```
import pandas as pd
```

Loading the **dataset**

```
df=pd.read_csv('/content/archive (1) (1) (1).zip')
df
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...
...	...	...	...	...	...	...	...	...	...	...	...
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	...
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	...
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	...

205 rows × 26 columns

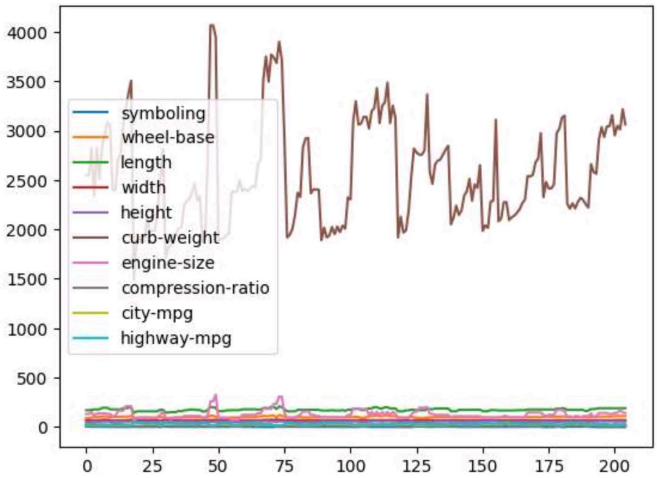
IMPORTING LIBRARIES REQUIRED FOR **VISUALIZATION**

```
import matplotlib.pyplot as plt
import seaborn as sns
```

UNIVARIATE

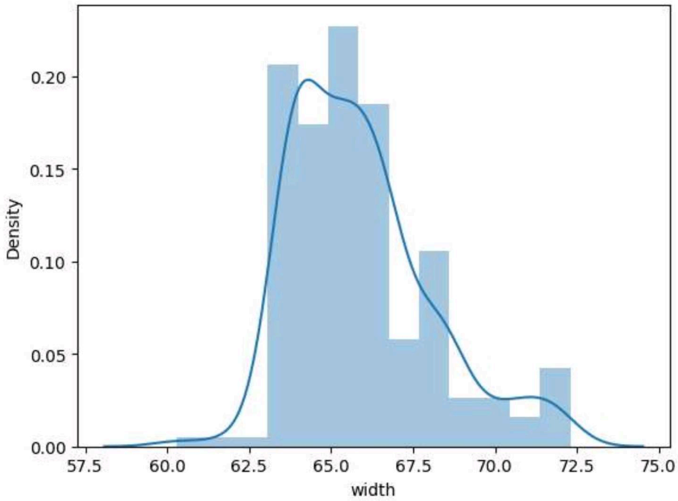
```
df.plot()
```

<Axes: >

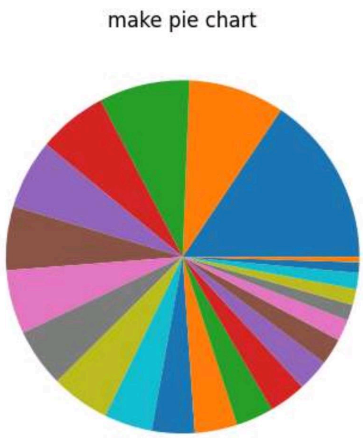


```
sns.distplot(df.width)
```

```
<ipython-input-6-b18b088420f6>:1: UserWarning:

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

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

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

  sns.distplot(df.width)
<Axes: xlabel='width', ylabel='Density'>
```



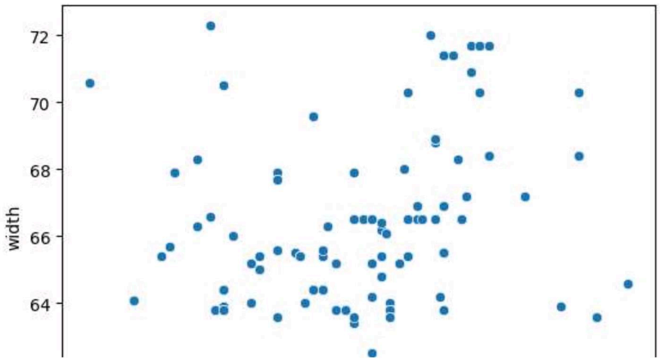

```
plt.pie(df.make.value_counts())
plt.title("make pie chart")
plt.show()
```



BI-VARIATE

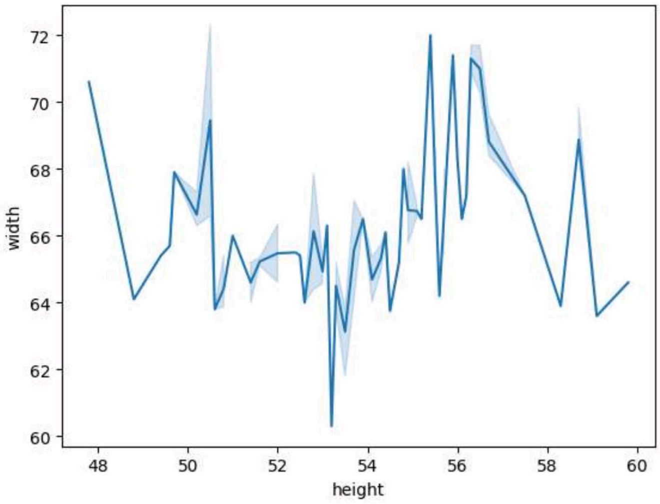
```
sns.scatterplot(x=df.height,y=df.width)
```

<Axes: xlabel='height', ylabel='width'>



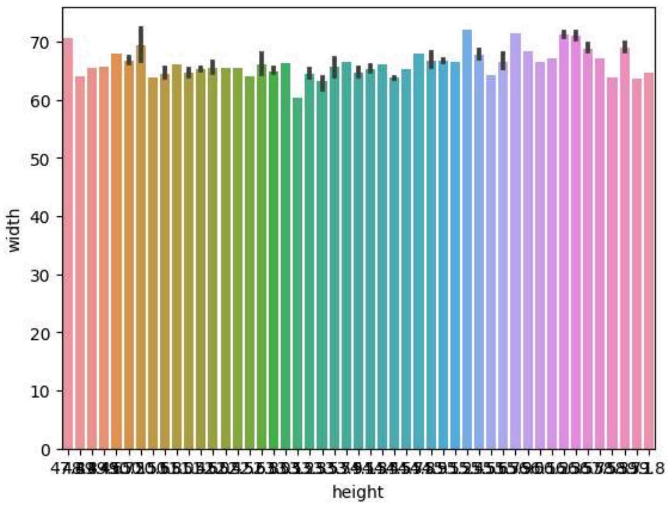

```
sns.lineplot(x=df.height,y=df.width)
```

<Axes: xlabel='height', ylabel='width'>




```
sns.barplot(x=df.height,y=df.width)
```

<Axes: xlabel='height', ylabel='width'>



MULTIVARIATE

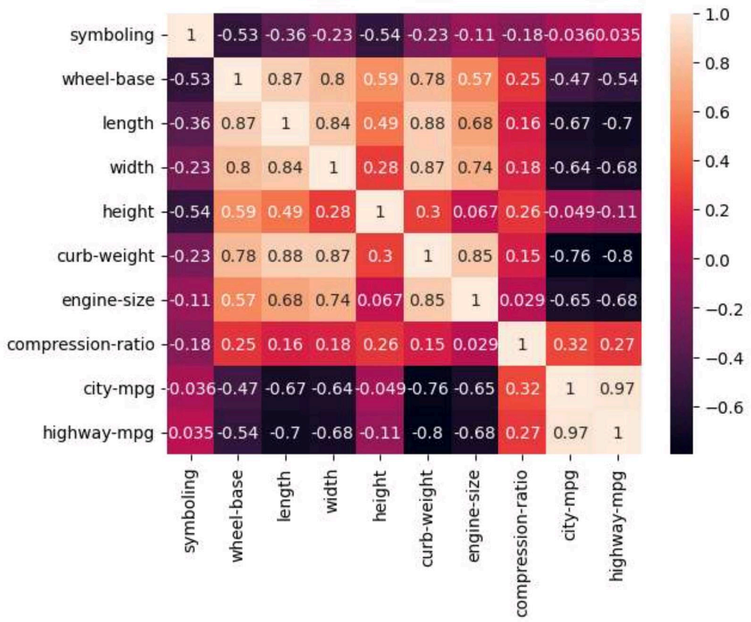
```
df.corr()
```

```
<ipython-input-11-2f6f6606aa2c>:1: FutureWarning: The default value of numeric_only in DataFrame.corr :
  df.corr()
```

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	cit
symboling	1.000000	-0.531954	-0.357612	-0.232919	-0.541038	-0.227691	-0.105790	-0.178515	-0.0
wheel-base	-0.531954	1.000000	0.874587	0.795144	0.589435	0.776386	0.569329	0.249786	-0.4
length	-0.357612	0.874587	1.000000	0.841118	0.491029	0.877728	0.683360	0.158414	-0.6
width	-0.232919	0.795144	0.841118	1.000000	0.279210	0.867032	0.735433	0.181129	-0.6
height	-0.541038	0.589435	0.491029	0.279210	1.000000	0.295572	0.067149	0.261214	-0.0
curb-weight	-0.227691	0.776386	0.877728	0.867032	0.295572	1.000000	0.850594	0.151362	-0.7
engine-size	-0.105790	0.569329	0.683360	0.735433	0.067149	0.850594	1.000000	0.028971	-0.6
compression-ratio	-0.178515	0.249786	0.158414	0.181129	0.261214	0.151362	0.028971	1.000000	0.3
city-mpg	-0.035823	-0.470414	-0.670909	-0.642704	-0.048640	-0.757414	-0.653658	0.324701	1.0
highway-mpg	0.034606	-0.544082	-0.704662	-0.677218	-0.107358	-0.797465	-0.677470	0.265201	0.9

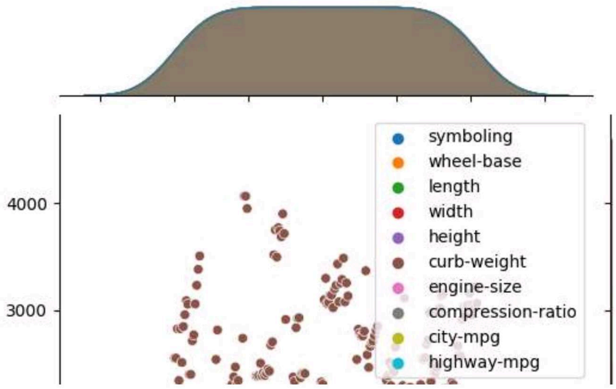
```
sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-12-8df7bcac526d>:1: FutureWarning: The default value of numeric_only in DataFrame.corr :
  sns.heatmap(df.corr(),annot=True)
<Axes: >
```




```
sns.jointplot(df)
```

```
<seaborn.axisgrid.JointGrid at 0x7ff06568f790>
```



4. Perform Data Preprocessing Handling missing values

```
df.isnull().any
```

```
<bound method NDFrame._add_numeric_operations.<locals>.any of      symboling  normalized-losses   make  fuel-type  aspiration  num-of-
doors  \
0        False              False  False      False       False        False
1        False              False  False      False       False        False
2        False              False  False      False       False        False
3        False              False  False      False       False        False
4        False              False  False      False       False        False
..         ...                ...    ...        ...         ...          ...
200      False              False  False      False       False        False
201      False              False  False      False       False        False
202      False              False  False      False       False        False
203      False              False  False      False       False        False
204      False              False  False      False       False        False

     body-style  drive-wheels  engine-location  wheel-base  ...  engine-size  \
0         False         False            False       False  ...        False
1         False         False            False       False  ...        False
2         False         False            False       False  ...        False
3         False         False            False       False  ...        False
4         False         False            False       False  ...        False
..          ...           ...              ...         ...  ...          ...
200       False         False            False       False  ...        False
201       False         False            False       False  ...        False
202       False         False            False       False  ...        False
203       False         False            False       False  ...        False
204       False         False            False       False  ...        False

     fuel-system   bore  stroke  compression-ratio  horsepower  peak-rpm  \
0          False  False   False              False       False     False
1          False  False   False              False       False     False
2          False  False   False              False       False     False
3          False  False   False              False       False     False
4          False  False   False              False       False     False
..           ...    ...     ...                ...         ...       ...
200        False  False   False              False       False     False
201        False  False   False              False       False     False
202        False  False   False              False       False     False
203        False  False   False              False       False     False
204        False  False   False              False       False     False

     city-mpg  highway-mpg  price
0       False        False  False
1       False        False  False
2       False        False  False
3       False        False  False
4       False        False  False
..        ...          ...    ...
200     False        False  False
201     False        False  False
202     False        False  False
203     False        False  False
204     False        False  False

[205 rows x 26 columns]>
```

```
df.isnull().sum()
```

```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
aspiration           0
num-of-doors         0
body-style           0
drive-wheels         0
engine-location      0
wheel-base           0
length               0
width                0
height               0
curb-weight          0
engine-type          0
num-of-cylinders     0
engine-size          0
fuel-system          0
bore                 0
stroke               0
compression-ratio    0
horsepower           0
peak-rpm             0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

Descriptive statistics.

```
df.describe()
```

	symboling	aspiration	wheel-base	length	width	height	curb-weight
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.00000(
mean	0.834146	0.180488	98.756585	174.049268	65.907805	53.724878	2555.56585‹
std	1.245307	0.385535	6.021776	12.337289	2.145204	2.443522	520.68020‹
min	-2.000000	0.000000	86.600000	141.100000	60.300000	47.800000	1488.00000(
25%	0.000000	0.000000	94.500000	166.300000	64.100000	52.000000	2145.00000(
50%	1.000000	0.000000	97.000000	173.200000	65.500000	54.100000	2414.00000(
75%	2.000000	0.000000	102.400000	183.100000	66.900000	55.500000	2935.00000(
max	3.000000	1.000000	120.900000	208.100000	72.300000	59.800000	4066.00000(

Correlation Check

```
df.make.value_counts()
```

```
toyota           32
nissan           18
mazda            17
mitsubishi       13
honda            13
volkswagen       12
subaru           12
peugot           11
volvo            11
dodge             9
mercedes-benz     8
bmw               8
audi              7
plymouth          7
saab              6
porsche           5
isuzu             4
jaguar            3
chevrolet         3
alfa-romero       3
renault           2
mercury           1
Name: make, dtype: int64
```

HANDLING CATEGORICAL VARIABLES(ENCODING)

Label encoding

```
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
```

```
df.aspiration=le.fit_transform(df.aspiration)
```

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	e
0	3	?	alfa-romero	gas	0	two	convertible	rwd	front	88.6	...	
1	3	?	alfa-romero	gas	0	two	convertible	rwd	front	88.6	...	
2	1	?	alfa-romero	gas	0	two	hatchback	rwd	front	94.5	...	
3	2	164	audi	gas	0	four	sedan	fwd	front	99.8	...	
4	2	164	audi	gas	0	four	sedan	4wd	front	99.4	...	

5 rows × 26 columns

ONE HOT ENCODING

```
df_main=pd.get_dummies(df,columns=['drive-wheels'])
df_main.tail()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	engine-location	wheel-base	length	...	strol
200	-1	95	volvo	gas	0	four	sedan	front	109.1	188.8	...	3.
201	-1	95	volvo	gas	1	four	sedan	front	109.1	188.8	...	3.
202	-1	95	volvo	gas	0	four	sedan	front	109.1	188.8	...	2.
203	-1	95	volvo	diesel	1	four	sedan	front	109.1	188.8	...	3
204	-1	95	volvo	gas	1	four	sedan	front	109.1	188.8	...	3.

5 rows × 28 columns

Splitting of x and y

```
X=df_main.drop(columns=['make','num-of-doors','fuel-type','aspiration','body-style','engine-location','fuel-system','num-of-cylinders','engir
                        'engine-size','horsepower','wheel-base','width','height','highway-mpg','city-mpg'],axis=1)
X.head()
```

	symboling	bore	stroke	compression-ratio	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd
0	3	3.47	2.68	9.0	0	0	1

```
y=df_main.length
y
```

```
0      168.8
1      168.8
2      171.2
3      176.6
4      176.6
       ...
200    188.8
201    188.8
202    188.8
203    188.8
204    188.8
Name: length, Length: 205, dtype: float64
```

✓ 0s completed at 8:21 PM

```
import pandas as pd
```

SCALING

```
df=pd.read_csv('/content/archive (1) (1) (1).zip')
```

```
df.dropna(inplace=True)
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
categorical_columns = ['make', 'body-style', 'fuel-type', 'aspiration']
le = LabelEncoder()
```

```
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
```

```
missing_values = ['?', 'NA', 'N/A', 'nan']  # Update with the actual missing value representations
df = df.replace(missing_values, pd.NA)
```

```
from sklearn.impute import SimpleImputer
```

```
numerical_columns = ['normalized-losses', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-size', 'bore', 'stroke', 'compres
df[numerical_columns] = df[numerical_columns].apply(pd.to_numeric)
imputer = SimpleImputer(strategy='mean')
df[numerical_columns] = imputer.fit_transform(df[numerical_columns])
```

```
categorical_columns = ['make', 'body-style', 'fuel-type', 'aspiration']
le = LabelEncoder()
for col in categorical_columns:
    df[col] = le.fit_transform(df[col])
```

```
scaler = StandardScaler()
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])
```

```
print(df.head())
```

```
   symboling  normalized-losses  make  fuel-type  aspiration num-of-doors  \
0          3           0.000000     0          1           0          two
1          3           0.000000     0          1           0          two
2          1           0.000000     0          1           0          two
3          2           1.328961     1          1           0         four
4          2           1.328961     1          1           0         four

   body-style drive-wheels engine-location  wheel-base  ...  engine-size  \
0           0          rwd           front   -1.690772  ...     0.074449
1           0          rwd           front   -1.690772  ...     0.074449
2           2          rwd           front   -0.708596  ...     0.604046
3           3          fwd           front    0.173698  ...    -0.431076
4           3          4wd           front    0.107110  ...     0.218885

   fuel-system      bore    stroke compression-ratio horsepower  peak-rpm  \
0         mpfi  0.519089 -1.839404         -0.288349   0.171065 -0.263484
1         mpfi  0.519089 -1.839404         -0.288349   0.171065 -0.263484
2         mpfi -2.404862  0.685920         -0.288349   1.261807 -0.263484
3         mpfi -0.517248  0.462157         -0.035973  -0.057230  0.787346
4         mpfi -0.517248  0.462157         -0.540725   0.272529  0.787346

   city-mpg  highway-mpg     price
0 -0.646553    -0.546059  0.036674
1 -0.646553    -0.546059  0.419498
2 -0.953012    -0.691627  0.419498
3 -0.186865    -0.109354  0.094639
4 -1.106241    -1.273900  0.540524

[5 rows x 26 columns]
```

BUILDING AND EVALUATING MODEL

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer


# Drop any rows with missing values
df = df.dropna()

# Split the dataset into features (X) and target variable (y)
X = df.drop('price', axis=1)
y = df['price']

# Perform scaling on numerical columns
numerical_columns = X.select_dtypes(include=['float64', 'int64']).columns
scaler = StandardScaler()
X[numerical_columns] = scaler.fit_transform(X[numerical_columns])

# Perform one-hot encoding on categorical columns
categorical_columns = X.select_dtypes(include=['object']).columns
encoder = OneHotEncoder()
ct = ColumnTransformer(transformers=[('encoder', encoder, categorical_columns)], remainder='passthrough')
X = ct.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build the machine learning model (Linear Regression)
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print('Mean Squared Error:', mse)
```

```
Mean Squared Error: 0.27223084303976924
```