Python Functions - matplotlib

Python Function

All codes will be Python unless otherwise marked.

```
def my_function():
    print("Hello from Function")
my_function()
```

```
Hello from Function
```

Let us write a function that shift and rescale its entries between 0 and 1.

Function 1:

```
def rescale_1(w):
    w0 = w
    # Keep w0 to print the original data at the end
    m = min(w)
    w1 = [x - m for x in w]
    # Subtract each element from its minimum

    r = max(w1) - min(w1)
    w2 = [x / r for x in w1]
    # then Divide each element with range of vector

    print("The original vector is \n", w0,
    "\n and the vector after rescaling is \n", w2)
```

```
x = [2, 5, 7, 9]
rescale_1(x)
```

```
The original vector is
 [2, 5, 7, 9]
 and the vector after rescaling is
 [0.0, 0.42857142857142855, 0.7142857142857143, 1.0]
```

With None value

Remove None

```
vec = [2, 5, 7,None,  9]
vec_1 = [x for x in vec if x is not None]
print("vector \n", vec,
"\nAfter removing None", vec_1)
```

```
vector
 [2, 5, 7, None, 9]
After removing None [2, 5, 7, 9]
```

SECOND FUNCTION

```
def rescale_2(w):
    w0 = w
    w = [x for x in w if x is not None]
    m = min(w)
    w1 = [x - m for x in w]

    r = max(w1) - min(w1)
    w2 = [x / r for x in w1]  # then Divide each element wi

    print("The original vector is \n", w0, "\n and the vect

rescale_2(vec)
```

```
The original vector is
 [2, 5, 7, None, 9]
 and the vector after rescaling is
 [0.0, 0.42857142857142855, 0.7142857142857143, 1.0]
```

With Infinity

```
import math
vec_i = [2, 5, 7, math.inf,  9]
# Because of infinity and any number divided by infinity is
# zero. It makes almost all values zero
rescale_2(vec_i)
```

```
The original vector is
 [2, 5, 7, inf, 9]
 and the vector after rescaling is
 [0.0, 0.0, 0.0, nan, 0.0]
```

THIRD FUNCTION

```
def rescale_3(w):
    w0 = w
    w = [x for x in w if x is not math.inf]
    m = min(w)
    w1 = [x - m for x in w]

    r = max(w1) - min(w1)
    w2 = [x / r for x in w1]

    print("The original vector is \n", w0,
    "\n and the vector after rescaling is \n", w2)

rescale_3(vec_i)
```

```
The original vector is
 [2, 5, 7, inf, 9]
 and the vector after rescaling is
 [0.0, 0.42857142857142855, 0.7142857142857143, 1.0]
```

The following will not work

```
vec_inf = [2, 5, 7,-math.inf, math.inf,  9]
rescale_3(vec_inf)
```

```
The original vector is
 [2, 5, 7, -inf, inf, 9]
 and the vector after rescaling is
 [nan, nan, nan, nan, nan]
```

Need to use

```
m = min(vec_inf)
M= max(vec_inf)
vec_inf.remove(m)
vec_inf.remove(M)
print(vec_inf)
```

```
[2, 5, 7, 9]
```

- as you can see we have removed infinities

FOURTH FUNCTION

```
def rescale_4(w):
    print("The original vector is ", w)
    m = min(w)
    M = max(w)
    w.remove(m)
    w.remove(M)
    # Now w does not contain inf and - inf
    m = min(w)
    w1 = [x - m for x in w]
    r = max(w1) - min(w1)
    w2 = [x / r for x in w1]

    print("The vector after rescaling is \n", w2)
```

```
v4 = [2, 5, 7,-math.inf, math.inf,  9]
rescale_4(v4)
```

```
The original vector is  [2, 5, 7, -inf, inf, 9]
The vector after rescaling is
 [0.0, 0.42857142857142855, 0.7142857142857143, 1.0]
```

ISSUE WITH THIS FUNCTION

There is some issue with this function. Can you find it?

```
v = [2, 5, 7, 9, -1]
rescale_4(v)
```

```
The original vector is  [2, 5, 7, 9, -1]
The vector after rescaling is
 [0.0, 0.6, 1.0]
```

FIFTH FUNCTION

```
def rescale_5(w):
    w0 = w
    w = [x for x in w if x is not None]
    m = min(w)
    M = max(w)
    if m == -1*math.inf:
        w.remove(m)
    if M == math.inf:
        w.remove(M)
    m = min(w)
    w1 = [x - m for x in w]
    r = max(w1) - min(w1)
    w2 = [x / r for x in w1]
    print("The original vector is \n", w0,
    "\n and the vector after rescaling is \n", w2)
```

```
v4 = [2, 5, 7,-math.inf, math.inf,  9]
rescale_5(v4)
```

```
The original vector is
 [2, 5, 7, -inf, inf, 9]
 and the vector after rescaling is
 [0.0, 0.42857142857142855, 0.7142857142857143, 1.0]
```

```
v = [2, 5, 7, 9, -1, None]
rescale_5(v)
```

```
The original vector is
 [2, 5, 7, 9, -1, None]
 and the vector after rescaling is
 [0.3, 0.6, 0.8, 1.0, 0.0]
```

Python Overview

In R I Want	In Python I Use
Base R	numpy
dplyr/tidyr	pandas
ggplot2	matplotlib/seaborn

Import Matplotlib and Seaborn, and Load Dataset

R

```
library(ggplot2)
data("mpg")
```

Importing libraries

Python: `import <package> as <alias>`.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
mpg = r.mpg
```

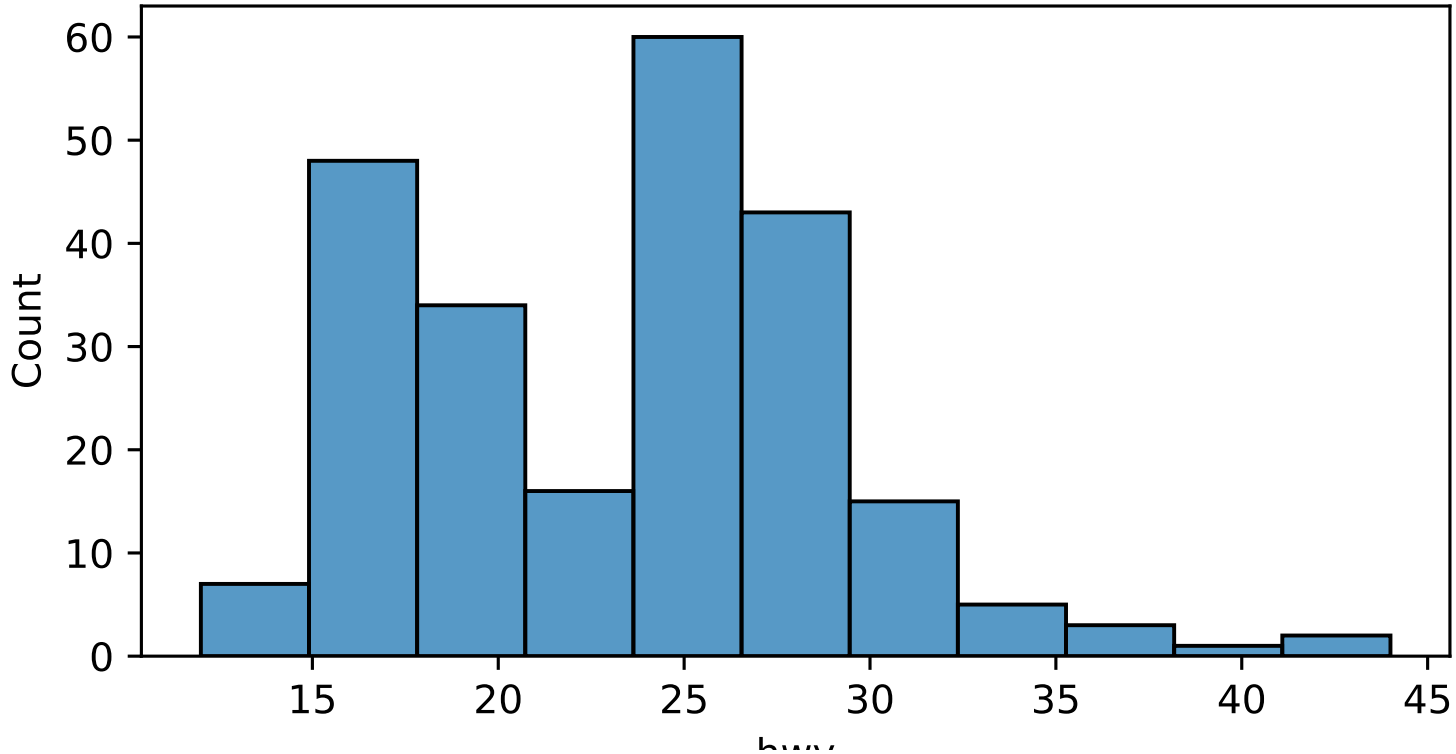
Show and clear plots.

- Use `plt.show()` to display a plot.
- Use `plt.clf()` to clear a figure when making a new plot.

One Quantitative Variable: Histogram

- `sns.histplot()` makes a histogram.

```
sns.histplot(x='hwy', data=mpg)
plt.show()
```

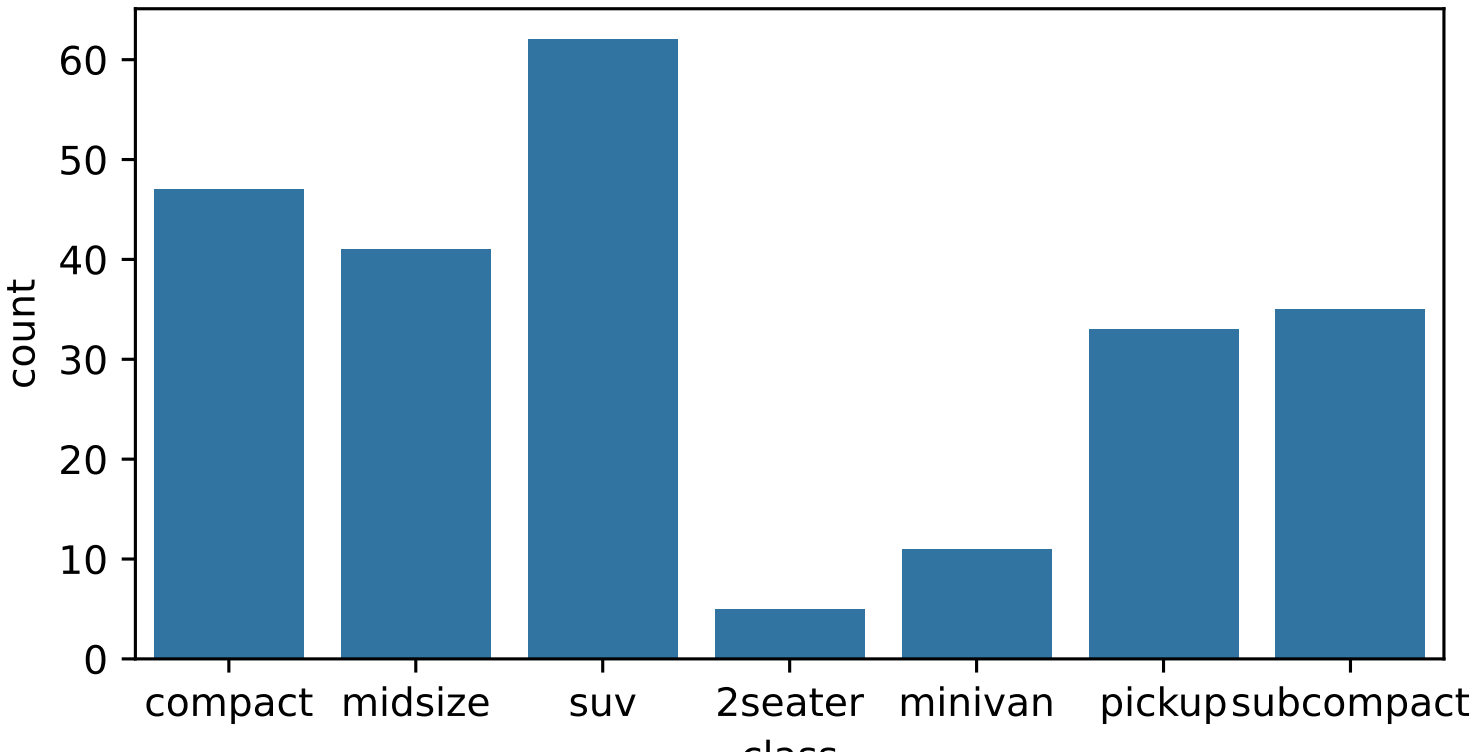



```
plt.clf()
```

One Categorical Variable: Barplot

- Use `sns.countplot()` to make a barplot to look at the distribution of a categorical variable:

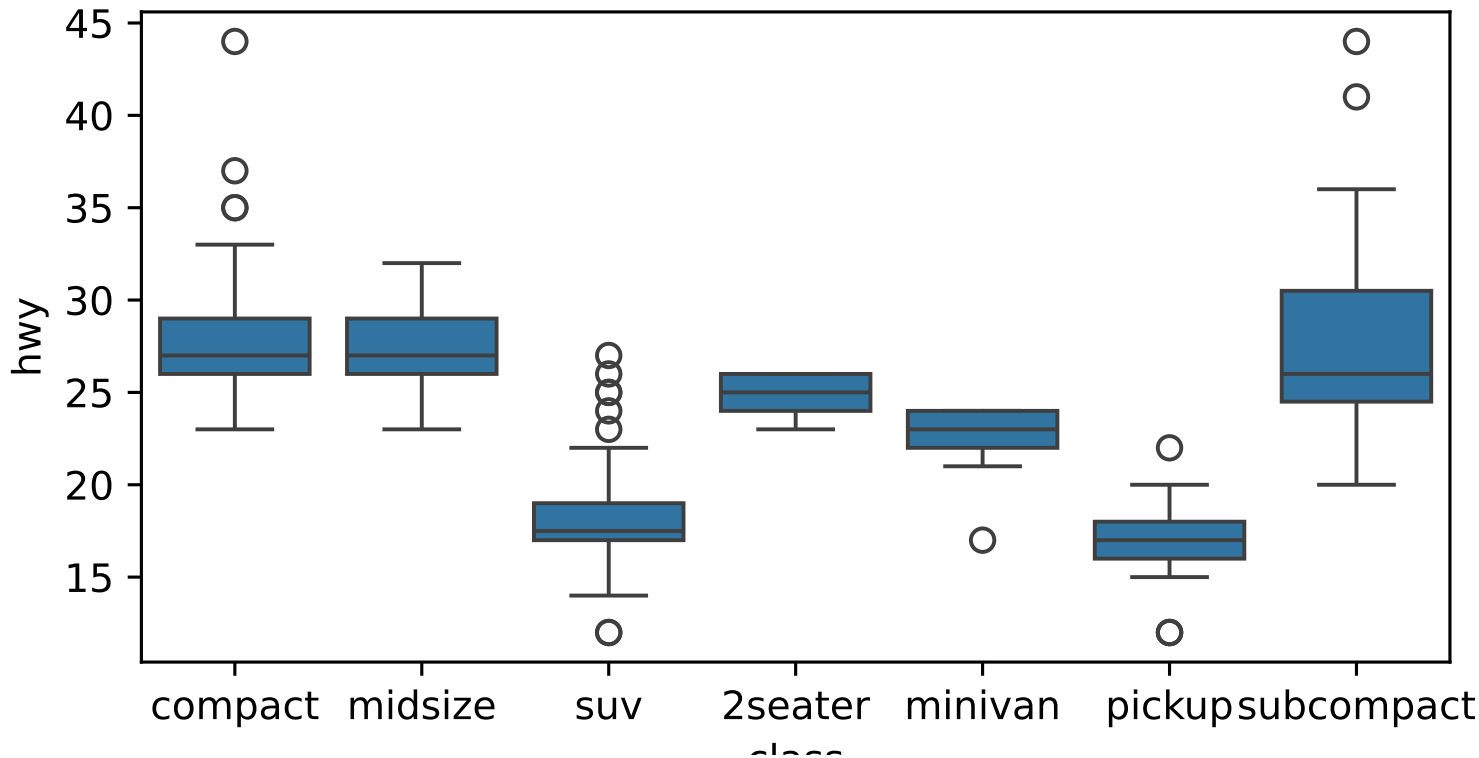
```
sns.countplot(x='class', data=mpg)
plt.show()
```



One Quantitative Variable, One Categorical Variable: Boxplot

- Use `sns.boxplot()` to make boxplots:

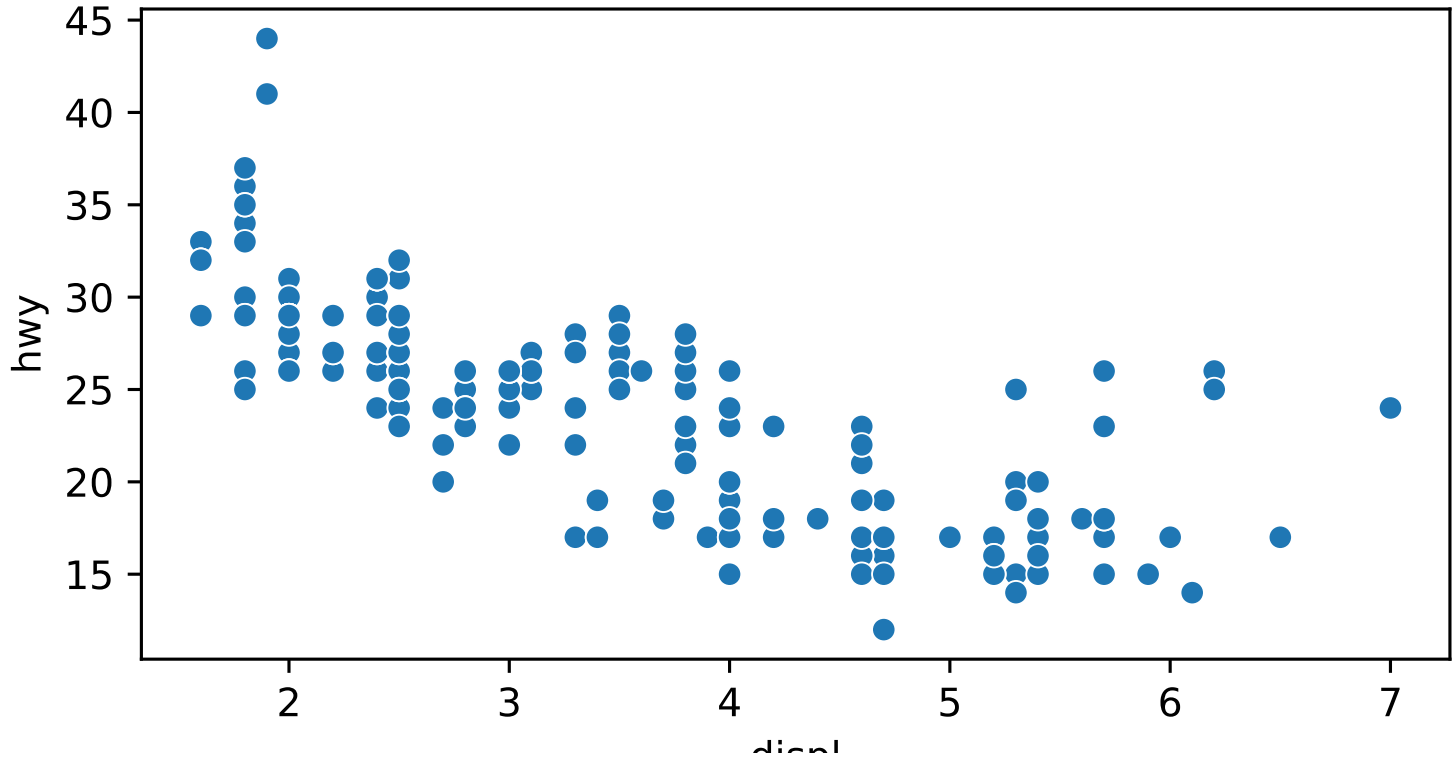
```
sns.boxplot(x='class', y='hwy', data=mpg)
plt.show()
```



Two Quantitative Variables: Scatterplot

- Use `sns.scatterplot()` to make a basic scatterplot.

```
sns.scatterplot(x='displ', y='hwy', data=mpg)
plt.show()
```

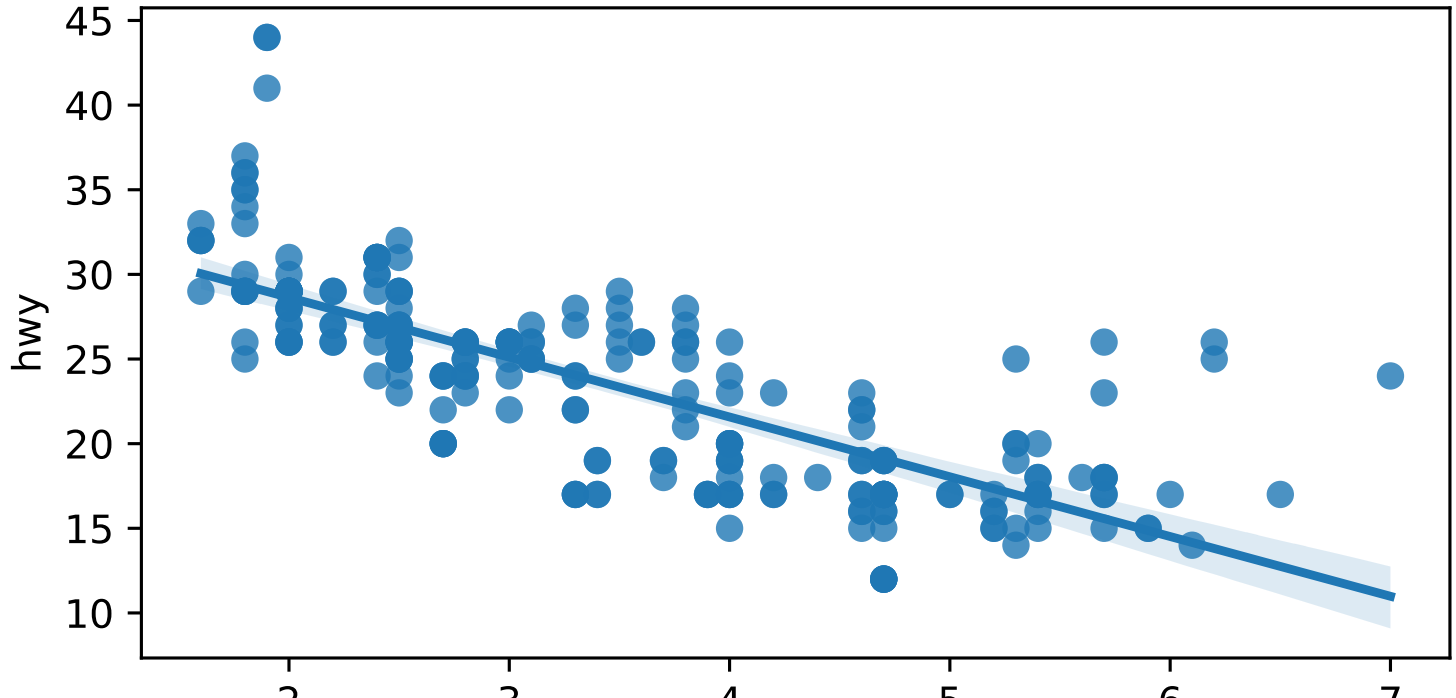



```
plt.clf()
```

Lines/Smoothers

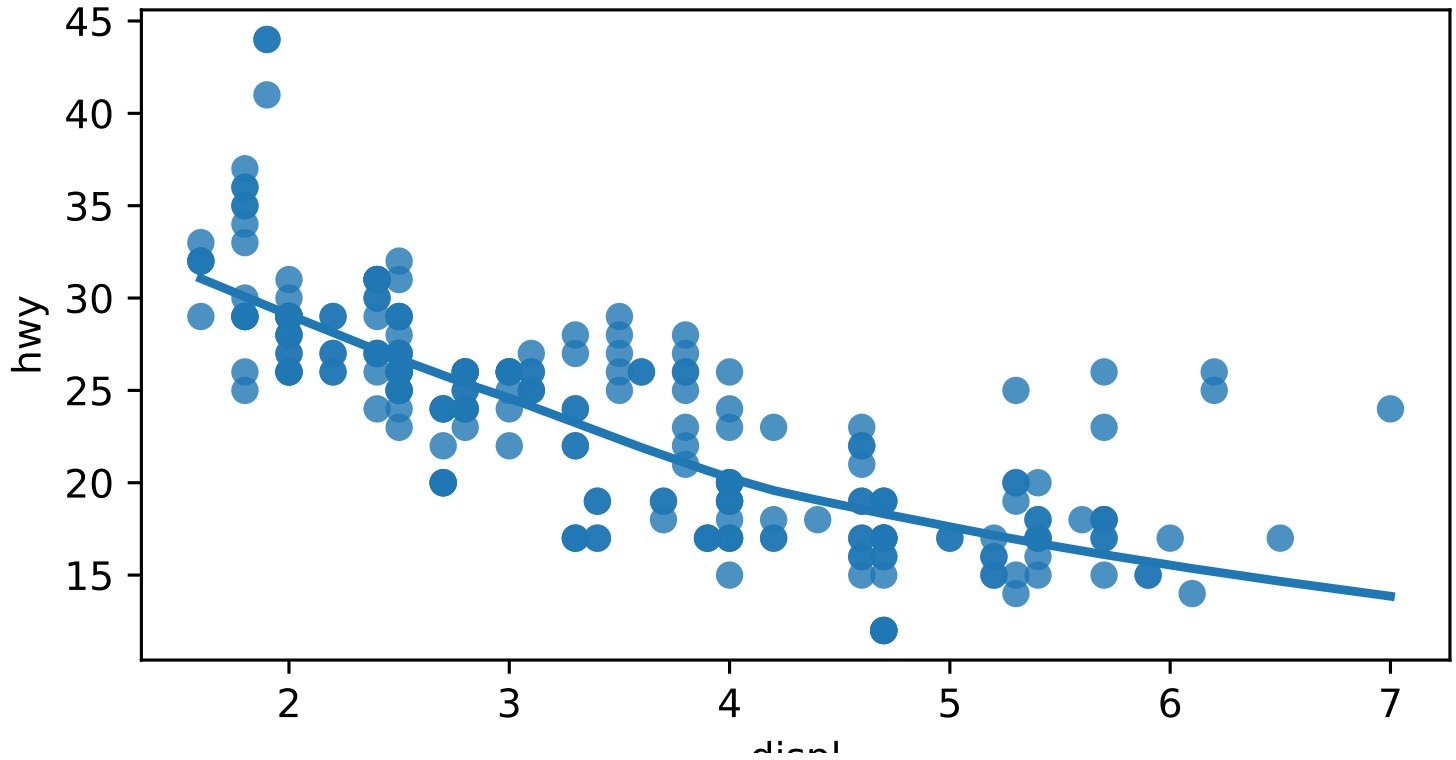
- Use `sns.regplot()` to make a scatterplot with a regression line or a loess smoother.
- Regression line with 95% Confidence interval

```
sns.regplot(x='displ', y='hwy', data=mpg)
plt.show()
```



Loess smoother with confidence interval removed.

```
sns.regplot(x='displ', y='hwy', data=mpg, lowess=True, ci=
plt.show()
```

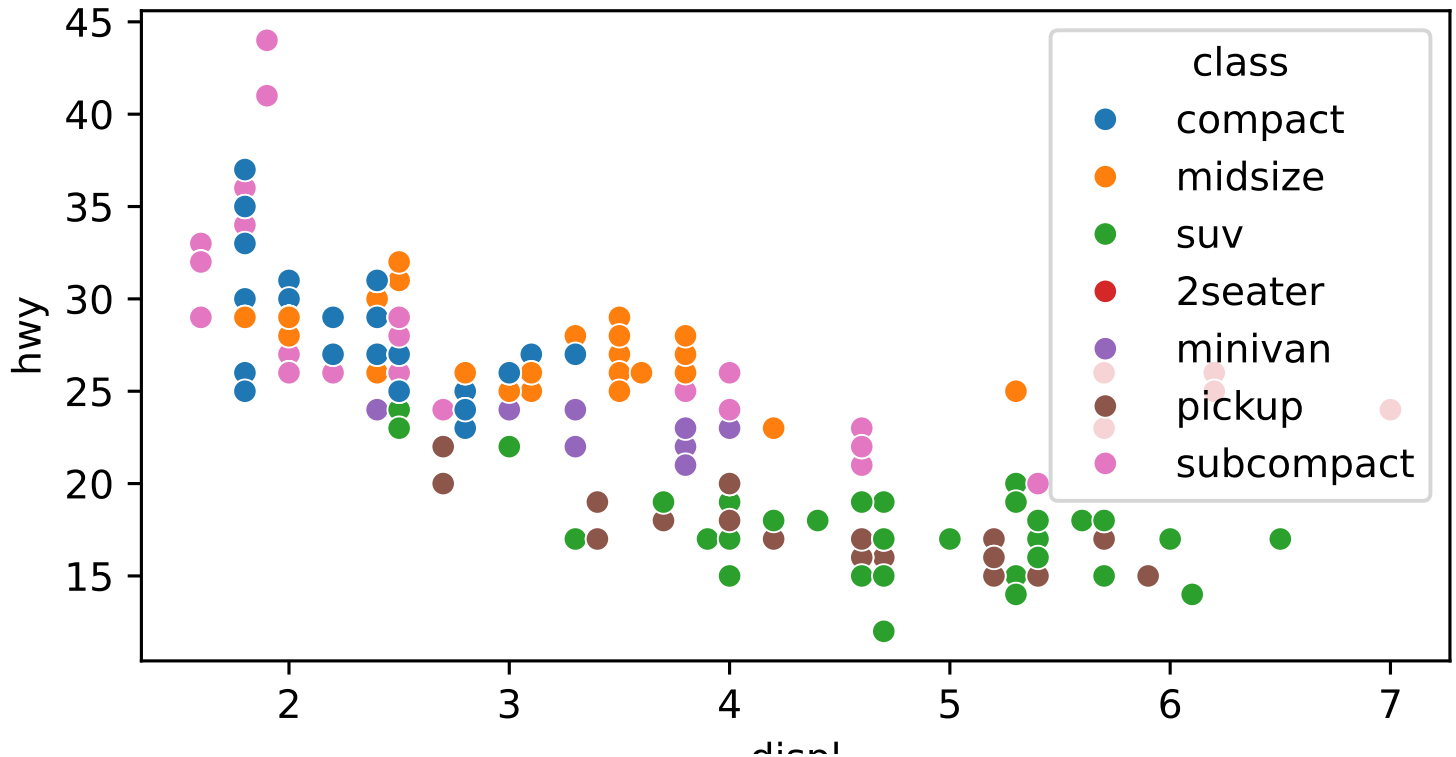



