Report

BOSTON HOUSE PRICE PREDICTION

Made By:

Rishabh Mathur

Email: Vigeagio@gmail.com

Phone No.: 9958814122

Content

Overview

We are going to work on a dataset that consists of information about the location of the house, price, and other aspects such as square feet, etc. To work with these sorts of data, we need to determine which columns are relevant to us and which aren't. It is our main goal to create a model that can predict the house's price based on other variables. We are planning to use Linear Regression and Random Forest Regressor for this dataset and see if it gives us reasonable accuracy or not.

In this Report, we are going to do implementing a salable model for predicting the house price prediction using some of the regression techniques based on some of the features in the dataset which is called Boston House Price Prediction. There are some processing techniques for creating a model.

Motivation

The motivation behind it is to learn more about Boston's house prices as well as to have an idea of what I can do during the lockdown.

Analysing the Problem Statement

Housing prices are an indication of the economy, and both buyers and sellers pay attention to housing price ranges. Most home buyers don't start by describing the height of the basement ceiling or the proximity to an east-west railroad when describing their dream house. The data set from this playground competition shows that much more influences price negotiations than the number of bedrooms or white-picket fences.

Dataset Description

Housing prices are an important reflection of the economy, and housing price ranges are of great interest to both buyers and sellers. In this project, house prices will be predicted given explanatory variables that cover many aspects of residential houses. The goal of this project is to create a regression model that is able to accurately estimate the price of the house given its features. This dataset was made for predicting the Boston House Price Prediction. Here I just show all of the features for each house separately. Such as the Number of Rooms, the Crime rate in the House's Area, and so on.

1. CRIM per capital crime rate by town

2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.

3. INDUS proportion of non-retail business acres per town

4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)

5. NOX nitric oxides concentration (parts per 10 million)

6. RM average number of rooms per dwelling

7. AGE proportion of owner-occupied units built prior to 1940

8. DIS weighted distances to five Boston employment centers

9. RAD index of accessibility to radial highways

10.TAX full-value property-tax rate per 10,000 USD

11.PTRATIO pupil-teacher ratio by town

12.Black $1000(Bk — 0.63)^2$ where Bk is the proportion of blacks by town

13.LSTAT % lower status of the population

Algorithms Used

The major aim of in this project is to predict the house prices based on the features using some of the regression techniques and algorithms.

1. Linear Regression

2. Random Forest Regressor

Technologies and Libraries Used

1. Python (Technology)
2. Numpy (Library)
3. Pandas (Library)
4. Seaborn (Library)
5. Matplotlib (Library)
6. Scikit Learn (Library)

Data Collection

Code for collecting data from CSV file into Jupyter Notebook

```
# Import libraries
import numpy as np
import pandas as pd
# Import the dataset
df = pd.read_csv("train.csv")
df.head()
```

	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	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	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	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	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	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

Data Pre-processing

In this Boston Dataset we need not to clean the data. The dataset already cleaned when we download from the classroom.

```
# Shape of dataset
print("Shape of Training dataset:", df.shape)
Shape of Training dataset: (333, 15)
# Checking null values for training dataset
df.isnull().sum()


ID         0
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

Exploratory Data Analysis (EDA)

Data sets are analyzed to summarize their main characteristics by exploratory data analysis (EDA), often using visual methods. The data can be analyzed in any way, but the primary goal of EDA is to find out what the data can tell us beyond formal modeling or hypothesis testing.

```
# Information about the dataset features
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 351 entries, 0 to 350
Data columns (total 15 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   ID       351 non-null    int64
 1   crim     351 non-null    float64
 2   zn       351 non-null    float64
 3   indus    351 non-null    float64
 4   chas     351 non-null    int64
 5   nox      351 non-null    float64
 6   rm       351 non-null    float64
 7   age      351 non-null    float64
 8   dis      351 non-null    float64
 9   rad      351 non-null    int64
 10  tax      351 non-null    int64
 11  ptratio  351 non-null    float64
 12  black    351 non-null    float64
 13  lstat    351 non-null    float64
 14  Price    351 non-null    float64
dtypes: float64(11), int64(4)
memory usage: 41.3 KB
```

