Session 4. Exploratory data analysis II: Visualization techniques

2026-02-12

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:

```
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("cntr_sp_basico")
data("cntr_sp_head")
data("cntr_sp_ipvs")
```

These data frames contain information about census tracts in the state of São Paulo. You can check the documentation in the usual way:

```
?cntr_sp_basico
```

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:

```
cntr_sp_basico = r.cntr_sp_basico
cntr_sp_head = r.cntr_sp_head
cntr_sp_ipvs = r.cntr_sp_ipvs
```

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

```
cntr = (cntr_sp_ipvs
.merge(cntr_sp_basico, left_on = 'COD_SETOR', right_on = 'code_tract', how = 'left')
.merge(cntr_sp_head, left_on = 'COD_SETOR', right_on = 'code_tract', 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?

```
cntr_sp_basico.loc[1:10, ["b_V001","b_V002"]]
```

```
    b_V001  b_V002
1    306.0   913.0
2    189.0   625.0
3    181.0   572.0
4    240.0   754.0
5    212.0   643.0
6    249.0   690.0
7    226.0   755.0
8    245.0   693.0
9    218.0   676.0
10   267.0   722.0
```

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:

```
cntr_sp_basico.loc[1:10, ["b_V001","b_V002"]].describe()
```

```
           b_V001      b_V002
count   10.000000   10.000000
mean   233.300000  704.300000
std     36.950568   92.851195
min    181.000000  572.000000
25%    213.500000  651.250000
50%    233.000000  691.500000
75%    248.000000  746.000000
max    306.000000  913.000000
```

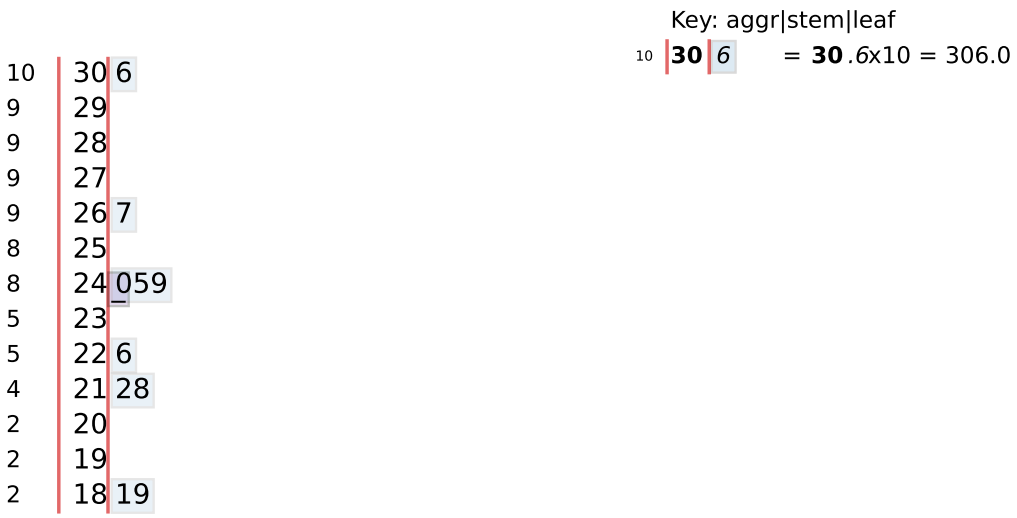
To make matters more difficult, this is only a small part of the full table (only ten rows and two columns). The task of identifying patterns becomes increasingly complicated as the number of observations and the number of variables grow.

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:

```
b_V001_10 = cntr_sp_basico.loc[1:10, "b_V001"]
b_V001_10
```

```
1     306.0
2     189.0
3     181.0
4     240.0
5     212.0
6     249.0
7     226.0
8     245.0
9     218.0
10    267.0
Name: b_V001, dtype: float64
```

```
stemgraphic.stem_graphic(b_V001_10, scale = 10)
plt.show()
```



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
5	7
6	34899
7	56
8	0
9	1

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:

```
cntr_sp_head.select_dtypes(include = ['number']).corr()
```

```
          h_V001    h_V002    h_V003    h_V004    h_V005    h_V006    h_V007  \
h_V001  1.000000  0.579719  0.394422  0.080053 -0.128780 -0.231540 -0.198893
h_V002  0.579719  1.000000  0.705431  0.351147  0.015881 -0.266842 -0.307038
h_V003  0.394422  0.705431  1.000000  0.674936  0.236218 -0.190847 -0.296864
```

```
h_V004  0.080053  0.351147  0.674936  1.000000  0.717134  0.225293 -0.038210
h_V005 -0.128780  0.015881  0.236218  0.717134  1.000000  0.686859  0.357584
h_V006 -0.231540 -0.266842 -0.190847  0.225293  0.686859  1.000000  0.814199
h_V007 -0.198893 -0.307038 -0.296864 -0.038210  0.357584  0.814199  1.000000
h_V008 -0.168446 -0.287756 -0.295657 -0.118503  0.193887  0.658285  0.830068
h_V009 -0.124726 -0.242110 -0.258339 -0.157216  0.050277  0.426794  0.646524
h_V010  0.317982  0.280266  0.332236  0.129315 -0.008825 -0.093251 -0.092251

          h_V008    h_V009    h_V010
h_V001 -0.168446 -0.124726  0.317982
h_V002 -0.287756 -0.242110  0.280266
h_V003 -0.295657 -0.258339  0.332236
h_V004 -0.118503 -0.157216  0.129315
h_V005  0.193887  0.050277 -0.008825
h_V006  0.658285  0.426794 -0.093251
h_V007  0.830068  0.646524 -0.092251
h_V008  1.000000  0.821656 -0.082637
h_V009  0.821656  1.000000 -0.061355
h_V010 -0.082637 -0.061355  1.000000
```

Now compare to:

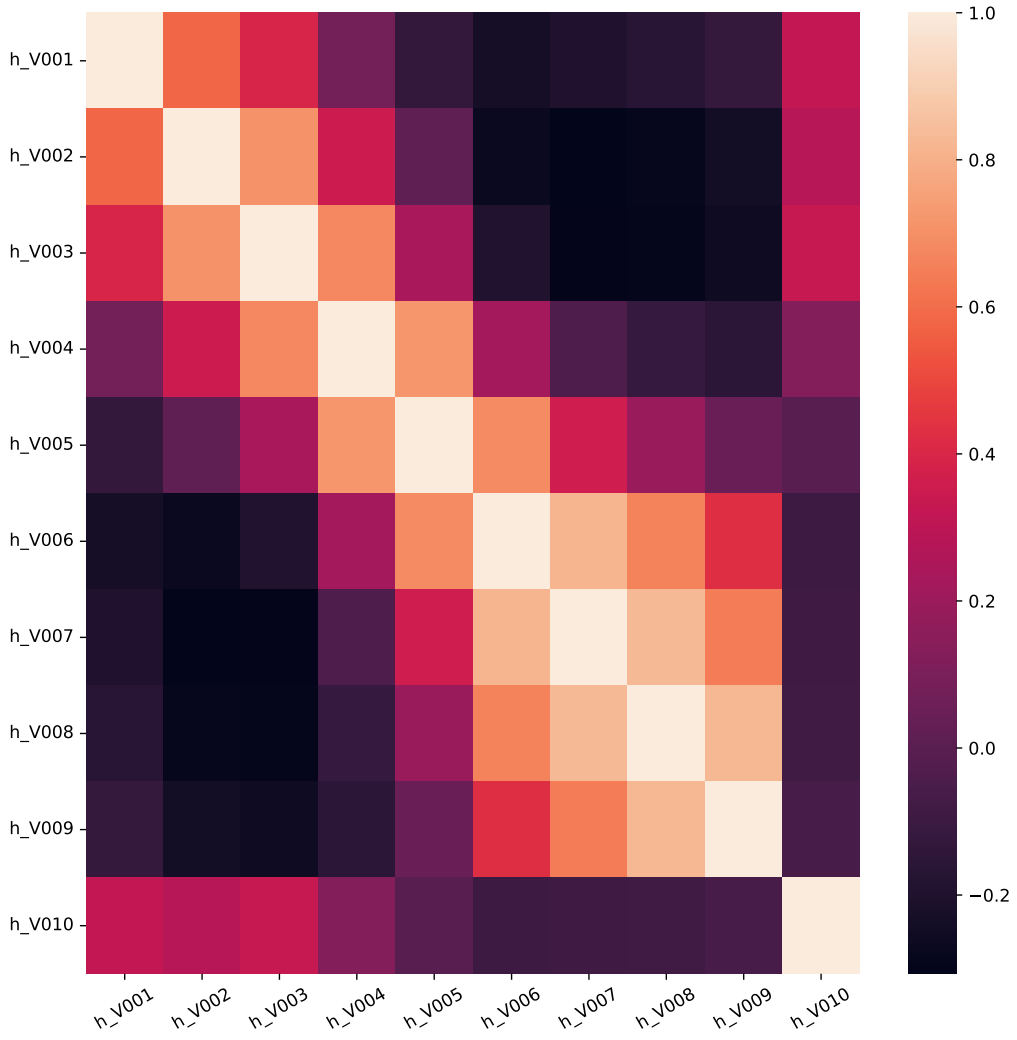
```
cntr_sp_head_corr = cntr_sp_head.select_dtypes(include = ['number']).corr()

fig, ax = plt.subplots(figsize = (10,10))
sns.heatmap(cntr_sp_head_corr)
plt.xticks(rotation = 30)
```

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]), [Text(0.5, 0, 'h_V001'), Text(1.5, 0, 'h_V0
```

```
plt.yticks(rotation = 0, ha = 'right')
```

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]), [Text(0, 0.5, 'h_V001'), Text(0, 1.5, 'h_V0
```

```
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.

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 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:

```
mask = np.triu(np.ones_like(cntr_sp_head_corr, dtype = bool), k = 1)  # upper triangle excluding diagona
corr_masked = cntr_sp_head_corr.mask(mask)
corr_long = corr_masked.stack().reset_index()
corr_long.columns = ['term_1', 'term_2', 'r']
corr_long = corr_long.dropna()
term1_order = ["h_V001", "h_V002", "h_V003", "h_V007"]
term2_order = ["h_V001", "h_V002", "h_V003"]
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()
```

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()
```

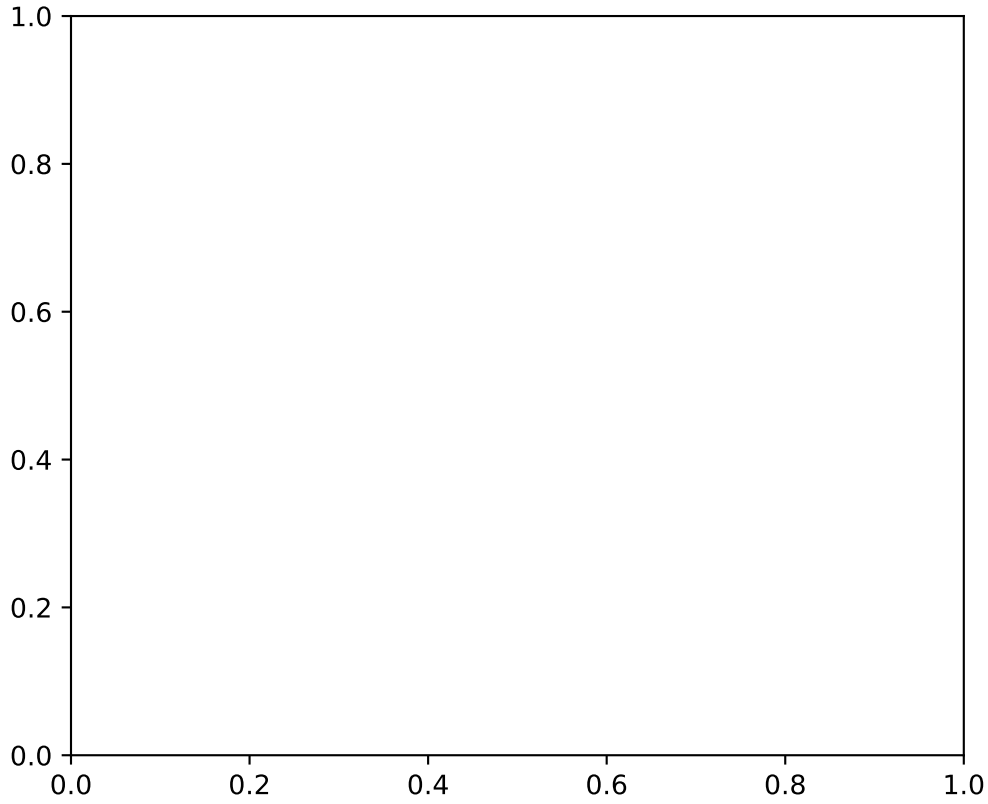
```
   term_1  term_2         r
0  h_V001  h_V001  1.000000
1  h_V002  h_V001  0.579719
2  h_V002  h_V002  1.000000
3  h_V003  h_V001  0.394422
4  h_V003  h_V002  0.705431
```

```
my_r_matrix.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9 entries, 0 to 23
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   term_1  9 non-null      category
 1   term_2  9 non-null      category
 2   r       9 non-null      float64
dtypes: category(2), float64(1)
memory usage: 498.0 bytes
```

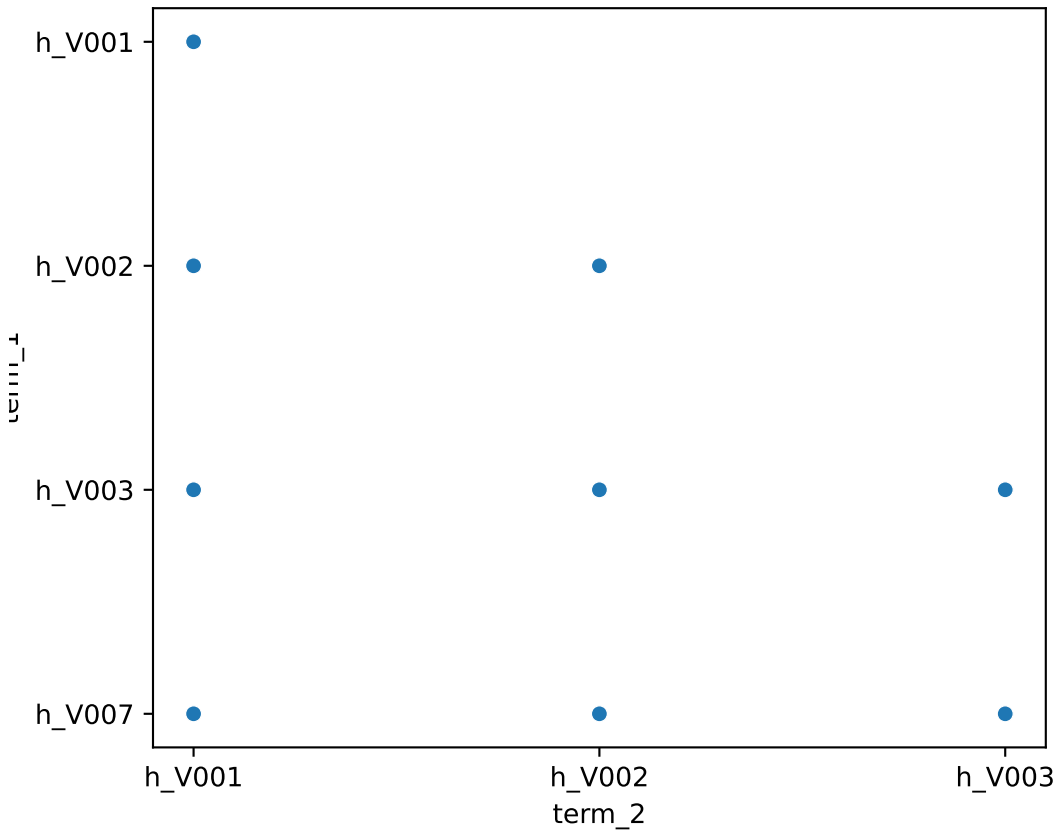
In `matplotlib`/`seaborn` we first create a figure and axes. This is our "blank canvas".

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



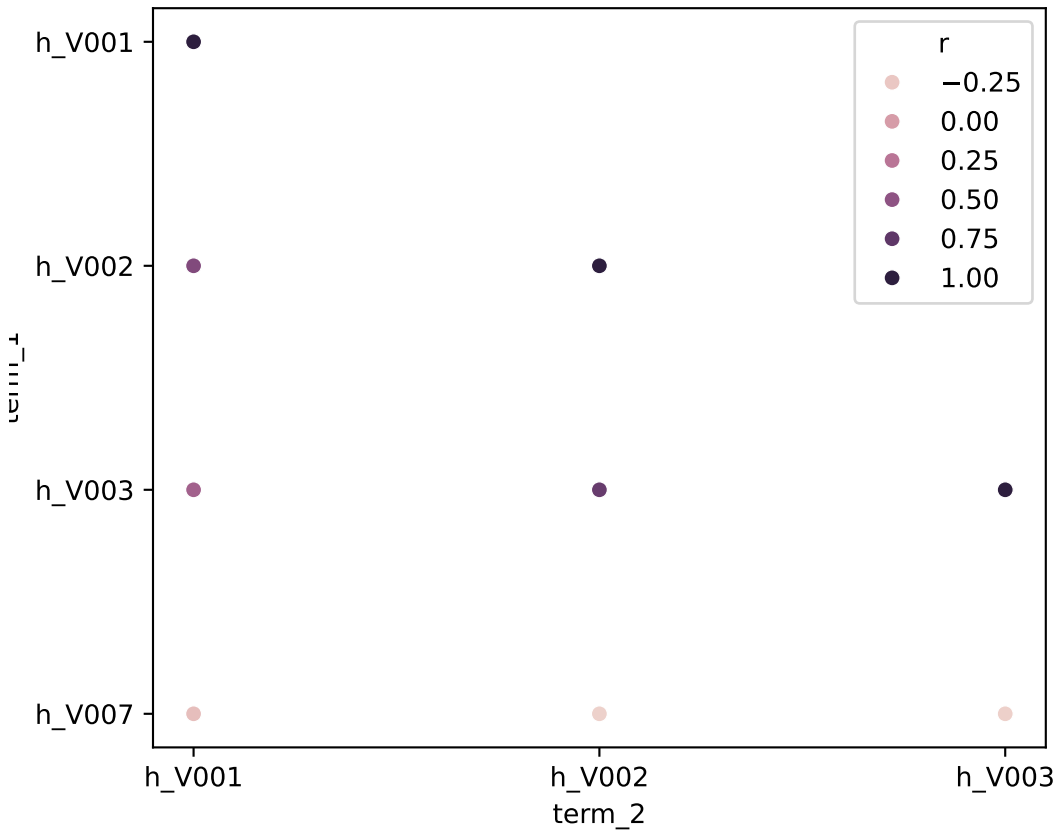
A blank canvas is not yet a statistical plot. Although we already created the figures and axes, we have not yet stated *what* we wish to plot. In this case, we wish to represent correlations as points, so we use `sns.scatterplot()`. Now we need to map variables in our table to the plot. Points need to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



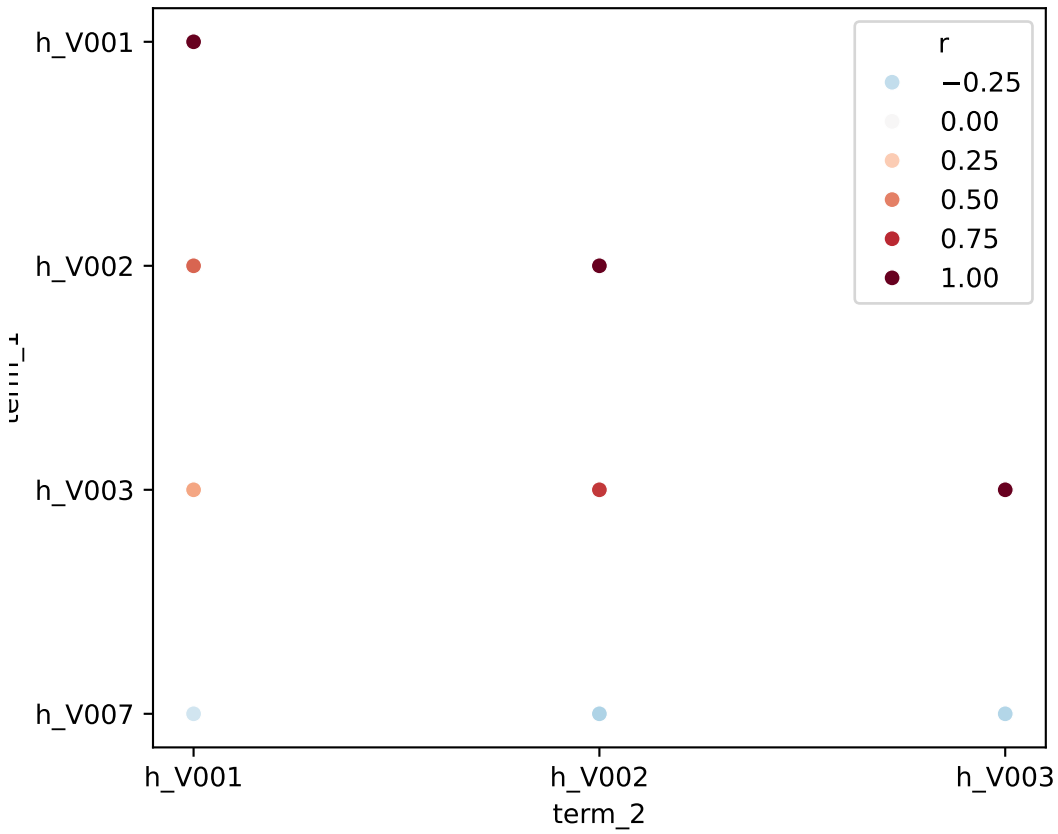
Now we have points in the figure. But so far we have encoded (i.e., mapped) only *position* (for which we used two variables, `term_1` and `term_2`). Next, we also want to represent the correlation, something that we can do by encoding variable `r` using color (`hue`):

```
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()
```



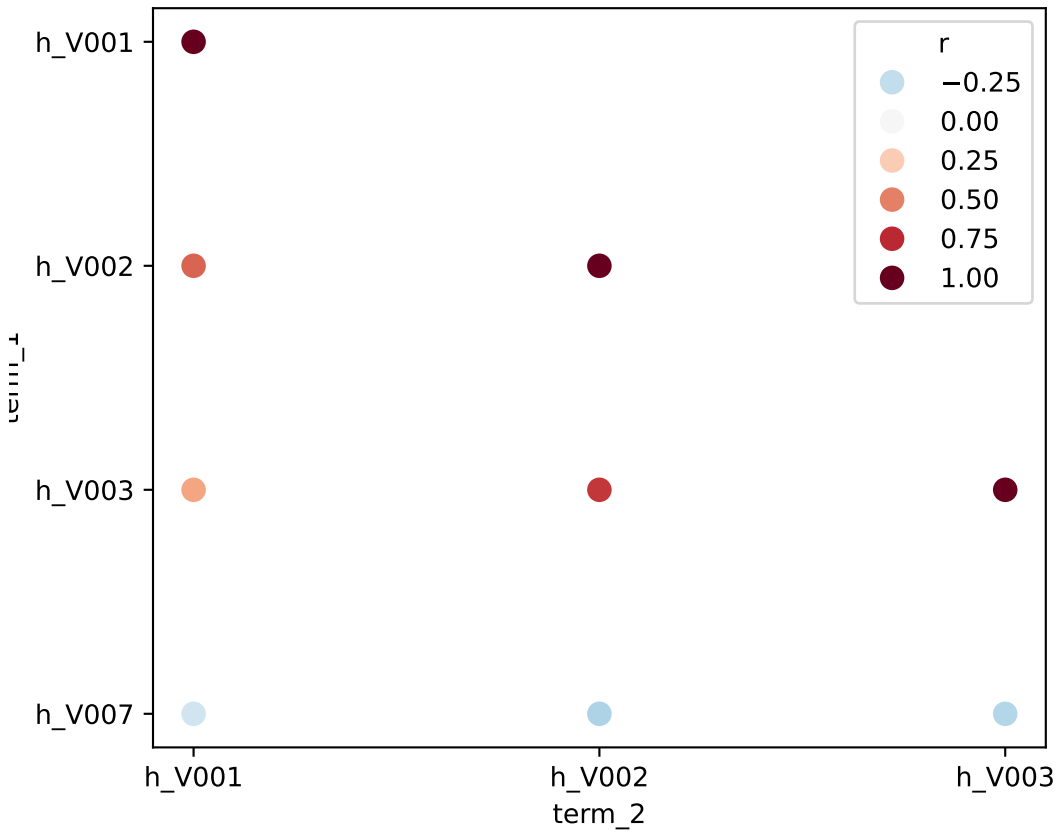
The points now map the correlation to a color. The default colors, though, are not very effective: correlation ranges between -1 and 1, but a sequential color scale does not correctly convey the change in valence in correlation (positive and negative). We can replace the default palette with a diverging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



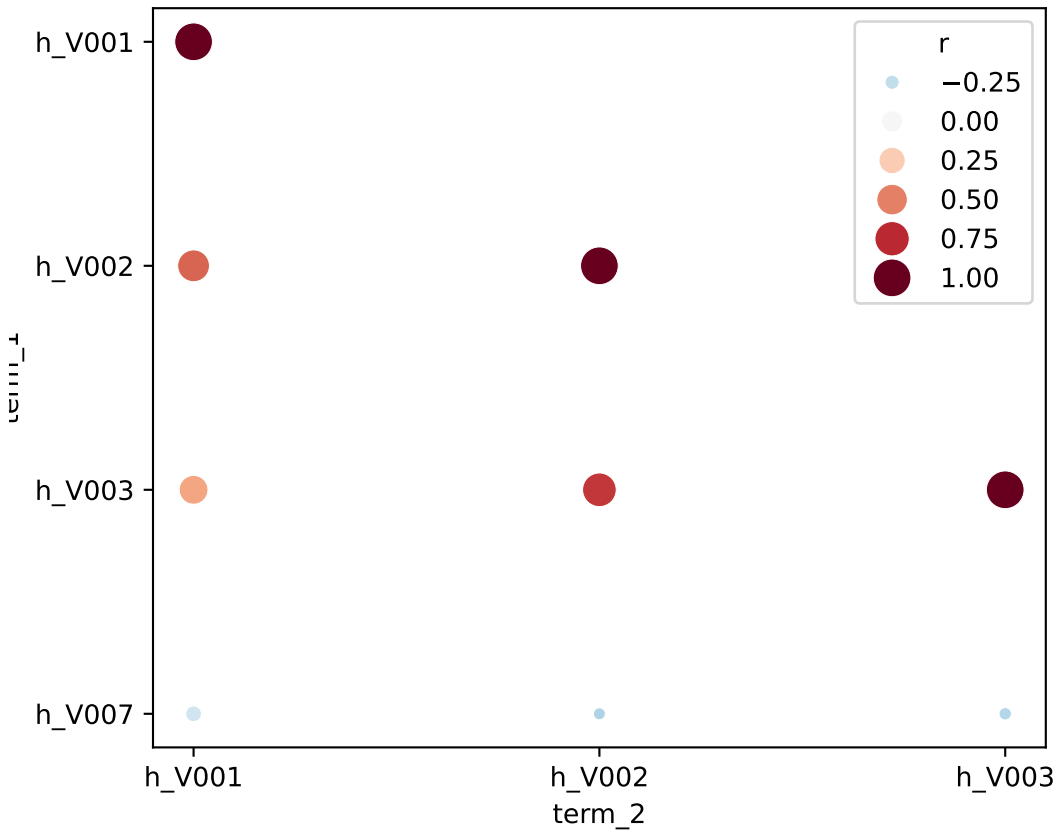
The colors let us more easily perceive the *magnitude* as well as the *direction* of the correlation. That said, the points are too small. We can control for the size of the cycles by changing the parameter `s`:

```
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()
```



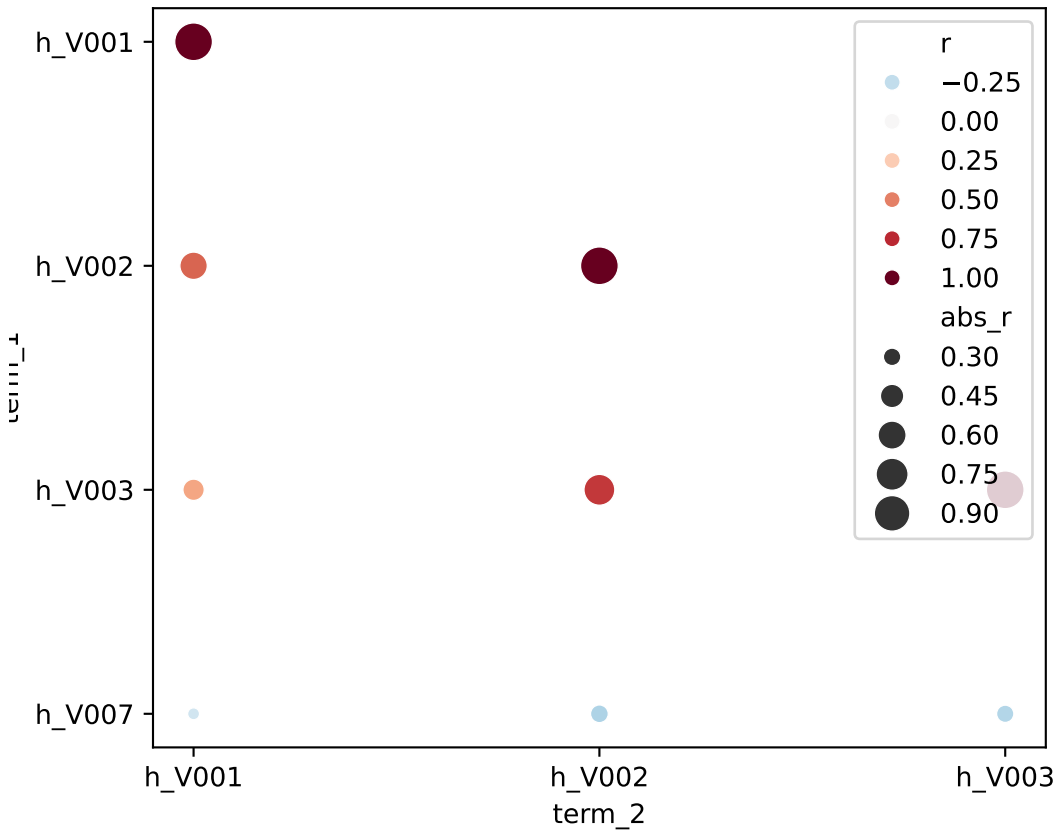
It is easier to see the colors. But notice how the `size` is a constant and does not map to any variable. This is why the points are all the same size. What if we mapped size to the correlation? in the chunk we map size `s` to the absolute value (`r`) of the correlation, so that strength is shown by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

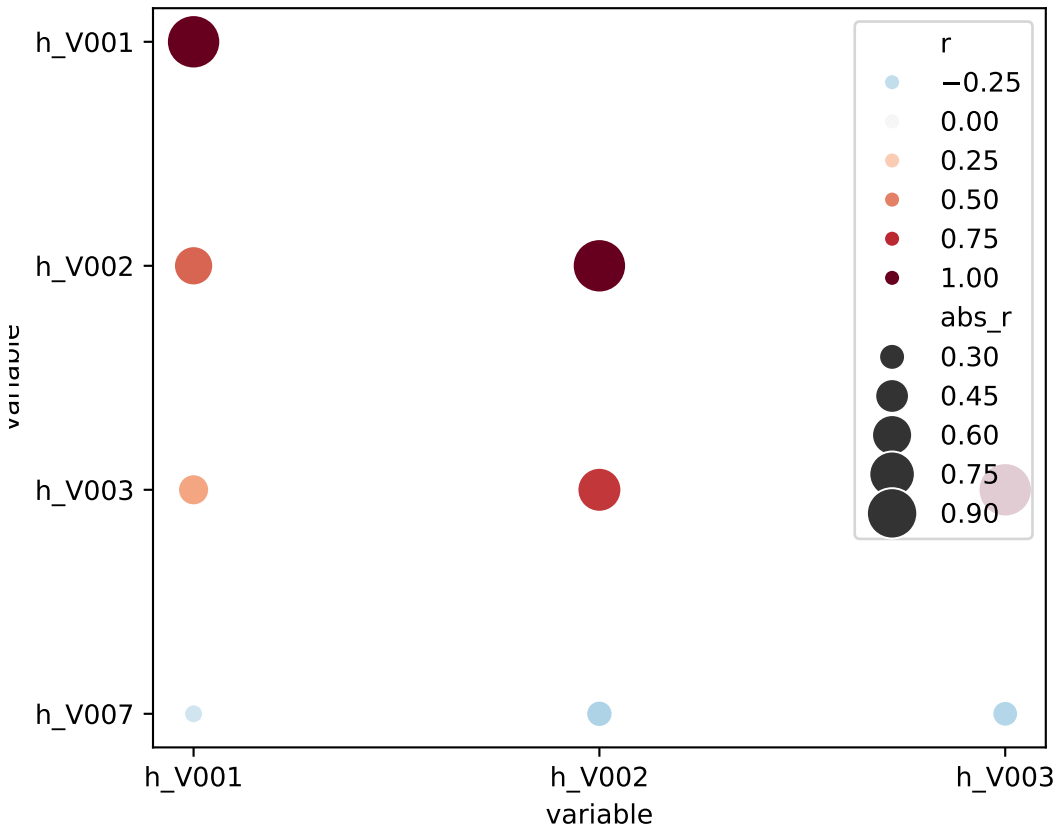
```
my_r_matrix["abs_r"] = my_r_matrix["r"].abs()

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.

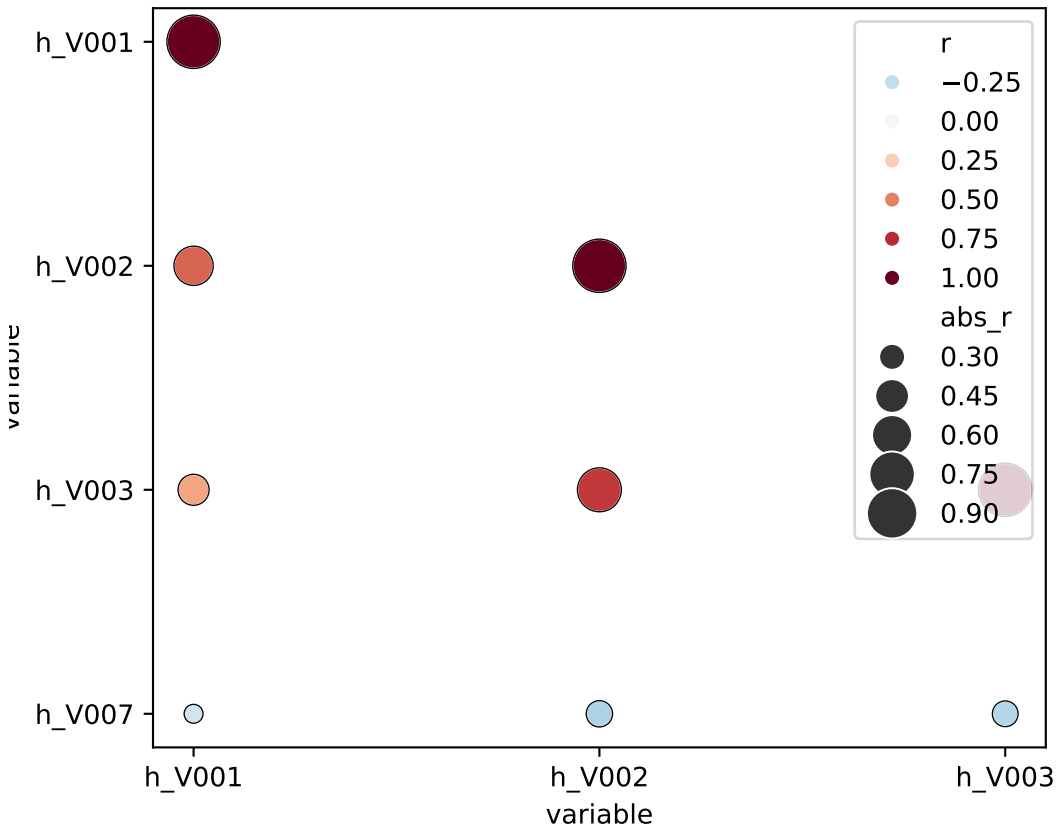
We can also adorn the phrase that builds our plot. For example, we can set the legend title for each axis:

```
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()
```



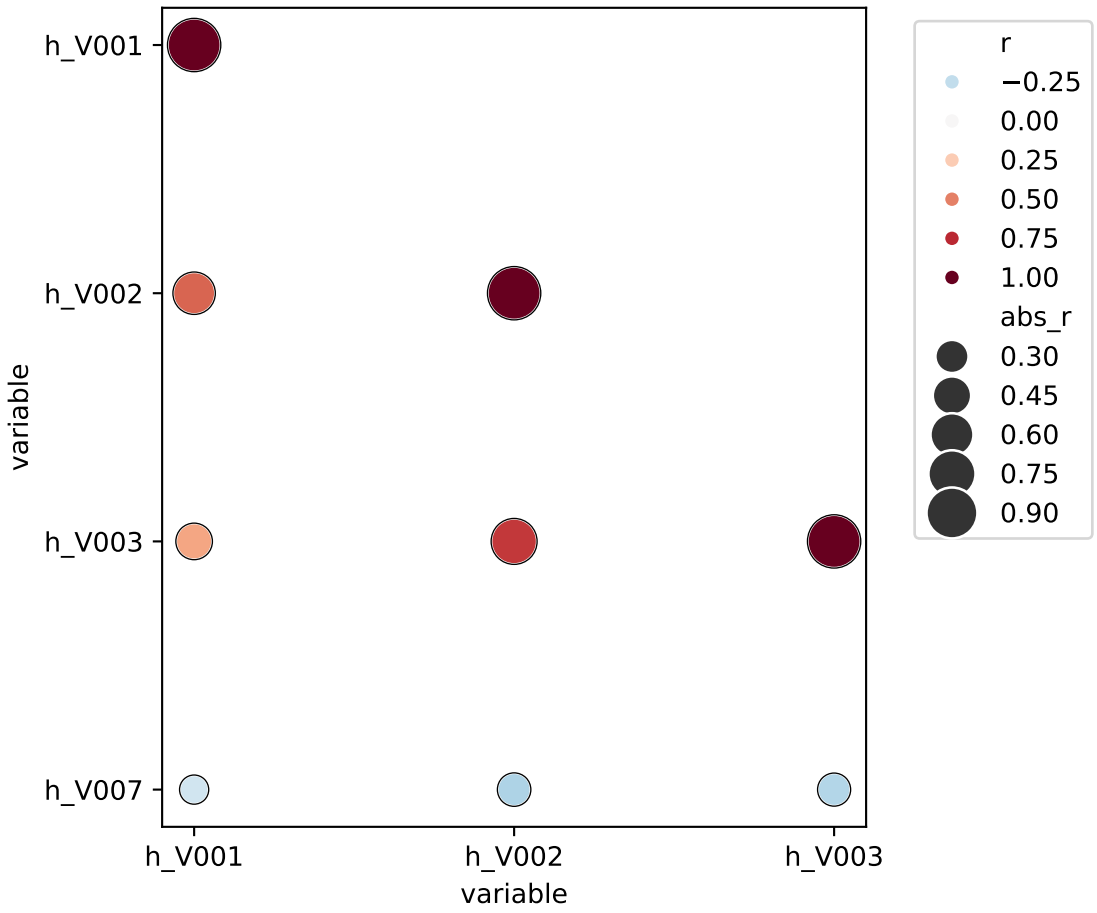
To make the symbols easier to see, we can layer another set of points using a different symbol (a circle without fill) that maps size to absolute correlation but uses only the default color (black) as a constant:

```
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:

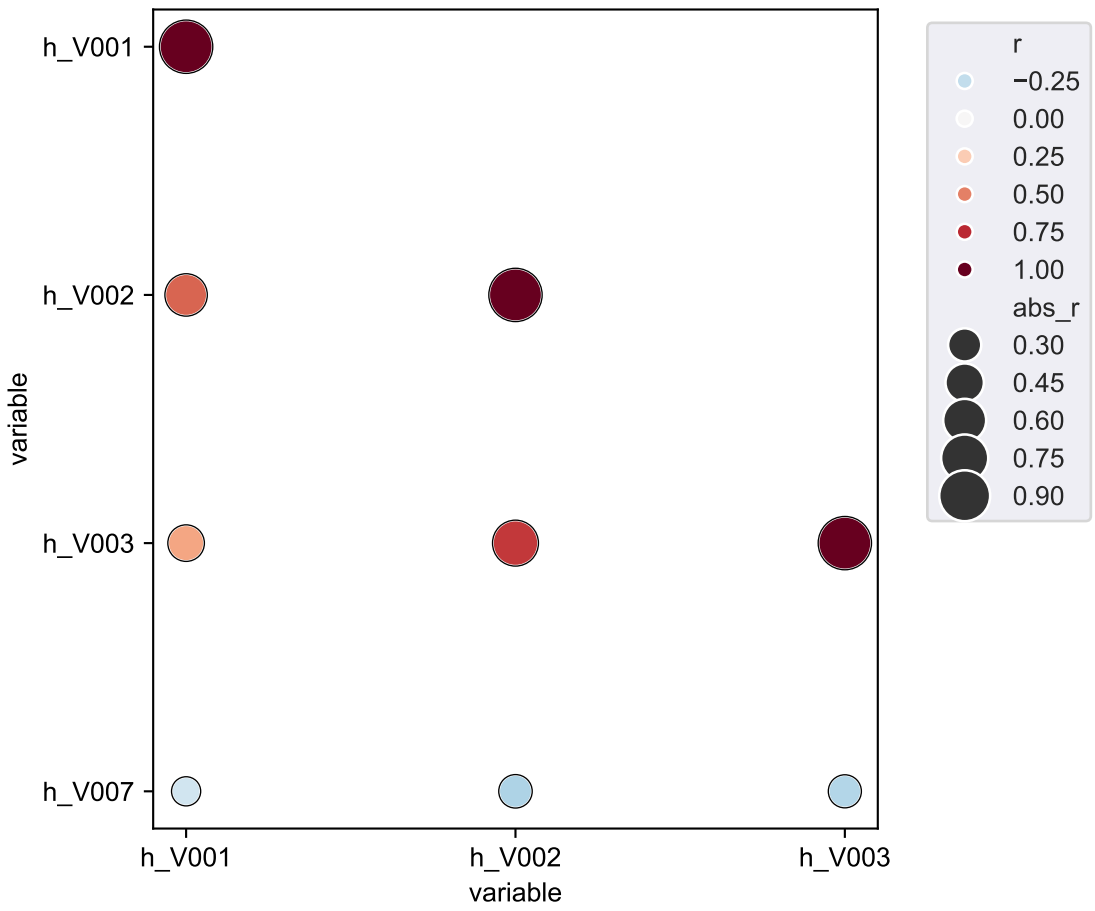
```
fig, ax = plt.subplots(figsize=(6,5))

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()
```



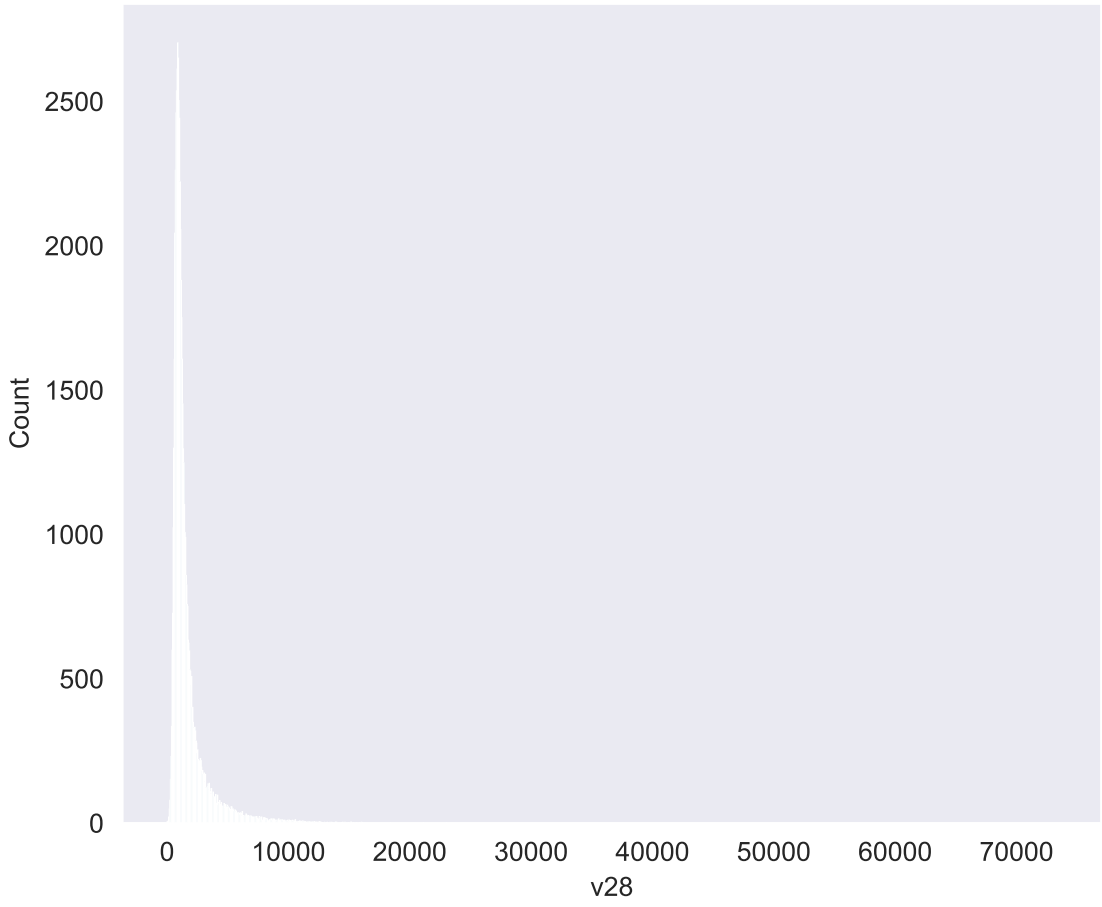
`sns.scatterplot` is one of the possible options to create plots in Python.

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 `v28` (average income of the person responsible for the household), we have:

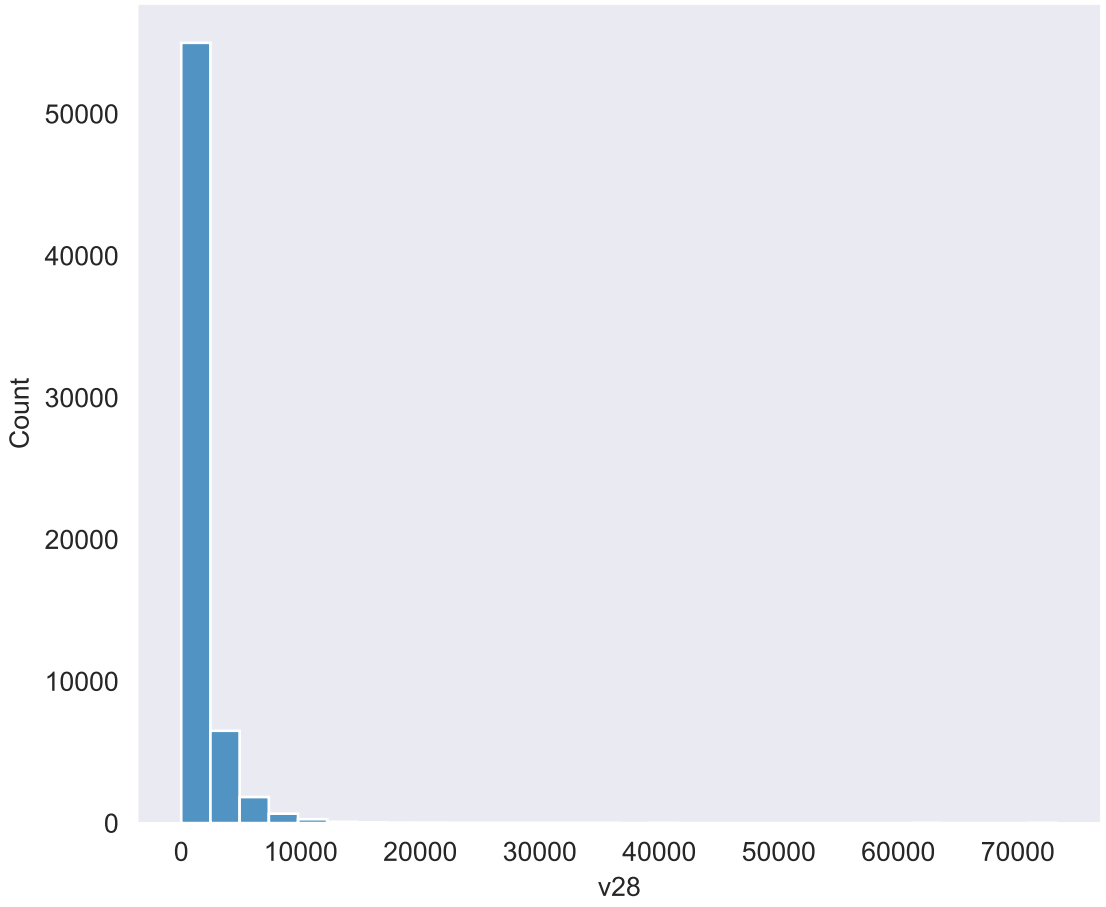
```
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = cntr, x = 'v28', ax = ax)
plt.tight_layout()
plt.show()
```



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 `v28`), 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).

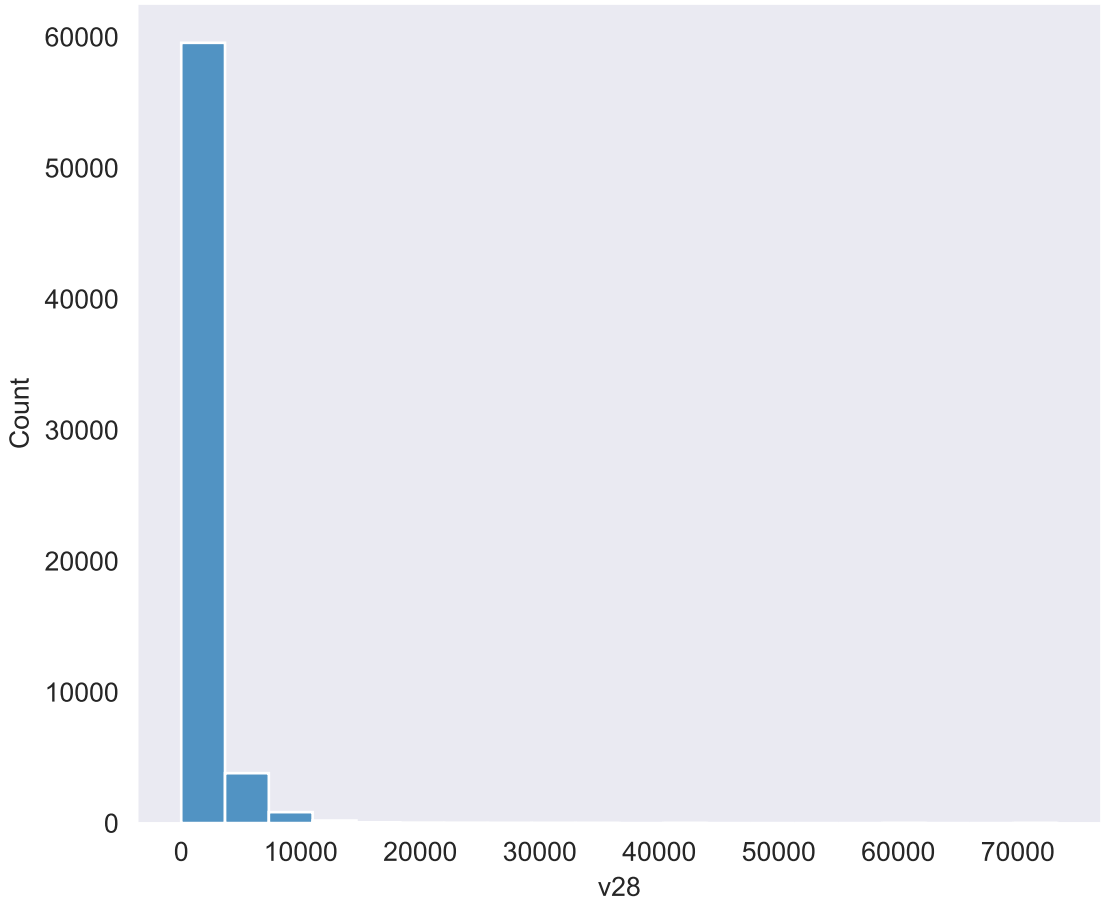
For instance, for 30 bins, we set the parameter `bins` equal to 30:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = cntr, x = 'v28',  bins = 30, ax = ax)
plt.tight_layout()
plt.show()
```



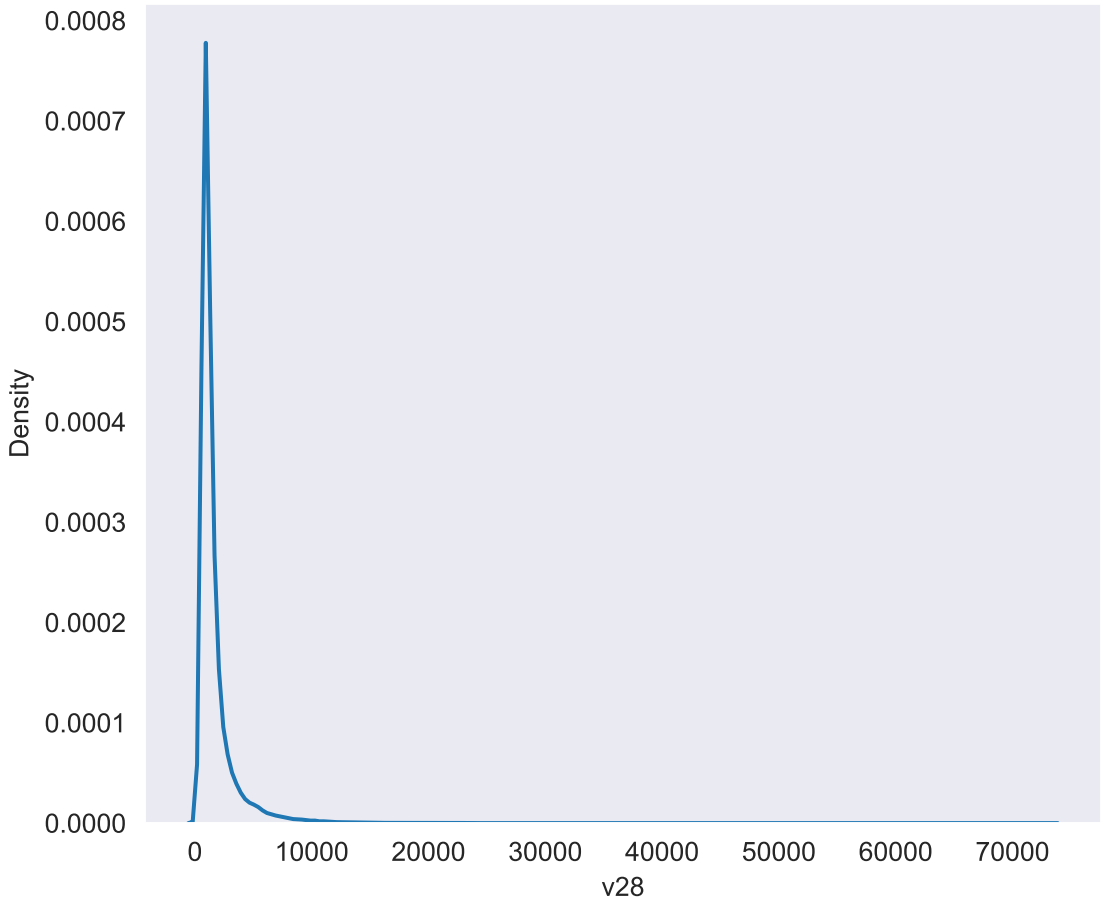
For 20 bins:

```
fig, ax = plt.subplots(figsize = (6,5))
sns.histplot(data = cntr, x = 'v28',  bins = 20, ax = ax)
plt.tight_layout()
plt.show()
```



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

```
fig, ax = plt.subplots(figsize = (6,5))
sns.kdeplot(data = cntr, x = 'v28', ax = ax)
plt.tight_layout()
plt.show()
```



These plots suggest that the average income of the head of household tend to be less than R$ 2,500, but in some relatively rare cases they can be higher than this value. The mean and median of this variable are:

```
mean_v28 = cntr['v28'].mean()
median_v28 = cntr['v28'].median()

mean_v28
```

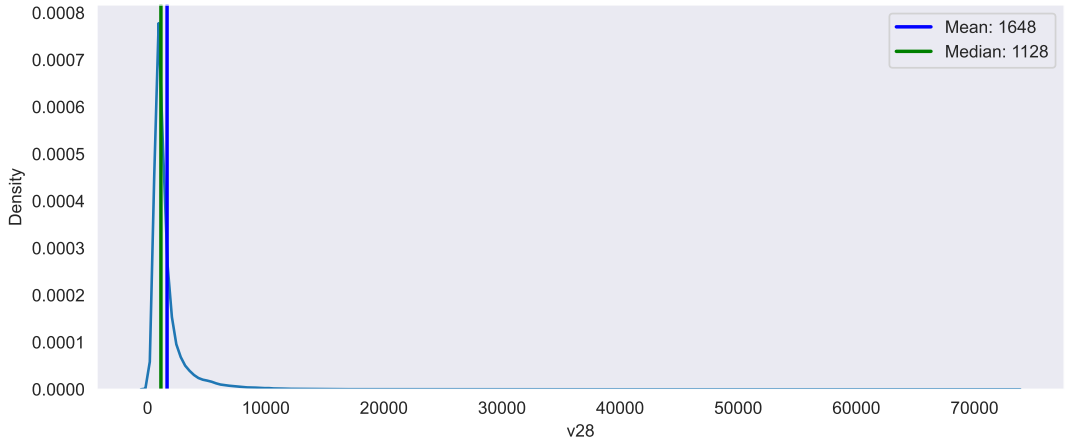
```
1648.450535366942
```

```
median_v28
```

```
1128.031915
```

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):

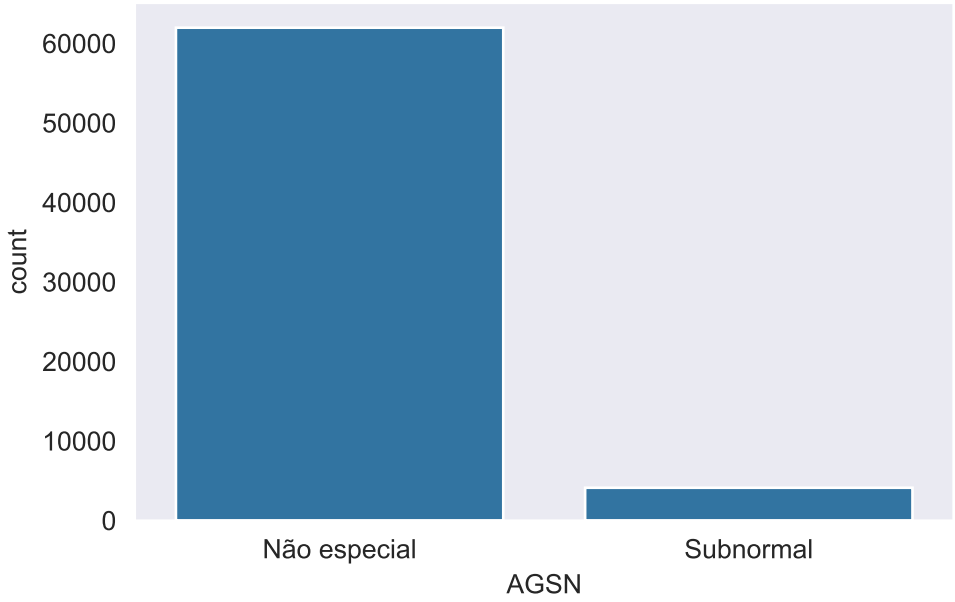
```
fig, ax = plt.subplots(figsize = (10,4))
sns.kdeplot(data = cntr, x = 'v28', ax = ax)
ax.axvline(mean_v28, color = 'blue', linewidth = 2, label = f'Mean: {mean_v28:.0f}')
ax.axvline(median_v28, color = 'green', linewidth = 2, label = f'Median: {median_v28:.0f}')
ax.legend()
plt.show()
```



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 = cntr, x = 'AGSN', ax = ax)
plt.show()
```

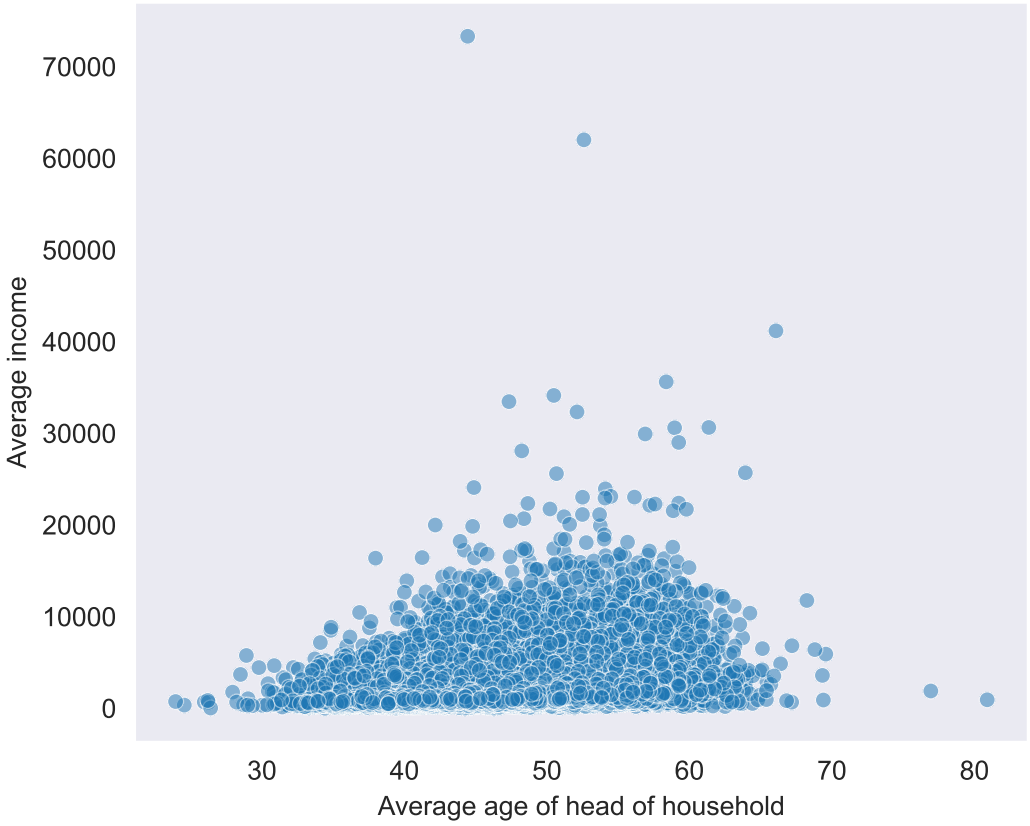


Bivariate description

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 = cntr, x = 'v27', y = 'v28', alpha = 0.5, ax = ax)
ax.set_xlabel("Average age of head of household")
ax.set_ylabel("Average income")
plt.show()
```



We can visualize the correlation between the two variables, `v27` and `v28`, by doing:

```
cntr[['v27', 'v28']].corr()
```

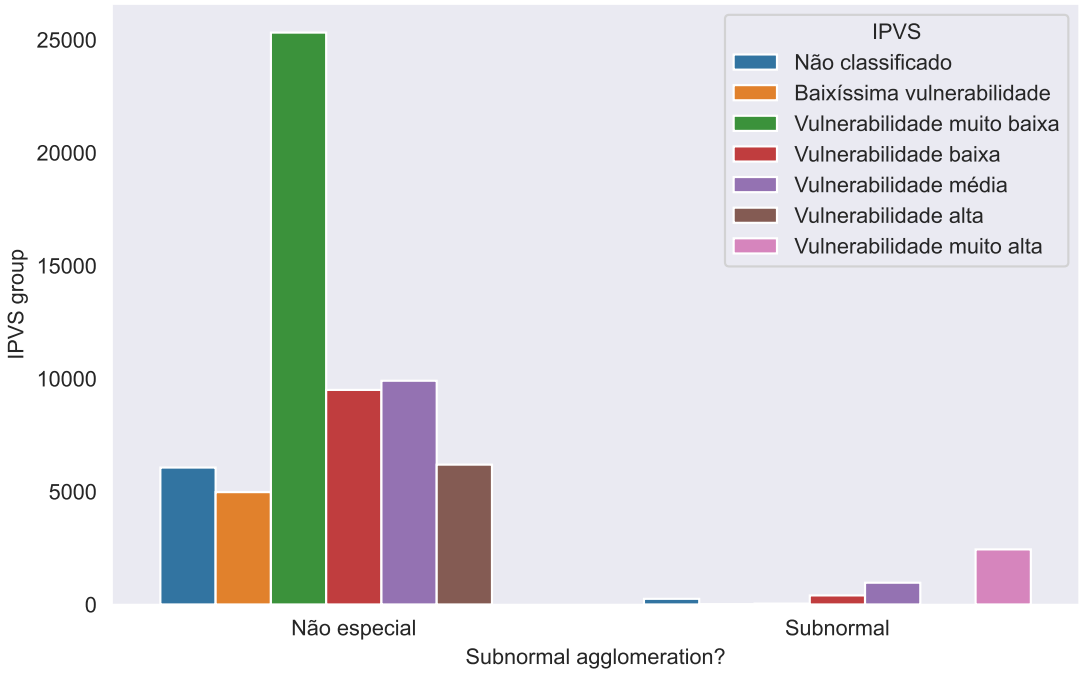
```
          v27       v28
v27  1.000000  0.298843
v28  0.298843  1.000000
```

How does this correlation explain the scatter plot shown above?

Two categorical variables

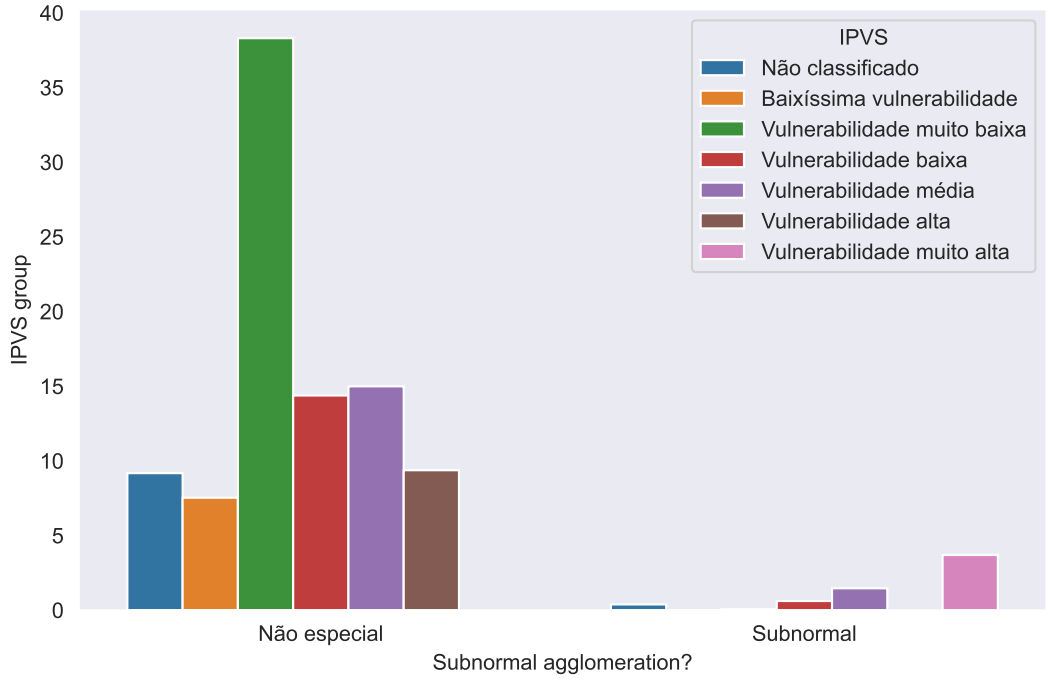
Two categorical variables can be explored by means of count plots:

```
fig, ax = plt.subplots(figsize = (8,5))
sns.countplot(x = "AGSN", hue =  "IPVS", data = cntr)
ax.set_xlabel("Subnormal agglomeration?")
ax.set_ylabel("IPVS group")
plt.show()
```



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

```
fig, ax = plt.subplots(figsize = (8,5))
sns.countplot(x = "AGSN", hue =  "IPVS", data = cntr, stat = "percent")
ax.set_xlabel("Subnormal agglomeration?")
ax.set_ylabel("IPVS group")
plt.show()
```



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

```
pd.crosstab(cntr['AGSN'], cntr['IPVS'])
```

```
IPVS          Não classificado  Baixíssima vulnerabilidade  \
AGSN
Não especial              6067                        4979
Subnormal                  256                           1

IPVS          Vulnerabilidade muito baixa  Vulnerabilidade baixa  \
AGSN
Não especial                        25321                   9504
Subnormal                              44                    406

IPVS          Vulnerabilidade média  Vulnerabilidade alta  \
AGSN
Não especial                   9910                  6196
Subnormal                       969                     0

IPVS          Vulnerabilidade muito alta
AGSN
Não especial                           0
Subnormal                           2443
```

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 `v28` by quality of `IPVS`) and then use `sns.barplot()` to visualize these two variables:

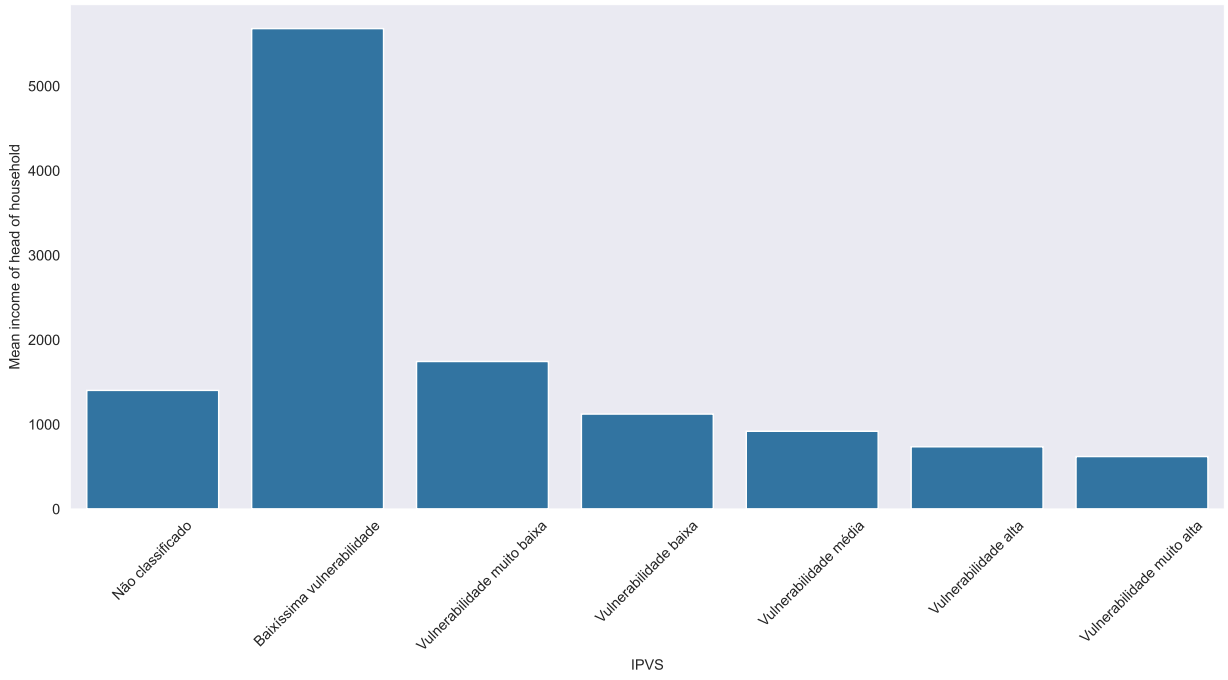
```
mean_by_ipvs = cntr.groupby('IPVS')['v28'].mean().reset_index()
```

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

```
fig, ax = plt.subplots(figsize = (12,10))
sns.barplot(data = mean_by_ipvs, x = 'IPVS', y = 'v28', ax = ax)
ax.set_ylabel("Mean income of head of household")
plt.xticks(rotation = 45)
```

```
([0, 1, 2, 3, 4, 5, 6], [Text(0, 0, 'Não classificado'), Text(1, 0, 'Baixíssima vulnerabilidade'), Text
```

```
plt.tight_layout()
plt.subplots_adjust(bottom = 0.5) # To adjust the size of the plot for the x-axis labels
plt.show()
```



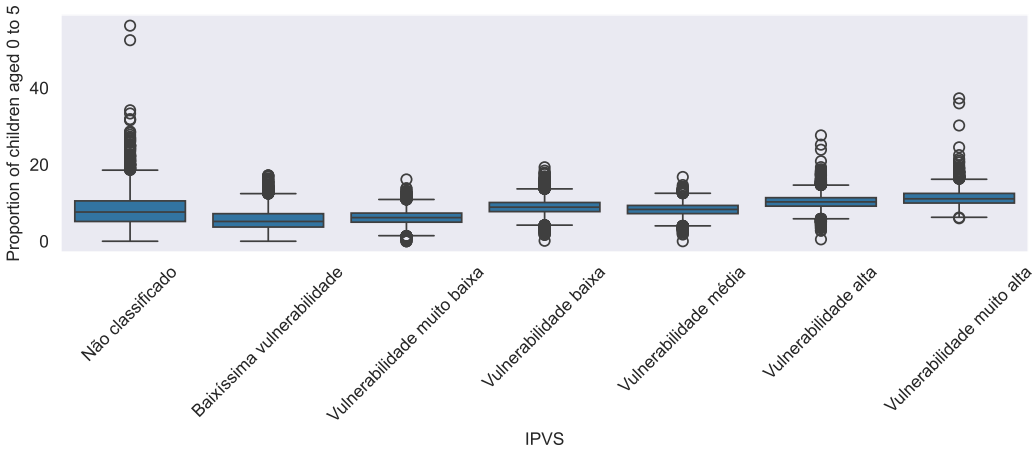
We see that the average salary of the head of household tends to be higher in the less vulnerable census tracts.

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()`:

```
fig, ax = plt.subplots(figsize = (10,5))
sns.boxplot(data = cntr, x = 'IPVS', y = 'v16', ax = ax)
ax.set_ylabel("Proportion of children aged 0 to 5")
plt.xticks(rotation = 45)
```

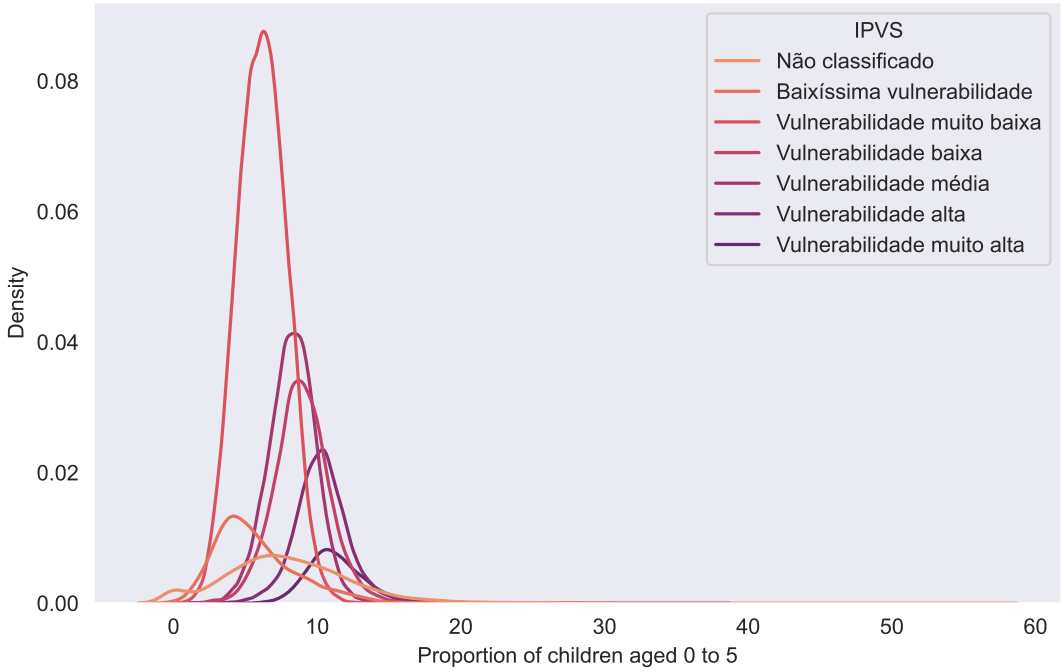
```
([0, 1, 2, 3, 4, 5, 6], [Text(0, 0, 'Não classificado'), Text(1, 0, 'Baixíssima vulnerabilidade'), Text
```

```
plt.subplots_adjust(bottom = 0.5) # To adjust the size of the plot for the x-axis labels
plt.show()
```



We see from this plot that the distribution of the proportion of children aged 0-5 in the population is more spread for census tracts not classified or classified with 'Baixissima vulnerabilidade' (very, very low vulnerability). 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 by using `sns.kdeplot()`:

```
fig, ax = plt.subplots(figsize = (8,5))
sns.kdeplot(data = cntr, x = 'v16', hue = 'IPVS',
            palette = 'flare', ax = ax)
ax.set_xlabel("Proportion of children aged 0 to 5")
plt.show()
```



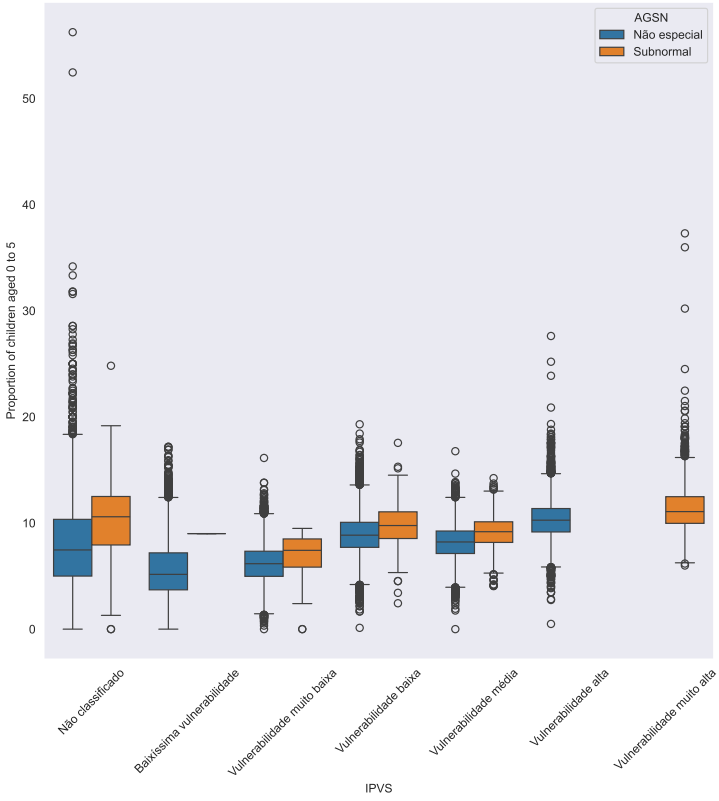
Multivariate description

Higher dimensional visualization can be created by encoding additional variables using available aesthetics. The following chunk of code recreates the boxplot of `IPVS` and `v16`, and further maps `AGSN` to colour:

```
fig, ax = plt.subplots(figsize = (10,20))
sns.boxplot(data = cntr, x = 'IPVS', y = 'v16', hue = 'AGSN', ax = ax)
ax.set_ylabel("Proportion of children aged 0 to 5")
plt.xticks(rotation = 45)
```

```
([0, 1, 2, 3, 4, 5, 6], [Text(0, 0, 'Não classificado'), Text(1, 0, 'Baixíssima vulnerabilidade'), Text
```

```
plt.subplots_adjust(bottom = 0.5)
plt.show()
```


AGSN
Não especial
Subnormal
Proportion of children aged 0 to 5
0
10
20
30
40
50
Não classificado
Baixíssima vulnerabilidade
Vulnerabilidade muito baixa
Vulnerabilidade baixa
Vulnerabilidade média
Vulnerabilidade alta
Vulnerabilidade muito alta
IPVS

This graph shows that for all levels of vulnerability, census tracts classified as “subnormal” tend to have a higher proportion of children aged 0 to 5 in the population.

The following example recreates the scatterplot of `v27` and `v28` that we did before, but now adds `IPVS` to plot, encoded to color and shape:

```
fig, ax = plt.subplots(figsize = (7,5))
sns.scatterplot(data = cntr, x = 'v27', y = 'v28', alpha = 0.5,
                hue = 'AGSN', style = 'AGSN', s = 30, ax = ax)
ax.set_xlabel("Average age")
ax.set_ylabel("Average income")
plt.show()
```

As we saw above, the correlation between `v27` and `v28` was a bit 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 even lower for non-special census tracts, and perhaps not so bad for subnormal census tracts. The correlation for the whole sample was 0.29. Let us calculate the correlation for non-special census tracts only:

```
corr_nao_especial = cntr[cntr['AGSN'] == 'Não especial'][['v27', 'v28']].corr().iloc[0,1]
corr_subnormal = cntr[cntr['AGSN'] == 'Subnormal'][['v27', 'v28']].corr().iloc[0,1]

print(f"Correlation for 'Não especial': {corr_nao_especial:.3f}")
```

```
Correlation for 'Não especial': 0.271
```

```
print(f"Correlation for 'Subnormal':    {corr_subnormal:.3f}")
```

```
Correlation for 'Subnormal':    0.311
```

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 cntr about census tracts in the state of São Paulo.
2. Create a statistical plot of variable `v28`. Use `axvline()` to plot the mean and the median of the variable.
3. Create a scatterplot of `v16` (proportion of children) and `v28` (income). Which variable would you put on the x-axis and why?
4. Calculate the correlation between `v16` and `v28`.
5. Recreate the scatterplot of `v16` and `v28` but to make it a multivariate plot, use the aesthetics of color and shape to encode `IPVS`. What do you learn from this plot?
6. Recalculate the correlation between `v16` and `v28` by `IPVS`. What do you learn from this?
7. Calculate the logarithm of `v28` 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 `v28` and discuss the results.