Regression: the importance of Transformations/Feature Engineering

Recall from our initial notebook on Linear Regression

- adding a *synthetic feature* x_1^2 (Size squared) greatly improved the Performance Metric

Feature Engineering involves

- taking *raw* features from the training dataset
- applying *transformations* to create *synthetic* features
- resulting in
    - additional synthetic features
    - removing un-informative raw features

Our toy example was just a teaser for the importance of Feature Engineering

- WIll be a subject of subsequent modules

Suffice it to say

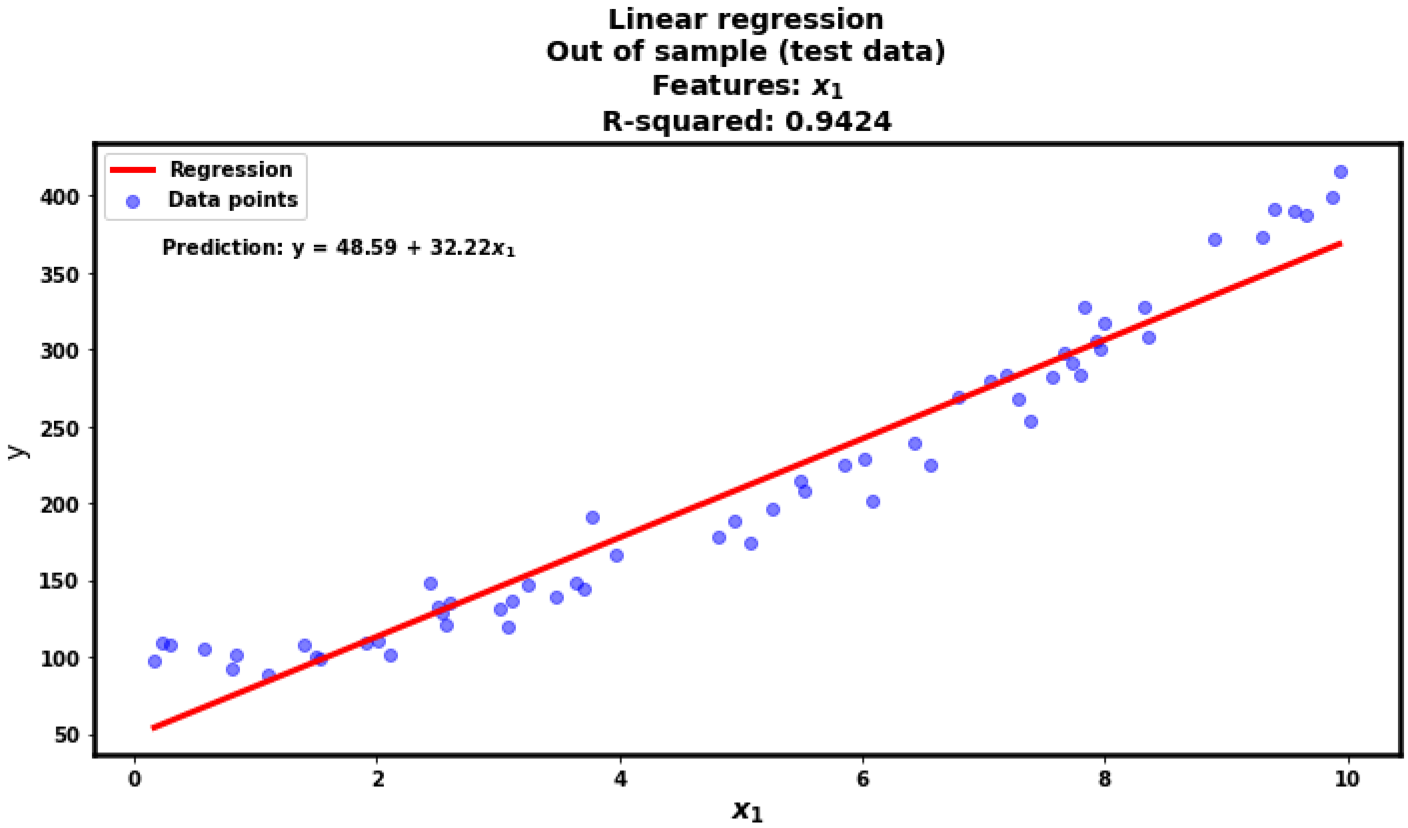
- a skill that distinguishes a good Data Scientist from just an average one
- is the ability to understand
    - aided by Exploratory Data Analysis
- what synthetic features
    - need to be created in order to increase the Performance Metric

Regression task: other models

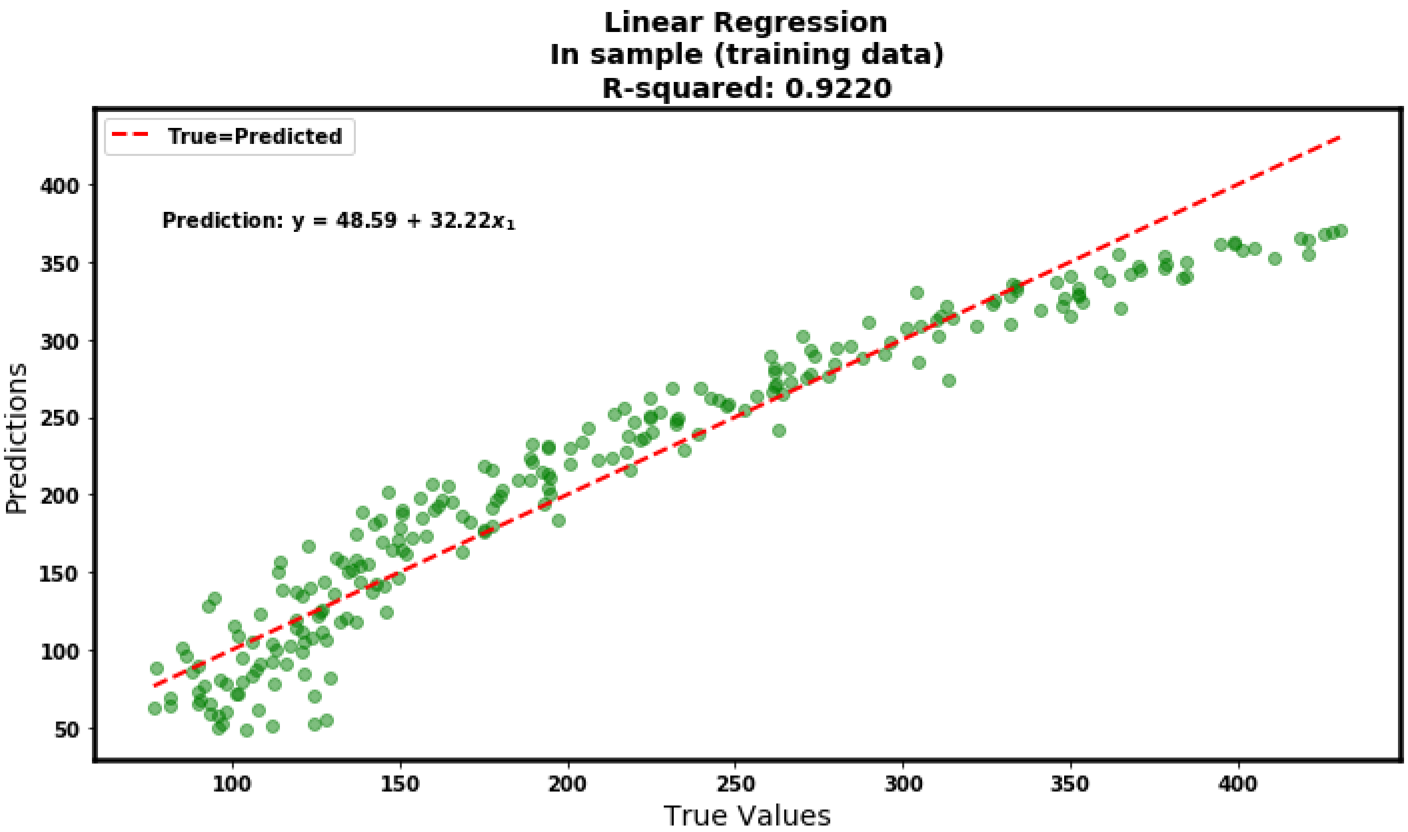
There is rarely a single model for solving a task in Machine Learning.

