Practical No 04

Create a Linear Regression Model using Python/R to predict home prices using Boston Housing Dataset (<https://www.kaggle.com/c/boston-housing>). The Boston Housing dataset contains information about various houses in Boston through different parameters. There are 506 samples and 14 feature variables in this dataset. The objective is to predict the value of prices of the house using the given features.

# Dataset Information

Boston House Prices Dataset was collected in 1978 and has 506 entries with 14 attributes or features for homes from various suburbs in Boston.

**Boston Housing Dataset Attribute Information (in order):**

* **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**

# Import modules

**In [1]:**

import pandas as pd import numpy as np import seaborn as sns

import matplotlib.pyplot as plt import warnings

%matplotlib inline warnings.filterwarnings('ignore')

# Loading the dataset

**In [2]:**

df = pd.read\_csv("Boston Dataset.csv") df.drop(columns=['Unnamed: 0'], axis=0, inplace=True) df.head()

**Out [2]:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| crim | zn | indus | chas | nox | rm | age | dis | rad | tax | ptratio | black | lstat | medv |
| 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 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 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 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 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 |

**In [3]:**

# statistical info df.describe()

**Out [3]:**

crim zn indus chas nox rm age dis rad

25% 0.082045 0.000000 5.190000 0.000000 0.449000 5.885500 45.025000 2.100175 4.000000 2

min 0.006320 0.000000 0.460000 0.000000 0.385000 3.561000 2.900000 1.129600 1.000000 1

std 8.601545 23.322453 6.860353 0.253994 0.115878 0.702617 28.148861 2.105710 8.707259 1

mean 3.613524 11.363636 11.136779 0.069170 0.554695 6.284634 68.574901 3.795043 9.549407 4

count 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000 506.000000 5

0

0

6

8

7

50% 0.256510 0.000000 9.690000 0.000000 0.538000 6.208500 77.500000 3.207450 5.000000 33

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | crim | | zn | indus | chas | nox | rm | age | dis | rad | |
|  |  | 75% | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 | 5.188425 | 24.000000 | 6 |
| **In** | **[4]:** | max 88.976200 100.000000 27.740000  **# datatype info df.info()**  **<class 'pandas.core.frame.DataFrame'> RangeIndex: 506 entries, 0 to 505 Data columns (total 14 columns):**  **# Column Non-Null Count Dtype** | | | | 1.000000 | 0.871000 | 8.780000 | 100.000000 | 12.126500 | 24.000000 | 7 |
|  |  | 1. **crim 506 non-null float64** 2. **zn 506 non-null float64** 3. **indus 506 non-null float64** 4. **chas 506 non-null int64** 5. **nox 506 non-null float64** 6. **rm 506 non-null float64** 7. **age 506 non-null float64** 8. **dis 506 non-null float64** 9. **rad 506 non-null int64** 10. **tax 506 non-null int64** 11. **ptratio 506 non-null float64** 12. **black 506 non-null float64** 13. **lstat 506 non-null float64** 14. **medv 506 non-null float64 dtypes: float64(11), int64(3)**   **memory usage: 55.5 KB** | | | | | | | | | | |
|  |  | 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** | | | | | | | | | | |
|  |  | Exploratory Data Analysis | | | | | | | | | | |
| **In** | **[6]:** | **# create box plots**  **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)** | | | | | | | | | | |

6

1

**In [7]:**

# create dist plot

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

**In [8]:**

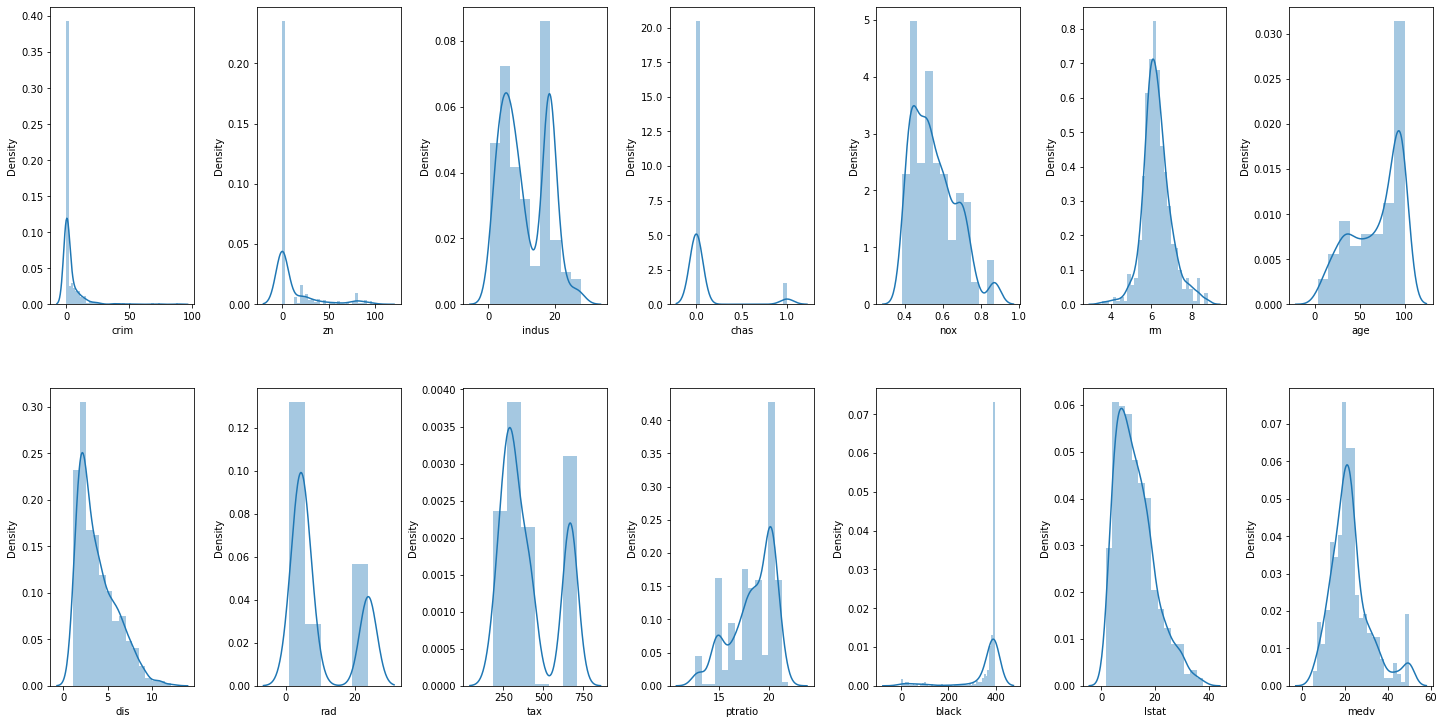
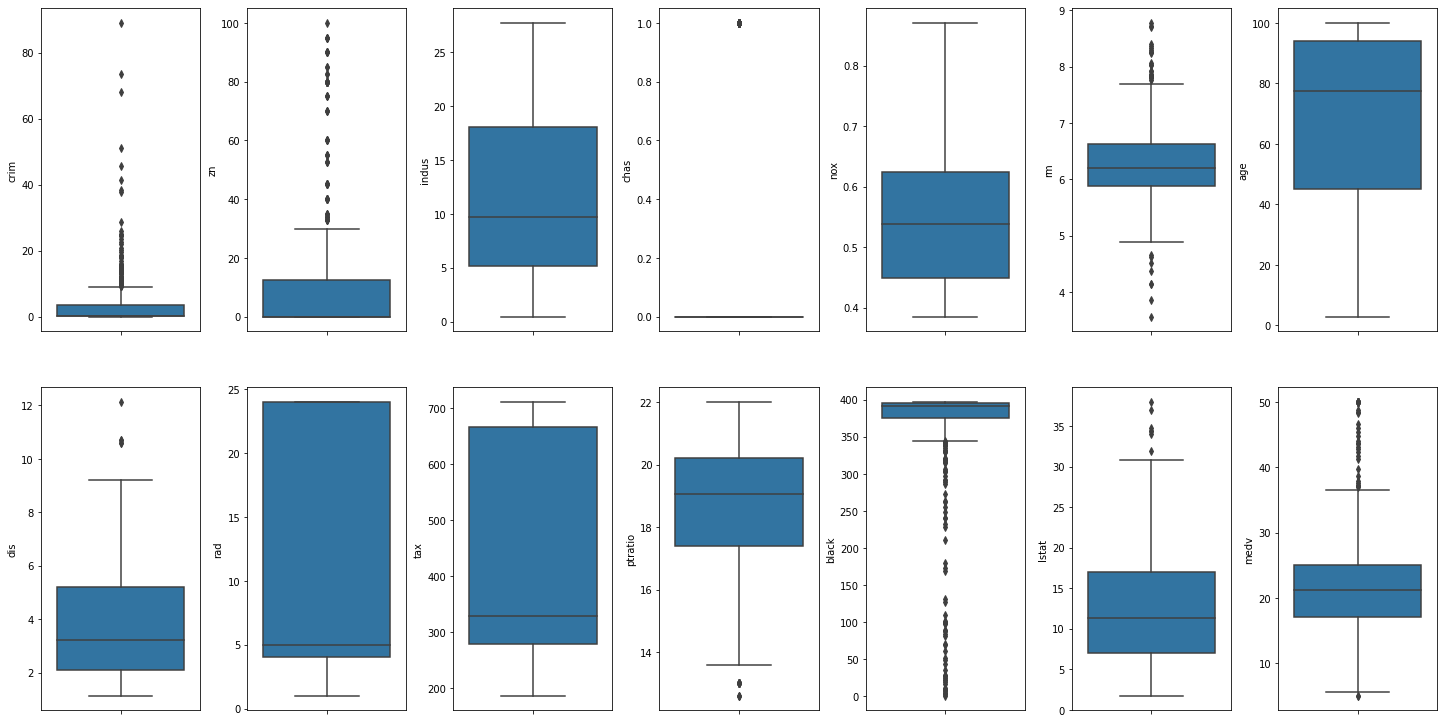
cols = ['crim', 'zn', 'tax', 'black'] for col in cols:

# find minimum and maximum of that column minimum = min(df[col])

maximum = max(df[col])

df[col] = (df[col] - minimum) / (maximum - minimum)

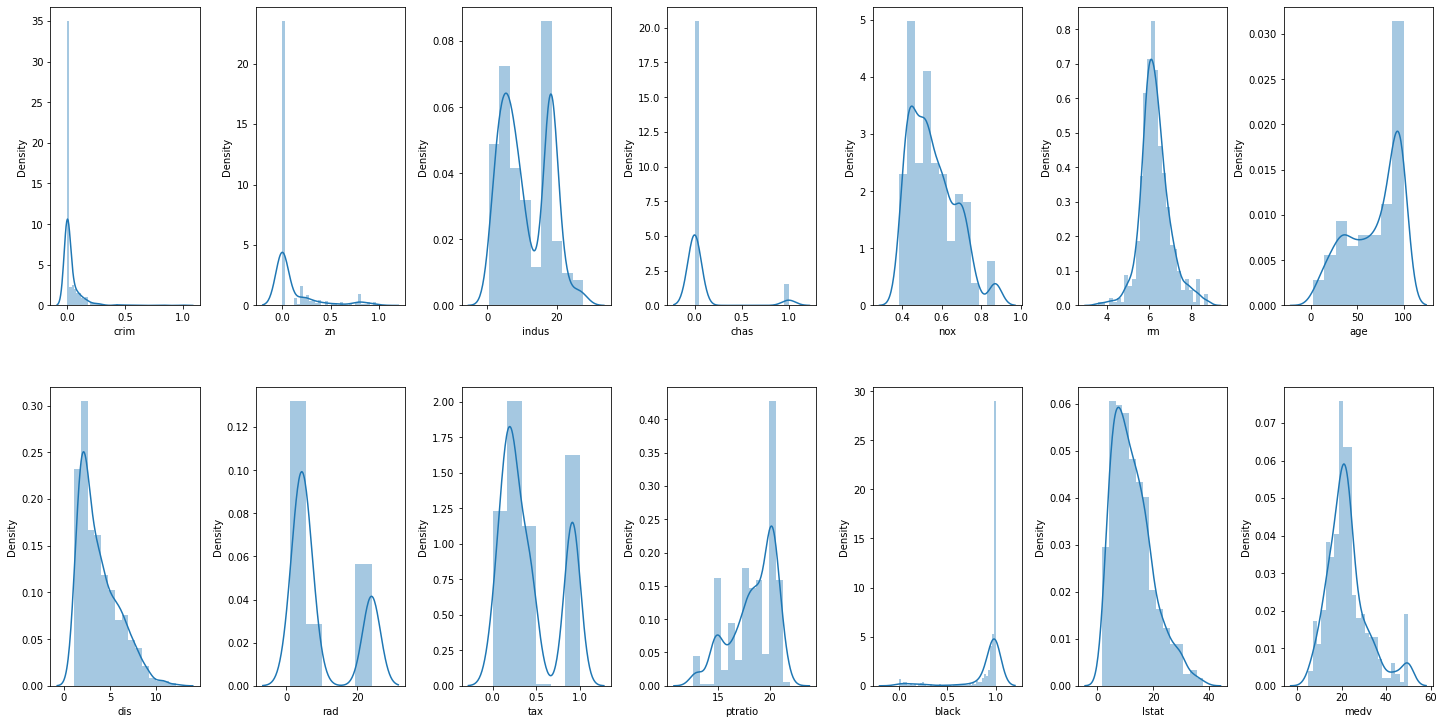
**In [9]:**



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)

**In [10]:**

# standardization

from sklearn import preprocessing scalar = preprocessing.StandardScaler()

# fit our data

scaled\_cols = scalar.fit\_transform(df[cols]) scaled\_cols = pd.DataFrame(scaled\_cols, columns=cols) scaled\_cols.head()

**Out [10]:**

|  |  |  |  |
| --- | --- | --- | --- |
| crim | zn | tax | black |
| 0 -0.419782 | 0.284830 | -0.666608 | 0.441052 |
| 1 -0.417339 | -0.487722 | -0.987329 | 0.441052 |
| 2 -0.417342 | -0.487722 | -0.987329 | 0.396427 |
| 3 -0.416750 | -0.487722 | -1.106115 | 0.416163 |
| 4 -0.412482 | -0.487722 | -1.106115 | 0.441052 |

**In [11]:**

for col in cols:

df[col] = scaled\_cols[col]

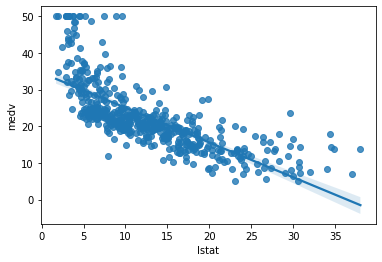
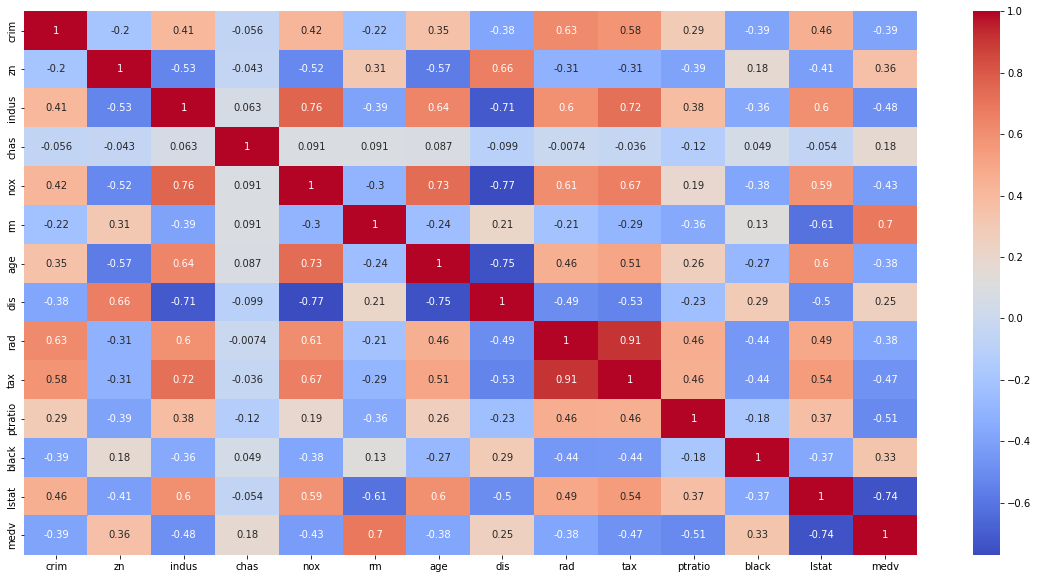
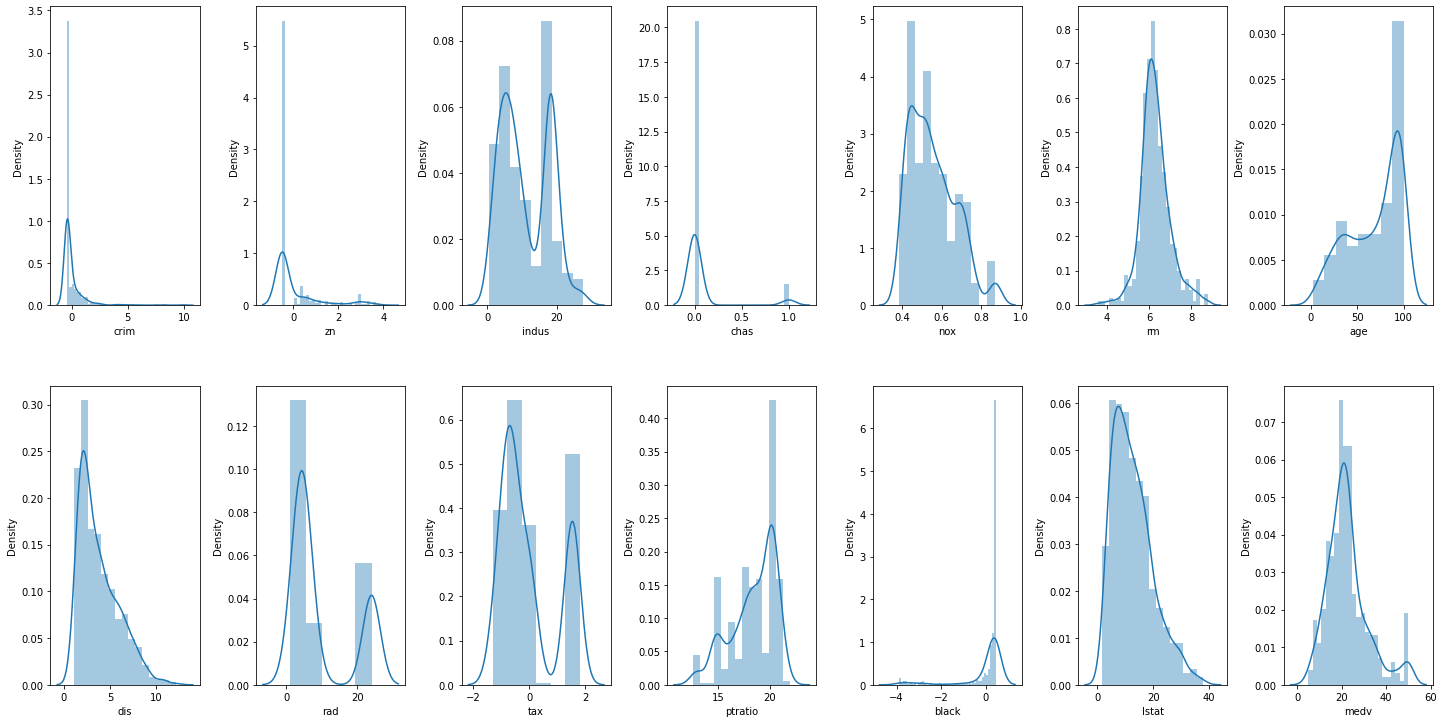
**In [12]:**

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)



# Coorelation Matrix

**In [13]:**

corr = df.corr() plt.figure(figsize=(20,10))

sns.heatmap(corr, annot=True, cmap='coolwarm')

**Out [13]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3db6fb20>**

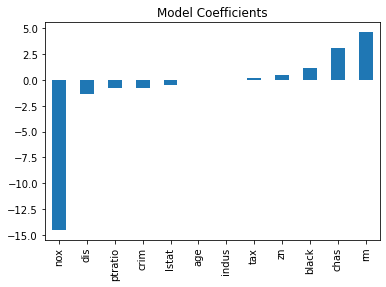
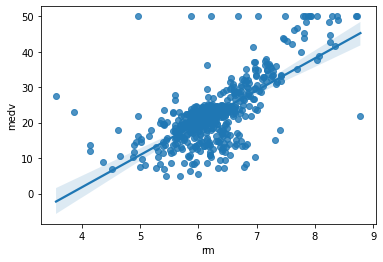
**In [14]:**

sns.regplot(y=df['medv'], x=df['lstat'])

**Out [14]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3a672160>**

**In [15]:**

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



**Out [15]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa38815af0>**

# Input Split

**In [16]:**

X = df.drop(columns=['medv', 'rad'], axis=1) y = df['medv']

# Model Training

**In [17]:**

from sklearn.model\_selection import cross\_val\_score, train\_test\_split from sklearn.metrics import mean\_squared\_error

def train(model, X, y): # train the model

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=42) model.fit(x\_train, y\_train)

# predict the training set pred = model.predict(x\_test)

# perform cross-validation

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 Report") print("MSE:",mean\_squared\_error(y\_test, pred)) print('CV Score:', cv\_score)

**In [18]:**

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: 23.8710050673649**

**CV Score: 35.581366210769204**

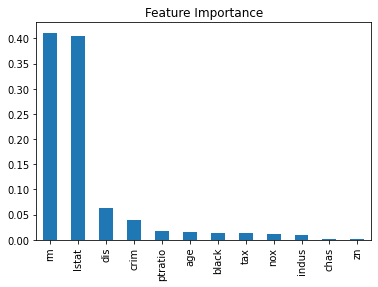
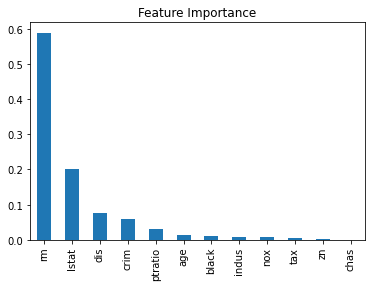
**Out [18]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3cb1ac10>**

**In [19]:**

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: 10.865118110236219**

**CV Score: 44.27399242865463**

**Out [19]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3cdd7d00>**

**In [20]:**

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: 9.302050535433075**

**CV Score: 21.95170135701804**

**Out [20]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3a382730>**

**In [21]:**

from sklearn.ensemble import ExtraTreesRegressor model = ExtraTreesRegressor()

train(model, X, y)

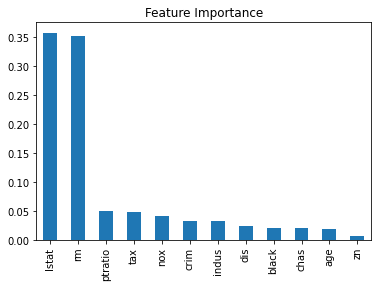
coef = pd.Series(model.feature\_importances\_, X.columns).sort\_values(ascending=False) coef.plot(kind='bar', title='Feature Importance')

**Model Report**

**MSE: 11.587783771653548**

**CV Score: 19.738606818210037**

**Out [21]: <matplotlib.axes.\_subplots.AxesSubplot at 0x7faa3a39ebe0>**



**c**

**In [ ]:**

**In [ ]:**

**Traceba**

**NameError**

**----> 1 pred = model.predict(x\_test) NameError: name 'x\_test' is not defined**

**In [22]:**

**pred**

**pred model.predict(x\_test)**

**=**