# 

[**Intro**](#_wrtlbi64ve3n) **1**

[**Python Tools**](#_roztv3pr515p) **2**

[**So What is Linear Regression?**](#_8sf42jvgvey4) **4**

[**Data preparation**](#_o2pcx434q0f8) **6**

[Loading data](#_pdcdz3i2aedp) 7

[Visualising data](#_99i2ns9m9q1b) 9

[**Formula**](#_6taok5pacp38) **10**

[**Training and Initialization**](#_il6pwcvt0i0y) **13**

[**Predictions**](#_usgxun7u6vrs) **14**

[**Let’s put it all together**](#_qewqpc3q5msr) **15**

[**Summary**](#_p108a33a154l) **18**

# 

# **Intro**

Let’s start this episode with a question… Why Linear Regression? My name is Ernesto Lee and in this episode we will begin our journey in Ernest… or Ernesto… no pun intended

Some of you may wonder, why the episodes about artificial intelligence begins with **basic Machine Learning algorithms** like Linear Regression. It’s actually very justifiable to start from Linear Regression. First of all, it is a very plain algorithm so the you can grasp an **understanding of fundamental Machine Learning concepts** like *Supervised Learning*, *Cost Functions*, and *Gradient Descent*. Additionally, after learning Linear Regression it is quite easy to understand Logistic Regression algorithm and believe or not it will serve as a great segway to Neural Networks. You can expect all of those and even more covered in the upcoming episodes!

# **Python Tools**

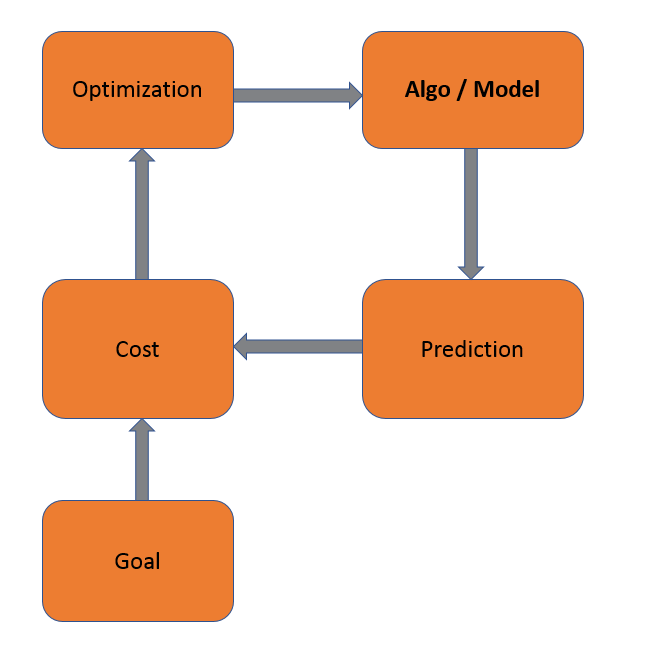
Let me introduce or reintroduce some of the **most popular libraries** that can be found in every Python based Machine Learning or Data Science related project.

* [NumPy](http://www.numpy.org/) — a library for scientific computing, perfect for Multivariable Calculus & Linear Algebra. It provides the [ndarray](https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.ndarray.html) class which can be compared to **a Python list that can be treated as a vector or matrix**.
* [Matplotlib](https://matplotlib.org/) — toolkit for **data visualisation**, allows you to create various 2d and 3d graphs.
* [Pandas](https://pandas.pydata.org/)—this library is a wrapper for Matplotlib and NumPy libraries. It provides [DataFrame](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) class. It **treats NumPy matrices as tables**, allowing access to rows and columns by their attached names. It is Very helpful in **data loading, saving, wrangling, and the data exploration process**. Provides an interface of functions that makes deployment faster.

Each library can be installed separately with using. They will be imported in the code of virtually every project in this series.