Here, we present the single feature Linear Regression model from the previous module.

In [8]:
```
fig, ax = lr_demo.plot(model, y_test, X_test[:,[0] ], feature_names=["$x_1$"],
                       title="Linear regression\nOut of sample (test data)\n",
                       showTrue=False)
```



In [9]:
```
fig, ax =  lr_demo.plot_pred_vs_true(model, y_train, X_train[:,[0] ], feature_names=["$x_1$"],
                                     title="Linear Regression\nIn sample (training data)\n")
```



And now, the Regression task

- on the same dataset
- solved by a different model: the Decision Tree Regressor

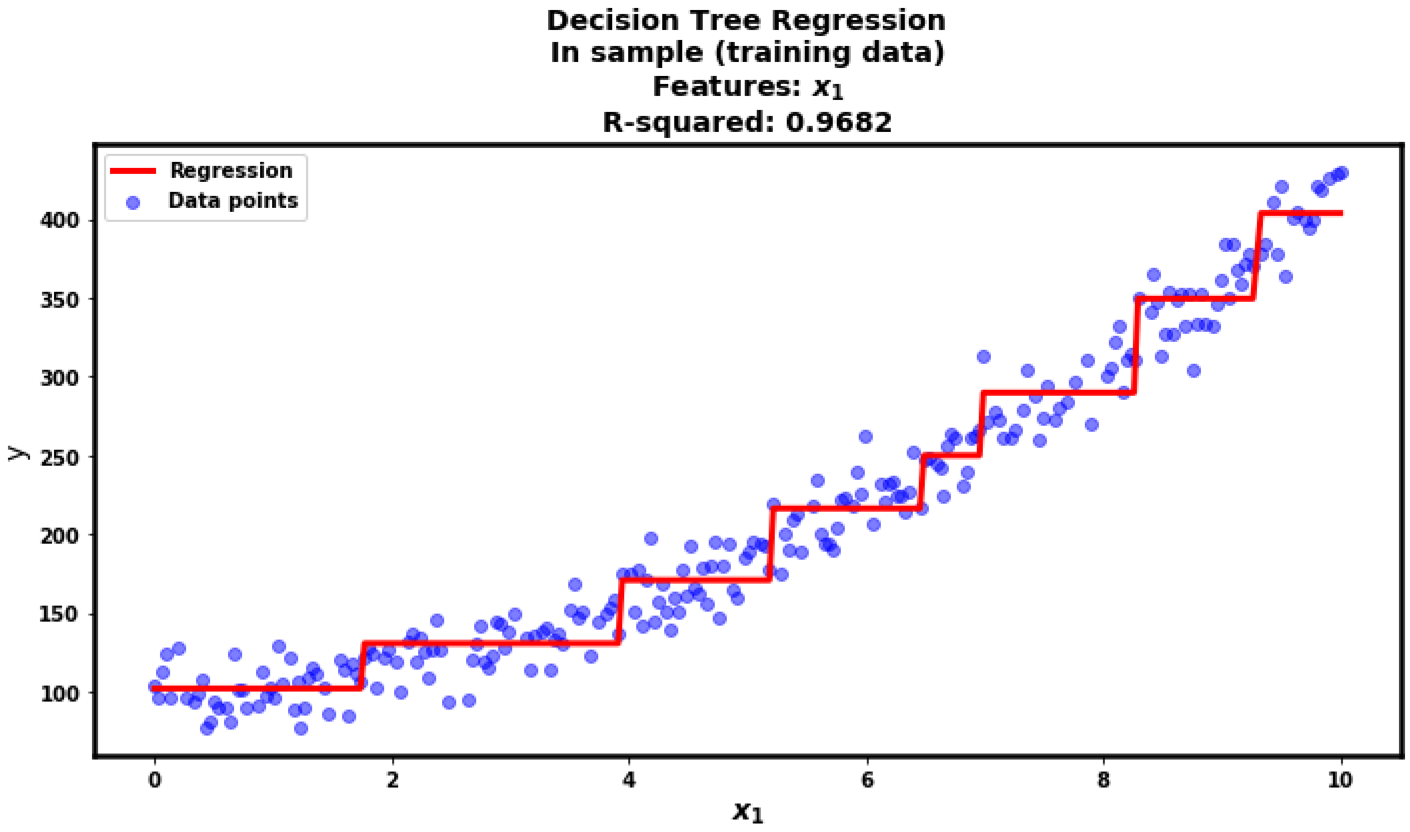
In [10]:

```
from sklearn.tree import DecisionTreeRegressor

# Create and fit the Decision Tree model
dt_model = DecisionTreeRegressor(max_depth=3)

# Note: we access the single feature as X_train[:, [9]] rather than X_train[:,
0] to ensure the result is 2D
# - with a singlteon final dimension, as required by sklearn
_= dt_model.fit(X_train[:,[0] ], y_train)
```

In [11]:
```
fig, ax = lr_demo.plot(dt_model, y_train, X_train[:,[0] ], feature_names=["$x_1$"],
                       title="Decision Tree Regression\nIn sample (training data)\n",
                       showEquation=False,
                       showTrue=False)
```



In [12]:
```
fig, ax =  lr_demo.plot_pred_vs_true(dt_model, y_train, X_train[:,[0] ], feature_names=["$x_1$"], showEquation=False,
                                     title="Decision Tree Regression\nIn sample (training data)\n")
```

The Decision Tree model creates a more complex prediction

- divides the feature x_1 into regions
- predicts the average target (from *training examples*) as the target for *all* values in the same region

So how do we choose between multiple models for a task ?

One argument: use the model with the best Performance Metric

- our goal is good out of sample prediction

But the prediction for Decision Tree presents some issues

- complex.
    - Complexity can lead to over fitting: good in sample performance, but poor out of sample performance
- all test examples within a region have the *same prediction*
    - may violate economic principles: more is better
- the prediction can not be summarized simply (i.e., in an equation)
    - we need a tree of questions to represent the decision

Ultimately

- the use case for the prediction may inform the choice of model

Why is Linear Regression so popular ?

Linear Regression is used very often, for a number of reasons.

First:

- the prediction fits a functional form: Linear
- the equation succinctly *explains* the prediction in terms of the features

Moreover, the weights (coefficients) in the Linear Regression equation are *interpretable*.

Consider

$$\hat{\mathbf{y}} = \mathbf{\Theta} \cdot \mathbf{x}$$

Lets take the derivative of the prediction with respect to any feature, e.g., the j^{th} feature

$$\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{x}_j} = \mathbf{\Theta}_j$$

That is

- the weight/coefficient Θ_j associated with the j^{th} feature
- is the *marginal increase in prediction* $\hat{\mathbf{y}}$ *for a unit increase in* $\mathbf{x}_j$

In our Housing Price prediction task

- we can use the coefficient of Size
- to predict how much the predicted Price will increase
- for each additional increase in Size

This advantage in interpretability often argues for using Linear Regression as the model for a Regression task

- we can test whether the sign of Θ_j conforms with economic intuition
    - it should be positive: bigger is costlier
- it is less of a "black box"

In [13]:

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

```
Done
```