CAP 5768: Introduction to Data Science

Introduction to Python and Data Visualization

Using Python

- In this course we will employ the Python for ML and Visualization
- Python is one of the popular languages used in data science (the other is R).
- You can use the IDE for of your comfort (i.e. PyCharm, Spyder, Vs Code)
- Recommended to use Jupyter Notebook: an interactive notebook documents that can contain live code, equations, visualizations, media and other computational outputs.
- Or you can use Google Colab:
  https://colab.google/

Installing the **Python** Packages

In this course, we will make use of a number of different packages, which I will make clear when needed.

However, initially we will need to install the **pandas, numpy, sklearn, matplotlib, seaborn** packages in the Command Prompt, Anaconda Prompt, or in Jupyter Notebook directly

```
pip install <package_name>

conda install <package_name>

!pip install <package_name>  - Jupyter
```

Optionally, create a virtual environment

Python “Virtual Environments” allow Python packages to be installed in an isolated location for a particular application, rather than being installed globally.

Create Virtual Env

```
Conda create --name <myenvname> python=3.9
```

Activate and Use Virtual Env

```
conda activate <myenvname>
```

List all available Virtual Env

```
conda info --envs
```

Loading packages for use in Python

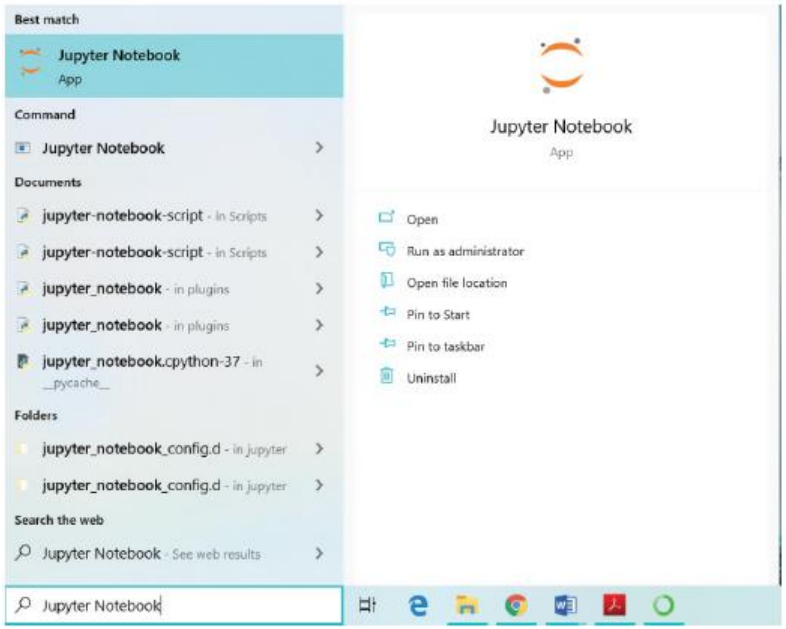
Import and Load the package

```
import <package_name>
```

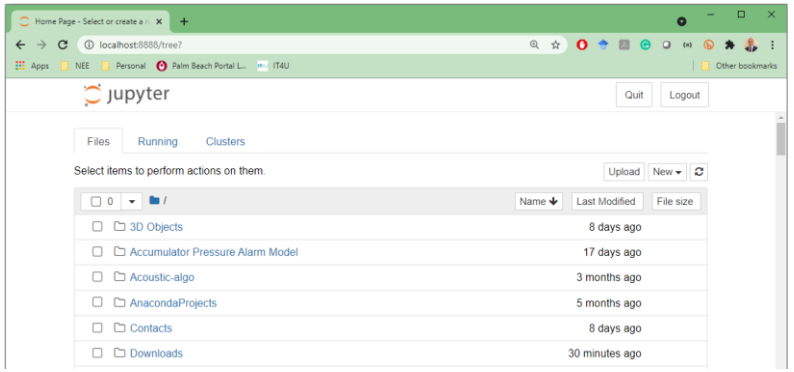
Giving an alias to the package:

```
import <package_name> as p
```

Starting with Jupyter Notebook



http://localhost:8888/



Or from the Anaconda Prompt:

```
Jupyter notebook
```

Python Variables and Data Types

- **Strings**

```
fname = 'Joe'
print(fname)
```

- **Integers**

```
age = 30
print(age)
print(type(age))
```

- **Floating Point Numbers**

```
weight = 12.6
print(weight)
```

- **Booleans**

```
male = True
print(type(male))
```

- **Lists**

```
grades = ["first", "second", "third"]
print(type(grades))
```

- **Tuples**

```
days = ("Saturday", "Sunday", "Monday")
print(type(days))
```

- **Dictionaries**

```
days2 = {1:"Saturday", 2:"Sunday", 3: "Monday"}
print(type(days2))
days2.keys()
days2.values()
```

Python Operations

- **Arithmetic Operations**

```
+       -       *       /       %       **
```

- **Logical Operations**

```
and     or      not
```

- **Comparison Operations**

```
==      !=      >       <       >=      <=
```

- **Assignment Operations**

```
=       +=      -=      *=      /=      %=      **=
```

Conditional Statements

- **If Statement**

```
X = 5
if x > 2:
    print ('x is greater than 2')
```

- **If-else Statement**

```
X = 5
if x > 2:
        print ('x is greater than 2')
else:
        print ('x is less than 2')
```

- **If-elseif Statement**

```
if 5 > 20:
        print ('5 is greater than 20')
elif 10 < 5:
        print ('10 is less than 5')
else:
        print ('None of the above')
```

Iteration Statements

- **For Loop:**

```
produces = (‘apple', ‘banana’, ‘cucumber’, ‘dill’, ’eggplant’
for p in produces:
        print (p)
```

- **While Loop:**

```
c = 0
while c < 10:
        print(c)
        c = c + 1
```

Functions

- **Functions are implemented in any programming language for segment of code that is required to be executed numerous times.**

```
def myfunction(name, age, job):
        return "My Name is  %s, I am %s, I work as %s" %(name,age,job)
```

Calling the function:

```
myfunction('John',37,'Data Scientist')
```

```
My Name is John and I am 37 and I work as Data Scientist
```

```
def net_income(base, bonus, tax_percentage):
    income = base + bonus
    deductions = income * (tax_percentage / 100)
    net = income - deductions
    return net

net_income(150000, 30000, 12)
```

```
158400.0
```

Where to find public datasets?

- **Kaggle**: Kaggle provides a vast container of datasets, sufficient for the enthusiast to the expert.

  https://www.kaggle.com/

- **Github:**

  https://www.kaggle.com/

- **UCI Machine Learning Repository**: The Machine Learning Repository at UCI provides an up-to-date resource for open-source datasets.

  https://archive.ics.uci.edu/ml/index.php

numpy library

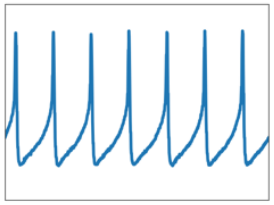
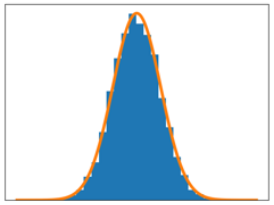
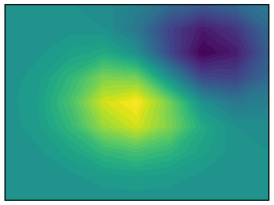
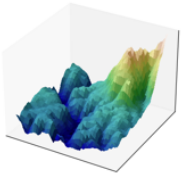
NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

```
np.mean(), np.sum(), np.std(), np.sin(), np.round(), np.exp(),
np.log(), np.power(), np.min(), np.max() …
```

https://numpy.org/doc/stable/contents.html

matplotlib library

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

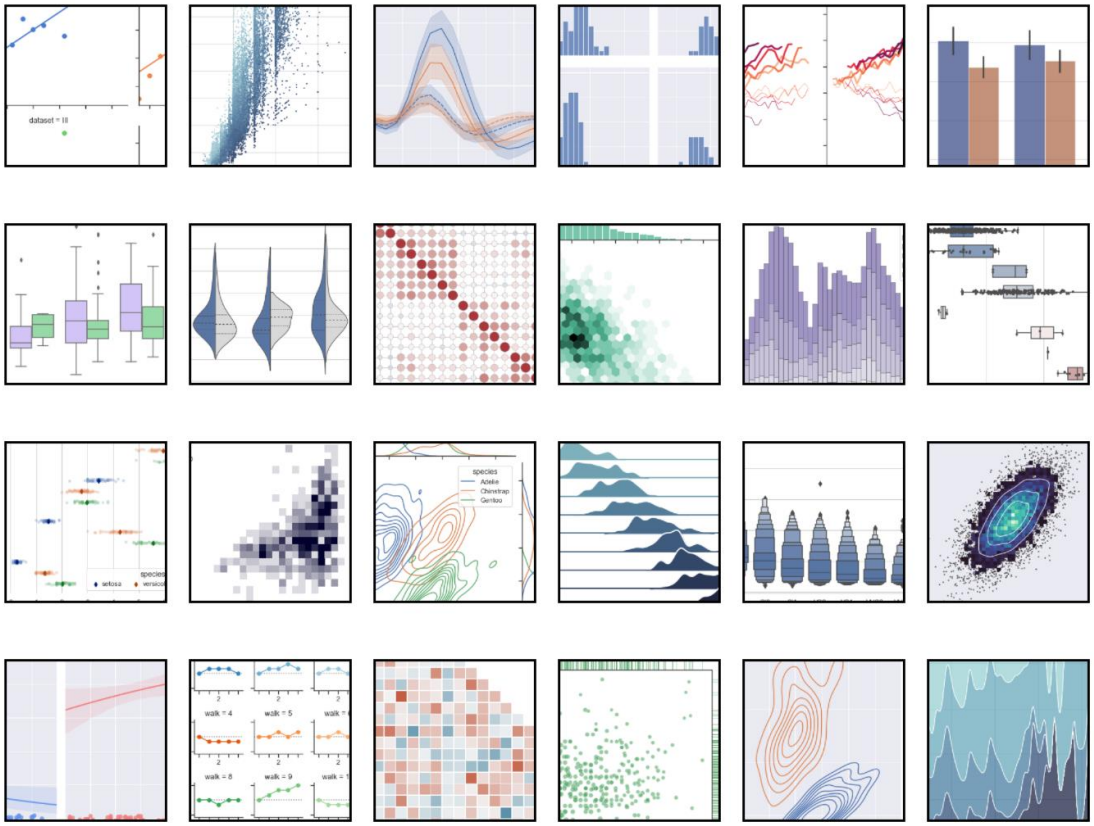





```
import matplotlib.pyplot as plt
```

https://matplotlib.org/

seaborn library

- Seaborn is built on top of the matplotlib library, with more styles and color palettes. More pleasing and aesthetic graphs



```
import seaborn as sns
```

https://seaborn.pydata.org/

pandas library

- A fast and efficient **DataFrame** object for data manipulation with integrated indexing;
- Tools for **reading and writing data** between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDF5 format;
- Intelligent label-based **slicing**, **fancy indexing**, and **subsetting** of large data sets;
- Aggregating or transforming data with a powerful **group by** engine allowing split-apply-combine operations on data sets;
- High performance **merging and joining** of data sets;
- **Time series**-functionality: date range generation and frequency conversion, moving window statistics, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data;

https://pandas.pydata.org/

os library

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see open(), if you want to manipulate paths, see the os.path module, and if you want to read all the lines in all the files on the command line see the fileinput module. For creating temporary files and directories see the tempfile module, and for high-level file and directory handling see the shutil module.

```
os.chdir(path)
os.getcwd()
os.listdir()
os.mkdir()
os.remove()
os.rename()
```

```
f = open("test.txt", "w")
f.write("Now the file has more content!")
f.close()
```

https://docs.python.org/3/library/os.html

scikit-learn library

- **In python is called `sklearn`, where it is powerful and extremely useful library for data science and machine learning in Python.**
- **The library contains many built-in modules to perform data preparation tasks such as feature engineering, feature scaling, outlier detection.**
- **We do classification, regression, and clustering using the sklearn library.**
- **https://scikit-learn.org/stable/**

Data Wrangling

Data Read

```
df = pandas.read_csv()
df = pandas.read_excel()
df = pandas.read_sql_table()
```

Descriptive

```
df.shape
df.describe()
df.head()
df.columns
df.sort_values(by = [‘column])
```

Filtering (Slice and Dice)

```
df[‘col’].contains()
df[df[‘column’]==‘XXX’ or 1234]]
df[df[‘column’]==‘XXX’ and 1234]]
df.filter([])
```

Cleaning and Transformation

```
df.fillna()
df.dropna()
df.copy()
df.drop(‘column’)
df=df[df[‘name’,number]]
```

Apply Function

```
new_df= df.column.apply(lambda x: x+2)
```

Coding basics (using Python as a calculator)

Python can be used as a basic calculator.

Typing the following code into notebook, with the results printed to the console in dark red typeface.

```
1 / 200 * 30
```

```
[1] 0.15
```

```
59 + 73 + 2) / 3
```

```
[1] 44.66667
```

Coding basics (creating new objects, or variables)

A new object can be created with the assignment operator '=' as

```
object_name = value
```

Typing the following into the console stores the value 12 into an object (variable) named x.

```
x = 3 * 4
```

No output is given to the console after this assignment.

This variable can then be used in downstream calculations

```
x * 2 + 3
27
```

Object names

Object names must start with a letter, and can only use **letters**, **numbers**, **underscores** '_', and **periods** '.'.

Python is a case sensitive language

```
LANGUAGE != language
```

Typically, we want objects to be as descriptive as possible.

It is good to be consistent with naming conventions when using multiple words, with some examples as

```
multiple_word_example

multipleWordExample

multiple.word.example

Multiple_word_EXAMPLE
```

Probably not best

Inspecting objects

Once an object has been created, its value can be inspected by typing the object name in the console.

For example, suppose we assign the value eight to the new variable

```
my_New_variable = 2 ** 3
```

Typing the variable name in the console yields its value

```
my_New_variable
8
```

Inspecting objects

Now, suppose that when inspecting the object we type a variable name incorrectly.

Then Python will inform us that this is an error and that the object does not exist.

Consider the following two examples (**`my_New_variable`**):

```
my_new_variable
```

```
NameError: name 'new_New_variable' is not defined
```

Calling functions

Python has a collection of built-in functions, which can be called as

```
function_name(arg1 = val1, arg2 = val2, ...)
```

Here **arg1**, **arg2**, ... are the names of the arguments (values) that the function takes as input, and **val1**, **val2**, ... are the respective values of these arguments.

```
np.mean(), np.sum(), np.std(), np.sin(), np.round(), np.exp(), np.log(),
np.power(), np.min(), np.max(), np.pi
```

```
np.mean([5,6,9,10])
```

```
7.5
```

```
np.sin(np.pi / 2)
```

```
1
```

```
round(5.3)
```

```
5
```

Example function to generate sequence of numbers

numpy has the built-in function **np.arange()** to generate a sequence of numbers.

Python index starts at 0

Specifically, the following lines would tell Python to generate a list of values between 1 and 10, and to output this list (or vector) to the console

```
np.arange(1, 10)
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

This sequence can also be stored in a variable as a list (vector) by using the assignment operator

```
x = np.arange(1, 10)
x
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Example function to generate sequence of numbers

The `np.arange()` function is far more flexible than what we have seen.

We can choose our start and end numbers arbitrarily

```
np.arange(6, 15)
```

```
array([6, 7, 8, 9, 10, 11, 12, 13, 14])
```

We can choose steps of increments

```
np.arange(1,20,3)
```

```
array([ 1, 4, 7, 10, 13, 16, 19])
```

Example function to generate sequence of numbers

We can also consider negative numbers

```
np.arange(-2,4)
array([-2, -1, 0, 1, 2, 3])
```

Or non-integer start and stop

```
np.arange(2.2,6.2)
array([2.2, 3.2, 4.2, 5.2])
```

Generating sequence using numpy.linspace()

linspace is an in-built function in Python's NumPy library. It is used to create an evenly spaced sequence in a specified interval.

```
numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)
```

```
np.linspace(2,4,20)
```

```
array([2. , 2.10526316, 2.21052632, 2.31578947, 2.42105263, 2.52631579, 2.63157895, 2.73684211, 2.84210526, 2.94736842,
3.05263158, 3.15789474, 3.26315789, 3.36842105, 3.47368421, 3.57894737, 3.68421053, 3.78947368, 3.89473684, 4. ])
```

Generate Random Numbers

```
from random import random, shuffle

# generate some random numbers
print(random(), random(), random())


# shuffle sequence
x= np.arange(1,10)
shuffle(x)


# generate set of random numbers
rand_list=[]
for no in range(10):
    rand_list.append(random())
rand_list
```

```
list.pop()

list.remove()
```

```
[0.7383984161083313, 0.7796939904150217, 0.3482968526250293, 0.03502745863079937, 0.014195704645138085, 0.5066095897033329, 0.548007980651612,
0.12500202260471083, 0.6309464565332288, 0.05211084885433437]
```

Operations on arrays

```
21 x =np.arange(-5, 4, )
22
23 print("x:", x)
24 print("len(x):",len(x))
25 print("np.size(x):", np.size(x))
26 print("max(x):", max(x))
27 print("min(x):", min(x))
28 print("np.mean(x):", np.mean(x))
29 print("np.median(x):", np.median(x))
30 print("np.abs(x):", np.abs(x))
31 print("np.sum(x):", np.sum(x))
32 print("np.cumsum(x): ",np.cumsum(x))
```

```
x: [-5 -4 -3 -2 -1  0  1  2  3]
len(x): 9
np.size(x): 9
max(x): 3
min(x): -5
np.mean(x): -1.0
np.median(x): -1.0
np.abs(x): [5 4 3 2 1 0 1 2 3]
np.sum(x): -9
np.cumsum(x):  [ -5  -9 -12 -14 -15 -15 -14 -12  -9]
```

How to get help about a function in Jupyter Notebook

Being in a code cell, just place the cursor on the Python function in question and press shift-Tab .

```
In [275]:  1 np.mean(x)

Out[275]:
Signature: np.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>)
Docstring:
Compute the arithmetic mean along the specified axis.
```

Data visualization in Python

So far we have discussed computations in Python, as well as ways in which to use functions (with the example `np.arange()` function).

With little knowledge, we can begin making beautiful and informative data visualizations provided that we have data tidied into a nice and convenient format known as a **data frame**.

We will use `matplotlib` and `seaborn` to make beautiful plots.

We can load those libraries the importing them as follows:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

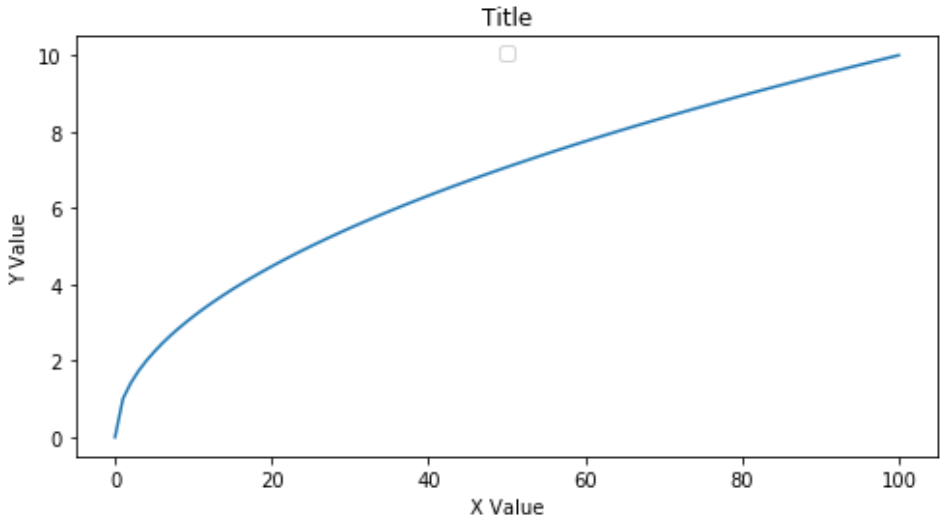
Line Plots

A line plot is a graph that models the relationship between two variables where one variable is plotted on the x-axis and the other on the y-axis.

```
import matplotlib.pyplot as plt
```

Line Plots:

```
plt.plot(x,y)
```

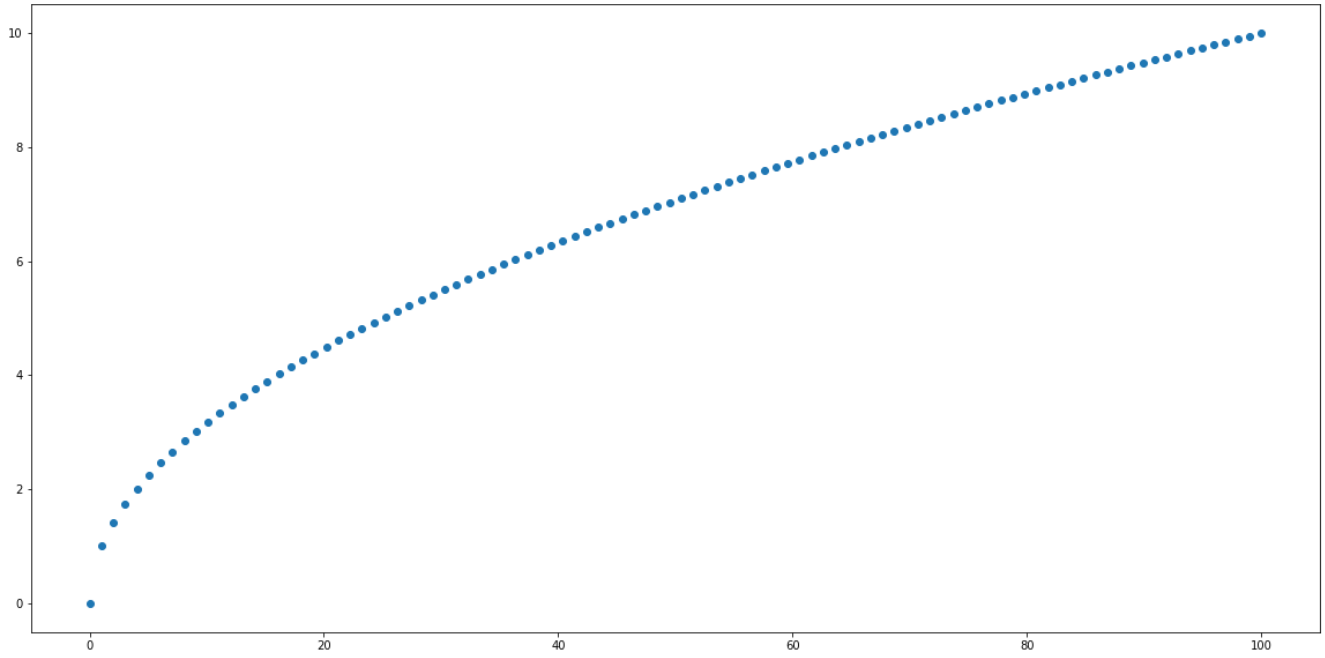


Scatter Plot

A scatterplot shows the relationship between two quantitative variables measured for the same individuals. The values of one variable appear on the horizontal axis, and the values of the other variable appear on the vertical axis.

Scatter Plots:

```
plt.scatter(x,y)
```

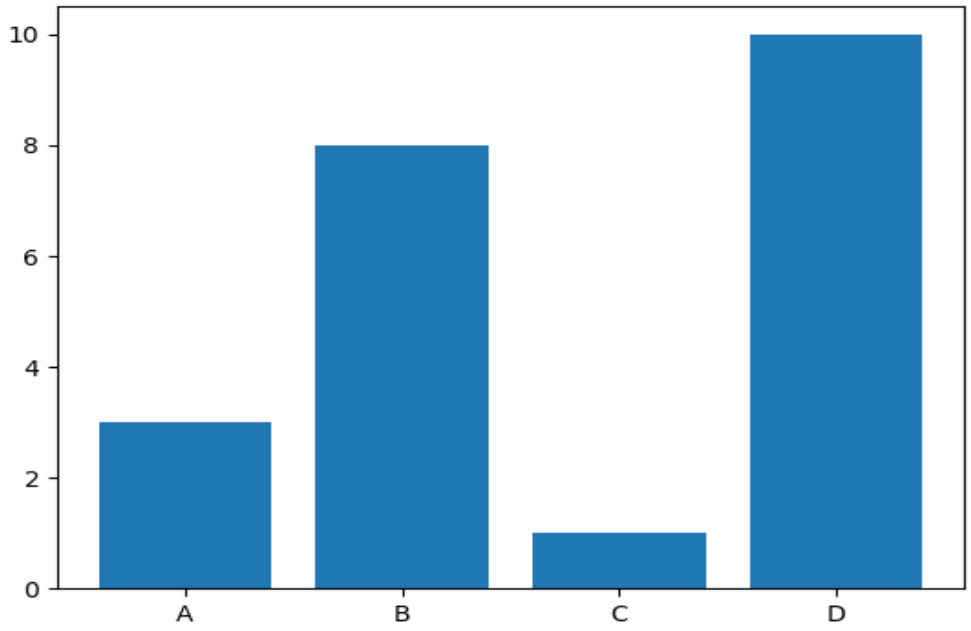


Bar Plot

A barplot (or barchart) is one of the most common types of graphic. It shows the relationship between a numeric and a categoric variable.

Bar Plots:

```
plt.bar(x,y)
```



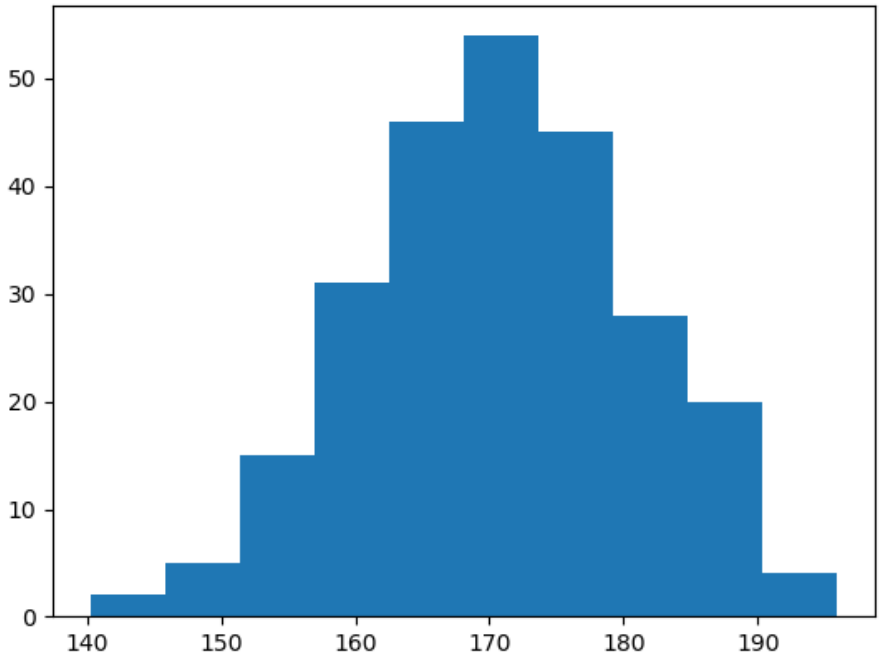
Histogram

A histogram is a bar graph-like representation of data that buckets a range of classes into columns along the horizontal x-axis.

The vertical y-axis represents the number count or percentage of occurrences in the data for each column.

Histograms:

```
plt.hist(x)
```



The `mpg` data frame

We will initially examine the **`mpg`** data that is available to download and use in Canvas under the Data module

A **data frame** is a data structure constructed with rows and columns, similar to a database or Excel spreadsheet (tabular format)

The mpg data frame contains 234 observations (rows) on 11 features (columns).

These observations are collected by the US Environmental Protection Agency on 38 models of cars.

Load and view the **mpg** data frame

```
Import pandas as pd

mpg = pd.read_csv("file_location\filename.csv")

mpg
```

```
In [277]: 1 mpg
```

Out[277]:

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	class
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
5	audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
6	audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
7	audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
8	audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
9	audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact
10	audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact
11	audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact
12	audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact
13	audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact
14	audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact
15	audi	a6 quattro	2.8	1999	6	auto(l5)	4	15	24	p	midsize
16	audi	a6 quattro	3.1	2008	6	auto(s6)	4	17	25	p	midsize
17	audi	a6 quattro	4.2	2008	8	auto(s6)	4	16	23	p	midsize
18	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14	20	r	suv
19	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	11	15	e	suv
20	chevrolet	c1500 suburban 2wd	5.3	2008	8	auto(l4)	r	14	20	r	suv

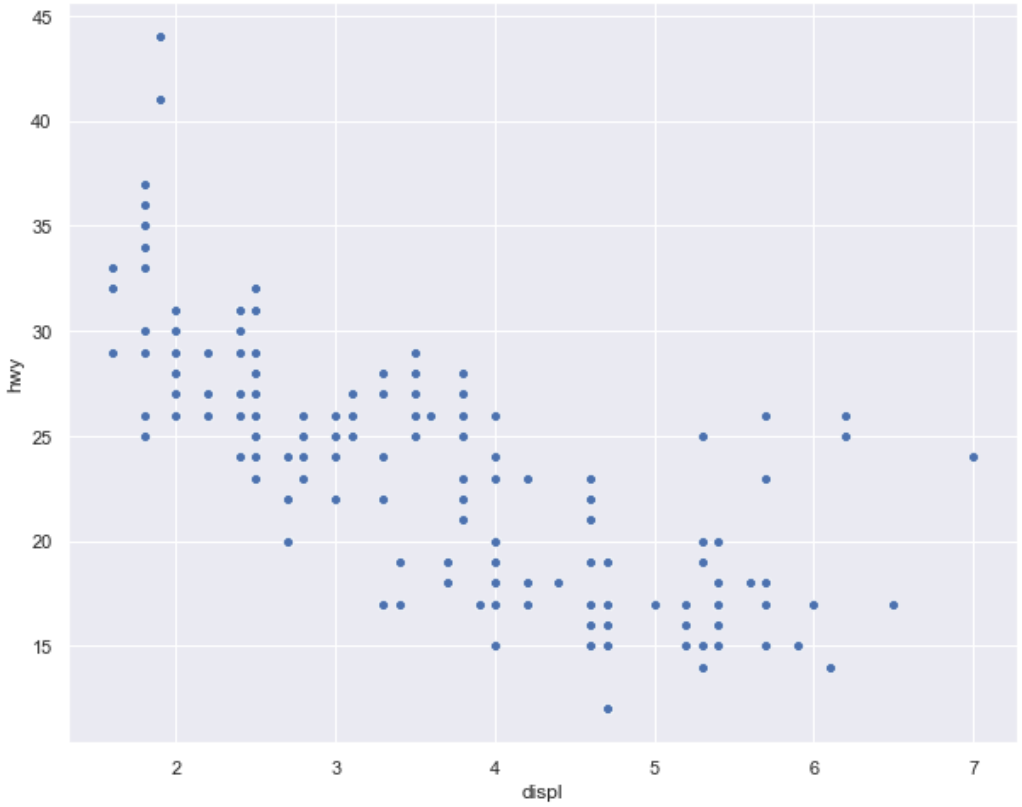
Learning about the `mpg` data frame

`manufacturer`	Car manufacturer
`model`	Model name
`displ`	Engine displacement (in liters)
`year`	Year of manufacture
`cyl`	Number of cylinders
`trans`	Type of transmission
`drv`	**`f`**=front-wheel, **`r`**=rear wheel, **`4`**=four wheel drive
`cty`	City miles per gallon
`hwy`	Highway miles per gallon
`fl`	Fuel type
`class`	Type of car

Creating a plot in **seaborn**

From the mpg data, we can examine the relationship of highway miles per gallon as a function of engine displacement using the command

```
import matplotlib.pyplot as plt

import seaborn as sns

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

plt.show()
```



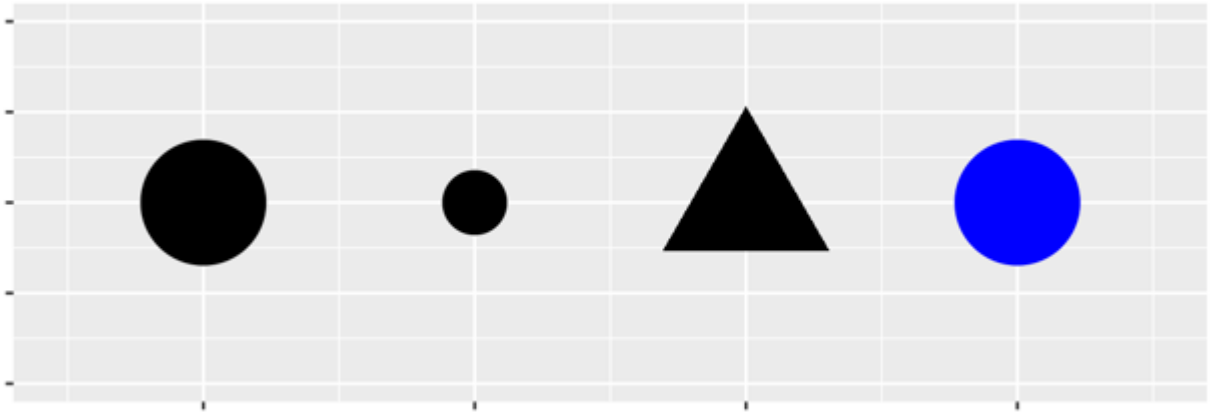
Aesthetics in `seaborn`

We can change is a visual property of objects in a plot.

Example aesthetics include size, shape, and color of points.

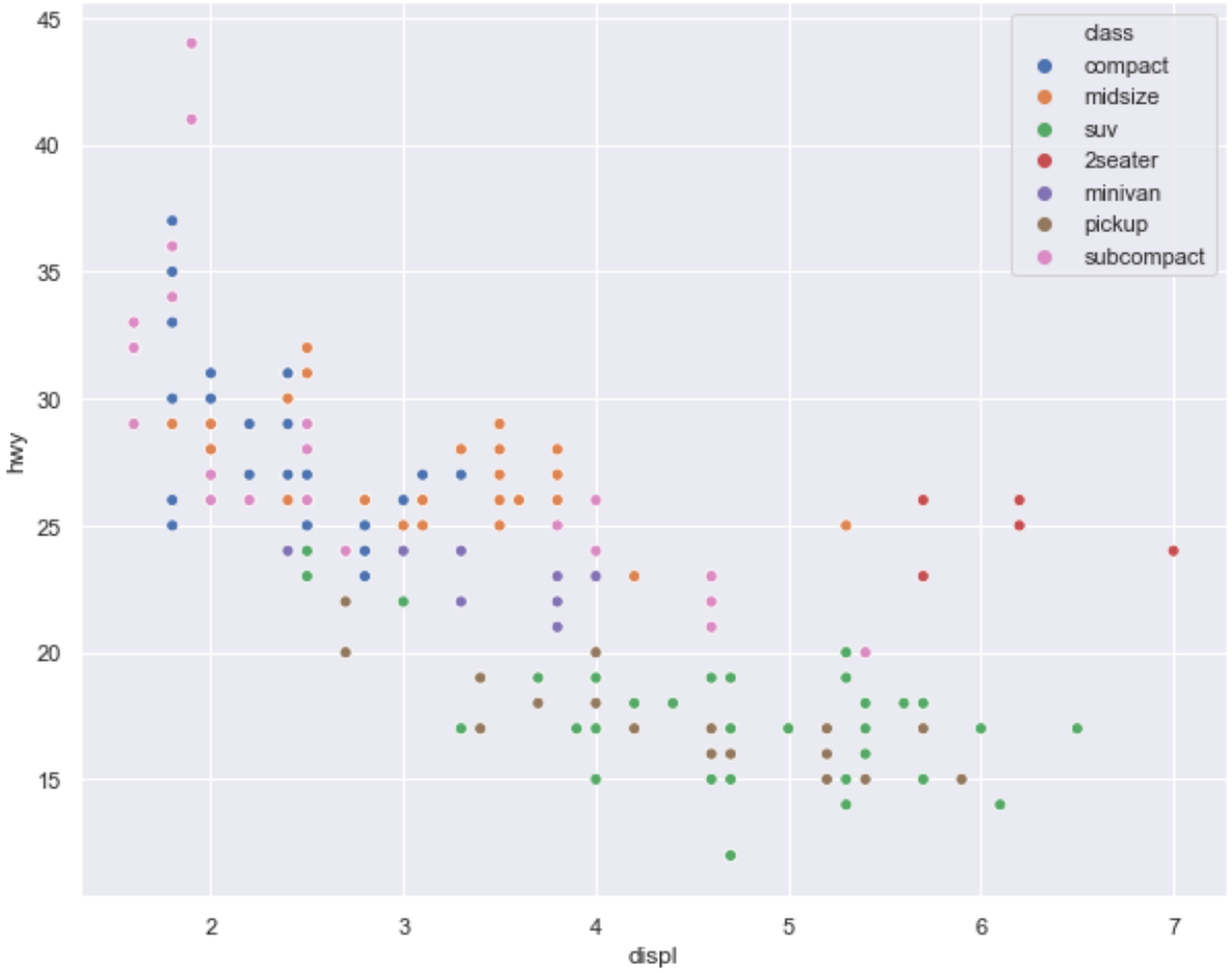
A point can be displayed in different ways by changing the values of its aesthetic properties.

Here, we can change the point's size, shape, and color to small, triangular, or blue, respectively.



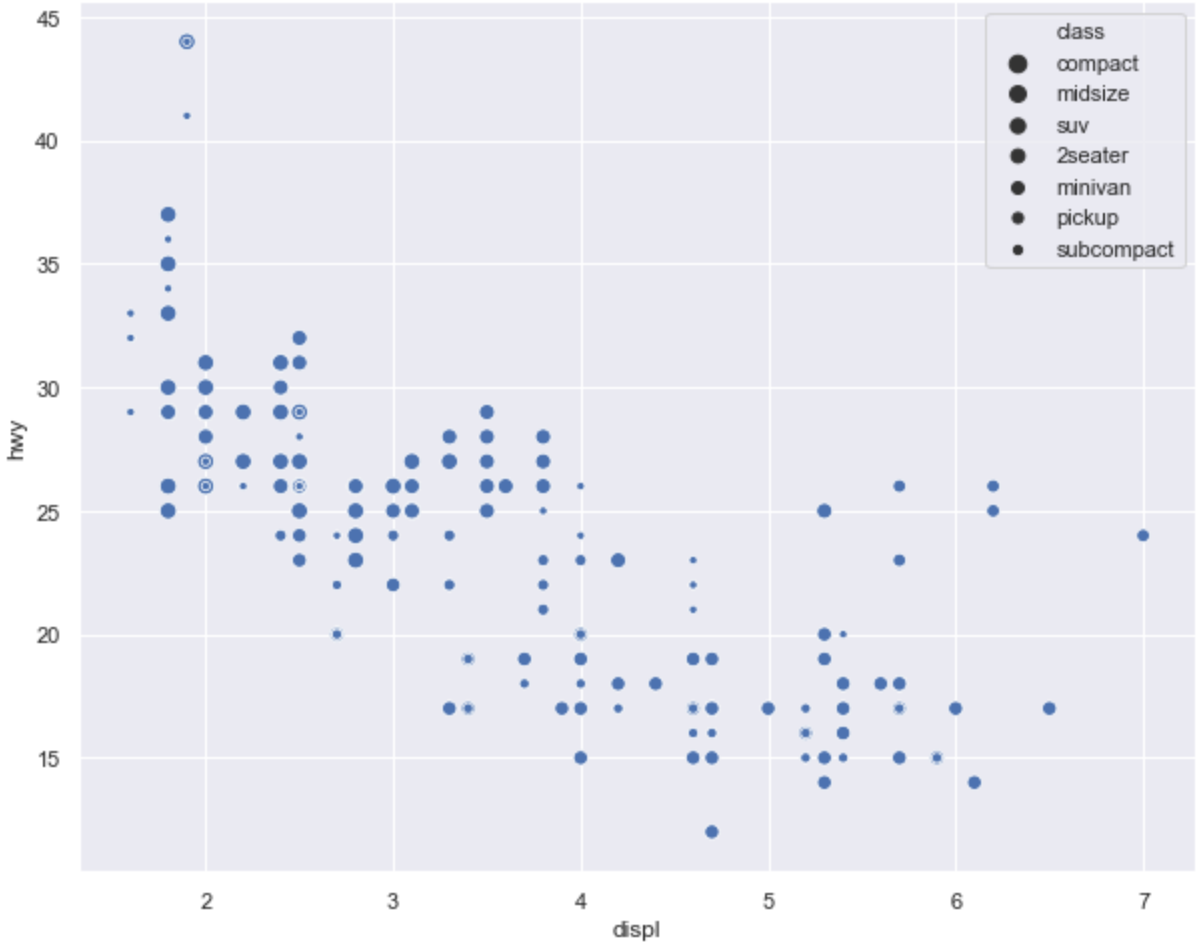
Assigning different colors based on car type/class

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



Assigning different sizes based on car type

```
sns.scatterplot(data=mpg, x='displ', y='hwy', size='class',
legend='full')

plt.show()
```

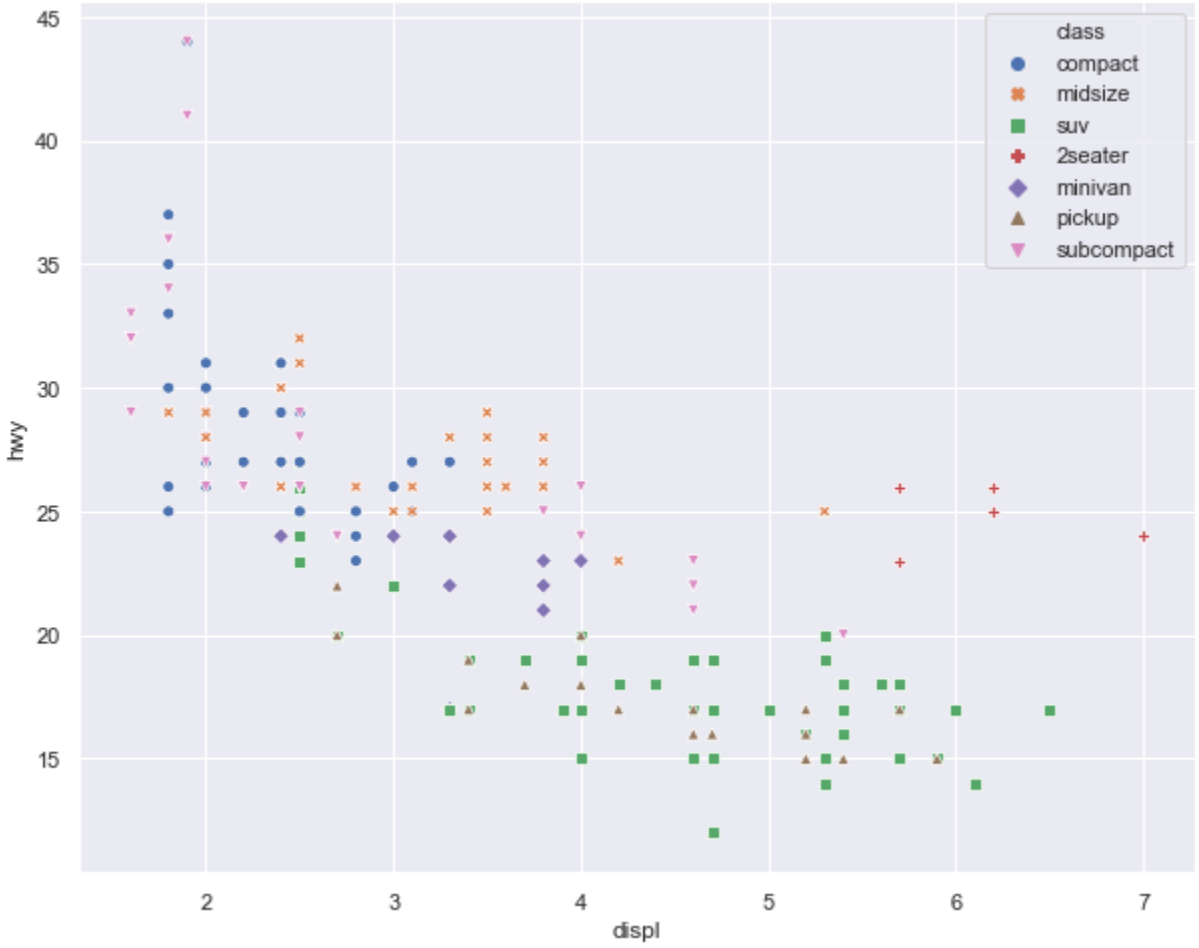


Assigning different transparencies based on car type

```
sns.scatterplot(data=mpg, x='displ', y='hwy', hue='class',
alpha=0.5)

plt.show()
```

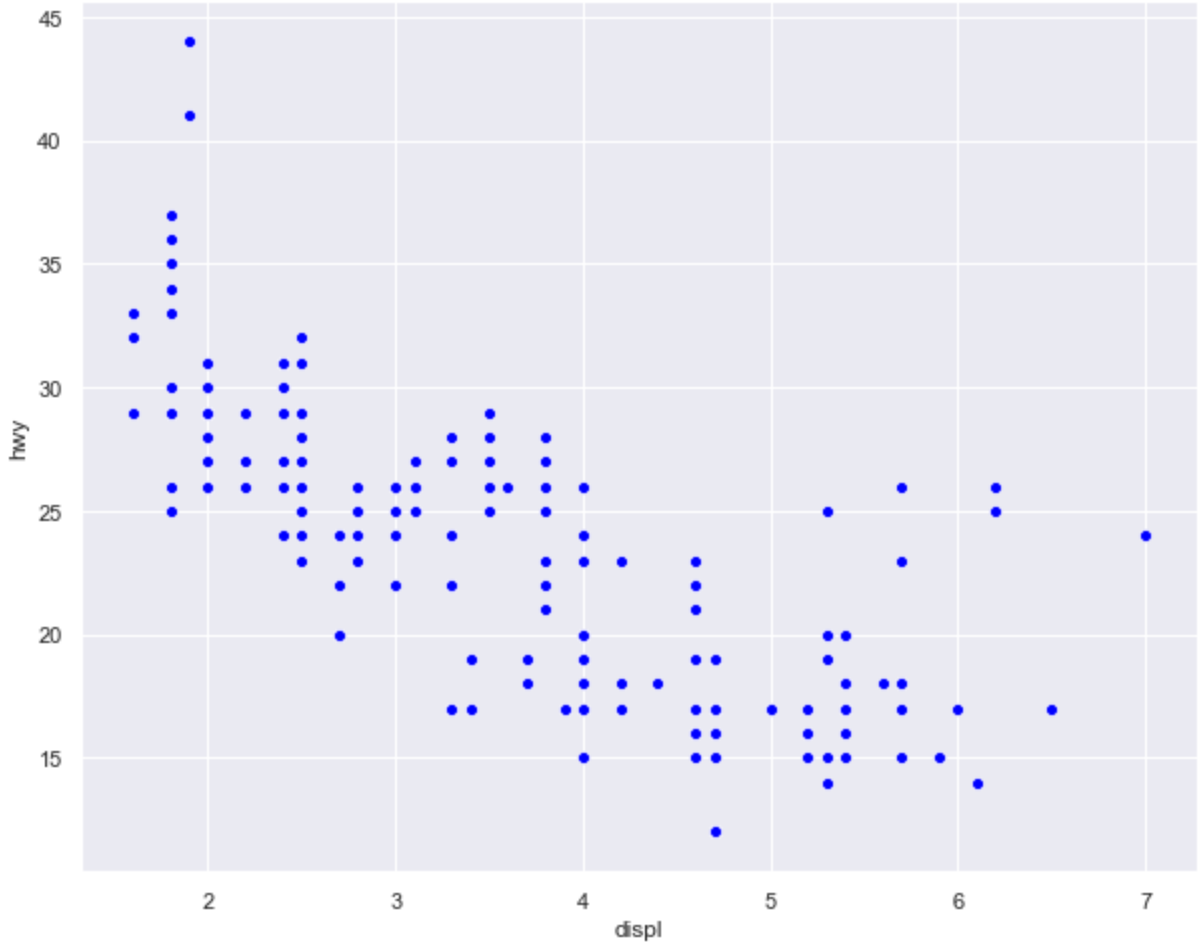
Assigning different shapes based on car type

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

plt.show()
```



Assigning same color to all points

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

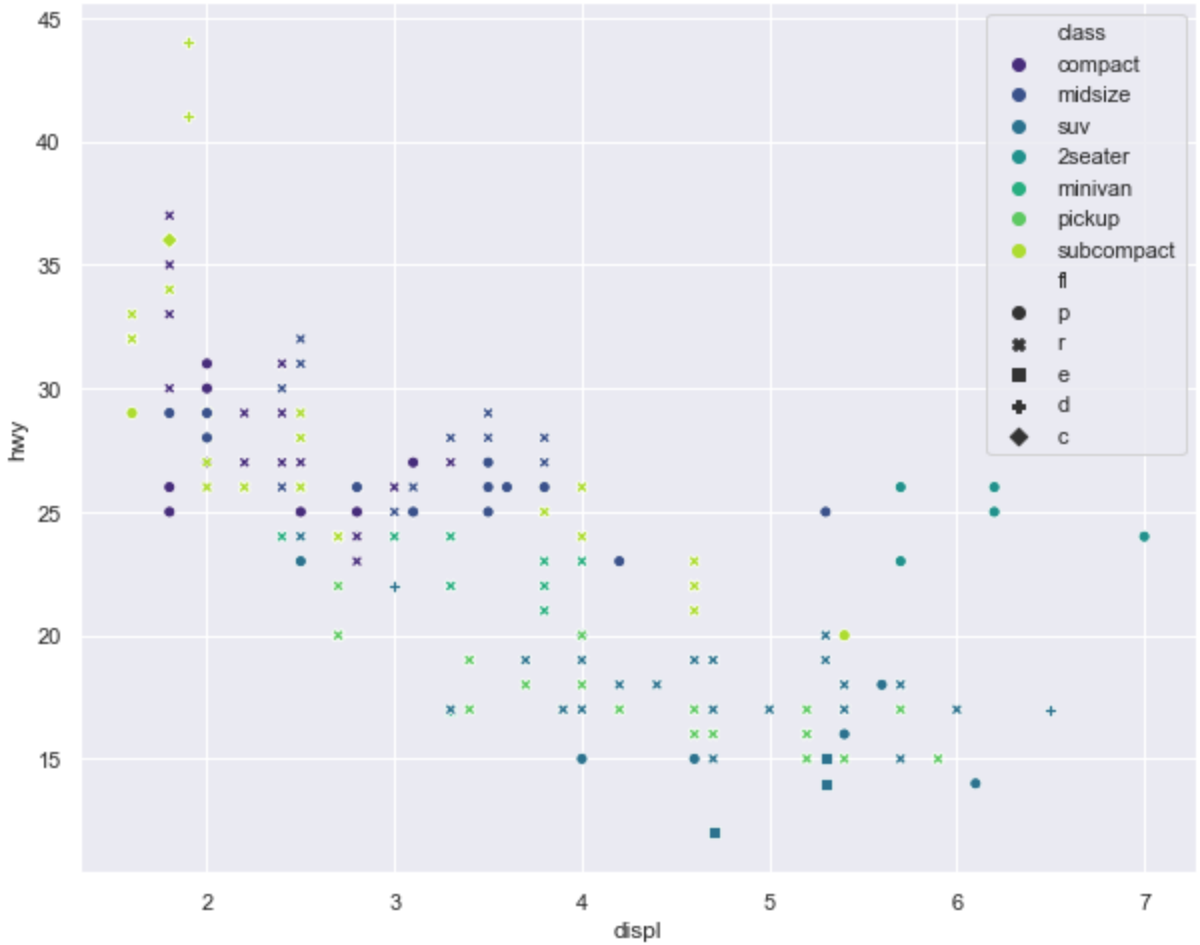


Another aesthetic example

```
sns.scatterplot(x='displ', y='hwy', data=mpg,
                marker='^', s=80, alpha=0.5, color='red',
                edgecolor='red')

plt.show()

# S is the size of the markers
```

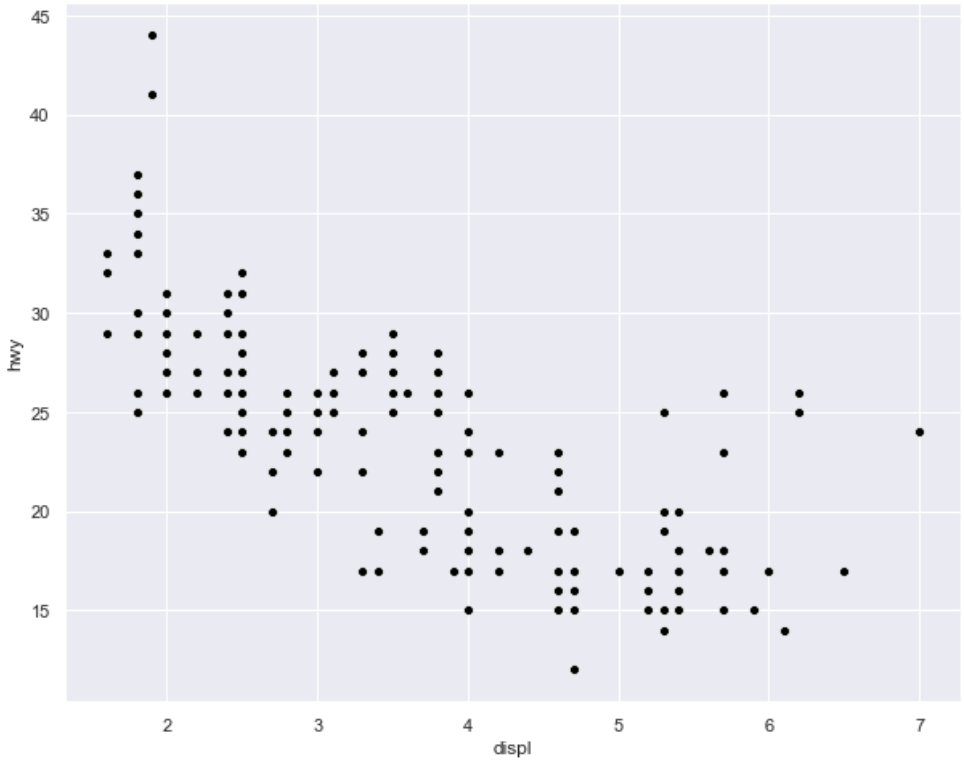
Mapping multiple aesthetics (fuel type and car type)

```
sns.scatterplot(x='displ', y='hwy', data=mpg,
                hue='class', style='fl', palette='viridis',
markers=True)

plt.show()
```



Dealing with overlapping points

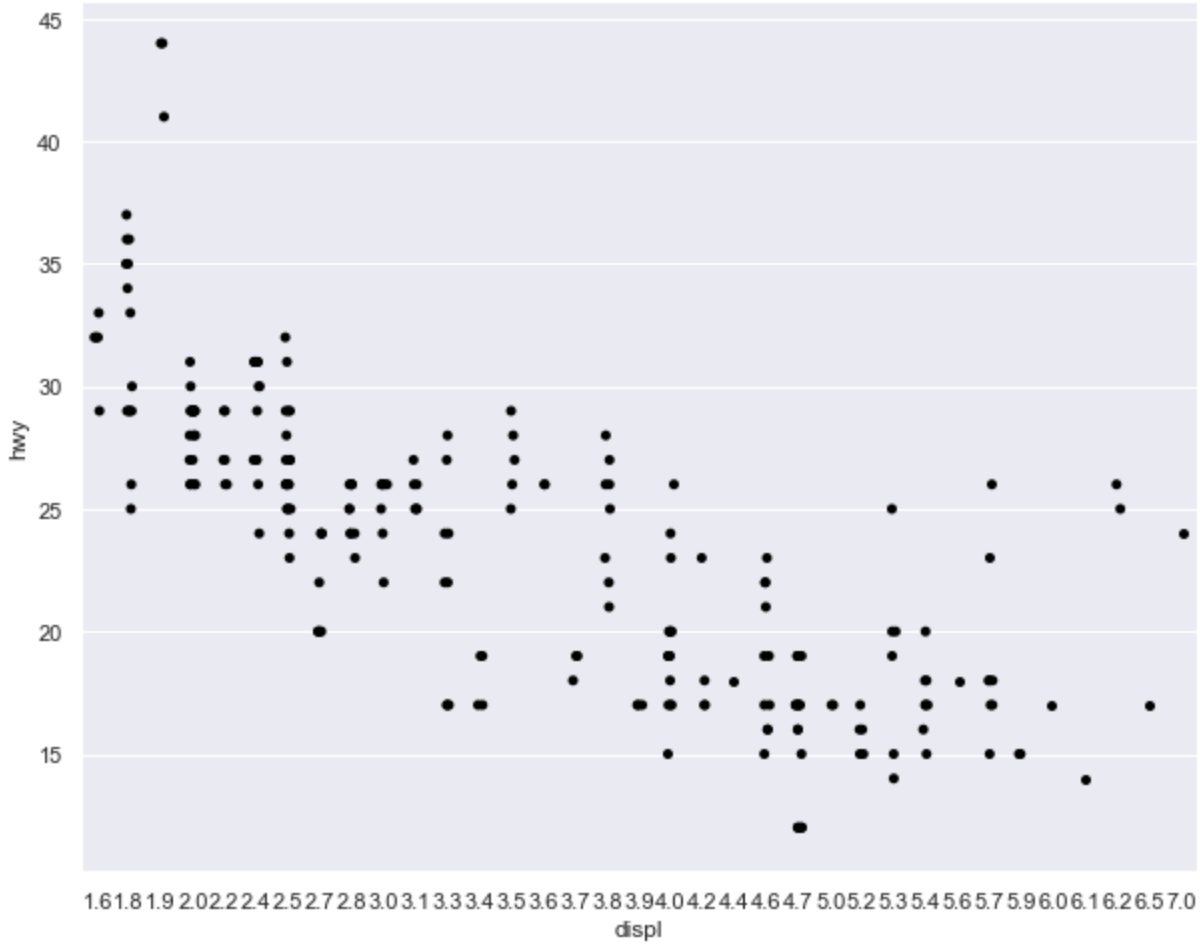
The **`mpg`** dataset has 235 observations, yet only 126 points exist in the following plot.



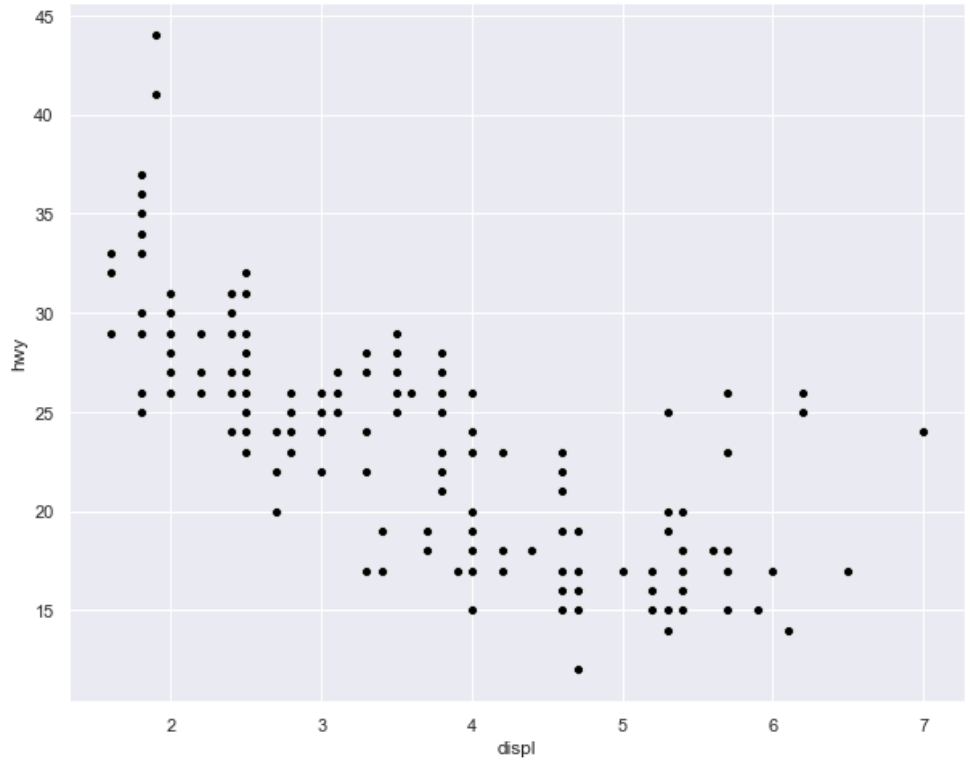
This problem is known as **overplotting**, and occurs here because the values of **`displ`** are rounded to one decimal place and the values of **`hwy`** are rounded to the nearest integer.

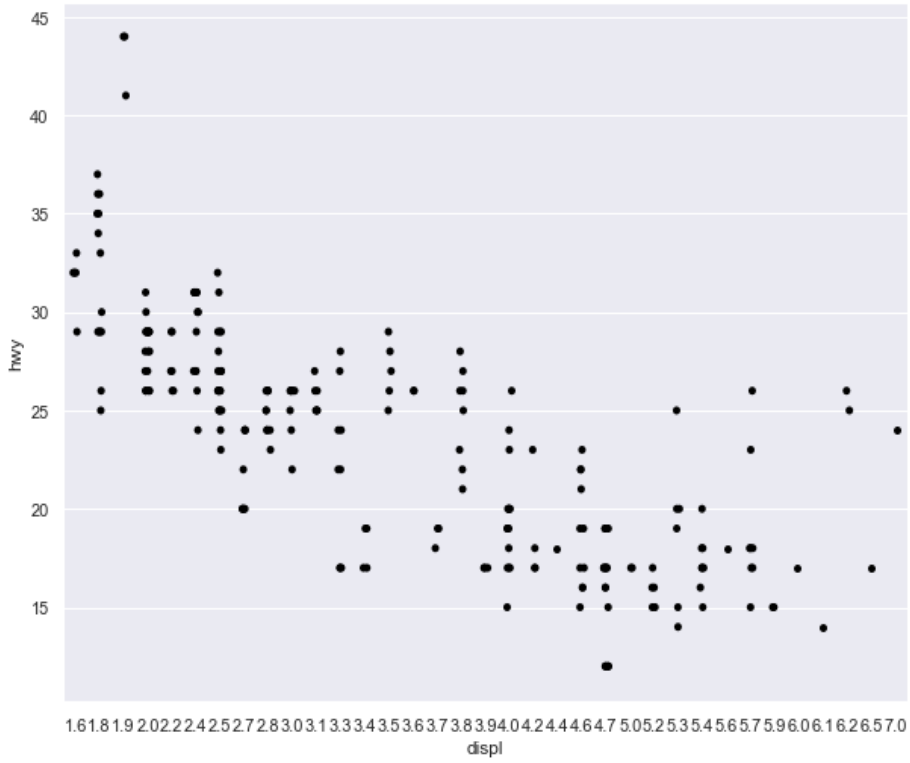
Dealing with overlapping points through **`stripplot()` and `jitter`**

```
sns.stripplot(x='displ', y='hwy', data=mpg, jitter=True,
color='black')

plt.show()
```



Dealing with overlapping points through `jitter`





Here, `jitter` is telling `stripplot()` to add some random noise to the x- and y-axis values of each point prior to it being plotted.

Facets

Aesthetics have been useful for visualizing sub-divided data so that it is possible to visualize if there are patterns in subsets of data.

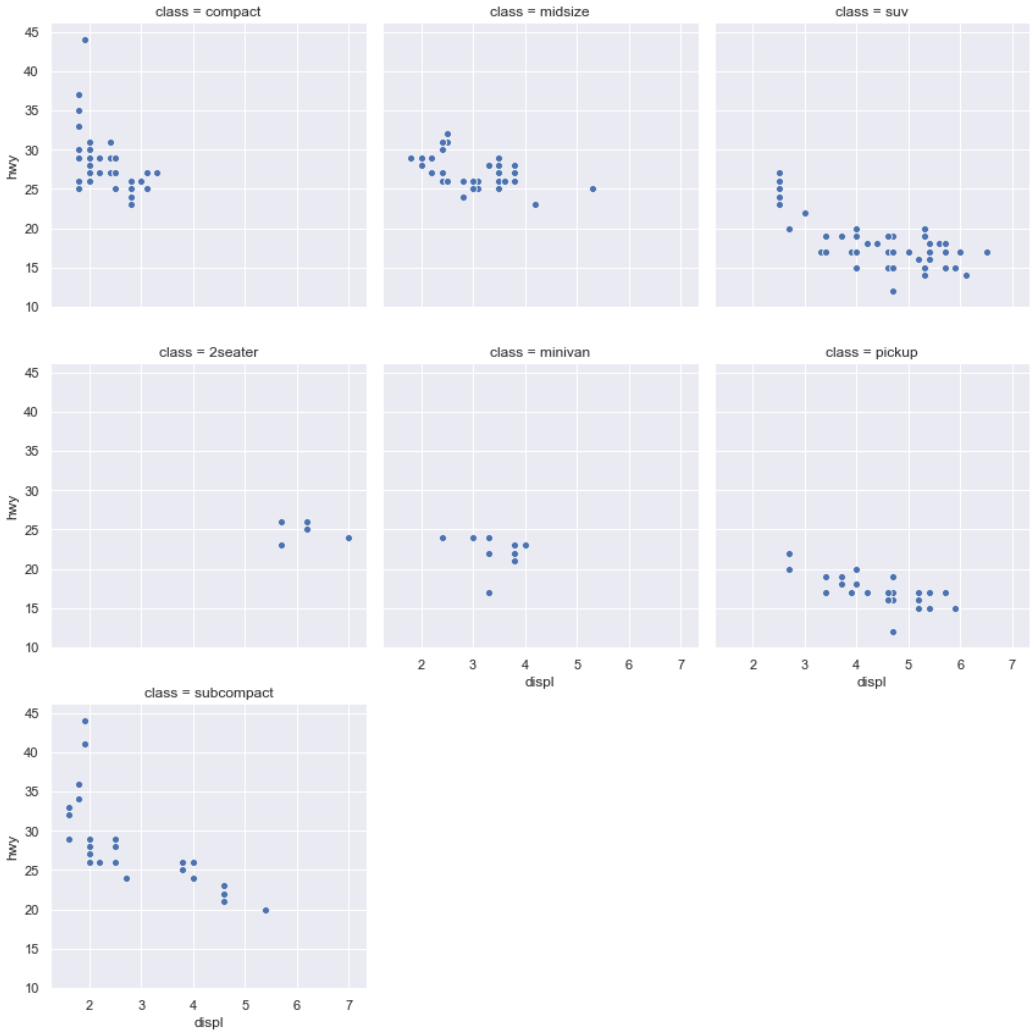
Another way to examine such patterns is to use subplots (or **facets**), where each subplot is a subset of the data.

Facets are particularly useful for categorical data, where each subplot represents a sensible subdivision of the data.

To facet (or subdivide) a plot by a single variable (*e.g.*, car type), then we can use the `facet_wrap()` function, where the first argument of the function is a formula, which is created using the tilde (~) character.

Splitting data based on car type across three columns

```
sns.relplot(x='displ', y='hwy', data=mpg, col='class',
col_wrap=3, height=4)

plt.show()
```



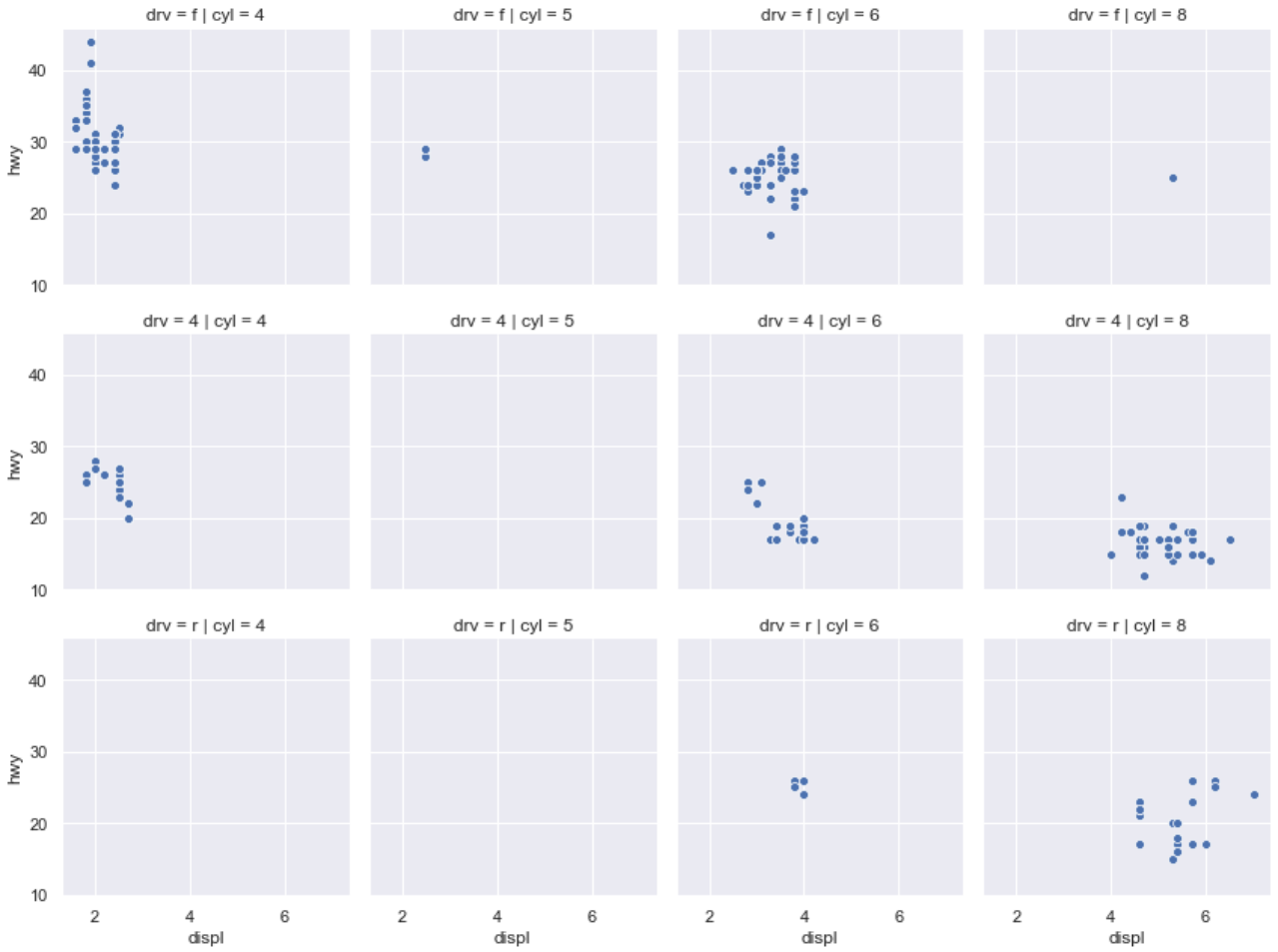
Faceting data across a combination of two features

So far we have considered color to visual facets, which allows us to split data based on a single feature.

However, it can sometimes be useful to examine two features at once.

Splitting data based on drive and number of cylinders

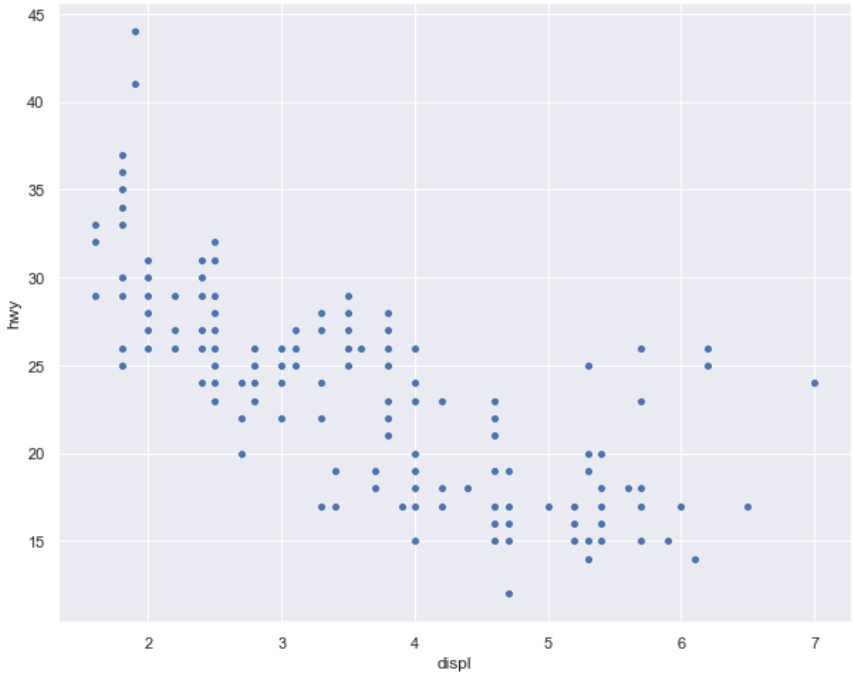
```
sns.relplot(x='displ', y='hwy', data=mpg, col='cyl',
row='drv', height=3)

plt.show()
```

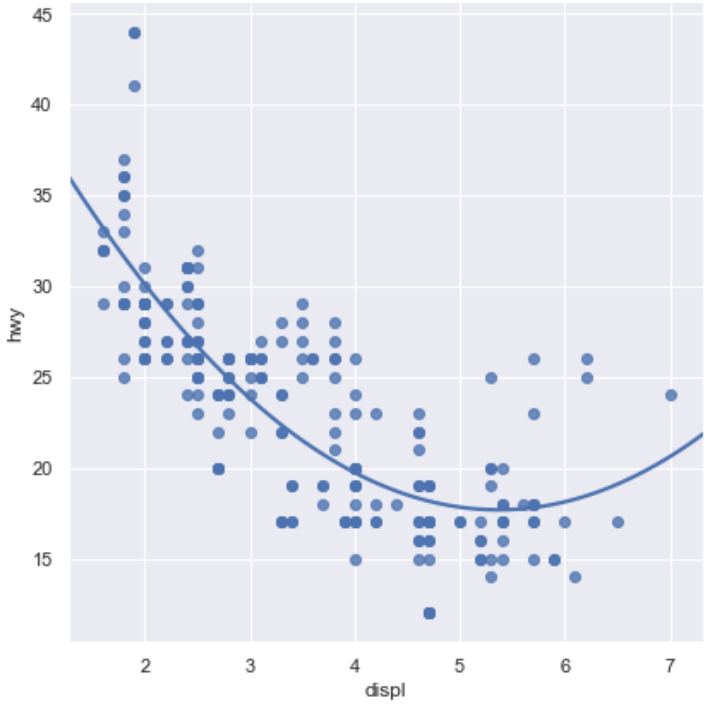


Adding different types of geometric objects

We have considered the functionality of **`seaborn`** to make a scatter plot of two features, one on the x- and one on the y- axis.

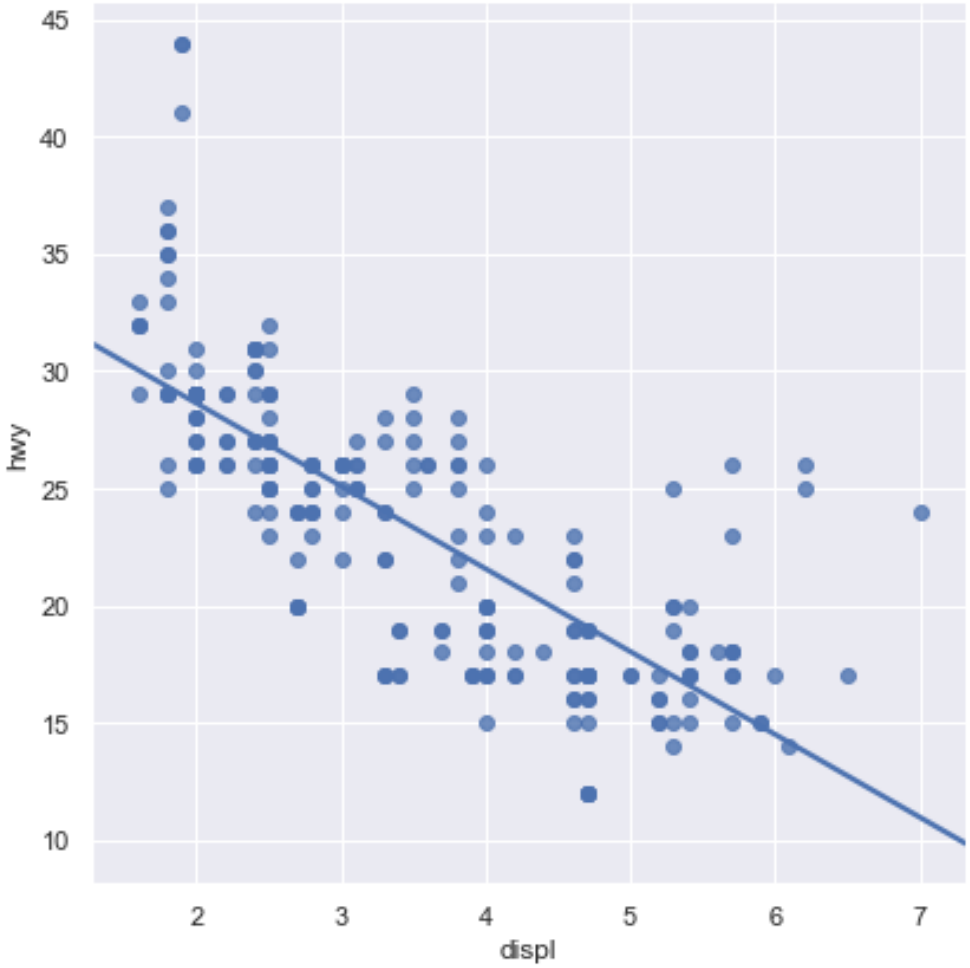
Consider the following plots, which are two representations of the same dataset, but using different geometrical objects (**lm**) to represent the data.





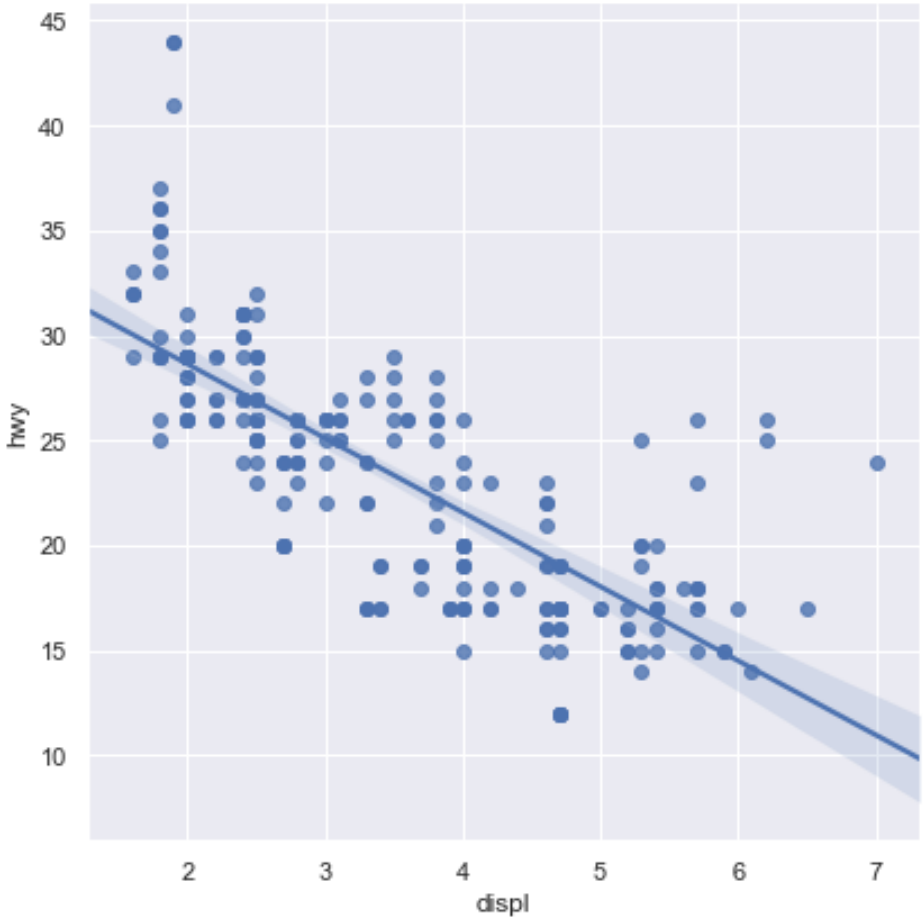
Plotting a curve fitted to the data

```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=None, height=6,
order=1)

plt.show()
```



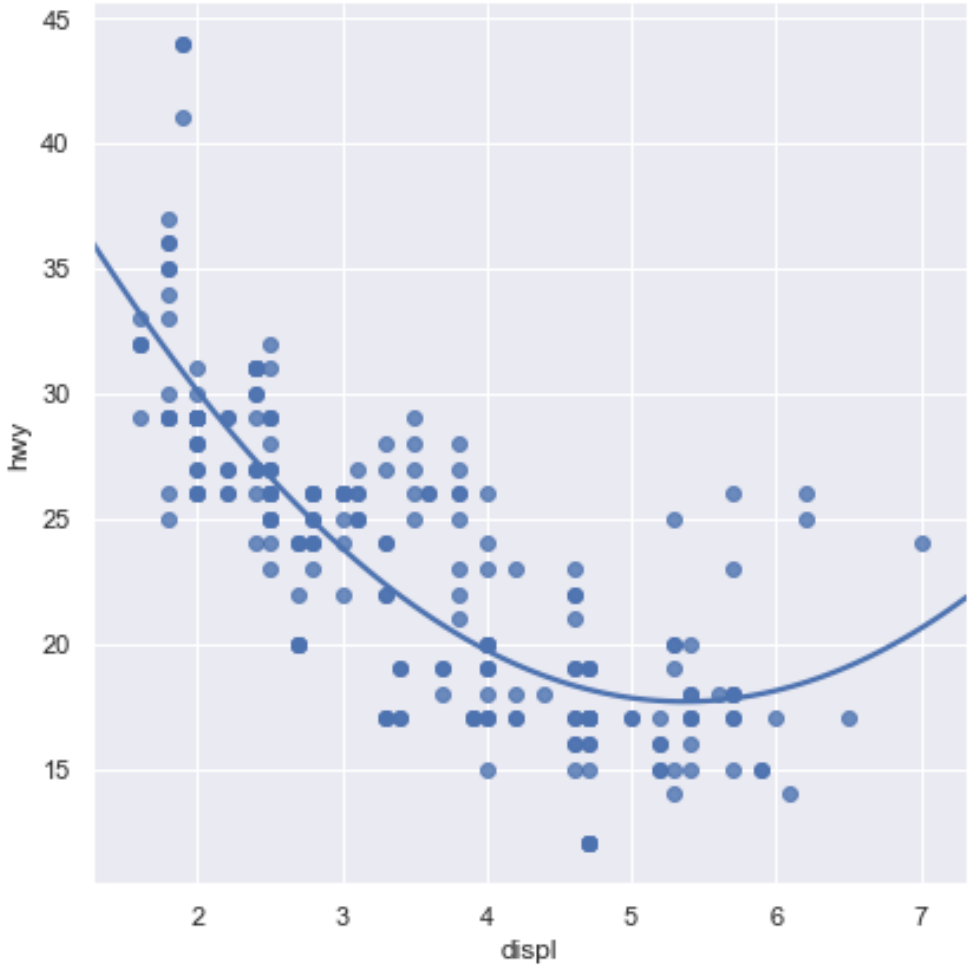
Plotting a curve fitted to the data

```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=95, height=6,
order=1)

plt.show()
```



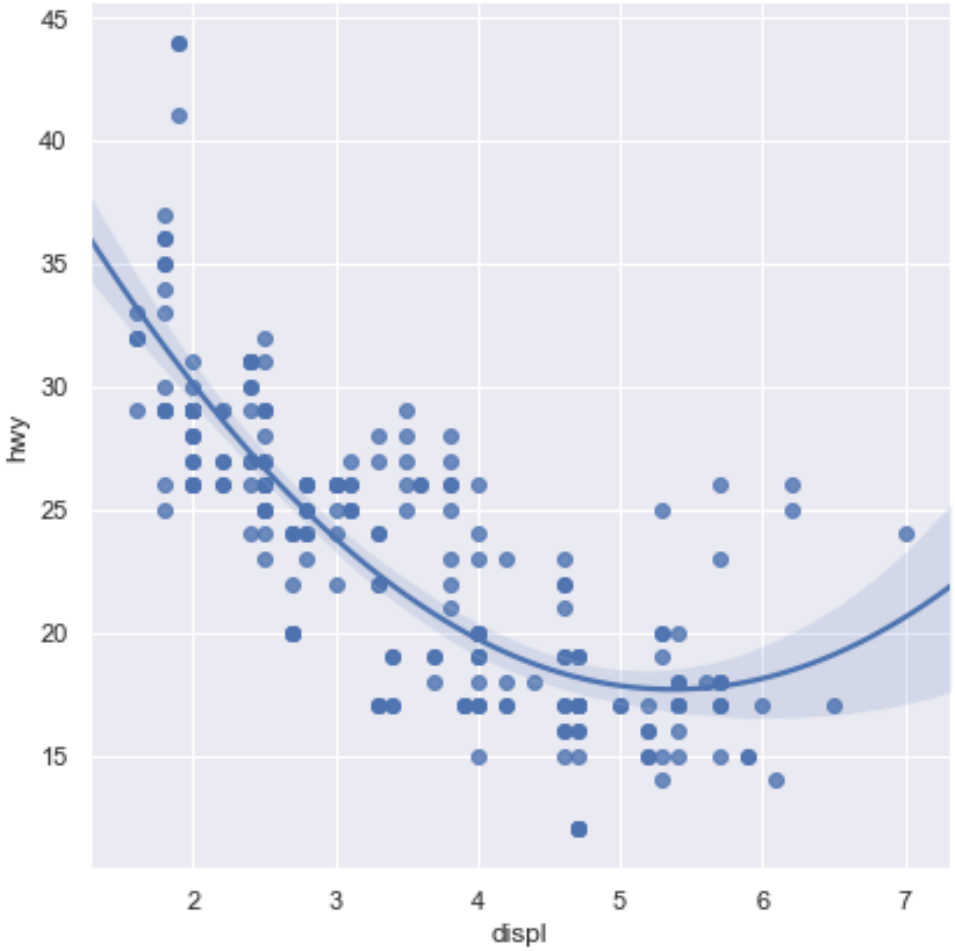
Plotting a curve fitted to the data

```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=None, height=6,
order=2)

plt.show()
```



Plotting a curve fitted to the data

```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=95, height=6,
order=2)

plt.show()
```



What `lmplot()` is plotting

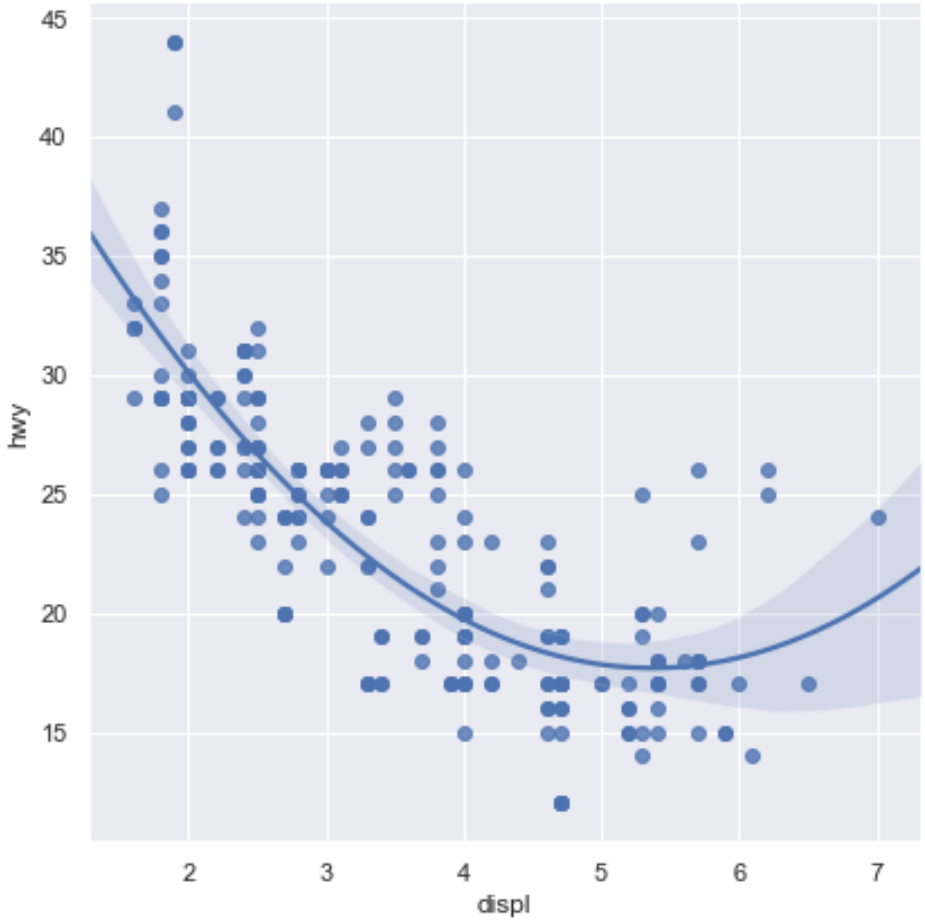
We told seaborn to plot a smoothed curve and to display 95% confidence intervals (CIs) by default about the smoothed line.

This confidence interval means that there is a 95% percent chance that the confidence interval that was calculated contains the true population mean (here, the fitted curve).

There are a number of arguments for the `lmlplot()` function that will permit flexibility, such as changing the type of confidence interval, not including the confidence interval, or even having separate smoothing lines and confidence intervals for subsets of the data.

Changing to a 99% confidence interval

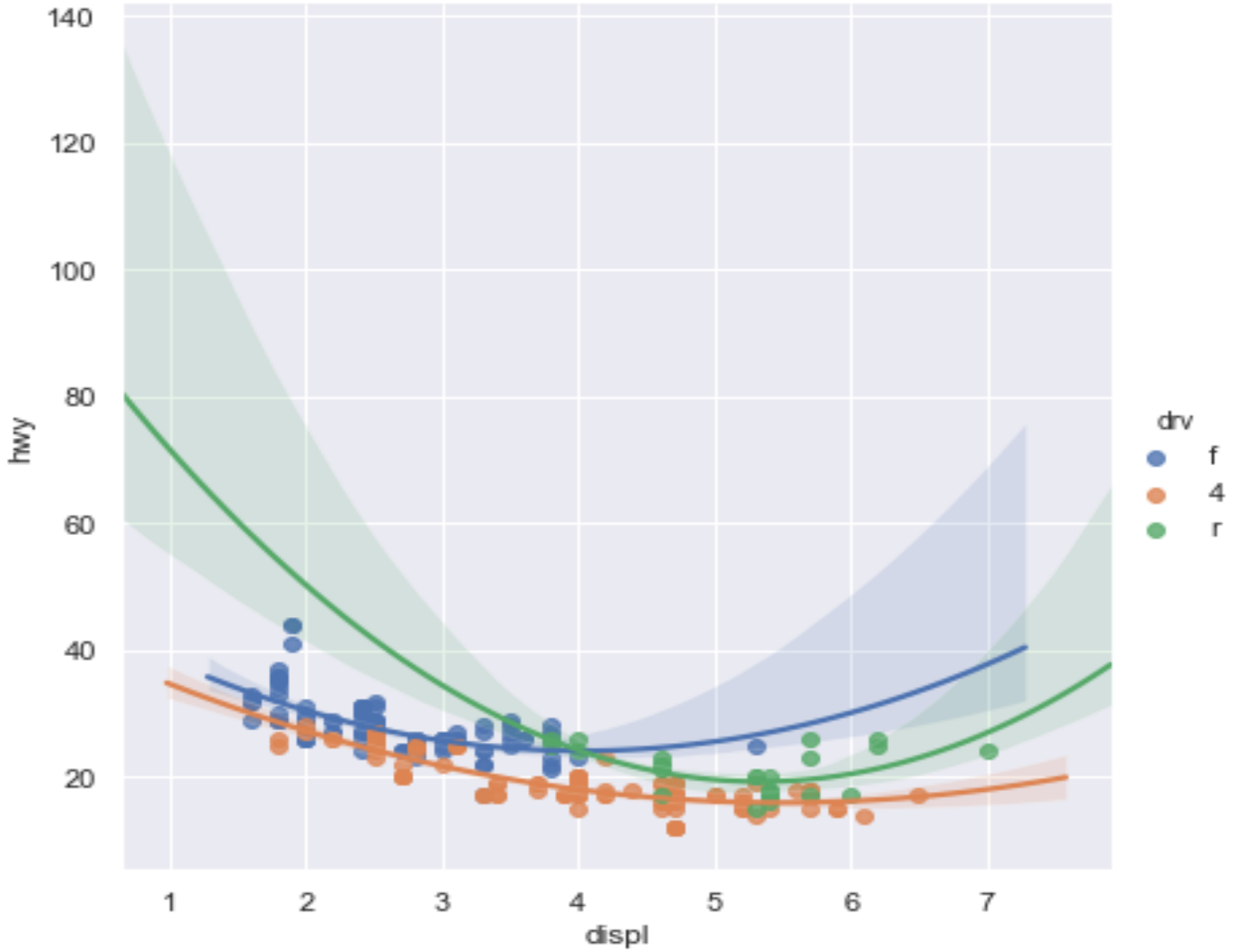
```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=99, height=6,
order=2)

plt.show()
```



It is difficult to see, but the bands around the curve are wider.

Plot with separate fit and color for each drive type

```
sns.lmplot(x='displ', y='hwy', data=mpg, hue='drv', ci=95,
height=6, order=2)

plt.show()
```



Adding layers of different geom plot types

We have examined two geom plot types so far, which are the `geom_point()` for scatter plots and the `geom_smooth()` for smoothed line fits with confidence bands.

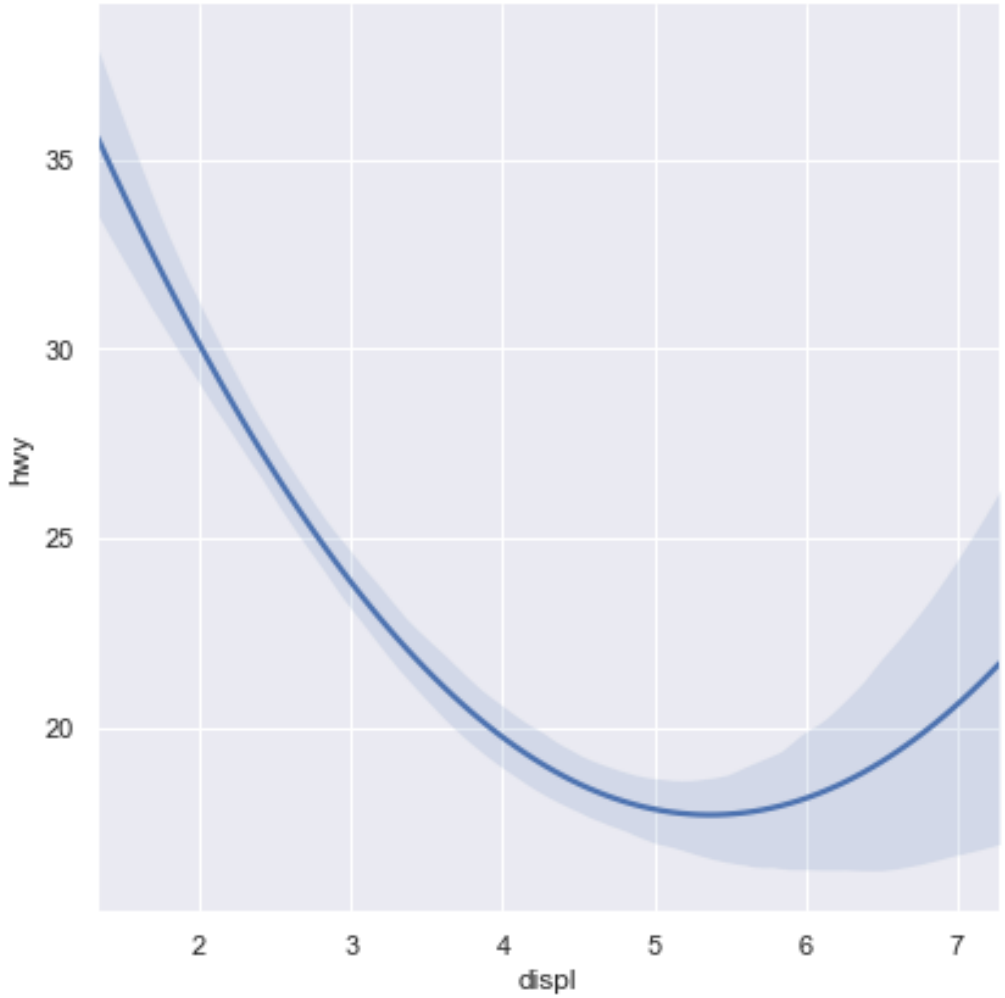
Recall that `ggplot()` simply initiates a blank plot, whereas `geom_point()` and `geom_smooth()` add layers to the plots.

Therefore, we can add multiple layers to the same plot to make arbitrarily complex graphs.

Adding a smoothed fit to a scatter plot permits the visualization of overall trends while being transparent about the observations that the trend is based upon.

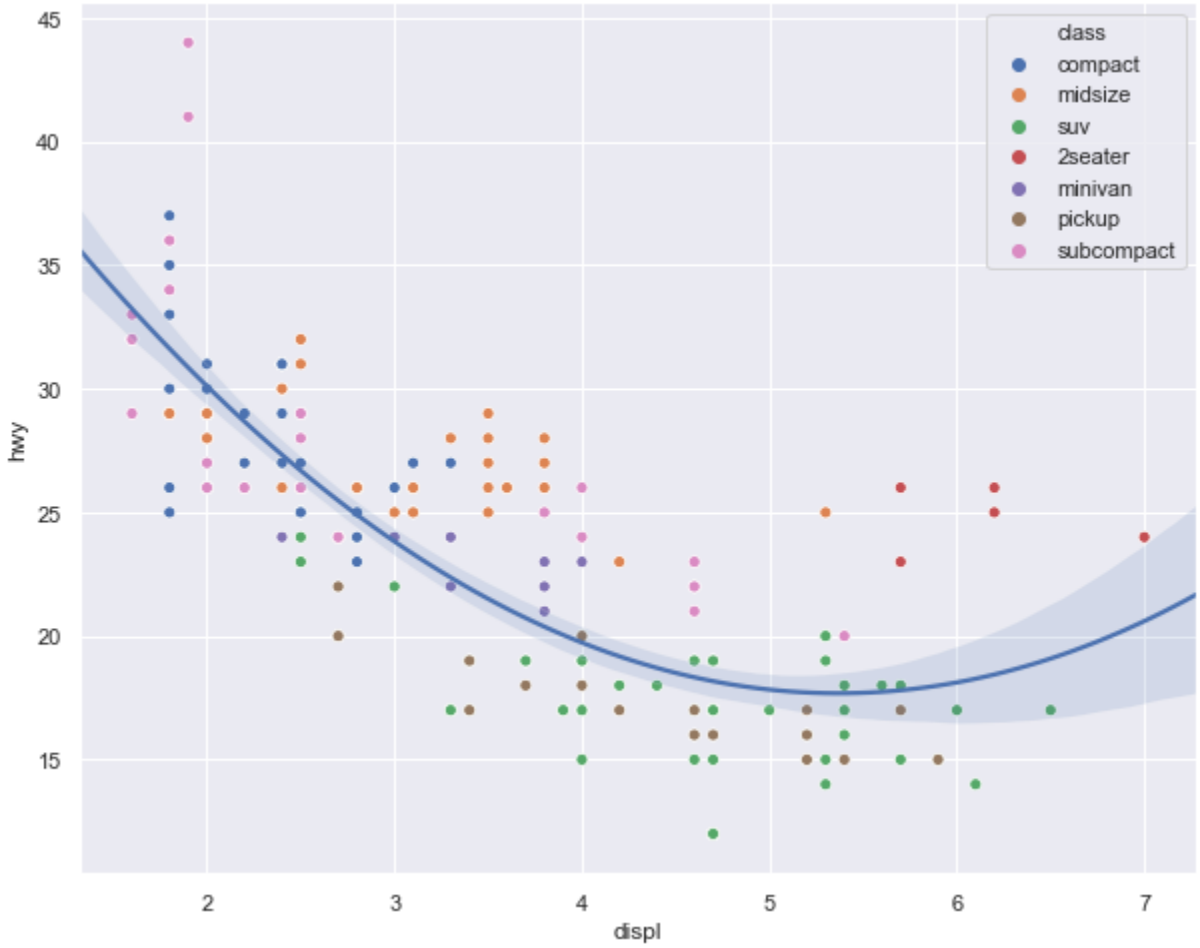
Display fitted curve and without points on same graph

```
sns.lmplot(x='displ', y='hwy', data=mpg, ci=99, height=6,
order=2, scatter=False)

plt.show()
```



Add color to the scatter plot based on car type

```
sns.scatterplot(x='displ', y='hwy', hue='class', data=mpg)
sns.regplot(x='displ', y='hwy', data=mpg, scatter=False, order=2)
plt.show()
```



Visualizing Distributions

Bar charts

Bar charts are generally useful when displaying relative counts or proportions across categorical data (the magnitude of some set of numbers or amounts).

Examples of such categories (e.g., brands of cars, cities, or sports) and a quantitative value for each category.

To explore the use of bar charts, we will examine a new dataset.

Consider a new data frame called **`diamonds`**, which is in Canvas.

This dataset has 53,940 rows (observations) with 10 columns (features) containing data on prices and other attributes of diamonds.

As you can see, this dataset is much larger than the **`mpg`** dataset.

A snapshot of the **diamonds** data frame

Type the following to see a snapshot of the data frame

```
diamonds= pd.read_csv("diamonds.csv")
diamonds
```

```
In [409]:  1 diamonds
```

Out[409]:

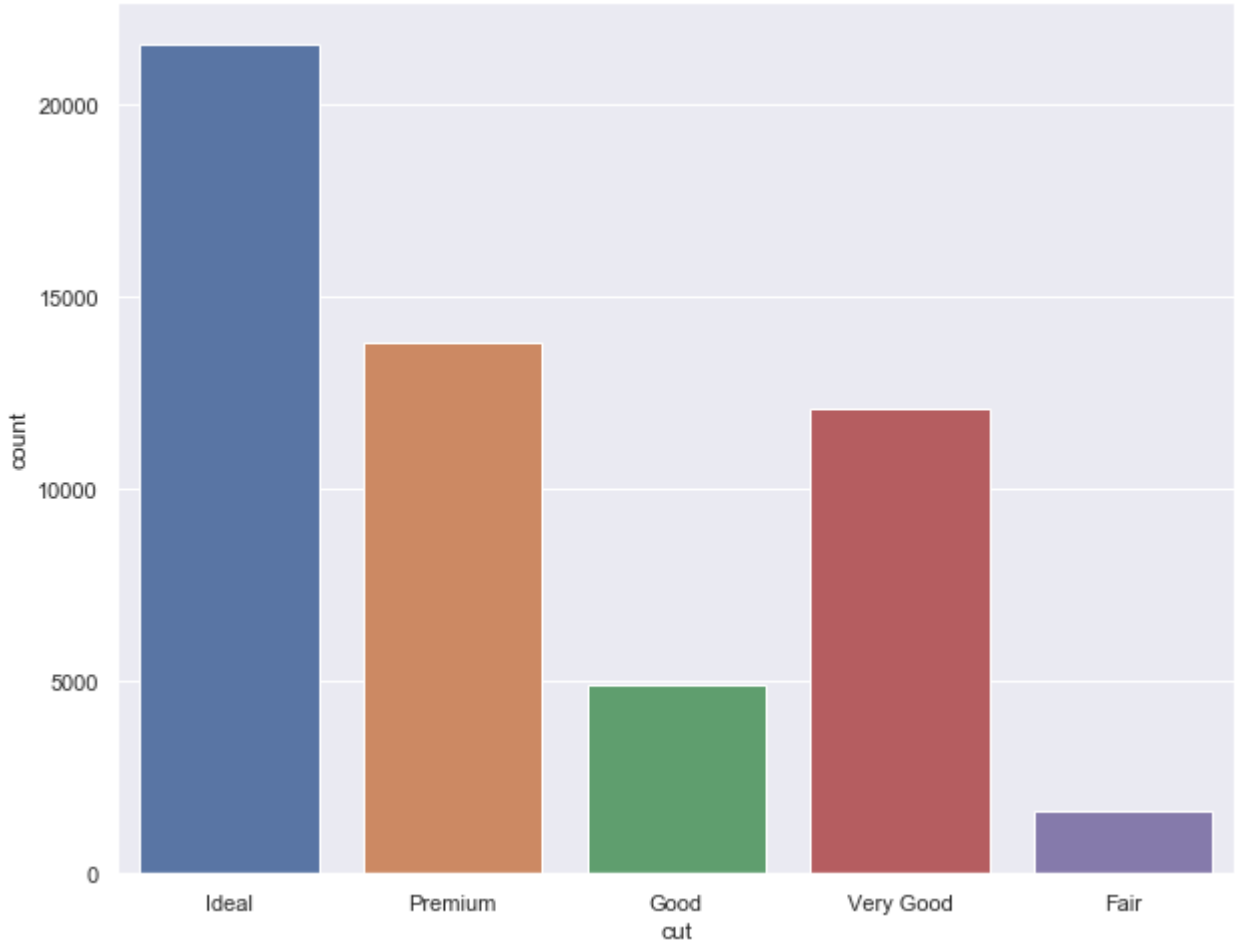
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
5	0.24	Very Good	J	VVS2	62.8	57.0	336	3.94	3.96	2.48
6	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
7	0.26	Very Good	H	SI1	61.9	55.0	337	4.07	4.11	2.53
8	0.22	Fair	E	VS2	65.1	61.0	337	3.87	3.78	2.49
9	0.23	Very Good	H	VS1	59.4	61.0	338	4.00	4.05	2.39
10	0.30	Good	J	SI1	64.0	55.0	339	4.25	4.28	2.73
11	0.23	Ideal	J	VS1	62.8	56.0	340	3.93	3.90	2.46
12	0.22	Premium	F	SI1	60.4	61.0	342	3.88	3.84	2.33
13	0.31	Ideal	J	SI2	62.2	54.0	344	4.35	4.37	2.71
14	0.20	Premium	E	SI2	60.2	62.0	345	3.79	3.75	2.27
15	0.32	Premium	E	I1	60.9	58.0	345	4.38	4.42	2.68
16	0.30	Ideal	I	SI2	62.0	54.0	348	4.31	4.34	2.68
17	0.30	Good	J	SI1	63.4	54.0	351	4.23	4.29	2.70
18	0.30	Good	J	SI1	63.8	56.0	351	4.23	4.26	2.71

Learning about the `diamonds` data frame

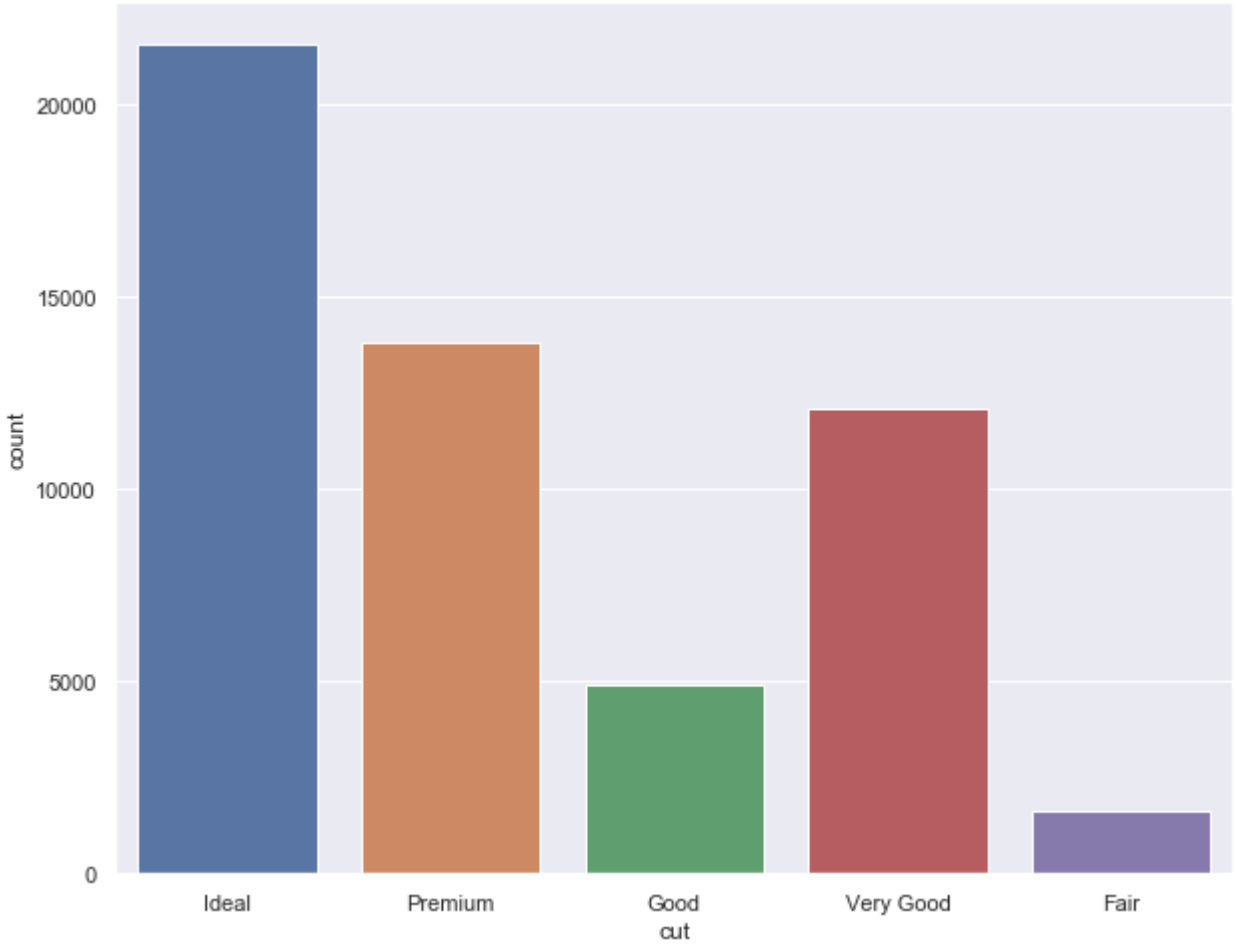
`price`	Price in US dollars ($326-$18,823)
`carat`	Weight of the diamond (0.2-5.01)
`cut`	Quality of the cut (Fair, Good, Very good, Premium, Ideal)
`color`	Diamond color, from D (best) to J (worst)
`clarity`	Measure of diamond clarity (I1 (worst), SI2, SI1,VS2, VS1, VVS2, VVS1, IF (best))
`x`	Length in mm (0-10.74)
`y`	Width in mm (0-58.9)
`z`	Depth in mm (0-31.8)
`depth`	Total depth %age = `z / mean(x, y)` (43-79)
`table`	Width of top relative to widest point (43-95)

Bar plot showing distribution of diamond quality

```
sns.countplot(x='cut', data=diamonds)

plt.show()
```



Bar plot showing distribution of diamond quality



The bar chart will display the quality of the diamond (**`cut`**) on the x-axis, and the y-axis will display the count (or the number of times) a diamond with that quality appears in the dataset.

Statistical transformations

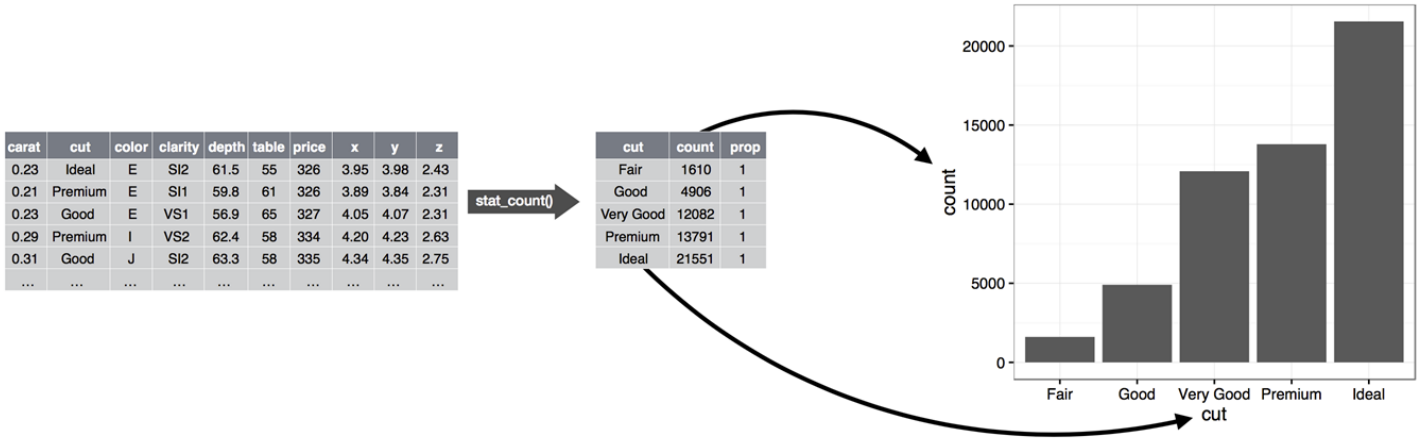
This bar chart permitted us to examine the data using a **statistical transformation** (or **stat**).

There is first a transformation of the data into counts, which is then returned for each diamond quality (`cut`).

That is, five values are returned, summarizing how often each quality of diamond appears in the dataset.

This summarized (transformed) data is then displayed on the y-axis as a function of diamond quality on the x-axis using the `value_counts()` function.

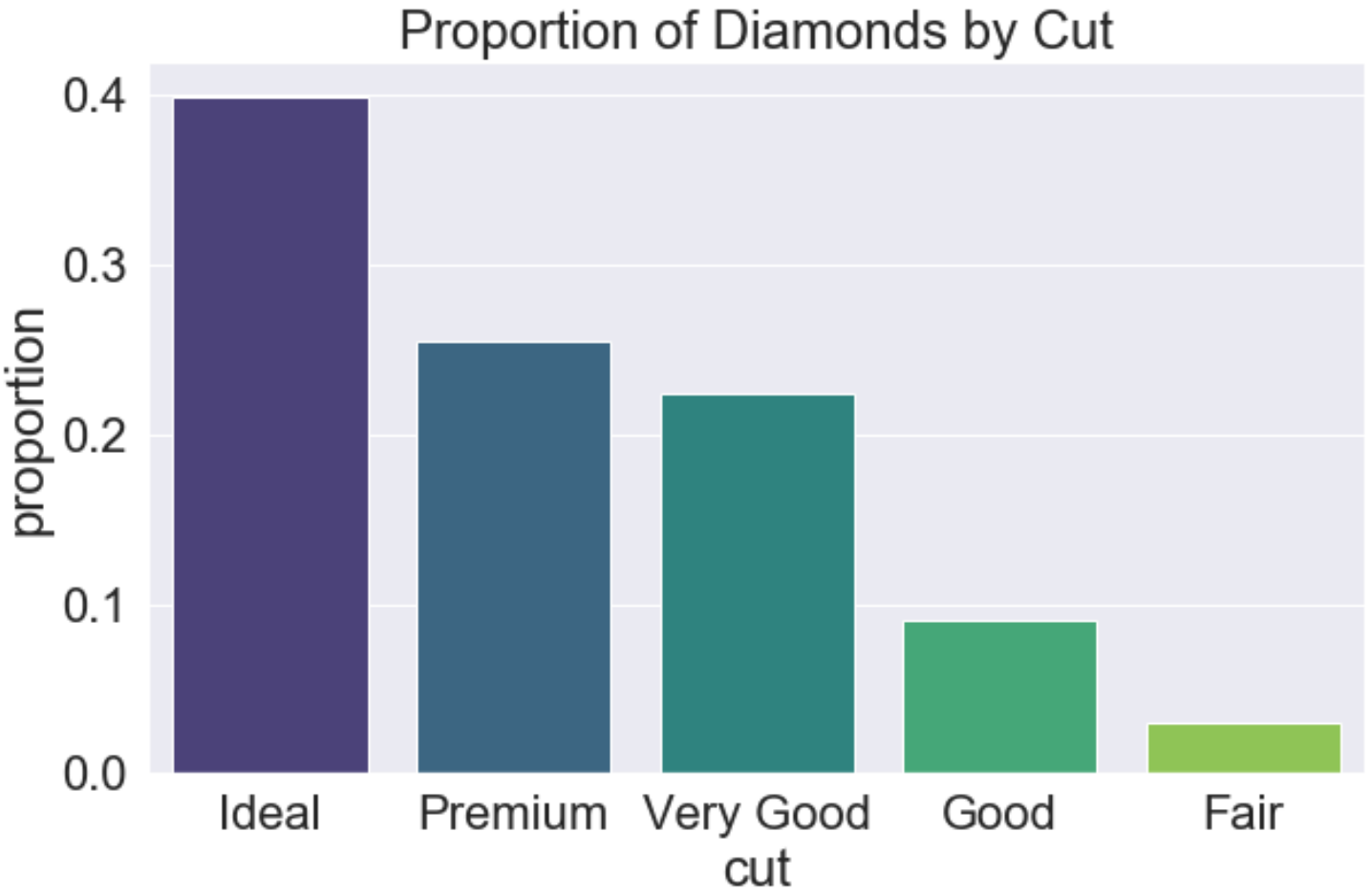
Algorithm used by `sns.countplot()`




```
In [416]:
1  diamonds.groupby('cut')['carat'].count()
2  diamonds['cut'].value_counts()

Out[416]: cut
          Fair          1610
          Good          4906
          Ideal        21551
          Premium      13791
          Very Good    12082
          Name: carat, dtype: int64

Out[416]: Ideal        21551
          Premium      13791
          Very Good    12082
          Good          4906
          Fair          1610
          Name: cut, dtype: int64
```

Displaying proportions rather than counts

```
proportions = diamonds.groupby('cut').count().reset_index()
proportions['proportion'] = proportions['count'] / proportions['count'].sum()
proportions = proportions.sort_values(by = 'proportion', ascending=False)

sns.barplot(x="cut", y="proportion", data=proportions, palette="viridis")
plt.title('Proportion of Diamonds by Cut')
```



The `seaborn` cheat sheet

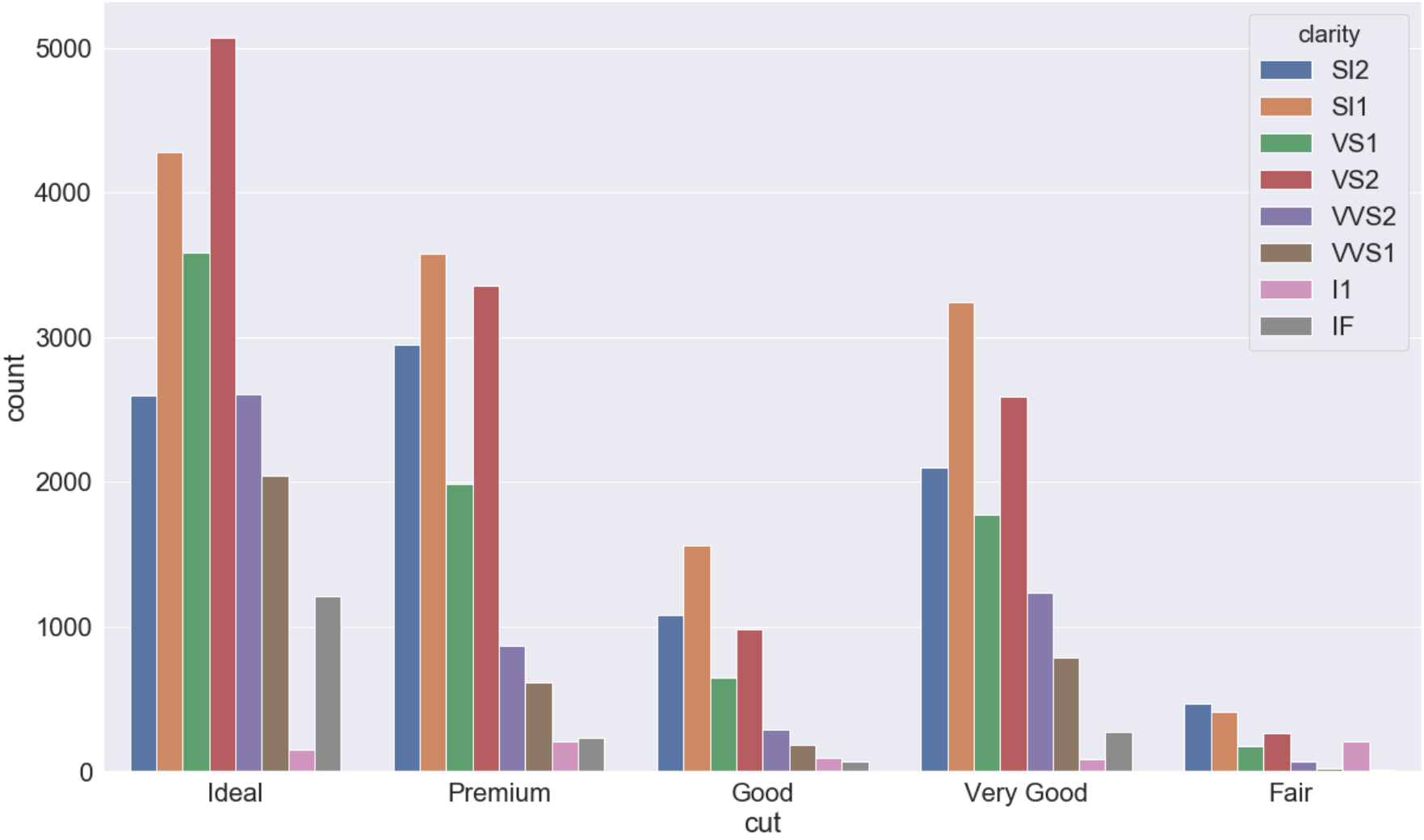
The flexibility of Python visualization can feel overwhelming at first.

Luckily, there is a **`seaborn`** cheat sheet that summarizes all the important plotting and visualization functions.

I have uploaded a copy of this cheat sheet to Canvas.

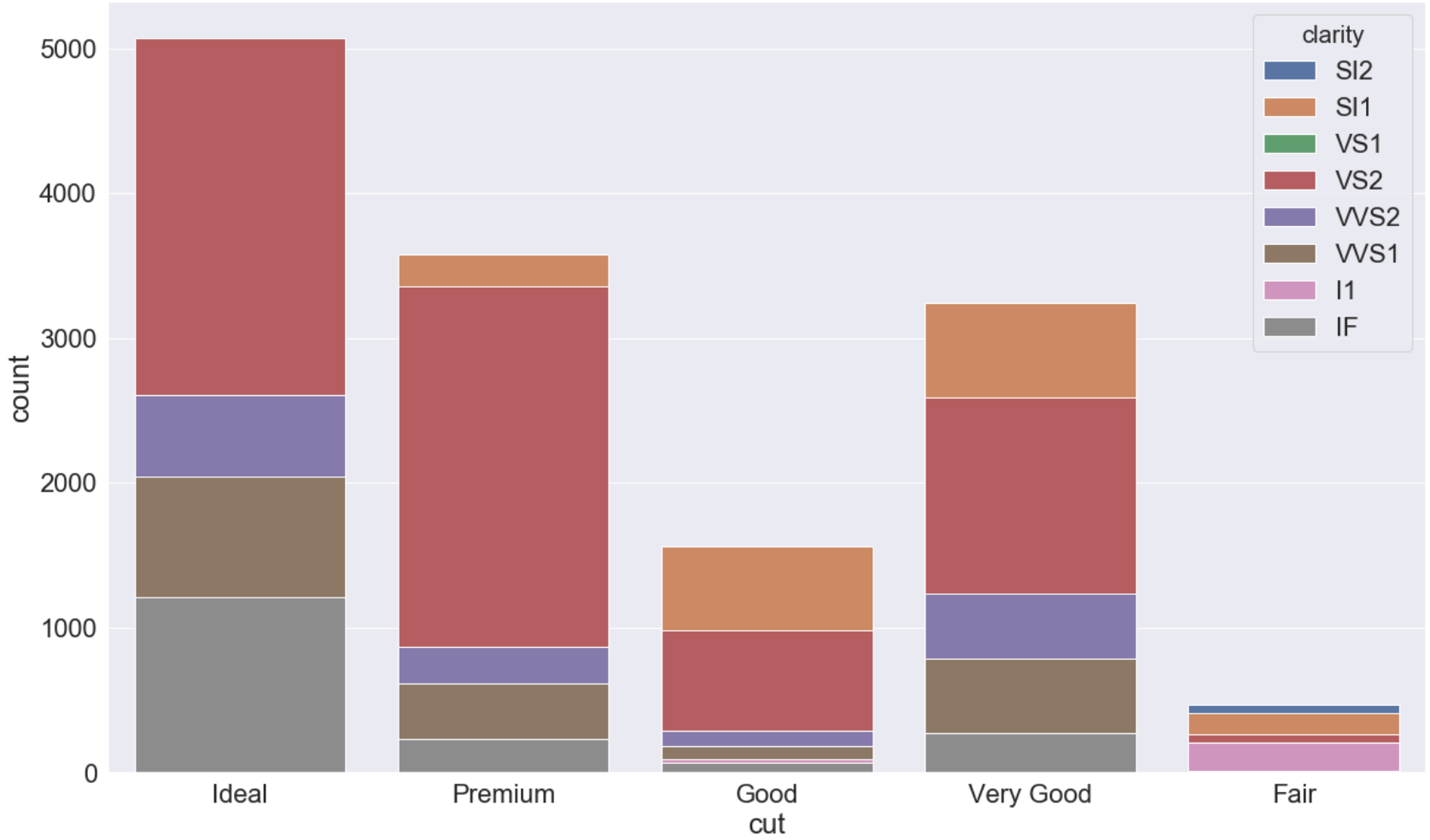
Stacked bar charts as function of different category

```
sns.countplot(x='cut', data=diamonds,
                       hue='clarity')
```



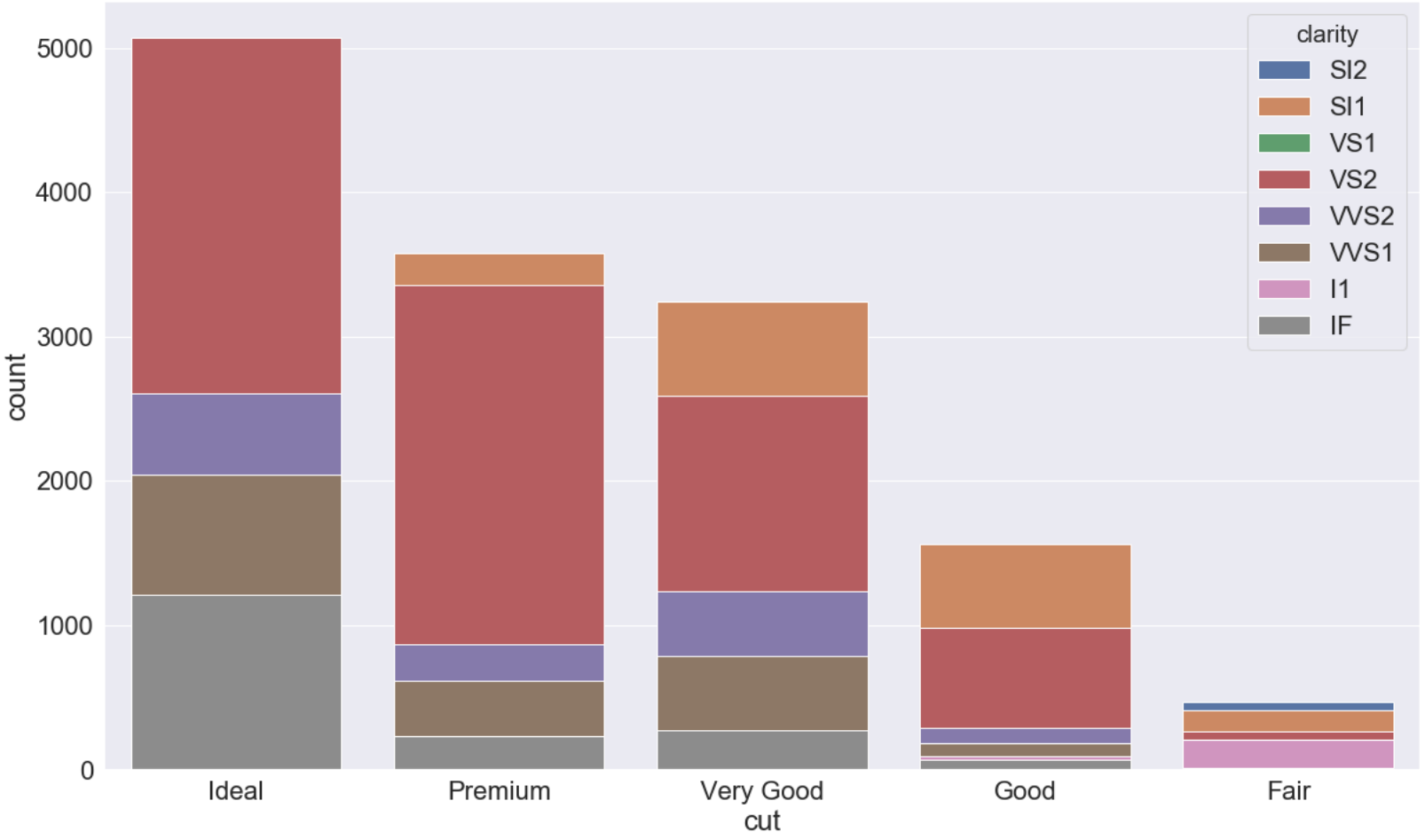
Stacked bar charts as function of different category

```
sns.countplot(x='cut', data=diamonds,
              hue='clarity',
              dodge=False)
```



Stacked bar charts as function of different category

```
sns.countplot(x = 'cut', data = diamonds,
              hue = 'clarity',
              dodge = False,
              order = diamonds['cut'].value_counts().index)
```



Histograms

Histograms are used to summarize distributions of continuous variables.

How a particular variable is distributed in a dataset.

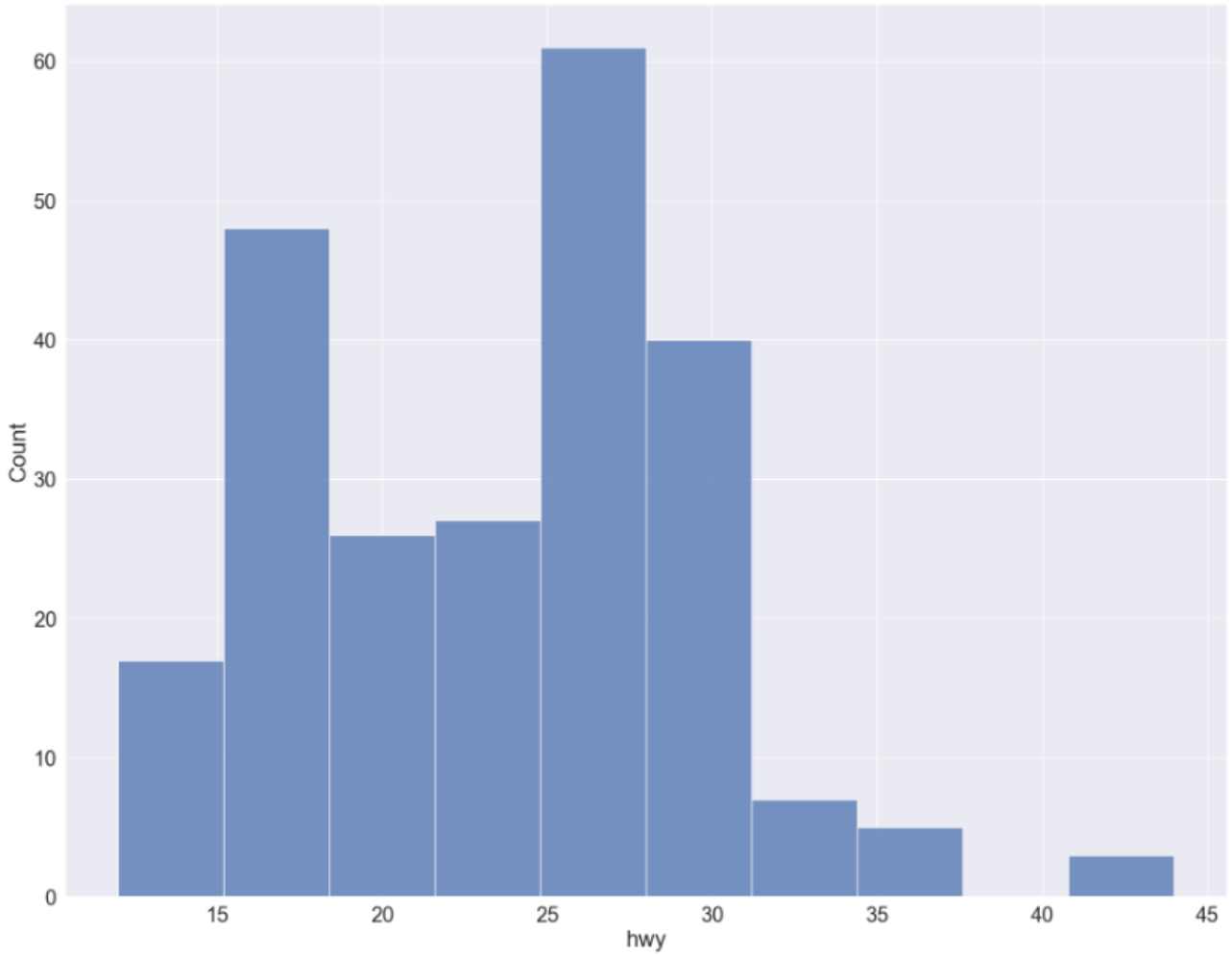
The following code summarizes the distribution of highway miles per gallon in the mpg data using a histogram.

```
sns.histplot(mpg['hwy'], bins=10)
plt.show()
```

It will make a histogram with 10 equally-spaced bins.

Histogram of highway miles per gallon

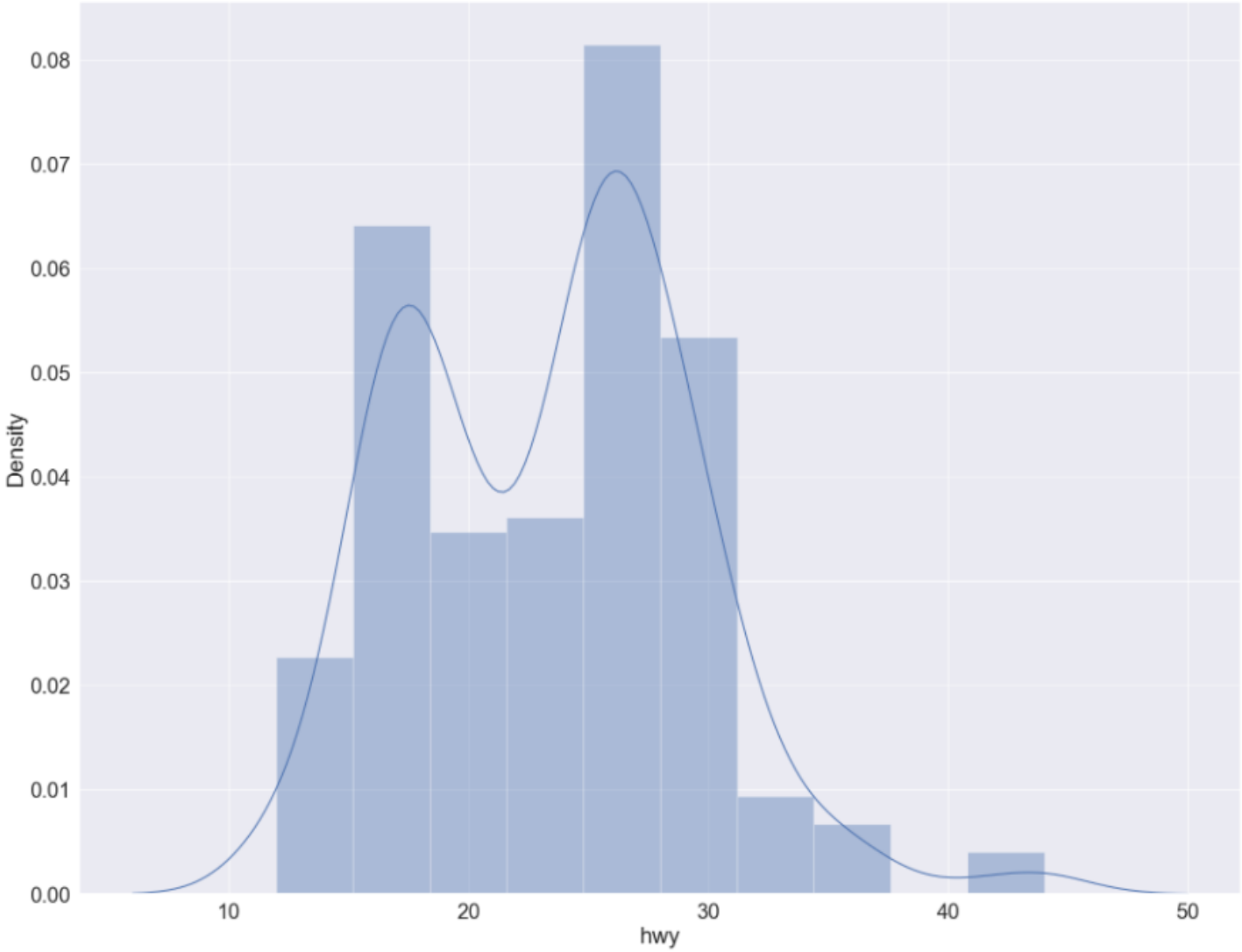
```
sns.histplot(mpg['hwy'], bins=30)
plt.show()
```



`distplot` of highway miles per gallon

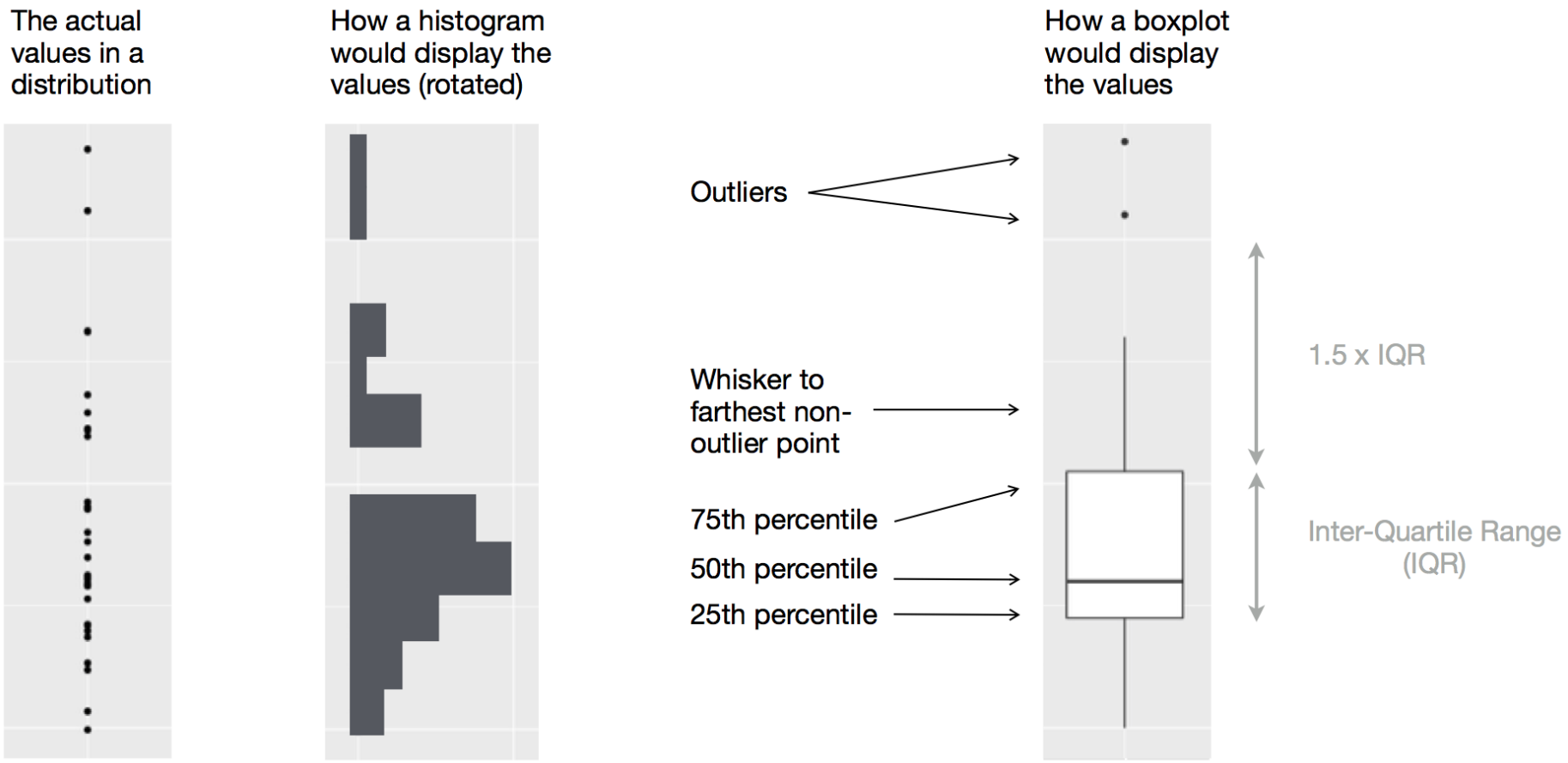
```
sns.distplot(mpg['hwy'], bins=10)
```

Allow you to plot the distributions of numeric variables.



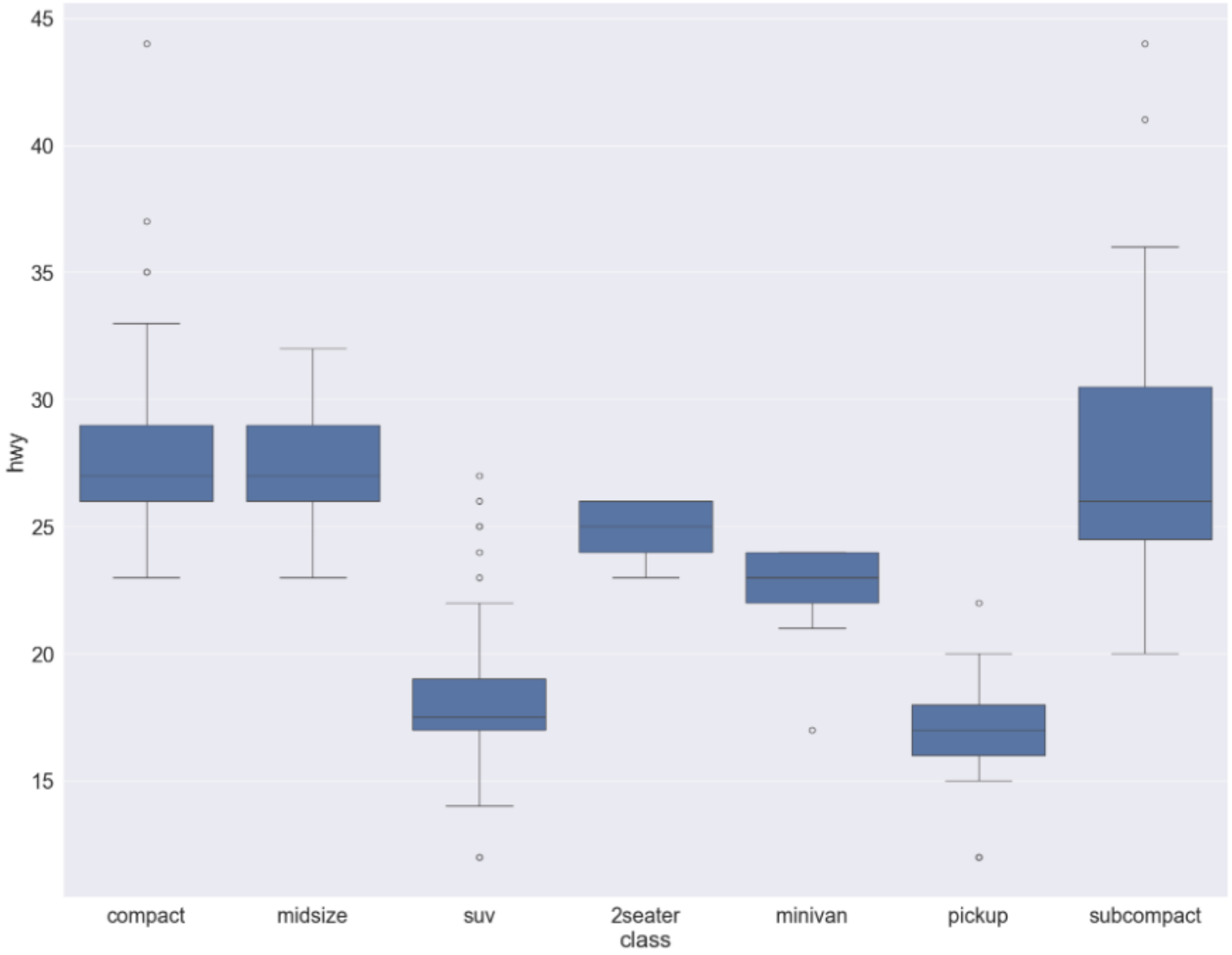
Box plots

Box plots are a compact way of summarizing the distribution of a set of points.



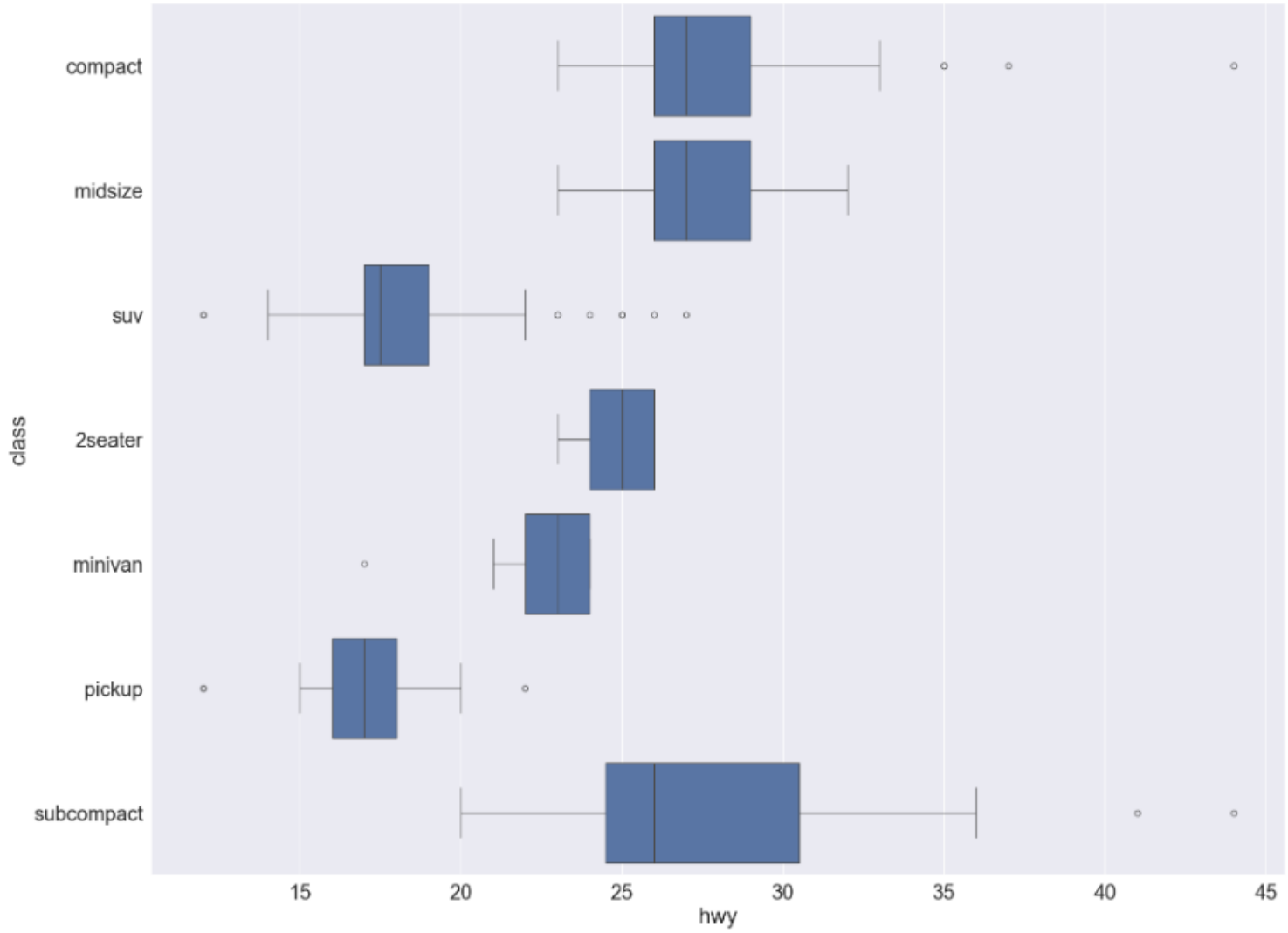
Creating box plots

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



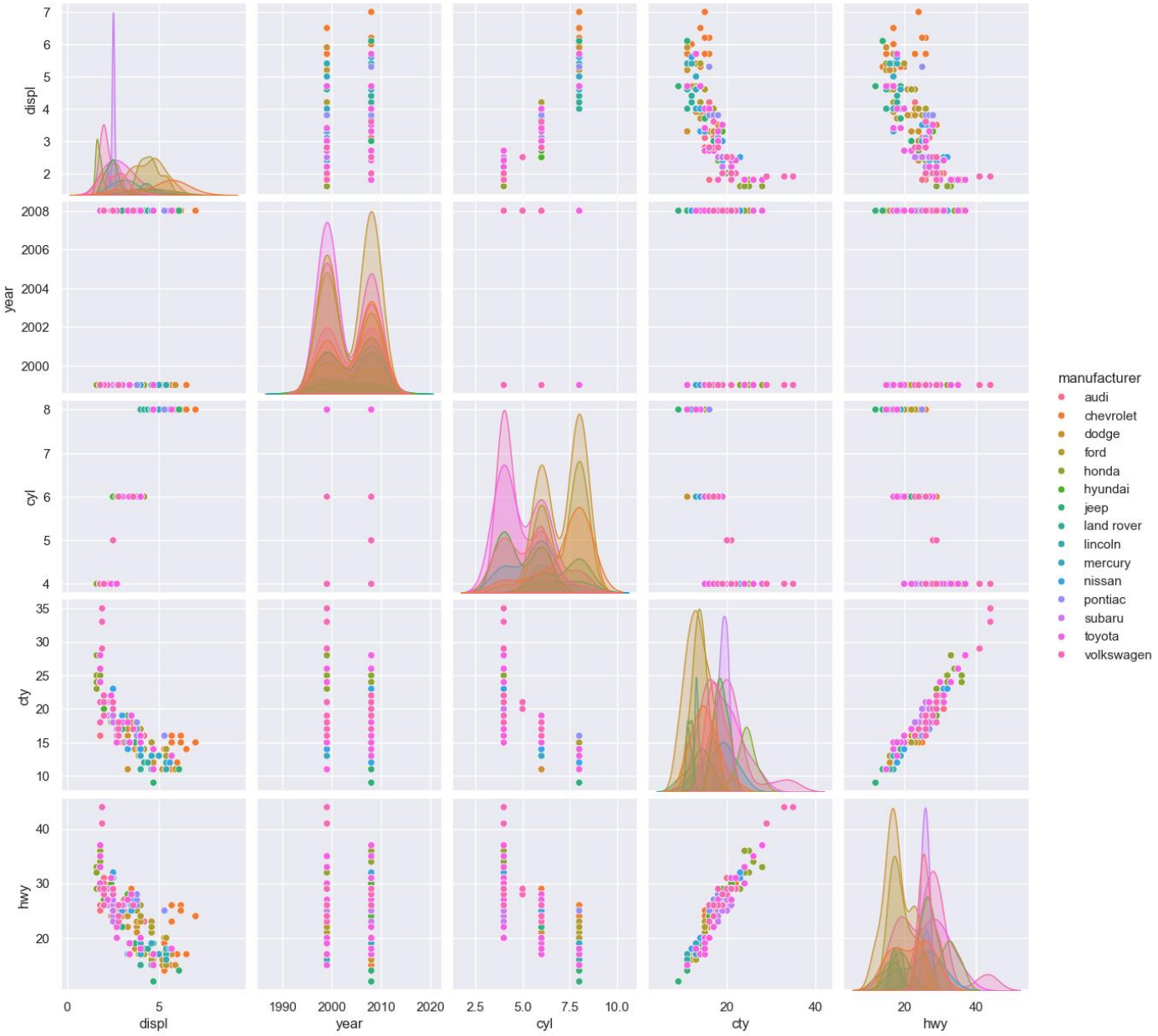
Flipping coordinates

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



Paiplot

```
sns.pairplot(data=mpg, hue='manufacturer')
```



Heatmaps

A heatmap is a graphical representation of data that uses a system of color coding to represent different values. Heatmaps are used in various forms of analytics but are most commonly used to show user behavior on specific web pages or webpage templates.

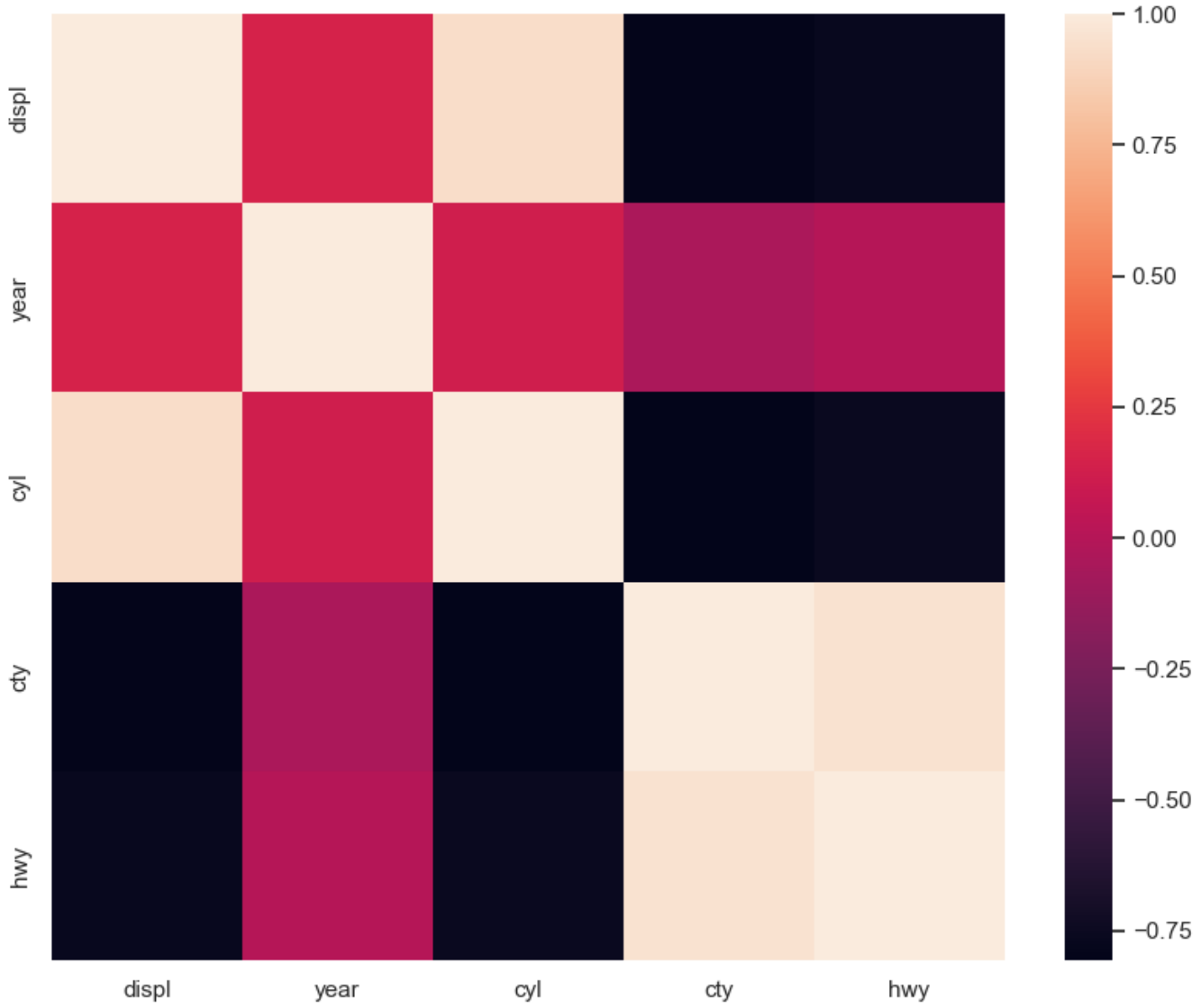
How to create heatmaps using pandas and seaborn?

```
mpg.corr()
```

	displ	year	cyl	cty	hwy
displ	1.000000	0.147843	0.930227	-0.798524	-0.766020
year	0.147843	1.000000	0.122245	-0.037232	0.002158
cyl	0.930227	0.122245	1.000000	-0.805771	-0.761912
cty	-0.798524	-0.037232	-0.805771	1.000000	0.955916
hwy	-0.766020	0.002158	-0.761912	0.955916	1.000000

Visualizing Heatmaps in **seaborn**

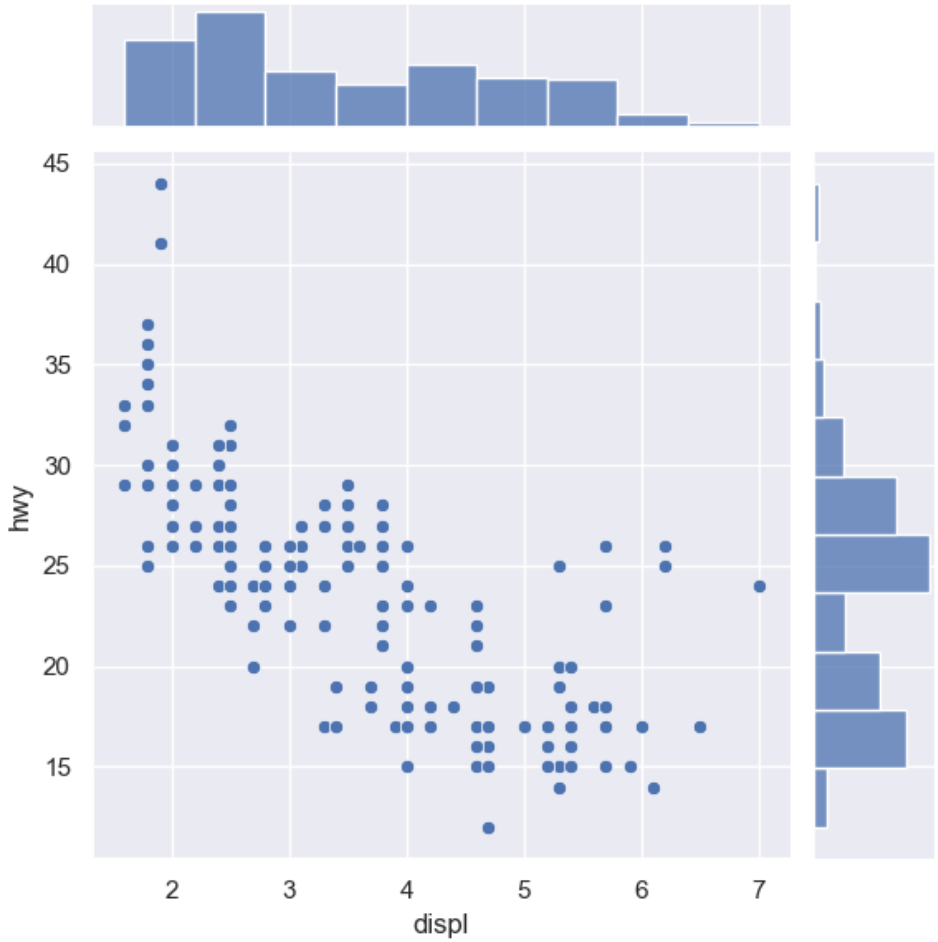
```
sns.heatmap(mpg.corr())
```



Joint Distribution Plot

- Joint distribution plots combine information from scatter plots and histograms to give you detailed information for bi-variate distributions.

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



Formatting – Size, Labels, Legends

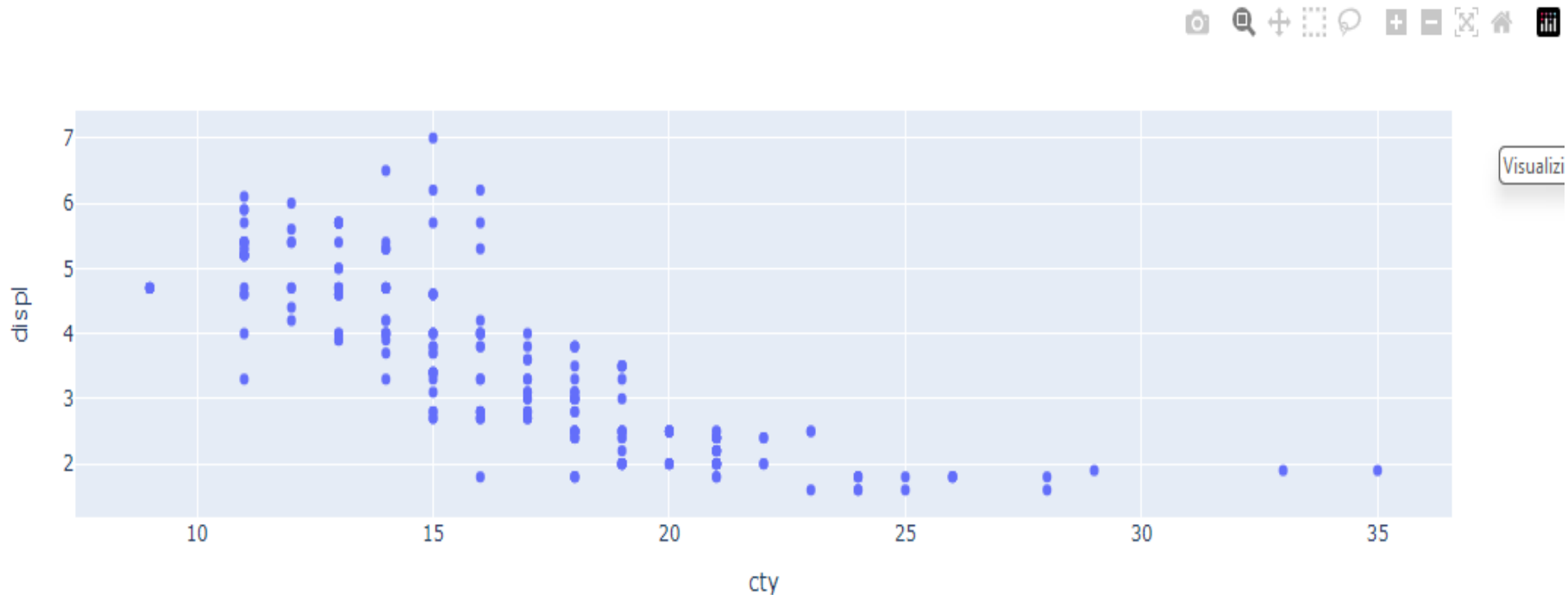
```
plt.xticks(rotation=90)

plt.figure(figsize=(10, 6))

x.set(xlabel='common xlabel', ylabel='common ylabel’)
```

https://www.geeksforgeeks.org/how-to-change-seaborn-legends-font-size-location-and-color/

Using `plotly`

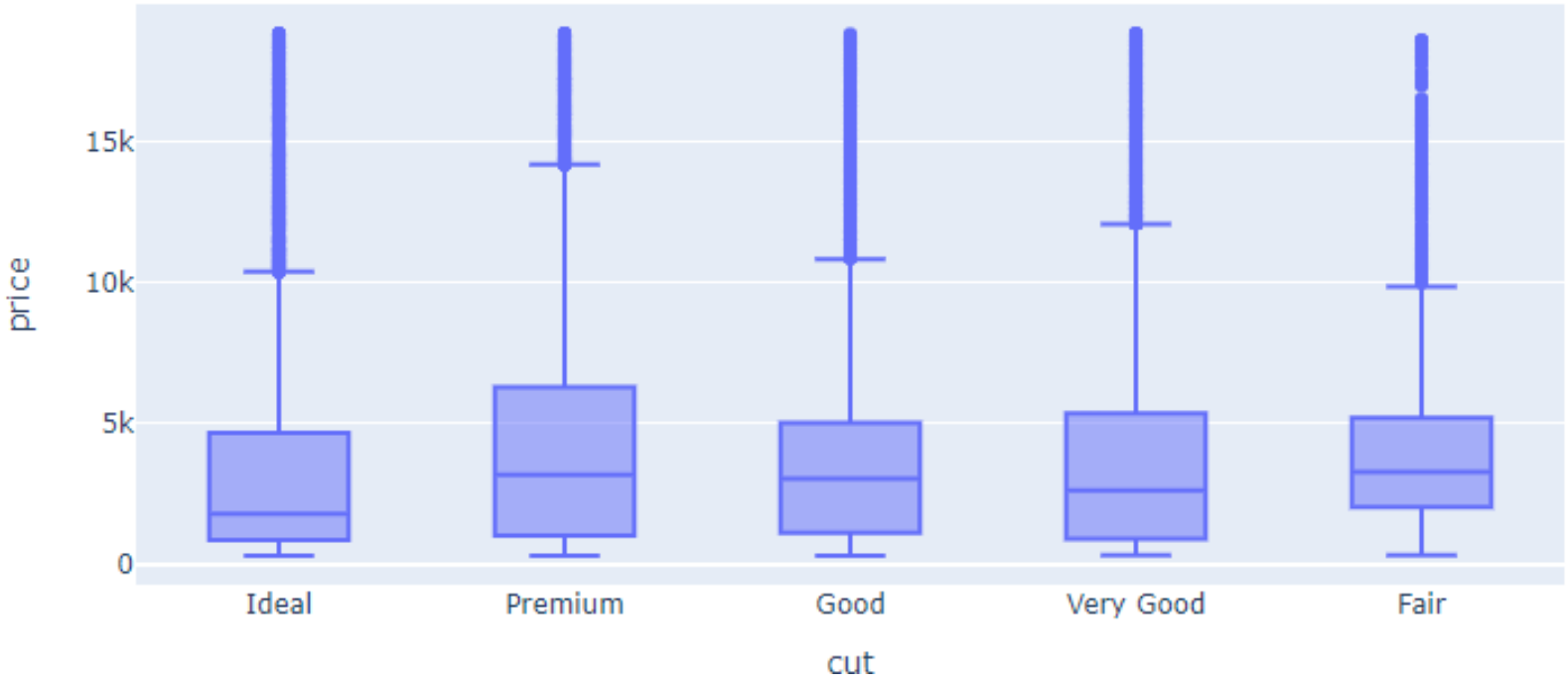
- An interactive, open-source, and browser-based graphing library. It offers Python-based charting, powered by plotly.js.

```
fig = px.scatter(mpg, x='cty', y='displ')
fig.write_html("plotly1.html")
```



Using `plotly`

- `px.box(diamonds, x='cut', y='price', width=800, height=400)`



Data Storytelling

Data storytelling is the concept of building a compelling narrative based on complex data and analytics that help tell your insights story and influence and inform a particular audience.

The benefits of data storytelling

Data storytelling is very similar to human storytelling but provides the added benefits of deeper insights and supporting evidence through graphs and charts. Through data storytelling, complicated information is simplified so that your audience can engage with your content and make critical decisions quicker and more confidently.

Constructing a data story that moves a person to take action can be a very powerful tool. Effective data storytelling can have a positive impact on people and your organization. Some benefits of successful data storytelling include:

- Adding value to your data and insights.
- Interpreting complex information and highlighting essential key points for the audience.
- Providing a human touch to your data.
- Offering value to your audience and industry.
- Building credibility as an industry and topic thought leader.

Data Storytelling

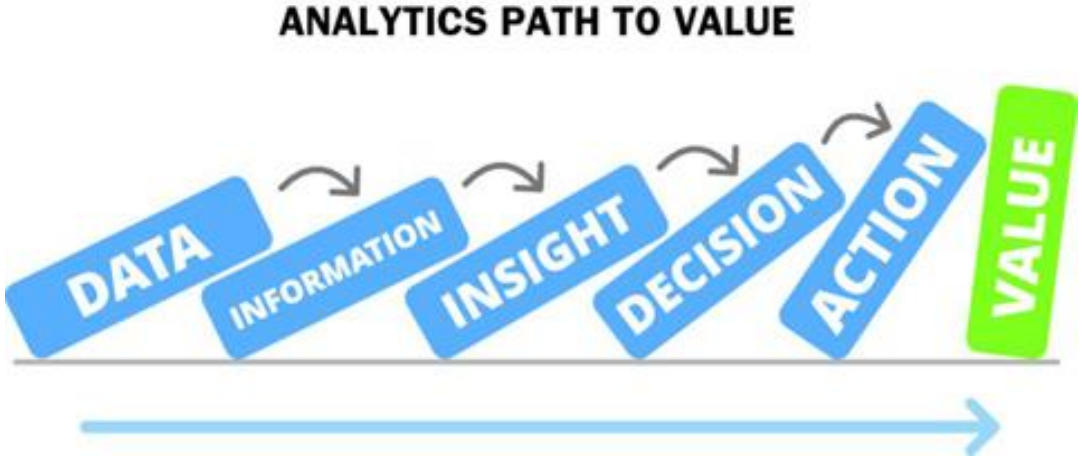
Using data visualization to enhance your data storytelling

- Reveal patterns, trends, and findings from an unbiased viewpoint.
- Provide context, interpret results, and articulate insights.
- Streamline data so your audience can process information.
- Improve audience engagement.

Driving Change through Insight

What is Insight?

- Insight comes from Middle English for “inner sight” or “sight with the ‘eyes’ of the mind” (Online Etymology Dictionary 2019).
- Psychologist Gary Klein defined an insight as “an unexpected shift in the way we understand things” (Gregoire 2013).
- These “unexpected shifts” in our knowledge can occur as we analyze and examine data.
- For example, we may uncover a new relationship, pattern, trend, or anomaly in the data that reshapes how we view things. While most insights are interesting, not all of them are valuable



Telling the Story of Your Data… Why?

- Humans Are Storytelling Creatures
- Stories Beat Statistics
- Storytelling is the most powerful way to put ideas into the world today.
- Storytelling has proven to be a powerful delivery mechanism for sharing insights and ideas in a way that is more memorable and persuasive than just pure facts.

Telling the Story of Your Data… Why?

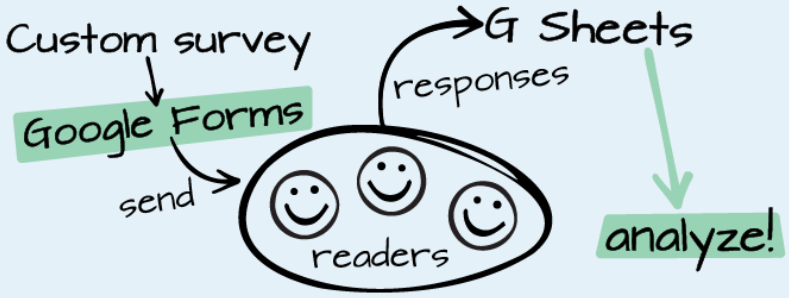
Step 1: Problem

We need to find out our readers' backgrounds and interests about data visualization, in order to write a better introductory guide that meets their needs.

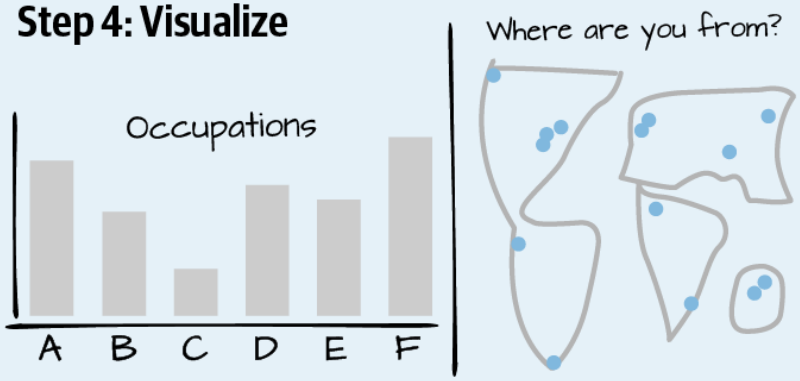
Step 2: Statement→Question

How do readers of our book describe their prior experience with data visualization, education level, and learning goals?

Step 3: Find data



Step 4: Visualize



Up Next ...

Data Transformations