-------------------------------------

# **The Big Picture of Supervised Regression**



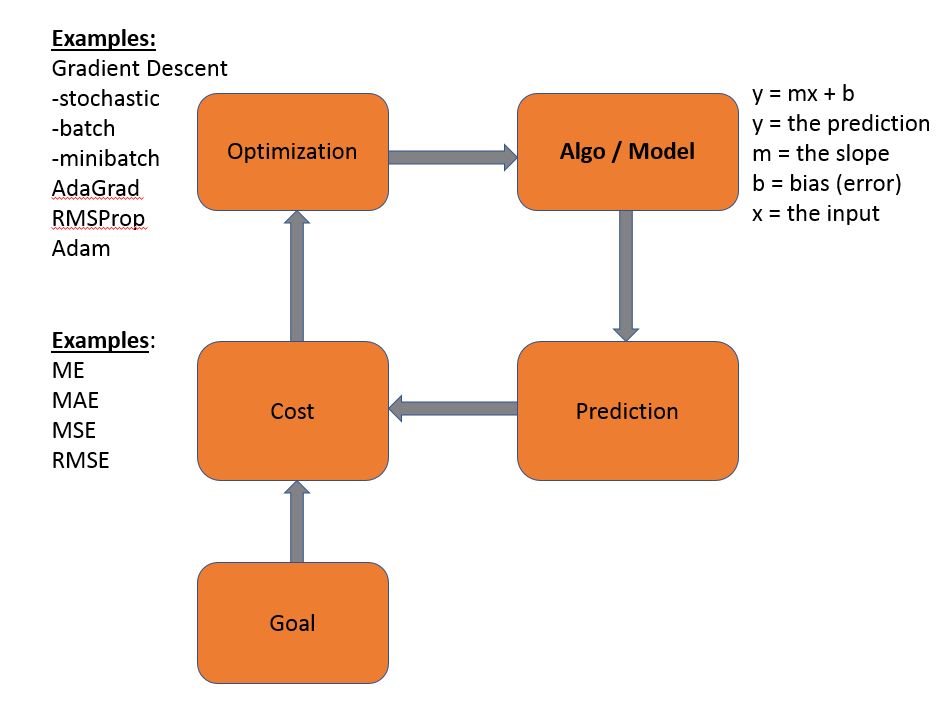
We start with an assumption about the shape of the data and we “guess” at an equation that we think would approximate the data. We do not know all of the coefficients that would generalize the algorithm however.

We then take TRAINING data and use the equation/algo to make a PREDICTION.

After that you need to calculate how far off you are from the actual value that you predicted. That is the COST.

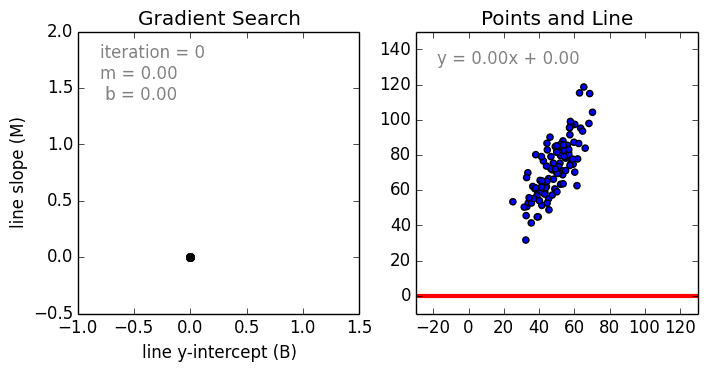
Now here is where the “learning” takes place. You need to OPTIMIZE your Model by updating the coefficients so you take the ERROR from the COST function and the GOAL and you use that data to give you a better estimate of your coefficients.

Finally, rinse, lather, and repeat.



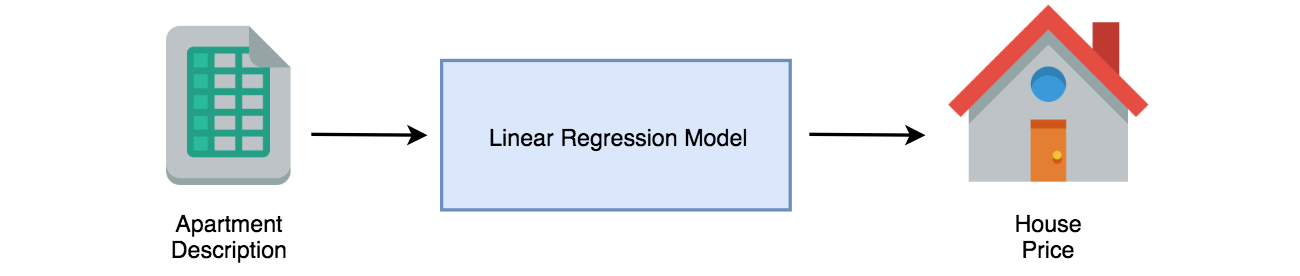
# **So What is Linear Regression?**

It’s a **Supervised Learning algorithm** with the goal being to **predict continuous, numerical values based on given data input**. From a geometrical perspective, each data sample is a point. With Linear Regression, it tries to **find parameters of the linear function**, so the **distance between all the points and the line is as small as possible**. Algorithm used for parameters update is called **Gradient Descent**. Gradient Descent is an algorithm used by algorithms and is also called an Optimizer function.

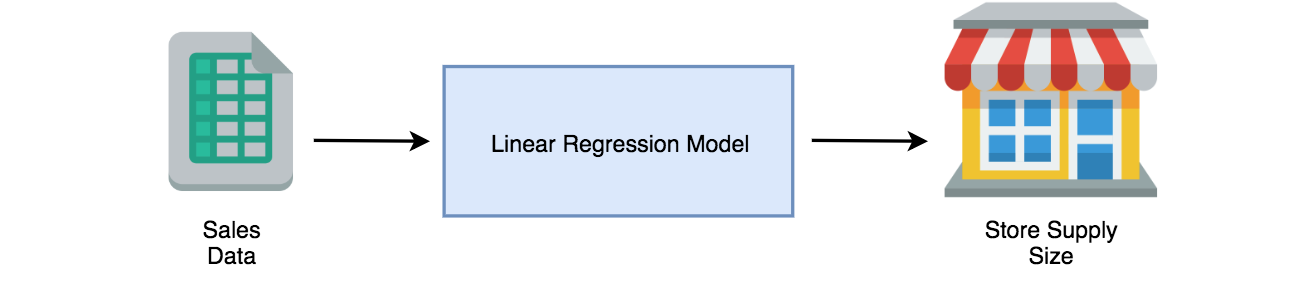


Training of Linear Regression model. The left graph displays the change of linear function parameters over time. The plot on the right renders the linear function using current parameters (source: [Siraj Raval GitHub](https://github.com/llSourcell/linear_regression_live)).

For example, if I have a dataset consisting of houses properties and their prices in some specific area, Linear Regression algorithm can be used to find a mathematical function which will try to estimate the value of different houses (outside of the dataset), based on its attributes.



Another example can be a prediction of a food supply size for the grocery store, based on sales data. Think Walmart… or FedEx when it comes to purchasing snow tires… That way the business can decrease unnecessary waste. **Such mapping is achievable for any correlated input-output data pairs.**

****

# **Data preparation**

Before coding Linear Regression, it would be good to have a real problem to solve. It is possible to find a lot of datasets on websites like [UCI Repository](https://archive.ics.uci.edu/ml/index.php) or [Kaggle](https://www.kaggle.com/). I would like to use a dataset that ties real estate properties to price. Let’s use the King County Real Estate Dataset ( <https://www.kaggle.com/harlfoxem/housesalesprediction> )

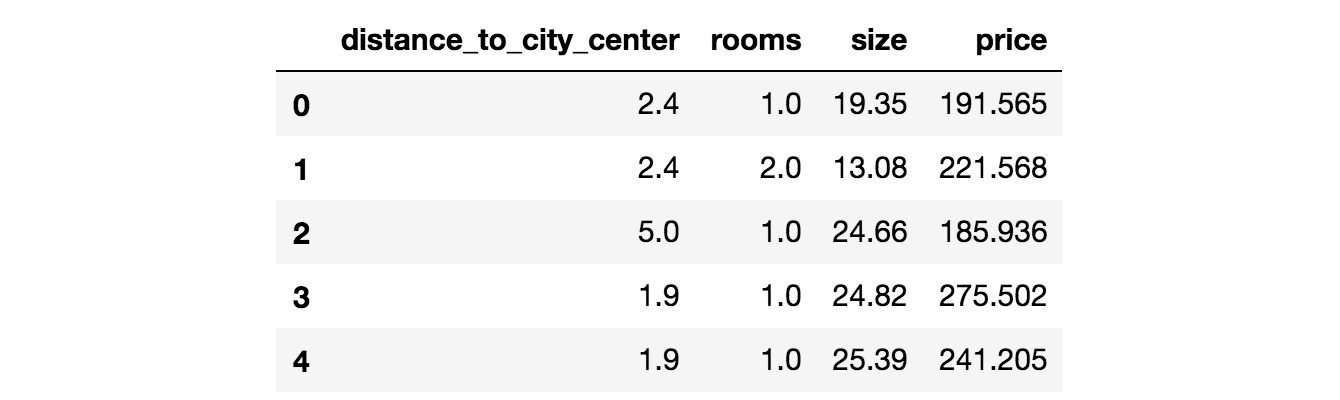
Let’s start by looking at the data: id, Date, **price**, bedrooms, bathrooms, sqft\_living, sqft\_lot, floors, waterfront, view, condition, grade, sqft\_above, sqft\_basement, yr\_built, yr\_renovated, zipcode, lat, long, **sqft\_living15**, sqft\_lot15

The goal is to **train a Linear Regression model capable of predicting house prices** in Seattle.

## **Loading data**

Let’s start by reading data from the .csv file to DataFrame object of Pandas and displaying a few data rows. To achieve that [read\_csv](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html) function will be used. Data is separated with colon character which is why sep="," parameter was added. Function [head](https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.head.html) renders first five rows of data in the form of the pleasantly readable HTML table.

The output of the code looks as following:



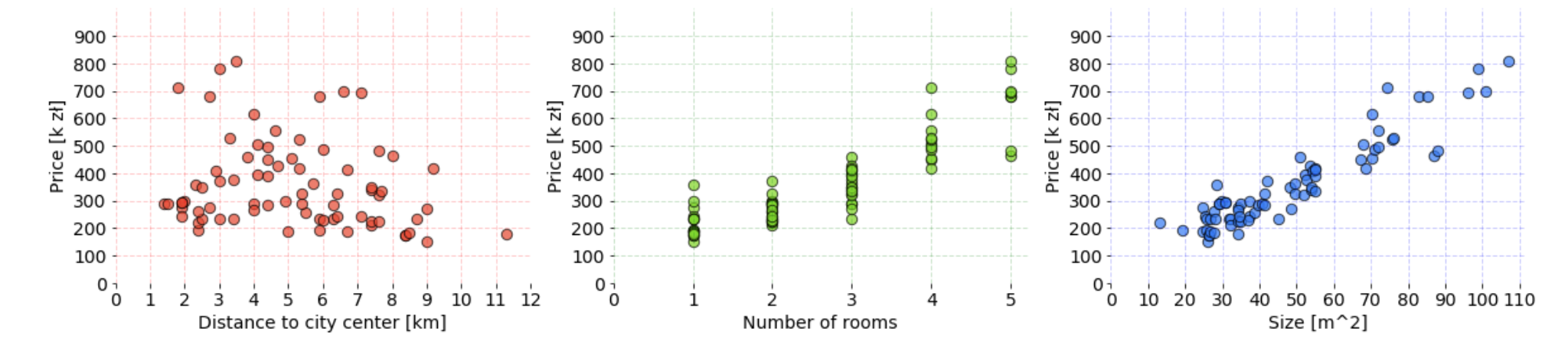
DataFrame visualisation in [Jupyter Notebook](http://jupyter.org/).

As presented in the table, there are **several features** describing house properties:

* **Bed rooms** - the number of bedrooms in the house - Bath Rooms as a separate data point
* **size** - the area of the house measured in square feet, the size of the lot
* **price** - target value (the one that needs to be predicted by the model), which is the cost of the house measured in dollars.
* Id : date : price : bedrooms : bathrooms : sqft\_living sqft\_lot : floors : waterfront : view : condition : grade : sqft\_above : sqft\_basement : yr\_built : yr\_renovated : zipcode : lat : long : sqft\_living : sqft\_lot

## **Visualising data**

It is very important to always understand the structure of data. The more features there are, the harder it is. In this case, a [scatter plot](https://en.wikipedia.org/wiki/Scatter_plot) is used to **display the relationship between target and training features**.



Charts shows the whole data from our dataset. It was prepared with Matplotlib library in Jupyter Notebook. The code used to create these charts can be found in your class [link](https://gist.github.com/FisherKK/0113b1eda361856a1cd29ad4fbd180d2).

Depending on what is necessary to show, some other types of visualization (e.g. [box plot](https://en.wikipedia.org/wiki/Box_plot)) and techniques could be useful (e.g. [clustering](https://en.wikipedia.org/wiki/Cluster_analysis)). Here, a **linear dependency between features can be observed** — with the increase of values on axis x, values on the y-axis are linearly increasing or decreasing accordingly. It’s great because if that was not the case (e.g. relationship would be exponential), then it would be hard to fit a line through all the points and different algorithm should be considered.

# **Formula**

The Linear Regression **model is a mathematical formula** that takes the **vector of numerical values** (attributes of single data sample) as an input and uses them to **make a prediction**.

Mapping the same statement in the context of the presented problem, there are many samples containing attributes of King County houses where each sample is a vector from mathematical perspective. Each **vector of features is paired with target value** (expected result from formula).



According to the algorithm, **every feature has a weight parameter assigned.** It represents it’s **importance** to the model. The goal is to find the values of weights so the following equation is met for every house data.

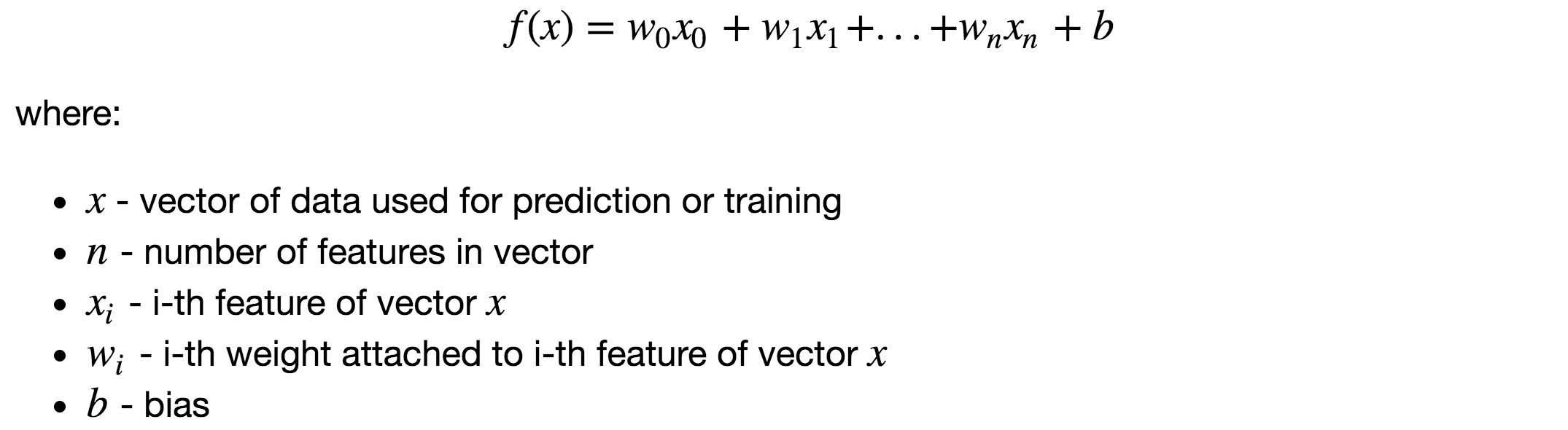


The left side of the equation is a **linear function**. The **manipulation of the weight values will change the angle of the line**.Although, there is still one element missing. Current function is always going through (0,0) point of the coordinate system. To fix that, another trainable parameter is added.



The parameter is named **bias and it gives the formula a freedom to move on the y-axis up and down**.

Every purple parameters belong to the model and is used for predictions for every incoming sample. That’s why finding a solution that works best for all samples is necessary. Formally the formula can be written as:



# **Training and Initialization**

It’s a phase where the **first version of a model is created**. After initialization, the model can be used for predictions but without going through a training process, the results will NOT be good. There are two things to be done:

* **create variables in code** that represents weights and bias parameters,
* **decide on starting values** of model parameters.

The initial values of the model parameters are very crucial for Neural Networks. In the case of Linear Regression **parameter values can be set to zero** at the start.

Function init(n) returns a dictionary containing model parameters. According to the terminology presented in the legend below the mathematical formula, *n is the number of features* used to describe the data sample. It is used by the [zeros](https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.zeros.html) function of NumPy library, to return a vector of ndarray type with n elements and zero value assigned to each. Bias is a scalar set to 0.0 and **it is a good practice to keep the variables as floats rather than integers**. Both weights and bias are accessible under “w” and “b” dictionary keys accordingly.

For the King County dataset there are multiple features describing each sample. Here is the result of calling init(3) :





# **Predictions**

Created model parameters can be used by the model for making a prediction. The formula has already been shown. Now it’s time to turn it into the Python code. First, every feature has to be multiplied by its corresponding weight and summed up. Then bias parameter needs to be added to the product of the previous operation. The outcome is a prediction.

Function predict(x, parameters) takes two arguments:

* vector x of features representing a data sample (e.g. single house),
* Python dictionary parameters which stores parameters of the model along with their current state.

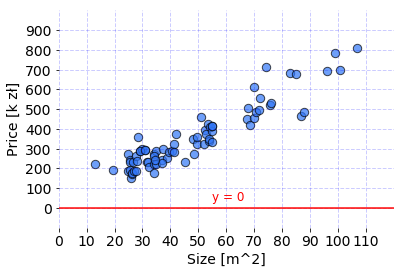
# **Let’s put it all together**

Let’s put together all code parts that were created and display all the results.

**Only one feature was used for prediction** whatreduced formula to form:



This was intentional as **displaying results on the data which has more than 1–2–3 dimensions becomes troublesome**, unless [Dimensionality Reduction](https://en.wikipedia.org/wiki/Dimensionality_reduction) techniques are used (e.g. [PCA](https://en.wikipedia.org/wiki/Principal_component_analysis)). From now on, for learning purposes all code development will be done only on **size** feature. When Linear Regression code is finished, the results with usage of all the features will be presented.



The Line is used to fit the data by Linear Regression model with current parameters. [link](https://gist.github.com/FisherKK/a78a54d4fa9bdd56c9512f24b98df5f9).

The model parameters were initialized with zero values which means that the output of the formula will always be equal to zero. Consequently, the prediction is a Python list of 76 zero values which are predicted prices for each house separately. But that’s ok for now. **Model behavior will improve after training with the Gradient Descent is used and explained.**

Bonus takeouts from the code snippet are:

* Features to be used by model and target value were stored in features and target Python lists. Thanks to that there is no need to modify the whole code if a different set of features should be used.
* It is possible to parse DataFrame object to ndarray by using [as\_matrix](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.as_matrix.html) function.

# **Summary**

In this episode, I have introduced the tools that I am going to use for the rest of Supervised Learning. I have also presented the problem that we are going to solve with the Linear Regression algorithm. At the end, I have shown how to create Linear Regression model and use it for making a prediction.

In the next episode, I will explain how to compare sets of parameters and measure model performance. Finally, I will show you how to update model parameters with Gradient Descent algorithm.