```
plt.clf()
```

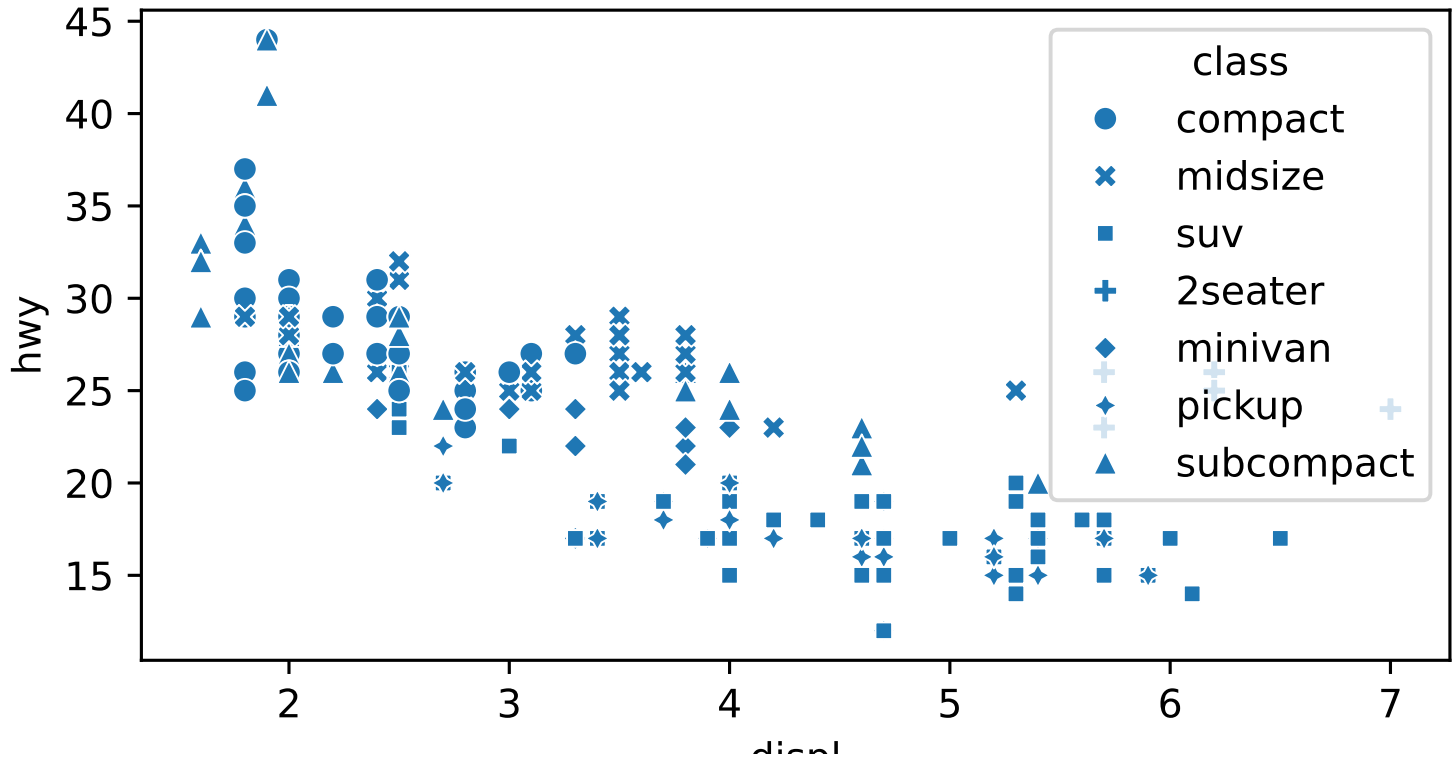
Annotating by Third Variable

- Use the `hue` or `style` arguments to annotate by a categorical variable:

```
sns.scatterplot(x='displ', y='hwy', hue='class', data=mpg)
plt.show()
```



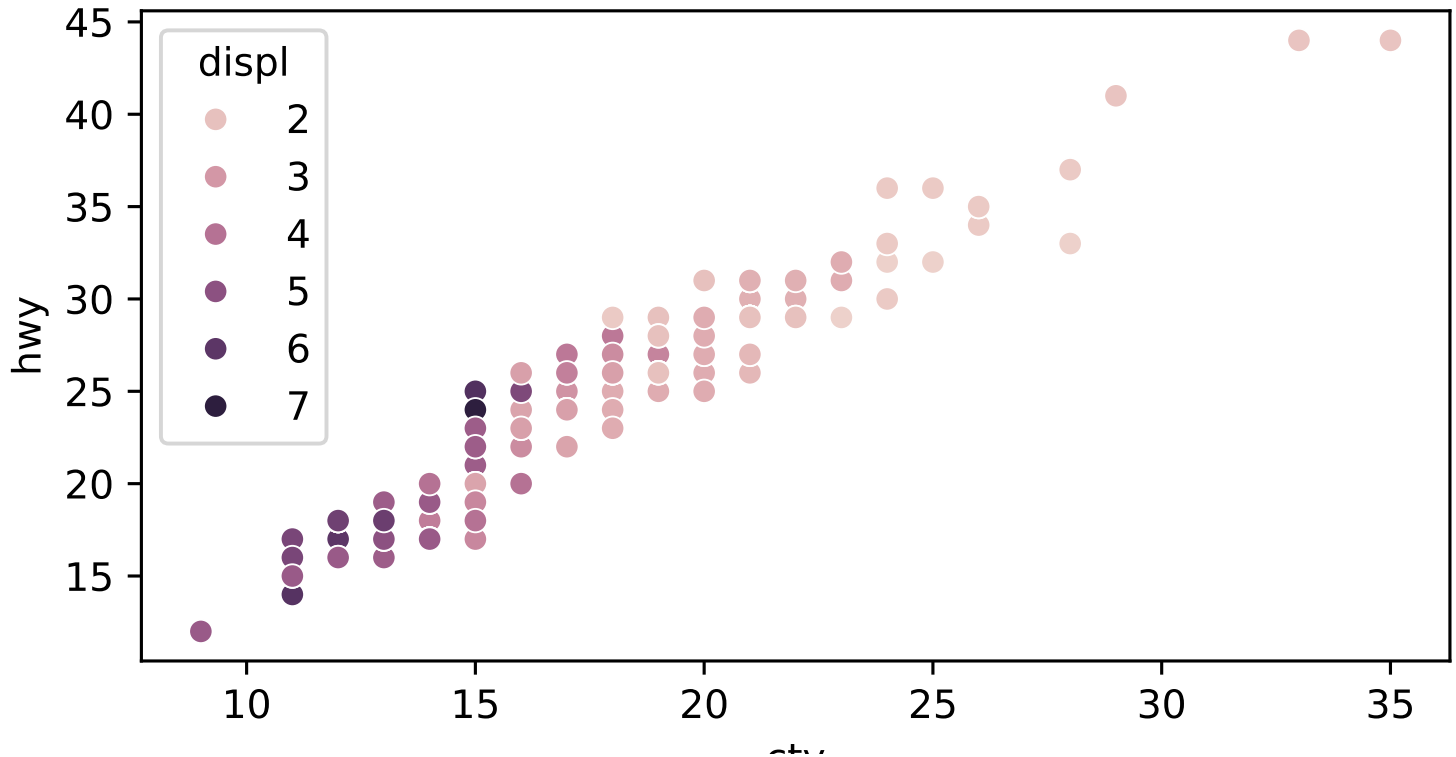
```
sns.scatterplot(x='displ', y='hwy', style='class', data=mpg
plt.show()
```




