Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import matplotlib.pyplot as plt
from seaborn_qqplot import pplot

%matplotlib inline

warnings.filterwarnings('ignore')
```

Importing dataset

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

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

Checking loaded data object

Features of the dataset

In [4]:
```
boston.feature_names
```

Out[4]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS',
'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

Description of the dataset

In [5]:
```
boston.DESCR
```

Out[5]:
```
".. _boston_dataset:\n\nBoston house prices dataset\n-----------
----------------\n\n**Data Set Characteristics:**  \n\n    :Numb
er of Instances: 506 \n\n    :Number of Attributes: 13 numeric/c
ategorical predictive. Median Value (attribute 14) is usually th
e target.\n\n    :Attribute Information (in order):\n        - C
RIM     per capita crime rate by town\n        - ZN       propor
tion of residential land zoned for lots over 25,000 sq.ft.\n    
- INDUS    proportion of non-retail business acres per town\n   
- CHAS     Charles River dummy variable (= 1 if tract bounds riv
er; 0 otherwise)\n        - NOX      nitric oxides concentration
(parts per 10 million)\n        - RM       average number of roo
ms per dwelling\n        - AGE      proportion of owner-occupied
units built prior to 1940\n        - DIS      weighted distances
to five Boston employment centres\n        - RAD      index of a
ccessibility to radial highways\n        - TAX      full-value p
roperty-tax rate per $10,000\n        - PTRATIO  pupil-teacher r
atio by town\n        - B        1000(Bk - 0.63)^2 where Bk is t
he proportion of black people by town\n        - LSTAT    % lowe
r status of the population\n        - MEDV     Median value of o
wner-occupied homes in $1000's\n\n    :Missing Attribute Values:
None\n\n    :Creator: Harrison, D. and Rubinfeld, D.L.\n\nThis i
s a copy of UCI ML housing dataset.\nhttps://archive.ics.uci.edu
/ml/machine-learning-databases/housing/\n\n\nThis dataset was ta
ken from the StatLib library which is maintained at Carnegie Mel
lon University.\n\nThe Boston house-price data of Harrison, D. a
nd Rubinfeld, D.L. 'Hedonic\nprices and the demand for clean air
', J. Environ. Economics & Management,\nvol.5, 81-102, 1978.   U
sed in Belsley, Kuh & Welsch, 'Regression diagnostics\n...', Wil
ey, 1980.   N.B. Various transformations are used in the table o
n\npages 244-261 of the latter.\n\nThe Boston house-price data h
as been used in many machine learning papers that address regres
sion\nproblems.   \n     \n.. topic:: References\n\n   - Belsley
, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
Data and Sources of Collinearity', Wiley, 1980. 244-261.\n   - Q
uinlan,R. (1993). Combining Instance-Based and Model-Based Learn
ing. In Proceedings on the Tenth International Conference of Mac
hine Learning, 236-243, University of Massachusetts, Amherst. Mo
rgan Kaufmann.\n"
```

DataFrame Creation

In [6]:
```
df = pd.DataFrame(boston.data, columns=boston.feature_names)
```

Addition of target feature in dataframe

In [7]:

```
df['Price'] = boston.target
```

In [8]:

```
df
```

Out[8]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	39
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	39
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	39
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	39
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	39
...	...	...	...	...	...	...	...	...	...	...	...	
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	39
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	39
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	39
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	39
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	39

Information Regarding Dataset

In [9]:
```
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
```

Statistical Summary of the features

In [10]:
```
df.describe()
```

Out[10]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	A
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000

Checking Missing Values

In [11]:
```
df.isnull().sum()
```

Out[11]:
```
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
```

Checking Datatypes of Features

In [12]:
```
df.dtypes
```

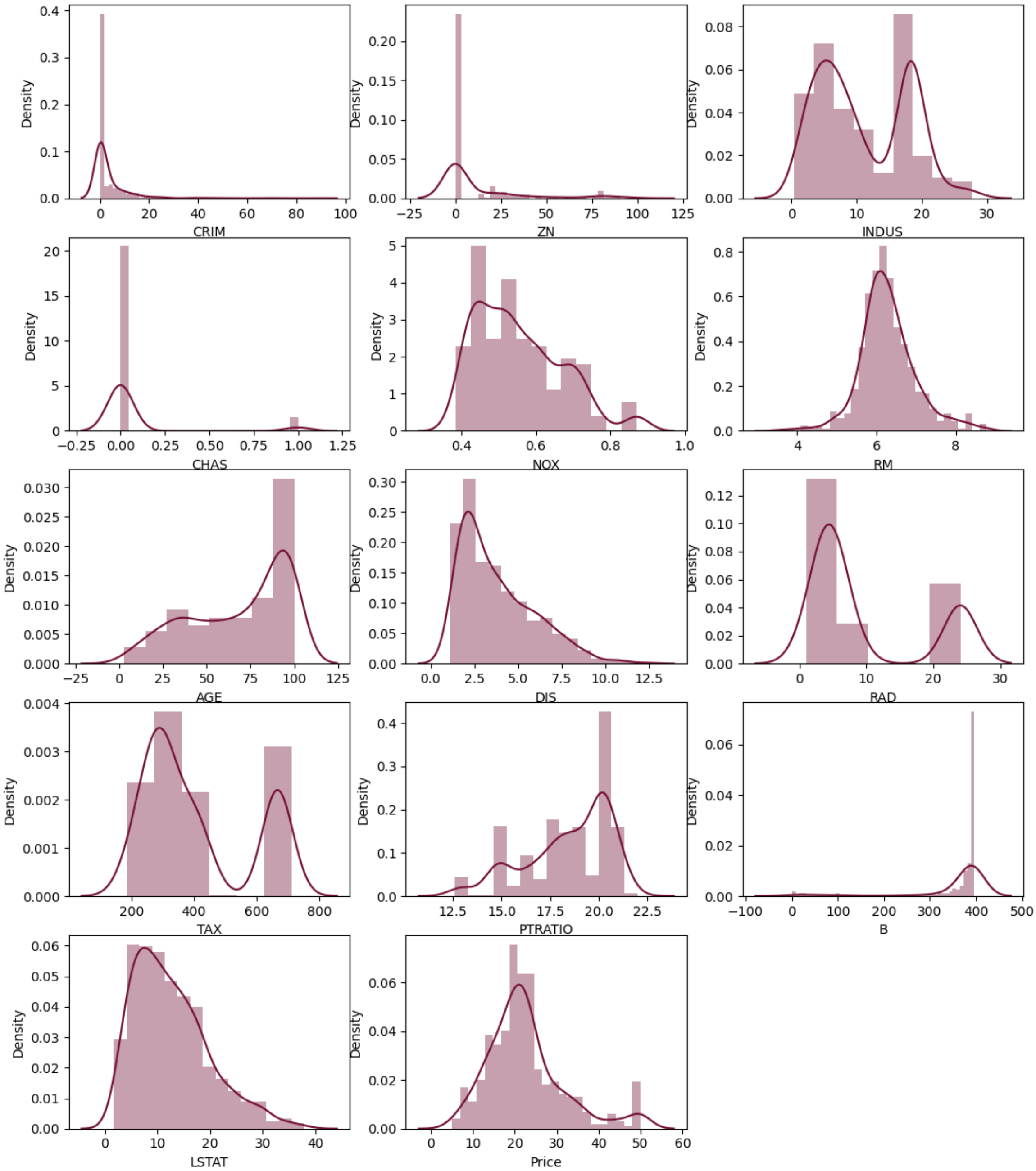
Out[12]:
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS       float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD        float64
TAX        float64
PTRATIO    float64
B          float64
LSTAT      float64
Price      float64
dtype: object
```

Exploratory Data Analysis

Distribution of the Features

In [13]:
```
pos=1
fig = plt.figure(figsize=(13,25))

for i in df.columns:
    ax = fig.add_subplot(8,3,pos)
    pos = pos + 1
    sns.distplot(df[i], ax=ax, color="#751238")
```



Correlation Among Numerical Features

In [14]:
```
df.corr()
```

Out[14]:

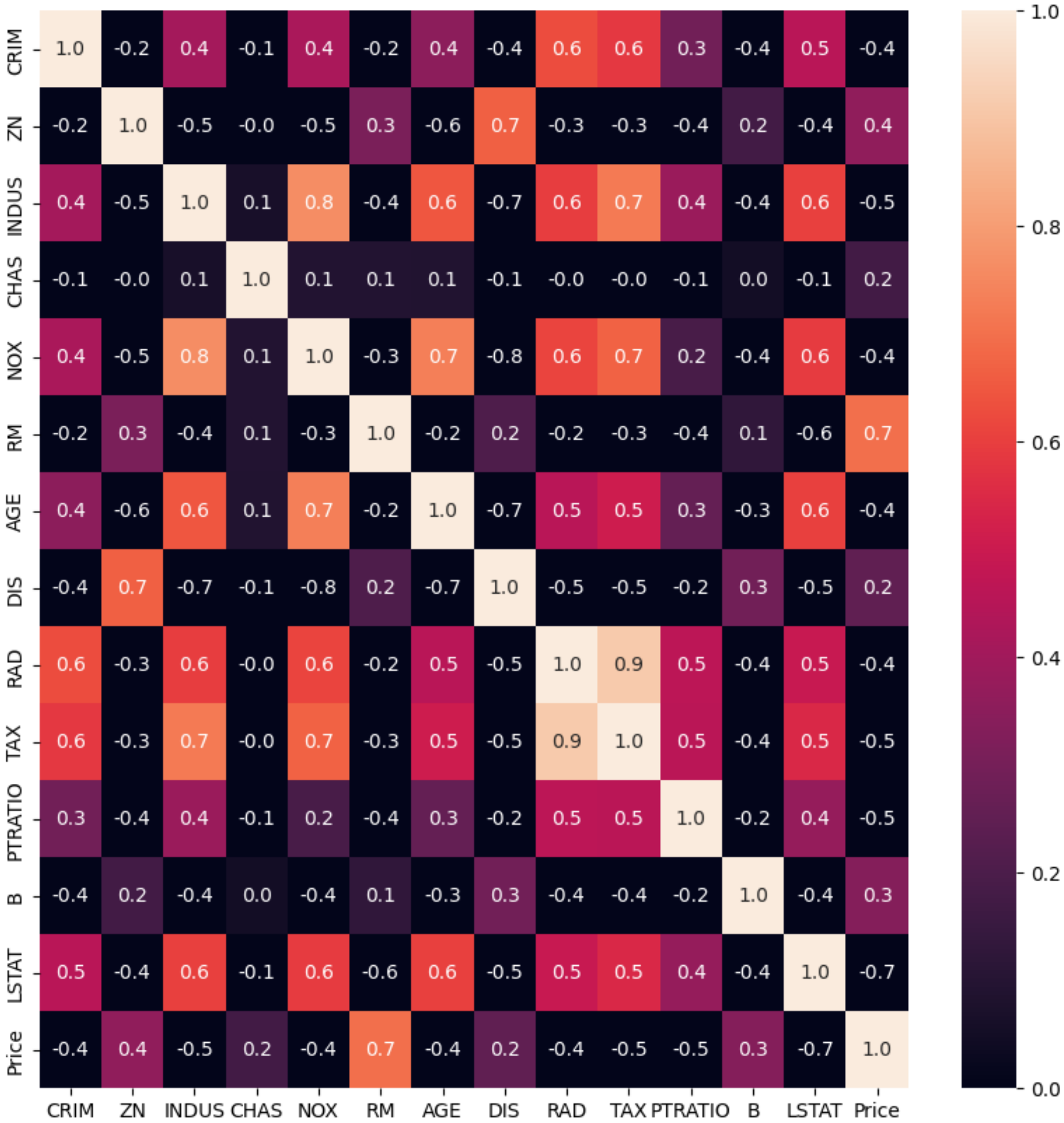
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.
...	...	...	...	...	...	...	...	
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.
Price	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.

HeatMap to visualize correlation

In [ ]:

In [15]:
```
plt.figure(figsize=(10, 10))
sns.heatmap(df.corr(), vmin=0, annot=True,fmt='.1f')
```

Out[15]: <AxesSubplot:>

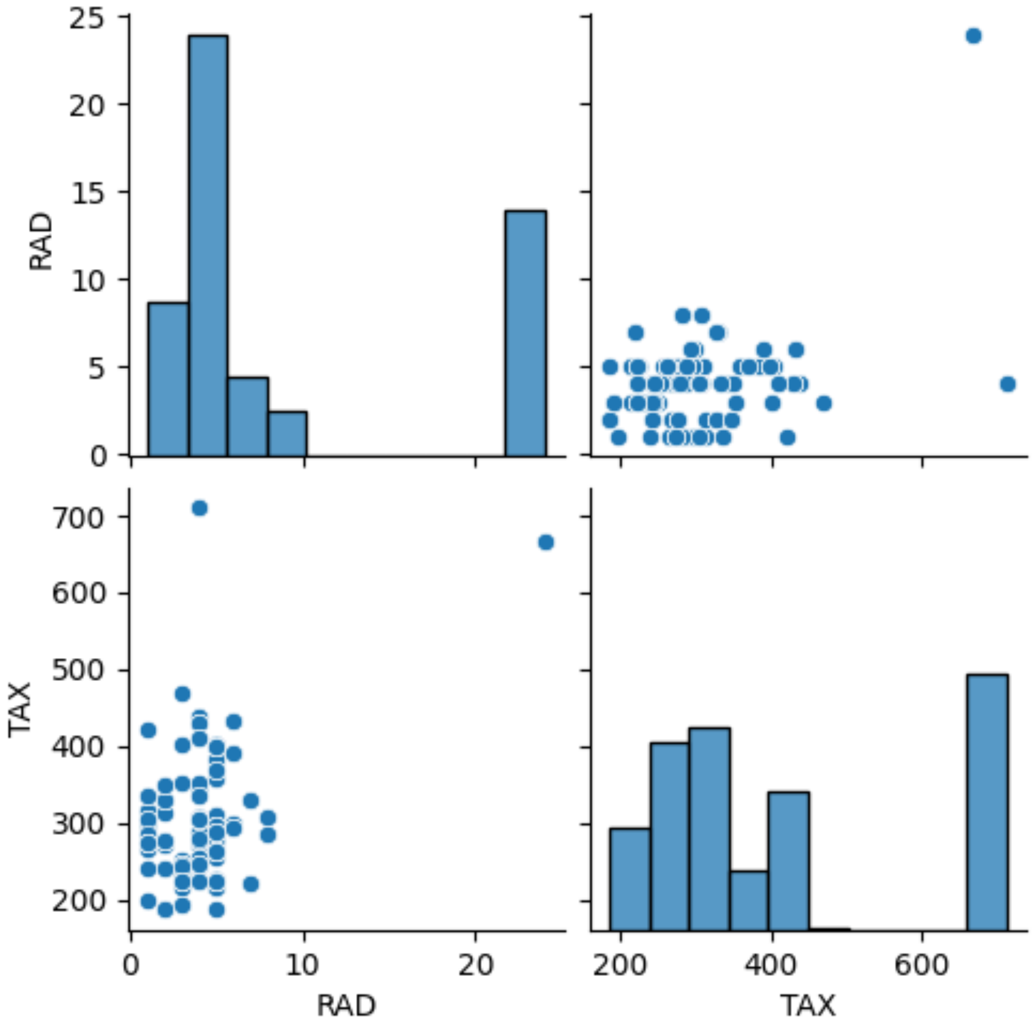


Creation of pair-plot with all the features

In [16]:
```
plt.figure(figsize=(9, 9))
sns.pairplot(df , vars=["RAD", "TAX"])
```

Out[16]:
```
<seaborn.axisgrid.PairGrid at 0x15ff5a740>
```

```
<Figure size 900x900 with 0 Axes>
```



In [17]:
```
df['TAX'].corr(df['RAD'])
```

Out[17]:
```
0.9102281885331849
```

OBSERVATION : Feature TAX & RAD are highly correlated.

Calculation of VIF (Variance Inflation Factor) value for the features

VIF = 1/(1-r2)

Keeping one feature at a time as dependent & others as independent.

Importing the Library

In [18]:
```
from statsmodels.stats.outliers_influence import variance_inflati
```

In [19]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Features & their respective VIF score

In [20]:
```
#Creation of VIF Score method for given dataset
def VIFscore(dataset):
    temp=[]
    for i in df.columns:
        X = np.array(df.drop(i,axis=1))
        y = np.array(df[i])
        lr = LinearRegression()
        lr.fit(X,y)
        y_pred = lr.predict(X)
        r2 = r2_score(y,y_pred)
        vif = 1/(1-r2)
        temp.append((i,vif))
    return temp
```

In [21]:
```
VIFscore(df)
```

Out[21]:
```
[('CRIM', 1.831536683713473),
 ('ZN', 2.352185889014947),
 ('INDUS', 3.9925031533175335),
 ('CHAS', 1.0952226687688211),
 ('NOX', 4.58692024225555),
 ('RM', 2.2603743566681325),
 ('AGE', 3.100842819545981),
 ('DIS', 4.3960072515073945),
 ('RAD', 7.808198432681462),
 ('TAX', 9.205542091810146),
 ('PTRATIO', 1.993015656553282),
 ('B', 1.3814629538442607),
 ('LSTAT', 3.5815848036702103),
 ('Price', 3.855684268833824)]
```

OBSERVATION : RAD & TAX are highly correlated hence leads to multicollinearity in the dataset. Hence we need to drop the column

RESULT : We're dropping RAD since it's correlation wrt Price is -0.4 whereas TAX has -0.5. Hence tax is highly correlated with target feature.

In [22]:
```
df.drop(columns=['RAD'], inplace=True)
```

In [23]:
```
VIFscore(df)
```

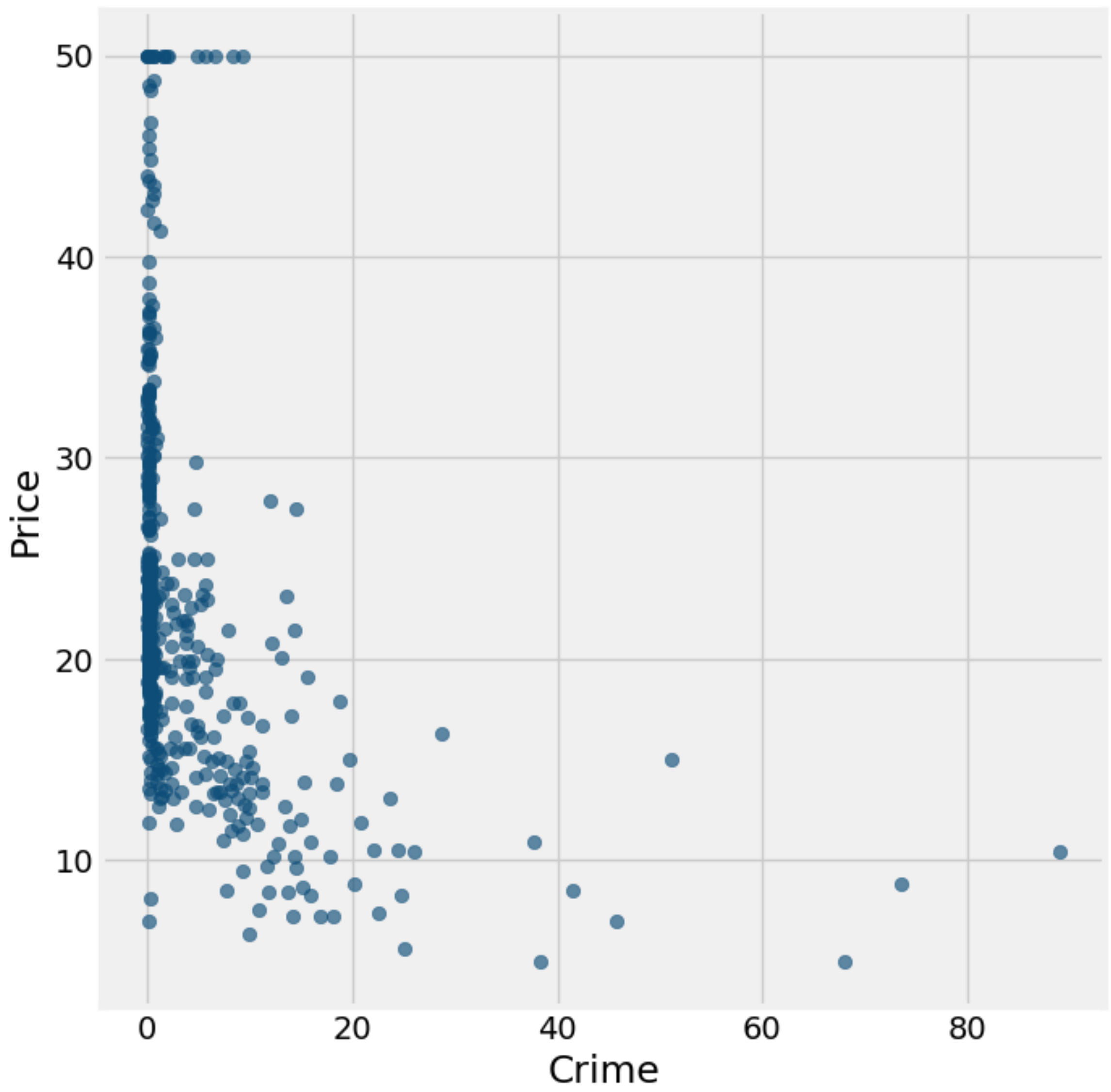
Out[23]:
```
[('CRIM', 1.679218009646556),
 ('ZN', 2.310513229090843),
 ('INDUS', 3.6892806919112746),
 ('CHAS', 1.0889011514418125),
 ('NOX', 4.441583206372164),
 ('RM', 2.250603560344789),
 ('AGE', 3.0835425023527283),
 ('DIS', 4.385150316922098),
 ('TAX', 3.41749811856695),
 ('PTRATIO', 1.8789370351875438),
 ('B', 1.3672470111320019),
 ('LSTAT', 3.5319614193394617),
 ('Price', 3.695840343820758)]
```

OBSERVATION : Now features doesn't look strongly correlated with threshold of 5. Hence we can proceed now.

Scatter Plot using crime & price

In [24]:
```
plt.figure(figsize=(8, 8))
plt.style.use("fivethirtyeight")
plt.scatter(df['CRIM'], df['Price'], color="#0d4d78", alpha=0.65)
plt.xlabel('Crime')
plt.ylabel("Price")
```

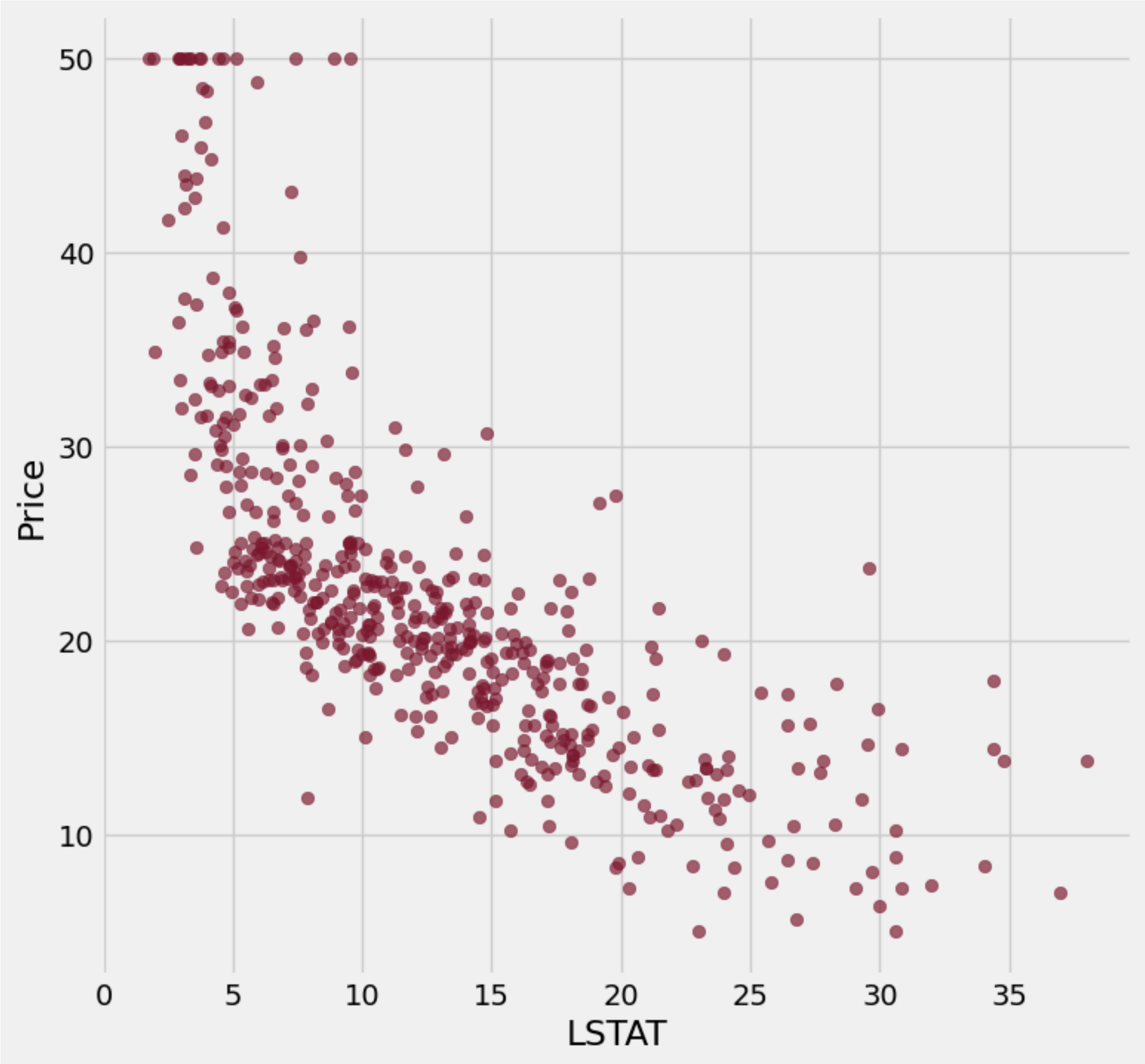
Out[24]: Text(0, 0.5, 'Price')



Scatter Plot using LSTAT & Price

In [25]:
```
plt.figure(figsize=(8, 8))
plt.style.use("fivethirtyeight")
plt.scatter(df['LSTAT'], df['Price'], color="#7a172e", alpha=0.68
plt.xlabel('LSTAT')
plt.ylabel("Price")
```

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



Creation of Independent & Dependent Features

In [26]:
```
X = df.iloc[:, :-1]
Y = df.iloc[:, -1]
```

Splitting the data into train-test sets

Importing the library

In [27]:
```
from sklearn.model_selection import train_test_split
```

In [28]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_si
```

In [29]:
```
X_train
```

Out[29]:

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

In [30]:
```
X_train.shape
```

Out[30]: (339, 12)

In [31]:
```
X_test.shape
```

Out[31]: (167, 12)

Dataset Standardization

In [32]:
```
from sklearn.preprocessing import StandardScaler
```

In [33]:
```
scaler = StandardScaler()
```

In [34]:
```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Model Training Using Linear Regression

In [35]:
```
regression = LinearRegression()
regression
```

Out[35]:
```
▾ LinearRegression
LinearRegression()
```

In [36]:
```
regression.fit(X_train, Y_train)
```

Out[36]:
```
▾ LinearRegression
LinearRegression()
```

Coefficients & Intercept of Linear Regression

In [37]:
```
regression.coef_
```

Out[37]:
```
array([-0.6948544 ,  0.67118376,  0.01739022,  0.9728182 , -1.66
523468,
        2.96696856, -0.47536504, -2.98774912,  0.34037056, -1.88
277704,
        0.98254063, -3.86612274])
```

In [38]:
```
regression.intercept_
```

Out[38]:
```
22.970796460176988
```

Prediction for the test-data

In [39]:
```
reg_pred = regression.predict(X_test)
```

In [40]:
```
reg_pred
```

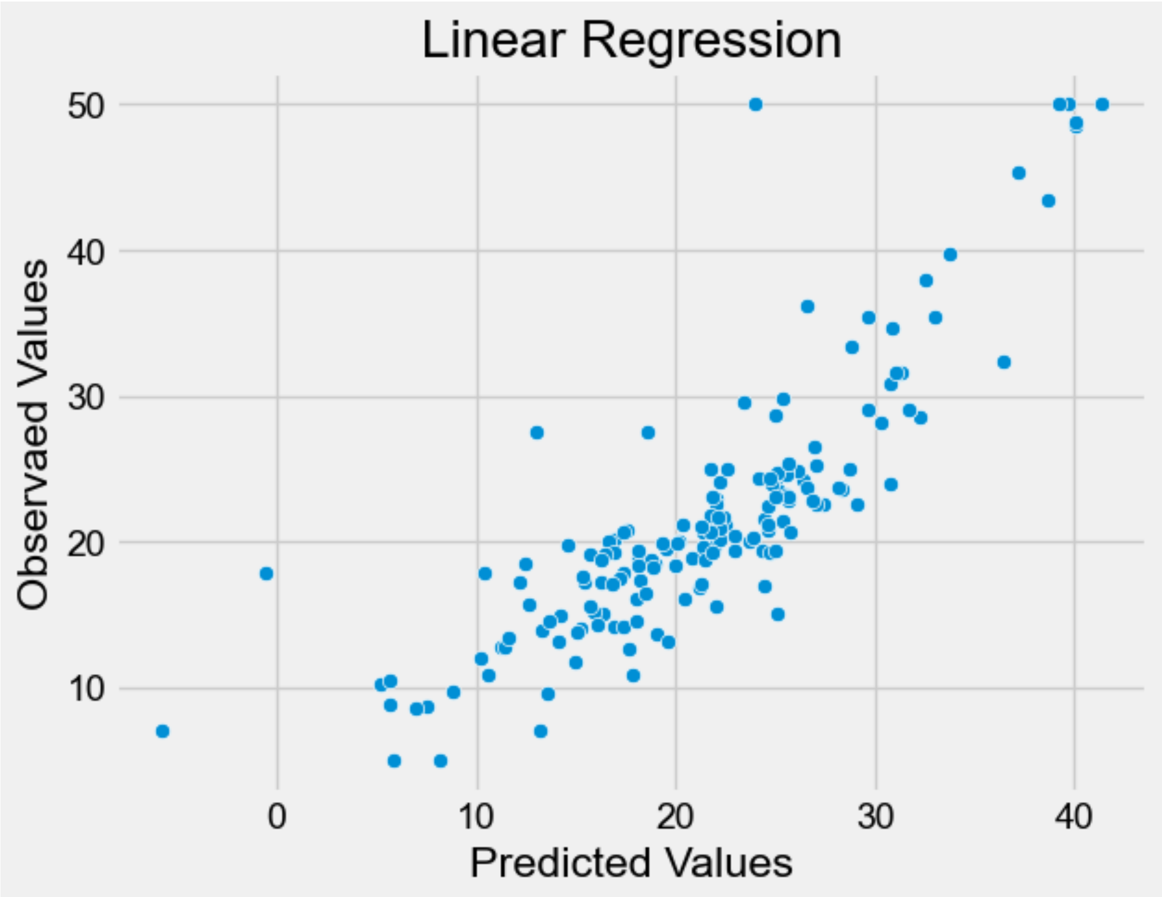
Out[40]:
```
array([28.40172358, 36.47498509, 19.09018005, 25.6431586 , 18.05
662783,
       23.71787099, 17.4157474 , 15.28841066, 21.92120584, 21.19
376141,
       24.44674664, 18.0996987 , -5.7420562 , 22.48235758, 18.94
505605,
       25.40706761, 18.82467685,  5.17605812, 39.72248792, 16.90
995649,
       27.08898935, 29.63203528, 11.26322145, 24.63025306, 17.36
365246,
       15.05212813, 23.86305287, 14.2366082 , 22.44978015, 19.95
775594,
       21.88213493, 25.05448432, 25.08659743, 17.5857199 , 15.66
578508,
       18.09798088, 30.87356526, 19.55667496, 24.21363187, 25.16
```

```
13869 ,
       14.58656191, 30.34309596, 41.33930137, 18.24351358, 27.41
648972,
       16.3814636 , 14.09447397, 26.38836234, 19.33680825, 30.79
40503 ,
       20.86419861, 33.01635263, 15.9208046 , 27.00881044, 38.66
14825 ,
       22.14959437, 18.15201927, 32.30366396, 24.80542817, 12.46
746558,
       21.74228885, 29.65302676, 31.1015131 , 16.91361104, 22.24
441455,
       16.65742999, 19.66383382, 26.12182208, 30.74474695, 11.44
287748,
       20.16227741, 26.60377933, 10.60634006, 17.3521976 , 24.64
580195,
        5.88594008, 22.21906523, 40.09291491, 17.83545768, 13.21
038916,
       21.82708152, 12.15318921, 22.21649199,  8.79769999, 22.99
86593 ,
       31.7404429 , 18.4741991 , 25.53141345, 28.71653044, 21.25
00915 ,
       25.52969724,  5.69480323, 21.35815279, 16.25359663, 13.00
92131 ,
       22.01120143, 24.00611521, -0.60254367, 13.5982796 , 15.38
983968,
       22.04715944, 25.36132253, 10.23152018, 20.08971611, 24.38
014699,
       11.58515778, 18.83090903, 25.59906266, 20.36262719, 25.09
055424,
        7.53425019, 18.624313  , 21.35619404, 26.60654339, 31.35
562117,
       15.01352309, 33.73493464, 13.30669887, 21.73025384, 28.15
987454,
       15.33804226, 24.7128603 ,  5.69198827, 24.72374341, 25.68
100438,
       22.9697817 , 25.63103151, 32.58390889, 22.03824149, 37.22
434284,
       12.61050316, 27.03833431, 18.04189922, 21.43576274, 10.43
560181,
       20.44988907, 21.7649578 , 31.09160628, 31.68705747, 15.71
130039,
       17.18390598, 29.10393446, 24.98092889, 16.89908065,  6.98
605289,
       25.76332428, 24.43557487, 16.83147135, 13.62842605, 39.22
343979,
       16.12686341, 17.68816858, 24.99869562, 24.60908248, 21.85
088036,
       21.85242629, 16.4182298 , 22.5647643 , 28.78949925,  8.21
851491,
       23.47704722, 16.22345119, 22.09352029, 24.99094456, 26.90
117234,
       21.29308585, 40.09956719])
```

Plotting predicted value vs real values

In [41]:
```
plt.xlabel("Predicted Values")
plt.ylabel("Observaed Values")
plt.title("Linear Regression")
sns.set_style("dark")
sns.scatterplot(x = reg_pred, y=Y_test, palette="plasma", ci=25)
```

Out[41]:
```
<AxesSubplot:title={'center':'Linear Regression'}, xlabel='Predi
cted Values', ylabel='Observaed Values'>
```



Result : Almost Linear Relation between Predicted Values & Target Values.

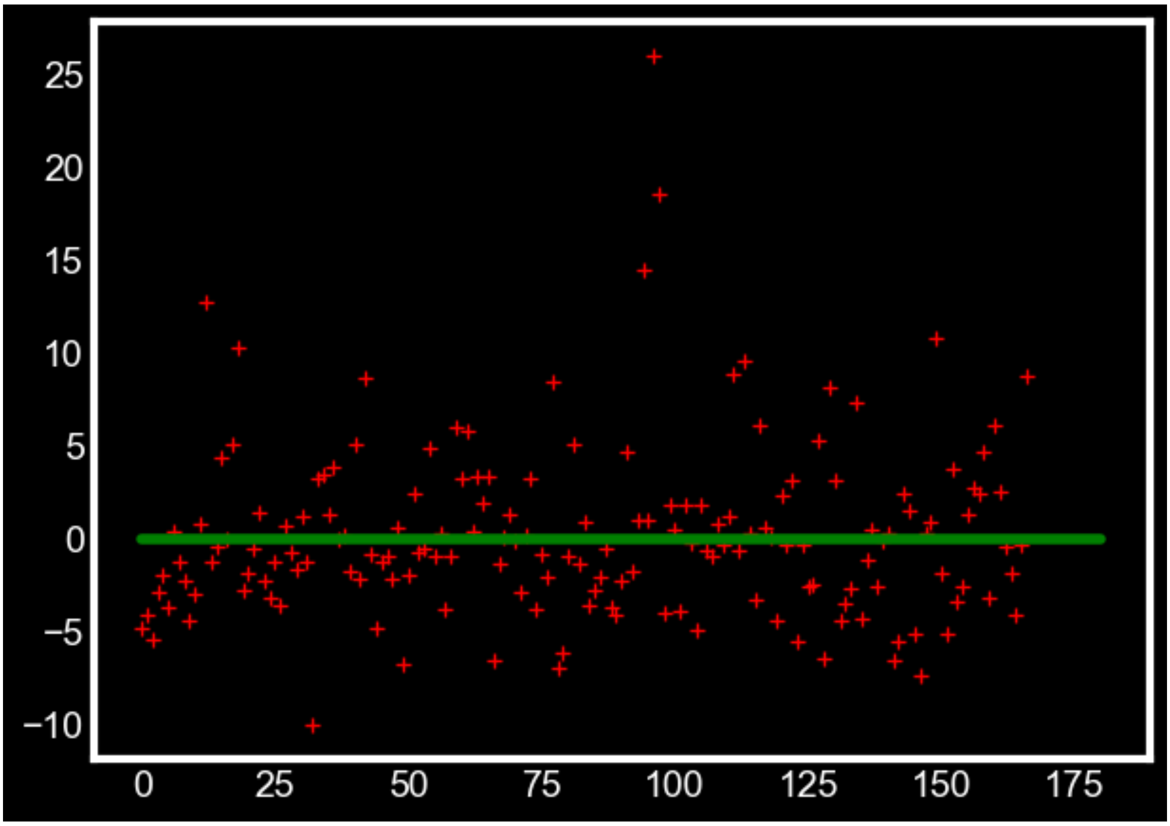
Calculating Erorr/Residuals

In [42]:
```
residuals = Y_test - reg_pred
```

Plotting Error

In [43]:
```
plt.style.use("dark_background")
plt.plot(np.arange(residuals.size), residuals, "r+")
sns.lineplot([0, 180], [0, 0], color='green')
```

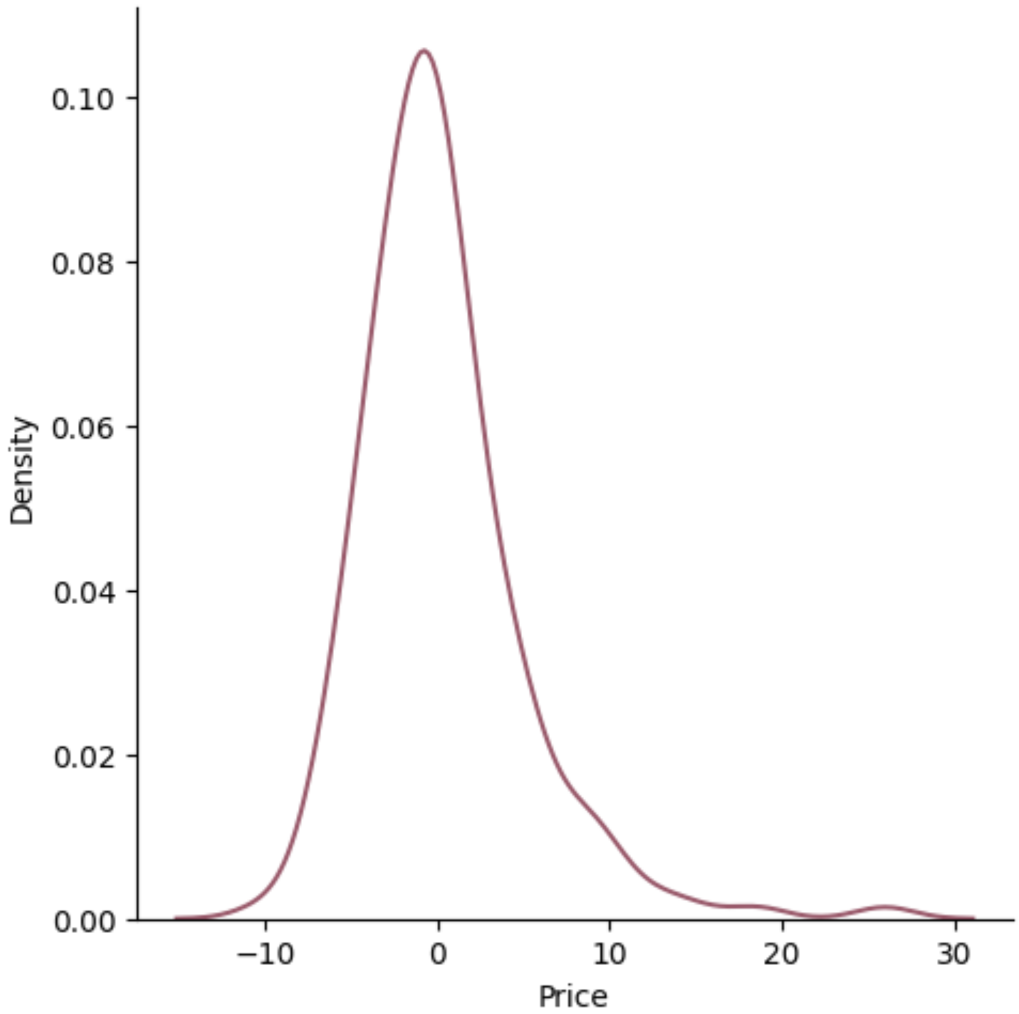
Out[43]: <AxesSubplot:>



Error Distribution

In [44]:
```
plt.style.use('default')
sns.displot(residuals, kind = "kde", color="#99596a")
```

Out[44]: <seaborn.axisgrid.FacetGrid at 0x169026230>

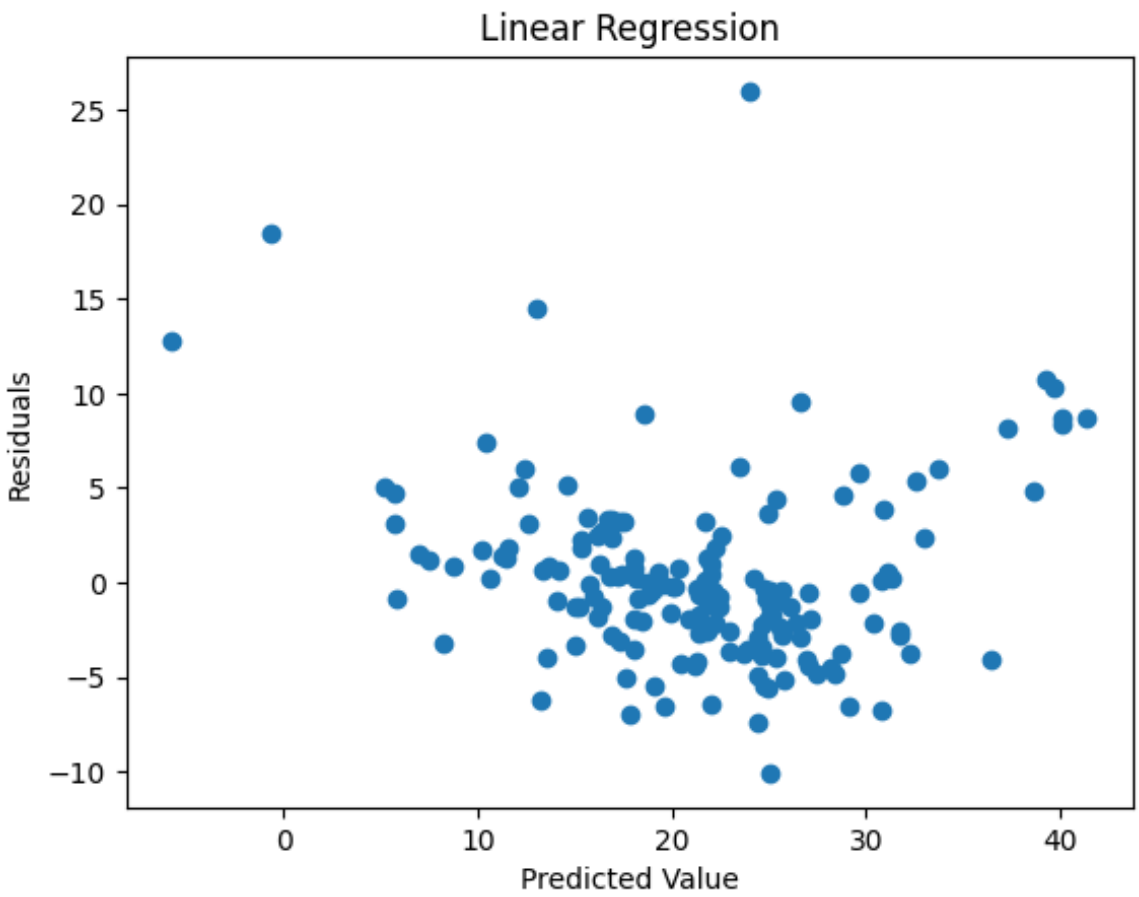


Result : Error distribution is approximately normally distibuted with little right-skewed tail.

Realtion between predicted values & residuals

In [45]:
```
plt.xlabel("Predicted Value")
plt.ylabel("Residuals")
plt.title("Linear Regression")
plt.scatter(reg_pred, residuals)
```

Out[45]:
```
<matplotlib.collections.PathCollection at 0x169097370>
```



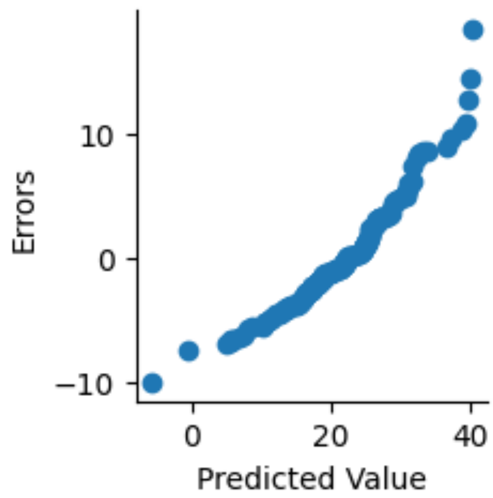
Creation of temporary dataframe

In [46]:
```
temp_df = pd.DataFrame([residuals.to_list(), reg_pred.tolist()])
temp_df = temp_df.T
temp_df.rename(columns = {0:"Errors", 1:"Predicted Value"},inplac
```

Plotting Quantile-Quantile Graph

In [47]:
```
pplot(data = temp_df, x = "Predicted Value", y = "Errors", kind="
```

Out[47]:
```
<seaborn.axisgrid.PairGrid at 0x1690c09d0>
```



RESULT : Approximate Uniform Distribution between Residuals & Predicted Values

Performance Metric

Importing Libraries

In [48]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
```

Total Error Calculated for different Cost Functions

In [49]:
```
print(f"Mean Sqaured Error: {mean_squared_error(Y_test, reg_pred)}
print(f"Mean Sqaured Error: {mean_absolute_error(Y_test, reg_pred
print(f"Mean Sqaured Error: {np.sqrt(mean_squared_error(Y_test, r
```

```
Mean Sqaured Error: 22.339013815006886
Mean Sqaured Error: 3.275010643648515
Mean Sqaured Error: 4.726416593467707
```

Calculation of R2 & Adjusted R2

In [50]:
```
r2 = r2_score(Y_test, reg_pred)
adjusted_r2 = 1 - ((1-r2)*(len(Y_test)-1)/(len(Y_test)-1-X.shape[

print(f"R2 Score: {r2}")
print(f"Adjusted R2 Score: {adjusted_r2}")
```

```
R2 Score: 0.7048169381846415
Adjusted R2 Score: 0.6818156606405876
```

Model Training Using Ridge Regression

Importing Libraries

In [51]:
```
from sklearn.linear_model import Ridge
```

In [52]:
```
ridge_regression = Ridge()
ridge_regression
```

Out[52]:
```
▾ Ridge
Ridge()
```

In [53]:
```
ridge_regression.fit(X_train, Y_train)
```

Out[53]:
```
▾ Ridge
Ridge()
```

Coefficients & Intercept of Ridge Regression

In [54]:
```
ridge_regression.coef_
```

Out[54]:
```
array([-0.69062437,  0.66221453,  0.01328714,  0.97337433, -1.63
441186,
        2.96787292, -0.4725375 , -2.94776997,  0.32560195, -1.87
48181 ,
        0.98093204, -3.85363594])
```

In [55]:
```
ridge_regression.intercept_
```

Out[55]:
```
22.970796460176988
```

Prediction for the test-data

In [56]:
```
reg_pred_ridge = ridge_regression.predict(X_test)
```

In [57]:
```
reg_pred_ridge
```

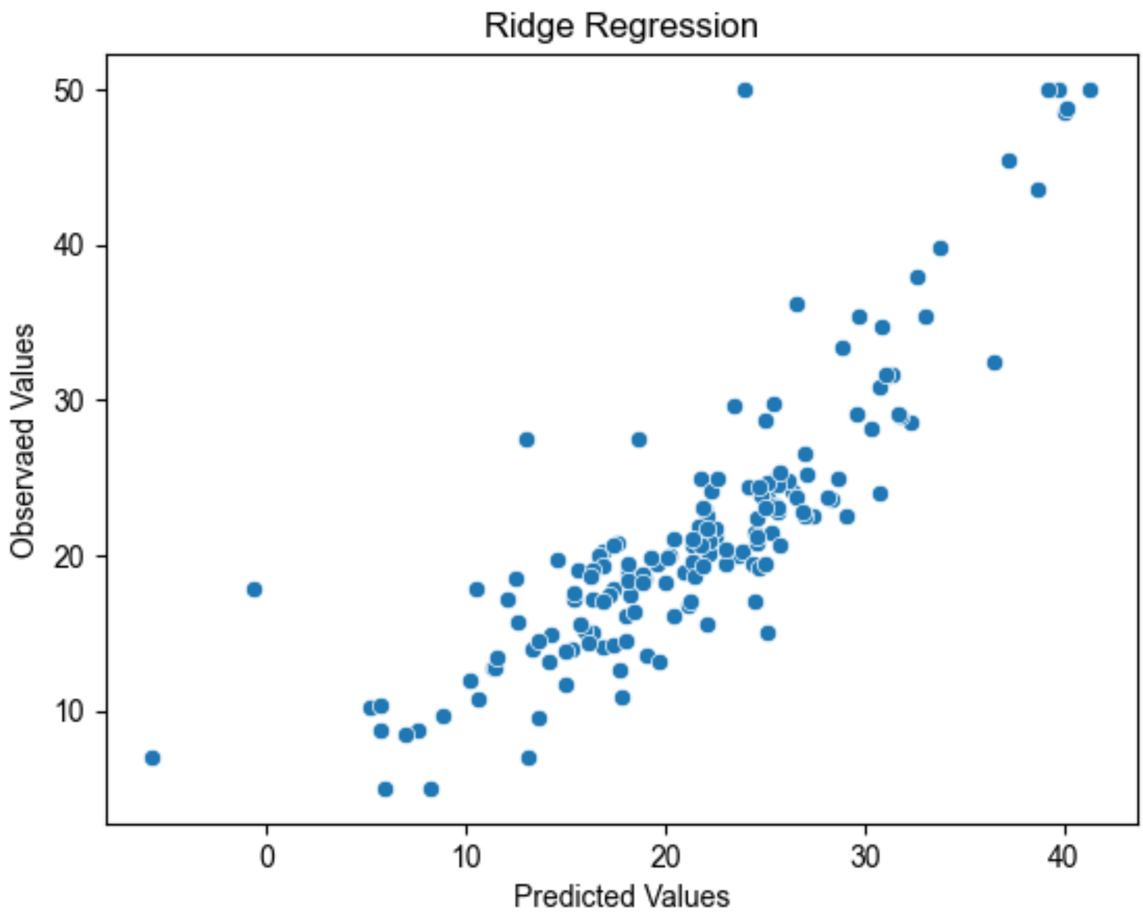
Out[57]:
```
array([28.37441779, 36.43582674, 19.05263672, 25.62994054, 18.06
687151,
       23.69523048, 17.44050253, 15.29219653, 21.88899758, 21.18
914263,
       24.44767278, 18.15019415, -5.68959958, 22.47589706, 18.96
451643,
       25.38300062, 18.82546514,  5.1959655 , 39.68079819, 16.92
238641,
       27.07561427, 29.59791629, 11.31006867, 24.62920471, 17.36
549555,
       15.03269632, 23.83905776, 14.26183929, 22.46162215, 19.95
891137,
       21.86311676, 25.0611695 , 25.06919467, 17.59221657, 15.65
297204,
       18.13673458, 30.87603901, 19.59419596, 24.22467221, 25.15
205761,
       14.59795719, 30.31347853, 41.28291444, 18.26058111, 27.39
676384,
       16.3804267 , 14.12571021, 26.37768345, 19.32130158, 30.78
156112,
       20.90069057, 33.00186772, 15.95579788, 26.98960867, 38.62
597342,
       22.13953108, 18.16223376, 32.2633411 , 24.81005889, 12.52
88791 ,
       21.7984193 , 29.67527026, 31.071926  , 16.92827518, 22.30
502286,
       16.6794717 , 19.65937541, 26.11401278, 30.71430047, 11.46
589734,
       20.17604472, 26.5669914 , 10.63300355, 17.42062643, 24.61
758404,
        5.9156098 , 22.20104468, 40.03801947, 17.83224652, 13.18
113074,
       21.83663755, 12.1341694 , 22.2406193 ,  8.81031251, 22.99
089421,
       31.73152828, 18.48977684, 25.53843729, 28.67593263, 21.24
241631,
       25.51701157,  5.71364195, 21.35446382, 16.31428231, 12.98
542204,
       22.00652868, 23.96233986, -0.59538068, 13.62213843, 15.37
342343,
       22.05011038, 25.338657  , 10.22953547, 20.11233656, 24.38
581401,
       11.58056315, 18.84663655, 25.61467036, 20.41013035, 25.11
813971,
        7.57212416, 18.60206628, 21.41121909, 26.58503527, 31.33
441417,
       15.02762924, 33.71700421, 13.34695567, 21.71638603, 28.15
367368,
```

```
       15.39001071, 24.73713543,  5.7334003 , 24.69827711, 25.68
982929,
       22.98008537, 25.64862627, 32.56062781, 22.08289941, 37.20
206934,
       12.62366775, 27.02600207, 18.07188664, 21.41977971, 10.49
166687,
       20.41844329, 21.79175174, 31.09233146, 31.66009425, 15.74
972137,
       17.21582617, 29.09313718, 24.95786357, 16.91766924,  7.01
392749,
       25.73435218, 24.47640316, 16.85027773, 13.6679839 , 39.19
538122,
       16.13953043, 17.70390101, 25.0274268 , 24.58550616, 21.85
769706,
       21.86952122, 16.40867078, 22.57588226, 28.81367641,  8.25
052947,
       23.47556473, 16.27288079, 22.07972447, 24.99720255, 26.87
942873,
       21.3216344 , 40.08199737])
```

Plotting predicted value vs real values

In [58]:
```
plt.xlabel("Predicted Values")
plt.ylabel("Observaed Values")
plt.title("Ridge Regression")
sns.set_style("whitegrid")
sns.scatterplot(x = reg_pred_ridge, y=Y_test, palette="plasma", c
```

Out[58]:
```
<AxesSubplot:title={'center':'Ridge Regression'}, xlabel='Predic
ted Values', ylabel='Observaed Values'>
```



Result : Almost Linear Relation between Predicted Values & Target Values.

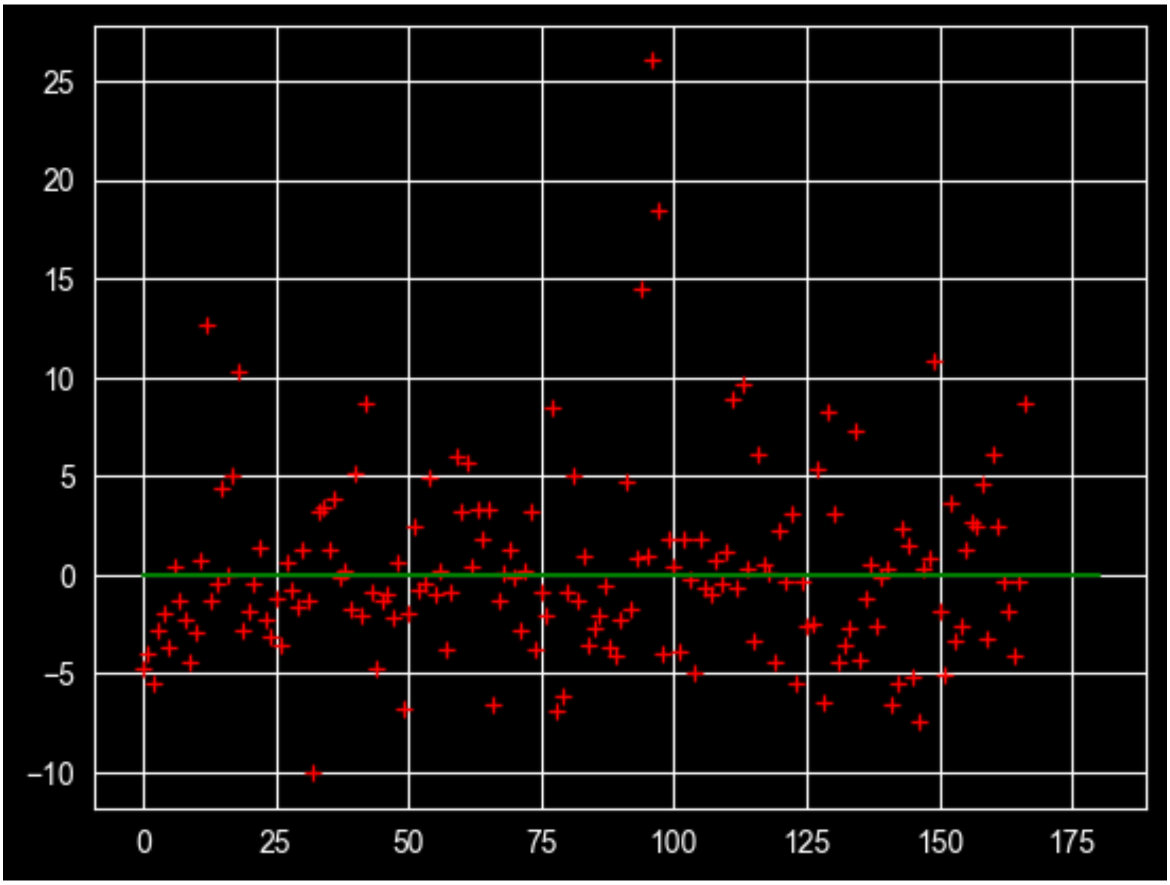
Calculating Erorr/Residuals

In [59]:
```
residuals_ridge = Y_test - reg_pred_ridge
```

Plotting Error

In [60]:
```
plt.style.use("dark_background")
plt.plot(np.arange(residuals_ridge.size), residuals_ridge, "r+")
sns.lineplot([0, 180], [0, 0], color='green')
```

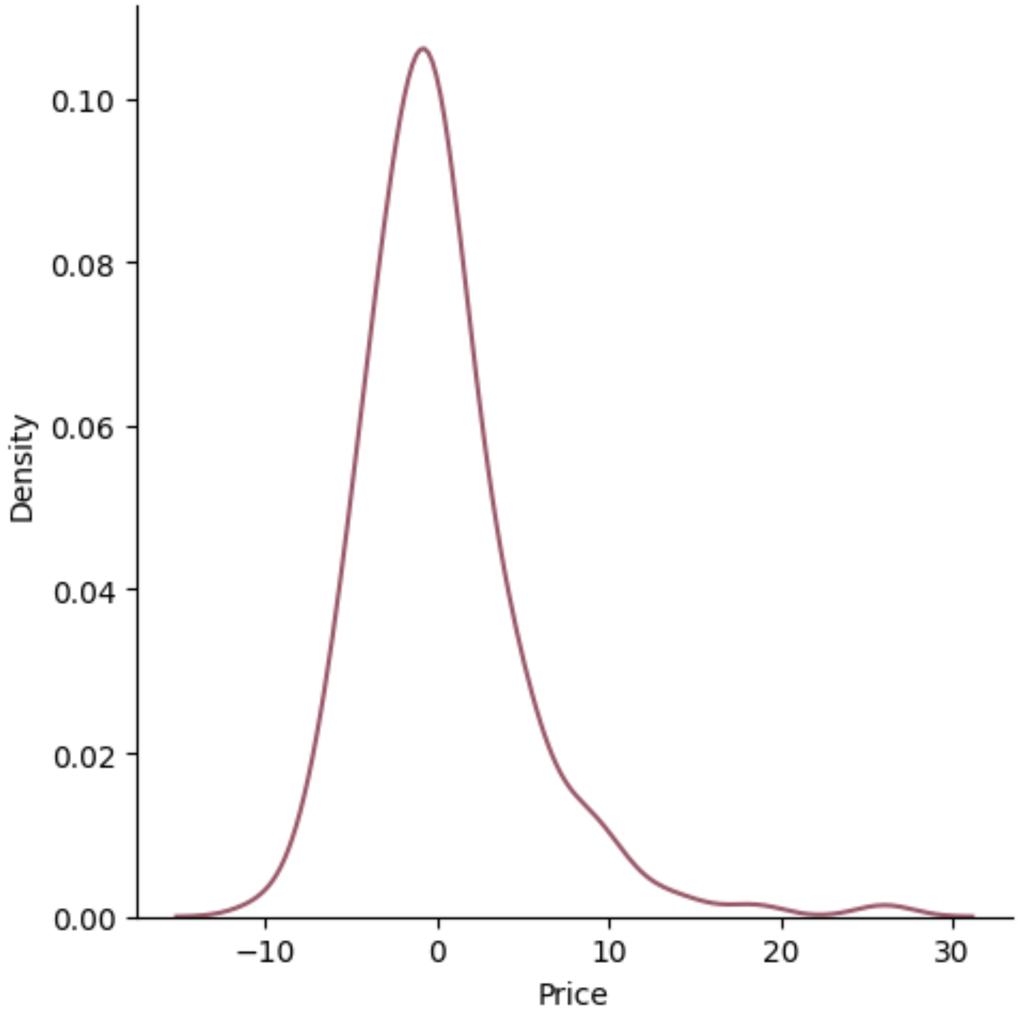
Out[60]: <AxesSubplot:>



Error Distribution

In [61]:
```
plt.style.use('default')
sns.displot(residuals_ridge, kind = "kde", color="#99596a")
```

Out[61]: <seaborn.axisgrid.FacetGrid at 0x16921f820>

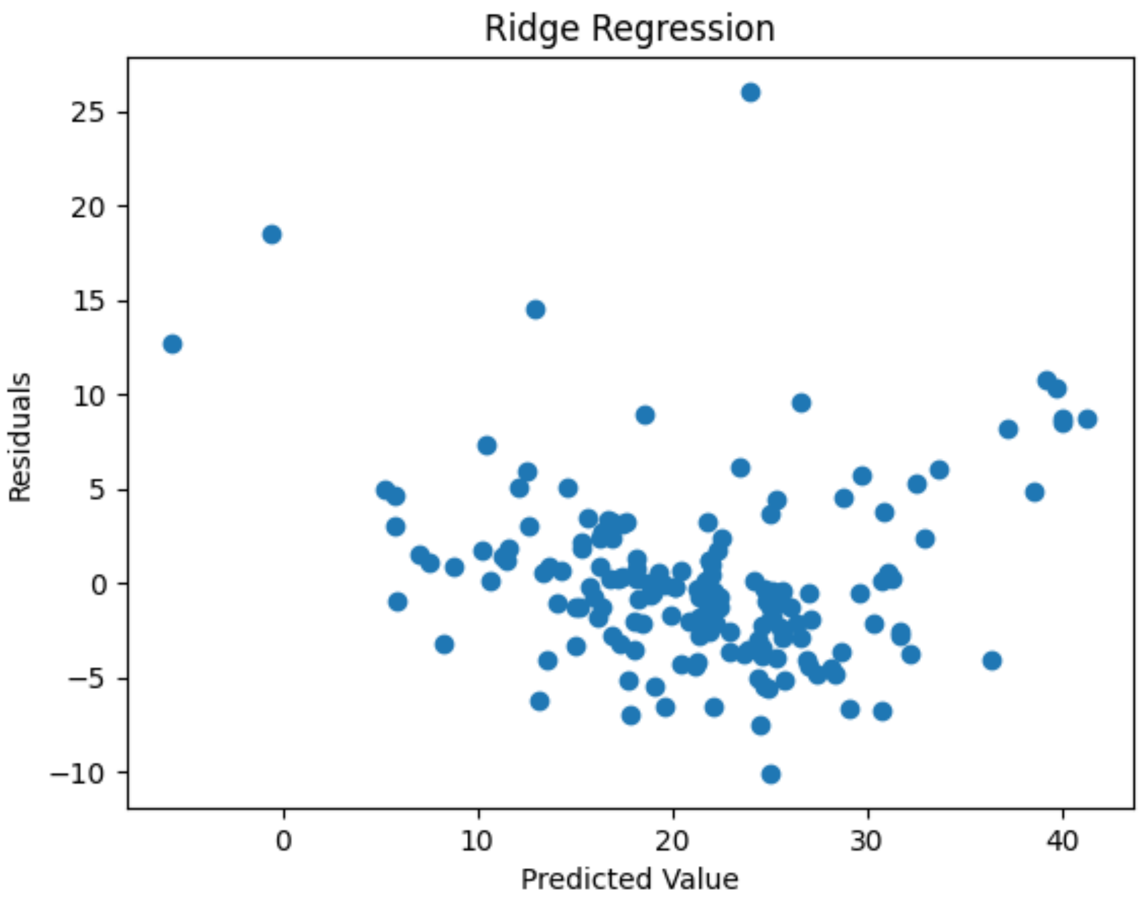


Result : Error distribution is approximately normally distibuted with little right-skewed tail.

Realtion between predicted values & residuals

In [62]:
```
plt.xlabel("Predicted Value")
plt.ylabel("Residuals")
plt.title("Ridge Regression")
plt.scatter(reg_pred_ridge, residuals_ridge)
```

Out[62]:
```
<matplotlib.collections.PathCollection at 0x1692fc5b0>
```



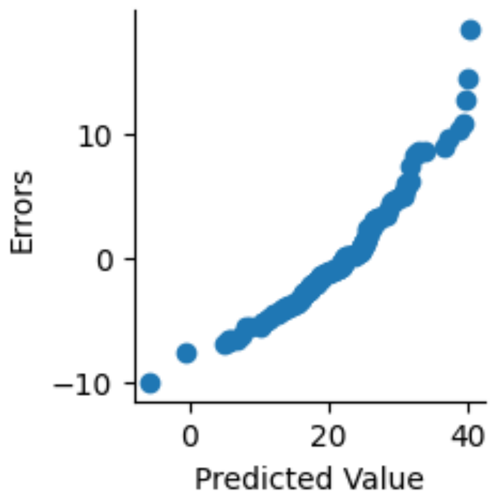
Creation of temporary dataframe

In [63]:
```
temp_df = pd.DataFrame([residuals_ridge.to_list(), reg_pred_ridge
temp_df = temp_df.T
temp_df.rename(columns = {0:"Errors", 1:"Predicted Value"},inplac
```

Plotting Quantile-Quantile Graph

In [64]:
```
pplot(data = temp_df, x = "Predicted Value", y = "Errors", kind="
```

Out[64]: <seaborn.axisgrid.PairGrid at 0x16926e110>



RESULT : Approximate Linear Distribution between Residuals & Predicted Values

Performance Metric

Total Error Calculated for different Cost Functions

In [65]:
```
print(f"Mean Sqaured Error: {mean_squared_error(Y_test, reg_pred_
print(f"Mean Sqaured Error: {mean_absolute_error(Y_test, reg_pred_
print(f"Mean Sqaured Error: {np.sqrt(mean_squared_error(Y_test, r
```

```
Mean Sqaured Error: 22.32981472820066
Mean Sqaured Error: 3.270648366122255
Mean Sqaured Error: 4.725443336682883
```

Calculation of R2 & Adjusted R2

In [66]:
```
r2 = r2_score(Y_test, reg_pred_ridge)
adjusted_r2 = 1 - ((1-r2)*(len(Y_test)-1)/(len(Y_test)-1-X.shape[

print(f"R2 Score: {r2}")
print(f"Adjusted R2 Score: {adjusted_r2}")
```

```
R2 Score: 0.7049384929959619
Adjusted R2 Score: 0.6819466872553874
```

Model Training Using Lasso Regression

Importing Libraries

In [67]:
```
from sklearn.linear_model import Lasso
```

In [68]:
```
lasso_regression = Lasso()
lasso_regression
```

Out[68]:
```
▾ Lasso
Lasso()
```

In [69]:
```
lasso_regression.fit(X_train, Y_train)
```

Out[69]:
```
▾ Lasso
Lasso()
```

Coefficients & Intercept of Lasso Regression

In [70]:
```
lasso_regression.coef_
```

Out[70]:
```
array([-0.        ,  0.        , -0.        ,  0.27140288, -0.
,
        2.62932086, -0.        , -0.        , -0.        , -1.21
106795,
        0.29872592, -3.81788427])
```

In [71]:
```
lasso_regression.intercept_
```

Out[71]:
```
22.970796460176988
```

Prediction for the test-data

In [72]:
```
reg_pred_lasso = lasso_regression.predict(X_test)
```

In [73]:
```
reg_pred_lasso
```

Out[73]:
```
array([26.08015455, 30.74800639, 17.78164865, 25.25224746, 19.28
387238,
       22.81161765, 18.31125159, 14.6359236 , 21.41277816, 20.44
276657,
       20.78573705, 21.00978439,  1.291015  , 22.48591108, 20.42
079939,
```

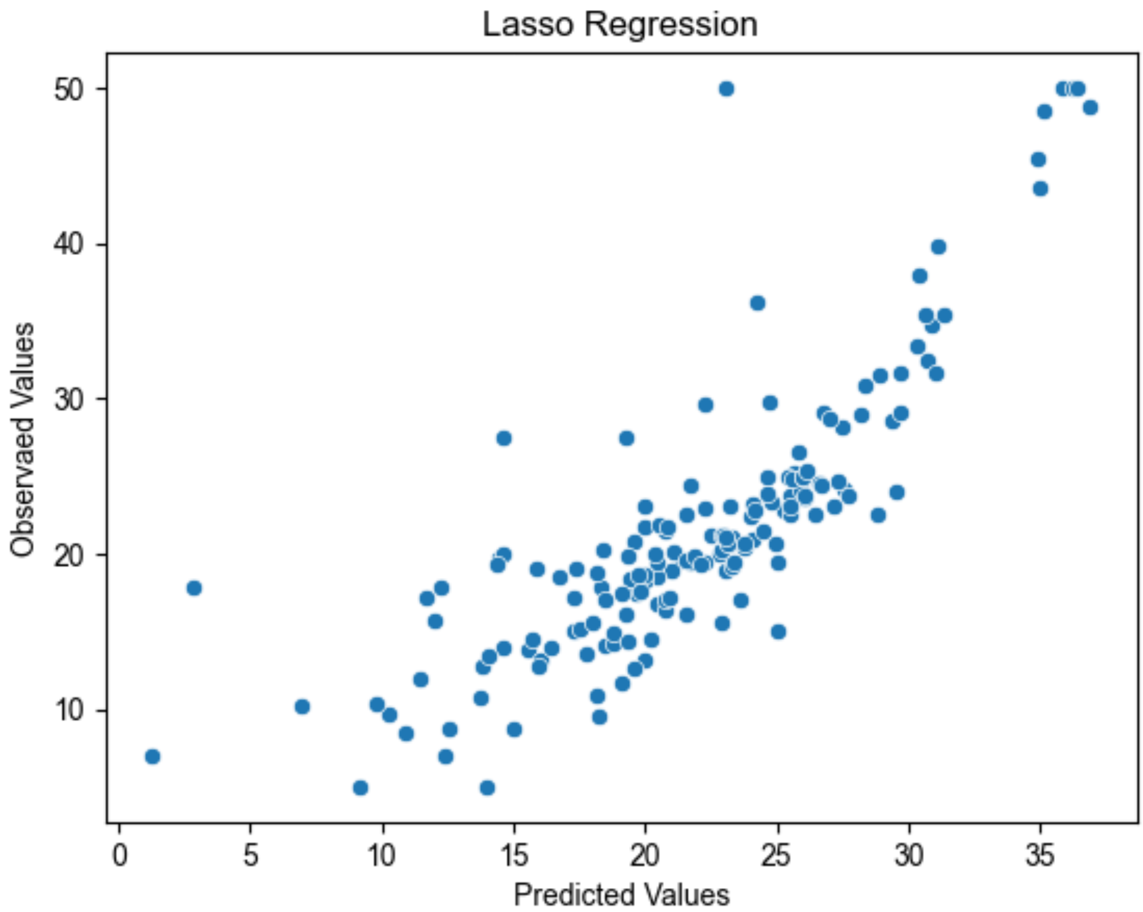
```
       24.7311525 , 18.16643037,  6.95747125, 35.82658712, 18.45
664322,
       25.66618065, 26.77096267, 13.79601985, 24.00317028, 18.83
6776   ,
       15.53225582, 22.93568029, 18.81410942, 19.96419947, 19.71
394574,
       19.99292728, 25.48086797, 25.07506372, 19.62299128, 15.87
164423,
       20.47826682, 30.90020626, 21.73740749, 21.69357954, 24.78
795178,
       14.48946285, 27.49872589, 36.28097532, 19.68302825, 25.54
695917,
       17.26691105, 16.01035547, 25.87512545, 19.37058388, 29.52
965179,
       23.10173703, 31.37342868, 17.55332683, 25.82107042, 34.98
857122,
       22.9126753 , 19.39674982, 29.34678428, 24.65125374, 16.72
971665,
       25.42537367, 30.67518447, 28.90511218, 18.42571757, 27.56
42666 ,
       14.62706935, 20.02272704, 25.60745031, 28.32959641, 15.91
971427,
       20.36020504, 26.04012229, 13.70562157, 23.19186594, 23.25
384089,
        9.1479159 , 21.08680481, 35.1320302 , 18.20120893, 12.40
579132,
       23.03574802, 11.70030468, 24.10234405, 10.23869508, 22.24
788459,
       28.20852166, 20.77401748, 26.01572289, 25.97666655, 20.77
471661,
       24.05595252,  9.7965817 , 21.55718522, 20.96232355, 14.58
941679,
       22.29462601, 23.04513088,  2.8781068 , 18.26028624, 17.31
405321,
       21.55660953, 24.48282524, 11.4677224 , 21.88129868, 25.04
349251,
       14.07796207, 19.97841704, 26.61705394, 23.30429148, 27.32
736054,
       12.59741116, 19.28050049, 24.94727931, 24.22470247, 29.72
519272,
       19.11634486, 31.14895738, 16.43050202, 20.50890153, 27.69
026986,
       19.80308005, 26.66386821, 15.01321246, 23.31466105, 26.15
446064,
       23.80801574, 27.15999779, 30.37432042, 22.93936082, 34.91
159766,
       11.9726422 , 26.4515336 , 20.25377777, 19.96681044, 12.21
677592,
       21.57201037, 23.11587981, 31.05309695, 29.72484235, 18.03
669392,
       19.12012663, 28.86792284, 23.41788473, 14.3667999 , 10.91
433826,
       23.78530314, 23.5888595 , 18.48704121, 15.72569037, 36.38
```

```
671494,
        19.3888044 , 19.56932158, 27.03174416, 22.95041994, 22.07
807956,
        23.22702513, 17.41528182, 24.66493955, 30.31735715, 13.99
988905,
        22.25446887, 19.75004552, 20.83506585, 25.47389698, 24.13
577664,
        23.02944583, 36.90324422])
```

Plotting predicted value vs real values

In [74]:
```
plt.xlabel("Predicted Values")
plt.ylabel("Observaed Values")
plt.title("Lasso Regression")
sns.set_style("whitegrid")
sns.scatterplot(x = reg_pred_lasso, y=Y_test, palette="plasma", c
```

Out[74]:
```
<AxesSubplot:title={'center':'Lasso Regression'}, xlabel='Predic
ted Values', ylabel='Observaed Values'>
```



Result : Almost Linear Relation between Predicted Values & Target Values.

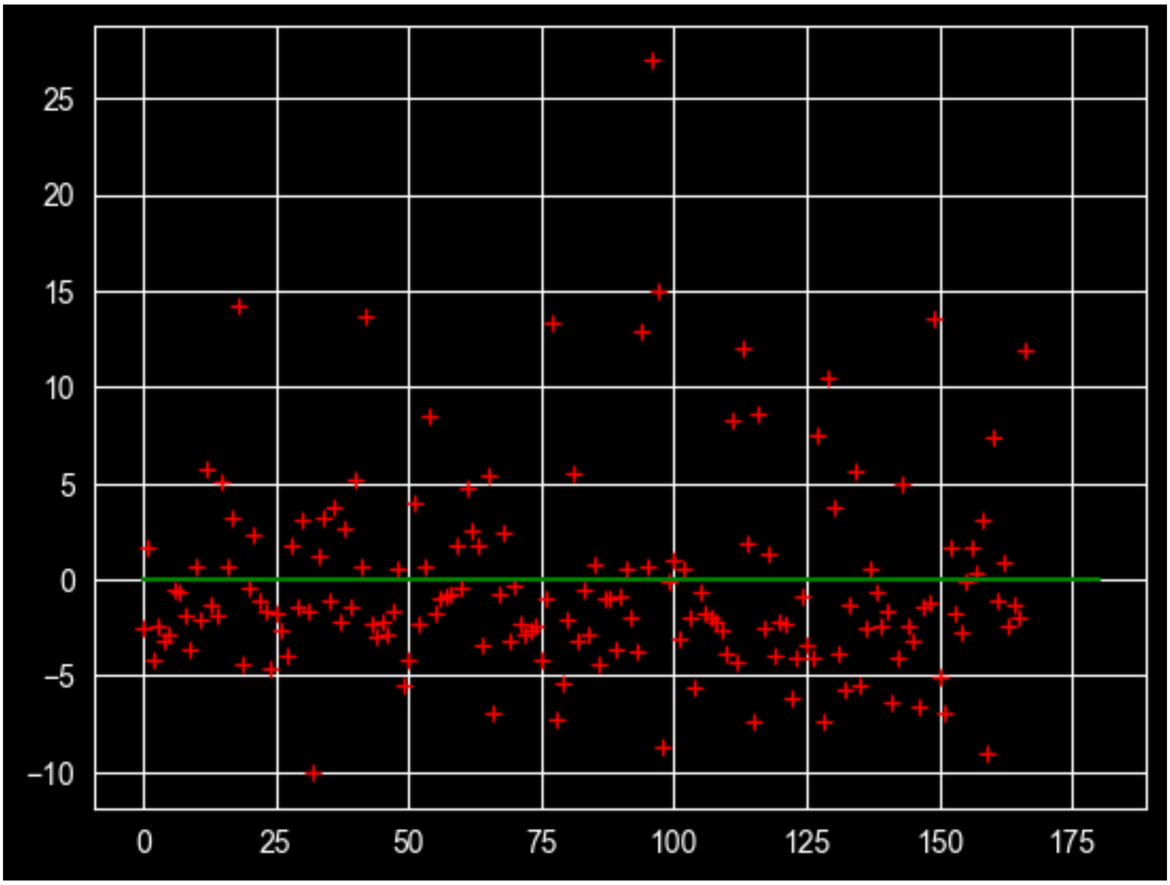
Calculating Erorr/Residuals

In [75]:
```
residuals_lasso = Y_test - reg_pred_lasso
```

Plotting Error

In [76]:
```
plt.style.use("dark_background")
plt.plot(np.arange(residuals_lasso.size), residuals_lasso, "r+")
sns.lineplot([0, 180], [0, 0], color='green')
```

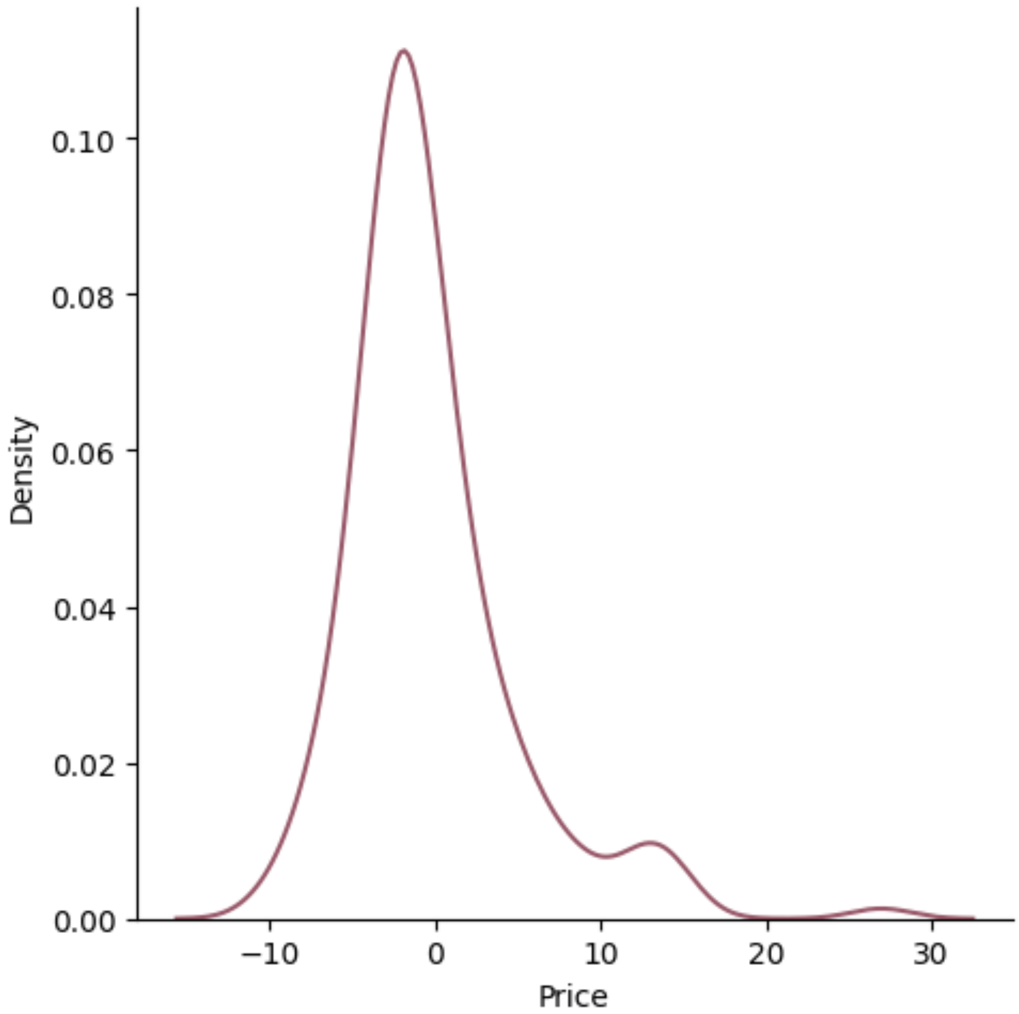
Out[76]: <AxesSubplot:>



Error Distribution

In [77]:
```
plt.style.use('default')
sns.displot(residuals_lasso, kind = "kde", color="#99596a")
```

Out[77]: <seaborn.axisgrid.FacetGrid at 0x1693895a0>

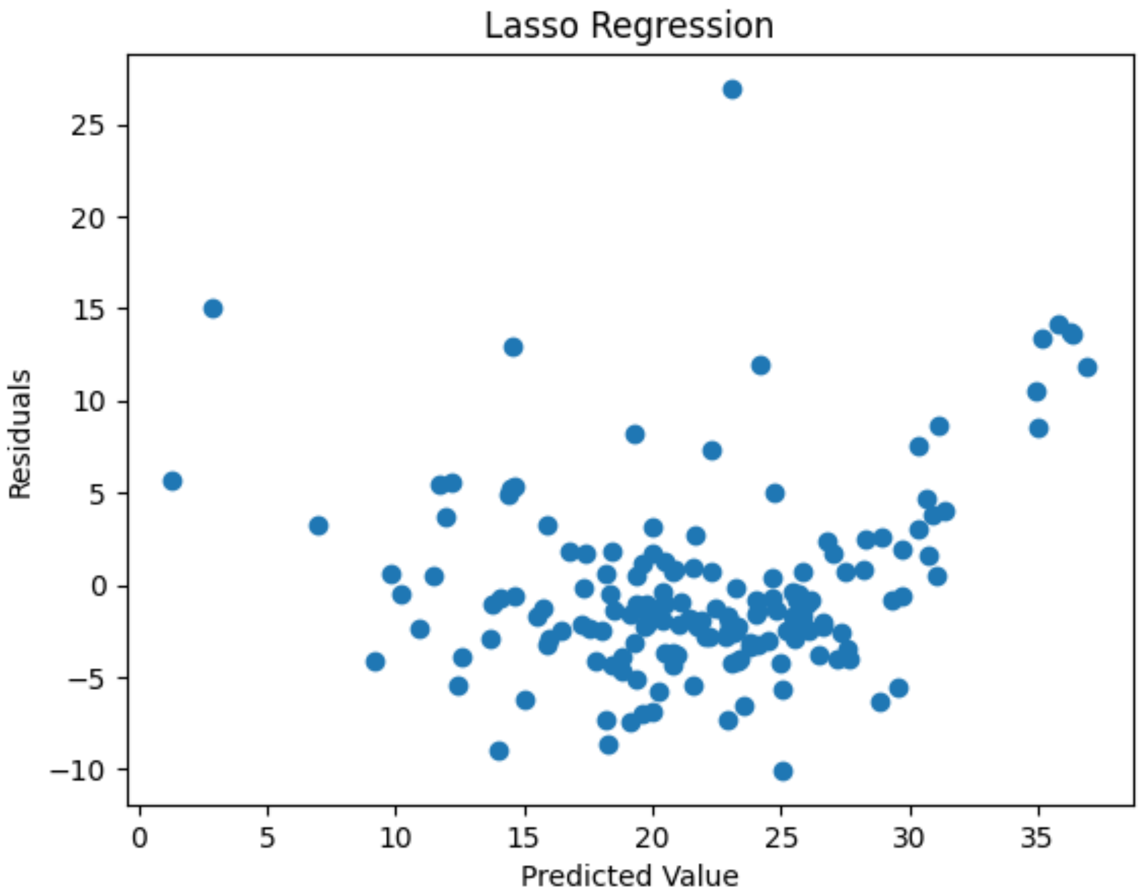


Result : Error distribution is approximately normally distibuted with little right-skewed tail.

Realtion between predicted values & residuals

In [78]:
```
plt.xlabel("Predicted Value")
plt.ylabel("Residuals")
plt.title("Lasso Regression")
plt.scatter(reg_pred_lasso, residuals_lasso)
```

Out[78]: <matplotlib.collections.PathCollection at 0x169477070>



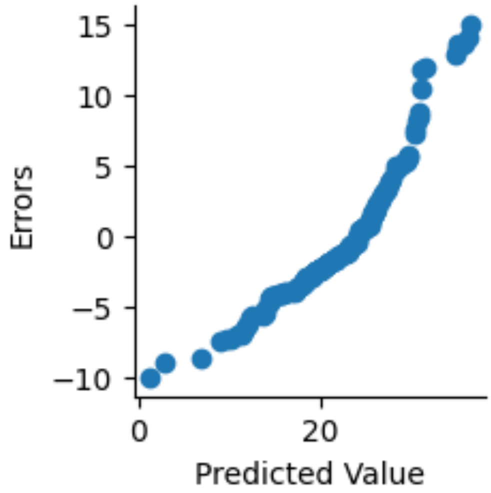
Creation of temporary dataframe

In [79]:
```
temp_df = pd.DataFrame([residuals_lasso.to_list(), reg_pred_lasso
temp_df = temp_df.T
temp_df.rename(columns = {0:"Errors", 1:"Predicted Value"},inplac
```

Plotting Quantile-Quantile Graph

In [80]:
```
pplot(data = temp_df, x = "Predicted Value", y = "Errors", kind="
```

Out[80]:
```
<seaborn.axisgrid.PairGrid at 0x1693208b0>
```



RESULT : Approximate Linear Distribution between Residuals & Predicted Values

Performance Metric

Total Error Calculated for different Cost Functions

In [81]:
```
print(f"Mean Sqaured Error: {mean_squared_error(Y_test, reg_pred_
print(f"Mean Sqaured Error: {mean_absolute_error(Y_test, reg_pred
print(f"Mean Sqaured Error: {np.sqrt(mean_squared_error(Y_test, r
```

```
Mean Sqaured Error: 26.166378490585732
Mean Sqaured Error: 3.646402767847218
Mean Sqaured Error: 5.115308249811123
```

Calculation of R2 & Adjusted R2

In [82]:
```
r2 = r2_score(Y_test, reg_pred_lasso)
adjusted_r2 = 1 - ((1-r2)*(len(Y_test)-1)/(len(Y_test)-1-X.shape[

print(f"R2 Score: {r2}")
print(f"Adjusted R2 Score: {adjusted_r2}")
```

```
R2 Score: 0.6542429409179247
Adjusted R2 Score: 0.6273008324180227
```

Model Training Using ElasticNet

Importing Libraries

In [83]:
```
from sklearn.linear_model import ElasticNet
```

In [84]:
```
en_regression = ElasticNet()
en_regression
```

Out[84]:
```
▾ ElasticNet
ElasticNet()
```

In [85]:
```
en_regression.fit(X_train, Y_train)
```

Out[85]:
```
▾ ElasticNet
ElasticNet()
```

Coefficients & Intercept of Elastic Net

In [86]:
```
en_regression.coef_
```

Out[86]:
```
array([-0.36520444,  0.        , -0.14337534,  0.63145886, -0.25
19316 ,
        2.34999332, -0.        , -0.        , -0.25649587, -1.23
951413,
        0.56384767, -2.56053065])
```

In [87]:
```
en_regression.intercept_
```

Out[87]:
```
22.970796460176988
```

Prediction for the test-data

In [88]:
```
reg_pred_en = en_regression.predict(X_test)
```

In [89]:
```
reg_pred_en
```

Out[89]:
```
array([26.04803111, 31.11448316, 18.09844331, 24.74715333, 19.13
029595,
       23.07194954, 19.84921427, 16.42920945, 20.9828073 , 21.03
041223,
       23.59247312, 22.40671604,  2.50341532, 22.86968815, 21.05
836859,
       23.53088527, 19.32940329,  9.24659647, 34.51755335, 18.33
111694,
       25.39963915, 26.53220578, 16.04213063, 23.68594856, 18.22
```

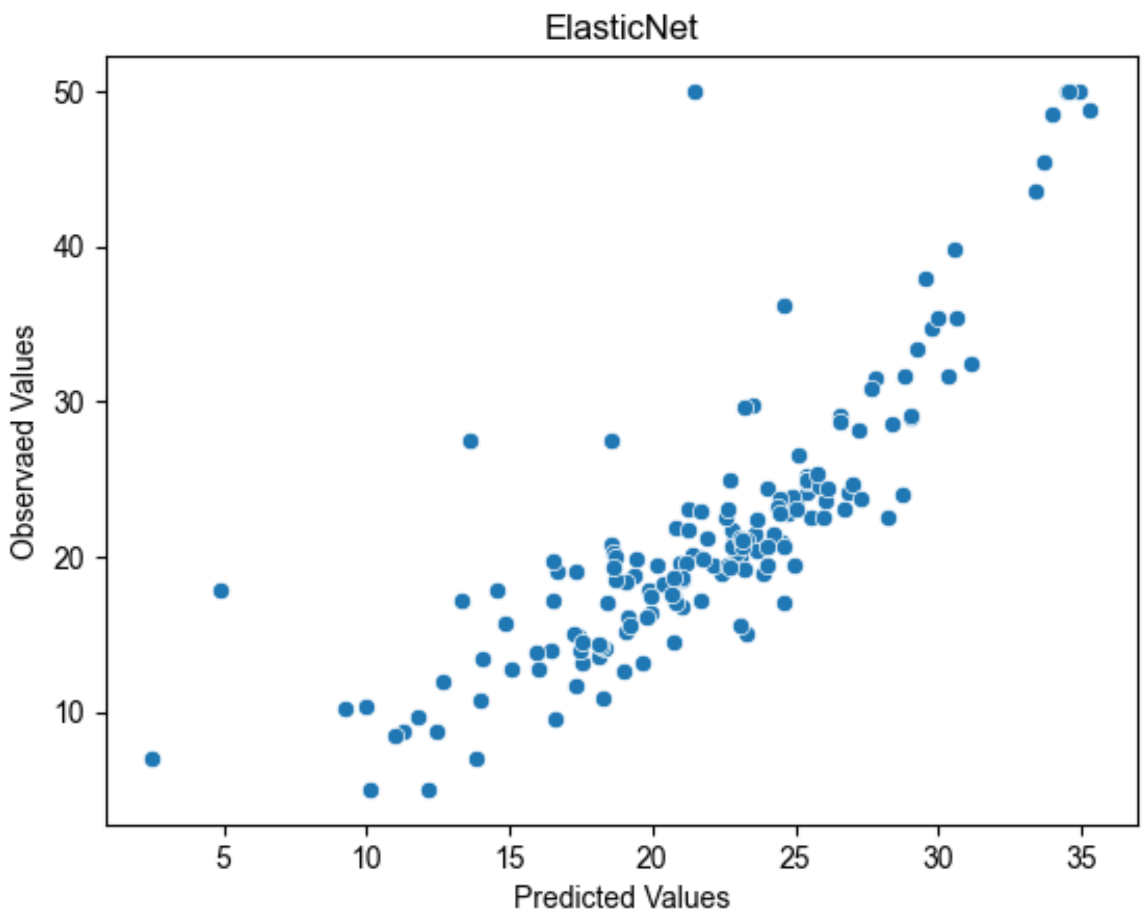
```
309554,
       15.90700742, 22.91791394, 17.40136275, 22.8088184 , 20.34
960421,
       21.28107928, 25.06647537, 23.2904073 , 18.52289617, 16.68
946882,
       20.1709906 , 29.7800025 , 22.08911865, 24.00624595, 24.52
109479,
       16.51540302, 27.2514299 , 34.89409579, 20.7523046 , 25.54
944873,
       17.27877557, 17.51068539, 25.42247325, 19.45141707, 28.72
44585 ,
       23.85816468, 30.64335485, 19.0577923 , 25.10137502, 33.43
673403,
       21.9368308 , 19.10068193, 28.38705951, 24.91075888, 18.68
821948,
       25.41736175, 29.96236683, 27.77368508, 18.6607811 , 26.83
457792,
       18.72984705, 19.66634603, 25.37569807, 27.64863265, 15.09
327629,
       21.62306487, 24.42217938, 14.00935252, 22.80342027, 23.31
056925,
       10.15386961, 21.4127077 , 33.98445079, 18.23197212, 13.83
629603,
       23.23841037, 13.33916112, 24.55298387, 11.80396482, 22.60
393214,
       29.04778391, 19.93865061, 25.40796729, 25.42536938, 20.79
625741,
       24.35938278,  9.98031505, 21.18830517, 21.70103385, 13.64
158565,
       21.70328121, 21.48779629,  4.89921861, 16.60651591, 16.48
691602,
       22.52663318, 24.23810543, 12.67234318, 21.73289084, 24.94
869962,
       14.02785889, 20.34560556, 25.81816974, 23.27665993, 26.98
968905,
       12.44611054, 18.53919211, 24.57037171, 24.59912077, 28.78
917397,
       17.35695972, 30.55891094, 17.49670466, 20.81636309, 27.26
669036,
       20.67057176, 26.10145646, 11.29180365, 23.22360552, 25.76
790497,
       23.62200371, 26.68355142, 29.5350619 , 23.09099057, 33.69
060211,
       14.86233006, 26.00701142, 20.72315145, 21.04261968, 14.58
582778,
       19.80159335, 23.1654195 , 30.31837693, 29.06727967, 19.21
665446,
       19.9643156 , 28.26947732, 24.0439545 , 18.60022888, 10.97
390803,
       23.98833426, 24.56611086, 18.41001381, 17.565798  , 34.56
534038,
       18.1200366 , 19.01049707, 26.55906535, 23.10161286, 22.69
122244,
```

```
        22.66142973, 17.29393051, 22.68213492, 29.22757274, 12.13
999205,
        23.24526047, 20.77586796, 21.27149077, 25.02037963, 24.44
502066,
        23.10652896, 35.30972894])
```

Plotting predicted value vs real values

In [90]:
```
plt.xlabel("Predicted Values")
plt.ylabel("Observaed Values")
plt.title("ElasticNet")
sns.set_style("whitegrid")
sns.scatterplot(x = reg_pred_en, y=Y_test, palette="plasma", ci=2!
```

Out[90]:
```
<AxesSubplot:title={'center':'ElasticNet'}, xlabel='Predicted Va
lues', ylabel='Observaed Values'>
```



Result : Almost Linear Relation between Predicted Values & Target Values.

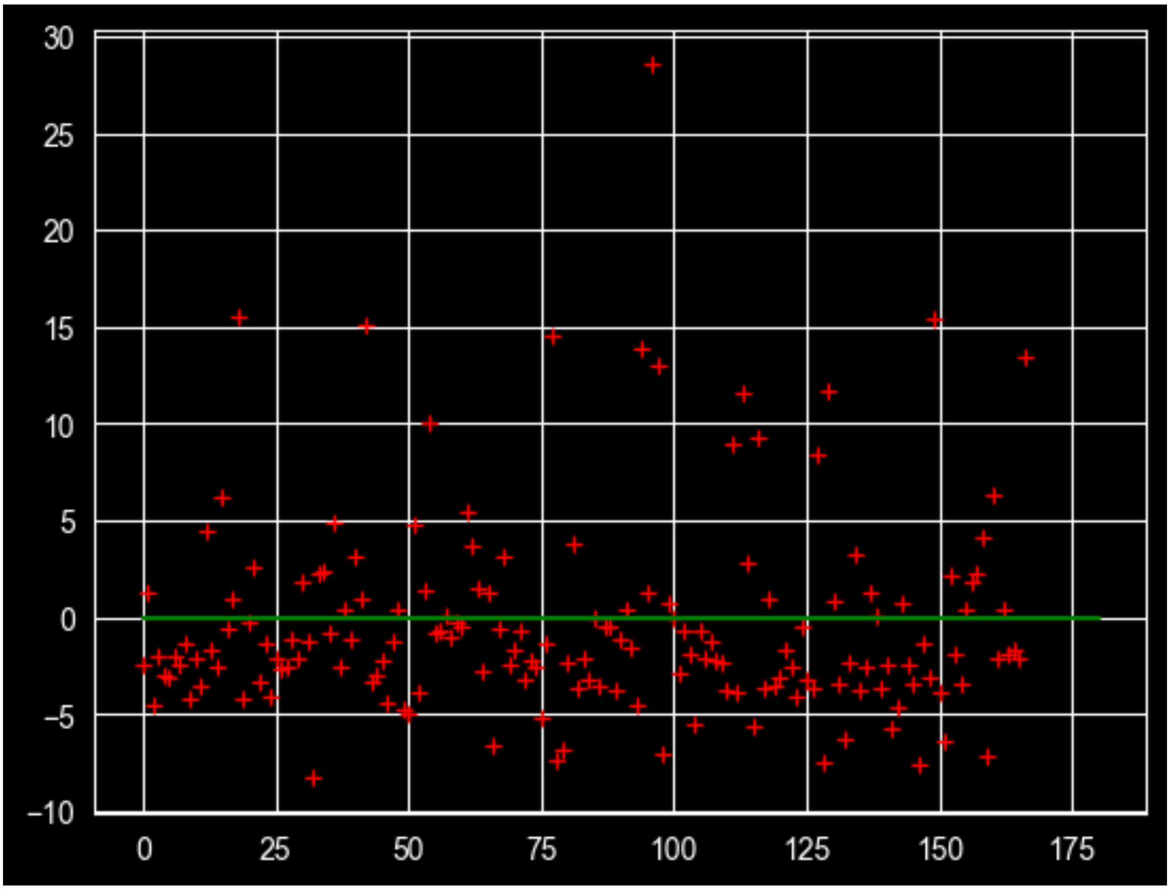
Calculating Erorr/Residuals

In [91]:
```
residuals_en = Y_test - reg_pred_en
```

Plotting Error

In [92]:
```
plt.style.use("dark_background")
plt.plot(np.arange(residuals_en.size), residuals_en, "r+")
sns.lineplot([0, 180], [0, 0], color='green')
```

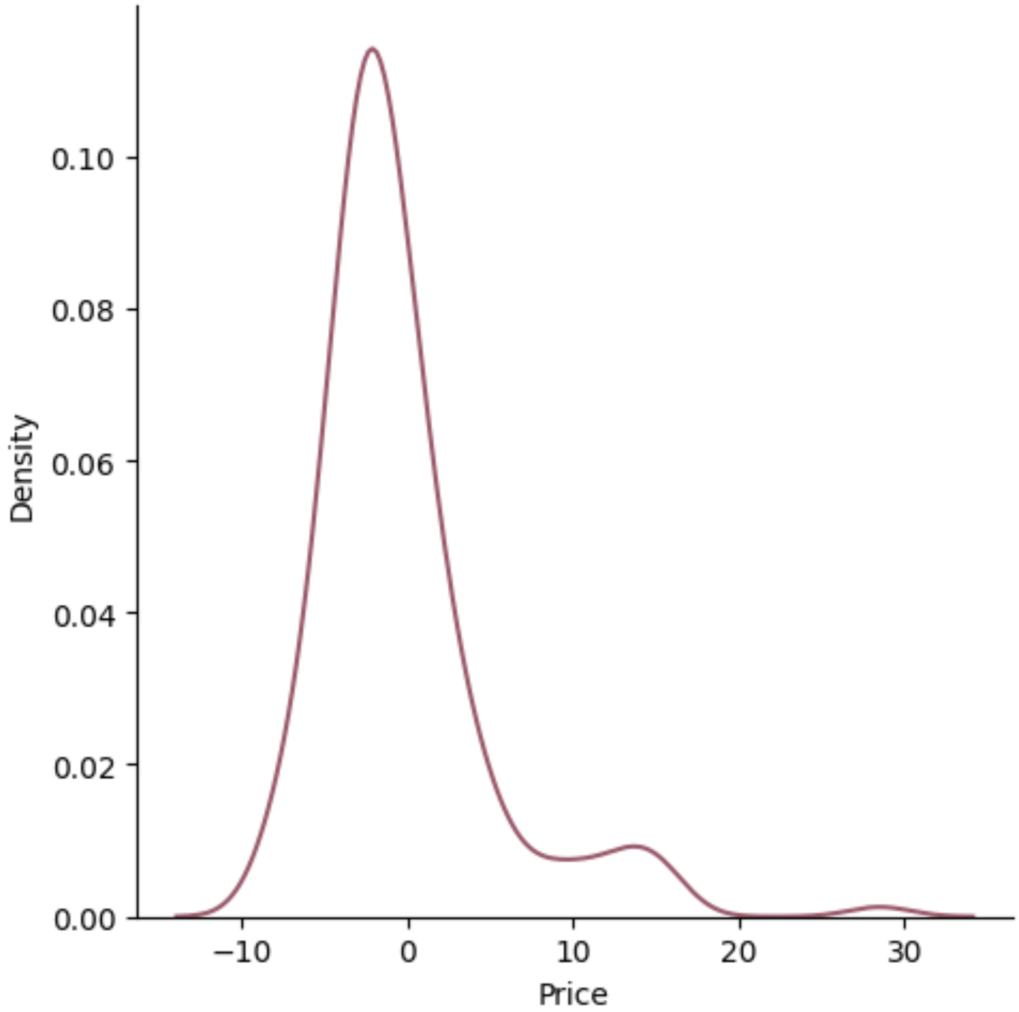
Out[92]: <AxesSubplot:>



Error Distribution

In [93]:
```
plt.style.use('default')
sns.displot(residuals_en, kind = "kde", color="#99596a")
```

Out[93]: <seaborn.axisgrid.FacetGrid at 0x169571c30>

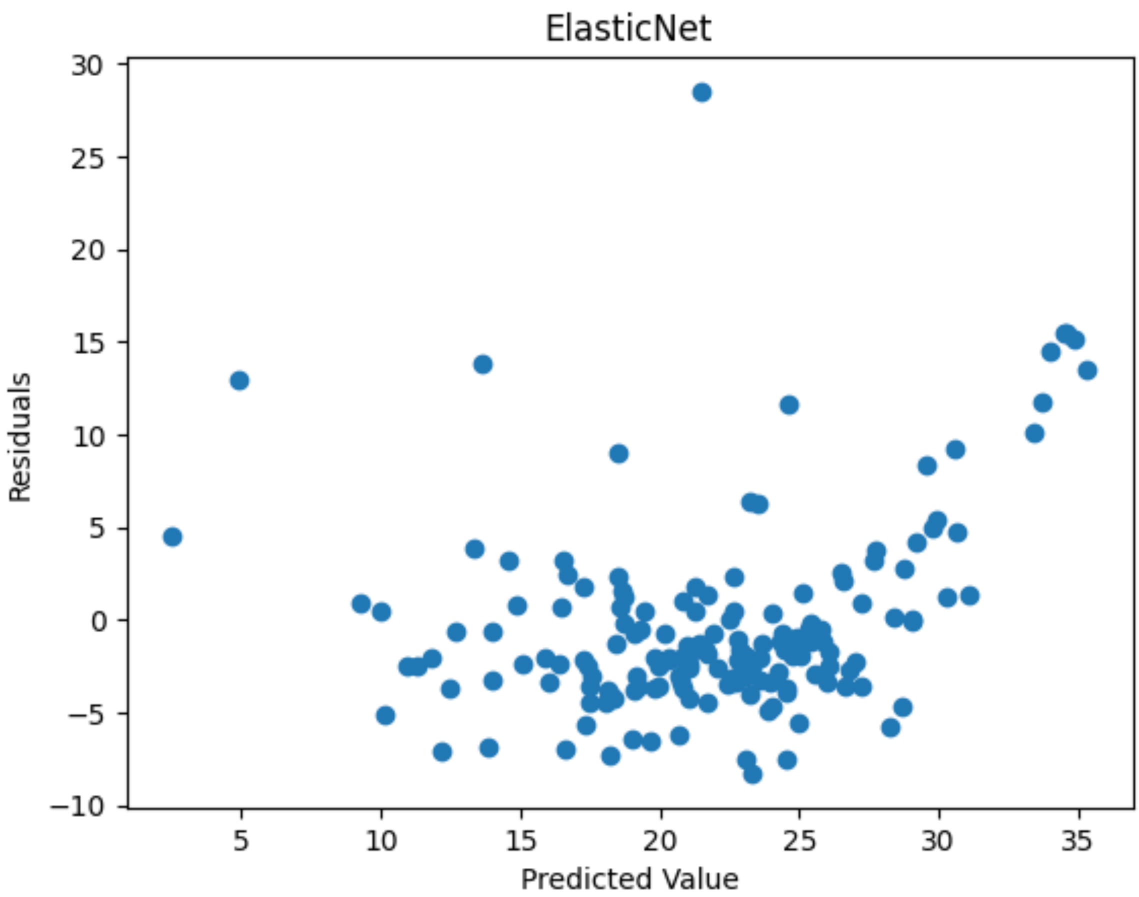


Result : Error distribution is approximately normally distibuted with little right-skewed tail.

Realtion between predicted values & residuals

In [94]:
```
plt.xlabel("Predicted Value")
plt.ylabel("Residuals")
plt.title("ElasticNet")
plt.scatter(reg_pred_en, residuals_en)
```

Out[94]: <matplotlib.collections.PathCollection at 0x16969de10>



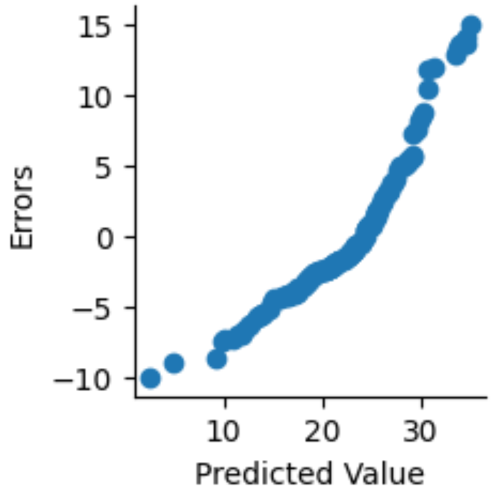
Creation of temporary dataframe

In [95]:
```
temp_df = pd.DataFrame([residuals_lasso.to_list(), reg_pred_en.to
temp_df = temp_df.T
temp_df.rename(columns = {0:"Errors", 1:"Predicted Value"},inplac
```

Plotting Quantile-Quantile Graph

In [96]:
```
pplot(data = temp_df, x = "Predicted Value", y = "Errors", kind="
```

Out[96]: <seaborn.axisgrid.PairGrid at 0x1695ef3a0>



RESULT : Approximate Linear Distribution between Residuals & Predicted Values

Performance Metric

Total Error Calculated for different Cost Functions

In [97]:
```
print(f"Mean Sqaured Error: {mean_squared_error(Y_test, reg_pred_
print(f"Mean Sqaured Error: {mean_absolute_error(Y_test, reg_pred
print(f"Mean Sqaured Error: {np.sqrt(mean_squared_error(Y_test, r
```

```
Mean Sqaured Error: 27.140174496152532
Mean Sqaured Error: 3.62774535074285
Mean Sqaured Error: 5.209623258562228
```

Calculation of R2 & Adjusted R2

In [98]:
```
r2 = r2_score(Y_test, reg_pred_en)
adjusted_r2 = 1 - ((1-r2)*(len(Y_test)-1)/(len(Y_test)-1-X.shape[

print(f"R2 Score: {r2}")
print(f"Adjusted R2 Score: {adjusted_r2}")
```

```
R2 Score: 0.6413754039314139
Adjusted R2 Score: 0.6134306302117838
```

Conclusion **:** As evident from R2 & Adjusted R2 metrics Ridge Regression Model has performed better as compared to other models. Hence this model will be recommended.