In [ ]:
```
### Ridge and Lasso Regrassion
### it can be used to reduce the overfitting of model
## it also know as L1 and L2 Regularization
```

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [4]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from feature_engine.outliers import Winsorizer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

In [5]:
```
### import data

car = pd.read_csv(r"C:\Users\shubham lokare\Downloads\Lasso, Ridge and ElasticNet Regression\Lasso, Ridge and E
```

In [6]:
```
car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 6 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MPG         81 non-null     float64
 1   Enginetype  81 non-null     object 
 2   HP          81 non-null     int64  
 3   VOL         81 non-null     int64  
 4   SP          81 non-null     float64
 5   WT          81 non-null     float64
dtypes: float64(3), int64(2), object(1)
memory usage: 3.9+ KB
```

In [7]:
```
car.head(10)
```

Out[7]:

	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
5	45.696322	petrol	70	89	113.185353	29.591768
6	50.013401	cng	55	92	105.461264	30.308480
7	46.716554	cng	62	50	102.598513	15.847758
8	46.716554	lpg	62	50	102.598513	16.359484
9	42.299078	petrol	80	94	115.645204	30.920154

In [10]:
```
### split the data into input and output variable

X = pd.DataFrame(car.iloc[: ,1:6])
Y = pd.DataFrame(car.iloc[: ,0])
```

In [11]:
```
### input variable

print(X)
```

```
   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 x 5 columns]
```

In [12]:
```
### target

print(Y)
```

```
          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 x 1 columns]
```

In [18]:
```
### check unique enginetypes
X['Enginetype'].unique()
```

Out[18]:
```
array(['petrol', 'hybrid', 'diesel', 'lpg', 'cng'], dtype=object)
```

In [19]:
```
### count engine types
X['Enginetype'].value_counts()
```

Out[19]:
```
diesel    26
petrol    16
hybrid    16
lpg       12
cng       11
Name: Enginetype, dtype: int64
```

In [20]:
```
### check missing values
car.isna().sum()
```

Out[20]:
```
MPG           0
Enginetype    0
HP            0
VOL           0
SP            0
WT            0
dtype: int64
```

In [14]:
```
### separate numerical columns and categorical columns
numerical_featute = X.select_dtypes(exclude=['object']).columns

print(numerical_featute)
```

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

In [16]:
```
### categorical data

categorical_feature = X.select_dtypes(include=['object']).columns
print(categorical_feature)
```

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

In [17]:
```
### check how many types of engine

categorical_feature.unique()
```

Out[17]:
```
Index(['Enginetype'], dtype='object')
```

In [21]:
```
### make pipeline for missing values

num_pipe = Pipeline([('impute' , SimpleImputer(strategy='mean'))])
```

In [27]:
```
process = ColumnTransformer(transformers=[('impute' , num_pipe , numerical_featute)])
```

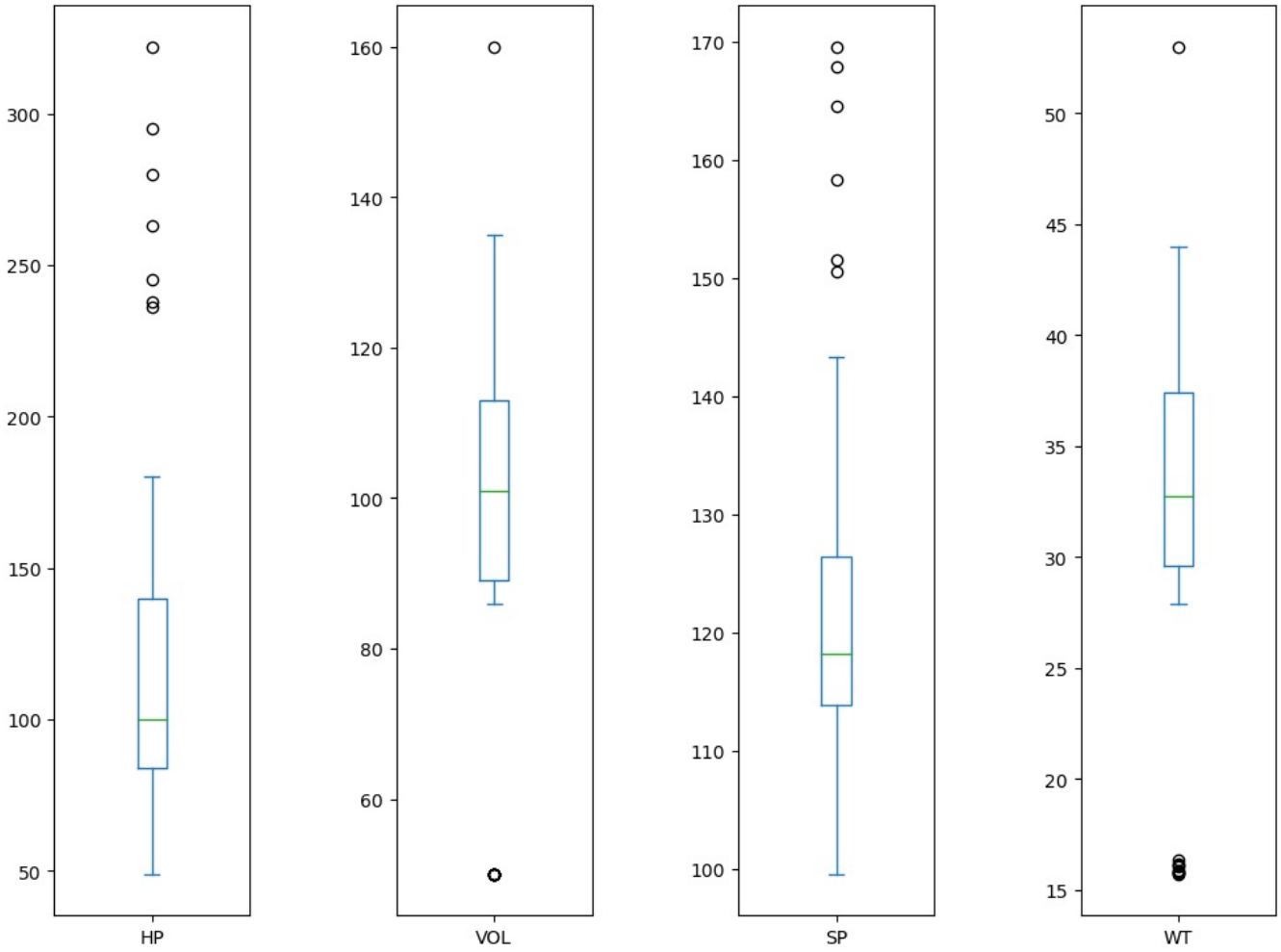
In [28]:
```
data = process.fit(X)
```

In [29]:
```
### add into DataFrame

clean = pd.DataFrame(data.transform(X) , columns=numerical_featute)
```

In [30]:
```
#### check outliers
### to check outliers we use boxplot

clean.plot(kind = 'box' ,subplots =True , figsize = (12,9))
plt.subplots_adjust(wspace=0.75)
plt.show()
```



In [33]:
```
#### to remove outliers we use winsor

winsor = Winsorizer(capping_method='iqr' ,
                    tail='both' ,
                   fold=1.5 ,
                   variables=list(clean.columns))
```

In [35]:
```
process1 = winsor.fit(clean)
```

In [47]:
```
### add into DataFrame

clean_data = pd.DataFrame(process1.transform(clean) , columns=numerical_featute)
```

In [48]:
```
clean_data
```

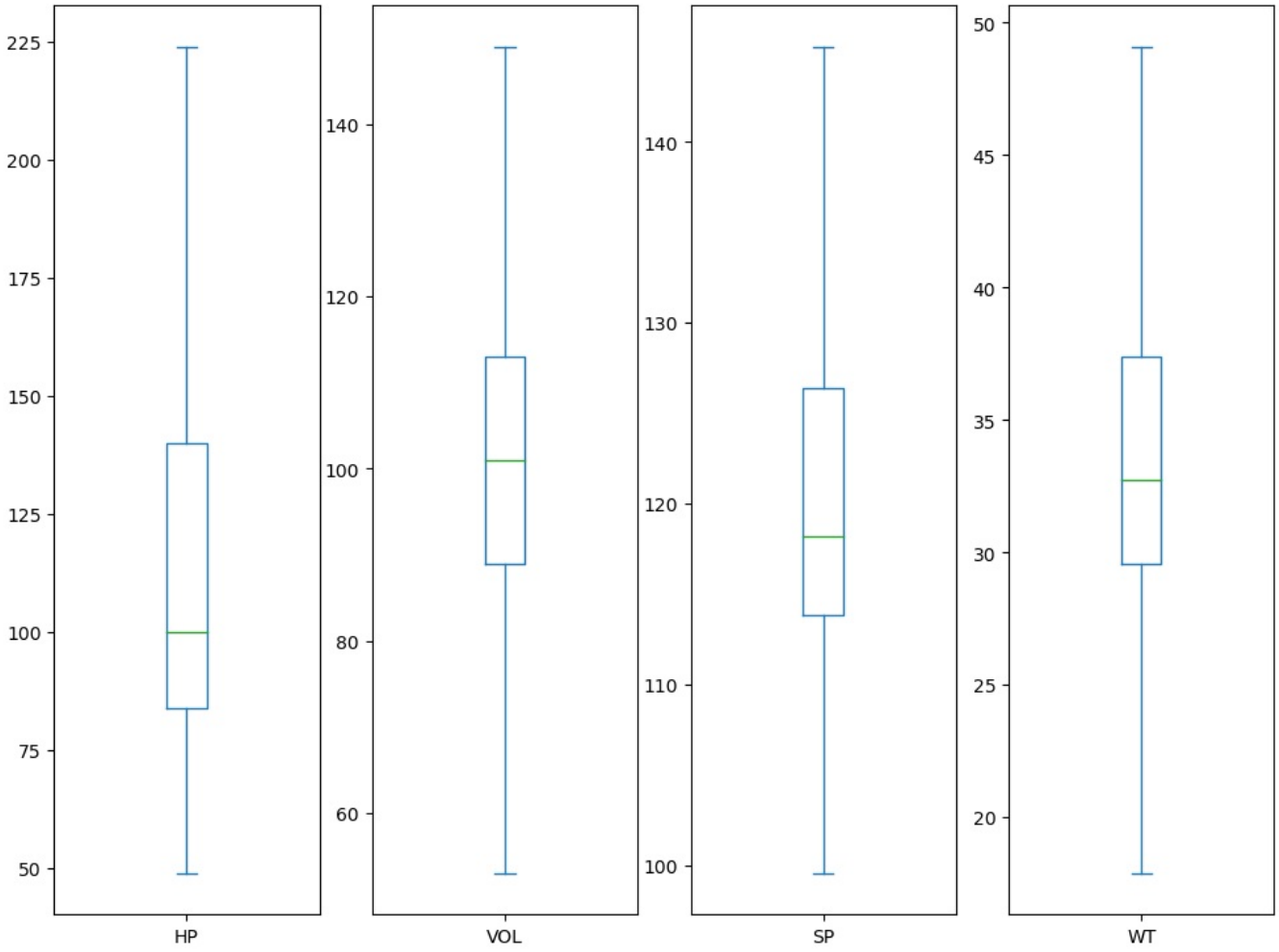
Out[48]:

	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	224.0	53.0	145.267062	17.890634
77	224.0	115.0	145.267062	37.923113
78	224.0	53.0	145.267062	17.890634
79	224.0	119.0	145.267062	39.423099
80	224.0	107.0	139.840817	34.948615

81 rows × 4 columns

In [49]:
```
#### check outliers
clean_data.plot(kind = 'box' ,subplots = True , figsize = (12,9))
```

Out[49]:
```
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 [56]:
```
#### make data into scale

scale_data = Pipeline([('scale' ,StandardScaler())])
```

In [57]:
```
process2 = ColumnTransformer([('scale'  ,scale_data , numerical_featute)])
```

In [58]:
```
data2 = process2.fit(clean_data)
```

In [59]:
```
### add into DataFrame

new_data = pd.DataFrame(data2.transform(clean_data) , columns=numerical_featute)
```

In [60]:
```
new_data
```

Out[60]:

	HP	VOL	SP	WT
0	-1.396782	-0.472540	-1.464507	-0.556902
1	-1.267109	-0.330251	-1.349385	-0.308167
2	-1.267109	-0.330251	-1.349385	-0.348033
3	-0.942928	-0.330251	-0.627565	-0.284051
4	-1.310333	-0.330251	-1.439612	-0.392454
...	...	...	...	...
76	2.385335	-2.180006	2.242191	-2.143098
77	2.385335	0.760631	2.242191	0.779743
78	2.385335	-2.180006	2.242191	-2.143098
79	2.385335	0.950349	2.242191	0.998598
80	2.385335	0.381194	1.752595	0.345748

81 rows × 4 columns

In [61]:
```
new_data.describe()
```

Out[61]:

	HP	VOL	SP	WT
count	8.100000e+01	8.100000e+01	8.100000e+01	8.100000e+01
mean	-6.579099e-17	2.809824e-17	2.275272e-16	4.817820e-16
std	1.006231e+00	1.006231e+00	1.006231e+00	1.006231e+00
min	-1.396782e+00	-2.180006e+00	-1.881398e+00	-2.143098e+00
25%	-6.403583e-01	-4.725396e-01	-5.943721e-01	-4.358429e-01
50%	-2.945648e-01	9.661591e-02	-1.992162e-01	2.270030e-02
75%	5.699189e-01	6.657714e-01	5.402530e-01	7.023272e-01
max	2.385335e+00	2.373238e+00	2.242191e+00	2.409582e+00

In [73]:
```
### for categorical variable
### we use onehot encoding

from sklearn.preprocessing import OneHotEncoder

cate_pipe = Pipeline([('onehot' ,OneHotEncoder(drop= 'first'))])
```

In [74]:
```
processer = ColumnTransformer([('one' , cate_pipe , categorical_feature)])
```

In [75]:
```
data3 = processer.fit(X)
```

In [76]:
```
### add into dataframe

new = pd.DataFrame(data3.transform(X).todense())
```

In [77]:
```
new.columns = data3.get_feature_names_out(input_features=X.columns)
```

In [78]:
```
new
```

Out[78]:

	one__Enginetype_diesel	one__Enginetype_hybrid	one__Enginetype_lpg	one__Enginetype_petrol
0	0.0	0.0	0.0	1.0
1	0.0	1.0	0.0	0.0
2	1.0	0.0	0.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	0.0	1.0
...	...	...	...	...
76	0.0	1.0	0.0	0.0
77	0.0	0.0	1.0	0.0
78	0.0	1.0	0.0	0.0
79	1.0	0.0	0.0	0.0
80	0.0	1.0	0.0	0.0

81 rows × 4 columns

In [80]:
```
#### combine both numerical feature and categorical feature

cleandata = pd.concat([new_data , new] , axis = 1)
```

In [81]:
```
cleandata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 81 entries, 0 to 80
Data columns (total 8 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   one__Enginetype_diesel  81 non-null     float64
 5   one__Enginetype_hybrid  81 non-null     float64
 6   one__Enginetype_lpg     81 non-null     float64
 7   one__Enginetype_petrol  81 non-null     float64
dtypes: float64(8)
memory usage: 5.2 KB
```

In [82]:
```
cleandata.describe().T
```

Out[82]:

	count	mean	std	min	25%	50%	75%	max
HP	81.0	-6.579099e-17	1.006231	-1.396782	-0.640358	-0.294565	0.569919	2.385335
VOL	81.0	2.809824e-17	1.006231	-2.180006	-0.472540	0.096616	0.665771	2.373238
SP	81.0	2.275272e-16	1.006231	-1.881398	-0.594372	-0.199216	0.540253	2.242191
WT	81.0	4.817820e-16	1.006231	-2.143098	-0.435843	0.022700	0.702327	2.409582
one__Enginetype_diesel	81.0	3.209877e-01	0.469765	0.000000	0.000000	0.000000	1.000000	1.000000
one__Enginetype_hybrid	81.0	1.975309e-01	0.400617	0.000000	0.000000	0.000000	0.000000	1.000000
one__Enginetype_lpg	81.0	1.481481e-01	0.357460	0.000000	0.000000	0.000000	0.000000	1.000000
one__Enginetype_petrol	81.0	1.975309e-01	0.400617	0.000000	0.000000	0.000000	0.000000	1.000000

In [83]:
```
cleandata
```

Out[83]:

	HP	VOL	SP	WT	one__Enginetype_diesel	one__Enginetype_hybrid	one__Enginetype_lpg	one__Enginetype_petrol
0	-1.396782	-0.472540	-1.464507	-0.556902	0.0	0.0	0.0	1.0
1	-1.267109	-0.330251	-1.349385	-0.308167	0.0	1.0	0.0	0.0
2	-1.267109	-0.330251	-1.349385	-0.348033	1.0	0.0	0.0	0.0
3	-0.942928	-0.330251	-0.627565	-0.284051	0.0	0.0	1.0	0.0
4	-1.310333	-0.330251	-1.439612	-0.392454	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...
76	2.385335	-2.180006	2.242191	-2.143098	0.0	1.0	0.0	0.0
77	2.385335	0.760631	2.242191	0.779743	0.0	0.0	1.0	0.0
78	2.385335	-2.180006	2.242191	-2.143098	0.0	1.0	0.0	0.0
79	2.385335	0.950349	2.242191	0.998598	1.0	0.0	0.0	0.0
80	2.385335	0.381194	1.752595	0.345748	0.0	1.0	0.0	0.0

81 rows × 8 columns

In [94]:
```
### apply model

from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn .metrics import r2_score

from sklearn.model_selection import GridSearchCV
```

In [87]:
```
X_train ,X_test , Y_train ,Y_test = train_test_split(cleandata ,Y , test_size = 0.2 , random_state=0)
```

In [88]:
```
## train data
X_train
```

Out[88]:

	HP	VOL	SP	WT	one__Enginetype_diesel	one__Enginetype_hybrid	one__Enginetype_lpg	one__Enginetype_petrol
66	1.110221	0.713201	1.088863	0.788036	0.0	0.0	1.0	0.0
41	-0.402625	0.333764	-0.421162	0.357211	0.0	0.0	0.0	1.0
3	-0.942928	-0.330251	-0.627565	-0.284051	0.0	0.0	1.0	0.0
69	2.385335	0.618342	2.242191	0.665735	1.0	0.0	0.0	0.0
7	-1.115824	-2.180006	-1.607683	-2.143098	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...
77	2.385335	0.760631	2.242191	0.779743	0.0	0.0	1.0	0.0
67	1.110221	1.329786	1.196740	1.312422	0.0	0.0	1.0	0.0
64	0.786040	1.045208	0.695813	1.106040	0.0	0.0	0.0	0.0
47	-0.337789	0.191475	-0.355830	0.194217	0.0	1.0	0.0	0.0
44	-0.445849	0.144046	-0.544583	0.072908	1.0	0.0	0.0	0.0

64 rows × 8 columns

In [89]:
```
### test data
Y_test
```

Out[89]:

	MPG
22	38.310606
27	38.411003
61	24.609132
13	44.652834
71	23.203569
74	19.086341
30	39.431235
55	27.856252
53	24.487367
26	38.411003
50	29.629936
42	34.070668
48	31.014131
33	36.285456
73	19.086341
2	50.013401
57	29.629936

In [95]:
```
model = LinearRegression()
```

In [96]:
```
model.fit(X_train , Y_train)
```

Out[96]:
```
▾ LinearRegression
LinearRegression()
```

In [97]:
```
## prediction

pred = model.predict(X_test)
```

In [98]:
```
pred
```

Out[98]:
```
array([[37.01828541],
       [37.98738549],
       [31.02539595],
       [42.9113601 ],
       [21.75138851],
       [26.2739096 ],
       [37.30608145],
       [28.22441872],
       [24.71899374],
       [35.42928331],
       [32.26703165],
       [38.30399094],
       [34.3422397 ],
       [34.37521542],
       [24.12101843],
       [42.36779421],
       [32.22608855]])
```

In [100…
```
### check r2 score

score = r2_score(Y_test , pred)
```

In [101…
```
score
```

Out[101]:
```
0.8027513141600642
```

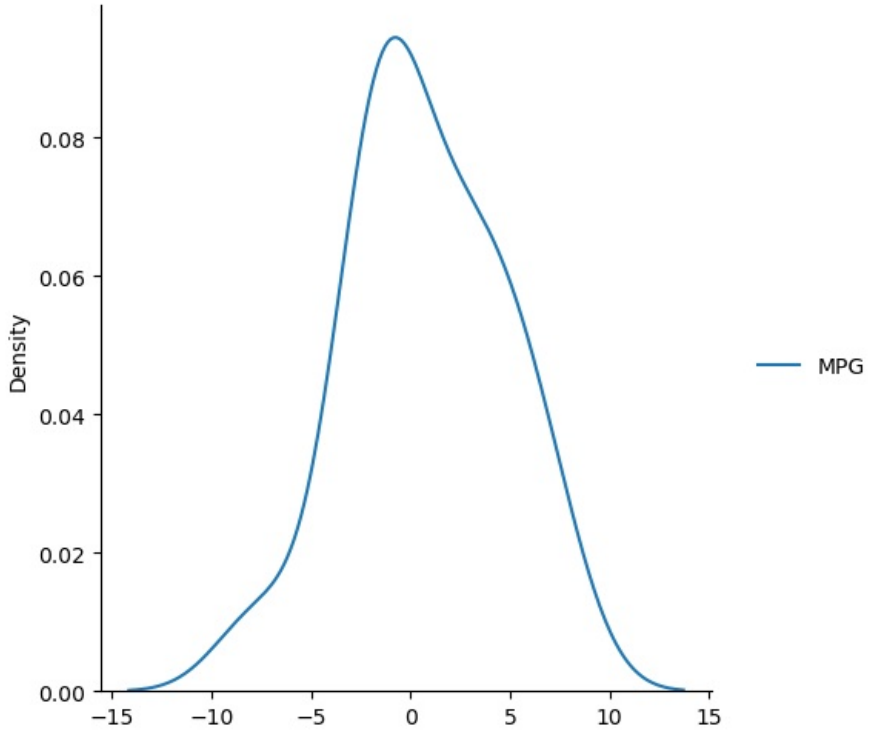
In [102…
```
## mean

np.mean(pred)
```

Out[102]:
```
32.9794047745186
```

In [132…
```
### plot kde plot to check variance
import seaborn as sns
plt.figure(figsize=(12,8))
sns.displot(pred-Y_test  , kind = 'kde')
plt.show()
```

```
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)
<Figure size 1200x800 with 0 Axes>
```



In [158…
```
### to reduce the overfitting and reduce to variance we use Ridge and Lasso  technique

from sklearn.linear_model import Ridge


ridge = Ridge(alpha=5)

ridge.fit(X_train ,Y_train)
```

Out[158]:
```
Ridge
Ridge(alpha=5)
```

In [159…
```
## predection

preds = ridge.predict(X_test)
```

In [160…
```
preds
```

Out[160]:
```
array([[35.51824268],
       [36.59983046],
       [31.99901542],
       [42.32125062],
       [21.86202929],
       [27.36174566],
       [37.28401524],
       [28.14236349],
       [21.9249024 ],
       [33.34678266],
       [33.54276198],
       [38.07661329],
       [33.15296603],
       [32.63976696],
       [24.05177698],
       [42.661865  ],
       [34.17234149]])
```

In [161…
```
score1 = r2_score(Y_test , preds)
```

In [162…
```
score1
```

Out[162]:
```
0.7352838701153385
```

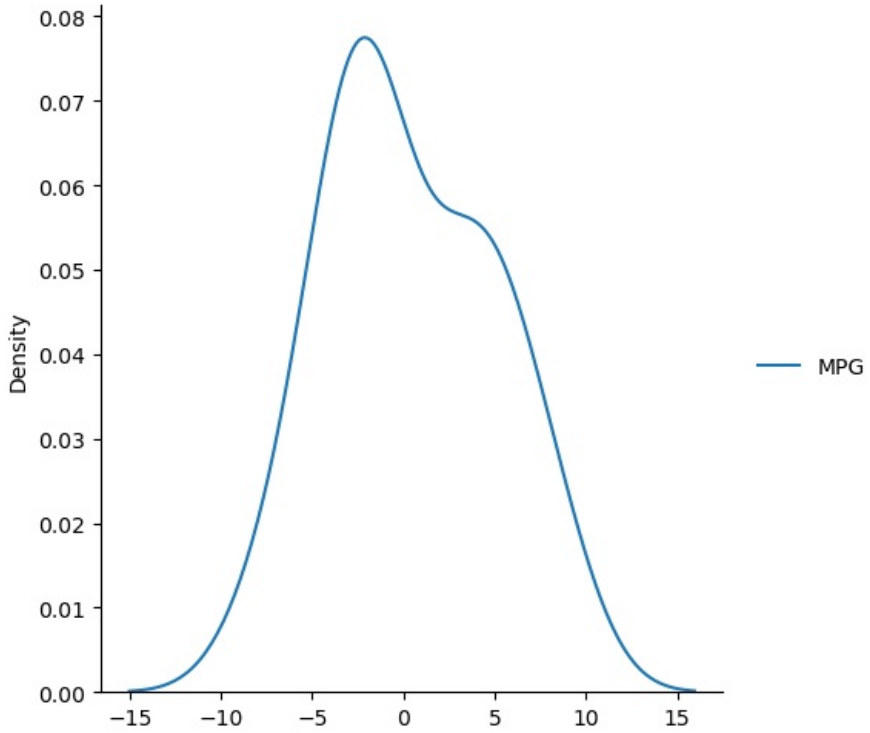
In [163…
```
### mean
np.mean(preds)
```

Out[163]:
```
32.62695703719638
```

In [164…
```
### plot
plt.figure(figsize = (12, 5))
```

```
sns.displot(preds -Y_test , kind = 'kde')
plt.show()
```

```
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)
<Figure size 1200x500 with 0 Axes>
```



In [137…
```
### use hypertunning to find best value of alpha

ridge_model =Ridge()
```

In [138…
```
ridge_model
```

Out[138]:
```
Ridge()
```

In [155…
```
### use hypertunning to find best value of alpha
parameter = {'alpha':[5,10,20,30,40 ,50,60,70,80 ,90]}

grid =GridSearchCV(ridge_model ,parameter , scoring='neg_mean_squared_error' , cv = 5 )
```

In [156…
```
grid.fit(X_train , Y_train)
```

Out[156]:
```
GridSearchCV
estimator: Ridge
Ridge
```

In [157…
```
grid.best_params_
```

Out[157]:
```
{'alpha': 5}
```

In [165…
```
print(grid.best_score_)
```

```
-17.49173326835934
```

In [166…
```
prediction = grid.predict(X_test)
```

In [167…
```
prediction
```

Out[167]:
```
array([[35.51824268],
       [36.59983046],
       [31.99901542],
       [42.32125062],
       [21.86202929],
       [27.36174566],
       [37.28401524],
       [28.14236349],
       [21.9249024 ],
       [33.34678266],
       [33.54276198],
       [38.07661329],
       [33.15296603],
       [32.63976696],
       [24.05177698],
       [42.661865  ],
       [34.17234149]])
```

In [168…
```
score = r2_score(Y_test , prediction)
```

In [169…
```
score
```

Out[169]:
```
0.7352838701153385
```

In [170…
```
#### also use Lasso


from sklearn.linear_model import Lasso

lasso_model = Lasso()
```

In [171…
```
lasso_model.fit(X_train ,Y_train)
```

Out[171]:
```
▾ Lasso
Lasso()
```

In [172…
```
lasso_pred = lasso_model.predict(X_test)
```

In [173…
```
lasso_pred
```

Out[173]:
```
array([36.92795998, 37.23996524, 31.88030347, 40.89438716, 24.36151915,
       27.61265527, 36.65014267, 29.61510245, 24.53317255, 35.27493037,
       34.24660739, 36.86783419, 34.32060965, 34.21939932, 25.17881417,
       42.8191516 , 34.42682016])
```

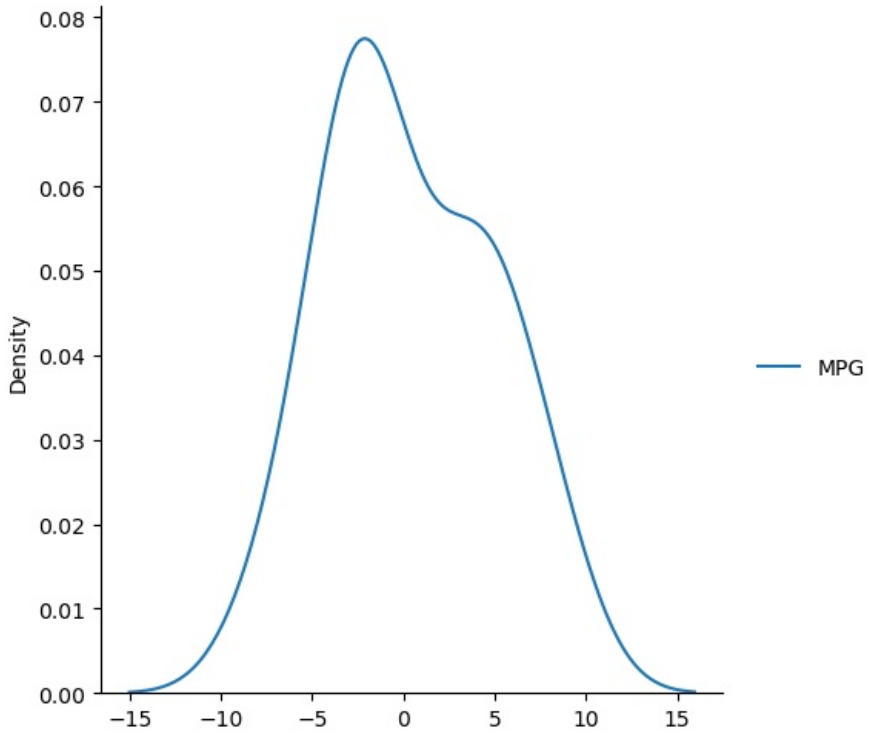
In [174…
```
score = r2_score(Y_test ,lasso_pred)
```

In [175…
```
score
```

Out[175]:
```
0.7398686715277072
```

In [179…
```
### plot
plt.figure(figsize= (12,8))
sns.displot(prediction-Y_test , kind='kde')
plt.show()
```

```
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)
<Figure size 1200x800 with 0 Axes>
```



In [ ]: