Session 4. Exploratory data analysis II: Visualization techniques

2026-02-13

Highlights:

This is my mini-reflection. Paragraphs must be indented.

It can contain multiple paragraphs.

Write the concepts that in your opinion are threshold concepts in this exercise. A threshold concept is a key idea that once you grasp it, it changes your understanding of a topic, phenomenon, subject, method, etc. Write between three and five threshold concepts that apply to your learning experience working on this exercise.

"The value of experience is not in seeing much, but in seeing wisely."

—– William Osler

Session outline

- Why visualization?
- The Grammar of Graphics in `Python` (`Seaborn` + `Matplotlib`)
- Building a Plot Step by Step
- Encoding Information
- 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

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

```python
import pandas as pd
import numpy as np
import stemgraphic
import matplotlib.pyplot as plt
import seaborn as sns

# Set display options to show all rows and columns
pd.set_option('display.max_rows', None)    # Show all rows
pd.set_option('display.max_columns', None) # Show all columns
pd.set_option('display.width', None)       # Auto-detect width
pd.set_option('display.max_colwidth', None) # Show full column content
```

We also will utilize some data from the `edashop` R package. To convert these R data files to `Python` files, we will use the `reticulate` package:

```
library(edashop)  # A Package for a Workshop on Exploratory Data Analysis
library(reticulate)
```

From `edashop`, we will also load the following data frames for this session:

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

These data frames contain information about real estate transactions in distressed markets in Italy. You can check the documentation in the usual way:

```
?auctions_amf
```

To be able to convert these datasets in Python, we need to convert them into structures that `Python` recognizes. Following chunk transform them into a Pandas DataFrame:

```
auctions_amf = r.auctions_amf
auctions_pf = r.auctions_pf
auctions_phy = r.auctions_phy
auctions_sef = r.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
            .merge(auctions_pf, on="id", how="left")
            .merge(auctions_phy, on="id", how="left")
            .merge(auctions_sef, on="id", how="left"))
```

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 characteristic 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 technique, 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 numerical items can you remember while reading the following table and trying to distinguish patterns?

```python
auctions_amf[["discount", "premium"]].head(10)
```

```
   discount   premium
0 -0.411289  0.046597
1 -0.300000  0.074098
2 -0.214257  0.047658
3 -0.521978  0.133090
4 -0.624500 -0.198417
5 -0.810868 -0.402248
6 -0.570835 -0.237040
7 -0.480928  0.230392
8 -0.538041  0.095014
9 -0.152365  0.025547
```

What is the central tendency of `b_V001` (permanent private households) taking into account only the numbers shown? Is the `b_V001` more or less spread than the `b_V002` (population living in permanent private households)? 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:

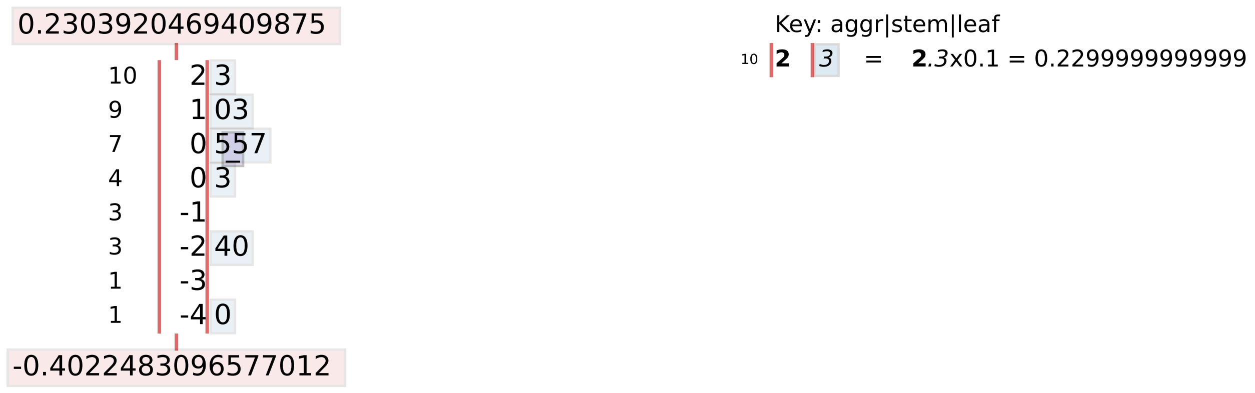
```python
auctions_amf[["discount", "premium"]].head(10).describe()
```

```
        discount    premium
count  10.000000  10.000000
mean   -0.462506  -0.018531
std     0.198819   0.195600
min    -0.810868  -0.402248
25%    -0.562637  -0.142426
50%    -0.501453   0.047127
75%    -0.327822   0.089785
max    -0.152365   0.230392
```

To make matters more difficult, this is only a small part of the full table (only ten rows and two columns). The task of identifying patterns becomes increasingly complicated as the number of observations and the number of variables grow.

Visualization techniques work by *encoding* the data in ways that make 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:

```python
discount_10 = auctions_amf["premium"].head(10).dropna()
stemgraphic.stem_graphic(discount_10, scale=0.1)
plt.show()
```



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* or their *magnitude*. We could use colors to do this: red for values bellow 700, and blue for values above 700. 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, among other spatial attributes. These encodings allow the brain to make sense of the underlying patterns 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_dtypes(include=['number']).corr()
```

```
                 days_on_market  number_auctions  discount   premium
days_on_market         1.000000         0.693218 -0.281579  0.082910
```

```
number_auctions        0.693218         1.000000 -0.545071  0.192099
discount              -0.281579        -0.545071  1.000000  0.437755
premium                0.082910         0.192099  0.437755  1.000000
```

Now compare to:

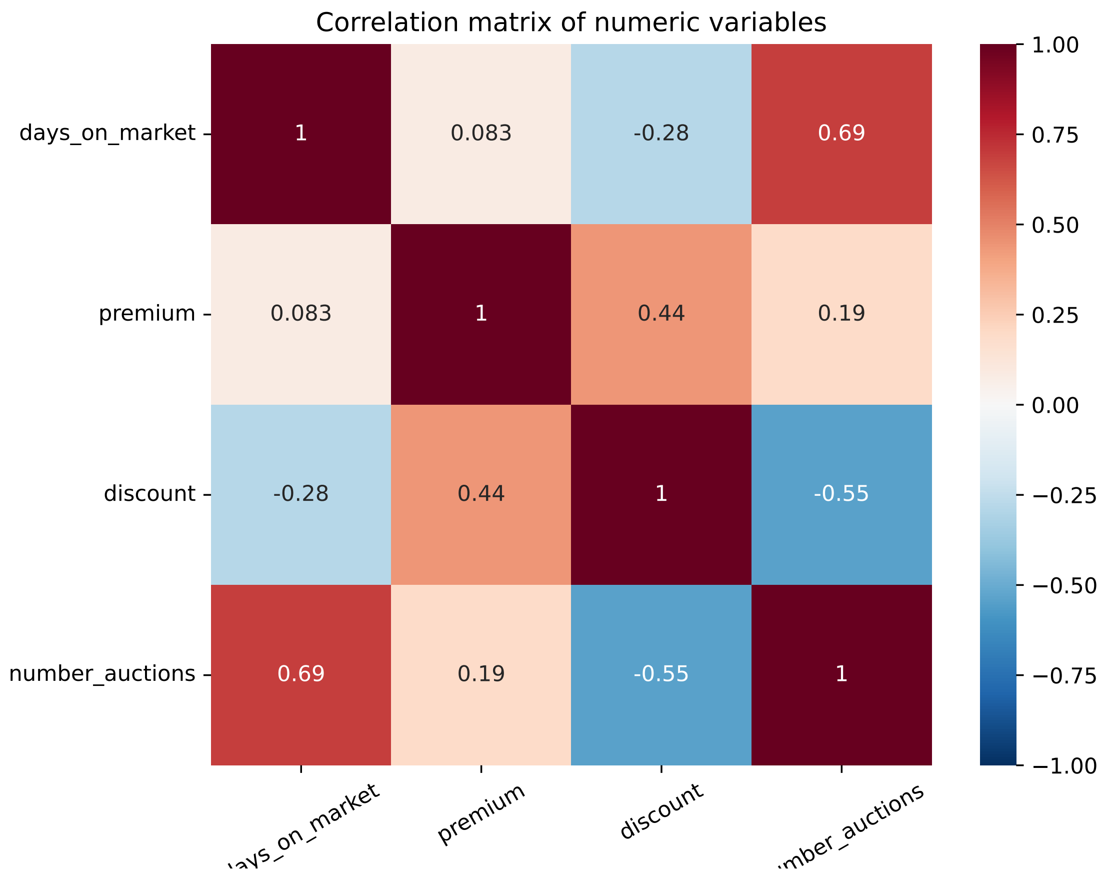
```
corr_matrix = auctions_amf[["days_on_market", "premium", "discount", "number_auctions"]].corr()

fig, ax = plt.subplots(figsize=(8,6))
sns.heatmap(corr_matrix, annot=True, cmap='RdBu_r', center=0, vmin=-1, vmax=1,
            square=True, ax=ax)
ax.set_title("Correlation matrix of numeric variables")
plt.xticks(rotation=30)
```

```
(array([0.5, 1.5, 2.5, 3.5]), [Text(0.5, 0, 'days_on_market'), Text(1.5, 0, 'premium'), Text(2.5, 0, 'di
```

```
plt.yticks(rotation=0)
```

```
(array([0.5, 1.5, 2.5, 3.5]), [Text(0, 0.5, 'days_on_market'), Text(0, 1.5, 'premium'), Text(0, 2.5, 'di
```

```
plt.show()
```



By encoding valence using colors, we can present the same information in a form that we naturally process with greater ease.

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.

Data visualization in Python

Python has different libraries that provide methods for visualizing data information. In this session, we will focus on probably the most well-known and traditional library: `Matplotlib.` As described by its authors, *"Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python."* This library was created by John Hunter during his postdoctoral research in neurobiology, aiming to create plots similar to those available in `MATLAB` (a programming and numeric computing platform popular with engineers, mathematicians, physicists, and researchers), and it is currently the most widely used visualization package for plotting in Python.

Based on `Matplotlib`, other libraries were created to expand its functionality, such as `seaborn`. In fact, we used `Matplotlib` and `seaborn` to plot the two correlation matrices from the two previous chunks. `seaborn` is a `Python` library that implements many of these ideas on top of matplotlib. It works directly with pandas DataFrames and allows us to map columns to visual properties (colour, size, shape, etc.) with simple keywords. Usually, when we need more control, we drop down to `Matplotlib` to adjust legends, axes, and annotations.

To demonstrate the use of both libraries, we will start with the following example: a scatter plot to visualize a correlation matrix. 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:

```python
corr_data = auctions_amf[["days_on_market", "premium", "discount", "number_auctions"]].corr()

mask = np.triu(np.ones_like(corr_data, dtype=bool), k=1)
corr_masked = corr_data.mask(mask)

corr_long = corr_masked.stack().reset_index()
corr_long.columns = ['term_1', 'term_2', 'r']
corr_long = corr_long.dropna()
term1_order = ["days_on_market", "premium", "discount", "number_auctions"]
term2_order = ["days_on_market", "number_auctions", "discount", "premium"]   # adjust to keep matrix sha

corr_long['term_1'] = pd.Categorical(corr_long['term_1'], categories=term1_order, ordered=True)
corr_long['term_2'] = pd.Categorical(corr_long['term_2'], categories=term2_order, ordered=True)
corr_long = corr_long.dropna()

my_r_matrix = corr_long.copy()
my_r_matrix["abs_r"] = my_r_matrix["r"].abs()
```

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`:

```
my_r_matrix.head()
```

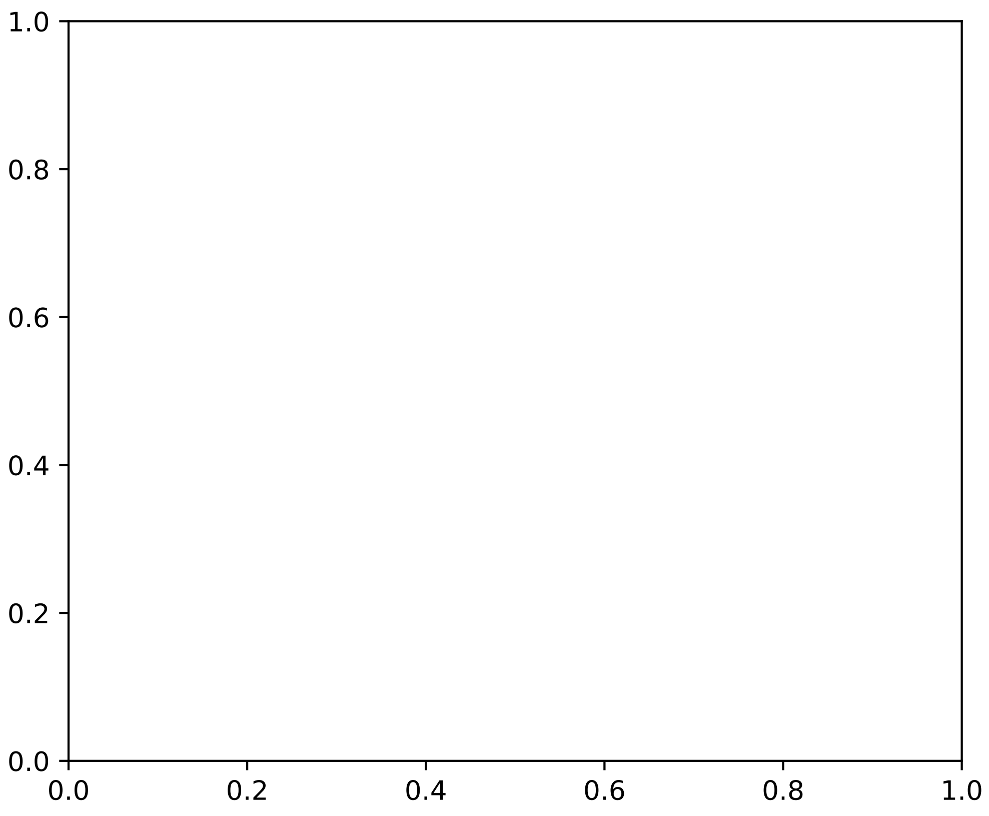
```
           term_1          term_2         r     abs_r
0  days_on_market  days_on_market  1.000000  1.000000
1         premium  days_on_market  0.082910  0.082910
2         premium         premium  1.000000  1.000000
3        discount  days_on_market -0.281579  0.281579
4        discount         premium  0.437755  0.437755
```

```
my_r_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   term_1  10 non-null     category
 1   term_2  10 non-null     category
 2   r       10 non-null     float64
 3   abs_r   10 non-null     float64
dtypes: category(2), float64(2)
memory usage: 716.0 bytes
```

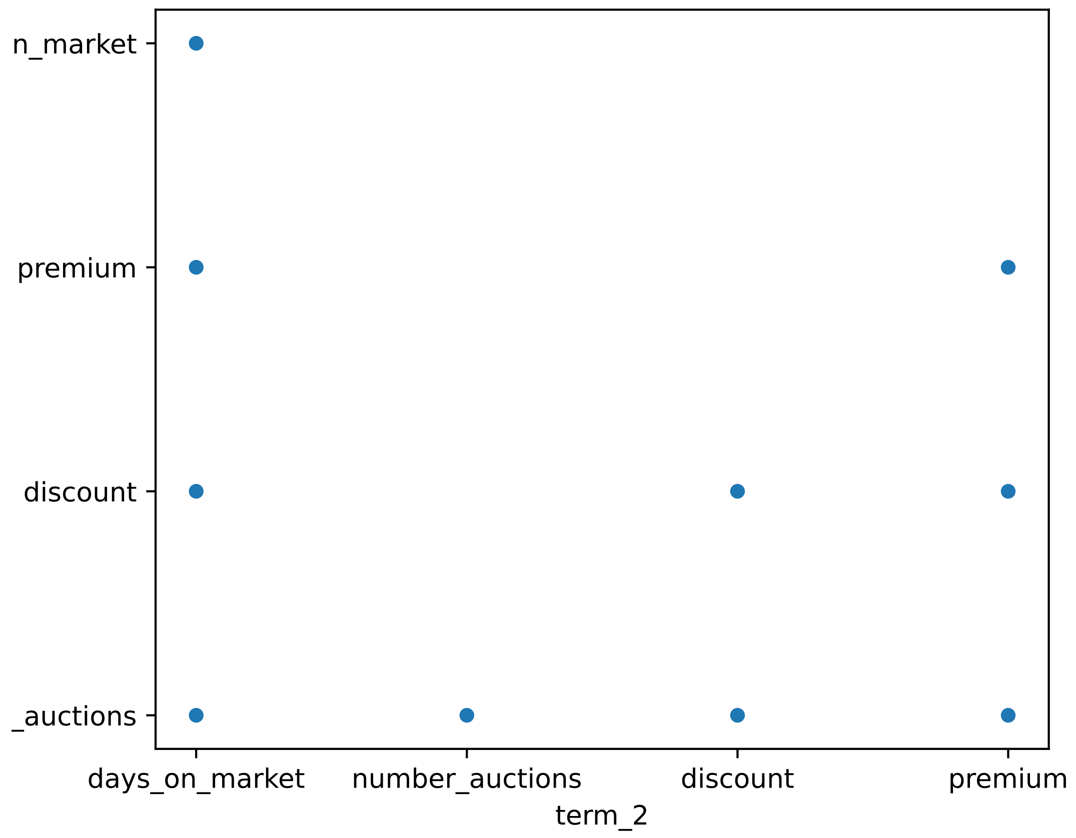
In `matplotlib`/`seaborn` we first create a figure and axes. This is our “blank canvas”.

```
fig, ax = plt.subplots(figsize = (6,5))
```



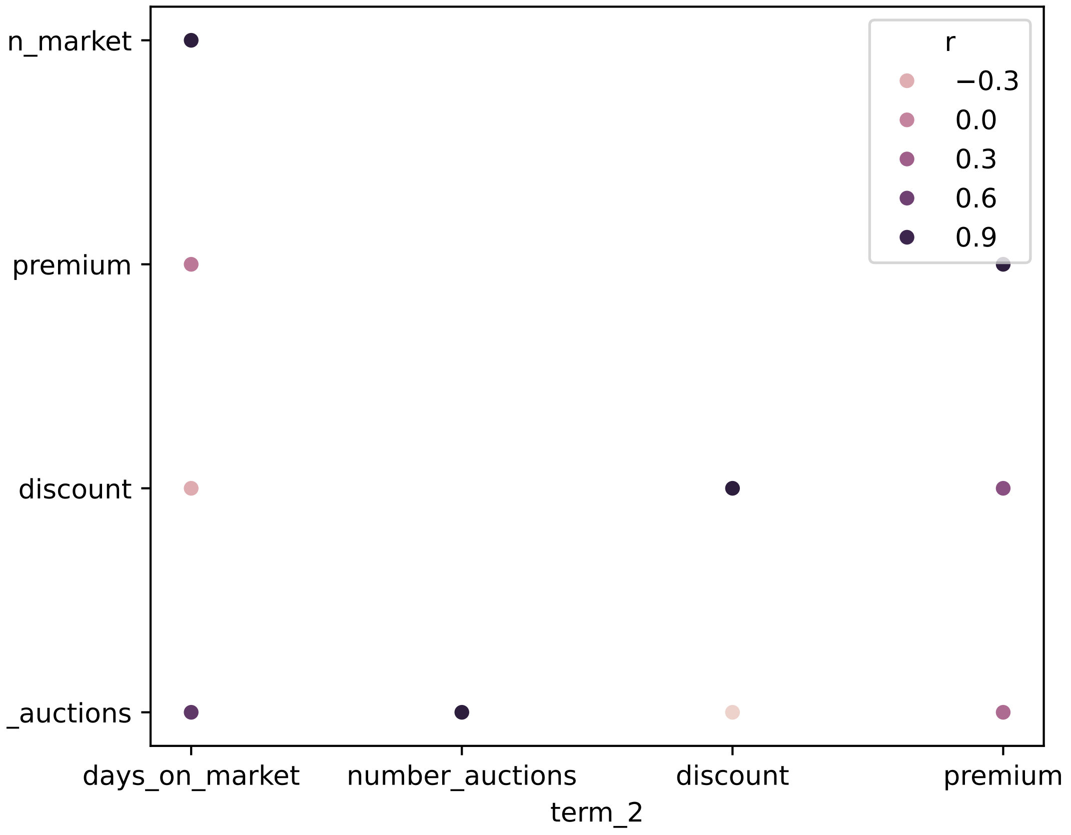
A blank canvas is not yet a statistical plot. Although we already created the figures and axes, we have not yet stated *what* we wish to plot. In this case, we wish to represent correlations as points, so we use `sns.scatterplot()`. Now we need to map variables in our table to the plot. Points need to 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 map `term_2` to x, `term_1` to y, setting the our dataset as the `data`:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1', ax = ax)
plt.show()
```



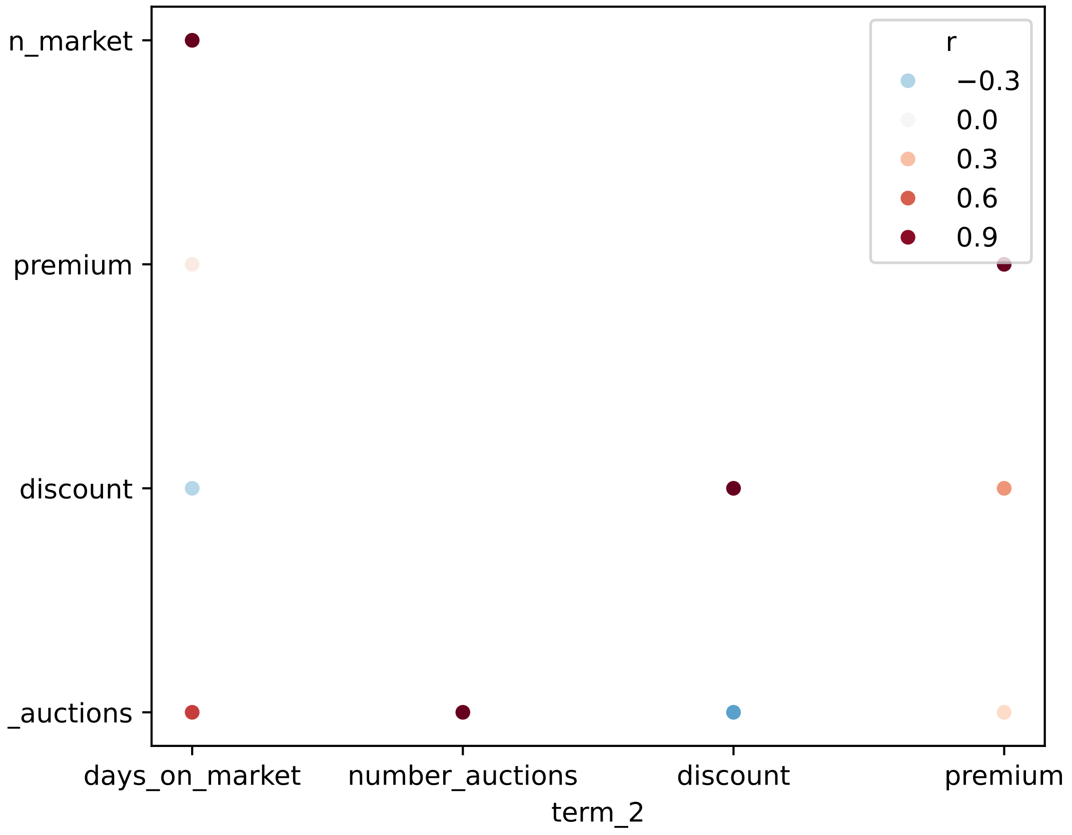
Now we have points in the figure. But so far we have encoded (i.e., mapped) only *position* (for which we used two variables, `term_1` and `term_2`). Next, we also want to represent the correlation, something that we can do by encoding variable `r` using color (`hue`):

```
fig, ax = plt.subplots(figsize = (6,5))
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1', hue = 'r', ax = ax)
plt.show()
```



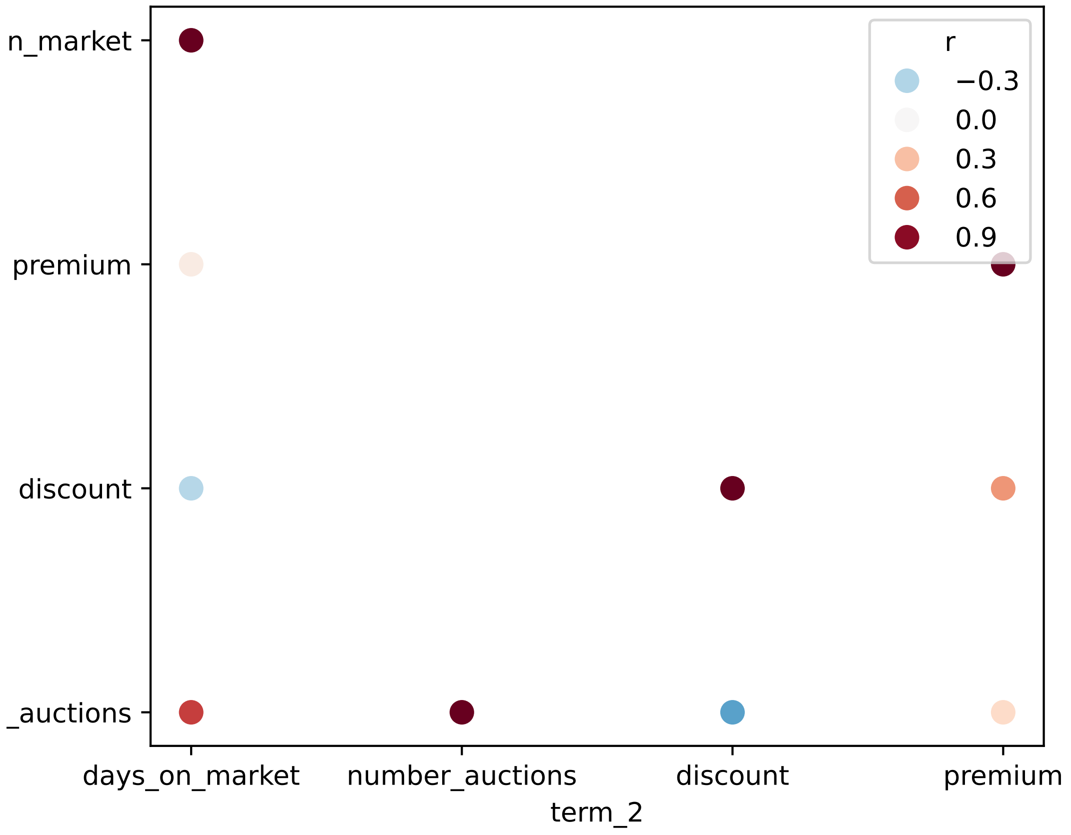
The points now map the correlation to a color. The default colors, though, are not very effective: correlation ranges between -1 and 1, but a sequential color scale does not correctly convey the change in valence in correlation (positive and negative). We can replace the default palette with a diverging one (`RdBu_r` that stands for red-white-blue). 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 (with `hue_norm` ranging from -1 to 1):

```
fig, ax = plt.subplots(figsize = (6,5))
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                hue ='r', palette = 'RdBu_r', hue_norm = (-1, 1), ax =ax)
plt.show()
```



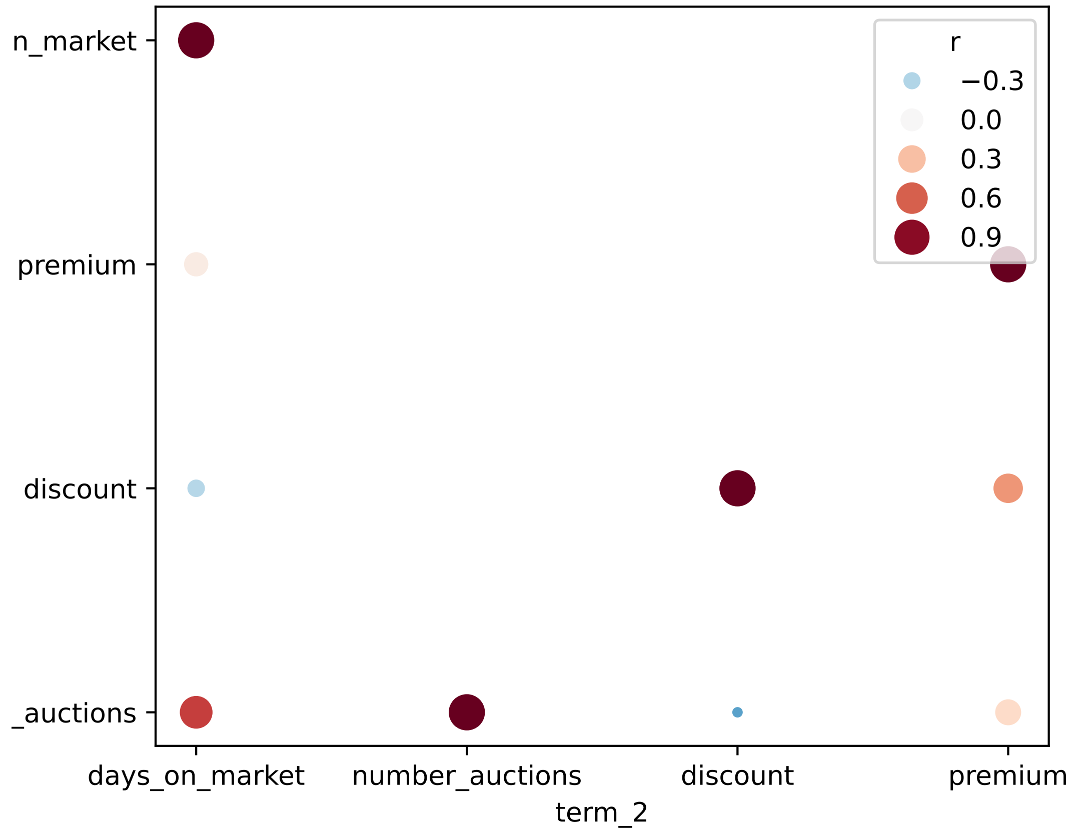
The colors let us more easily perceive the *magnitude* as well as the *direction* of the correlation. That said, the points are too small. We can control for the size of the cycles by changing the parameter `s`:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                s=100, ax = ax)
plt.show()
```



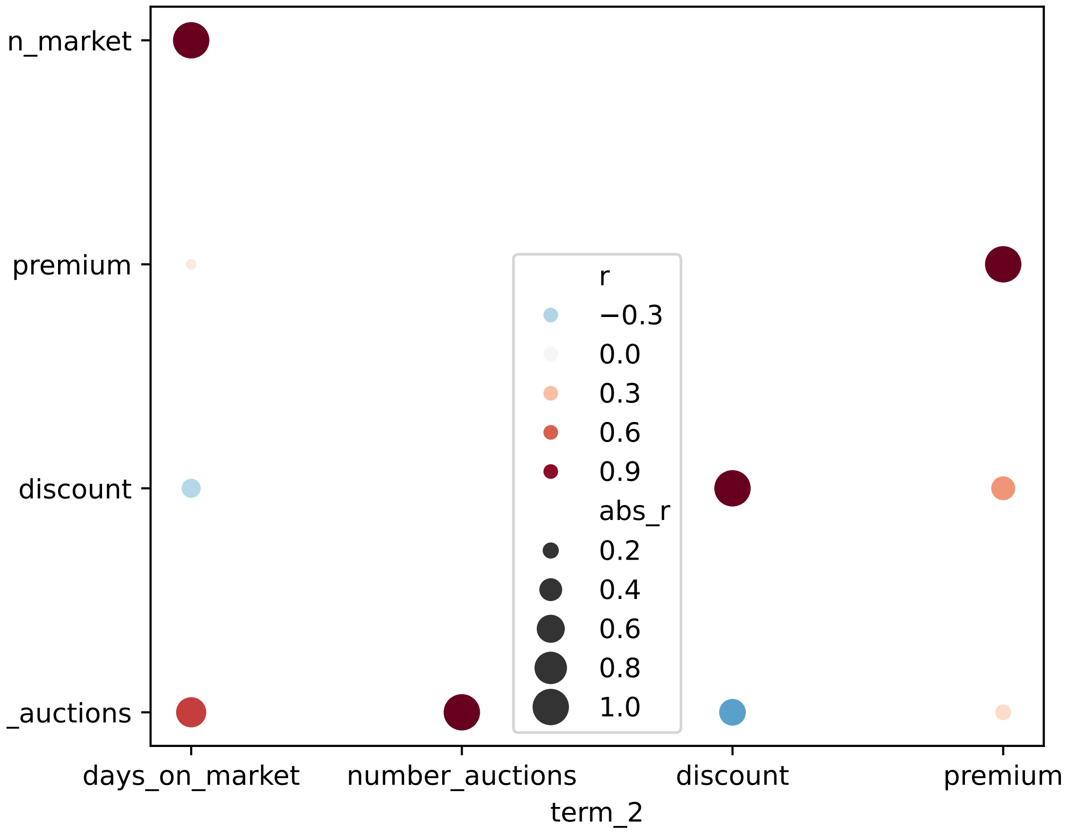
It is easier to see the colors. But notice how the `size` is a constant and does not map to any variable. This is why the points are all the same size. What if we mapped size to the correlation? in the chunk we map size `s` to the absolute value (`r`) of the correlation, so that strength is shown by size:

```
fig, ax = plt.subplots(figsize = (6,5))
sc = sns.scatterplot(data = my_r_matrix, x ='term_2', y = 'term_1',
                     hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                     size ="r", sizes = (20, 200), ax = ax)
plt.show()
```



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:

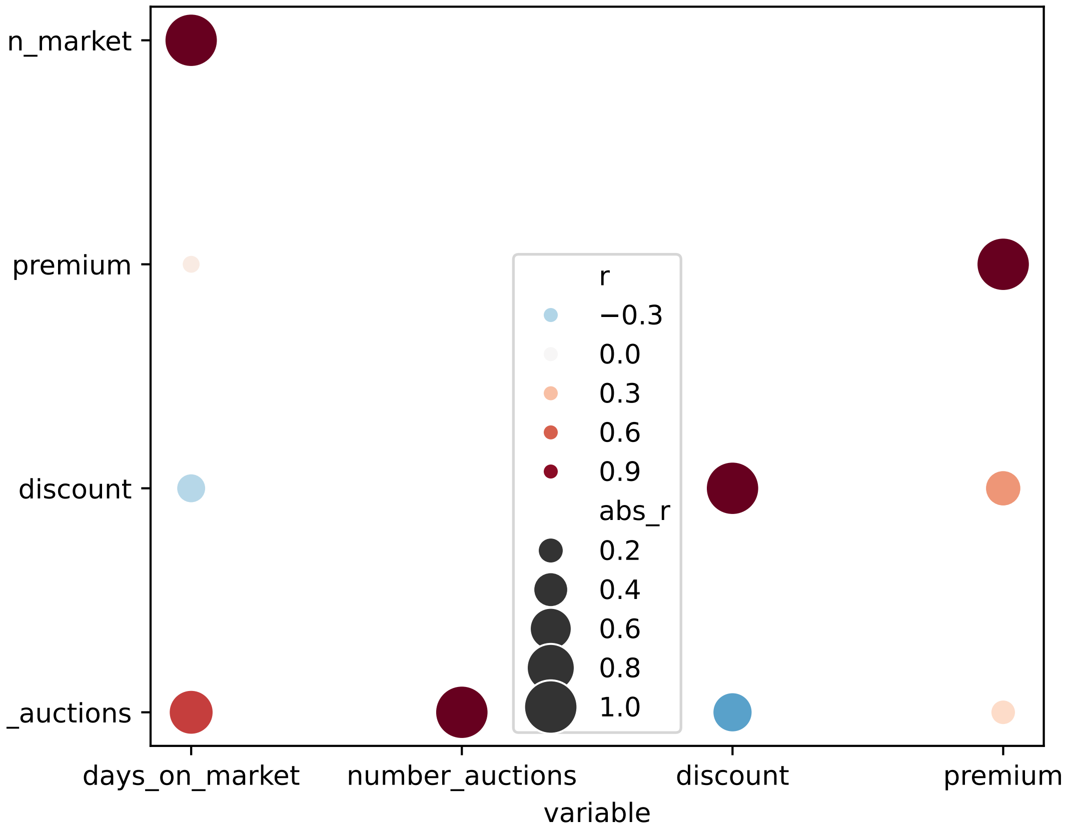
```
my_r_matrix["abs_r"] = my_r_matrix["r"].abs()

fig, ax = plt.subplots(figsize=(6,5))
sc = sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                     hue='r', palette='RdBu_r', hue_norm=(-1, 1),
                     size = "abs_r", sizes = (20, 200), ax = ax)
plt.show()
```



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

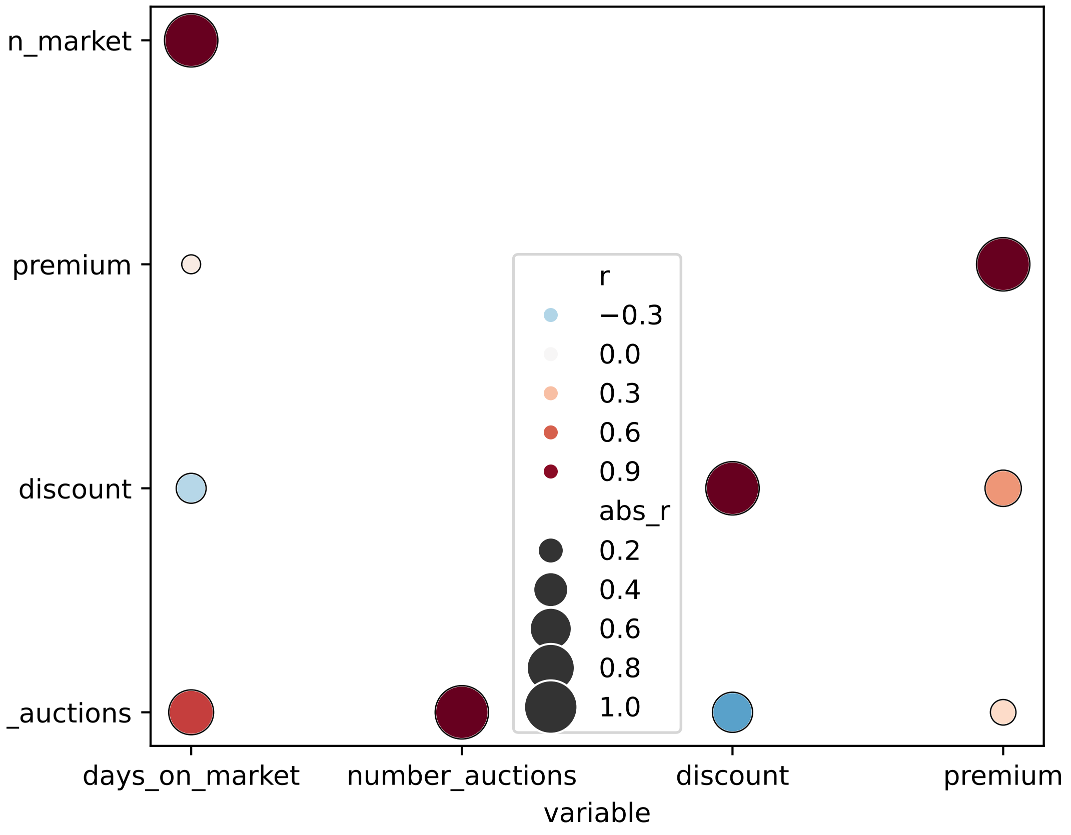
We can also adorn the phrase that builds our plot. For example, we can set the legend title for each axis:

```
fig, ax = plt.subplots(figsize = (6,5))
sc = sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                     hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                      size = "abs_r", sizes = (50, 400), ax = ax)

ax.set_xlabel("variable")
ax.set_ylabel("variable")
plt.show()
```



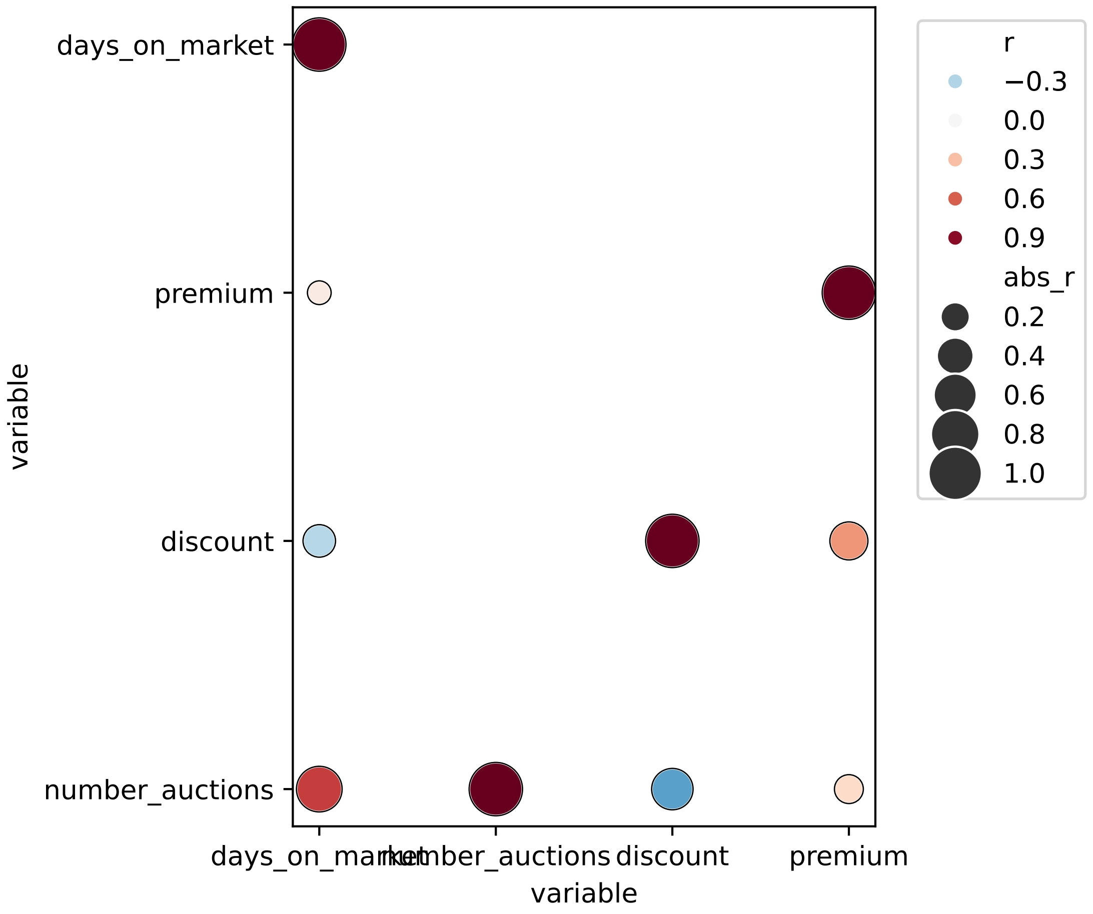
To make the symbols easier to see, we can layer another set of points using a different symbol (a circle without fill) that maps size to absolute correlation but uses only the default color (black) as a constant:

```
fig, ax = plt.subplots(figsize = (6,5))

# coloured filled points
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                 size = "abs_r", sizes = (50, 400),
                legend = 'brief', ax = ax)

# black outline
sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                 size = "abs_r", sizes = (50, 400),
                facecolor = 'none', edgecolor = 'black', linewidth = 0.5,
                legend = False, ax=ax)

ax.set_xlabel("variable")
ax.set_ylabel("variable")
plt.show()
```



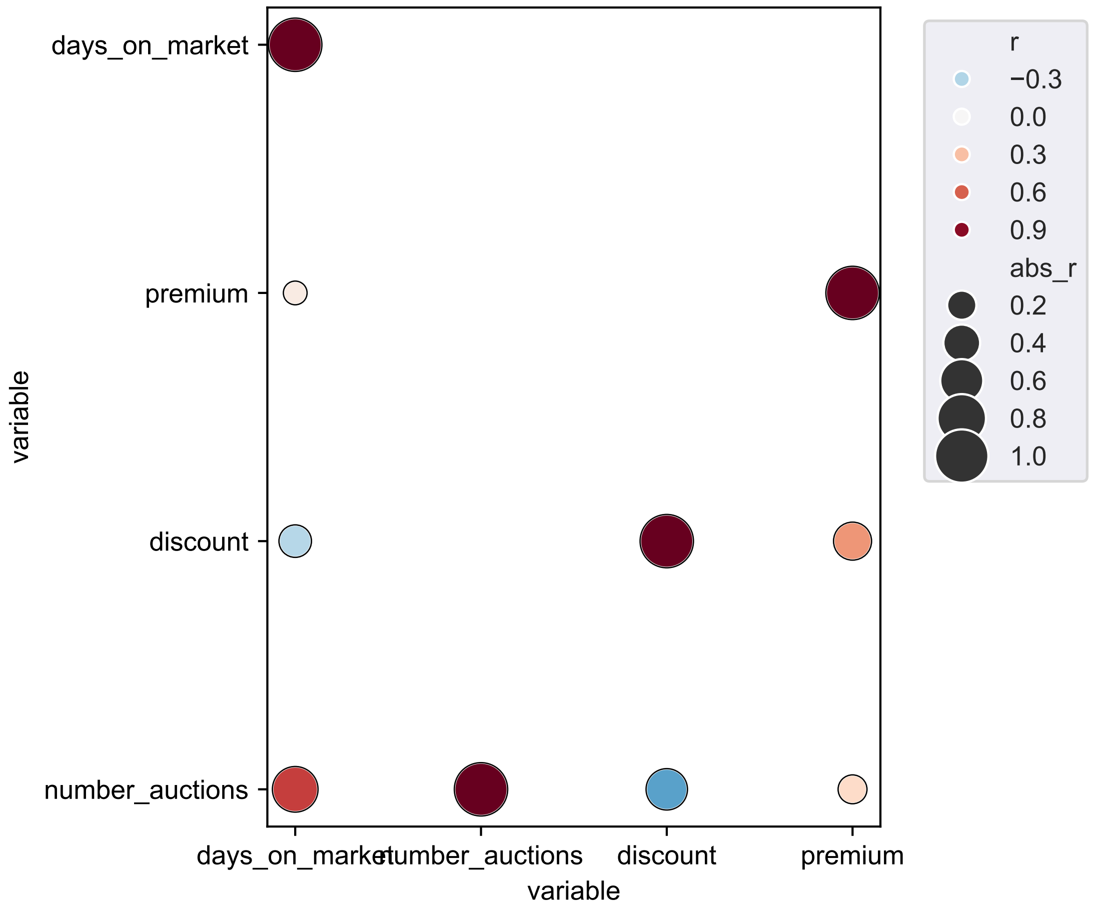
We can move the legend, rotate text, etc:

```
fig, ax = plt.subplots(figsize=(6,5))

sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                size = 'abs_r', sizes = (50, 400), size_norm = (0, 1),
                legend =  'brief' , ax = ax)

sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                size = 'abs_r', sizes = (50, 400), size_norm = (0, 1),
                facecolor = 'none', edgecolor = 'black', linewidth = 0.5,
                legend = False, ax = ax)

ax.set_xlabel("variable")
ax.set_ylabel("variable")

legend = ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



A number of pre-defined themes exist that give different looks to a plot. There are five preset seaborn themes: `darkgrid`, `whitegrid`, `dark`, `white`, and `ticks`, with the default theme being `darkgrid`:

```
fig, ax = plt.subplots(figsize=(6,5))

sns.set_style("dark")

sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                hue = 'r', palette = 'RdBu_r', hue_norm = (-1, 1),
                size = 'abs_r', sizes = (50, 400), size_norm = (0, 1),
                legend =  'brief' , ax = ax)

sns.scatterplot(data = my_r_matrix, x = 'term_2', y = 'term_1',
                size = 'abs_r', sizes = (50, 400), size_norm = (0, 1),
                facecolor = 'none', edgecolor = 'black', linewidth = 0.5,
                legend = False, ax = ax)

ax.set_xlabel("variable")
ax.set_ylabel("variable")

legend = ax.legend(bbox_to_anchor=(1.05, 1), loc='upper left')

plt.tight_layout()
plt.show()
```



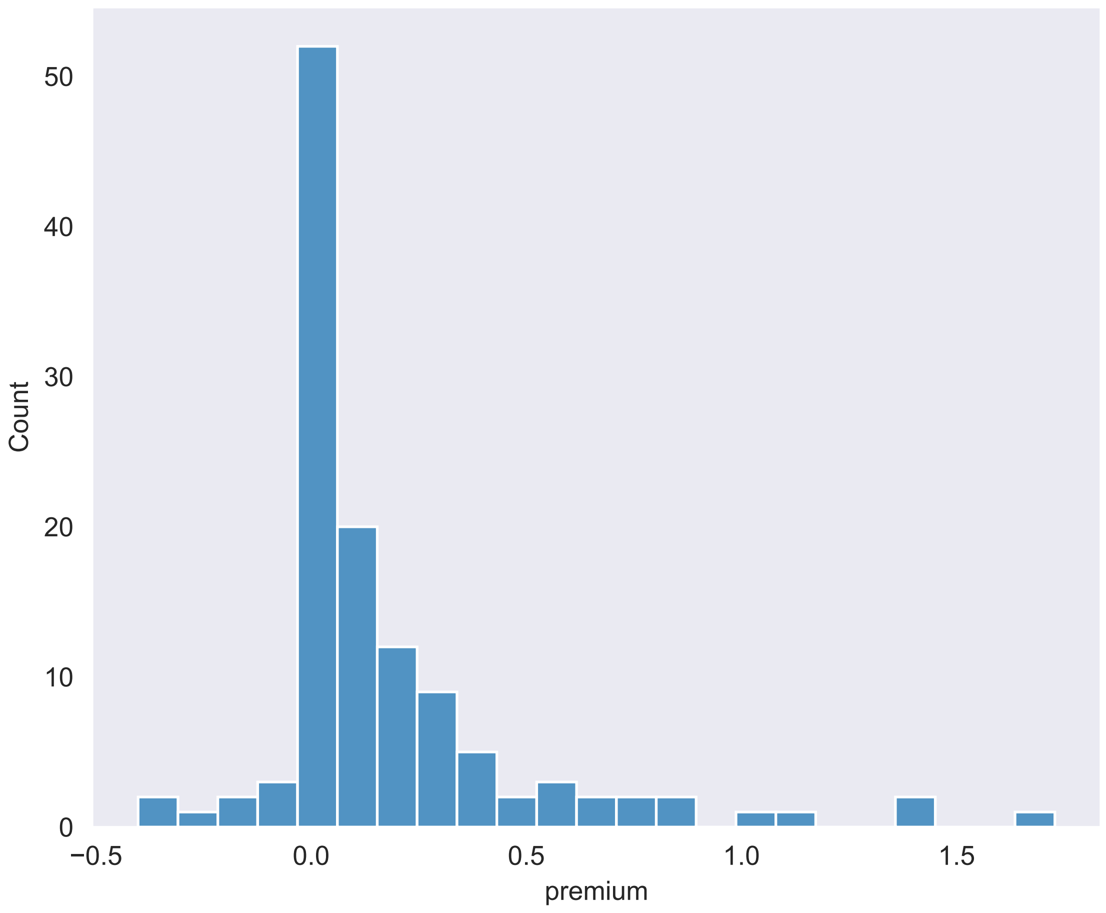
`sns.scatterplot` is one of the possible options to create plots in Python.

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 functions for statistical plots. Which functions 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 function is a histogram, implemented as `sns.histplot()`. Using the variable `premium` (premium paid by the winner bidder to win the auction calculated as the percentage variation between the last listing value and the final selling price), we have:

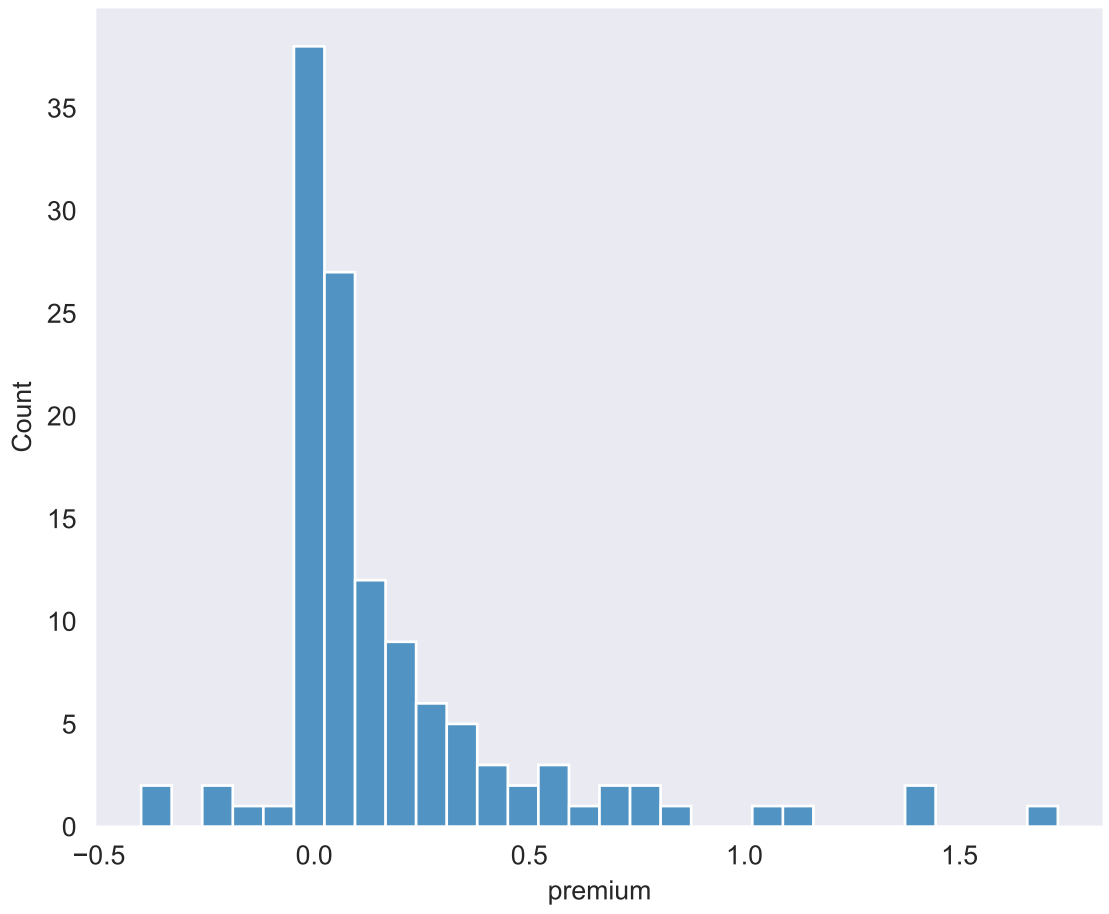
```
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = auctions, x = 'premium', ax = ax)
plt.tight_layout()
plt.show()
```



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. Since we did not defined the number of bins for the previous plot, seaborn determined the number of equally spaced bins to span the range of the data (similar to what expected for a frequency plot).

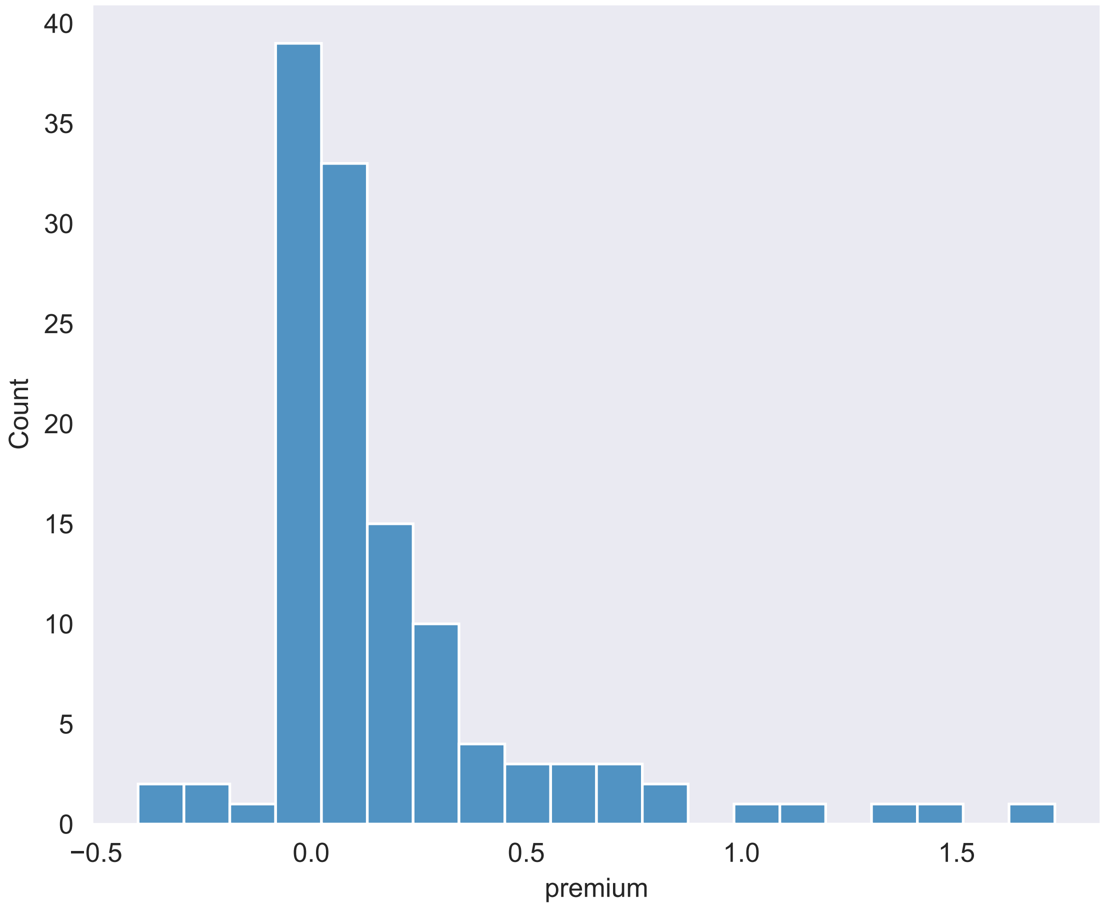
For instance, for 30 bins, we set the parameter `bins` equal to 30:

```python
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = auctions, x = 'premium',  bins = 30, ax = ax)
plt.tight_layout()
plt.show()
```



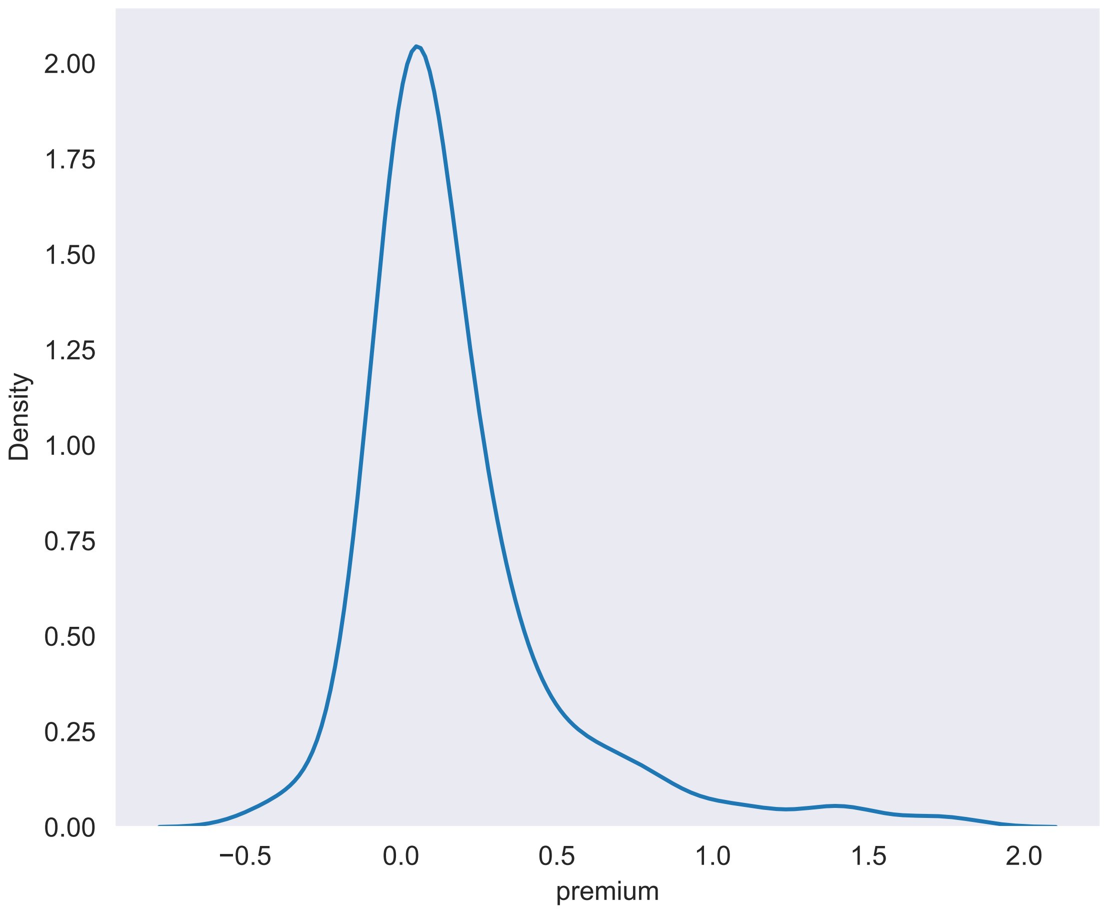
For 20 bins:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = auctions, x = 'premium',  bins = 20, ax = ax)
plt.tight_layout()
plt.show()
```



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

```python
fig, ax = plt.subplots(figsize = (6,5))
sns.kdeplot(data = auctions, x = 'premium', ax = ax)
plt.tight_layout()
plt.show()
```



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

```
mean_premium = auctions['premium'].mean()
median_premium = auctions['premium'].median()

mean_premium
```

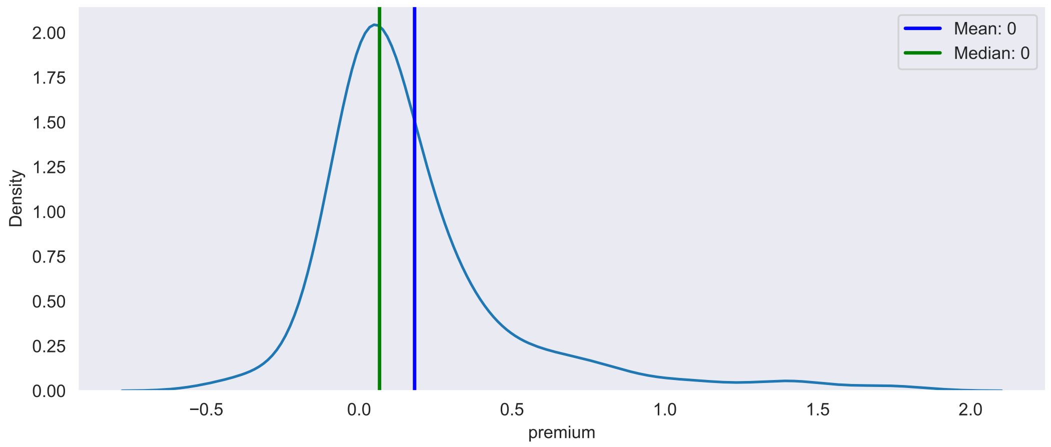
```
0.18147271028825585
```

```
median_premium
```

```
0.06676808541464666
```

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 `axvline()` to draw vertical lines (the mean in blue, the median in green):

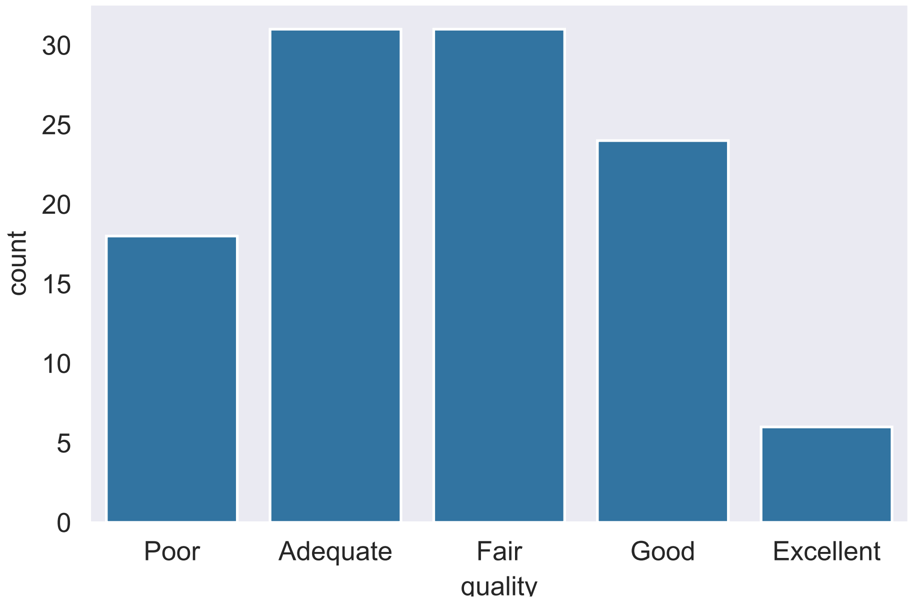
```
fig, ax = plt.subplots(figsize = (10,4))
sns.kdeplot(data = auctions, x = 'premium', ax = ax)
ax.axvline(mean_premium, color = 'blue', linewidth = 2, label = f'Mean: {mean_premium:.0f}')
ax.axvline(median_premium, color = 'green', linewidth = 2, label = f'Median: {median_premium:.0f}')
ax.legend()
plt.show()
```



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 `sns.countplot()`. 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:

```
fig, ax = plt.subplots()
sns.countplot(data = auctions, x = 'quality', ax = ax)
plt.show()
```

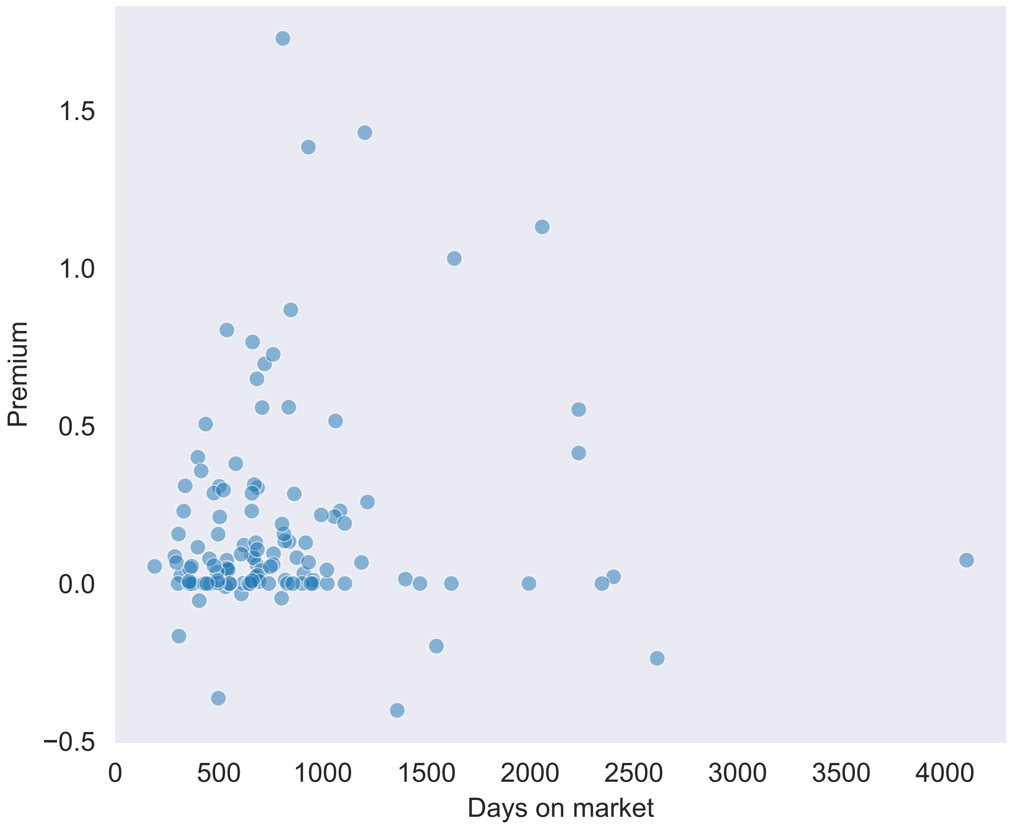


Bivariate description

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:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.scatterplot(data = auctions, x = 'days_on_market', y = 'premium', alpha = 0.5, ax = ax)
ax.set_xlabel("Days on market")
ax.set_ylabel("Premium")
plt.show()
```



We can visualize the correlation between the two variables, `days_on_market` and `premium`, by doing:

```
auctions[['days_on_market', 'premium']].corr()
```

```
                days_on_market  premium
days_on_market         1.00000  0.08291
premium                0.08291  1.00000
```

How does this correlation explain the scatter plot shown above?

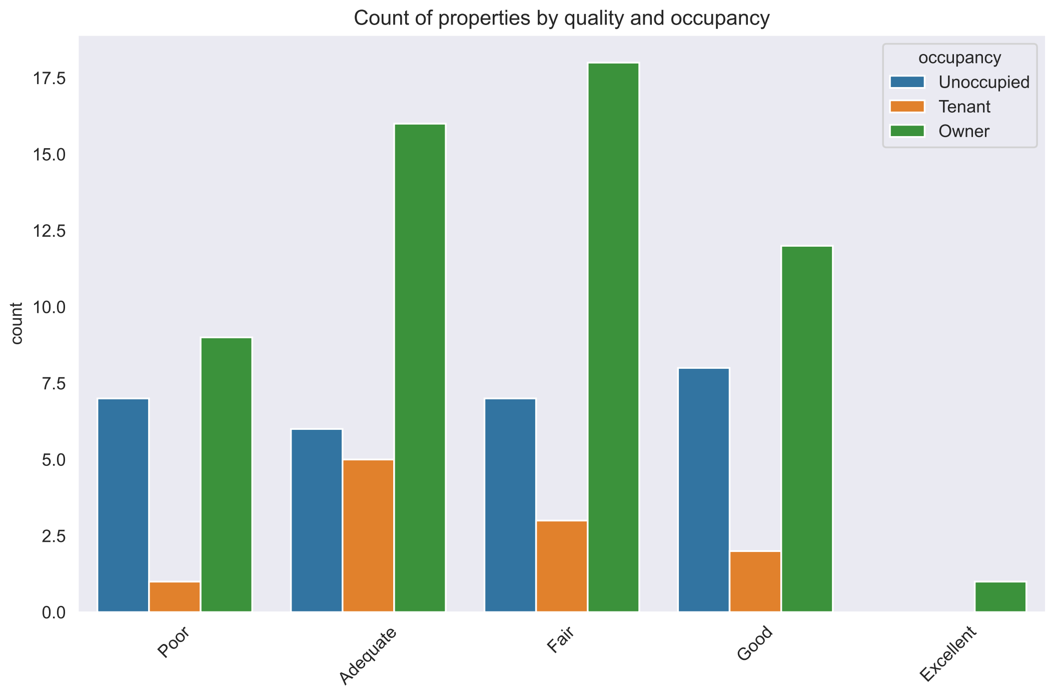
Two categorical variables

Two categorical variables can be explored by means of count plots:

```
fig, ax = plt.subplots(figsize=(10,6))
sns.countplot(data=auctions, x='quality', hue='occupancy', ax=ax)
ax.set_title("Count of properties by quality and occupancy")
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4], [Text(0, 0, 'Poor'), Text(1, 0, 'Adequate'), Text(2, 0, 'Fair'), Text(3, 0, 'Good'), 
```

```
plt.show()
```

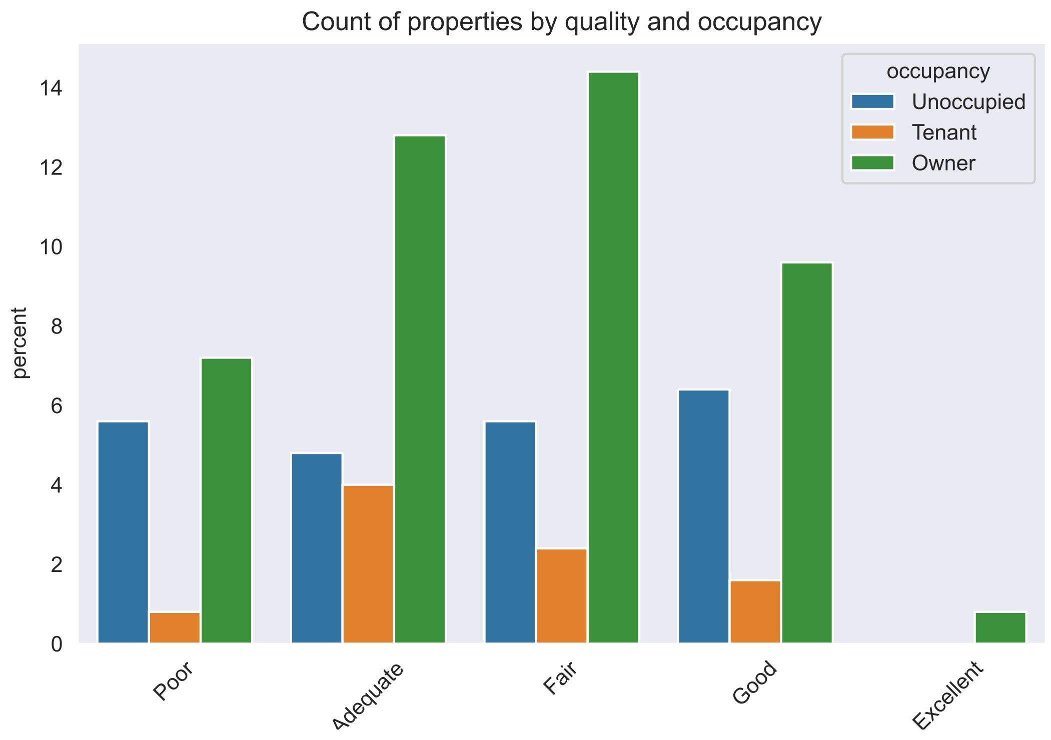


This information can plotted in terms of percentages by setting `stat = percent`:

```
fig, ax = plt.subplots(figsize = (8,5))
sns.countplot(x = "quality", hue =  "occupancy", data = auctions, stat = "percent")
ax.set_title("Count of properties by quality and occupancy")
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4], [Text(0, 0, 'Poor'), Text(1, 0, 'Adequate'), Text(2, 0, 'Fair'), Text(3, 0, 'Good'), 
```

```
plt.show()
```



Count plots are a visual alternative to a cross-tabulation.

```
pd.crosstab(auctions['quality'], auctions['occupancy'])
```

```
occupancy  Unoccupied  Tenant  Owner
quality
Poor                7       1      9
Adequate            6       5     16
Fair                7       3     18
Good                8       2     12
Excellent           0       0      1
```

As you can see, it is much easier to retain this information using a visual technique.

One categorical and one quantitative variable

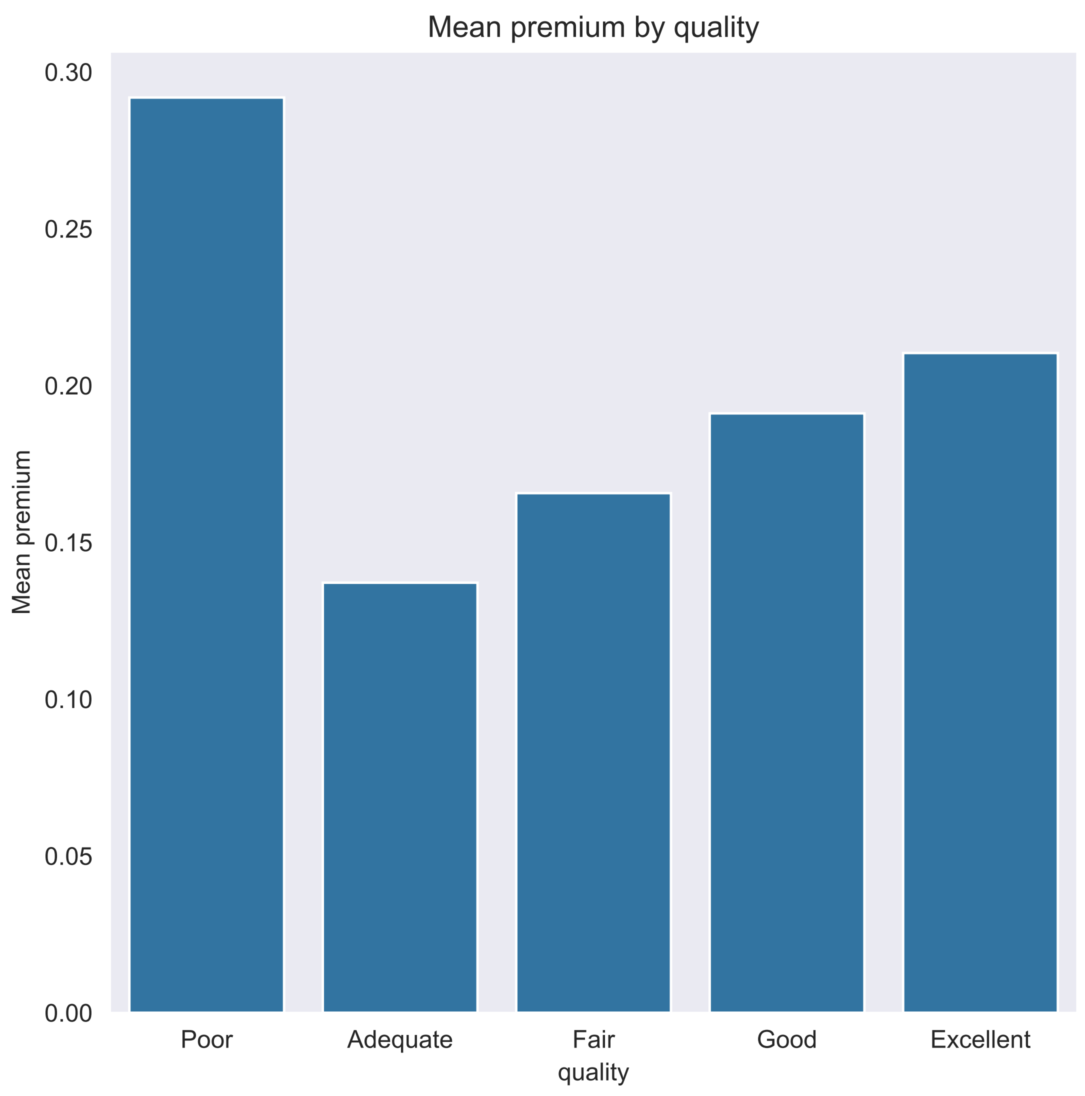
Unlike summary statistics that are defined in the bivariate case for categorical data only or for qualitative data only, visualization approaches are more accommodating and it is possible to visually explore quantitative-categorical combinations of variables too.

There are a few visualization techniques that accommodate combinations of one categorical and one quantitative variable. Column plots can map a categorical variable to one axis and a quantitative variable to the other. In this example, we calculate a summary statistic by group (the mean `premium` by quality of `occupancy`) and then use `sns.barplot()` to visualize these two variables:

```
mean_by_occupancy = auctions.groupby('occupancy')['premium'].mean().reset_index()
```

```
<string>:1: FutureWarning: The default of observed=False is deprecated and will be changed to True in a
```

```python
fig, ax = plt.subplots(figsize=(7,7))
sns.barplot(data=auctions, x = 'quality', y = 'premium',
            order=['Poor', 'Adequate', 'Fair', 'Good', 'Excellent'],
            errorbar = None, ax=ax)
ax.set_ylabel("Mean premium")
ax.set_title("Mean premium by quality")
plt.show()
```



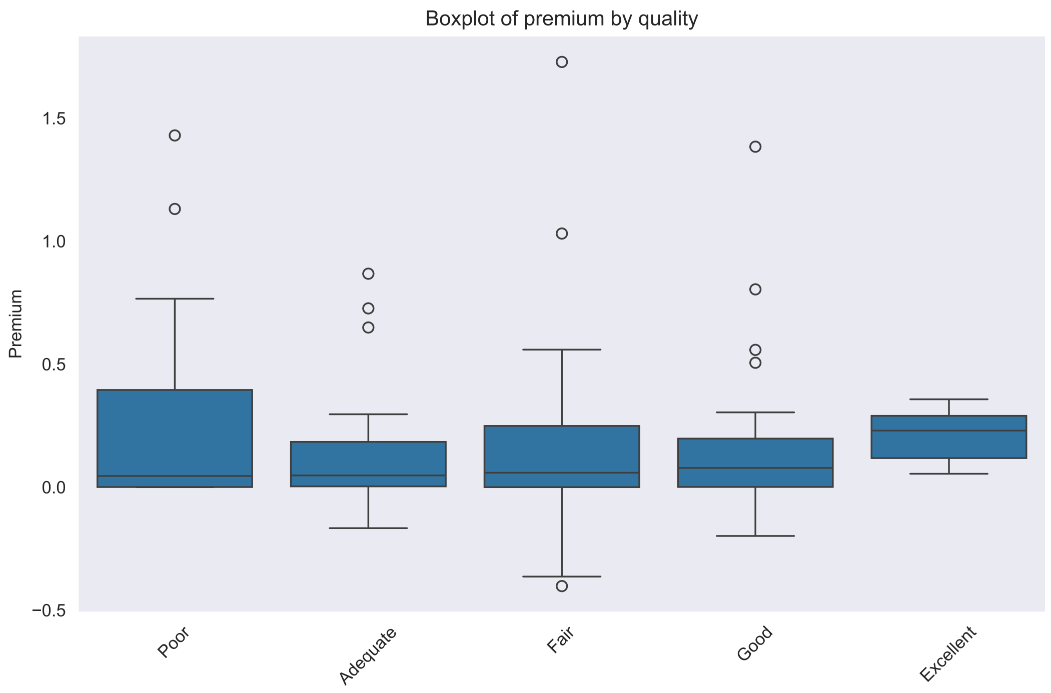
We see that the premium tends to be higher on average for properties of poor quality, and it is interesting to notice that despite being lower for other levels of quality, it tends to increase as the quality improves. Why could this be?

Another useful technique is the *boxplot*. This kind of plot uses rectangles to identify the first and third quantiles of the distribution, whiskers to show the value of 1.5 times the interquartile range (IQR, a measure of spread), and dots to represent extreme values (those beyond 1.5 times the IQR). The median of the distribution is a line in the box. Boxplots are implemented as `sns.boxplot()`:

```python
fig, ax = plt.subplots(figsize=(10,6))
sns.boxplot(data=auctions, x='quality', y='premium',
            order=['Poor', 'Adequate', 'Fair', 'Good', 'Excellent'], ax=ax)
ax.set_ylabel("Premium")
ax.set_title("Boxplot of premium by quality")
plt.xticks(rotation=45)
```

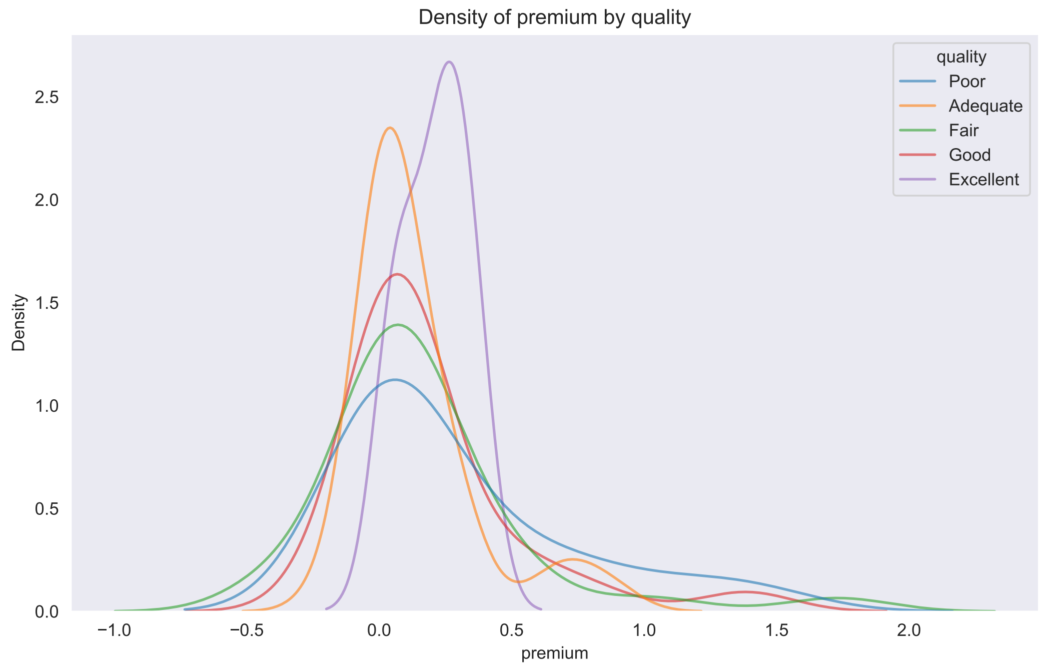
```
([0, 1, 2, 3, 4], [Text(0, 0, 'Poor'), Text(1, 0, 'Adequate'), Text(2, 0, 'Fair'), Text(3, 0, 'Good'), '
```

```python
plt.show()
```



We see from this plot that the distribution of the premium is more spread for properties of fair quality. The boxplot does obscure some of the detail of the underlying distribution of values. Ridge plots address this by plotting the density of the distribution instead. In Python we can use `sns.kdeplot()` with the `hue` parameter:

```python
fig, ax = plt.subplots(figsize=(10,6))
sns.kdeplot(data=auctions, x='premium', hue='quality',
            hue_order=['Poor', 'Adequate', 'Fair', 'Good', 'Excellent'],
            common_norm=False, alpha=0.6, ax=ax)
ax.set_title("Density of premium by quality")
plt.show()
```



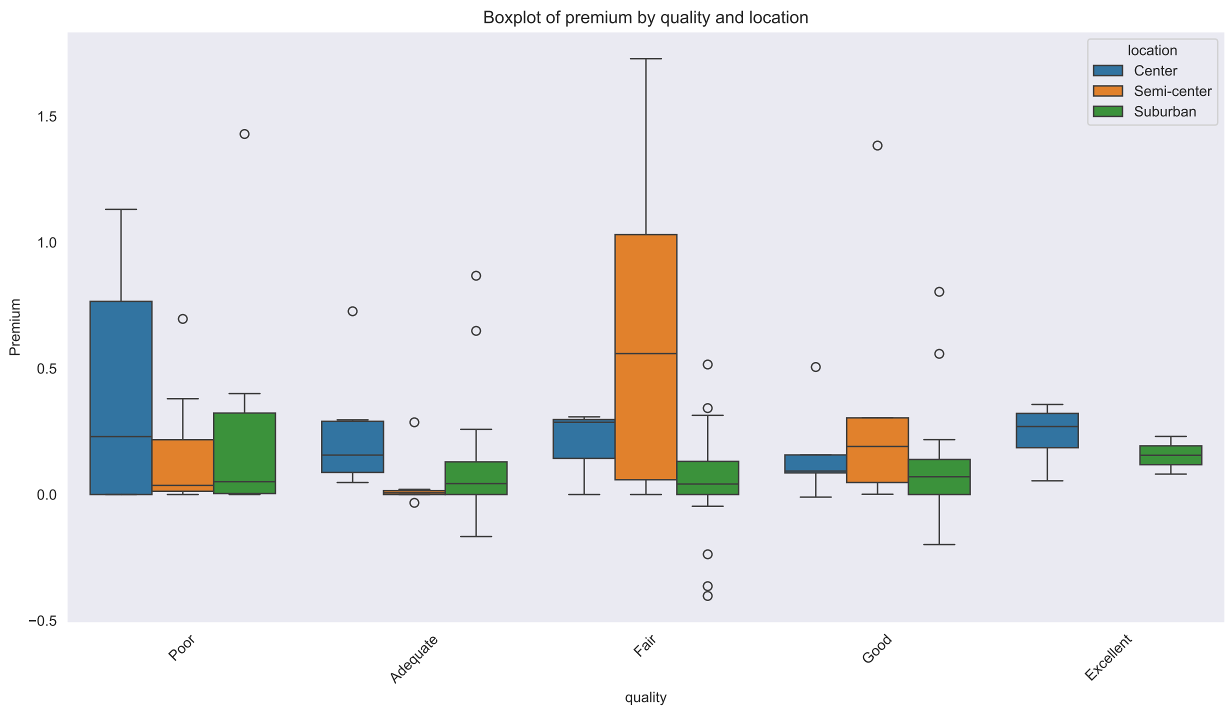
Multivariate description

Higher dimensional visualization can be created by encoding additional variables using available aesthetics. The following chunk of code recreates the boxplot of `quality` and `premium`, and further maps `location` to colour:

```
fig, ax = plt.subplots(figsize=(12,7))
sns.boxplot(data=auctions, x='quality', y='premium', hue='location',
            order=['Poor', 'Adequate', 'Fair', 'Good', 'Excellent'], ax=ax)
ax.set_ylabel("Premium")
ax.set_title("Boxplot of premium by quality and location")
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4], [Text(0, 0, 'Poor'), Text(1, 0, 'Adequate'), Text(2, 0, 'Fair'), Text(3, 0, 'Good'), 
```

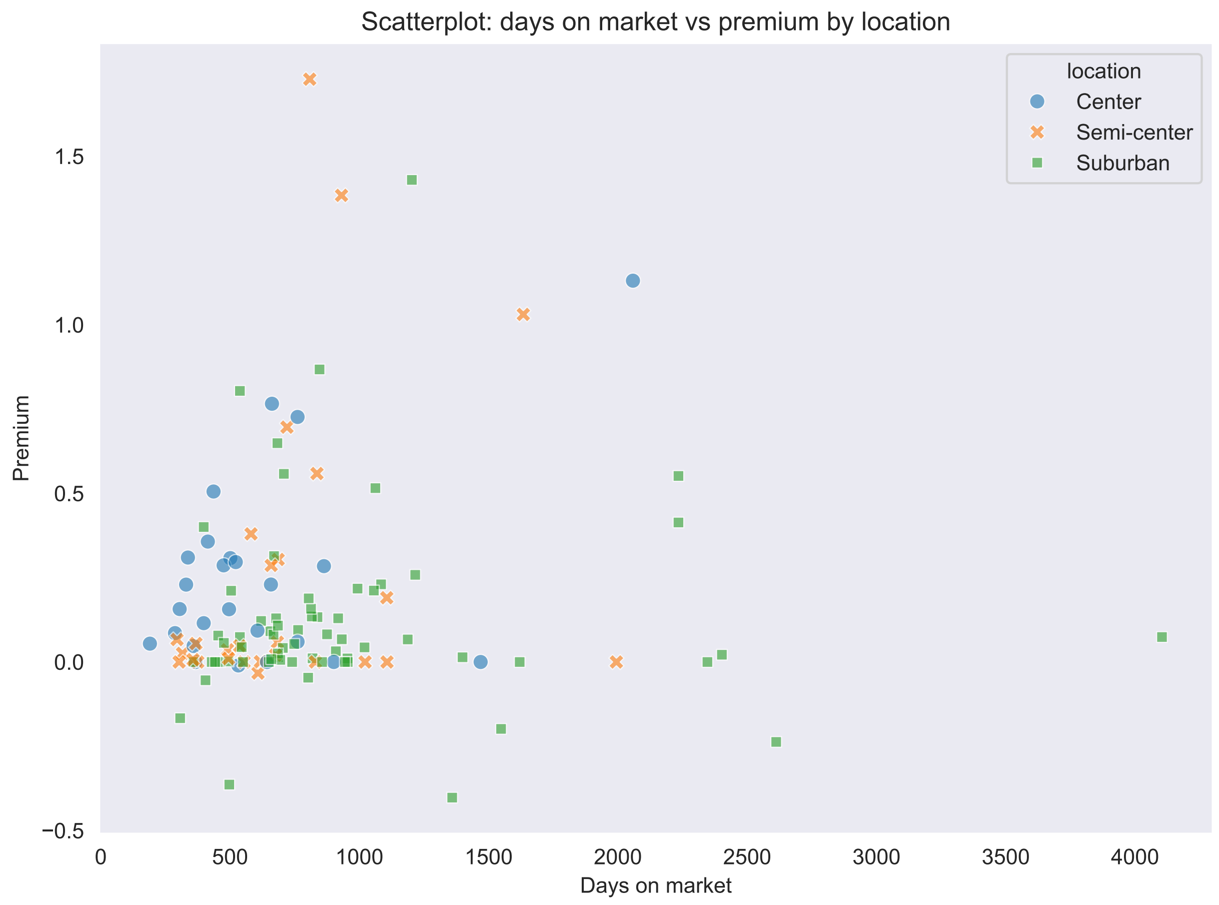
```
plt.tight_layout()
plt.show()
```



This plot reveals that the extreme values are more often for suburban properties than not.

The following example recreates the scatterplot of `days_on_market` and `premium` that we did before, but now adds `location` to plot, encoded to color and shape:

```
fig, ax = plt.subplots(figsize=(8,6))
sns.scatterplot(data=auctions, x='days_on_market', y='premium',
                hue='location', style='location', alpha=0.6, s=50, ax=ax)
ax.set_xlabel("Days on market")
ax.set_ylabel("Premium")
ax.set_title("Scatterplot: days on market vs premium by location")
plt.tight_layout()
plt.show()
```



As we saw above, the correlation between `days_on_market` and `premium` was rather low. But when we look under the surface of the original plot by making it multivariate, we see that the association between these two variables may be particularly poor for suburban properties, and perhaps not so poor for non-suburban properties. The correlation for the whole sample was 0.083. Let us calculate the correlation for suburban properties only:

```
corr_sub = auctions[auctions['location'] == 'Suburban'][['days_on_market', 'premium']].corr().iloc[0,1]
corr_non_sub = auctions[auctions['location'] != 'Suburban'][['days_on_market', 'premium']].corr().iloc[
corr_center = auctions[auctions['location'] == 'Center'][['days_on_market', 'premium']].corr().iloc[0,1

print(f"Correlation (Suburban):     {corr_sub:.3f}")
```

```
Correlation (Suburban):     0.015
```

```
print(f"Correlation (Non-Suburban): {corr_non_sub:.3f}")
```

```
Correlation (Non-Suburban): 0.345
```

```
print(f"Correlation (Center):       {corr_center:.3f}")
```

```
Correlation (Center):       0.484
```

This example illustrates how, in the words of Tukey, "[t]he greatest value of a picture is when it forces us to notice what we never expected to see."

Practice

1. Use the data set with auctions in distressed markets in Italy for this practice.
2. Create a statistical plot of variable `premium`. Use `axvline()` to plot the mean and the median of the variable.
3. Create a scatterplot of `discount` and `days_on_market`. Which of these variables would you encode on the x-axis and why?
4. Calculate the correlation between `discount` and `days_on_market`.
5. Recreate the scatterplot of `discount` and `days_on_market` but to make it a multivariate plot, use the aesthetics of color and shape to encode `quality`. What do you learn from this plot?
6. Recalculate the correlation between `discount` and `days_on_market` by `quality`. What do you learn from this?
7. Calculate the logarithm of `days_on_market` and create a statistical plot of this new variable. Add the mean and the median of the variable to the plot. Discuss this plot.
8. Repeat questions 5 and 6 using the log-transformation of `days_on_market` and discuss the results.