Seaborn

January 27, 2026

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt   # Not strictly necessary to load, but it lets␣
 ↪you tweak things more
import seaborn as sns             # It's customary to use sns as the short name␣
 ↪for the seaborn package
sns.set_theme()                   # This applies the default Seaborn theme
```

1 Seaborn

Excellent plotting library that is great for making 2D visualizations. It's particularly strong at visualizing data tables that contain many columns and with it's heavy focus on statistics, it makes it very easy to plot linear regression for example.

Website: https://seaborn.pydata.org/

The library itself is based on Matplotlib and many of the plot display options can be directly applied to plots produced by Seaborn.

It also works really well with Pandas DataFrame. If you have not heard of Pandas before, it is pretty much the standard in Python. It's a very fast library that can load and write data and has very powerful features that lets you disect and manipulate data. It is not the focus of this workshop so we'll not dive into it.

1.1 Data sources

Seaborn comes with a large amount of example datasets in exactly that format so for this session, we can only focus on visualization and not on data wrangling.

Let's load the mpg one which shows how fuel efficient certain cars are.

[4]:
```
mpg = sns.load_dataset("mpg")
mpg
```

[4]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	\
0	18.0	8	307.0	130.0	3504	12.0	
1	15.0	8	350.0	165.0	3693	11.5	
2	18.0	8	318.0	150.0	3436	11.0	
3	16.0	8	304.0	150.0	3433	12.0	
4	17.0	8	302.0	140.0	3449	10.5	

```
..    …        …            …           …      …           …
393  27.0          4         140.0        86.0    2790          15.6
394  44.0          4          97.0        52.0    2130          24.6
395  32.0          4         135.0        84.0    2295          11.6
396  28.0          4         120.0        79.0    2625          18.6
397  31.0          4         119.0        82.0    2720          19.4

     model_year  origin                       name
0            70     usa  chevrolet chevelle malibu
1            70     usa          buick skylark 320
2            70     usa         plymouth satellite
3            70     usa              amc rebel sst
4            70     usa                ford torino
..          …     …                          …
393          82     usa            ford mustang gl
394          82  europe                  vw pickup
395          82     usa              dodge rampage
396          82     usa                ford ranger
397          82     usa                 chevy s-10

[398 rows x 9 columns]
```

There are many columns here. Documentation of these datasets is unfortunately not great. Most columns are self-explanatory but not all. This is my best guess

Column	Meaning
mpg	Miles per gallon, i.e. fuel efficiency
cylinders	Number of cylinders in the engine
displacement	Engine size expressed as total volume of all cylinders
horsepower	Engine power
weight	Weight of the car in lbs
acceleration	Acceleration in seconds from 0 to 60 mph
model_year	Release year of this car model
origin	area where released
name	name of the brand and model

In any case, these are handy datasets to show what you can do with Seaborn for plotting.

2 Plotting basics

One of the more elementary plots it the trusty scatter plot where you plot one column against another column and display the data as scattered points. The function for that is `sns.scatterplot`. It takes at least three arguments. It needs to know which dataset you want to plot, which column is the x-axis, and which column is the y-axis.

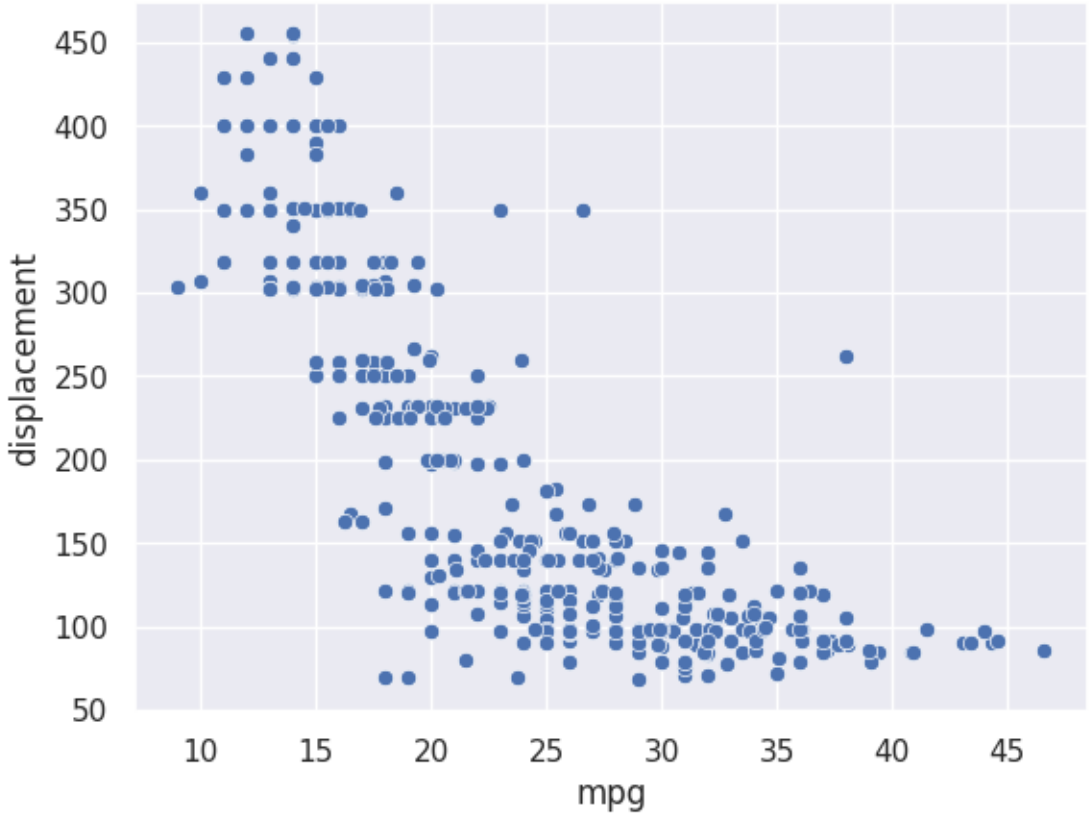
There are many more optional arguments, but we will get to that later.

In this example, we'll plot mpg versus displacement. The syntax of the plotting function is that you specify which dataset you want to plot using `data=`. In most cases, this will be a Pandas DataFrame, though other data is also possible. DataFrames are definitely more convenient so I would suggest converting your array into a DataFrame first.

The next thing you need to tell the plot function (if using a DataFrame) are which column should be plotted against which column using `x=` and `y=`.

[7]:
```
sns.scatterplot(data=mpg, x="mpg", y="displacement")
```

[7]:
```
<Axes: xlabel='mpg', ylabel='displacement'>
```



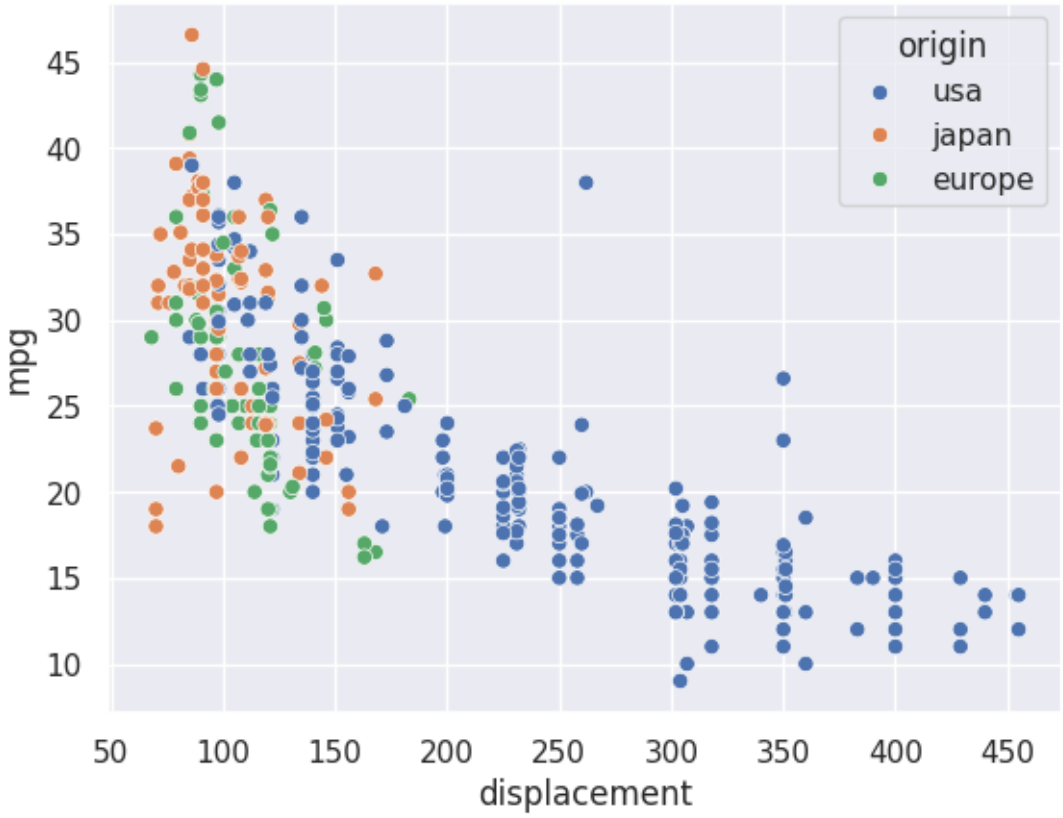
That that already shows that larger engines has poorer efficiency which is not really unexpected. There are some outliers though.

We do have many other columns of data that we are not using. We can't really plot all of the columns or we'd have a 9-dimensional plot which is a bit hard to visualize. At least, my brain only goes up to three dimensions. One thing we can however do with Seaborn is style the markers using data from the other columns. The keywords for that are `hue=`, `style=`, and `size=` which respectively change the colour, shape, and size of the markers.

For example, we can drill down even further by also marking the point with a colour to indicate the country of origin.

[11]:
```
sns.scatterplot(data=mpg, x="displacement", y="mpg", hue="origin")
```

[11]:
```
<Axes: xlabel='displacement', ylabel='mpg'>
```

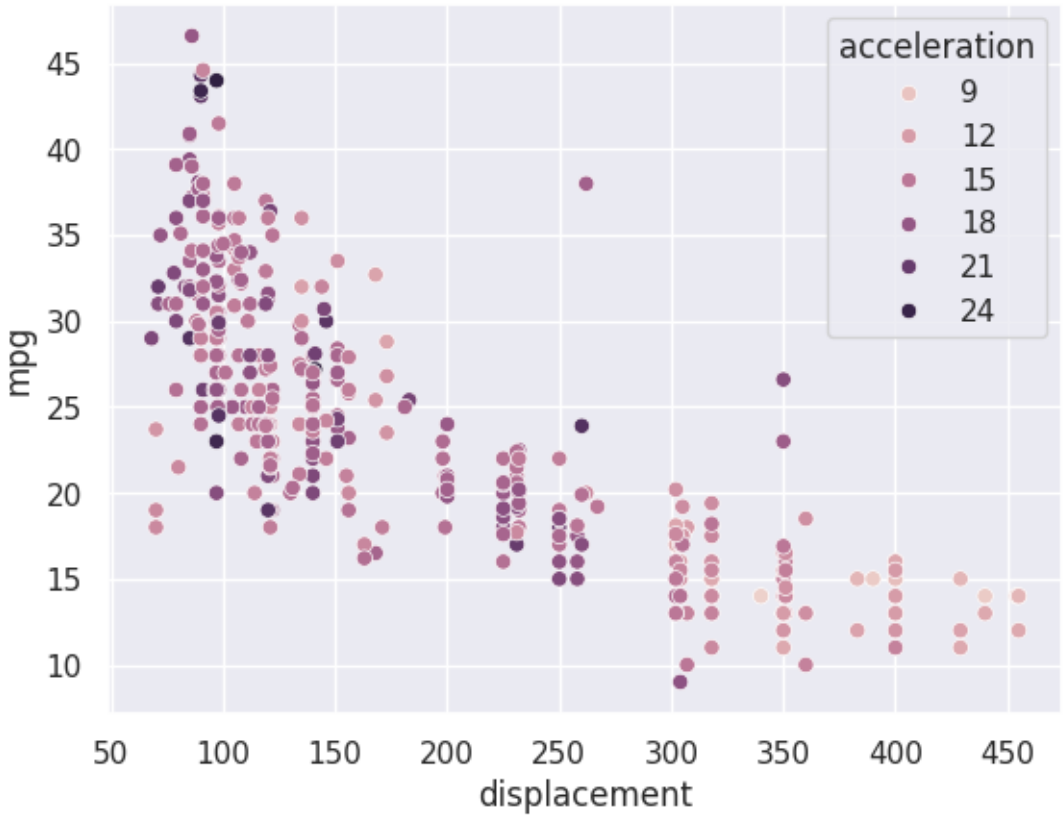


Now we clearly see that American cars have way bigger engines than European or Japanese cars and generally have way better fuel efficiency too. Note that it automatically generated a legend too as well as assigned colours to each of the categories.

Seaborn automatically detects is something is a discrete or continuous range. The origin column is a category (discrete) so they get distinct colours. Something like acceleration however is a continuous variable to when colouring that, Seaborn automatically applies a colour scale.

[21]:
```
sns.scatterplot(data=mpg, x="displacement", y="mpg", hue="acceleration")
```

[21]:
```
<Axes: xlabel='displacement', ylabel='mpg'>
```



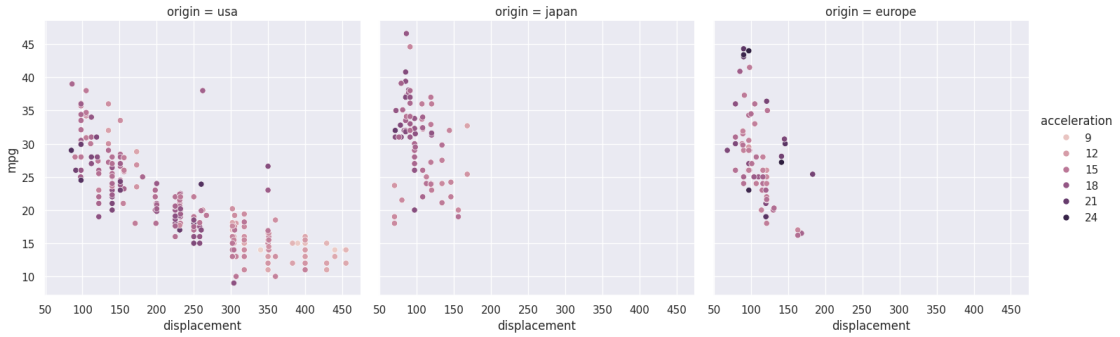
The legend now has light colours to dark colours. The default colour palette is chosen such that colourblind people can also easily view the plot.

2.1 Exercise 1

Play around with this. The extra parameters are `hue=` are `style=` and `size=`. You can even combine them.

[20]:
```
sns.relplot(data=mpg, x="displacement", y="mpg", hue="acceleration",␣
↪col="origin")
```

[20]:
```
<seaborn.axisgrid.FacetGrid at 0x14b538179a50>
```

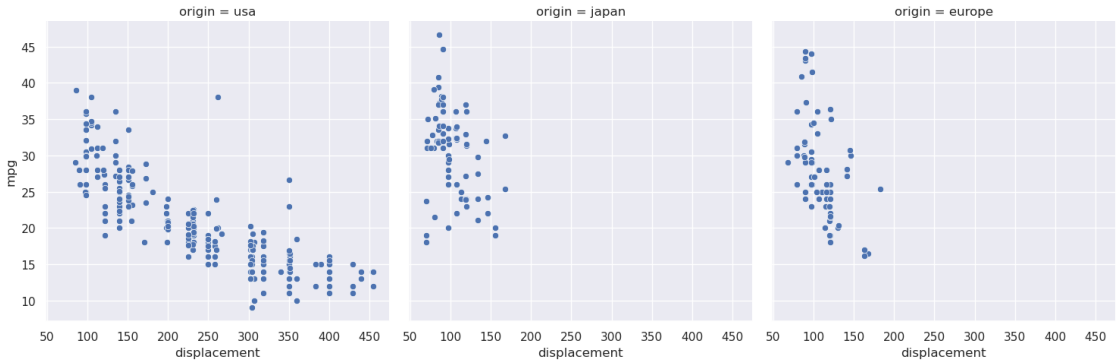


[ ]:

[ ]:

You may have noticed in exercise 1 that it can get quite crowded. Another thing you can do is to separate out the graphs for different categories. You will have to use `relplot` for this as this function knows how to make "FacetPlots". This function takes the extra arguments `col=` and `row=` for splitting by column and/or row.

[6]:
```
p = sns.relplot(data=mpg, x="displacement", y="mpg", col="origin")
```



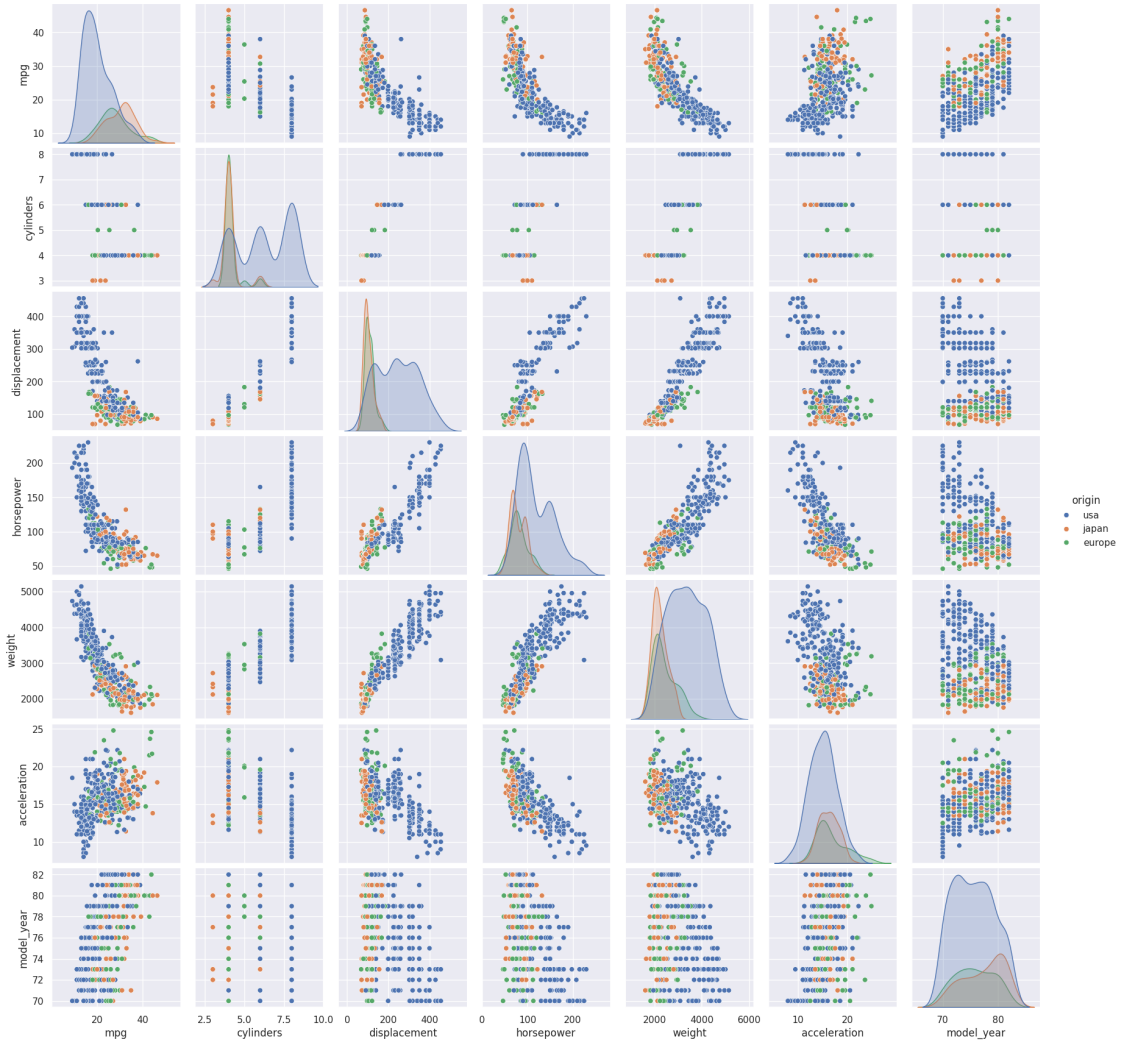
This technically lets you plot 7 variables at the same time (x, y, hue, style, size, row, col) though that will of course look awful!

2.2 Other plot types

There are many other plottypes available as well. One really nice one to see relations between variables at a glance is the pairplot. It makes a scatterplot for every possible combination of columns. It also takes the same `hue`, `size`, and `style` options as before. Let's see what that looks like.

[24]:
```
sns.pairplot(data=mpg, hue="origin")
```

[24]: <seaborn.axisgrid.PairGrid at 0x14b52c282250>

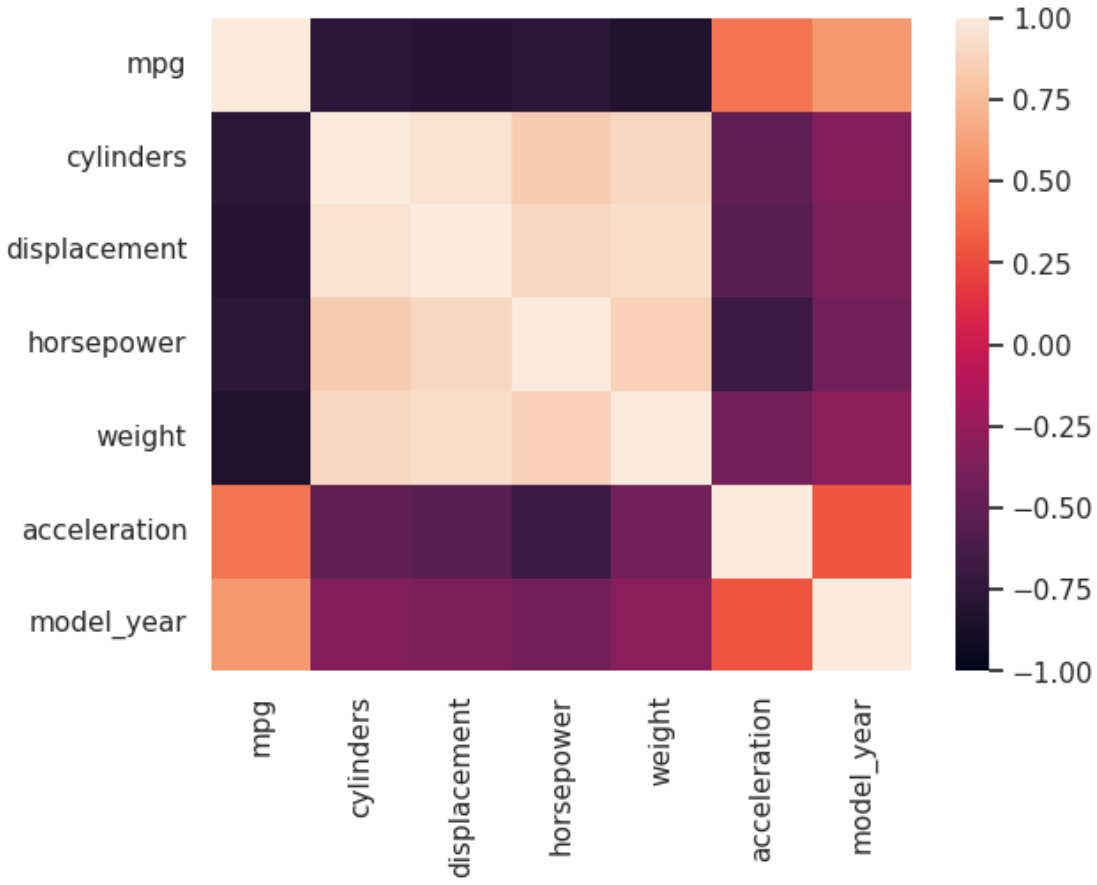


This immediately shows a bunch of interesting things. Greater fuel efficiency can be had from lower weight, less horsepower, and smaller engines. You can also see that newer cars generally have better fuel efficiency too. American cars are usually worse at fuel efficiency too.

Another thing we can show is the correlation matrix as a heatmap. A correlation is a value between -1 and 1 that tells you how correlated a column is with another. 1 means the two columns are proportionally related, -1 means that they are inversely related, and 0 means they are not related at all.

[27]:
```
corrmatrix = mpg.select_dtypes(exclude="object").corr()  # Pandas can calculate␣
↪the correlation matrix
sns.heatmap(corrmatrix, vmin=-1, vmax=1)                 # Plot the heatmap and␣
↪force the colours to range from -1 to 1
```

[27]: <Axes: >



This looks ok, but there is a better way of showing this. It would be much better if "not correlated", doesn't show as prominently. We'd ideally like to have white for the value 0 and opposing colours for positive and negative colours.

Enter colourmaps!

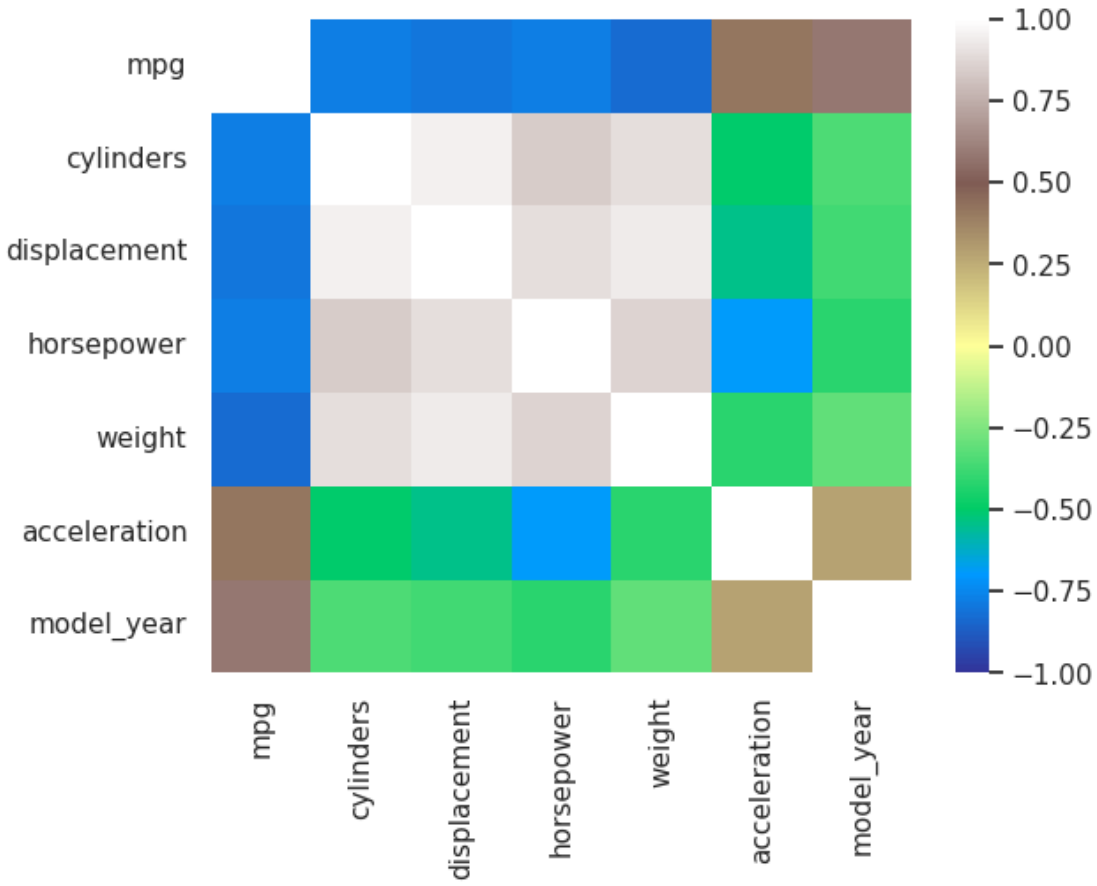
The Seaborn documentation has a lot of colours to choose from as well as tools to create your own. Additionally, these colourmaps are based on the Matplotlib ones, so all of those are available as well.

2.3 Exercise 2

Add the `cmap=` parameter to the heatmap plot command above and choose a palette that better shows positive and negative correlations.

[33]:
```
sns.heatmap(corrmatrix, vmin=-1, vmax=1, cmap="terrain")
```

[33]: <Axes: >



[ ]:

[ ]:

[ ]:

[ ]:

3 Theming

Speaking of colours, there is a lot more you can do than just altering the colour of the data. Seaborn has a `set_theme` function that we used at the very top of this notebook to set the default theme, but you can also give it options to change the theme. These are documented online. It's a great way to have a consistent look in your graphs.

The `set_theme` functions affects all plots, so you would typically call it right after importing the seaborn package. It has options for the context, style, palette, and fonts as well as some extra Matplotlib options that can be added.

- The context are things like "notebook" (default for Jupyter Notebooks) as well as "paper",

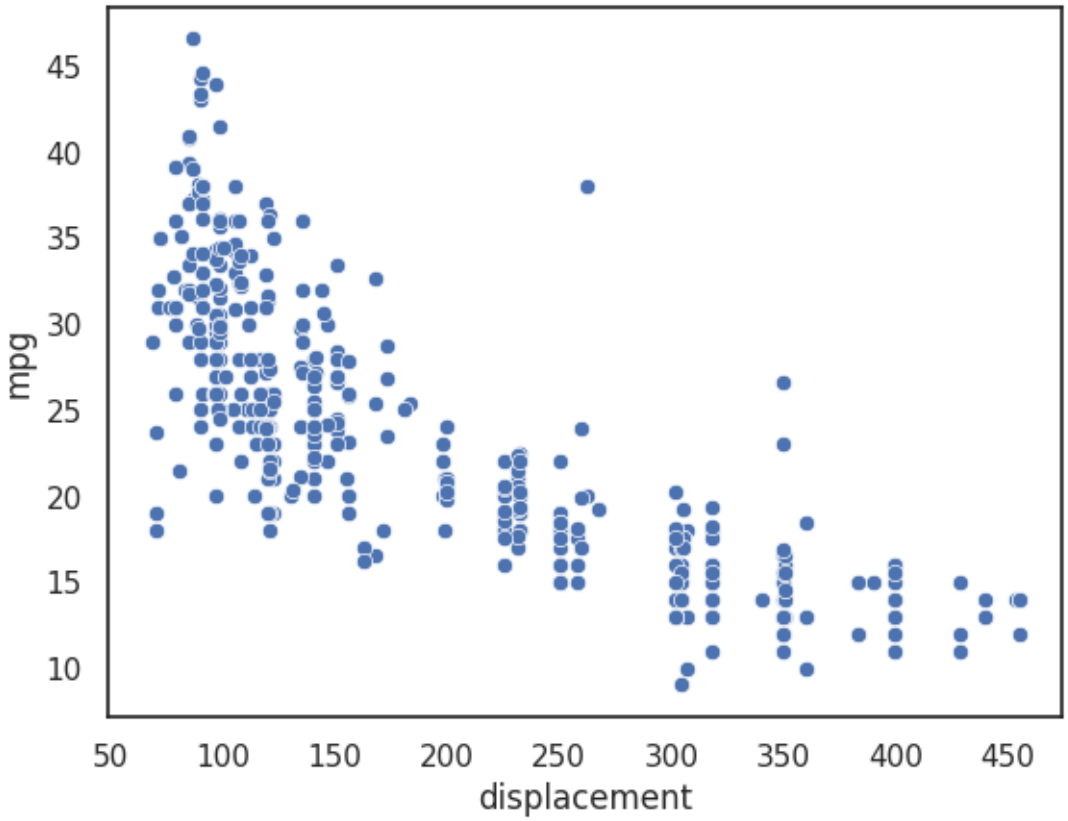
and “talk”. These affect things like linewidth, making the graph look better ot that specific medium.

- The style affect the general look of the graph and it sets colours and size. There are prebuilt styles or you can build your own.
- The pallete, we have seen before, but by using `set_theme`, the chosen palette applies to all graphs.

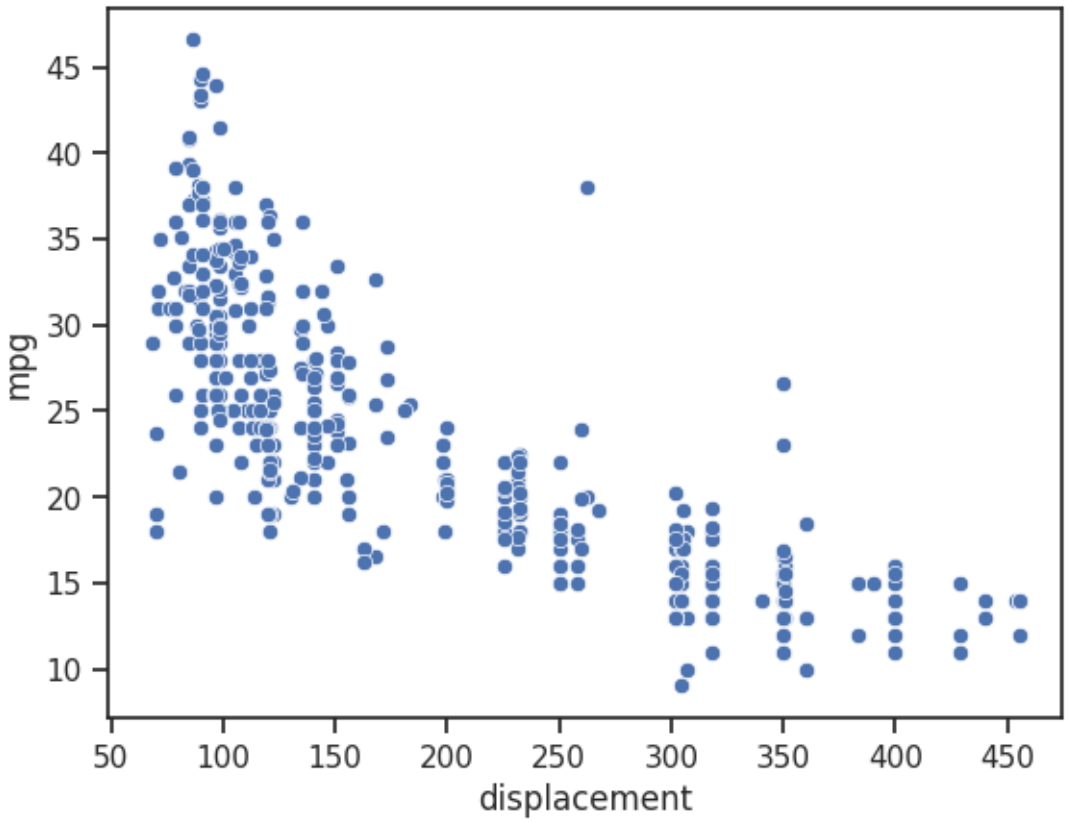
There are five preset styles to choose from. “darkgrid” is the default.

[34]:
```
for t in ["white", "ticks", "dark", "whitegrid", "darkgrid"]:
    sns.set_theme(style=t)
    sns.scatterplot(data=mpg, x="displacement", y="mpg")
    print(t)
    plt.show()
```

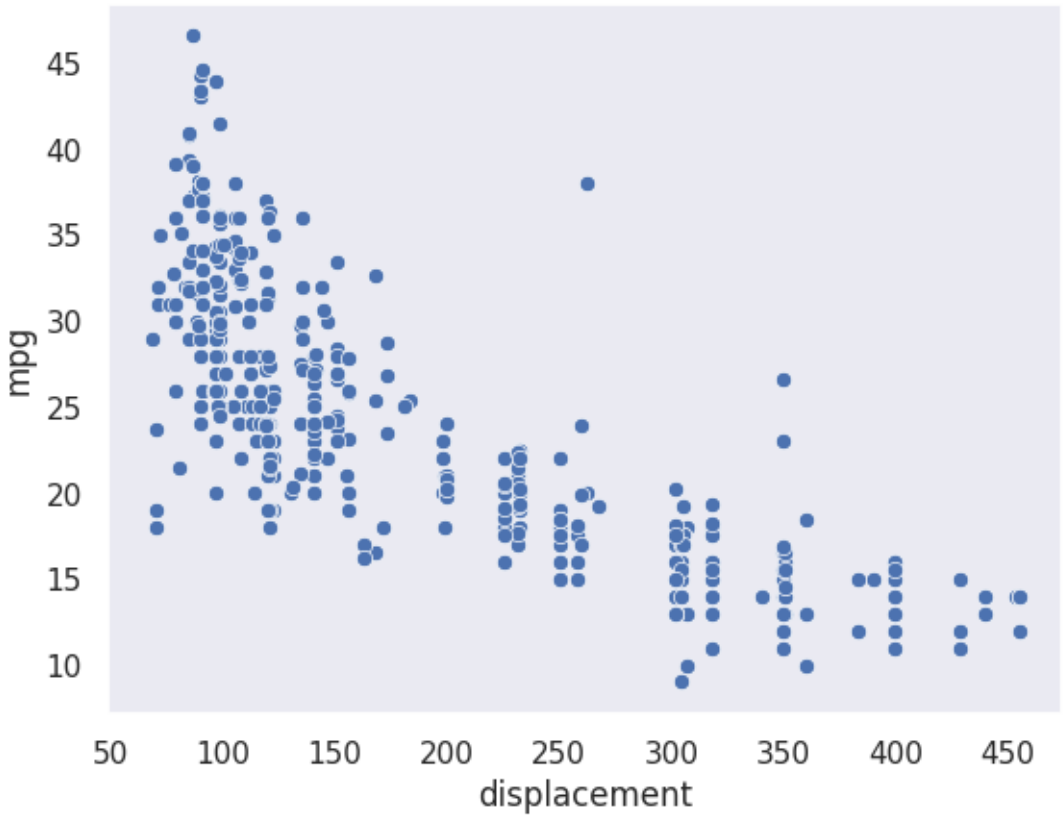
```
white
```



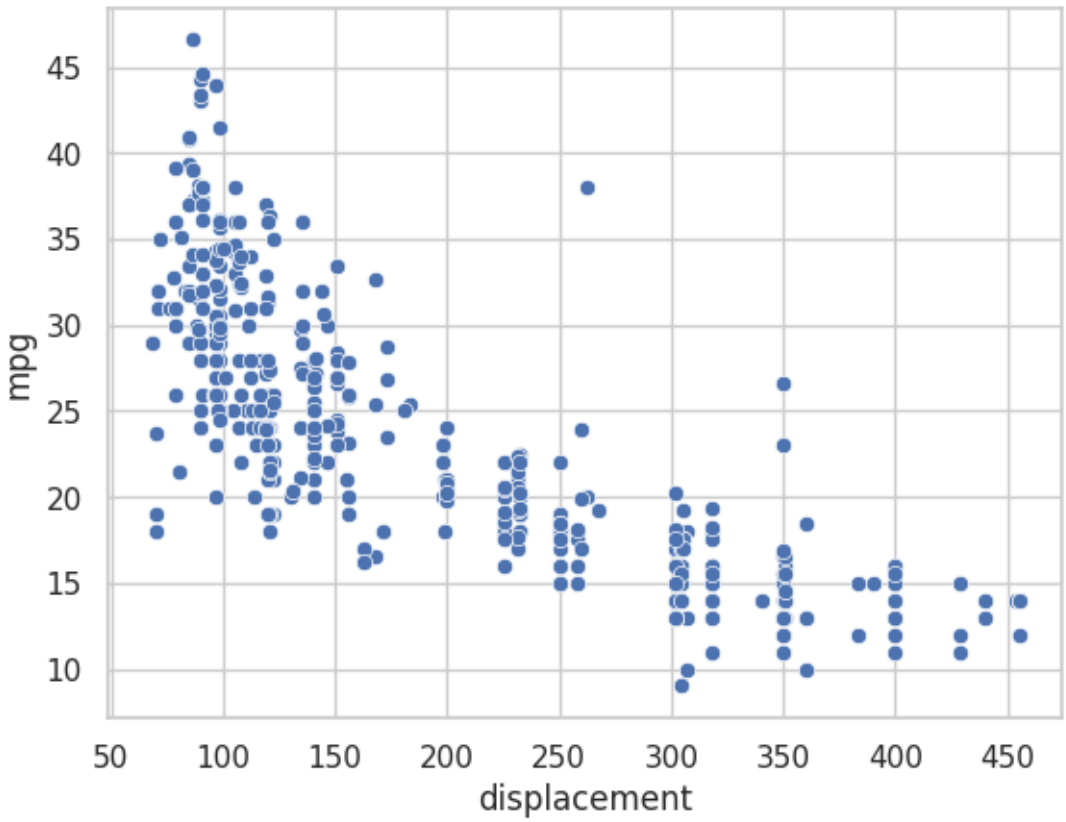

```
ticks
```


mpg
displacement
45
40
35
30
25
20
15
10
50
100
150
200
250
300
350
400
450

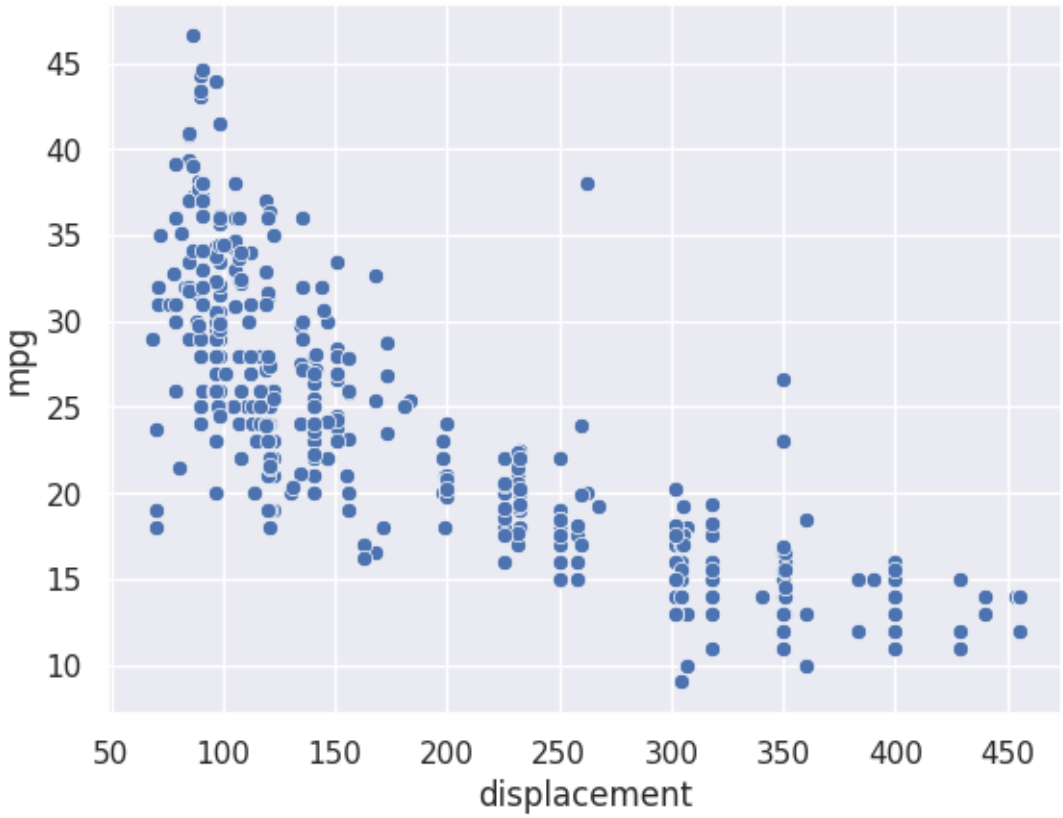
```
dark
```


45
40
35
30
25
20
15
10
mpg
50
100
150
200
250
300
350
400
450
displacement

```
whitegrid
```


45
40
35
30
25
20
15
10
mpg
50
100
150
200
250
300
350
400
450
displacement

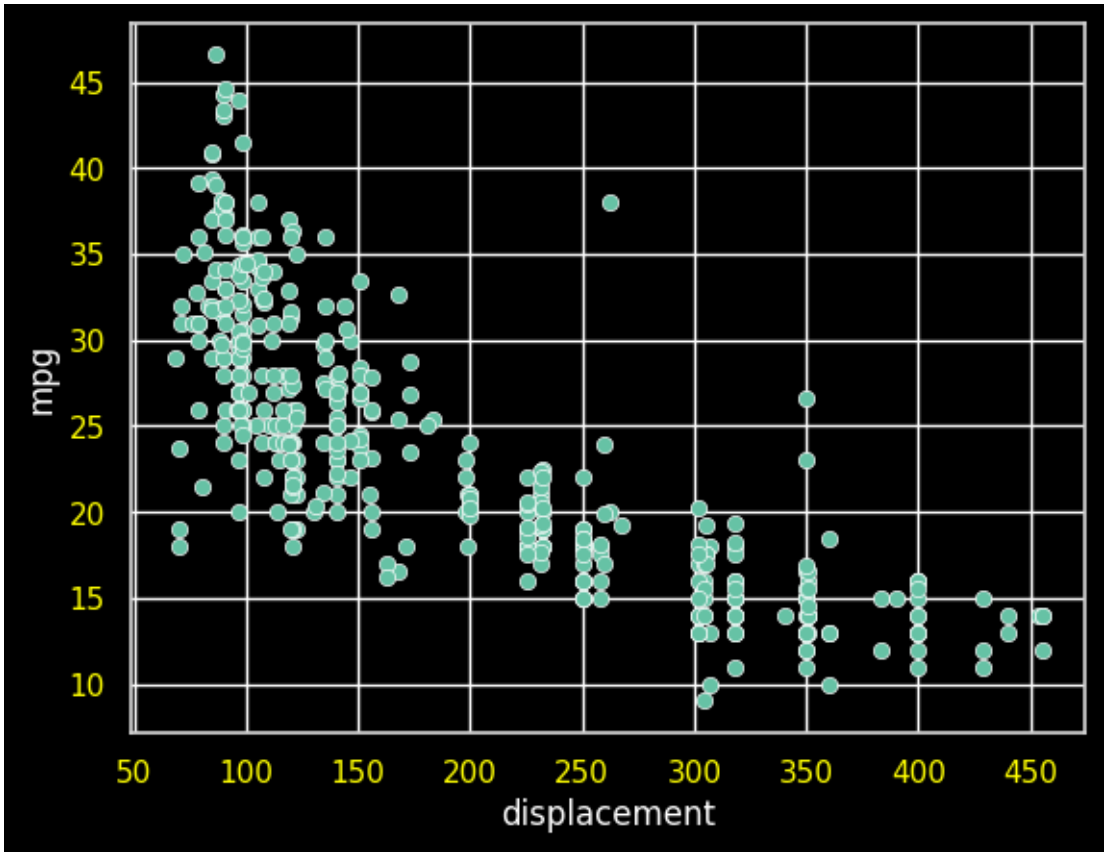
```
darkgrid
```



It is also possible to customoize these style. You can supply the `rc=` parameter to the `sns.set_theme` function to load customization on top of an prebuilt theme. For example, to get the style I used for the intro slide with a black background, I change the colours of the grid and text.

[38]:
```
custom_style = {
    'axes.facecolor': 'black',
    'axes.edgecolor': 'lightgrey',
    'figure.facecolor': 'black',
    'axes.labelcolor': 'white',
    'text.color': 'white',
    'xtick.color': 'yellow',
    'ytick.color': 'yellow',
}
sns.set_theme(style="darkgrid", palette="Set2", rc=custom_style)
sns.scatterplot(data=mpg, x="displacement", y="mpg")
```

[38]:
```
<Axes: xlabel='displacement', ylabel='mpg'>
```



You can view the settings for a built-in style with `axes_style`. This will tell you what is available to tweak as well.

[39]:
```
sns.axes_style("darkgrid")
```

[39]:
```
{'figure.facecolor': 'white',
 'axes.labelcolor': '.15',
 'xtick.direction': 'out',
 'ytick.direction': 'out',
 'xtick.color': '.15',
 'ytick.color': '.15',
 'axes.axisbelow': True,
 'grid.linestyle': '-',
 'text.color': '.15',
 'font.family': ['sans-serif'],
 'font.sans-serif': ['Arial',
  'DejaVu Sans',
  'Liberation Sans',
  'Bitstream Vera Sans',
  'sans-serif'],
 'lines.solid_capstyle': 'round',
```

```
 'patch.edgecolor': 'w',
 'patch.force_edgecolor': True,
 'image.cmap': 'rocket',
 'xtick.top': False,
 'ytick.right': False,
 'axes.grid': True,
 'axes.facecolor': '#EAEAF2',
 'axes.edgecolor': 'white',
 'grid.color': 'white',
 'axes.spines.left': True,
 'axes.spines.bottom': True,
 'axes.spines.right': True,
 'axes.spines.top': True,
 'xtick.bottom': False,
 'ytick.left': False}
```

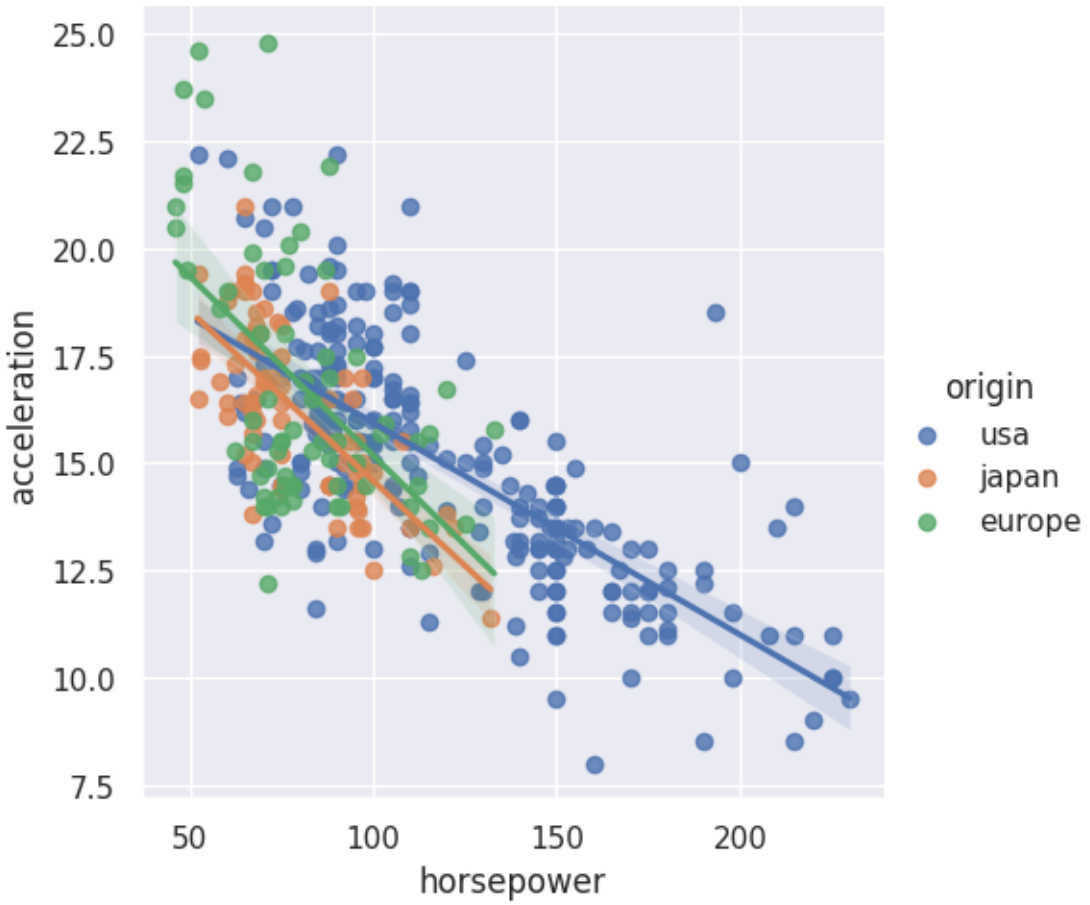
[43]:
```
sns.set_theme() # Reset the theme back to default
```

4 Statistics

One of the great strengths of Seaborn is the statistical plots it can show. For example, it can create plots with linear regression built in plus the confidence intervals.

[44]:
```
sns.lmplot(data=mpg, x="horsepower", y="acceleration", hue="origin")
```

[44]:
```
<seaborn.axisgrid.FacetGrid at 0x14b52462d450>
```

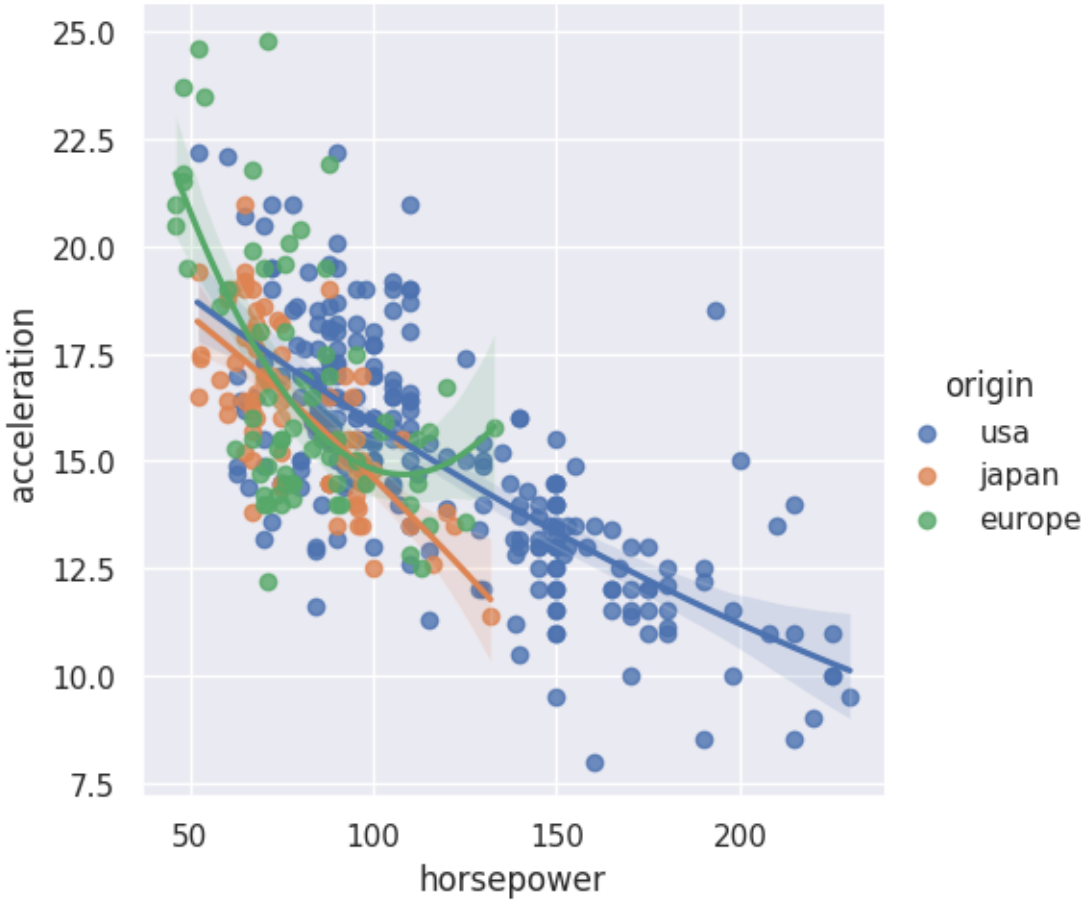


Looks like Japanese cars are better at translating more horsepower into faster acceleration. Of note is that `lmplot` does not need to do only linear regression. It can do other types too like polynomial and logistic.

For example, 2nd order polynomial:

[45]:
```
sns.lmplot(data=mpg, x="horsepower", y="acceleration", hue="origin", order=2)
```

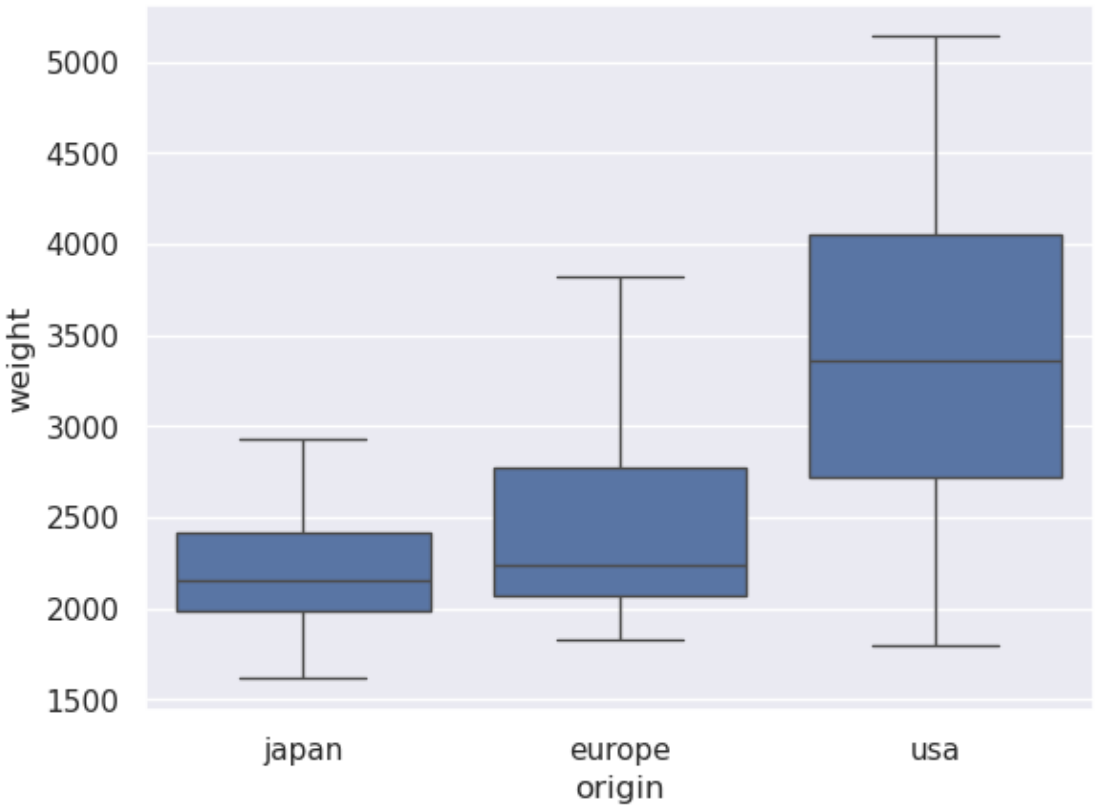
[45]:
```
<seaborn.axisgrid.FacetGrid at 0x14b5245a4ad0>
```



Another interesting one is weight for the various markets. That's nicely shown with a boxplot which shows the quartiles of the distribution as well. I also picked a particular order for this with the `order=` parameter.

[14]:
```
sns.boxplot(data=mpg, x="origin", y="weight", order=["japan", "europe", "usa"])
```

[14]:
```
<Axes: xlabel='origin', ylabel='weight'>
```

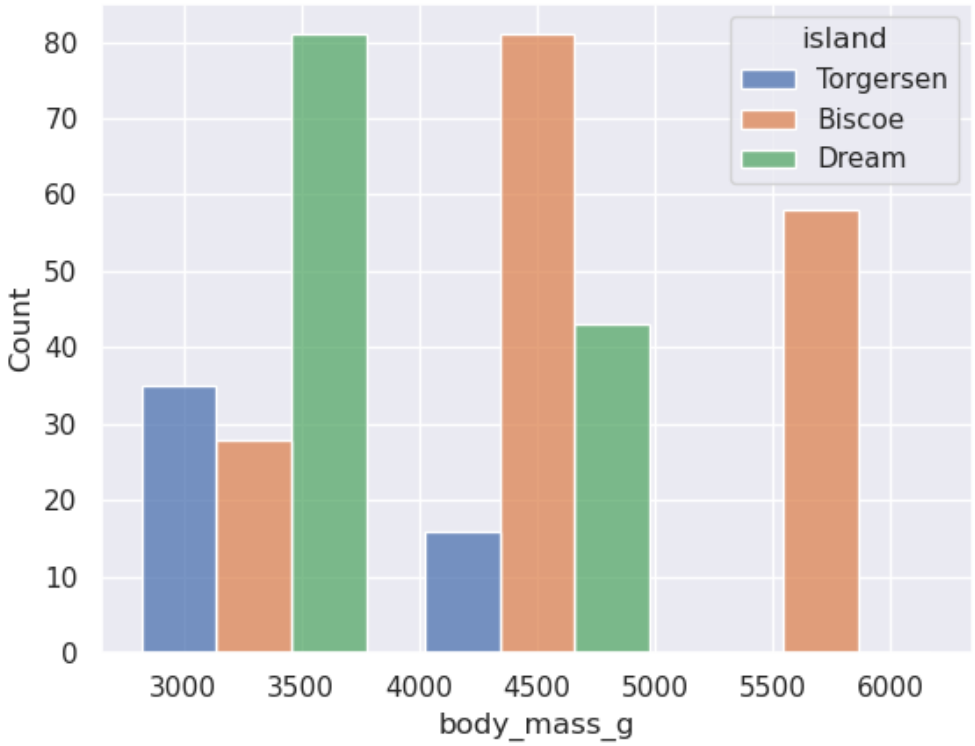


The API reference has a detailed overview of all the plotting functions and its arguments. They all come with example code as well which helps getting the plot you want right. All of the options a plotting function takes are explained thoroughly. The documentation is really good.

https://seaborn.pydata.org/api.html

5 Exercise 3

Load the penguins data set and try to reproduce the following graph using the API reference.



It's a bit of a nonsensical graph as the binning is displayed pretty badly this way, but it's just to let you see how the documentation works.

[15]:
```
penguins = sns.load_dataset("penguins")
```

[ ]:

[ ]:

[ ]:

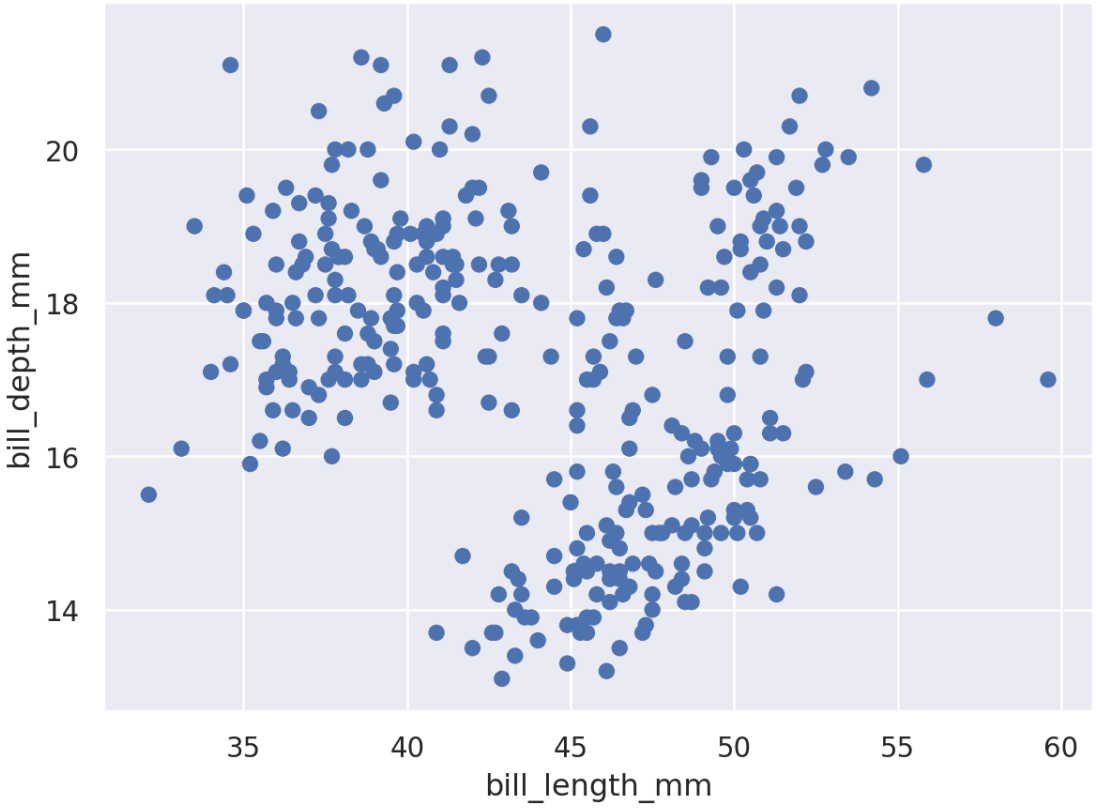
6 Additional notes

6.1 Alternative plot syntax

Seaborn has developed an additional declaritive syntax for creating plot which is a bit similar to ggplot in R. That is, you create a Plot object and then add stuff to it. It allows you to create all the plots as before but in a more logical way for some people. That might be you!

[18]:
```
import seaborn.objects as so
(
    so.Plot(penguins, x="bill_length_mm", y="bill_depth_mm")
    .add(so.Dot())
)
```

[18]:



There is a good help page on that here: https://seaborn.pydata.org/tutorial/objects_interface.html

6.2 Data structure

Another important thing to be aware of is that data in Seaborn is (mostly) expected to be "long-form data" as opposed to "wide-form data".

This is "long-form data" where every row is a single data point with multiple columns

[19]:
```
flights = sns.load_dataset("flights")
flights
```

[19]:

	year	month	passengers
0	1949	Jan	112
1	1949	Feb	118
2	1949	Mar	132
3	1949	Apr	129
4	1949	May	121
..	…	…	…
139	1960	Aug	606
140	1960	Sep	508
141	1960	Oct	461

```
142  1960   Nov         390
143  1960   Dec         432

[144 rows x 3 columns]
```

This is "wide-form data" where it's more like a spreadsheet. That is, the rows and columns are variables by themselves.

[20]:
```
wide_flights = flights.pivot(index="year", columns="month", values="passengers")
wide_flights
```

[20]:

year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	112	118	132	129	121	135	148	148	136	119	104	118
1950	115	126	141	135	125	149	170	170	158	133	114	140
1951	145	150	178	163	172	178	199	199	184	162	146	166
1952	171	180	193	181	183	218	230	242	209	191	172	194
1953	196	196	236	235	229	243	264	272	237	211	180	201
1954	204	188	235	227	234	264	302	293	259	229	203	229
1955	242	233	267	269	270	315	364	347	312	274	237	278
1956	284	277	317	313	318	374	413	405	355	306	271	306
1957	315	301	356	348	355	422	465	467	404	347	305	336
1958	340	318	362	348	363	435	491	505	404	359	310	337
1959	360	342	406	396	420	472	548	559	463	407	362	405
1960	417	391	419	461	472	535	622	606	508	461	390	432

To convert back to long-form, you can use the pandas function `melt`. I also use `reset_index` to turn the year index into a column for melt to use.

[21]:
```
pd.melt(wide_flights.reset_index(), id_vars=["year"], value_name="passengers")
```

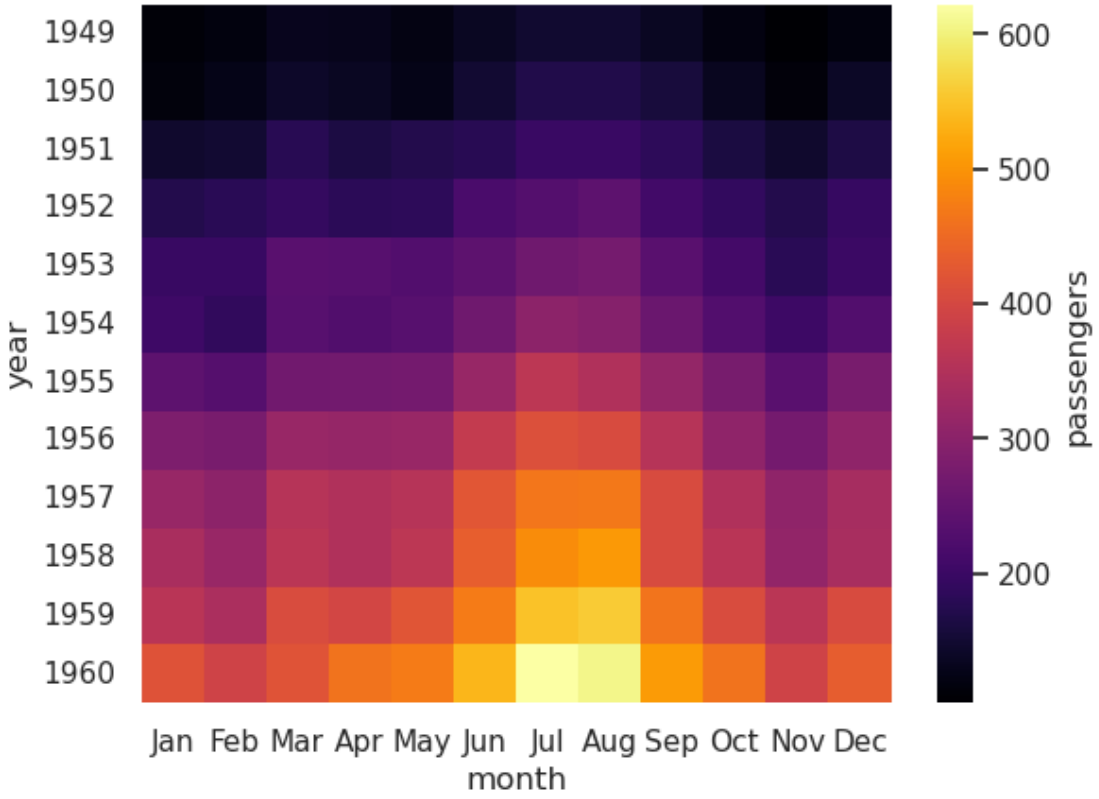
[21]:
```
      year month  passengers
0     1949   Jan         112
1     1950   Jan         115
2     1951   Jan         145
3     1952   Jan         171
4     1953   Jan         196
..     …   …          …
139   1956   Dec         306
140   1957   Dec         336
141   1958   Dec         337
142   1959   Dec         405
143   1960   Dec         432

[144 rows x 3 columns]
```

Wide-form data does have its use in Seaborn though. For example, it's great for plotting heatmaps.

[22]:
```
sns.heatmap(data=wide_flights, cmap="inferno", cbar_kws={"label": "passengers"})
```

[22]: <Axes: xlabel='month', ylabel='year'>



[ ]: