Session 4. Exploratory data analysis II: Visualization techniques

Antonio Paez
My Name

26 June, 2022

Highlights:

This is my mini-reflection. Paragraphs must be indented.
It can contain multiple paragraphs.

Threshold Concepts:

threshold concept 1
threshold concept 2
threshold concept 3
threshold concept 4

> "The value of experience is not in seeing much, but in seeing wisely."
>
> — William Osler

Session outline

- Why visualization?
- {ggplot2}: a grammar of plots
- Creating empty plots
- Mapping data to aesthetics
- Geometric objects
- Univariate description
- Bivariate description
- Multivariate description

Reminder

Exploratory Data Analysis is like detective work: we are interested in the fundamental characteristics of the data, like their central tendency, spread, and association, and we try to approach the data with as few assumptions as possible.

Preliminaries

Clear the workspace from *all* objects:

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

Load packages. Remember, packages are units of shareable code that augment the functionality of base `R`. For this session, the following package/s is/are used:

```
library(corrr) # Correlations in R
library(dplyr) # A Grammar of Data Manipulation
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

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

```
library(edashop) # A Package for a Workshop on Exploratory Data Analysis
library(ggplot2) # Create Elegant Data Visualisations Using the Grammar of Graphics
library(ggridges) # Ridgeline Plots in 'ggplot2'
library(ggthemes) # Extra Themes, Scales and Geoms for 'ggplot2'
library(janitor) # Simple Tools for Examining and Cleaning Dirty Data
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

```
library(kableExtra) # Construct Complex Table with 'kable' and Pipe Syntax
```

```
##
## Attaching package: 'kableExtra'

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

```
library(skimr) # Compact and Flexible Summaries of Data
```

```
##
## Attaching package: 'skimr'

## The following object is masked from 'package:corrr':
##
##     focus
```

```
library(tidyr) # Tidy Messy Data
```

We will also load the following data frames for this session:

```
data("auctions_amf")
data("auctions_pf")
data("auctions_phy")
data("auctions_sef")
```

These are the same data frames that we used in Session 3, which for convenience we will join into a single table:

```
auctions <- auctions_amf |>
  left_join(auctions_pf,
            by = "id") |>
  left_join(auctions_phy,
            by = "id") |>
  left_join(auctions_sef,
            by = "id")
```

Why visualization?

In Session 3 of the workshop we discussed the use of summary statistics for exploring data. Summary statistics are data reduction techniques that focus on one, or possibly a small number of characteristics of the data. They are usually a single number that aims to describe the data from the perspective of the characteristic(s) of interest: for instance, their central tendency or their spread.

Summary statistics are very important, but as with any data reduction techniques, they are *insufficient* and do not convey other aspects of the data. This is by design. Remember that the aim of EDA is to help us understand the data with our available cognitive capabilities, while avoiding overload. This is why summaries are so useful: they allow us to see, not a lot but wisely.

A complementary approach to summary statistics is the use of visualization techniques. Statistical plots exploit the wonderful ability of the human brain to process information visually. The cognitive power of brains to process alpha-numerical information is limited by our ability to retain information in short-term memory. How many number items can you remember while reading the following table and trying to distinguish patterns?

```
auctions_amf |>
  select(discount, premium) |>
  slice_head(n = 10)
```

```
## # A tibble: 10 x 2
##    discount premium
##       <dbl>   <dbl>
##  1   -0.411  0.0466
##  2   -0.3    0.0741
##  3   -0.214  0.0477
##  4   -0.522  0.133
##  5   -0.624 -0.198
##  6   -0.811 -0.402
##  7   -0.571 -0.237
##  8   -0.481  0.230
##  9   -0.538  0.0950
## 10   -0.152  0.0255
```

What is the central tendency of `discount` taking into account only the numbers shown? Is the `discount` more or less spread than the `premium`? These properties of the data are not readily evident from a quick visual scan of the numbers. Summary statistics retrieve the desired information for us by "flattening" the data:

```
auctions_amf |>
  select(discount, premium) |>
  slice_head(n = 10) |>
  summary()
```

```
##     discount          premium
##  Min.   :-0.8109   Min.   :-0.40225
##  1st Qu.:-0.5626   1st Qu.:-0.14243
##  Median :-0.5015   Median : 0.04713
##  Mean   :-0.4625   Mean   :-0.01853
##  3rd Qu.:-0.3278   3rd Qu.: 0.08979
##  Max.   :-0.1524   Max.   : 0.23039
```

To complicate matters, this is only a small part of the full table (only ten rows and six columns). The task of identifying patterns becomes increasingly complex as the number of observations and the number of variables grow.

Visualization techniques *encode* the data in a way that makes fuller use of our visual data processing powers. Last session we introduced a simple visualization technique, called stem-and-leaf tables. The following stem-and-leaf tables present the *exact same information* as that seen above, but reorganized in a way that engages our ability to process information visually:

stem	leaf
-0.1	5
-0.2	1
-0.3	0
-0.4	18
-0.5	237
-0.6	2
-0.7	
-0.8	1

stem	leaf
-0.4	0
-0.3	
-0.2	3
-0.1	9
0.0	24479
0.1	3
0.2	3

Stem-and-leaf tables encode one aspect of the data (frequency of values) in the form of *lengths*. We organize the data in such a way that the most common values appear as *long* sequences of numbers, and the least common as *short* sequences of numbers. Length is only one possible way of encoding aspects of data. Suppose that we wished to encode another aspect of the data, say, their *valence*. We could use colors to do this: red for negative values, and blue for non-negative values. This is shown in the following (modified) stem-and-leaf table:

stem	leaf
-0.4	0
-0.3	
-0.2	3
-0.1	9
0.0	24479
0.1	3
0.2	3

The power of visualization techniques is that we can process multiple information channels *in parallel*. Shapes and colors are only two ways to encode statistical information; in addition, we can distinguish shapes, angles, areas, and positions. These encodings allow the brain to make immediate sense of the underlying values, in the blink of an eye (see Franconeri et al. 2021), although with less precision than with summary statistics.

To better appreciate this power, consider the matrix of correlations of Session 3:

```
auctions_amf |>
  select(where(is.numeric)) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs") |>
  shave()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

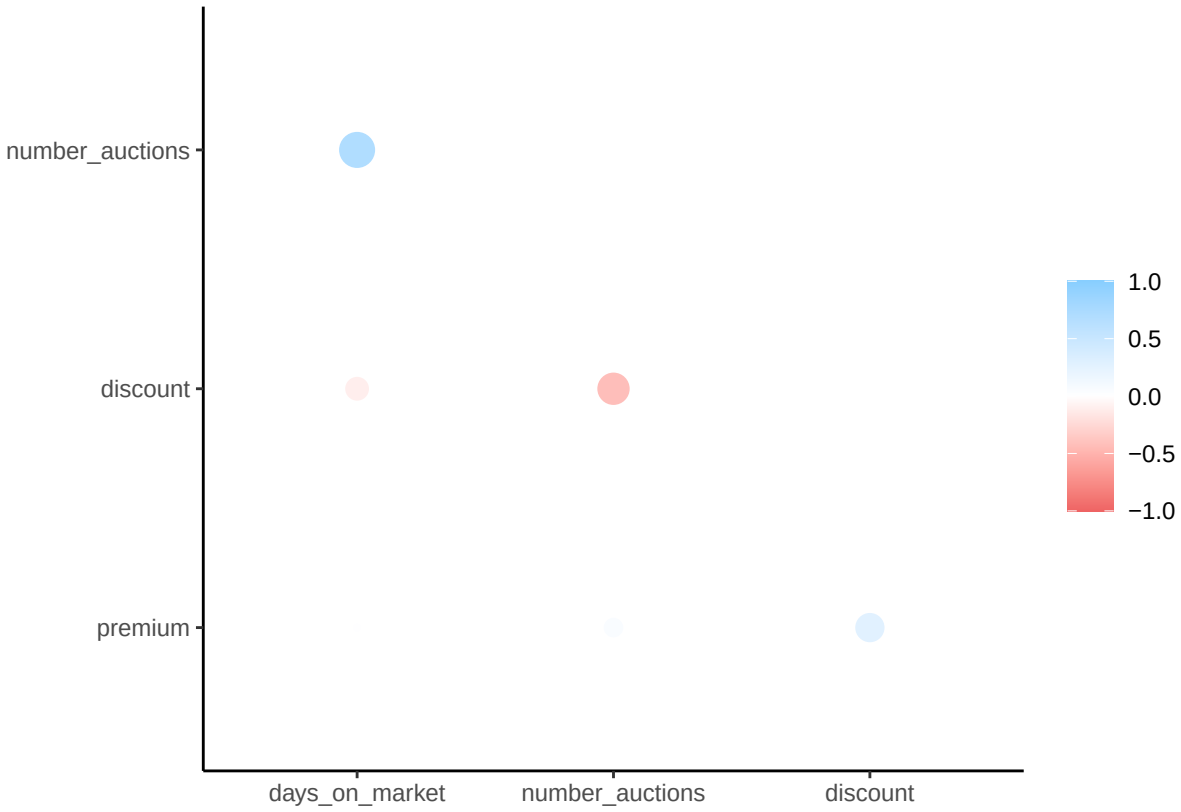
```
## # A tibble: 4 x 5
##   term            days_on_market number_auctions discount premium
##   <chr>                    <dbl>           <dbl>    <dbl>   <dbl>
## 1 days_on_market         NA               NA       NA          NA
## 2 number_auctions         0.693           NA       NA          NA
## 3 discount               -0.282           -0.545   NA          NA
## 4 premium                 0.0829           0.192    0.438      NA
```

Now compare to:

```
auctions_amf |>
  select(where(is.numeric)) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs") |>
  shave() |>
  rplot()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

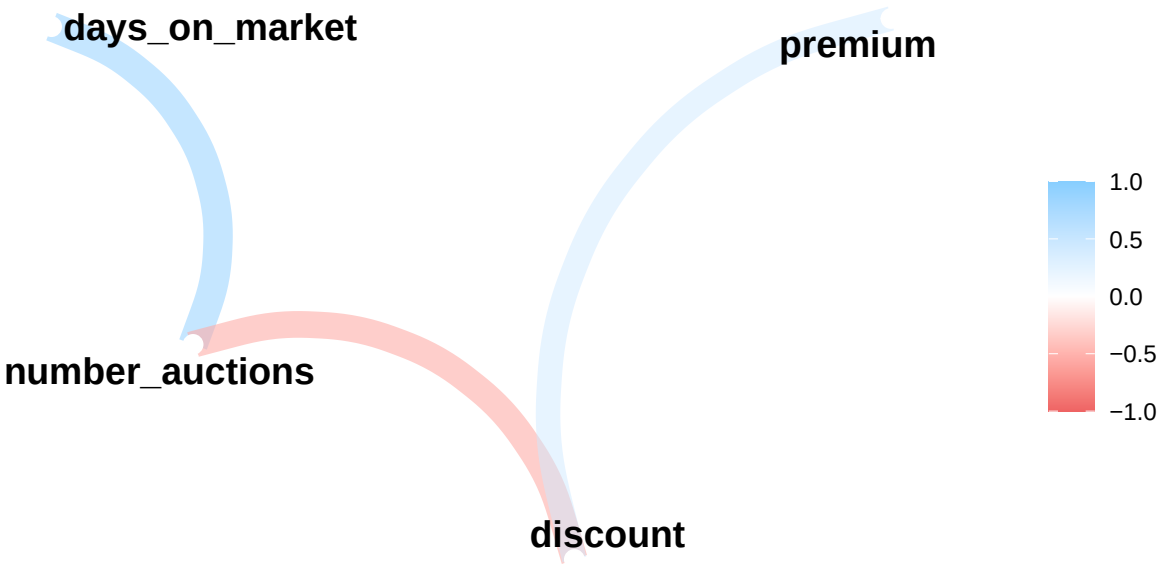
```
## Don't know how to automatically pick scale for object of type noquote. Defaulting to continuous.
```



By encoding valence using colors and magnitude using size, we can present the same information in a form that we naturally process with greater ease. Now compare to:

```
auctions_amf |>
  select(where(is.numeric)) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs") |>
  network_plot()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```



Again, same information, but easier to see how the variables correlate with each other. This plot encodes relationships with colored lines that vary in width and color, and it only requires that we call function `rplot()`. This convenience, on the other hand, comes at a price: what if we wanted to change the *size* of the text? Or the *position* of the legend? Or the *width* of the lines? Or the rate at which the *color* of the lines fades? By hard-coding a specific type of graph, `rplot()` is easy to use but relatively inflexible. A more flexible approach would generalize the common aspects of creating graphs so that they can be applied in a consistent but flexible fashion.

Before proceeding, pause for a moment and think about statistical visualization techniques that you might already be familiar. How do they encode data in visual form?

-
-
-

As a side note, if you are interested in learning more about how the brain processes visual information, Michael Friendly has some fantastic online resources about the psychology of data visualization.

{ggplot2}: A grammar of plots

Just like the grammar of data manipulation discussed in Session 3 of the workshop was a convenient way to think about data operations, creating statistical plots is probably more flexible and natural when presented in the form of a grammar. A grammar of graphics was formalized by statistician and computer scientist Leland Wilkinson in his book A Grammar of Graphics. This book is the direct inspiration for package {ggplot2}, one of the most versatile and intuitive plotting packages around (see Wickham, 2010)

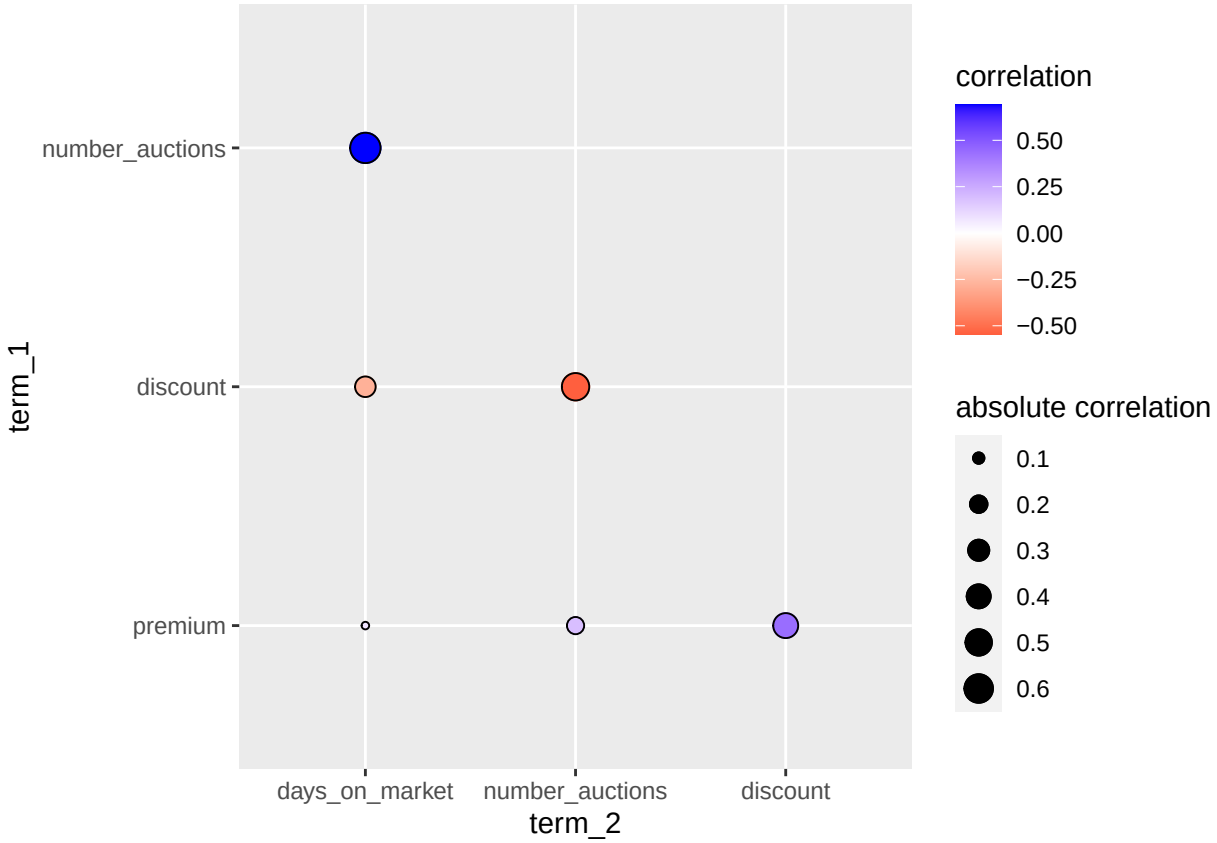
Package {ggplot2} (the name stands for "grammar of graphics in 2-dimensions) implements a *layered* grammar of graphics. Each function acts as an element in a phrase of visualization, and by adding them together we can create fully developed phrases of graphics. Unlike easy to implement functions (like `rplot()`

and `network_plot()` above), a grammar of graphics requires us to explicitly state what we wish to plot and how we wish to plot it. It is more verbose, but in contrast we can create more elloquent plots.

As an illustration of this, the following chunk recreates the correlation matrix as a plot (do not worry too much about the details for the time being):

```
# Data manipulation
auctions_amf |>
  select(where(is.numeric)) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs") |>
  shave() |>
  # Pivot the table so that all the correlations are in a single column
  pivot_longer(-term,
               names_to = "term_2",
               values_to = "r") |>
  # Drop all missing observations
  drop_na() |>
  # Transmute the data frame
  transmute(term_1 = factor(term,
                            levels = c("days_on_market", "premium", "discount", "number_auctions")),
            term_2 = factor(term_2,
                            levels = c("days_on_market", "number_auctions", "discount")),
            r) |>
  # Plotting
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  geom_point(aes(x = term_2,
                 y = term_1,
                 size = abs(r)),
             shape = 1) +
  scale_size(name = "absolute correlation",
             range = c(1, 5)) +
  scale_color_gradient2(name = "correlation",
                        low = "red",
                        mid = "white",
                        high = "blue",
                        midpoint = 0)
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```



The first thing to notice is that we combine data manipulation and plotting in a longer sentence. Something very similar to this code exists inside of function `rplot()`; the advantage of making the sentence explicit is that we can more easily manipulate it. For instance, try changing the low and high colors (in `scale_color_gradient2()`) to "brown" and "green". Or change the range (in `scale_size()` to c(2, 10)). Or layer a title with `ggtitle("My matrix of correlations")` (use the `+` sign to *add* statements to the plot; this is similar to piping).

A grammar of data visualization

A plot is a two-dimensional surface on paper or a screen where we render information. What are the parts of a plot?

-
-
-

As hinted by the example above, package {ggplot2} works by *layering* statements based on the grammar of graphics. The basic structure of a {ggplot2} phrase includes some or all of the following pieces:

- Create a {ggplot2} object: this is similar to creating a blank canvas for a statistical plot (not optional)
- Populate the canvas with data (optional)
- Map data attributes to aesthetics on the plot (optional)
- Add geometric objects to the canvas (add data if you have not done so, and map attributes to aesthetics)
- Adjust the look of the plot

Mapping attributes to *aesthetics* is equivalent to encoding the data in visual form. Which variable do we want to encode as colors? Which variable do we want to encode as shapes? Which geometric objects will be used to represent the data?

We will review each of these elements of the grammar next.

Let us return to the plot that we created before and break down the sentence in parts. To reduce the amount of typing, we will begin by naming the output of our data manipulations:

```
my_r_matrix <- auctions_amf |>
  select(where(is.numeric)) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs") |>
  shave() |>
  # Pivot the table so that all the correlations are in a single column
  pivot_longer(-term,
               names_to = "term_2",
               values_to = "r") |>
  # Drop all missing observations
  drop_na() |>
  # Transmute the data frame
  transmute(term_1 = factor(term,
                            levels = c("days_on_market", "premium", "discount", "number_auctions")),
            term_2 = factor(term_2,
                            levels = c("days_on_market", "number_auctions", "discount")),
            r)
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

A correlation is a bivariate statistic, so we need to know which two variables the statistic corresponds to: these are `term_1` and `term_2` in the data frame. The correlation is stored in column `r`:

```
str(my_r_matrix)
```

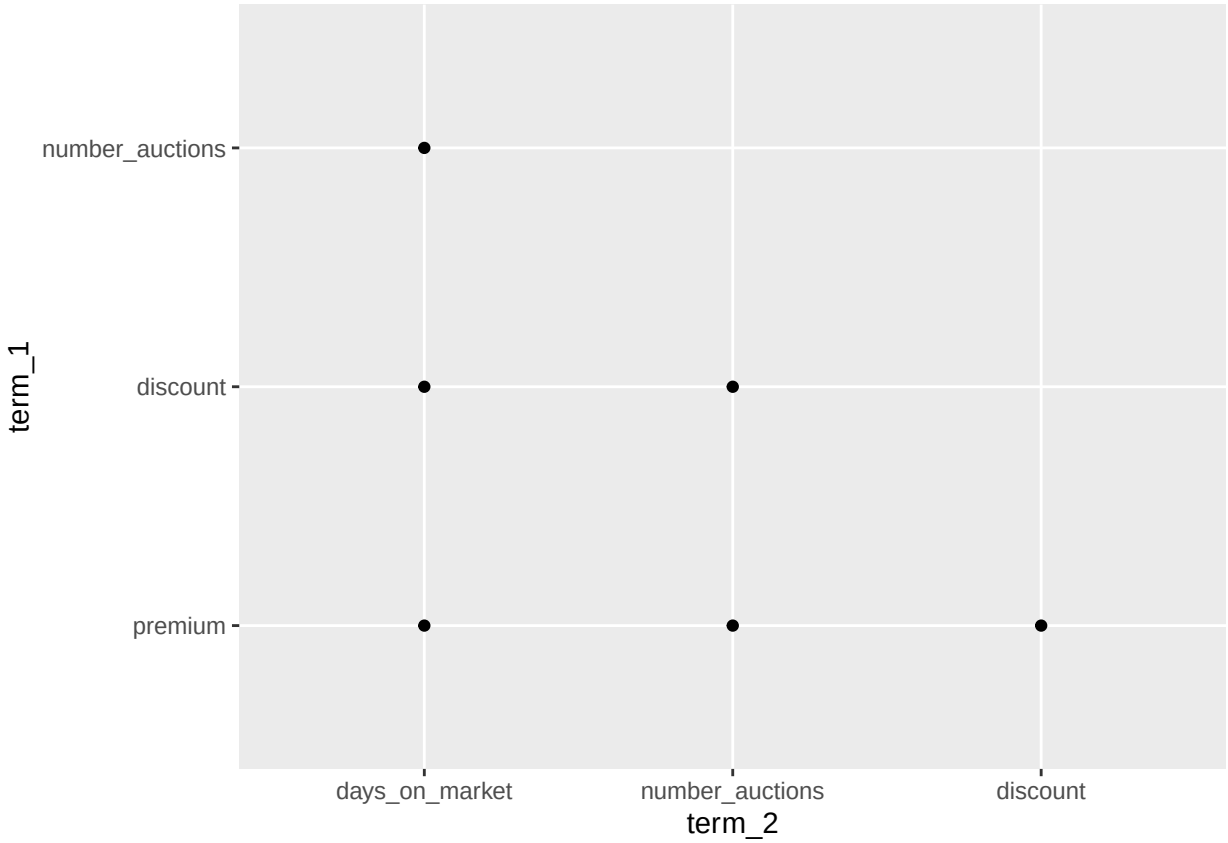
```
## tibble [6 x 3] (S3: tbl_df/tbl/data.frame)
##  $ term_1: Factor w/ 4 levels "days_on_market",..: 4 3 3 2 2 2
##  $ term_2: Factor w/ 3 levels "days_on_market",..: 1 1 2 1 2 3
##  $ r     : num [1:6] 0.6932 -0.2816 -0.5451 0.0829 0.1921 ...
```

The first step to recreate the plot is to populate a blank canvas with our data frame:

```
my_r_matrix |>
  ggplot()
```

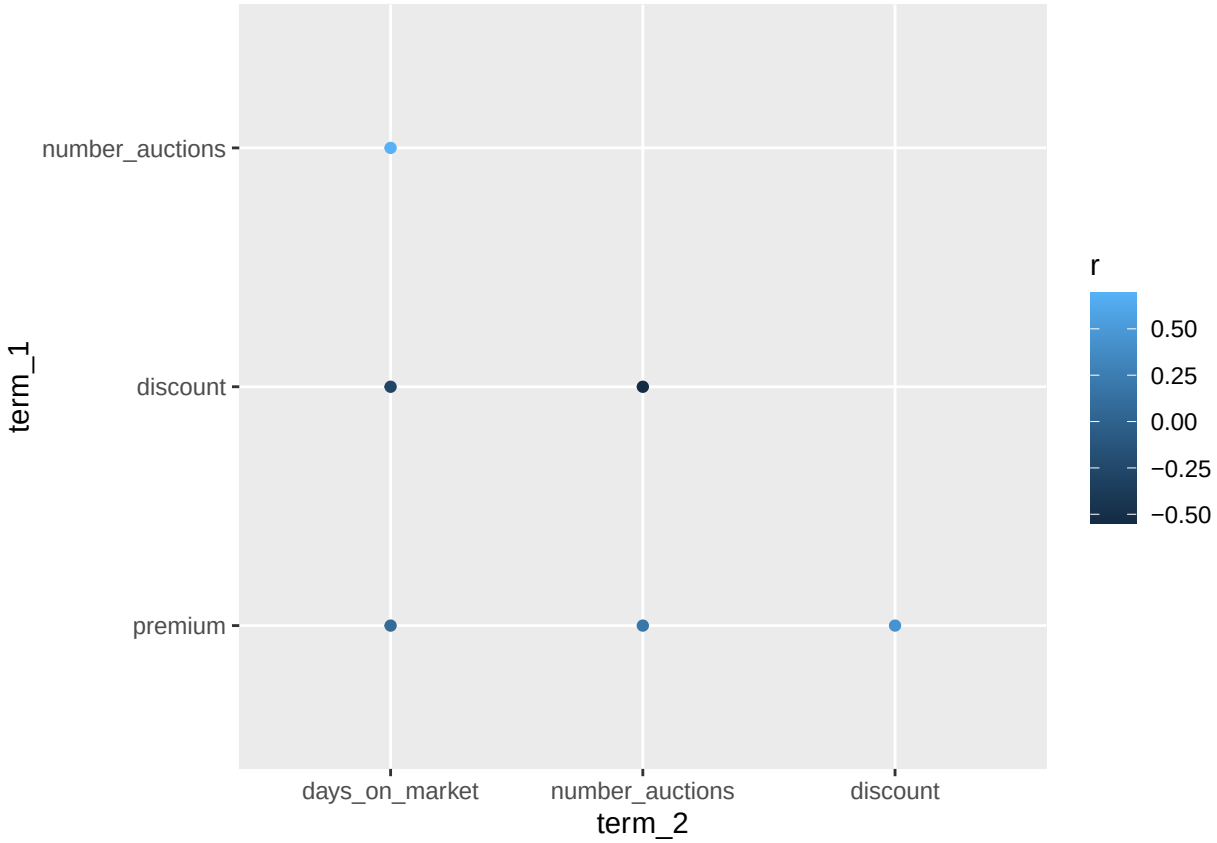
Notice that the plot is still blank. A blank canvas is not yet a statistical plot. Although we already populated the {ggplot2} object with data, we have not yet stated *what* we wish to plot. For this we need to select an appropriate geometric object. Geometric objects are layered by using a function of the family `geom_*`. In this case, we wish to represent correlations as points, so we use `geom_point()`. Now we need to map variables in our table to the plot. Points need to be placed on the plane with "x" and "y" coordinates, so we need to choose which variable maps to the x-axis and which variable maps to the y-axis. To do this, we declare our encoding by means of `aes()`:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1))
```



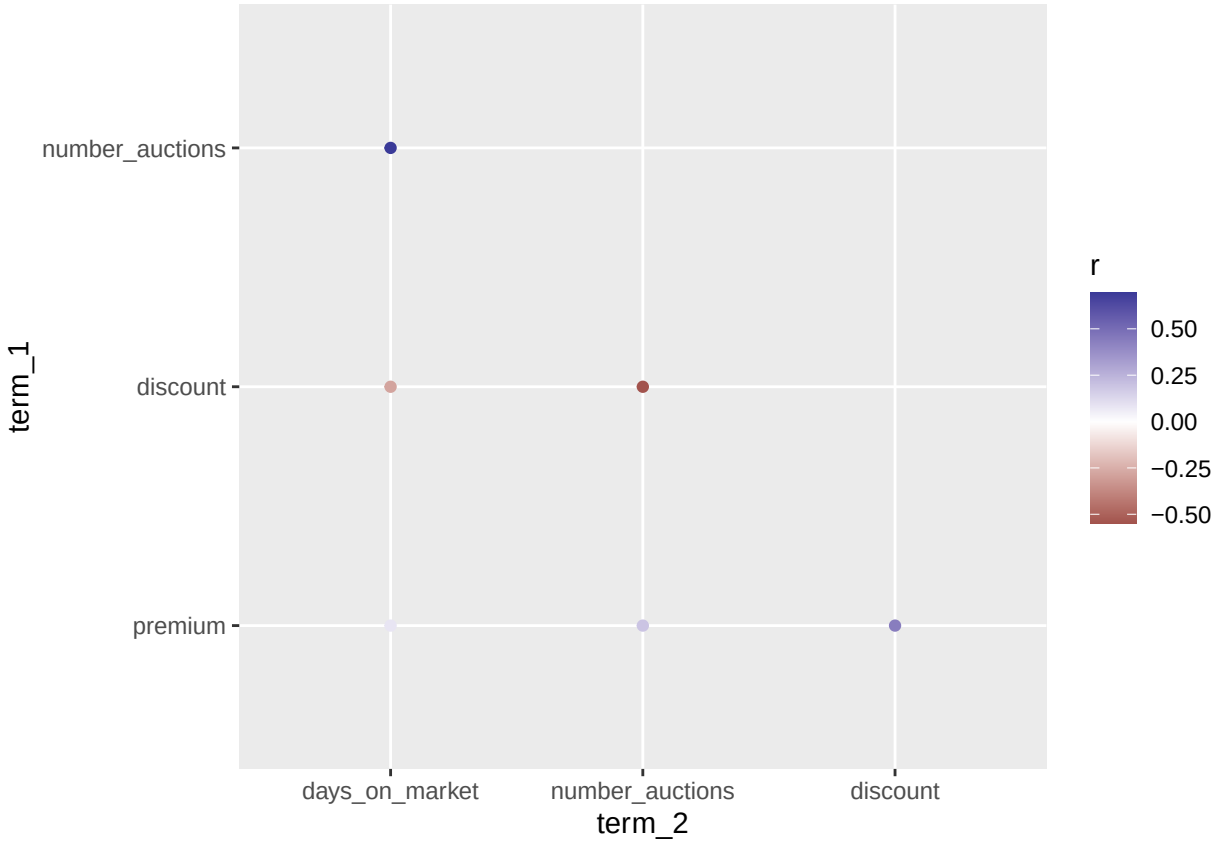
Now we have points in the canvas. But so far we have encoded (i.e., mapped) only *position* (for which we used two variables, `term_1` and `term_2`). Next, we also want to represent the correlation, something that we can do by encoding variable `r` using color:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r))
```



