In [ ]:
```
## Mutilinear Regrassion
```

In [37]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
```

In [2]:
```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from feature_engine.outliers import Winsorizer
```

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

In [4]:
```
data = pd.read_csv(r"C:\Users\shubham lokare\Downloads\Multiple_Linear Regression\Multiple_Linear Regression\Ca
```

In [5]:
```
data
```

Out[5]:

	MPG	Enginetype	HP	VOL	SP	WT
0	53.700681	petrol	49	89	104.185353	28.762059
1	50.013401	hybrid	55	92	105.461264	30.466833
2	50.013401	diesel	55	92	105.461264	30.193597
3	45.696322	lpg	70	92	113.461264	30.632114
4	50.504232	petrol	53	92	104.461264	29.889149
...	...	...	...	...	...	...
76	36.900000	hybrid	322	50	169.598513	16.132947
77	19.197888	lpg	238	115	150.576579	37.923113
78	34.000000	hybrid	263	50	151.598513	15.769625
79	19.833733	diesel	295	119	167.944460	39.423099
80	12.101263	hybrid	236	107	139.840817	34.948615

81 rows × 6 columns

In [6]:
```
### check how many engine type
data['Enginetype'].unique()
```

Out[6]:
```
array(['petrol', 'hybrid', 'diesel', 'lpg', 'cng'], dtype=object)
```

In [7]:
```
data['Enginetype'].value_counts()
```

Out[7]:
```
diesel    26
petrol    16
hybrid    16
lpg       12
cng       11
Name: Enginetype, dtype: int64
```

In [8]:
```
### check missing values
data.isna().sum()
```

Out[8]:
```
MPG           0
Enginetype    0
HP            0
VOL           0
SP            0
WT            0
dtype: int64
```

In [9]:
```
### split the data into input and output variable
# Seperating input and output variables
X = pd.DataFrame(data.iloc[:, 1:6])
y = pd.DataFrame(data.iloc[:, 0])
```

In [10]:
```
X   ### input variable
```

Out[10]:

	Enginetype	HP	VOL	SP	WT
0	petrol	49	89	104.185353	28.762059
1	hybrid	55	92	105.461264	30.466833
2	diesel	55	92	105.461264	30.193597
3	lpg	70	92	113.461264	30.632114
4	petrol	53	92	104.461264	29.889149
...	...	...	...	...	...
76	hybrid	322	50	169.598513	16.132947
77	lpg	238	115	150.576579	37.923113
78	hybrid	263	50	151.598513	15.769625
79	diesel	295	119	167.944460	39.423099
80	hybrid	236	107	139.840817	34.948615

81 rows × 5 columns

In [11]:
```
y       ### output variable
```

Out[11]:

	MPG
0	53.700681
1	50.013401
2	50.013401
3	45.696322
4	50.504232
...	...
76	36.900000
77	19.197888
78	34.000000
79	19.833733
80	12.101263

81 rows × 1 columns

In [12]:
```
#### seprate the numerical and categorical columns

numerical_feature = X.select_dtypes(exclude=['object']).columns
numerical_feature
```

Out[12]:
```
Index(['HP', 'VOL', 'SP', 'WT'], dtype='object')
```

In [13]:
```
### categorical data
categorical_feature = X.select_dtypes(include=['object']).columns
categorical_feature
```

Out[13]:
```
Index(['Enginetype'], dtype='object')
```

In [14]:
```
#### make pipeline for missing values
numerical = Pipeline([('impute' , SimpleImputer(strategy = 'mean'))])
```

In [15]:
```
### transform into columns
process = ColumnTransformer([('impute' , numerical , numerical_feature)])
```

In [16]:
```
data1 = process.fit(X)
```

In [17]:
```
new_data = pd.DataFrame(data1.transform(X) , columns =numerical_feature)
```

In [18]:
```
new_data
```

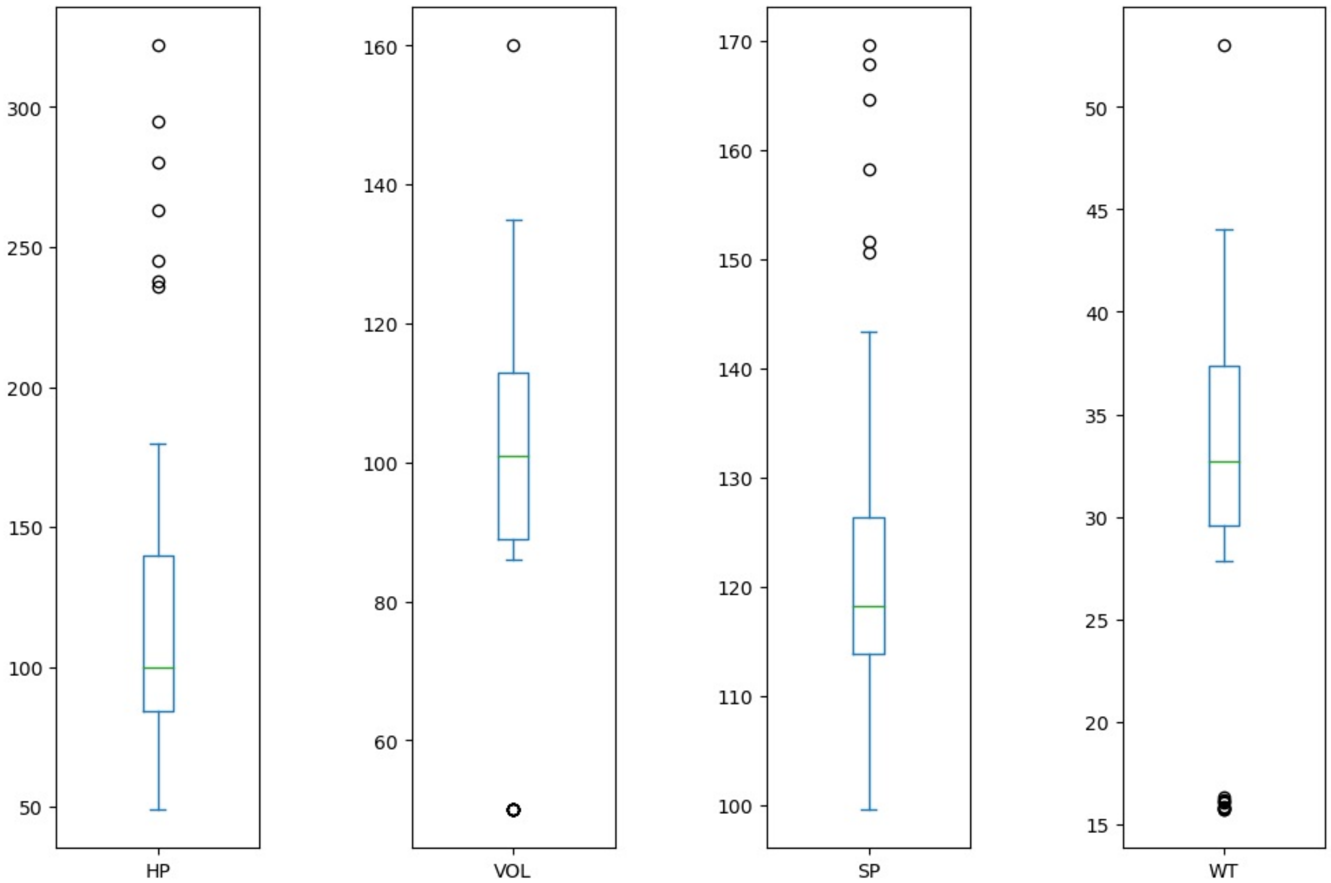
Out[18]:

	HP	VOL	SP	WT
0	49.0	89.0	104.185353	28.762059
1	55.0	92.0	105.461264	30.466833
2	55.0	92.0	105.461264	30.193597
3	70.0	92.0	113.461264	30.632114
4	53.0	92.0	104.461264	29.889149
...	...	...	...	...
76	322.0	50.0	169.598513	16.132947
77	238.0	115.0	150.576579	37.923113
78	263.0	50.0	151.598513	15.769625
79	295.0	119.0	167.944460	39.423099
80	236.0	107.0	139.840817	34.948615

81 rows × 4 columns

In [19]:

```
### check outliers
X.plot(kind= 'box' , subplots = True, figsize=(12,8))
plt.subplots_adjust(wspace= 0.75)
plt.show()
```



In [20]:

```
# Winsorization for outlier treatment
winsor = Winsorizer(capping_method = 'iqr', # choose  IQR rule boundaries or gaussian for mean and std
                          tail = 'both', # cap left, right or both tails
                          fold = 1.5,
                          variables = list(new_data.columns))
```

In [21]:

```
clean = winsor.fit(new_data)
```

In [22]:

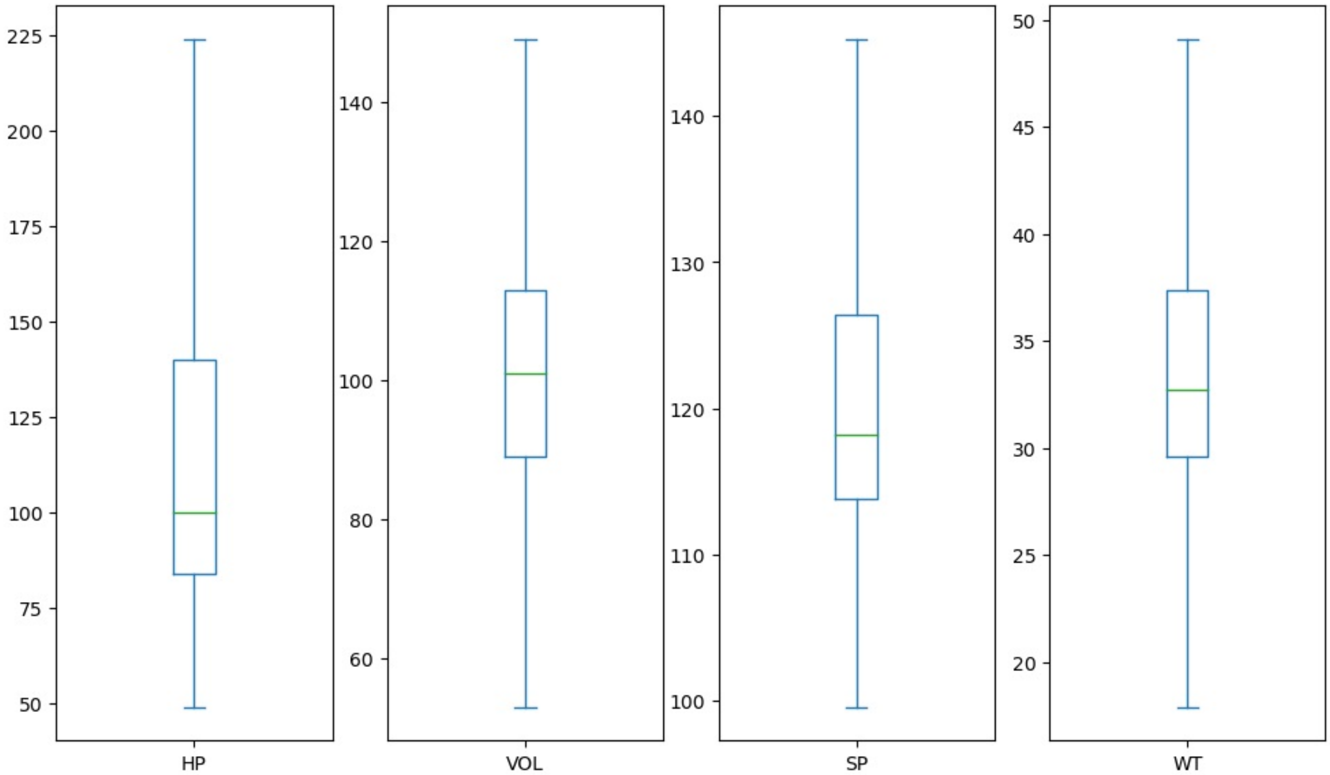
```
new = pd.DataFrame(clean.transform(new_data), columns = numerical_feature)
```

In [23]:

```
### check outliers
new.plot(kind= 'box' , subplots =True , figsize=(12,7))
```

Out[23]:

```
HP        Axes(0.125,0.11;0.168478x0.77)
VOL    Axes(0.327174,0.11;0.168478x0.77)
SP     Axes(0.529348,0.11;0.168478x0.77)
WT     Axes(0.731522,0.11;0.168478x0.77)
dtype: object
```



In [24]:
```
# Scaling
## Scaling with MinMaxScaler
scale = Pipeline([('scale', MinMaxScaler())])
```

In [25]:
```
process3 = ColumnTransformer([('scale', scale, numerical_feature)])
# Skips the transformations for remaining columns
```

In [26]:
```
data2 = process3.fit(new)
```

In [27]:
```
clean_data = pd.DataFrame(data2.transform(new) , columns = numerical_feature)
```

In [28]:
```
clean_data
```

Out[28]:

	HP	VOL	SP	WT
0	0.000000	0.375000	0.101099	0.348409
1	0.034286	0.406250	0.129017	0.403044
2	0.034286	0.406250	0.129017	0.394287
3	0.120000	0.406250	0.304063	0.408341
4	0.022857	0.406250	0.107136	0.384530
...	...	...	...	...
76	1.000000	0.000000	1.000000	0.000000
77	1.000000	0.645833	1.000000	0.642004
78	1.000000	0.000000	1.000000	0.000000
79	1.000000	0.687500	1.000000	0.690076
80	1.000000	0.562500	0.881269	0.546677

81 rows × 4 columns

In [29]:
```
clean_data.describe()
```

Out[29]:

	HP	VOL	SP	WT
count	81.000000	81.000000	81.000000	81.000000
mean	0.369312	0.478781	0.456253	0.470733
std	0.266050	0.220992	0.244018	0.221019
min	0.000000	0.000000	0.000000	0.000000
25%	0.200000	0.375000	0.312113	0.375000
50%	0.291429	0.500000	0.407941	0.475719
75%	0.520000	0.625000	0.587268	0.625000
max	1.000000	1.000000	1.000000	1.000000

In [30]:

```
## Encoding
# Categorical features
encoding_pipeline = Pipeline([('onehot', OneHotEncoder())])
```

In [31]:

```
preprocess_pipeline = ColumnTransformer([('categorical', encoding_pipeline, categorical_feature)])

clean =  preprocess_pipeline.fit(X)   # Works with categorical features only
```

In [34]:

```
encode_data = pd.DataFrame(clean.transform(X).todense())


# To get feature names for Categorical columns after Onehotencoding
encode_data.columns = clean.get_feature_names_out(input_features = X.columns)
encode_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 5 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   categorical__Enginetype_cng     81 non-null     float64
 1   categorical__Enginetype_diesel  81 non-null     float64
 2   categorical__Enginetype_hybrid  81 non-null     float64
 3   categorical__Enginetype_lpg     81 non-null     float64
 4   categorical__Enginetype_petrol  81 non-null     float64
dtypes: float64(5)
memory usage: 3.3 KB
```

In [35]:

```
cleandata = pd.concat([clean_data, encode_data], axis = 1)
# concatenated data will have new sequential index
cleandata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 9 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   HP                              81 non-null     float64
 1   VOL                             81 non-null     float64
 2   SP                              81 non-null     float64
 3   WT                              81 non-null     float64
 4   categorical__Enginetype_cng     81 non-null     float64
 5   categorical__Enginetype_diesel  81 non-null     float64
 6   categorical__Enginetype_hybrid  81 non-null     float64
 7   categorical__Enginetype_lpg     81 non-null     float64
 8   categorical__Enginetype_petrol  81 non-null     float64
dtypes: float64(9)
memory usage: 5.8 KB
```

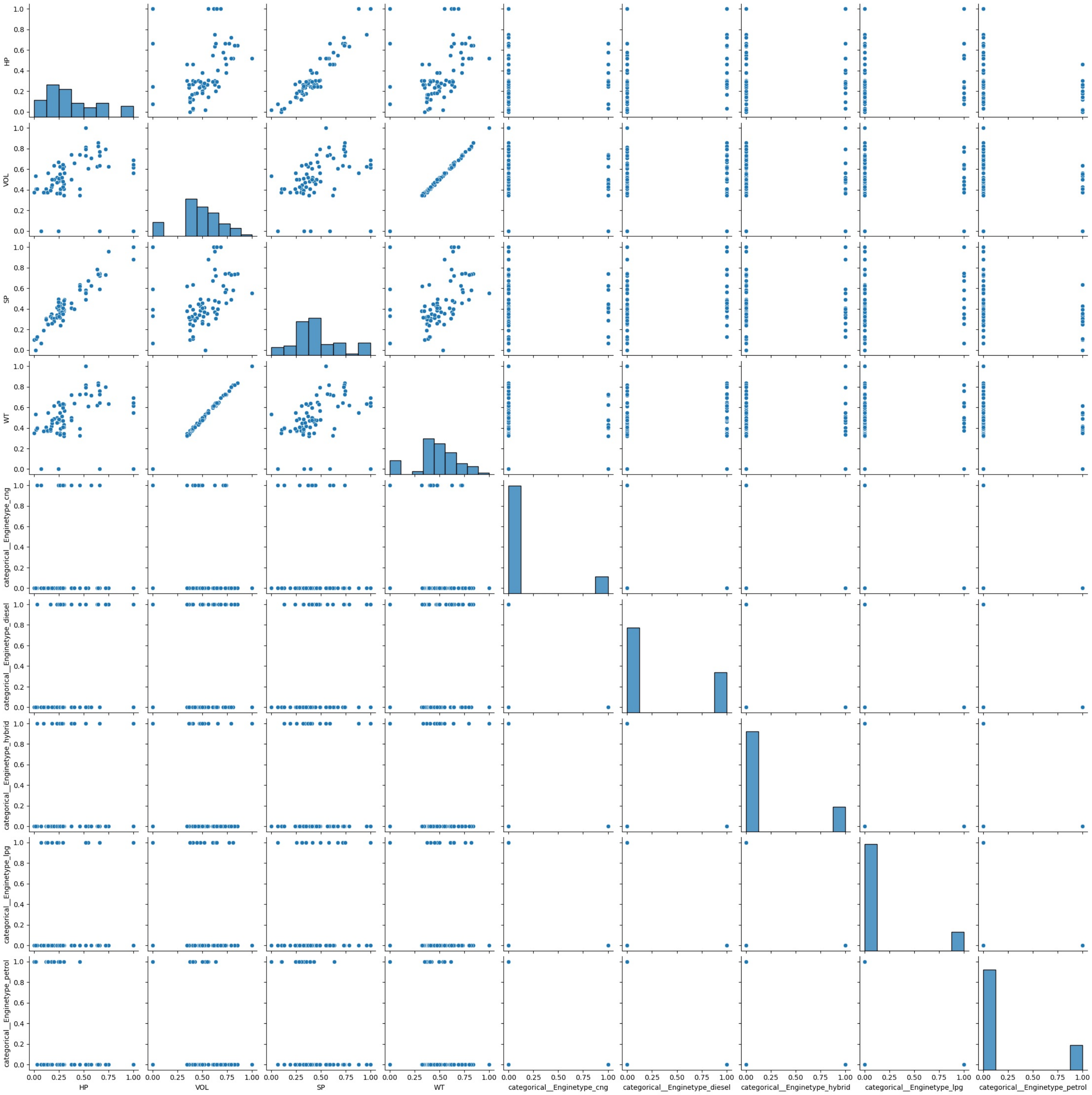
In [38]:

```
### plot pairplot
sns.pairplot(cleandata)
```

```
C:\Users\shubham lokare\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x1fb6210e490>
```



In [40]:
```
### # Library to call OLS model
# import statsmodels.api as sm

# Build a vanilla model on full dataset
from sklearn.model_selection import train_test_split
# import statsmodels.formula.api as smf
import statsmodels.api as sm

from sklearn.linear_model import LinearRegression
```

In [45]:
```
X_train ,X_test , Y_train ,Y_test =train_test_split(cleandata ,y , test_size = 0.2 , random_state = 0)
```

In [48]:
```
## Build the best model Model building with OLS
model = sm.OLS(Y_train, X_train).fit()
model.summary()
```

Out[48]: OLS Regression Results

Dep. Variable:	MPG	**R-squared:**	0.835
Model:	OLS	**Adj. R-squared:**	0.811
Method:	Least Squares	**F-statistic:**	34.78
Date:	Sun, 09 Jun 2024	**Prob (F-statistic):**	7.76e-19
Time:	12:36:00	**Log-Likelihood:**	-174.70
No. Observations:	64	**AIC:**	367.4
Df Residuals:	55	**BIC:**	386.8
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
HP	-37.5602	8.292	-4.530	0.000	-54.177	-20.944
VOL	-25.3553	56.330	-0.450	0.654	-138.244	87.533
SP	14.6843	9.005	1.631	0.109	-3.362	32.731
WT	10.6664	56.722	0.188	0.852	-103.007	124.340
categorical__Enginetype_cng	47.9789	2.096	22.893	0.000	43.779	52.179
categorical__Enginetype_diesel	47.8560	2.082	22.986	0.000	43.684	52.028
categorical__Enginetype_hybrid	50.2603	1.882	26.701	0.000	46.488	54.033
categorical__Enginetype_lpg	48.6177	2.185	22.249	0.000	44.238	52.997
categorical__Enginetype_petrol	49.9100	1.893	26.369	0.000	46.117	53.703

Omnibus:	1.100	**Durbin-Watson:**	2.001
Prob(Omnibus):	0.577	**Jarque-Bera (JB):**	0.878
Skew:	0.286	**Prob(JB):**	0.645
Kurtosis:	2.952	**Cond. No.**	166.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [49]:
```
pred = model.predict(X_test)
```

In [50]:
```
pred
```

Out[50]:
```
22    37.018285
27    37.987385
61    31.025396
13    42.911360
71    21.751389
74    26.273910
30    37.306081
55    28.224419
53    24.718994
26    35.429283
50    32.267032
42    38.303991
48    34.342240
33    34.375215
73    24.121018
2     42.367794
57    32.226089
dtype: float64
```

In [53]:
```
## check r2score
from sklearn.metrics import r2_score
```

In [54]:
```
score = r2_score(Y_test ,pred)
```

In [55]:
```
score
```

Out[55]:
```
0.8027513141600667
```

In [56]:
```
## mean
np.mean(pred)
```

Out[56]:
```
32.97940477451849
```

In [57]:
```
# Train residual values
error = Y_test.MPG -pred
```

In [58]:
```
error
```

Out[58]:
```
22    1.292321
27    0.423618
61   -6.416264
13    1.741474
71    1.452180
74   -7.187569
30    2.125154
55   -0.368167
53   -0.231627
26    2.981720
50   -2.637096
42   -4.233323
48   -3.328109
33    1.910241
73   -5.034678
2     7.645607
57   -2.596153
dtype: float64
```

In [59]:
```
### ## Scores with Cross Validation (cv)

lm = lm = LinearRegression()
```

In [75]:
```
from sklearn.model_selection import cross_val_score

cross =cross_val_score(lm ,X_train ,Y_train , scoring ='neg_mean_squared_error' ,cv=8)
```

In [76]:
```
cross
```

Out[76]:
```
array([-28.49153209, -25.55480712, -13.76367249, -15.54257037,
       -18.35795527,  -7.80334964, -40.0516394 ,  -5.1595768 ])
```

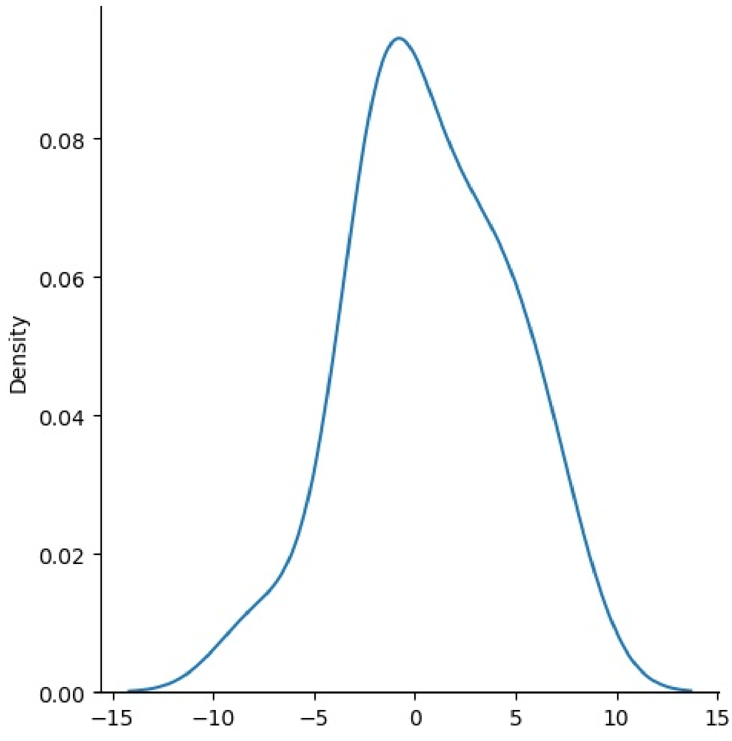
In [79]:
```
####check mean
np.mean(cross)
```

Out[79]:
```
-19.340637897928936
```

In [83]:
```
### to check th model is good or not if variance is low then model is good
sns.displot(pred-Y_test.MPG ,kind='kde')
```

```
C:\Users\shubham lokare\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

Out[83]:
```
<seaborn.axisgrid.FacetGrid at 0x1fb06fa88d0>
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js