In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import OLSInfluence
```

In [2]:
```
df = pd.read_csv('Boston.csv')
df_copy = df.copy()
df_copy.rename(columns={'Unnamed: 0': 'i'}, inplace=True)
df_copy.head()
# 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
```

Out[2]:

	i	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	me
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	2
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	2
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	3
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	3
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	3

Simple Linear Regression Models

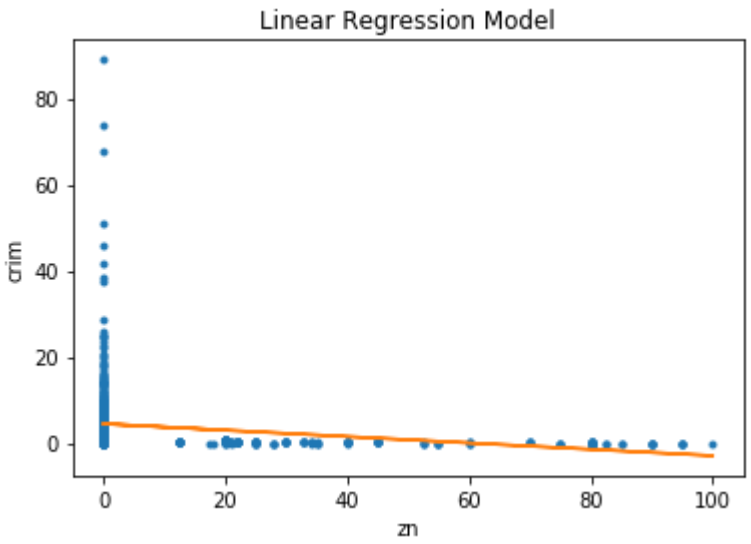
In [3]:
```
# Extracts the response column
y_true = np.asarray(df_copy['crim'])

# Tracks old predictor coefficients
old_column_values = []
```

Problem a

MODEL 1: Only 'zn' column

In [4]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['zn'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('zn')
plt.ylabel('crim')
```

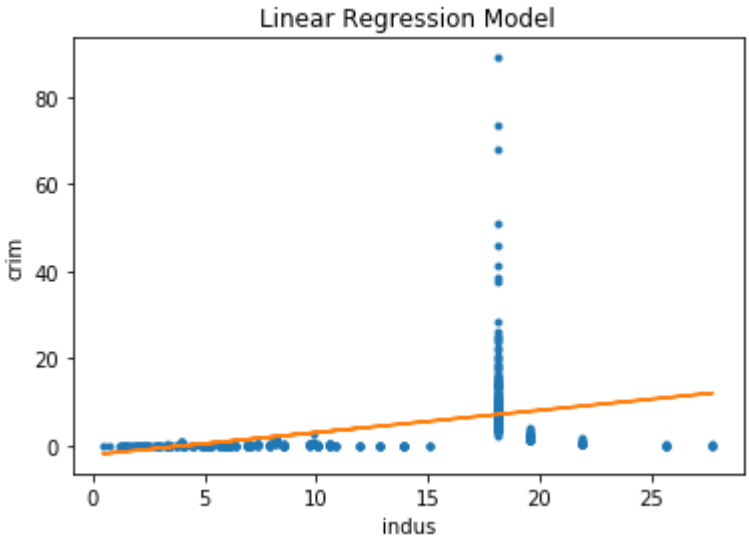
Out[4]: Text(0,0.5,'crim')



