In [1]:
```
import pandas as pd
import seaborn as sns
sns.set(color_codes=True)
```

In [2]:
```
import pandas as pd

# Corrected file path with double backslashes
file_path = 'C:\\Users\\Lenovo\\Downloads\\Automobile+%281%29.csv'

# Read the CSV file
auto = pd.read_csv(file_path)
```

In [3]:
```
auto.head()
```

Out[3]:

	symboling	normalized_losses	make	fuel_type	aspiration	number_of_doors	body_sty
0	3	168	alfa-romero	gas	std	two	convertib
1	3	168	alfa-romero	gas	std	two	convertib
2	1	168	alfa-romero	gas	std	two	hatchba
3	2	164	audi	gas	std	four	sed
4	2	164	audi	gas	std	four	sed

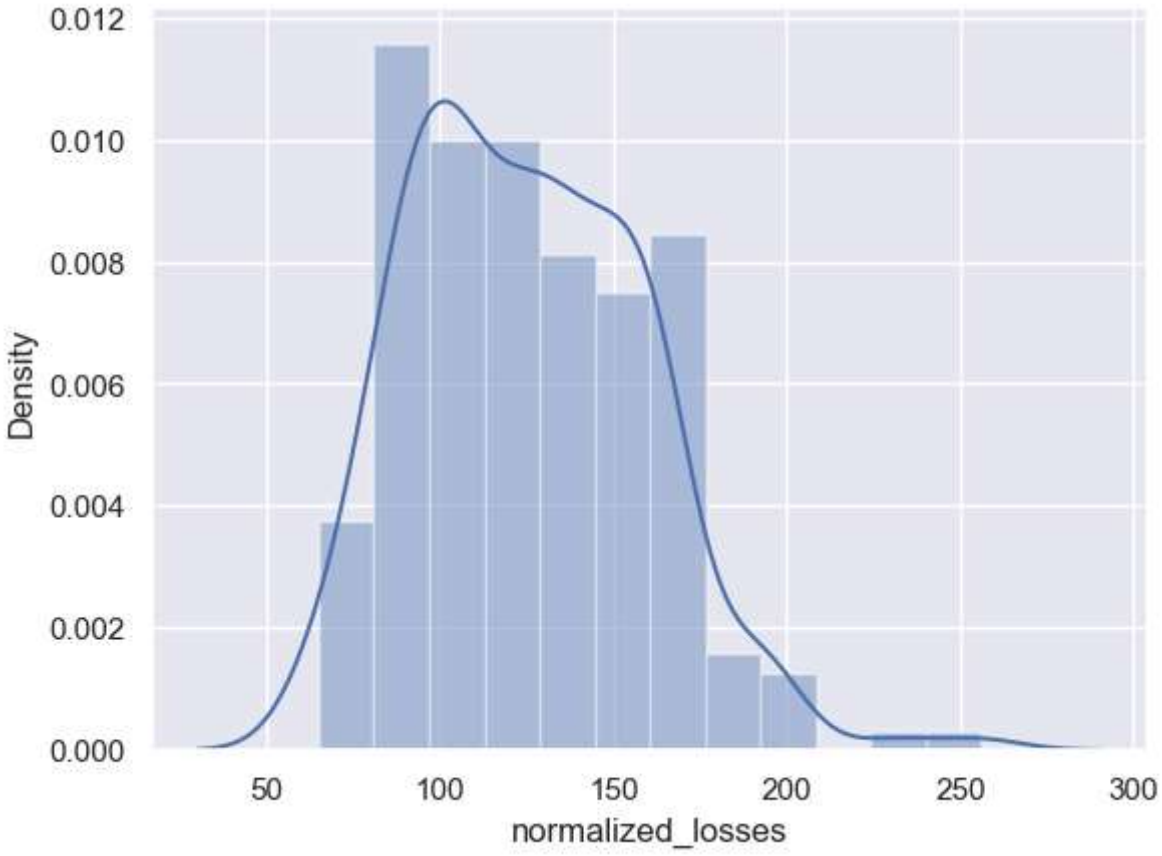
5 rows × 26 columns

Plotting univariate distributions The most convenient way to take a quick look at a univariate distribution in seaborn is the distplot() function. By default, this will draw a histogram and fit a kernel density estimate (KDE).

In [5]:
```
sns.distplot(auto['normalized_losses'])
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_9532\624818176.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(auto['normalized_losses'])
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

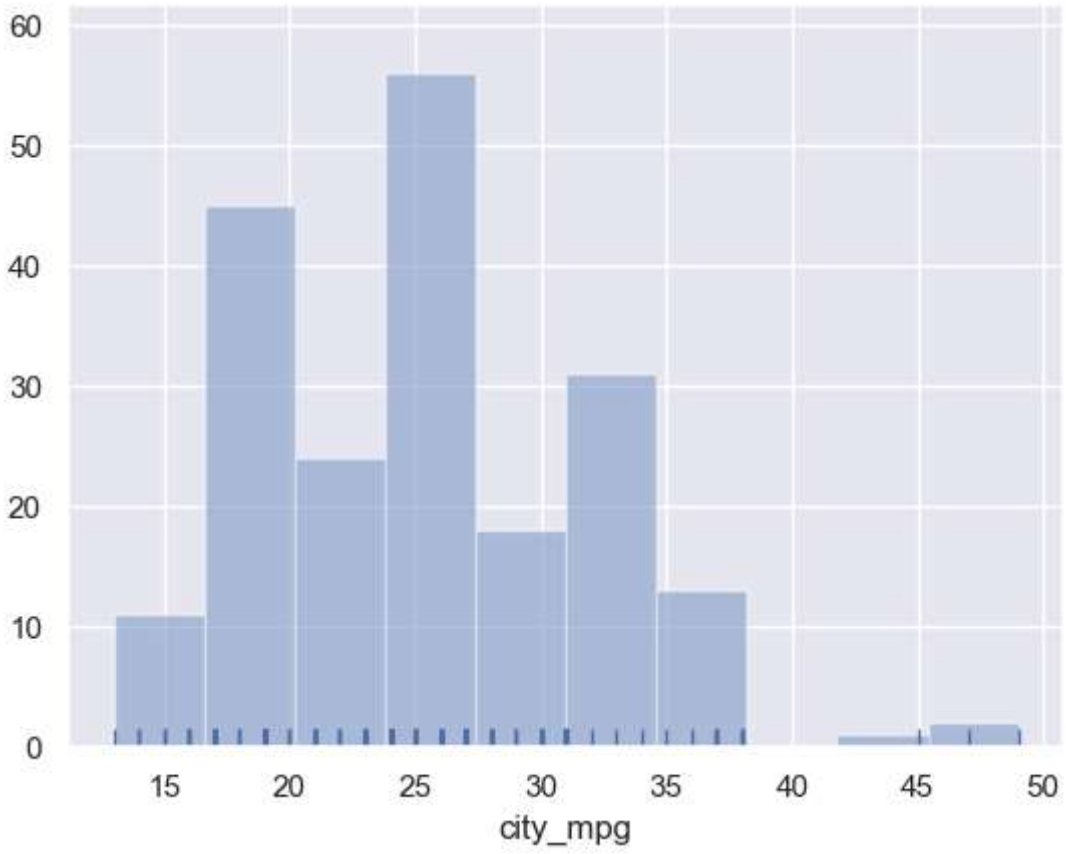
Out[5]: <Axes: xlabel='normalized_losses', ylabel='Density'>



Histograms Histograms are likely familiar, and a hist function already exists in matplotlib. A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin. To illustrate this, let's remove the density curve and add a rug plot, which draws a small vertical tick at each observation. You can make the rug plot itself with the rugplot() function, but it is also available in distplot():

In [6]:
```
sns.distplot(auto['city_mpg'], kde=False, rug=True);
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_9532\1339220190.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(auto['city_mpg'], kde=False, rug=True);
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

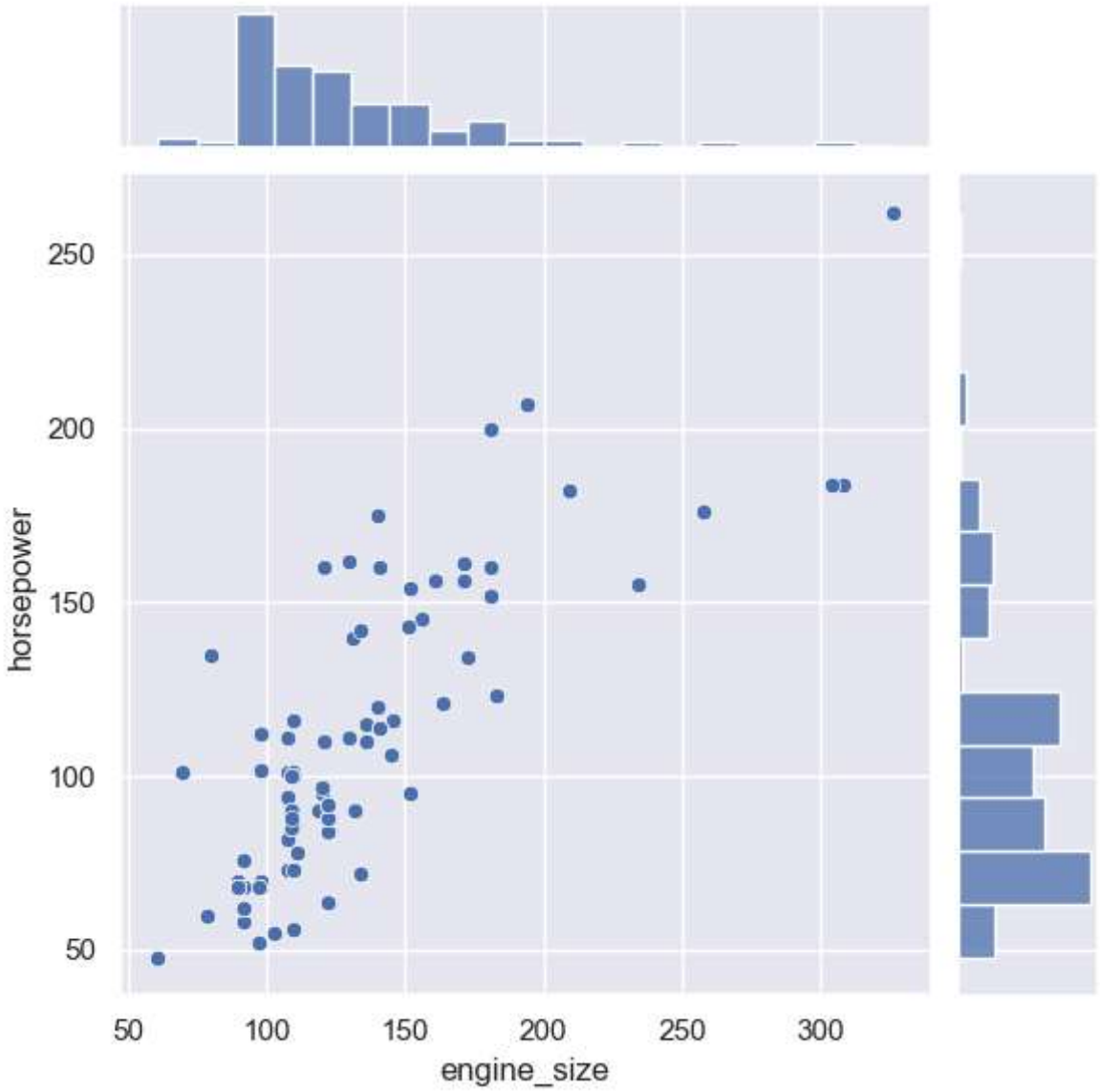


Plotting bivariate distributions It can also be useful to visualize a bivariate distribution of two variables. The easiest way to do this in seaborn is to just use the jointplot() function, which creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes.

Scatterplots The most familiar way to visualize a bivariate distribution is a scatterplot, where each observation is shown with point at the x and y values. This is analgous to a rug plot on two dimensions. You can draw a scatterplot with the matplotlib plt.scatter function, and it is also the default kind of plot shown by the jointplot() function:

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a joint plot for 'engine_size' vs 'horsepower'
sns.jointplot(x=auto['engine_size'], y=auto['horsepower'])

# Show the plot
plt.show()
```

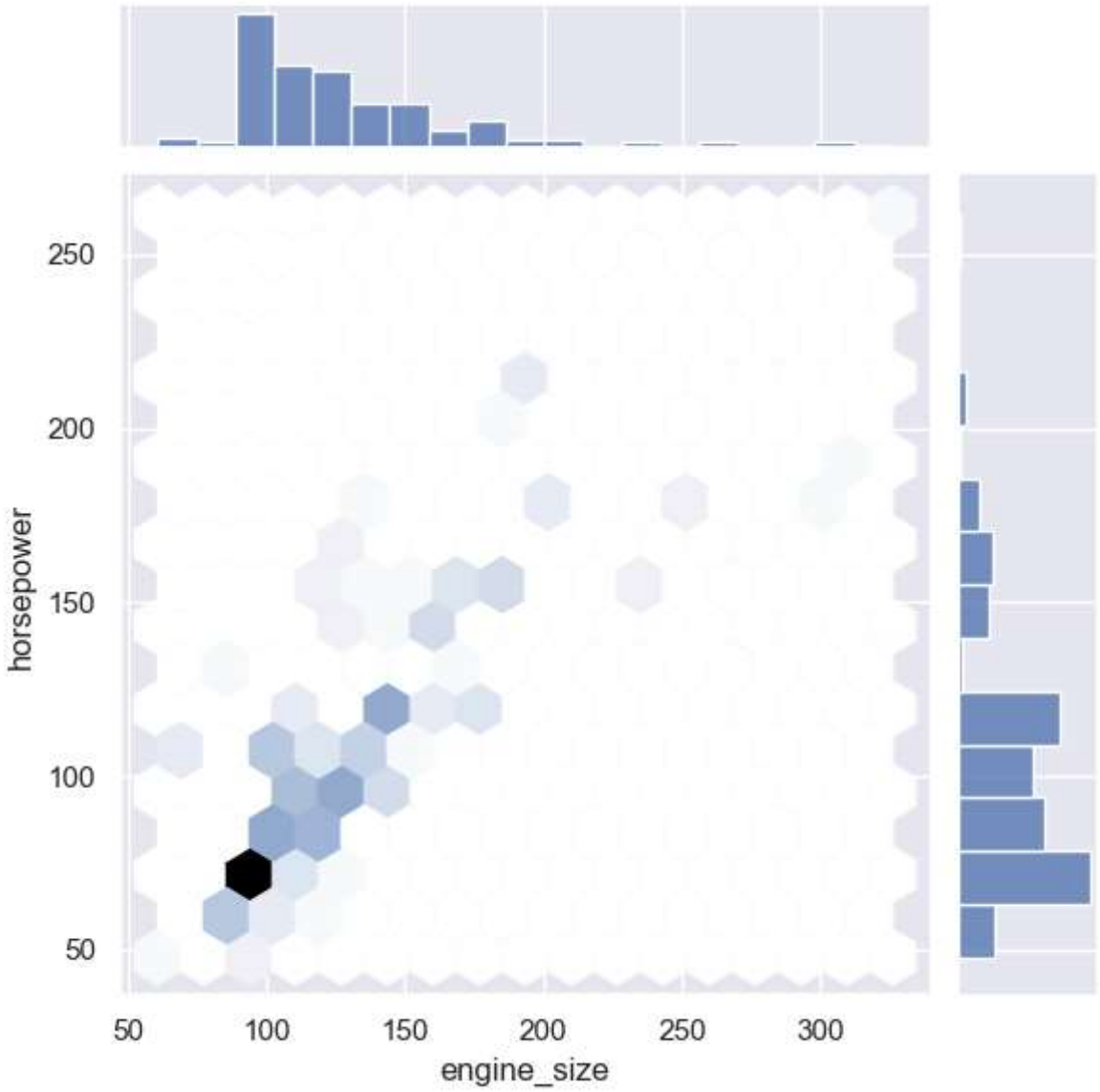
```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Hex Bin Plots

In [10]:
```
sns.jointplot(x=auto['engine_size'], y=auto['horsepower'], kind="hex")

plt.show()
```

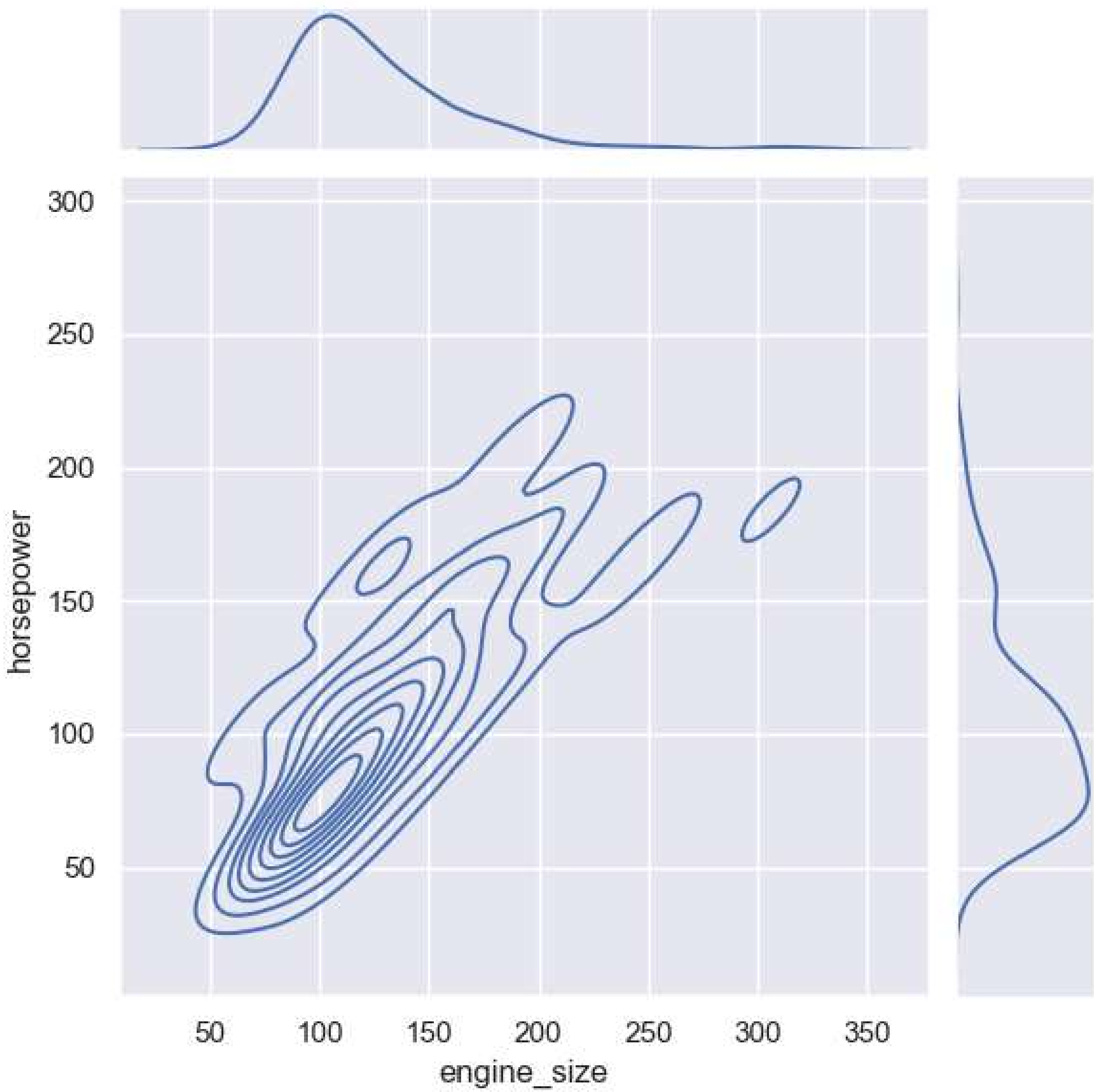
```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Kernel Density Estimation

In [23]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.jointplot(x=auto['engine_size'], y=auto['horsepower'], kind="kde")

plt.show()
```

```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



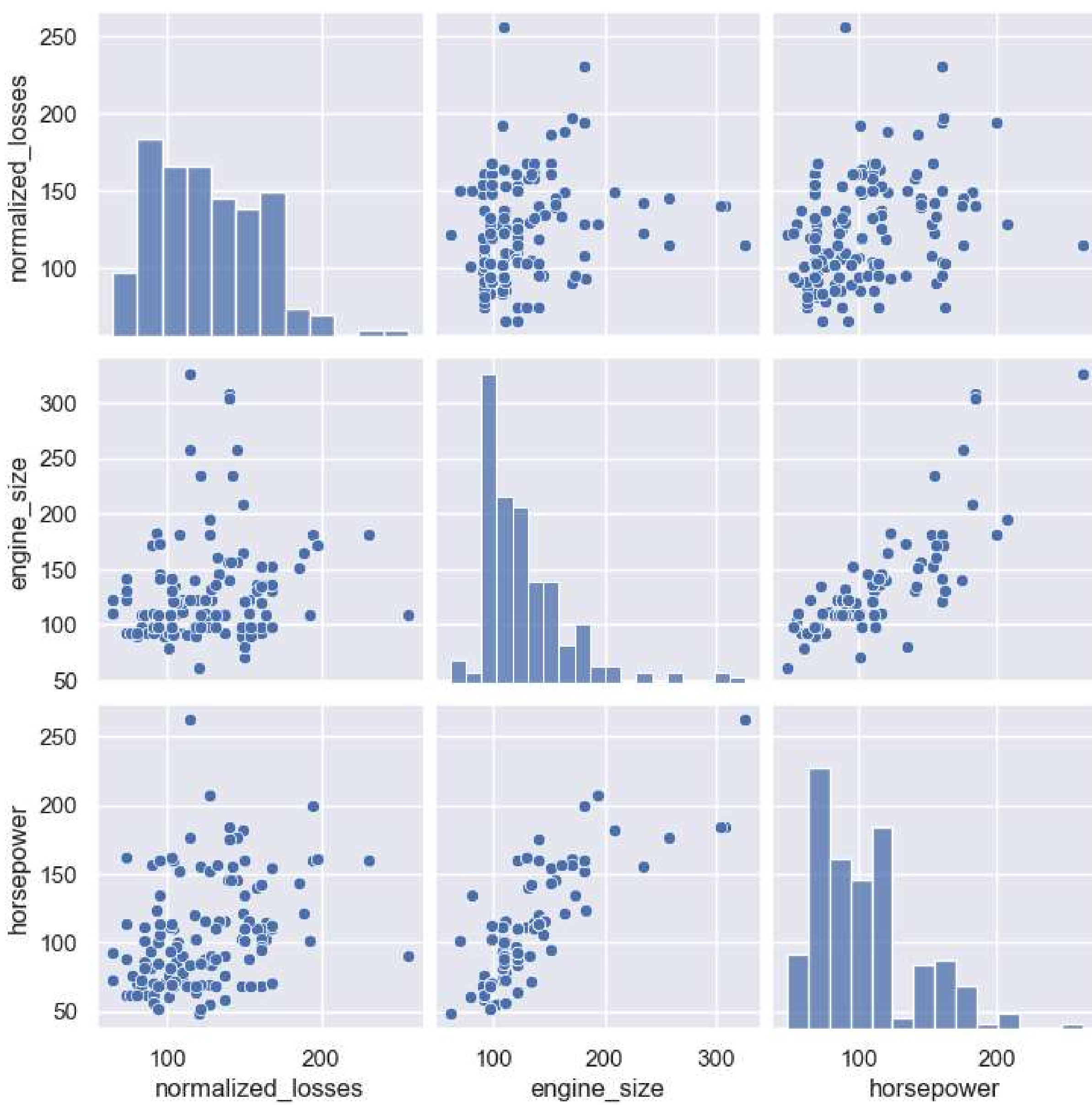
Visualizing pairwise relationships in a dataset To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function. This creates a matrix of axes and shows the

relationship for each pair of columns in a DataFrame. by default, it also draws the univariate distribution of each variable on the diagonal Axes:

In [25]:
```
sns.pairplot(auto[['normalized_losses', 'engine_size', 'horsepower']])
```

```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

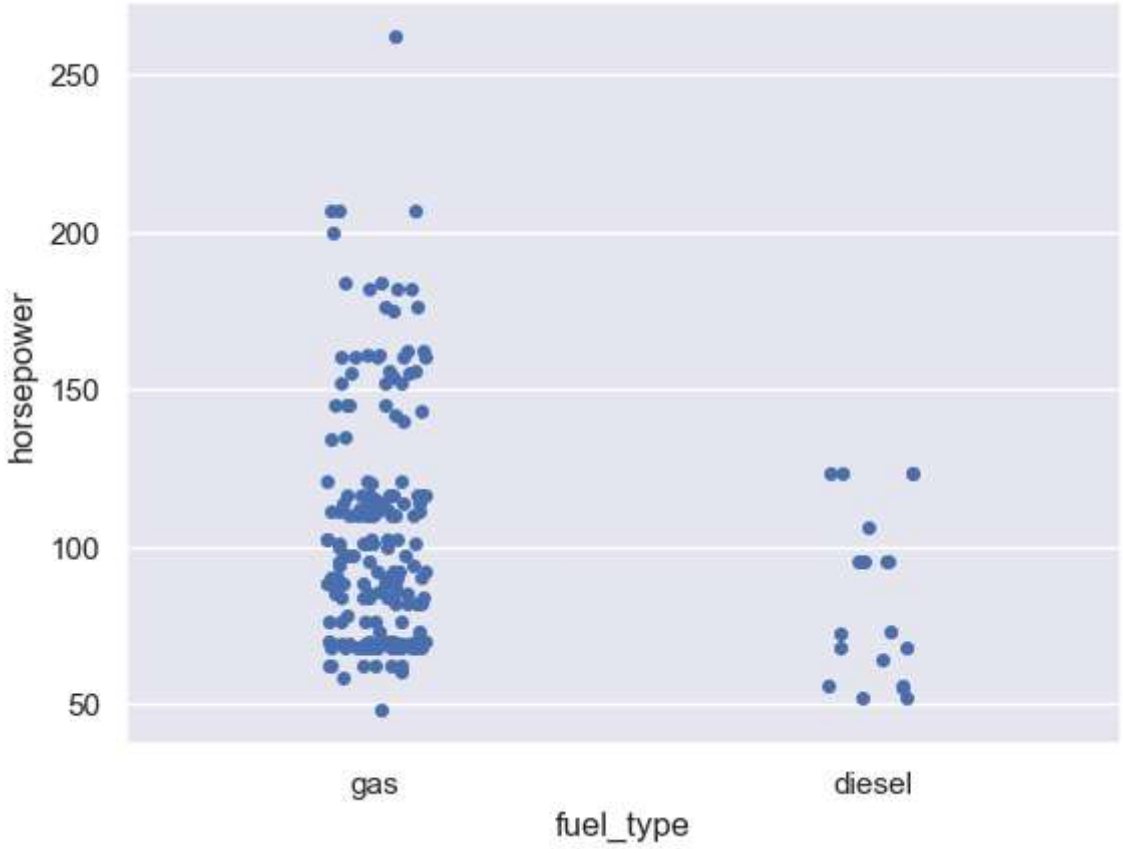
Out[25]: <seaborn.axisgrid.PairGrid at 0x161386b3a90>



Plotting with categorical data

In [29]:
```
sns.stripplot(x=auto['fuel_type'], y=auto['horsepower'])

plt.show()
```

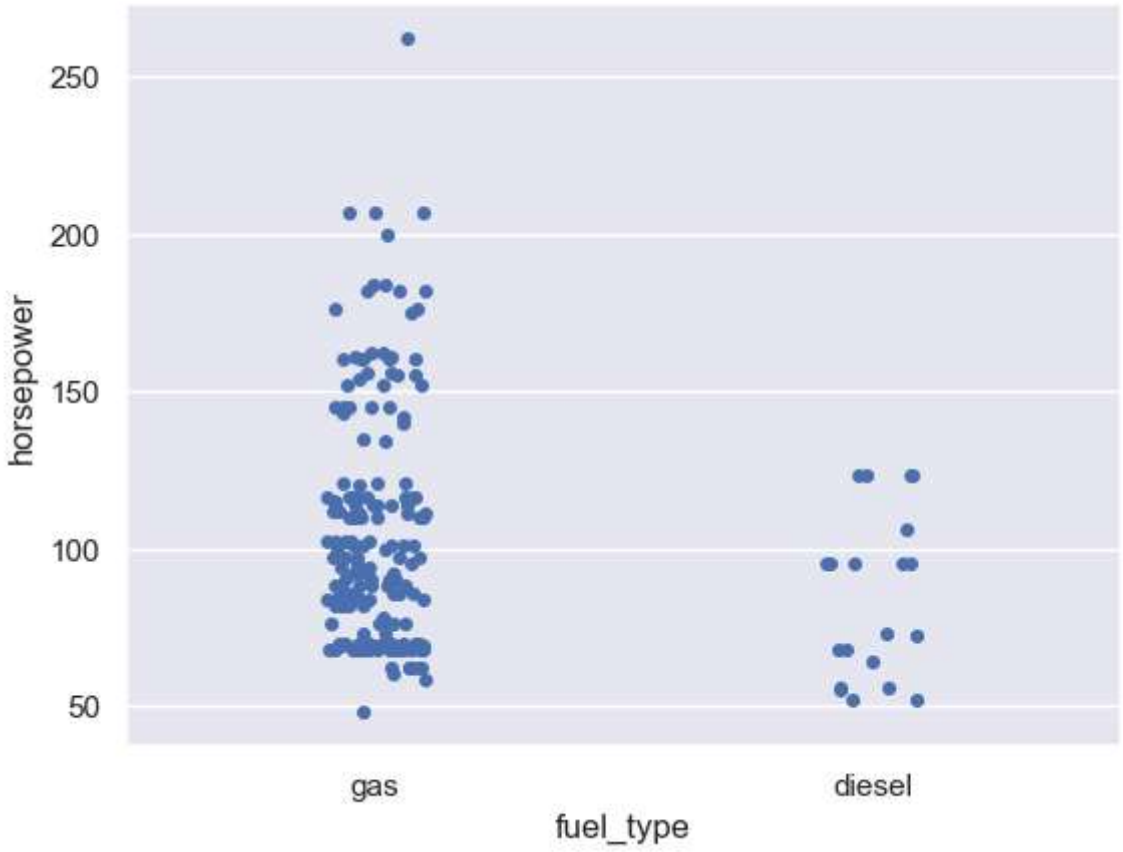
```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In a strip plot, the scatterplot points will usually overlap. This makes it difficult to see the full distribution of data. One easy solution is to adjust the positions (only along the categorical axis) using some random “jitter"

In [33]:
```
sns.stripplot(x='fuel_type', y='horsepower', data=auto, jitter=True)

plt.show()
```

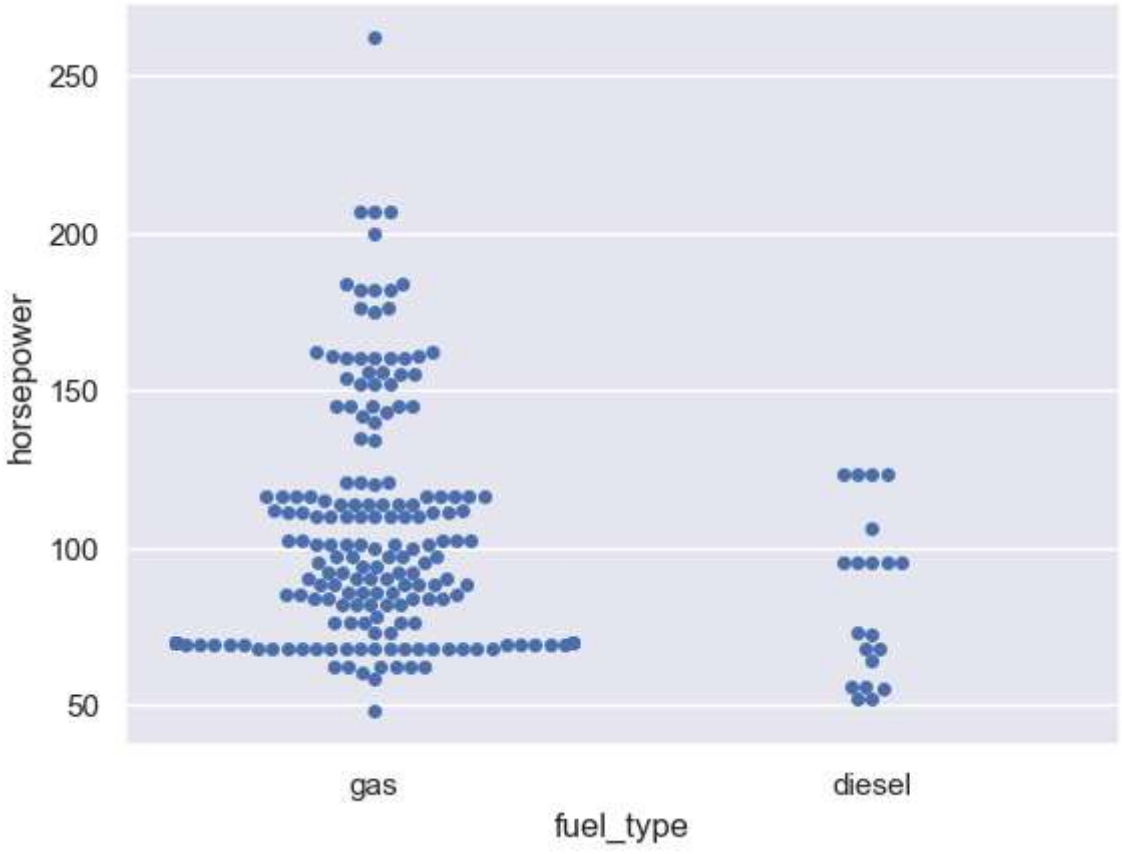
```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



A different approach would be to use the function swarmplot(), which positions each scatterplot point on the categorical axis with an algorithm that avoids overlapping points:

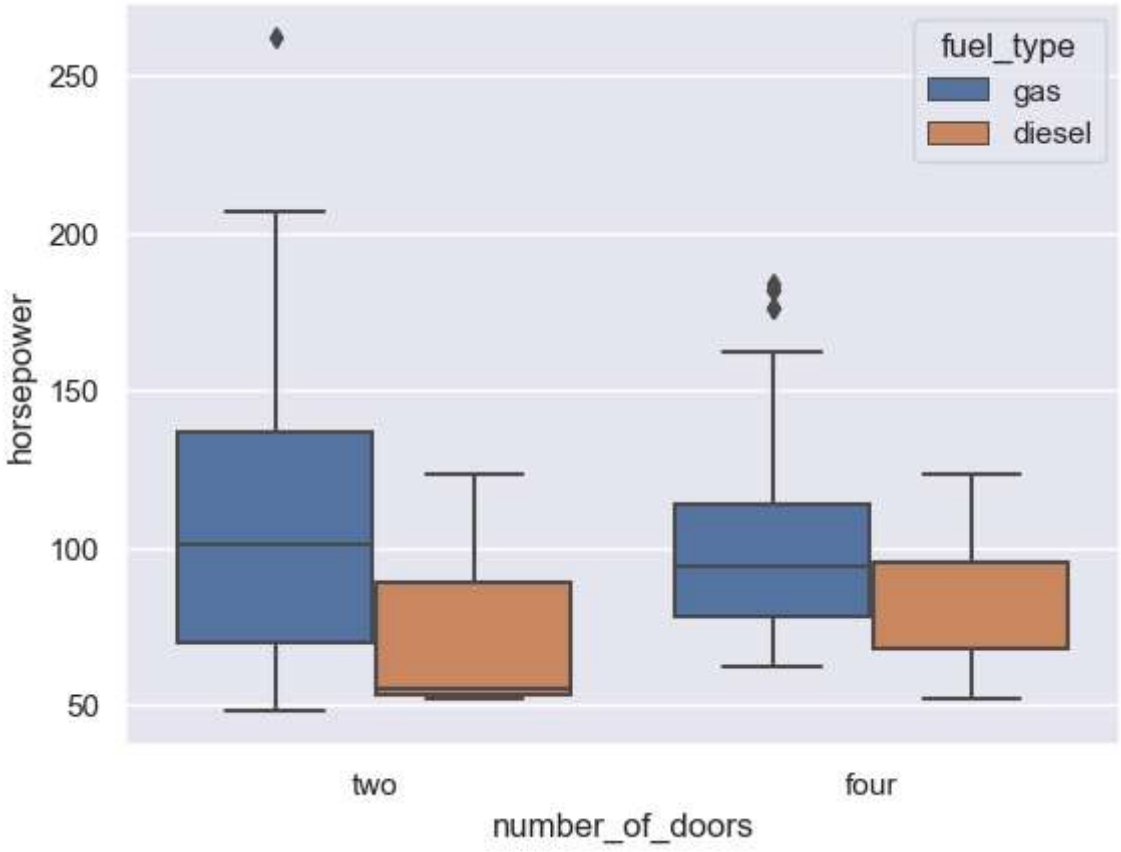
In [37]:
```
sns.swarmplot(x='fuel_type', y='horsepower', data=auto)

plt.show()
```

```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



Boxplots The first is the familiar boxplot(). This kind of plot shows the three quartile values of the distribution along with extreme values. The "whiskers" extend to points that lie within 1.5 IQRs of the lower and upper quartile, and then observations that fall outside this range are displayed independently. Importantly, this means that each value in the boxplot corresponds to an actual observation in the data:

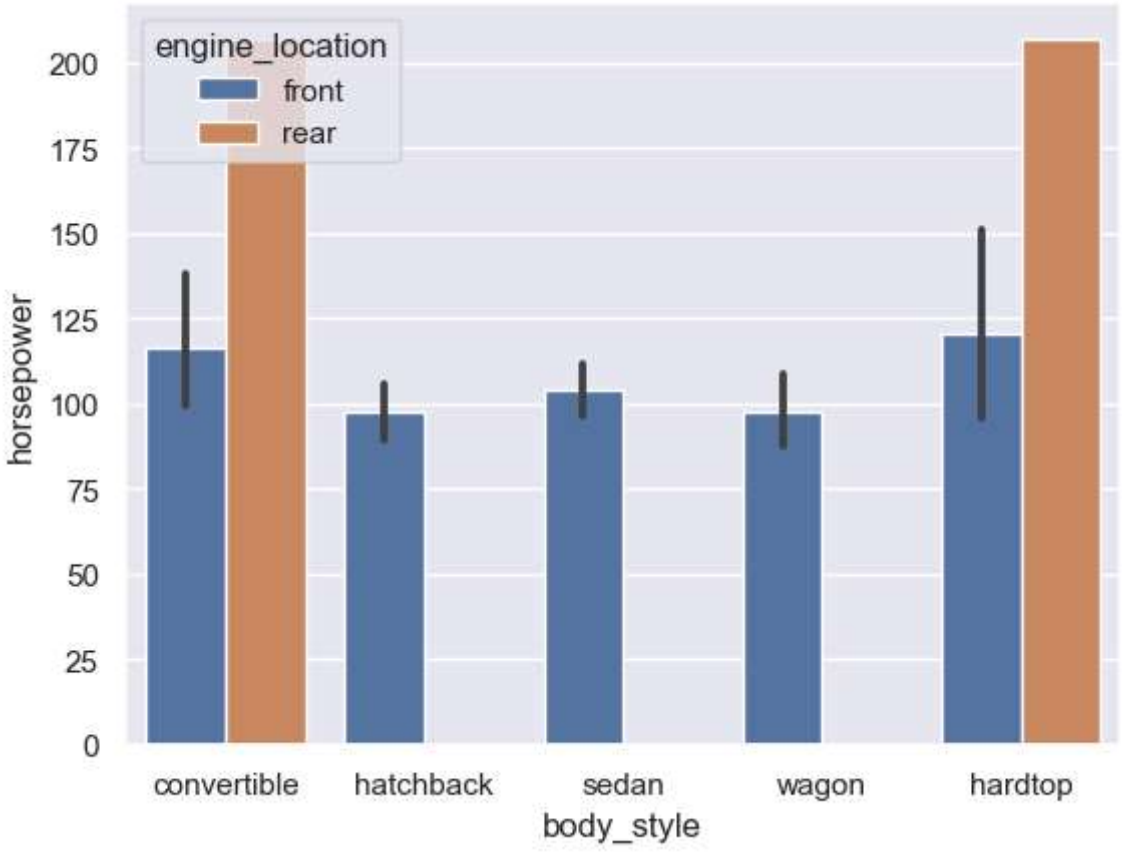
In [39]:
```
sns.boxplot(x='number_of_doors', y='horsepower', hue='fuel_type', data=auto)

plt.show()
```



Bar plots A familiar style of plot that accomplishes this goal is a bar plot. In seaborn, the barplot() function operates on a full dataset and shows an arbitrary estimate, using the mean by default. When there are multiple observations in each category, it also uses bootstrapping to compute a confidence interval around the estimate and plots that using error bars:

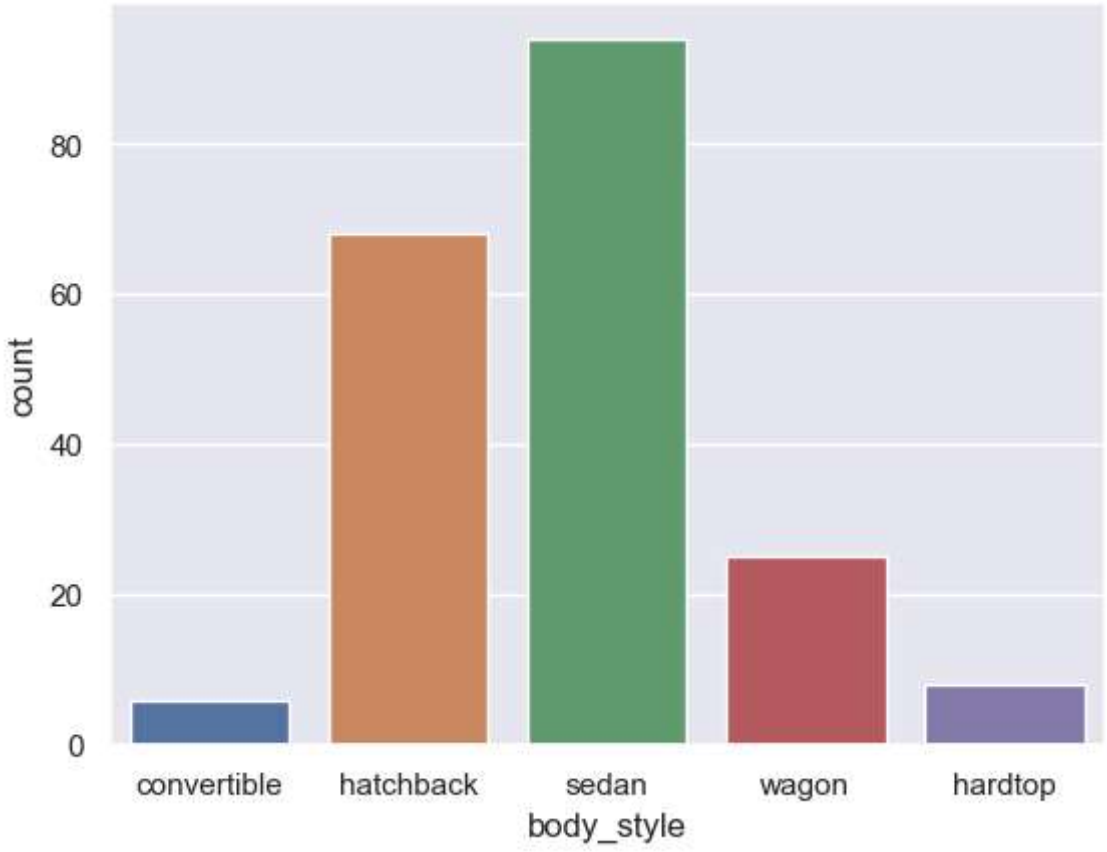
Bar plots include 0 in the quantitative axis range, and they are a good choice when 0 is a meaningful value for the quantitative variable, and you want to make comparisons against it.

In [41]:
```
sns.barplot(x='body_style', y='horsepower', hue='engine_location', data=auto)

plt.show()
```



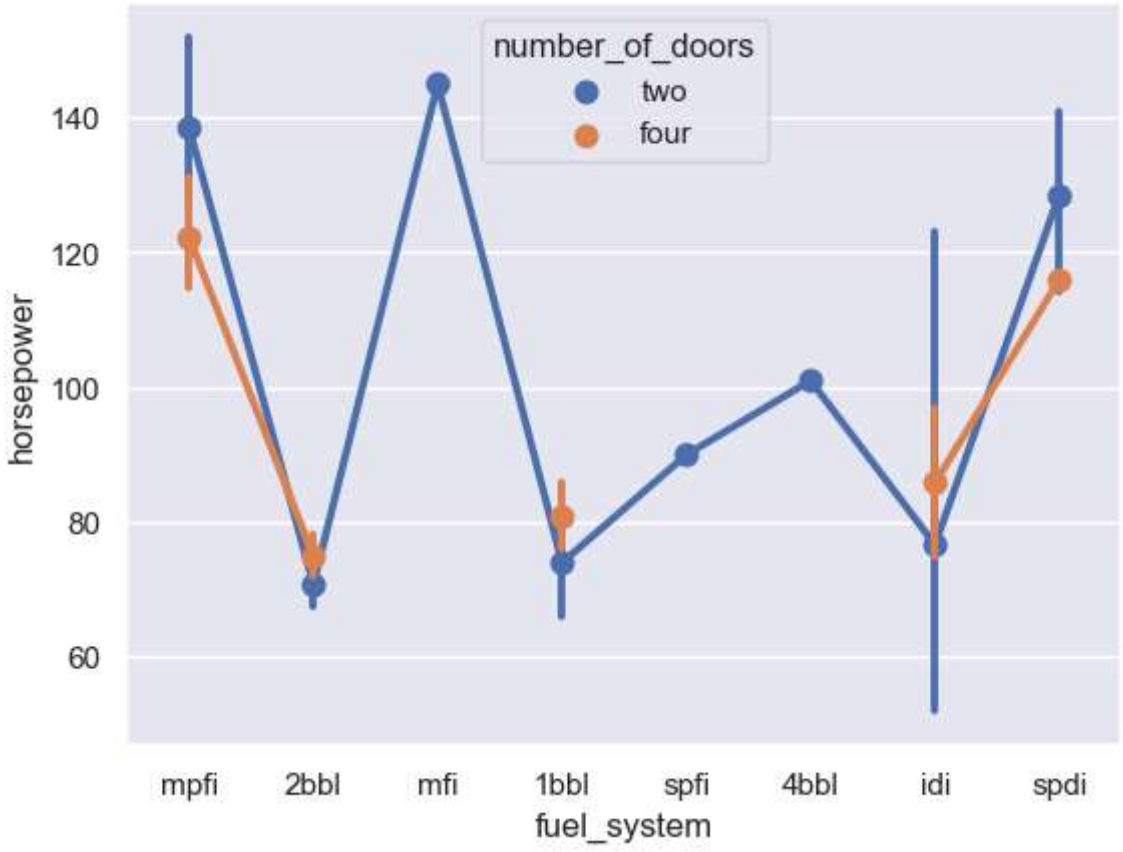
A special case for the bar plot is when you want to show the number of observations in each category rather than computing a statistic for a second variable. This is similar to a histogram over a categorical, rather than quantitative, variable. In seaborn, it's easy to do so with the countplot() function:

In [43]:
```
sns.countplot(x='body_style', data=auto)

plt.show()
```



Point plots An alternative style for visualizing the same information is offered by the pointplot() function. This function also encodes the value of the estimate with height on the other axis, but rather than show a full bar it just plots the point estimate and confidence interval. Additionally, pointplot connects points from the same hue category. This makes it easy to see how the main relationship is changing as a function of a second variable, because your eyes are quite good at picking up on differences of slopes:

In [45]:
```
sns.pointplot(x='fuel_system', y='horsepower', hue='number_of_doors', data=auto)

plt.show()
```



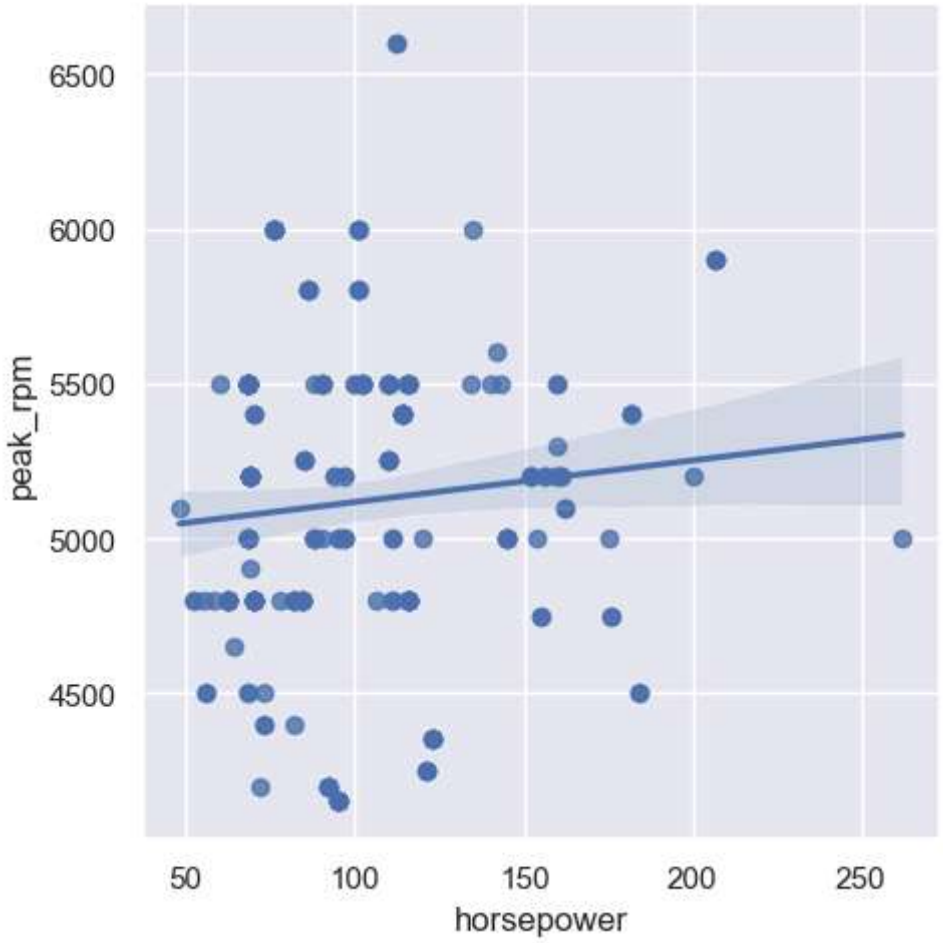
Drawing multi-panel categorical plots

In [47]:
```
sns.catplot(x="fuel_type",
            y="horsepower",
            hue="number_of_doors",
            col="engine_location",
            data=auto,
            kind="swarm")

plt.show()
```

```
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning:
use_inf_as_na option is deprecated and will be removed in a future version. Convert
inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\categorical.py:3544: UserWarnin
g: 6.2% of the points cannot be placed; you may want to decrease the size of the mar
kers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\Lenovo\anaconda3\Lib\site-packages\seaborn\categorical.py:3544: UserWarnin
g: 5.1% of the points cannot be placed; you may want to decrease the size of the mar
kers or use stripplot.
  warnings.warn(msg, UserWarning)
```

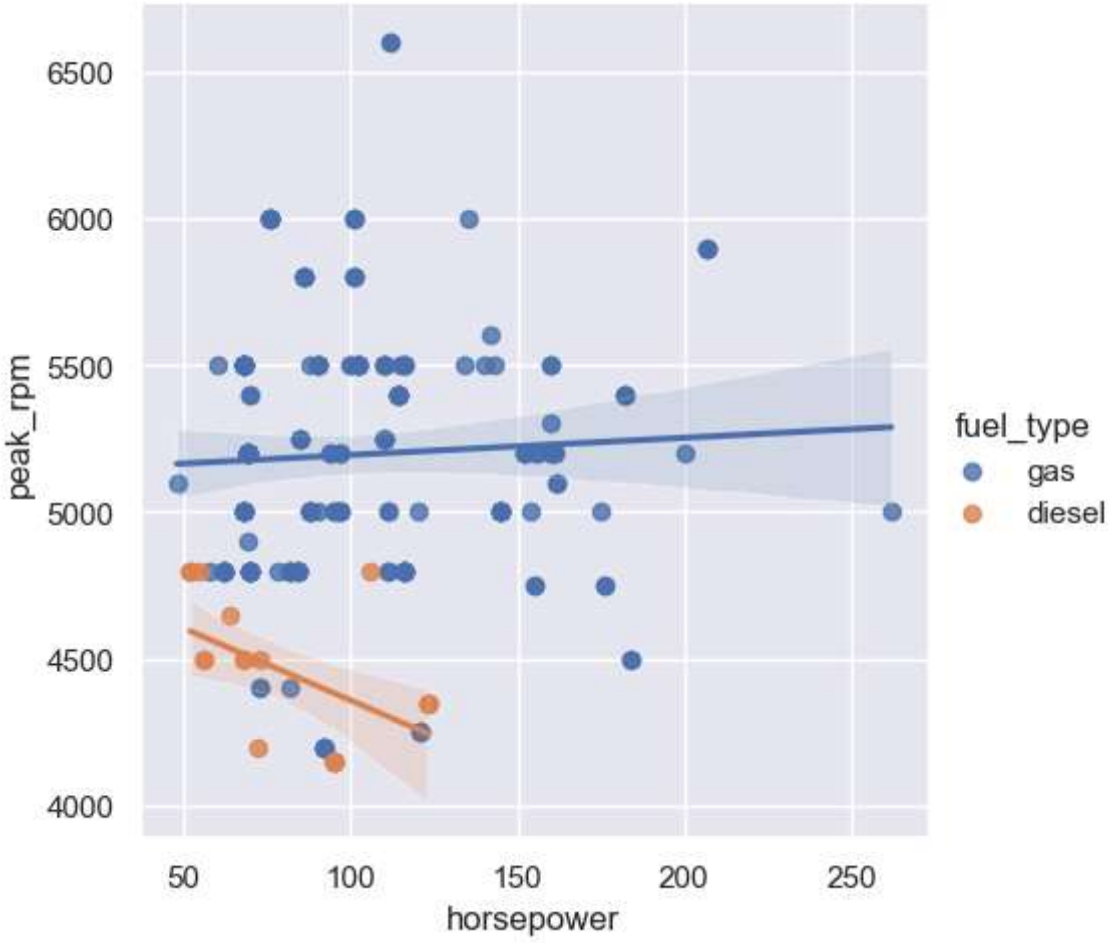
Function to draw linear regression models lmplot() is one of the most widely used function to quickly plot the Linear Relationship b/w 2 variables

In [49]:
```
sns.lmplot(x="horsepower", y="peak_rpm", data=auto)

plt.show()
```



In [51]:
```
sns.lmplot(x="horsepower", y="peak_rpm",hue="fuel_type", data=auto)
```

Out[51]: <seaborn.axisgrid.FacetGrid at 0x16138a62710>



In [ ]:

In [ ]:

In [ ]:

In [ ]: