*Report*

*BOSTON HOUSE PRICE PREDICTION*

*Made By:*

*Rishabh Mathur*

Email: [Vigeagio@gmail.com](mailto:Vigeagio@gmail.com)

Phone No.: 9958814122

Content

* Overview
* Motivation
* Analysing the Problem Statement
* Dataset Description
* Algorithm Used
* Technologies and Libraries Used
* Data Collection
* Data Pre-processing
* Exploratory Data Analysis (EDA)
* Feature Observation
* Feature Selection
* Model Building
* Model Performances
* Prediction and Final Score
* Conclusion

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()

Text, letter

Description automatically generated

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

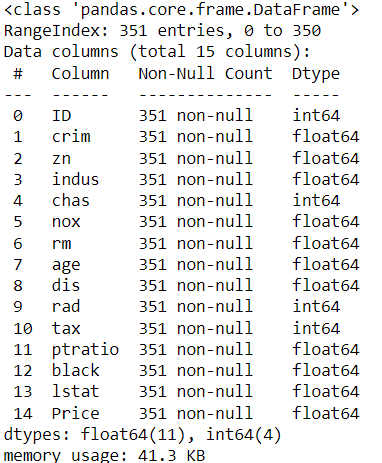
dtype: int64

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.

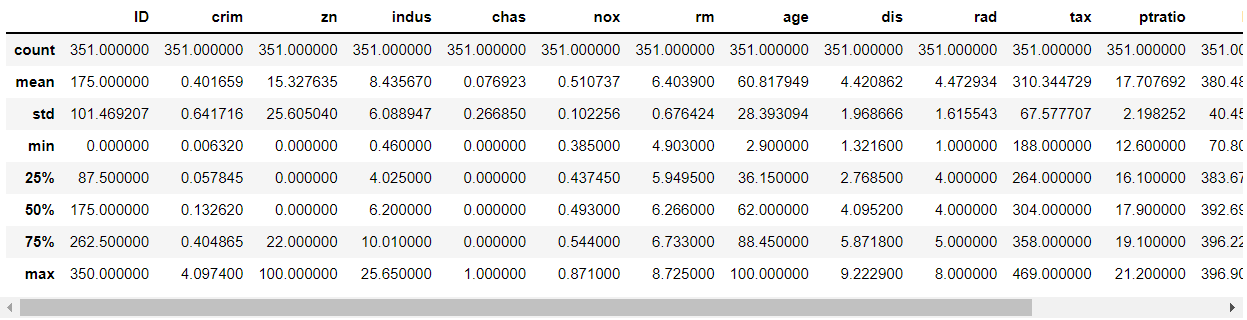
# Information about the dataset features

df.info()



# Describe

df.describe()



Feature Observation

# Finding out the correlation between the features

corr = df.corr()

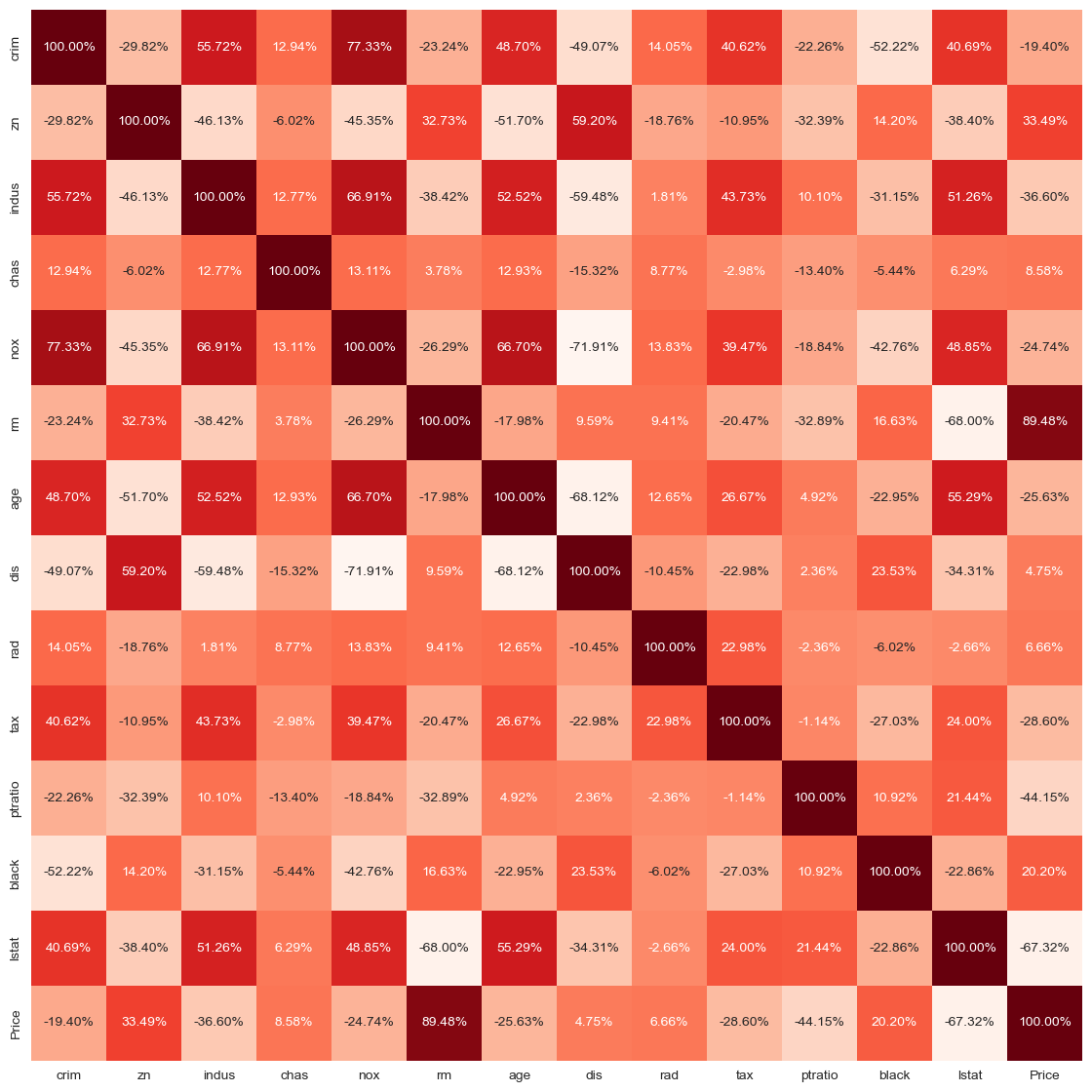
corr.shape

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’)

Chart, shape

Description automatically generated

Note: There are no null or missing values here.

sns.set\_style(‘whitegrid’)

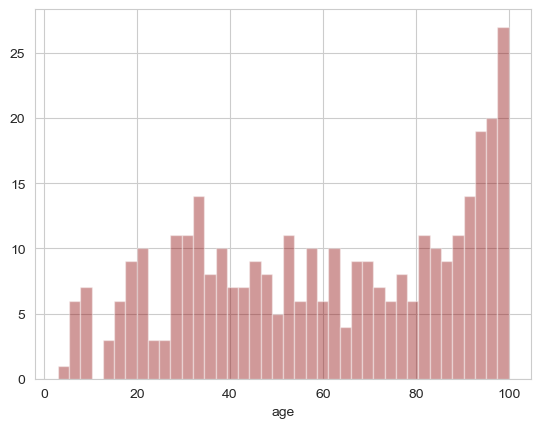
sns.countplot(x=’rad’,data=df)

Chart, bar chart

Description automatically generated

# 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)

Chart, bar chart

Description automatically generated

# Plotting a graph showing relation between Chas and Rad

sns.set\_style('whitegrid')

sns.countplot(x='chas',hue='rad',data=df,palette='RdBu\_r')

Chart, bar chart, histogram

Description automatically generated

# Plotting the graph for per capita crime rate by town

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

Chart, histogram

Description automatically generated

# Plotting the graph on Average number of rooms Distribution

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

Chart, histogram

Description automatically generated

# 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()

Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated Chart, scatter chart

Description automatically generated

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

Table

Description automatically generated

# Displaying the Best Five Features

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

Text

Description automatically generated

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()

Chart, scatter chart

Description automatically generated

# 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()

Chart, scatter chart

Description automatically generated

# 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()

Chart, line chart, histogram

Description automatically generated

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()

Chart, line chart

Description automatically generated

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.