iris

November 7, 2019

[43]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import statsmodels
```

[44]:

```
auto = sns.load_dataset('mpg')
```

[45]:

```
auto
# miles-per-gallon
```

[45]:

```
      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]
```

[46]:

```
type(auto)
```

[46]:

```
pandas.core.frame.DataFrame
```

[47]:

```
auto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
mpg             398 non-null float64
cylinders       398 non-null int64
displacement    398 non-null float64
horsepower      392 non-null float64
weight          398 non-null int64
acceleration    398 non-null float64
model_year      398 non-null int64
origin          398 non-null object
name            398 non-null object
dtypes: float64(4), int64(3), object(2)
memory usage: 28.1+ KB
```

[48]:

```
auto['origin'].unique()
```

[48]:

```
array(['usa', 'japan', 'europe'], dtype=object)
```

[49]:

```
auto.max()
```

[49]:

```
mpg                         46.6
cylinders                      8
displacement                 455
horsepower                   230
weight                      5140
acceleration                24.8
model_year                    82
origin                       usa
name            vw rabbit custom
dtype: object
```

[50]:

```
auto.min()
```

[50]:

```
mpg                                  9
cylinders                            3
displacement                        68
horsepower                          46
weight                            1613
acceleration                         8
model_year                          70
```

```
origin                           europe
name            amc ambassador brougham
dtype: object
```

[51]:

```
auto.mean()
```

[51]:

```
mpg               23.514573
cylinders          5.454774
displacement     193.425879
horsepower       104.469388
weight          2970.424623
acceleration      15.568090
model_year        76.010050
dtype: float64
```

[78]:

```
auto.std()
```

[78]:

```
mpg               7.815984
cylinders         1.701004
displacement    104.269838
horsepower       38.491160
weight          846.841774
acceleration      2.757689
model_year        3.697627
dtype: float64
```

[83]:

```
auto.columns
```

[83]:

```
Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model_year', 'origin', 'name'],
      dtype='object')
```

[85]:

```
list(auto.columns)
```

[85]:

```
['mpg',
 'cylinders',
 'displacement',
 'horsepower',
 'weight',
 'acceleration',
 'model_year',
 'origin',
 'name']
```

[86]:

```
for col in auto.columns:
    print(col)
```

```
mpg
cylinders
displacement
horsepower
weight
acceleration
model_year
origin
name
```

[88]:
```
for col in auto.columns:
    print(col)
    print(auto[col].std())
```

```
mpg
7.815984312565782
cylinders
1.7010042445332119
displacement
104.26983817119591
horsepower
38.49115993282849
weight
846.8417741973268
acceleration
2.757688929812676
model_year
3.697626646732623
origin
```

```
       ␣
 ↪---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call␣
 ↪last)

        ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
 ↪axis, skipna, **kwds)
        119                 else:
    --> 120                     result = alt(values, axis=axis, skipna=skipna,␣
 ↪**kwds)
        121             except Exception:


        ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
 ↪nanvar(values, axis, skipna, ddof, mask)
```

```
       767     # See https://en.wikipedia.org/wiki/
↪Algorithms_for_calculating_variance
   --> 768     avg = _ensure_numeric(values.sum(axis=axis, dtype=np.float64)) /␣
↪count
       769     if axis is not None:


       ~/.local/lib/python3.6/site-packages/numpy/core/_methods.py in _sum(a,␣
↪axis, dtype, out, keepdims, initial, where)
        37          initial=_NoValue, where=True):
   ---> 38     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
        39


       ValueError: could not convert string to float: 'usa'


   During handling of the above exception, another exception occurred:


       ValueError                                Traceback (most recent call␣
↪last)

       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
       122                 try:
   --> 123                     result = alt(values, axis=axis, skipna=skipna,␣
↪**kwds)
       124                 except ValueError as e:


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
↪nanvar(values, axis, skipna, ddof, mask)
       767     # See https://en.wikipedia.org/wiki/
↪Algorithms_for_calculating_variance
   --> 768     avg = _ensure_numeric(values.sum(axis=axis, dtype=np.float64)) /␣
↪count
       769     if axis is not None:


       ~/.local/lib/python3.6/site-packages/numpy/core/_methods.py in _sum(a,␣
↪axis, dtype, out, keepdims, initial, where)
        37          initial=_NoValue, where=True):
   ---> 38     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
        39
       ValueError: could not convert string to float: 'usa'


   During handling of the above exception, another exception occurred:


       TypeError                                 Traceback (most recent call␣
↪last)

       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
       119                 else:
   --> 120                     result = alt(values, axis=axis, skipna=skipna,␣
↪**kwds)
       121             except Exception:


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
↪nanstd(values, axis, skipna, ddof, mask)
       710     """
   --> 711     result = np.sqrt(nanvar(values, axis=axis, skipna=skipna,␣
↪ddof=ddof, mask=mask))
       712     return _wrap_results(result, values.dtype)


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in _f(*args,␣
↪**kwargs)
        69                 with np.errstate(invalid="ignore"):
   ---> 70                     return f(*args, **kwargs)
        71             except ValueError as e:


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
       130                     if is_object_dtype(values):
   --> 131                         raise TypeError(e)
       132                     raise


       TypeError: could not convert string to float: 'usa'


   During handling of the above exception, another exception occurred:


       ValueError                                Traceback (most recent call␣
↪last)
```

```
       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
       119                 else:
   --> 120                     result = alt(values, axis=axis, skipna=skipna,␣
↪**kwds)
       121             except Exception:


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
↪nanvar(values, axis, skipna, ddof, mask)
       767     # See https://en.wikipedia.org/wiki/
↪Algorithms_for_calculating_variance
   --> 768     avg = _ensure_numeric(values.sum(axis=axis, dtype=np.float64)) /␣
↪count
       769     if axis is not None:


       ~/.local/lib/python3.6/site-packages/numpy/core/_methods.py in _sum(a,␣
↪axis, dtype, out, keepdims, initial, where)
        37          initial=_NoValue, where=True):
   ---> 38     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
        39


       ValueError: could not convert string to float: 'usa'


   During handling of the above exception, another exception occurred:


       ValueError                                Traceback (most recent call␣
↪last)

       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
       122                 try:
   --> 123                     result = alt(values, axis=axis, skipna=skipna,␣
↪**kwds)
       124                 except ValueError as e:


       ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
↪nanvar(values, axis, skipna, ddof, mask)
       767     # See https://en.wikipedia.org/wiki/
↪Algorithms_for_calculating_variance
```

```
   --> 768     avg = _ensure_numeric(values.sum(axis=axis, dtype=np.float64)) /␣
↪count
       769     if axis is not None:


       ~/.local/lib/python3.6/site-packages/numpy/core/_methods.py in _sum(a,␣
↪axis, dtype, out, keepdims, initial, where)
        37          initial=_NoValue, where=True):
   ---> 38     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
        39


       ValueError: could not convert string to float: 'usa'


   During handling of the above exception, another exception occurred:


       TypeError                                 Traceback (most recent call␣
↪last)

       <ipython-input-88-85729dd86791> in <module>
         1 for col in auto.columns:
         2     print(col)
   ----> 3     print(auto[col].std())


       ~/.local/lib/python3.6/site-packages/pandas/core/generic.py in␣
↪stat_func(self, axis, skipna, level, ddof, numeric_only, **kwargs)
     11638             )
     11639         return self._reduce(
   > 11640             f, name, axis=axis, numeric_only=numeric_only,␣
↪skipna=skipna, ddof=ddof
     11641         )
     11642


       ~/.local/lib/python3.6/site-packages/pandas/core/series.py in␣
↪_reduce(self, op, name, axis, skipna, numeric_only, filter_type, **kwds)
      4088                 )
      4089             with np.errstate(all="ignore"):
   -> 4090                 return op(delegate, skipna=skipna, **kwds)
      4091
      4092         # TODO(EA) dispatch to Index
```

```
    ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in _f(*args,␣
↪**kwargs)
         68             try:
         69                 with np.errstate(invalid="ignore"):
    ---> 70                     return f(*args, **kwargs)
         71             except ValueError as e:
         72                 # we want to transform an object array


    ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
        121             except Exception:
        122                 try:
    --> 123                     result = alt(values, axis=axis, skipna=skipna,␣
↪**kwds)
        124                 except ValueError as e:
        125                     # we want to transform an object array


    ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in␣
↪nanstd(values, axis, skipna, ddof, mask)
        709     1.0
        710     """
    --> 711     result = np.sqrt(nanvar(values, axis=axis, skipna=skipna,␣
↪ddof=ddof, mask=mask))
        712     return _wrap_results(result, values.dtype)
        713


    ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in _f(*args,␣
↪**kwargs)
         68             try:
         69                 with np.errstate(invalid="ignore"):
    ---> 70                     return f(*args, **kwargs)
         71             except ValueError as e:
         72                 # we want to transform an object array


    ~/.local/lib/python3.6/site-packages/pandas/core/nanops.py in f(values,␣
↪axis, skipna, **kwds)
        129
        130                     if is_object_dtype(values):
    --> 131                         raise TypeError(e)
        132                     raise
        133
        TypeError: could not convert string to float: 'usa'
```

[92]:

```
for col in auto.columns:
    print(col)
    print(auto[col].dtypes)
    if (auto[col].dtypes == 'int64' or auto[col].dtypes == 'float64'):
        print(auto[col].std())
    print()
```

```
mpg
float64
7.815984312565782

cylinders
int64
1.7010042445332119

displacement
float64
104.26983817119591

horsepower
float64
38.49115993282849

weight
int64
846.8417741973268

acceleration
float64
2.757688929812676

model_year
int64
3.697626646732623

origin
object

name
object
```

[53]:

```
auto['horsepower'][0]
```

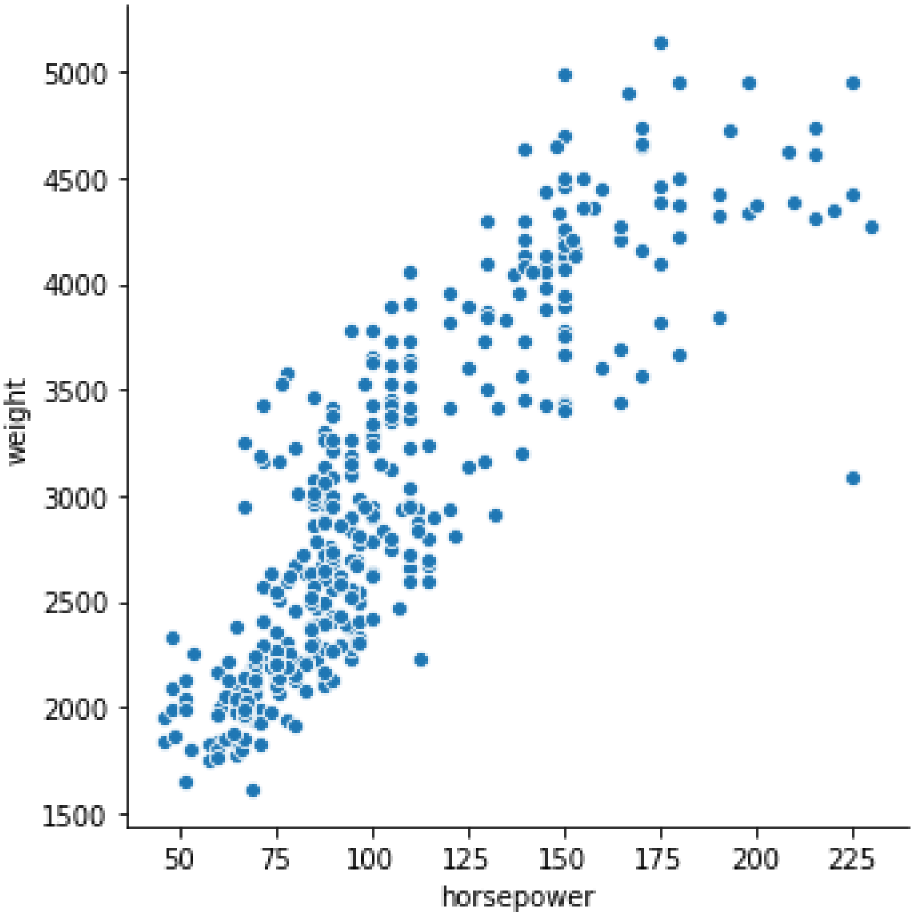
```
[53]: 130.0
```

```
[77]: auto.loc[0]
```

```
[77]: mpg                                 18
      cylinders                            8
      displacement                       307
      horsepower                         130
      weight                            3504
      acceleration                        12
      model_year                          70
      origin                             usa
      name            chevrolet chevelle malibu
      Name: 0, dtype: object
```

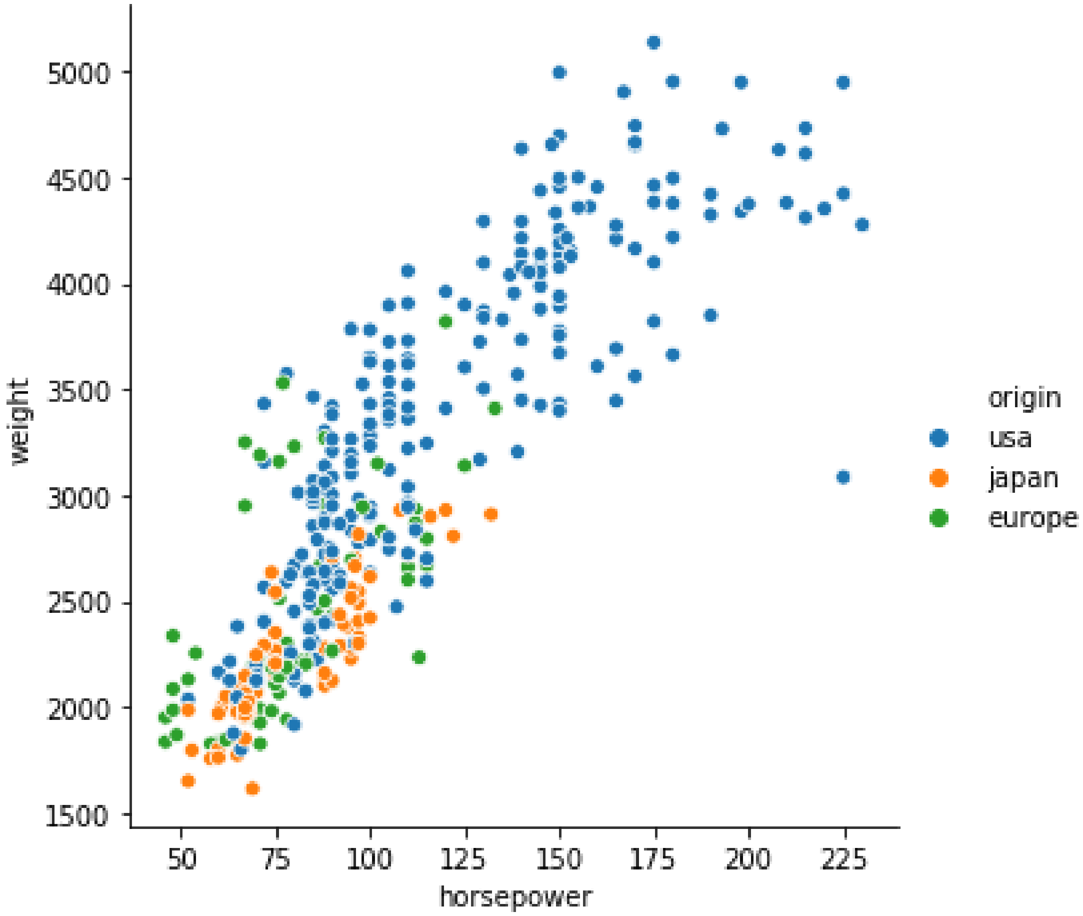
```
[55]: sns.relplot(x = 'horsepower',
                  y = 'weight',
                  data = auto)
```

```
[55]: <seaborn.axisgrid.FacetGrid at 0x7fd60b2df240>
```



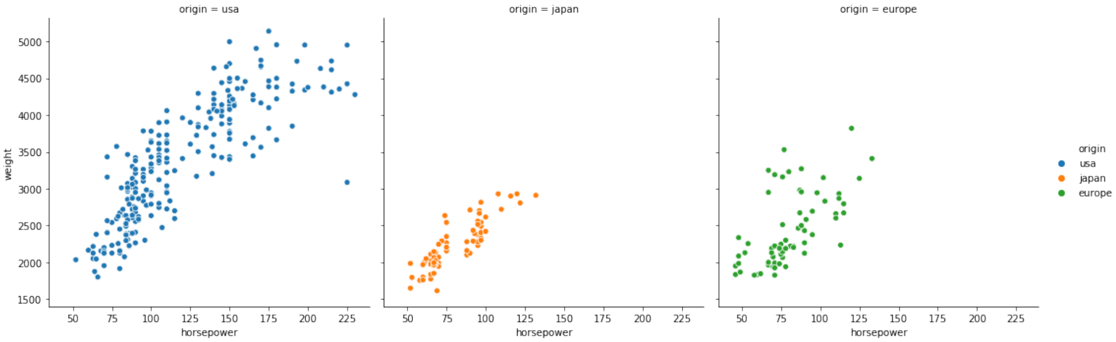
[56]:
```
sns.relplot(x = 'horsepower',
            y = 'weight',
            hue = 'origin',
            data = auto)
```

[56]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd60bcf1048>
```



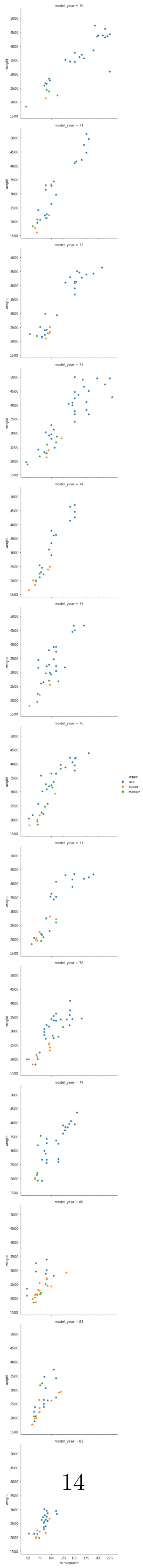
[57]:
```
sns.relplot(x = 'horsepower',
            y = 'weight',
            hue = 'origin',
            col = 'origin',
            data = auto)
```

[57]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd60c2550f0>
```



```
[58]: sns.relplot(x = 'horsepower',
                  y = 'weight',
                  hue = 'origin',
                  row = 'model_year',
                  data = auto)
```

```
[58]: <seaborn.axisgrid.FacetGrid at 0x7fd60be0a588>
```

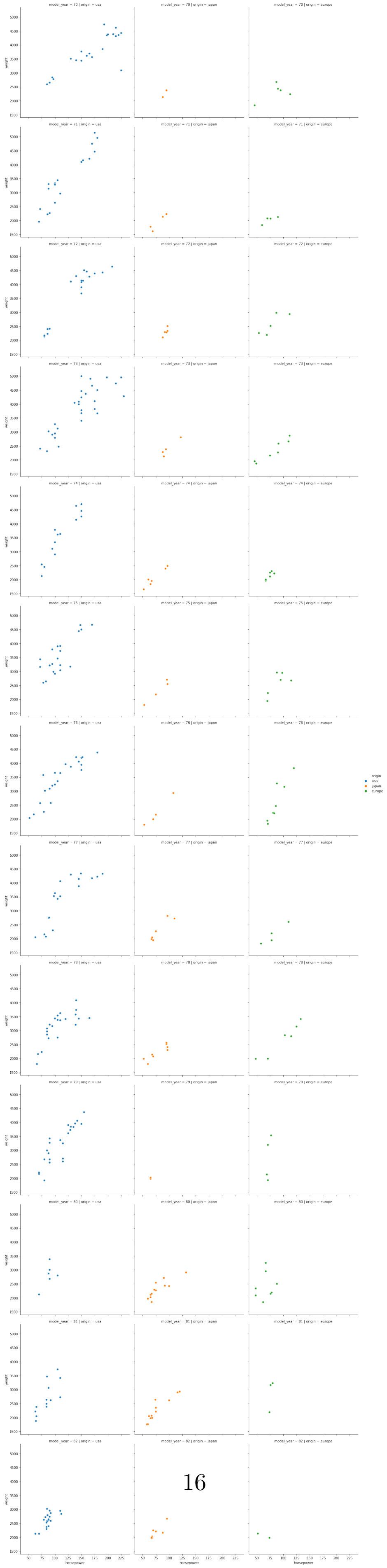

model_year = 70
model_year = 71
model_year = 72
model_year = 73
model_year = 74
model_year = 75
model_year = 76
model_year = 77
model_year = 78
model_year = 79
model_year = 80
model_year = 81
model_year = 82
weight
5000
4500
4000
3500
3000
2500
2000
1500
horsepower
50
75
100
125
150
175
200
225
origin
usa
japan
europe

[60]:

```
sns.relplot(x = 'horsepower',
            y = 'weight',
            hue = 'origin',
            row = 'model_year',
            col = 'origin',
            data = auto)
```

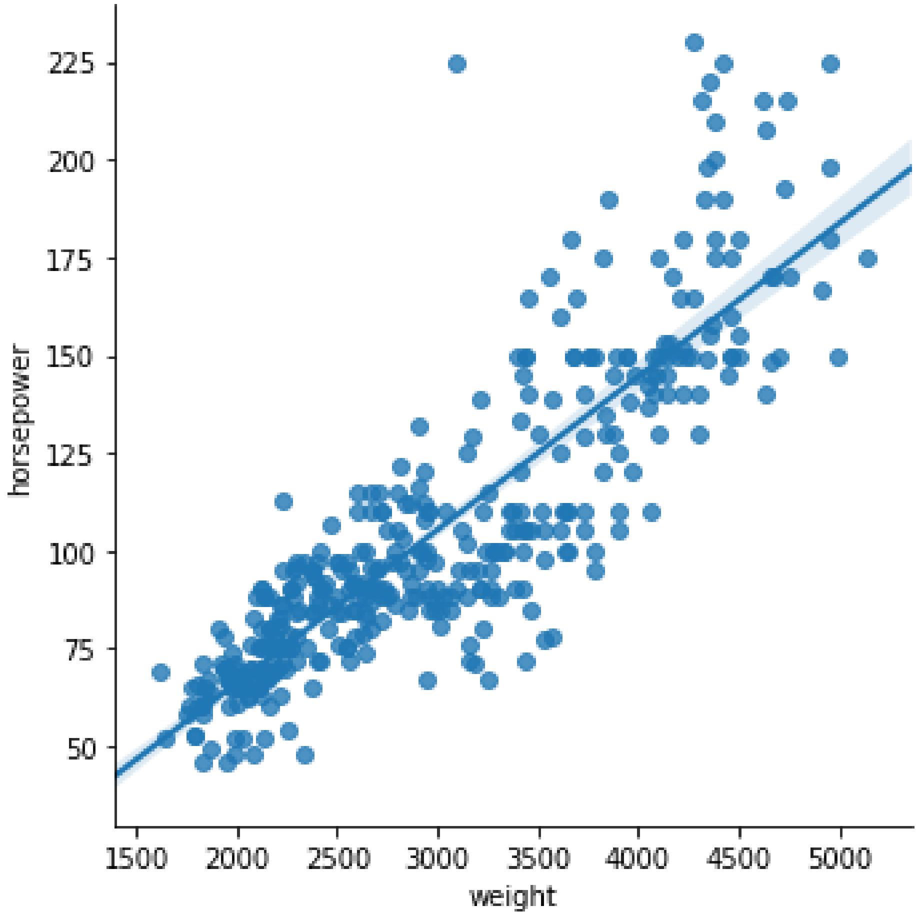
[60]:

```
<seaborn.axisgrid.FacetGrid at 0x7fd60b4a7630>
```


model_year = 70 | origin = usa
model_year = 70 | origin = japan
model_year = 70 | origin = europe
model_year = 71 | origin = usa
model_year = 71 | origin = japan
model_year = 71 | origin = europe
model_year = 72 | origin = usa
model_year = 72 | origin = japan
model_year = 72 | origin = europe
model_year = 73 | origin = usa
model_year = 73 | origin = japan
model_year = 73 | origin = europe
model_year = 74 | origin = usa
model_year = 74 | origin = japan
model_year = 74 | origin = europe
model_year = 75 | origin = usa
model_year = 75 | origin = japan
model_year = 75 | origin = europe
model_year = 76 | origin = usa
model_year = 76 | origin = japan
model_year = 76 | origin = europe
model_year = 77 | origin = usa
model_year = 77 | origin = japan
model_year = 77 | origin = europe
model_year = 78 | origin = usa
model_year = 78 | origin = japan
model_year = 78 | origin = europe
model_year = 79 | origin = usa
model_year = 79 | origin = japan
model_year = 79 | origin = europe
model_year = 80 | origin = usa
model_year = 80 | origin = japan
model_year = 80 | origin = europe
model_year = 81 | origin = usa
model_year = 81 | origin = japan
model_year = 81 | origin = europe
model_year = 82 | origin = usa
model_year = 82 | origin = japan
model_year = 82 | origin = europe
weight
5000
4500
4000
3500
3000
2500
2000
1500
50
75
100
125
150
175
200
225
horsepower
origin
usa
japan
europe

```
[61]: sns.lmplot(x='weight', y='horsepower', data=auto)
```

```
[61]: <seaborn.axisgrid.FacetGrid at 0x7fd60ab1f748>
```



```
[62]: help(sns.lmplot)
```

```
Help on function lmplot in module seaborn.regression:

lmplot(x, y, data, hue=None, col=None, row=None, palette=None, col_wrap=None,
height=5, aspect=1, markers='o', sharex=True, sharey=True, hue_order=None,
col_order=None, row_order=None, legend=True, legend_out=True, x_estimator=None,
x_bins=None, x_ci='ci', scatter=True, fit_reg=True, ci=95, n_boot=1000,
units=None, order=1, logistic=False, lowess=False, robust=False, logx=False,
x_partial=None, y_partial=None, truncate=False, x_jitter=None, y_jitter=None,
scatter_kws=None, line_kws=None, size=None)
    Plot data and regression model fits across a FacetGrid.
```

```
This function combines :func:`regplot` and :class:`FacetGrid`. It is
intended as a convenient interface to fit regression models across
conditional subsets of a dataset.

When thinking about how to assign variables to different facets, a general
rule is that it makes sense to use ``hue`` for the most important
comparison, followed by ``col`` and ``row``. However, always think about
your particular dataset and the goals of the visualization you are
creating.

There are a number of mutually exclusive options for estimating the
regression model. See the :ref:`tutorial <regression_tutorial>` for more
information.

The parameters to this function span most of the options in
:class:`FacetGrid`, although there may be occasional cases where you will
want to use that class and :func:`regplot` directly.

Parameters
----------
x, y : strings, optional
    Input variables; these should be column names in ``data``.
data : DataFrame
    Tidy ("long-form") dataframe where each column is a variable and each
    row is an observation.
hue, col, row : strings
    Variables that define subsets of the data, which will be drawn on
    separate facets in the grid. See the ``*_order`` parameters to control
    the order of levels of this variable.
palette : palette name, list, or dict, optional
    Colors to use for the different levels of the ``hue`` variable. Should
    be something that can be interpreted by :func:`color_palette`, or a
    dictionary mapping hue levels to matplotlib colors.
col_wrap : int, optional
    "Wrap" the column variable at this width, so that the column facets
    span multiple rows. Incompatible with a ``row`` facet.
height : scalar, optional
    Height (in inches) of each facet. See also: ``aspect``.
aspect : scalar, optional
    Aspect ratio of each facet, so that ``aspect * height`` gives the width
    of each facet in inches.
markers : matplotlib marker code or list of marker codes, optional
    Markers for the scatterplot. If a list, each marker in the list will be
    used for each level of the ``hue`` variable.
share{x,y} : bool, 'col', or 'row' optional
    If true, the facets will share y axes across columns and/or x axes
    across rows.
```

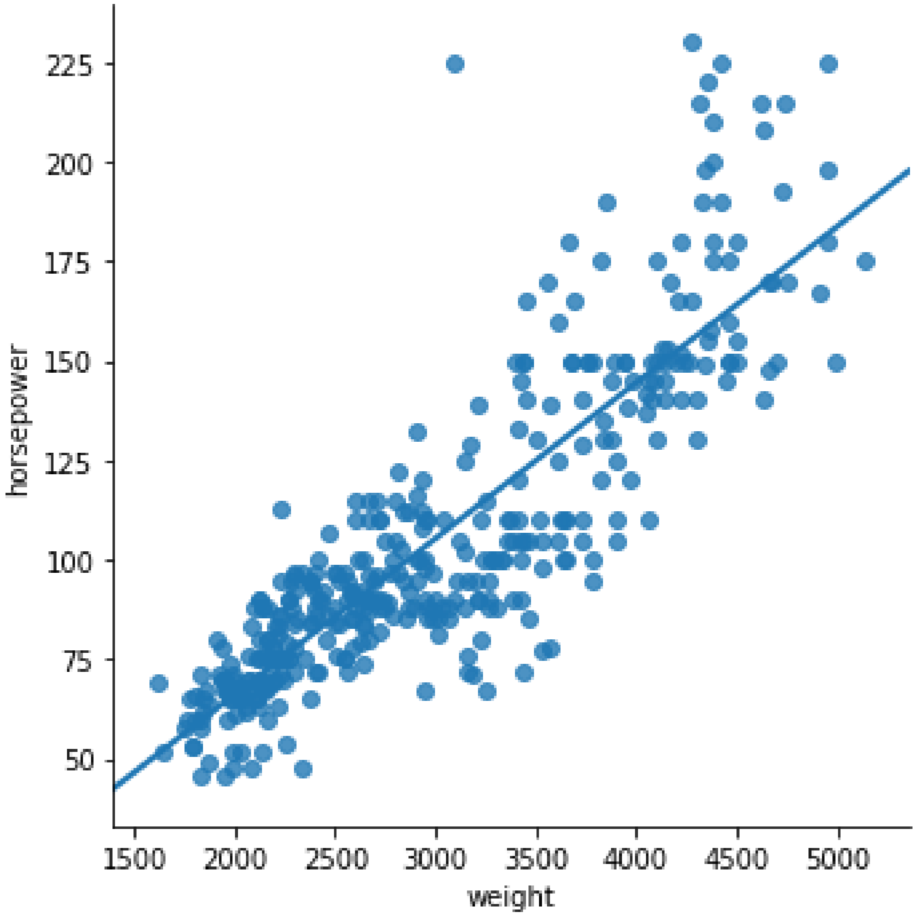
```
{hue,col,row}_order : lists, optional
    Order for the levels of the faceting variables. By default, this will
    be the order that the levels appear in ``data`` or, if the variables
    are pandas categoricals, the category order.
legend : bool, optional
    If ``True`` and there is a ``hue`` variable, add a legend.
legend_out : bool, optional
    If ``True``, the figure size will be extended, and the legend will be
    drawn outside the plot on the center right.
x_estimator : callable that maps vector -> scalar, optional
    Apply this function to each unique value of ``x`` and plot the
    resulting estimate. This is useful when ``x`` is a discrete variable.
    If ``x_ci`` is given, this estimate will be bootstrapped and a
    confidence interval will be drawn.
x_bins : int or vector, optional
    Bin the ``x`` variable into discrete bins and then estimate the central
    tendency and a confidence interval. This binning only influences how
    the scatterplot is drawn; the regression is still fit to the original
    data.  This parameter is interpreted either as the number of
    evenly-sized (not necessary spaced) bins or the positions of the bin
    centers. When this parameter is used, it implies that the default of
    ``x_estimator`` is ``numpy.mean``.
x_ci : "ci", "sd", int in [0, 100] or None, optional
    Size of the confidence interval used when plotting a central tendency
    for discrete values of ``x``. If ``"ci"``, defer to the value of the
    ``ci`` parameter. If ``"sd"``, skip bootstrapping and show the
    standard deviation of the observations in each bin.
scatter : bool, optional
    If ``True``, draw a scatterplot with the underlying observations (or
    the ``x_estimator`` values).
fit_reg : bool, optional
    If ``True``, estimate and plot a regression model relating the ``x``
    and ``y`` variables.
ci : int in [0, 100] or None, optional
    Size of the confidence interval for the regression estimate. This will
    be drawn using translucent bands around the regression line. The
    confidence interval is estimated using a bootstrap; for large
    datasets, it may be advisable to avoid that computation by setting
    this parameter to None.
n_boot : int, optional
    Number of bootstrap resamples used to estimate the ``ci``. The default
    value attempts to balance time and stability; you may want to increase
    this value for "final" versions of plots.
units : variable name in ``data``, optional
    If the ``x`` and ``y`` observations are nested within sampling units,
    those can be specified here. This will be taken into account when
    computing the confidence intervals by performing a multilevel bootstrap
    that resamples both units and observations (within unit). This does not
    otherwise influence how the regression is estimated or drawn.
order : int, optional
    If ``order`` is greater than 1, use ``numpy.polyfit`` to estimate a
    polynomial regression.
logistic : bool, optional
    If ``True``, assume that ``y`` is a binary variable and use
    ``statsmodels`` to estimate a logistic regression model. Note that this
    is substantially more computationally intensive than linear regression,
    so you may wish to decrease the number of bootstrap resamples
    (``n_boot``) or set ``ci`` to None.
lowess : bool, optional
    If ``True``, use ``statsmodels`` to estimate a nonparametric lowess
    model (locally weighted linear regression). Note that confidence
    intervals cannot currently be drawn for this kind of model.
robust : bool, optional
    If ``True``, use ``statsmodels`` to estimate a robust regression. This
    will de-weight outliers. Note that this is substantially more
    computationally intensive than standard linear regression, so you may
    wish to decrease the number of bootstrap resamples (``n_boot``) or set
    ``ci`` to None.
logx : bool, optional
    If ``True``, estimate a linear regression of the form y ~ log(x), but
    plot the scatterplot and regression model in the input space. Note that
    ``x`` must be positive for this to work.
{x,y}_partial : strings in ``data`` or matrices
    Confounding variables to regress out of the ``x`` or ``y`` variables
    before plotting.
truncate : bool, optional
    By default, the regression line is drawn to fill the x axis limits
    after the scatterplot is drawn. If ``truncate`` is ``True``, it will
    instead by bounded by the data limits.
{x,y}_jitter : floats, optional
    Add uniform random noise of this size to either the ``x`` or ``y``
    variables. The noise is added to a copy of the data after fitting the
    regression, and only influences the look of the scatterplot. This can
    be helpful when plotting variables that take discrete values.
{scatter,line}_kws : dictionaries
    Additional keyword arguments to pass to ``plt.scatter`` and
    ``plt.plot``.

See Also
--------
regplot : Plot data and a conditional model fit.
FacetGrid : Subplot grid for plotting conditional relationships.
pairplot : Combine :func:`regplot` and :class:`PairGrid` (when used with
           ``kind="reg"``).

Notes
```

```
-----

The :func:`regplot` and :func:`lmplot` functions are closely related, but
the former is an axes-level function while the latter is a figure-level
function that combines :func:`regplot` and :class:`FacetGrid`.

Examples
--------

These examples focus on basic regression model plots to exhibit the
various faceting options; see the :func:`regplot` docs for demonstrations
of the other options for plotting the data and models. There are also
other examples for how to manipulate plot using the returned object on
the :class:`FacetGrid` docs.

Plot a simple linear relationship between two variables:

.. plot::
    :context: close-figs

    >>> import seaborn as sns; sns.set(color_codes=True)
    >>> tips = sns.load_dataset("tips")
    >>> g = sns.lmplot(x="total_bill", y="tip", data=tips)

Condition on a third variable and plot the levels in different colors:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips)

Use different markers as well as colors so the plot will reproduce to
black-and-white more easily:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
    …                markers=["o", "x"])

Use a different color palette:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
    …                palette="Set1")
```

```
Map ``hue`` levels to colors with a dictionary:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
    …                палette=dict(Yes="g", No="m"))

Plot the levels of the third variable across different columns:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", col="smoker", data=tips)

Change the height and aspect ratio of the facets:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="size", y="total_bill", hue="day", col="day",
    …                data=tips, height=6, aspect=.4, x_jitter=.1)

Wrap the levels of the column variable into multiple rows:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", col="day", hue="day",
    …                data=tips, col_wrap=2, height=3)

Condition on two variables to make a full grid:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", row="sex", col="time",
    …                data=tips, height=3)

Use methods on the returned :class:`FacetGrid` instance to further tweak
the plot:

.. plot::
    :context: close-figs

    >>> g = sns.lmplot(x="total_bill", y="tip", row="sex", col="time",
    …                data=tips, height=3)
    >>> g = (g.set_axis_labels("Total bill (US Dollars)", "Tip")
    ...        .set(xlim=(0, 60), ylim=(0, 12),
    ...             xticks=[10, 30, 50], yticks=[2, 6, 10])
    ...        .fig.subplots_adjust(wspace=.02))
```

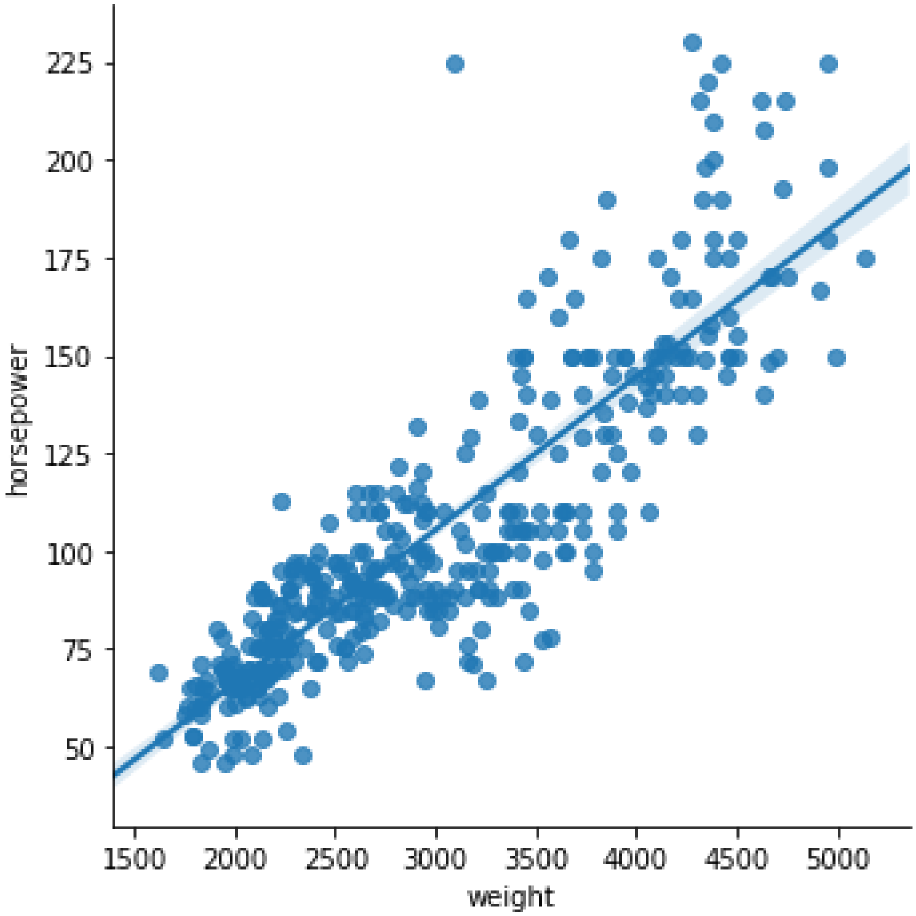
[63]:
```
sns.lmplot(x = 'weight',
           y = 'horsepower',
           ci = None,
           data = auto)
```

[63]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd60aa6cf28>
```



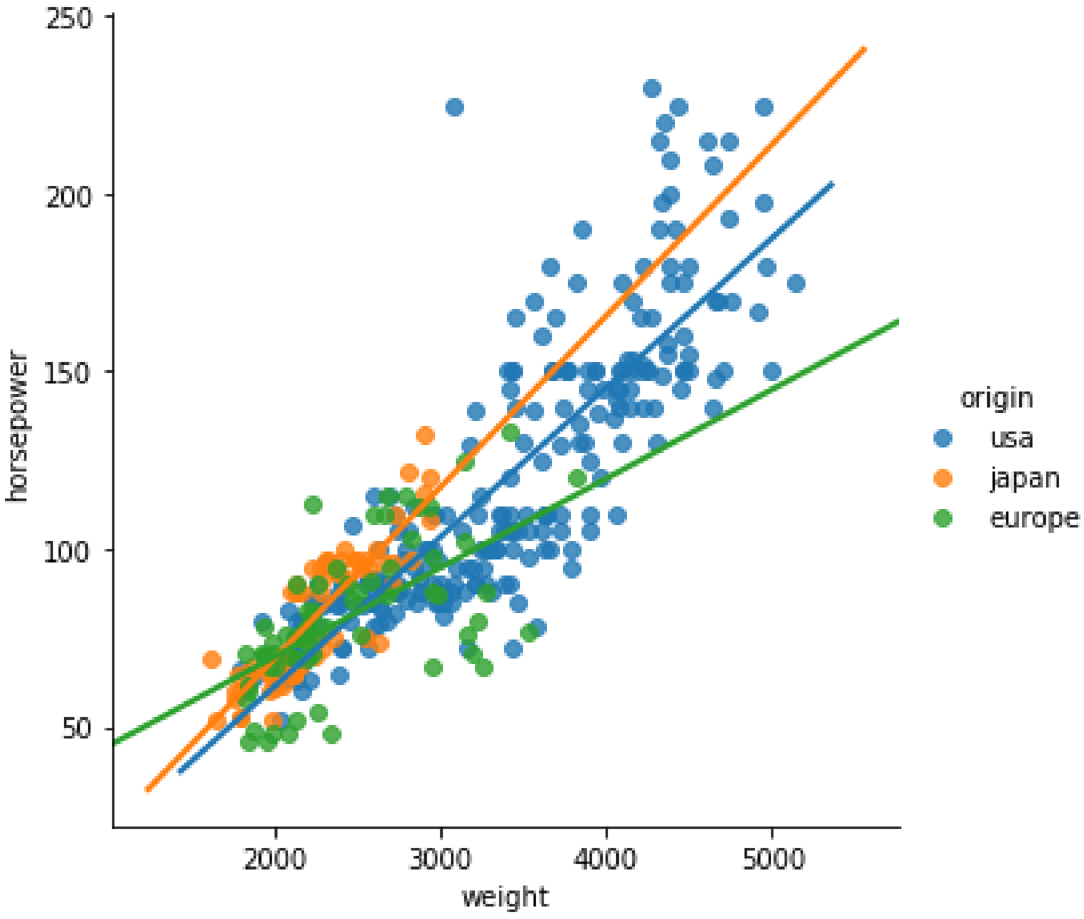
[64]:
```
sns.lmplot(x = 'weight',
           y = 'horsepower',
           ci = 95,
           data = auto)
```

[64]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd60aa33cc0>
```



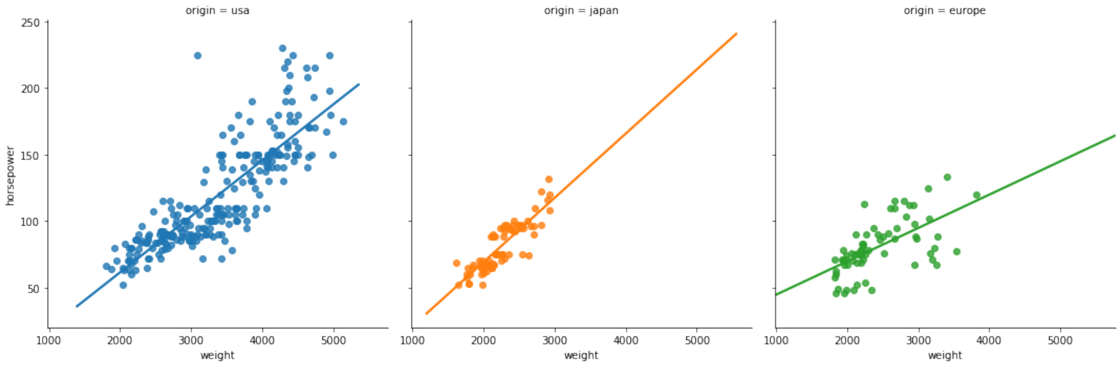
[65]:
```
sns.lmplot(x = 'weight',
           y = 'horsepower',
           hue = 'origin',
           ci = None,
           data = auto)
```

[65]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd60a9f1a90>
```



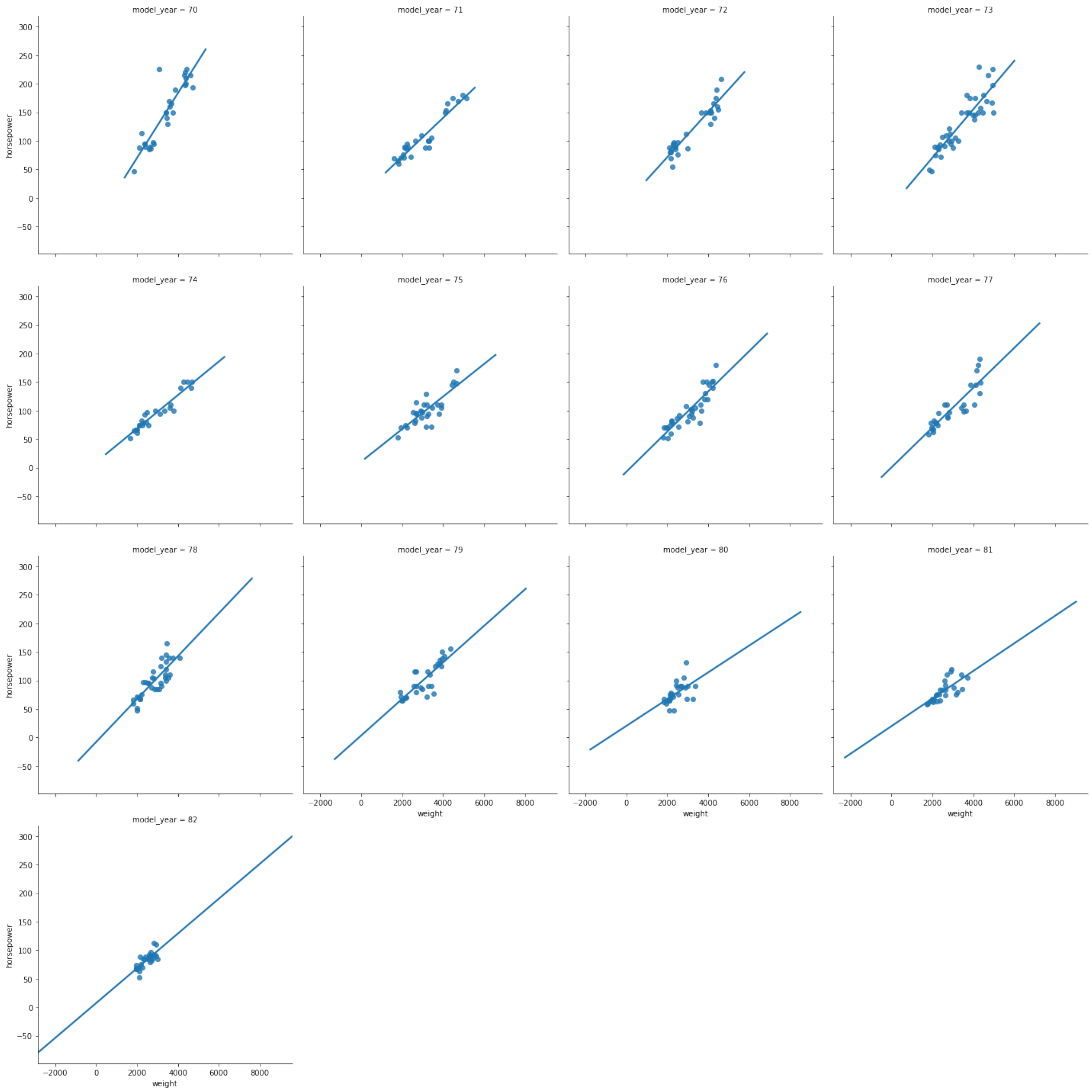
```
[66]: sns.lmplot(x = 'weight',
                 y = 'horsepower',
                 hue = 'origin',
                 ci = None,
                 col = 'origin',
                 data = auto)
```

```
[66]: <seaborn.axisgrid.FacetGrid at 0x7fd60a91c080>
```



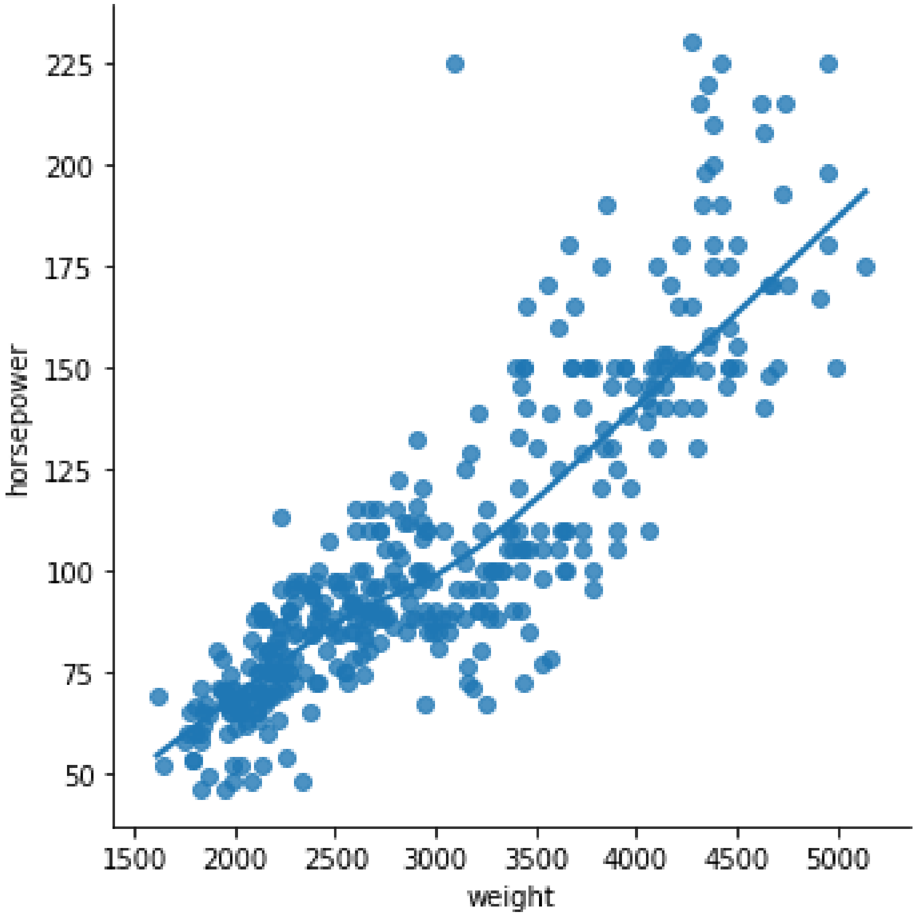
```
[67]: sns.lmplot(x = 'weight',
                 y = 'horsepower',
                 ci = None,
                 col = 'model_year',
                 col_wrap = 4,
                 data = auto)
```

```
[67]: <seaborn.axisgrid.FacetGrid at 0x7fd60aa06550>
```



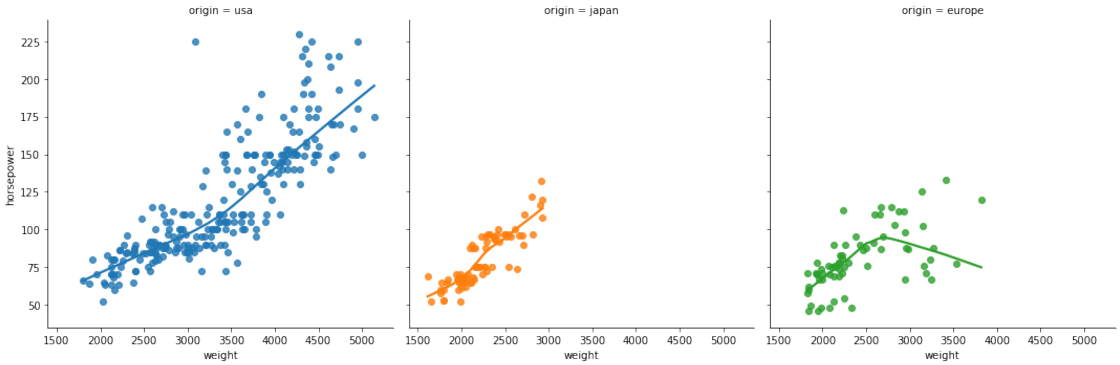
[68]:
```
sns.lmplot(x = 'weight',
           y = 'horsepower',
           ci = None,
           lowess = True,
           data = auto)
```

[68]:
```
<seaborn.axisgrid.FacetGrid at 0x7fd609c07e48>
```



```
[69]: sns.lmplot(x = 'weight',
                 y = 'horsepower',
                 hue = 'origin',
                 col = 'origin',
                 ci = None,
                 lowess = True,
                 data = auto)
```

```
[69]: <seaborn.axisgrid.FacetGrid at 0x7fd609cb2208>
```

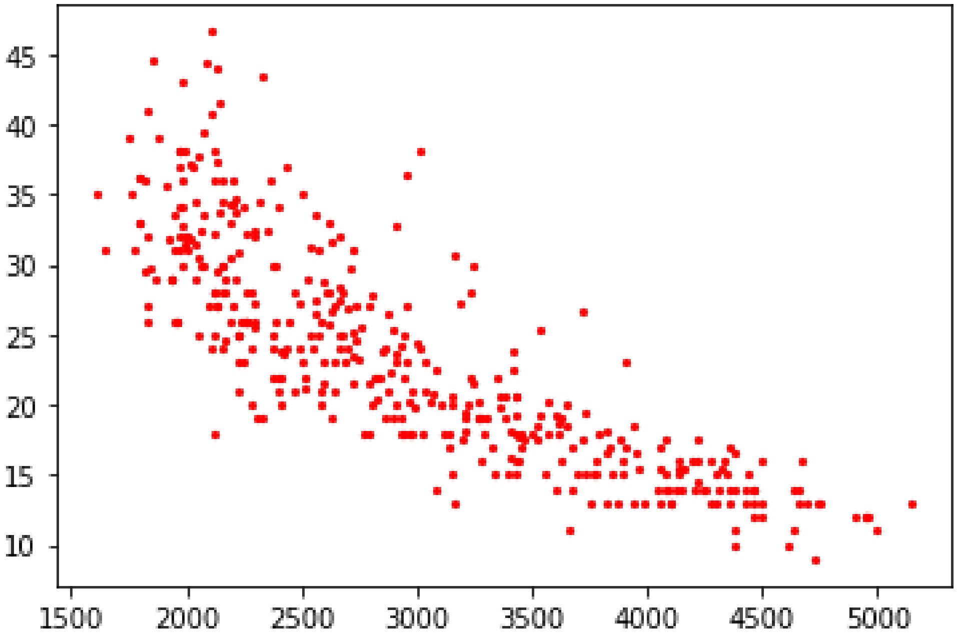


[70]:

```
plt.scatter(auto['weight'], auto['mpg'], label='data', color='red', marker='o',␣
 ↪s = 5)
```

[70]:

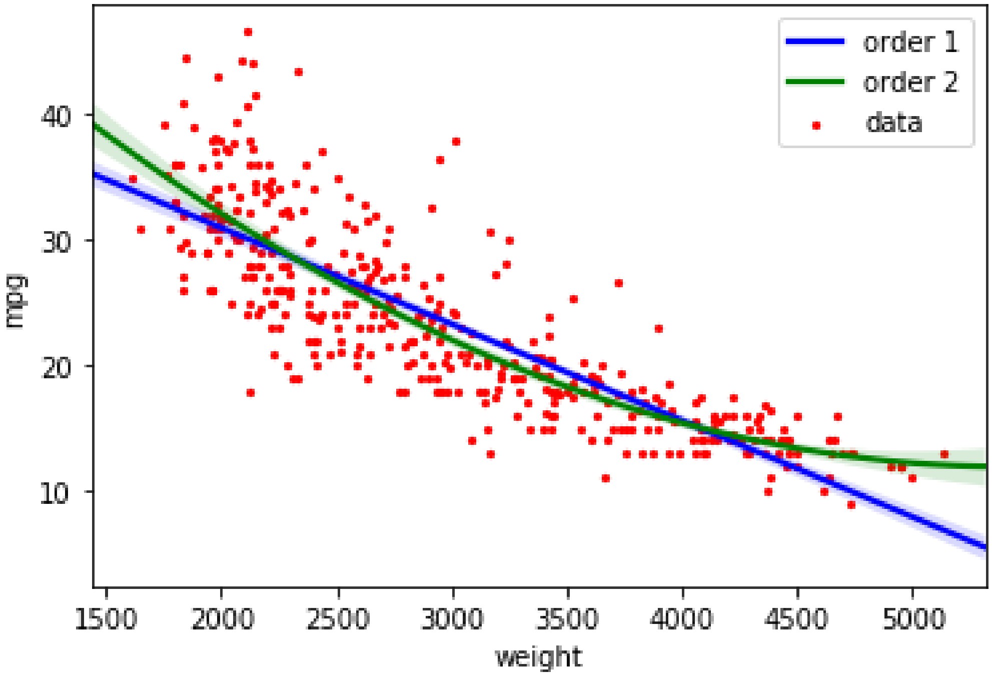
```
<matplotlib.collections.PathCollection at 0x7fd609a5e1d0>
```



[71]:

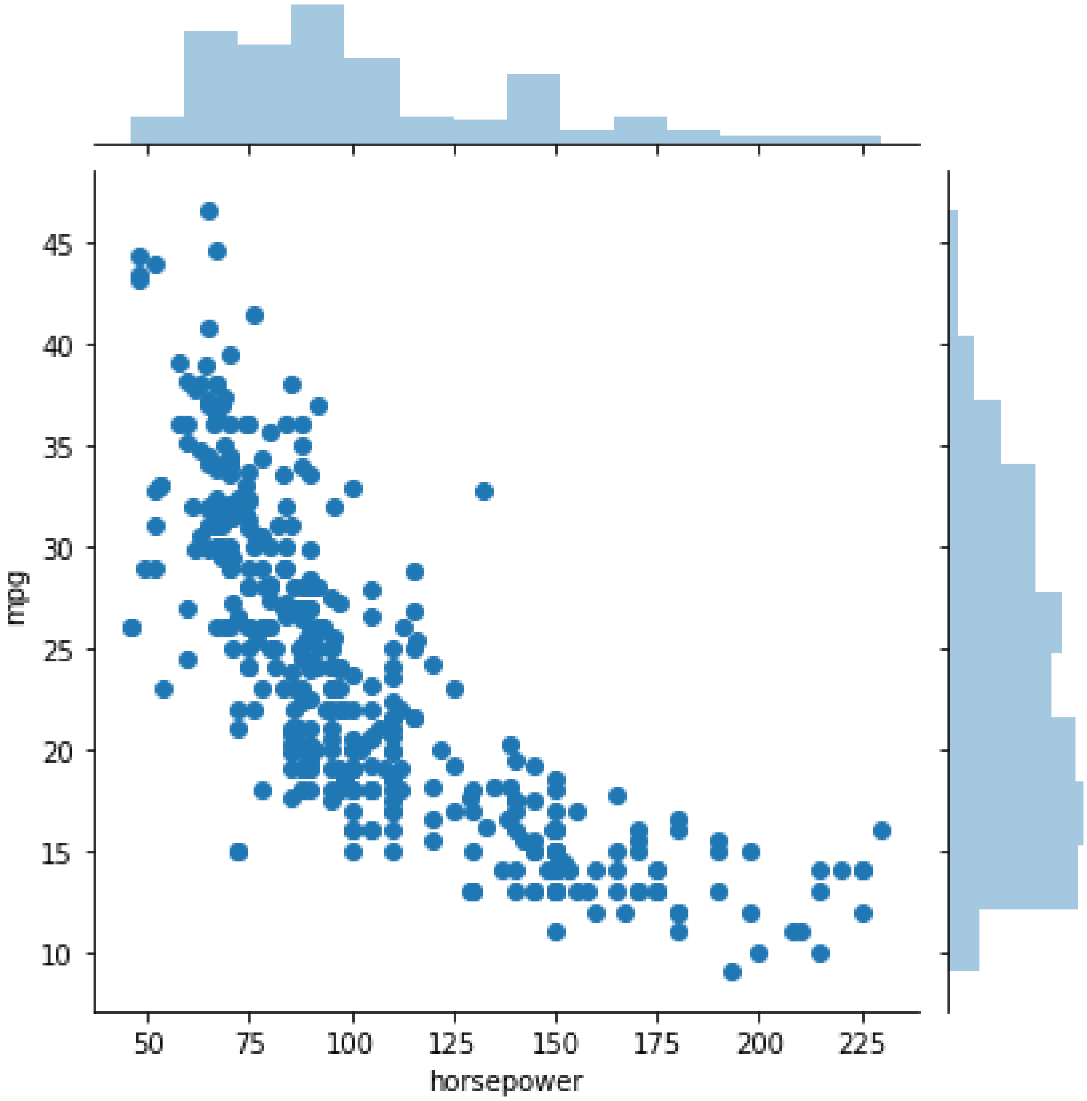
```
plt.scatter(auto['weight'], auto['mpg'], label='data', color='red', marker='o',␣
 ↪s = 5)
sns.regplot(x='weight', y='mpg', data=auto, color='blue', scatter=None,␣
 ↪label='order 1')
sns.regplot(x='weight', y='mpg', data=auto, color='green', order=2,␣
 ↪label='order 2',scatter=None)
plt.legend(loc='upper right')
```

```
[71]: <matplotlib.legend.Legend at 0x7fd609a76588>
```



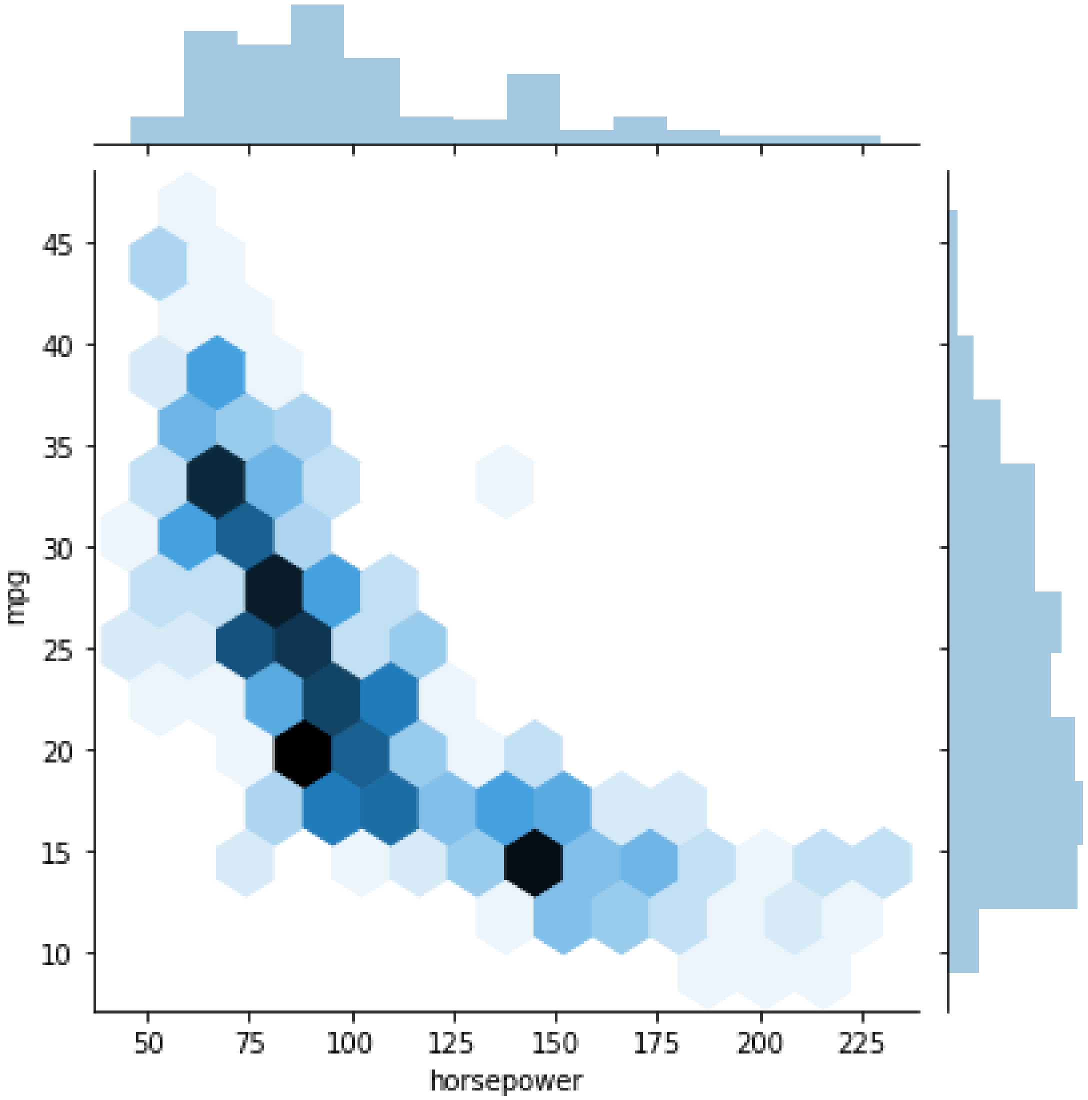
```
[93]: sns.jointplot(x='horsepower',y='mpg',data=auto)
```

```
[93]: <seaborn.axisgrid.JointGrid at 0x7fd60b170ac8>
```



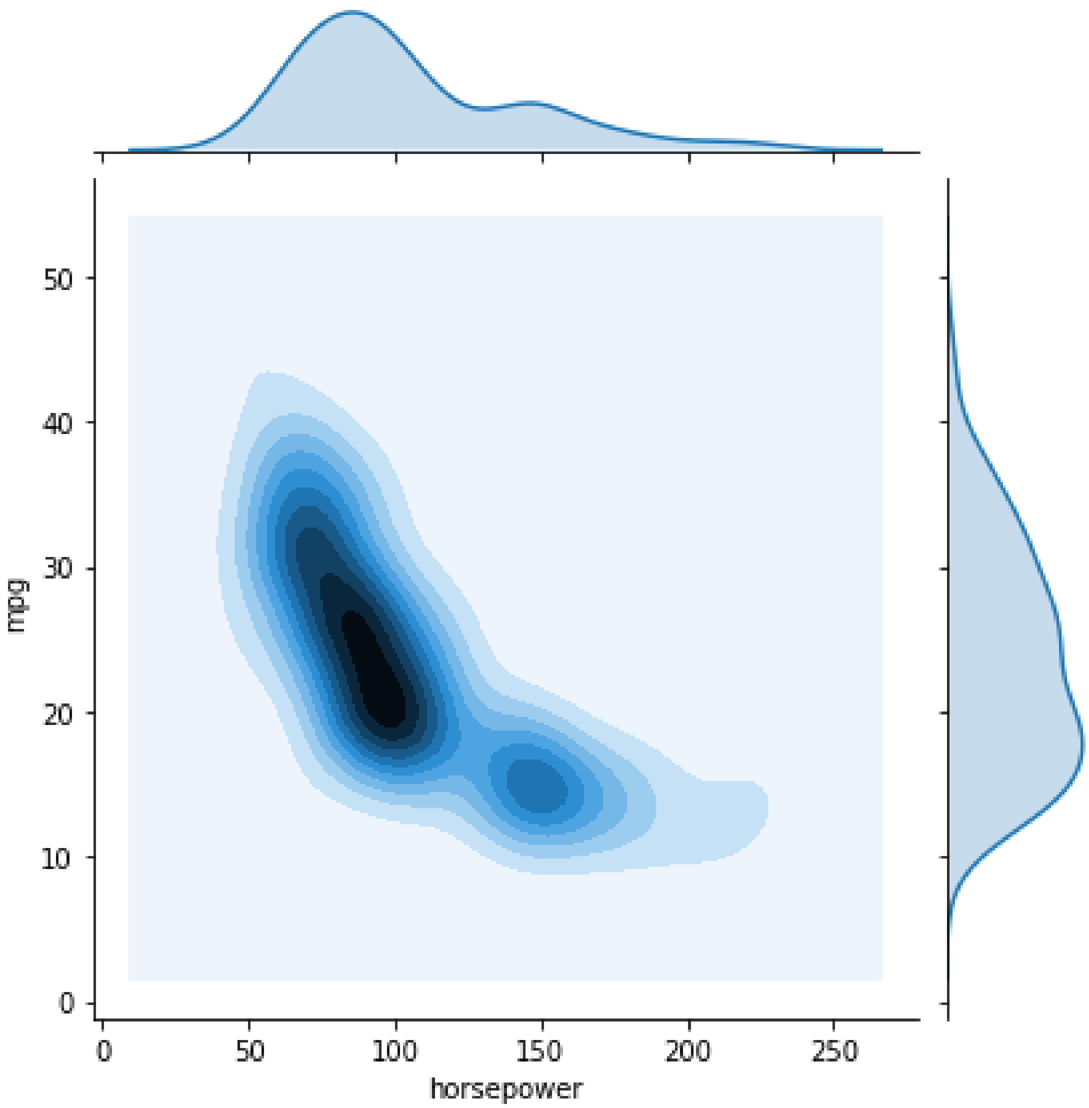
```
[95]: sns.jointplot(x='horsepower',y='mpg',kind = 'hex', data=auto)
```

```
[95]: <seaborn.axisgrid.JointGrid at 0x7fd60b45c0b8>
```



```
[96]: sns.jointplot(x='horsepower',y='mpg',kind = 'kde', data=auto)
```

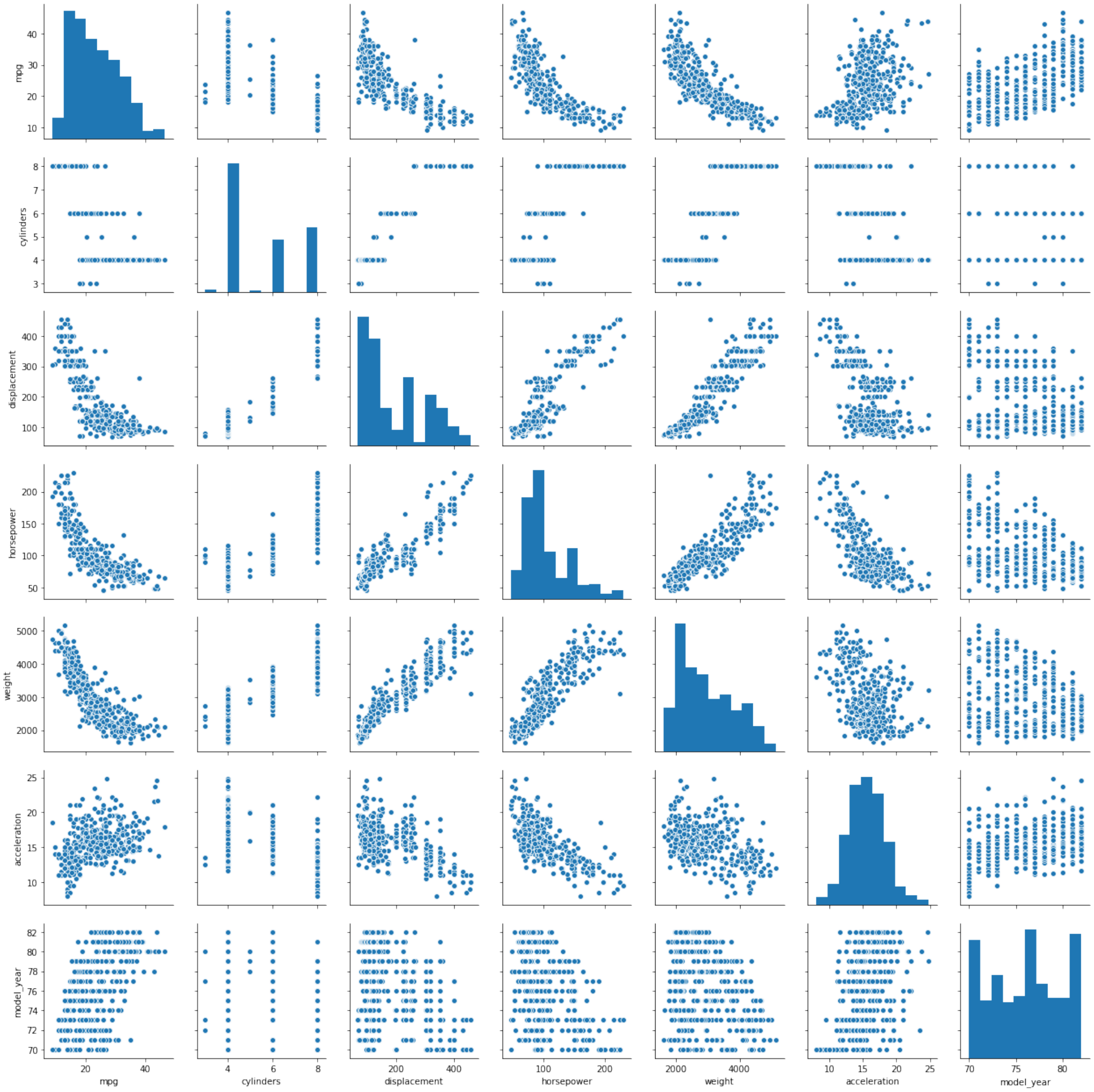
```
[96]: <seaborn.axisgrid.JointGrid at 0x7fd60b45c048>
```



[73]:
```
sns.pairplot(data=auto)
```

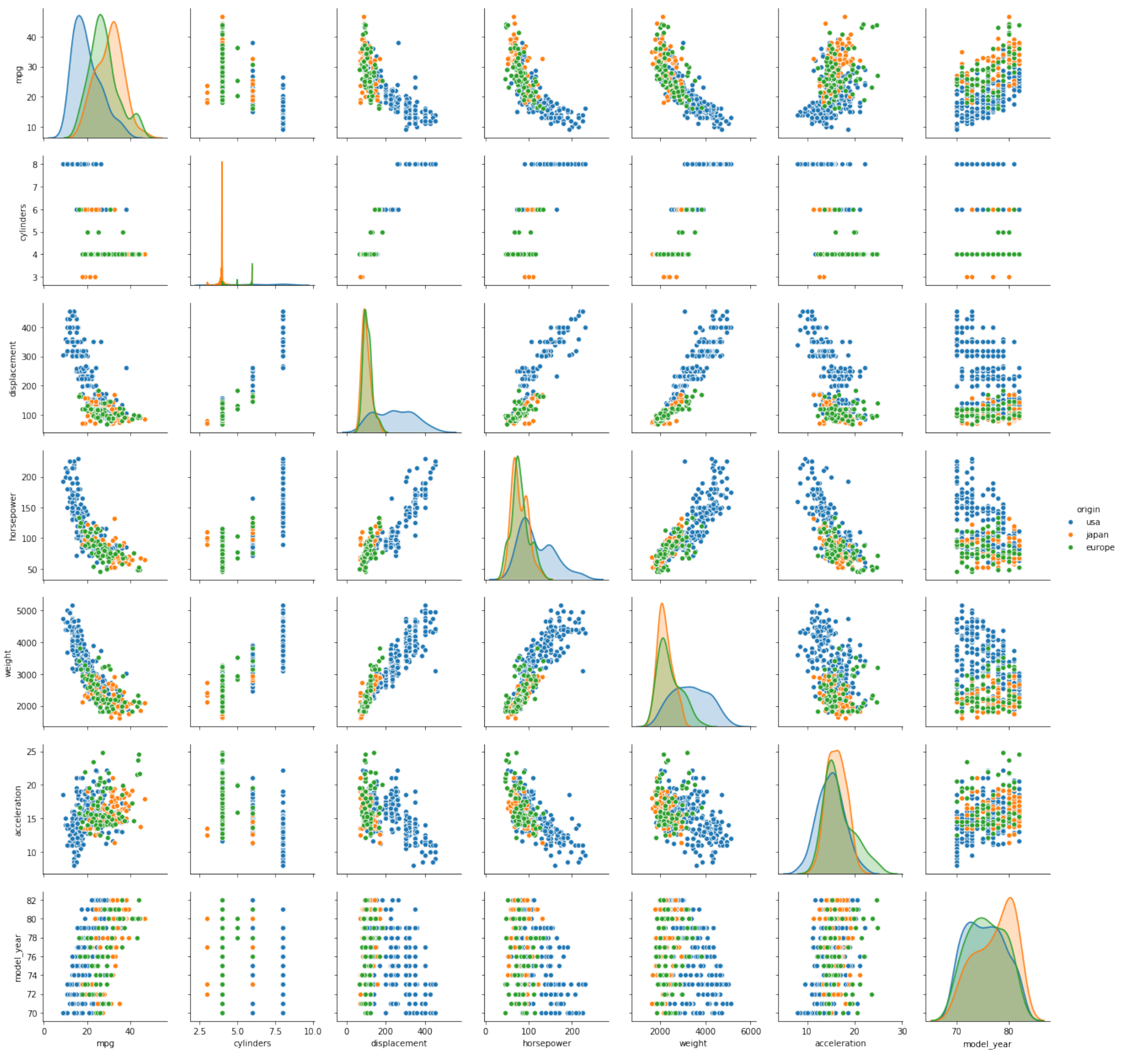
```
/home/kroye/.local/lib/python3.6/site-packages/numpy/lib/histograms.py:829:
RuntimeWarning: invalid value encountered in greater_equal
  keep = (tmp_a >= first_edge)
/home/kroye/.local/lib/python3.6/site-packages/numpy/lib/histograms.py:830:
RuntimeWarning: invalid value encountered in less_equal
  keep &= (tmp_a <= last_edge)
```

[73]:
```
<seaborn.axisgrid.PairGrid at 0x7fd609789400>
```



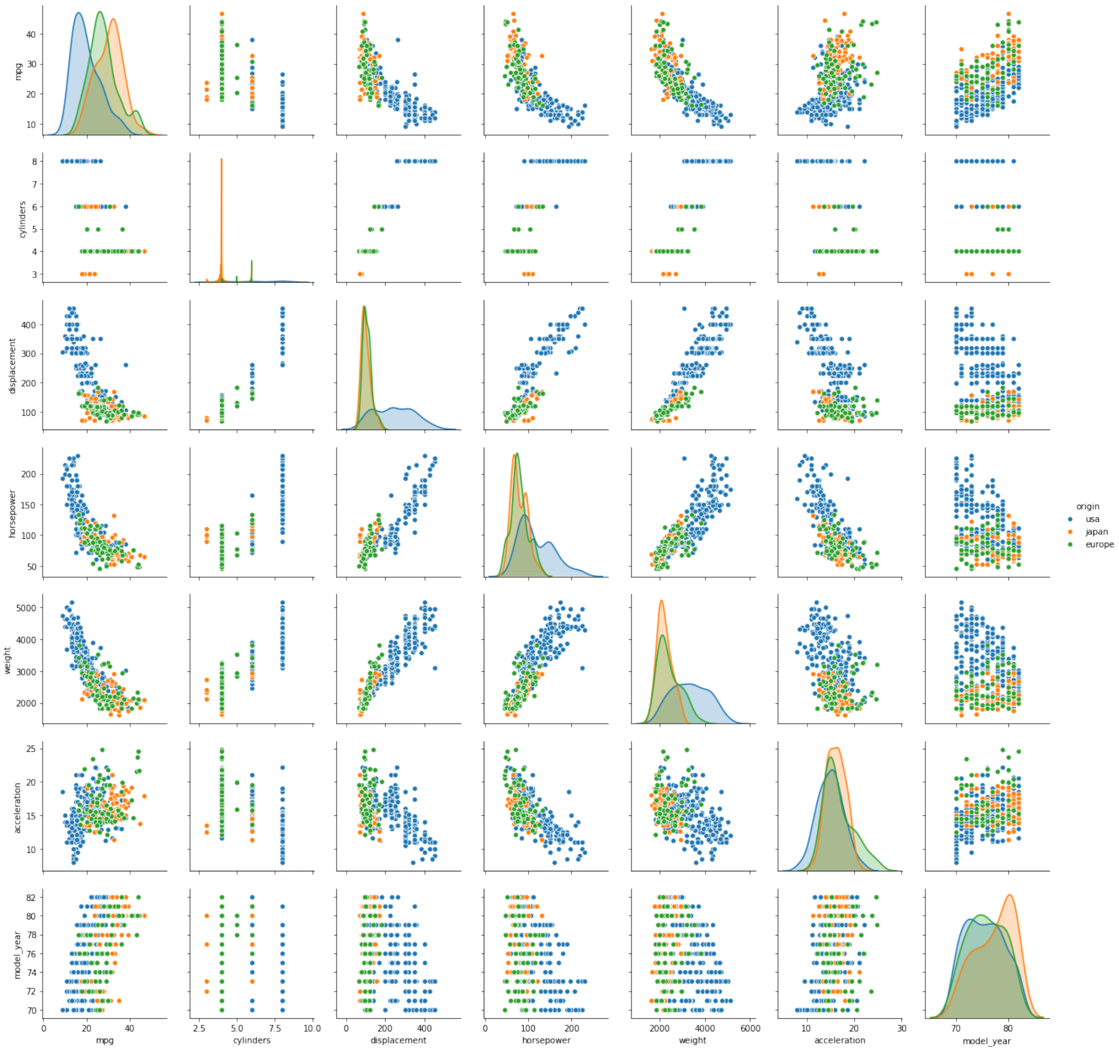
```
[74]: sns.pairplot(data=auto,hue='origin')
```

```
[74]: <seaborn.axisgrid.PairGrid at 0x7fd606c14400>
```



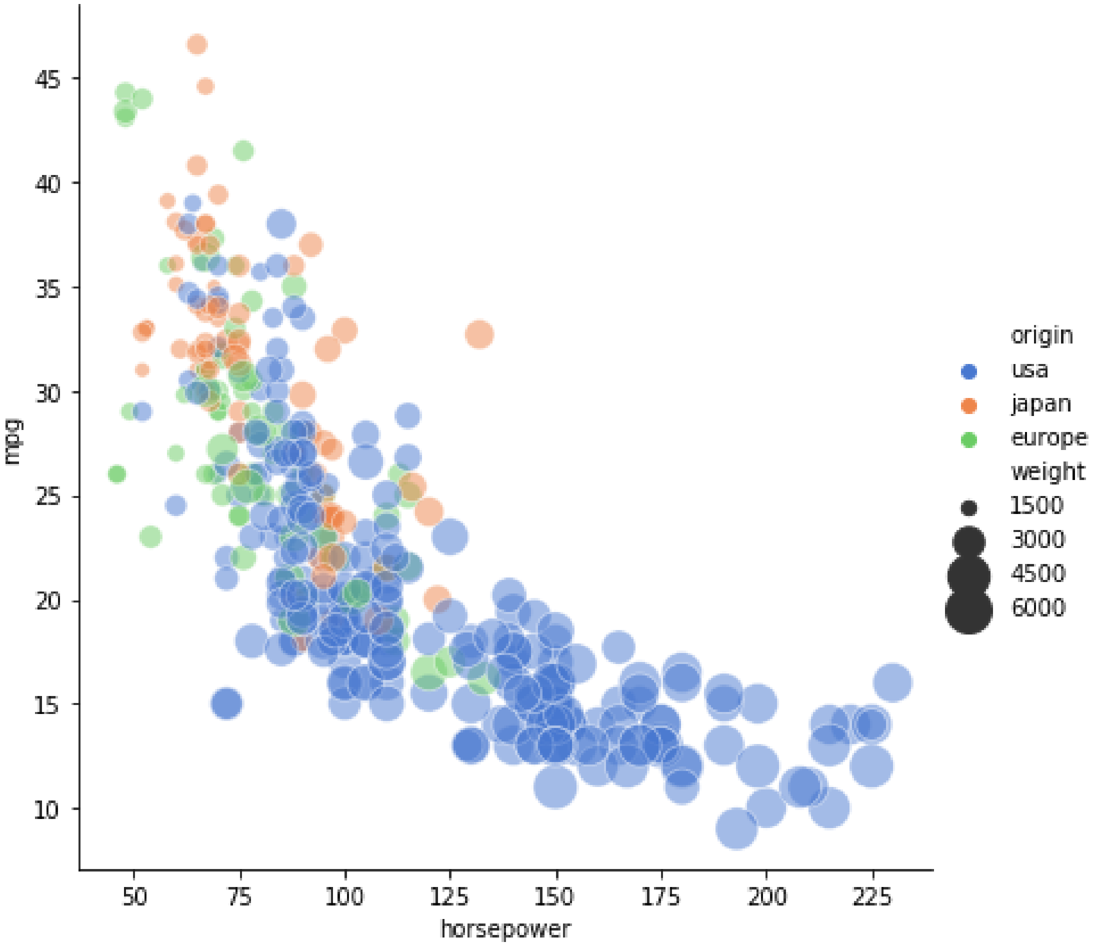
[75]:

```
sns_fig = sns.pairplot(data=auto,hue='origin')
sns_fig.savefig('test.png')
```



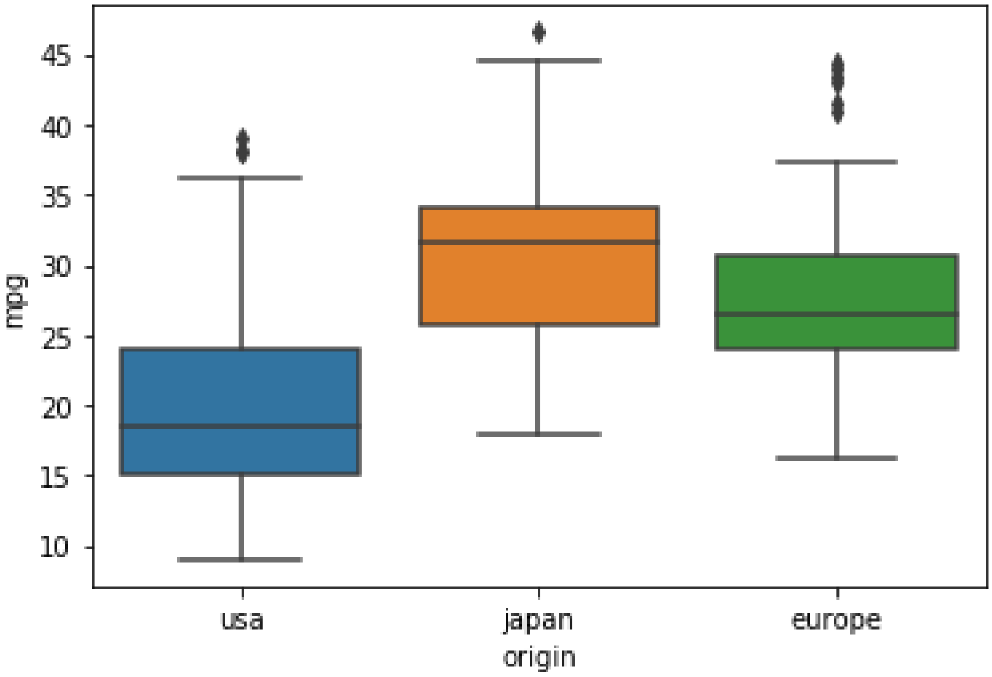
```
[76]: sns.relplot(x="horsepower", y="mpg", hue="origin", size="weight",
                  sizes=(40, 400), alpha=.5, palette="muted",
                  height=6, data=auto)
```

```
[76]: <seaborn.axisgrid.FacetGrid at 0x7fd6032bf6a0>
```



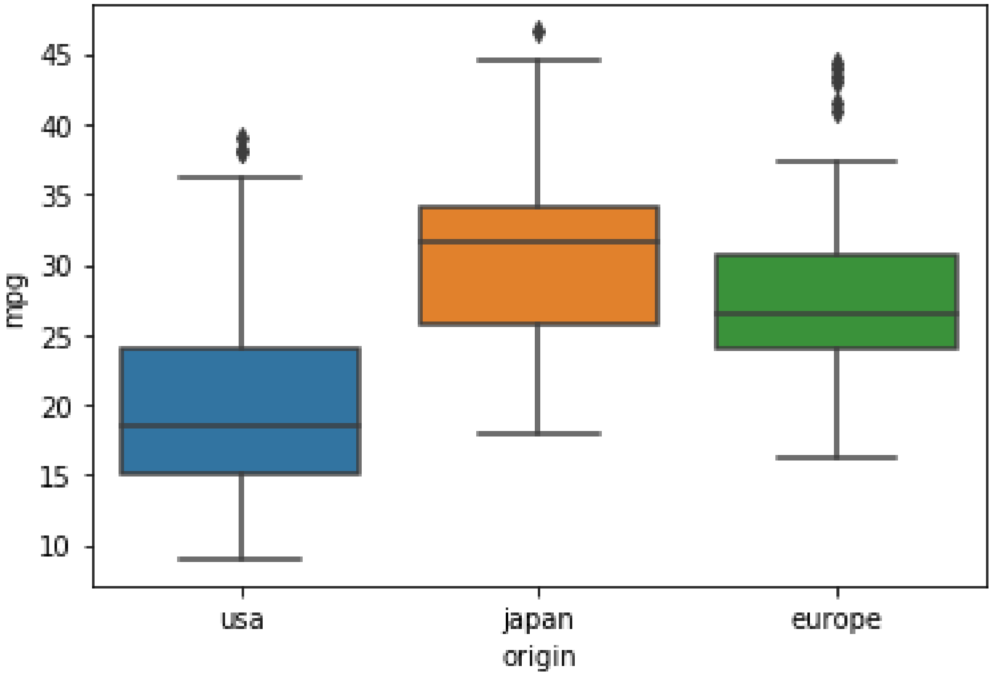
```
[100]: sns.boxplot(x="origin", y="mpg", data = auto)
```

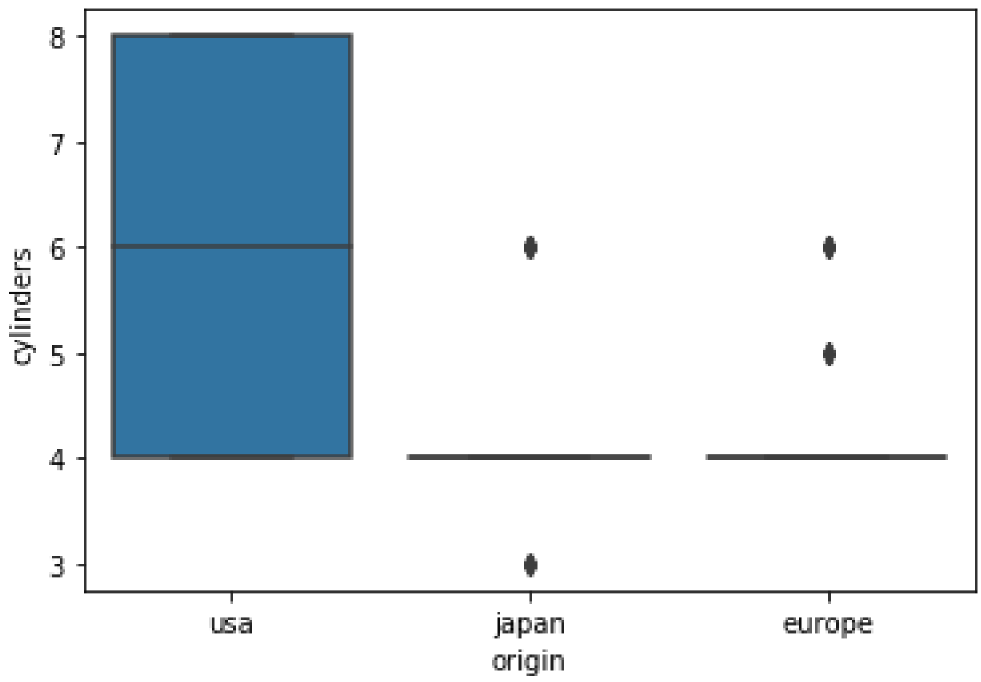
```
[100]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd60a685e48>
```

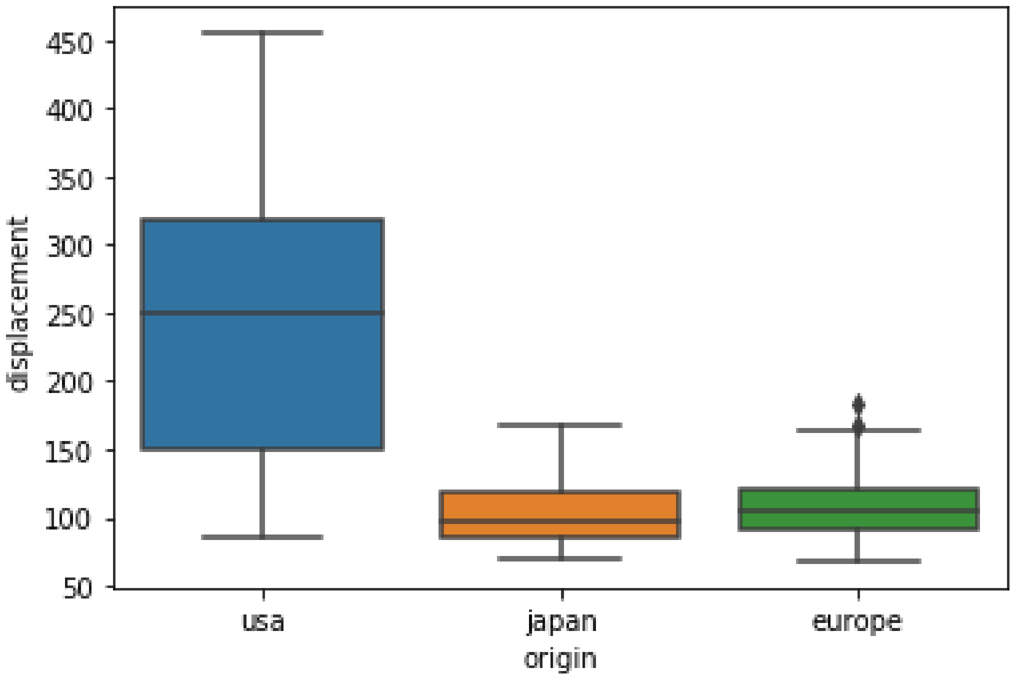


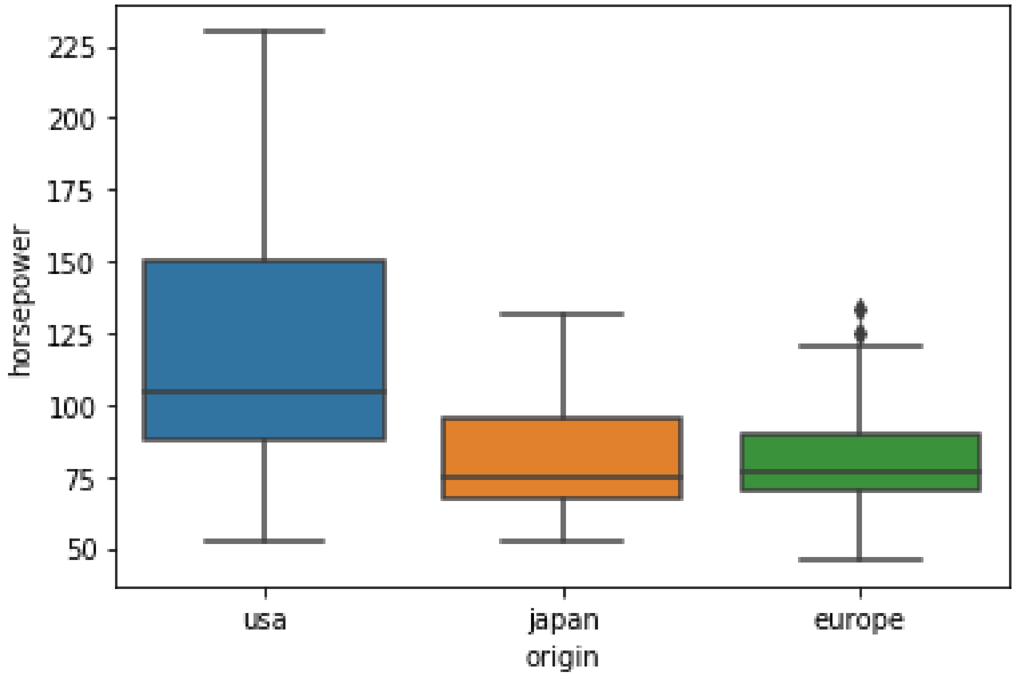
[106]:

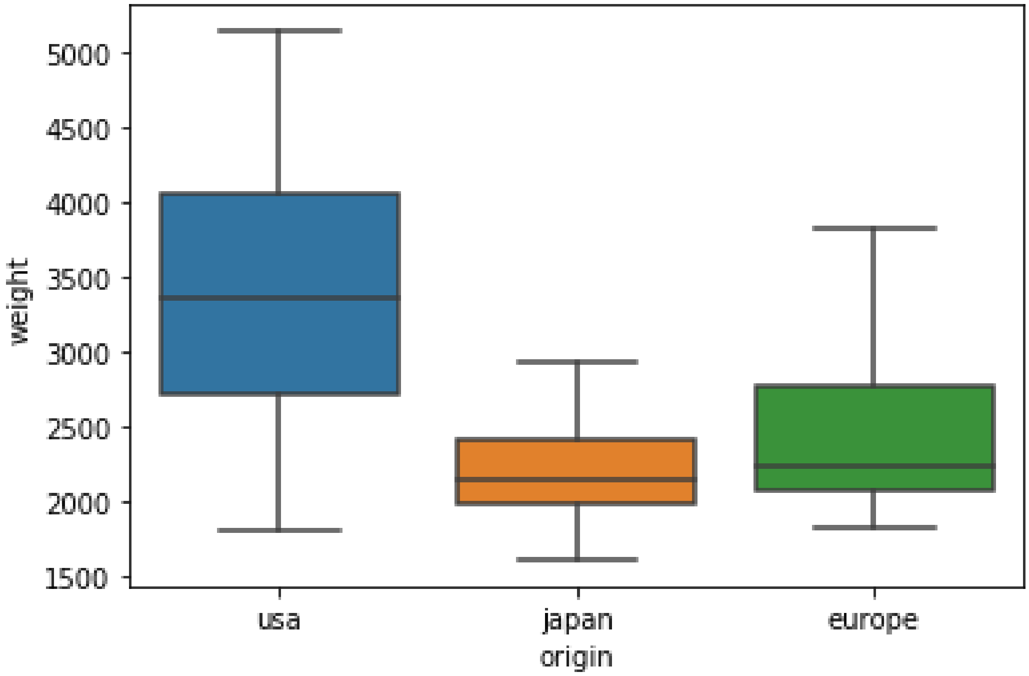
```
for col in auto.columns:
    if (auto[col].dtypes == 'int64' or auto[col].dtypes == 'float64'):
        sns.boxplot(x="origin", y = col, data = auto)
        plt.show()
```

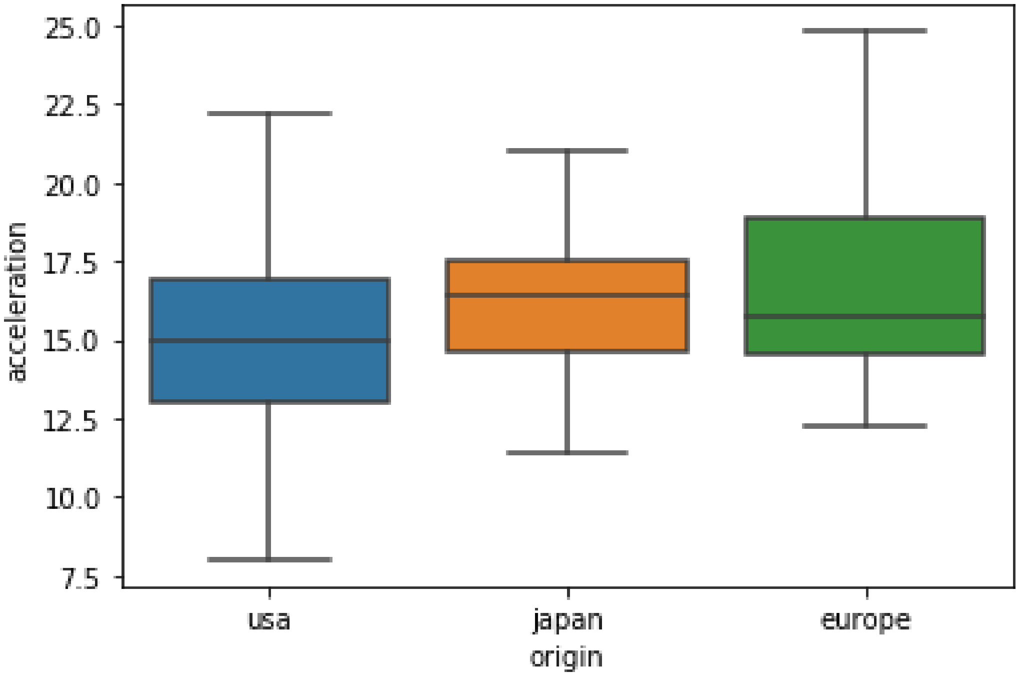


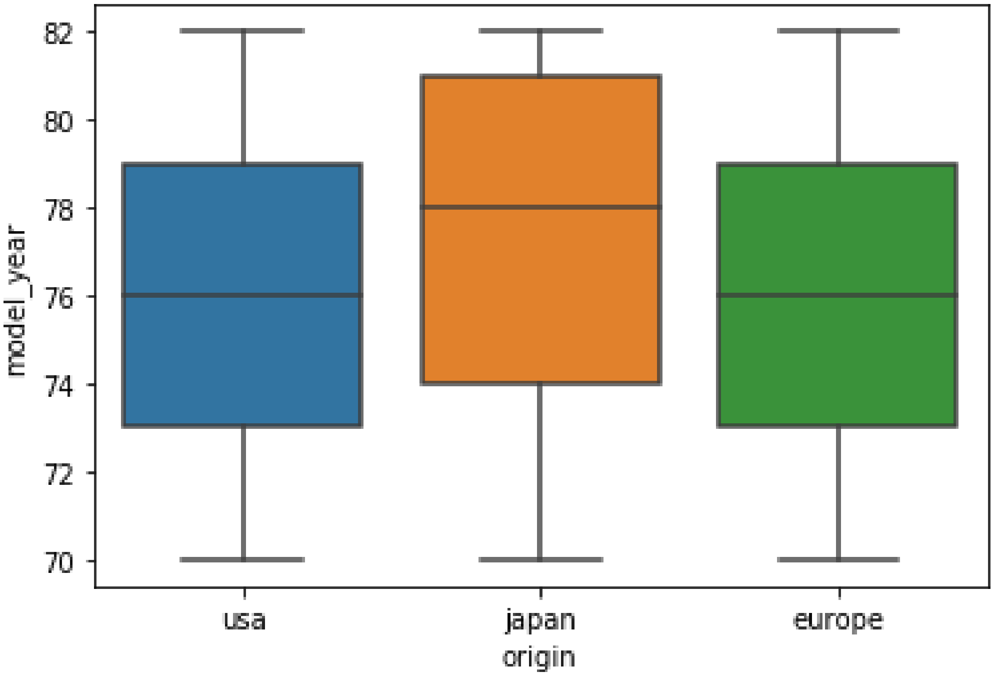

8
7
6
5
4
3
cylinders
usa
japan
europe
origin


450
400
350
300
250
200
150
100
50
displacement
usa
japan
europe
origin


225
200
175
150
125
100
75
50
horsepower
usa
japan
europe
origin


5000
4500
4000
3500
3000
2500
2000
1500
weight
usa
japan
europe
origin


25.0
22.5
20.0
17.5
15.0
12.5
10.0
7.5
acceleration
usa
japan
europe
origin


82
80
78
76
74
72
70
model_year
usa
japan
europe
origin

```
[ ]:
```