MODEL 2: Only 'indus' column

In [5]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['indus'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('indus')
plt.ylabel('crim')
```

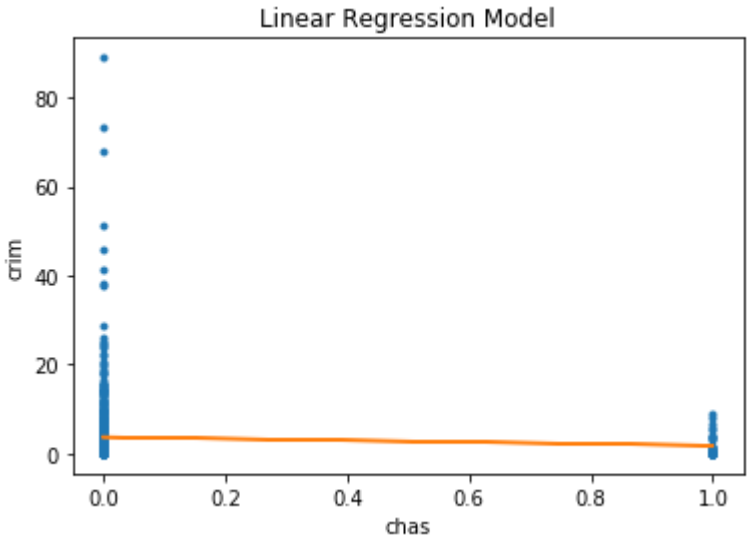
Out[5]:
```
Text(0,0.5,'crim')
```



MODEL 3: Only 'chas' column

In [6]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['chas'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('chas')
plt.ylabel('crim')
```

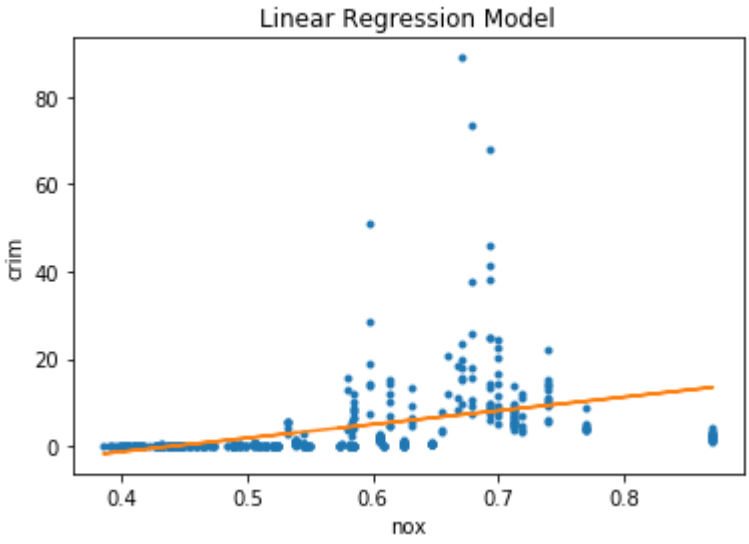
Out[6]: Text(0,0.5,'crim')



MODEL 4: Only 'nox' column

In [7]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['nox'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('nox')
plt.ylabel('crim')
```

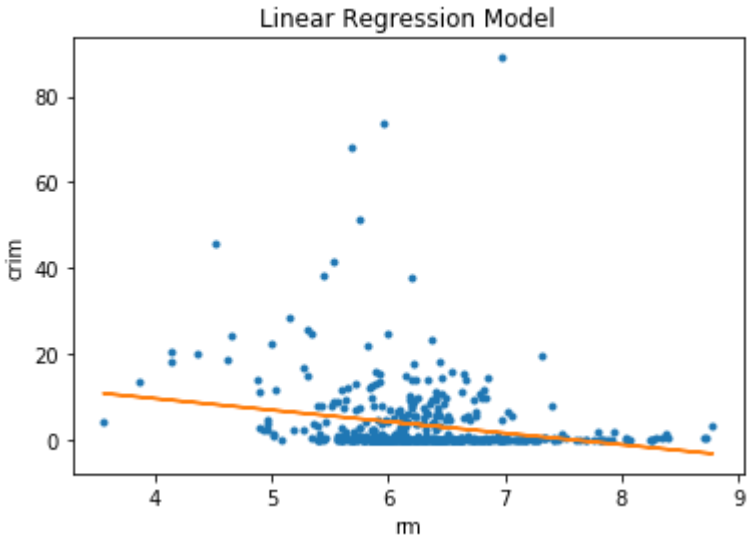
Out[7]: Text(0,0.5,'crim')



MODEL 5: Only 'rm' column

In [8]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['rm'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('rm')
plt.ylabel('crim')
```

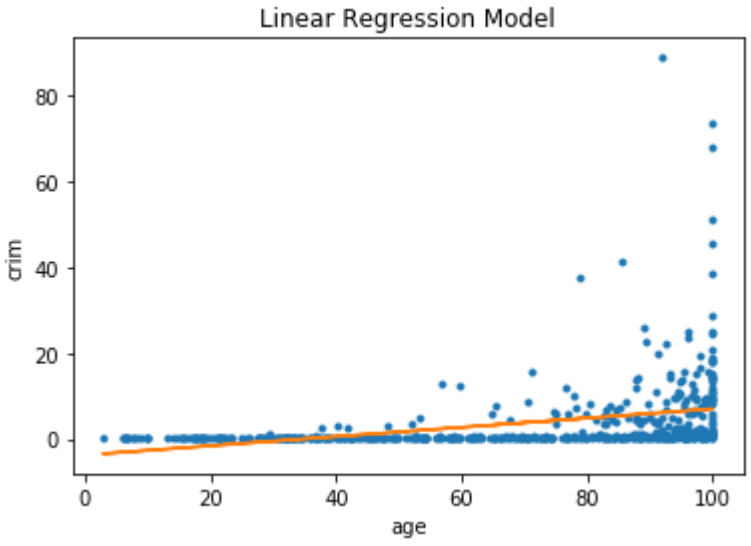
Out[8]: Text(0,0.5,'crim')



MODEL 6: Only 'age' column

In [9]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['age'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('age')
plt.ylabel('crim')
```

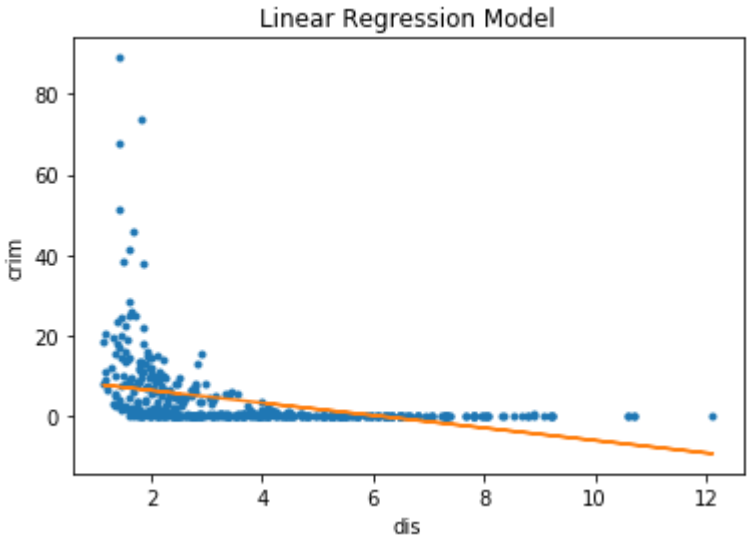
Out[9]: Text(0,0.5,'crim')



MODEL 7: Only 'dis' column

In [10]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['dis'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('dis')
plt.ylabel('crim')
```

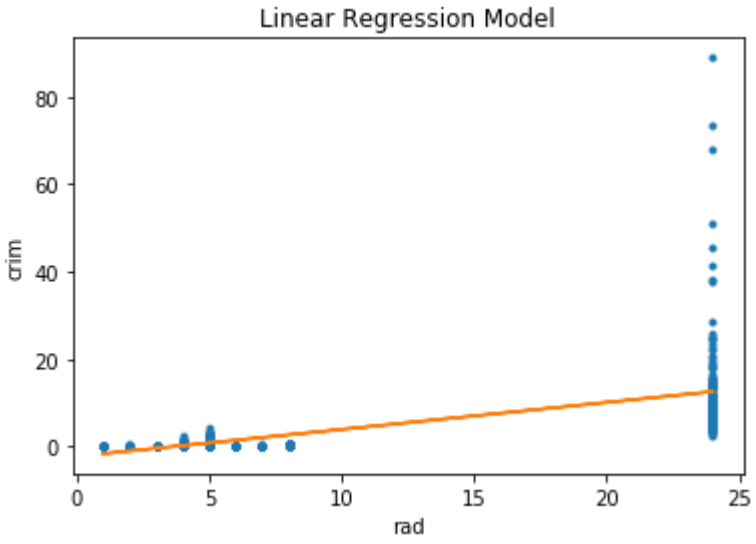
Out[10]: Text(0,0.5,'crim')



MODEL 8: Only 'rad' column

In [11]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['rad'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('rad')
plt.ylabel('crim')
```

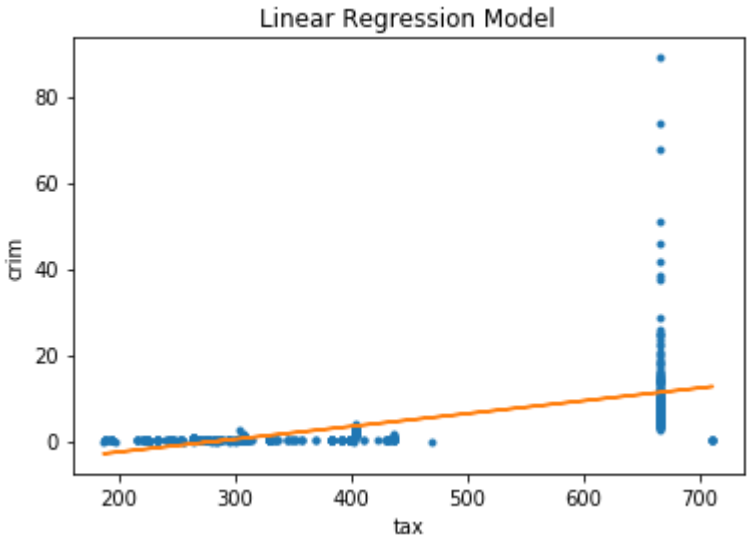
Out[11]: Text(0,0.5,'crim')



MODEL 9: Only 'tax' column

In [12]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['tax'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('tax')
plt.ylabel('crim')
```

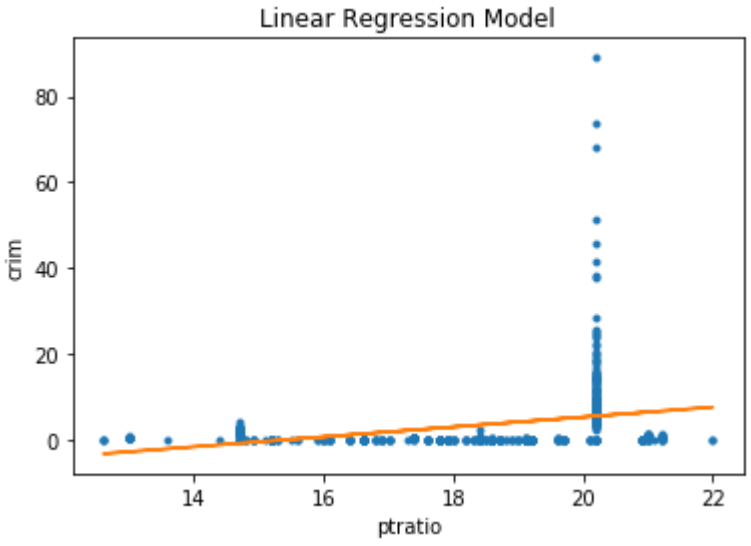
Out[12]: Text(0,0.5,'crim')



MODEL 10: Only 'ptratio' column

In [13]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['ptratio'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('ptratio')
plt.ylabel('crim')
```

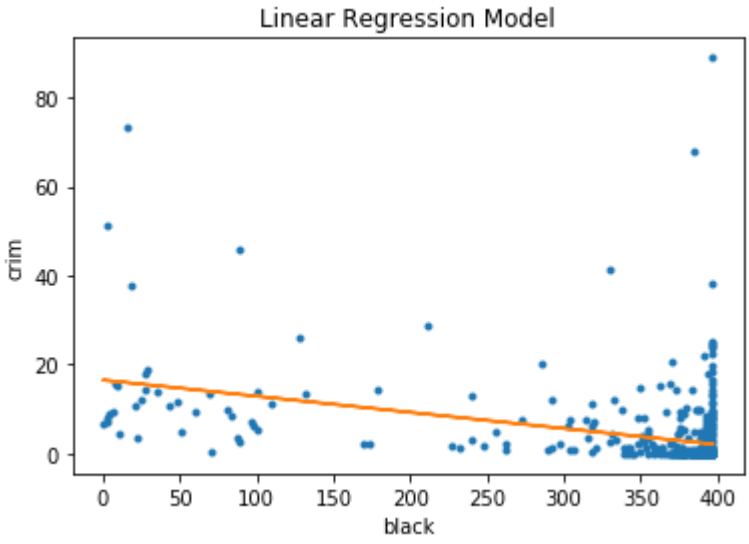
Out[13]: Text(0,0.5,'crim')



MODEL 11: Only 'black' column

In [14]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['black'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('black')
plt.ylabel('crim')
```

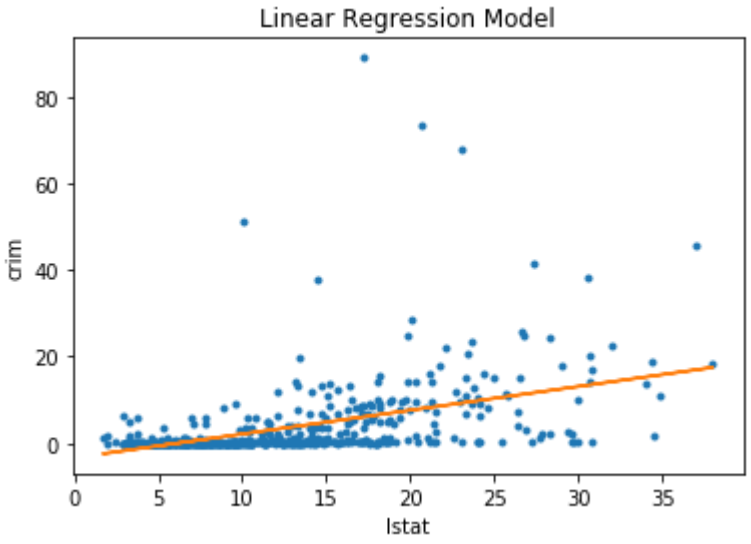
Out[14]: Text(0,0.5,'crim')



MODEL 12: Only 'lstat' column

In [15]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['lstat'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('lstat')
plt.ylabel('crim')
```

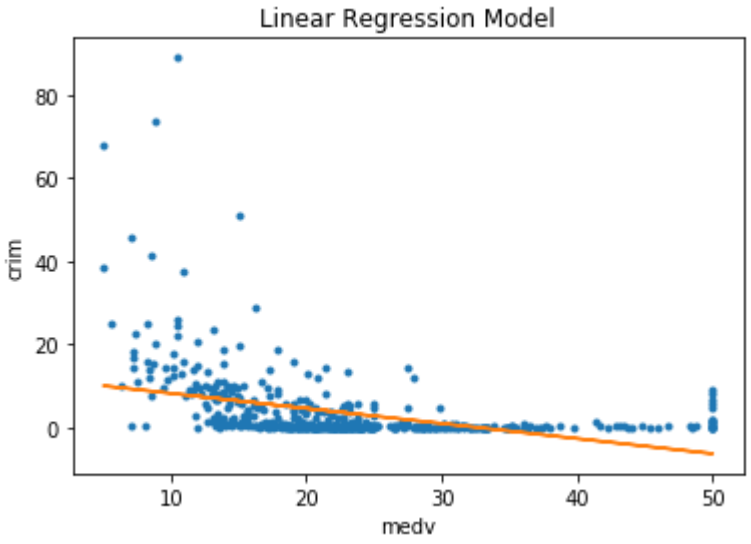
Out[15]: Text(0,0.5,'crim')



MODEL 13: Only 'medv' column

In [16]:
```
# Extracts the predictor columns, and creates design matrix
X_var = np.asarray(df_copy['medv'])
X_design = sm.add_constant(X_var)

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients
intercept = result.params[0]
slope = result.params[1]

old_column_values.append(slope)

# Create scatterplot with linear regression model
plt.plot(X_var, y_true, '.') # Plots the data points
plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
plt.title('Linear Regression Model')
plt.xlabel('medv')
plt.ylabel('crim')
```

Out[16]: Text(0,0.5,'crim')



Problem b

Multiple Linear Regression Model

In [17]:
```
# Extracts the predictor columns, and creates design matrix
column_names = list(df_copy.iloc[:,2:].columns)
X_var = np.asarray(df_copy.iloc[:,2:])
X_design = sm.add_constant(X_var)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places

# Extracts the slope and intercept coefficients

# # Create scatterplot with linear regression model
# plt.plot(X_var, y_true, '.') # Plots the data points
# plt.plot(X_var, slope*X_var + intercept) # Plots the linear regression line
# plt.title('Linear Regression Model')
# plt.xlabel('medv')
# plt.ylabel('crim')
```

Out[17]:

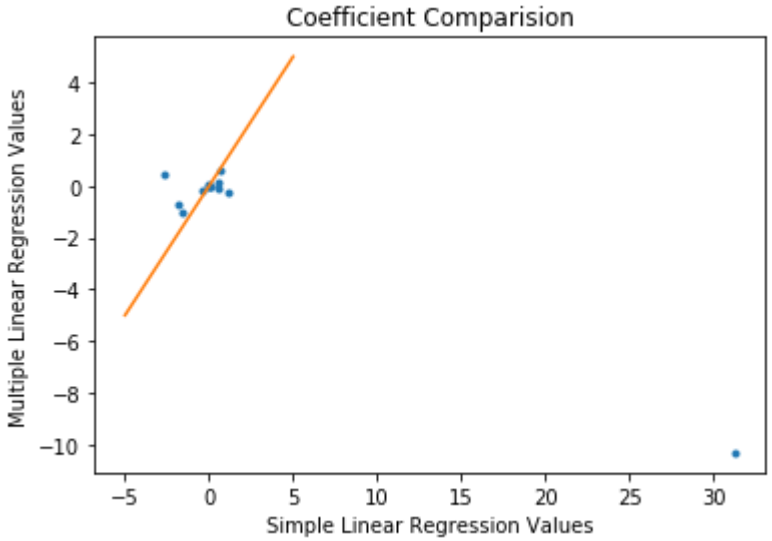
	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	17.0332	2.3543	0.0189
1	zn	0.0449	2.3943	0.0170
2	indus	-0.0639	-0.7656	0.4443
3	chas	-0.7491	-0.6348	0.5259
4	nox	-10.3135	-1.9550	0.0512
5	rm	0.4301	0.7019	0.4831
6	age	0.0015	0.0810	0.9355
7	dis	-0.9872	-3.5029	0.0005
8	rad	0.5882	6.6804	0.0000
9	tax	-0.0038	-0.7332	0.4638
10	ptratio	-0.2711	-1.4539	0.1466
11	black	-0.0075	-2.0520	0.0407
12	lstat	0.1262	1.6667	0.0962
13	medv	-0.1989	-3.2865	0.0011

Problem c

Comparison of Coefficients

In [18]:
```
plt.plot(old_column_values, result.params[1:], '.') # Plots the data points
plt.plot(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
plt.title('Coefficient Comparision')
plt.xlabel('Simple Linear Regression Values')
plt.ylabel('Multiple Linear Regression Values')
```

Out[18]: Text(0,0.5,'Multiple Linear Regression Values')



Non-linear Associations

In [19]:
```
# Extracts the response column
y_true = np.asarray(df_copy['crim'])
```

'zn'

In [21]:
```
# Creates nonlinear columns
df_temp = df_copy[['zn']].copy()
df_temp['zn^2'] = df_temp['zn'] * df_temp['zn']
df_temp['zn^3'] = df_temp['zn^2'] * df_temp['zn']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[21]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	4.8461	11.1922	0.0000
1	zn	-0.3322	-3.0252	0.0026
2	zn^2	0.0065	1.6791	0.0938
3	zn^3	-0.0000	-1.2030	0.2295

'indus'

In [22]:
```
# Creates nonlinear columns
df_temp = df_copy[['indus']].copy()
df_temp['indus^2'] = df_temp['indus'] * df_temp['indus']
df_temp['indus^3'] = df_temp['indus^2'] * df_temp['indus']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[22]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	3.6626	2.3269	0.0204
1	indus	-1.9652	-4.0773	0.0001
2	indus^2	0.2519	6.4070	0.0000
3	indus^3	-0.0070	-7.2920	0.0000

'chas'

In [24]:
```
# Creates nonlinear columns
df_temp = df_copy[['chas']].copy()
df_temp['chas^2'] = df_temp['chas'] * df_temp['chas']
df_temp['chas^3'] = df_temp['chas^2'] * df_temp['chas']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[24]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	3.7444	9.4530	0.0000
1	chas	-0.6309	-1.2567	0.2094
2	chas^2	-0.6309	-1.2567	0.2094
3	chas^3	-0.6309	-1.2567	0.2094

'nox'

In [25]:
```
# Creates nonlinear columns
df_temp = df_copy[['nox']].copy()
df_temp['nox^2'] = df_temp['nox'] * df_temp['nox']
df_temp['nox^3'] = df_temp['nox^2'] * df_temp['nox']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[25]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	233.0866	6.9282	0.0
1	nox	-1279.3713	-7.5082	0.0
2	nox^2	2248.5441	8.0334	0.0
3	nox^3	-1245.7029	-8.3446	0.0

'rm'

In [26]:
```
# Creates nonlinear columns
df_temp = df_copy[['rm']].copy()
df_temp['rm^2'] = df_temp['rm'] * df_temp['rm']
df_temp['rm^3'] = df_temp['rm^2'] * df_temp['rm']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[26]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	112.6246	1.7457	0.0815
1	rm	-39.1501	-1.2503	0.2118
2	rm^2	4.5509	0.9084	0.3641
3	rm^3	-0.1745	-0.6615	0.5086

'age'

In [27]:
```
# Creates nonlinear columns
df_temp = df_copy[['age']].copy()
df_temp['age^2'] = df_temp['age'] * df_temp['age']
df_temp['age^3'] = df_temp['age^2'] * df_temp['age']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[27]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	-2.5488	-0.9204	0.3578
1	age	0.2737	1.4683	0.1427
2	age^2	-0.0072	-1.9878	0.0474
3	age^3	0.0001	2.7237	0.0067

'dis'

In [29]:
```
# Creates nonlinear columns
df_temp = df_copy[['dis']].copy()
df_temp['dis^2'] = df_temp['dis'] * df_temp['dis']
df_temp['dis^3'] = df_temp['dis^2'] * df_temp['dis']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[29]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	30.0476	12.2850	0.0
1	dis	-15.5544	-8.9600	0.0
2	dis^2	2.4521	7.0783	0.0
3	dis^3	-0.1186	-5.8135	0.0

'rad'

In [30]:
```
# Creates nonlinear columns
df_temp = df_copy[['rad']].copy()
df_temp['rad^2'] = df_temp['rad'] * df_temp['rad']
df_temp['rad^3'] = df_temp['rad^2'] * df_temp['rad']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[30]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	-0.6055	-0.2954	0.7678
1	rad	0.5127	0.4913	0.6234
2	rad^2	-0.0752	-0.5061	0.6130
3	rad^3	0.0032	0.7031	0.4823

'tax'

In [31]:
```
# Creates nonlinear columns
df_temp = df_copy[['tax']].copy()
df_temp['tax^2'] = df_temp['tax'] * df_temp['tax']
df_temp['tax^3'] = df_temp['tax^2'] * df_temp['tax']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[31]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	19.1836	1.6263	0.1045
1	tax	-0.1533	-1.6023	0.1097
2	tax^2	0.0004	1.4877	0.1375
3	tax^3	-0.0000	-1.1668	0.2439

'ptratio'

In [32]:
```
# Creates nonlinear columns
df_temp = df_copy[['ptratio']].copy()
df_temp['ptratio^2'] = df_temp['ptratio'] * df_temp['ptratio']
df_temp['ptratio^3'] = df_temp['ptratio^2'] * df_temp['ptratio']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[32]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	477.1840	3.0434	0.0025
1	ptratio	-82.3605	-2.9793	0.0030
2	ptratio^2	4.6353	2.8821	0.0041
3	ptratio^3	-0.0848	-2.7433	0.0063

'black'

In [33]:
```
# Creates nonlinear columns
df_temp = df_copy[['black']].copy()
df_temp['black^2'] = df_temp['black'] * df_temp['black']
df_temp['black^3'] = df_temp['black^2'] * df_temp['black']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[33]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	18.2637	7.9238	0.0000
1	black	-0.0836	-1.4834	0.1386
2	black^2	0.0002	0.7162	0.4742
3	black^3	-0.0000	-0.6078	0.5436

'lstat'

In [34]:
```
# Creates nonlinear columns
df_temp = df_copy[['lstat']].copy()
df_temp['lstat^2'] = df_temp['lstat'] * df_temp['lstat']
df_temp['lstat^3'] = df_temp['lstat^2'] * df_temp['lstat']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[34]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	1.2010	0.5920	0.5541
1	lstat	-0.4491	-0.9660	0.3345
2	lstat^2	0.0558	1.8522	0.0646
3	lstat^3	-0.0009	-1.5170	0.1299

'medv'

In [23]:
```
# Creates nonlinear columns
df_temp = df_copy[['medv']].copy()
df_temp['medv^2'] = df_temp['medv'] * df_temp['medv']
df_temp['medv^3'] = df_temp['medv^2'] * df_temp['medv']

X_var = np.asarray(df_temp)
X_design = sm.add_constant(X_var)

column_names = list(df_temp.columns)
column_names.insert(0, 'Intercept')

# Create the Simple Linear Regression Model and fit it
MLRmodel = sm.OLS(y_true, X_design)
result = MLRmodel.fit()

# Create table of model prediction results
data = {'Attributes': column_names,
        'Coefficient Beta_i': result.params,
        't-Values': result.tvalues,
        'p-Values': result.pvalues
       }
ModelResults = pd.DataFrame(data)
ModelResults.round(4) # Round values in table to 4-decimal places
```

Out[23]:

	Attributes	Coefficient Beta_i	t-Values	p-Values
0	Intercept	53.1655	15.8405	0.0
1	medv	-5.0948	-11.7438	0.0
2	medv^2	0.1555	9.0455	0.0
3	medv^3	-0.0015	-7.3120	0.0

In [ ]: