Python data visualization with Seaborn Library

This work aims at depicting various plots and visualization techniques which can be done using seaborn library.It covers the basic to advance level plotting functions of the library.It contains several examples which will give you hands-on experience in generating plots in python.

Table of Contents

1.Introduction to Seaborn

Seaborn provides a high-level interface to Matplotlib, a powerful but sometimes unwieldy Python visualization library.On Seaborn's official website, they state:

If matplotlib "tries to make easy things easy and hard things possible", seaborn tries to make a well-defined set of hard things easy too.

Features of Seaborn :

- Using default themes that are aesthetically pleasing.
- Setting custom color palettes.
- Making attractive statistical plots.
- Easily and flexibly displaying distributions.
- Visualizing information from matrices and DataFrames.

Those last three points are why **Seaborn is our tool of choice for Exploratory Analysis**. It makes it very easy to "get to know" your data quickly and efficiently.

2.Seaborn vs Matplotlib

It is summarized that if Matplotlib "tries to make easy things easy and hard things possible", Seaborn tries to make a well-defined set of hard things easy too."

Seaborn helps resolve the two major problems faced by Matplotlib; the problems are –

- Default Matplotlib parameters
- Working with data frames

As Seaborn compliments and extends Matplotlib, the learning curve is quite gradual. If you know Matplotlib, you are already half way through Seaborn.

3.Import libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
sns.set(color_codes=True)
import matplotlib.pyplot as plt
import matplotlib as mpl
```

In [2]:
```
auto = pd.read_csv('/input/automobile-dataset/Automobile_data.csv')
```

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

Out[3]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compr
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.4	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.4	

5 rows × 26 columns

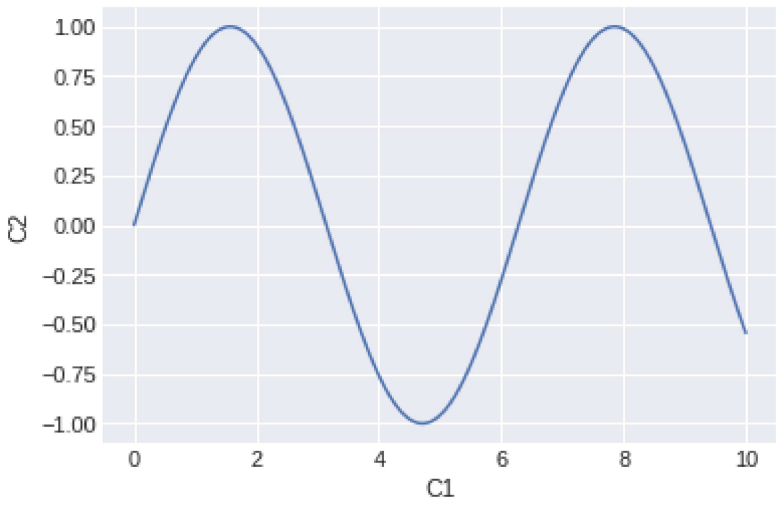
4,Line Charts

In [4]:
```
col1 = np.linspace(0, 10, 1000)
col2 = np.sin(col1)
df = pd.DataFrame({"C1" : col1 , "C2" :col2})
df.head(10)
```

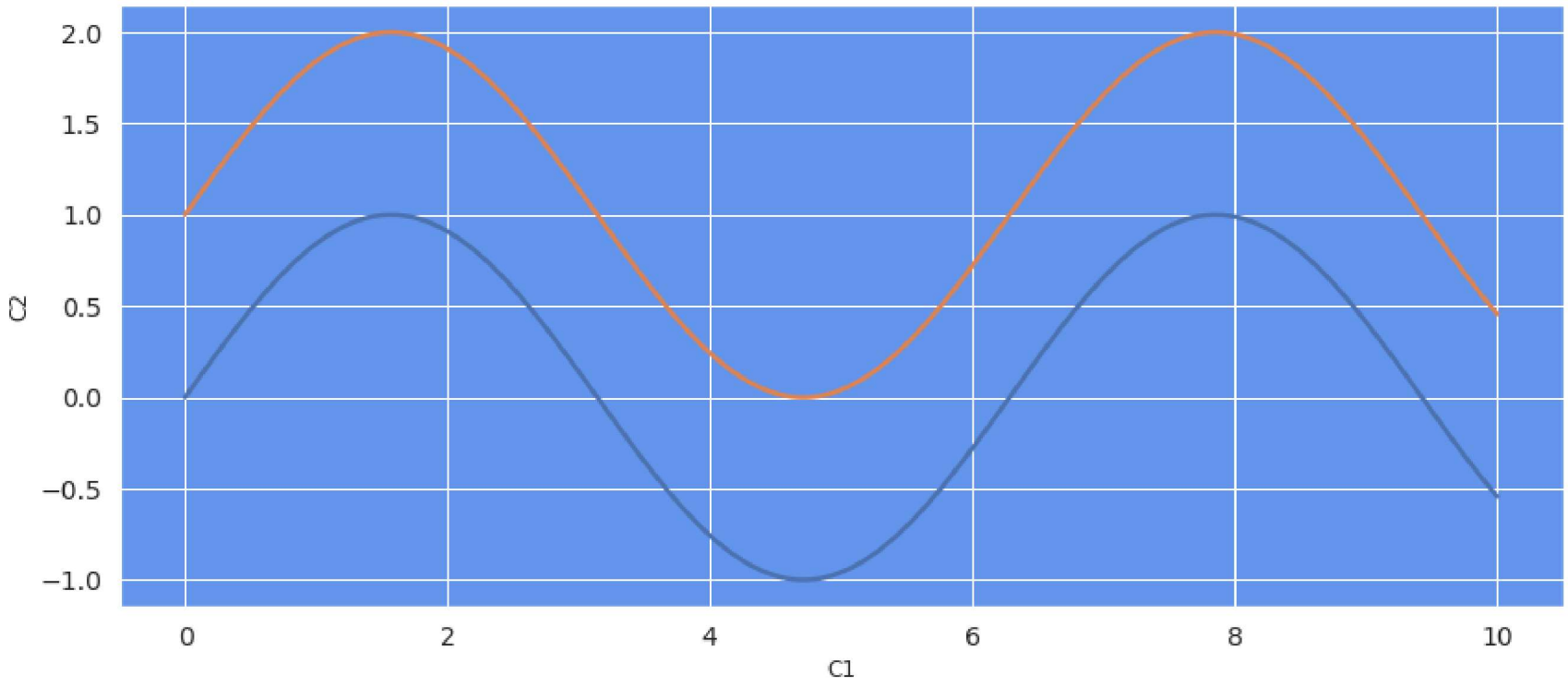
Out[4]:

	C1	C2
0	0.00000	0.000000
1	0.01001	0.010010
2	0.02002	0.020019
3	0.03003	0.030026
4	0.04004	0.040029
5	0.05005	0.050029
6	0.06006	0.060024
7	0.07007	0.070013
8	0.08008	0.079995
9	0.09009	0.089968

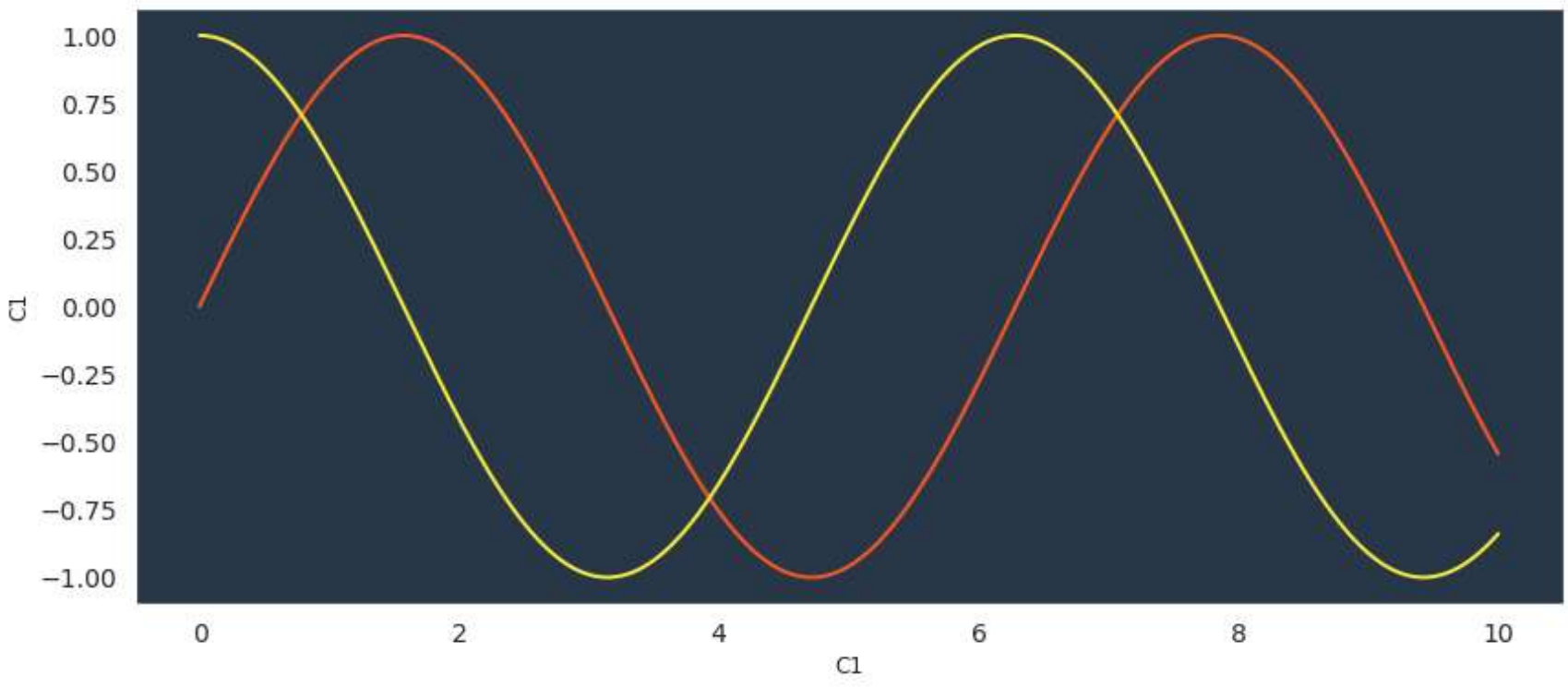
In [5]:
```
# Plotting lineplot using sns.lineplot()
plt.style.use('seaborn-darkgrid')
%matplotlib inline
sns.lineplot(x=df.C1,y=df.C2,data=df)
plt.show()
```



In [6]:
```
""" - Adjusting background color using axes.facecolor
- Changing label size using xtick.labelsize , ytick.labelsize """
plt.figure(figsize=(14,6))
sns.set(rc={'axes.facecolor':'cornflowerblue',"axes.grid":True,'xtick.labelsize':14,'ytick.labelsize':14})
sns.lineplot(x=df.C1,y=df.C2,data=df , linewidth = 2.5)
sns.lineplot(x=df.C1,y=df.C2+1,data=df , linewidth = 2.5)
plt.show()
```

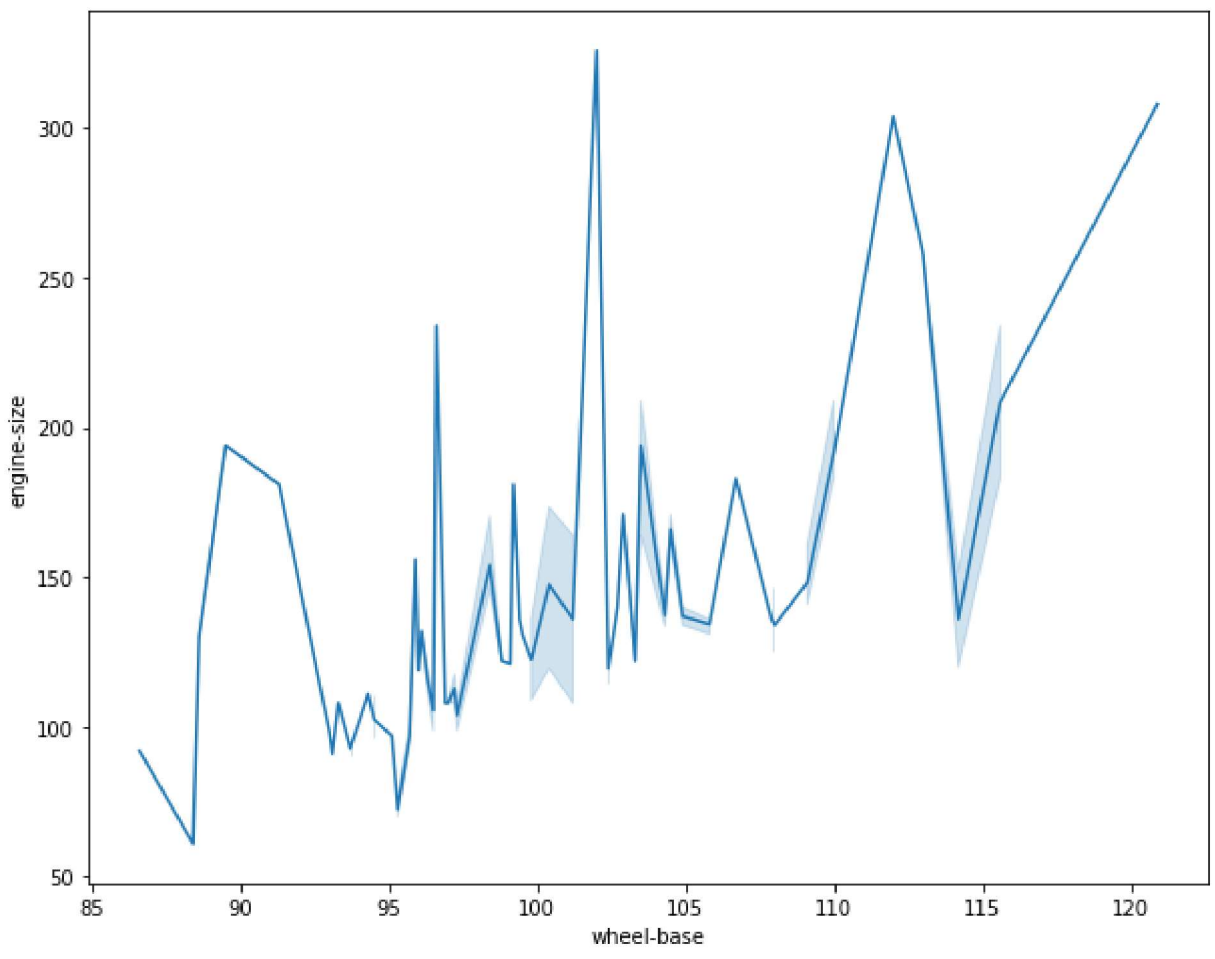


In [7]:
```
""" - Adjusting background color using axes.facecolor
- Changing label size using xtick.labelsize , ytick.labelsize """
plt.figure(figsize=(14,6))
sns.set(rc={"axes.facecolor":"#283747", "axes.grid":False,'xtick.labelsize':14,'ytick.labelsize':14})
sns.lineplot(x=df.C1,y=df.C2,data=df , color = "#FF5722" , linewidth = 2 )
sns.lineplot(x=df.C1,y=np.cos(df.C1),data=df , color = "#FFEB3B" , linewidth = 2)
plt.show()
```

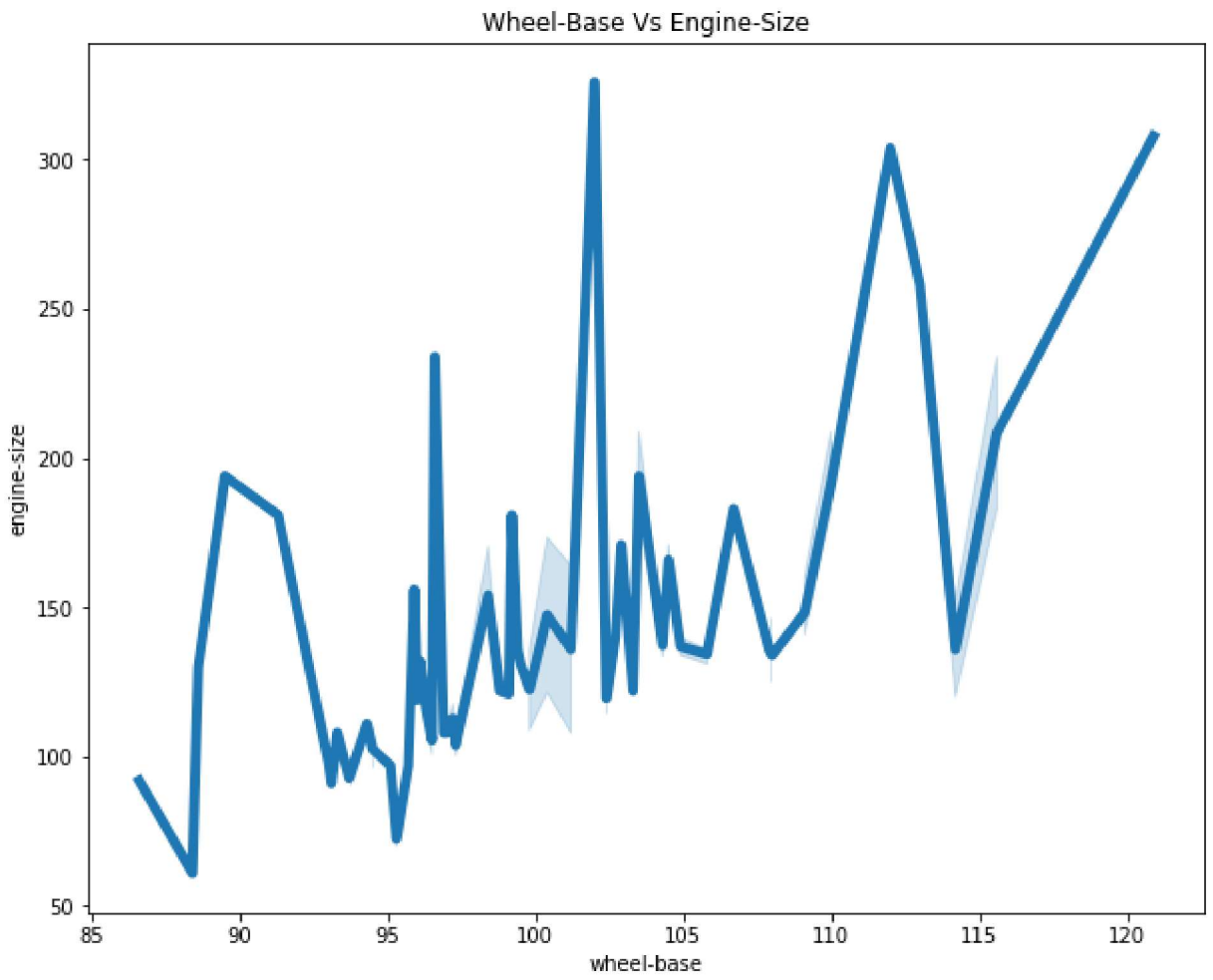


In [8]:
```
#Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

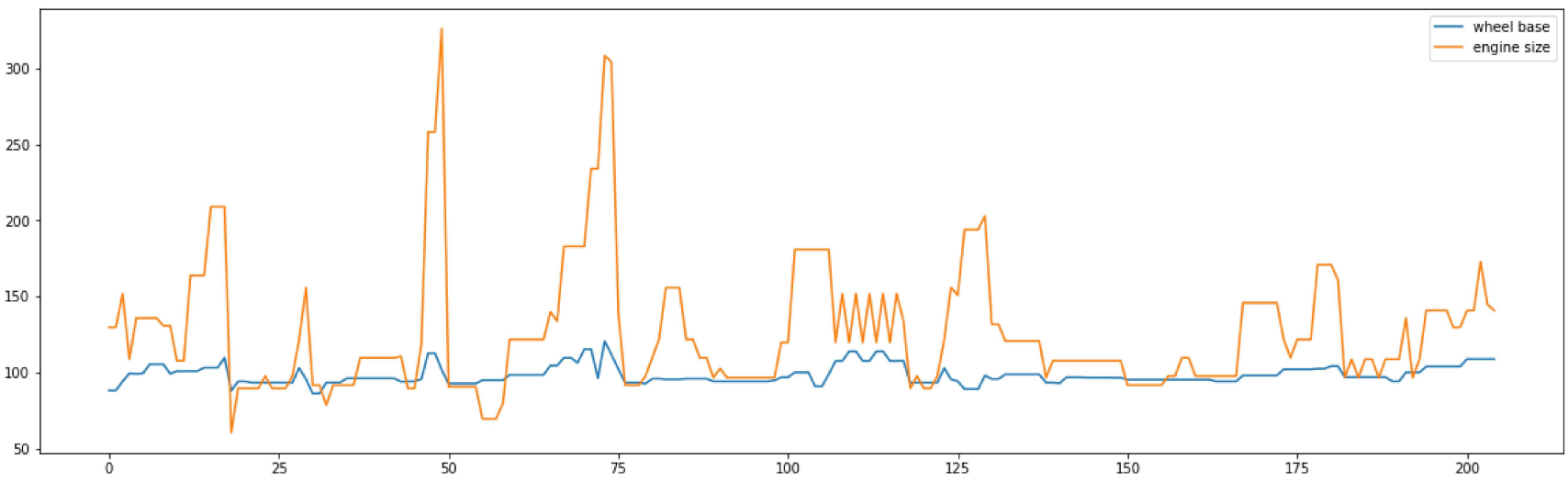
In [9]:
```
plt.figure(figsize=(10,8))
sns.lineplot(x="wheel-base",y="engine-size",data=auto)
plt.show()
```



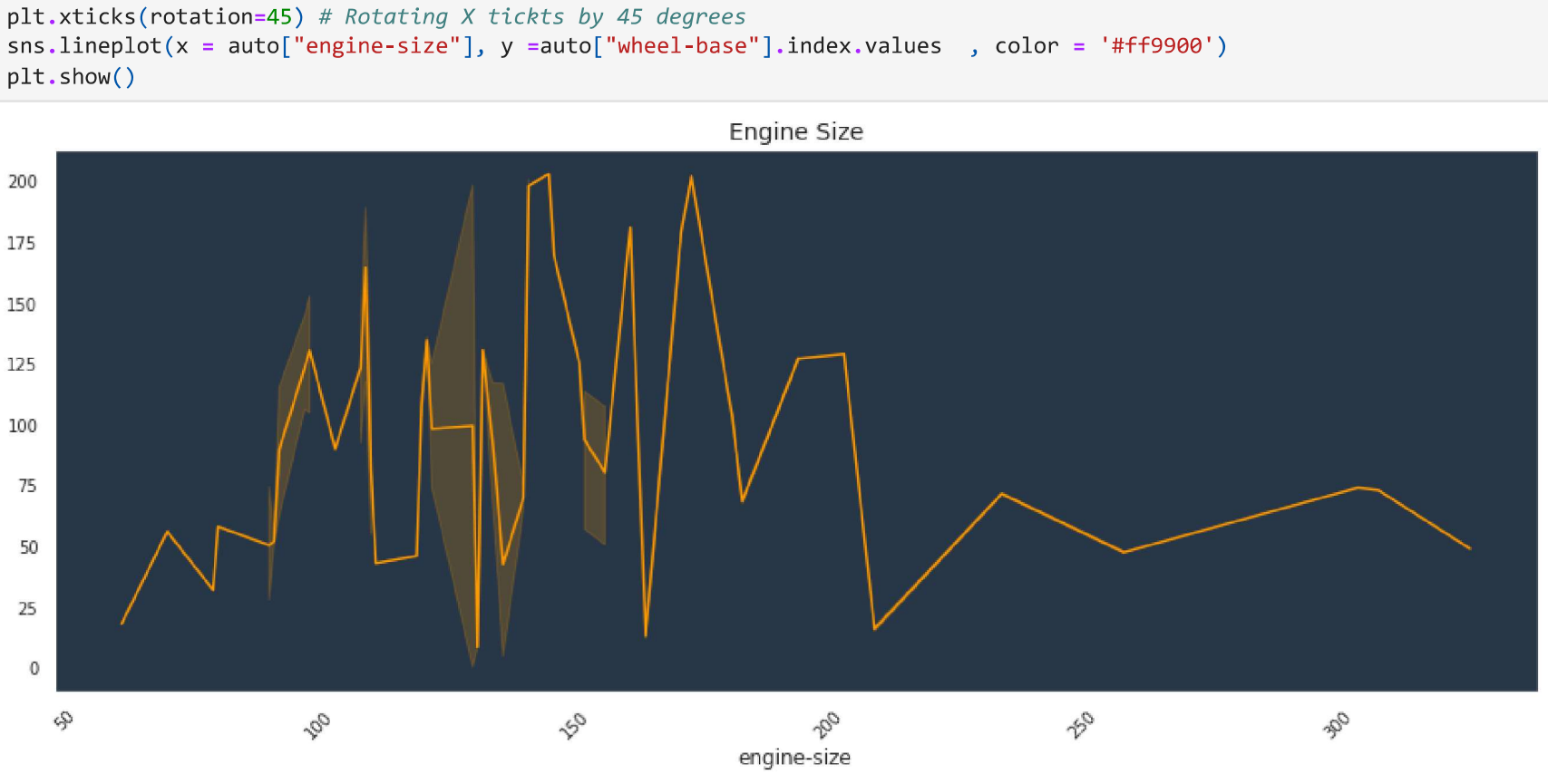
In [10]:
```
plt.figure(figsize=(10,8))
sns.lineplot(x="wheel-base",y="engine-size",data=auto,linewidth = 5)
plt.title("Wheel-Base Vs Engine-Size")
plt.show()
```



In [11]:
```
plt.figure(figsize=(20,6))
sns.lineplot(data=auto['wheel-base'],linewidth = 1.5 , label = 'wheel base')
sns.lineplot(data=auto['engine-size'],linewidth = 1.5 , label = 'engine size')
plt.show()
```

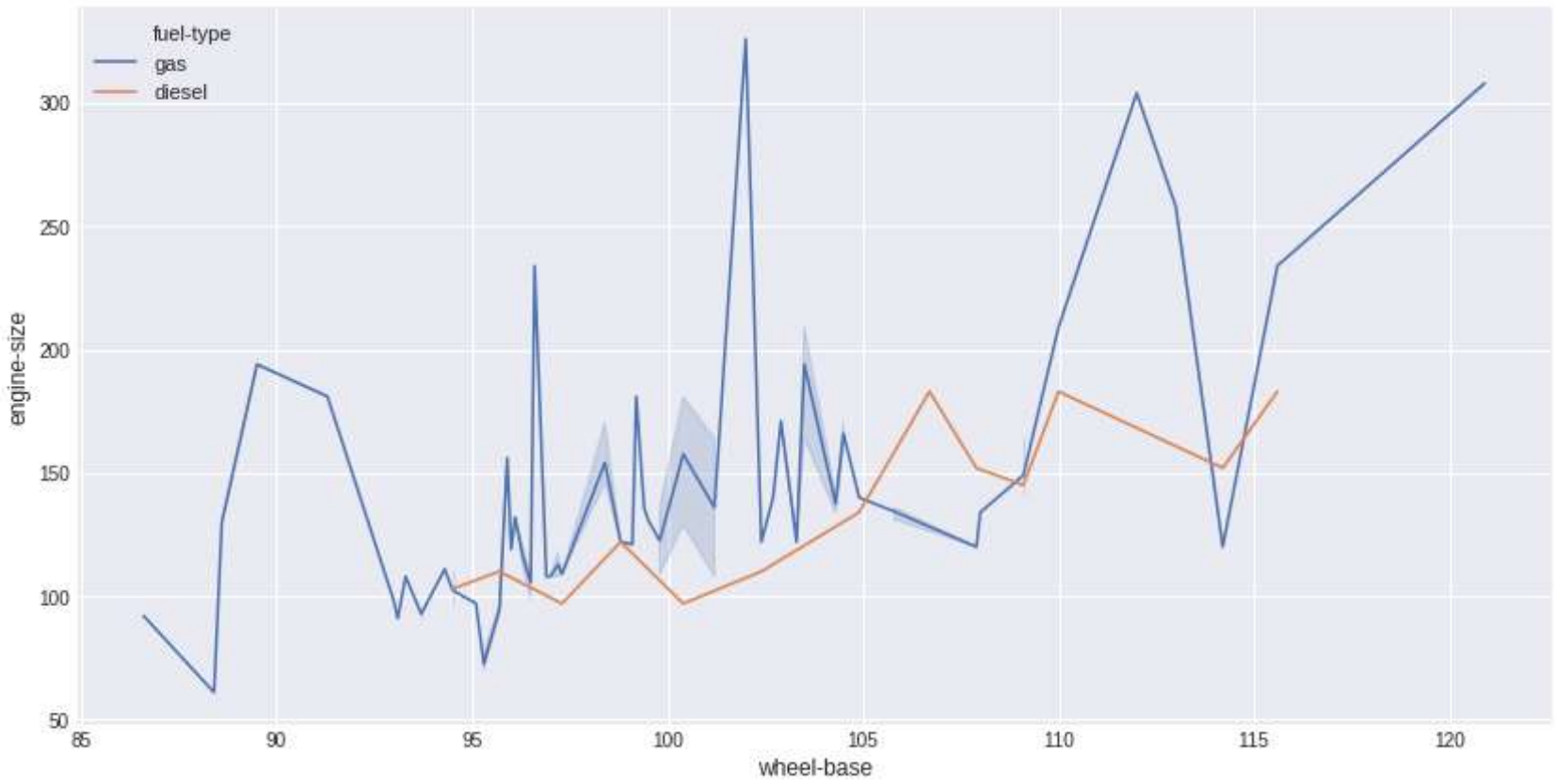


In [12]:
```
plt.figure(figsize=(16,6))
sns.set(rc={"axes.facecolor":"#283747", "axes.grid":False,'xtick.labelsize':10,'ytick.labelsize':10})
plt.title("Engine Size",fontsize = 14)
plt.xticks(rotation=45) # Rotating X tickts by 45 degrees
sns.lineplot(x = auto["engine-size"], y =auto["wheel-base"].index.values  , color = '#ff9900')
plt.show()
```

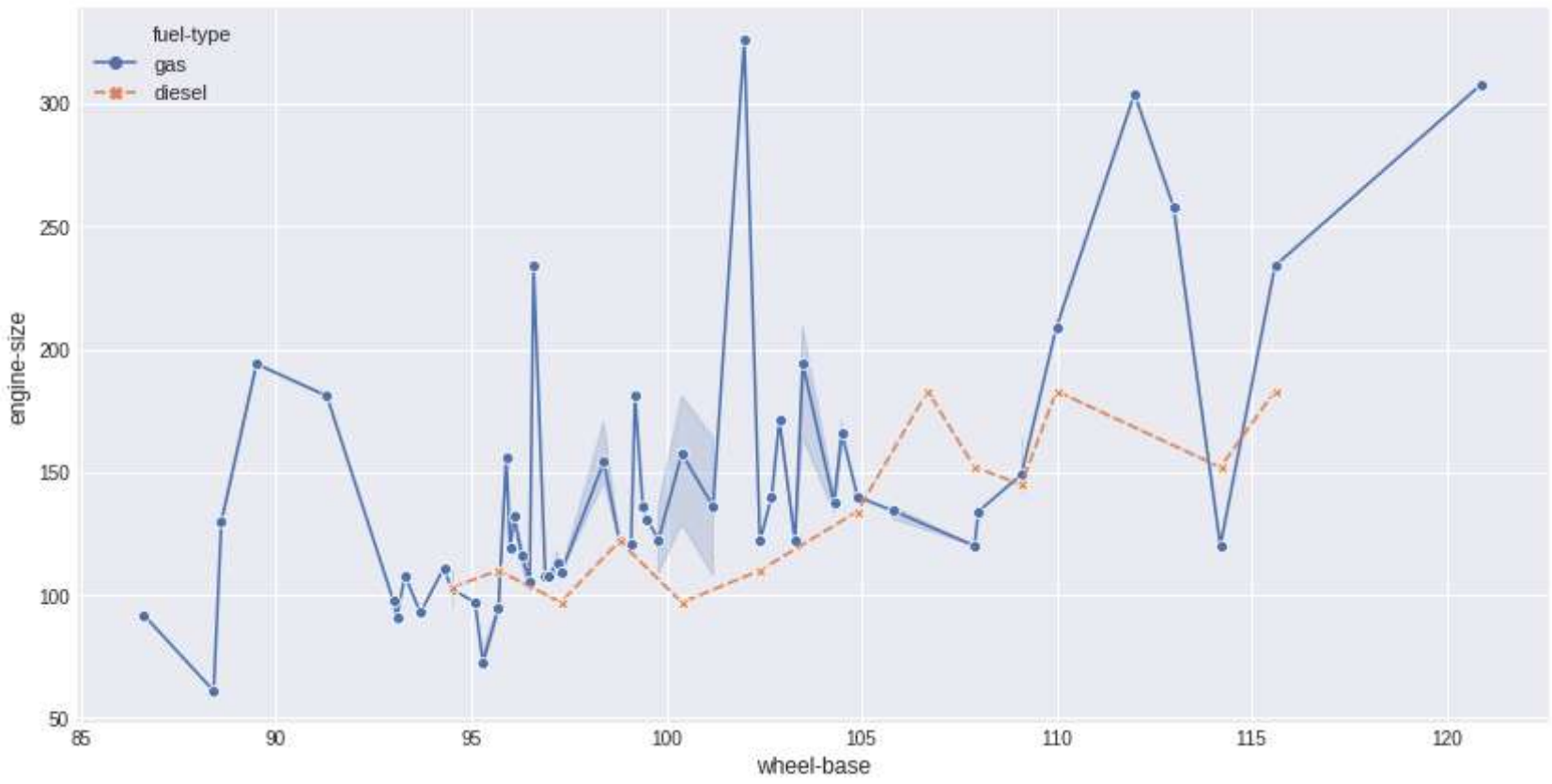


In [13]:
```
# Show groups with different colors using "hue"
plt.figure(figsize=(14,7))
plt.style.use('seaborn-darkgrid')
# Group variable using "hue" that will produce lines with different colors
```

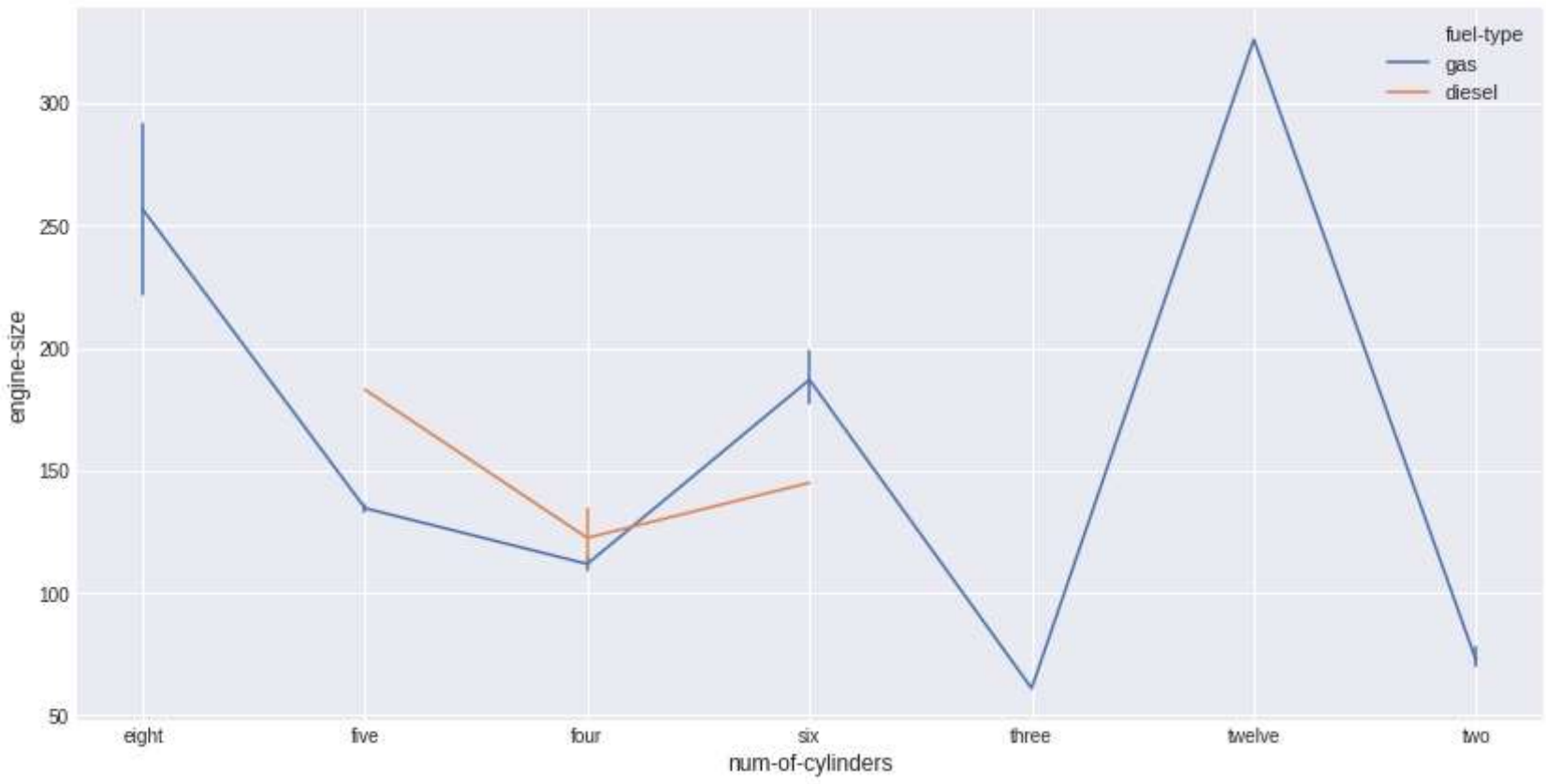
```
sns.lineplot(x="wheel-base" , y="engine-size" , hue="fuel-type" , data=auto)
plt.show()
```



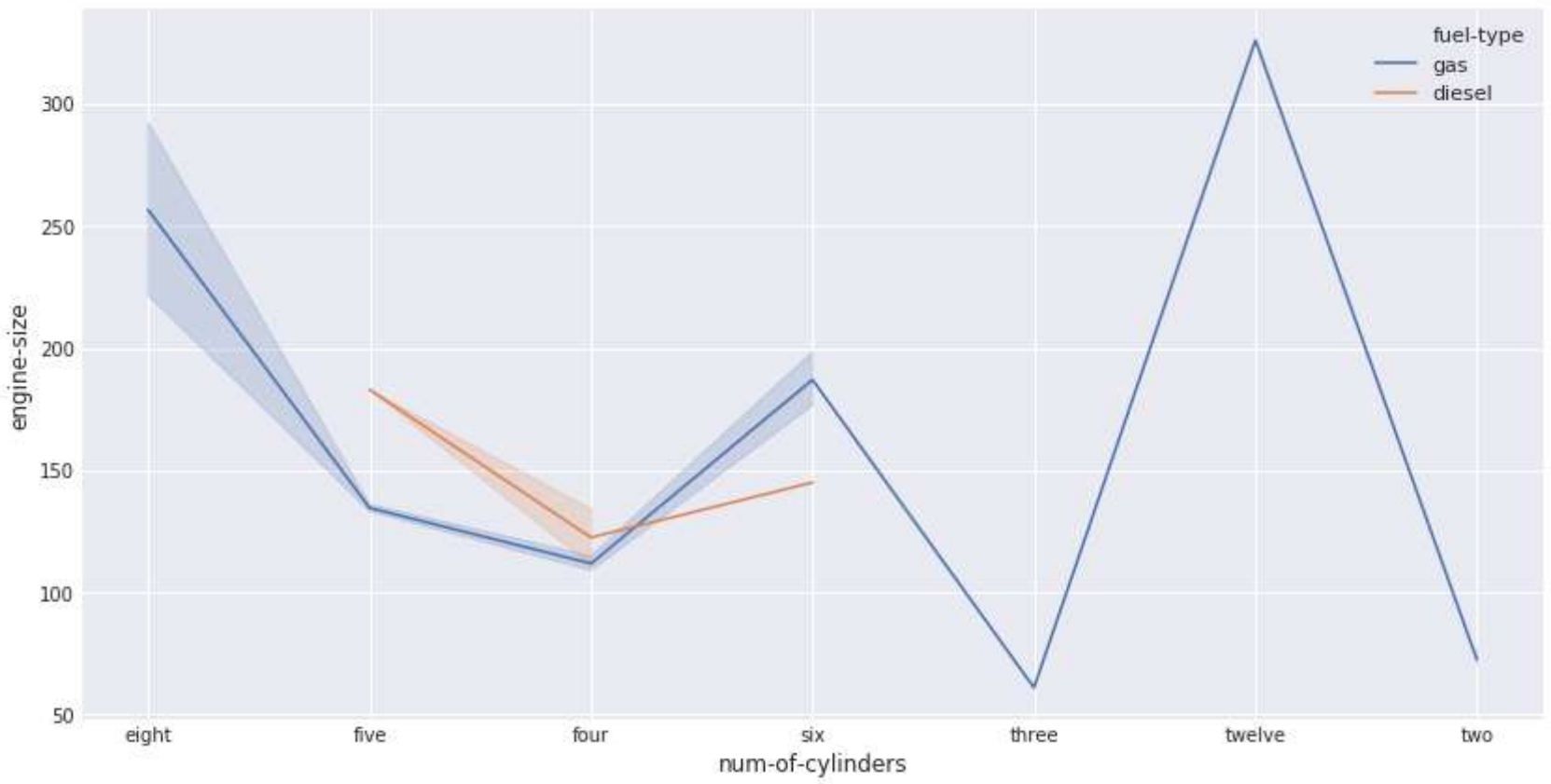
In [14]:
```
# Using markers to identify groups
plt.figure(figsize=(14,7))
plt.style.use('seaborn-darkgrid')
sns.lineplot(x="wheel-base" , y="engine-size" , hue = "fuel-type" ,style="fuel-type" , markers=True ,data=auto)
plt.show()
```



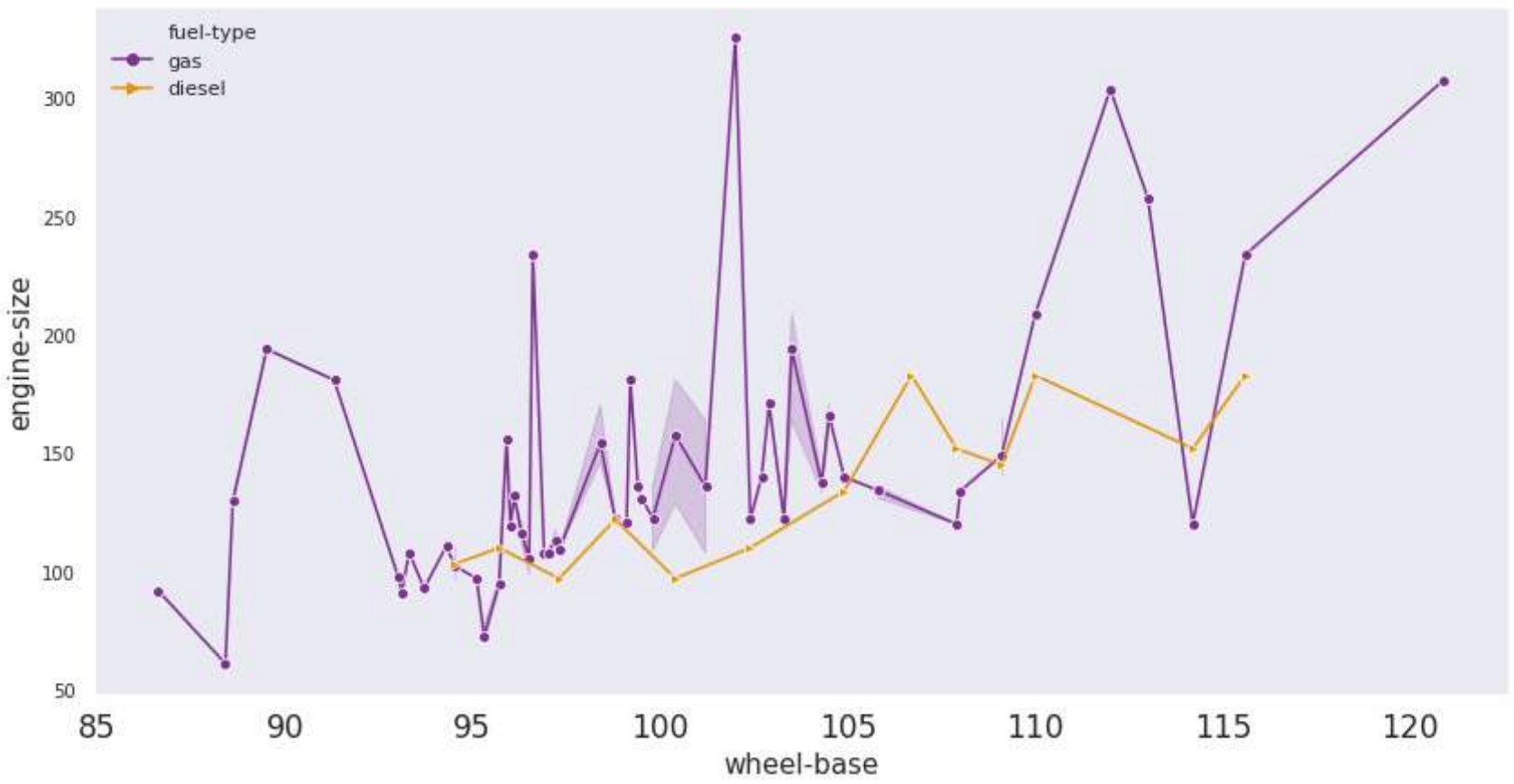
In [15]:
```
plt.figure(figsize=(14,7))
plt.style.use('seaborn-darkgrid')
sns.lineplot(x="num-of-cylinders" , y="engine-size" , hue = "fuel-type" , err_style="bars",data=auto)
plt.show()
```



In [16]:
```
plt.figure(figsize=(14,7))
sns.lineplot(x = "num-of-cylinders", y = "engine-size",data=auto,hue="fuel-type")
sns.set(style='dark',)
plt.show()
```

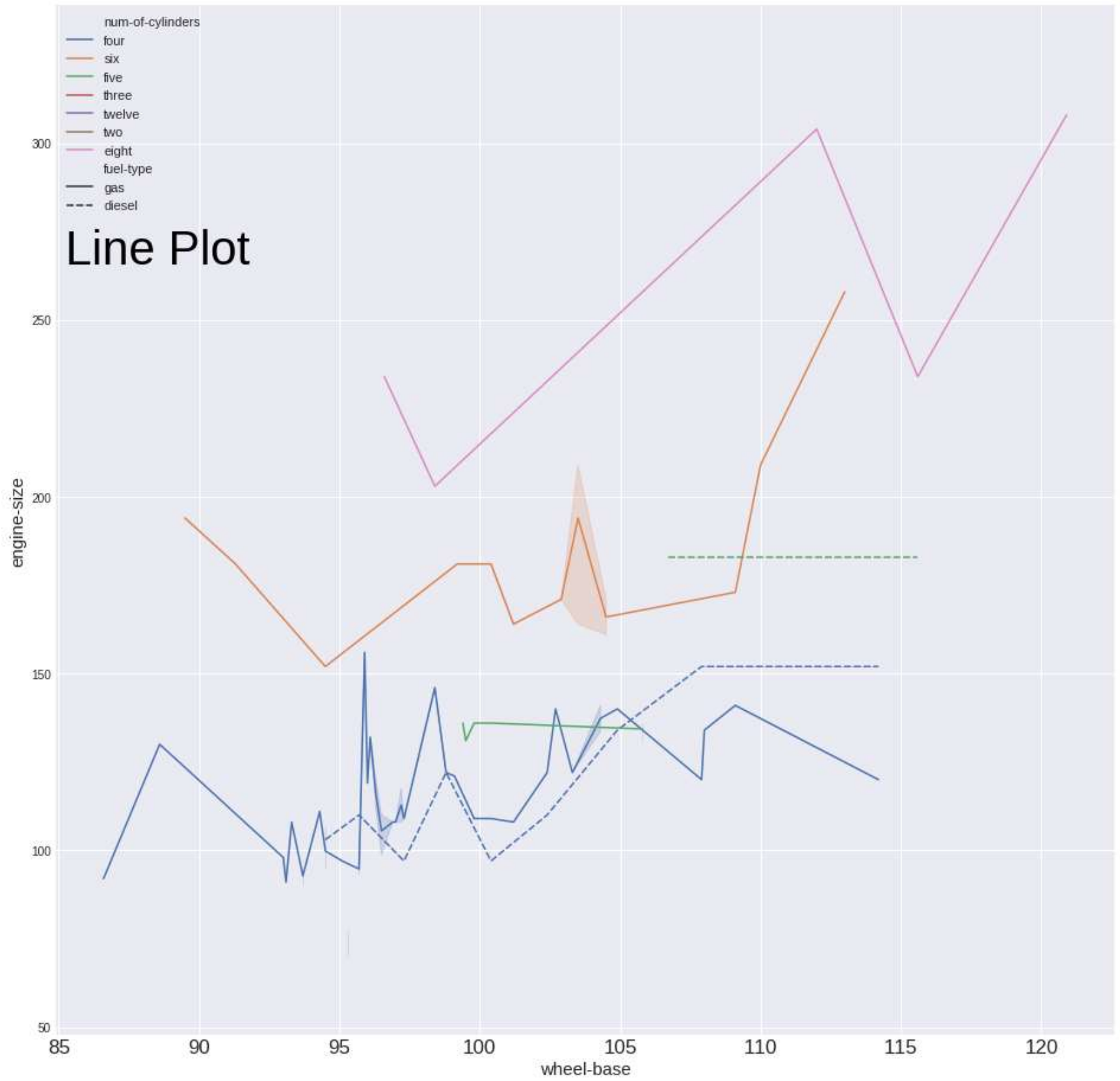


In [17]:
```
plt.figure(figsize=(14,7))
sns.set(rc={'xtick.labelsize':17,'ytick.labelsize':10,'axes.labelsize':15 , "axes.grid":False})
# Use "Pallete" to specify the colors to be used for different levels of the hue
sns.lineplot(x="wheel-base" , y="engine-size" , hue = "fuel-type" ,style="fuel-type" , dashes = False,palette = 'CMRmap
plt.show()
```

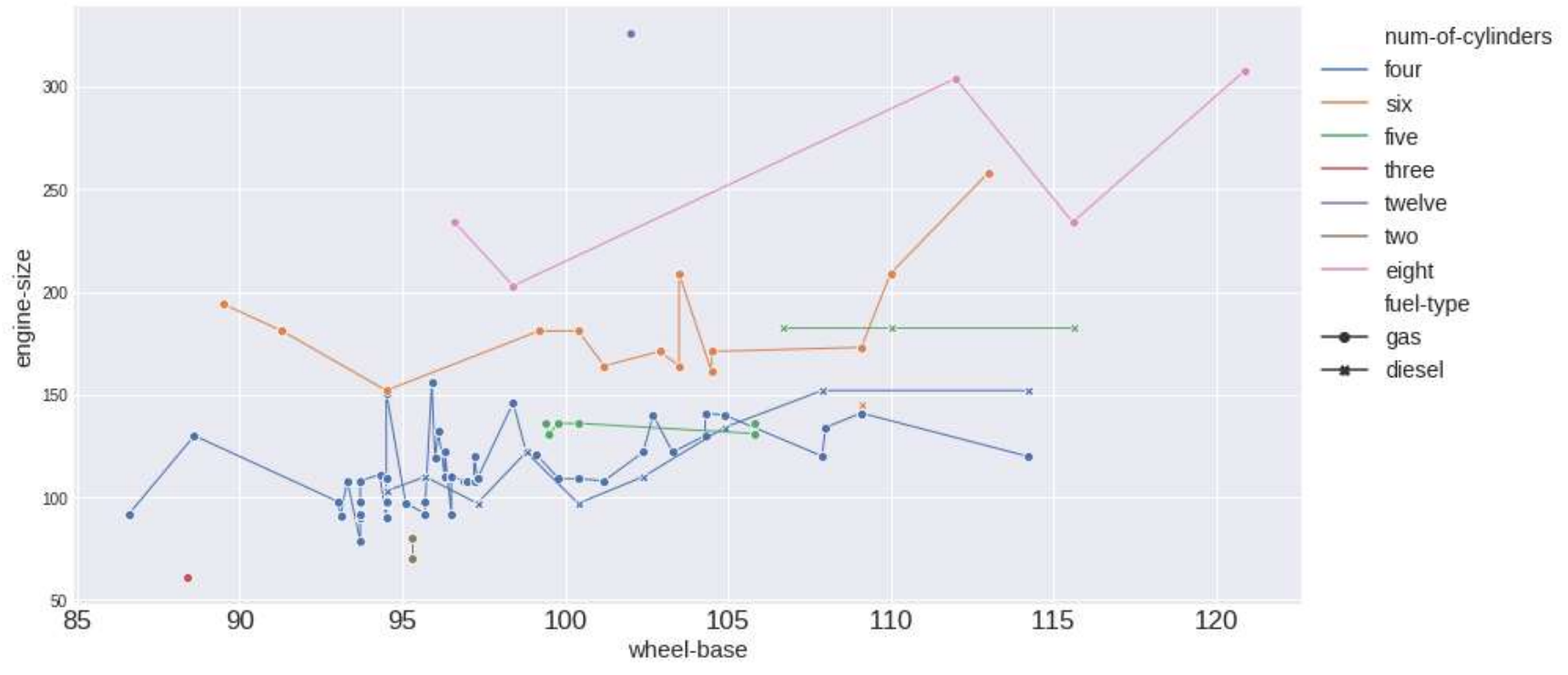


In [18]:
```
# Color and line dashing to represent 2 different grouping variables using "hue" & "style"
plt.figure(figsize=(16,16))
plt.style.use('seaborn-darkgrid')
plt.gcf().text(.2, .70, "Line Plot", fontsize = 40, color='Black' ,ha='center', va='center')
sns.lineplot(x="wheel-base" , y="engine-size" , hue = "num-of-cylinders" ,style="fuel-type" ,data=auto)
```

Out[18]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7bf6335afb70>
```



In [19]:
```
#Showing all experiments instead of Aggregate using "units" and "estimator"
plt.figure(figsize=(14,7))
plt.style.use('seaborn-darkgrid')
sns.lineplot(x="wheel-base" , y="engine-size" , hue = "num-of-cylinders" ,style="fuel-type", units='num-of-cylinders',m
plt.legend(bbox_to_anchor=(1.0, 1.0) , shadow=True, fontsize='large')
plt.show()
```



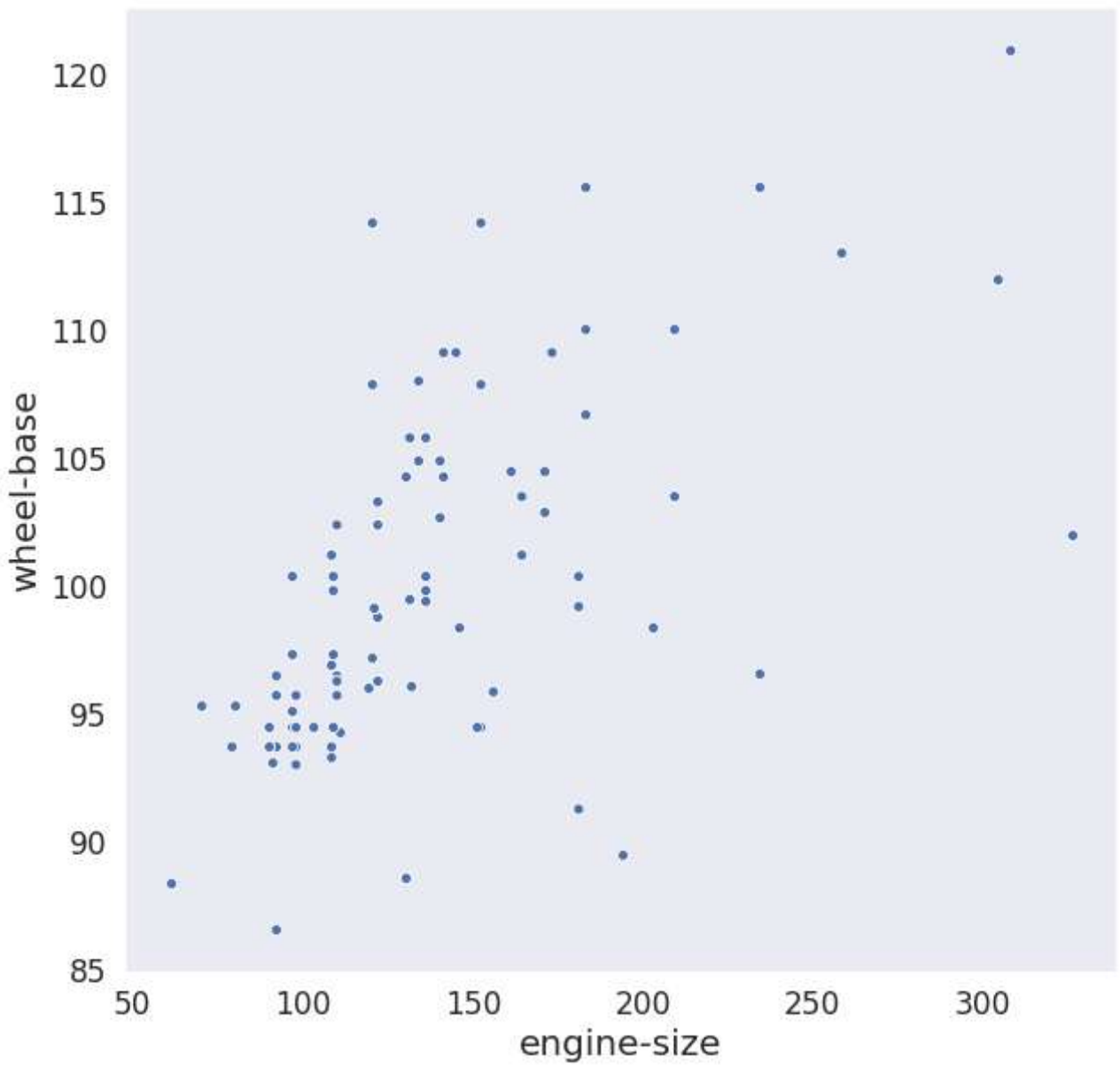
In [20]:
```
# Combining lineplots using relplot

plt.figure(figsize=(10,10))
sns.set(rc={'xtick.labelsize':17,'ytick.labelsize':17,'axes.labelsize':20 , "axes.grid":False})
sns.relplot(x="wheel-base" , y="engine-size" , hue="num-of-cylinders" , col="fuel-type",kind='line', height=8.5,data=au
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```

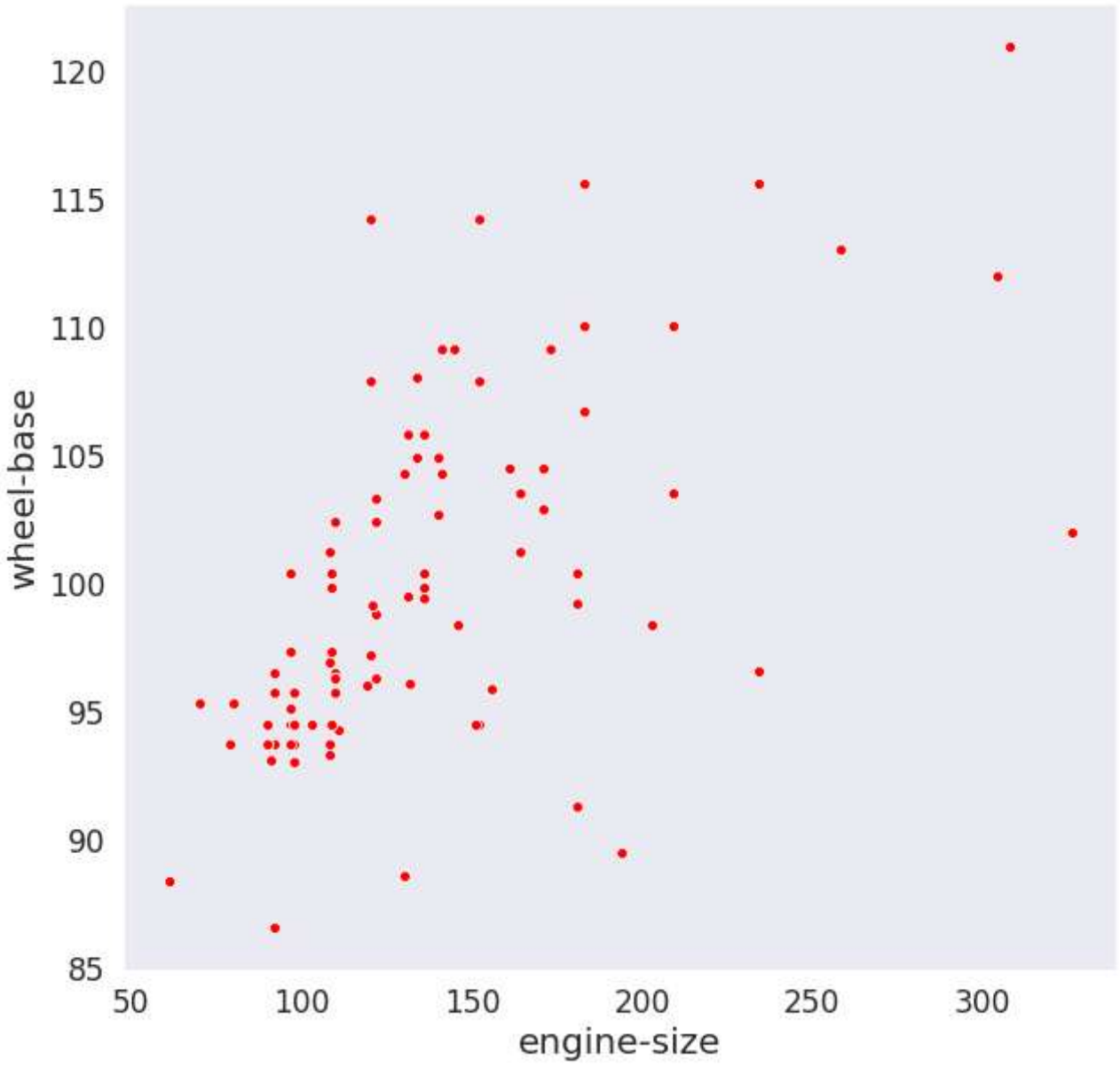
5.Scatterplots

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:

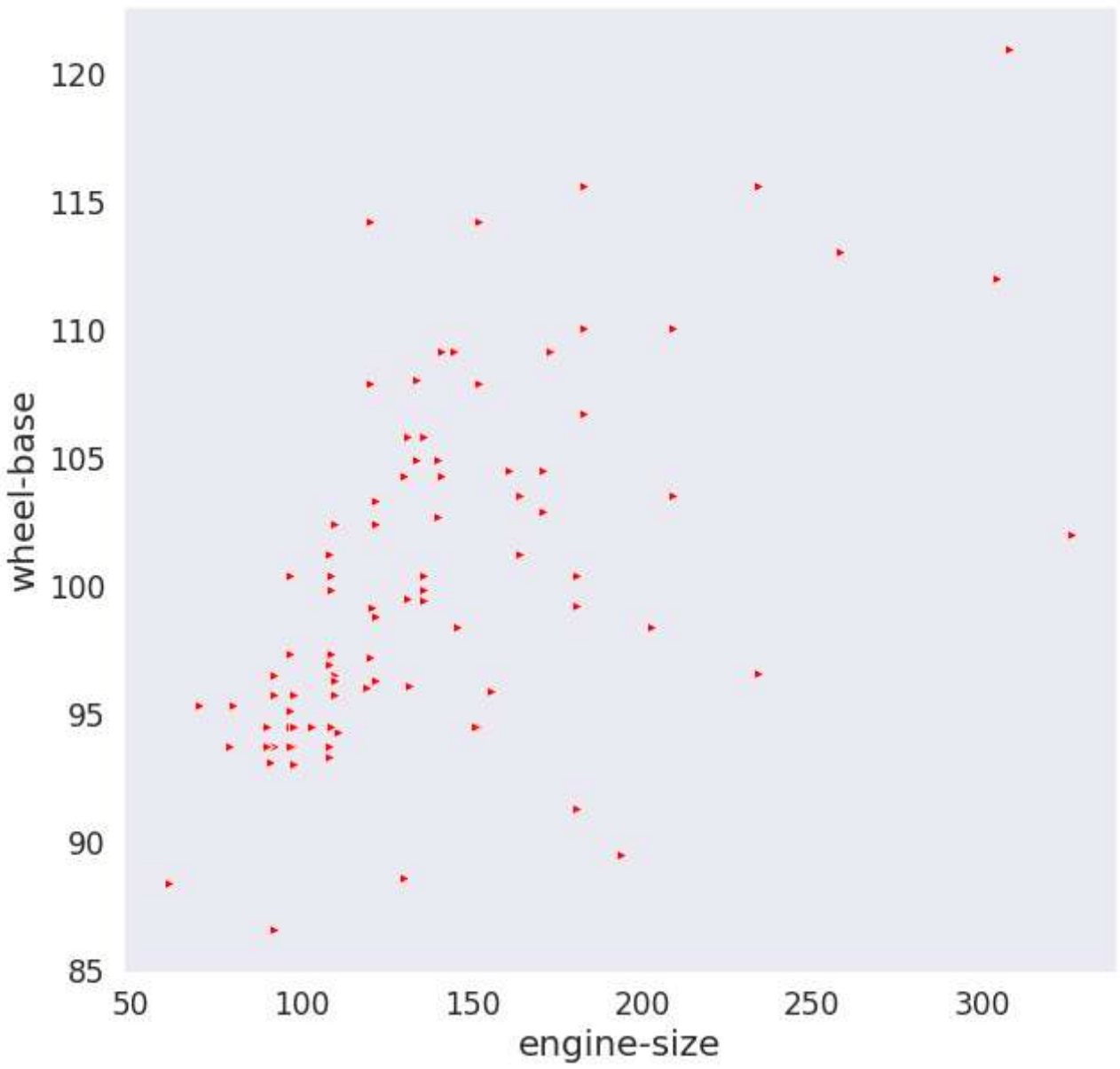
In [21]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',data=auto)
plt.show()
```



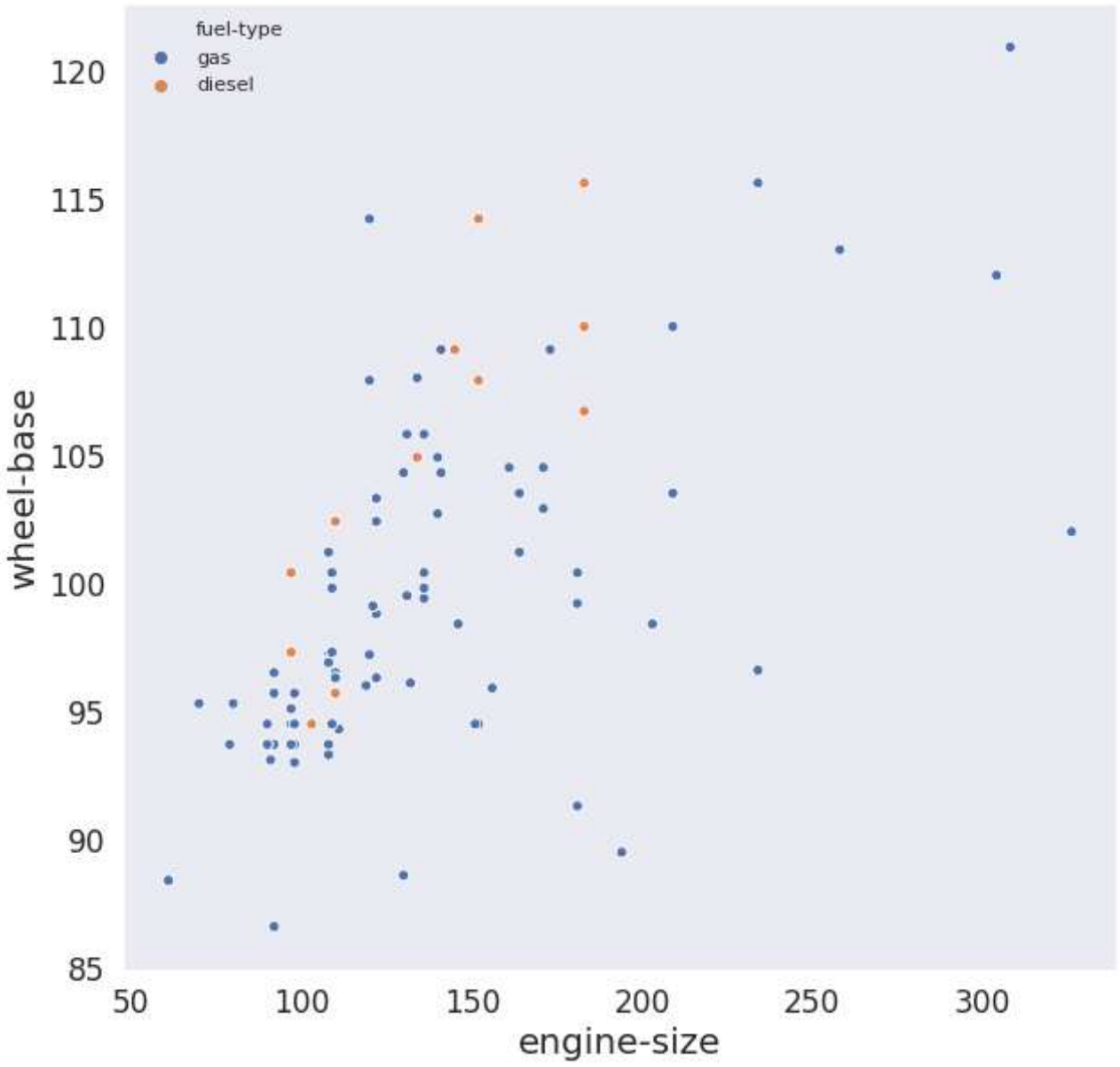
In [22]:
```
#Changing the color of data points using "color" parameter

plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',data=auto,color='#FF0000')
plt.show()
```



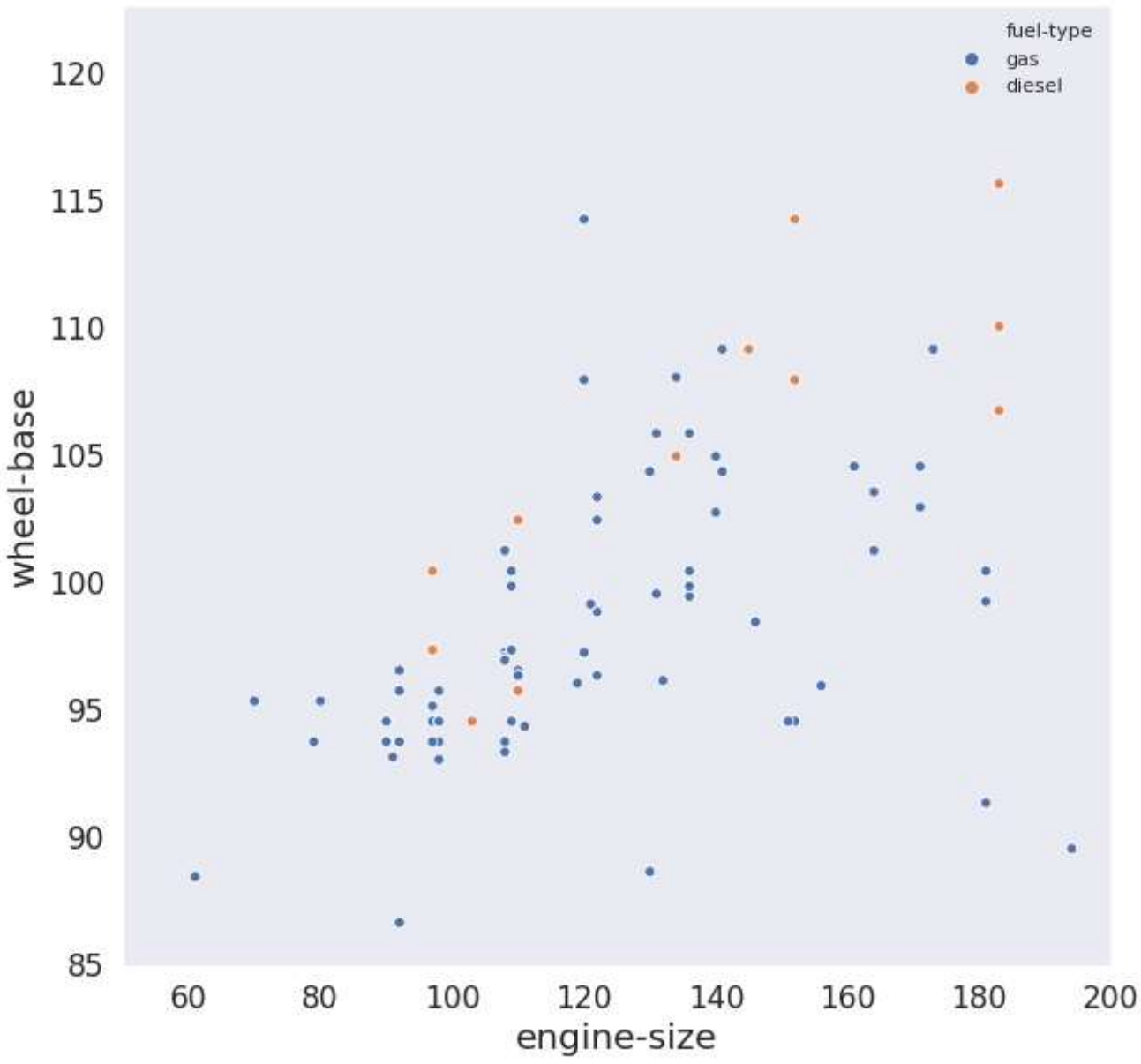
In [23]:
```
#Changing the shape of data points using "marker" parameter
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',data=auto,color='#FF0000',marker='>')
plt.show()
```



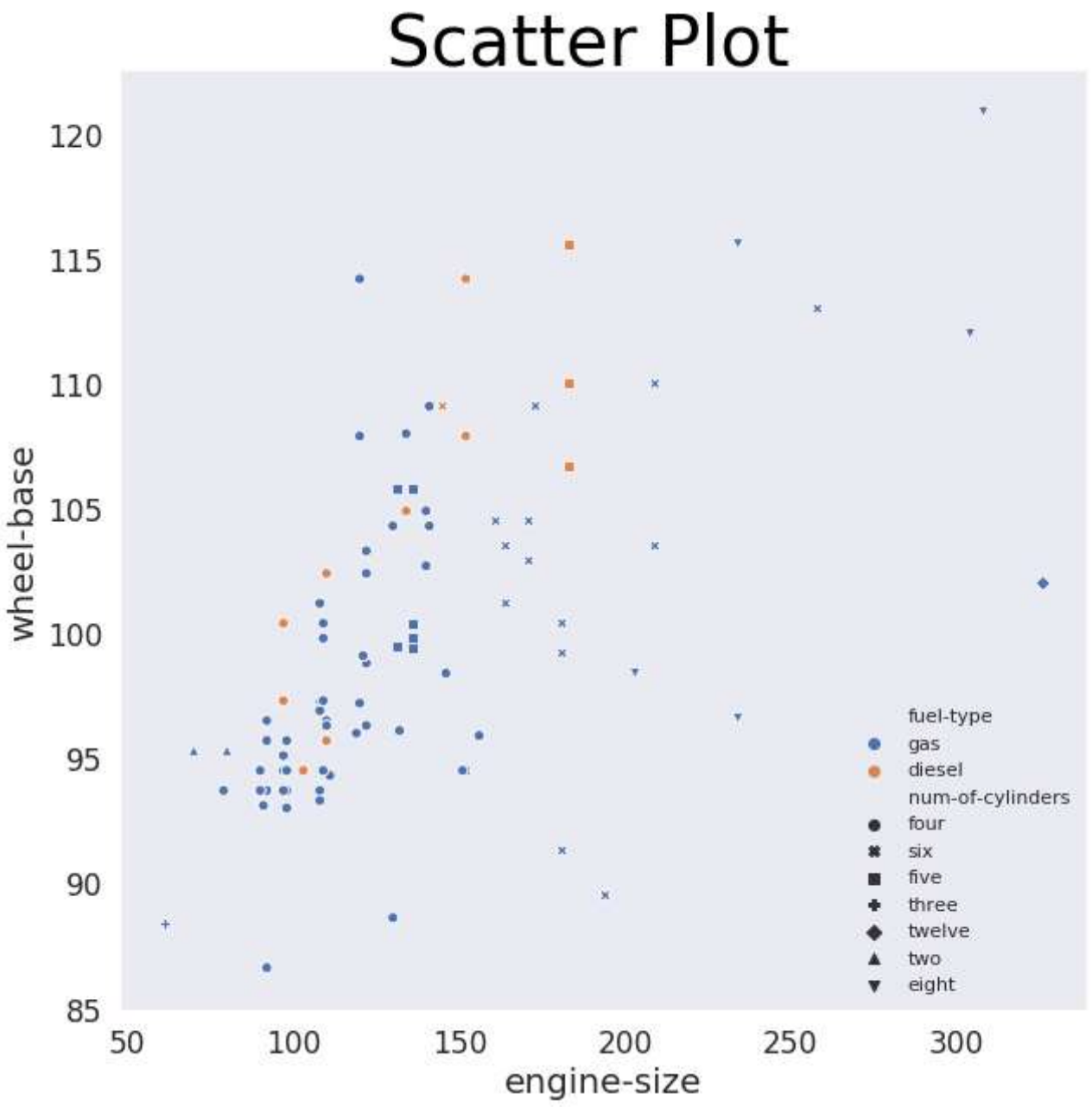
In [24]:
```
# Show groups with different colors using "hue"

plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',data=auto)
plt.show()
```



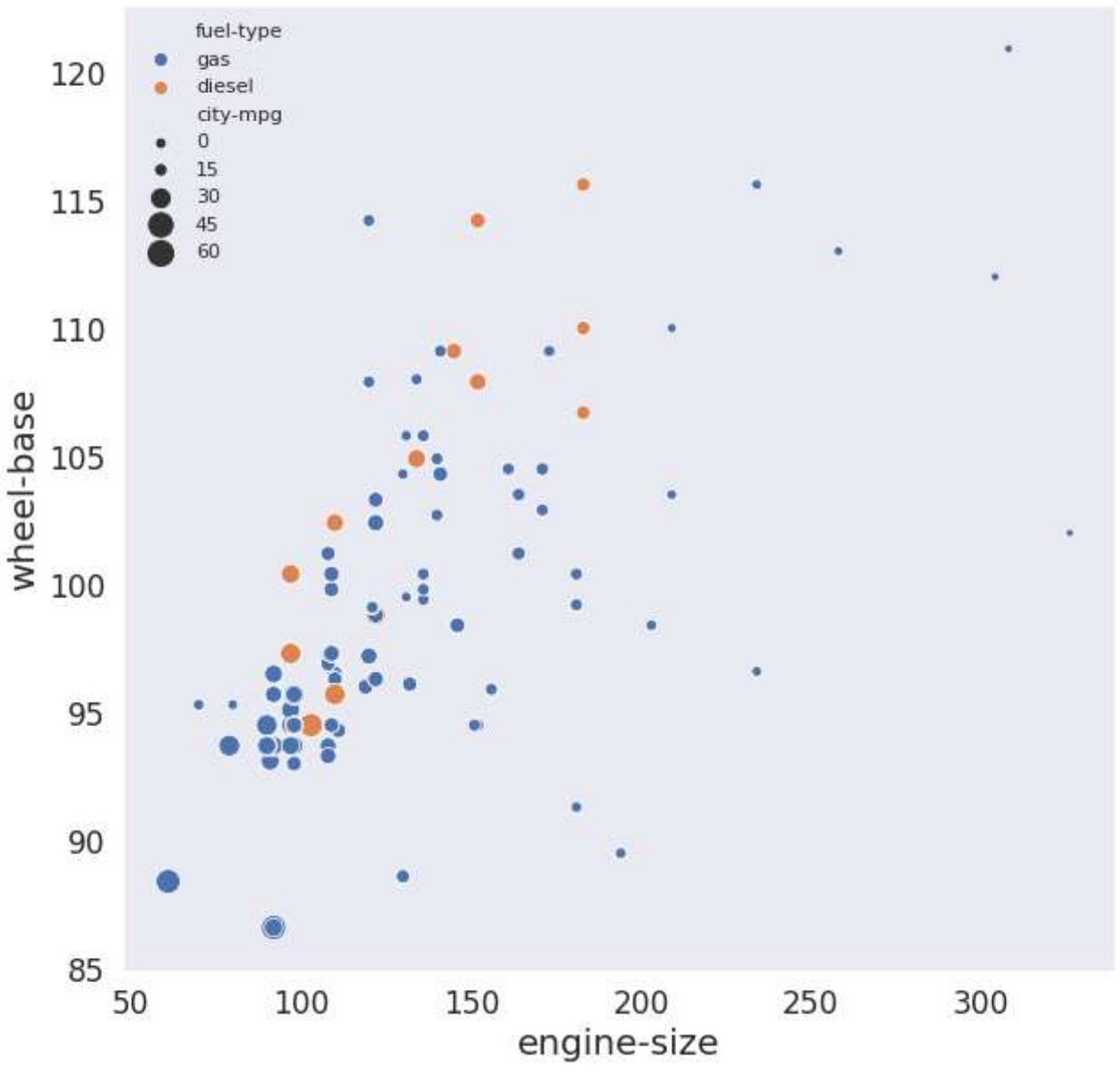
In [25]:
```
#Setting X limit using "plt.xlim"
plt.figure(figsize=(10,10))
plt.xlim([50,200])
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',data=auto)
plt.show()
```



In [26]:
```
# Showing two different grouping variables using "hue" amd "style" parameter

plt.figure(figsize=(10,10))
plt.gcf().text(.5, .9, "Scatter Plot", fontsize = 40, color='Black' ,ha='center', va='center')
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',style='num-of-cylinders',data=auto)
plt.show()
```

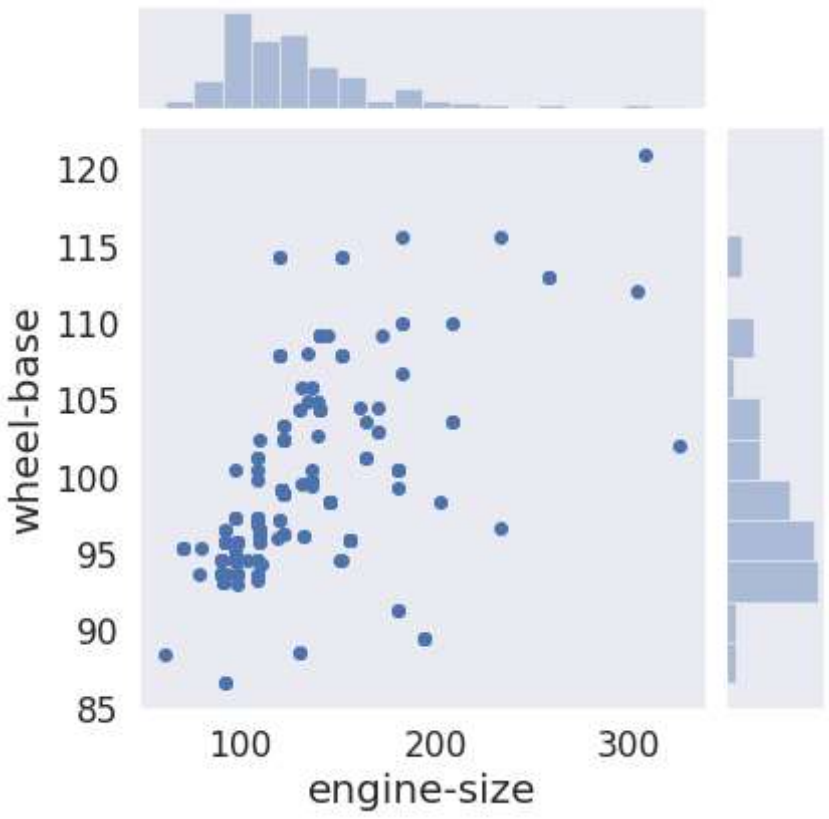


In [27]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',size='city-mpg',sizes=(20,200),data=auto)
plt.show()
```



In [28]:
```
# Using the jointplot
sns.jointplot(auto['engine-size'],auto['wheel-base'])
```

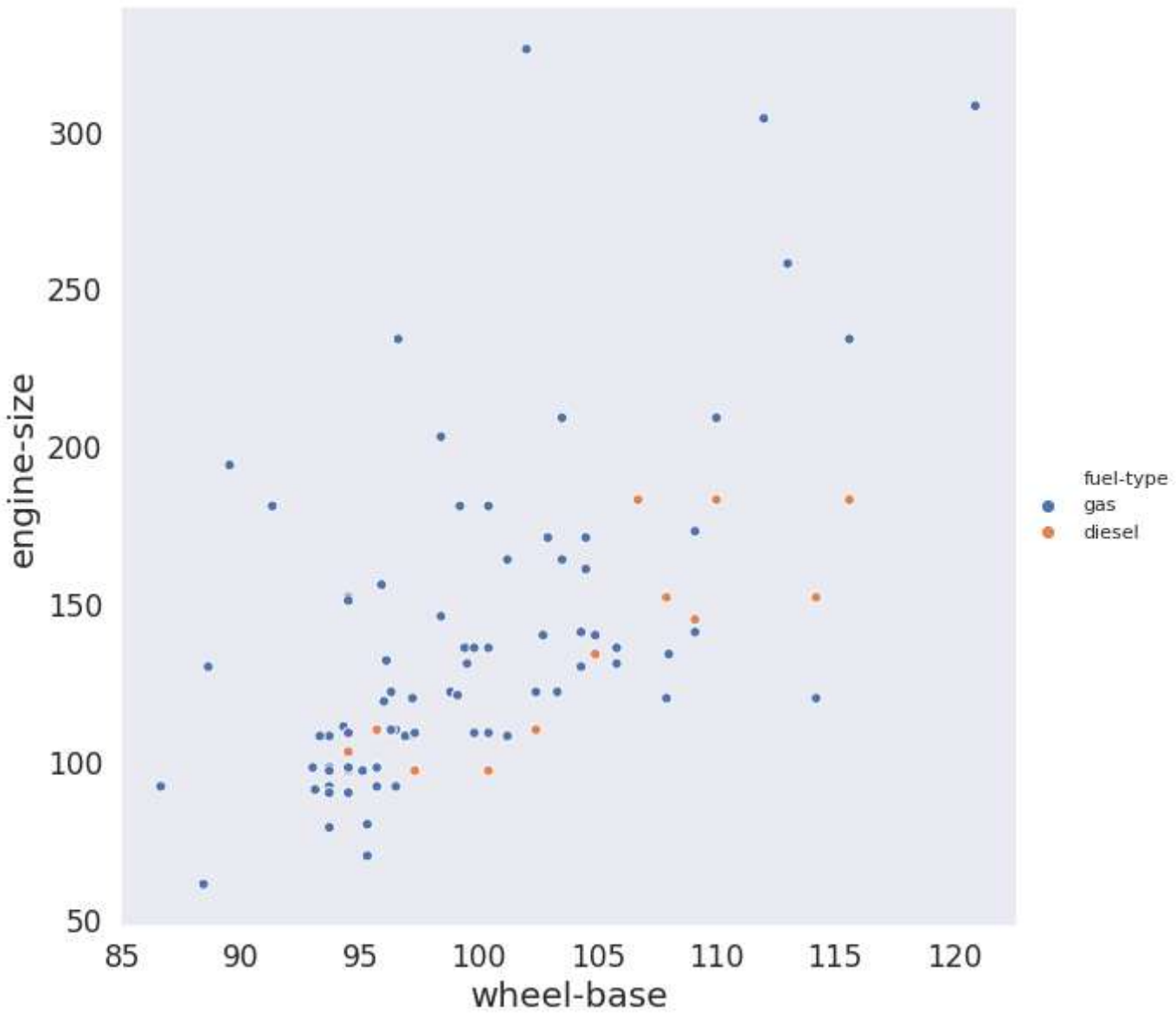
Out[28]:
```
<seaborn.axisgrid.JointGrid at 0x7bf635991b70>
```



6. Rel Plot

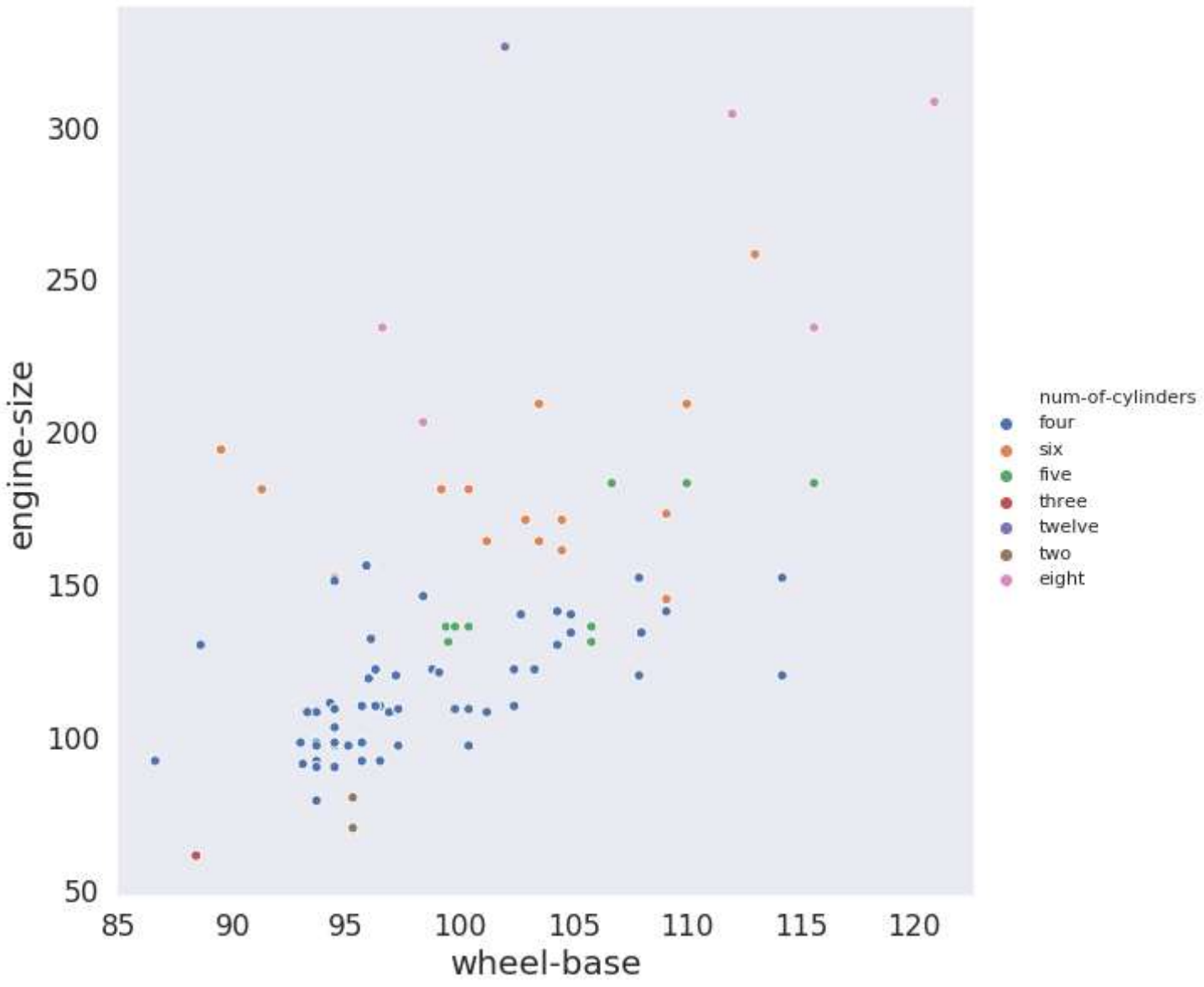
In [29]:
```
# Drawing scatter plot using relplot
plt.figure(figsize=(14,16))
sns.relplot(x="wheel-base" , y="engine-size" , hue="fuel-type" ,kind='scatter', height=8.5,aspect =1,data=auto)
plt.show()
```

```
<Figure size 1008x1152 with 0 Axes>
```



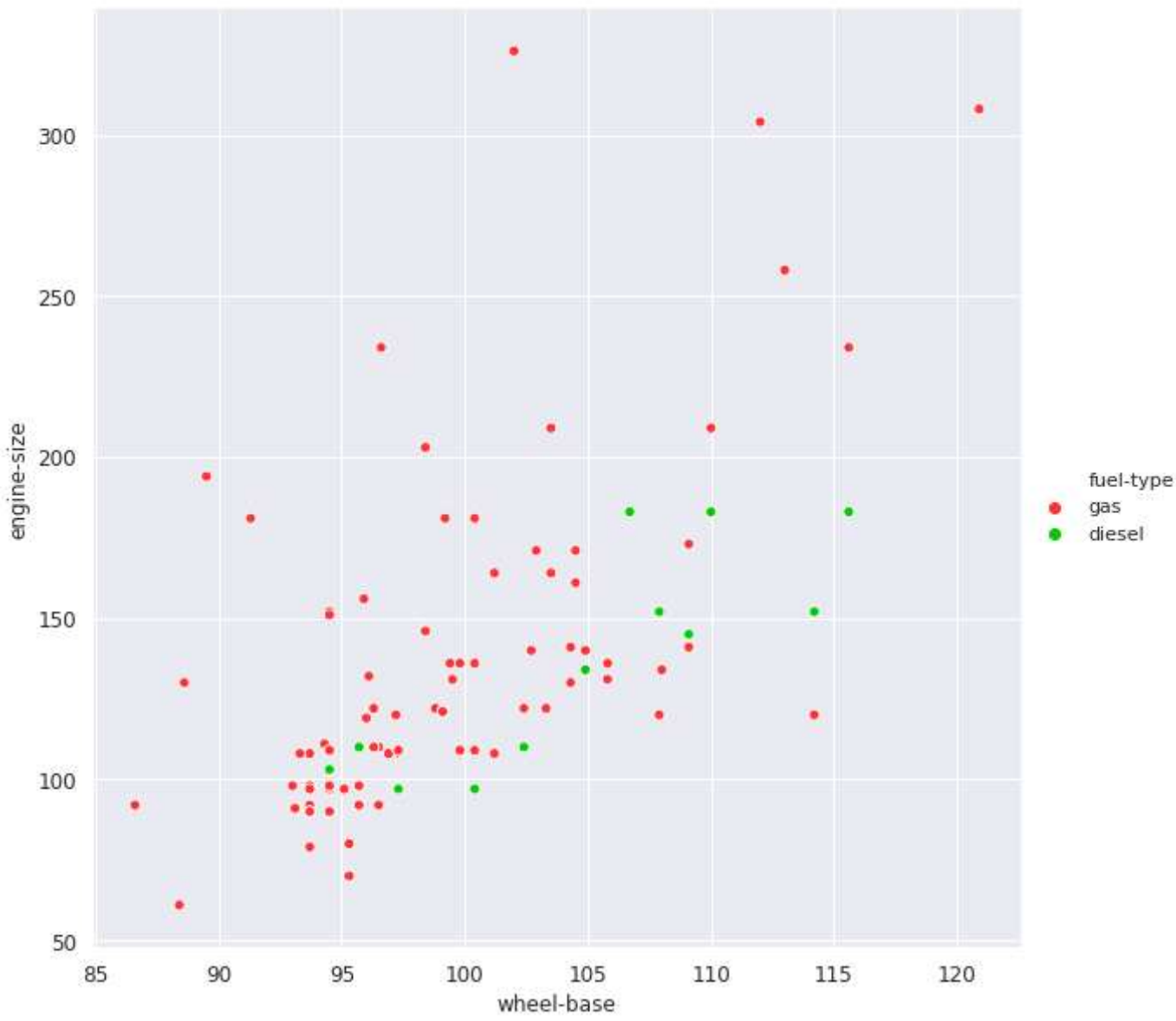
In [30]:
```
# Drawing scatter plot using relplot
plt.figure(figsize=(14,16))
sns.relplot(x="wheel-base" , y="engine-size" , hue="num-of-cylinders" ,kind='scatter', height=8.5,aspect =1,data=auto)
plt.show()
```

```
<Figure size 1008x1152 with 0 Axes>
```



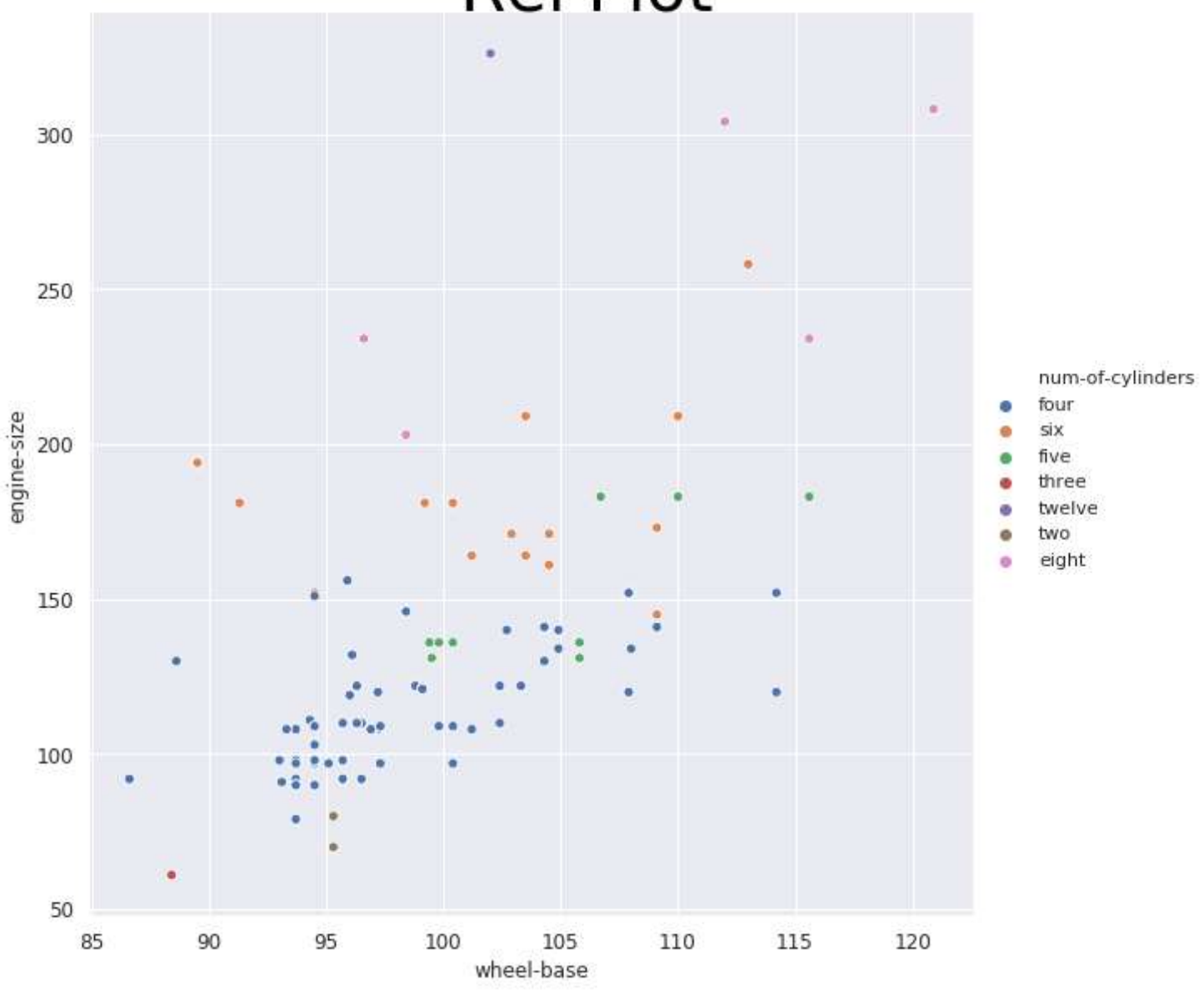
In [31]:
```
# Use "Pallete" to specify the colors to be used for different levels of the hue

plt.figure(figsize=(14,16))
sns.set(rc={'xtick.labelsize':12,'ytick.labelsize':12,'axes.labelsize':12})
sns.relplot(x="wheel-base" , y="engine-size" , hue="fuel-type" ,kind='scatter',palette=["#FF3333" ,"#00CC00"], height=8
plt.show()
```

```
<Figure size 1008x1152 with 0 Axes>
```



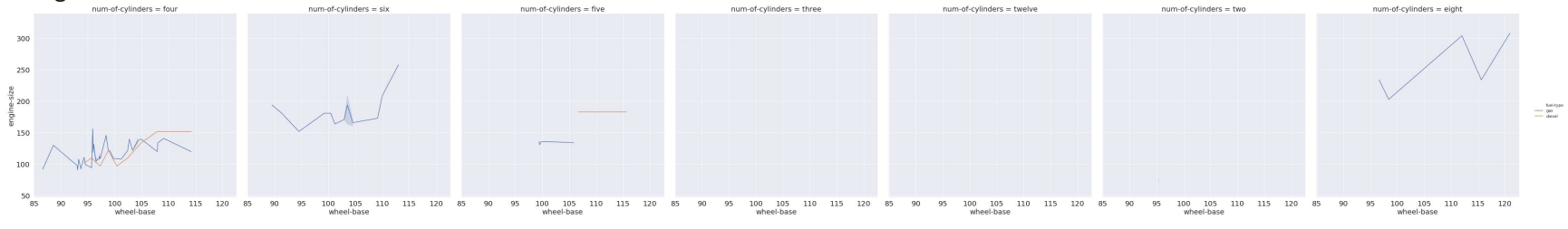
In [32]:
```
plt.figure(figsize=(14,16))
sns.relplot(x="wheel-base" , y="engine-size" , hue="num-of-cylinders" ,kind='scatter', height=8.5,aspect =1,data=auto)
plt.gcf().text(.5, .99, "Rel Plot", fontsize = 40, color='Black' ,ha='center', va='center')
plt.show()
```

```
<Figure size 1008x1152 with 0 Axes>
```

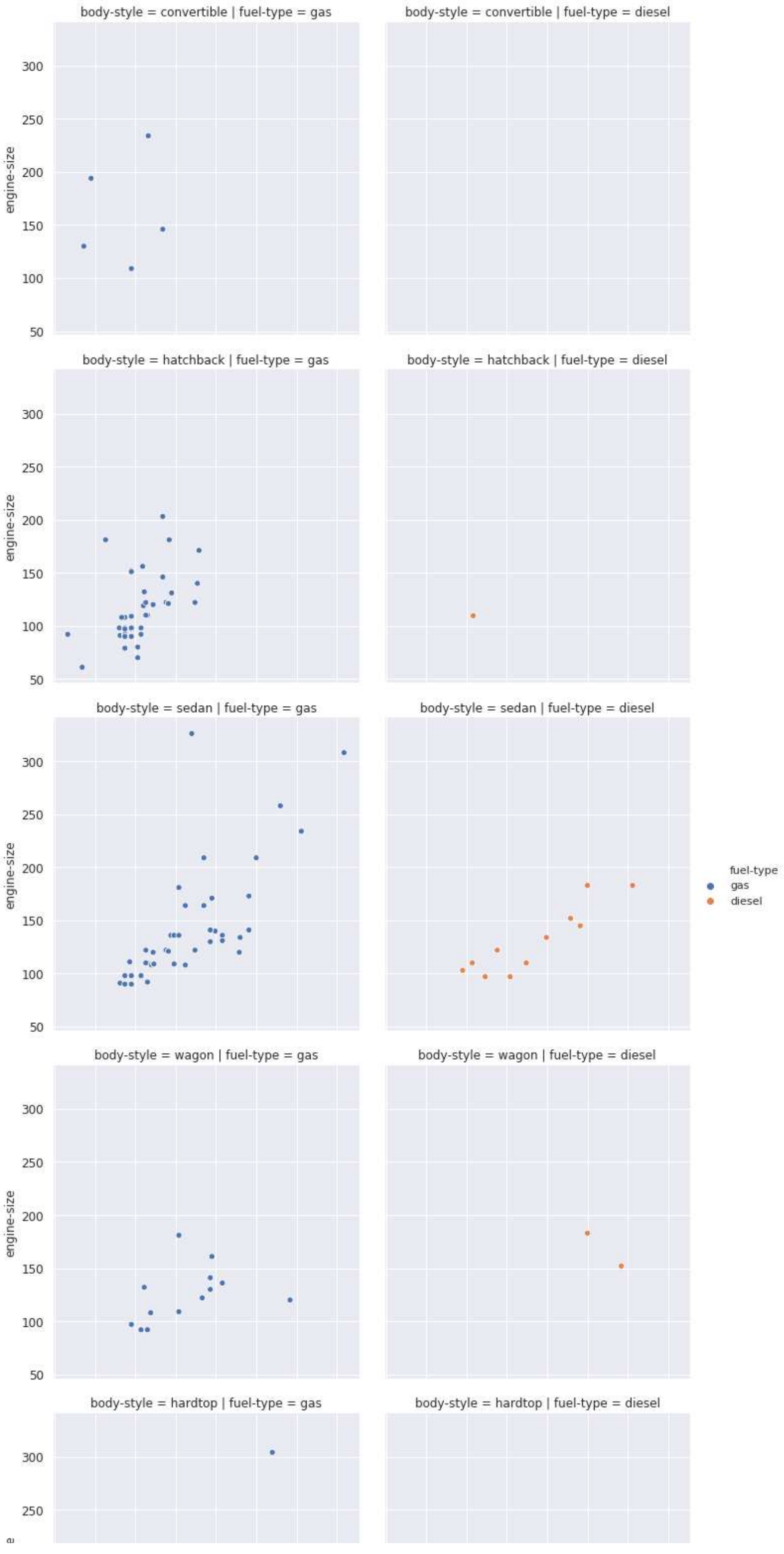


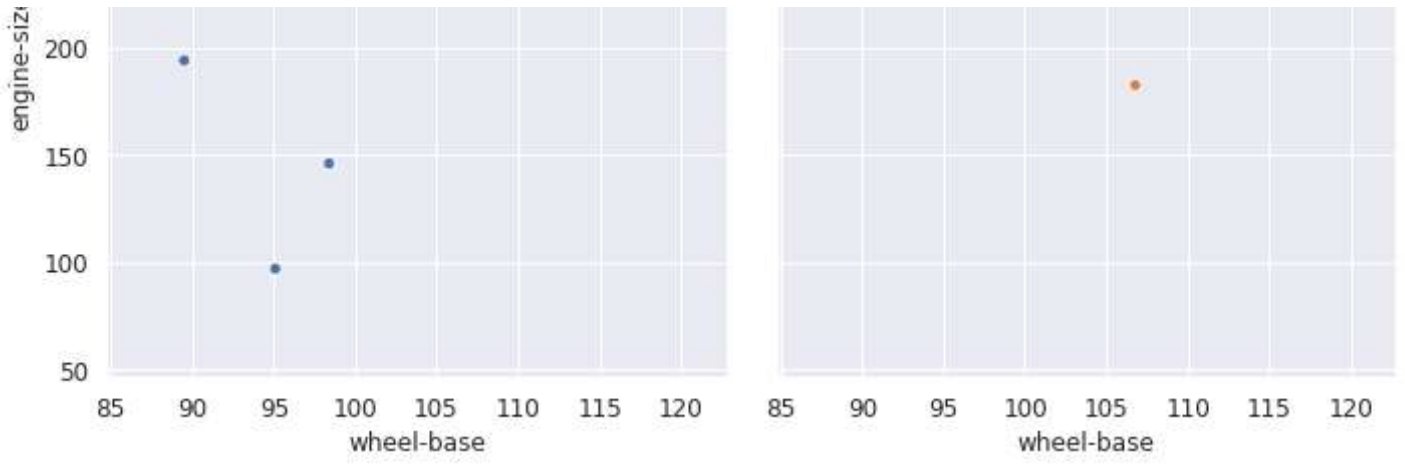
In [33]:
```
# Facet along the columns to show a categorical variable using "col" parameter

plt.figure(figsize=(14,20))
sns.set(rc={'xtick.labelsize':20,'ytick.labelsize':20,'axes.labelsize':20})
sns.relplot(x="wheel-base" , y="engine-size" , hue="fuel-type" ,kind='line',col="num-of-cylinders", height=8.5,aspect =
plt.show()
```

```
<Figure size 1008x1440 with 0 Axes>
```

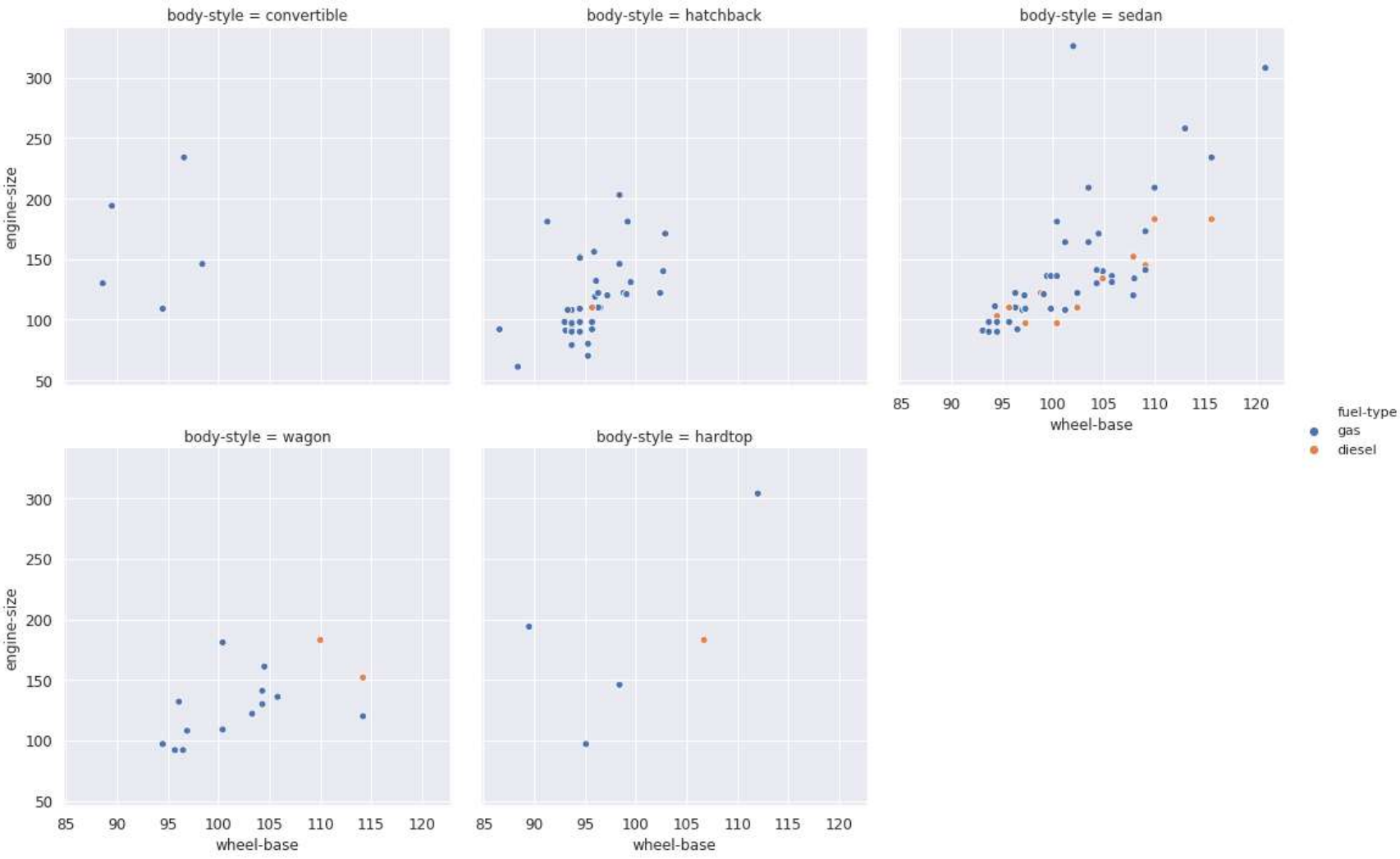


In [34]:
```
# Facet along the columns and rows to show a categorical variable using "col" and "row" par
sns.set(rc={'xtick.labelsize':12,'ytick.labelsize':12,'axes.labelsize':12})
sns.relplot(x="wheel-base", y="engine-size", hue="fuel-type",col="fuel-type", row="body-style", data=auto)
plt.show()
```


body-style = convertible | fuel-type = gas
body-style = convertible | fuel-type = diesel
body-style = hatchback | fuel-type = gas
body-style = hatchback | fuel-type = diesel
body-style = sedan | fuel-type = gas
body-style = sedan | fuel-type = diesel
body-style = wagon | fuel-type = gas
body-style = wagon | fuel-type = diesel
body-style = hardtop | fuel-type = gas
body-style = hardtop | fuel-type = diesel
engine-size
300
250
200
150
100
50
fuel-type
gas
diesel



In [35]:
```
# Limiting the number of columns using "col_wrap"

sns.set(rc={'xtick.labelsize':12,'ytick.labelsize':12,'axes.labelsize':12})
sns.relplot(x="wheel-base", y="engine-size", hue="fuel-type",col="body-style",col_wrap=3, data=auto)
plt.show()
```

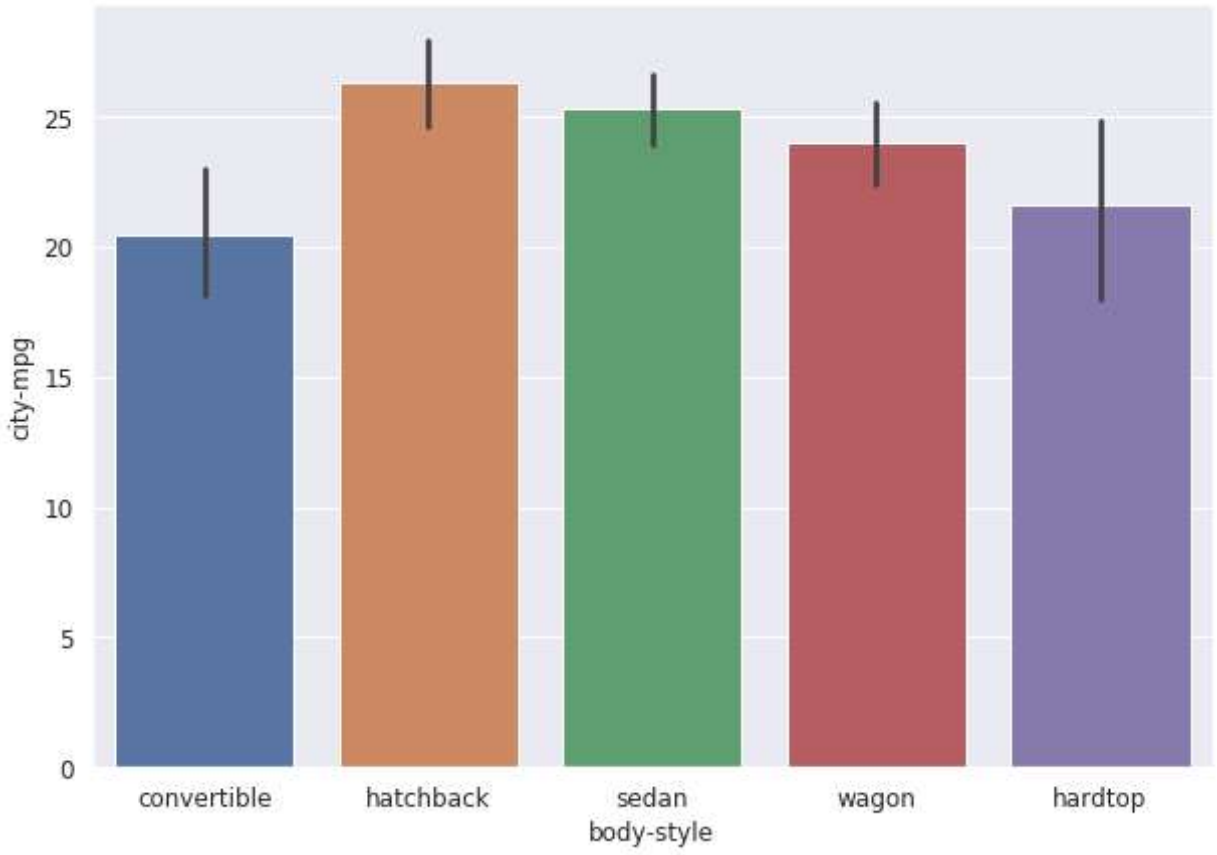


7. Bar Plot

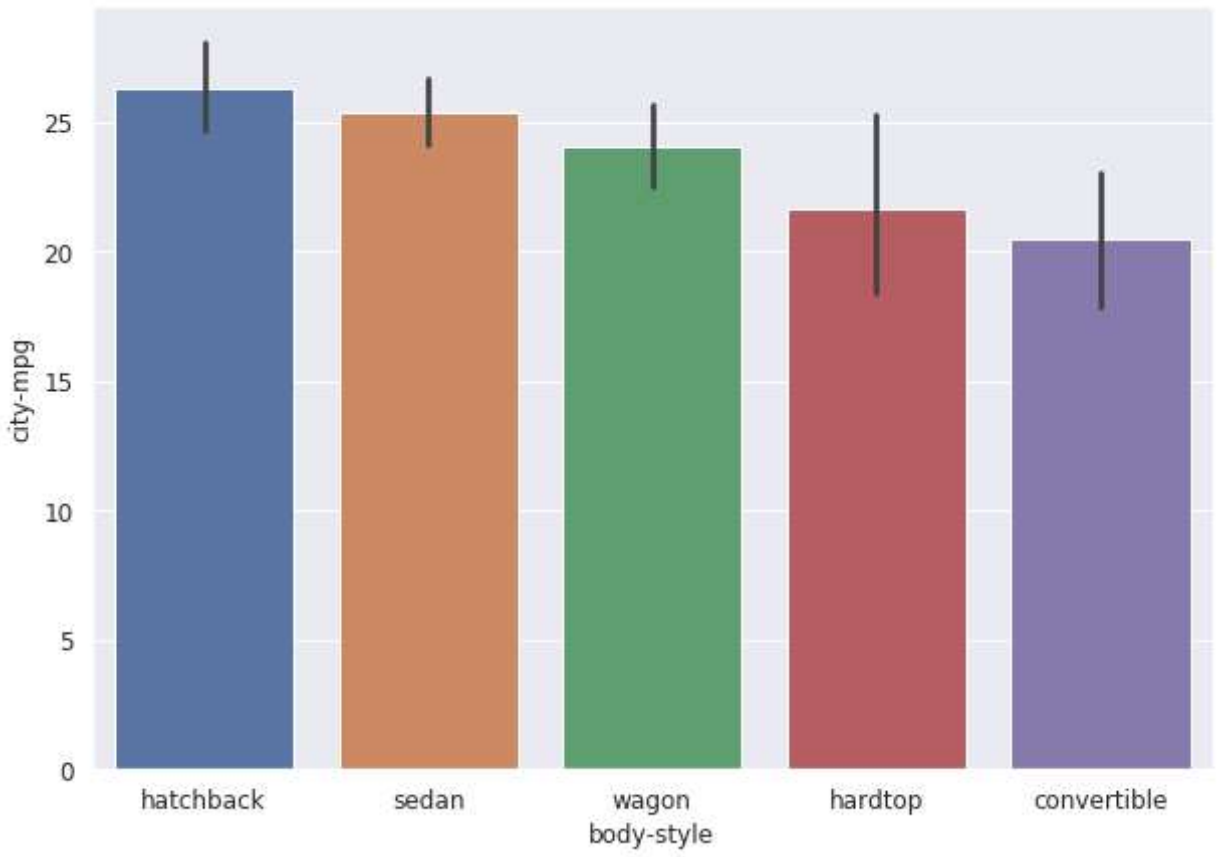
Bar Plot shows the relationship between a numerical variable and a categorical variable.A familiar style of plot that accomplishes this goal is a bar plot. In seaborn, the barplot() function operates on a full dataset and shows an arbitrary estimate, using the mean by default. When there are multiple 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.

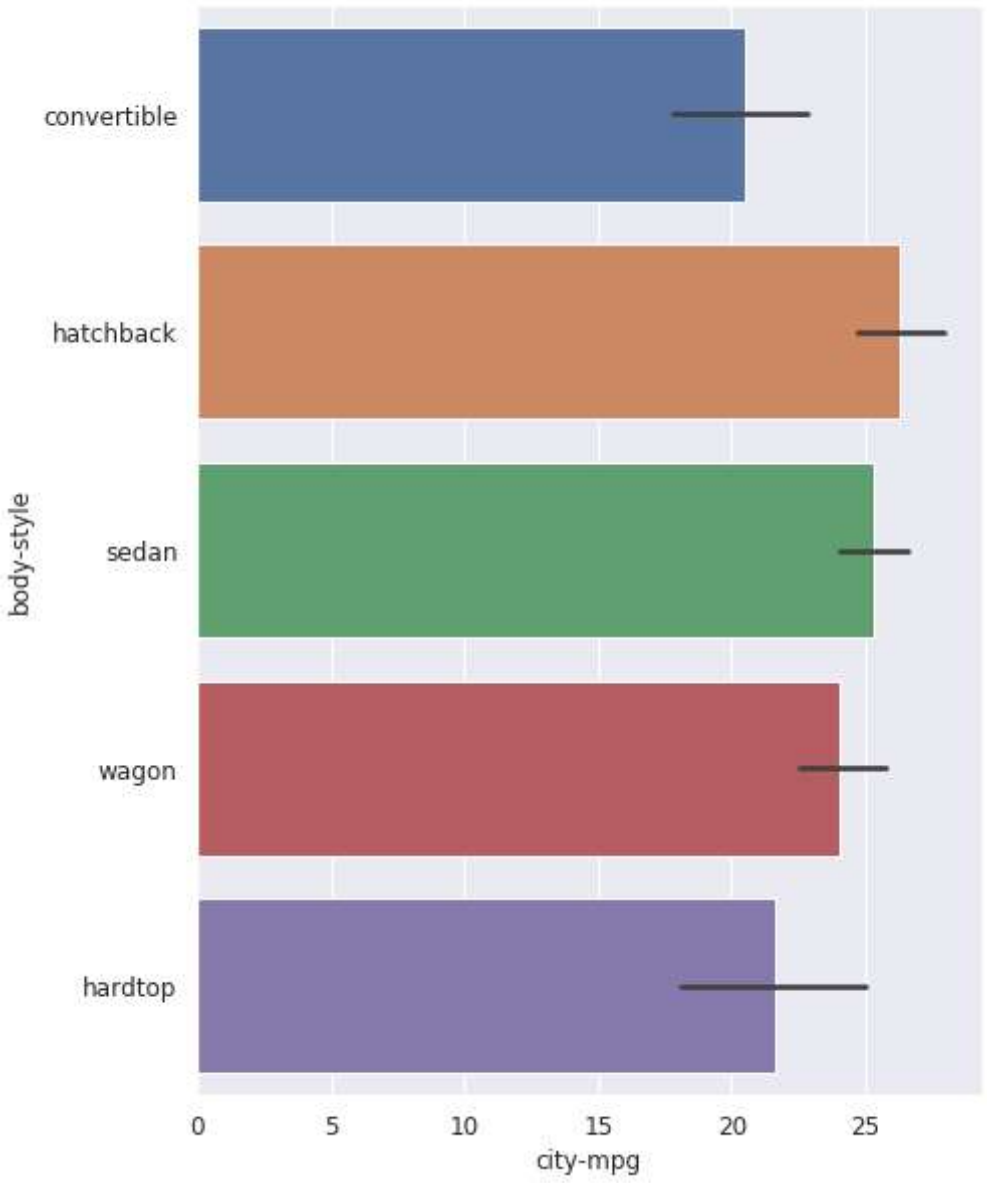
In [36]:
```
plt.figure(figsize=(10,7))
sns.barplot(auto['body-style'], auto['city-mpg'])
plt.show()
```



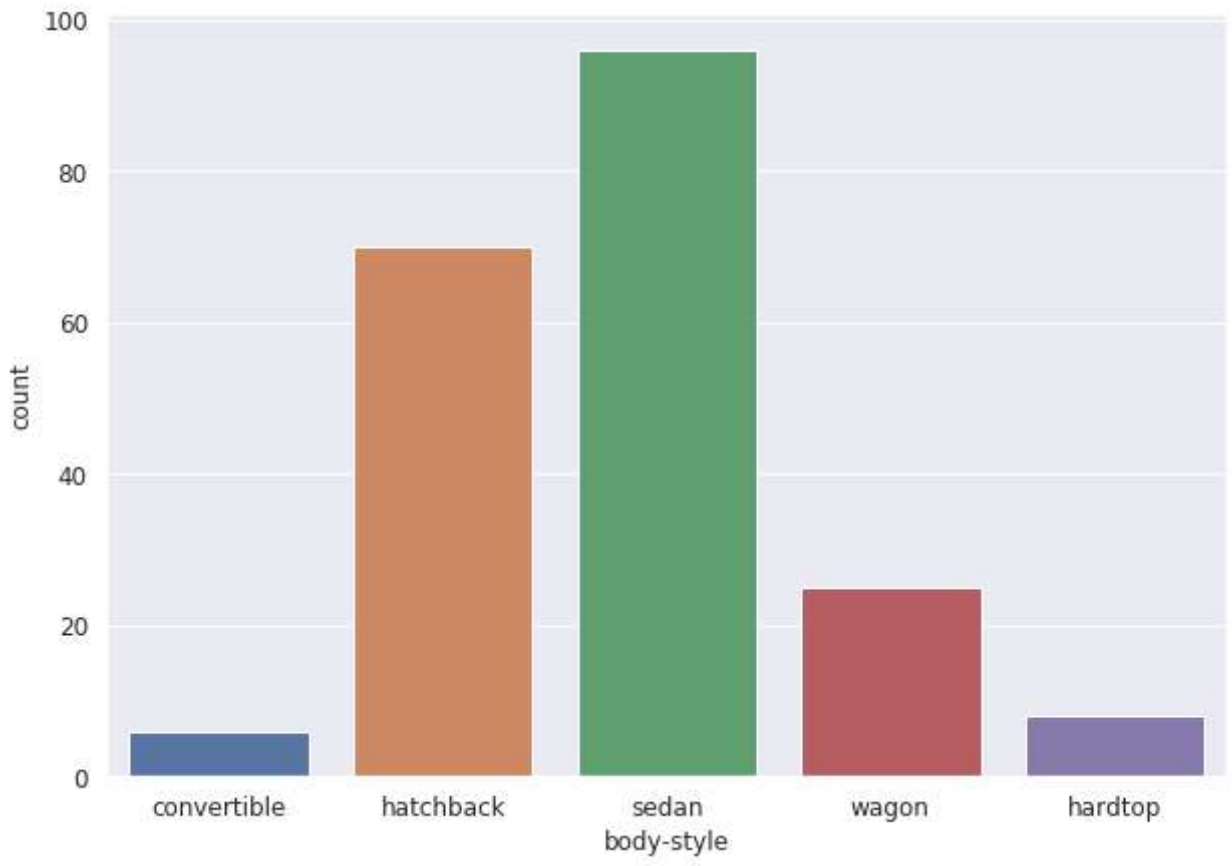
In [37]:
```
#Sorted Bar Plot

plt.figure(figsize=(10,7))
order = auto.groupby(['body-style']).mean().sort_values('city-mpg' , ascending = False).index
sns.barplot(auto['body-style'], auto['city-mpg'],order=order)
plt.show()
```



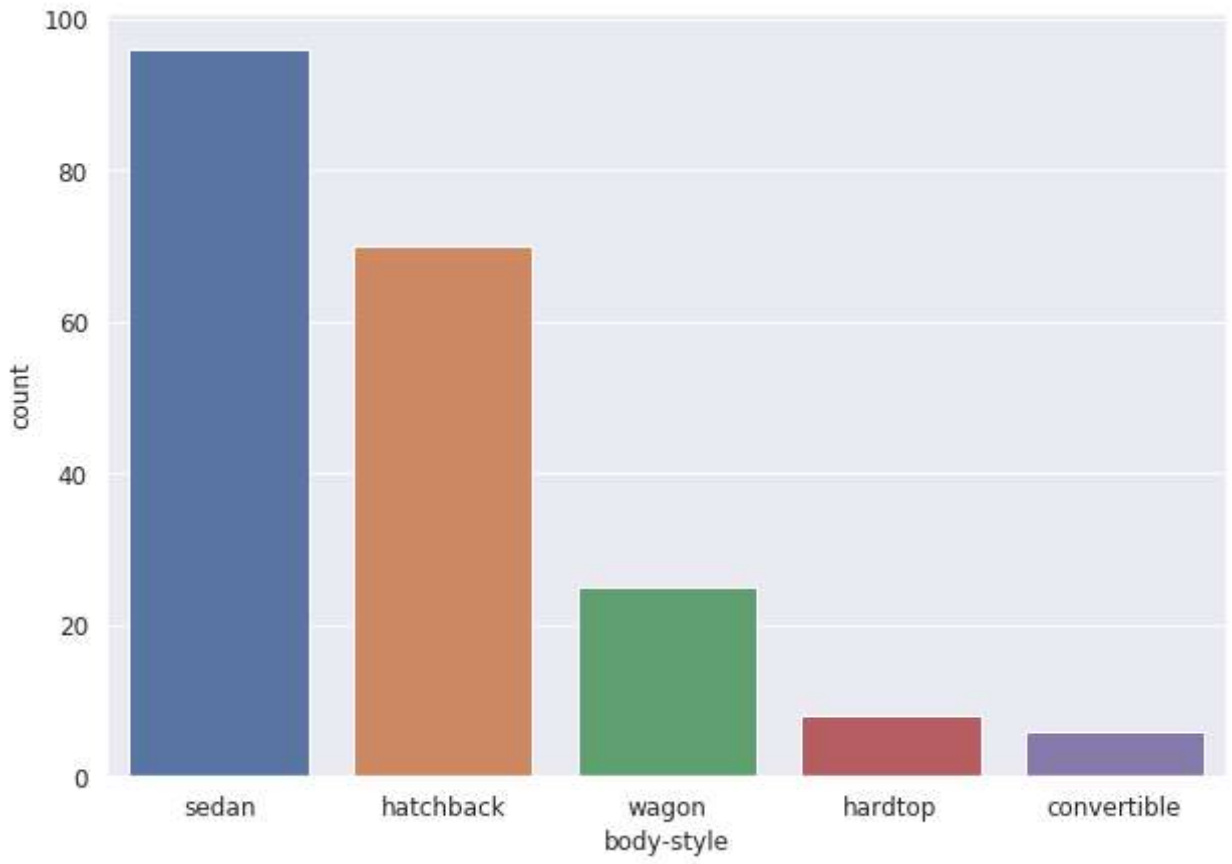
In [38]:
```
# Horizontal Bar plot
plt.figure(figsize=(7,10))
sns.barplot(auto['city-mpg'], auto['body-style'])
plt.show()
```



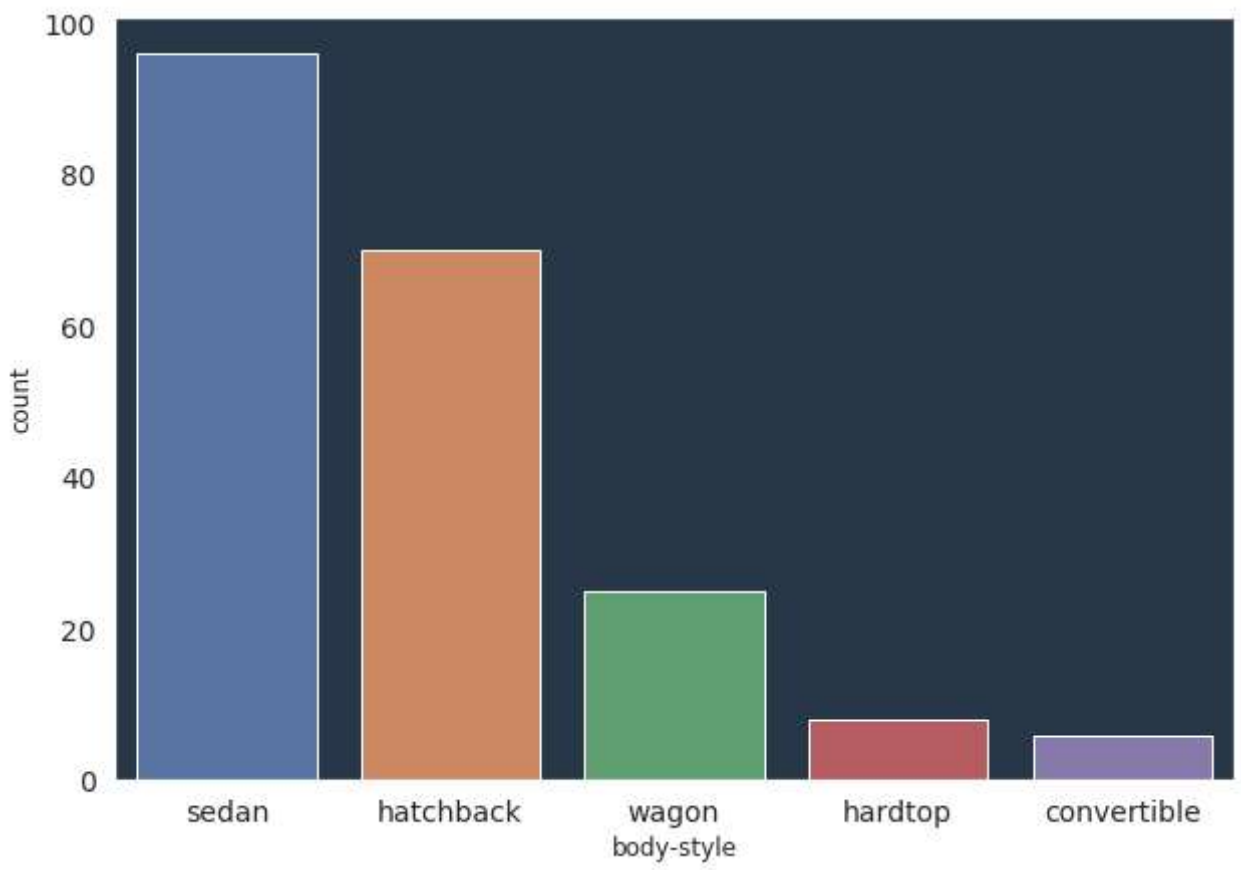
In [39]:
```
plt.figure(figsize=(10,7))
sns.countplot(auto['body-style'])
plt.show()
```



In [40]:
```
plt.figure(figsize=(10,7))
sns.countplot(auto['body-style'], order=auto['body-style'].value_counts().index)
plt.show()
```

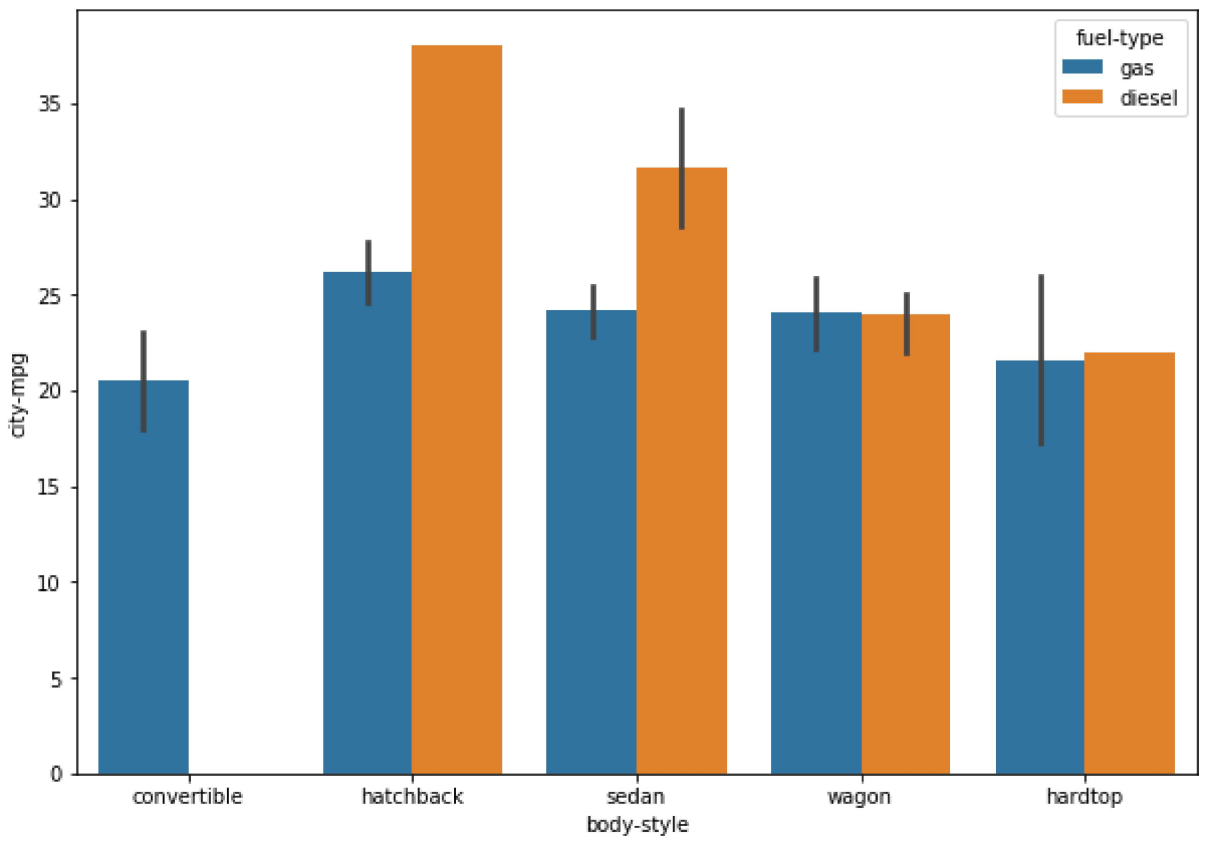


In [41]:
```
#Changing the background of bar plot
plt.figure(figsize=(10,7))
sns.set(rc={"axes.facecolor":"#283747", "axes.grid":False,'xtick.labelsize':14,'ytick.labelsize':14})
sns.countplot(auto['body-style'], order=auto['body-style'].value_counts().index)
plt.show()
```



In [42]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

In [43]:
```
# Adding hue
plt.figure(figsize=(10,7))
sns.barplot(auto['body-style'], auto['city-mpg'], hue=auto['fuel-type'])
plt.show()
```

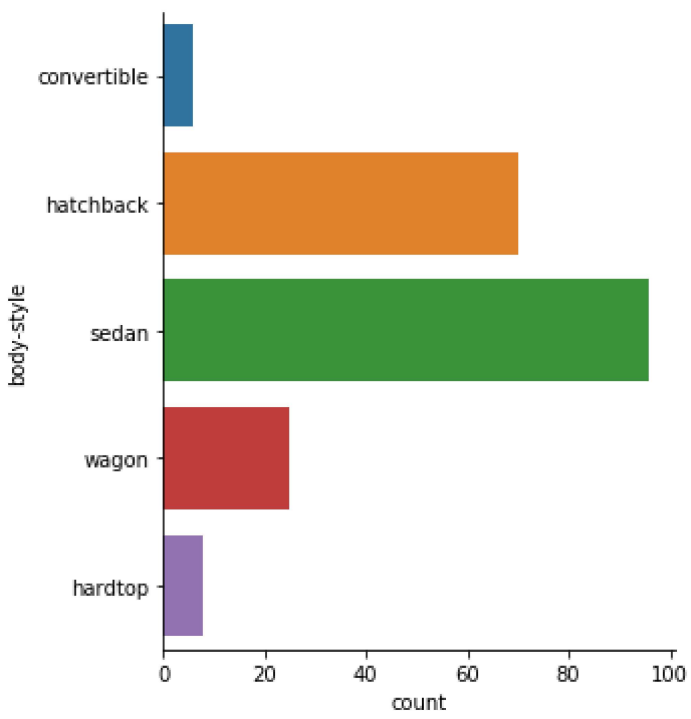


8.Cat plot

Cat Plot provides access to several axes-level functions ("point", "bar", "strip", "swarm","box", "violin", "count" or "boxen") that show the relationship between a numerical and one or more categorical variables

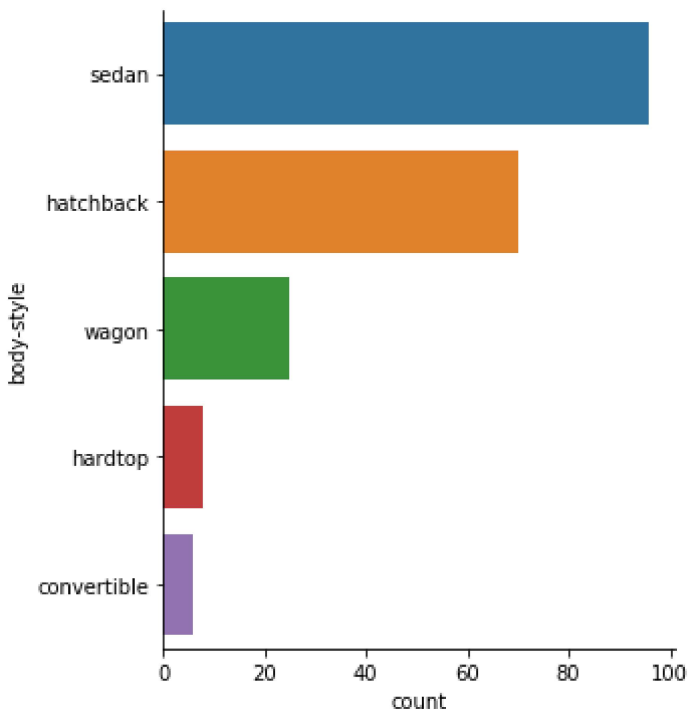
In [44]:
```
# Use Count Plot to visualize data
plt.figure(figsize=(10,10))
sns.catplot(y ='body-style', kind = "count",data = auto)
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



In [45]:
```
# Use sorted  Count Plot to visualize data
plt.figure(figsize=(7,10))
sns.catplot(y ='body-style', kind = "count",data = auto,order=auto['body-style'].value_counts().index)
plt.show()
```

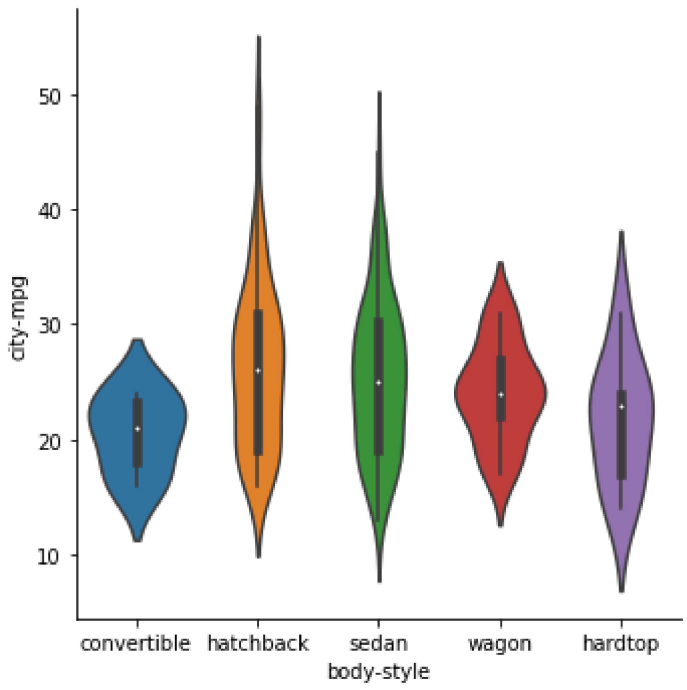
```
<Figure size 504x720 with 0 Axes>
```



In [46]:
```
# Use Voilin Plot to visualize data
plt.figure(figsize=(12,10))
```

```
sns.catplot(x="body-style" , y = "city-mpg" ,kind="violin" ,data=auto)
plt.show()
```

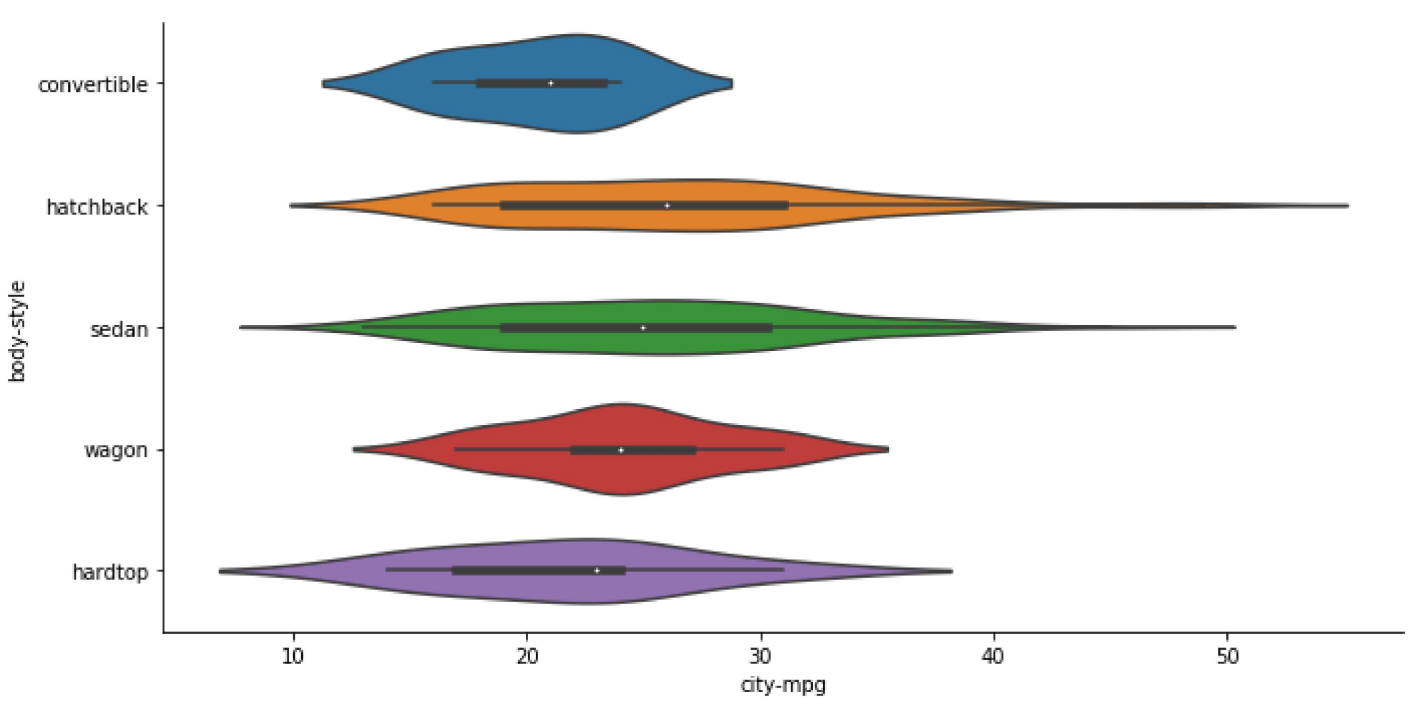
```
<Figure size 864x720 with 0 Axes>
```



In [47]:

```
plt.figure(figsize=(11,9))
sns.catplot(x = "city-mpg" , y="body-style" ,kind="violin" ,data=auto ,height=5, aspect=2)
plt.show()
```

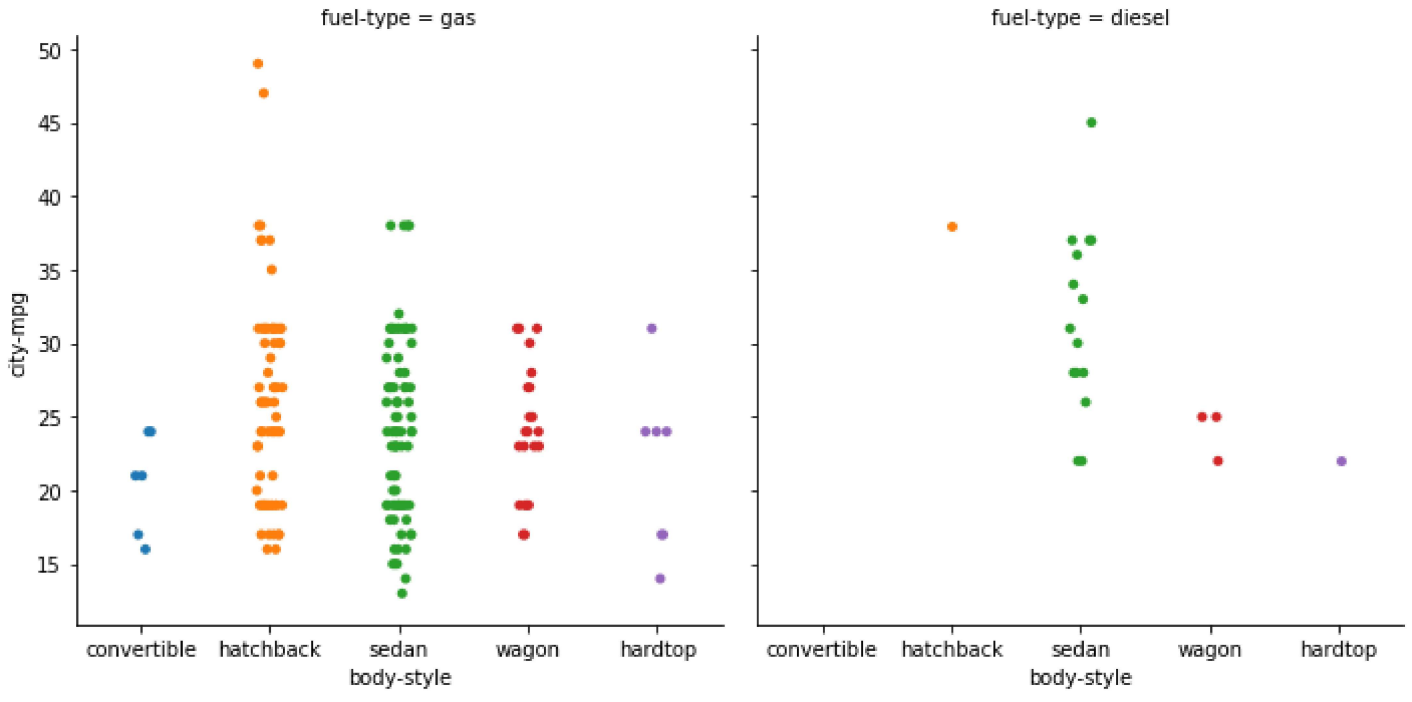
```
<Figure size 792x648 with 0 Axes>
```



In [48]:

```
# Facet along the columns to show a categorical variable using "col" parameter
plt.figure(figsize=(10,10))
sns.catplot(x='body-style' , y='city-mpg' , data=auto , col="fuel-type")
plt.show()
```

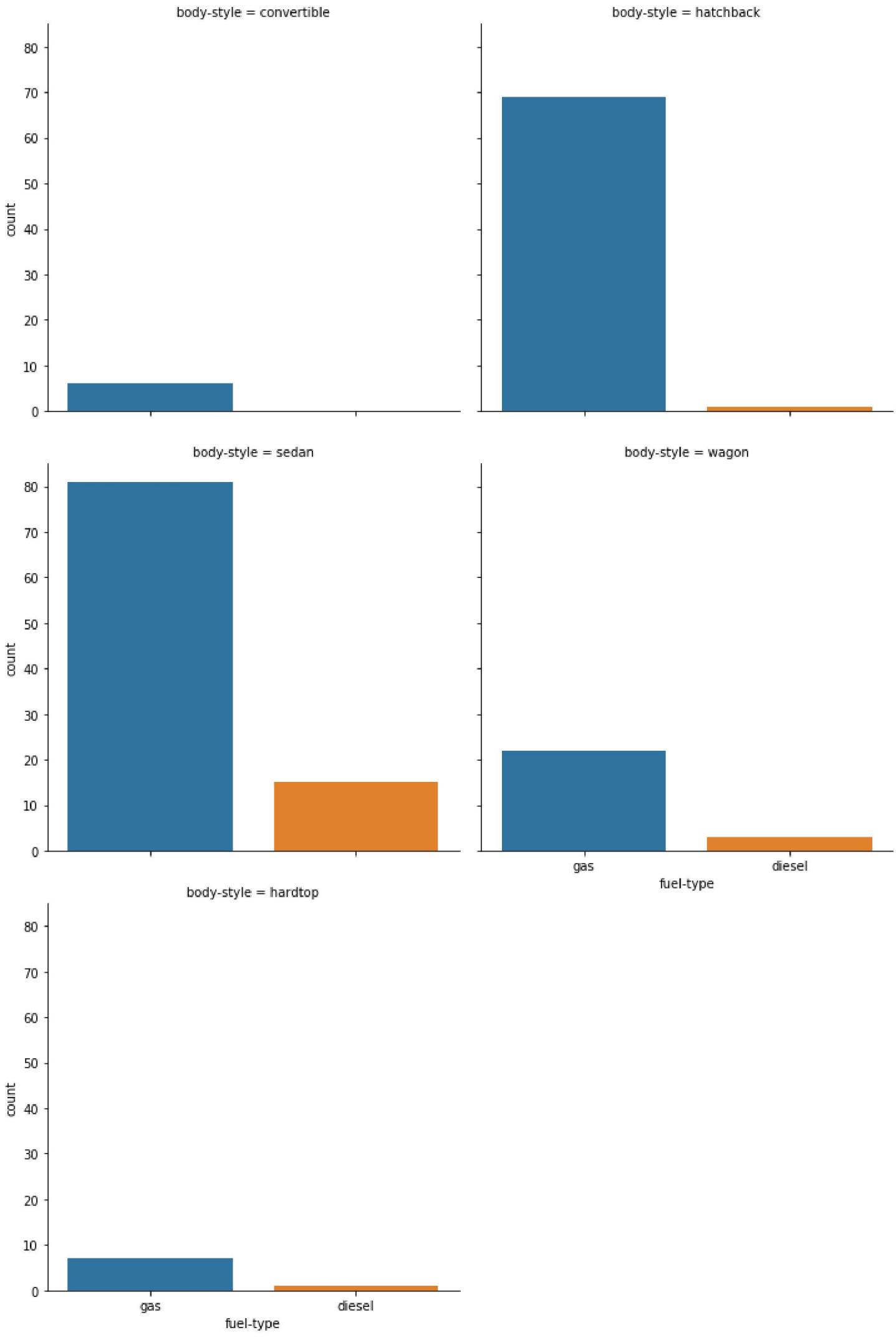
```
<Figure size 720x720 with 0 Axes>
```



In [49]:

```
""" Facet along the columns to show a categorical variable using "col" parameter and
Limiting the number of columns using "col_wrap" """
plt.figure(figsize=(20,10))
sns.catplot("fuel-type", col="body-style",col_wrap=2,data=auto,kind="count",height=5, aspect=1)
plt.show()
```

```
<Figure size 1440x720 with 0 Axes>
```

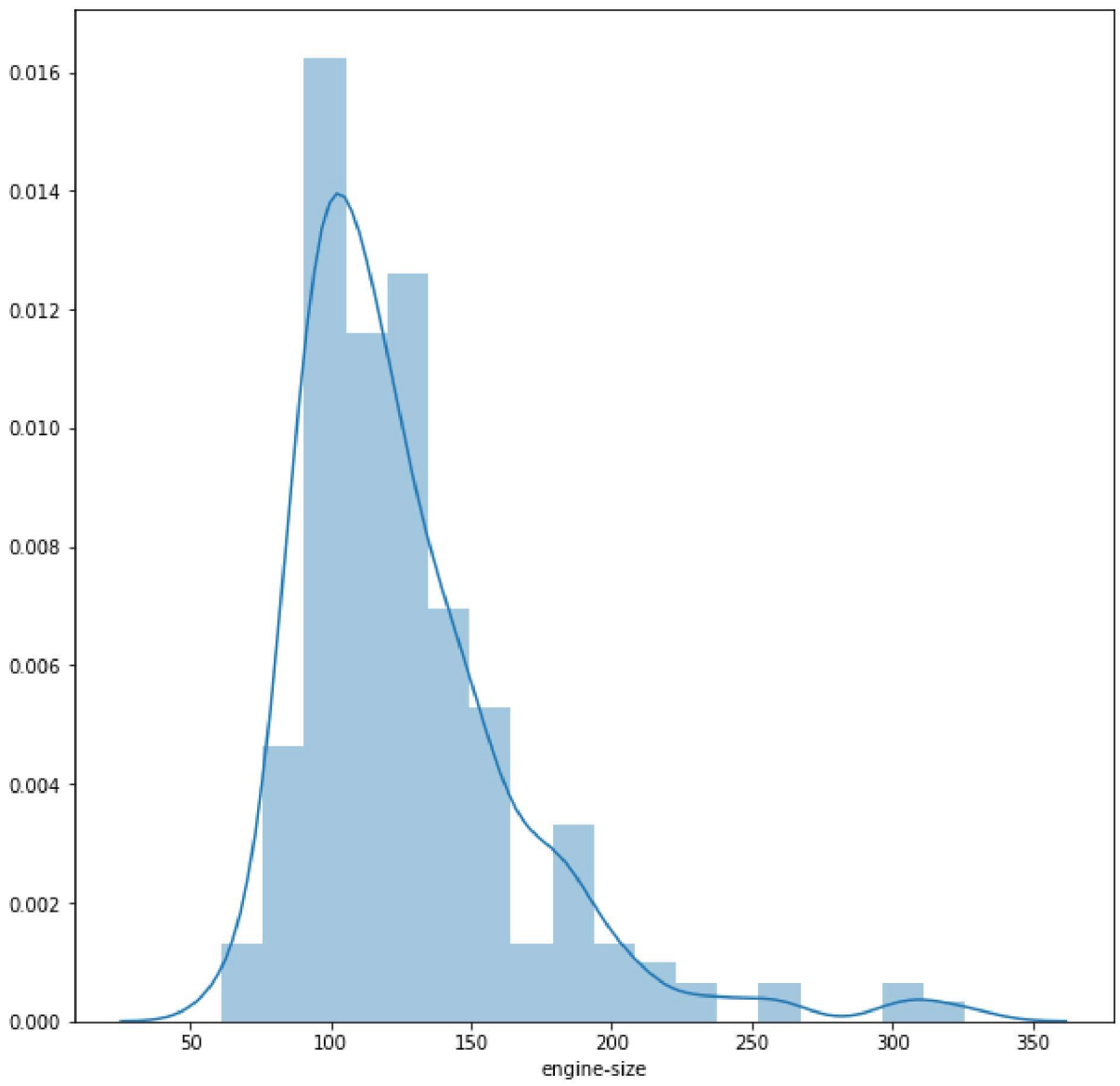


9, Dist Plot

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 [50]:
```
plt.figure(figsize=(10,10))
sns.distplot(auto['engine-size'])
plt.show()
```

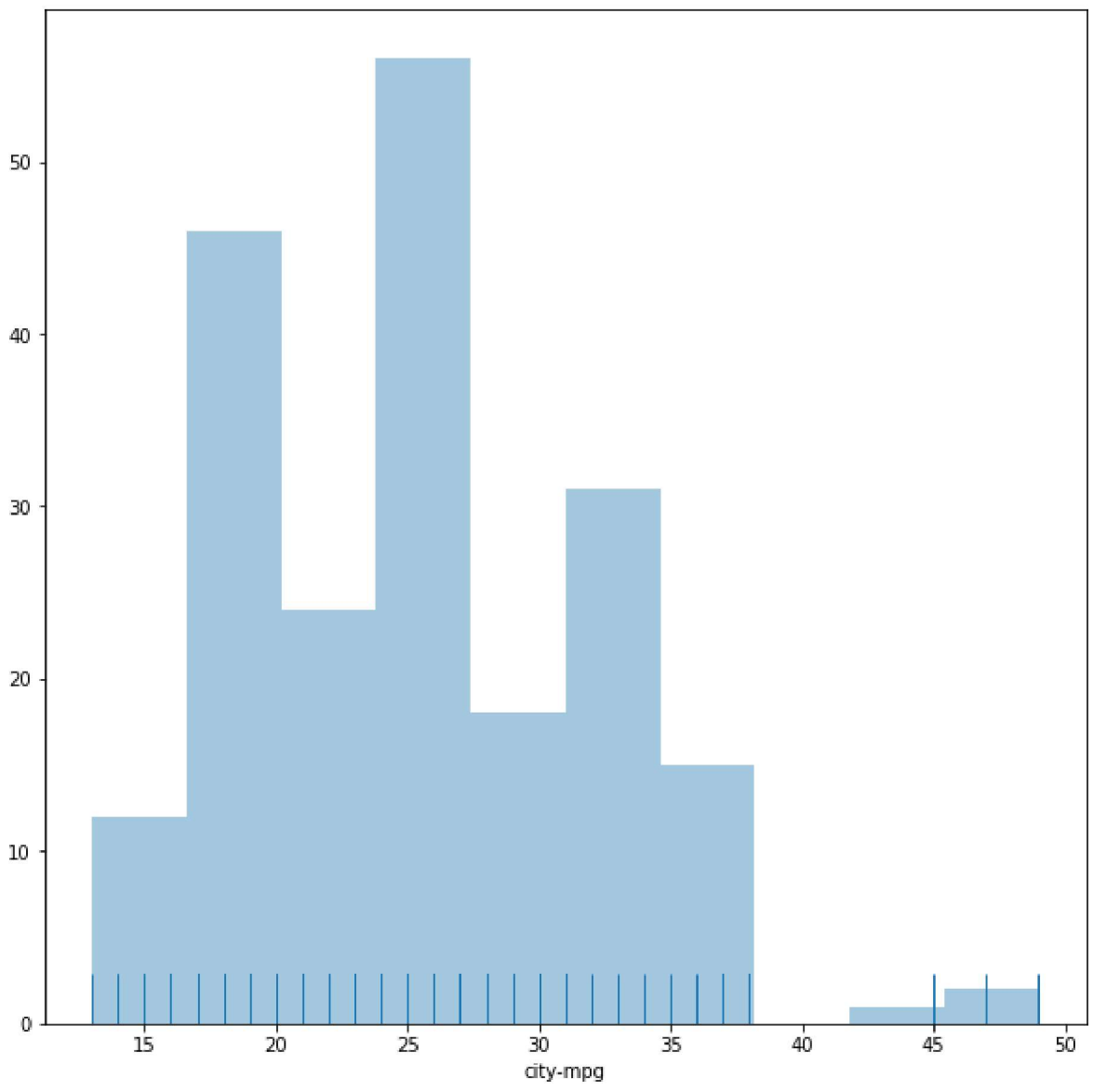


Histograms

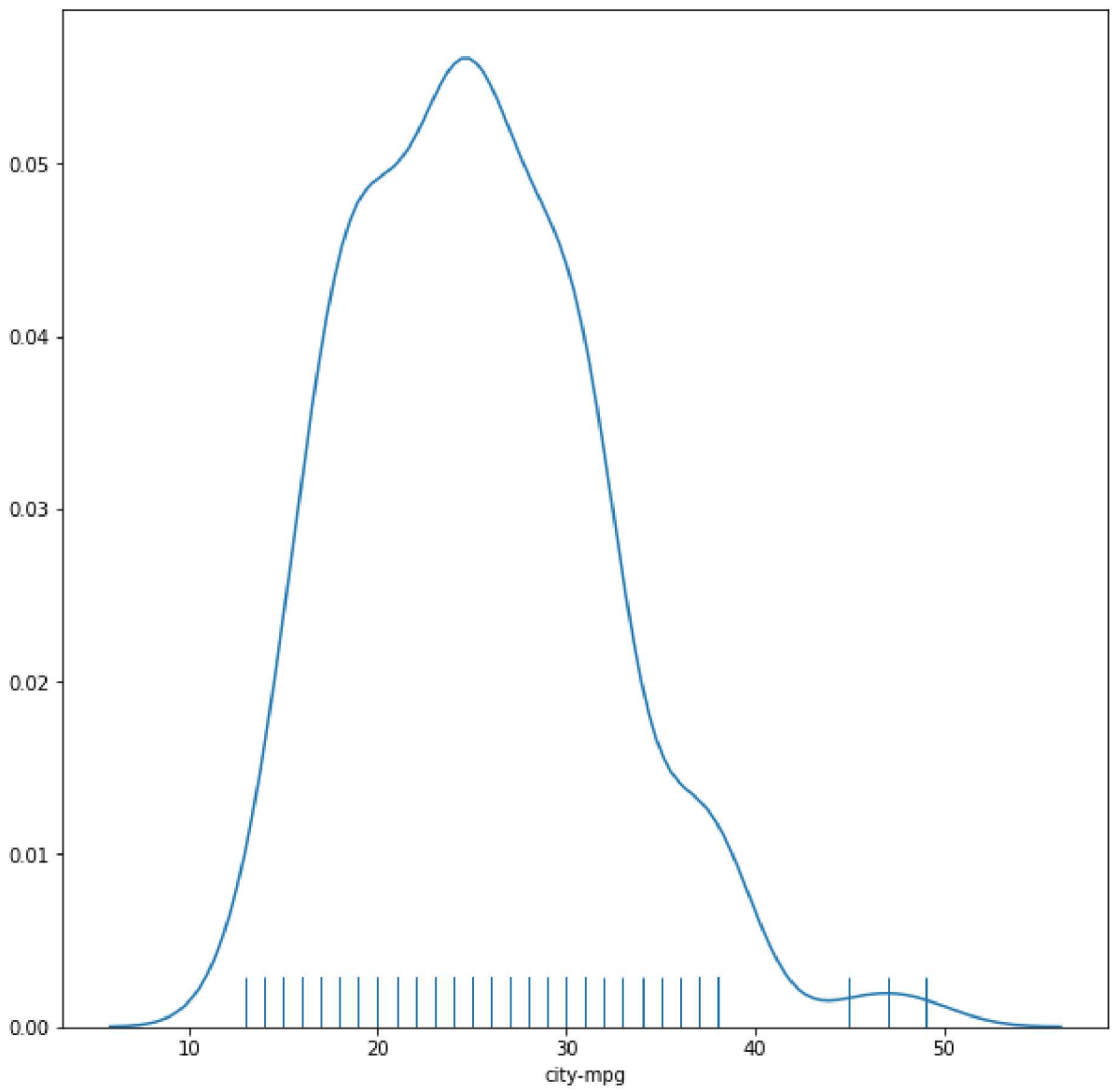
Histograms are likely familiar, and a hist function already exists in matplotlib. A histogram represents the distribution of data by forming bins along the range of the data and then drawing bars to show the number of observations that fall in each bin.

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():

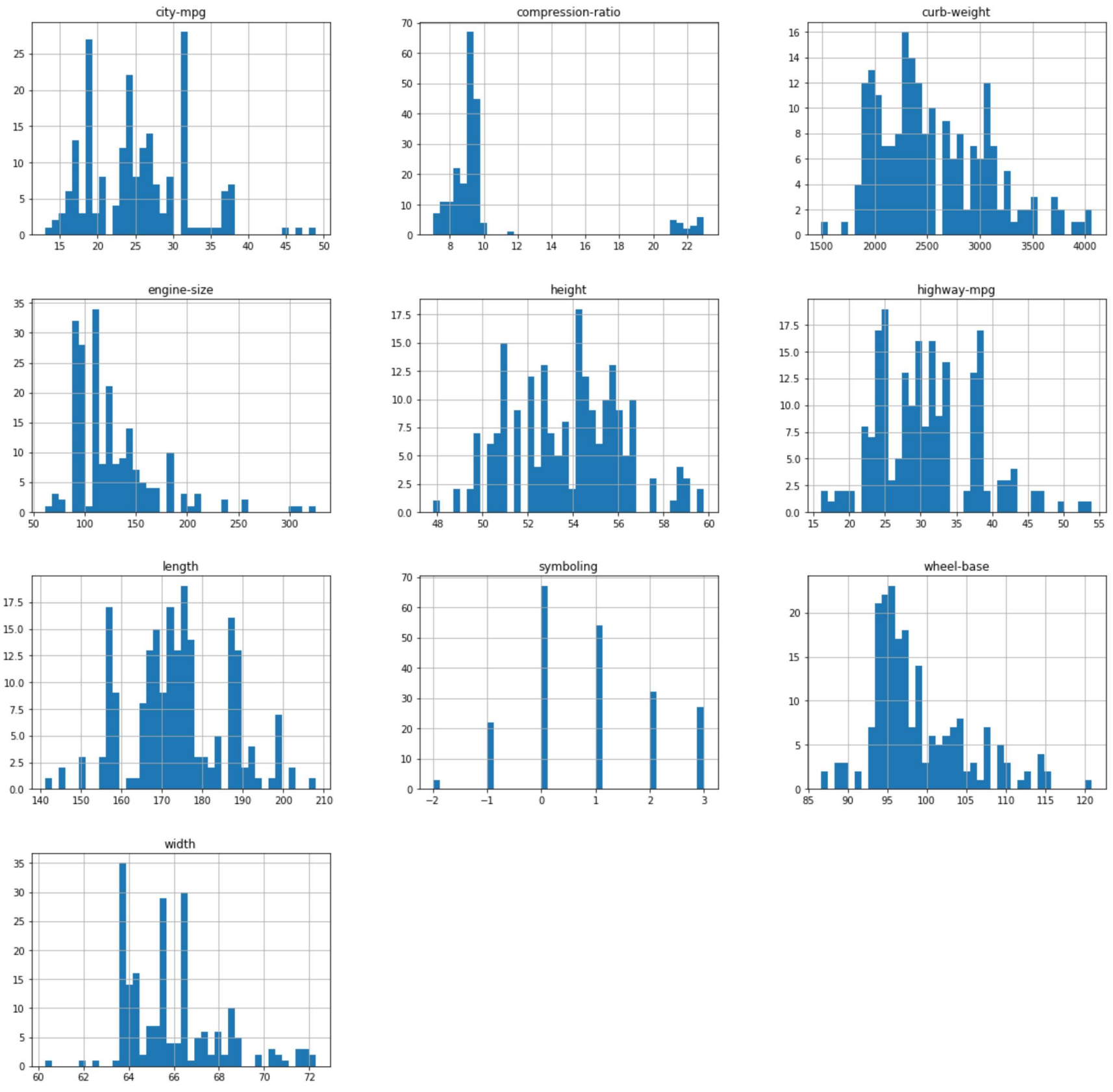
In [51]:
```
plt.figure(figsize=(10,10))
sns.distplot(auto['city-mpg'], kde=False, rug=True)
plt.show()
```



In [52]:
```
# Histogram with rugplot and kde
plt.figure(figsize=(10,10))
sns.distplot(auto['city-mpg'], kde=True, rug=True,hist=False)
plt.show()
```

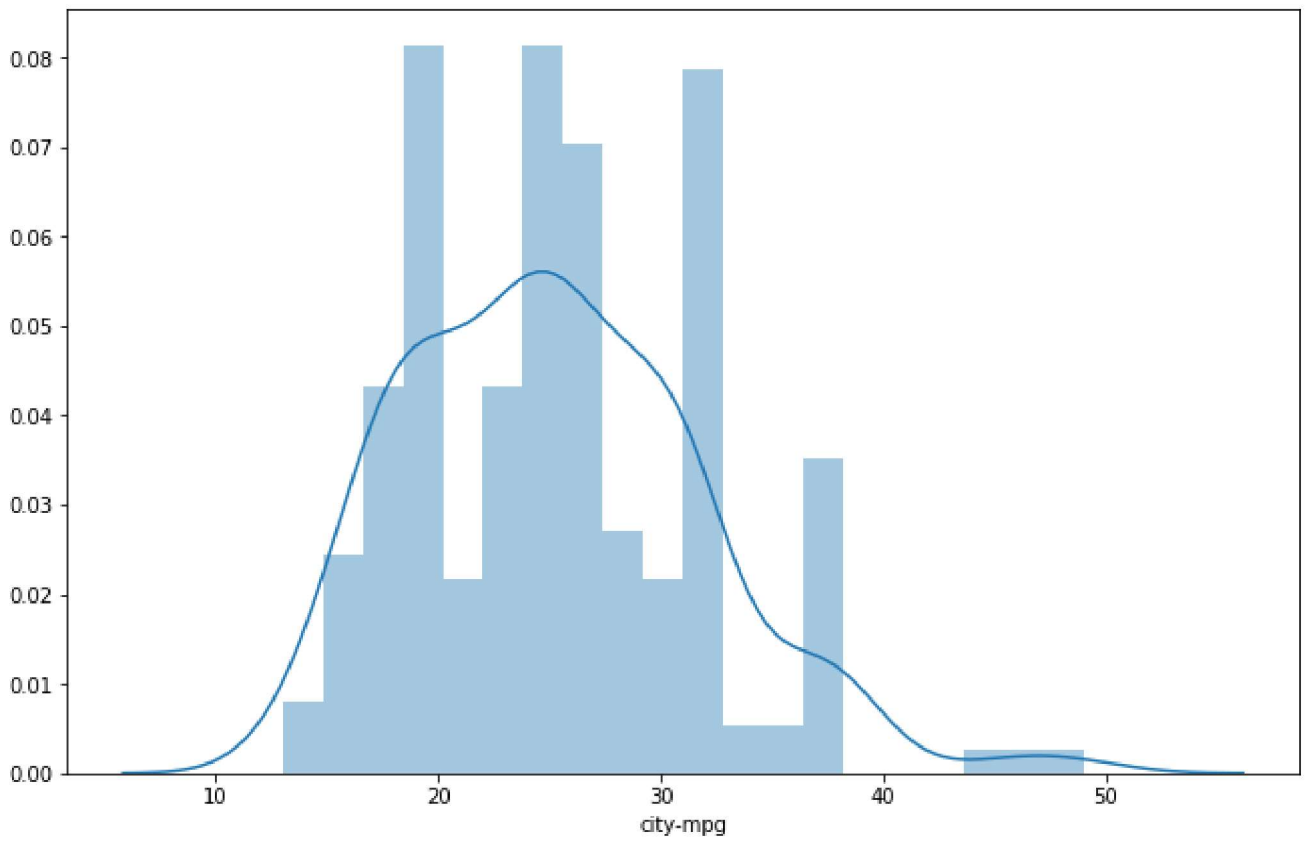


In [53]:
```
auto.hist(bins=40 , figsize=(20,20)) #Pandas Hist function
plt.show()
```

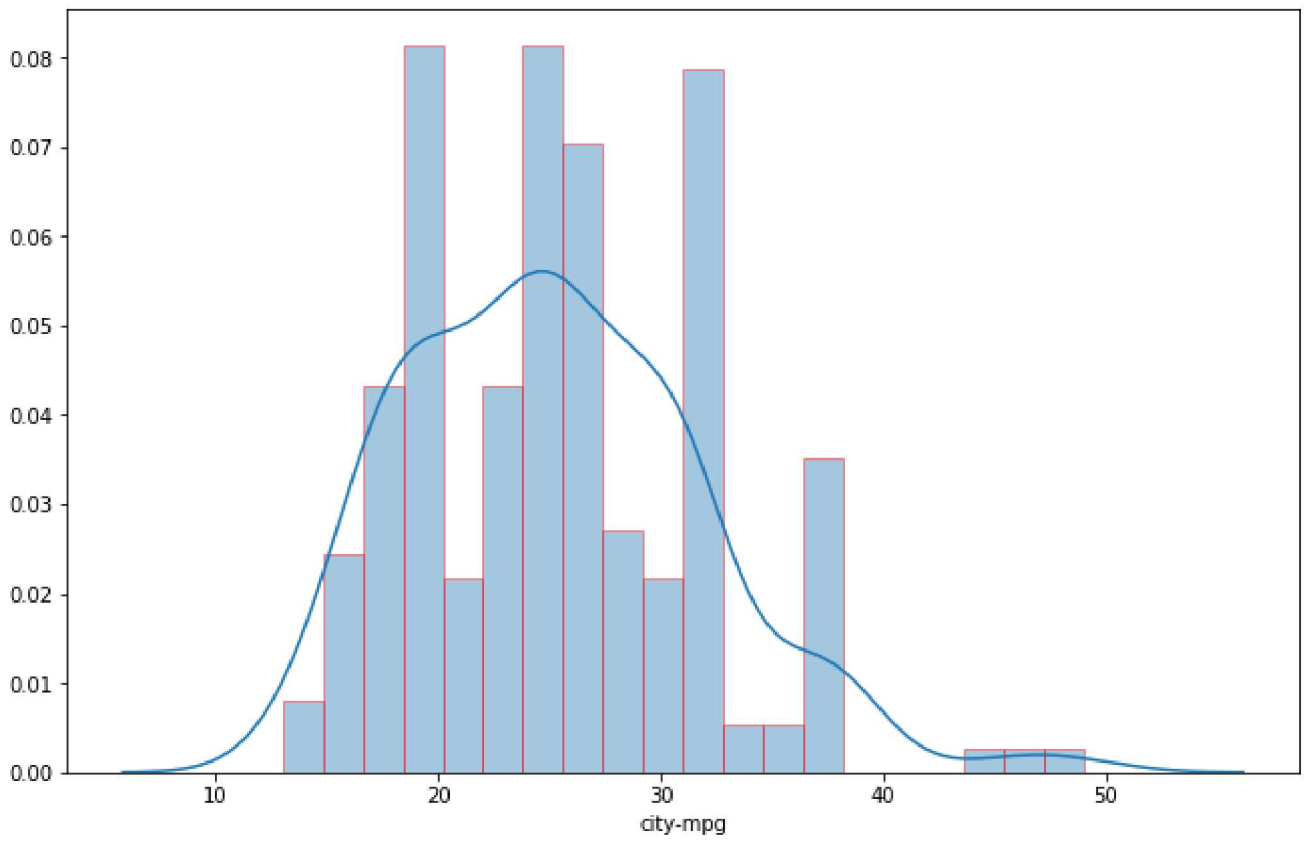


In [54]:
```
plt.figure(figsize=(11,7))
sns.distplot(auto['city-mpg'], bins=20)
```

```
plt.show()
```



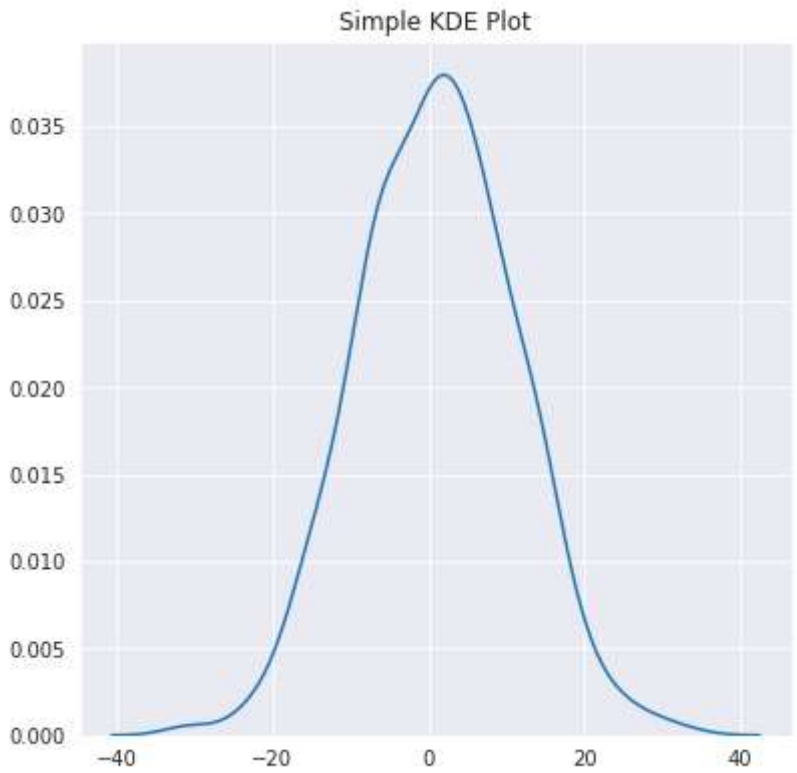
In [55]:
```
plt.figure(figsize=(11,7))
sns.distplot(auto['city-mpg'], bins=20 ,hist_kws=dict(edgecolor = '#FF0000'))
plt.show()
```

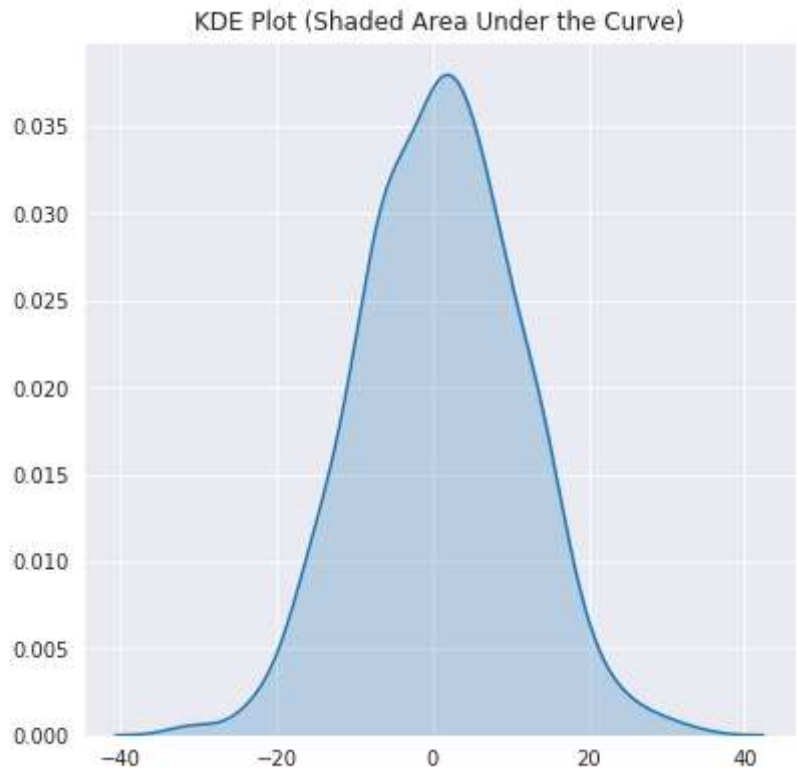


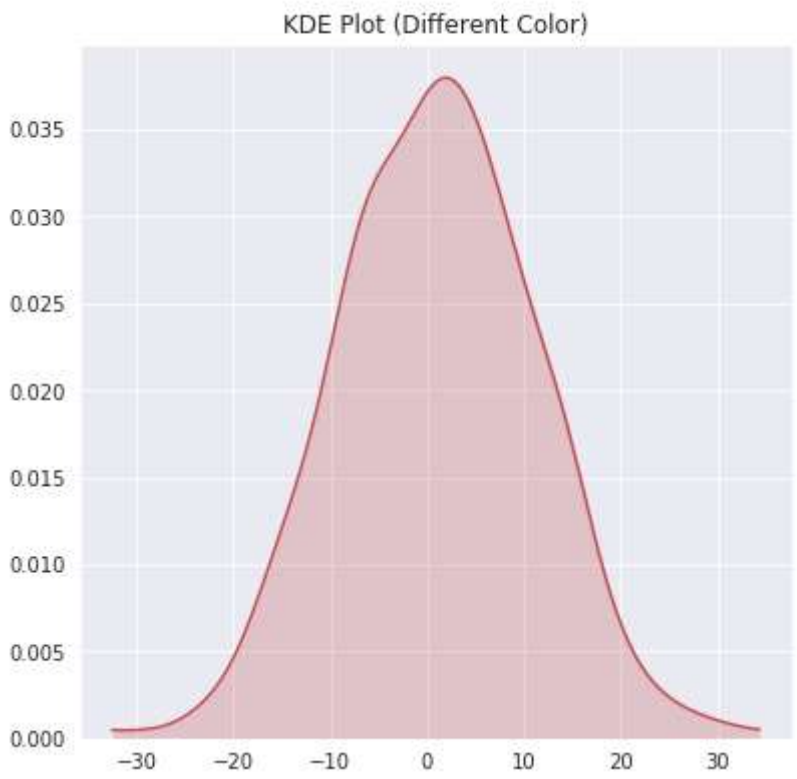
10.KDE Plot

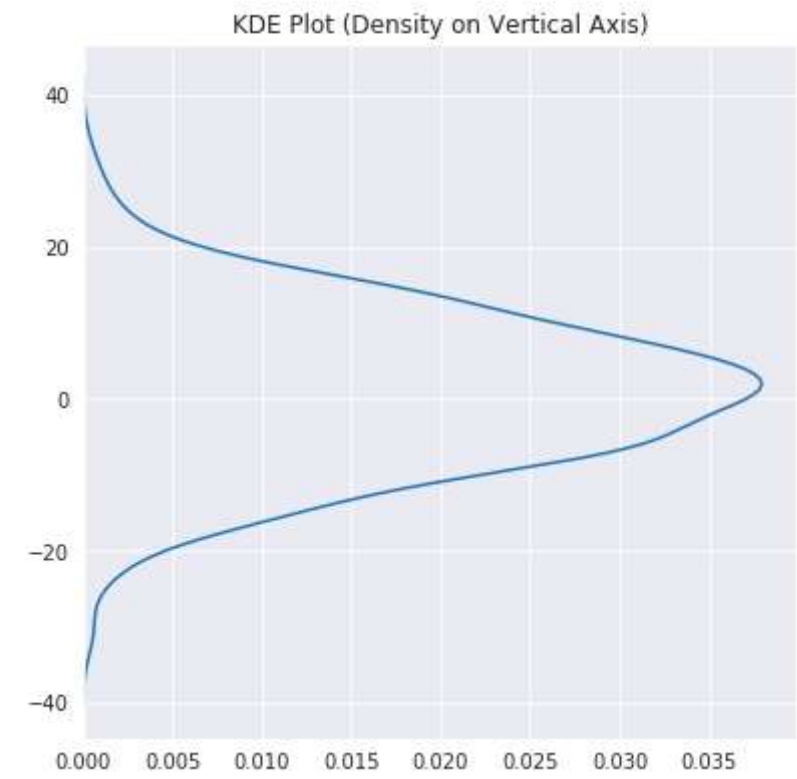
KDE Plot is used to estimate the probability density function of a continuous random variable.

In [56]:
```
sns.set_style("darkgrid")
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (14,14))
x = np.random.normal(1,10,1000)
#Simple KDE Plot
axes[0,0].set_title("Simple KDE Plot")
sns.kdeplot(x,ax=axes[0,0])
# Shade under the density curve using the "shade" parameter
axes[0,1].set_title("KDE Plot (Shaded Area Under the Curve)")
sns.kdeplot(x,shade=True,ax=axes[0,1])
# Shade under the density curve using the "shade" parameter and use a different color.
axes[1,0].set_title("KDE Plot (Different Color)")
sns.kdeplot(x,ax=axes[1,0],color = 'r',shade=True,cut=0)
#Plotting the density on the vertical axis
axes[1,1].set_title("KDE Plot (Density on Vertical Axis)")
sns.kdeplot(x,vertical=True)
plt.show()
```



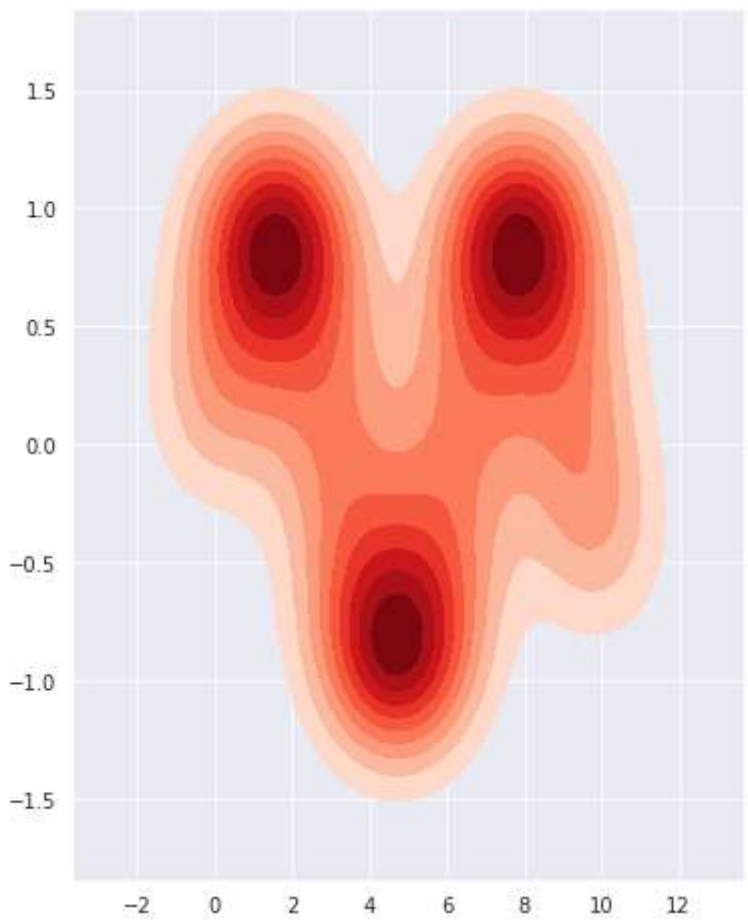




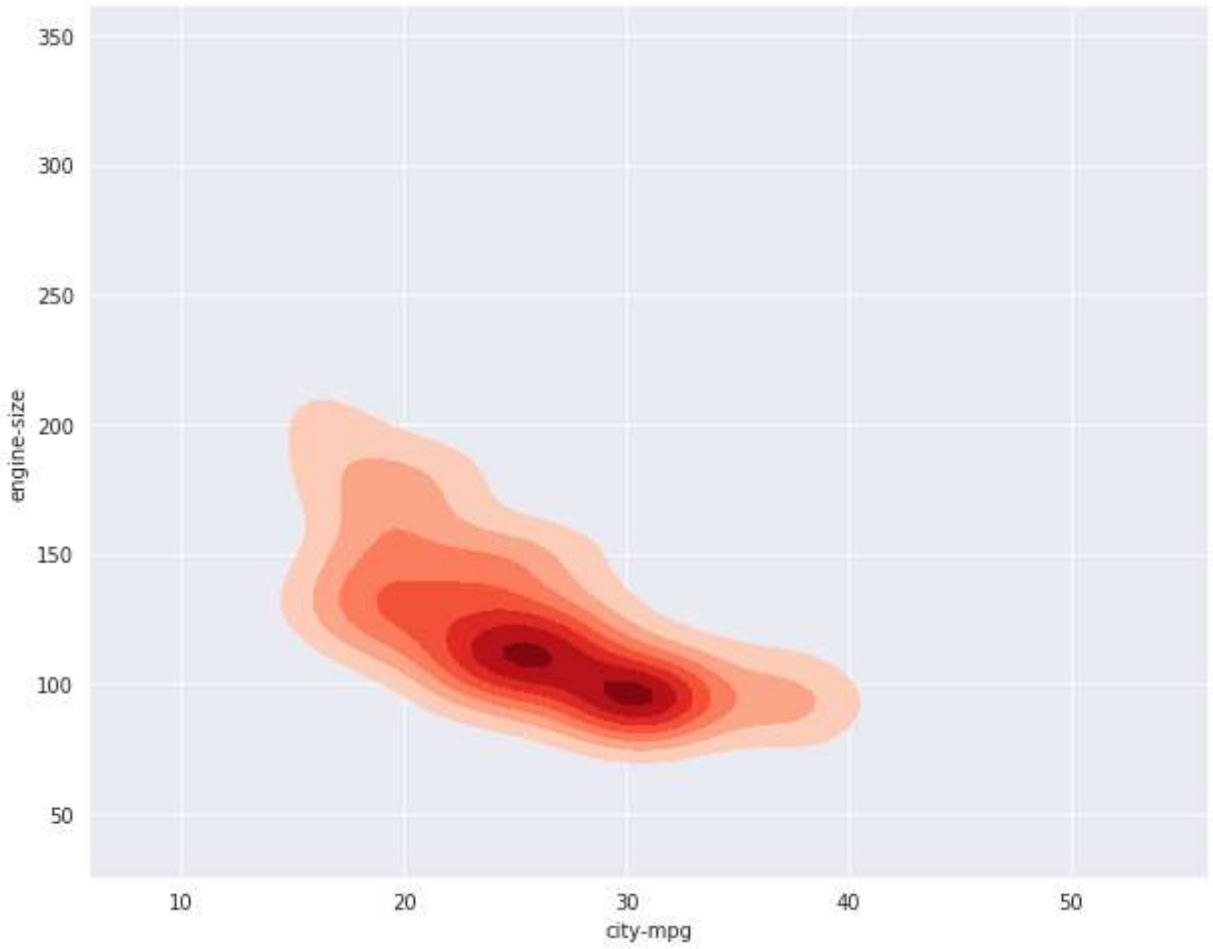


In [57]:
```
plt.figure(figsize=(6,8))
x = np.linspace(0, 10, 100)
y = np.sin(x)
sns.kdeplot(x,y,shade=True,cmap="Reds", shade_lowest=False)
```

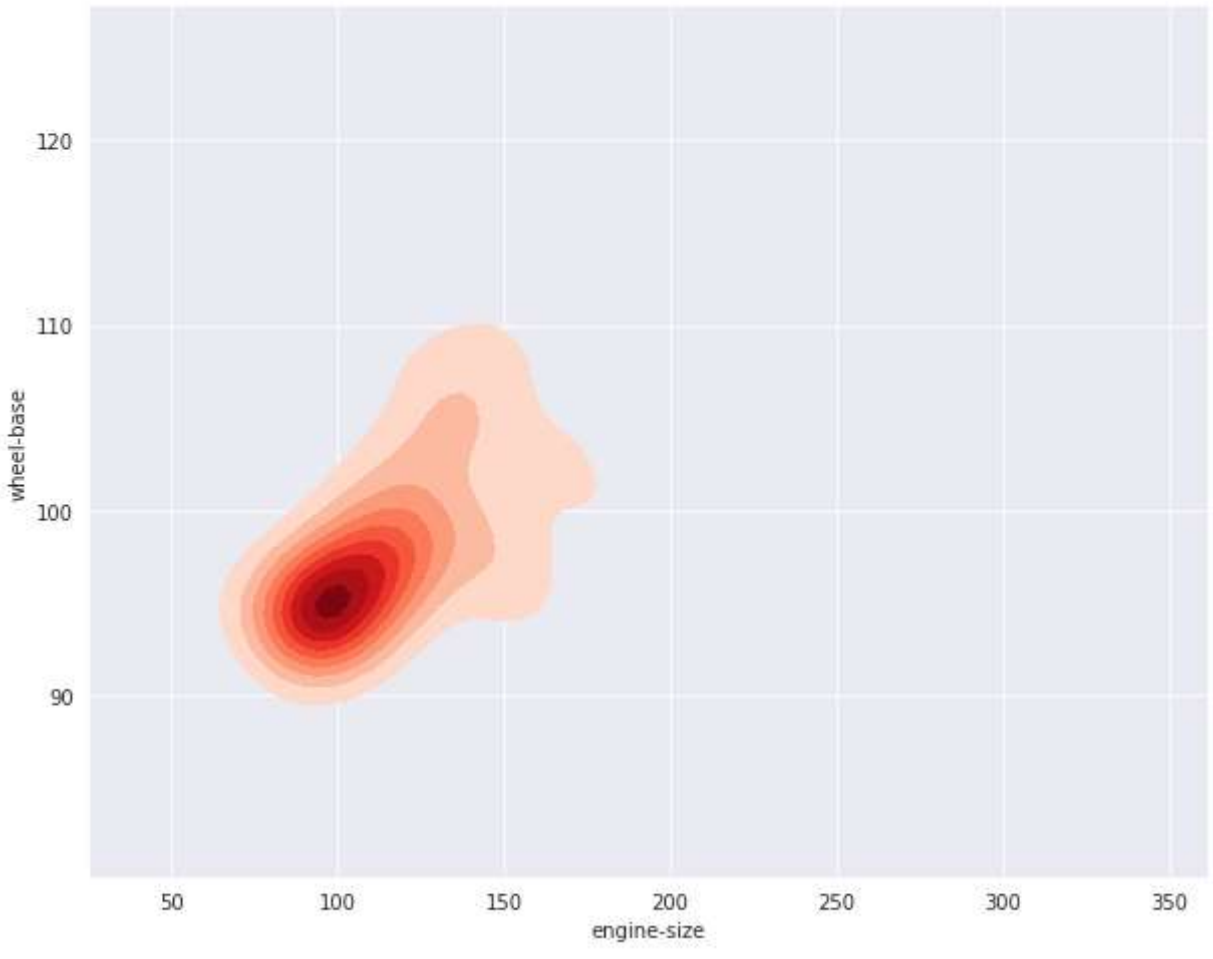
Out[57]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7bf6310a4a20>
```



In [58]:
```
plt.figure(figsize=(10,8))
sns.kdeplot(auto["city-mpg"],auto["engine-size"],shade=True,cmap="Reds", shade_lowest=False)
plt.show()
```

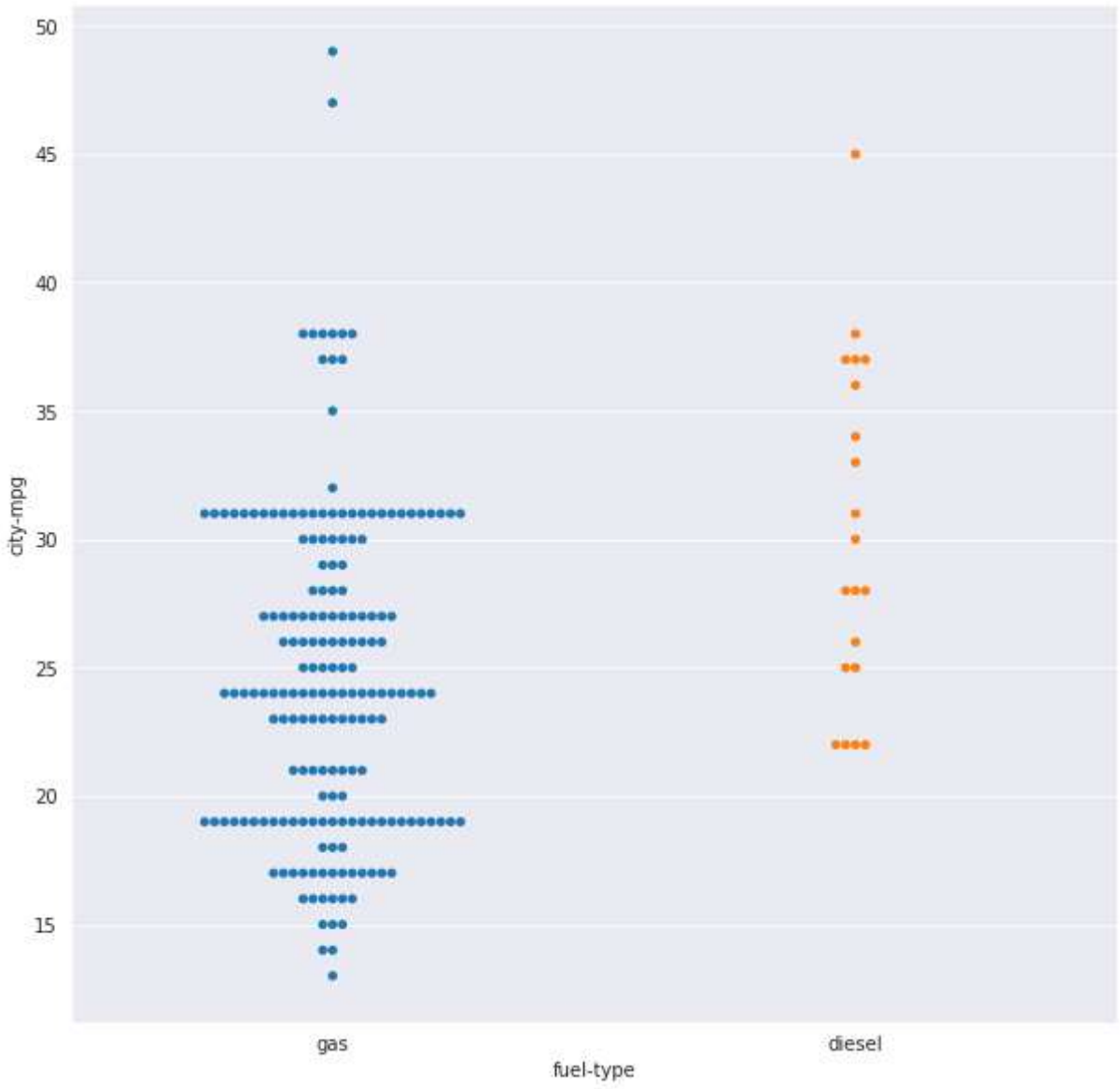


In [59]:
```
plt.figure(figsize=(10,8))
sns.kdeplot(auto["engine-size"],auto["wheel-base"],cmap="Reds", shade=True, shade_lowest=False)
plt.show()
```

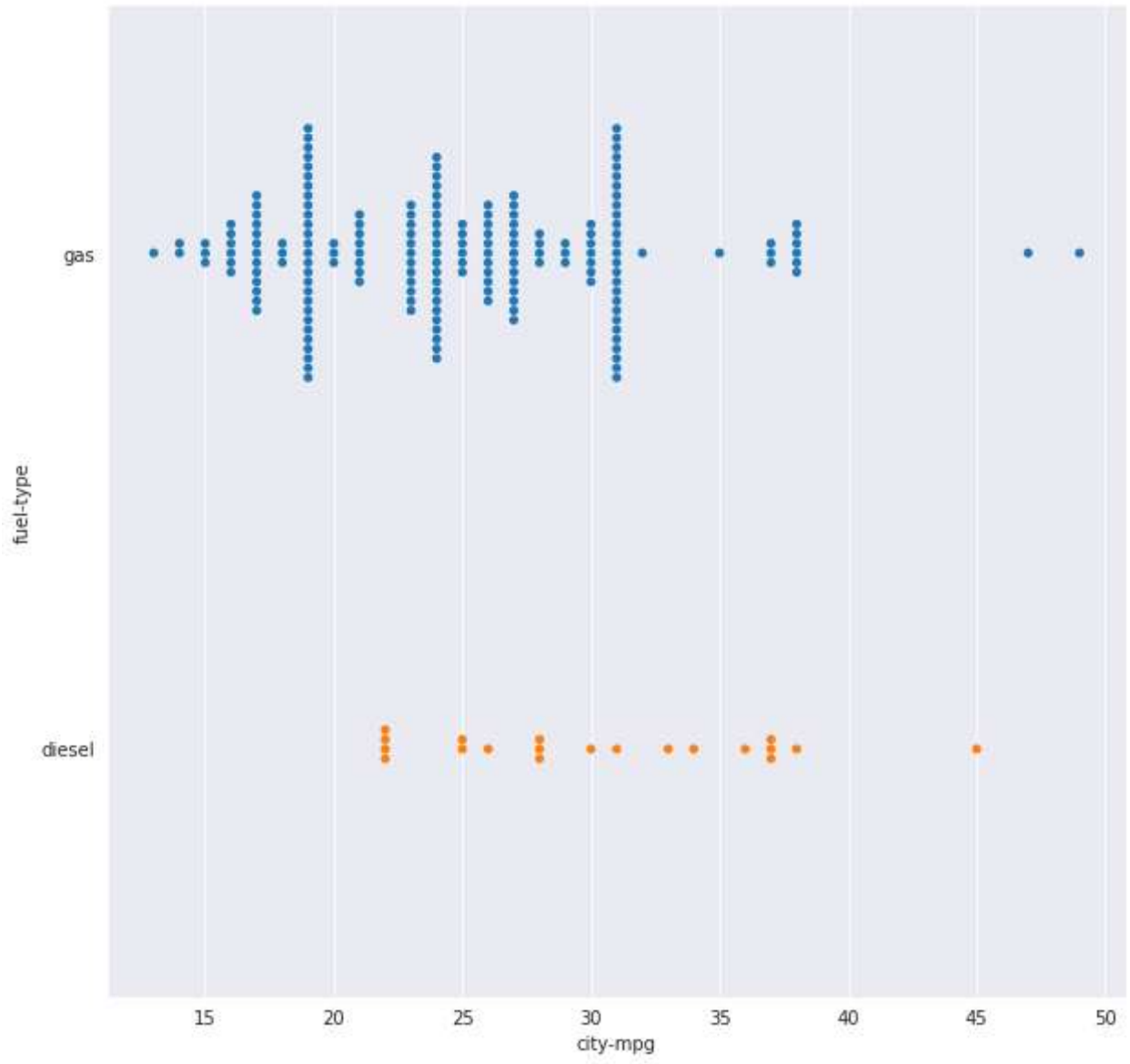


11. Swarm Plot

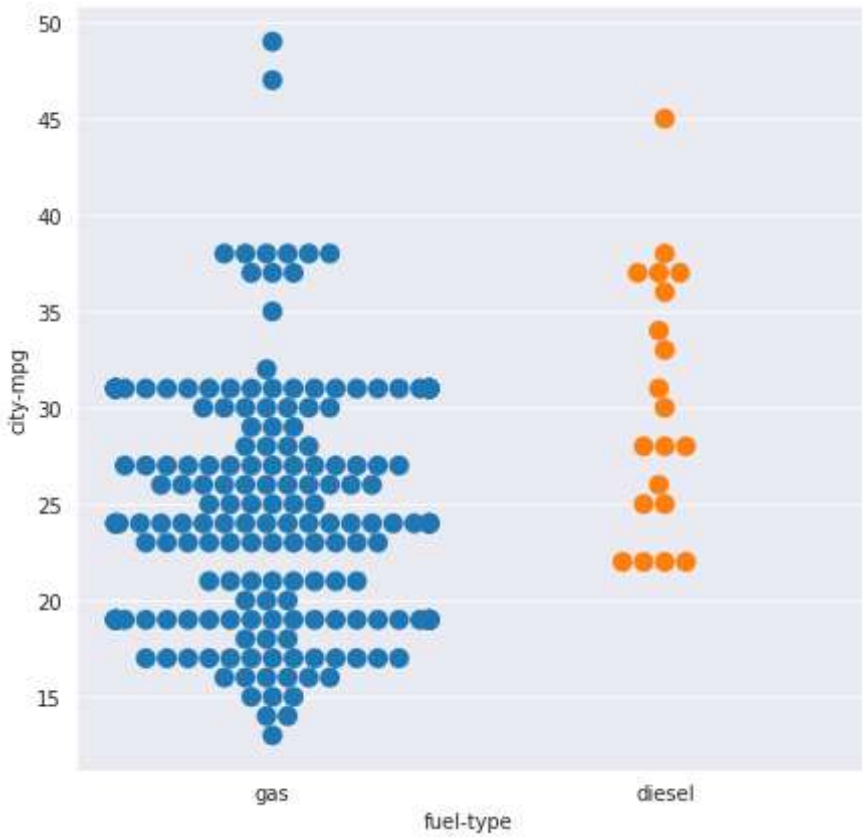
In [60]:
```
#Simple Swarm plot
plt.figure(figsize=(10,10))
sns.swarmplot(auto['fuel-type'], auto['city-mpg'])
plt.show()
```



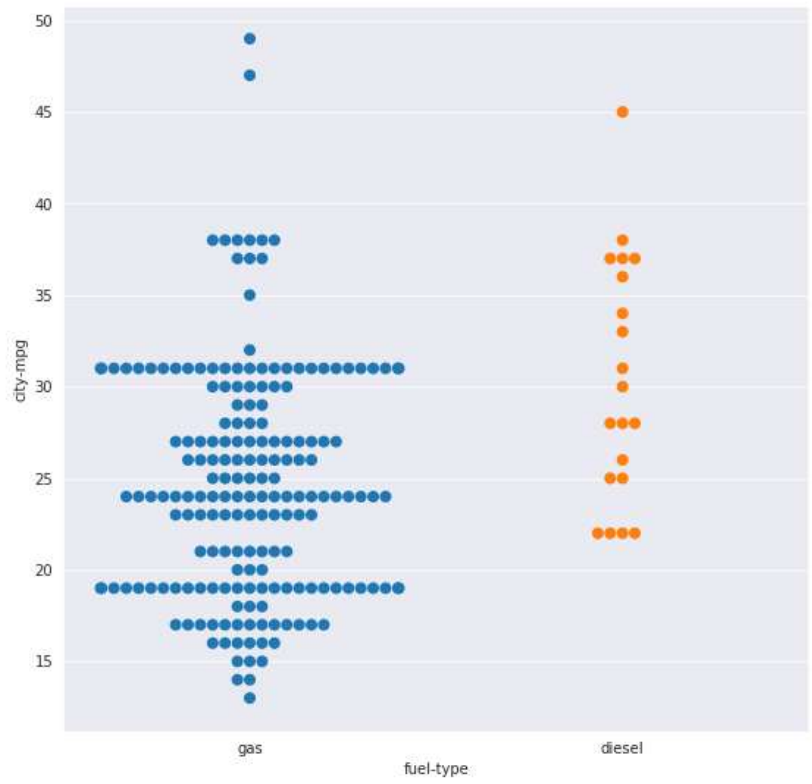
In [61]:
```
# Draw horizontal swarm plot
plt.figure(figsize=(10,10))
sns.swarmplot(auto['city-mpg'], auto['fuel-type'])
plt.show()
```

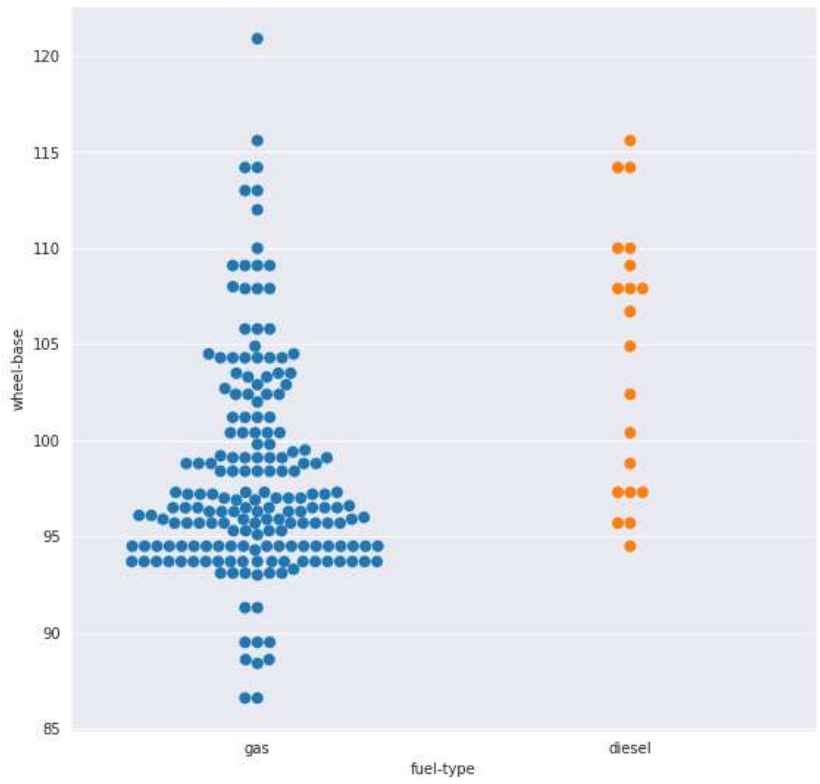


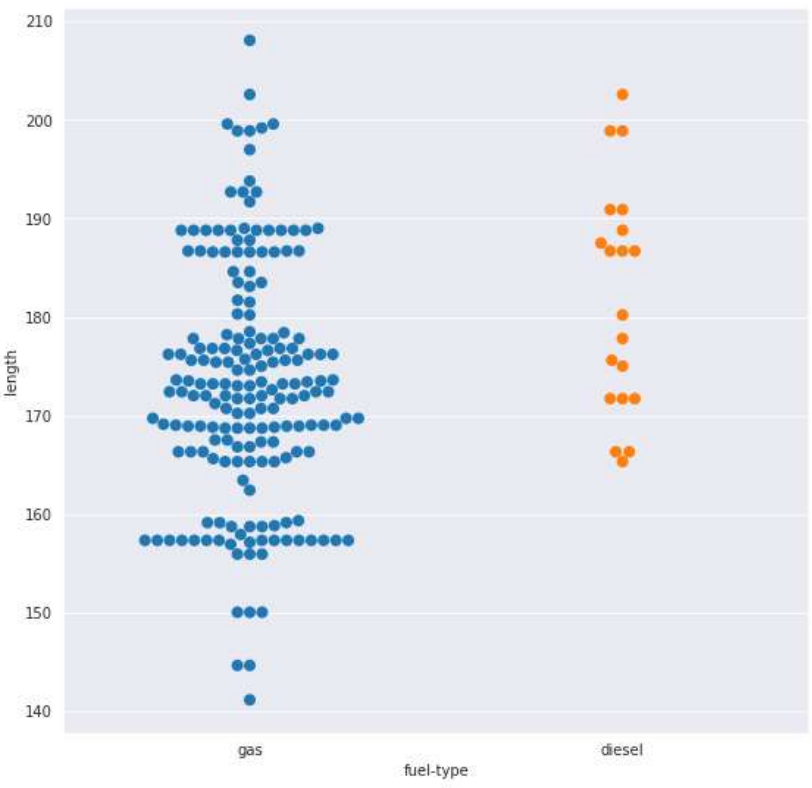
In [62]:
```
# Defining the size of the plots
plt.figure(figsize=(7,7))
sns.swarmplot(auto['fuel-type'], auto['city-mpg'], size=10)
plt.show()
```

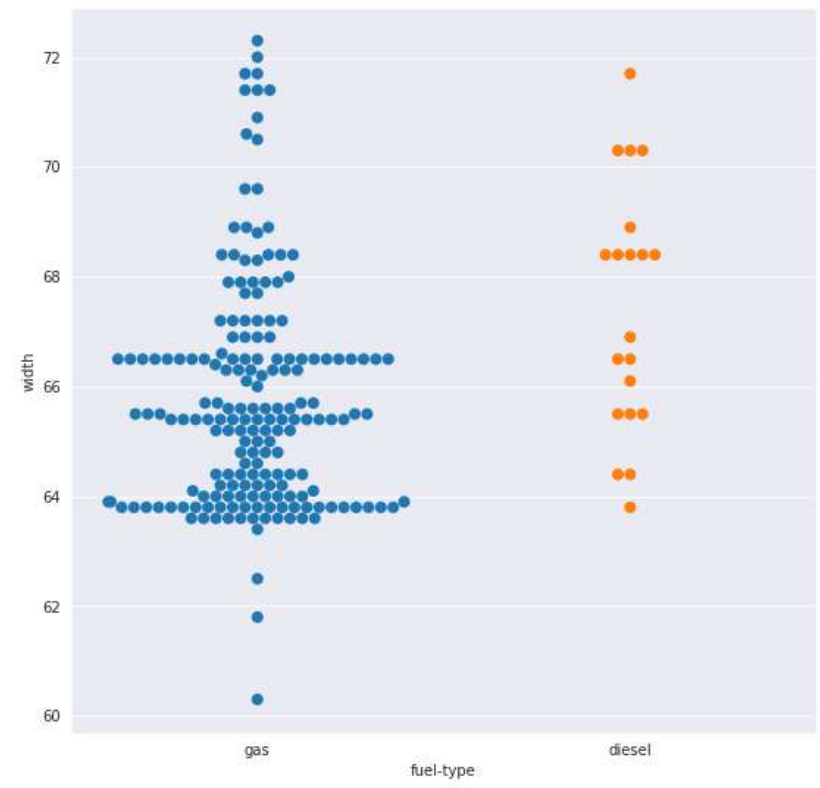


In [63]:
```
# Displaying multiple swarmplots using subplot function.
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.swarmplot(auto['fuel-type'], auto['city-mpg'], ax = axes[0,0], size=8)
sns.swarmplot(auto['fuel-type'] , auto['wheel-base'] ,ax = axes[0,1], size=8)
sns.swarmplot(auto['fuel-type'] ,auto['length'] , ax = axes[1,0] , size=8)
sns.swarmplot(auto['fuel-type'] , auto['width'] , ax = axes[1,1] , size=8)
plt.show()
```

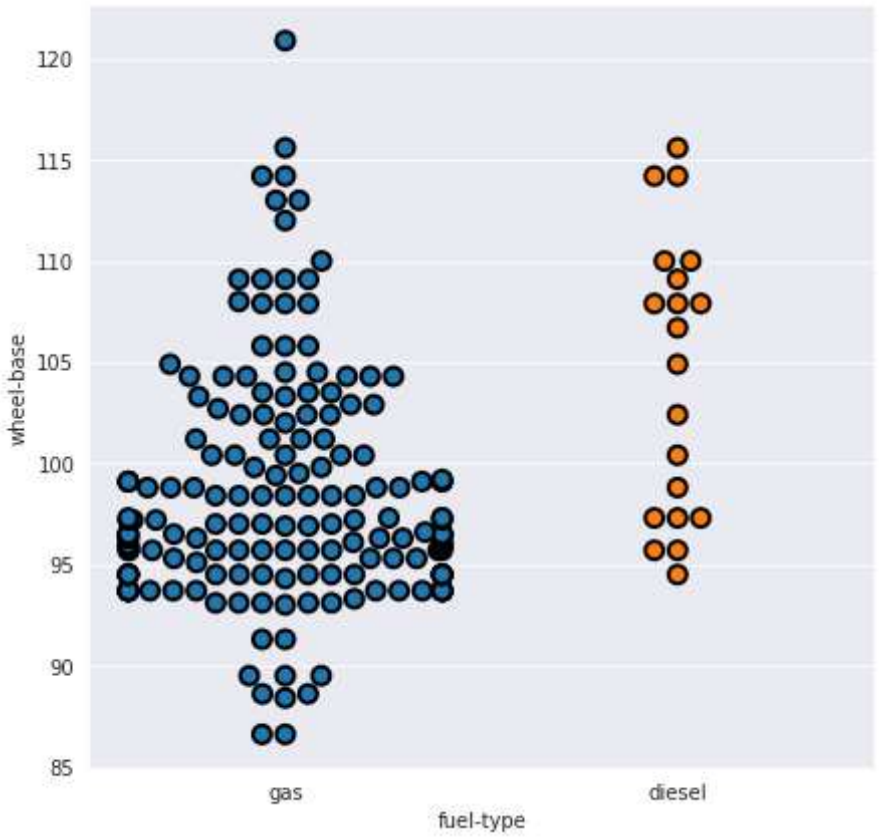






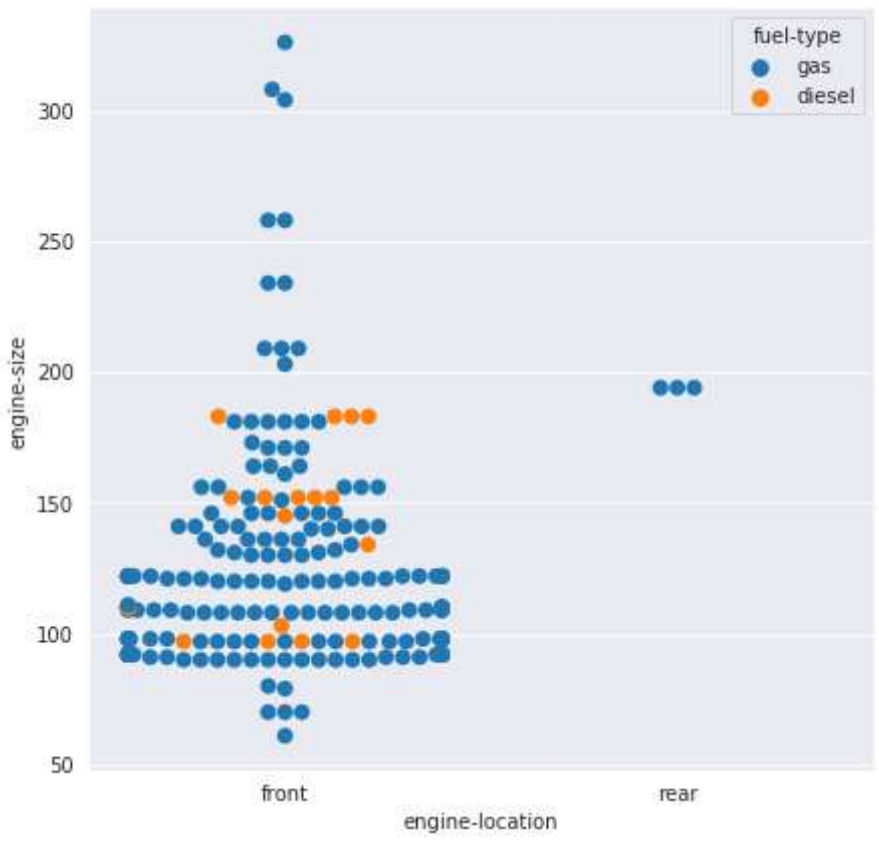


In [64]:
```
#Changing edge color , size and linewidth of data points
plt.figure(figsize=(7,7))
sns.swarmplot(x= "fuel-type", y = "wheel-base", size = 9 , linewidth= 2 , edgecolor="black" , data =auto)
plt.show()
```

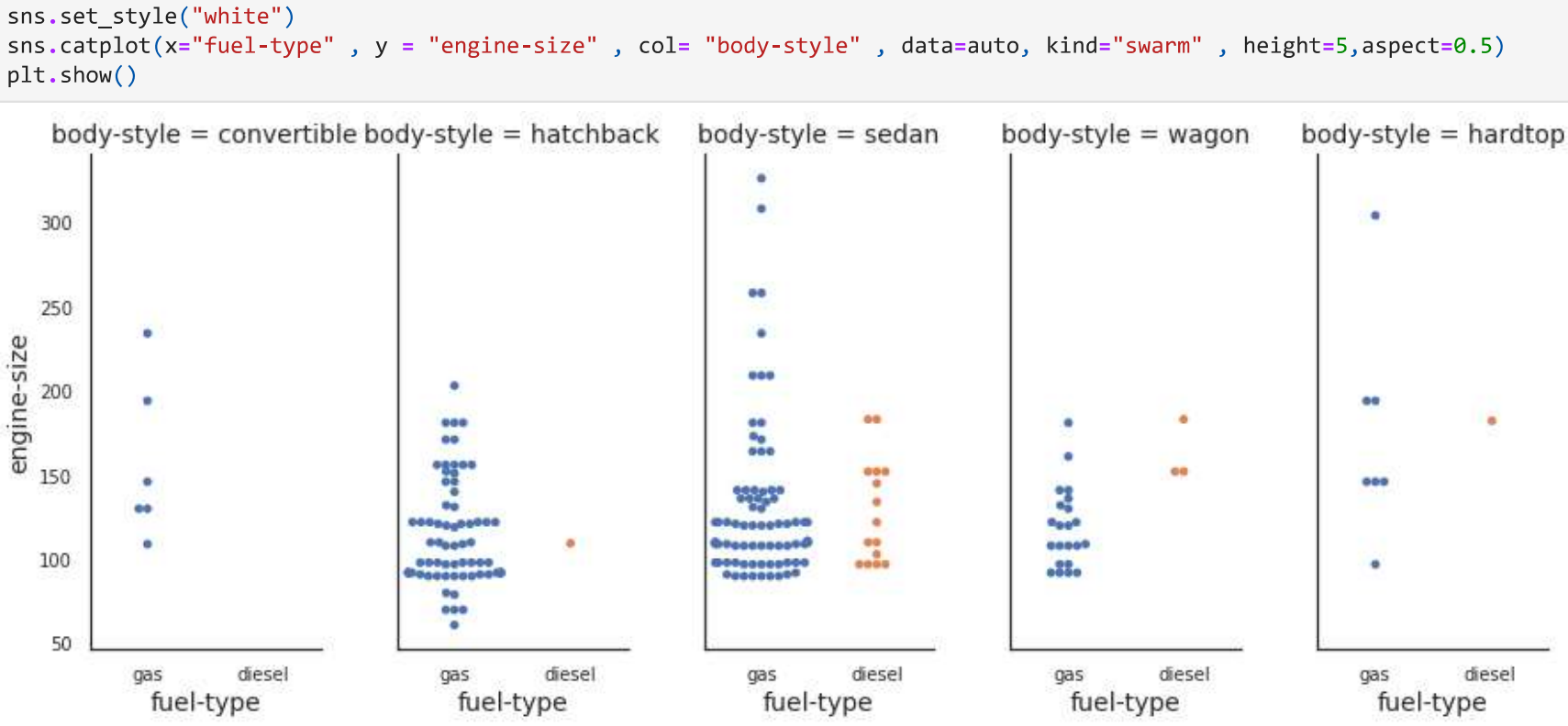


In [65]:
```
# Show groups with different colors using "hue"
plt.figure(figsize=(7,7))
sns.swarmplot(x= "engine-location", y = "engine-size", hue="fuel-type", size = 8 , data = auto)
```

Out[65]: <matplotlib.axes._subplots.AxesSubplot at 0x7bf630df8ac8>



In [66]:
```
# Facet along the columns to show a categorical variable using "col" parameter
sns.set(rc={'xtick.labelsize':10,'ytick.labelsize':10,'axes.labelsize':14})
sns.set_style("white")
sns.catplot(x="fuel-type" , y = "engine-size" , col= "body-style" , data=auto, kind="swarm" , height=5,aspect=0.5)
plt.show()
```

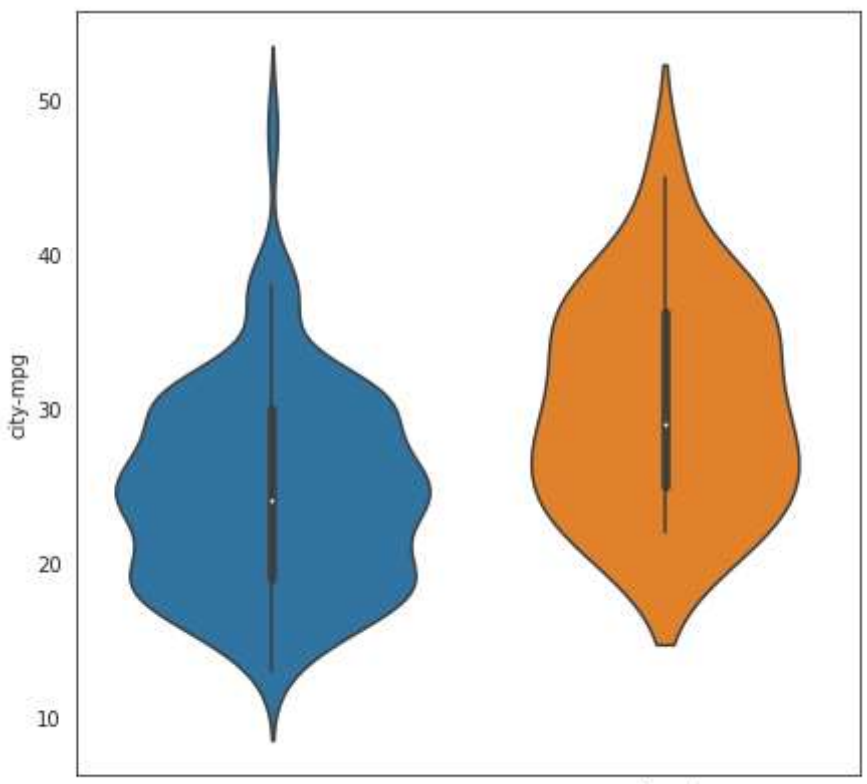


12. Violin Plot

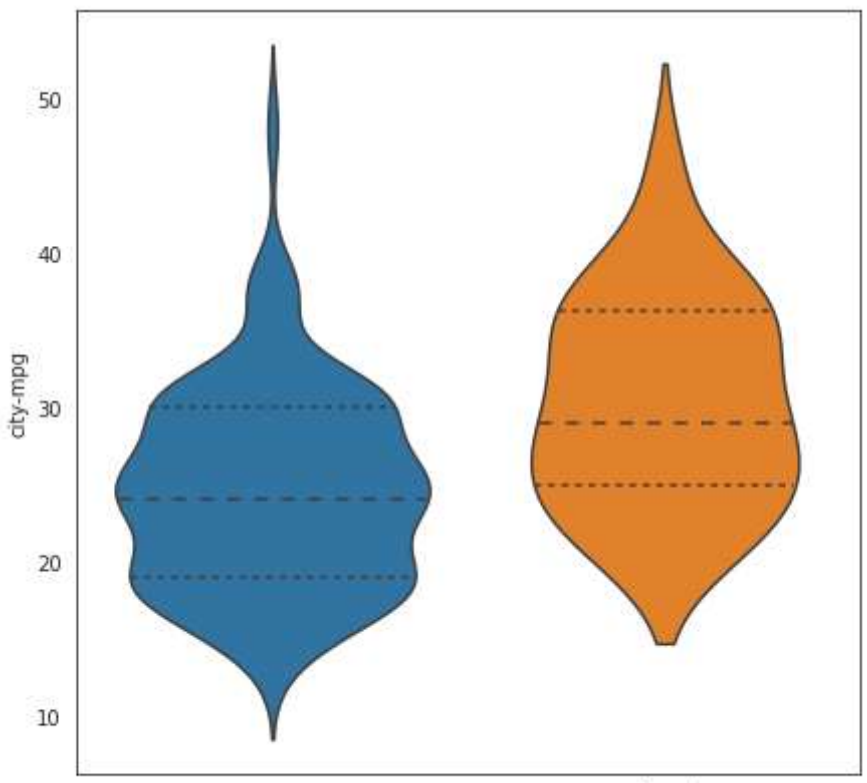
Violinplots summarize numeric data over a set of categories.Violin plots are similar to box plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.

In [67]:
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

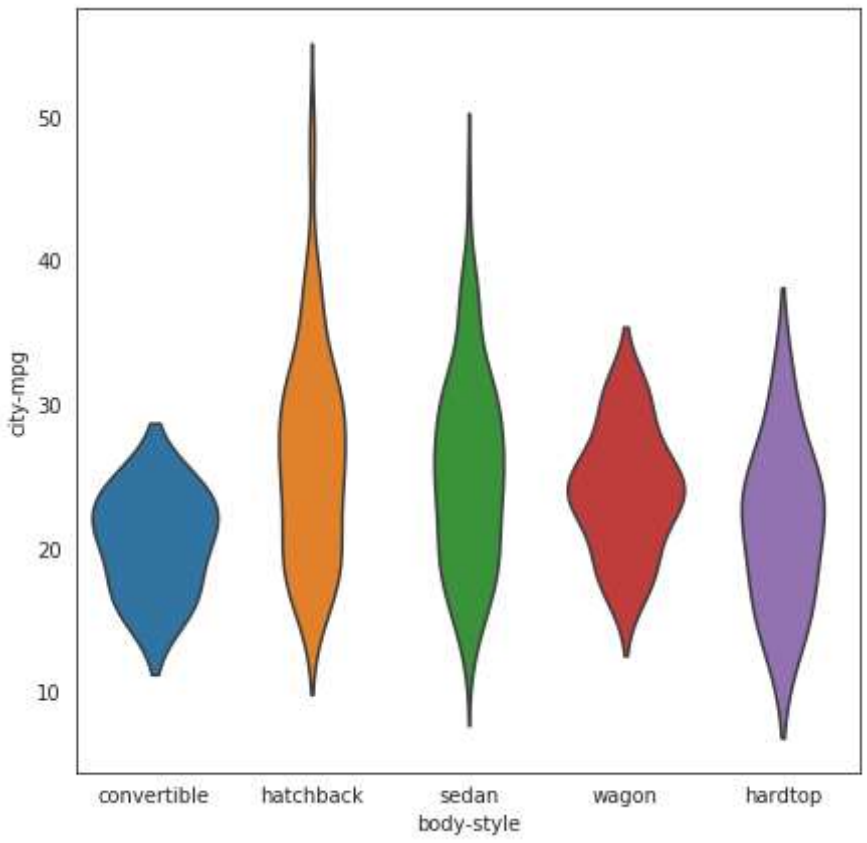
In [68]:
```
# Simple Violin Plot
plt.figure(figsize=(7,7))
sns.violinplot(auto['fuel-type'], auto['city-mpg'])
plt.show()
```



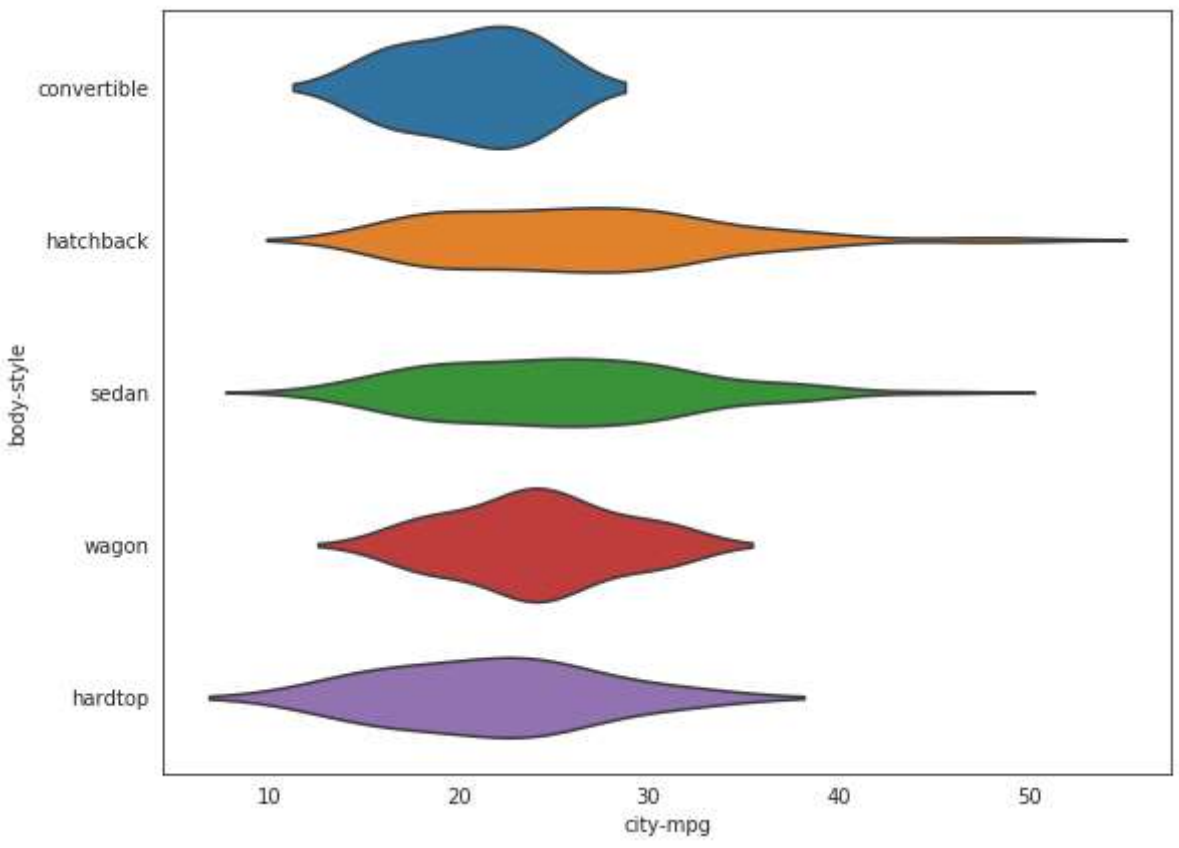
In [69]:
```
# Drawing the quartiles as horizontal lines instead of a mini-box using (inner="quartile")
plt.figure(figsize=(7,7))
sns.violinplot(x="fuel-type" , y = "city-mpg" ,data=auto , inner="quartile")
plt.show()
```



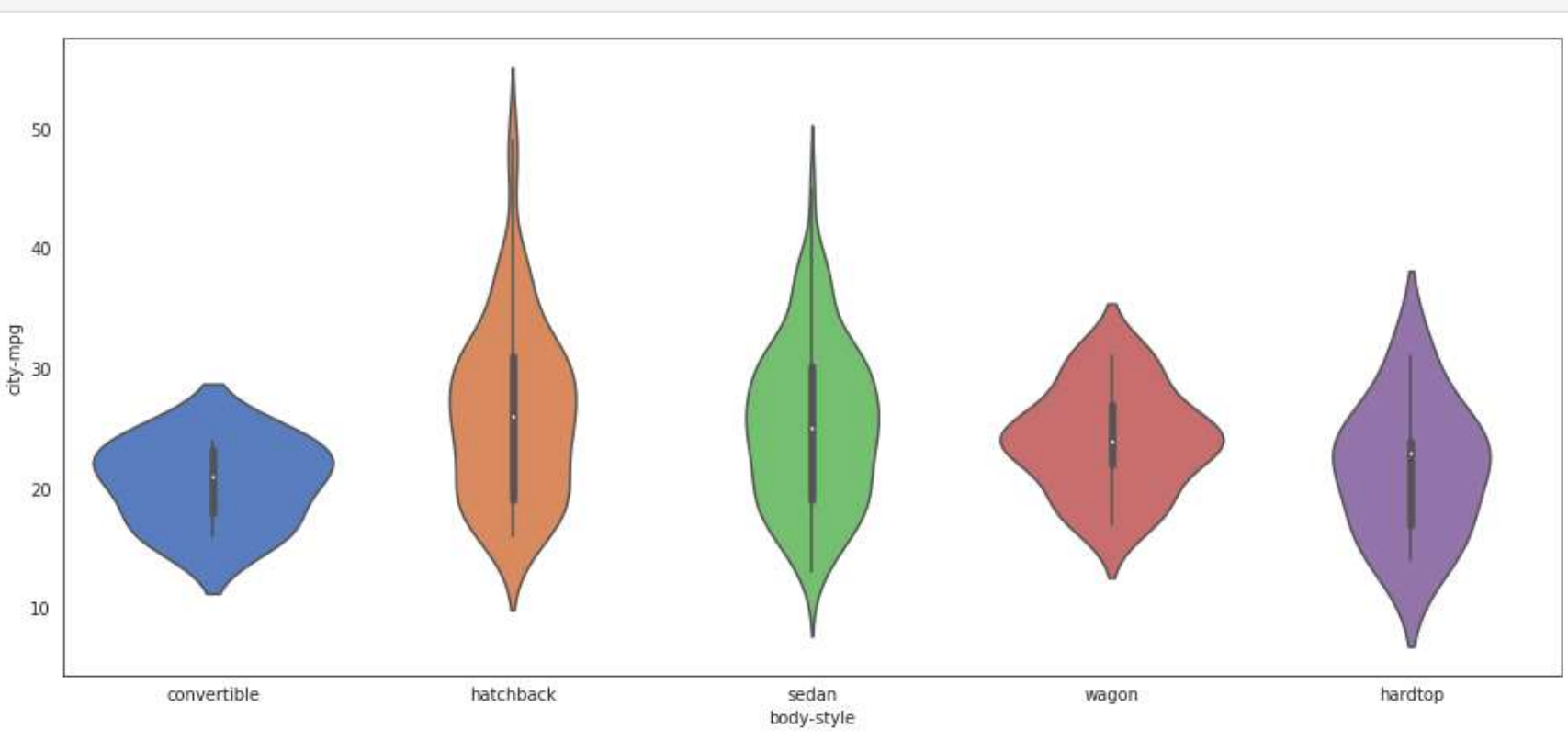
In [70]:
```
# Remove interior section of the Violin plot
plt.figure(figsize=(7,7))
sns.violinplot(x="body-style" , y = "city-mpg" ,data=auto , inner=None)
plt.show()
```



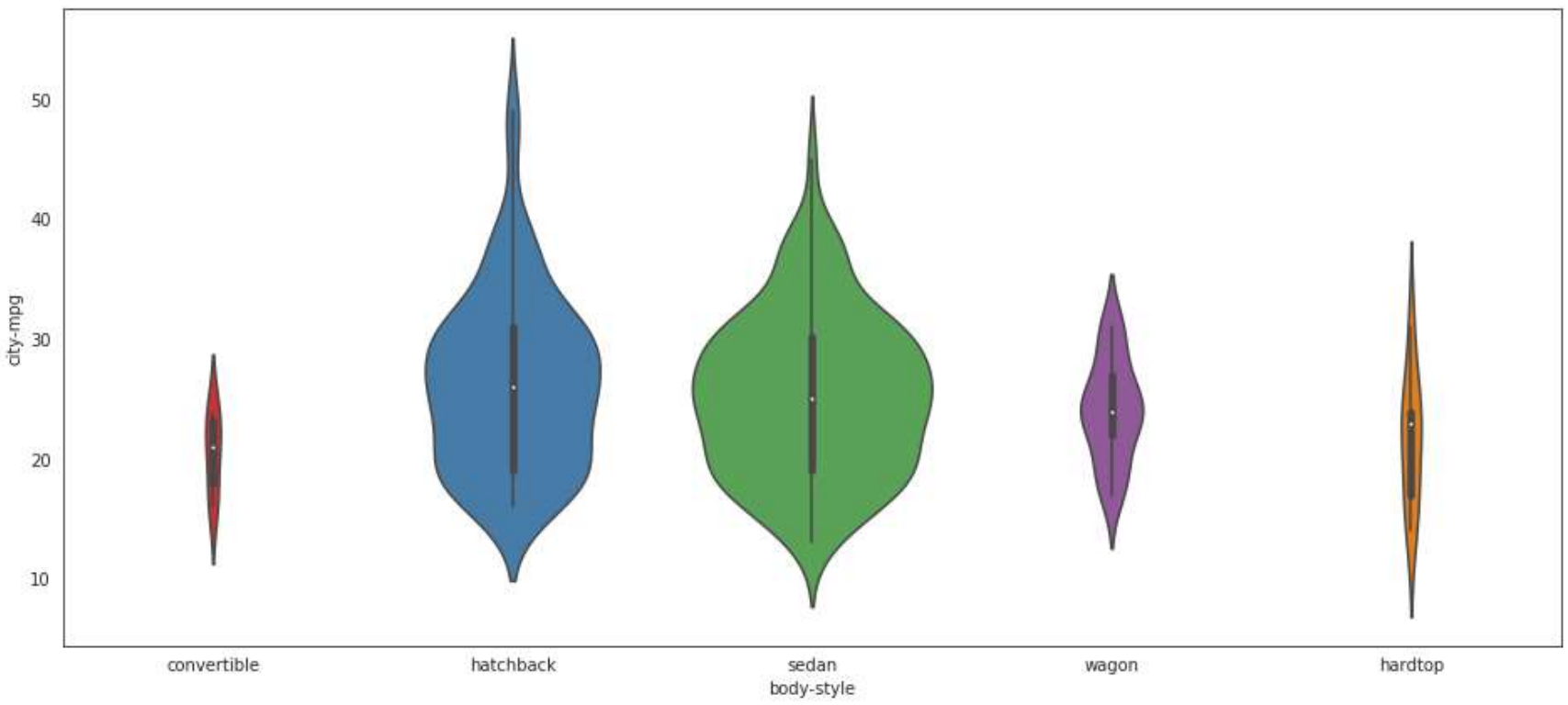
In [71]:
```
# horizontal violin plot
plt.figure(figsize=(9,7))
sns.violinplot(y="body-style" , x = "city-mpg" ,data=auto , inner=None)
plt.show()
```



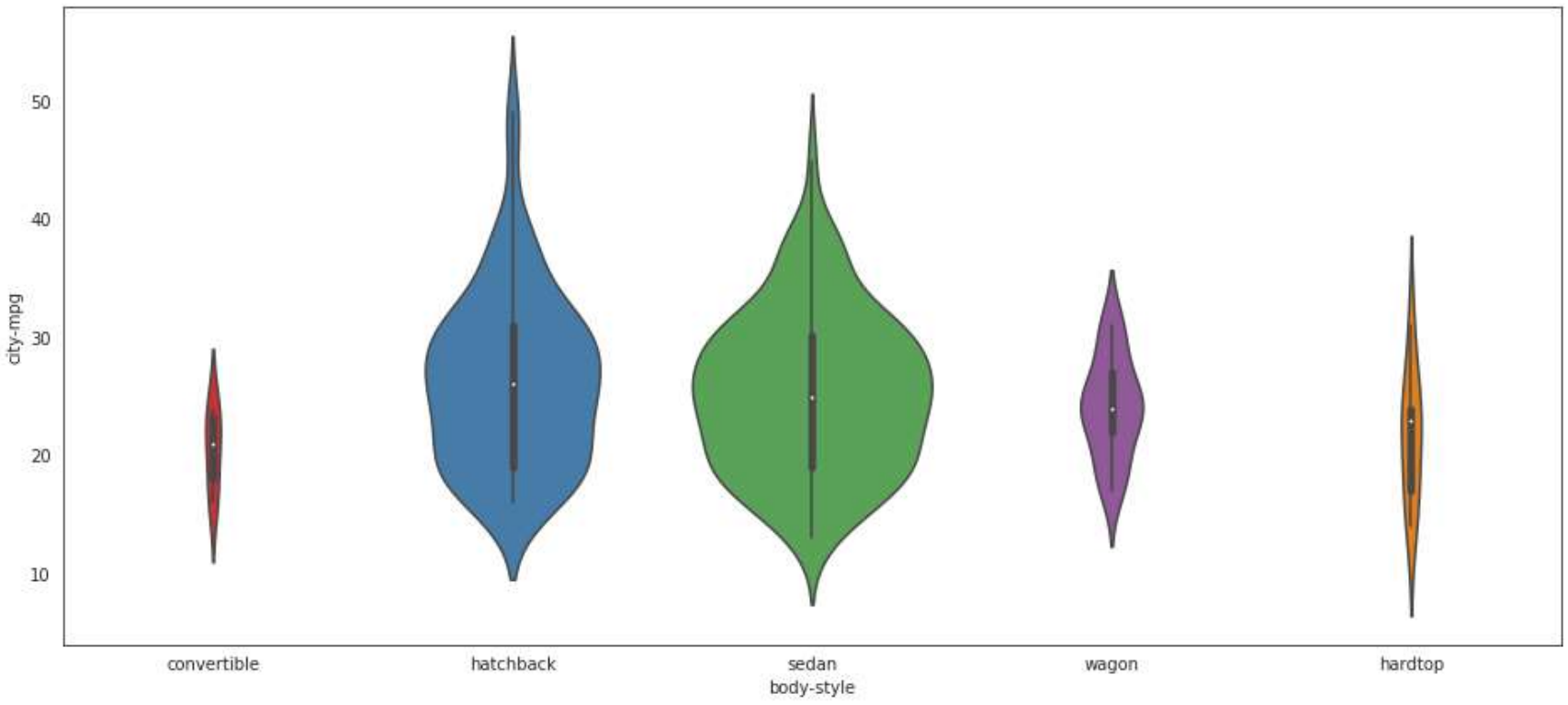
In [72]:
```
# Muted palette
plt.figure(figsize=(16,7))
sns.violinplot(x=auto["body-style"] , y = auto["city-mpg"] , palette="muted")
plt.show()
```



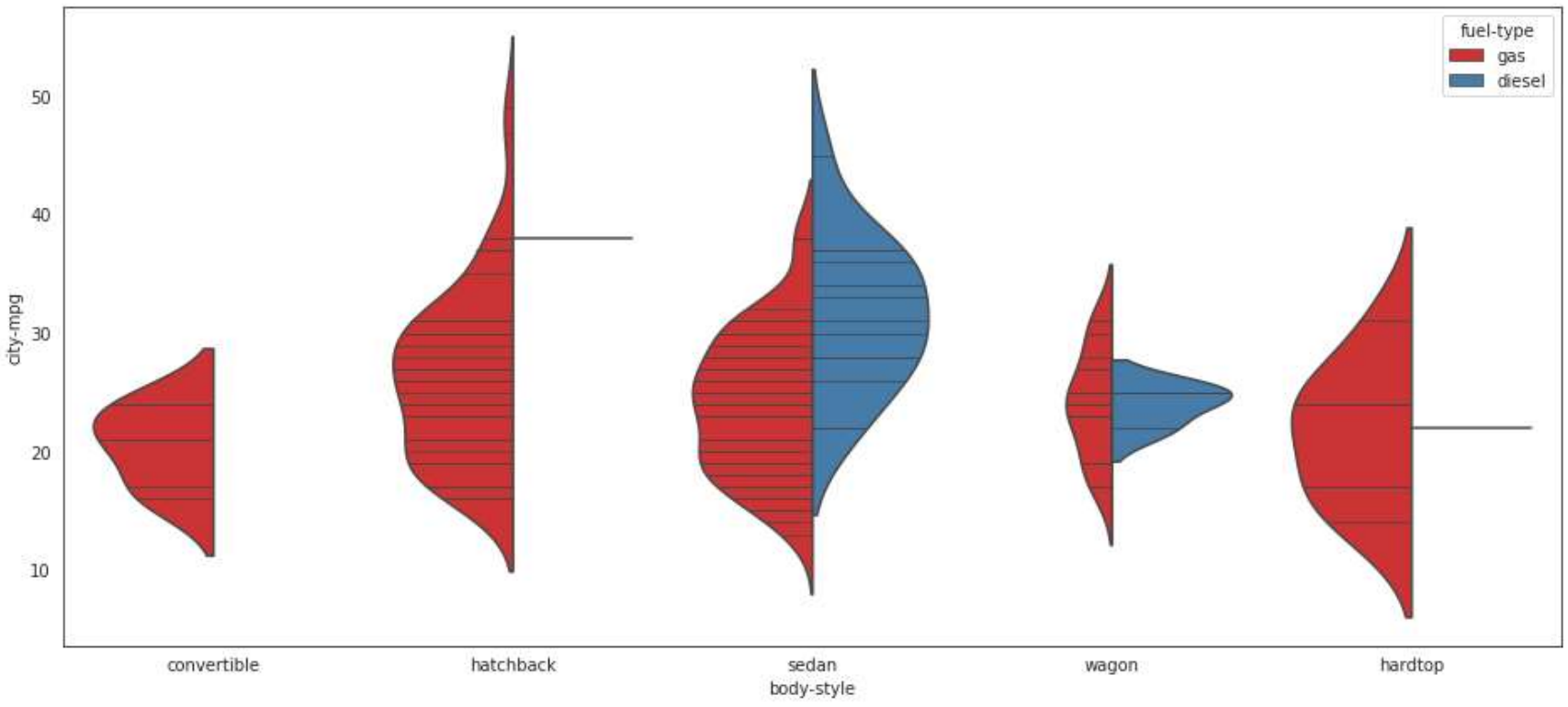
In [73]:
```
#Scale the density relative to the counts across all bins
plt.figure(figsize=(16,7))
sns.violinplot(x=auto["body-style"] , y = auto["city-mpg"] , palette="Set1" , scale="count")
plt.show()
```



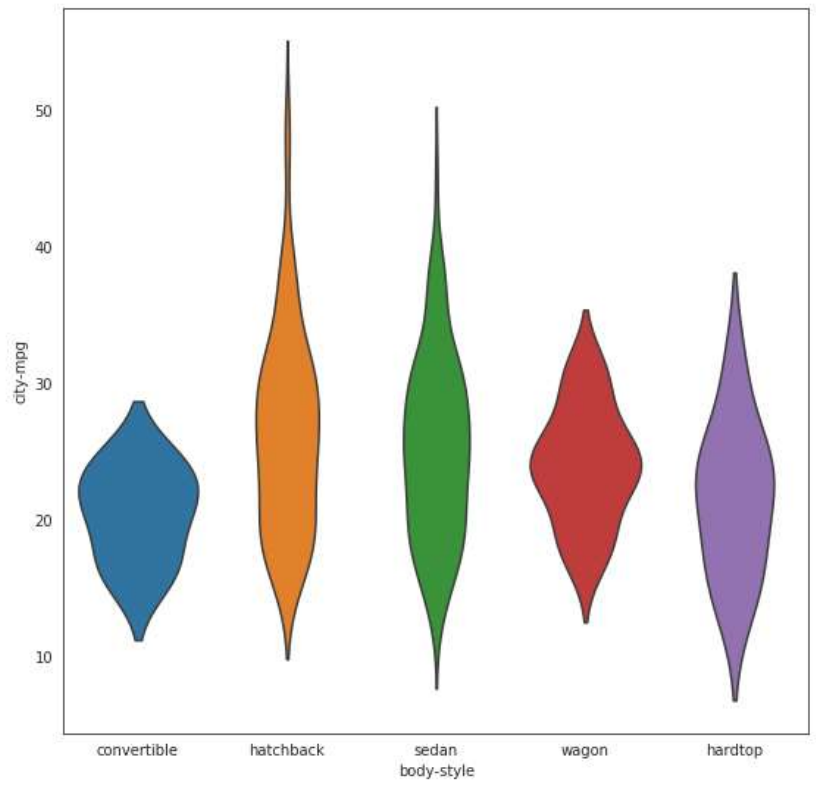
In [74]:
```
# Adjust the bandwidth of the KDE filtering parameter using "bw"
plt.figure(figsize=(16,7))
sns.violinplot(x=auto["body-style"] , y = auto["city-mpg"] , palette="Set1" , scale="count",bw='silverman')
plt.show()
```

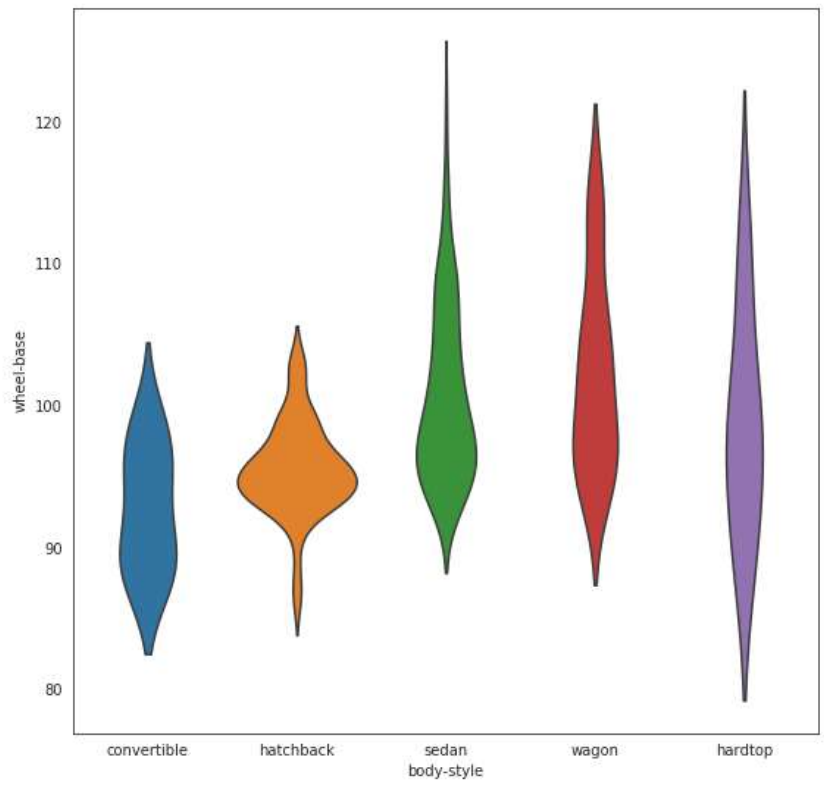


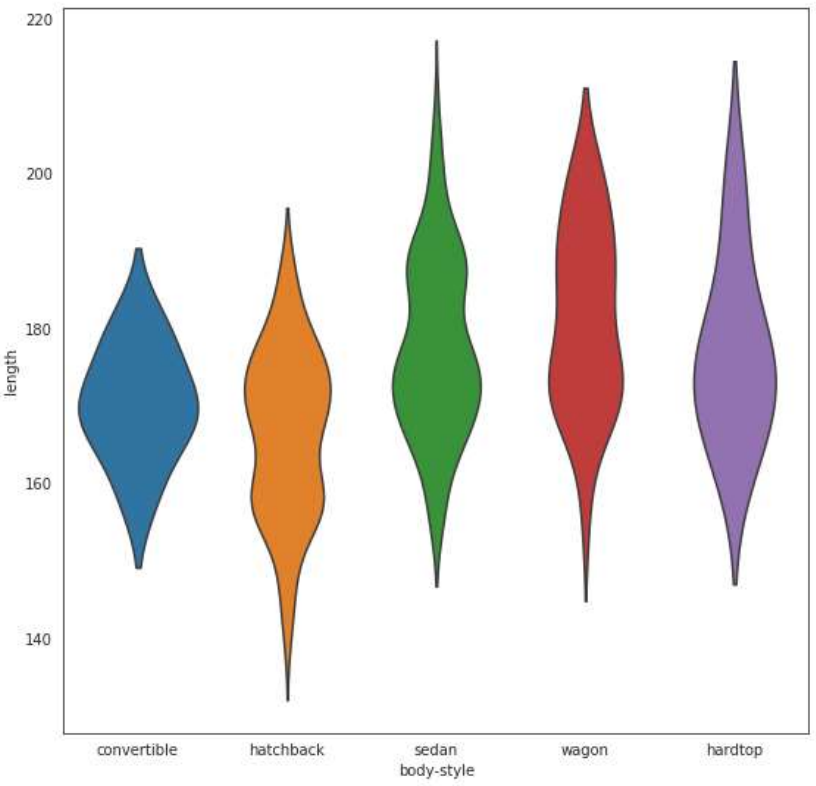
In [80]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=auto["body-style"] , y = auto["city-mpg"] , palette="Set1", hue= auto["fuel-type"],split=True ,inner="
plt.show()
```

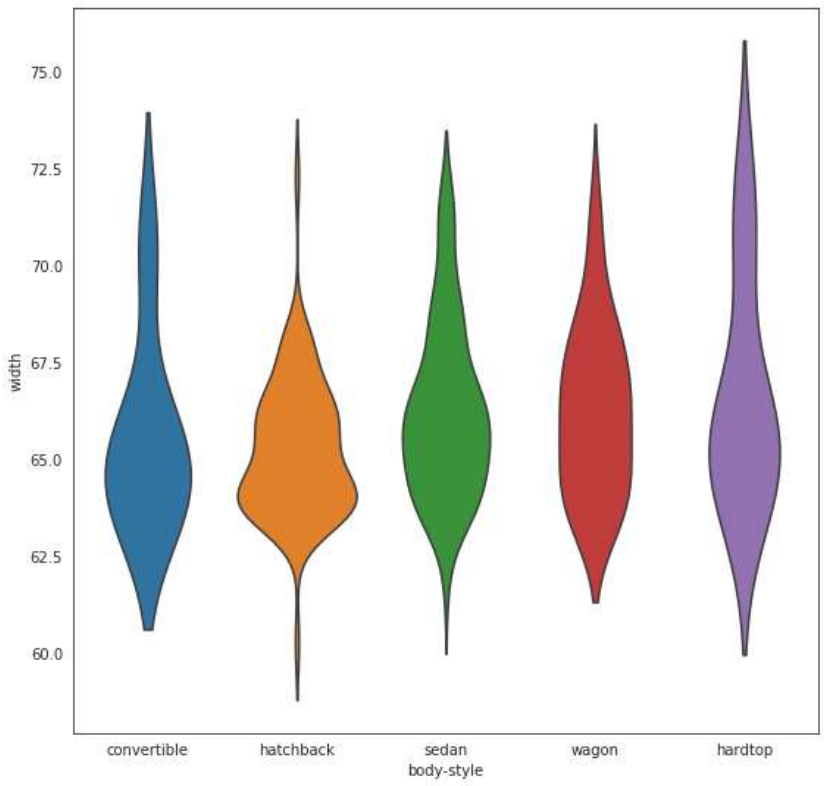


In [81]:
```
# Displaying multiple violin plots using subplot function.
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.violinplot(x="body-style" , y = "city-mpg" , ax = axes[0,0] ,data=auto , inner=None)
sns.violinplot(x="body-style" , y = "wheel-base" ,ax = axes[0,1] , data=auto , inner=None)
sns.violinplot(x="body-style" , y = "length" , ax = axes[1,0] , data=auto, inner=None)
sns.violinplot(x="body-style" , y = "width" , ax = axes[1,1] , data=auto, inner=None )
plt.show()
```









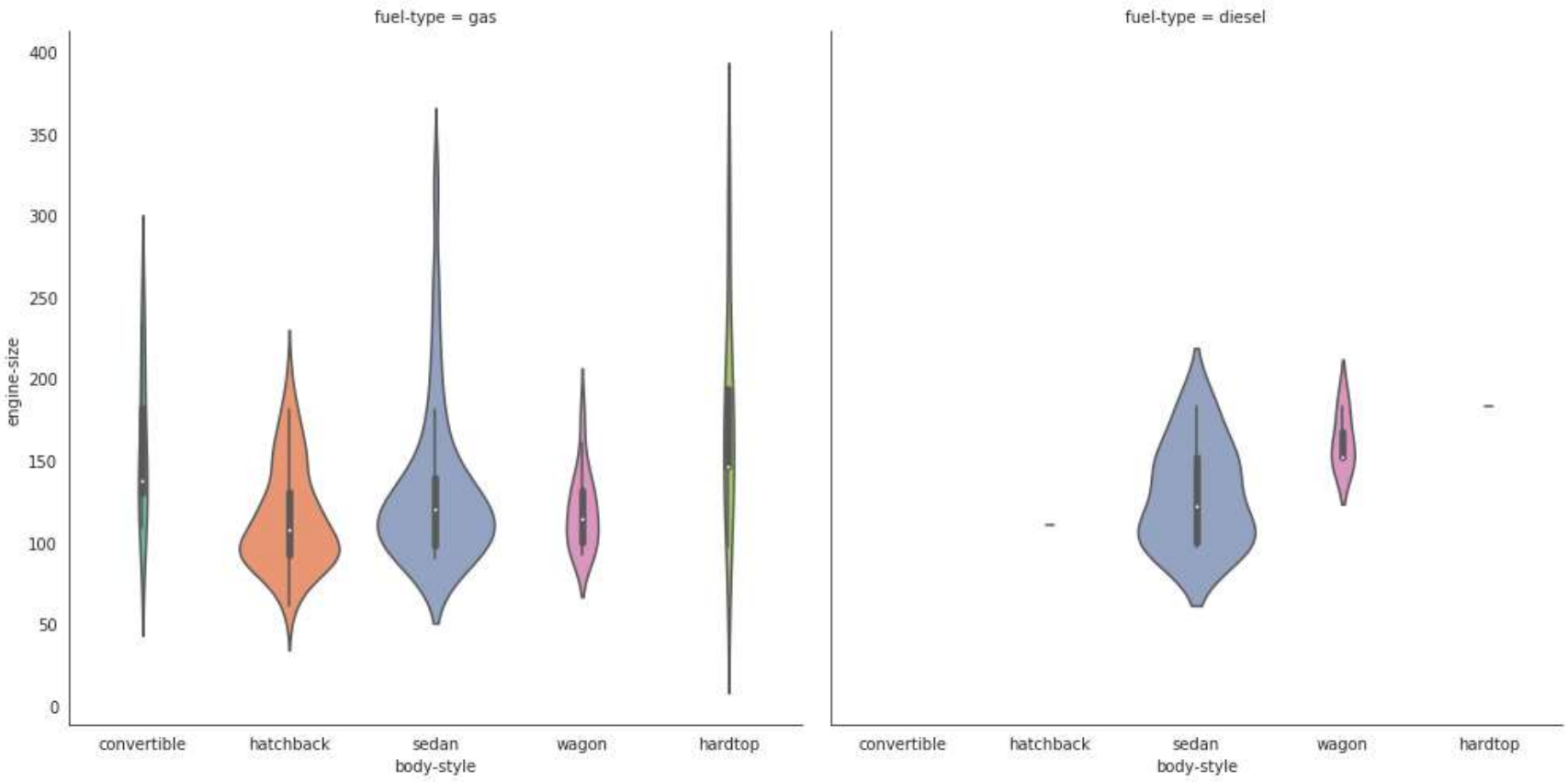
In [82]:
```
iris = pd.read_csv("/input/iris/Iris.csv")
iris.head()
```

Out[82]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

In [84]:
```
# Facet along the columns to show a categorical variable using "col" parameter
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type", kind="violin",palette="Set2" , height=  7,scale = "cou
plt.show()
```

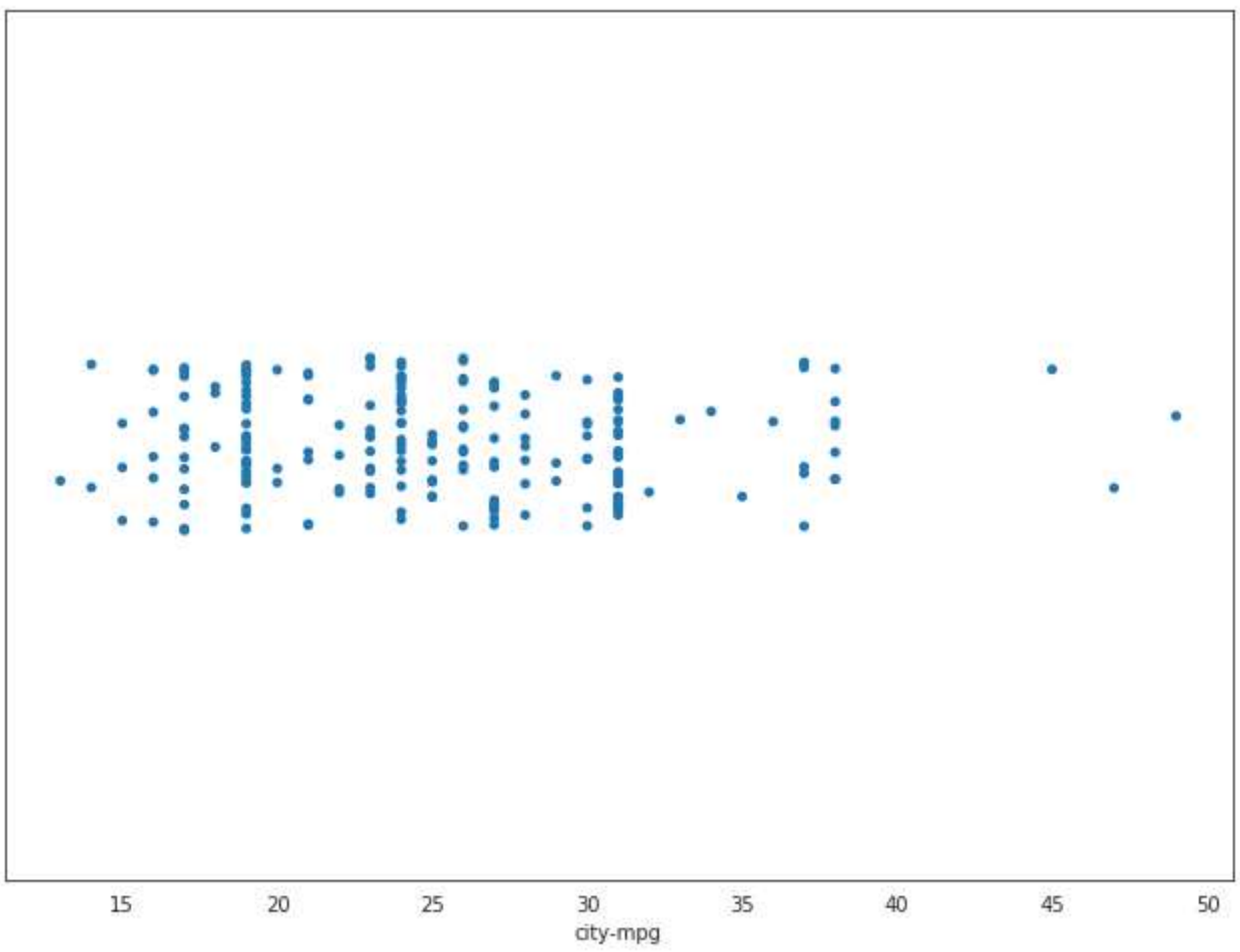
```
<Figure size 792x648 with 0 Axes>
```



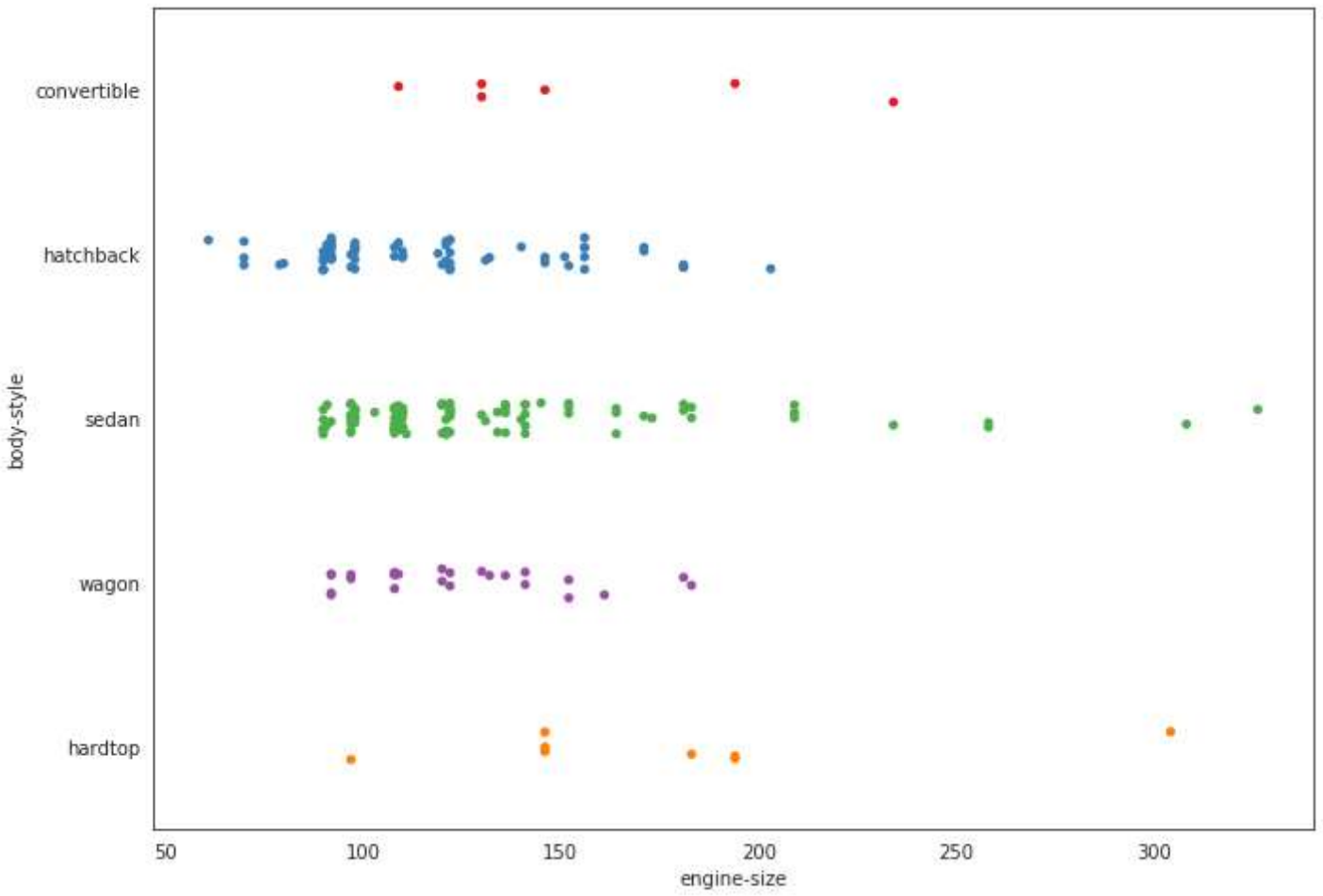
13.Strip Plot

Strip plot is a scatter plot where one of the variables is categorical.

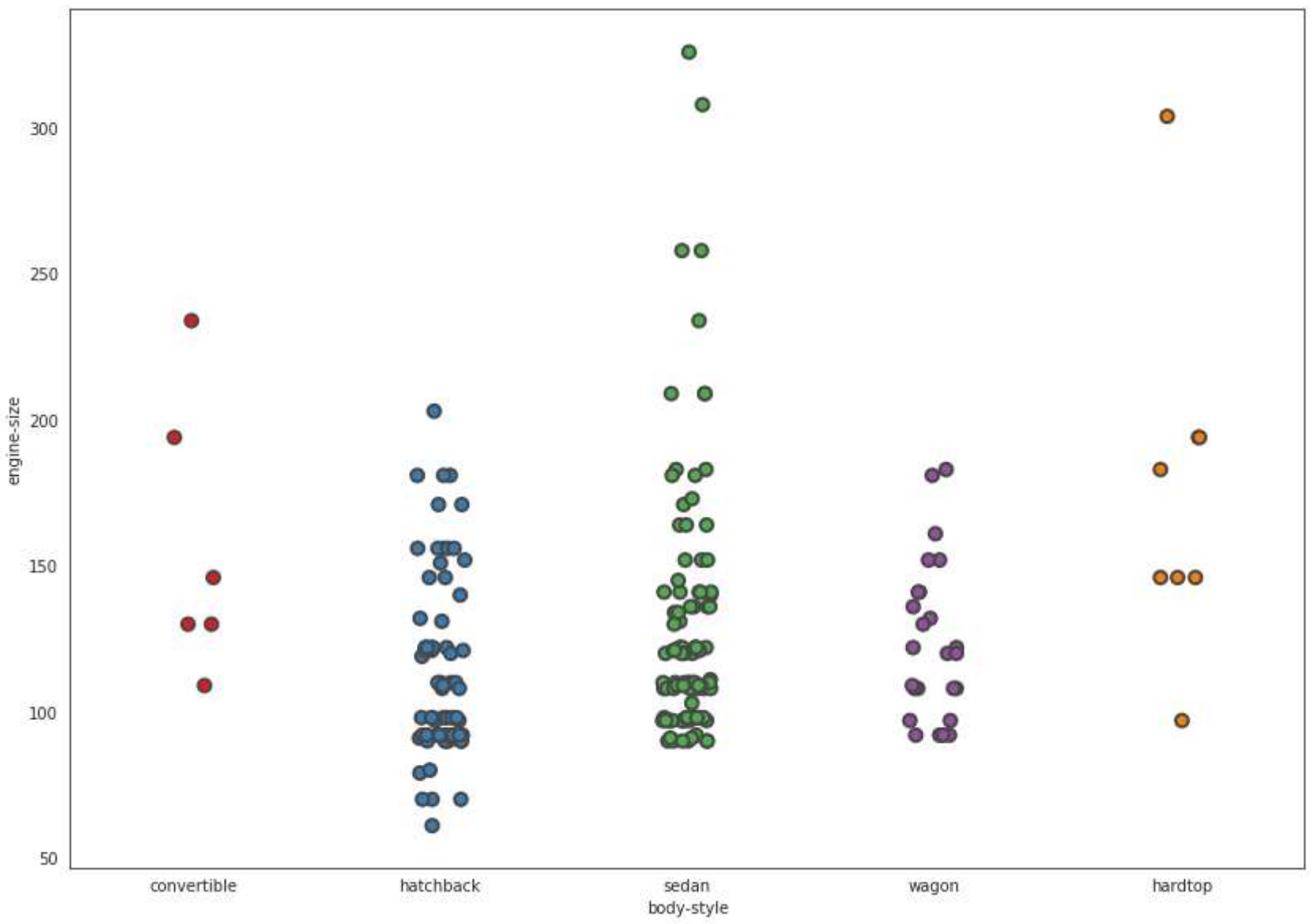
In [85]:
```
plt.figure(figsize=(11,8))
sns.stripplot(auto['city-mpg'])
plt.show()
```



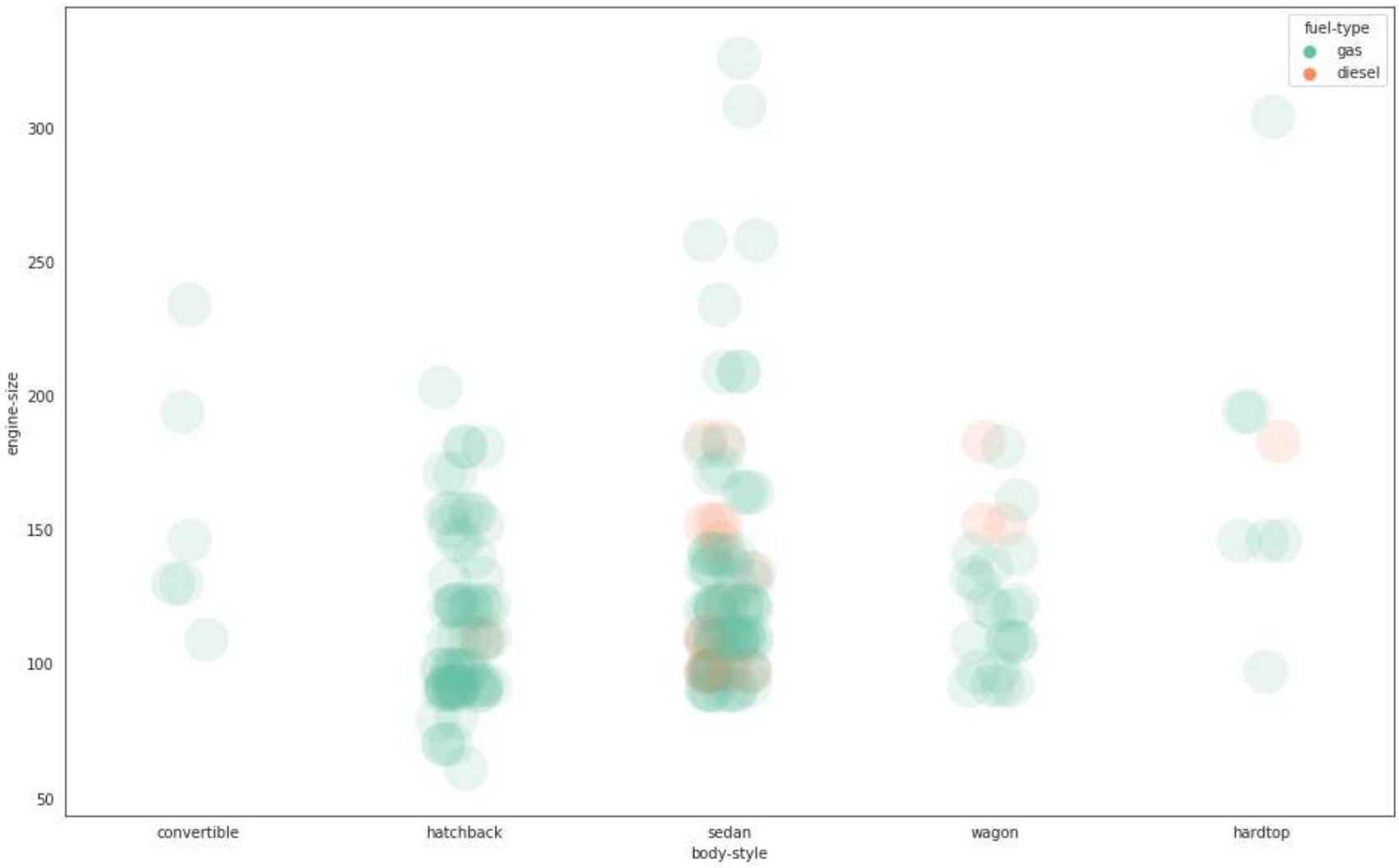
In [89]:
```
# Flip x and y inputs to make a horizontal strip plot
plt.figure(figsize=(11,8))
sns.stripplot(y=auto["body-style"] ,palette="Set1", x = auto["engine-size"] , jitter=True)
plt.show()
```



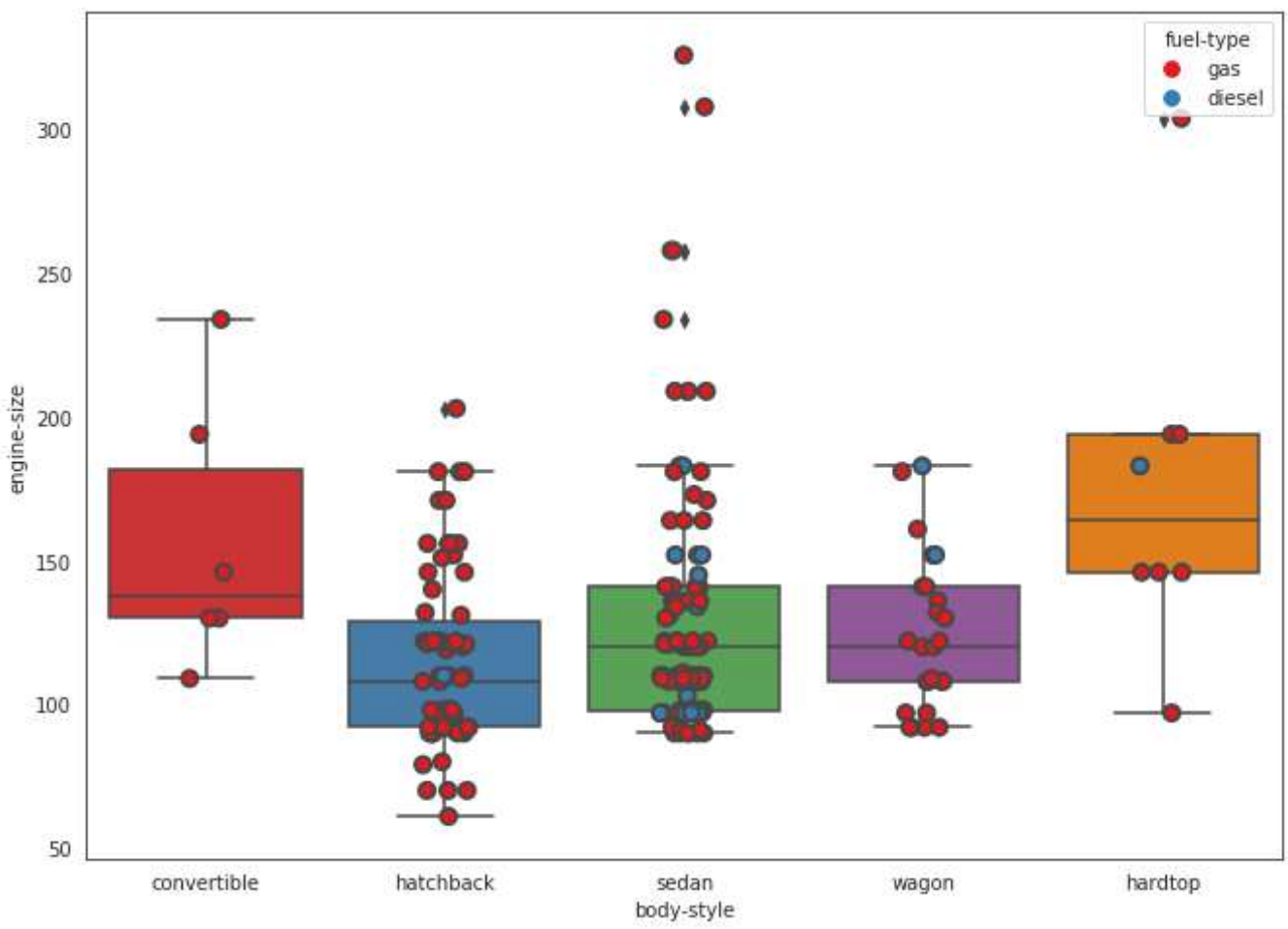
In [90]:
```
# Adjust the linewidth of the edges of the circles using "linewidth" parameter
# Adjust the size of the circles using the "size" parameter
plt.figure(figsize=(14,10))
sns.stripplot(x=auto["body-style"] ,palette="Set1", y = auto["engine-size"],linewidth=2,size=8)
plt.show()
```



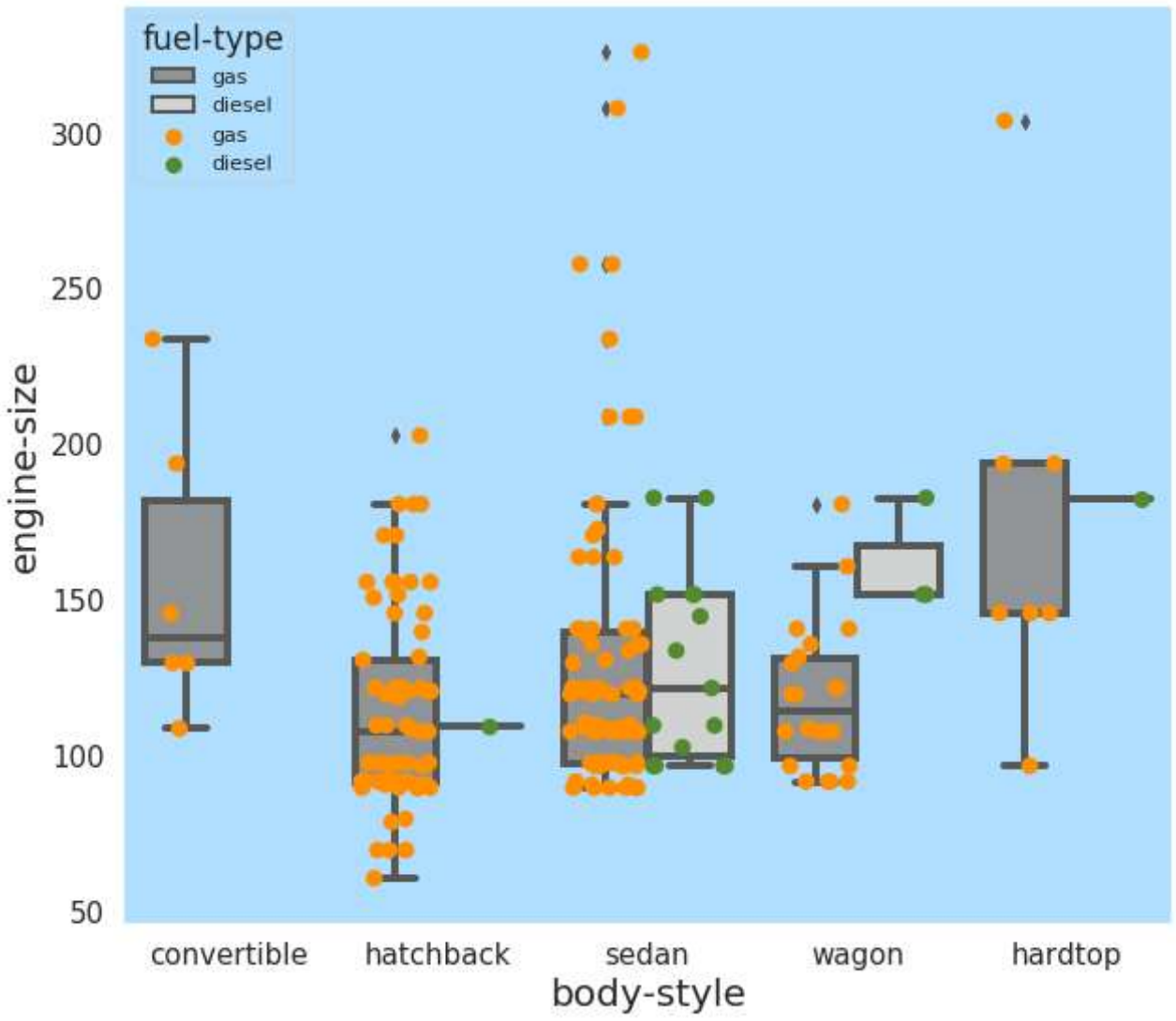
In [93]:
```
plt.figure(figsize=(16,10))
sns.stripplot(x=auto["body-style"] ,palette="Set2", y = auto["engine-size"],hue=auto["fuel-type"],marker = "o",size=30,
plt.show()
```



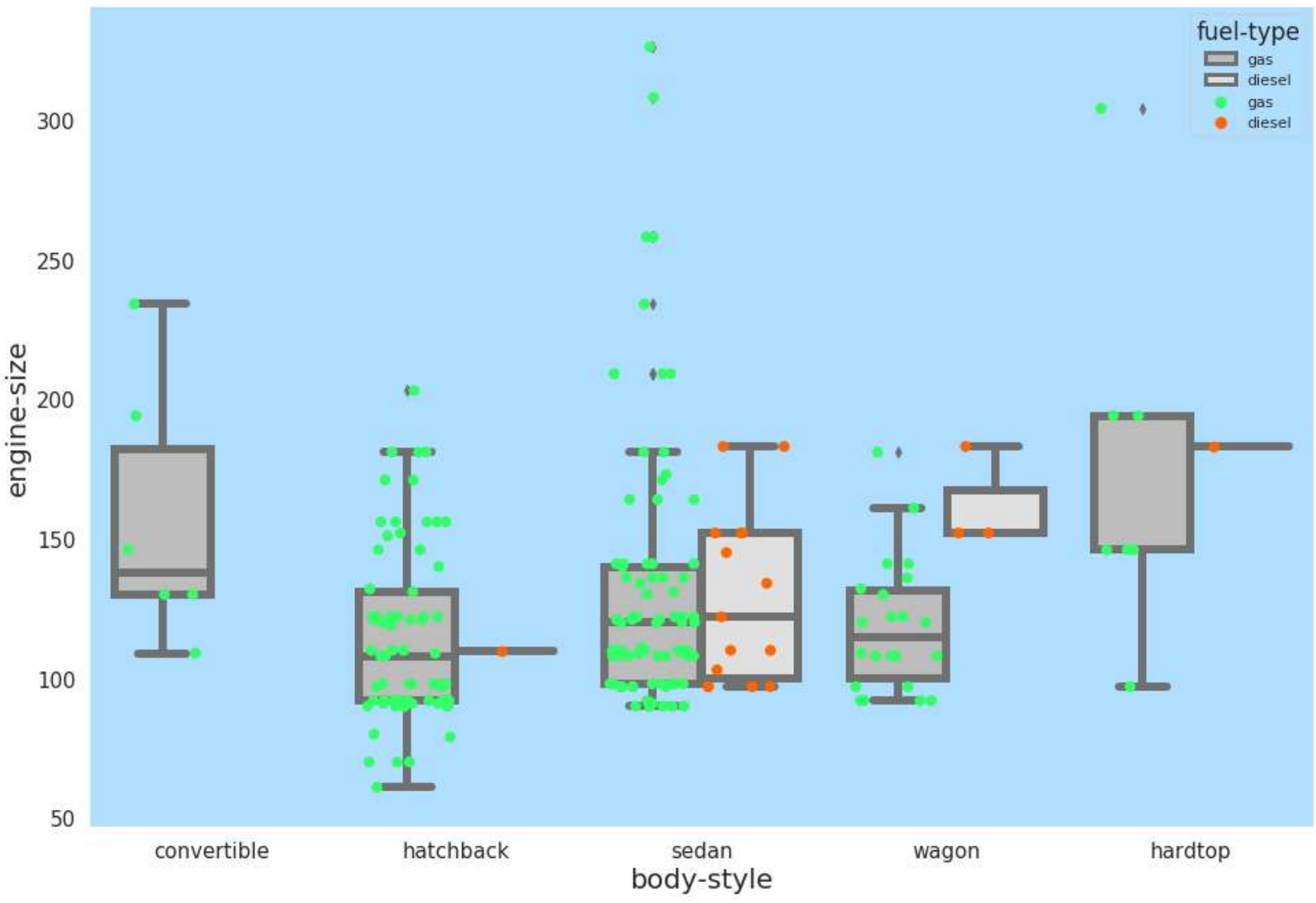
In [94]:
```
# Drawing stripplot on top of a box plot

plt.figure(figsize=(11,8))
sns.stripplot(x=auto["body-style"] ,palette="Set1", y = auto["engine-size"],hue=auto["fuel-type"],jitter=True, color="b
sns.boxplot(x=auto["body-style"] ,palette="Set1", y = auto["engine-size"] , color='black')
plt.show()
```



In [95]:
```
plt.figure(figsize=(10,9))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
'xtick.labelsize':15,'ytick.labelsize':15,
'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
params = dict(data=auto ,x = auto["body-style"] ,y = auto["engine-size"] ,hue=auto["fuel-type"],dodge=True)
sns.stripplot(**params , size=9,jitter=0.35,palette=['#FF8F00','#558B2F'],edgecolor='black')
sns.boxplot(**params ,palette=['#909497','#D0D3D4'],linewidth=4)
plt.show()
```

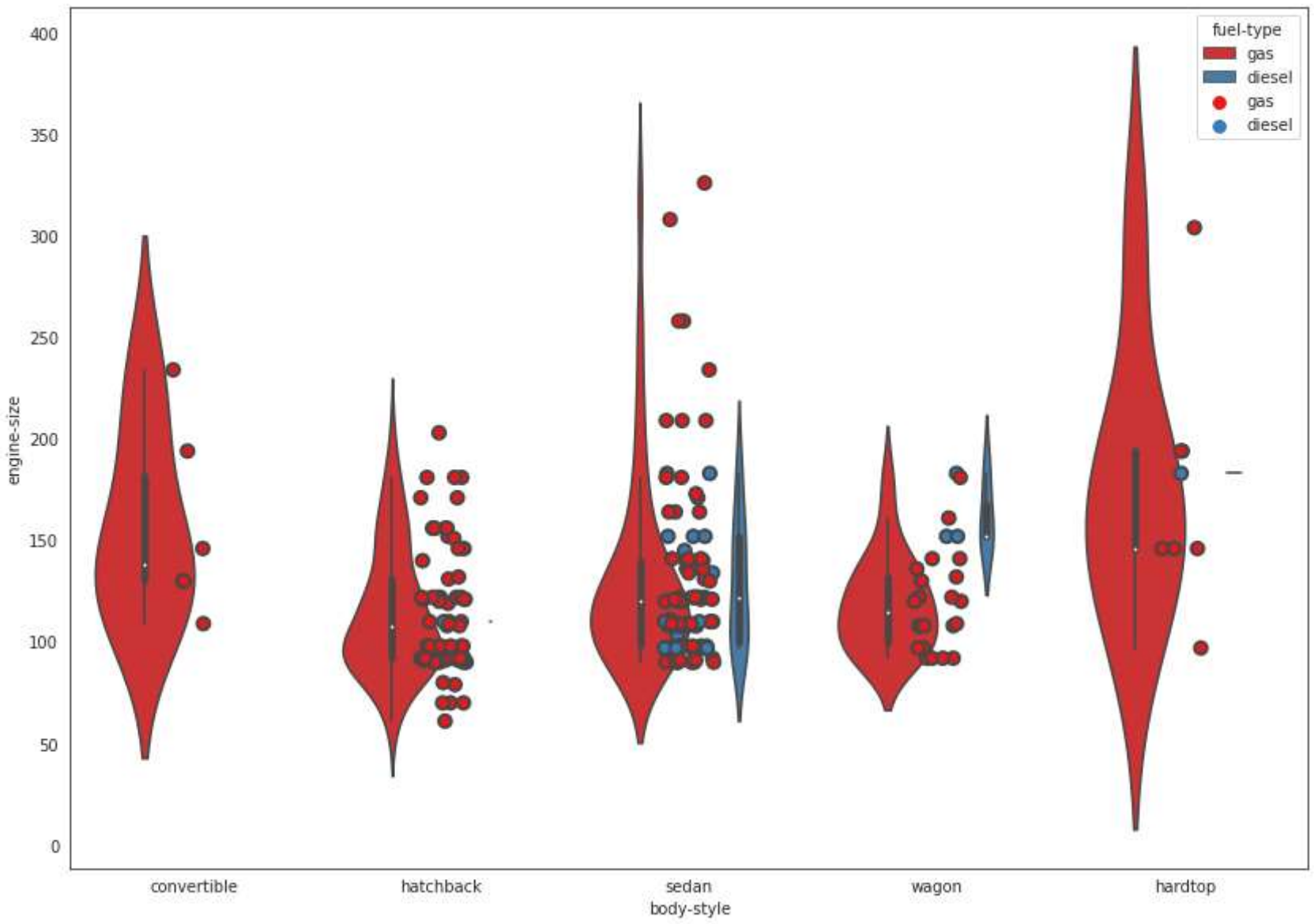


In [96]:
```
plt.figure(figsize=(16,11))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
'xtick.labelsize':15,'ytick.labelsize':15,
'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
params = dict(data=auto ,x = auto["body-style"] ,y = auto["engine-size"] ,hue=auto["fuel-type"],dodge=True)
sns.stripplot(**params , size=8,jitter=0.35,palette=['#33FF66','#FF6600'],edgecolor='black')
sns.boxplot(**params ,palette=['#BDBDBD','#E0E0E0'],linewidth=6)
plt.show()
```



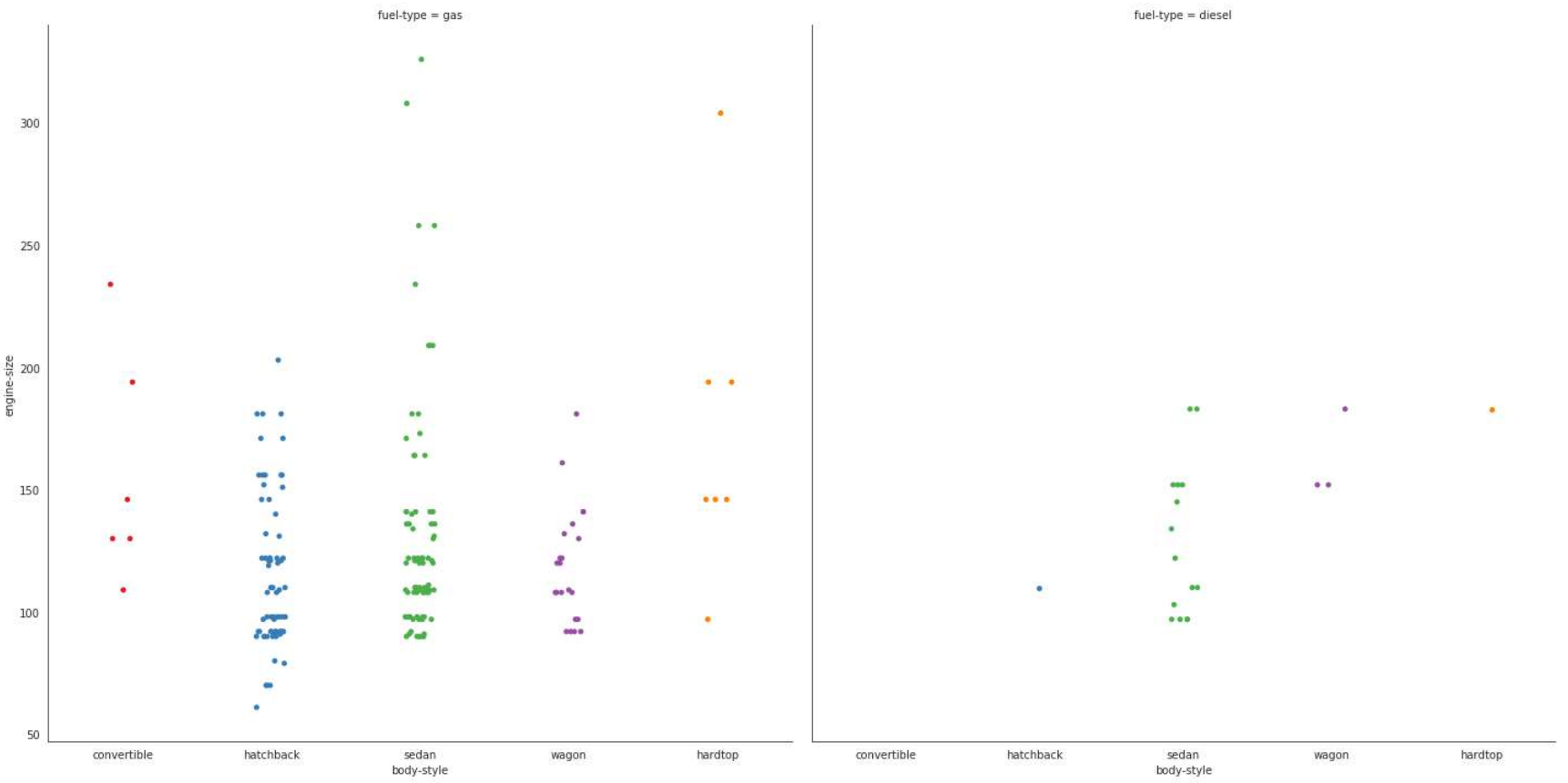
In [97]:
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

In [98]:
```
# Drawing stripplot on top of a violin plot
plt.figure(figsize=(14,10))
sns.stripplot(x=auto["body-style"] ,palette="Set1", y = auto["engine-size"],hue=auto["fuel-type"],jitter=True, color="b
sns.violinplot(x=auto["body-style"] ,palette="Set1", y = auto["engine-size"],hue=auto["fuel-type"],scale="count")
plt.show()
```



In [99]:
```
# Facet along the columns to show a categorical variable using "col" parameter
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type", kind="strip",palette="Set1" , height=10,data=auto)
plt.show()
```

```
<Figure size 792x648 with 0 Axes>
```



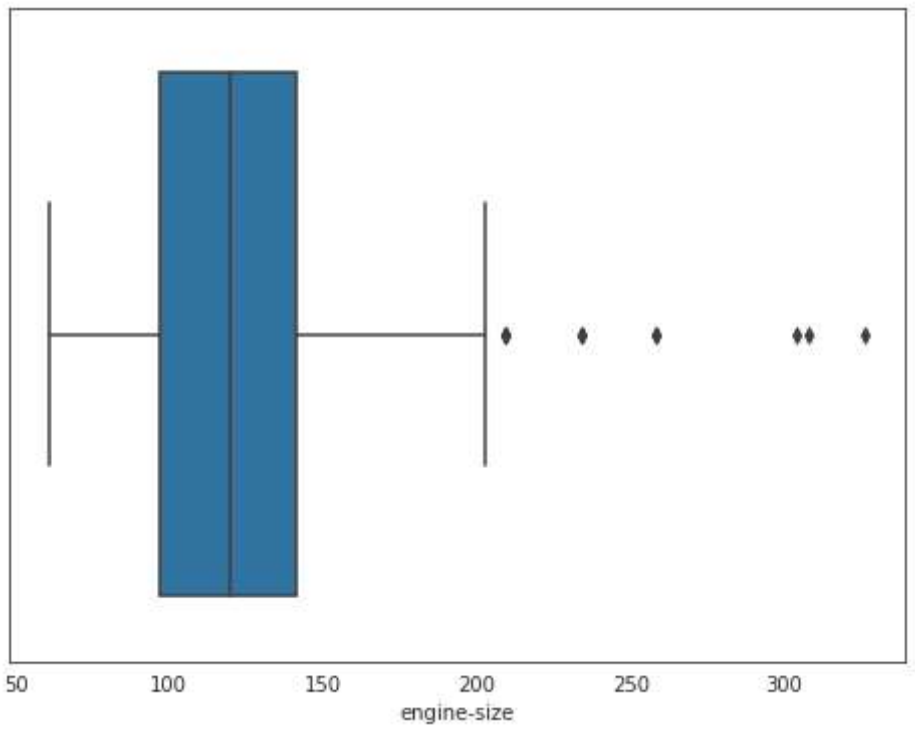
14.Boxplot

The box plot is a standardized way of displaying the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum.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 [100…
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

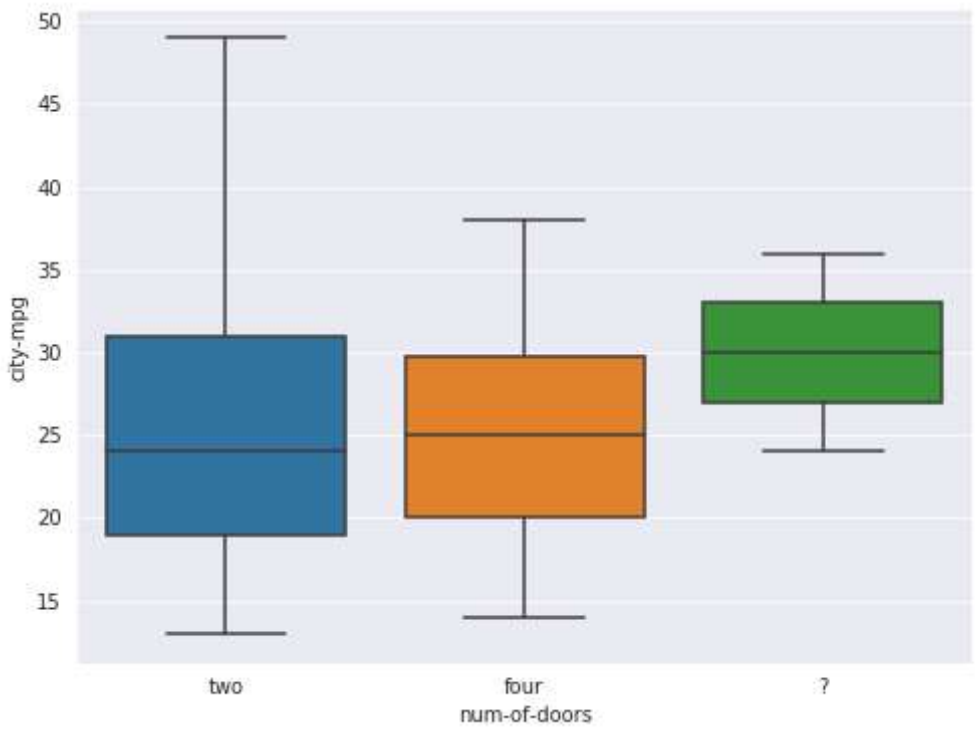
In [101…
```
# Simple boxplot
plt.figure(figsize=(8,6))
sns.boxplot(auto["engine-size"])
```

Out[101]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7bf630ddee80>
```



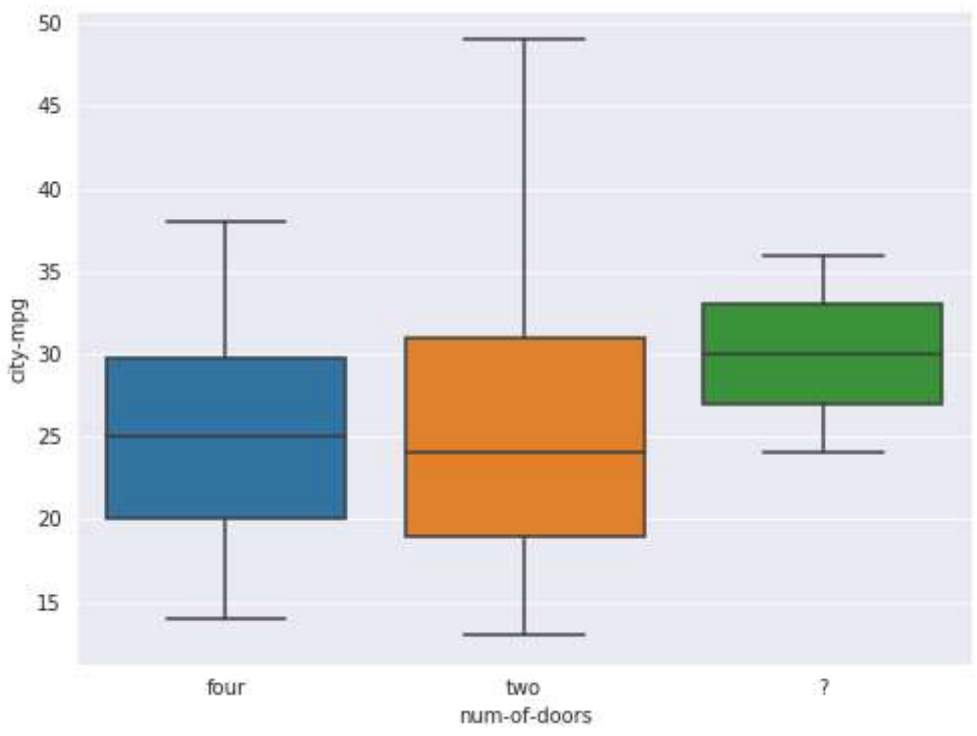
In [102…
```
# Vertical boxplot
sns.set_style("darkgrid")
plt.figure(figsize=(8,6))
sns.boxplot(auto['num-of-doors'], auto['city-mpg'])
```

Out[102]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7bf630e1ed30>
```

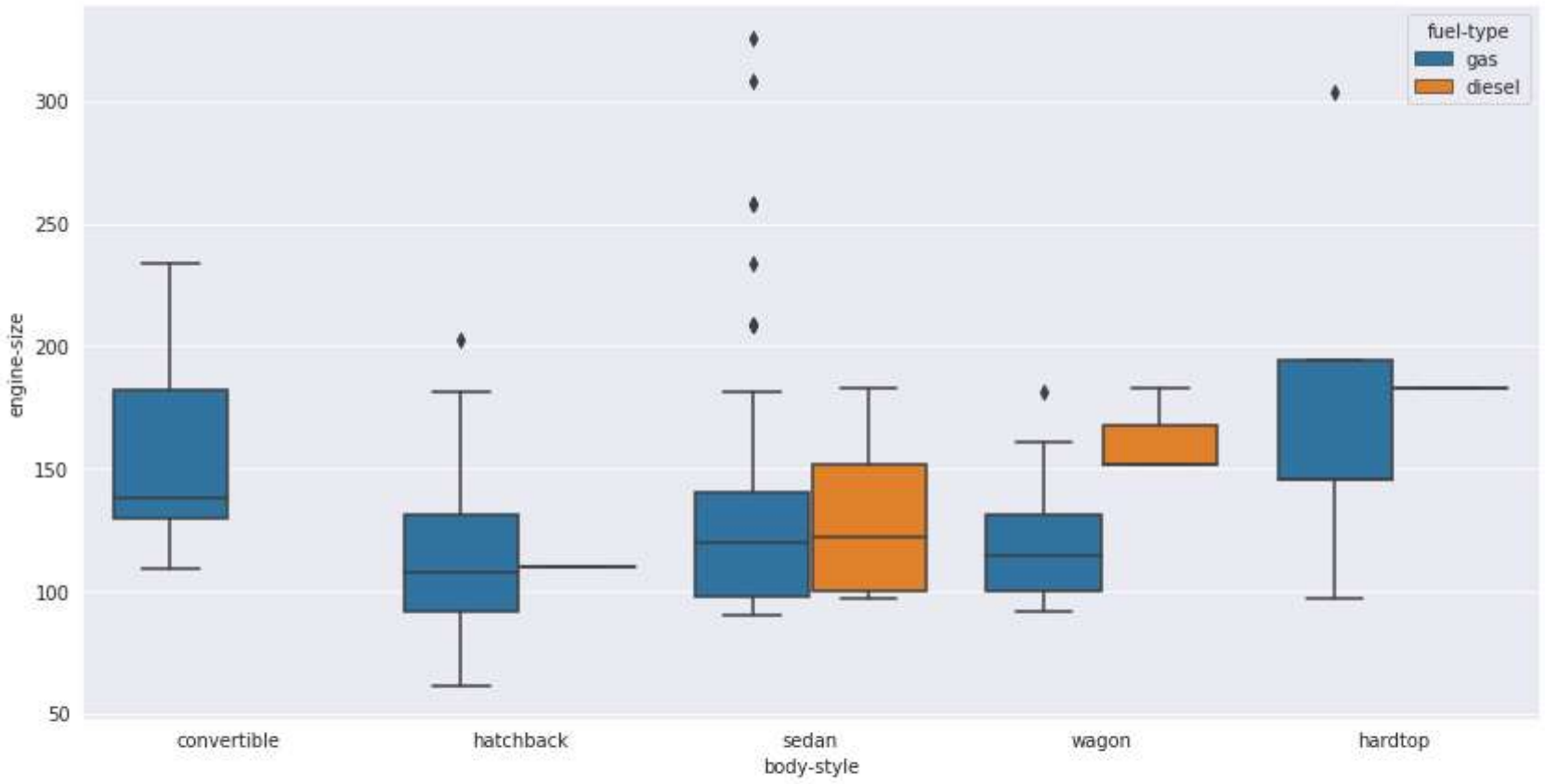


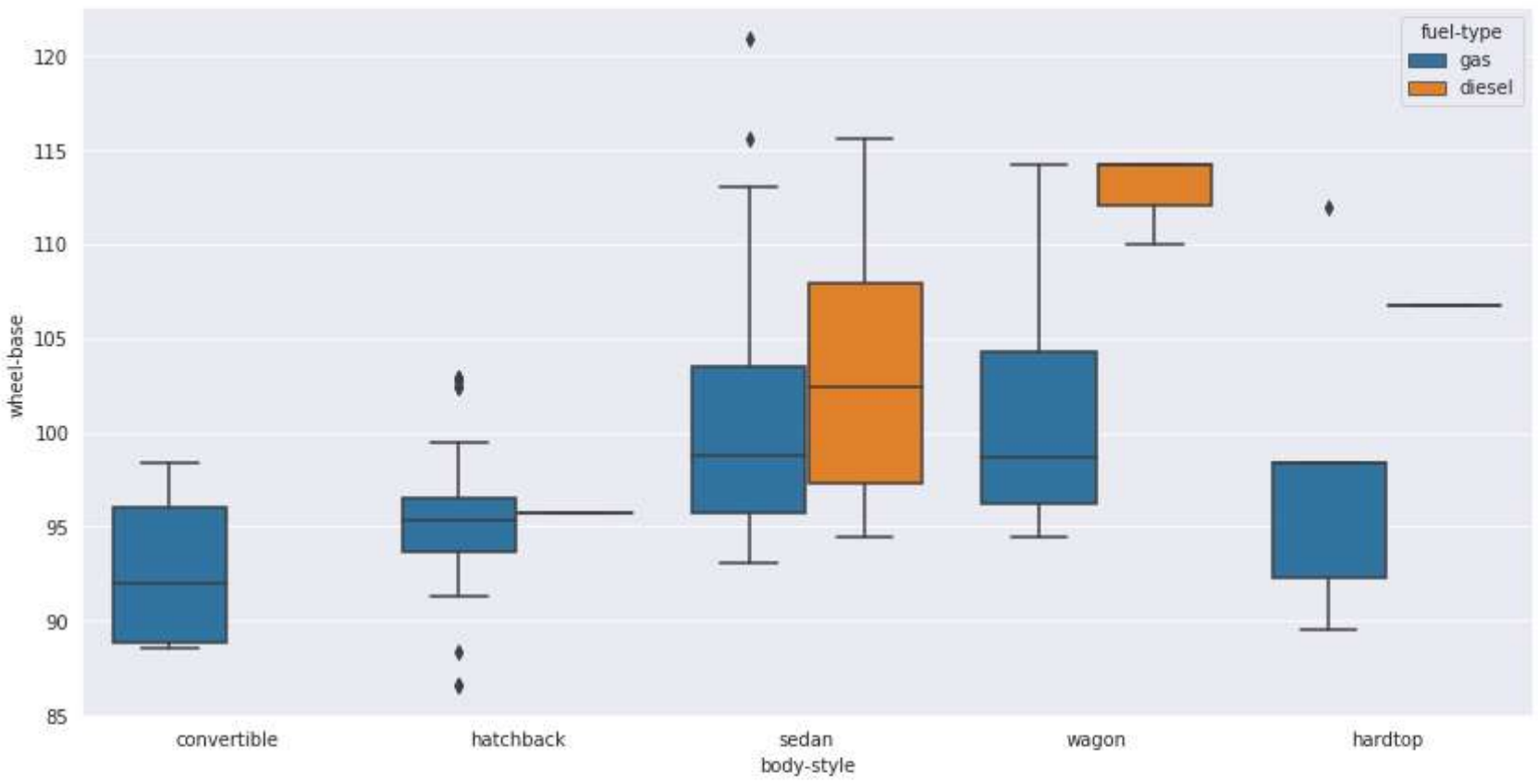
In [103…
```
#Explicit ordering using "order" parameter
sns.set_style("darkgrid")
plt.figure(figsize=(8,6))
sns.boxplot(auto['num-of-doors'], auto['city-mpg'],order = ['four' , 'two','?'])
```

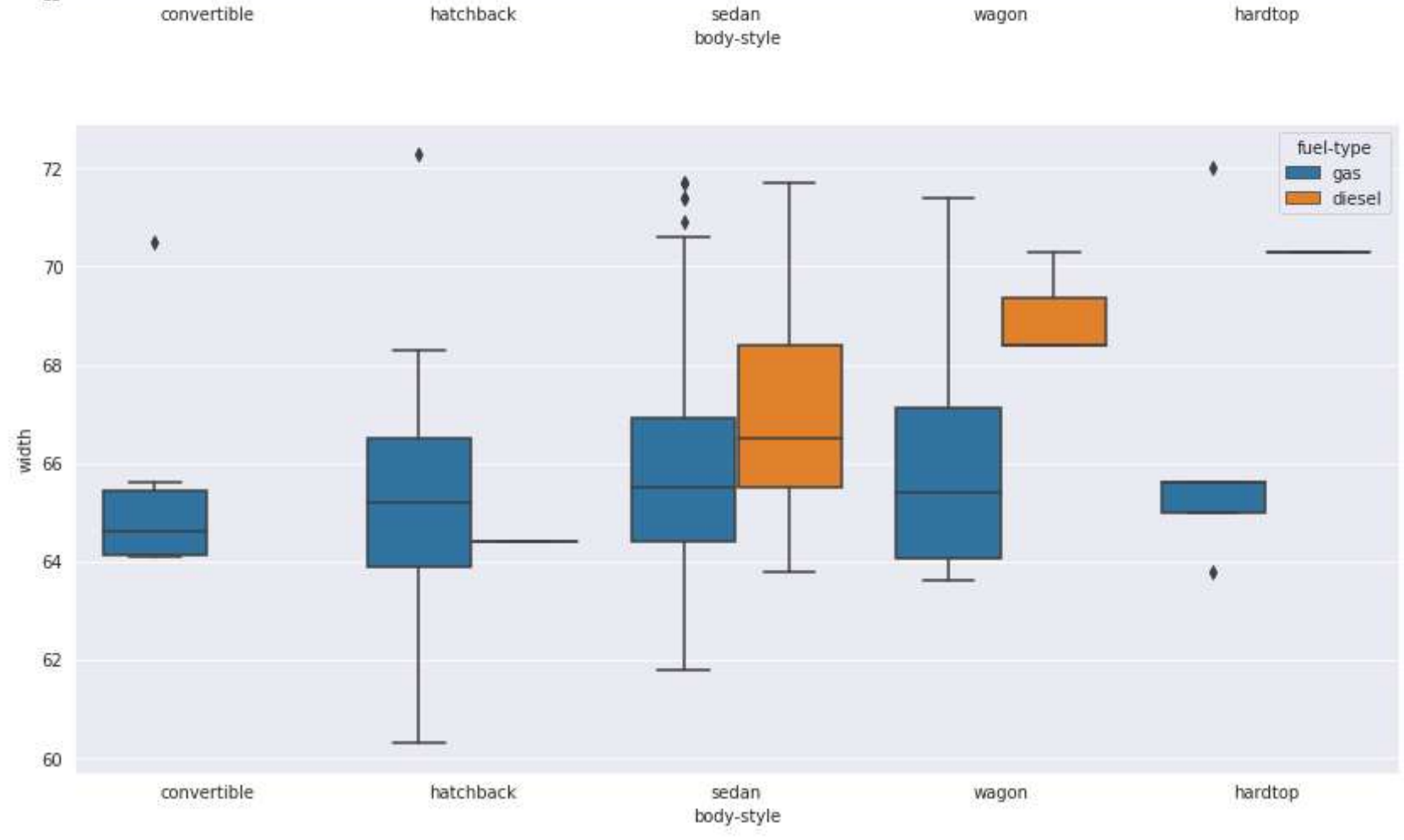
Out[103]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7bf63352d860>
```



In [104…
```
#Plotting subplots
plt.subplots(figsize = (14,24))
plt.subplot(3,1,1)
sns.boxplot(x= auto['body-style'] , y= auto['engine-size'] ,hue= auto["fuel-type"])
plt.subplot(3,1,2)
sns.boxplot(x=auto['body-style'], y=auto["wheel-base"], hue=auto["fuel-type"])
plt.subplot(3,1,3)
sns.boxplot(x=auto['body-style'],y=auto["width"] , hue=auto["fuel-type"])
plt.show()
```

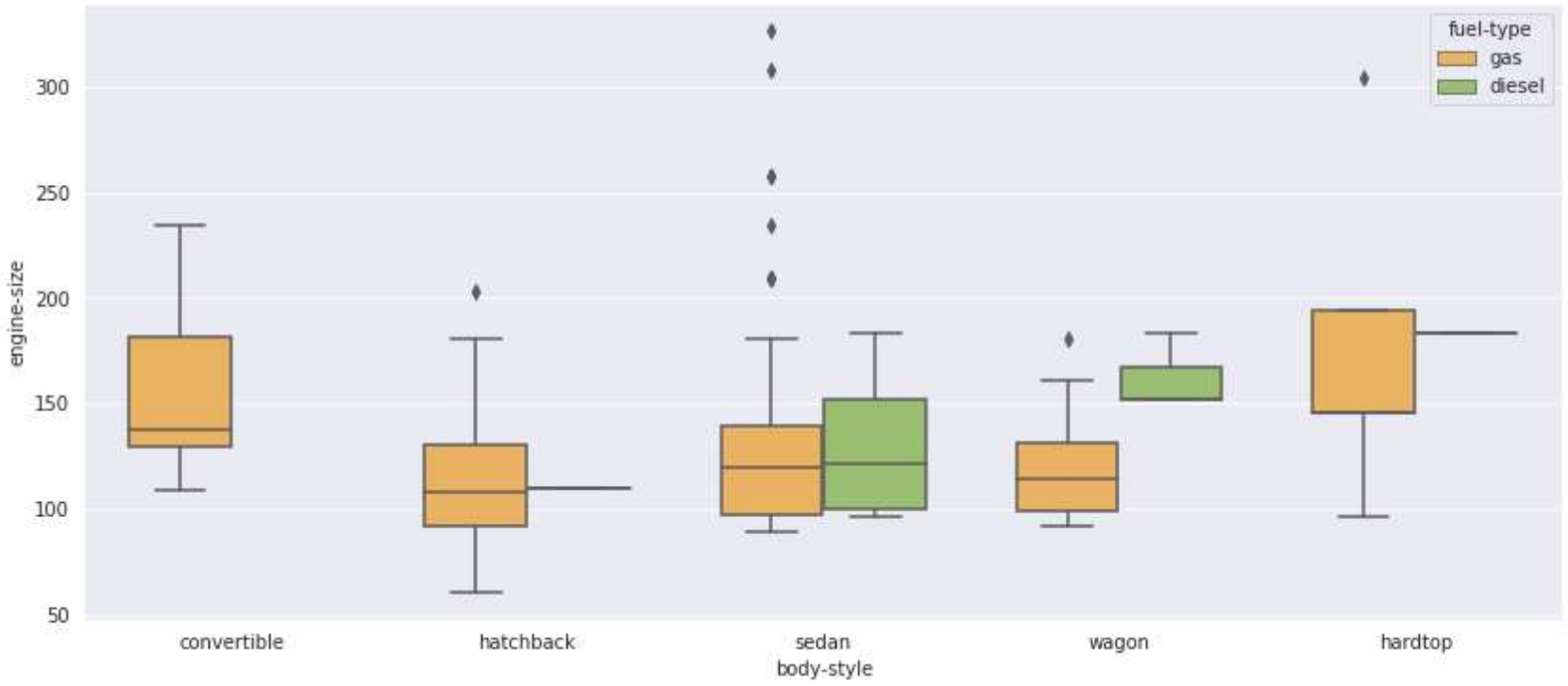






In [105…

```
plt.figure(figsize = (14,6))
sns.boxplot(x= auto['body-style'] , y= auto['engine-size'] ,hue= auto["fuel-type"],width=.7,palette= {"gas":'#FFB74D' ,
plt.show()
```

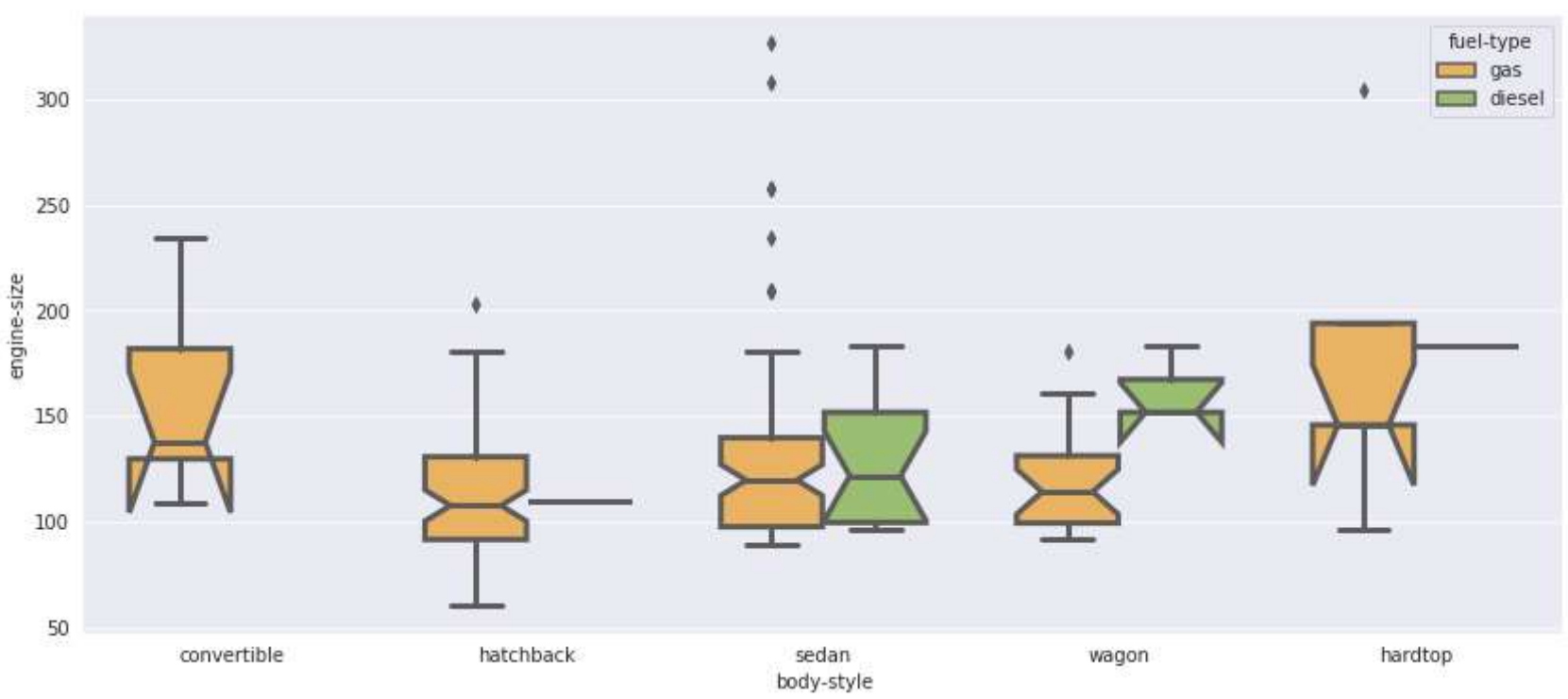


In [106…

```
plt.figure(figsize = (14,6))
sns.boxplot(x= auto['body-style'] , y= auto['engine-size'] ,hue= auto["fuel-type"],width=.7,palette= {"gas":'#FFB74D' ,
sns.despine()
# More about sns.despine() here - https://seaborn.pydata.org/tutorial/aesthetics.html
plt.show()
```

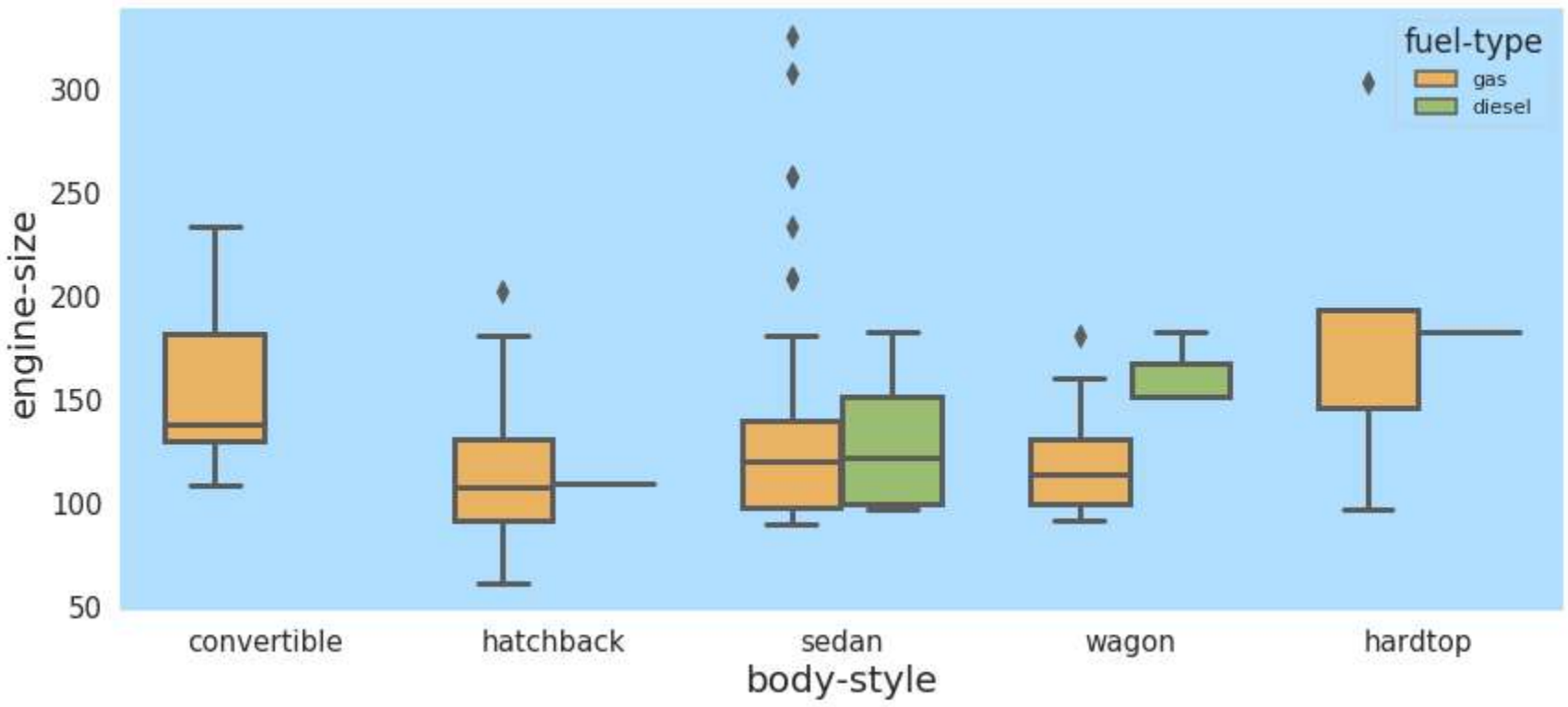

In [109…

```
# Add a notch to the box

plt.figure(figsize = (14,6))
sns.boxplot(x= auto['body-style'] , y= auto['engine-size'] ,hue= auto["fuel-type"],width=.7,palette= {"gas":'#FFB74D' ,
sns.despine(left=True)
# More about sns.despine() here - https://seaborn.pydata.org/tutorial/aesthetics.html
plt.show()
```



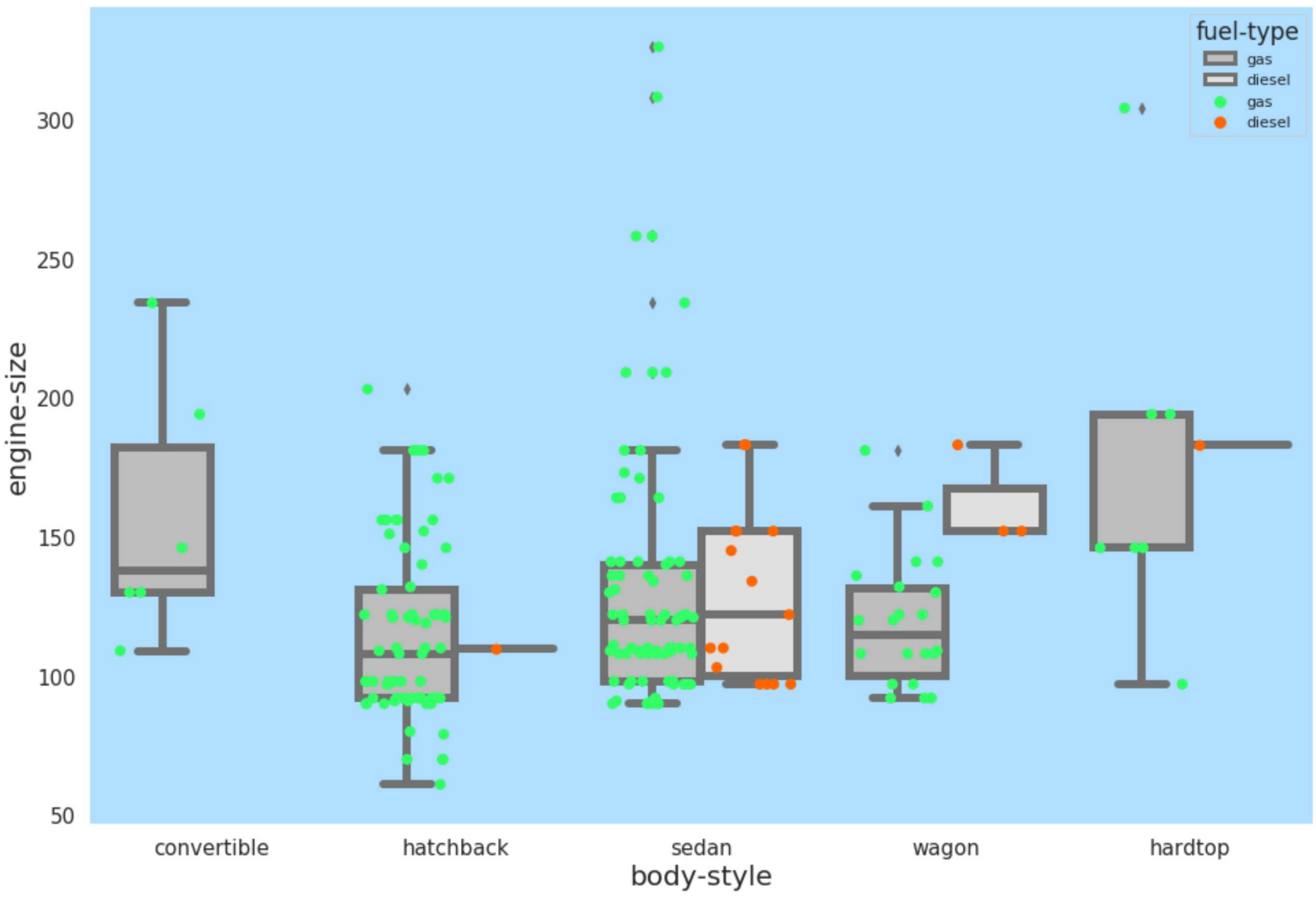
In [110…

```
# Change the size of outlier markers using fliersize
plt.figure(figsize = (14,6))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
'xtick.labelsize':15,'ytick.labelsize':15,
'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
sns.boxplot(x= auto['body-style'] , y= auto['engine-size'] ,hue= auto["fuel-type"],width=.7,palette= {"gas":'#FFB74D' ,
sns.despine(left=True)
# More about sns.despine() here - https://seaborn.pydata.org/tutorial/aesthetics.html
plt.show()
```



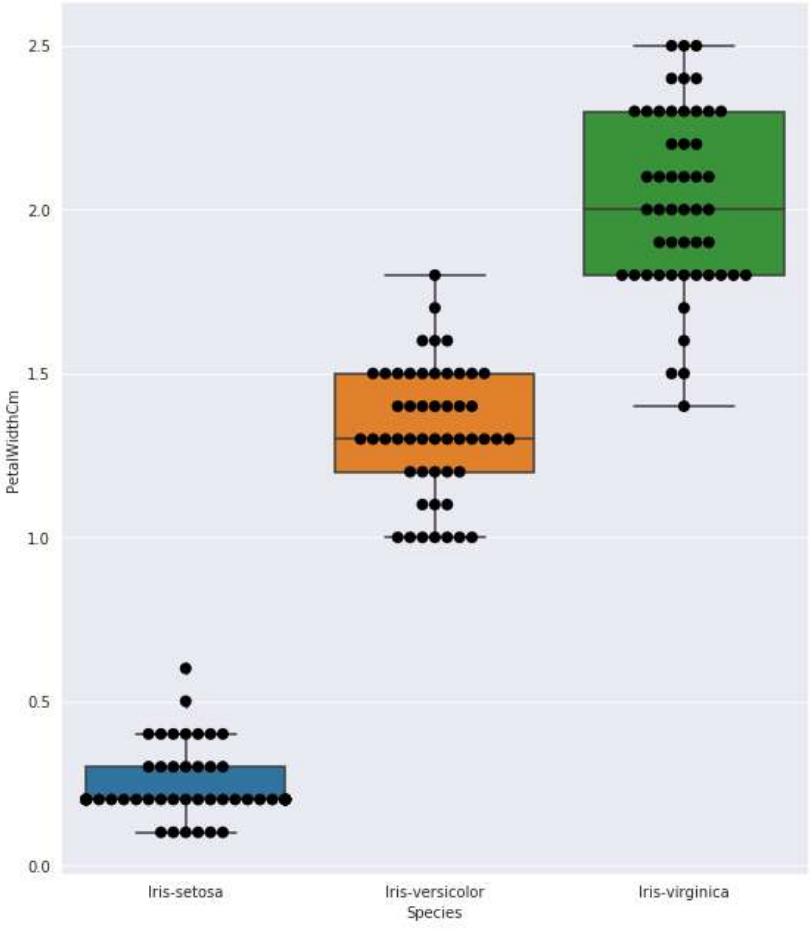
In [111…
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

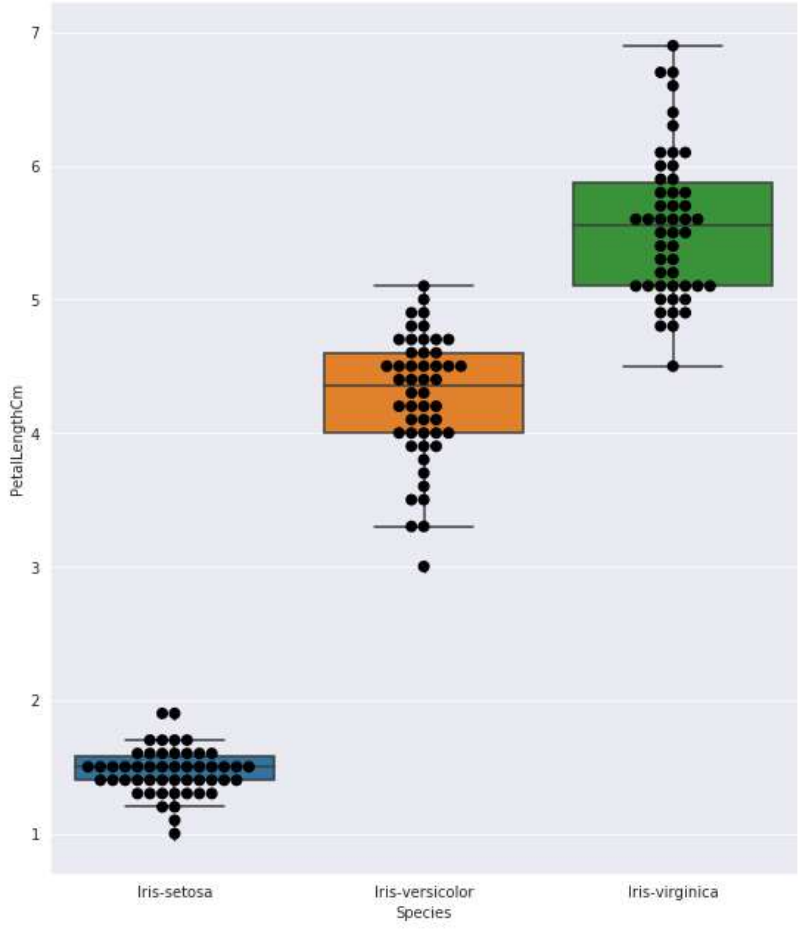
In [112…
```
plt.figure(figsize=(16,11))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
'xtick.labelsize':15,'ytick.labelsize':15,
'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
params = dict(data=auto ,x = auto["body-style"] ,y = auto["engine-size"] ,hue=auto["fuel-type"],dodge=True)
sns.stripplot(**params , size=8,jitter=0.35,palette=['#33FF66','#FF6600'],edgecolor='black')
sns.boxplot(**params ,palette=['#BDBDBD','#E0E0E0'],linewidth=6)
plt.show()
```

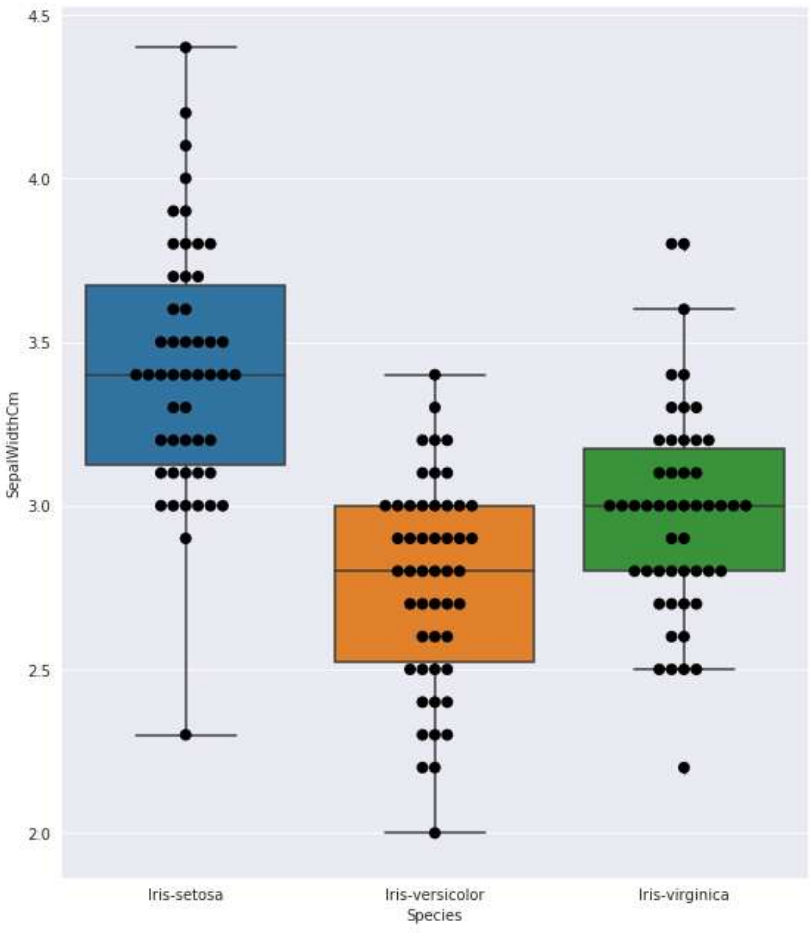


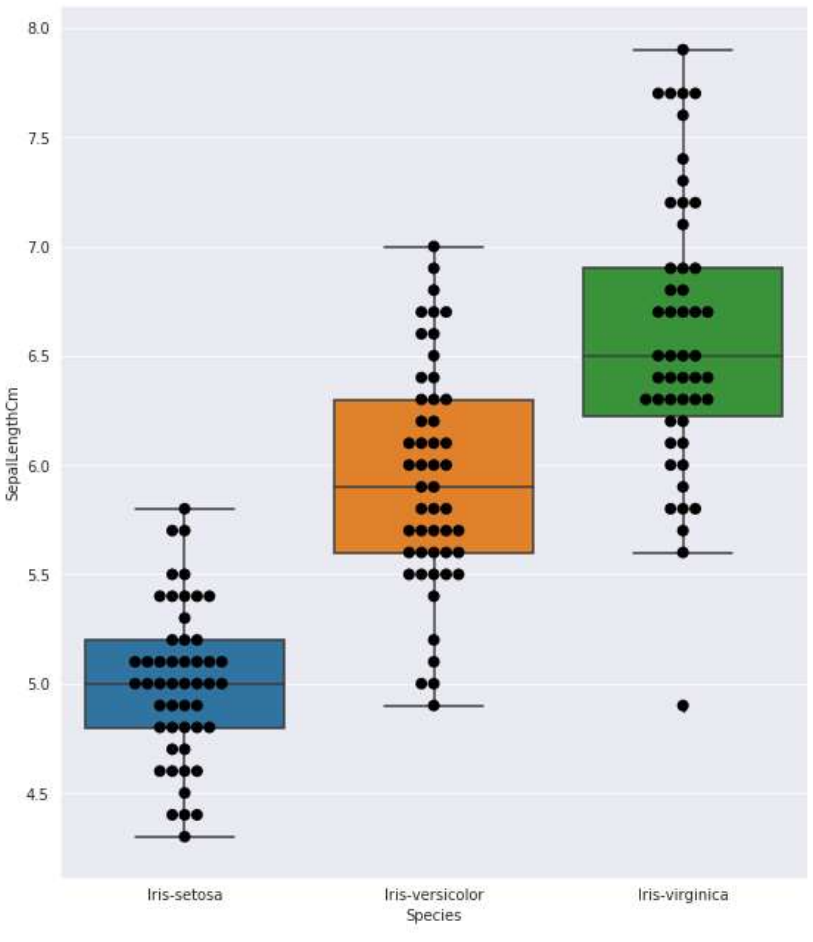
In [113…
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("darkgrid")
```

In [114…
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,24))
sns.swarmplot(x="Species" , y = "PetalWidthCm" , ax = axes[0,0] ,data=iris, size=8 ,color="black")
sns.boxplot(x="Species" , y = "PetalWidthCm" , ax = axes[0,0] ,data=iris )
sns.swarmplot(x="Species" , y = "PetalLengthCm" ,ax = axes[0,1] , data=iris , size=8,color="black")
sns.boxplot(x="Species" , y = "PetalLengthCm" ,ax = axes[0,1] , data=iris )
sns.swarmplot(x="Species" , y = "SepalWidthCm" , ax = axes[1,0] , data=iris , size=8,color="black")
sns.boxplot(x="Species" , y = "SepalWidthCm" , ax = axes[1,0] , data=iris )
sns.swarmplot(x="Species" , y = "SepalLengthCm" , ax = axes[1,1] , data=iris, size=8,color="black")
sns.boxplot(x="Species" , y = "SepalLengthCm" , ax = axes[1,1] , data=iris )
plt.show()
```







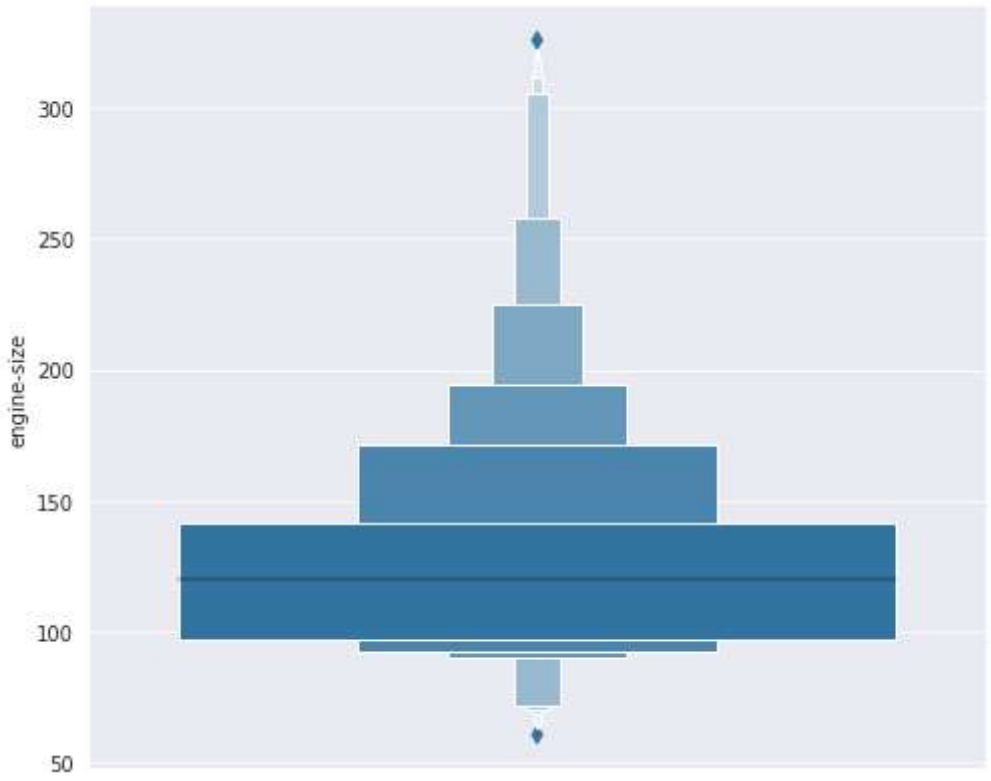


15. Boxen Plot

The Boxen Plot shows a large number of quantiles.By plotting more quantiles, it provides more information about the shape of the distribution, particularly in the tails.

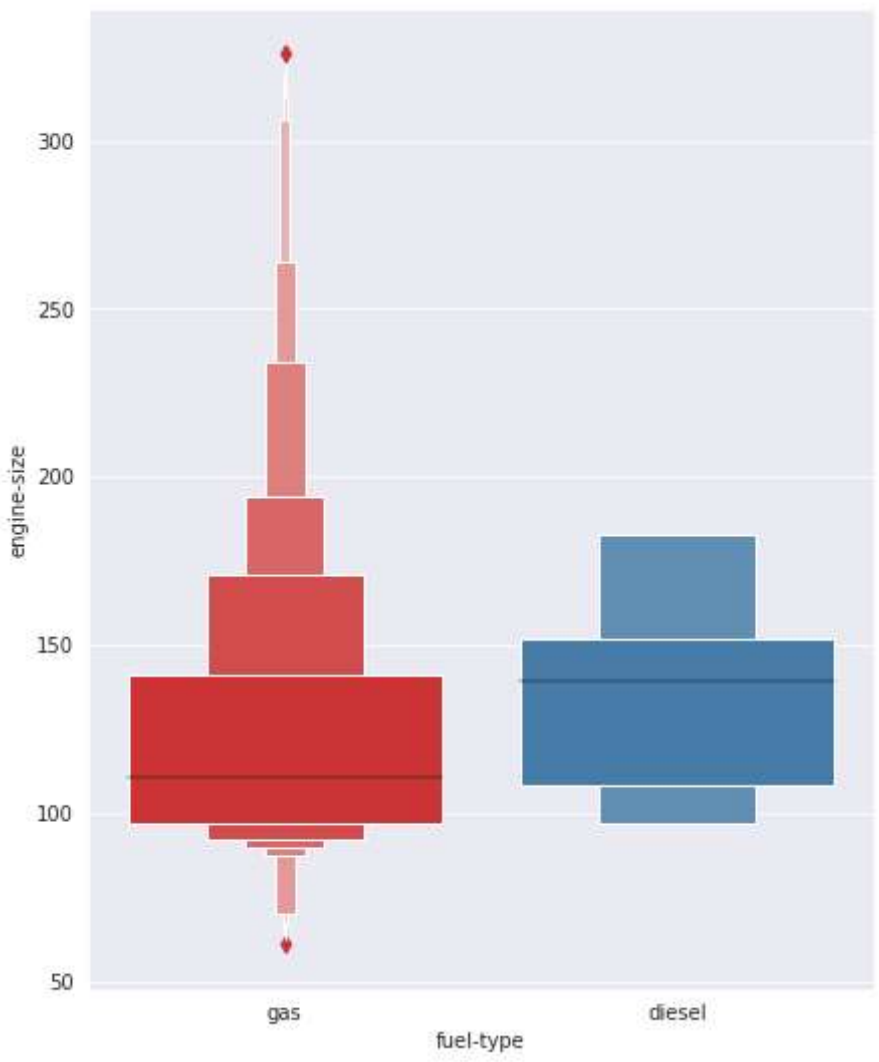
In [115…
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("darkgrid")
```

In [116…
```
# Simple Boxen Plot
plt.figure(figsize=(8,7))
sns.boxenplot(y=auto["engine-size"])
plt.show()
```



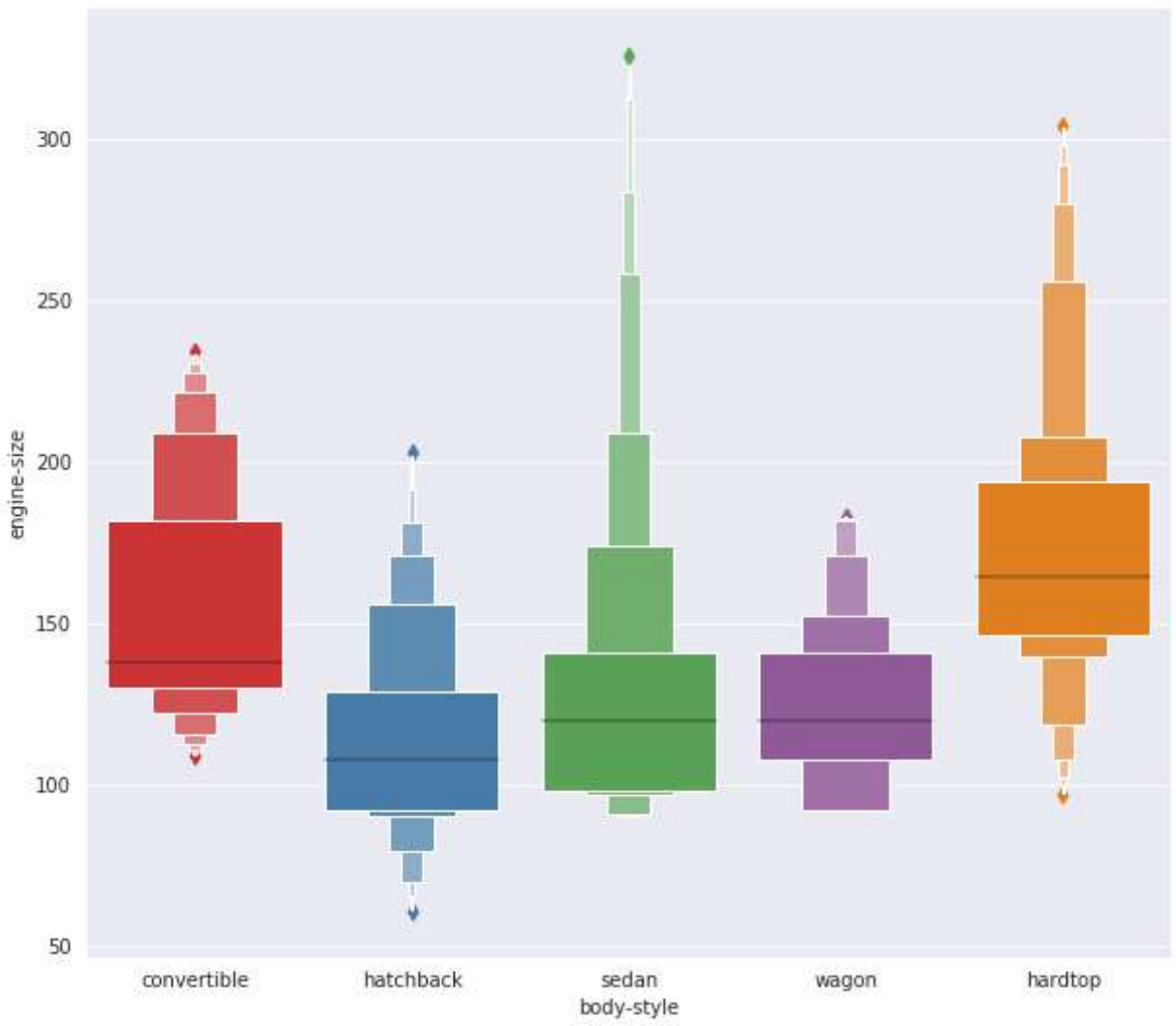
In [117…

```
plt.figure(figsize=(7,9))
sns.boxenplot(x=auto["fuel-type"] , y = auto["engine-size"] ,palette="Set1")
plt.show()
```



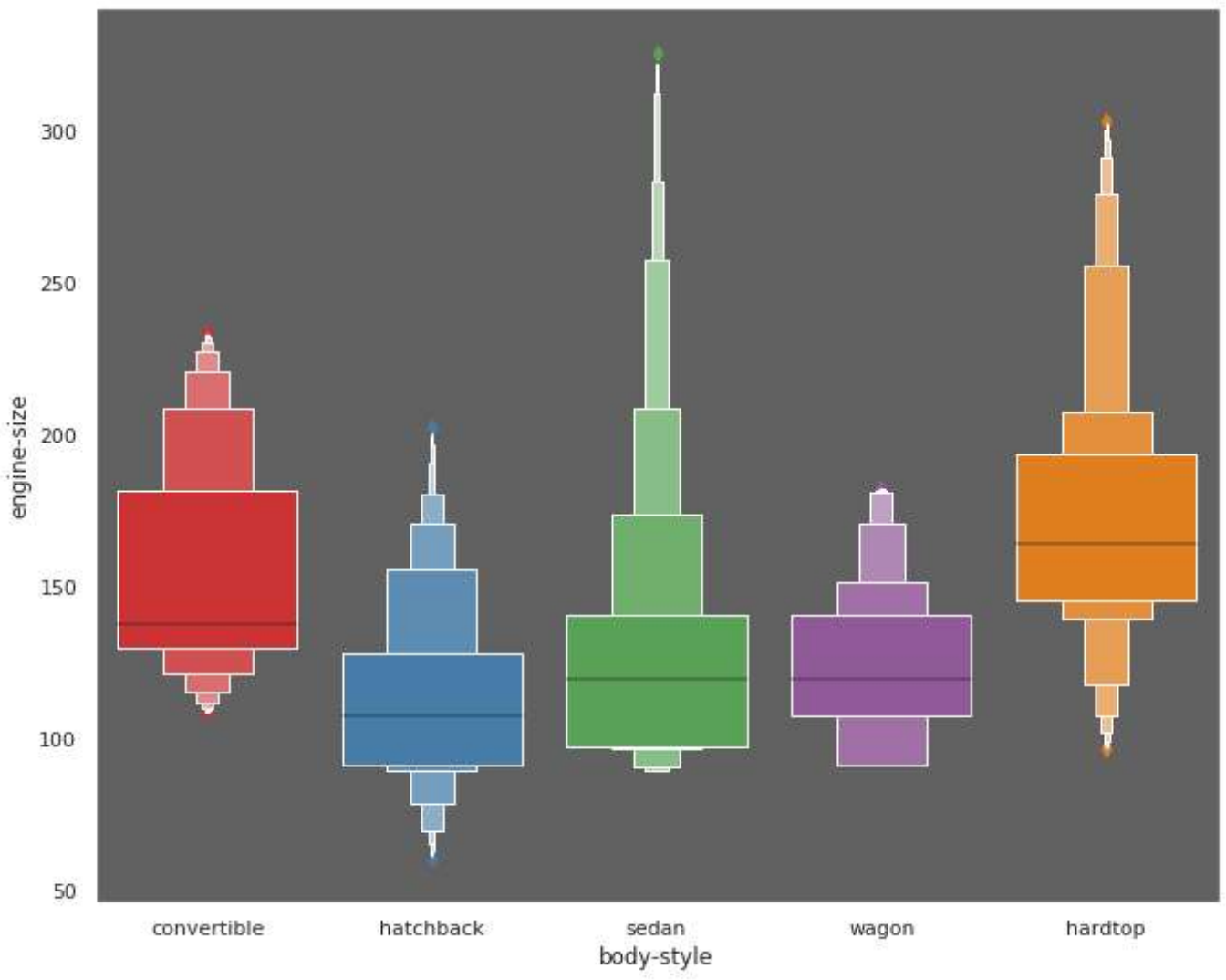
In [118…

```
# Drawing a vertical boxenplot grouped by a categorical variable
plt.figure(figsize=(10,9))
sns.boxenplot(x=auto["body-style"] , y = auto["engine-size"] ,palette="Set1")
plt.show()
```



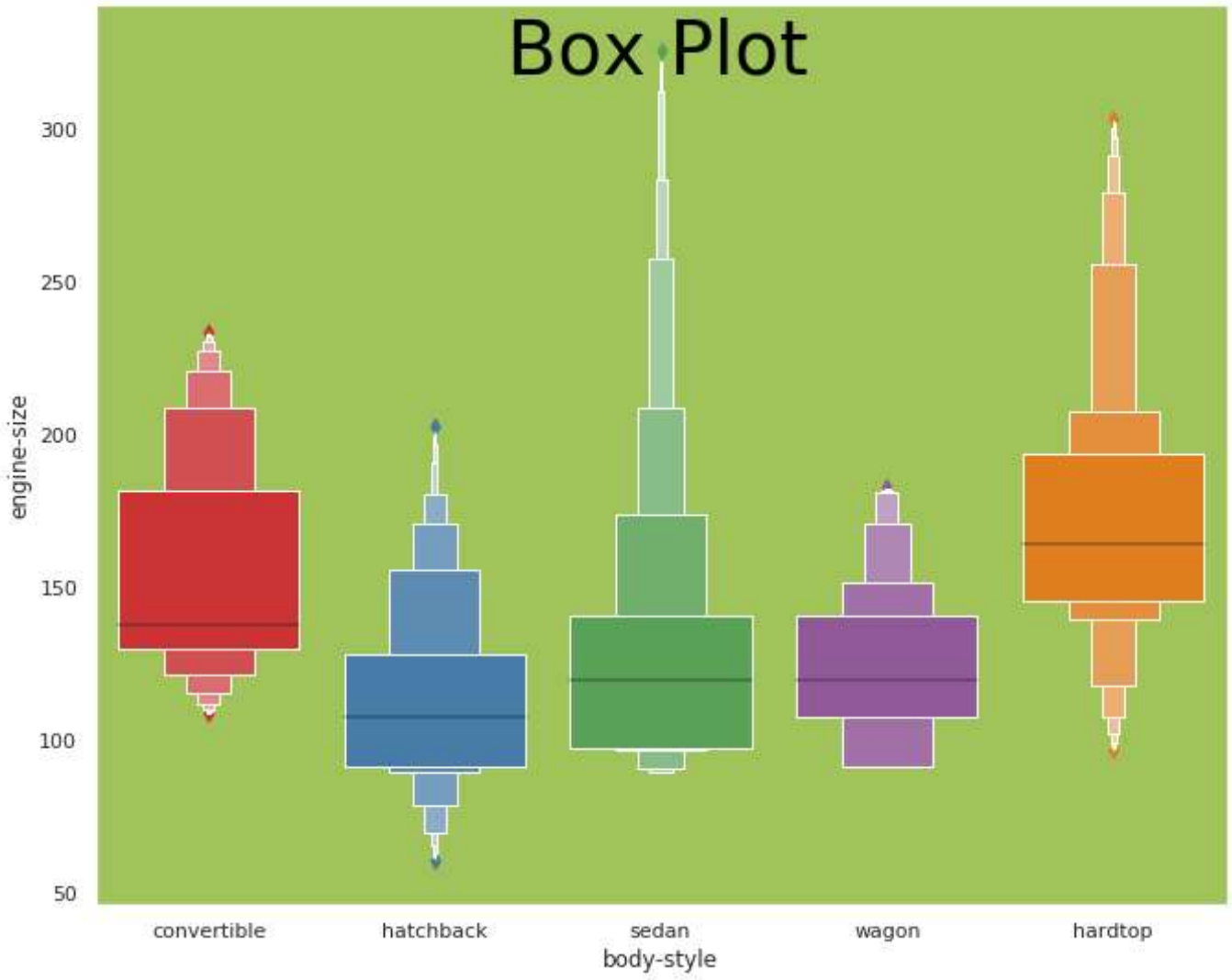
In [120…

```
sns.set(rc={"axes.facecolor":"#616161" , "axes.grid" : False})
plt.figure(figsize=(11,9))
sns.boxenplot(x=auto["body-style"] , y = auto["engine-size"],palette="Set1")
plt.show()
```



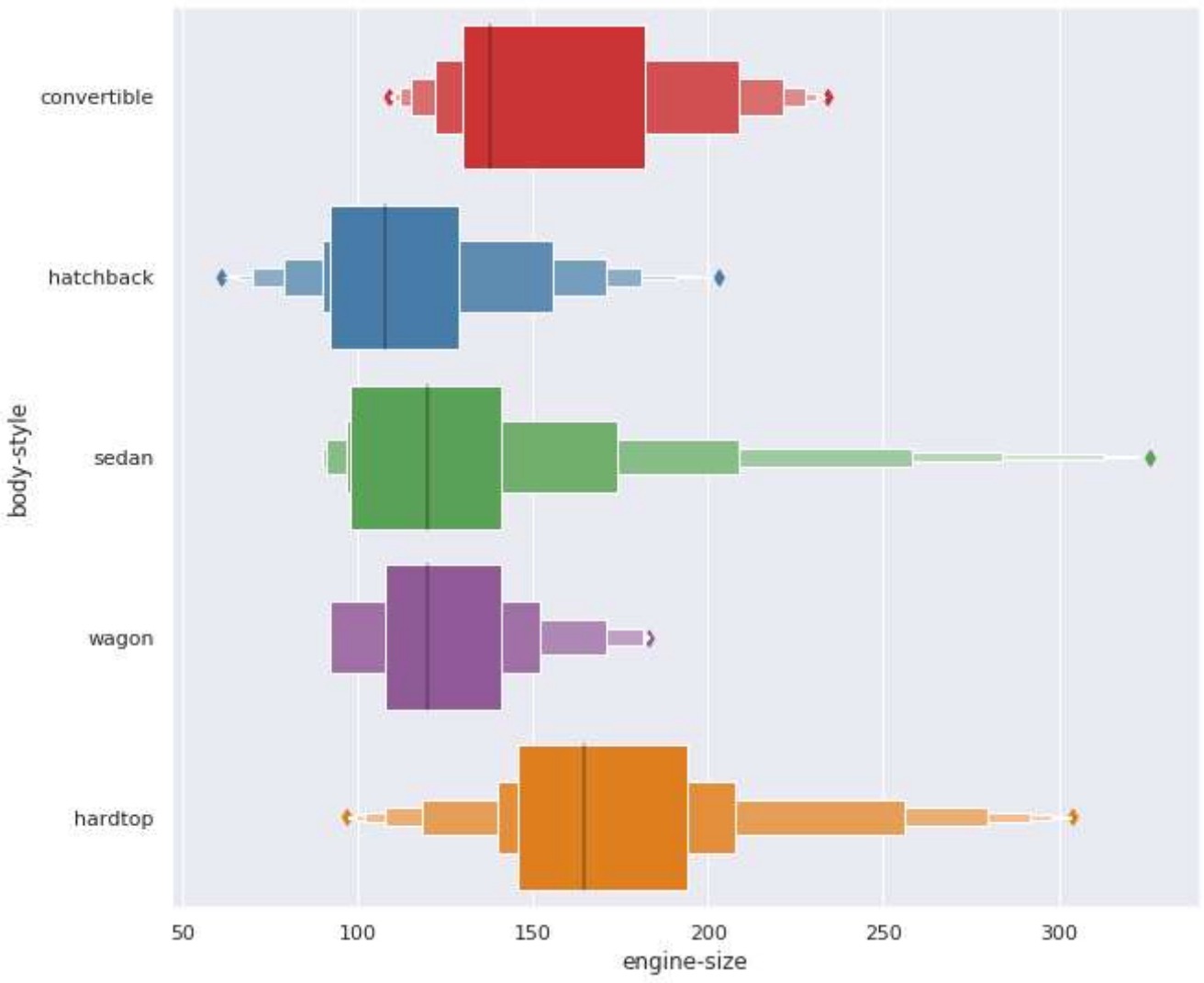
In [121…

```
sns.set(rc={"axes.facecolor":"#a1c45a" , "axes.grid" : False})
plt.figure(figsize=(11,9))
plt.gcf().text(.51, .84, "Box Plot", fontsize = 40, color='Black' ,ha='center', va='center')
sns.boxenplot(x=auto["body-style"] , y = auto["engine-size"],palette="Set1")
plt.show()
```



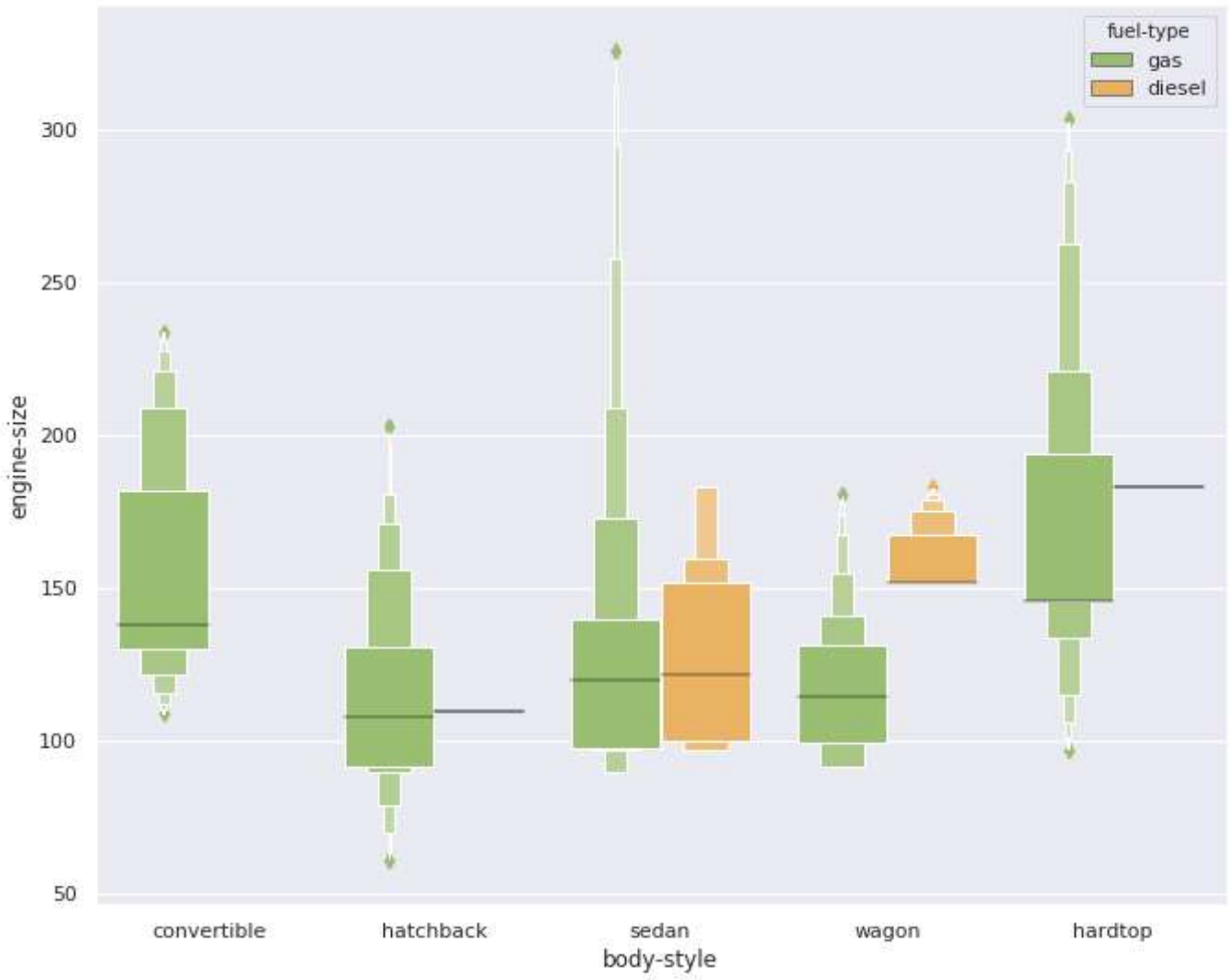
In [122…

```
# Horizontal Boxen plot
sns.set_style("darkgrid")
plt.figure(figsize=(10,9))
sns.boxenplot(x = auto["engine-size"] ,y= auto["body-style"] ,palette="Set1")
plt.show()
```



In [123…

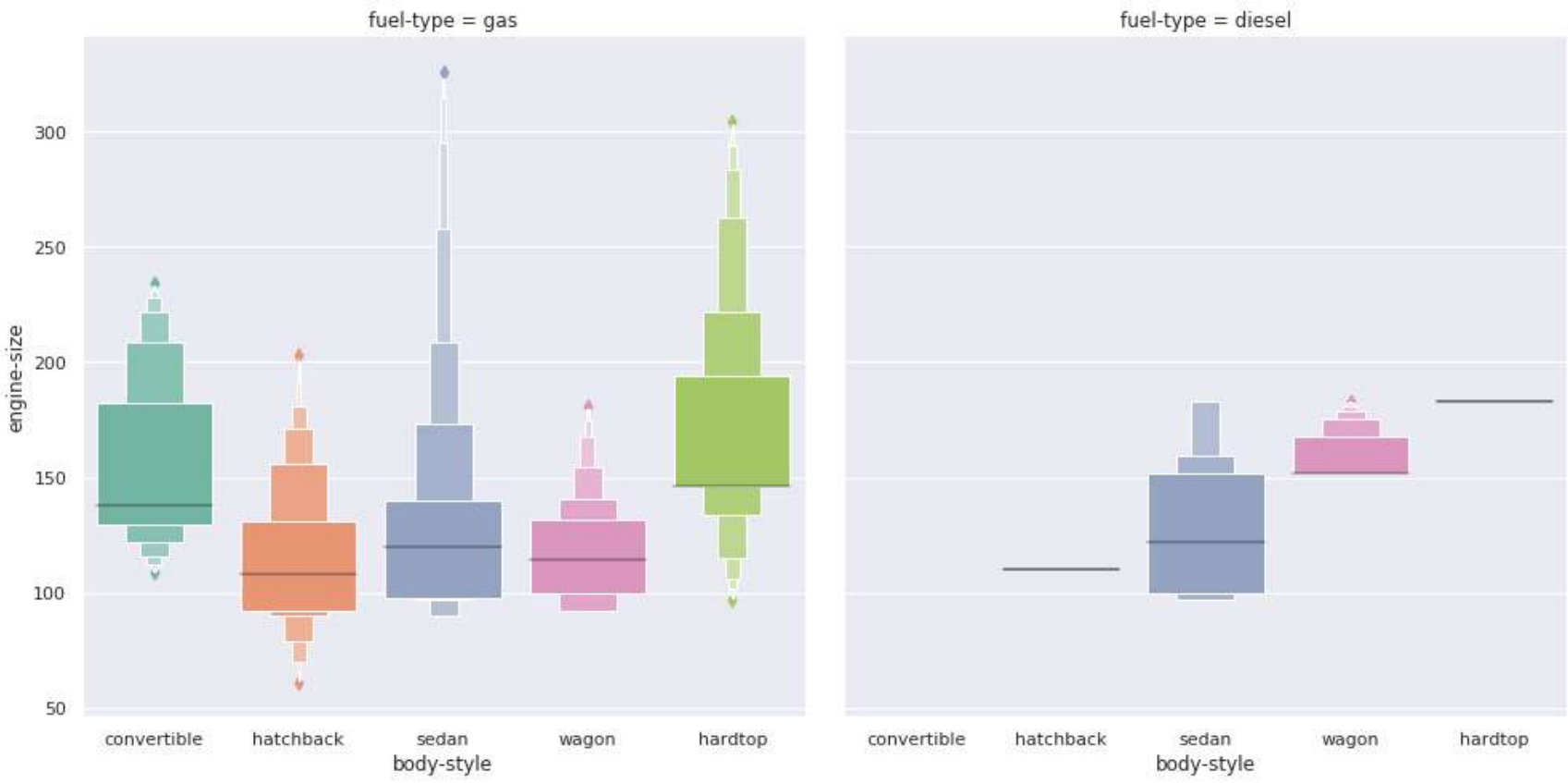
```
# Show groups with different colors using "hue" (Nested grouping by two categorical variabl
plt.figure(figsize=(11,9))
sns.boxenplot(x=auto["body-style"] , y = auto["engine-size"], hue=auto["fuel-type"], palette={"diesel":'#FFB74D' , "gas
plt.show()
```



In [124…

```
# Facet along the columns to show a categorical variable using "col" parameter
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type", kind="boxen",palette="Set2" , height=7,data=auto)
plt.show()
```

```
<Figure size 792x648 with 0 Axes>
```



16. Pair Plot

Pair Plot is used for plotting 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:

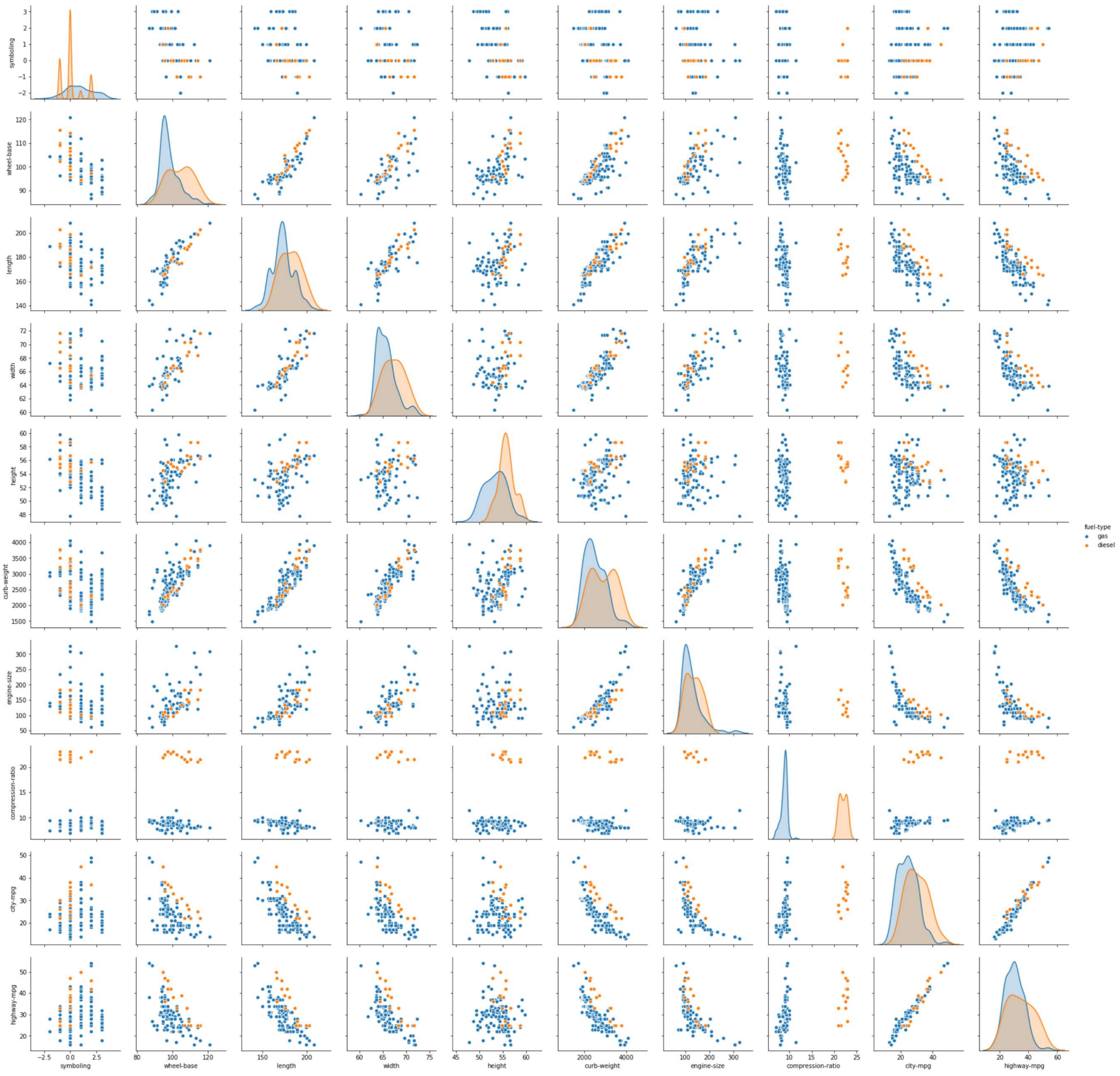
In [125…

```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

In [126…

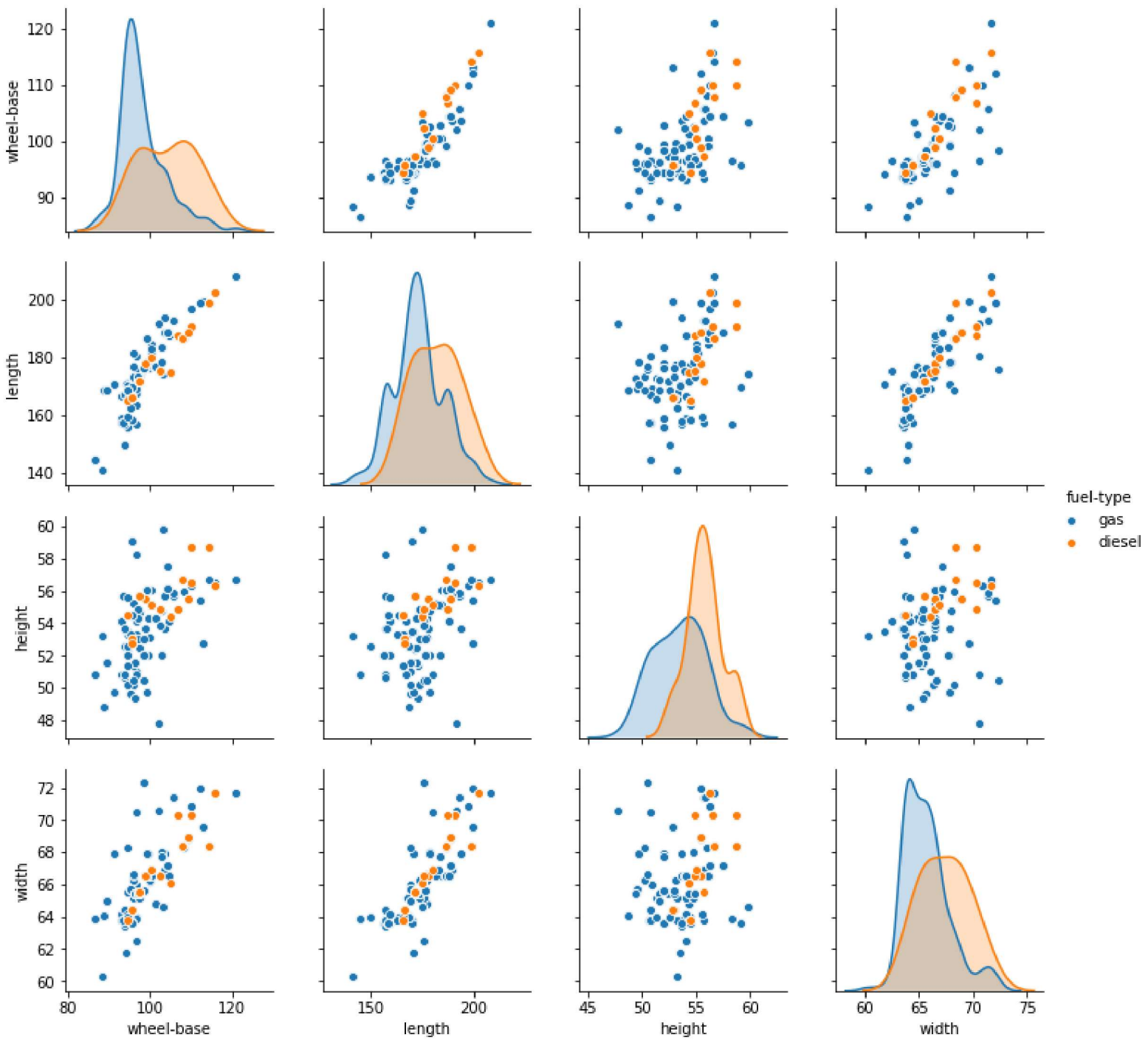
```
# Draw scatterplots for joint relationships and histograms for univariate distributions
plt.figure(figsize=(11,9))
sns.pairplot(auto,hue = 'fuel-type')
plt.show()
```

```
<Figure size 792x648 with 0 Axes>
```



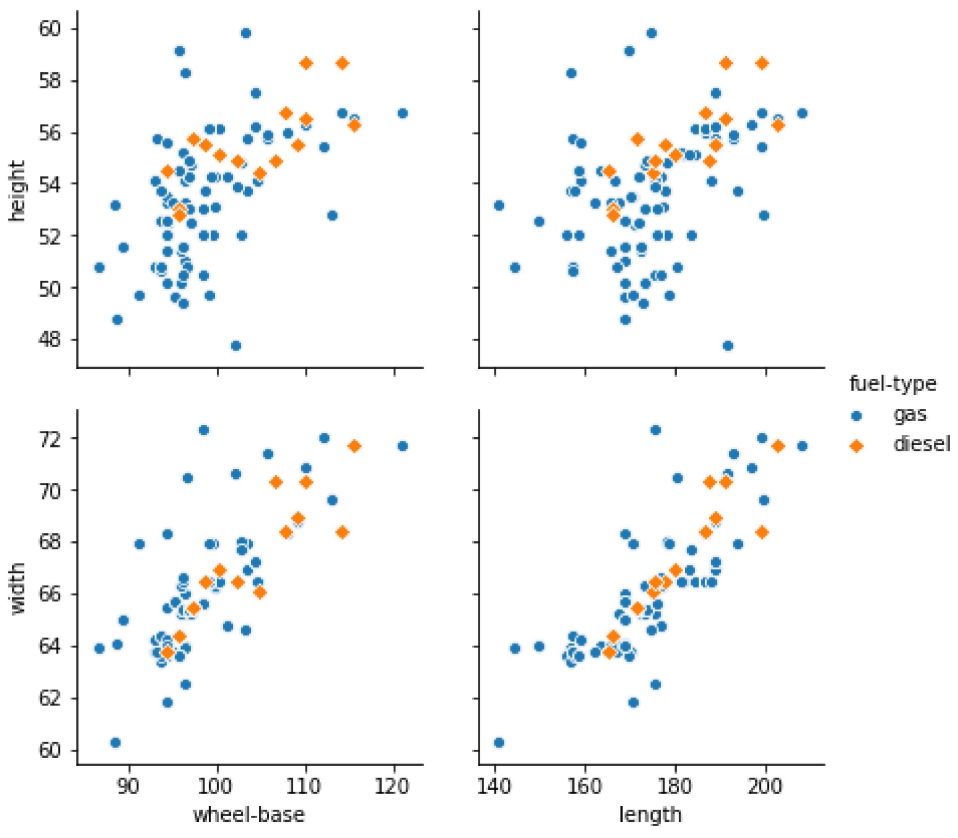
In [128…

```
# Plot a subset of variables
sns.pairplot(auto,hue = 'fuel-type',vars=["wheel-base", "length" , "height" , "width"] )
plt.show()
```



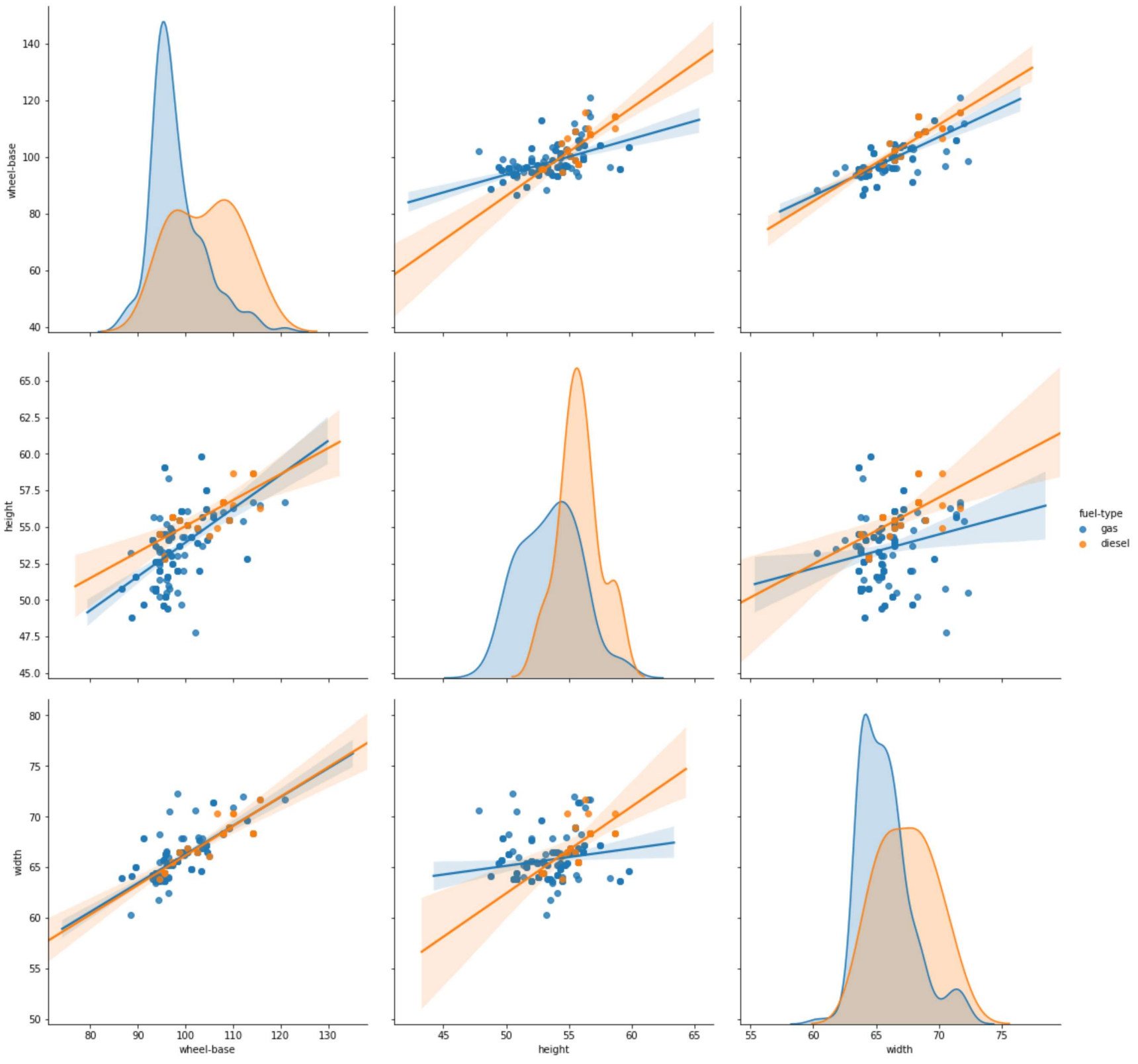
In [130…

```
# Defining the y_vars also

# Use different markers for each level of the hue variable
sns.pairplot(auto,hue = 'fuel-type',x_vars=["wheel-base", "length"] , y_vars=["height" , "width"],markers= ['o' , 'D' ])
plt.show()
```



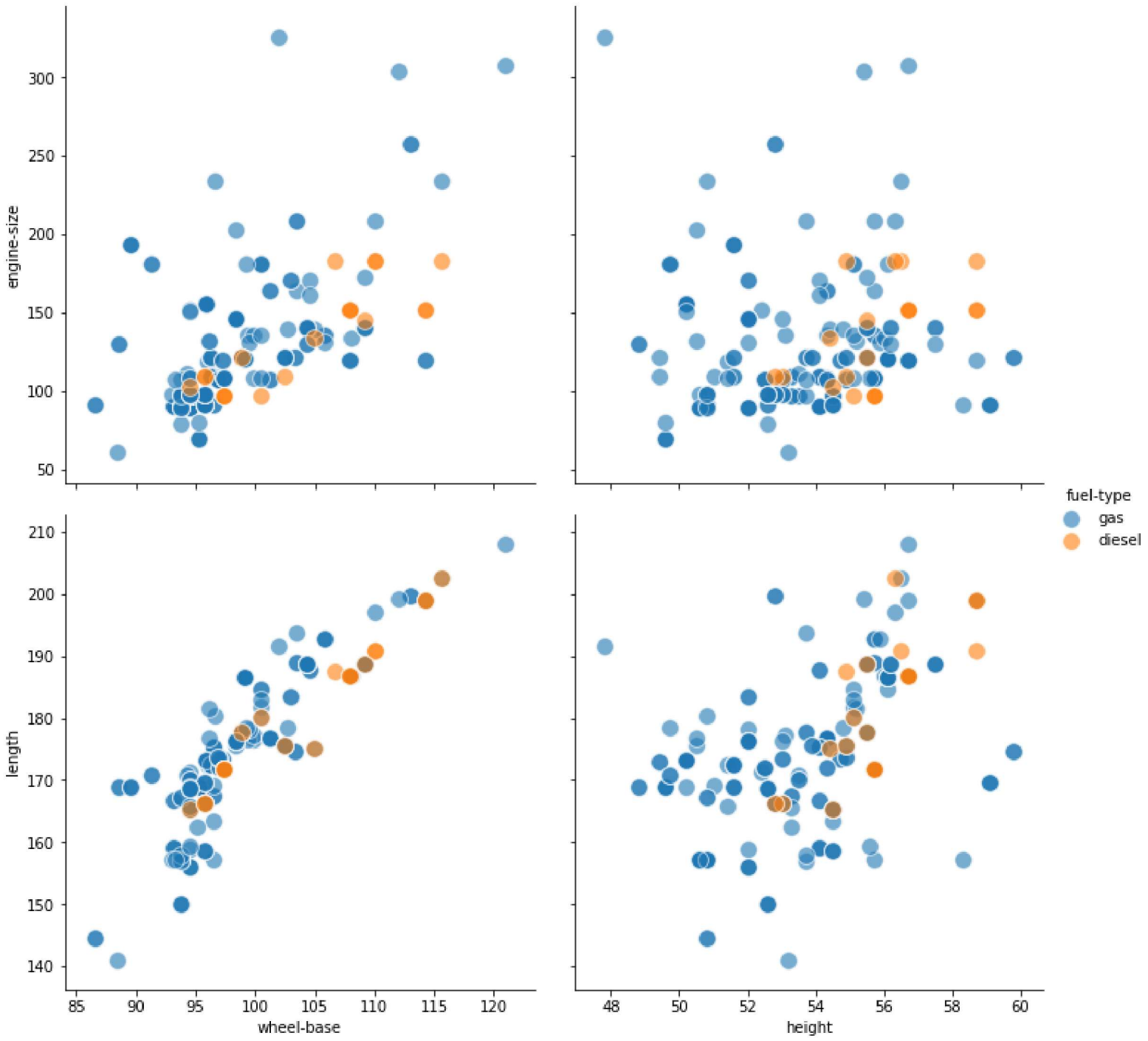
In [131…

```
# Fit linear regression models to the scatter plots
sns.pairplot(auto,hue = 'fuel-type',vars=["wheel-base" , "height" , "width"] , kind="reg",height=5, aspect=1)
plt.show()
```



In [133…

```
#Adjusting the line width and the height
sns.pairplot(auto,hue = 'fuel-type',x_vars=["wheel-base" , "height"] ,y_vars=["engine-size" , "length"] ,plot_kws=dict(
plt.show()
```

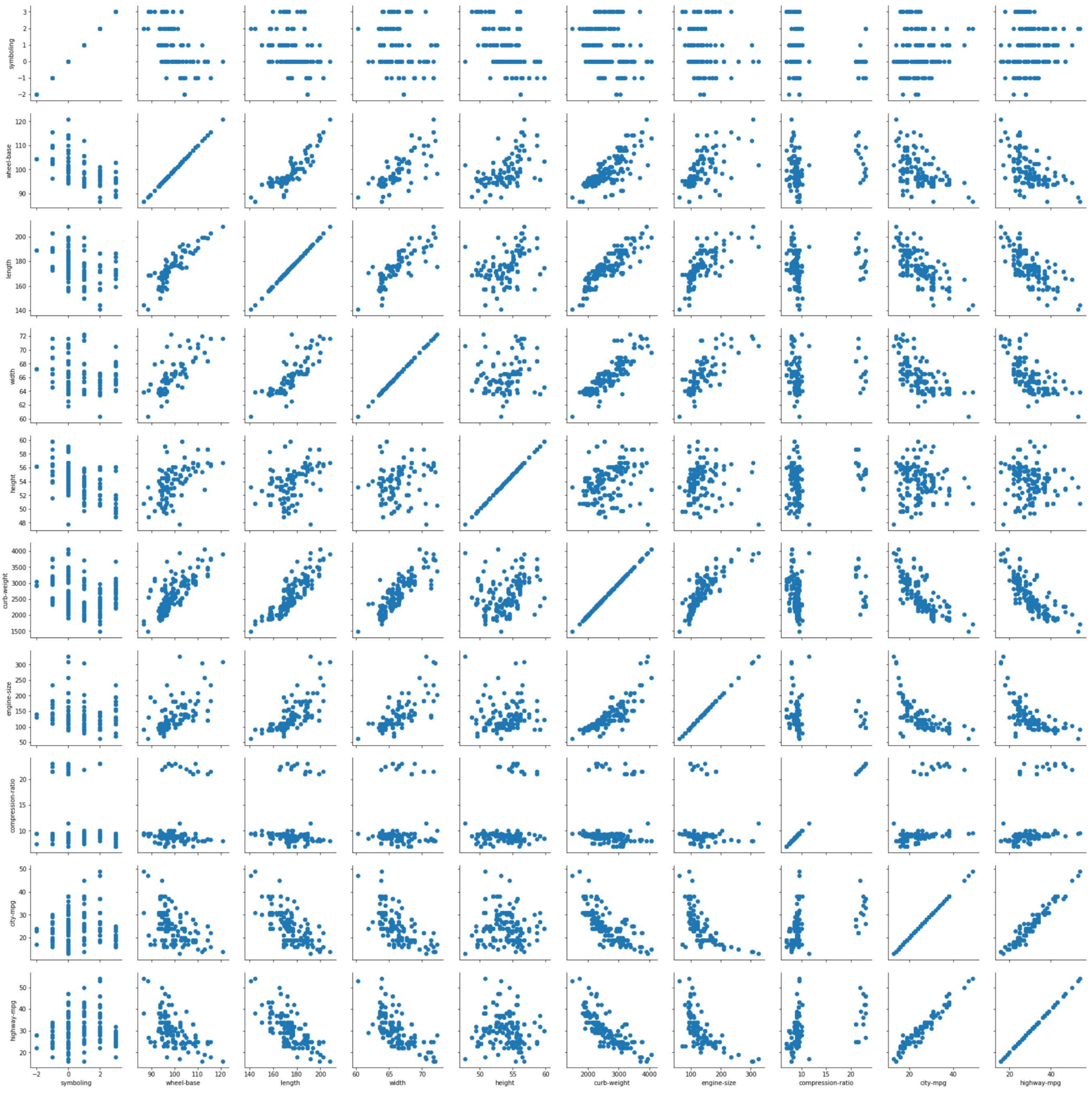


17. Pair Grid

Pair Grid is a Subplot grid for plotting pairwise relationships in a dataset. Different axes-level plotting functions can be used to draw bivariate plots in the upper and lower triangles, and the the marginal distribution of each variable can be shown on the diagonal.

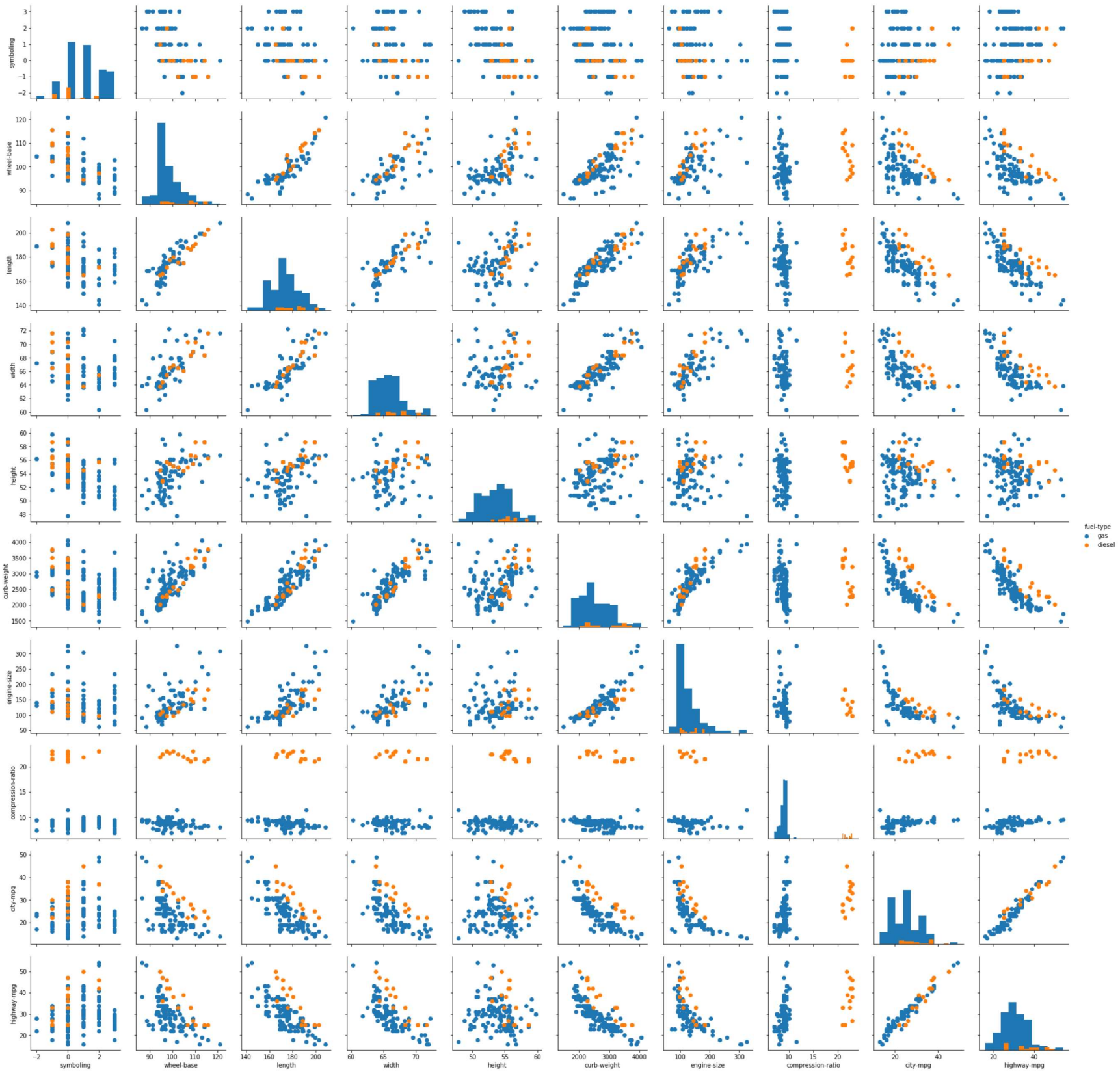
In [134…

```
#scatterplot for each pairwise relationship
g = sns.PairGrid(auto)
g = g.map(plt.scatter)
plt.show()
```



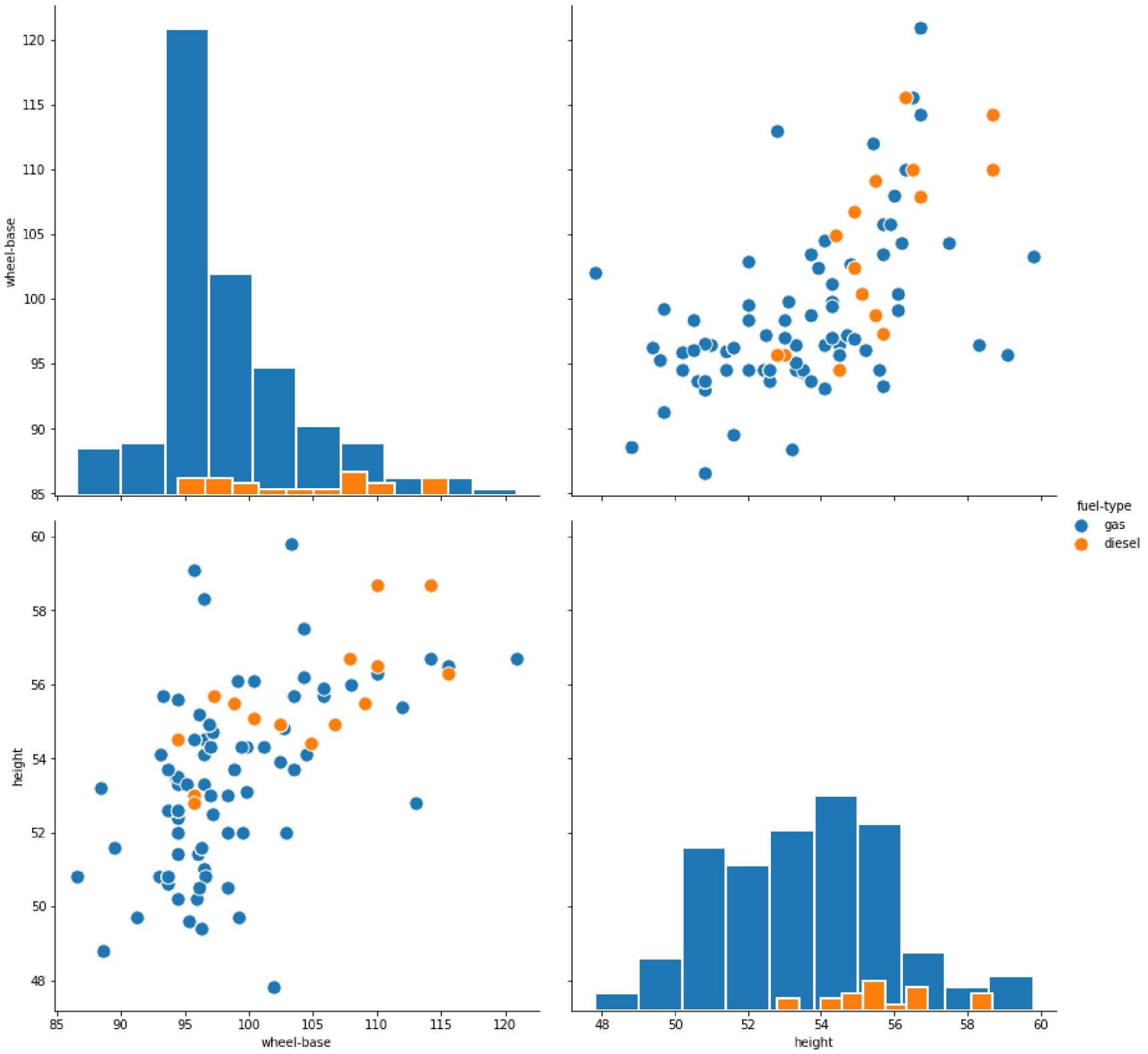
In [136…

```
# Show groups with different colors using "hue"
g = sns.PairGrid(auto , hue='fuel-type')
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
plt.show()
```



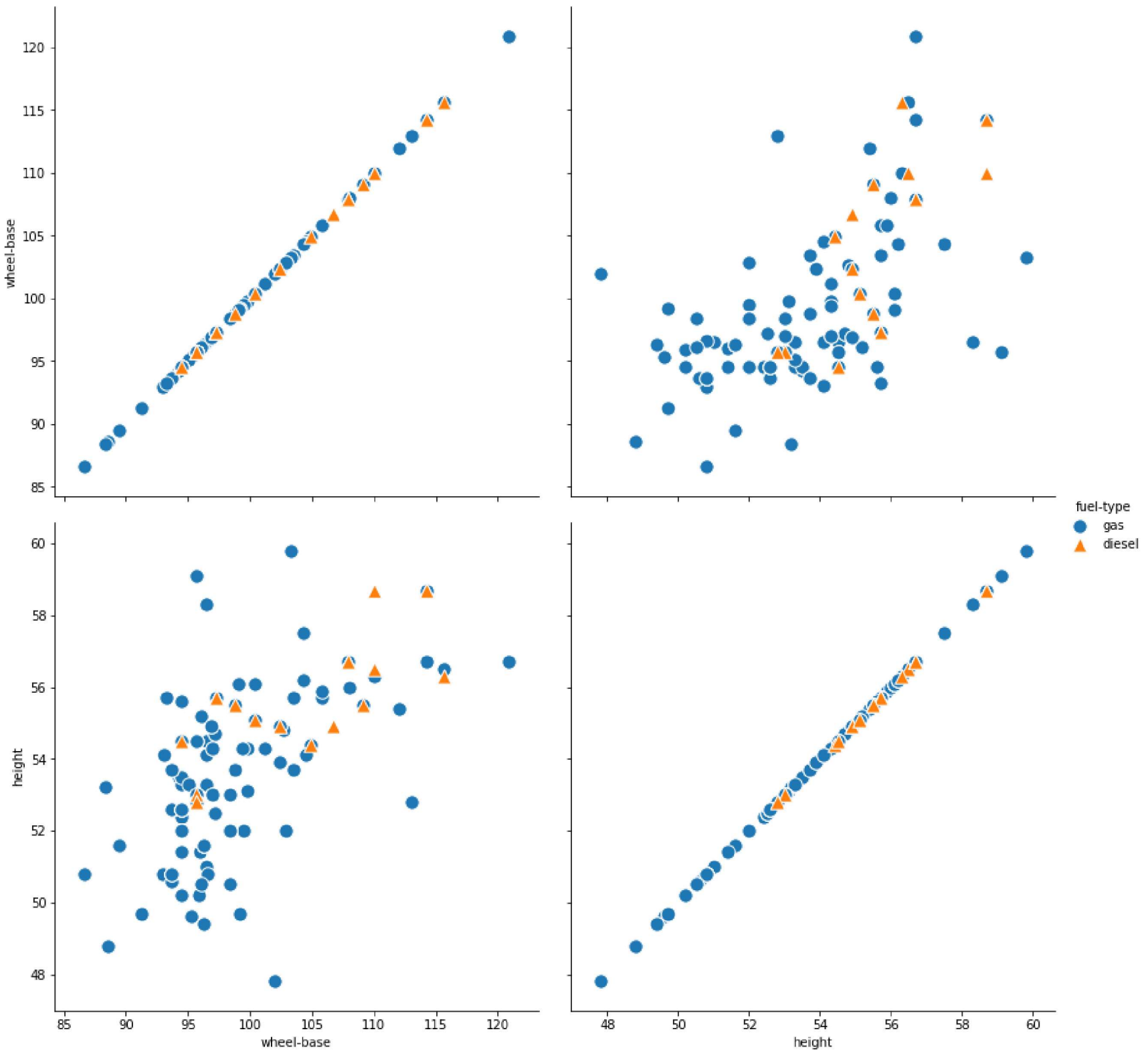
In [140…

```
# Plot a subset of variables
g = sns.PairGrid(auto , hue='fuel-type' ,x_vars=["wheel-base" , "height"],y_vars=["wheel-base" , "height"],height=6, as
g = g.map_offdiag(plt.scatter , edgecolor="w", s=130)
g = g.map_diag(plt.hist , edgecolor ='w', linewidth=2)
g = g.add_legend()
plt.show()
```



In [141…

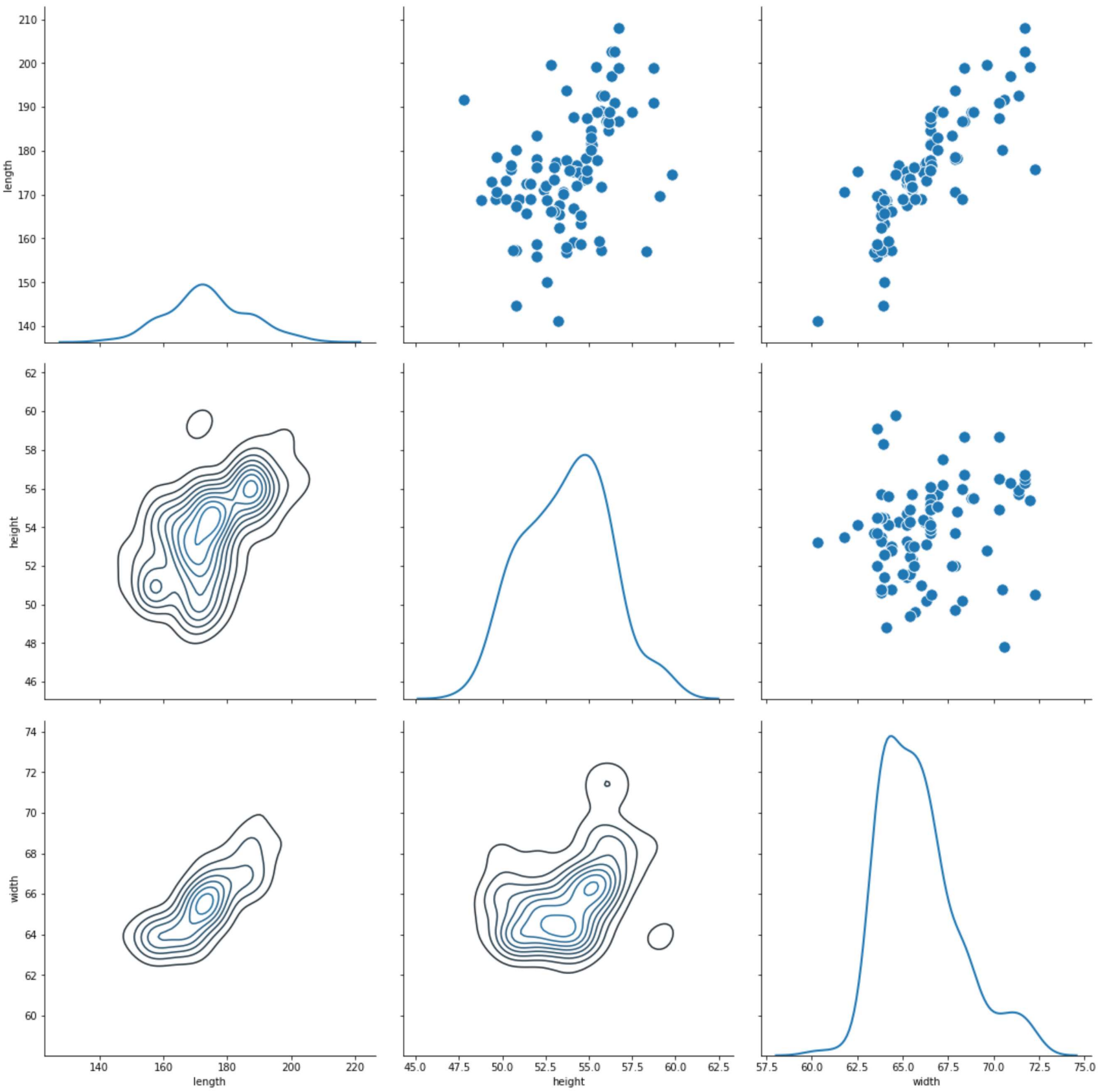
```
g = sns.PairGrid(auto , hue='fuel-type' ,x_vars=["wheel-base" ,"height"],y_vars=["wheel-base" , "height"],height=6, asp
g = g.map(plt.scatter , edgecolor="w", s=130)
g = g.add_legend()
plt.show()
```



In [142…

```
# Using different plots on the upper and lower triangles
g = sns.PairGrid(auto ,vars=["length" , "height" , "width"],height=5, aspect=1)
g = g.map_upper(sns.scatterplot , edgecolor="w", s=130)
g = g.map_lower(sns.kdeplot)
g = g.map_diag(sns.kdeplot , lw= 2)
g = g.add_legend()
plt.show()
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/legend.py:449: UserWarning: The handle <matplotlib.patches.Patch obje
ct at 0x7bf629154978> has a label of '_nolegend_' which cannot be automatically added to the legend.
  'legend.'.format(handle, label))
```

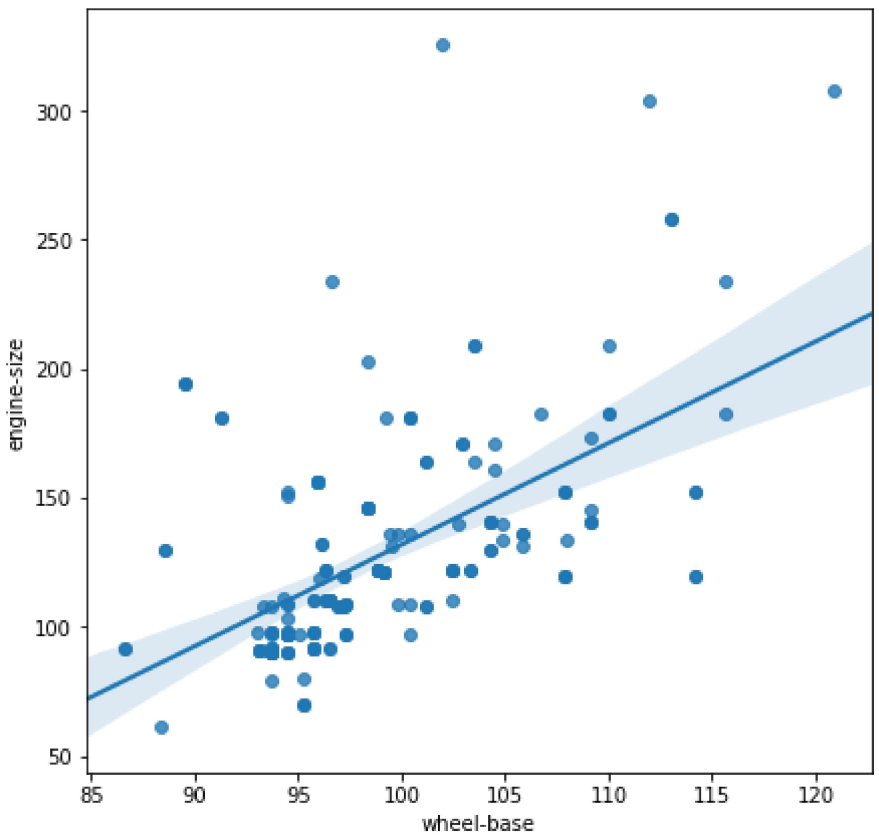


18. Regression plots

lmplot() is one of the most widely used function to quickly plot the Linear Relationship b/w 2 variables. Regression plot is used to plot data and a linear regression model fit.

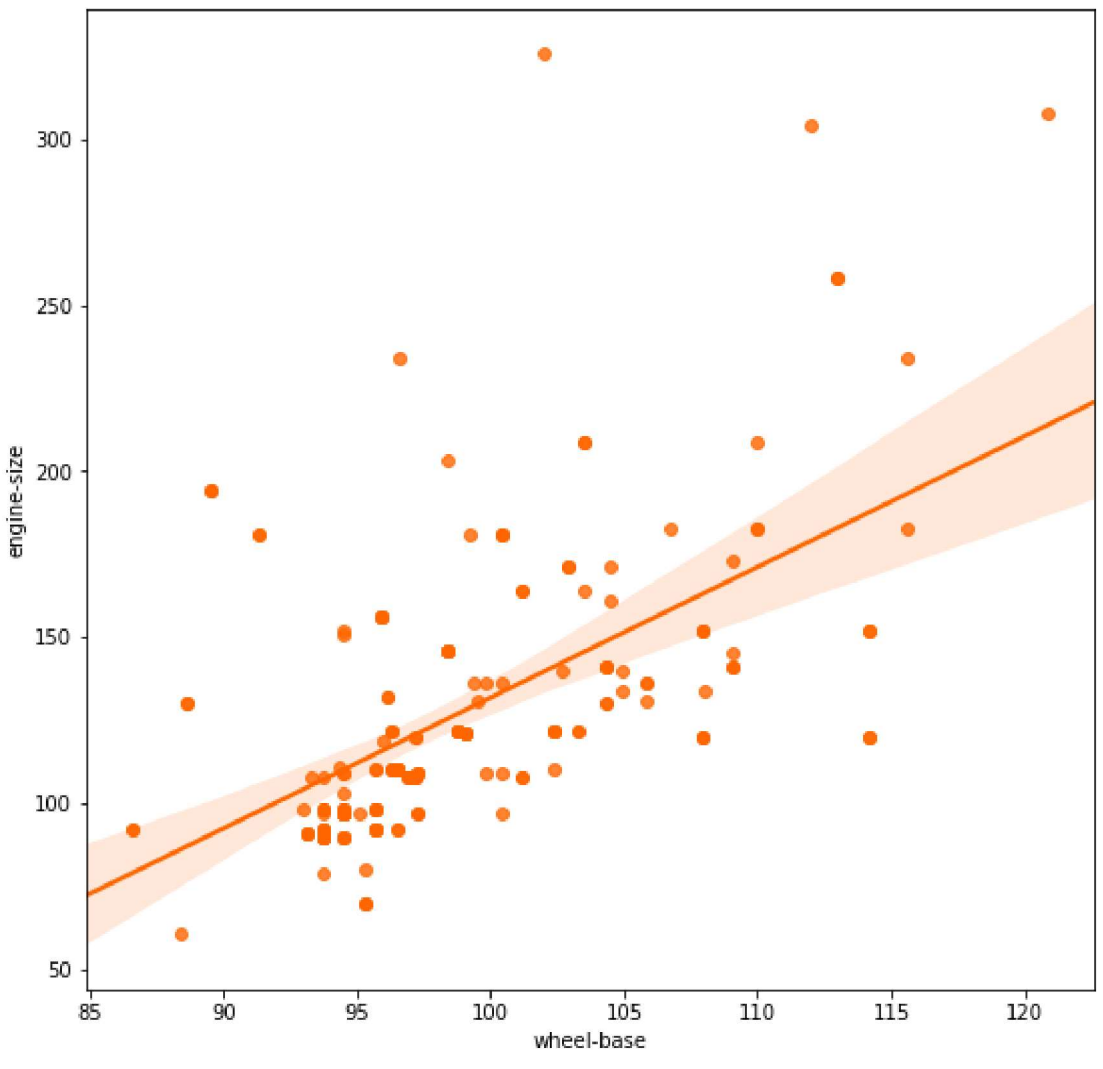
In [143…
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

In [144…
```
# Plot the relationship between two variables
plt.figure(figsize=(7,7))
sns.regplot(x=auto["wheel-base"] , y=auto["engine-size"])
plt.show()
```



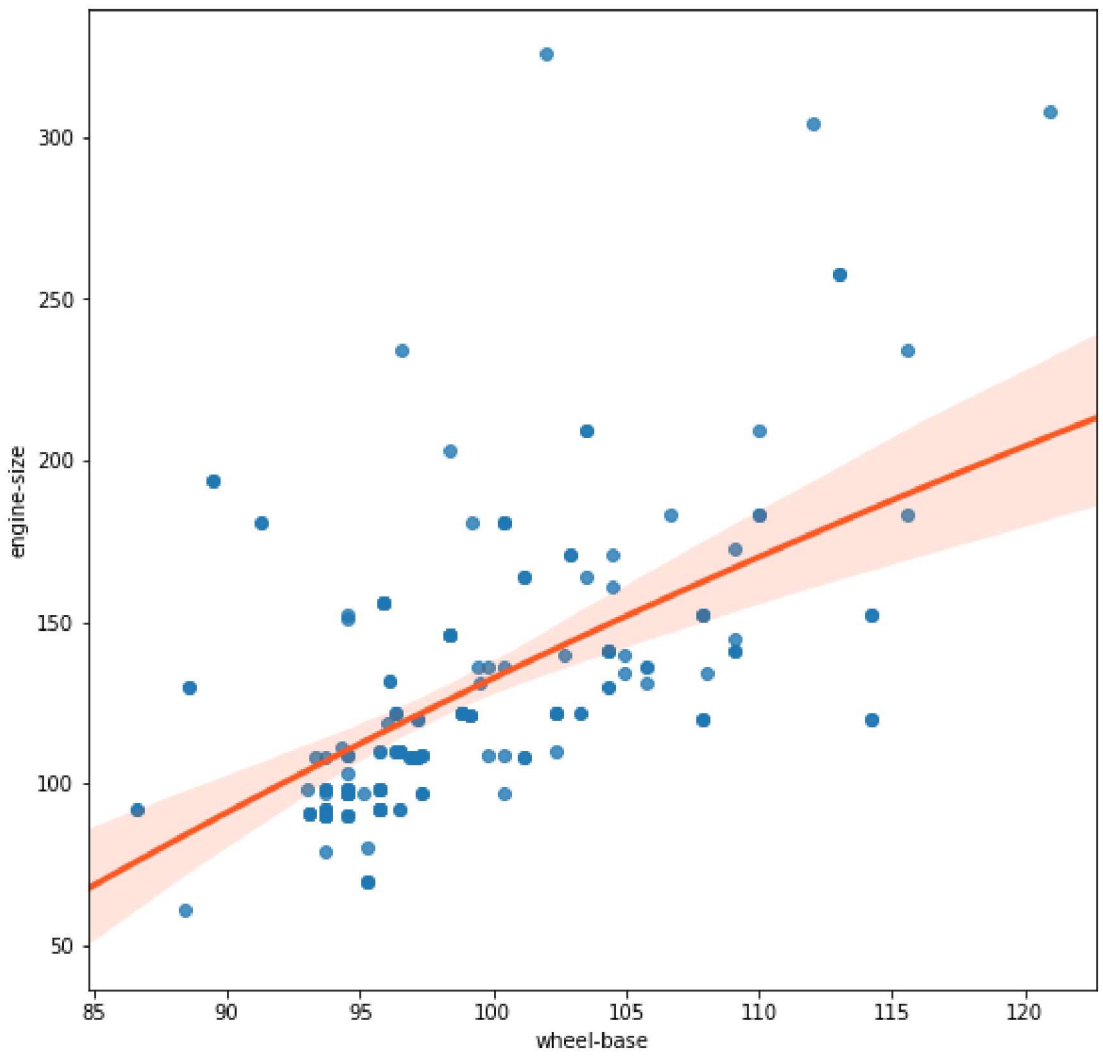
In [145…

```
# Plot the relationship between two variables and use a different color
plt.figure(figsize=(9,9))
sns.regplot(x=auto["wheel-base"] , y=auto["engine-size"] , color='#FF6600')
plt.show()
```



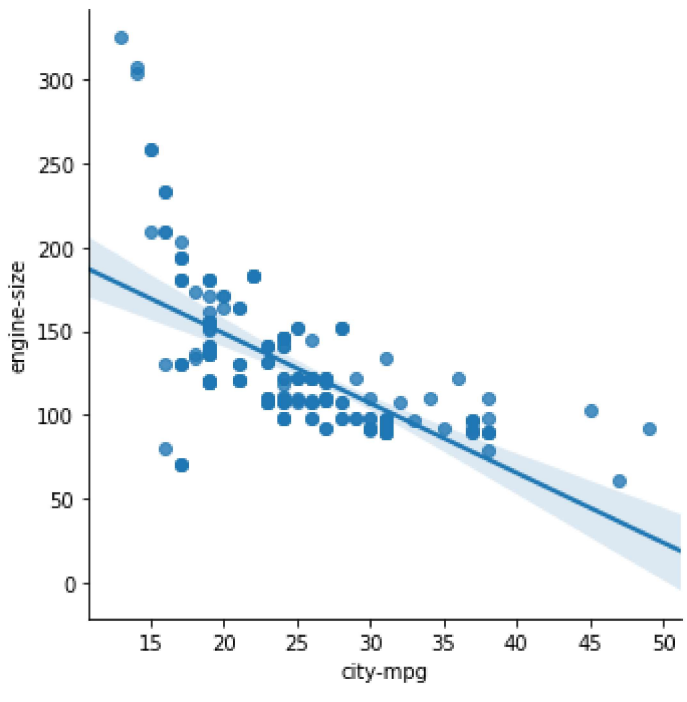
In [149…

```
# Change the color and width of regression line -> line_kws={"color":"#FF5722","lw":3}
plt.figure(figsize=(9,9))
sns.regplot(x=auto["wheel-base"] , y=auto["engine-size"] , logx=True , line_kws={"color":"#FF5722","lw":3})
plt.show()
```



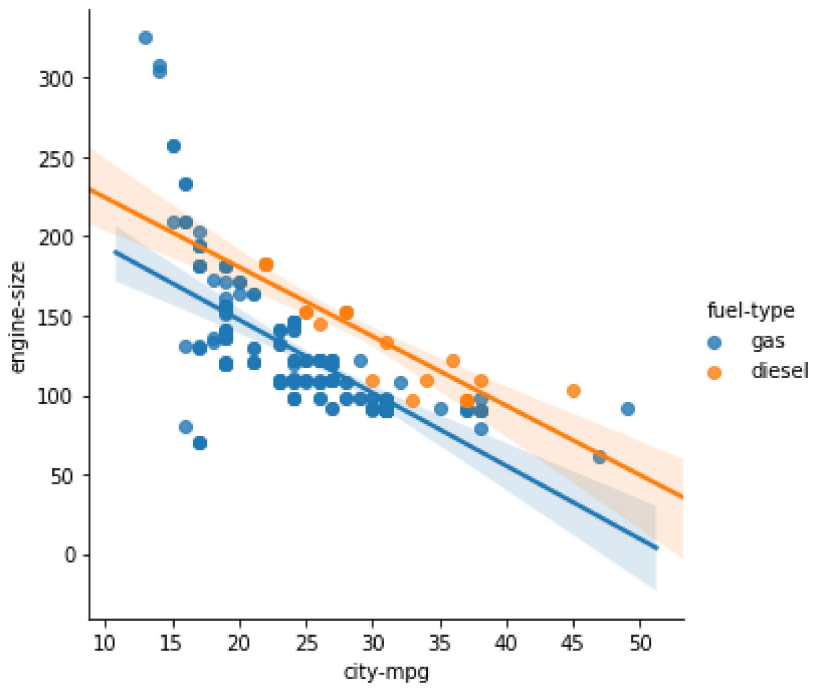
In [150…
```
#lmplot
sns.lmplot(x="city-mpg", y="engine-size",data=auto)
```

Out[150]:
```
<seaborn.axisgrid.FacetGrid at 0x7bf63365e5c0>
```

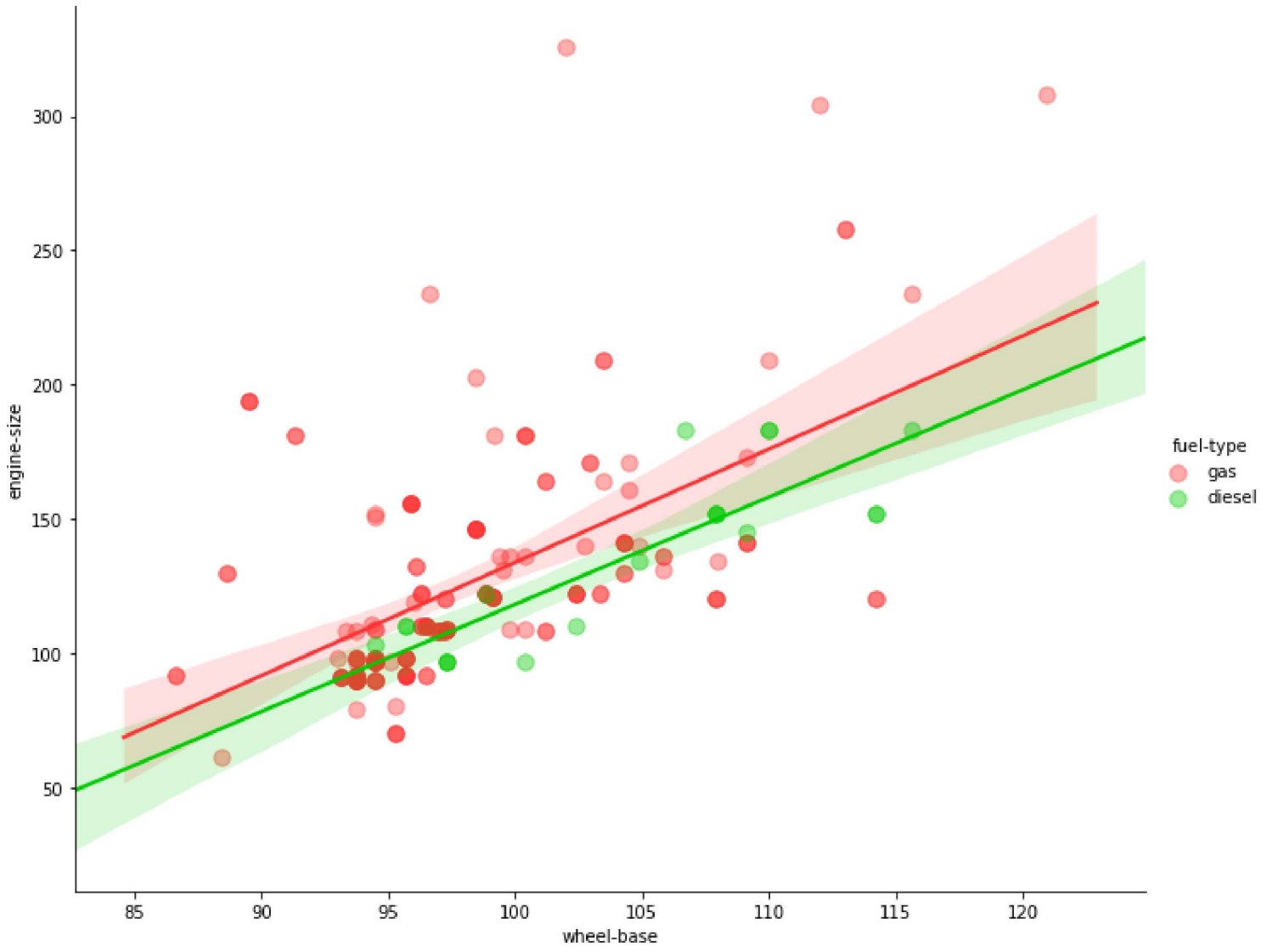


In [151…
```
#Fit a regression model for a categorical variable
sns.lmplot(x="city-mpg", y="engine-size",hue="fuel-type", data=auto)
```

Out[151]:
```
<seaborn.axisgrid.FacetGrid at 0x7bf6310294e0>
```

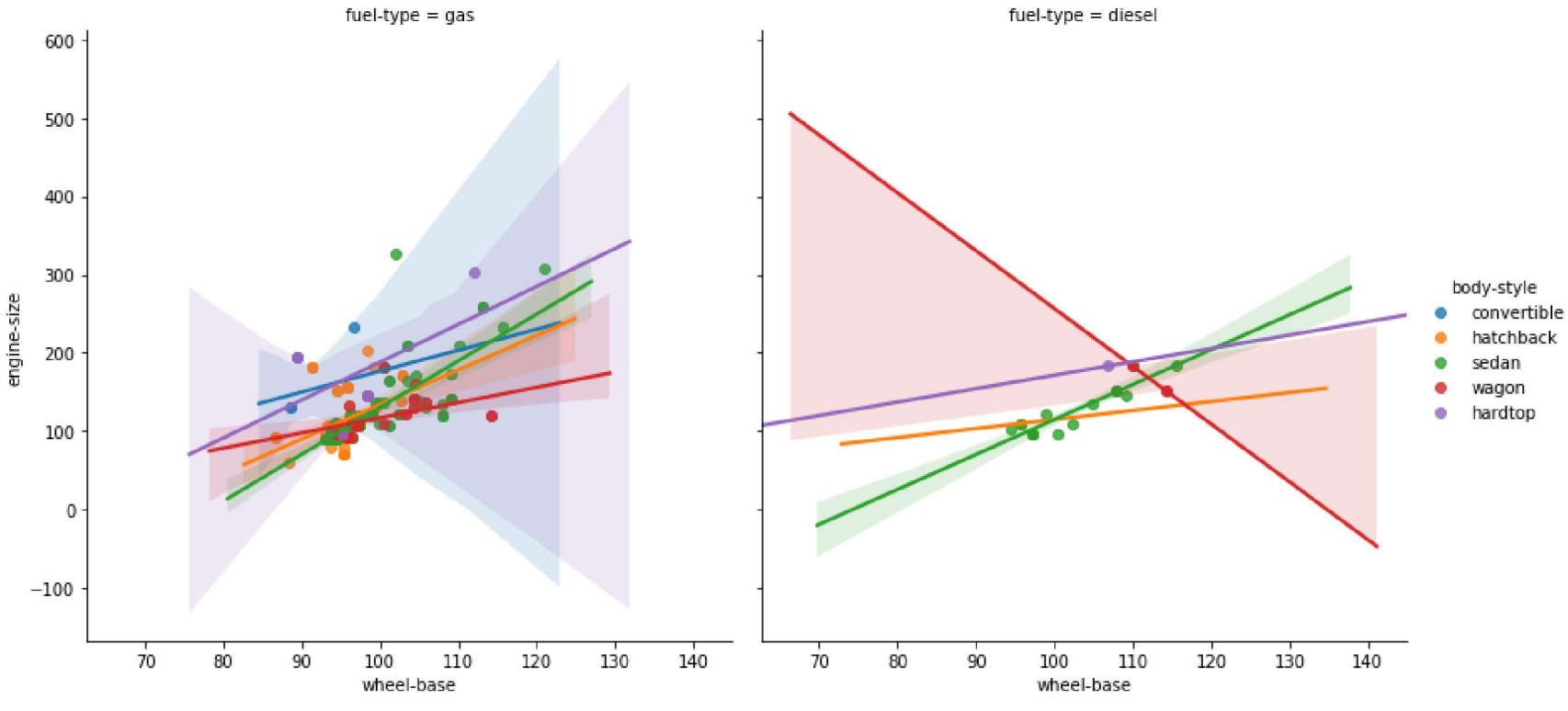


In [152…
```
sns.lmplot(x="wheel-base" , y="engine-size" , hue="fuel-type" , data=auto , height=8,aspect=1.2 ,scatter_kws ={'s':90 ,
plt.show()
```



In [153…
```
# Plot the levels of the third variable across different columns
sns.lmplot(x="wheel-base", y="engine-size", hue="body-style", col="fuel-type", data=auto ,height=6,aspect=1)
```

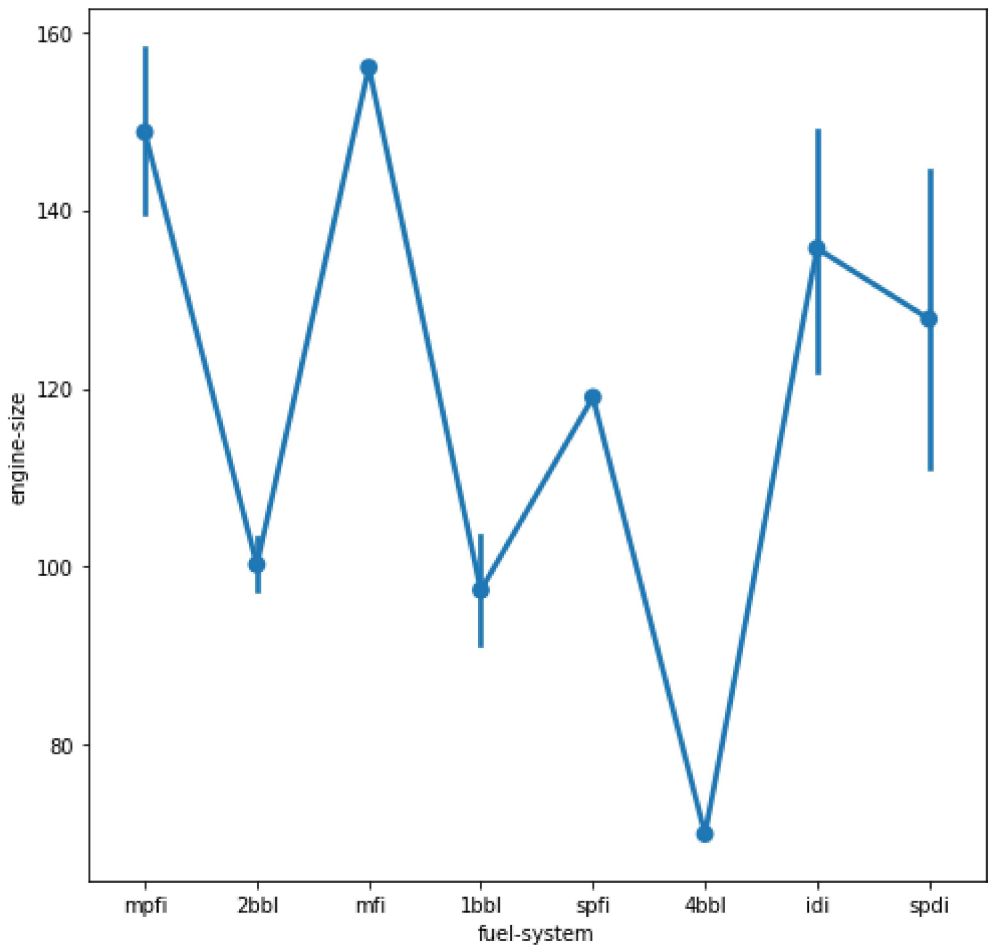
Out[153]: <seaborn.axisgrid.FacetGrid at 0x7bf62b7215c0>



19.Point plots

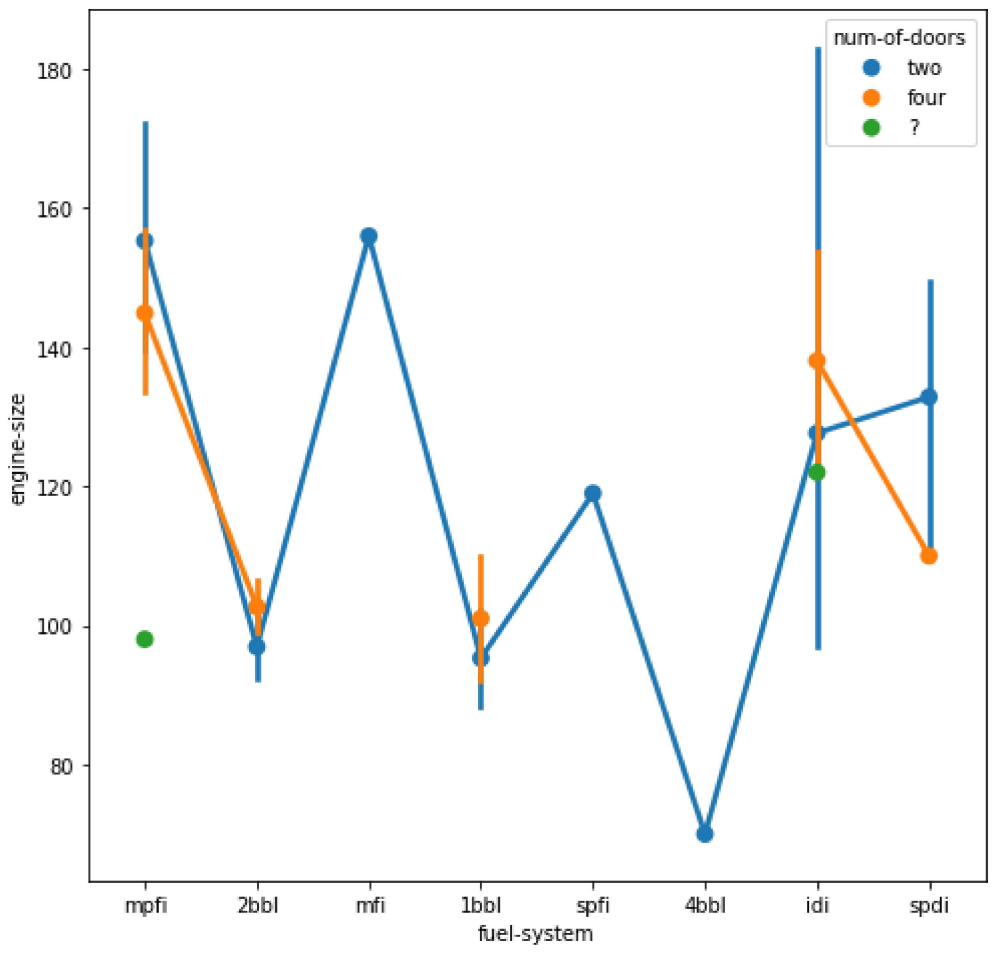
An alternative style for visualizing the same information is offered by the pointplot() function. This function also encodes the value of the estimate with height on the other axis, but rather than show a full bar it just plots the point estimate and confidence interval. Additionally, pointplot connects 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 [154…
```
plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'])
plt.show()
```



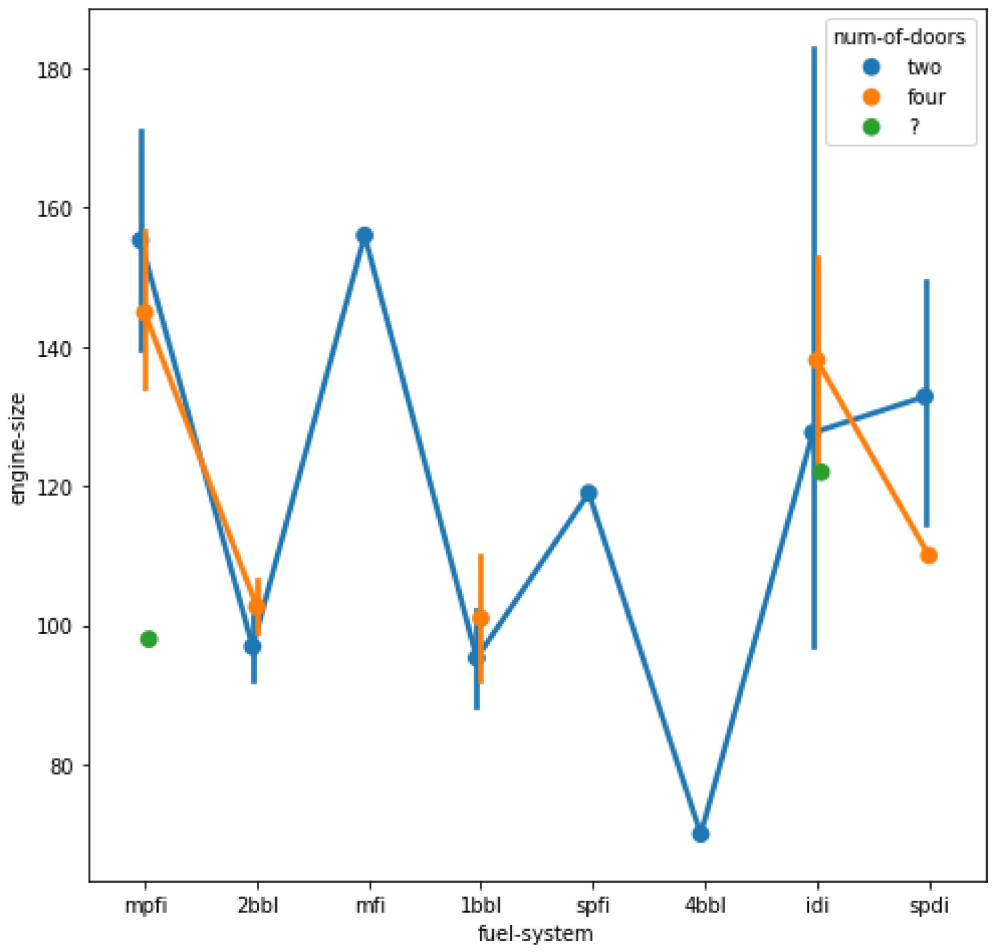
In [155…

```
plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'], hue=auto['num-of-doors'])
plt.show()
```



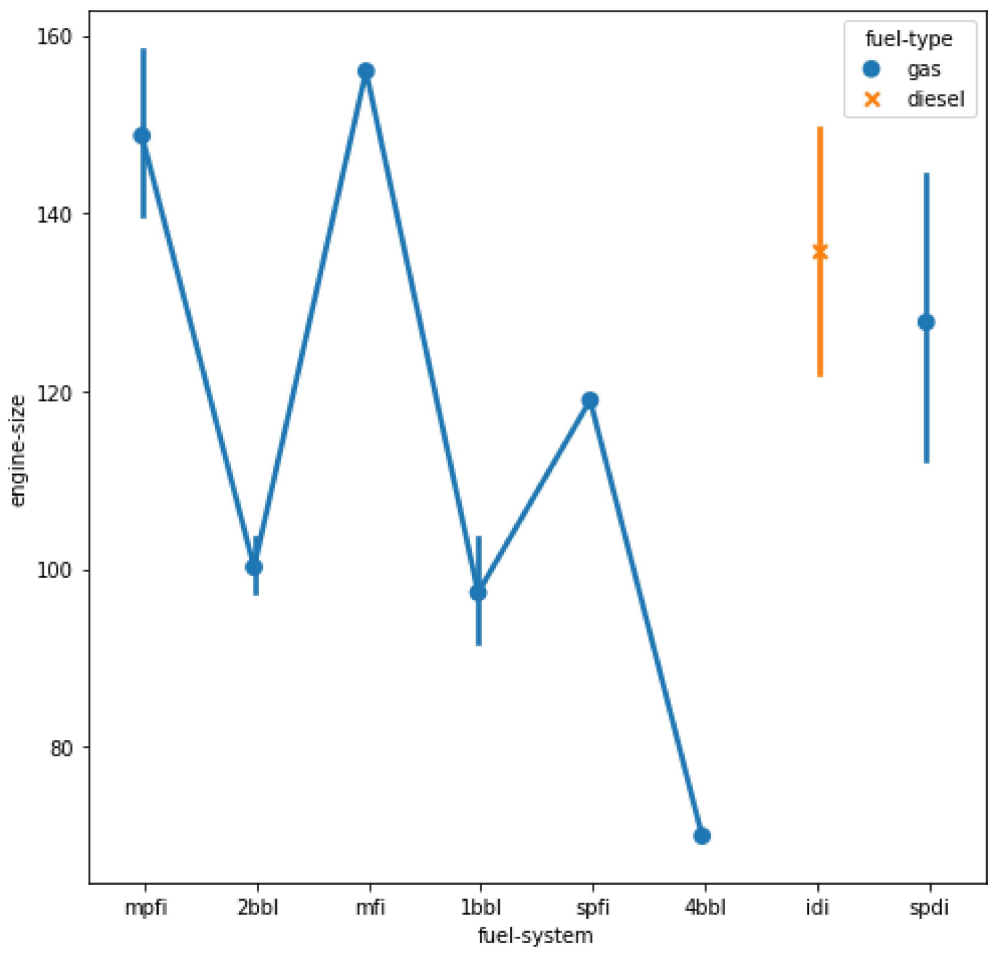
In [156…

```
#Separate the points for different hue levels along the categorical axis:

plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'], hue=auto['num-of-doors'],dodge=True)
plt.show()
```



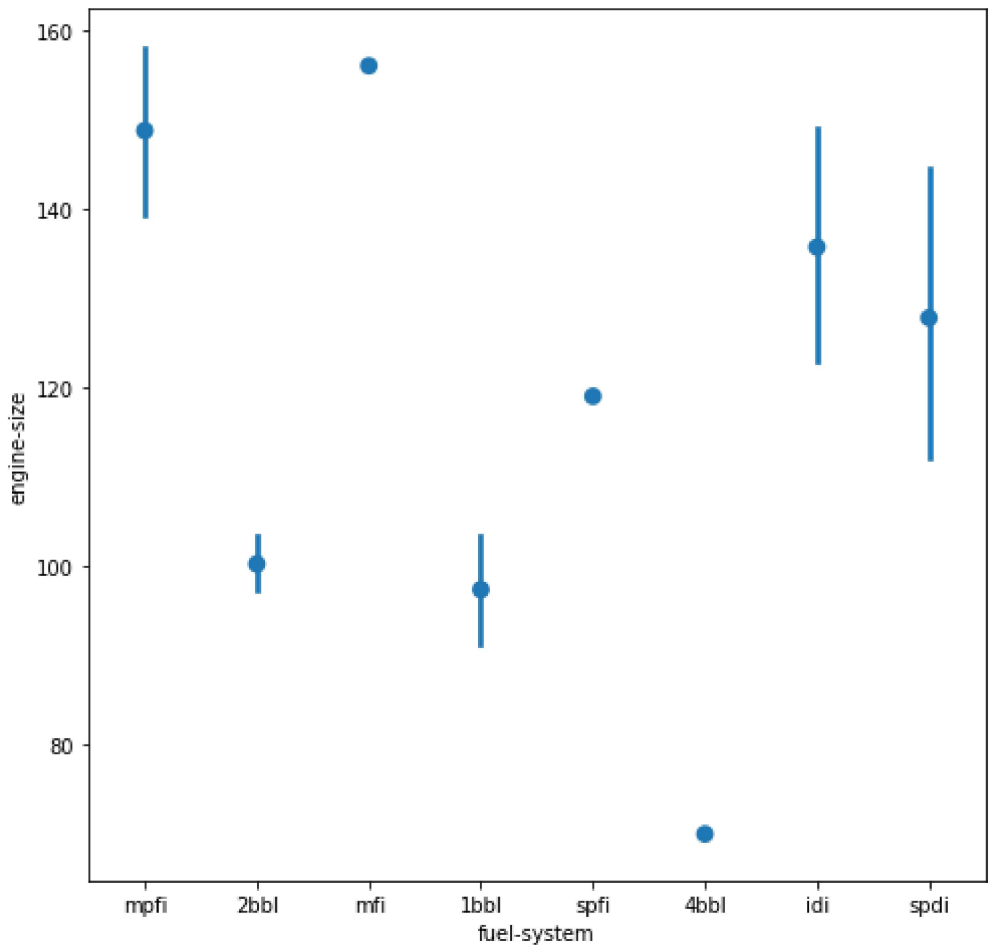
In [157…

```
# use a different markers

plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'], hue=auto['fuel-type'],dodge=True,markers=["o", "x"],linestyles=
plt.show()
```



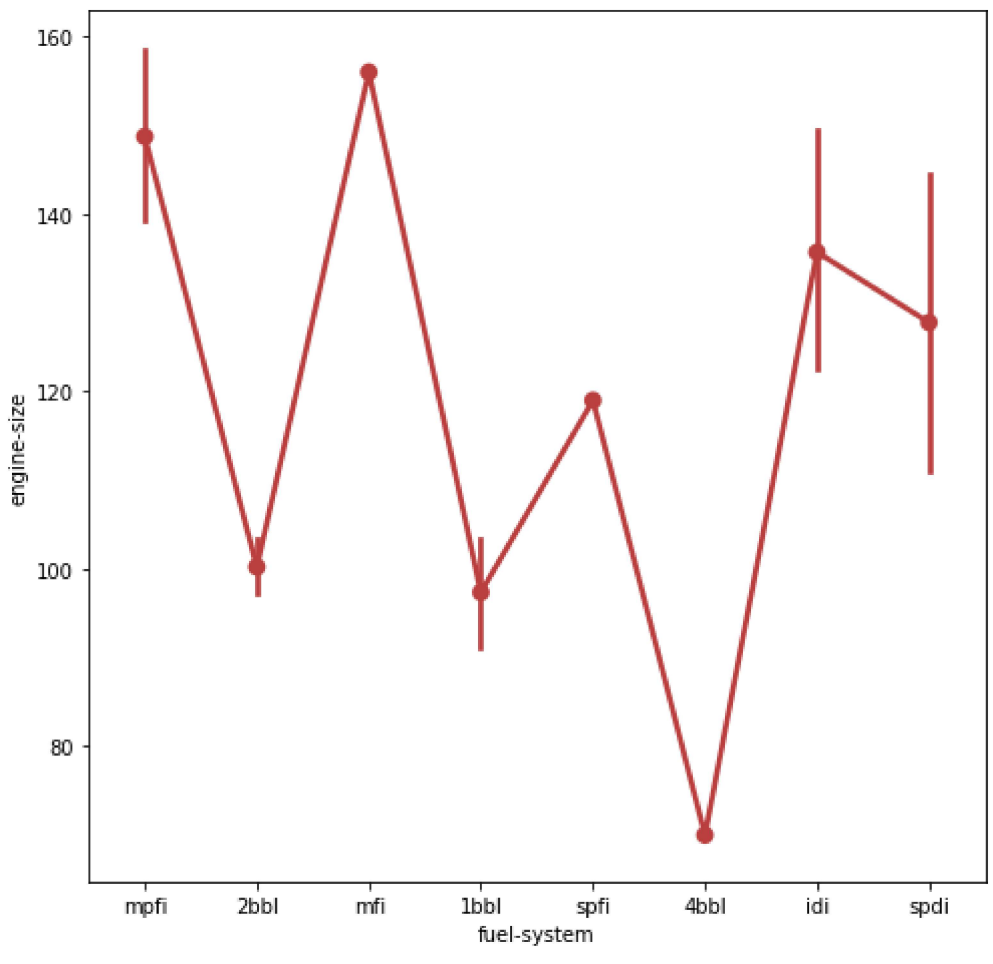
In [158…

```
#Don’t draw a line connecting each point:
plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'],join=False)
plt.show()
```



In [159…

```
plt.figure(figsize=(8,8))
sns.pointplot(auto['fuel-system'], auto['engine-size'],color="#bb3f3f")
plt.show()
```



20.Facet Grid

Multi-plot grid for plotting. A FacetGrid can be drawn with up to three dimensions – row, col, and hue.

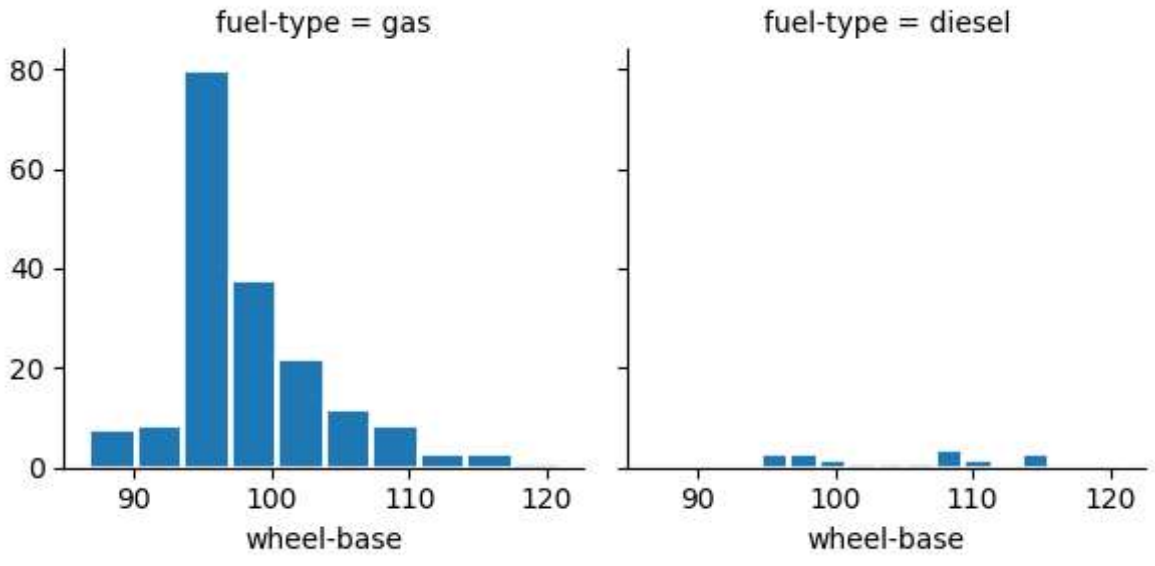
In [161…

```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
plt.rcParams[ 'axes.labelsize'] = 10
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10
```

In [162…

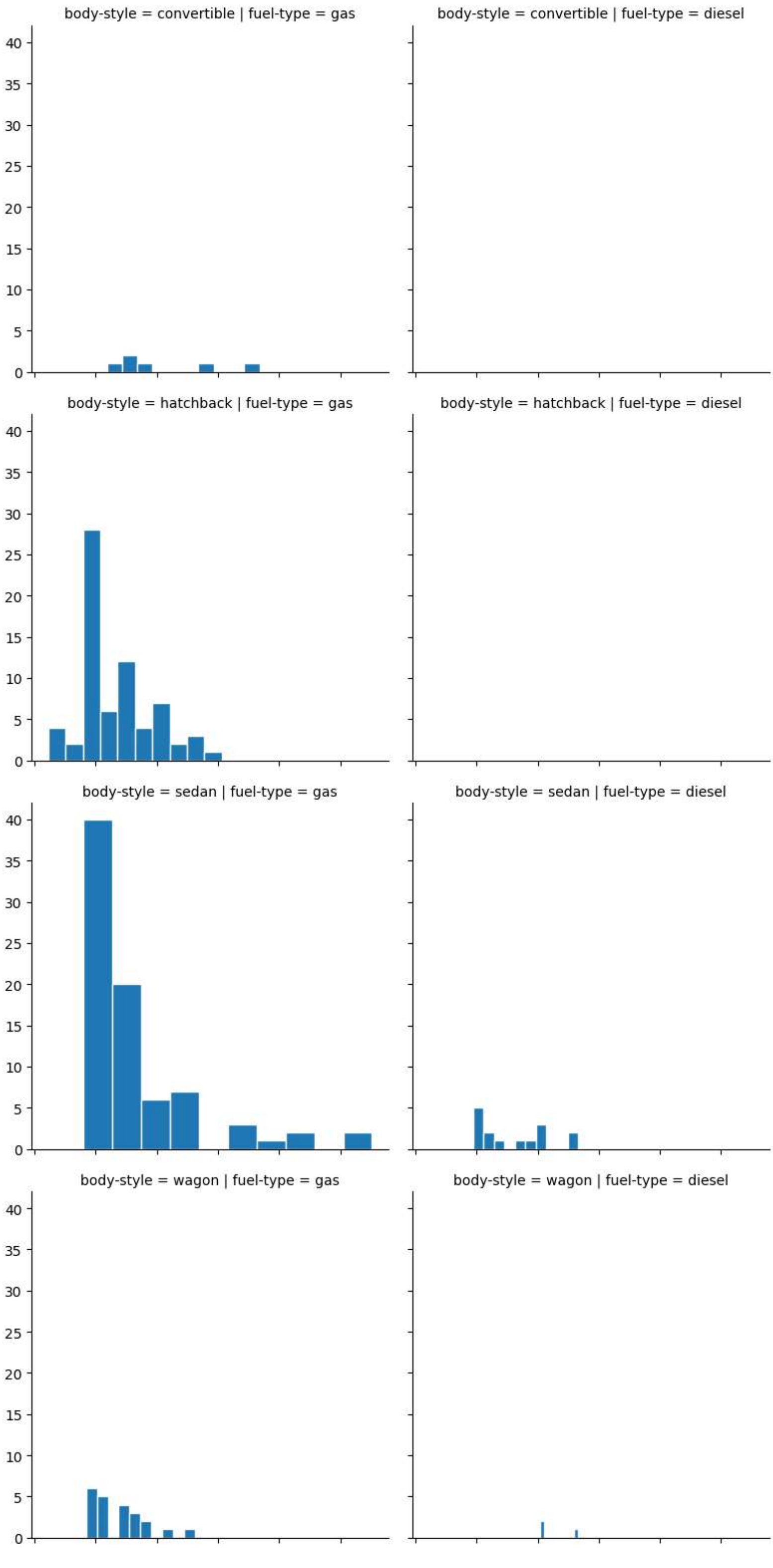
```
# Facet along the columns to show a categorical variables using "col" parameter (univariate)
plt.figure(figsize=(8,8))
g = sns.FacetGrid(auto, col="fuel-type")
g = g.map(plt.hist, "wheel-base", edgecolor ='w', linewidth=2)
plt.show()
```

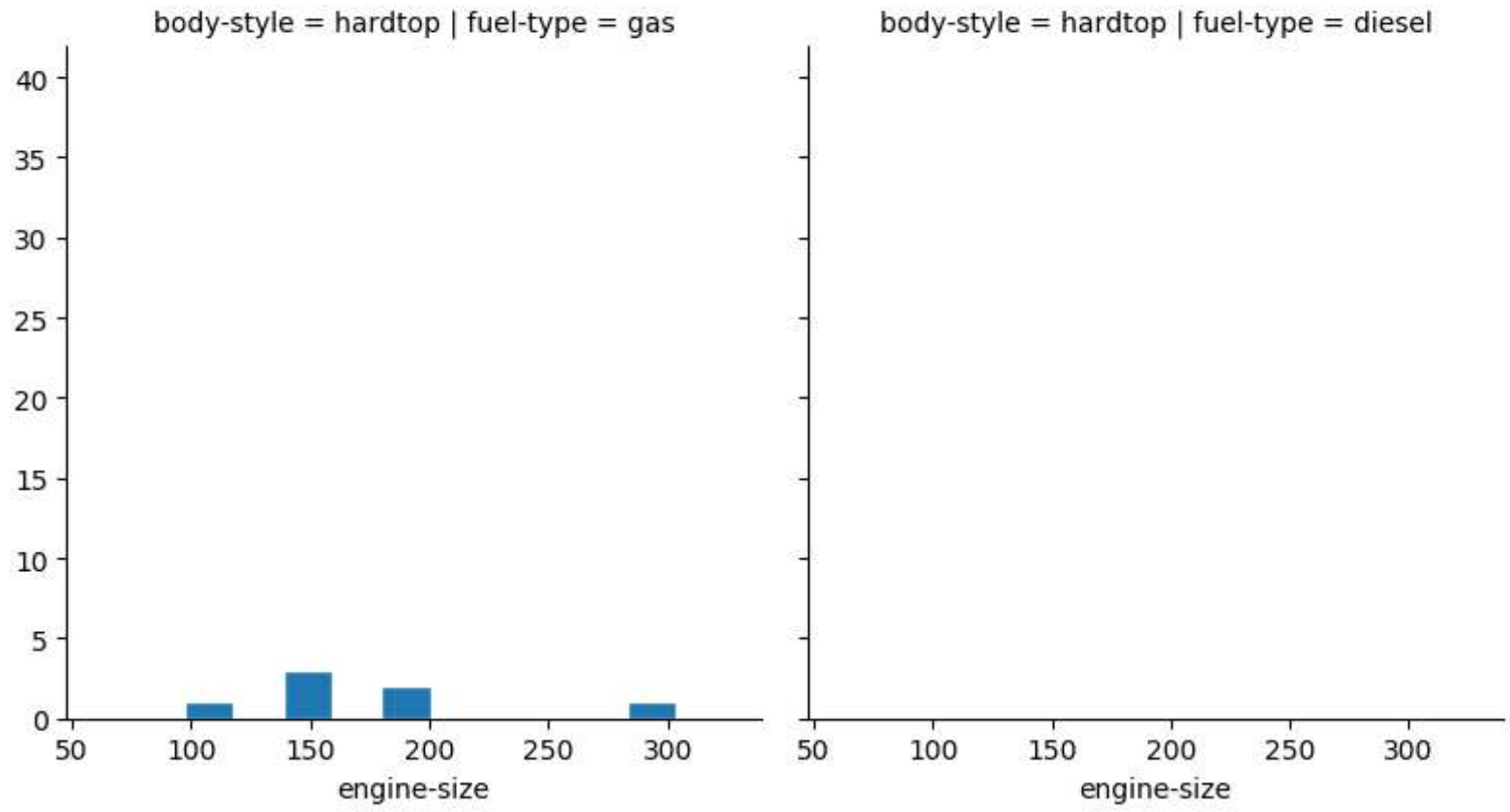
```
<Figure size 800x800 with 0 Axes>
```



In [163…

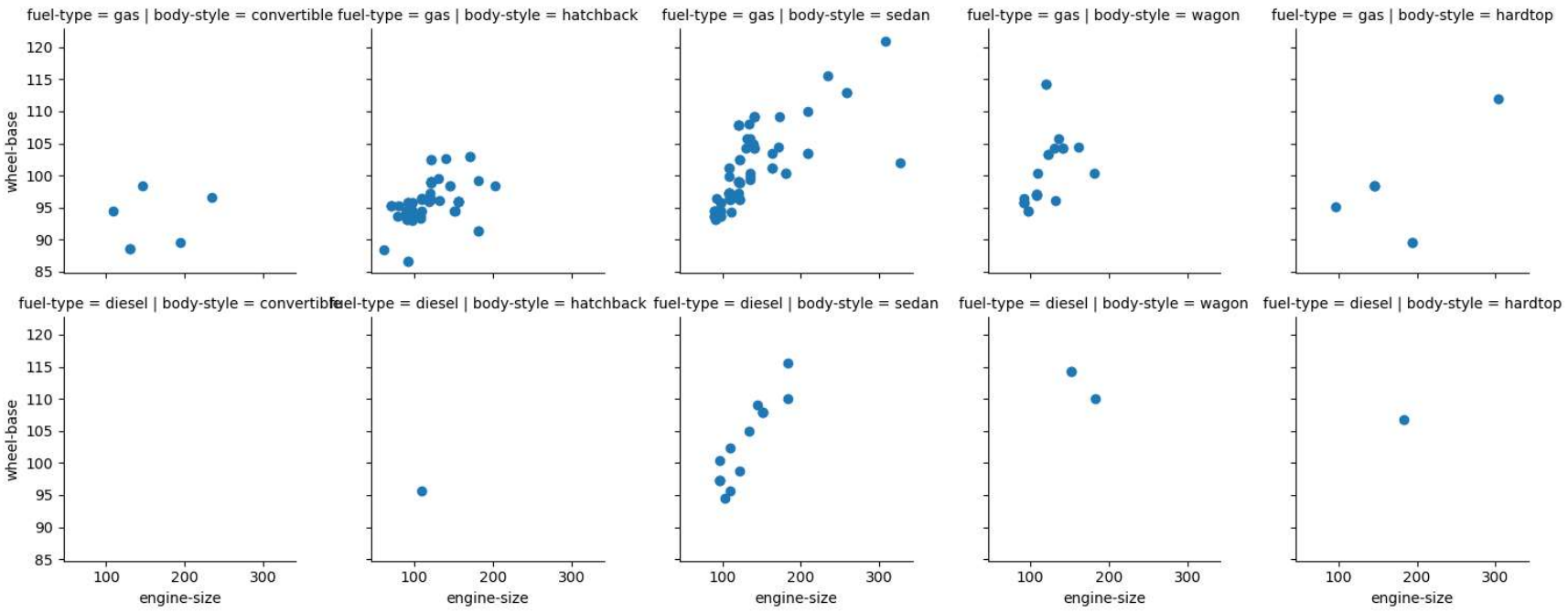
```
# Facet along the columns and rows to show a categorical variables using "col" and "row" pa
g = sns.FacetGrid(auto, col="fuel-type" , row = "body-style" , height=4, aspect=1)
g = g.map(plt.hist, "engine-size",edgecolor ='w', linewidth=1)
```


body-style = convertible | fuel-type = gas
body-style = convertible | fuel-type = diesel
body-style = hatchback | fuel-type = gas
body-style = hatchback | fuel-type = diesel
body-style = sedan | fuel-type = gas
body-style = sedan | fuel-type = diesel
body-style = wagon | fuel-type = gas
body-style = wagon | fuel-type = diesel
0
5
10
15
20
25
30
35
40



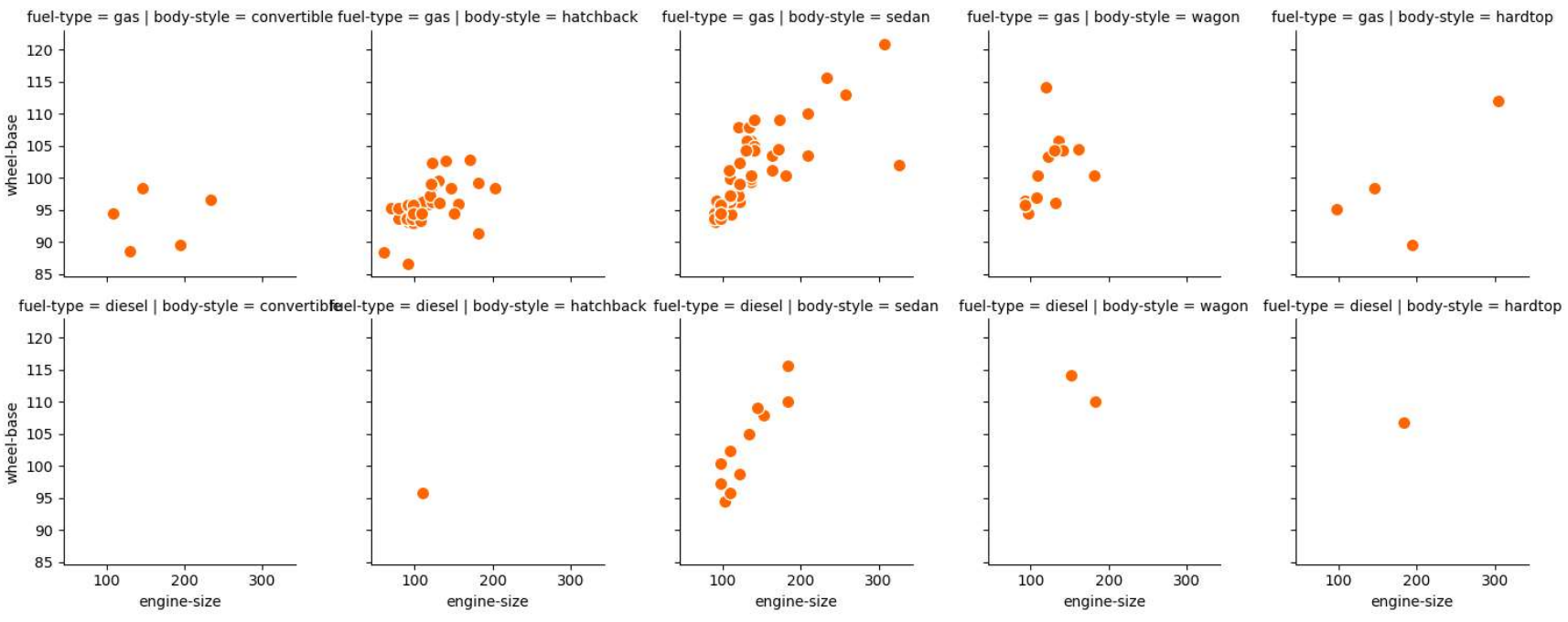
In [165…

```
# Bivariate function on each facet
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , height=3, aspect=1)
g = g.map(plt.scatter, "engine-size" , "wheel-base")
```



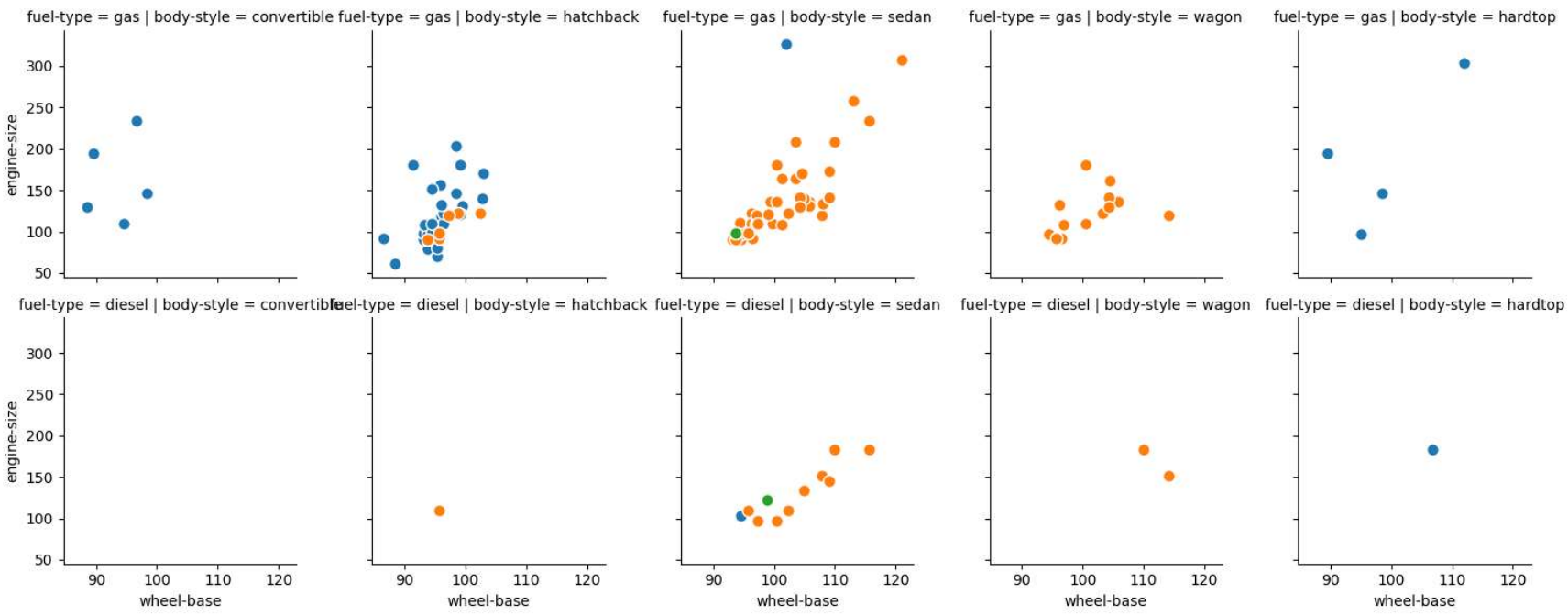
In [166…

```
# Changing size of dots -> s = 80
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , height=3, aspect=1)
g = g.map(plt.scatter, "engine-size" , "wheel-base" , color = '#FF6600',edgecolor="w", s=80)
```



In [167…

```
# Show groups with different colors using "hue"
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , hue="num-of-doors" ,height=3, aspect=1)
g = g.map(plt.scatter, "wheel-base" , "engine-size" , edgecolor="w", s=70)
```

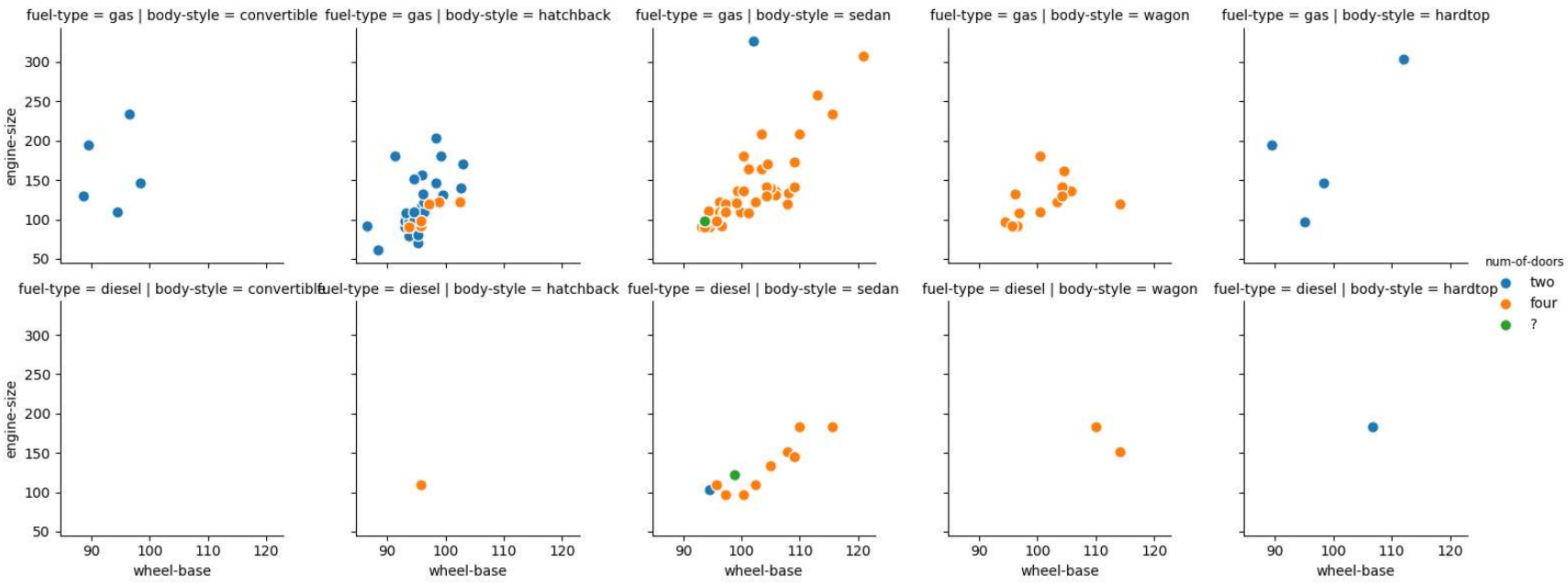


In [168…

```
# Showing legend -> g.add_legend()
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , hue="num-of-doors" ,height=3, aspect=1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",edgecolor="w", s=70)
g.add_legend()
```

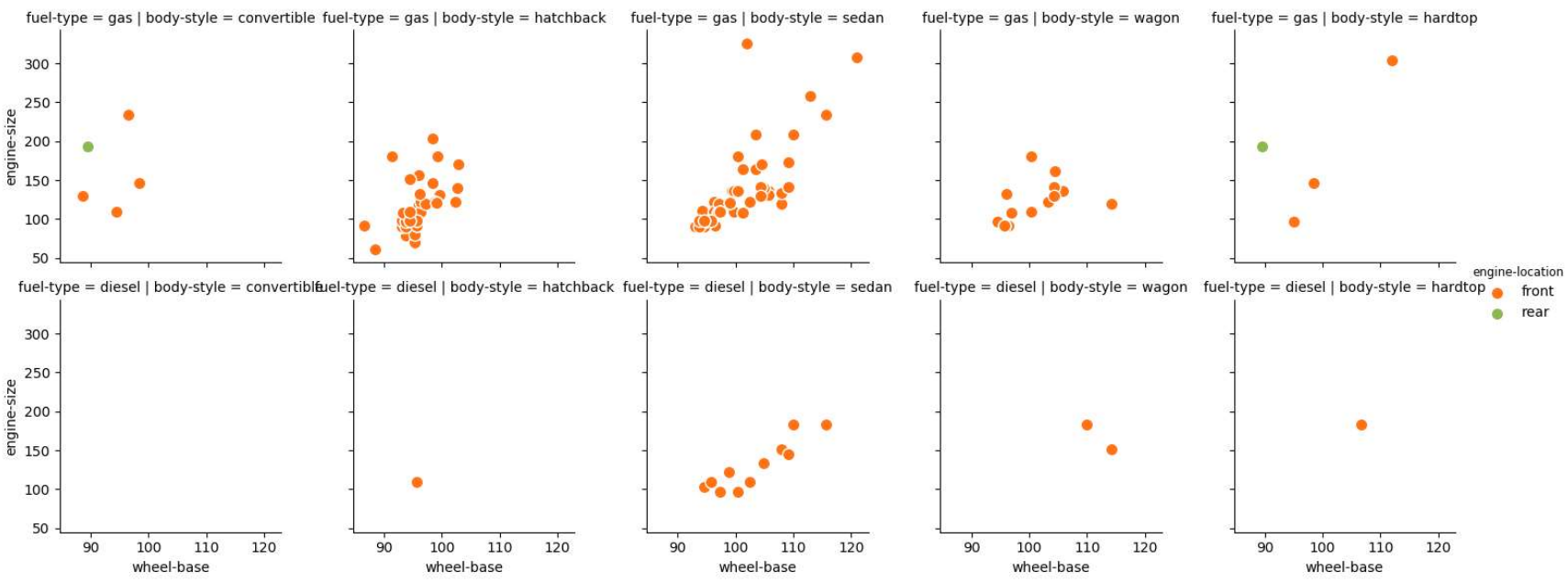
Out[168]:

```
<seaborn.axisgrid.FacetGrid at 0x7bf63351e780>
```



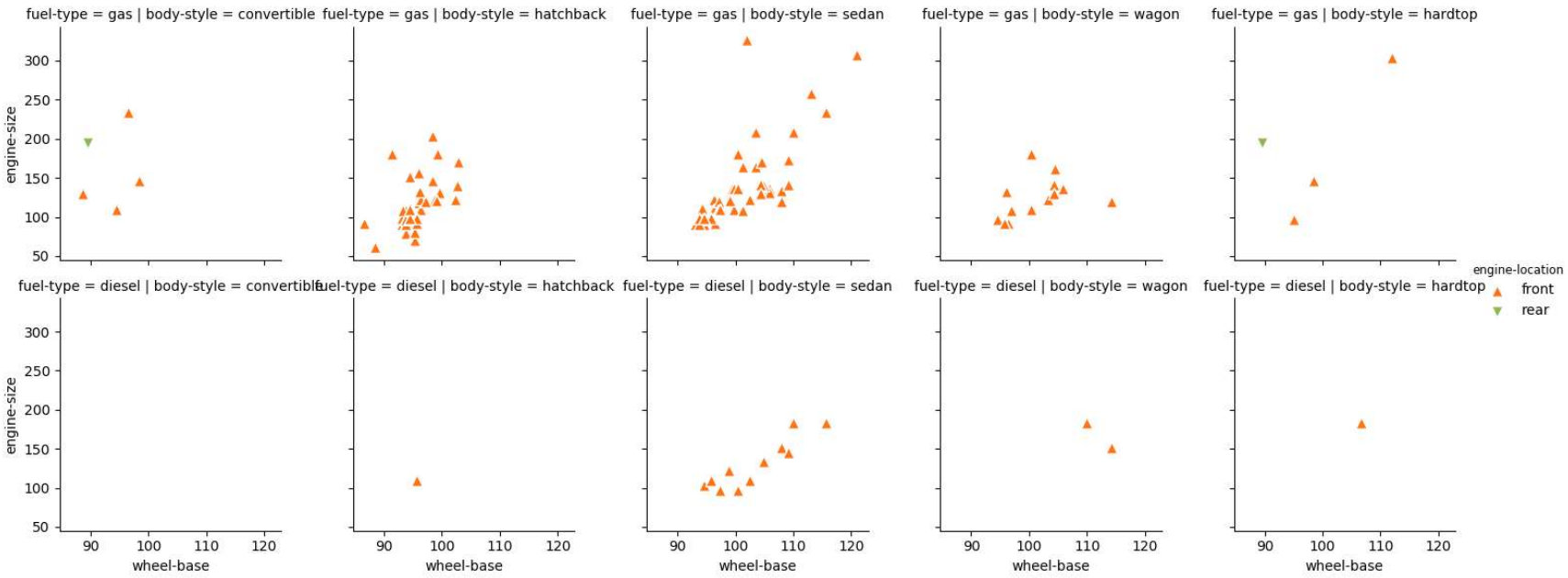
In [170…

```
pal1 = dict(front="#ff7315", rear="#8cba51")
kws = dict(s=80, linewidth=1, edgecolor="w")
kws1 = dict(height=3, aspect=1,palette=pal1)
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , hue="engine-location" , **kws1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",**kws)
g.add_legend()
plt.show()
```



In [171…

```
# Use different markers for each level of the hue variable
pal1 = dict(front="#ff7315", rear="#8cba51")
kws = dict(s=70, linewidth=1, edgecolor="w")
kws1 = dict(height=3, aspect=1,palette=pal1 , hue_kws=dict(marker=["^", "v"]))
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" , hue="engine-location" , **kws1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",**kws)
g.add_legend()
plt.show()
```

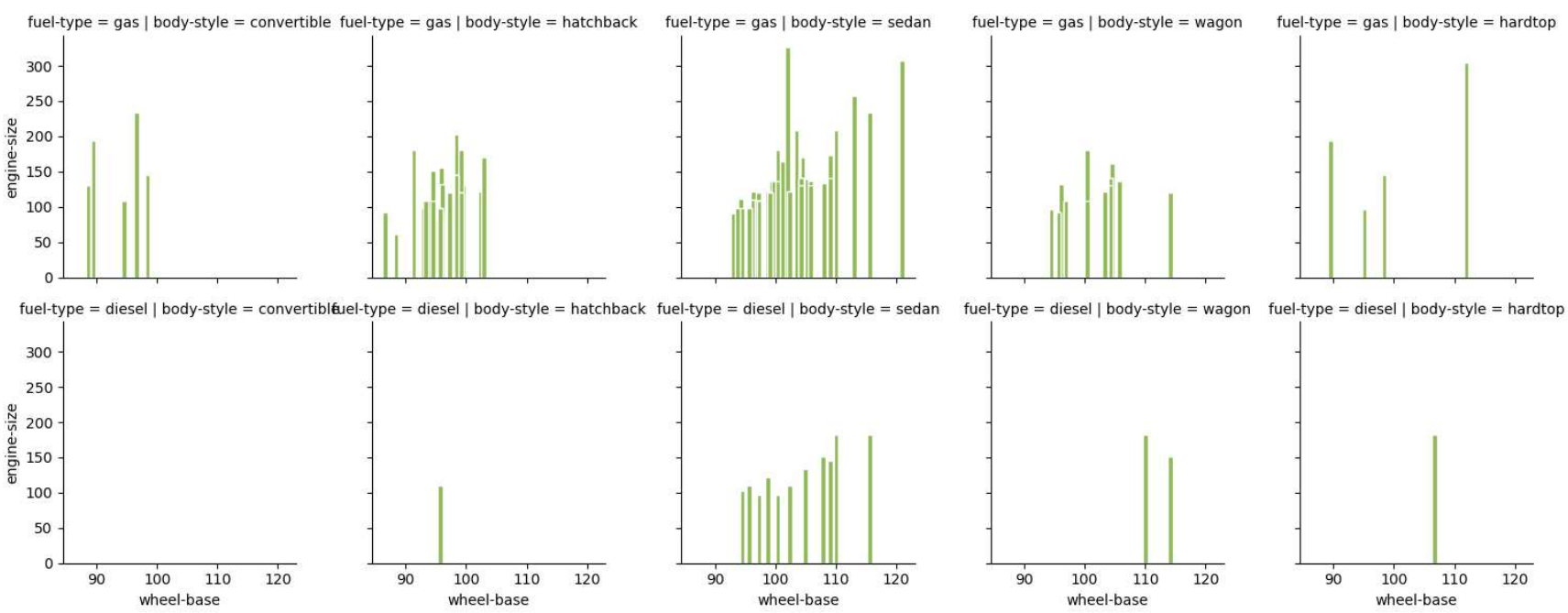


In [172…

```
g = sns.FacetGrid(auto, col="body-style" , row = "fuel-type" ,height=3, aspect=1)
g = g.map(plt.bar, "wheel-base" , "engine-size",edgecolor="w" , color = '#8cba51')
g.add_legend()
```

Out[172]:

```
<seaborn.axisgrid.FacetGrid at 0x7bf6298f3da0>
```



21.Joint Plot

Joint plot is used to draw a plot of two variables with bivariate graph and univariate graphs in the margin.

In [173…

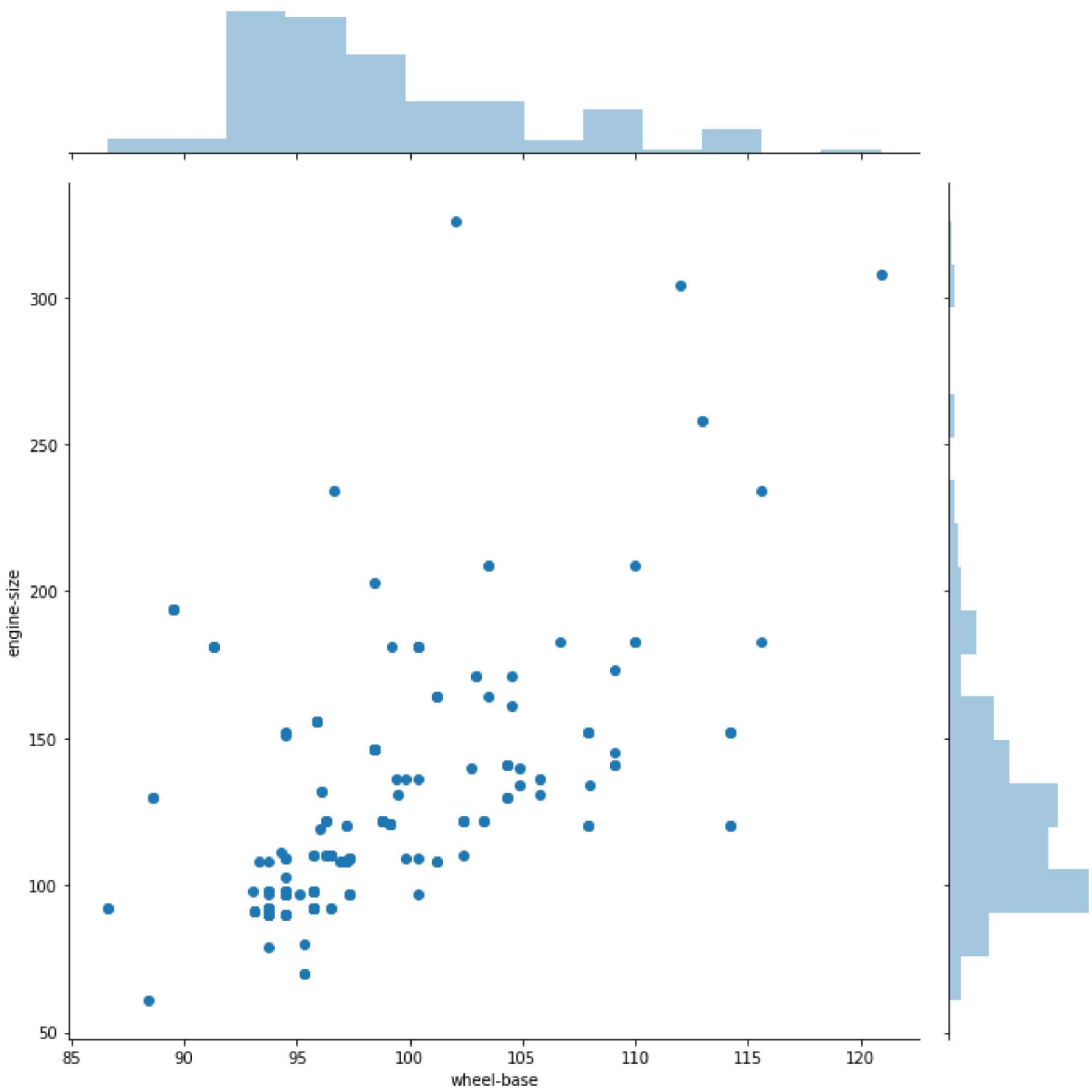
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

In [174…

```
# scatterplot with marginal histograms
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10)
```

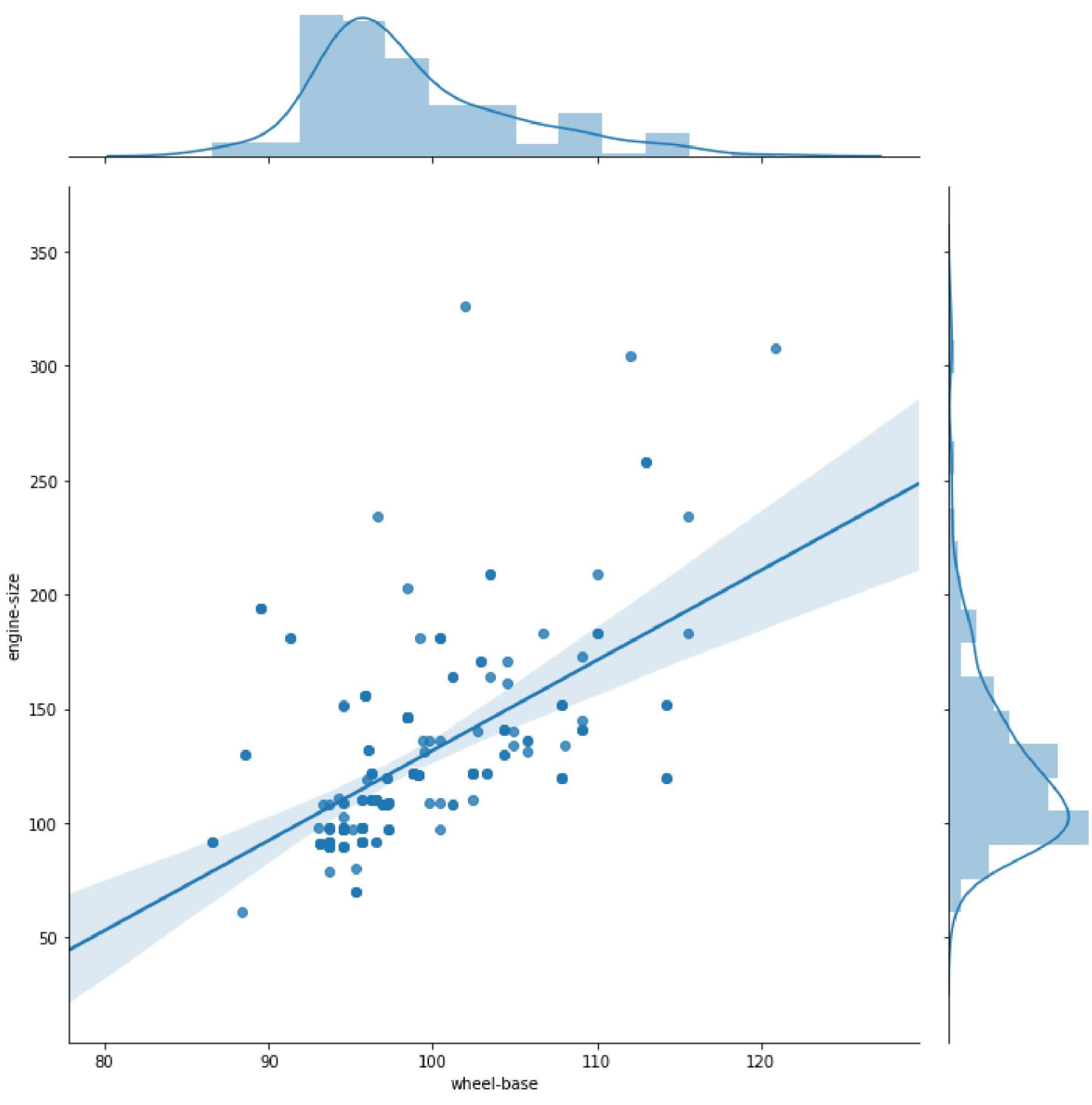
Out[174]:

```
<seaborn.axisgrid.JointGrid at 0x7bf62bb4b5f8>
```



In [175…
```
# Regplot with marginal distplot
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , kind="reg")
```

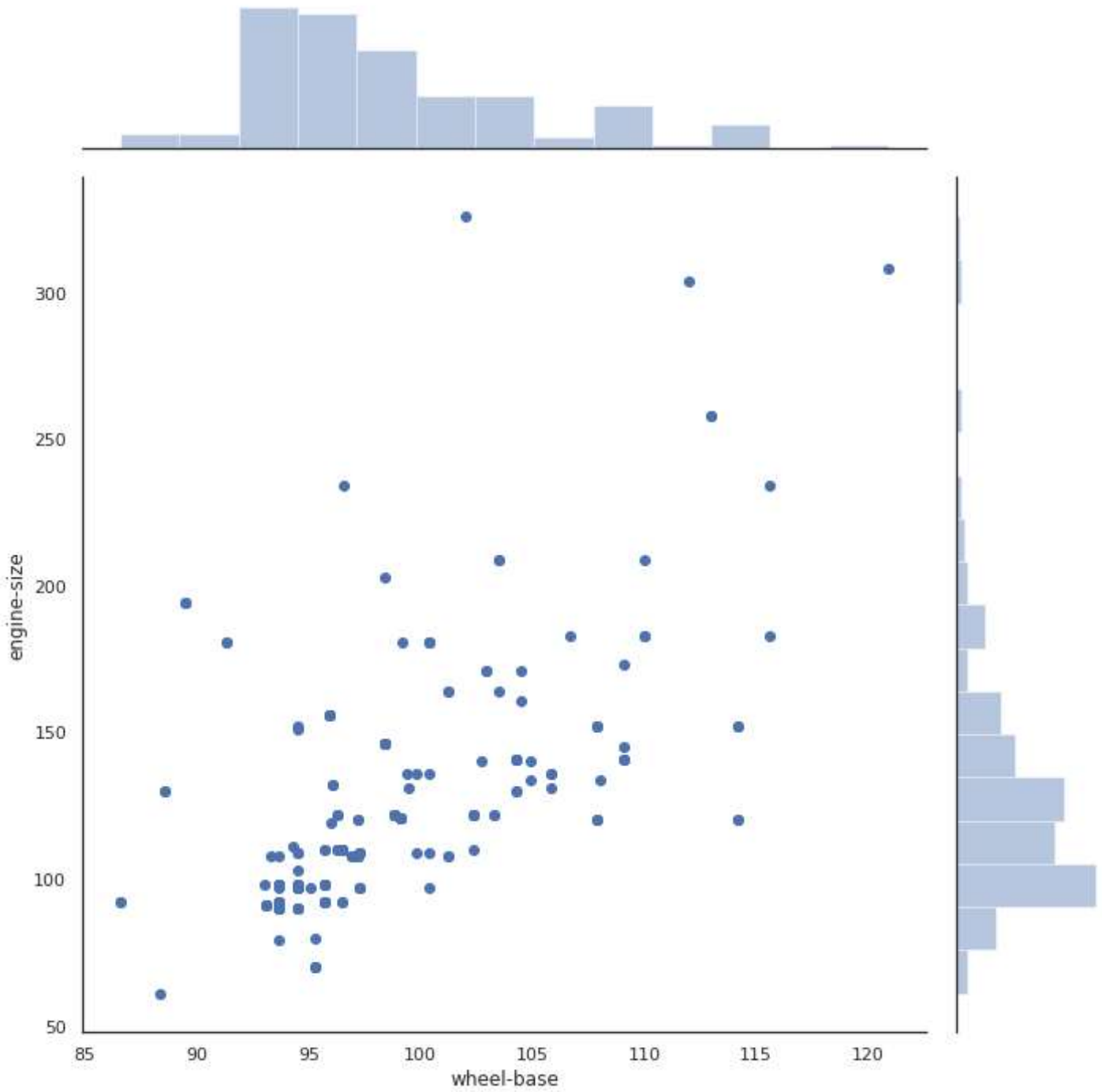
Out[175]:
```
<seaborn.axisgrid.JointGrid at 0x7bf62ab43908>
```



In [176…
```
sns.set(style="white", color_codes=True)
```

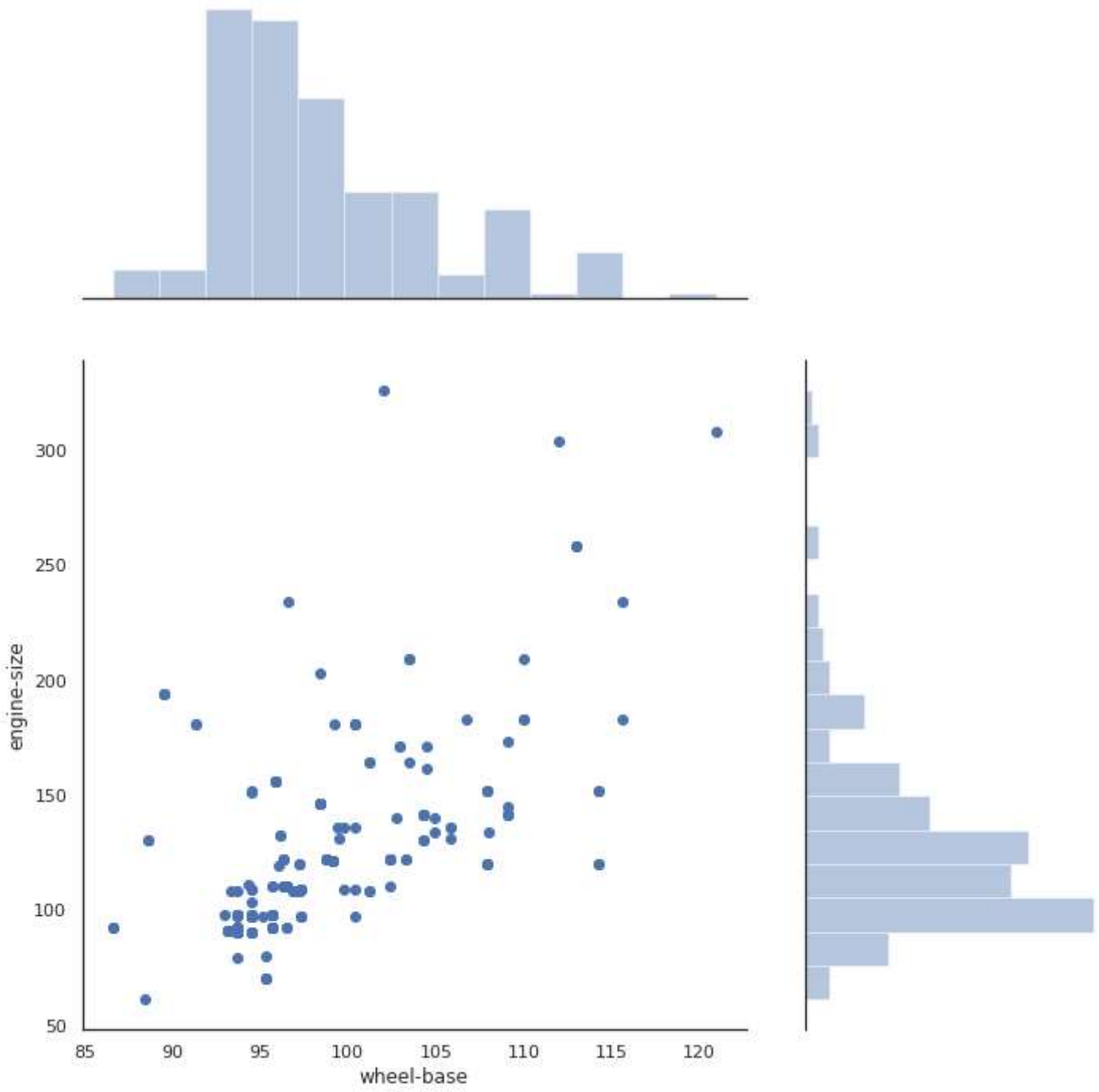
In [177…
```
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10)
```

Out[177]:
```
<seaborn.axisgrid.JointGrid at 0x7bf6359239e8>
```



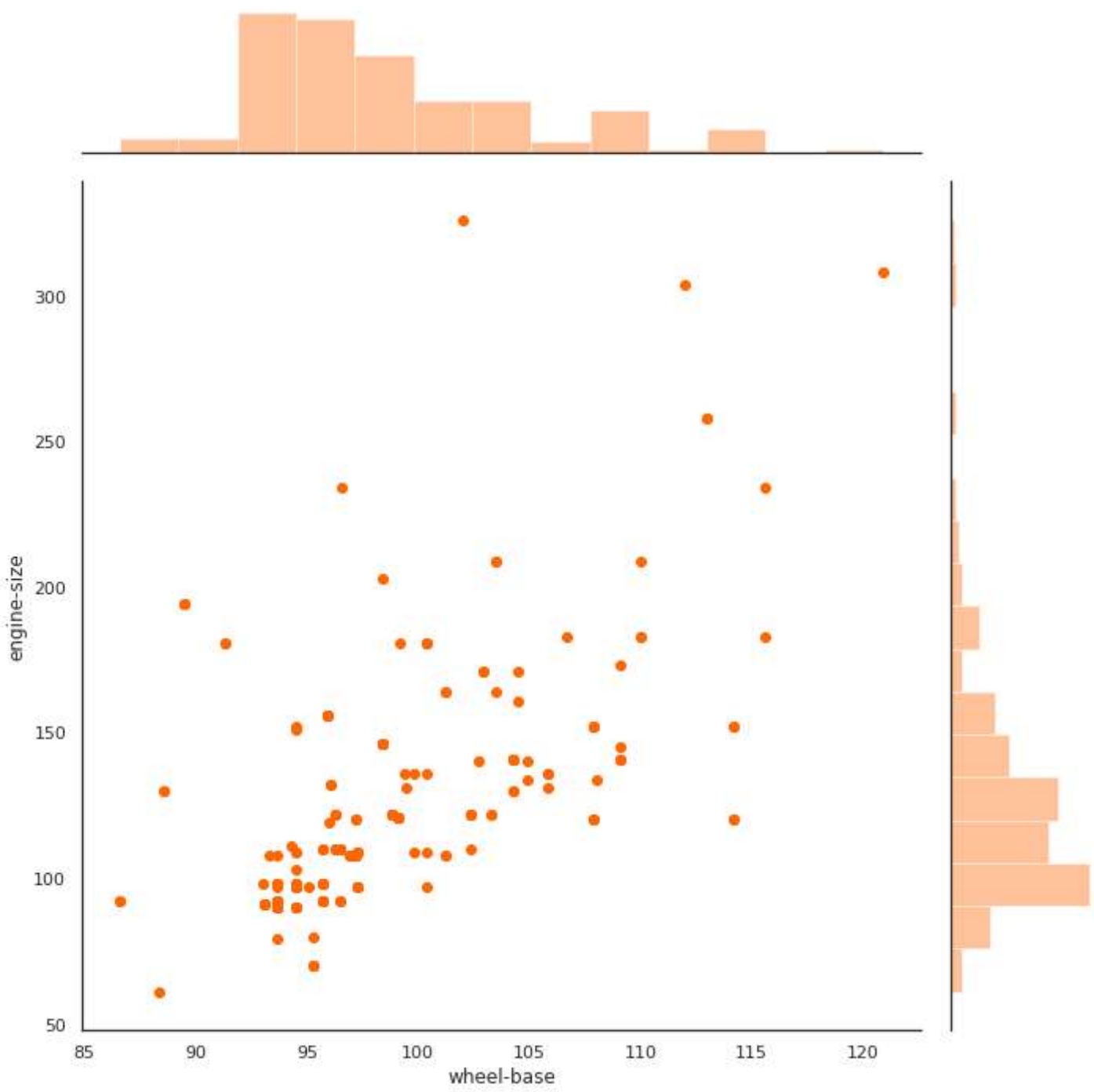
In [179…
```
# "ratio" adjusts the relative size of the marginal plots
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10, ratio=2)
```

Out[179]:
```
<seaborn.axisgrid.JointGrid at 0x7bf630de5048>
```



In [180…

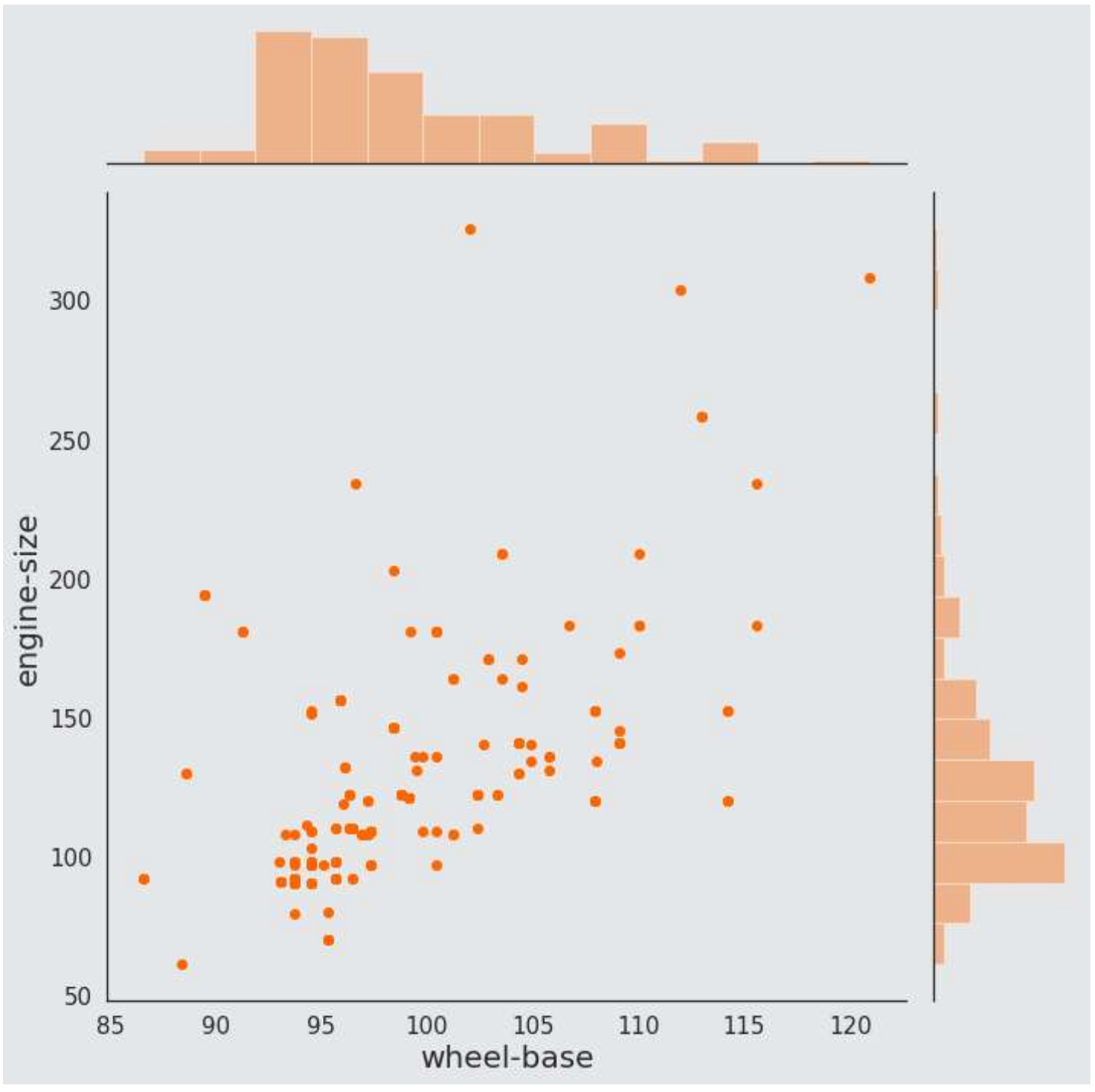
```
# Change the color of the joint plot
g = sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , color = '#FF6600')
```



In [181…

```
plt.rcParams['figure.facecolor'] = "#E5E7E9"
plt.rcParams['axes.facecolor'] = "#E5E7E9"
plt.rcParams[ 'axes.labelsize'] = 20
```

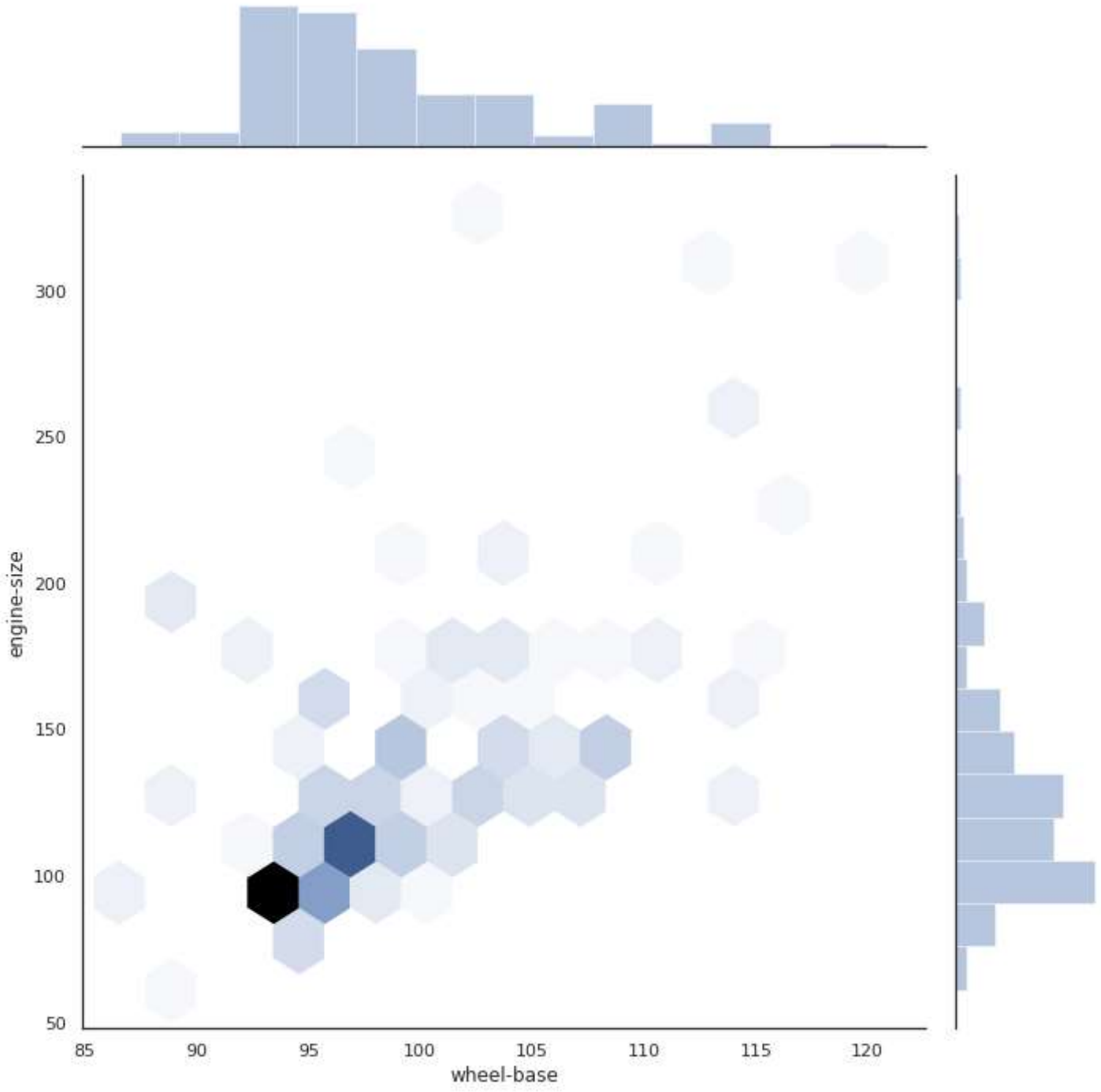
```
plt.rcParams['xtick.labelsize'] = 15
plt.rcParams['ytick.labelsize'] = 15
g = sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , color = '#FF6600')
```



In [182…
```
# Recover default matplotlib settings
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set(style="white", color_codes=True)
```

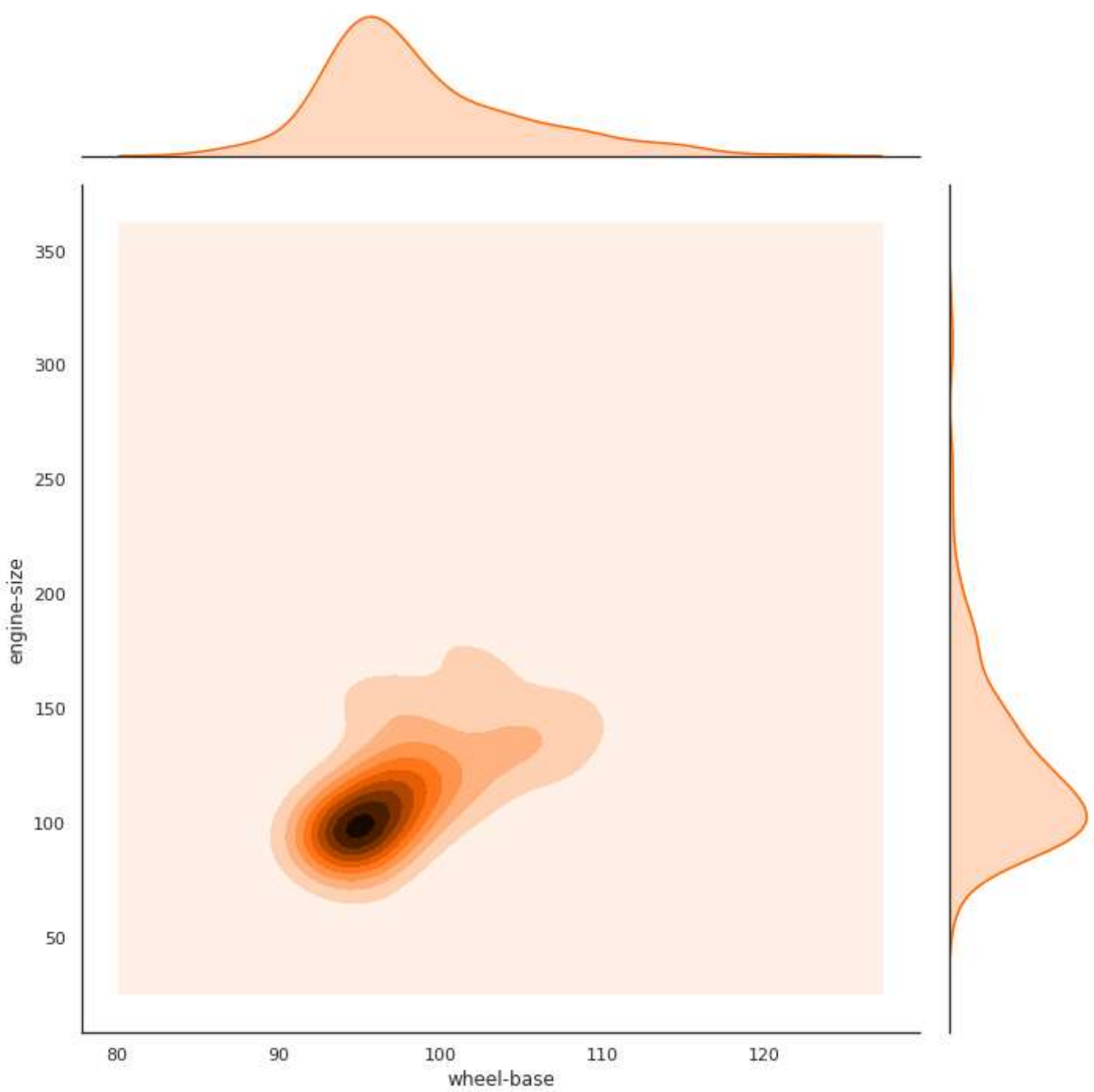
In [183…
```
# Replace the scatterplot with a joint histogram using hexagonal bins
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , kind="hex")
```

Out[183]:
```
<seaborn.axisgrid.JointGrid at 0x7bf62bb78160>
```



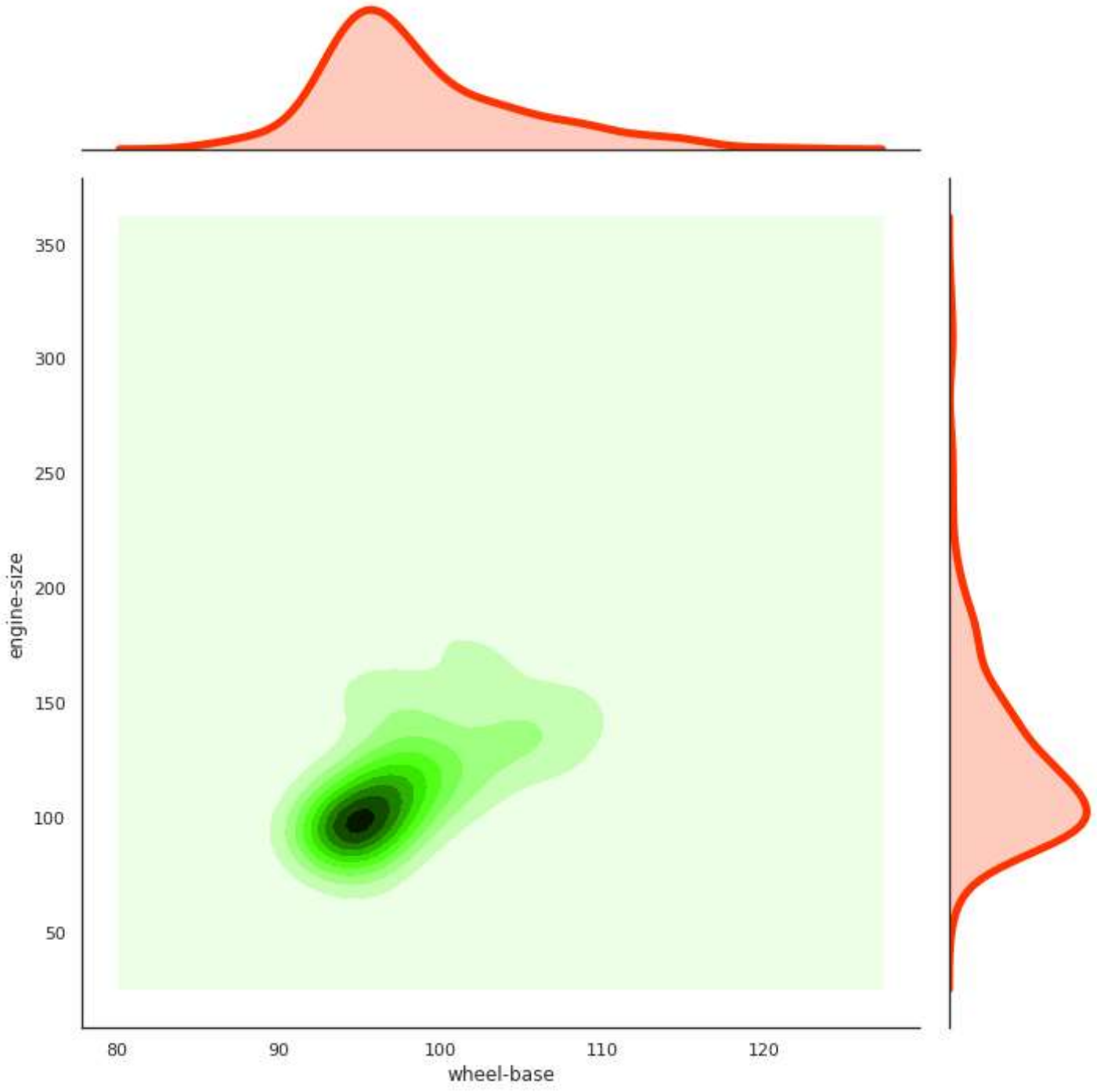
In [185…
```
""" Replace the scatterplot & joint histogram with kde plot
in the margins and the interior into a shaded countour plot """
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , kind="kde" , color="#FF6600")
```

Out[185]: <seaborn.axisgrid.JointGrid at 0x7bf62b6492e8>

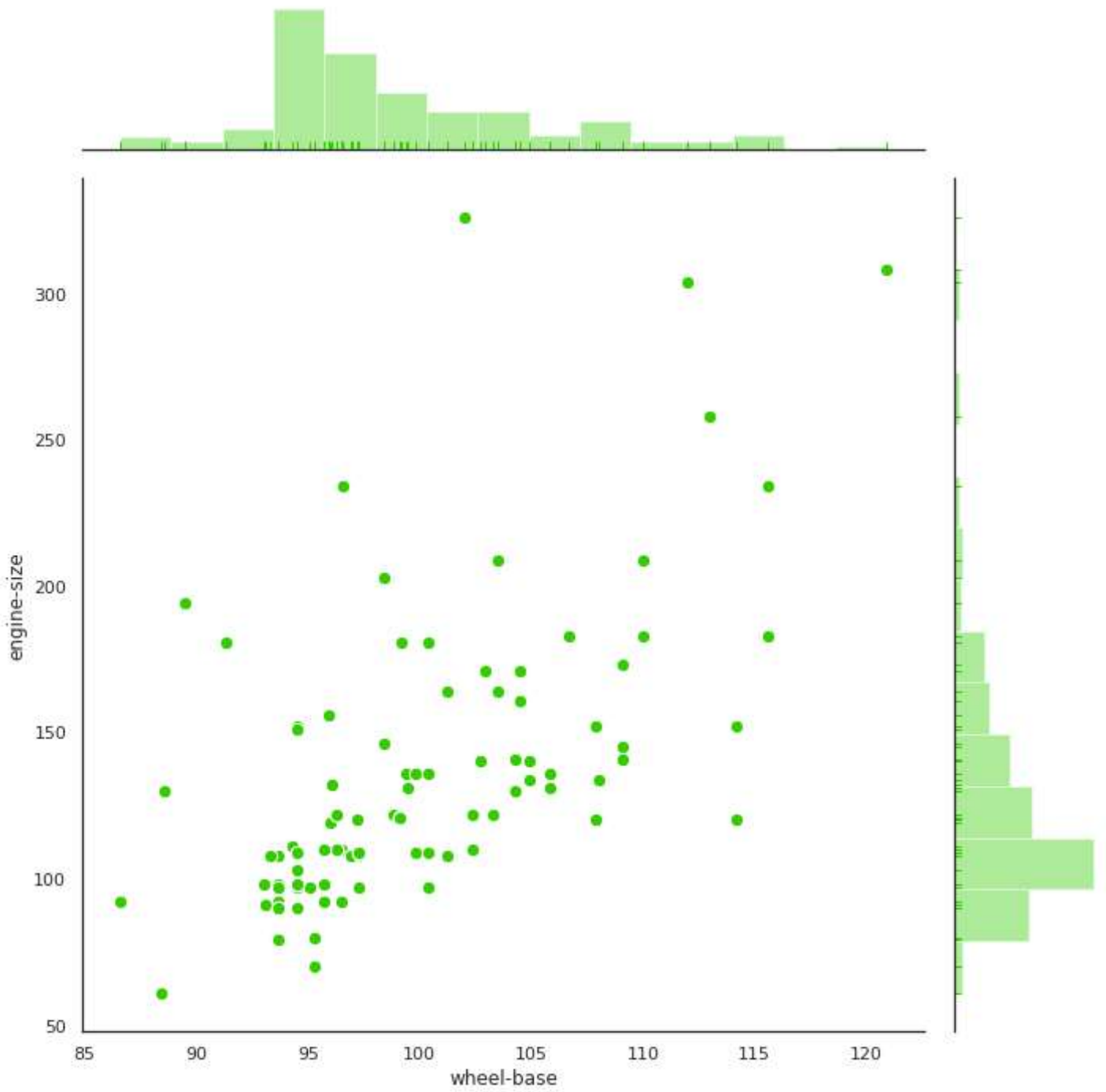


In [187…
```
# Change formatting of marginal graphs
sns.jointplot(x="wheel-base", y="engine-size", data=auto , height = 10 , kind="kde" ,color="#33CC00" , marginal_kws={'l
```

Out[187]: <seaborn.axisgrid.JointGrid at 0x7bf629d71eb8>



In [190…
```
sns.jointplot("wheel-base", "engine-size", data=auto, s=70, edgecolor="w", linewidth=1, height =10,color ="#33CC00",mar
plt.show()
```

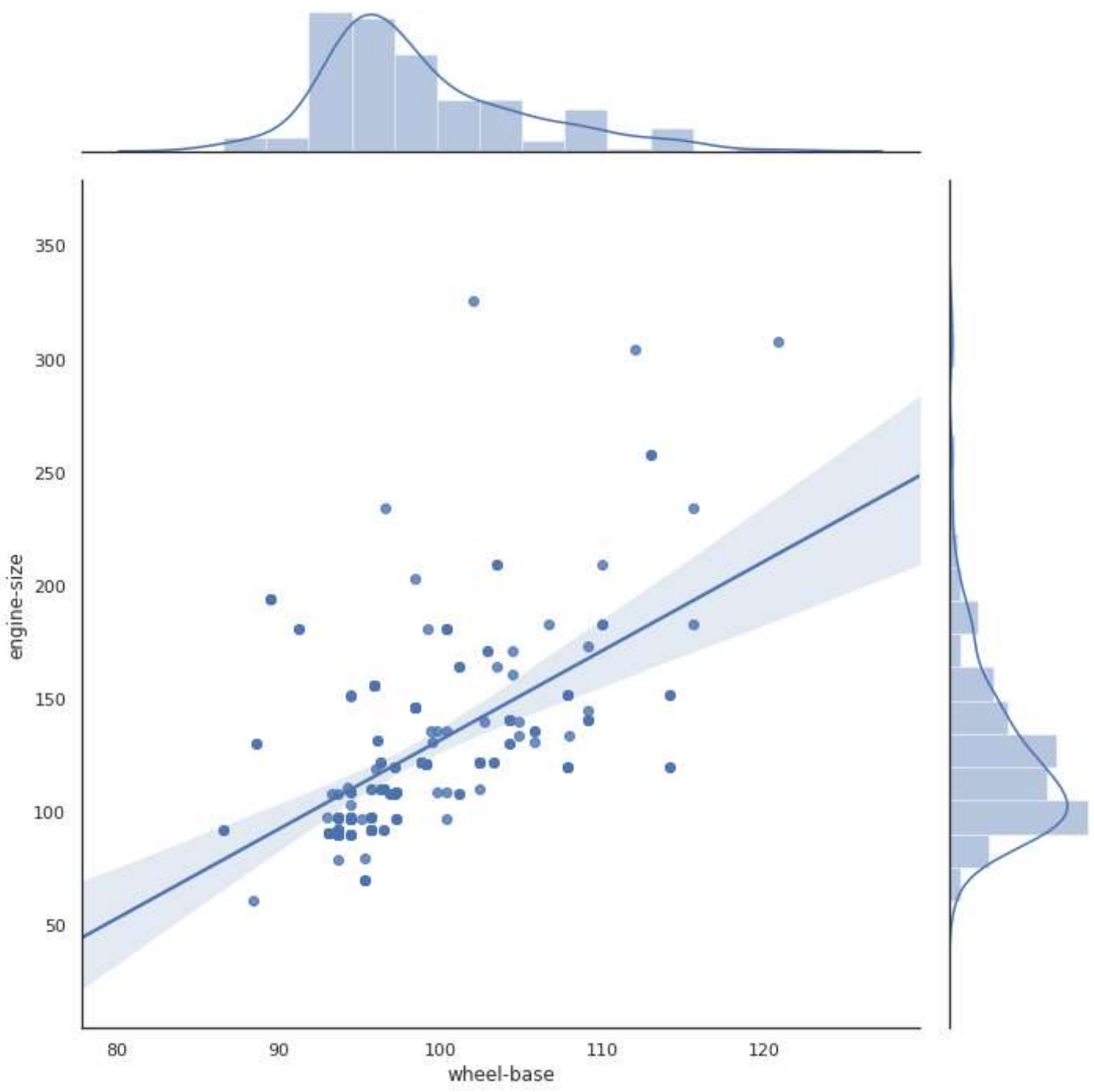


22. Joint Grid

Grid for drawing a bivariate plot with marginal univariate plots.

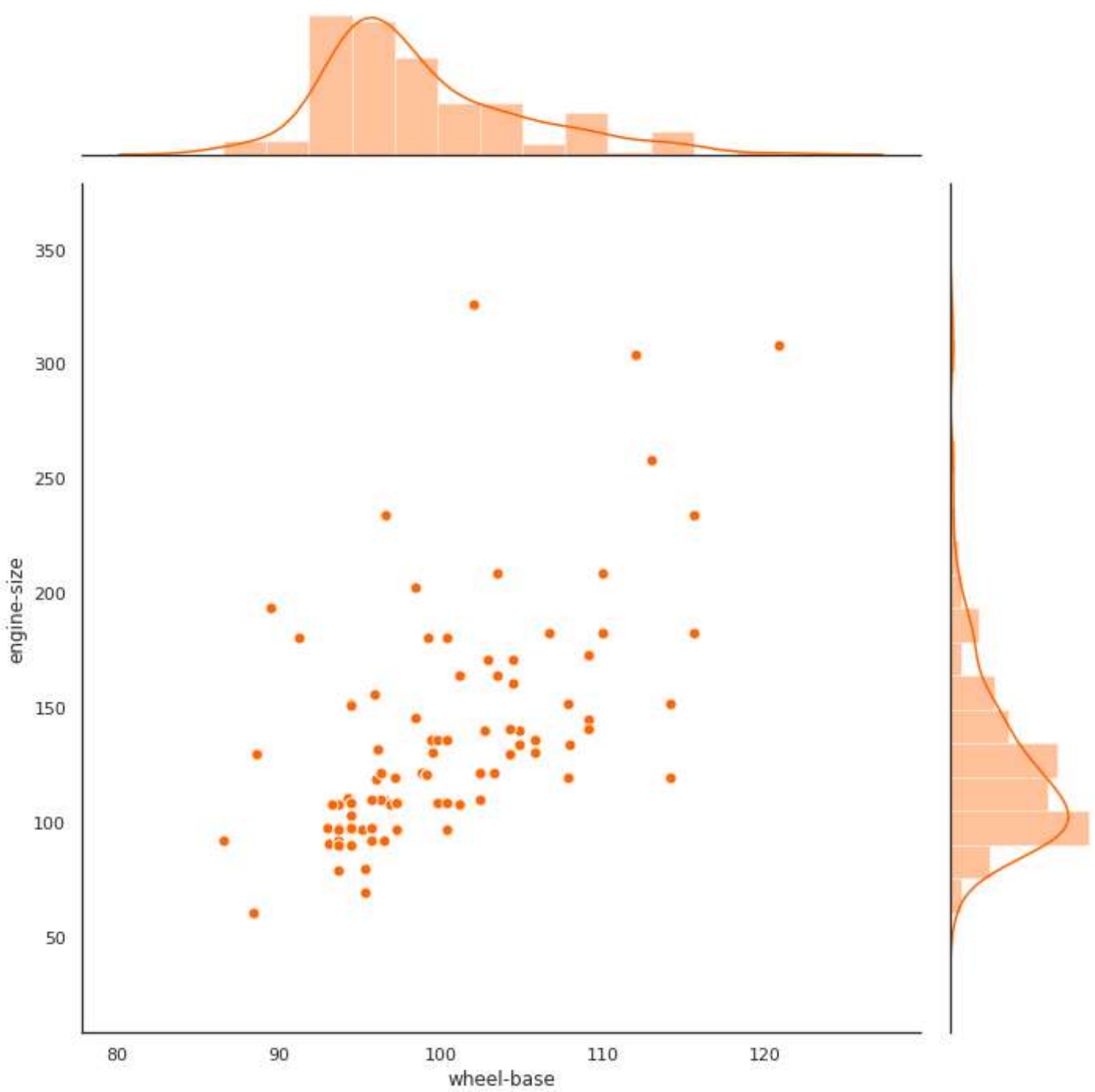
In [191…

```
#Add plots using default parameters
g = sns.JointGrid(x="wheel-base", y="engine-size", data=auto, height = 10)
g = g.plot(sns.regplot, sns.distplot)
```



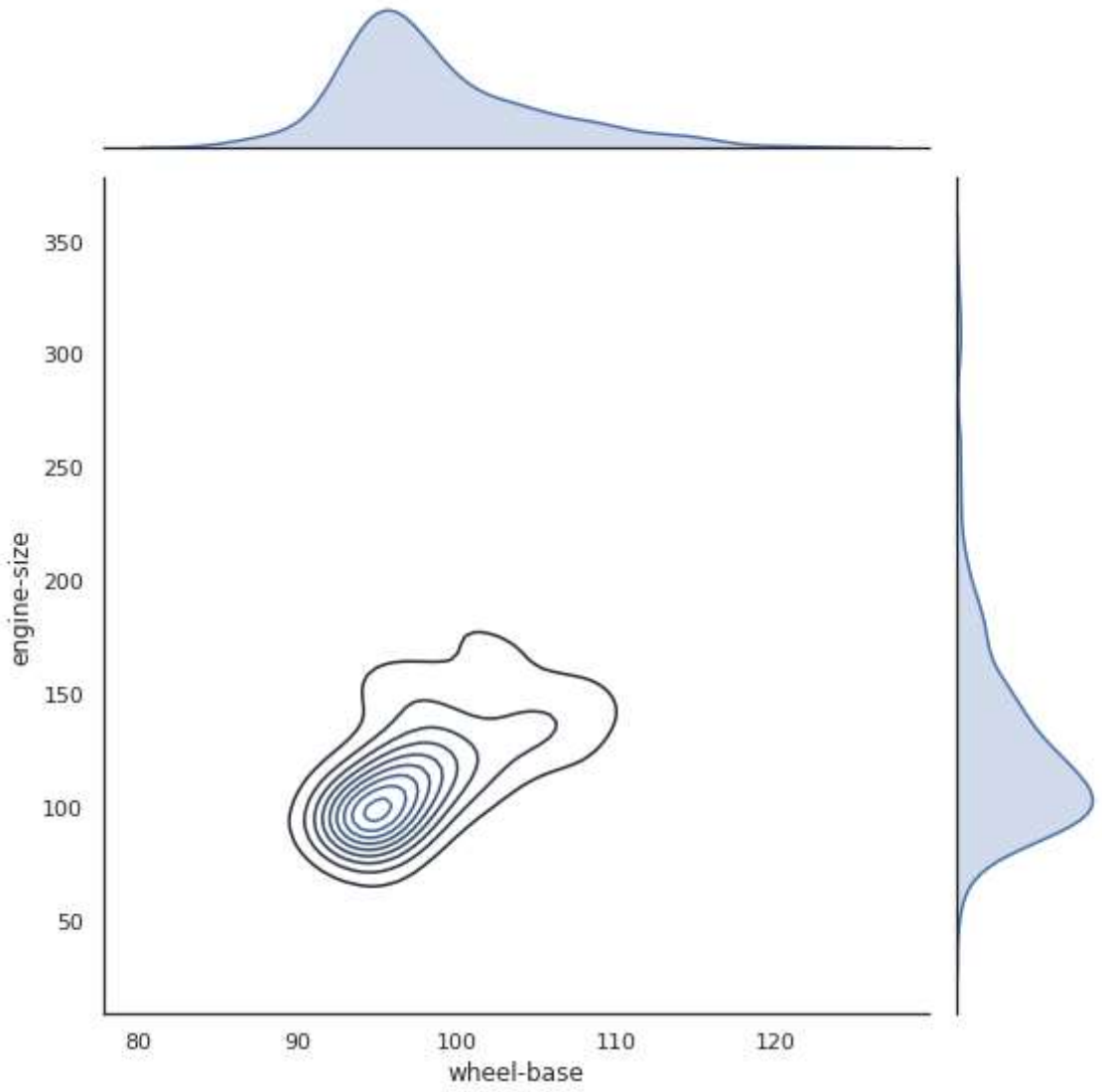
In [192…

```
# Changing color of plots
g = sns.JointGrid(x="wheel-base", y="engine-size", data=auto, height = 10)
g = g.plot_joint(sns.scatterplot, color="#FF6600",s=50)
g = g.plot_marginals(sns.distplot, color="#FF6600")
```



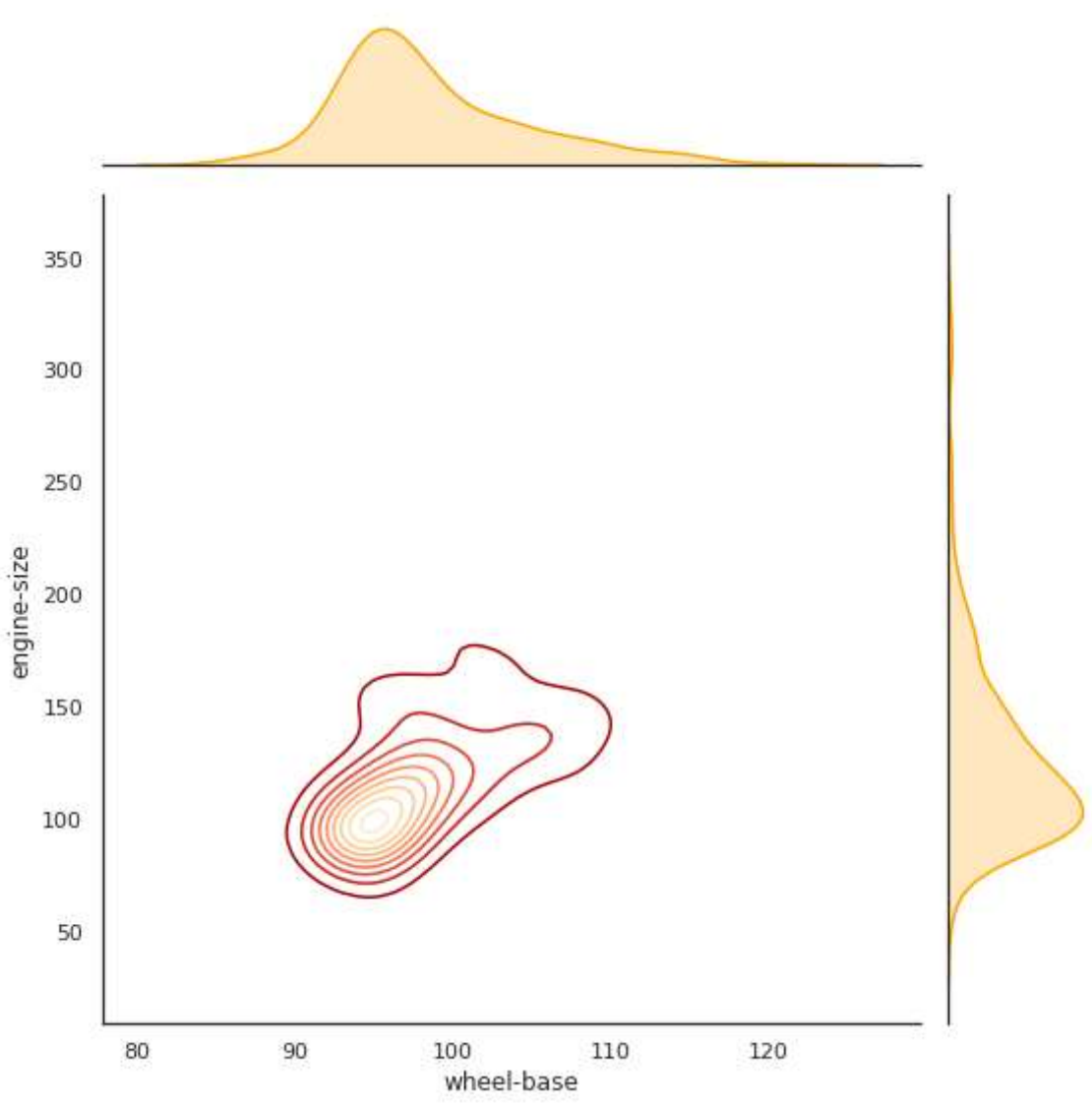
In [194…

```
# kde plot in the margins and the interior into a shaded countour plot
g = sns.JointGrid(x="wheel-base", y="engine-size", data=auto, height = 8)
g = g.plot_joint(sns.kdeplot)
g = g.plot_marginals(sns.kdeplot, shade=True)
```



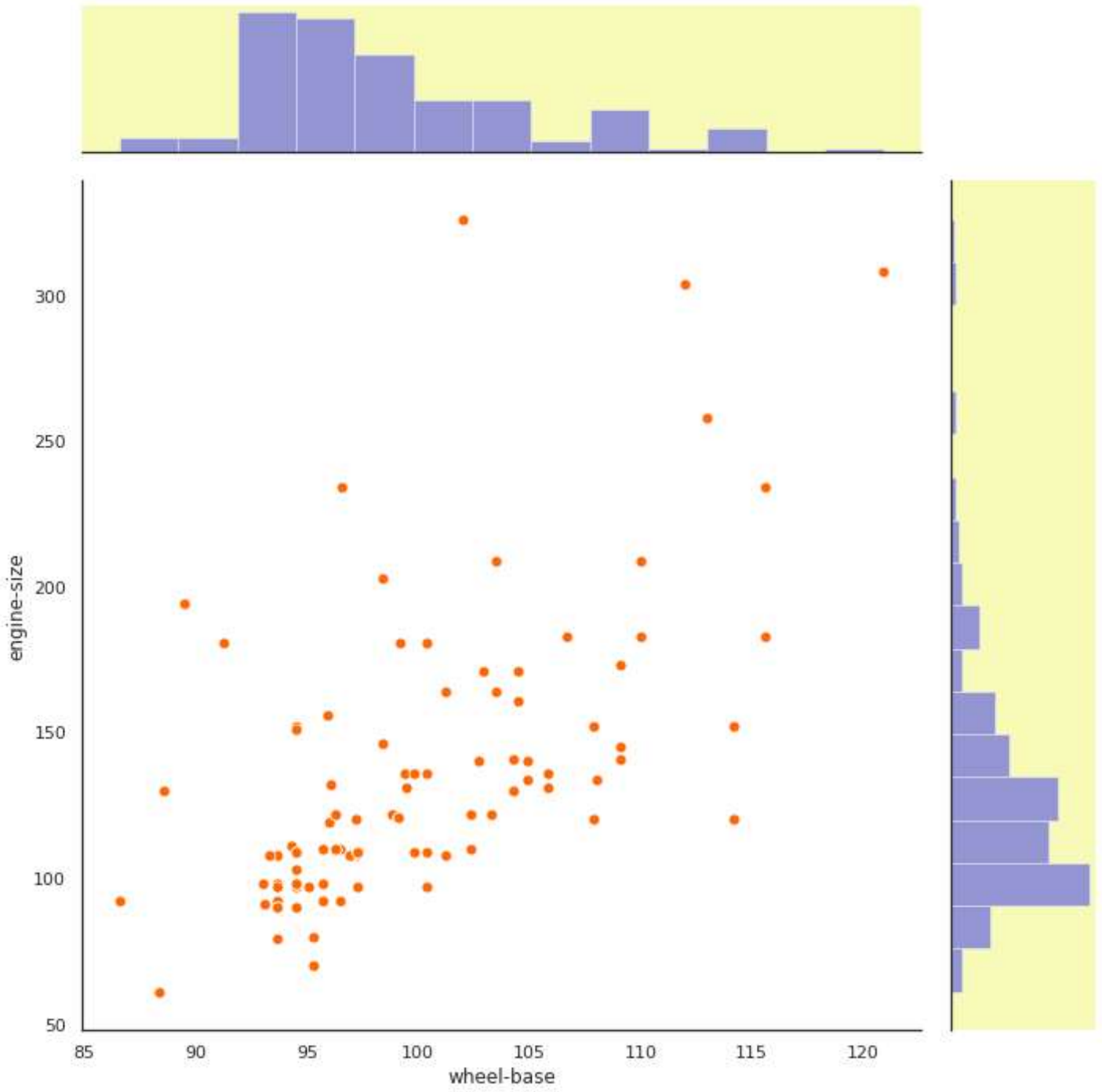
In [196…

```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=auto, height = 8)
g = g.plot_joint(sns.kdeplot , cmap="OrRd_r")
g = g.plot_marginals(sns.kdeplot, shade=True , color = 'orange')
```



In [198…

```
# Changing background of marginal graphs
g = sns.JointGrid(x="wheel-base", y="engine-size", data=auto, height = 10)
g = g.plot_joint(sns.scatterplot, color="#FF6600",s=50)
g= g.plot_marginals(sns.distplot,kde=False , color = 'Blue')
g.ax_marg_x.set_facecolor('#f8fab8')
g.ax_marg_y.set_facecolor('#f8fab8')
```

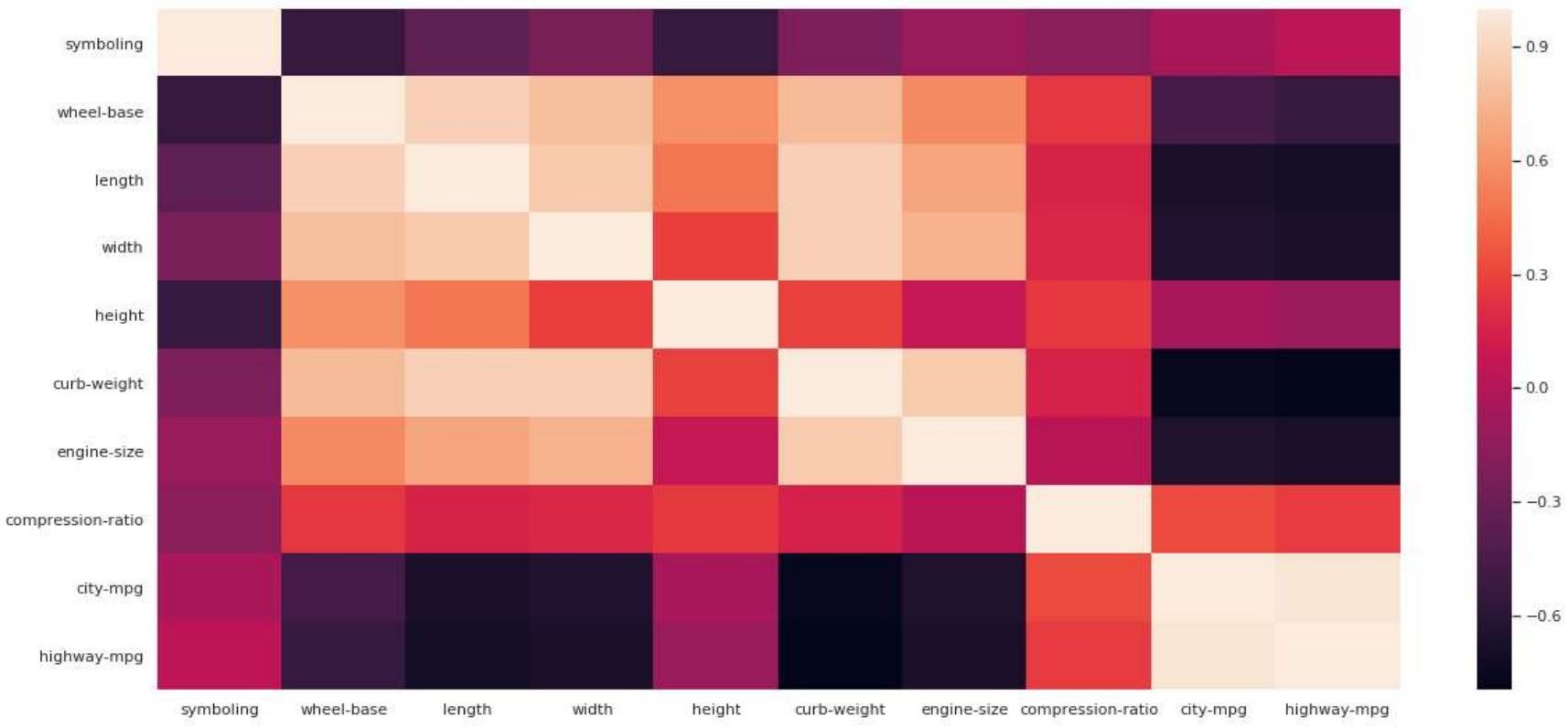


23.Heat Map

A heat map is a data visualization technique that shows magnitude of a phenomenon as color in two dimensions. The variation in color may be by hue or intensity, giving obvious visual cues to the reader about how the phenomenon is clustered or varies over space.

In [199…

```
corr = auto.corr()
plt.figure(figsize=(20,9))
ax = sns.heatmap(corr)
plt.yticks(rotation=0)
plt.show()
```

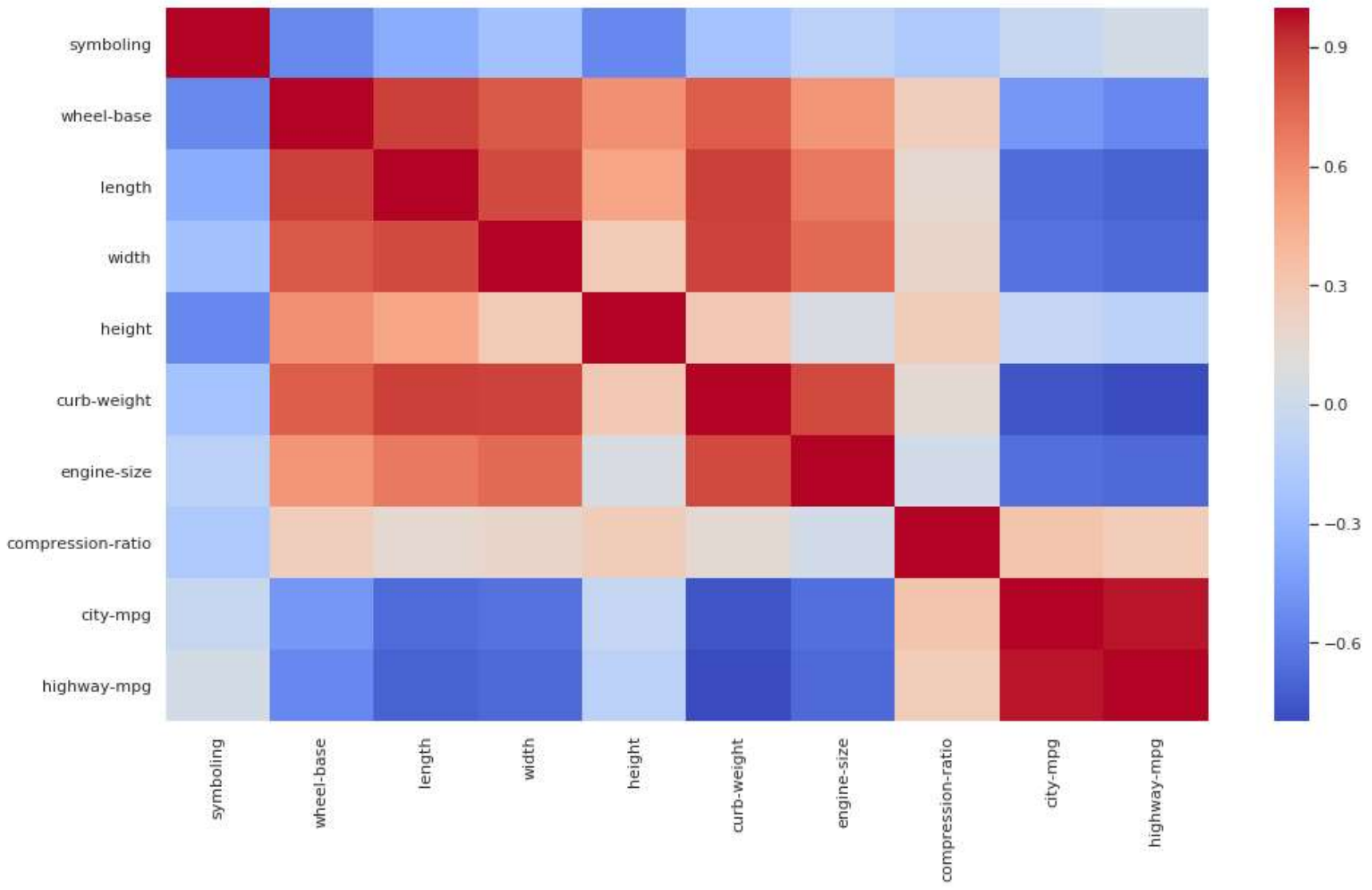


In [202…

```
# Changing cmap
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="YlGnBu")
plt.yticks(rotation=0)
plt.show()
```

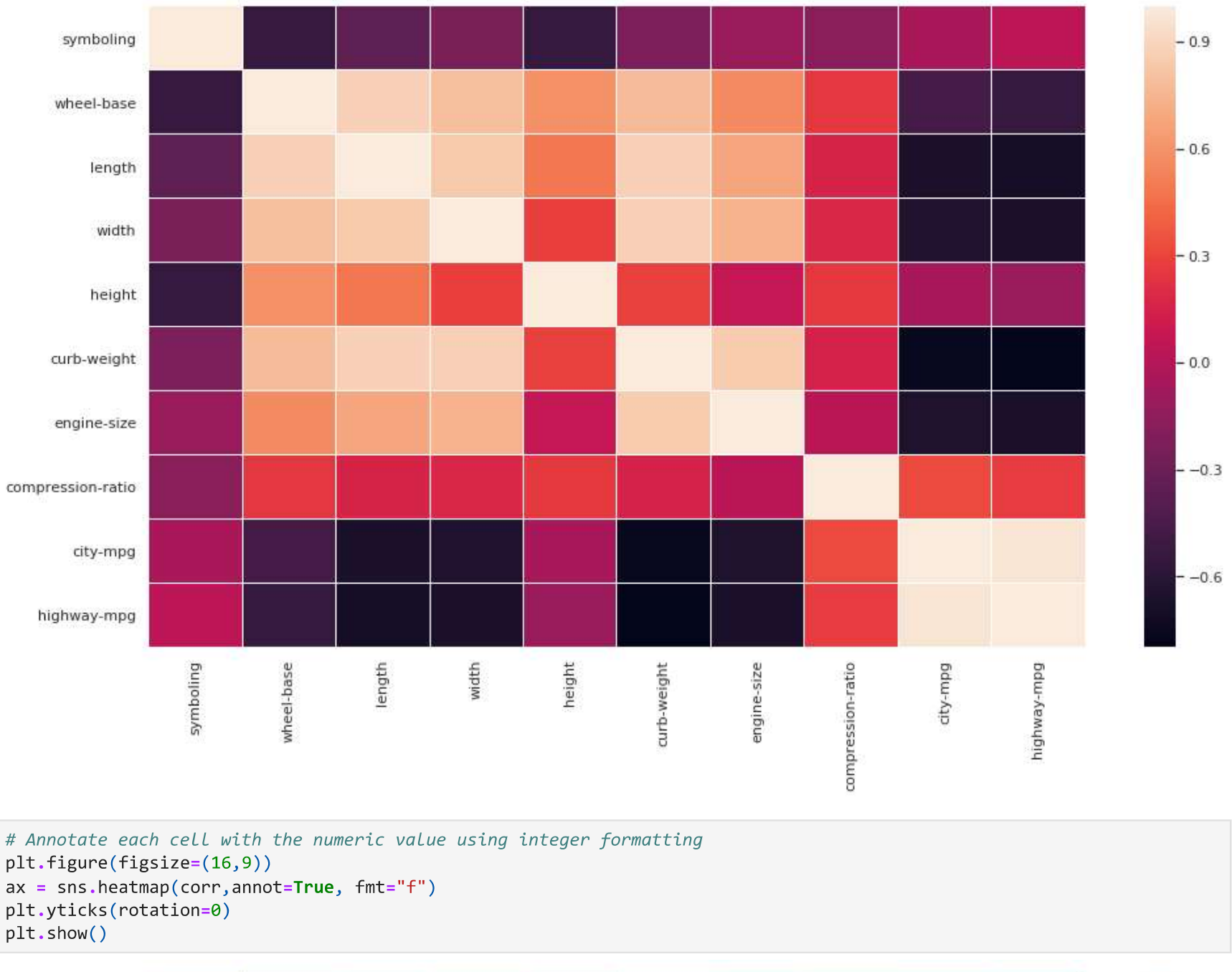
In [203…

```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="coolwarm")
plt.yticks(rotation=0)
plt.show()
```



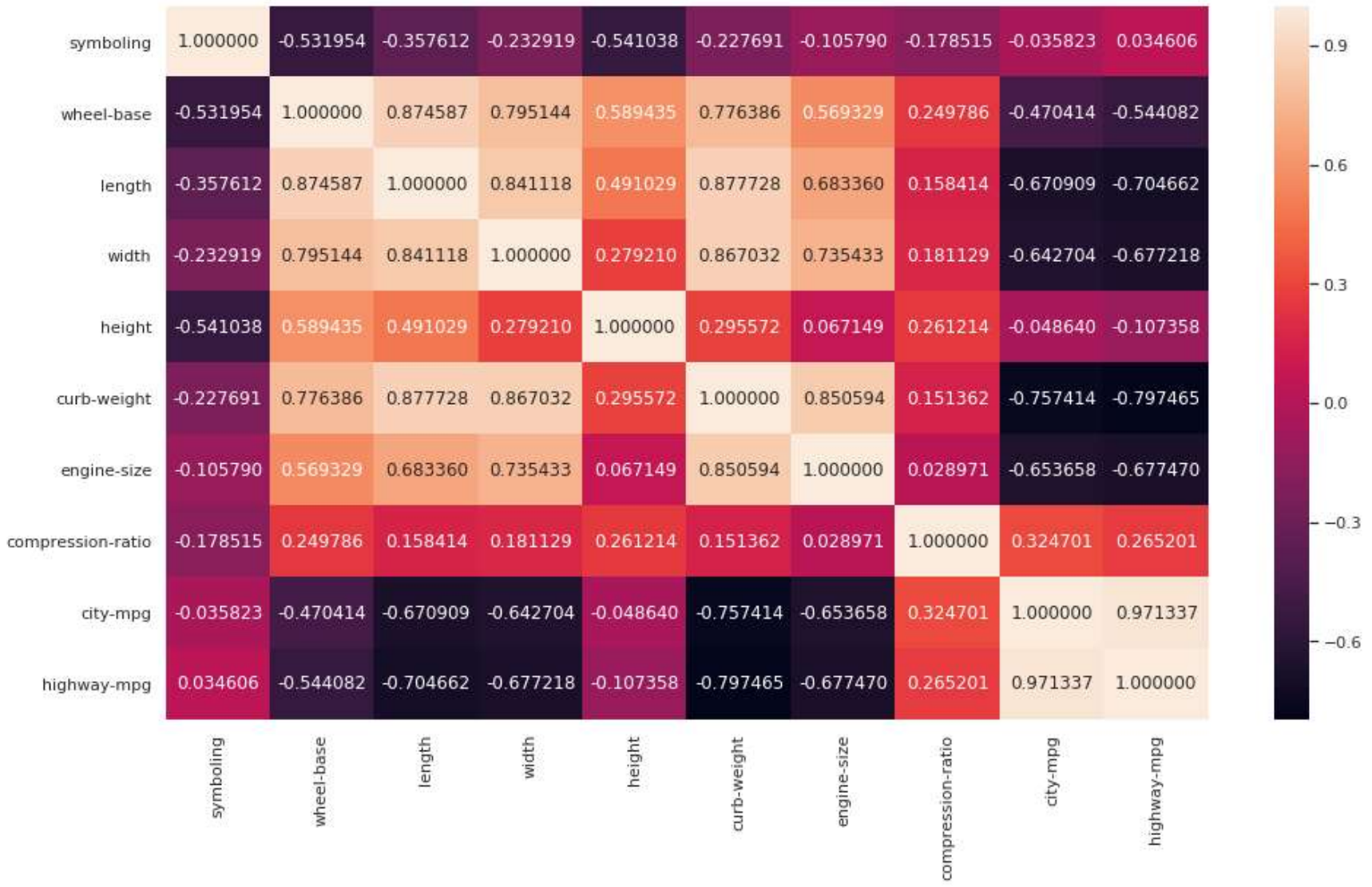
In [204…

```
# Add lines between each cell
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,linewidths=.1)
plt.yticks(rotation=0)
plt.show()
```



In [205…

```
# Annotate each cell with the numeric value using integer formatting
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,annot=True, fmt="f")
plt.yticks(rotation=0)
plt.show()
```

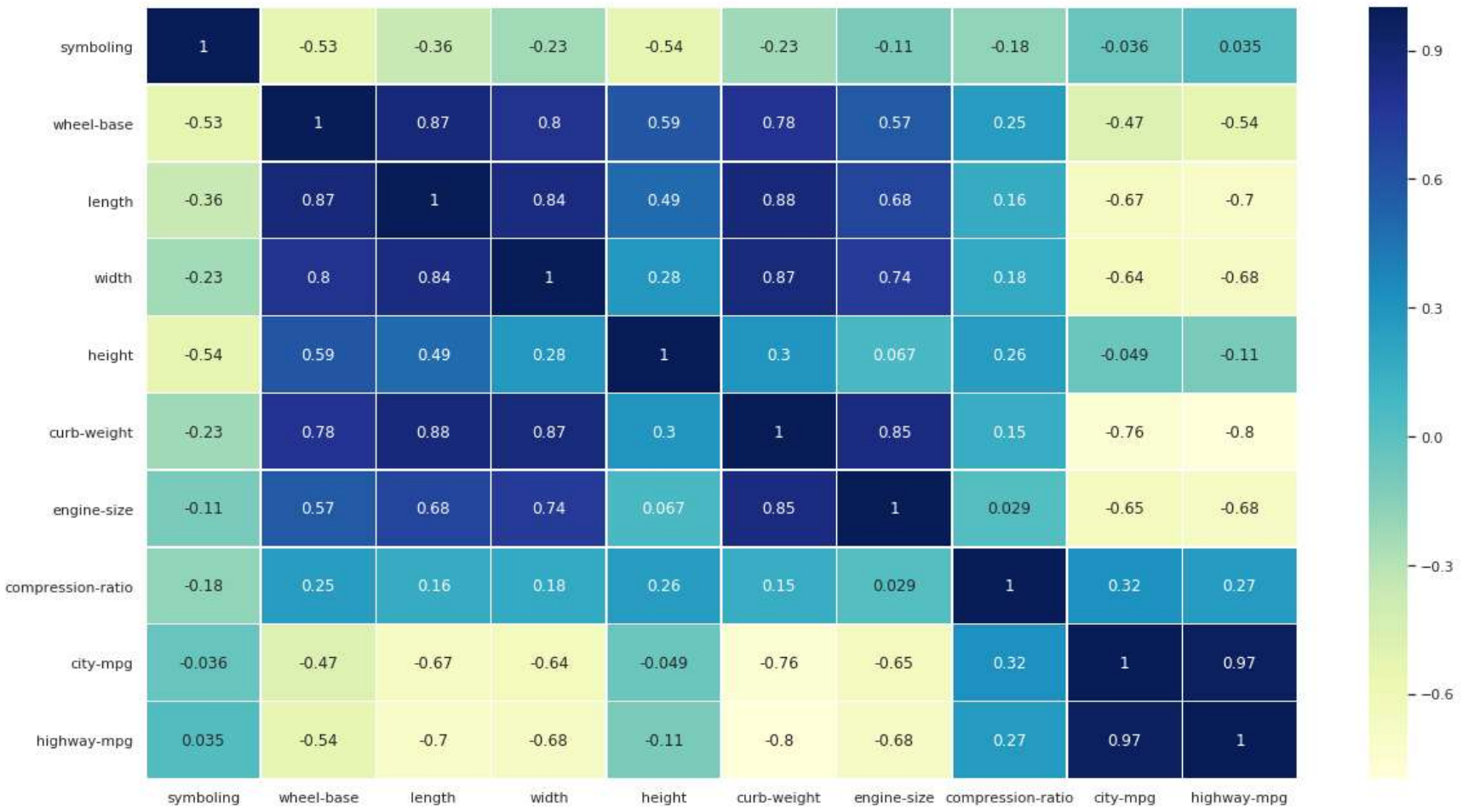


In [206…

```
# Annotate each cell with the numeric value using decimal formatting
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="YlGnBu", annot=True ,fmt=".2f")
plt.yticks(rotation=0)
plt.show()
```

In [207…

```
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,cmap="YlGnBu", linewidths=.5, annot=True )
plt.yticks(rotation=0)
plt.show()
```



In [208…

```
# Turn off the colorbar -> cbar=False
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,vmin=0, vmax=8000,cmap="YlGnBu", linewidths=.5,
annot=True ,annot_kws={'size':14} ,fmt=".1f" , cbar=False)
plt.yticks(rotation=0)
plt.show()
```

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
symboling	1.0	-0.5	-0.4	-0.2	-0.5	-0.2	-0.1	-0.2	-0.0	0.0
wheel-base	-0.5	1.0	0.9	0.8	0.6	0.8	0.6	0.2	-0.5	-0.5
length	-0.4	0.9	1.0	0.8	0.5	0.9	0.7	0.2	-0.7	-0.7
width	-0.2	0.8	0.8	1.0	0.3	0.9	0.7	0.2	-0.6	-0.7
height	-0.5	0.6	0.5	0.3	1.0	0.3	0.1	0.3	-0.0	-0.1
curb-weight	-0.2	0.8	0.9	0.9	0.3	1.0	0.9	0.2	-0.8	-0.8
engine-size	-0.1	0.6	0.7	0.7	0.1	0.9	1.0	0.0	-0.7	-0.7
compression-ratio	-0.2	0.2	0.2	0.2	0.3	0.2	0.0	1.0	0.3	0.3
city-mpg	-0.0	-0.5	-0.7	-0.6	-0.0	-0.8	-0.7	0.3	1.0	1.0
highway-mpg	0.0	-0.5	-0.7	-0.7	-0.1	-0.8	-0.7	0.3	1.0	1.0

In [209…

```
# Force the aspect ratio of the blocks to be equal using "square" parameter
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,cmap="YlGnBu", linewidths=.5,
annot=True ,annot_kws={'size':14} ,fmt=".1f" , cbar=False ,square = True)
plt.yticks(rotation=0)
plt.show()
```