lab0-visualization-auto_mpg

September 22, 2022

1 Visualization

1.1 Topics

1. Matplotlib core framework
2. Pandas plot()
3. Seaborn statistical visualization
4. (not covered) Grammar of graphics (ggplot2 see plotnine)
5. (not covered) Interactive plotting

1.2 Resources

1. Ch 9 in Python for Data Analysis, 2nd Ed, Wes McKinney (UCalgary library and https://github.com/wesm/pydata-book)
2. Ch 4 in Python Data Science Handbook, Jake VanderPlas (Ucalgary library and https://github.com/jakevdp/PythonDataScienceHandbook)
3. Fundamentals of Data Visualization, Claus O. Wilke (Ucalgary library and https://serialmentor.com/dataviz/index.html)
4. Overview by Jake VanderPlas https://www.youtube.com/watch?v=FytuB8nFHPQ

1.3 Matplotlib

> Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.
> Matplotlib tries to make easy things easy and hard things possible.
> For simple plotting the pyplot module provides a MATLAB-like interface

https://matplotlib.org

Importing matplotlib looks like this

[ ]:
```
%matplotlib inline

import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

1.3.1 Two interfaces

There are two ways to interact with Matplot lib: a Matlab style and an object oriented style interface.

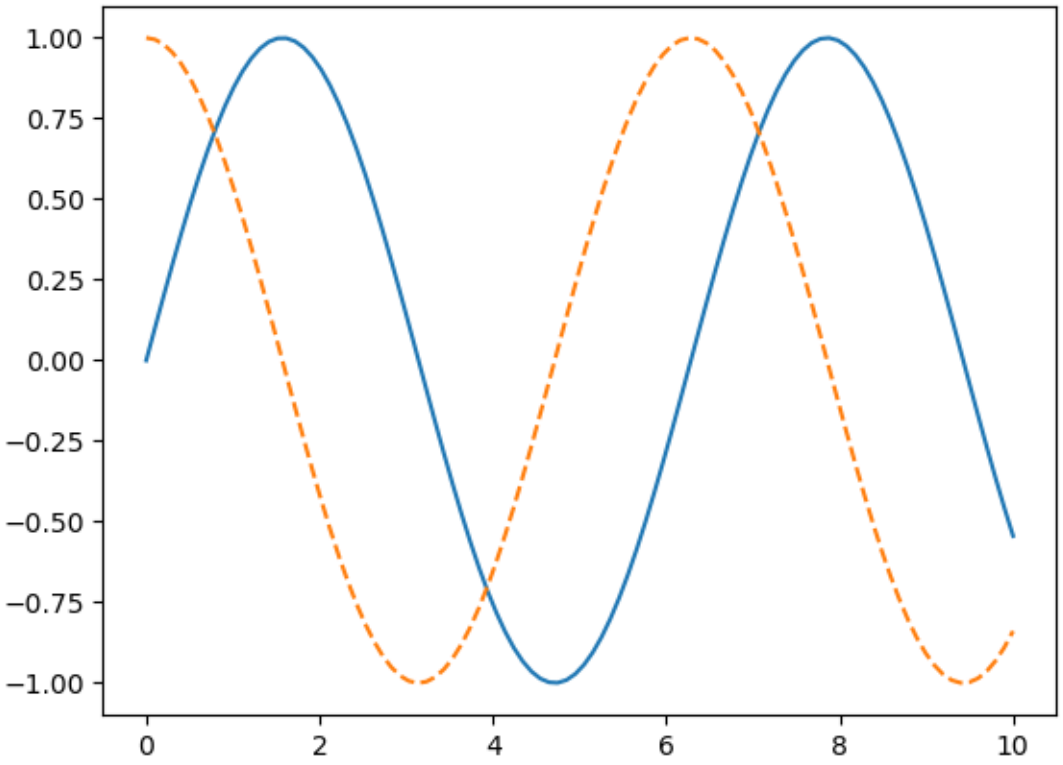
See Ch 4 in Python Data Science Handbook, Jake VanderPlas

- Two Interfaces for the Price of One, pp. 222
- Matplotlib Gotchas, pp. 232

1.3.2 Matlab style interface

[ ]:
```
x = np.linspace(0, 10, 100)

plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--');
```

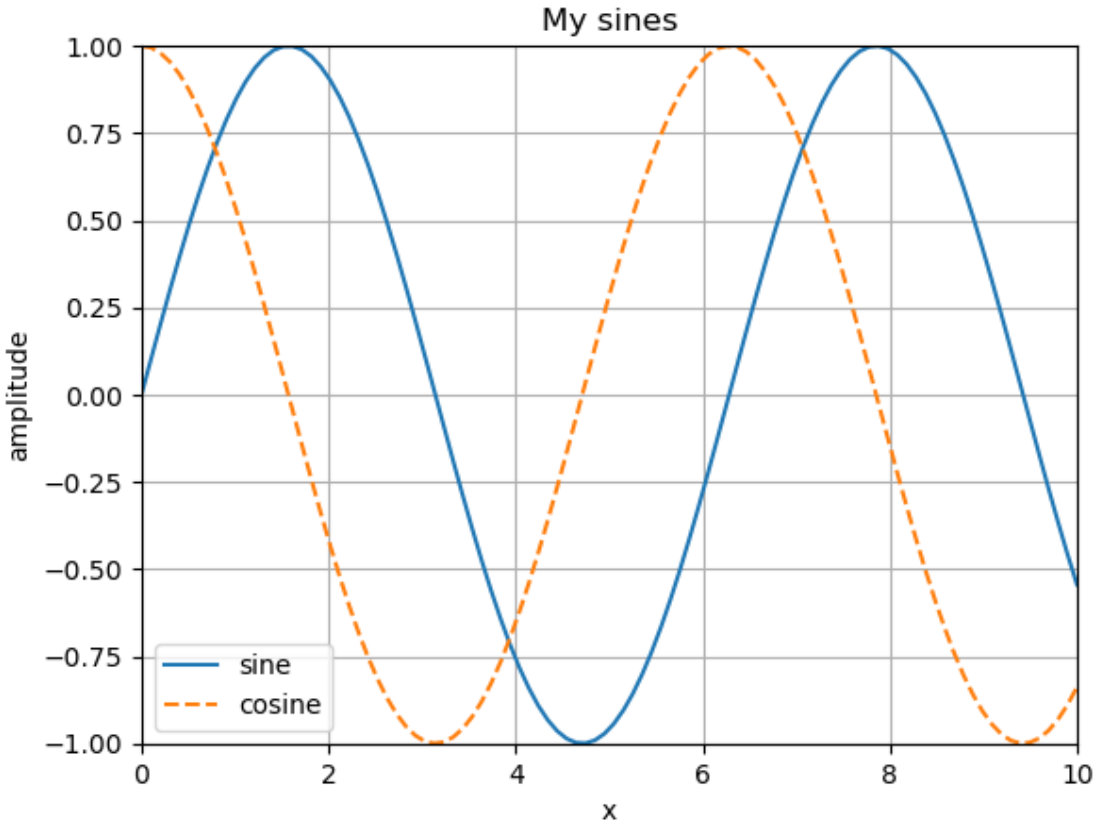


Adding decorations to the plot is done by repeatatly calling functions on the imported `plt` module. All calls within the cell will be applied to the current figure and axes.

[ ]:
```
plt.plot(x, np.sin(x), '-', label='sine')
plt.plot(x, np.cos(x), '--', label ='cosine')