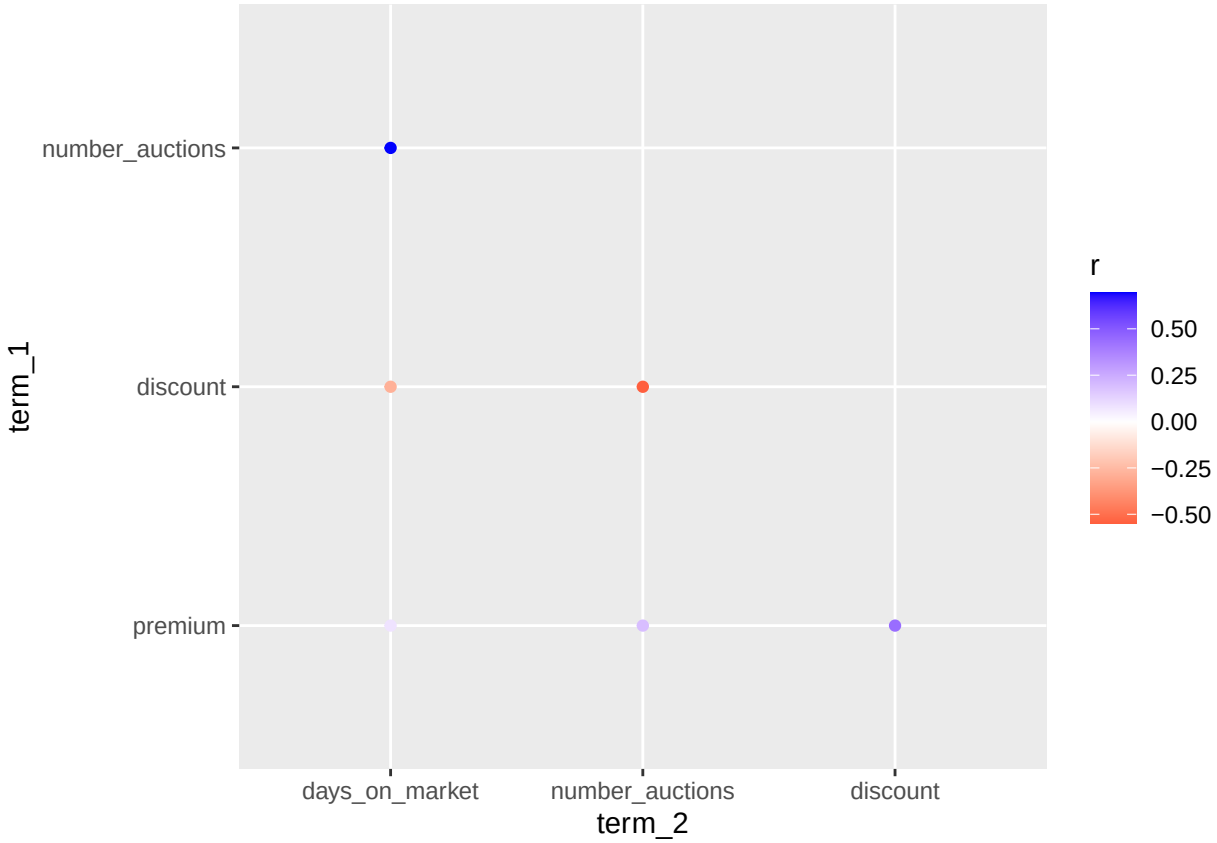
The points now map the correlation to a color. The default colors, though, are not very effective: correlation ranges between -1 and 1, but a sequential color scale does not correctly convey the change in valence in correlation (positive and negative). We can modify the color by using a function of the family `scale_color_*`. Each aesthetic value has a scale and the scale can be manipulated with its corresponding scale function. Function `scale_color_gradient2()` is useful here, because it assumes that colors map to a variable that has *diverging* values (for instance increasingly positive and increasingly negative). Here is our extended phrase:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r)) +
  scale_color_gradient2()
```



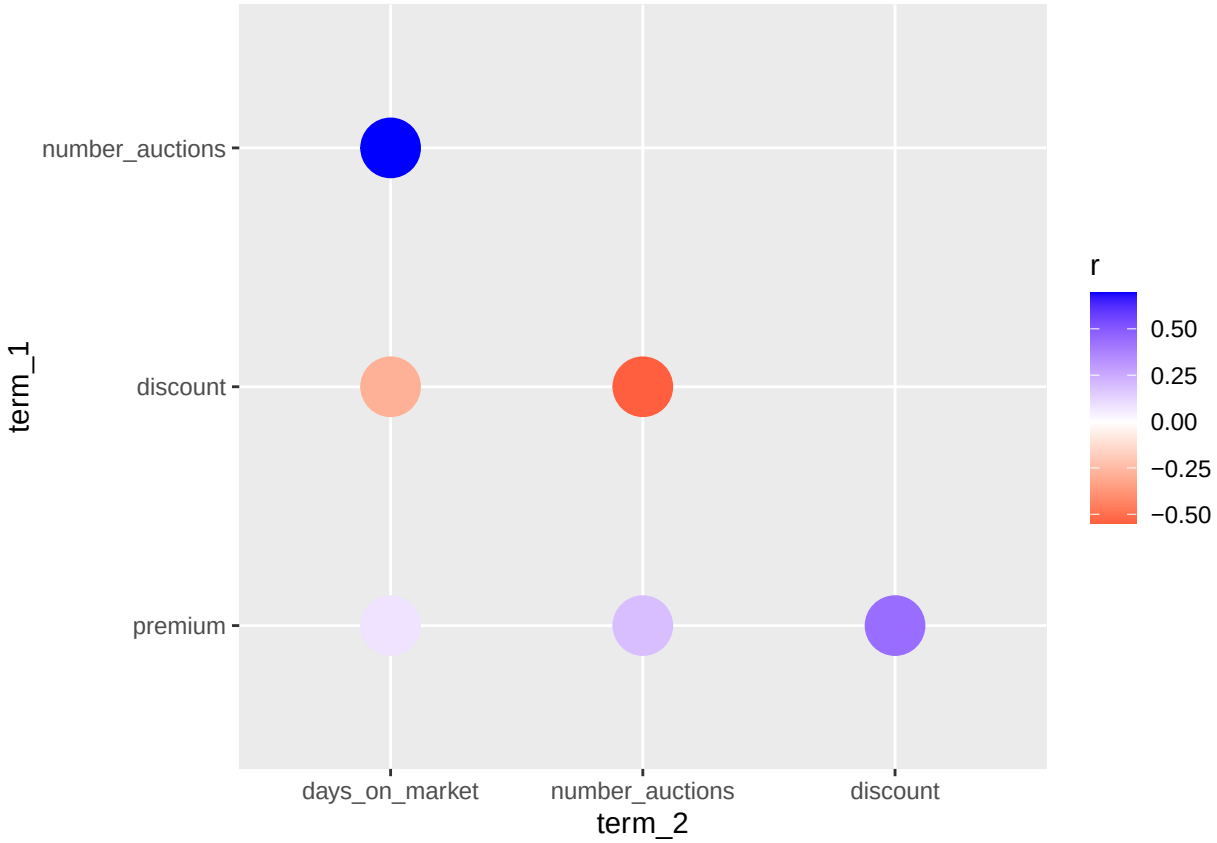
By default, the midpoint of the scale is set to zero, but we can make this explicit, as well as the colors for the high and low values, and even the midpoint:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r)) +
  scale_color_gradient2(high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0)
```



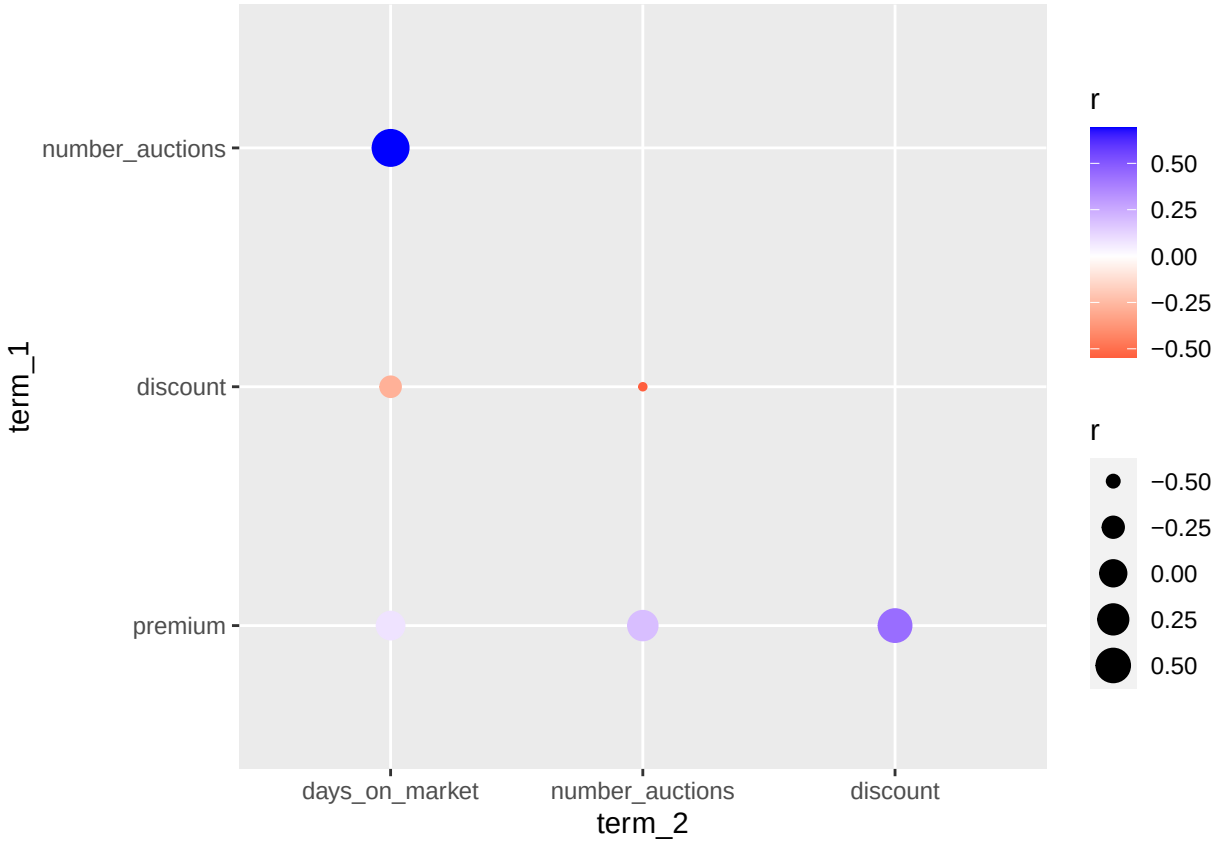
The colors let us more easily perceive the *magnitude* as well as the *direction* of the correlation. That said, the points are too small. We can change the size of the points as follows:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r),
             size = 10) +
  scale_color_gradient2(high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0)
```



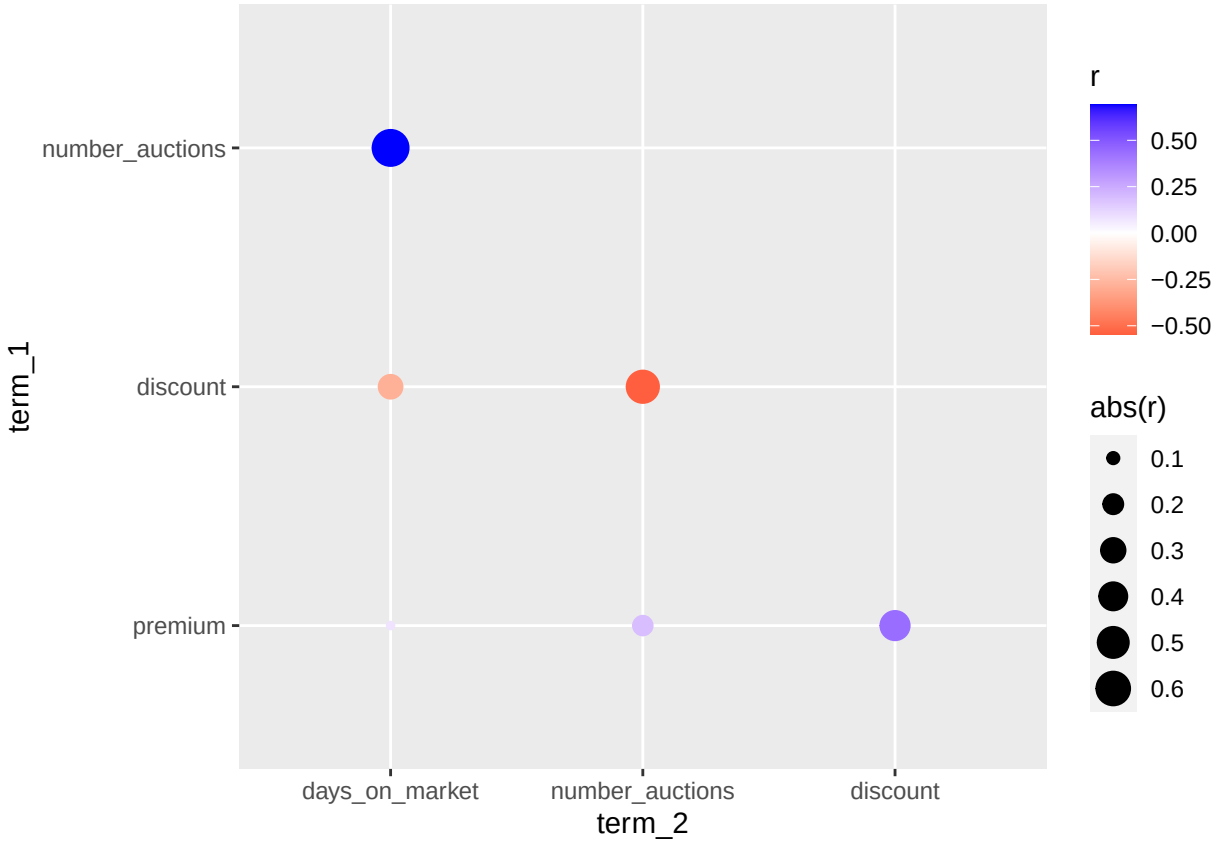
It is easier to see the colors. But notice how the `size` is a constant and does not map to any variable. This is why the points are all the same size. What if we mapped size to the correlation? Now size goes inside `aes()`; this is where we map (encode) aspects of the plot to variables:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = r)) +
  scale_color_gradient2(high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0)
```



We have a small problem here: a correlation of -0.5 is as strong as a correlation of 0.5, but the size of the points is different for each! Similar to our color encoding, we would like the size to understand that -0.5 and 0.5 are identical levels of correlation, even if their valence is different. We can solve this by using a mathematical trick: we are going to map size to the *absolute* value of the correlation:

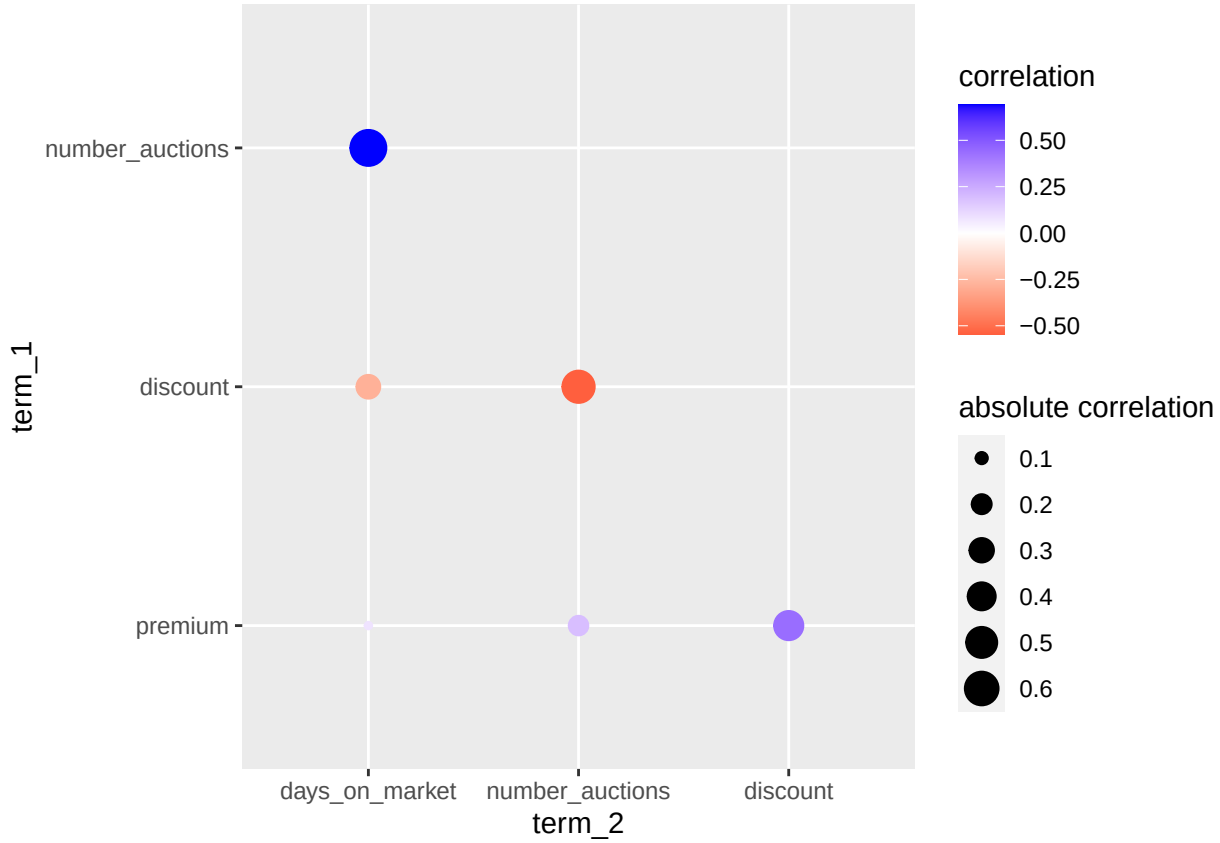
```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  scale_color_gradient2(high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0)
```



Now the sizes reflect the magnitude of the correlation, while the colors reflect both the magnitude and valence.

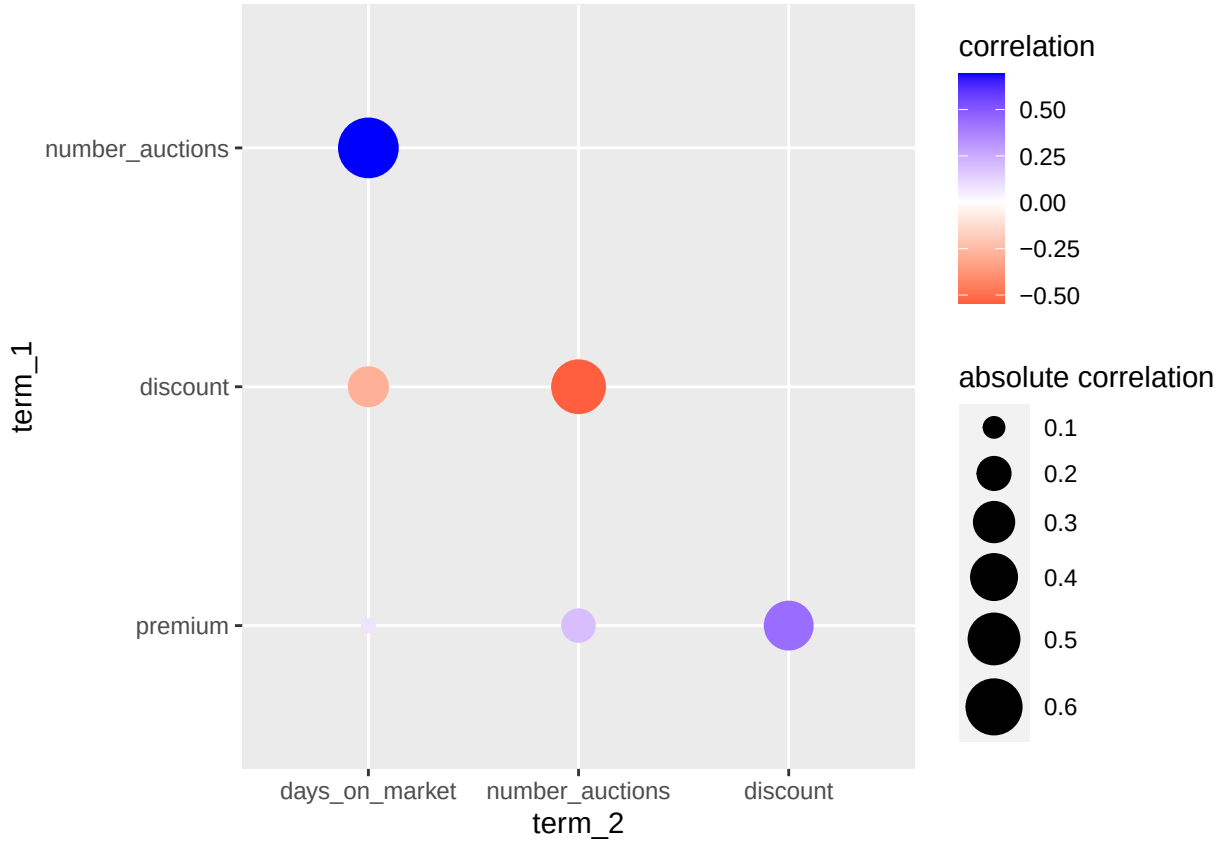
So far we have created a canvas, populated it, chosen an appropriate geometry for the data at hand, mapped variables to aesthetics, and adjusted the scale of one aesthetic value. We can also adorn the phrase that builds our plot. For example, we can use the `scale_*` to change the name of the attribute for the legend:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation")
```



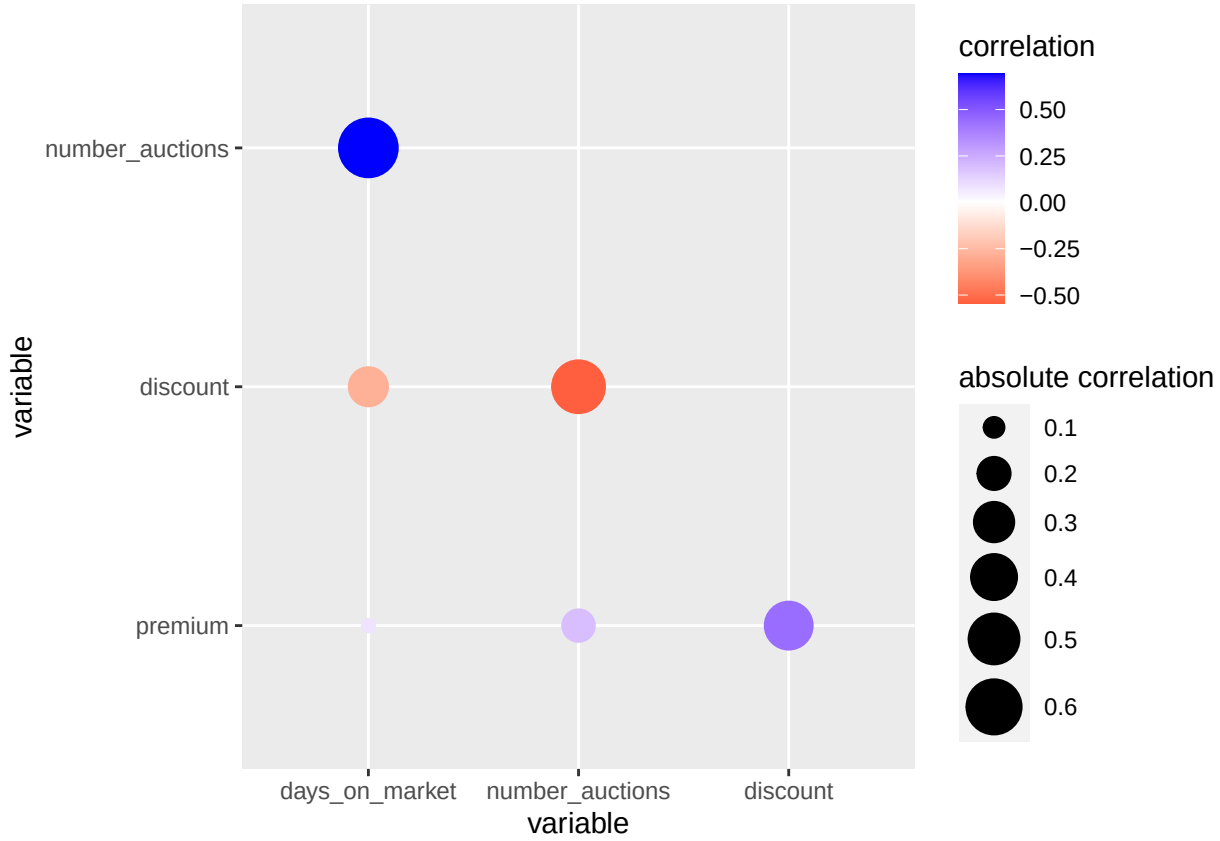
We can modify adjust the range of values for the size of the points:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10))
```



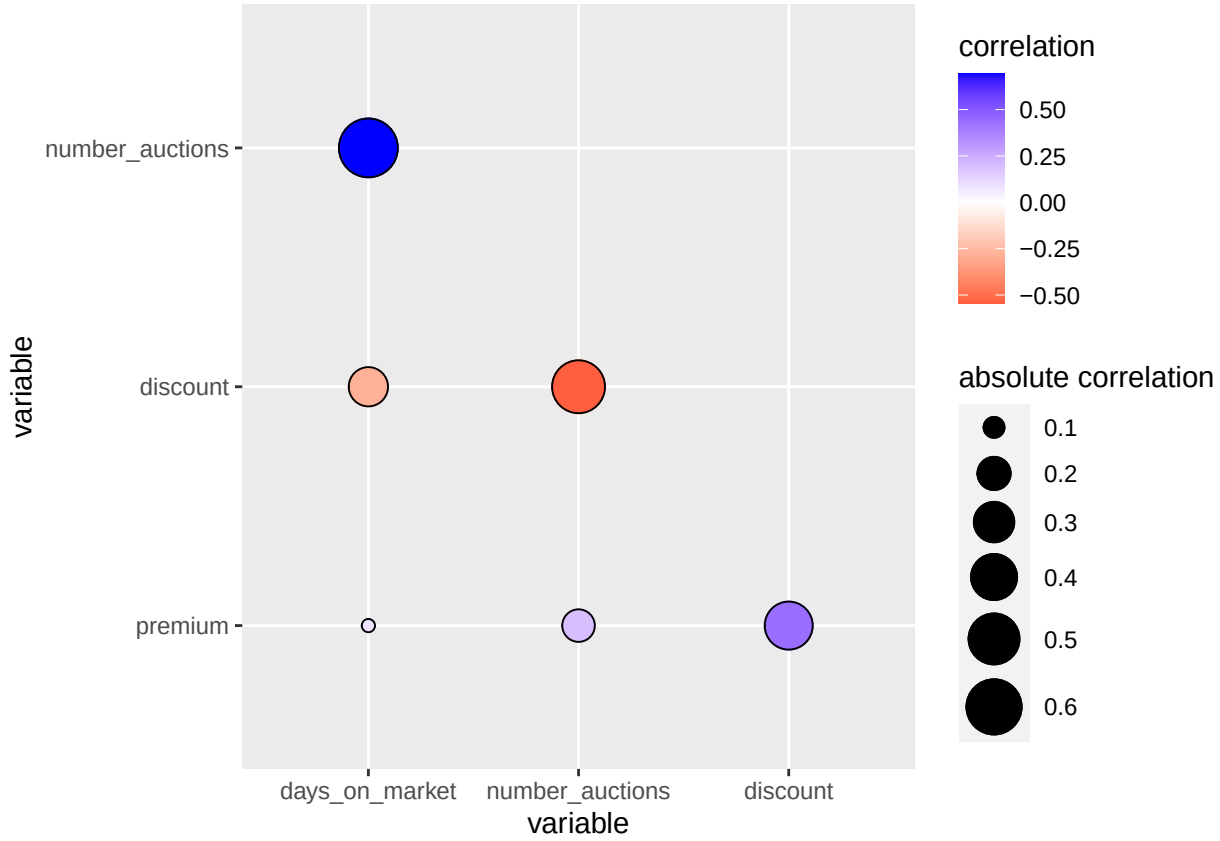
We can also modify the labels for the axes:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10)) +
  xlab("variable") +
  ylab("variable")
```



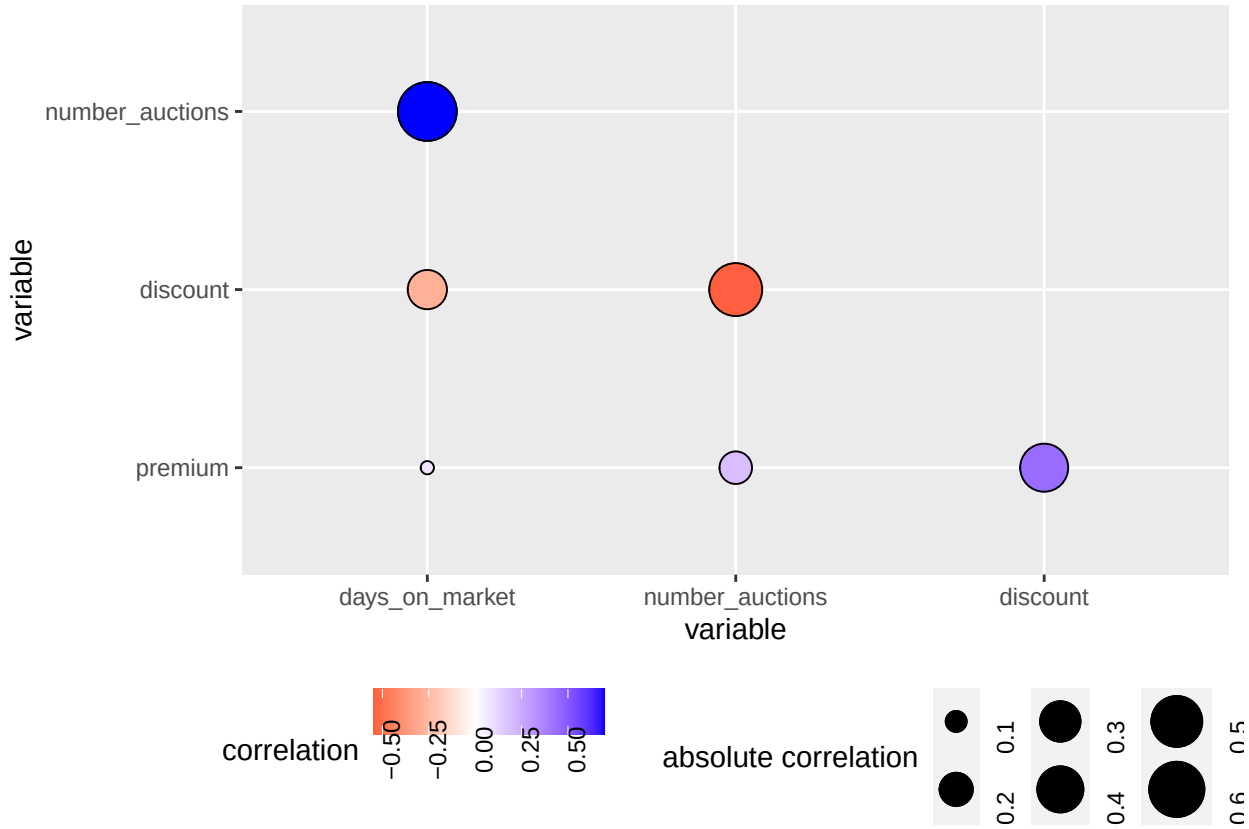
To make the symbols easier to see, we can layer another set of points using a different symbol (a circle without fill) that maps size to absolute correlation but uses only the default color (black) as a constant:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  geom_point(aes(x = term_2,
                 y = term_1,
                 size = abs(r)),
             shape = 1) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10)) +
  xlab("variable") +
  ylab("variable")
```



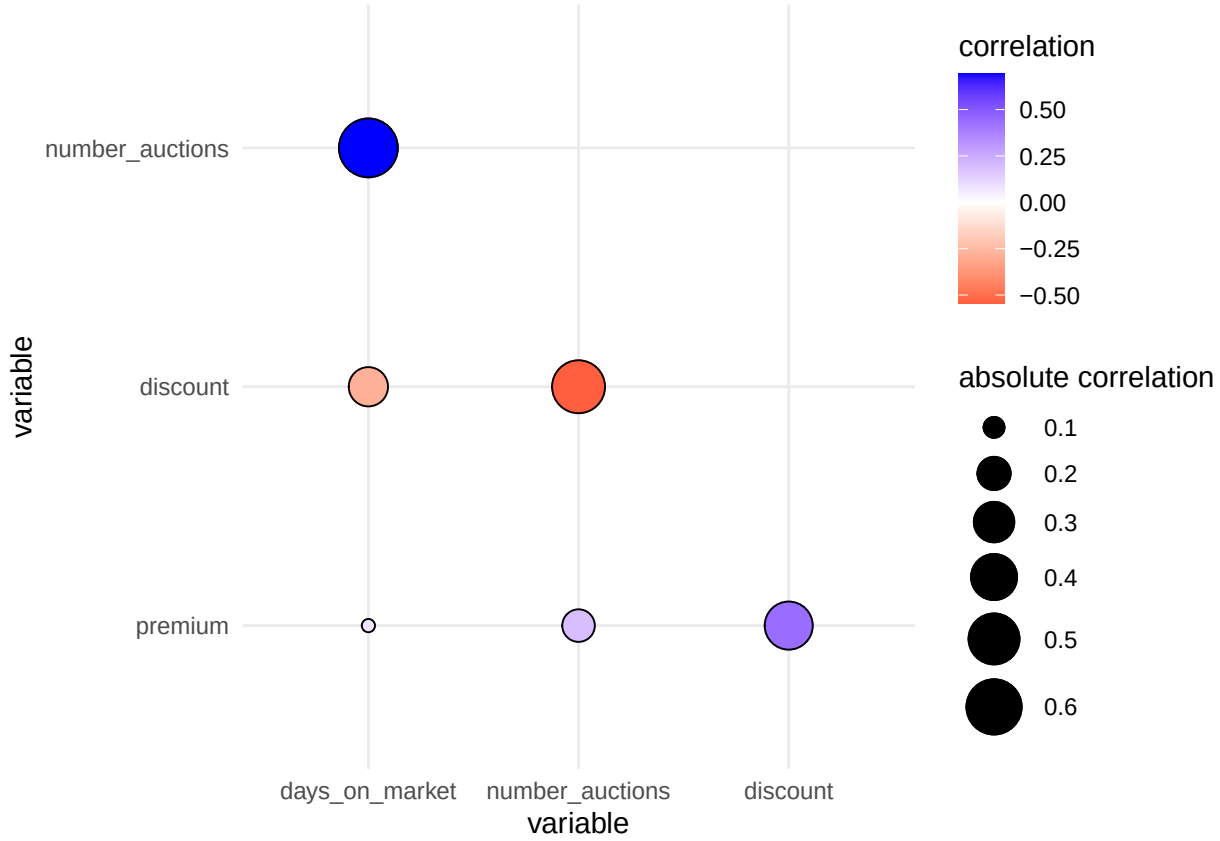
Other aspects of the plot can be modified by means of *themes*. This includes the background color, the color of the grid lines, or whether there are gridlines at all, the orientation of text, and and so on. The parts of the plot are named, and correspond to a certain *element*. For example, the legend has a position and also text. By modifying the text *element* we can adjust the orientation of the text:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  geom_point(aes(x = term_2,
                 y = term_1,
                 size = abs(r)),
             shape = 1) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10)) +
  xlab("variable") +
  ylab("variable") +
  theme(legend.position = "bottom",
        legend.text = element_text(angle = 90))
```



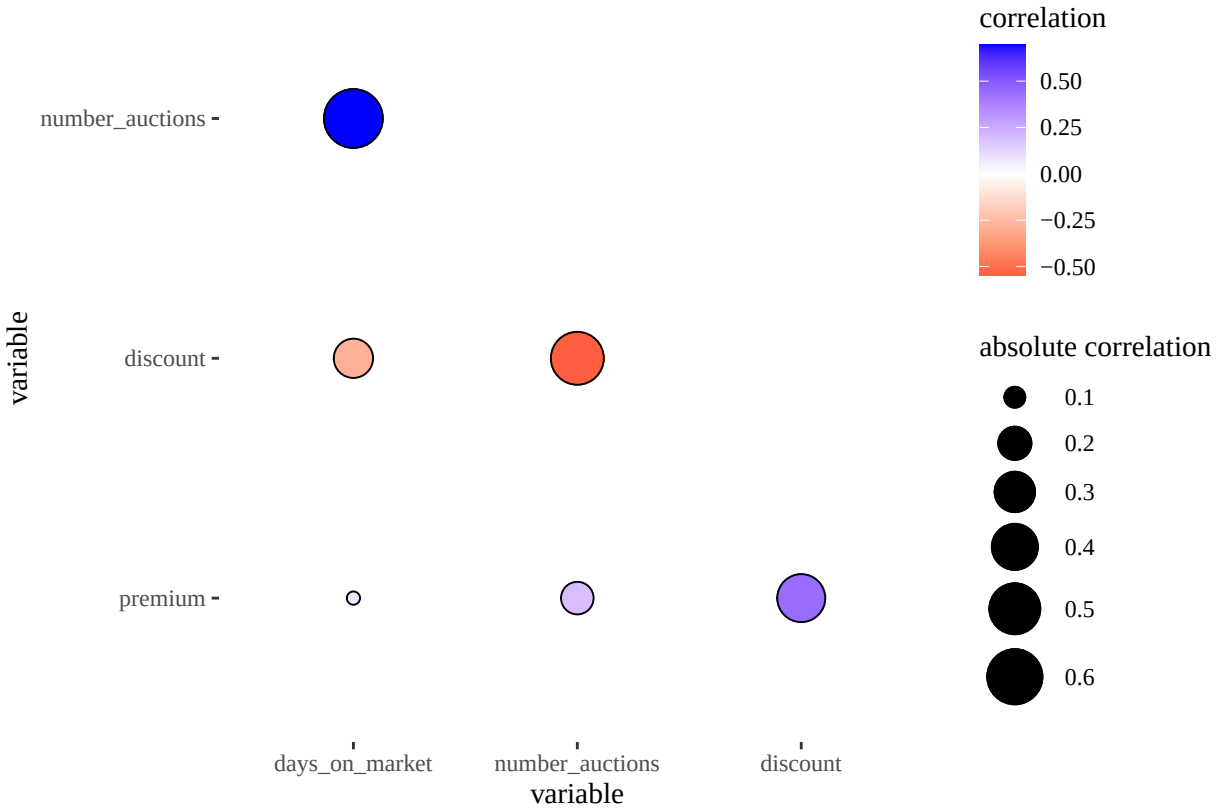
A number of pre-defined themes exist that give different looks to a plot. For example, the following is a minimalist theme that reduces the amount of "ink" used in the creation of the plot:

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  geom_point(aes(x = term_2,
                 y = term_1,
                 size = abs(r)),
             shape = 1) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10)) +
  xlab("variable") +
  ylab("variable") +
  theme_minimal()
```



Other themes include `theme_bw()`, `theme_light()`, `theme_classic()`, and `theme_linedraw()`. Package {ggthemes} includes many additional themes, some of which replicate known themes, for example those used by Tufte, The Economist (`theme_economist()`), Stata (`theme_stata()`), or the Wall Street Journal (`theme_wsj()`):

```
my_r_matrix |>
  ggplot() +
  geom_point(aes(x = term_2,
                 y = term_1,
                 color = r,
                 size = abs(r))) +
  geom_point(aes(x = term_2,
                 y = term_1,
                 size = abs(r)),
             shape = 1) +
  scale_color_gradient2(name = "correlation",
                        high = "blue",
                        mid = "white",
                        low = "red",
                        midpoint = 0) +
  scale_size("absolute correlation",
             range = c(2, 10)) +
  xlab("variable") +
  ylab("variable") +
  theme_tufte()
```



To state the phrase that will create a plot we must meticulously think about what we wish to achieve with it. This involves more work than using a function built for a specific type of graph, since we have to painstakingly control all aspects of the plot. On the other hand, we can be more precise and deliberate about the outcome. And if we need to make changes, for instance if we are experimenting with colors, shapes, etc., a programmatic effort means that we just need to make small adjustments and re-run the code. This approach offers high replicability too.

Appropriate geometric objects by scale of measurement

It will not surprise you, seeing how the scale of measurement was an important consideration when selecting an appropriate summary statistic, that it is also something to think about when choosing geometric objects for statistical plots. Which geometric objects are appropriate will depend on whether the plot is univariate, bivariate, or multivariate and whether the variables that we are trying to encode are categorical or quantitative. We explore this next.

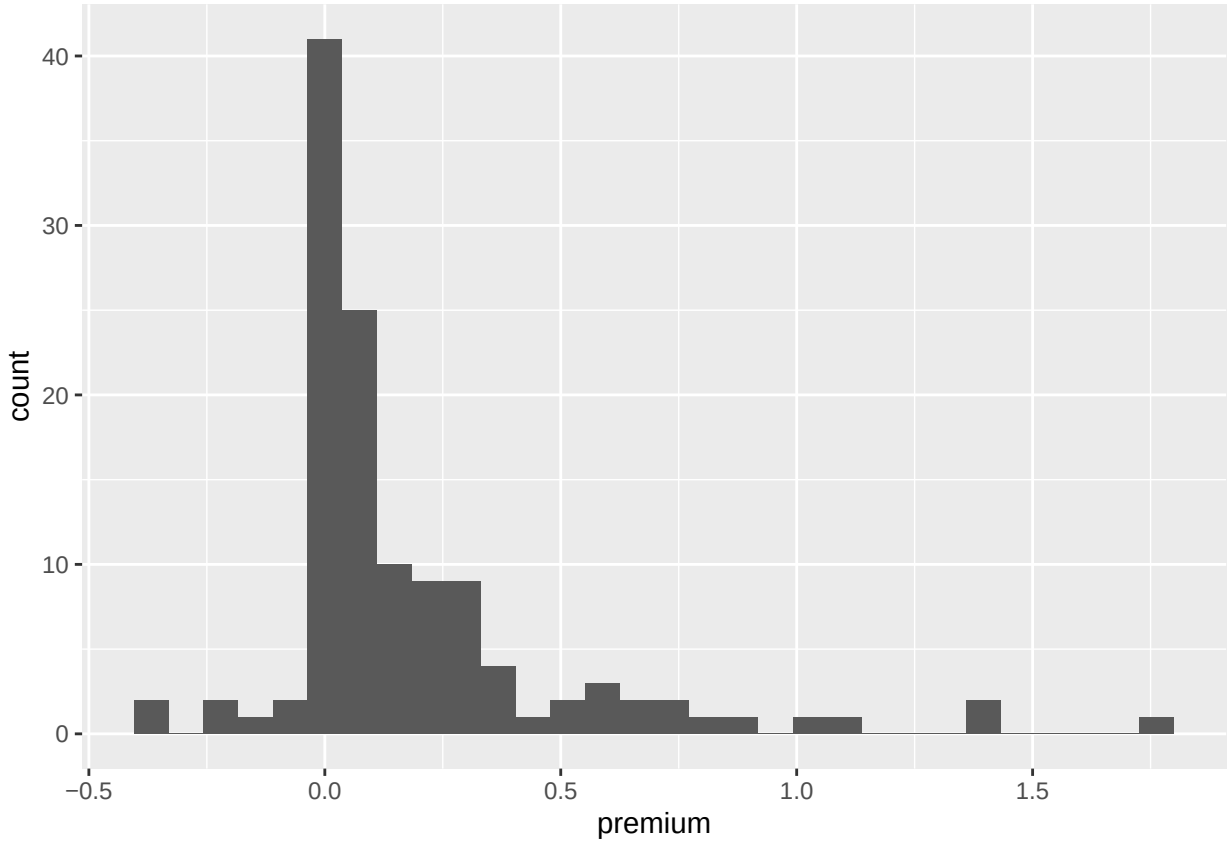
Univariate description

Univariate description involves exploring the main attributes of a single variable, typically its central tendency and spread. For quantitative variables, an appropriate geometric object is a histogram, implemented as `geom_hist()`. For instance:

```
ggplot(data = auctions,
       aes(x = premium)) +
  geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

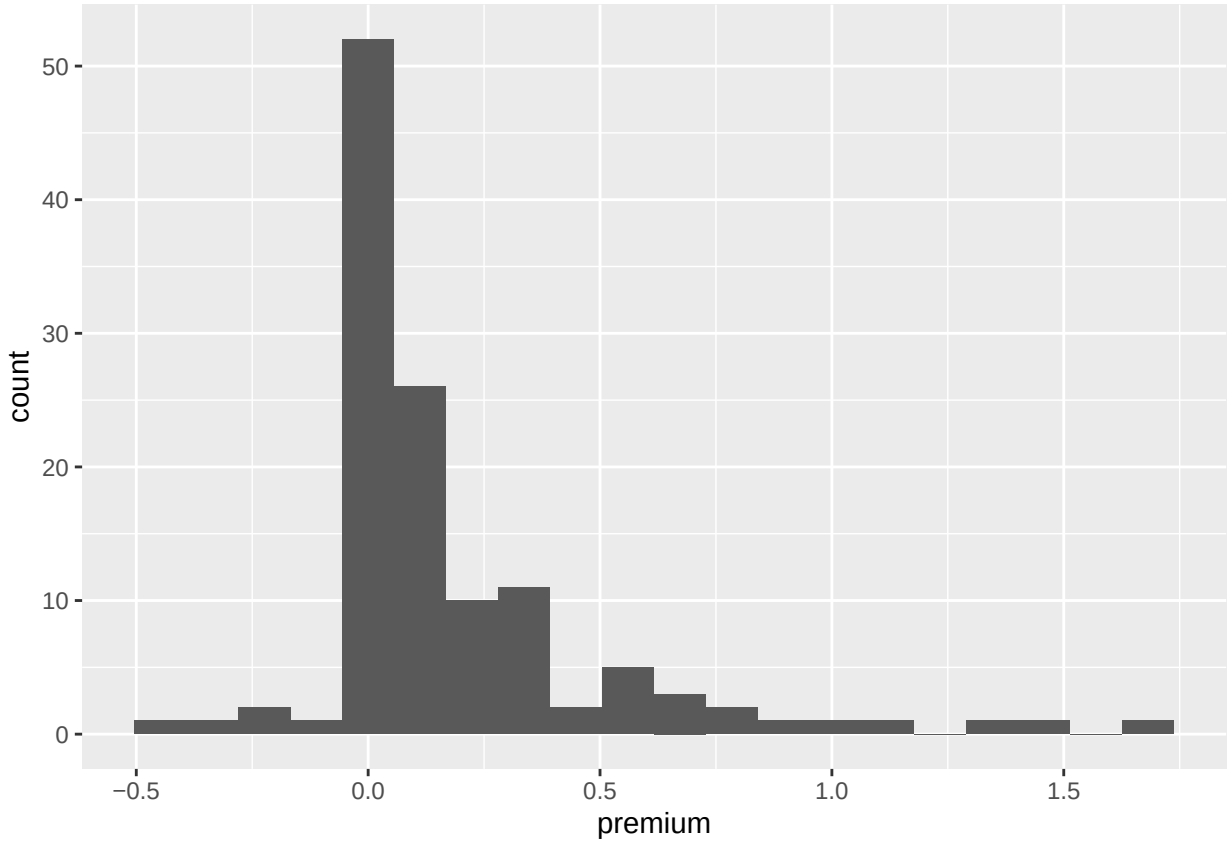
```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```



A histogram is the number of cases (the *count* of cases) by ranges of values. We only need to encode a single variable (in the example above the `premium`), because the "count" on the y-axis is a computed statistic. The default number of bins in `geom_hist()` is 30, but this can be adjusted:

```
ggplot(data = auctions,
       aes(x = premium)) +
  geom_histogram(bins = 20)
```

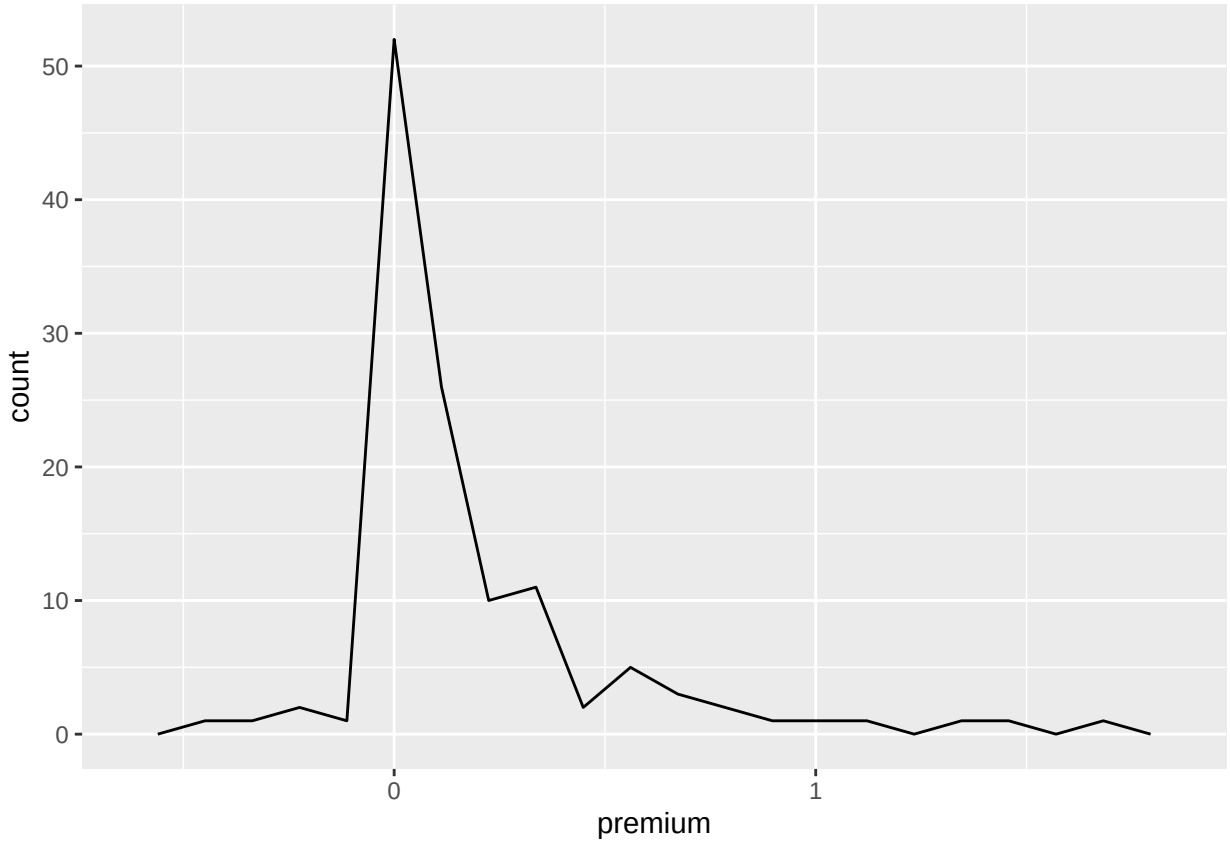
```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```



An alternative geometric object is a frequency polygon, as shown here:

```
ggplot(data = auctions,
       aes(x = premium)) +
  geom_freqpoly(bins = 20)
```

```
## Warning: Removed 3 rows containing non-finite values (stat_bin).
```

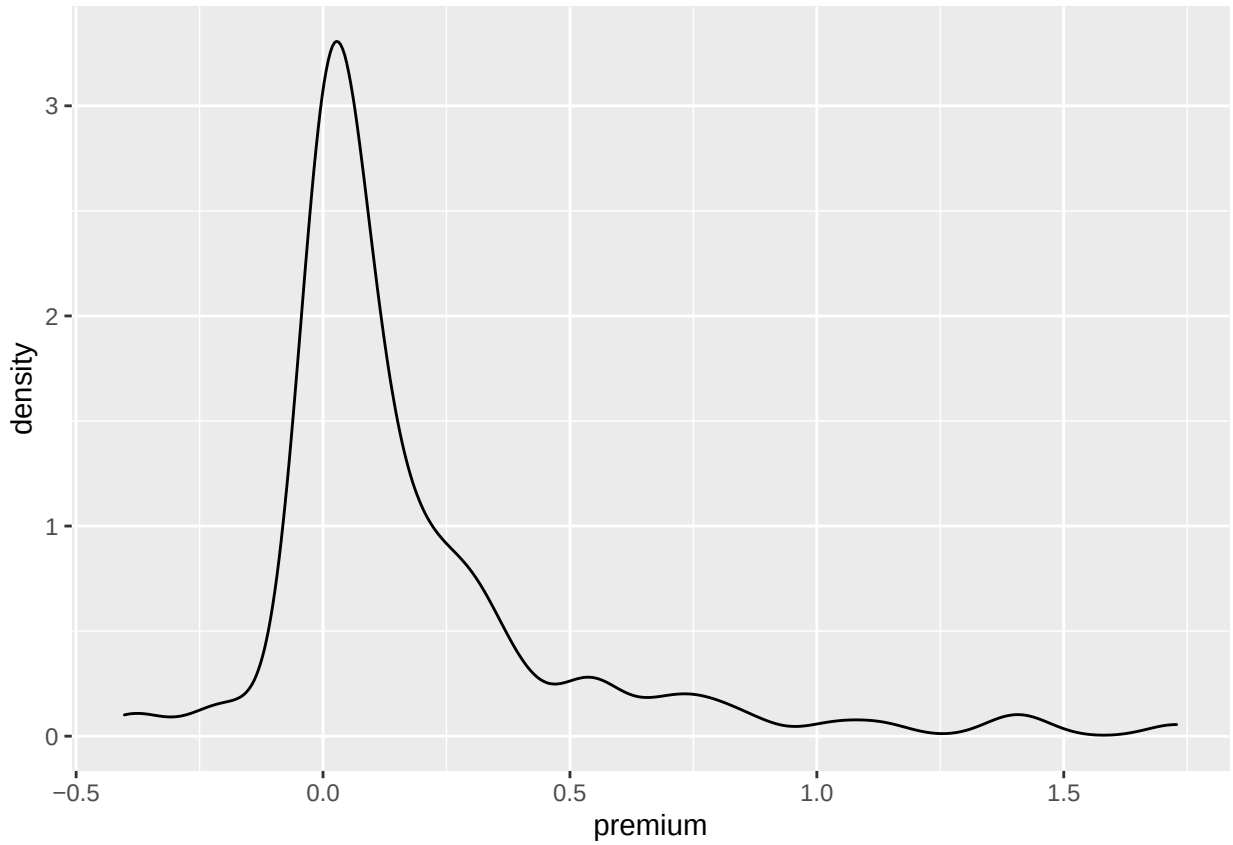


A density plot is a smoother version of a frequency polygon:

```
ggplot(data = auctions,
       aes(x = premium)) +
  geom_density(bins = 20)
```

```
## Warning: Ignoring unknown parameters: bins
```

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



These plots suggest that premiums tend to be positives (sometimes quite large), but in some relatively rare cases they can be negative The mean and median of this variable are:

```
mean_premium <- auctions |>
  pull(premium) |>
  mean(na.rm = TRUE)

median_premium <- auctions |>
  pull(premium) |>
  median(na.rm = TRUE)

mean_premium
```

```
## [1] 0.1814727
```

```
median_premium
```

```
## [1] 0.06676809
```

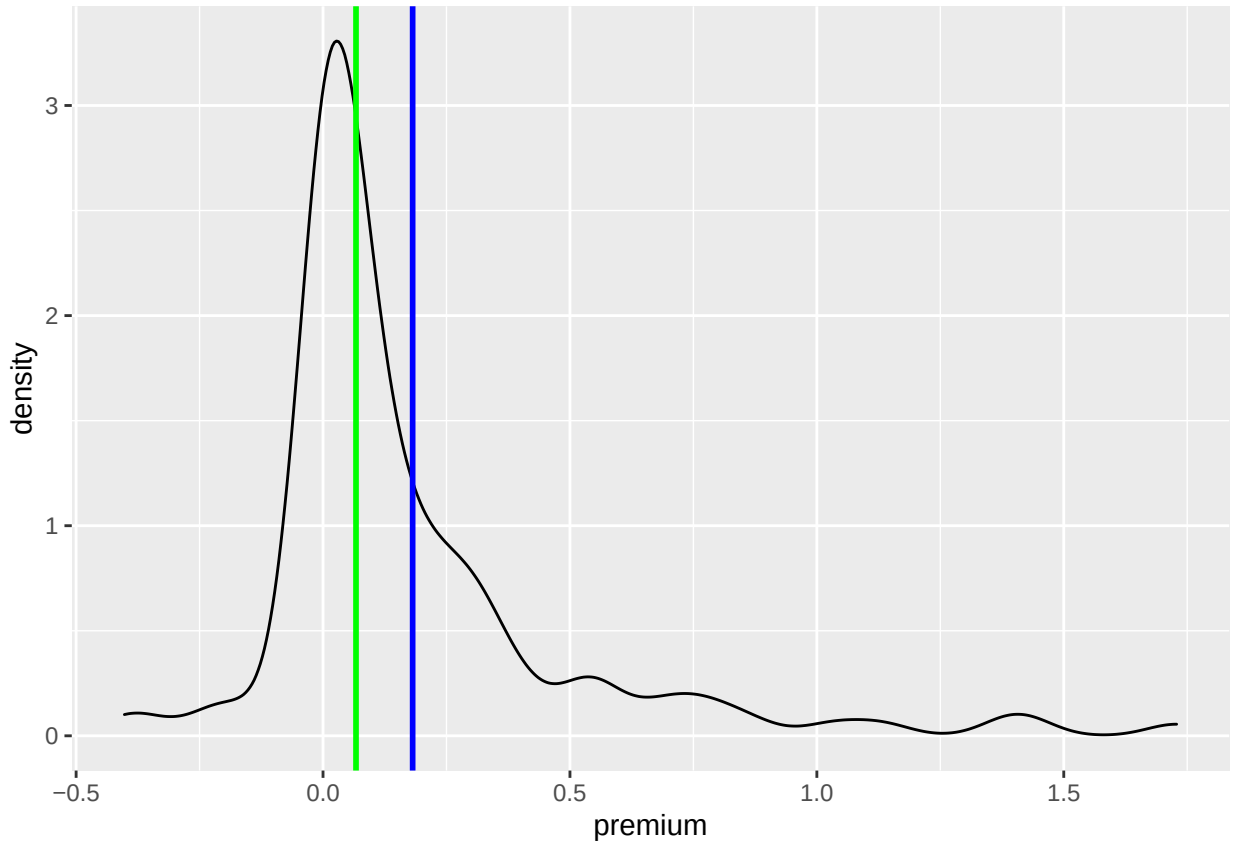
The difference between the mean and the median is due to the lack of symmetry of the distribution. The mean tends to be pulled towards the longer tail of a distribution, as seen below, where we use `geom_vline()` to draw vertical lines (the mean in blue, the median in green):

```
ggplot(data = auctions,
       aes(x = premium)) +
  geom_density(bins = 20) +
```

```
  geom_vline(xintercept = mean_premium,
             color = "blue",
             size = 1) +
  geom_vline(xintercept = median_premium,
             color = "green",
             size = 1)
```

```
## Warning: Ignoring unknown parameters: bins
```

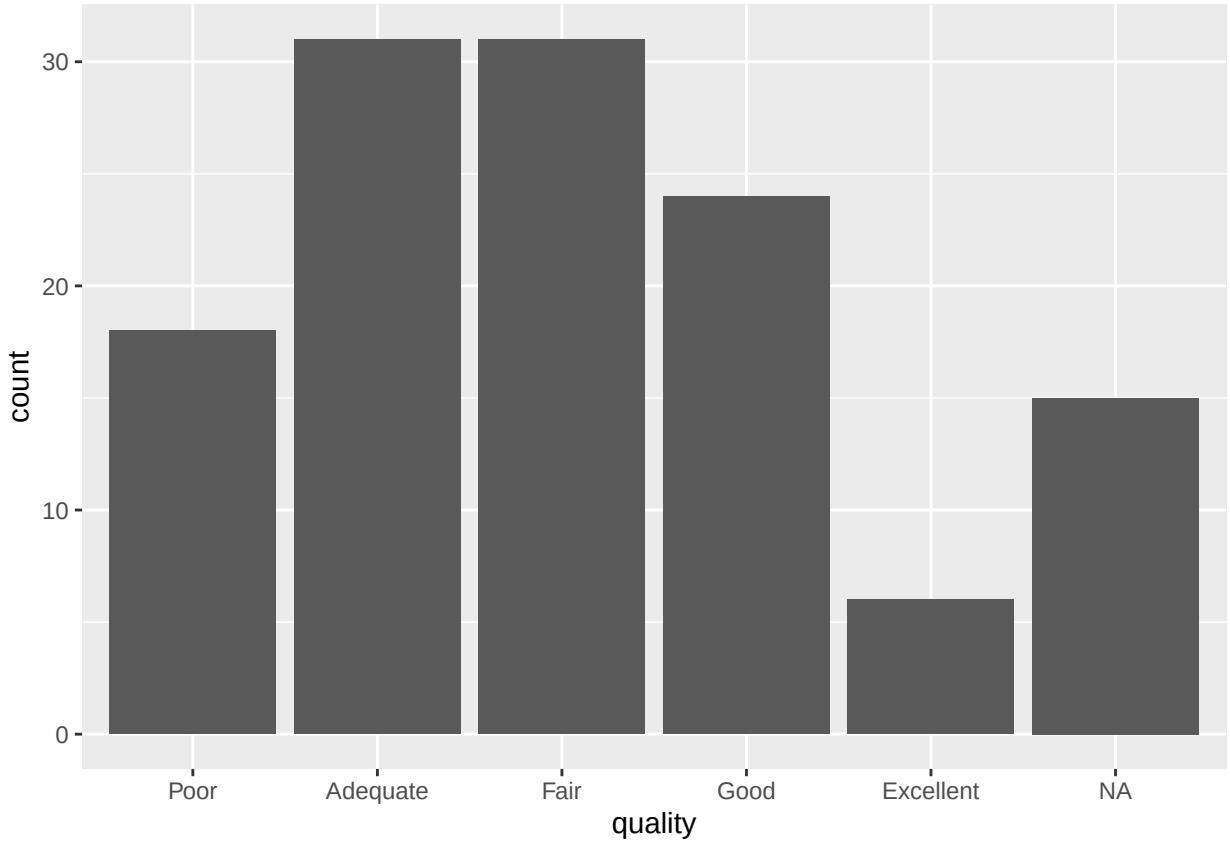
```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



The median is considered a more *robust* measure of central tendency because it is not affected by few unusual values like the mean is.

When the variable of interest is categorical, an appropriate geometric object is a bar chart, implemented as `geom_bar()`. Superficially bar charts look like histograms, but they are different in two important respects: the order of the categories does not necessarily matter, and there are no "ranges" of values, just the labels themselves.This is illustrated next:

```
ggplot() +
  geom_bar(data = auctions,
           aes(x = quality))
```



In the chunk of code above I deliberately populated the data in `geom_bar()` instead of `ggplot()` to illustrate that this is a possibility. Any data declared in `ggplot()` will be used by default in any `geom_*` function in the sentence, unless other data are specified. Data entered in a `geom_*` function will only be used for that geometric object.

Bivariate description

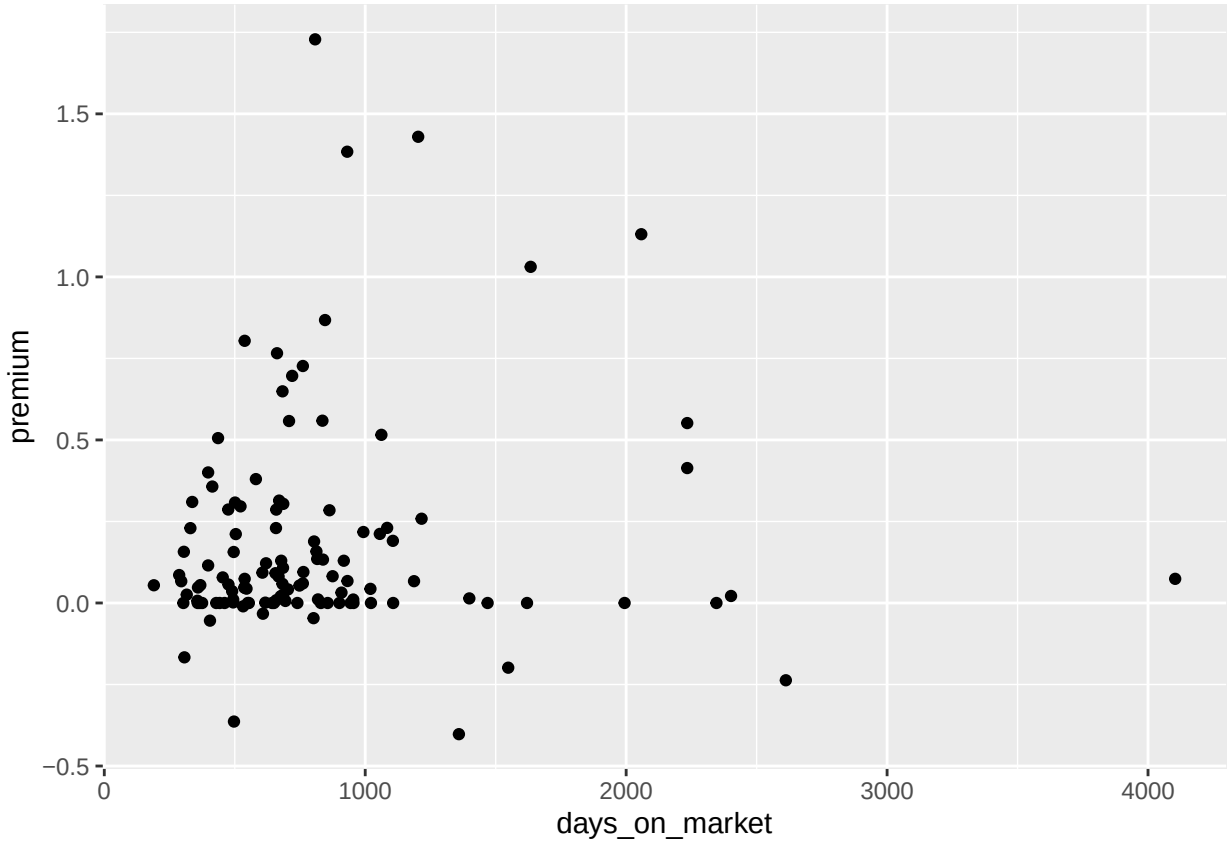
Unlike summary statistics that are defined in the bivariate case for categorical data only or for qualitative data only, visualization approaches are more accommodating and it is possible to visually explore quantitative-categorical combinations of variables too.

Two quantitative variables

Let us begin with perhaps the best well-known visualization method for two quantitative variables, the scatterplot. A scatterplot is nothing but a plot of two variables where the values are mapped using points to positions in the x- and y- axes:

```
ggplot(data = auctions) +
  geom_point(aes(x = days_on_market,
                 y = premium))
```

```
## Warning: Removed 6 rows containing missing values (geom_point).
```

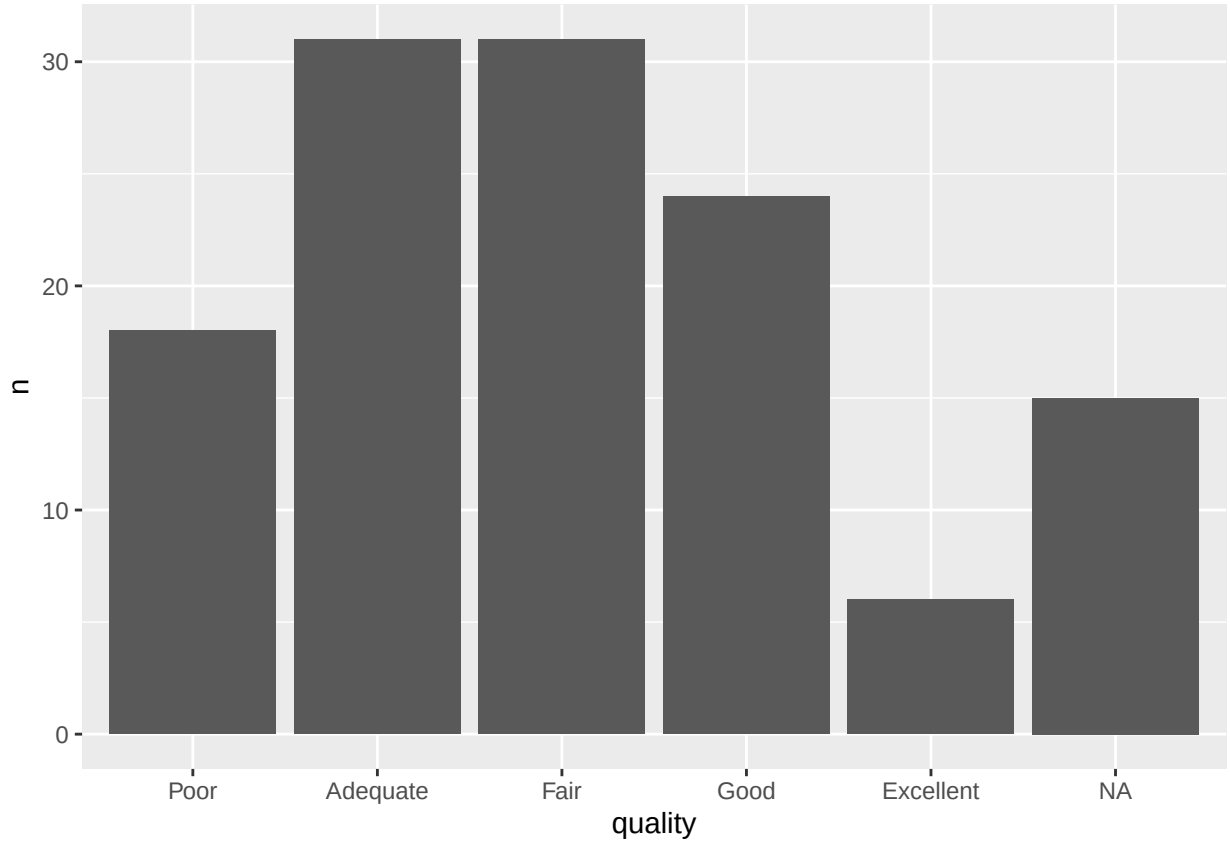


There is quite a bit of scatter in this plot, which explains why the correlation between these two variables was so low ($r = 0.083$).

Two categorical variables

As an alternative, we can tabulate the data first and then use `geom_col()`

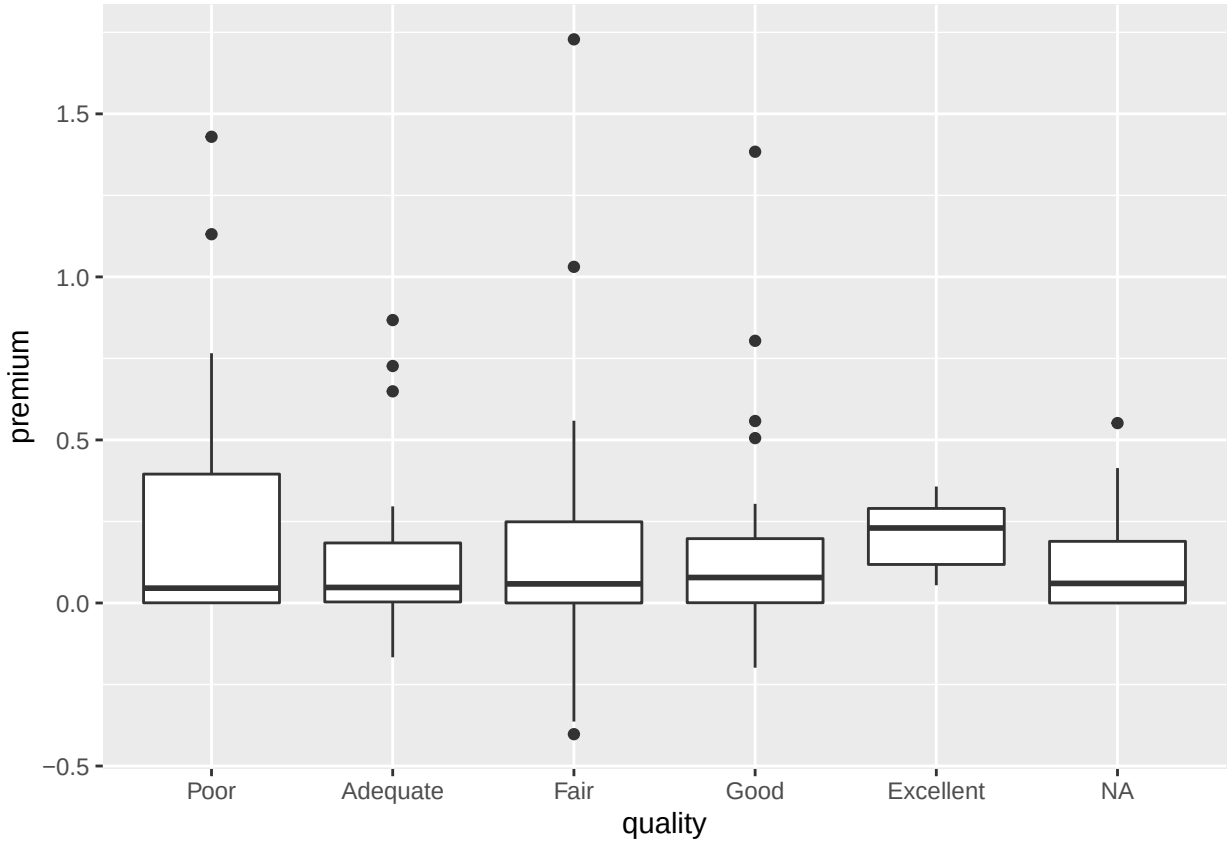
```
auctions |>
  tabyl(quality) |>
  ggplot() +
  geom_col(aes(x = quality,
               y = n))
```



One categorical and one quantitative variable

```
ggplot(data = auctions) +
  geom_boxplot(aes(x = quality,
                   y = premium))
```

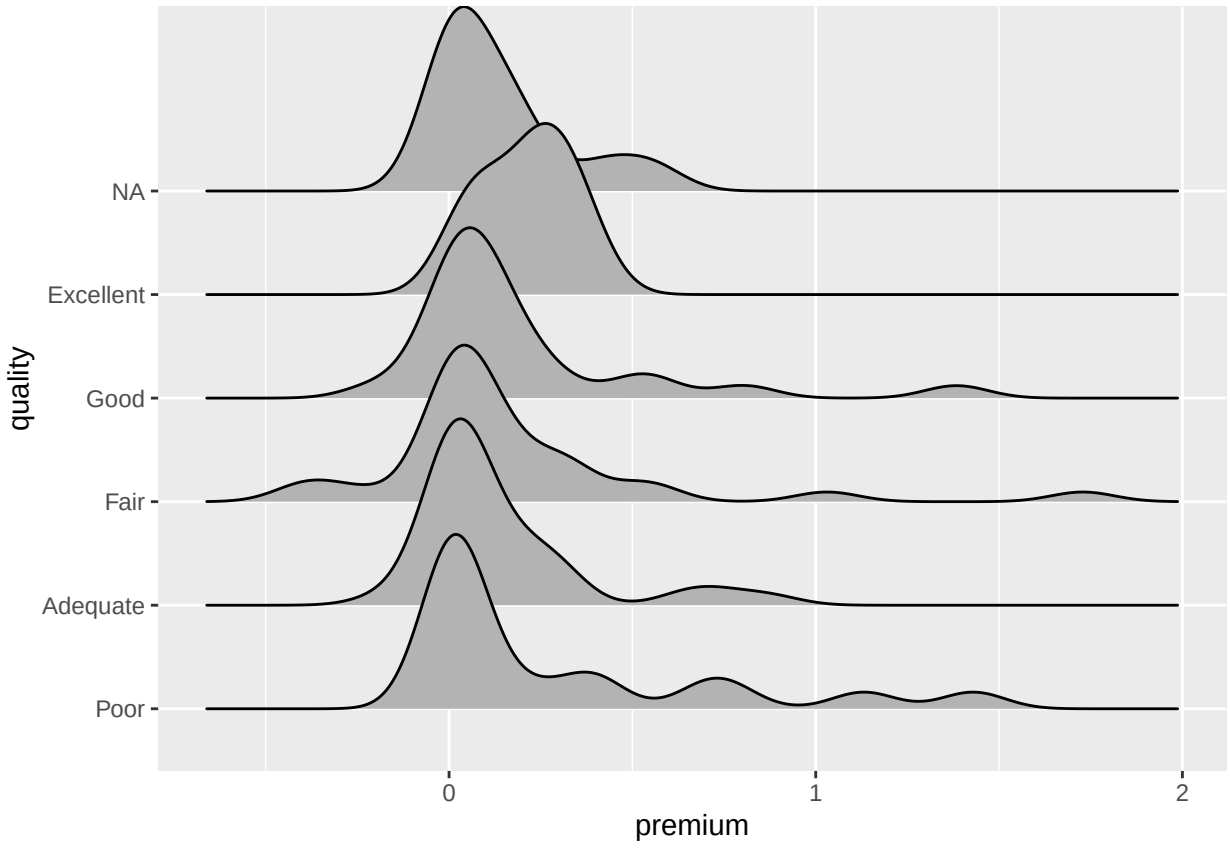
```
## Warning: Removed 3 rows containing non-finite values (stat_boxplot).
```




```
ggplot(data = auctions) +
  geom_density_ridges(aes(x = premium,
                          y = quality))
```

```
## Picking joint bandwidth of 0.0862
```

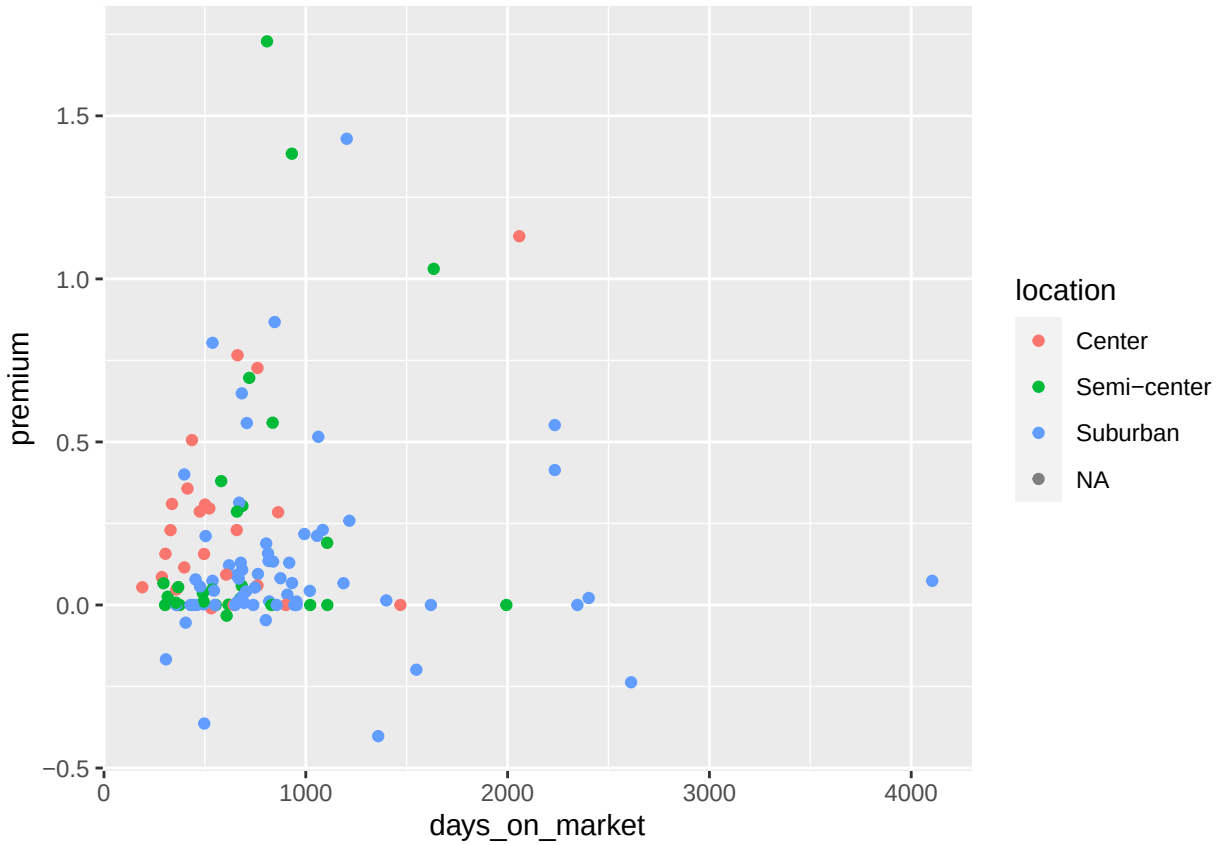
```
## Warning: Removed 3 rows containing non-finite values (stat_density_ridges).
```



Multivariate description

```
ggplot(data = auctions) +
  geom_point(aes(x = days_on_market,
                 y = premium,
                 color = location))
```

```
## Warning: Removed 6 rows containing missing values (geom_point).
```




```
auctions |>
  filter(location == "Center") |>
  select(days_on_market,
         premium) |>
  correlate(method = "pearson",
            use = "pairwise.complete.obs")
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
## # A tibble: 2 x 3
##   term           days_on_market premium
##   <chr>                   <dbl>   <dbl>
## 1 days_on_market         NA       0.484
## 2 premium                 0.484  NA
```

The correlation between days on market and premium is actually fairly high for properties in the center of their respective cities.

Practice

1. Join tables `auctions_amf`, `auctions_pf`, `auctions_phy`, and `auctions_sef`.
2. Imagine that you are interested in the discount in distressed property sales. The discount (check `?auction_amf`) is the percent variation between the initial listed price (market value as apprised) and selling price. Describe and discuss this variable.

3. How does the discount relate to other variables in the data set? Discuss.

4. Propose some hypotheses about discount based on your exploration of the data set. How would you propose to investigate these hypotheses?

5. Imagine now that you are interested in the state of maintenance of properties that are sold in distressed conditions (check `?auction_phy`). Repeat questions 3 and 4 but for this variable.

6. What can you say so far about the relationship between discount and the categorical variables in the data set? Or between state of maintenance and the quantitative variables in the data set? Propose an approach to explore a combination of categorical and quantitative variables.