In [1]:
```
# Importing the Libraries

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
sns.set()

import plotly
from plotly.offline import plot, iplot, init_notebook_mode, download_plotlyjs
init_notebook_mode(connected = True)

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.feature_selection import f_regression
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor
from sklearn.svm import SVR
from xgboost import XGBRegressor
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error

import scipy.stats as stats

import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.outliers_influence import variance_inflation_factor

from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as smf

import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
# Reading the data

data = pd.read_csv("boston_house_prices.csv")

# Creating a copy of the data

df = data.copy()
```

Basic EDA

In [3]:
```
# Checking the first five records

df.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

In [4]:
```
# Checking the last five records

df.tail()
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

In [5]:
```
# Checking five random records

df.sample(5)
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
165	2.92400	0.0	19.58	0	0.605	6.101	93.0	2.2834	5	403	14.7	240.16	9.81	25.0
162	1.83377	0.0	19.58	1	0.605	7.802	98.2	2.0407	5	403	14.7	389.61	1.92	50.0
70	0.08826	0.0	10.81	0	0.413	6.417	6.6	5.2873	4	305	19.2	383.73	6.72	24.2
190	0.09068	45.0	3.44	0	0.437	6.951	21.5	6.4798	5	398	15.2	377.68	5.10	37.0
408	7.40389	0.0	18.10	0	0.597	5.617	97.9	1.4547	24	666	20.2	314.64	26.40	17.2

In [6]:
```
# Checking the shape of the data

df.shape
```

Out[6]:
```
(506, 14)
```

In [7]:
```
# Checking the features

df.columns
```

Out[7]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

In [8]:
```
# Checking the datatypes of the features

df.dtypes.sort_values()
```

Out[8]:
```
CHAS         int64
RAD          int64
TAX          int64
CRIM       float64
ZN         float64
INDUS      float64
NOX        float64
RM         float64
AGE        float64
DIS        float64
PTRATIO    float64
B          float64
LSTAT      float64
MEDV       float64
dtype: object
```

In [9]:
```
# Checking the metadata of the dataset

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    int64  
 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    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [10]:
```
# Checking the number of null entries in the data

print('\033[1m' + f"The total number of null values in the dataset : {df.isna().sum().sum()}\n"

if df.isna().sum().sum() != 0:
    null_col = [col for col in df.columns if col]
    pd.DataFrame({
        "Feature" : [col for col in null_col],
        "Null_Count" : [df[col].isna().sum() for col in null_col],
        "Null_Percentage" : [str(np.round((df[col].isna().sum() / len(df)) * 100, 2)) + " %" for
        "Data_Type" : [df[col].dtype for col in null_col]
    }).sort_values("Null_Count", ascending = False).set_index("Feature")
else:
    pass
```

The total number of null values in the dataset : 0

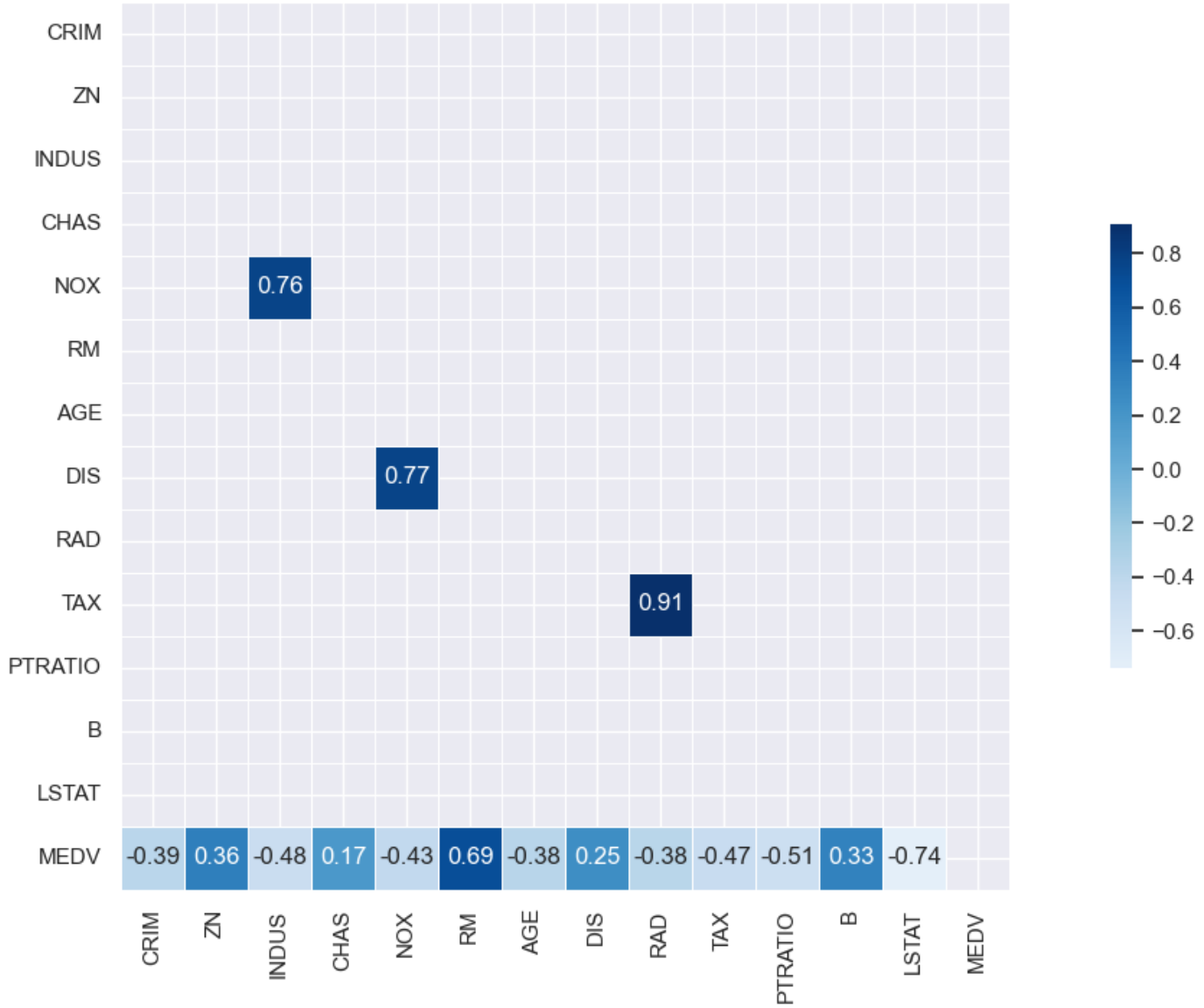
In [11]:
```
# Checking the statistical summary of the dataset

df.describe()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000

In [12]:

```
# Checking the correlation

def correlation(data):
    plt.figure(figsize = (18, 8))
    corr = np.round(abs(data.corr()))[abs(data.corr()).iloc[:-1, :-1] > 0.75].fillna({"MEDV" : da
    mask = np.triu(np.ones_like(corr, dtype = bool))
    sns.heatmap(corr, mask = mask, cmap = "Blues", annot = True, center = 0, square = True, line
    sns.set(style = "darkgrid")
    plt.show()

correlation(data)
```



The feature "NOX" is highly correlated, so they will be dropped during pre-processing.

Since "RAD" is a categorical feature, we cannot calculate its correlation with numerical features.

In [13]:
```
# Creating a function to check the count of duplicated records and drop them

def duplicate_data(data):

    print('\033[1m' + f"The total number of duplicate records in the dataset : {data.duplicated(

    if data.duplicated().sum() != 0:
        data.drop_duplicates(inplace = True)
    else:
        pass

duplicate_data(df)
```

The total number of duplicate records in the dataset : 0

In [14]:
```
# Checking the count of unique values in each of the feature

pd.DataFrame({
    "Feature" : [col for col in df.columns],
    "Unique_Values" : [df[col].nunique() for col in df.columns],
    "Data_Type" : [df[col].dtype for col in df.columns]
}).set_index("Feature").sort_values("Unique_Values")
```

Out[14]:

Feature	Unique_Values	Data_Type
CHAS	2	int64
RAD	9	int64
ZN	26	float64
PTRATIO	46	float64
TAX	66	int64
INDUS	76	float64
NOX	81	float64
MEDV	229	float64
AGE	356	float64
B	357	float64
DIS	412	float64
RM	446	float64
LSTAT	455	float64
CRIM	504	float64

The features "CHAS" needs to be converted to categorical type during pre-processing.

In [15]:
```
# Checking the unique values in each of the feature

# for col in df.columns:

#     print('\033[1m' + f"The {col} features consists of {df[col].nunique()} unique values, whic
#     print(f"{[val for val in df[col].unique()]}\n")
```

In [16]:
```
# Checking the value count in each of the feature

# for col in df.columns:

#     print('\n\033[1m' + f'value counts on "{col}"' + '\033[0m\n')
#     for v, c in df[col].value_counts().items():
#         print(f"{v} : {c}")
```

Pre-Processing

In [17]:
```
def transformation(data):

    # converting the "RAD" and "CHAS" feature to object type
    data["RAD"] = data["RAD"].astype("O")
    data["CHAS"] = data["CHAS"].astype("O")

#     # converting the "RM" feature to int type
#     data["RM"] = data["RM"].astype("int")
#     # converting the "RM" feature to categorical type
#     data["RM"] = data["RM"].astype("object")

    # dropping the correlated feature "NOX"
    data.drop(columns = ["NOX"], inplace = True)
```

```
    # dropping the non-significant feature "B"
    data.drop(columns = ["B"], inplace = True)

    # transforming the "MEDV" feature
    data["MEDV"] = data["MEDV"] * 1000

    # transforming the "NOX" feature
#     data["NOX"] = data["NOX"] * 100

transformation(df)
```

Exploratory Data Analysis (EDA)

Univariate Analysis (Numerical Features)

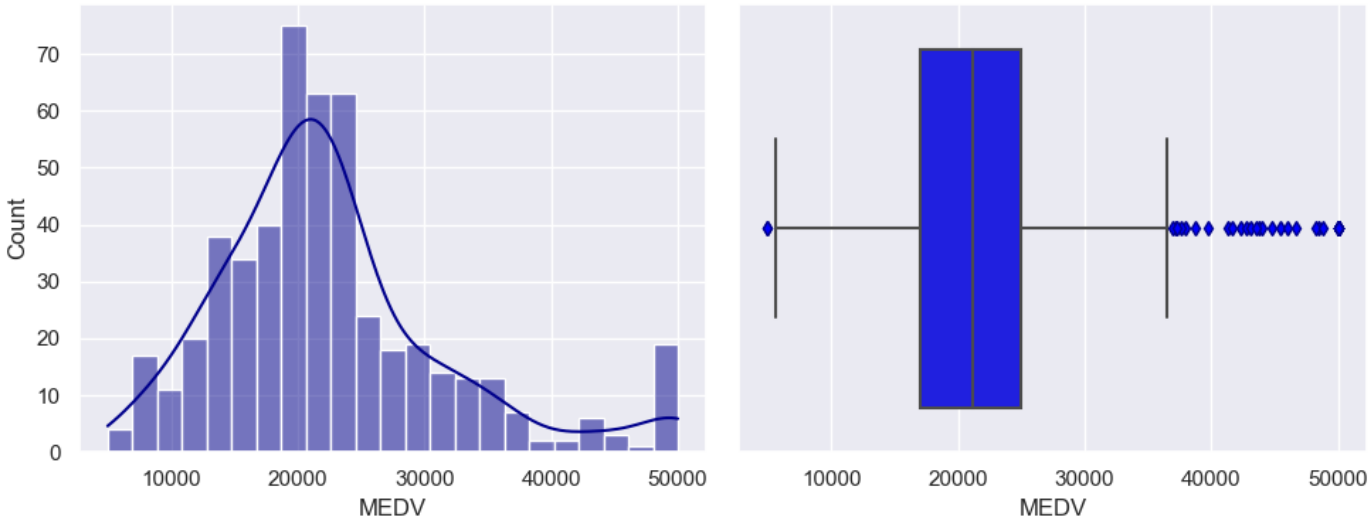
In [18]:
```
# Creating a function to perform Univariate Analysis on Numerical Features

def univariate_eda_num(data, col):

    print('\033[1m' + f"\t\t\t\t\t\t\t{col}" + '\033[0m\n')

    # checking the statistical summary of the feature
    print('\033[1m' + "Statistical Summary" + '\033[0m\n')
    stat_sum = np.round(data[col].describe(), 2)
    stat_sum["skewness"] = np.round(data[col].skew(), 2)

    # checking for outliers using IQR method
    Q1, Q3 = data[col].quantile([0.25, 0.75])
    IQR = Q3 - Q1
    lower_bound = Q1 - (1.5 * IQR)
    upper_bound = Q3 + (1.5 * IQR)
    outlier_count = len(list(data[col][(data[col] < lower_bound) | (data[col] > upper_bound)].va
    stat_sum["outliers"] = outlier_count

    print(f"{pd.DataFrame(stat_sum)}\n")

    # checking the distribution and outliers by plotting graphs
    fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (10, 4))
    sns.histplot(x = data[col], color = "darkblue", ax = axes[0], kde = True)
    sns.boxplot(x = data[col], color = "blue", ax = axes[1], flierprops = {"markerfacecolor" : "
    plt.tight_layout()
```

In [19]:
```
# Exploring the "MEDV" feature

univariate_eda_num(df, "MEDV")
```

MEDV

Statistical Summary

	MEDV
count	506.00
mean	22532.81
std	9197.10
min	5000.00
25%	17025.00
50%	21200.00
75%	25000.00
max	50000.00
skewness	1.11
outliers	40.00



Insights

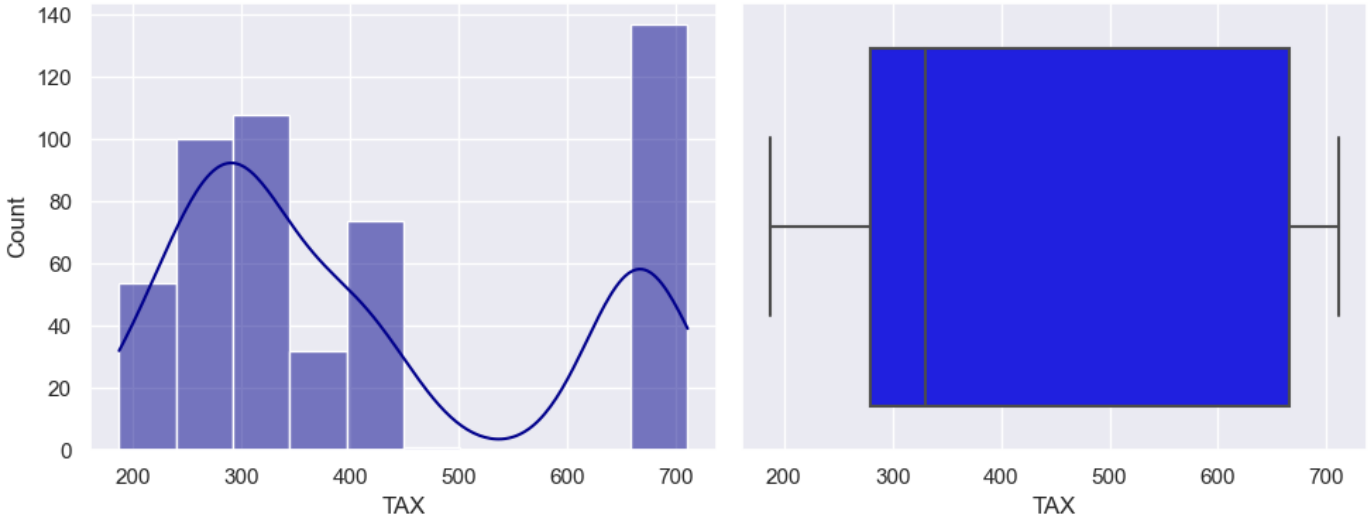
- The average median house value in Boston is worth about $22,532.81.
- The "MEDV" feature has a positively skewed distribution, with most of the outliers on the right side and few on the left side.
- There is a heavy concentration of house prices on the left side of distribution, indicating that 75% of the houses have a median house value below $25,000.

In [20]:
```
# Exploring the "TAX" feature

univariate_eda_num(df, "TAX")
```

TAX

Statistical Summary

	TAX
count	506.00
mean	408.24
std	168.54
min	187.00
25%	279.00
50%	330.00
75%	666.00
max	711.00
skewness	0.67
outliers	0.00



Insights

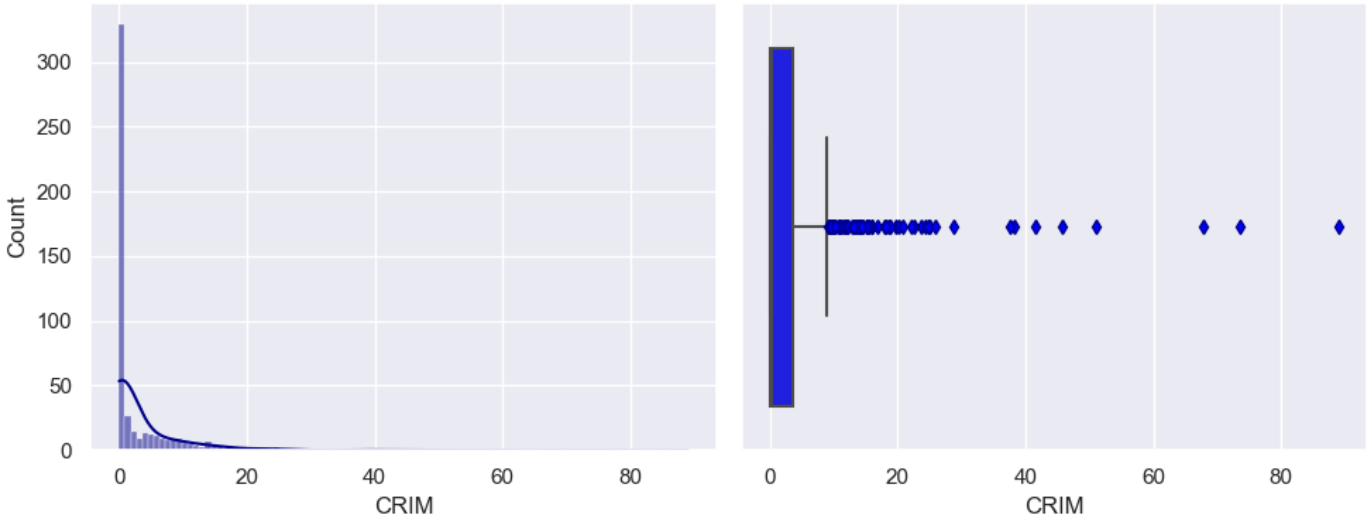
- The average tax rate in Boston is $408.24.
- The "TAX" feature has a positively skewed distribution, but there are no outliers according to the IQR method.
- There is a heavy concentration of tax rates on the left side of the distribution, indicating that 75% of the houses have a tax rate below $666.

In [21]:
```
# Exploring the "CRIM" feature

univariate_eda_num(df, "CRIM")
```

CRIM

Statistical Summary

	CRIM
count	506.00
mean	3.61
std	8.60
min	0.01
25%	0.08
50%	0.26
75%	3.68
max	88.98
skewness	5.22
outliers	66.00



Insights

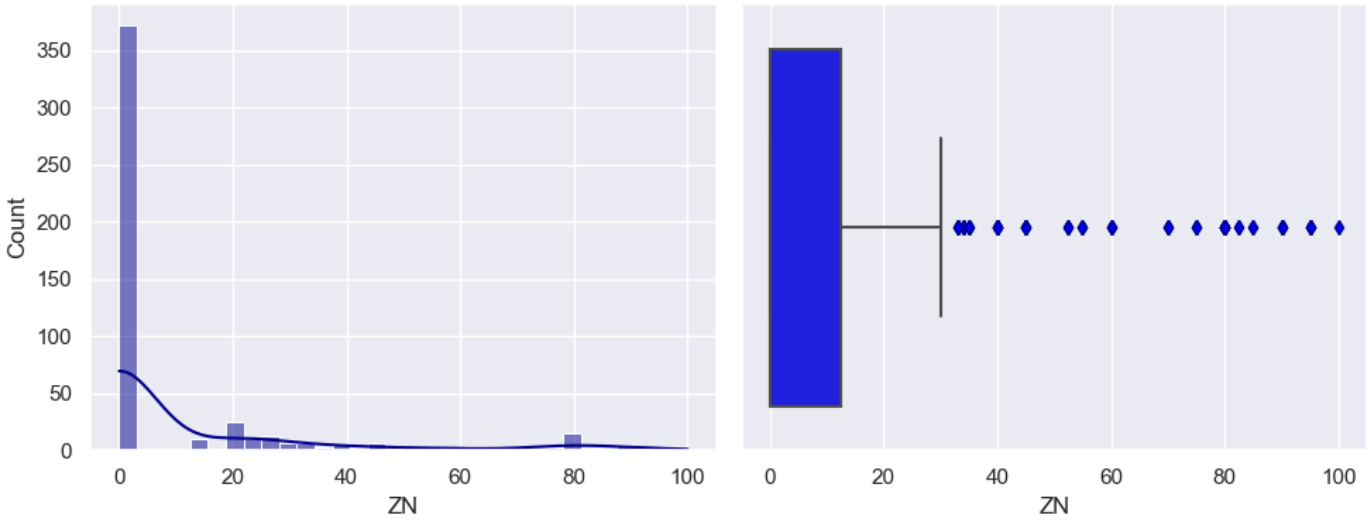
- The average crime rate in Boston is 3.61, which indicates a lower per capita crime rate, suggesting a safer environment.
- The "CRIM" feature has a skewness value of 5.22, which indicates that it is highly positively skewed, with most of the outliers on the right side.

In [22]:
```
# Exploring the feature "ZN"

univariate_eda_num(df, "ZN")
```

ZN

Statistical Summary

	ZN
count	506.00
mean	11.36
std	23.32
min	0.00
25%	0.00
50%	0.00
75%	12.50
max	100.00
skewness	2.23
outliers	68.00



Insights

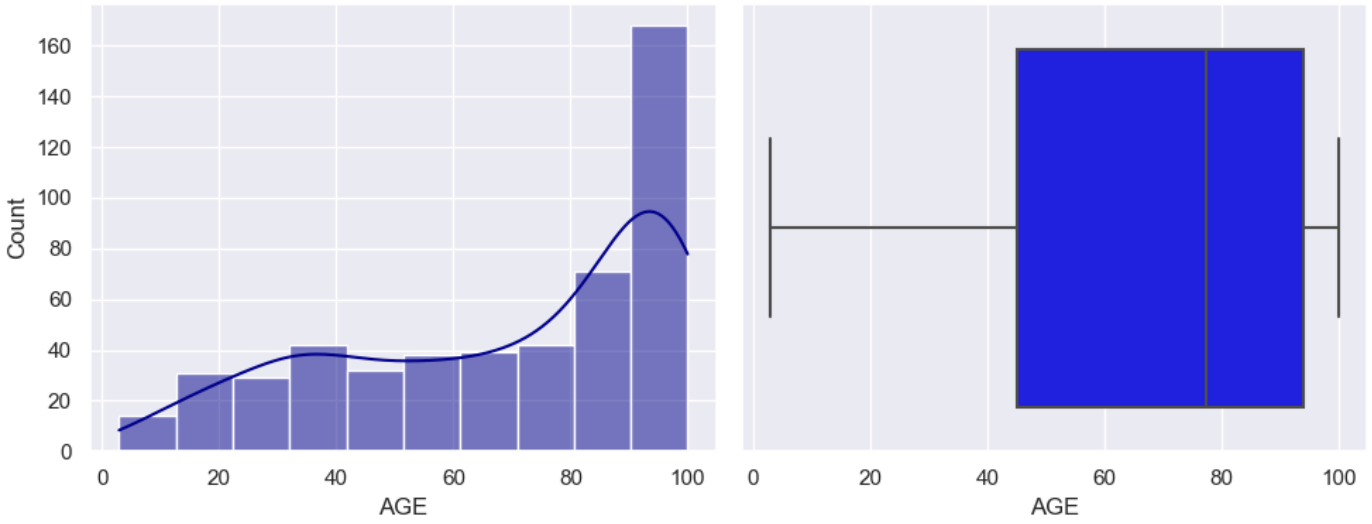
- The average residental land zoned is 11.61, this land helps in Preservation of Green Spaces, Maintaining Privacy and Exclusivity & Addressing Density and Congestion.
- The "ZN" feature has a positively skewed distribution, with most of the outliers on the right side.
- There is heavy concentration of residental land zoned on the left side, indicating that 75% of the houses have a residental land zoned below 12.50.

In [23]:
```
# Exploring the "AGE" feature

univariate_eda_num(df, "AGE")
```

AGE

Statistical Summary

	AGE
count	506.00
mean	68.57
std	28.15
min	2.90
25%	45.02
50%	77.50
75%	94.07
max	100.00
skewness	-0.60
outliers	0.00



Insights

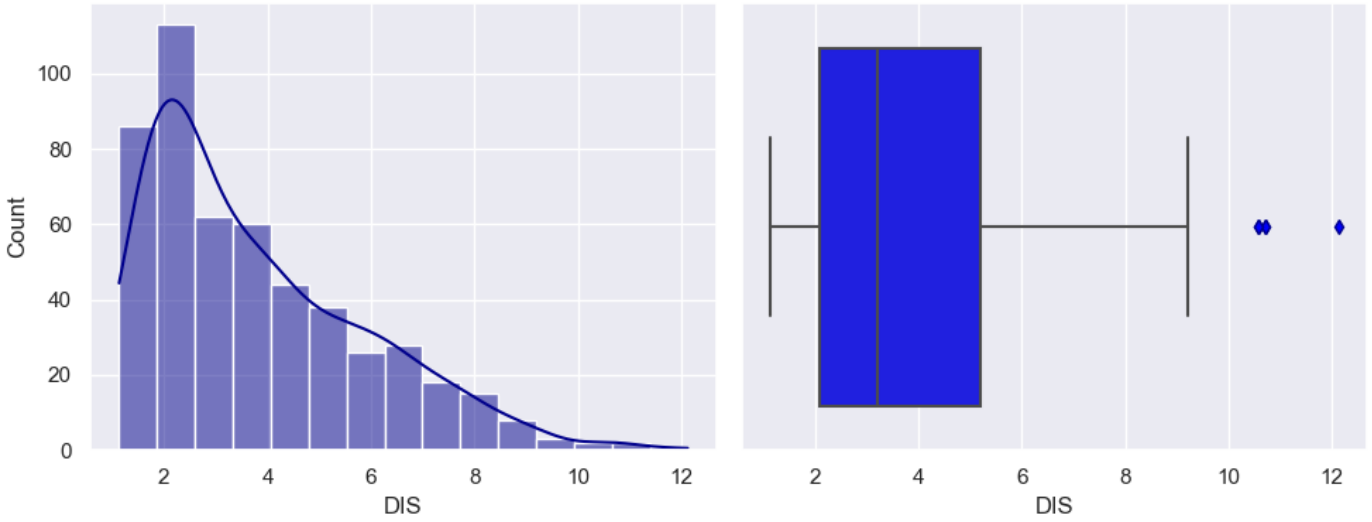
- The average proportion of owner-occupied units built prior to 1940 is 68.57, indicating that the houses were built before 1940 are generally older and less expensive than homes that were built more recently.
- There are no outliers in the "AGE" feature.
- There is a heavy concentration of owner-occupied units on the right side, indicating majority of the houses were built before 1940s.

In [24]:
```
# Exploring the "DIS" feature

univariate_eda_num(df, "DIS")
```

DIS

Statistical Summary

	DIS
count	506.00
mean	3.80
std	2.11
min	1.13
25%	2.10
50%	3.21
75%	5.19
max	12.13
skewness	1.01
outliers	5.00



Insights

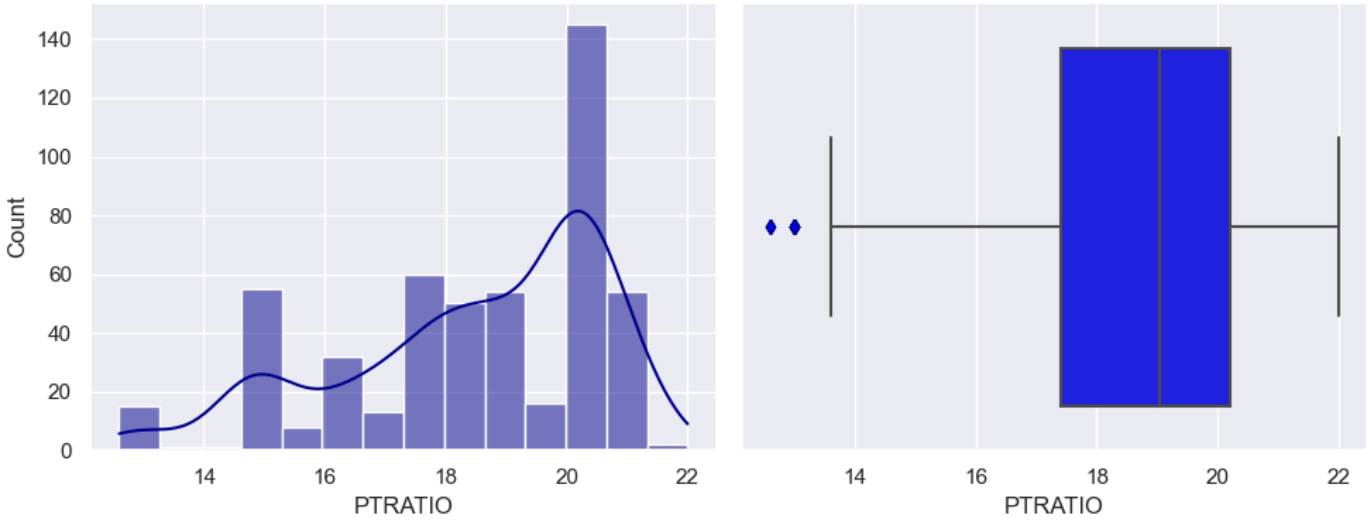
- The average weighted distance between a property to employement centers is 3, indicating most of the towns are closer to employement centers.
- The "DIS" feature has a positively skewed distribution, with a few outliers on both tails.
- There is heavy concentration of weighted distance on the left side of the distribution, indicating 75% of the houses are closer to employment centers.

In [25]:
```
# Exploring the "PTRATIO" feature

univariate_eda_num(df, "PTRATIO")
```

PTRATIO

Statistical Summary

	PTRATIO
count	506.00
mean	18.46
std	2.16
min	12.60
25%	17.40
50%	19.05
75%	20.20
max	22.00
skewness	-0.80
outliers	15.00



Insights

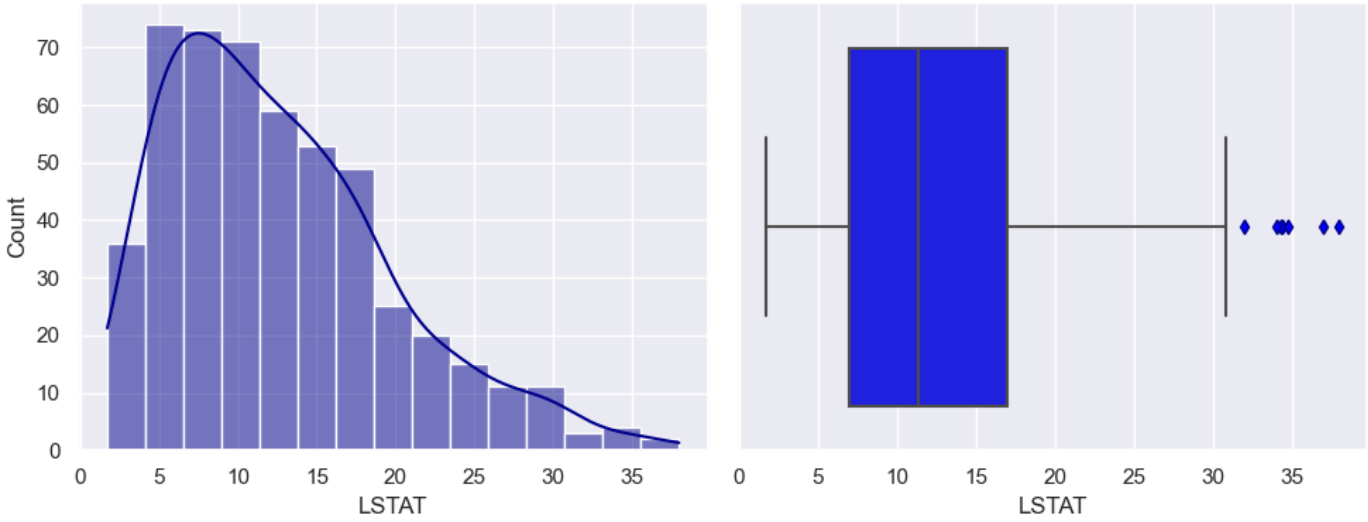
- The average pupil-teacher ratio is 18.46, indicating that there are more teachers available to students, which can lead to smaller class sizes and more individualized attention.
- The "PTRATIO" feature has a negatively skewed distribution, with a few outliers on the left side.
- There is heavy concentration of pupil-teacher ratio on the right side of the distribution, indicating 75% of the houses have the ratio below 20 suggesting there are fewer teachers available to students, which can lead to larger class sizes and less individualized attention.

In [26]:
```
# Exploring the "LSTAT" feature

univariate_eda_num(df, "LSTAT")
```

LSTAT

Statistical Summary

	LSTAT
count	506.00
mean	12.65
std	7.14
min	1.73
25%	6.95
50%	11.36
75%	16.96
max	37.97
skewness	0.91
outliers	7.00



Insights

- The average lower socioeconomic status is 12.65 that is more likely to live in neighborhoods with higher crime rates, lower quality schools, and less access to amenities, all of which can make housing in those neighborhoods less desirable.
- The "LSTAT" feature has a positively skewed distribution, with outliers on the right side.
- There is heavy concentration of lower socioenomic status on the left side of the distribution, indicating 75% of the houses have resident status below 16.96 suggesting lower socioenomic resident.

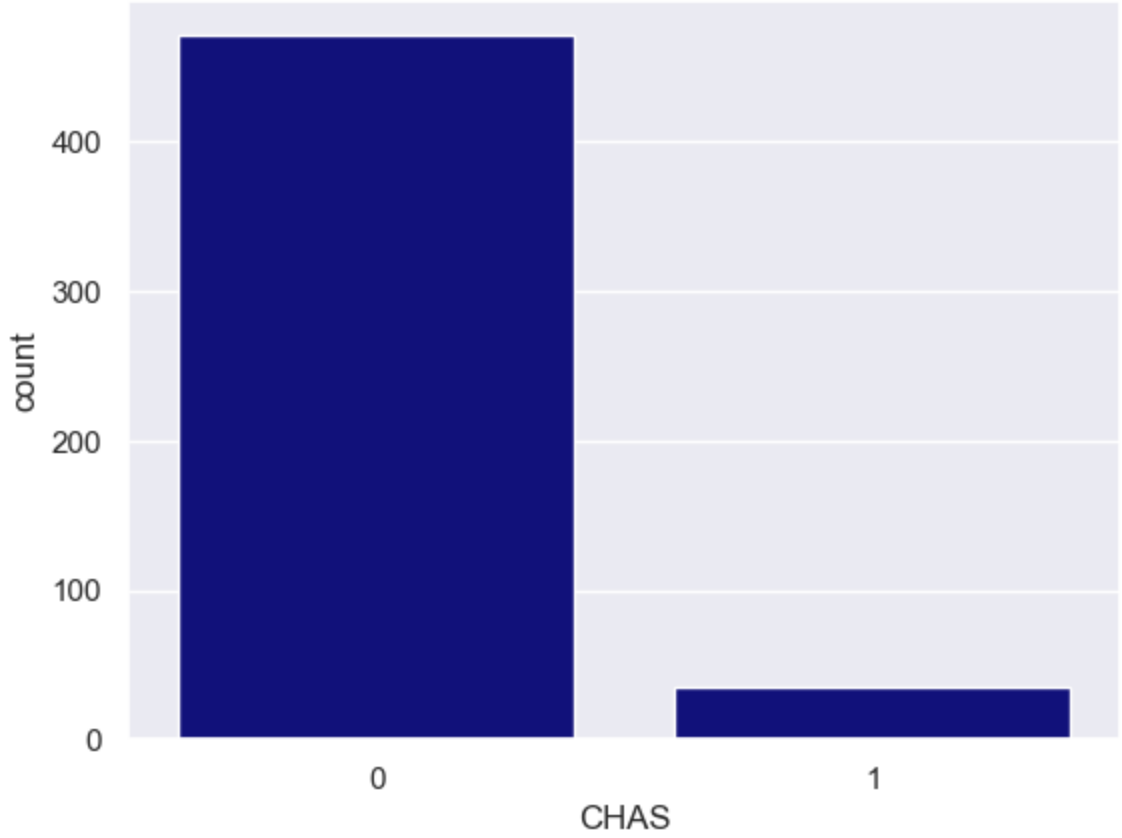
Univariate Analysis (Categorical Features)

In [27]:
```
# Creating a function to perform Univariate analysis on Categorical features

def univariate_eda_cat(data, col):

    # checking the value_counts of the feature
    d = dict(data[col].value_counts(ascending = False))
    print('\033[1m' + f'The "{col}" feature consists of {data[col].nunique()} unique values, whi

    # plotting the distribution across categories
    sns.countplot(x = data[col], color = "darkblue")
```

In [28]:
```
# Exploring the "CHAS" feature

univariate_eda_cat(df, "CHAS")
```

The "CHAS" feature consists of 2 unique values, which are as follows :

```
{0: 471, 1: 35}
```



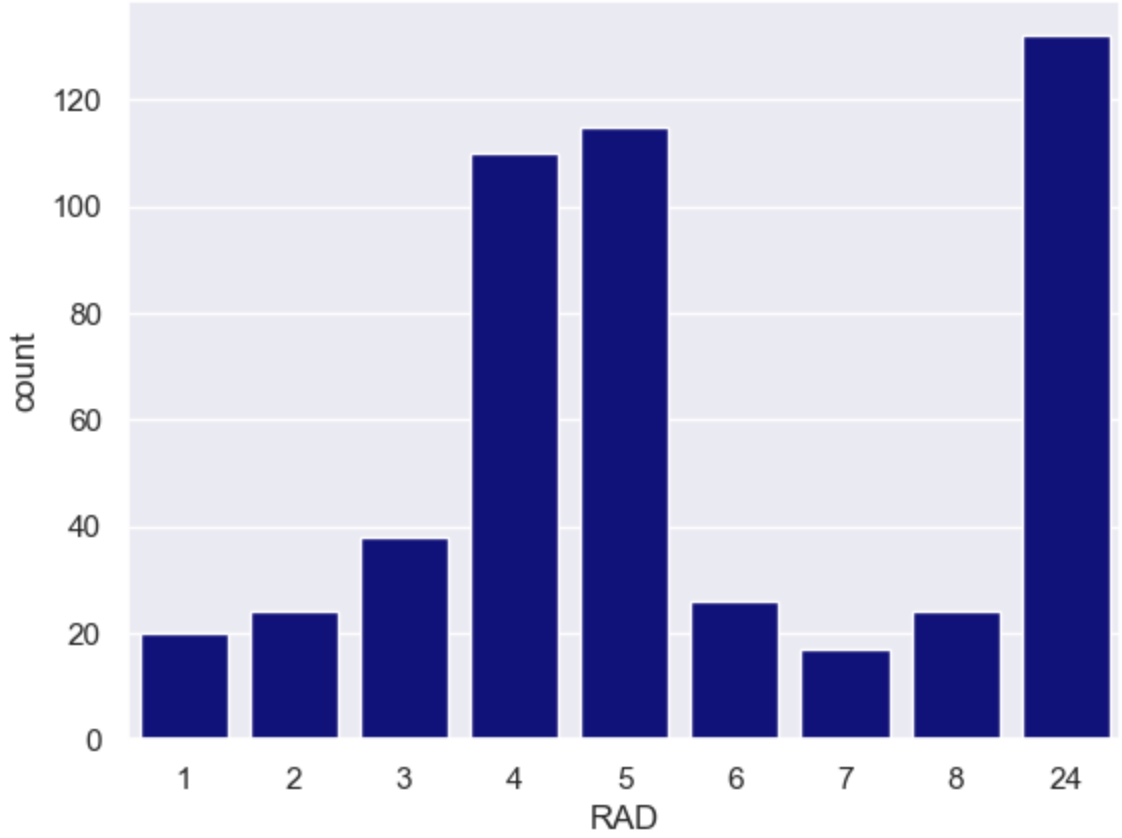
Insights

- Most houses in Boston do not have the Charles River as a boundary is due to higher costs associated having a scenic view, cooler summers and milder winters.

In [29]:
```
# Exploring the "RAD" feature

univariate_eda_cat(df, "RAD")
```

The "RAD" feature consists of 9 unique values, which are as follows :

{24: 132, 5: 115, 4: 110, 3: 38, 6: 26, 2: 24, 8: 24, 1: 20, 7: 17}



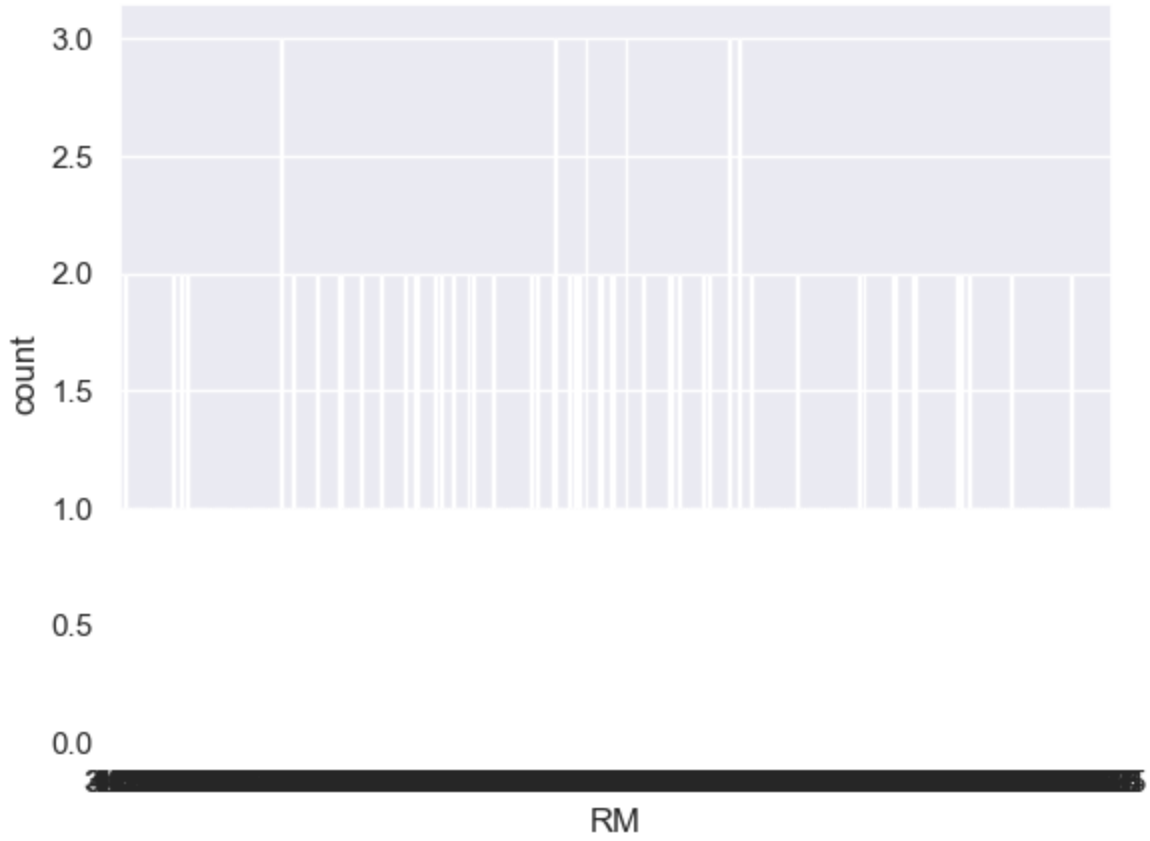
Insights

- Houses in Boston with an index value of 24 are more likely to be located closer to radial highways.

In [30]:
```
# Exploring the "RM" feature

univariate_eda_cat(df, "RM")
```

The "RM" feature consists of 446 unique values, which are as follows :

{5.713: 3, 6.167: 3, 6.127: 3, 6.229: 3, 6.405: 3, 6.417: 3, 6.782: 2, 6.951: 2, 6.63: 2, 6.312: 2, 6.38: 2, 7.82: 2, 5.304: 2, 6.727: 2, 6.376: 2, 6.162: 2, 6.635: 2, 5.888: 2, 6.185: 2, 6.144: 2, 6.315: 2, 6.98: 2, 6.211: 2, 6.122: 2, 5.404: 2, 6.152: 2, 5.936: 2, 4.138: 2, 6.728: 2, 5.757: 2, 6.794: 2, 6.326: 2, 6.431: 2, 5.854: 2, 5.856: 2, 5.39: 2, 5.875: 2, 6.193: 2, 6.03: 2, 5.961: 2, 6.251: 2, 5.966: 2, 6.209: 2, 5.813: 2, 6.495: 2, 6.968: 2, 5.926: 2, 5.935: 2, 6.108: 2, 6.096: 2, 6.009: 2, 6.004: 2, 5.983: 2, 7.185: 2, 7.206: 1, 7.203: 1, 6.395: 1, 5.968: 1, 5.985: 1, 6.112: 1, 6.059: 1, 7.47: 1, 5.895: 1, 5.92: 1, 5.56: 1, 5.869: 1, 6.24: 1, 6.037: 1, 6.538: 1, 6.398: 1, 6.31: 1, 6.212: 1, 7.241: 1, 7.52: 1, 8.398: 1, 5.884: 1, 7.327: 1, 7.014: 1, 6.579: 1, 6.49: 1, 5.663: 1, 6.939: 1, 6.516: 1, 8.297: 1, 5.898: 1, 6.758: 1, 6.014: 1, 6.874: 1, 6.696: 1, 6.54: 1, 7.691: 1, 6.565: 1, 6.316: 1, 6.266: 1, 4.973: 1, 5.972: 1, 7.088: 1, 6.849: 1, 7.42: 1, 6.616: 1, 7.236: 1, 6.982: 1, 6.59: 1, 6.871: 1, 6.453: 1, 7.041: 1, 6.345: 1, 5.79: 1, 6.549: 1, 6.23: 1, 6.678: 1, 7.148: 1, 6.861: 1, 6.023: 1, 6.567: 1, 6.031: 1, 5.705: 1, 5.706: 1, 5.868: 1, 6.854: 1, 6.083: 1, 7.267: 1, 6.826: 1, 6.482: 1, 6.812: 1, 6.415: 1, 5.708: 1, 6.041: 1, 6.426: 1, 6.113: 1, 6.382: 1, 5.782: 1, 5.914: 1, 5.362: 1, 7.645: 1, 7.923: 1, 6.333: 1, 6.575: 1, 5.803: 1, 6.459: 1, 6.701: 1, 6.081: 1, 6.301: 1, 5.976: 1, 6.525: 1, 7.393: 1, 6.297: 1, 6.655: 1, 6.749: 1, 6.341: 1, 6.485: 1, 6.513: 1, 6.219: 1, 6.406: 1, 5.818: 1, 5.627: 1, 6.461: 1, 6.629: 1, 6.208: 1, 6.436: 1, 6.425: 1, 6.833: 1, 6.317: 1, 5.759: 1, 6.202: 1, 5.454: 1, 6.12: 1, 6.593: 1, 6.027: 1, 5.569: 1, 6.019: 1, 5.794: 1, 5.67: 1, 5.707: 1, 5.093: 1, 5.414: 1, 5.905: 1, 5.952: 1, 6.114: 1, 5.871: 1, 5.762: 1, 7.061: 1, 6.75: 1, 6.242: 1, 6.484: 1, 5.427: 1, 6.437: 1, 6.003: 1, 6.348: 1, 5.837: 1, 8.78: 1, 6.223: 1, 5.036: 1, 6.051: 1, 4.88: 1, 5.0: 1, 4.652: 1, 5.277: 1, 4.368: 1, 5.52: 1, 5.536: 1, 6.545: 1, 7.333: 1, 6.471: 1, 6.649: 1, 7.313: 1, 4.906: 1, 6.216: 1, 7.016: 1, 6.683: 1, 4.97: 1, 3.863: 1, 4.963: 1, 3.561: 1, 5.887: 1, 5.747: 1, 5.896: 1, 4.628: 1, 5.565: 1, 6.103: 1, 5.648: 1, 6.006: 1, 6.411: 1, 6.824: 1, 5.957: 1, 6.434: 1, 4.519: 1, 5.155: 1, 6.657: 1, 5.453: 1, 6.852: 1, 5.617: 1, 5.608: 1, 5.683: 1, 5.531: 1, 5.349: 1, 6.404: 1, 6.343: 1, 5.987: 1, 5.852: 1, 6.842: 1, 6.481: 1, 8.704: 1, 6.302: 1, 8.069: 1, 6.163: 1, 6.625: 1, 6.249: 1, 6.442: 1, 7.079: 1, 7.007: 1, 6.121: 1, 6.015: 1, 6.389: 1, 6.619: 1, 6.781: 1, 5.874: 1, 6.232: 1, 6.14: 1, 6.279: 1, 6.286: 1, 6.273: 1, 6.245: 1, 6.065: 1, 5.885: 1, 5.594: 1, 7.416: 1, 6.137: 1, 7.454: 1, 5.731: 1, 6.335: 1, 5.822: 1, 6.372: 1, 6.458: 1, 5.637: 1, 5.693: 1, 5.613: 1, 5.986: 1, 5.879: 1, 5.87: 1, 5.872: 1, 5.851: 1, 6.021: 1, 6.176: 1, 5.928: 1, 6.254: 1, 6.092: 1, 5.913: 1, 6.715: 1, 6.195: 1, 6.474: 1, 5.836: 1, 5.878: 1, 5.787: 1, 6.29: 1, 5.456: 1, 5.95: 1, 6.072: 1, 6.674: 1, 6.047: 1, 5.599: 1, 5.924: 1, 6.142: 1, 5.965: 1, 5.57: 1, 5.727: 1, 5.99: 1, 7.104: 1, 5.834: 1, 5.949: 1, 5.889: 1, 6.377: 1, 5.631: 1, 6.172: 1, 6.012: 1, 6.43: 1, 7.147: 1, 6.998: 1, 5.701: 1, 5.933: 1, 5.841: 1, 5.85: 1, 6.762: 1, 6.456: 1, 5.741: 1, 5.927: 1, 6.145: 1, 6.816: 1, 6.383: 1, 7.249: 1, 5.998: 1, 6.511: 1, 6.115: 1, 5.963: 1, 5.602: 1, 5.399: 1, 5.786: 1, 5.682: 1, 6.069: 1, 6.169: 1, 6.77: 1, 7.024: 1, 6.595: 1, 5.942: 1, 6.454: 1, 5.857: 1, 6.642: 1, 7.686: 1, 7.163: 1, 8.04: 1, 8.725: 1, 8.266: 1, 6.618: 1, 6.879: 1, 6.164: 1, 6.373: 1, 5.951: 1, 6.182: 1, 7.135: 1, 5.412: 1, 6.375: 1, 5.807: 1, 5.96: 1, 5.344: 1, 6.064: 1, 5.783: 1, 5.891: 1, 8.034: 1, 7.853: 1, 6.552: 1, 5.981: 1, 7.412: 1, 8.337: 1, 5.876: 1, 8.259: 1, 6.957: 1, 6.438: 1, 6.487: 1, 6.718: 1, 6.433: 1, 6.226: 1, 5.605: 1, 5.593: 1, 6.393: 1, 6.358: 1, 6.095: 1, 6.897: 1, 6.606: 1, 6.421: 1, 7.358: 1, 6.631: 1, 6.086: 1, 6.726: 1, 8.247: 1, 7.61: 1, 6.975: 1, 6.151: 1, 5.709: 1, 6.101: 1, 8.375: 1, 7.802: 1, 7.489: 1, 6.25: 1, 6.51: 1, 6.066: 1, 6.943: 1, 5.272: 1, 6.129: 1, 5.012: 1, 7.274: 1, 5.597: 1, 5.186: 1, 4.926: 1, 5.628: 1, 6.13: 1, 4.903: 1, 5.468: 1, 5.403: 1, 5.019: 1, 6.174: 1, 7.929: 1, 5.877: 1, 6.319: 1, 6.402: 1, 7.107: 1, 7.287: 1, 7.875: 1, 6.604: 1, 6.8: 1, 7.178: 1, 6.739: 1, 6.556: 1, 7.831: 1, 6.153: 1, 5.604: 1, 6.563: 1, 7.155: 1, 7.765: 1, 6.86: 1, 6.02: 1, 6.546: 1, 5.859: 1, 6.416: 1, 5.572: 1, 5.88: 1, 6.976: 1}



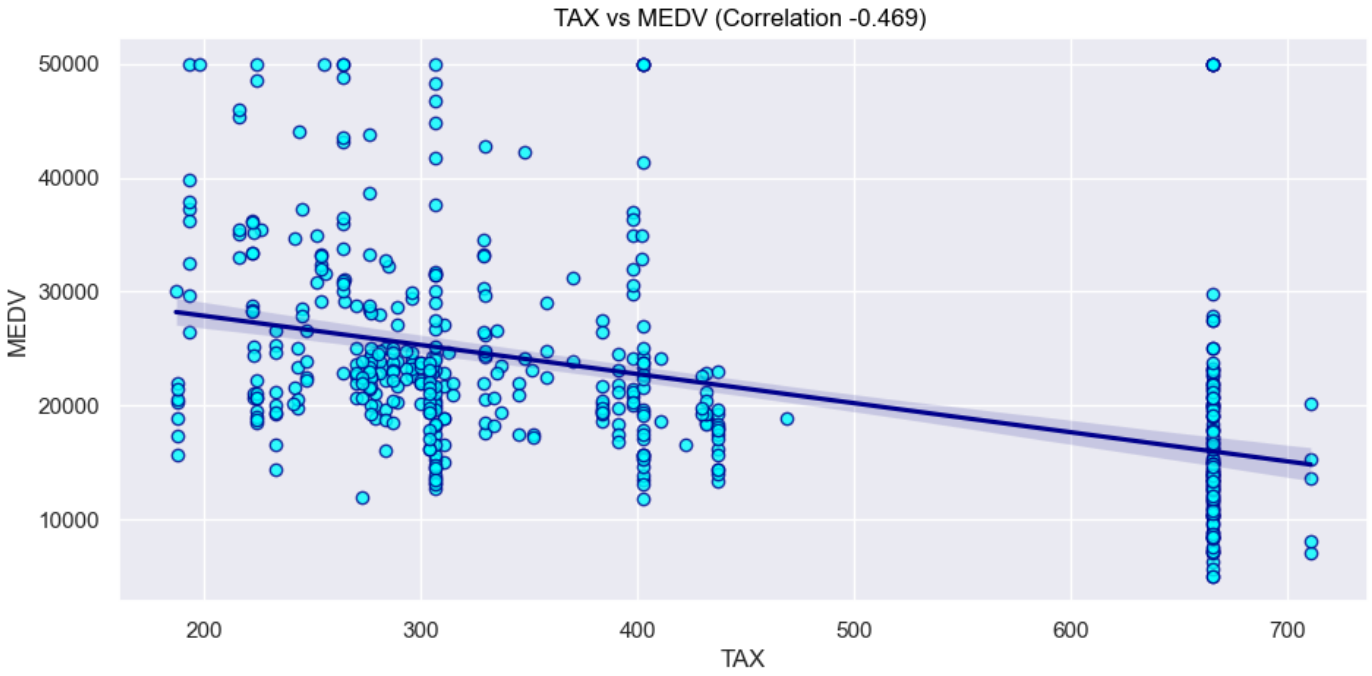
Insights

- The majority of houses in Boston typically comprise six rooms, followed by those with seven rooms. There are also a substantial number of houses with five rooms, while a smaller proportion have three, four, or eight rooms.

Bivariate Analysis (Numerical vs Numerical Features)

In [31]:
```
# Creating a function to perform Bivariate analysis on Numerical vs Numerical features

def bivariate_eda_nvn(data, col1, col2):

    # checking the correlation
    corr = np.round(data[[col1, col2]].corr().iloc[0, 1], 3)

    # plotting the relationship
    plt.figure(figsize = (11, 5))
    sns.regplot(x = data[col1], y = data[col2], scatter_kws = {"facecolors" : "cyan", "edgecolor
    plt.title(f"{col1} vs {col2} (Correlation {corr}) ", color = "black");
```

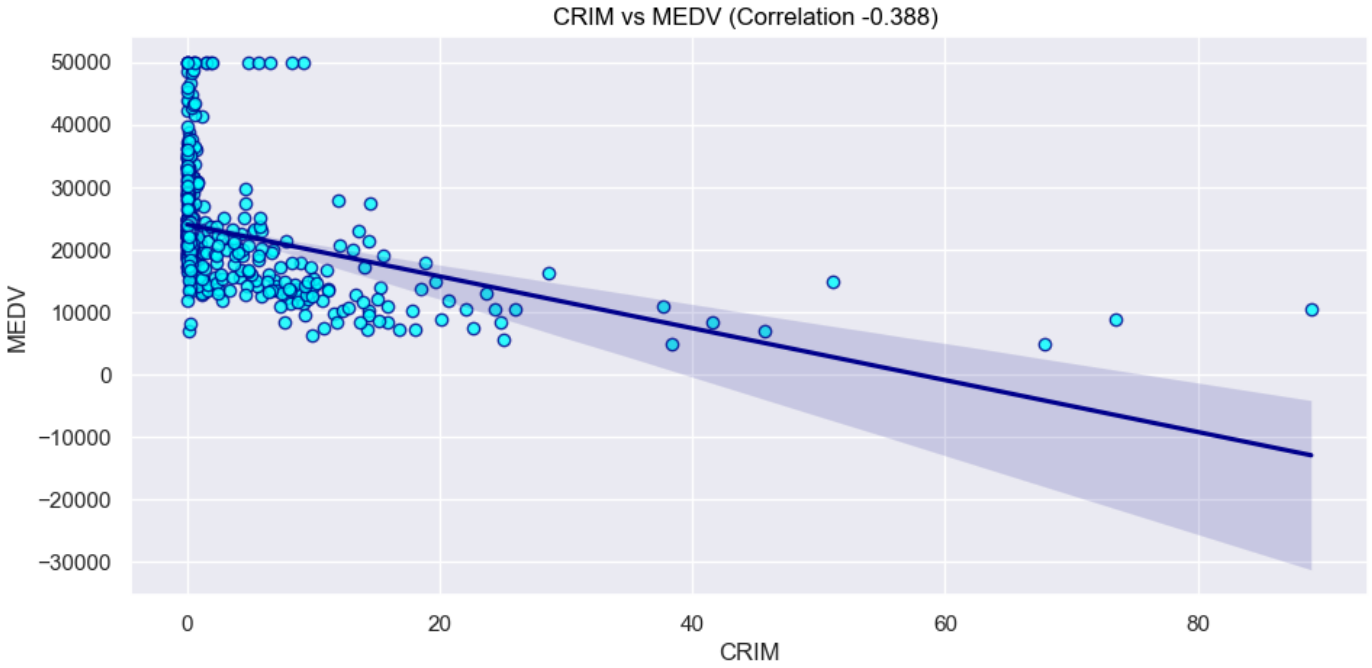
In [32]:
```
# Checking the relation between "TAX" and "MEDV"

bivariate_eda_nvn(df, "TAX", "MEDV")
```



Insights

- The correlation coefficient (r) of -0.469 indicates a moderate negative correlation (inverse relationship) between "TAX" and "MEDV". This means that as the tax rate increases, the price of houses tends to decrease.
- The finding that houses in high-tax areas are often sold at a lower price than those in low-tax areas is not surprising. This is because higher tax rates can make it more expensive to own a home, but the higher-tax areas may offer more local amenities or services that offset the negative impact on property prices.

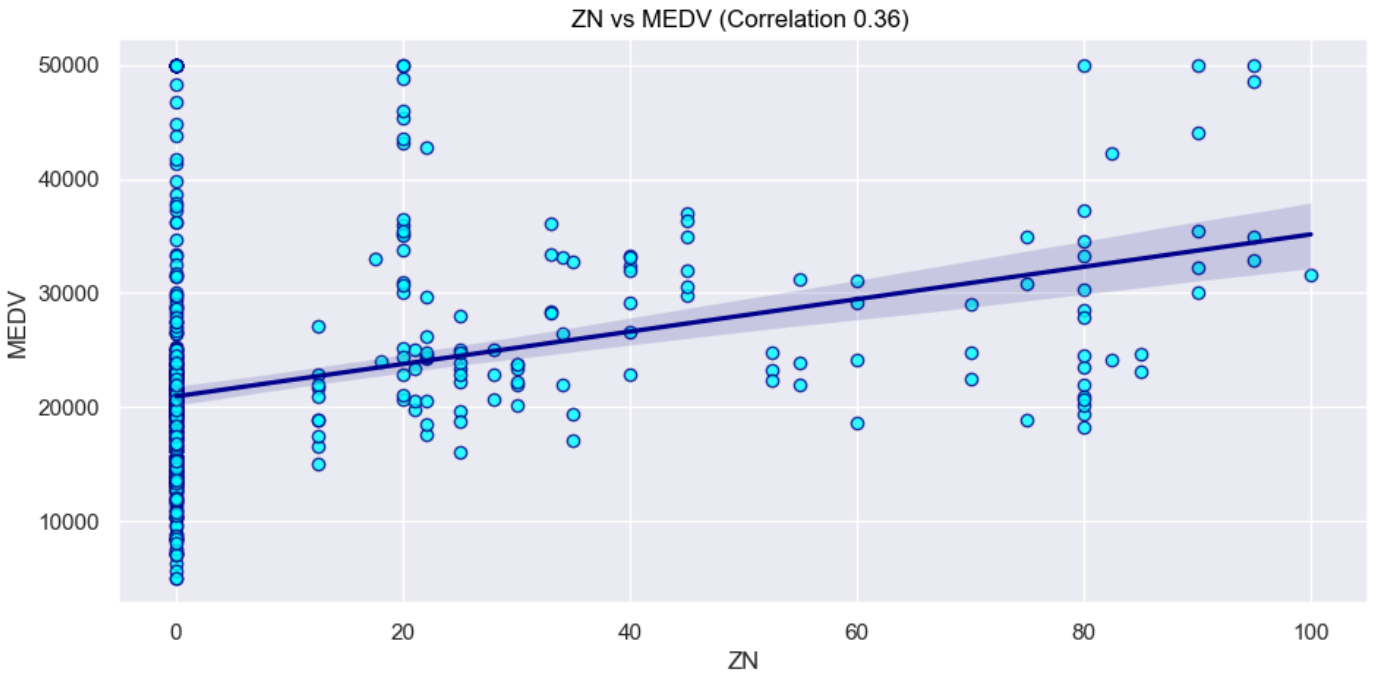
In [33]:
```
# Checking the relationship between "CRIM" and "MEDV"

bivariate_eda_nvn(df, "CRIM", "MEDV")
```



Insights

- The correlation coefficient (r) of -0.388 indicates a moderate negative correlation (inverse relationship) between "Crime_Rate" and "MEDV". This means that as the crime rate increases, the price of houses tends to decrease.
- People are less likely to want to live in areas with high crime rates, which can lead to a decrease in demand for housing in those areas. This can cause housing prices to go down.

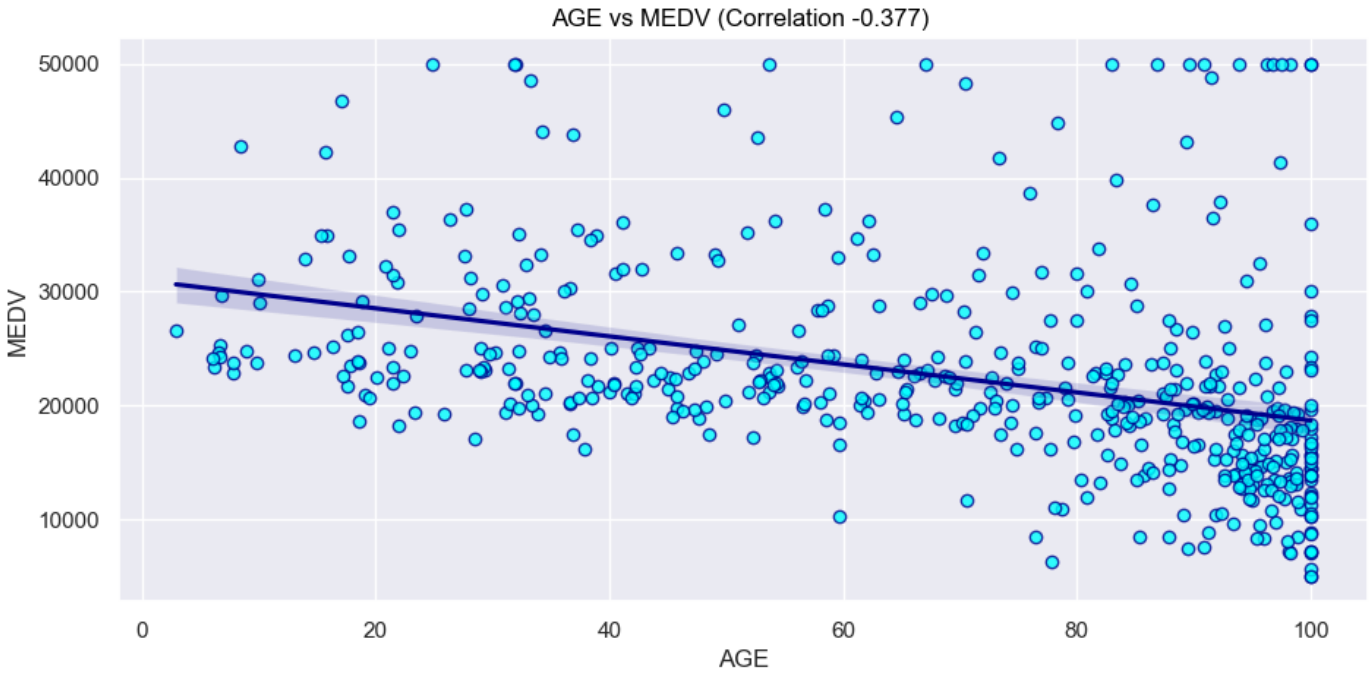
In [34]:
```
# Checking the relationship between "ZN" and "MEDV"

bivariate_eda_nvn(df, "ZN", "MEDV")
```



Insights

- The correlation coefficient (r) of 0.36 indicates a moderate positive correlation (direct relationship) between "ZN" and "MEDV". This means that as the proportion of residential land zoned for large single-family homes increases, the price of houses also tends to increase, but not by a significant amount.
- People's preference for spacious homes and well-maintained properties often leads them to buy houses in these areas at a higher price.

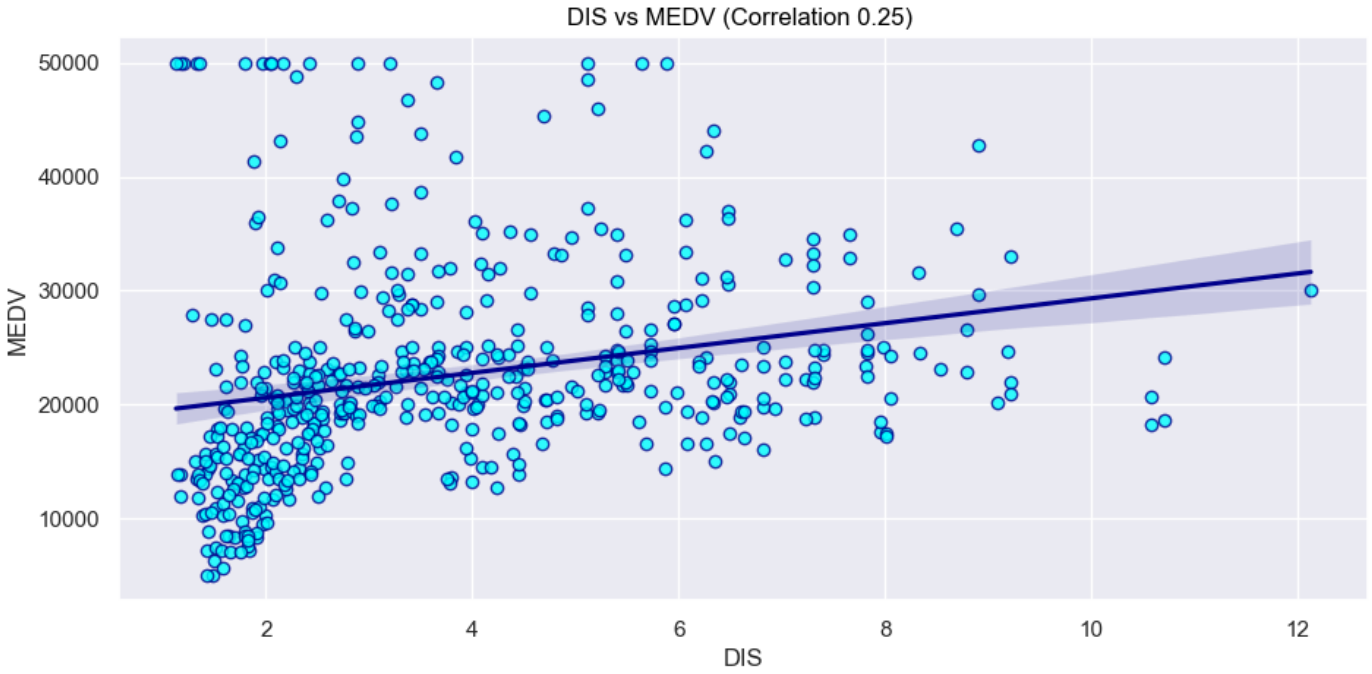
In [35]:
```
# Checking the relationship between "AGE" and "MEDV"

bivariate_eda_nvn(df, "AGE", "MEDV")
```



Insights

- The correlation coefficient (r) of -0.377 indicates a moderate negative correlation (inverse relationship) between "AGE" and "MEDV". This means that as the proportion of owner-occupied units built prior to 1940 increases, the price of the house tends to decrease.
- Houses built before 1940 may be less energy efficient and have more deferred maintenance than houses built after 1940. Additionally, houses built in older neighborhoods may be located in less desirable areas, such as those with high crime rates or poor schools, which can also lower their value.

In [36]:
```
# Checking the relationship between "DIS" and "MEDV"

bivariate_eda_nvn(df, "DIS", "MEDV")
```



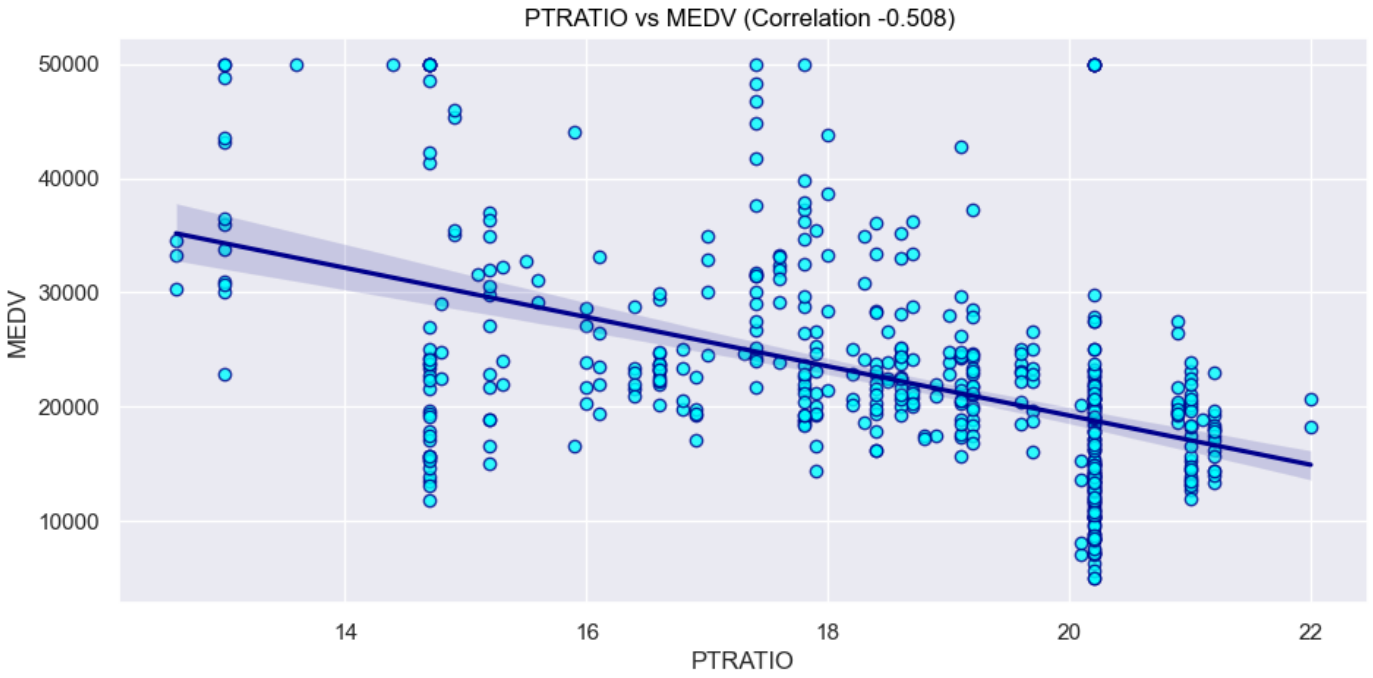
Insights

- The correlation coefficient (r) of 0.25 indicates a moderate positive correlation (direct relationship) between "DIS" and "MEDV". This means that as the distance to employment centers increases, the price

of houses tends to decrease.

- Houses that are further away from employment centers are less desirable to buyers, as they may have to commute longer distances to work.

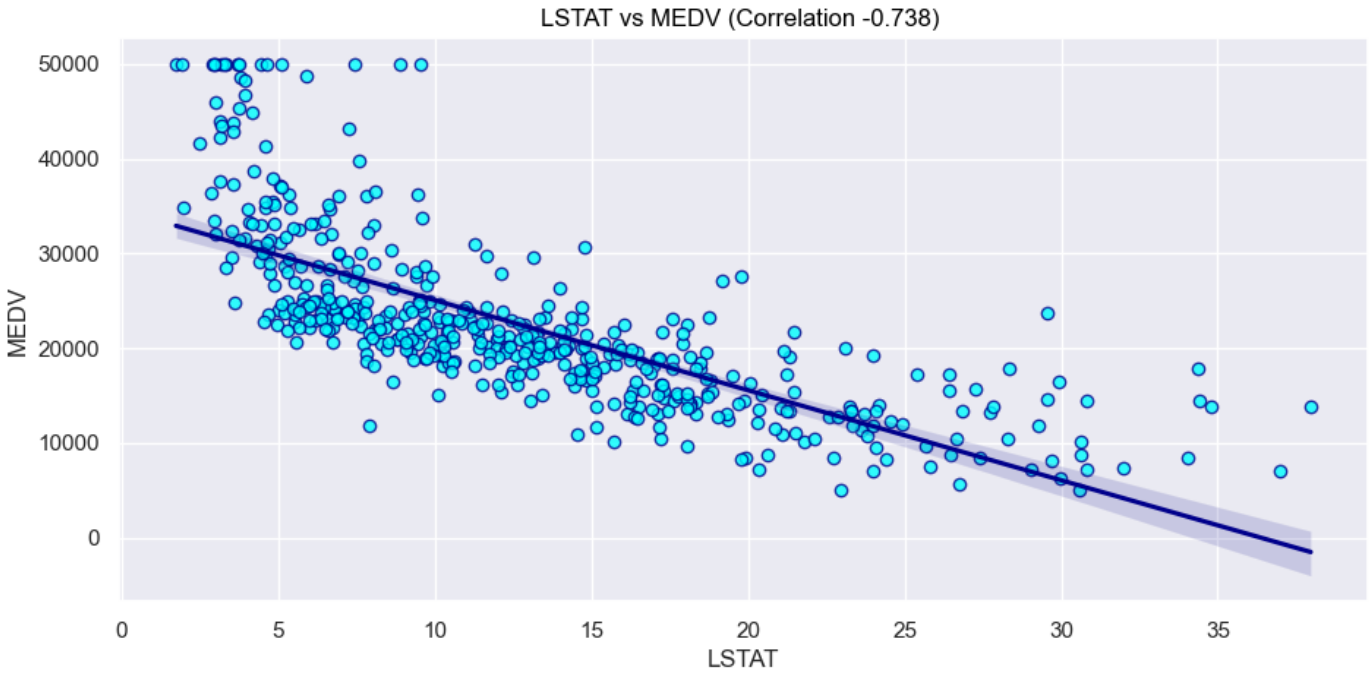
In [37]:
```
# Checking the relationship between "PTRATIO" and "MEDV"

bivariate_eda_nvn(df, "PTRATIO", "MEDV")
```



Insights

- The correlation coefficient (r) of -0.508 indicates a moderate negative correlation (inverse relationship) between "PTRATIO" and "MEDV". This means that as pupil-teacher ratio increases, the price of houses tends to decrease.
- Neighborhoods with better schools tend to have higher home prices. This is because people are willing to pay more for a home in a neighborhood with good schools, as they believe that their children will receive a better education.

In [38]:
```
# Checking the relationship between "B" and "MEDV"

bivariate_eda_nvn(df, "LSTAT", "MEDV")
```



Insights

- The correlation coefficient (r) of -0.738 indicates a moderate negative correlation (inverse relationship) between "LSTAT" and "MEDV". This means that as the value of lower-socioenomic status increases, the price of houses decreases. In other words, houses in areas with a lower socioeconomic status tend to be less expensive.
- People of lower socioeconomic status often have lower incomes, which means that they have less money to spend on housing.

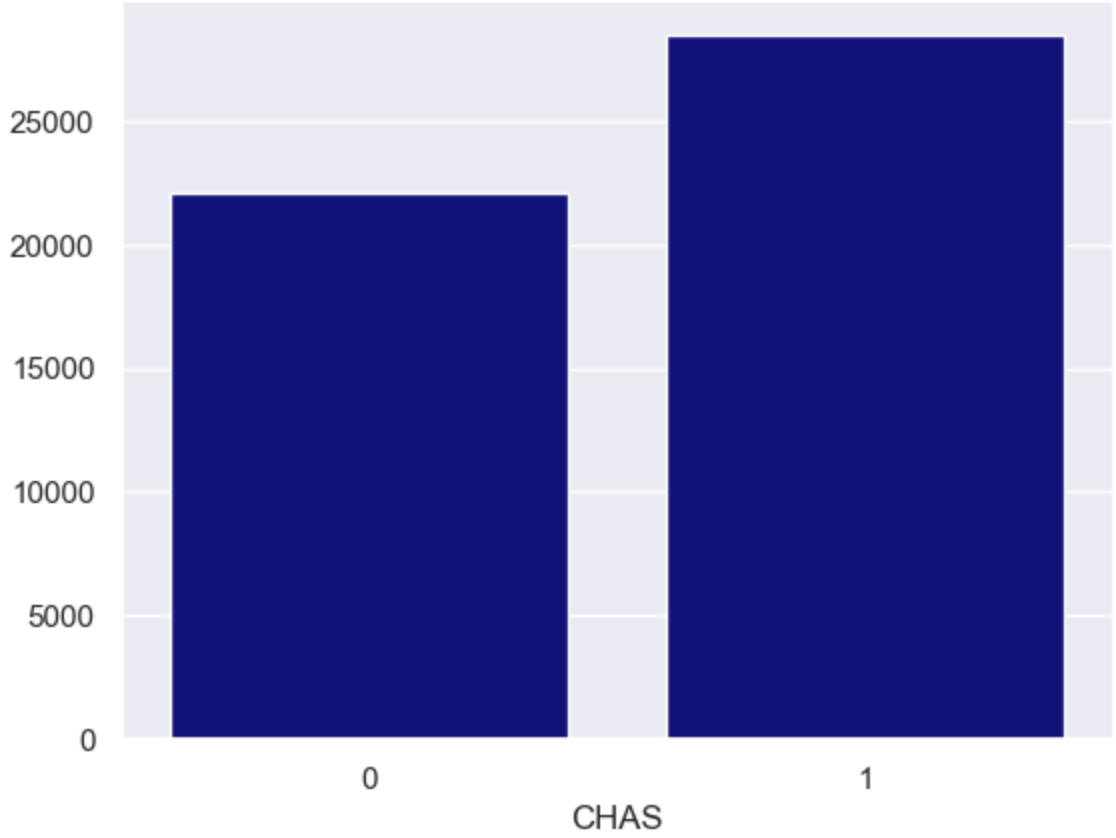
Bivariate Analysis (Numerical vs Categorical Features)

In [39]:
```
# Creating a function to perform Bivariate analysis on Numerical vs Categorical features

def bivariate_eda_nvc(data, col1, col2):

    # checking the distribution across different categories
    print('\033[1m' + f"Distribution of {col2} across different categories of the {col1} feature
    print(np.round(data.groupby(col1)[col2].mean(), 2))
    sns.barplot(x = data.groupby(col1)[col2].mean().index, y = data.groupby(col1)[col2].mean().v
```

In [40]:
```
# Checking the distribution between "CHAS" and "MEDV"

bivariate_eda_nvc(df, "CHAS", "MEDV")
```

Distribution of MEDV across different categories of the CHAS feature

```
CHAS
0    22093.84
1    28440.00
Name: MEDV, dtype: float64
```



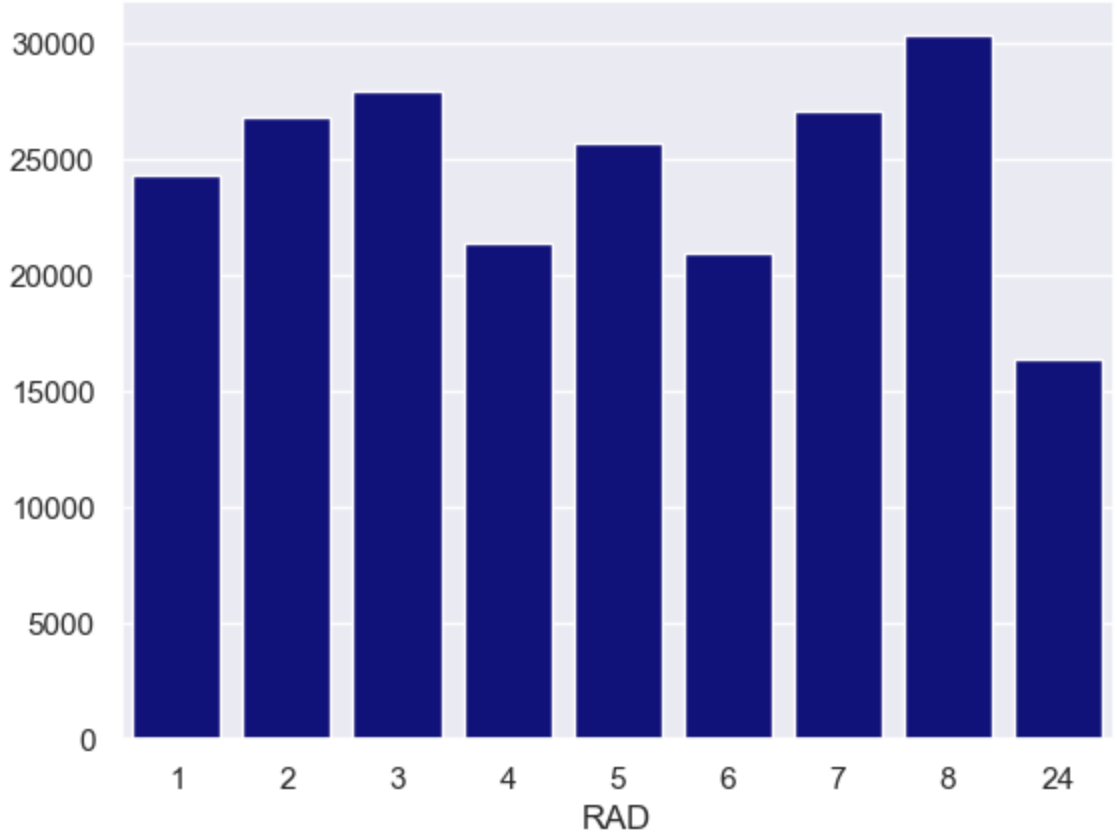
Insights

- Houses located near the Charles River have a higher price of 28,440 as they are associated having a scenic view, cooler summers and milder winters, compared to houses that are not situated near the river, which have a price of 22,093.

In [41]:
```
# Checking the distribution between "RAD" and "MEDV"

bivariate_eda_nvc(df, "RAD", "MEDV")
```

Distribution of MEDV across different categories of the RAD feature

```
RAD
1     24365.00
2     26833.33
3     27928.95
4     21387.27
5     25706.96
6     20976.92
7     27105.88
8     30358.33
24    16403.79
Name: MEDV, dtype: float64
```



Insights

- Houses possessing a RAD score of 8 exhibit a higher price by $30,358, followed closely by properties with RAD scores of 3 and 7.

Feature Transformation

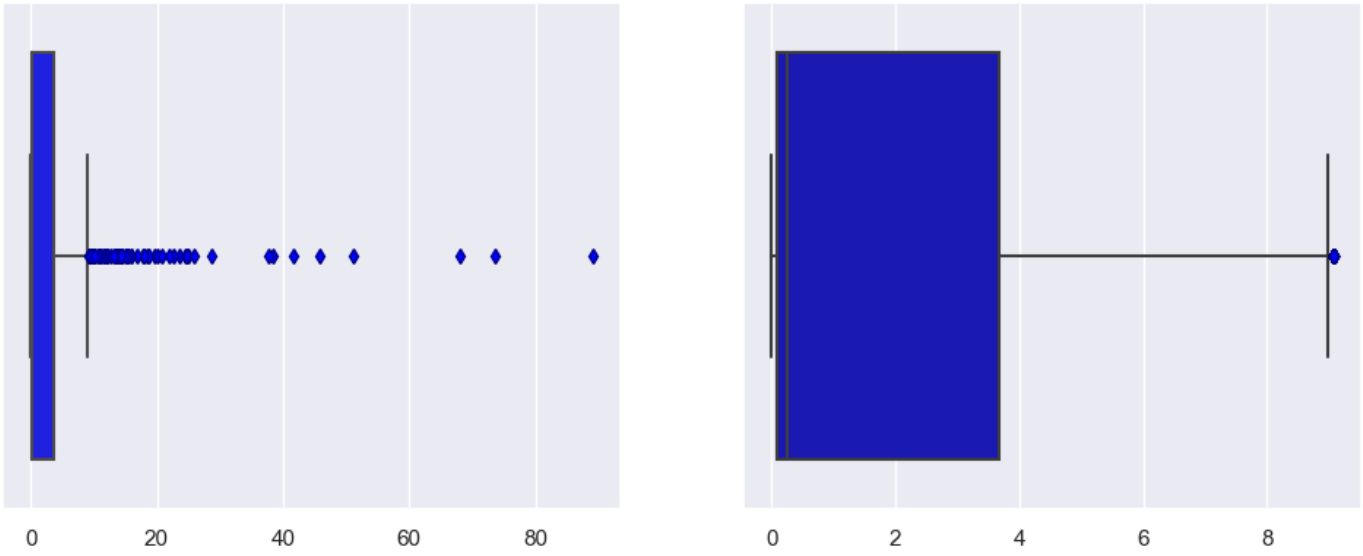
Outlier Treatment

In [42]:
```
# Creating a function to handle the Outliers

def outlier_treatment(data):

    for col in data.select_dtypes(exclude = ["O"]).columns[:-1]:
    # calculating the lower and upper bound
        Q1, Q3 = data[col].quantile([0.25, 0.75])
        IQR = Q3 - Q1
        lower_bound = Q1 - (1.5 * IQR)
        upper_bound = Q3 + (1.5 * IQR)
        outlier_count = len(data[(data[col] < lower_bound) | (data[col] > upper_bound)])

        # checking for outliers
        if outlier_count != 0:
            fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize = (12, 4.5))
            sns.boxplot(x = data[col], color = "blue", ax = axes[0], flierprops = {"markerfaceco
            data[col] = np.where(data[col] < lower_bound, np.round(lower_bound, 2), np.where(dat
            sns.boxplot(x = data[col], color = "mediumblue", ax = axes[1], flierprops = {"marker
            axes[0].set_xlabel(" ")
            axes[1].set_xlabel(" ")
            plt.show()
        else:
            pass
```

```
outlier_treatment(df)
```

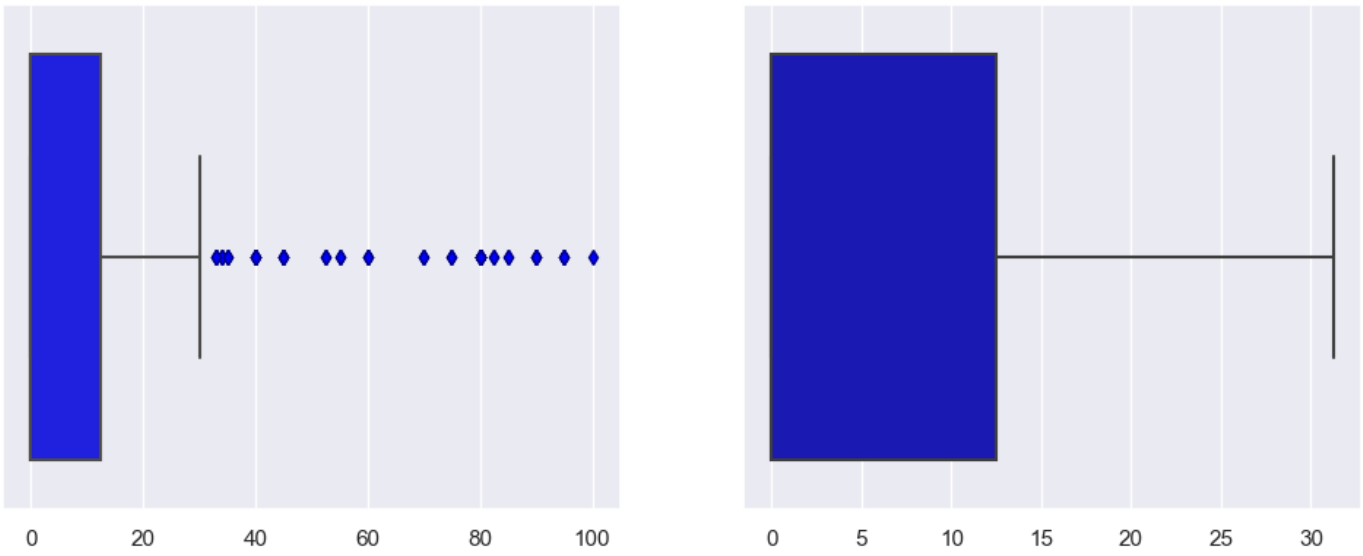
CRIM

Before & After Outlier Treatment



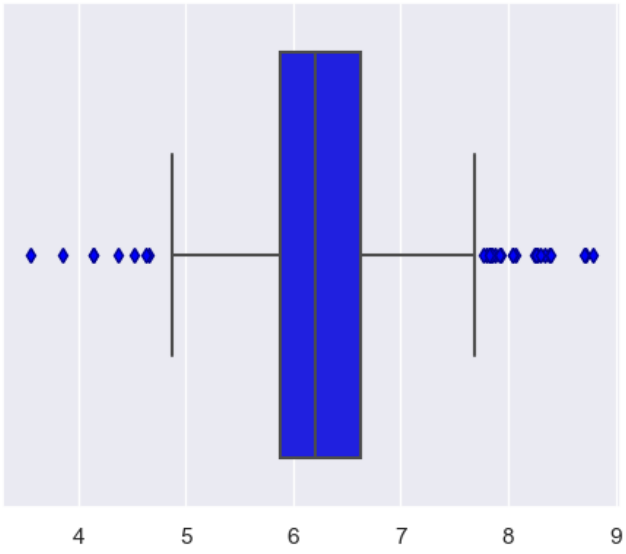
ZN

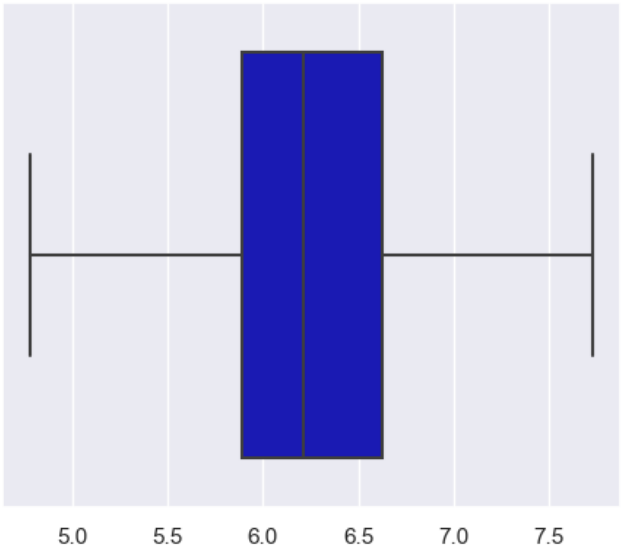
Before & After Outlier Treatment



RM

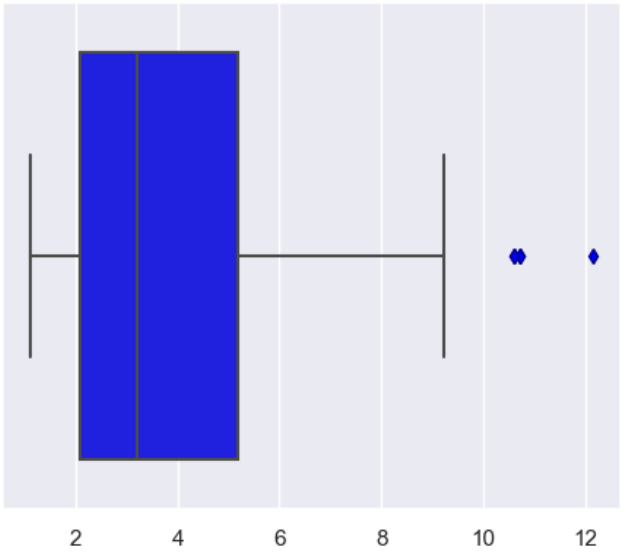
Before & After Outlier Treatment

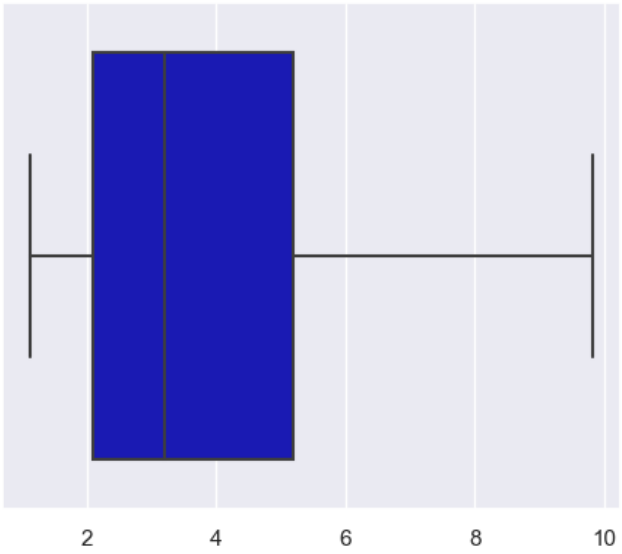




DIS

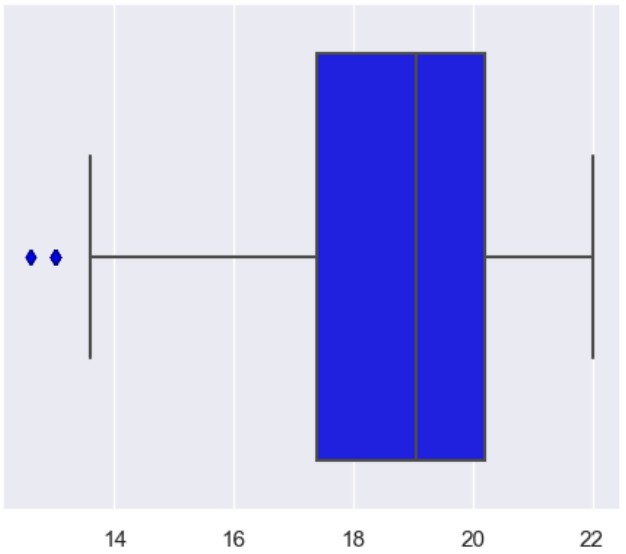
Before & After Outlier Treatment

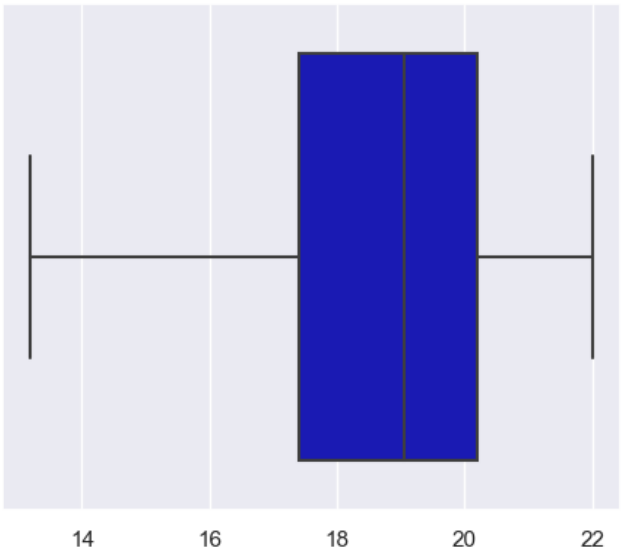


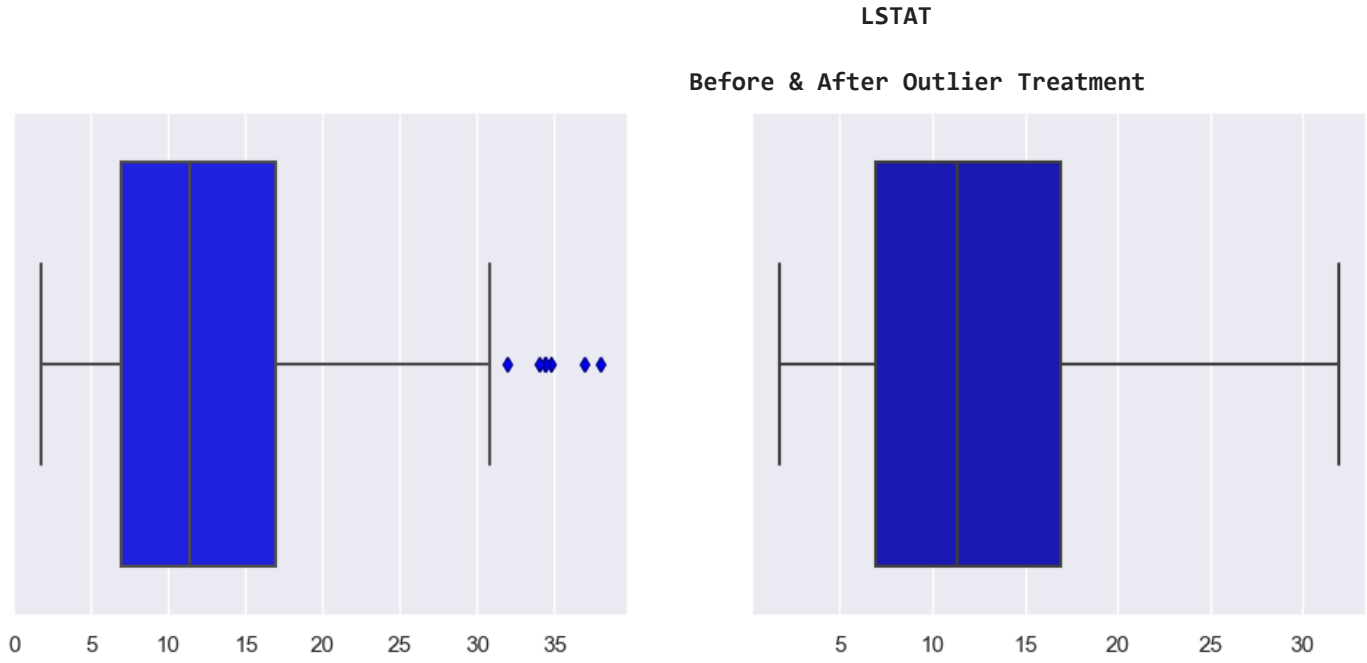


PTRATIO

Before & After Outlier Treatment







Successfully, capped the Outliers.

Multicollinearity Check - I

In [1]:
```
# Creating a function to check multicollinearity

def multicollinearity_check(data, target):

    # splitting the data into dependent and independent features
    X = data.drop(columns = target)

    # applying Standardization to scale the data
    X_sc = StandardScaler().fit_transform(X)

    vif_features = {}
    drop_flag = True

    while drop_flag:

        vif_df = pd.DataFrame()
        vif_df["VIF"] = [np.round(variance_inflation_factor(X_sc, i), 3) for i in range(X.shape[
        vif_df["Feature"] = X.columns
        max_vif_feature = vif_df.loc[vif_df['VIF'].idxmax()]

        if max_vif_feature['VIF'] > 5:
            X_sc = np.delete(X_sc, max_vif_feature.name, axis=1)
            X.drop(columns = max_vif_feature['Feature'], inplace = True)
            vif_features[max_vif_feature["Feature"]] = max_vif_feature["VIF"]
        else:
            drop_flag = False

    if len(vif_features) == 0:
        print('\033[1m' + "There is no Multicollinearity." + '\033[0m')
    else:
        print('\033[1m' + f'Features (VIF > 5)\n' + '\033[0m')
        for feature, vif in vif_features.items():
            print(f"{feature} : {vif}")

multicollinearity_check(df, "MEDV")
```

```
# Dropping the features having a high VIF

df.drop(columns = ["RAD", "TAX"], inplace = True)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 35
     32         for feature, vif in vif_features.items():
     33             print(f"{feature} : {vif}")
---> 35 multicollinearity_check(df, "MEDV")
     37 # Dropping the features having a high VIF
     39 df.drop(columns = ["RAD", "TAX"], inplace = True)

NameError: name 'df' is not defined
```

Forward Selection

In [44]:
```
# Creating a function to check statistical significance of the features

def feature_significance(data, target):

        p_values = pd.Series(np.round(f_regression(data.drop(columns = [target]), data[target])[
    non_significant_features = p_values[p_values > 0.05].to_dict()

    if len(non_significant_features) != 0:
        print('\033[1m' + f'Non-significant Features (p-value > 0.05)\n' + '\033[0m')
        for col, pval in non_significant_features.items():
            if pval > 0.05:
                print(f"{col} : {pval}")
            else:
                pass
    else:
        print('\033[1m' + "All features are statistically significant." + '\033[0m')

feature_significance(df, "MEDV")
```

```
All features are statistically significant.
```

In [45]:
```
    # Creating a function to perform Encoding

    # creating a copy of the data

    df_model = df.copy()

    def encoder(data):

        # applying one-hot encoding to the object features
        for col in data.select_dtypes("O").columns:
            data[col] = data[col].astype(str)
            en_df = pd.DataFrame(OneHotEncoder(sparse = False, drop = [data[col].value_counts().
            data = pd.concat([data, en_df], axis = 1)
            data.drop(columns = [col], inplace = True)

        return data

    df_model = encoder(df_model)

    df_model
```

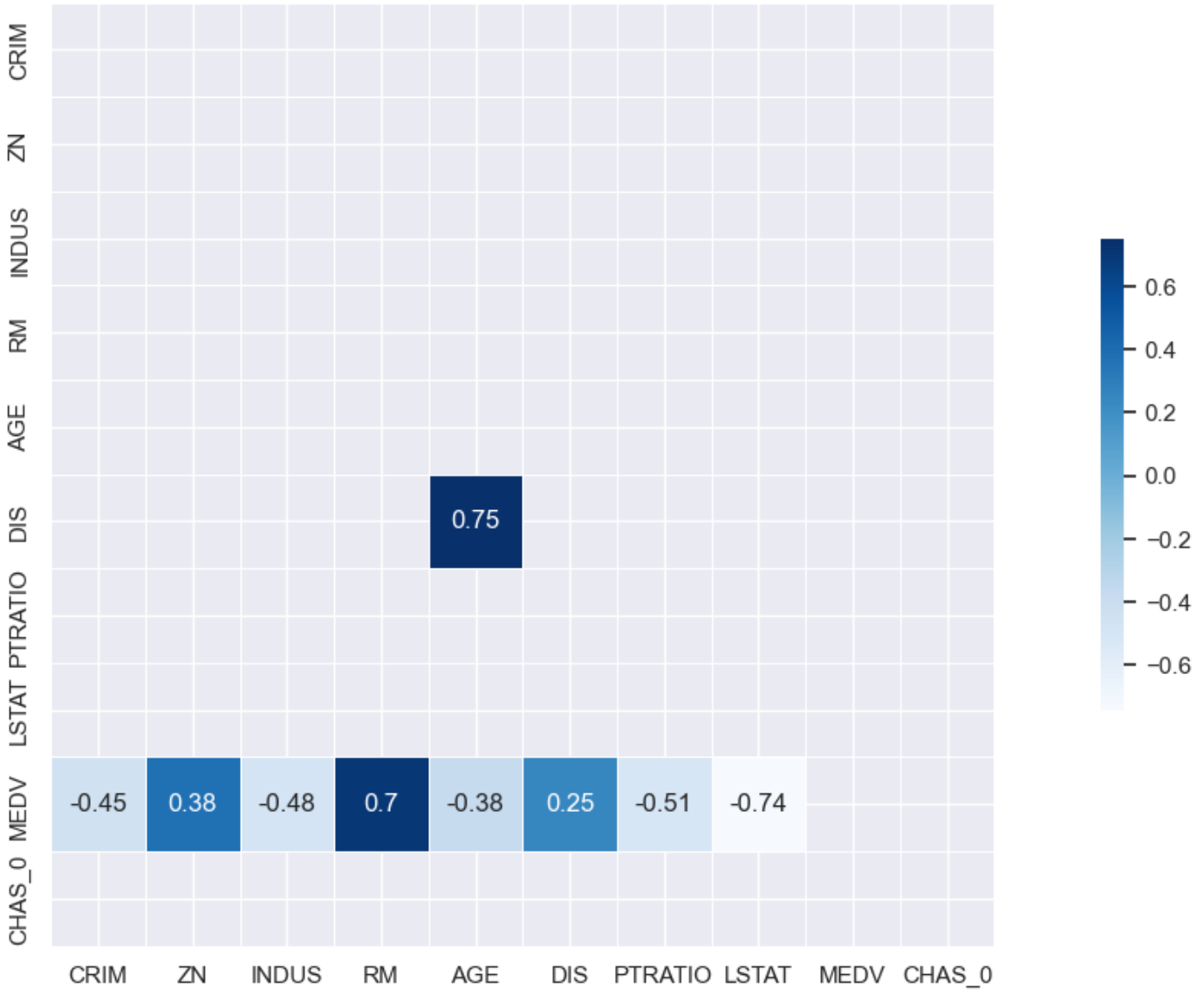
Out[45]:

	CRIM	ZN	INDUS	RM	AGE	DIS	PTRATIO	LSTAT	MEDV	CHAS_0
0	0.00632	18.0	2.31	6.575	65.2	4.0900	15.3	4.98	24000.0	1
1	0.02731	0.0	7.07	6.421	78.9	4.9671	17.8	9.14	21600.0	1
2	0.02729	0.0	7.07	7.185	61.1	4.9671	17.8	4.03	34700.0	1
3	0.03237	0.0	2.18	6.998	45.8	6.0622	18.7	2.94	33400.0	1
4	0.06905	0.0	2.18	7.147	54.2	6.0622	18.7	5.33	36200.0	1
...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	6.593	69.1	2.4786	21.0	9.67	22400.0	1
502	0.04527	0.0	11.93	6.120	76.7	2.2875	21.0	9.08	20600.0	1
503	0.06076	0.0	11.93	6.976	91.0	2.1675	21.0	5.64	23900.0	1
504	0.10959	0.0	11.93	6.794	89.3	2.3889	21.0	6.48	22000.0	1
505	0.04741	0.0	11.93	6.030	80.8	2.5050	21.0	7.88	11900.0	1

506 rows × 10 columns

Collinearity

In [46]:
```
# Checking the correlation

correlation(df_model)
```



Multicollinearity Check - II

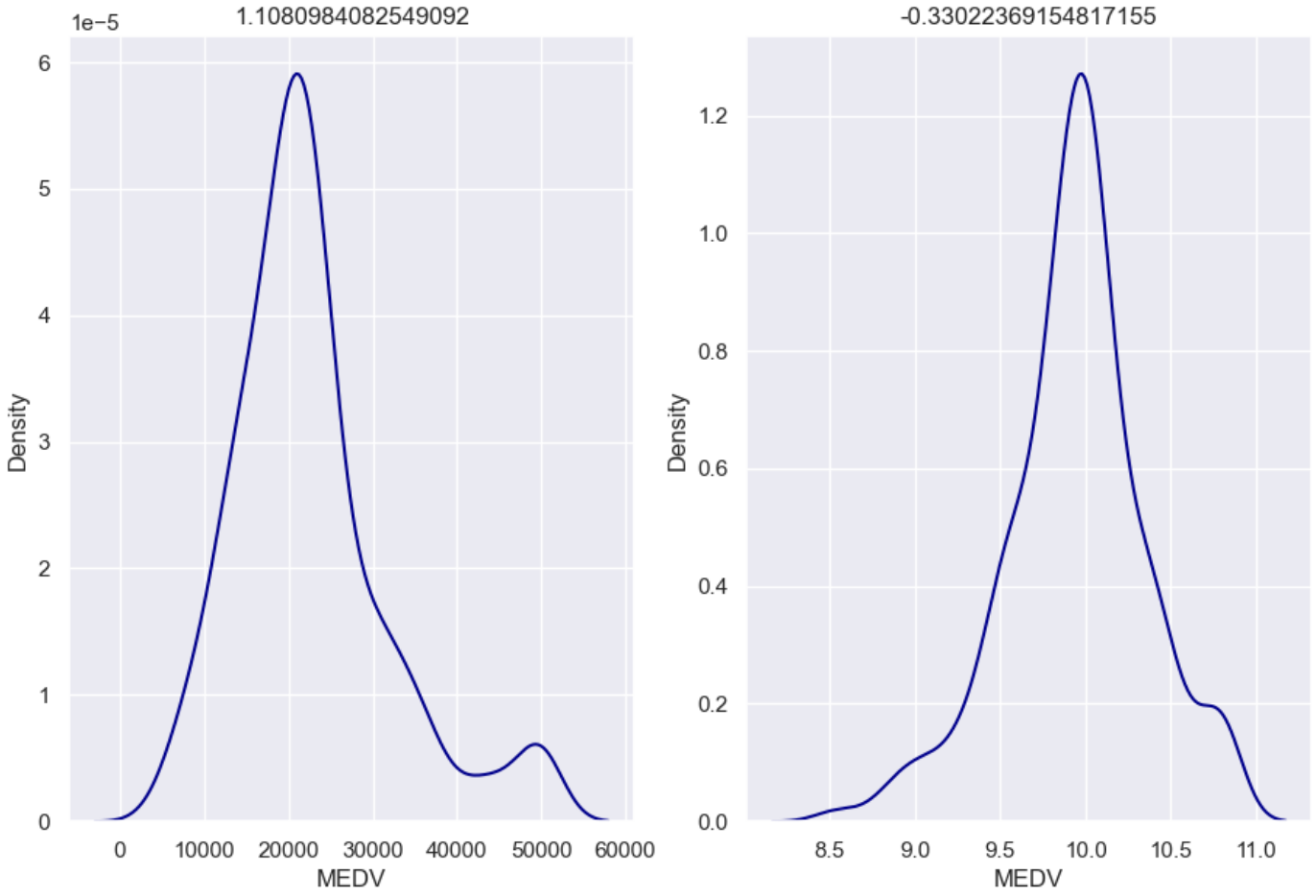
In [47]:
```
# Checking for multicollinearity

multicollinearity_check(df_model, "MEDV")
```

```
There is no Multicollinearity.
```

Log Transformation

In [48]:
```
# Checking the distribution of the "price" feature

plt.figure(figsize = (11, 7))

plt.subplot(1, 2, 1)
sns.kdeplot(df_model["MEDV"], color = "darkblue")
plt.title(df_model["MEDV"].skew())

print('\033[1m' + 'Since the "MEDV" feature is not normally distributed, there won\'t be a linea

y_log = np.log1p(df_model["MEDV"])
plt.subplot(1, 2, 2)
sns.kdeplot(y_log, color = "darkblue")
plt.title(y_log.skew())
plt.show()
```

Since the "MEDV" feature is not normally distributed, there won't be a linear relationship betwe
en the dependent variables and "price", which means the Linearity assumption won't be satisfied.
However, applying a log transformation to the "MEDV" feature will make it normally distributed a
nd later take exponential to get the original predictions.



Feature Scaling

In [49]:
```
# Splitting the data into dependent and independent features

X = df_model.drop(columns = "MEDV")
y = df_model["MEDV"]

# Applying standardization to scale the data

X_sc = StandardScaler().fit_transform(X)
```

Model Building & Evaluation

In [50]:
```
def regression_model(MODEL, train, test):

    print('\033[1m' + f"\t\t\t\t\tModel : {MODEL}" + '\033[0m\n')

    model = MODEL

    # splitting the data for training and test
    X_train, X_test, y_train, y_test = train_test_split(train, test, test_size = 0.2, random_sta

    # creating a list to store measurements
    metrics = []
```

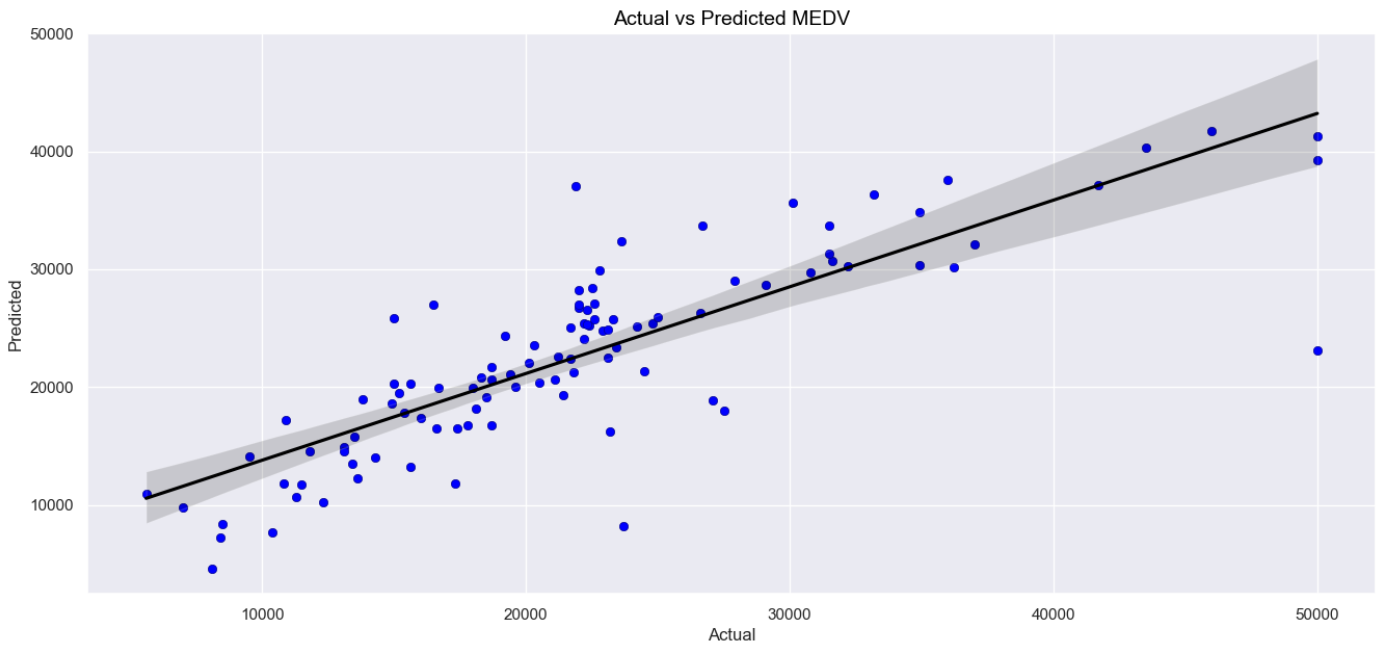
```python
# training the model
model.fit(X_train, y_train)

# making predictions on the training and testing data
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# calculating the r2_score and adjusted_r2_score
train_r2_score = np.round(r2_score(y_train, y_train_pred), 3)
test_r2_score = np.round(r2_score(y_test, y_test_pred), 3)
adj_r2_train = np.round(1 - (1 - r2_score(y_train, y_train_pred)) * (len(y_train) - 1) / (le
adj_r2_test = np.round(1 - (1 - r2_score(y_test, y_test_pred)) * (len(y_test) - 1) / (len(y_
metrics.extend([train_r2_score, test_r2_score, adj_r2_train, adj_r2_test])

# calculating the variance to check whether the model is overfitting or underfitting
variance = str(np.round((train_r2_score * 100) - (test_r2_score * 100), 3)) + "%"
metrics.append(variance)

print('\033[1m' + "r2 Score" + '\033[0m\n')
print(f"r2_score_train : {train_r2_score}")
print(f"r2_score_test : {test_r2_score}")
print(f"Variance : {variance}")
print(f"Adjusted_r2_score_train : {adj_r2_train}")
print(f"Adjusted_r2_score_test : {adj_r2_test}\n")

# calculating the metrics
mse = np.round(mean_squared_error(y_test, y_test_pred), 2)
rmse = np.round(np.sqrt(mse), 2)
mae = np.round(mean_absolute_error(y_test, y_test_pred), 2)
metrics.extend([mse, rmse, mae])
print('\033[1m' + f"Model Measurements (Metrics)" + '\033[0m\n')
print(f"MSE : {mse}\nRMSE : {rmse}\nMAE : {mae}")
print()

# plotting the actual vs predicted results
plt.figure(figsize = (16, 7))
sns.scatterplot(x = y_test, y = y_test_pred, color = "blue", edgecolor = "darkblue")
sns.regplot(x = y_test, y = y_test_pred, scatter = False, color = "black", line_kws = {"colo
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title(f"Actual vs Predicted MEDV", color = "black", size = 14)
plt.show()
print()

# checking the normality of the residuals
plt.figure(figsize = (16, 7))
residuals = y_test - y_test_pred
plt.subplot(1, 2, 1)
sns.kdeplot(residuals, color = "darkblue")
plt.subplot(1, 2, 2)
stats.probplot(residuals, dist = "norm", plot = plt)
plt.suptitle("Residuals Distribution", color = "black", size = 14)
plt.show()
print()

# checking for homoscedasticity
plt.figure(figsize = (16, 7))
sns.scatterplot(x = y_test_pred, y = residuals, color = "blue")
plt.axhline(y = 0, color = "red", linewidth = 2)
plt.xlabel("Predictions")
plt.ylabel("Residuals")
plt.title("Predicted vs Residuals", color = "black", size = 14)
print()
```

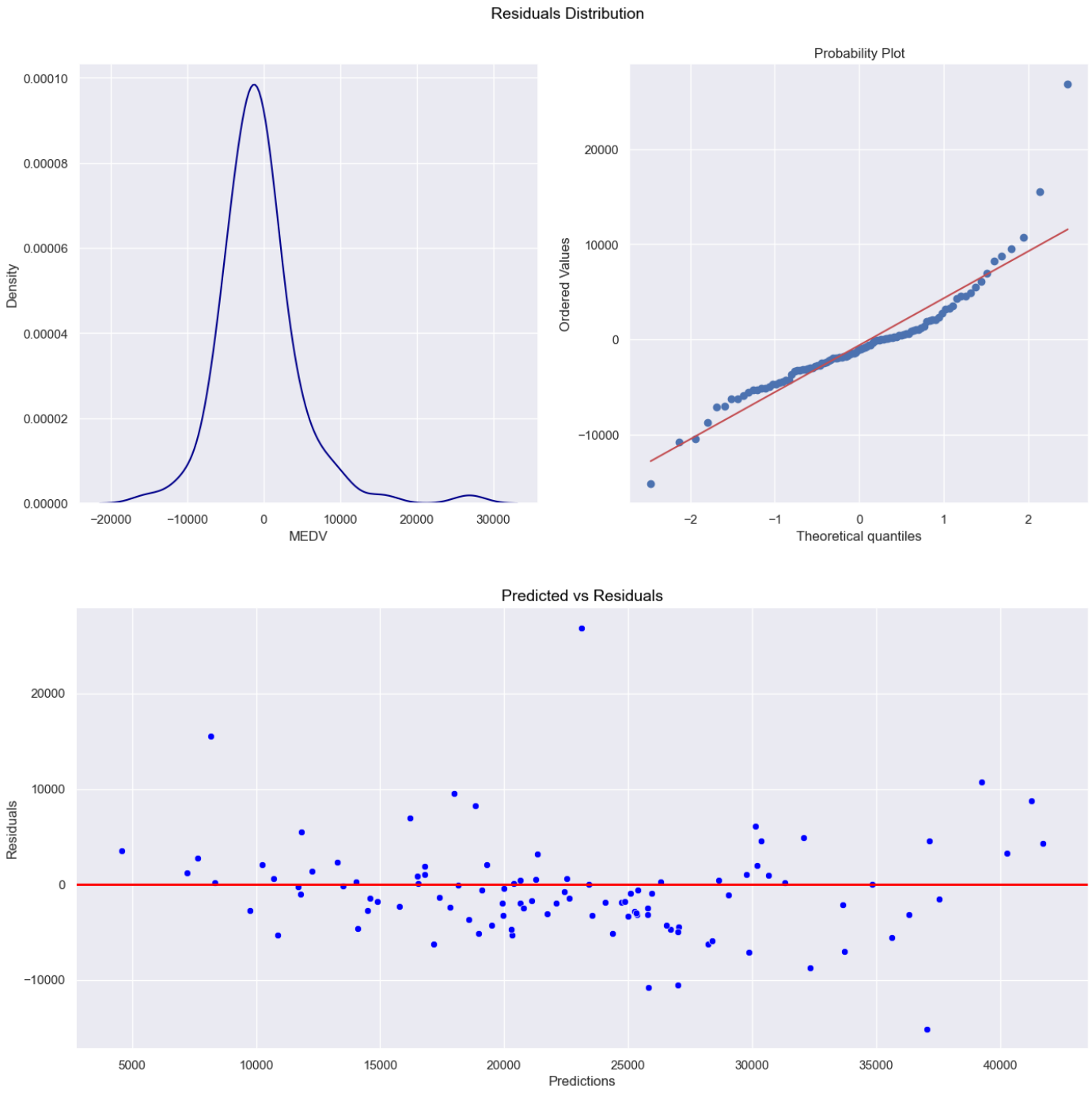
```
    return model, metrics
```

Linear Regression

In [51]:
```
lr_model, lr_metrics = regression_model(LinearRegression(), X_sc, y)
```

```
                                        Model : LinearRegression()

r2 Score

r2_score_train : 0.717
r2_score_test : 0.68
Variance : 3.7%
Adjusted_r2_score_train : 0.711
Adjusted_r2_score_test : 0.648

Model Measurements (Metrics)

MSE : 27486727.36
RMSE : 5242.78
MAE : 3554.02
```





Linear Regression (Log Transformation)

In [52]:
```
lr_log_model, lr_log_metrics = regression_model(LinearRegression(), X_sc, y_log)
```

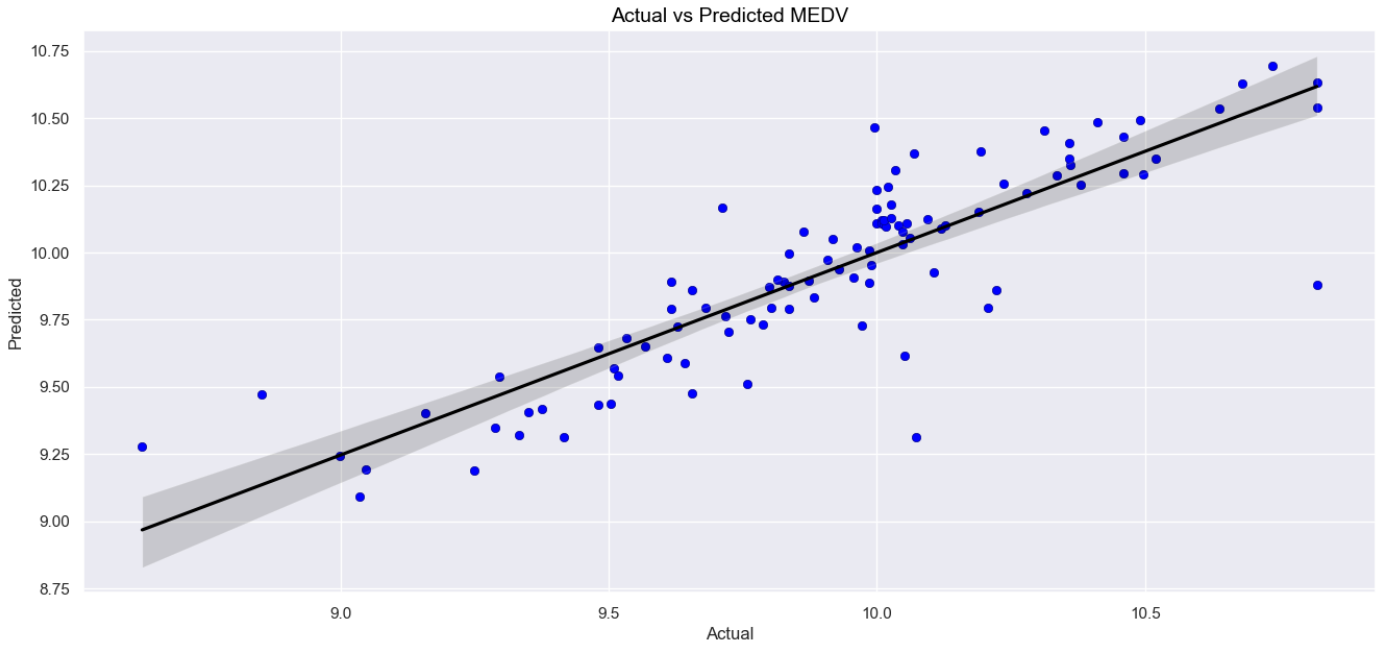
Model : LinearRegression()

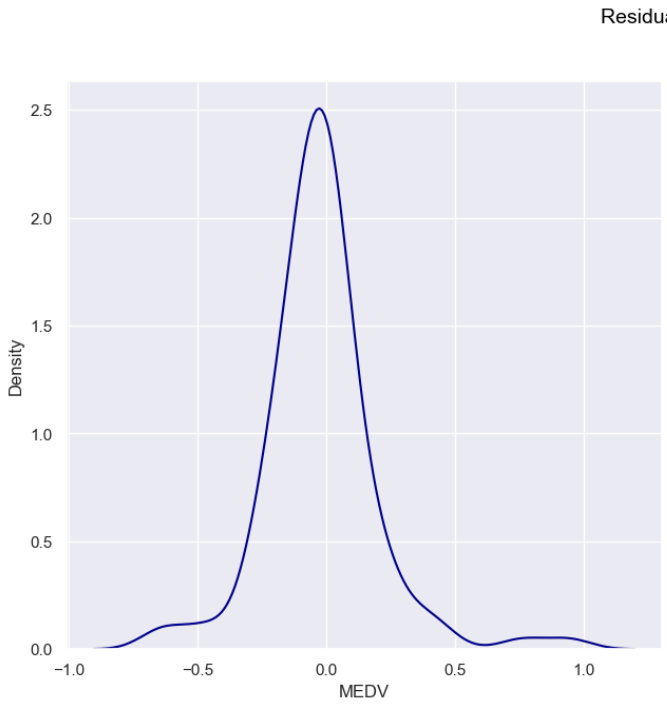
r2 Score

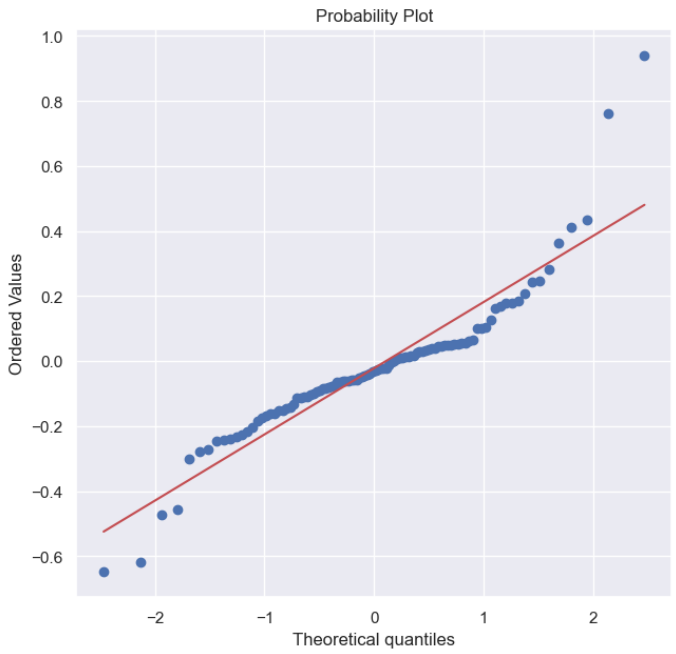
r2_score_train : 0.753
r2_score_test : 0.745
Variance : 0.8%
Adjusted_r2_score_train : 0.748
Adjusted_r2_score_test : 0.72

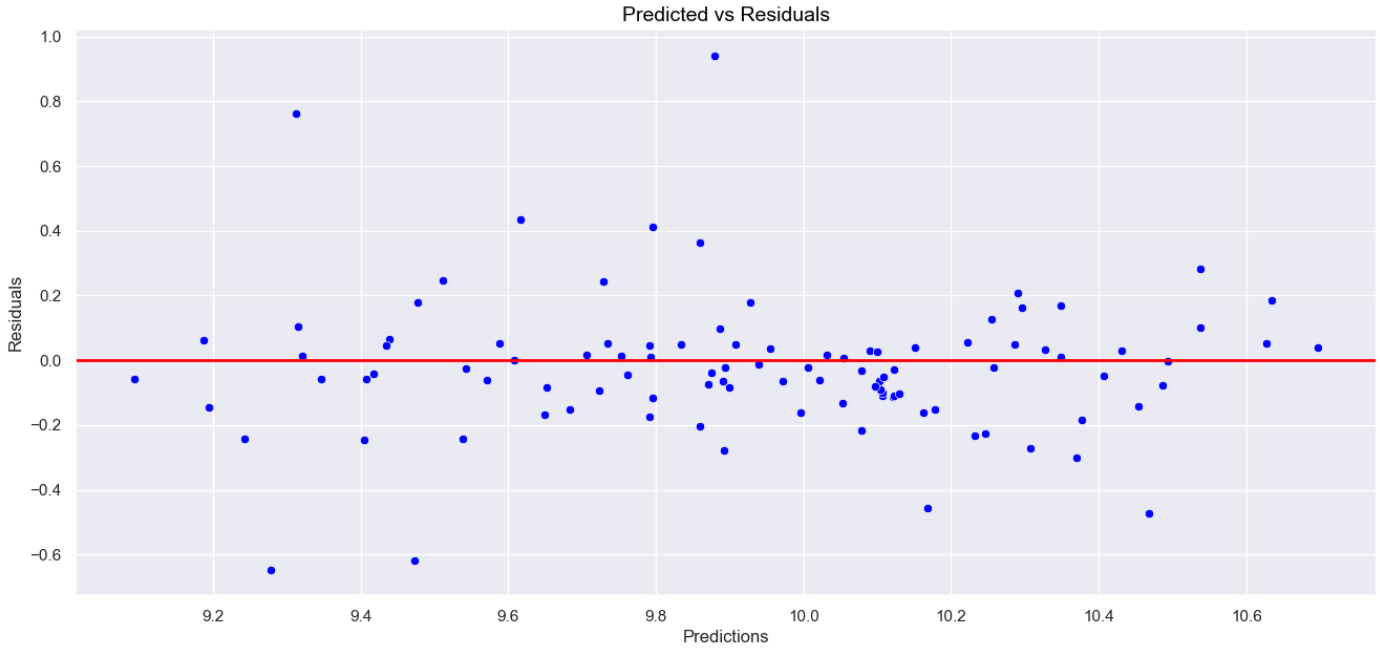
Model Measurements (Metrics)

MSE : 0.05
RMSE : 0.22
MAE : 0.14









OLS Method

In [53]:
```
lr = LinearRegression()
X_train, X_test, y_train, y_test = train_test_split(X, y_log, test_size = 0.2, random_state = 11
lr.fit(X_train, y_train)

reg_model = smf.OLS(endog = y_train, exog = X_train).fit()
reg_model.summary()
```

Out[53]:

OLS Regression Results

Dep. Variable:	MEDV	R-squared (uncentered):	0.997
Model:	OLS	Adj. R-squared (uncentered):	0.997
Method:	Least Squares	F-statistic:	1.515e+04
Date:	Wed, 30 Aug 2023	Prob (F-statistic):	0.00
Time:	20:00:44	Log-Likelihood:	-320.71
No. Observations:	404	AIC:	659.4
Df Residuals:	395	BIC:	695.4
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
CRIM	-0.0961	0.012	-8.005	0.000	-0.120	-0.072
ZN	0.0051	0.003	1.492	0.137	-0.002	0.012
INDUS	0.0285	0.006	4.505	0.000	0.016	0.041
RM	1.0383	0.035	29.467	0.000	0.969	1.108
AGE	0.0027	0.002	1.597	0.111	-0.001	0.006
DIS	0.0752	0.024	3.091	0.002	0.027	0.123
PTRATIO	0.1318	0.013	9.998	0.000	0.106	0.158
LSTAT	0.0246	0.006	3.887	0.000	0.012	0.037
CHAS_0	0.0689	0.109	0.633	0.527	-0.145	0.283

Omnibus:	27.497	Durbin-Watson:	2.007
Prob(Omnibus):	0.000	Jarque-Bera (JB):	47.861
Skew:	0.438	Prob(JB):	4.05e-11
Kurtosis:	4.440	Cond. No.	315.

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Linear / OLS Assumptions

A1 - Linearity

- **Actual and Predicted Prices are linearly related.**

A2 - No Multicollinearity

- **All the independent features have VIF < 5, which indicates the absence of multicollinearity among them.**

A3 - Normality

- **Residuals are normally distributed.**

A4 - Homoscedasticity

- **There is equal variance of residuals.**

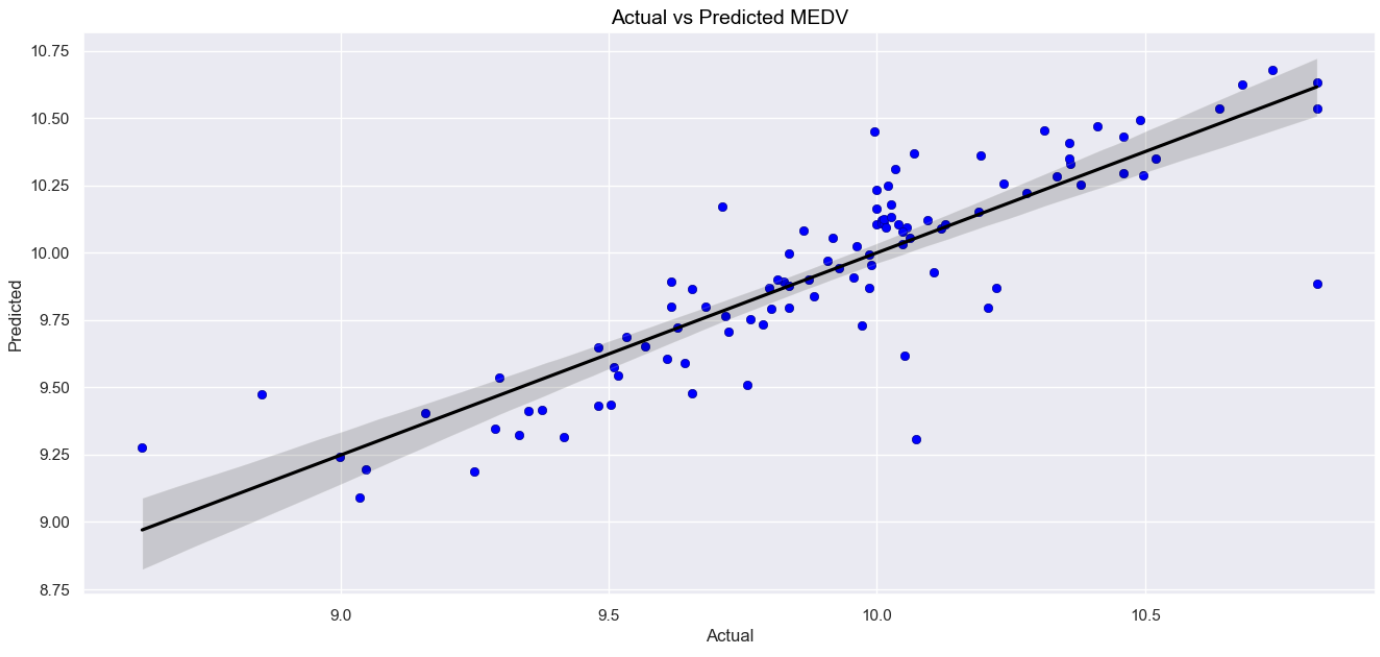
A5 - No Autocorrelation

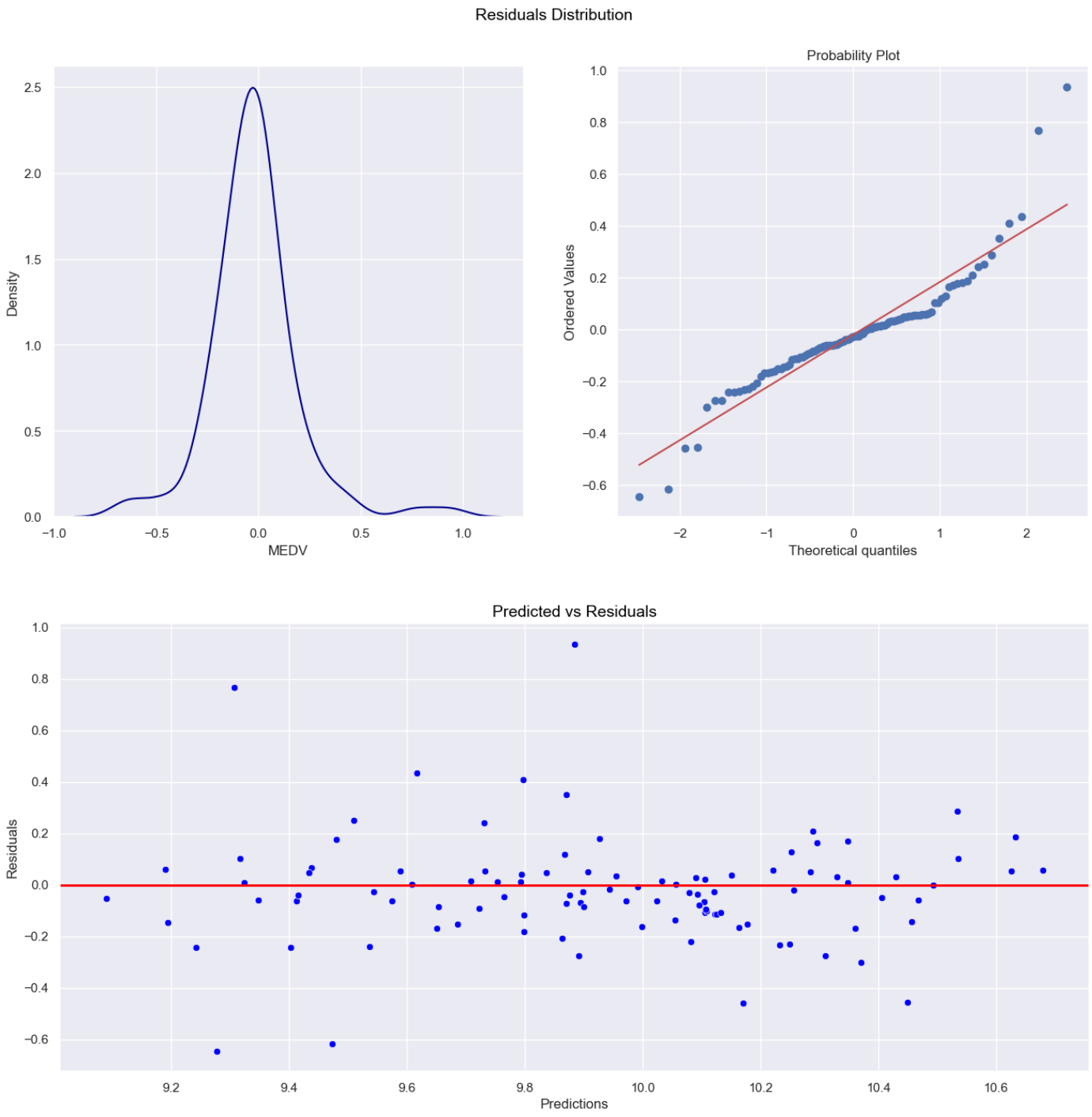
- **The Durbin-Watson statistic value is 2.007, which is nearly equal to 2, suggesting the absence of autocorrelation.**

Lasso Regression (L1 Regularization)

In [54]:
```
# param_grid = {"alpha" : [0.001, 0.01, 0.1, 1, 5, 10, 50, 100]}

# grid_model = GridSearchCV(estimator = Lasso(), param_grid = param_grid, scoring = "neg_mean_sq

# print(grid_model.best_params_)

lasso_model, lasso_metrics = regression_model(Lasso(alpha = 0.001), X, y_log)
```

```
                                    Model : Lasso(alpha=0.001)

r2 Score

r2_score_train : 0.753
r2_score_test : 0.745
Variance : 0.8%
Adjusted_r2_score_train : 0.748
Adjusted_r2_score_test : 0.721

Model Measurements (Metrics)

MSE : 0.05
RMSE : 0.22
MAE : 0.14
```





Ridge Regression (L2 Regularization)

In [55]:
```
# param_grid = {"alpha" : [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 10]}

# grid_model = GridSearchCV(Ridge(), param_grid, scoring = "neg_mean_squared_error").fit(X_train

# print(grid_model.best_params_)

ridge_model, ridge_metrics = regression_model(Ridge(1), X, y_log)
```

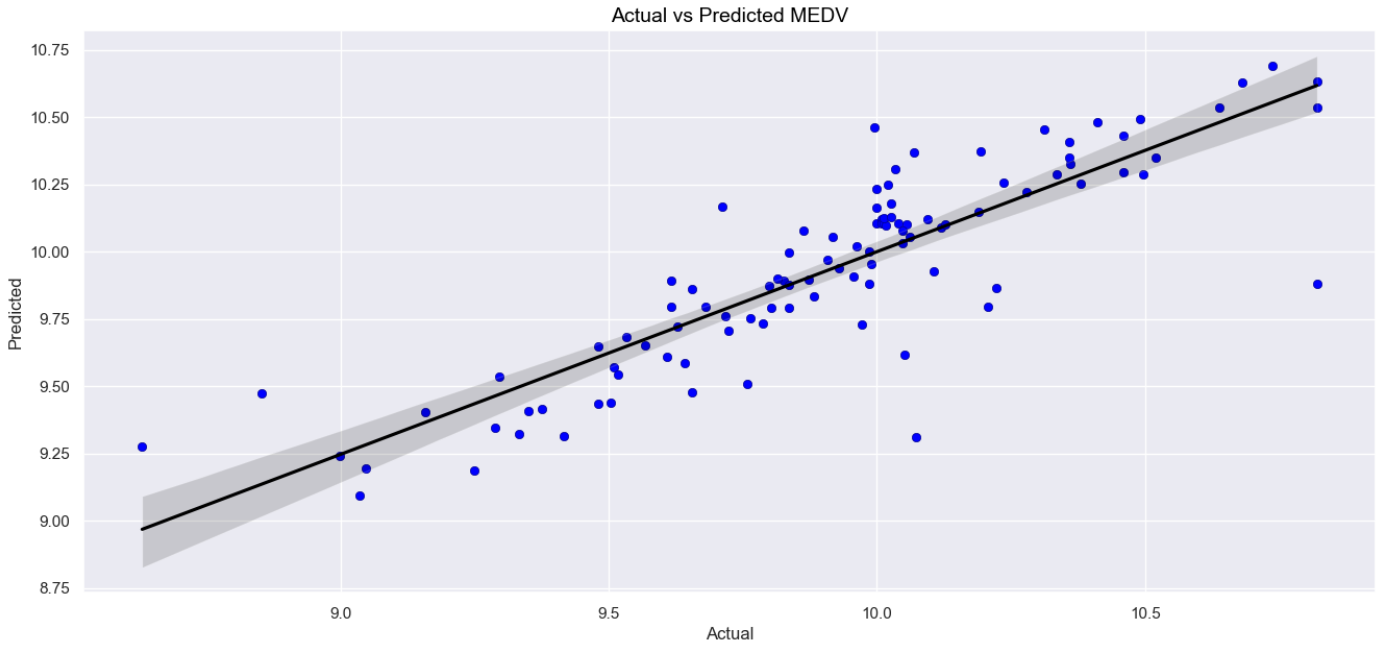
Model : Ridge(alpha=1)

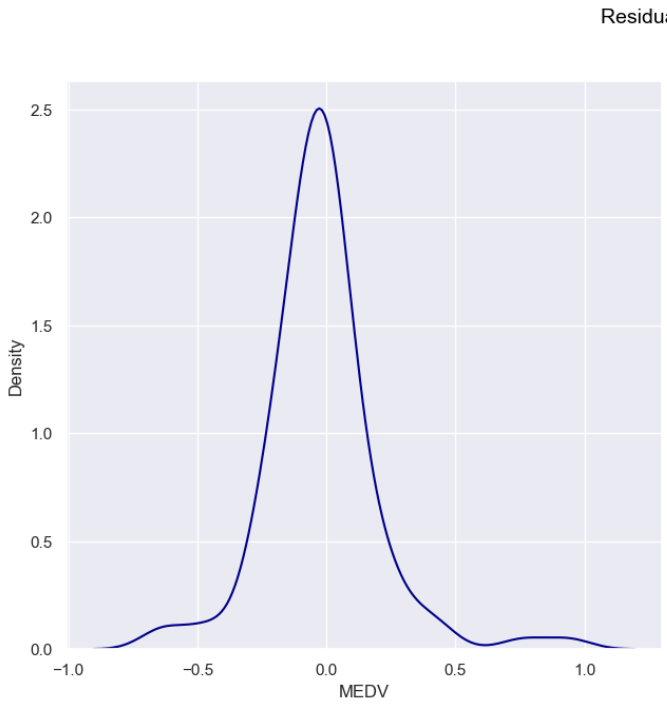
r2 Score

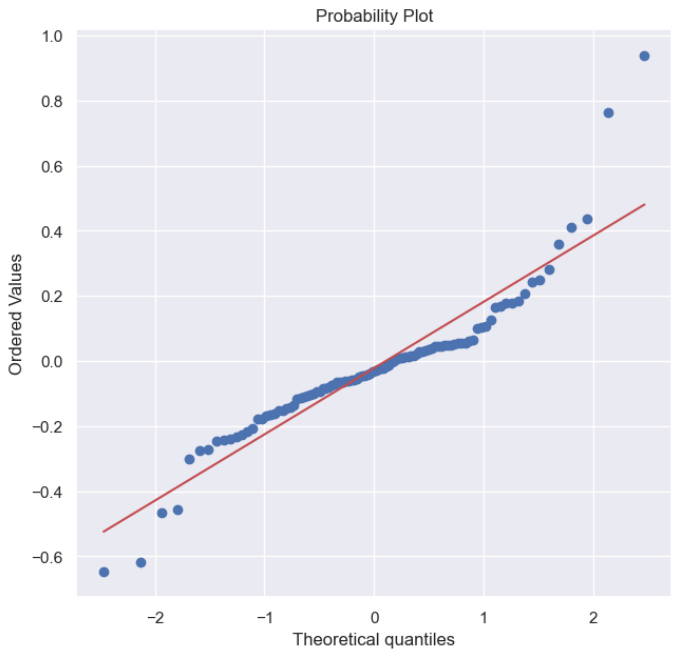
r2_score_train : 0.753
r2_score_test : 0.745
Variance : 0.8%
Adjusted_r2_score_train : 0.748
Adjusted_r2_score_test : 0.72

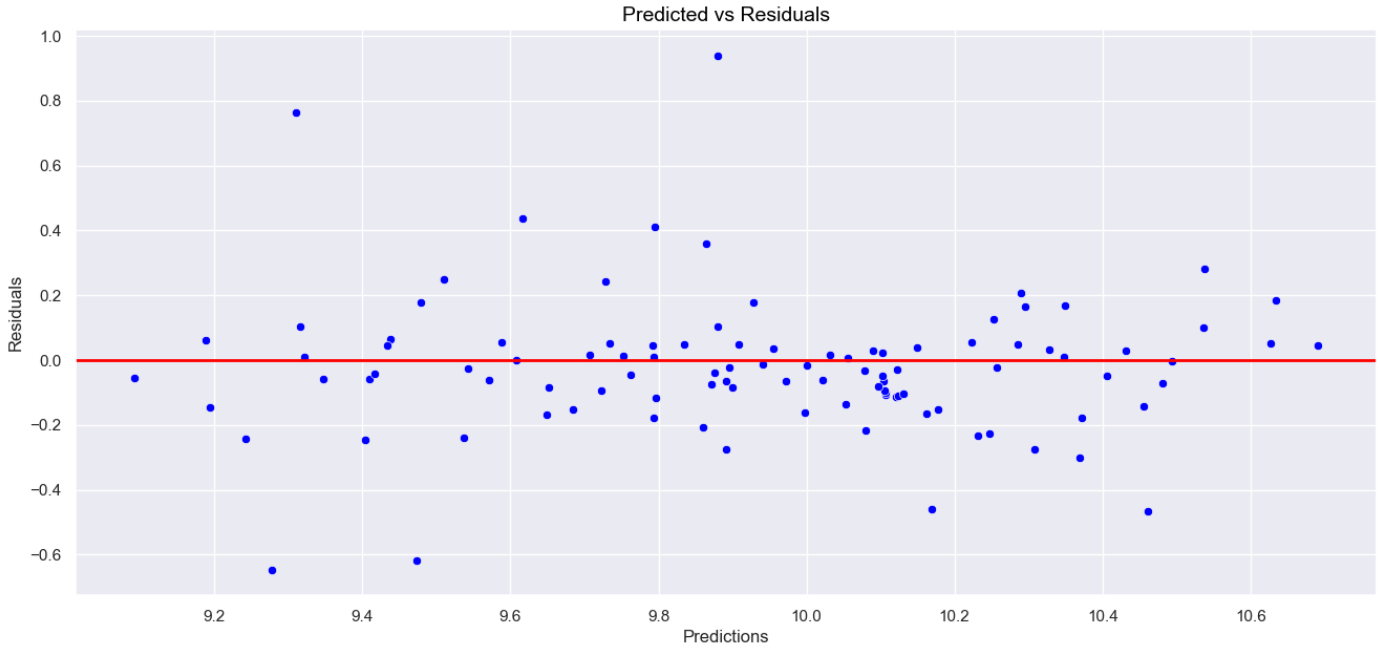
Model Measurements (Metrics)

MSE : 0.05
RMSE : 0.22
MAE : 0.14





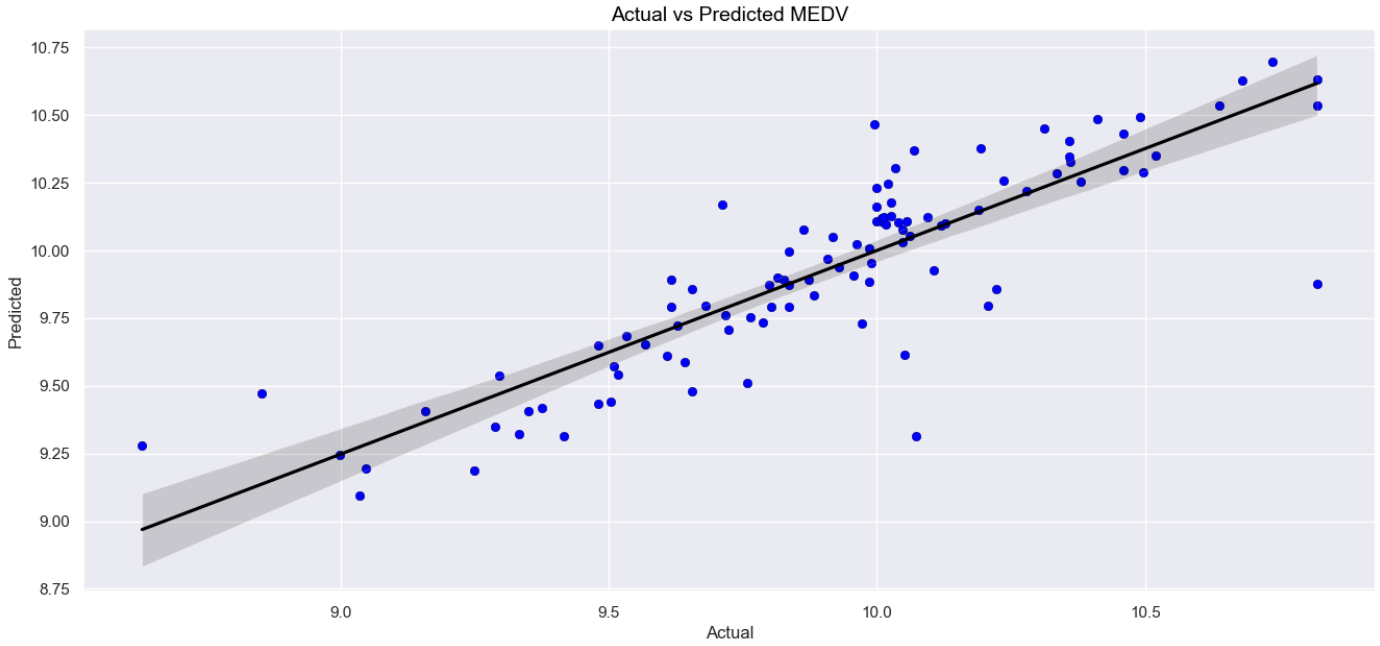


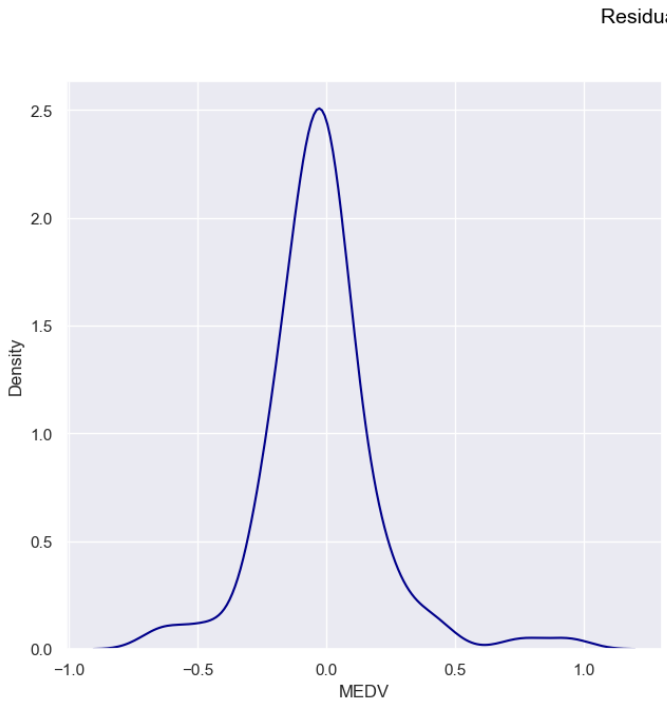


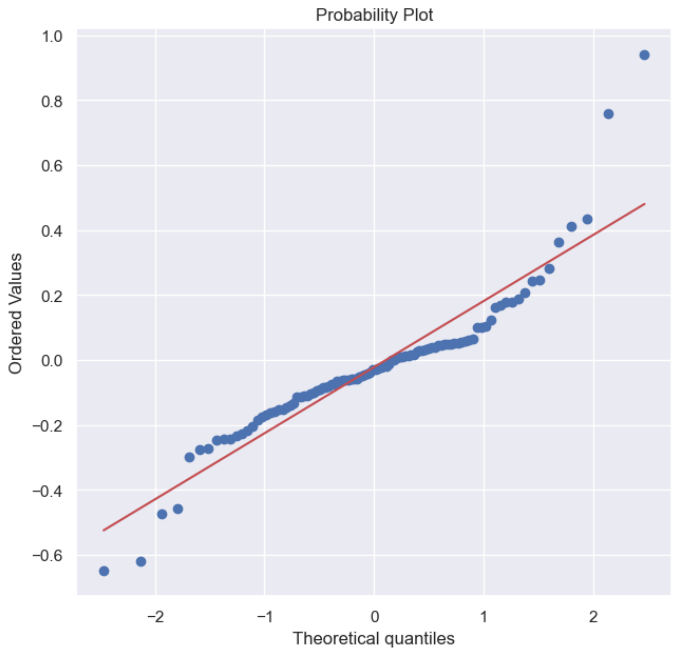
ElasticNet

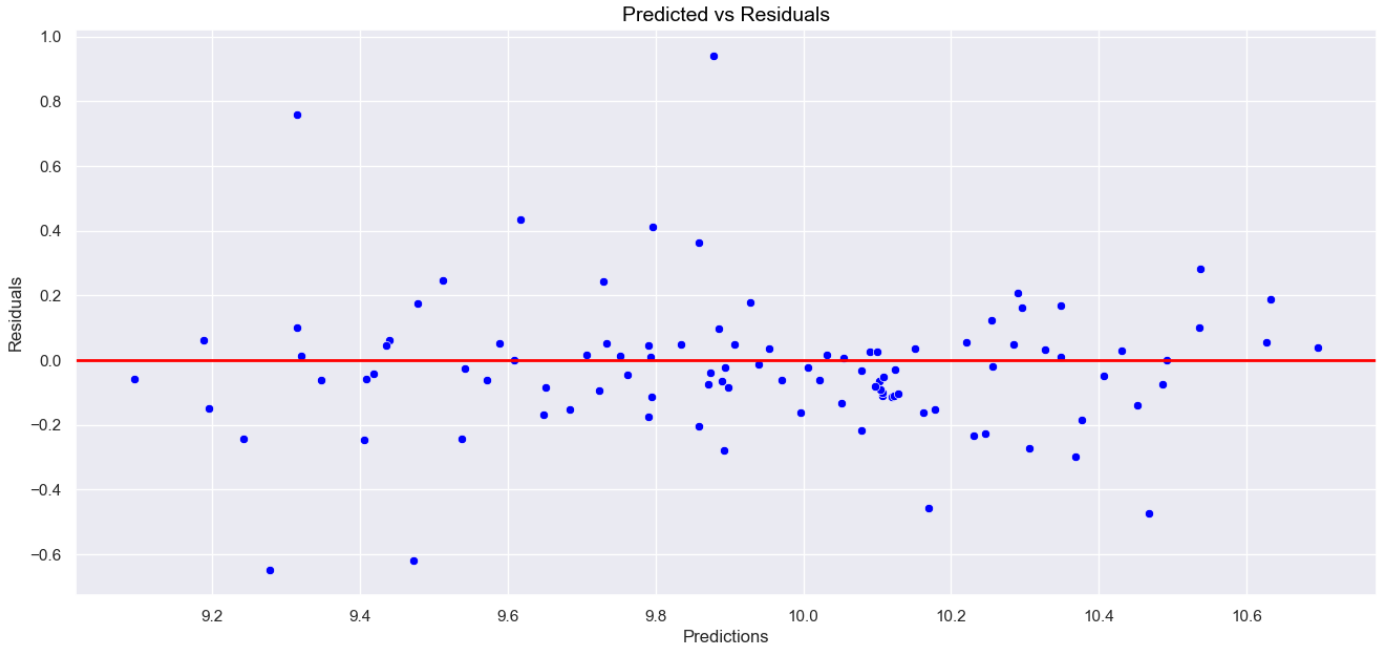
In [56]:
```
# param_grid = {"l1_ratio" : [.1, .5, .7, .9, .50, .95, .97, .99, 1],
#               "alpha" : [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 10],
#               "max_iter" : [1000]}

# grid_model = GridSearchCV(ElasticNet(), param_grid, scoring = "neg_mean_squared_error").fit(X_

# print(grid_model.best_params_)

elnt_model, elnt_metrics = regression_model(ElasticNet(alpha = 0.001, l1_ratio = 0.1, max_iter =
```

```
                                    Model : ElasticNet(alpha=0.001, l1_ratio=0.1)

r2 Score

r2_score_train : 0.753
r2_score_test : 0.745
Variance : 0.8%
Adjusted_r2_score_train : 0.748
Adjusted_r2_score_test : 0.72

Model Measurements (Metrics)

MSE : 0.05
RMSE : 0.22
MAE : 0.14
```


Actual vs Predicted MEDV
Predicted
Actual
10.75
10.50
10.25
10.00
9.75
9.50
9.25
9.00
8.75
9.0
9.5
10.0
10.5


Residuals Distribution
Density
MEDV
2.5
2.0
1.5
1.0
0.5
0.0
−1.0
−0.5
0.0
0.5
1.0

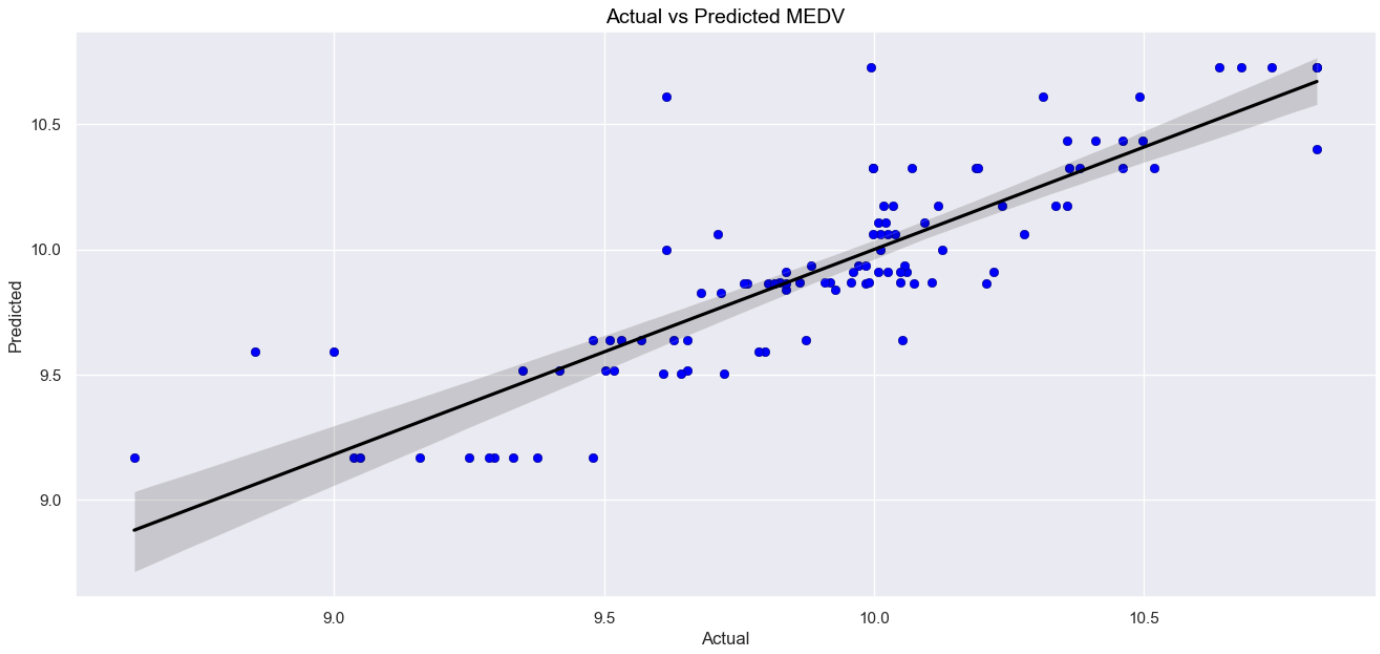

Probability Plot
Ordered Values
Theoretical quantiles
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−2
−1
0
1
2

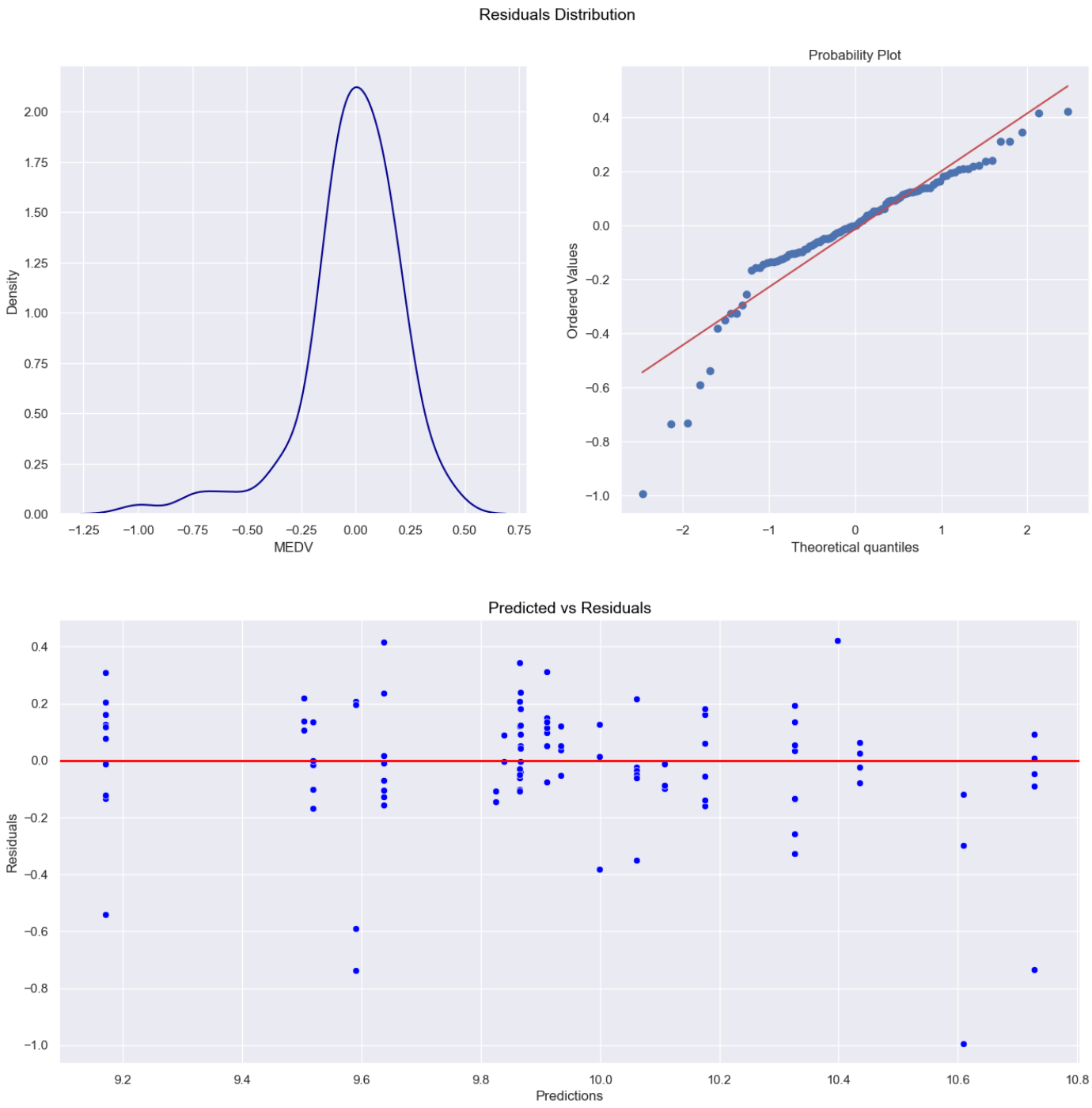

Predicted vs Residuals
Residuals
Predictions
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
9.2
9.4
9.6
9.8
10.0
10.2
10.4
10.6

Decision Tree Regressor

In [57]:
```
# param_grid = {
#     "max_depth" : [None, 1, 2, 3, 4, 5, 10, 15],
#     "min_samples_split" : [2, 5, 10, 20, 30],
#     "min_samples_leaf" : [1, 2, 3, 4, 5],
#     "max_features" : ['auto', 'sqrt', 'log2']
# }

# grid_model = GridSearchCV(DecisionTreeRegressor(), param_grid, scoring = "neg_mean_squared_err

# print(grid_model.best_params_)

dt_model, dt_metrics = regression_model(DecisionTreeRegressor(max_depth = 15, max_features = "au
```

```
                                        Model : DecisionTreeRegressor(max_depth=15, max_features
='auto', min_samples_leaf=5,
                      min_samples_split=30)

r2 Score

r2_score_train : 0.856
r2_score_test : 0.72
Variance : 13.6%
Adjusted_r2_score_train : 0.853
Adjusted_r2_score_test : 0.692

Model Measurements (Metrics)

MSE : 0.05
RMSE : 0.22
MAE : 0.15
```





Random Forest Regressor

In [58]:
```
# param_grid = {"max_depth" : [2, 3, 4, 10],
#               "bootstrap" : [True, False],
#               "max_features" : ['auto', 'sqrt', 'log2', None],
#               "min_samples_leaf" : [1, 2, 4],
#               "min_samples_split" : [2, 5, 10],
#               "n_estimators" : [10, 50, 100, 150, 200]
#               }

# grid_model = GridSearchCV(RandomForestRegressor(), param_grid, scoring = "r2").fit(X_train, y_

# print(grid_model.best_params_)

rf_model, rf_metrics = regression_model(RandomForestRegressor(bootstrap = True, max_depth = 10,
```

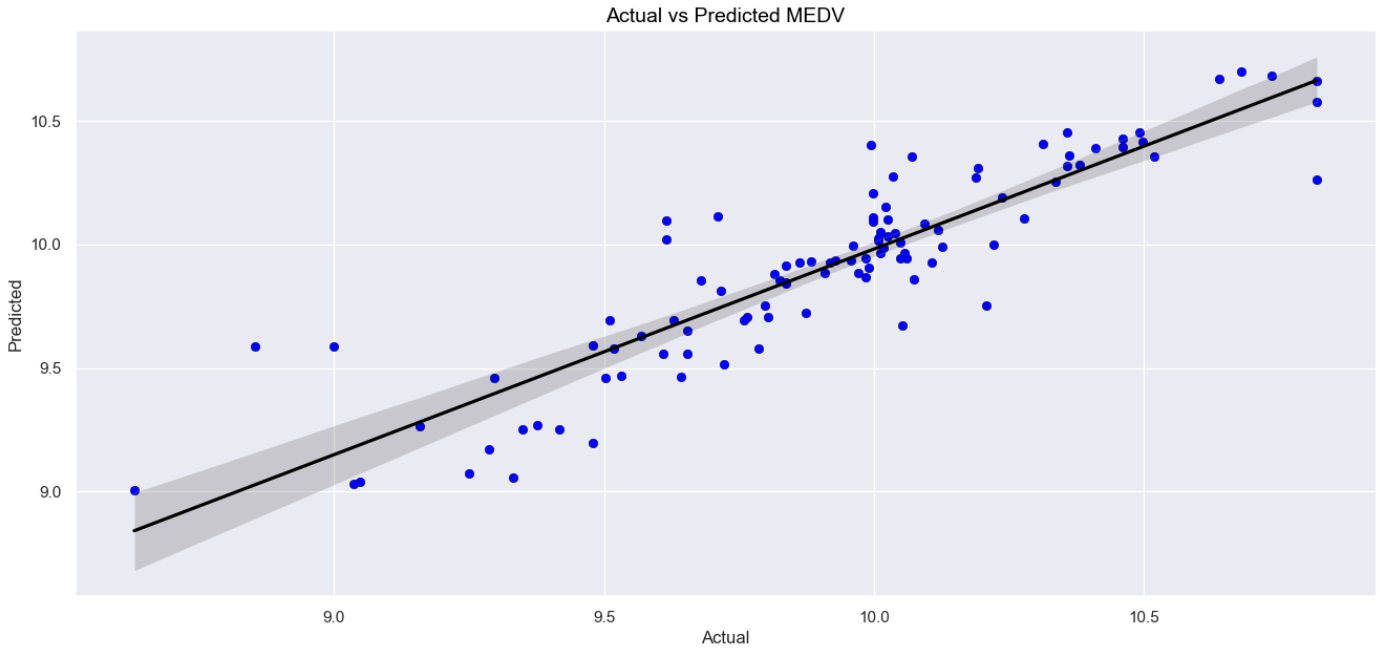
Model : RandomForestRegressor(max_depth=10, max_features='sqrt')

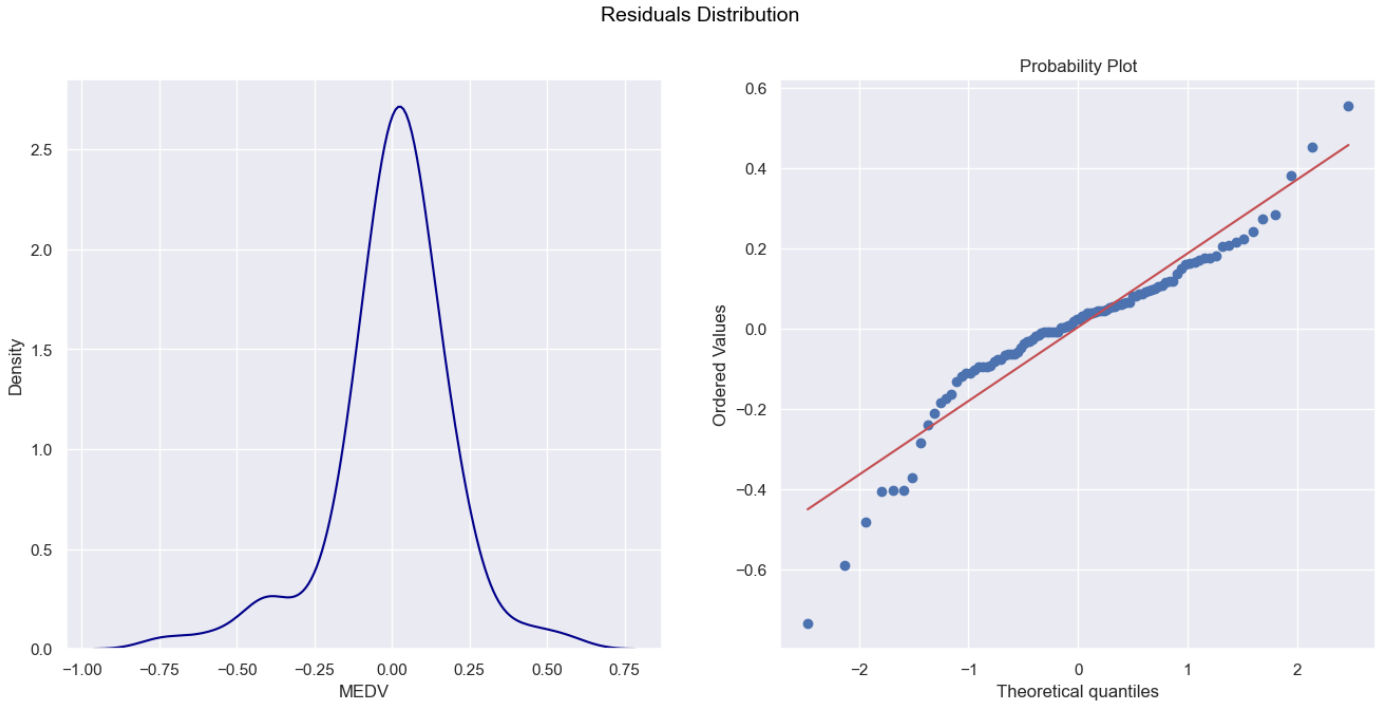
r2 Score

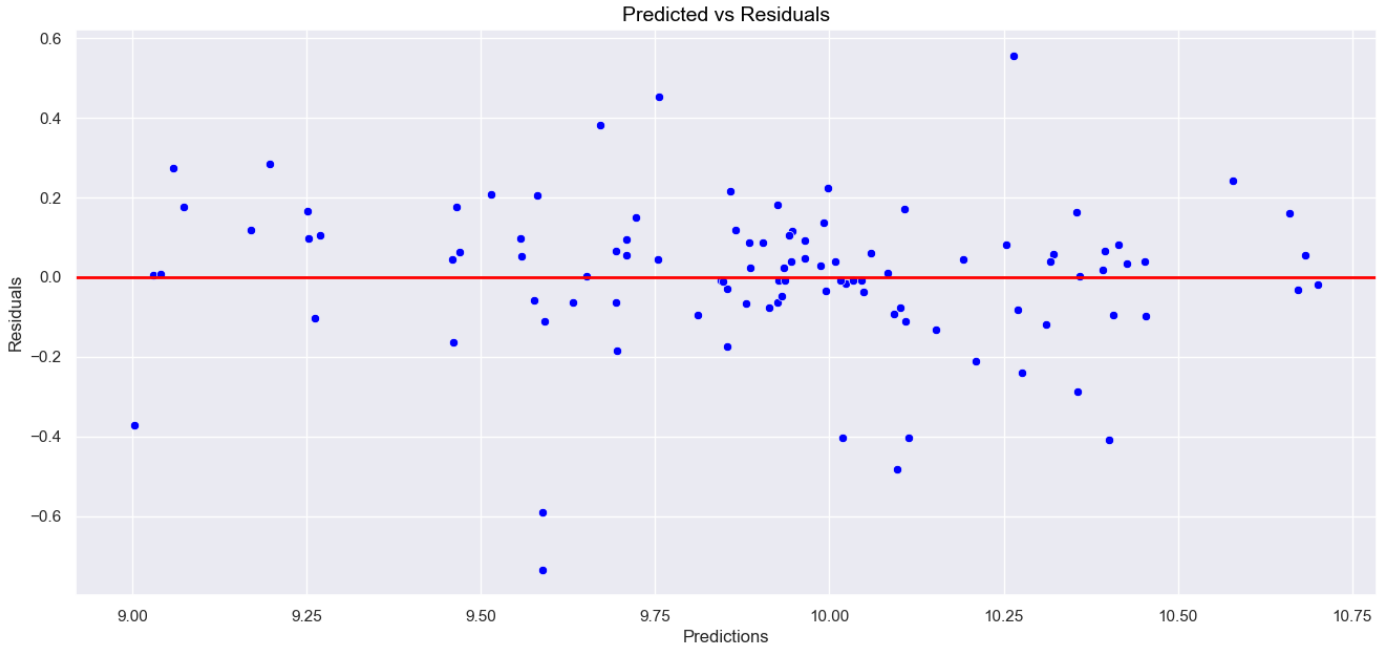
r2_score_train : 0.97
r2_score_test : 0.802
Variance : 16.8%
Adjusted_r2_score_train : 0.969
Adjusted_r2_score_test : 0.783

Model Measurements (Metrics)

MSE : 0.04
RMSE : 0.2
MAE : 0.13



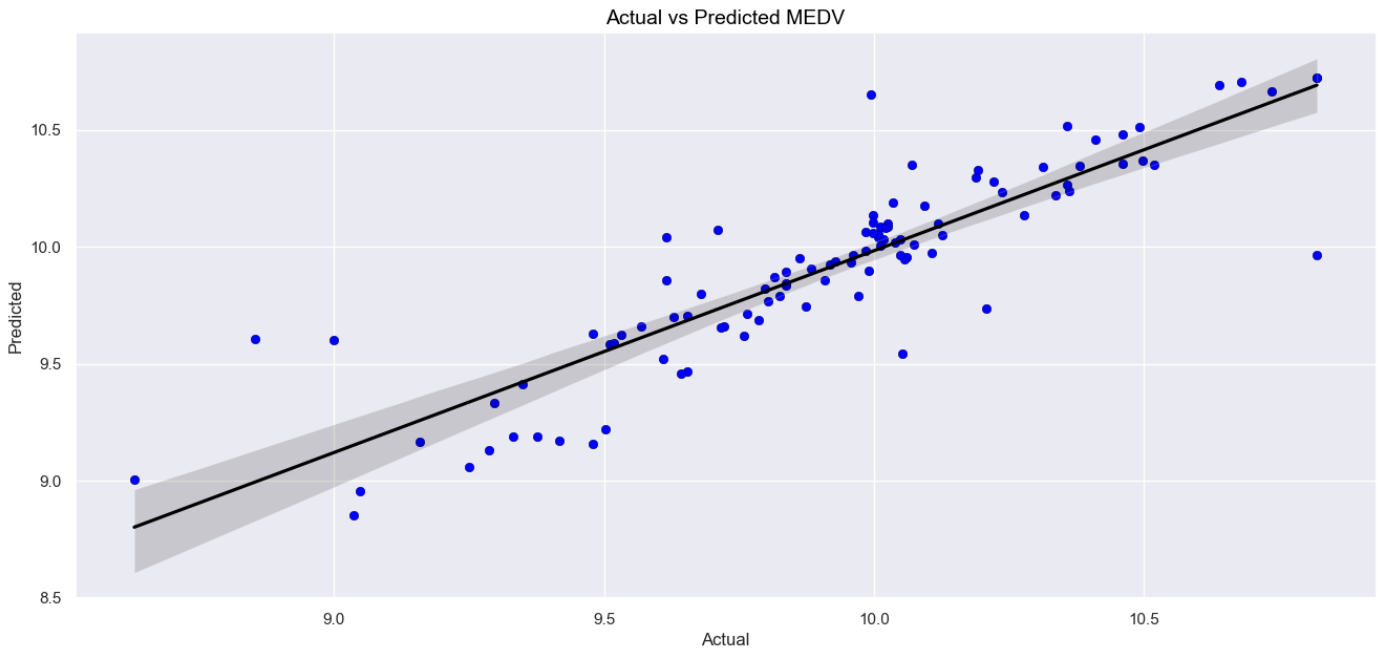


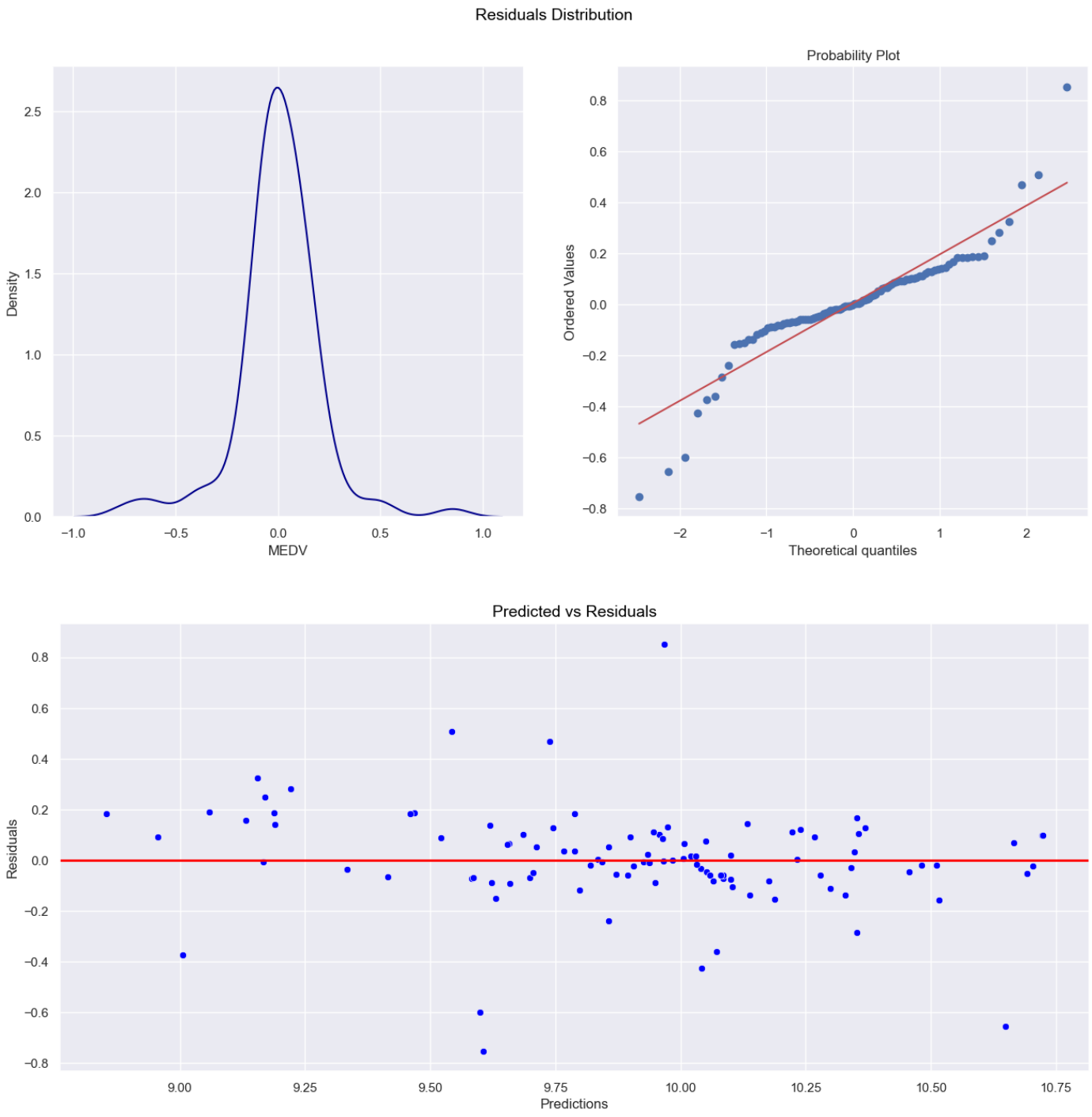


Support Vector Regressor

In [59]:
```
svr_model, svr_metrics = regression_model(SVR(), X_sc, y_log)
```

```
                    Model : SVR()

r2 Score

r2_score_train : 0.879
r2_score_test : 0.773
Variance : 10.6%
Adjusted_r2_score_train : 0.876
Adjusted_r2_score_test : 0.751

Model Measurements (Metrics)

MSE : 0.04
RMSE : 0.2
MAE : 0.13
```

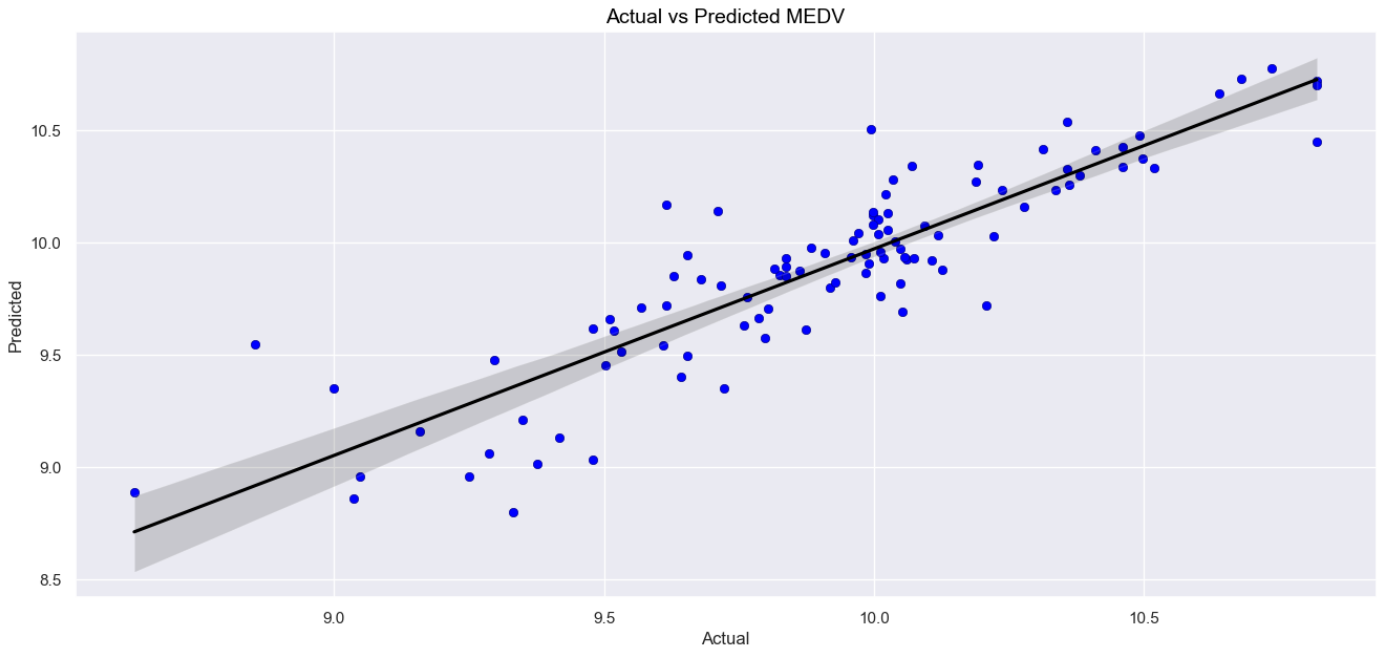


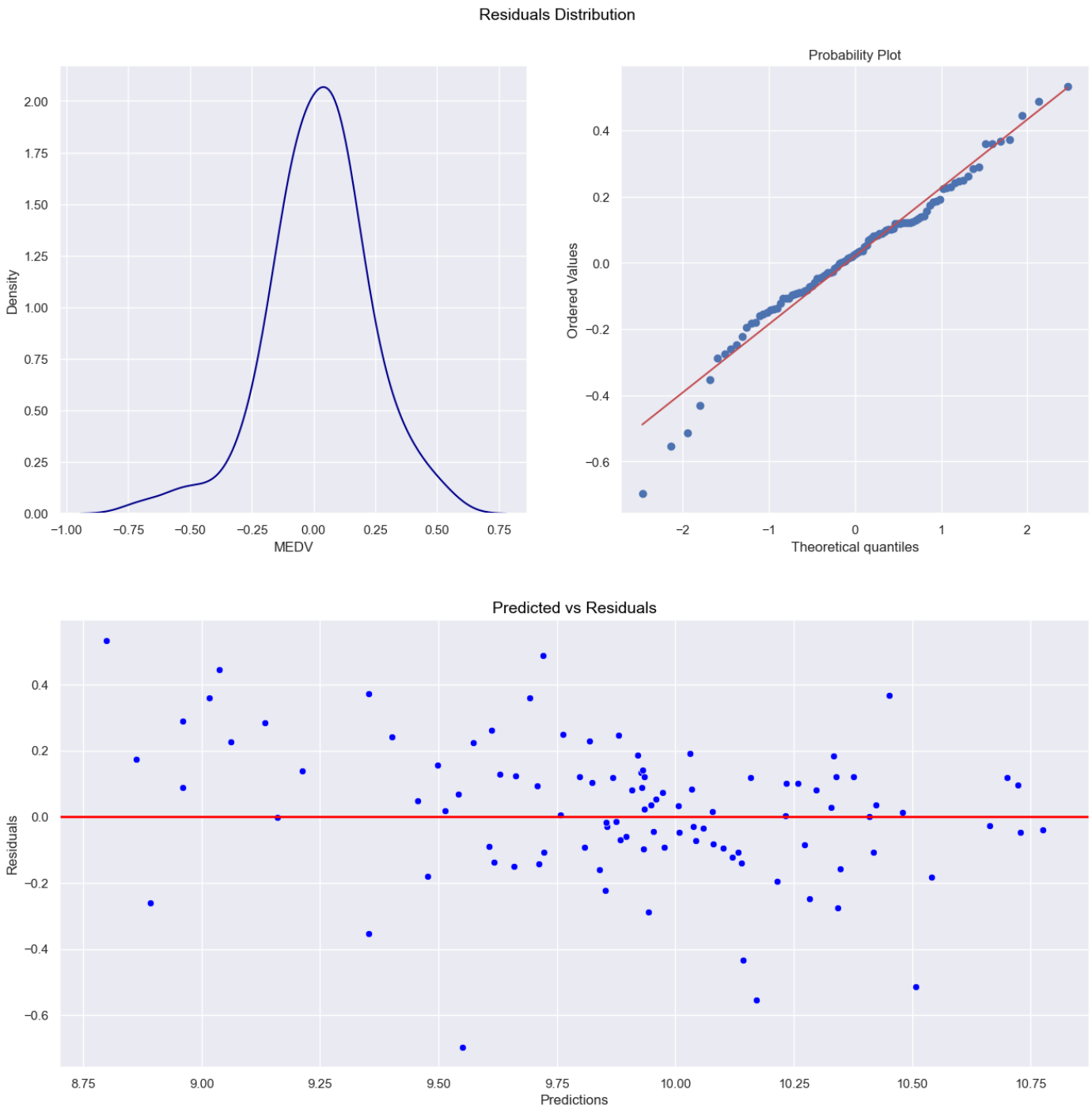


XGB Regressor

In [60]:
```
xgb_model, xgb_metrics = regression_model(XGBRegressor(), X_sc, y_log)
```

```
                                        Model : XGBRegressor(base_score=None, booster=None, call
backs=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)

r2 Score

r2_score_train : 1.0
r2_score_test : 0.764
Variance : 23.6%
Adjusted_r2_score_train : 1.0
Adjusted_r2_score_test : 0.741

Model Measurements (Metrics)

MSE : 0.04
RMSE : 0.2
MAE : 0.16
```





Final Results

In [61]:
```
pd.DataFrame(
    [lr_log_metrics, lasso_metrics, ridge_metrics, elnt_metrics, dt_metrics, rf_metrics, svr_met
    columns = ["r2_score (Train)", "r2_score (Test)", "Adj_r2 (Train)", "Adj_r2 (Test)", "Varian
    index = ["Linear Regression", "Lasso Regression", "Ridge Regression", "Elastic Net", "Decisi
)
```

Out[61]:

	r2_score (Train)	r2_score (Test)	Adj_r2 (Train)	Adj_r2 (Test)	Variance	MSE	RMSE	MAE
Linear Regression	**0.753**	**0.745**	**0.748**	**0.720**	**0.8%**	**0.05**	**0.22**	**0.14**
Lasso Regression	**0.753**	**0.745**	**0.748**	**0.721**	**0.8%**	**0.05**	**0.22**	**0.14**
Ridge Regression	**0.753**	**0.745**	**0.748**	**0.720**	**0.8%**	**0.05**	**0.22**	**0.14**
Elastic Net	**0.753**	**0.745**	**0.748**	**0.720**	**0.8%**	**0.05**	**0.22**	**0.14**
Decision Tree Regression	**0.856**	**0.720**	**0.853**	**0.692**	**13.6%**	**0.05**	**0.22**	**0.15**
Random Forest Regression	**0.970**	**0.802**	**0.969**	**0.783**	**16.8%**	**0.04**	**0.20**	**0.13**
Support Vector Regression	**0.879**	**0.773**	**0.876**	**0.751**	**10.6%**	**0.04**	**0.20**	**0.13**
XGB Regressor	**1.000**	**0.764**	**1.000**	**0.741**	**23.6%**	**0.04**	**0.20**	**0.16**