Polynomial Regression

In [1]:

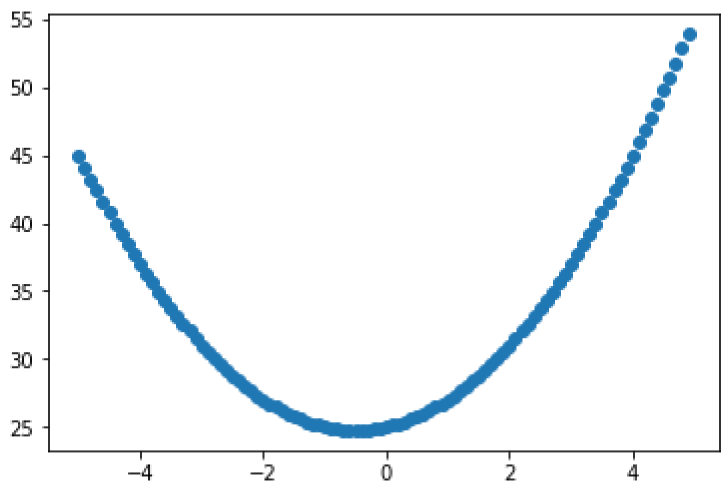
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:

```
x = np.arange(-5.0, 5.0, 0.1)
# y = ax^2 + bx + c
y = 1 * x **2 + 1 * x + 25
plt.scatter(x, y)
```

Out[2]:

```
<matplotlib.collections.PathCollection at 0x282fb4c19a0>
```

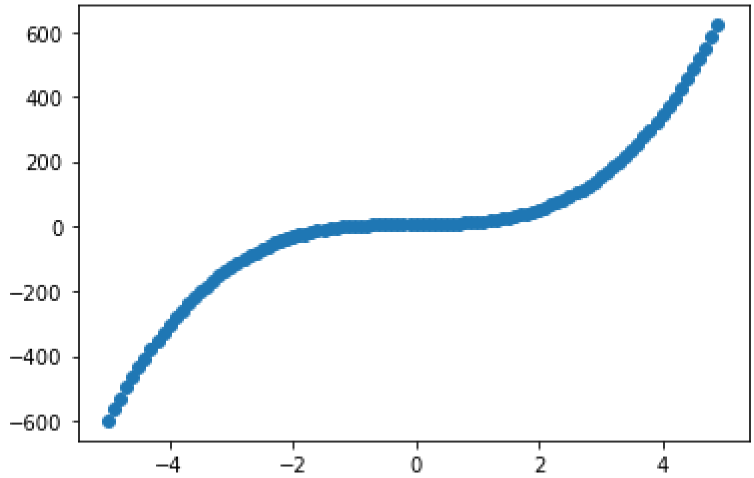


In [3]:

```
x = np.arange(-5.0, 5.0, 0.1)
# y = ax^3 + bx^2 + cx + d
y = 5 * x ** 3 + 1 * x **2 + 1 * x + 5

plt.scatter(x, y)
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x282fb573bb0>
```

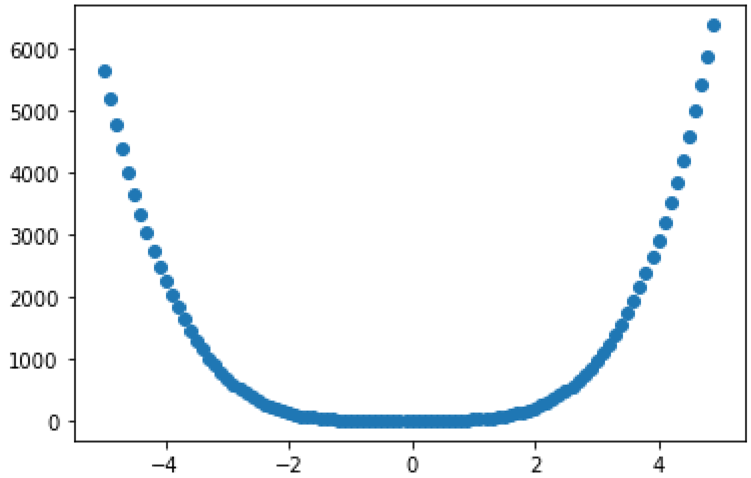


In [4]:

```
x = np.arange(-5.0, 5.0, 0.1)
# y = ax^3 + bx^2 + cx + d
y = 10*x**4 + 5 * x ** 3 + 1 * x **2 + 1 * x + 5

plt.scatter(x, y)
```

Out[4]:

```
<matplotlib.collections.PathCollection at 0x282fb5cda90>
```



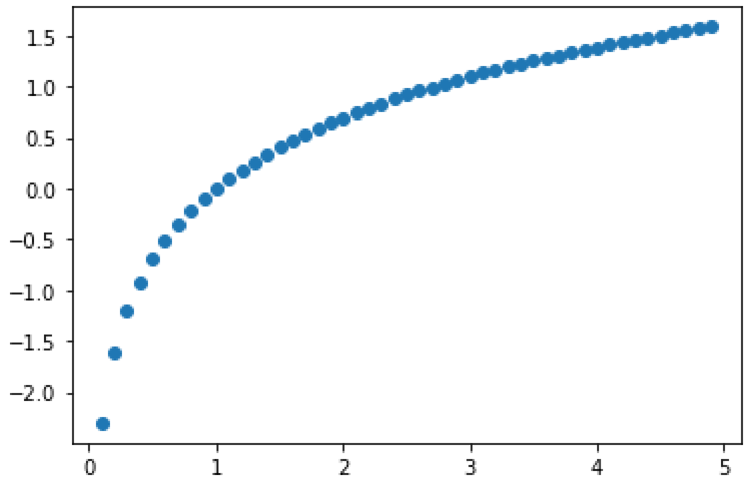
In [5]:

```
x = np.arange(-5.0, 5.0, 0.1)
# y = ax^3 + bx^2 + cx + d
y = np.log(x)

plt.scatter(x, y)
```

```
<ipython-input-5-5d10ce4dcea9>:3: RuntimeWarning: invalid value encountered
in log
  y = np.log(x)
```

Out[5]:

```
<matplotlib.collections.PathCollection at 0x282fb61cf40>
```

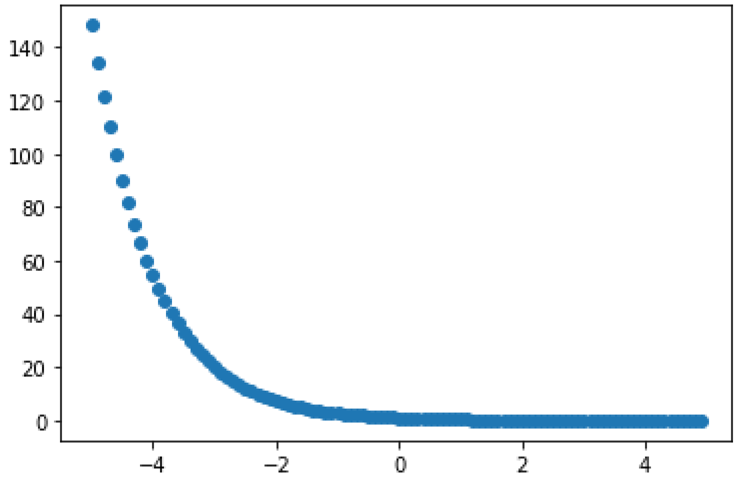


In [6]:

```
x = np.arange(-5.0, 5.0, 0.1)
# y = ax^3 + bx^2 + cx + d
y = 1/np.exp(x)

plt.scatter(x, y)
```

Out[6]:

```
<matplotlib.collections.PathCollection at 0x282fb683250>
```



Polynomial Regression with one variable

Step1: Define Business Use Case

Our Use case is to predict the CO2Emissions of a person based on few features

In [7]:

```
co2 = pd.read_csv('https://raw.githubusercontent.com/AP-State-Skill-Development-Corporation
```

Step2: Data Exploration

In [8]:

```
co2.head()
```

Out[8]:

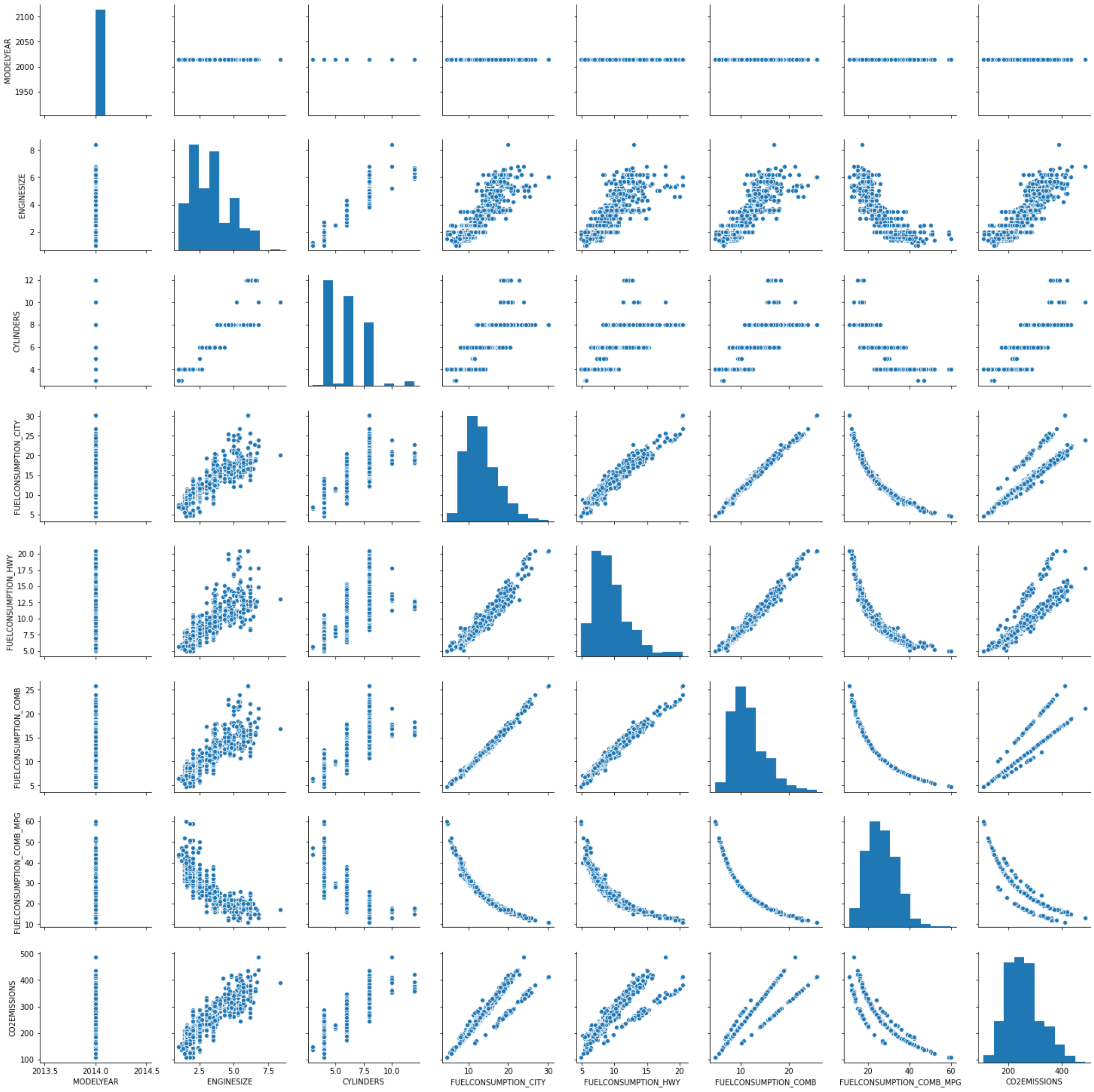
	MODELYEAR	MAKE	MODEL	VEHICLECLASS	ENGINESIZE	CYLINDERS	TRANSMISSION
0	2014	ACURA	ILX	COMPACT	2.0	4	AS5
1	2014	ACURA	ILX	COMPACT	2.4	4	M6
2	2014	ACURA	ILX HYBRID	COMPACT	1.5	4	AV7
3	2014	ACURA	MDX 4WD	SUV - SMALL	3.5	6	AS6
4	2014	ACURA	RDX AWD	SUV - SMALL	3.5	6	AS6

In [9]:

```
import seaborn as sns

sns.pairplot(co2)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x1d7f56222e0>
```

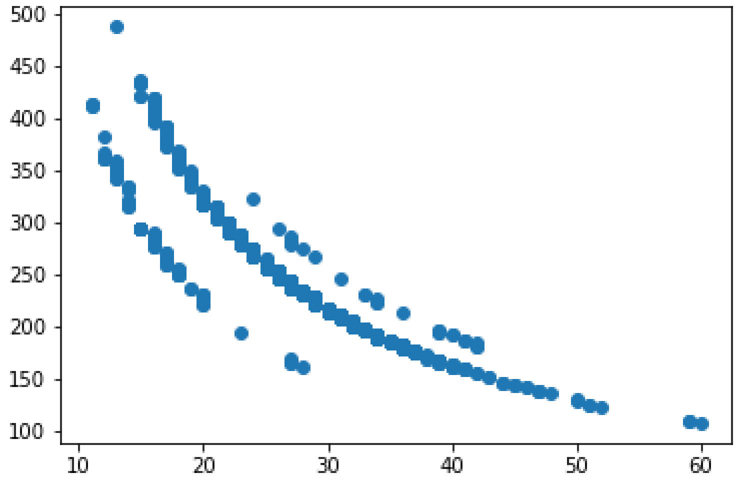


In [10]:

```
X = co2['FUELCONSUMPTION_COMB_MPG'].values.reshape(-1, 1)
Y = co2['CO2EMISSIONS']

plt.scatter(X,Y)
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0x282fb5ec6a0>
```



$$Y = ax^4 + bx^3 + cx^2 + dx + e$$

$$m_1 = x^4$$
$$m_2 = x^3$$
$$m_3 = x^2$$

$$Y = am_1 + bm_2 + cm_3 + dm_4 + e$$

$$Y = ax^2 + bx + c$$

1. Transform our data non linear equation
2. Linear Regression with multiple variables

In [11]:

```
from sklearn.preprocessing import PolynomialFeatures
```

In [12]:

```
poly4 = PolynomialFeatures(degree=4)
poly2 = PolynomialFeatures(degree=2)
```

In [13]:

```
np.exp(x)
```

Out[13]:

```
array([6.73794700e-03, 7.44658307e-03, 8.22974705e-03, 9.09527710e-03,
       1.00518357e-02, 1.11089965e-02, 1.22773399e-02, 1.35685590e-02,
       1.49955768e-02, 1.65726754e-02, 1.83156389e-02, 2.02419114e-02,
       2.23707719e-02, 2.47235265e-02, 2.73237224e-02, 3.01973834e-02,
       3.33732700e-02, 3.68831674e-02, 4.07622040e-02, 4.50492024e-02,
       4.97870684e-02, 5.50232201e-02, 6.08100626e-02, 6.72055127e-02,
       7.42735782e-02, 8.20849986e-02, 9.07179533e-02, 1.00258844e-01,
       1.10803158e-01, 1.22456428e-01, 1.35335283e-01, 1.49568619e-01,
       1.65298888e-01, 1.82683524e-01, 2.01896518e-01, 2.23130160e-01,
       2.46596964e-01, 2.72531793e-01, 3.01194212e-01, 3.32871084e-01,
       3.67879441e-01, 4.06569660e-01, 4.49328964e-01, 4.96585304e-01,
       5.48811636e-01, 6.06530660e-01, 6.70320046e-01, 7.40818221e-01,
       8.18730753e-01, 9.04837418e-01, 1.00000000e+00, 1.10517092e+00,
       1.22140276e+00, 1.34985881e+00, 1.49182470e+00, 1.64872127e+00,
       1.82211880e+00, 2.01375271e+00, 2.22554093e+00, 2.45960311e+00,
       2.71828183e+00, 3.00416602e+00, 3.32011692e+00, 3.66929667e+00,
       4.05519997e+00, 4.48168907e+00, 4.95303242e+00, 5.47394739e+00,
       6.04964746e+00, 6.68589444e+00, 7.38905610e+00, 8.16616991e+00,
       9.02501350e+00, 9.97418245e+00, 1.10231764e+01, 1.21824940e+01,
       1.34637380e+01, 1.48797317e+01, 1.64446468e+01, 1.81741454e+01,
       2.00855369e+01, 2.21979513e+01, 2.45325302e+01, 2.71126389e+01,
       2.99641000e+01, 3.31154520e+01, 3.65982344e+01, 4.04473044e+01,
       4.47011845e+01, 4.94024491e+01, 5.45981500e+01, 6.03402876e+01,
       6.66863310e+01, 7.36997937e+01, 8.14508687e+01, 9.00171313e+01,
       9.94843156e+01, 1.09947172e+02, 1.21510418e+02, 1.34289780e+02])
```

In [14]:

```
X
```

Out[14]:

```
array([[33],
       [29],
       [48],
       ...,
       [24],
       [25],
       [22]], dtype=int64)
```

In [15]:

```
33 ** 4
```

Out[15]:

```
1185921
```

In [16]:

```
x_p4 = poly4.fit_transform(X)

x_p4
```

Out[16]:

```
array([[1.000000e+00, 3.300000e+01, 1.089000e+03, 3.593700e+04,
        1.185921e+06],
       [1.000000e+00, 2.900000e+01, 8.410000e+02, 2.438900e+04,
        7.072810e+05],
       [1.000000e+00, 4.800000e+01, 2.304000e+03, 1.105920e+05,
        5.308416e+06],
       ...,
       [1.000000e+00, 2.400000e+01, 5.760000e+02, 1.382400e+04,
        3.317760e+05],
       [1.000000e+00, 2.500000e+01, 6.250000e+02, 1.562500e+04,
        3.906250e+05],
       [1.000000e+00, 2.200000e+01, 4.840000e+02, 1.064800e+04,
        2.342560e+05]])
```

In [17]:

```
from sklearn.model_selection import train_test_split

x_tr, x_tt, y_tr, y_tt = train_test_split(x_p4, Y, test_size =  0.3, random_state = 42)
```

In [18]:

```
from sklearn.linear_model import LinearRegression

model  = LinearRegression()

model.fit(x_tr, y_tr)
```

Out[18]:

```
LinearRegression()
```

In [19]:

```
model.coef_
```

Out[19]:

```
array([ 0.00000000e+00,  3.17533326e+01, -2.23472682e+00,  4.95447565e-02,
       -3.65025726e-04])
```

$$Y = 0.0x^4 + 3.17x^3 - 2.234x^2 + 4.95x - 3.650$$

In [20]:

```
y_pred = model.predict(x_tt)

model.score(x_tt, y_tt)
```

Out[20]:

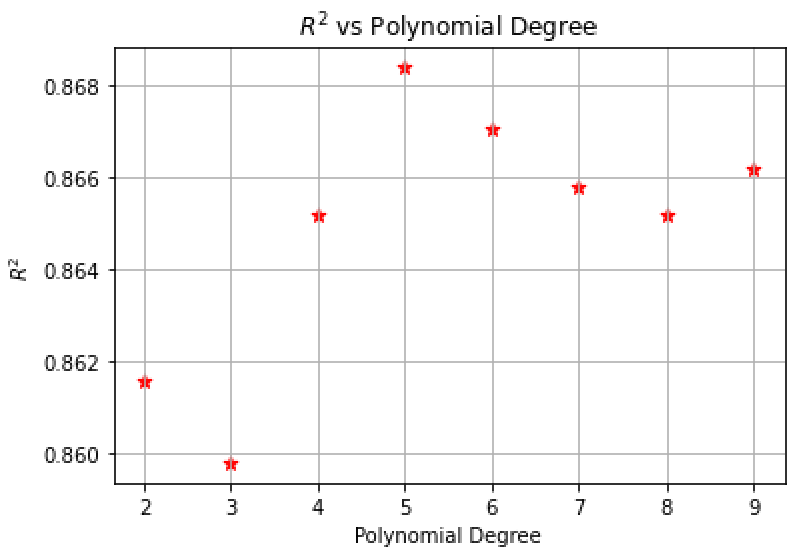
```
0.8652099608757728
```

In [21]:

```
score = []
for i in range(2, 10):
    poly = PolynomialFeatures(degree=i)
    x_poly = poly.fit_transform(X)
    x_tr, x_tt, y_tr, y_tt = train_test_split(x_poly, Y, test_size =  0.3, random_state = 4
    model  = LinearRegression()
    model.fit(x_tr, y_tr)
    y_pred = model.predict(x_tt)
    score.append(model.score(x_tt, y_tt))
```

In [22]:

```
plt.scatter(range(2, 10), score, marker = '*', s = 50, c = 'r')
plt.grid()
plt.title("$R^2$ vs Polynomial Degree")
plt.xlabel("Polynomial Degree")
plt.ylabel("$R^2$")
plt.show()
```



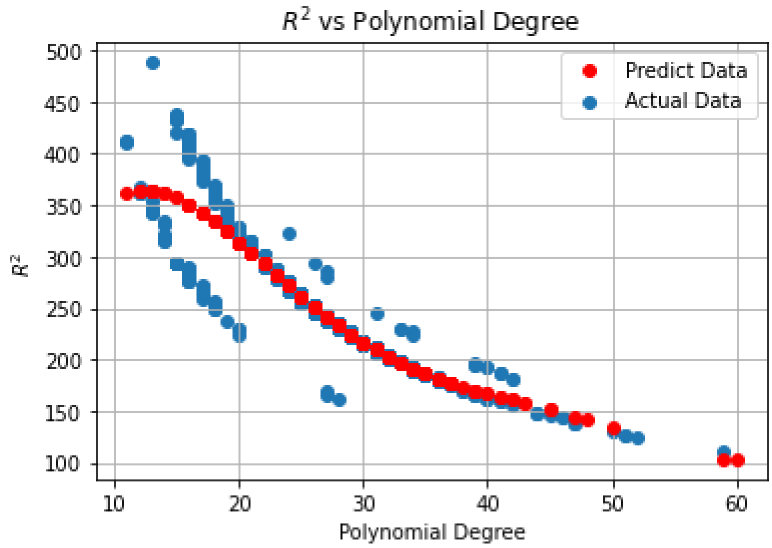
From the above graph we can observe at degree 5 we are getting the best r^2 score

In [25]:

```
poly = PolynomialFeatures(degree=5)

x_tr, x_tt, y_tr, y_tt = train_test_split(X, Y, test_size =  0.3, random_state = 42)

xtr_poly = poly.fit_transform(x_tr)
xtt_poly = poly.fit_transform(x_tt)

model  = LinearRegression()

model.fit(xtr_poly, y_tr)

y_pred = model.predict(xtt_poly)
```

In [27]:

```
plt.scatter(x_tr, y_tr, label = 'Actual Data')
plt.plot(x_tt, y_pred, 'ro', label = 'Predict Data')
plt.grid()
plt.legend()
plt.title("$R^2$ vs Polynomial Degree")
plt.xlabel("Polynomial Degree")
plt.ylabel("$R^2$")
plt.show()
```



Polynomial Regression with Multiple Variables

In [28]:

```
from sklearn.datasets import load_boston
```

In [29]:

```
data = load_boston()

data.keys()
```

Out[29]:

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

In [30]:

```
data.data
```

Out[30]:

```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

In [31]:

```
print(data.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value
(attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,0
00 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds rive
r; 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

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (http
s://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Car
negie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnost
ics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers
 that address regression
problems.

.. topic:: References
```

```
   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influenti
al Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning.
In Proceedings on the Tenth International Conference of Machine Learning,
 236-243, University of Massachusetts, Amherst. Morgan Kaufmann.
```

In [32]:

```
data.feature_names
```

Out[32]:

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

In [39]:

```
df = pd.DataFrame(data.data, columns = data.feature_names)

df.head()
```

Out[39]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]
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	[illegible]

In [40]:

```
df['MEDV'] = data.target
```

In [41]:

```
df.shape
```

Out[41]:

```
(506, 14)
```

In [42]:

```
df.corr()
```

Out[42]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996
MEDV	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929

In [43]:

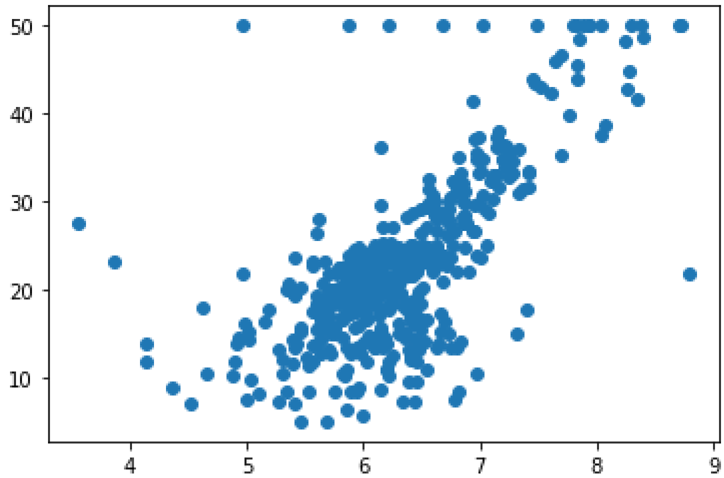
```
x = df[['RM', 'LSTAT']]
y = df['MEDV']
```

In [44]:

```
plt.scatter(df['RM'],y)
```

Out[44]:

<matplotlib.collections.PathCollection at 0x28286e50a90>

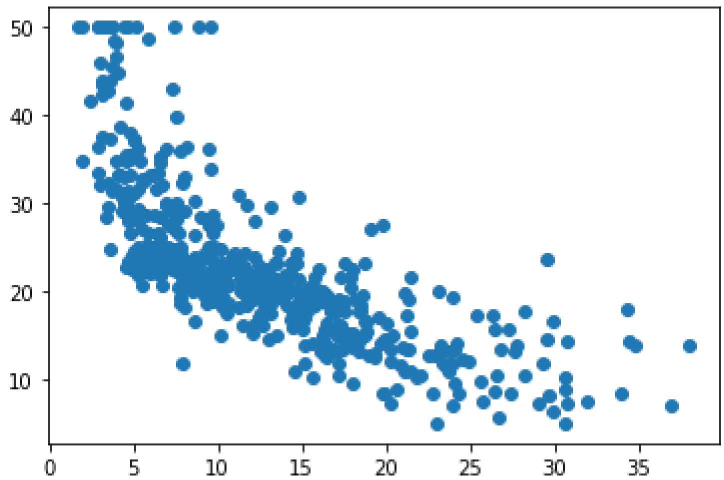


In [45]:

```
plt.scatter(df['LSTAT'], y)
```

Out[45]:

<matplotlib.collections.PathCollection at 0x28286f39af0>



In [54]:

```
poly = PolynomialFeatures(degree = 2)

x_poly = poly.fit_transform(x)

x_poly
```

Out[54]:

```
array([[ 1.      ,  6.575   ,  4.98    , 43.230625, 32.7435  , 24.8004  ],
       [ 1.      ,  6.421   ,  9.14    , 41.229241, 58.68794 , 83.5396  ],
       [ 1.      ,  7.185   ,  4.03    , 51.624225, 28.95555 , 16.2409  ],
       ...,
       [ 1.      ,  6.976   ,  5.64    , 48.664576, 39.34464 , 31.8096  ],
       [ 1.      ,  6.794   ,  6.48    , 46.158436, 44.02512 , 41.9904  ],
       [ 1.      ,  6.03    ,  7.88    , 36.3609  , 47.5164  , 62.0944  ]])
```

In [55]:

```
from sklearn.model_selection import train_test_split

x_tr, x_tt, y_tr, y_tt = train_test_split(x_poly, y, test_size =  0.3, random_state = 42)
```

In [56]:

```
model_poly = LinearRegression()


model_poly.fit(x_tr, y_tr)
```

Out[56]:

```
LinearRegression()
```

In [57]:

```
x_tt.shape, x_tr.shape
```

Out[57]:

```
((152, 6), (354, 6))
```

In [58]:

```
y_test = model_poly.predict(x_tt)
```

In [59]:

```
model_poly.score(x_tt, y_tt)
```

Out[59]:

```
0.750780855092862
```

In [60]:

```
model_poly.score(x_tr, y_tr)
```

Out[60]:

```
0.7553107581437002
```