Performing EDA on Car information dataset

Importing Required Libraries

In [1]:

```
# Importing libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

from mpl_toolkits.mplot3d import Axes3D

# Import warnings to ignore any warnings
import warnings
warnings.filterwarnings('ignore')
```

Insert dataset

In [2]:
```
# Load dataset
data = pd.read_csv('Vehicle Manufacturing.csv')
```

read_csv() function : Read a comma-separated values (csv) file into DataFrame.

Displaying data

In [3]:
```
# Display data
data
```

Out[3]:

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	
...	...	...	...	...	...	...	...	...	
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	eur
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	

398 rows × 9 columns

Displaying first 5 data

In [4]:
```
# Display the first 5 rows
data.head()
```

Out[4]:

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin
0	chevrolet chevelle malibu	18.0	8	307.0	130.0	3504	12.0	70	usa
1	buick skylark 320	15.0	8	350.0	165.0	3693	11.5	70	usa
2	plymouth satellite	18.0	8	318.0	150.0	3436	11.0	70	usa
3	amc rebel sst	16.0	8	304.0	150.0	3433	12.0	70	usa
4	ford torino	17.0	8	302.0	140.0	3449	10.5	70	usa

- To view first 5 rows we use head() function

- This function returns the first `n` rows for the object basedon position.

- It is useful for quickly testing if your objecthas the right type of data in it.

- If n is larger than the number of rows, this function returns all rows.

Displaying last 5 rows

In [5]:
```
# Display the last 5 rows
data.tail()
```

Out[5]:

	name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	ori
393	ford mustang gl	27.0	4	140.0	86.0	2790	15.6	82	u
394	vw pickup	44.0	4	97.0	52.0	2130	24.6	82	euro
395	dodge rampage	32.0	4	135.0	84.0	2295	11.6	82	u
396	ford ranger	28.0	4	120.0	79.0	2625	18.6	82	u
397	chevy s-10	31.0	4	119.0	82.0	2720	19.4	82	u

- To view last 5 rows we use tail() function

- This function returns last `n` rows from the object based on position.

- It is useful for quickly verifying data, for example,after sorting or appending rows.

Display more information about all attributes.

In [6]:
```
# Get more information about dataset
data.info()
```

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

- This method prints information about a DataFrame includingthe index dtype and columns, non-null values and memory usage.

Observation

- We get comumns name, non-null count, data type

- To get more information about dataset we use info() function

---

Changing the datatype

In [7]:
```
# The following lines convert the column in the DataFrame to the 'category
data['cylinders'] = data['cylinders'].astype('category')
data['model_year'] = data['model_year'].astype('category')
data['origin'] = data['origin'].astype('category')
```

Observation

- The column is converted to the 'category' data type

To get number of rows and columns

In [8]:
```
# Get number of rows and columns
data.shape
```

Out[8]:
```
(398, 9)
```

Observation

- To get number of rows and columns we use shape function

- number of rows are 398

- number of columns are 9

---

checking for mising values

In [9]:
```
# Check for missing values
data.isnull().sum()
```

Out[9]:
```
name            0
mpg             0
cylinders       0
displacement    0
horsepower      6
weight          0
acceleration    0
model_year      0
origin          0
dtype: int64
```

Detect missing values.

Observation

- we get null values by using isnull() functiion

- There is null value in horsepower

Handling null values

In [10]:
```
# Handling null values
data.dropna(inplace=True)
```

Note

- Null Values can be halded by replacing or dropping them

- in this situation we use droping method as the valus are less and cannot effect the dataset

Reset index

In [11]:
```
#after deleting index is set
data.reset_index(drop=True, inplace=True)
data.shape
```

Out[11]:
```
(392, 9)
```

Note

reste_index() method used to

Reset the index of the DataFrame, and use the default one instead.

If the DataFrame has a Multiindex, this method can remove one or more levels.

In [12]:
```
# check for nullvalue again
data.isnull().sum()
```

Out[12]:
```
name            0
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
model_year      0
origin          0
dtype: int64
```

Note

isnull() method is use to Detect missing values.

Observation

- After dropping null values we can observe that all the null values are removed

---

Generate descriptive statistics

In [13]:
```
# Summary statistics of numerical variables
data.describe()
```

Out[13]:

	mpg	displacement	horsepower	weight	acceleration
count	392.000000	392.000000	392.000000	392.000000	392.000000
mean	23.445918	194.411990	104.469388	2977.584184	15.541327
std	7.805007	104.644004	38.491160	849.402560	2.758864
min	9.000000	68.000000	46.000000	1613.000000	8.000000
25%	17.000000	105.000000	75.000000	2225.250000	13.775000
50%	22.750000	151.000000	93.500000	2803.500000	15.500000
75%	29.000000	275.750000	126.000000	3614.750000	17.025000
max	46.600000	455.000000	230.000000	5140.000000	24.800000

Note

describe() method Generate descriptive statistics.

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution

For numeric data, the result's index will include count, mean, std, min, max as well as lower, 50 and upper percentiles. By default the lower percentile is 25 and the upper percentile is 75, The

---

Dublicate values

In [14]:
```
# Check for Duplicate Value
data[data.duplicated()]
```

Out[14]:

name	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin

Observation

- it check row vise data

- there is no dublicates in dataset

Number of vehicle models year vise

In [15]:
```
# Display total number of vehicles
d = data.groupby('model_year')['name'].count().sort_values(ascending=Fals
print(d)
```

```
model_year
73    40
78    36
76    34
75    30
82    30
70    29
79    29
72    28
77    28
81    28
71    27
80    27
74    26
Name: name, dtype: int64
```

Observation

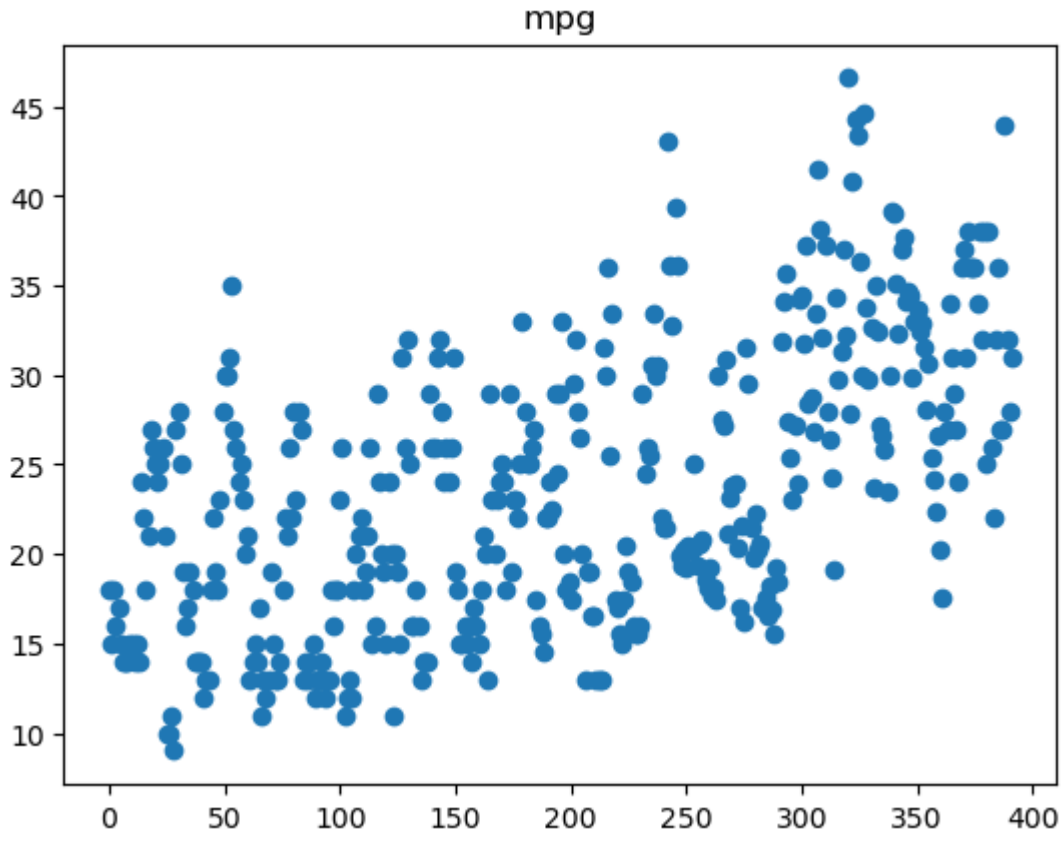
- The 'd' Series displays the total number of vehicles for each 'model_year', sorted in descending order based on the count.

- The highest count appears at the top, indicating the year with the most vehicles, while the lowest count appears at the bottom.

Univariate Data Visualizations

Univariate scatter plot

In [16]:
```
# Scatter plot of 'mpg' against the DataFrame index

plt.scatter(data.index,data['mpg'])
plt.title('mpg')
plt.show()
```



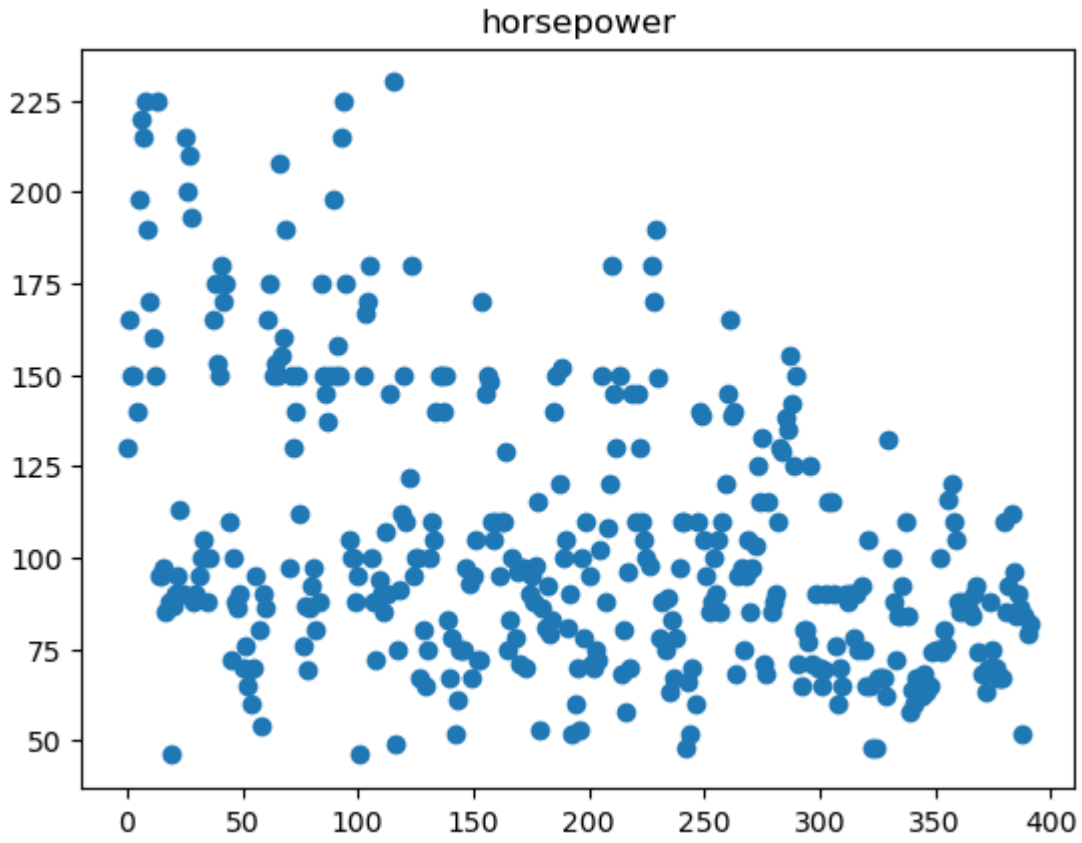
Observation

-this plot shows the value of mpg according to there serial no.

- Scatter plots are useful for identifying patterns, trends, and potential outliers in the data.

In [17]:

```
# Scatter plot of 'horsepower' against the DataFrame index

plt.scatter(data.index,data['horsepower'])
plt.title('horsepower')
plt.show()
```



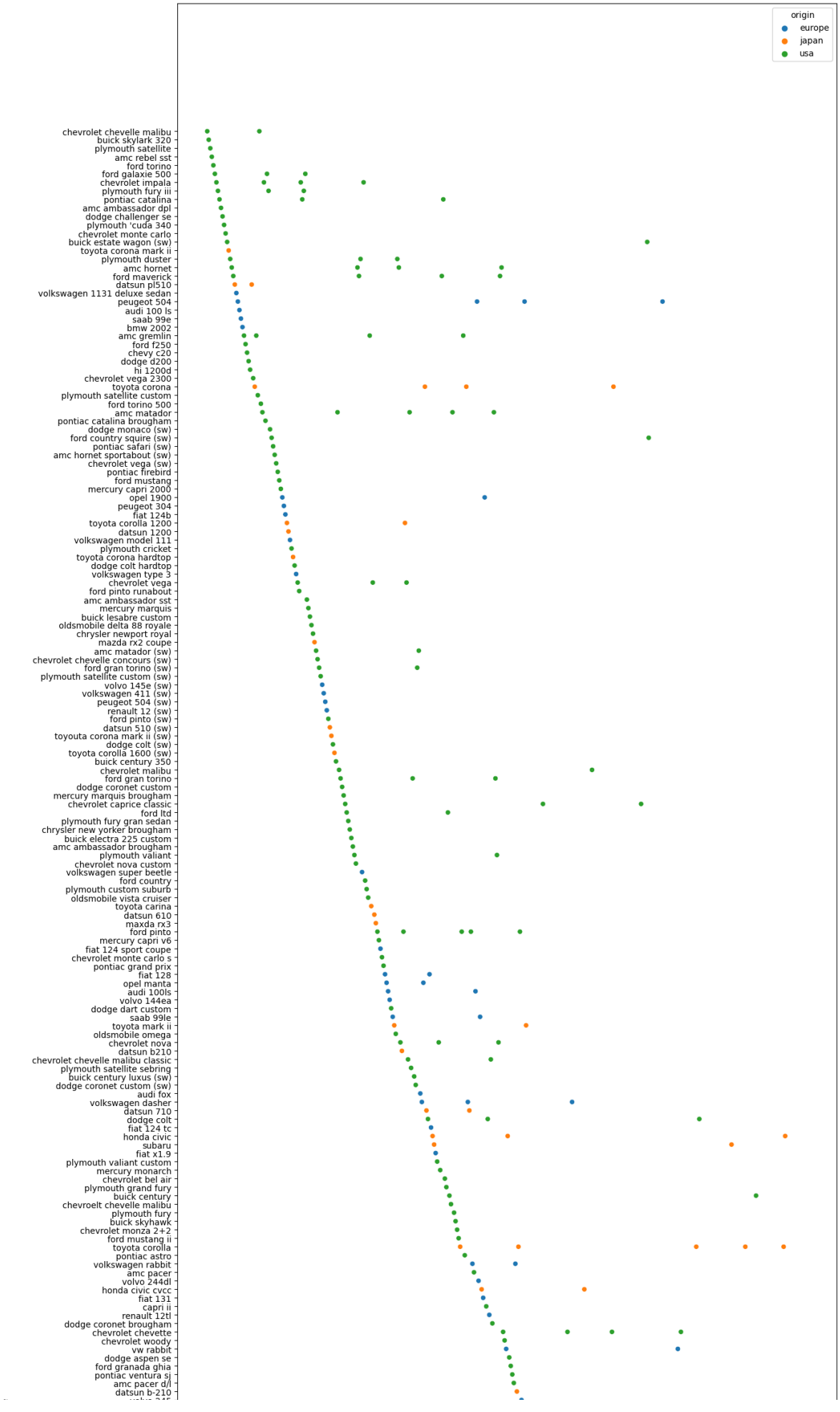
Observation

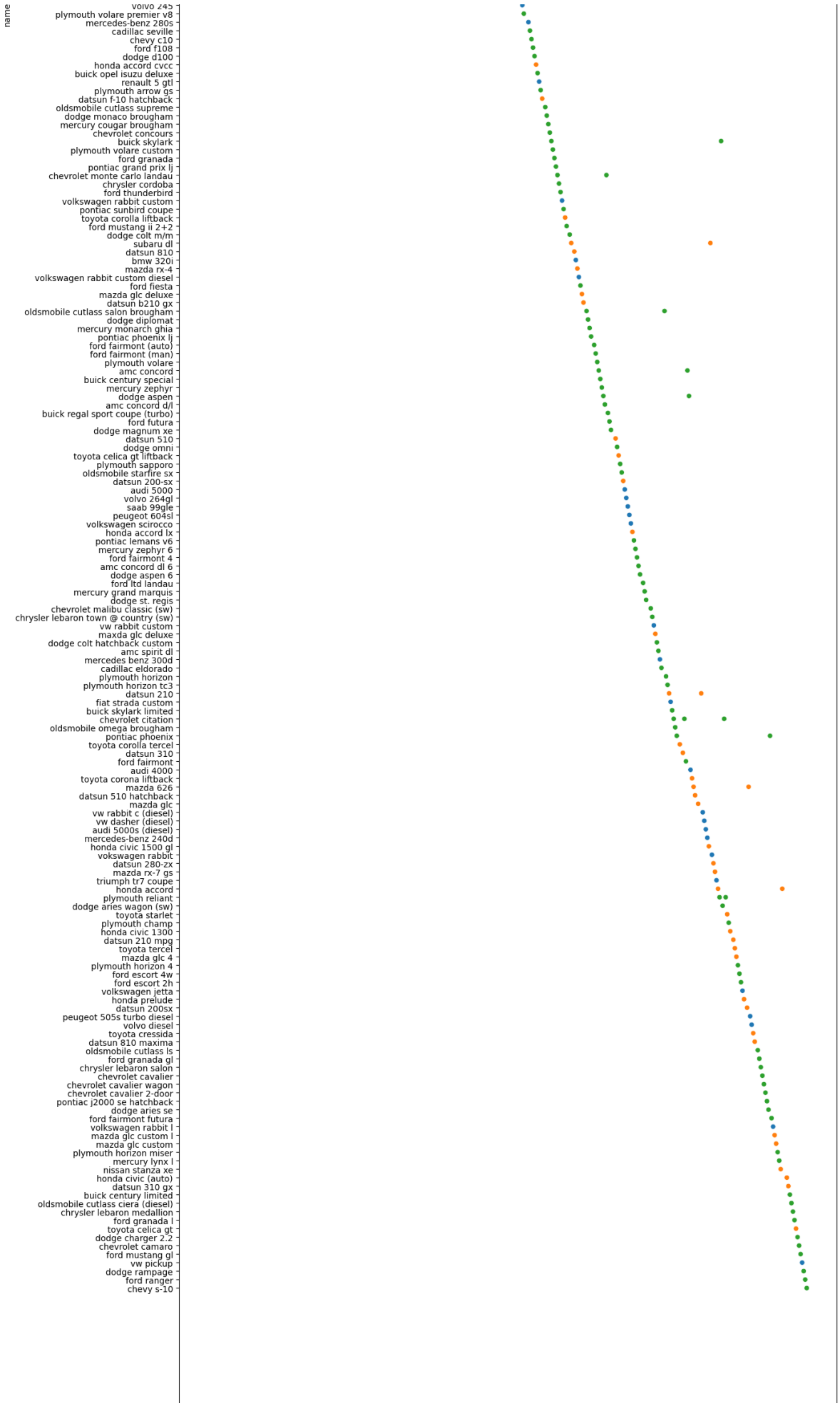
- the plot shows the horse p[ower of vahicles seral wise

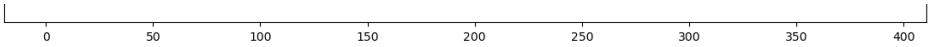
Univariate scatter plot by considering its origin

In [18]:
```
# Scatter plot of 'name' against the DataFrame index, colored by 'origin'
plt.figure(figsize=(14,60))
sns.scatterplot(x=data.index,y=data['name'],hue=data['origin'])
```

Out[18]:
```
<Axes: ylabel='name'>
```


origin
europe
japan
usa
chevrolet chevelle malibu
buick skylark 320
plymouth satellite
amc rebel sst
ford torino
ford galaxie 500
chevrolet impala
plymouth fury iii
pontiac catalina
amc ambassador dpl
dodge challenger se
plymouth 'cuda 340
chevrolet monte carlo
buick estate wagon (sw)
toyota corona mark ii
plymouth duster
amc hornet
ford maverick
datsun pl510
volkswagen 1131 deluxe sedan
peugeot 504
audi 100 ls
saab 99e
bmw 2002
amc gremlin
ford f250
chevy c20
dodge d200
hi 1200d
chevrolet vega 2300
toyota corona
plymouth satellite custom
ford torino 500
amc matador
pontiac catalina brougham
dodge monaco (sw)
ford country squire (sw)
pontiac safari (sw)
amc hornet sportabout (sw)
chevrolet vega (sw)
pontiac firebird
ford mustang
mercury capri 2000
opel 1900
peugeot 304
fiat 124b
toyota corolla 1200
datsun 1200
volkswagen model 111
plymouth cricket
toyota corona hardtop
dodge colt hardtop
volkswagen type 3
chevrolet vega
ford pinto runabout
amc ambassador sst
mercury marquis
buick lesabre custom
oldsmobile delta 88 royale
chrysler newport royal
mazda rx2 coupe
amc matador (sw)
chevrolet chevelle concours (sw)
ford gran torino (sw)
plymouth satellite custom (sw)
volvo 145e (sw)
volkswagen 411 (sw)
peugeot 504 (sw)
renault 12 (sw)
ford pinto (sw)
datsun 510 (sw)
toyouta corona mark ii (sw)
dodge colt (sw)
toyota corolla 1600 (sw)
buick century 350
chevrolet malibu
ford gran torino
dodge coronet custom
mercury marquis brougham
chevrolet caprice classic
ford ltd
plymouth fury gran sedan
chrysler new yorker brougham
buick electra 225 custom
amc ambassador brougham
plymouth valiant
chevrolet nova custom
volkswagen super beetle
ford country
plymouth custom suburb
oldsmobile vista cruiser
toyota carina
datsun 610
maxda rx3
ford pinto
mercury capri v6
fiat 124 sport coupe
chevrolet monte carlo s
pontiac grand prix
fiat 128
opel manta
audi 100ls
volvo 144ea
dodge dart custom
saab 99le
toyota mark ii
oldsmobile omega
chevrolet nova
datsun b210
chevrolet chevelle malibu classic
plymouth satellite sebring
buick century luxus (sw)
dodge coronet custom (sw)
audi fox
volkswagen dasher
datsun 710
dodge colt
fiat 124 tc
honda civic
subaru
fiat x1.9
plymouth valiant custom
mercury monarch
chevrolet bel air
plymouth grand fury
buick century
chevroelt chevelle malibu
plymouth fury
buick skyhawk
chevrolet monza 2+2
ford mustang ii
toyota corolla
pontiac astro
volkswagen rabbit
amc pacer
volvo 244dl
honda civic cvcc
fiat 131
capri ii
renault 12tl
dodge coronet brougham
chevrolet chevette
chevrolet woody
vw rabbit
dodge aspen se
ford granada ghia
pontiac ventura sj
amc pacer d/l
datsun b-210


name
volvo 245
plymouth volare premier v8
mercedes-benz 280s
cadillac seville
chevy c10
ford f108
dodge d100
honda accord cvcc
buick opel isuzu deluxe
renault 5 gtl
plymouth arrow gs
datsun f-10 hatchback
oldsmobile cutlass supreme
dodge monaco brougham
mercury cougar brougham
chevrolet concours
buick skylark
plymouth volare custom
ford granada
pontiac grand prix lj
chevrolet monte carlo landau
chrysler cordoba
ford thunderbird
volkswagen rabbit custom
pontiac sunbird coupe
toyota corolla liftback
ford mustang ii 2+2
dodge colt m/m
subaru dl
datsun 810
bmw 320i
mazda rx-4
volkswagen rabbit custom diesel
ford fiesta
mazda glc deluxe
datsun b210 gx
oldsmobile cutlass salon brougham
dodge diplomat
mercury monarch ghia
pontiac phoenix lj
ford fairmont (auto)
ford fairmont (man)
plymouth volare
amc concord
buick century special
mercury zephyr
dodge aspen
amc concord d/l
buick regal sport coupe (turbo)
ford futura
dodge magnum xe
datsun 510
dodge omni
toyota celica gt liftback
plymouth sapporo
oldsmobile starfire sx
datsun 200-sx
audi 5000
volvo 264gl
saab 99gle
peugeot 604sl
volkswagen scirocco
honda accord lx
pontiac lemans v6
mercury zephyr 6
ford fairmont 4
amc concord dl 6
dodge aspen 6
ford ltd landau
mercury grand marquis
dodge st. regis
chevrolet malibu classic (sw)
chrysler lebaron town @ country (sw)
vw rabbit custom
maxda glc deluxe
dodge colt hatchback custom
amc spirit dl
mercedes benz 300d
cadillac eldorado
plymouth horizon
plymouth horizon tc3
datsun 210
fiat strada custom
buick skylark limited
chevrolet citation
oldsmobile omega brougham
pontiac phoenix
toyota corolla tercel
datsun 310
ford fairmont
audi 4000
toyota corona liftback
mazda 626
datsun 510 hatchback
mazda glc
vw rabbit c (diesel)
vw dasher (diesel)
audi 5000s (diesel)
mercedes-benz 240d
honda civic 1500 gl
vokswagen rabbit
datsun 280-zx
mazda rx-7 gs
triumph tr7 coupe
honda accord
plymouth reliant
dodge aries wagon (sw)
toyota starlet
plymouth champ
honda civic 1300
datsun 210 mpg
toyota tercel
mazda glc 4
plymouth horizon 4
ford escort 4w
ford escort 2h
volkswagen jetta
honda prelude
datsun 200sx
peugeot 505s turbo diesel
volvo diesel
toyota cressida
datsun 810 maxima
oldsmobile cutlass ls
ford granada gl
chrysler lebaron salon
chevrolet cavalier
chevrolet cavalier wagon
chevrolet cavalier 2-door
pontiac j2000 se hatchback
dodge aries se
ford fairmont futura
volkswagen rabbit l
mazda glc custom l
mazda glc custom
plymouth horizon miser
mercury lynx l
nissan stanza xe
honda civic (auto)
datsun 310 gx
buick century limited
oldsmobile cutlass ciera (diesel)
chrysler lebaron medallion
ford granada l
toyota celica gt
dodge charger 2.2
chevrolet camaro
ford mustang gl
vw pickup
dodge rampage
ford ranger
chevy s-10



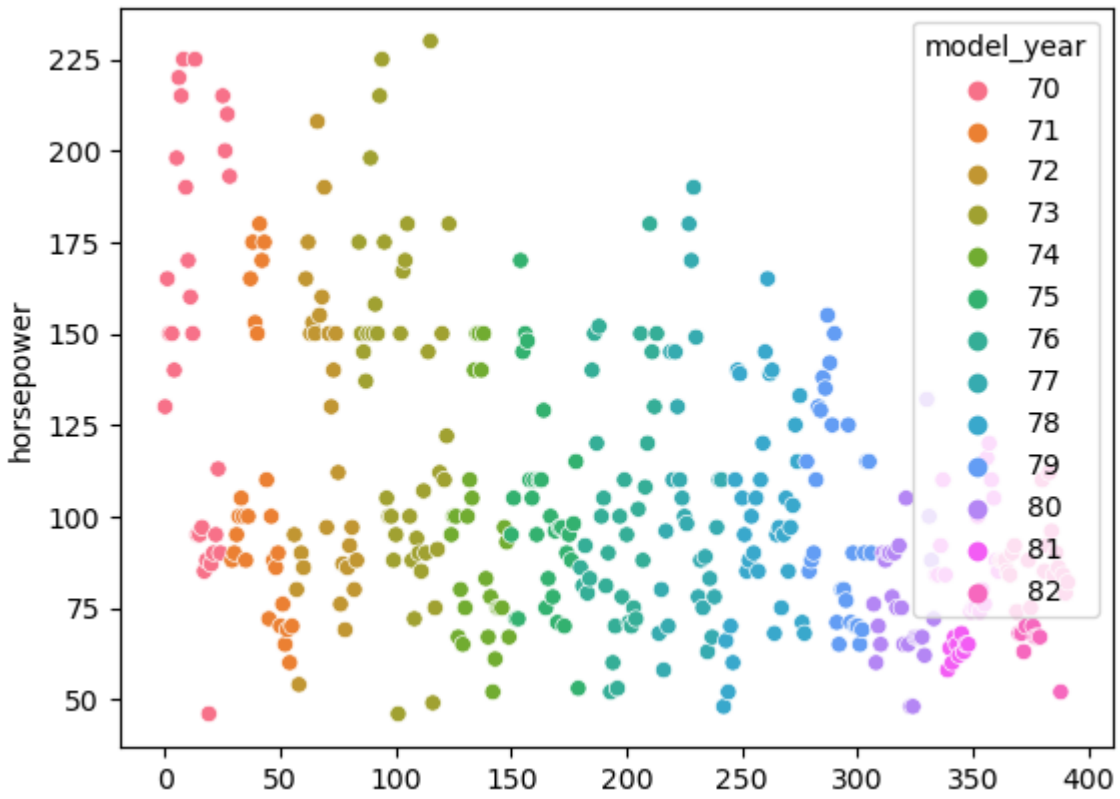
Observation

- the colors represent where the vehicles are origin and it seems that maximum cars are made in usa

Univariate scatter plot by consudering its year

In [19]:
```
# Scatter plot of 'horsepower' against the DataFrame index, colored by 'mo
sns.scatterplot(x=data.index,y=data['horsepower'],hue=data['model_year'])
```

Out[19]: <Axes: ylabel='horsepower'>

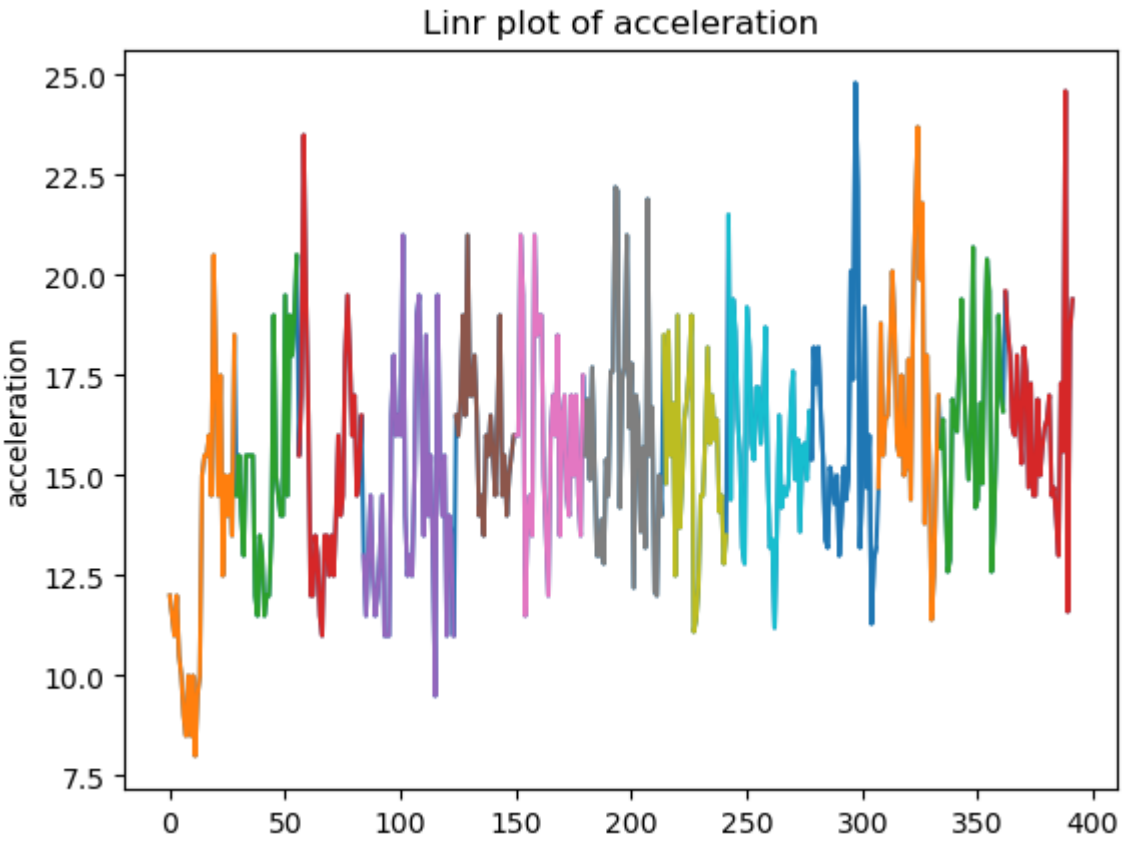


Observation

- The scatter plot visualizes the relationship between 'horsepower' and the DataFrame index, the colors shows the the modal year

Univariate linr plot

In [20]:
```
# Line plot of 'acceleration' against the DataFrame index, grouped by 'mod

plt.title('Linr plot of acceleration')
plt.ylabel('acceleration')

# Plotting the line plot for the entire dataset
plt.plot(data.index,data['acceleration'])

# Grouping the data by 'model_year' and plotting line plots for each group
for name,group in data.groupby('model_year'):
    plt.plot(group.index,group['acceleration'])

plt.show()
```



Observation

- The code above creates a line plot of the 'acceleration' values against the DataFrame index.

Univariate Strip plot

In [21]:
```
# Strip plot of 'cylinders'
sns.stripplot(y=data['cylinders'])
```

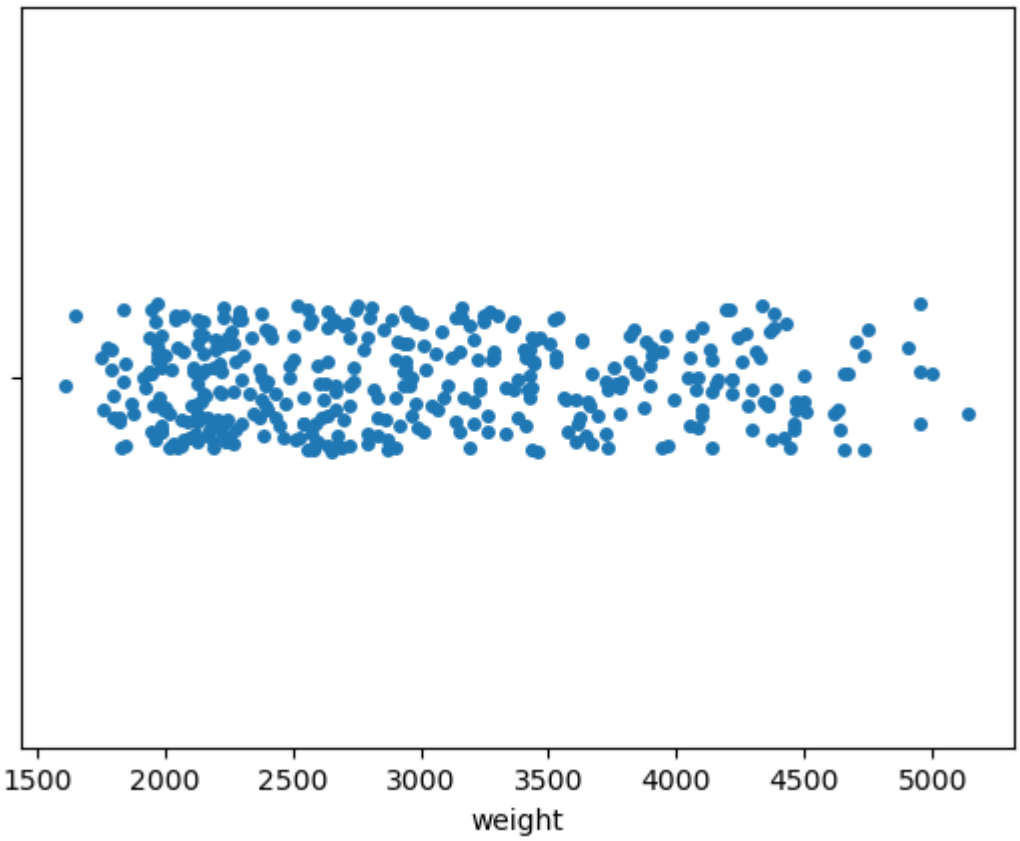
Out[21]: <Axes: ylabel='cylinders'>

Observation

- The strip plot visualizes the distribution of the 'cylinders' variable, which likely represents the number of cylinders in vehicles, as it is plotted on the y-axis.

In [22]:
```
# Strip plot of 'weight'
sns.stripplot(x=data['weight'])
```

Out[22]: <Axes: xlabel='weight'>



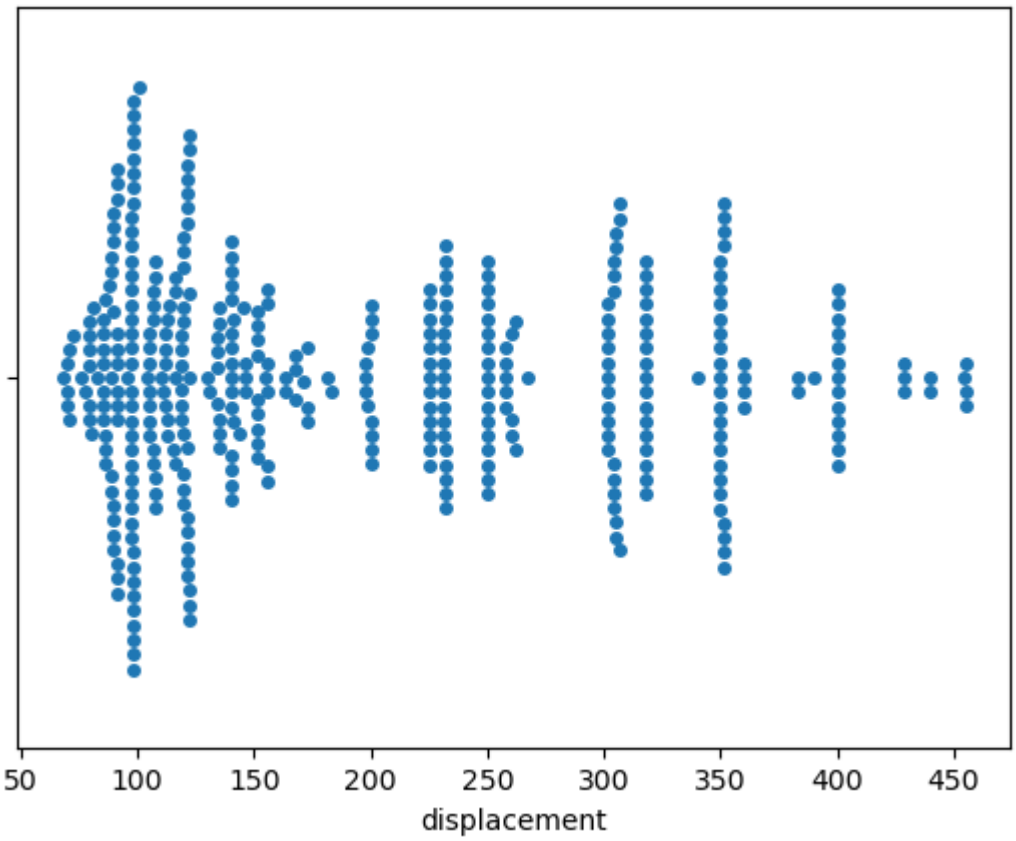
Observation

- The scatter plot visualizes the relationship between 'weight' and the DataFrame index, allowing us to examine how the 'weight' values are distributed along the index.

Univariate swarmp plot

In [23]:
```
# performing swamp plot of 'displacement'
sns.swarmplot(x= data['displacement'])
```

Out[23]: <Axes: xlabel='displacement'>

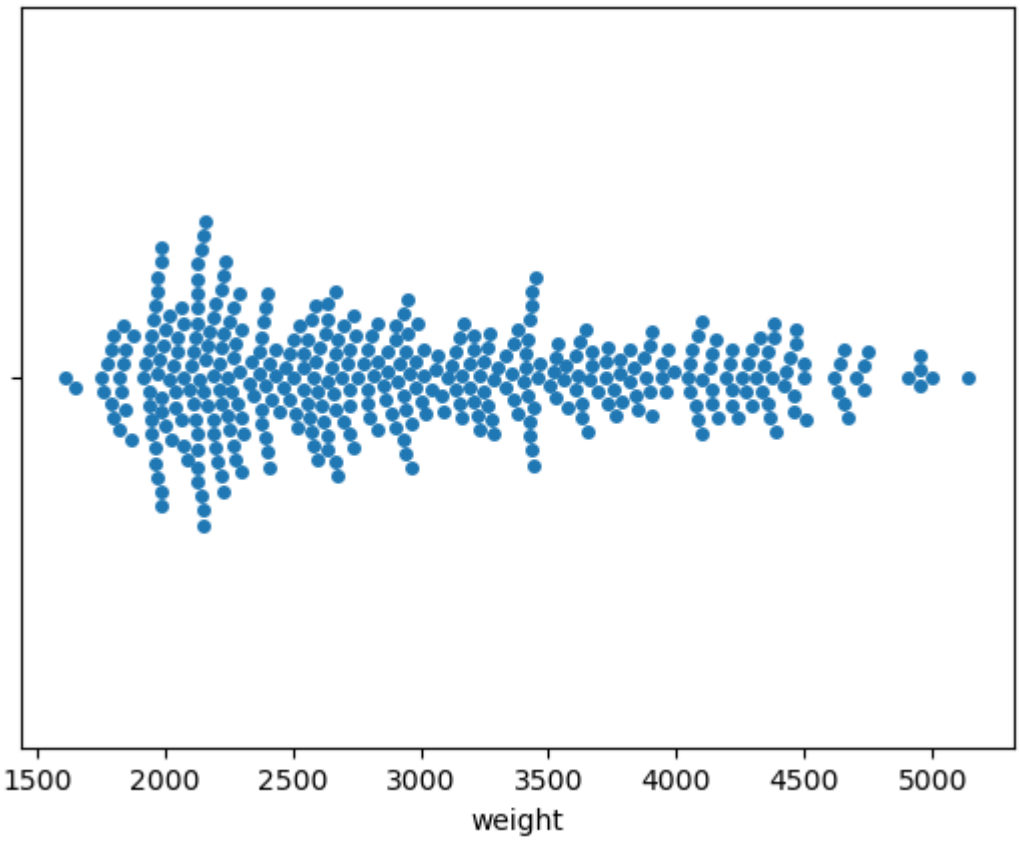


Observation

- The code above creates a swarm plot of the 'displacement' values. The 'displacement' values are plotted on the x-axis.

In [24]:
```
# performing swamp plot of 'weight'
sns.swarmplot(x=data['weight'])
```

Out[24]: <Axes: xlabel='weight'>



Observation

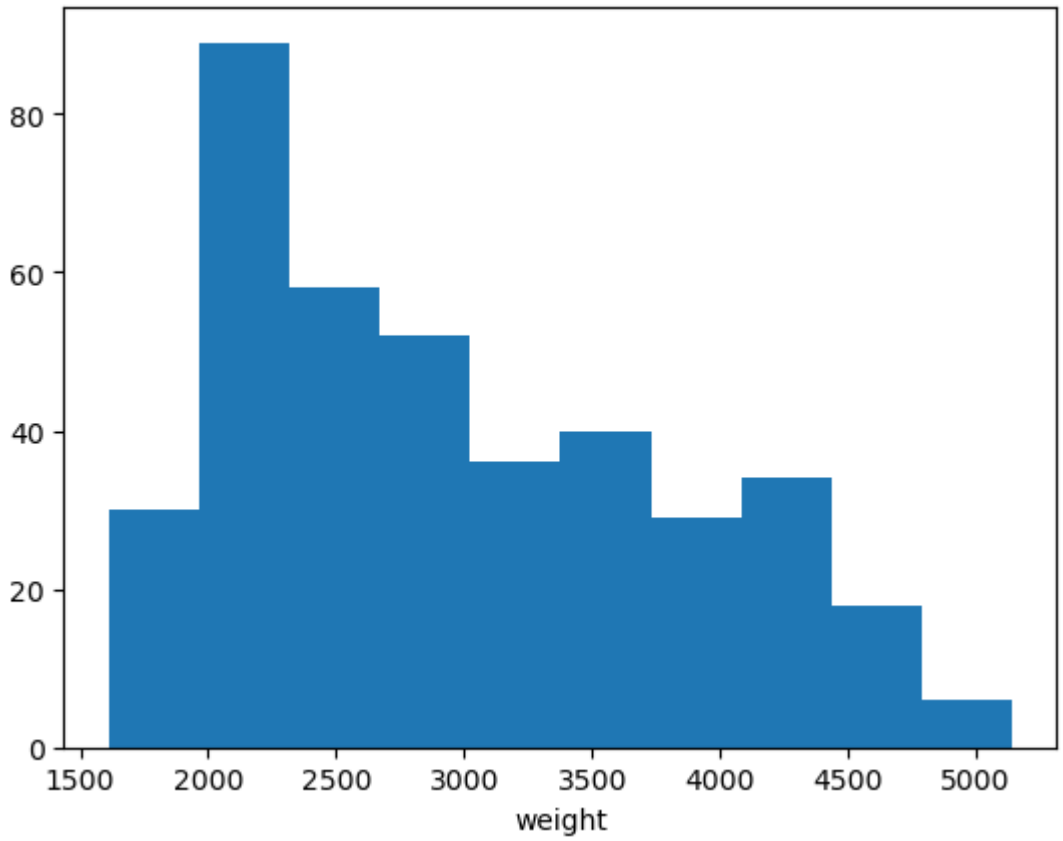
- The code above creates a swarm plot of the 'weight' values. The 'weight' values are plotted on the x-axis.

- The swarm plot is a categorical scatter plot that displays individual data points for the 'weight' variable along a single axis.<\i><\p>

Univariate Histogram()

In [25]:
```
# Performing histogram of 'weight'
plt.hist(x=data['weight'])
plt.xlabel('weight')
```

Out[25]:
```
Text(0.5, 0, 'weight')
```



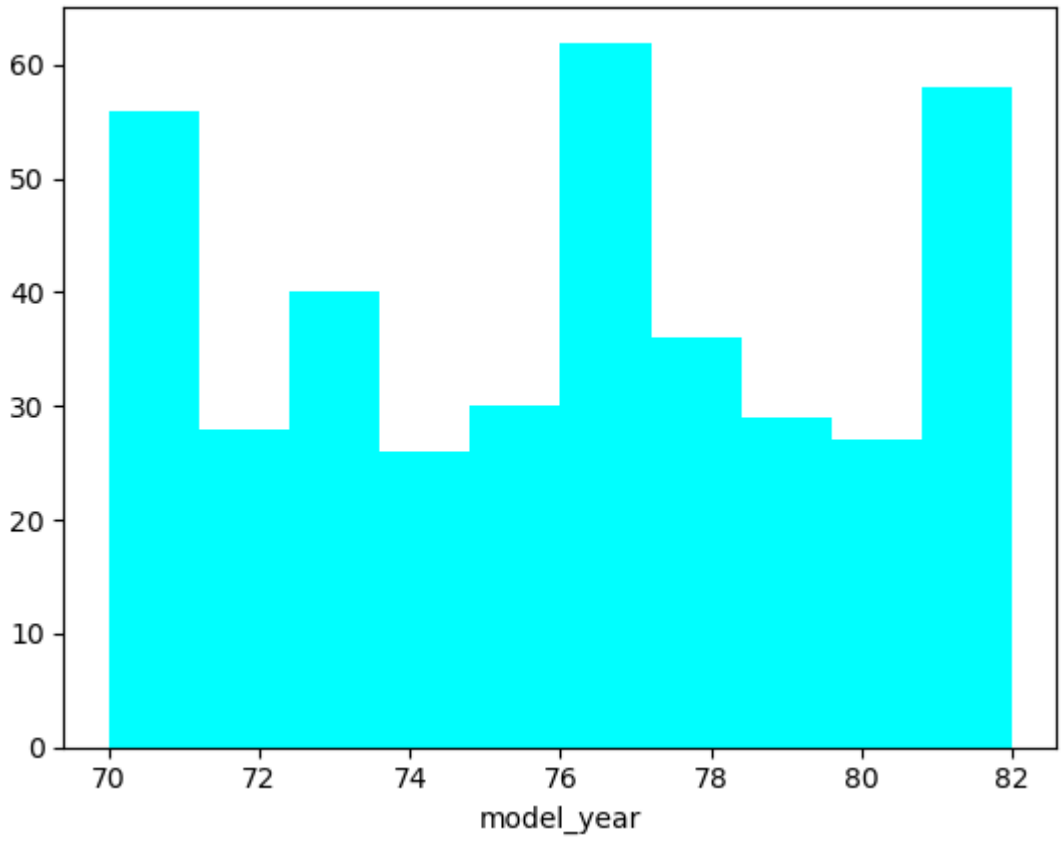
Observation

- The histogram effectively displays the distribution of 'weight' values, showing the frequency of data points falling within different 'weight' bins or intervals.

- Histograms are useful for visualizing the underlying distribution of continuous variables, such as 'weight', and provide insights into the data's central tendency and spread.

In [26]:
```
#  Histogram of 'model_year'
plt.hist(x=data['model_year'],color='cyan')
plt.xlabel('model_year')
```

Out[26]: Text(0.5, 0, 'model_year')



Observation

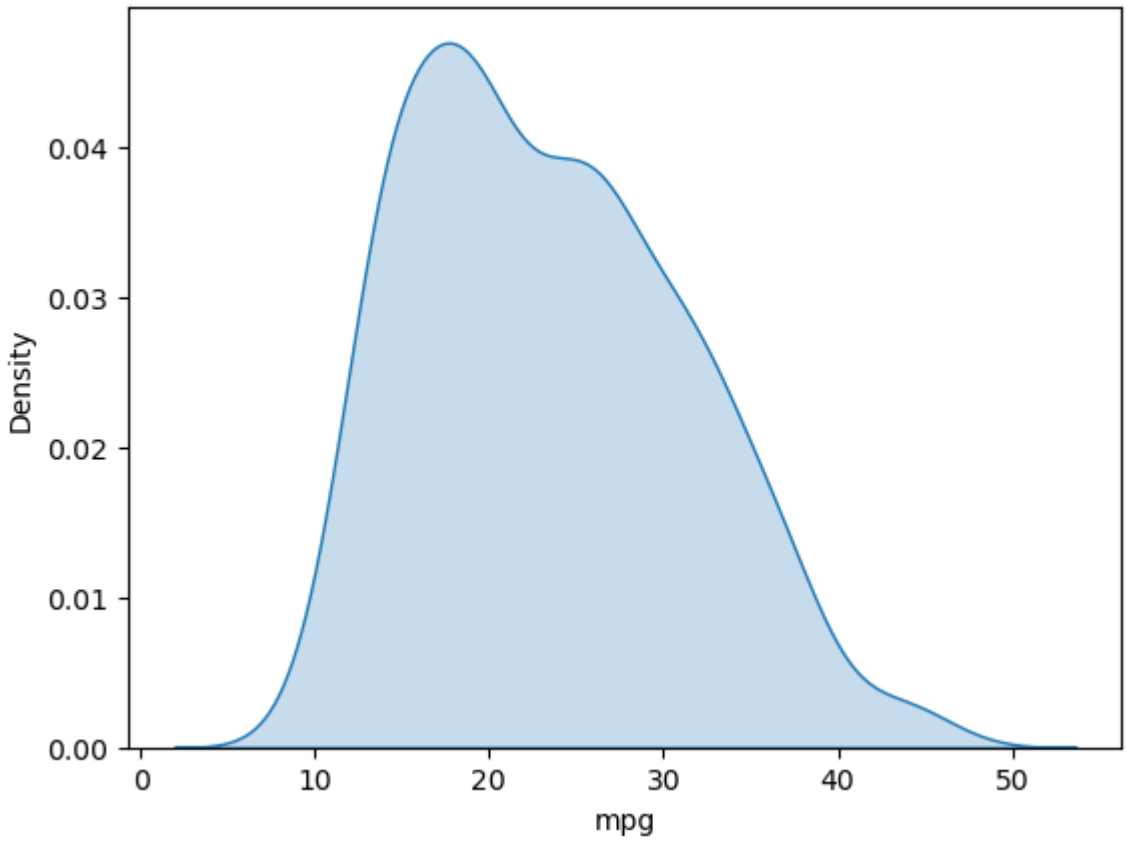
- The histogram effectively displays the distribution of 'model_year' values, showing the frequency of data points for each year.

- The x-axis represents the 'model_year' values, and the y-axis represents the frequency or count of data points corresponding to each year.

Univariate Density plot

In [27]:
```
# Kernel Density Estimation (KDE) plot of 'mpg'
sns.kdeplot(data['mpg'],shade=True)
```

Out[27]:
```
<Axes: xlabel='mpg', ylabel='Density'>
```



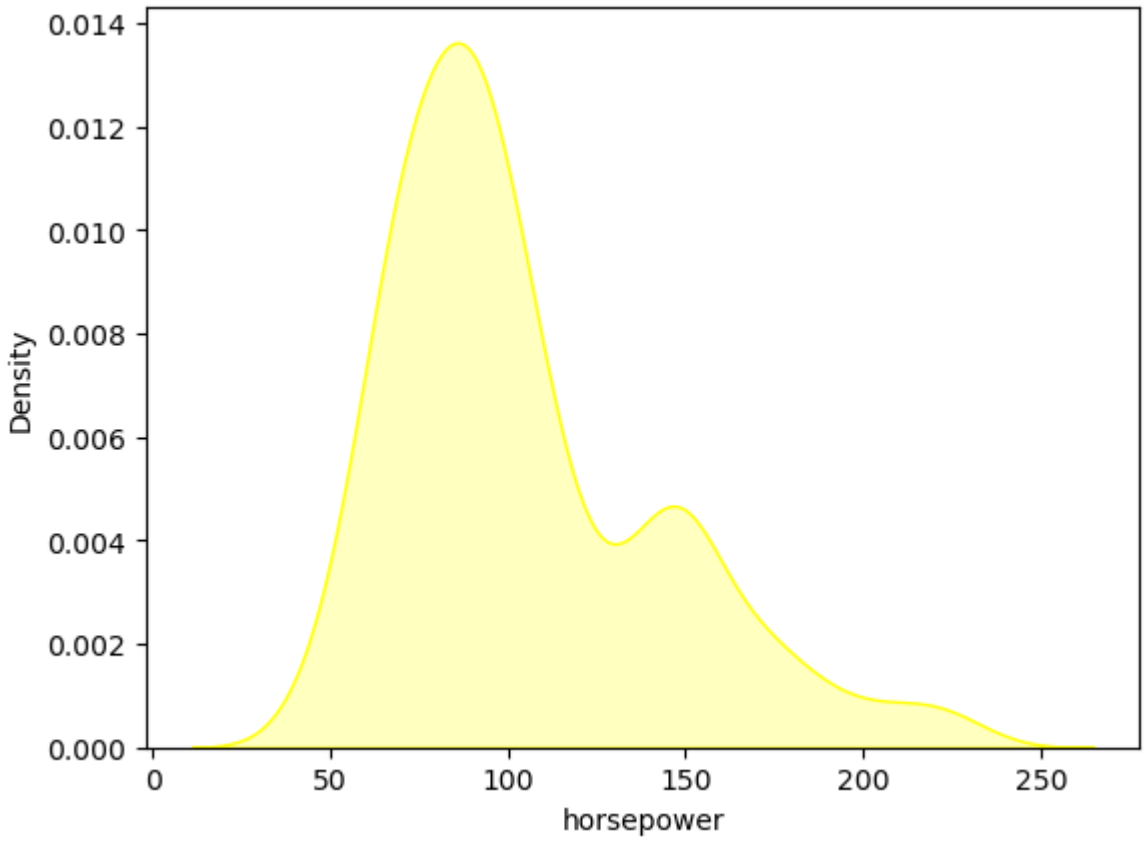
Observation

- The KDE plot provides a smooth and continuous estimation of the probability density function of the 'mpg' variable.

- The x-axis represents the 'mpg' values, and the y-axis represents the estimated density of data points at each 'mpg' value.

In [28]:
```
# Kernel Density Estimation (KDE) plot of 'horsepower'
sns.kdeplot(data['horsepower'],shade=True,color='yellow')
```

Out[28]: <Axes: xlabel='horsepower', ylabel='Density'>



Observation

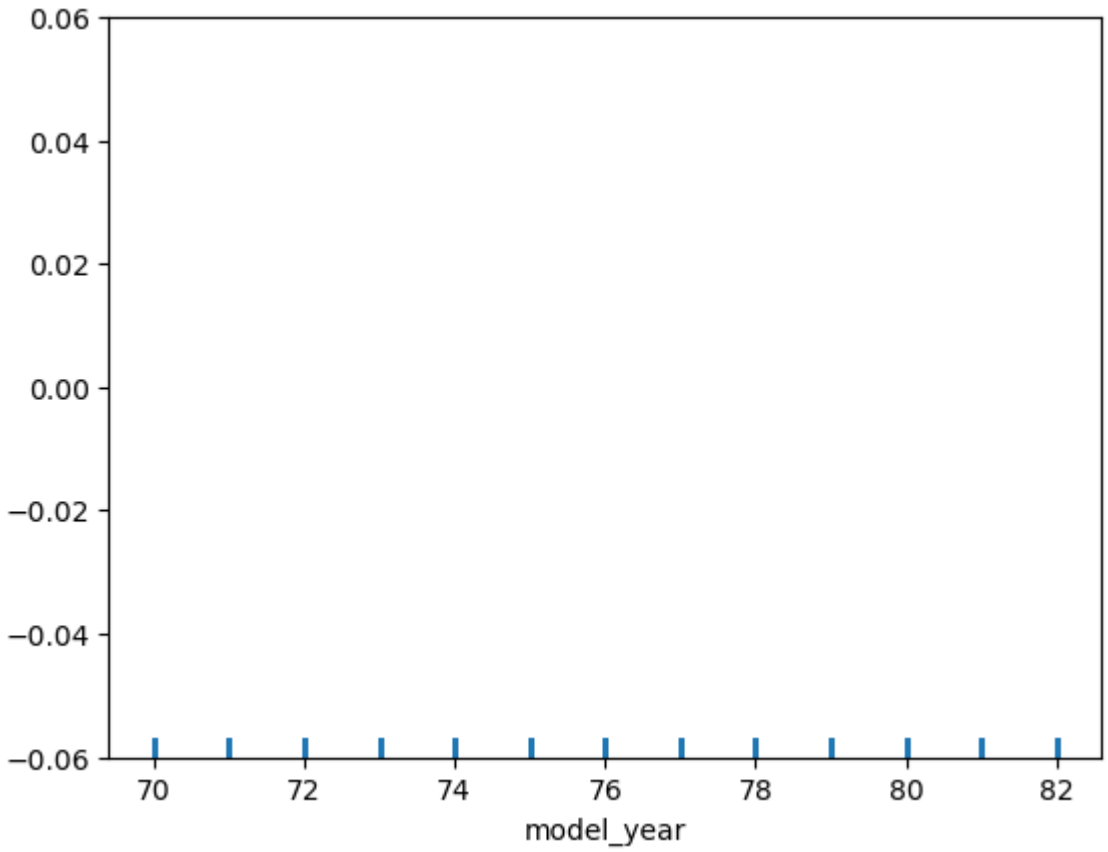
- The 'shade=True' parameter fills the area under the curve, making the plot visually appealing and easier to interpret. The 'color='yellow'' parameter sets the color of the KDE curve to yellow.

- KDE plots are useful for visualizing the underlying distribution of continuous variables, such as 'horsepower', without the need for specifying bin sizes (unlike histograms).

Univariate RUG plot

In [29]:
```
# Rug plot of 'model_year'
sns.rugplot(data['model_year'])
```

Out[29]: <Axes: xlabel='model_year'>



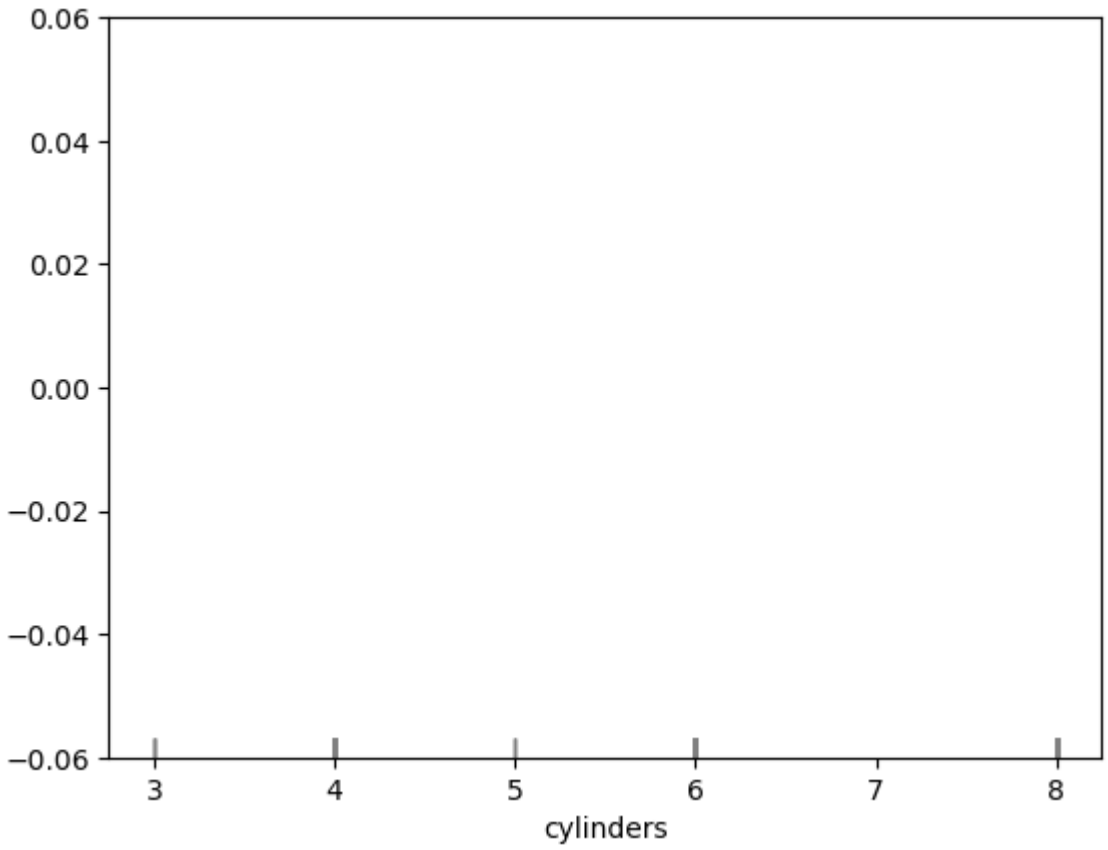
Observation

- The rug plot is a simple one-dimensional plot that displays individual data points as ticks along a single axis.

- The x-axis represents the 'model_year' values, and each tick (or "rug") along the axis indicates the occurrence of a single data point at that specific year.

In [30]:
```
# Rug plot of 'model_year'
sns.rugplot(data['cylinders'],color='grey')
```

Out[30]: <Axes: xlabel='cylinders'>



Observation

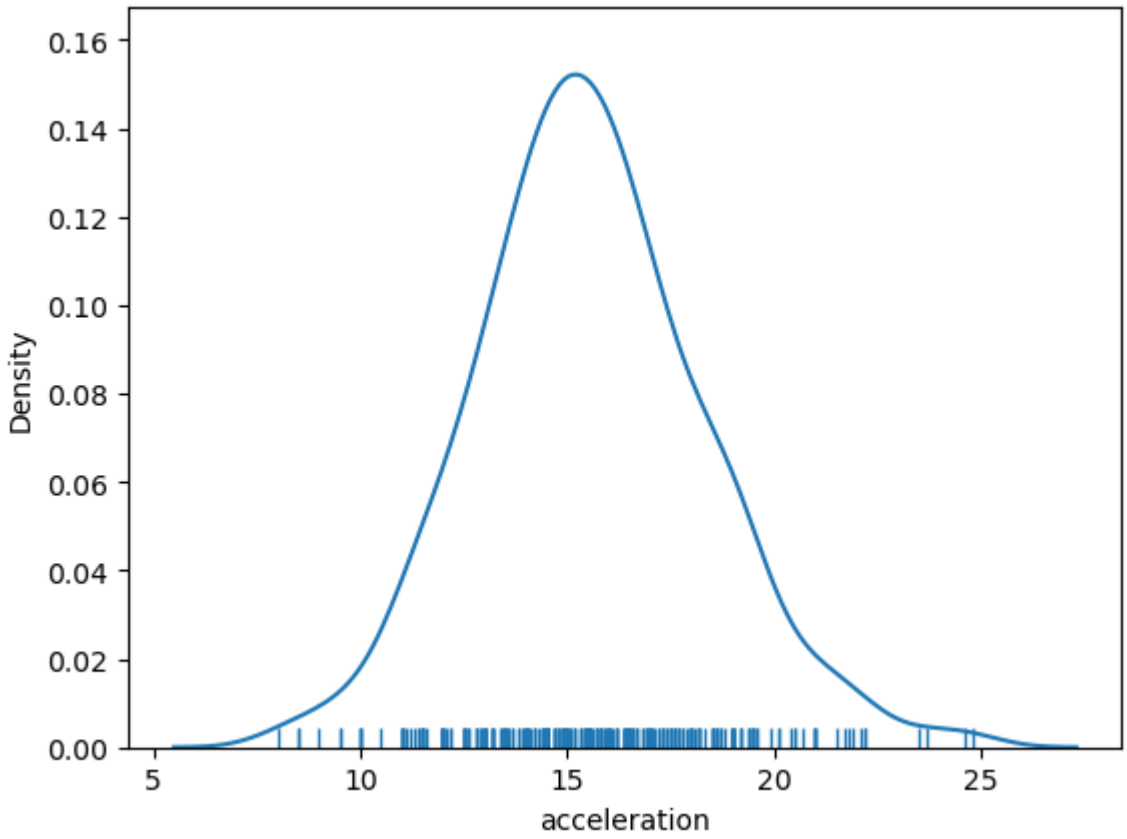
- Rug plots are used to visualize the distribution and density of data points along an axis, providing a basic overview of the data's positions.

- The rug plot is useful for understanding the distribution of values and quickly identifying the presence and density of data points for each year.

Univariate Kernel Density plot along with rugs

In [31]:
```
# density plot for' acceleration' with rug plot
sns.distplot(data['acceleration'],rug=True,hist=False)
```

Out[31]: <Axes: xlabel='acceleration', ylabel='Density'>



Observation

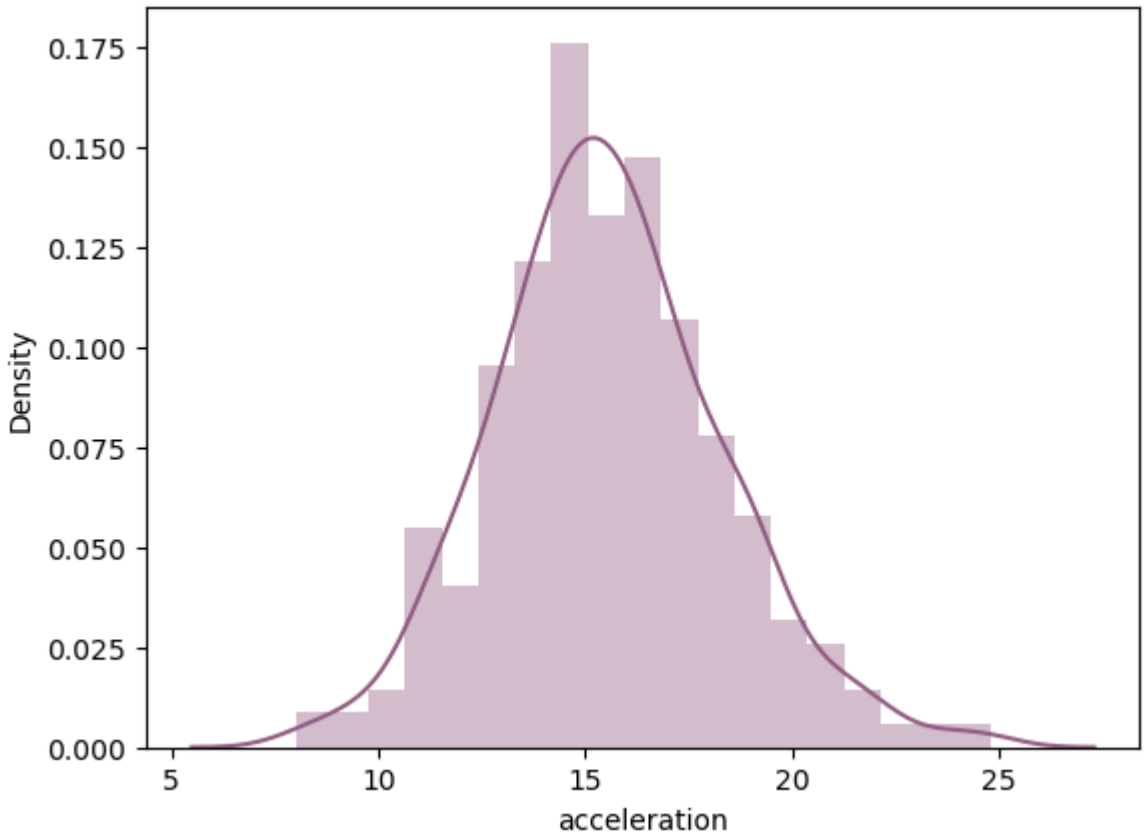
- The distribution plot, combined with the rug plot, provides a visual representation of the 'acceleration' data distribution on a continuous axis.

- The rug plot displays individual data points as ticks along the x-axis, indicating the occurrence of each 'acceleration' value.

Univariate Distplot

In [32]:
```
# Distribution plot of 'acceleration' with a custom color
sns.distplot(data['acceleration'],color='#915c83')
```

Out[32]: <Axes: xlabel='acceleration', ylabel='Density'>



Observation

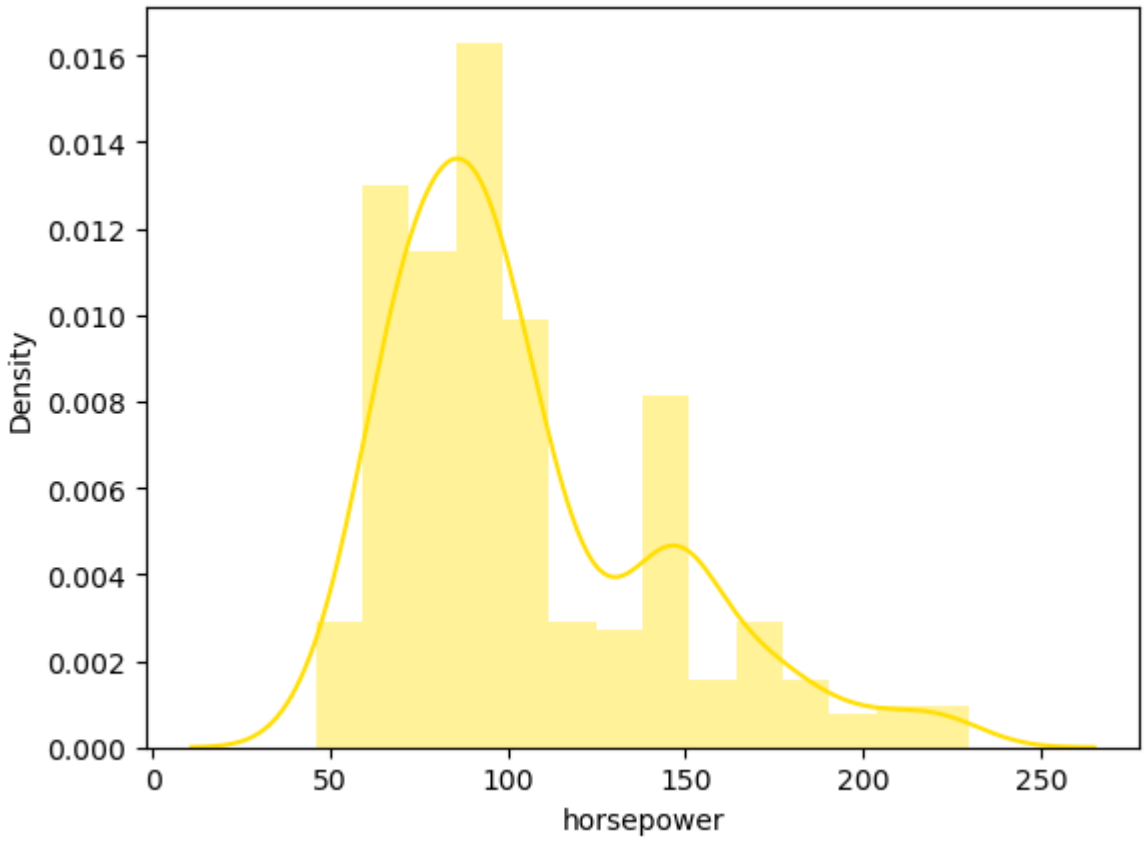
- The distribution plot effectively displays the distribution of 'acceleration' values using a combination of a histogram, kernel density estimation (KDE) curve, and rug plot.

- The histogram bars show the binned frequency of 'acceleration' values, providing insights into the data's central tendency and spread.

- The KDE curve provides a smooth estimation of the probability density function, offering a continuous representation of the 'acceleration' distribution.

In [33]:
```
# Distribution plot of 'horsepower' with a custom color
sns.distplot(data['horsepower'],color='#ffdf00')
```

Out[33]: <Axes: xlabel='horsepower', ylabel='Density'>

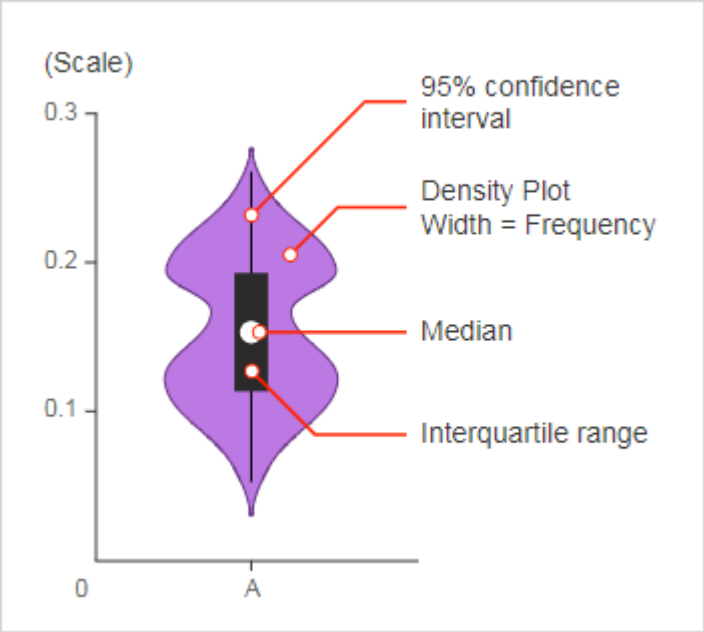


Observation

- The distribution plot with the custom color '#ffdf00' is a visually appealing and informative visualization to explore the distribution of 'horsepower' values in the dataset.

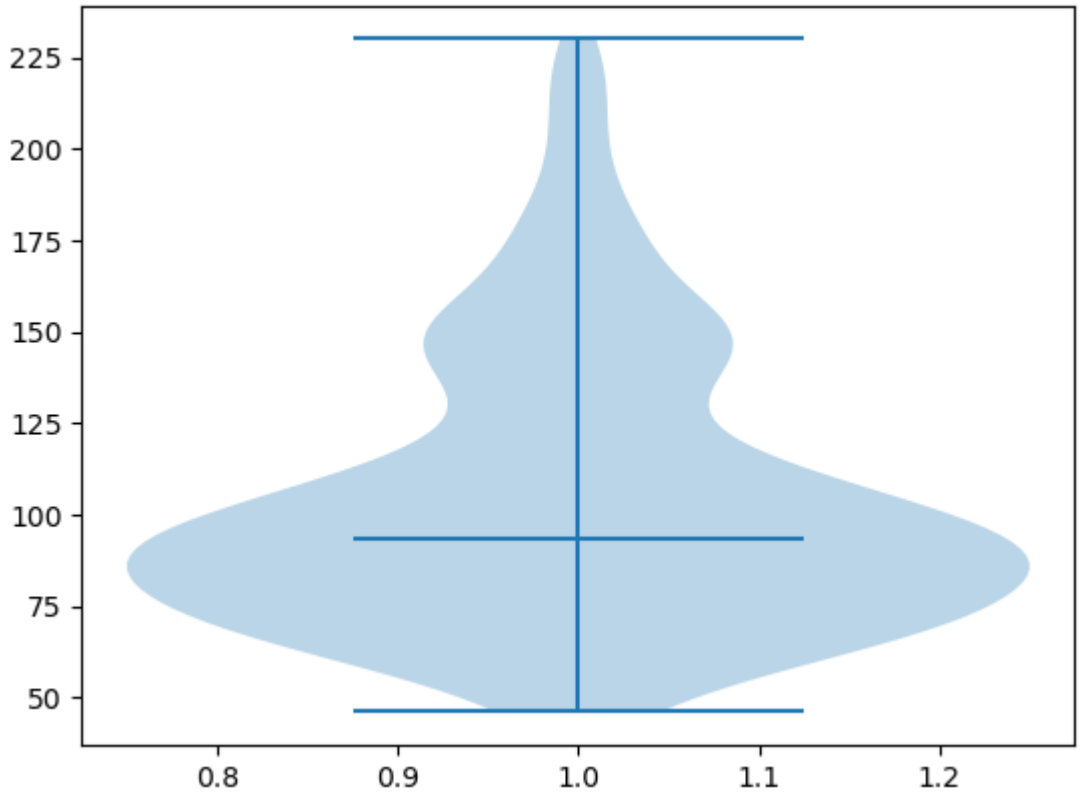
- is an essential step in the exploratory data analysis process.

Univariate Violin Plots



In [34]:
```
# Code: Violin plot of 'horsepower' with median markers displayed
plt.violinplot(data.horsepower,showmedians=True)
```

Out[34]:
```
{'bodies': [<matplotlib.collections.PolyCollection at 0x2985ef89c90>],
 'cmaxes': <matplotlib.collections.LineCollection at 0x2985ef89c00>,
 'cmins': <matplotlib.collections.LineCollection at 0x2985ef8a290>,
 'cbars': <matplotlib.collections.LineCollection at 0x2985ef8a5f0>,
 'cmedians': <matplotlib.collections.LineCollection at 0x2985ef8a950>}
```



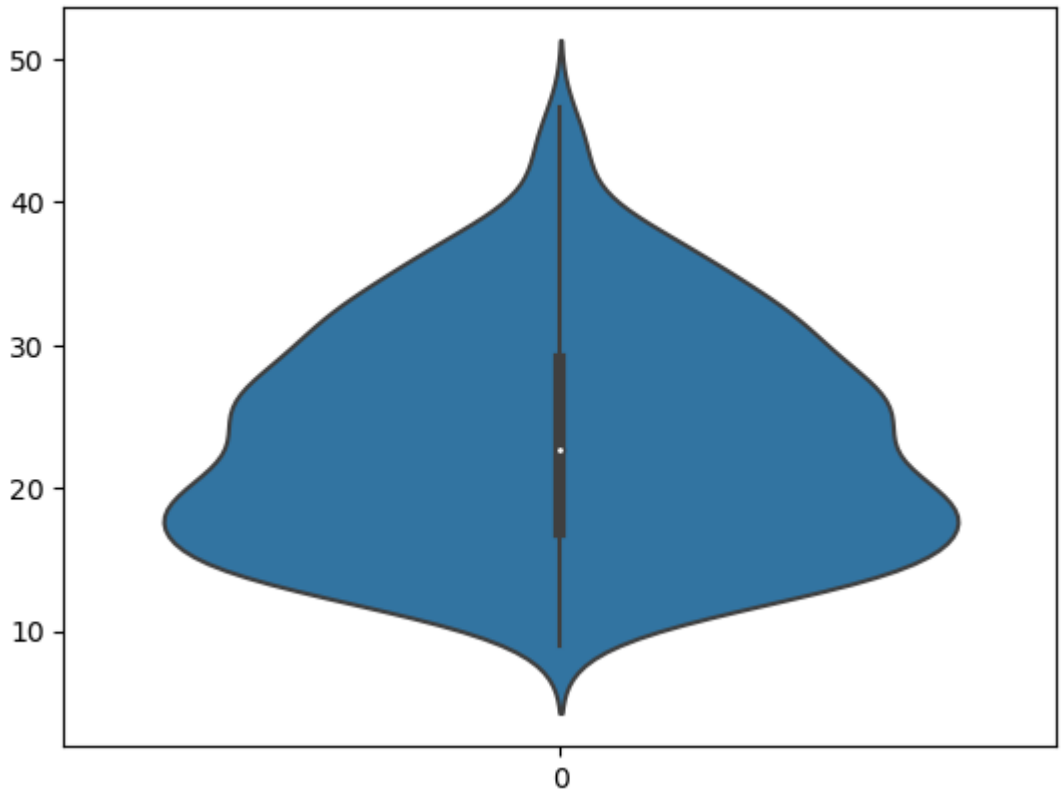
Observation

- It combines the benefits of a box plot and a kernel density plot

- The median markers displayed inside the violins further help understand the central tendency of the data.

In [35]:
```
# Violin plot of 'mpg'
sns.violinplot(data['mpg'])
```

Out[35]: <Axes: >



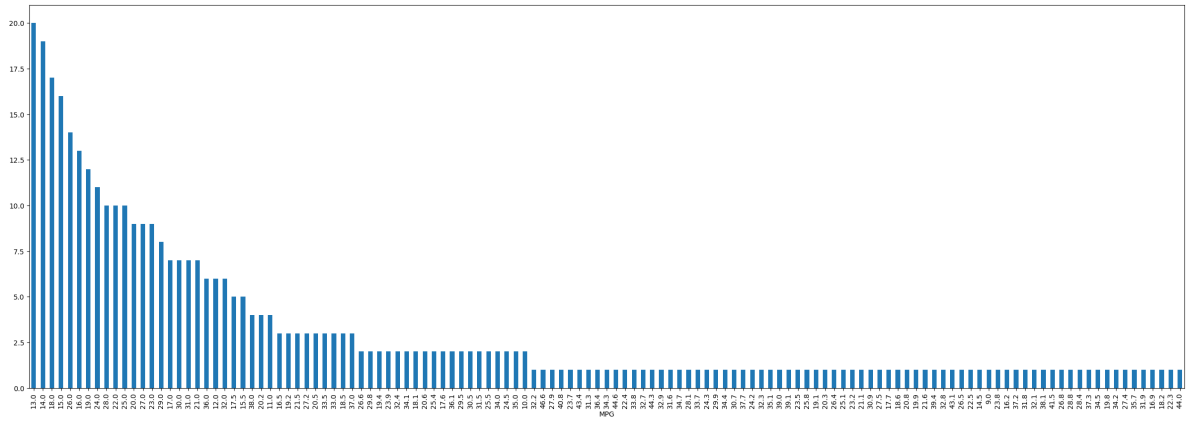
Observation

- The plot is particularly effective for understanding the density and variability of 'mpg' across different data points.

Univariate Bar Plot

In [36]:
```
# Bar plot of 'mpg' value counts

plt.figure(figsize=(30,10))
plt.xlabel('MPG')
data['mpg'].value_counts().plot.bar()
```

Out[36]:
```
<Axes: xlabel='MPG'>
```



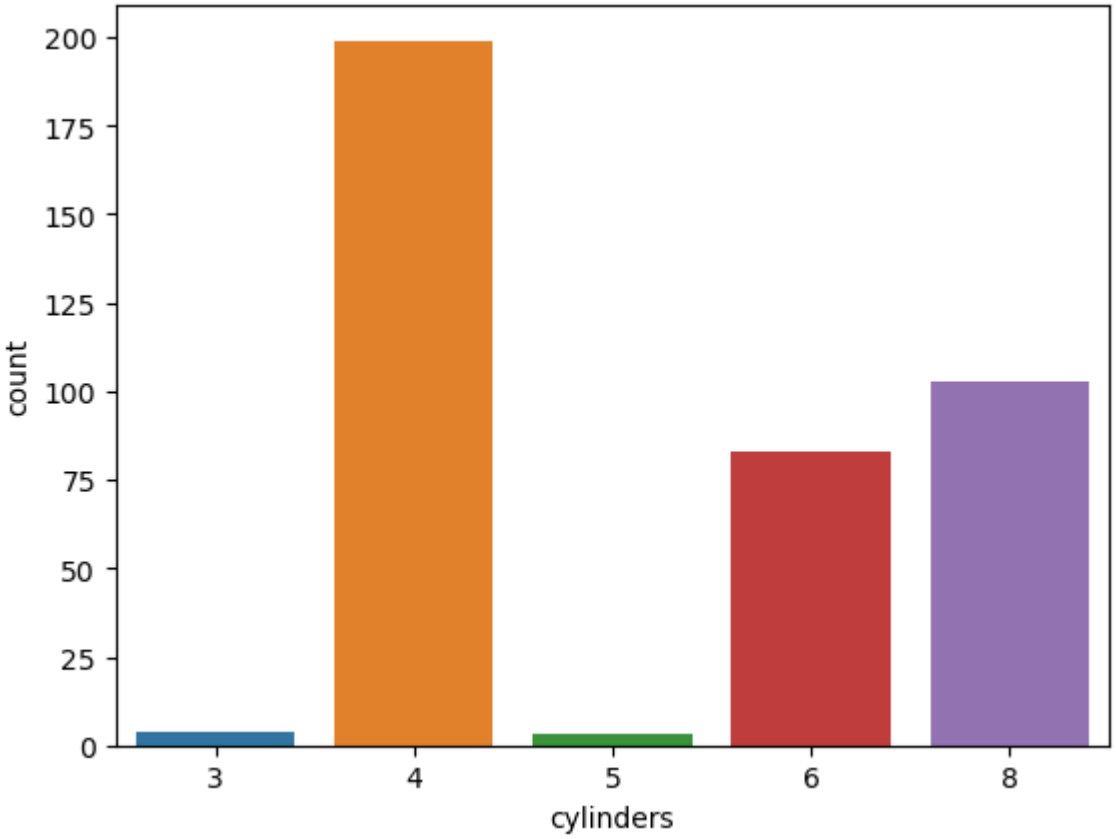
Observation

- The bar plot displays the count of each unique 'mpg' value as vertical bars along the x-axis.

In [37]:

```
# Count plot of 'cylinders' using Seaborn
sns.countplot(x=data['cylinders'])
print(data['cylinders'].value_counts())
print((data['cylinders'].value_counts()/398)*100)
```

```
4    199
8    103
6     83
3      4
5      3
Name: cylinders, dtype: int64
4    50.000000
8    25.879397
6    20.854271
3     1.005025
5     0.753769
Name: cylinders, dtype: float64
```



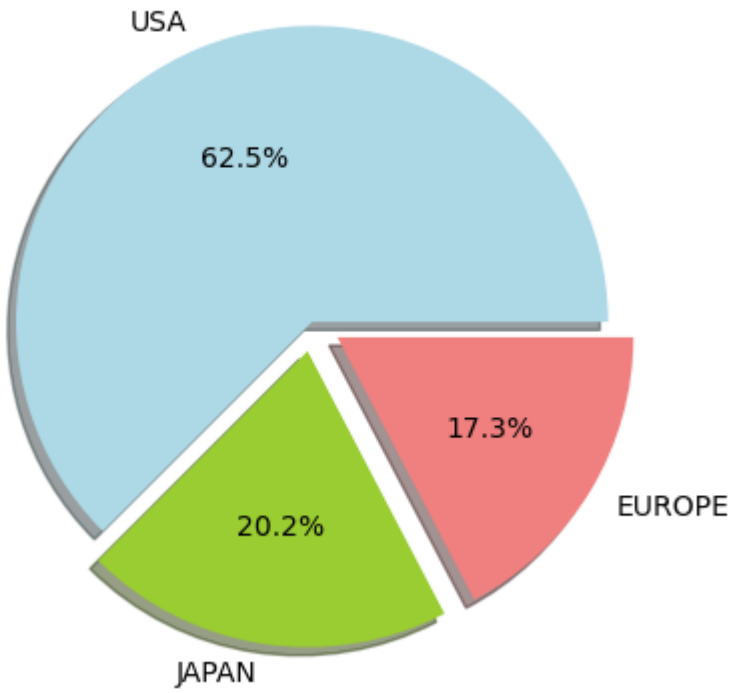
Observation

- The count plot helps us understand the distribution and prevalence of each category in the dataset.

From the count values and percentages, we can observe the relative frequency of each category in the dataset.

Univariate Pie chart

In [38]:
```
# pie chart of origin
# data
lables=['USA','JAPAN','EUROPE']
x=[245,79,68]
colors=['lightblue','yellowgreen','lightcoral','red','lightcoral']
explode=(0,0.1,0.1)

#plot
plt.pie(x,explode=explode,labels=lables,colors=colors,autopct='%1.1f%%',sl
plt.show()
```



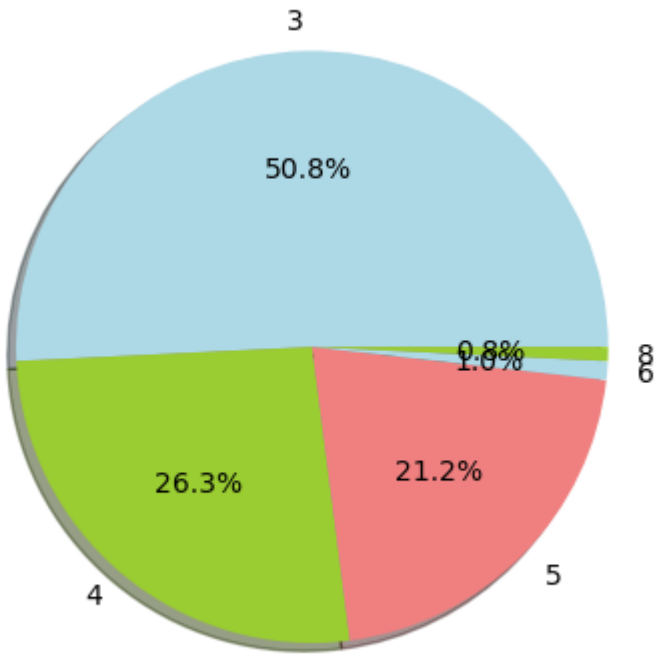
Observation

- A pie chart is the most common way used to visualize the numerical proportion occupied by each of the categories.

in this pie chart we can observe number of vehicles origin in which country

In [39]:

```
# pie chart of number of cylinders
# data
lables=['3','4','5','6','8']
x=[199,103,83,4,3]
colors=['lightblue','yellowgreen','lightcoral']
explode=(0,0,0,0,0)

#plot
plt.pie(x,explode=explode,labels=lables,colors=colors,autopct='%1.1f%%',sl
plt.show()
```

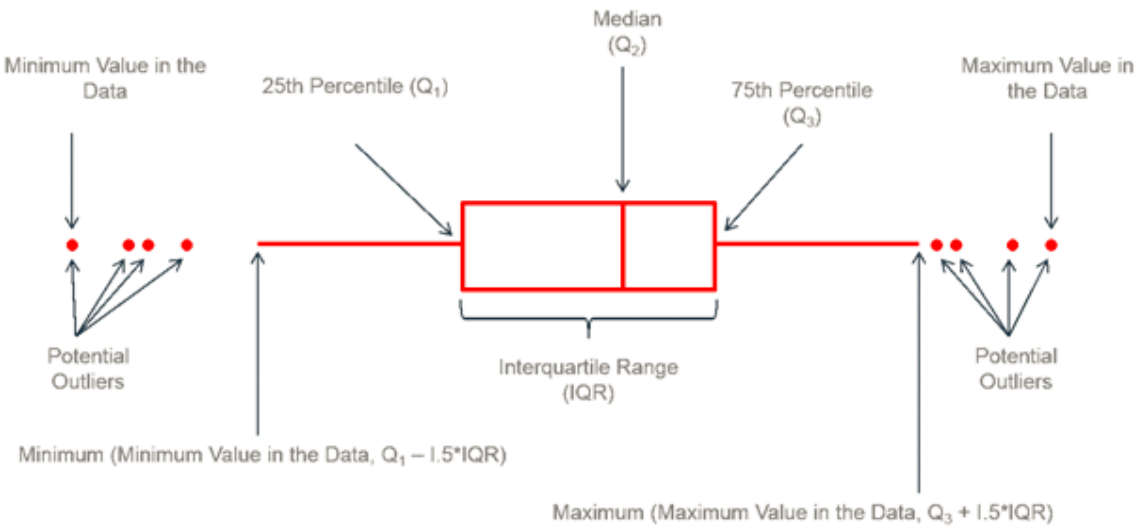


Observation

- Use the plt.pie() function to plot a pie chart.

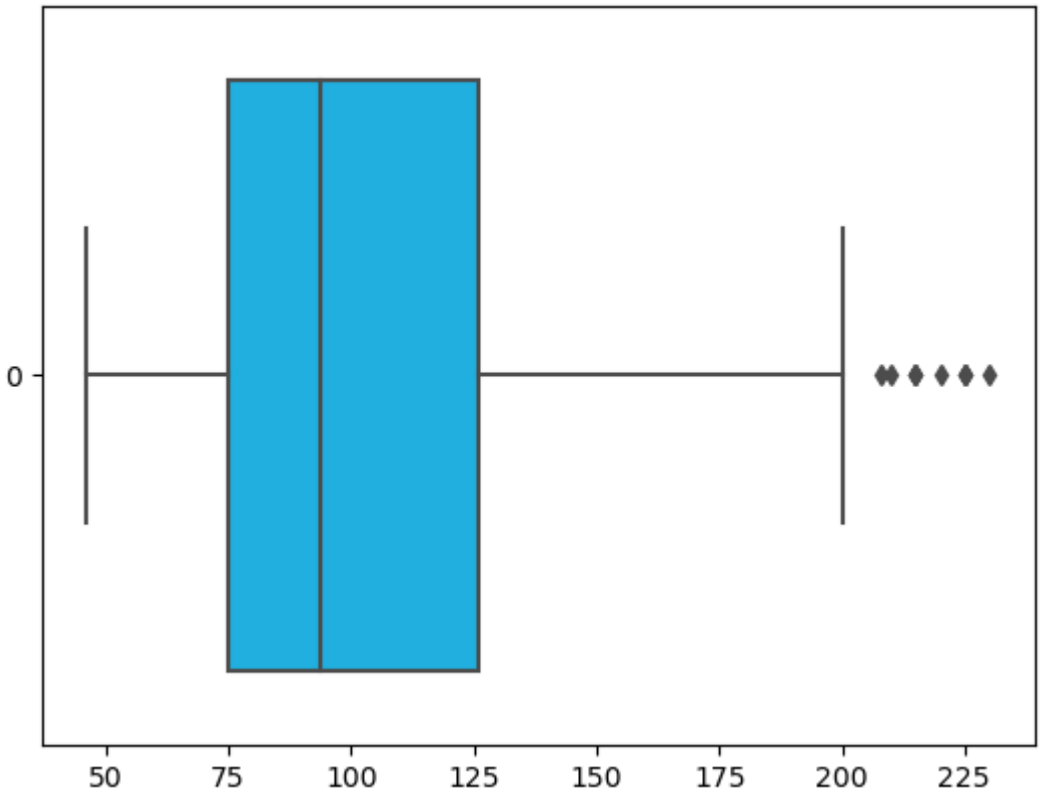
- in this pie chart we can observe number of cylinders in vehicles

Univariate Box Plot



In [40]:
```
1 # Box plot of horsepower
2 sns.boxplot(data['horsepower'],orient='horizantal',color='deepskyblue')
```

Out[40]: <Axes: >



Observation

- The box plot displays the distribution of the 'horsepower' variable in the dataset.

- The box represents the interquartile range (IQR), and the line inside the box represents the median value of the data

The whiskers extend from the box to the minimum and maximum non-outlier data points.

Any data points outside the whiskers are considered potential outliers and plotted as individual

Handling the outliers using flooring and capping method

In [41]:
```
# flooring and capping method

outlier_list = []

# sorting data
data['horsepower'] = sorted(data['horsepower'])

# Finding first and third qurtile
q1 = np.percentile(data['horsepower'], 25)
q3 = np.percentile(data['horsepower'], 75)
print("The Val of Q1 and Q2",q1, q3)

IQR = q3-q1

lwr_bound = q1-(1.5*IQR)
upr_bound = q3+(1.5*IQR)

    # print("The lower & Upper Bound",lwr_bound, upr_bound)

for i in data['horsepower']:
    if (i<lwr_bound or i>upr_bound):
        data['horsepower'] = data['horsepower'].replace(i,np.median(data[
```

```
The Val of Q1 and Q2 75.0 126.0
```

Observation

- The provided code is implementing the flooring and capping method to handle outliers in the 'horsepower' column of the data.

- The flooring and capping method is a simple approach to handle outliers by replacing them with the median value

In [42]:
```
# verifying outlyer

# Initialize a flag to check if there are any outliers
outlier_present = False

for i in data['horsepower']:
    if (i<lwr_bound or i>upr_bound):
        print(i)
        outlier_present = True

# If the flag remains False, no outliers were found
if not outlier_present:
    print("There are no outliers.")
```

```
There are no outliers.
```

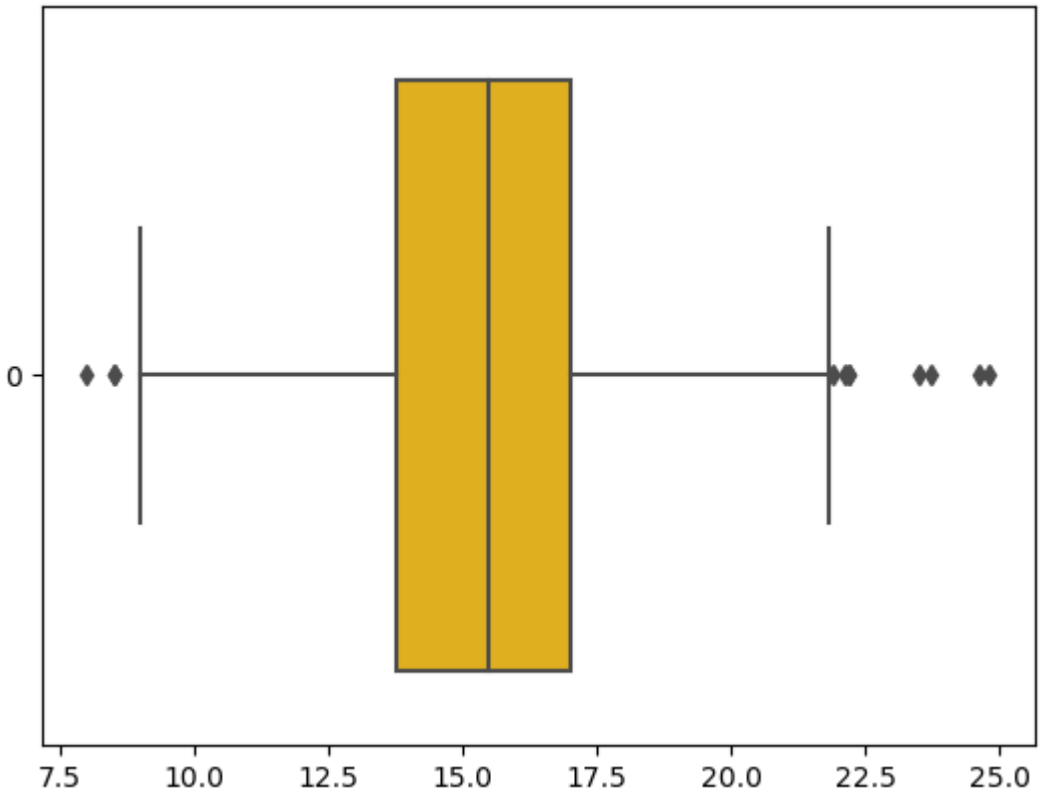
Observation

- The provided code is verifying the presence of outliers in the 'horsepower' column of the data using the lower bound (lwr_bound) and upper bound (upr_bound) values calculated previously with the flooring and capping method.

- If it finds any outliers, it prints those outlier values. If no outliers are found, it informs that there are no outliers in the 'horsepower' column.

In [43]:
```
# box plot of accelaration
sns.boxplot(data['acceleration'],orient='horizantal',color='#ffbf00')
```

Out[43]: <Axes: >



Observation

- TThe provided code generates a horizontal box plot for the 'acceleration' variable using Seaborn's boxplot function.

Handling the outliers usingTrimming and Capping method

In [44]:
```
# Handling the outliers
acc = data['acceleration']

tenth_percentile = np.percentile(acc, 10)
ninetieth_percentile = np.percentile(acc, 90)

print('10th,tenth_percentile',tenth_percentile)
print('90th,tenth_percentile',ninetieth_percentile)

b = np.where(acc<tenth_percentile, tenth_percentile, acc)

b1 = np.where(b>ninetieth_percentile, ninetieth_percentile, b)

data['acceleration'] = b1
```

```
10th,tenth_percentile 12.0
90th,tenth_percentile 19.0
```

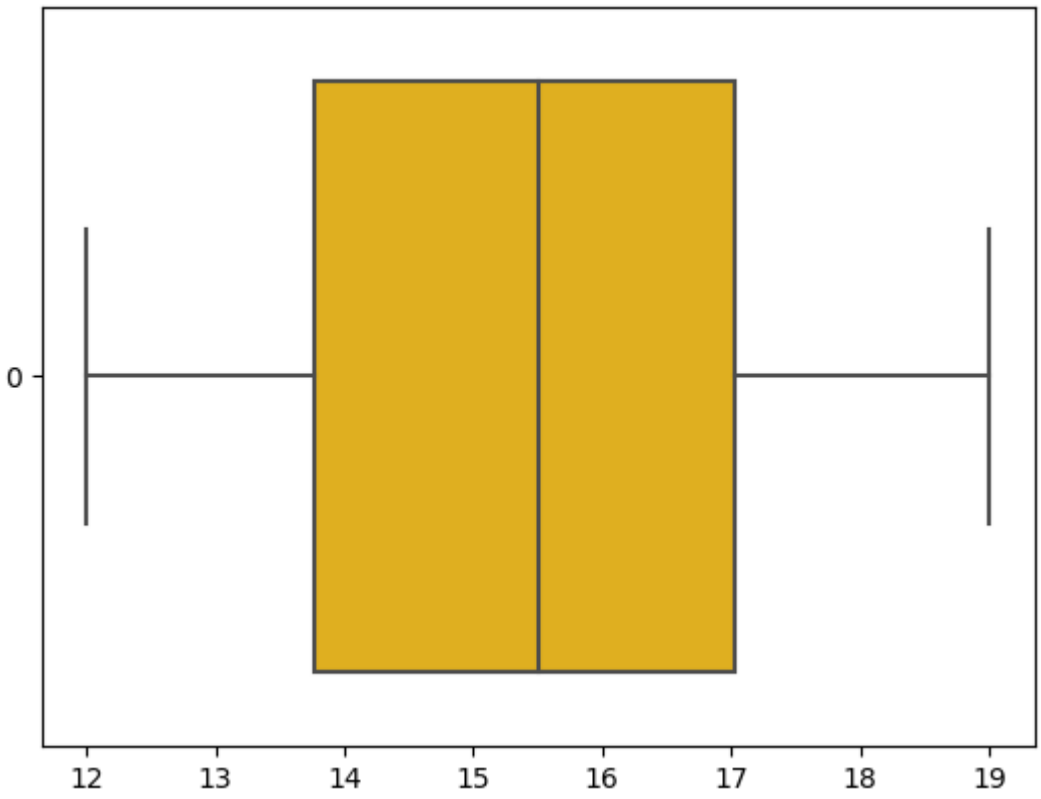
Observation

- The provided code is handling the outliers in the 'acceleration' column of the data using a capping method

By capping the outliers within the 10th and 90th percentiles, the code ensures that extreme values are brought closer to the rest of the data

In [45]:
```
# verifying outlyers
sns.boxplot(data['acceleration'],orient='horizantal',color='#ffbf00')
```

Out[45]: <Axes: >



Observation

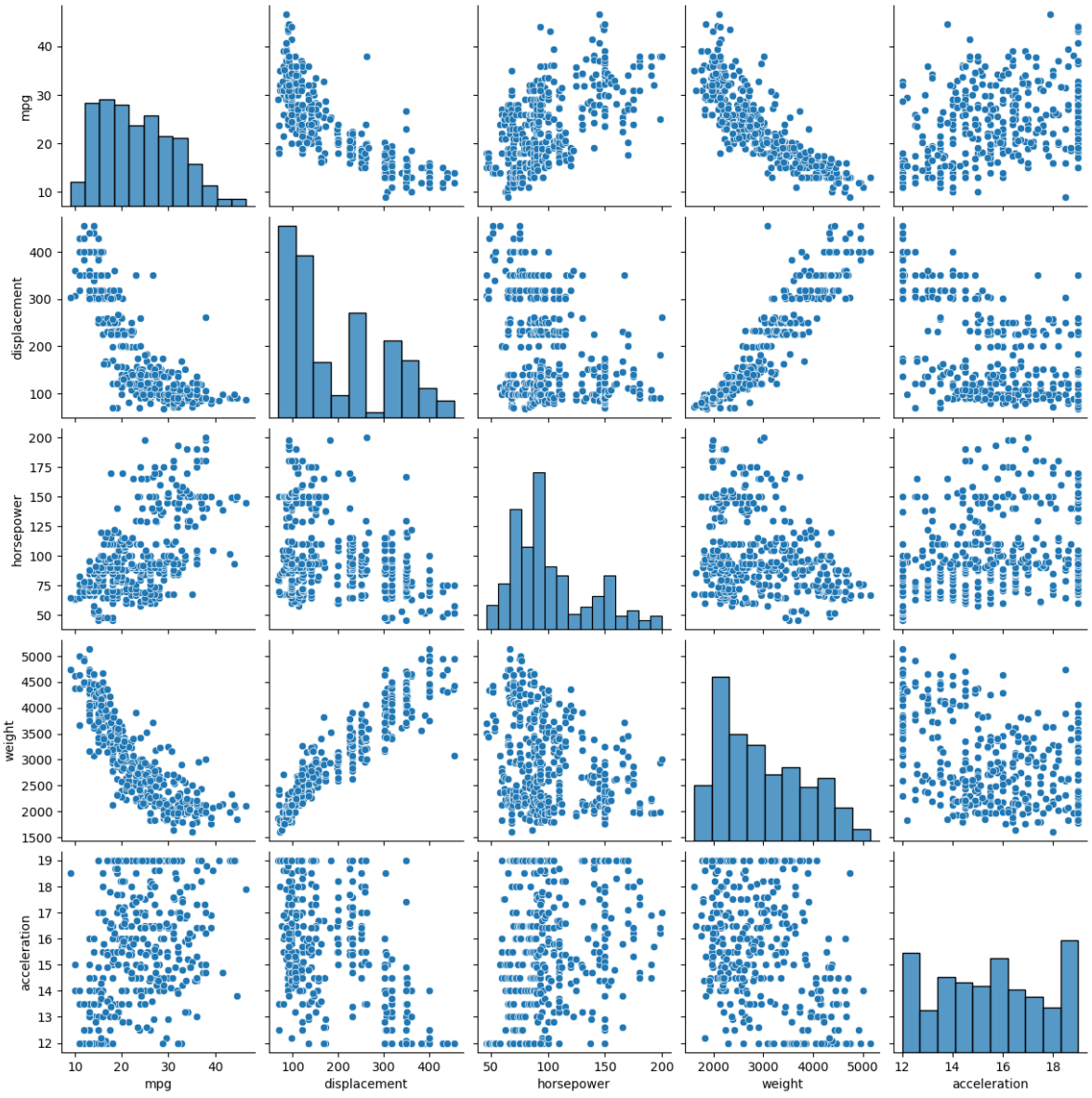
- after performing caping method, to verify it works we again run the code to perform box plot

---

Multivariate data visualization

Scatter Plot Matrix (Pair Plot):

In [46]:
```
# pairplot of given dataset
sns.pairplot(data)
plt.show()
```



Observation

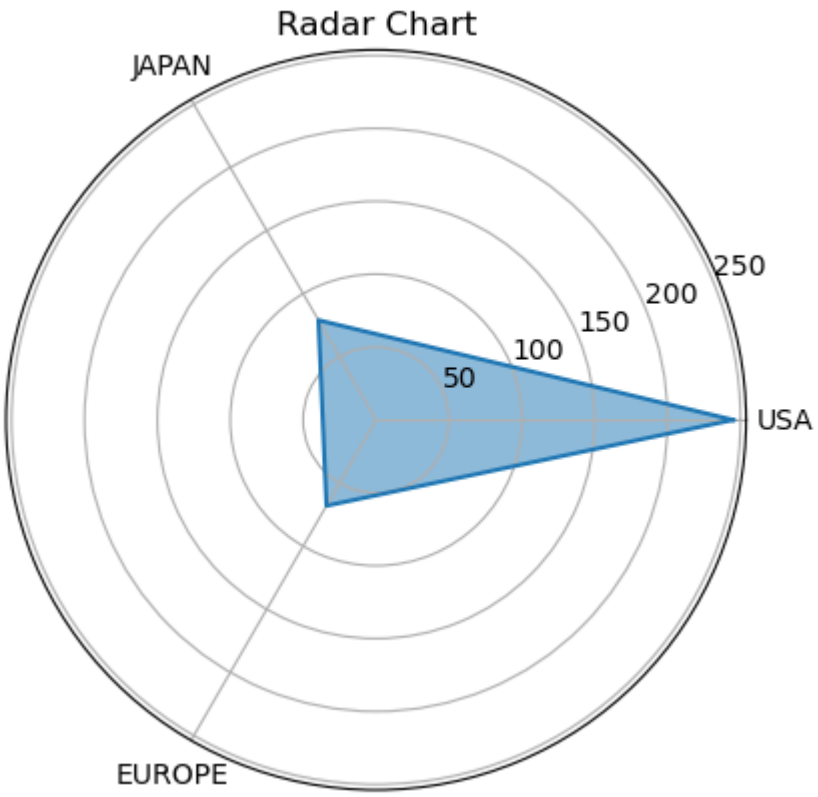
- pair plot shows plots for all atributs

Type *Markdown* and LaTeX: α^2

Radar Chart (Spider Chart):

In [47]:

```
# Data for the radar chart
categories = ['USA', 'JAPAN', 'EUROPE']
values = [245, 79, 68]

# Duplicate the first value to complete the circle (closed shape)
values += values[:1]

# Calculate the angles for each category to plot on the polar axes
angles = [n / float(len(categories)) * 2 * np.pi for n in range(len(categ
angles += angles[:1]

# Create the polar plot
plt.polar(angles, values)

# Fill the area inside the plot with color to create a filled radar chart
plt.fill(angles, values, alpha=0.5)

# Set the category names as labels on the x-axis
plt.xticks(angles[:-1], categories)

# Add a title to the radar chart
plt.title('Radar Chart')

# Display the radar chart
plt.show()
```



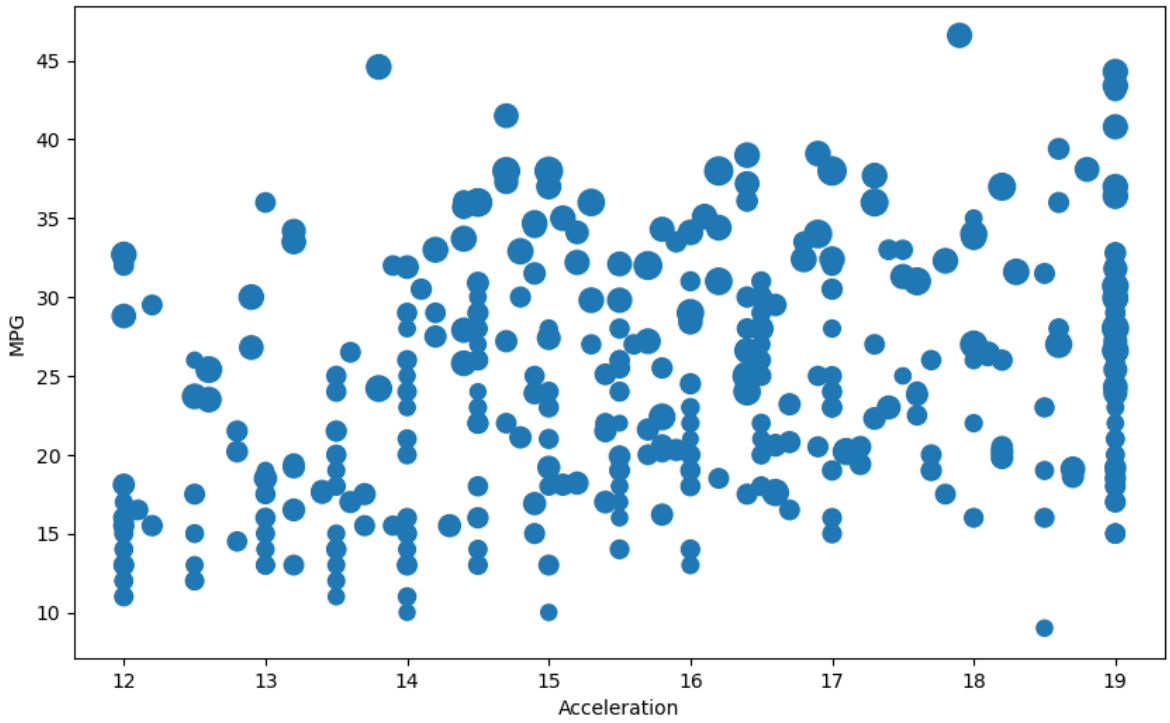
Observation

The resulting radar chart will display the values for each category, forming a closed shape with lines connecting the points for each category.

Bubble Chart:

In [48]:
```
# Extract data for x-axis, y-axis, and sizes
x = data['acceleration']
y = data['mpg']
sizes = data['horsepower']

# Create the bubble plot with different sizes for each point
plt.figure(figsize=(10, 6))
plt.scatter(x, y, s=sizes)

# Set labels for x-axis and y-axis
plt.xlabel('Acceleration')
plt.ylabel('MPG')

```

Out[48]:
```
Text(0, 0.5, 'MPG')
```



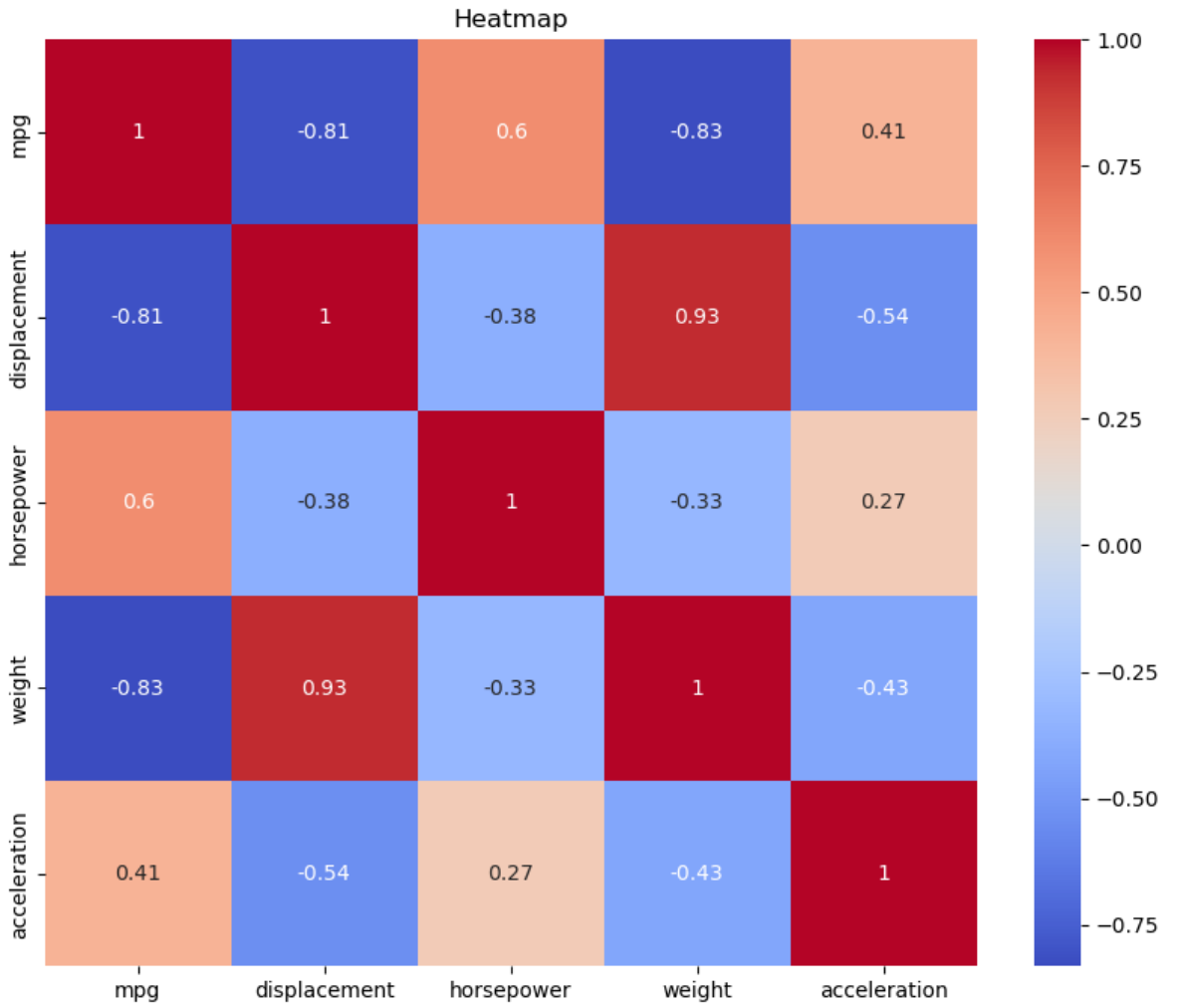
Observation

The bubble plot is a variant of a scatter plot where the size of each data point (bubble) is determined by the 'horsepower' values from the dataset.

Heatmap

In [49]:

```
# Generate the correlation matrix and create a heatmap and red represents
plt.figure(figsize=(10, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm')

# Set the title for the heatmap
plt.title('Heatmap')

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



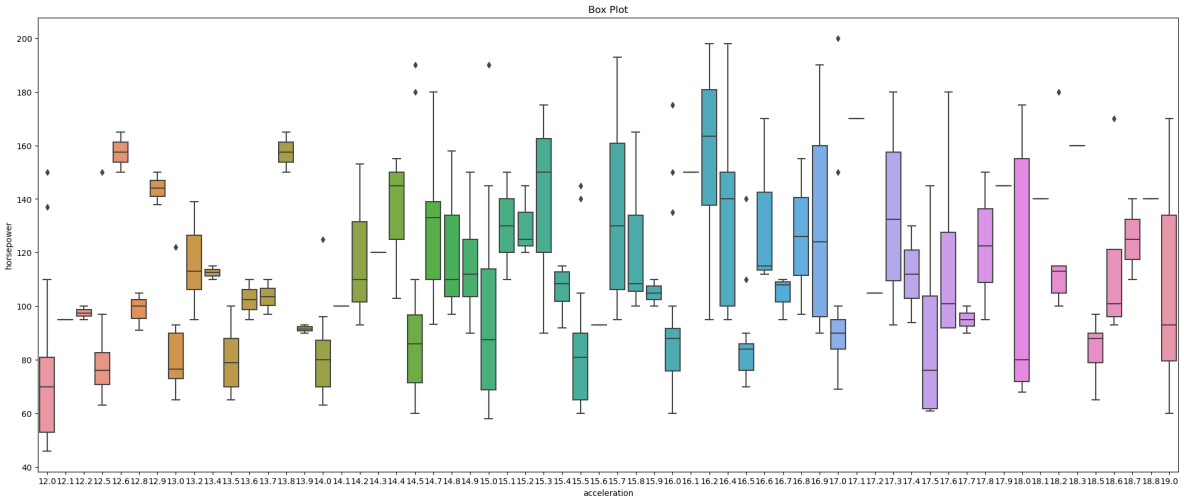
Observation

The heatmap is a graphical representation of the correlation matrix of the dataset. It uses colors to visualize the correlation coefficients between pairs of numeric variables. 'coolwarm' colormap is used, where blue represents negative correlation, and red represents positive correlation

Box Plot:

In [50]:

```
# Create a box plot to visualize the distribution of 'horsepower' for each
# The 'x' axis represents the 'acceleration' values, and the 'y' axis repr
plt.figure(figsize=(25, 10))
sns.boxplot(x='acceleration', y='horsepower', data=data)

# Set the title for the box plot
plt.title('Box Plot')

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



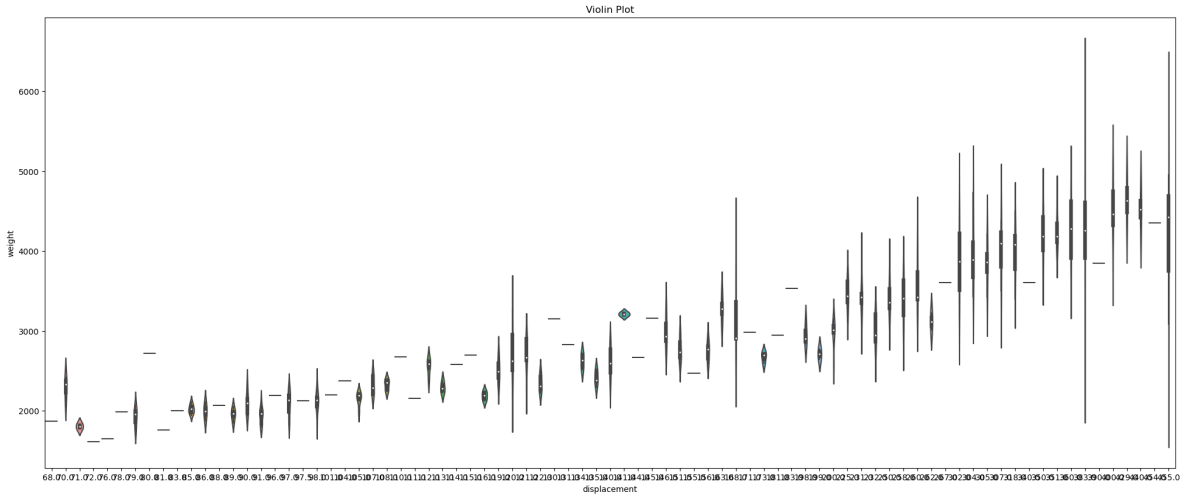
Observation

The box plot provides a visual representation of the distribution of 'horsepower' for each 'acceleration' category.The box plot allows us to quickly assess the central tendency, variability, and presence of outliers in the 'horsepower' distribution for each 'acceleration' category.

Violin Plot:

In [51]:

```
# Create a violin plot to visualize the distribution of 'weight' for each
# The 'x' axis represents the 'displacement' values, and the 'y' axis repr
plt.figure(figsize=(25, 10))
sns.violinplot(x='displacement', y='weight', data=data)

# Set the title for the violin plot
plt.title('Violin Plot')

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



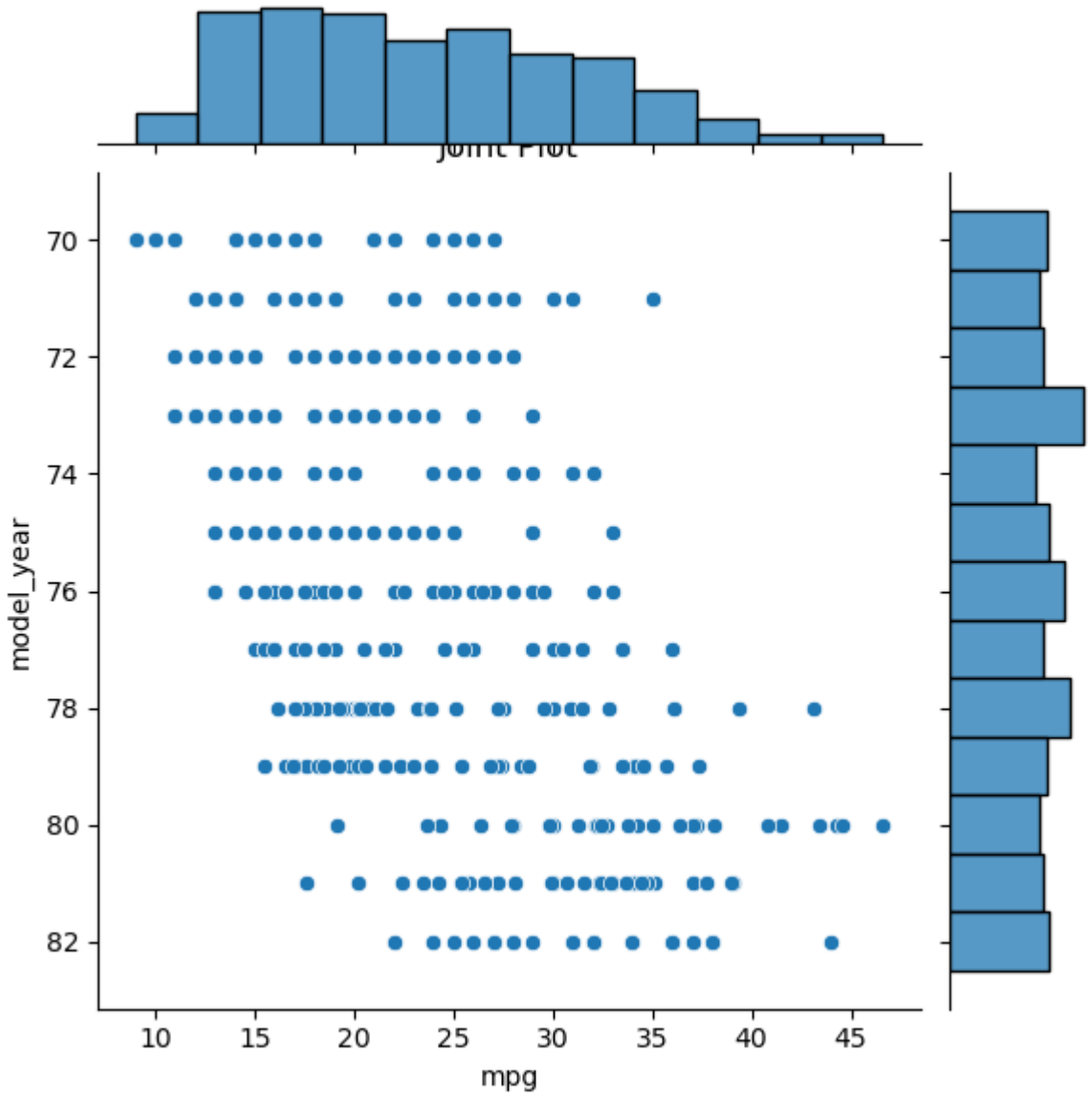
Observation

It provides a more informative representation than a traditional box plot as it includes the kernel density plot, allowing us to observe the shape of the distribution and potential multimodality.

Joint Plot:

In [52]:
```
# Create a joint plot to visualize the relationship between 'mpg' and 'mod
plt.figure(figsize=(10, 8))
sns.jointplot(x='mpg', y='model_year', data=data, kind='scatter')

# Set the title for the joint plot
plt.title('Joint Plot')

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

```
<Figure size 1000x800 with 0 Axes>
```



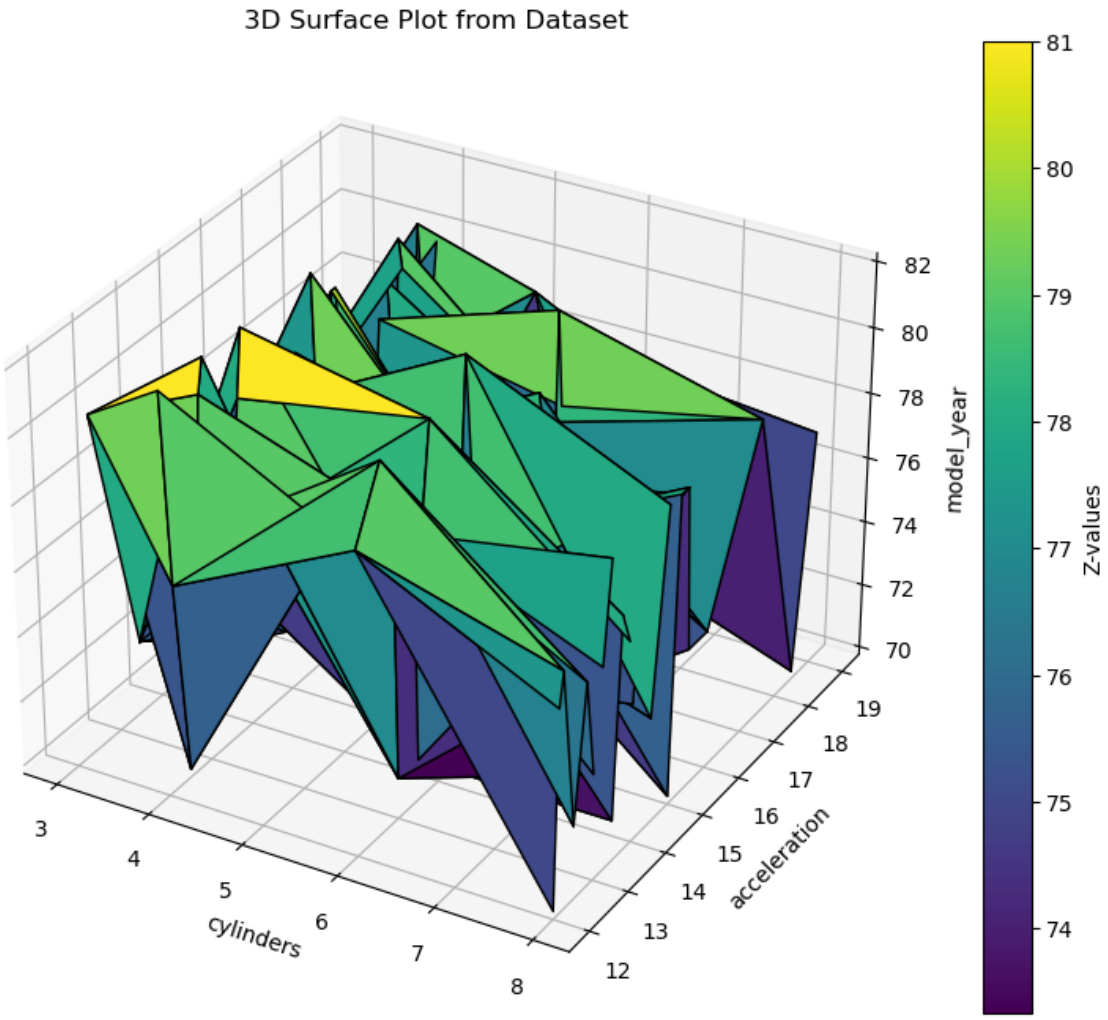
Observation

The 'x' axis represents the 'mpg' values, and the 'y' axis represents the 'model_year' values

3D Surface Plot:

In [53]:

```
# 3D surfact plot

X = data['cylinders']
Y = data['acceleration']
Z = data['model_year']

# Create the 3D surface plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the surface
ax.plot_trisurf(X, Y, Z, cmap='viridis', edgecolor='k')

# Set labels for X, Y, and Z axes
ax.set_xlabel('cylinders')
ax.set_ylabel('acceleration')
ax.set_zlabel('model_year')

# Add a color bar
fig.colorbar(ax.collections[0], label='Z-values')

plt.title('3D Surface Plot from Dataset')
plt.show()

```



Observation

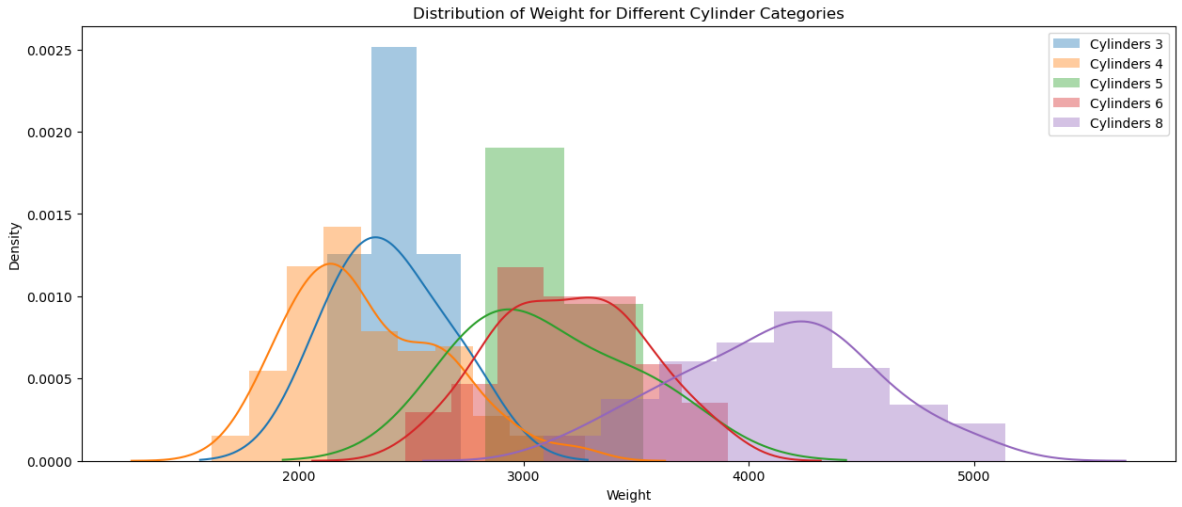
It allows us to explore the relationship between these variables in a three-dimensional space

Type *Markdown* and LaTeX: α^2

Distbution plot according to vehicles cylinder

In [54]:

```
# Create a distribution plot (distplot) for 'weight' based on the 'cylinde
plt.figure(figsize=(15, 6))

# Distplot for cylinders with value 3
sns.distplot(data[data['cylinders'] == 3]['weight'], label='Cylinders 3')

# Distplot for cylinders with value 4
sns.distplot(data[data['cylinders'] == 4]['weight'], label='Cylinders 4')

# Distplot for cylinders with value 5
sns.distplot(data[data['cylinders'] == 5]['weight'], label='Cylinders 5')

# Distplot for cylinders with value 6
sns.distplot(data[data['cylinders'] == 6]['weight'], label='Cylinders 6')

# Distplot for cylinders with value 8
sns.distplot(data[data['cylinders'] == 8]['weight'], label='Cylinders 8')

# Add a legend to identify each 'cylinders' category
plt.legend()

# Set labels for X and Y axes
plt.xlabel('Weight')
plt.ylabel('Density')

# Set the title for the distribution plot
plt.title('Distribution of Weight for Different Cylinder Categories')

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

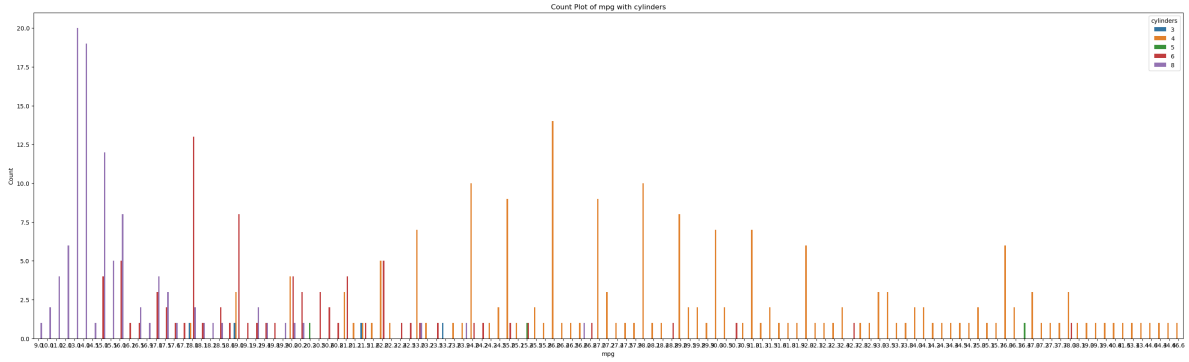


Observation

The distribution plot allows us to understand how values are distributed within each category. It

In [55]:

```
# Create a count plot of 'mpg' with 'cylinders'
# The 'x' axis represents the 'mpg' values, and the count of occurrences
# The 'hue' parameter is set to 'cylinders' to differentiate the count of
plt.figure(figsize=(35, 10))
sns.countplot(x='mpg', data=data, hue='cylinders')

# Set the title for the count plot
plt.title('Count Plot of mpg with cylinders')

# Set labels for X and Y axes
plt.xlabel('mpg')
plt.ylabel('Count')

# Show the count plot
plt.show()

# Crosstab to calculate count for each combination of 'mpg' and 'cylinders
crosstab_result = pd.crosstab(data['cylinders'], data['mpg'])

# Display the count crosstab
print("Count Crosstab:")
print(crosstab_result)
```




```
Count Crosstab:
mpg         9.0   10.0  11.0  12.0  13.0  14.0  14.5  15.0  15.5  16.0  ...  \
cylinders                                                                ...
3              0     0     0     0     0     0     0     0     0     0  ...
4              0     0     0     0     0     0     0     0     0     0  ...
5              0     0     0     0     0     0     0     0     0     0  ...
6              0     0     0     0     0     0     0     4     0     5  ...
8              1     2     4     6    20    19     1    12     5     8  ...

mpg         39.1  39.4  40.8  41.5  43.1  43.4  44.0  44.3  44.6  46.6
cylinders
3              0     0     0     0     0     0     0     0     0     0
4              1     1     1     1     1     1     1     1     1     1
5              0     0     0     0     0     0     0     0     0     0
6              0     0     0     0     0     0     0     0     0     0
8              0     0     0     0     0     0     0     0     0     0

[5 rows x 127 columns]
```

Observation

The count bars are color-coded based on the 'cylinders' categories. Each bar represents the count of cars with a specific number of cylinders for the corresponding 'mpg' value. The count crosstab provides a tabular representation of the counts for each combination of 'mpg' and 'cylinders.'