BOSTON HOUSE PRICE PREDICTION

INTRODUCTION

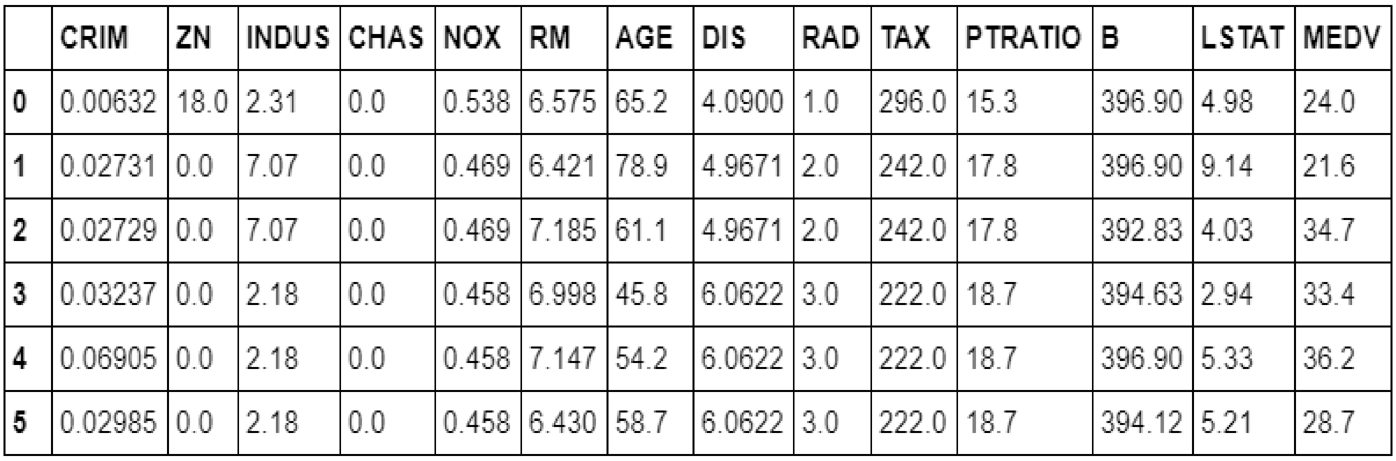
Machine learning is a subfield of Artificial Intelligence (AI) that works with algorithms and technologies to extract useful information from data. Machine learning methods are appropriate in big data since attempting to manually process vast volumes of data would be impossible without the support of machines. Machine learning in computer science attempts to solve problems algorithmically rather than purely mathematically. Therefore, it is based on creating algorithms that permit the machine to learn.

In this project, we will develop and evaluate the performance and the predictive power of a model trained and tested on data collected from houses in Boston's suburbs. Once we get a good fit, we will use this model to predict the monetary value of a house located at the Boston's area. A model like this would be very valuable for a real state agent who could make use of the information provided in a daily basis.

GETTING THE DATA

The dataset used in this project comes from information collected by the U.S. Census Service concerning housing in the area of Boston MA.

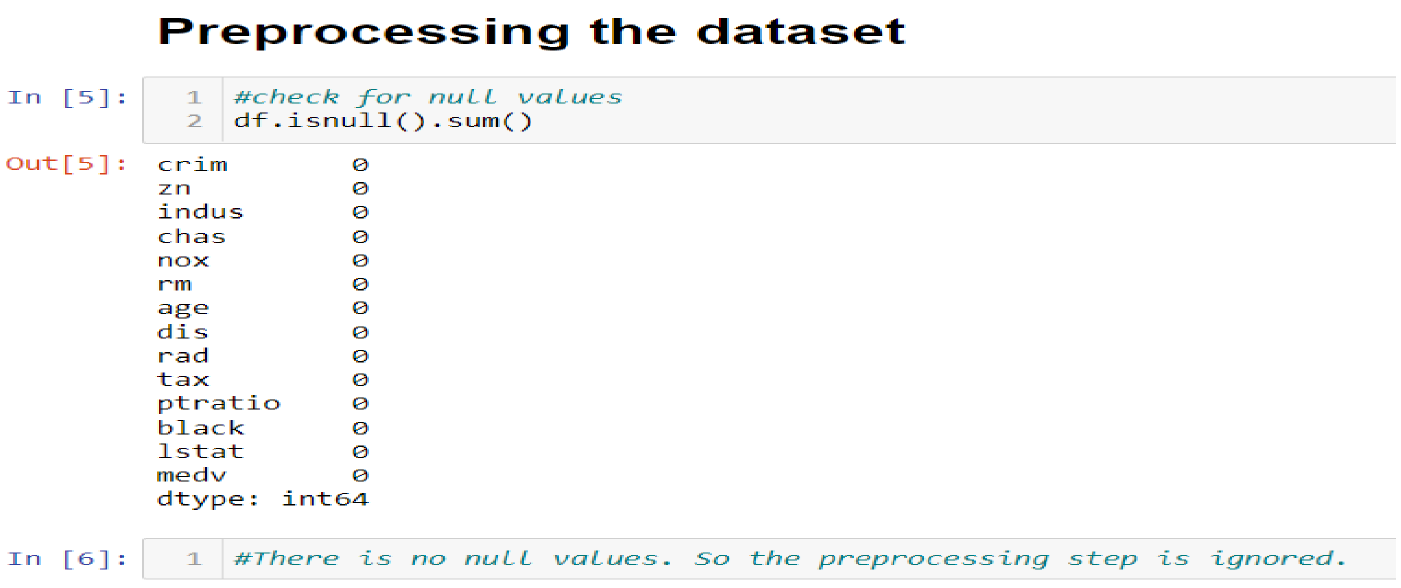
This is an overview of the original dataset, with its original features:



	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	1.0	296.0	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	2.0	242.0	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	2.0	242.0	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	3.0	222.0	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	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7

PREPROCESSING THE DATASET

Here first we have to check whether our dataset contains any null values or not. If there are any null values, then that will be filled with the average values.



Preprocessing the dataset

```
In [5]:  #check for null values
         df.isnull().sum()
Out[5]:  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

In [6]:  #There is no null values. So the preprocessing step is ignored.
```

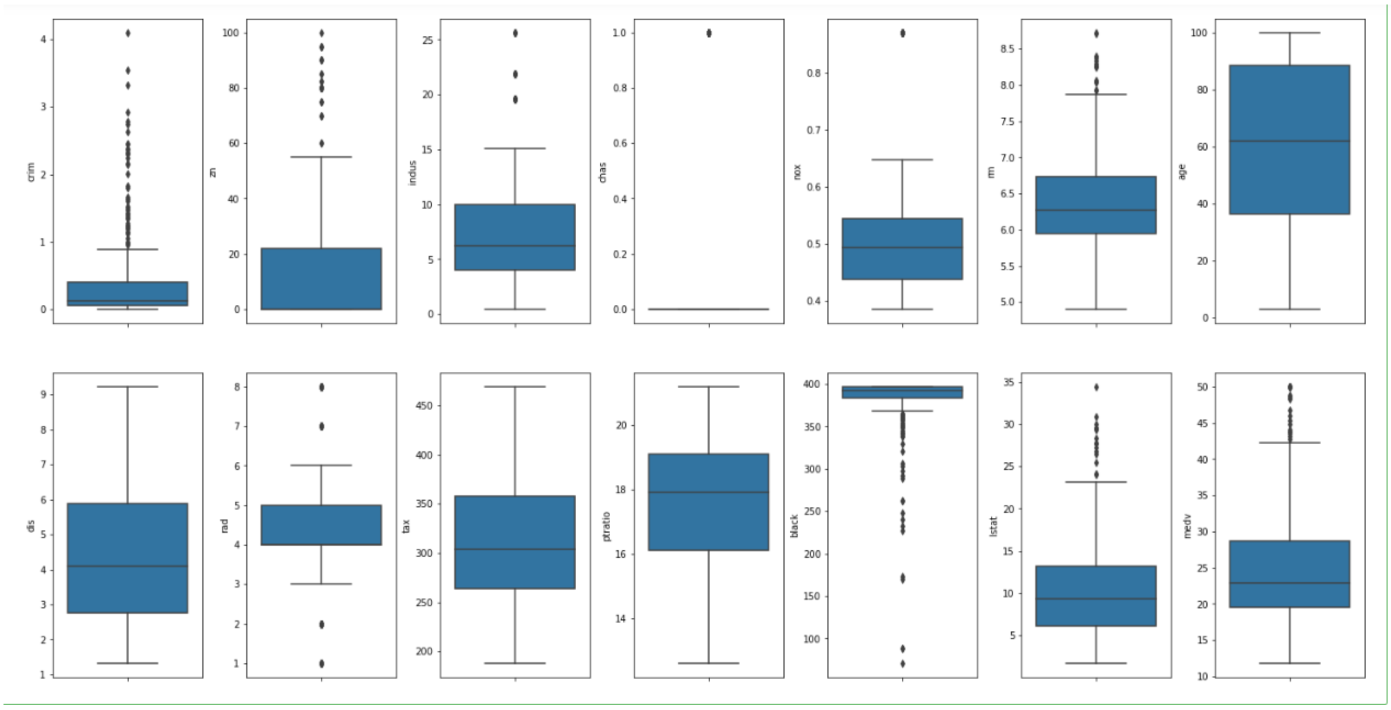
As our dataset don't contain any null values, so we will ignore this step. And go to the next step.

EXPLORATORY DATA ANALYSIS

Box Plot and Disc Plot

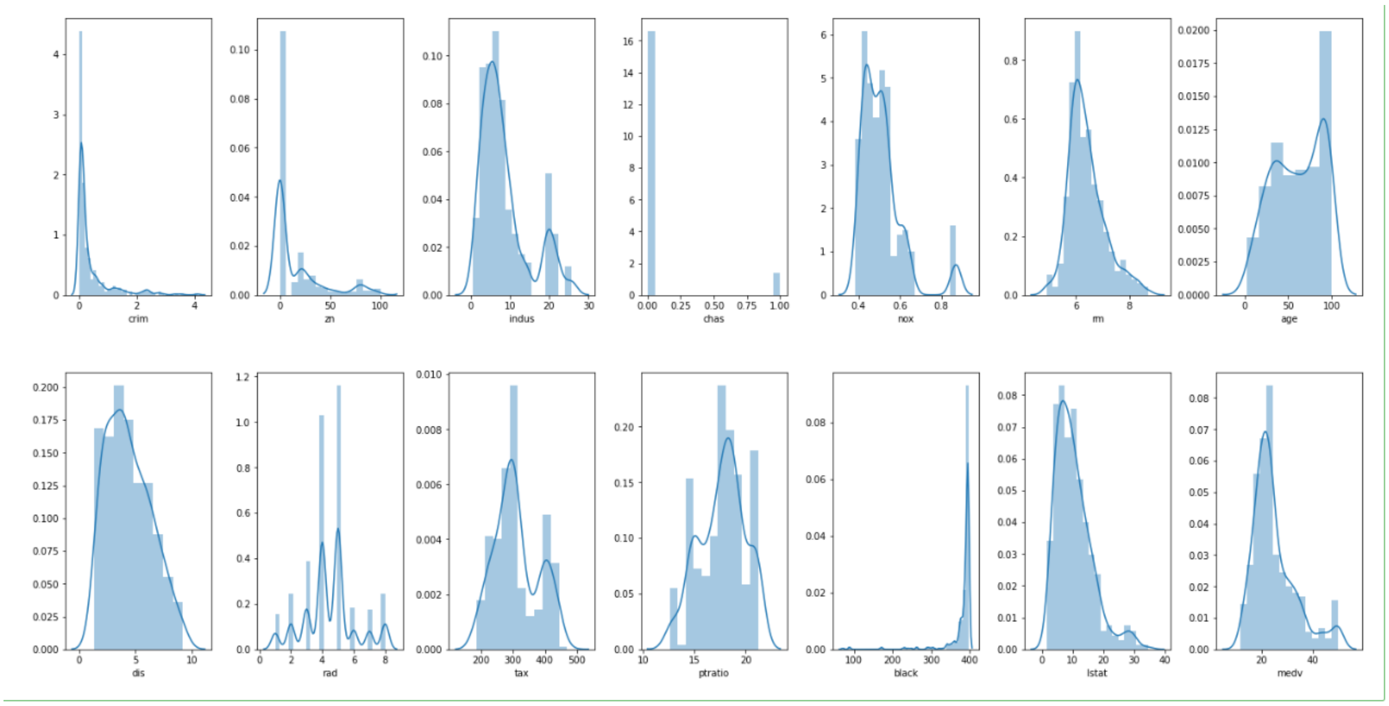
We will start by creating a boxplot matrix that will allow us to visualize the pair-wise relationships and correlations between the different features. It is also quite useful to have a quick overview of how the data is distributed and whether it contains outliers or not. Here instead of creating boxplots for each one, box plots were created in a single loop. For that subplot are created.

```
fig,ax=plt.subplots(ncols=7, nrows=2, figsize=(20,10))
index=0
ax=ax.flatten()
for col,value in df.items():
    sns.boxplot(y=col, data=df, ax=ax[index])
    index+=1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



After creating the subplot, we observed that there are many overlapping, so hyper paramated tuning was done to display the graph properly.

```
fig,ax=plt.subplots(ncols=7, nrows=2, figsize=(20,10))
index=0
ax=ax.flatten()
for col,value in df.items():
    sns.distplot(value, ax=ax[index])
    index+=1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



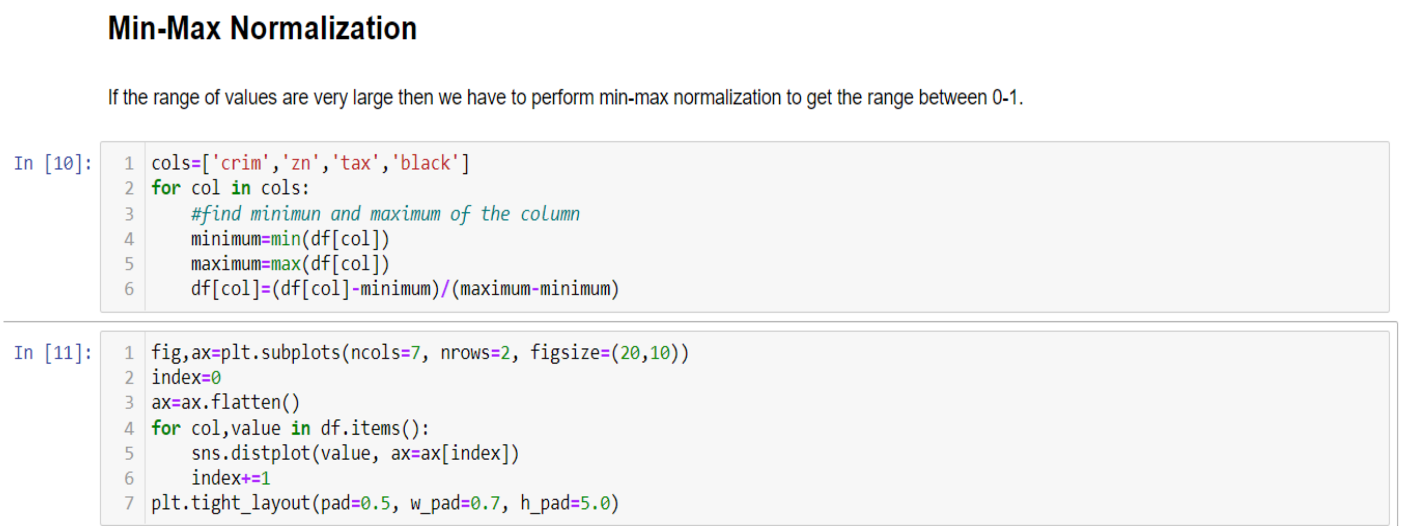
MIN-MAX NORMALIZATION

If the range of values are very large then we have to perform min-max normalization to get the range between 0-1. It is one of the most common ways to normalize data. For every feature, the minimum value of that feature gets transformed into 0, the maximum value gets transformed into 1 and every other value gets transformed into decimal between 0 and 1. The main idea behind normalization is variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias. Thus, to deal with this potential problem feature-wise normalization such as Min-Max Scaling is usually used prior to model fitting.

Here in our dataset columns like crim, zn, tax and black had value in very higher range, so we applied min-max normalization here and brought their values between range 0 and 1.

Formula for this:

$$X=\frac{X-Minimun}{Maximum-Minimum}$$

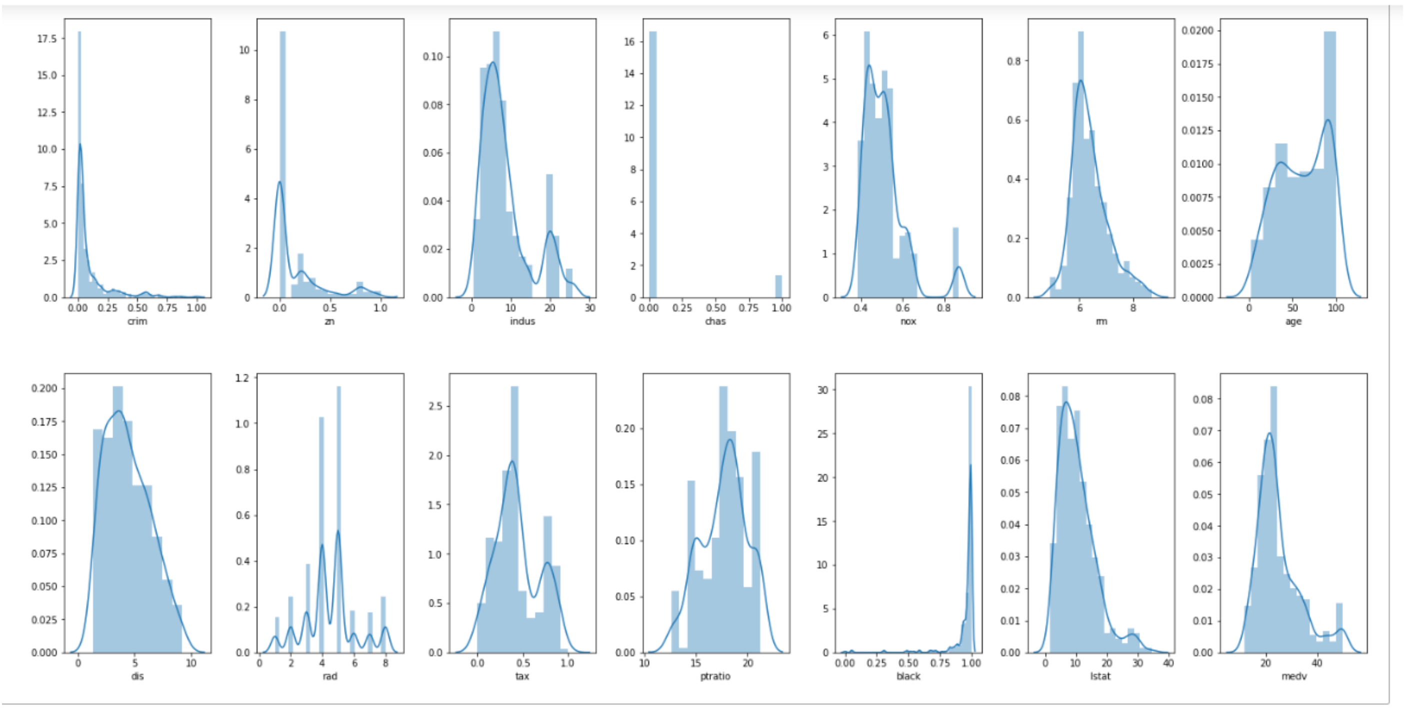


Min-Max Normalization

If the range of values are very large then we have to perform min-max normalization to get the range between 0-1.

In [10]:
```python
cols=['crim','zn','tax','black']
for col in cols:
    #find minimun and maximum of the column
    minimum=min(df[col])
    maximum=max(df[col])
    df[col]=(df[col]-minimum)/(maximum-minimum)
```

In [11]:
```python
fig,ax=plt.subplots(ncols=7, nrows=2, figsize=(20,10))
index=0
ax=ax.flatten()
for col,value in df.items():
    sns.distplot(value, ax=ax[index])
    index+=1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



We can see after min-max normalization, all the values where min-max normalization was performed are in the range of 0-1. If the values have large difference means we have to use min-max normalization.

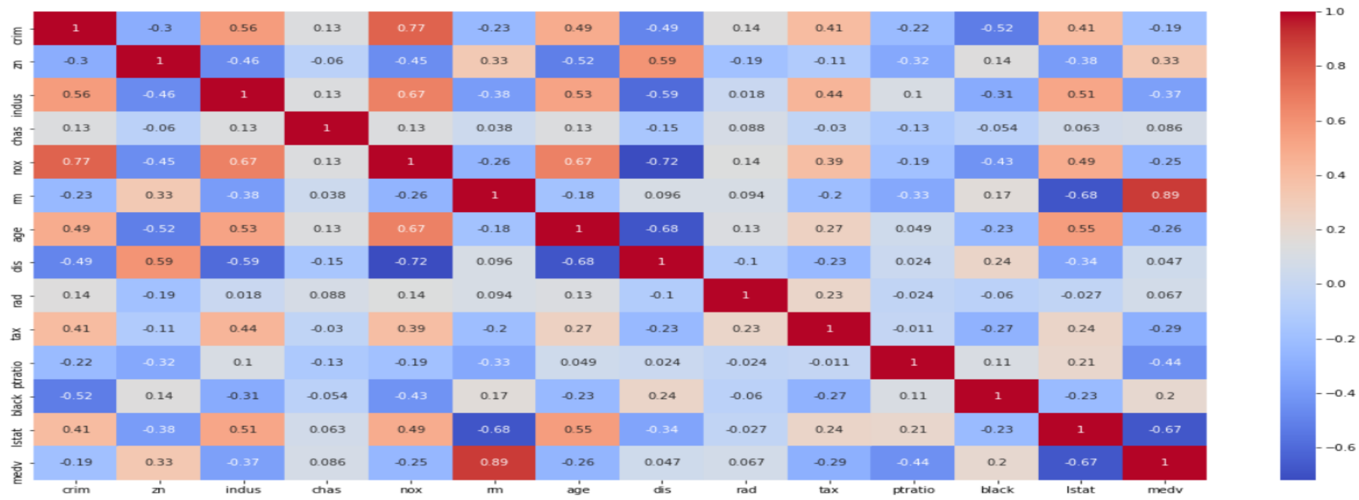
COORELATION MATRIX

A correlation matrix is simply a table which displays the correlation coefficients for different variables. The matrix depicts the correlation between all the possible pairs of values in a table. It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data. We can create a correlation matrix to quantify and summarize the relationships between the variables.

Coorelation Matrix

In [14]:
```
corr=df.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x213d5cf5eb0>
```



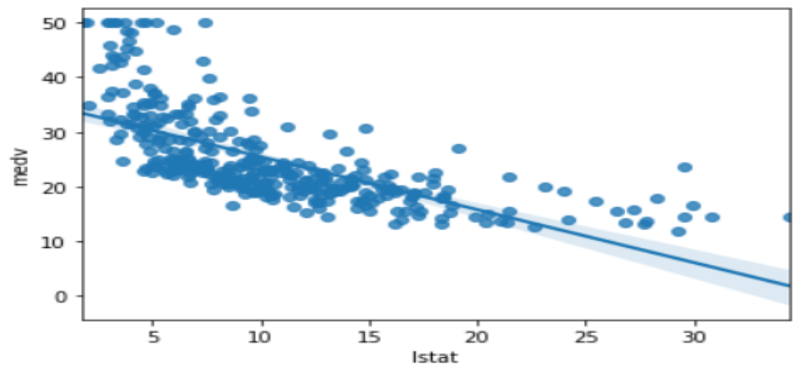
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
crim	1	-0.3	0.56	0.13	0.77	-0.23	0.49	-0.49	0.14	0.41	-0.22	-0.52	0.41	-0.19
