LINEAR REGRESSION IMPLEMENTATION

Problem Statement :

The problem that we are going to solve here is that given a set of features that describe a house in Boston, our machine learning model must predict the house price. To train our machine learning model with boston housing data, we will be using scikit-learn's boston dataset.

Data Collection:

The dataset for this project originates from the UCI Machine Learning Repository. The Boston housing data was collected in 1978 and each of the 506 entries represent aggregated data about 14 features for homes from various suburbs in Boston, Massachusetts. This is a copy of UCI ML housing dataset. https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)

Importing important libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Importing Data (Boston Housing Dataset)

In [2]:
```
from sklearn.datasets import load_boston
```

In [3]:
```
boston = load_boston()
```

In [4]:
```
boston
```

Out[4]:
```
{'data': array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
         4.9800e+00],
        [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
         9.1400e+00],
        [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
         4.0300e+00],
        ...,
        [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         5.6400e+00],
        [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
         6.4800e+00],
        [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
         7.8800e+00]]),
 'target': array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9, 15. ,
        18.9, 21.7, 20.4, 18.2, 19.9, 23.1, 17.5, 20.2, 18.2, 13.6, 19.6,
        15.2, 14.5, 15.6, 13.9, 16.6, 14.8, 18.4, 21. , 12.7, 14.5, 13.2,
        13.1, 13.5, 18.9, 20. , 21. , 24.7, 30.8, 34.9, 26.6, 25.3, 24.7,
        21.2, 19.3, 20. , 16.6, 14.4, 19.4, 19.7, 20.5, 25. , 23.4, 18.9,
        35.4, 24.7, 31.6, 23.3, 19.6, 18.7, 16. , 22.2, 25. , 33. , 23.5,
        19.4, 22. , 17.4, 20.9, 24.2, 21.7, 22.8, 23.4, 24.1, 21.4, 20. ,
        20.8, 21.2, 20.3, 28. , 23.9, 24.8, 22.9, 23.9, 26.6, 22.5, 22.2,
        23.6, 28.7, 22.6, 22. , 22.9, 25. , 20.6, 28.4, 21.4, 38.7, 43.8,
        33.2, 27.5, 26.5, 18.6, 19.3, 20.1, 19.5, 19.5, 20.4, 19.8, 19.4,
        21.7, 22.8, 18.8, 18.7, 18.5, 18.3, 21.2, 19.2, 20.4, 19.3, 22. ,
        20.3, 20.5, 17.3, 18.8, 21.4, 15.7, 16.2, 18. , 14.3, 19.2, 19.6,
        23. , 18.4, 15.6, 18.1, 17.4, 17.1, 13.3, 17.8, 14. , 14.4, 13.4,
        15.6, 11.8, 13.8, 15.6, 14.6, 17.8, 15.4, 21.5, 19.6, 15.3, 19.4,
        17. , 15.6, 13.1, 41.3, 24.3, 23.3, 27. , 50. , 50. , 50. , 22.7,
        25. , 50. , 23.8, 23.8, 22.3, 17.4, 19.1, 23.1, 23.6, 22.6, 29.4,
        23.2, 24.6, 29.9, 37.2, 39.8, 36.2, 37.9, 32.5, 26.4, 29.6, 50. ,
        32. , 29.8, 34.9, 37. , 30.5, 36.4, 31.1, 29.1, 50. , 33.3, 30.3,
        34.6, 34.9, 32.9, 24.1, 42.3, 48.5, 50. , 22.6, 24.4, 22.5, 24.4,
        20. , 21.7, 19.3, 22.4, 28.1, 23.7, 25. , 23.3, 28.7, 21.5, 23. ,
        26.7, 21.7, 27.5, 30.1, 44.8, 50. , 37.6, 31.6, 46.7, 31.5, 24.3,
        31.7, 41.7, 48.3, 29. , 24. , 25.1, 31.5, 23.7, 23.3, 22. , 20.1,
        22.2, 23.7, 17.6, 18.5, 24.3, 20.5, 24.5, 26.2, 24.4, 24.8, 29.6,
        42.8, 21.9, 20.9, 44. , 50. , 36. , 30.1, 33.8, 43.1, 48.8, 31. ,
        36.5, 22.8, 30.7, 50. , 43.5, 20.7, 21.1, 25.2, 24.4, 35.2, 32.4,
        32. , 33.2, 33.1, 29.1, 35.1, 45.4, 35.4, 46. , 50. , 32.2, 22. ,
        20.1, 23.2, 22.3, 24.8, 28.5, 37.3, 27.9, 23.9, 21.7, 28.6, 27.1,
        20.3, 22.5, 29. , 24.8, 22. , 26.4, 33.1, 36.1, 28.4, 33.4, 28.2,
        22.8, 20.3, 16.1, 22.1, 19.4, 21.6, 23.8, 16.2, 17.8, 19.8, 23.1,
        21. , 23.8, 23.1, 20.4, 18.5, 25. , 24.6, 23. , 22.2, 19.3, 22.6,
        19.8, 17.1, 19.4, 22.2, 20.7, 21.1, 19.5, 18.5, 20.6, 19. , 18.7,
        32.7, 16.5, 23.9, 31.2, 17.5, 17.2, 23.1, 24.5, 26.6, 22.9, 24.1,
        18.6, 30.1, 18.2, 20.6, 17.8, 21.7, 22.7, 22.6, 25. , 19.9, 20.8,
        16.8, 21.9, 27.5, 21.9, 23.1, 50. , 50. , 50. , 50. , 50. , 13.8,
        13.8, 15. , 13.9, 13.3, 13.1, 10.2, 10.4, 10.9, 11.3, 12.3,  8.8,
         7.2, 10.5,  7.4, 10.2, 11.5, 15.1, 23.2,  9.7, 13.8, 12.7, 13.1,
        12.5,  8.5,  5. ,  6.3,  5.6,  7.2, 12.1,  8.3,  8.5,  5. , 11.9,
        27.9, 17.2, 27.5, 15. , 17.2, 17.9, 16.3,  7. ,  7.2,  7.5, 10.4,
         8.8,  8.4, 16.7, 14.2, 20.8, 13.4, 11.7,  8.3, 10.2, 10.9, 11. ,
         9.5, 14.5, 14.1, 16.1, 14.3, 11.7, 13.4,  9.6,  8.7,  8.4, 12.8,
```

```
        10.5, 17.1, 18.4, 15.4, 10.8, 11.8, 14.9, 12.6, 14.1, 13. , 13.4,
        15.2, 16.1, 17.8, 14.9, 14.1, 12.7, 13.5, 14.9, 20. , 16.4, 17.7,
        19.5, 20.2, 21.4, 19.9, 19. , 19.1, 19.1, 20.1, 19.9, 19.6, 23.2,
        29.8, 13.8, 13.3, 16.7, 12. , 14.6, 21.4, 23. , 23.7, 25. , 21.8,
        20.6, 21.2, 19.1, 20.6, 15.2,  7. ,  8.1, 13.6, 20.1, 21.8, 24.5,
        23.1, 19.7, 18.3, 21.2, 17.5, 16.8, 22.4, 20.6, 23.9, 22. , 11.9]),
 'feature_names': array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DI
S', 'RAD',
        'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7'),
 'DESCR': ".. _boston_dataset:\n\nBoston house prices dataset\n----------------
-----------\n\n**Data Set Characteristics:**  \n\n    :Number of Instances: 506
\n\n    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.\n\n    :Attribute Information (in orde
r):\n        - CRIM     per capita crime rate by town\n        - ZN       propo
rtion of residential land zoned for lots over 25,000 sq.ft.\n        - INDUS
proportion of non-retail business acres per town\n        - CHAS     Charles Ri
ver dummy variable (= 1 if tract bounds river; 0 otherwise)\n        - NOX
nitric oxides concentration (parts per 10 million)\n        - RM       average
number of rooms per dwelling\n        - AGE      proportion of owner-occupied u
nits built prior to 1940\n        - DIS      weighted distances to five Boston
employment centres\n        - RAD      index of accessibility to radial highway
s\n        - TAX      full-value property-tax rate per $10,000\n        - PTRAT
IO  pupil-teacher ratio by town\n        - B        1000(Bk - 0.63)^2 where Bk
is the proportion of black people by town\n        - LSTAT    % lower status of
the population\n        - MEDV     Median value of owner-occupied homes in $100
0's\n\n    :Missing Attribute Values: None\n\n    :Creator: Harrison, D. and Ru
binfeld, D.L.\n\nThis is a copy of UCI ML housing dataset.\nhttps://archive.ic
s.uci.edu/ml/machine-learning-databases/housing/\n\n\nThis dataset was taken fr
om the StatLib library which is maintained at Carnegie Mellon University.\n\nTh
e Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic\nprices
and the demand for clean air', J. Environ. Economics & Management,\nvol.5, 81-1
02, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wile
y, 1980.   N.B. Various transformations are used in the table on\npages 244-261
of the latter.\n\nThe Boston house-price data has been used in many machine lea
rning papers that address regression\nproblems.   \n     \n.. topic:: Reference
s\n\n   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influenti
al Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Quinlan,R. (1
993). Combining Instance-Based and Model-Based Learning. In Proceedings on the
Tenth International Conference of Machine Learning, 236-243, University of Mass
achusetts, Amherst. Morgan Kaufmann.\n",
 'filename': 'boston_house_prices.csv',
 'data_module': 'sklearn.datasets.data'}
```

In [5]:
```
# The information provided in the dataset is as follows:
boston.keys()
```

Out[5]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename', 'data_modul
e'])
```

In [6]:
```
# The description about the data is as follows:
print(boston.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (att
ribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 s
q.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 o
therwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of black people
by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://arc
hive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie
Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that
 address regression
problems.

.. topic:: References
```

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Da
ta and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In P
roceedings on the Tenth International Conference of Machine Learning, 236-243,
 University of Massachusetts, Amherst. Morgan Kaufmann.
```

In simple terms :-

- Data :- Independent Variables also known as the x values.
- feature_names :- The column names of the data.
- target :- The target variable or the price of the houses(dependent variable) alse known as y value.
- we are trying to predict the house of price based on all independent features

The data :

In [7]:
```
print(boston.data)
```

```
[[6.3200e-03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0959e-01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e-02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]
```

The target :

In [8]:
```
print(boston.target)
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21.  12.7 14.5 13.2 13.1 13.5 18.9 20.  21.  24.7 30.8 34.9 26.6
 25.3 24.7 21.2 19.3 20.  16.6 14.4 19.4 19.7 20.5 25.  23.4 18.9 35.4
 24.7 31.6 23.3 19.6 18.7 16.  22.2 25.  33.  23.5 19.4 22.  17.4 20.9
 24.2 21.7 22.8 23.4 24.1 21.4 20.  20.8 21.2 20.3 28.  23.9 24.8 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22.  22.9 25.  20.6 28.4 21.4 38.7
 43.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.3 22.  20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18.  14.3 19.2 19.6 23.  18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14.  14.4 13.4 15.6 11.8 13.8 15.6 14.6 17.8 15.4 21.5 19.6 15.3 19.4
 17.  15.6 13.1 41.3 24.3 23.3 27.  50.  50.  50.  22.7 25.  50.  23.8
 23.8 22.3 17.4 19.1 23.1 23.6 22.6 29.4 23.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50.  32.  29.8 34.9 37.  30.5 36.4 31.1 29.1 50.
 33.3 30.3 34.6 34.9 32.9 24.1 42.3 48.5 50.  22.6 24.4 22.5 24.4 20.
 21.7 19.3 22.4 28.1 23.7 25.  23.3 28.7 21.5 23.  26.7 21.7 27.5 30.1
 44.8 50.  37.6 31.6 46.7 31.5 24.3 31.7 41.7 48.3 29.  24.  25.1 31.5
 23.7 23.3 22.  20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44.  50.  36.  30.1 33.8 43.1 48.8 31.  36.5 22.8
 30.7 50.  43.5 20.7 21.1 25.2 24.4 35.2 32.4 32.  33.2 33.1 29.1 35.1
 45.4 35.4 46.  50.  32.2 22.  20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29.  24.8 22.  26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.3 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21.  23.8 23.1
 20.4 18.5 25.  24.6 23.  22.2 19.3 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19.  18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25.  19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50.  50.  50.  50.  50.  13.8 13.8 15.  13.9 13.3
 13.1 10.2 10.4 10.9 11.3 12.3  8.8  7.2 10.5  7.4 10.2 11.5 15.1 23.2
  9.7 13.8 12.7 13.1 12.5  8.5  5.   6.3  5.6  7.2 12.1  8.3  8.5  5.
 11.9 27.9 17.2 27.5 15.  17.2 17.9 16.3  7.   7.2  7.5 10.4  8.8  8.4
 16.7 14.2 20.8 13.4 11.7  8.3 10.2 10.9 11.   9.5 14.5 14.1 16.1 14.3
 11.7 13.4  9.6  8.7  8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13.  13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20.  16.4 17.7
 19.5 20.2 21.4 19.9 19.  19.1 19.1 20.1 19.9 19.6 23.2 29.8 13.8 13.3
 16.7 12.  14.6 21.4 23.  23.7 25.  21.8 20.6 21.2 19.1 20.6 15.2  7.
  8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.3 21.2 17.5 16.8 22.4 20.6 23.9
 22.  11.9]
```

FEATURES

In [9]:
```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

Features Information:

The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

- **CRIM** - per capita crime rate by town
- **ZN** - proportion of residential land zoned for lots over 25,000 sq.ft.
- **INDUS** - proportion of non-retail business acres per town.
- **CHAS** - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- **NOX** - nitric oxides concentration (parts per 10 million)
- **RM** - average number of rooms per dwelling
- **AGE** - proportion of owner-occupied units built prior to 1940
- **DIS** - weighted distances to five Boston employment centres
- **RAD** - index of accessibility to radial highways
- **TAX** - full-value property-tax rate per 1000 dollars
- **PTRATIO** - pupil-teacher ratio by town
- **B** - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- **LSTAT** - % lower status of the population
- **MEDV** - Median value of owner-occupied homes in $1000's

Preparing Dataframe

In [10]:
```
df = pd.DataFrame(boston.data , columns = boston.feature_names)
```

In [11]:
```
df.head()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Creating Output Feature > 'Price'

In [12]:
```
df['Price'] = boston.target
```

In [13]:
```
df.head()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Sample Data

In [14]:
```
df.sample(10)
```

Out[14]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
277	0.06127	40.0	6.41	1.0	0.447	6.826	27.6	4.8628	4.0	254.0	17.6	393.45	
394	13.35980	0.0	18.10	0.0	0.693	5.887	94.7	1.7821	24.0	666.0	20.2	396.90	1
111	0.10084	0.0	10.01	0.0	0.547	6.715	81.6	2.6775	6.0	432.0	17.8	395.59	1
219	0.11425	0.0	13.89	1.0	0.550	6.373	92.4	3.3633	5.0	276.0	16.4	393.74	1
357	3.84970	0.0	18.10	1.0	0.770	6.395	91.0	2.5052	24.0	666.0	20.2	391.34	1
303	0.10000	34.0	6.09	0.0	0.433	6.982	17.7	5.4917	7.0	329.0	16.1	390.43	
16	1.05393	0.0	8.14	0.0	0.538	5.935	29.3	4.4986	4.0	307.0	21.0	386.85	
442	5.66637	0.0	18.10	0.0	0.740	6.219	100.0	2.0048	24.0	666.0	20.2	395.69	1
315	0.25356	0.0	9.90	0.0	0.544	5.705	77.7	3.9450	4.0	304.0	18.4	396.42	1
88	0.05660	0.0	3.41	0.0	0.489	7.007	86.3	3.4217	2.0	270.0	17.8	396.90	

EDA

Basic Info about data

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  Price    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

Observation:

- The datafeame has 14 rows where the rows from 0 to 12 are Independent Features and 1 dependent Feature -'Price'.
- All the Features have float datatype and are Numerical.
- The memory usage is 55.5 KB .

Statistical Description

In [16]:
```
df.describe()
```

Out[16]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.00
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.79
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.10
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.20
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

Checking for missing Values

In [17]:
```
df.isnull().sum()
```

Out[17]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
Price      0
dtype: int64
```

Observation:

There are no null values in the dataset

Univariate analysis :

variance in the variables:

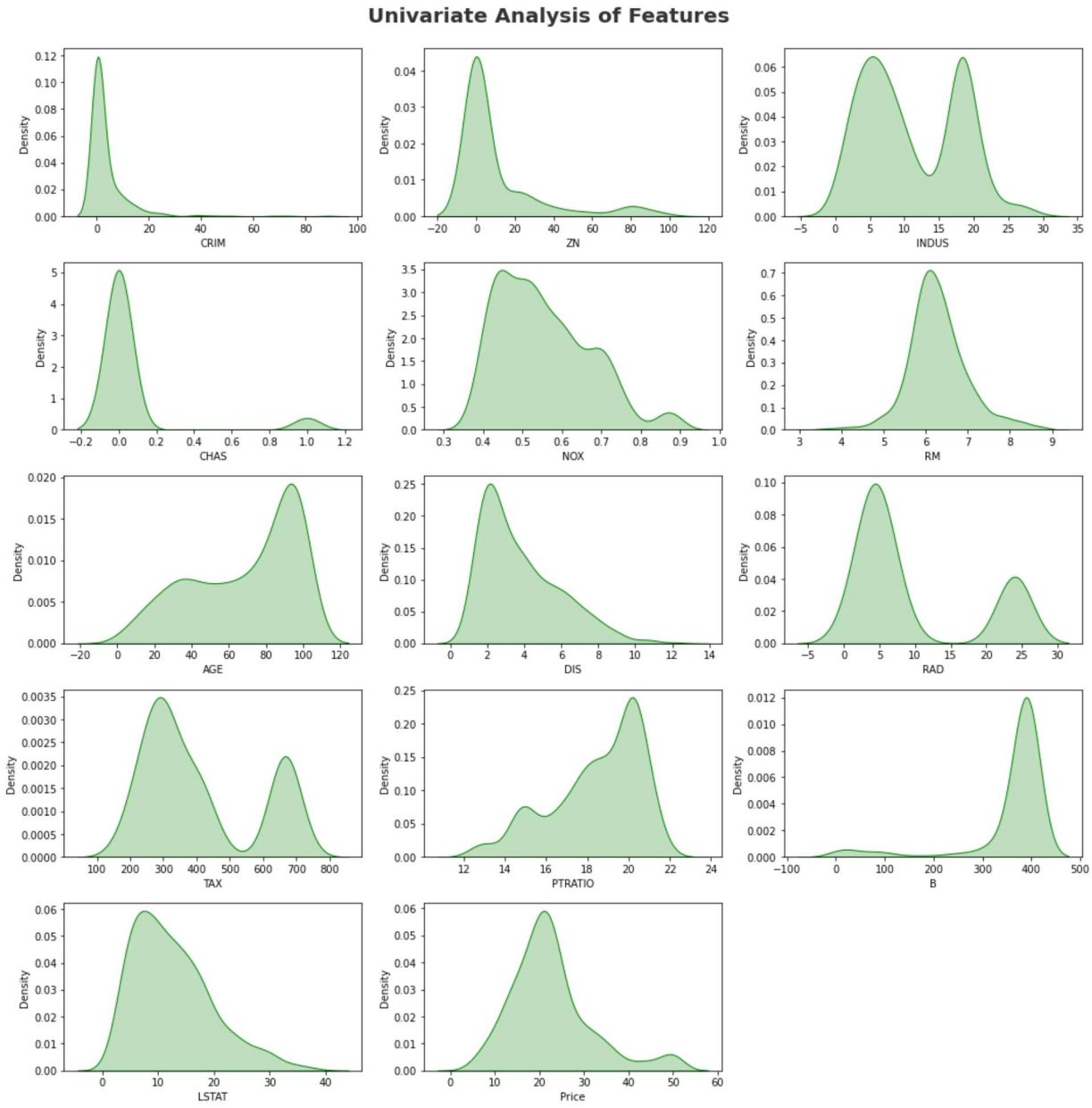
In [18]:
```
df.var()
```

Out[18]:
```
CRIM          73.986578
ZN           543.936814
INDUS         47.064442
CHAS           0.064513
NOX            0.013428
RM             0.493671
AGE          792.358399
DIS            4.434015
RAD           75.816366
TAX        28404.759488
PTRATIO        4.686989
B           8334.752263
LSTAT         50.994760
Price         84.586724
dtype: float64
```

Kernal Density Estimator plot for each feature

In [19]:
```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Features', fontsize=20, fontweight='bold', a

for item in range(0,len(df.columns)):
    plt.subplot(5, 3 , item+1)
    sns.kdeplot(x=df[df.columns[item]],shade=True, color='g')
    plt.xlabel(df.columns[item])
    plt.tight_layout()
```



Observation:

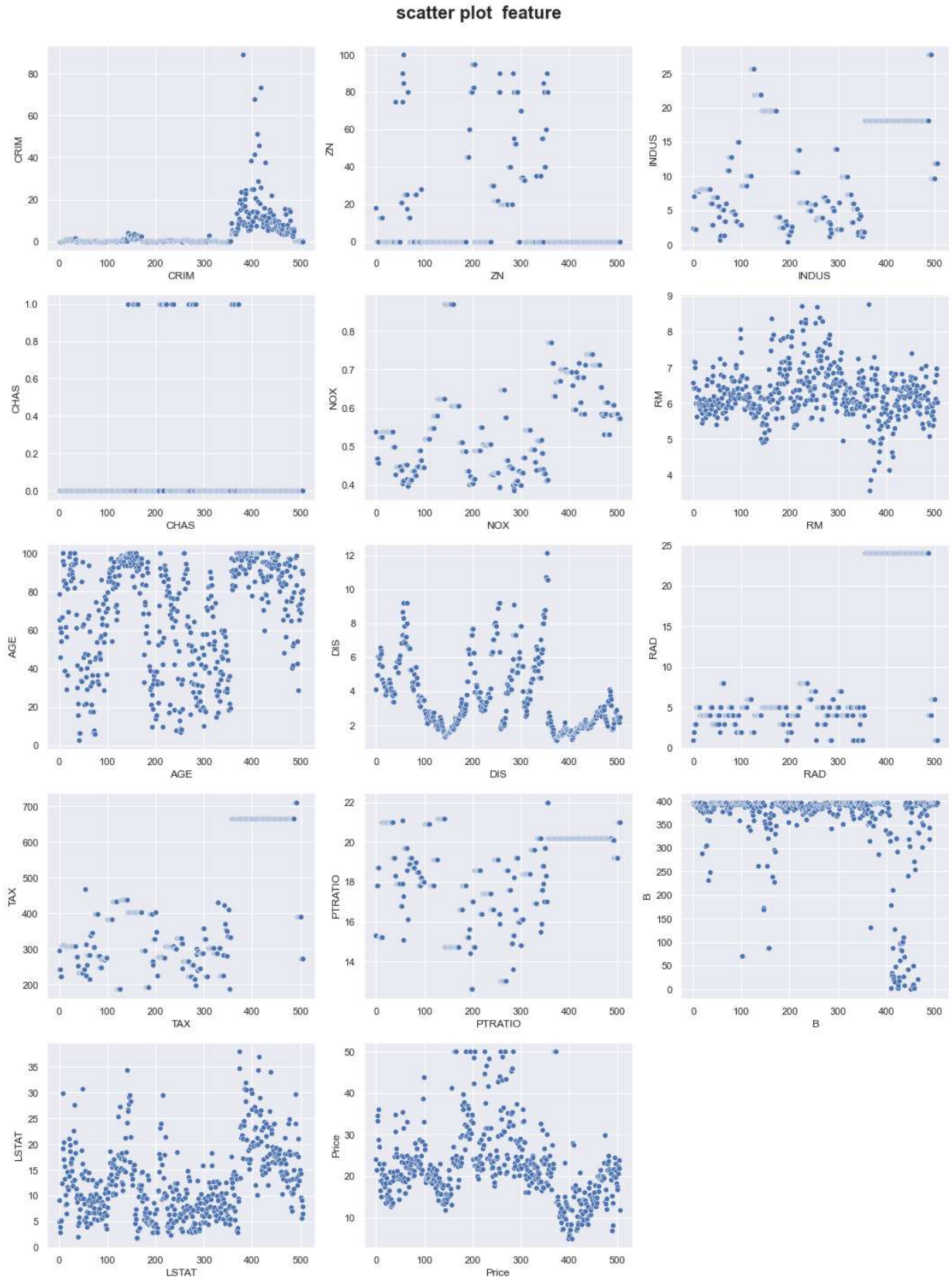
- Most of the features have approx normal distribution

- Features - CRIM , ZN , CHAS , NDS , LSTAT and Price are positively skewed
- Features - B and Age are negatively skewed
- Features- TAX , RAD, CHAS , INDUS are BIMODAL
- features PTRATION , NOX and Age have non normal distribution

Scatter plot for trends:

In [93]:
```
plt.figure(figsize=(15, 20))
plt.suptitle('scatter plot  feature', fontsize=20, fontweight='bold', alpha=1, y=

for i in range(0, len(df.columns)):
    plt.subplot(5, 3, i+1)
    sns.scatterplot(y=df[df.columns[i]], x=df.index, data=df )
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



Bivariate analysis:

Correlation between features

In [21]:
```
df.corr()
```

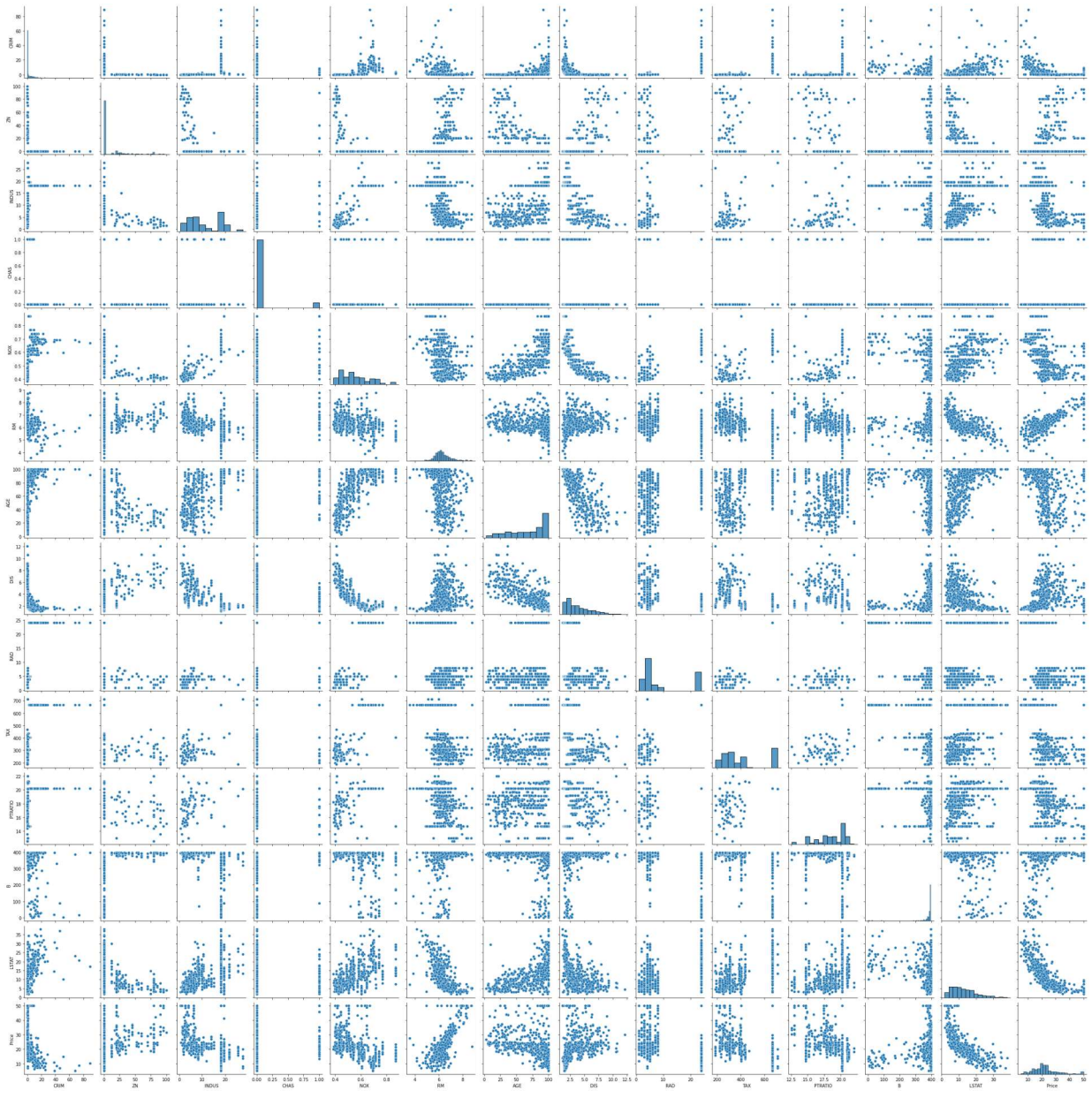
Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.

Pairplot

In [22]:
```
sns.pairplot(df)
```

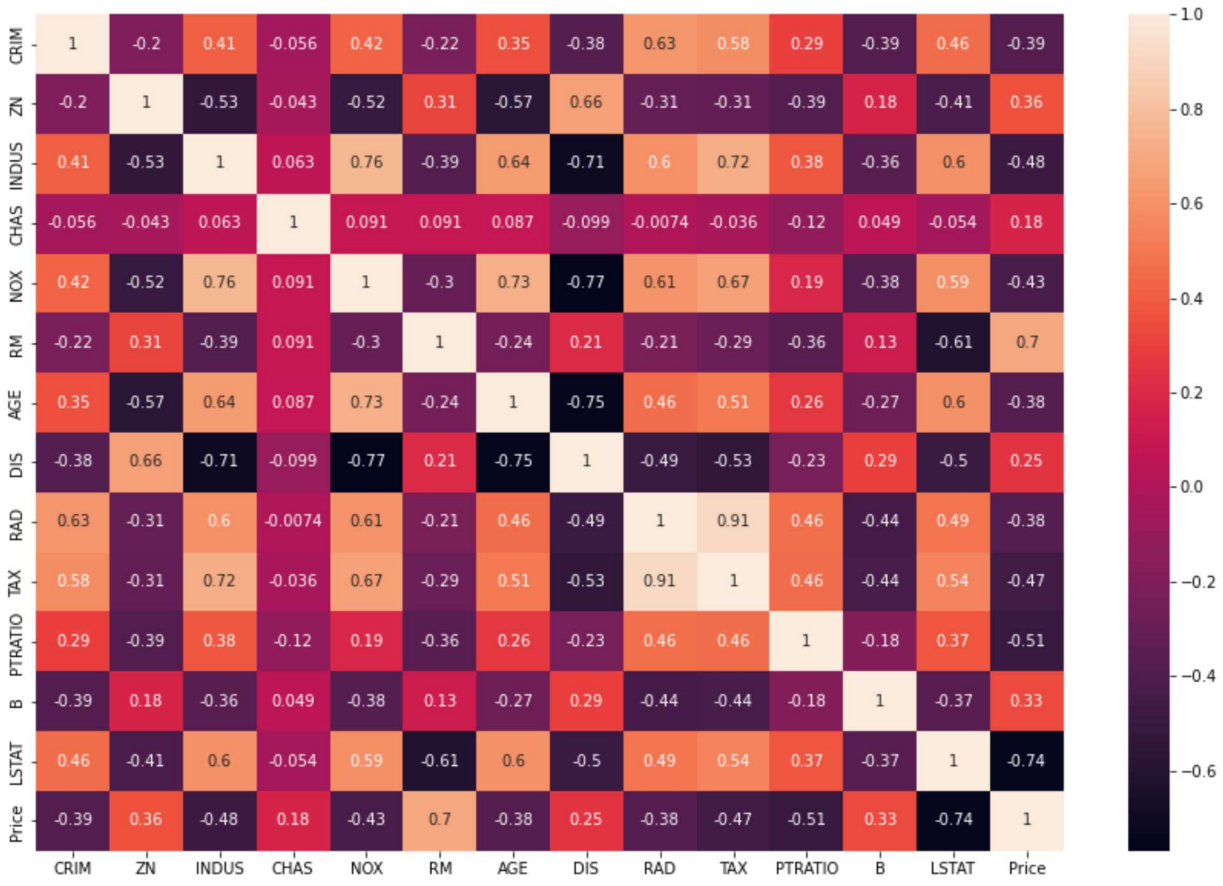
Out[22]:
```
<seaborn.axisgrid.PairGrid at 0x1bcc1fb4e20>
```



Heatmap for correlation

In [23]:
```
plt.figure(figsize = (15,10))
sns.heatmap(df.corr() , annot =True)
```

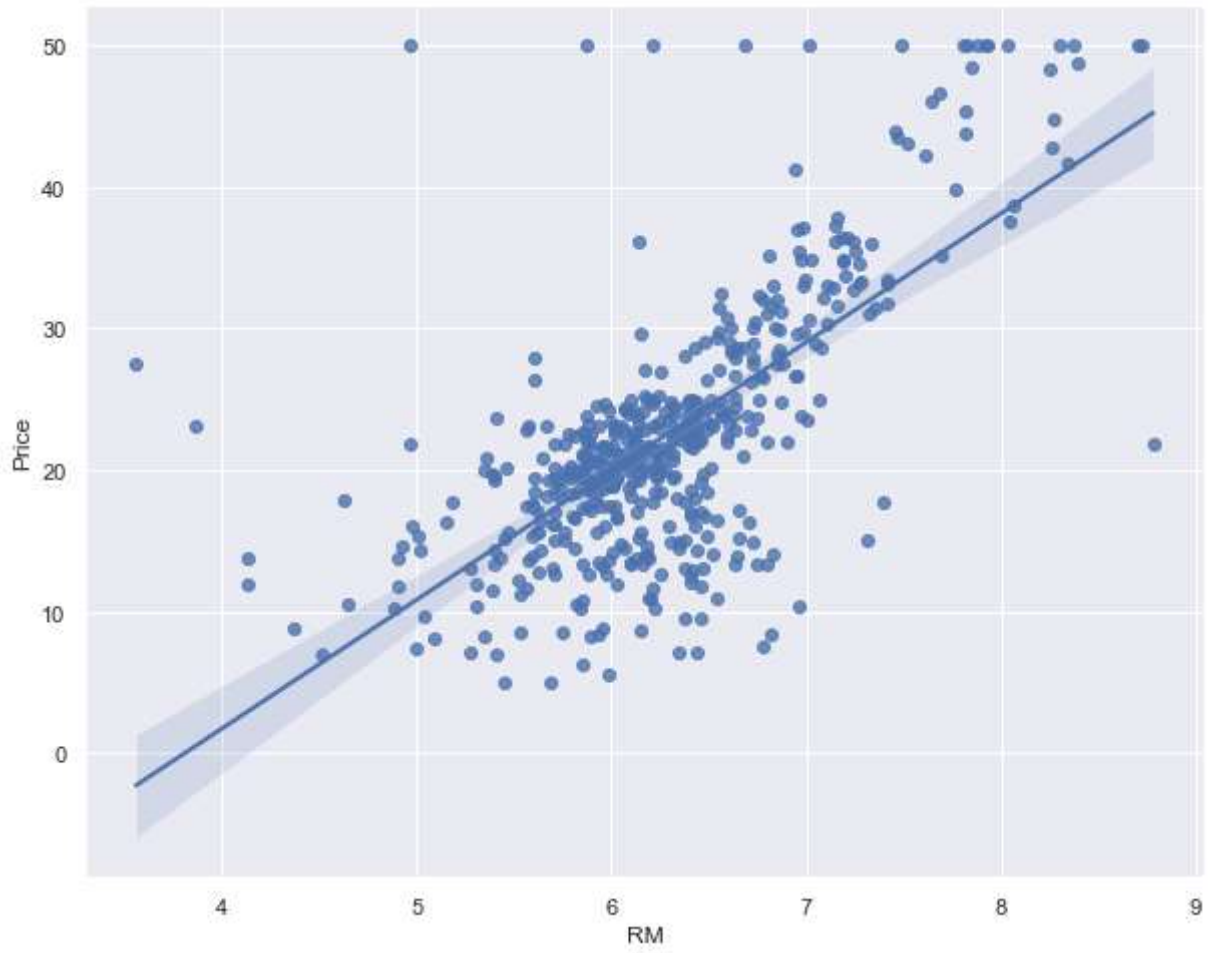
Out[23]: <AxesSubplot:>



Regression Plot for some Features:

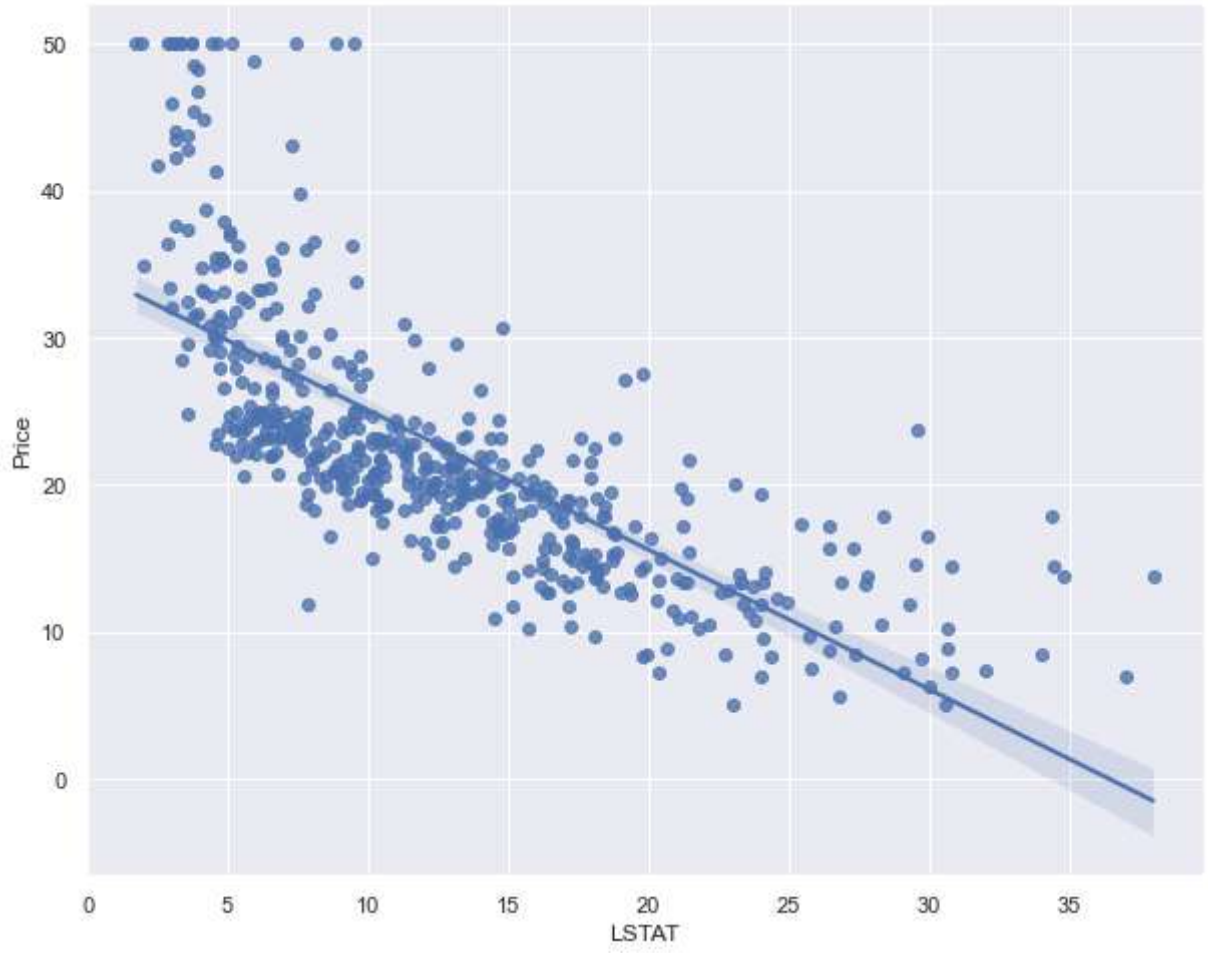
In [90]:
```
sns.regplot(x = "RM" , y ="Price" , data = df)
# This shaded region is ridge lasso region
```

Out[90]: <AxesSubplot:xlabel='RM', ylabel='Price'>



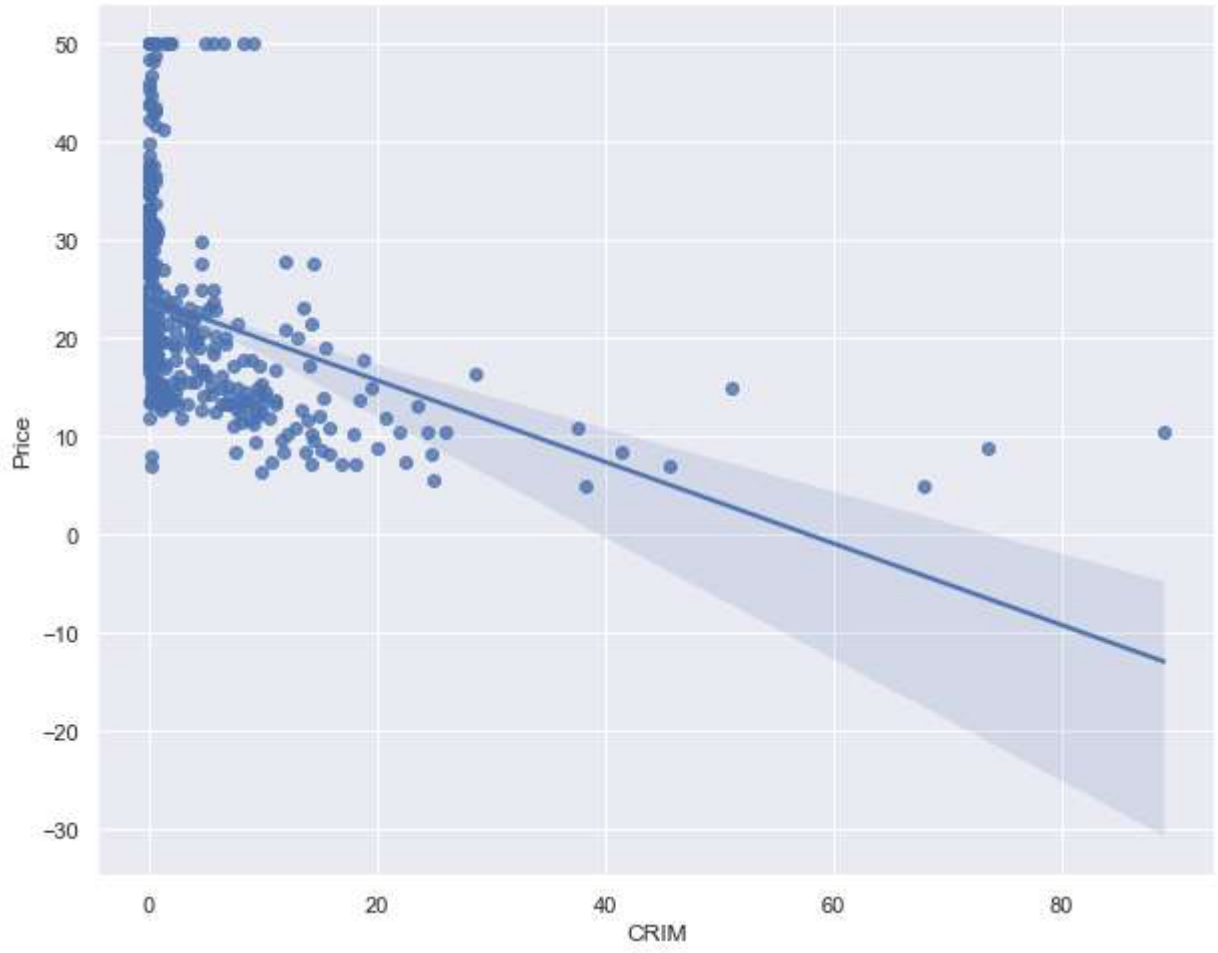
In [91]:
```
sns.regplot(x = "LSTAT" , y ="Price" , data = df)
```

Out[91]: <AxesSubplot:xlabel='LSTAT', ylabel='Price'>



In [92]:
```
sns.regplot(x = "CRIM" , y ="Price" , data = df)
```

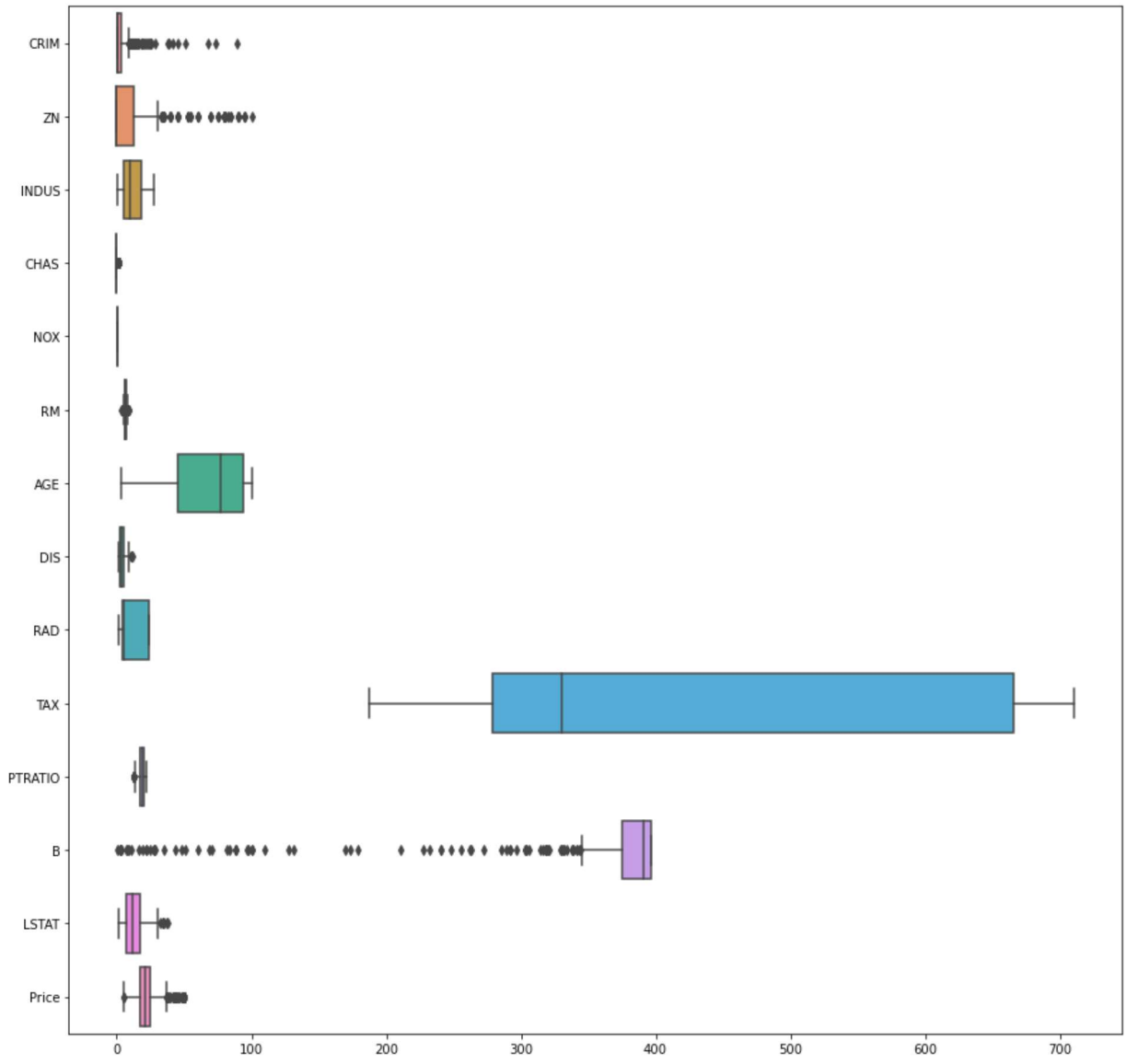
Out[92]:
```
<AxesSubplot:xlabel='CRIM', ylabel='Price'>
```



Outliers in the data

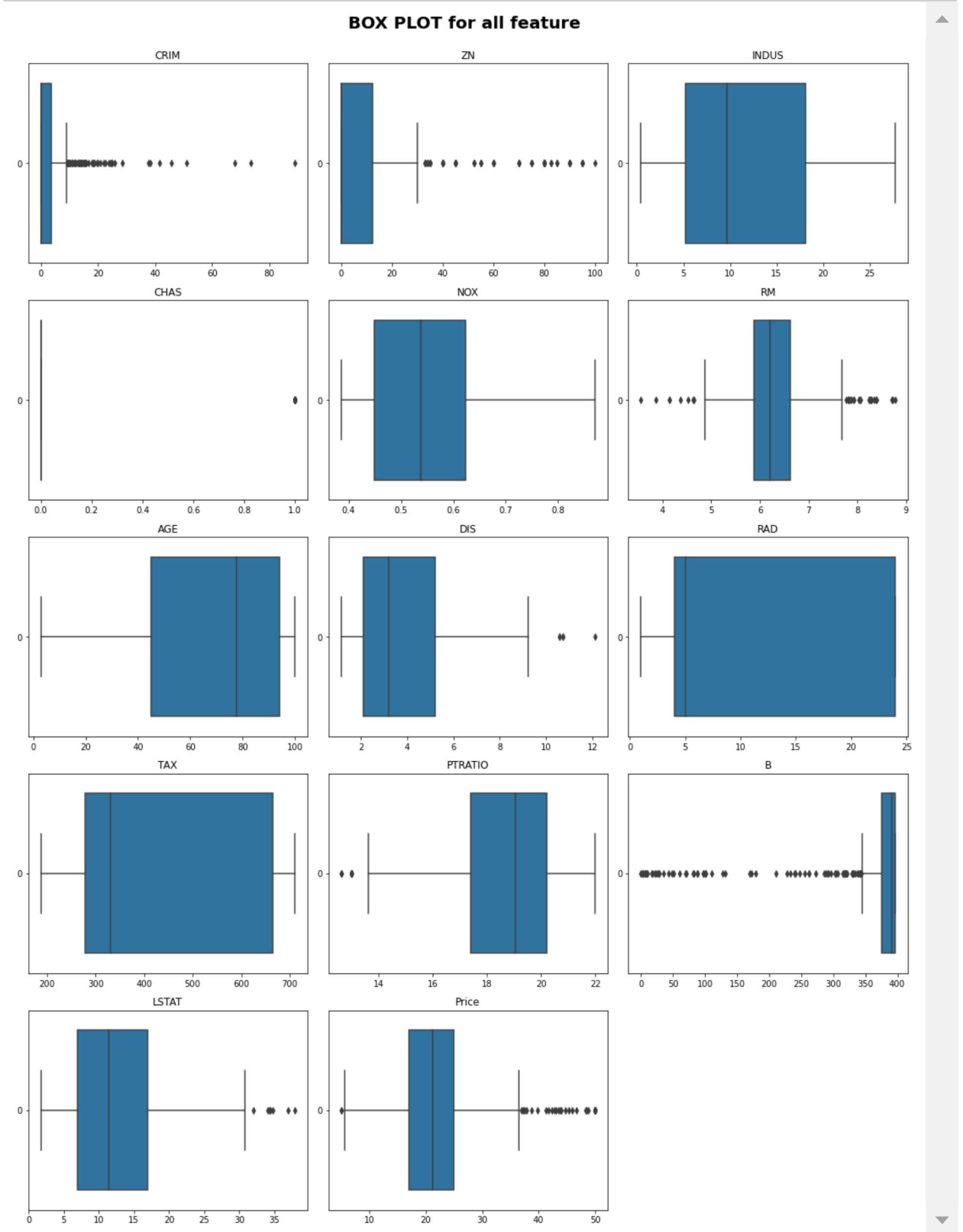
In [27]:
```
plt.figure(figsize=(15,15))
sns.boxplot(data = df , orient = "h")
```

Out[27]: <AxesSubplot:>



In [28]:
```
plt.figure(figsize=(15, 20))
plt.suptitle('BOX PLOT for all feature', fontsize=20, fontweight='bold', alpha=1,

for i in range(0, len(df.columns)):
    plt.subplot(5, 3, i+1)
    sns.boxplot(data=df[df.columns[i]] , orient = 'h' )
    plt.title(label = df.columns[i])
    plt.tight_layout()
```



Observation:

- CRIM , ZN , RM , B , Price feature haev a large number of outliers.
- LSTAT , PTRATIO , DIS , CHAS columns have very less outliers.
- INDUS, NOX , AGE , RAD have no outliers .

TRAINING THE MODEL

Seperating Independent and dependent Features

In [29]:
```
df.head()
```

Out[29]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

In [30]:
```
# indep and dep features
```

In [31]:
```
x= df.iloc[:,:-1]
y=df.iloc[:,-1]
```

In [32]:
```
y  # we usually have a series as dependent fearure and dataframe as independent f
```

Out[32]:
```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
505    11.9
Name: Price, Length: 506, dtype: float64
```

So we have x as independent features as dataframe and y == Price which is dependent Feature and is in series form.

Splitting The Training and Testing Data

In [33]:
```
# we split data in train and test dataset   using  sklearn:  train_test_split
```

In [34]:
```
from sklearn.model_selection import train_test_split
```

In [36]:
```
# now split using this syntax

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_
```

- test size tells how much percent of data will go to test data
- random_state is for randomization
- x_train is the indep train data , y_train is its corresponding training data
- x_test and y_test are data to be tested upon.

Viewing splitted data

In [37]:
```
X_train
```

Out[37]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
478	10.23300	0.0	18.10	0.0	0.614	6.185	96.7	2.1705	24.0	666.0	20.2	379.70	18
26	0.67191	0.0	8.14	0.0	0.538	5.813	90.3	4.6820	4.0	307.0	21.0	376.88	14
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19
492	0.11132	0.0	27.74	0.0	0.609	5.983	83.5	2.1099	4.0	711.0	20.1	396.90	13
108	0.12802	0.0	8.56	0.0	0.520	6.474	97.1	2.4329	5.0	384.0	20.9	395.24	12
...	...	...	...	...	...	...	...	...	...	...	...	...	
106	0.17120	0.0	8.56	0.0	0.520	5.836	91.9	2.2110	5.0	384.0	20.9	395.67	18
270	0.29916	20.0	6.96	0.0	0.464	5.856	42.1	4.4290	3.0	223.0	18.6	388.65	13
348	0.01501	80.0	2.01	0.0	0.435	6.635	29.7	8.3440	4.0	280.0	17.0	390.94	5
435	11.16040	0.0	18.10	0.0	0.740	6.629	94.6	2.1247	24.0	666.0	20.2	109.85	23
102	0.22876	0.0	8.56	0.0	0.520	6.405	85.4	2.7147	5.0	384.0	20.9	70.80	10

339 rows × 13 columns

In [38]:
```
y_train
```

Out[38]:
```
478    14.6
26     16.6
7      27.1
492    20.1
108    19.8
       ...
106    19.5
270    21.1
348    24.5
435    13.4
102    18.6
Name: Price, Length: 339, dtype: float64
```

In [39]:
```
X_test
```

Out[39]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LST
173	0.09178	0.0	4.05	0.0	0.510	6.416	84.1	2.6463	5.0	296.0	16.6	395.50	9.
274	0.05644	40.0	6.41	1.0	0.447	6.758	32.9	4.0776	4.0	254.0	17.6	396.90	3.
491	0.10574	0.0	27.74	0.0	0.609	5.983	98.8	1.8681	4.0	711.0	20.1	390.11	18.
72	0.09164	0.0	10.81	0.0	0.413	6.065	7.8	5.2873	4.0	305.0	19.2	390.91	5.
452	5.09017	0.0	18.10	0.0	0.713	6.297	91.8	2.3682	24.0	666.0	20.2	385.09	17.
...	...	...	...	...	...	...	...	...	...	...	...	...	
110	0.10793	0.0	8.56	0.0	0.520	6.195	54.4	2.7778	5.0	384.0	20.9	393.49	13.
321	0.18159	0.0	7.38	0.0	0.493	6.376	54.3	4.5404	5.0	287.0	19.6	396.90	6.
265	0.76162	20.0	3.97	0.0	0.647	5.560	62.8	1.9865	5.0	264.0	13.0	392.40	10.
29	1.00245	0.0	8.14	0.0	0.538	6.674	87.3	4.2390	4.0	307.0	21.0	380.23	11.
262	0.52014	20.0	3.97	0.0	0.647	8.398	91.5	2.2885	5.0	264.0	13.0	386.86	5.

167 rows × 13 columns

In [40]:
```
y_test
```

Out[40]:
```
173    23.6
274    32.4
491    13.6
72     22.8
452    16.1
       ...
110    21.7
321    23.1
265    22.8
29     21.0
262    48.8
Name: Price, Length: 167, dtype: float64
```

Shape of splitted data

In [41]:
```
print(X_train.shape , y_train.shape , X_test.shape , y_test.shape)
```

```
(339, 13) (339,) (167, 13) (167,)
```

STANDARDIZATION

In [42]:
```
## standardize the dataset
## With standardization we reach the global minima faster

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

In [43]:
```
scaler   # object of StandardScaler()
```

Out[43]:
```
StandardScaler()
```

Fitting the train and test data

In [44]:
```
X_train=scaler.fit_transform(X_train)     #we fit and transform the training data
X_test=scaler.transform(X_test)

# we only  transorm in test and not fit_transorm because to avoid data leakage
```

The fitted Data is:

In [45]:
```
X_train
```

Out[45]:
```
array([[ 0.89624872, -0.51060139,  0.98278223, ...,  0.86442095,
         0.24040357,  0.77155612],
       [-0.34895881, -0.51060139, -0.44867555, ...,  1.22118698,
         0.20852839,  0.32248963],
       [-0.41764058,  0.03413008, -0.48748013, ..., -1.36536677,
         0.43481957,  0.92775316],
       ...,
       [-0.43451148,  2.97567999, -1.32968321, ..., -0.56264319,
         0.36745216, -0.90756208],
       [ 1.01703049, -0.51060139,  0.98278223, ...,  0.86442095,
        -2.80977992,  1.50233514],
       [-0.40667333, -0.51060139, -0.38831288, ...,  1.17659123,
        -3.25117205, -0.26046005]])
```

In [46]:
```
X_test
```

Out[46]:
```
array([[-0.42451319, -0.51060139, -1.03649306, ..., -0.74102621,
         0.41899501, -0.48220406],
       [-0.42911576,  1.2325393 , -0.6973123 , ..., -0.29506866,
         0.43481957, -1.25063772],
       [-0.42269508, -0.51060139,  2.36824941, ...,  0.8198252 ,
         0.35807046,  0.77713459],
       ...,
       [-0.33727525,  0.36096896, -1.04799071, ..., -2.34647337,
         0.38395492, -0.28556314],
       [-0.30591027, -0.51060139, -0.44867555, ...,  1.22118698,
         0.2463943 , -0.07218683],
       [-0.36872487,  0.36096896, -1.04799071, ..., -2.34647337,
         0.32133488, -0.91871901]])
```

Model Training

In [47]:
```
from sklearn.linear_model import LinearRegression
regression=LinearRegression()  # object for linear_reg so it will have all featur
regression  # object of linear regression
```

Out[47]:
```
LinearRegression()
```

To find regression line:

In [48]:
```
regression.fit(X_train,y_train)   # here we not need to transform as already done
```

Out[48]:
```
LinearRegression()
```

Regression Coefficients

In [49]:
```
print(regression.coef_)
```

```
[-0.98858032  0.86793276  0.40502822  0.86183791 -1.90009974  2.80813518
 -0.35866856 -3.04553498  2.03276074 -1.36400909 -2.0825356   1.04125684
 -3.92628626]
```

Intercept

In [50]:
```
print(regression.intercept_)
```

```
22.970796460176988
```

Predicting test

In [51]:
```
# prediction for test data
reg_pred = regression.predict(X_test)
```

The predicted data is as follows:

In [52]:
```
reg_pred
```

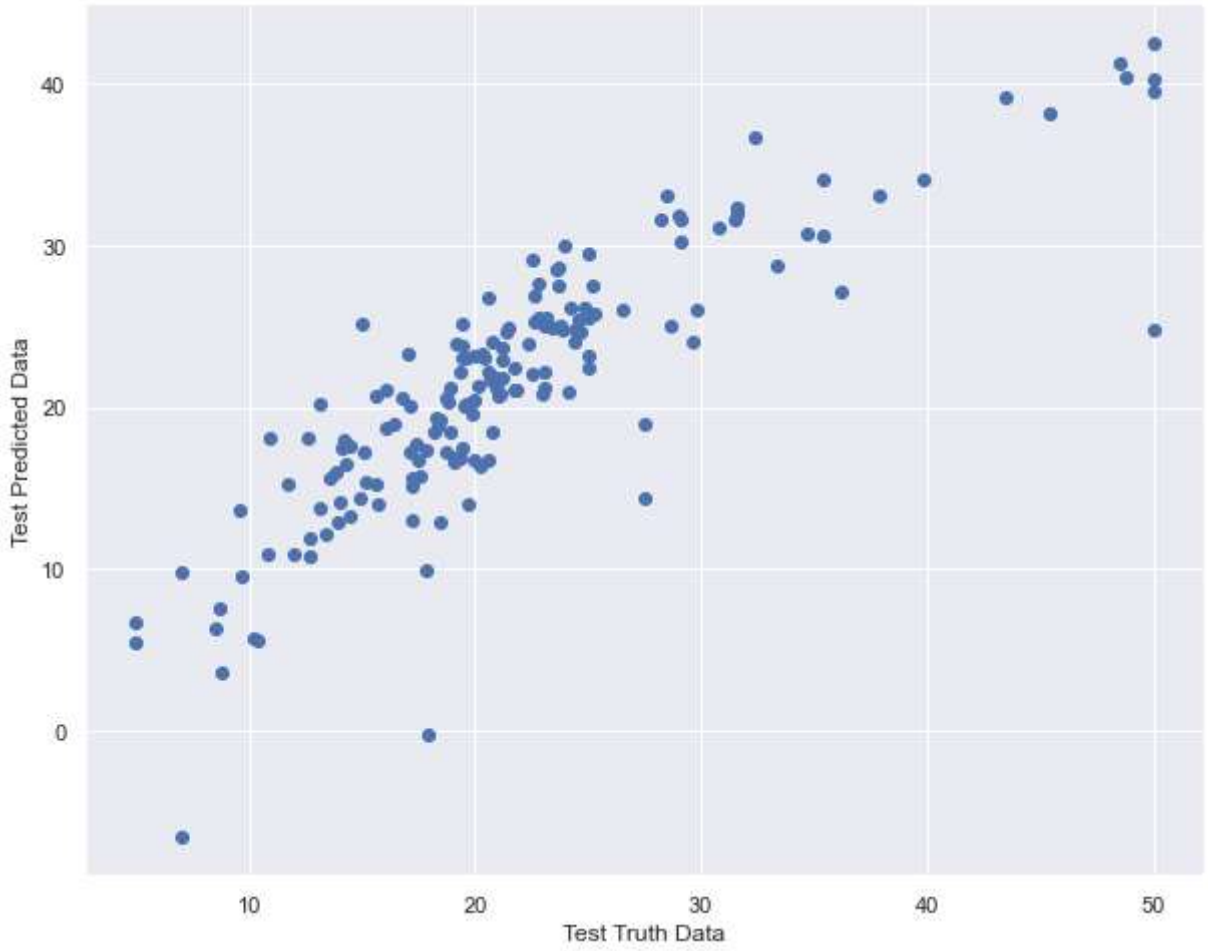
Out[52]:
```
array([28.53469469, 36.6187006 , 15.63751079, 25.5014496 , 18.7096734 ,
       23.16471591, 17.31011035, 14.07736367, 23.01064388, 20.54223482,
       24.91632351, 18.41098052, -6.52079687, 21.83372604, 19.14903064,
       26.0587322 , 20.30232625,  5.74943567, 40.33137811, 17.45791446,
       27.47486665, 30.2170757 , 10.80555625, 23.87721728, 17.99492211,
       16.02608791, 23.268288  , 14.36825207, 22.38116971, 19.3092068 ,
       22.17284576, 25.05925441, 25.13780726, 18.46730198, 16.60405712,
       17.46564046, 30.71367733, 20.05106788, 23.9897768 , 24.94322408,
       13.97945355, 31.64706967, 42.48057206, 17.70042814, 26.92507869,
       17.15897719, 13.68918087, 26.14924245, 20.2782306 , 29.99003492,
       21.21260347, 34.03649185, 15.41837553, 25.95781061, 39.13897274,
       22.96118424, 18.80310558, 33.07865362, 24.74384155, 12.83640958,
       22.41963398, 30.64804979, 31.59567111, 16.34088197, 20.9504304 ,
       16.70145875, 20.23215646, 26.1437865 , 31.12160889, 11.89762768,
       20.45432404, 27.48356359, 10.89034224, 16.77707214, 24.02593714,
        5.44691807, 21.35152331, 41.27267175, 18.13447647,  9.8012101 ,
       21.24024342, 13.02644969, 21.80198374,  9.48201752, 22.99183857,
       31.90465631, 18.95594718, 25.48515032, 29.49687019, 20.07282539,
       25.5616062 ,  5.59584382, 20.18410904, 15.08773299, 14.34562117,
       20.85155407, 24.80149389, -0.19785401, 13.57649004, 15.64401679,
       22.03765773, 24.70314482, 10.86409112, 19.60231067, 23.73429161,
       12.08082177, 18.40997903, 25.4366158 , 20.76506636, 24.68588237,
        7.4995836 , 18.93015665, 21.70801764, 27.14350579, 31.93765208,
       15.19483586, 34.01357428, 12.85763091, 21.06646184, 28.58470042,
       15.77437534, 24.77512495,  3.64655689, 23.91169589, 25.82292925,
       23.03339677, 25.35158335, 33.05655447, 20.65930467, 38.18917361,
       14.04714297, 25.26034469, 17.6138723 , 20.60883766,  9.8525544 ,
       21.06756951, 22.20145587, 32.2920276 , 31.57638342, 15.29265938,
       16.7100235 , 29.10550932, 25.17762329, 16.88159225,  6.32621877,
       26.70210263, 23.3525851 , 17.24168182, 13.22815696, 39.49907507,
       16.53528575, 18.14635902, 25.06620426, 23.70640231, 22.20167772,
       21.22272327, 16.89825921, 23.15518273, 28.69699805,  6.65526482,
       23.98399958, 17.21004545, 21.0574427 , 25.01734597, 27.65461859,
       20.70205823, 40.38214871])
```

Assumptions in linear regression

1. Linear Relationship

Linear relationship: There exists a linear relationship between the independent variable, x, and the dependent variable, y. Here we want linear relationship between y_test which is the actual test data and reg_pred which is the predicted data. Visually it can be check by making a scatter plot between dependent and independent variable

In [89]:
```
plt.scatter(y_test,reg_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
sns.set(rc={'figure.figsize':(10,8)})
```



We can clearly see a **Linear Relation** between the Test and predicted data , which tells our model is fitted correctly and the linearity assumption holds

2. Residuals

Residual = actual - predicted , it is also called Error Residuals as we know are the differences between the true value and the predicted value.

- One of the assumptions of linear regression is that the mean of the residuals should be approx zero as in Normal Dist.
- Also the distribution plot of residuals should be Normal
- The qq plot should be a straight line

In [54]:
```
# Finding Residuals
residuals=y_test-reg_pred
```

In [55]:
```
residuals   # these all are errors
```

Out[55]:
```
173   -4.934695
274   -4.218701
491   -2.037511
72    -2.701450
452   -2.609673
         ...
110    0.642557
321   -1.917346
265   -4.854619
29     0.297942
262    8.417851
Name: Price, Length: 167, dtype: float64
```

Check if mean is Zero:

In [56]:
```
np.mean(residuals)
```

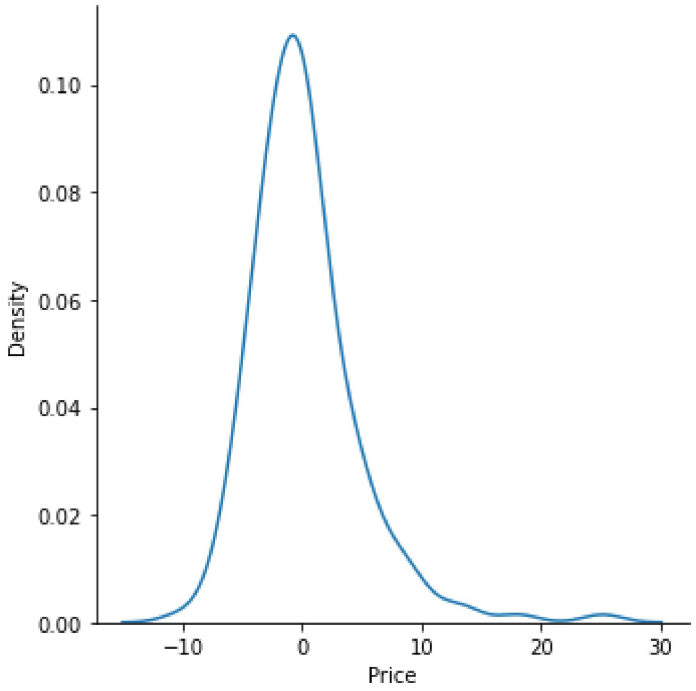
Out[56]:
```
0.20231610858010818
```

Clearly the mean is near to zero , so no problem here

Check for normal dist:

In [57]:
```
# create kde plot for errors
sns.displot(residuals,kind = "kde")
```

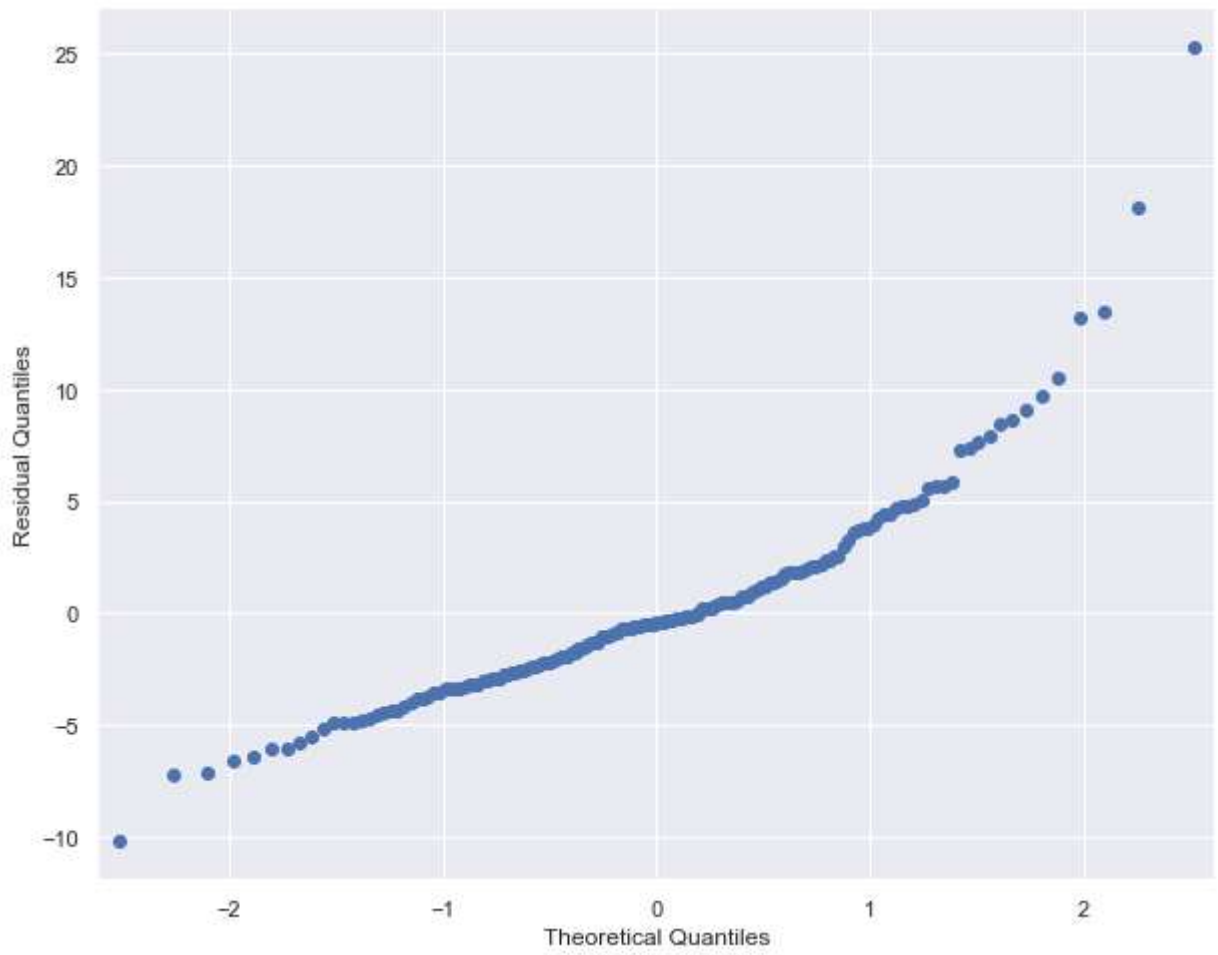
Out[57]: <seaborn.axisgrid.FacetGrid at 0x1bccefb5160>



The residuals are approx normally distributed
this graph is guassian and slighly skewed positive, so can perform ridge lasso

Check qq plot:

In [88]:
```
import statsmodels.api as sm
sm.qqplot(residuals, ylabel = "Residual Quantiles" )
sns.set(rc={'figure.figsize':(10,8)})
```



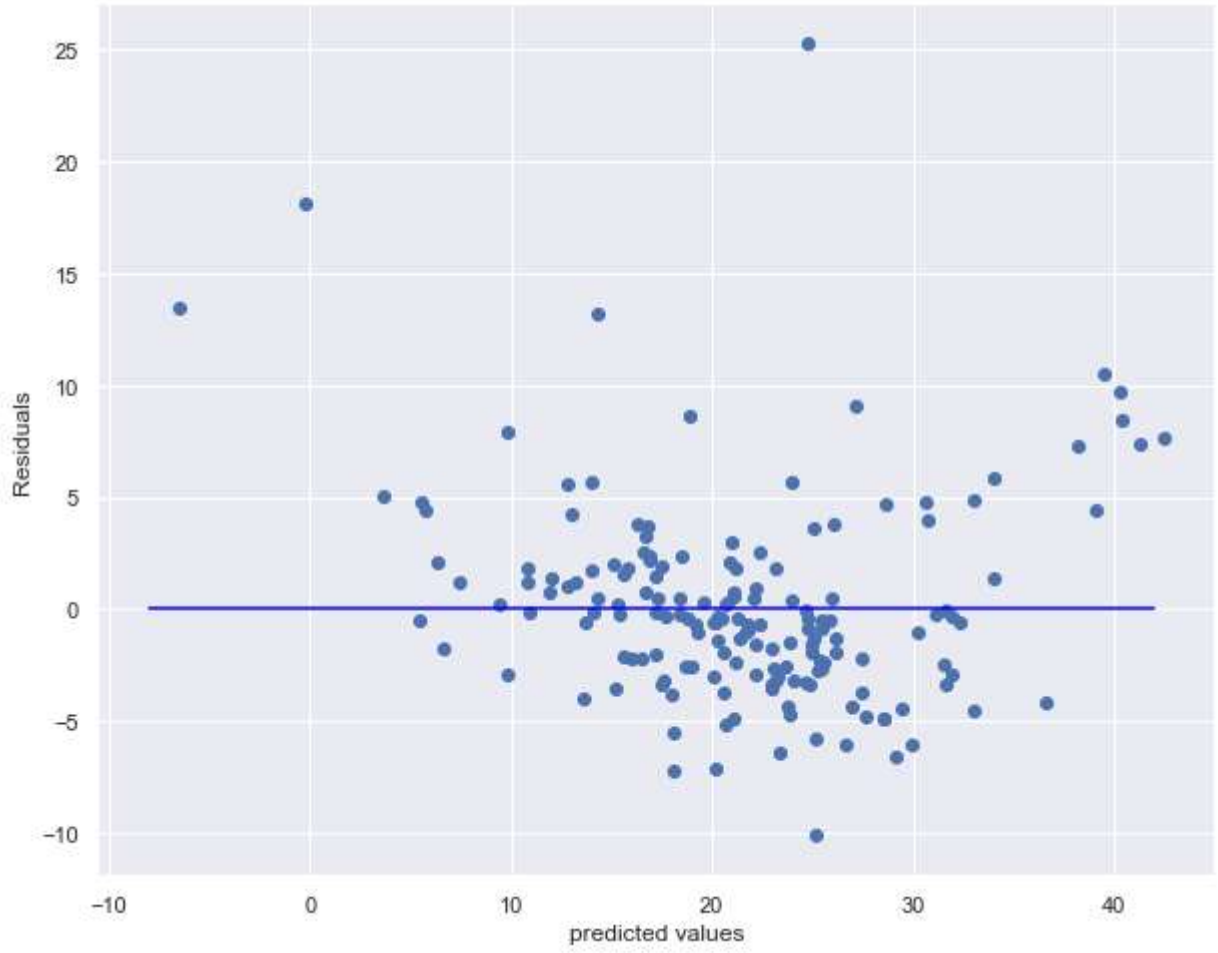
we see an **approx straight line** which tells us this Assumption is satisfied

3. Homoscedasticity

Homoscedasticity means that the residuals have equal or almost equal variance across the regression line. By plotting the error terms with predicted terms we can check that there should not be any pattern in the error term
Visually they should have no trend in the plot and have a uniform like distribution of data

In [87]:
```
sns.set(rc={'figure.figsize':(10,8)})
plt.scatter(reg_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[87]:
```
<AxesSubplot:xlabel='predicted values', ylabel='Residuals'>
```



We can observe there is **No trend or pattern** in the plot so this assumption is satisfied

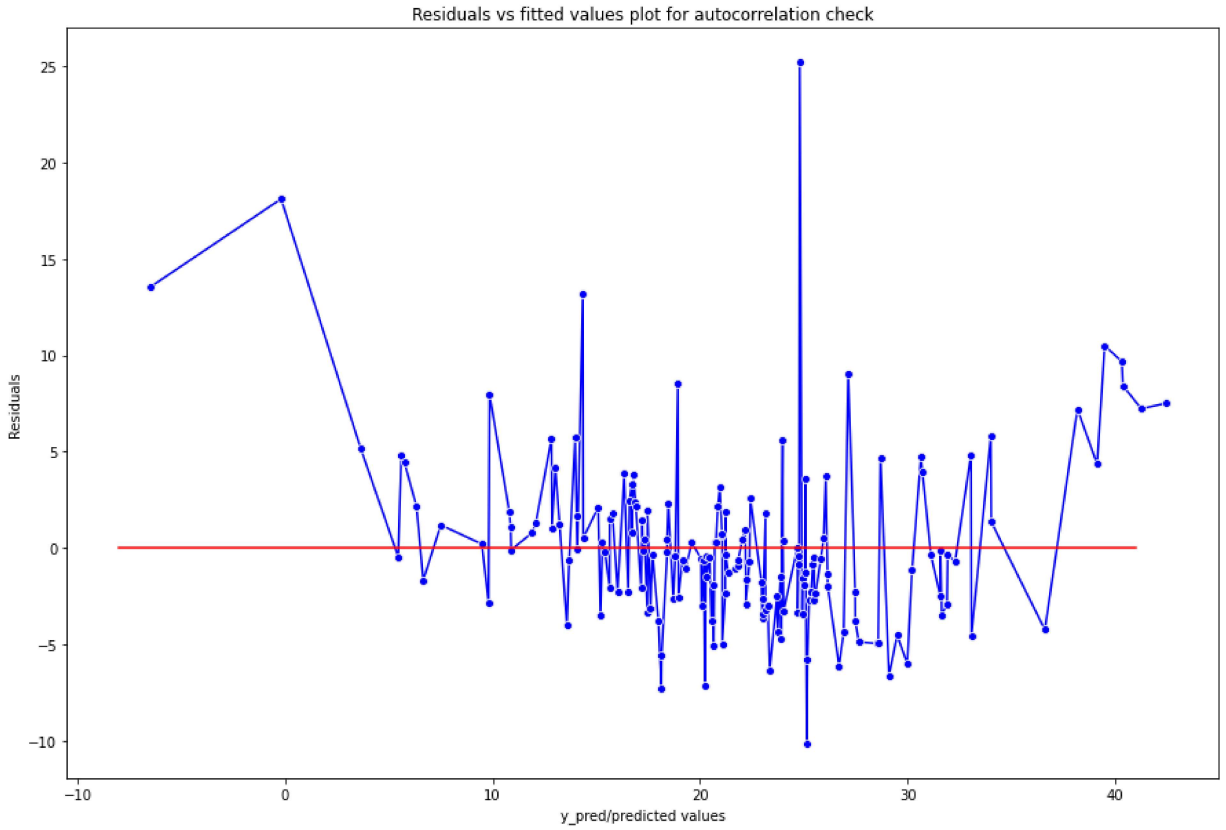
4. No Autocorrelation of residuals

When the residuals are autocorrelated, it means that the current value is dependent of the previous (historic) values and that there is a definite unexplained pattern in the Y variable that shows up in the error terms. Though it is more evident in time series data.

In plain terms autocorrelation takes place when there's a pattern in the rows of the data. This is usual in time series data as there is a pattern of time for eg. Week of the day effect which is a very famous pattern seen in stock markets where people tend to buy stocks more towards the beginning of weekends and tend to sell more on Mondays. There's been great study about this phenomenon and it is still a matter of research as to what actual factors cause this trend.

There should not be autocorrelation in the data so the error terms should not form any pattern.

In [60]:
```
plt.figure(figsize=(15,10))
p = sns.lineplot(reg_pred,residuals,marker='o',color='blue')
plt.xlabel('y_pred/predicted values')
plt.ylabel('Residuals')
p = sns.lineplot([-8,41],[0,0],color='red')
p = plt.title('Residuals vs fitted values plot for autocorrelation check')
```



No pattern indicates no autocorrelation

Performance Metrics For Regression

To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics or evaluation metrics. To evaluate the performance of a classification model, different metrics are used, and some of them are as follows: Following are the popular metrics that are used to evaluate the performance of Regression models.

- Mean Absolute Error(MAE)
- Mean Squared Error(MSE)
- Root Mean Squared Error(RMSE)
- R2 or R Squared Score

- Adjusted R2

Checking MSE , MAE and RMSE

In [61]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(" The mean squared error is : " , mean_squared_error(y_test,reg_pred))
print(" The mean Absolute error is : " ,mean_absolute_error(y_test,reg_pred))
print(" The root mean squared error is : " , np.sqrt(mean_squared_error(y_test,re
```

```
 The mean squared error is :  20.72402343733975
 The mean Absolute error is :  3.1482557548168324
 The root mean squared error is :  4.552364598463061
```

R squared and adjusted R squared

In [62]:
```
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(" The R Squared score is : " , score)
```

```
 The R Squared score is :  0.7261570836552478
```

Therefore R squared is 71.65%

In [63]:
```
#display adjusted R-squared
r2adj_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R2 Score is:  " , r2adj_score)
```

```
The Adjusted R2 Score is:   0.7028893848808571
```

Ridge Regression Model

Ridge regression is a model tuning method that is used to analyse any data that suffers from multicollinearity. This method performs L2 regularization. When the issue of multicollinearity occurs, least-squares are unbiased, and variances are large, this results in predicted values being far away from the actual values.

In [64]:
```
from sklearn.linear_model import Ridge
```

In [65]:
```
ridge=Ridge()
```

In [66]:
```
ridge
```

Out[66]:
```
Ridge()
```

Fitting the Ridge model:

In [68]:
```
ridge.fit(X_train,y_train)
```

Out[68]:
```
Ridge()
```

Ridge Regression Coefficients and Intercept

In [71]:
```
print("The ridge regression Coefficient for indepent features is : " , ridge.coef
print("The ridge Intercept  is : " , ridge.intercept_)
```

```
The ridge regression Coefficient for indepent features is :  [-0.97541551  0.84
608896  0.37564928  0.86738391 -1.86077739  2.81535042
 -0.36108635 -3.00177053  1.95063015 -1.29462251 -2.06972563  1.03867858
 -3.91121554]
The ridge Intercept  is :  22.970796460176988
```

prediction for test data:

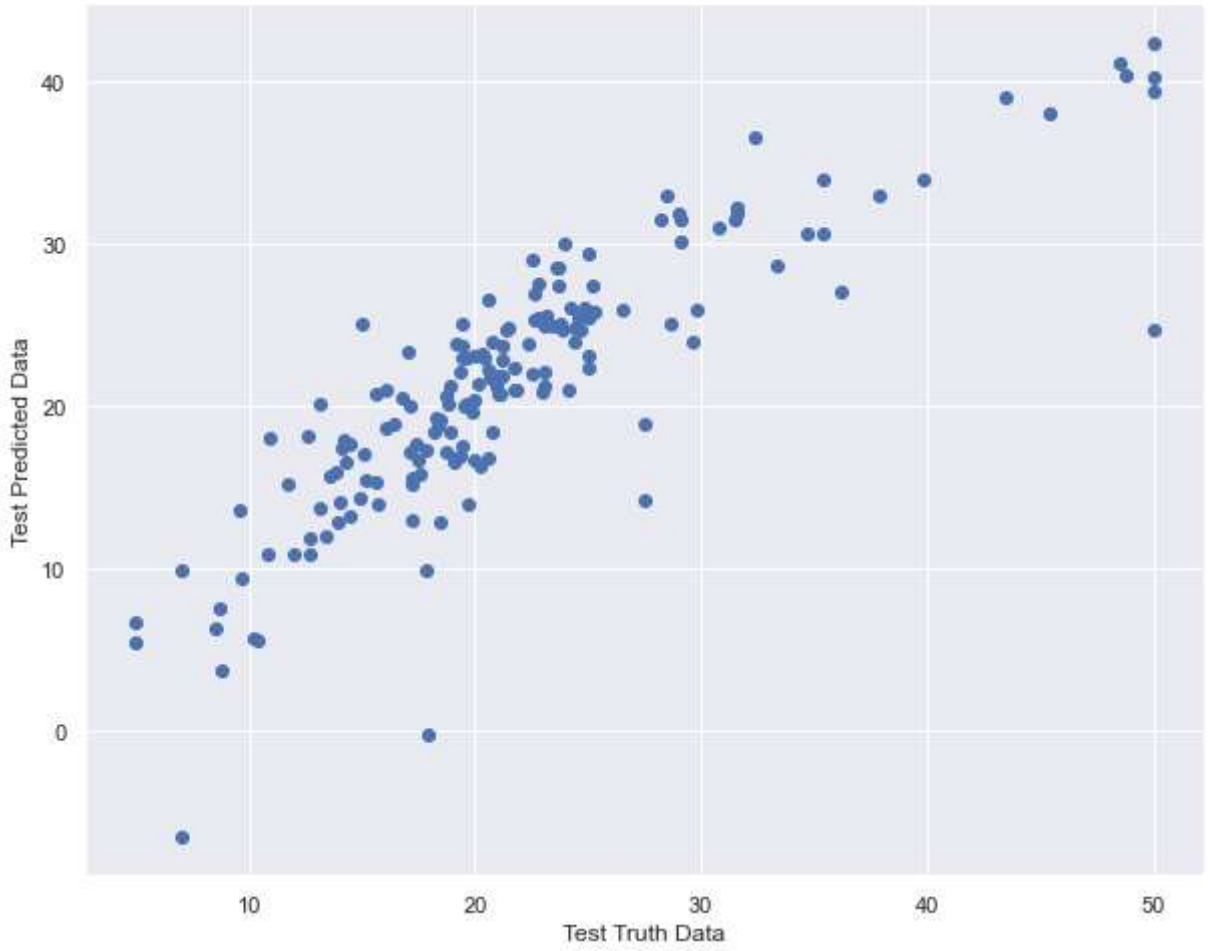
In [72]:
```
ridge_pred = ridge.predict(X_test)
ridge_pred
```

Out[72]:
```
array([28.50742044, 36.56623172, 15.74385228, 25.49107605, 18.70449805,
       23.16555674, 17.3353633 , 14.11553585, 22.94790581, 20.56605905,
       24.89034838, 18.45231813, -6.44592912, 21.85588536, 19.15767075,
       26.02049072, 20.20418625,  5.75334034, 40.26109346, 17.45757103,
       27.43625   , 30.15008183, 10.86542778, 23.8922577 , 17.98002216,
       15.97783016, 23.27084129, 14.39145132, 22.39046559, 19.33913789,
       22.14761798, 25.0633962 , 25.12323651, 18.44886131, 16.56498494,
       17.51980787, 30.71926279, 20.06321173, 24.0051703 , 24.93961773,
       14.01804472, 31.55716108, 42.36411402, 17.74514996, 26.92815913,
       17.13671991, 13.73088718, 26.14522758, 20.23784138, 30.01688815,
       21.23834632, 33.98097943, 15.46847103, 25.98238451, 39.0872846 ,
       22.93270116, 18.79729577, 32.98900997, 24.75121546, 12.88534635,
       22.44620063, 30.6177806 , 31.5496446 , 16.3728183 , 21.07398052,
       16.71642532, 20.21399742, 26.13275151, 31.06390949, 11.90776462,
       20.45341375, 27.424314  , 10.91082555, 16.86277875, 24.02466788,
        5.49260171, 21.36935091, 41.1562018 , 18.12590293,  9.9134628 ,
       21.27916165, 12.98165074, 21.84047835,  9.47562845, 22.9796062 ,
       31.89358365, 18.96118579, 25.49195924, 29.40639573, 20.09806745,
       25.55440575,  5.61779356, 20.21319857, 15.20694421, 14.27823198,
       20.87913507, 24.73423572, -0.20608345, 13.60245079, 15.61960778,
       22.04117361, 24.70915804, 10.84401433, 19.6404862 , 23.76546818,
       12.0612365 , 18.43781737, 25.45763751, 20.79157763, 24.72828941,
        7.53954069, 18.89986796, 21.74415493, 27.09734457, 31.89508098,
       15.20451115, 33.98082344, 12.90997237, 21.08195942, 28.5634719 ,
       15.81062962, 24.79975721,  3.74945983, 23.92684052, 25.82363831,
       23.03894509, 25.37633896, 33.01041733, 20.74845835, 38.12699454,
       13.96268847, 25.34137677, 17.65545489, 20.63383355,  9.92384983,
       21.01822914, 22.20503183, 32.23576441, 31.55073046, 15.34316201,
       16.75621922, 29.10097721, 25.14507288, 16.89567228,  6.37702315,
       26.61598967, 23.43020988, 17.25262089, 13.2787411 , 39.46160064,
       16.5377085 , 18.15230812, 25.09351869, 23.72728145, 22.19137133,
       21.26128221, 16.8779622 , 23.15293077, 28.72634311,  6.73699279,
       23.9593386 , 17.21668437, 21.08725762, 25.0201436 , 27.60453343,
       20.74915073, 40.35440842])
```

Checking Assumptions for Ridge model:

1. Linearity

In [85]:
```
plt.scatter(y_test,ridge_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
sns.set(rc={'figure.figsize':(10,8)})
```



We can see a linear trend here so this assumption is satisfied

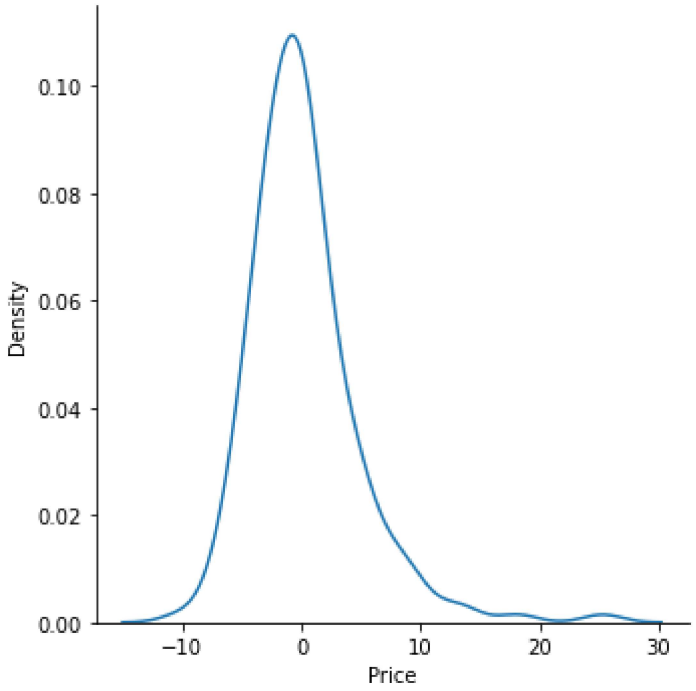
2. Residuals

In [74]:
```
residuals=y_test- ridge_pred
residuals
```

Out[74]:
```
173   -4.907420
274   -4.166232
491   -2.143852
72    -2.691076
452   -2.604498
         ...
110    0.612742
321   -1.920144
265   -4.804533
29     0.250849
262    8.445592
Name: Price, Length: 167, dtype: float64
```

In [75]:
```
sns.displot(residuals,kind="kde")
```

Out[75]:
```
<seaborn.axisgrid.FacetGrid at 0x1bcd15da910>
```

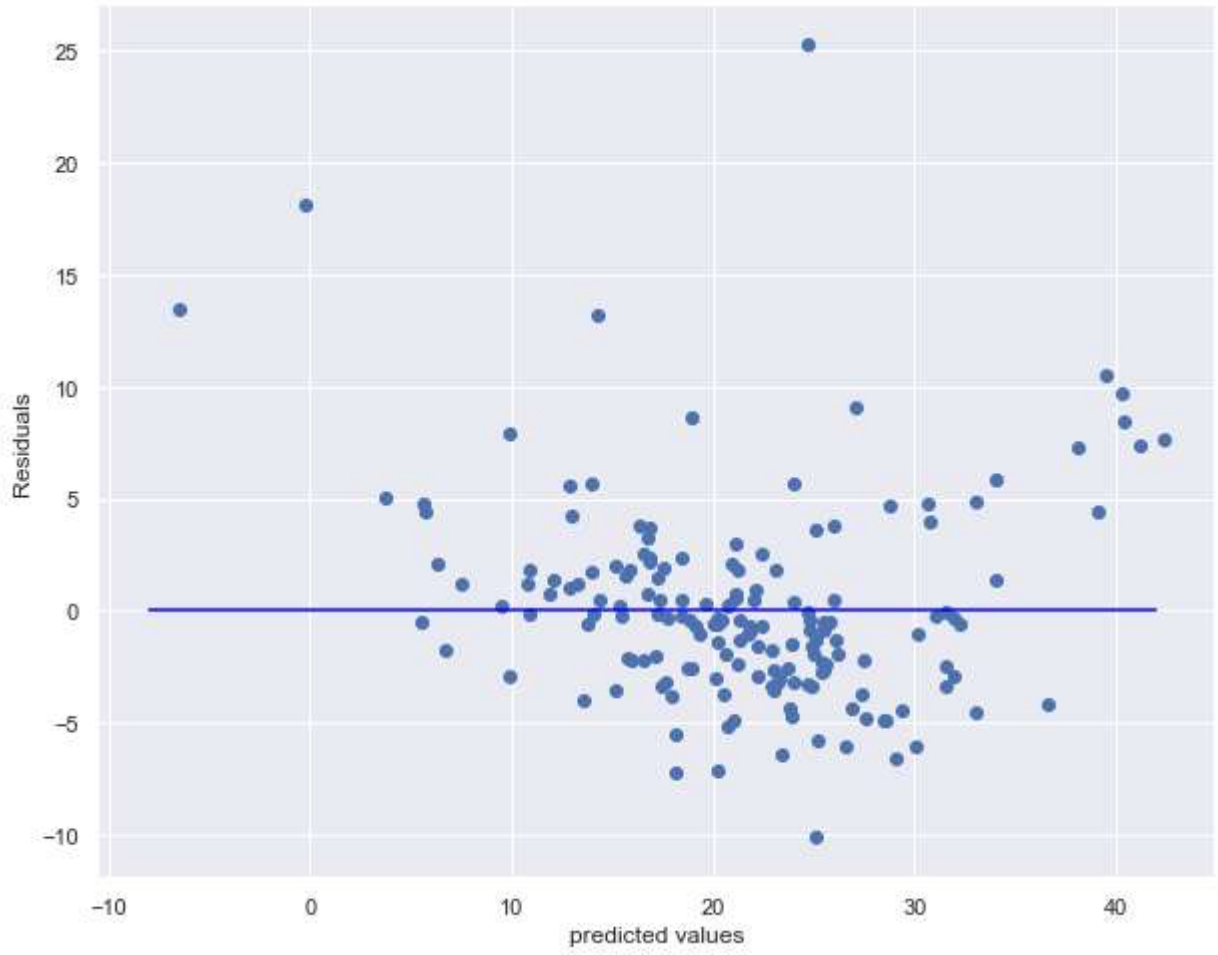


Since the plot for residuals is approximately Normal so we can say that this assumption is also satisfied.

3. Homoscedasticity

In [94]:
```
plt.scatter(ridge_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

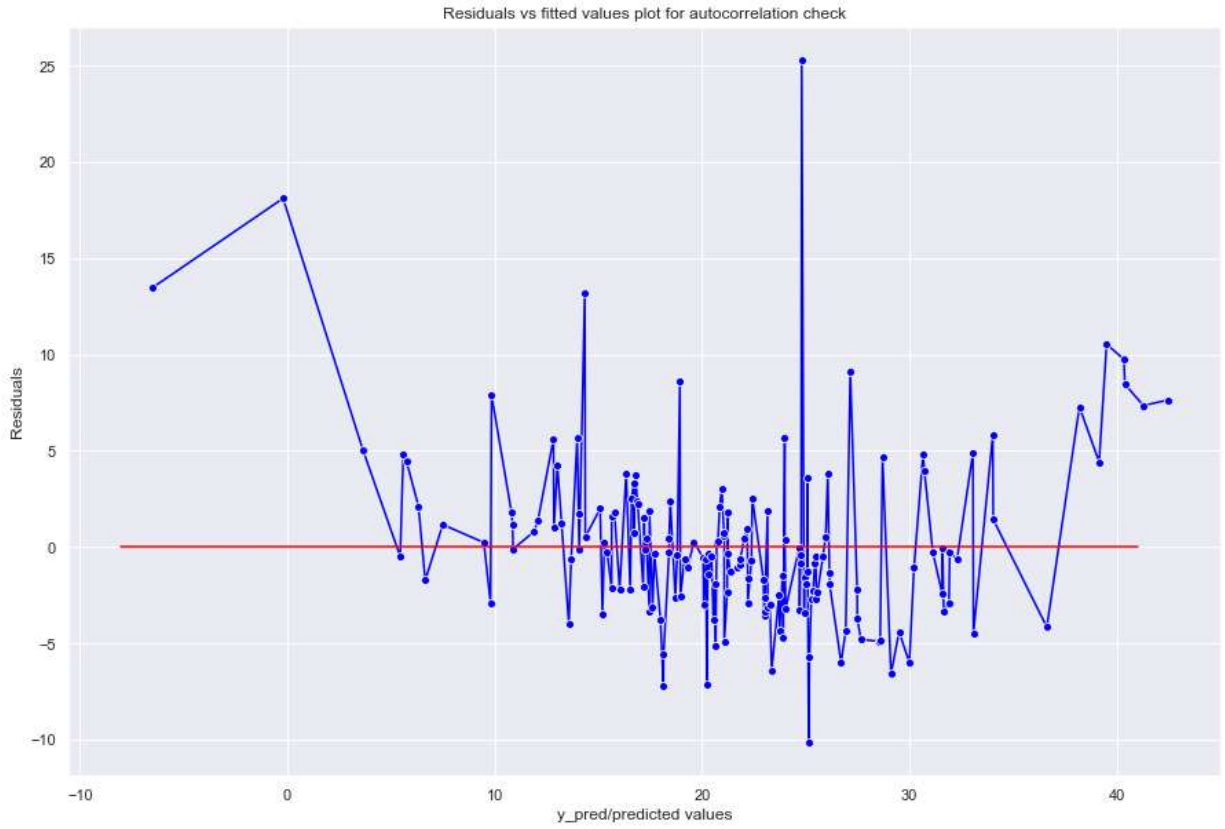
Out[94]:
```
<AxesSubplot:xlabel='predicted values', ylabel='Residuals'>
```



We see no pattern here and data is Uniformly distributed.hence this assumption is Satisfied

4. No Autocorr

In [95]:
```
plt.figure(figsize=(15,10))
p = sns.lineplot(reg_pred,residuals,marker='o',color='blue')
plt.xlabel('y_pred/predicted values')
plt.ylabel('Residuals')
p = sns.lineplot([-8,41],[0,0],color='red')
p = plt.title('Residuals vs fitted values plot for autocorrelation check')
```



There is no pattern so no auto corr. between indep and dep Features.

Performance Metrics

MAE, MSE , RMSE:

In [97]:
```
print(" The mean squared error is : " , mean_squared_error(y_test,ridge_pred))
print(" The mean Absolute error is : " ,mean_absolute_error(y_test,ridge_pred))
print(" The root mean squared error is : " , np.sqrt(mean_squared_error(y_test,ri
```

```
 The mean squared error is :  20.752416320800354
 The mean Absolute error is :  3.1460114626616966
 The root mean squared error is :  4.555482007515819
```

R2 and Adjusted R2

In [98]:
```
score=r2_score(y_test,ridge_pred)
print(" The R Squared score is : " , score)
```

```
 The R Squared score is :  0.7257819060246202
```

In [99]:
```
#display adjusted R-squared
r2adj_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R2 Score is:  " , r2adj_score)
```

```
The Adjusted R2 Score is:   0.702482329412333
```

Lasso Regression Model

Lasso regression is a type of linear regression that uses shrinkage. Shrinkage is where data values are shrunk towards a central point, like the mean. The lasso procedure encourages simple, sparse models (i.e. models with fewer parameters). This particular type of regression is well-suited for models showing high levels of muticollinearity or when you want to automate certain parts of model selection, like variable selection/parameter elimination.

The acronym “LASSO” stands for Least Absolute Shrinkage and Selection Operator.

Lasso regression performs L1 regularization, which adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can result in sparse models with few coefficients; Some coefficients can become zero and eliminated from the model.

In [100]:
```
from sklearn.linear_model import Lasso
lasso = Lasso()
lasso
```

Out[100]:
```
Lasso()
```

In [102]:
```
lasso.fit(X_train,y_train)
```

Out[102]:
```
Lasso()
```

Coefficient and Intercept:

In [103]:
```
print("The ridge regression Coefficient for indepent features is : " , lasso.coef
print("The ridge Intercept  is : " , lasso.intercept_)
```

```
The ridge regression Coefficient for indepent features is :  [-0.          0.
-0.          0.27140271 -0.          2.62932147
 -0.         -0.         -0.         -0.         -1.21106809  0.29872625
 -3.81788375]
The ridge Intercept  is :  22.970796460176988
```

Perdiction

In [104]:
```
lasso_pred = lasso.predict(X_test)
lasso_pred
```

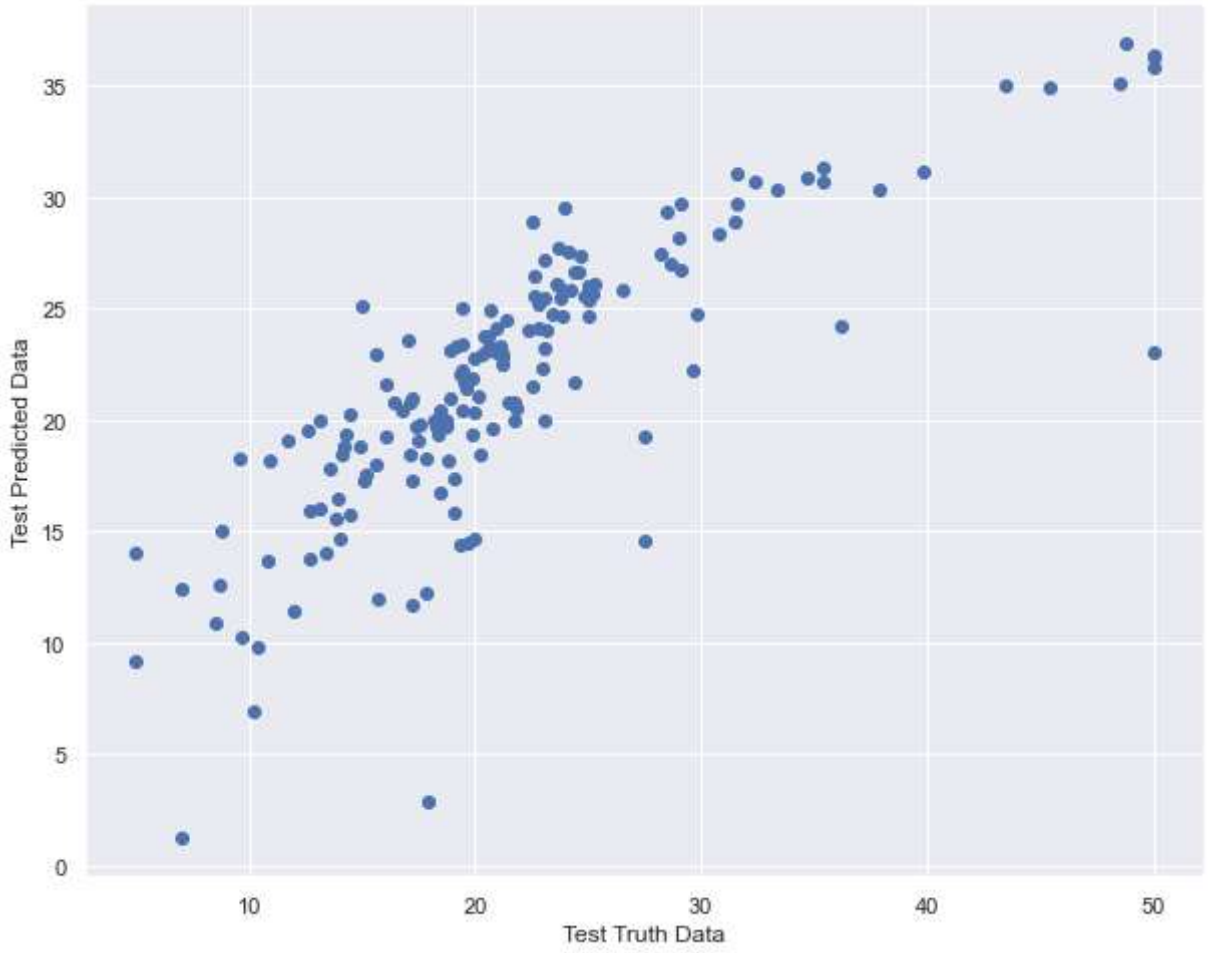
Out[104]:
```
array([26.08015466, 30.7480057 , 17.78164882, 25.25224684, 19.28387274,
       22.81161765, 18.31125182, 14.6359243 , 21.41277818, 20.44276659,
       20.7857368 , 21.00978479,  1.29101416, 22.48591111, 20.4207989 ,
       24.73115299, 18.16643043,  6.95747132, 35.82658816, 18.45664358,
       25.66618031, 26.77096265, 13.79601995, 24.00317031, 18.83677575,
       15.53225538, 22.93567982, 18.81410882, 19.96419904, 19.71394554,
       19.9929271 , 25.48086778, 25.07506471, 19.62299031, 15.87164442,
       20.47826644, 30.90020658, 21.73740698, 21.69357896, 24.78795141,
       14.48946282, 27.49872616, 36.28097645, 19.68302782, 25.54695918,
       17.26691093, 16.01035524, 25.87512519, 19.3705841 , 29.52965183,
       23.10173719, 31.37342903, 17.55332715, 25.82107048, 34.98857199,
       22.91267519, 19.3967501 , 29.34678421, 24.65125376, 16.72971658,
       25.42537393, 30.6751849 , 28.90511192, 18.42571639, 27.56426639,
       14.62706882, 20.02272756, 25.60745002, 28.32959623, 15.91971307,
       20.36020491, 26.04012236, 13.70562148, 23.19186499, 23.2538407 ,
        9.14791655, 21.08680468, 35.13203126, 18.20120981, 12.40579126,
       23.03574753, 11.70030485, 24.10234373, 10.23869501, 22.24788446,
       28.20852115, 20.77401763, 26.01572261, 25.97666619, 20.77471688,
       24.05595237,  9.79658092, 21.55718522, 20.96232324, 14.58941397,
       22.29462592, 23.04513053,  2.87810564, 18.26028545, 17.31405284,
       21.55660947, 24.48282506, 11.46772233, 21.88129799, 25.04349207,
       14.07796126, 19.97841644, 26.61705358, 23.30429098, 27.32736035,
       12.59741065, 19.28050072, 24.94727892, 24.22470232, 29.72519314,
       19.11634391, 31.14895846, 16.43050137, 20.50890111, 27.69026978,
       19.80307948, 26.66386801, 15.01321139, 23.31466084, 26.15446018,
       23.80801526, 27.15999771, 30.37432077, 22.93935948, 34.91159865,
       11.97264266, 26.45153342, 20.25377754, 19.96681079, 12.21677635,
       21.57200937, 23.11587937, 31.05309711, 29.72484228, 18.03669387,
       19.12012649, 28.8679226 , 23.41788443, 14.36679948, 10.91433849,
       23.78530314, 23.58885901, 18.48704182, 15.72569049, 36.3867166 ,
       19.38880373, 19.56932184, 27.03174387, 22.95041998, 22.07807906,
       23.22702436, 17.41528186, 24.66493926, 30.31735718, 13.9998893 ,
       22.25446895, 19.75004517, 20.8350658 , 25.47389672, 24.13577633,
       23.02944605, 36.90324597])
```

Checking Assumptions for Lasso model:

1. Linearity

In [105]:
```
plt.scatter(y_test,lasso_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[105]:
```
Text(0, 0.5, 'Test Predicted Data')
```



We can see a linear trend here so this assumption is satisfied

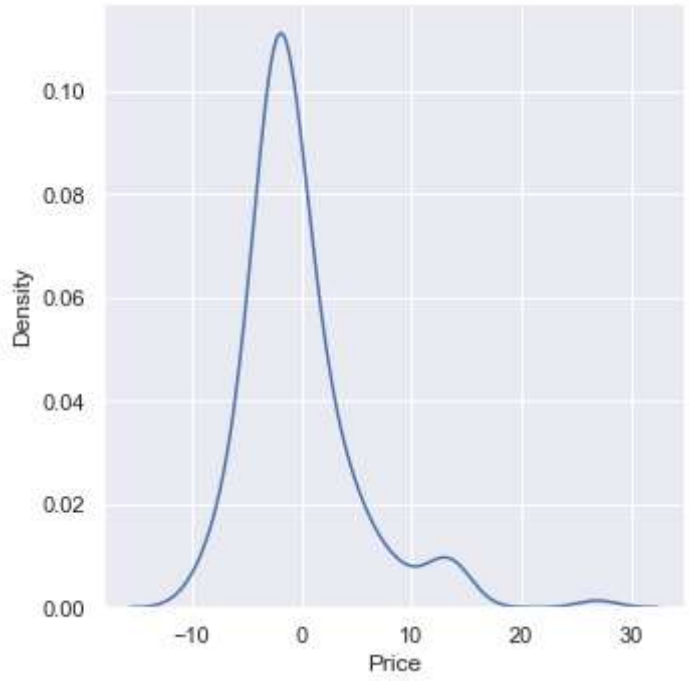
2. Residuals

In [106]:
```
residuals=y_test- lasso_pred
residuals
```

Out[106]:
```
173    -2.480155
274     1.651994
491    -4.181649
72     -2.452247
452    -3.183873
         ...
110     0.864934
321    -2.373897
265    -1.335776
29     -2.029446
262    11.896754
Name: Price, Length: 167, dtype: float64
```

In [107]:
```
sns.displot(residuals,kind="kde")
```

Out[107]:
```
<seaborn.axisgrid.FacetGrid at 0x1bcd3a0efd0>
```

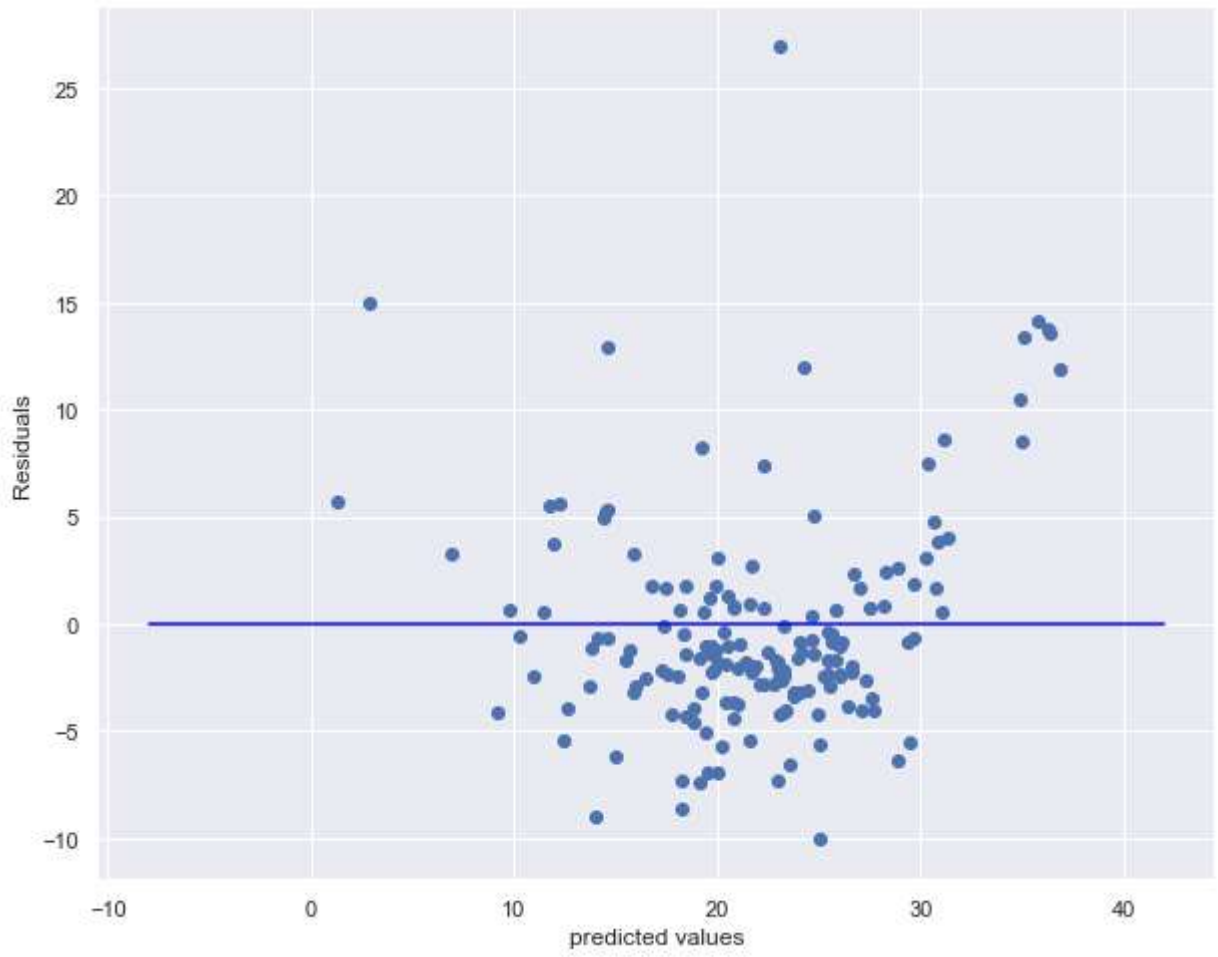


Since the plot for residuals is approximately Normal so we can say that this assumption is also satisfied.

3. Homoscedasticity

In [108]:
```
plt.scatter(lasso_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

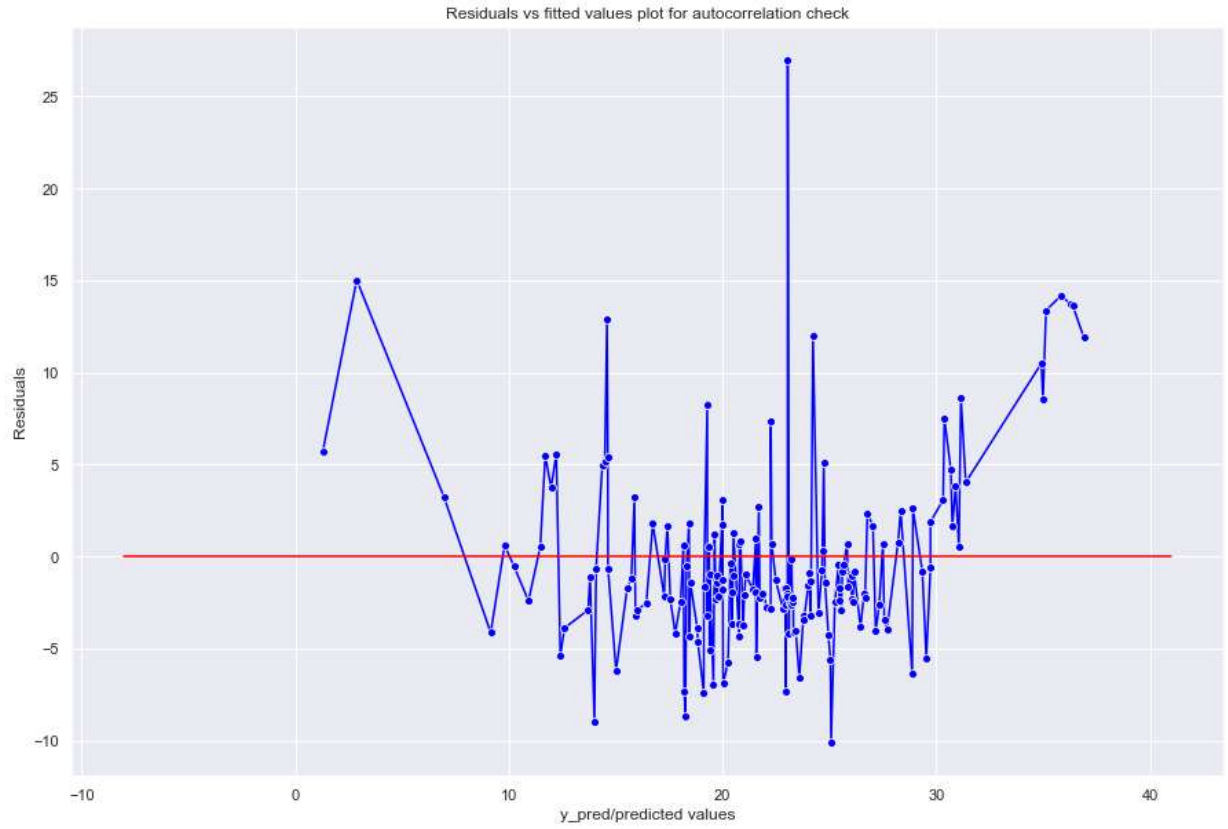
Out[108]:
```
<AxesSubplot:xlabel='predicted values', ylabel='Residuals'>
```



We see no pattern here and data is Uniformly distributed.hence this assumption is Satisfied

4. No Autocorr

In [109]:
```
plt.figure(figsize=(15,10))
p = sns.lineplot(lasso_pred,residuals,marker='o',color='blue')
plt.xlabel('y_pred/predicted values')
plt.ylabel('Residuals')
p = sns.lineplot([-8,41],[0,0],color='red')
p = plt.title('Residuals vs fitted values plot for autocorrelation check')
```



There is no pattern so no auto corr. between indep and dep Features.

Performance Metrics

MAE, MSE , RMSE:

In [110]:
```
print(" The mean squared error is : " , mean_squared_error(y_test,lasso_pred))
print(" The mean Absolute error is : " ,mean_absolute_error(y_test,lasso_pred))
print(" The root mean squared error is : " , np.sqrt(mean_squared_error(y_test,la
```

```
 The mean squared error is :  26.16637721498099
 The mean Absolute error is :  3.6464026430077423
 The root mean squared error is :  5.11530812512609
```

R2 and Adjusted R2

In [111]:
```
score=r2_score(y_test,lasso_pred)
print(" The R Squared score is : " , score)
```

```
 The R Squared score is :  0.6542429577734992
```

In [112]:
```
#display adjusted R-squared
r2adj_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R2 Score is:  " , r2adj_score)
```

```
The Adjusted R2 Score is:   0.6248649084339926
```

The adjusted R2 Score indicate this is a not so good model

ElasticNet Regression Model

Elastic net linear regression uses the penalties from both the lasso and ridge techniques to regularize regression models. The technique combines both the lasso and ridge regression methods by learning from their shortcomings to improve the regularization of statistical models. Elastic net is a popular type of regularized linear regression that combines two popular penalties, specifically the L1 and L2 penalty functions.

In [113]:
```
from sklearn.linear_model import ElasticNet
elsc = ElasticNet()
elsc
```

Out[113]:
```
ElasticNet()
```

Fitting the Ridge model:

In [114]:
```
elsc.fit(X_train,y_train)
```

Out[114]:
```
ElasticNet()
```

ElasticNet Regression Coefficients and Intercept

In [115]:
```
print("The ridge regression Coefficient for indepent features is : " , elsc.coef_
print("The ridge Intercept  is : " , elsc.intercept_)
```

```
The ridge regression Coefficient for indepent features is :  [-0.36520114  0.
-0.14336748  0.63145824 -0.25193148  2.34999448
 -0.         -0.         -0.         -0.25649969 -1.23951556  0.56384945
 -2.56053213]
The ridge Intercept  is :  22.970796460176988
```

prediction for test data:

In [125]:
```
elsc_pred = elsc.predict(X_test)
elsc_pred
```

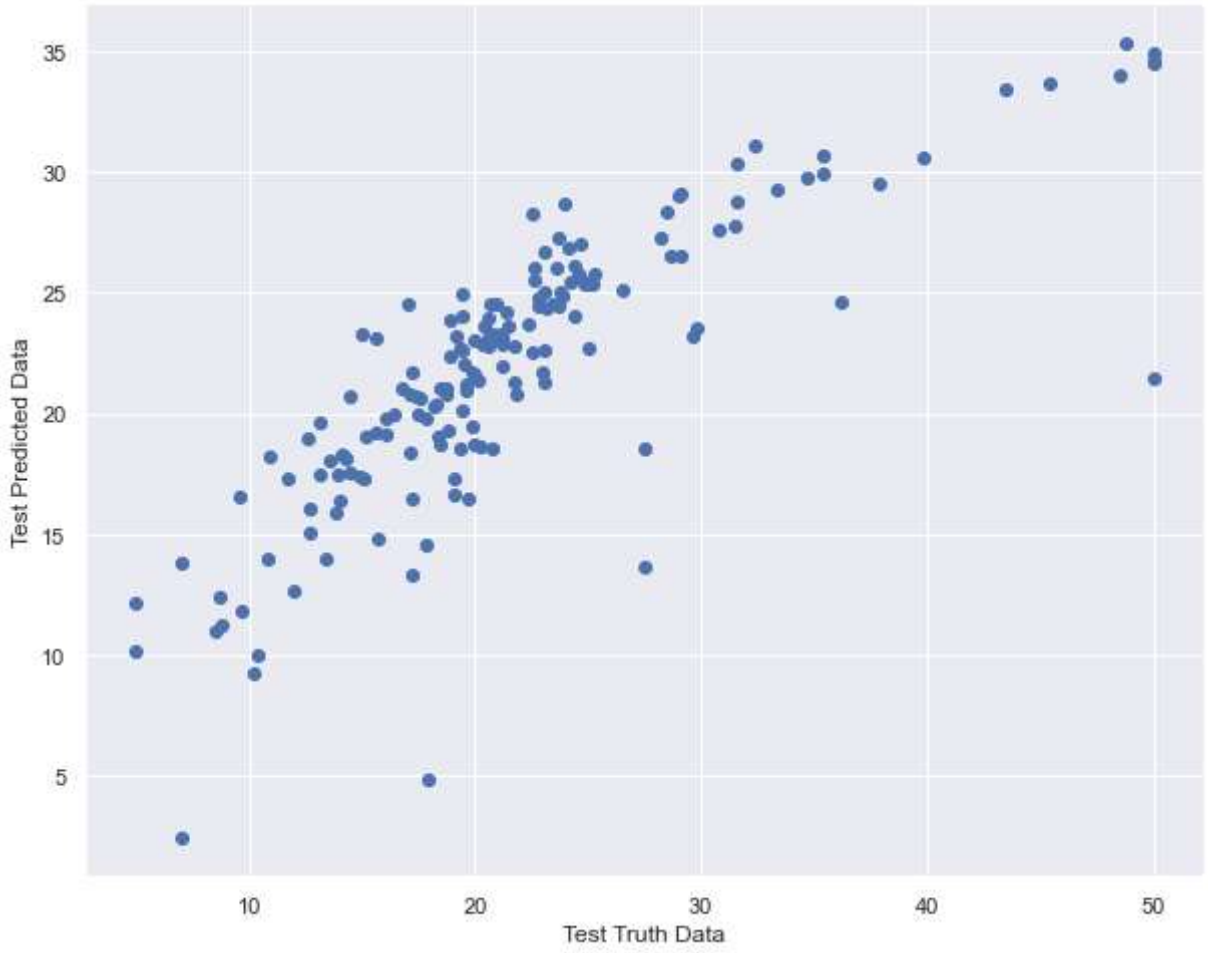
Out[125]:
```
array([26.04802695, 31.11448131, 18.09845158, 24.74715491, 19.13029713,
       23.07195028, 19.8492127 , 16.42921582, 20.98280883, 21.03040905,
       23.59247585, 22.4067143 ,  2.50342106, 22.86968897, 21.05836477,
       23.53088819, 19.32942155,  9.24659633, 34.51755093, 18.33111982,
       25.39963891, 26.53220506, 16.04212388, 23.68595117, 18.22309609,
       15.9070075 , 22.91791506, 17.40135861, 22.80881602, 20.34960072,
       21.28107265, 25.0664737 , 23.29041734, 18.52289666, 16.68946719,
       20.17099878, 29.78000437, 22.08911412, 24.00624402, 24.52109601,
       16.51539744, 27.25142517, 34.8940966 , 20.75229792, 25.54944362,
       17.27877681, 17.51067948, 25.422475  , 19.45141801, 28.72445431,
       23.85816391, 30.64335445, 19.05778782, 25.10137208, 33.43673587,
       21.9368327 , 19.10068361, 28.38705767, 24.91075492, 18.68821158,
       25.41735754, 29.96236233, 27.77368373, 18.66077461, 26.83456776,
       18.72984267, 19.66634919, 25.37569386, 27.64862833, 15.09326887,
       21.6230625 , 24.42218348, 14.00935207, 22.80340884, 23.31057037,
       10.15388121, 21.41270277, 33.98445117, 18.23197774, 13.83630127,
       23.23840504, 13.33915878, 24.55297788, 11.80396525, 22.6039322 ,
       29.04777925, 19.93864988, 25.40796591, 25.4253774 , 20.79626634,
       24.35937771,  9.98031645, 21.1883148 , 21.7010251 , 13.64158366,
       21.70329066, 21.48780024,  4.89921219, 16.60651403, 16.48691364,
       22.52662793, 24.23810699, 12.67234464, 21.73288769, 24.94869622,
       14.02785155, 20.34560161, 25.81816846, 23.27665603, 26.98968083,
       12.4461064 , 18.53919342, 24.57036836, 24.5991159 , 28.78917289,
       17.35695925, 30.55890904, 17.49669771, 20.81635985, 27.26668735,
       20.67056474, 26.10145269, 11.29182557, 23.22360335, 25.76790464,
       23.6220014 , 26.68354582, 29.53505955, 23.09099441, 33.69060315,
       14.86234562, 26.00700282, 20.72314768, 21.04261631, 14.58583453,
       19.80159048, 23.16541552, 30.31837307, 29.06727633, 19.21664987,
       19.96431131, 28.26947089, 24.04394758, 18.60022435, 10.97392042,
       23.98834146, 24.56611841, 18.41001674, 17.5657926 , 34.56534242,
       18.12003269, 19.01050052, 26.55906015, 23.10161055, 22.69122071,
       22.66142713, 17.29393495, 22.68213782, 29.22756893, 12.14001806,
       23.24525465, 20.77586134, 21.27148575, 25.02037796, 24.44501814,
       23.10652629, 35.30973171])
```

Checking Assumptions for ElasticNet model:

1. Linearity

In [126]:
```
plt.scatter(y_test,elsc_pred)
plt.xlabel("Test Truth Data")
plt.ylabel("Test Predicted Data")
```

Out[126]:
```
Text(0, 0.5, 'Test Predicted Data')
```



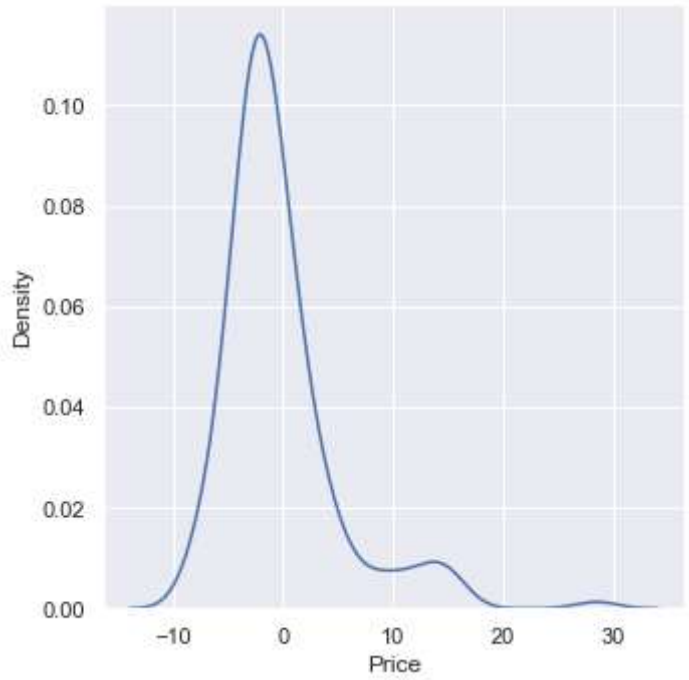
2. Residuals

In [127]:
```
residuals=y_test- elsc_pred
residuals
```

Out[127]:
```
173    -2.448027
274     1.285519
491    -4.498452
72     -1.947155
452    -3.030297
         ...
110     0.428514
321    -1.920378
265    -1.645018
29     -2.106526
262    13.490268
Name: Price, Length: 167, dtype: float64
```

In [128]:
```
sns.displot(residuals,kind="kde")
```

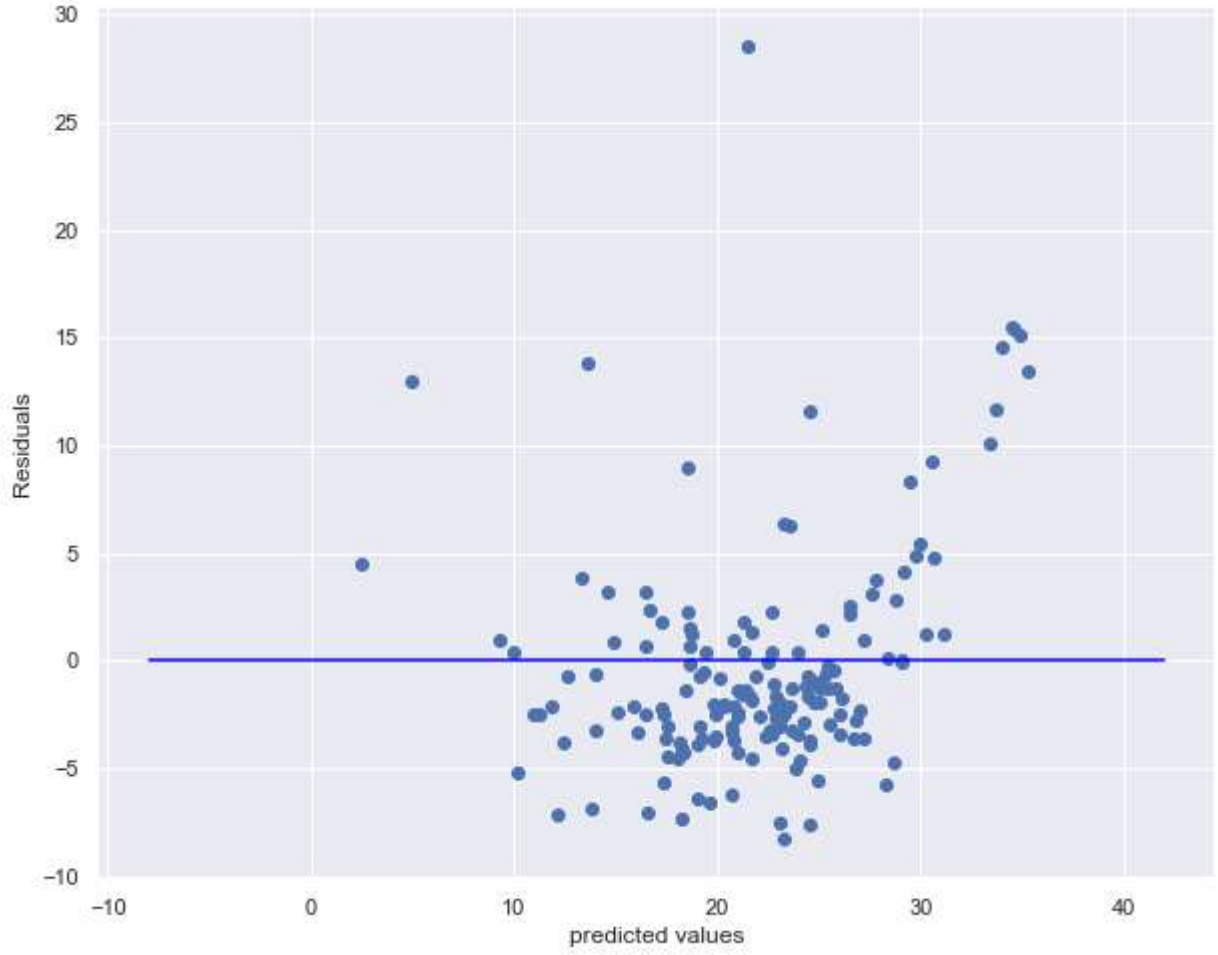
Out[128]:
```
<seaborn.axisgrid.FacetGrid at 0x1bcd5cd6550>
```



3. Homoscedasticity

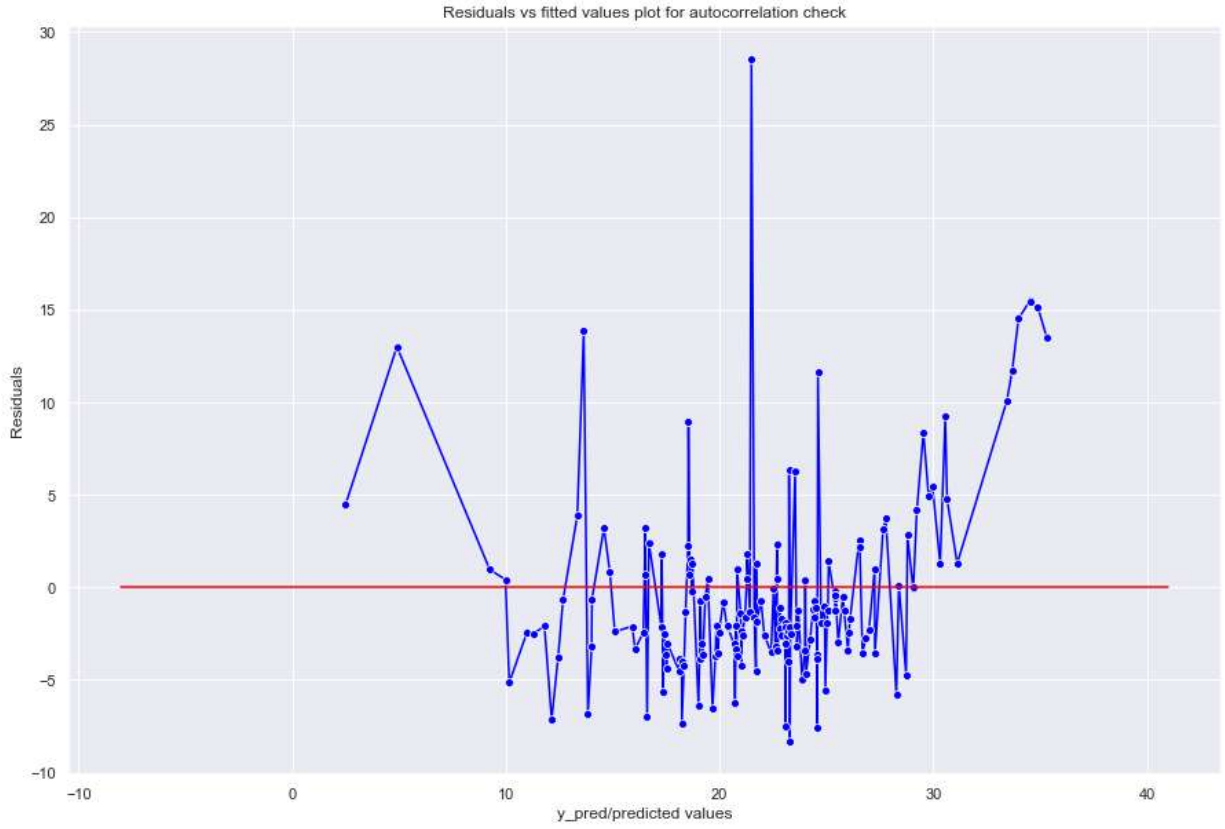
In [129]:
```
plt.scatter(elsc_pred,residuals)
plt.xlabel('predicted values')
plt.ylabel('Residuals')
sns.lineplot([-8,42],[0,0],color='blue')
```

Out[129]: <AxesSubplot:xlabel='predicted values', ylabel='Residuals'>



4. No Autocorr

In [130]:
```
plt.figure(figsize=(15,10))
p = sns.lineplot(elsc_pred,residuals,marker='o',color='blue')
plt.xlabel('y_pred/predicted values')
plt.ylabel('Residuals')
p = sns.lineplot([-8,41],[0,0],color='red')
p = plt.title('Residuals vs fitted values plot for autocorrelation check')
```



There is no pattern so no auto corr. between indep and dep Features.

Performance Metrics

MAE, MSE , RMSE:

In [131]:
```
print(" The mean squared error is : " , mean_squared_error(y_test,elsc_pred))
print(" The mean Absolute error is : " ,mean_absolute_error(y_test,elsc_pred))
print(" The root mean squared error is : " , np.sqrt(mean_squared_error(y_test,el
```

```
 The mean squared error is :  27.140175406489984
 The mean Absolute error is :  3.627745135070299
 The root mean squared error is :  5.209623345932984
```

R2 and Adiusted R2

In [132]:
```
score=r2_score(y_test,elsc_pred)
print(" The R Squared score is : " , score)
```

```
 The R Squared score is :  0.641375391902405
```

In [133]:
```
#display adjusted R-squared
r2adj_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
print("The Adjusted R2 Score is:  " , r2adj_score)
```

```
The Adjusted R2 Score is:   0.610904019972544
```

```
We can see that the adjusted R squared score is low so this model is also not
recommended.
```

THANK YOU

-performed by: Ankit Dubey

In [ ]: