Machine Learning

Mohamad Ghassany

2017-01-09

Contents

Welcome

Welcome to this course. It is only a little introduction to Machine Learning.

The aim of Machine Learning is to build computer systems that can adapt to their environments and learn form experience. Learning techniques and methods from this field are successfully applied to a variety of learning tasks in a broad range of areas, including, for example, spam recognition, text classification, gene discovery, financial forecasting. The course will give an overview of many concepts, techniques, and algorithms in machine learning, beginning with topics such as linear regression and classification and ending up with topics such as kmeans and Expectation Maximization. The course will give the student the basic ideas and intuition behind these methods, as well as a more formal statistical and computational understanding. Students will have an opportunity to experiment with machine learning techniques in R and apply them to a selected problem.

Introduction

What is Machine Learning ?

What is Machine Learning?

Two definitions of Machine Learning are offered. Arthur Samuel described it as: “the field of study that gives computers the ability to learn without being explicitly programmed.” This is an older, informal definition.

Tom Mitchell provides a more modern definition: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Machine Learning is also called Statistical Learning.

Example: playing checkers.

E = the experience of playing many games of checkers

T = the task of playing checkers.

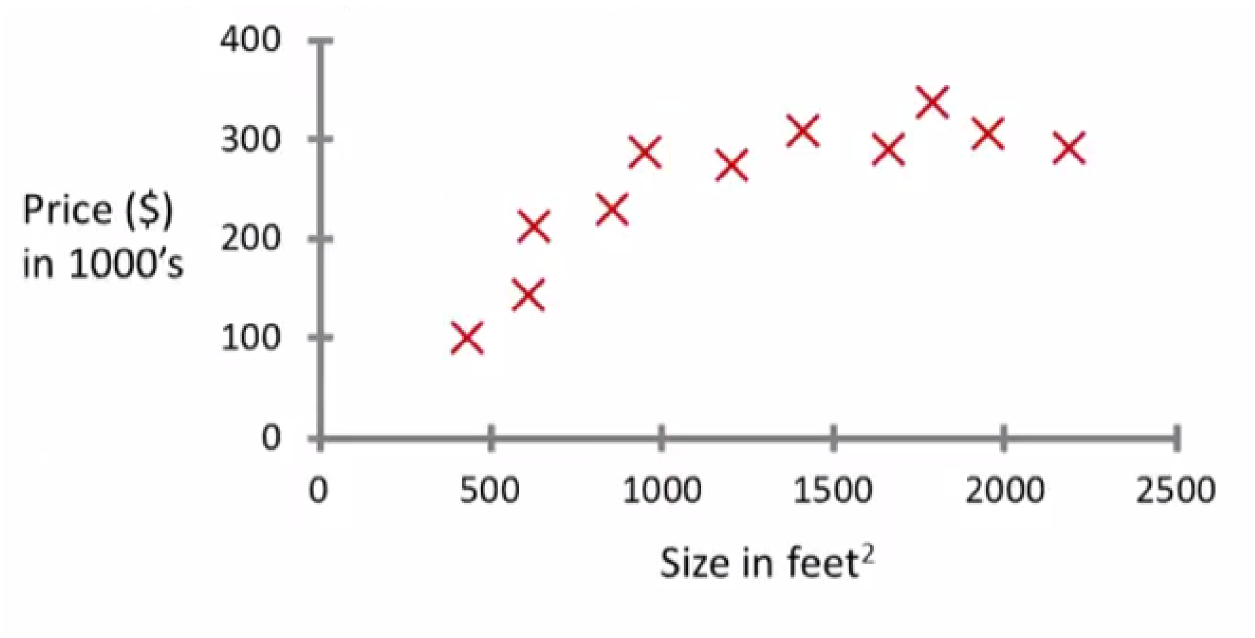
P = the probability that the program will win the next game.

In general, any machine learning problem can be assigned to one of two broad classifications:

Supervised learning and Unsupervised learning.

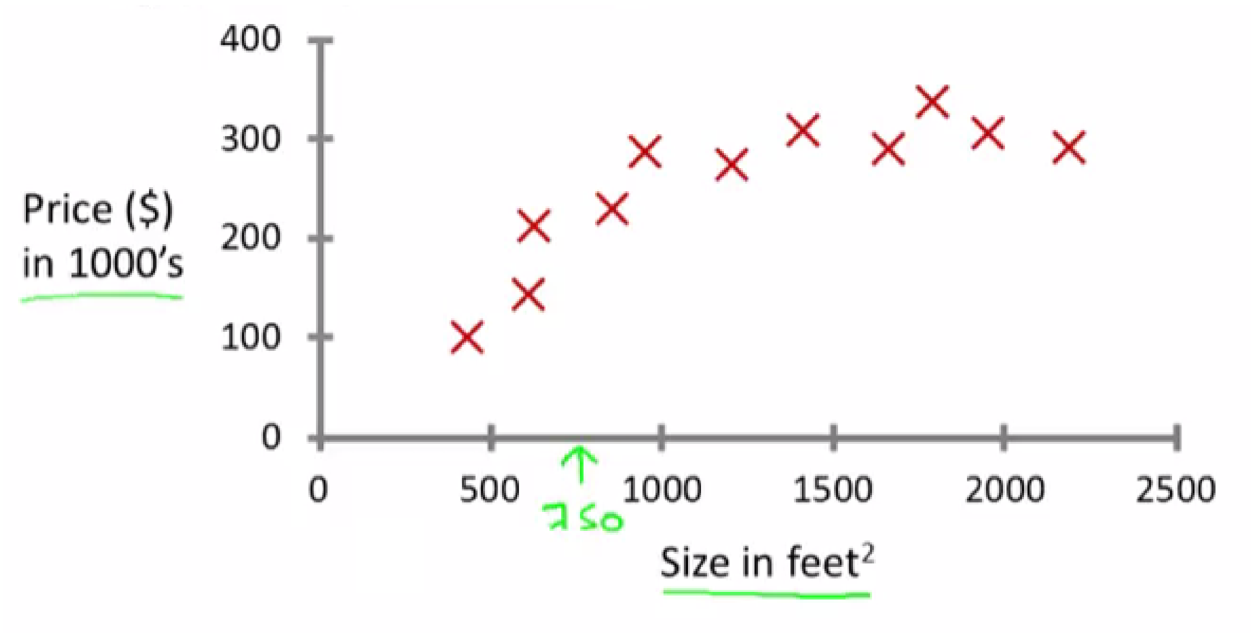
Supervised Learning

Supervised Learning is probably the most common type of machine learning problem. Let’s start with an example of what is it. Let’s say we want to predict housing prices. We plot a data set and it looks like this.



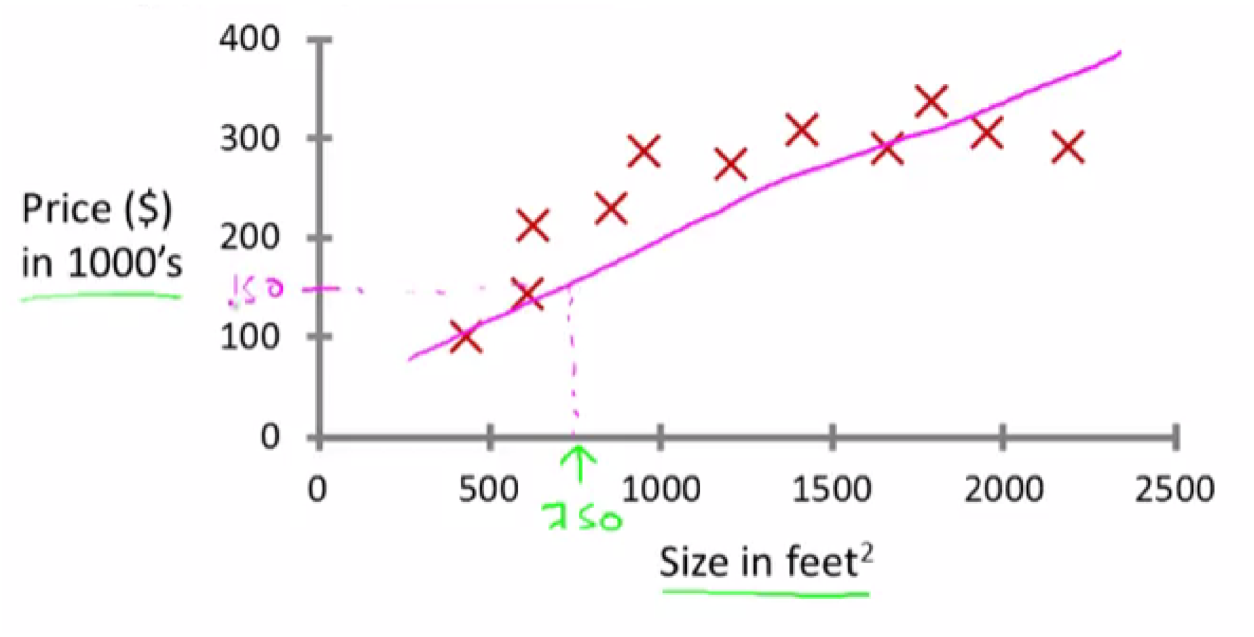
Here on the horizontal axis, the size of different houses in square feet, and on the vertical axis, the price of different houses in thousands of dollars.

So. Given this data, let's say we own a house that is, say 750 square feet and hoping to sell the house and we want to know how much we can get for the house.

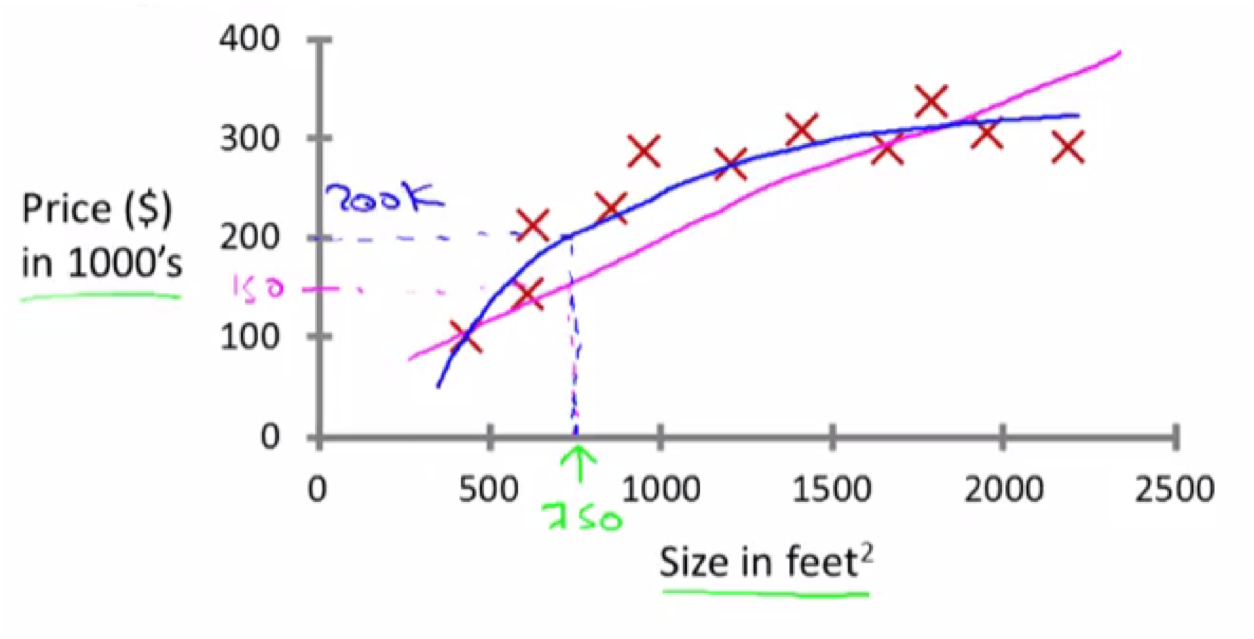


So how can the learning algorithm help?

One thing a learning algorithm might be able to do is put a straight line through the data or to **"fit"** a straight line to the data and, based on that, it looks like maybe the house can be sold for maybe about \$150,000.



But maybe this isn't the only learning algorithm we can use. There might be a better one. For example, instead of sending a straight line to the data, we might decide that it's better to fit a *quadratic function* or a *second-order polynomial* to this data.



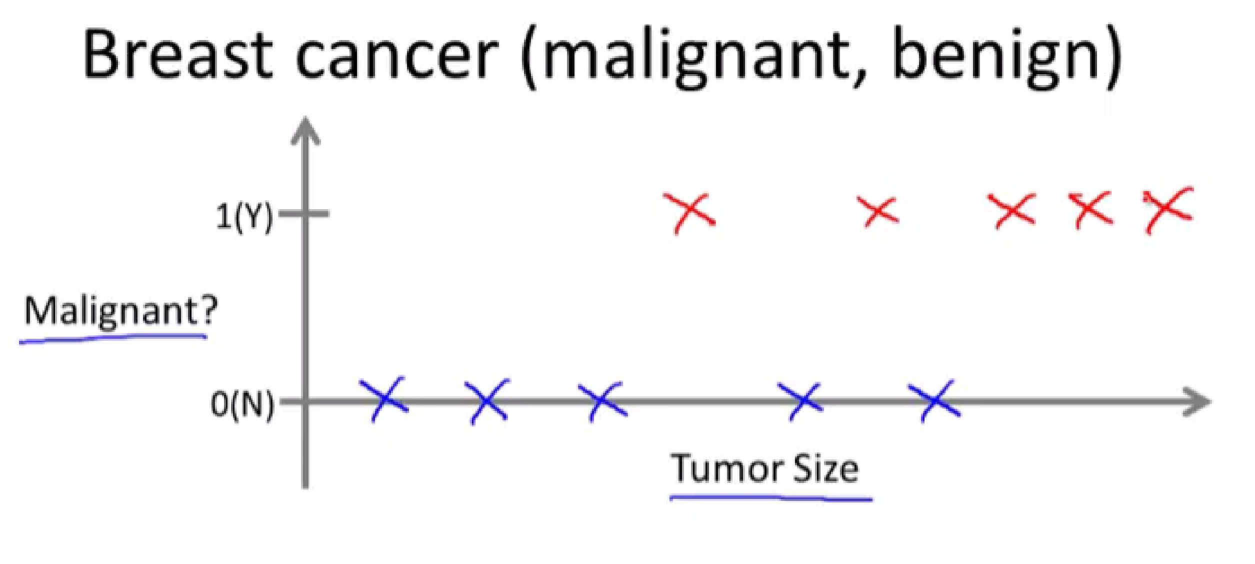
If we do that, and make a prediction here, then it looks like, well, maybe we can sell the house for closer to $200,000.

This is an example of a supervised learning algorithm.

The term supervised learning refers to the fact that we gave the algorithm a data set in which the **"right answers"** were given.

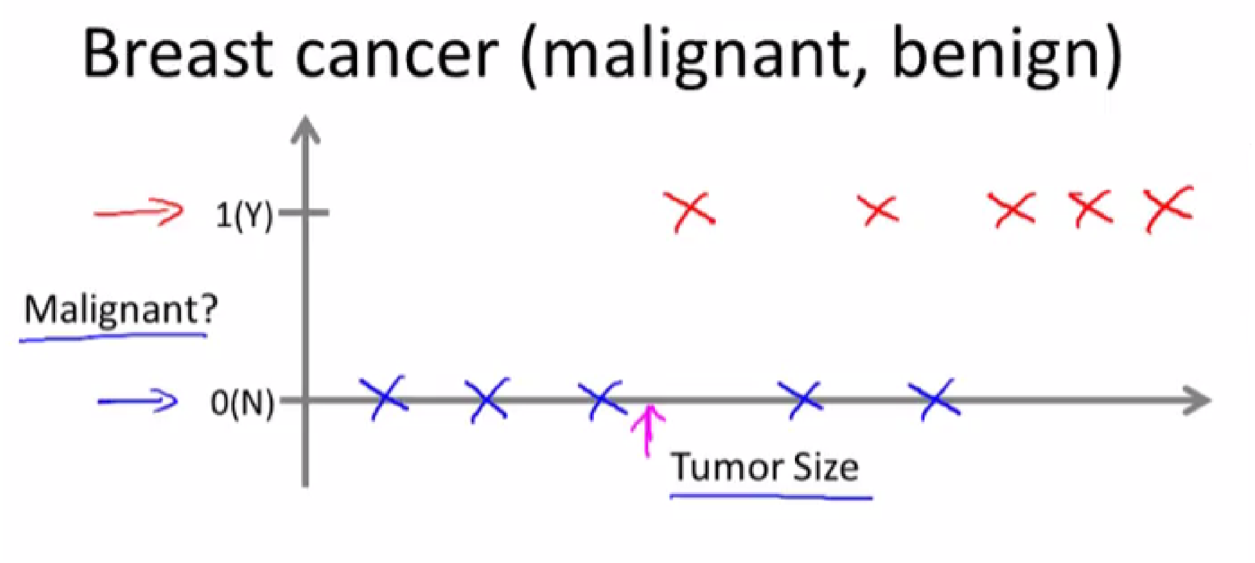
The example above is also called a regression problem. A regression problem is when we try to predict a **continuous** value output. Namely the price in the example.

Here's another supervised learning example. Let's say we want to look at medical records and try to predict of a breast cancer as malignant or benign. If someone discovers a breast tumor, a lump in their breast, a malignant tumor is a tumor that is harmful and dangerous and a benign tumor is a tumor that is harmless. Let's see a collected data set and suppose in the data set we have the size of the tumor on the horizontal axis and on the vertical axis we plot one or zero, yes or no, whether or not these are examples of tumors we've seen before are malignant (which is one) or zero if not malignant or benign.



In this data set we have five examples of benign tumors, and five examples of malignant tumors.

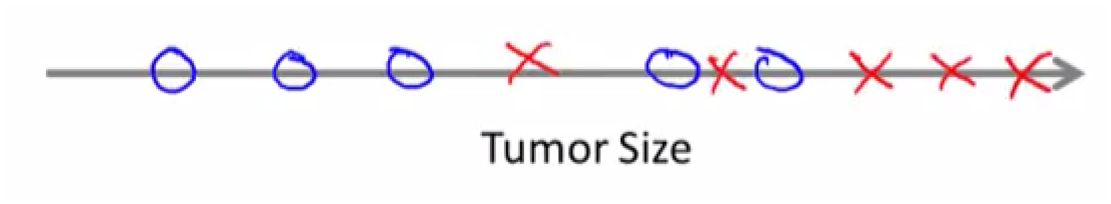
Let's say a person who tragically has a breast tumor, and let's say her breast tumor size is known (rose arrow in the following figure).



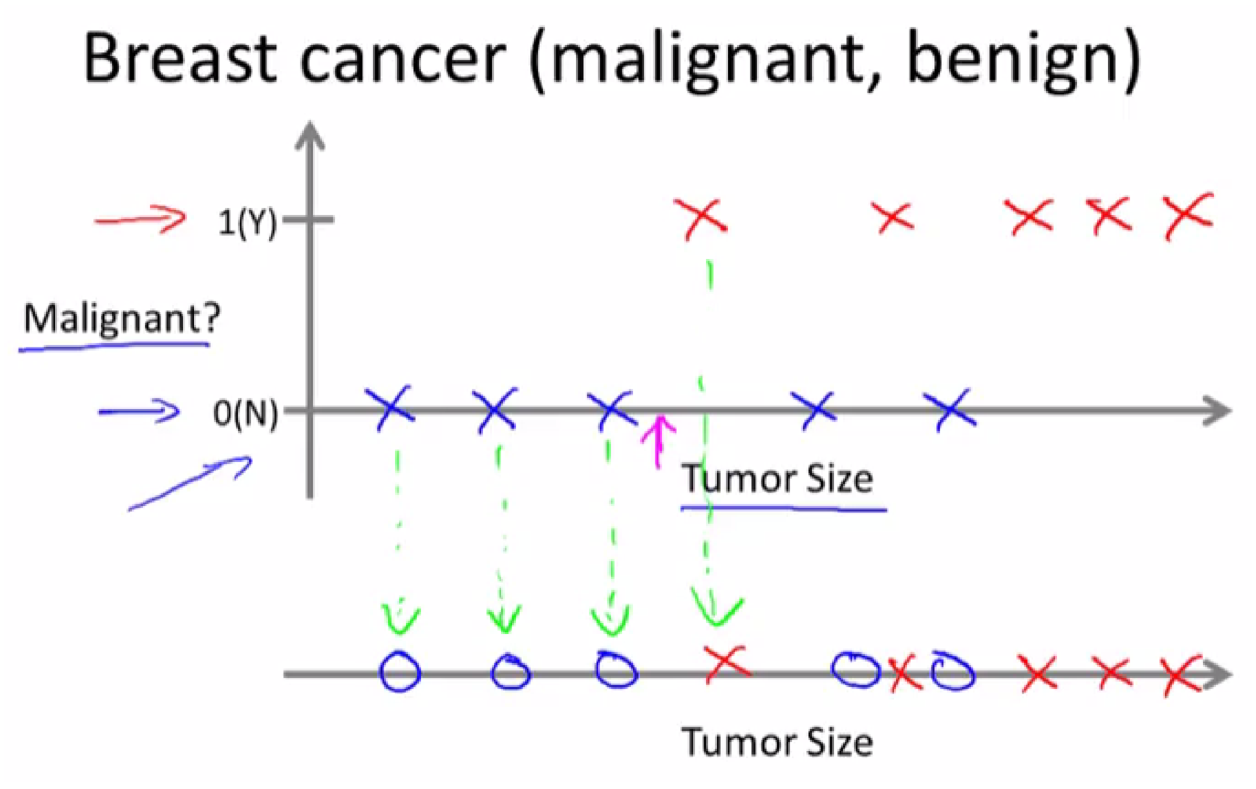
The machine learning question is, can you estimate what is the probability that a tumor is malignant versus benign? To introduce a bit more terminology this is an example of a ***classification*** problem.

The term classification refers to the fact that here we're trying to predict a **discrete** value output: zero or one, malignant or benign. And it turns out that in classification problems sometimes you can have more than two values for the two possible values for the output.

In classification problems there is another way to plot this data. Let's use a slightly different set of symbols to plot this data. So if tumor size is going to be the attribute that we are going to use to predict malignancy or benignness, we can also draw the data like this.



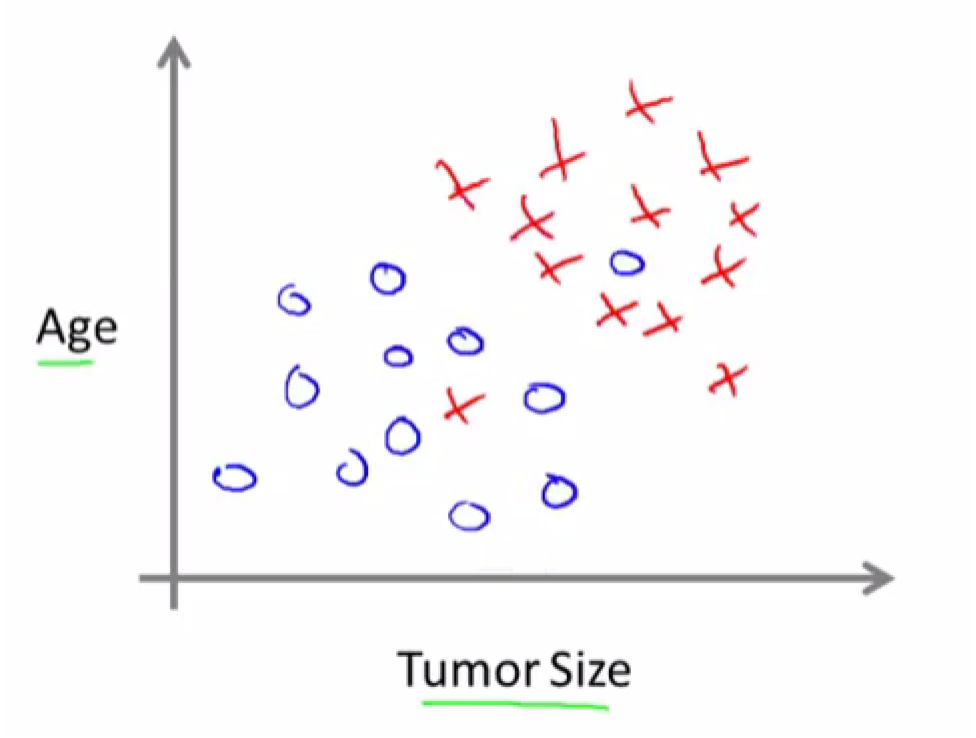
All we did was we took the data set on top and just mapped it down using different symbols. So instead of drawing crosses, we are now going to draw `O`'s for the benign tumors.



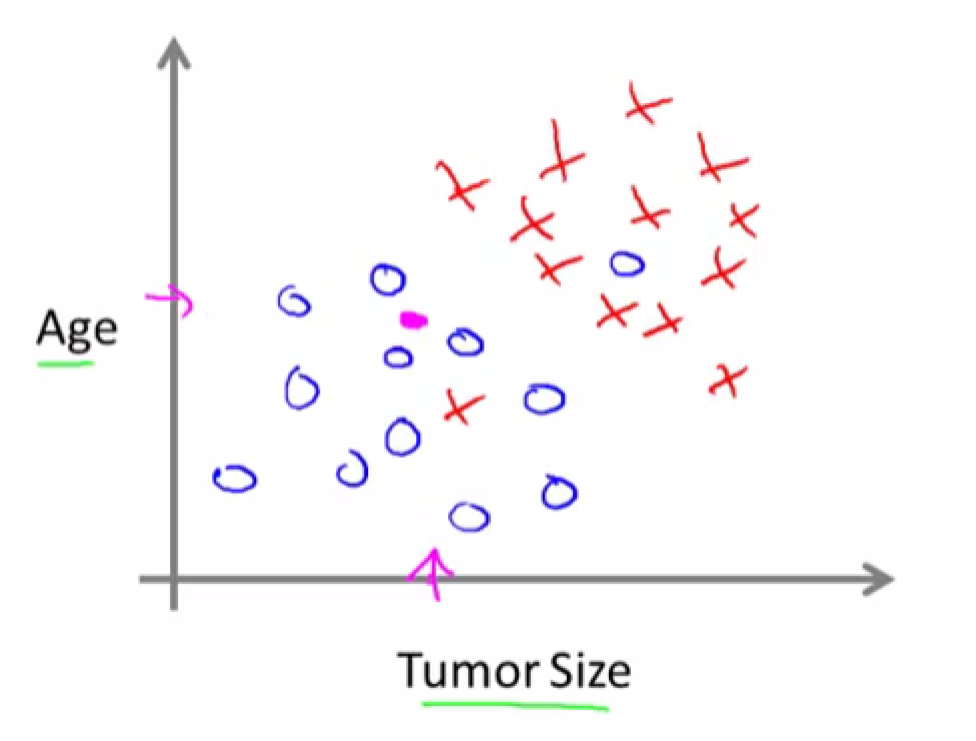
Now, in this example we use only one **feature** or one attribute, mainly, the *tumor size* in order to predict whether the tumor is malignant or benign.

In other machine learning problems we may have more than one feature.

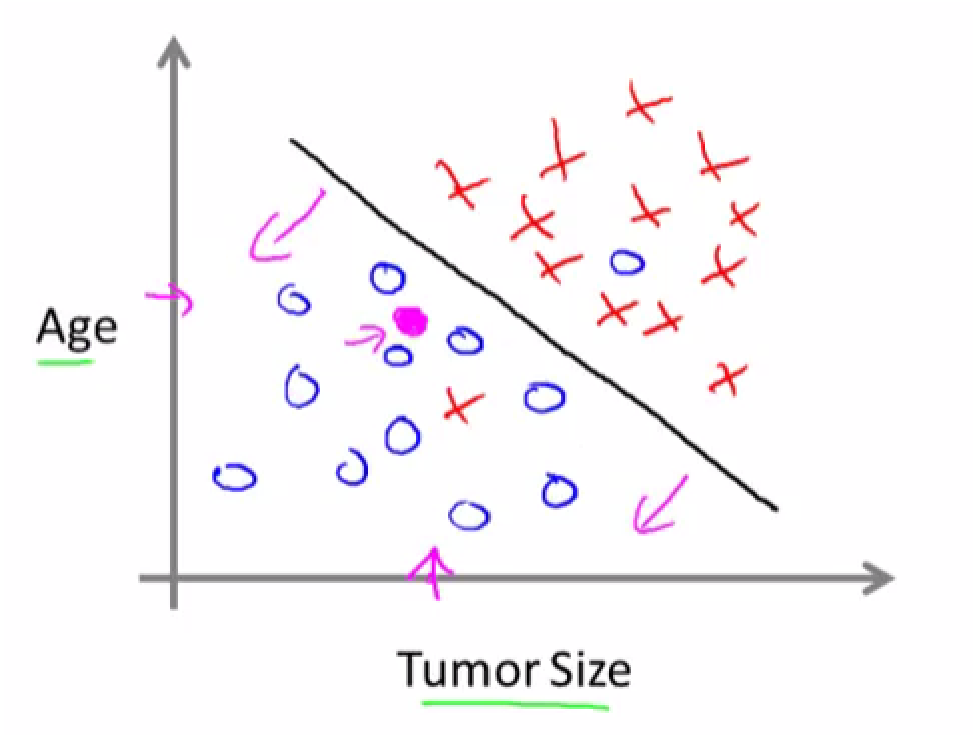
Here's an example. Let's say that instead of just knowing the tumor size, we know both the age of the patients and the tumor size. In that case maybe the data set will look like this.



So, let's say a person who tragically has a tumor. And maybe, their tumor size and age falls around there (rose point):



So given a data set like this, what the learning algorithm might do is throw a straight line through the data to try to separate out the malignant tumors from the benign ones. And with this, hopefully we can decide that the person's tumor falls on this benign side and is therefore more likely to be benign than malignant.



In this example we had **two features**, namely, the age of the patient and the size of the tumor. In other machine learning problems we will often have more features.

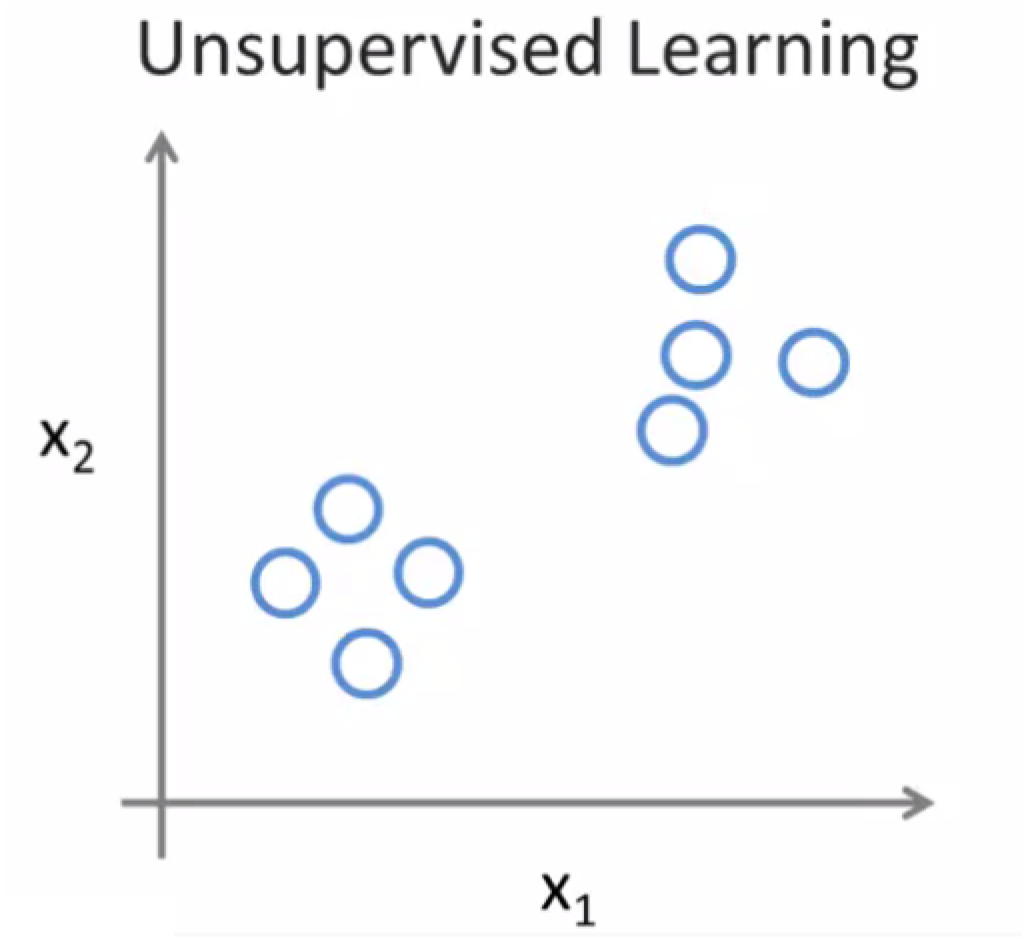
Most interesting learning algorithms is a learning algorithm that can deal with, not just two or three or five features, but an **infinite number of features**. So how do you deal with an infinite number of features. How do you even store an infinite number of things on the computer when your computer is gonna run out of memory.

Unsupervised Learning

The second major type of machine learning problem is called Unsupervised Learning.

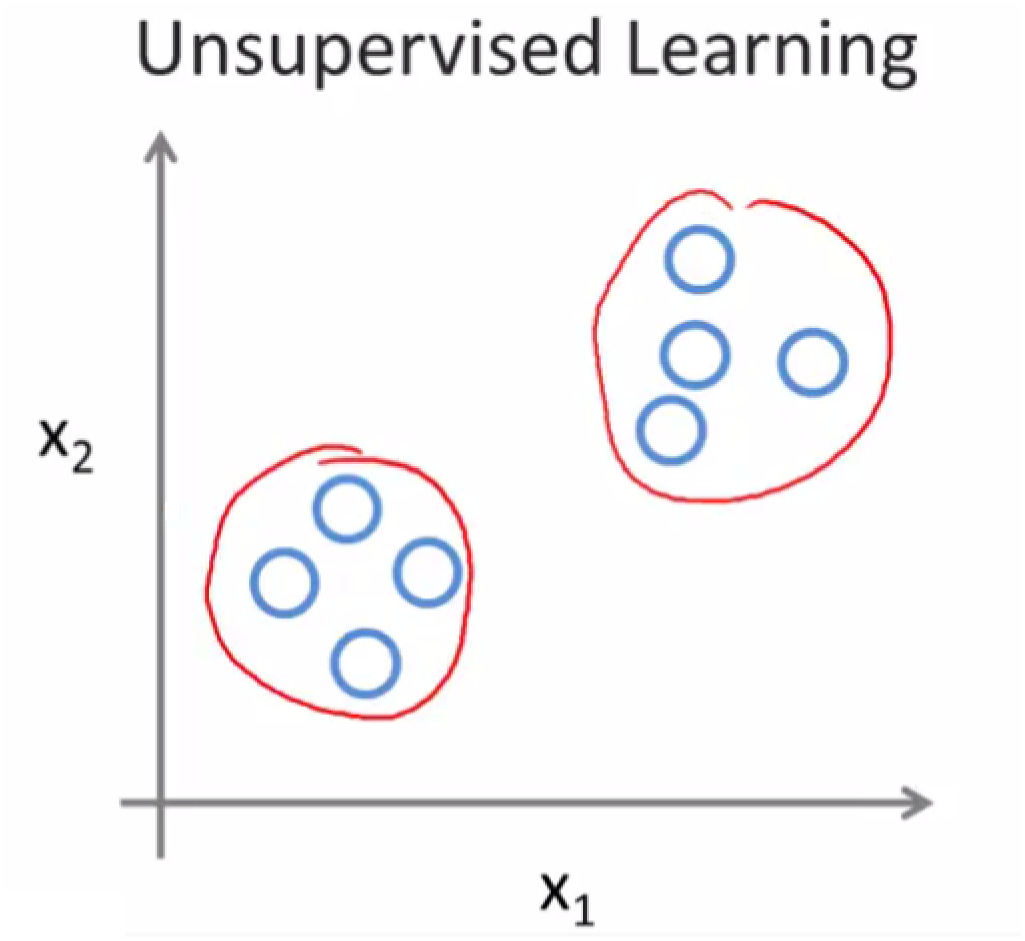
The difference between Unsupervised Learning and Supervised Learning is that in Supervised Learning we are told explicitly what is the so-called right answers (data are labeled).

In Unsupervised Learning, we're given data that doesn't have any labels or that all has the same label or really no labels. Like in this example:



So we're given the data set and we're not told what to do with it and we're not told what each data point is. Instead we're just told, here is a data set. Can you find some structure in the data?

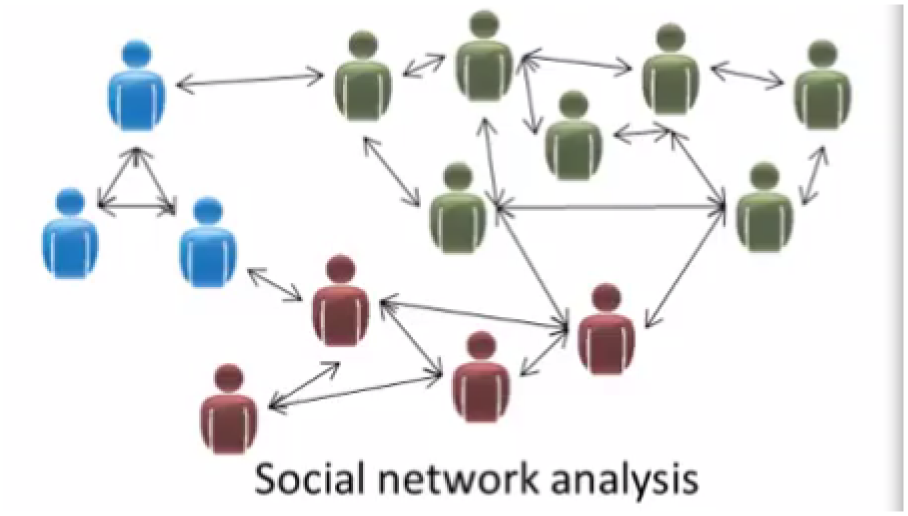
Given this data set, an Unsupervised Learning algorithm might decide that the data lives in two different clusters.



This is called a **clustering** algorithm.

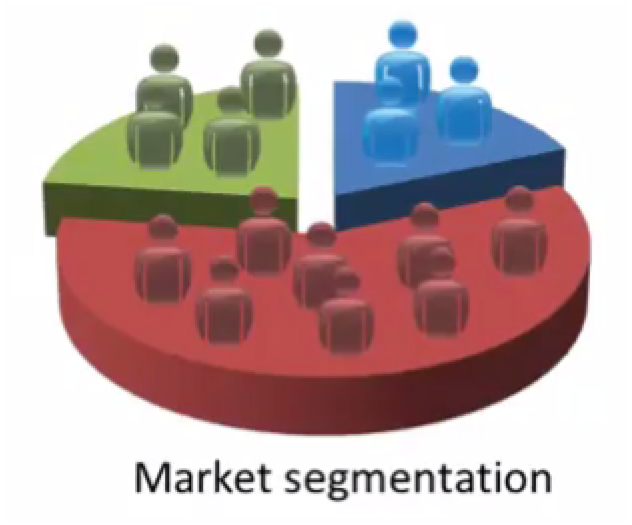
Here are two examples where Unsupervised Learning or clustering is used.

Social network analysis:



So given knowledge about which friends you email the most or given your Facebook friends or your Google+ circles, can we automatically identify which are cohesive groups of friends, also which are groups of people that all know each other?

Market segmentation:



Many companies have huge databases of customer information. So, can you look at this customer data set and automatically discover market segments and automatically group your customers into different market segments so that you can automatically and more efficiently sell or market your different market segments together?

This is Unsupervised Learning because we have all this customer data, but we don't know in advance what are the market segments and for the customers in our data set, we don't know in advance who is in market segment one, who is in market segment two, and so on. But we have to let the algorithm discover all this just from the data.

Part I

Supervised Learning

Part II

Regression

Chapter 1

Linear Regression

1.1 Notation

In general, we will let x_{ij} represent the value of the jth variable for the ith observation, where $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, p$. We will use i to index the samples or observations (from 1 tp n) and j will be used to index the variables (or features) (from 1 to p). We let $\mathbf{X}$ denote a $n \times p$ matrix whose (i, j)th element is x_{ij}. That is,

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1p} \\ x_{21} & x_{22} & x_{23} & \ldots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \ldots & x_{np} \end{pmatrix}$$

Note that it is useful to visualize $\mathbf{X}$ as a spreadsheet of numbers with n rows and p columns. We will write the rows of $\mathbf{X}$ as $x_1, x_2, \ldots, x_n$. Here x_i is a vector of length p, containing the p variable measurements for the ith observation. That is,

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{ip} \end{pmatrix}$$

(Vectors are by default represented as columns.)

We will write the columns of $\mathbf{X}$ as $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_p$. Each is a vector of length n. That is,

$$\mathbf{x}_j = \begin{pmatrix} \mathbf{x}_{1j} \\ \mathbf{x}_{2j} \\ \vdots \\ \mathbf{x}_{nj} \end{pmatrix}$$

Using this notation, the matrix $\mathbf{X}$ can be written as

$$\mathbf{X} = (\mathbf{x}_1 \mathbf{x}_2 \ldots \mathbf{x}_p)$$

or

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ x_2^T \\ \vdots \\ x_n^T \end{pmatrix}$$

The ^{T} notation denotes the transpose of a matrix or vector.

We use y_i to denote the ith observation of the variable on which we wish to make predictions. We write the set of all n observations in vector form as

$$\mathbf{y} = \begin{pmatrix} y_1^T \\ y_2^T \\ \vdots \\ y_n^T \end{pmatrix}$$

Then the observed data consists of $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where each x_i is a vector of length p. (If $p = 1$, then x_i is simply a scalar).

1.2 Model Representation

Let's consider the example about predicting housing prices. We're going to use this data set as an example,

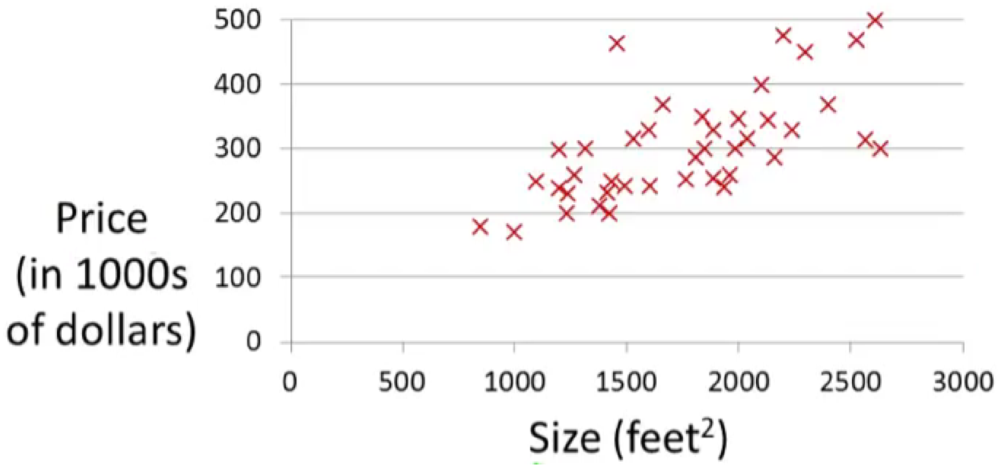


Figure 1.1:

Suppose that there is a person trying to sell a house of size 1250 square feet and he wants to know how much he might be able to sell the house for. One thing we could do is fit a model. Maybe fit a straight line to this data. Looks something like this,

and based on that, maybe he can sell the house for around $220,000. Recall that this is an example of a supervised learning algorithm. And it's supervised learning because we're given the "right answer" for each of our examples. More precisely, this is an example of a regression problem where the term regression refers to the fact that we are predicting a real-valued output namely the price.

More formally, in supervised learning, we have a data set and this data set is called a **training set**. So for housing prices example, we have a training set of different housing prices and our job is to learn from this data how to predict prices of the houses.

Let's define some notation from this data set:

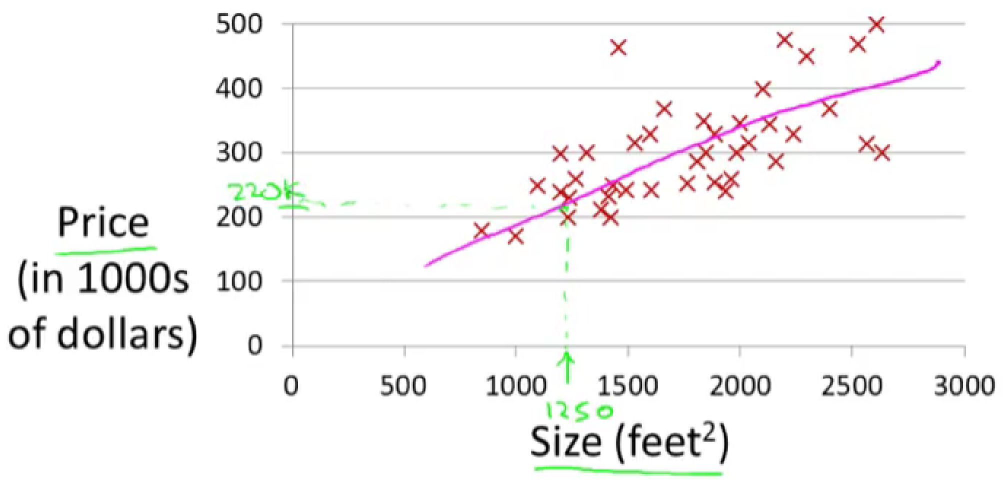
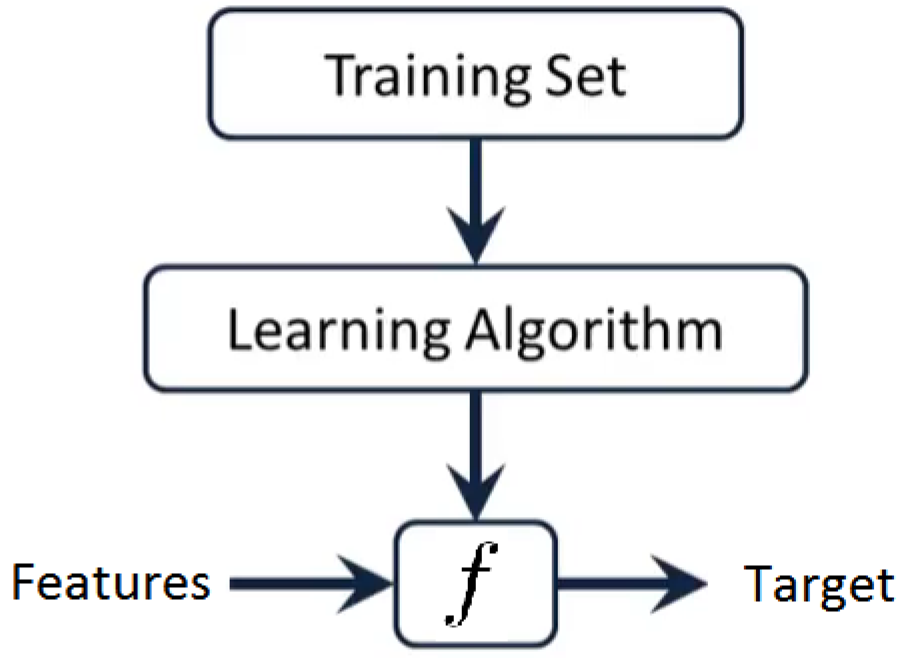


Figure 1.2:

- The size of the house is the input variable.
- The house price is the output variable.
- The input variables are typically denoted using the variable symbol X,
- The inputs go by different names, such as *predictors*, *independent variables*, *features*, *predictor* or sometimes just *variables*.
- The output variable is often called the *response*, *dependent variable* or *target*, and is typically denoted using the symbol Y.
- (x_i, y_i) is the ith training example.
- The set of $\{(x_i, y_i)\}$ is the training set.
- n is the number of training examples.

So here's how this supervised learning algorithm works. Suppose that we observe a quantitative response Y and p different predictors, $X_1, X_2, \ldots, X_p$. We assume that there is some relationship between Y and $X = (X_1, X_2, \ldots, X_p)$, which can be written in the very general form

$$Y = f(X) + \epsilon$$



Here f is some fixed but unknown function of $X_1, X_2, \ldots, X_p$, and ϵ is a random error term, which is independent of X and has mean zero. The f function is also called *hypothesis* in Machine Learning. In general, the function f may involve more than one input variable. In essence, Supervised Learning refers to a set of approaches for estimating f.

1.3 Why Estimate f ?

There are two main reasons that we may wish to estimate f: *prediction* and *inference.*

Prediction

In many situations, a set of inputs X are readily available, but the output Y cannot be easily obtained. In this setting, since the error term averages to zero, we can predict Y using

$$\hat{Y} = \hat{f}(X)$$

where $\hat{f}$ represents our estimate for f, and $\hat{Y}$ represents the resulting prediction for Y. Like in the example above about predicting housing prices.

We can measure the accuracy of $\hat{Y}$ by using a **cost function**. In the regression models, the most commonly-used measure is the *mean squared error* (MSE), given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2$$

Inference

We are often interested in understanding the way that Y is affected as $X_1, X_2, \ldots, X_p$ change. In this situation we wish to estimate f , but our goal is not necessarily to make predictions for Y. We instead want to understand the relationship between X and Y, or more specifically, to understand how Y changes as a function of $X_1, X_2, \ldots, X_p$. In this case, one may be interested in answering the following questions:

- Which predictors are associated with the response?
- What is the relationship between the response and each predictor?
- Can the relationship between Y and each predictor be adequately summarized using a linear equation, or is the relationship more complicated?

1.4 Simple Linear Regression Model

Simple linear regression is a very straightforward approach for predicting a quantitative response Y on the basis of a single predictor variable X. It assumes that there is approximately a linear relationship between X and Y. Mathematically, we can write this linear relationship as

$$\begin{aligned} Y &= \beta_0 + \beta_1 X + \epsilon \\ Y &\approx \beta_0 + \beta_1 X \end{aligned}$$

where β_0 and β_1 are two unknown constants that represent the *intercept* and *slope*, also known as ***coefficients*** or *parameters*, and ϵ is the error term.

Given some estimates $\hat{\beta_0}$ and $\hat{\beta_1}$ for the model coefficients, we predict future inputs x using

$$\hat{y} = \hat{\beta_0} + \hat{\beta_1} x$$

where $\hat{y}$ indicates a prediction of Y on the basis of $X = x$. The *hat* symbol, $\hat{}$, denotes an estimated value.

1.5 Estimating the Coefficients

Let $\hat{y_i} = \hat{\beta_0} + \hat{\beta_1} x_i$ be the prediction for Y based on the ith value of X. Then $e_i = y_i - \hat{y_i}$ represents the ith ***residual***.

We define the ***residual sum of squares*** (**RSS**) as

$$\begin{aligned} RSS &= e_1^2 + e_2^2 + \ldots + e_n^2 \\ &= \sum_{i=1}^{n} e_i^2 \end{aligned}$$

or equivantly as

$$\begin{aligned} RSS &= (y_1 - \hat{\beta_0} - \hat{\beta_1} x_1)^2 + (y_2 - \hat{\beta_0} - \hat{\beta_1} x_2)^2 + \ldots + (y_n - \hat{\beta_0} - \hat{\beta_1} x_n)^2 \\ &= \sum_{i=1}^{n} (y_i - \hat{\beta_0} - \hat{\beta_1} x_i)^2 \end{aligned}$$

The *least squares* approach chooses $\hat{\beta_0}$ and $\hat{\beta_1}$ to minimize the RSS. The minimizing values can be show to be

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{s_{xy}}{s_x^2}$$

and

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

where:

- $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ is the *sample mean.*
- $s_x^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2$ is the *sample variance*. The sample standard deviation is $s_x = \sqrt{s_x^2}$.
- $s_{xy} = \frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$ is the *sample covariance*. It measures the degree of linear association between $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$. Once scaled by $s_x s_y$, it gives the *sample correlation coefficient*, $r_{xy} = \frac{s_{xy}}{s_x s_y}$.

Click here to see the influence of the distance employed in the sum of squares. Try to minimize the sum of squares for the different datasets. The choices of intercept and slope that minimize the sum of squared distances for a kind of distance are not the optimal for a different kind of distance.

1.6 Assessing the Accuracy of the Coefficient Estimates

The standard error of an estimator reflects how it varies under repeated sampling. We have

$$\mathrm{SE}(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\mathrm{SE}(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2}\right]$$

where $\sigma^2 = Var(\epsilon)$

In general, σ^2 is know known, but can be estimated from the data. The estimate of σ is known as the *residual standard error*, and is given by

$$\mathrm{RSE} = \sqrt{\frac{\mathrm{RSS}}{(n-2)}}$$

These standard errors can be used to compute *confidence intervals*. A 95% confidence interval is defined as a range of values such that with 95% probability, the range will contain the true unknown value of the parameter. It has the form

$$\hat{\beta}_1 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_1)$$

That is, there is approximately a 95% chance that the interval

$$\left[\hat{\beta}_1 - 2 \cdot \mathrm{SE}(\hat{\beta}_1), \hat{\beta}_1 + 2 \cdot \mathrm{SE}(\hat{\beta}_1)\right]$$

will contain the true value of β_1. Similarly, a confidence interval for β_0 approximately takes the form

$$\hat{\beta}_0 \pm 2 \cdot \mathrm{SE}(\hat{\beta}_0)$$

Hypothesis testing

Standard errors can also be used to perform *hypothesis tests* on the coefficients. The most common hypothesis test involves testing the *null hypothesis* of

$$H_0 : \text{There is no relationship between } X \text{ and } Y$$

versus the *alternative hypothesis*

$$H_1 : \text{There is some relationship between } X \text{ and } Y$$

Mathematically, this corresponds to testing

$$H_0 : \beta_1 = 0$$

versus

$$H_1 : \beta_1 \neq 0$$

since if $\beta_1 = 0$ then the simple linear regression model reduces to $Y = \beta_0 + \epsilon$, and X is not associated with Y.

To test the null hypothesis H_0, we compute a ***t-statistic***, given by

$$t = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}(\hat{\beta}_1)}$$

This will have a t-distribution (*Student*) with $n - 2$ degrees of freedom, assuming $\beta_1 = 0$.

Using statistical software, it is easy to compute the probability of observing any value equal to $|t|$ or larger. We call this probability the ***p-value***.

If p-value is small enough (typically under 0.01 (1% error) or 0.05 (5% error)) we reject the null hypothesis, that is we declare a relationship to exist between X and Y.

1.7 ANOVA and model fit

1.7.1 ANOVA

In this section we will see how the variance of Y is decomposed into two parts, each one corresponding to the regression and to the error, respectively. This decomposition is called the *ANalysis Of VAriance* (ANOVA).

Before explaining ANOVA, it is important to recall an interesting result: *the mean of the fitted values* $\hat{Y}_1, \ldots, \hat{Y}_n$ *is the mean of* $Y_1, \ldots, Y_n$. This is easily seen if we plug-in the expression of $\hat{\beta}_0$:

$$\frac{1}{n}\sum_{i=1}^n \hat{Y}_i = \frac{1}{n}\sum_{i=1}^n \left(\hat\beta_0 + \hat\beta_1 X_i\right) = \hat\beta_0 + \hat\beta_1 \bar{X} = \left(\bar{Y} - \hat\beta_1 \bar{X}\right) + \hat\beta_1 \bar{X} = \bar{Y}.$$

The ANOVA decomposition considers the following measures of variation related with the response:

- $\text{SST} = \sum_{i=1}^n \left(Y_i - \bar{Y}\right)^2$, the **total sum of squares**. This is the *total variation* of $Y_1, \ldots, Y_n$, since $\text{SST} = ns_y^2$, where s_y^2 is the sample variance of $Y_1, \ldots, Y_n$.
- $\text{SSR} = \sum_{i=1}^n \left(\hat{Y}_i - \bar{Y}\right)^2$, the **regression sum of squares**[1]. This is the variation explained by the regression line, that is, *the variation from $\bar{Y}$ that is explained by the estimated conditional mean* $\hat{Y}_i = \hat\beta_0 + \hat\beta_1 X_i$. $\text{SSR} = ns_{\hat{y}}^2$, where $s_{\hat{y}}^2$ is the sample variance of $\hat{Y}_1, \ldots, \hat{Y}_n$.
- $\text{SSE} = \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2$, the **sum of squared errors**[2]. Is the variation around the conditional mean. Recall that $\text{SSE} = \sum_{i=1}^n \hat\varepsilon_i^2 = (n-2)\hat\sigma^2$, where $\hat\sigma^2$ is the sample variance of $\hat\varepsilon_1, \ldots, \hat\varepsilon_n$.

The ANOVA decomposition is

$$\underbrace{\text{SST}}_{\text{Variation of } Y_i' s} = \underbrace{\text{SSR}}_{\text{Variation of } \hat{Y}_i' s} + \underbrace{\text{SSE}}_{\text{Variation of } \hat\varepsilon_i' s}$$

The graphical interpretation of this equation is shown in the following figures.

Click here to see the ANOVA decomposition and its dependence on σ^2 and $\hat\sigma^2$.

The ANOVA table summarizes the decomposition of the variance. Here is given in the layout employed by `R`.

	Degrees of freedom	Sum Squares	Mean Squares	F-value	p-value
Predictor	1	SSR	$\frac{\text{SSR}}{1}$	$\frac{\text{SSR}/1}{\text{SSE}/(n-2)}$	p
Residuals	$n-2$	SSE	$\frac{\text{SSE}}{n-2}$		

The `anova` function in `R` takes a model as an input and returns the ANOVA table.

The "F-value" of the ANOVA table represents the value of the F-statistic $\frac{\text{SSR}/1}{\text{SSE}/(n-2)}$. This statistic is employed to test

$$H_0 : \beta_1 = 0 \quad \text{vs.} \quad H_1 : \beta_1 \neq 0,$$

that is, the hypothesis of no linear dependence of Y on X. The result of this test is completely equivalent

[1] Recall that SSR is different from RSS (Residual Sum of Squares

[2] Recall that SSE and RSS (for $(\hat\beta_0, \hat\beta_1)$) are just different names for referring to the same quantity: $\text{SSE} = \sum_{i=1}^n \left(Y_i - \hat{Y}_i\right)^2 = \sum_{i=1}^n \left(Y_i - \hat\beta_0 - \hat\beta_1 X_i\right)^2 = \text{RSS}\left(\hat\beta_0, \hat\beta_1\right)$.

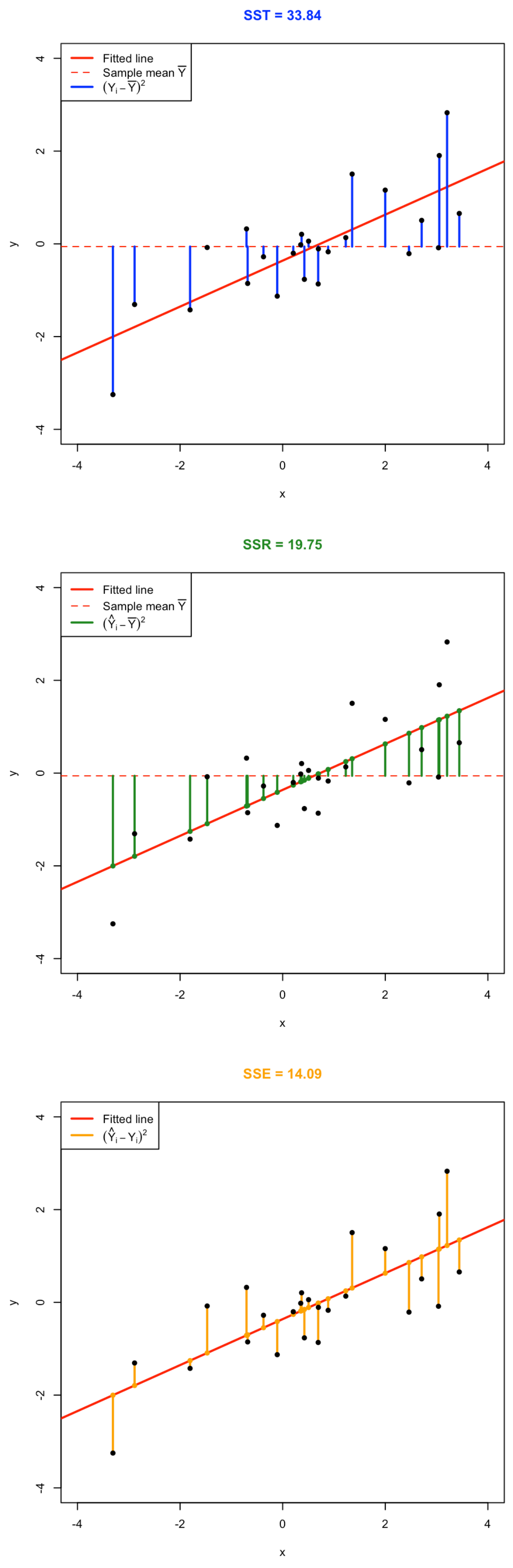


Figure 1.3: Visualization of the ANOVA decomposition. SST measures the variation of $Y_1, \ldots, Y_n$ with respect to $\bar{Y}$. SST measures the variation with respect to the conditional means, $\hat{\beta}_0 + \hat{\beta}_1 X_i$. SSE collects the variation of the residuals.

to the t-test for β_1 that we saw previously in the Hypothesis testing (this is something *specific for simple linear regression* – the F-test will not be equivalent to the t-test for β_1 in the Mulitple Linear Regression).

It happens that

$$F = \frac{\text{SSR}/1}{\text{SSE}/(n-2)} \overset{H_0}{\sim} F_{1,n-2},$$

where $F_{1,n-2}$ is the *Snedecor's F distribution*[3] with 1 and $n-2$ degrees of freedom.

If H_0 is true, then F is expected to be *small* since SSR will be close to zero. The p-value of this test is the same as the p-value of the t-test for $H_0 : \beta_1 = 0$.

1.7.2 The R^2 Statistic

To calculate R^2, we use the formula

$$R^2 = \frac{\text{TSS} - \text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

where $\text{TSS} = \sum(y_i - \bar{y})^2$ is the *total sum of squared.*

R^2 measures the *proportion of variability in Y that can be explained using X*. An R^2 statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression. A number near 0 indicates that the regression did not explain much of the variability in the response; this might occur because the linear model is wrong, or the inherent error σ^2 is high, or both.

It can be shown that in this simple linear linear regression setting that $R^2 = r^2$, where r is the correlation between X and Y:

$$r = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$$

⚠ R^2 does not measure the correctness of a linear model but its **usefulness** (for prediction, for *explaining the variance* of Y), assuming the model is correct.

Trusting blindly the R^2 can lead to catastrophic conclusions, since the model may not be correct.

So remember:

💡 A large R^2 means *nothing* if the **assumptions of the model do not hold**. R^2 is the proportion of variance of Y explained by X, but, of course, *only when the linear model is correct.*

[3]The $F_{n,m}$ distribution arises as the quotient of two independent random variables χ^2_n and χ^2_m, $\frac{\chi^2_n/n}{\chi^2_m/m}$.

PW 1

1.8 Some R basics

1.8.1 Basic Commands

R uses functions to perform operations. To run a function called `funcname`,we type `funcname(input1, input2)` , where the inputs (or arguments) `input1` and `input2` tell R how to run the function. A function can have any number of inputs. For example, to create a vector of numbers, we use the function `c()` (for *concatenate*).

```
x <- c(1,3,2,5)
x
#ans> [1] 1 3 2 5
```

Note that the `>` is not part of the command; rather, it is printed by R to indicate that it is ready for another command to be entered. We can also save things using `=` rather than `<-`. Note that the answer in the code above is followed by `#ans>` while in the R console it is not.

```
x = c(1,6,2)
x
#ans> [1] 1 6 2
y = c(1,4,3)
length(x)
#ans> [1] 3
length(y)
#ans> [1] 3
x+y
#ans> [1]  2 10  5
```

Hitting the *up arrow* multiple times will display the previous commands, which can then be edited. This is useful since one often wishes to repeat a similar command.

The `ls()` function allows us to look at a list of all of the objects, such `ls()` as data and functions, that we have saved so far. The `rm()` function can be used to delete any object that we don't want.

```
ls()
#ans> [1] "x" "y"
rm(x)
ls()
#ans> [1] "y"
```

1.8.2 Vectors

```r
# A handy way of creating sequences is the operator :
# Sequence from 1 to 5
1:5
#ans> [1] 1 2 3 4 5

# Storing some vectors
vec <- c(-4.12, 0, 1.1, 1, 3, 4)
vec
#ans> [1] -4.12  0.00  1.10  1.00  3.00  4.00

# Entry-wise operations
vec + 1
#ans> [1] -3.12  1.00  2.10  2.00  4.00  5.00
vec^2
#ans> [1] 16.97  0.00  1.21  1.00  9.00 16.00

# If you want to access a position of a vector, use [position]
vec[6]
#ans> [1] 4

# You also can change elements
vec[2] <- -1
vec
#ans> [1] -4.12 -1.00  1.10  1.00  3.00  4.00

# If you want to access all the elements except a position, use [-position]
vec[-2]
#ans> [1] -4.12  1.10  1.00  3.00  4.00

# Also with vectors as indexes
vec[1:2]
#ans> [1] -4.12 -1.00

# And also
vec[-c(1, 2)]
#ans> [1] 1.1 1.0 3.0 4.0
```

Do the following:

- Create the vector $x = (1, 7, 3, 4)$.
- Create the vector $y = (100, 99, 98, ..., 2, 1)$.
- Compute $x_2 + y_4$ and $\cos(x_3) + \sin(x_2)e^{-y_2}$. (Answers: `104`, `-0.9899925`)
- Set $x_3 = 0$ and $y_2 = -1$. Recompute the previous expressions. (Answers: `97`, `2.785875`)
- Index y by $x + 1$ and store it as `z`. What is the output? (Answer: `z` is `c(-1, 100, 97, 96)`)

1.8.3 Matrices, data frames and lists

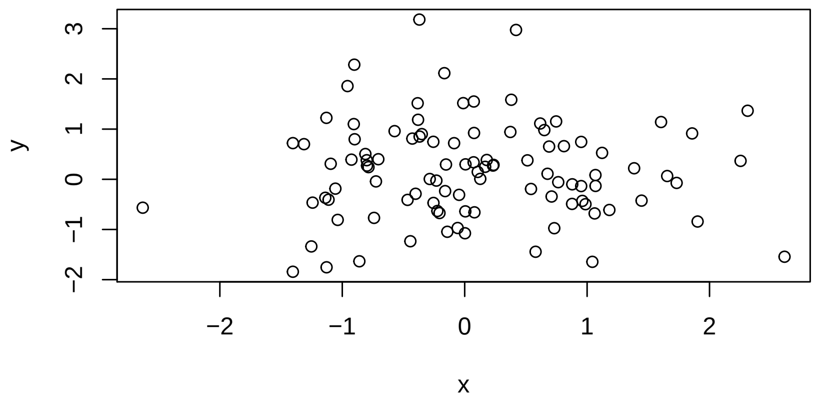
```
# A matrix is an array of vectors
A <- matrix(1:4, nrow = 2, ncol = 2)
A
#ans>      [,1] [,2]
#ans> [1,]    1    3
#ans> [2,]    2    4

# Another matrix
B <- matrix(1:4, nrow = 2, ncol = 2, byrow = TRUE)
B
#ans>      [,1] [,2]
#ans> [1,]    1    2
#ans> [2,]    3    4

# Binding by rows or columns
rbind(1:3, 4:6)
#ans>      [,1] [,2] [,3]
#ans> [1,]    1    2    3
#ans> [2,]    4    5    6
cbind(1:3, 4:6)
#ans>      [,1] [,2]
#ans> [1,]    1    4
#ans> [2,]    2    5
#ans> [3,]    3    6

# Entry-wise operations
A + 1
#ans>      [,1] [,2]
#ans> [1,]    2    4
#ans> [2,]    3    5
A * B
#ans>      [,1] [,2]
#ans> [1,]    1    6
#ans> [2,]    6   16

# Accessing elements
A[2, 1] # Element (2, 1)
#ans> [1] 2
A[1, ] # First row
#ans> [1] 1 3
A[, 2] # First column
#ans> [1] 3 4

# A data frame is a matrix with column names
# Useful when you have multiple variables
myDf <- data.frame(var1 = 1:2, var2 = 3:4)
myDf
#ans>   var1 var2
#ans> 1    1    3
#ans> 2    2    4
```

```
# You can change names
names(myDf) <- c("newname1", "newname2")
myDf
#ans>   newname1 newname2
#ans> 1        1        3
#ans> 2        2        4

# The nice thing is that you can access variables by its name with the $ operator
myDf$newname1
#ans> [1] 1 2

# And create new variables also (it has to be of the same
# length as the rest of variables)
myDf$myNewVariable <- c(0, 1)
myDf
#ans>   newname1 newname2 myNewVariable
#ans> 1        1        3             0
#ans> 2        2        4             1

# A list is a collection of arbitrary variables
myList <- list(vec = vec, A = A, myDf = myDf)

# Access elements by names
myList$vec
#ans> [1] -4.12 -1.00  1.10  1.00  3.00  4.00
myList$A
#ans>      [,1] [,2]
#ans> [1,]    1    3
#ans> [2,]    2    4
myList$myDf
#ans>   newname1 newname2 myNewVariable
#ans> 1        1        3             0
#ans> 2        2        4             1

# Reveal the structure of an object
str(myList)
#ans> List of 3
#ans>  $ vec : num [1:6] -4.12 -1 1.1 1 3 4
#ans>  $ A   : int [1:2, 1:2] 1 2 3 4
#ans>  $ myDf:'data.frame': 2 obs. of  3 variables:
#ans>   ..$ newname1     : int [1:2] 1 2
#ans>   ..$ newname2     : int [1:2] 3 4
#ans>   ..$ myNewVariable: num [1:2] 0 1
str(myDf)
#ans> 'data.frame': 2 obs. of  3 variables:
#ans>  $ newname1     : int  1 2
#ans>  $ newname2     : int  3 4
#ans>  $ myNewVariable: num  0 1

# A less lengthy output
names(myList)
#ans> [1] "vec"  "A"    "myDf"
```

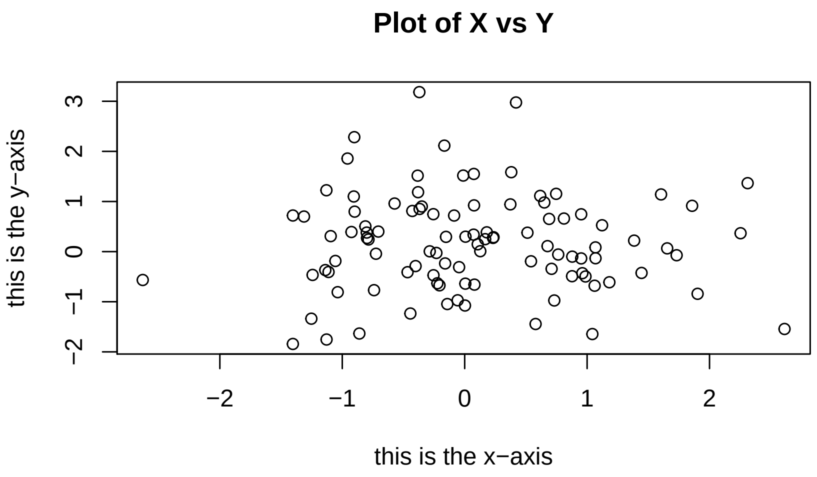
1.8.4 Graphics

The `plot()` function is the primary way to plot data in `R` . For instance, `plot(x,y)` produces a scatterplot of the numbers in `x` versus the numbers in `y`. There are many additional options that can be passed in to the `plot()` function. For example, passing in the argument `xlab` will result in a label on the `x-axis`. To find out more information about the `plot()` function, type `?plot`.

```
x=rnorm(100)
# The rnorm() function generates a vector of random normal variables,
# rnorm() with first argument n the sample size. Each time we call this
# function, we will get a different answer.
y=rnorm(100)
plot(x,y)
```

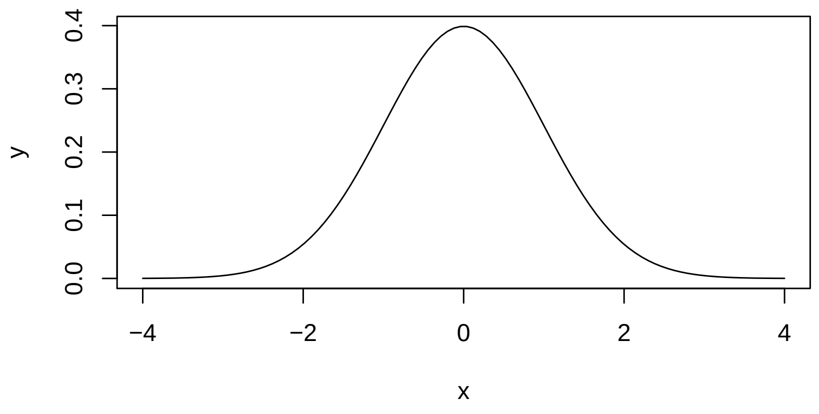


```
# with titles
plot(x,y,xlab="this is the x-axis",ylab="this is the y-axis",
main="Plot of X vs Y")
```



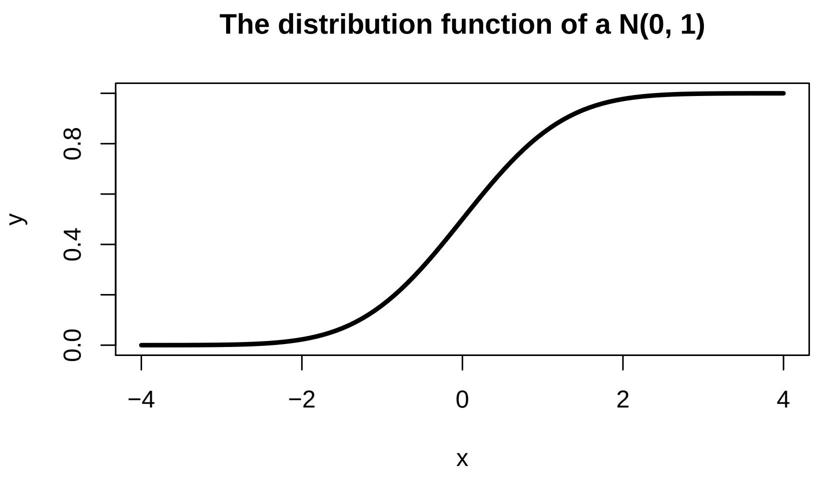
1.8.5 Distributions

```r
# R allows to sample [r], compute density/probability mass [d],
# compute distribution function [p] and compute quantiles [q] for several
# continuous and discrete distributions. The format employed is [rdpq]name,
# where name stands for:
# - norm -> Normal
# - unif -> Uniform
# - exp -> Exponential
# - t -> Student's t
# - f -> Snedecor's F (Fisher)
# - chisq -> Chi squared
# - pois -> Poisson
# - binom -> Binomial
# More distributions: ?Distributions


# Sampling from a Normal - 100 random points from a N(0, 1)
rnorm(n = 10, mean = 0, sd = 1)
#ans>  [1]  0.189  1.246  1.302 -2.141  0.533 -0.725 -1.291 -1.815 -0.558 -1.132

# If you want to have always the same result, set the seed of the random number
# generator
set.seed(45678)
rnorm(n = 10, mean = 0, sd = 1)
#ans>  [1]  1.440 -0.720  0.671 -0.422  0.378 -1.667 -0.508  0.443 -1.799 -0.618

# Plotting the density of a N(0, 1) - the Gauss bell
x <- seq(-4, 4, l = 100)
y <- dnorm(x = x, mean = 0, sd = 1)
plot(x, y, type = "l")
```

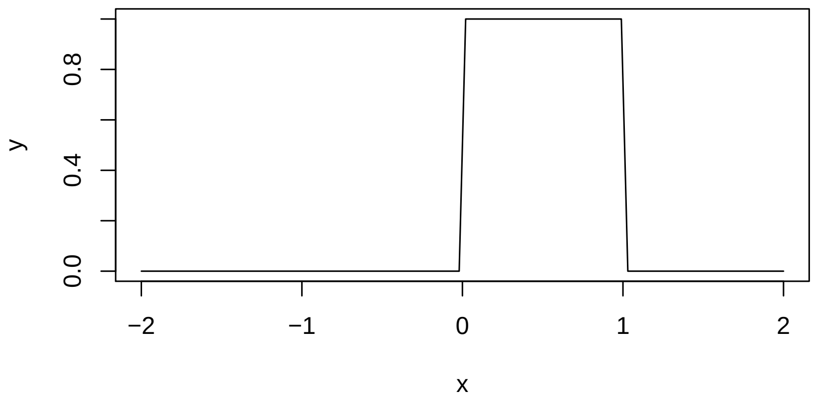


```r
# Plotting the distribution function of a N(0, 1)
x <- seq(-4, 4, l = 100)
```

```r
y <- pnorm(q = x, mean = 0, sd = 1)
plot(x, y, type = "l", lwd = 3, main="The distribution function of a N(0, 1)")
```



```r
# Computing the 95% quantile for a N(0, 1)
qnorm(p = 0.95, mean = 0, sd = 1)
#ans> [1] 1.64

# All distributions have the same syntax: rname(n,...), dname(x,...), dname(p,...)
# and qname(p,...), but the parameters in ... change. Look them in ?Distributions
# For example, here is que same for the uniform distribution

# Sampling from a U(0, 1)
set.seed(45678)
runif(n = 10, min = 0, max = 1)
#ans>  [1] 0.9251 0.3340 0.2359 0.3366 0.7489 0.9327 0.3365 0.2246 0.6474 0.0808

# Plotting the density of a U(0, 1)
x <- seq(-2, 2, l = 100)
y <- dunif(x = x, min = 0, max = 1)
plot(x, y, type = "l")
```



```r
# Computing the 95% quantile for a U(0, 1)
qunif(p = 0.95, min = 0, max = 1)
#ans> [1] 0.95
```

Do the following:

- Compute the 90%, 95% and 99% quantiles of a F distribution with `df1 = 1` and `df2 = 5`. (Answer: `c(4.060420, 6.607891, 16.258177)`)
- Sample 100 points from a Poisson with `lambda = 5`.
- Plot the density of a t distribution with `df = 1` (use a sequence spanning from `-4` to `4`). Add lines of different colors with the densities for `df = 5`, `df = 10`, `df = 50` and `df = 100`.

1.8.6 Working directory

Your *working directory* is the folder on your computer in which you are currently working. When you ask R to open a certain file, it will look in the working directory for this file, and when you tell R to save a data file or figure, it will save it in the working directory.

To set your working directory within RStudio you can go to `Tools / Set working directory`, or use the command `setwd()`, we put the complete path of the directory between the brackets, do not forget to put the path into quotation marks `""`.

To know the actual working directory we use `getwd()`.

1.8.7 Loading Data

The `read.table()` function is one of the primary ways to import a data set into `R`. The help file `?read.table()` contains details about how to use this function. We can use the function `write.table()` to export data.

Next we will show how to load the data set `Auto.data`.

```
Auto=read.table("Auto.data",header=T,na.strings ="?")
# For this file we needed to tell R that the first row is the
# names of the variables.
# na.strings tells R that any time it sees a particular character
# or set of characters (such as a question mark), it should be
# treated as a missing element of the data matrix.
```

- If the file is of csv format, we use `read.csv`.
- Always try to look to the file before importing it to R (Open it in a text editor. See for example if the first row containes the variables names, if the columns are separated by `,` or `;` or ..
- For text editors, I suggest `Sublime Text` or `Atom`.

```
dim(Auto) # To see the dimensions of the data set
#ans> [1] 397   9
nrow(Auto) # To see the number of rows
#ans> [1] 397
ncol(Auto) # To see the number of columns
#ans> [1] 9
Auto[1:4,] # The first 4 rows of the data set
#ans>   mpg cylinders displacement horsepower weight acceleration year origin
#ans> 1  18         8          307        130   3504         12.0   70      1
#ans> 2  15         8          350        165   3693         11.5   70      1
#ans> 3  18         8          318        150   3436         11.0   70      1
#ans> 4  16         8          304        150   3433         12.0   70      1
#ans>                        name
#ans> 1 chevrolet chevelle malibu
#ans> 2         buick skylark 320
#ans> 3        plymouth satellite
#ans> 4             amc rebel sst
```

```
# Once the data are loaded correctly, we can use names()
# to check the variable names.
names(Auto)
#ans> [1] "mpg"          "cylinders"    "displacement" "horsepower"
#ans> [5] "weight"       "acceleration" "year"         "origin"
#ans> [9] "name"
```

Take a look at this (very) short introduction to R. It can be useful.

1.9 Regression

1.9.1 The `lm` function

We are going to employ the `EU` dataset. The `EU` dataset contains 28 rows with the member states of the European Union (Country), the number of seats assigned under different years (Seats2011, Seats2014), the Cambridge Compromise apportionment (CamCom2011), and the countries population (Population2010,Population2013).

```
# Load the dataset, when we load an .RData using load()
# function we do not attribute it to a name like we did
# when we used read.table() or when we use read.csv()

load("EU.RData")
```

There is two ways to tell `R` where is the file you want to load/use/import or where to save a file when you write/export/save :

1. write the complete path of the files.
2. set a working directory and put the files in it.

```
# lm (for linear model) has the syntax:
# lm(formula = response ~ predictor, data = data)
# The response is the y in the model. The predictor is x.
# For example (after loading the EU dataset)
mod <- lm(formula = Seats2011 ~ Population2010, data = EU)

# We have saved the linear model into mod, which now contains all the output of lm
# You can see it by typing
mod
#ans>
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans>
#ans> Coefficients:
#ans>    (Intercept)  Population2010
#ans>       7.91e+00        1.08e-06

# mod is indeed a list of objects whose names are
names(mod)
#ans>  [1] "coefficients"  "residuals"     "effects"       "rank"
#ans>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#ans>  [9] "na.action"     "xlevels"       "call"          "terms"
#ans> [13] "model"

# We can access these elements by $
# For example
mod$coefficients
#ans>    (Intercept) Population2010
#ans>       7.91e+00       1.08e-06
```

```
# The residuals
mod$residuals
#ans>        Germany         France United Kingdom          Italy          Spain
#ans>         2.8675        -3.7031        -1.7847         0.0139        -3.5084
#ans>         Poland        Romania    Netherlands         Greece        Belgium
#ans>         1.9272         1.9434         0.2142         1.8977         2.3994
#ans>       Portugal Czech Republic        Hungary         Sweden        Austria
#ans>         2.6175         2.7587         3.2898         2.0163         2.0575
#ans>       Bulgaria        Denmark       Slovakia        Finland        Ireland
#ans>         1.9328        -0.8790        -0.7606        -0.6813        -0.7284
#ans>      Lithuania         Latvia       Slovenia        Estonia         Cyprus
#ans>         0.4998        -1.3347        -2.1175        -3.3552        -2.7761
#ans>     Luxembourg          Malta
#ans>        -2.4514        -2.3553

# The fitted values
mod$fitted.values
#ans>        Germany         France United Kingdom          Italy          Spain
#ans>          96.13          77.70          74.78          72.99          57.51
#ans>         Poland        Romania    Netherlands         Greece        Belgium
#ans>          49.07          31.06          25.79          20.10          19.60
#ans>       Portugal Czech Republic        Hungary         Sweden        Austria
#ans>          19.38          19.24          18.71          17.98          16.94
#ans>       Bulgaria        Denmark       Slovakia        Finland        Ireland
#ans>          16.07          13.88          13.76          13.68          12.73
#ans>      Lithuania         Latvia       Slovenia        Estonia         Cyprus
#ans>          11.50          10.33          10.12           9.36           8.78
#ans>     Luxembourg          Malta
#ans>           8.45           8.36

# Summary of the model
sumMod <- summary(mod)
sumMod
#ans>
#ans> Call:
#ans> lm(formula = Seats2011 ~ Population2010, data = EU)
#ans>
#ans> Residuals:
#ans>    Min     1Q Median     3Q    Max
#ans> -3.703 -1.951  0.014  1.980  3.290
#ans>
#ans> Coefficients:
#ans>                Estimate Std. Error t value Pr(>|t|)
#ans> (Intercept)    7.91e+00   5.66e-01    14.0  2.6e-13 ***
#ans> Population2010 1.08e-06   1.92e-08    56.3  < 2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> Residual standard error: 2.29 on 25 degrees of freedom
#ans>   (1 observation deleted due to missingness)
#ans> Multiple R-squared:  0.992,   Adjusted R-squared:  0.992
#ans> F-statistic: 3.17e+03 on 1 and 25 DF,  p-value: <2e-16
```

The following table contains a handy cheat sheet of equivalences between `R` code and some of the statistical concepts associated to linear regression.

R	Statistical concept
`x`	Predictor $X_1, \ldots, X_n$
`y`	Response $Y_1, \ldots, Y_n$
`data <- data.frame(x = x, y = y)`	Sample $(X_1, Y_1), \ldots, (X_n, Y_n)$
`model <- lm(y ~ x, data = data)`	Fitted linear model
`model$coefficients`	Fitted coefficients $\hat{\beta}_0, \hat{\beta}_1$
`model$residuals`	Fitted residuals $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_n$
`model$fitted.values`	Fitted values $\hat{Y}_1, \ldots, \hat{Y}_n$
`model$df.residual`	Degrees of freedom $n-2$
`summaryModel <- summary(model)`	Summary of the fitted linear model
`summaryModel$sigma`	Fitted residual standard deviation $\hat{\sigma}$
`summaryModel$r.squared`	Coefficient of determination R^2
`summaryModel$fstatistic`	F-test
`anova(model)`	ANOVA table

Do the following:

- Download The ‘EU’ dataset from here as an `.RData` file and load it using the function `load`.
- Compute the regression of `CamCom2011` into `Population2010`. Save that model as the variable `myModel`.
- Access the objects `residuals` and `coefficients` of `myModel`.
- Compute the summary of `myModel` and store it as the variable `summaryMyModel`.
- Access the object `sigma` of `myModel`.

1.9.2 Predicting House Value: Boston dataset

We are going to use a dataset called Boston which is part of the `MASS` package. It recordes the median value of houses for 506 neighborhoods around Boston. Our task is to predict the median house value (`medv`) using only one predictor (`lstat`: percent of households with low socioeconomic status).

```r
# First, install the MASS package using the command: install.packages("MASS")

# load MASS package
library(MASS)

# Check the dimensions of the Boston dataset
dim(Boston)
#ans> [1] 506  14
```

STEP 1: Split the dataset

```r
# Split the data by using the first 400 observations as the training
# data and the remaining as the testing data
train = 1:400
test = -train
```

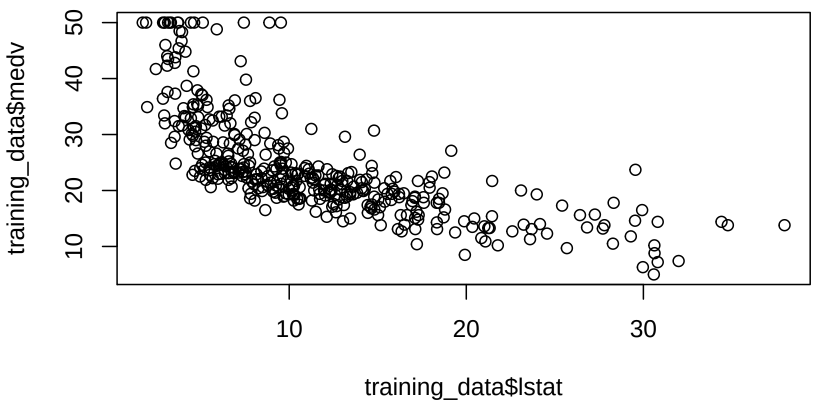
```
# Speficy that we are going to use only two variables (lstat and medv)
variables = which(names(Boston) ==c("lstat", "medv"))
training_data = Boston[train, variables]
testing_data = Boston[test, variables]

# Check the dimensions of the new dataset
dim(training_data)
#ans> [1] 400   2
```

STEP 2: Check for Linearity

In order to perfom linear regression in `R`, we will use the function `lm()`to fit a simple linear regression with `medv` as the response (dependent variable) and `lstat` as the predictor or independent variable, and then save it in `model`.

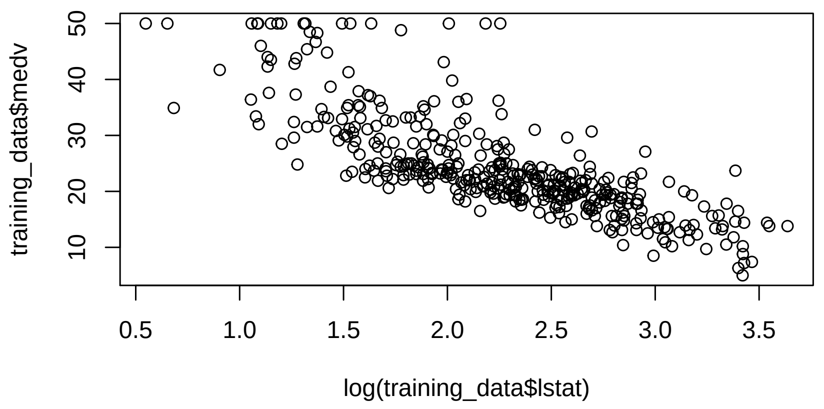
But before we run our model, let's visually check if the relationship between x and y is linear.

```
# Scatterplot of lstat vs. medv
plot(training_data$lstat, training_data$medv)
```



According to the plot, we see that the relationship is not linear. Let's try a transformation of our explanatory variable `lstat`.

```
# Scatterplot of log(lstat) vs. medv
plot(log(training_data$lstat), training_data$medv)
```



Look at the plot, it is more linear, so we can proceed and perform `lm()`:

STEP 3: Run the linear regression model

```
model = lm(medv ~ log(lstat), data = training_data)
model
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Coefficients:
#ans> (Intercept)   log(lstat)
#ans>        51.8        -12.2
```

Notice that basic information when we print `model`. This only give us the slope (-12.2) and the intercept (51.8) of the linear model. Note that here we are looking at `log(lstat)` and not `lstat` anymore. So for every one unit increase in `lstat`, the median value of the house will decrease by $e^{12.2}$. For more detailed information, we can use the `summary()` function:

```
summary(model)
#ans>
#ans> Call:
#ans> lm(formula = medv ~ log(lstat), data = training_data)
#ans>
#ans> Residuals:
#ans>     Min      1Q  Median      3Q     Max
#ans> -11.385  -3.908  -0.779   2.245  25.728
#ans>
#ans> Coefficients:
#ans>             Estimate Std. Error t value Pr(>|t|)
#ans> (Intercept)   51.783      1.097    47.2   <2e-16 ***
#ans> log(lstat)   -12.203      0.472   -25.9   <2e-16 ***
#ans> ---
#ans> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#ans>
#ans> Residual standard error: 5.6 on 398 degrees of freedom
```

```
#ans> Multiple R-squared:  0.627,   Adjusted R-squared:  0.626
#ans> F-statistic:  669 on 1 and 398 DF,  p-value: <2e-16
```

Now, we have access to p-values and standard errors for the coefficients, as well as the R^2.

- The output states that the slope is statistically significant and different from 0 and with a t-value$= -25.9$ (p-value < 0.05), which means that there is a significant relationship between the percentage of households with low socioeconomic income and the median house value.
- This relationship is negative. That is as the percantage of household with low socioeconomic income increases, the median house value decreases.
- Looking at R^2, we can deduce that 62.7% of the model variation is being explained by the predictor `log(lstat)`. This is probably low, but indeed it would increase if we had more independent (explanatory) variables. We can use the `names()` function to see what other pieces of information are stored in our linear model (`model`).

```
names(model)
#ans>  [1] "coefficients" "residuals"     "effects"       "rank"
#ans>  [5] "fitted.values" "assign"        "qr"            "df.residual"
#ans>  [9] "xlevels"       "call"          "terms"         "model"
```

```
model$coefficients
#ans> (Intercept)  log(lstat)
#ans>        51.8       -12.2
```

To obtain the confidence intervel for the linear model (`model`), we can use the `confint()` function:

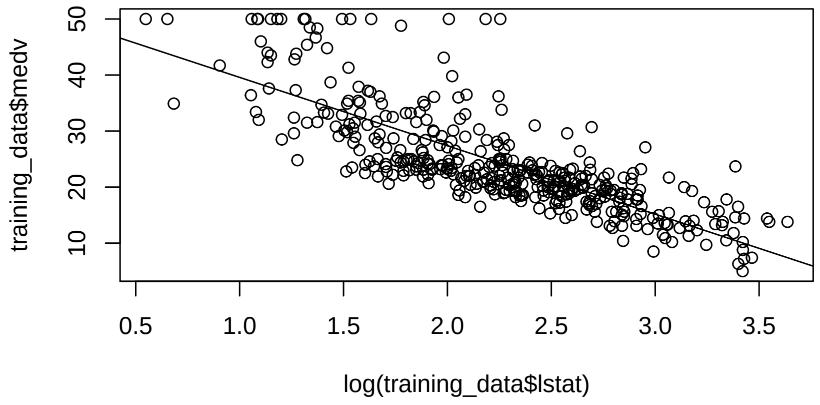
```
confint(model, level = 0.95)
#ans>             2.5 % 97.5 %
#ans> (Intercept)  49.6   53.9
#ans> log(lstat)  -13.1  -11.3
```

So, a 95% confidence interval for the slope of `log(lstat)` is $(-13.13, -11.28)$. Notice that this confidence interval gives us the same result as the hypothesis test performed earlier, by stating that we are 95% confident that the slope of `lstat` is not zero (in fact it is less than zero, which means that the relationship is negative.)

STEP 4: Plot the regression model

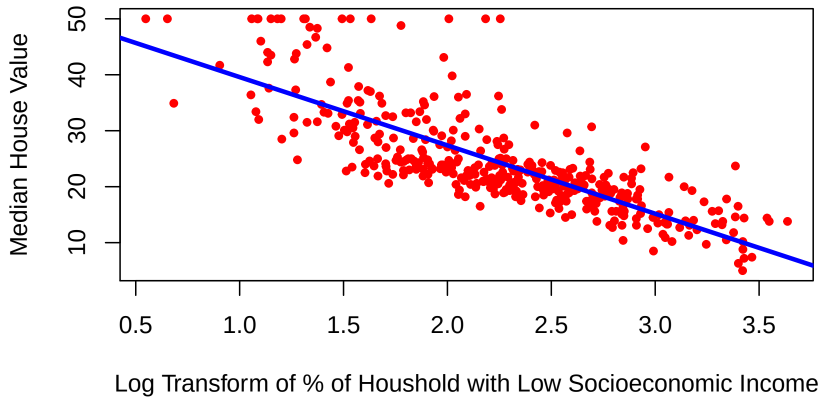
Now, let's plot our regression line on top of our data.

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv)

# Add the regression line to the existing scatterplot
abline(model)
```



Let's play with the look of the plot, and makes it perttier!

```
# Scatterplot of lstat vs. medv
plot(log(training_data$lstat), training_data$medv,
xlab = "Log Transform of % of Houshold with Low Socioeconomic Income",
ylab = "Median House Value",
col = "red",
pch = 20)

# Make the line color blue, and the line's width =3 (play with the width!)
abline(model, col = "blue", lwd =3)
```



STEP 5: Assess the model

Final thing we will do is to predict using our fitted model. We can use the `predict()` function for this purpose:

```
# Predict what is the median value of the house with lstat= 5%
predict(model, data.frame(lstat = c(5)))
#ans>    1
#ans> 32.1
```

```
# Predict what is the median values of houses with lstat= 5%, 10%, and 15%
predict(model, data.frame(lstat = c(5,10,15), interval = "prediction"))
#ans>    1    2    3
#ans> 32.1 23.7 18.7
```

Now let’s assess our model, by computing th mean squared error (MSE). To assess the model we created, then we will be using the test data!

```
# Save the testing median values for houses (testing y) in y
y = testing_data$medv

# Compute the predicted value for this y (y hat)
y_hat = predict(model, data.frame(lstat = testing_data$lstat))

# Now we have both y and y_hat for our testing data.
# let's find the mean square error
error = y-y_hat
error_squared = error^2
MSE = mean(error_squared)
MSE
#ans> [1] 17.7
```