zn	-0.3	1	-0.46	-0.06	-0.45	0.33	-0.52	0.59	-0.19	-0.11	-0.32	0.14	-0.38	0.33
indus	0.56	-0.46	1	0.13	0.67	-0.38	0.53	-0.59	0.018	0.44	0.1	-0.31	0.51	-0.37
chas	0.13	-0.06	0.13	1	0.13	0.038	0.13	-0.15	0.088	-0.03	-0.13	-0.054	0.063	0.086
nox	0.77	-0.45	0.67	0.13	1	-0.26	0.67	-0.72	0.14	0.39	-0.19	-0.43	0.49	-0.25
rm	-0.23	0.33	-0.38	0.038	-0.26	1	-0.18	0.096	0.094	-0.2	-0.33	0.17	-0.68	0.89
age	0.49	-0.52	0.53	0.13	0.67	-0.18	1	-0.68	0.13	0.27	0.049	-0.23	0.55	-0.26
dis	-0.49	0.59	-0.59	-0.15	-0.72	0.096	-0.68	1	-0.1	-0.23	0.024	0.24	-0.34	0.047
rad	0.14	-0.19	0.018	0.088	0.14	0.094	0.13	-0.1	1	0.23	-0.024	-0.06	-0.027	0.067
tax	0.41	-0.11	0.44	-0.03	0.39	-0.2	0.27	-0.23	0.23	1	-0.011	-0.27	0.24	-0.29
ptratio	-0.22	-0.32	0.1	-0.13	-0.19	-0.33	0.049	0.024	-0.024	-0.011	1	0.11	0.21	-0.44
black	-0.52	0.14	-0.31	-0.054	-0.43	0.17	-0.23	0.24	-0.06	-0.27	0.11	1	-0.23	0.2
lstat	0.41	-0.38	0.51	0.063	0.49	-0.68	0.55	-0.34	-0.027	0.24	0.21	-0.23	1	-0.67
medv	-0.19	0.33	-0.37	0.086	-0.25	0.89	-0.26	0.047	0.067	-0.29	-0.44	0.2	-0.67	1

From the coorelation matrix we can see lstat is highly negatively coorelated and rm is highly positively coorelated.

Now we will display rm and lstat how ther are coorelating to target variable.

In [15]:
```
sns.regplot(y=df['medv'], x=df['lstat'])
```

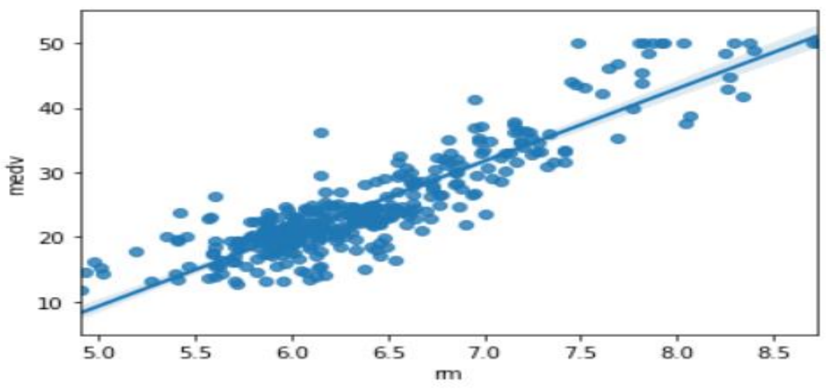
Out[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x213d4c43880>
```



In [16]:
```
# Here price is decreasing with increase of lstat
```

Here we can see price is decreasing with lstat increasing, so it is negatively coorelated.

```
In [17]:  sns.regplot(y=df['medv'], x=df['rm'])
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x213d5002ee0>
```



```
In [18]:  #Here price of the house increases with increase in rm
```

Here we can see price is increasing with rm increasing, so it is positively coorelated.

SPLITING THE DATA

For this section we will take the Boston housing dataset and split the data into training and testing subsets. Typically, the data is also shuffled into a random order when creating the training and testing subsets to remove any bias in the ordering of the dataset.

Model Training

After splitting the dataset randomly, training was done on it. And some basic models were tested on it to check the success rate of it. I used

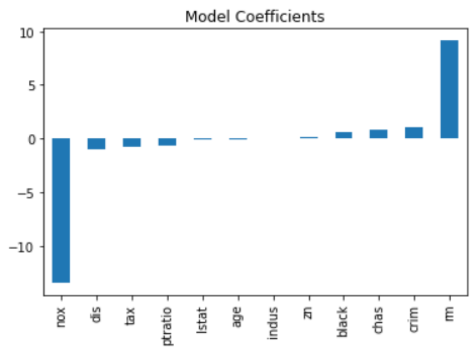
- Linear Regressor
- Decision tree Regresson
- Random Forest Regressor

Linear Regressor

In [21]:
```python
from sklearn.linear_model import LinearRegression
model=LinearRegression(normalize=True)
train(model,X,Y)
coef=pd.Series(model.coef_,X.columns).sort_values()
coef.plot(kind='bar',title='Model Coefficients')
```

```
Model Report
MSE: 11.52011269819824
CV Score: 13.001589421597137
```

Out[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x213d6434b20>
```



For training with splitted dataset we have MSE of 11 and CV Score of 13. As we can see for Linear Regression rm has high positive coefficient and nox has high negative coefficient

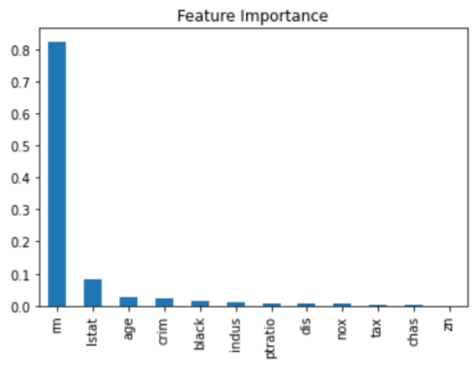
We can see that for training with splitted dataset we have MSE of 11 and CV Score of 13. In Linear Regressor rm has high positive coefficient and nox has high negative coefficient.

Decision Tree Regressor

In [22]:
```python
from sklearn.tree import DecisionTreeRegressor
model=DecisionTreeRegressor()
train(model,X,Y)
coef=pd.Series(model.feature_importances_, X.columns).sort_values(ascending=False)
coef.plot(kind='bar',title='Feature Importance')
```

```
Model Report
MSE: 19.392045454545453
CV Score: 17.552902213279676
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x213d57167f0>
```



Now we have MSE as 17 and CV Score as 15. It has CV Score higher than Linear Regression and for estimation we need to prefer CV Score only. So this is not the best model

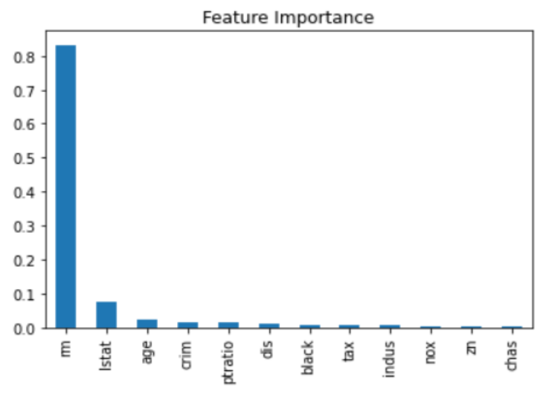
Now we have MSE as 17 and CV Score as 15. It has CV Score higher than Linear Regression and for estimation, we need to prefer CV Score only. So, this is not the best model.

Random Forest Regressor

```
In [23]:
from sklearn.ensemble import RandomForestRegressor
model=RandomForestRegressor()
train(model,X,Y)
coef=pd.Series(model.feature_importances_, X.columns).sort_values(ascending=False)
coef.plot(kind='bar',title='Feature Importance')
```

```
Model Report
MSE: 11.731553897727276
CV Score: 10.411582285191153
```

```
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x213d4995af0>
```



Here we get MSE as 11 and CV Score as 10. Here CV Score is lowest so we got the best model among three i.e. Random Forest Regressor.

Here we get MSE as 11 and CV Score as 10. Here CV Score is lowest so we got the best model among three i.e., Random Forest Regressor.

Piyush Ranjan Srichandan

Mob-7077191016

Silicon Institute of Technology, Bhubaneswar