A demonstration of overfitting with a simple polynomial fit

In [3]:
```
# import libraries that are required
import numpy as np
import sys
import matplotlib.pyplot as plt
import os
```

In [4]:
```
# change the operating system
os.chdir('/Users/karimaidrissi/Desktop/DSSA 5104 DL/week13')
# Load data from file into N by 2 numpy array
filename = "abalone.data.txt"
X = np.loadtxt( filename, delimiter=',', usecols=(2,4) )
```

In [5]:
```
diameter = X[:,0] # Put first column into N by 1 array called diameter
mass_observed = X[:,1] # and second column into N by 1 array called mass
_observed
mass_observed #diameter
```

Out[5]:
```
array([0.514 , 0.2255, 0.677 , ..., 1.176 , 1.0945, 1.9485])
```

Random Simple of 10 Data Points

In [6]:
```
# using random.choice to generate 10 random indices out of a total
# of diameter.size indices. Then create smaller sample vectors that
# are just 10 in size from the complete dataset
np.random.seed(10)
print('Total dataset has ',diameter.size,' data points')
sample_size = 10
print('Will use a random sample of only ',sample_size,' data points')
sample_indices = np.random.choice(diameter.size,sample_size)
diameter_sample = diameter[sample_indices]
mass_observed_sample = mass_observed[sample_indices]
```

```
Total dataset has  4177  data points
Will use a random sample of only  10  data points
```

First Degree of Polynomial

In [7]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 1 is a linear model y = a x + b
degree_of_polynomial_to_fit = 1
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

Out[7]:
```
array([0.14672203, 0.16038732, 0.17405261, 0.1877179 , 0.20138318,
       0.21504847, 0.22871376, 0.24237904, 0.25604433, 0.26970962,
       0.28337491, 0.29704019, 0.31070548, 0.32437077, 0.33803605,
       0.35170134, 0.36536663, 0.37903192, 0.3926972 , 0.40636249,
       0.42002778, 0.43369306, 0.44735835, 0.46102364, 0.47468893,
       0.48835421, 0.5020195 , 0.51568479, 0.52935007, 0.54301536,
       0.55668065, 0.57034593, 0.58401122, 0.59767651, 0.6113418 ,
       0.62500708, 0.63867237, 0.65233766, 0.66600294, 0.67966823,
       0.69333352, 0.70699881, 0.72066409, 0.73432938, 0.74799467,
       0.76165995, 0.77532524, 0.78899053, 0.80265582, 0.8163211 ,
       0.82998639, 0.84365168, 0.85731696, 0.87098225, 0.88464754,
       0.89831282, 0.91197811, 0.9256434 , 0.93930869, 0.95297397,
       0.96663926, 0.98030455, 0.99396983, 1.00763512, 1.02130041,
       1.0349657 , 1.04863098, 1.06229627, 1.07596156, 1.08962684,
       1.10329213, 1.11695742, 1.13062271, 1.14428799, 1.15795328,
       1.17161857, 1.18528385, 1.19894914, 1.21261443, 1.22627971,
       1.239945  , 1.25361029, 1.26727558, 1.28094086, 1.29460615,
       1.30827144, 1.32193672, 1.33560201, 1.3492673 , 1.36293259,
       1.37659787, 1.39026316, 1.40392845, 1.41759373, 1.43125902,
       1.44492431, 1.4585896 , 1.47225488, 1.48592017, 1.49958546])
```

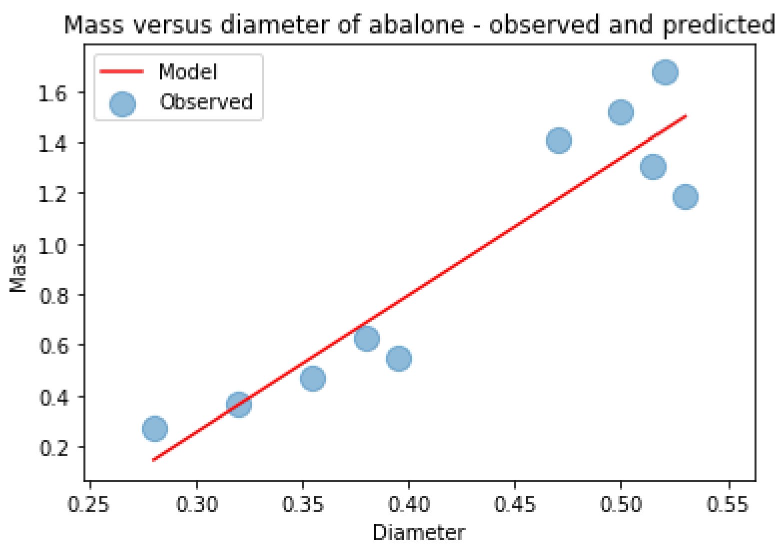
In [8]:

```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  1 st order
y = 

5.411 x - 1.368
```

Second Degree of Polynomial

In [9]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 2 is a quadratic model y = a x^2 + b x +
c
degree_of_polynomial_to_fit = 2
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

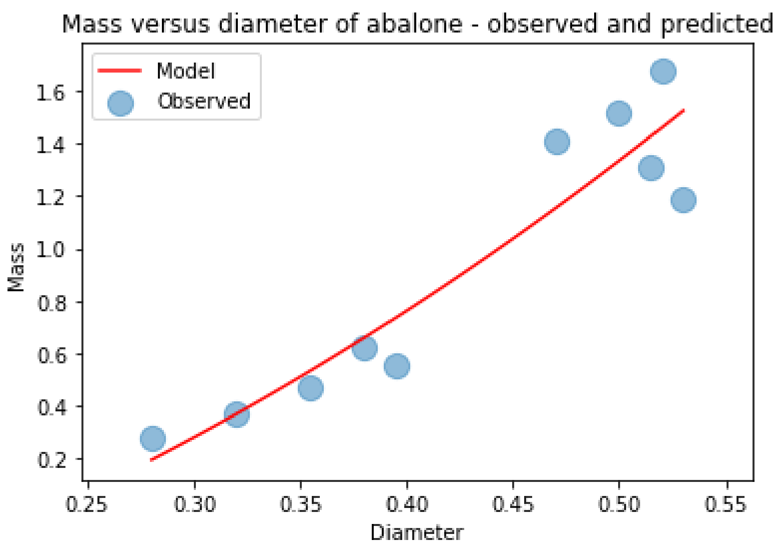
Out[9]:
```
array([0.19437136, 0.20497502, 0.21563651, 0.22635585, 0.23713303,
       0.24796805, 0.25886091, 0.26981161, 0.28082015, 0.29188654,
       0.30301076, 0.31419283, 0.32543274, 0.33673049, 0.34808608,
       0.35949951, 0.37097078, 0.3824999 , 0.39408685, 0.40573165,
       0.41743429, 0.42919477, 0.44101309, 0.45288925, 0.46482326,
       0.4768151 , 0.48886479, 0.50097231, 0.51313768, 0.52536089,
       0.53764194, 0.54998083, 0.56237757, 0.57483214, 0.58734456,
       0.59991482, 0.61254292, 0.62522885, 0.63797264, 0.65077426,
       0.66363372, 0.67655103, 0.68952617, 0.70255916, 0.71564999,
       0.72879866, 0.74200517, 0.75526952, 0.76859172, 0.78197175,
       0.79540963, 0.80890535, 0.8224589 , 0.8360703 , 0.84973955,
       0.86346663, 0.87725155, 0.89109432, 0.90499492, 0.91895337,
       0.93296966, 0.94704379, 0.96117576, 0.97536558, 0.98961323,
       1.00391872, 1.01828206, 1.03270324, 1.04718226, 1.06171912,
       1.07631382, 1.09096636, 1.10567675, 1.12044497, 1.13527104,
       1.15015495, 1.1650967 , 1.18009629, 1.19515372, 1.21026899,
       1.22544211, 1.24067306, 1.25596186, 1.2713085 , 1.28671297,
       1.3021753 , 1.31769546, 1.33327346, 1.3489093 , 1.36460299,
       1.38035452, 1.39616389, 1.41203109, 1.42795615, 1.44393904,
       1.45997977, 1.47607834, 1.49223476, 1.50844902, 1.52472112])
```

In [10]:
```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  2 nd order
y = 
       2
4.535 x + 1.648 x - 0.6226
```

Third Degree of Polynomial

In [11]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 3 is a cubic model y = a x^3 + b x^2 + c
x + d
degree_of_polynomial_to_fit = 3
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

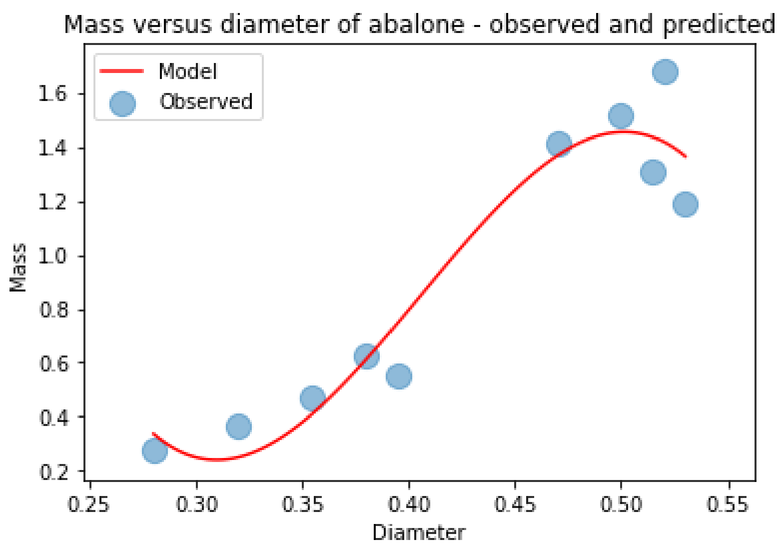
Out[11]:
```
array([0.3372889 , 0.32094857, 0.30622981, 0.29309932, 0.28152382,
       0.27147002, 0.26290462, 0.25579435, 0.2501059 , 0.24580599,
       0.24286133, 0.24123863, 0.2409046 , 0.24182595, 0.24396939,
       0.24730163, 0.25178938, 0.25739936, 0.26409826, 0.27185281,
       0.28062972, 0.29039569, 0.30111743, 0.31276165, 0.32529507,
       0.3386844 , 0.35289634, 0.36789761, 0.38365491, 0.40013496,
       0.41730447, 0.43513015, 0.4535787 , 0.47261684, 0.49221128,
       0.51232873, 0.5329359 , 0.5539995 , 0.57548624, 0.59736284,
       0.61959599, 0.64215241, 0.66499882, 0.68810192, 0.71142842,
       0.73494503, 0.75861847, 0.78241544, 0.80630266, 0.83024683,
       0.85421467, 0.87817288, 0.90208818, 0.92592727, 0.94965687,
       0.97324369, 0.99665444, 1.01985582, 1.04281456, 1.06549735,
       1.08787091, 1.10990195, 1.13155718, 1.15280331, 1.17360706,
       1.19393512, 1.21375421, 1.23303105, 1.25173234, 1.26982479,
       1.28727512, 1.30405003, 1.32011623, 1.33544044, 1.34998936,
       1.36372971, 1.37662819, 1.38865152, 1.39976641, 1.40993956,
       1.41913769, 1.42732751, 1.43447572, 1.44054905, 1.44551419,
       1.44933786, 1.45198677, 1.45342763, 1.45362716, 1.45255205,
       1.45016902, 1.44644479, 1.44134606, 1.43483954, 1.42689194,
       1.41746998, 1.40654036, 1.39406979, 1.38002499, 1.36437266])
```

In [12]:
```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  3 th order
y = 
         3          2
-344.5 x + 419.2 x - 160.5 x + 19.98
```

Random Simple of 100 Data Points

In [13]:
```
# using random.choice to generate 100 random indices out of a total
# of diameter.size indices. Then create smaller sample vectors that
# are just 100 in size from the complete dataset
np.random.seed(100)
print('Total dataset has ',diameter.size,' data points')
sample_size = 100
print('Will use a random sample of only ',sample_size,' data points')
sample_indices = np.random.choice(diameter.size,sample_size)
diameter_sample = diameter[sample_indices]
mass_observed_sample = mass_observed[sample_indices]
```

```
Total dataset has  4177  data points
Will use a random sample of only  100  data points
```

First Degree of Polynomial using 100 data points

In [14]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 1 is a linear model y = a x + b
degree_of_polynomial_to_fit = 1
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

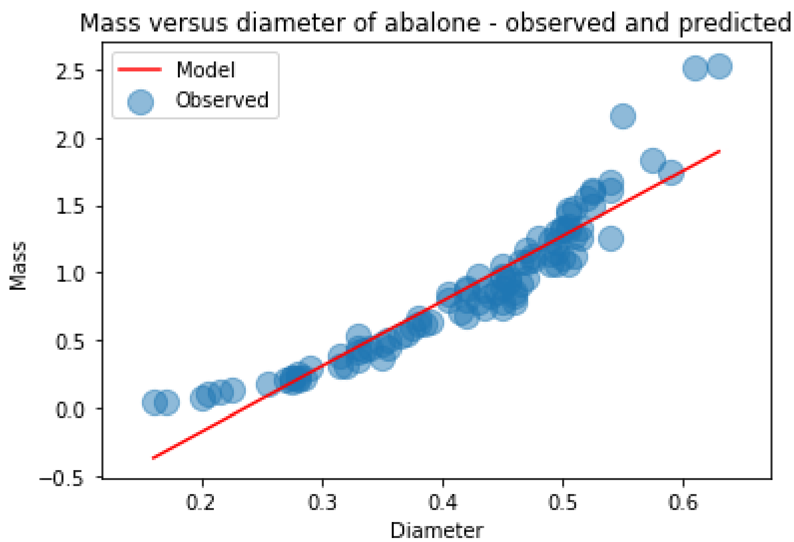
Out[14]:
```
array([-3.64703251e-01, -3.41867152e-01, -3.19031052e-01, -2.96194952e-
01,
       -2.73358852e-01, -2.50522752e-01, -2.27686653e-01, -2.04850553e-
01,
       -1.82014453e-01, -1.59178353e-01, -1.36342254e-01, -1.13506154e-
01,
       -9.06700540e-02, -6.78339542e-02, -4.49978544e-02, -2.21617546e-
02,
        6.74345206e-04,  2.35104450e-02,  4.63465448e-02,  6.91826446e-
02,
        9.20187444e-02,  1.14854844e-01,  1.37690944e-01,  1.60527044e-
01,
        1.83363144e-01,  2.06199243e-01,  2.29035343e-01,  2.51871443e-
01,
        2.74707543e-01,  2.97543642e-01,  3.20379742e-01,  3.43215842e-
01,
        3.66051942e-01,  3.88888042e-01,  4.11724141e-01,  4.34560241e-
01,
        4.57396341e-01,  4.80232441e-01,  5.03068541e-01,  5.25904640e-
01,
        5.48740740e-01,  5.71576840e-01,  5.94412940e-01,  6.17249040e-
01,
        6.40085139e-01,  6.62921239e-01,  6.85757339e-01,  7.08593439e-
01,
        7.31429538e-01,  7.54265638e-01,  7.77101738e-01,  7.99937838e-
01,
        8.22773938e-01,  8.45610037e-01,  8.68446137e-01,  8.91282237e-
01,
        9.14118337e-01,  9.36954437e-01,  9.59790536e-01,  9.82626636e-
01,
        1.00546274e+00,  1.02829884e+00,  1.05113494e+00,  1.07397104e+
00,
        1.09680714e+00,  1.11964323e+00,  1.14247933e+00,  1.16531543e+
00,
        1.18815153e+00,  1.21098763e+00,  1.23382373e+00,  1.25665983e+
00,
        1.27949593e+00,  1.30233203e+00,  1.32516813e+00,  1.34800423e+
00,
        1.37084033e+00,  1.39367643e+00,  1.41651253e+00,  1.43934863e+
00,
        1.46218473e+00,  1.48502083e+00,  1.50785693e+00,  1.53069303e+
00,
        1.55352913e+00,  1.57636523e+00,  1.59920133e+00,  1.62203743e+
00,
        1.64487353e+00,  1.66770963e+00,  1.69054573e+00,  1.71338183e+
00,
        1.73621793e+00,  1.75905403e+00,  1.78189013e+00,  1.80472623e+
00,
        1.82756233e+00,  1.85039843e+00,  1.87323453e+00,  1.89607063e+
00])
```

In [15]:
```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  1 st order
y = 

4.81 x - 1.134
```

Second Degree of Polynomial using 100 data points

In [16]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 2 is a quadratic model y = a x^2 + b x +
c
degree_of_polynomial_to_fit = 2
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

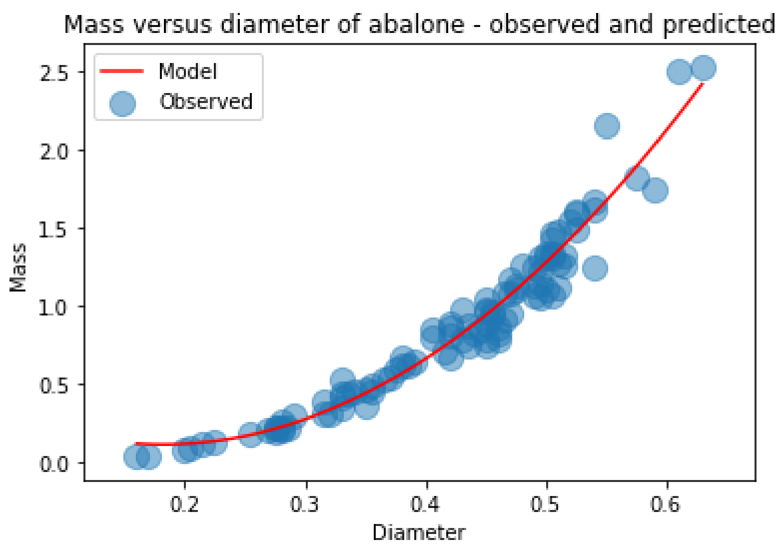
Out[16]:
```
array([0.12334098, 0.12120802, 0.11959256, 0.11849459, 0.11791413,
       0.11785116, 0.1183057 , 0.11927773, 0.12076726, 0.12277428,
       0.12529881, 0.12834084, 0.13190036, 0.13597738, 0.14057191,
       0.14568393, 0.15131344, 0.15746046, 0.16412498, 0.17130699,
       0.17900651, 0.18722352, 0.19595803, 0.20521004, 0.21497955,
       0.22526656, 0.23607106, 0.24739307, 0.25923257, 0.27158957,
       0.28446407, 0.29785607, 0.31176557, 0.32619256, 0.34113706,
       0.35659905, 0.37257855, 0.38907554, 0.40609003, 0.42362201,
       0.4416715 , 0.46023849, 0.47932297, 0.49892496, 0.51904444,
       0.53968142, 0.5608359 , 0.58250787, 0.60469735, 0.62740433,
       0.6506288 , 0.67437077, 0.69863024, 0.72340721, 0.74870168,
       0.77451365, 0.80084312, 0.82769008, 0.85505454, 0.88293651,
       0.91133597, 0.94025293, 0.96968738, 0.99963934, 1.0301088 ,
       1.06109575, 1.0926002 , 1.12462215, 1.1571616 , 1.19021855,
       1.223793  , 1.25788495, 1.29249439, 1.32762133, 1.36326578,
       1.39942772, 1.43610716, 1.47330409, 1.51101853, 1.54925047,
       1.5879999 , 1.62726683, 1.66705126, 1.70735319, 1.74817262,
       1.78950955, 1.83136398, 1.8737359 , 1.91662532, 1.96003225,
       2.00395667, 2.04839859, 2.09335801, 2.13883492, 2.18482934,
       2.23134125, 2.27837066, 2.32591758, 2.37398199, 2.4225639 ])
```

In [17]:
```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  2 nd order
y = 
       2
11.48 x - 4.177 x + 0.4978
```

Third Degree of Polynomial using 100 data points

In [18]:
```
# Set the degree of the polynomial for our model
# degree_of_polynomial_to_fit = 3 is a cubic model y = a x^3 + b x^2 + c
x + d
degree_of_polynomial_to_fit = 3
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the
data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_
of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sa
mple.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter va
lues I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

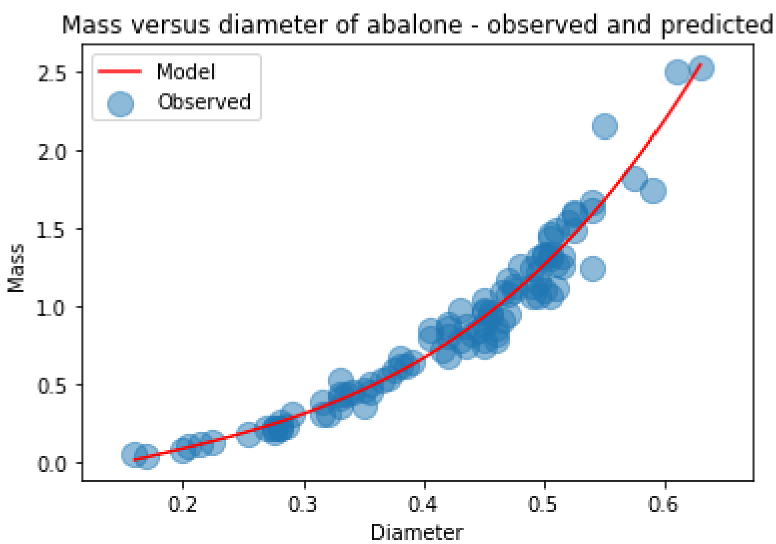
Out[18]:
```
array([0.0164899 , 0.02493162, 0.0333822 , 0.04185218, 0.05035208,
       0.05889246, 0.06748384, 0.07613676, 0.08486176, 0.09366938,
       0.10257015, 0.11157461, 0.12069329, 0.12993675, 0.1393155 ,
       0.14884009, 0.15852105, 0.16836892, 0.17839424, 0.18860755,
       0.19901938, 0.20964027, 0.22048075, 0.23155137, 0.24286265,
       0.25442514, 0.26624938, 0.27834589, 0.29072522, 0.30339791,
       0.31637448, 0.32966549, 0.34328146, 0.35723293, 0.37153043,
       0.38618452, 0.40120571, 0.41660456, 0.43239158, 0.44857734,
       0.46517235, 0.48218716, 0.4996323 , 0.51751831, 0.53585573,
       0.55465509, 0.57392694, 0.5936818 , 0.61393022, 0.63468272,
       0.65594986, 0.67774216, 0.70007017, 0.72294441, 0.74637543,
       0.77037376, 0.79494994, 0.82011451, 0.84587801, 0.87225096,
       0.89924391, 0.92686739, 0.95513195, 0.98404811, 1.01362642,
       1.04387741, 1.07481162, 1.10643958, 1.13877184, 1.17181893,
       1.20559138, 1.24009973, 1.27535453, 1.3113663 , 1.34814559,
       1.38570292, 1.42404885, 1.4631939 , 1.50314861, 1.54392351,
       1.58552916, 1.62797607, 1.6712748 , 1.71543587, 1.76046982,
       1.80638719, 1.85319852, 1.90091434, 1.94954519, 1.99910161,
       2.04959413, 2.1010333 , 2.15342964, 2.20679369, 2.261136  ,
       2.3164671 , 2.37279752, 2.4301378 , 2.48849849, 2.5478901 ])
```

In [19]:
```
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  3 th order
y = 
       3         2
16.41 x - 7.915 x + 3.05 x - 0.3362
```

- the model fits more the observed data by using the third degree of polynomial with 100 data points

Using Fit Degree of 16 Polynomial

In [20]:
```
# Set the degree of the polynomial for our model
degree_of_polynomial_to_fit = 16
# Fit the polynomial model using the numpy polyfit function
# It returns the coefficients of an n-th degree polynomial that fits the data given to it
# It also returns the best fit that minimises the squared error
coefficients = np.polyfit(diameter_sample, mass_observed_sample, degree_of_polynomial_to_fit)
# I want to plot the model as a line plot so I generate a large
# set of ordered diameter values for which I can find corresponding
# model mass predictions using the polynomial function I just fitted
diameter_sample_for_plot = np.linspace(diameter_sample.min(),diameter_sample.max(),100)
# Use the fitted model coefficients to create the model function
model_for_mass = np.poly1d(coefficients)
model_for_mass
# Make predictions for mass using the model function and the diameter values I generated earlier
mass_prediction = model_for_mass(diameter_sample_for_plot)
mass_prediction
```

```
/Users/karimaidrissi/opt/anaconda3/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3331: RankWarning: Polyfit may be poorly conditioned
  exec(code_obj, self.user_global_ns, self.user_ns)
```

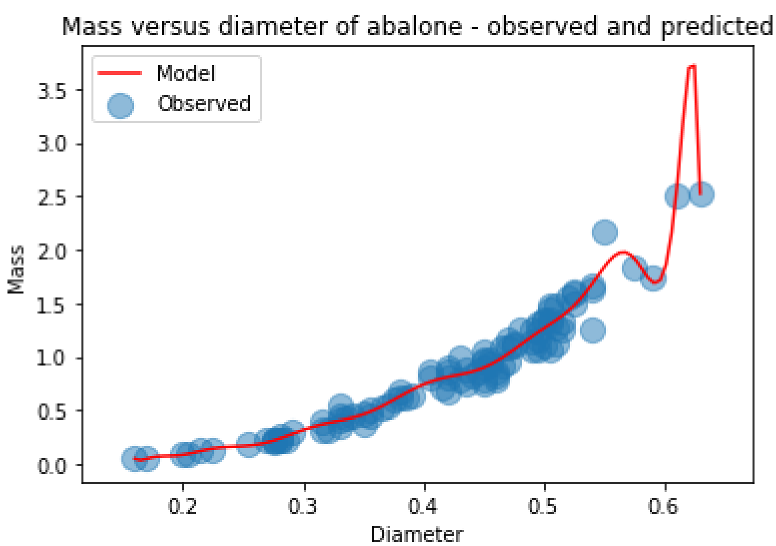
Out[20]:
```
array([0.04864977, 0.03433651, 0.04671377, 0.06110225, 0.06986642,
       0.07337119, 0.07471172, 0.07696194, 0.0820207 , 0.09039776,
       0.10148624, 0.11402343, 0.12655803, 0.13782506, 0.14698741,
       0.15374129, 0.15830492, 0.16132157, 0.16371117, 0.16650269,
       0.17067421, 0.17702071, 0.18606194, 0.19799566, 0.21269593,
       0.22975034, 0.248528  , 0.26826693, 0.28816961, 0.30749614,
       0.32564503, 0.3422149 , 0.3570419 , 0.37021086, 0.38204068,
       0.39304652, 0.40388303, 0.41527583, 0.4279466 , 0.44253913,
       0.45955431, 0.47929748, 0.50184467, 0.52702956, 0.55445267,
       0.58351162, 0.61345114, 0.64342603, 0.67257577, 0.70010065,
       0.7253368 , 0.74781823, 0.7673269 , 0.78392161, 0.7979404 ,
       0.80998279, 0.82086574, 0.83155804, 0.84309927, 0.85650726,
       0.87268597, 0.8923385 , 0.91589768, 0.94347857, 0.97486173,
       1.00951847, 1.04666427, 1.08535089, 1.12459266, 1.16350015,
       1.2014339 , 1.23812497, 1.27378522, 1.30914665, 1.34543997,
       1.38428324, 1.42747798, 1.47671237, 1.53319543, 1.59724149,
       1.66787559, 1.74248944, 1.81668555, 1.88436716, 1.93819654,
       1.97049581, 1.974679  , 1.94720035, 1.88992018, 1.81262855,
       1.73526776, 1.68905025, 1.71519529, 1.85957815, 2.16062912,
       2.62703608, 3.20040262, 3.69674978, 3.71831646, 2.52539409])
```

In [21]:
```python
# Now I can do the plots
plt.scatter(diameter_sample,mass_observed_sample, marker = 'o', s=150, a
lpha = 0.5,label='Observed')
plt.plot(diameter_sample_for_plot,mass_prediction,color='red',label='Mod
el')

#plt.text(0.05,2.2,str(loss))
plt.xlabel('Diameter')
plt.ylabel('Mass')
plt.title('Mass versus diameter of abalone - observed and predicted')
plt.legend(loc='best')
plt.show()

if degree_of_polynomial_to_fit == 1:
    s='st order'
elif degree_of_polynomial_to_fit == 2:
    s='nd order'
else:
    s='th order'

print('Fitted model is ',degree_of_polynomial_to_fit,s)
print('y = ')
print(model_for_mass)
```




```
Fitted model is  16 th order
y = 
            16             15             14             13
-1.981e+12 x  + 1.017e+13 x  - 2.321e+13 x  + 3.054e+13 x
             12            11             10            9             8
 - 2.463e+13 x  + 1.1e+13 x  - 2.918e+11 x  - 3.37e+12 x + 2.703e+12 x
             7             6             5             4             3
 - 1.246e+12 x + 3.946e+11 x - 8.966e+10 x + 1.469e+10 x - 1.699e+09 x
             2
 + 1.318e+08 x - 6.155e+06 x + 1.309e+05
```

- the model fits more the observed data by using the 16 degree of polynomial with 100 data points
- the model is overfitted by increasing the polynomial degree to 16 also there's a very large coeficients associated to our model.