Assignment 0

by Ling Lee
UCID: 10085095
lplee@ucalgary.ca

ENSF 611 – Machine Learning for Software Engineers
Professor Vikram Kumar
University of Calgary
Calgary, Alberta
September 22nd 2022

Screenshot

Lab0-Pandas-auto_mpg.ipynb

Load data from file

Most often data will come from somewhere, often csv files, and using `pd.read_csv()` will allow smooth creation of DataFrames.

Let's load that same heart-attack.csv that we used in Numpy before:

In [31]:
```
auto_data_df = pd.read_csv("auto-mpg.data", delim_whitespace=True, header=None)
auto_data_df.columns =["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "model year", "origin", "
```

After loading data, it is good practice to check what we have. Usually, the sequences is:

1. Check dimension
2. Peek at the first rows
3. Get info on data types and missing values
4. Summarize columns

In [20]:
```
# Check dimension (rows, columns)
auto_data_df.shape
```

Out[20]:
```
(398, 9)
```

In [21]:
```
# Peek at the first rows
auto_data_df.head()
```

Out[21]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

In [22]:
```
# Column names are
auto_data_df.columns
```

Out[22]:
```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model year', 'origin', 'car name'],
      dtype='object')
```

In [32]:
```
# Get info on data types and missing values
auto_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    398 non-null    object
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car name      398 non-null    object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

Summarize values

What is the mean, std, min, max in each column?

In [24]:
```
auto_data_df.mean()
```

```
C:\Users\lingl\AppData\Local\Temp\ipykernel_6192\3221062464.py:1: FutureWarning: Dropping of nuisance columns in DataFrame redu
ctions (with 'numeric_only=None') is deprecated; in a future version this will raise TypeError.  Select only valid columns befo
re calling the reduction.
  auto_data_df.mean()
```

Out[24]:
```
mpg               23.514573
cylinders          5.454774
displacement     193.425879
weight          2970.424623
acceleration      15.568090
model year        76.010050
origin             1.572864
dtype: float64
```

In [33]:

```
# where are the other columns? Check data types
auto_data_df.dtypes
```

Out[33]:

```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight          float64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

Notice that many columns are of type object, which is not a number. Maybe this has to do with missing values? We know from peeking at the first rows that there are '?' values in there. Let's replace these with the string NaN for not-a-number.

In [34]:

```
# replace '?' with 'NaN'
auto_data_df = auto_data_df.replace({'?': 'NaN'})
auto_data_df.head()
```

Out[34]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino

Pandas knows that 'NaN' probably means that numbers are missing. Now we can convert the data type from object to float

In [35]:

```
# convert dtypes
auto_data_df[["cylinders", "horsepower", "model year", "origin"]] = auto_data_df[["cylinders", "horsepower", "model year", "
auto_data_df.dtypes
```

Out[35]:

```
mpg             float64
cylinders       float64
displacement    float64
horsepower      float64
weight          float64
acceleration    float64
model year      float64
origin          float64
car name         object
dtype: object
```

We could have loaded the data with the `na_values` argument to indicate that '?' means missing number:

In [37]:
```
auto_data_df = pd.read_csv("auto-mpg.data", delim_whitespace=True, header=None, na_values="?")
auto_data_df.columns = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "model year", "origin",
auto_data_df.dtypes
```

Out[37]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower      float64
weight          float64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

This worked nicely. Now we can describe all columns, meaning printing basic statistics. Note that by default Pandas ignores NaN, whereas Numpy does not.

In [38]:
```
auto_data_df.describe() # ignores NaN
```

Out[38]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

We could be interested by these statistics in each of the genders. To get these, we first group values by gender, then ask for the description. We will only look at age for clarity

In [46]:
```
auto_data_df.groupby(by='car name').describe().mpg
```

Out[46]:

car name	count	mean	std	min	25%	50%	75%	max
amc ambassador brougham	1.0	13.00	NaN	13.0	13.000	13.00	13.000	13.0
amc ambassador dpl	1.0	15.00	NaN	15.0	15.000	15.00	15.000	15.0
amc ambassador sst	1.0	17.00	NaN	17.0	17.000	17.00	17.000	17.0
amc concord	2.0	21.85	3.464823	19.4	20.625	21.85	23.075	24.3
amc concord d/l	1.0	18.10	NaN	18.1	18.100	18.10	18.100	18.1
...	...	...	...	...	...	...	...	...
vw dasher (diesel)	1.0	43.40	NaN	43.4	43.400	43.40	43.400	43.4
vw pickup	1.0	44.00	NaN	44.0	44.000	44.00	44.000	44.0
vw rabbit	2.0	35.25	8.838835	29.0	32.125	35.25	38.375	41.5
vw rabbit c (diesel)	1.0	44.30	NaN	44.3	44.300	44.30	44.300	44.3
vw rabbit custom	1.0	31.90	NaN	31.9	31.900	31.90	31.900	31.9

305 rows × 8 columns

Find NaNs

How many NaNs in each column?

We can ask which entries are null, which produces a boolean array

In [41]:
```
auto_data_df.isnull()
```

Out[41]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
393	False	False	False	False	False	False	False	False	False
394	False	False	False	False	False	False	False	False	False
395	False	False	False	False	False	False	False	False	False
396	False	False	False	False	False	False	False	False	False
397	False	False	False	False	False	False	False	False	False

Applying `sum()` to this boolean array will count the number of `True` values in each column

In [42]:
```
auto_data_df.isnull().sum()
```

Out[42]:
```
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model year      0
origin          0
car name        0
dtype: int64
```

We get complementary information from `info()`

In [44]:
```
auto_data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           398 non-null    float64
 1   cylinders     398 non-null    int64
 2   displacement  398 non-null    float64
 3   horsepower    392 non-null    float64
 4   weight        398 non-null    float64
 5   acceleration  398 non-null    float64
 6   model year    398 non-null    int64
 7   origin        398 non-null    int64
 8   car name      398 non-null    object
dtypes: float64(5), int64(3), object(1)
memory usage: 28.1+ KB
```

We can fill (replace) these missing values, for example with the minimum value in each column

In [47]:
```
auto_data_df.fillna(auto_data_df.min()).describe()
```

Out[47]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin
count	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	103.587940	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	38.859575	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	92.000000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	125.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

Count unique values (a histogram)

We finish off, with our good friend the histogram

In [48]:
```
auto_data_df['mpg'].value_counts()
```

Out[48]:
```
13.0    20
14.0    19
18.0    17
15.0    16
26.0    14
        ..
31.9     1
16.9     1
18.2     1
22.3     1
44.0     1
Name: mpg, Length: 129, dtype: int64
```

In [ ]:
```

```

Lab0-Visualization-auto_mpg.ipynb

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

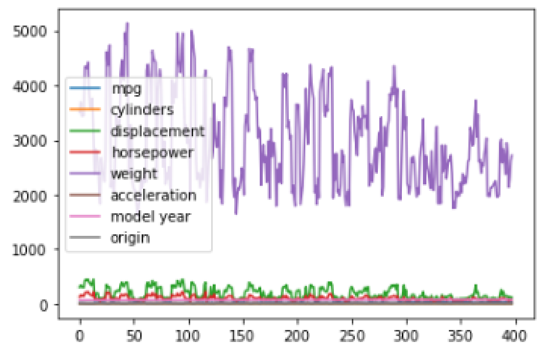
Let's load the heart attack dataset

In [3]:
```
auto_data_df = pd.read_csv("auto-mpg.data", delim_whitespace=True, header=None, na_values="?")
auto_data_df.columns = ["mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration"
```

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

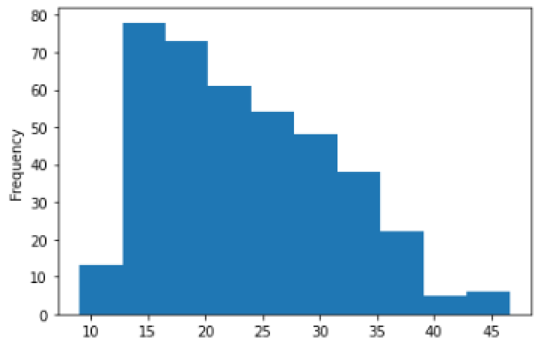
In [4]:
```
auto_data_df.plot()
```

Out[4]: <AxesSubplot:>



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

In [5]:
```
auto_data_df['mpg'].plot.hist();
```



How many male and female samples do we have?

In [6]:
```
auto_data_df.origin.value_counts()
```

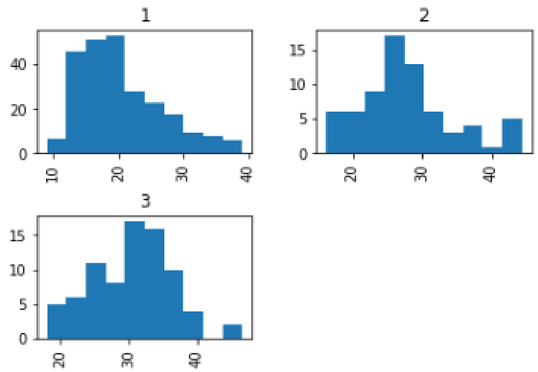
Out[6]:
```
1    249
3     79
2     70
Name: origin, dtype: int64
```

Notice that we accessed the gender 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 ages in females and males?
Plotting two histograms for each gender side beside directly form the dataframe:

In [7]:

```
axs = auto_data_df.hist(column='mpg', by='origin')
```



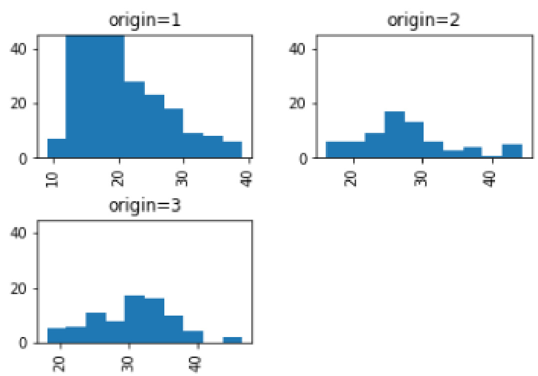
To format this plot, we can work on the axes (array) that is returned by the plot call. We use Matplotlib object oriented interface methods to do this

In [18]:

```
axs = auto_data_df.hist(column='mpg', by='origin')
axs[0][0].set(title='origin=1', ylim=[0, 45])
axs[0][1].set(title='origin=2', ylim=[0, 45])
axs[1][0].set(title="origin=3", ylim=[0, 45])
```

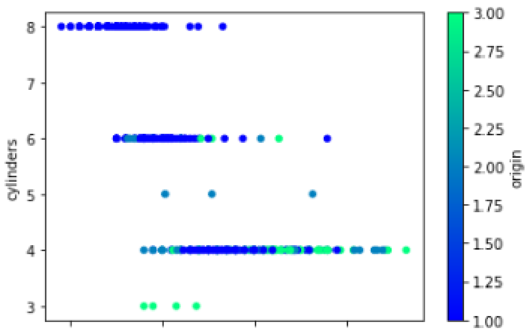
Out[18]:

```
[Text(0.5, 1.0, 'origin=3'), (0.0, 45.0)]
```



Is age and blood pressure correlated? Maybe it is different for females and males?
Let's have a look with a scatter plot.

In [20]:
```
auto_data_df.plot.scatter('mpg', 'cylinders', c='origin', colormap='winter');
```

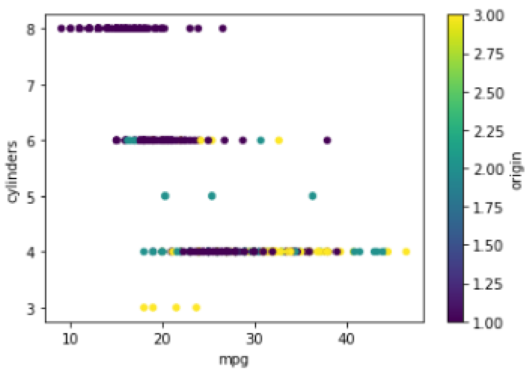


According to:
https://stackoverflow.com/questions/43578976/pandas-missing-x-tick-labels

the missing x-labels are a pandas bug.

Workaraound is to create axes prior to calling plot

In [21]:
```
fig, ax = plt.subplots()
auto_data_df.plot.scatter('mpg', 'cylinders', c='origin', colormap='viridis', ax=ax);
```



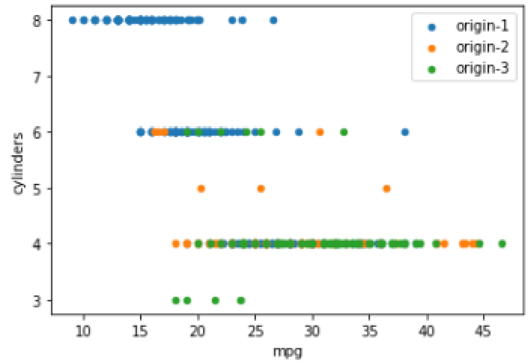
It is a bit annoying that there is a colorbar, we know gender is categorical.

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

In [28]:

```
colors = {1: 'tab:blue', 2: 'tab:orange', 3: 'tab:green'}
fig, ax = plt.subplots()
for key, group in auto_data_df.groupby(by='origin'):
    group.plot.scatter('mpg', 'cylinders', c=colors[group.iloc[1]["origin"]], label=("origin-" + s
```



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

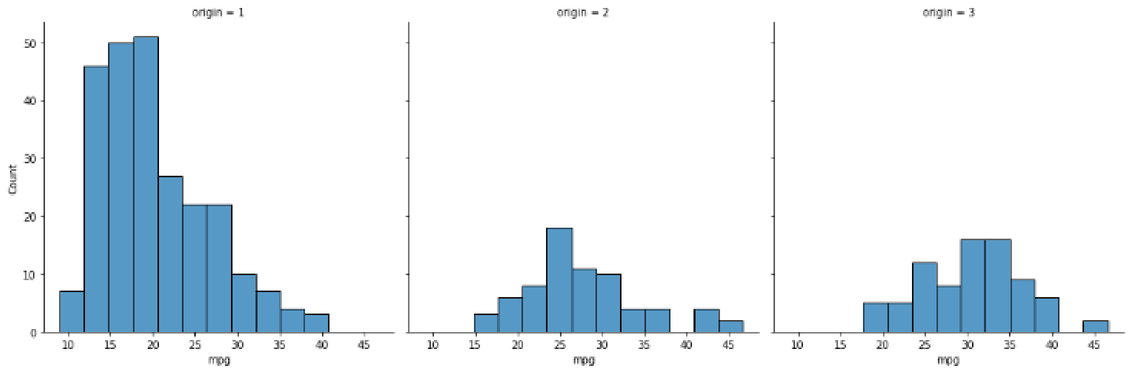
Seaborn is usually imported as `sns`

In [29]:

```
import seaborn as sns
```

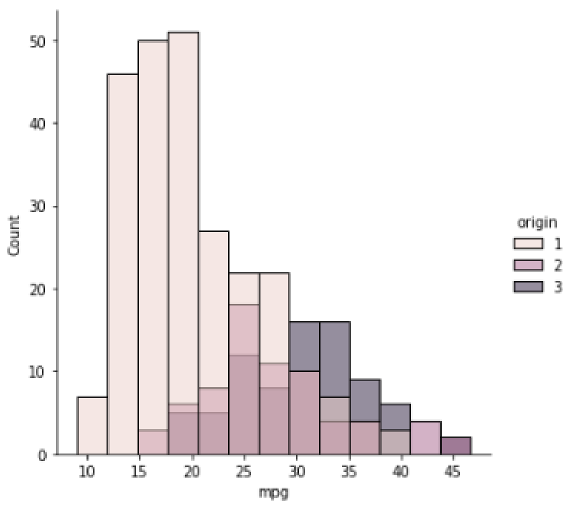
Let's re-create the histograms by gender with seaborn with the figure level `displot()` function.

In [30]:
```
# Use gender to split age into columns
sns.displot(x='mpg', col='origin', data=auto_data_df);
```



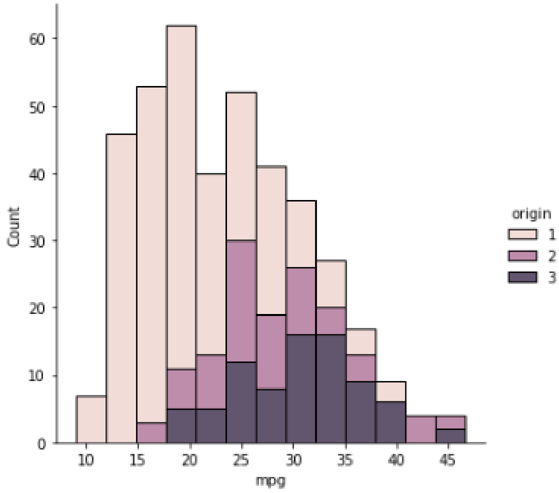
We can display the counts in the same plot, one on top of the other.

In [32]:
```
# Use gender to color (hue) in the same plot
sns.displot(x='mpg', hue='origin', data=auto_data_df);
```



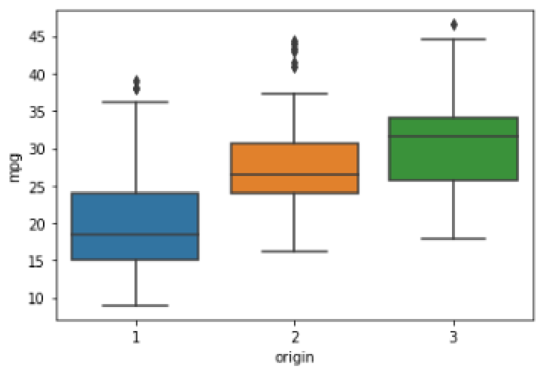
To have an idea of the split between male and female, we can stack the counts, adding up to total.

In [33]:
```
sns.displot(x='mpg', hue='origin', data=auto_data_df, multiple='stack');
```



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

In [34]:
```
sns.boxplot(x='origin', y='mpg', data=auto_data_df);
```

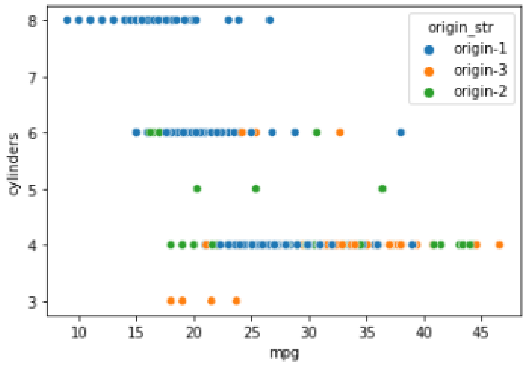


Let's re-create the scatter plot to see if age and blood pressure are correlated by gender.

To make the legend show strings we will create a gender string column with female and male strings rather than 0 and 1.

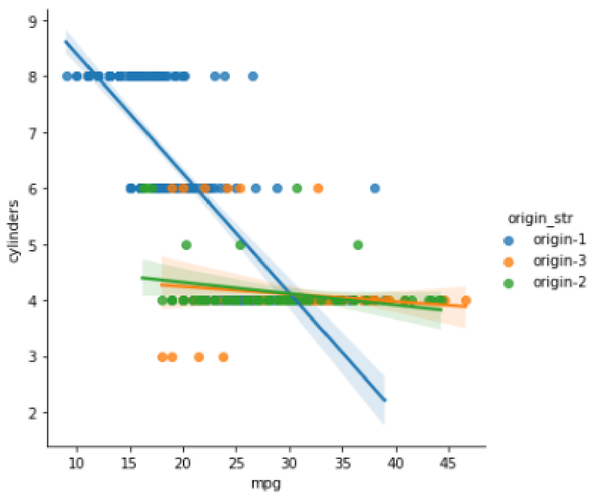
In [35]:
```
auto_data_df['origin_str'] = auto_data_df['origin'].replace({1: "origin-1", 2: "origin-2", 3: "ori
```

In [36]:
```
ax = sns.scatterplot(x='mpg', y='cylinders', data=auto_data_df, hue='origin_str')
```



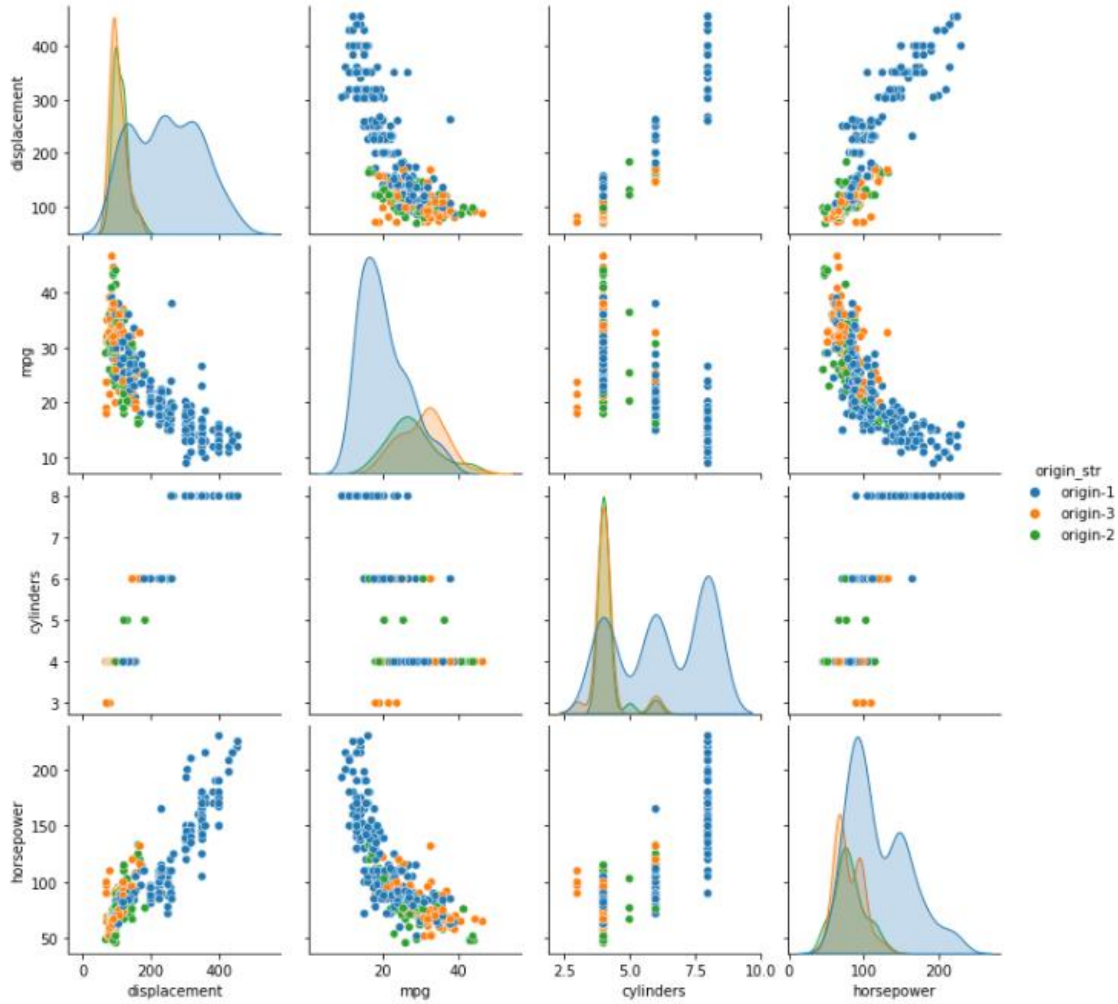
Adding a regression line helps with visualizing the relationship

In [37]:
```
ax = sns.lmplot(x='mpg', y='cylinders', data=auto_data_df, hue='origin_str')
```



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

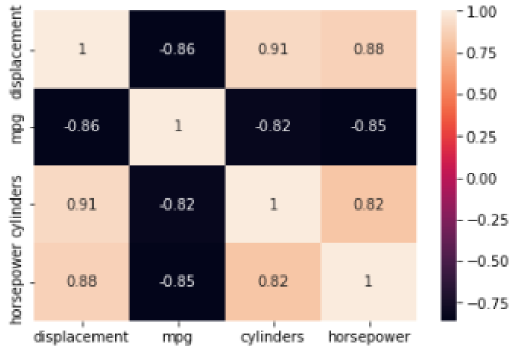
In [38]:
```
sns.pairplot(auto_data_df, vars=['displacement', 'mpg', 'cylinders', 'horsepower'], hue='origin_st
```



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

In [39]:

```python
g = sns.heatmap(auto_data_df[['displacement', 'mpg', 'cylinders', 'horsepower']].corr(method='spea
            annot=True)
```



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

In [ ]:

```python
```