plt.xlim([0, 10])
```

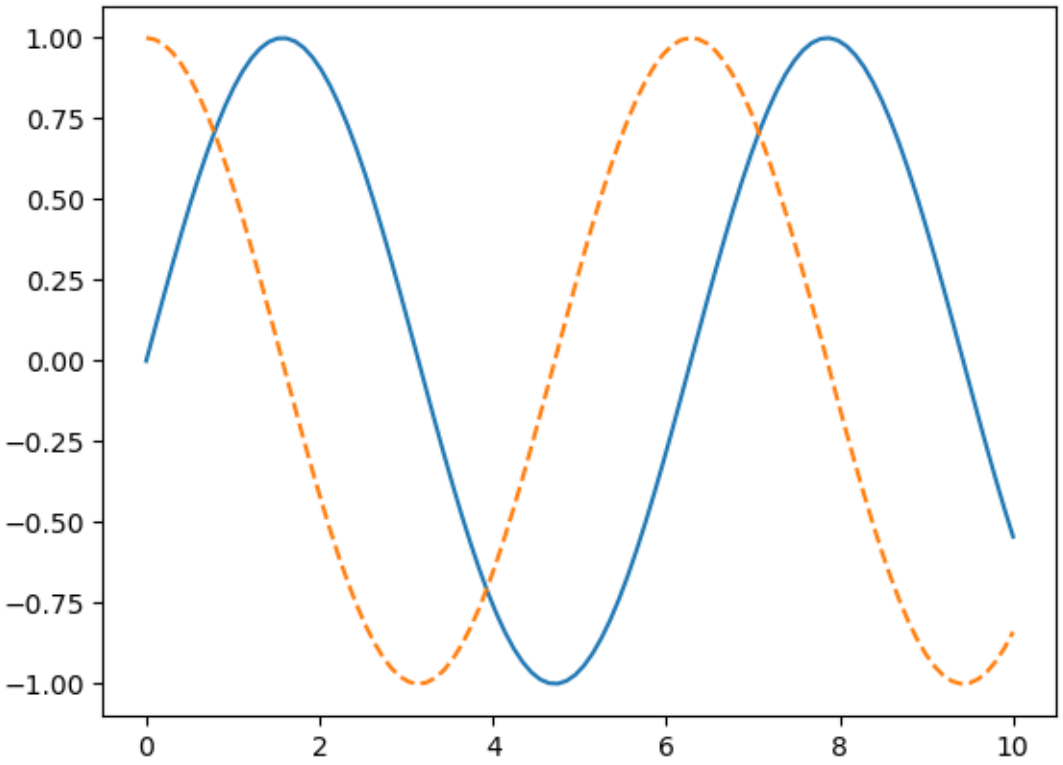
```
plt.ylim([-1, 1])
plt.xlabel('x')
plt.ylabel('amplitude')
plt.title('My sines')
plt.grid()
plt.legend();
```



1.3.3 Object oriented interface

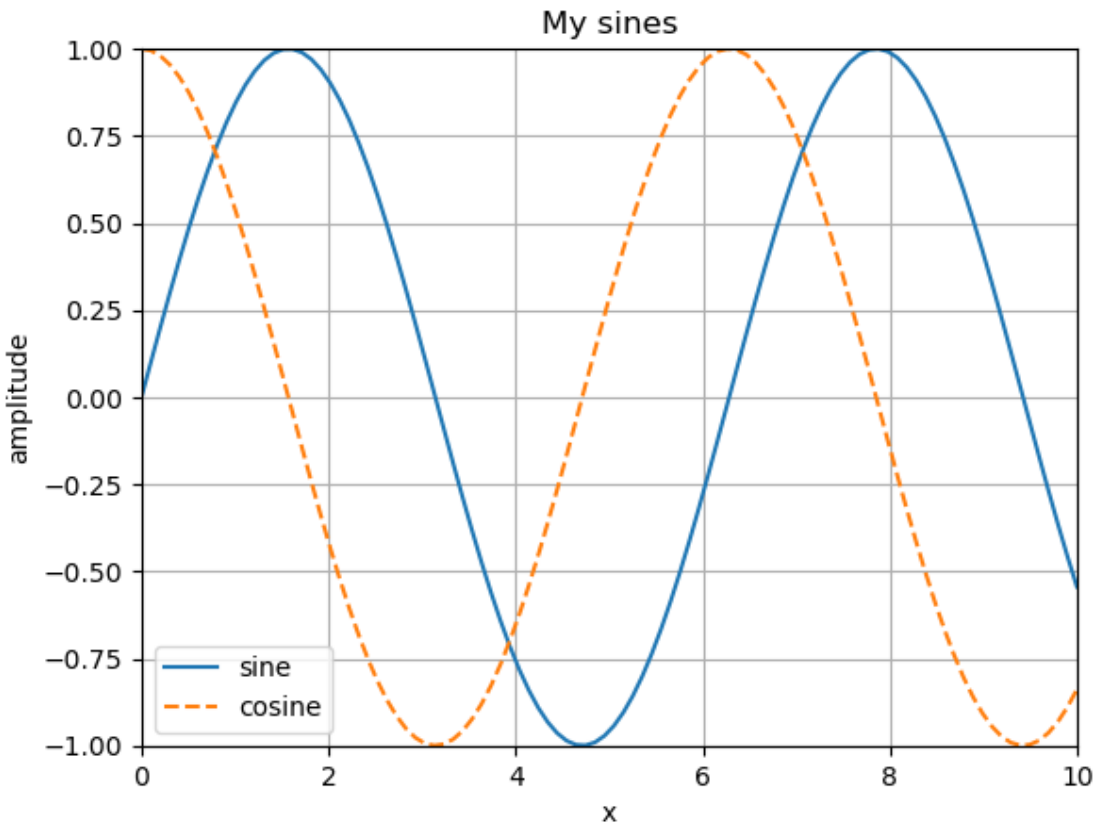
With this interface, you first create a figure and an axes object, then call their methods to change the plot.

[ ]:
```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, np.sin(x), '-')
ax.plot(x, np.cos(x), '--');
```



[ ]:
```
fig = plt.figure()
ax = plt.axes()
ax.plot(x, np.sin(x), '-', label='sine')
ax.plot(x, np.cos(x), '--', label ='cosine')

ax.set(xlim=[0, 10], ylim=[-1, 1],
       xlabel='x', ylabel='amplitude',
       title='My sines');
ax.grid()
ax.legend();
```



1.3.4 Save to file

With the figure object at hand, we can save to file

[ ]:
```
fig.savefig('sines.pdf')
```

[ ]:
```
!ls *.pdf
```

```
sines.pdf
```

1.4 Plotting with pandas

> We use the standard convention for referencing the matplotlib API … We provide the basics in pandas to easily create decent looking plots.

https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html

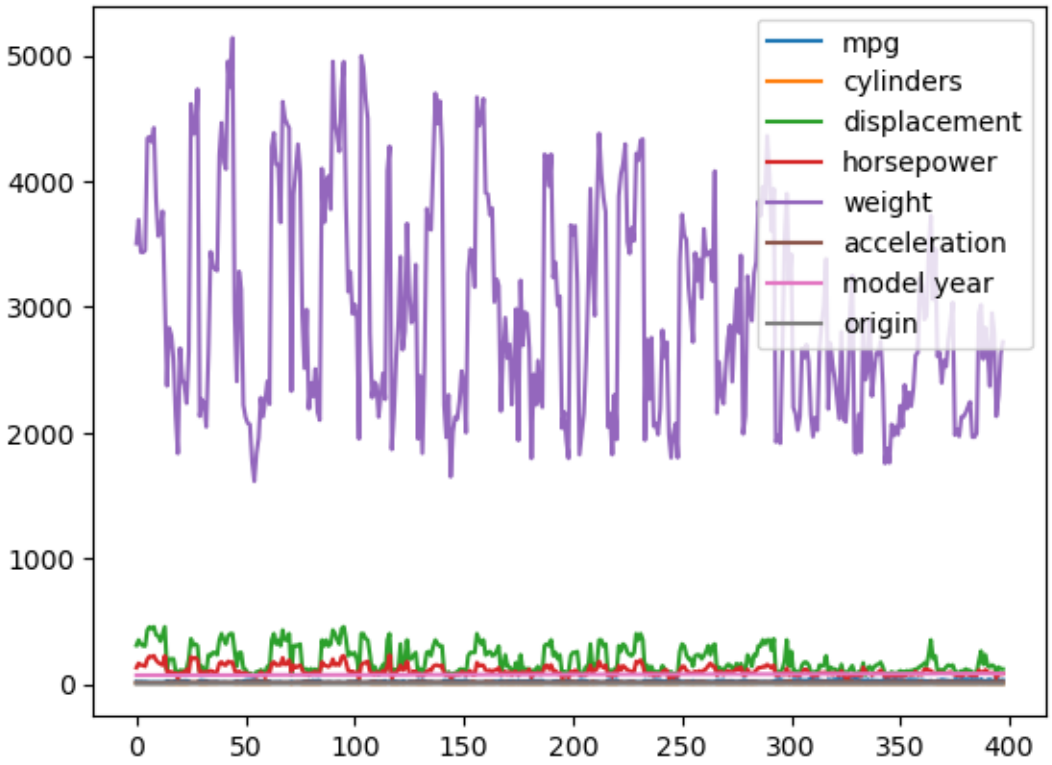
Let's load the auto mpg dataset

[ ]:
```
data = pd.read_csv('auto-mpg.csv', na_values='?')
```

Plotting all columns, works, but does not provide a lot of insight.

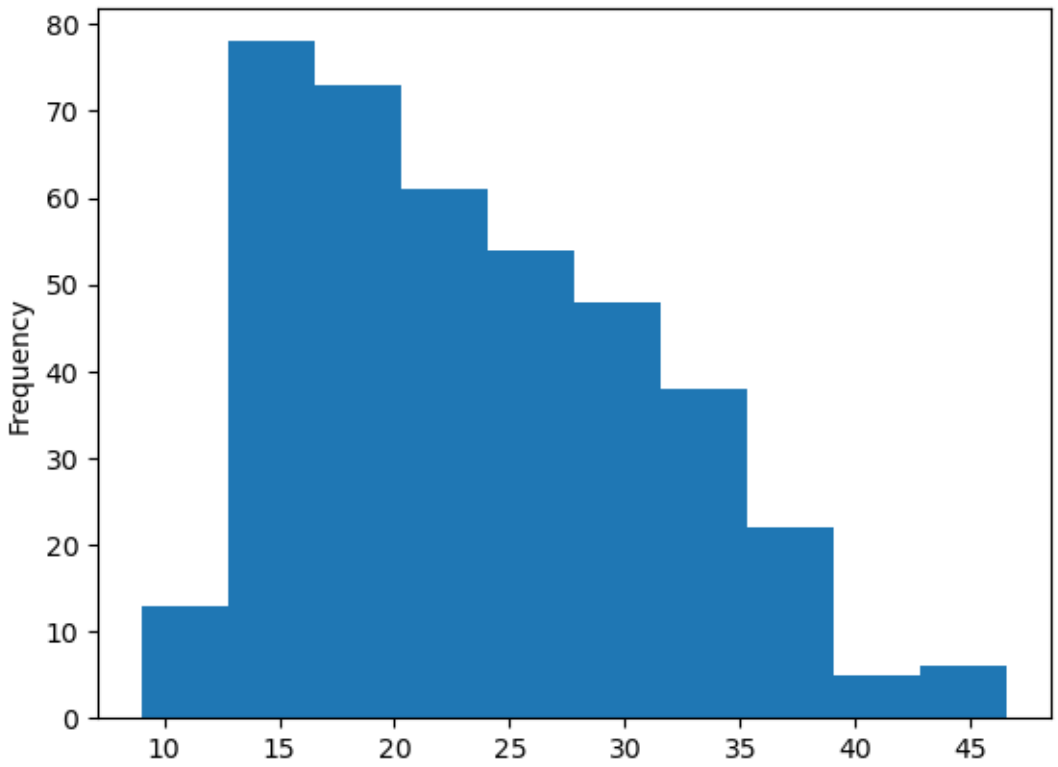
[ ]:
```
data.plot()
```

[ ]: <AxesSubplot:>



Let's look at the mpg distribution (a histogram)

[ ]:
```
data['mpg'].plot.hist();
```



How many samples from each origin do we have?

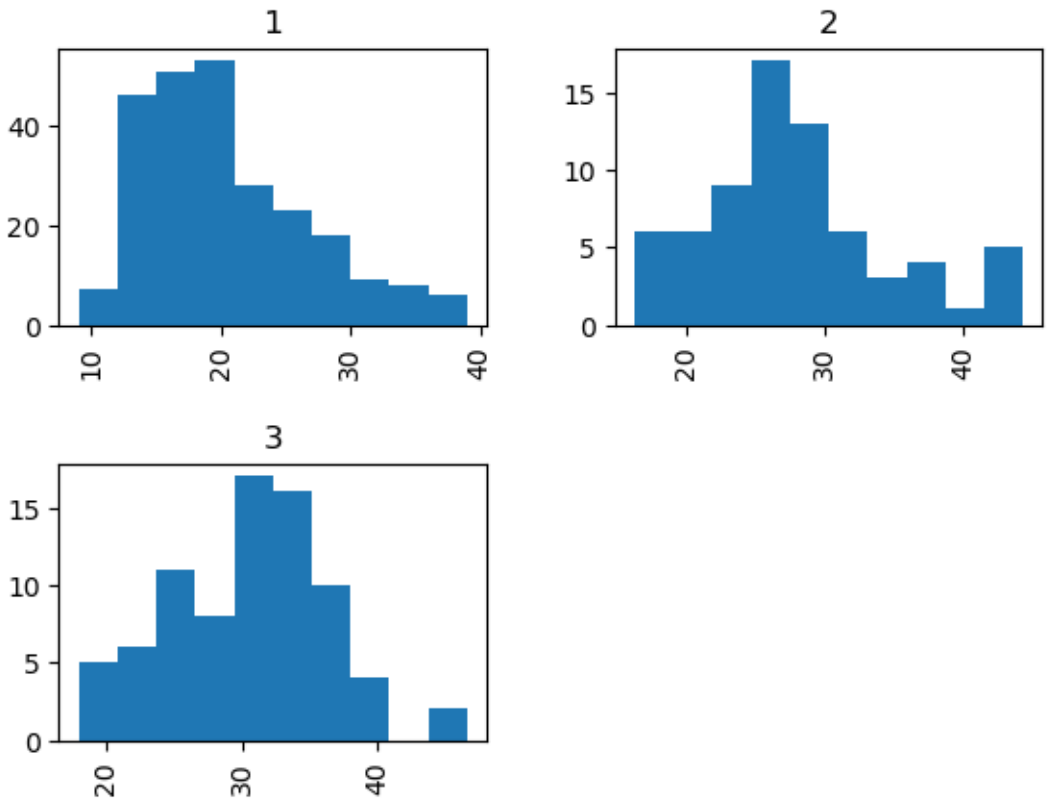
[ ]:
```
data.origin.value_counts()
```

[ ]:
```
1    249
3     79
2     70
Name: origin, dtype: int64
```

Notice that we accessed the origin column with dot notation. This can be done whenever the column name is 'nice' enough to be a python variable name.

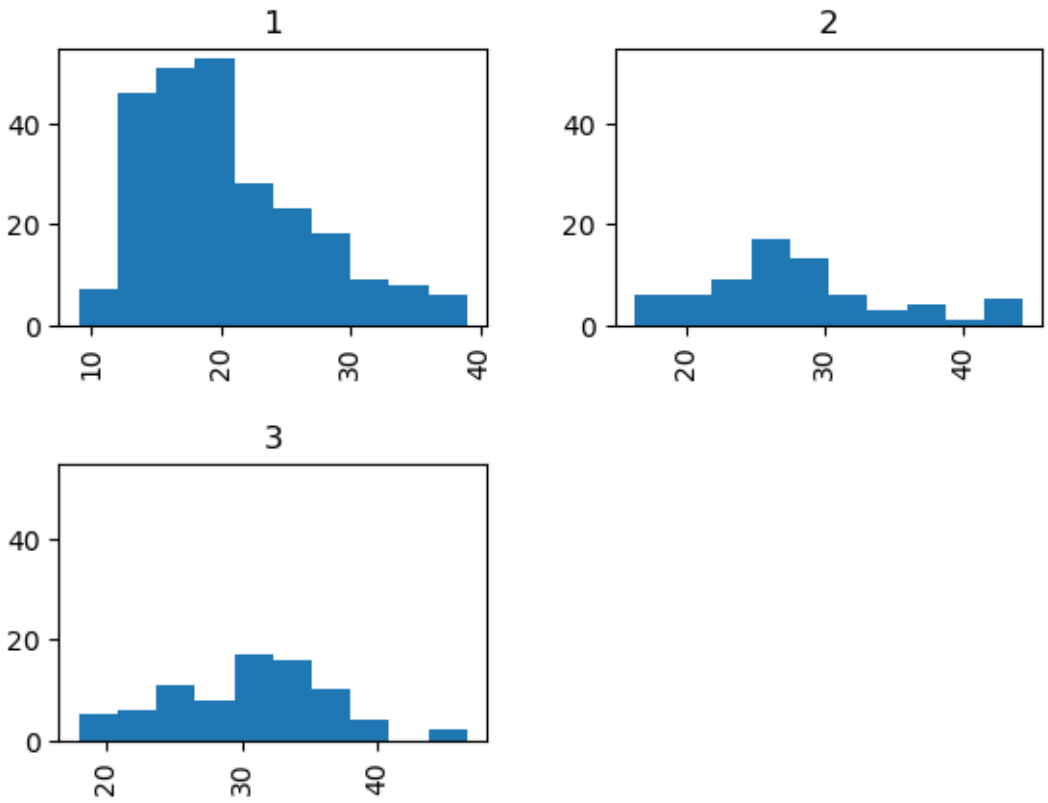
Do we have similar mpg based on car origin?
Plotting three histograms for each origin directly form the dataframe:

[ ]:
```
axs = data.hist(column='mpg', by='origin')
```



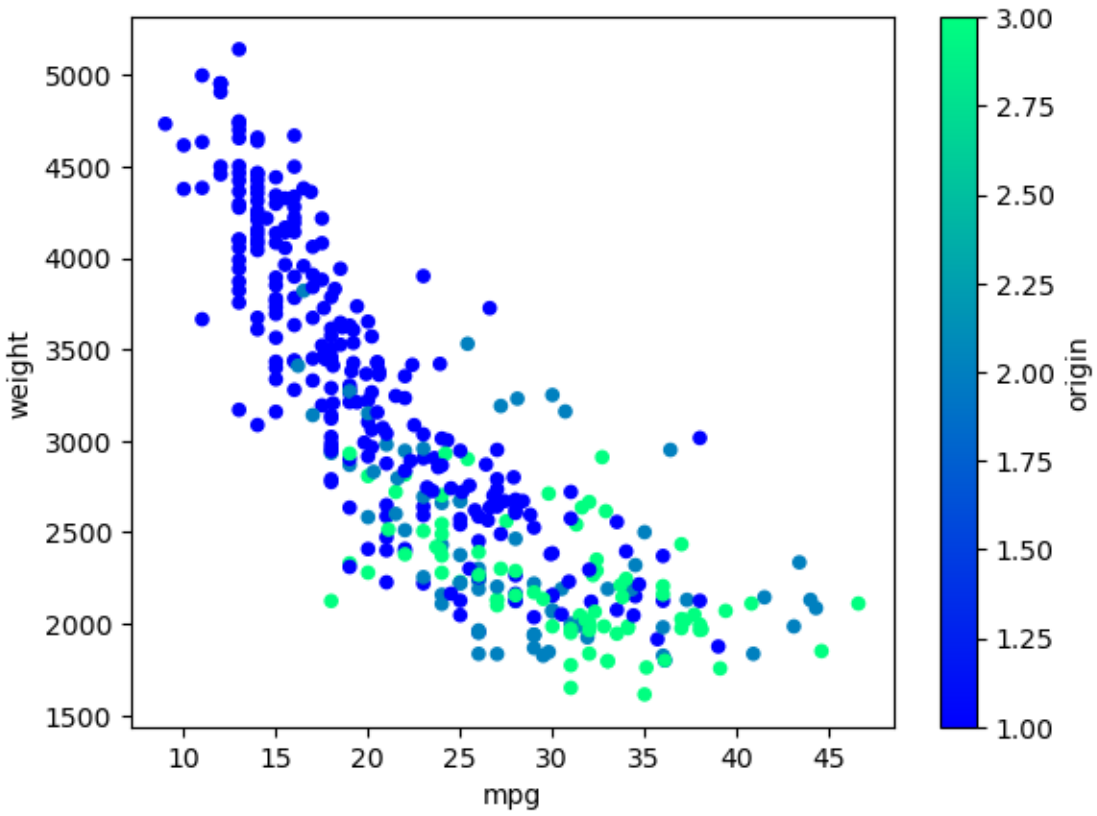
To format this plot, we can work on the axes (array) that is returned by the plot call. We use Matplotlib object oriented interface methods to do this

[ ]:
```
axs = data.hist(column='mpg', by='origin')
axs[0,0].set(title='1', ylim=[0, 55])
axs[0,1].set(title='2', ylim=[0, 55])
axs[1,0].set(title='3', ylim=[0, 55]);
```



Is mpg and car weight correlated? Maybe it is different for car origin?
Let's have a look with a scatter plot.

[ ]:
```
data.plot.scatter('mpg', 'weight', c='origin', colormap='winter');
```



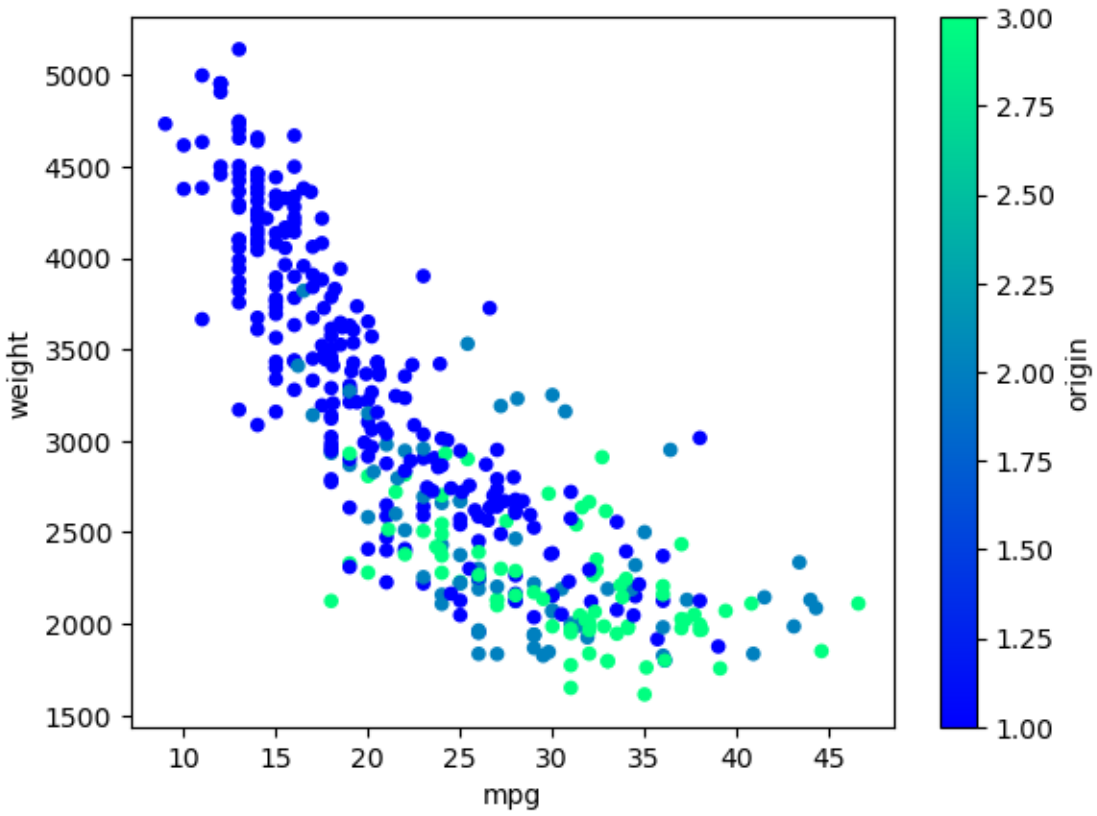
According to:
https://stackoverflow.com/questions/43578976/pandas-missing-x-tick-labels

the missing x-labels are a pandas bug.

Workaraound is to create axes prior to calling plot.

This bug appears to have been fixed, but this cell and code remains.

[ ]:
```
fig, ax = plt.subplots()
data.plot.scatter('mpg', 'weight', c='origin', colormap='winter', ax=ax);
```

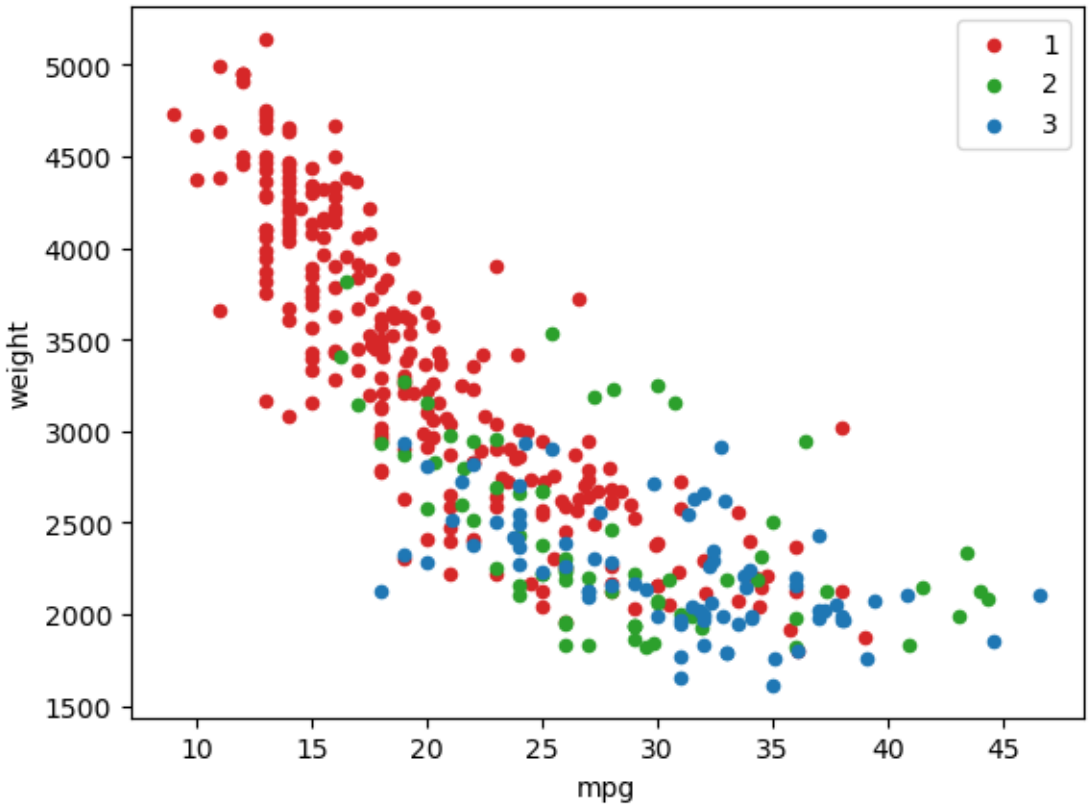


It is a bit annoying that there is a colorbar, we know origin is categorical.

One way to avoid the colorbar is to loop over the categories and assign colors based on the category.

See: https://stackoverflow.com/questions/26139423/plot-different-color-for-different-categorical-levels-using-matplotlib

[ ]:
```
colors = {1: 'tab:red', 2: 'tab:green', 3: 'tab:blue'}
fig, ax = plt.subplots()
for key, group in data.groupby(by='origin'):
    group.plot.scatter('mpg', 'weight', c=colors[key], label=key, ax=ax);
```



1.5 Seaborn

Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

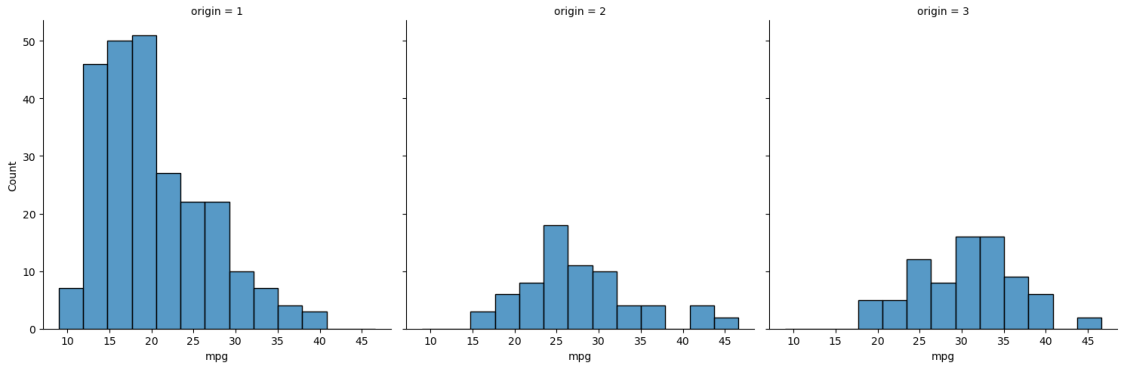
http://seaborn.pydata.org/index.html

Seaborn is usually imported as `sns`

[ ]:
```
import seaborn as sns
```

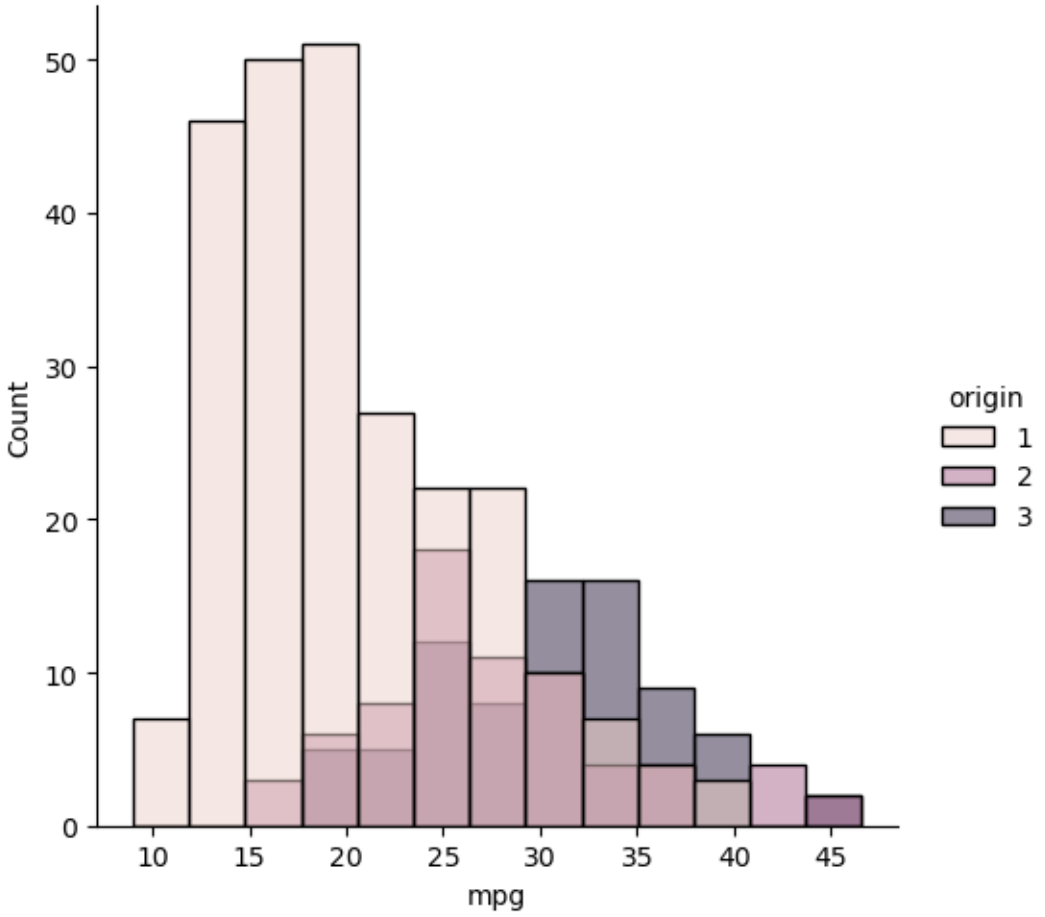
Let's re-create the histograms by origin with seaborn with the figure level `displot()` function.

[ ]:
```
# Use origin to split mpg into columns
sns.displot(x='mpg', col='origin', data=data);
```



We can display the counts in the same plot, one on top of the other.

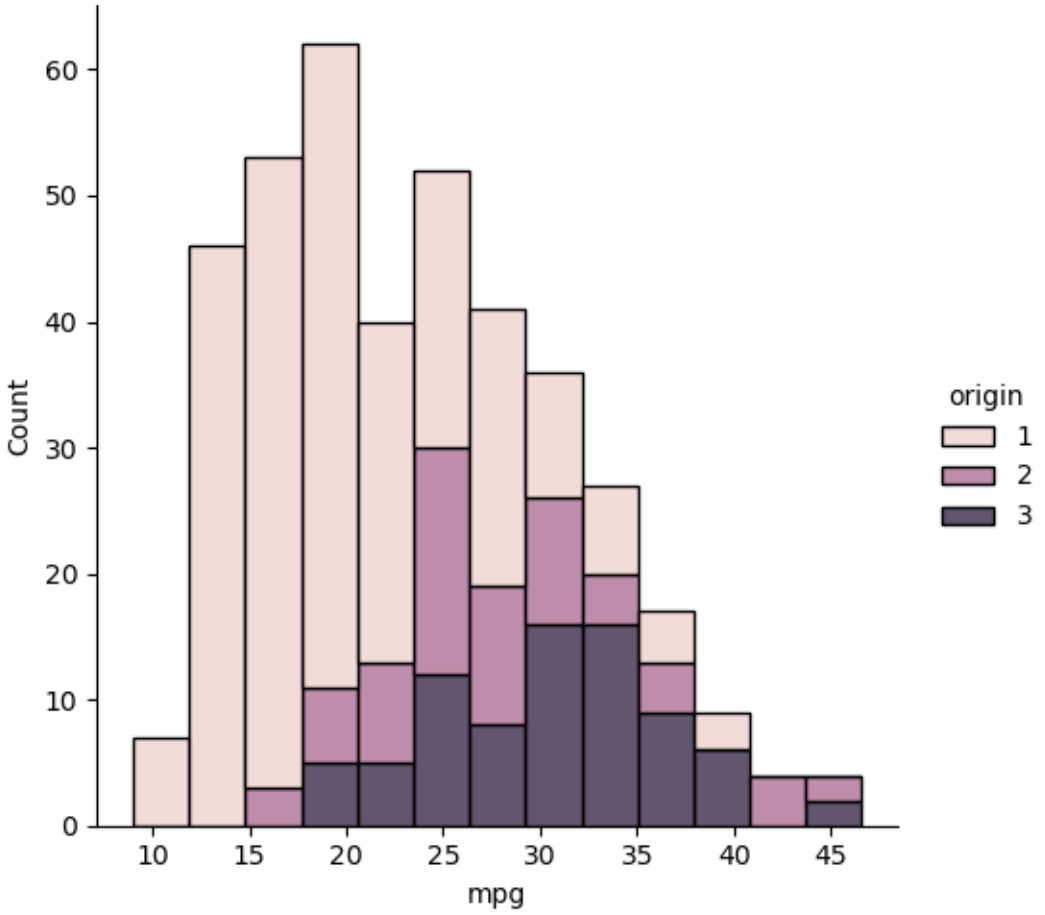
[ ]:
```
# Use origin to color (hue) in the same plot
sns.displot(x='mpg', hue='origin', data=data);
```



To have an idea of the split between origin 1, 2, and 3, we can stack the counts, adding up to total.

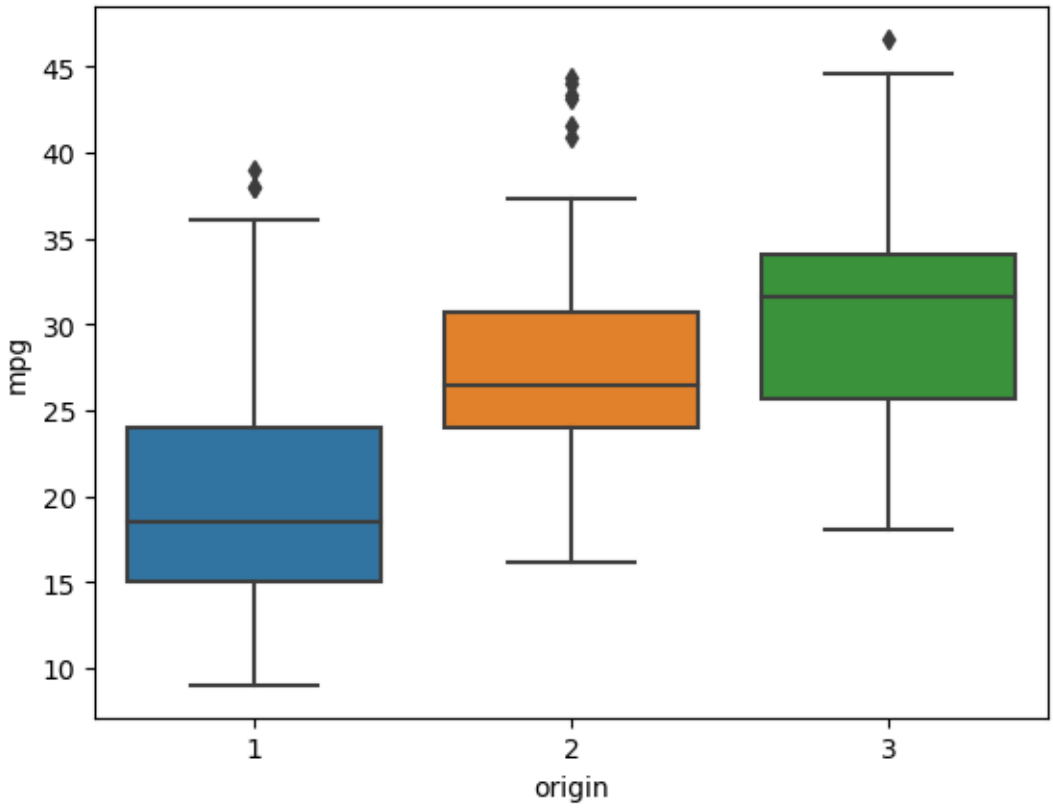
[ ]:
```
sns.displot(x='mpg', hue='origin', data=data, multiple='stack');
```

```
/home/graeme/anaconda3/envs/ensf-611/lib/python3.9/site-
packages/seaborn/distributions.py:254: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either
`df[df.columns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i,
newvals)`
  baselines.iloc[:, cols] = (curves
```



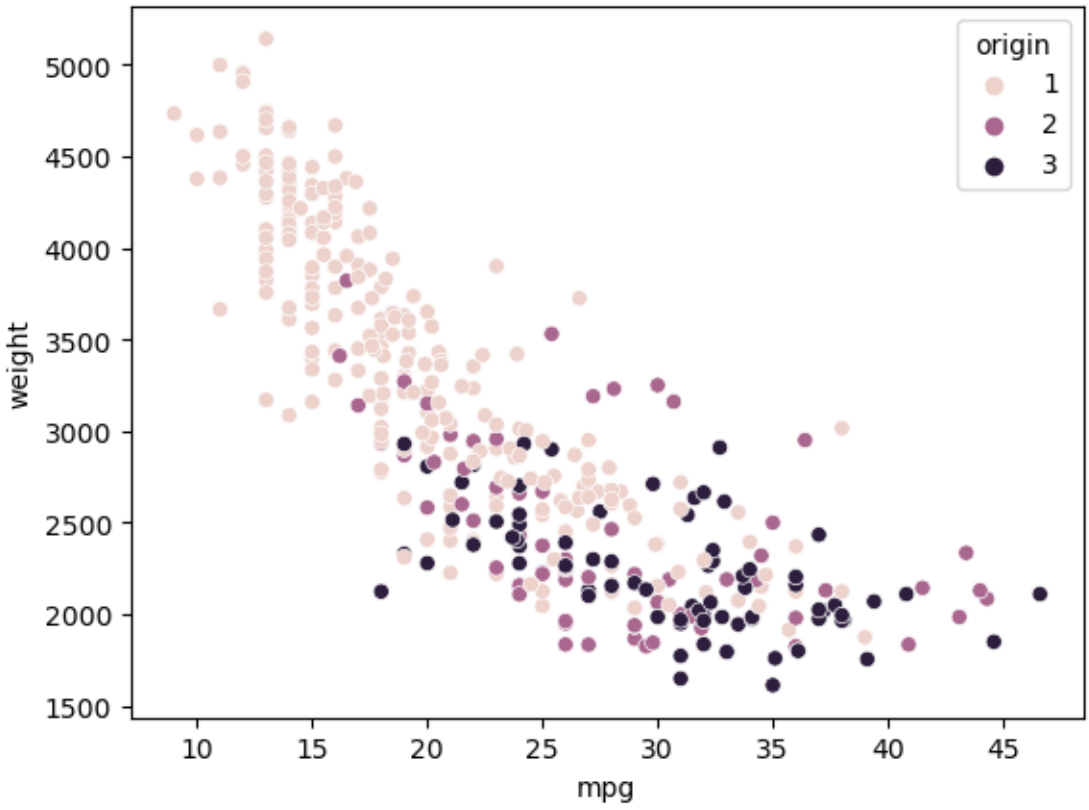
We can look at the differences in mpg with a boxplot too

[ ]:
```
sns.boxplot(x='origin', y='mpg', data=data);
```



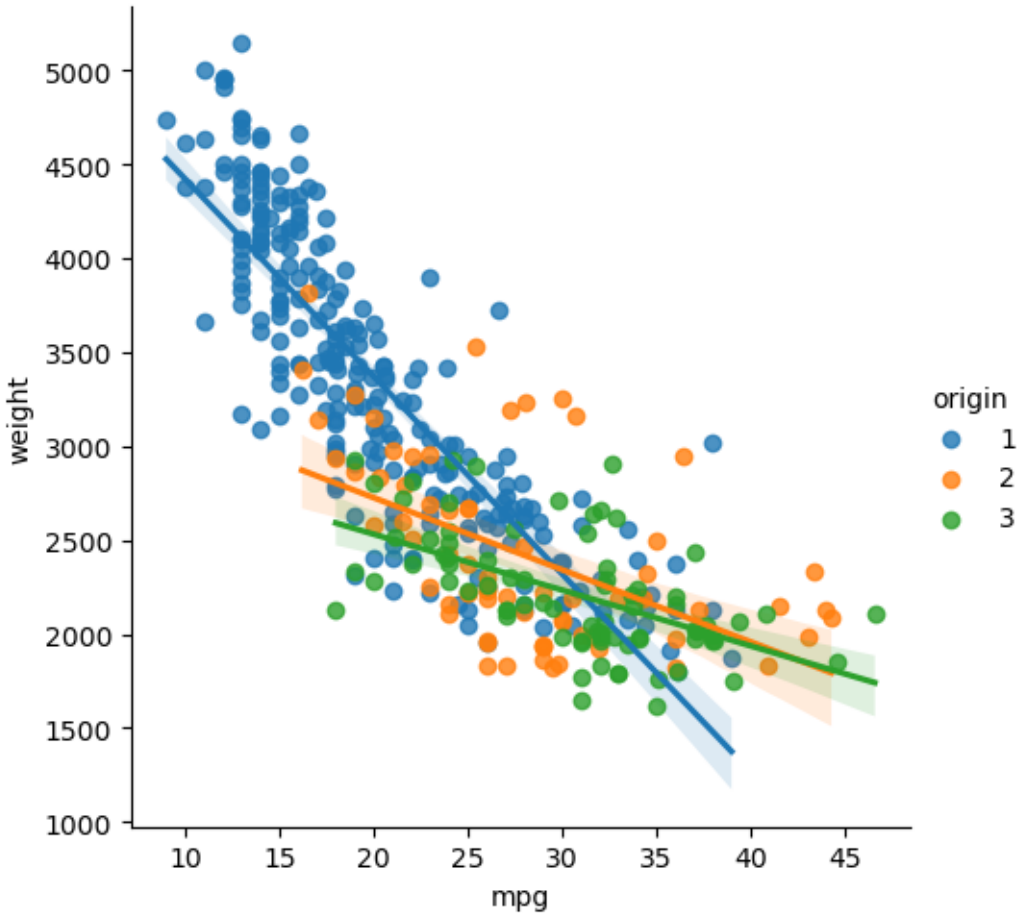
Let's re-create the scatter plot to see if mpg and car weight are correlated by origin.

[ ]:
```
ax = sns.scatterplot(x='mpg', y='weight', data=data, hue='origin')
```



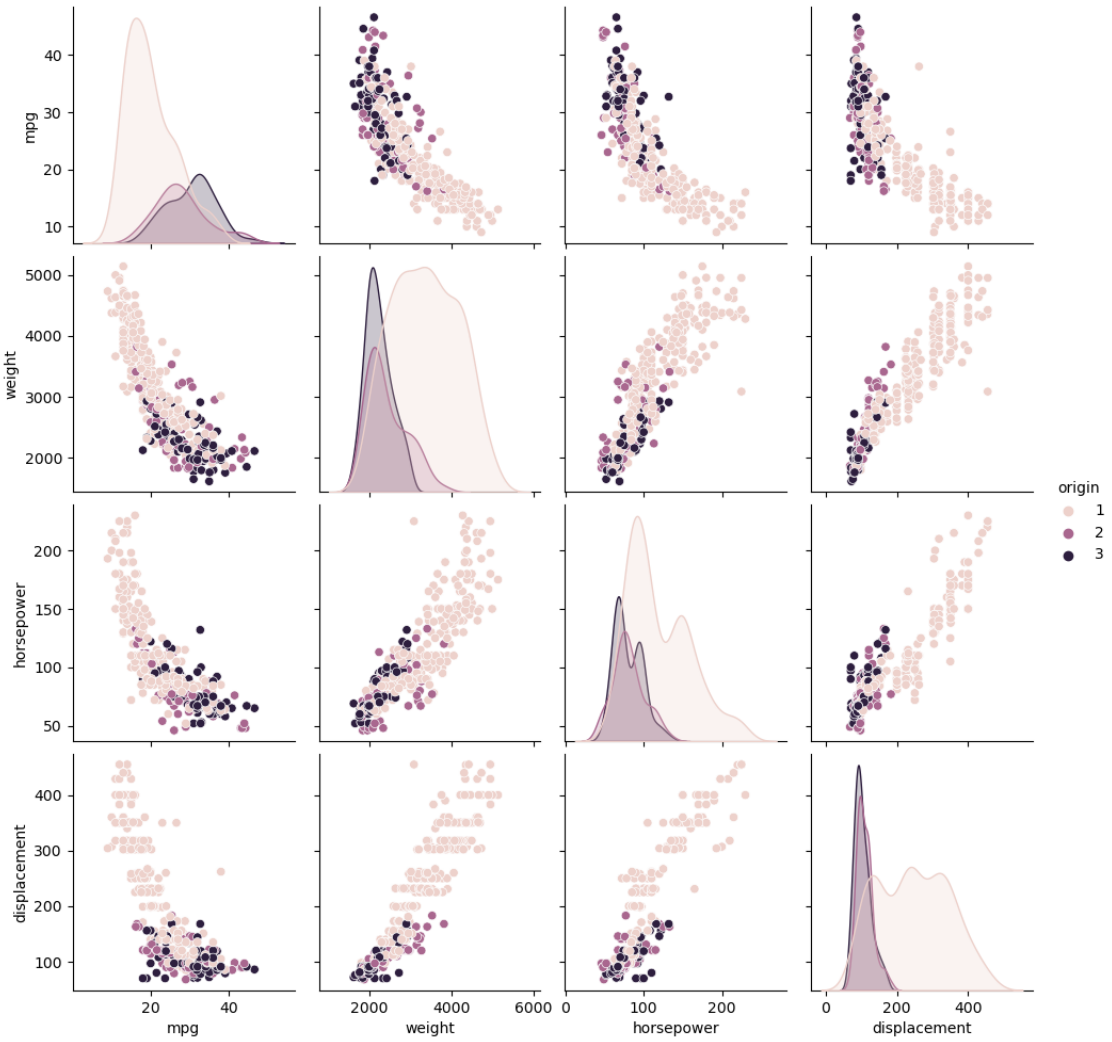
Adding a regression line helps with visualizing the relationship

[ ]:
```
ax = sns.lmplot(x='mpg', y='weight', data=data, hue='origin')
```



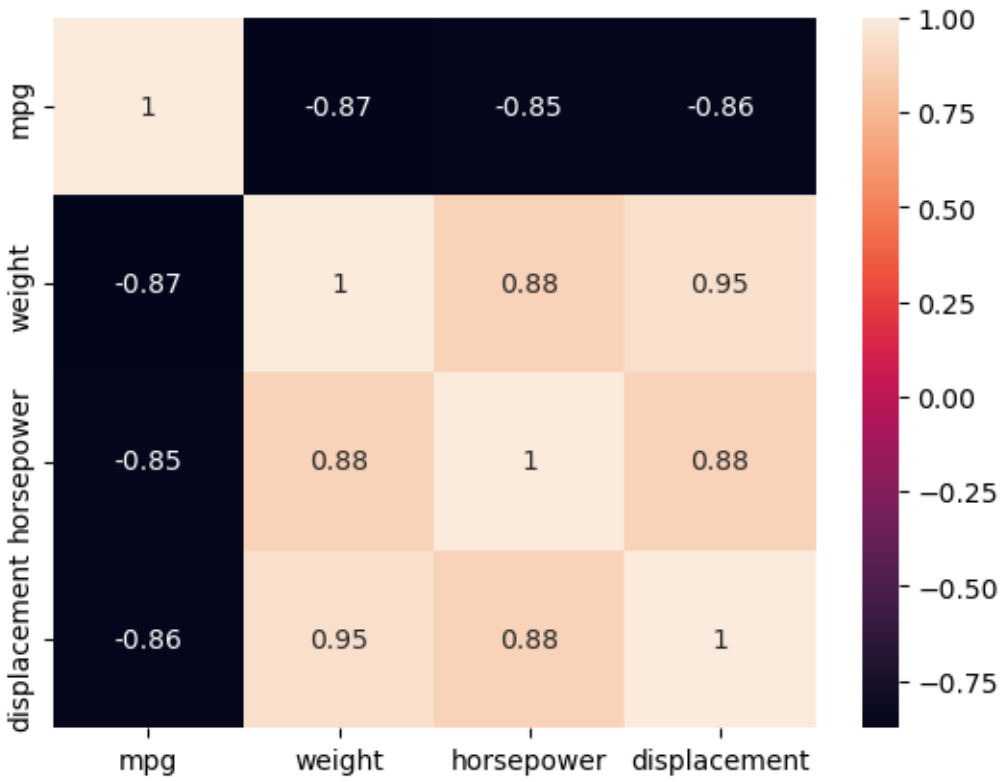
Maybe there are other correlations in the data set. Pairplot is a great way to get an overview

[ ]:
```
sns.pairplot(data, vars=['mpg', 'weight', 'horsepower', 'displacement'],␣
↪hue='origin');
```



As an alternative, we can visualize the correlation matrix as a heatmap

[ ]:
```
g = sns.heatmap(data[['mpg', 'weight', 'horsepower', 'displacement']].
↪corr(method='spearman'),
                annot=True)
```



There are nice tutorials on the Seaborn website, be sure to check these out.

[ ]: