Lecture 2 Regression Techniques

May 10, 2022

1 Regression Workbook

Welcome to our Regression notebook. Today we will learn Basic and Regularized Regression.

1.1 1. Linear Regression

First let's start with importing libraries as usual.

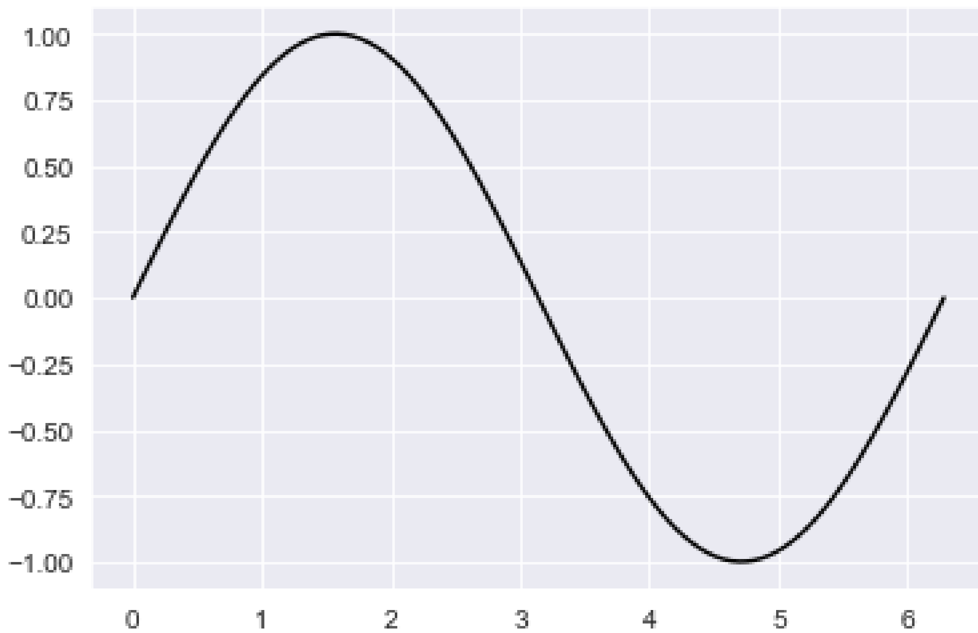
[1]:
```
import numpy as np
import pandas as pd

from matplotlib import pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set_style('darkgrid')
```

Let's create a sine wave.

[2]:
```
x = np.linspace(0, 2*np.pi, 100)
```

[3]:
```
plt.plot(x, np.sin(x),'k')
```

[3]:
```
[<matplotlib.lines.Line2D at 0x138b55400>]
```



Now let's add some noise to the sine wave.

[4]:
```
# noise
np.random.seed(321)
noise = np.random.normal(0, .5, 100)
```

[5]:
```
# target variable
y = np.sin(x) + noise
```

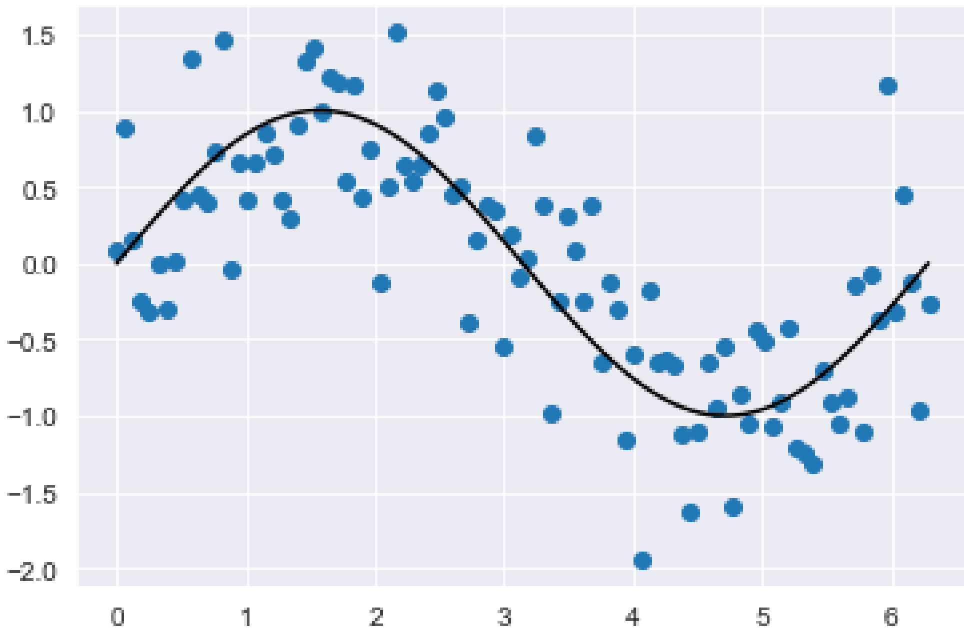
[6]:
```
# Create DataFrame with x and y
df = pd.DataFrame({'x' : x, 'y': y})
df.head()
```

[6]:

	x	y
0	0.000000	0.086260
1	0.063467	0.881165
2	0.126933	0.145261
3	0.190400	-0.252824
4	0.253866	-0.320448

[7]:
```
# Scatterplot of x and y
plt.scatter(df.x, df.y)

# Overlay the sine wave
plt.plot(df.x, np.sin(df.x), color='k', label='Sine wave')
```

```
plt.show()
```



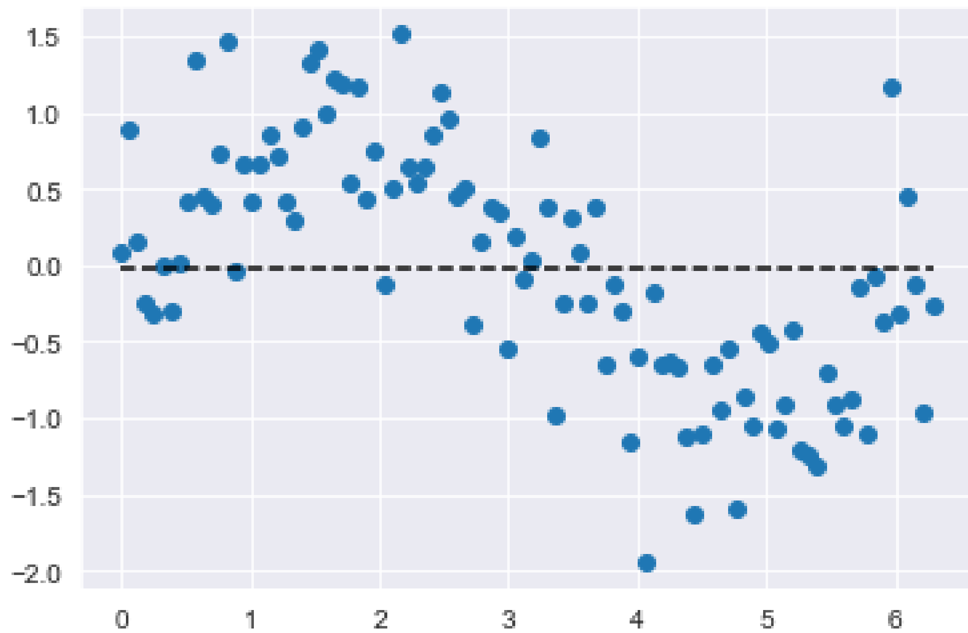
Let's save this as a new dataframe.

1.1.1 1.1 Mean model

Mean model is the most basic prediction model.

[8]:
```
# Build model
pred = np.mean(df.y)
```

[9]:
```
# Scatterplot of x and y
plt.scatter(df.x, df.y)

# Overlay horizontal line for the prediction
plt.plot(df.x, [pred]*len(df.x), 'k--' )
plt.show()
```



What do you think about the mean model?

1.1.2 1.2 Linear Regression Model

With linear regression, we are simply trying to fit the following formula: y = w0x0 + c

[10]:
```
from sklearn.linear_model import LinearRegression
```

[11]:
```
# Initialize instance of linear regression
linear_reg_model = LinearRegression()
```

[12]:
```
# Separate our input features and target variable
features = df.drop('y', axis=1)
target = df.y
```

[13]:
```
# Fit model to the data
linear_reg_model.fit(features, target)
```

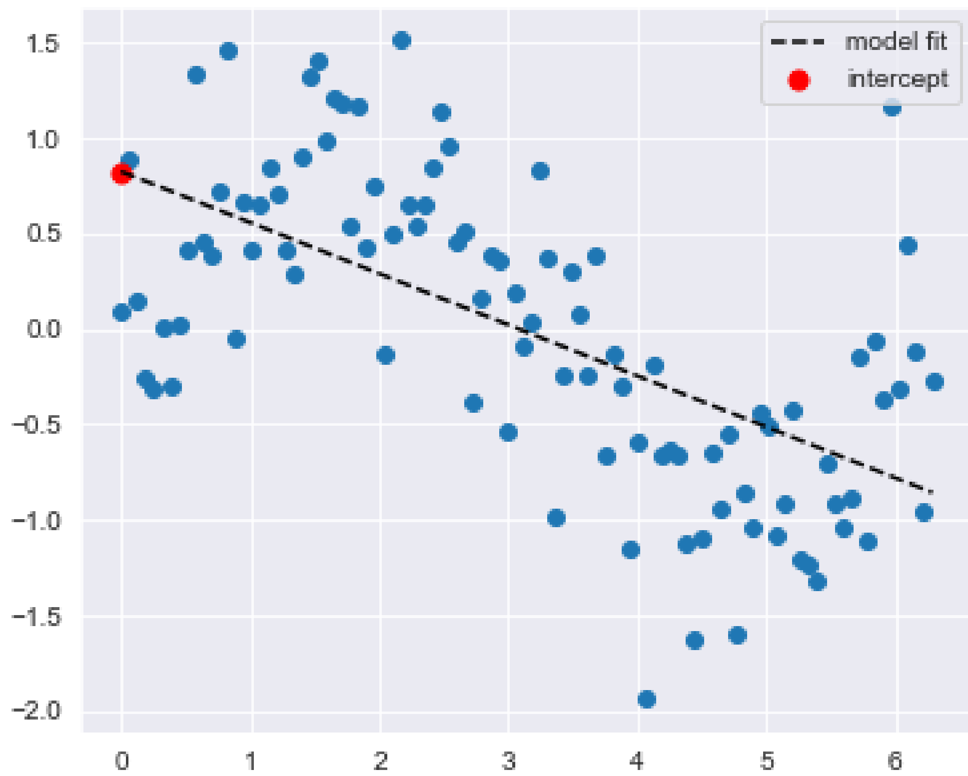
[13]:
```
LinearRegression()
```

[14]:
```
print(linear_reg_model.intercept_)
print(linear_reg_model.coef_)
```

```
0.8238675120273313
[-0.26773758]
```

Let's see what these values are on the plot!

[15]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df.x, df.y)
# Plot predicted values of y
ax.plot(df.x, linear_reg_model.predict(features), 'k--', label='model fit')
ax.scatter(0,linear_reg_model.intercept_, s=50, c='r', label='intercept')

ax.legend(loc=1)
plt.show()
```



[16]:
```
print(linear_reg_model.intercept_ + 5*linear_reg_model.coef_)
```

```
[-0.51482041]
```

[17]:
```
print(linear_reg_model.predict(np.zeros([1,1])+5))
```

```
[-0.51482041]
```

1.2 2. Polynomial Regression

With polynomial regression, we are simply trying to fit the following formula: y = wx + wx^2 + c

[18]:
```python
# Make a copy of df
df_poly = df.copy()
```

[19]:
```python
# Create the x^2 input feature
df_poly['x^2'] = np.power(df_poly.x, 2)
```

[20]:
```python
df_poly.head()
```

[20]:

	x	y	x^2
0	0.000000	0.086260	0.000000
1	0.063467	0.881165	0.004028
2	0.126933	0.145261	0.016112
3	0.190400	-0.252824	0.036252
4	0.253866	-0.320448	0.064448

[21]:
```python
linear_model = LinearRegression()

features = df_poly.drop('y', axis=1)
target = df_poly.y

linear_model.fit(features, target)
```

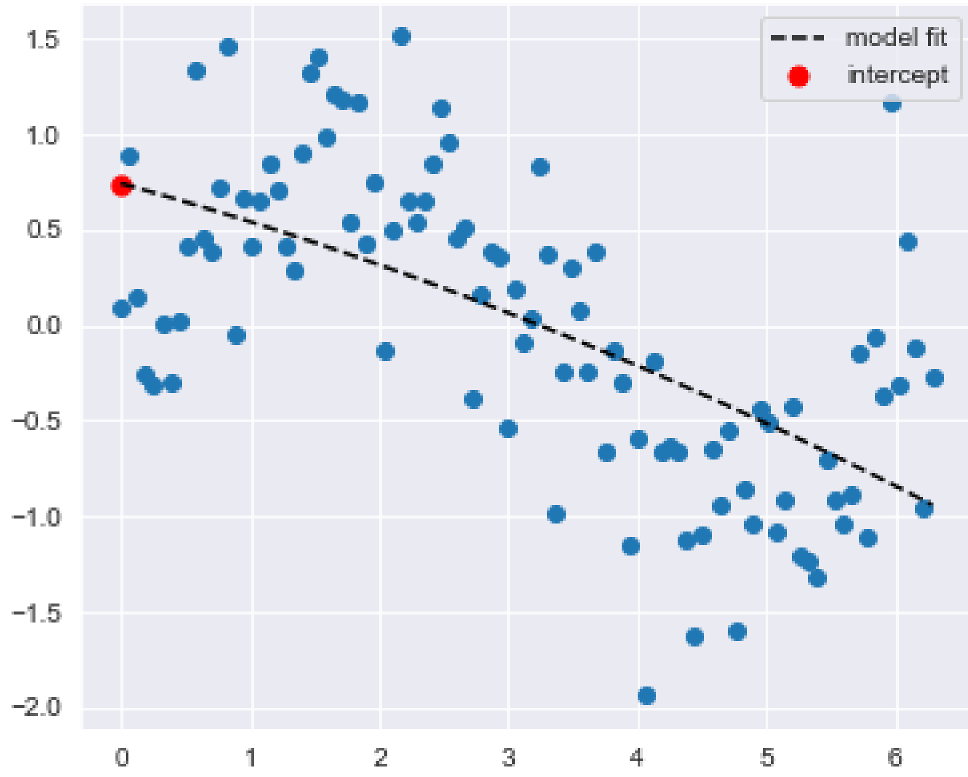
[21]:
```
LinearRegression()
```

[22]:
```python
target
```

[22]:
```
0     0.086260
1     0.881165
2     0.145261
3    -0.252824
4    -0.320448
        …
95   -0.312249
96    0.443255
97   -0.122903
98   -0.961879
99   -0.268629
Name: y, Length: 100, dtype: float64
```

[23]:
```python
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, df_poly.y)
# Plot predicted values of y
ax.plot(df_poly.x, linear_model.predict(features), 'k--', label='model fit')
ax.scatter(0,linear_model.intercept_, s=50, c='r', label='intercept')
```

```
ax.legend(loc=1)
plt.show()
```



1.2.1 Higher order polynomials

Well that was a bit disappointing. Let's see how we can predict these curves in the data.

[24]:
```
third_order_df = df.copy()
```

[25]:
```
third_order_df['x^2'] = np.power(third_order_df.x, 2)
```

[26]:
```
third_order_df['x^3'] = np.power(third_order_df.x, 3)
```

[27]:
```
third_order_df.head()
```

[27]:

	x	y	x^2	x^3
0	0.000000	0.086260	0.000000	0.000000
1	0.063467	0.881165	0.004028	0.000256
2	0.126933	0.145261	0.016112	0.002045
3	0.190400	-0.252824	0.036252	0.006902
4	0.253866	-0.320448	0.064448	0.016361

[28]:
```
features = third_order_df.drop('y', axis=1)
target = third_order_df.y
```

[29]:
```
features
```

[29]:

	x	x^2	x^3
0	0.000000	0.000000	0.000000
1	0.063467	0.004028	0.000256
2	0.126933	0.016112	0.002045
3	0.190400	0.036252	0.006902
4	0.253866	0.064448	0.016361
..	…	…	…
95	6.029319	36.352690	219.181976
96	6.092786	37.122038	226.176626
97	6.156252	37.899442	233.318526
98	6.219719	38.684902	240.609211
99	6.283185	39.478418	248.050213

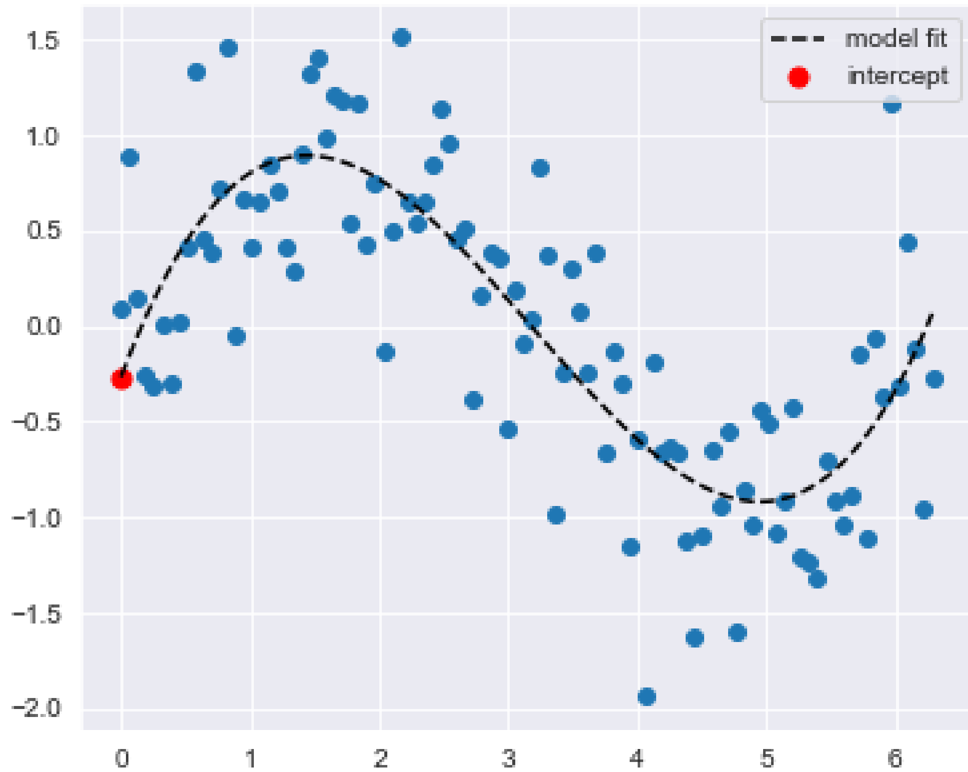
[100 rows x 3 columns]

[30]:
```
third_order_model = LinearRegression()
```

[31]:
```
third_order_model.fit(features, target)
```

[31]:
```
LinearRegression()
```

[32]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, df_poly.y)
# Plot predicted values of y
ax.plot(third_order_df.x, third_order_model.predict(features), 'k--',␣
 ↪label='model fit')
ax.scatter(0,third_order_model.intercept_, s=50, c='r', label='intercept')

ax.legend(loc=1)
plt.show()
```



How do we make it so that we don't have to increase the lines of code we write exponentially as we increase the order?

1.3 3. Logistic Regression

[33]:
```
from sklearn.linear_model import LogisticRegression
```

[34]:
```
# Logistic regression
model = LogisticRegression()
```

[35]:
```
target
```

[35]:
```
0      0.086260
1      0.881165
2      0.145261
3     -0.252824
4     -0.320448
         …
95    -0.312249
96     0.443255
97    -0.122903
98    -0.961879
```

```
99   -0.268629
Name: y, Length: 100, dtype: float64
```

[36]:
```
model.fit(features, target)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [36], in <cell line: 1>()
----> 1 model.fit(features, target)

File ~/Library/Python/3.8/lib/python/site-packages/sklearn/linear_model/
 ↪_logistic.py:1345, in LogisticRegression.fit(self, X, y, sample_weight)
   1340     _dtype = [np.float64, np.float32]
   1342 X, y = self._validate_data(X, y, accept_sparse='csr', dtype=_dtype,
   1343                            order="C",
   1344                            accept_large_sparse=solver != 'liblinear')
-> 1345 check_classification_targets(y)
   1346 self.classes_ = np.unique(y)
   1348 multi_class = _check_multi_class(self.multi_class, solver,
   1349                                  len(self.classes_))

File ~/Library/Python/3.8/lib/python/site-packages/sklearn/utils/multiclass.py:
 ↪172, in check_classification_targets(y)
    169 y_type = type_of_target(y)
    170 if y_type not in ['binary', 'multiclass', 'multiclass-multioutput',
    171                   'multilabel-indicator', 'multilabel-sequences']:
--> 172     raise ValueError("Unknown label type: %r" % y_type)

ValueError: Unknown label type: 'continuous'
```

[37]:
```
print(target)
```

```
0     0.086260
1     0.881165
2     0.145261
3    -0.252824
4    -0.320448
        …
95   -0.312249
96    0.443255
97   -0.122903
98   -0.961879
99   -0.268629
Name: y, Length: 100, dtype: float64
```

What does this error mean?

[38]:
```
print(target)
```

```
0     0.086260
1     0.881165
2     0.145261
3    -0.252824
4    -0.320448
        …
95   -0.312249
96    0.443255
97   -0.122903
98   -0.961879
99   -0.268629
Name: y, Length: 100, dtype: float64
```

[39]:
```
target_categorical = np.zeros(len(target))
```

[40]:
```
for i in range(len(target)):
    if (target[i] <=0.):
        target_categorical[i] = 0
    else:
        target_categorical[i] = 1
```

[41]:
```
print(target_categorical, target_categorical.dtype)
```

```
[1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 1. 0.
 1. 0. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.
 1. 0. 0. 0.] float64
```

[42]:
```
target_categorical = target_categorical.astype(int)
```

[43]:
```
print(target_categorical, target_categorical.dtype)
```

```
[1 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0] int64
```

[44]:
```
model.fit(features, target_categorical)
```

[44]:
```
LogisticRegression()
```

[45]:
```
# predict()
model.predict(features)
```

[45]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

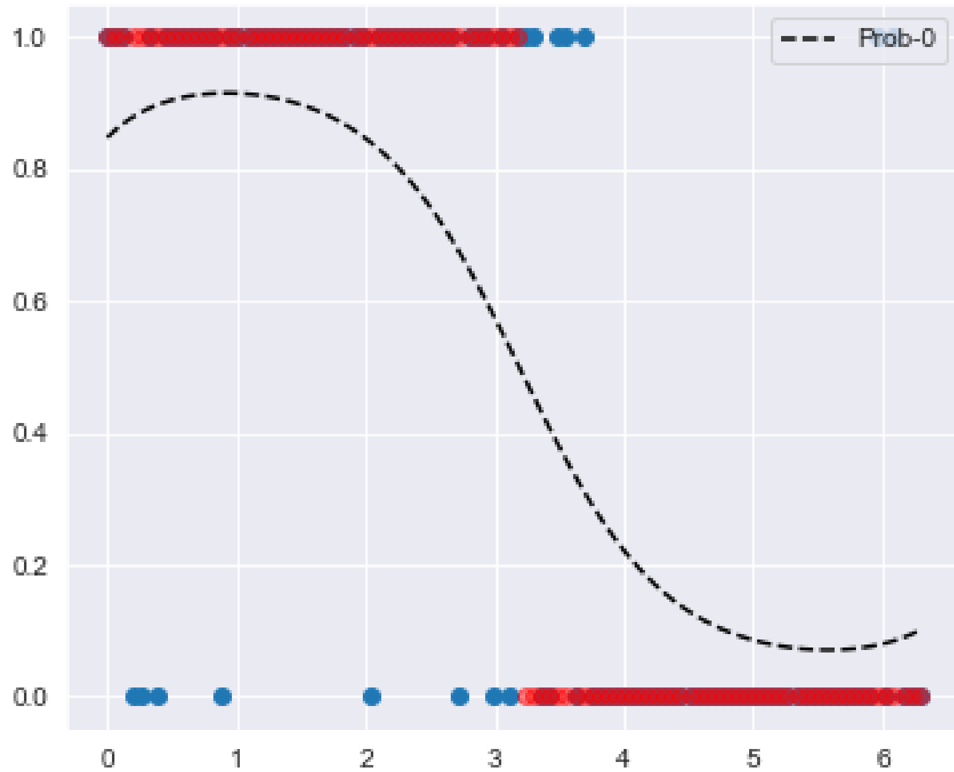
[46]:
```
# # predict_proba()
model.predict_proba(features[:10])
```

[46]:
```
array([[0.15195352, 0.84804648],
       [0.14040579, 0.85959421],
       [0.13045881, 0.86954119],
       [0.12191011, 0.87808989],
       [0.11458442, 0.88541558],
       [0.10833136, 0.89166864],
       [0.10302282, 0.89697718],
       [0.09855037, 0.90144963],
       [0.09482285, 0.90517715],
       [0.09176414, 0.90823586]])
```

[47]:
```
df.x
```

[47]:
```
0     0.000000
1     0.063467
2     0.126933
3     0.190400
4     0.253866
        …
95    6.029319
96    6.092786
97    6.156252
98    6.219719
99    6.283185
Name: x, Length: 100, dtype: float64
```

[48]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, target_categorical)
ax.scatter(df_poly.x, model.predict(features), color='r', alpha=0.5)
# Plot predicted values of y
ax.plot(third_order_df.x, model.predict_proba(features)[:,1], 'k--',␣
 ↪label='Prob-0')

ax.legend(loc=1)
plt.show()
```



How does Logistic Regression learn?

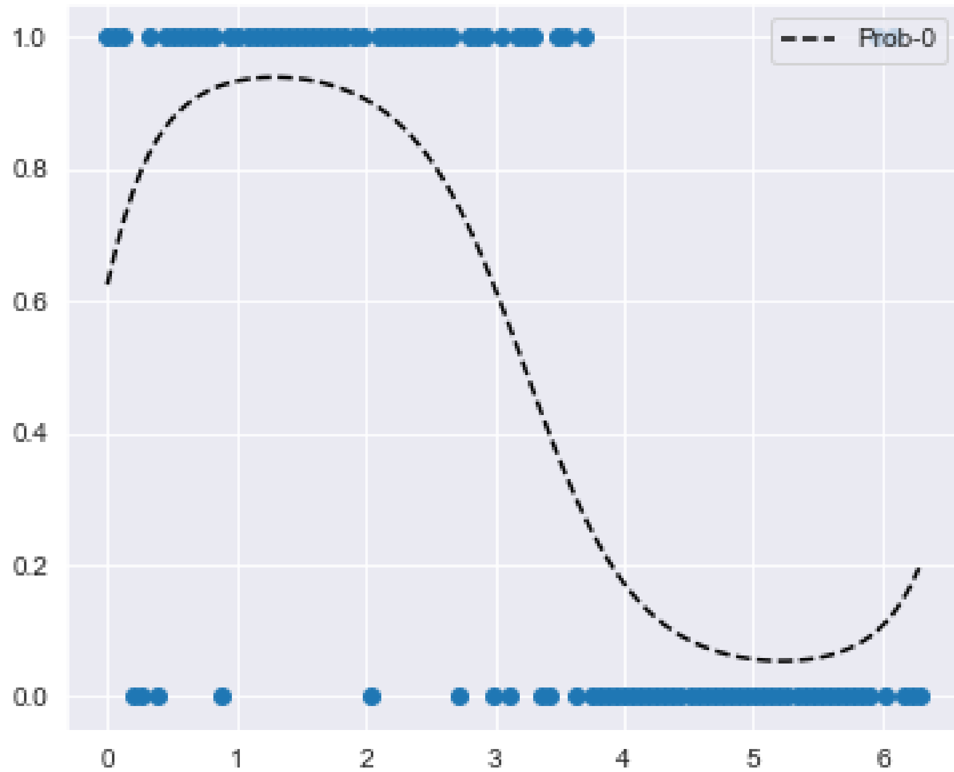
[49]:
```
model2 = LogisticRegression(C=5)
```

[50]:
```
LogisticRegression?
```

[51]:
```
model2.fit(features, target_categorical)
```

[51]: LogisticRegression(C=5)

[52]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, target_categorical)
# Plot predicted values of y
ax.plot(third_order_df.x, model2.predict_proba(features)[:,1], 'k--',␣
 ↪label='Prob-0')

ax.legend(loc=1)
plt.show()
```

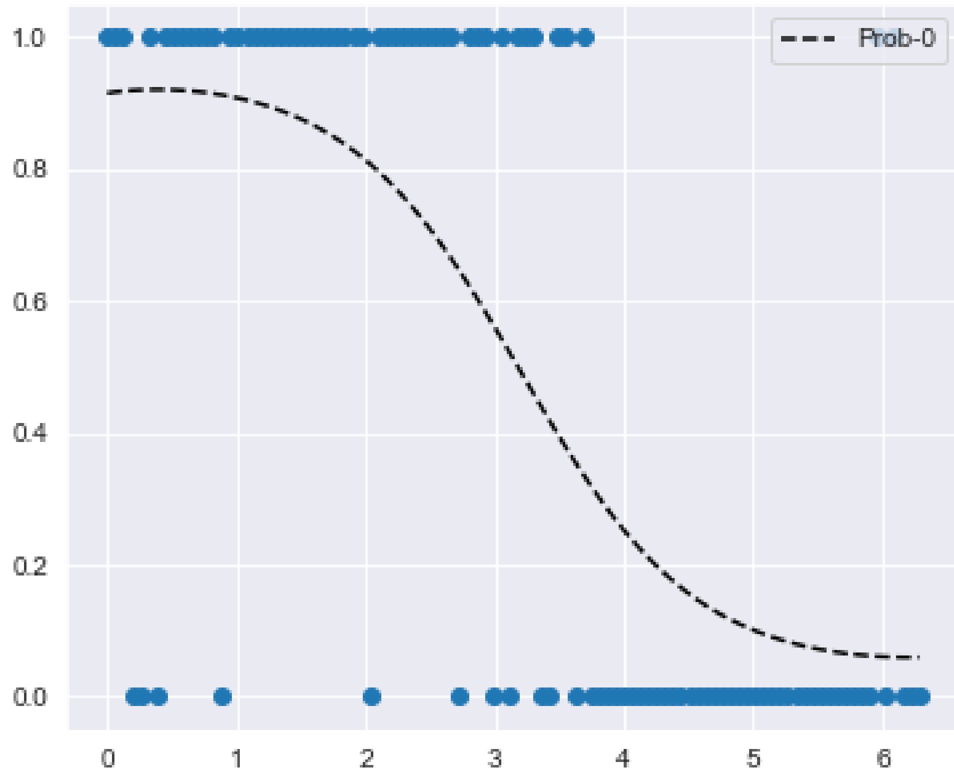


[53]:
```
model3 = LogisticRegression(C=0.25)
```

[54]:
```
model3.fit(features, target_categorical)
```

[54]: LogisticRegression(C=0.25)

[55]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, target_categorical)
# Plot predicted values of y
ax.plot(third_order_df.x, model3.predict_proba(features)[:,1], 'k--',␣
 ↪label='Prob-0')

ax.legend(loc=1)
plt.show()
```



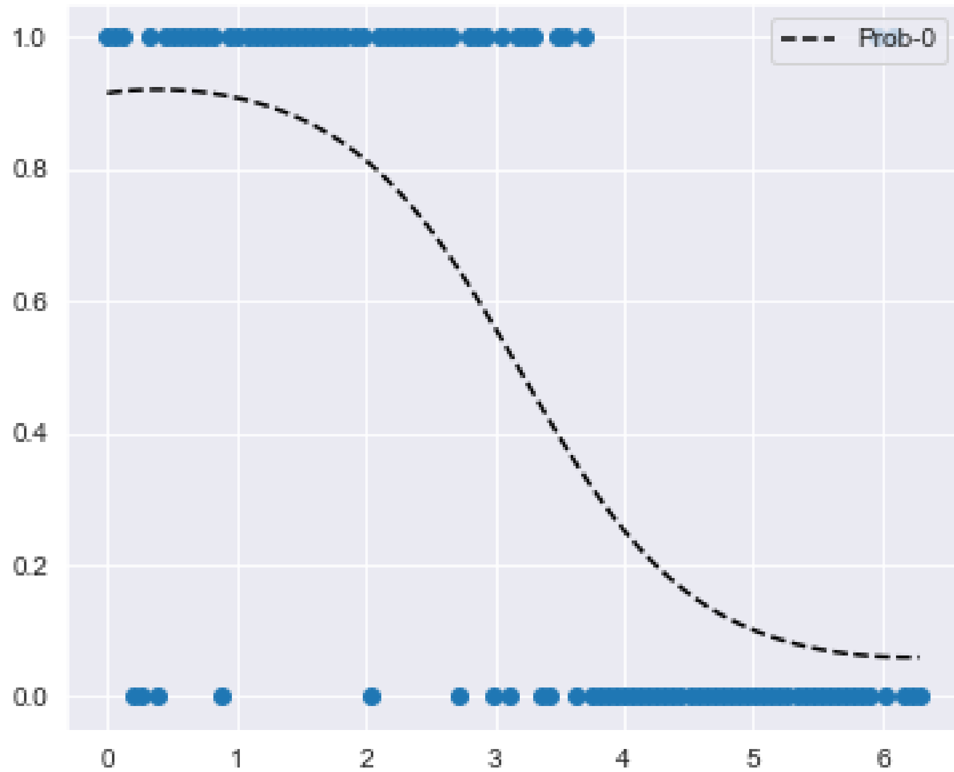
This is how the logistic regression behaves with different penalties, (values of C). Is there other penalty types?

[56]:
```
model4 = LogisticRegression(penalty='l2', C=0.25)
```

[57]:
```
model4.fit(features, target_categorical)
```

[57]: LogisticRegression(C=0.25)

[58]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_poly.x, target_categorical)
# Plot predicted values of y
ax.plot(third_order_df.x, model4.predict_proba(features)[:,1], 'k--',␣
 ↪label='Prob-0')

ax.legend(loc=1)
plt.show()
```



1.4 4. Regularized Regression

Let's start with a similar dataframe, but with more parameters.

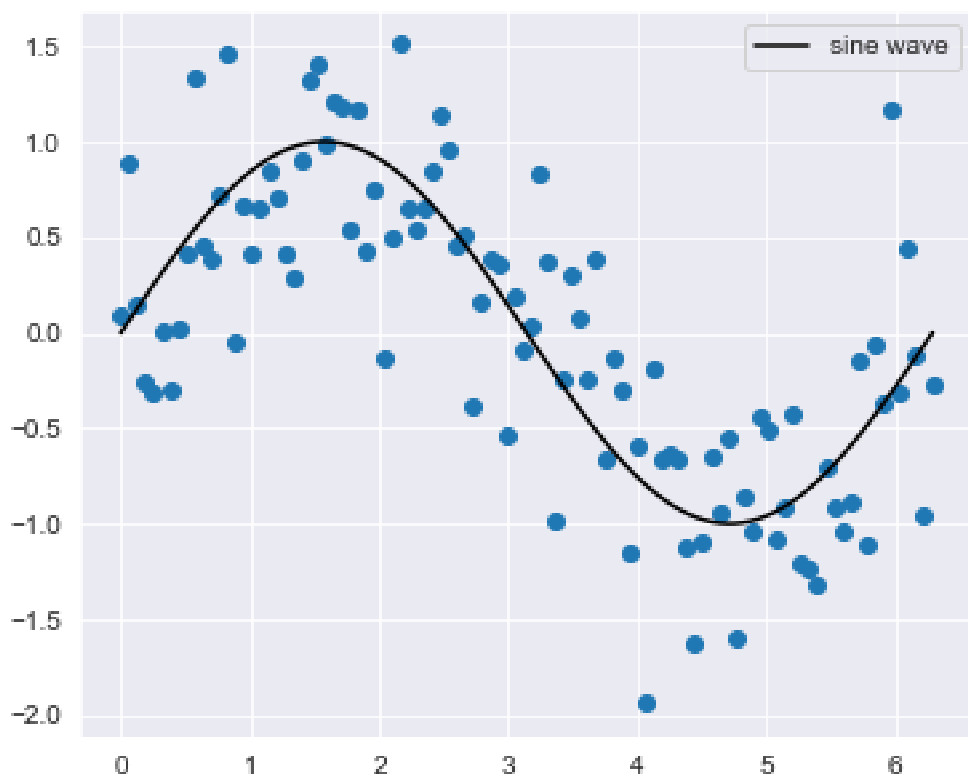
[59]:
```
# input feature
x = np.linspace(0, 2*np.pi, 100)

# noise
np.random.seed(321)
noise = np.random.normal(0, .5, 100)

# target variable
y = np.sin(x) + noise

# Create DataFrame with x and y
df = pd.DataFrame({'x' : x, 'y': y})
df.head()
```

[59]:

	x	y
0	0.000000	0.086260
1	0.063467	0.881165
2	0.126933	0.145261

```
3  0.190400 -0.252824
4  0.253866 -0.320448
```

[60]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df.x, df.y)
ax.plot(df.x, np.sin(df.x), color='k', label='sine wave')

ax.legend(loc=1)
plt.show()
```



[61]:
```
df_sample = df.copy()

# Set seed for reproducible results
np.random.seed(555)

# Create 100 randomfeatures
for n in range(100):
    df_sample['s{}'.format(n)] = np.random.uniform(0, 2*np.pi, 100)
```

```
/var/folders/d0/xtw4cvdd52nfnnxn7pz_jdfw0000gn/T/ipykernel_85760/2307497470.py:8
: PerformanceWarning: DataFrame is highly fragmented.  This is usually the
result of calling `frame.insert` many times, which has poor performance.
Consider joining all columns at once using pd.concat(axis=1) instead.  To get a
de-fragmented frame, use `newframe = frame.copy()`
  df_sample['s{}'.format(n)] = np.random.uniform(0, 2*np.pi, 100)
```

[62]:
```python
df_sample.head()
```

[62]:
```
          x         y        s0        s1        s2        s3        s4  \
0  0.000000  0.086260  4.510285  4.243360  5.077364  5.775407  0.870371
1  0.063467  0.881165  0.300683  2.204225  0.492213  5.097731  0.541594
2  0.126933  0.145261  5.934292  4.499745  5.629712  0.721088  4.499597
3  0.190400 -0.252824  4.312653  1.798761  6.126937  0.804294  4.019667
4  0.253866 -0.320448  3.651833  4.475723  5.188299  3.456177  3.971505

         s5        s6        s7  …       s90       s91       s92       s93  \
0  1.042189  3.288845  1.005865  …  0.401207  2.986740  3.419816  2.210869
1  0.343164  4.840217  4.343630  …  2.432486  1.130922  0.067541  2.857747
2  1.054771  3.207139  2.356321  …  3.335856  5.061090  0.854903  1.861820
3  5.314475  3.989411  2.739722  …  0.430104  3.887071  0.294676  0.168721
4  1.095373  6.279545  1.951886  …  6.061850  5.710574  1.732244  4.721764

        s94       s95       s96       s97       s98       s99
0  4.116052  5.651802  2.931066  1.542499  2.299154  2.986066
1  5.934834  5.213061  5.579050  4.092146  4.464422  5.644063
2  0.216191  4.707951  1.171044  3.402163  5.652485  1.765618
3  4.298122  5.752676  5.008056  3.983089  5.742110  1.424242
4  4.733892  5.386840  1.901648  3.090707  1.974244  4.010548

[5 rows x 102 columns]
```

[63]:
```python
X_sample = df_sample.drop('y', axis=1)
```

1.4.1 4.1 Lasso Regression

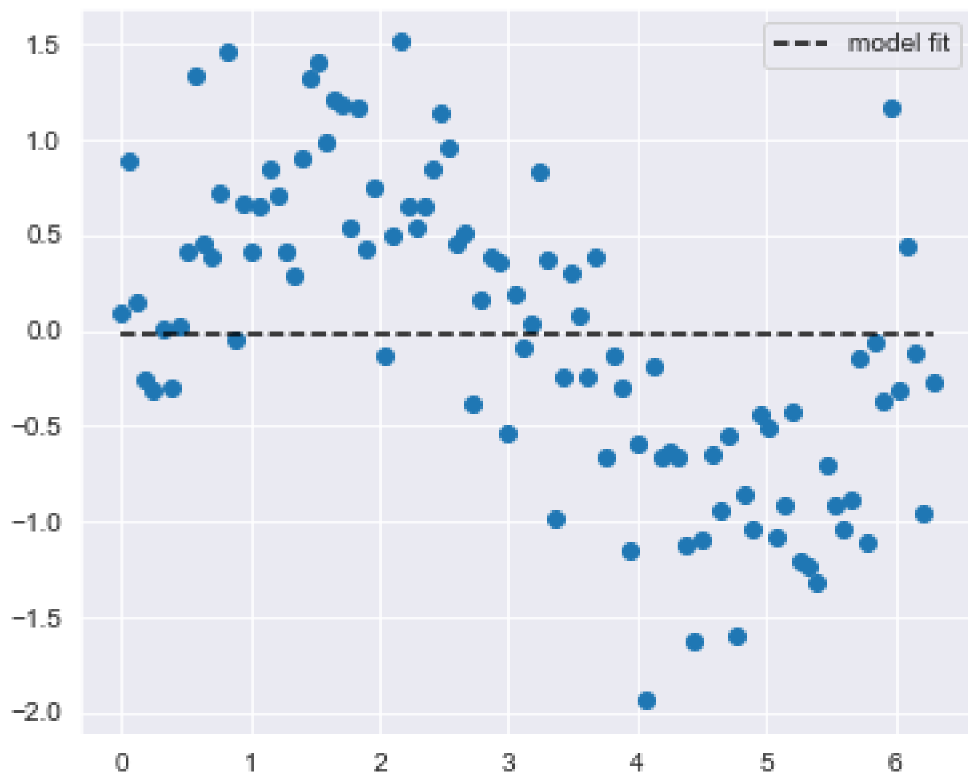
[64]:
```python
from sklearn.linear_model import Lasso
```

[65]:
```python
lasso = Lasso(random_state=1234)
lasso.fit(X_sample, y)
```

[65]:
```
Lasso(random_state=1234)
```

[66]:
```python
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
```

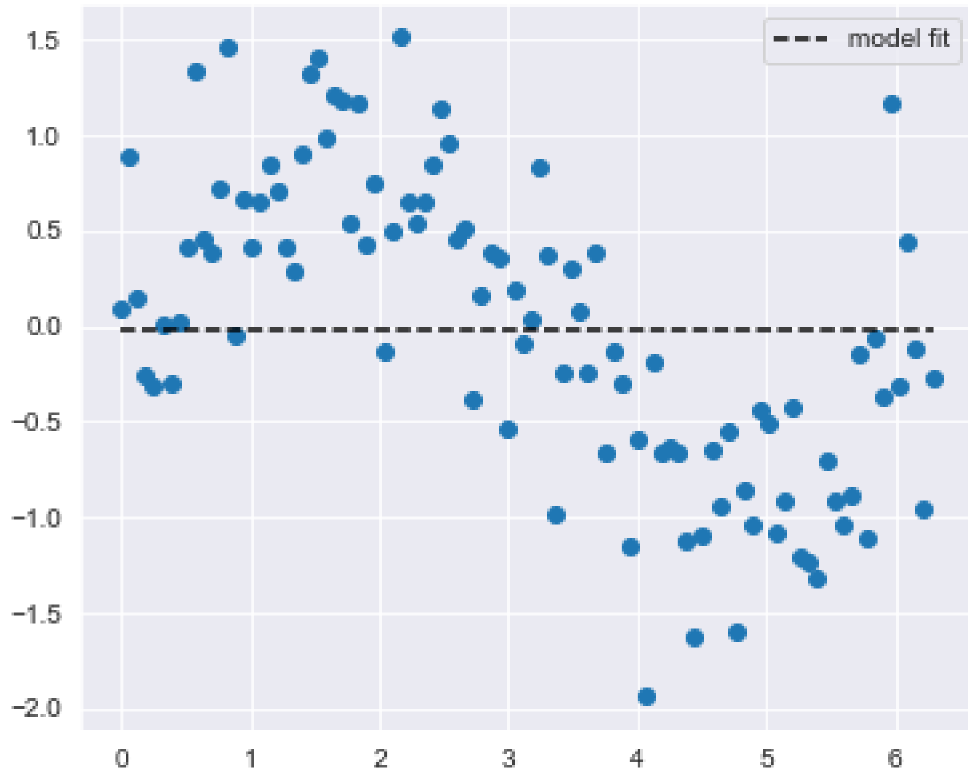
```
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, lasso.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



[67]:
```
lasso = Lasso(alpha=2.0, random_state=1234)
lasso.fit(X_sample, y)
```

[67]: Lasso(alpha=2.0, random_state=1234)

[68]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, lasso.predict(X_sample), 'k--', label='model fit')

ax.legend(loc=1)
plt.show()
```



[69]:
```
print(lasso.coef_)
```

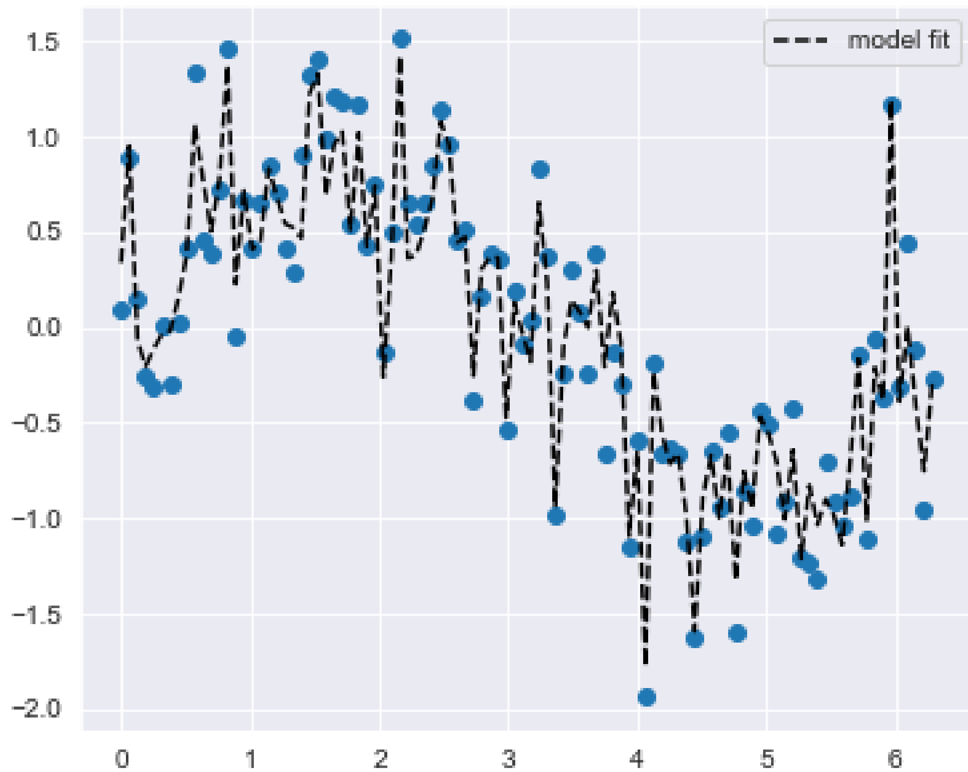
```
[-0.  0. -0. -0.  0.  0. -0. -0. -0.  0.  0. -0. -0.  0.  0. -0. -0. -0.
 -0.  0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0. -0. -0.  0. -0.  0. -0.
  0.  0.  0. -0. -0. -0.  0.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0. -0.
  0. -0. -0.  0. -0.  0. -0.  0. -0. -0. -0.  0. -0. -0.  0.  0.  0.  0.
  0.  0.  0.  0.  0. -0. -0.  0.  0.  0. -0. -0. -0.  0. -0.  0.  0. -0.
 -0.  0. -0.  0.  0.  0.  0. -0. -0.  0.  0.]
```

[70]:
```
lasso = Lasso(alpha=0.01, random_state=1234)
lasso.fit(X_sample, y)
```

[70]: Lasso(alpha=0.01, random_state=1234)

[71]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, lasso.predict(X_sample), 'k--', label='model fit')
```

```
ax.legend(loc=1)
plt.show()
```



[72]:
```
print(lasso.coef_)
```

```
[-2.51917168e-01  4.10314859e-02  5.45436757e-03  1.93983589e-02
 -0.00000000e+00  3.71926650e-04 -2.91626816e-02 -7.30117603e-02
 -0.00000000e+00  2.35664422e-02  1.15328229e-02 -1.37269139e-01
 -8.55021573e-02  0.00000000e+00 -0.00000000e+00  0.00000000e+00
 -0.00000000e+00  2.31116907e-02 -7.69147144e-03  1.43637115e-02
 -4.59460085e-02  6.32005952e-02  0.00000000e+00  3.35372015e-02
 -2.04031104e-02 -2.47692101e-02  1.01483461e-02 -1.98289793e-02
  0.00000000e+00 -1.69208119e-02 -6.38448583e-02  4.88311922e-02
 -4.92076031e-02 -5.77076974e-02  1.46127834e-02  6.17103039e-02
  1.08025418e-02 -9.24093430e-03  1.39007203e-02 -3.45517512e-02
  7.75704597e-03  1.34676884e-02 -6.55351081e-02  0.00000000e+00
 -2.51704549e-02  2.19660309e-02 -6.18336806e-02 -0.00000000e+00
 -1.34811001e-03  8.01130519e-03  4.94648308e-02  2.74071927e-02
 -1.71572289e-02 -7.66719433e-02 -1.53841329e-02  4.25935667e-02
  0.00000000e+00  0.00000000e+00 -5.14947316e-02  5.38733880e-02
 -5.87659526e-02 -1.99279915e-02 -0.00000000e+00 -2.85859781e-02
  9.23077028e-03  3.66337147e-02 -8.92423823e-02 -3.19575719e-02
```

```
-3.20845176e-02  6.37818412e-02  5.52303012e-02  9.54397855e-02
 0.00000000e+00  3.95943641e-03  2.42927819e-02 -4.34199201e-02
-3.26335368e-02 -2.74058877e-02 -6.17839339e-02  0.00000000e+00
-4.88893697e-02 -5.27947085e-02 -0.00000000e+00 -4.22261956e-02
-3.17840242e-02  7.87870931e-02 -8.59615538e-02  6.18403170e-02
 8.91858350e-02  4.48748149e-06  1.86464362e-02  0.00000000e+00
-8.18425037e-03  5.11828963e-02 -0.00000000e+00 -0.00000000e+00
-2.09412108e-02  0.00000000e+00 -1.16449350e-01 -2.46500974e-02
-1.53086323e-02]
```

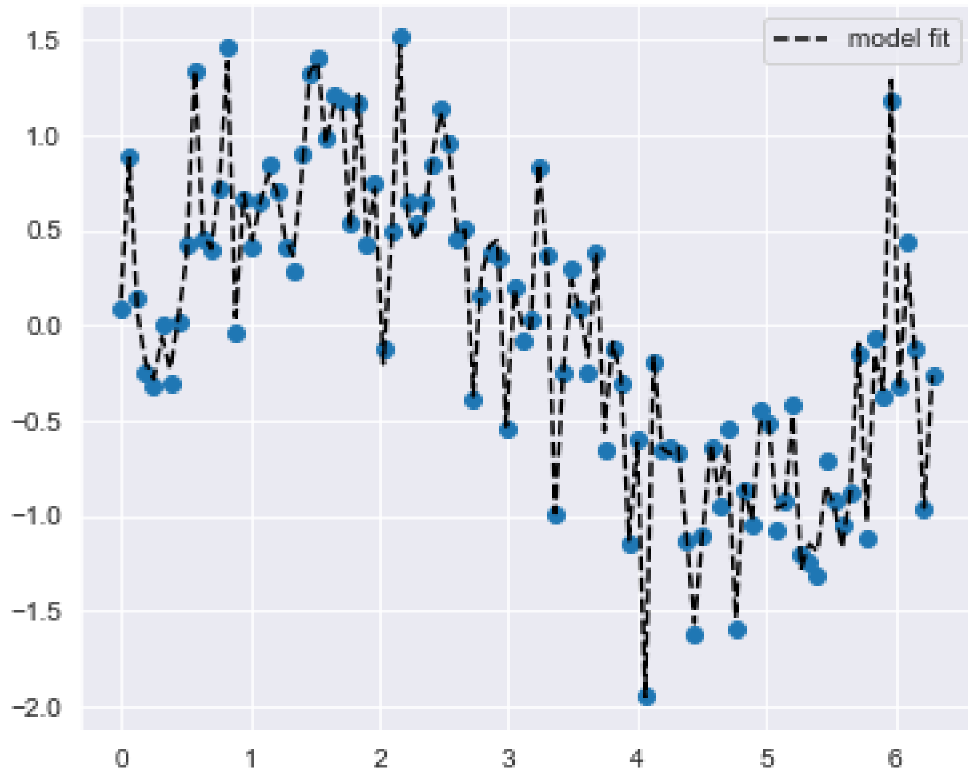
1.4.2 4.2 Ridge Regression

[73]:
```
from sklearn.linear_model import Ridge
```

[74]:
```
# Default alpha=1.0
ridge = Ridge(alpha=1.0, random_state=1234)
ridge.fit(X_sample, y)
```

[74]:
```
Ridge(random_state=1234)
```

[75]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, ridge.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



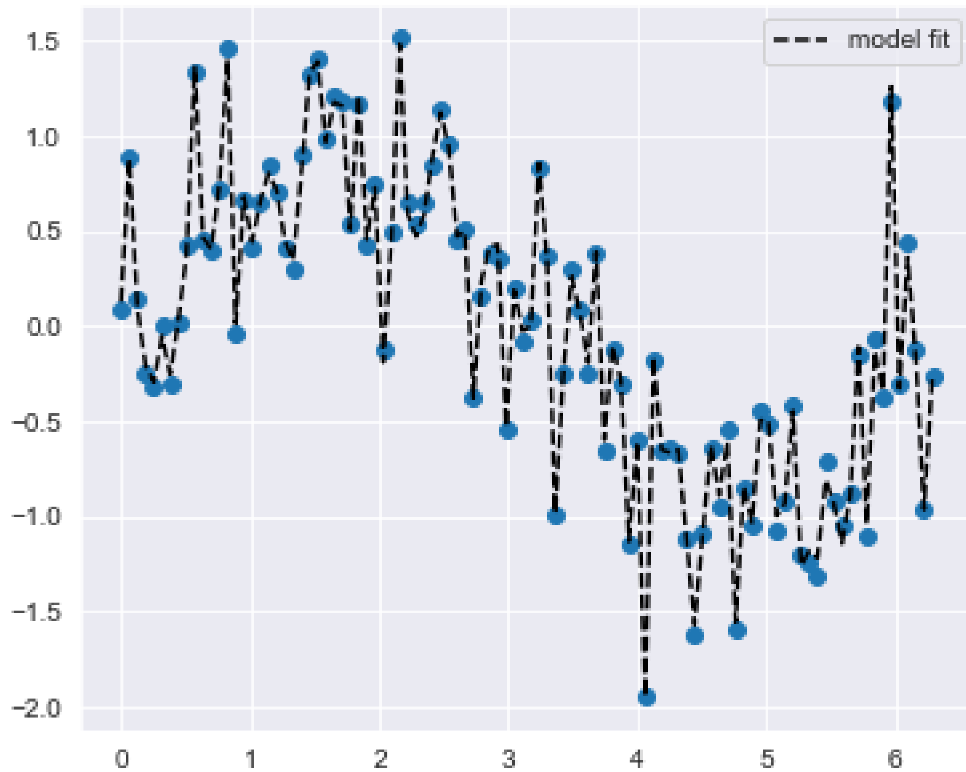
[76]:
```
print( ridge.coef_ )
```

```
[-0.29684715  0.07811928 -0.01081815  0.07752363 -0.05209391  0.06597177
 -0.09941976 -0.02095678  0.04224917  0.03330623  0.02933095 -0.22990378
 -0.05383043 -0.09501552 -0.00916561  0.02168726  0.01117362  0.0247942
 -0.04128874 -0.06015788 -0.07177257  0.07528721 -0.05468463  0.08178794
  0.00990867 -0.07854772  0.02718976 -0.02008819  0.06053268 -0.05168845
 -0.11625843  0.14997976 -0.11653335 -0.14694831 -0.00530649  0.05605462
  0.07446295 -0.09937726  0.01051123 -0.1072047   0.01843767 -0.00293992
 -0.12148397 -0.11080856 -0.05319224  0.03874233 -0.05698693 -0.05486004
 -0.06322066  0.02182041  0.06517839  0.04999928 -0.07282591 -0.19374842
  0.01673706  0.0749228   0.02540259  0.02090189 -0.01943677  0.11922506
 -0.14289904 -0.093589   -0.03210794 -0.03192064  0.05836246  0.12613361
 -0.17169424 -0.10278082 -0.1097883   0.05785305  0.08117485  0.09895021
  0.03154747 -0.08589195  0.04408258 -0.11580108 -0.09865743  0.05249017
 -0.09796656 -0.05948369 -0.05389152 -0.11718357 -0.02775835 -0.06816065
 -0.08087182  0.06917164 -0.09438014  0.0637011   0.18535325 -0.03448473
  0.02340283 -0.10506904  0.03868299  0.13380026  0.01774494  0.00232615
 -0.05014508  0.03333691 -0.17311278 -0.03324787 -0.05294611]
```

[77]:
```
ridge = Ridge(alpha=0.5, random_state=1234)
ridge.fit(X_sample, y)
```

[77]: Ridge(alpha=0.5, random_state=1234)

[78]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, ridge.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



[79]:
```
print( ridge.coef_ )
```

```
[-0.30208389  0.0783116  -0.008561    0.08204715 -0.06670881  0.07344217
 -0.11053133 -0.01485217  0.04515549  0.04721998  0.05048001 -0.25218637
 -0.04353109 -0.13977087 -0.01749479  0.01417613  0.01891266 -0.00095329
 -0.05954435 -0.07859067 -0.06973464  0.07329633 -0.06157125  0.10146629
  0.01389787 -0.10182674  0.0189875  -0.02820367  0.08443438 -0.06468595
 -0.10173674  0.17575185 -0.12726992 -0.17792542 -0.02502084  0.06368206
  0.09362423 -0.11517384  0.01386887 -0.13302902  0.00483455 -0.01414969
 -0.11959204 -0.15977016 -0.07539503  0.01780257 -0.02981305 -0.04971948
```

```
-0.08259787  0.02318377  0.04660821  0.04864542 -0.08192852 -0.21225363
 0.0346915   0.07696095  0.01383101  0.01961826 -0.00164144  0.1371216
-0.16776161 -0.11472388 -0.05048734 -0.02597299  0.06294517  0.1578618
-0.1916048  -0.14130211 -0.1370536   0.05279126  0.10142856  0.09115189
 0.02825284 -0.11447375  0.06534487 -0.1014944  -0.11054924  0.09896938
-0.10404176 -0.08390335 -0.04848419 -0.11209537 -0.03301159 -0.07710636
-0.08741069  0.07106847 -0.07141465  0.08216359  0.22342028 -0.05613415
 0.00582488 -0.14286738  0.05970706  0.16863328  0.00186668  0.00269799
-0.04637963  0.06639775 -0.18638498 -0.03430204 -0.05040116]
```

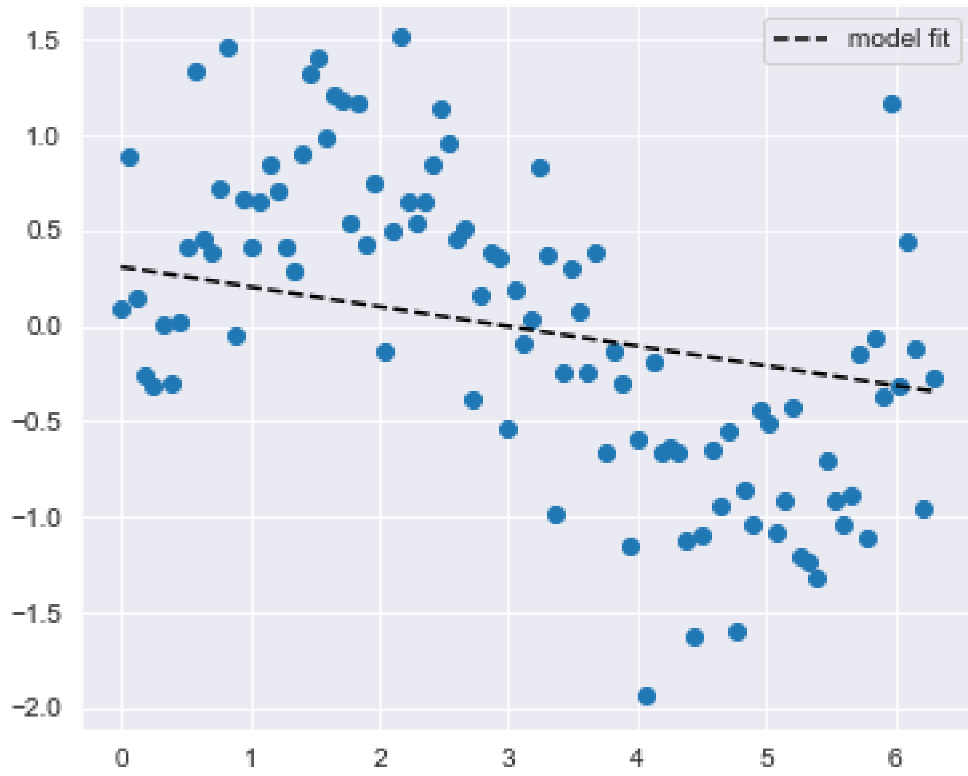
1.4.3 4.3 Elastic Net Regression

[80]:
```python
from sklearn.linear_model import ElasticNet
```

[81]:
```python
enet = ElasticNet(alpha=1.0,
                  l1_ratio=0.5,
                  random_state=1234)

enet.fit(X_sample, y)
```

[81]:
```
ElasticNet(random_state=1234)
```

[82]:
```python
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, enet.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



[83]:
```
print( enet.coef_ )
```

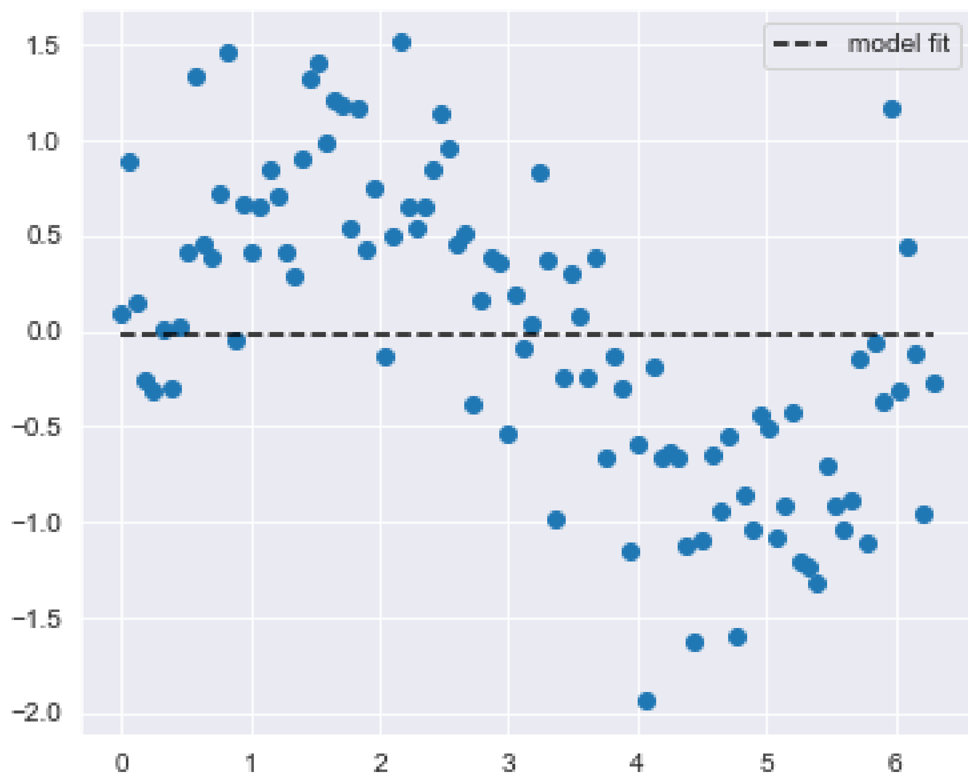
```
[-0.1033666  0.          -0.          -0.           0.           0.
 -0.         -0.          -0.           0.           0.          -0.
 -0.          0.           0.          -0.          -0.           0.
 -0.          0.          -0.           0.           0.           0.
 -0.         -0.          -0.          -0.          -0.          -0.
 -0.         -0.           0.          -0.           0.          -0.
  0.          0.          -0.          -0.          -0.          -0.
  0.          0.          -0.           0.           0.           0.
  0.         -0.           0.           0.           0.          -0.
  0.         -0.          -0.           0.          -0.           0.
 -0.          0.          -0.          -0.          -0.           0.
 -0.         -0.           0.           0.           0.           0.
  0.          0.           0.          -0.           0.          -0.
 -0.          0.           0.           0.          -0.          -0.
 -0.          0.          -0.           0.           0.          -0.
  0.          0.          -0.           0.           0.           0.
  0.         -0.          -0.           0.          -0.        ]
```

[84]:
```
enet = ElasticNet(alpha=1.0,
                  l1_ratio=1.0,
```

```
                    random_state=1234)

enet.fit(X_sample, y)
```

[84]: ElasticNet(l1_ratio=1.0, random_state=1234)

[85]:
```
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, enet.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



[86]:
```
print( enet.coef_ )
```

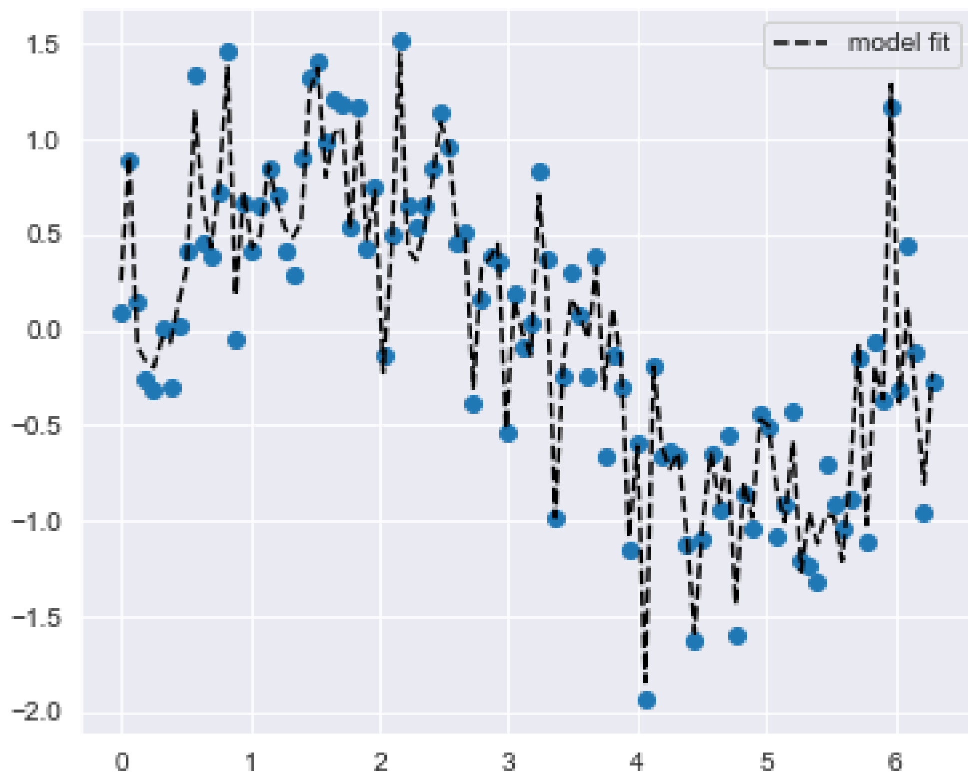
```
[-0.  0. -0. -0.  0.  0. -0. -0. -0.  0.  0. -0. -0.  0.  0. -0. -0. -0.
 -0.  0. -0.  0.  0.  0.  0. -0. -0. -0. -0. -0. -0. -0.  0. -0.  0. -0.
  0.  0.  0. -0. -0. -0.  0.  0. -0.  0.  0.  0.  0. -0.  0.  0.  0. -0.
  0. -0. -0.  0. -0.  0. -0.  0. -0. -0. -0.  0. -0. -0.  0.  0.  0.  0.
```

```
 0.  0.  0.  0.  0. -0. -0.  0.  0.  0. -0. -0. -0.  0. -0.  0.  0. -0.
-0.  0. -0.  0.  0.  0.  0. -0. -0.  0.  0.]
```

[87]:
```python
enet = ElasticNet(alpha=0.01,
                  l1_ratio=0.5,
                  random_state=1234)

enet.fit(X_sample, y)
```

[87]:
```
ElasticNet(alpha=0.01, random_state=1234)
```

[88]:
```python
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, enet.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



[89]:
```python
print( enet.coef_ )
```

```
[-0.26486045  0.04298242  0.00742904  0.03848651 -0.00369298  0.01630889
 -0.03969584 -0.06838583  0.          0.01223765  0.         -0.16079311
 -0.08590186 -0.          0.          0.00337793  0.          0.05439315
 -0.00560836  0.         -0.05292692  0.06151592 -0.          0.03678179
 -0.01437262 -0.01604844  0.04530877 -0.0125807   0.00112914 -0.02412691
 -0.09274653  0.07007337 -0.06400848 -0.06148557  0.01562378  0.05619311
  0.00425738 -0.03156357  0.01463145 -0.04938065  0.01605828  0.02144099
 -0.09605786  0.         -0.00775791  0.04290143 -0.07727029 -0.02384968
 -0.00279812  0.01140535  0.0814785   0.03202166 -0.04087497 -0.1070377
 -0.02090587  0.0609347   0.00652478  0.         -0.05690183  0.06219168
 -0.08034505 -0.03593893  0.         -0.02842273  0.01739034  0.05569333
 -0.11668482 -0.02039673 -0.02872962  0.06056502  0.05273333  0.09800944
  0.02009541 -0.          0.01641543 -0.08282496 -0.05206842 -0.02491
 -0.06816818 -0.         -0.05921067 -0.07776435 -0.00641733 -0.0514239
 -0.05477854  0.07539555 -0.11812429  0.05463899  0.1042674   0.00037661
  0.0363306  -0.         -0.00105311  0.05472615  0.         -0.00185223
 -0.04425153 -0.01161046 -0.11871534 -0.03399284 -0.04752461]
```

[90]:
```python
enet = ElasticNet(alpha=1.0,
                  l1_ratio=0,
                  random_state=1234)

enet.fit(X_sample, y)
```

```
/Users/dsozturk/Library/Python/3.8/lib/python/site-
packages/sklearn/linear_model/_coordinate_descent.py:529: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 8.882786828150797, tolerance: 0.006233383868788853
  model = cd_fast.enet_coordinate_descent(
```

[90]:
```
ElasticNet(l1_ratio=0, random_state=1234)
```

[91]:
```python
fig = plt.figure(figsize=[6,5])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df_sample.x, df_sample.y)
# Plot predicted values of y
ax.plot(df_sample.x, enet.predict(X_sample), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```

[92]:
```
print( enet.coef_ )
```

```
[-0.1599266   0.01901336 -0.00363695 -0.00701331 -0.00779949 -0.00975995
 -0.01587245 -0.04868295 -0.01002532  0.02207153  0.01044788 -0.06245985
 -0.05939289  0.01227431 -0.00862264  0.00531654 -0.0201262   0.00629576
 -0.00366068  0.03645775 -0.03281847  0.03151895  0.02204958  0.01871277
 -0.01021932 -0.03683324 -0.00489909 -0.01869302 -0.00999319 -0.02760615
 -0.03203678  0.00263973 -0.01396257 -0.02441583  0.02614446  0.01233672
  0.01448048  0.00963112  0.00520644 -0.02321304 -0.01128931  0.01108419
 -0.01789174  0.03177709 -0.0185849   0.01705269 -0.01919242  0.0113091
  0.0014417  -0.00225509  0.03813596  0.01481051  0.0012886  -0.03626768
 -0.00779874  0.00184868 -0.02142783 -0.0055303  -0.03341371  0.03054187
 -0.04082618 -0.00516867 -0.02344869 -0.01758168  0.01009945  0.01836767
 -0.03814949 -0.01978484  0.0018485   0.04231834  0.04378333  0.05589915
 -0.00504676  0.03452261  0.03459374 -0.01737254 -0.0053648  -0.03800603
 -0.02031135  0.01148725 -0.01919232 -0.01684551 -0.00358288 -0.0261021
 -0.01160044  0.04244081 -0.04482131  0.04204703  0.04420235  0.0025316
  0.02382157  0.02529897 -0.01754931  0.02673218 -0.00045326  0.02576185
 -0.00069439 -0.00704855 -0.06489623 -0.00116484 -0.00629574]
```

1.5 5. Decision Tree Regressor

[93]:
```
from sklearn.tree import DecisionTreeRegressor
```

[94]:
```
from sklearn.tree import export_graphviz
from graphviz import Source #comment out if you can't install graphviz
from IPython.display import Image
```

Let's start by creating a noisy sine wave like we did with Regression

[95]:
```
x = np.linspace(0, 2*np.pi, 100)
```

[96]:
```
# noise
np.random.seed(321)
noise = np.random.normal(0, .5, 100)
```

[97]:
```
# target variable
y = np.sin(x) + noise
```

Saving the new values into a dataframe.

[98]:
```
# Create DataFrame with x and y
df = pd.DataFrame({'x' : x, 'y': y})
df.head()
```

[98]:

	x	y
0	0.000000	0.086260
1	0.063467	0.881165
2	0.126933	0.145261
3	0.190400	-0.252824
4	0.253866	-0.320448

[99]:
```
# Splits input features from target variable
features = df.drop('y', axis=1)
target = df.y
```

Now we can create an unconstrained decision tree model.

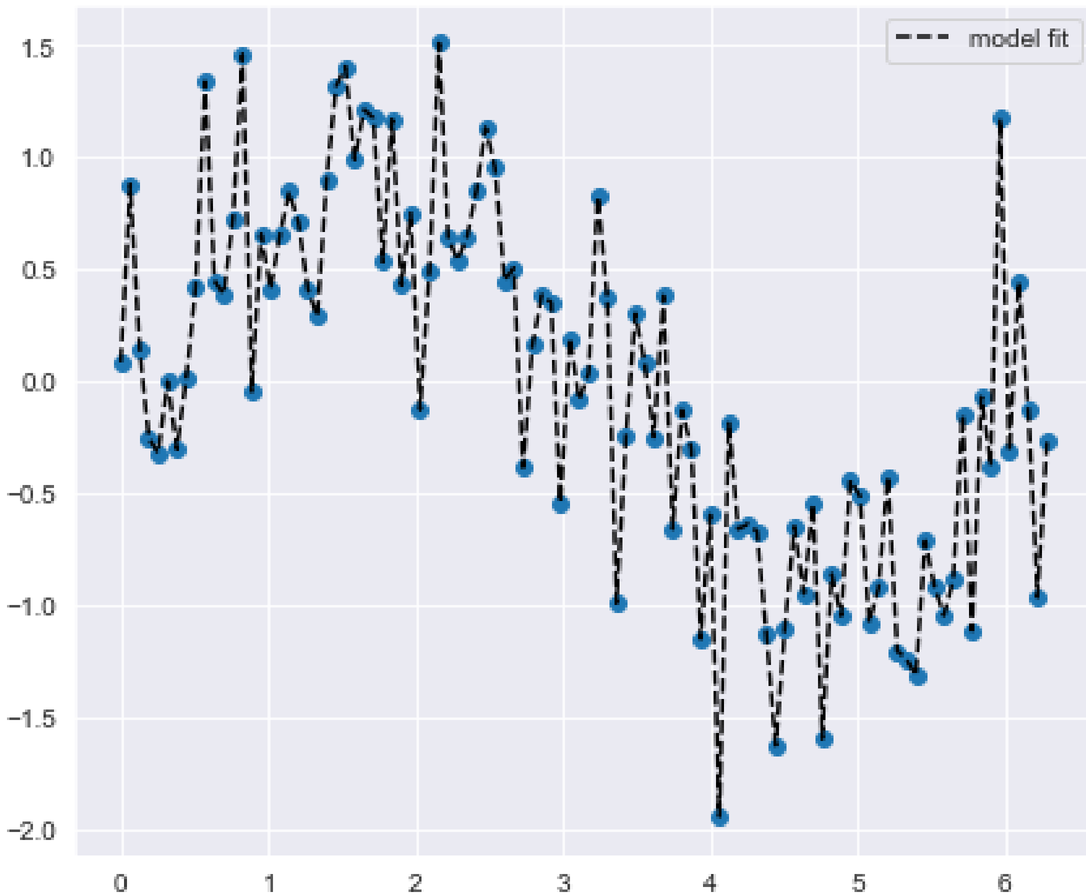
[100]:
```
# Unconstrained Decision Tree
dt_model = DecisionTreeRegressor()
```

[101]:
```
# Fit model
dt_model.fit(features, target)
```

[101]:
```
DecisionTreeRegressor()
```

[102]:
```
fig = plt.figure(figsize=[7,6])
```

```
ax = plt.subplot(111)
# Plot original points
ax.scatter(df.x, df.y)
# Plot predicted values of y
ax.plot(df.x, dt_model.predict(features), 'k--', label='model fit')
ax.legend(loc=1)
plt.show()
```



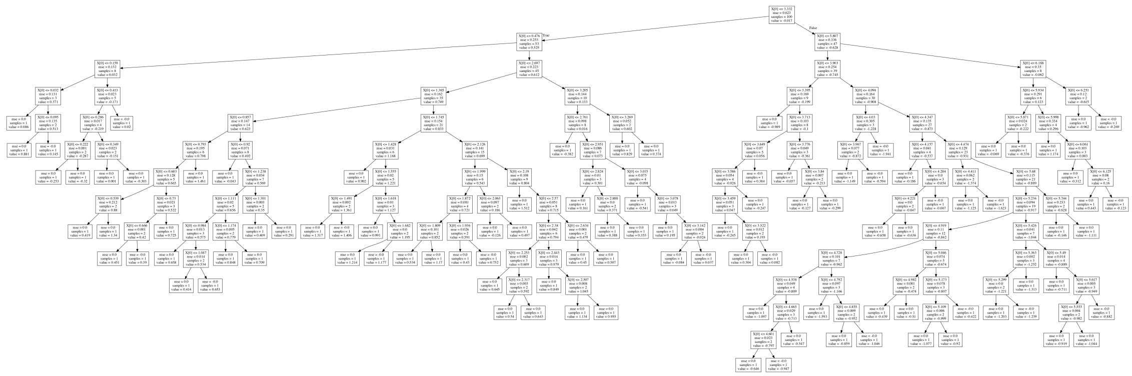
In case you had trouble downloading graphviz, you can omit the following steps.

[103]:
```
# Export tree structure and save to Source object
graph = Source( export_graphviz(dt_model, out_file=None) ) #comment out if you␣
↪can't install graphviz
```

[104]:
```
tree_png = graph.pipe(format='png') #comment out if you can't install graphviz
```

[105]:
```
Image(tree_png) #comment out if you can't install graphviz
```

[105]:



Let's see if this is a good fit.

[106]:
```
dt_model.tree_.node_count
```

[106]:
```
199
```

[107]:
```
from sklearn.metrics import mean_squared_error as mse
```

[108]:
```
print('The mean squared error: ',mse(df.y,dt_model.predict(features)))
```

```
The mean squared error:  0.0
```

[109]:
```
dt_model.feature_importances_
```

[109]:
```
array([1.])
```

1.6 Constrained Decision Trees

1.6.1 Min samples per leaf

Let's explore how to constrain a decision tree by limiting the amount of samples that we can have in a leaf.

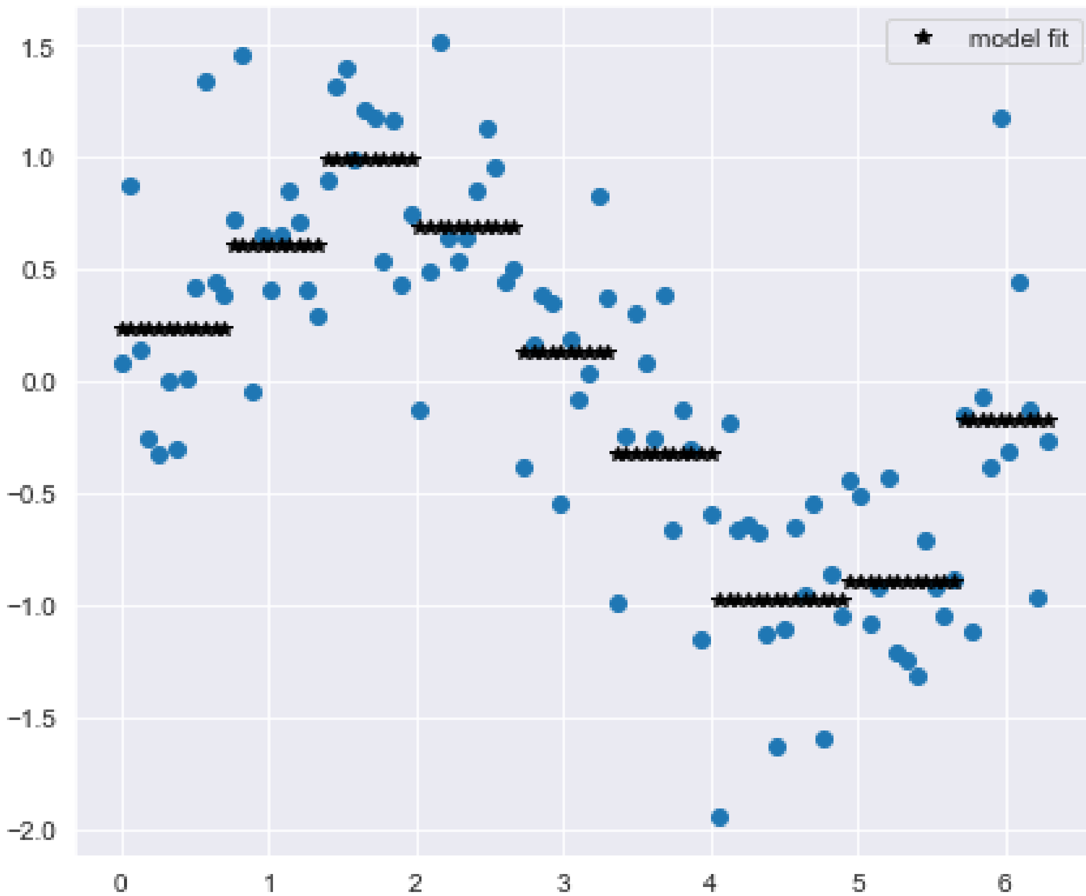
[110]:
```
dt_model2 = DecisionTreeRegressor(min_samples_leaf = 10)
```

[111]:
```
# Fit model
dt_model2.fit(features, target)
```

[111]:
```
DecisionTreeRegressor(min_samples_leaf=10)
```

[112]:
```
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df.x, df.y)
```

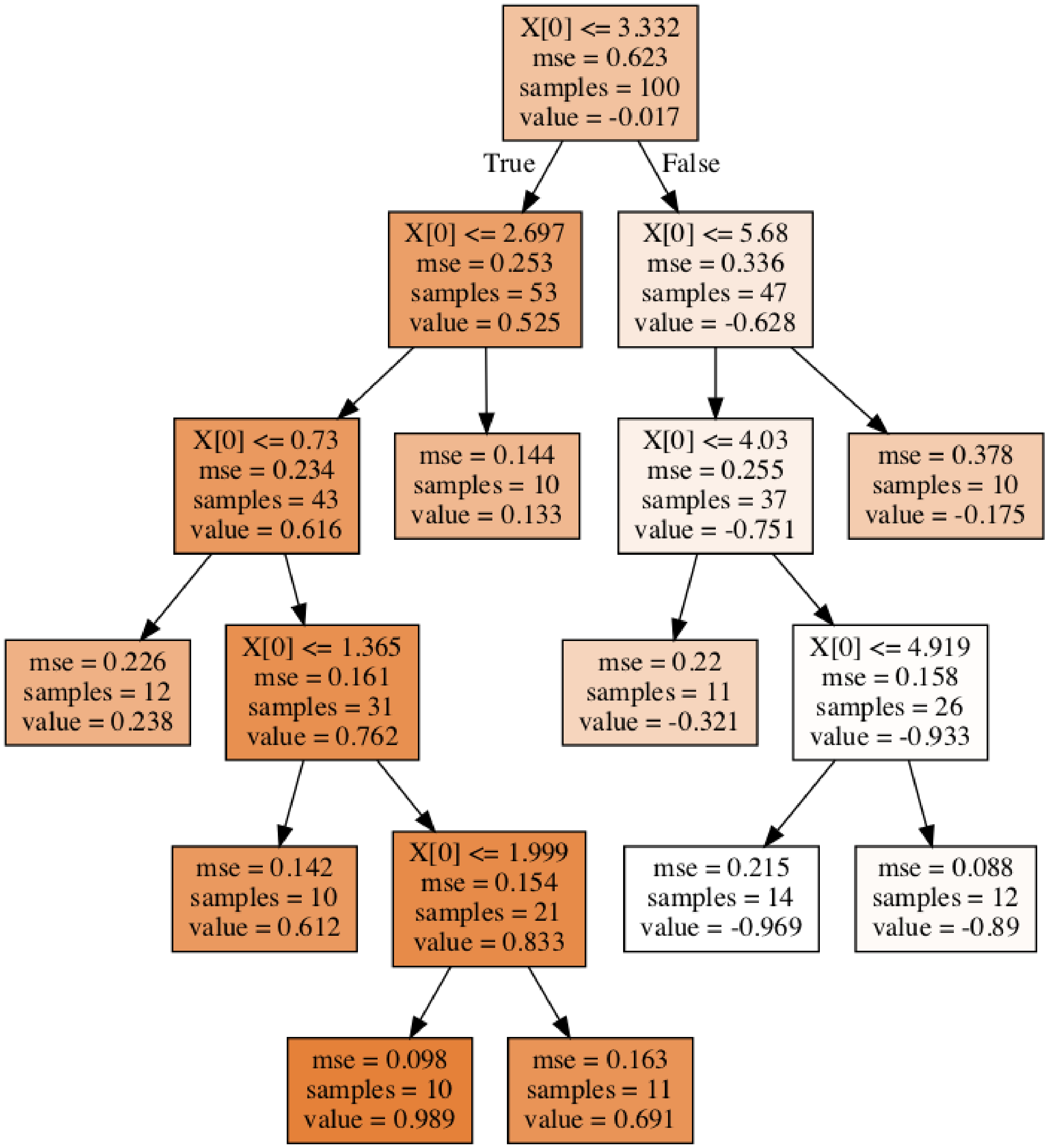
```
# Plot predicted values of y
ax.plot(df.x, dt_model2.predict(features), 'k*', label='model fit')
ax.legend(loc=1)
plt.show()
```



[113]:

```
# Export tree structure and save to Source object`
graph = Source( export_graphviz(dt_model2, filled=True, out_file=None) )
 ↪#comment out if you can't install graphviz

tree_png = graph.pipe(format='png') #comment out if you can't install graphviz
Image(tree_png) #comment out if you can't install graphviz
```

[113]:



Is this a good fit?

[114]:
```
dt_model2.tree_.children_left
```

[114]:
```
array([ 1,  2,  3, -1,  5, -1,  7, -1, -1, -1, 11, 12, -1, 14, -1, -1, -1])
```

[115]:
```
print('The mean squared error: ',mse(df.y,dt_model2.predict(features)))
```

```
The mean squared error:  0.18609158651134527
```

1.6.2 Maximum depth

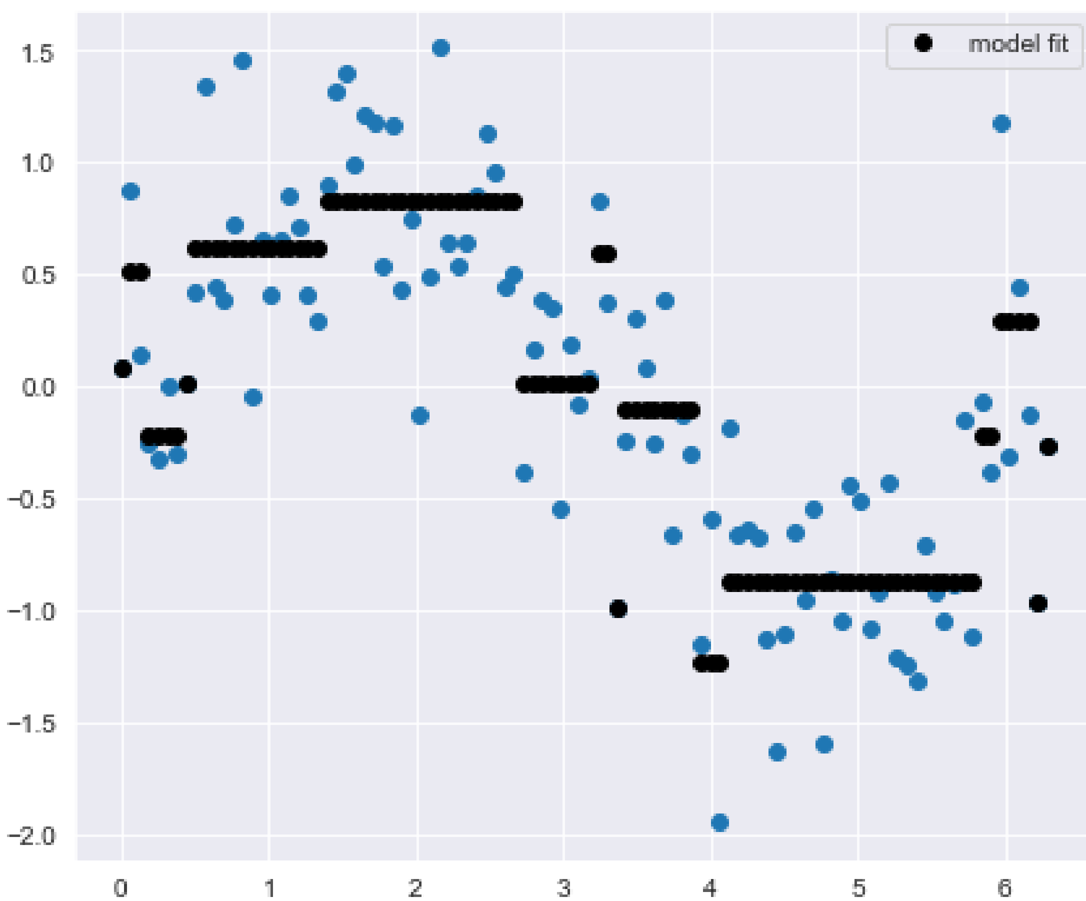
Here is another way to constrain a decision tree.

[116]:
```
dt_model3 = DecisionTreeRegressor(max_depth=4)
```

[117]:
```
# Fit model
dt_model3.fit(features, target)
```

[117]:
```
DecisionTreeRegressor(max_depth=4)
```

[118]:
```
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
# Plot original points
ax.scatter(df.x, df.y)
# Plot predicted values of y
ax.plot(df.x, dt_model3.predict(features), 'ko', label='model fit')
ax.legend(loc=1)
plt.show()
```

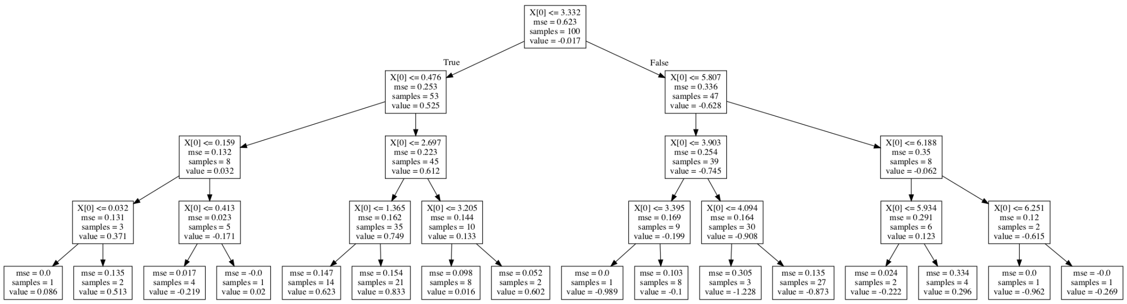


[119]:
```python
# Export tree structure and save to Source object`
graph = Source( export_graphviz(dt_model3, out_file=None) )
```

[120]:
```python
tree_png = graph.pipe(format='png')
```

[121]:
```python
Image(tree_png)
```

[121]:



How about this fit?

[122]:
```python
print('The mean squared error: ',mse(df.y,dt_model3.predict(features)))
```

```
The mean squared error:  0.13297571554901563
```

What can we say about depth vs samples for this example?

[123]:
```python
print('Model 1 has: ', dt_model.tree_.node_count, 'nodes.')
print('Model 2 has: ', dt_model2.tree_.node_count, 'nodes.')
print('Model 3 has: ', dt_model3.tree_.node_count, 'nodes.')
```

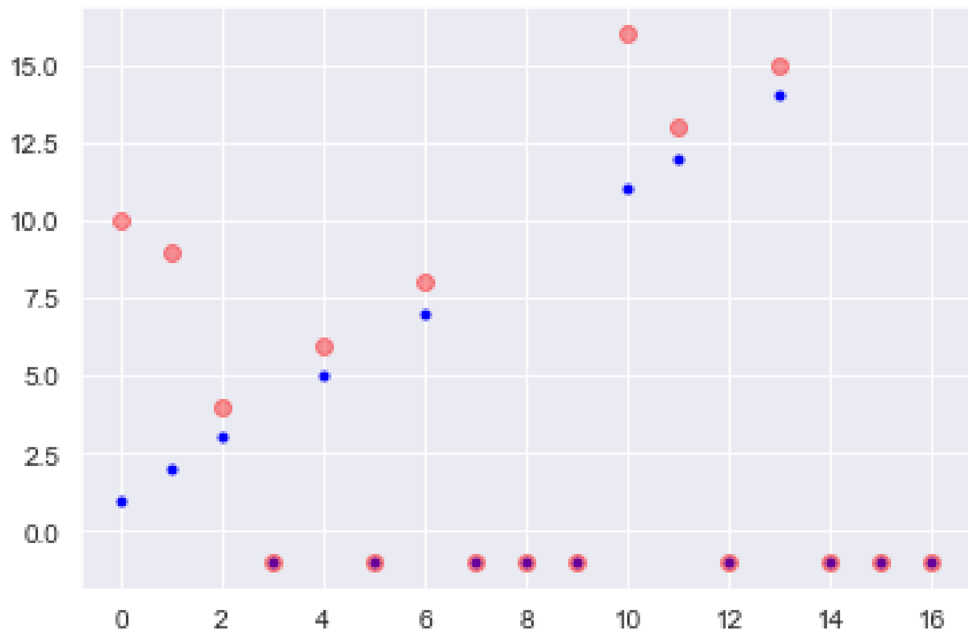
```
Model 1 has:  199 nodes.
Model 2 has:  17 nodes.
Model 3 has:  31 nodes.
```

[124]:
```python
print('Model 1 mse is: ', mse(df.y,dt_model.predict(features)))
print('Model 2 mse is: ', mse(df.y,dt_model2.predict(features)))
print('Model 3 mse is: ', mse(df.y,dt_model3.predict(features)))
```

```
Model 1 mse is:  0.0
Model 2 mse is:  0.18609158651134527
Model 3 mse is:  0.13297571554901563
```

[125]:
```python
plt.figure()
plt.plot(dt_model2.tree_.children_left, 'b.', label='left')
plt.plot(dt_model2.tree_.children_right, 'ro', alpha=0.4, label='right')
```

[125]: `[<matplotlib.lines.Line2D at 0x13d06ce80>]`



1.7 6. Random Forest Regressor

Random forests are ensemble models of decision trees. Let's start by importing our library.

[126]:
```
from sklearn.ensemble import RandomForestRegressor
```

Random forests need randomness, let's set the random_state to 123.

[127]:
```
rf =  RandomForestRegressor(random_state=123)
```

[128]:
```
# Fit the model.
rf.fit(features, target)
```

[128]:
```
RandomForestRegressor(random_state=123)
```

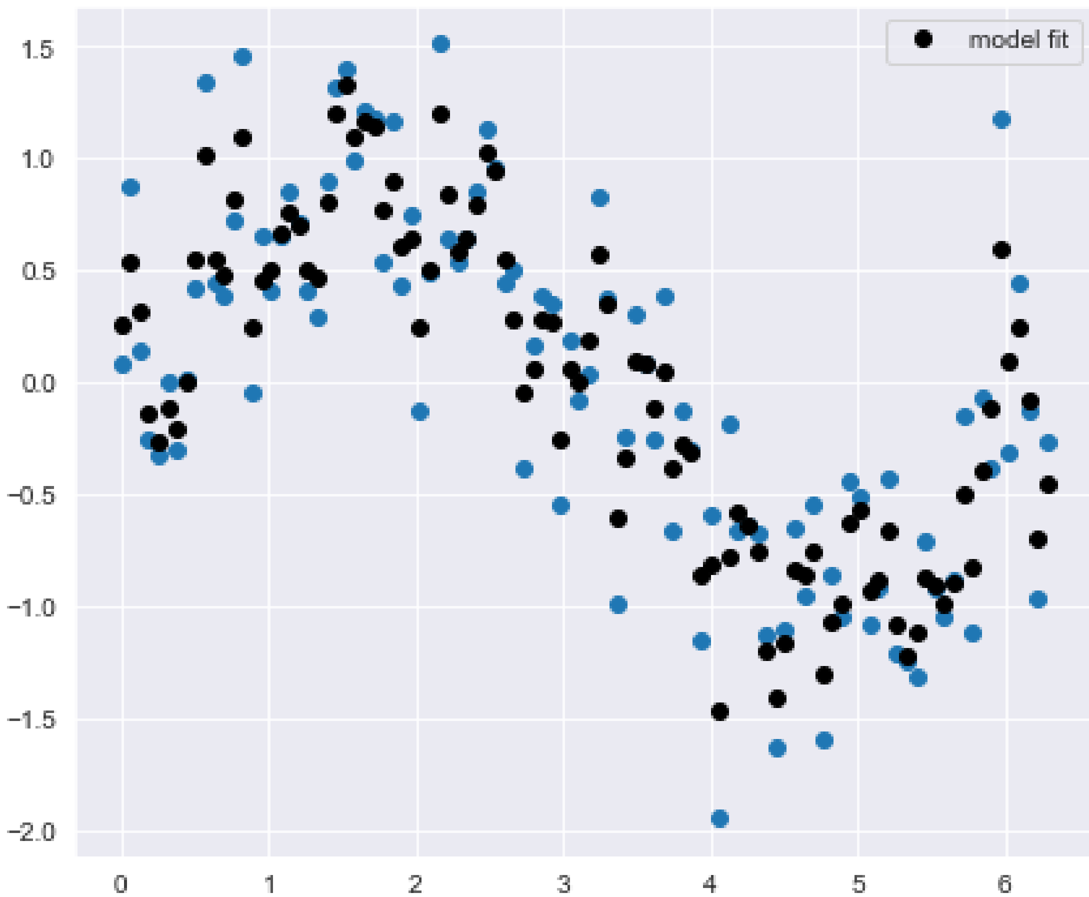
[129]:
```
# Number of decision trees in the random forest
print(len(rf.estimators_))

# Display first decision tree
print(rf.estimators_[60])
```

```
100
DecisionTreeRegressor(max_features='auto', random_state=395587690)
```

Let's plot the results.

[130]:

```
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
ax.scatter(df.x, df.y)
ax.plot(df.x, rf.predict(features), 'ko', label='model fit')
ax.legend(loc=1)
plt.show()
```



[131]:

```
graph1 = Source( export_graphviz(rf.estimators_[0], out_file=None) )
```

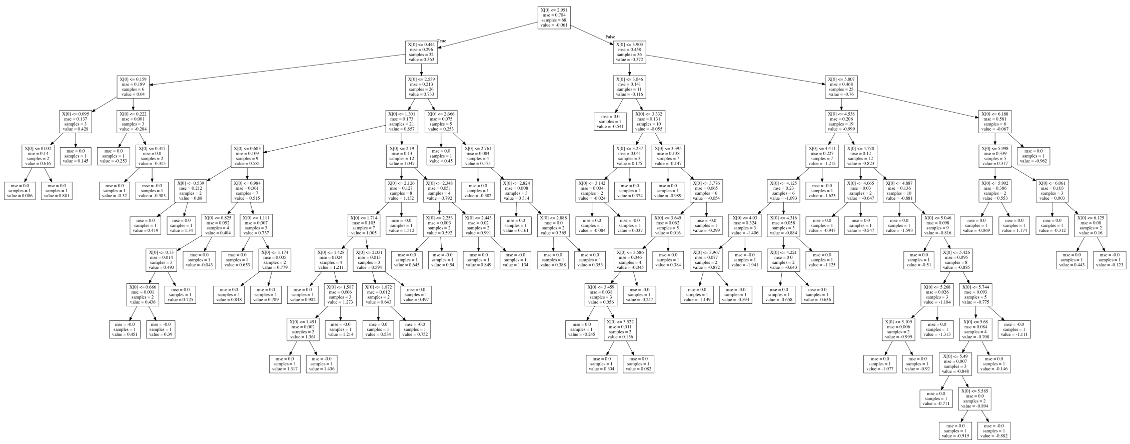
[132]:

```
tree_png1 = graph1.pipe(format='png')
```

[133]:

```
Image(tree_png1)
```

[133]:

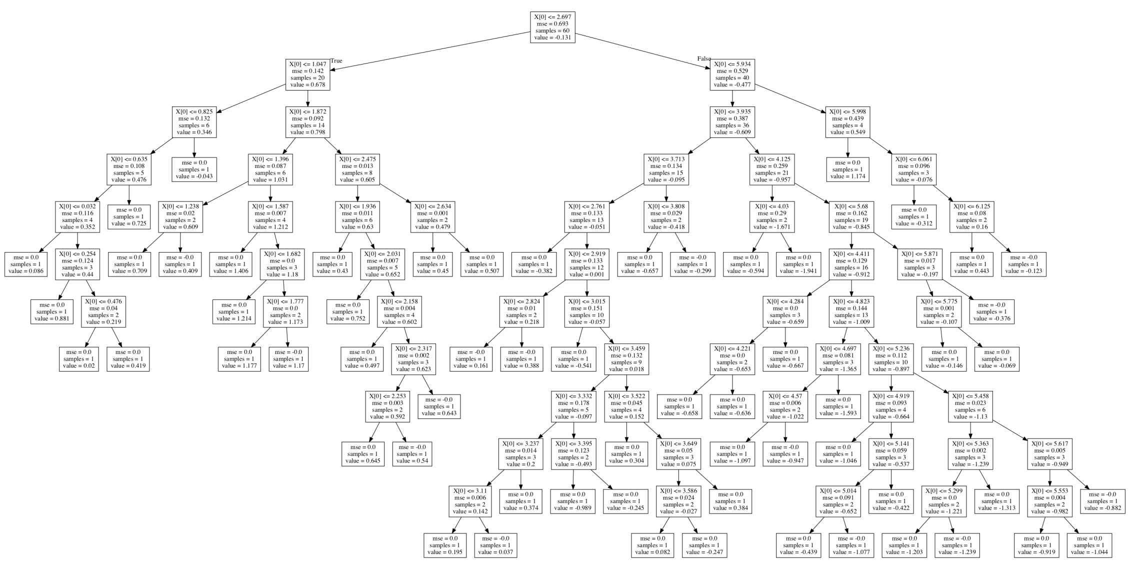


[134]:
```
graph2 = Source( export_graphviz(rf.estimators_[70], out_file=None) )
```

[135]:
```
tree_png2 = graph2.pipe(format='png')
```

[136]:
```
Image(tree_png2)
```

[136]:



[137]:
```
print('The mean squared error: ',mse(df.y,dt_model2.predict(features)))
```

```
The mean squared error:  0.18609158651134527
```

[138]:
```
rf_2 =  RandomForestRegressor(n_estimators=5, min_samples_leaf=3,␣
 ↪random_state=123)
# Fit the model.
rf_2.fit(features, target)
```

```
# Number of decision trees in the random forest
print(len(rf_2.estimators_))
```

5

[139]:
```
# Export tree structure and save to Source object`
graph = Source( export_graphviz(rf_2.estimators_[1], filled=True,
 ↪out_file=None) ) #comment out if you can't install graphviz

tree_png = graph.pipe(format='png') #comment out if you can't install graphviz
Image(tree_png) #comment out if you can't install graphviz
```

[139]:

[140]:

```
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
ax.scatter(df.x, df.y)
ax.plot(df.x, rf.predict(features), 'ko', label='n=100')
ax.plot(df.x, rf_2.predict(features), 'ro', label='n=5')
ax.legend(loc=1, fontsize=12)
ax.set_ylabel('y', fontsize=12)
ax.set_xlabel('x', fontsize=12)
```

```
plt.show()
```

1.7.1 Example with a more complex data set

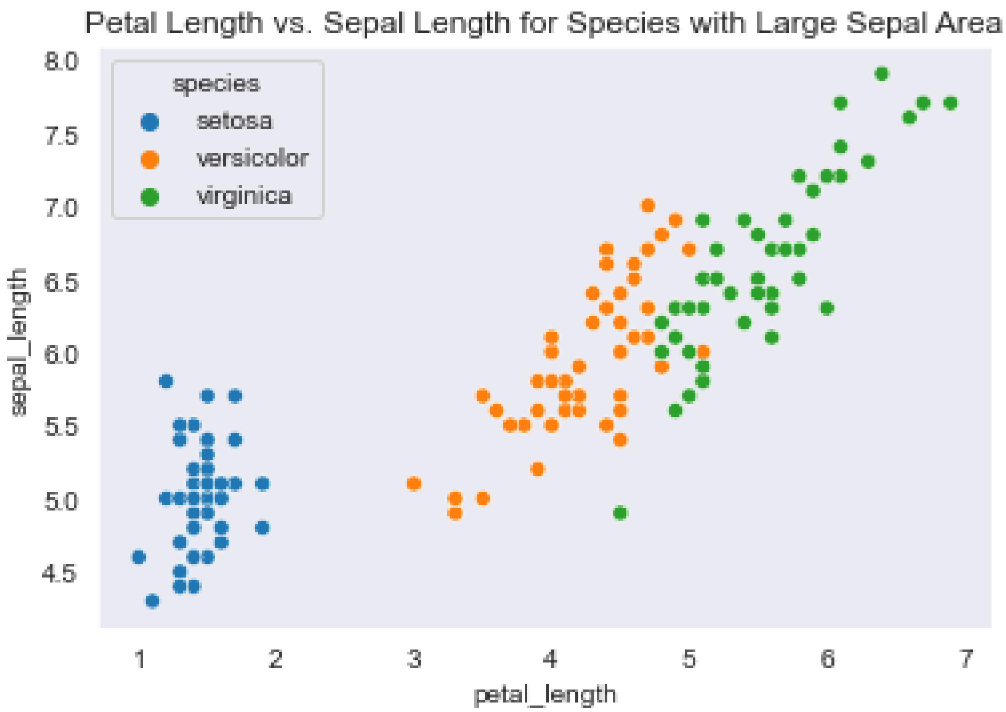
[145]:
```
from sklearn.datasets import load_iris
iris_df = pd.read_csv('./iris.csv')
```

[146]:
```
iris_df.head()
```

[146]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[147]:
```
fig, ax = plt.subplots(figsize=[6,4])
sns.scatterplot(ax=ax, x='petal_length', y='sepal_length', data=iris_df,␣
  ↪hue='species')
plt.grid()
plt.title('Petal Length vs. Sepal Length for Species with Large Sepal Area')
plt.show()
```



[148]:
```
versicolor_df = iris_df[iris_df.species=='versicolor']
```

[149]:
```
versicolor_df.head()
```

[149]:

	sepal_length	sepal_width	petal_length	petal_width	species
50	7.0	3.2	4.7	1.4	versicolor
51	6.4	3.2	4.5	1.5	versicolor
52	6.9	3.1	4.9	1.5	versicolor
53	5.5	2.3	4.0	1.3	versicolor
54	6.5	2.8	4.6	1.5	versicolor

[150]:
```
versicolor_df = versicolor_df.reset_index(drop=True)
```

[151]:
```
versicolor_df.head()
```

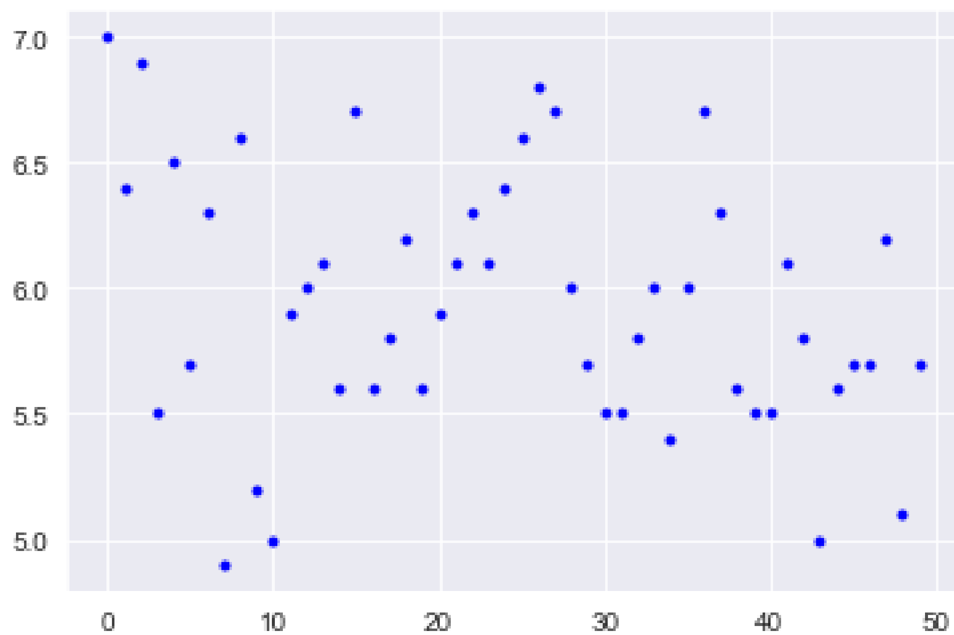
[151]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	7.0	3.2	4.7	1.4	versicolor
1	6.4	3.2	4.5	1.5	versicolor
2	6.9	3.1	4.9	1.5	versicolor
3	5.5	2.3	4.0	1.3	versicolor
4	6.5	2.8	4.6	1.5	versicolor

[152]:
```
y = versicolor_df['petal_length']
X = versicolor_df['sepal_length']
```

[153]:
```
fig, ax = plt.subplots()
ax.plot(X, 'b.')
```

[153]: [<matplotlib.lines.Line2D at 0x13dc056a0>]



[154]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
X_scaled = scaler.fit_transform(X.values.reshape(-1,1))
```

[155]:
```
fig, ax = plt.subplots()
ax.plot(X, 'b.', label='X')
ax.plot(X_scaled, 'r.', label='X_scaled')
```

[155]: [<matplotlib.lines.Line2D at 0x13dc46c70>]

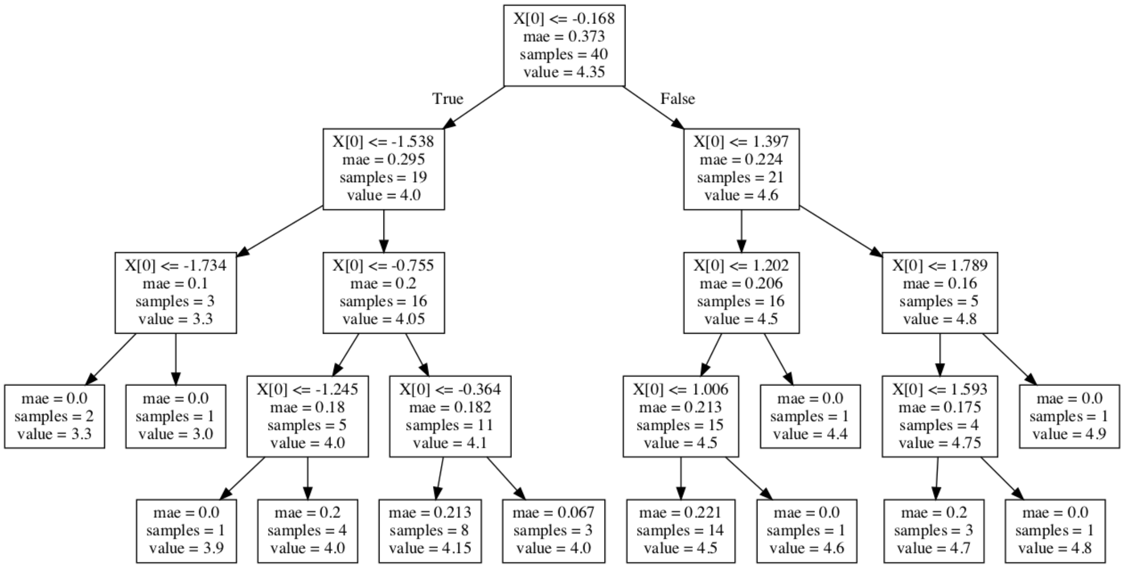
[156]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train,y_test=train_test_split(X_scaled, y, test_size=0.20,␣
↪random_state=123)
```

[157]:
```
dt_estimate = DecisionTreeRegressor(criterion='mae', max_depth=4)
dt_estimate.fit(X_train, y_train)
```

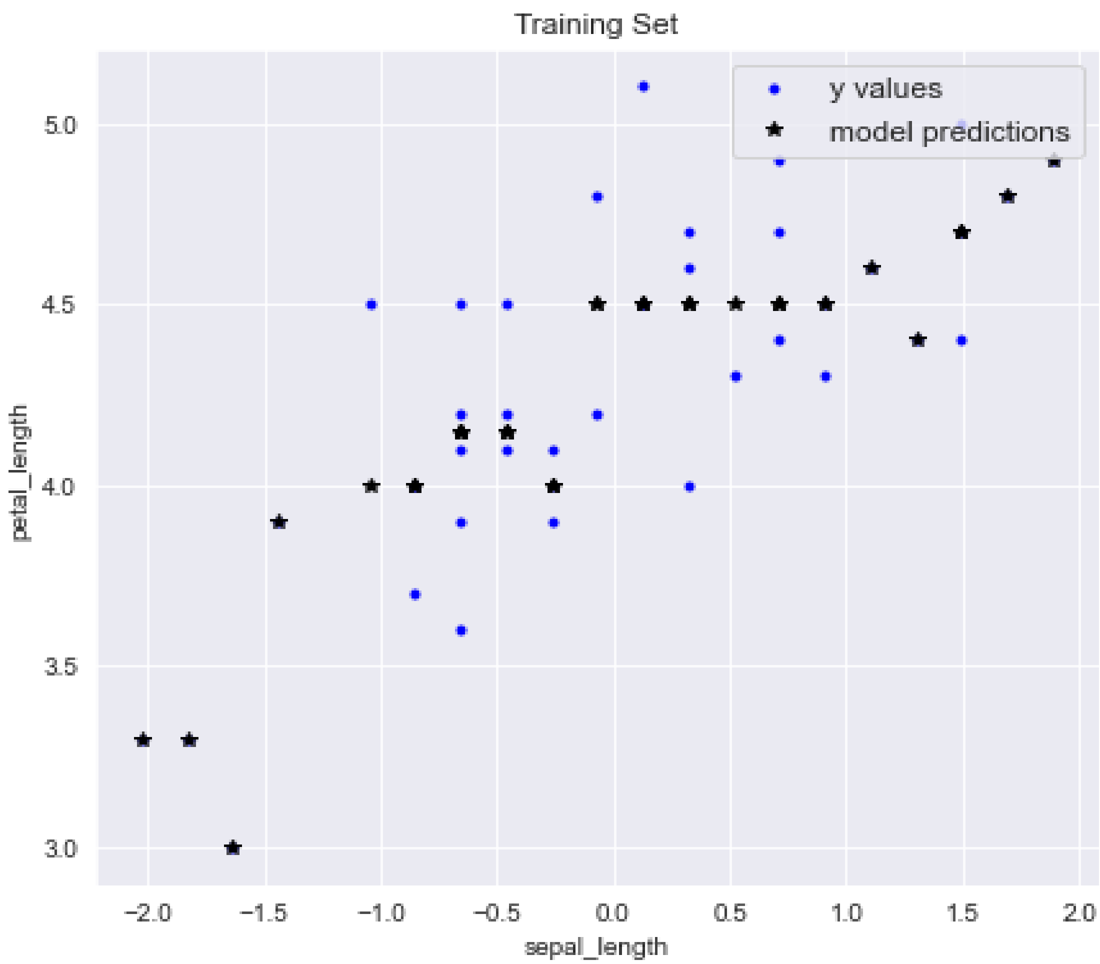
[157]: DecisionTreeRegressor(criterion='mae', max_depth=4)

[158]:
```
# Export tree structure and save to Source object`
graph = Source( export_graphviz(dt_estimate, out_file=None))
tree_png = graph.pipe(format='png')
Image(tree_png)
```

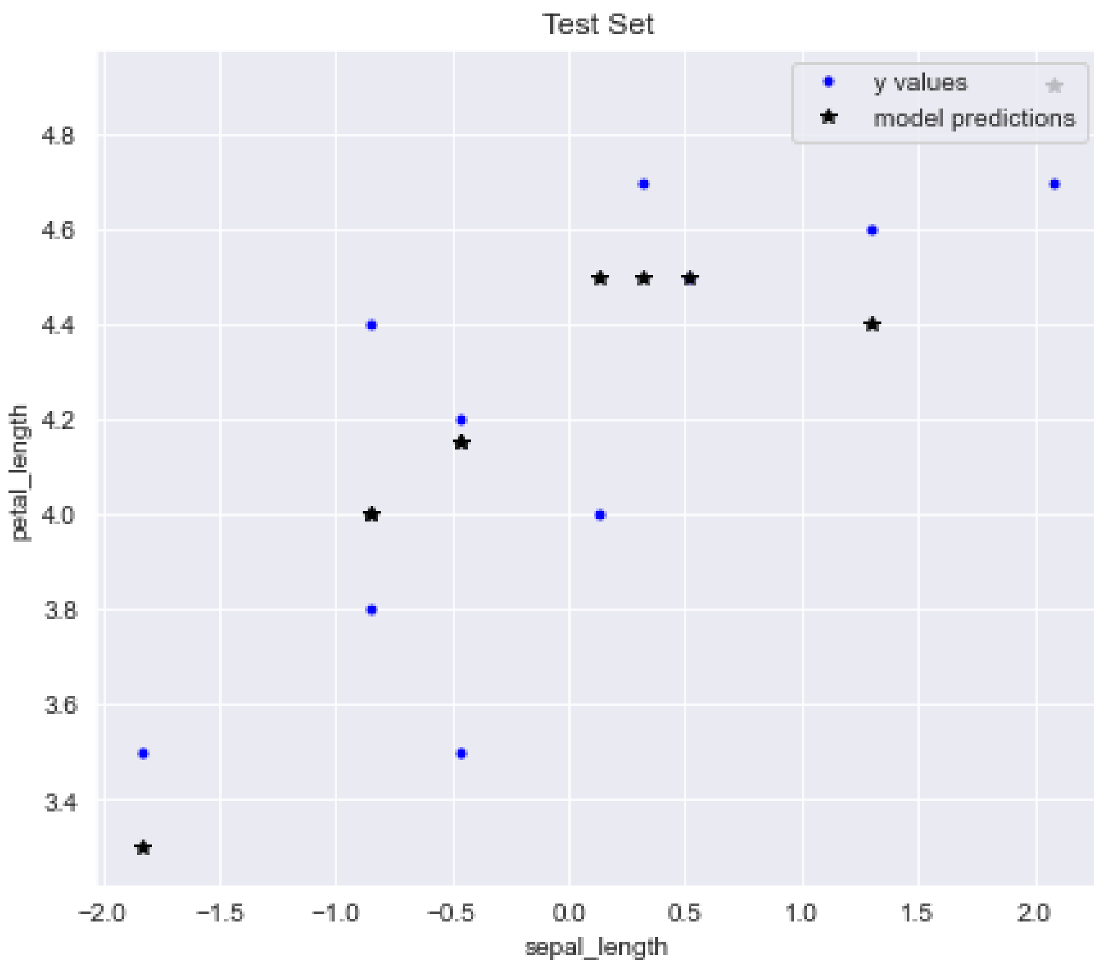
[158]:



[159]:
```python
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
# Plot original points
ax.plot(X_train, y_train, 'b.', label='y values')
# Plot predicted values of y
ax.plot(X_train, dt_estimate.predict(X_train), 'k*', label='model predictions')
ax.legend(loc=1, fontsize=12)
ax.set_ylabel('petal_length')
ax.set_xlabel('sepal_length')
ax.set_title('Training Set')
plt.show()
```



[160]:
```
fig = plt.figure(figsize=[7,6])

ax = plt.subplot(111)
# Plot original points
ax.plot(X_test, y_test, 'b.', label='y values')
# Plot predicted values of y
ax.plot(X_test, dt_estimate.predict(X_test), 'k*', label='model predictions')
ax.legend(loc=1)
ax.set_ylabel('petal_length')
ax.set_xlabel('sepal_length')
ax.set_title('Test Set')
plt.show()
```




```
[161]: len(y_test), len(dt_estimate.predict(X_test))
```

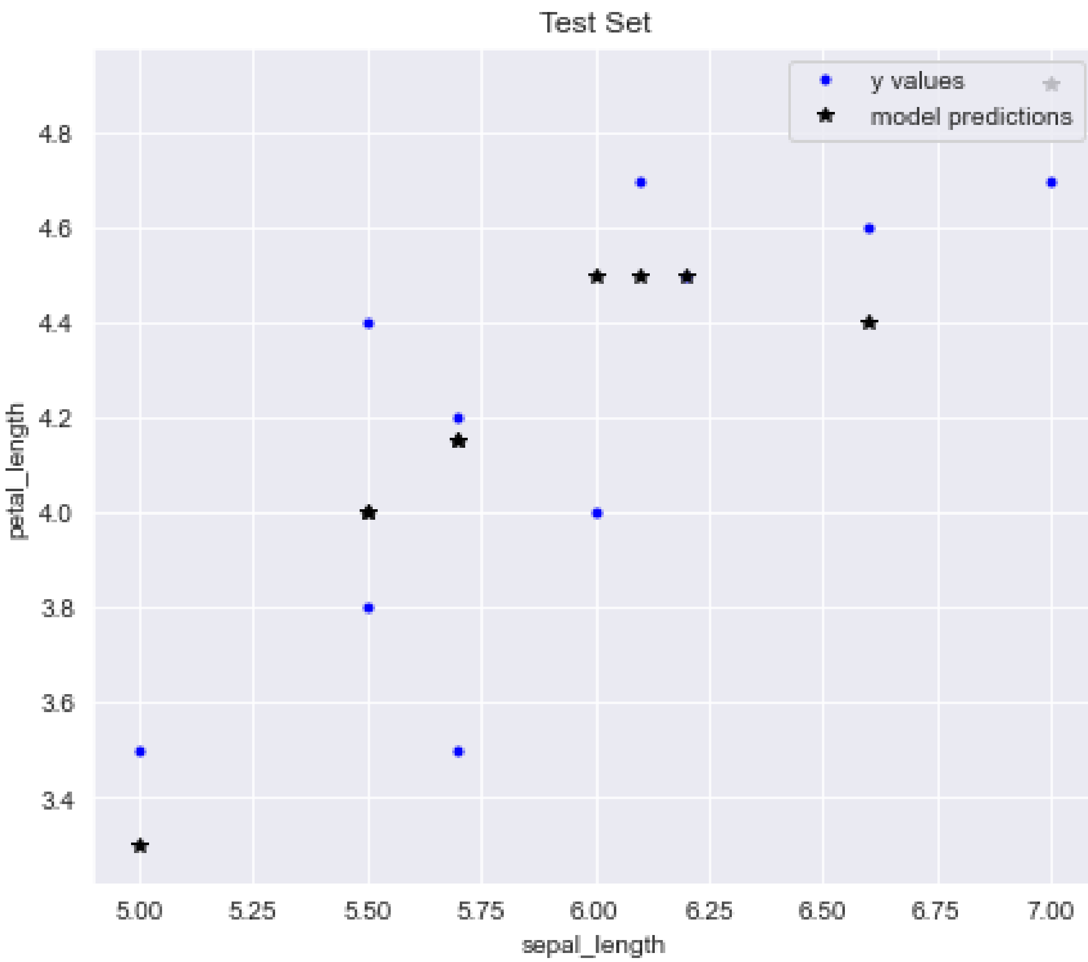
```
[161]: (10, 10)
```

What is wrong in these plots?

```
[162]: X_plot_train = scaler.inverse_transform(X_train)
       X_plot_test = scaler.inverse_transform(X_test)
```

```
[163]: fig = plt.figure(figsize=[7,6])

       ax = plt.subplot(111)
       # Plot original points
       ax.plot(X_plot_test, y_test, 'b.', label='y values')
       # Plot predicted values of y
       ax.plot(X_plot_test, dt_estimate.predict(X_test), 'k*', label='model␣
        ↪predictions')
```

```
ax.legend(loc=1)
ax.set_ylabel('petal_length')
ax.set_xlabel('sepal_length')
ax.set_title('Test Set')
plt.show()
```



[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

[ ]:

```

```

```
[ ]:
```

```
[ ]:
```