```
# Describe
df.describe()
```

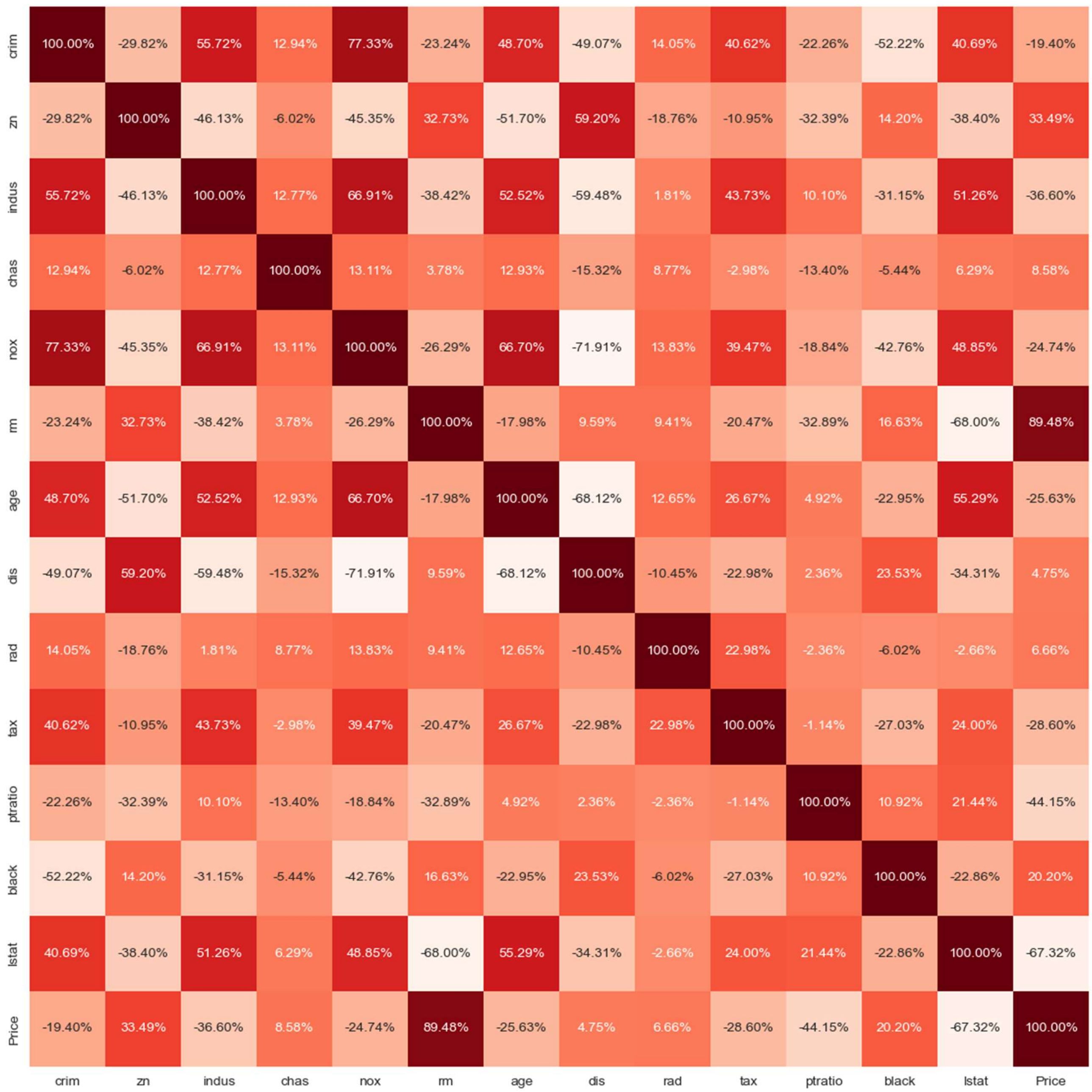
	ID	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	
count	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.000000	351.0
mean	175.000000	0.401659	15.327635	8.435670	0.076923	0.510737	6.403900	60.817949	4.420862	4.472934	310.344729	17.707692	380.4
std	101.469207	0.641716	25.605040	6.088947	0.266850	0.102256	0.676424	28.393094	1.968666	1.615543	67.577707	2.198252	40.4
min	0.000000	0.006320	0.000000	0.460000	0.000000	0.385000	4.903000	2.900000	1.321600	1.000000	188.000000	12.600000	70.8
25%	87.500000	0.057845	0.000000	4.025000	0.000000	0.437450	5.949500	36.150000	2.768500	4.000000	264.000000	16.100000	383.67
50%	175.000000	0.132620	0.000000	6.200000	0.000000	0.493000	6.266000	62.000000	4.095200	4.000000	304.000000	17.900000	392.6
75%	262.500000	0.404865	22.000000	10.010000	0.000000	0.544000	6.733000	88.450000	5.871800	5.000000	358.000000	19.100000	396.2
max	350.000000	4.097400	100.000000	25.650000	1.000000	0.871000	8.725000	100.000000	9.222900	8.000000	469.000000	21.200000	396.9

Feature Observation

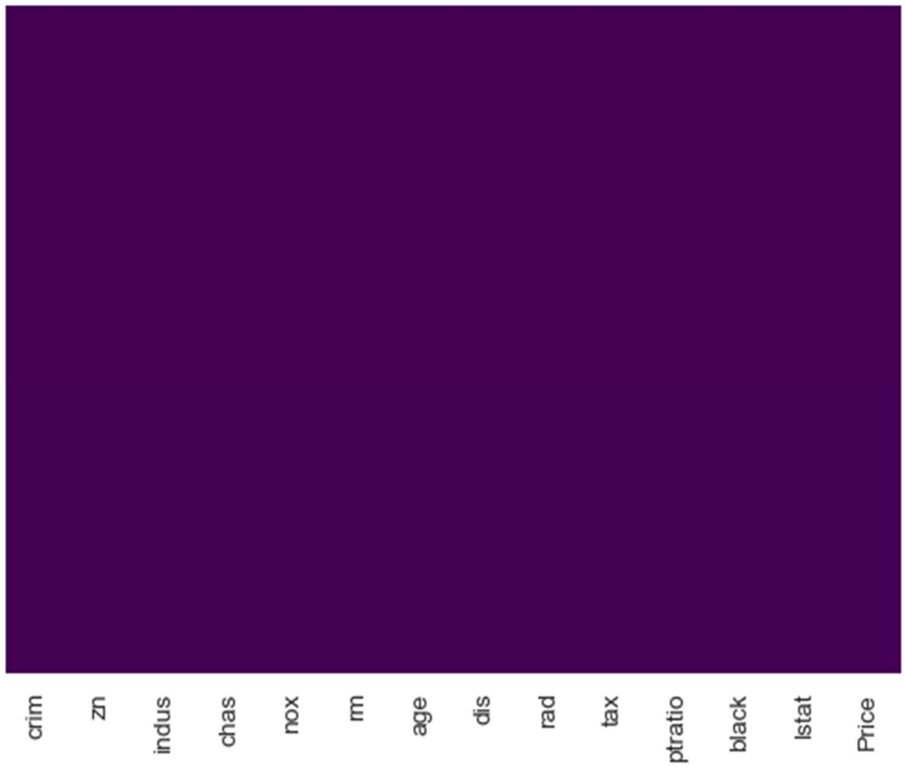
```
# Finding out the correlation between the features
corr = df.corr()
corr.shape
```

First Understanding the correlation of features between target and other features

