uxf4377wh

March 11, 2023

1 Automobile

Here we will be learning visualisation using graph through **seaborn** library. ## 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. ## Seaborn vs Matplotlib - 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

1.1 Import libraries

[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
```

[2]:
```
db = pd.read_csv('C:/Users/suman/Desktop/DS learn/Project/Study/Automobile/
↪Automobile_data.csv')
```

[3]:
```
db.head()
```

[3]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	\
0	3	?	alfa-romero	gas	std	two	
1	3	?	alfa-romero	gas	std	two	
2	1	?	alfa-romero	gas	std	two	
3	2	164	audi	gas	std	four	
4	2	164	audi	gas	std	four	

	body-style	drive-wheels	engine-location	wheel-base	…	engine-size	\
0	convertible	rwd	front	88.6	…	130	

```
1  convertible          rwd           front        88.6  …          130
2    hatchback          rwd           front        94.5  …          152
3        sedan          fwd           front        99.8  …          109
4        sedan          4wd           front        99.4  …          136

   fuel-system  bore  stroke compression-ratio horsepower  peak-rpm city-mpg  \
0         mpfi  3.47    2.68               9.0        111      5000       21
1         mpfi  3.47    2.68               9.0        111      5000       21
2         mpfi  2.68    3.47               9.0        154      5000       19
3         mpfi  3.19     3.4              10.0        102      5500       24
4         mpfi  3.19     3.4               8.0        115      5500       18

  highway-mpg  price
0          27  13495
1          27  16500
2          26  16500
3          30  13950
4          22  17450

[5 rows x 26 columns]
```

[4]:
```
db['normalized-losses'] = db['normalized-losses'].replace('?','Unknown')
```

1.2 Line chart

The code generates a NumPy array **col1** containing 1000 evenly spaced values between 0 and 10, and another NumPy array **col2** containing the sine of each value in **col1**. It then creates a pandas DataFrame with two columns, **C1** and **C2**, using the NumPy arrays as the data.

[5]:
```
col1 = np.linspace(0, 10, 1000)
col2 = np.sin(col1)
df = pd.DataFrame({"C1" : col1 , "C2" :col2})
df.head(10)
```

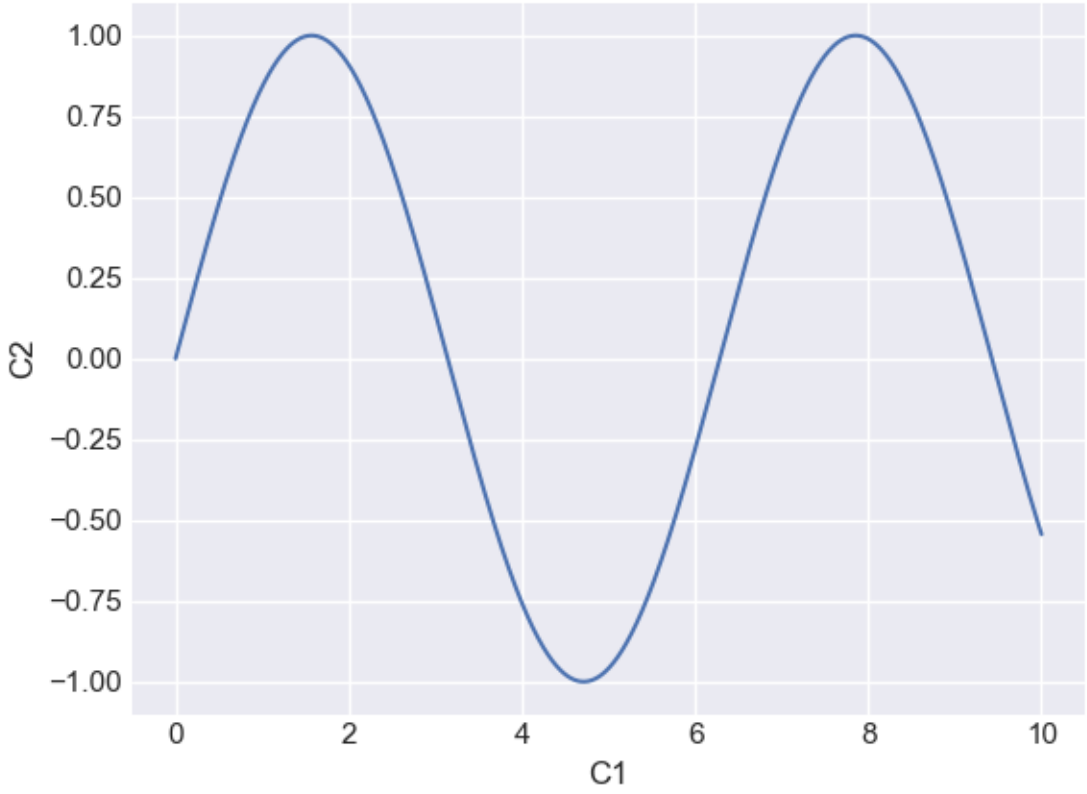
[5]:
```
         C1        C2
0  0.00000  0.000000
1  0.01001  0.010010
2  0.02002  0.020019
3  0.03003  0.030026
4  0.04004  0.040029
5  0.05005  0.050029
6  0.06006  0.060024
7  0.07007  0.070013
8  0.08008  0.079995
9  0.09009  0.089968
```

- Here **seaborn-darkgrid** is a type of plot with drid and dark back ground. It also includes a Jupyter Notebook magic command **%matplotlib inline** which enables the output of mat-

plotlib plots to be displayed inline in the notebook.

- It then uses the seaborn library to create a line plot using the **sns.lineplot()** function. The **x** and **y** arguments are set to **df.C1** and **df.C2**, respectively, indicating that the values from these columns in the DataFrame should be used as the x and y coordinates for the plot. The data argument is set to **df**, indicating that the data for the plot is contained in the DataFrame df.
- Finally, the **plt.show()** function is used to display the plot.
- This will generate a line plot of the **sine function** over the interval [0, 10] with 1000 points.

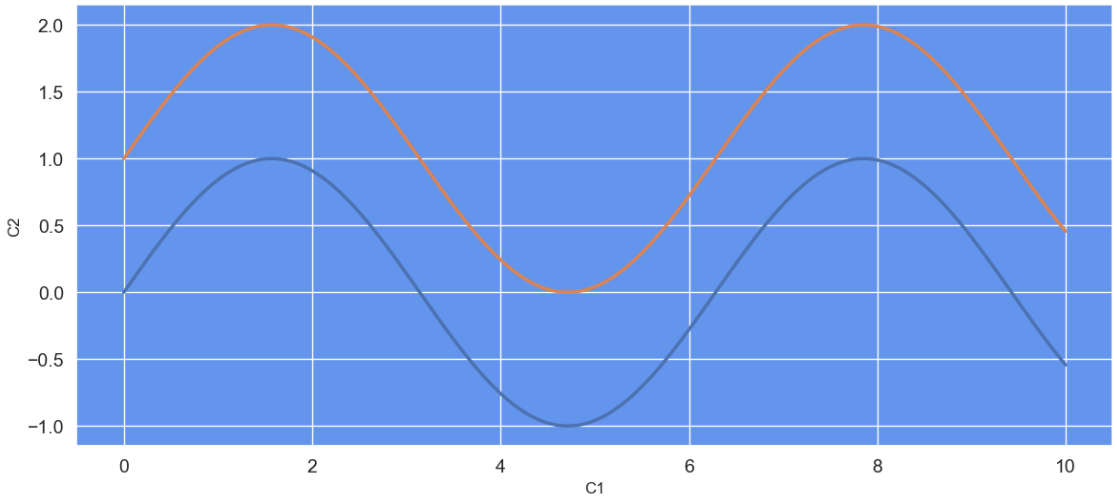
[6]:
```
plt.style.use('seaborn-darkgrid')
%matplotlib inline
sns.lineplot(x=df.C1,y=df.C2,data=df)
plt.show()
```



- He it creates a line plot using Seaborn's **lineplot()** function to display two lines on the same plot: one for the **C2** column of the **df** DataFrame, and another for the **C2** column shifted upwards by **1**.
- The **plt.figure(figsize=(14,6))** line sets the figure size to 14 inches wide and 6 inches tall. The **sns.set()** function sets several plot parameters, including the background color of the plot **(axes.facecolor)**, whether to display a grid **(axes.grid)**, and the font size of the tick labels **(xtick.labelsize and ytick.labelsize)**.
- **Result:** a plot with two lines: one in blue for **C2**, and one in orange for **C2+1**, with the

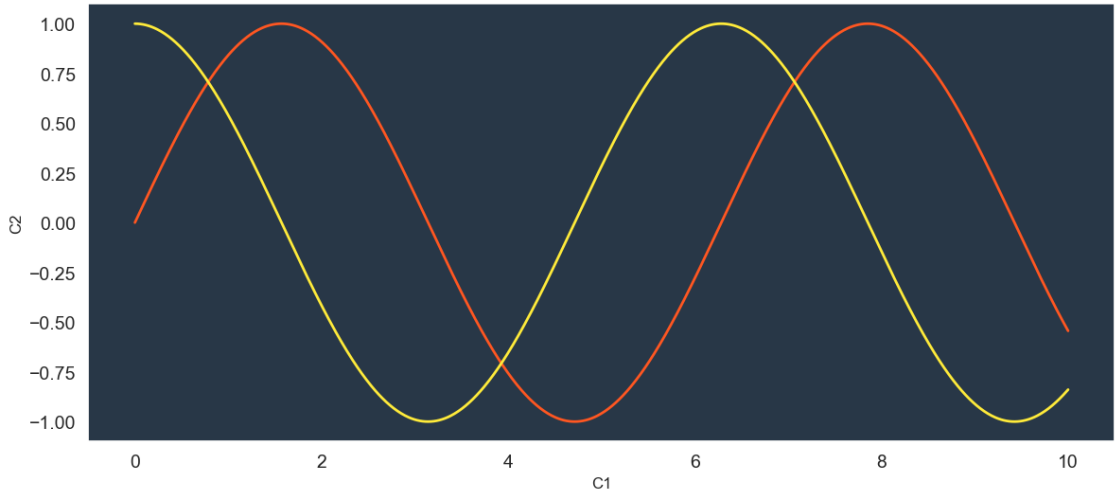
x-axis labeled with values from C1. The plot also has a dark blue background, a grid, and tick labels with a font size of 14. The **linewidth** parameter sets the thickness of the lines.

[7]:
```
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()
```



- This is same as the above plot with slight difference in colour where instead of using name of the colour **HEX colour code** is used.

[8]:
```
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()
```



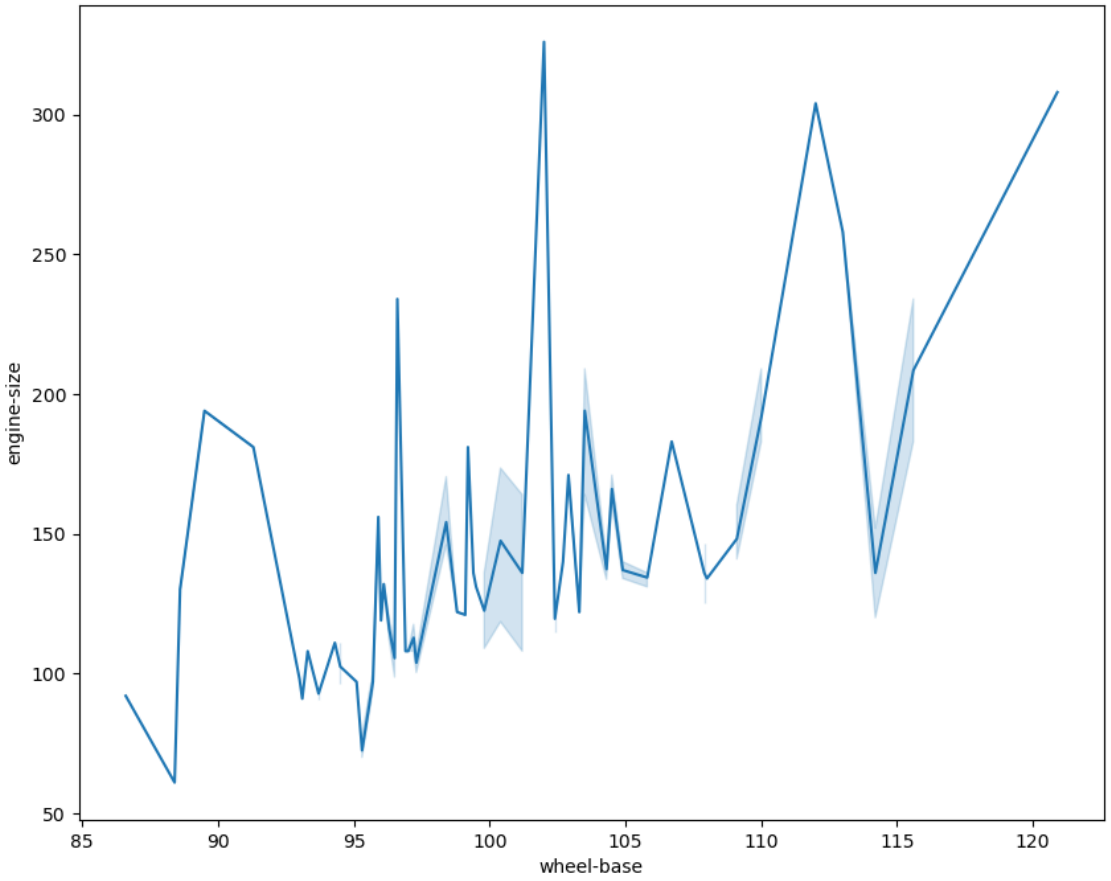
- **mpl.rcParams.update(mpl.rcParamsDefault)** resets all matplotlib parameters to their default values, so any customizations that were made previously will be discarded.
- **%matplotlib inline** is a magic command used in Jupyter notebooks to display plots inline, meaning that the plots will be displayed within the notebook itself rather than in a separate window or file.

[9]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

- This creates a figure with a size of 10 by 8 inches and then creates a line plot using the seaborn library.
- The x-axis represents the wheel base values, and the y-axis represents the engine size values.
- The data used for plotting is contained in a Pandas DataFrame named **db**.
- Finally, the **plt.show()** function is called to display the plot.

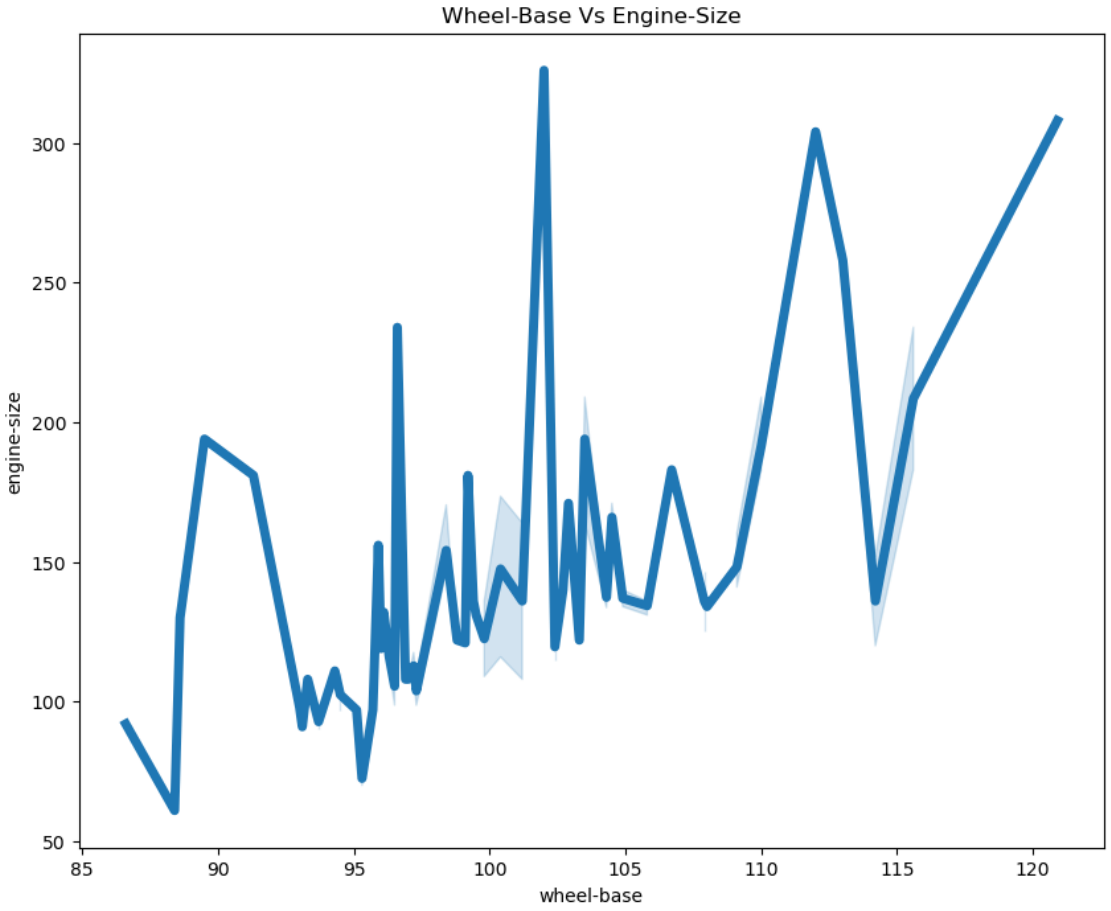
Note that **plt** is usually an alias for **matplotlib.pyplot**, which is commonly imported as **import matplotlib.pyplot as plt**.

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



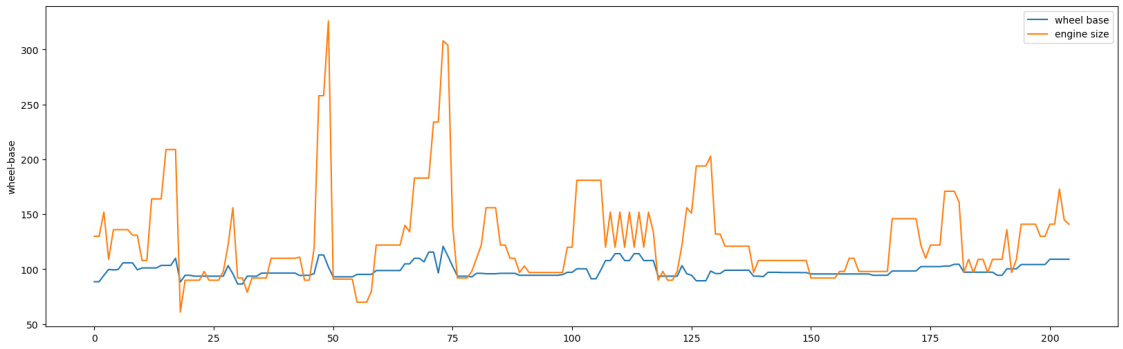
- Same as the above plot but changed the line width to **5**.

[11]:
```
plt.figure(figsize=(10,8))
sns.lineplot(x="wheel-base",y="engine-size",data=db,linewidth = 5)
plt.title("Wheel-Base Vs Engine-Size")
plt.show()
```



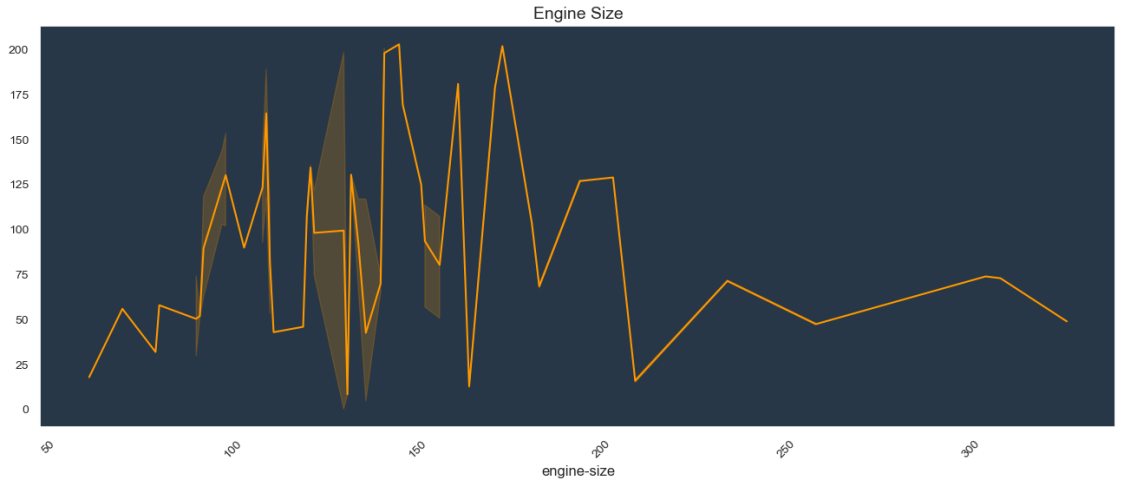
- Here two separate lines are plotted on the same figure using the **sns.lineplot()** function.
- The first line represents the wheel base values and is specified by passing **db['wheel-base']** as the data argument, **1.5** as the linewidth, and 'wheel base' as the label argument.
- The second line represents the engine size values and is specified by passing **db['engine-size']** as the data argument, **1.5** as the linewidth, and 'engine size' as the label argument.

[12]:
```
plt.figure(figsize=(20,6))
sns.lineplot(data=db['wheel-base'],linewidth = 1.5 , label = 'wheel base')
sns.lineplot(data=db['engine-size'],linewidth = 1.5 , label = 'engine size')
plt.show()
```



- **sns.set()** is used to set some parameters for the plot.
- **"axes.facecolor":"#283747"** sets the background color to a dark blue, **"axes.grid":False** removes the gridlines from the plot, and **'xtick.labelsize':10** and **'ytick.labelsize':10** set the font size for the x and y axis labels to 10.
- **plt.title()** is used to set the title of the plot to **"Engine Size"**, with a font size of 14. **plt.xticks(rotation=45)** rotates the x-axis tick labels by 45 degrees.
- Finally, **sns.lineplot()** is used to create a line plot with the engine size values on the x-axis and the index values of the wheel-base column on the y-axis.
- The color parameter sets the color of the line to a shade of orange.

[13]:
```
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)
sns.lineplot(x = db["engine-size"], y =db["wheel-base"].index.values  , color =␣
 ↪'#ff9900')
plt.show()
```



- It creates a figure with a size of 14 by 7 inches and applies the **seaborn-darkgrid** style using the **plt.style.use()** function.
- **sns.lineplot()** is then used to create a line plot with the wheel base values on the x-axis and the engine size values on the y-axis.
- The hue parameter is set to **"fuel-type"**, which means that the plot will display separate lines for each unique value in the fuel-type column of the **db** DataFrame.

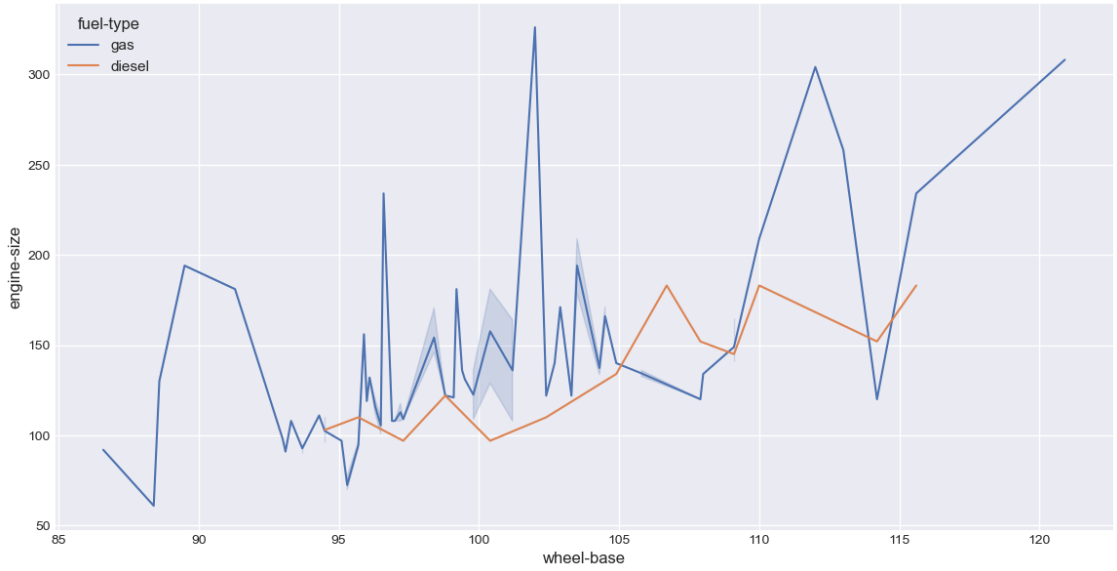
In Python's Seaborn library, **hue** is a parameter used in various plotting functions to group the data by a categorical variable and differentiate the groups by color.

When using hue, the data is divided into subsets based on the unique values of the categorical variable specified in the hue parameter. Then, each subset is plotted using a different color or marker style to distinguish the groups.

For example, if we have a dataset with car information and we want to visualize the relationship between engine size and fuel efficiency, we can use the hue parameter to differentiate the cars by fuel type. The resulting plot will have a separate line or marker style for each fuel type, making it easy to see how each fuel type affects the relationship between engine size and fuel efficiency.

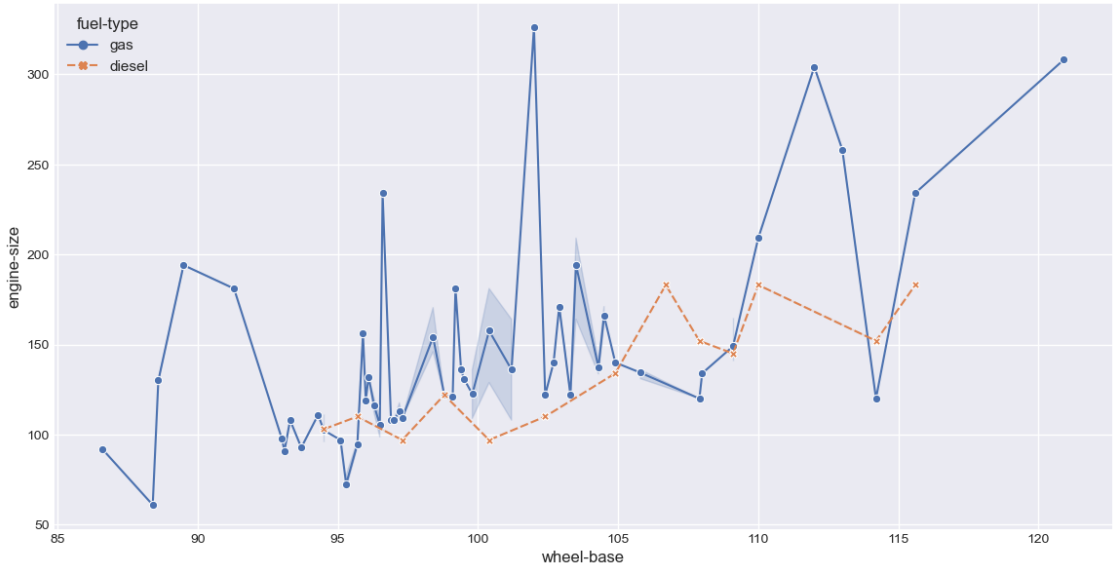
The specific options available for hue depend on the type of plot and the columns in the dataset. In general, hue is used for categorical variables, such as strings or discrete numbers, and can be applied to most Seaborn plotting functions that support it.

[14]:
```
plt.figure(figsize=(14,7))
plt.style.use('seaborn-darkgrid')
sns.lineplot(x="wheel-base" , y="engine-size" , hue="fuel-type" , data=db)
plt.show()
```



This plot is same as the above graph but **markers=True** is used to add markers to each data point.

[15]:
```
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=db)
plt.show()
```



This graph is also similar to the previous graph but instead of wheel-base num-of-cylinders is used and the **err_style** parameter is set to "bars" to add error bars to the plot.

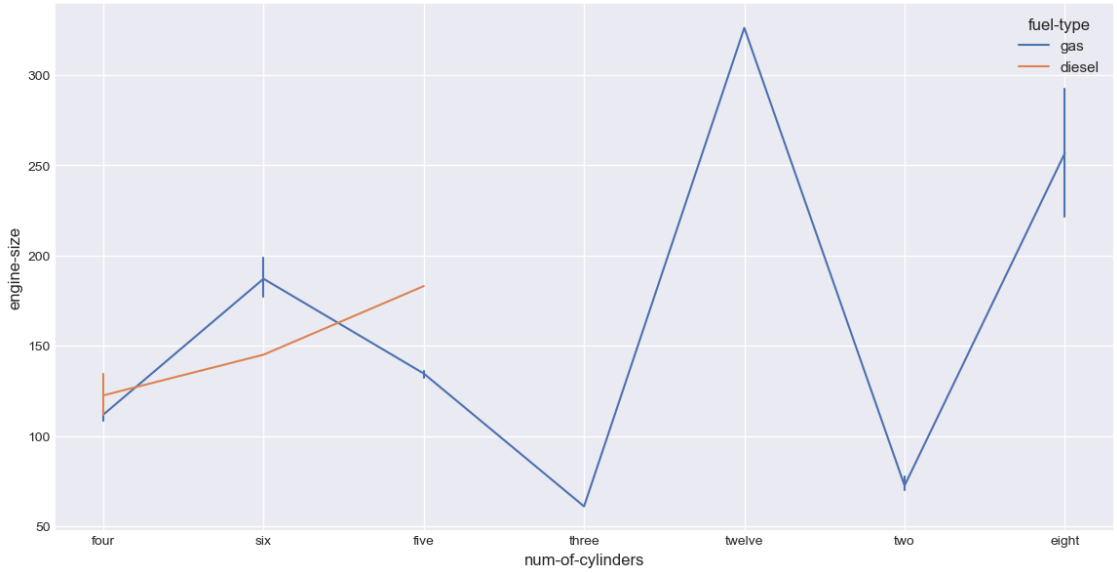
- In Python's Seaborn library, **err_style** is a parameter used in certain plotting functions to control the style of error bars.
- **Error bars** can be used to visualize uncertainty in the data, and they can be added to various plots using the **err_style** parameter in Seaborn.

The available styles for err_style parameter include:

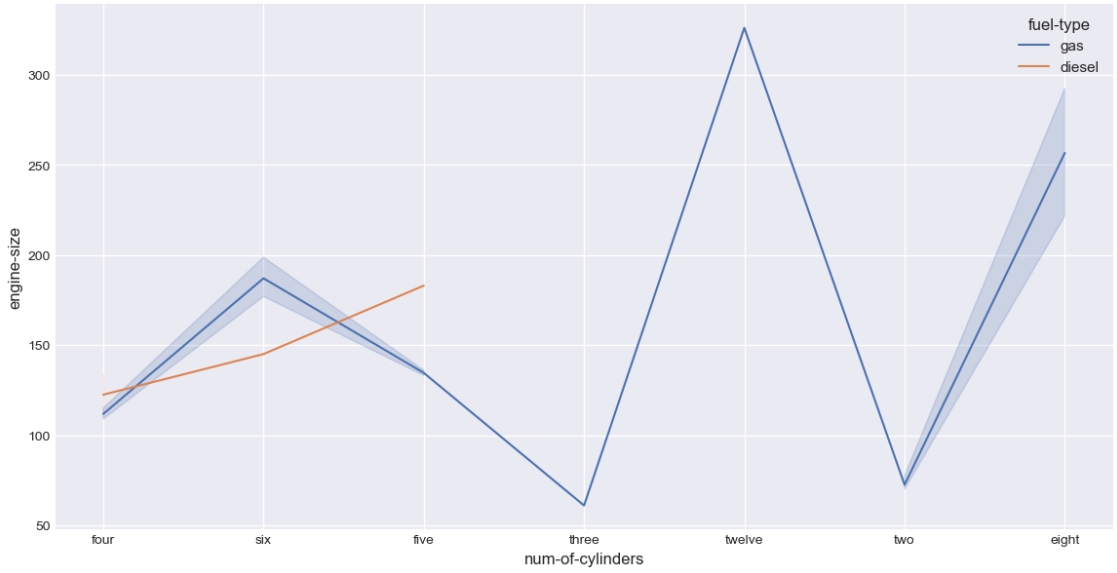
- **"bars"**: Adds error bars to the plot using vertical or horizontal bars.
- **"band"**: Adds error bars to the plot using shaded areas around the line plot.
- **"boot_traces"**: Uses bootstrapping to add error bars to the plot.
- **"bars"** is the default err_style in most Seaborn plotting functions.

The specific err_style options available may depend on the type of plot and the version of Seaborn being used.

[16]:
```
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=db)
plt.show()
```



[17]:
```
plt.figure(figsize=(14,7))
sns.lineplot(x = "num-of-cylinders", y = "engine-size",data=db,hue="fuel-type")
sns.set(style='dark',)
plt.show()
```



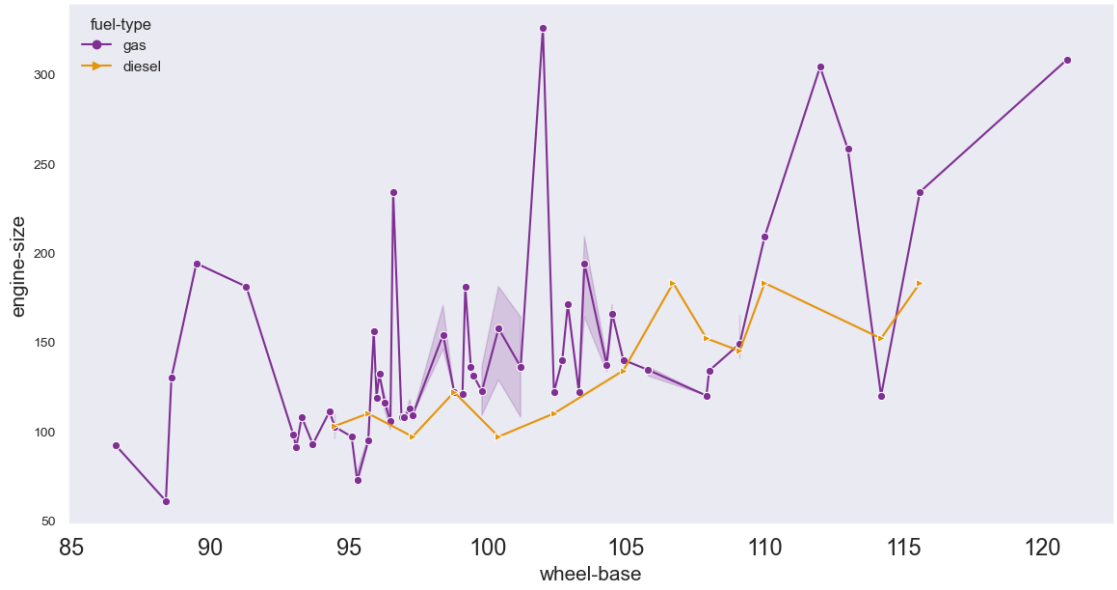
- The **sns.set()** function is then used to set the style parameters for the plot.
- Specifically, it sets the size of the x and y tick labels using **xtick.labelsize** and **ytick.labelsize**, respectively.

- It also sets the font size for the x and y axis labels using **axes.labelsize**.
- Additionally, it sets **axes.grid** to False to turn off the grid lines in the plot.
- The **sns.lineplot()** function is then used to create the line plot.
- The x-axis is set to "wheel-base" and the y-axis is set to "engine-size".
- The "fuel-type" column is used to group the data points and differentiate the lines in the plot.
- The style parameter is set to "fuel-type" so that the style of each line is determined by the "fuel-type" column.
- The **dashes** parameter is set to False to display the lines as solid.
- The palette parameter is set to **'CMRmap'** to specify the color map to be used for the plot.
- Finally, **markers=["o", ">"]** is used to add markers to each data point with circles for one fuel type and triangles for the other.
- **CMRmap** is a color map or palette available in the Seaborn library in Python.
- A color map is a mapping between a range of values and a range of colors.
- In Seaborn, color maps are used to specify the color scheme of a plot.
- CMRmap is a color map that goes from dark blue, through green, yellow and orange, to dark red.

The name CMRmap stands for Center for Marine Resources, and it was developed by the team at the Center for Coastal and Ocean Mapping at the University of New Hampshire. It is particularly useful for visualizing data that ranges from positive to negative values, as it provides a clear way to differentiate positive and negative values with contrasting colors.

Here, CMRmap is used as the color map for the hue parameter in the sns.lineplot() function. It is passed as an argument to the palette parameter. This means that different colors from the CMRmap color map will be used to differentiate the lines based on the fuel-type column.

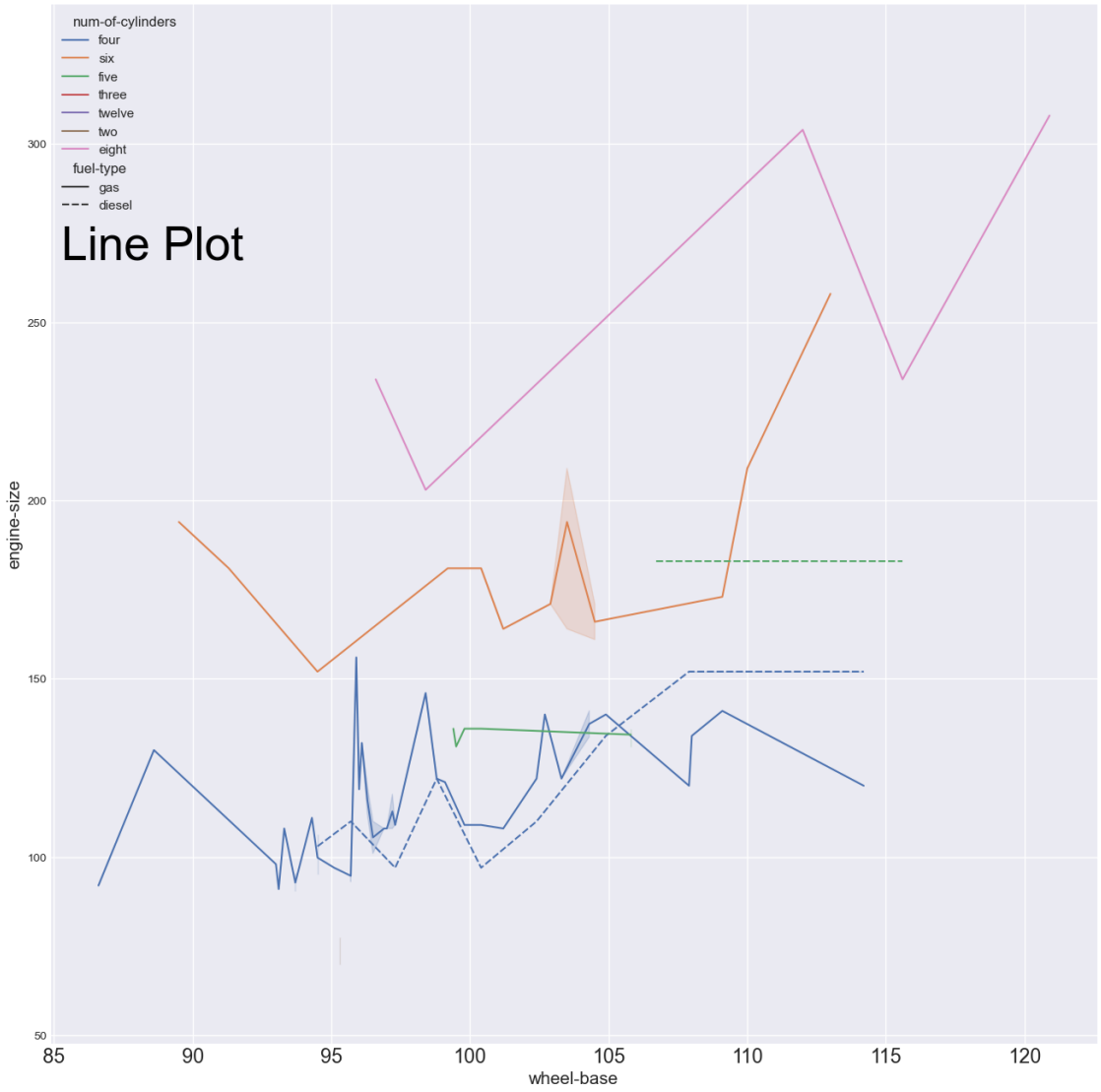
[18]:
```
plt.figure(figsize=(14,7))
sns.set(rc={'xtick.labelsize':17,'ytick.labelsize':10,'axes.labelsize':15 ,␣
 ↪"axes.grid":False})
sns.lineplot(x="wheel-base" , y="engine-size" , hue = "fuel-type"␣
 ↪,style="fuel-type" , dashes = False,palette = 'CMRmap',markers=["o", ">"]␣
 ↪,data=db)
plt.show()
```



- The **plt.gcf().text()** function is used to add a title to the plot. This function places text on the current figure at the specified position. In this case, it places the text **"Line Plot"** at position (0.2, 0.7) of the figure with fontsize 40, black color, centered horizontally and vertically using **ha='center', va='center'**.
- The **sns.lineplot()** function is then used to create the line plot. The x-axis is set to "wheel-base" and the y-axis is set to "engine-size". The "num-of-cylinders" column is used to group the data points and differentiate the lines in the plot. The style parameter is set to "fuel-type" so that the style of each line is determined by the "fuel-type" column. The data used to create the plot is stored in the db variable.
- Since no marker or color is specified, the plot will use the default colors and markers to differentiate the lines based on the num-of-cylinders column.

[19]:
```
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=db)
```

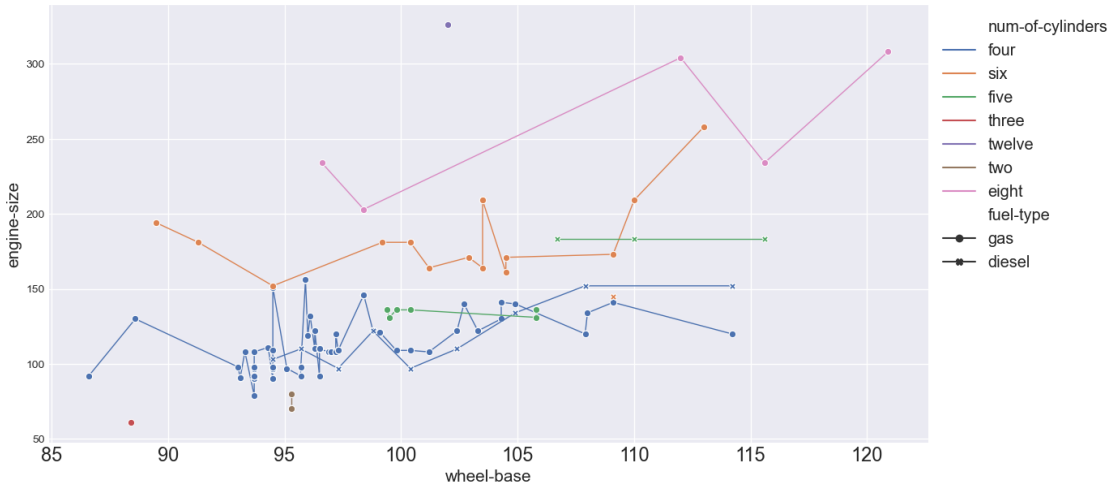
[19]:
```
<AxesSubplot:xlabel='wheel-base', ylabel='engine-size'>
```



- The x-axis is set to "wheel-base" and the y-axis is set to "engine-size". The "num-of-cylinders" column is used to group the data points and differentiate the lines in the plot. The style parameter is set to "fuel-type" so that the style of each line is determined by the "fuel-type" column.
- The **units parameter** is set to 'num-of-cylinders', which means that the lines will be split and drawn separately for each unique value of the "num-of-cylinders" column.
- The **markers parameter** is set to True, which means that markers will be added to the lines. The **dashes parameter** is set to False, which means that the lines will not be dashed.
- The **estimator parameter** is set to None, which means that the function will not try to estimate the central tendency and confidence intervals for each group. The **lw** parameter is set to 1, which means that the line width will be set to 1.

- The **data parameter** specifies the dataset to be used to create the plot, and in this case it is stored in the auto variable.
- The **plt.legend()** function is used to add a legend to the plot. The **bbox_to_anchor parameter** sets the location of the legend to the top right corner of the plot, and **shadow and fontsize parameters** set the shadow effect and font size of the legend, respectively.

[20]:
```
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',markers=True,dashes=False␣
 ↪,estimator=None,lw=1 ,data=db)
plt.legend(bbox_to_anchor=(1.0, 1.0) , shadow=True, fontsize='large')
plt.show()
```



- This code creates a facet grid of line plots using the Seaborn library. The **plt.figure(figsize=(10,10))** function sets the figure size to 10 by 10 inches, and **sns.set(rc={'xtick.labelsize':17,'ytick.labelsize':17,'axes.labelsize':20 , "axes.grid":False})** sets the style of the plot by changing the font size and turning off the grid.
- The **sns.relplot()** function is used to create the facet grid of line plots. The x-axis is set to "wheel-base" and the y-axis is set to "engine-size". The "num-of-cylinders" column is used to group the data points and differentiate the lines in the plot. The hue parameter is set to "num-of-cylinders" to create a different line for each unique value of the "num-of-cylinders" column.
- The col parameter is set to "fuel-type", which creates a separate plot for each unique value of the "fuel-type" column. The **kind parameter** is set to "line", which means that the plot type is a line plot. The **height parameter** is set to 8.5, which sets the height of each plot in the facet grid to 8.5 inches.

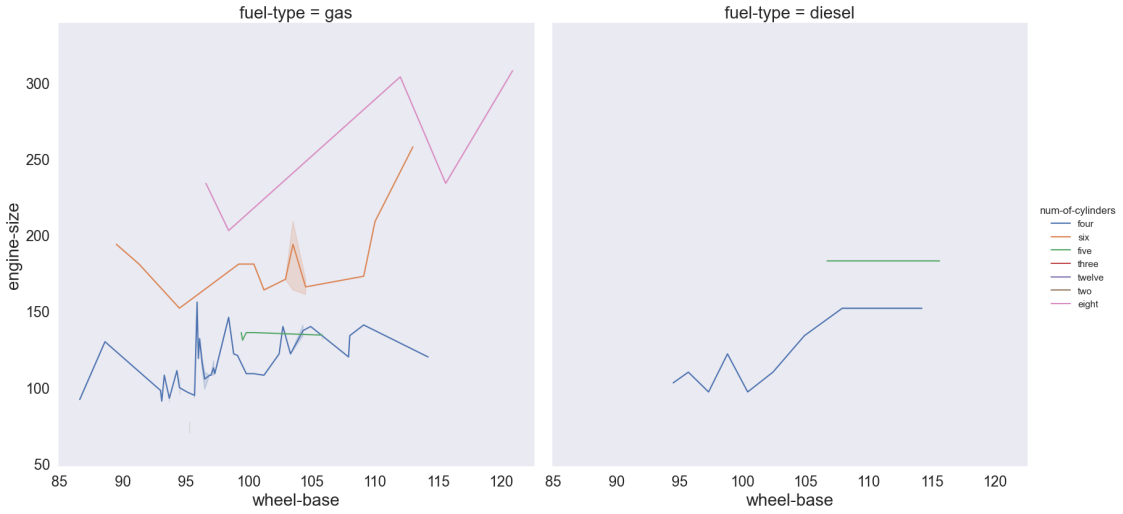
A facet grid is a plotting layout that displays multiple plots, or facets, arranged in a grid format.

Each plot shows a subset of the data based on the values of one or more categorical variables. In a facet grid, each row or column of plots corresponds to a unique combination of values from one or more categorical variables.

Facet grids are useful for exploring the relationships between different variables in a dataset and comparing patterns across different subsets of the data. They can be created using various plotting libraries in Python, such as Seaborn and ggplot.

[21]:
```
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=db)
plt.show()
```

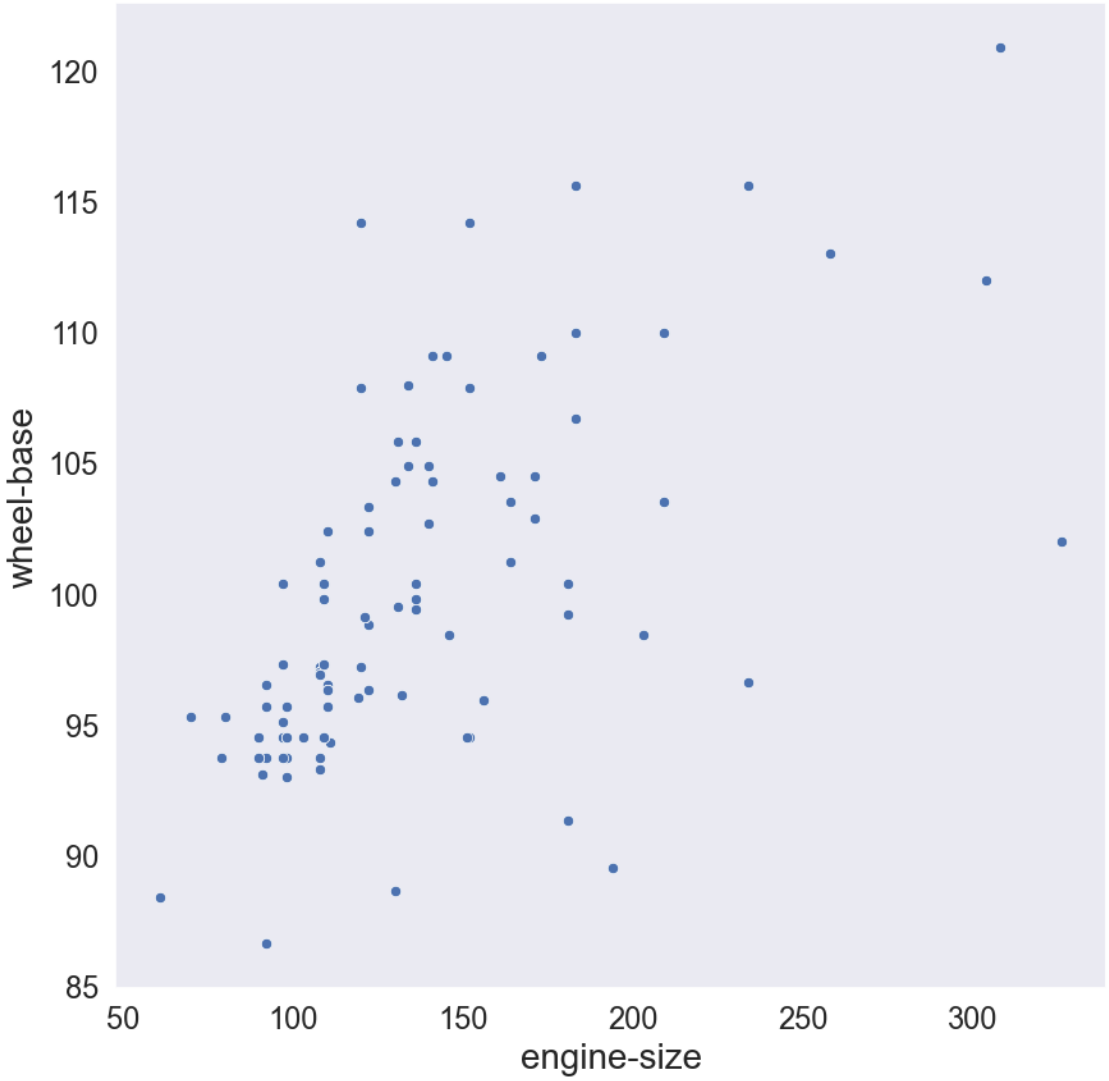
```
<Figure size 1000x1000 with 0 Axes>
```



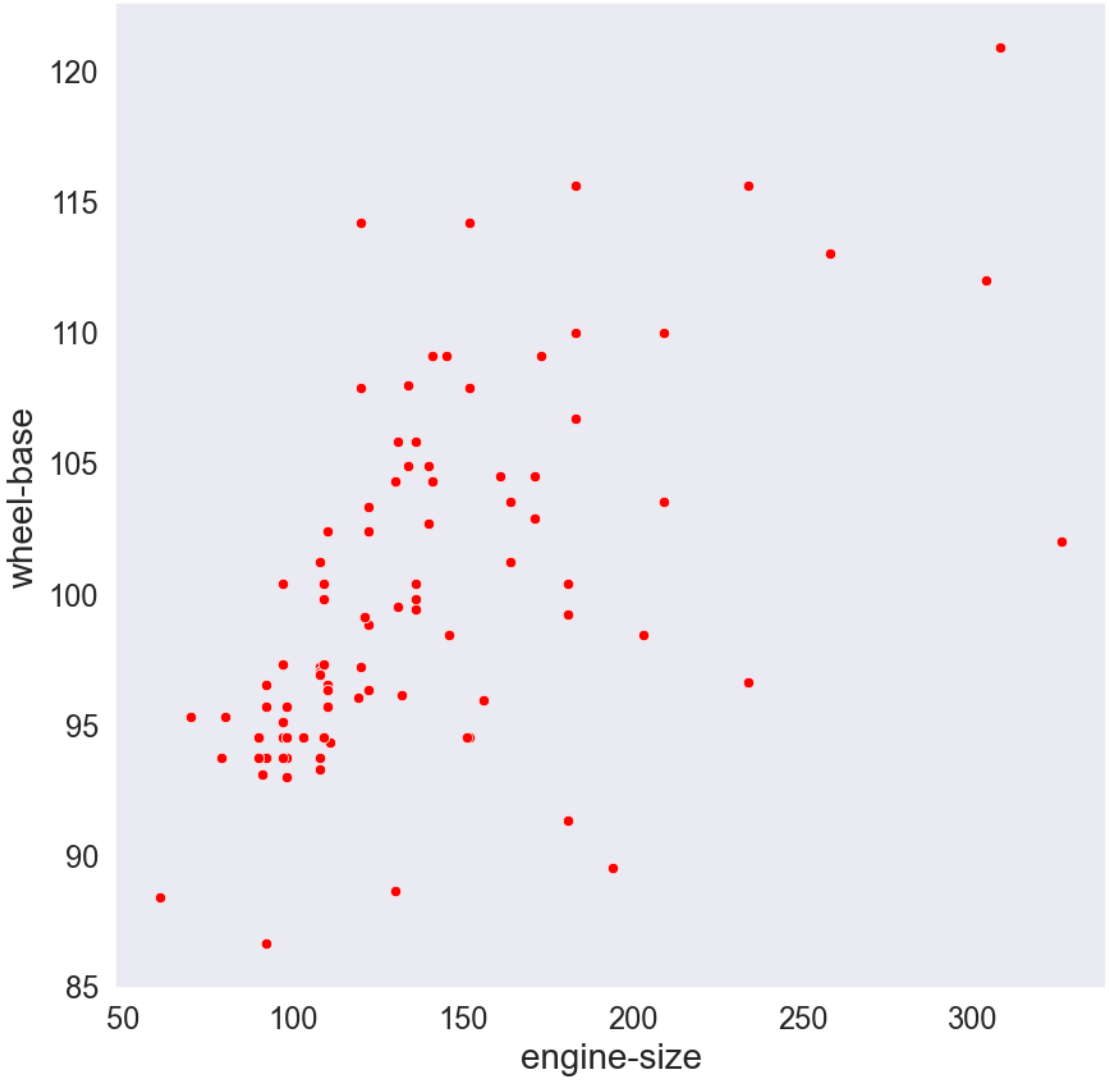
1.3 Scatterplots

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.

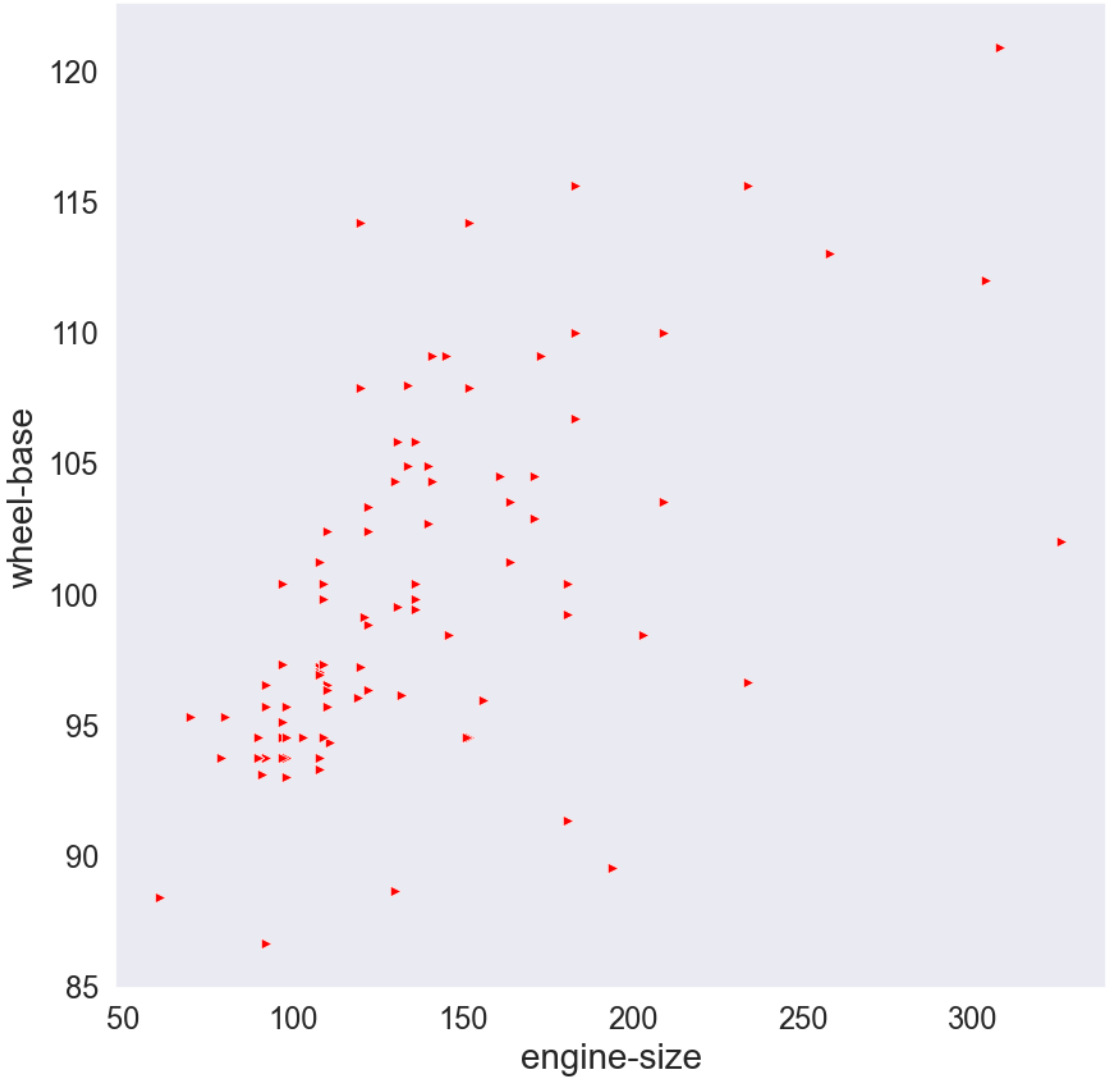
[22]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',data=db)
plt.show()
```



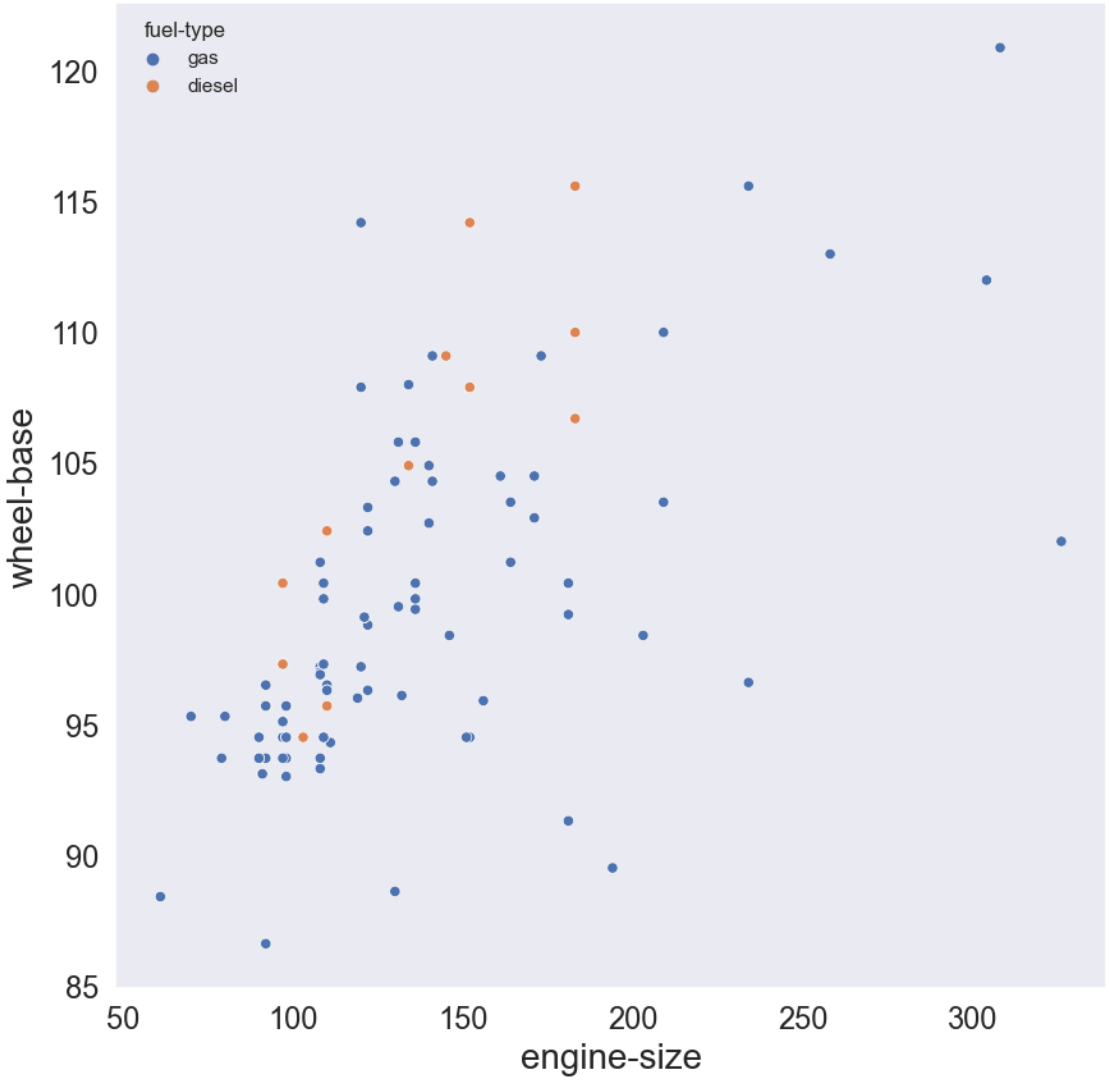
[23]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',data=db,color='#FF0000')
plt.show()
```



[24]:
```
plt.figure(figsize=(10,10))
sns.
 ↪scatterplot(x='engine-size',y='wheel-base',data=db,color='#FF0000',marker='>')
plt.show()
```

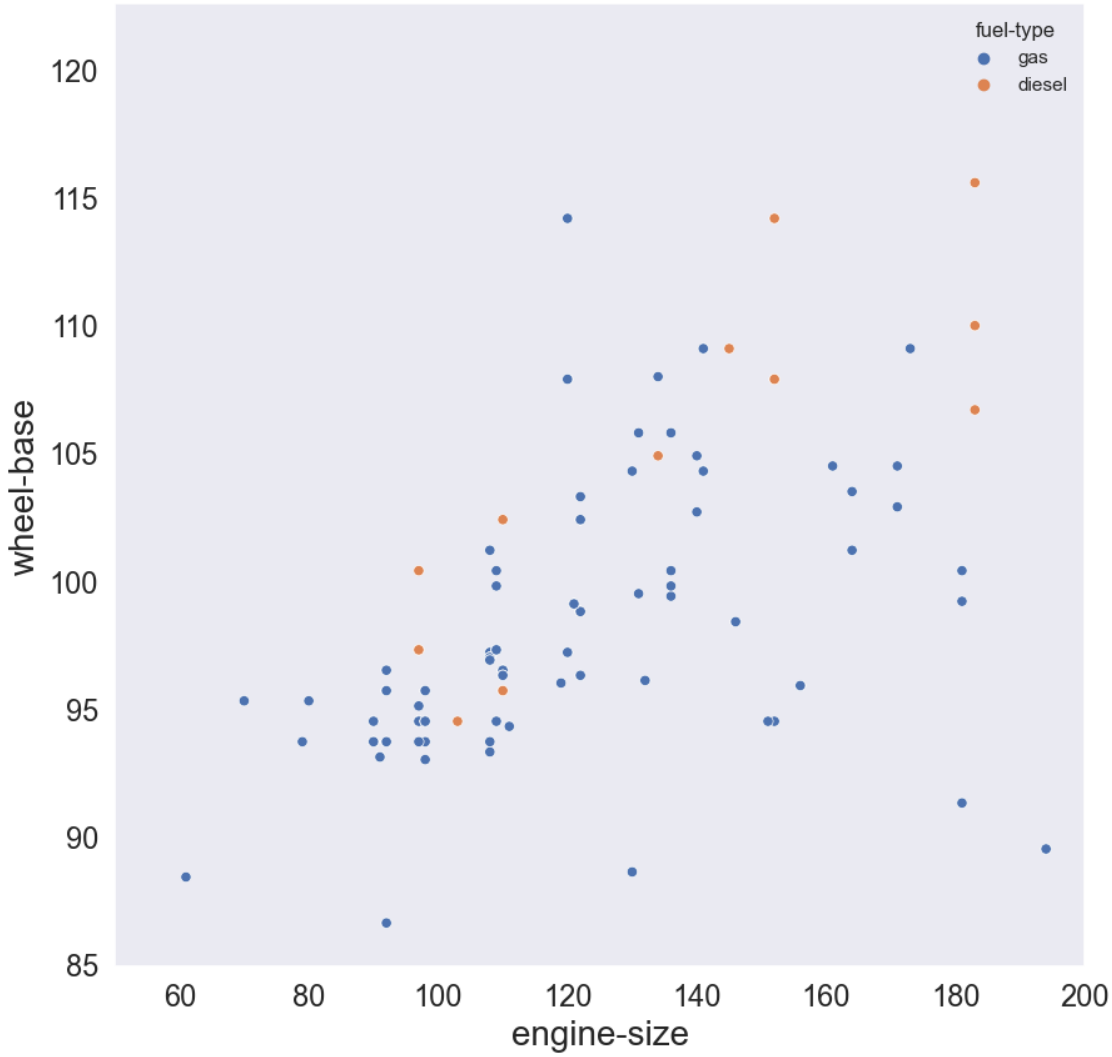


[25]:
```
plt.figure(figsize=(10,10))
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',data=db)
plt.show()
```

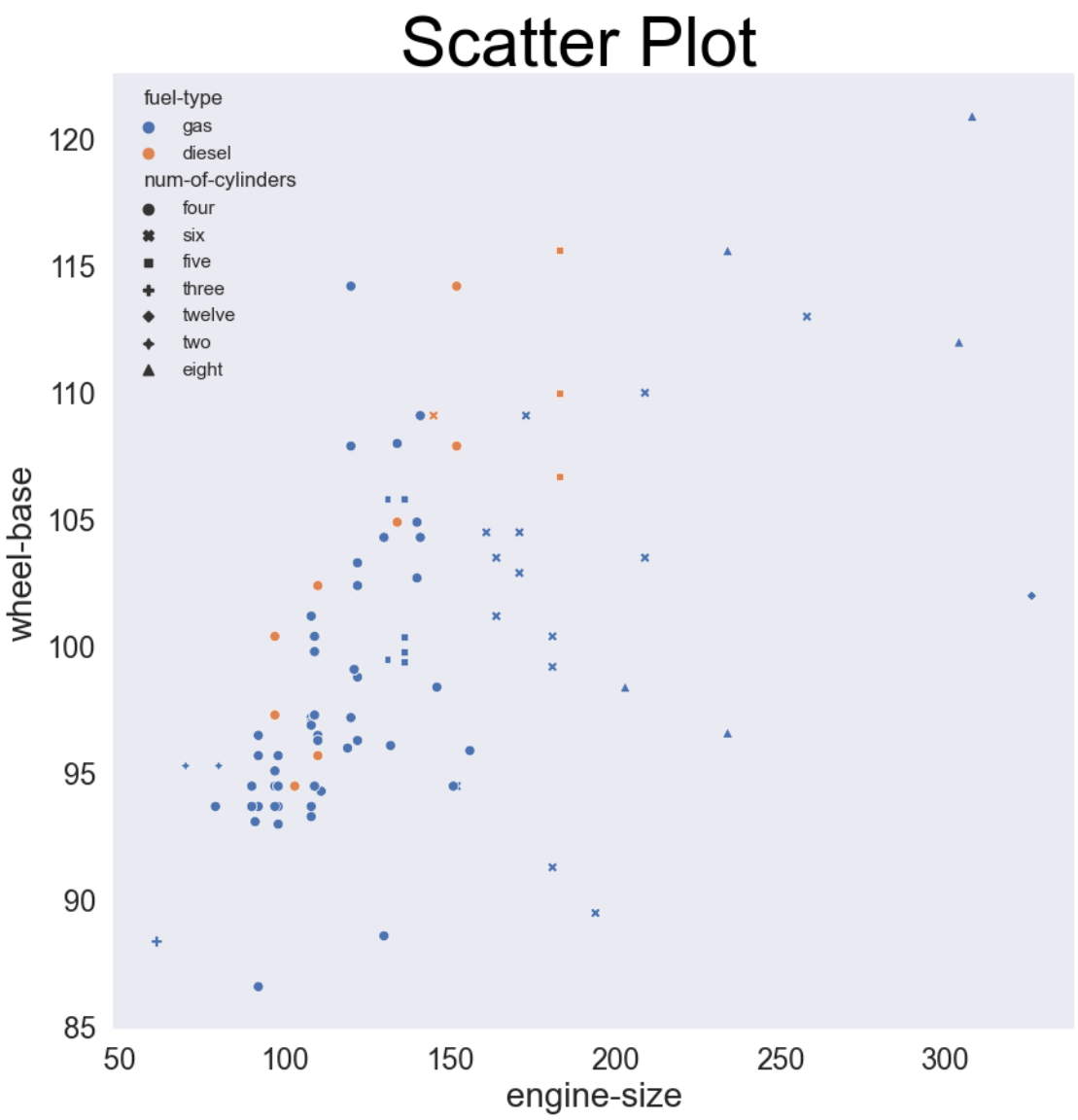


The **plt.xlim([50,200])** line sets the limits of the x-axis to range from 50 to 200. This means that the x-axis will only display values between 50 and 200.

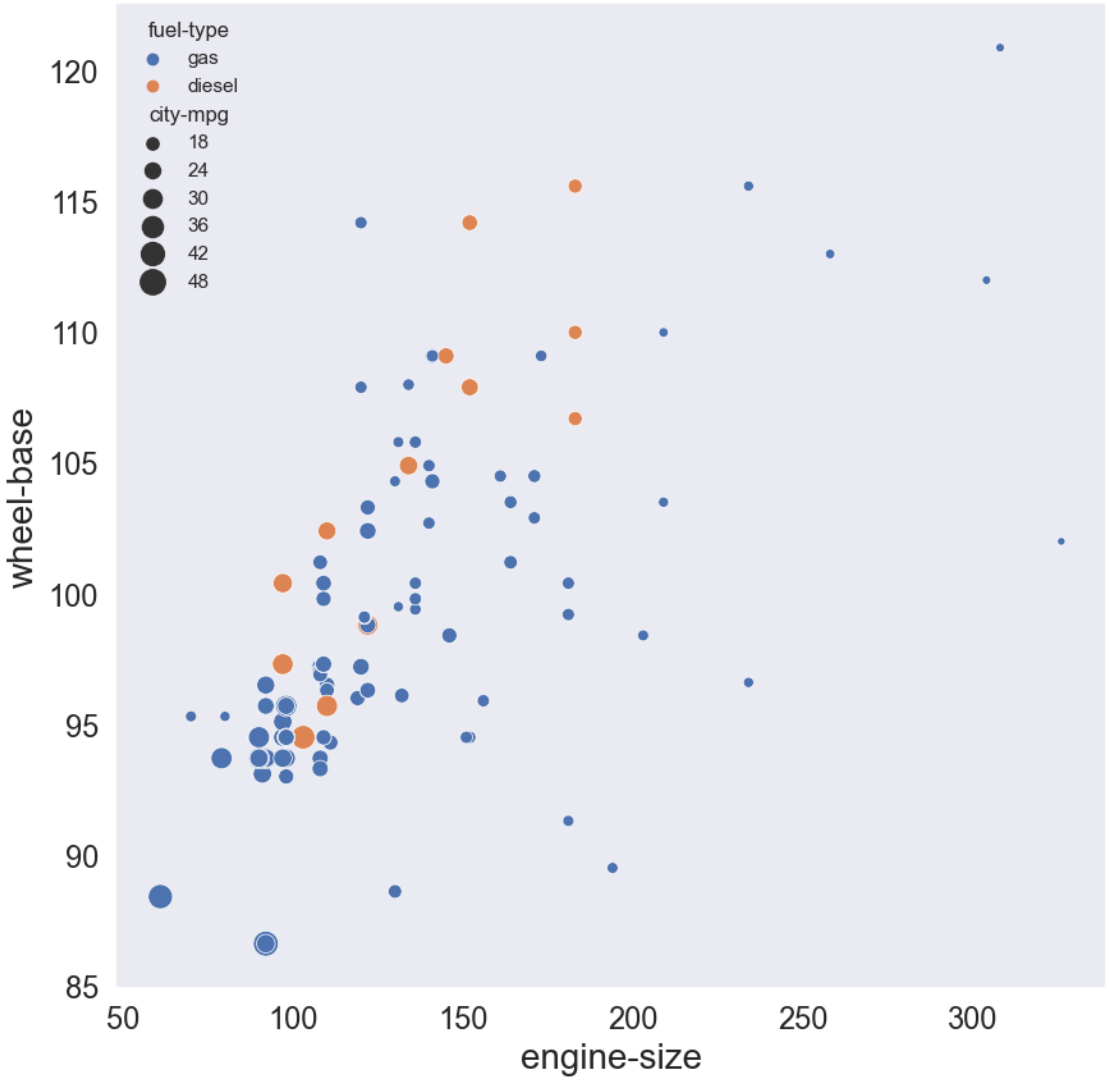
[26]:
```
plt.figure(figsize=(10,10))
plt.xlim([50,200])
sns.scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',data=db)
plt.show()
```



[27]:
```
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=db
plt.show()
```



[28]:
```
plt.figure(figsize=(10,10))
sns.
  ↪scatterplot(x='engine-size',y='wheel-base',hue='fuel-type',size='city-mpg',sizes=(20,200),d
plt.show()
```

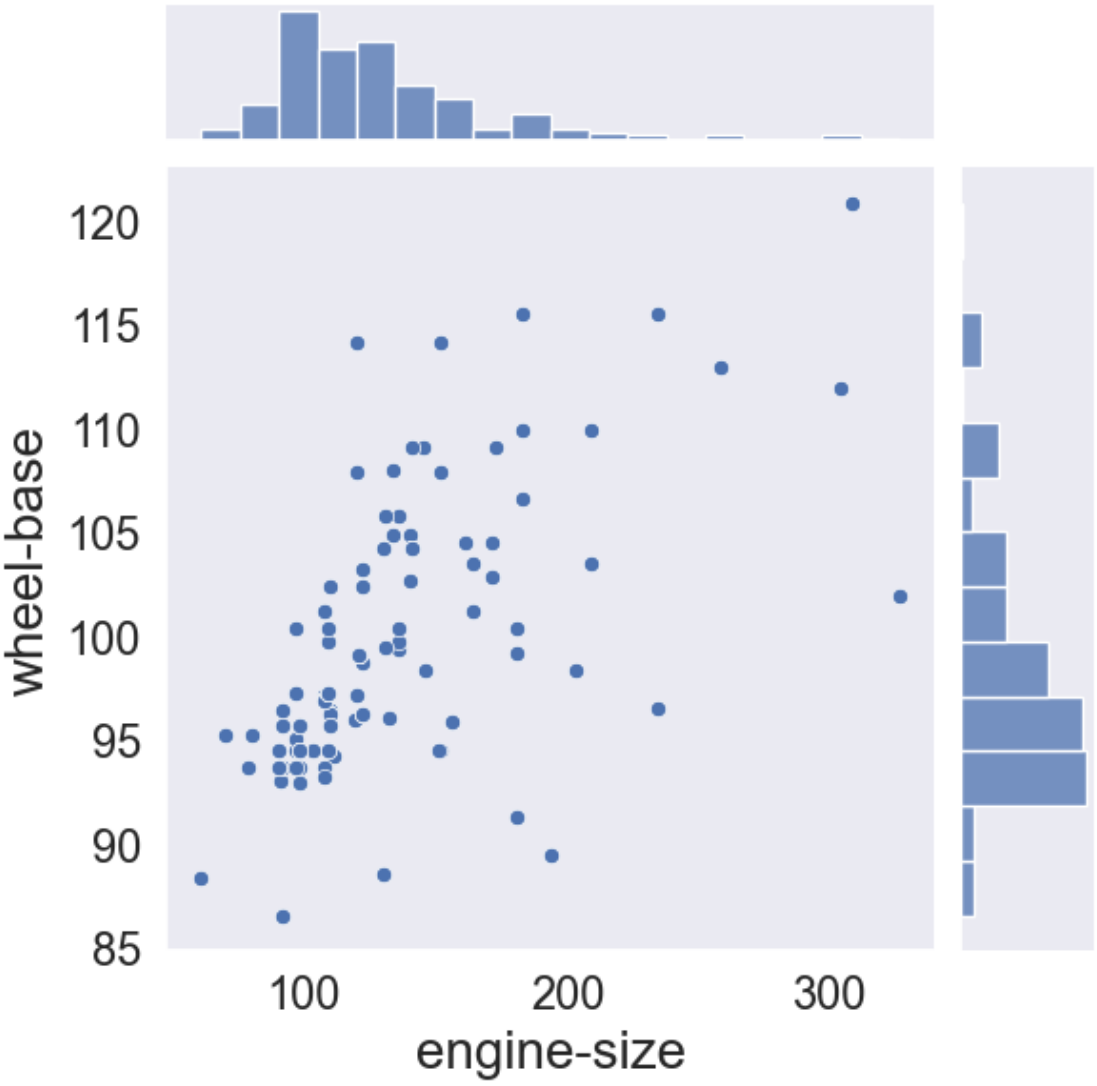


- DataFrame containing the relevant data, **sns.jointplot(db['engine-size'],db['wheel-base'])** would create a joint plot showing the relationship between the "engine-size" and "wheel-base" variables in the dataset.
- A **joint plot** is a type of plot in seaborn that allows you to visualize the relationship between two variables, along with the distribution of each variable separately. Specifically, the joint-plot function creates a scatter plot of the two variables with a **regression line**, along with **histograms** of each variable on their respective axes.
- So, in this case, the joint plot would show how the "engine-size" and "wheel-base" variables are related to each other, and provide information about their individual distributions as well. This can help to identify any patterns or correlations between the variables, which can be useful for further analysis.

[29]:
```
sns.jointplot(db['engine-size'],db['wheel-base'])
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[29]: <seaborn.axisgrid.JointGrid at 0x21718c4fdf0>



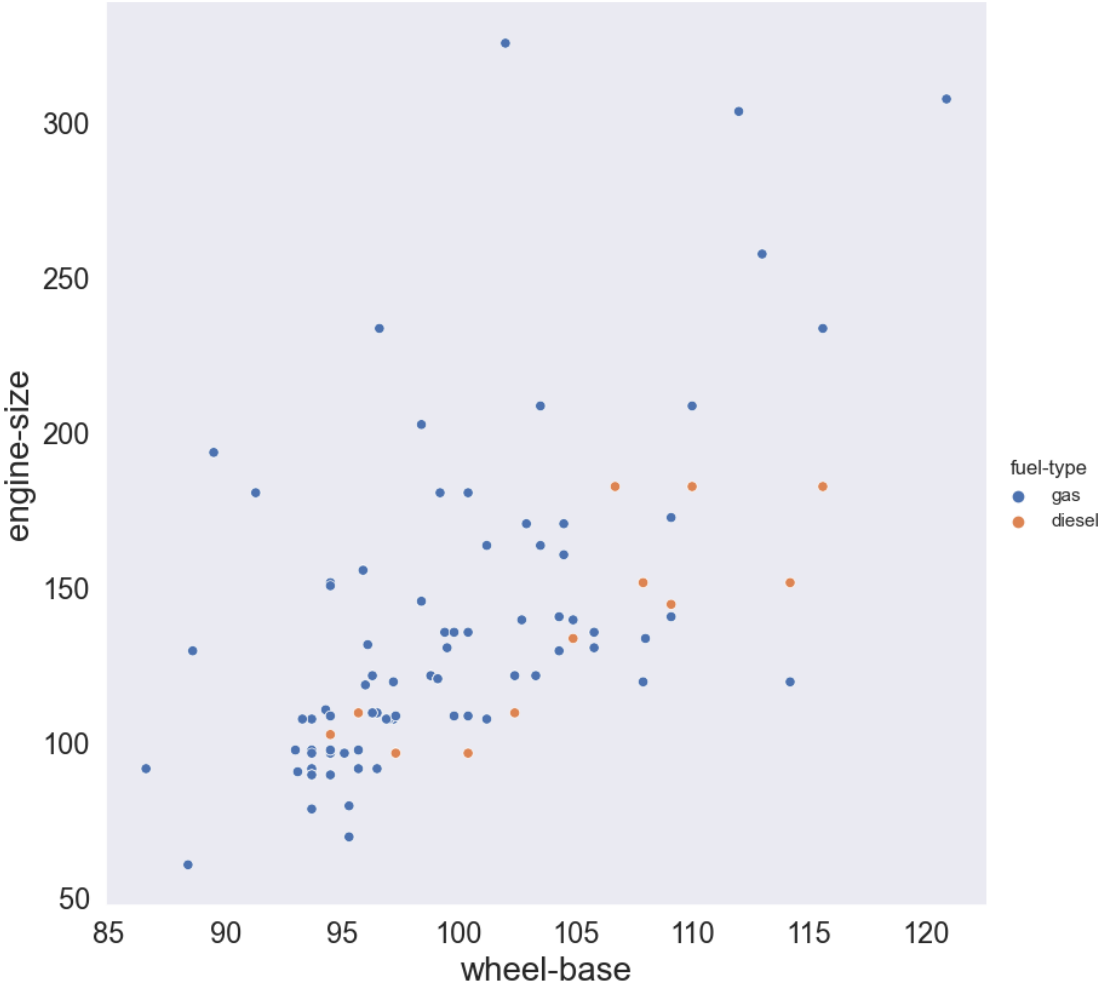
1.4 Rel plot

- **relplot** function can be used to create a variety of visualizations that show the relationship between two or more variables.

- relplot is short for **“relational plot”**, and it is a high-level interface for creating different types of plot using seaborn. It is used to create scatter plots, line plots, and other visualizations that show the relationship between two or more variables.
- The type of plot that is created by relplot depends on the kind parameter passed to the function. For example, kind=“scatter” will create a scatter plot, while kind=“line” will create a line plot. Other available kinds include kind=“strip”, kind=“swarm”, kind=“box”, kind=“violin”, kind=“bar”, and kind=“count”, each of which creates a different type of plot.
- In summary, relplot is a versatile function in the seaborn library that can be used to create a wide range of visualizations for exploring the relationship between variables in a dataset.

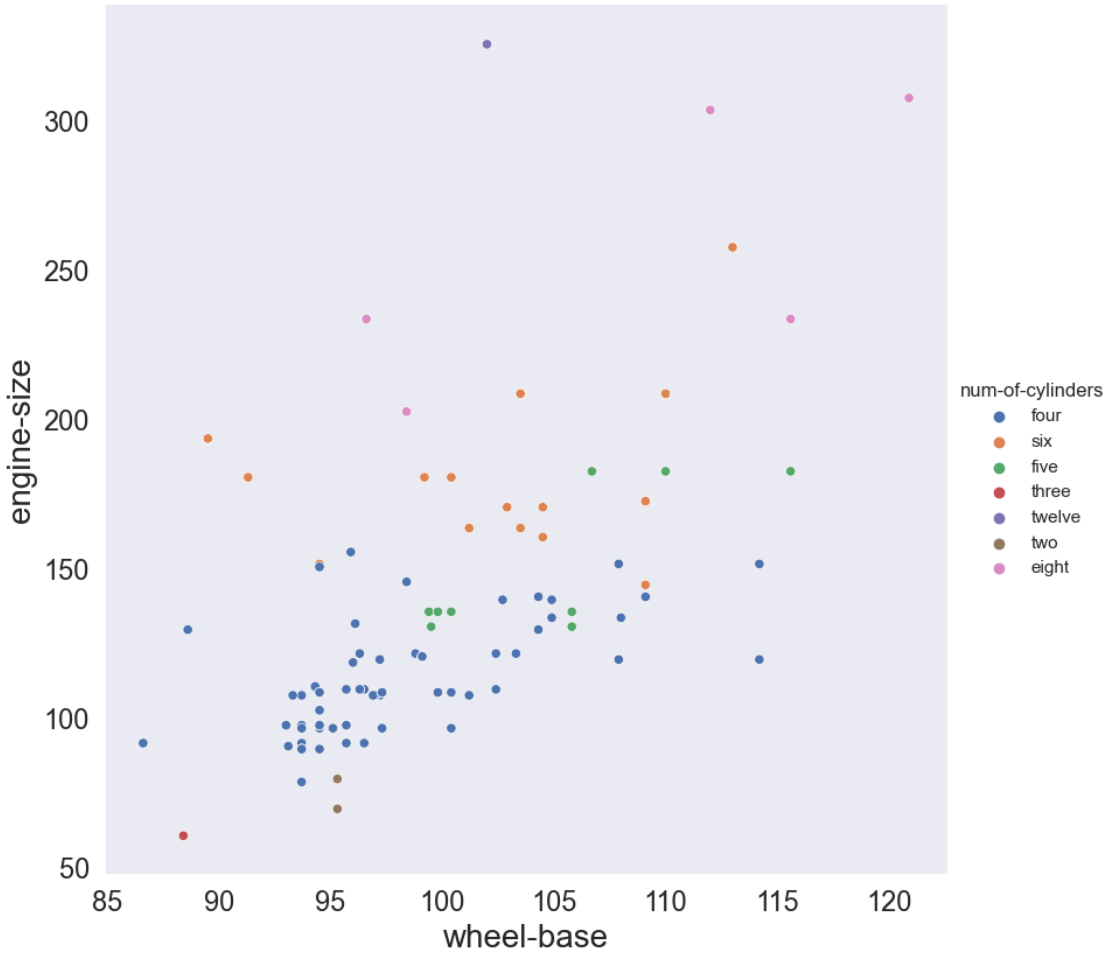
[30]:
```
plt.figure(figsize=(14,16))
sns.relplot(x="wheel-base" , y="engine-size" , hue="fuel-type" ,kind='scatter',␣
↪height=8.5,aspect =1,data=db)
plt.show()
```

```
<Figure size 1400x1600 with 0 Axes>
```



[31]:
```
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=db)
plt.show()
```

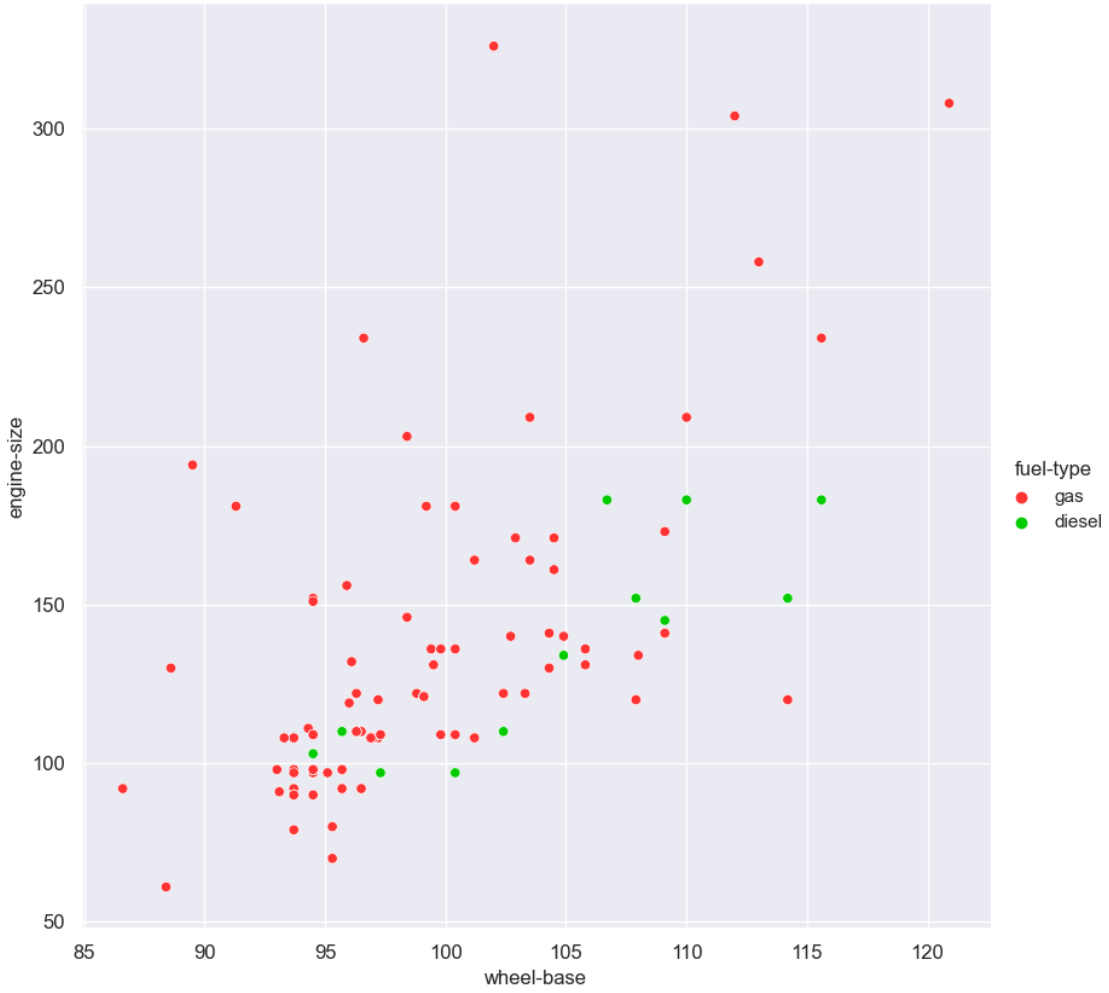
```
<Figure size 1400x1600 with 0 Axes>
```



Pallete function is used

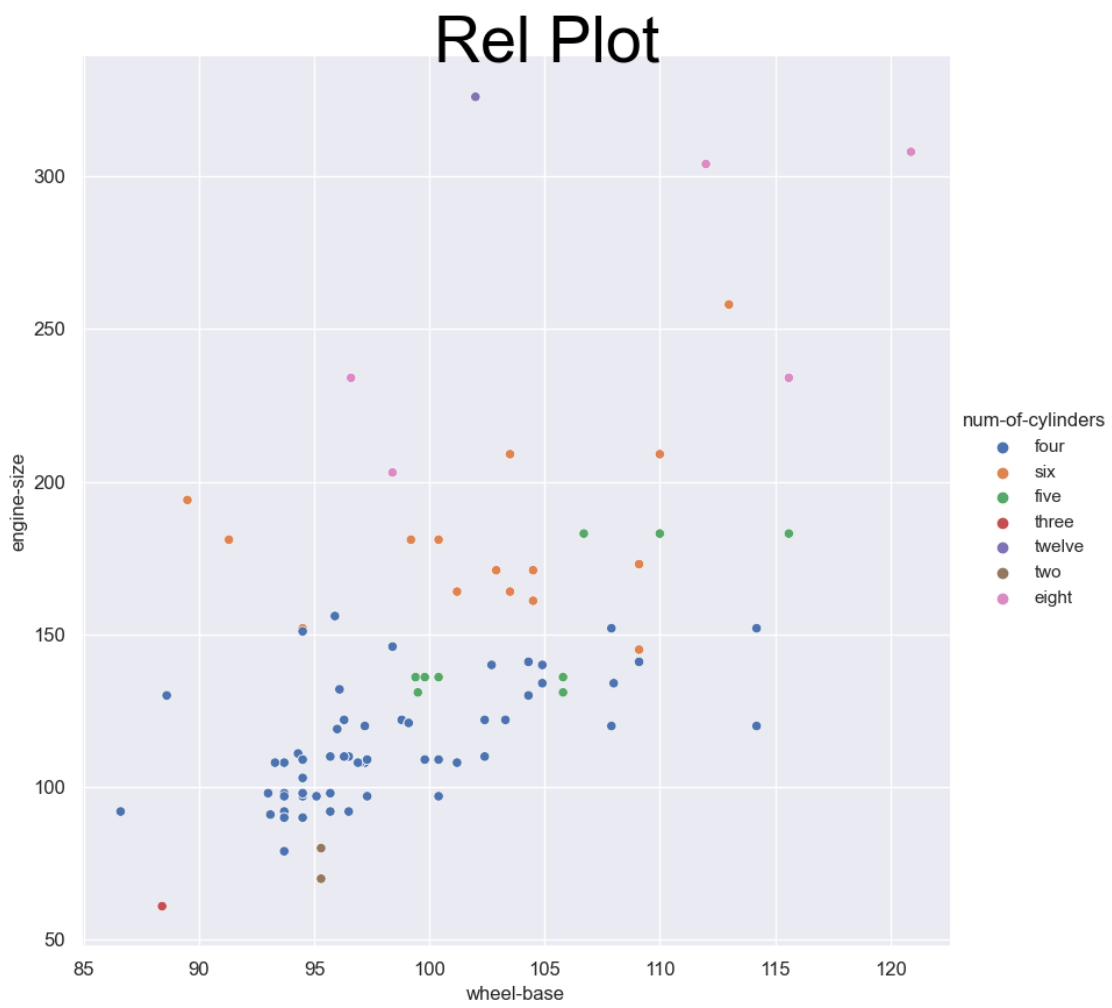
[32]:
```
plt.figure(figsize=(14,16))
sns.set(rc={'xtick.labelsize':12,'ytick.labelsize':12,'axes.labelsize':12})
sns.relplot(x="wheel-base" , y="engine-size" , hue="fuel-type"␣
 ↪,kind='scatter',palette=["#FF3333" ,"#00CC00"], height=8.5,aspect =1,data=db)
plt.show()
```

```
<Figure size 1400x1600 with 0 Axes>
```



[33]:
```
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=db)
plt.gcf().text(.5, .99, "Rel Plot", fontsize = 40, color='Black' ,ha='center',␣
 ↪va='center')
plt.show()
```

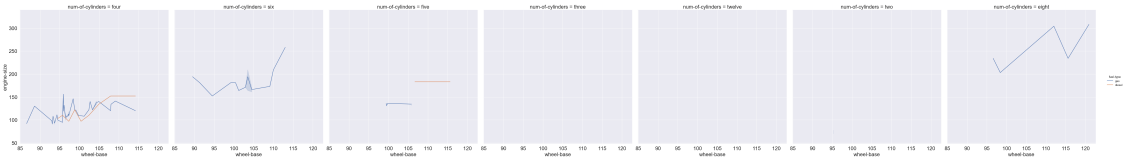
```
<Figure size 1400x1600 with 0 Axes>
```



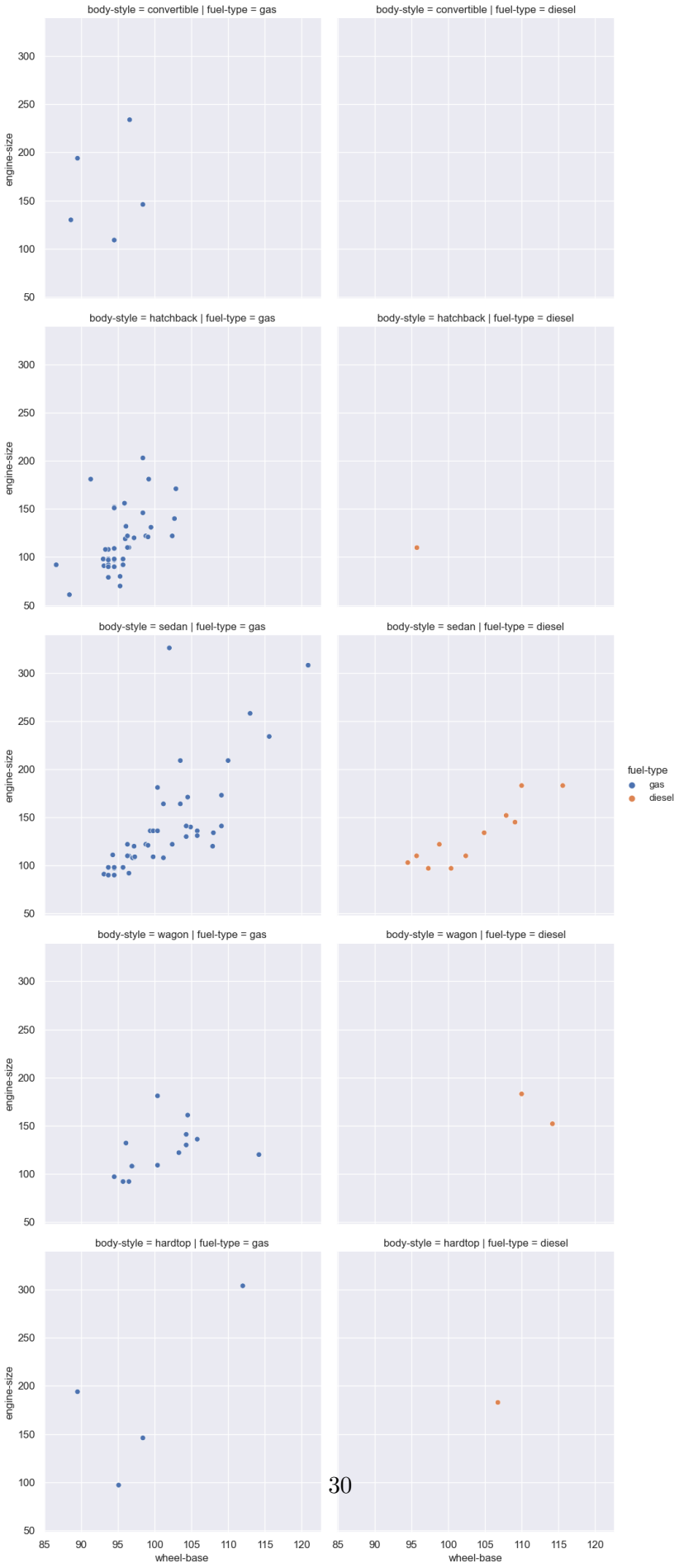
Facet is used.

[34]:
```
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 =1,data=db)
plt.show()
```

```
<Figure size 1400x2000 with 0 Axes>
```

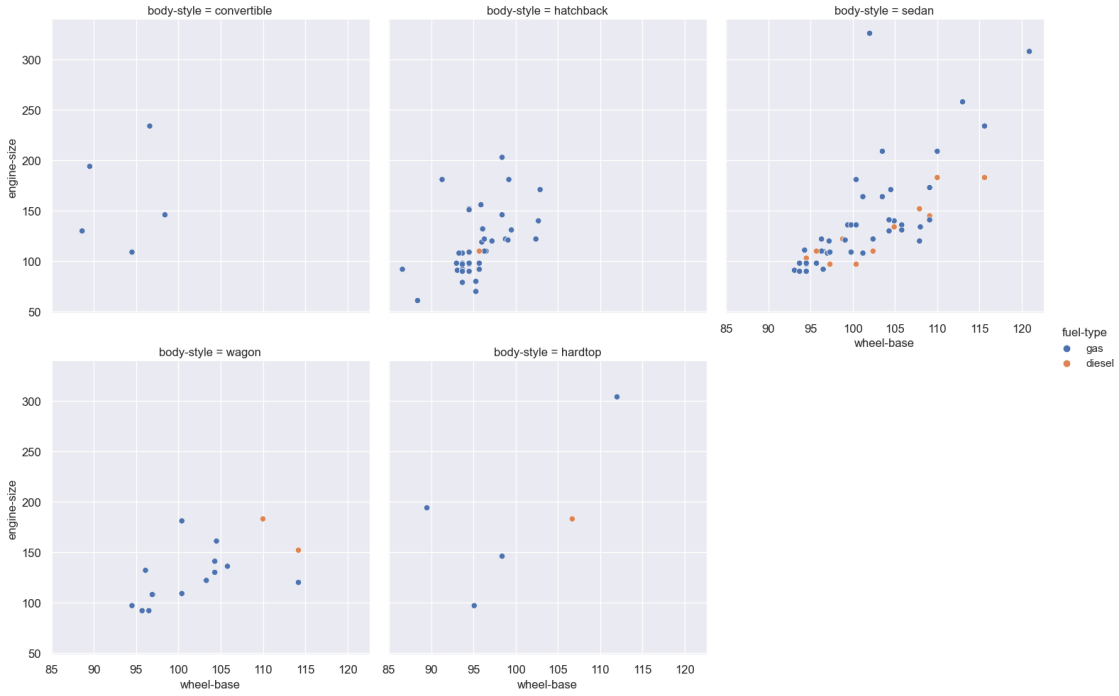


[35]:
```
sns.set(rc={'xtick.labelsize':12,'ytick.labelsize':12,'axes.labelsize':12})
sns.relplot(x="wheel-base", y="engine-size", hue="fuel-type",col="fuel-type",␣
 ↪row="body-style", data=db)
plt.show()
```


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
wheel-base
fuel-type
gas
diesel

Limiting the number of columns using **col_wrap**.

[36]:
```
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=db)
plt.show()
```



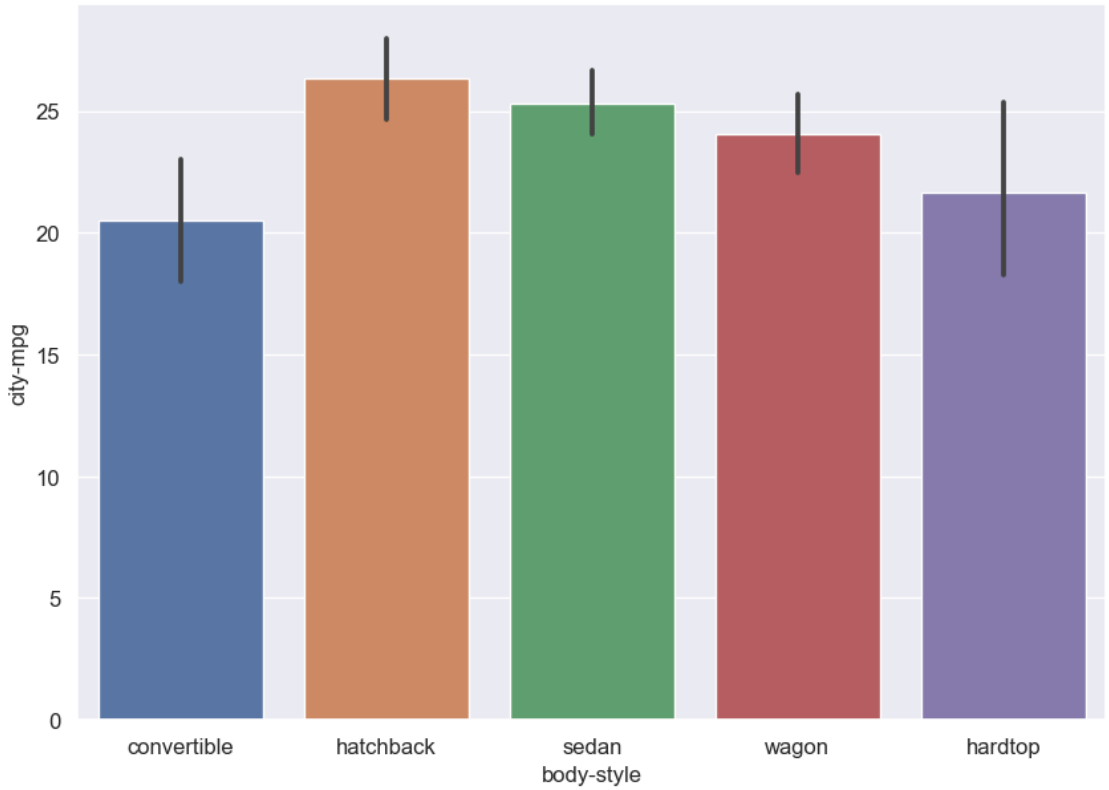
1.5 Bar Plot

- Bar Plot shows the relationship between a numerical variable and a categorical variable.
- 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.

[37]:
```
plt.figure(figsize=(10,7))
sns.barplot(db['body-style'], db['city-mpg'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
```

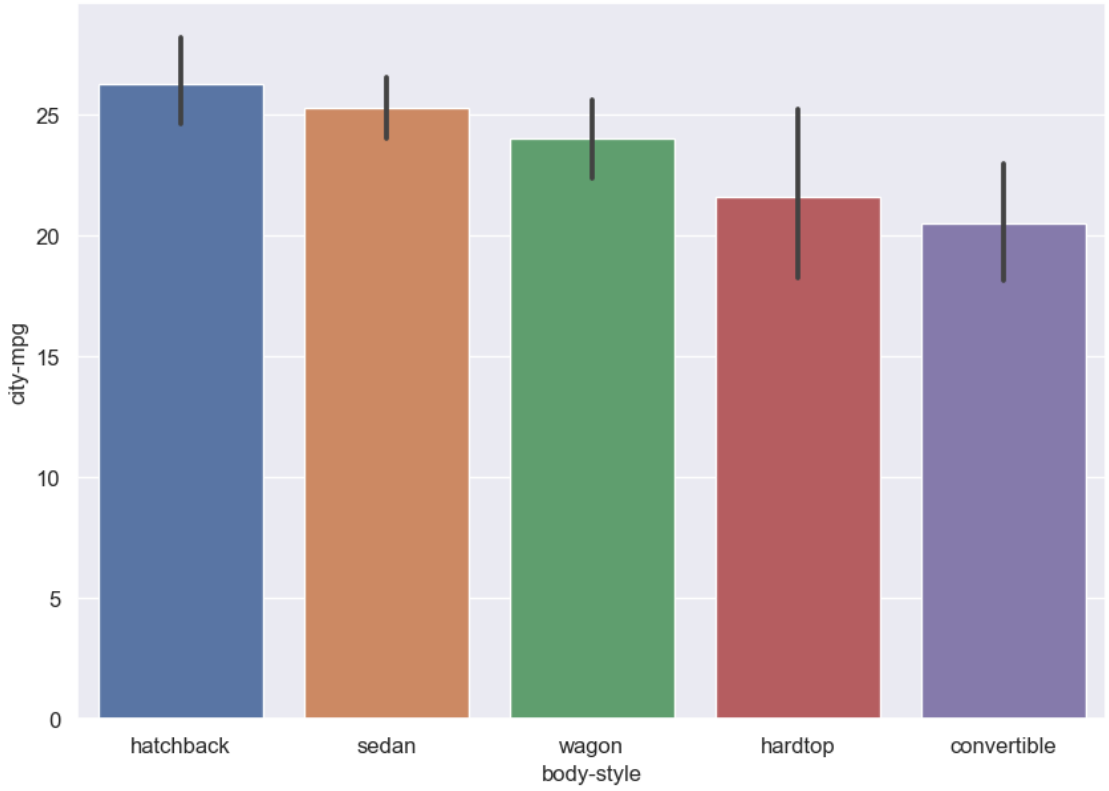
```
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



Sorted bar

[38]:
```
plt.figure(figsize=(10,7))
order = db.groupby(['body-style']).mean().sort_values('city-mpg' , ascending =␣
  ↪False).index
sns.barplot(db['body-style'], db['city-mpg'],order=order)
plt.show()
```

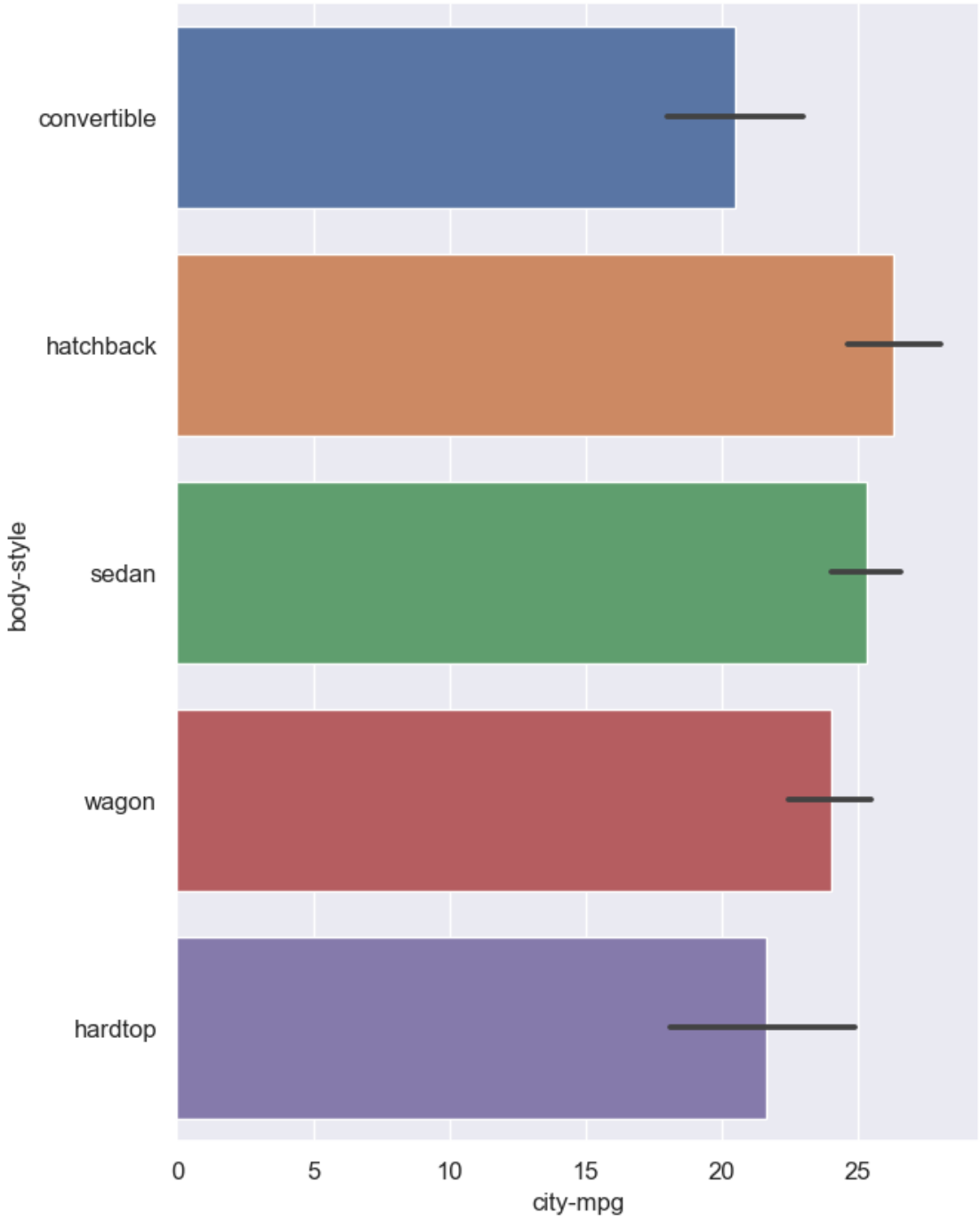
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



Horizontal bar plot

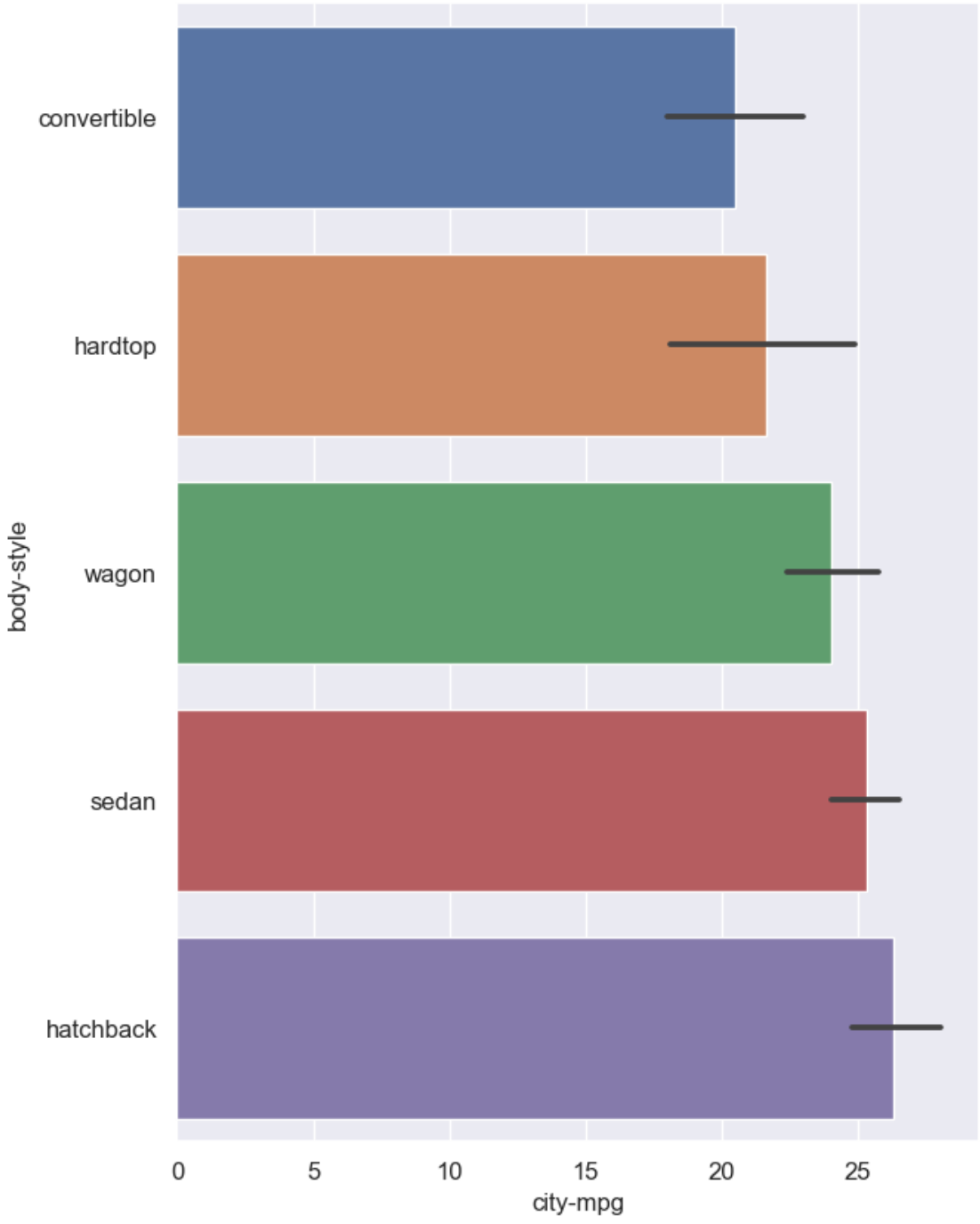
[39]:
```
plt.figure(figsize=(7,10))
sns.barplot(db['city-mpg'], db['body-style'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[40]:
```
plt.figure(figsize=(7,10))
order = db.groupby(['body-style']).mean().sort_values('city-mpg' , ascending =␣
↪True).index
sns.barplot(db['city-mpg'], db['body-style'],order=order)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



The **countplot** function in the seaborn library is used to create a bar plot that shows the counts of observations in each categorical bin using bars. This is a useful function for visualizing the distribution of categorical variables in a dataset.

The countplot function takes in one or more categorical variables as input and displays the **count of each category** in the dataset as a bar plot. It can be used to quickly summarize the distribution of a categorical variable or to compare the distribution of multiple categorical variables side-by-side.

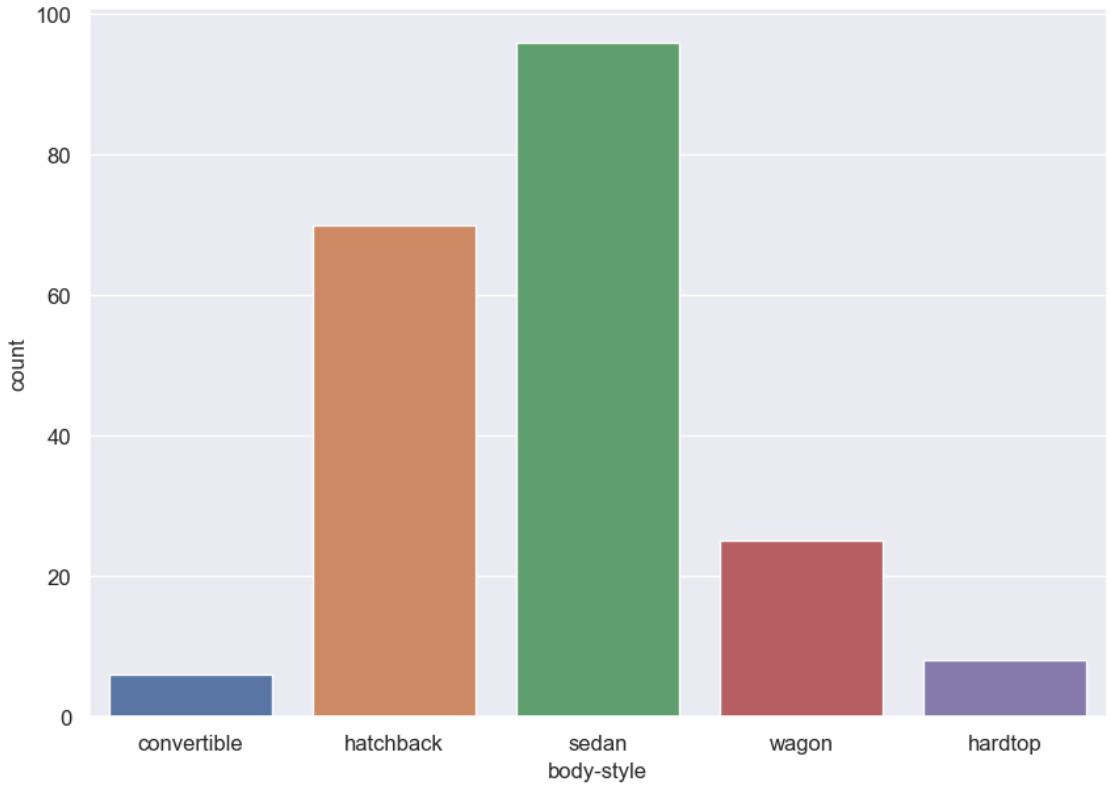
Some common use cases for the countplot function include:

- Exploring the distribution of a single categorical variable: You can use countplot to create a bar chart that shows the count of each category in the variable. This can help you understand the most common categories and identify any imbalances in the distribution of the variable.
- Comparing the distribution of multiple categorical variables: You can use countplot to create side-by-side bar charts that show the count of each category in multiple categorical variables. This can help you identify any patterns or differences between the variables.
- Visualizing the relationship between categorical and numerical variables: You can use countplot to create a bar chart that shows the count of each category in a categorical variable, with the height of the bars representing a numerical variable. This can help you visualize the relationship between the categorical and numerical variables.

Overall, the countplot function is a powerful and versatile tool for exploring the distribution of categorical variables in a dataset.

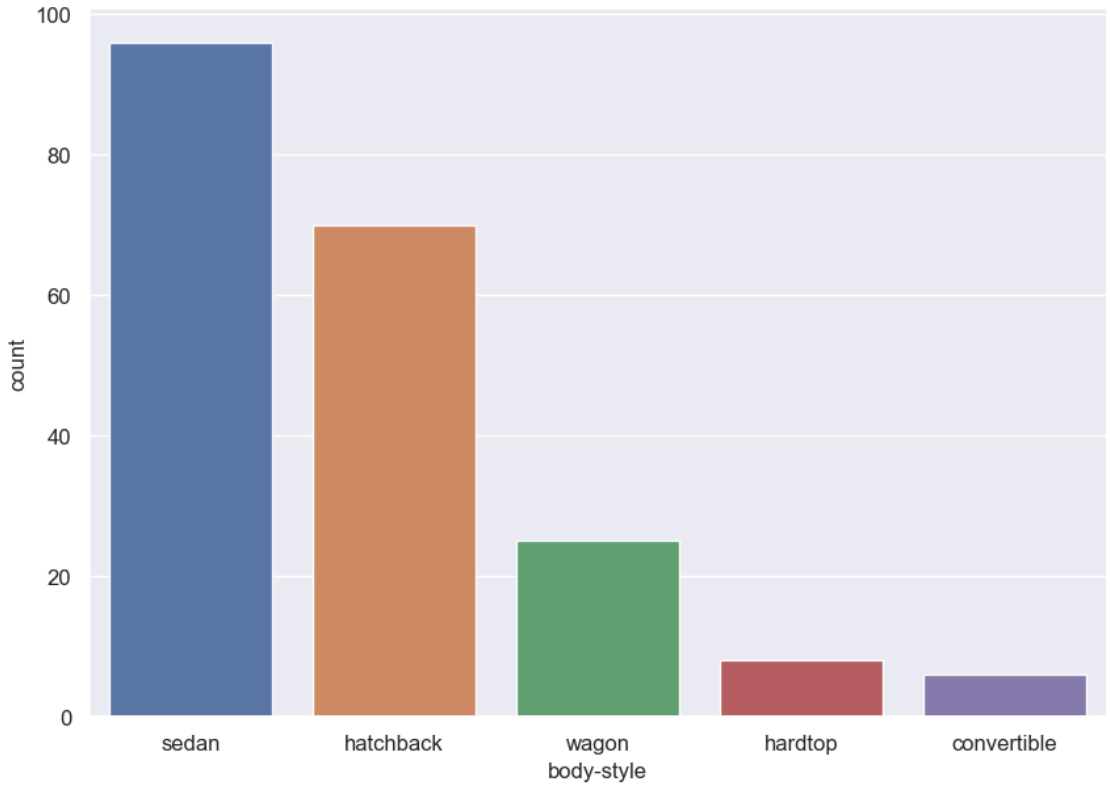
[41]:
```
plt.figure(figsize=(10,7))
sns.countplot(db['body-style'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[42]:
```
plt.figure(figsize=(10,7))
sns.countplot(db['body-style'], order=db['body-style'].value_counts().index)
plt.show()
```

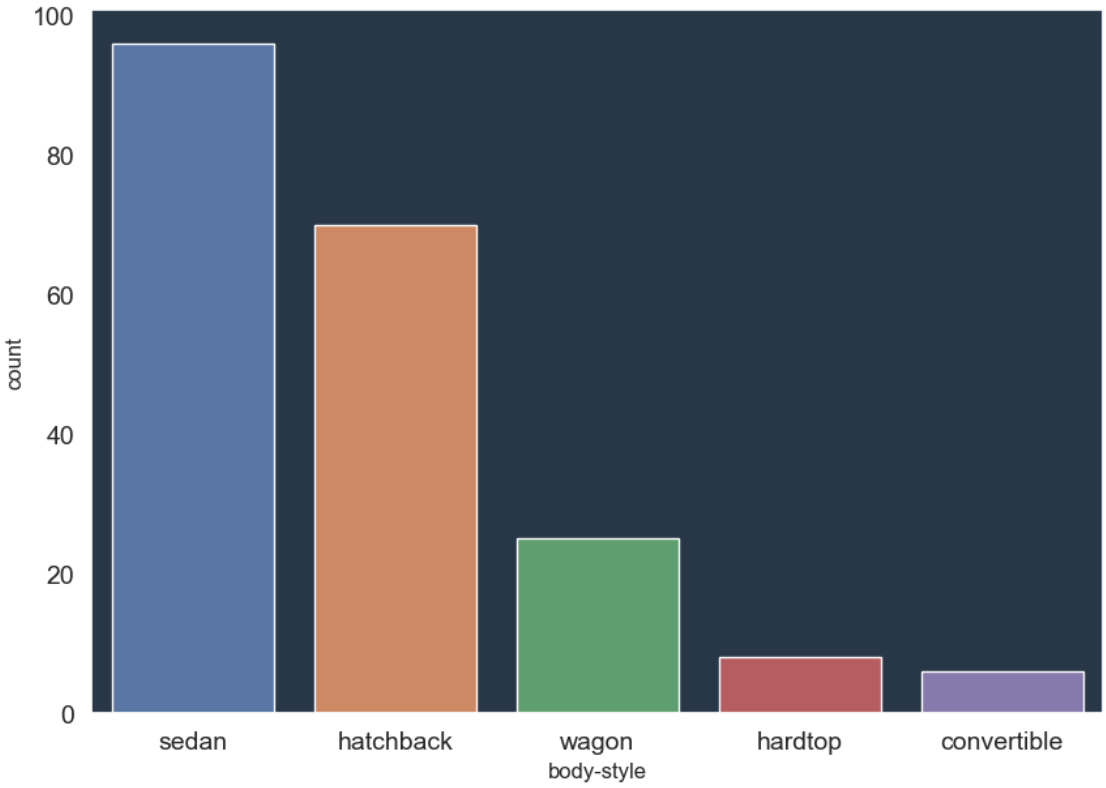
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



"axes.grid":False,'xtick.labelsize':14,'ytick.labelsize':14}): sets the seaborn style settings for the plot. In this case, it sets the facecolor of the plot to a dark blue color (#283747), turns off the grid, and sets the font size of the x and y tick labels to 14.

[43]:
```
plt.figure(figsize=(10,7))
sns.set(rc={"axes.facecolor":"#283747", "axes.grid":False,'xtick.labelsize':
 ↪14,'ytick.labelsize':14})
sns.countplot(db['body-style'], order=db['body-style'].value_counts().index)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



- The first line, **mpl.rcParams.update(mpl.rcParamsDefault)**, resets the matplotlib rcParams dictionary to its default values. This ensures that any customizations made to the plot style in previous code cells are reset, and that the plot is displayed with the default style.
- The second line,**%matplotlib inline**, is a magic command that sets the backend of matplotlib to display plots inline in the notebook or console output. This means that any plot created with matplotlib or seaborn will be displayed directly below the code cell that created it, without opening a separate window or generating a file.

Together, these two lines ensure that the plot is displayed with the default style and is shown inline in the Jupyter notebook or IPython console output.

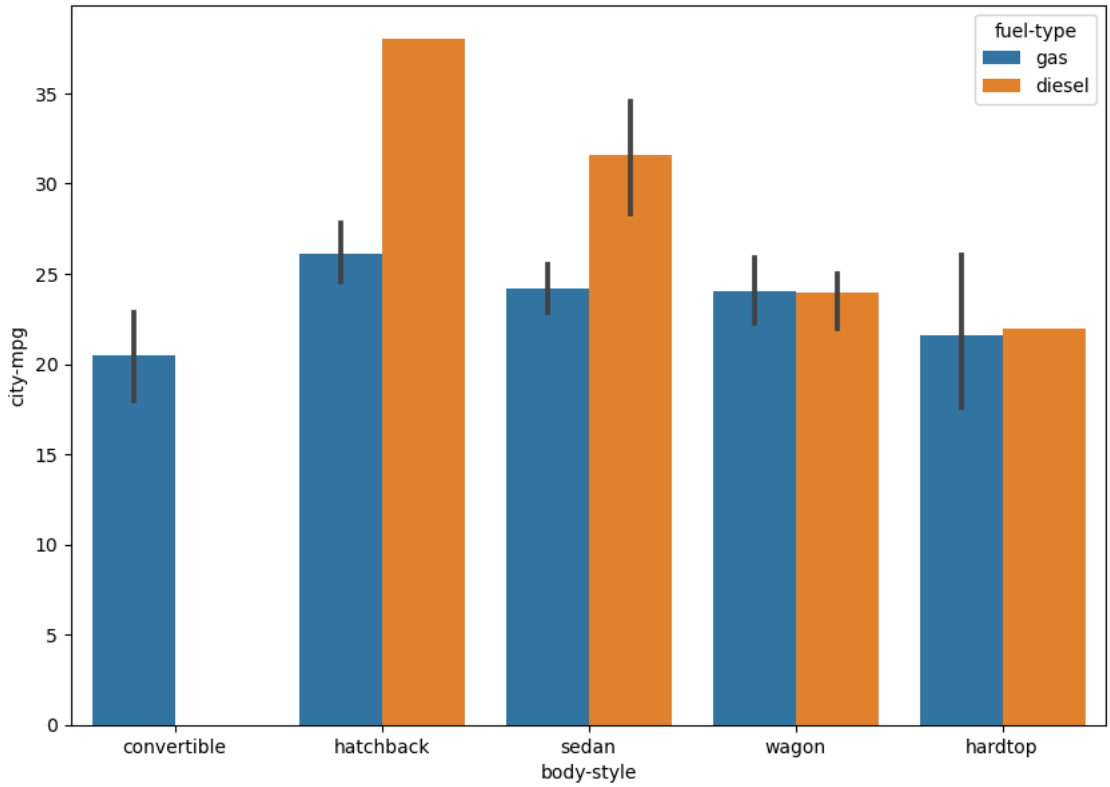
[44]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

Adding Hue

[45]:
```
plt.figure(figsize=(10,7))
sns.barplot(db['body-style'], db['city-mpg'], hue=db['fuel-type'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
```

```
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



1.6 Cat plot

catplot is a function in the seaborn library in Python that is used to create categorical plots. It is a high-level interface for creating different types of categorical plots, including but not limited to: strip plots, swarm plots, box plots, violin plots, and bar plots.

catplot can be used to visualize the relationship between one or more categorical variables and one or more numerical variables in a dataset. The kind parameter of catplot specifies the type of categorical plot to create.

Some of the different types of categorical plots that can be created using catplot include:

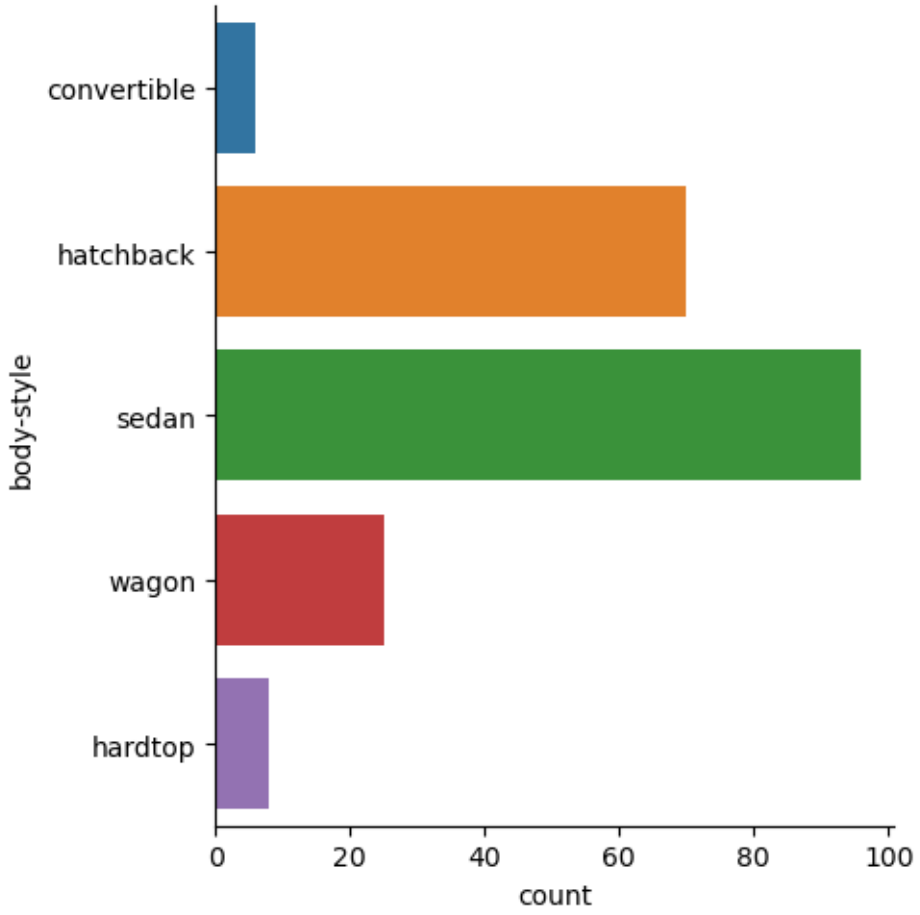
- Strip plot: A scatter plot that shows the distribution of points for a categorical variable along a numeric axis.
- Swarm plot: Similar to a strip plot, but points are adjusted so they don't overlap.
- Box plot: A plot that shows the distribution of a numerical variable within different categories using boxes.
- Violin plot: Similar to a box plot, but instead of showing a box, it shows a kernel density estimate of the distribution within each category.

- Bar plot: A plot that shows the values of a numerical variable for different categories using bars.

catplot is a very versatile function and can be used to create a wide variety of categorical plots. It is particularly useful when exploring the relationship between categorical and numerical variables in a dataset.

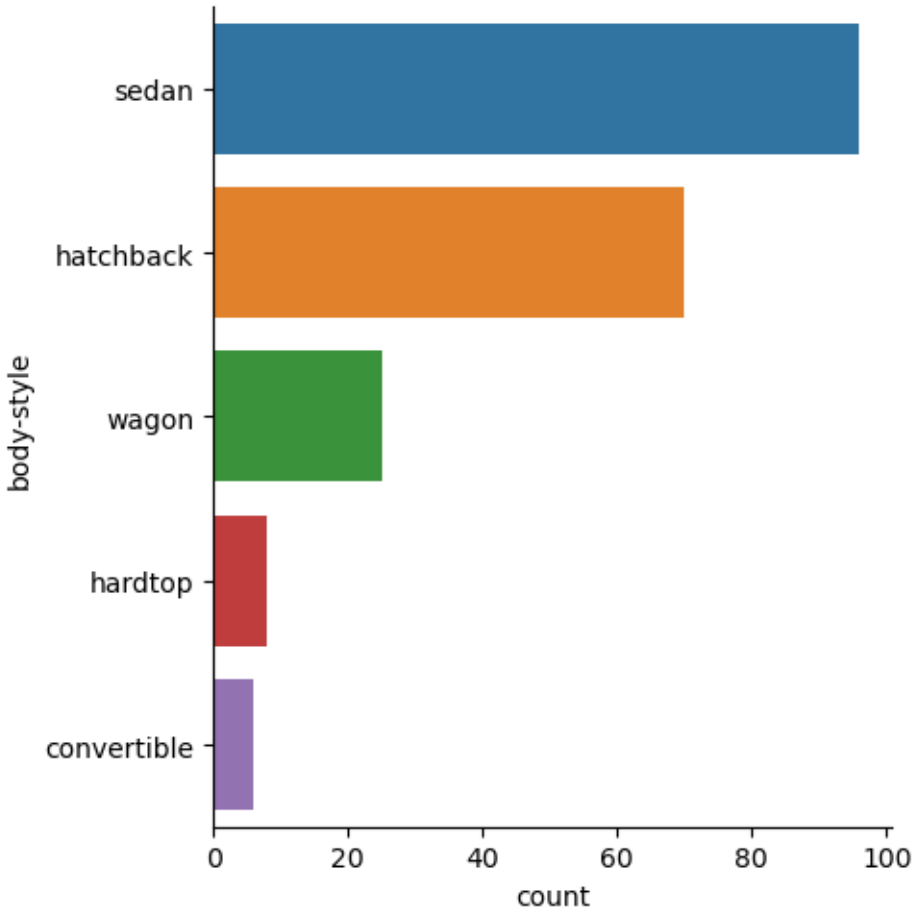
[46]:
```
plt.figure(figsize=(10,10))
sns.catplot(y ='body-style', kind = "count",data = db)
plt.show()
```

```
<Figure size 1000x1000 with 0 Axes>
```



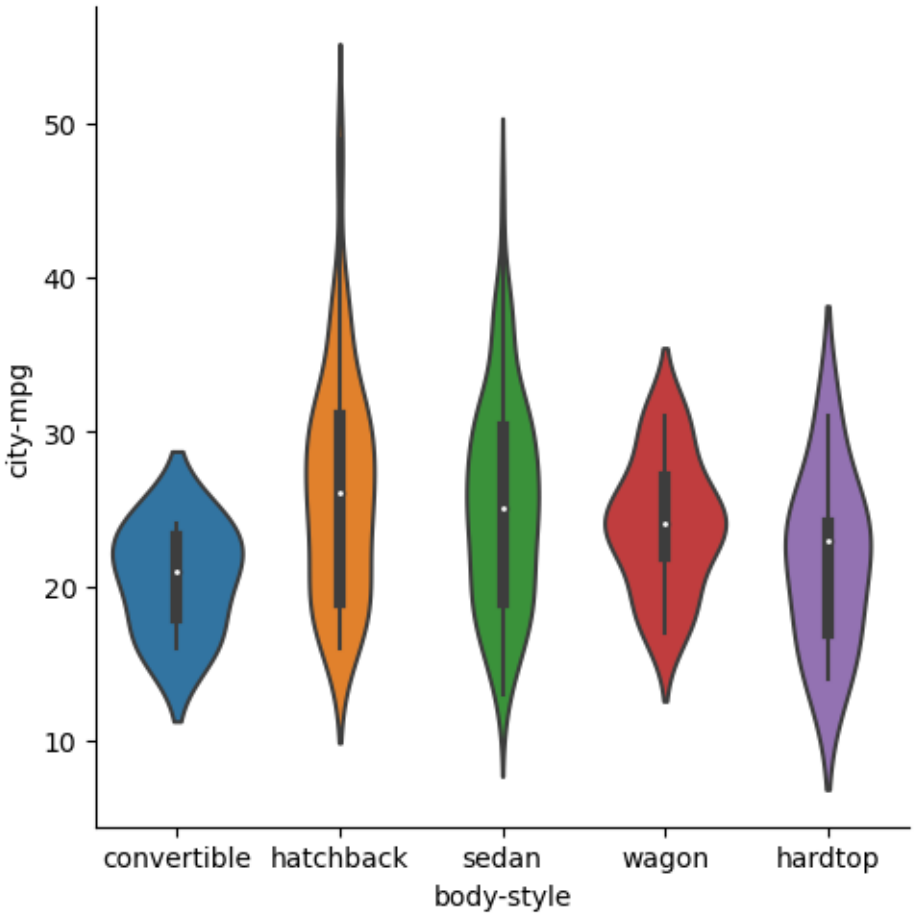
[47]:
```
plt.figure(figsize=(7,10))
sns.catplot(y ='body-style', kind = "count",data = db,order=db['body-style'].
  ↪value_counts().index)
plt.show()
```

```
<Figure size 700x1000 with 0 Axes>
```



[48]:
```
plt.figure(figsize=(12,10))
sns.catplot(x="body-style" , y = "city-mpg" ,kind="violin" ,data=db)
plt.show()
```

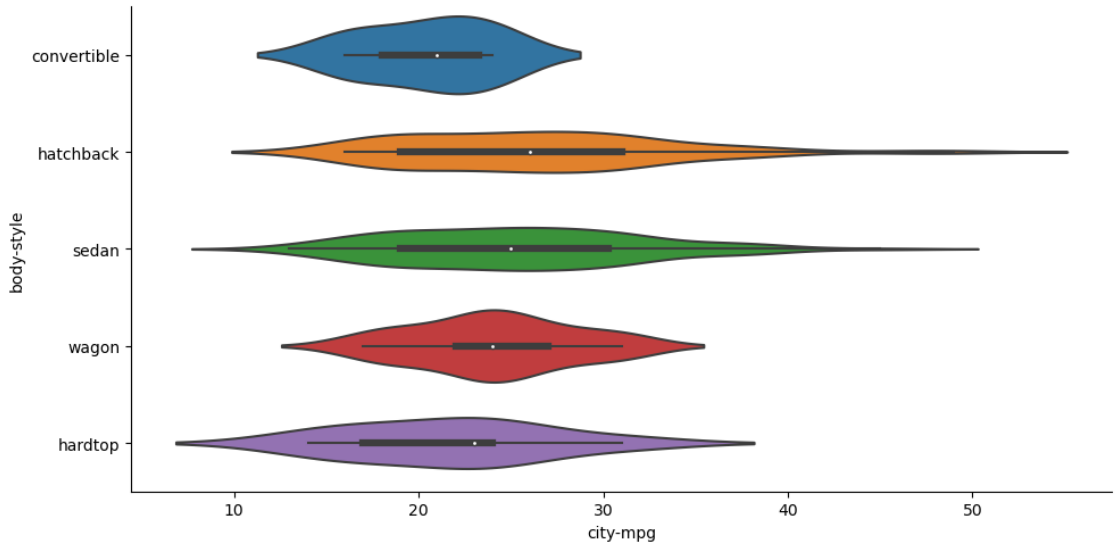
```
<Figure size 1200x1000 with 0 Axes>
```



aspect=2: this sets the aspect ratio of the plot to 2:1 (i.e., the plot will be twice as wide as it is tall).

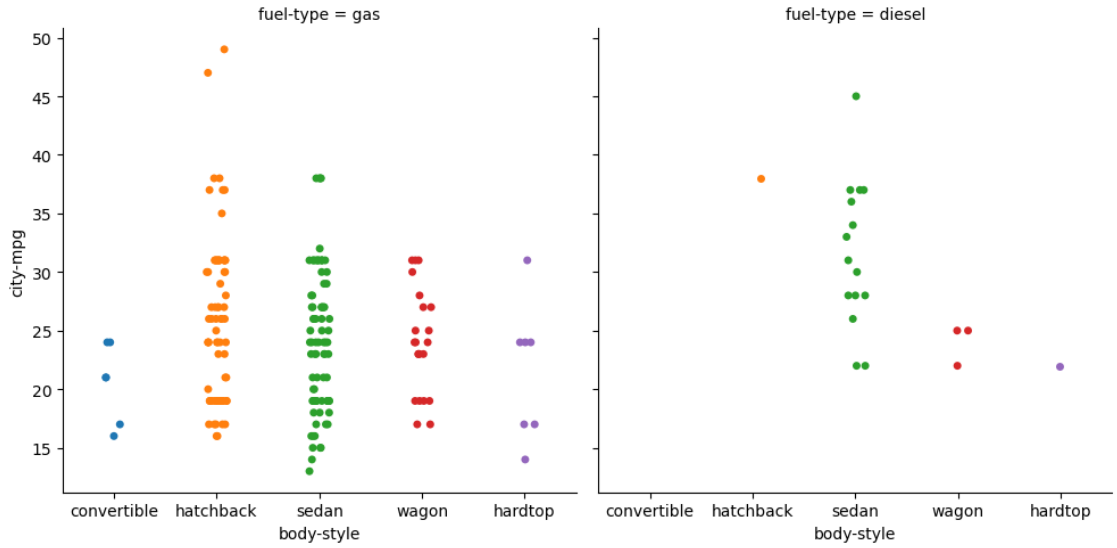
[49]:
```
plt.figure(figsize=(11,9))
sns.catplot(x = "city-mpg" , y="body-style" ,kind="violin" ,data=db ,height=5,␣
↪aspect=2)
plt.show()
```

```
<Figure size 1100x900 with 0 Axes>
```



[50]:
```
plt.figure(figsize=(10,10))
sns.catplot(x='body-style' , y='city-mpg' , data=db , col="fuel-type")
plt.show()
```

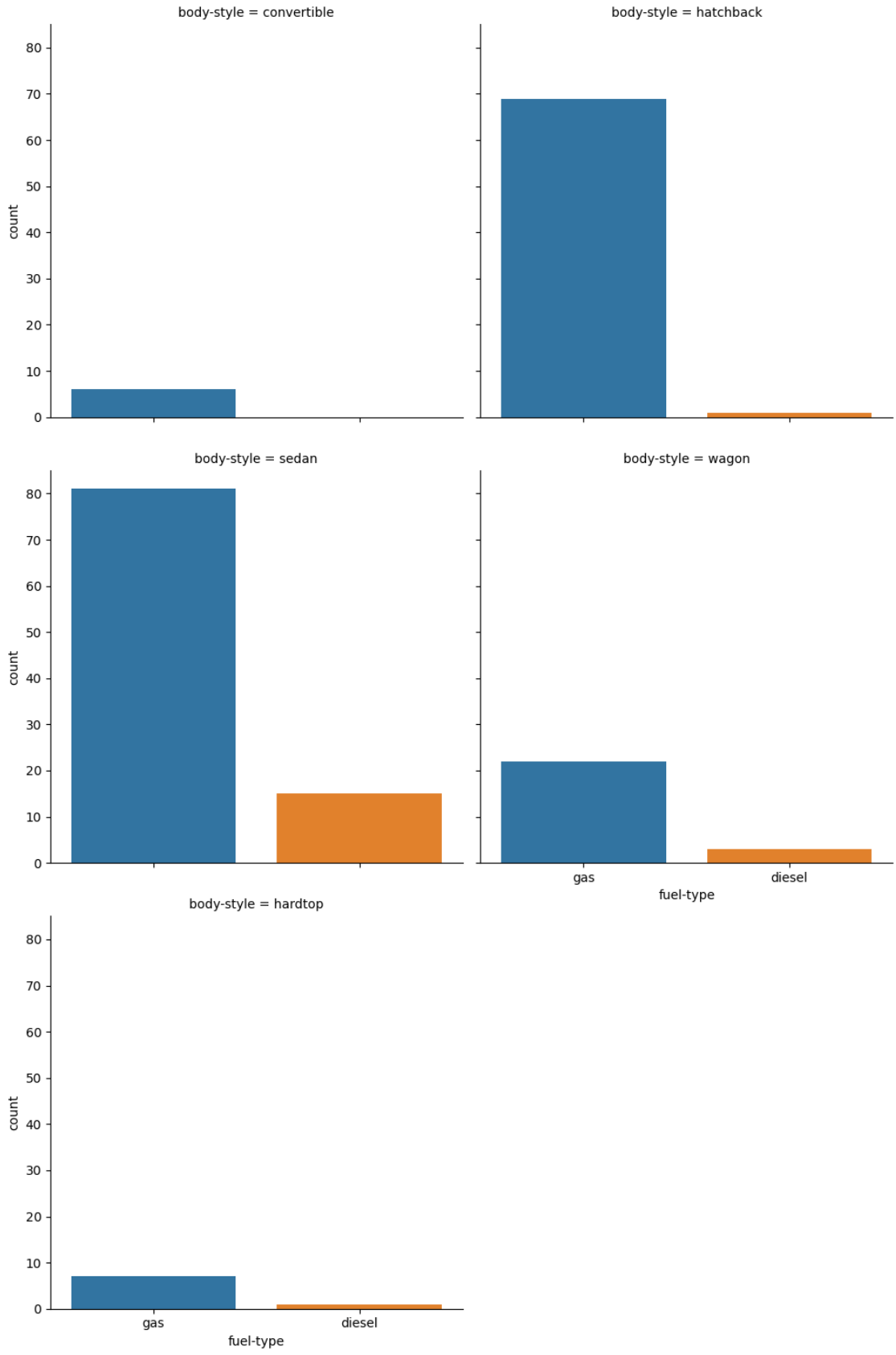
```
<Figure size 1000x1000 with 0 Axes>
```



[51]:
```
plt.figure(figsize=(20,10))
sns.catplot("fuel-type",␣
  ↪col="body-style",col_wrap=2,data=db,kind="count",height=5, aspect=1)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

```
<Figure size 2000x1000 with 0 Axes>
```

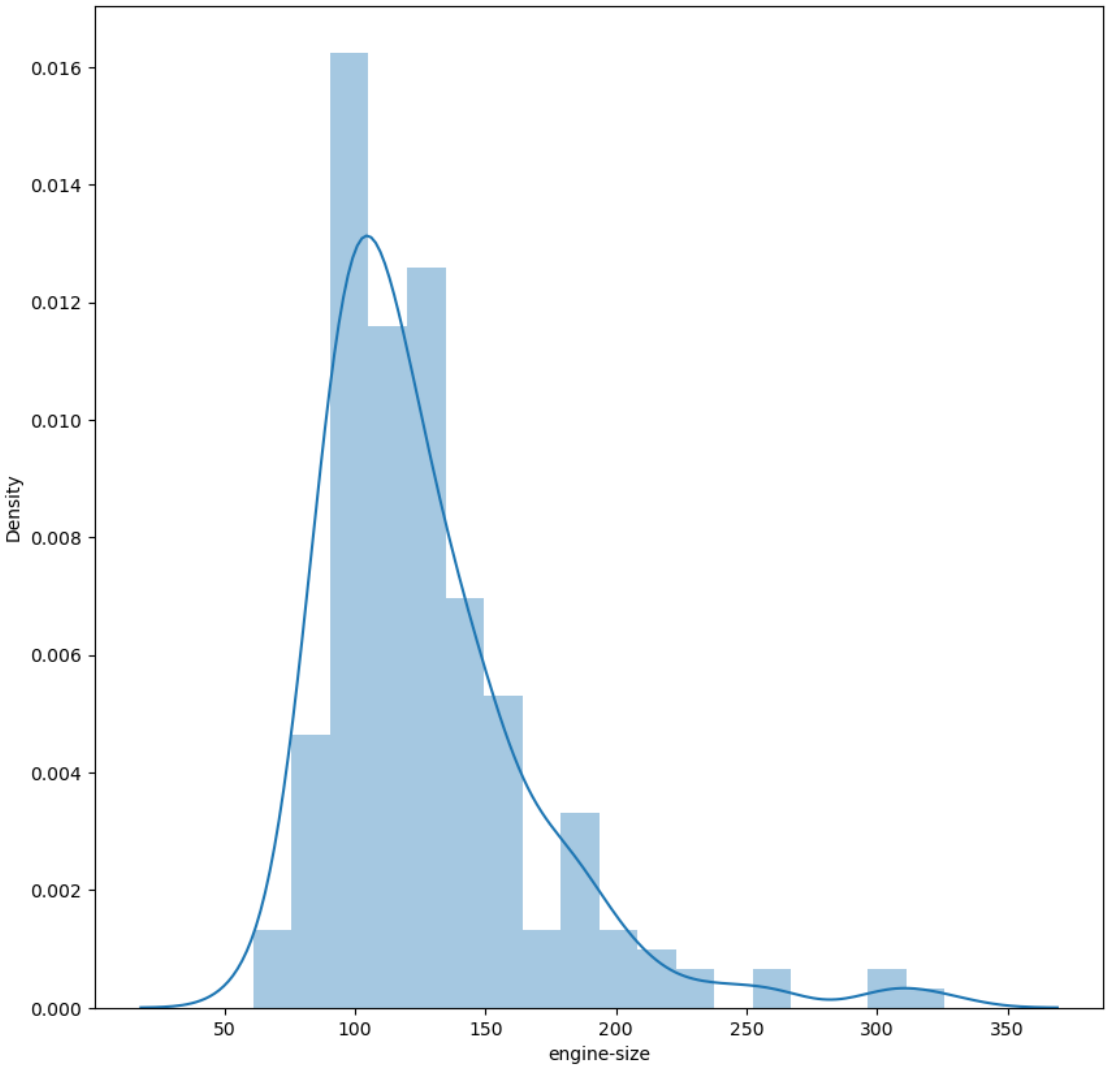

body-style = convertible
body-style = hatchback
body-style = sedan
body-style = wagon
body-style = hardtop
count
0
10
20
30
40
50
60
70
80
gas
diesel
fuel-type

1.7 Dist Plot

A **distplot** is a seaborn library function used to visualize the distribution of a dataset. It combines a histogram with a kernel density estimate **(KDE)** plot, which is a smoothed version of the histogram. The seaborn library is built on top of the Matplotlib library and provides a higher-level interface to create more attractive and informative statistical graphics.

[52]:
```
plt.figure(figsize=(10,10))
sns.distplot(db['engine-size'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



1.7.1 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()**:

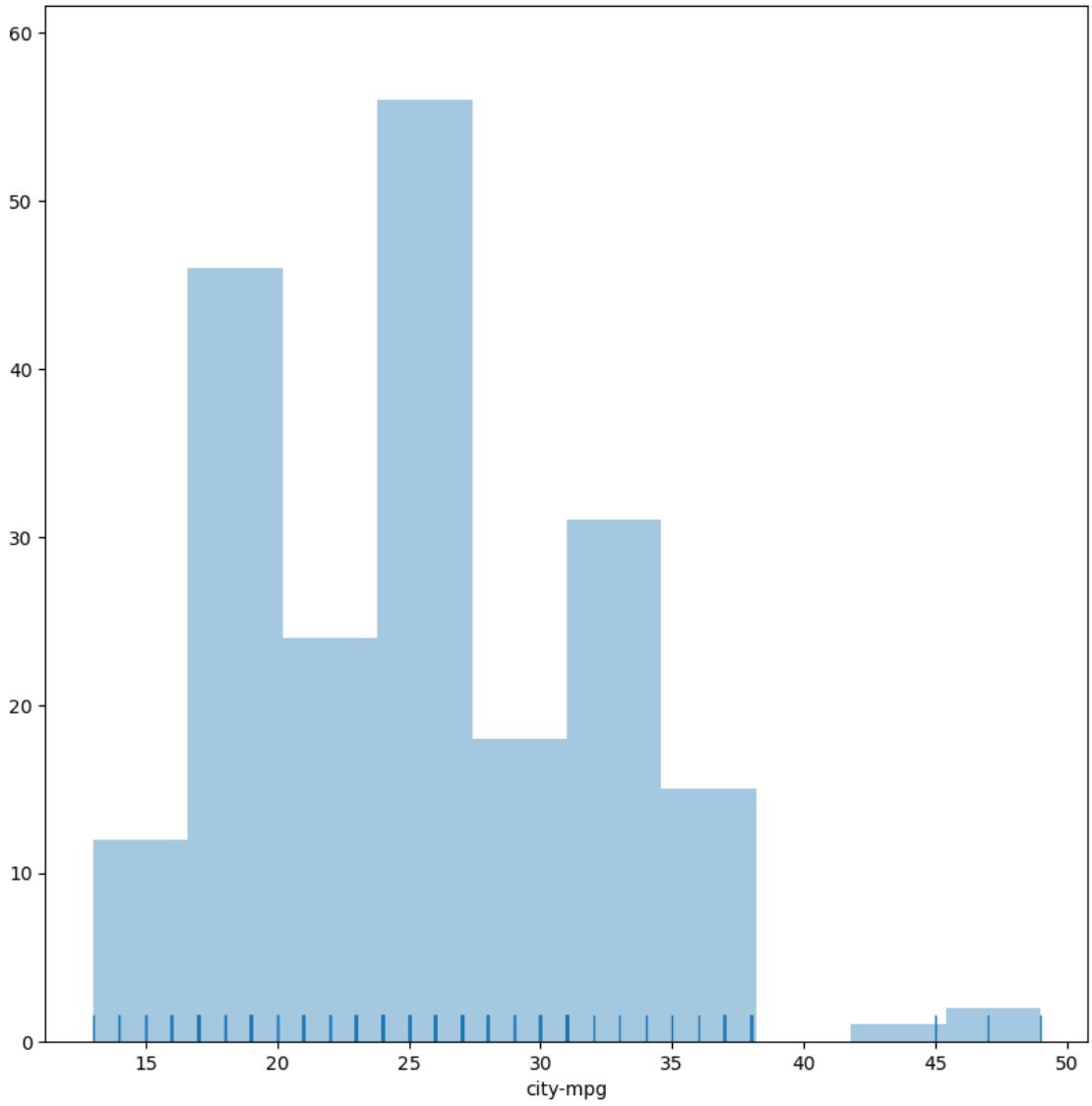
A **rug plot** is a type of data visualization technique used to display the distribution of data along a single axis. It consists of small vertical lines or **"ticks"** drawn along a single axis, usually the x-axis, at the location of each data point in the dataset.

A rug plot is often used in combination with a histogram or a **kernel density estimate (KDE) plot** to provide additional information about the distribution of data. By adding the rug plot,

you can see where each data point falls along the axis and get a better sense of the underlying distribution.

[53]:
```
plt.figure(figsize=(10,10))
sns.distplot(db['city-mpg'], kde=False, rug=True)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2103:
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```

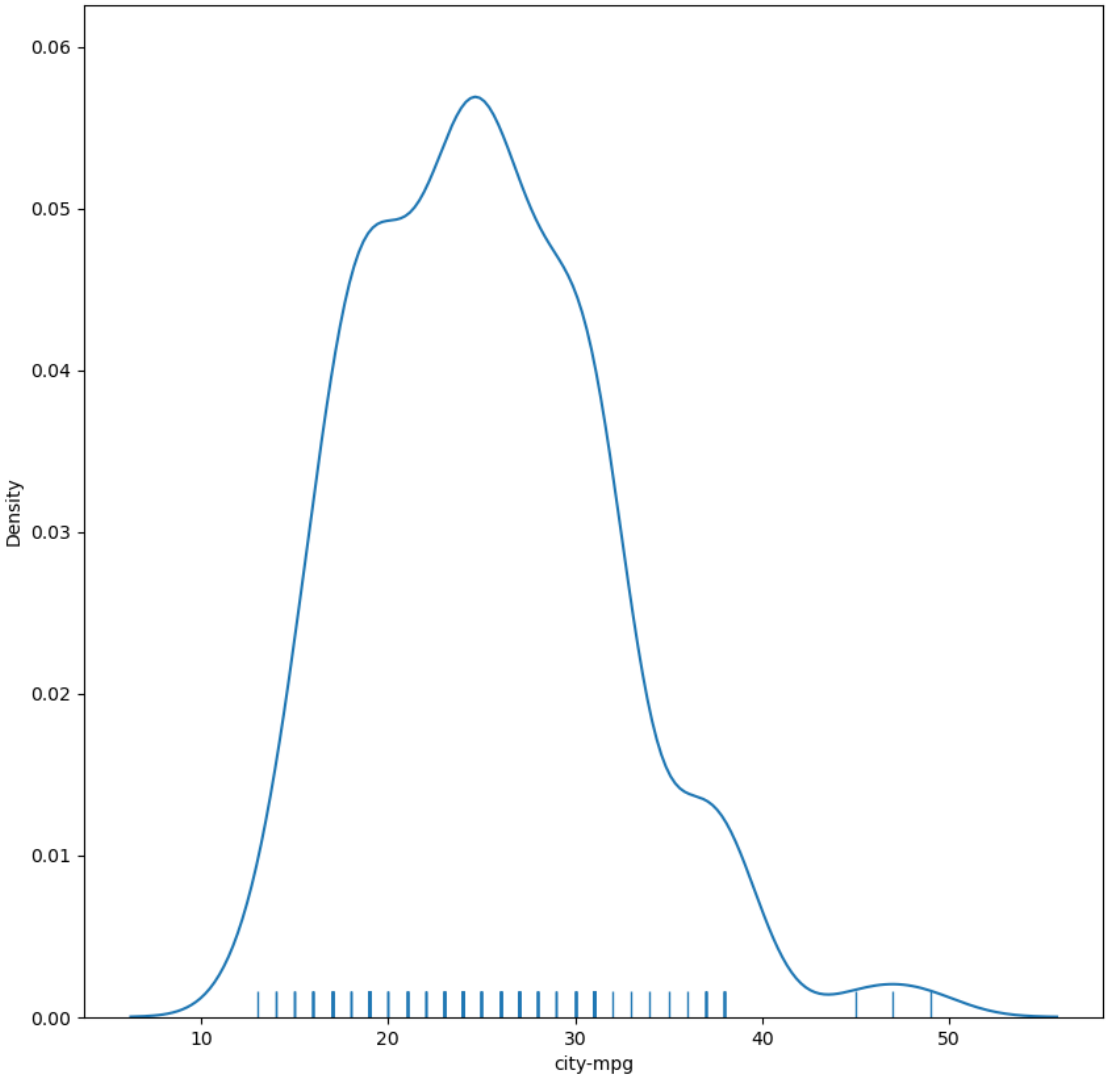


Histogram with **rugplot** and **kde**

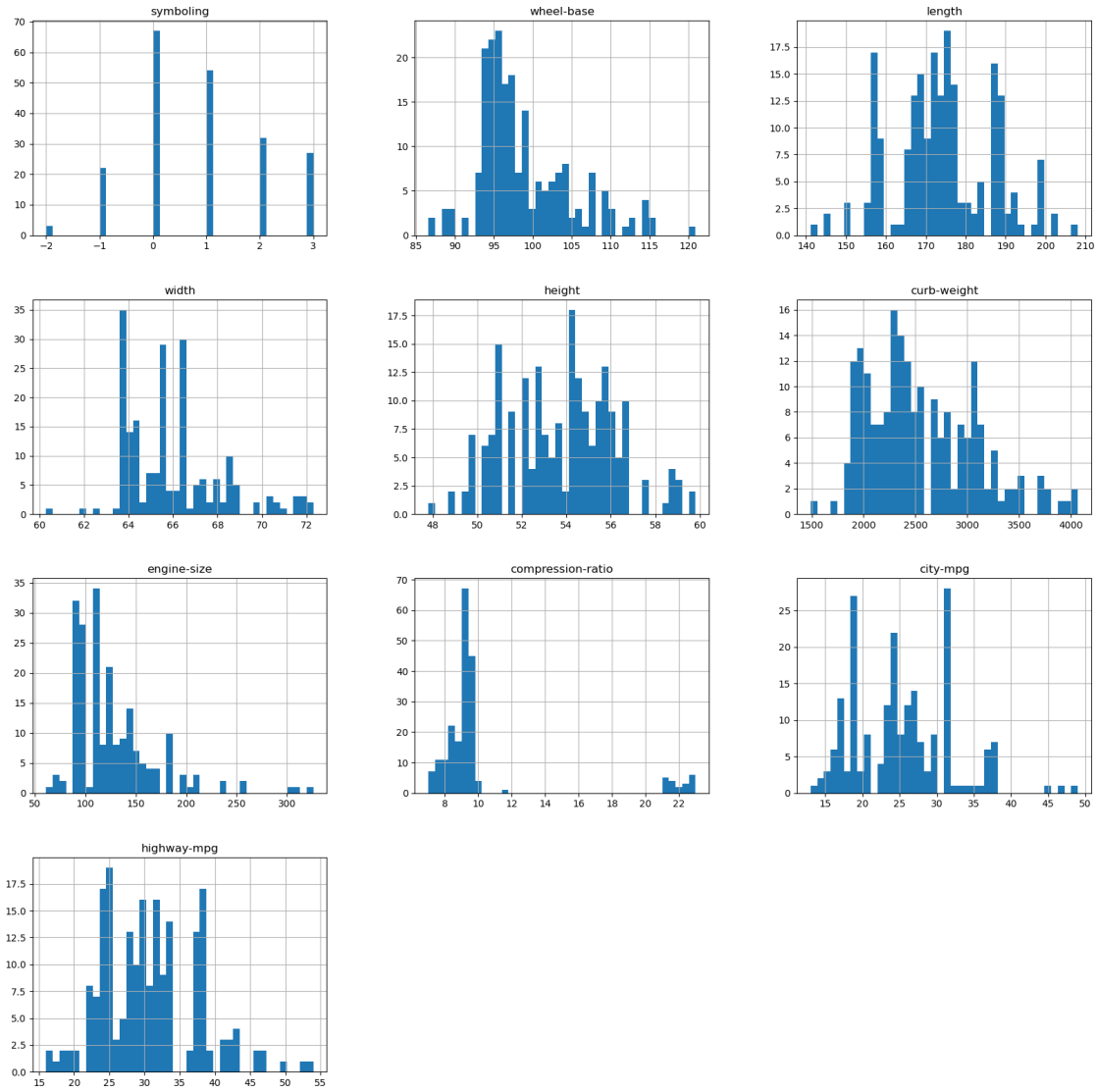
[54]:
```
plt.figure(figsize=(10,10))
sns.distplot(db['city-mpg'], kde=True, rug=True,hist=False)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `kdeplot` (an axes-level function for
kernel density plots).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2103:
```

```
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```



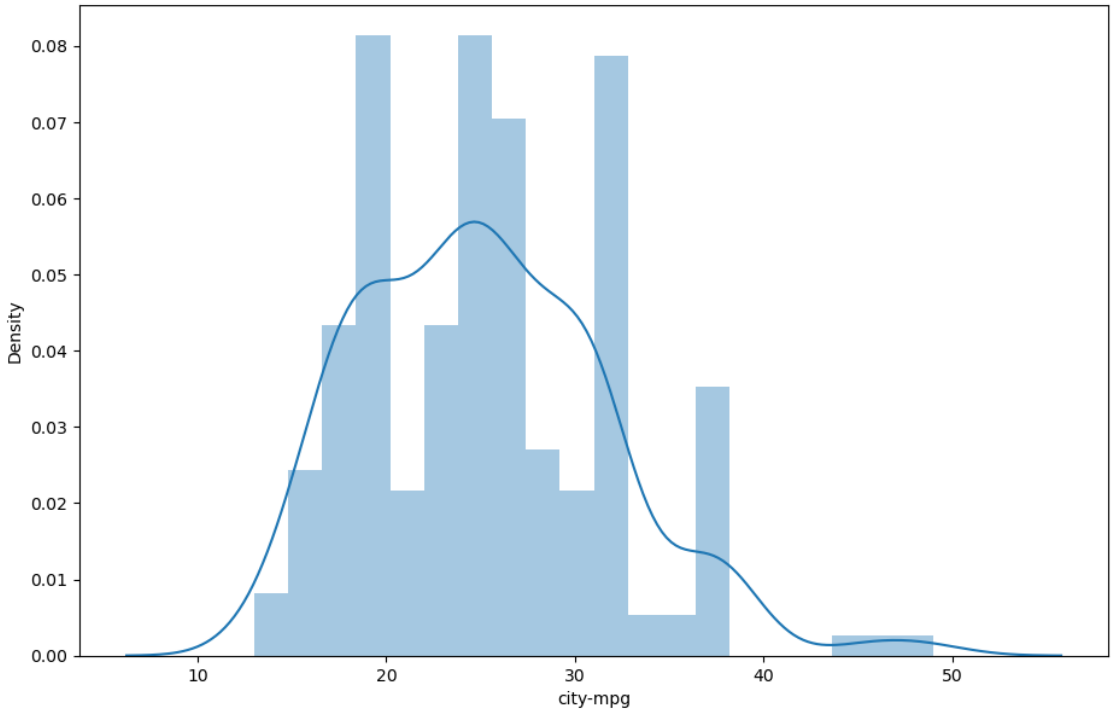
[55]:
```
db.hist(bins=40 , figsize=(20,20))
plt.show()
```



- In the context of creating a histogram using a Python data visualization library like Seaborn or Matplotlib, **"bins"** refer to the number of intervals or **"buckets"** into which the data is divided.
- When plotting a histogram, the range of the data is divided into equal-sized bins or intervals. The height of each bar in the histogram represents the number of data points that fall within that interval.
- In the code you provided, **sns.distplot(db['city-mpg'], bins=20)**, the parameter **bins=20** specifies that the range of the data will be divided into 20 equal-sized bins. This means that the histogram will have 20 bars, each representing the number of "city-mpg" values that fall within a particular range of values.
- The choice of the number of bins can impact the appearance of the histogram and how the data is interpreted. Too few bins can result in a histogram that is too general.

[56]:
```
plt.figure(figsize=(11,7))
sns.distplot(db['city-mpg'], bins=20)
plt.show()
```

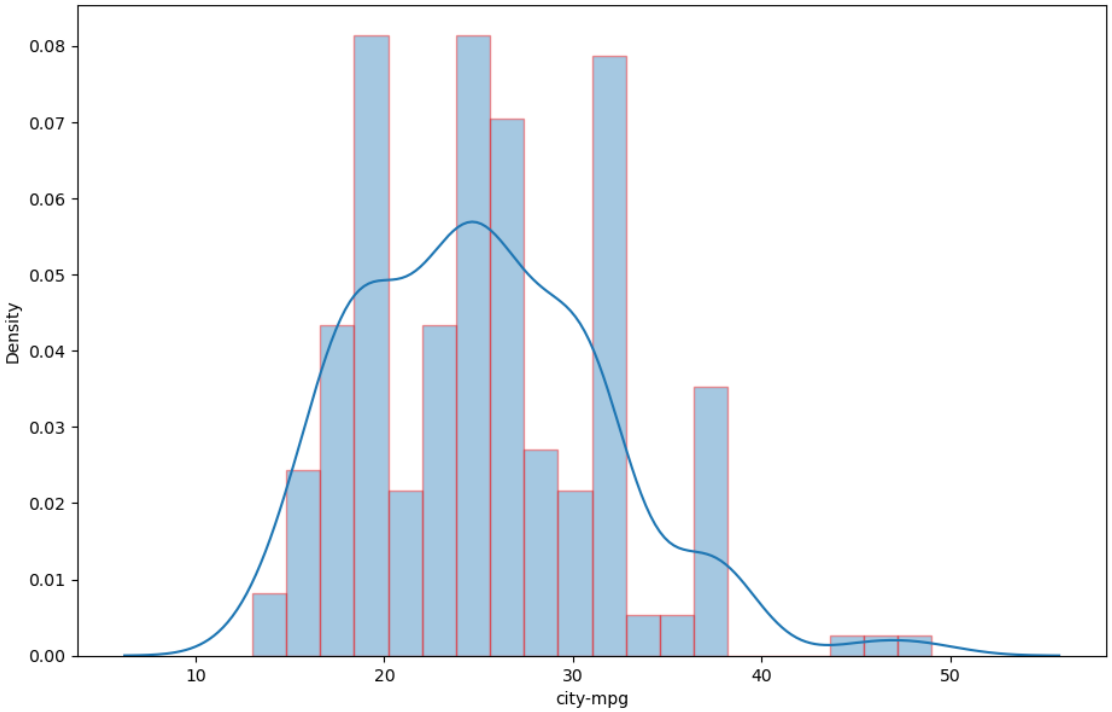
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



The **hist_kws** parameter is used to customize the properties of the histogram, such as the **edgecolor** of each bar.

[57]:
```
plt.figure(figsize=(11,7))
sns.distplot(db['city-mpg'], bins=20 ,hist_kws=dict(edgecolor = '#FF0000'))
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



1.8 KDE Plot

KDE stands for **Kernel Density Estimation**, which is a non-parametric method for estimating the probability density function of a random variable. A KDE plot, also known as a density plot, is a type of plot that visualizes the distribution of a dataset by estimating its probability density function.

A KDE plot consists of a curve that is drawn based on the data points in the dataset. The curve represents the estimated probability density function of the data, which is a smooth estimate of the underlying distribution. The area under the curve is normalized to one, which means that the total area under the curve represents the total probability of the data.

KDE plots are often used to visualize the distribution of continuous variables, such as age or income. They are similar to histograms, but they provide a smoother and continuous estimate of the distribution, without being dependent on the bin size or placement. Additionally, KDE plots can help to identify features in the data such as modes, skewness or outliers.

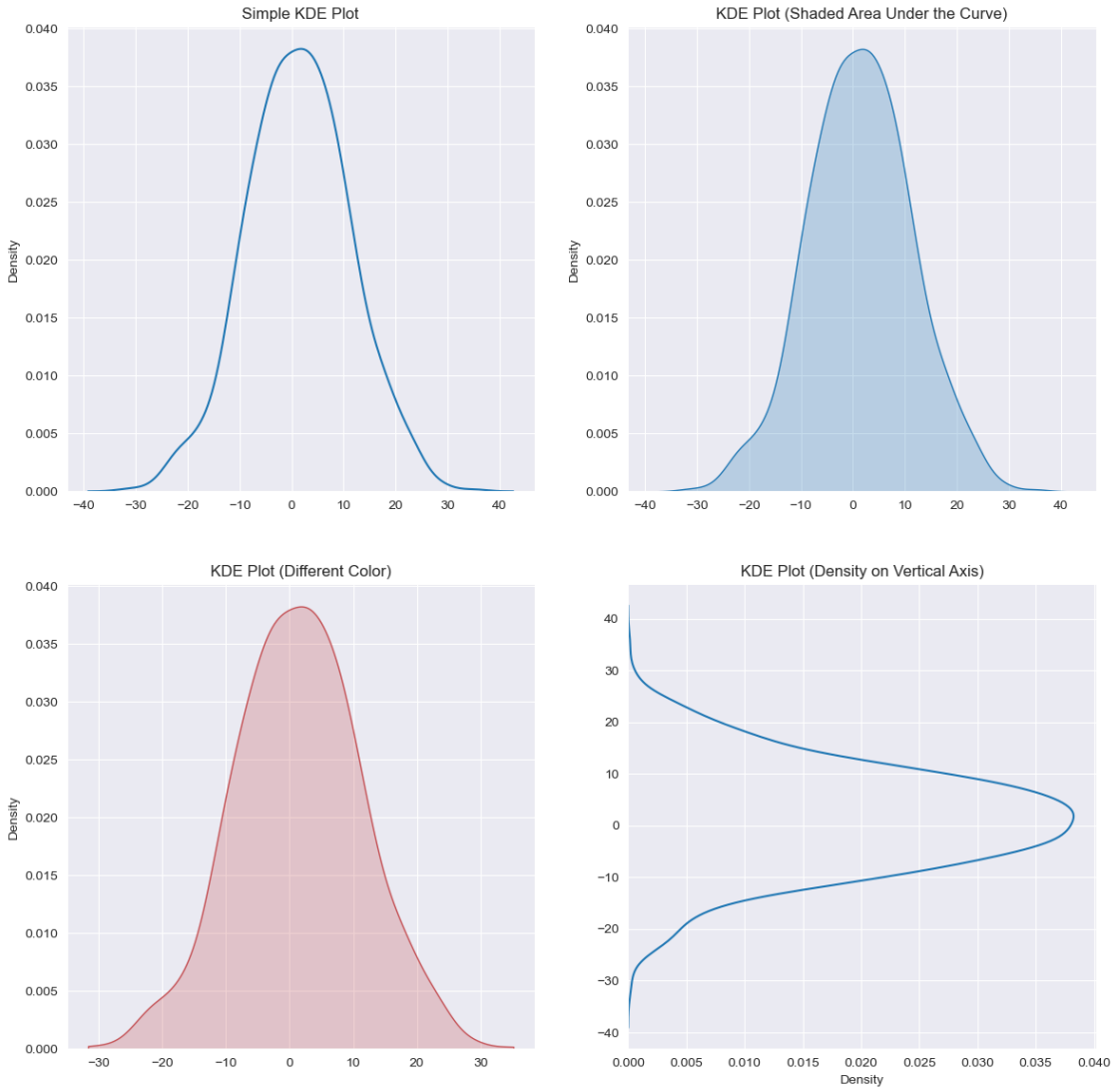
- A normal distribution with a mean of 1 and a standard deviation of 10 is generated using **np.random.normal()** function, and is assigned to the variable x.
- The figure is then divided into a 2x2 grid of subplots using the **plt.subplots()** function, and the resulting axes are stored in the variable 'axes'.
- The code then creates four different plots using **sns.kdeplot()** function to plot the kernel density estimation of the data in x.
- The **first plot** is a simple KDE plot with no shading, plotted on the first subplot using

axes[0,0].

- The **second plot** is also a KDE plot, but the shaded area under the curve is filled with color using the shade=True parameter, and is plotted on the second subplot using axes[0,1].
- The **third plot** is a KDE plot with a different color (red) and no lines outside the range of the data, set by the cut=0 parameter. This plot is plotted on the third subplot using axes[1,0].
- The **fourth plot** is also a KDE plot, but with the density on the vertical axis instead of the horizontal axis, set by the vertical=True parameter. This plot is plotted on the fourth subplot using axes[1,1].

[58]:
```
sns.set_style("darkgrid")
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (14,14))
x = np.random.normal(1,10,1000)
axes[0,0].set_title("Simple KDE Plot")
sns.kdeplot(x,ax=axes[0,0])
axes[0,1].set_title("KDE Plot (Shaded Area Under the Curve)")
sns.kdeplot(x,shade=True,ax=axes[0,1])
axes[1,0].set_title("KDE Plot (Different Color)")
sns.kdeplot(x,ax=axes[1,0],color = 'r',shade=True,cut=0)
axes[1,1].set_title("KDE Plot (Density on Vertical Axis)")
sns.kdeplot(x,vertical=True)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1689:
FutureWarning: The `vertical` parameter is deprecated and will be removed in a
future version. Assign the data to the `y` variable instead.
  warnings.warn(msg, FutureWarning)
```



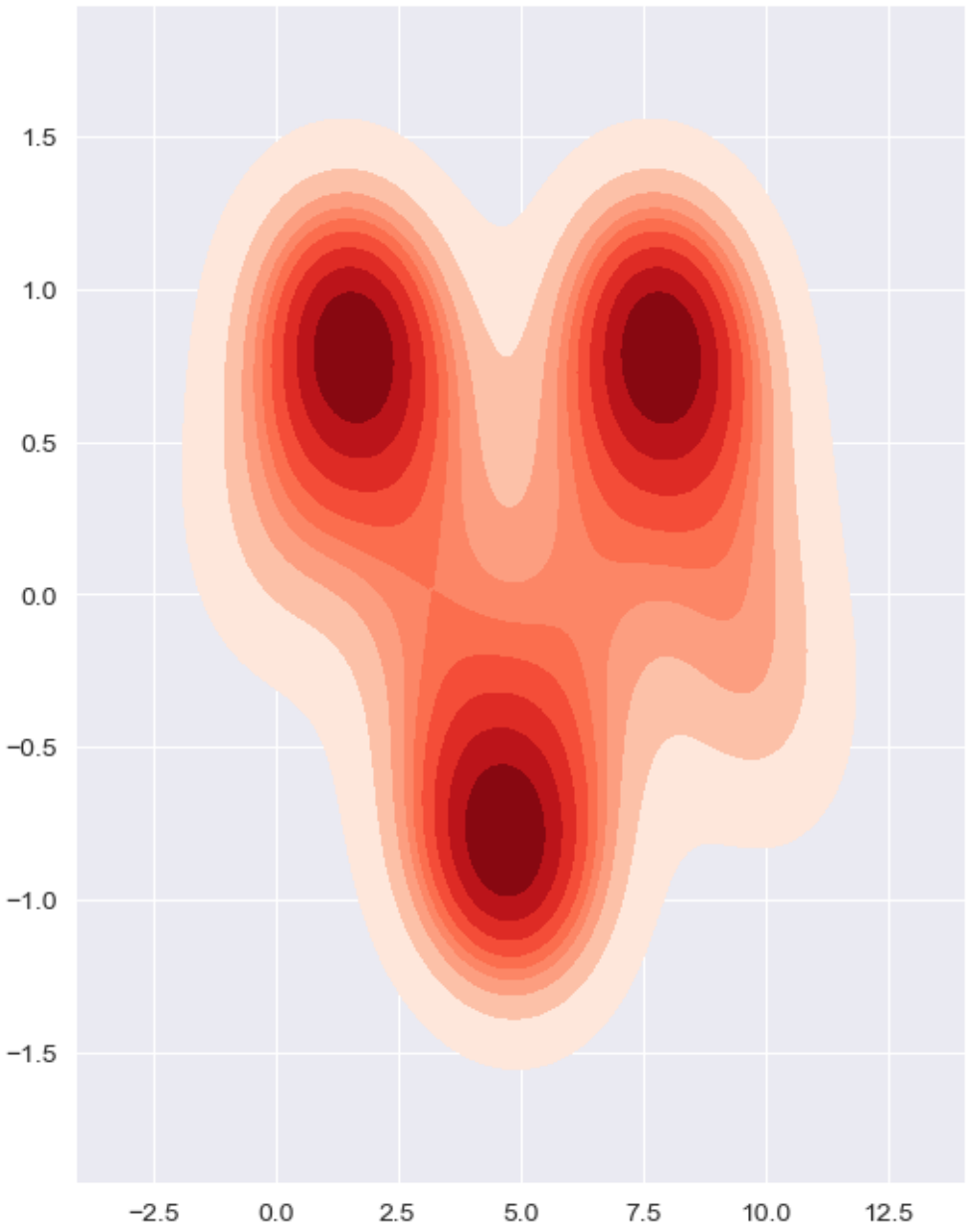
- This code creates a 2-dimensional KDE plot using Seaborn library. A figure size of 6 inches wide and 8 inches tall is set using **plt.figure(figsize=(6,8))**.
- A NumPy array is created using **np.linspace()** function with 100 points, evenly spaced between 0 and 10, and assigned to the variable x. The sine function of x is computed using **np.sin()** function and assigned to the variable y.
- The **sns.kdeplot()** function is then called with x and y as inputs to create a 2-dimensional KDE plot. The **shade=True** parameter fills the area under the curve with color, and the **cmap="Reds"** parameter sets the color scheme of the plot to shades of red. The **shade_lowest=False** parameter ensures that the lowest contour is not shaded.
- The resulting KDE plot shows the distribution of the sine function of x along the x and y axes. The color of the plot indicates the density of the distribution, with darker areas indicating higher density. Since this is a 2-dimensional KDE plot, the contours represent regions of equal

probability density

[59]:
```
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)
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1718:
UserWarning: `shade_lowest` is now deprecated in favor of `thresh`. Setting
`thresh=0.05`, but please update your code.
  warnings.warn(msg, UserWarning)
```

[59]:
```
<AxesSubplot:>
```



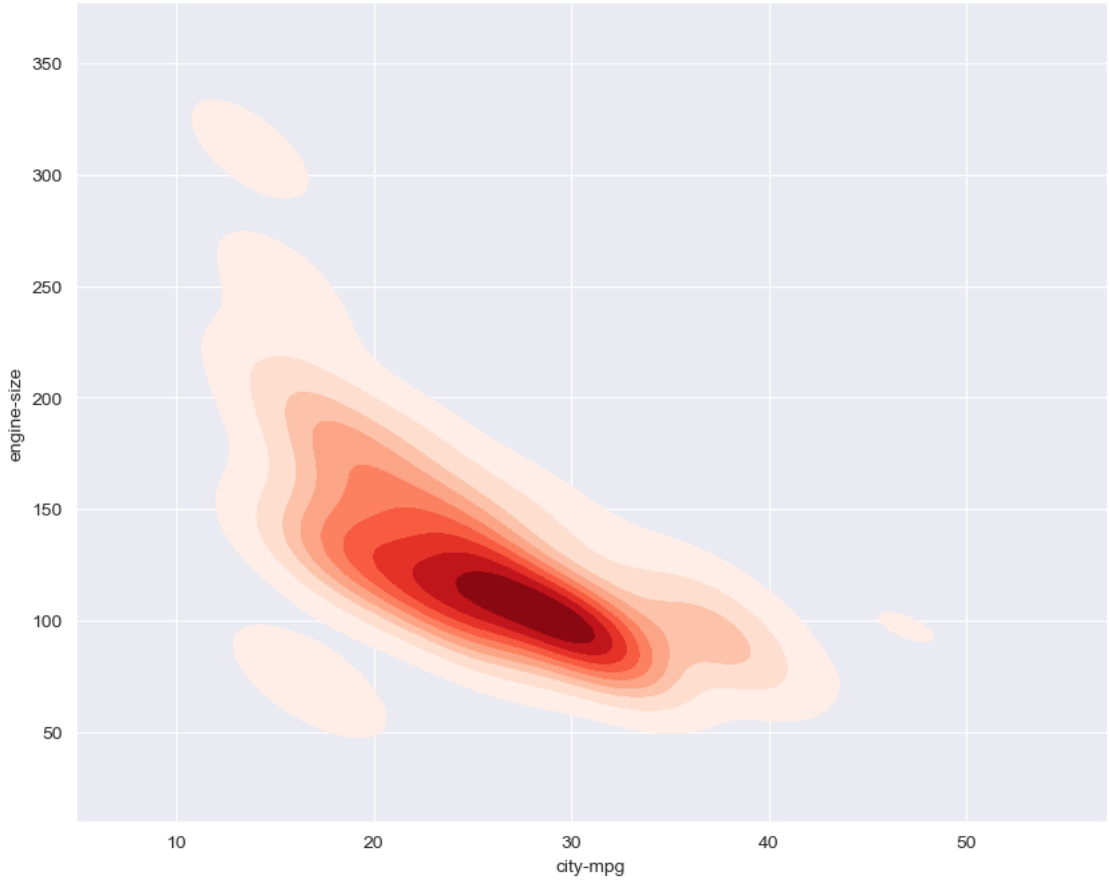
The **sns.kdeplot()** function is called with two variables **db[“city-mpg”]** and **db[“engine-size”]**, which represent the city miles per gallon and engine size of a car respectively. The **shade=True** parameter fills the area under the curve with color, and the **cmap=“Reds”** parameter sets the color scheme of the plot to shades of red. The **shade_lowest=False** parameter ensures that the lowest contour is not shaded.

The resulting KDE plot shows the **joint distribution** of the two variables along the x and y axes. The color of the plot indicates the density of the distribution, with darker areas indicating higher density. Since this is a 2-dimensional KDE plot, the contours represent regions of equal probability density. The plot allows us to see if there is any relationship between the two variables, such as if

there is a correlation between the size of an engine and the fuel efficiency of a car.

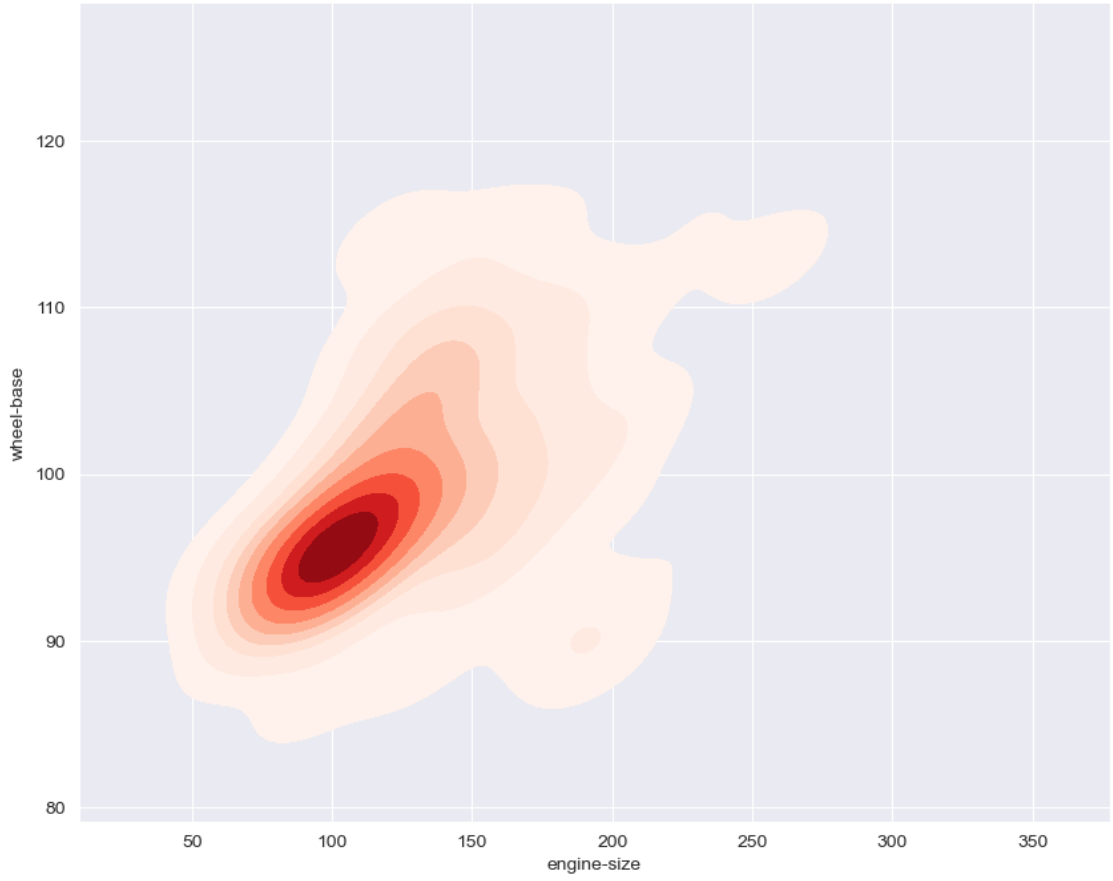
[60]:
```
plt.figure(figsize=(10,8))
sns.kdeplot(db["city-mpg"],db["engine-size"],shade=True,cmap="Reds",␣
 ↪shade_lowest=False)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1718:
UserWarning: `shade_lowest` is now deprecated in favor of `thresh`. Setting
`thresh=0.05`, but please update your code.
  warnings.warn(msg, UserWarning)
```



[61]:
```
plt.figure(figsize=(10,8))
sns.kdeplot(db["engine-size"],db["wheel-base"],cmap="Reds", shade=True,␣
 ↪shade_lowest=False)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1718:
UserWarning: `shade_lowest` is now deprecated in favor of `thresh`. Setting
`thresh=0.05`, but please update your code.
  warnings.warn(msg, UserWarning)
```

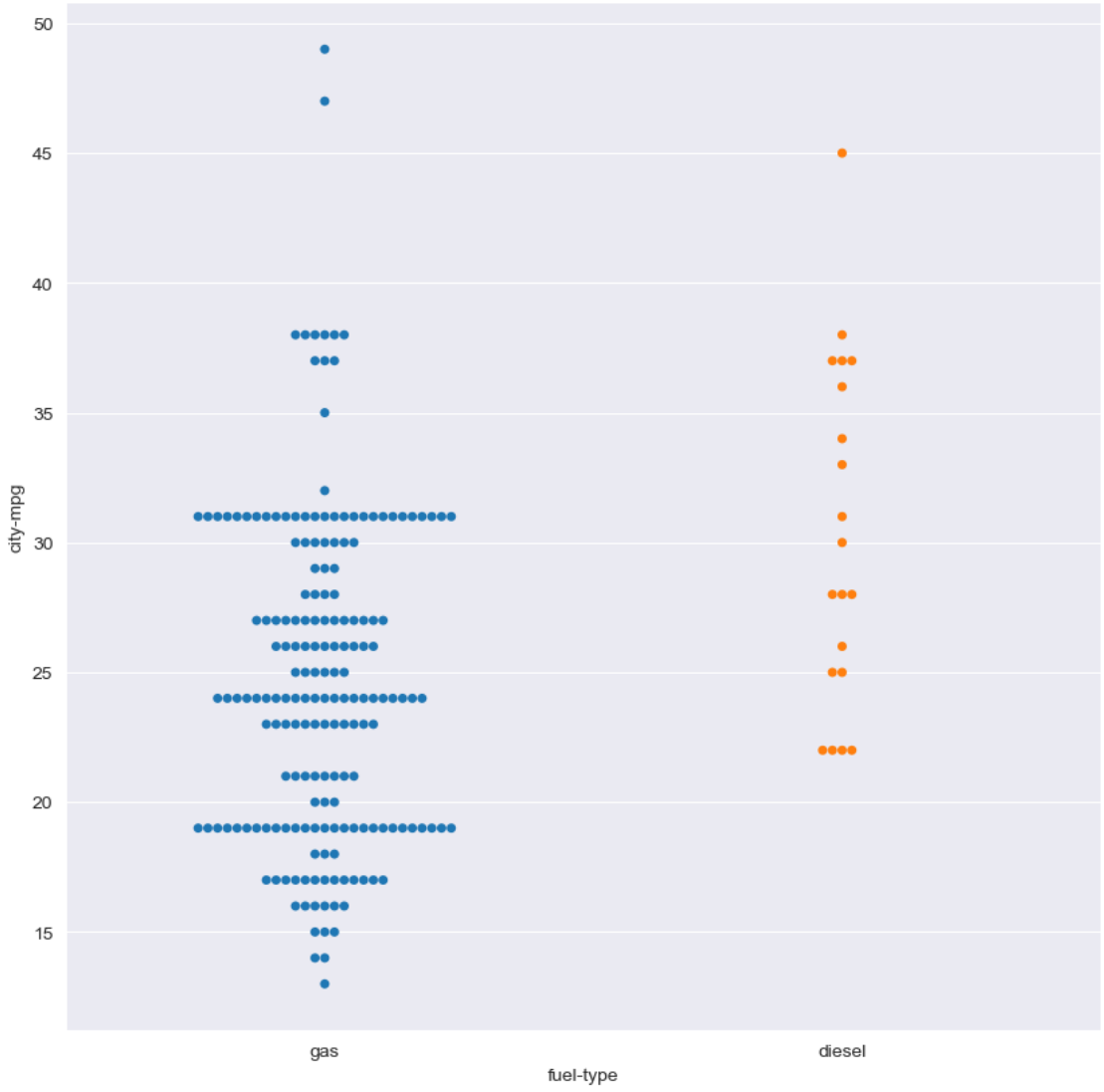


1.9 Swarm Plot

- A swarm plot is a type of categorical scatter plot used to display the distribution of data points in a dataset. It is similar to a scatter plot, but with a categorical variable on one axis, and the data points are evenly spaced out to avoid overlap.
- In a swarm plot, each data point is plotted as a point along the categorical axis, with its exact location determined by the data value. This creates a compact, yet informative visualization of the distribution of data, allowing for easy comparison between categories.
- Swarm plots are particularly useful when dealing with small to moderate sized datasets, and when the focus is on the distribution of data points rather than the underlying statistics of the data. They can be created using the Seaborn library in Python.

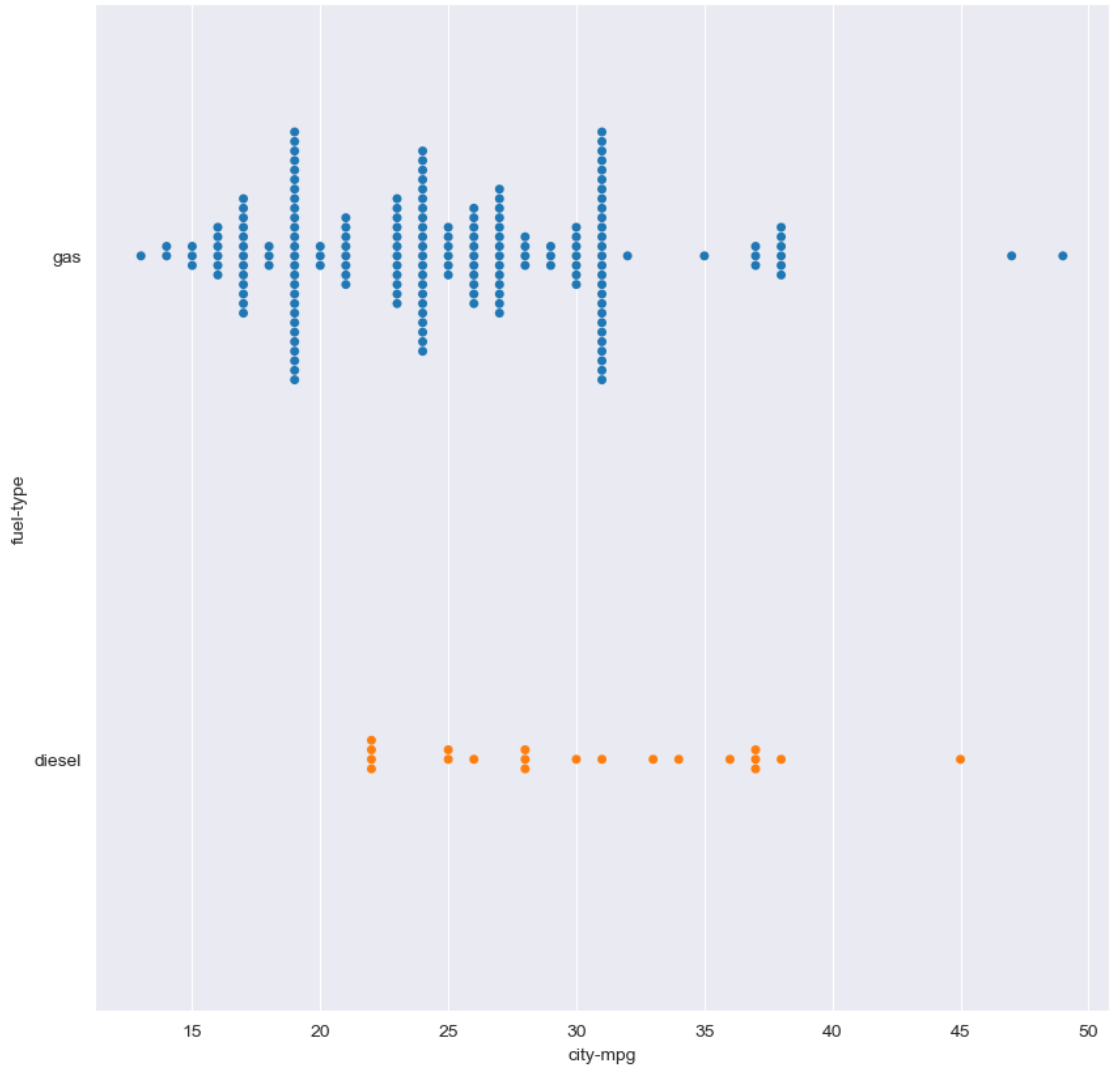
[62]:
```
plt.figure(figsize=(10,10))
sns.swarmplot(db['fuel-type'], db['city-mpg'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



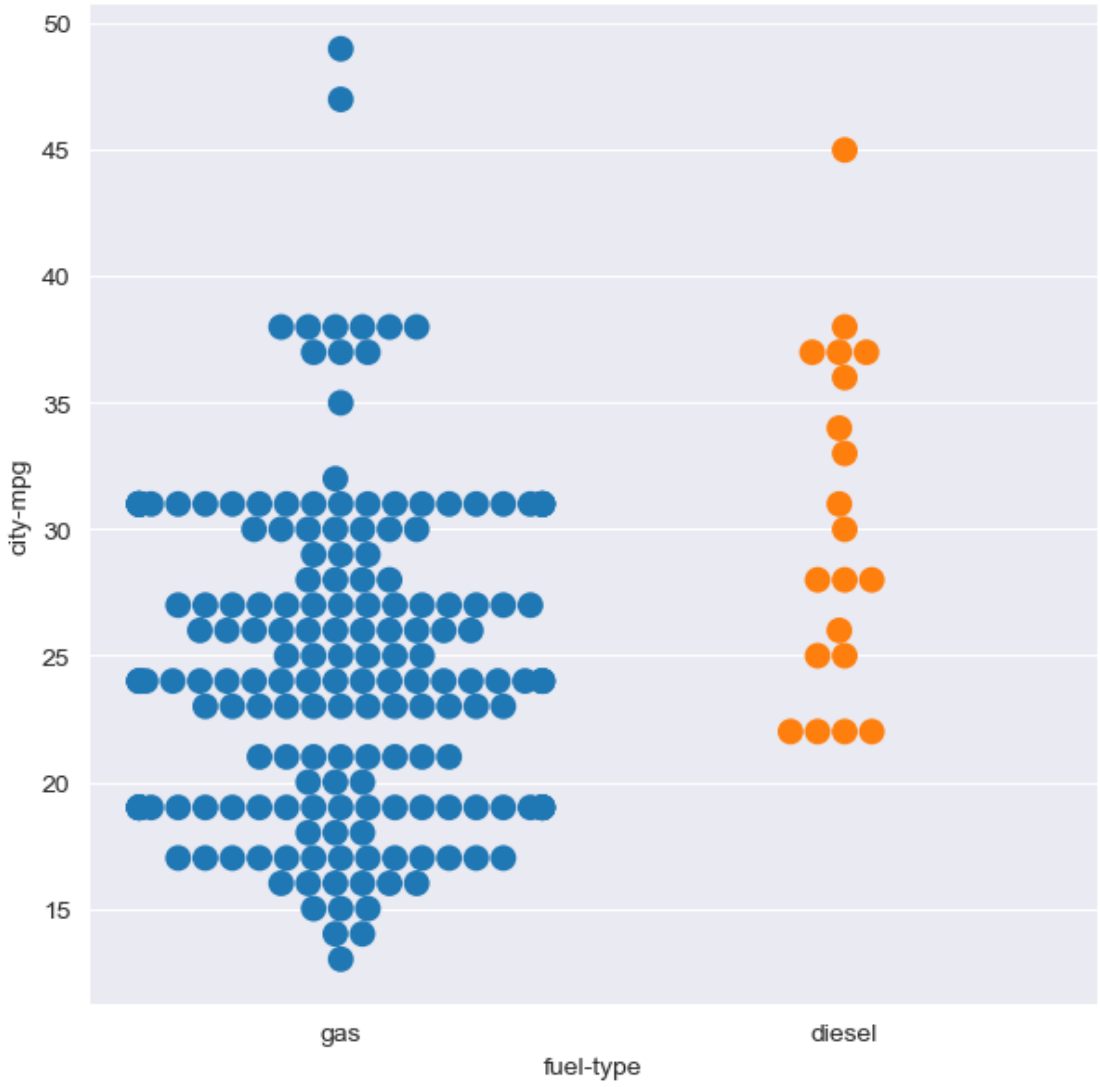
[63]:
```
plt.figure(figsize=(10,10))
sns.swarmplot(db['city-mpg'], db['fuel-type'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[64]:
```
plt.figure(figsize=(7,7))
sns.swarmplot(db['fuel-type'], db['city-mpg'], size=10)
plt.show()
```

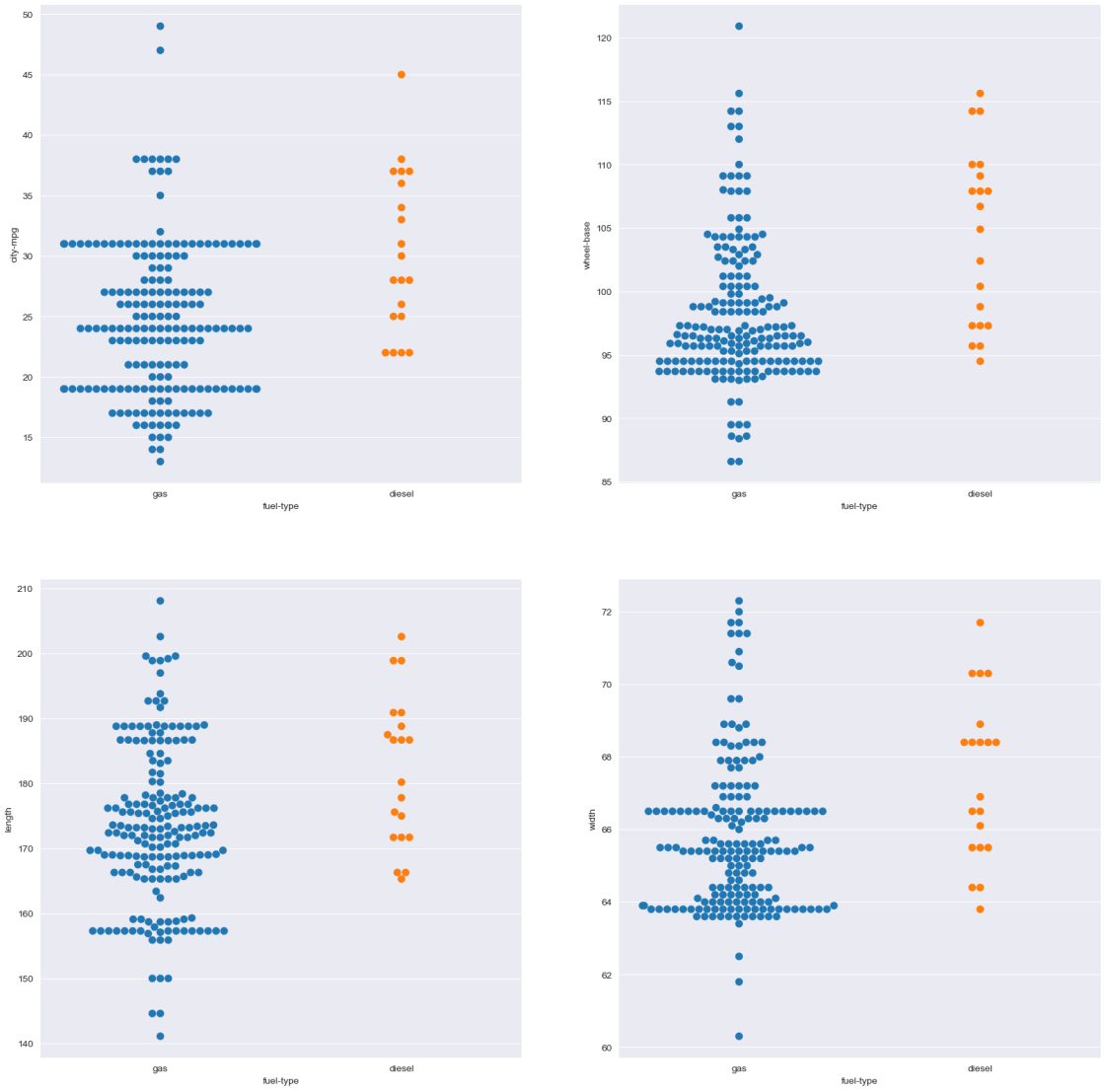
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 16.8% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



[65]:
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.swarmplot(db['fuel-type'], db['city-mpg'], ax = axes[0,0], size=8)
sns.swarmplot(db['fuel-type'] , db['wheel-base'] ,ax = axes[0,1], size=8)
sns.swarmplot(db['fuel-type'] ,db['length'] , ax = axes[1,0] , size=8)
sns.swarmplot(db['fuel-type'] , db['width'] , ax = axes[1,1] , size=8)
plt.show()
```

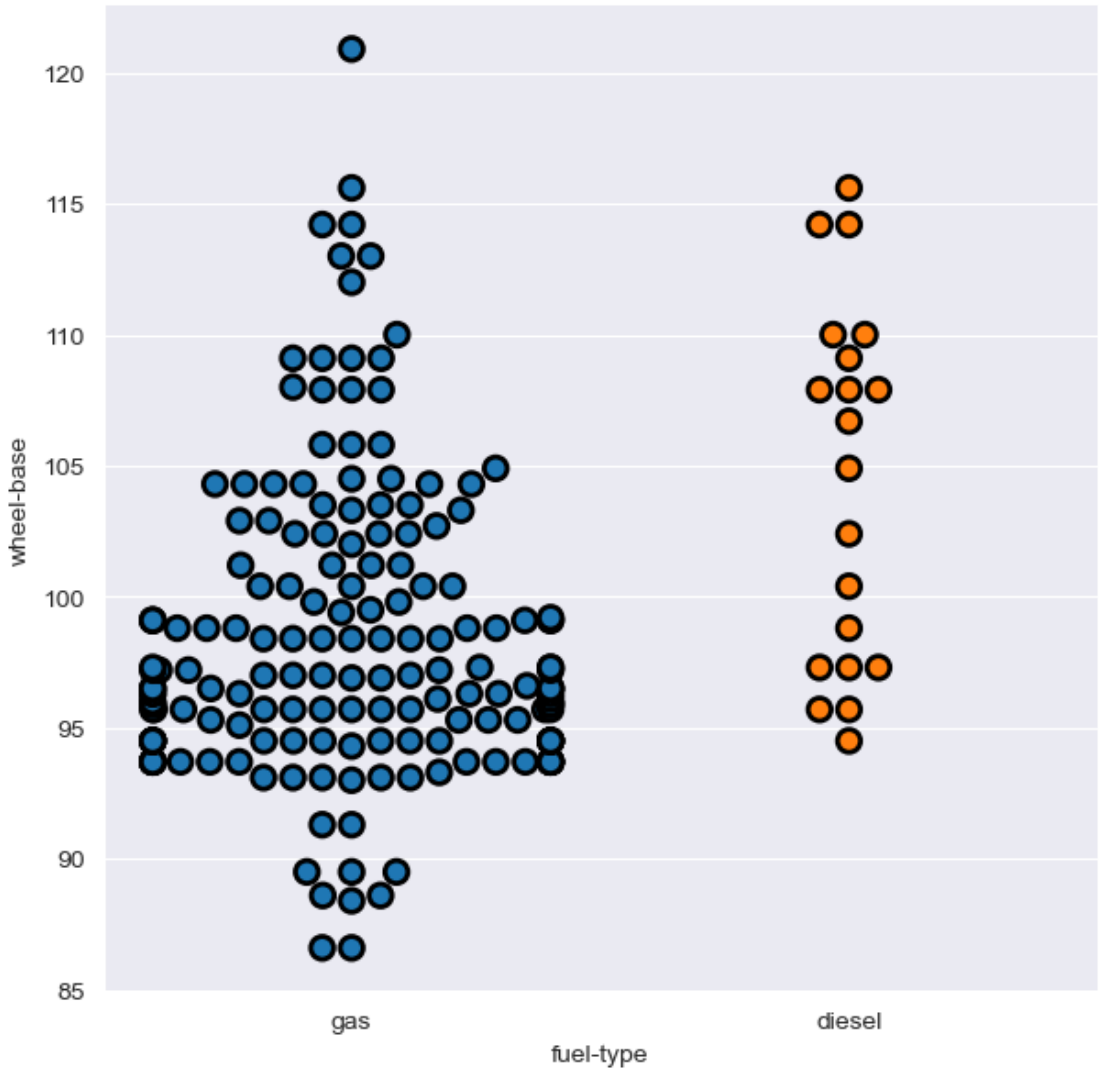
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
```

```
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[66]:
```
plt.figure(figsize=(7,7))
sns.swarmplot(x= "fuel-type", y = "wheel-base", size = 9 , linewidth= 2 ,␣
↪edgecolor="black" , data =db)
plt.show()
```

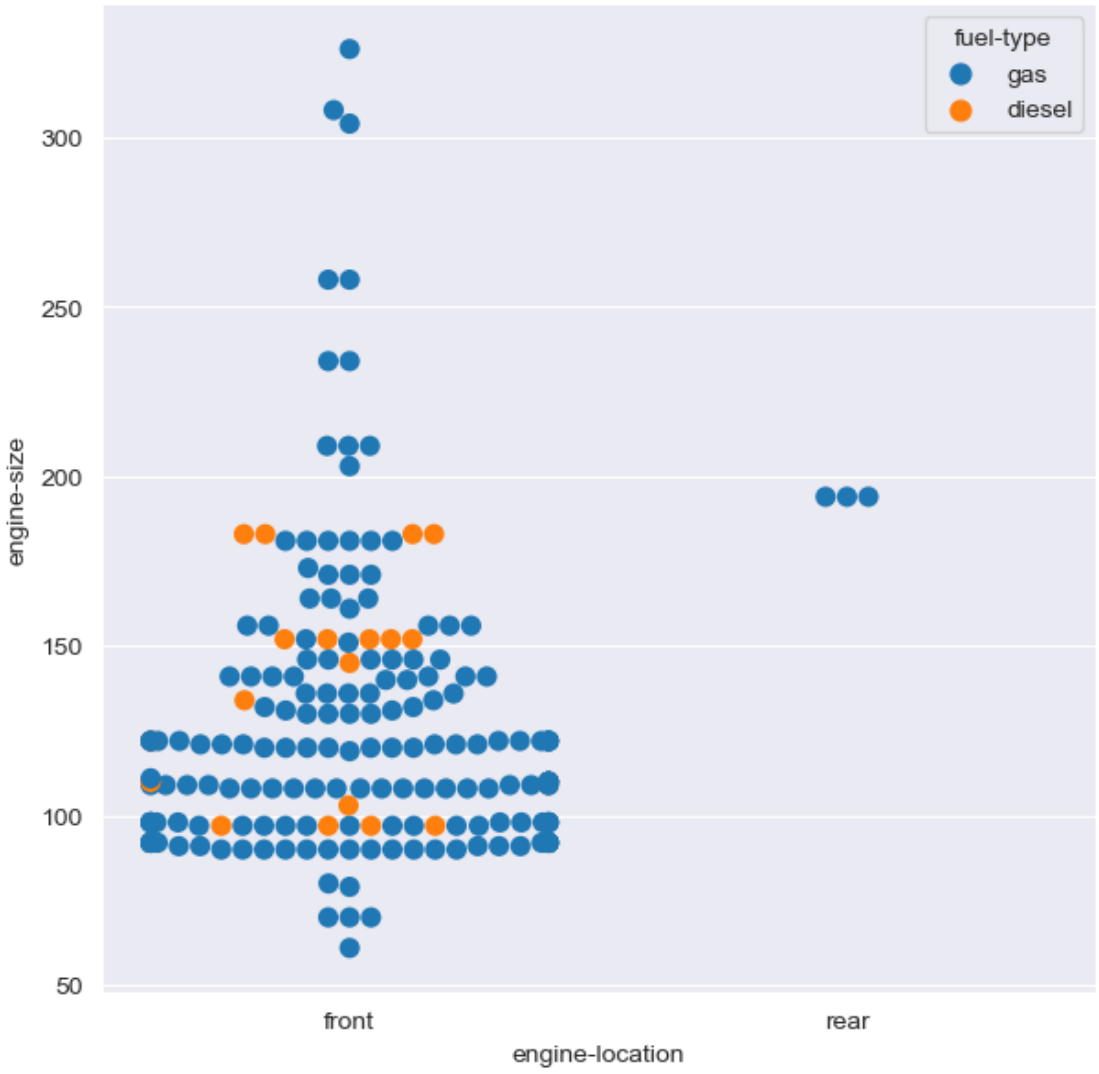
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 31.4% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



[67]:
```
plt.figure(figsize=(7,7))
sns.swarmplot(x= "engine-location", y = "engine-size", hue="fuel-type", size =␣
↪8 , data = db)
```

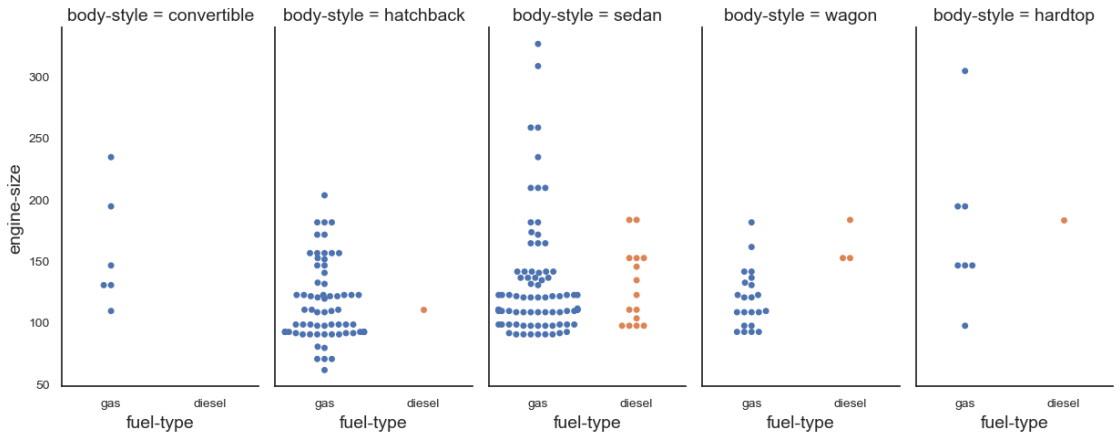
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 23.8% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

[67]:
```
<AxesSubplot:xlabel='engine-location', ylabel='engine-size'>
```



[68]:
```
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=db,␣
 ↪kind="swarm" , height=5,aspect=0.5)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 11.6% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 13.6% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



[69]:
```
sns.set(rc={'xtick.labelsize':10,'ytick.labelsize':10,'axes.labelsize':14})
sns.set_style("white")
sns.catplot(x= "engine-location", y = "engine-size", hue="fuel-type" , col=␣
 ↪"body-style" , data=db, kind="swarm" , height=5,aspect=0.5)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 11.4% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1296:
UserWarning: 18.8% of the points cannot be placed; you may want to decrease the
size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

1.10 Violin Plot

A violin plot is a type of data visualization that combines the features of a **box plot** and a **kernel density plot**. It is used to show the distribution of data across different categories, and to compare the shape and spread of these distributions.

In a violin plot, each category is represented by a **"violin"**, which is a mirrored density plot that shows the distribution of the data. The **width** of the violin represents the density of the data at that point, with wider parts indicating a higher density of data points. The narrow **"waist"** of the violin indicates the range of the **middle 50%**of the data, and the **white dot** in the middle of the violin represents the **median value**.

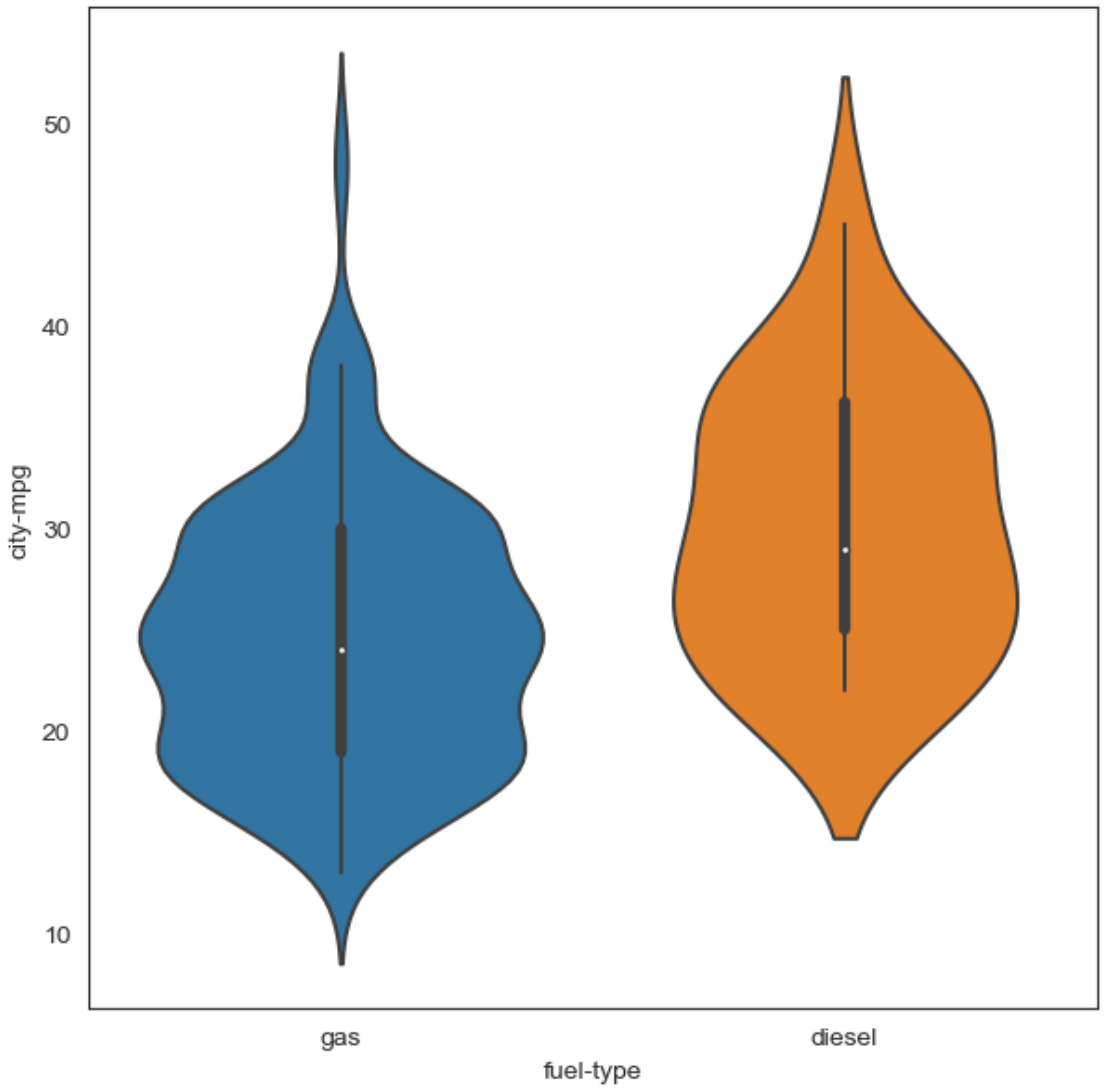
In addition to the violin shape, violin plots often include a box plot inside the violin, which shows the **quartiles** of the data and any **outliers**. This allows for easy comparison between categories, and can reveal differences in shape and spread that might not be apparent from looking at box plots or histograms alone.

Violin plots are a useful tool for exploring and visualizing complex datasets, and can be created using the Seaborn library in Python.

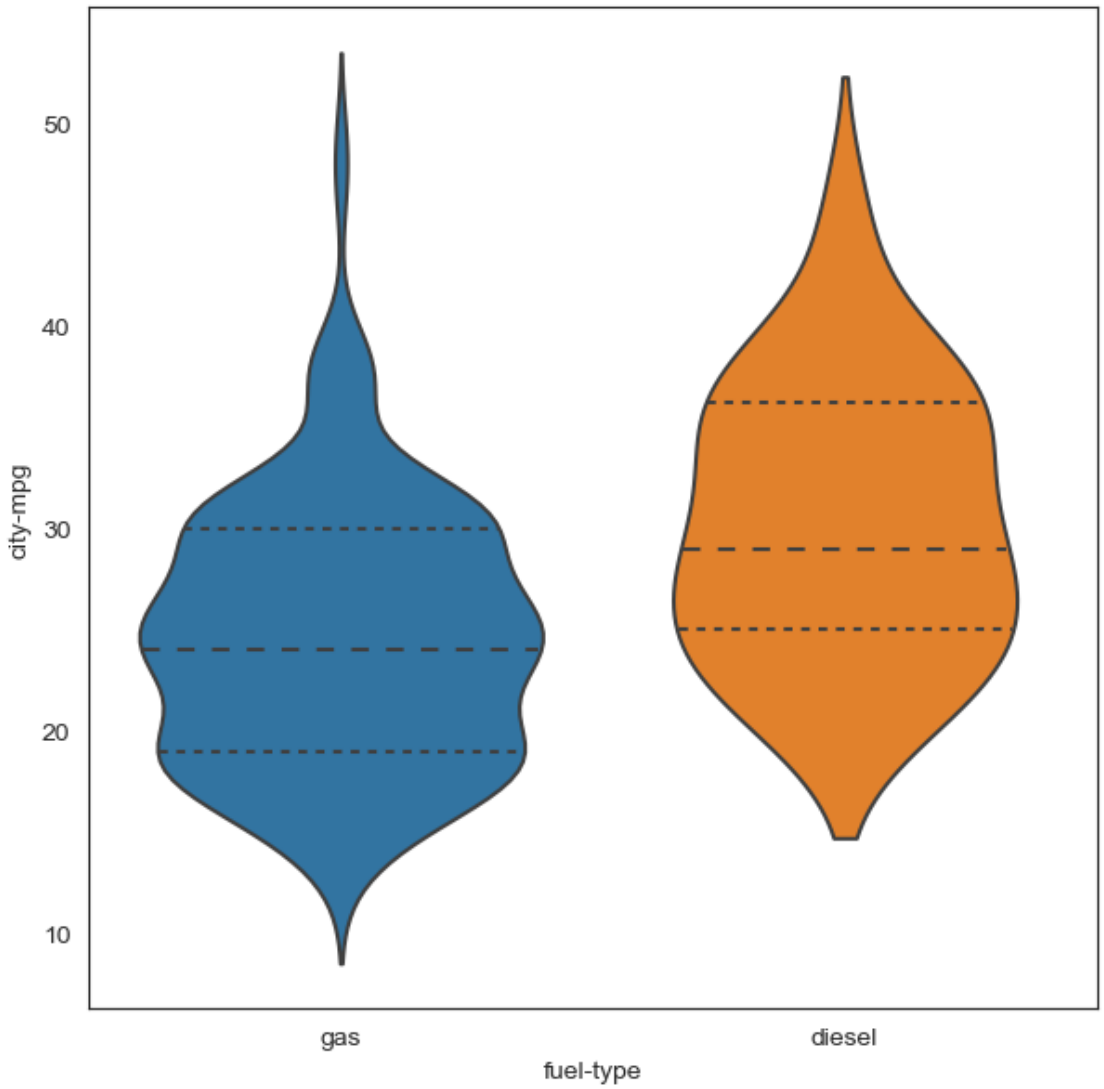
[70]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

[71]:
```
plt.figure(figsize=(7,7))
sns.violinplot(db['fuel-type'], db['city-mpg'])
plt.show()
```

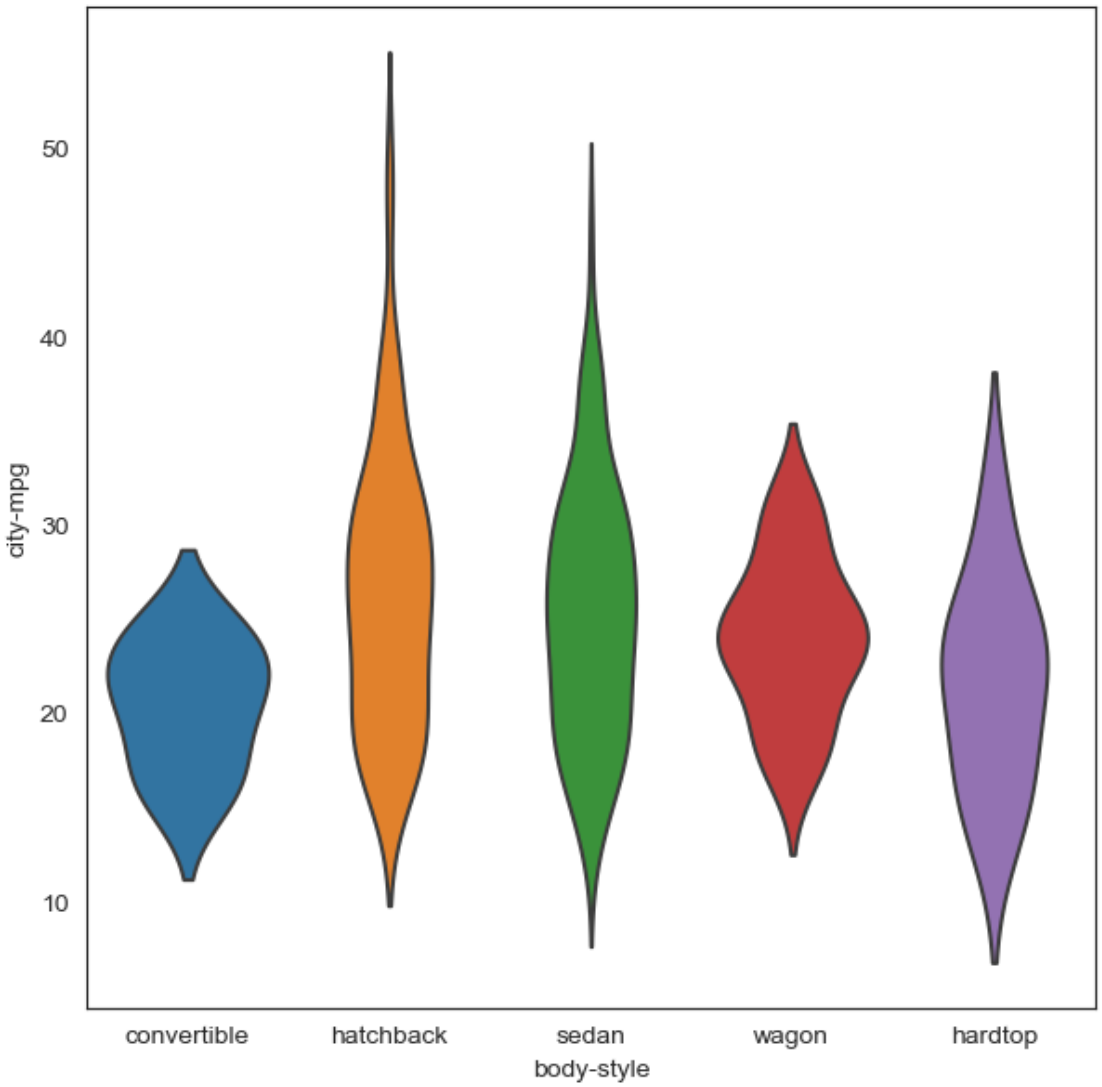
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



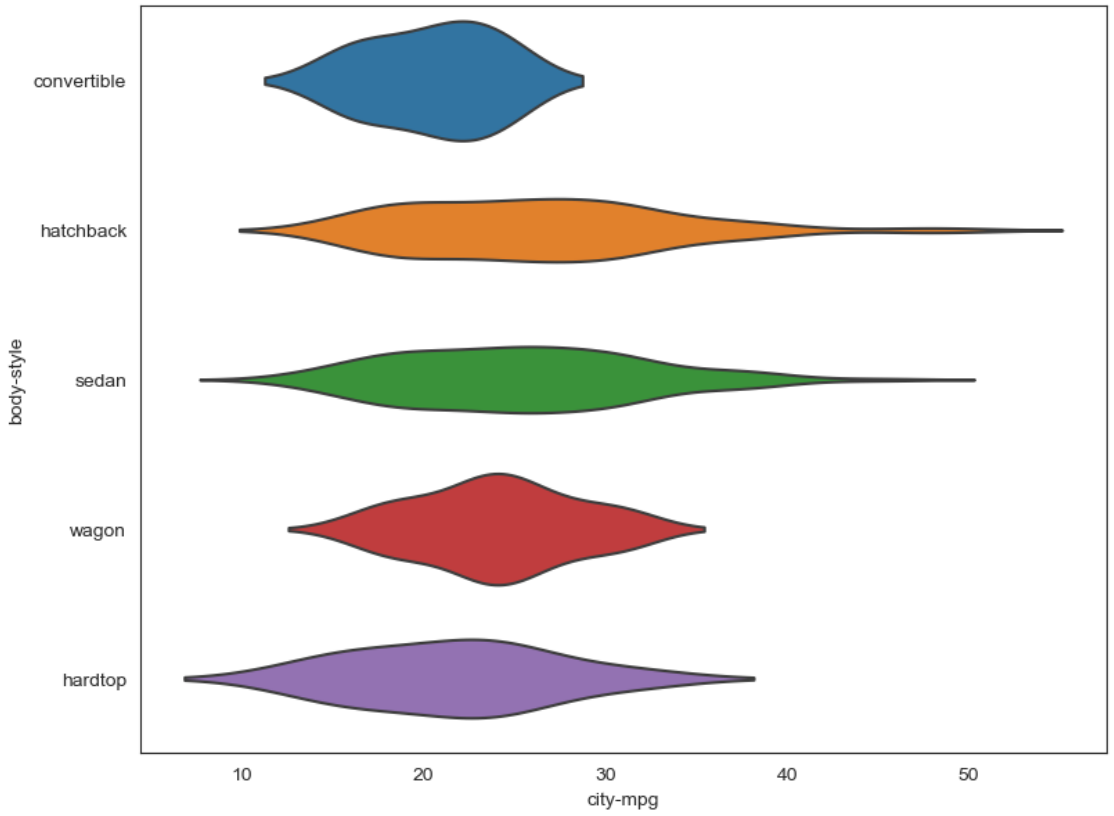
[72]:
```
plt.figure(figsize=(7,7))
sns.violinplot(x="fuel-type" , y = "city-mpg" ,data=db , inner="quartile")
plt.show()
```



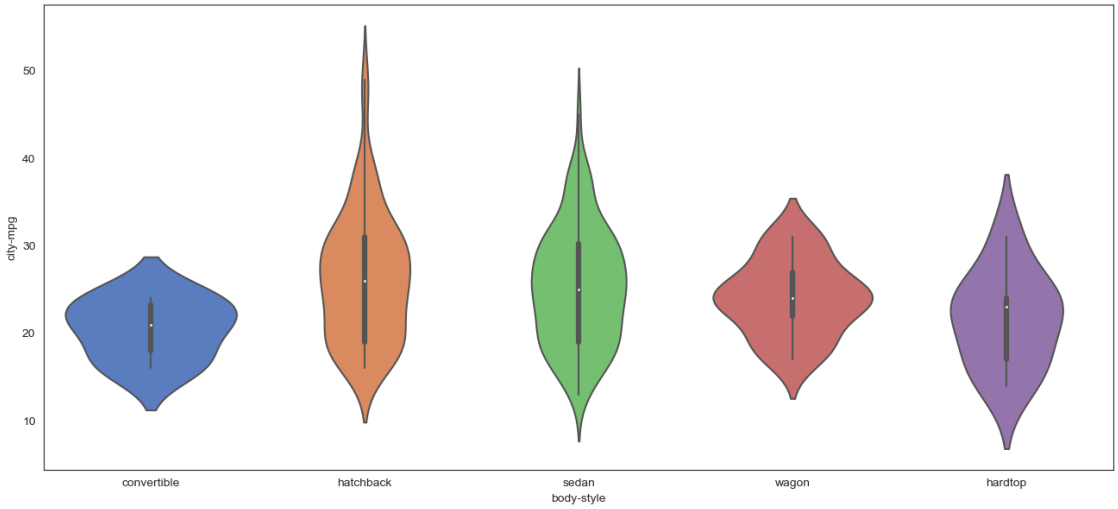
[73]:
```
plt.figure(figsize=(7,7))
sns.violinplot(x="body-style" , y = "city-mpg" ,data=db , inner=None)
plt.show()
```



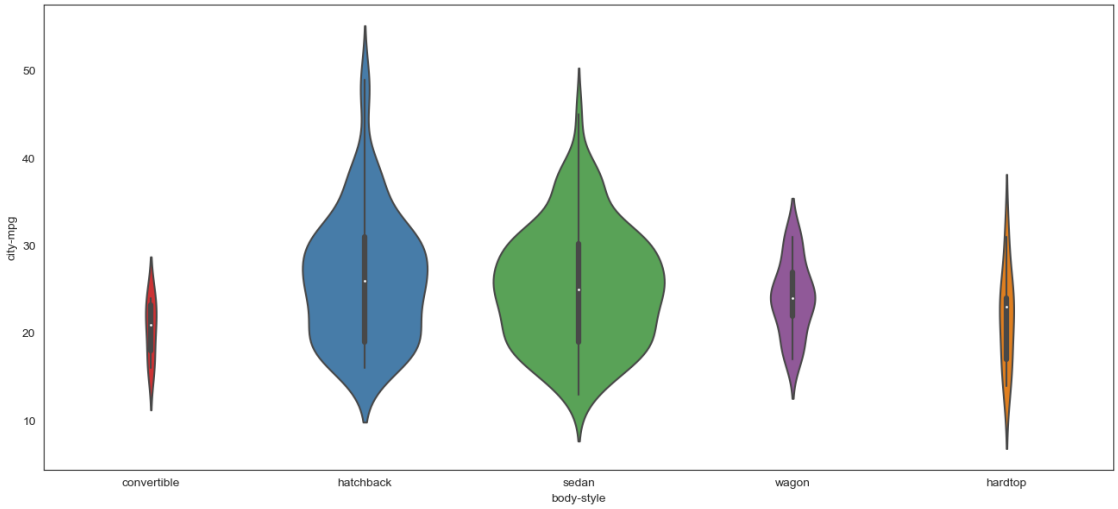
[74]:
```
plt.figure(figsize=(9,7))
sns.violinplot(y="body-style" , x = "city-mpg" ,data=db , inner=None)
plt.show()
```



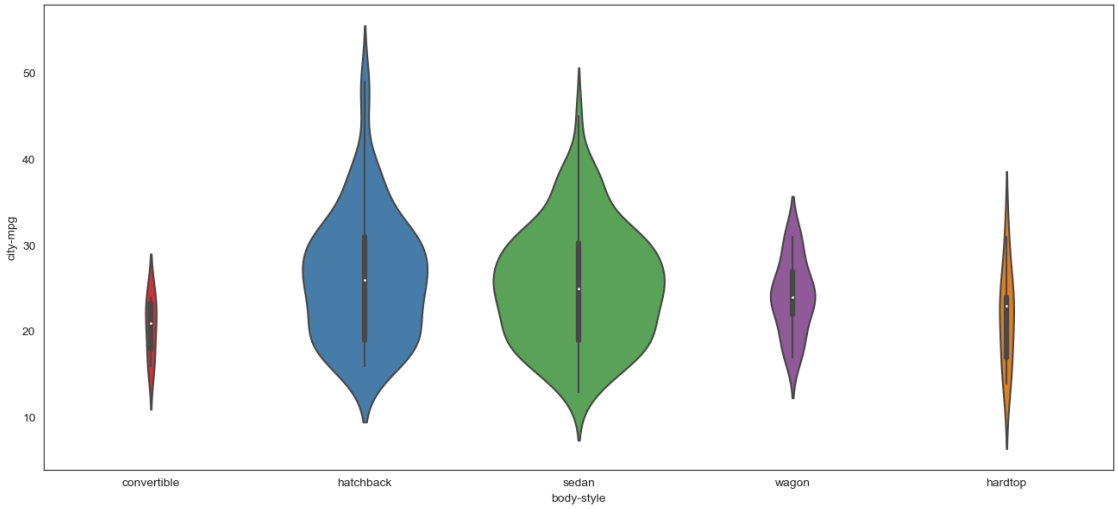
[75]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="muted")
plt.show()
```



[76]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1" ,␣
↪scale="count")
plt.show()
```

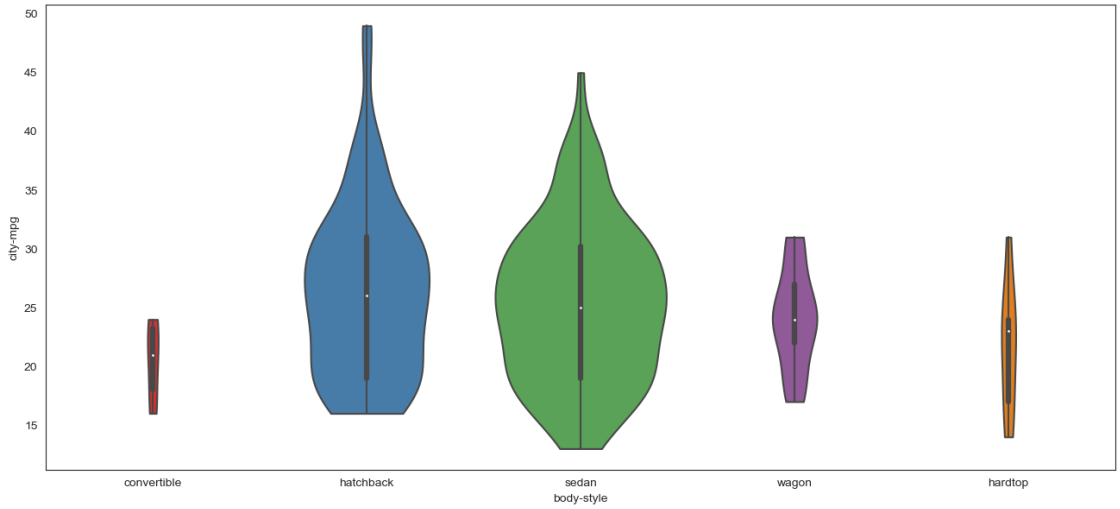


[77]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1" ,␣
↪scale="count",bw='silverman')
plt.show()
```

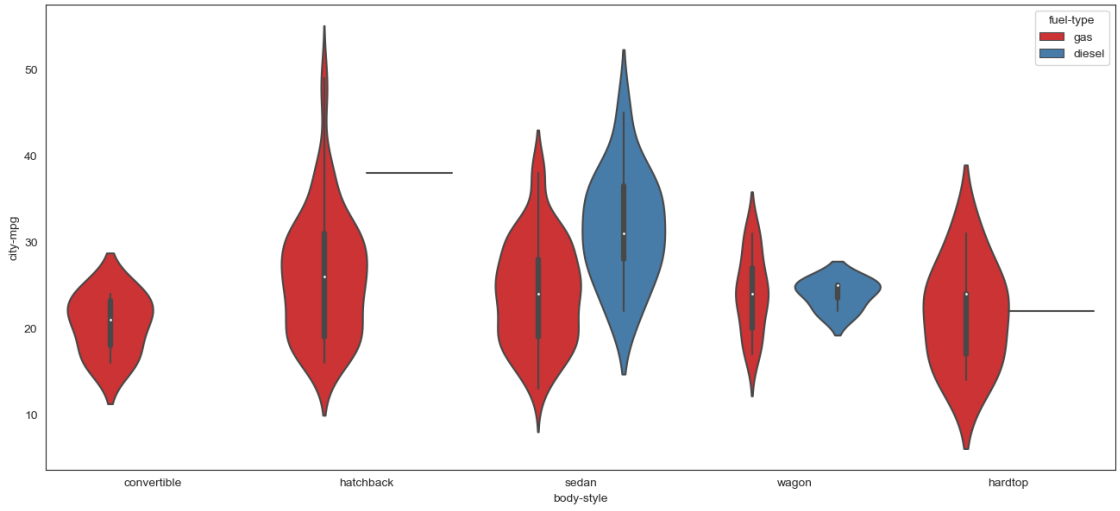


Limit the violin range within the range of the observed data using **cut =0**

[78]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1" ,␣
↪scale="count", cut =0)
plt.show()
```

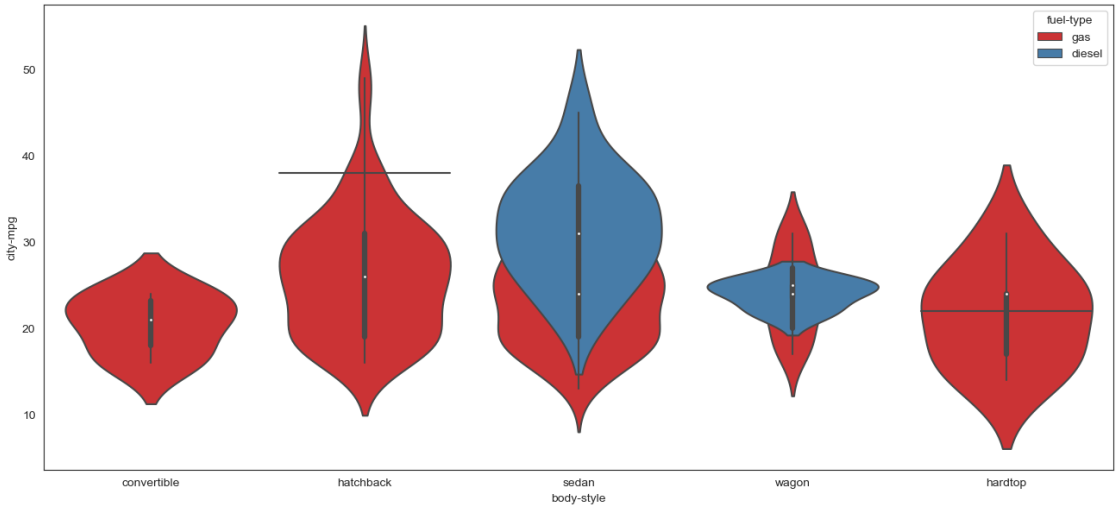


[79]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1", hue=␣
↪db["fuel-type"])
plt.show()
```

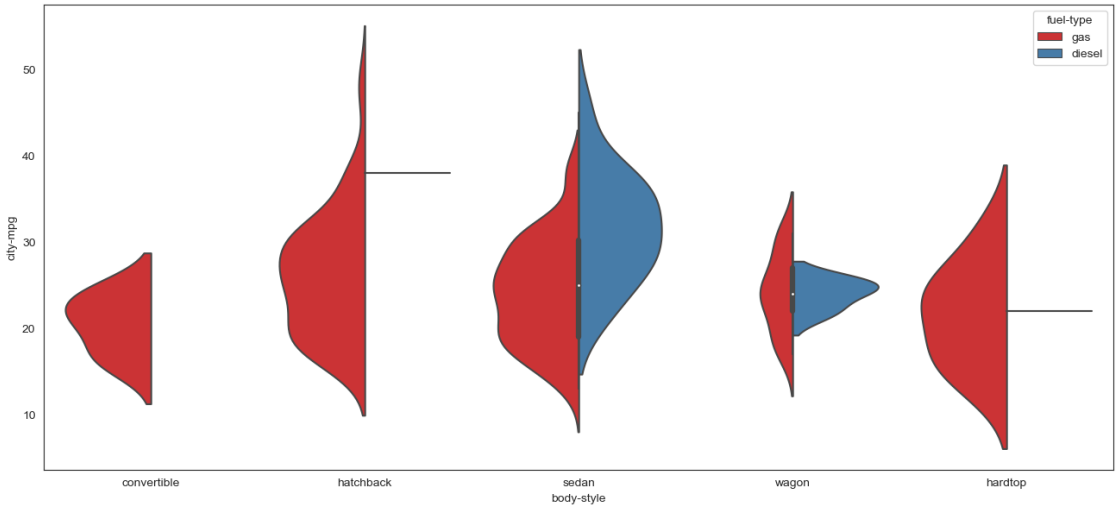


Overlap the violinplots for different hue levels along the categorical axis using **dodge=False**

[80]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1", hue=␣
↪db["fuel-type"],dodge=False)
plt.show()
```

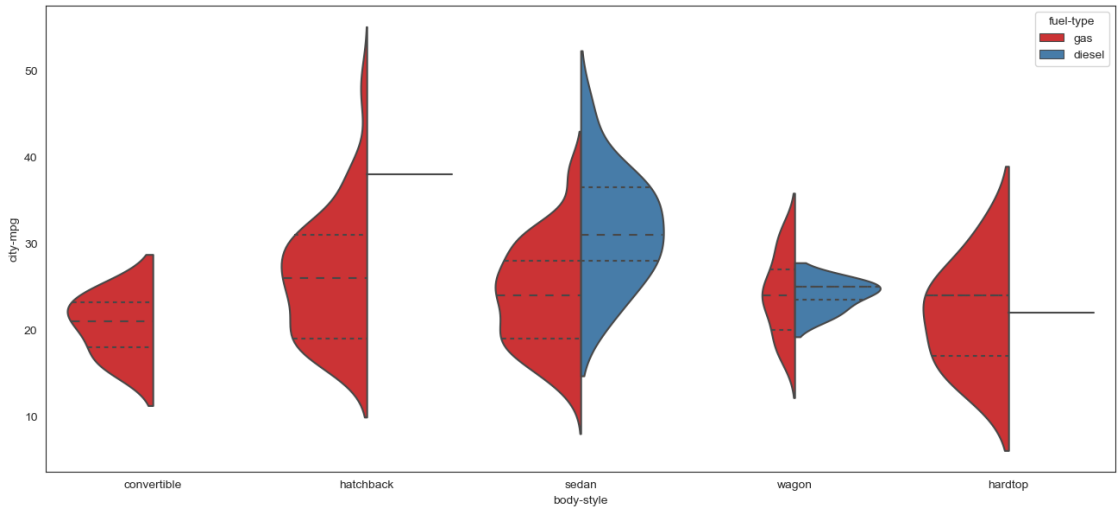


[81]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1", hue=␣
↪db["fuel-type"],split=True)
plt.show()
```



[82]:
```
plt.figure(figsize=(16,7))
```

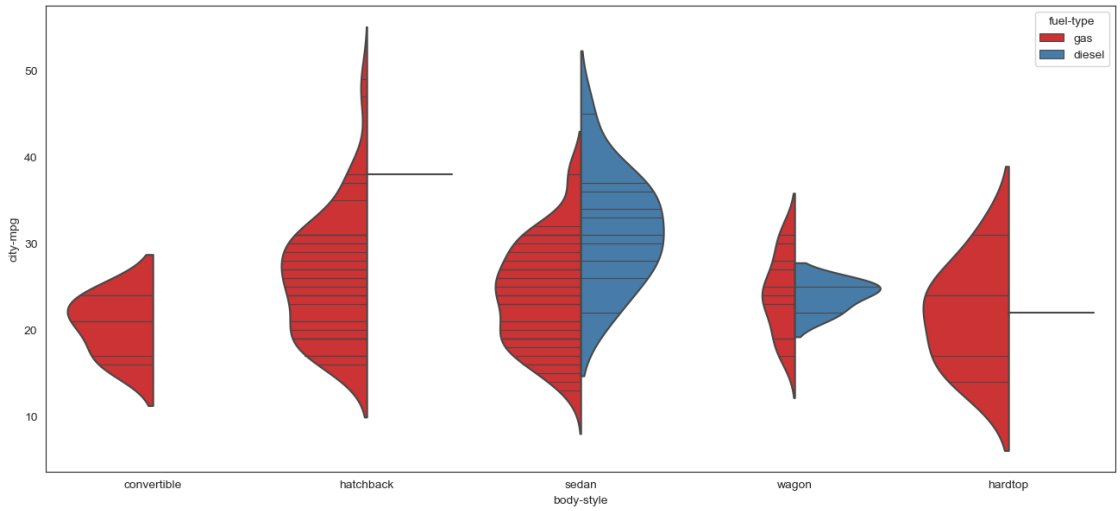
```
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1", hue=␣
↪db["fuel-type"],split=True ,inner="quartile")
plt.show()
```



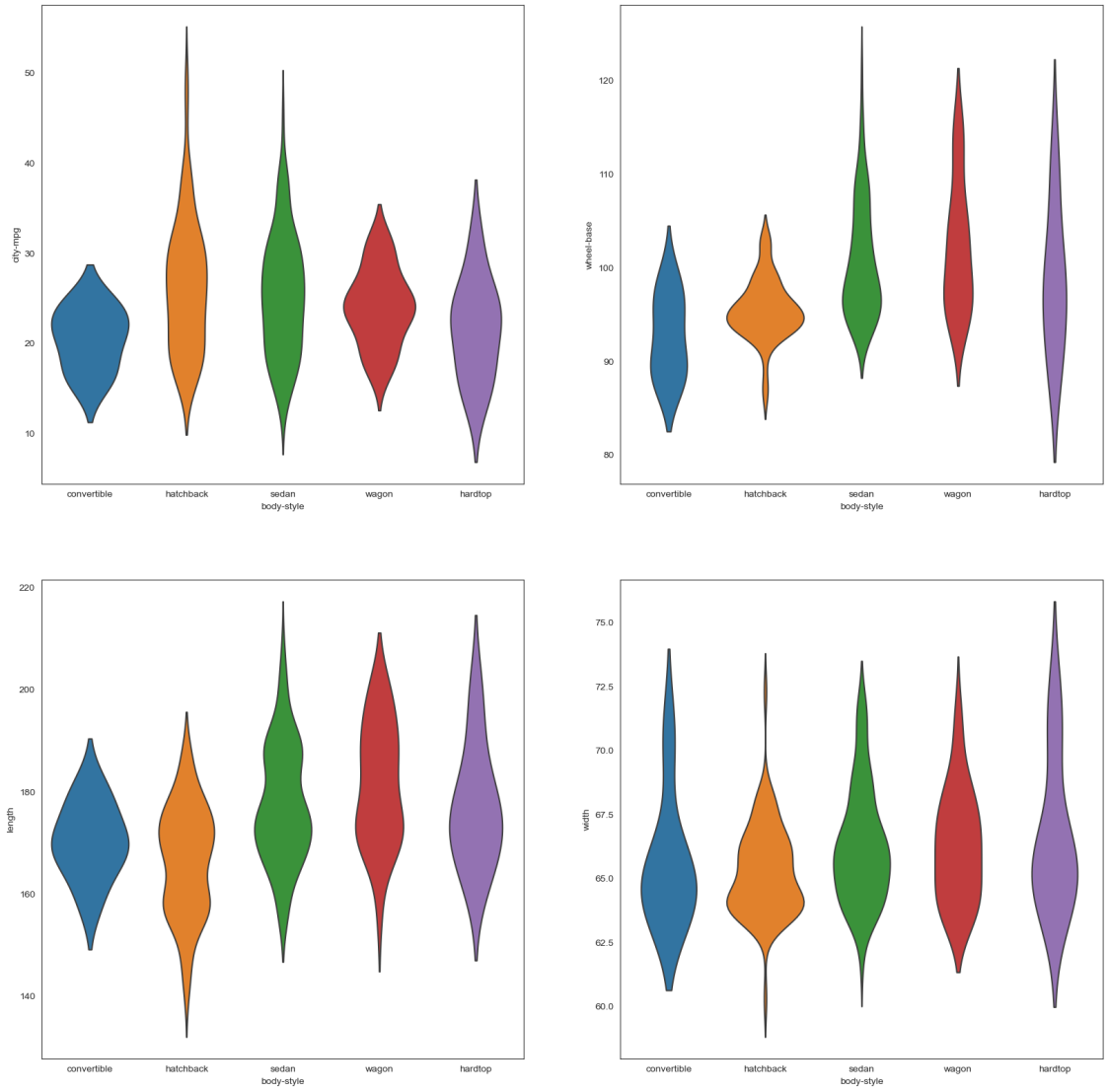
The **stick** option means that small vertical lines, or sticks, are drawn at each observation. This allows the viewer to see the individual data points and their distribution within each violin. The sticks are placed at random positions along the width of the violin, which can help to avoid overplotting when there are many data points.

Alternatively, you can also use other options such as “box”, “point”, “quartile”, “stick”, “mean”, “median”, or “none” to specify the type of data representation inside the violins.

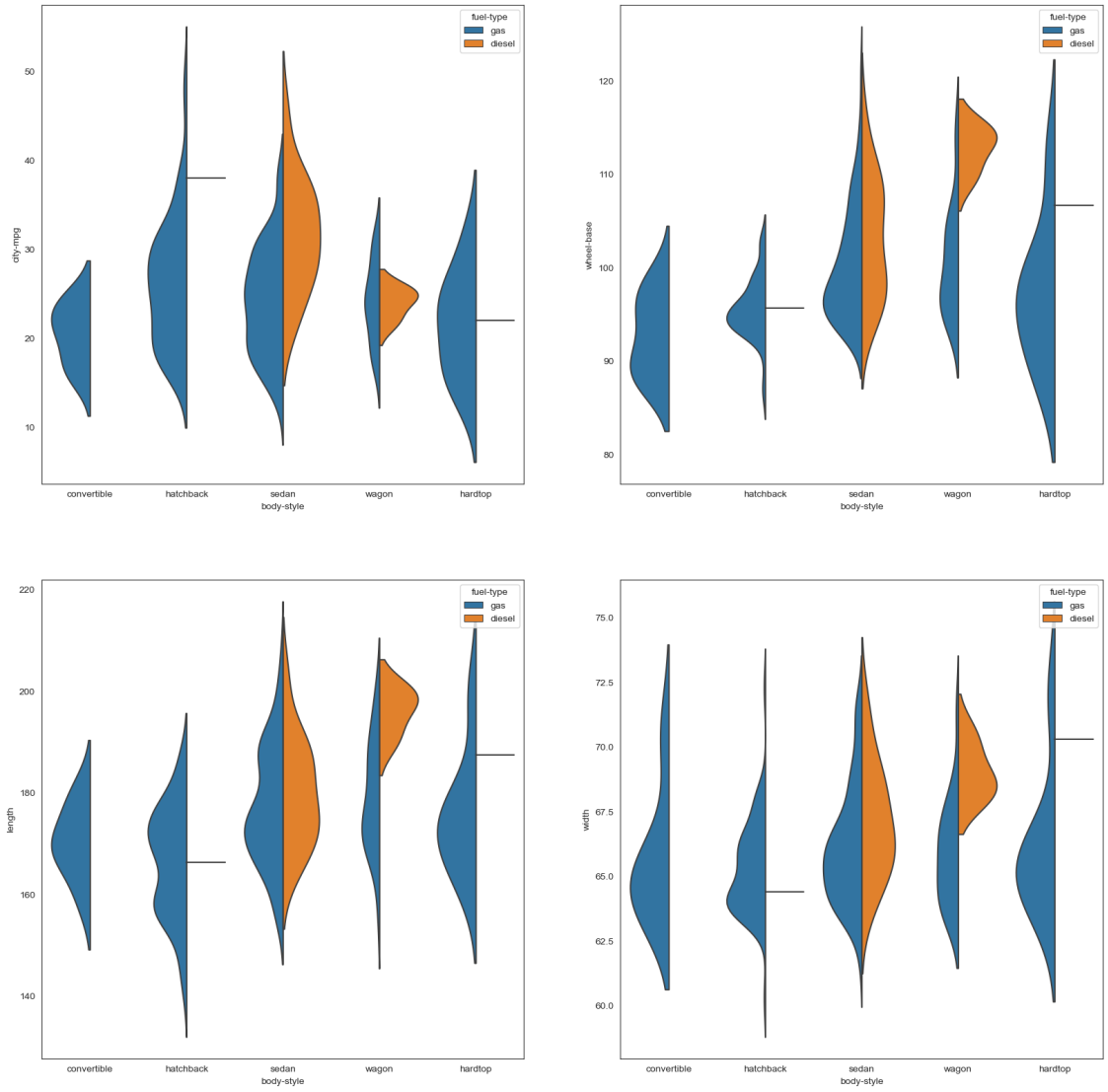
[83]:
```
plt.figure(figsize=(16,7))
sns.violinplot(x=db["body-style"] , y = db["city-mpg"] , palette="Set1", hue=␣
↪db["fuel-type"],split=True ,inner="stick")
plt.show()
```



[84]:
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.violinplot(x="body-style" , y = "city-mpg" , ax = axes[0,0] ,data=db ,␣
 ↪inner=None)
sns.violinplot(x="body-style" , y = "wheel-base" ,ax = axes[0,1] , data=db ,␣
 ↪inner=None)
sns.violinplot(x="body-style" , y = "length" , ax = axes[1,0] , data=db,␣
 ↪inner=None)
sns.violinplot(x="body-style" , y = "width" , ax = axes[1,1] , data=db,␣
 ↪inner=None )
plt.show()
```



[85]:
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.violinplot(x="body-style" , y = "city-mpg" , hue= db["fuel-type"] ,␣
 ↪split=True , ax = axes[0,0] ,data=db , inner=None)
sns.violinplot(x="body-style" , y = "wheel-base" , hue= db["fuel-type"] ,␣
 ↪split=True , ax = axes[0,1] , data=db , inner=None)
sns.violinplot(x="body-style" , y = "length" , hue= db["fuel-type"] ,split=True␣
 ↪, ax = axes[1,0] , data=db, inner=None)
sns.violinplot(x="body-style" , y = "width" , hue= db["fuel-type"] ,split=True␣
 ↪, ax = axes[1,1] , data=db, inner=None )
plt.show()
```

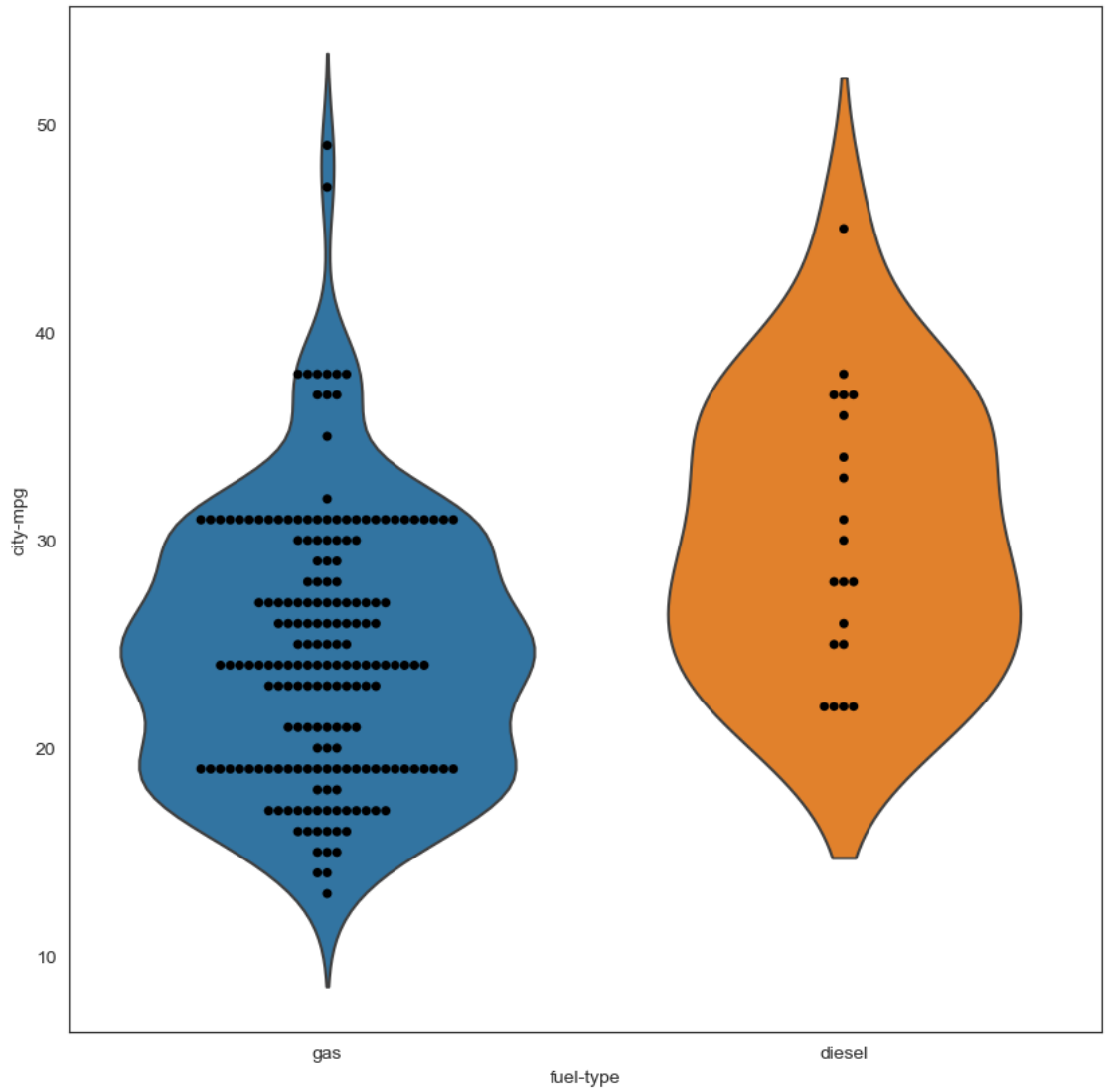


Displaying swarmplot on top of violin plot

[86]:
```
plt.figure(figsize=(10,10))
sns.swarmplot(db['fuel-type'], db['city-mpg'], color="#000000")
sns.violinplot(db['fuel-type'], db['city-mpg'], inner=None)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
```

```
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

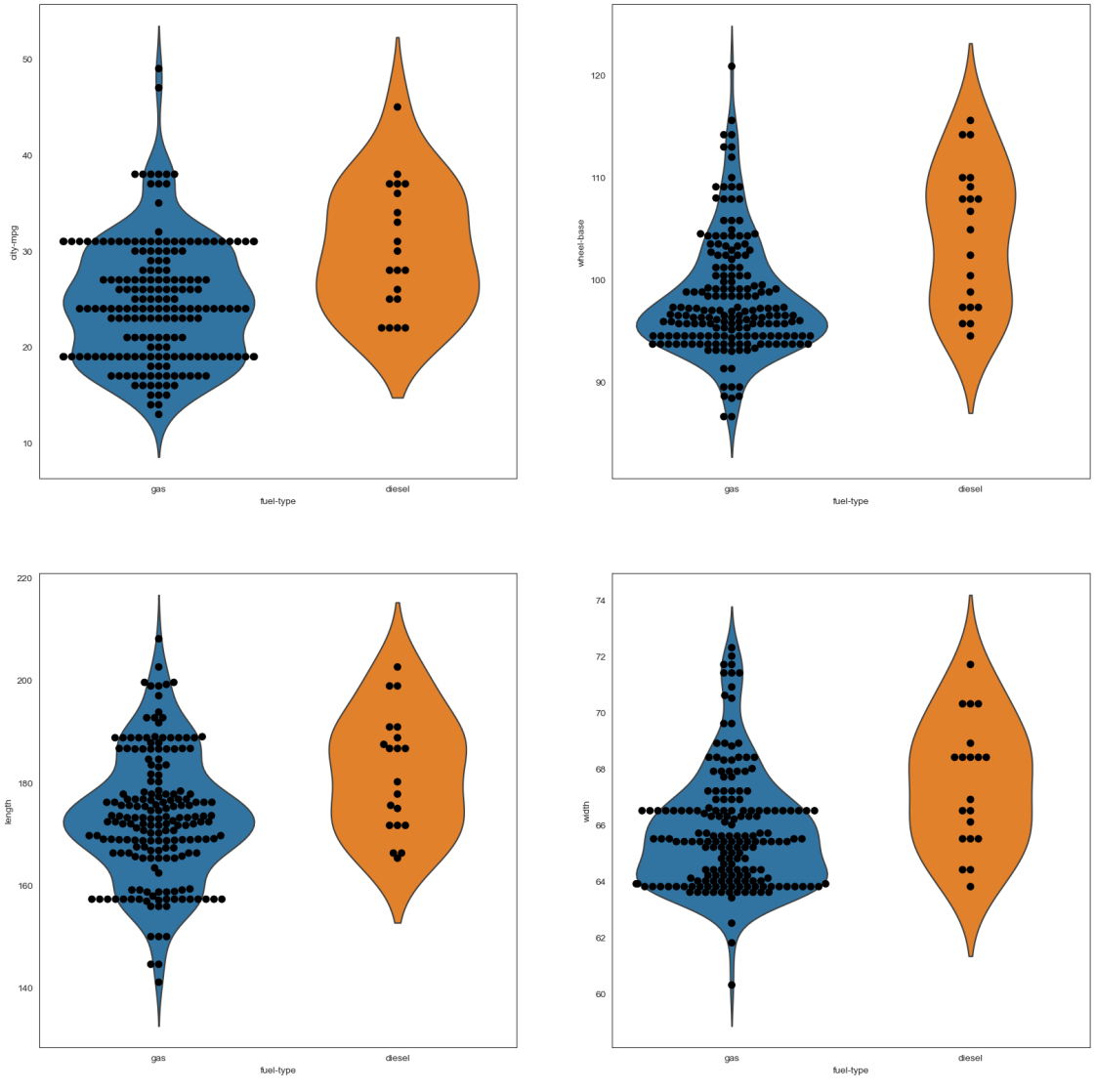


[87]:
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.swarmplot(db['fuel-type'], db['city-mpg'], ax = axes[0,0], size=8,␣
 ↪color="#000000")
sns.violinplot(db['fuel-type'], db['city-mpg'], ax = axes[0,0], size=8,␣
 ↪inner=None)
sns.swarmplot(db['fuel-type'] , db['wheel-base'] ,ax = axes[0,1], size=8,␣
 ↪color="#000000")
```

```
sns.violinplot(db['fuel-type'] , db['wheel-base'] ,ax = axes[0,1], size=8,␣
 ↪inner=None)
sns.swarmplot(db['fuel-type'] ,db['length'] , ax = axes[1,0] , size=8,␣
 ↪color="#000000")
sns.violinplot(db['fuel-type'] ,db['length'] , ax = axes[1,0] , size=8,␣
 ↪inner=None)
sns.swarmplot(db['fuel-type'] , db['width'] , ax = axes[1,1] , size=8,␣
 ↪color="#000000")
sns.violinplot(db['fuel-type'] , db['width'] , ax = axes[1,1] , size=8,␣
 ↪inner=None)
plt.show()
```

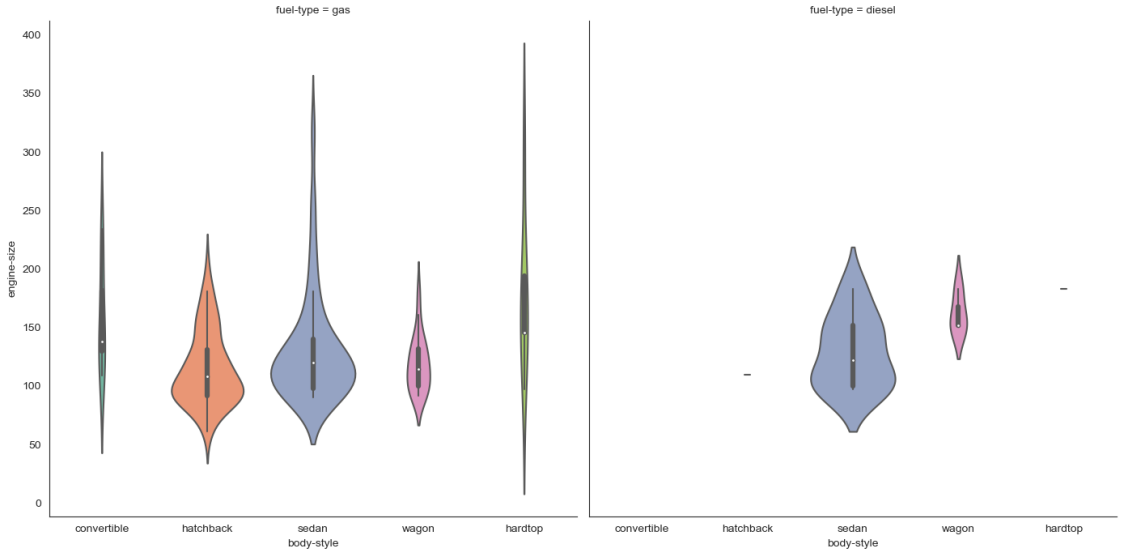
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
```

```
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[88]:
```
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type",␣
 ↪kind="violin",palette="Set2" , height=  7,scale =␣
 ↪"count",color="#000000",data=db)
plt.show()
```

```
<Figure size 1100x900 with 0 Axes>
```



1.11 Strip Plot

In data visualization, a **strip plot** is a type of plot that displays the distribution of a continuous variable by plotting each data point as a point along an axis. Strip plots are useful for visualizing the distribution of a variable and identifying outliers and patterns in the data.

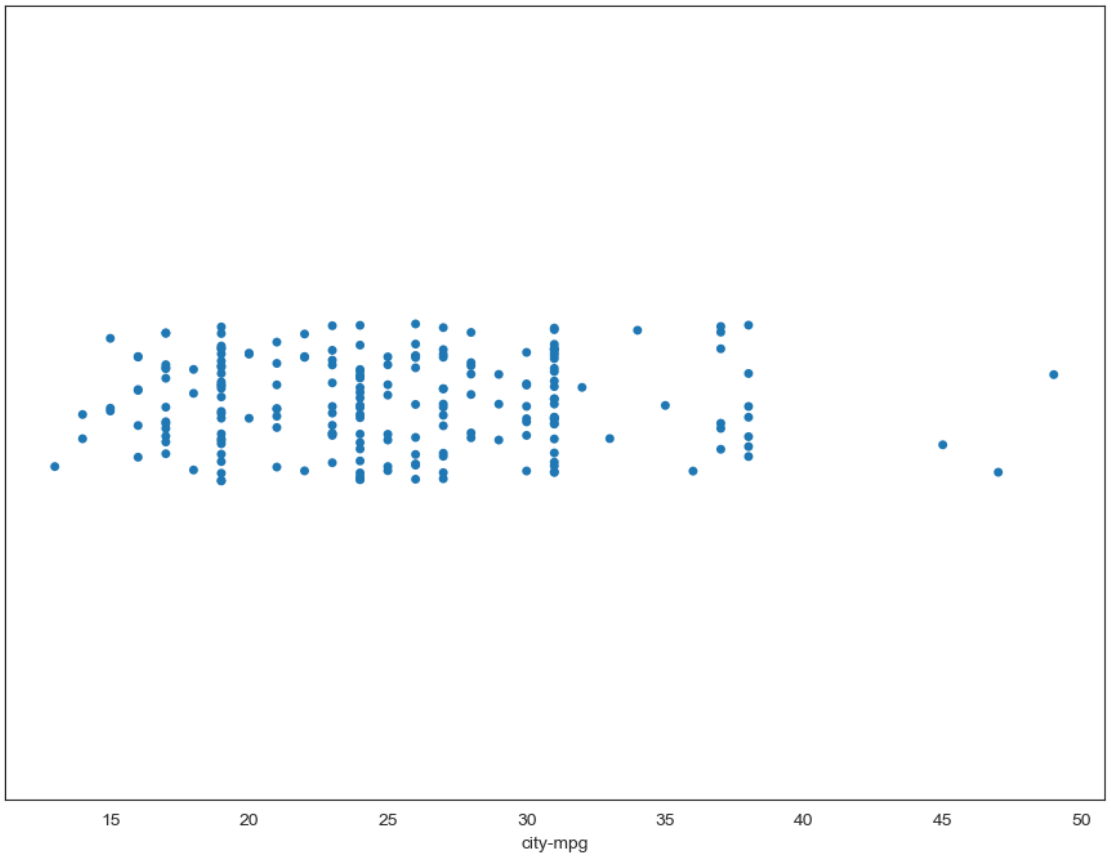
In a strip plot, the data points are arranged along the axis based on their value, with similar values grouped together. This can create clusters of points that represent areas of high density in the data.

Strip plots can be enhanced by using color or other visual cues to represent additional variables. For example, different colors or shapes can be used to represent groups or categories within the data.

Strip plots are similar to scatter plots, but they are typically used for visualizing a single variable rather than the relationship between two variables. They are also similar to beeswarm plots, but beeswarm plots use an algorithm to arrange the data points in a way that minimizes overlap, while strip plots simply plot the points along the axis.

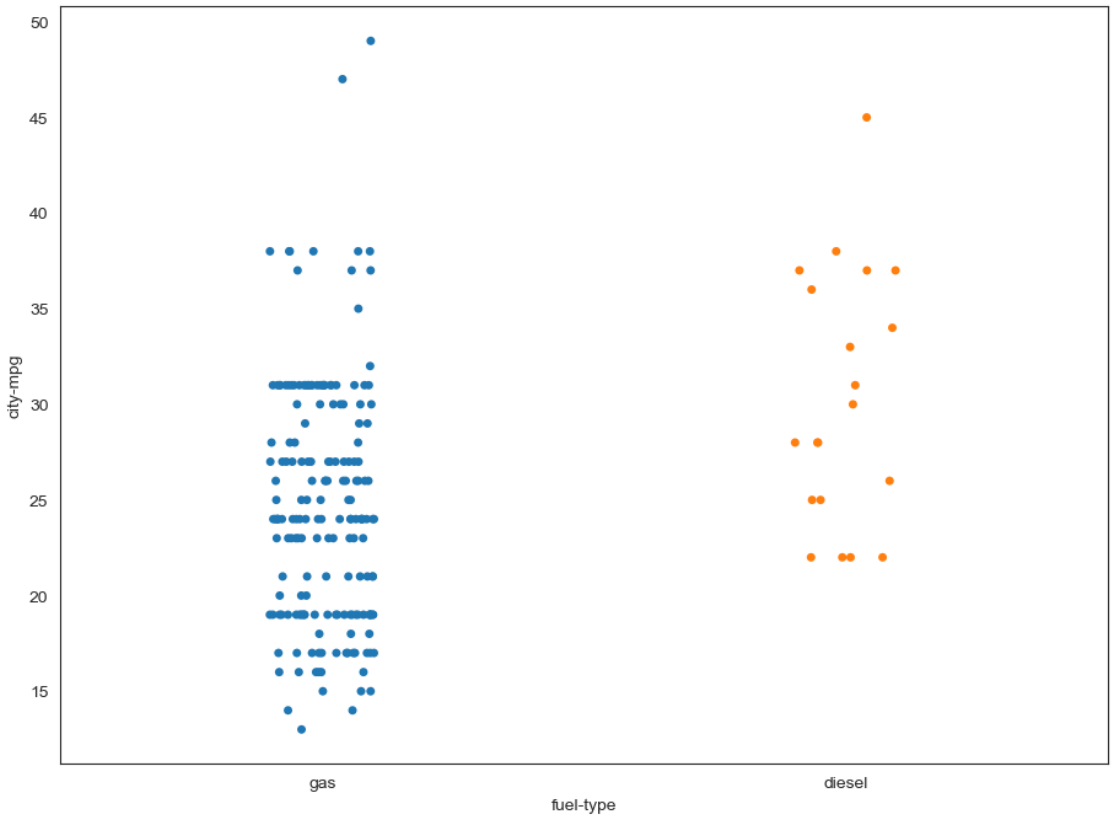
[89]:
```
plt.figure(figsize=(11,8))
sns.stripplot(db['city-mpg'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

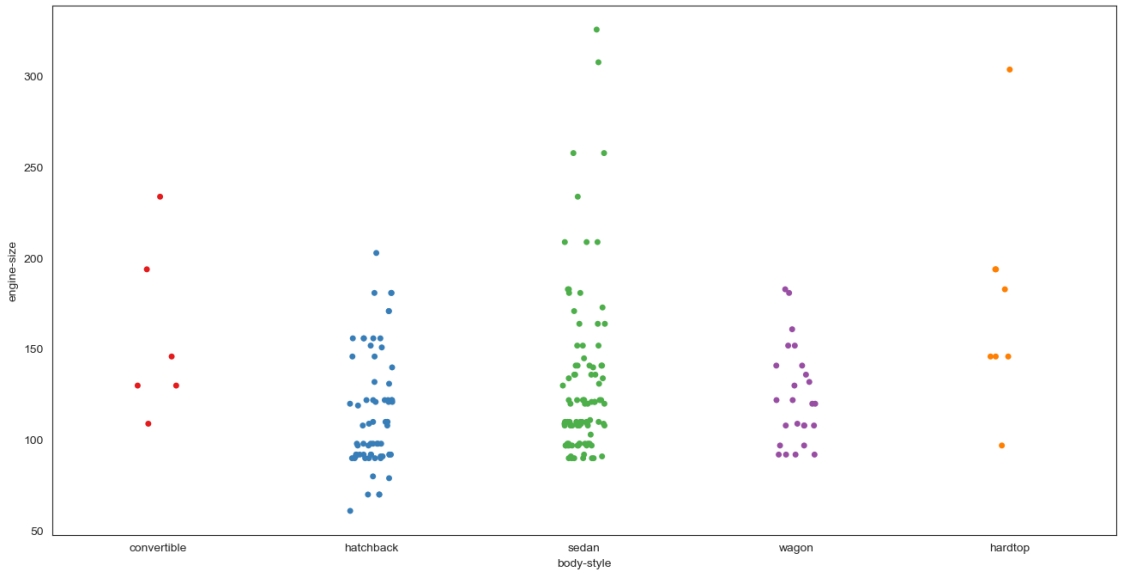


[90]:
```
plt.figure(figsize=(11,8))
sns.stripplot(db['fuel-type'], db['city-mpg'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[91]:
```
plt.figure(figsize=(16,8))
sns.stripplot(x=db["body-style"] ,palette="Set1", y = db["engine-size"])
plt.show()
```

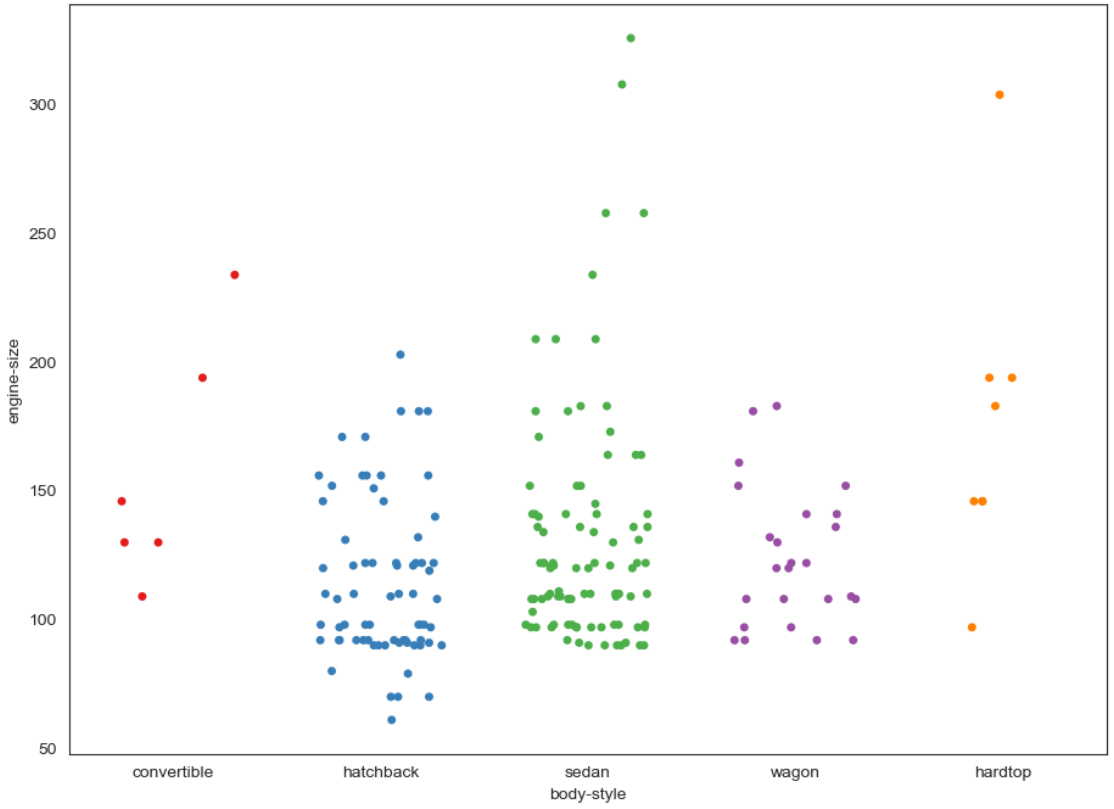


The **jitter** parameter controls the amount of jitter (**random noise**) applied to the position of the points along the categorical axis.

When plotting a strip plot, multiple points with the same categorical value can overlap, making it difficult to see the true distribution of the data. By setting jitter to a small value (in this case, **0.3**), the positions of the points are randomly adjusted by a small amount to prevent them from overlapping.

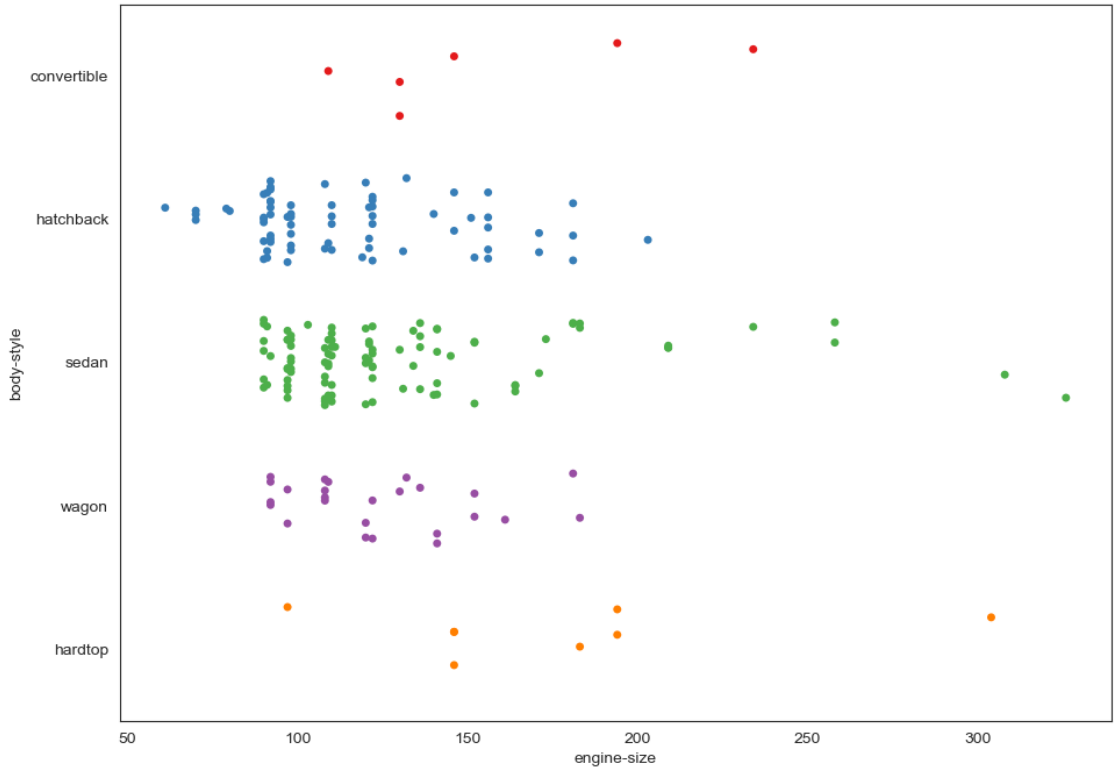
Increasing the value of jitter will result in more random variation in the positions of the points, which can help to make the distribution of the data more apparent. However, if the value of jitter is set too high, the positions of the points may become too spread out, making it difficult to see patterns in the data.

[92]:
```
plt.figure(figsize=(11,8))
sns.stripplot(x=db["body-style"] ,palette="Set1", y = db["engine-size"],jitter=.
↪3)
plt.show()
```

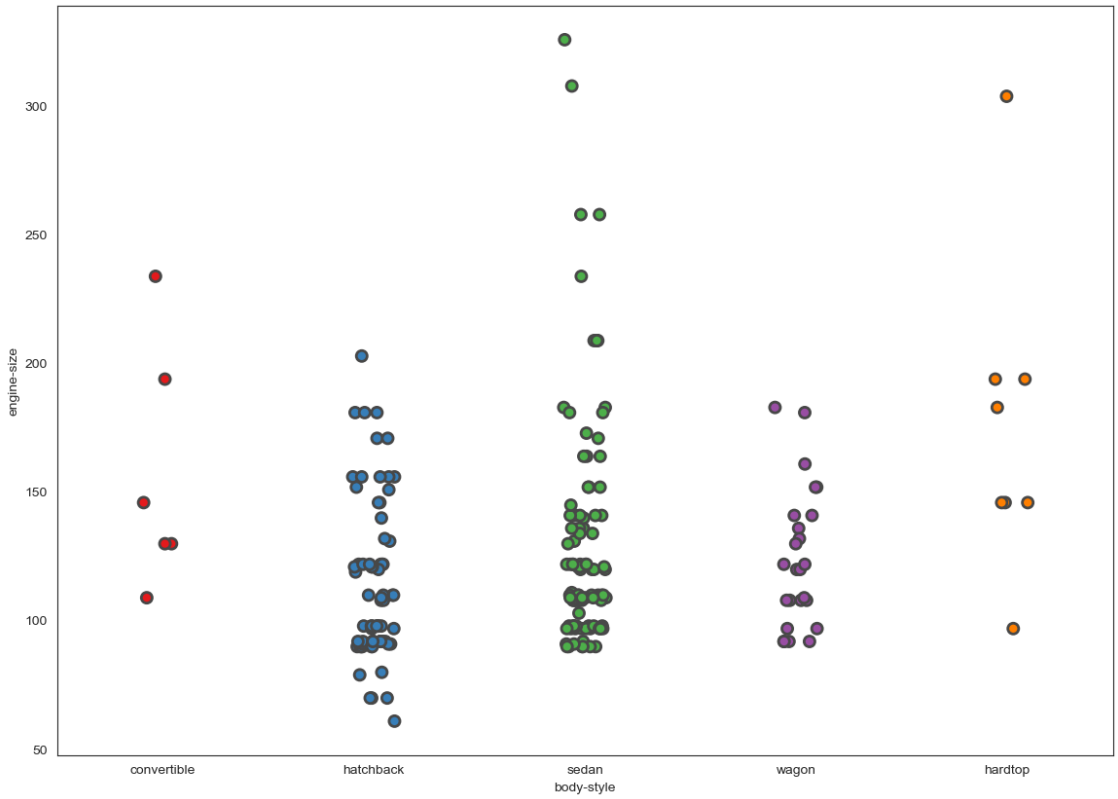


[93]:
```
plt.figure(figsize=(11,8))
sns.stripplot(y=db["body-style"] ,palette="Set1", x = db["engine-size"] ,␣
↪jitter=0.3)
```

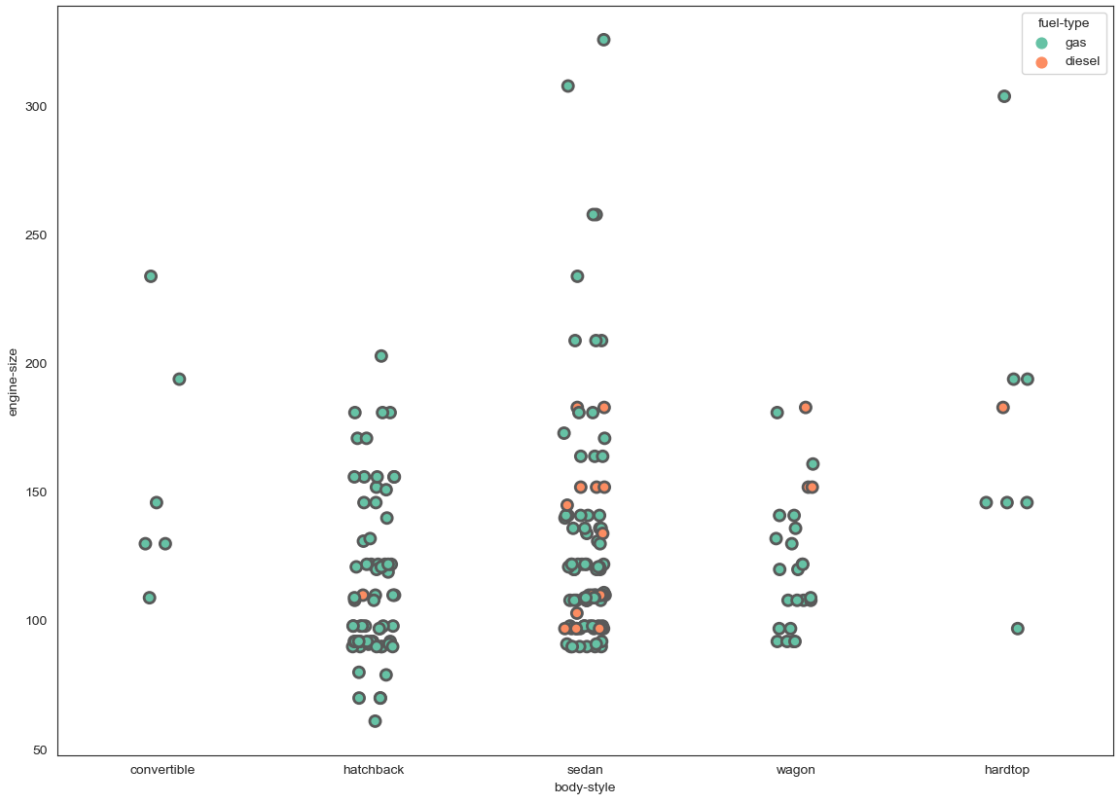
```
plt.show()
```



[94]:
```
plt.figure(figsize=(14,10))
sns.stripplot(x=db["body-style"] ,palette="Set1", y =␣
  ↪db["engine-size"],linewidth=2,size=8)
plt.show()
```

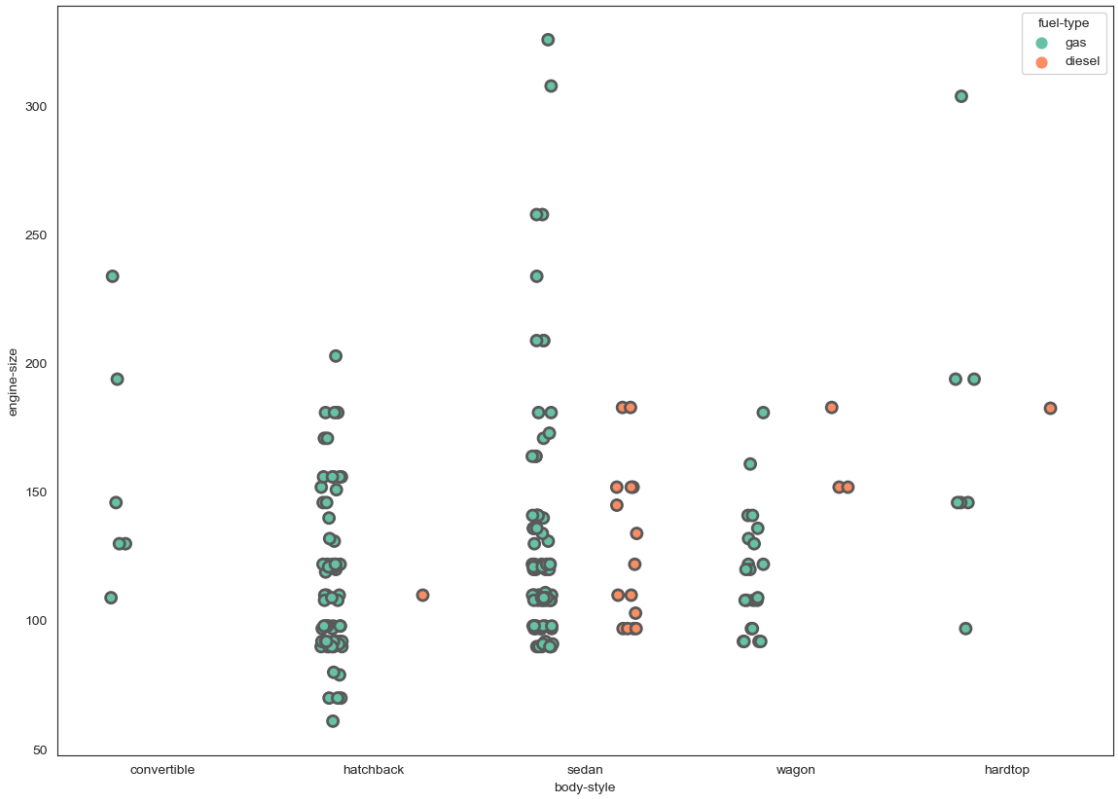


[95]:
```
plt.figure(figsize=(14,10))
sns.stripplot(x=db["body-style"] ,palette="Set2", y =␣
↪db["engine-size"],hue=db["fuel-type"],linewidth=2,size=8)
plt.show()
```



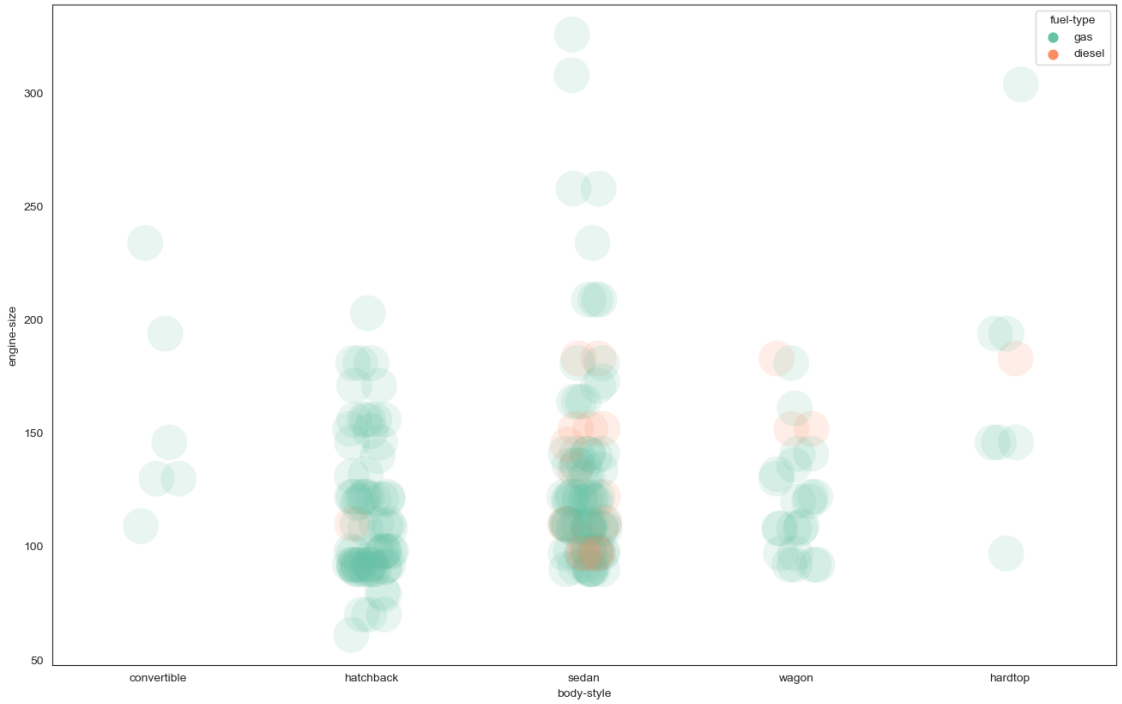
dodge parameter is used to seperate the hue values.

[96]:
```
plt.figure(figsize=(14,10))
sns.stripplot(x=db["body-style"] ,palette="Set2", y =␣
↪db["engine-size"],hue=db["fuel-type"],linewidth=2,size=8,dodge=True)
plt.show()
```

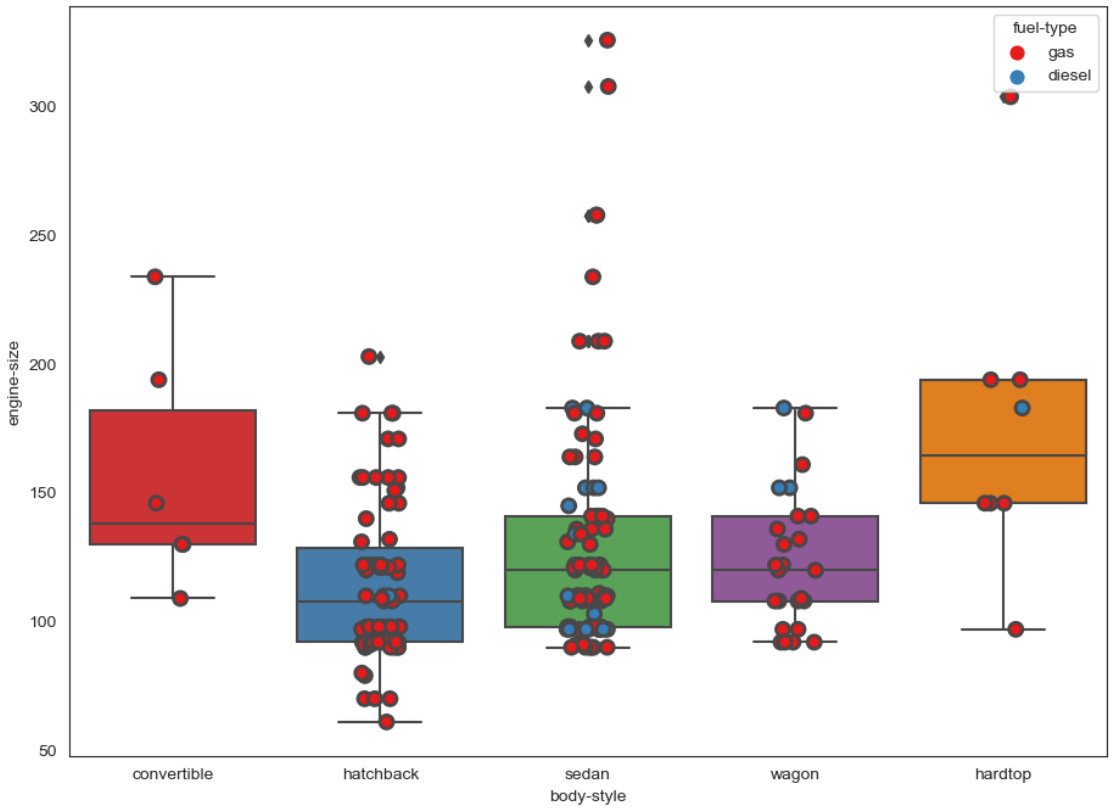


alpha sets the transparency of the markers, which in this case is set to 0.15.

[97]:
```
plt.figure(figsize=(16,10))
sns.stripplot(x=db["body-style"] ,palette="Set2", y =␣
 ↪db["engine-size"],hue=db["fuel-type"],marker = "o",size=30,alpha = .15)
plt.show()
```



[98]:
```
plt.figure(figsize=(11,8))
sns.stripplot(x=db["body-style"] ,palette="Set1", y =␣
 ↪db["engine-size"],hue=db["fuel-type"],jitter=True,␣
 ↪color="black",linewidth=2,size=8)
sns.boxplot(x=db["body-style"] ,palette="Set1", y = db["engine-size"] ,␣
 ↪color='black')
plt.show()
```

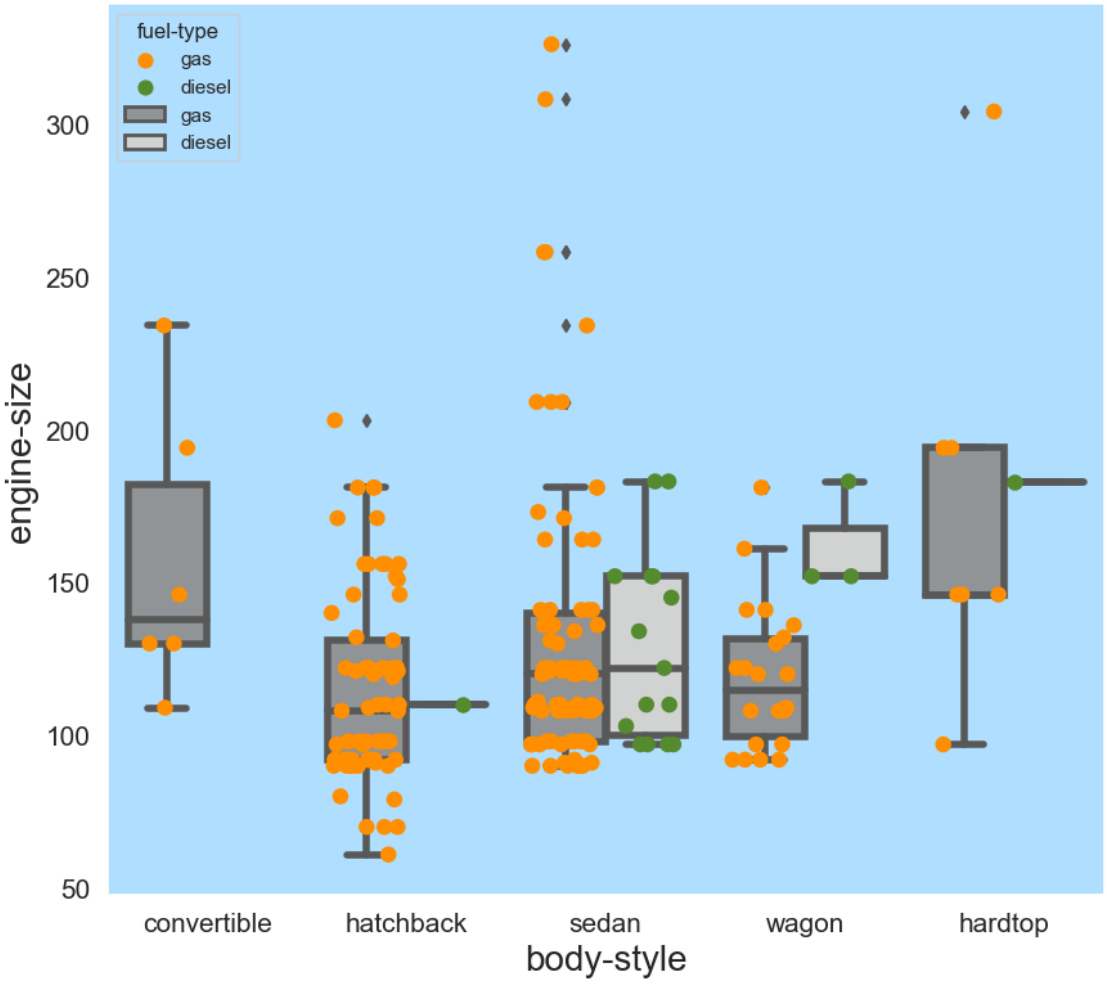


The term **params** is short for "parameters", which are values or arguments that are passed to a function or method in order to modify its behavior or output.

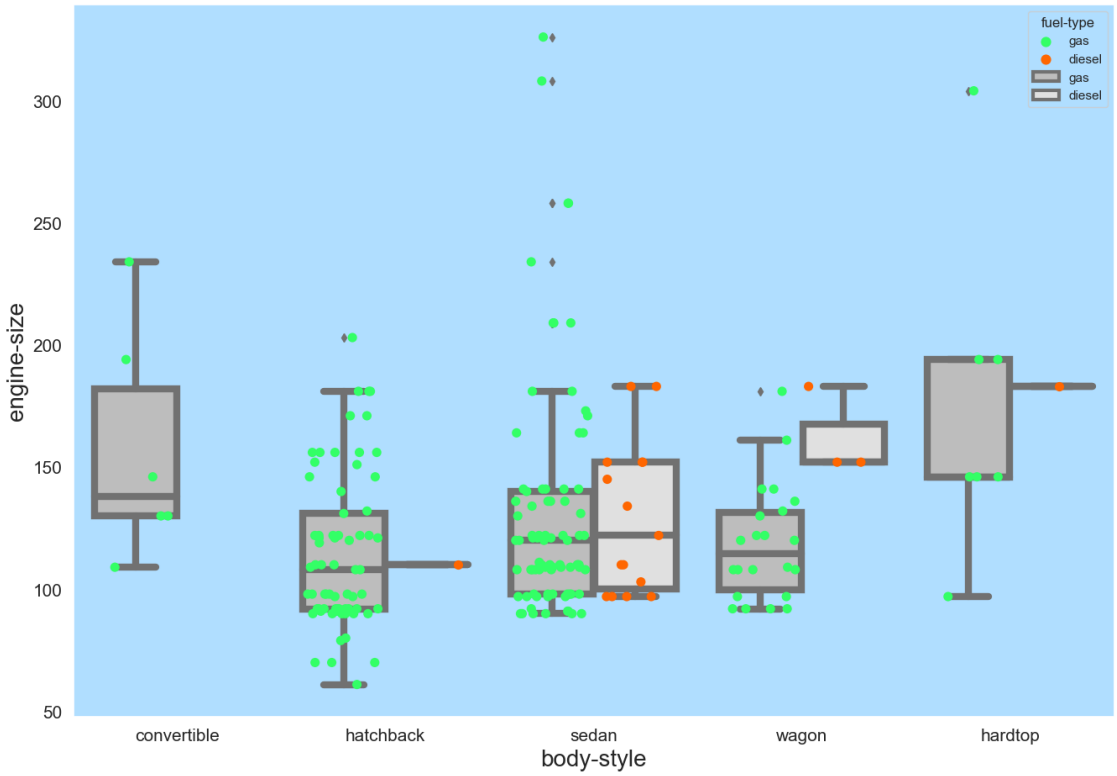
In data visualization libraries like Matplotlib or Seaborn, many plotting functions take a wide range of parameters that can be used to customize the appearance and behavior of the plot. These parameters may include things like the size of the plot, the colors to use, the type of marker to use, the data to plot, and many other options.

For example, in one of the previous plot, the parameters include **x, y, hue, palette, marker, size, and alpha,** all of which modify the behavior or appearance of the strip plot in some way.

[99]:
```
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=db ,x = db["body-style"] ,y = db["engine-size"]␣
 ↪,hue=db["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()
```

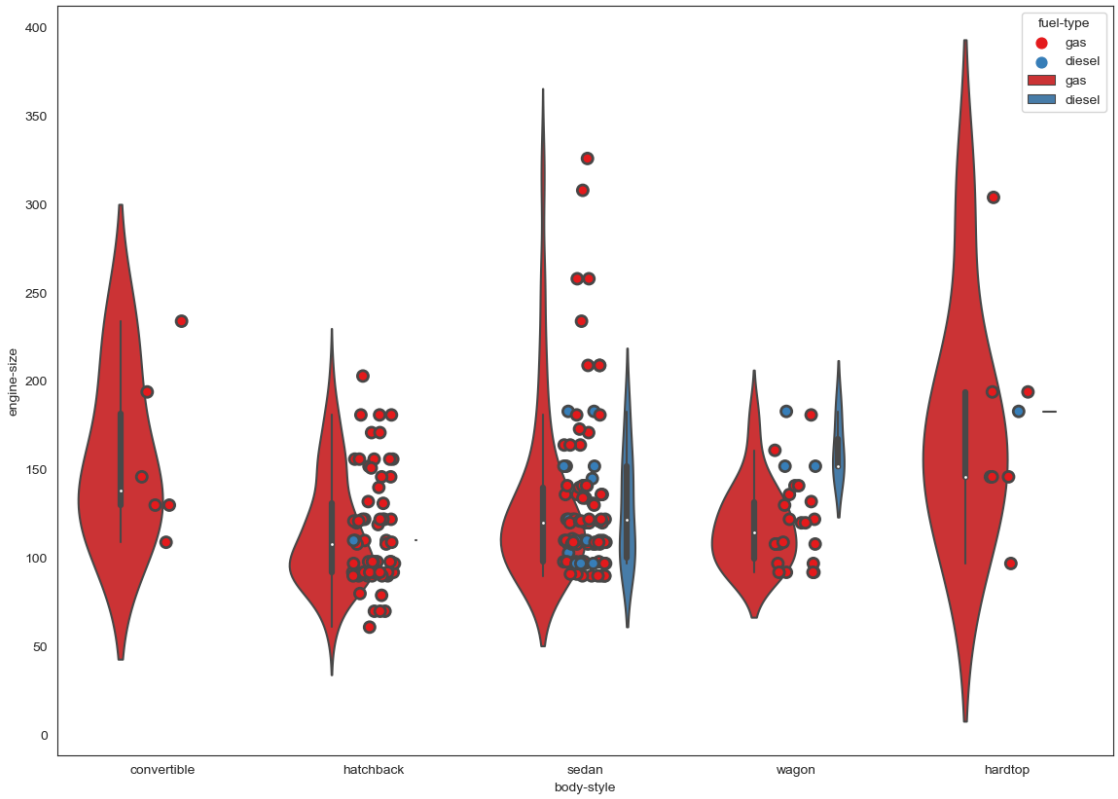


[100]:
```
plt.figure(figsize=(16,11))
sns.set(rc={"axes.facecolor":"#b0deff","axes.grid":False,
'xtick.labelsize':15,'ytick.labelsize':15,
'axes.labelsize':20,'figure.figsize':(20.0, 9.0)})
params = dict(data=db ,x = db["body-style"] ,y = db["engine-size"]␣
 ↪,hue=db["fuel-type"],dodge=True)
sns.stripplot(**params , size=8,jitter=0.
 ↪35,palette=['#33FF66','#FF6600'],edgecolor='black')
sns.boxplot(**params ,palette=['#BDBDBD','#E0E0E0'],linewidth=6)
plt.show()
```



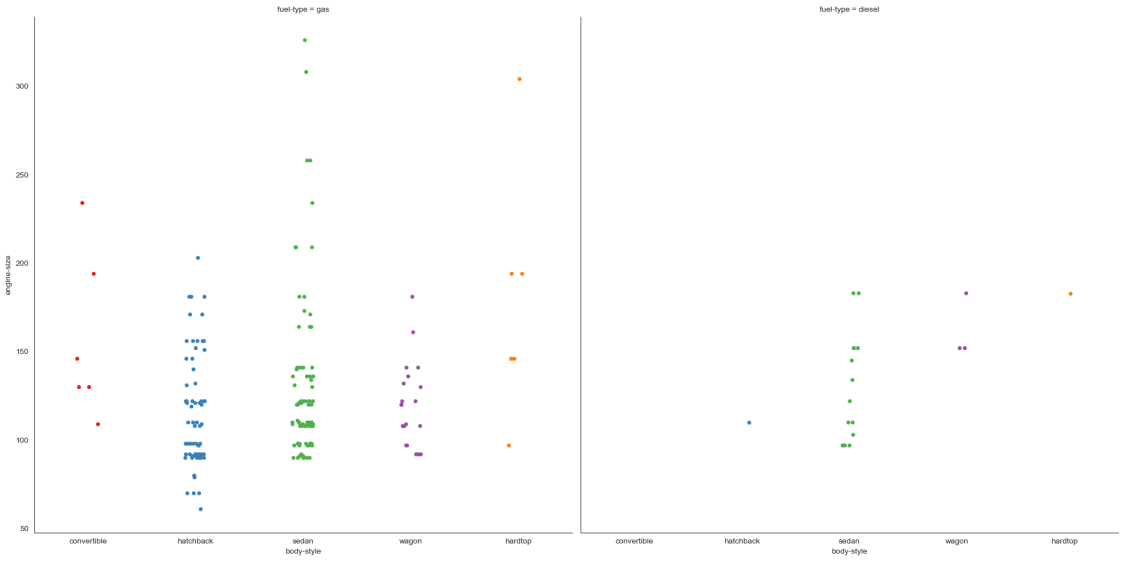
[101]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

[102]:
```
plt.figure(figsize=(14,10))
sns.stripplot(x=db["body-style"] ,palette="Set1", y =␣
 ↪db["engine-size"],hue=db["fuel-type"],jitter=True,␣
 ↪color="black",linewidth=2,size=8)
sns.violinplot(x=db["body-style"] ,palette="Set1", y =␣
 ↪db["engine-size"],hue=db["fuel-type"],scale="count")
plt.show()
```



[103]:
```
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type",␣
 ↪kind="strip",palette="Set1" , height=10,data=db)
plt.show()
```

```
<Figure size 1100x900 with 0 Axes>
```



1.12 Boxplot

A **box plot**, also known as a **box-and-whisker plot**, is a graphical representation of a data set that shows the distribution of the data, its median, and quartiles. It is commonly used in statistics to summarize the distribution of a set of data, particularly in exploratory data analysis and data visualization.

A box plot consists of a rectangular box, which represents the **interquartile range (IQR)**, that is, the middle 50% of the data. The lower edge of the box represents the **first quartile (Q1)**, which is the 25th percentile of the data, and the upper edge of the box represents the **third quartile (Q3)**, which is the 75th percentile of the data. The line inside the box represents the **median**, which is the 50th percentile of the data.

Whiskers extend from the box to show the range of the data, excluding any **outliers**. Outliers are data points that fall outside of the range of the box, and are represented by individual points beyond the whiskers.

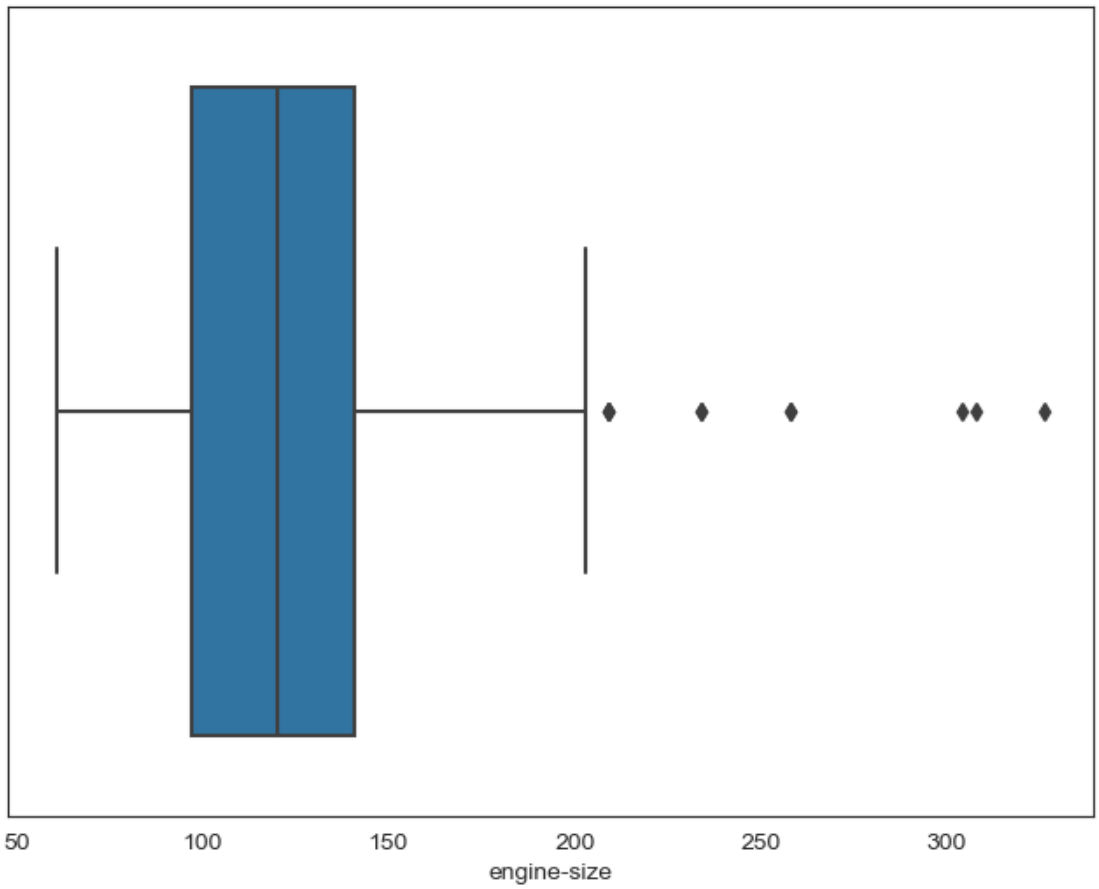
Box plots are useful for identifying the spread of the data, the **skewness** of the distribution, and for comparing distributions across different groups or categories. They can also be used to detect the presence of outliers and to examine the symmetry or asymmetry of the data.

[104]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("white")
```

[105]:
```
plt.figure(figsize=(8,6))
sns.boxplot(db["engine-size"])
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

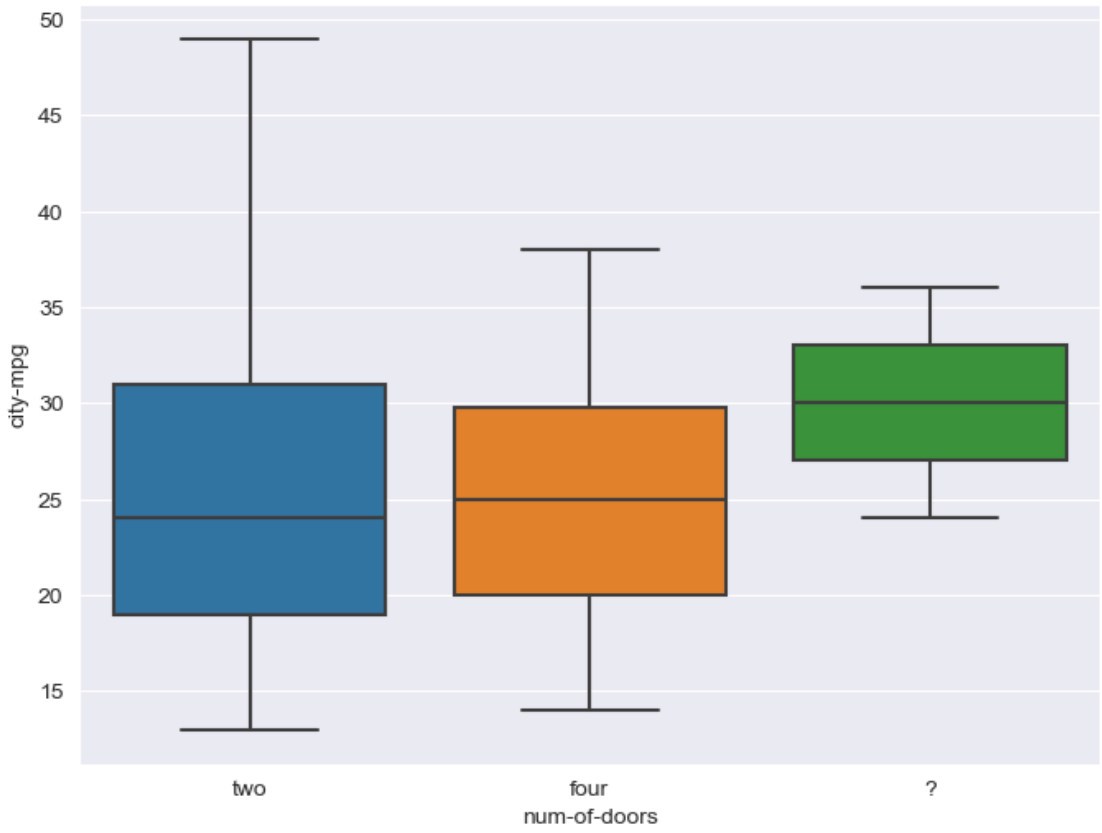
[105]: `<AxesSubplot:xlabel='engine-size'>`



[106]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(8,6))
sns.boxplot(db['num-of-doors'], db['city-mpg'])
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

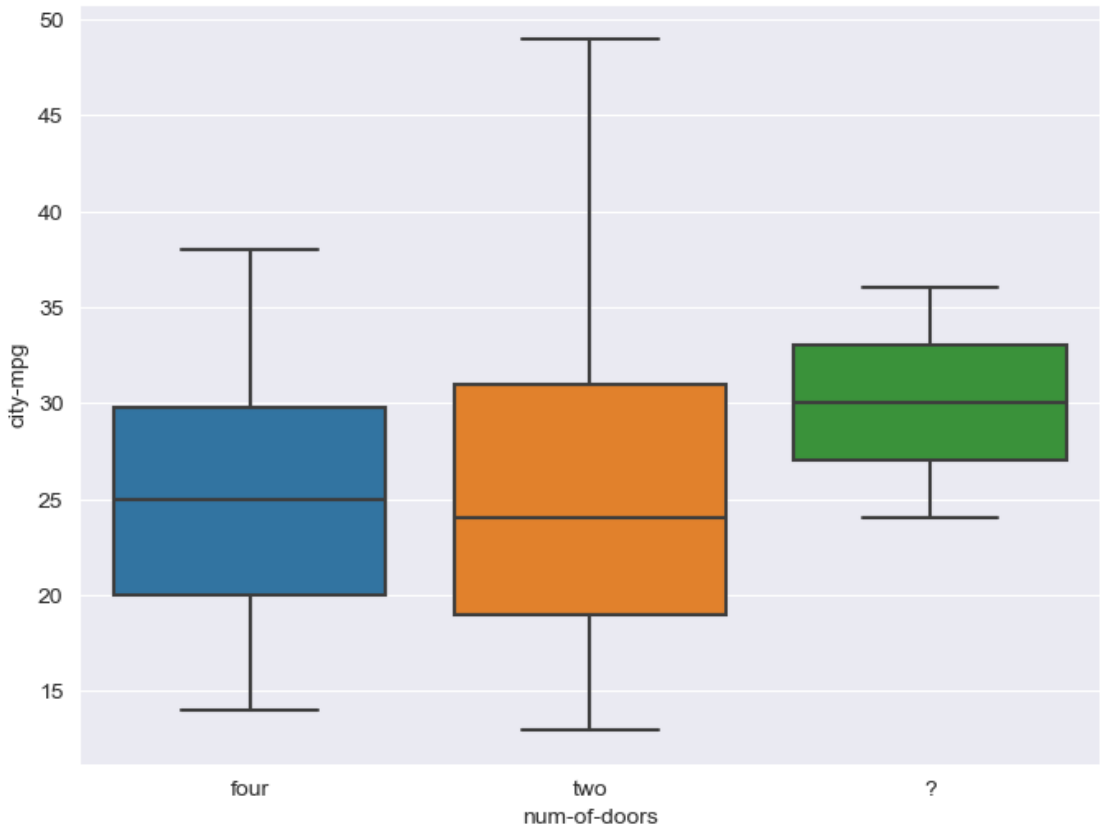
[106]:
```
<AxesSubplot:xlabel='num-of-doors', ylabel='city-mpg'>
```



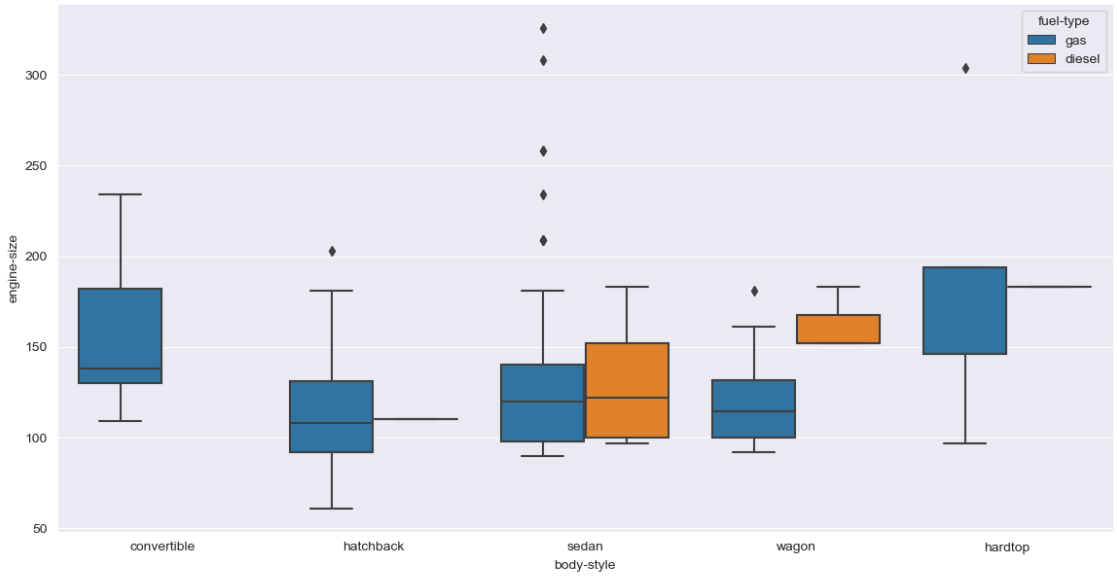
[107]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(8,6))
sns.boxplot(db['num-of-doors'], db['city-mpg'],order = ['four' , 'two','?'])
```

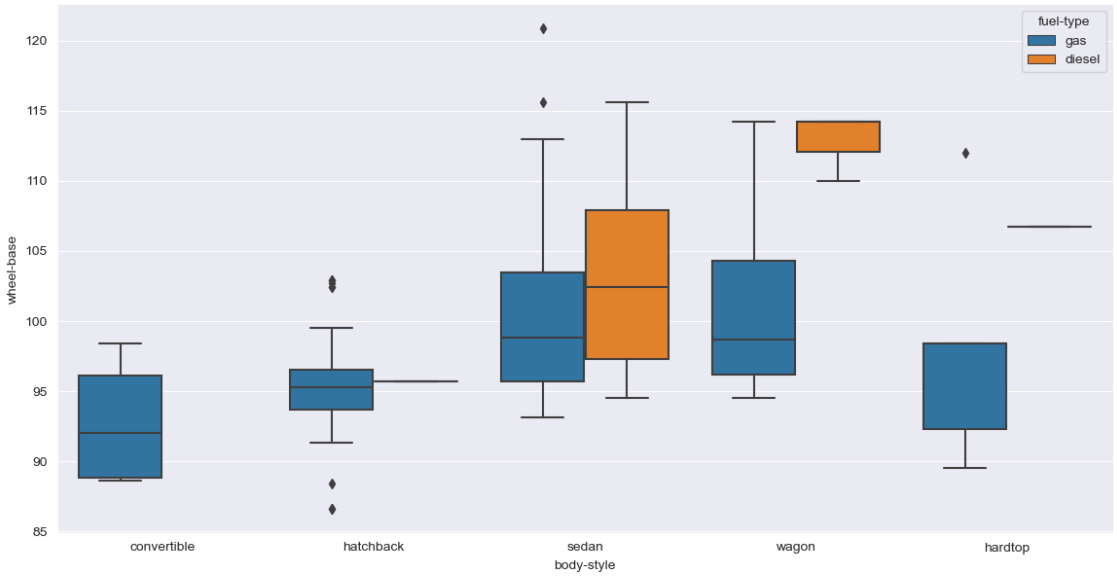
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

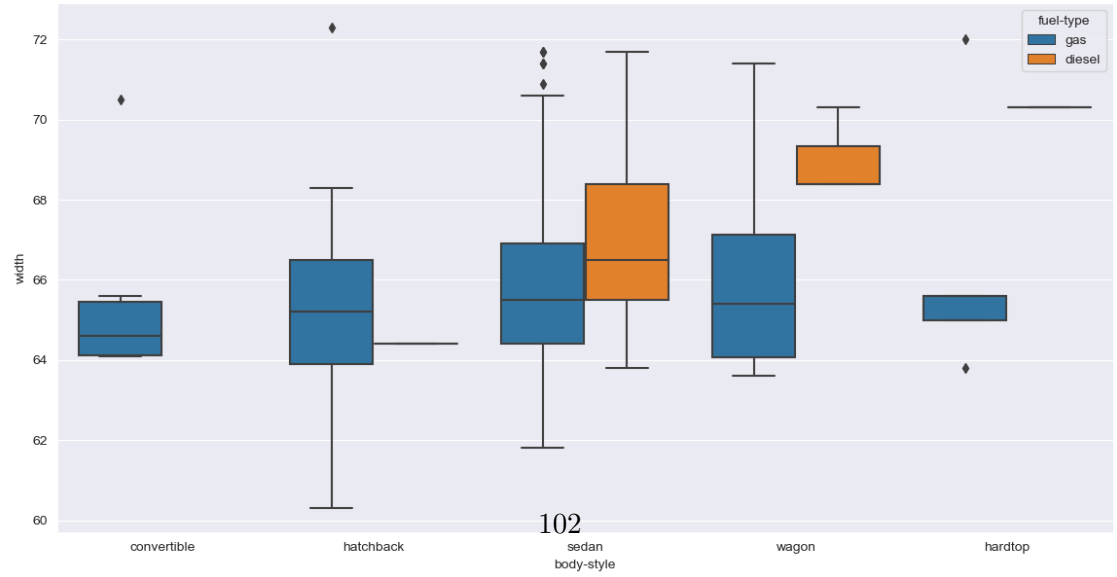
[107]:
```
<AxesSubplot:xlabel='num-of-doors', ylabel='city-mpg'>
```



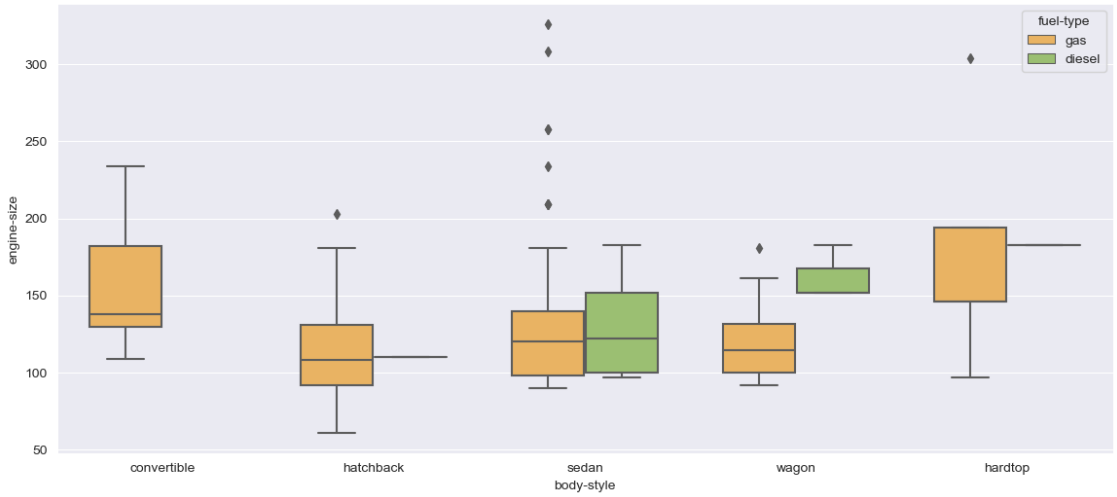
[108]:
```
plt.subplots(figsize = (14,24))
plt.subplot(3,1,1)
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue= db["fuel-type"])
plt.subplot(3,1,2)
sns.boxplot(x=db['body-style'], y=db["wheel-base"], hue=db["fuel-type"])
plt.subplot(3,1,3)
sns.boxplot(x=db['body-style'],y=db["width"] , hue=db["fuel-type"])
plt.show()
```


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


fuel-type
gas
diesel
wheel-base
120
115
110
105
100
95
90
85
convertible
hatchback
sedan
wagon
hardtop
body-style

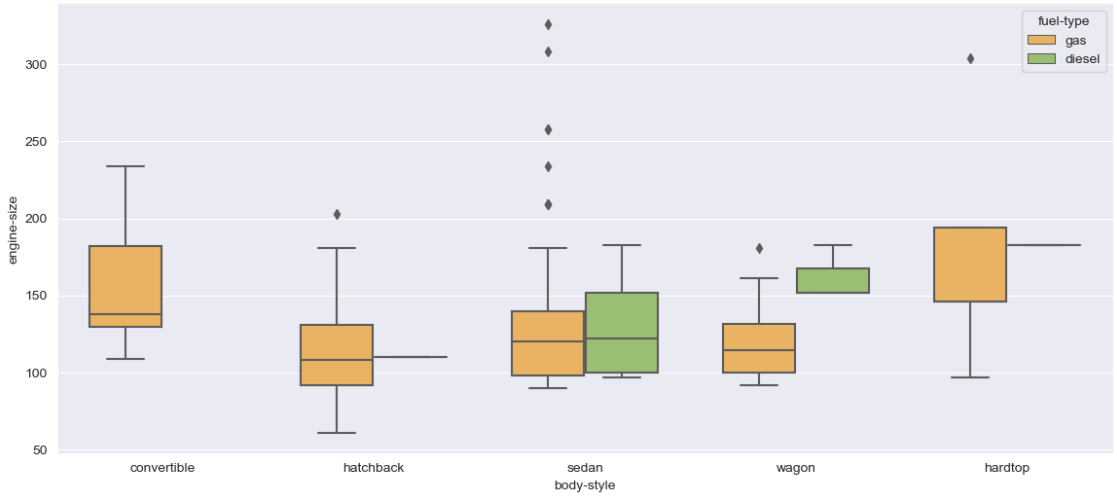

fuel-type
gas
diesel
width
72
70
68
66
64
62
60
convertible
hatchback
sedan
wagon
hardtop
body-style

[109]:
```
plt.figure(figsize = (14,6))
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":'#9CCC65'})
plt.show()
```



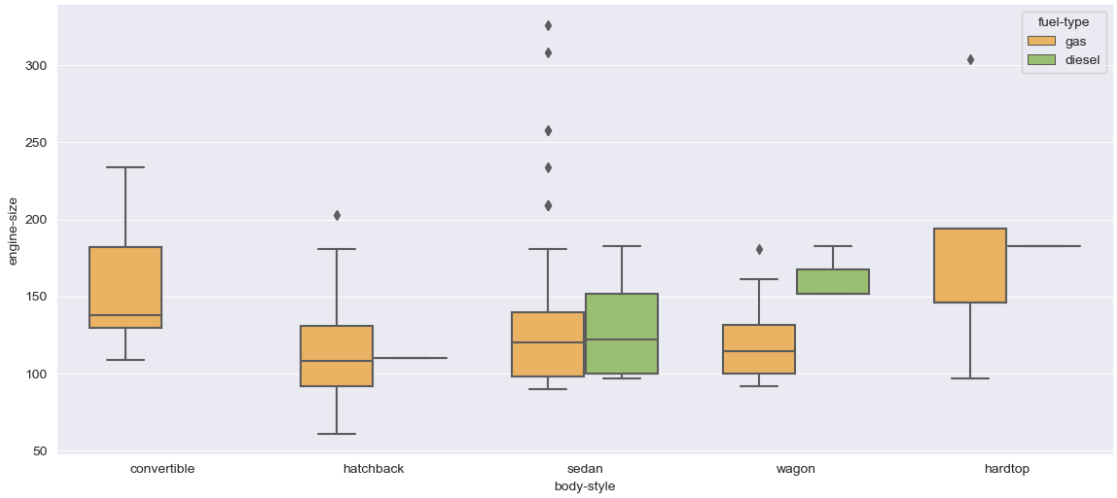
By default, **sns.despine()** removes the top and right spines from the current plot. However, you can customize the behavior of the function by passing various arguments to it. For example, you can specify which spines to remove, change the offset of the spines, and set the limits of the plot.

[110]:
```
plt.figure(figsize = (14,6))
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":'#9CCC65'})
sns.despine()
plt.show()
```



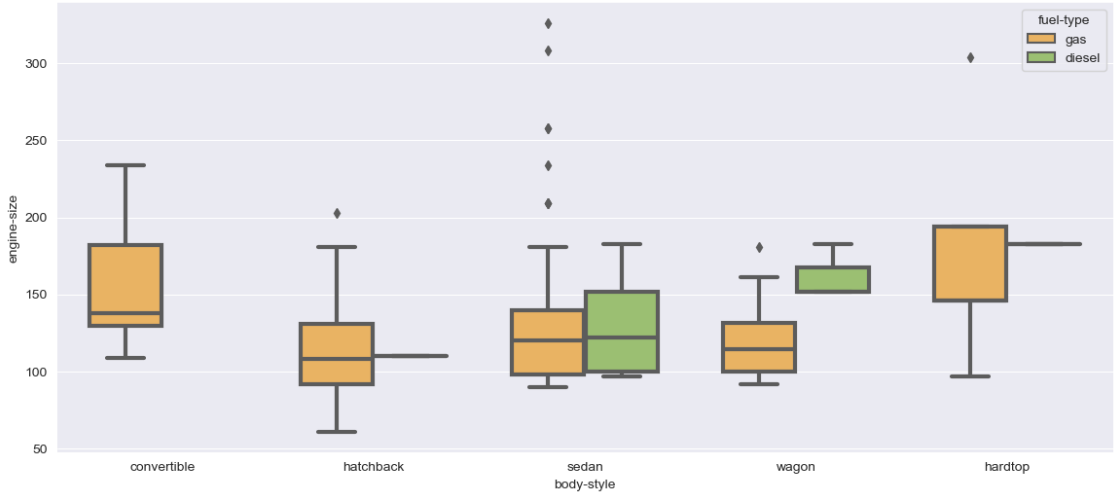
The **left=True** argument in **sns.despine(left=True)** specifies that the left spine of the plot should be removed. If you set **left=False**, the left spine will not be removed.

[111]:
```
plt.figure(figsize = (14,6))
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":'#9CCC65'})
sns.despine(left=True)
plt.show()
```



[112]:
```
plt.figure(figsize = (14,6))
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":
 ↪'#9CCC65'},linewidth = 3)
```

```
sns.despine(left=True)
plt.show()
```



The notch in a box plot is used to provide an estimate of the median uncertainty and compare the medians of two or more groups. The notch represents a confidence interval around the median, and its width is determined by the **sample size** and the **interquartile range (IQR)**.

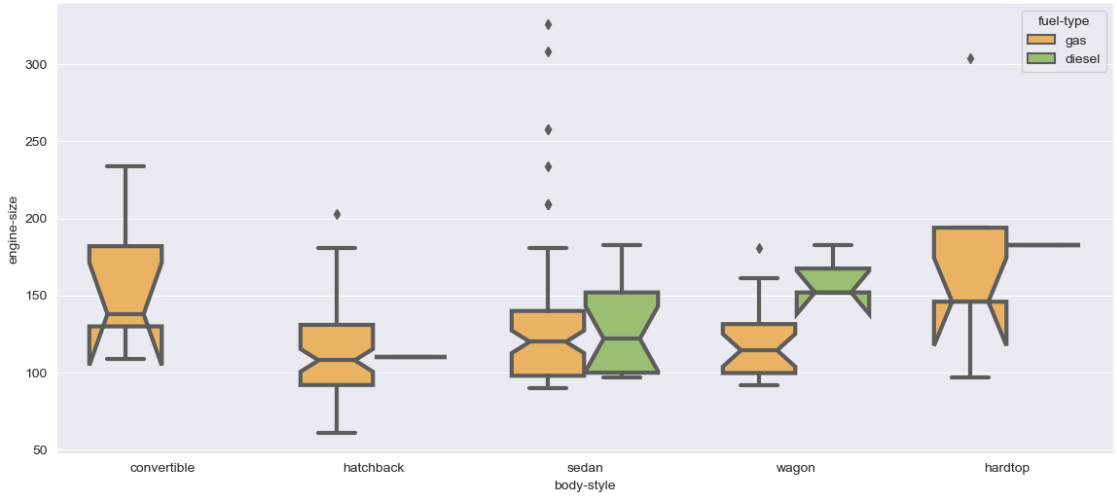
The notch is a small indentation located at the center of the box in a box plot. The upper and lower boundaries of the notch are calculated as follows:

- Calculate the median of the data.
- Divide the data into two halves at the median.
- Calculate the median of each half, which gives you two values.
- Calculate the 95% confidence interval of the difference between the two medians.
- Draw the notch at the median of the data and extend the boundaries of the notch to the upper and lower limits of the confidence interval.

If the notches of two box plots do not overlap, it suggests that the medians of the two groups are significantly different at the 95% confidence level. If the notches overlap, it does not necessarily mean that the medians are not different, but it does indicate that there is not enough evidence to conclude that the medians are different at the 95% confidence level.

Therefore, the purpose of the notch in a box plot is to provide a quick and easy way to compare the medians of two or more groups and determine if there is a significant difference between them.

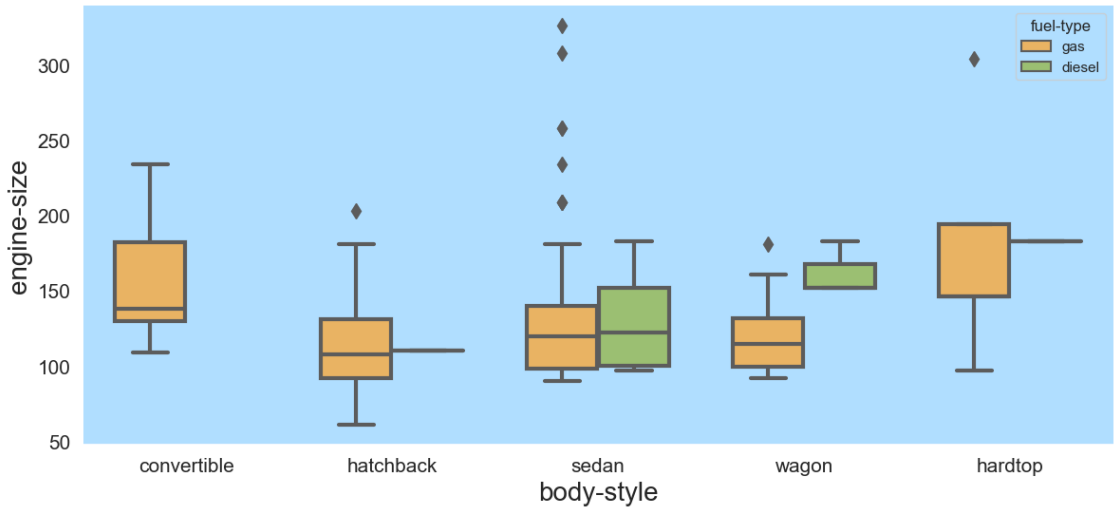
[113]:
```
plt.figure(figsize = (14,6))
sns.boxplot(x= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":
 ↪'#9CCC65'},linewidth = 3,notch=True)
sns.despine(left=True)
plt.show()
```



fliersize=8 is an argument that can be passed to the boxplot() function in Python to specify the size of the markers that represent outliers in the plot.

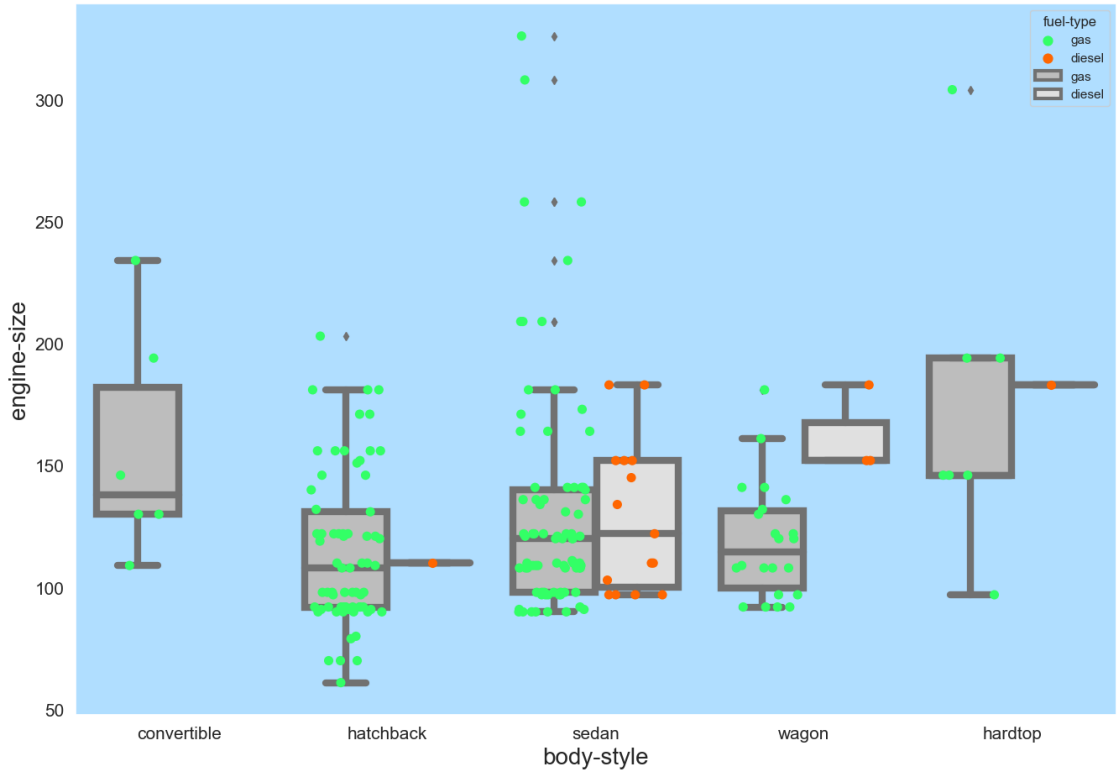
In a box plot, outliers are data points that fall outside the whiskers of the plot. By default, these outliers are represented by small circles or dots. The fliersize argument allows you to adjust the size of these markers to make them more visible or less prominent in the plot.

[114]:
```
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= db['body-style'] , y= db['engine-size'] ,hue=␣
 ↪db["fuel-type"],width=.7,palette= {"gas":'#FFB74D' , "diesel":
 ↪'#9CCC65'},linewidth = 3,fliersize=8)
sns.despine(left=True)
plt.show()
```



[115]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

[116]:
```
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=db ,x = db["body-style"] ,y = db["engine-size"],␣
 ↪,hue=db["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()
```

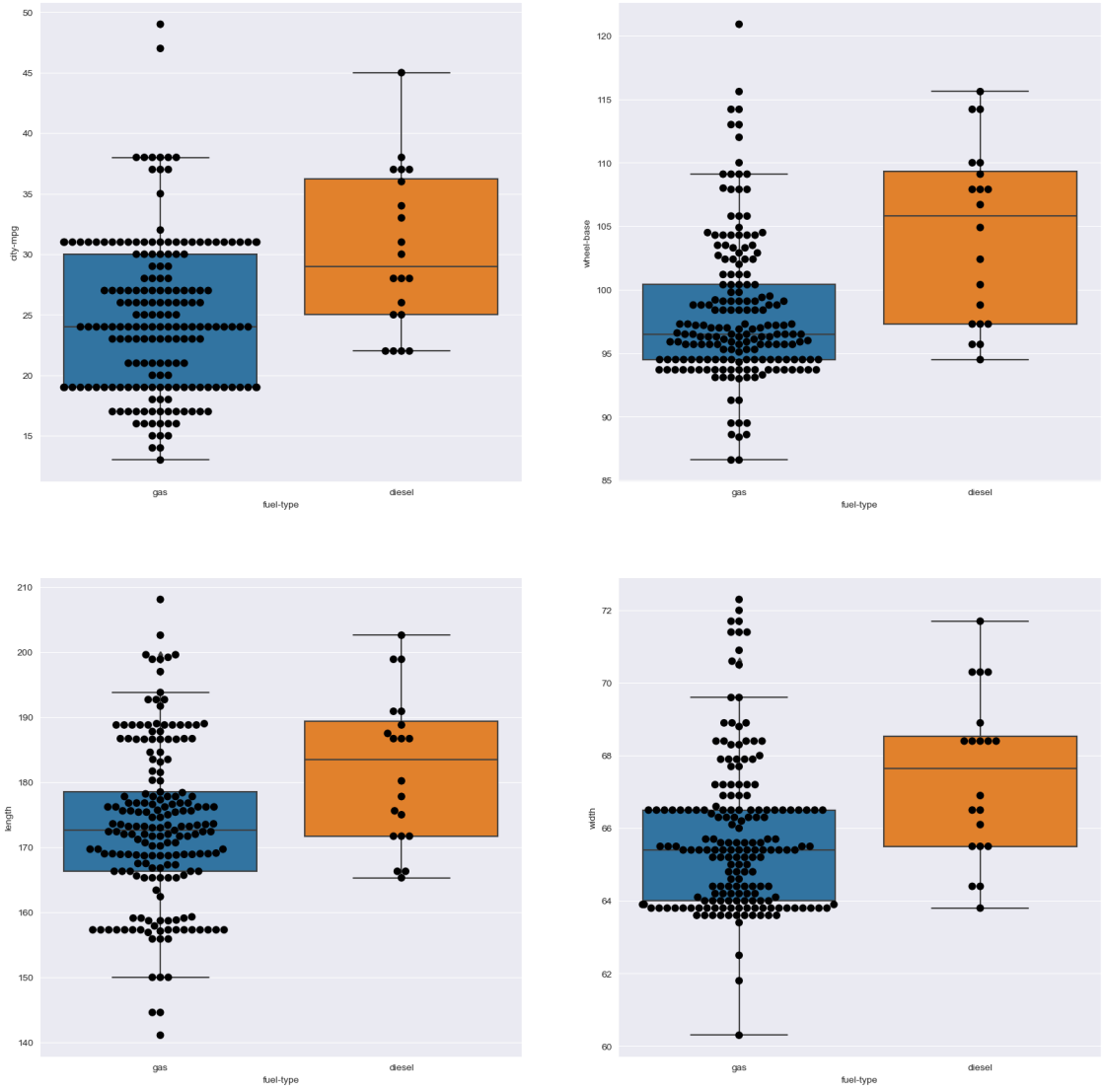


[117]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("darkgrid")
```

[118]:
```
fig1 , axes = plt.subplots(nrows=2,ncols=2 , figsize = (20,20))
sns.swarmplot(db['fuel-type'], db['city-mpg'], ax = axes[0,0], size=8,␣
 ↪color="black")
sns.boxplot(db['fuel-type'], db['city-mpg'], ax = axes[0,0])
sns.swarmplot(db['fuel-type'] , db['wheel-base'] ,ax = axes[0,1], size=8,␣
 ↪color="black")
sns.boxplot(db['fuel-type'] , db['wheel-base'] ,ax = axes[0,1])
sns.swarmplot(db['fuel-type'] ,db['length'] , ax = axes[1,0] , size=8,␣
 ↪color="black")
sns.boxplot(db['fuel-type'] ,db['length'] , ax = axes[1,0])
sns.swarmplot(db['fuel-type'] , db['width'] , ax = axes[1,1] , size=8,␣
 ↪color="black")
sns.boxplot(db['fuel-type'] , db['width'] , ax = axes[1,1])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
```

```
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



1.13 Boxen Plot

A **Boxen Plot**, also known as a **Letter Value Plot** or a **Caterpillar Plot**, is a type of data visualization that was introduced by **Hadley Wickham** in 2017 as an extension of a box plot. Boxen plots are similar to box plots but provide more detailed information about the distribution of the data.

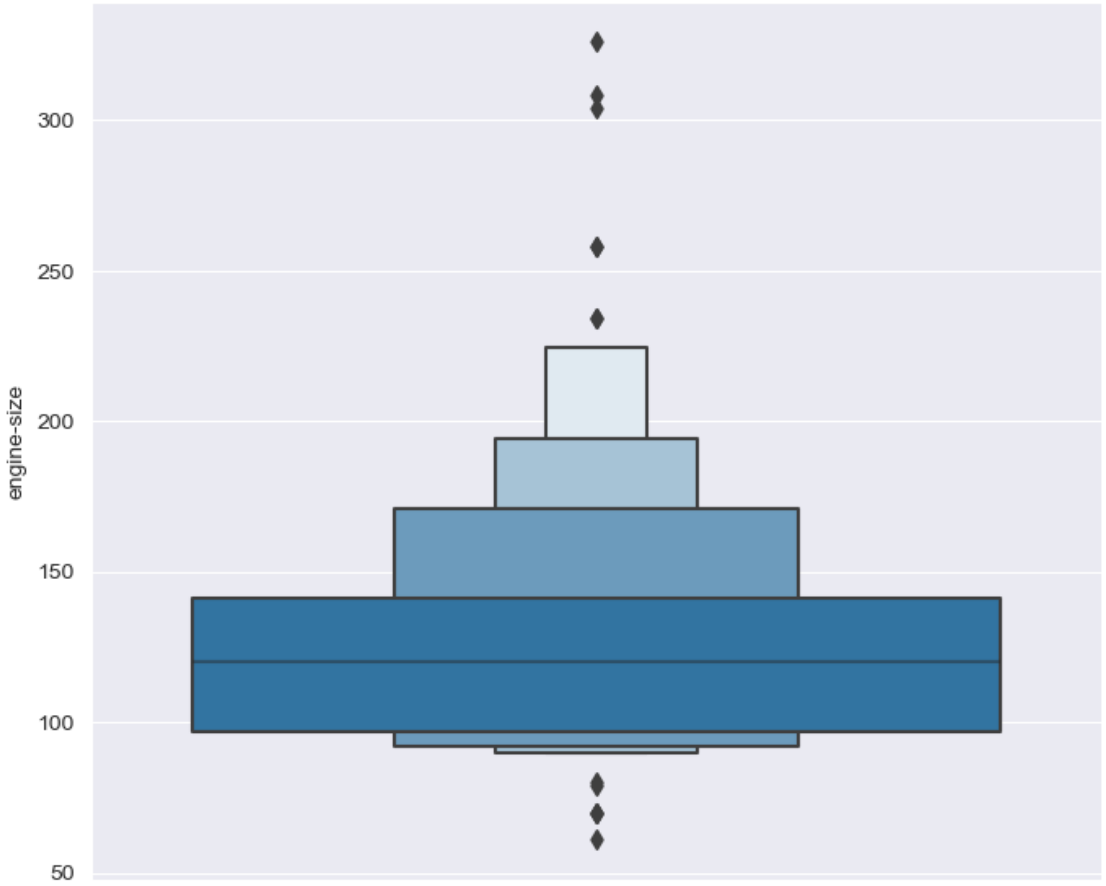
In a boxen plot, the horizontal axis represents the variable being plotted, and the vertical axis represents the values of the variable. The plot is divided into boxes, each of which represents a range of values. The boxes are stacked on top of each other to form a "caterpillar" shape, hence the name "Caterpillar Plot".

The height of each box is proportional to the number of observations within that range, and the width of the box is proportional to the density of observations in that range. This makes the boxen plot more informative than a box plot, especially when dealing with large datasets or highly variable data.

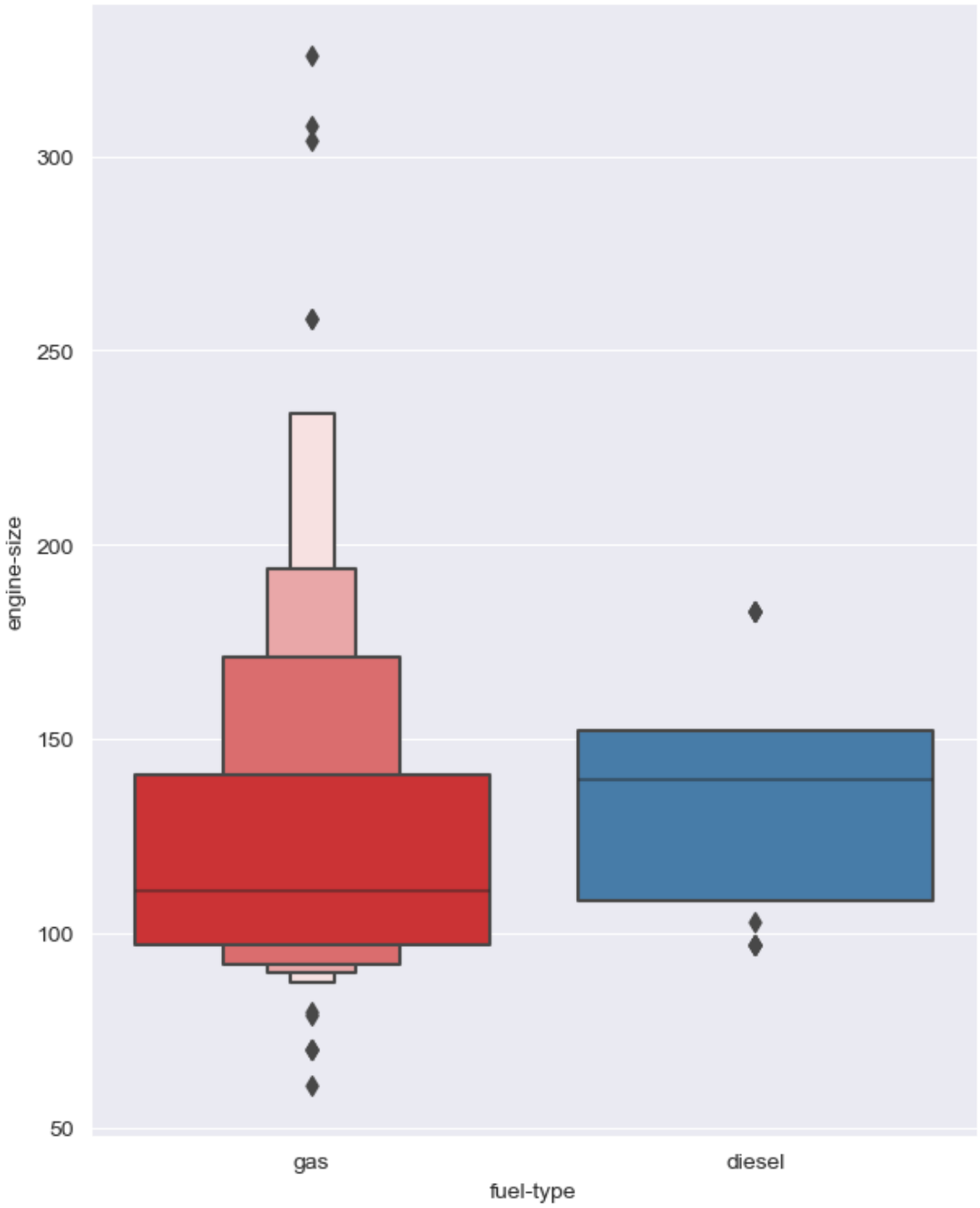
In addition to the boxes, a boxen plot may also include whiskers, which indicate the range of the data beyond the boxes. Outliers may also be displayed as individual points, similar to a box plot.

[119]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set_style("darkgrid")
```

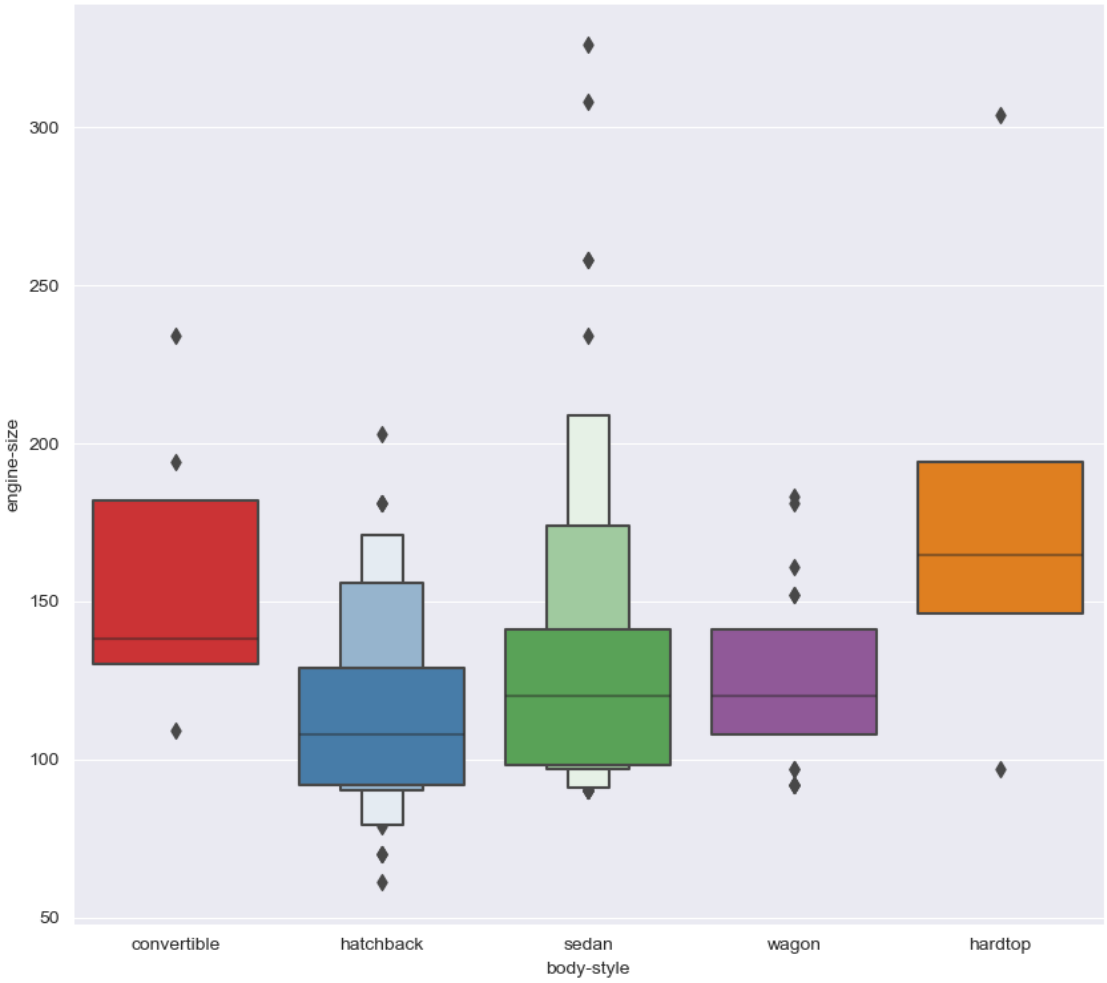
[120]:
```
plt.figure(figsize=(8,7))
sns.boxenplot(y=db["engine-size"])
plt.show()
```



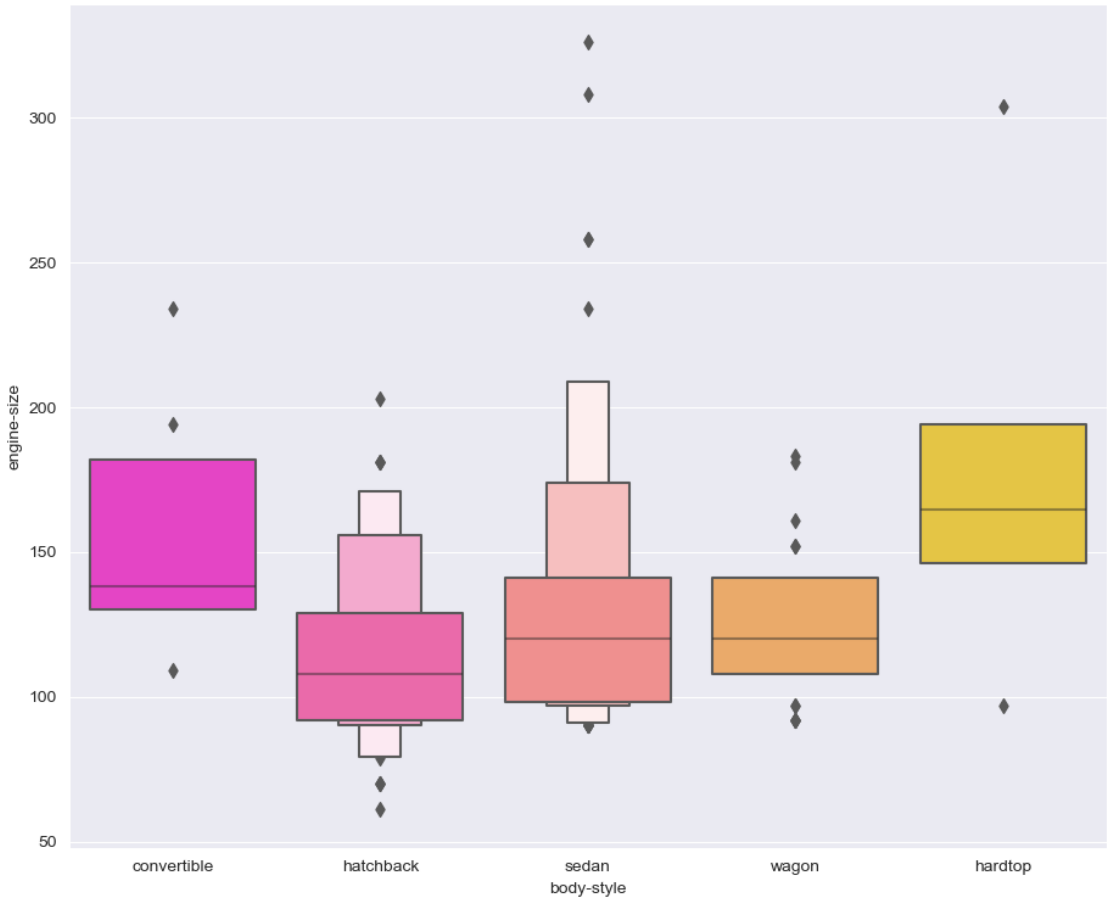
[121]:
```
plt.figure(figsize=(7,9))
sns.boxenplot(x=db["fuel-type"] , y = db["engine-size"] ,palette="Set1")
plt.show()
```



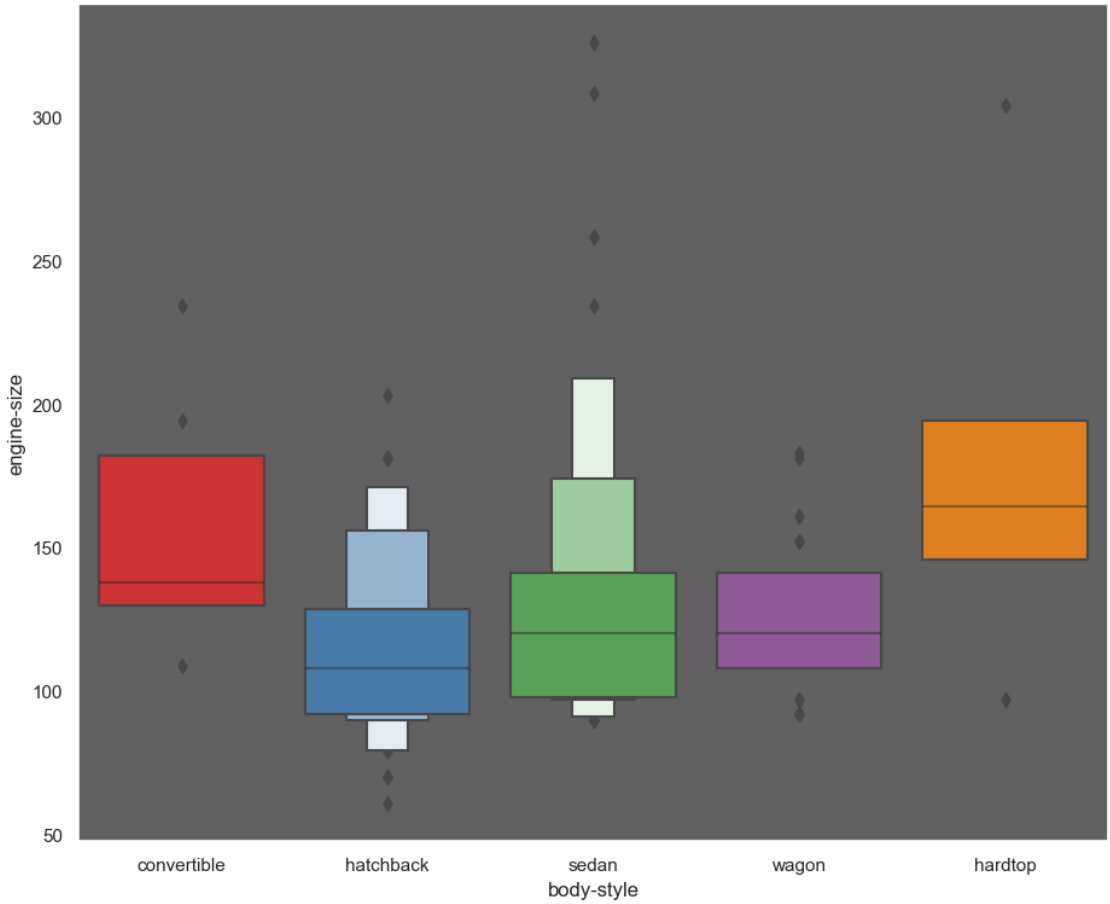
[122]:
```
plt.figure(figsize=(10,9))
sns.boxenplot(x=db["body-style"] , y = db["engine-size"] ,palette="Set1")
plt.show()
```



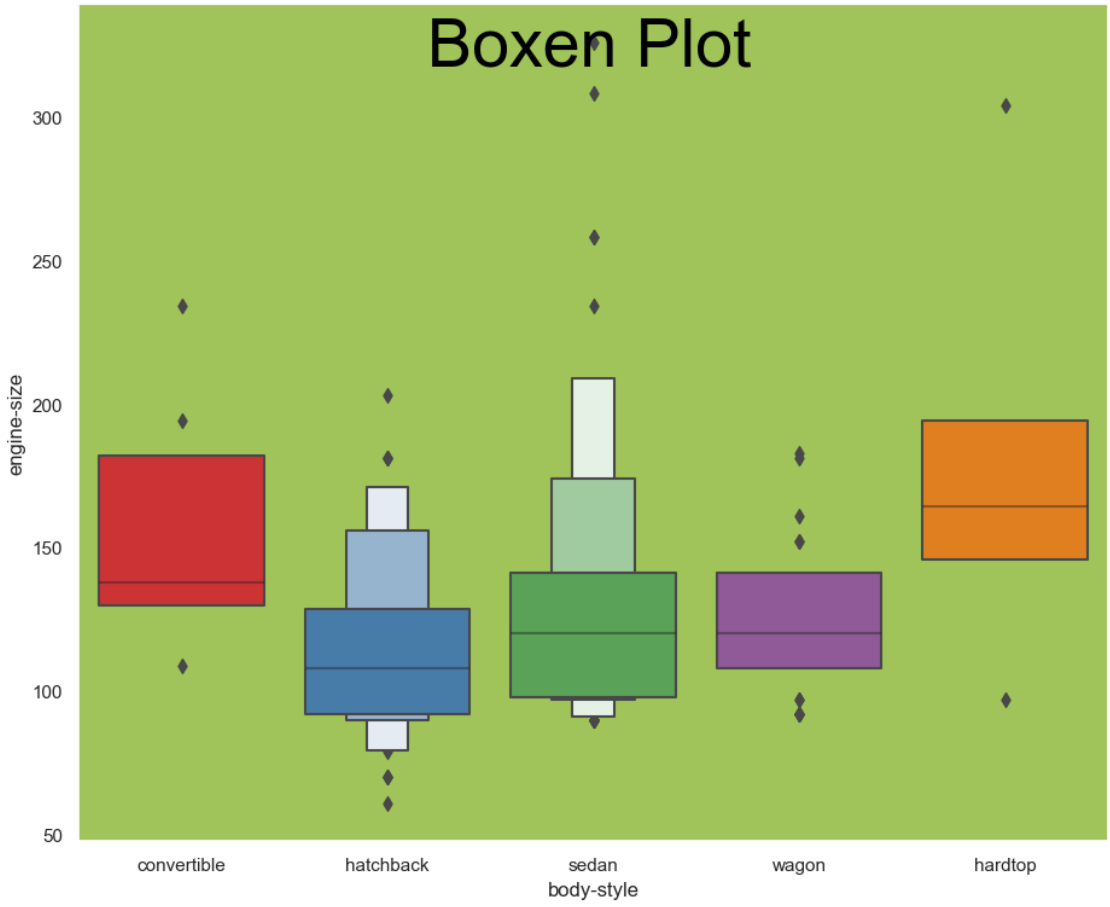
[123]:
```
plt.figure(figsize=(11,9))
sns.boxenplot(x=db["body-style"] , y = db["engine-size"],palette="spring")
plt.show()
```



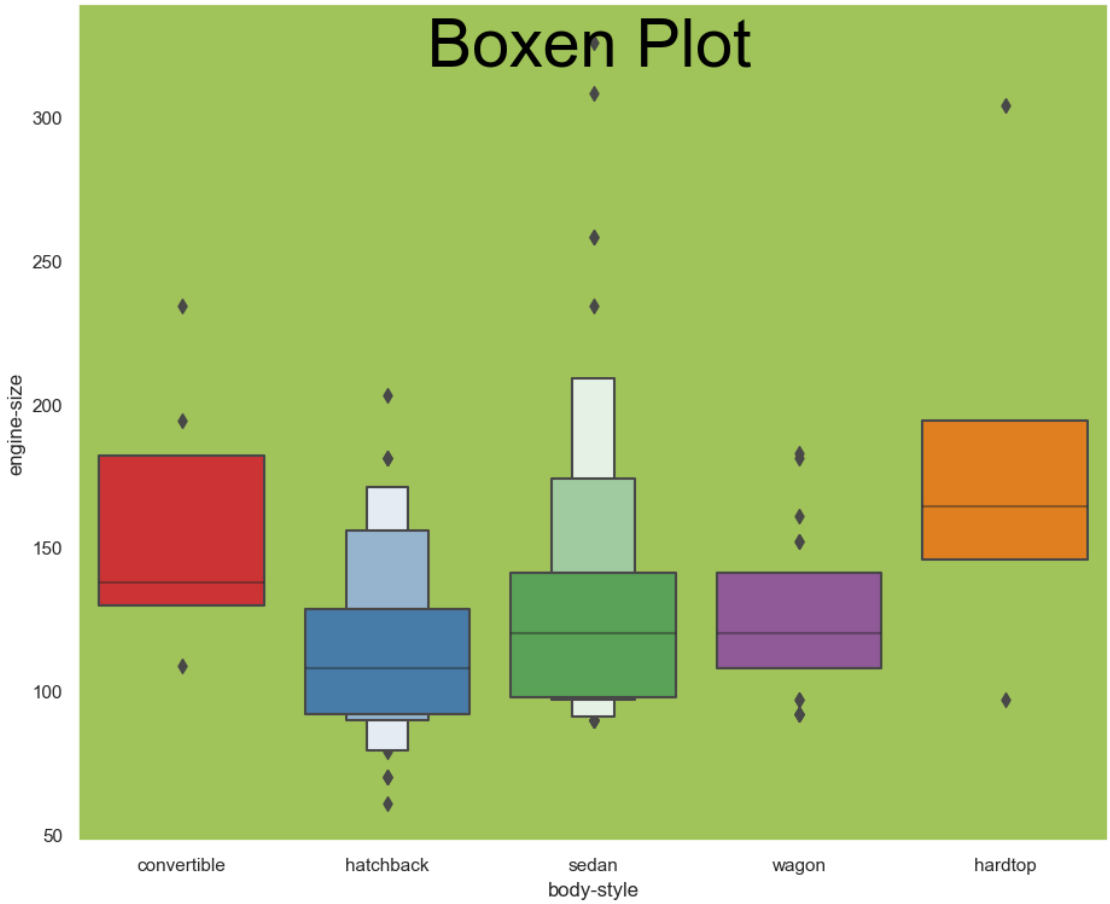
[124]:
```
sns.set(rc={"axes.facecolor":"#616161" , "axes.grid" : False})
plt.figure(figsize=(11,9))
sns.boxenplot(x=db["body-style"] , y = db["engine-size"],palette="Set1")
plt.show()
```



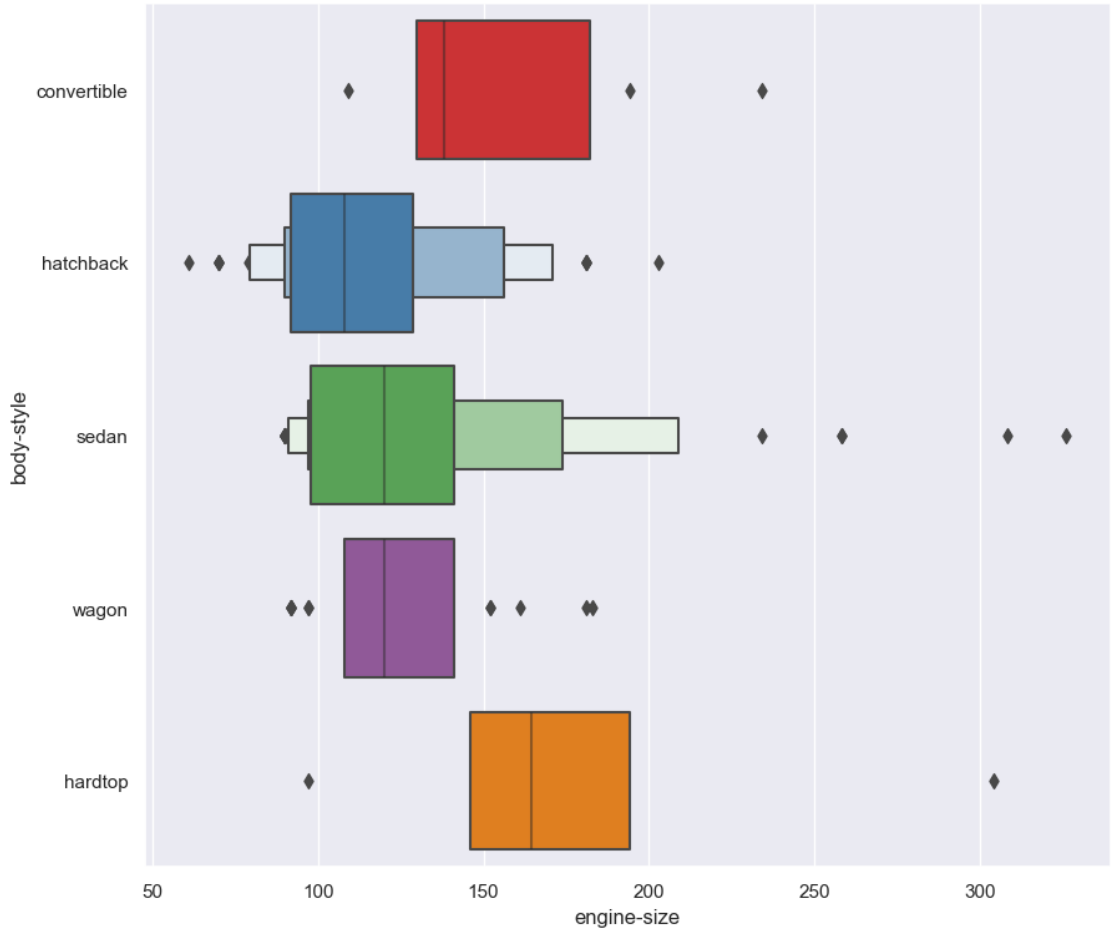
[125]:
```
sns.set(rc={"axes.facecolor":"#a1c45a" , "axes.grid" : False})
plt.figure(figsize=(11,9))
plt.gcf().text(.51, .84, "Boxen Plot", fontsize = 40, color='Black'␣
↪,ha='center', va='center')
sns.boxenplot(x=db["body-style"] , y = db["engine-size"],palette="Set1")
plt.show()
```



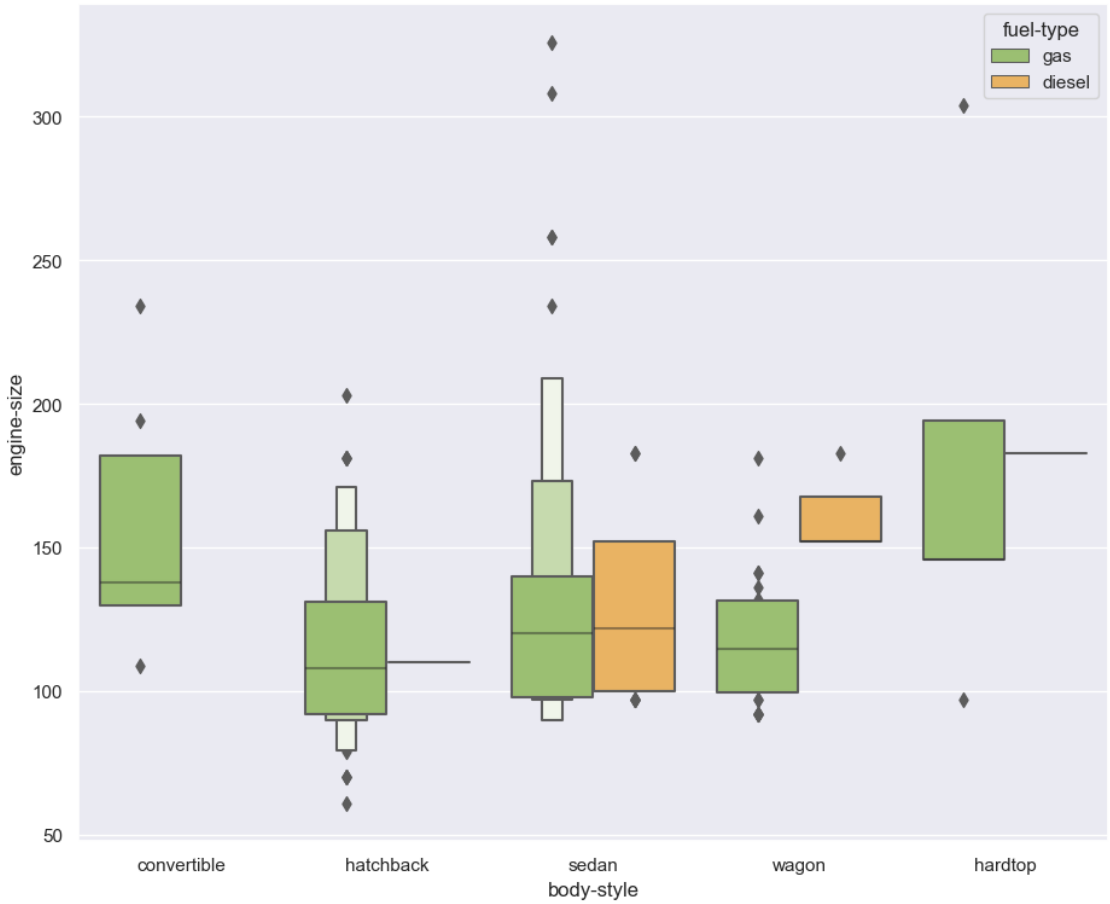
[126]:
```
sns.set(rc={"axes.facecolor":"#a1c45a" , "axes.grid" : False})
plt.figure(figsize=(11,9))
plt.gcf().text(.51, .84, "Boxen Plot", fontsize = 40, color='Black'␣
↪,ha='center', va='center')
sns.boxenplot(x=db["body-style"] , y = db["engine-size"],palette="Set1")
plt.show()
```



[127]:
```
sns.set_style("darkgrid")
plt.figure(figsize=(10,9))
sns.boxenplot(x = db["engine-size"] ,y= db["body-style"] ,palette="Set1")
plt.show()
```

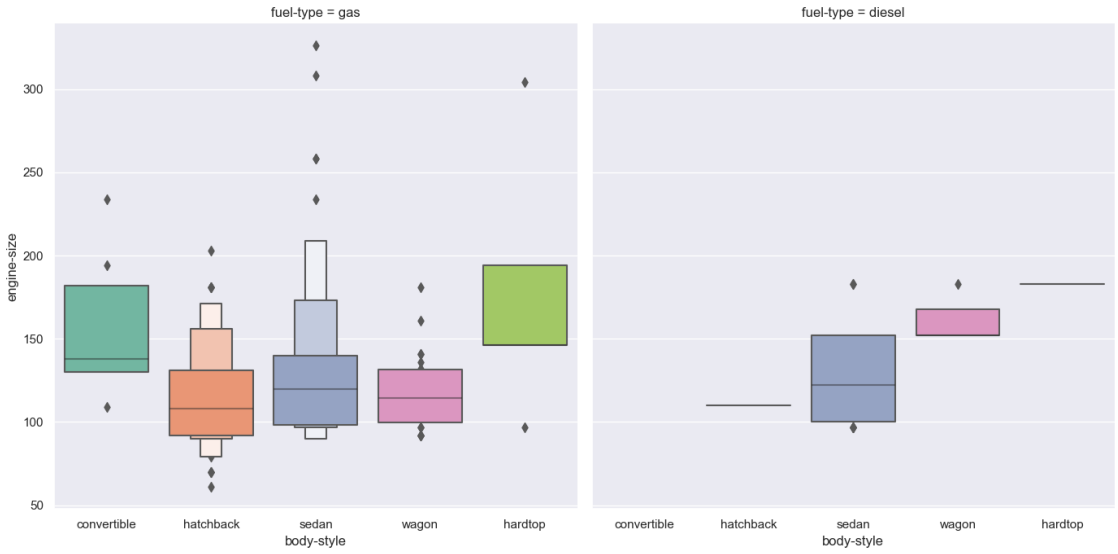


[128]:
```
plt.figure(figsize=(11,9))
sns.boxenplot(x=db["body-style"] , y = db["engine-size"], hue=db["fuel-type"],␣
↪palette={"diesel":'#FFB74D' , "gas":'#9CCC65'})
plt.show()
```



[129]:
```
plt.figure(figsize=(11,9))
sns.catplot(x="body-style" , y = "engine-size", col="fuel-type",␣
  ↪kind="boxen",palette="Set2" , height=7,data=db)
plt.show()
```

```
<Figure size 1100x900 with 0 Axes>
```



1.14 Pair plot

A **pair plot** is a type of data visualization that shows the **pairwise** relationships between variables in a dataset. It is a useful tool for exploring the correlations between multiple variables and identifying patterns in the data.

Pair plots typically consist of a matrix of scatter plots, with each scatter plot showing the relationship between two variables. The diagonal of the matrix shows the distribution of each variable.

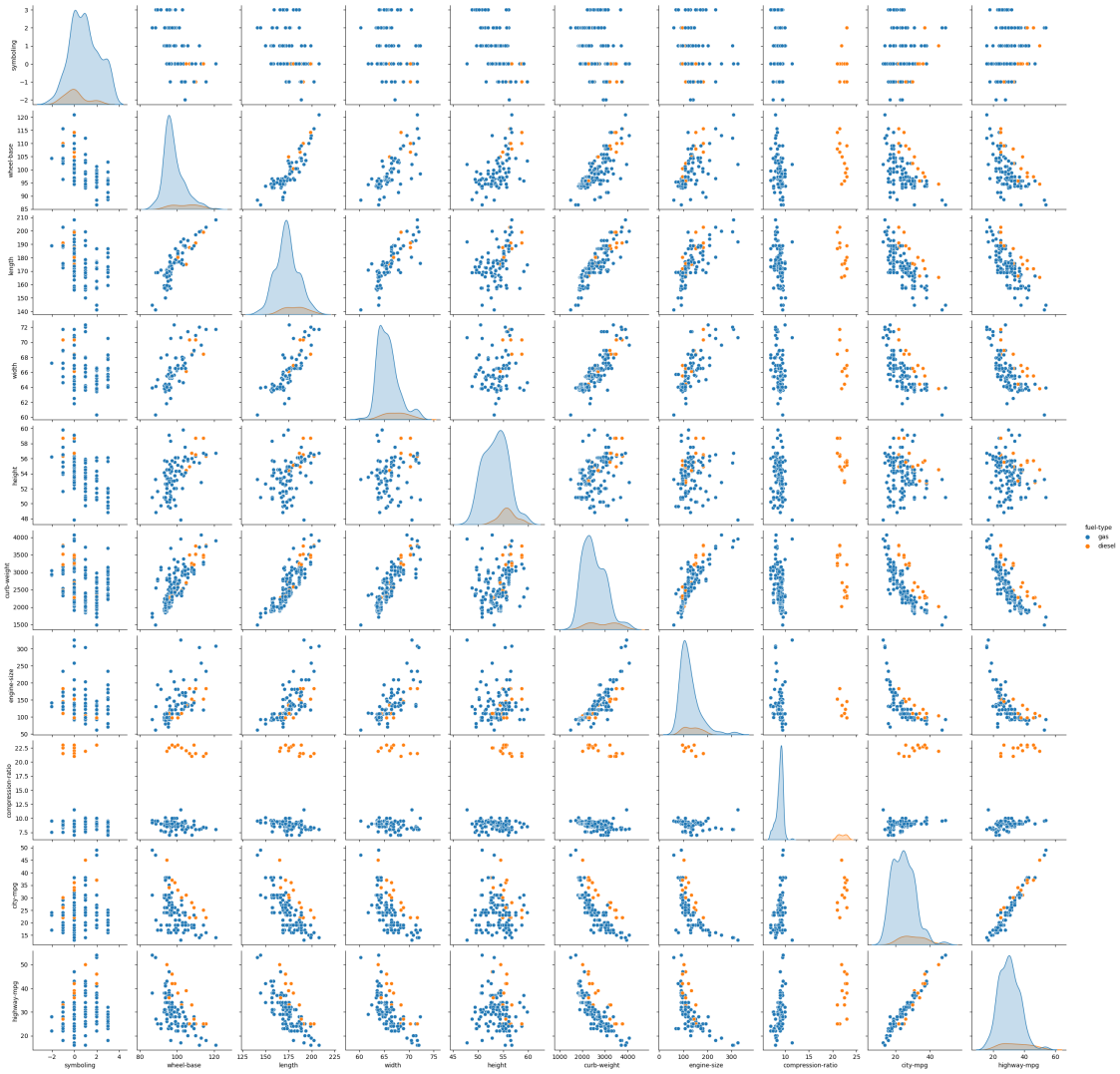
Pair plots are particularly useful for identifying correlations between multiple variables, and for identifying patterns that may not be immediately obvious from examining individual variables in isolation. They can also be used to identify outliers, clusters, and other features in the data.

Pair plots are commonly used in **exploratory data analysis (EDA)**, as they can help analysts gain a better understanding of the structure of the data before applying more advanced statistical techniques. They are also used in machine learning to visualize the relationships between input variables and target variables.

[130]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

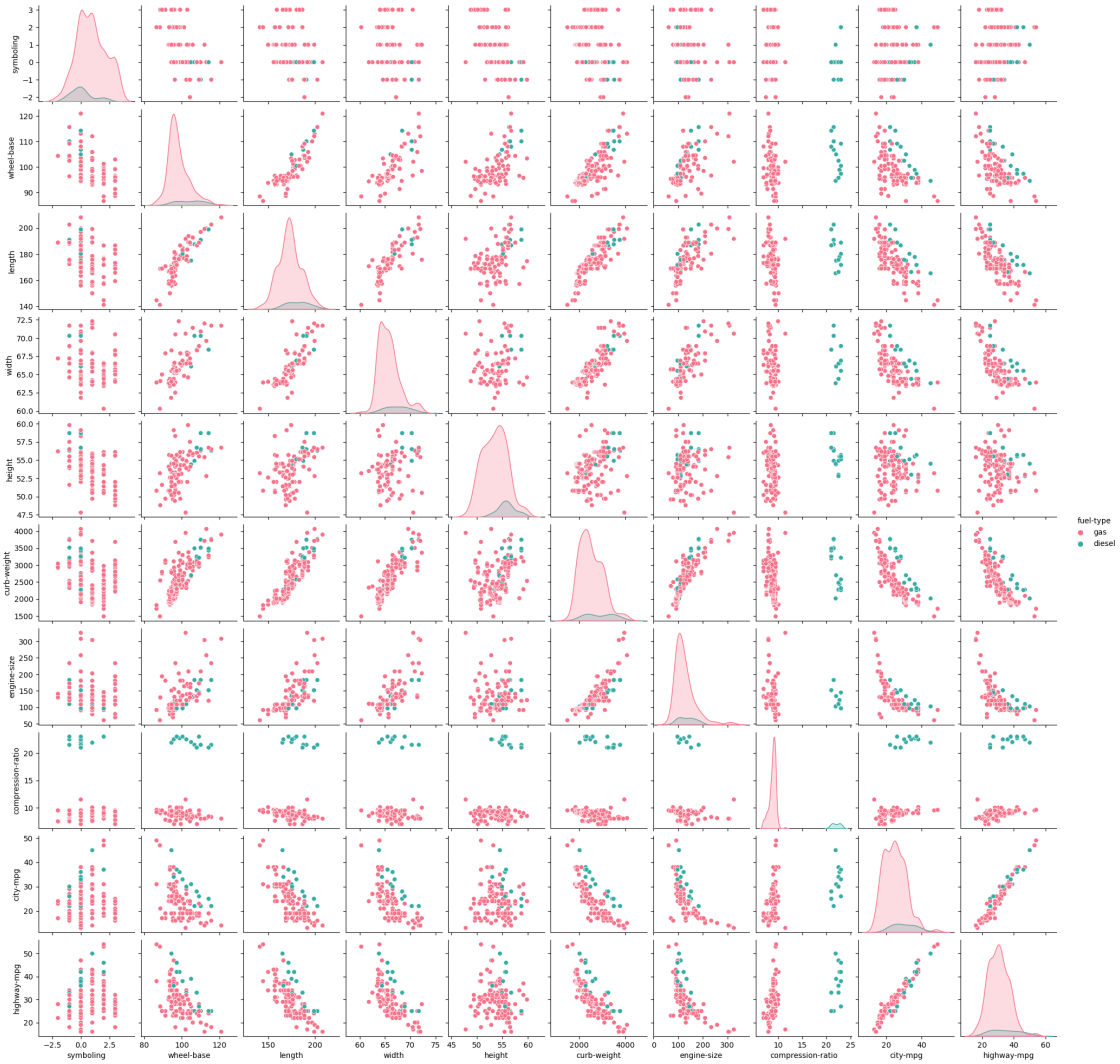
[131]:
```
plt.figure(figsize=(11,9))
sns.pairplot(db,hue = 'fuel-type')
plt.show()
```

```
<Figure size 1100x900 with 0 Axes>
```

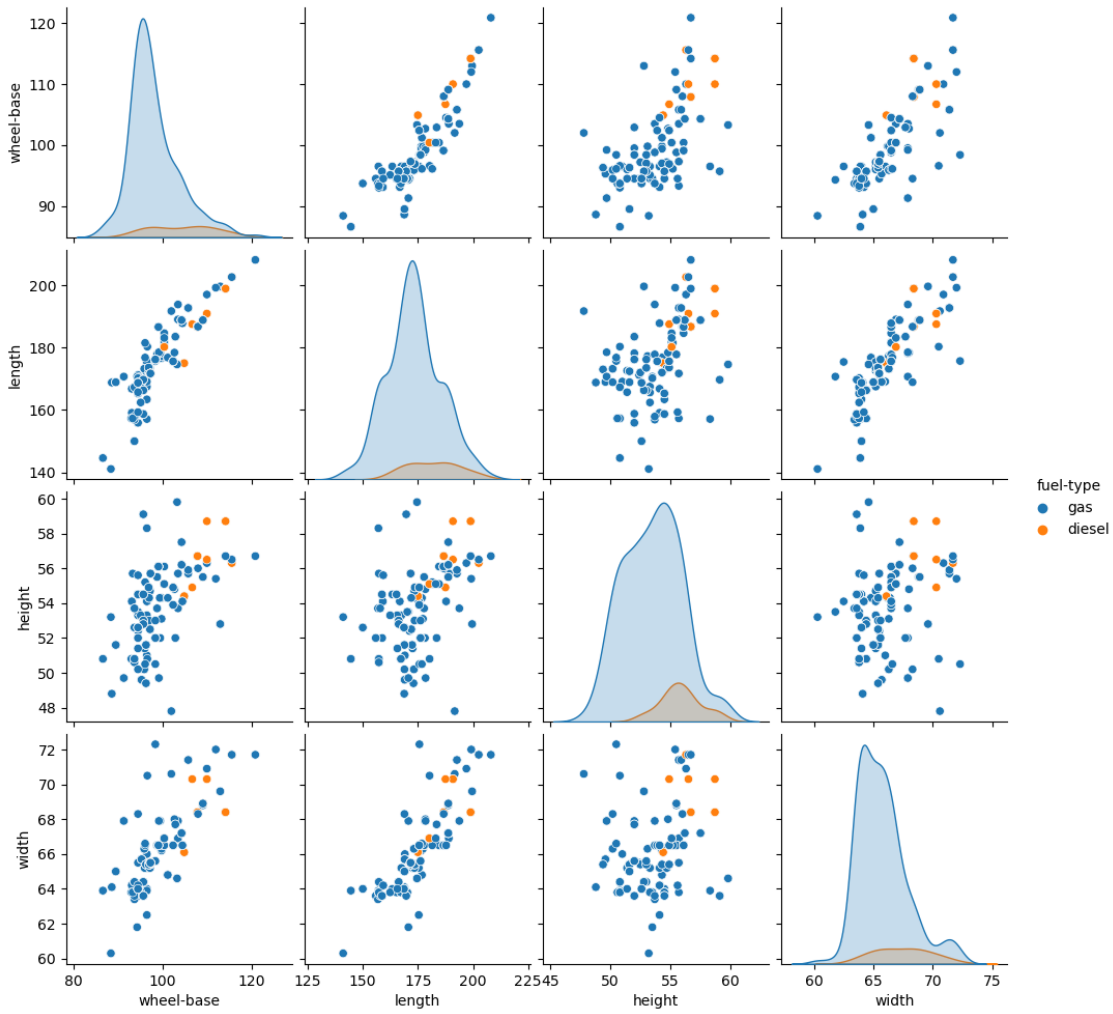


[132]:
```
sns.pairplot(db,hue = 'fuel-type',palette="husl",size=2)
plt.show()
```

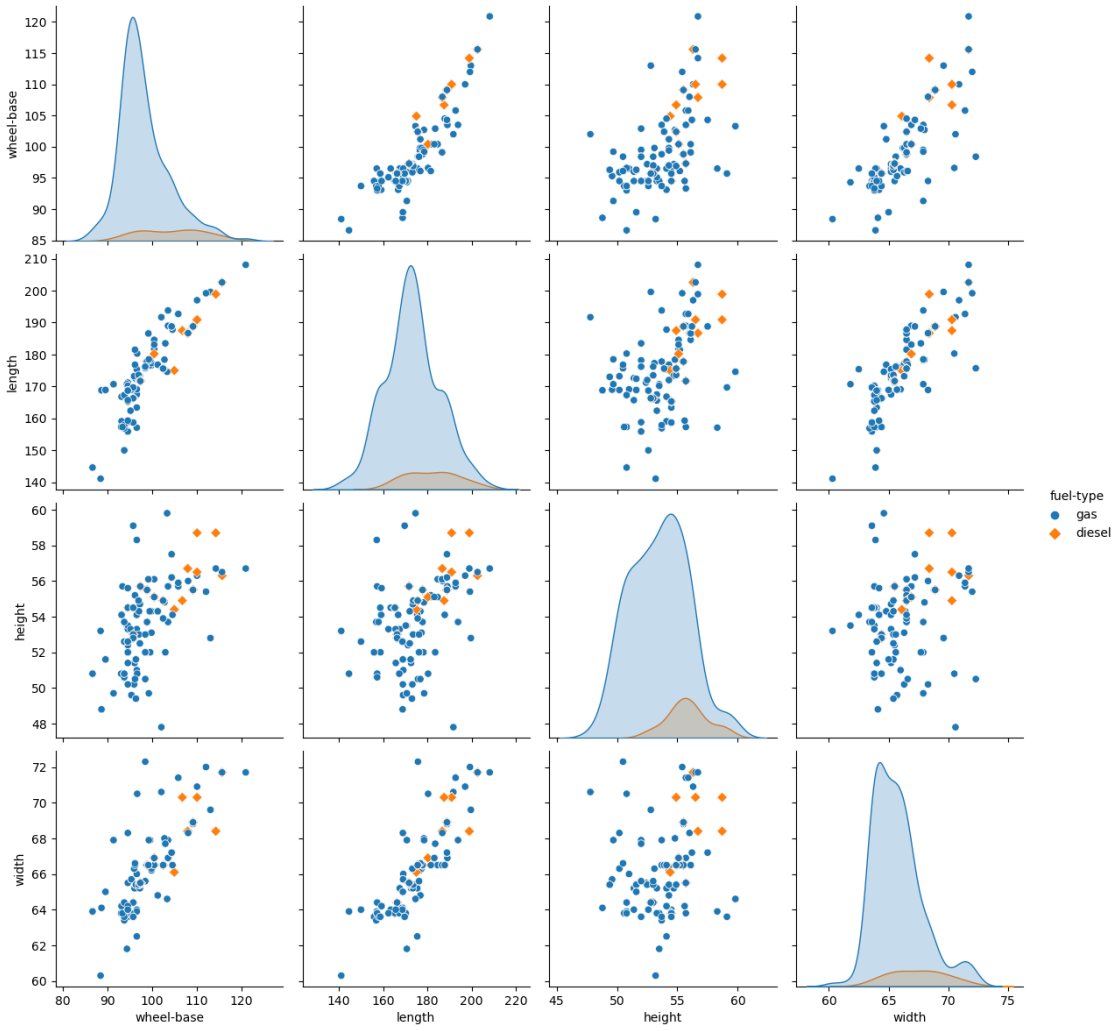
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\axisgrid.py:2076: UserWarning:
The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
```



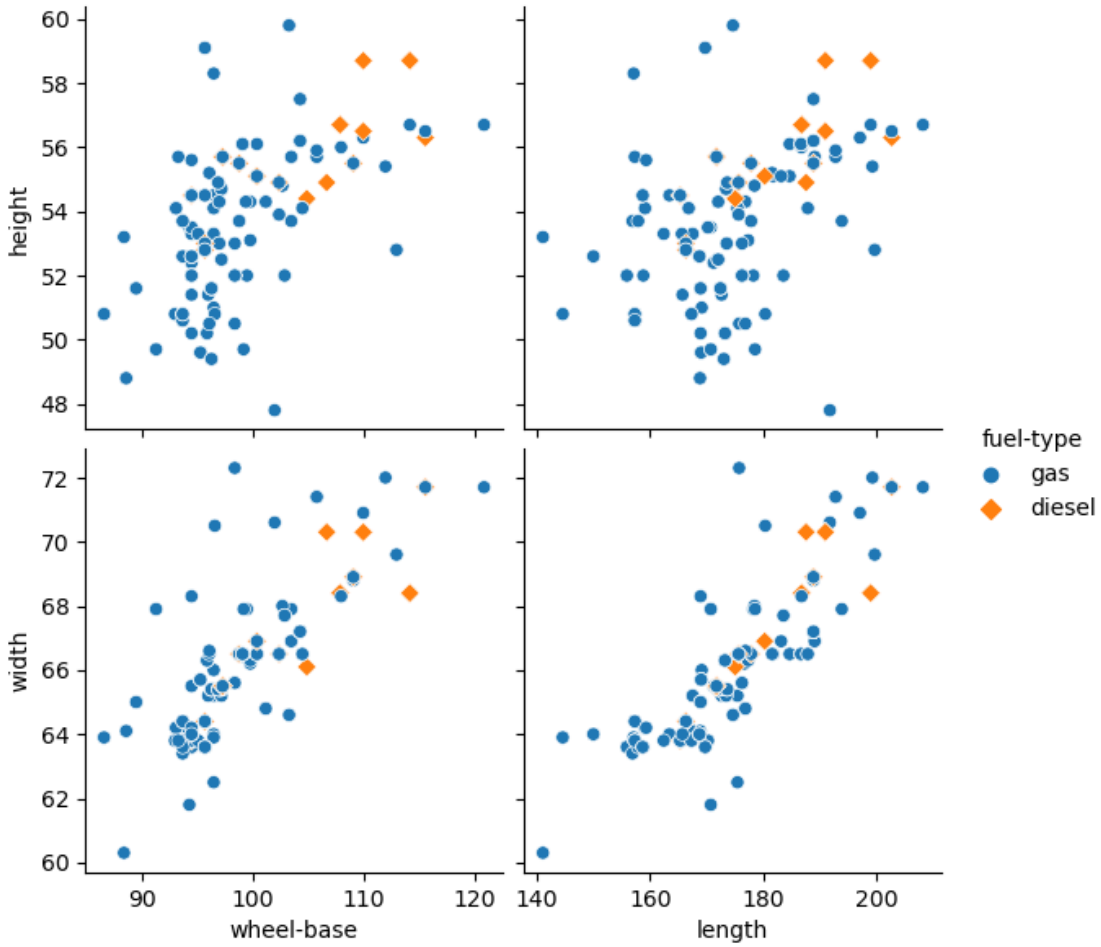
[133]:
```
sns.pairplot(db,hue = 'fuel-type',vars=["wheel-base", "length" , "height" ,␣
↪"width"] )
plt.show()
```



[134]:
```
sns.pairplot(db,hue = 'fuel-type',vars=["wheel-base", "length" , "height" ,␣
↪"width"] ,markers= ['o' , 'D' ] , height=3, aspect=1)
plt.show()
```



[135]:
```
sns.pairplot(db,hue = 'fuel-type',x_vars=["wheel-base", "length"] ,␣
↪y_vars=["height" , "width"],markers= ['o' , 'D' ] , height=3, aspect=1)
plt.show()
```

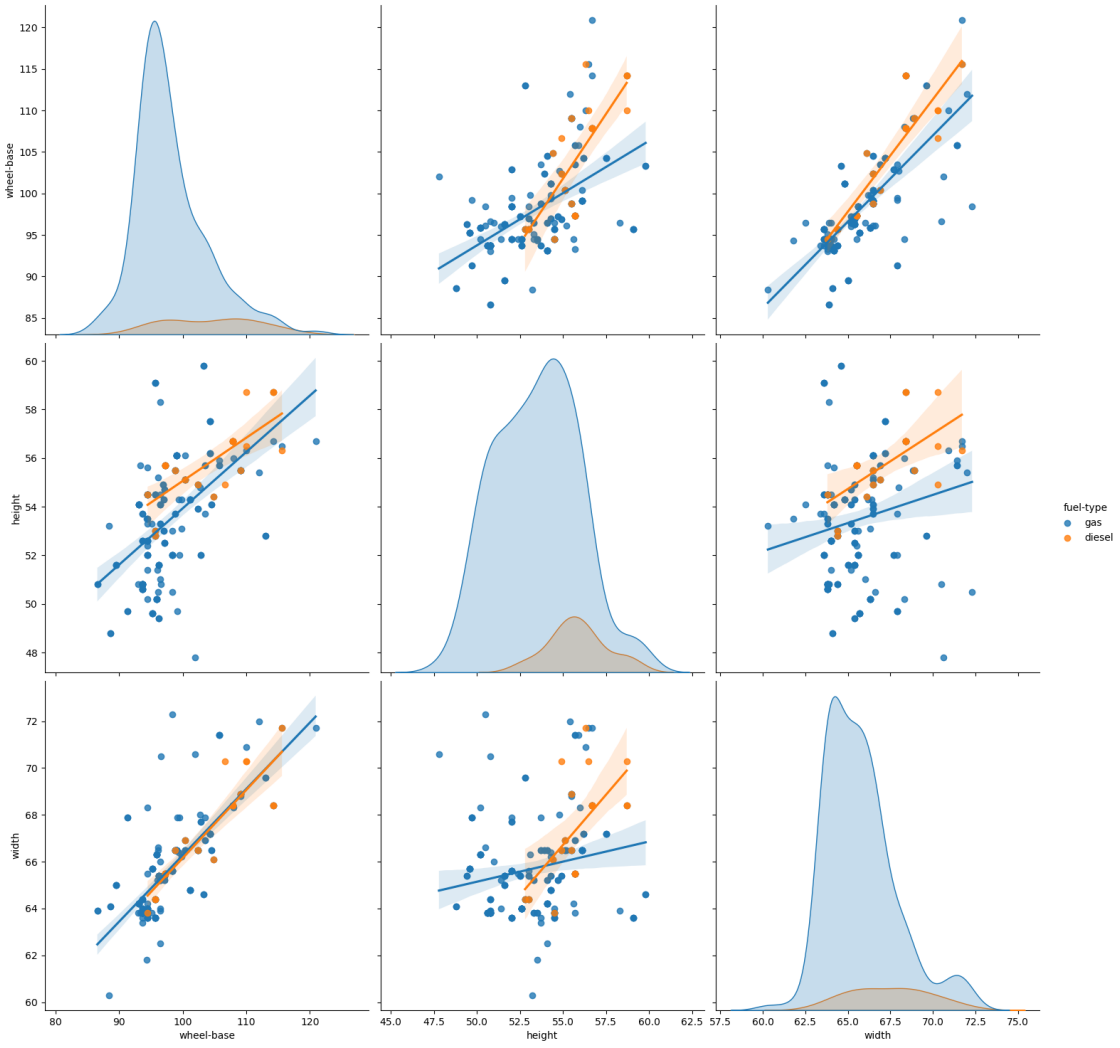


In data visualization, **kind="reg"** is a parameter used in the seaborn library's **pairplot()** function to plot a **regression line** along with the scatter plot in the diagonal cells of a pair plot. This line represents the linear relationship between two variables and can help to identify any **patterns** or **trends** in the data.

When kind="reg", seaborn's pairplot() function generates a pair plot matrix, where the diagonal cells contain a **histogram** of the variable values, and the off-diagonal cells contain scatter plots of the pairwise relationships between variables. The scatter plot in each diagonal cell also includes a regression line that shows the overall trend between the two variables being plotted.

The kind="reg" parameter is useful for quickly visualizing the linear relationships between variables in a dataset and can provide insights into how variables are related to each other. It can also be used to identify outliers or any other unusual patterns in the data.

[136]:
```
sns.pairplot(db,hue = 'fuel-type',vars=["wheel-base" , "height" , "width"] ,␣
↪kind="reg",height=5, aspect=1)
plt.show()
```



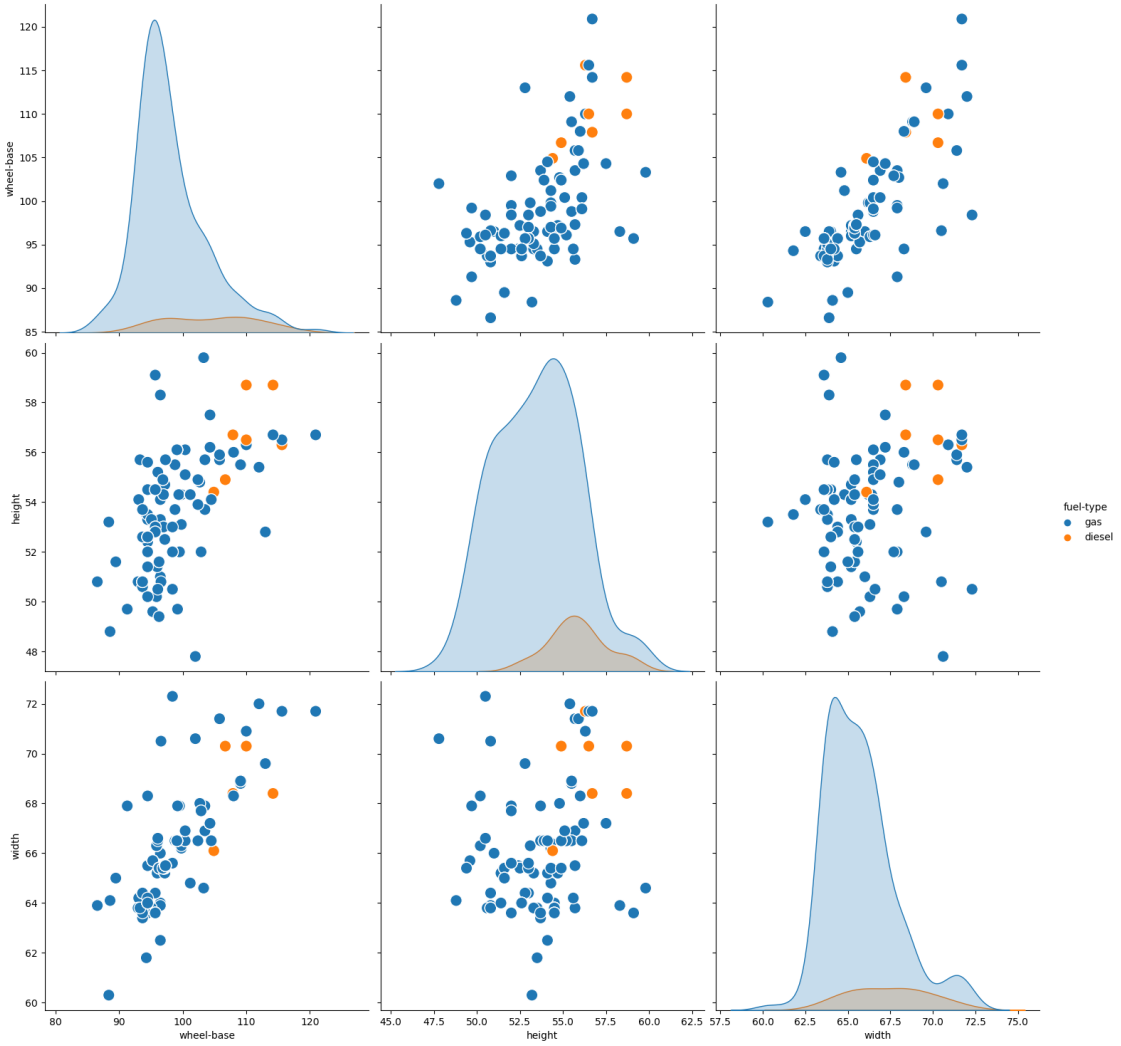
In data visualization, **plot_kws=dict(s=140)** is a parameter used in the seaborn library's pairplot() function to specify the keyword arguments that control the appearance of scatter plots in a pair plot.

The **plot_kws** parameter is a dictionary that accepts various keyword arguments that are passed to the underlying matplotlib functions used to generate the plots. In this case, the **s** argument is used to control the size of the markers in the scatter plot.

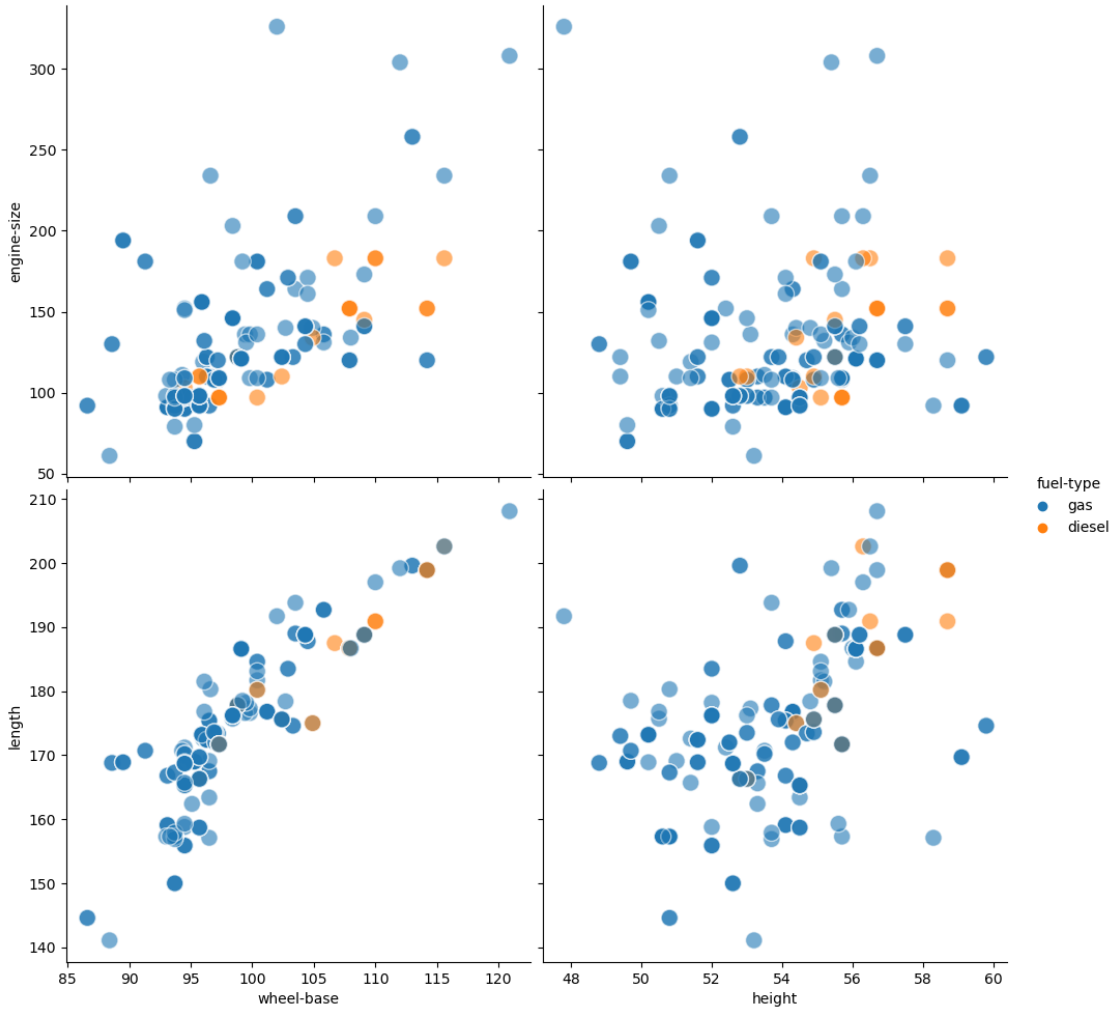
By setting **s=140**, the size of each marker in the scatter plot will be set to 140 points, making them larger and more visible on the plot. This can be useful when plotting datasets with many points, as it can help to make the plot more readable and easier to interpret.

Other keyword arguments that can be specified in the plot_kws dictionary include alpha for controlling the transparency of the markers, color for setting the color of the markers, and marker for specifying the shape of the markers used in the plot.

[137]:
```
sns.pairplot(db,hue = 'fuel-type',vars=["wheel-base" , "height" , "width"] ,␣
↪plot_kws=dict(s=140),height=5, aspect=1)
plt.show()
```



[138]:
```
sns.pairplot(db,hue = 'fuel-type',x_vars=["wheel-base" , "height"]␣
↪,y_vars=["engine-size" , "length"] ,plot_kws=dict(s=140, linewidth=1,alpha= .
↪6),height=5, aspect=1)
plt.show()
```



1.15 Pair grid

In data visualization, a **pair grid** is a grid of subplots that shows pairwise relationships between variables in a dataset. It is similar to a pair plot, but offers more flexibility and control over the layout and appearance of the subplots.

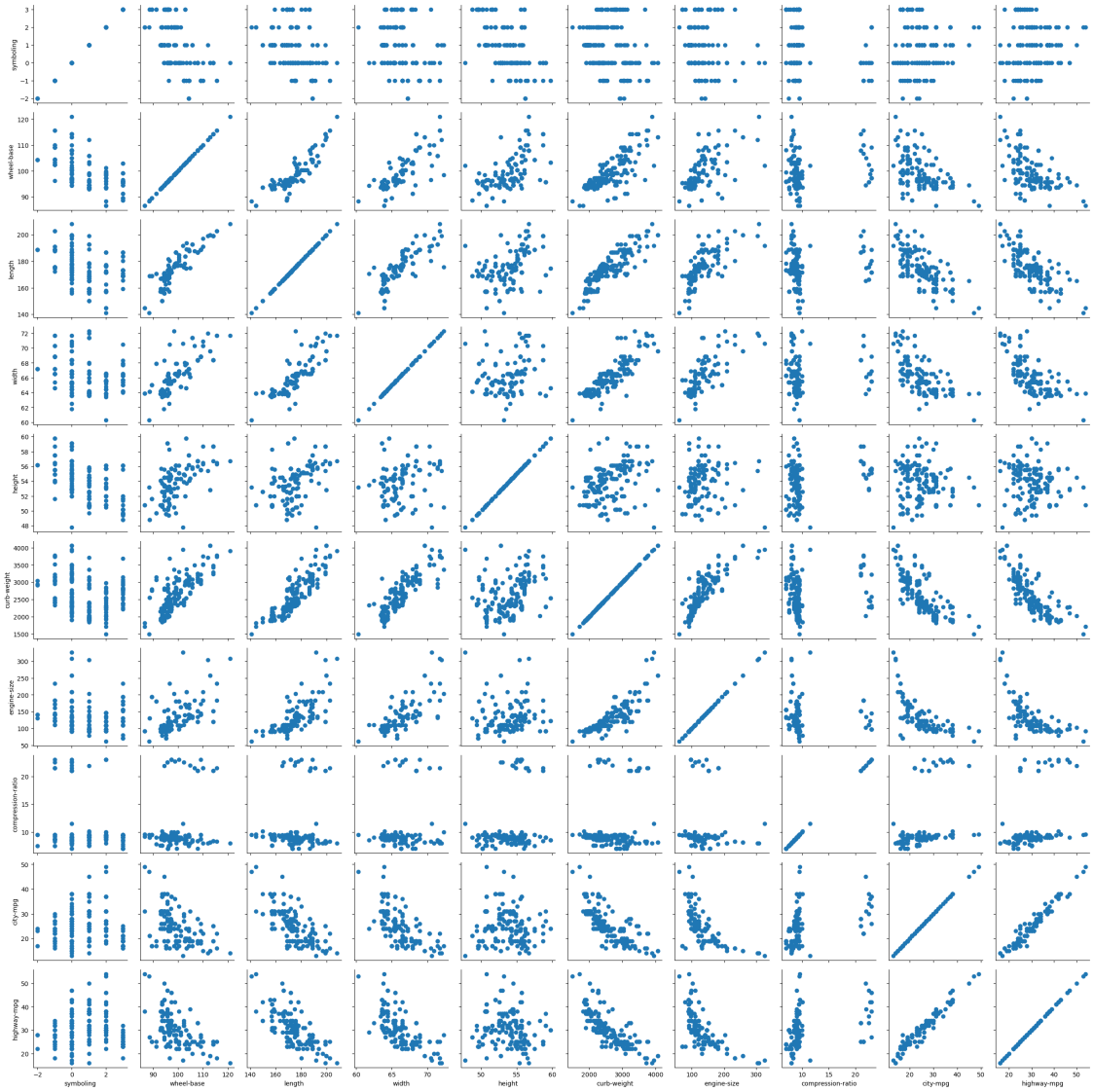
A pair grid can be created using the seaborn library's **PairGrid** function. This function takes a dataset as input and allows the user to specify which variables to include in the grid, as well as various other options such as the type of plot to use for each subplot.

Once a pair grid has been created, the user can customize the appearance of each subplot by accessing the individual axes objects and applying styling commands using matplotlib. This can be useful for creating complex visualizations with multiple layers of information.

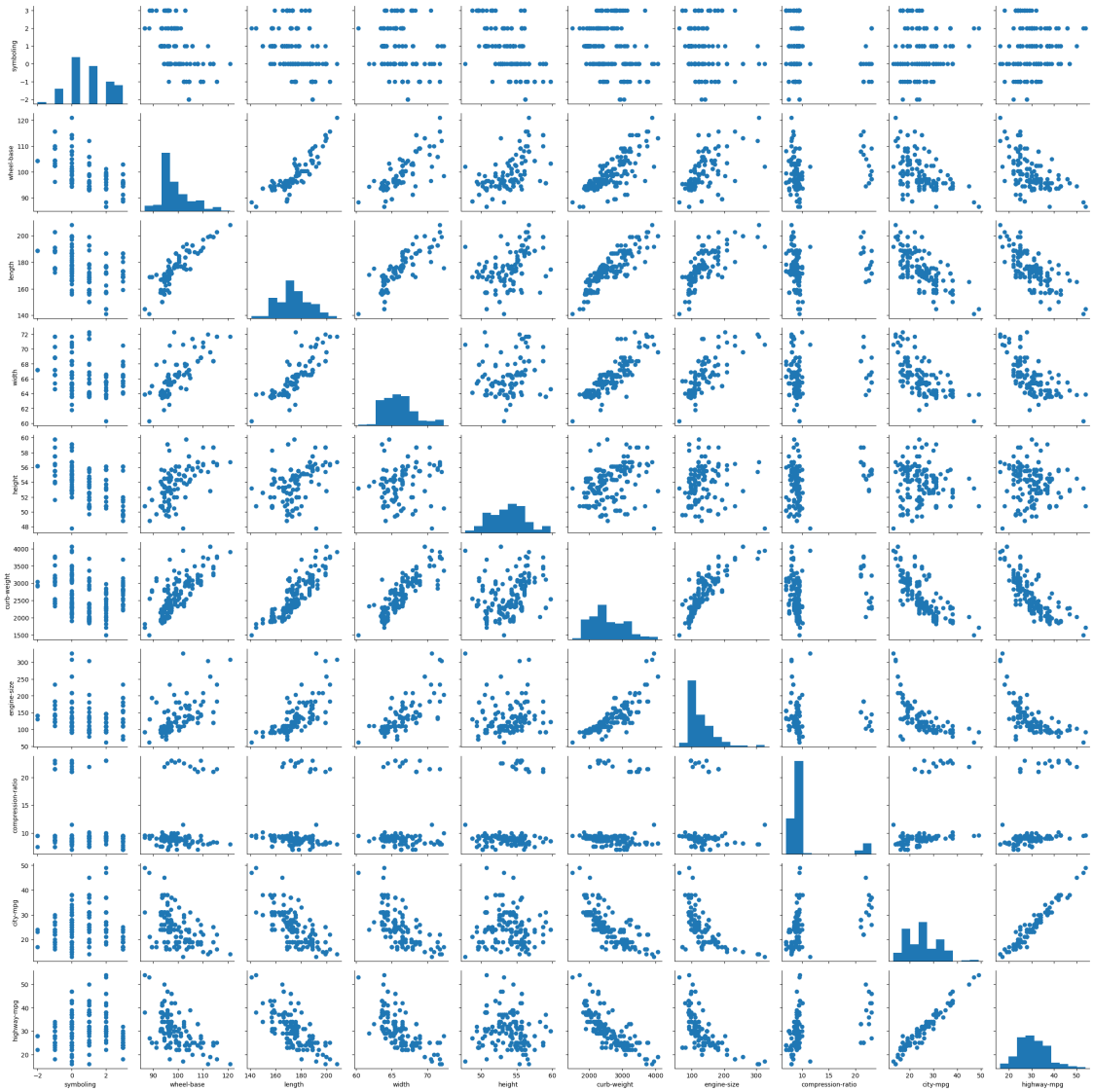
Pair grids are a powerful tool for exploring complex datasets and can help to reveal patterns and relationships that may not be immediately obvious from examining individual variables in isolation. They are commonly used in exploratory data analysis (EDA) and can also be used in

machine learning to visualize the relationships between input variables and target variables.

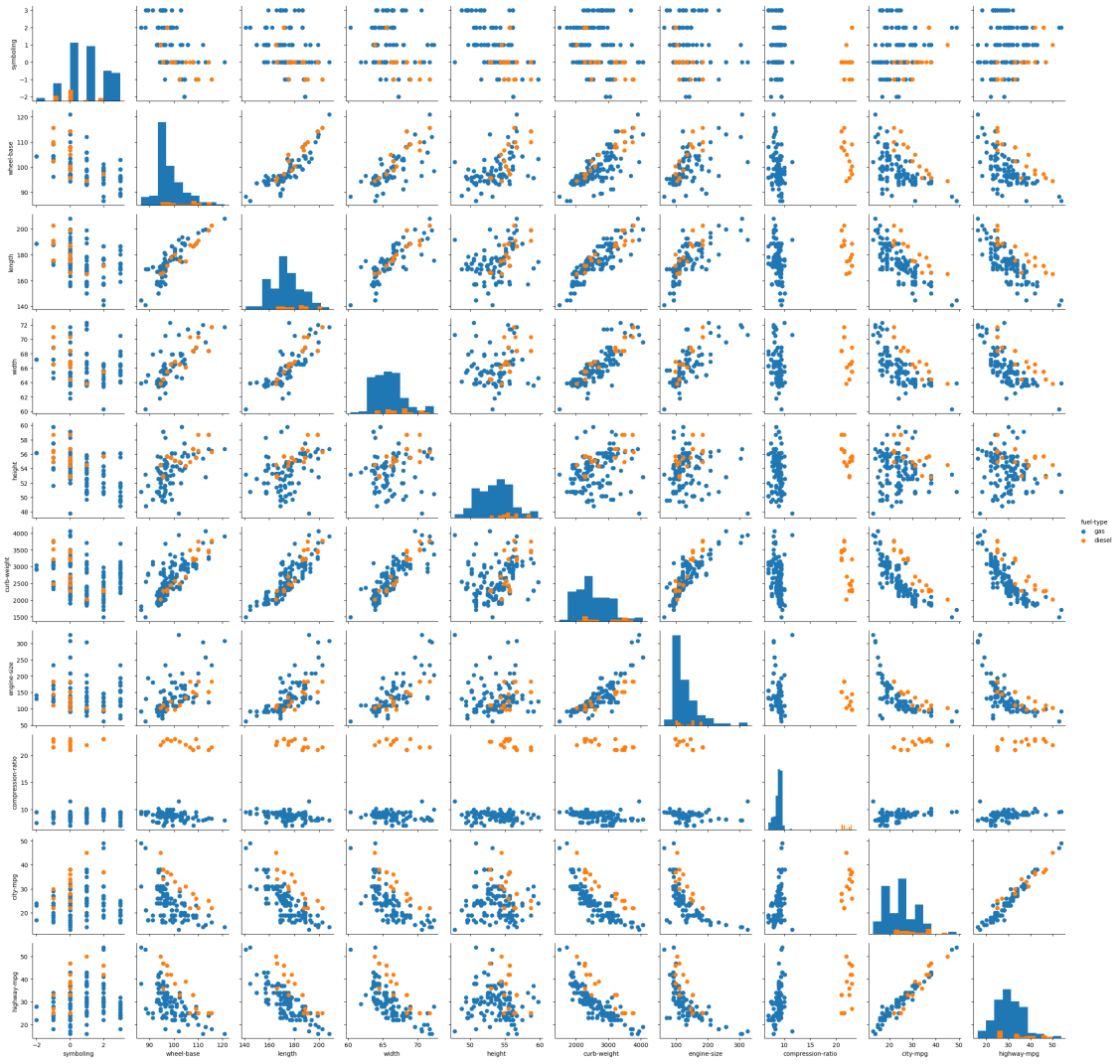
[139]:
```
g = sns.PairGrid(db)
g = g.map(plt.scatter)
plt.show()
```



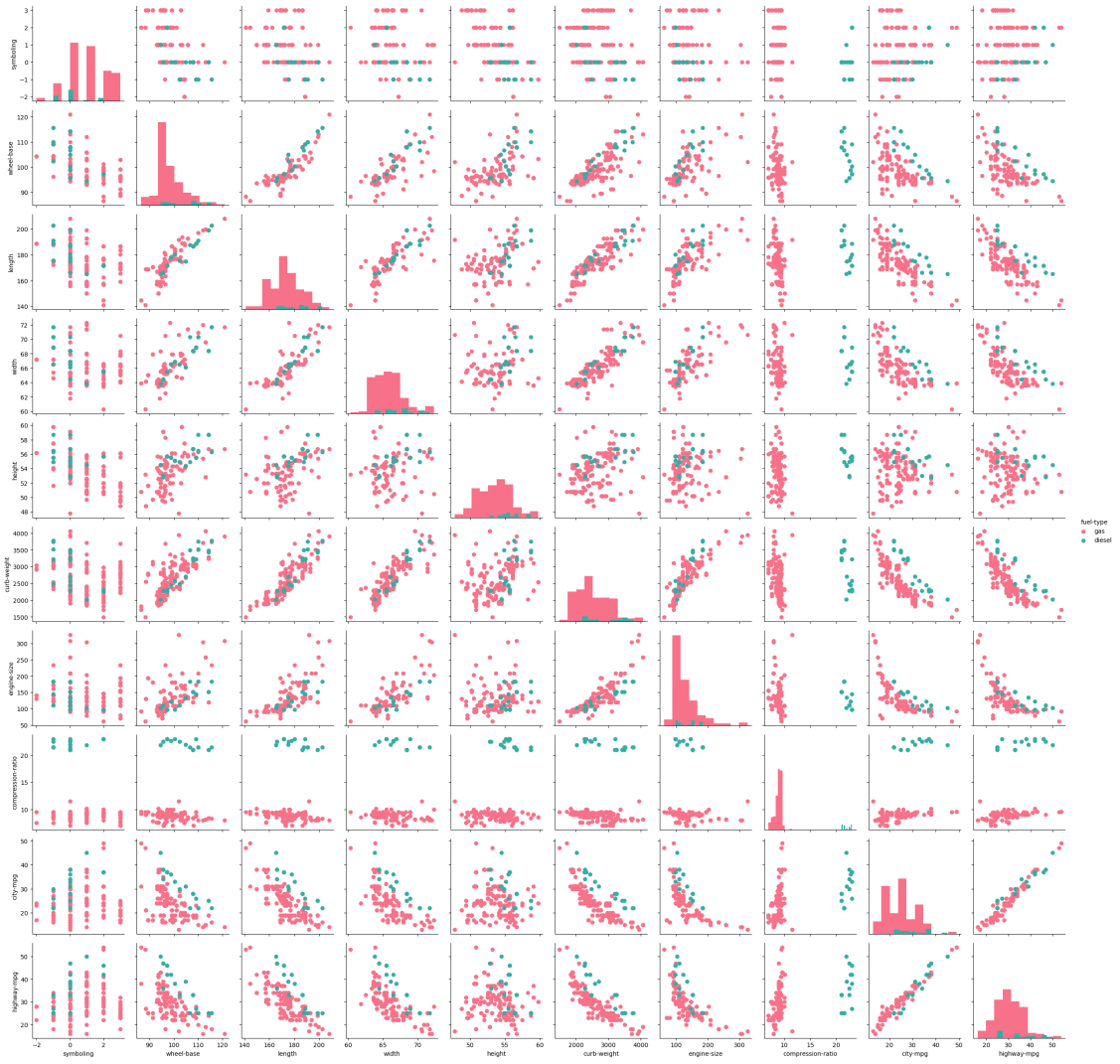
[140]:
```
g = sns.PairGrid(db)
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
plt.show()
```



[141]:
```
g = sns.PairGrid(db , hue='fuel-type')
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
plt.show()
```



[142]:
```
g = sns.PairGrid(db , hue='fuel-type', palette="husl")
g = g.map_offdiag(plt.scatter)
g = g.map_diag(plt.hist)
g = g.add_legend()
plt.show()
```



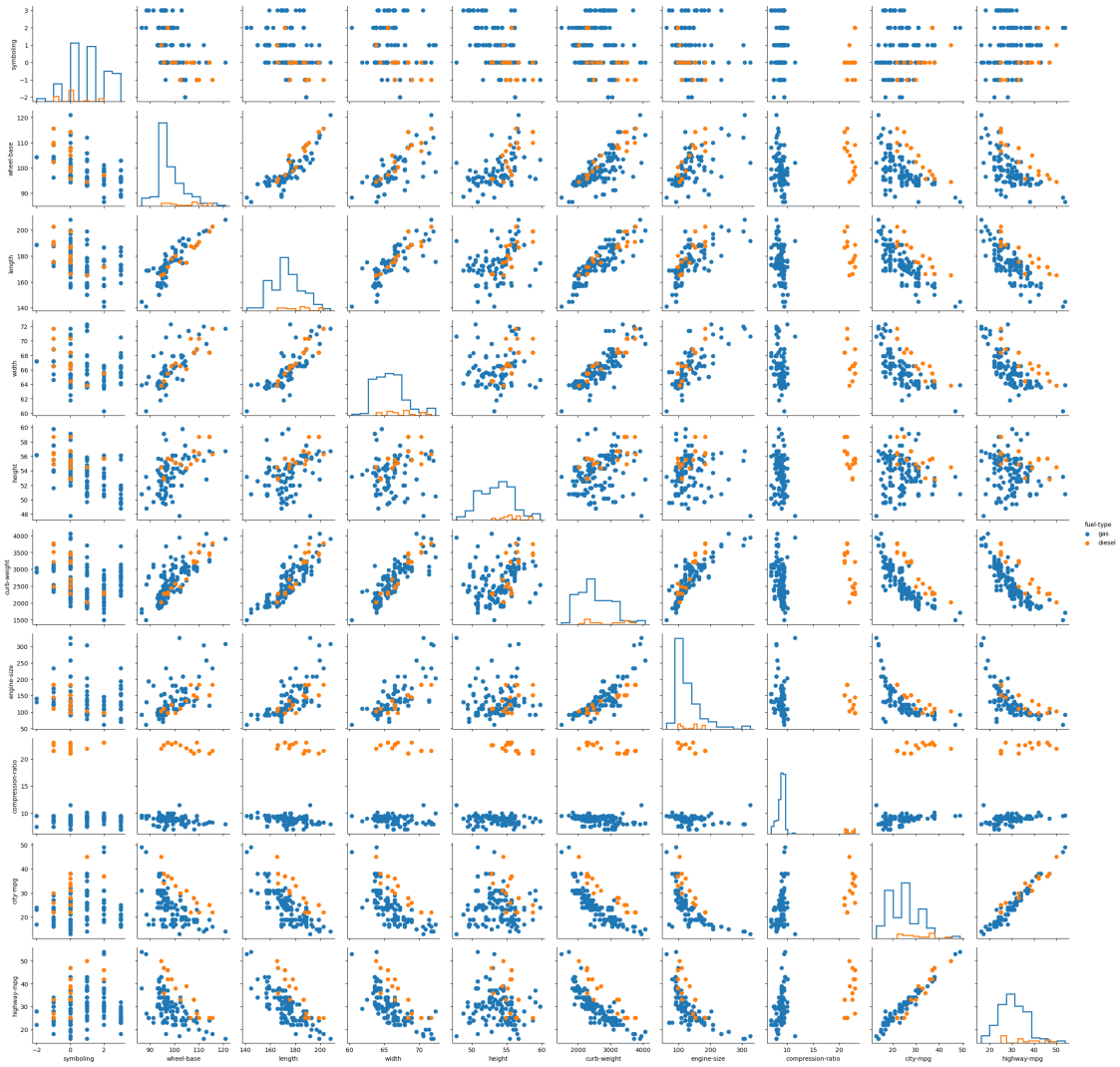
In data visualization, **histtype="step"** is a parameter used in the matplotlib library's **hist()** function to specify the type of histogram to be plotted. When histtype="step", the histogram is plotted as a line chart rather than as a bar chart.

When a histogram is plotted with histtype="step", each bin is represented by a vertical line segment that is connected to the neighboring segments, forming a continuous line chart that shows the distribution of the data. This type of histogram is useful for visualizing the shape of the distribution and for identifying any patterns or outliers in the data.

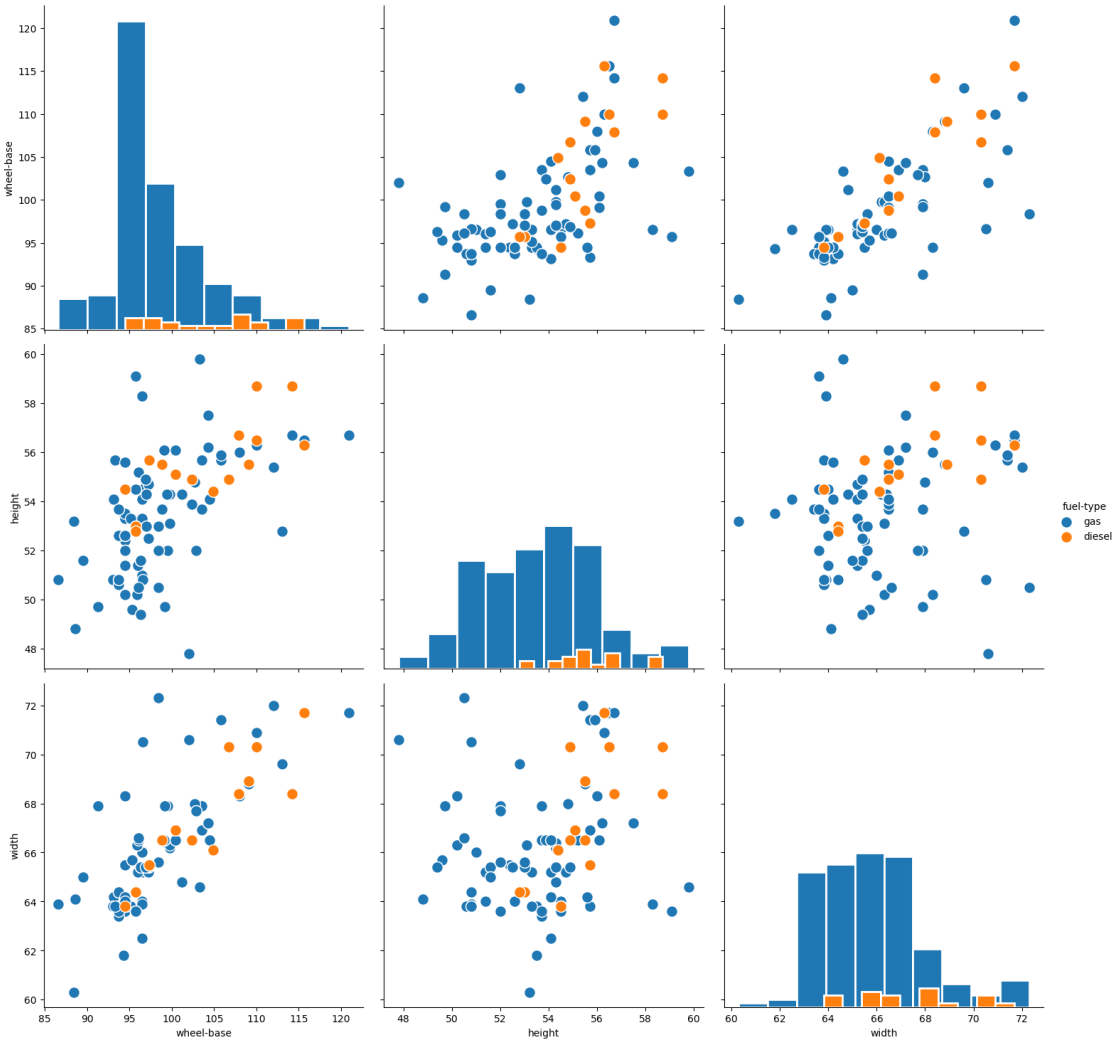
Other options for histtype include **"bar"**, which plots the histogram as a bar chart, and **"step-filled"**, which plots the histogram as a line chart with the area below the line filled in with color. The choice of **histtype** depends on the specific data being plotted and the visual representation that is most appropriate for the analysis.

[143]:
```
g = sns.PairGrid(db , hue='fuel-type')
g = g.map_offdiag(plt.scatter)
```

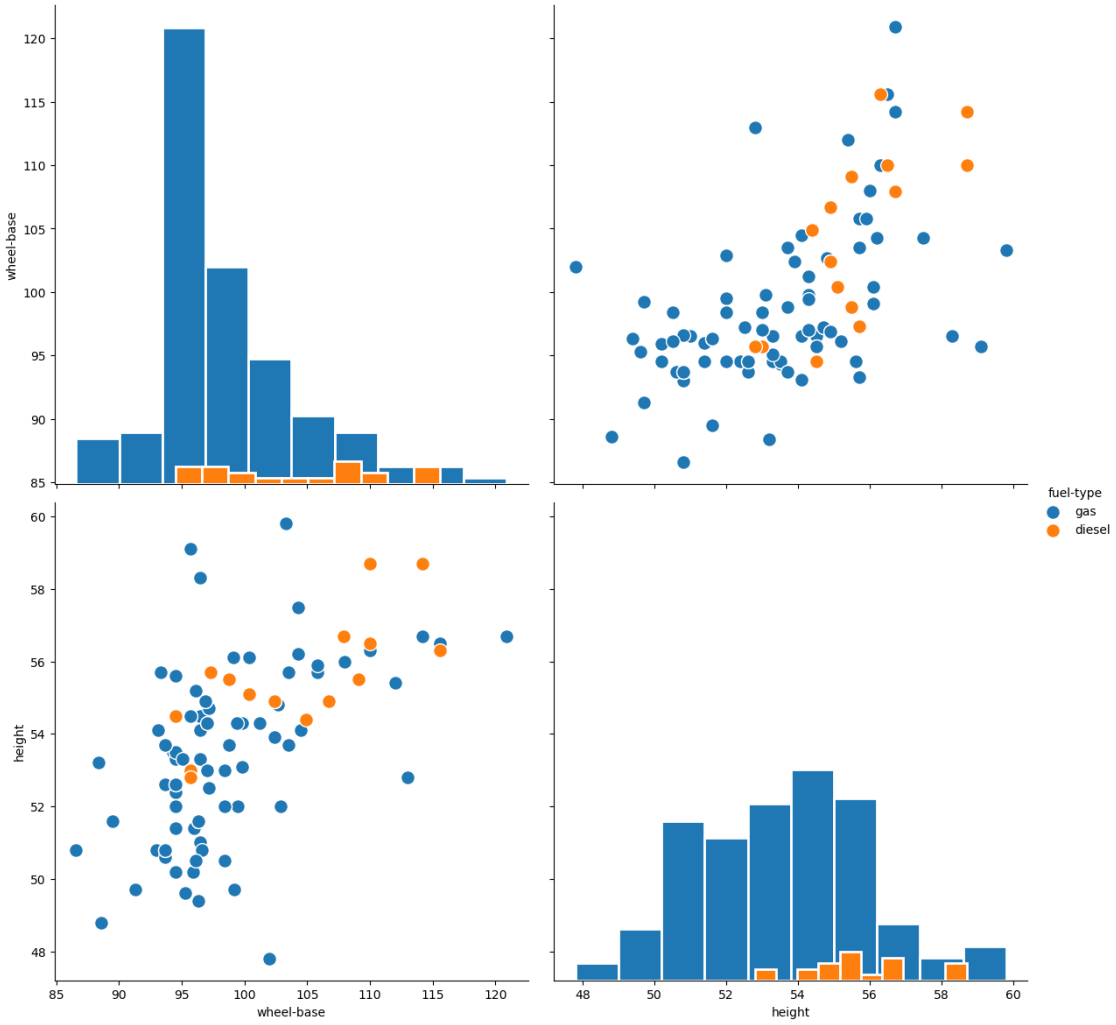
```
g = g.map_diag(plt.hist , histtype="step", linewidth=2)
g = g.add_legend()
plt.show()
```



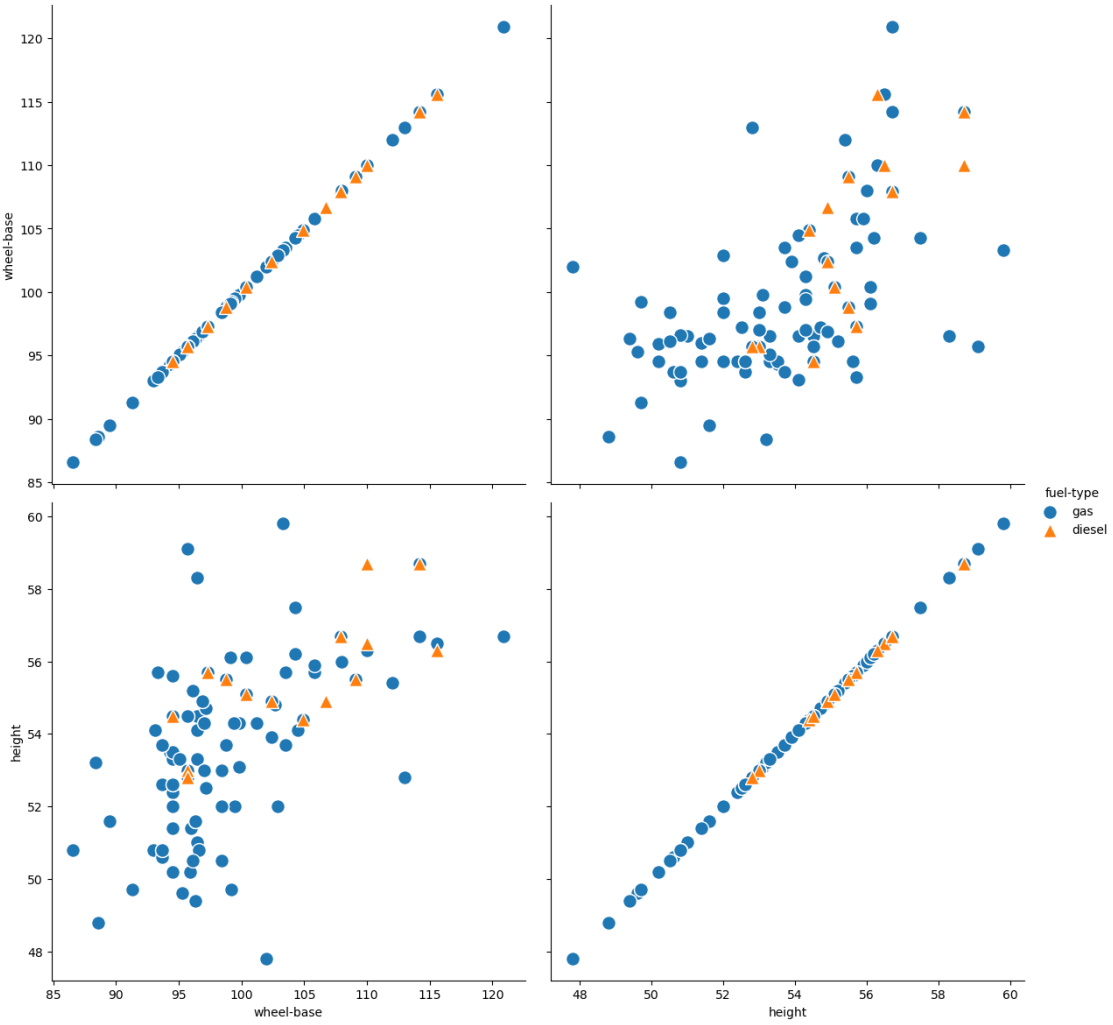
[144]:
```
g = sns.PairGrid(db , hue='fuel-type' ,vars=["wheel-base" , "height" ,␣
  ↪"width"],height=5, aspect=1)
g = g.map_offdiag(plt.scatter , edgecolor="w", s=130)
g = g.map_diag(plt.hist , edgecolor ='w', linewidth=2)
g = g.add_legend()
plt.show()
```



[145]:
```
g = sns.PairGrid(db , hue='fuel-type' ,x_vars=["wheel-base" ,␣
↪"height"],y_vars=["wheel-base" , "height"],height=6, aspect=1)
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()
```



[146]:
```
g = sns.PairGrid(db , hue='fuel-type' ,x_vars=["wheel-base"␣
↪,"height"],y_vars=["wheel-base" , "height"],height=6, aspect=1 ,␣
↪hue_kws={"marker": ["o", "^"]})
g = g.map(plt.scatter , edgecolor="w", s=130)
g = g.add_legend()
plt.show()
```



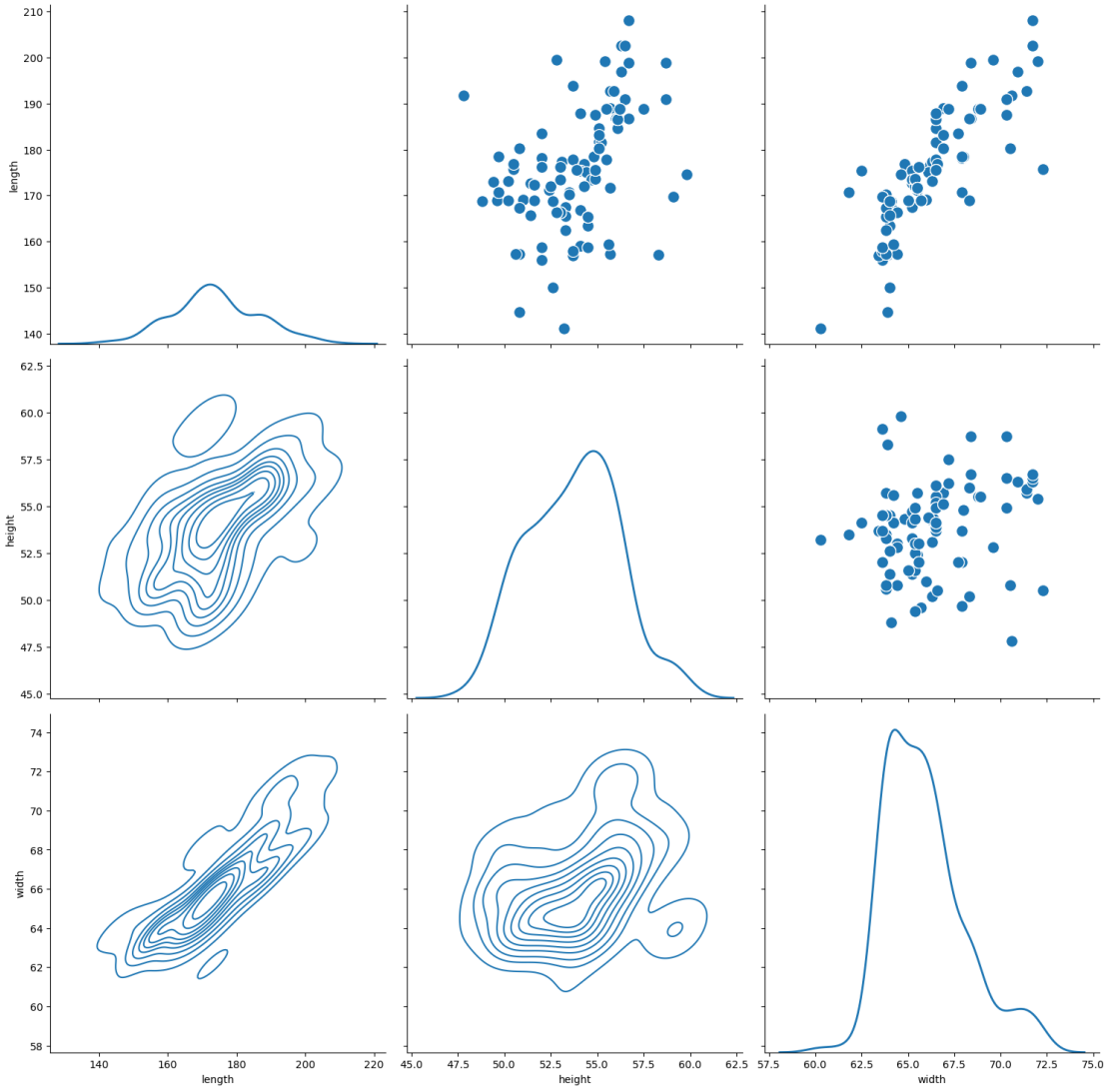
In data visualization, **g = g.map_lower(sns.kdeplot)** is a line of code that can be used in the seaborn library to create a lower-triangle of a pair grid that displays the **bivariate kernel density estimate (KDE) plots** of the pairwise relationships between variables in a dataset.

The **map_lower()** method is used to apply a plotting function to the lower triangle of a pair grid. In this case, the plotting function being applied is **sns.kdeplot**, which is a function in the seaborn library that creates a KDE plot of the data.

The KDE plot is a non-parametric way to estimate the probability density function of a random variable. It is a smoothed version of a histogram and shows the distribution of data in a continuous way. By displaying KDE plots in the lower triangle of a pair grid, the user can visualize the joint distribution of two variables, as well as the marginal distributions of each variable separately.

This technique is useful for identifying patterns and trends in the data, such as bimodal distributions or correlations between variables. It is commonly used in exploratory data analysis (EDA) to gain insights into the structure of the data before applying more advanced statistical techniques.

[147]:
```
g = sns.PairGrid(db ,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()
```



1.16 Regression plots

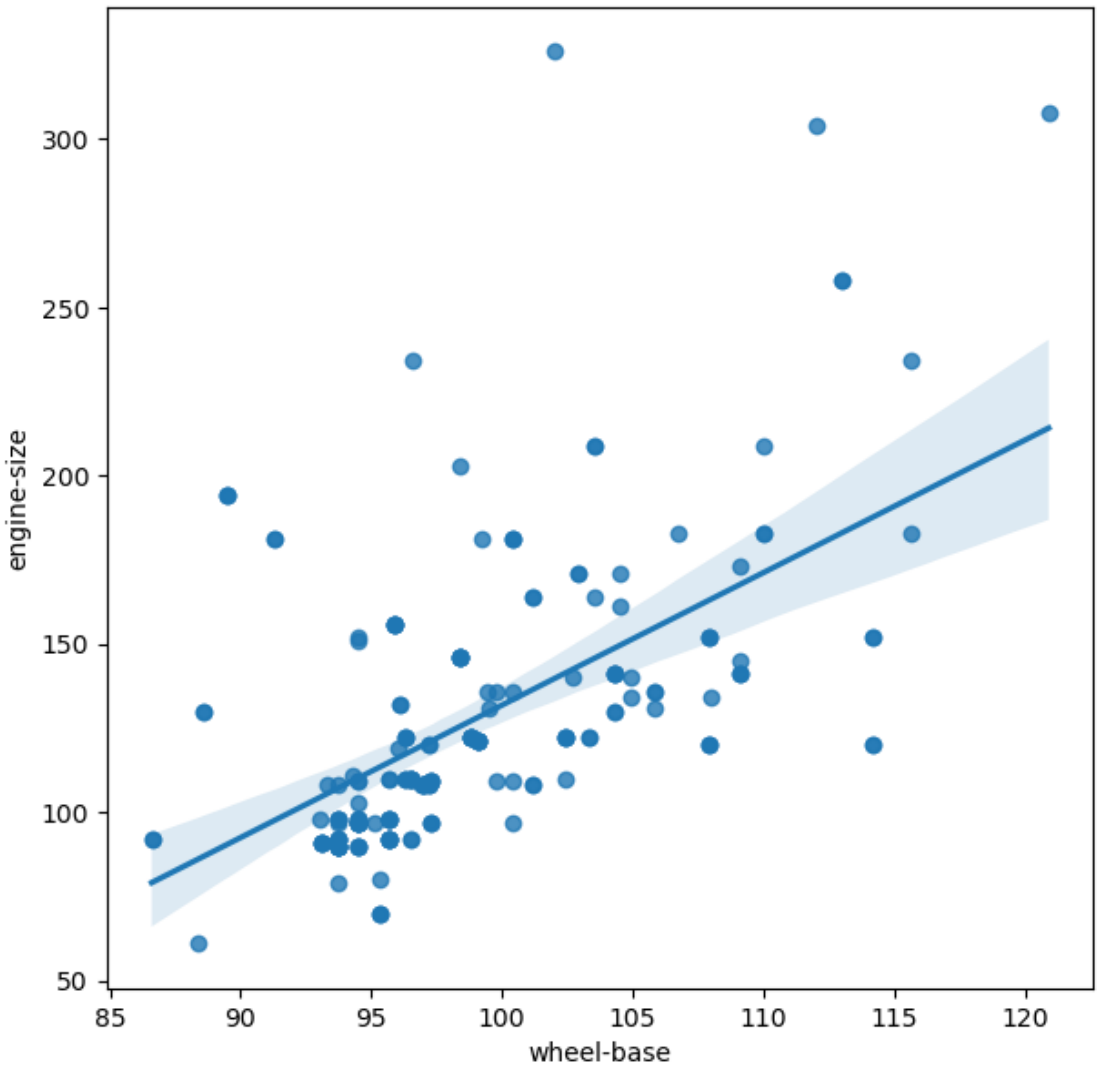
Regression plots are a type of data visualization that are used to explore the relationship between two variables in a dataset, with a focus on identifying patterns and trends in the data. Regression plots are often used in data analysis and statistical modeling to determine whether there is a **linear relationship** between two variables, and to estimate the strength and direction of that relationship.

A regression plot typically shows a scatter plot of the data points, with a line or curve fitted to the data that represents the relationship between the variables. The line or curve is usually based on a regression analysis, which is a statistical technique that estimates the parameters of a linear or nonlinear model that describes the relationship between the variables.

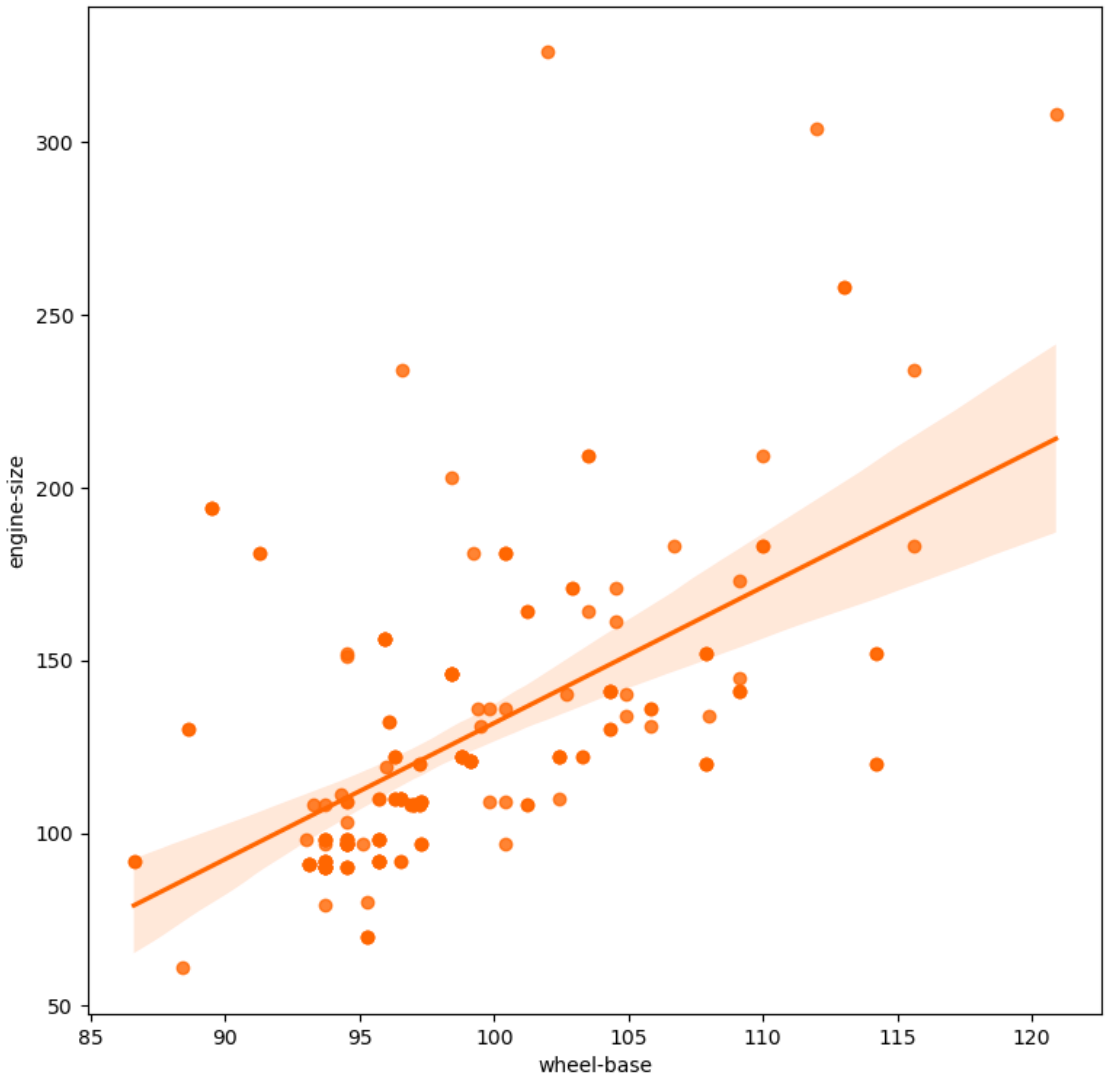
Regression plots can be created using various data visualization libraries in Python, including seaborn and matplotlib. Seaborn's **regplot()** function, for example, is a convenient way to create a regression plot in which a linear regression line is fitted to the data and displayed on the plot. Other types of regression plots include **residual plots**, which show the difference between the observed values and the predicted values from a regression model, and **partial regression plots**, which show the relationship between two variables while controlling for the effects of other variables in the model.

[148]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

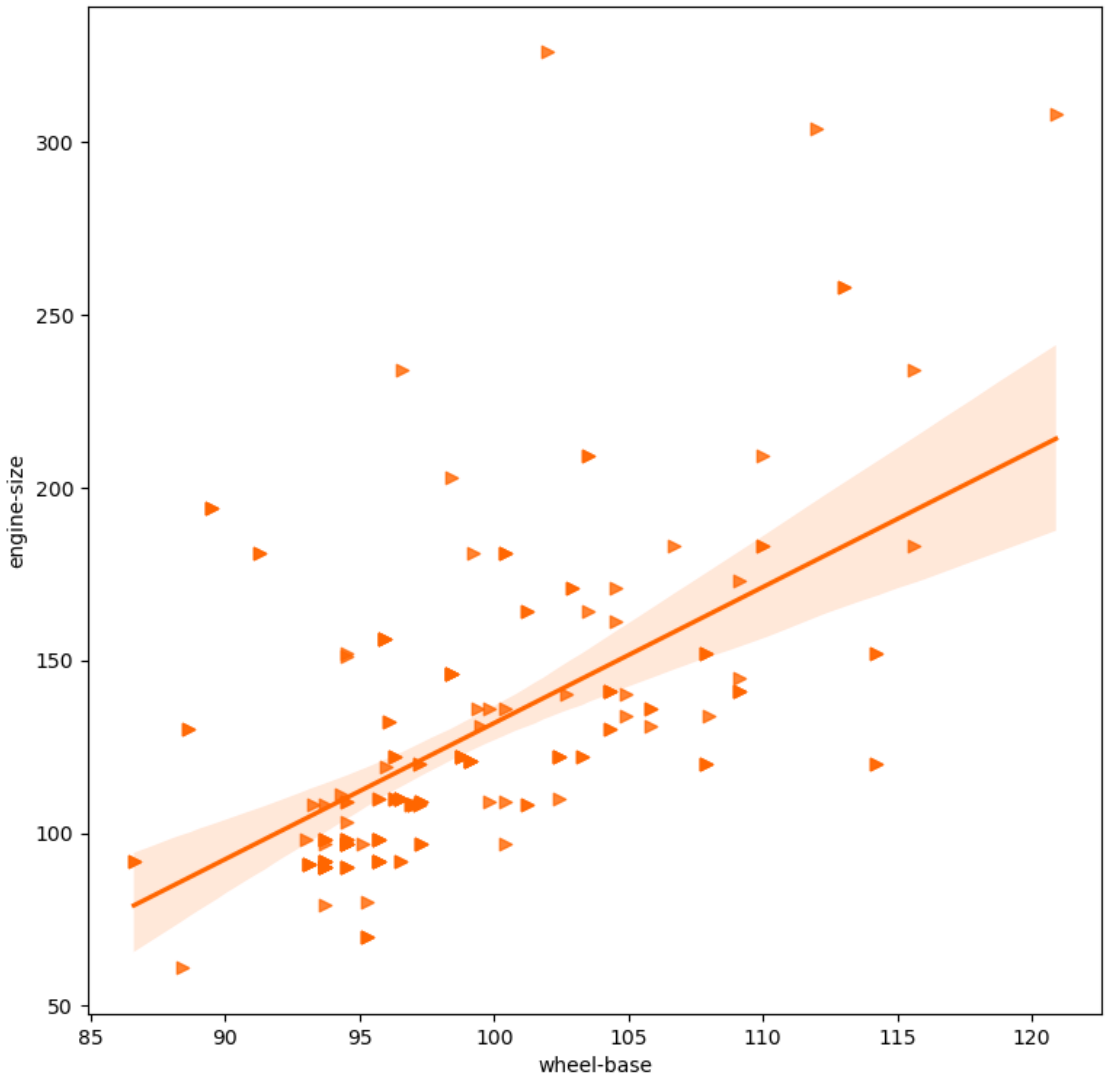
[149]:
```
plt.figure(figsize=(7,7))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"])
plt.show()
```



[150]:
```
plt.figure(figsize=(9,9))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"] , color='#FF6600')
plt.show()
```



[151]:
```
plt.figure(figsize=(9,9))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"] , color='#FF6600' ,␣
  ↪marker='>')
plt.show()
```



In data visualization, **ci=50** is a parameter used in the seaborn library's **regplot()** function to specify the size of the confidence interval to be displayed on a regression plot.

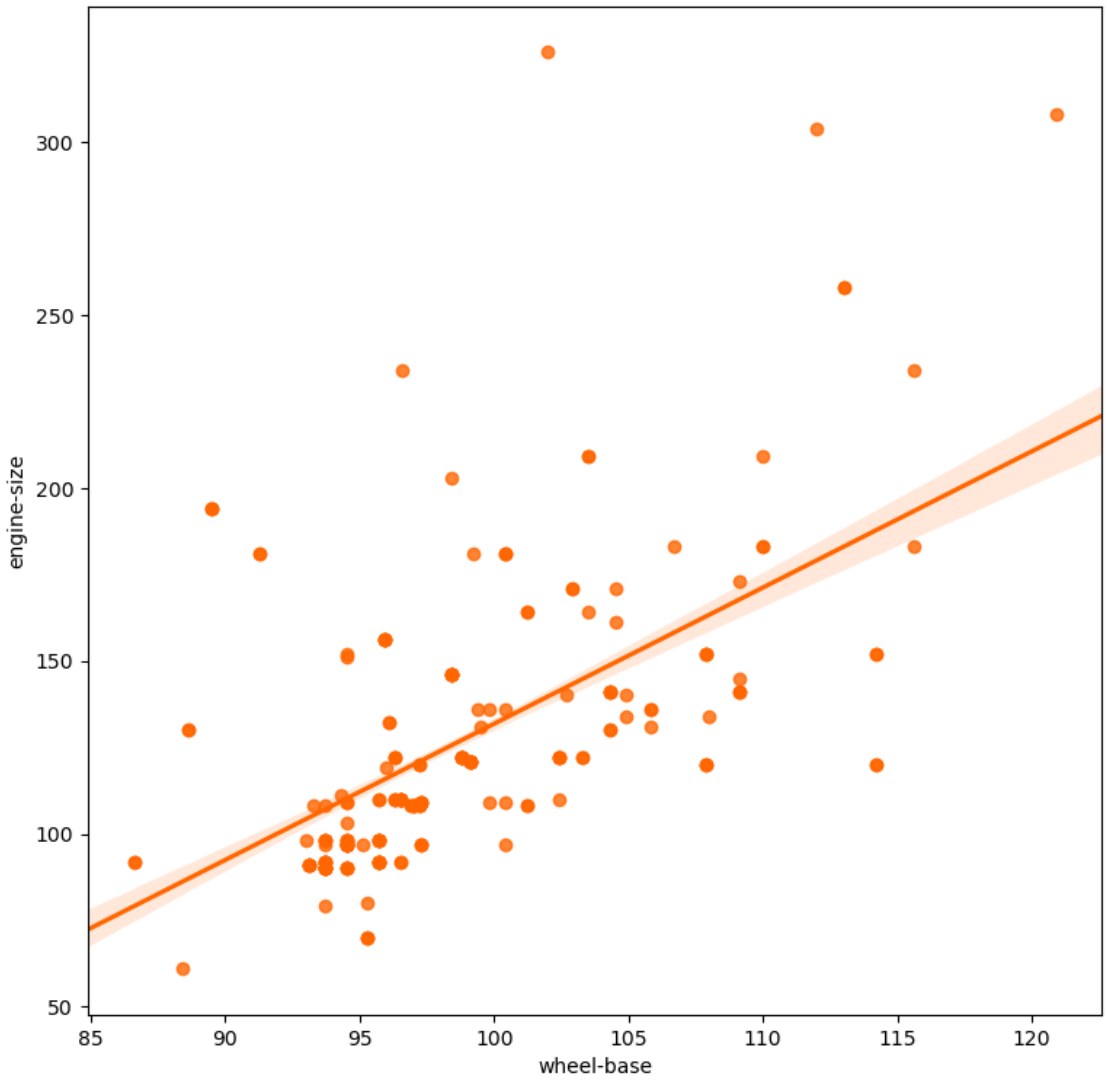
The **ci** parameter controls the width of the confidence interval around the regression line, which represents the uncertainty in the estimate of the relationship between the variables. By default, **ci=95**, which means that the confidence interval displayed on the plot will include 95% of the data points.

When **ci=50**, the confidence interval displayed on the plot will be **narrower**, including only 50% of the data points. This can be useful for creating a more focused visualization of the relationship between the variables, particularly if there are a large number of data points or if the relationship is weak and difficult to discern.

Other values for ci can be specified, including 68, 90, and 99, among others. The choice of ci depends on the specific data being plotted and the level of uncertainty that is acceptable in the

analysis.

[152]:
```
plt.figure(figsize=(9,9))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"] , color='#FF6600' , ci=50␣
↪, truncate=False)
plt.show()
```



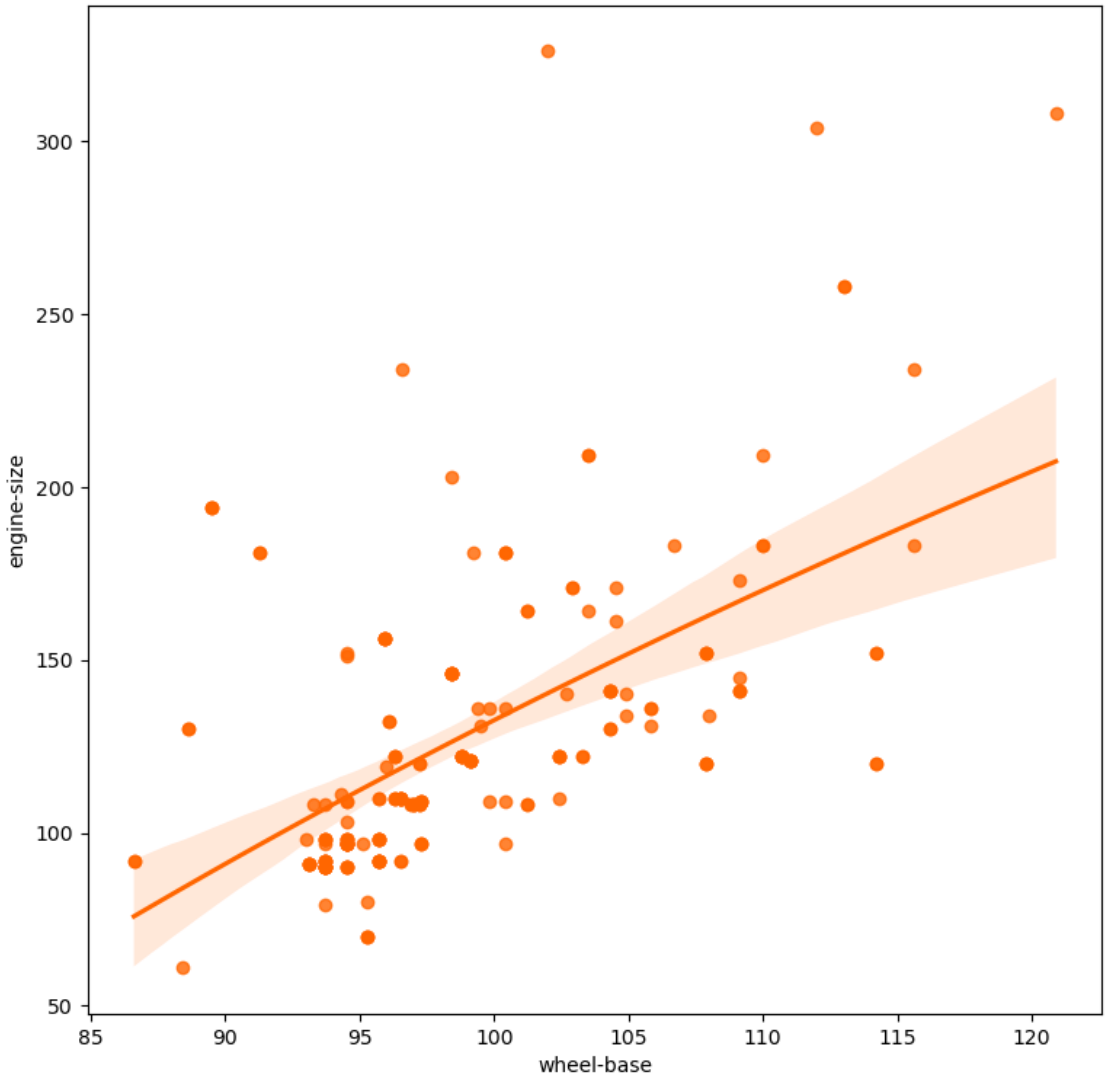
In data visualization, **logx=True** is a parameter used in various plotting functions to indicate that the x-axis should be displayed on a **logarithmic** scale instead of a **linear** scale.

A logarithmic scale is a way to display data that covers a wide range of values, such as orders of magnitude, in a compact and visually informative way. In a **logarithmic scale**, the distance between two tick marks is proportional to the logarithm of the values they represent, rather than to the values themselves. This means that small changes in values near the lower end of the scale are more clearly visible than small changes near the upper end of the scale.

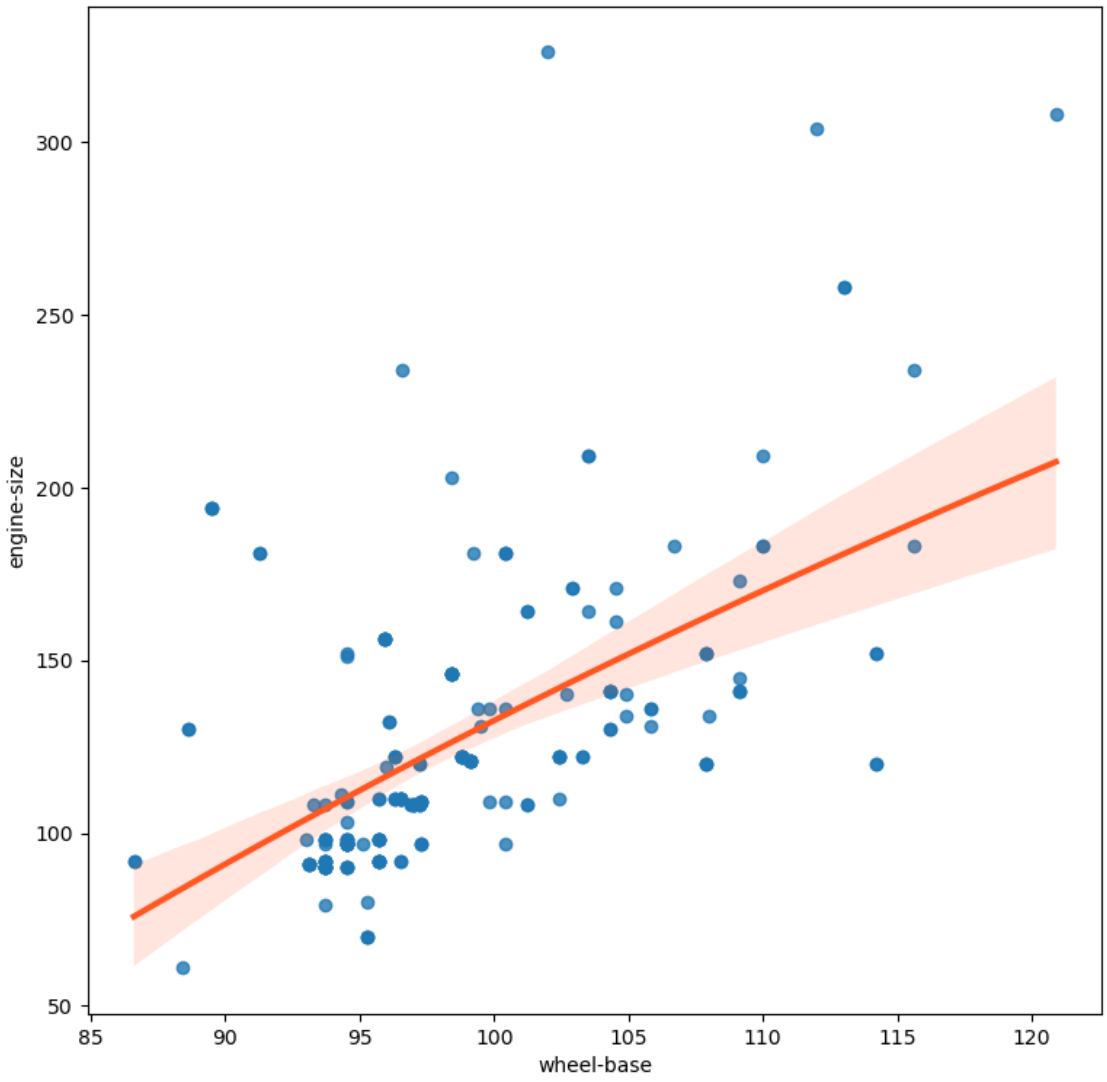
When **logx=True**, the x-axis of a plot is displayed on a logarithmic scale, which can be useful for visualizing data that covers a wide range of values or that exhibits **exponential growth** or decay. For example, a plot of population growth over time might benefit from a logarithmic scale on the x-axis, since the population will likely grow exponentially over time.

It's important to note that a logarithmic scale can distort the appearance of the data, particularly if there are large gaps in the values or if there are negative or zero values in the data. It's also important to label the axes and tick marks clearly to avoid misinterpretation of the data.

[153]:
```
plt.figure(figsize=(9,9))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"] , color='#FF6600' ,␣
 ↪logx=True)
plt.show()
```



[154]:
```
plt.figure(figsize=(9,9))
sns.regplot(x=db["wheel-base"] , y=db["engine-size"] , logx=True ,␣
 ↪line_kws={"color":"#FF5722","lw":3})
plt.show()
```



sns.lmplot(x="city-mpg", y="engine-size", data=db) is a line of code that uses the seaborn library to create a scatter plot with a fitted regression line (i.e., a **linear model** plot) between two variables, **"city-mpg"** and **"engine-size"**.

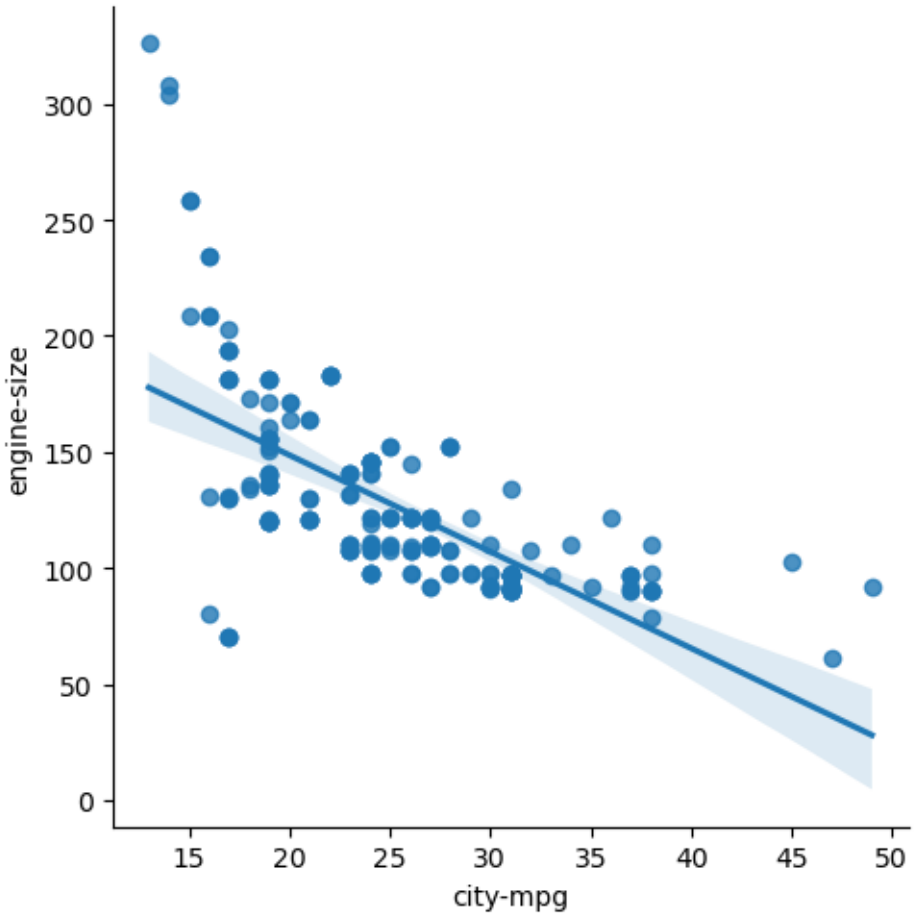
The **lmplot()** function in seaborn is a convenient way to create a scatter plot with a fitted regression line that describes the relationship between the two variables. The regression line is fitted using a linear model that estimates the slope and intercept of the line that best fits the data.

This type of visualization can be useful for exploring the relationship between two variables in a dataset, and for identifying patterns or trends in the data. It can also be used to evaluate the

strength and direction of the relationship between the variables, and to make predictions about the value of one variable based on the value of the other variable.

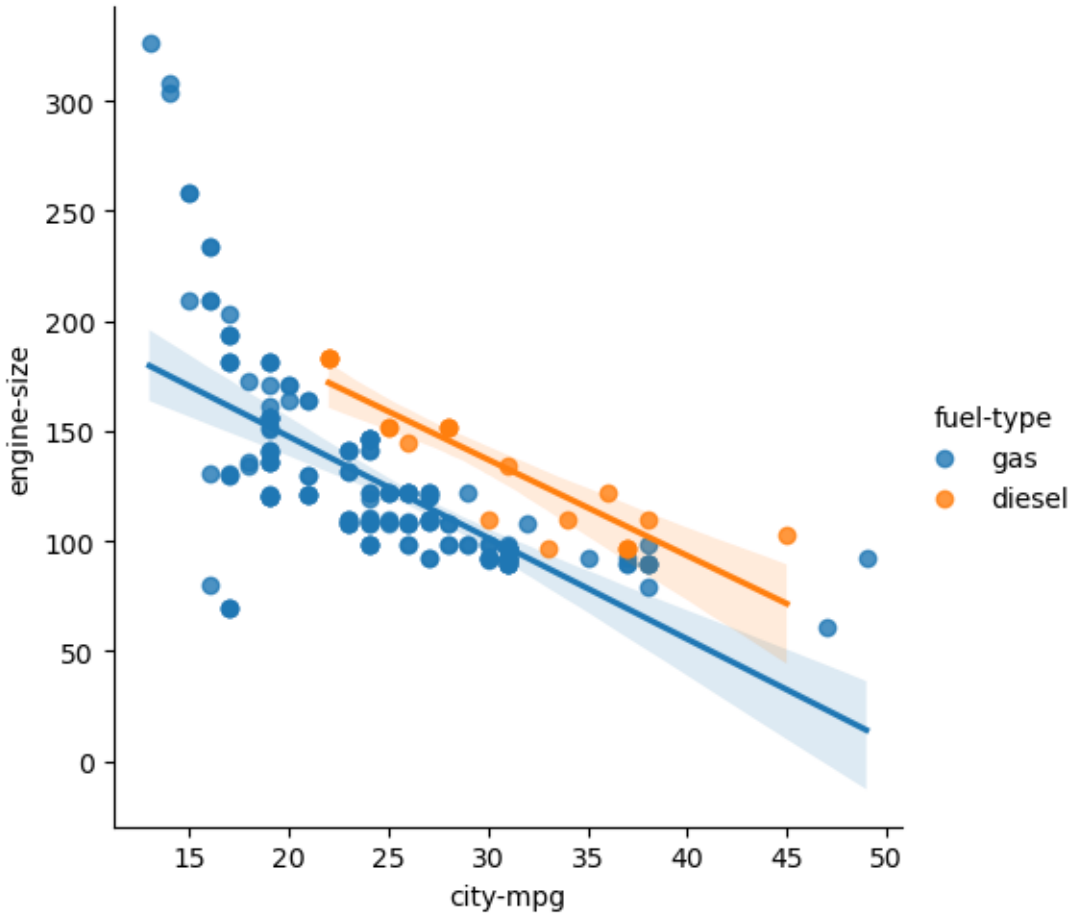
[155]:
```
sns.lmplot(x="city-mpg", y="engine-size",data=db)
```

[155]: <seaborn.axisgrid.FacetGrid at 0x2171d2e9910>

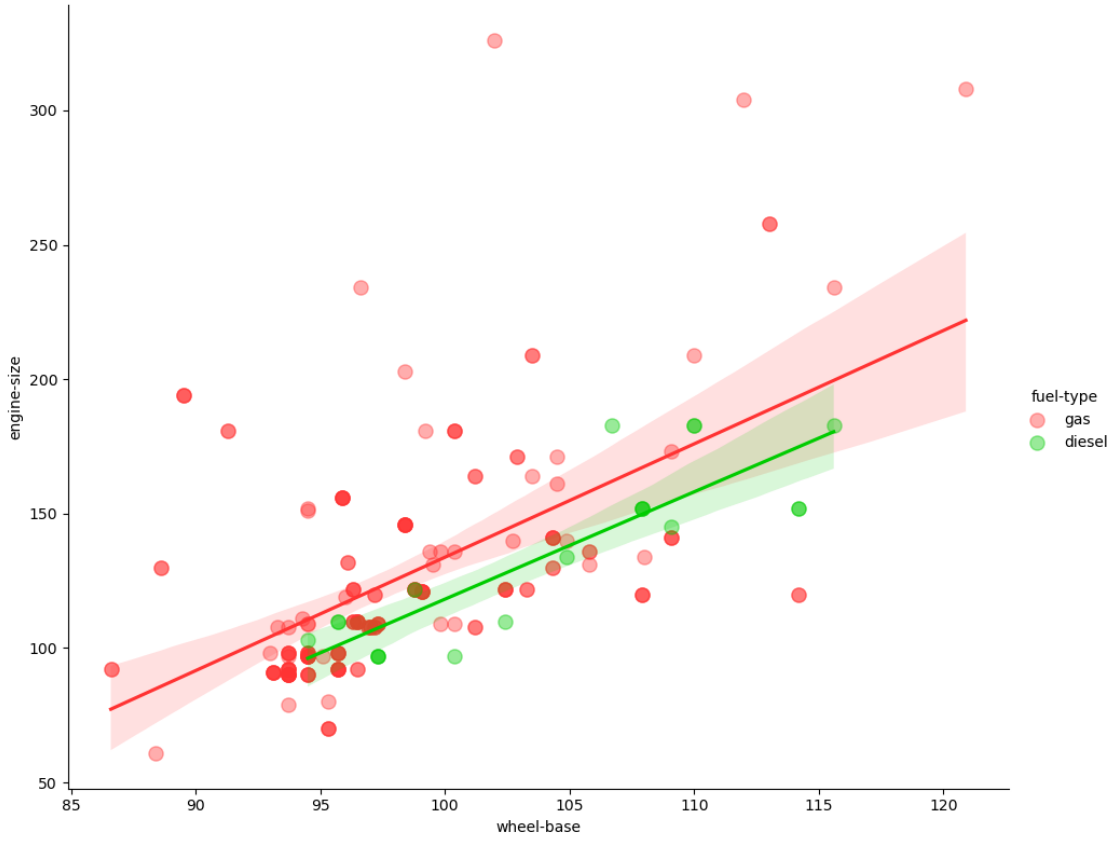


[156]:
```
sns.lmplot(x="city-mpg", y="engine-size",hue="fuel-type", data=db)
```

[156]: <seaborn.axisgrid.FacetGrid at 0x21719e9d5e0>

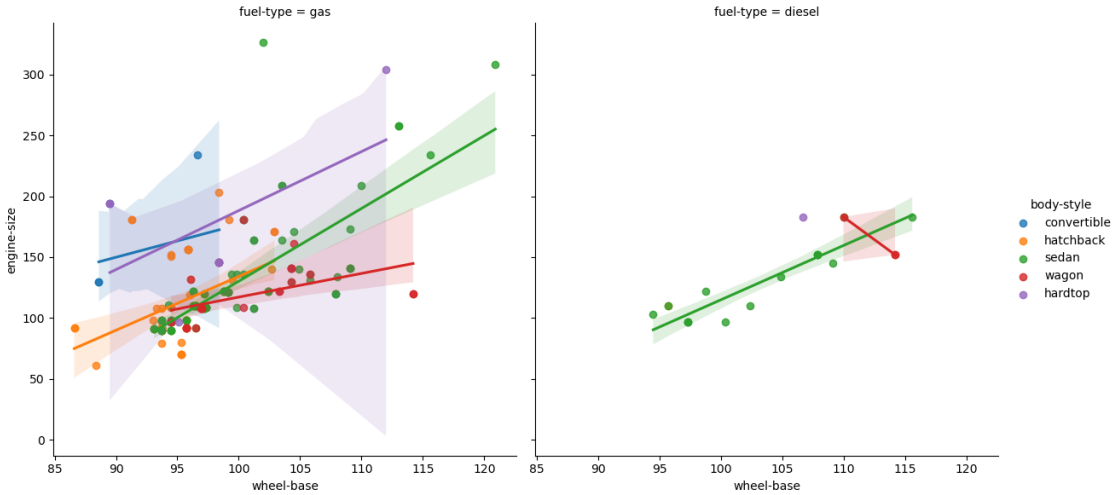


[157]:
```
sns.lmplot(x="wheel-base" , y="engine-size" , hue="fuel-type" , data=db ,␣
↪height=8,aspect=1.2 ,scatter_kws ={'s':90 , 'alpha' : .4} ,␣
↪palette=["#FF3333" ,"#00CC00"])
plt.show()
```



[158]:
```
sns.lmplot(x="wheel-base", y="engine-size", hue="body-style", col="fuel-type",␣
↪data=db ,height=6,aspect=1)
```

[158]: <seaborn.axisgrid.FacetGrid at 0x217197642e0>



1.17 Point plots

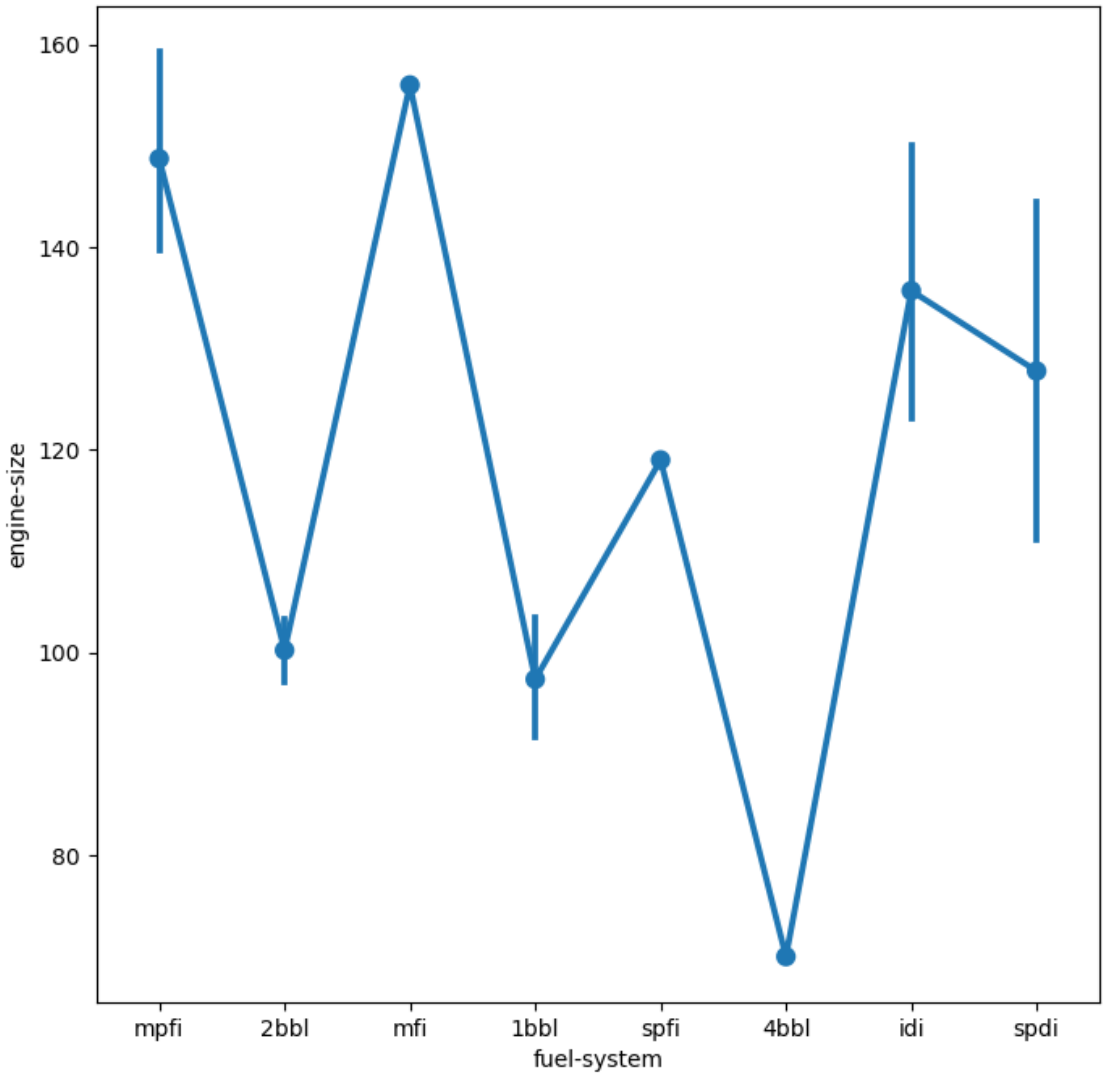
A point plot is a type of data visualization that displays the **average value** (e.g., mean, median, etc.) of a numerical variable for each category of a categorical variable, along with an indication of the **variability** (e.g., confidence intervals, standard deviation, etc.) of the data within each category. Point plots are often used to compare the average values of a numerical variable across different groups or categories.

In a point plot, the data are represented by points, which are placed at the average value of the numerical variable for each category. **Error bars** or **confidence intervals** are often included on the plot to indicate the variability of the data within each category.

Point plots can be created using various data visualization libraries in Python, including seaborn and matplotlib. Seaborn's **pointplot()** function, for example, is a convenient way to create a point plot that displays the average value of a numerical variable for each category of a categorical variable, with confidence intervals or standard deviation bars displayed on the plot. Other types of plots that are similar to point plots include bar plots, which display the average value of a numerical variable for each category as a bar, and box plots, which display the distribution of the data within each category using **boxes and whiskers**.

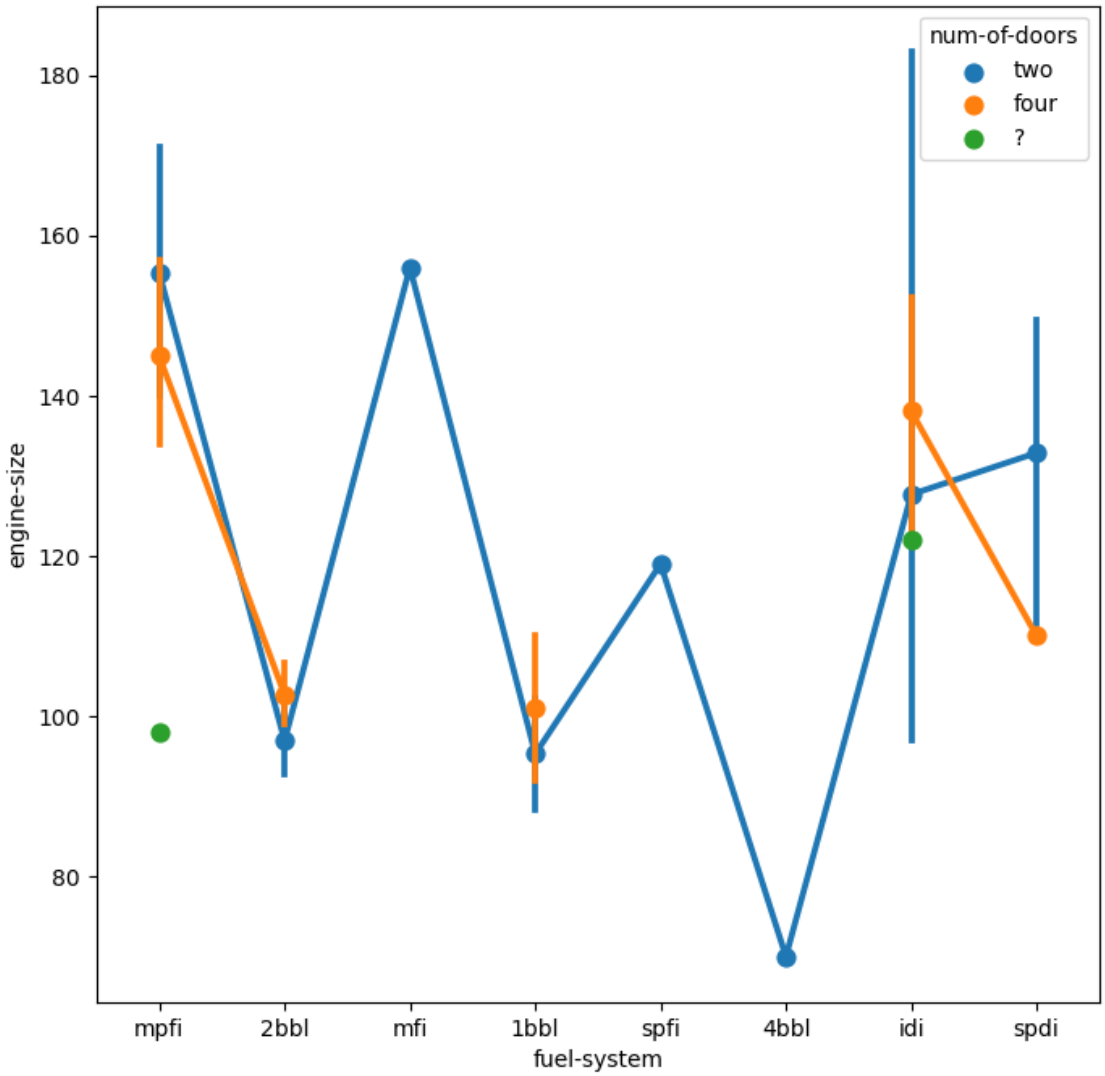
[159]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



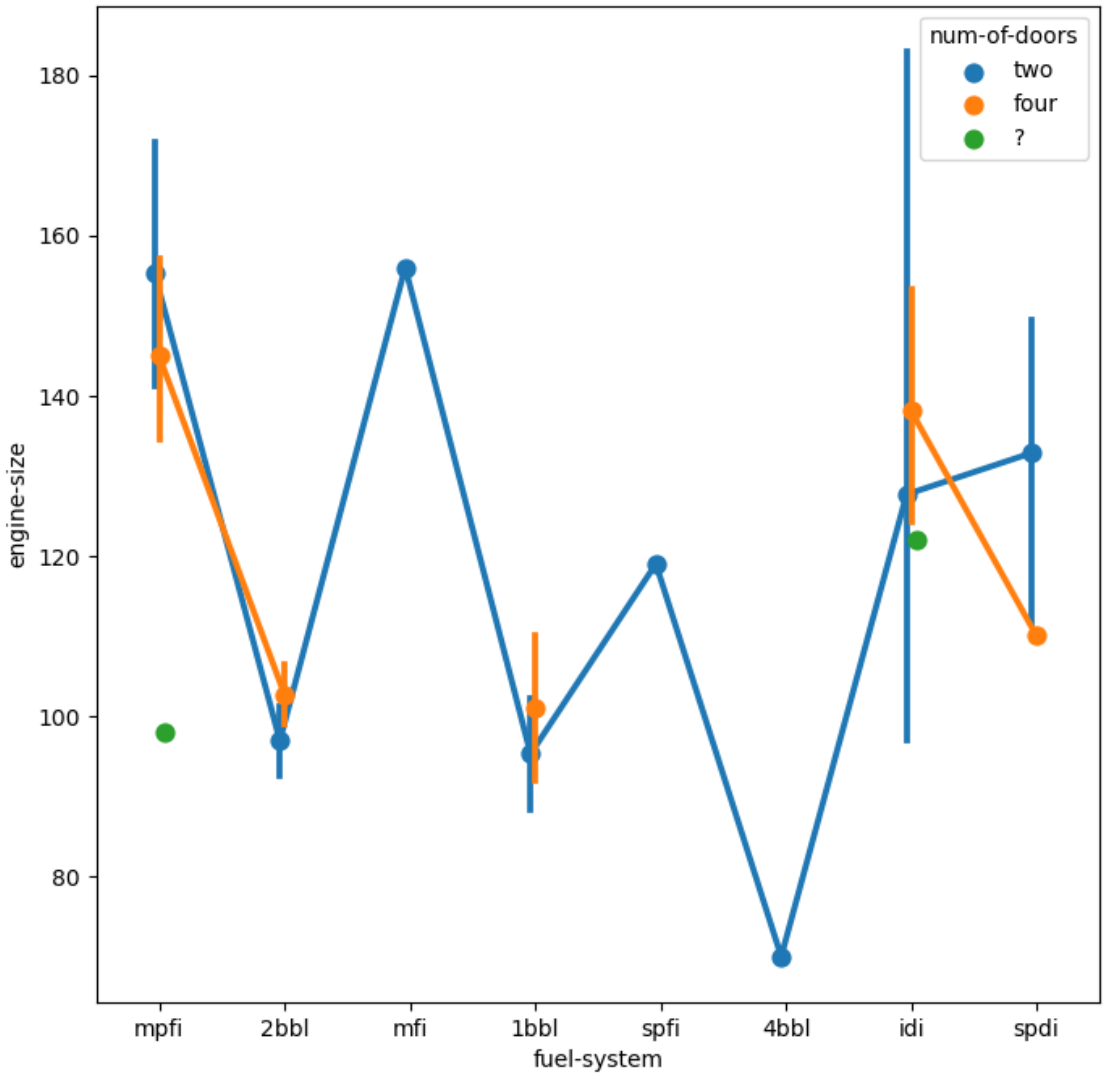
[160]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'], hue=db['num-of-doors'])
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[161]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'],␣
 ↪hue=db['num-of-doors'],dodge=True)
plt.show()
```

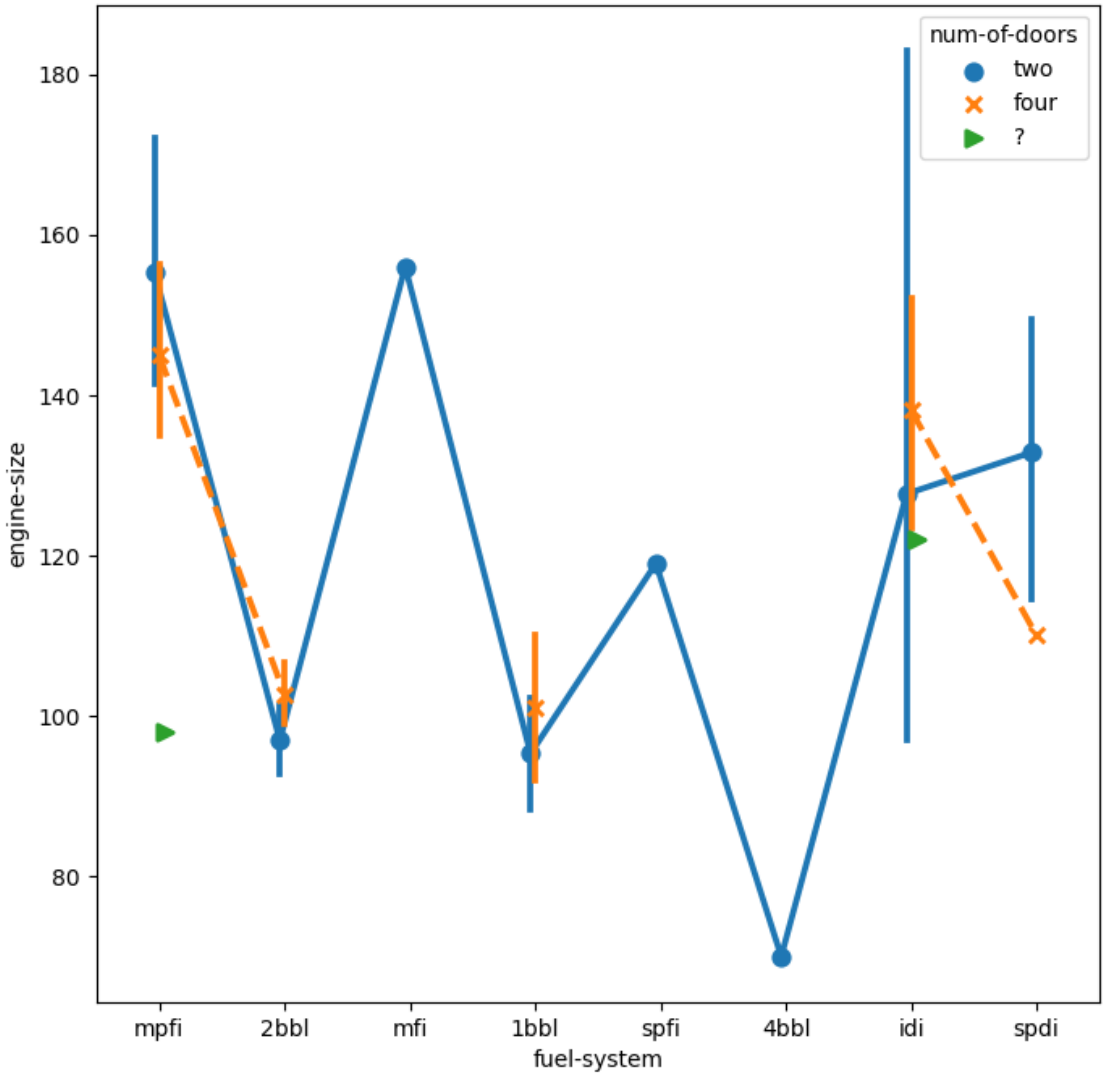
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[162]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'], hue=db['num-of-doors'],␣
 ↪dodge=True, markers=["o", "x", ">"], linestyles=["-", "--", "--"])
plt.show()
```

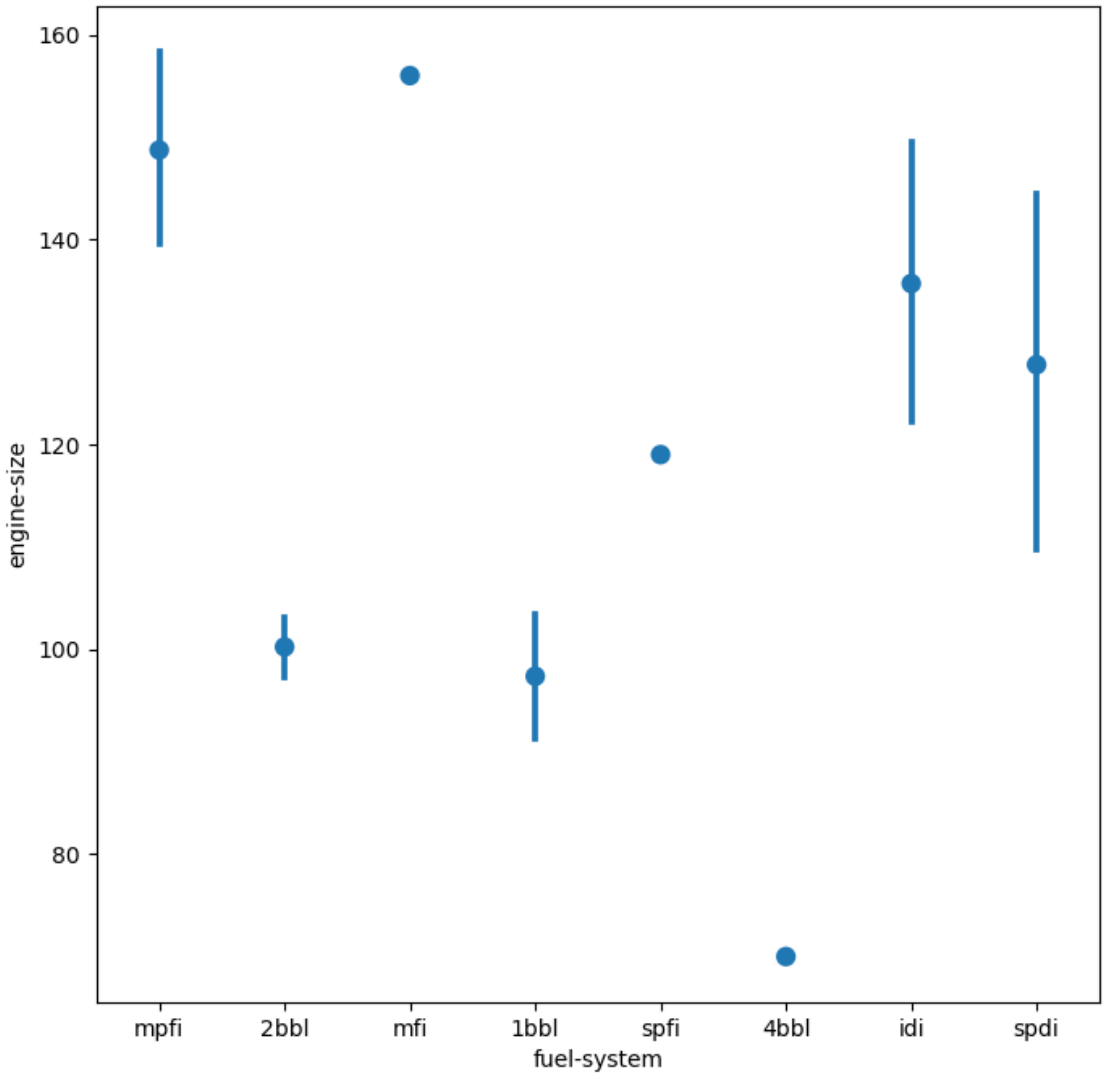
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\categorical.py:1781:
UserWarning: You passed a edgecolor/edgecolors ((1.0, 0.4980392156862745,
0.054901960784313725)) for an unfilled marker ('x').  Matplotlib is ignoring the
```

```
edgecolor in favor of the facecolor.  This behavior may change in the future.
  ax.scatter(x, y, label=hue_level,
```



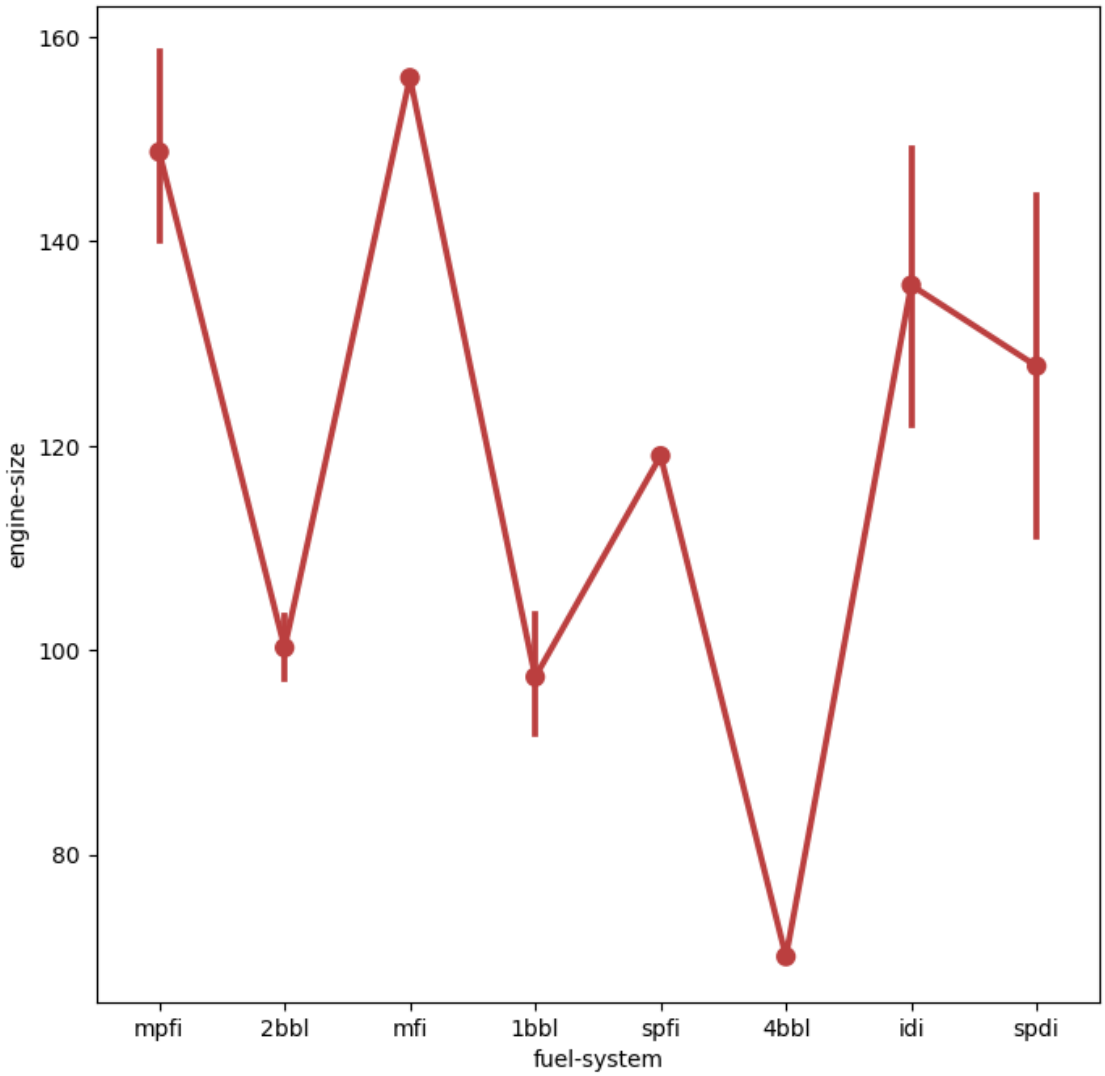
[163]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'],join=False)
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



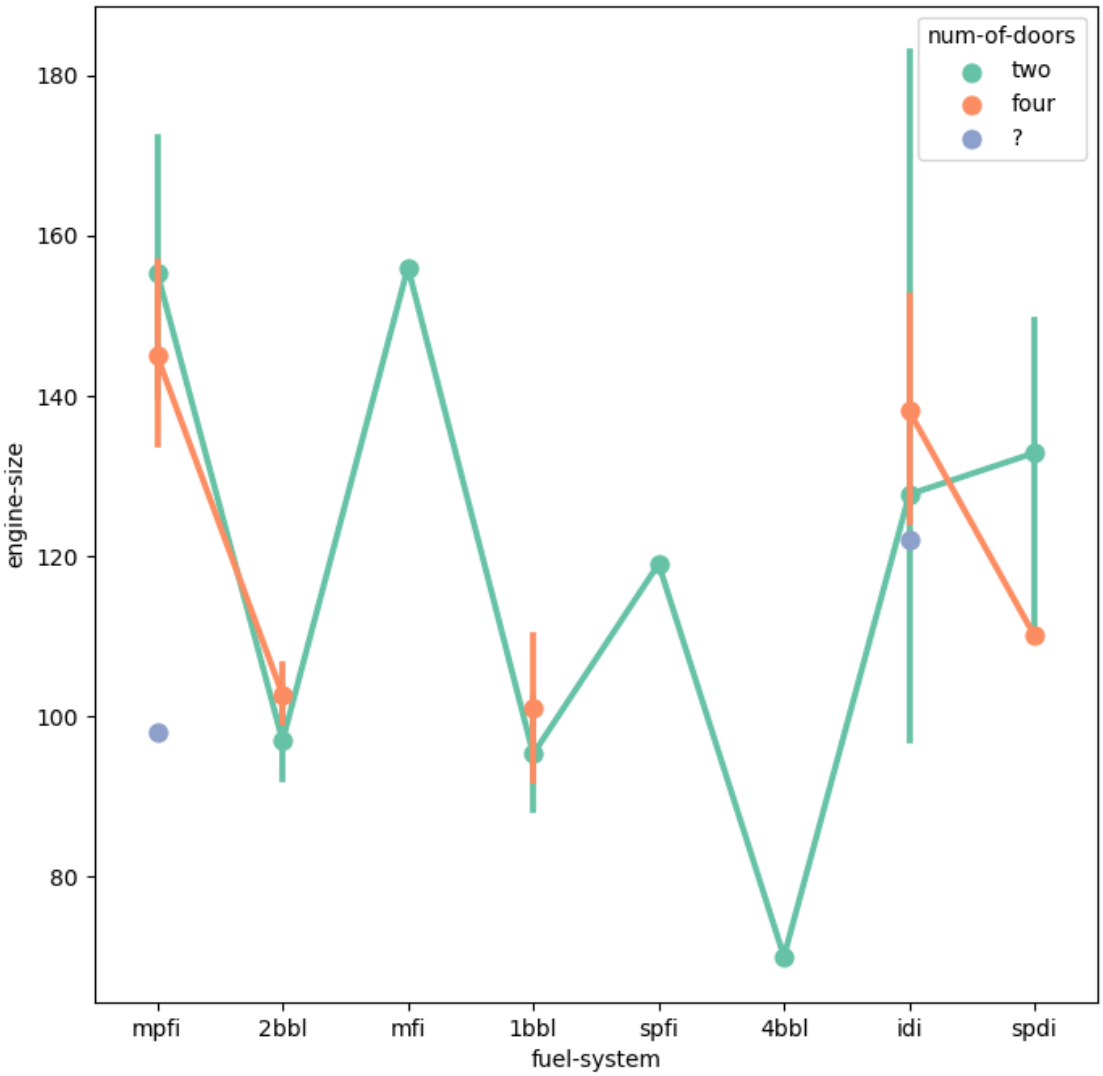
[164]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'],color="#bb3f3f")
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



[165]:
```
plt.figure(figsize=(8,8))
sns.pointplot(db['fuel-system'], db['engine-size'], hue=db['num-of-doors'],␣
 ↪palette="Set2")
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```



1.18 FacetGrid

FacetGrid is a powerful tool in the Seaborn library that allows you to create multiple small plots or visualizations based on the subsets of the data using one or more categorical variables. It is a way of visualizing a dataset with multiple subplots based on the levels of one or more categorical variables.

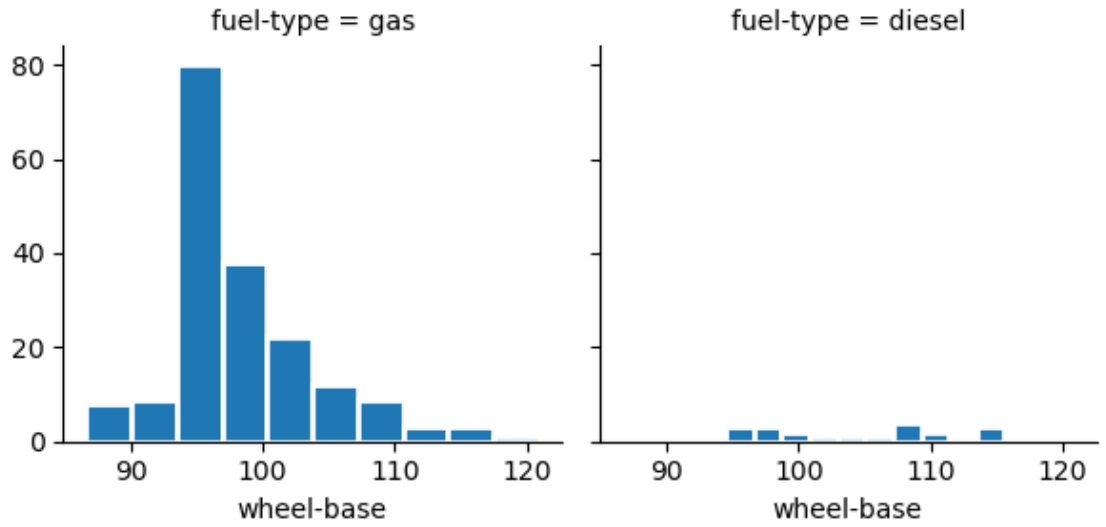
The FacetGrid object is created using the Seaborn **FacetGrid()** function, which takes a pandas DataFrame and one or more categorical variables as arguments. The FacetGrid can then be used to create different types of plots, such as scatterplots, line plots, or histograms, by calling the appropriate plotting function on the FacetGrid object.

[166]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
plt.rcParams[ 'axes.labelsize'] = 10
```

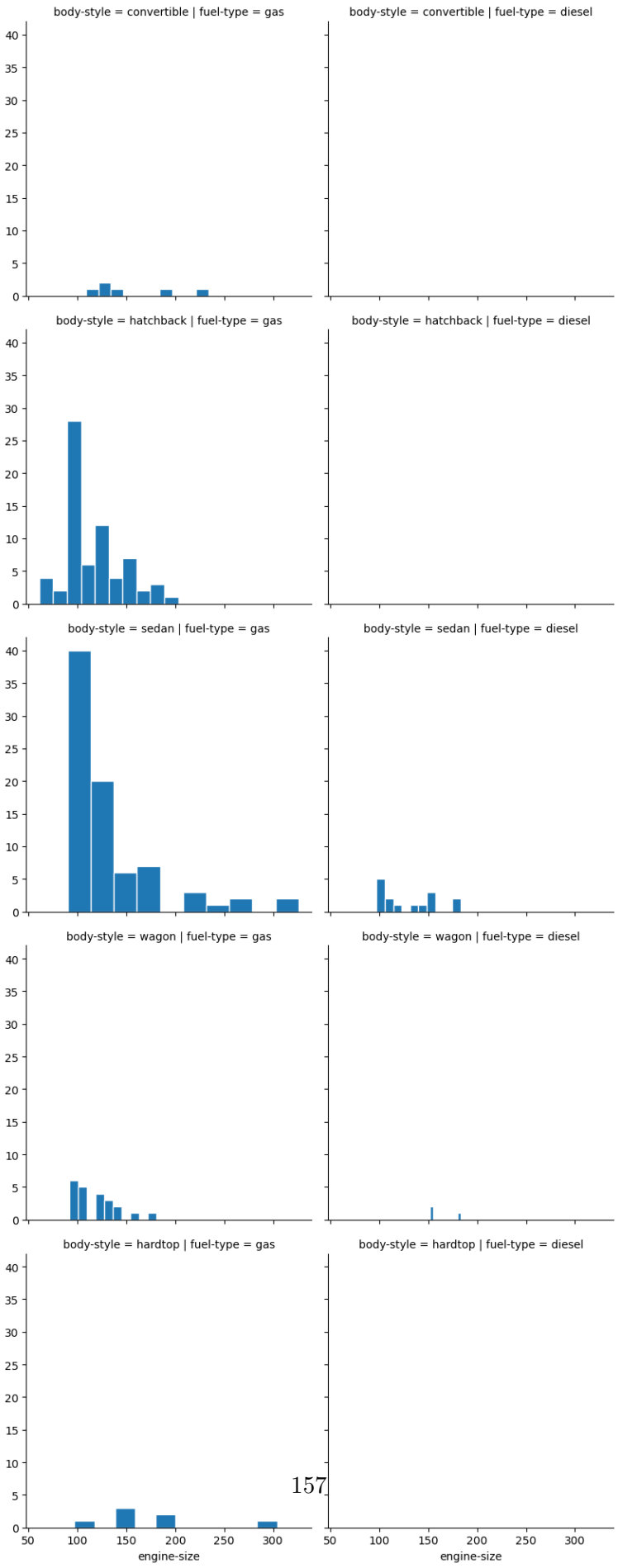
```
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10
```

[167]:
```
plt.figure(figsize=(8,8))
g = sns.FacetGrid(db, col="fuel-type")
g = g.map(plt.hist, "wheel-base", edgecolor ='w', linewidth=2)
plt.show()
```

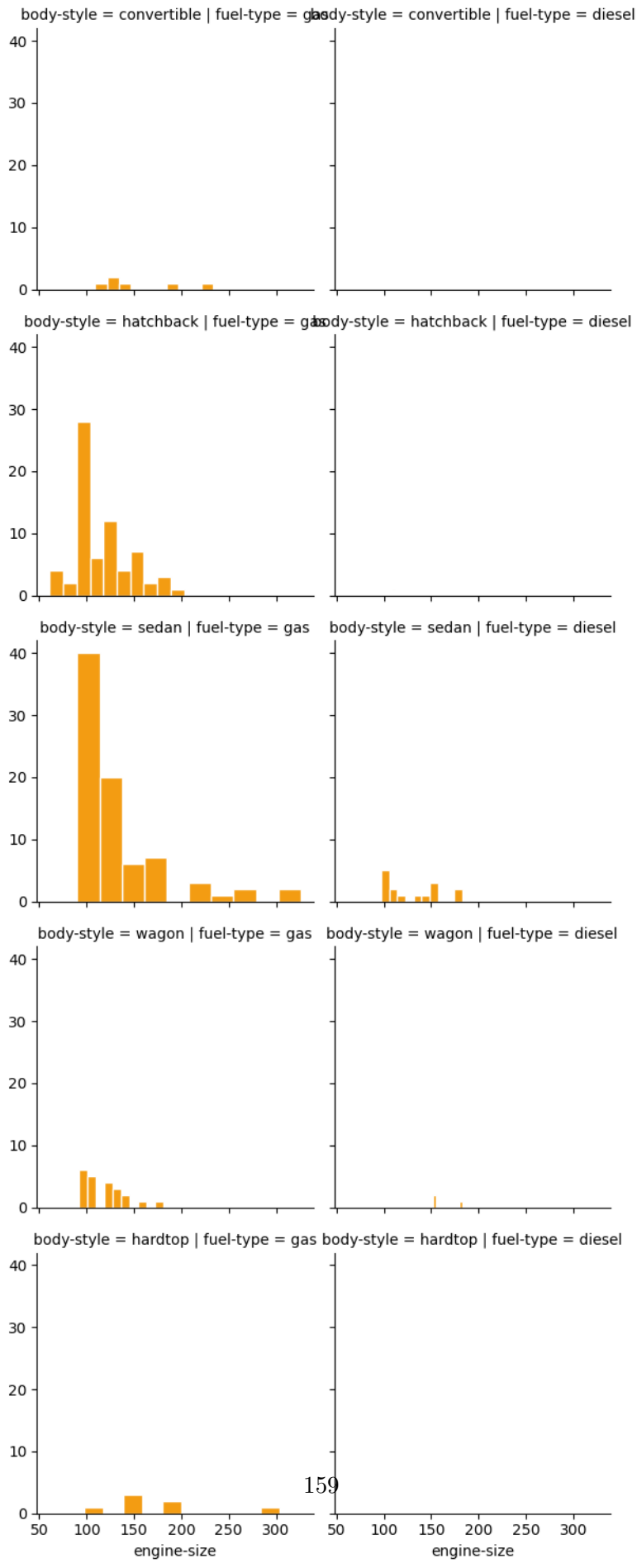
```
<Figure size 800x800 with 0 Axes>
```



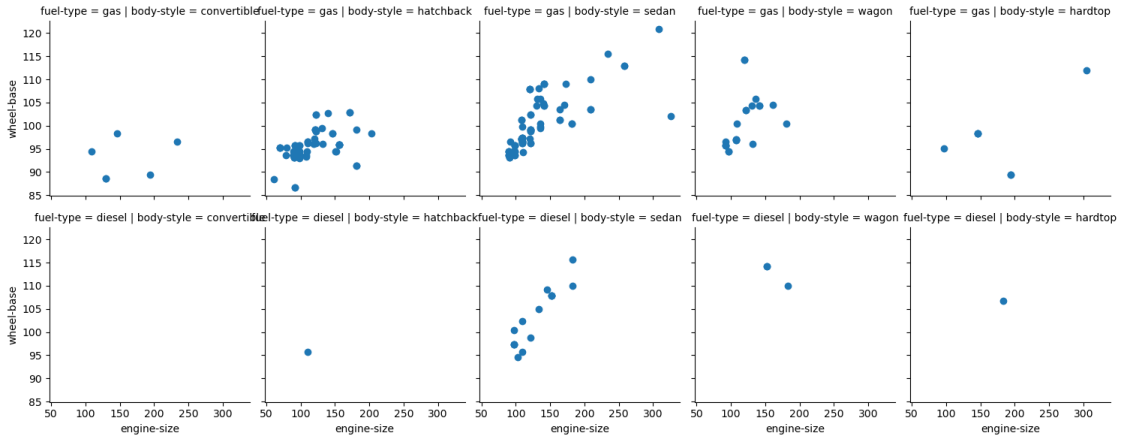
[168]:
```
g = sns.FacetGrid(db, col="fuel-type" , row = "body-style" , height=4, aspect=1)
g = g.map(plt.hist, "engine-size",edgecolor ='w', linewidth=1)
plt.show()
```


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
0 5 10 15 20 25 30 35 40
50 100 150 200 250 300
engine-size
engine-size

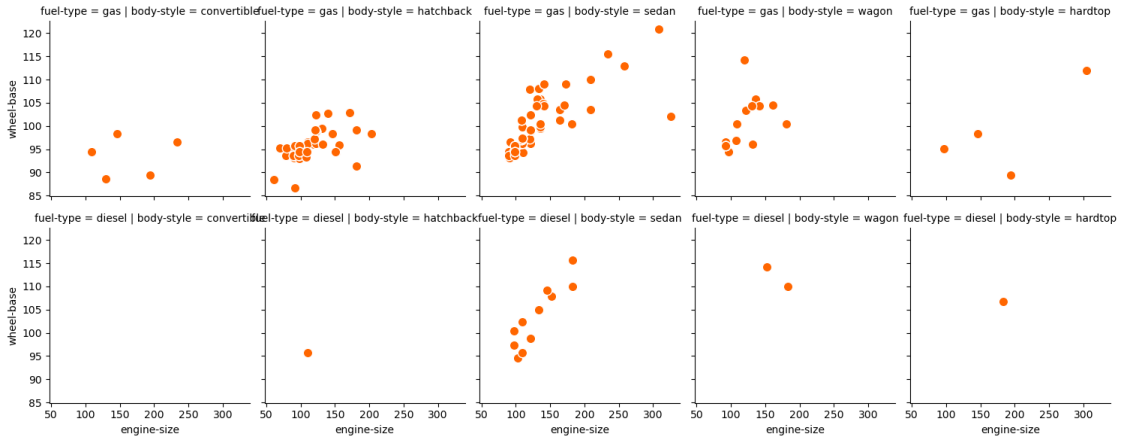
[169]:
```
g = sns.FacetGrid(db, col="fuel-type" , row = "body-style" , height=3, aspect=1)
g = g.map(plt.hist, "engine-size" , color = '#F39C12',edgecolor ='w',␣
 ↪linewidth=1)
plt.show()
```


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
0
10
20
30
40
50
100
150
200
250
300
engine-size

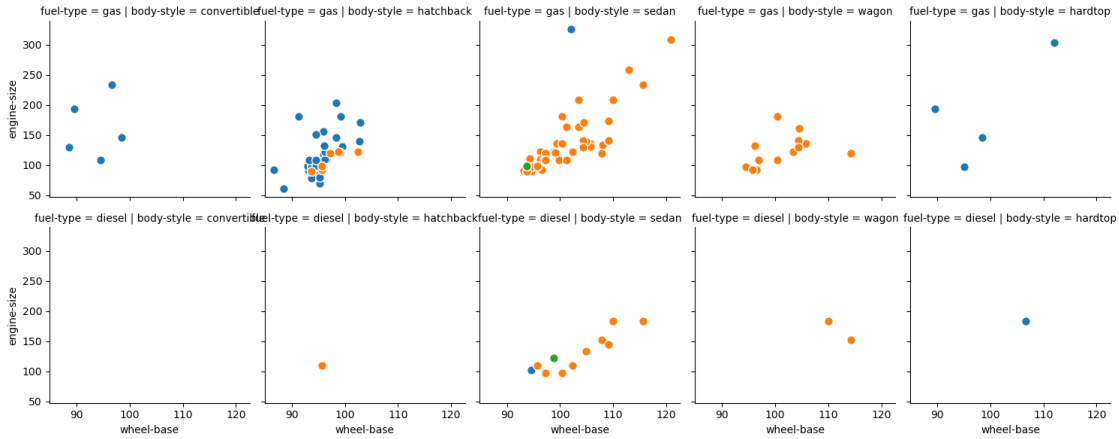
[170]:
```
g = sns.FacetGrid(db, col="body-style" , row = "fuel-type" , height=3, aspect=1)
g = g.map(plt.scatter, "engine-size" , "wheel-base")
plt.show()
```



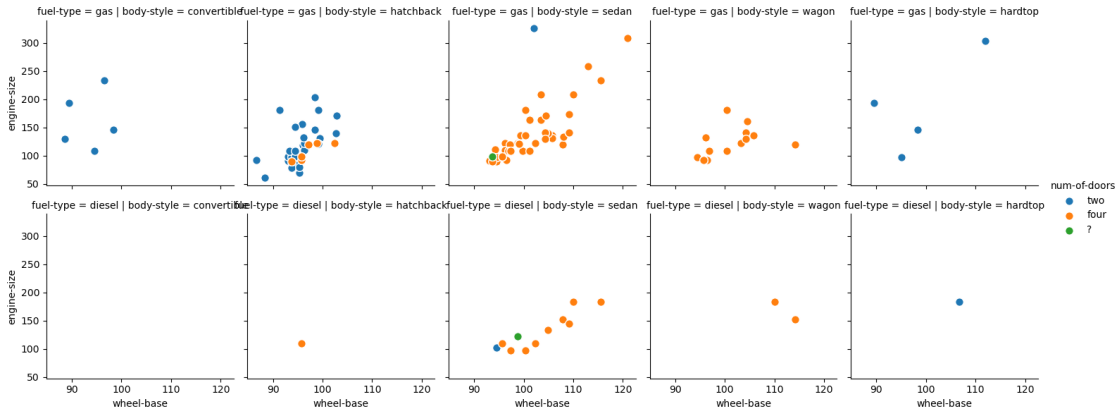
[171]:
```
g = sns.FacetGrid(db, col="body-style" , row = "fuel-type" , height=3, aspect=1)
g = g.map(plt.scatter, "engine-size" , "wheel-base" , color =␣
↪'#FF6600',edgecolor="w", s=80)
plt.show()
```



[172]:
```
g = sns.FacetGrid(db, 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)
plt.show()
```

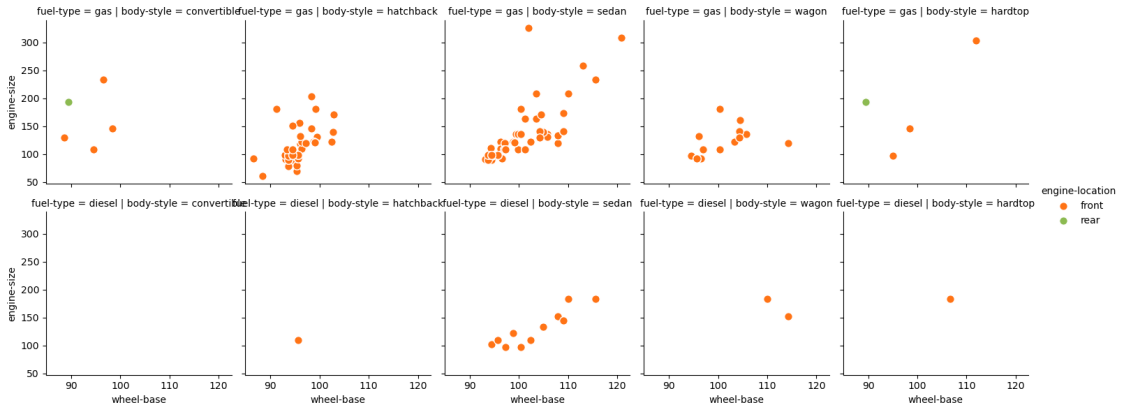


[173]:
```
g = sns.FacetGrid(db, col="body-style" , row = "fuel-type" , hue="num-of-doors"␣
↪,height=3, aspect=1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",edgecolor="w", s=70)
g.add_legend()
plt.show()
```

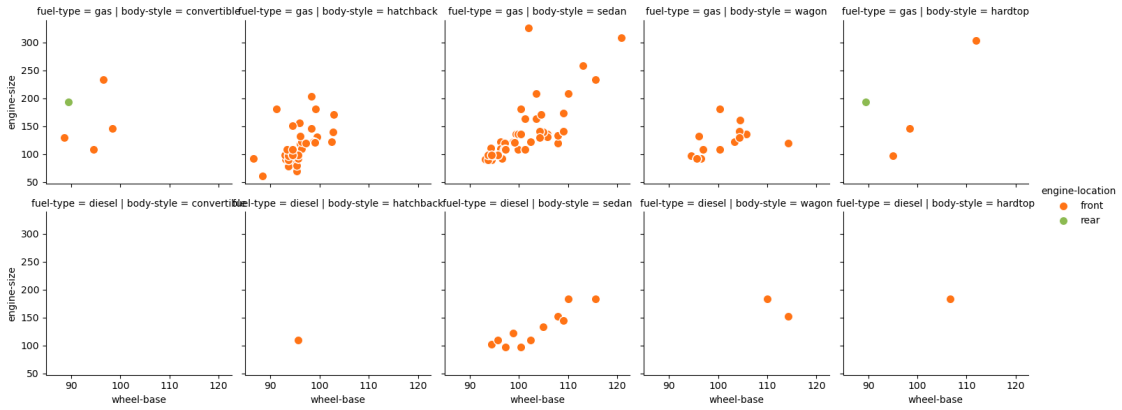


pal1 is a custom palette.

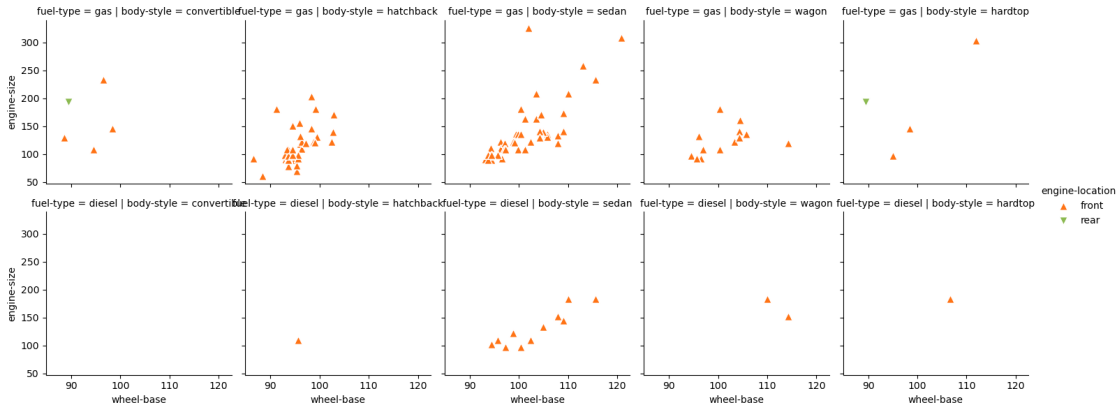
[174]:
```
pal1 = dict(front="#ff7315", rear="#8cba51")
g = sns.FacetGrid(db, col="body-style" , row = "fuel-type" ,␣
↪hue="engine-location" ,height=3, aspect=1,palette=pal1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",edgecolor="w", s=70)
g.add_legend()
plt.show()
```



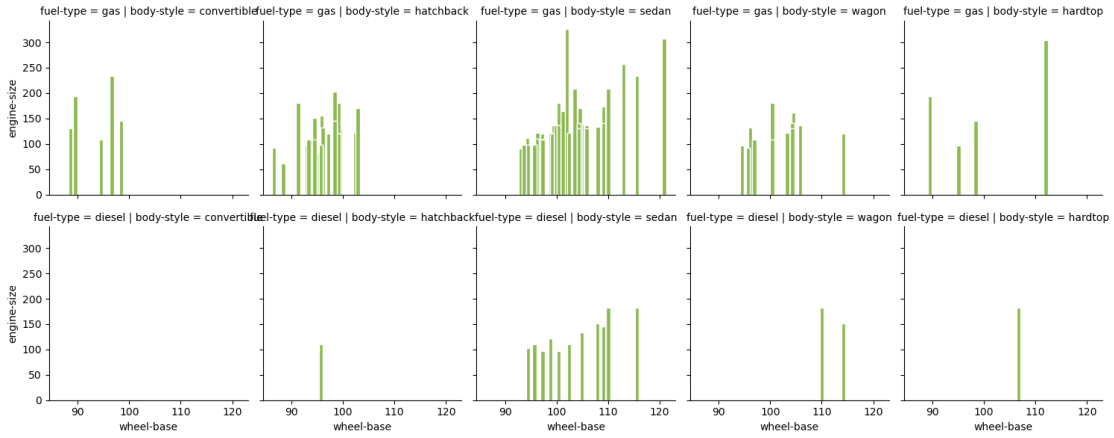
[175]:
```
pal1 = dict(front="#ff7315", rear="#8cba51")
kws = dict(s=80, linewidth=1, edgecolor="w")
kws1 = dict(height=3, aspect=1,palette=pal1)
g = sns.FacetGrid(db, col="body-style" , row = "fuel-type" ,␣
 ↪hue="engine-location" , **kws1)
g = g.map(plt.scatter, "wheel-base" , "engine-size",**kws)
g.add_legend()
plt.show()
```



[176]:
```
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(db, 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()
```



[177]:
```
g = sns.FacetGrid(db, 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()
plt.show()
```



1.19 Joint Plot

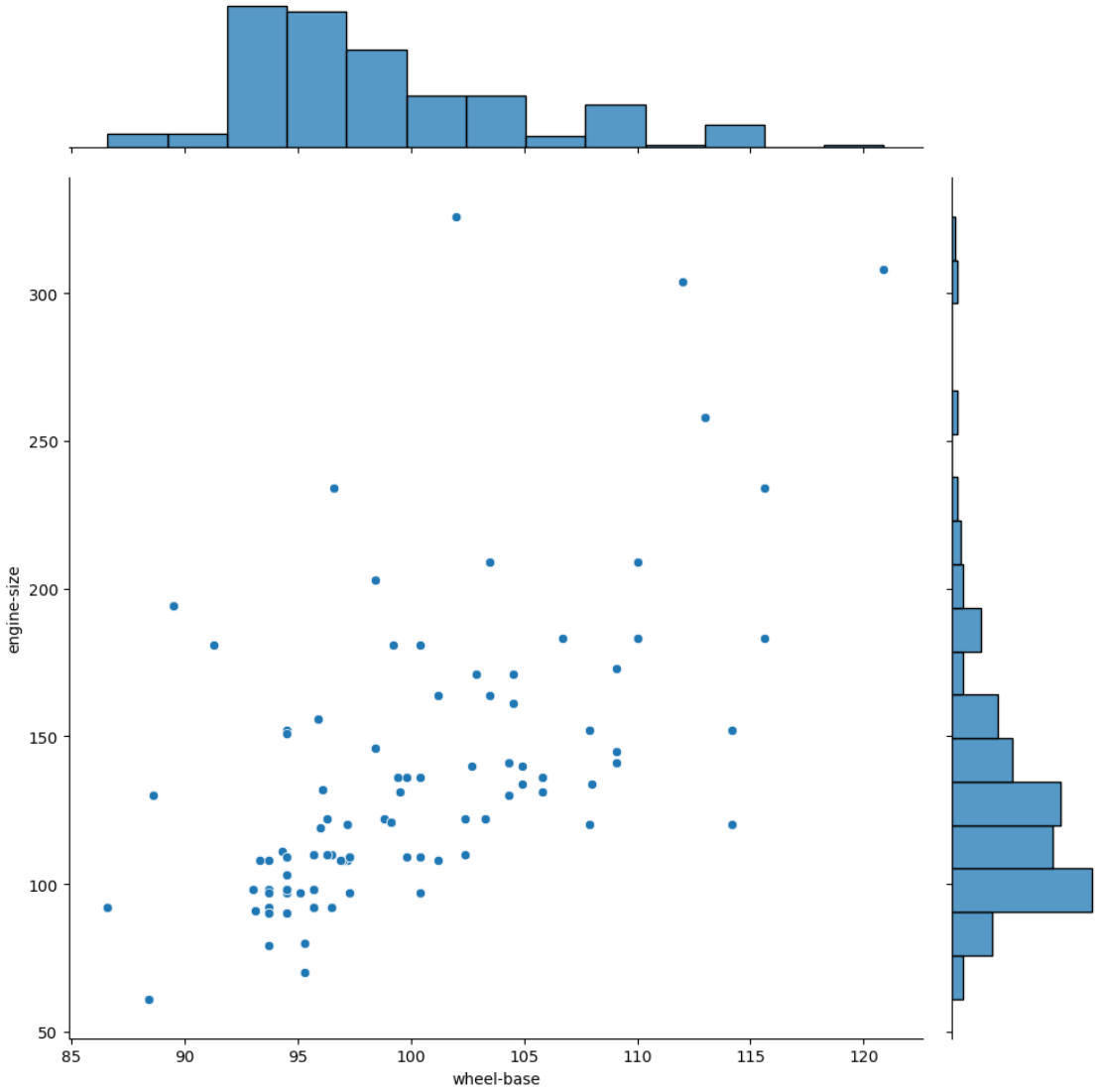
Joint plot is a type of plot in the Seaborn library that combines two different plots into a single figure: a **scatterplot and a histogram**. It is used to visualize the relationship between two variables, showing both their individual distributions and their joint distribution.

The joint plot function in Seaborn can be accessed using the **sns.jointplot()** function. The function takes two variables to be plotted as inputs, along with several optional parameters to customize the plot.

[178]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
```

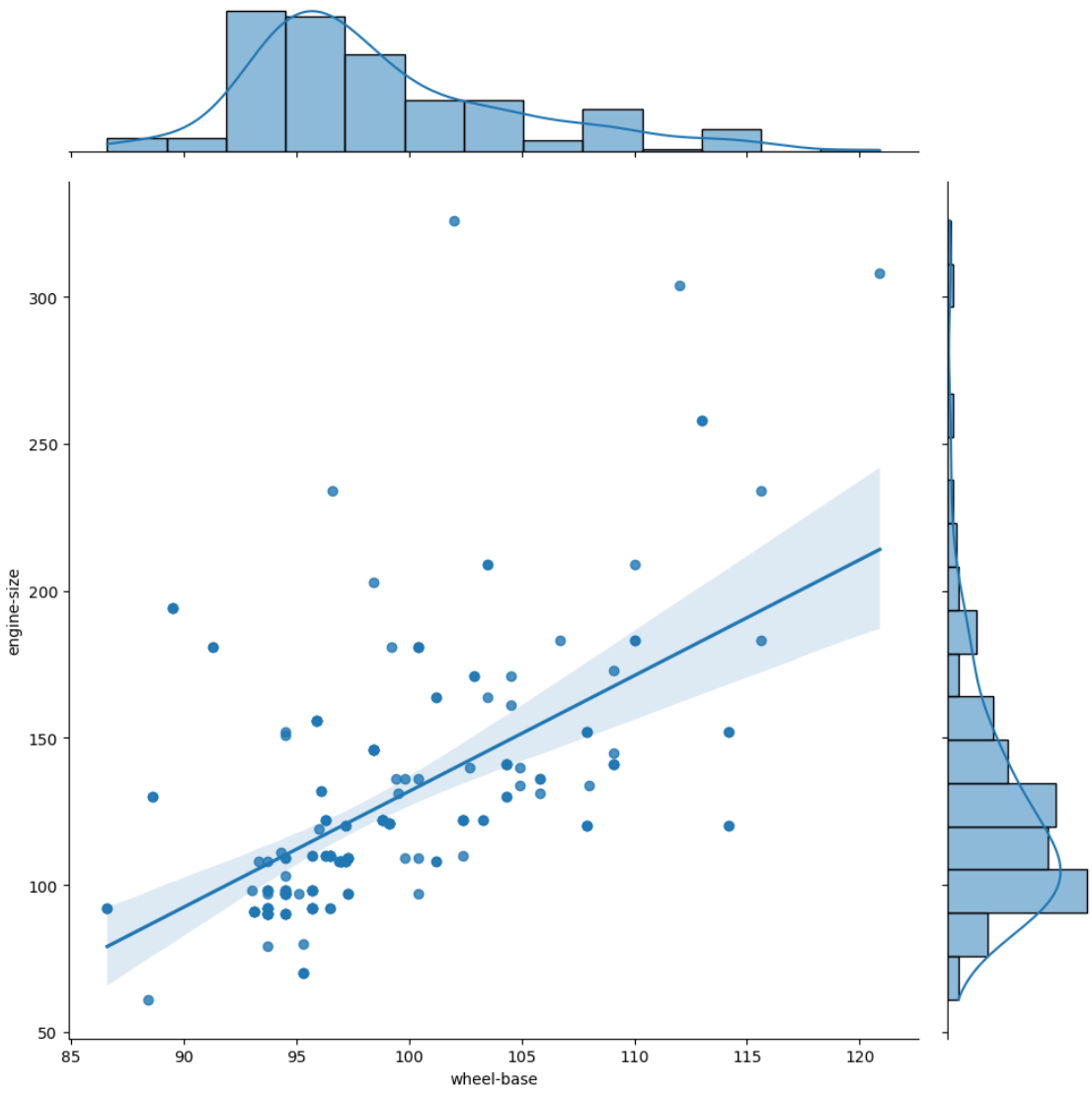
[179]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10)
```

[179]: <seaborn.axisgrid.JointGrid at 0x21721f3e3d0>



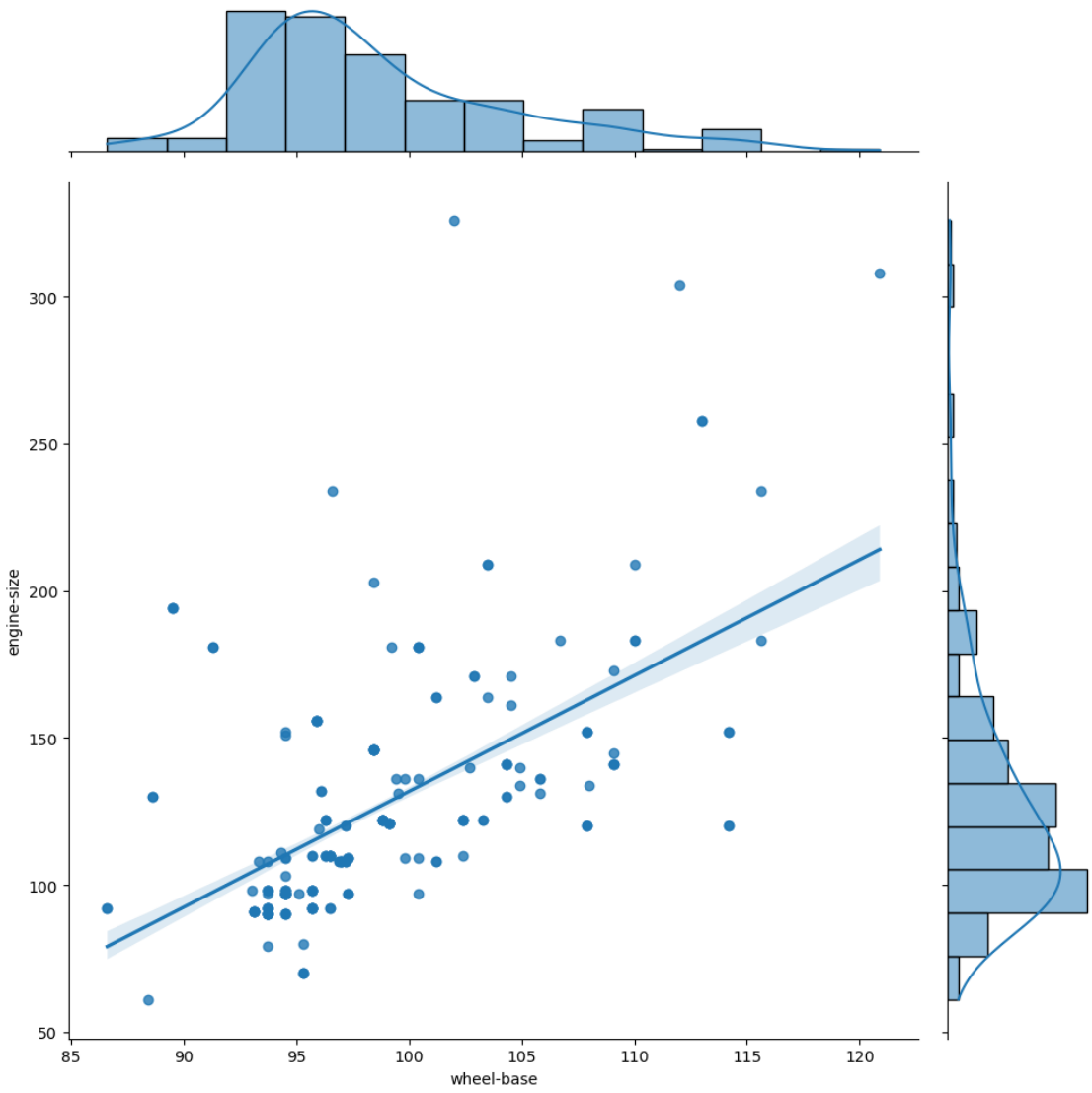
[180]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="reg")
```

[180]: <seaborn.axisgrid.JointGrid at 0x217216e7ee0>



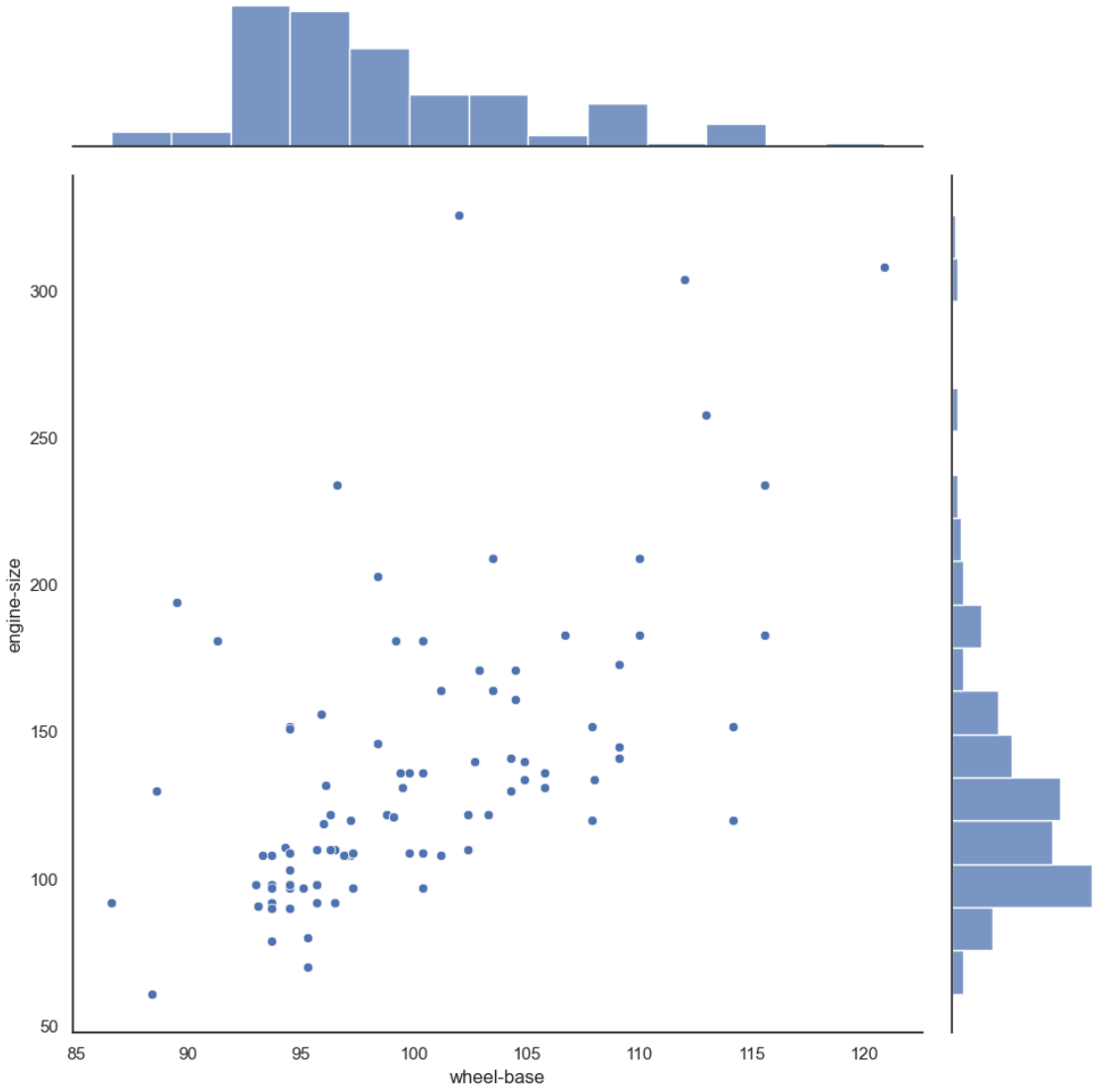
[181]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="reg", ci=50)
```

[181]: <seaborn.axisgrid.JointGrid at 0x2171c1c9b20>



[182]:
```
sns.set(style="white", color_codes=True)
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10)
```

[182]: <seaborn.axisgrid.JointGrid at 0x2171c09e7c0>

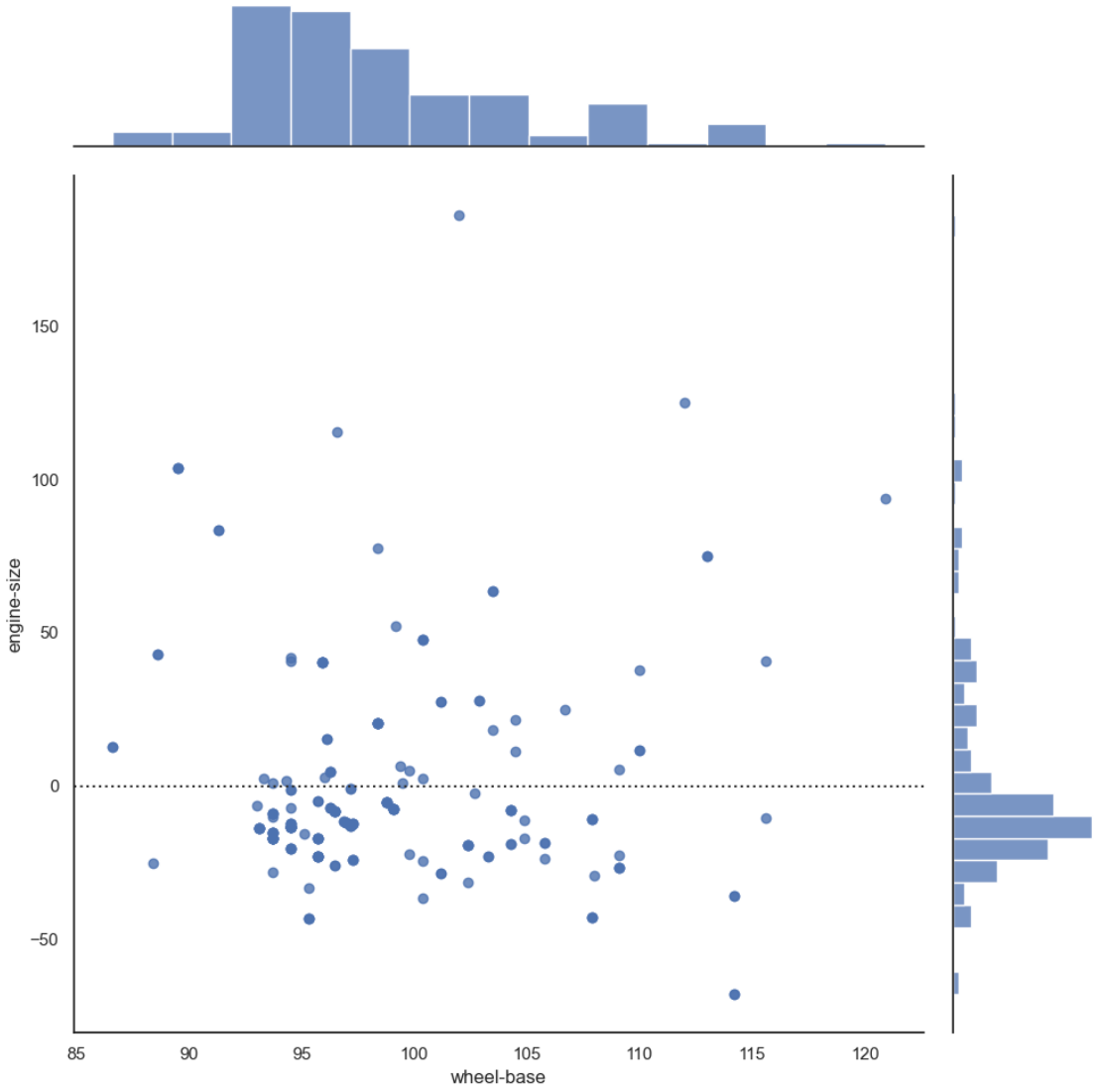


In Seaborn, the kind parameter in regression plots specifies the type of plot to be drawn. One of the options for kind is **"resid"**, which creates a **residual plot**.

A residual plot is a type of plot used to diagnose the fit of a **linear regression model**. It displays the **residuals**, which are the **differences** between the actual values and the predicted values, on the **y-axis**, and the predictor variable on the **x-axis**. The residuals should be randomly scattered around the x-axis with no discernible pattern. If there is a pattern, it suggests that the linear regression model may not be the best fit for the data.

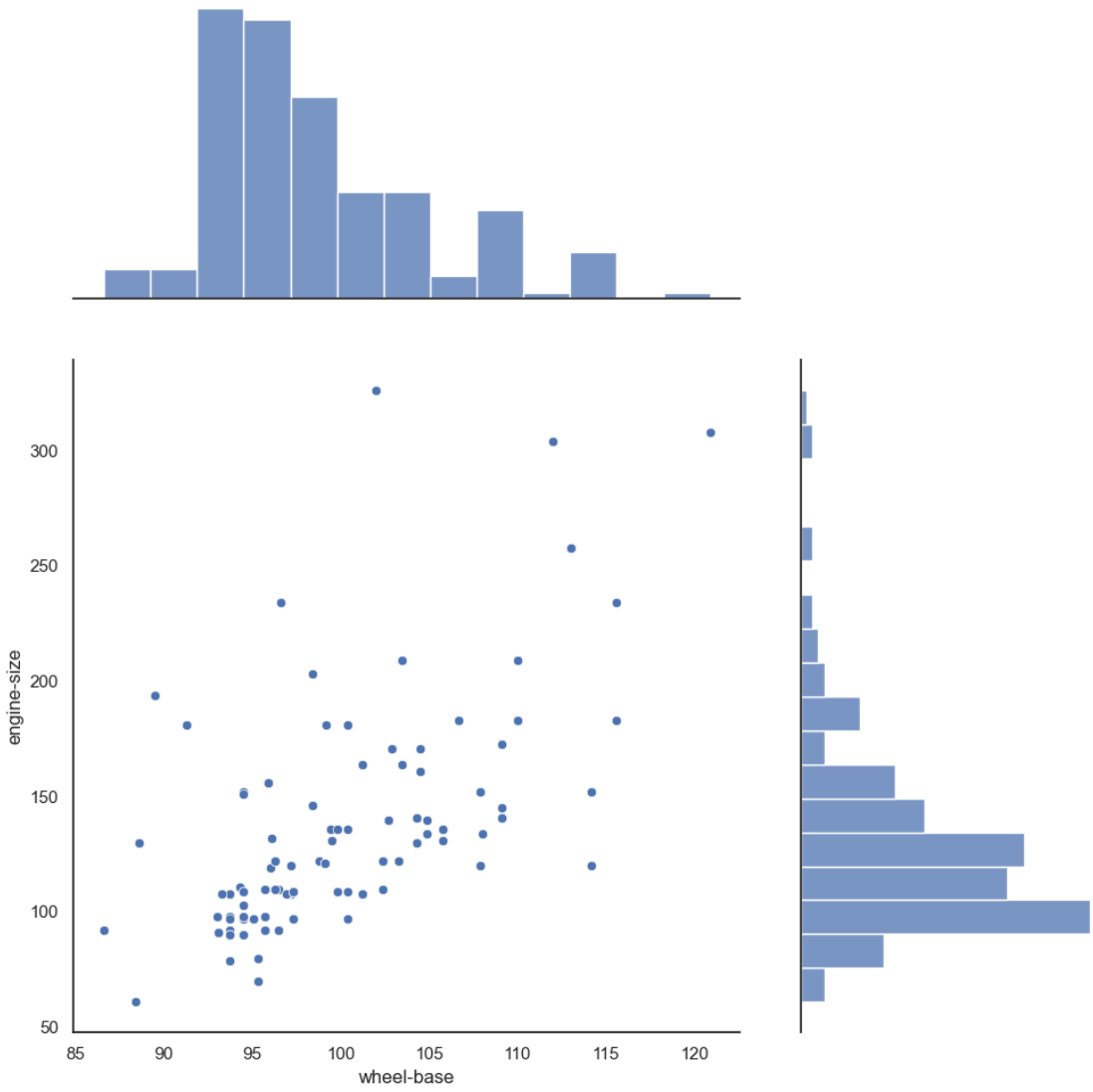
[183]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="resid")
```

[183]:
```
<seaborn.axisgrid.JointGrid at 0x21720fccca0>
```

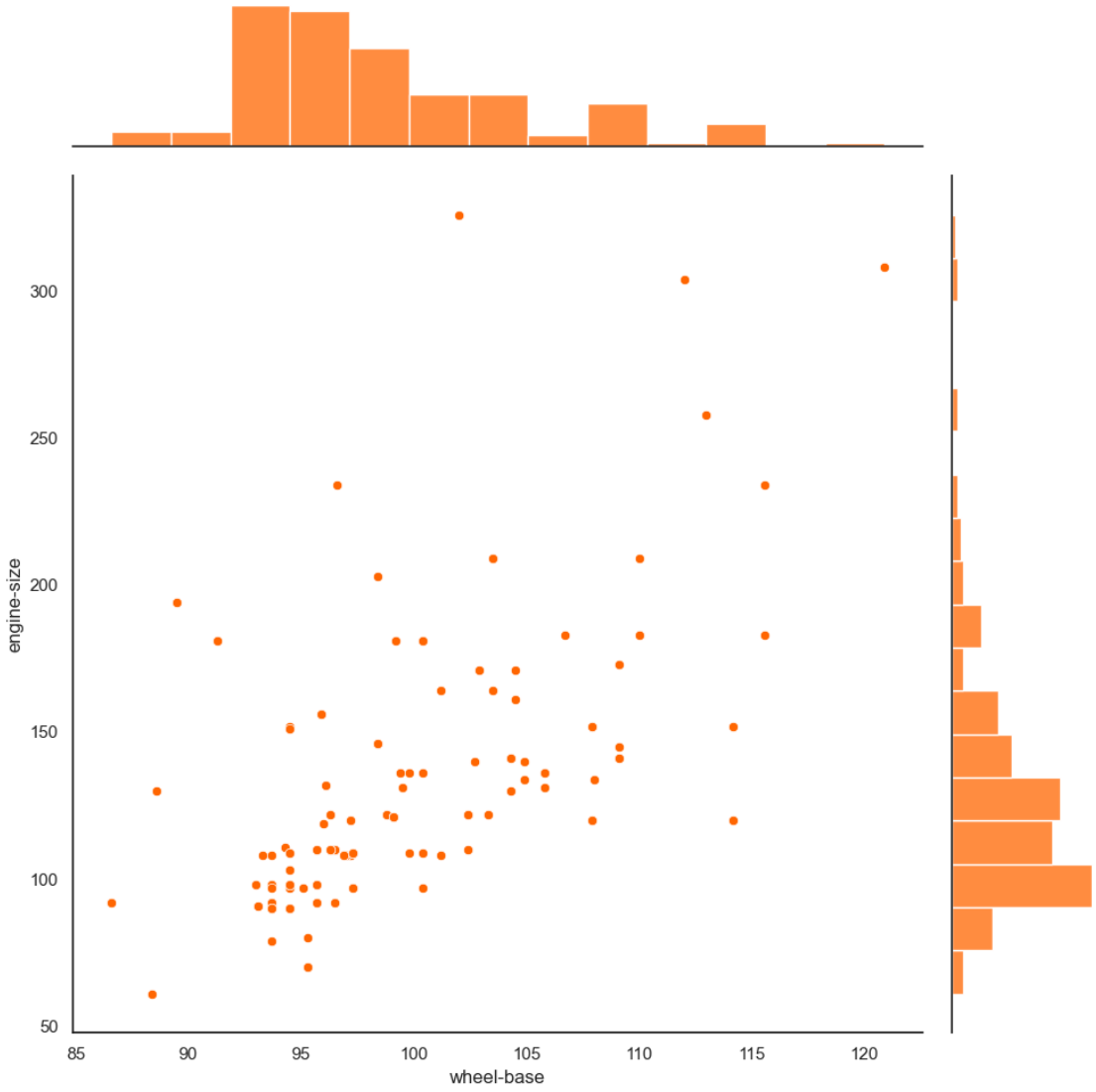


[184]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10, ratio=2)
```

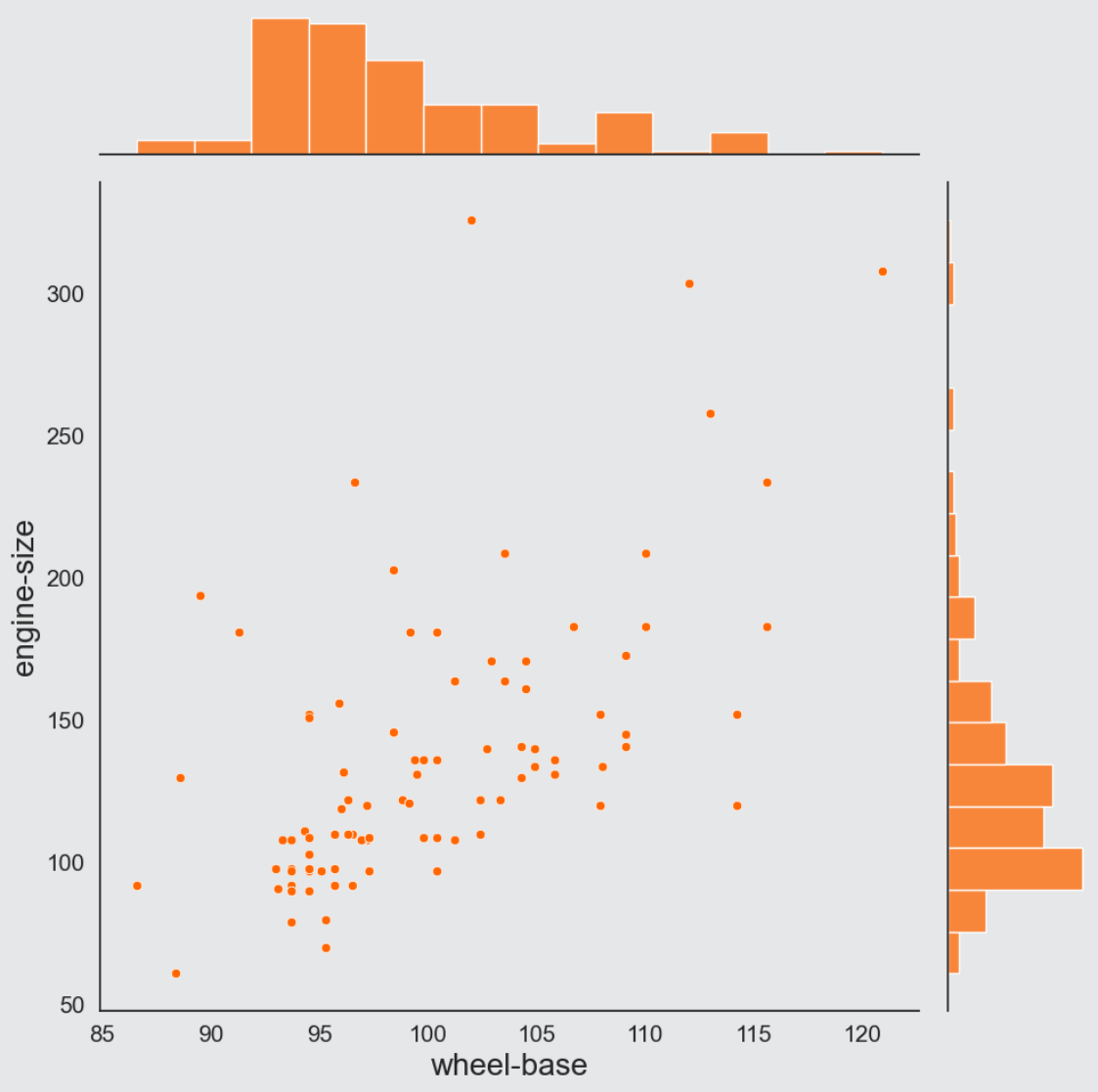
[184]: <seaborn.axisgrid.JointGrid at 0x21719350e50>



[185]:
```
g = sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪color = '#FF6600')
```



[186]:
```
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=db , height = 10 ,␣
↪color = '#FF6600')
```



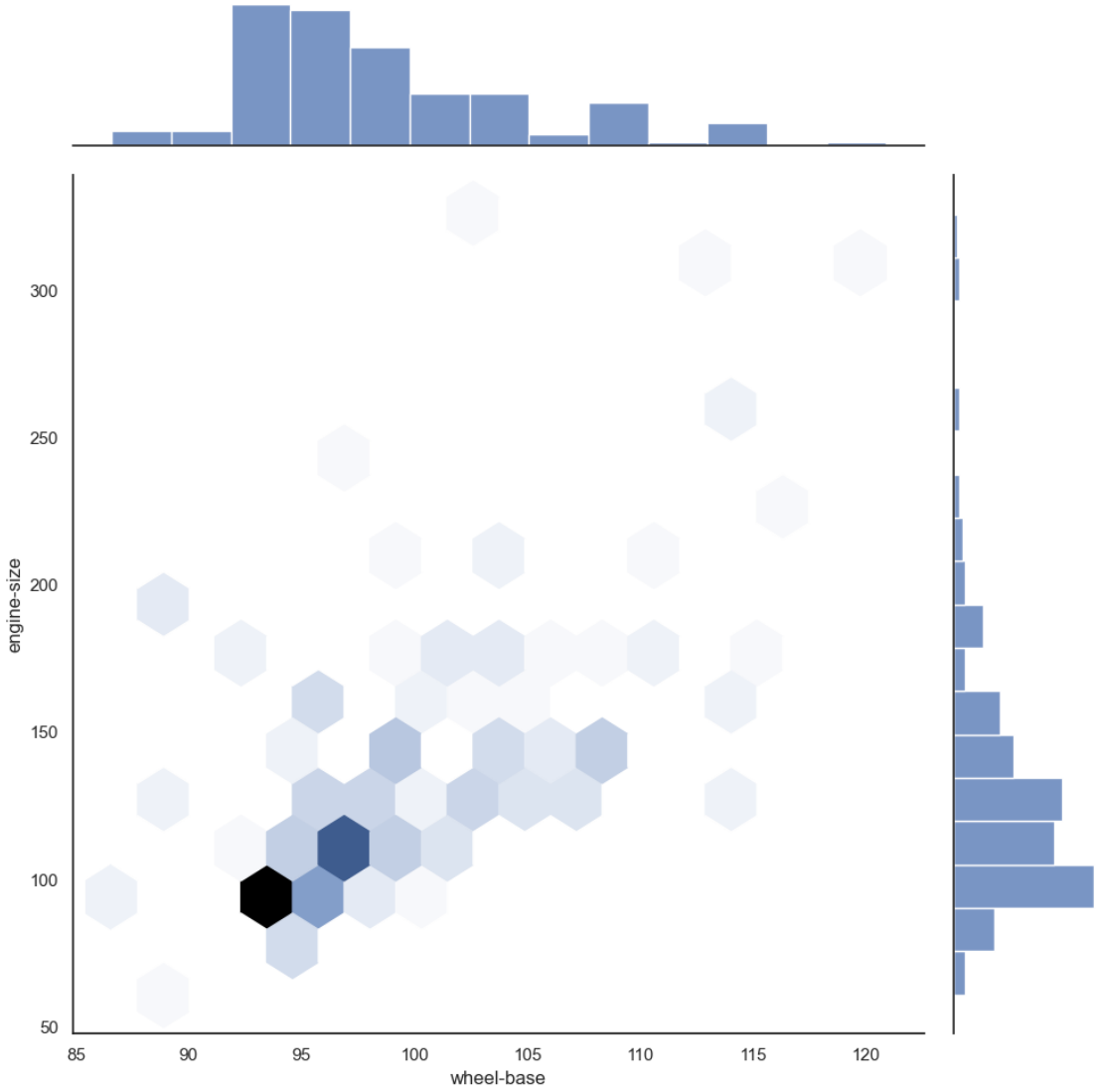
[187]:
```
mpl.rcParams.update(mpl.rcParamsDefault)
%matplotlib inline
sns.set(style="white", color_codes=True)
```

In Seaborn, the kind parameter in **jointplot()** function specifies the type of plot to be drawn. One of the options for kind is **"hex"**, which creates a **hexbin plot**.

A **hexbin plot** is a type of plot used to visualize the distribution of two variables in a bivariate dataset. It is similar to a scatter plot, but instead of showing individual data points, the plot aggregates the data into hexagonal bins and displays the count of data points in each bin with a color scale.

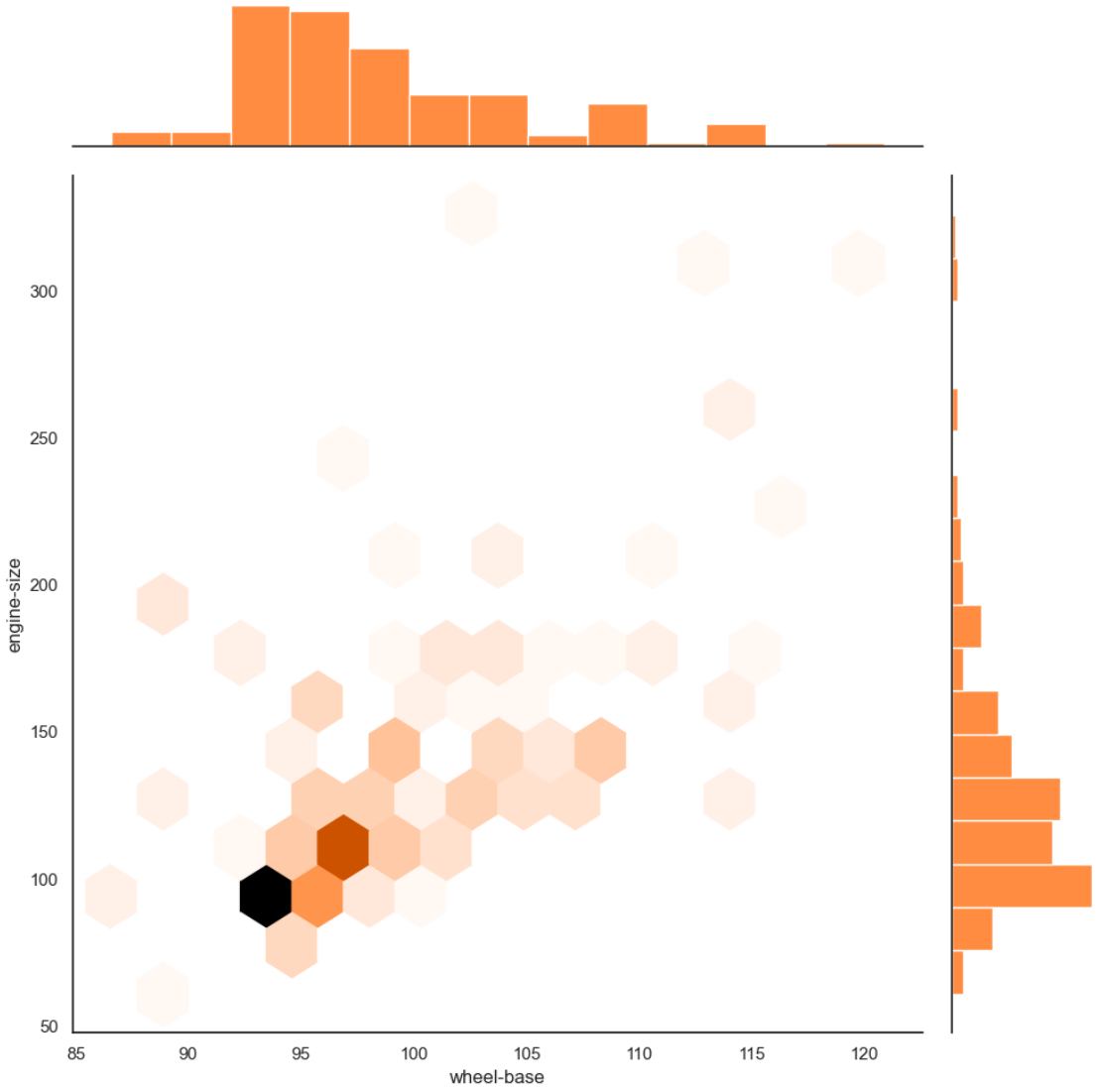
[188]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="hex")
```

[188]: <seaborn.axisgrid.JointGrid at 0x2171cb58c10>



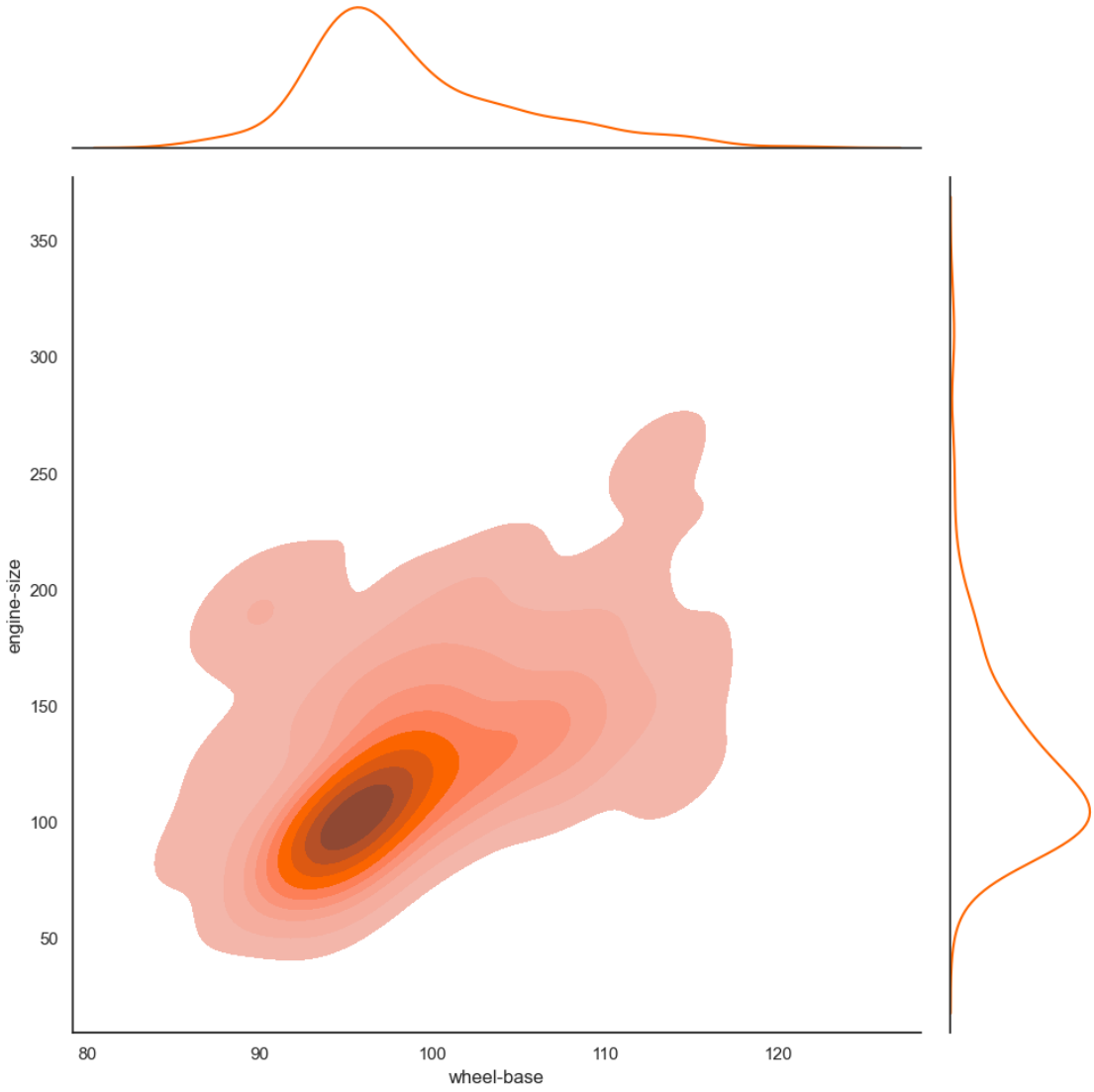
[189]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="hex" , color="#FF6600")
```

[189]: <seaborn.axisgrid.JointGrid at 0x2171d6624c0>



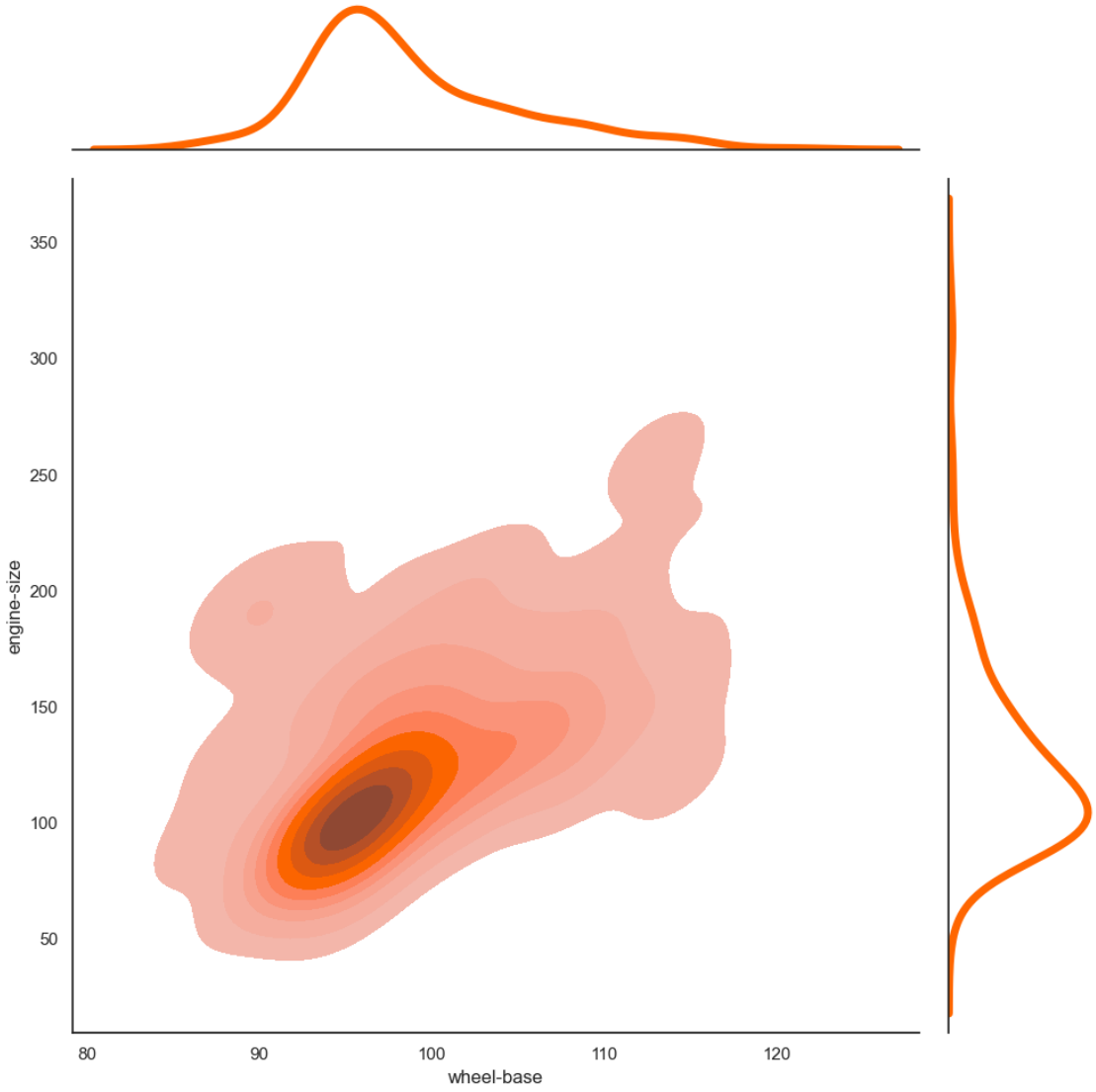
[190]:
```
sns.jointplot(x='wheel-base', y='engine-size', data=db, kind='kde', height=10,␣
↪color='#FF6600', joint_kws=dict(shade=True))
```

[190]: <seaborn.axisgrid.JointGrid at 0x217145c3c70>



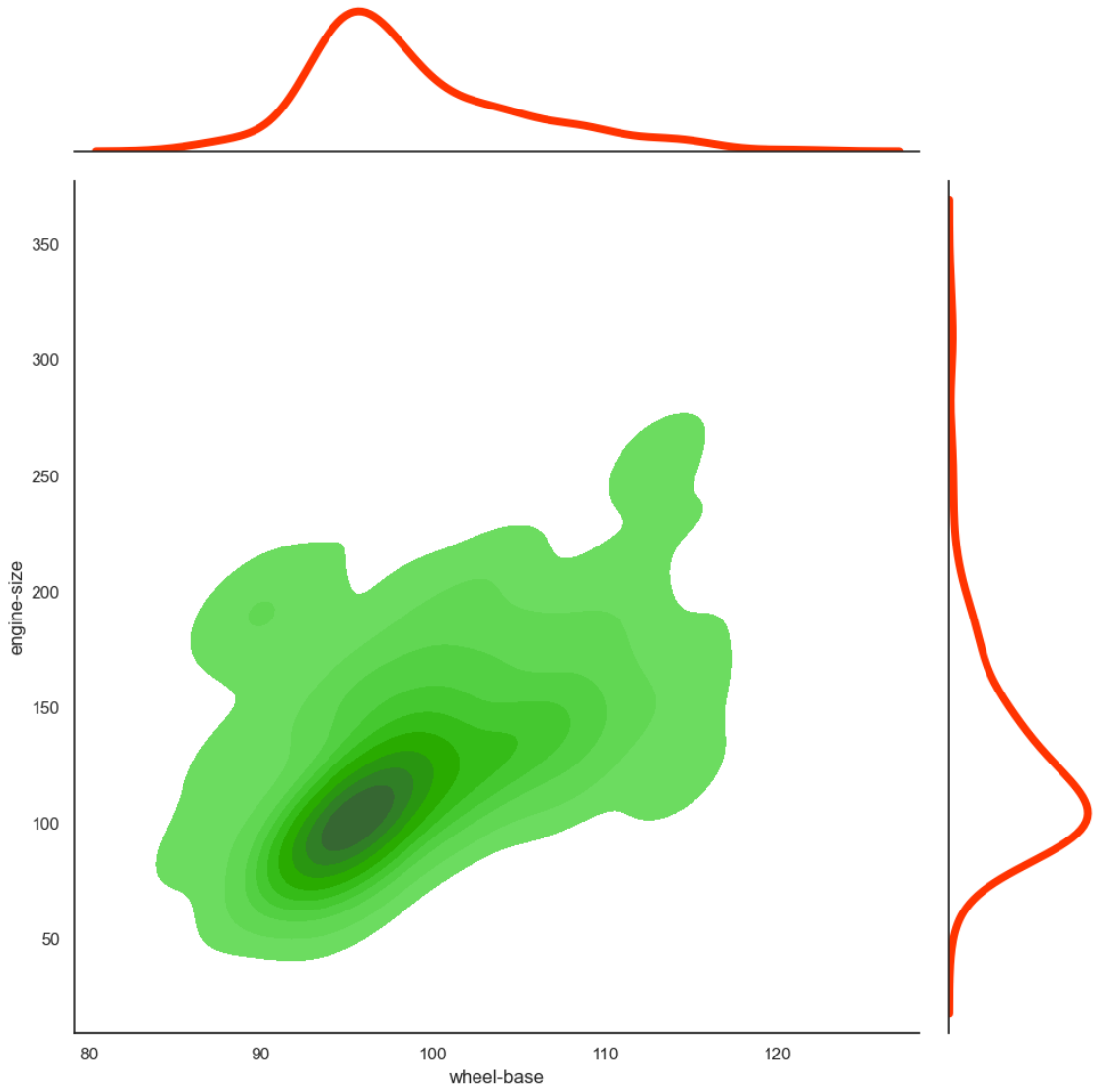
[191]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="kde" ,color="#FF6600" , joint_kws=dict(shade=True), marginal_kws={'lw':
↪5})
```

[191]: <seaborn.axisgrid.JointGrid at 0x21719a87730>



[192]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="kde" ,color="#33CC00" ,joint_kws=dict(shade=True), marginal_kws={'lw':
↪5 , 'color' : "#FF3300"})
```

[192]: <seaborn.axisgrid.JointGrid at 0x2171d60a550>

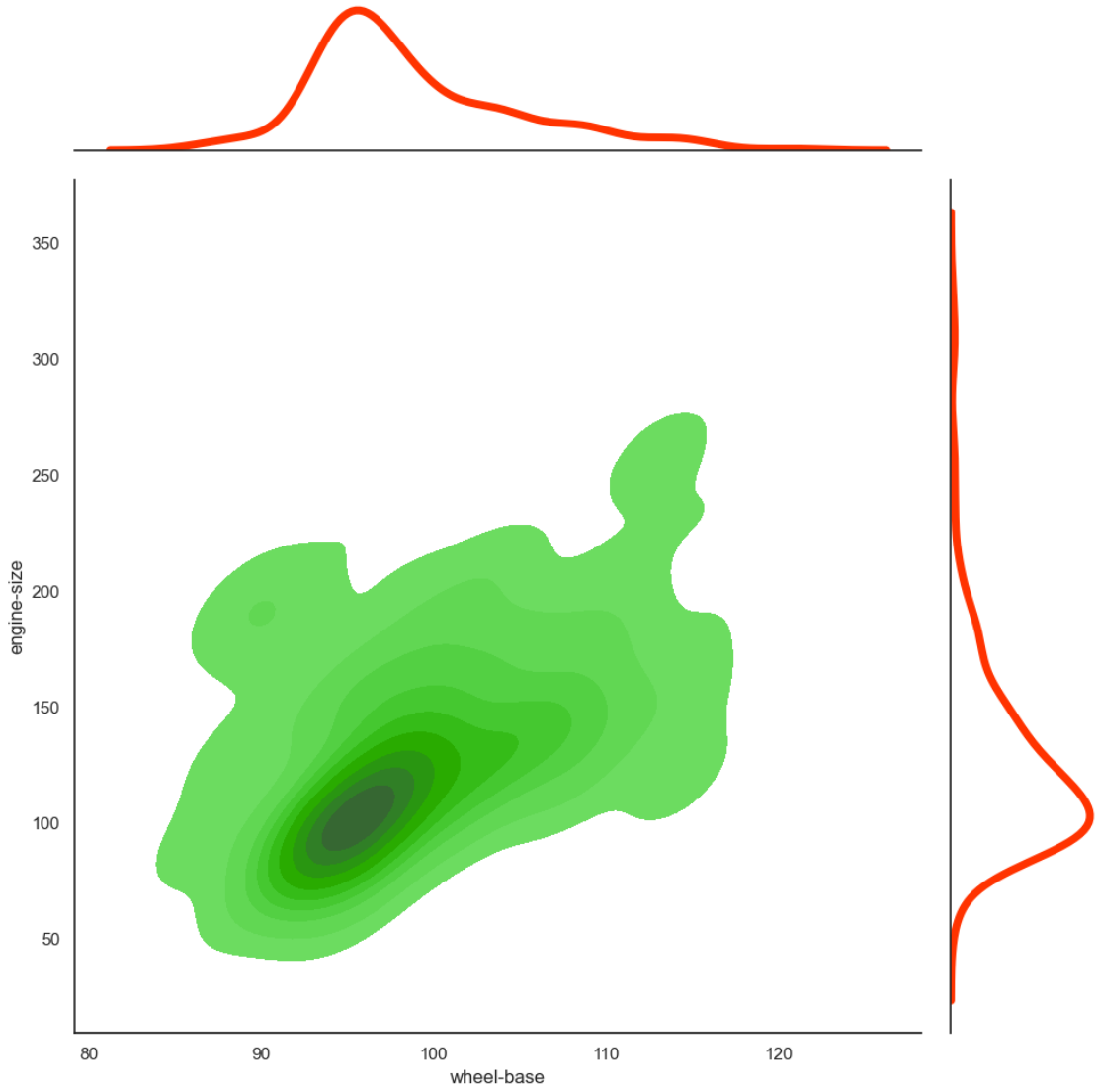


[193]:
```
sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
↪kind="kde" ,color="#33CC00" , marginal_kws={'lw':5 , 'color' : "#FF3300" ,␣
↪'bw' :.3},joint_kws=dict(shade=True))
```

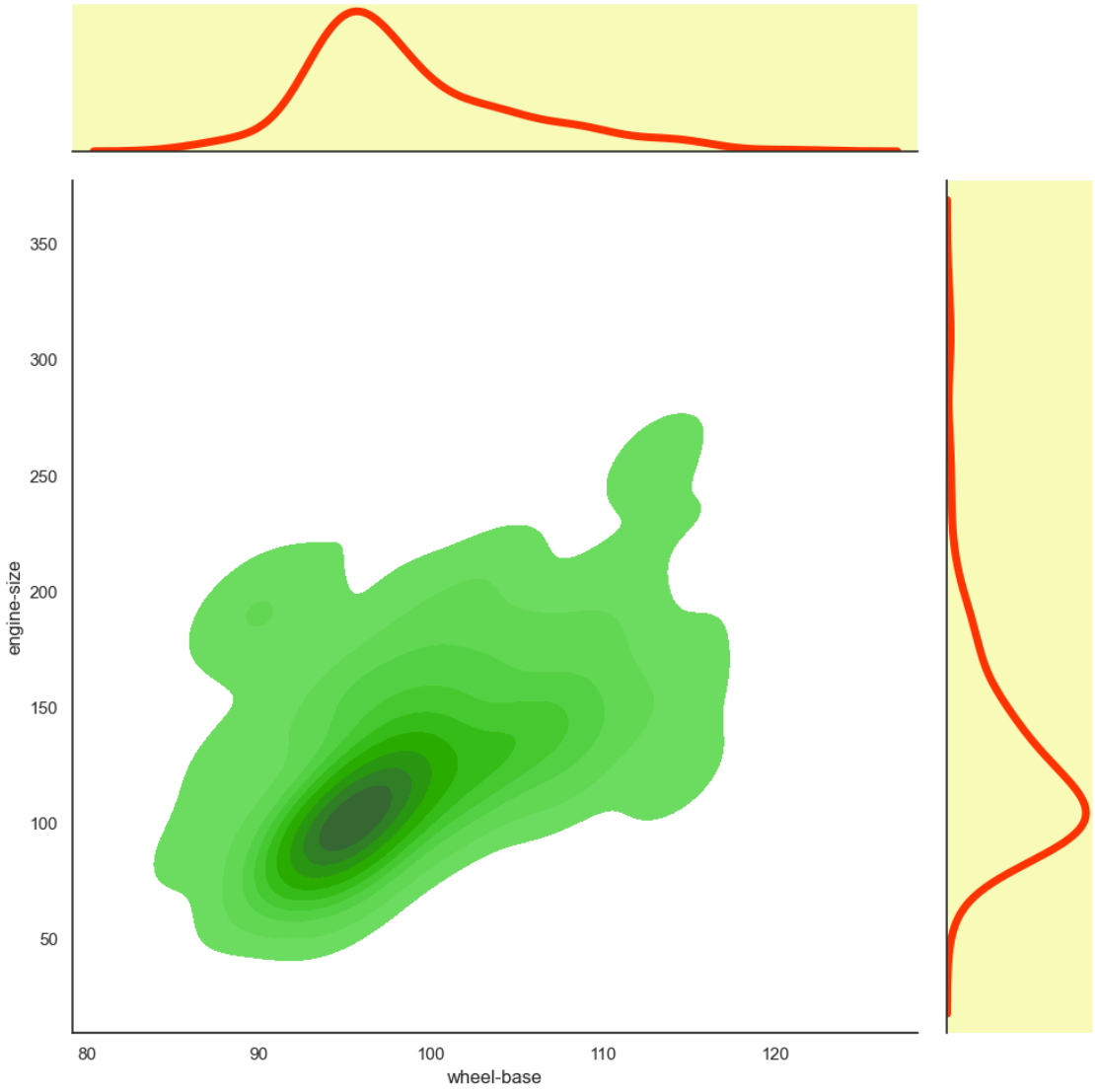
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1699:
FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and
`bw_adjust`. Using 0.3 for `bw_method`, but please see the docs for the new
parameters and update your code.
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1699:
FutureWarning: The `bw` parameter is deprecated in favor of `bw_method` and
`bw_adjust`. Using 0.3 for `bw_method`, but please see the docs for the new
parameters and update your code.
```

```
  warnings.warn(msg, FutureWarning)
```

[193]:
```
<seaborn.axisgrid.JointGrid at 0x217226a8e80>
```



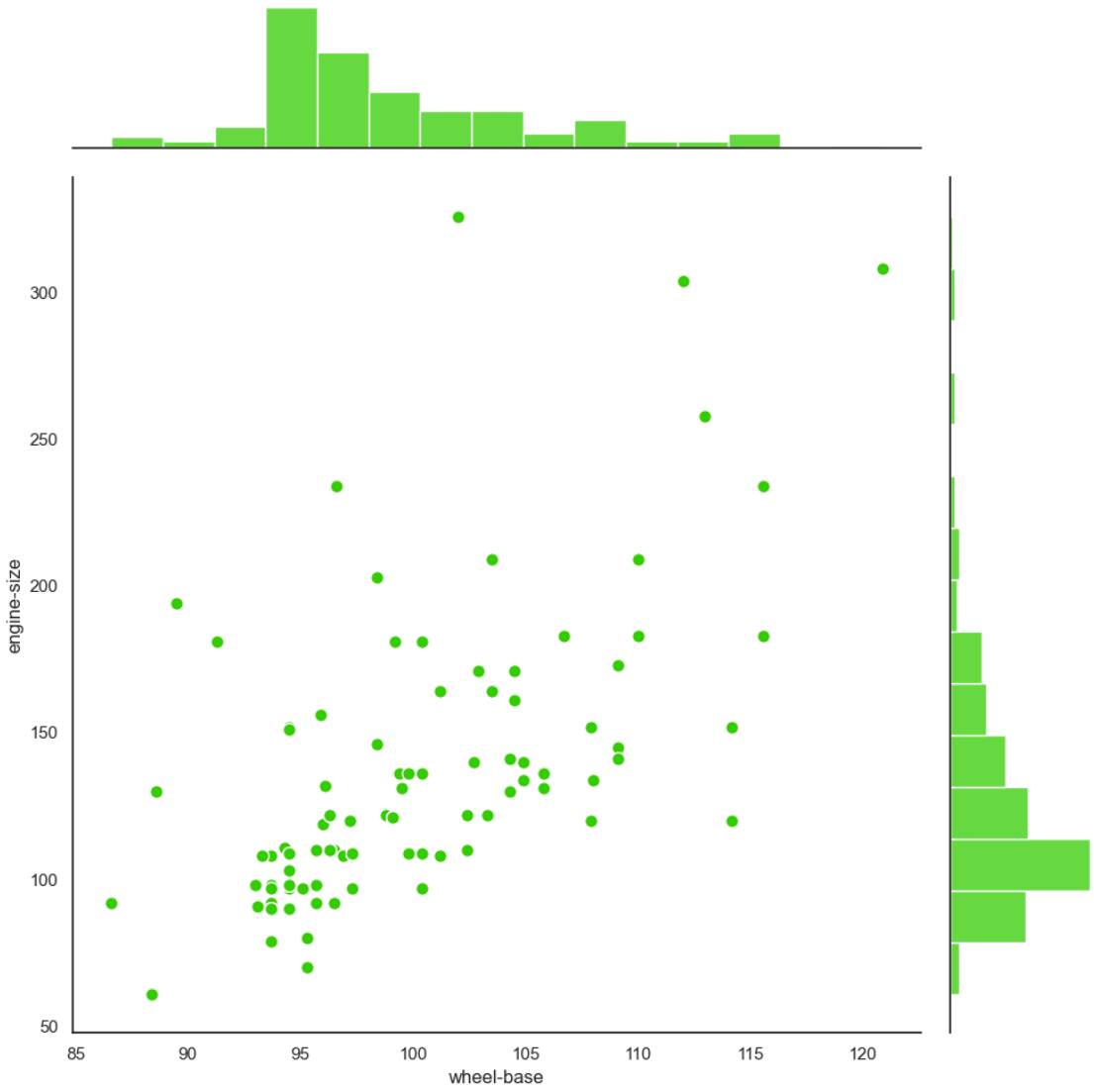
[194]:
```
g = sns.jointplot(x="wheel-base", y="engine-size", data=db , height = 10 ,␣
 ↪kind="kde" ,color="#33CC00" , marginal_kws={'lw':5 , 'color' :␣
 ↪"#FF3300"},joint_kws=dict(shade=True))
g.ax_marg_x.set_facecolor('#f8fab8')
g.ax_marg_y.set_facecolor('#f8fab8')
```



[195]:
```
sns.jointplot("wheel-base", "engine-size", data=db, s=70, edgecolor="w",␣
 ↪linewidth=1, height =10,color ="#33CC00",marginal_kws=dict(bins=15,␣
 ↪rug=True))
plt.show()
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variables as keyword args: x, y. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
C:\Users\suman\anaconda\lib\site-packages\seaborn\axisgrid.py:2203: UserWarning:
The marginal plotting function has changed to `histplot`, which does not accept
```

```
the following argument(s): rug.
  warnings.warn(msg, UserWarning)
```



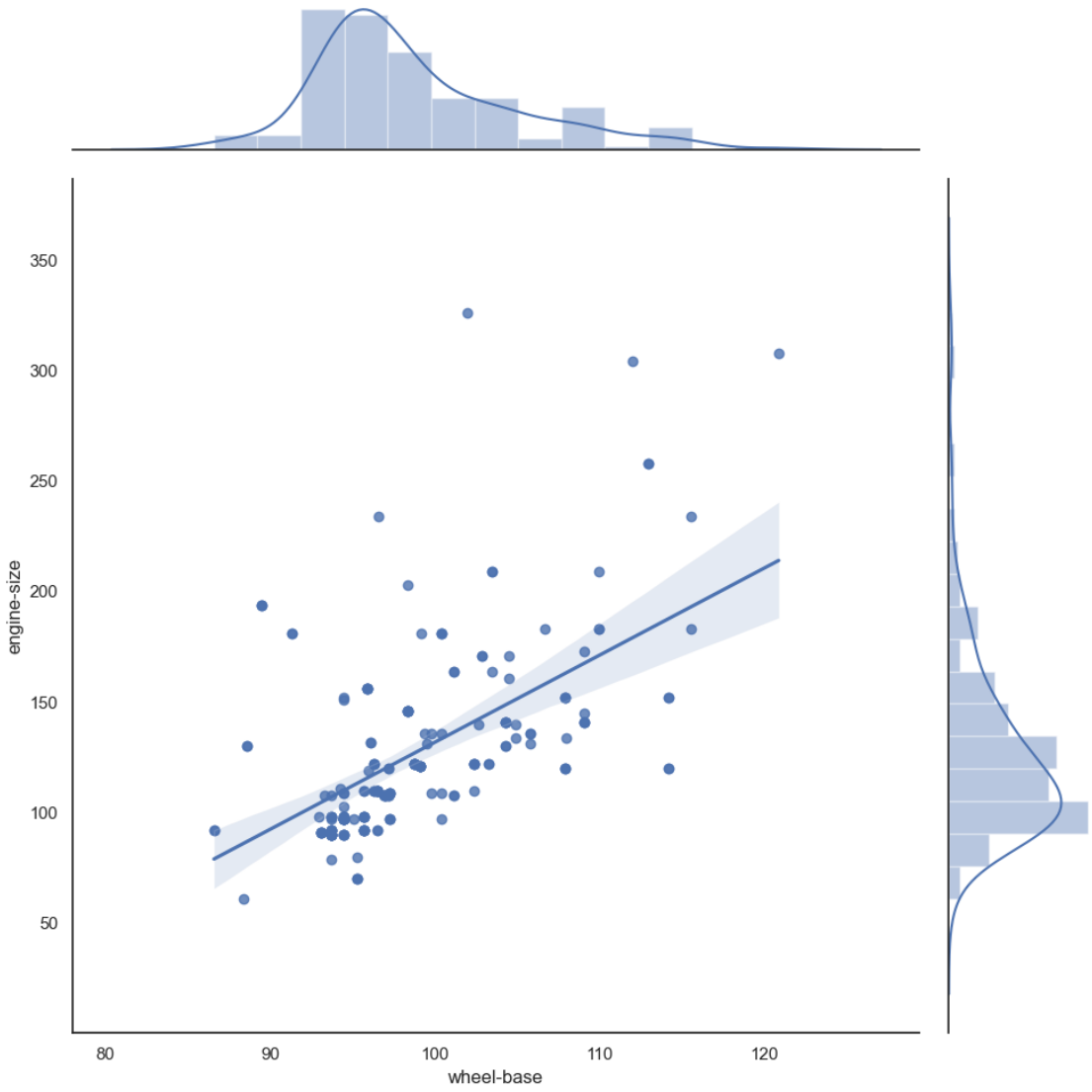
1.20 Joint grid

JointGrid is a class in Seaborn that allows you to create custom joint plots with more flexibility than the **jointplot()** function. It is essentially a container for a Figure object and a set of Axes objects, which you can use to create your own joint plots.

With JointGrid, you can customize the individual plots in a joint plot and add additional plots to the margins. For example, you can create a scatter plot in the center, a histogram of the x variable on top, and a histogram of the y variable on the right.

[196]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 10)
g = g.plot(sns.regplot, sns.distplot)
```

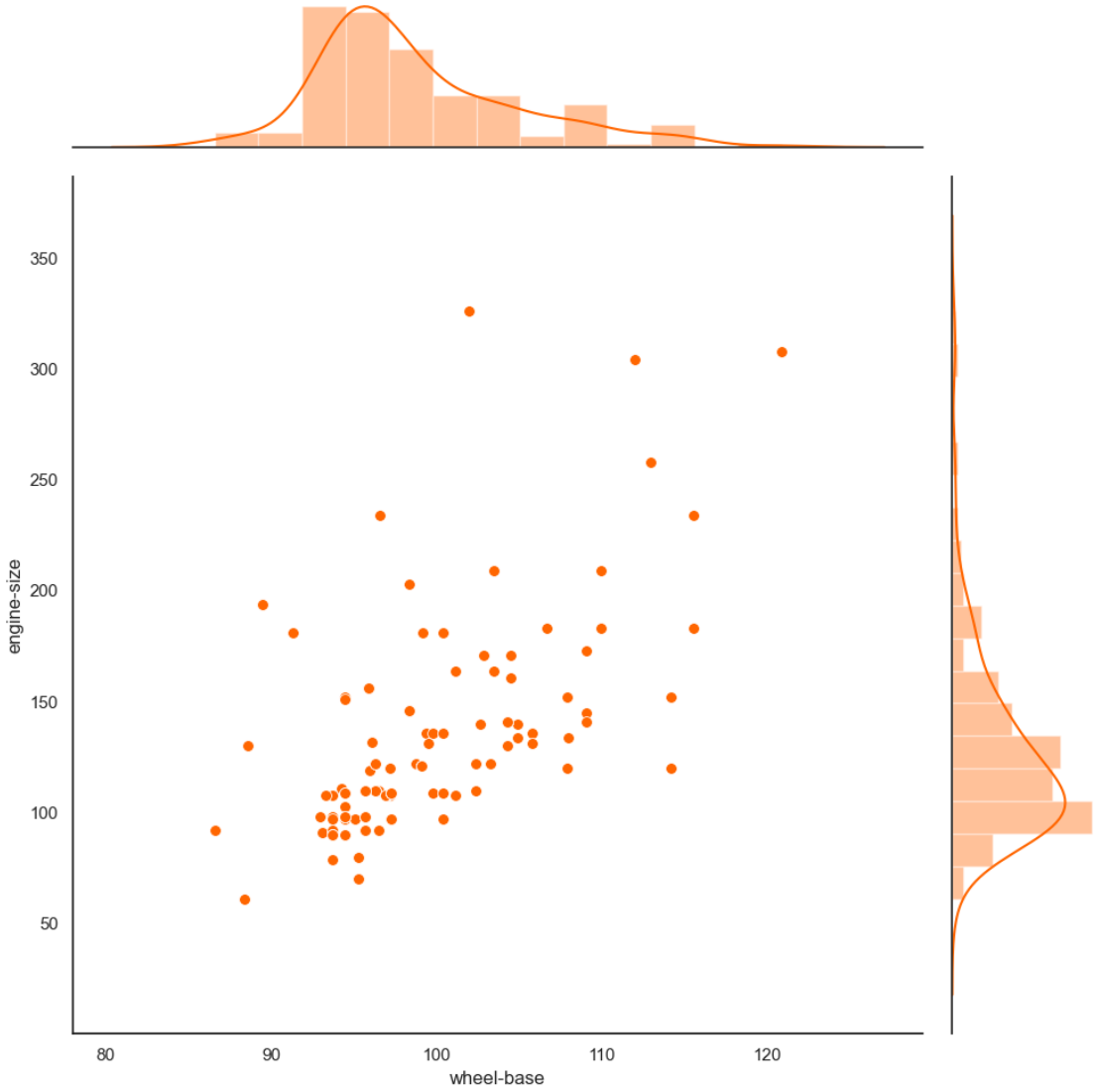
```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1689:
FutureWarning: The `vertical` parameter is deprecated and will be removed in a
future version. Assign the data to the `y` variable instead.
  warnings.warn(msg, FutureWarning)
```



[197]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 10)
g = g.plot_joint(sns.scatterplot, color="#FF6600",s=50)
g = g.plot_marginals(sns.distplot, color="#FF6600")
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
```

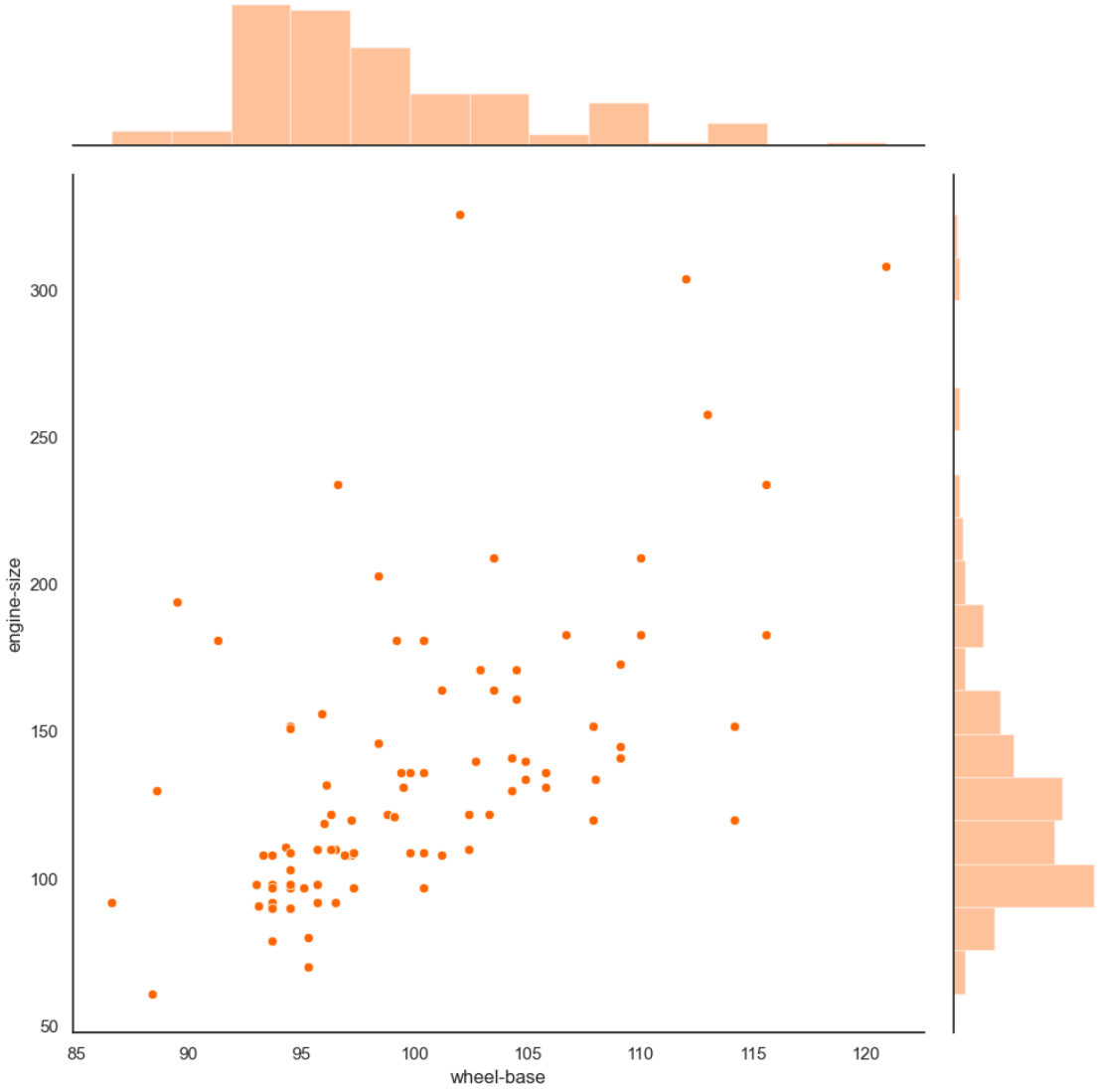
```
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:1689:
FutureWarning: The `vertical` parameter is deprecated and will be removed in a
future version. Assign the data to the `y` variable instead.
  warnings.warn(msg, FutureWarning)
```



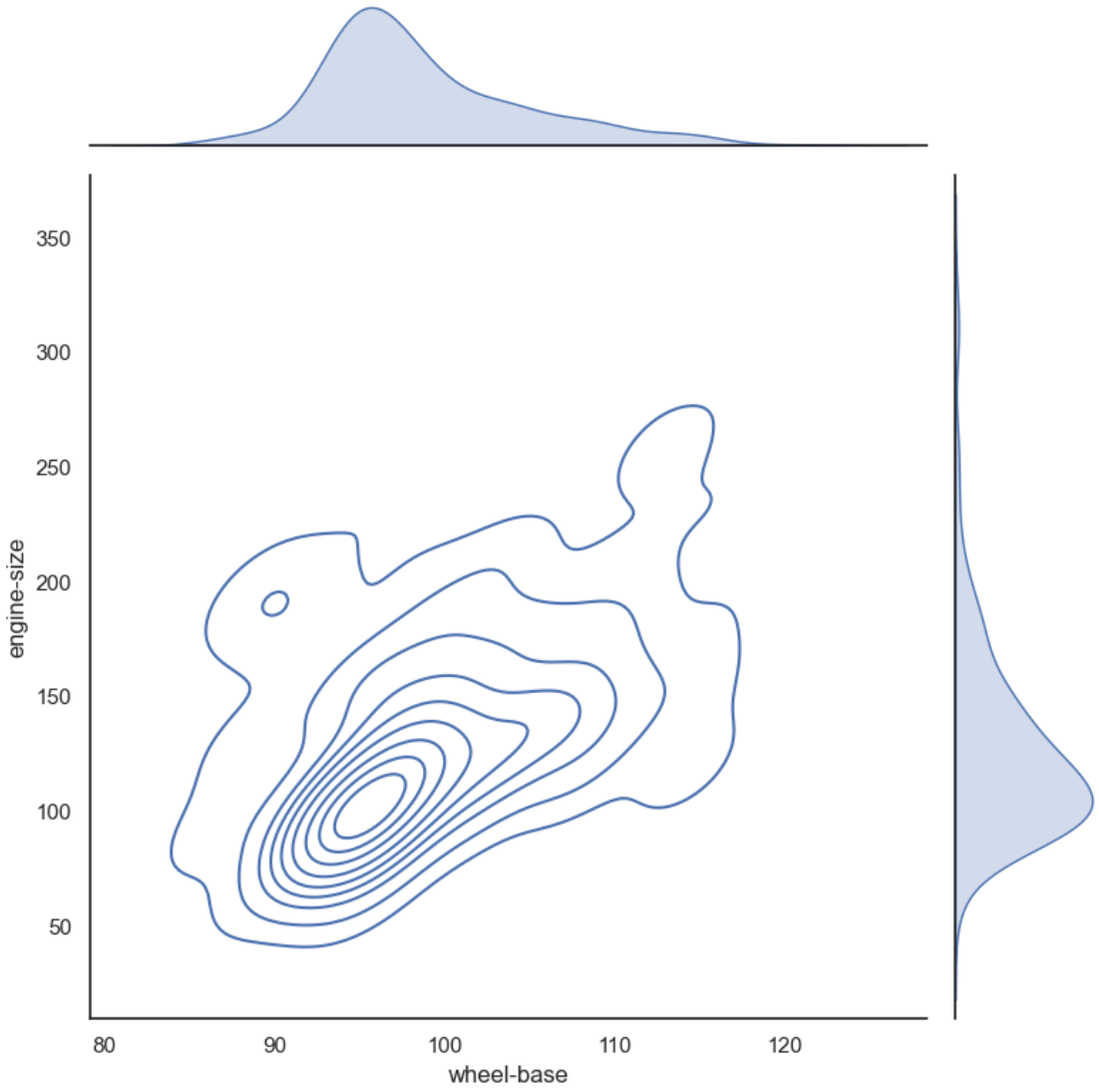
[198]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 10)
g = g.plot_joint(sns.scatterplot, color="#FF6600")
g = g.plot_marginals(sns.distplot,kde= False, color="#FF6600")
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
```

```
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



[199]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 8)
g = g.plot_joint(sns.kdeplot)
g = g.plot_marginals(sns.kdeplot, shade=True)
```



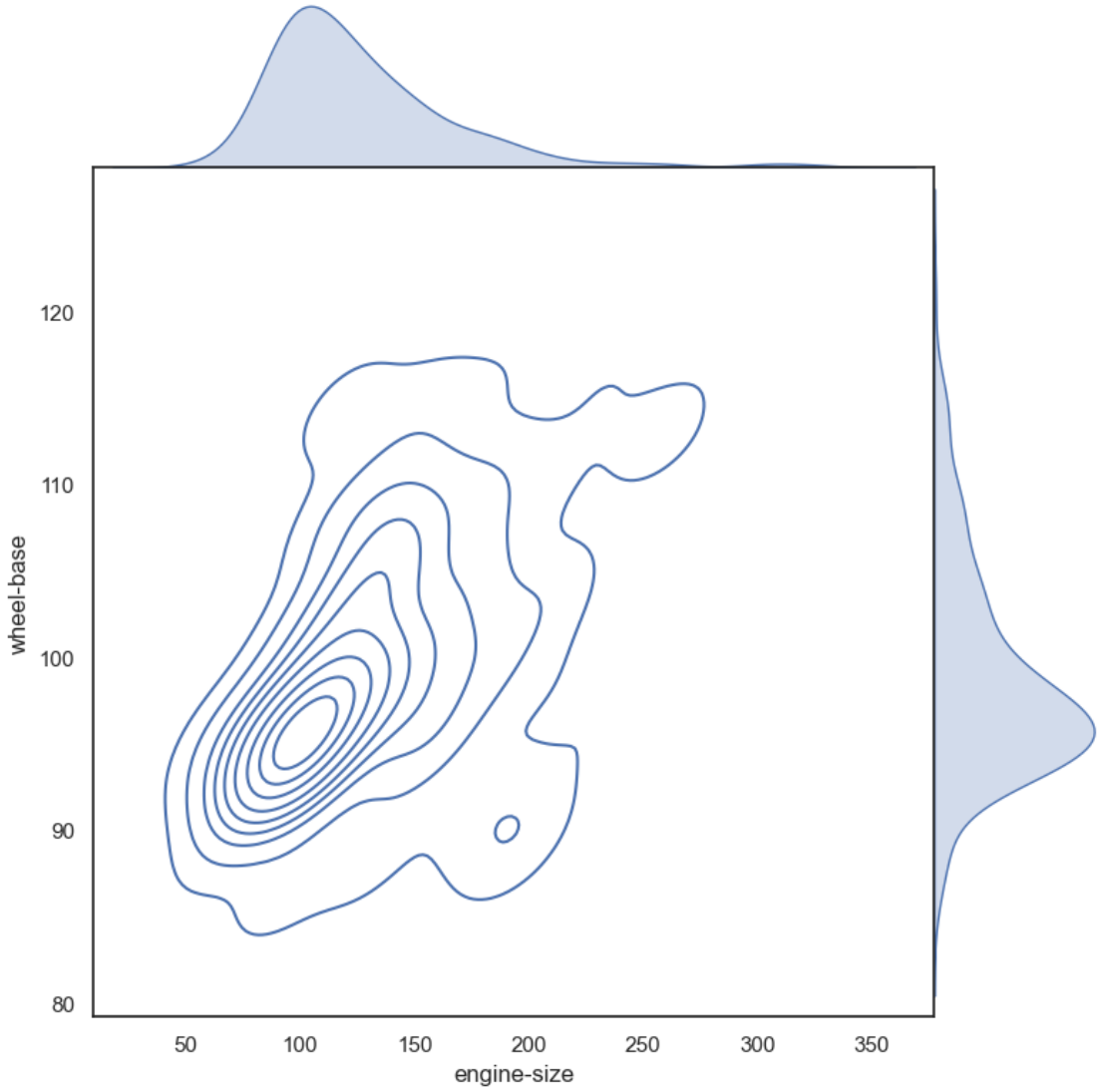
In Seaborn's JointGrid object, the **space** parameter determines the amount of space left between the joint plot and the marginal plots. Specifically, space controls the **ratio** of the size of the marginal plots to the size of the joint plot.

A value of **0** for the space parameter means that the marginal plots will share the same space as the joint plot. Increasing the value of space will increase the size of the marginal plots relative to the joint plot.

In the code you provided, space=0 means that there is no space left between the joint plot and the marginal plots. This can be useful when you want to maximize the use of space on the figure and have the marginal plots share the same axis with the joint plot.

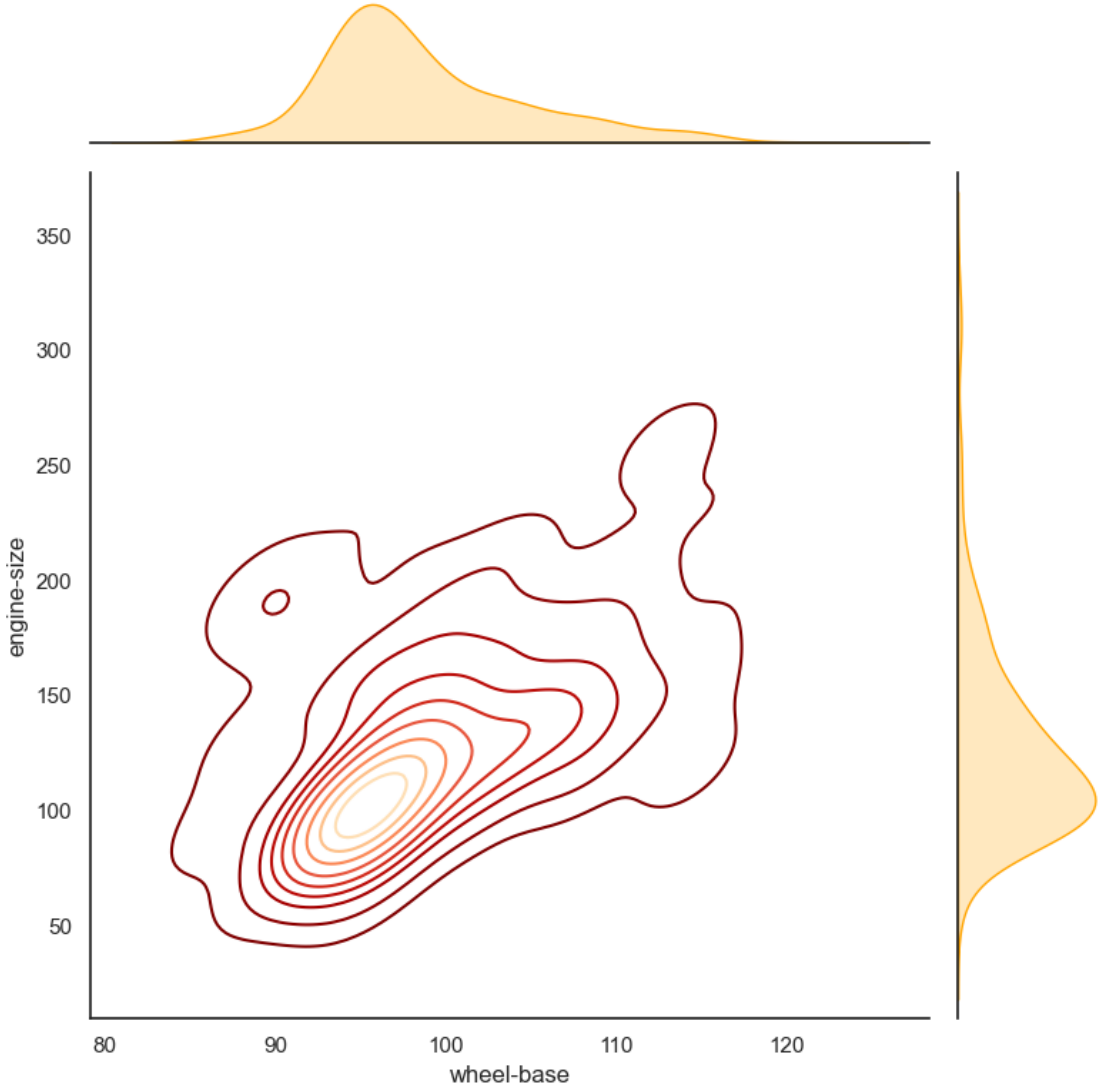
[200]:
```
g = sns.JointGrid(x="engine-size", y="wheel-base", data=db, height = 8 , space␣
↪=0)
```

```
g = g.plot_joint(sns.kdeplot)
g = g.plot_marginals(sns.kdeplot, shade=True)
```



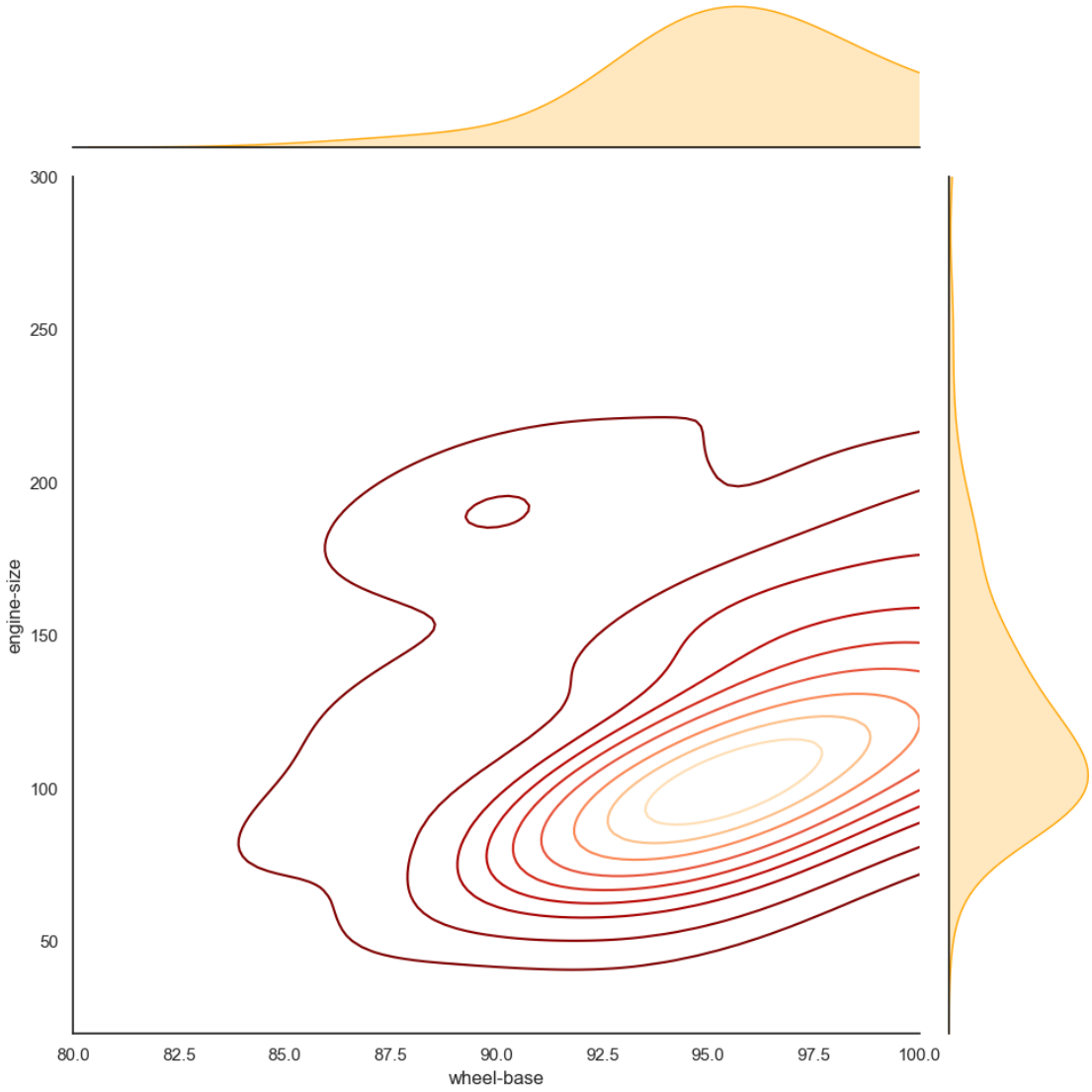
The **cmap="OrRd_r"** color map is a reversed version of the **"OrRd"** color map, which stands for **"Orange-Red"**. This color map is a sequential color map that goes from light yellow to dark red, with the lighter colors representing lower values and the darker colors representing higher values.

[201]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 8)
g = g.plot_joint(sns.kdeplot , cmap="OrRd_r")
g = g.plot_marginals(sns.kdeplot, shade=True , color = 'orange')
```



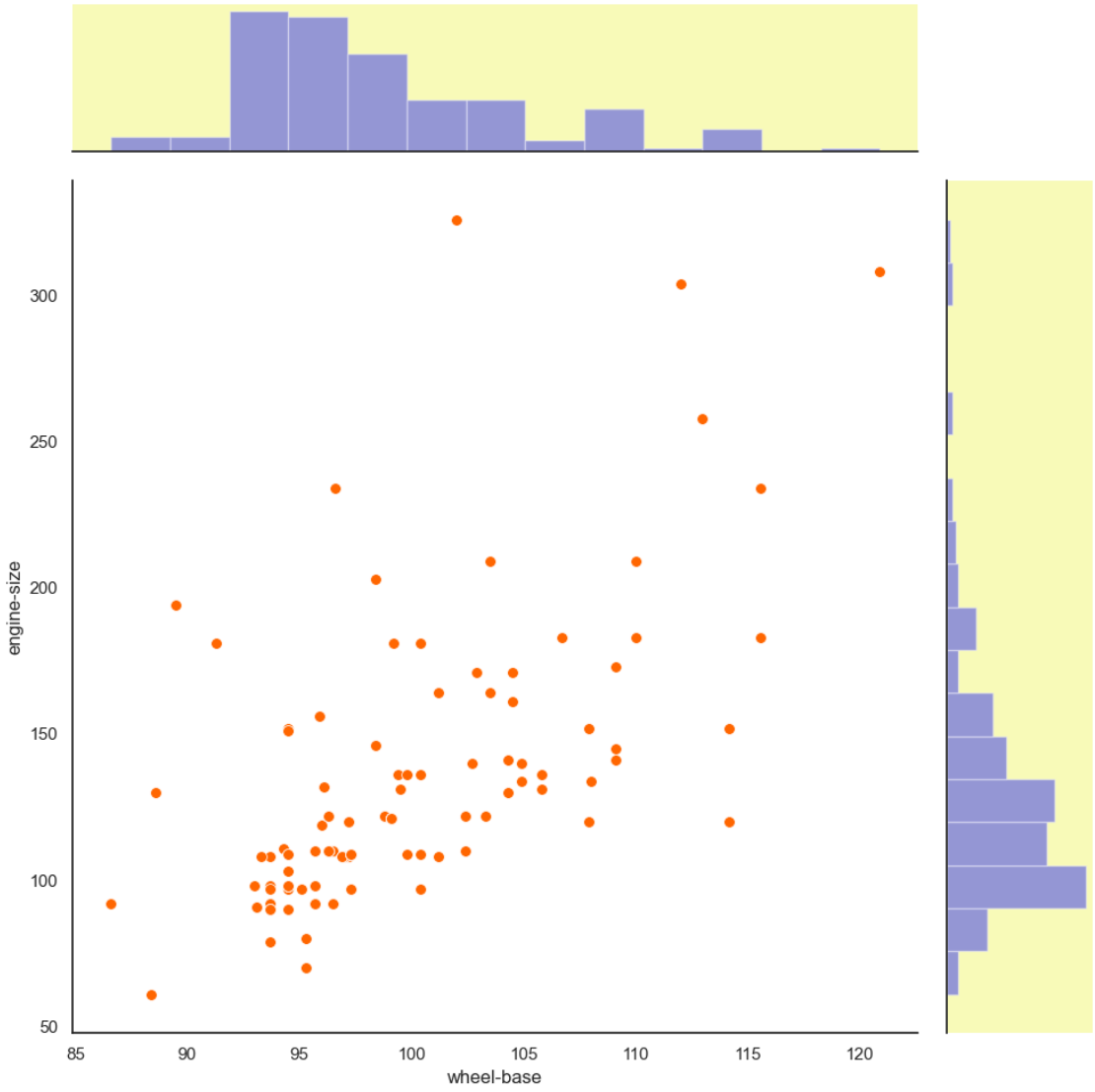
Set Axis limits.

[202]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, height = 10 ,␣
↪xlim=(80,100) , ylim=(20,300))
g = g.plot_joint(sns.kdeplot , cmap="OrRd_r")
g = g.plot_marginals(sns.kdeplot, shade=True , color = 'orange')
```



[203]:
```
g = sns.JointGrid(x="wheel-base", y="engine-size", data=db, 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')
```

```
C:\Users\suman\anaconda\lib\site-packages\seaborn\distributions.py:2619:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```



1.21 Heat map

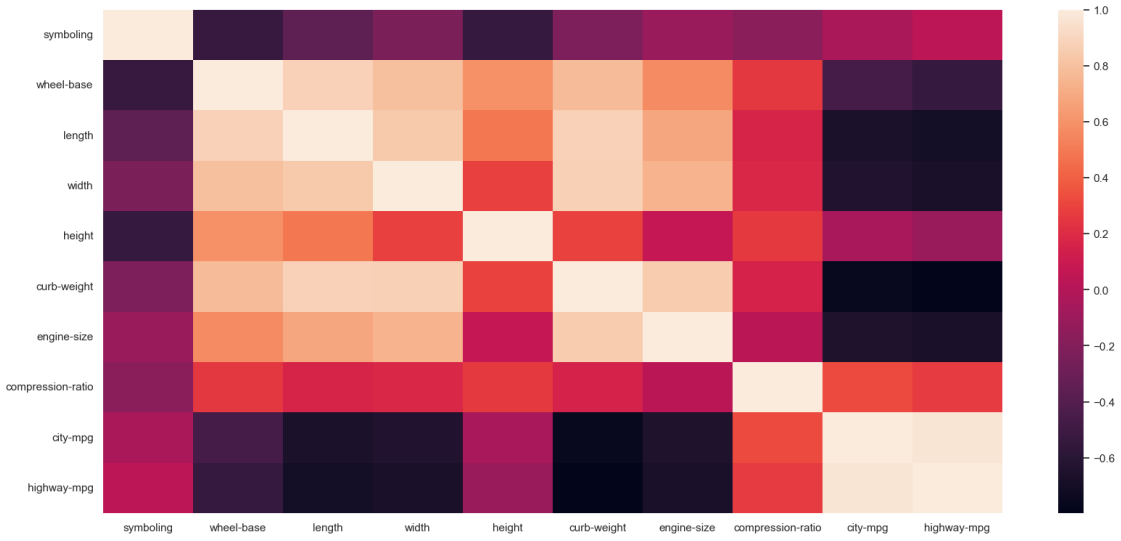
A **heat map** is a graphical representation of data that uses colors to represent different values. Heat maps are commonly used to visualize large amounts of data and to identify patterns and trends.

In a heat map, each data point is represented as a colored square or rectangle, with the color corresponding to the value of the data point. The color scale used in the heat map can be a gradient of one or multiple colors, with lighter colors representing lower values and darker colors representing higher values.

Heat maps are often used to visualize data in two dimensions, such as a table of numbers, where each row represents an observation and each column represents a variable. Heat maps can also be used to visualize data in three dimensions, by using color to represent a third variable.

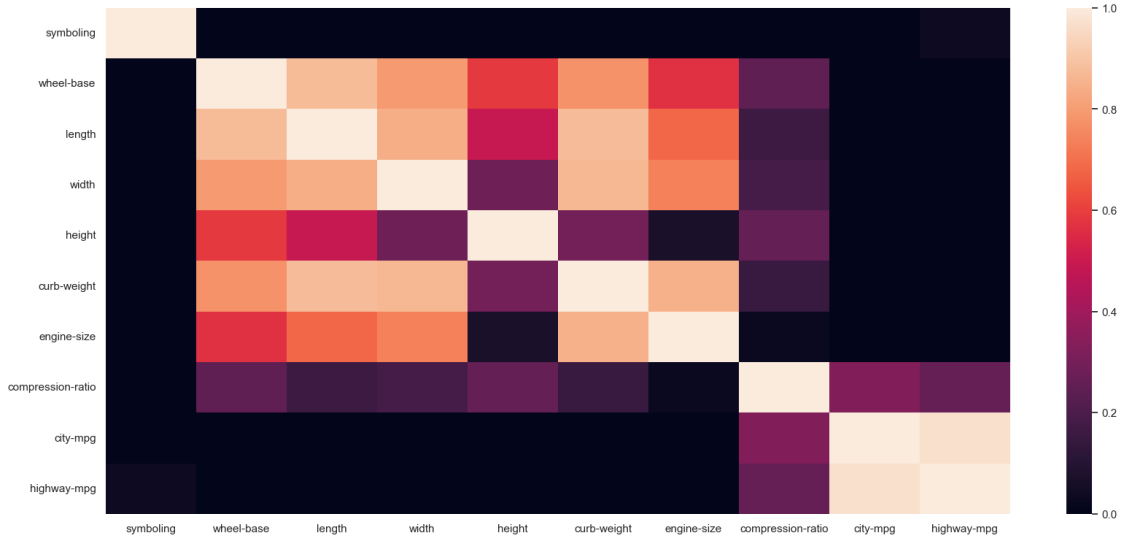
Heat maps are particularly useful for identifying patterns and trends in large datasets, especially when there are many variables involved. They can also be used to identify outliers, to explore relationships between variables, and to communicate complex data in a simple and intuitive way.

[204]:
```
corr = db.corr()
plt.figure(figsize=(20,9))
ax = sns.heatmap(corr)
plt.yticks(rotation=0)
plt.show()
```

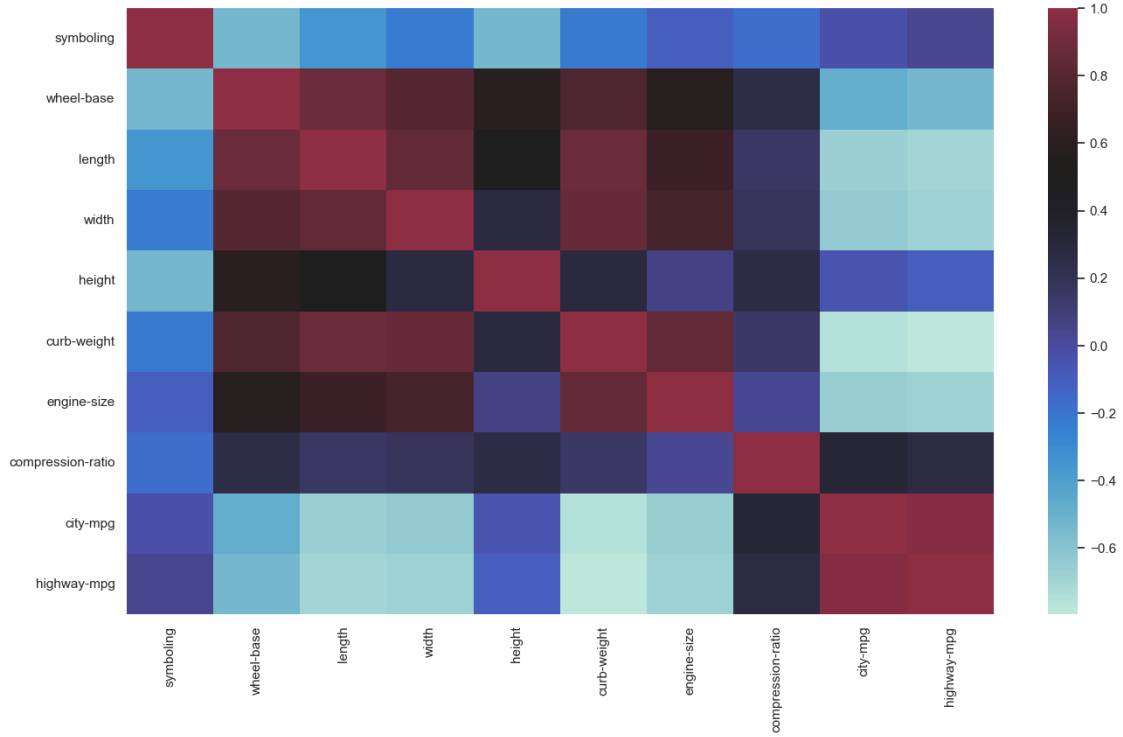


Removes negetive values.

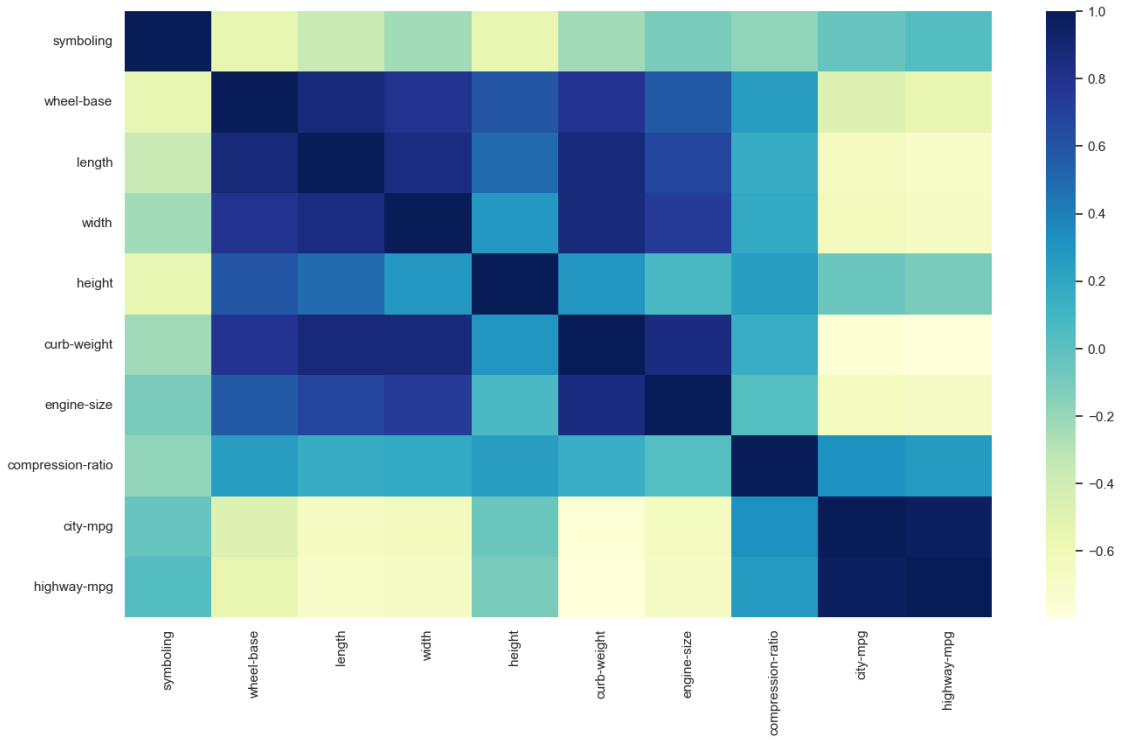
[205]:
```
plt.figure(figsize=(20,9))
ax = sns.heatmap(corr,vmin=0, vmax=1)
plt.yticks(rotation=0)
plt.show()
```



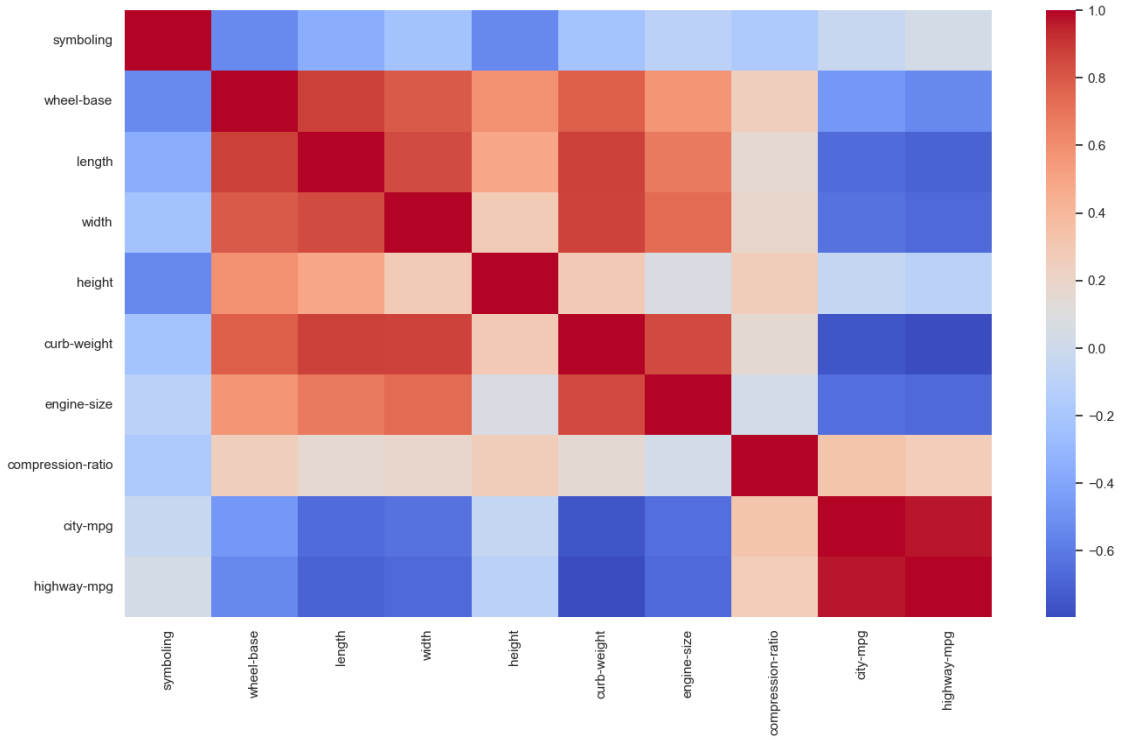
[206]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,center=0.5)
plt.yticks(rotation=0)
plt.show()
```



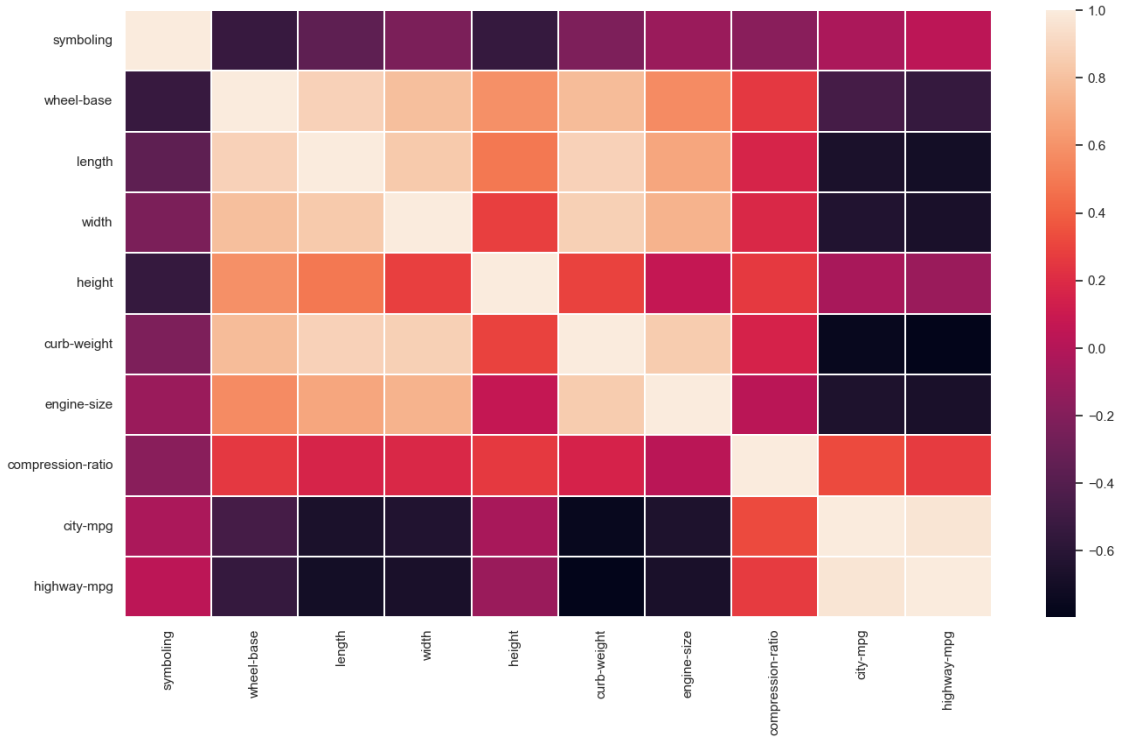
[207]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="YlGnBu")
plt.yticks(rotation=0)
plt.show()
```



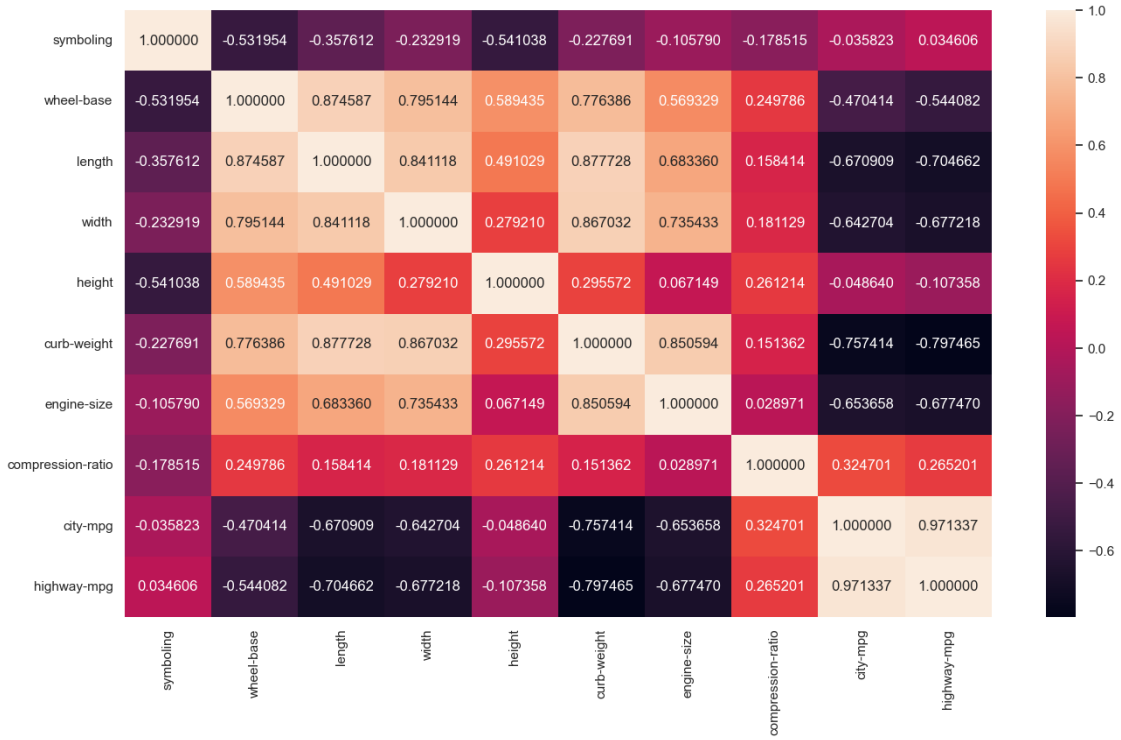
[208]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="coolwarm")
plt.yticks(rotation=0)
plt.show()
```



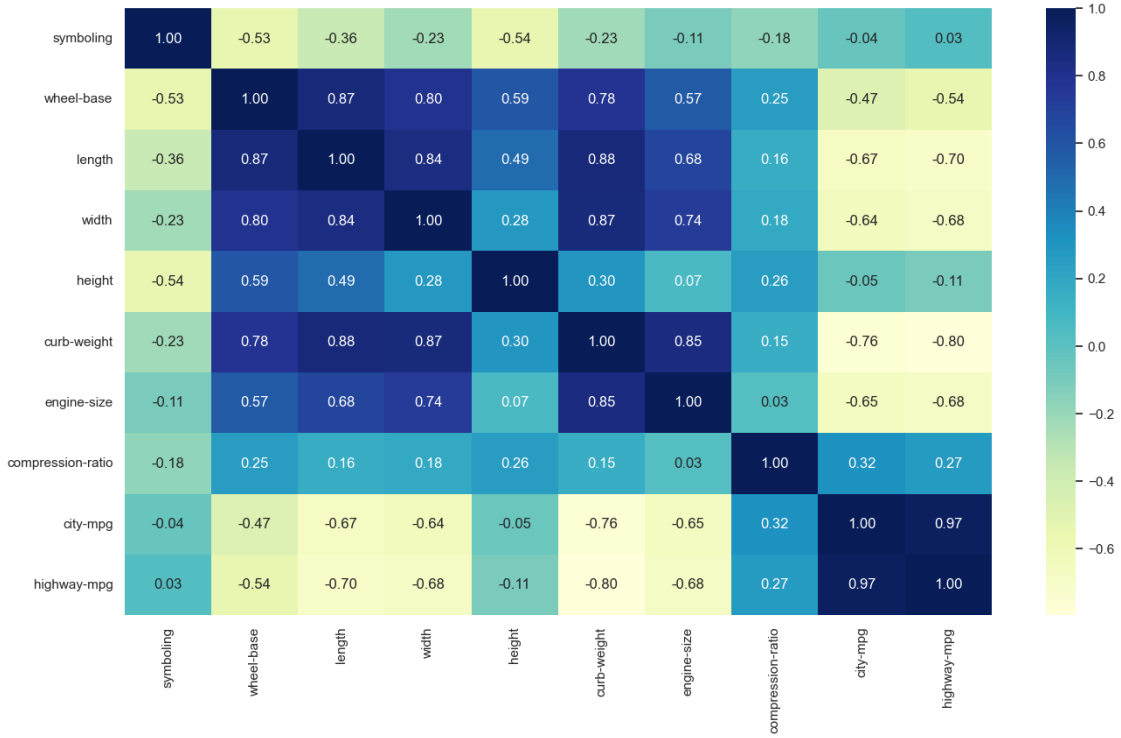
[209]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,linewidths=.1)
plt.yticks(rotation=0)
plt.show()
```



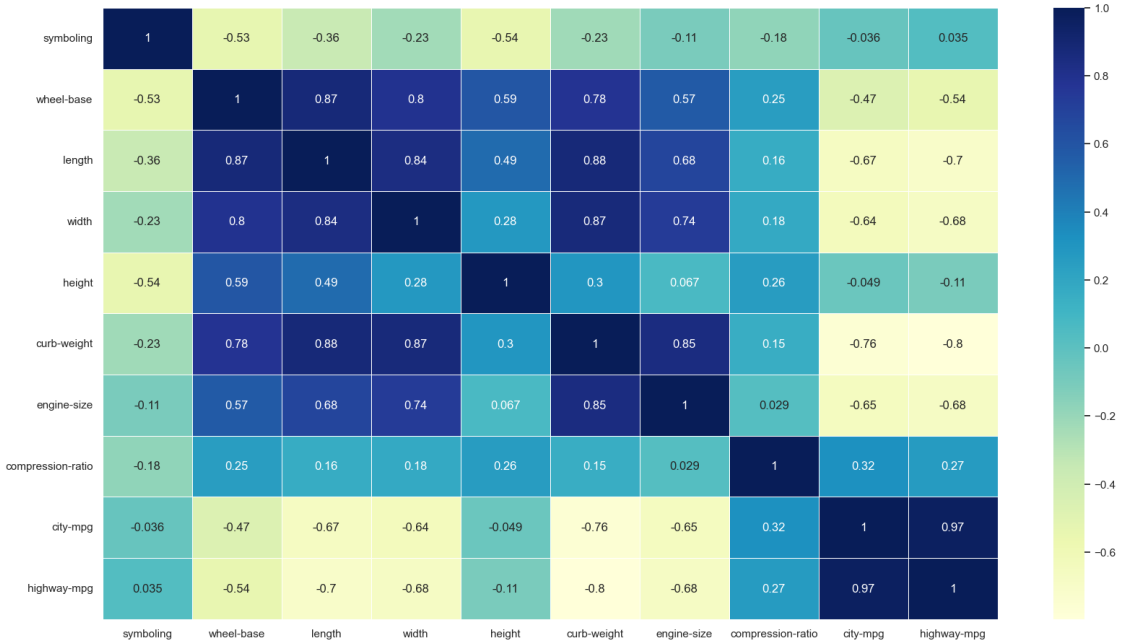
[210]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,annot=True, fmt="f")
plt.yticks(rotation=0)
plt.show()
```



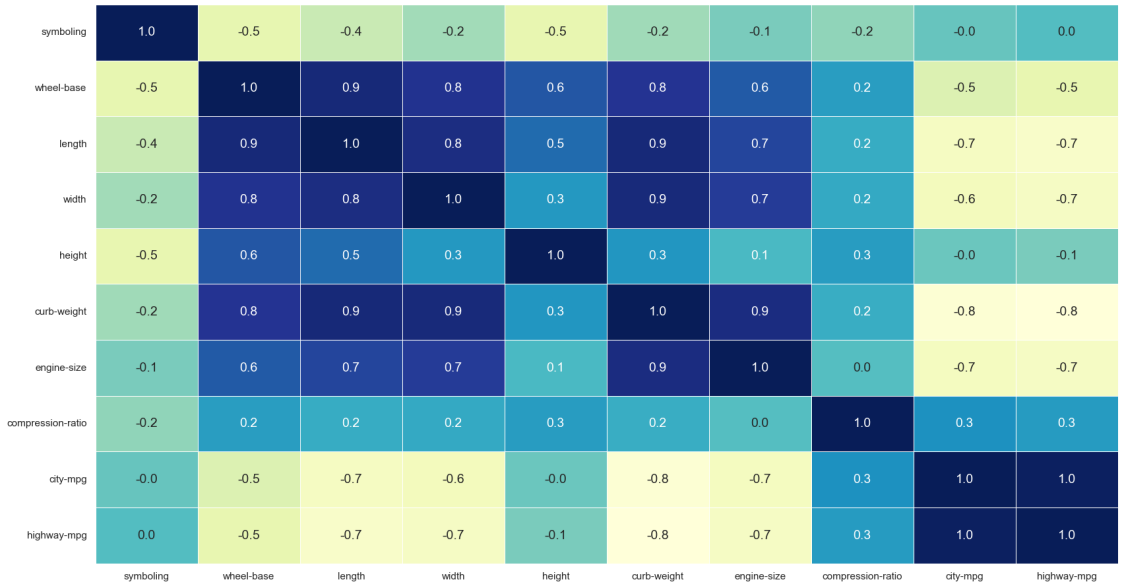
[211]:
```
plt.figure(figsize=(16,9))
ax = sns.heatmap(corr,cmap="YlGnBu", annot=True ,fmt=".2f")
plt.yticks(rotation=0)
plt.show()
```



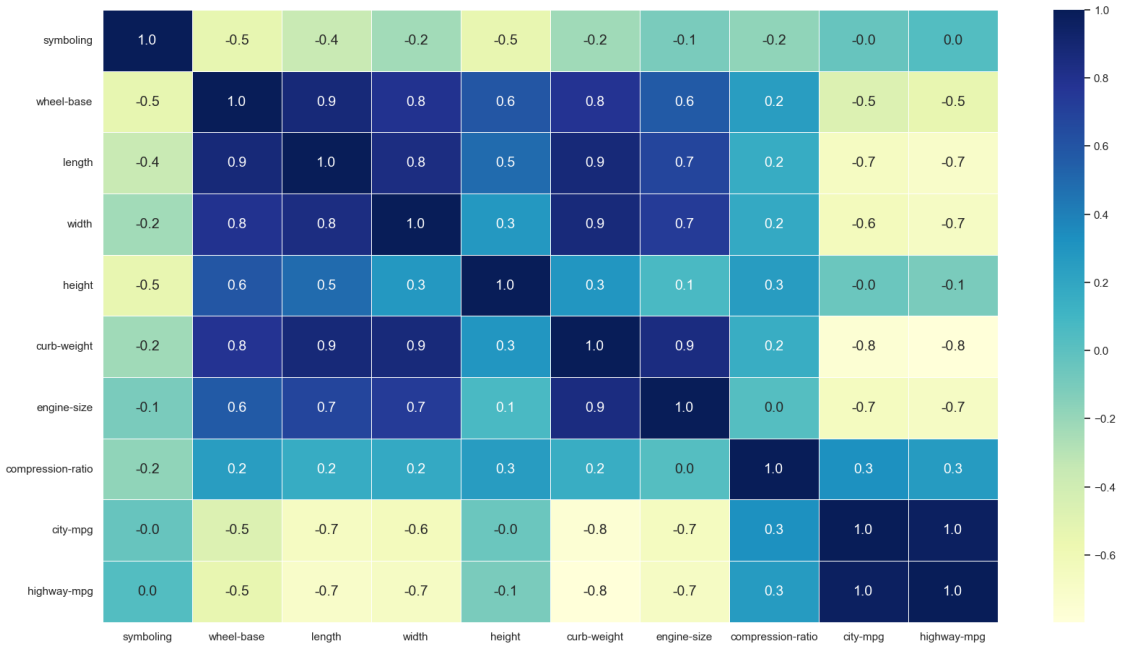
[212]:
```
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,cmap="YlGnBu", linewidths=.5, annot=True )
plt.yticks(rotation=0)
plt.show()
```



[213]:
```
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,cmap="YlGnBu", linewidths=.5,annot=True␣
 ↪,annot_kws={'size':14} ,fmt=".1f" , cbar=False)
plt.yticks(rotation=0)
plt.show()
```



[214]:
```
plt.figure(figsize=(20,11))
ax = sns.heatmap(corr,cmap="YlGnBu", linewidths=.5,annot=True␣
 ↪,annot_kws={'size':14} ,fmt=".1f" , cbar=True)
plt.yticks(rotation=0)
plt.show()
```



[215]:
```
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()
```

2 The End

2.1 Thank you