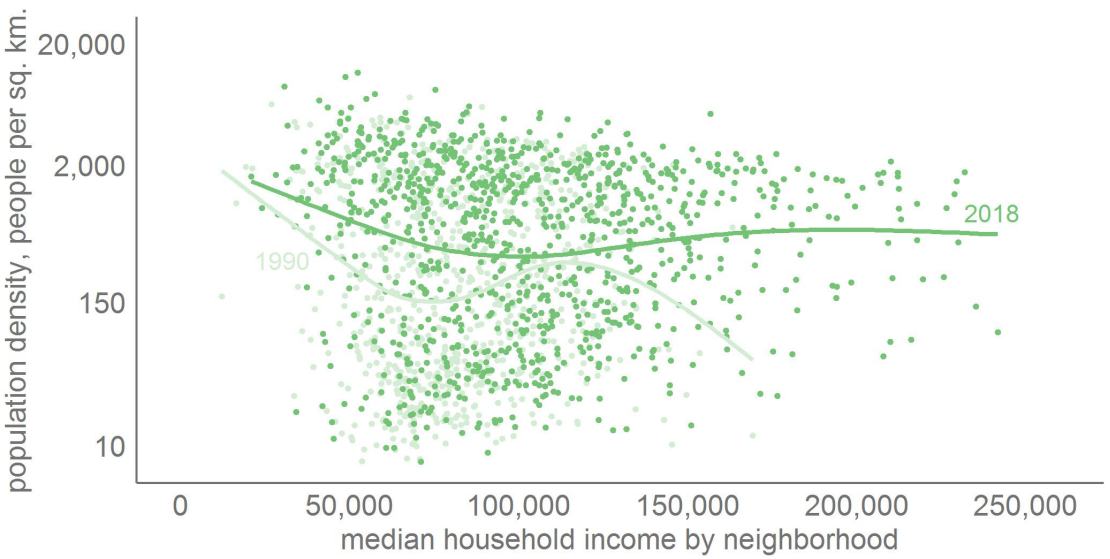
```
cln <-
  ggplot() +
  geom_point(data = arl.p,
             mapping = aes(x = ln.LandValueAmt, y = ln.ImprovementValueAmt)) +
  labs(title = "with logs")
cln
```



This is clearly better than the first one in the sense of seeing all the data. Of course, logged values are also more difficult to understand!

One way to deal with this issue of logged variables is to label the axis of the logged variables with the levels, rather than the logged values. As an example, my graph below, using different data, does this. It shows the relationship between neighborhood population density and neighborhood median income for the exurban jurisdictions of the greater Washington area. The y axis values are logged, but I report the level values, rather than the logged ones. Note that the spacing isn't exactly equal between the axis numbers (by construction).

```
## Warning in knitr::include_graphics("H:/center_for_washington_area_studies/
## state_of_the_capital_region/r_output/2020/2020_report/ch01/
## ch01_g5_20200328.jpg"): It is highly recommended to use relative paths for
## images. You had absolute paths: "H:/center_for_washington_area_studies/
## state_of_the_capital_region/r_output/2020/2020_report/ch01/ch01_g5_20200328.jpg"
```



C.2. Call out particular areas

If you're using a scatterplot for presentation purposes, you're frequently interested in calling out an area or group. In this section we'll call out properties with delinquent taxes to see if they systematically differ from properties at large. I'll continue using the logged values here, since I found that graph easier to intepret.

We begin by making a marker for tax delinquency, using the `TaxBalanceAmt` variable. Using `summary()`, I see that `NA`s are non-delinquent, so I make those 0, and set all others to 1. I check my results with `table()`.

```
# lets call out those delinquent on their taxes
summary(arl.p$TaxBalanceAmt)
```

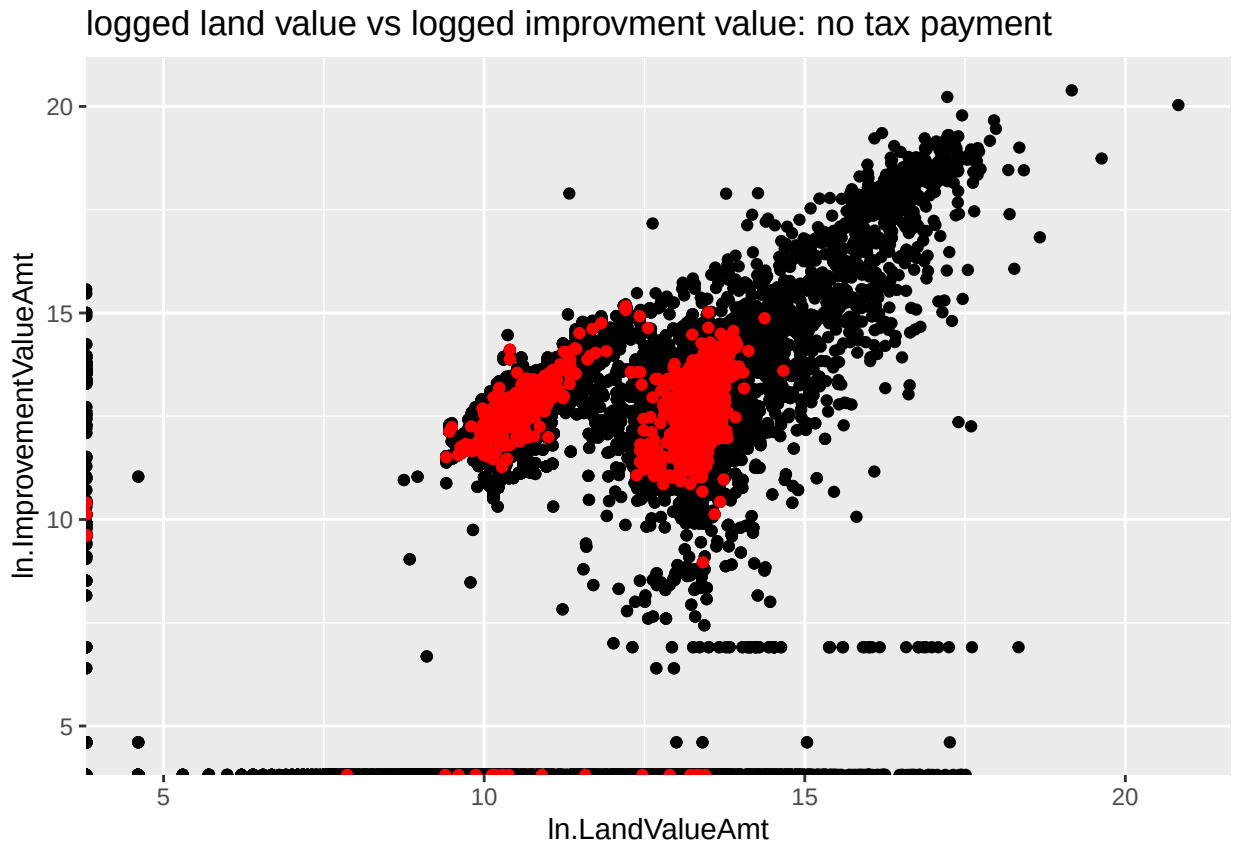
```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.     NA's
## -40509.1  -3917.3  -2227.3  -2691.3   -556.8   1797.8    64390
```

```
arl.p$tax.late <- ifelse(is.na(arl.p$TaxBalanceAmt) == TRUE,0,1)
table(arl.p$tax.late)
```

```
##
##     0     1
## 64390  1401
```

Now I repeat the previous chart, but with a second layer of red points for the late tax observations. It's important to put this call second, as R stacks points, beginning with the first layer. If this tax delinquent layer were first, it would be invisible since it would be covered by the following layer. I use `color = "red"` inside the `aes()` command to show the delinquent points.

```
latetax <-
  ggplot() +
  geom_point(data = arl.p, aes(x = ln.LandValueAmt, y = ln.ImprovementValueAmt)) +
  geom_point(data = arl.p[which(arl.p$TotalAssessedAmt < 5000000),],
             mapping = aes(x = ln.LandValueAmt, y = ln.ImprovementValueAmt)) +
  geom_point(data = arl.p[which(arl.p$TotalAssessedAmt < 5000000 & arl.p$tax.late == 1),],
             mapping = aes(x = ln.LandValueAmt, y = ln.ImprovementValueAmt),
             color = "red") +
  labs(title = "logged land value vs logged improvment value: no tax payment")
latetax
```



This figure shows that delinquent assessments are much more likely for lower-valued properties. Smaller dots might make this relationship more clear visually.

C.3. Small multiples

Another way to call attention to comparisons in a distribution is to use small multiples. Below I use `facet_grid()` to compare the relationship between land and improvement value for residential and non-residential property. Below I make a graph with two rows; you can also make columns, or a grid with multiple rows and columns.

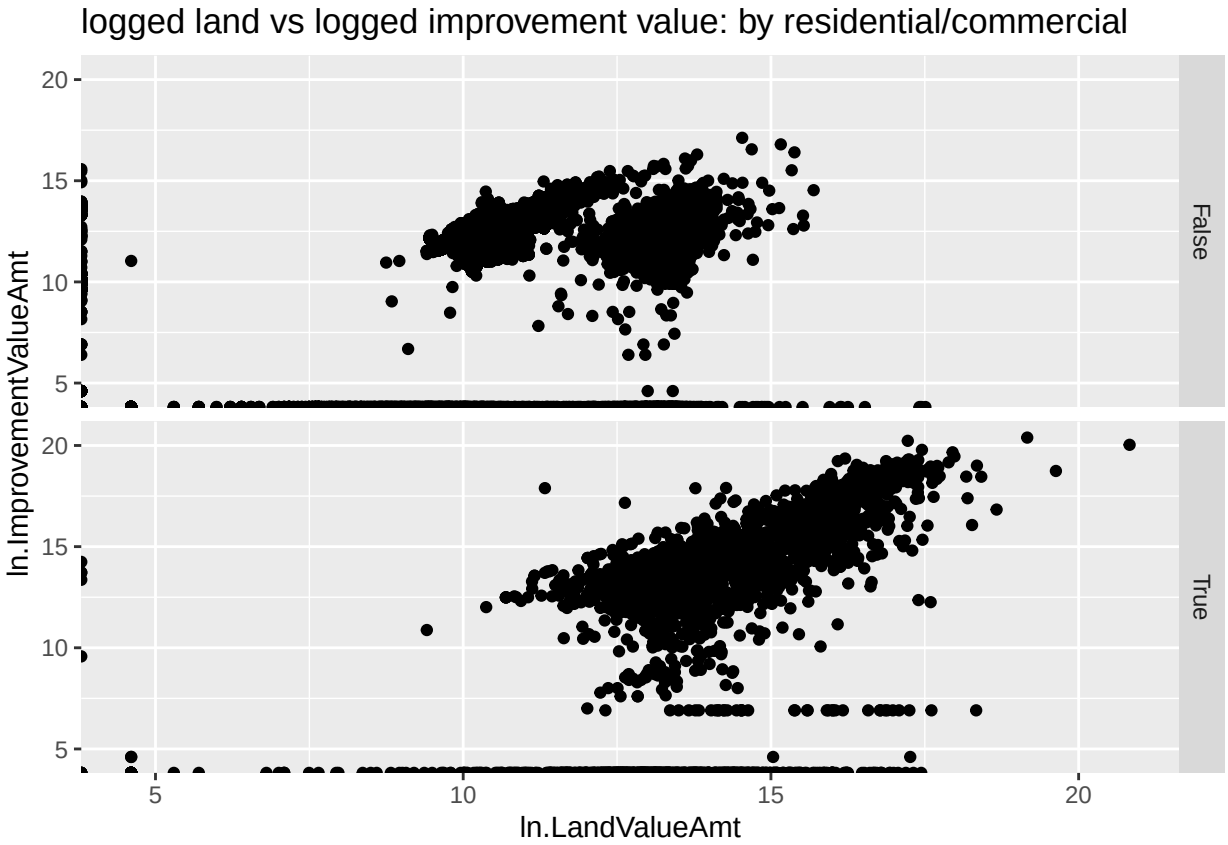
I don't think this is the best use of small multiples, but I did want to show you what they do.

I use the `CommercialInd` variable, which takes on the values of `TRUE` and `FALSE` for this step.

```
# residential vs commercial
table(arl.p$CommercialInd)
```

```
##
## False  True
## 63302  2489
```

```
h1 <-
  ggplot(data = arl.p,
         mapping = aes(x = ln.LandValueAmt, y = ln.ImprovementValueAmt)) +
  geom_point() +
  facet_grid(rows = vars(CommercialInd)) +
  labs(title = "logged land vs logged improvement value: by residential/commercial")
h1
```



I ask you to think about what these graphs tell us in the homework.

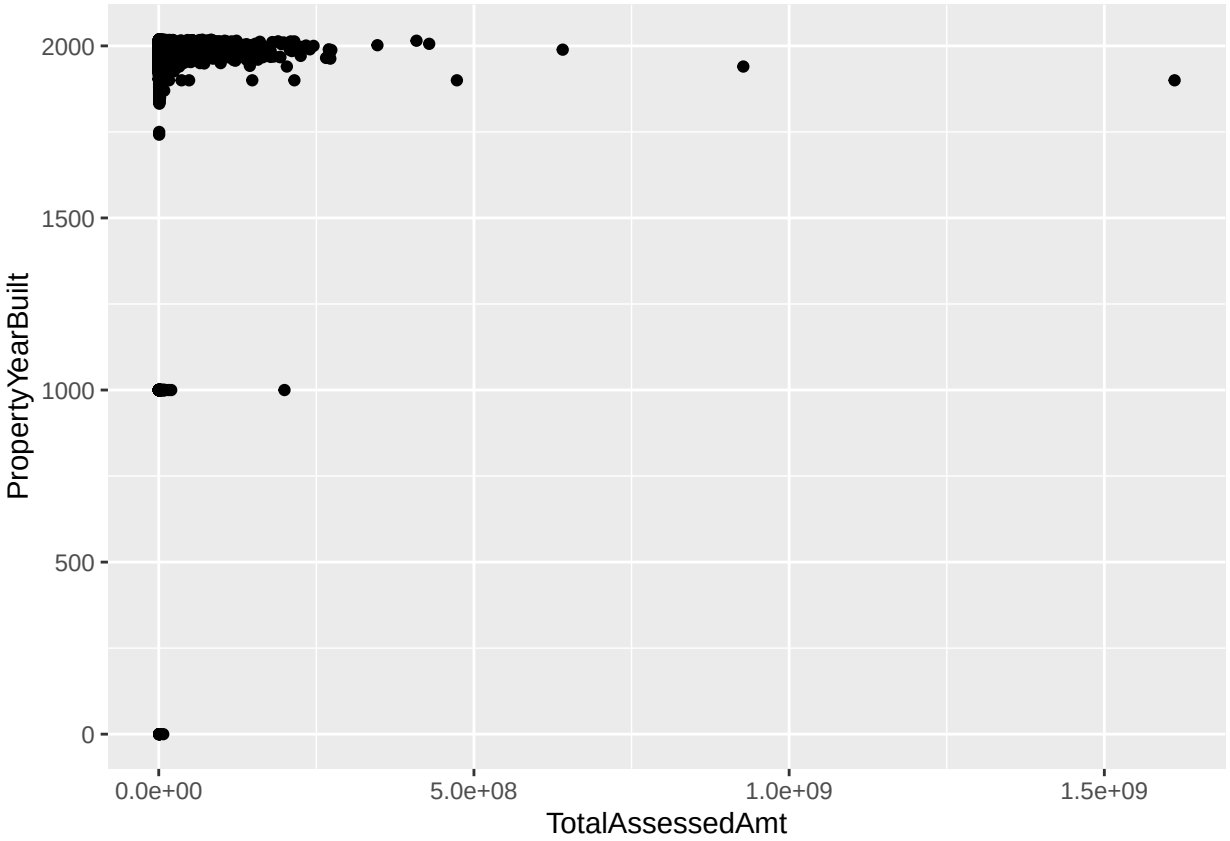
D. A limited scatter: year built and value

We now edge toward looking at a Cleveland dot plot by considering the relationship between the year a structure was built (`PropertyYearBuilt`) and its value (`TotalAssessedAmt`).

This is a `ggplot geom_point()` as before, but recall that the year takes on discrete (as in an integer), rather than continuous, values.

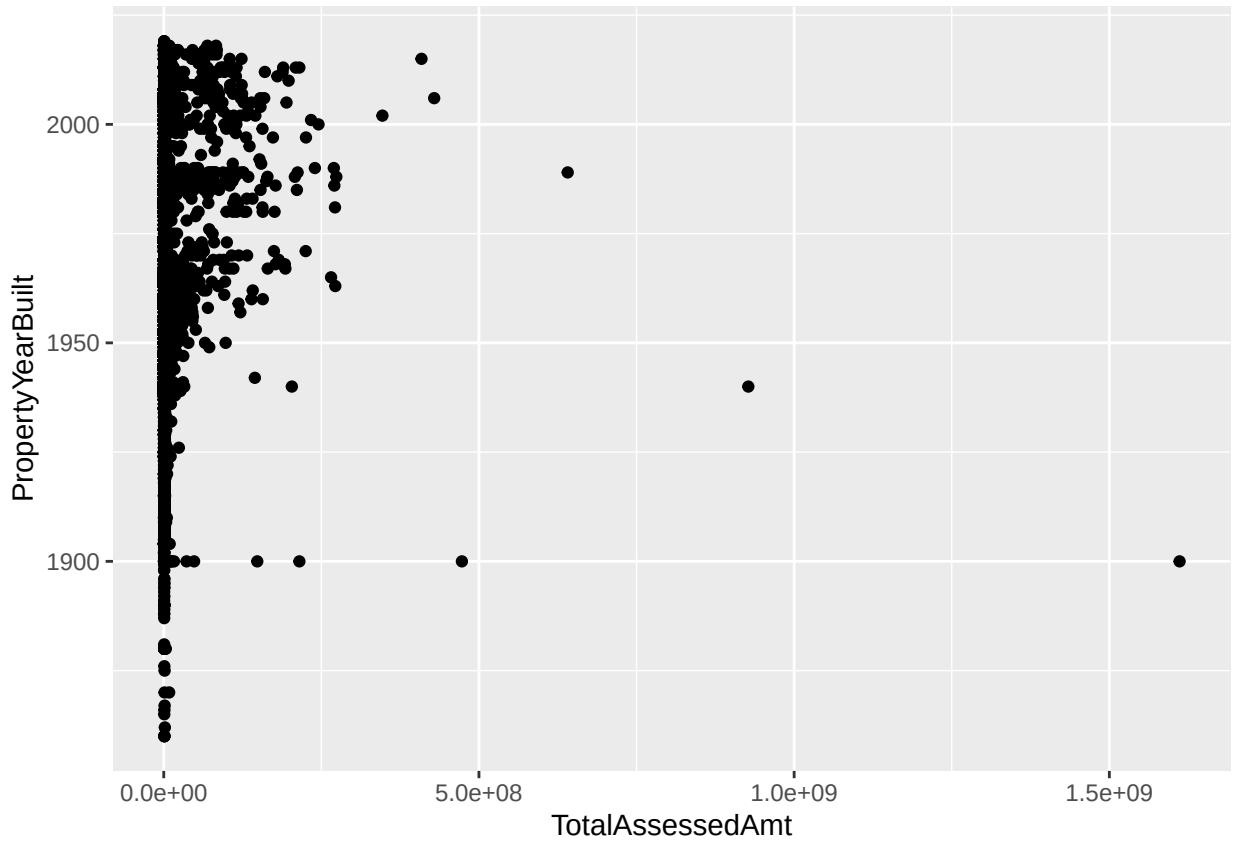
```
# distribution of assessed value by year built
c1 <-
  ggplot() +
  geom_point(data = arl.p,
             mapping = aes(x = TotalAssessedAmt, y = PropertyYearBuilt))
c1
```

```
## Warning: Removed 3392 rows containing missing values (geom_point).
```



This chart immediately points out some data problems. There are no structures in Arlington from the year 1000. Let's limit to years 1850 and onward for a better looking chart.

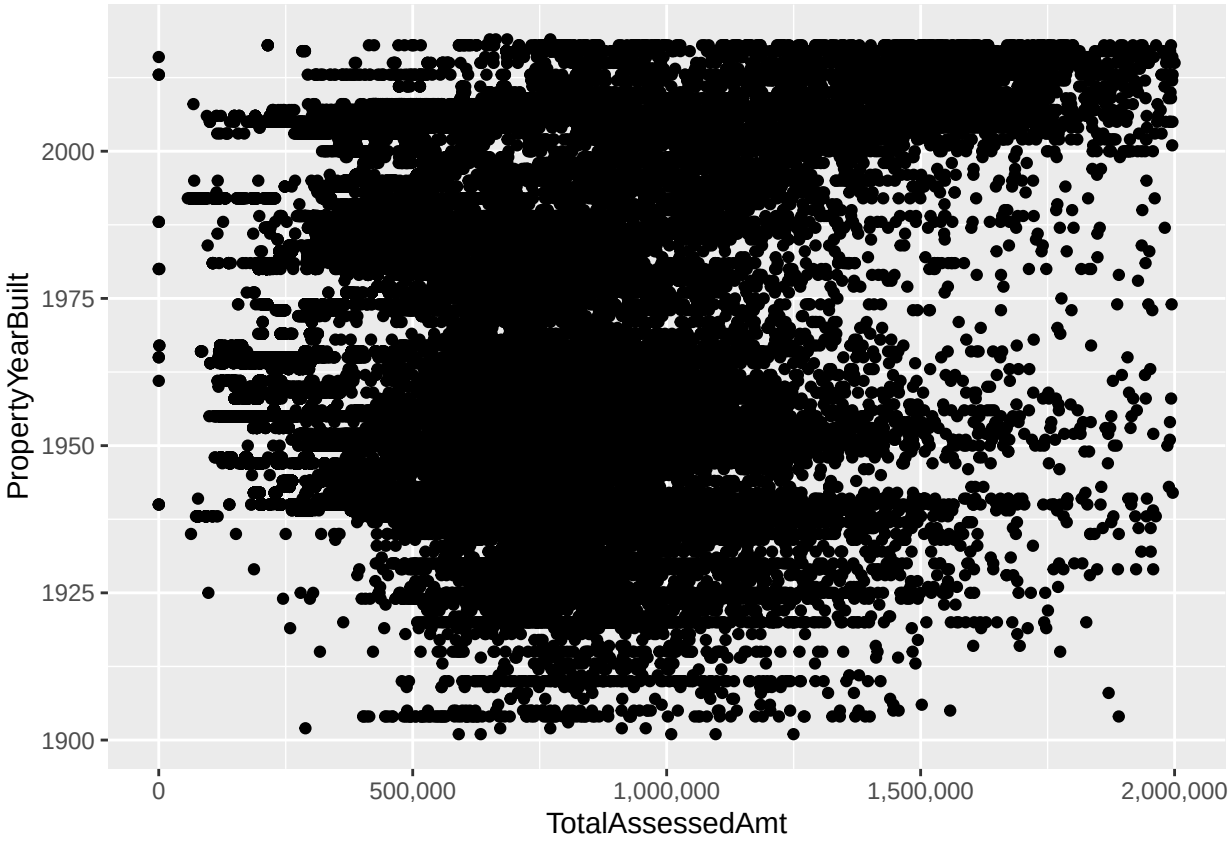
```
# distribution without some crazy years
c2 <-
  ggplot() +
  geom_point(data = arl.p[which(arl.p$PropertyYearBuilt >1850),],
             mapping = aes(x = TotalAssessedAmt, y = PropertyYearBuilt))
c2
```



This fix highlights the need to limit the amount of assessed values we consider; for purposes of graphing I set this maximum at 2,000,000, using the `limits()` option in `scale_x_continuous()`.

```
# distribution without some crazy assessed values
c4 <-
  ggplot() +
  geom_point(data = arl.p[which(arl.p$PropertyYearBuilt >1900),],
             mapping = aes(x = TotalAssessedAmt, y = PropertyYearBuilt)) +
  scale_x_continuous(label=comma, limits=c(min(0),
                                           max(2000000)))
c4
```

```
## Warning: Removed 1312 rows containing missing values (geom_point).
```



Regardless of these fixes, these charts suffer from having so many dots that it's hard to understand what's going on. This is where you need to mix statistics and graphics. We'll calculate the median assessed value in each year and then limit the plot to this information.

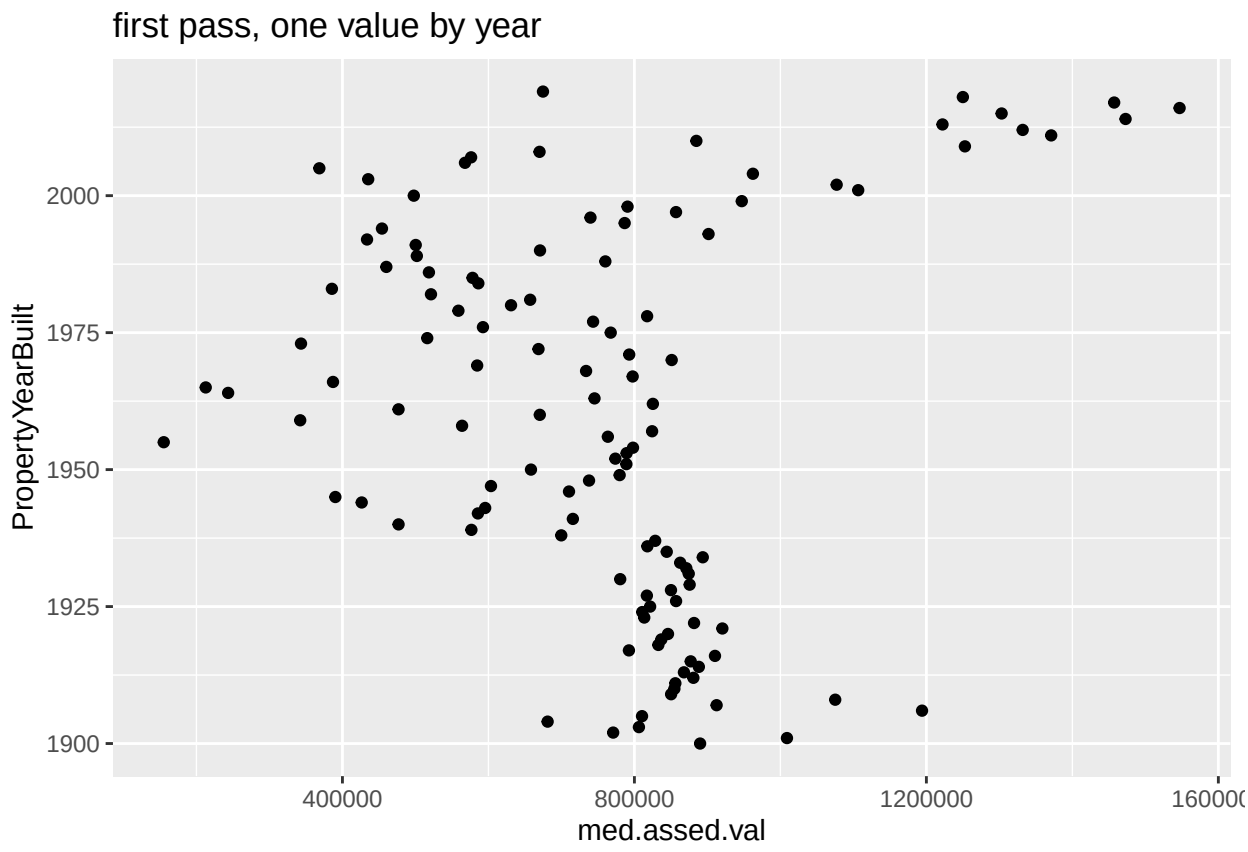
We begin with the combination of `group_by()` and `summarize()` that we've used before to get a dataframe that has one observation by `PropertyYearBuilt`.

```
# find the points
arl.p.byy <- group_by(.data = arl.p, PropertyYearBuilt)
byyear <- summarize(.data = arl.p.byy,
                    med.assed.val = median(TotalAssessedAmt),
                    med.imp.val = median(ImprovementValueAmt),
                    med.land.val = median(LandValueAmt))
head(byyear)
```

```
## # A tibble: 6 x 4
##   PropertyYearBuilt med.assed.val med.imp.val med.land.val
##               <int>         <dbl>       <dbl>        <dbl>
## 1                 0        681400      598200        70700
## 2              1000        437800      263300       352800
## 3              1742        642800       30800       612000
## 4              1750        693200      147800       545400
## 5              1832        958800      327500       631300
## 6              1836       1122500      199050       923450
```

With these data in hand, we turn to the plot:

```
# plot them
e1 <-
  ggplot() +
  geom_point(data = byyear[which(byyear$PropertyYearBuilt >= 1900),],
             mapping = aes(x = med.assed.val,
                           y = PropertyYearBuilt)) +
  labs(title = "first pass, one value by year")
e1
```



This is easier to understand, but doesn't do a good job highlighting the horizontal distance, which is the key value in this chart. In fact, the default horizontal distance doesn't even start at 0, which makes the comparison across years misleading.

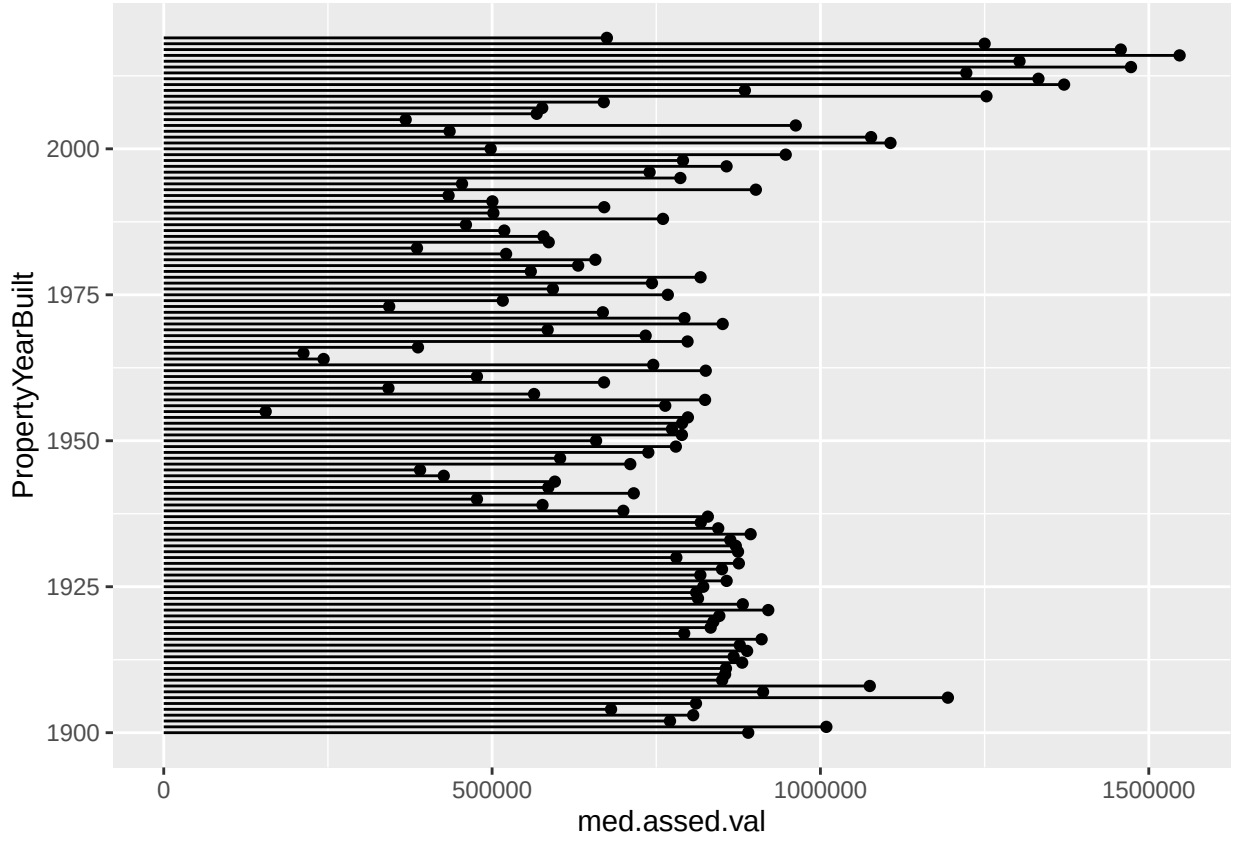
A lollipop graph – a version of the Cleveland dot plot – does a better job. In general, I would recommend this for fewer observations that we have here, though this might look fine in a large format. We also use `geom_segment()`, which takes `xend` and `yend` in addition to `x` and `y`. This draws a segment from `(x,xend)` to `(y,yend)`. In our case, we want a straight line, so `y` and `yend` are the same. The segment starts at `x=0` and ends at the median assessed value for that year.

```
# lollipop version
e1 <-
  ggplot() +
  geom_point(data = byyear[which(byyear$PropertyYearBuilt >= 1900),],
             mapping = aes(x = med.assed.val,
                           y = PropertyYearBuilt)) +
  geom_segment(data = byyear[which(byyear$PropertyYearBuilt >= 1900),],
               mapping = aes(x = 0, xend = med.assed.val,
```

```
                              y = PropertyYearBuilt, yend = PropertyYearBuilt))
  labs(title = "first pass, one value by year")
```

```
## $title
## [1] "first pass, one value by year"
##
## attr(,"class")
## [1] "labels"
```

```
e1
```



E. More Cleveland dot plots

Now we move to a more serious consideration of the power of dot plots, with many thanks to this excellent tutorial.

E.1. Just one point

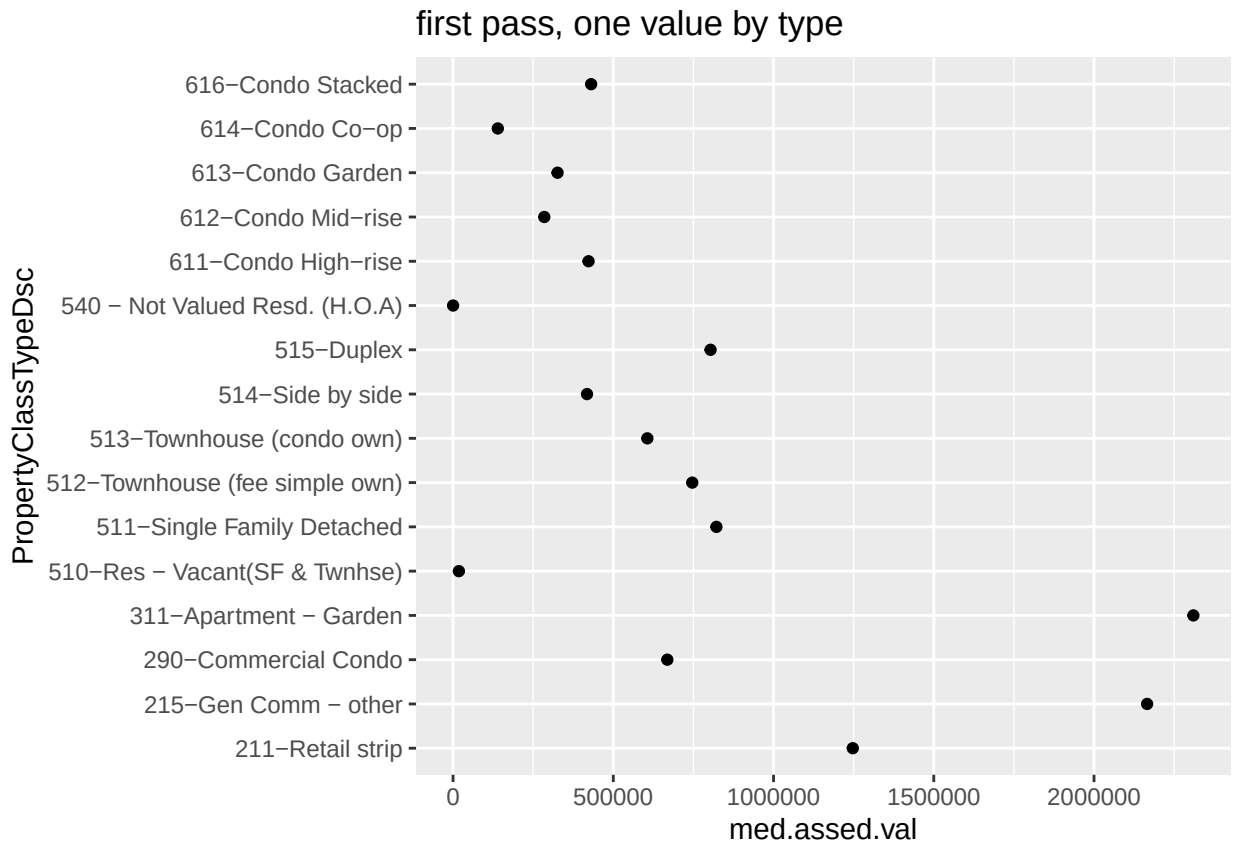
We need a smaller set of categories to make a reasonable looking chart, so we will look at the `PropertyClassTypeDsc` variable (words for the `PropertyClassTypeCode` variable) and limit our analysis to codes with more than 200 properties. These codes describe the type of property – single family, townhome, commercial, etc.

We start by finding the median assessed value by property class using `group_by()` and `summarize()`.

```
# find the points
arl.p.bypc <- group_by(.data = arl.p, PropertyClassTypeDsc)
bypc <- summarize(.data = arl.p.bypc,
                  med.assed.val = median(TotalAssessedAmt),
                  med.imp.val = median(ImprovementValueAmt),
                  med.land.val = median(LandValueAmt),
                  obs = n())
```

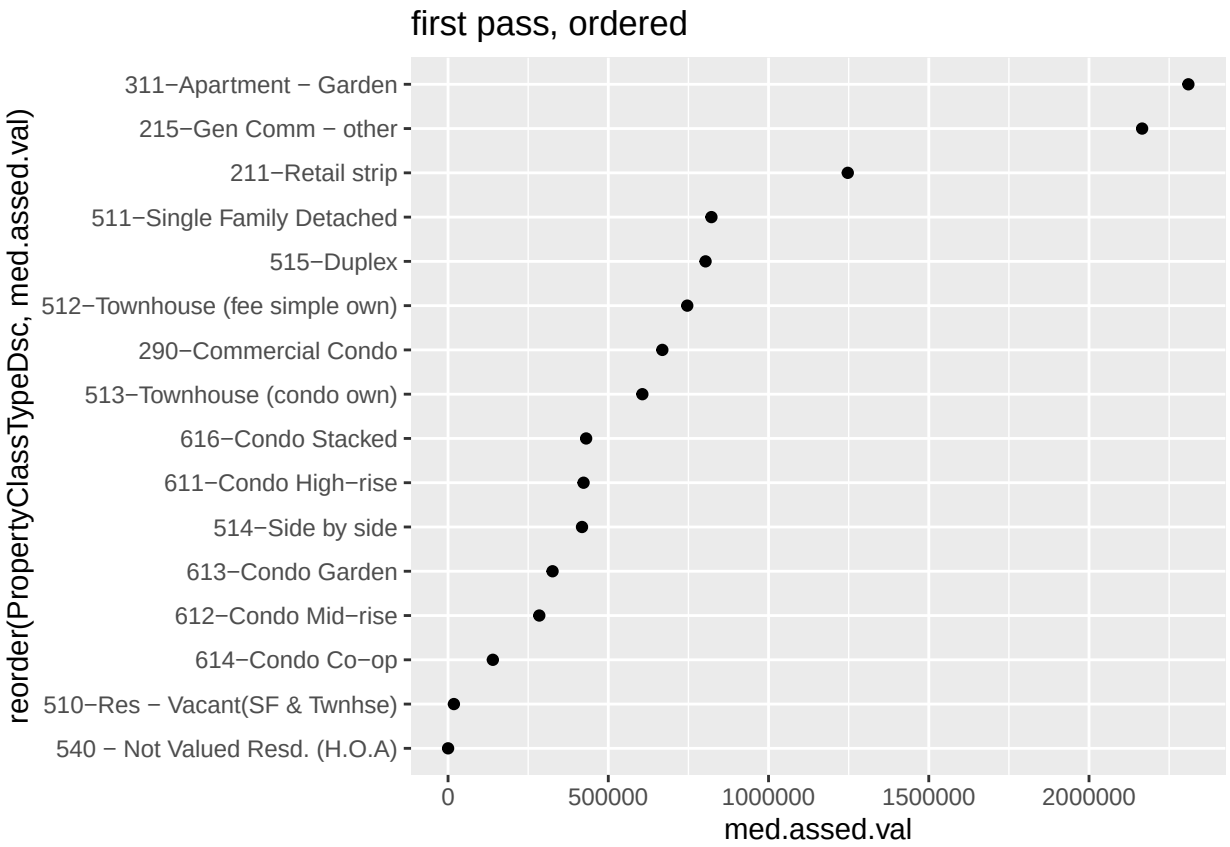
If you look at the `bypc` dataframe, you'll see that there are many property types with few observations (small `obs`). For legibility – and to concentrate on the bulk of the distribution, we graph types with more than 200 properties. We use `geom_point()` as a starting point.

```
# plot them if there are > 200 obs
e1 <-
  ggplot() +
  geom_point(data = bypc[which(bypc$obs > 200),],
             mapping = aes(x = med.assed.val,
                           y = PropertyClassTypeDsc)) +
  labs(title = "first pass, one value by type")
e1
```



This is not very helpful because the points are not in order. We can order points many ways; here I use `reorder()` when I call the `y` variable. The syntax for this command is `reorder(graph variable, ordering variable)`.

```
# lets get them in order!
e1 <-
  ggplot() +
  geom_point(data = bypc[which(bypc$obs > 200),],
             mapping = aes(x = med.assed.val,
                           y = reorder(PropertyClassTypeDsc, med.assed.val))) +
  labs(title = "first pass, ordered")
e1
```



E.2. Two points

The real power of this dot plot is in making comparisons between two measures, as in the *Wall Street Journal* graphic we discussed in class. However, we've just calculated one statistic for each property type; to revise the chart, we'll need to calculate more than one. Below I use the `quantile()` command to find the 25th and 75th percentiles of the assessed value distribution by year. This command has at least two required parts: the variable from which to find the distribution, and the point in the distribution for which you're looking.

```
# let find the 25th and 75th percentiles
# two points
arl.p.bypc <- group_by(.data = arl.p, PropertyClassTypeDsc)
bypc <- summarize(.data = arl.p.bypc,
                  p50.assed.val = median(TotalAssessedAmt, na.rm = TRUE),
                  p25.assed.val = quantile(TotalAssessedAmt, 0.25, na.rm = TRUE),
                  p75.assed.val = quantile(TotalAssessedAmt, 0.75, na.rm = TRUE),
                  obs = n())
```

To use the best of R's tricks for graphing, we need to make these data long. As we have in previous tutorials, we'll use `pivot_longer()` to do this. Look in the previous tutorials for a more in-depth explanation of this command, or see the cheat sheet here.

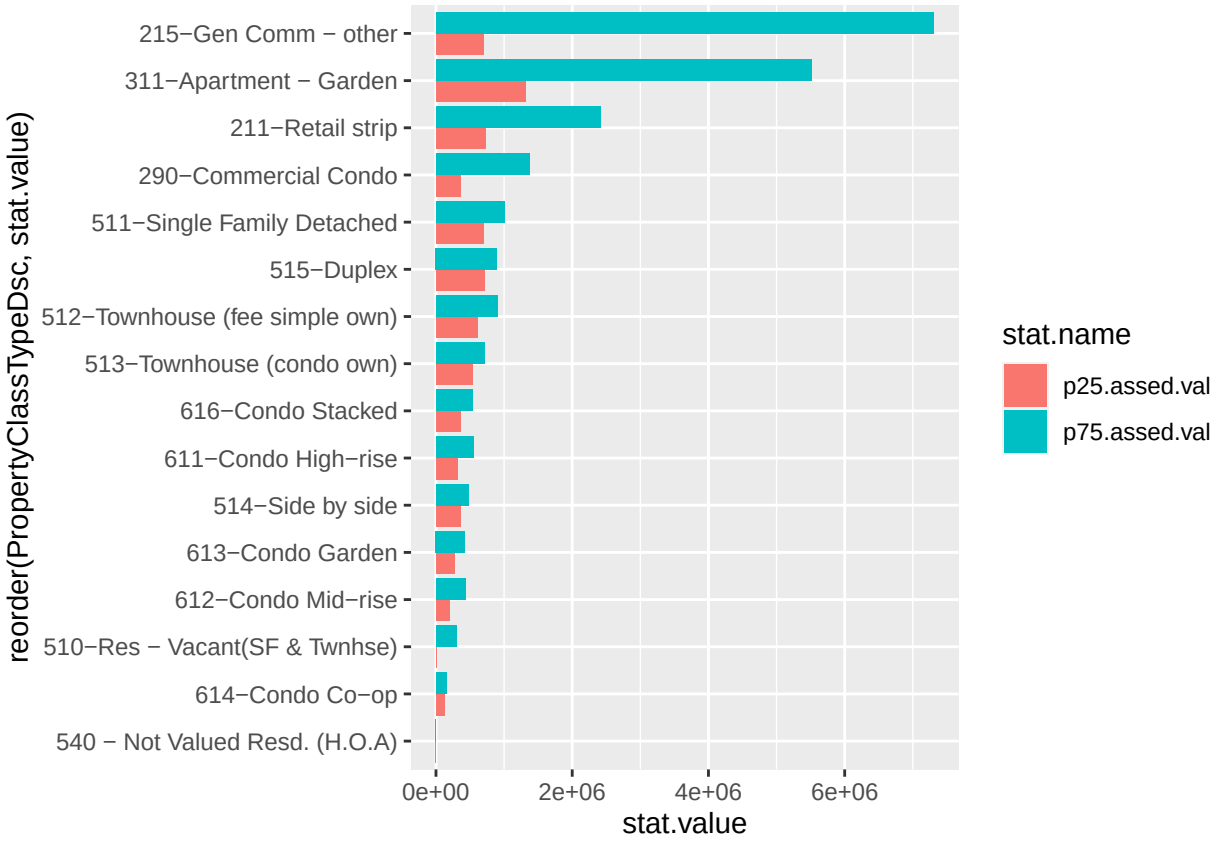
```
# keep in mind that -obs- now refers to the # of the entire category
bypc.long <- pivot_longer(data = bypc,
                          cols = c("p50.assed.val","p25.assed.val","p75.assed.val"),
                          names_to = "stat.name",
                          values_to = "stat.value")
# pull out the statistic
```

```
bypc.long$pnum <- substring(bypc.long$stat.name,2,3)
head(bypc.long)
```

```
## # A tibble: 6 x 5
##   PropertyClassTypeDsc             obs stat.name     stat.value pnum
##   <chr>                          <int> <chr>              <dbl> <chr>
## 1 100-Off Bldg-VacLand-no s.plan    10 p50.assed.val     722100 50
## 2 100-Off Bldg-VacLand-no s.plan    10 p25.assed.val     122375 25
## 3 100-Off Bldg-VacLand-no s.plan    10 p75.assed.val    1046050 75
## 4 101-Off Bldg-VacLand-site plan    56 p50.assed.val    1943300 50
## 5 101-Off Bldg-VacLand-site plan    56 p25.assed.val     272125 25
## 6 101-Off Bldg-VacLand-site plan    56 p75.assed.val    6073950 75
```

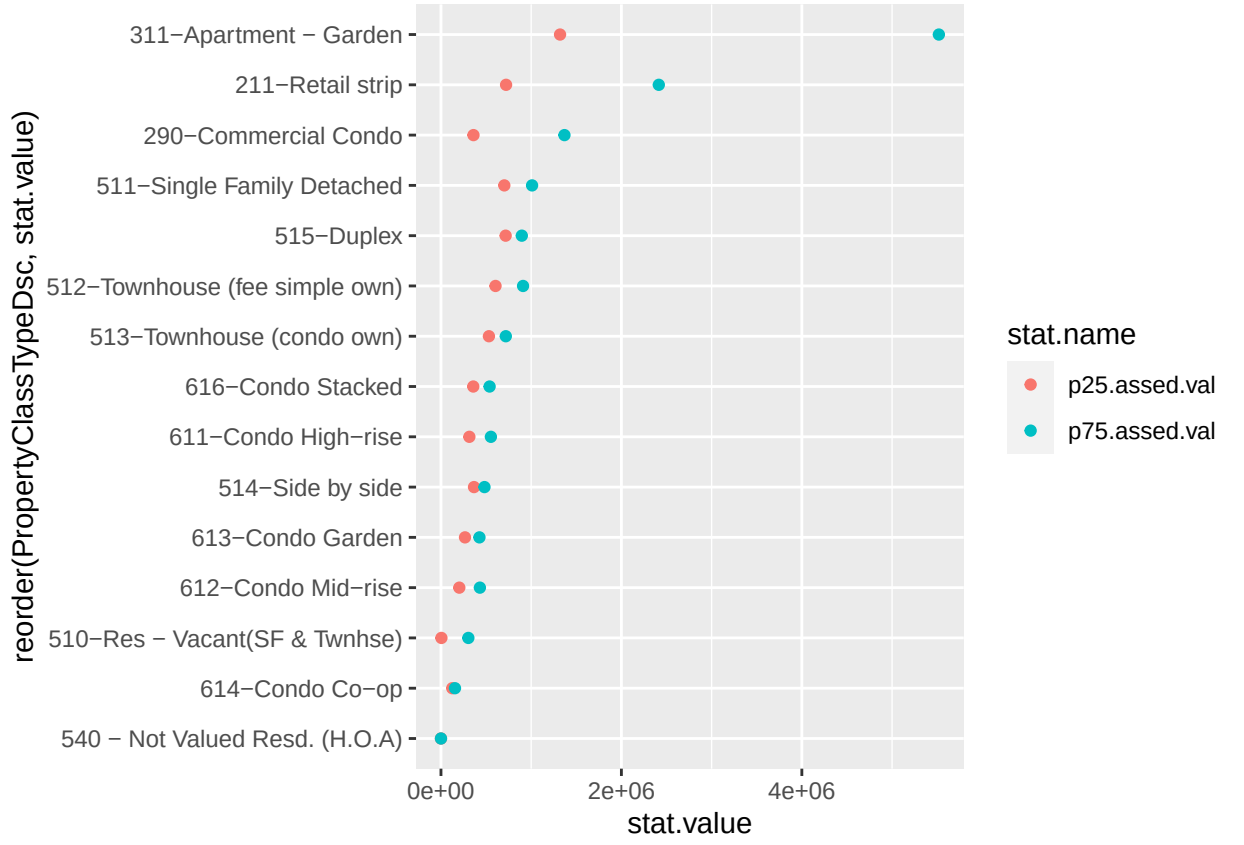
One way we've used to make comparisons across a pair at multiple levels or categories is a paired or grouped bar. Let's start with that for comparison. Recall that we use `geom_col` for a bar graph where we've already calculated the required statistics. We use `position = "dodge"` to make paired bars. The `aes()` option `fill =` is for our statistic type. We limit the sample to property types with more than 200 properties and to the 75th and 25th percentiles (no median, so `bypc.long$pnum != "50"`).

```
# use the comparison to bar charts
badbar <-
  ggplot() +
  geom_col(data = bypc.long[which(bypc.long$pnum != "50" & bypc.long$obs > 200),],
           mapping = aes(x = reorder(PropertyClassTypeDsc, stat.value),
                         y = stat.value, fill = stat.name),
           position = "dodge") +
  coord_flip()
badbar
```



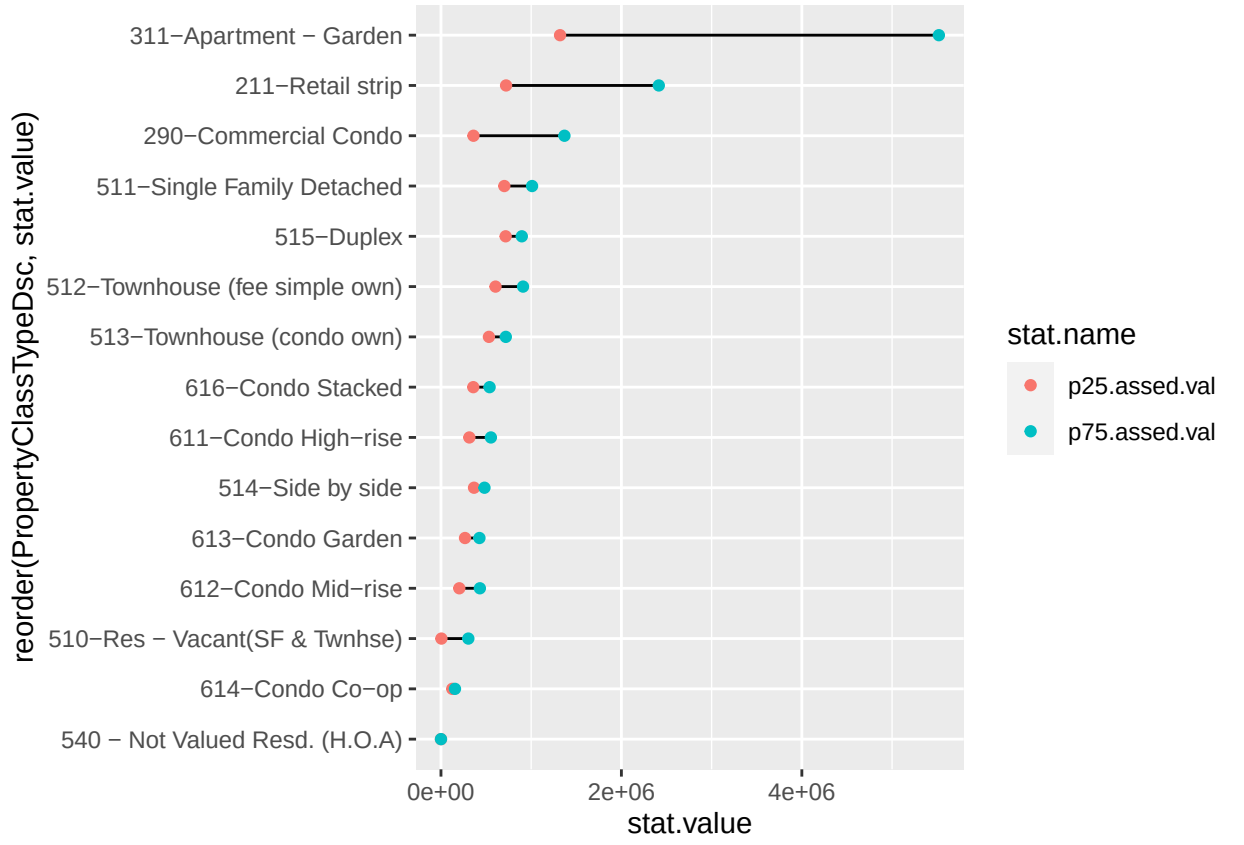
This picture is not a disaster – but it's cluttered and doesn't always make the best visual comparison between the relevant two bar lengths. Instead, let's use a dot plot. We'll start by just plotting both the 25th and 75th percentiles (and omitting the category "215", which is commercial buildings, which has a high 75th percentile and makes the graph look odd).

```
# two colors
twoc <-
  ggplot() +
  geom_point(data = bypc.long[which(bypc.long$pnum != "50"
                                    & bypc.long$obs > 200
                                    & bypc.long$PropertyClassTypeDsc != "215-Gen Comm - other"),],
             aes(x = stat.value, y = reorder(PropertyClassTypeDsc, stat.value), color = stat.name))
twoc
```



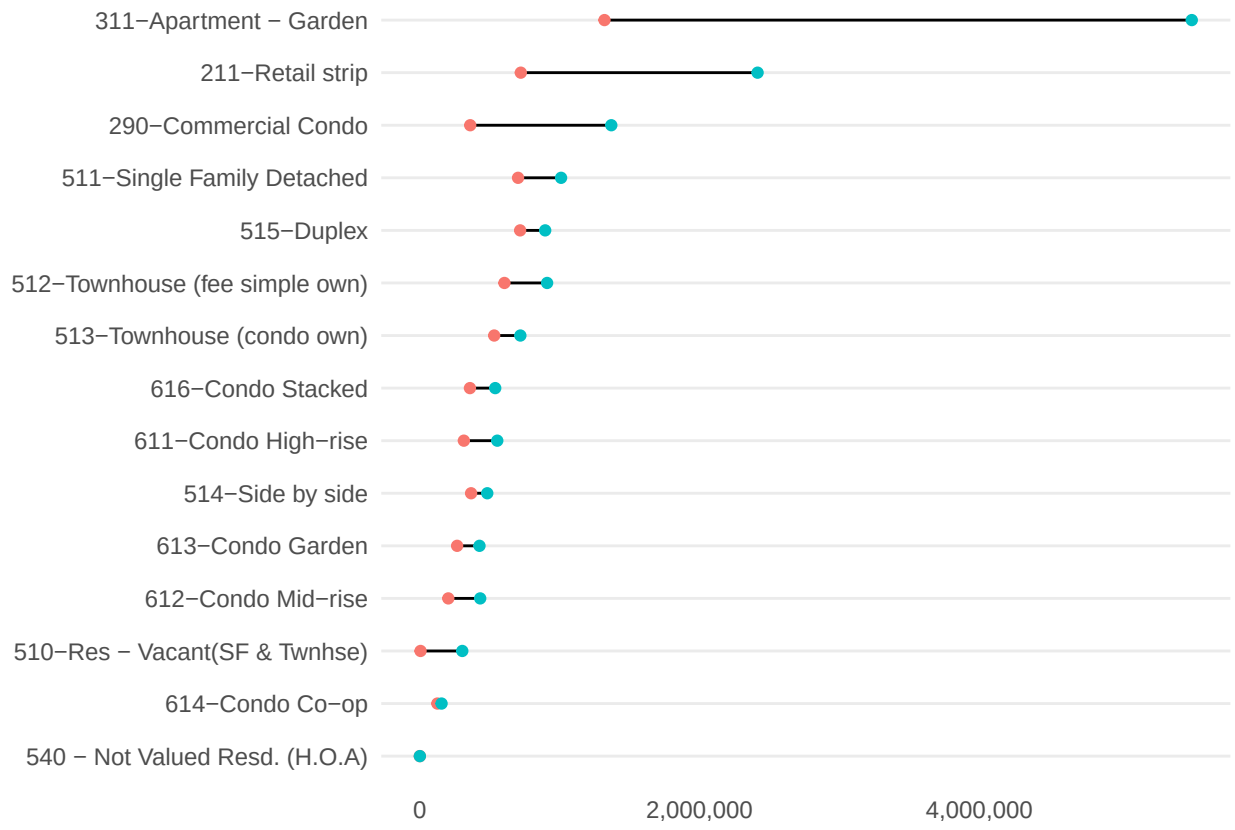
Now let's do some fixes to make this more clear. We'll add a line to emphasize the horizontal comparison we want readers to make, and we put all the data info into the `ggplot()` call since it is the same for both the `geom_point()` and the `geom_line()` commands. The `geom_line()` command uses `aes(group = PropertyClassTypeDsc)` to let R know that the line should be by the `y` variable. We put the `geom_line()` before `geom_point()` so that the points cover the line and it looks neater.

```
# two colors, make it look decent
twoc <-
  ggplot(data = bypc.long[which(bypc.long$pnum != "50"
                                & bypc.long$obs > 200
                                & bypc.long$PropertyClassTypeDsc != "215-Gen Comm - other"),],
         mapping = aes(x = stat.value,
                       y = reorder(PropertyClassTypeDsc, stat.value))) +
  geom_line(aes(group = PropertyClassTypeDsc)) +
  geom_point(aes(color = stat.name))
twoc
```



This graph has all the basics. Let's do a little more clean-up on the overall look by getting commas in the scales and fixing the background with the `theme()` command.

```
## make it presentable
twocd <-
  ggplot(data = bypc.long[which(bypc.long$pnum != "50"
                                & bypc.long$obs > 200
                                & bypc.long$PropertyClassTypeDsc != "215-Gen Comm - other"),],
         aes(x = stat.value, y = reorder(PropertyClassTypeDsc, stat.value))) +
  geom_line(aes(group = PropertyClassTypeDsc)) +
  geom_point(aes(color = stat.name)) +
  scale_x_continuous(label=comma) +
  labs(y = "structure assessed value") +
  theme_minimal() +
  theme(axis.title = element_blank(),
        panel.grid.major.x = element_blank(),
        panel.grid.minor = element_blank(),
        legend.position = "none",
        plot.title = element_text(size = 20, margin = margin(b = 10)))
twocd
```



We could still do some fixes here. One key fix is labeling the 75th and 25th percentiles somewhere on the graph.

Some other concerns: Why these two colors? I think two shades of the same color might be more intuitive. I'd make the joining line less dark and distracting. I'd also look into the suspiciously high 75th percentile value (higher than for single family homes!) for garden apartments.

F. Homework

Q1

Use a density plot to evaluate whether residential or commercial structures have a greater variance in assessed value. To do this, I encourage you to use the `facet_grid()` we learned this class in combination with `geom_density()`. To discriminate between residential and commercial structures, use `arl.p$CommercialInd`.

Q2

Re-do the graph in section D that plots the average value by year built. Rather than each year, plot years 1900 to 2010 (but only every 10 years, so the picture is legible). Add a legend for the 25th and 75th percentiles by color.

In case this seems overwhelming, here is the order of operations I followed:

- group the data
- summarize to the year level
- keep only decade years
  - alternatively, create a decade variable and summarize by that
- keep only years >= 1900
- make the dataframe long
- plot it

Q3

Make a nice-looking scatter from a different dataset.