```
# Plotting the heatmap of correlation between features
plt.figure(figsize=(14,14))
sns.heatmap(corr, cbar=False, square= True, fmt='.2%', annot=True, cmap='Greens')
```

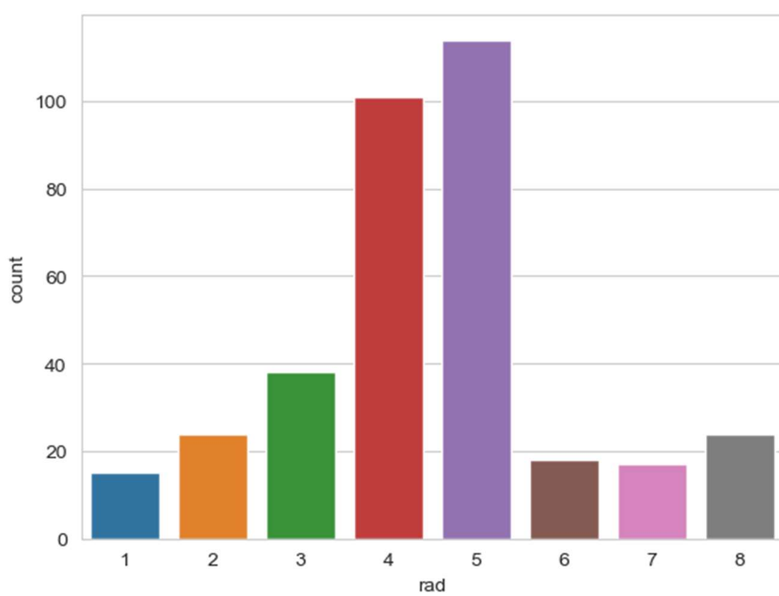


# Checking the null values using heatmap
# There is any null values are occupyed here
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')



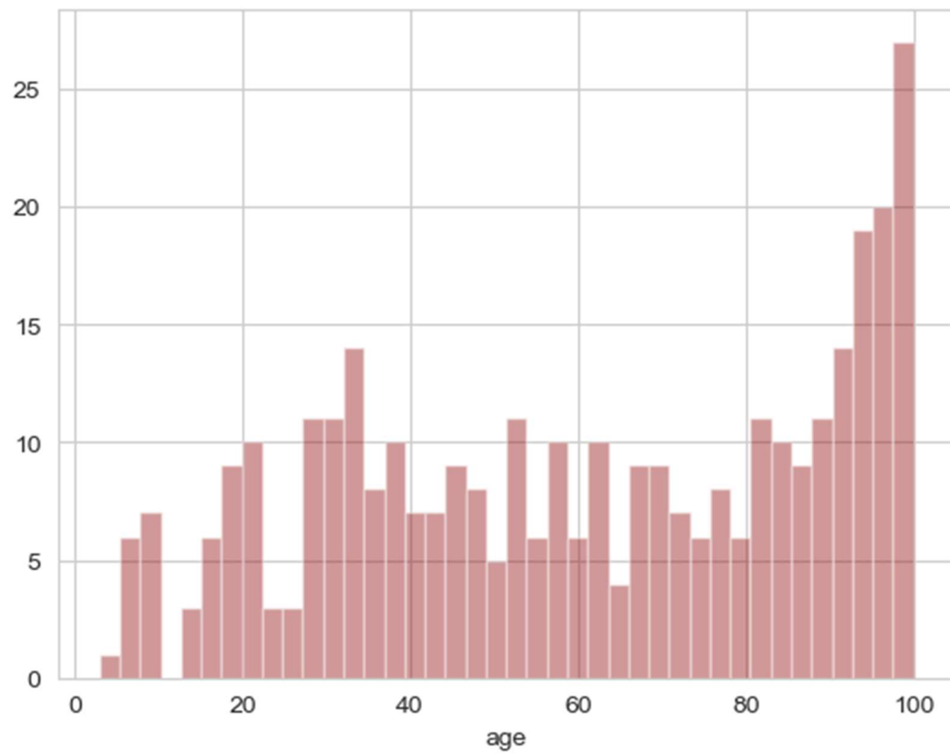
Note: There are no null or missing values here.

sns.set_style('whitegrid')
sns.countplot(x='rad',data=df)



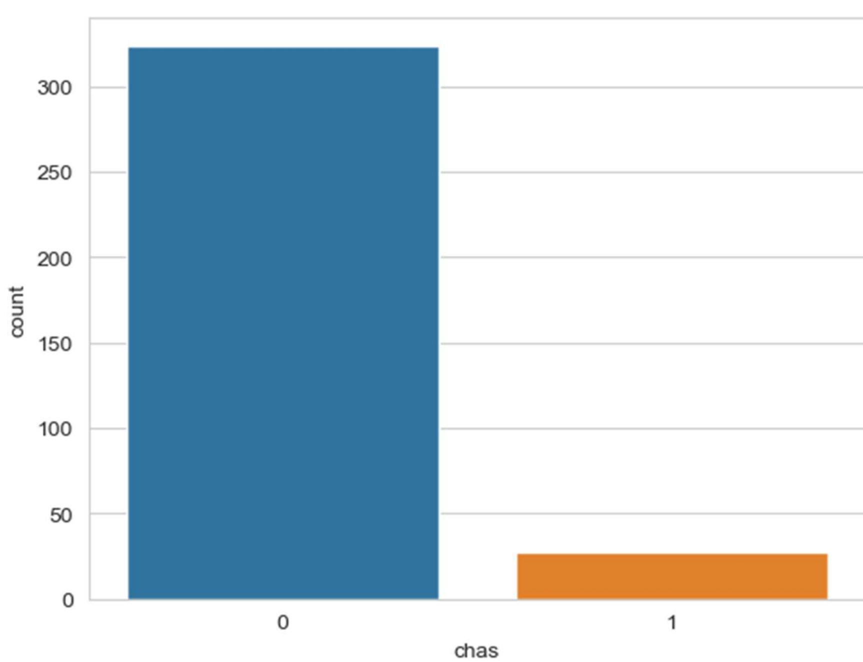
The graph shows the number of owner-occupied units that were built prior to 1940

```
sns.distplot(df['age'].dropna(),kde=False,color='darkred',bins=40)
```



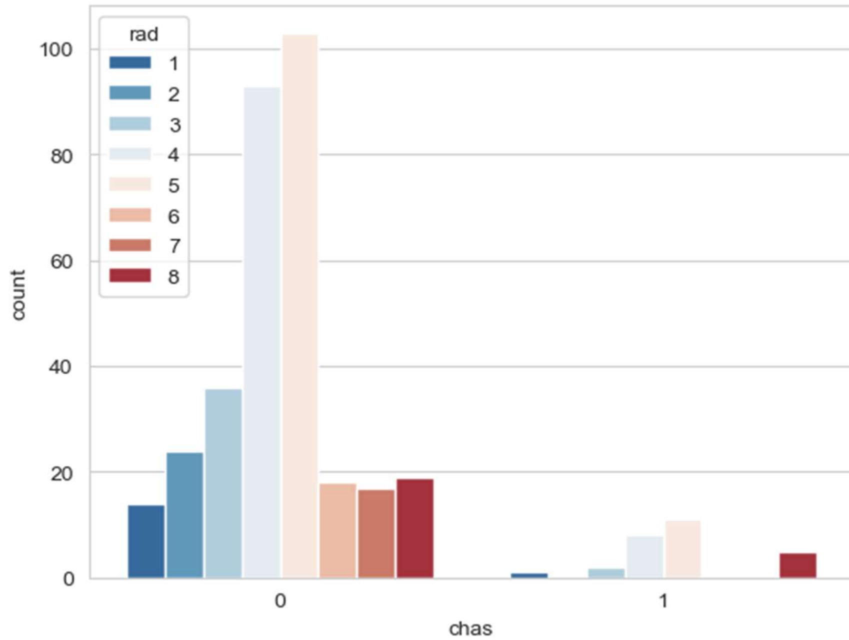
Plotting the graph for Charles River dummy variable

```
sns.set_style('whitegrid')
sns.countplot(x='chas',data=df)
```



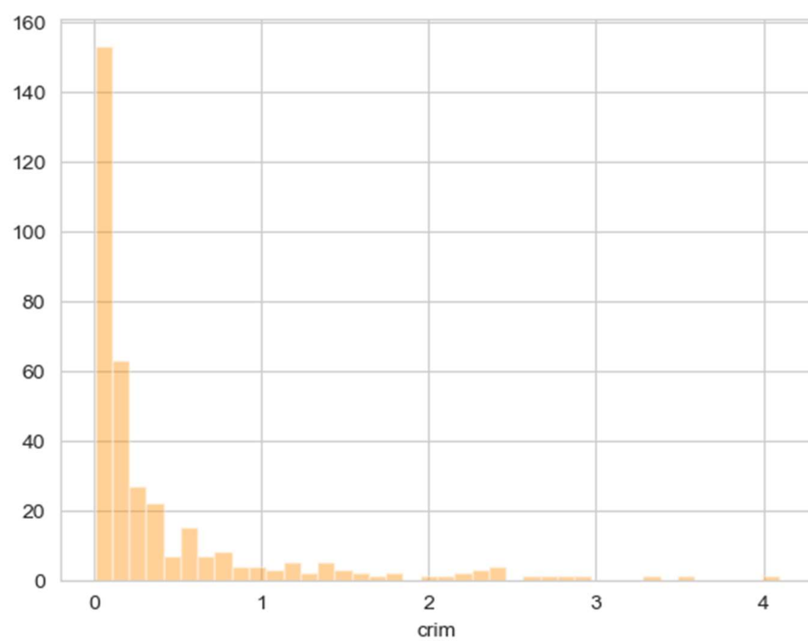
Plotting a graph showing relation between Chas and Rad

```
sns.set_style('whitegrid')
sns.countplot(x='chas',hue='rad',data=df,palette='RdBu_r')
```



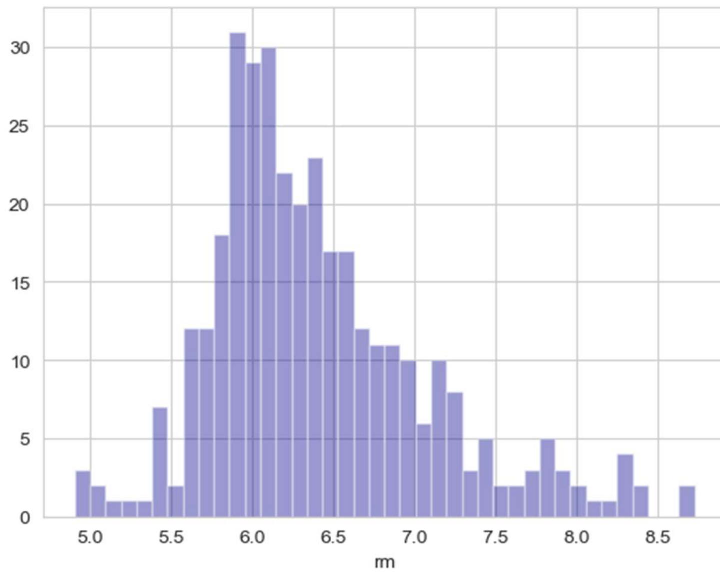
Plotting the graph for per capita crime rate by town

```
sns.distplot(df['crim'].dropna(),kde=False,color='darkorange',bins=40)
```



Plotting the graph on Average number of rooms Distribution

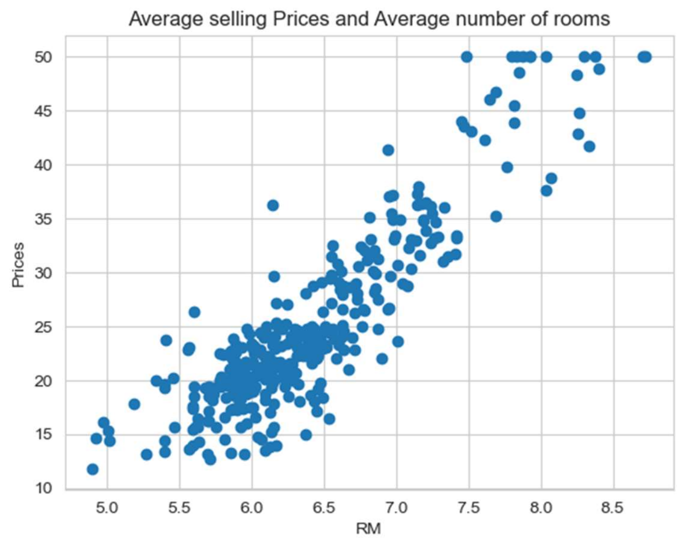
sns.distplot(df['rm'].dropna(),kde=False,color='darkblue',bins=40)

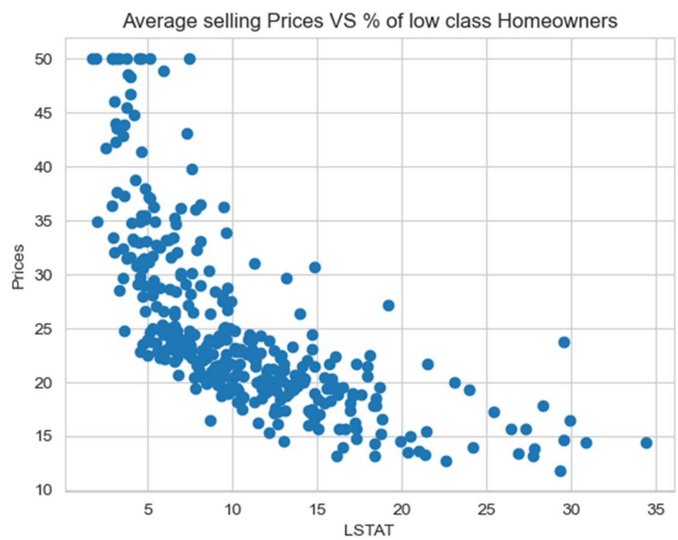


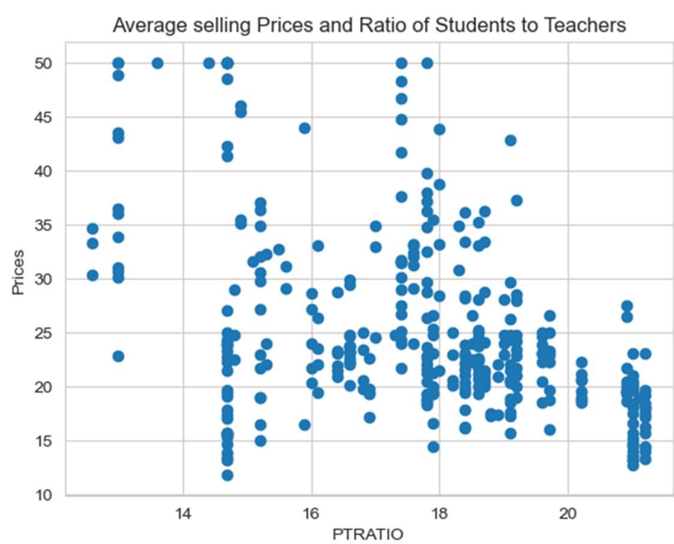

```
#                    RM VS PRICES
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.scatter(df['rm'], df['Price'])
#Lables & Title
plt.title("Average selling Prices and Average number of rooms")
plt.xlabel("RM")
plt.ylabel("Prices")
plt.show()

#                    LSTAT VS PRICES
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.scatter(df['lstat'], df['Price'])
plt.title("Average selling Prices VS % of low class Homeowners")
plt.xlabel("LSTAT")
plt.ylabel("Prices")
plt.show()

#                    PTRATIO VS PRICES
fig=plt.figure()
ax=fig.add_subplot(1, 1, 1)
ax.scatter(df['ptratio'], df['Price'])
plt.title("Average selling Prices and Ratio of Students to Teachers")
plt.xlabel("PTRATIO")
plt.ylabel("Prices")
plt.show()
```


Average selling Prices and Average number of rooms
Prices
RM
10
15
20
25
30
35
40
45
50
5.0
5.5
6.0
6.5
7.0
7.5
8.0
8.5


Average selling Prices VS % of low class Homeowners
Prices
LSTAT
10
15
20
25
30
35
40
45
50
5
10
15
20
25
30
35


Average selling Prices and Ratio of Students to Teachers
Prices
PTRATIO
10
15
20
25
30
35
40
45
50
14
16
18
20

Feature Selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

```
# Lets try to understand which are important feature for this dataset
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
X = df.iloc[:,0:13]
y = df.iloc[:,-1] #target column i.e price range
```

Note: If we want to identify the best features for the target variables. We should make sure that the target variable should be int Values. That's why I convert into the int value from the floating point value

```
y = np.round(df['Price'])
#Apply SelectKBest class to extract top 5 best features
bestfeatures = SelectKBest(score_func=chi2, k=5)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
# Concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score'] #naming the dataframe columns
featureScores
```

	Specs	Score
0	crim	116.409048
1	zn	3360.496357
2	indus	426.640225
3	chas	33.428537
4	nox	2.280787
5	rm	20.688799
6	age	1575.417982
7	dis	44.701259
8	rad	23.356787
9	tax	932.795817
10	ptratio	31.118120
11	black	263.409996
12	lstat	756.911851

Displaying the Best Five Features

print(featureScores.nlargest(5,'Score'))

```
     Specs        Score
1       zn  3360.496357
6      age  1575.417982
9      tax   932.795817
12   lstat   756.911851
2    indus   426.640225
```

Model Building

Linear Regression

```
# Values Assigning
X = df.iloc[:,0:13]
y = df.iloc[:,-1]
```

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=0)
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train,y_train)
```

Random Forest Regressor

```
# Values Assigning
X = df.iloc[:,[-1,5,10,4,9]]
y = df.iloc[:,[-1]]
```

Train Test Split

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.20,random_state=0)
from sklearn.ensemble import RandomForestRegressor
reg = RandomForestRegressor()
reg.fit(X_train,y_train)
```

Model Performance

Linear Regression

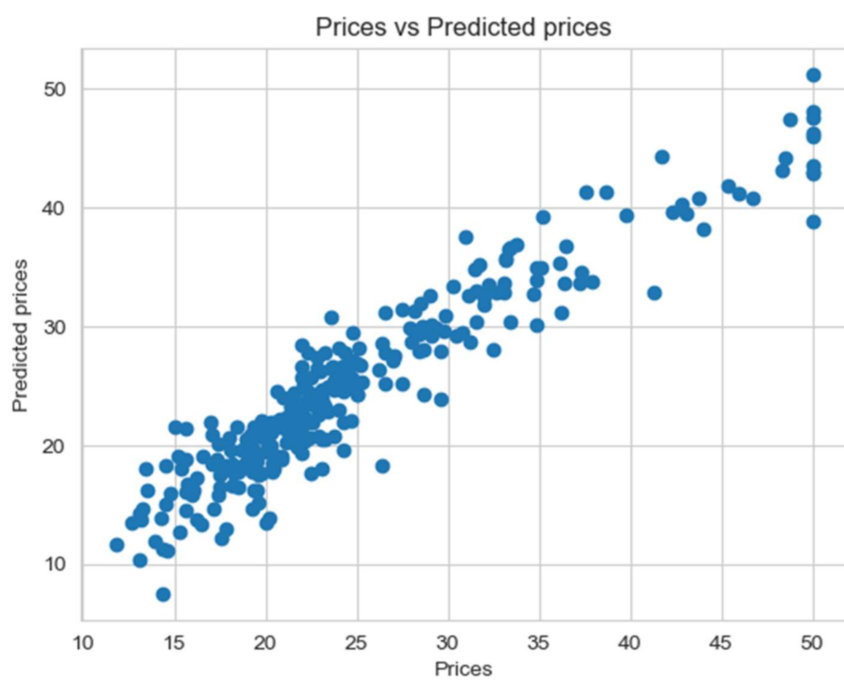
```
y_pred = model.predict(X_train)
print("Training Accuracy:",model.score(X_train,y_train)*100)
print("Testing Accuracy:",model.score(X_test,y_test)*100)
```

```
Training Accuracy: 88.87288540479278
Testing Accuracy: 79.89583263244569
```

```
from sklearn.metrics import mean_squared_error, r2_score
print("Model Accuracy:",r2_score(y,model.predict(X))*100)
```

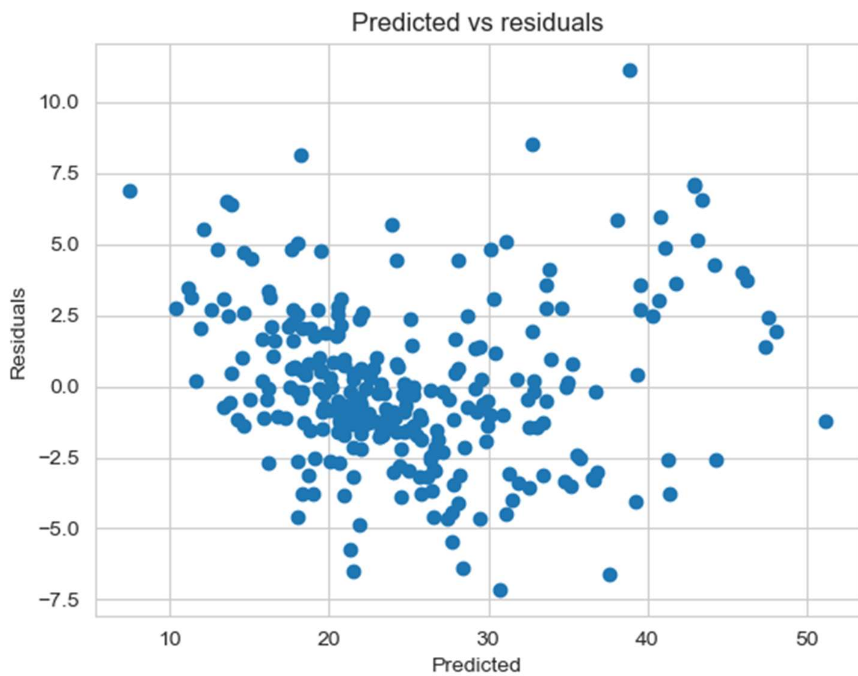
```
Model Accuracy: 87.291549204434
```

```
plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



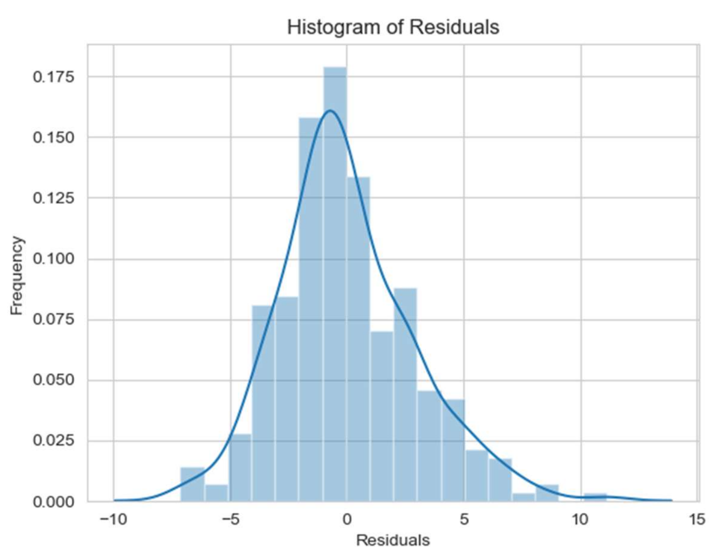
Checking for the Residuals

```
plt.scatter(y_pred,y_train-y_pred)
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```



Checking for the Normality of Errors

```
sns.distplot(y_train-y_pred)
plt.title("Histogram of Residuals")
plt.xlabel("Residuals")
plt.ylabel("Frequency")
plt.show()
```



Random Forest Regressor

```
y_pred = reg.predict(X_train)
print("Training Accuracy:",reg.score(X_train,y_train)*100)
```

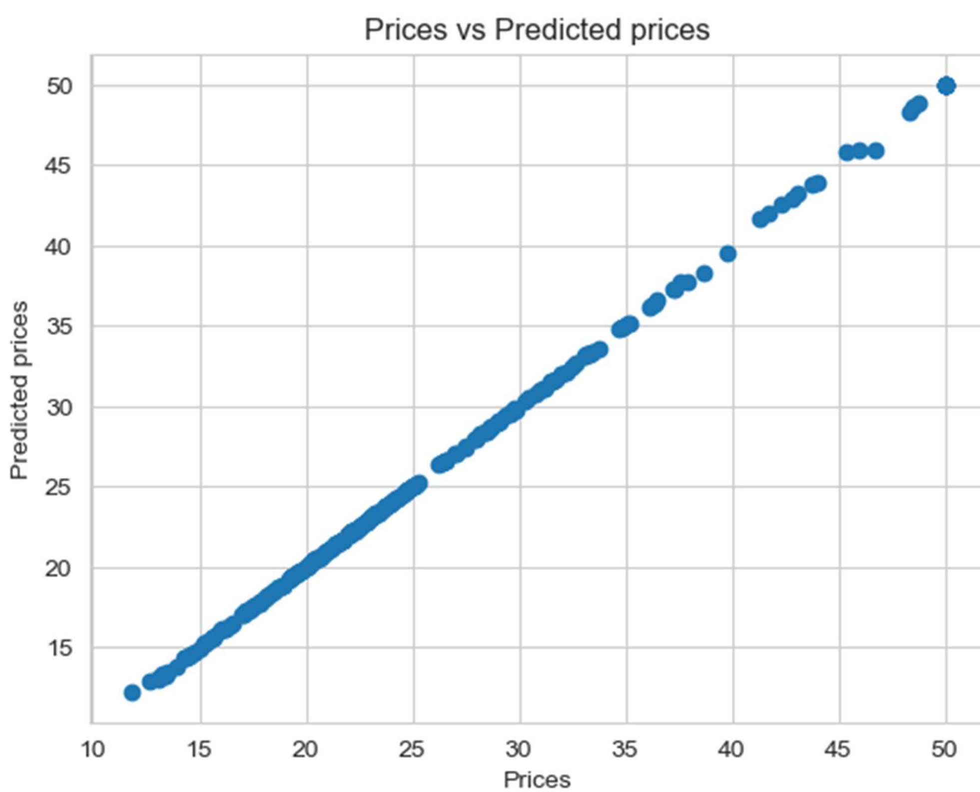
```
Training Accuracy: 99.99040288260335
```

```
print("Testing Accuracy:",reg.score(X_test, y_test) * 100)
```

```
Testing Accuracy: 99.9704666504295
```

```
# Visualizing the differences between Actual prices and Predicted values

plt.scatter(y_train, y_pred)
plt.xlabel("Prices")
plt.ylabel("Predicted prices")
plt.title("Prices vs Predicted prices")
plt.show()
```



Prediction and Final Score

Training Dataset

Linear Regression

Model Score: 87.3% Accuracy
Training Accuracy: 88.9% Accuracy
Testing Accuracy: 79.9% Accuracy

Random Forest Regressor

Training Accuracy: 99.9% Accuracy
Testing Accuracy: 99.96% Accuracy

Testing Dataset

Linear Regression

Model Score: 68.1% Accuracy
Training Accuracy: 69.4% Accuracy
Testing Accuracy: 55.0% Accuracy

Random Forest Regressor

Training Accuracy: 99.97% Accuracy
Testing Accuracy: 99.91% Accuracy

Training + Testing Dataset

Linear Regression

Model Score: 73.7% Accuracy
Training Accuracy: 77.3% Accuracy
Testing Accuracy: 58.9% Accuracy

Random Forest Regressor

Training Accuracy: 99.99% Accuracy
Testing Accuracy: 99.98% Accuracy

Conclusion

From the Exploratory Data Analysis, we could generate insight from the data. How each of the features relates to the target. Also, it can be seen from the evaluation of three models that Random Forest Regressor performed better than Linear Regression.