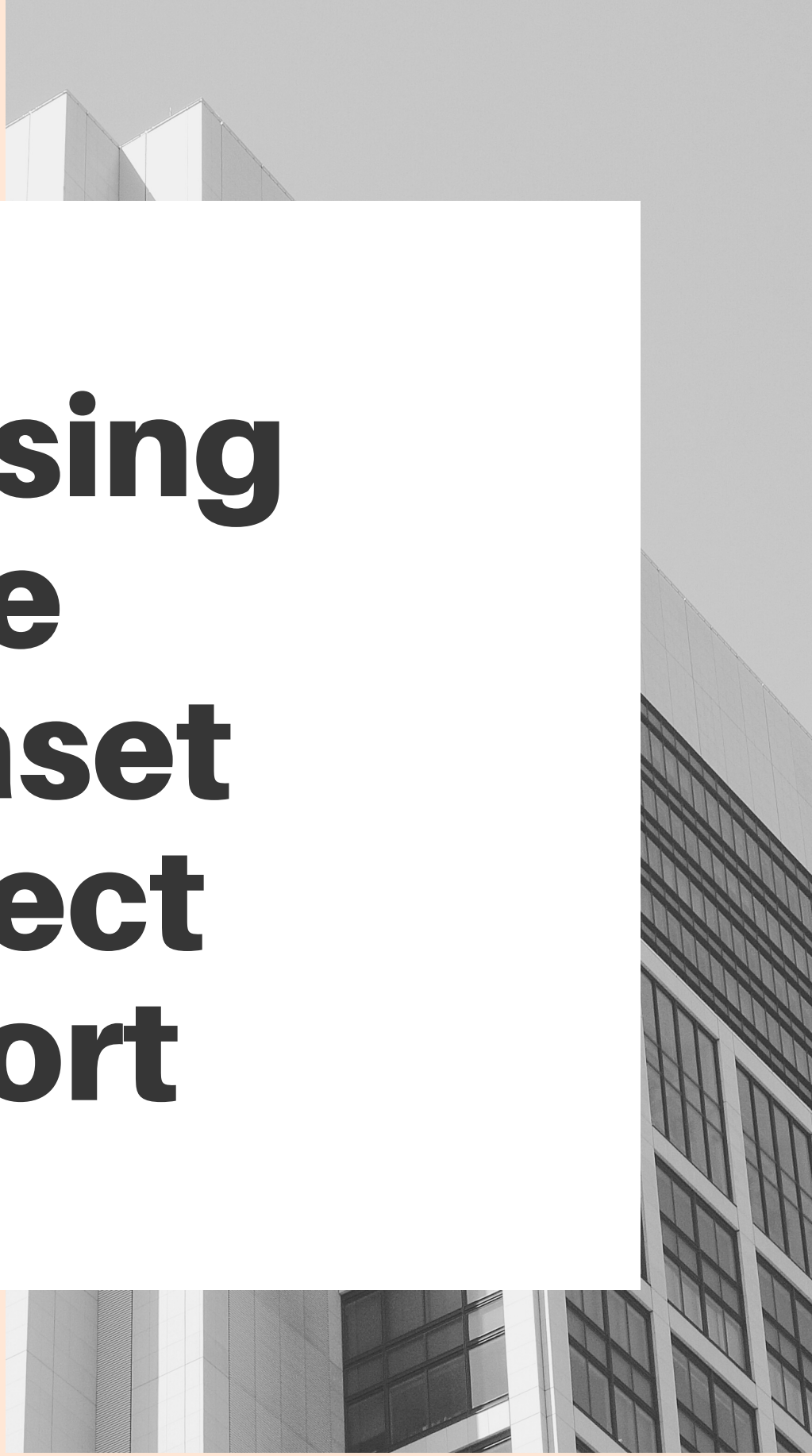
FEB 2020

Housing price dataset project Report



Prepared by Hansaj Kukreti
Approved by Shivam Baslas

About the project

Your neighbor is a real estate agent and wants some help predicting housing prices for regions in the USA.
It would be great if you could somehow create a model for him that allows him to put in a few features of
a house and returns back an estimate of what the house would sell for.

He has asked you if you could help him out with your new data science skills. You say yes,and now you have to decide that which
model will fit best on this dataset

Your neighbor then gives you some information about a bunch of houses in regions of the United States,
it is all in the data set: housing.csv.

Dataset

For this project we are using housing.csv .

tThe link for this csv file is - https://github.com/hansaj8/Data-science-projects/blob/master/HousingData.csv

Attribute Description

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

Objective

To predict the house price based on above features.

Let's get started

First, we will import the required libraries.

```
#import the required library
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Next, we will load the housing data

```
#now load the data . We will load housing data

df = pd.read_csv(r"C:\Users\dell\Downloads\Housingdata.csv")
```

UNDERSTANDING THE DATASET

first checking the head of the data

```
# first checking the head of the data
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	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	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	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	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	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

The prices of the house indicated by the variable MEDV is our target variable and the remaining are the feature variables based on which we will predict the value of a house.

DATASET INFORMATION

Now we will understand the data ie, how many rows and columns are there in the data by using info func

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
CRIM        486 non-null float64
ZN          486 non-null float64
INDUS       486 non-null float64
CHAS        486 non-null float64
NOX         506 non-null float64
RM          506 non-null float64
AGE         486 non-null float64
DIS         506 non-null float64
RAD         506 non-null int64
TAX         506 non-null int64
PTRATIO     506 non-null float64
B           506 non-null float64
LSTAT       486 non-null float64
MEDV        506 non-null float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

Now our first goal is to understand which columns are "Categorical" and which are "Continuous" .So we took a guess that if the unique values in a column is less than 100 , we will check the column .

```
# so here we made a function of this

def unique(data):
    for i in data.columns:
        if(data[i].nunique()<100):
            print (i,data[i].nunique())

unique(df)
```

```
ZN 26
INDUS 76
CHAS 2
NOX 81
RAD 9
TAX 66
PTRATIO 46
```

```
# from it we understand that CHAS and RAD  AND TAX
```

Finding the missing values

Now its time to check the missing values in the data frame

```
df.isnull().sum()
```

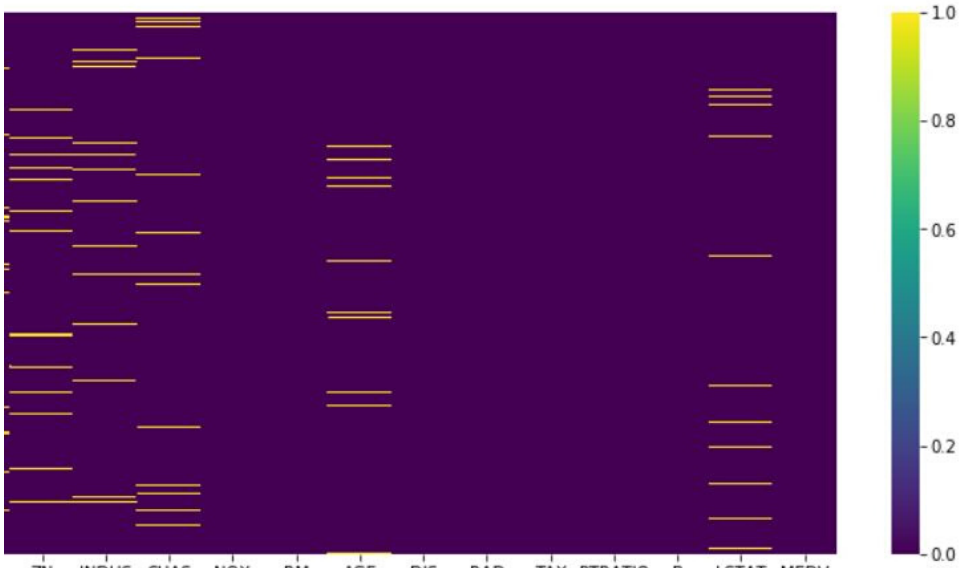
```
CRIM       20
ZN         20
INDUS      20
CHAS       20
NOX         0
RM          0
AGE        20
DIS         0
RAD         0
TAX         0
PTRATIO     0
B           0
LSTAT      20
MEDV        0
dtype: int64
```

here we got the numbers of missing values ie. their sum in each column.but to understand it more better we can use visualisation.So we are using heat map to understand missing values in a better way .

```
igure(figsize=(12,6))
eatmap(df.isnull() ,yticklabels = False,cmap = "viridis")
```

```
can clearly see the null values present in the columns of the dataset.Now its time to impute those null
```

```
lotlib.axes._subplots.AxesSubplot at 0x1769b4edbe0>
```



Now its time to make a dataset a provide missing value percentage in that dataset

```
l=[]
for i in df.columns:
    l.append(df[i].isnull().sum()/len(df)*100)
pd.DataFrame(data = {"Columns": ['CRIM', 'ZN', 'INDUS', 'CH
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],"Missing value perc
```

	Columns	Missing value percentage
0	CRIM	3.952569
1	ZN	3.952569
2	INDUS	3.952569
3	CHAS	3.952569
4	NOX	0.000000
5	RM	0.000000

from df1 we saw that in each column there is only 4 percent missing value present . 4 % is not a big percentage as far as missing value is considered . so we will simply drop those missing values from the dataset

```
df1=df.dropna() #to permanently ammend the data we can use inplace
```

```
df1.isna().sum() # here we can see that no missing value is present
```

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
```

Descriptive analysis

```
df1.describe(include = "all")
# here we used include = all. but its not showing any "top
```

	CRIM	ZN	INDUS	CHAS	NOX	
count	394.000000	394.000000	394.000000	394.000000	394.000000	394
mean	3.690136	11.460660	11.000863	0.068528	0.553215	6
std	9.202423	23.954082	6.908364	0.252971	0.113112	0
min	0.006320	0.000000	0.460000	0.000000	0.389000	3
25%	0.081955	0.000000	5.130000	0.000000	0.453000	5
50%	0.268880	0.000000	8.560000	0.000000	0.538000	6
75%	3.435973	12.500000	18.100000	0.000000	0.624000	6
max	88.976200	100.000000	27.740000	1.000000	0.871000	8

now its time to understand this statistical table.

Here we can see the first row ie. count of each variable is same ie 394 .it means there is no null value present in the data set which make sense because we just dropped all the missing values from the data set .

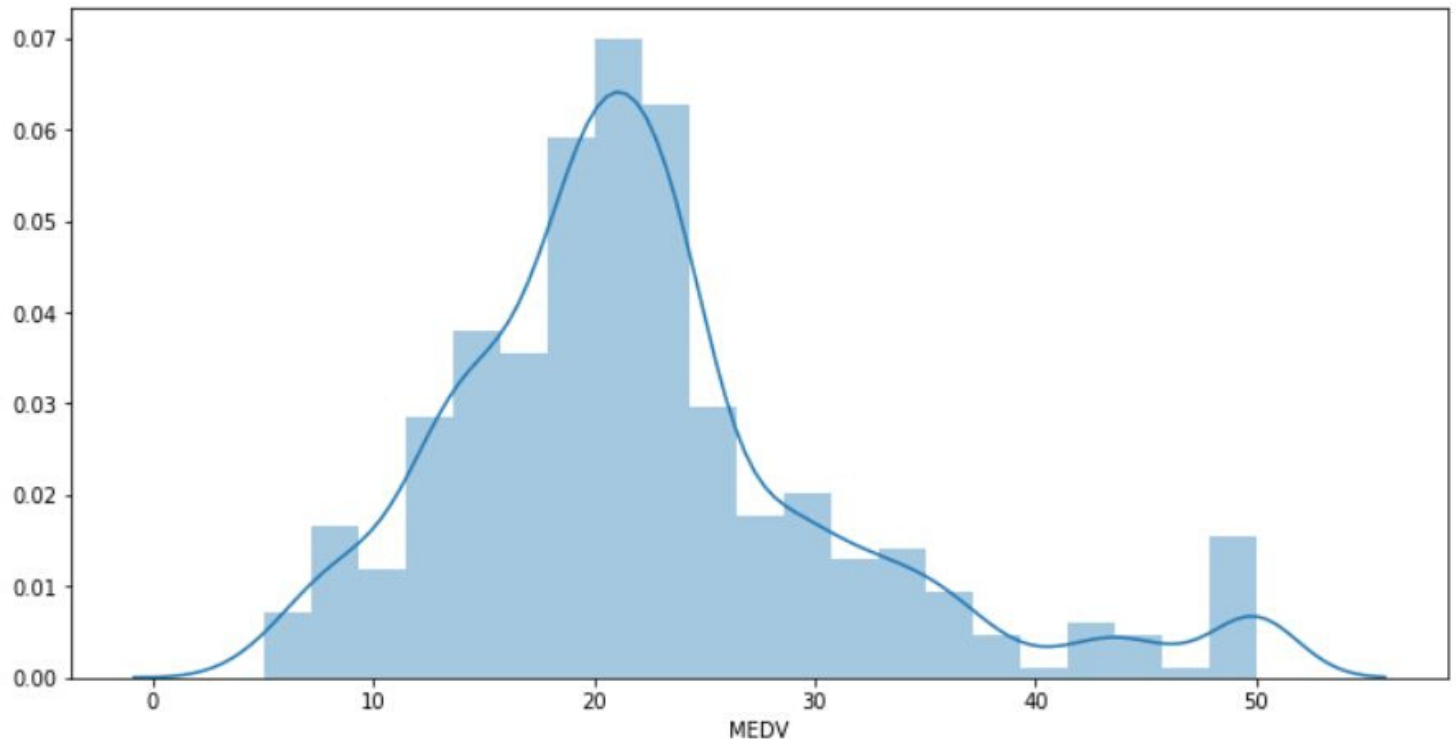
Exploratory Data Analysis

Exploratory Data Analysis is a very important step before training the model. In this section, we will use some visualizations to understand the relationship of the target variable with other features.

Let's first plot the distribution of the target variable MEDV. We will use the distplot function from the seaborn library.

```
plt.figure(figsize=(12,6))
sns.distplot(df1["MEDV"])
#for optimal results we would be looking for normal distribuition and here it is.
```

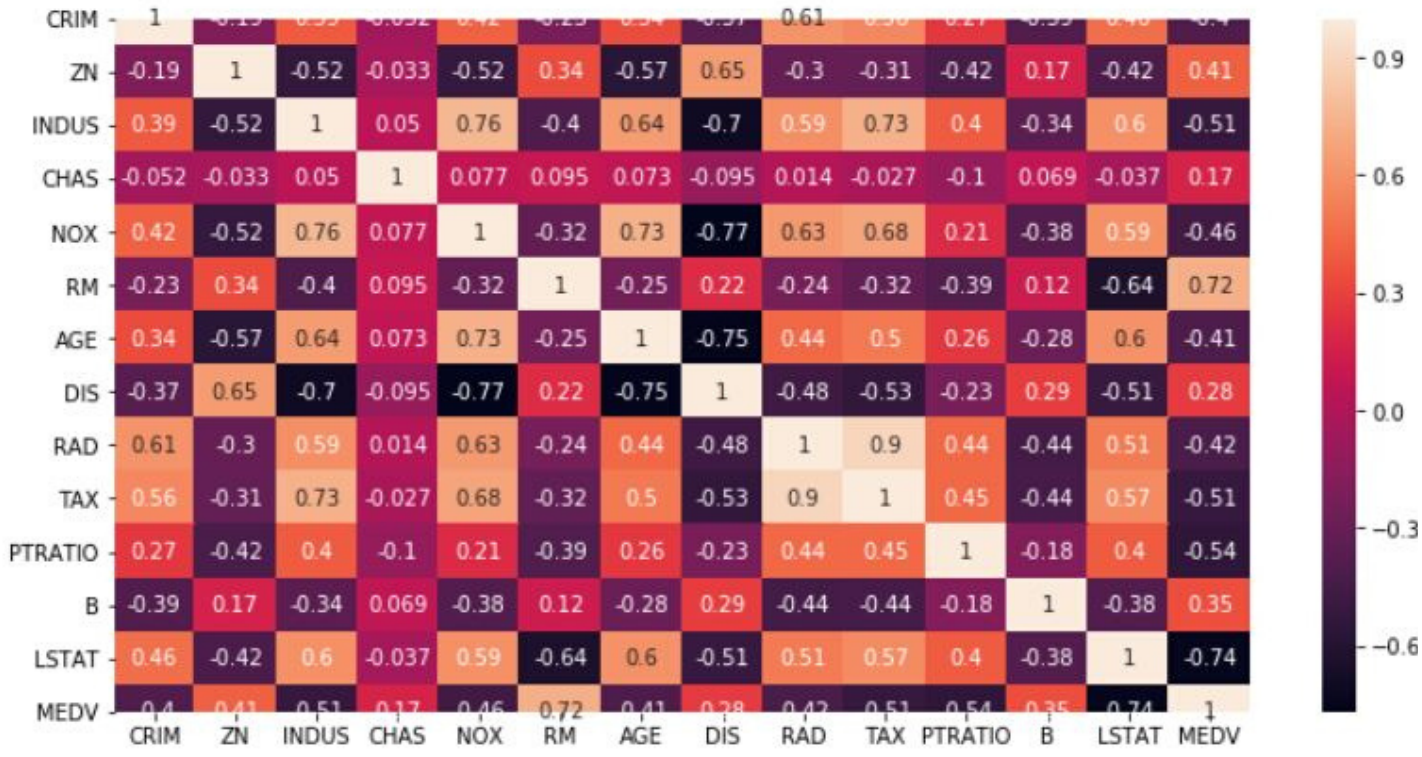
```
<matplotlib.axes._subplots.AxesSubplot at 0x22ec250f438>
```



We see that the values of MEDV are distributed normally with few outliers.Next, we create a correlation matrix that measures the linear relationships between the variables. The correlation matrix can be formed by using the corr function from the pandas dataframe library. We will use the heatmap function from the seaborn library to plot the correlation matrix.

```
plt.figure(figsize = (12,6))
sns.heatmap(df1.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x22ec2fc6cf8>
```



The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation.Observations:

Observation

By looking at the correlation matrix we can see that RM has a strong positive correlation with MEDV (0.72) where as LSTAT has a high negative correlation with MEDV(-0.74).

Multi-co-linearity

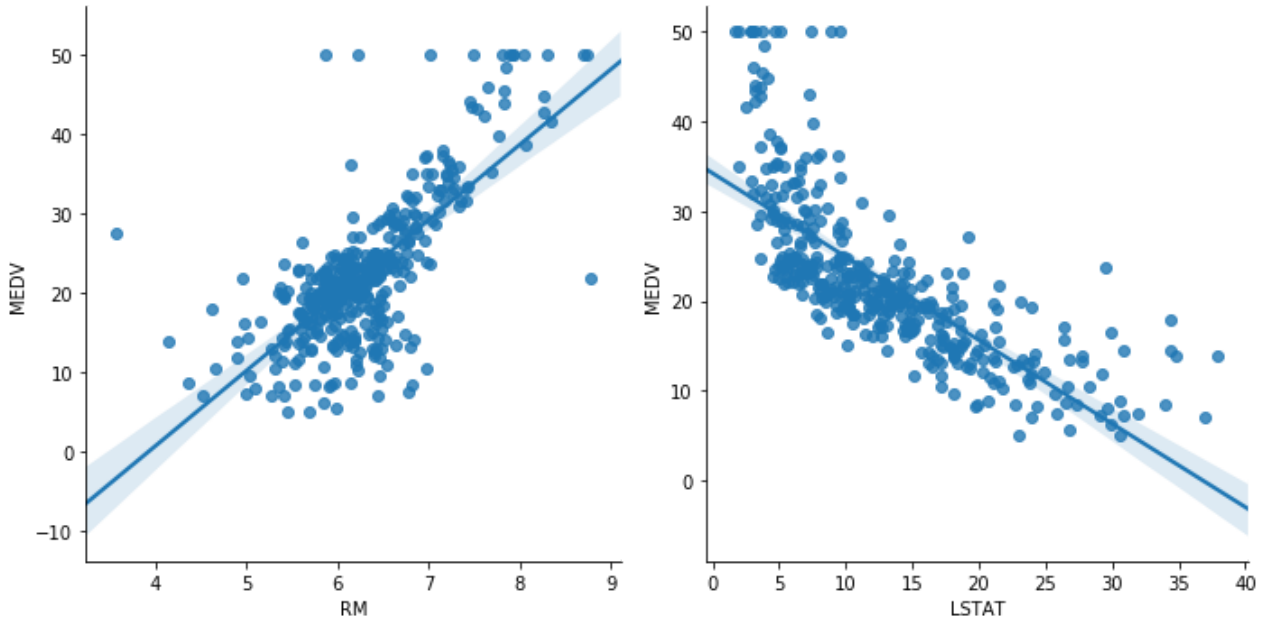
The features RAD, TAX have a correlation of 0.91. These feature pairs are strongly correlated to each other. We should not select both these features together for training the model.
Same goes for the features DIS and AGE which have a correlation of -0.75.

Explanation

Consider the simplest case where YY is regressed against XX and ZZ and where XX and ZZ are highly positively correlated. Then the effect of XX on YY is hard to distinguish from the effect of ZZ on YY because any increase in XX tends to be associated with an increase in ZZ.

Based on the above observations we will use RM and LSTAT as our features. Using a scatter plot or lmplot let's see how these features vary with MEDV.

```
sns.lmplot(x="RM",y="MEDV",data=df1)
sns.lmplot(x="LSTAT",y="MEDV",data=df1)
```



Observation

The prices increase as the value of RM increases linearly. There are few outliers and the data seems to be capped at 50.The prices tend to decrease with an increase in LSTAT. Though it doesn't look to be following exactly a linear line.

Preparing the data by selecting important features

first create a separate dataframe of feature varibles and a target varible

```
X = df1[["RM",,"LSTAT"]]
y = df1["MEDV"]
```

Splitting the data into training and testing sets

Next, we split the data into training and testing sets. We train the model with 80% of the samples and test with the remaining 20%. We do this to assess the model's performance on unseen data. To split the data we use train_test_split function provided by scikit-learn library. We finally print the sizes of our training and test set to verify if the splitting has occurred properly.

```
# import the required library
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(  X, y, test_size=0.2, random_s
```

Training and testing the model

We have completely split our data set. Now its time to fit linear regression on the training data.
we can do this by traditional method ie. obtaining the cost func and then minimising that cost func but here we will use sk learn library

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Model evaluation

We will evaluate our model using RMSE and R2-score.

```
rmse=np.sqrt(metrics.mean_squared_error(y_test,predictions))
r2=metrics.r2_score(y_test,predictions)

print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
print("\n")
# model evaluation for testing set

rmse = (np.sqrt(metrics.mean_squared_error(y_test, predictions
r2 = metrics.r2_score(y_test, predictions)

print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for training set
--------------------------------------
RMSE is 6.143273034007016
R2 score is 0.5525619081372948


The model performance for testing set
--------------------------------------
RMSE is 6.143273034007016
```

now make the function of the model so that we can ammend the features as per our requirements.

```
# now make the func to run the model its r score ,rmse
def lm():
    X = df1[['RM', "LSTAT"]]
    y = df1["MEDV"]
    X_train, X_test, y_train, y_test = train_test_split(  X, y, test_size=0.2, random_s
    lm = LinearRegression()
    lm.fit(X_train,y_train)
    predictions = lm.predict(X_test)
    rmse = (np.sqrt(metrics.mean_squared_error(y_test, predictions)))
    r2 = metrics.r2_score(y_test, predictions)
    print("The model performance for training set")
    print("--------------------------------------")
    print('RMSE is {}'.format(rmse))
    print('R2 score is {}'.format(r2))
    print("\n")
# model evaluation for testing set

    rmse = (np.sqrt(metrics.mean_squared_error(y_test, predictions)))
    r2 = metrics.r2_score(y_test, predictions)

    print("The model performance for testing set")
    print("--------------------------------------")
    print('RMSE is {}'.format(rmse))
    print('R2 score is {}'.format(r2))

#now it will be easier for us to change the feature variable and quickly test the resul
```

```
lm()
```

```
The model performance for training set
--------------------------------------
RMSE is 6.143273034007016
R2 score is 0.5525619081372948


The model performance for testing set
--------------------------------------
RMSE is 6.143273034007016
R2 score is 0.5525619081372948
```

As we can see that accuracy is not that good . so lets choose a different model and check the accuracy.

Decision Tree Regressor

```
from sklearn.tree import DecisionTreeRegressor

d= DecisionTreeRegressor(max_depth=11)
d.fit(X_train,y_train)
d_pred = d.predict(X_train)
```

```
def dt():


    X = df1[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX',
       'PTRATIO', 'B', 'LSTAT']]
    y = df1["MEDV"]
    d= DecisionTreeRegressor(max_depth=10)
    d.fit(X_train,y_train)
    d_pred = d.predict(X_train)
    rmse = np.sqrt(metrics.mean_squared_error(y_train,d_pred))
    r2 = metrics.r2_score(y_train,d_pred)
    print("The model performance for training set")
    print("RMSE is", rmse)
    print("r2_score", r2)
    dt= d.predict(X_test)
    rmse = np.sqrt(metrics.mean_squared_error(y_test,dt))
    r2 = metrics.r2_score(y_test,dt)
    print("The model performance for testing set")
    print("RMSE is", rmse)
    print("r2_score", r2)
```

```
dt()
```

```
The model performance for training set
RMSE is 0.6991160960074149
r2_score 0.9940734971822461
The model performance for testing set
RMSE is 4.8197253768685835
r2_score 0.7245911413681386
```

It feels like decision tree is a better model for this dataset .
lets check the random forest also.

Random forest Regressor

```
from sklearn.ensemble import RandomForestRegressor
```

```
df.columns
```

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

```
def rf():
    X = df1[['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TA
       'PTRATIO', 'B', 'LSTAT']]
    y = df1["MEDV"]
    r= RandomForestRegressor()
    r.fit(X_train,y_train)
    pred = r.predict(X_train)
    rmse = np.sqrt(metrics.mean_squared_error(y_train,pred))
    r2 = metrics.r2_score(y_train,pred)
    print("The model performance for training set")
    print("RMSE is", rmse)
    print("r2_score", r2)
    p = r.predict(X_test)
    rmse = np.sqrt(metrics.mean_squared_error(y_test,p))
    r2 = metrics.r2_score(y_test,p)
    print("The model performance for testing set")
    print("RMSE is", rmse)
    print("r2_score", r2)
```

```
rf()
```

```
The model performance for training set
RMSE is 1.5897163568505217
r2_score 0.9693564427782331
The model performance for testing set
RMSE is 5.476845044874797
r2_score 0.6443734093552573
```

Conclusion

In this story, we applied the concepts of linear regression , Decision Tree and Random forest on the housing dataset. We observed that decision tree regressor is the best suited model for this problem.