MINI PROJECT REPORT
BOSTON HOUSE PRICE PREDICTION

Name: Avinash R C

- **Objective:**
  1. The objective of this project is to create a ML model for predicting the cost of the houses in Boston, US by using the given Dataset.

- **Tools used:**
  1. **Platform:**
     - Jupyter Notebook
  2. **Language used:**
     - Python
  3. **Libraries:**
     - Scikit-learn
     - Matplotlib
     - Seaborn
     - Pandas
     - NumPy

- **Dataset used:**
  1. Boston_Test.csv
  2. Boston_Train.csv
  3. Combined.csv

- **Steps involved:**

1. Importing Libraries
2. Uploading the given dataset
3. Data Pre-processing
4. Exploratory Data Analysis
5. Min-Max Normalization
6. Correlation Matrix
7. Splitting the dataset into Train and Test data
8. Training the Model
9. Using diff. Regression techniques
10. Conclusion

- **Explanation of the Steps used:**

1. Importing the Libraries

- This is the first step of any ML problems.
- Importing the necessary python libraries for imparting EDA, pre-processing techniques, regression etc., to develop a ML model

Importing Libraries

In [3]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn import linear_model
```

2. Uploading the given Dataset

- Here, we just upload the dataset that we are going to use for the further process.

Uploading the Dataset (combined)

In [4]:
```
cb=pd.read_csv("C:/Users/rcavi/Desktop/mini project/combined.csv")
cb
```

Out[4]:

	Unnamed: 0	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	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	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	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	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	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

506 rows × 15 columns

3. Data pre-processing

- It is an important step before going further into the project.
- It is a data mining technique which is used to transform a raw data in a useful and efficient format.
- Here, we checked if any attributes of the given data set has null value (or) any other useless values.

Pre-processing

In [10]:
```
cb.isnull().sum()#checkin for null values
```

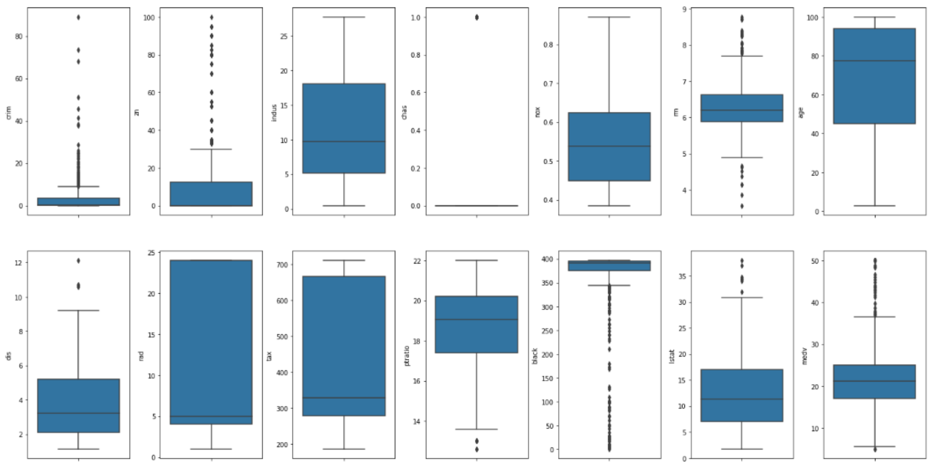
Out[10]:
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
black      0
lstat      0
medv       0
dtype: int64
```

4. Exploratory Data Analysis

- EDA is the process of analyzing the given data for finding some useful relationships between the dataset, useful patterns, statistical representation, mean, median, finding any outliers etc.,
- In this step , I used Box-plot representation using *'seaborn'* library for spotting out the outliers.

EDA

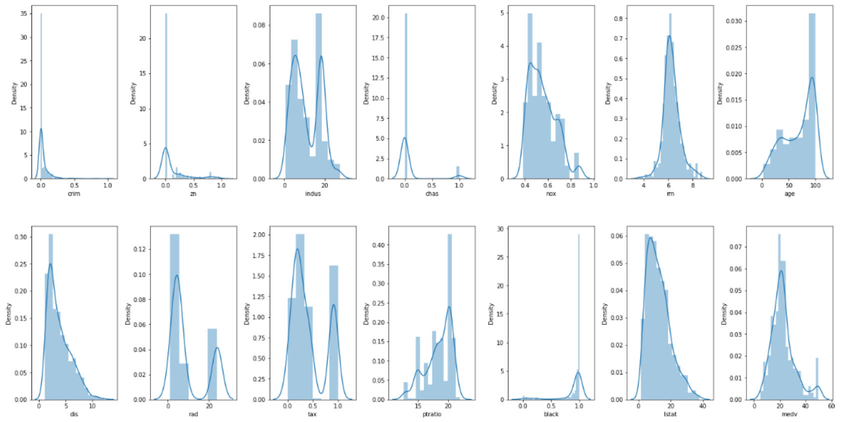
In [11]:
```
#boxplots for finding outliers

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



5. Min-Max Normalization

- It is a generally used Normalization technique.
- The goal of normalization is to make every datapoint have the same scale so each feature is equally important.

```
In [14]: fig,ax=plt.subplots(ncols=7,nrows=2,figsize=(20,10))
         index=0
         ax=ax.flatten()
         for col,value in cb.items():
             sns.distplot(value,ax=ax[index])
             index+=1
         plt.tight_layout(pad=0.5,w_pad=0.7,h_pad=5.0)
```

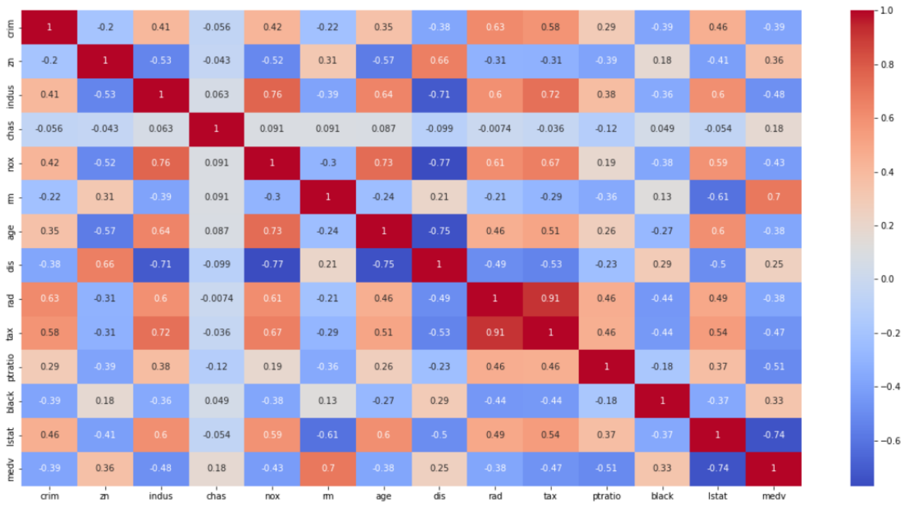


6. Correlation Matrix

- A correlation matrix is simply a table which displays the correlation coefficients for different variables.
- It is a powerful tool to summarize a large dataset and to identify and visualize patterns in the given data.

Correlation Matrix

```
In [18]: corr=cb.corr()
         plt.figure(figsize=(20,10))
         sns.heatmap(corr,annot=True,cmap="coolwarm")

Out[18]: <AxesSubplot:>
```

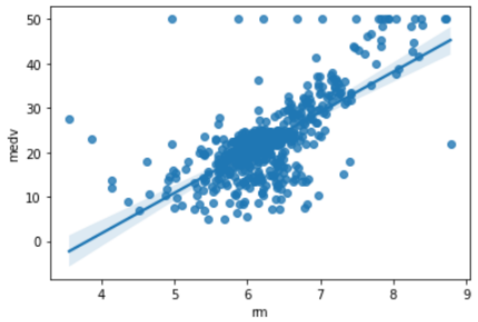


Observation 1:

From the above Plot (medv vs lstat), we can say that, the PRICE decreases when the LSTAT is increasing

```
sns.regplot(y=cb['medv'],x=cb['rm'])
```

```
<AxesSubplot:xlabel='rm', ylabel='medv'>
```



Observation 2:

From the above plot (medv vs rm) ,we can say that, the PRICE increases when the RM is increasing

7. Splitting the Dataset into train data and test data

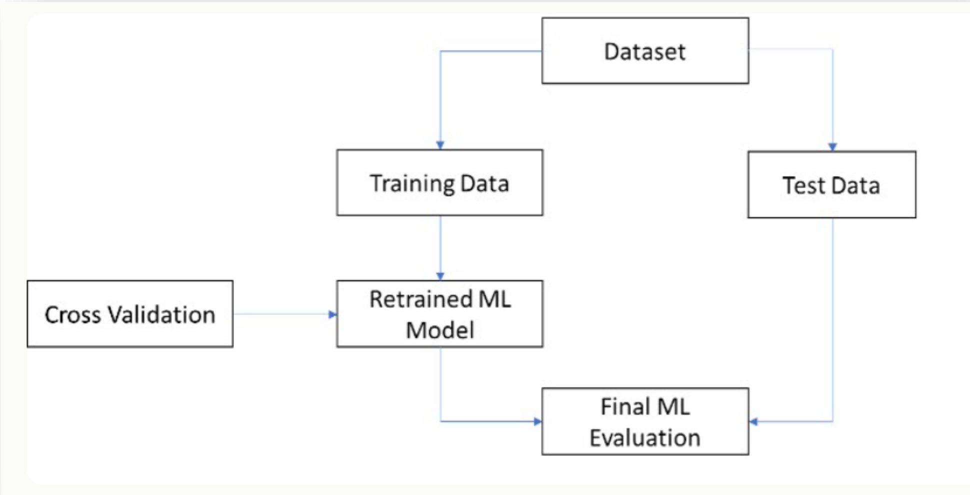
- The train-test split procedure is used to estimate the performance of machine learning algorithms whether they are used to make predictions on data, and not used to train the model.
  - **Train Dataset:** Used to fit the machine learning model.
  - **Test Dataset**: Used to evaluate the fit machine learning model.

```python
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import mean_squared_error

def train(model,X,y):
    model.fit(X,y)
    x_train,x_test,y_train,y_test=train_test_split(X,y,random_state=42)
    model.fit(x_train,y_train)
    pred=model.predict(x_test)

    cv_score=cross_val_score(model,X,y,scoring='neg_mean_squared_error',cv=5)
    cv_score=np.abs(np.mean(cv_score))

    print("MODEL INFERENCE")
    print("Mean Squared Error : ",mean_squared_error(y_test,pred))
    print("CV Score: ",cv_score)
```

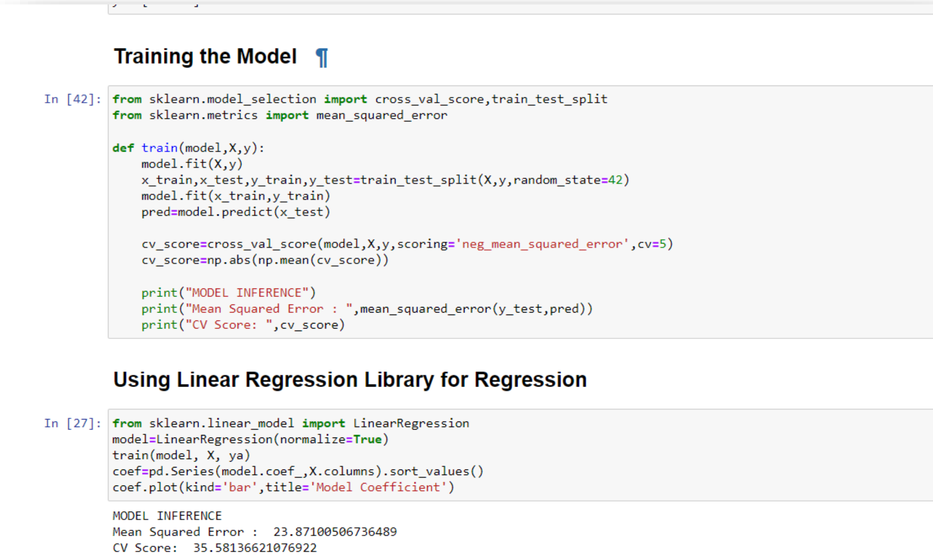
8. Training the model & 9. Using Regression techniques

- This part of the step gives us the desired output.
- It is the most important of any ML/ Data science problems
- It is a process in which a machine learning (ML) algorithm is fed with sufficient training data to learn from.
- Here I used different type of regression techniques to develop a model:
    - Linear regression
    - Random Forrest
    - XGBoost
    - Decision-Tree
- The output of these models gives us two readings:
    - **Mean Squared Error (MSE)**
    - **Cross Validation Score (CV Score)**

- The lesser the above two values, the greater the efficiency of the models.
- **Cross-validation** is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.
- It is a statistical technique employed to estimate a machine learning's overall accuracy.



- **MSE** is the error of deviation from the actual and the predicted data. So, MSE should be less for getting efficient output.



Training the Model ¶

In [42]:
```
from sklearn.model_selection import cross_val_score,train_test_split
from sklearn.metrics import mean_squared_error

def train(model,X,y):
    model.fit(X,y)
    x_train,x_test,y_train,y_test=train_test_split(X,y,random_state=42)
    model.fit(x_train,y_train)
    pred=model.predict(x_test)

    cv_score=cross_val_score(model,X,y,scoring='neg_mean_squared_error',cv=5)
    cv_score=np.abs(np.mean(cv_score))

    print("MODEL INFERENCE")
    print("Mean Squared Error : ",mean_squared_error(y_test,pred))
    print("CV Score: ",cv_score)
```

Using Linear Regression Library for Regression

In [27]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression(normalize=True)
train(model, X, ya)
coef=pd.Series(model.coef_,X.columns).sort_values()
coef.plot(kind='bar',title='Model Coefficient')
```

```
MODEL INFERENCE
Mean Squared Error :  23.87100506736489
CV Score:  35.58136621076922
```

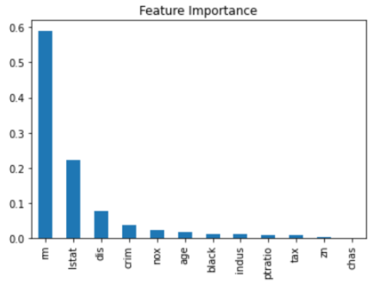
Using Decision-Tree Library for Regression

In [28]:

```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 INFERENCE
Mean Squared Error :  19.826850393700788
CV Score:  41.139343234323434
```

Out[28]: `<AxesSubplot:title={'center':'Feature Importance'}>`



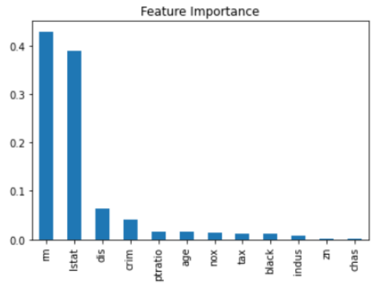
Using Random-Forest Library for Regression

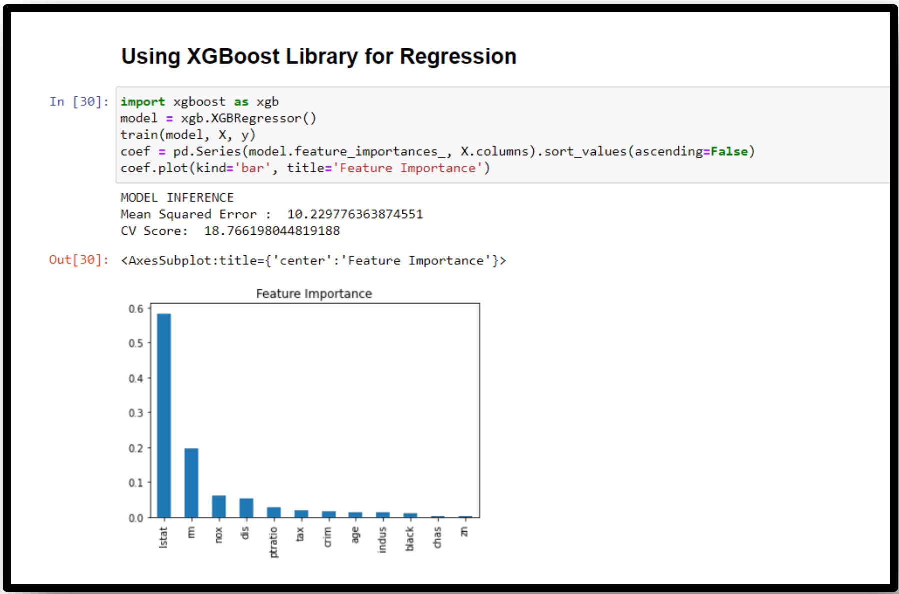
In [29]:

```python
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 INFERENCE
Mean Squared Error :  10.198220259842529
CV Score:  21.149270378198402
```

Out[29]: `<AxesSubplot:title={'center':'Feature Importance'}>`





Using XGBoost Library for Regression

In [30]:
```
import xgboost as xgb
model = xgb.XGBRegressor()
train(model, X, y)
coef = pd.Series(model.feature_importances_, X.columns).sort_values(ascending=False)
coef.plot(kind='bar', title='Feature Importance')
```

```
MODEL INFERENCE
Mean Squared Error :  10.229776363874551
CV Score:  18.766198044819188
```

Out[30]: <AxesSubplot:title={'center':'Feature Importance'}>

10. Conclusion

- W.r.t **'medv'** - Out of all the Regression Techniques,
- **XGBoost Method** only gives the output while training the model i.e., MSE and CV score is least for this technique. So, after pre-processing and EDA techniques**, the model developed using XGBoost is apt for the given Boston House Prediction Dataset** with CV readings and MSE:
  - **MSE** : 10.22977
  - **CV Score** : 18.7661