Data Visualization with Python

Categorical Data Analysis using Catplot

Hernández Pérez Erick Fernando

The simplest example

```
import seaborn as sns

df = sns.load_dataset("titanic")
sns.catplot(data=df, x="age", y="class")
```

Carplot is a function within the Seaborn library. We can say that its functionality lies in showing the relationship between a numerical variable (that which represents quantities, being continuous and discrete) and one or more categorical variables (that which represents categories or groups, their values are qualitative and can be divided into nominal and ordinal), providing us with multiple possible representations for this purpose.

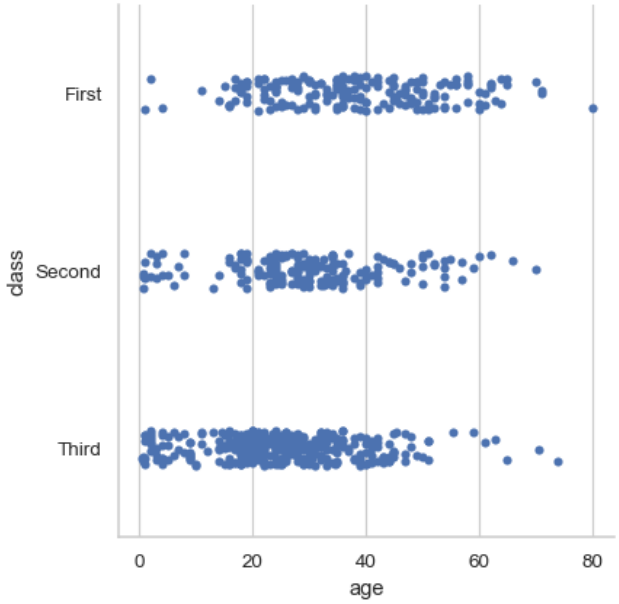

Figure 1: The simplest example

In the simplest example we can see that our numerical variable is age (values with a minimum of 0.42 and a maximum of 80, although they can be easily discretized by rounding down), while the categorical variable is class (first, second and third class).

From this graph we can obtain information in a very simple way. We can conclude that in the third class there were many people between a little before 20 years old and 40 years old; this being the class with the highest density. The second class is also centered on the 20-40 interval but is clearly more distributed across age ranges. Finally, the first class is the class that seems to be most evenly distributed between the 20-60 range, as is the class with the maximum age of 80 years.

Similarly, this example shows that the default visual representation is a jittered strip plot. But, obviously, we are not limited to this as we will see later.

1 Parameters

Before looking at other application examples, let's analyze the most important parameters that will allow us to start using catplot in our analysis.

- **data**: It is the dataset where Seaborn will look for the variables that you indicate. It can be a Pandas DataFrame, a list, a dictionary, an array, etc.
- **x**: The variable that will go on the X-axis.
- **y**: The variable that will go on the Y-axis.
- **hue**: This parameter allows you to specify a categorical variable that will be used to differentiate the data by color. The color of the dots or bars will change according to the values of the variable specified in hue. You can see how to plot a third dimension (Graph of complex variable functions for example).
- **row, col**: These are categorical variables that are used to divide the data into subgraphs (facets) and create a "grid" of graphs. Each combination of row and col values generates a different graph on a grid.
- **kind**: Define the type of chart you want to create. This parameter indicates what type of visualization you are going to use for your categorical data.
  1. "strip": A plot of points (one for each observation), scattered along the X-axis.
  2. "swarm": Similar to the strip, but the points are arranged to avoid overlapping. "box": A box plot showing the median, quartiles and outliers.
  3. "violin": Combines a box plot with a density distribution, similar to a violin.
  4. "boxen": An advanced variant of the box plot that shows more details of the distribution.
  5. "point": A dot plot representing averages and standard errors, useful for comparing categories.
  6. "bar": A bar chart showing the mean or quantity of each category.
  7. "count: A bar chart showing the number of items in each category.
- **legend**: Controls how the legend (the part of the chart that explains the meaning of the mark colors or styles) is drawn. You have several options.
  1. "auto": The legend will automatically adjust between a brief or full representation depending on the number of levels (categories) in hue or size.
  2. "brief": The legend will show only a sample of evenly spaced values, useful when you have many categories or levels in hue or size.
  3. "full": Each group or category will have a full entry in the legend, even if there are many categories.
  4. False: No legend is drawn, even if hue or size is used.

And that's it, of course there are other parameters that were left out, but these should be enough for you to try your own graphs with catplot.

Before continuing, we will see how the type of graph can give us more or less information.

Compare the analysis given for the simplest example and, as an exercise, complete it with your own analysis.

The simplest example with 'boxen'

```
sns.catplot(data=df, x="age", y="class", kind='boxen')
```

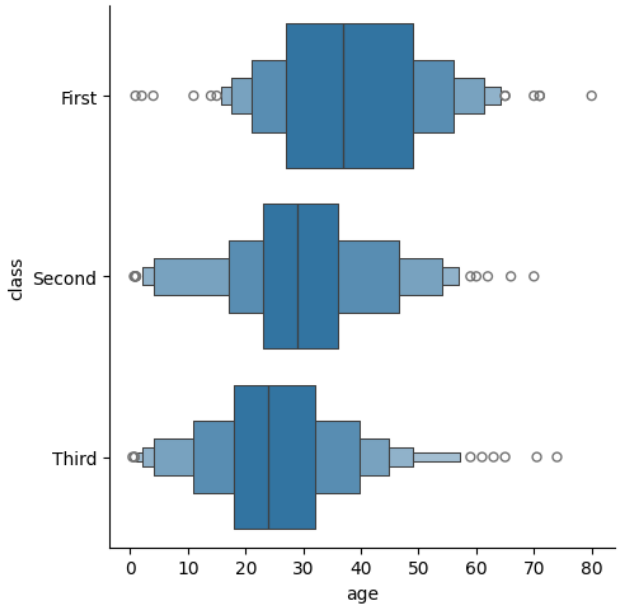

Figure 2: The simplest example but with 'boxen

2 Applied cases

We will consider the dataset Seaborn has 'tips', which is a dataset containing information about tips given in a restaurant. The reason for this choice is that it has several categorical variables.

tips

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

tips = sns.load_dataset("tips")
tips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   total_bill  244 non-null    float64
 1   tip         244 non-null    float64
 2   sex         244 non-null    category
 3   smoker      244 non-null    category
 4   day         244 non-null    category
 5   time        244 non-null    category
 6   size        244 non-null    int64
dtypes: category(4), float64(2), int64(1)
memory usage: 7.4 KB
```

Figure 3: tips.info()

tips

```
sns.catplot(data=tips, x='day', y='tip', hue='sex', kind='violin')
```

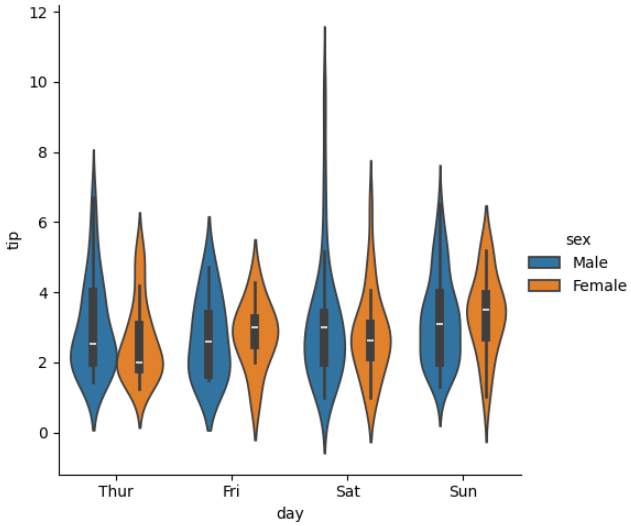

Figure 4: Tips given by each sex on each day

tips

```
sns.catplot(data=tips, x='day', y='total_bill', hue='sex', kind='violin')
```

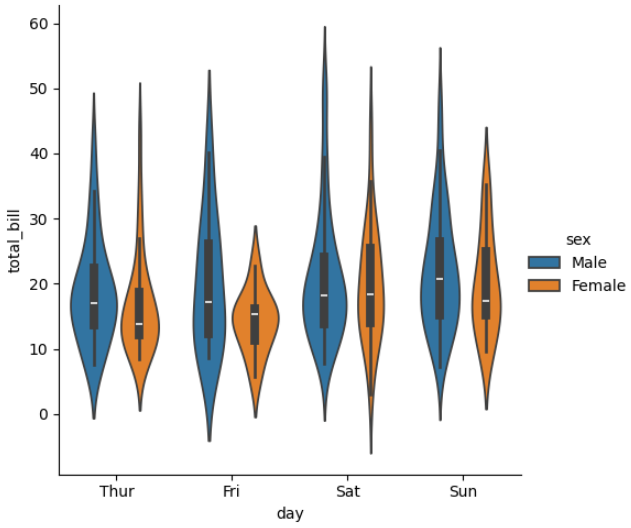

Figure 5: Total bill for each day depending on sex

tips

```
encoder = LabelEncoder()

tips['sex'] = encoder.fit_transform(tips['sex']) # 0-F 1-M
tips['smoker'] = encoder.fit_transform(tips['smoker'])
```

```
tips['time'] = encoder.fit_transform(tips['time']) # 0-D 1-L
day_mapping = {'Thur': 1, 'Fri': 2, 'Sat': 3, 'Sun': 4}
tips['day'] = tips['day'].map(day_mapping)

df_melt = pd.melt(tips, id_vars=['time'], value_vars=['smoker', 'sex'])
df_melt = df_melt.groupby(['time', 'variable'])['value'].value_counts().to_frame()

g = sns.catplot(df_melt, x='variable', y='count', hue='value',
                col='time',kind='bar')

new_labels = ["[Female, No]", "[Male, Yes]"]
for t, l in zip(g._legend.texts, new_labels):
    t.set_text(l)
```

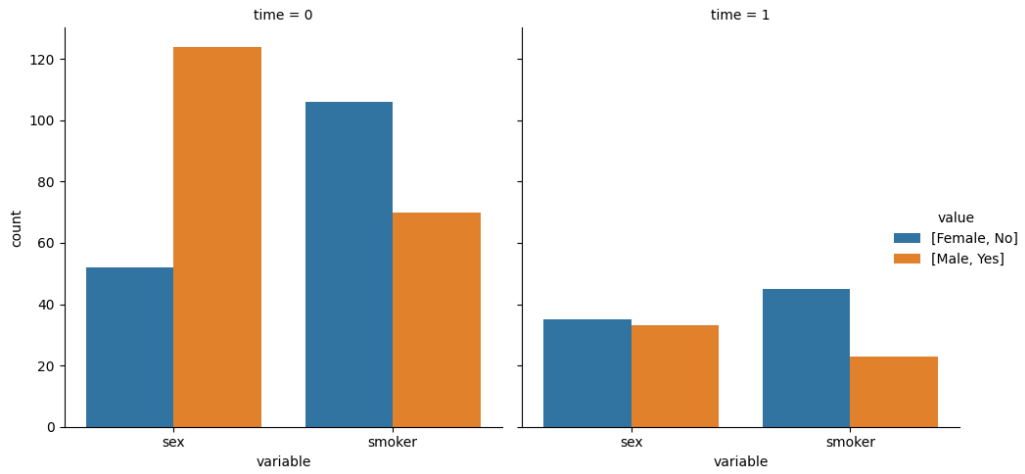

Figure 6: How combinations of smoker (smoker/non-smoker) and sex (gender) are distributed by time of day (lunch or dinner)

3 References

Waskom, M. L., (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021, https://doi.org/10.21105/joss.03021.