```
plt.clf()
```

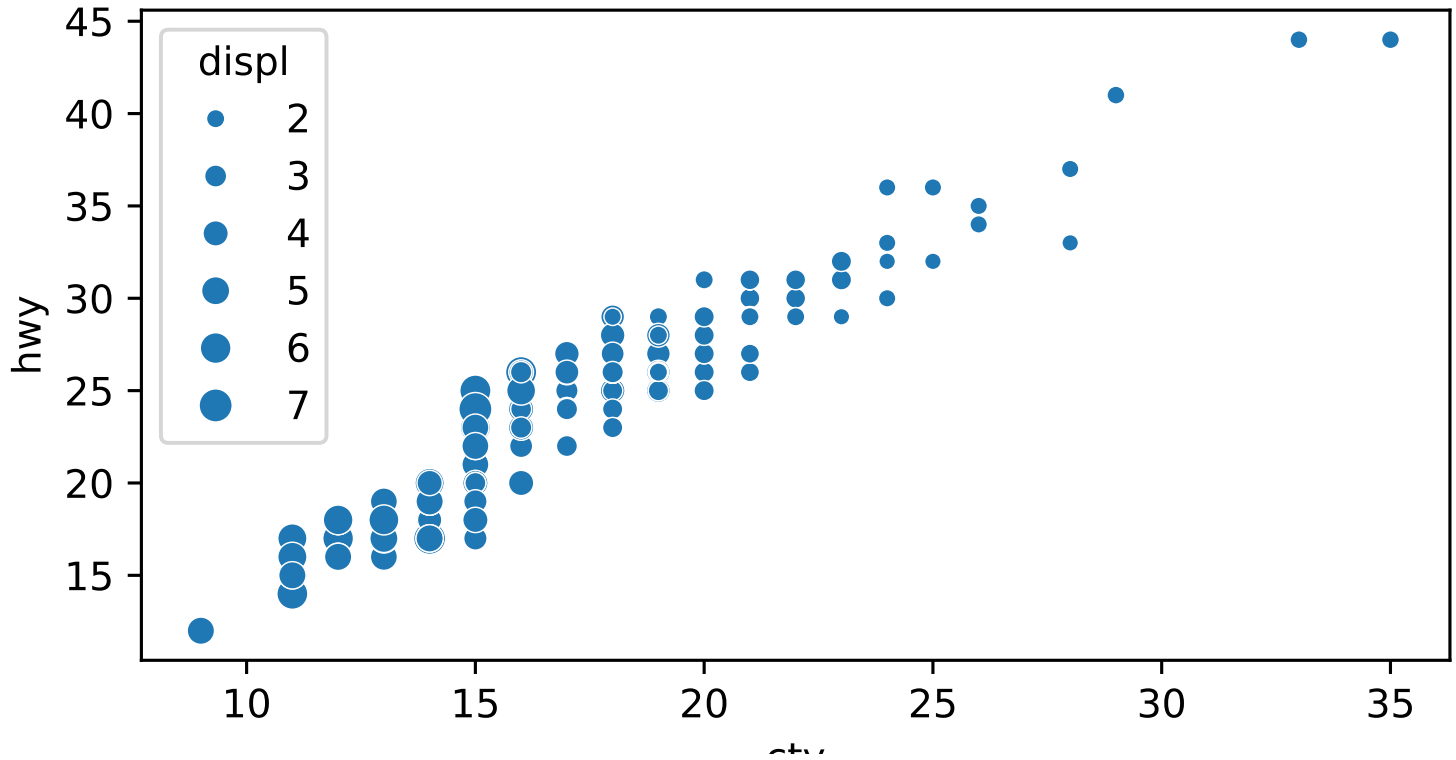
Use the `hue` or `size` arguments to annotate by a quantitative variable:

```
sns.scatterplot(x='cty', y='hwy', hue='displ', data=mpg)
plt.show()
```




```
plt.clf()
```

```
sns.scatterplot(x='cty', y='hwy', size='displ', data=mpg)
plt.show()
```




```
plt.clf()
```

Two Categorical Variables: Mosaic Plot

- Usually, you should just show a table of proportions when you have two categorical variables.

```
pd.crosstab(mpg['class'], mpg['drv'], normalize='all')
```

drv	4	f	r
class			
2seater	0.000000	0.000000	0.021368
compact	0.051282	0.149573	0.000000
midsize	0.012821	0.162393	0.000000
minivan	0.000000	0.047009	0.000000
pickup	0.141026	0.000000	0.000000
subcompact	0.017094	0.094017	0.038462
suv	0.217949	0.000000	0.047009

```
pd.crosstab(mpg['class'], mpg['drv'], normalize='index')
```

drv	4	f	r
class			
2seater	0.000000	0.000000	1.000000
compact	0.255319	0.744681	0.000000
midsize	0.073171	0.926829	0.000000
minivan	0.000000	1.000000	0.000000
pickup	1.000000	0.000000	0.000000
subcompact	0.114286	0.628571	0.257143
suv	0.822581	0.000000	0.177419

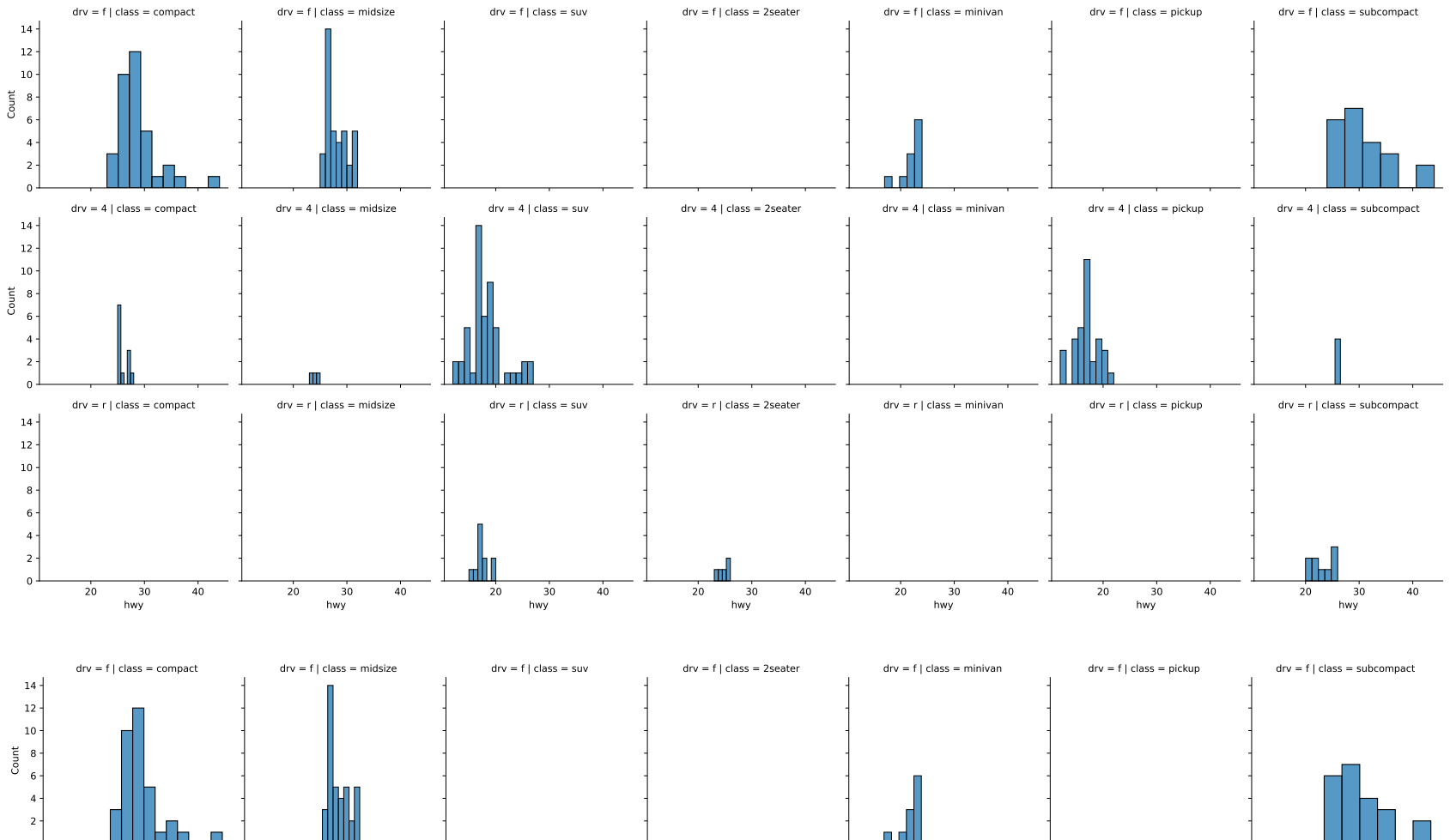
```
pd.crosstab(mpg['class'], mpg['drv'], normalize='columns')
```

drv	4	f	r
class			
2seater	0.000000	0.000000	0.20
compact	0.116505	0.330189	0.00
midsize	0.029126	0.358491	0.00
minivan	0.000000	0.103774	0.00
pickup	0.320388	0.000000	0.00
subcompact	0.038835	0.207547	0.36
suv	0.495146	0.000000	0.44

Facets

- Use `sns.FacetGrid()` followed by the `map()` method to plot facets.

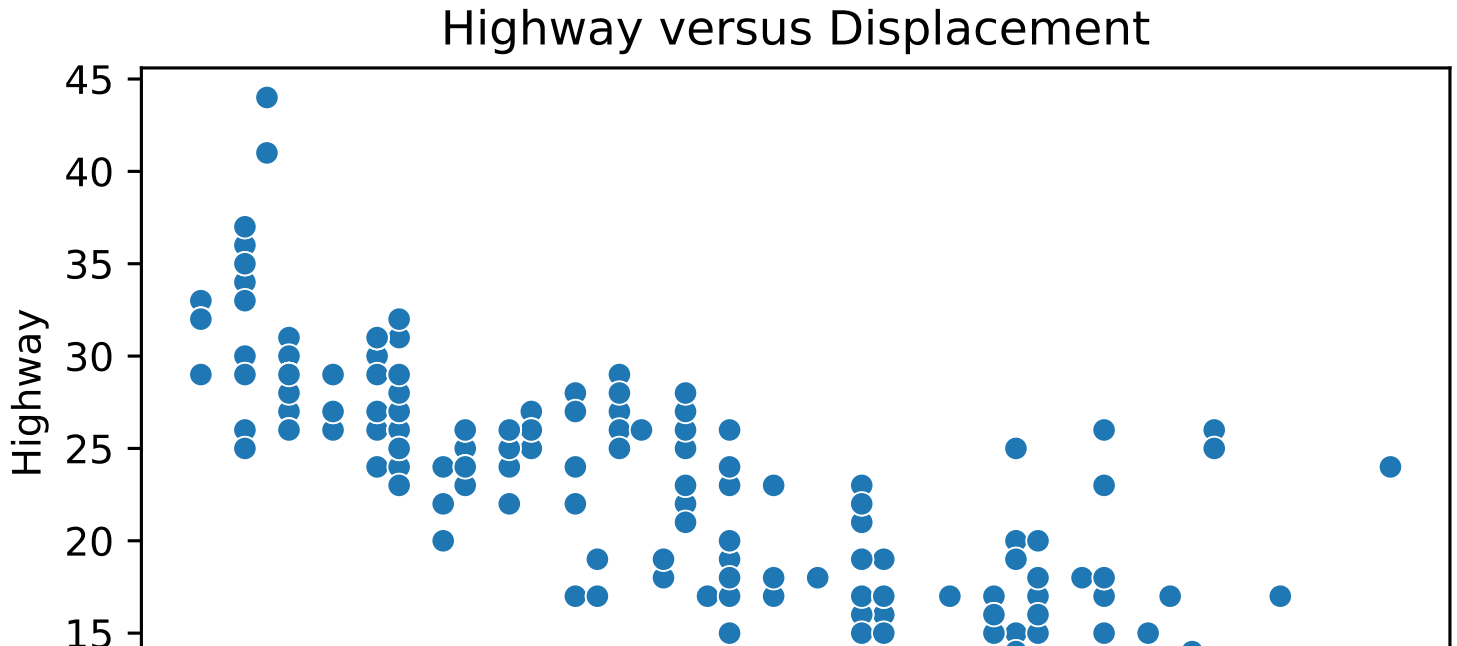
```
g = sns.FacetGrid(data=mpg, row='drv', col='class')
g.map(sns.histplot, 'hwy', kde=False)
```



Labels

- Assign plot to an object. Then use the `set_*()` methods to add labels.

```
scatter = sns.scatterplot(x='displ', y='hwy', data=mpg)
scatter.set_xlabel('Displacement')
scatter.set_ylabel('Highway')
scatter.set_title('Highway versus Displacement')
plt.show()
```



Saving Figures

1. First, assign a figure to an object.

```
scatter = sns.scatterplot(x='displ', y='hwy', data=mpg)
```

2. Extract the figure. Assign this to an object.

```
fig = scatter.get_figure()
```

3. Save the figure.

```
fig.savefig('./scatter.pdf')
```

You can do all of these steps using piping.

```
sns.scatterplot(x='displ', y='hwy', data=mpg) \
  .get_figure() \
  .savefig('../graphix/scatter.pdf')
```