In [2]:
```
# My standard magic !  You will see this in almost all my notebooks.

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Reload all modules imported with %aimport
%load_ext autoreload
%autoreload 1

%matplotlib inline
```

In [3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

import recipe_helper
%aimport recipe_helper
```

Linear models and matrix notation

Our visualization of the data suggests that a reasonable first hypothesis is a linear relation.

$$h_\Theta(\mathbf{x}) = \Theta_0 + \Theta_1 \mathbf{x}$$

In our toy example there is only a single feature so $n = ||\mathbf{x}|| = 1$

so that our linear hypothesis can be re-written as

$$h_\Theta(\mathbf{x}') = \hat{\mathbf{y}} = \Theta^T \cdot \mathbf{x}'$$

Notice that the Cost function (optimization objective, evaluated over the training set) mirrors our Performance Measure (evaluted over the *test* set).

Hopefully, any Θ which does a good job on the training set also does a good job on the test set.

The critical assumptions are

- that the training and test sets are samples from the same underlying distribution
- m is sufficiently large so that a Θ estimated over the training set generalizes to unseen data

For now, the Cost function and the Performance Measure happen to be identical in form

- *but* one is evaluted over the training set; the other over the test set.

Linear Model with higher order features

Our error analysis of the toy problem suggested that a straight line was perhaps not the best fit

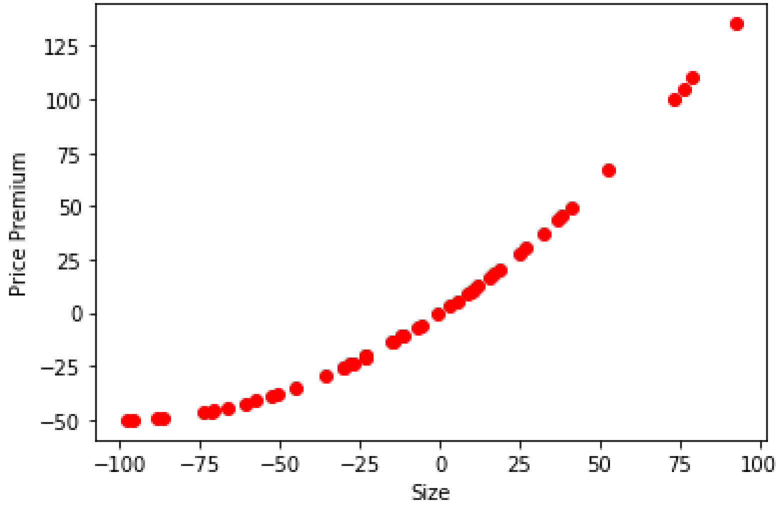
- positive errors in the extremes
- negative errors in the center

Perhaps a "curve" would be a better hypothesis ? What if our data is not linear ?

Here's what the first dataset looked like

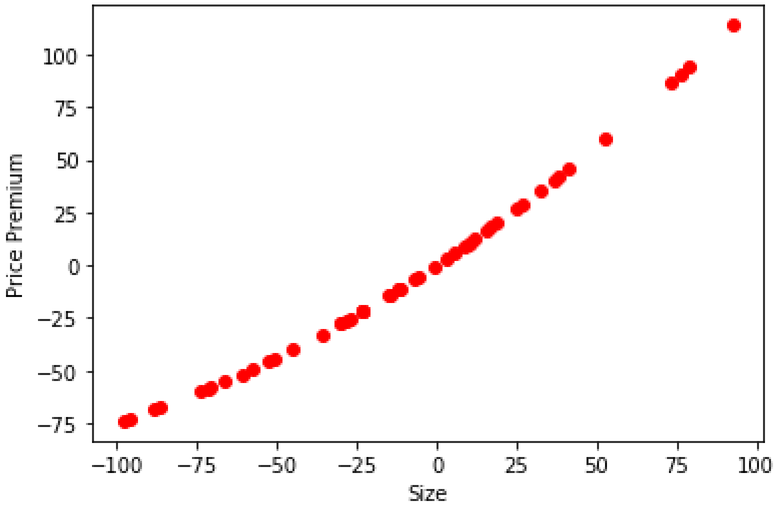
In [4]:

```
(xlabel, ylabel) = ("Size", "Price Premium")

# I will give you the data via a function (so I can easily alter the data in sub
sequent examples)
v1, a1 = 1, .005
lin = recipe_helper.Recipe_Helper(v = v1, a = a1)
X_lin, y_lin = lin.gen_data(num=50)


v2, a2 = v1, a1*2
curv = recipe_helper.Recipe_Helper(v = v2, a = a2)
X_curve, y_curve = curv.gen_data(num=50)
_= curv.gen_plot(X_curve,y_curve, xlabel, ylabel)
```



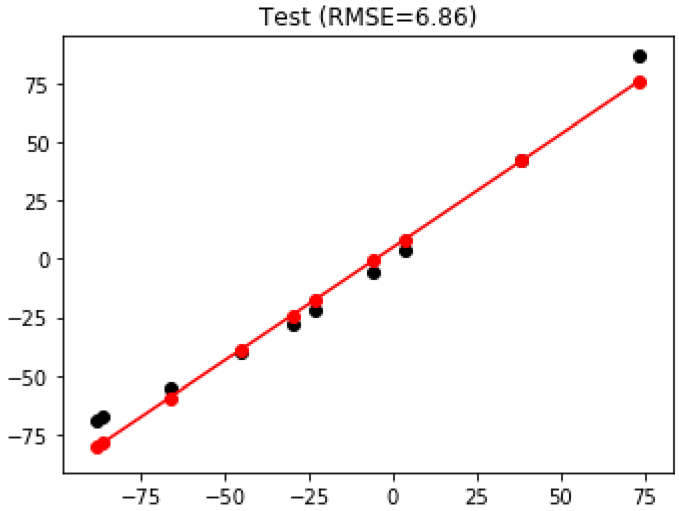
In [5]:

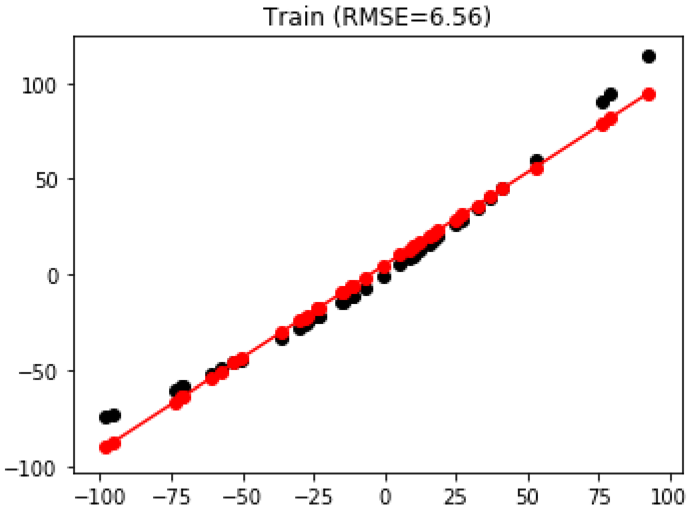
```
X_orig, y_orig = X_lin, y_lin

_ = lin.gen_plot(X_orig, y_orig, xlabel, ylabel)
```



In [6]:
```
_= lin.run_regress(X_orig, y_orig)
```

```
Coefficients:
 [4.93224426] [[0.96836946]]


R-squared (test): 0.98
Root Mean squared error (test): 6.86


R-squared (train): 0.98
Root Mean squared error (train): 6.56
```

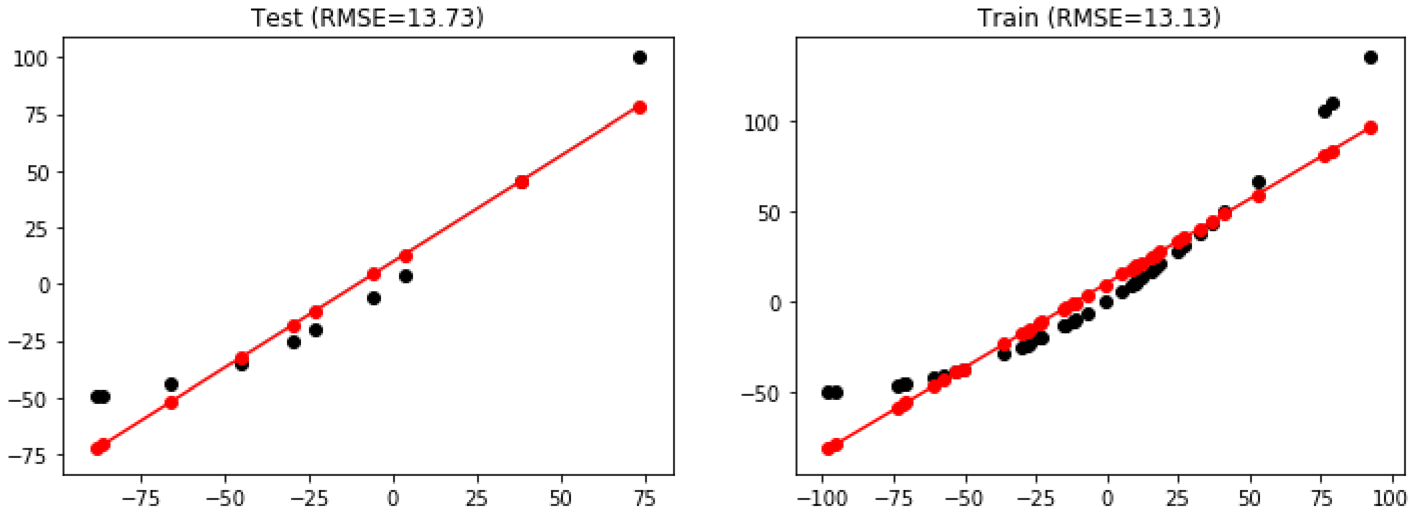




We will make our point by creating a similar dataset (the "curvy" dataset) that exagerates the curvature.

In [7]:
```
_= curv.run_regress(X_curve, y_curve)
```

```
Coefficients:
 [9.86448852] [[0.93673892]]


R-squared (test): 0.91
Root Mean squared error (test): 13.73


R-squared (train): 0.91
Root Mean squared error (train): 13.13
```



Compared to the original, the "curvy" data set has a lot more curvature

- the R^2 is still over 90%
- but the Performance Metric (RMSE) is twice as big

Curvature in a linear model

Our (first-order) linear model was

$$y = \Theta_0 + \Theta_1 x$$

We can create a *second order* linear model by adding a feature x^2:

$$y = \Theta_0 + \Theta_1 x + \Theta_2 x^2$$

y is a second order polynomial, whose plot is a curve

- but it is linear in features x, x^2

In other words, we are performing feature iteration

- in this case: adding the missing feature x^2

Let's modify $\mathbf{x}^{(\mathbf{i})}$ from a vector of length 1:

$$\mathbf{x^{(i)}} = (\mathbf{x_1^{(i)}})$$

to a vector of length 2:

$$\mathbf{x^{(i)}} = (\mathbf{x_1^{(i)}}, \mathbf{x_1^{(i)}}^2)$$

by adding a squared term to the vector $\mathbf{x^{(i)}}$, for each i.

The modified $\mathbf{X}'$ becomes:

$$\mathbf{X} = \begin{pmatrix} 1 & \mathbf{x}_1^{(1)} & (\mathbf{x}_1^{(1)})^2 \\ 1 & \mathbf{x}_1^{(2)} & (\mathbf{x}_1^{(2)})^2 \\ \vdots & \vdots & \\ 1 & \mathbf{x}_1^{(m)} & (\mathbf{x}_1^{(m)})^2 \end{pmatrix}$$

Note that this modified $\mathbf{X}'$ fits perfectly within our Linear hypothesis

$$\hat{\mathbf{y}} = \mathbf{X}'\Theta$$

The requirement is that the model be linear in its *features*, **not** that the features be linear !

In [9]:

```
print("Done !")
```

```
Done !
```