# Module 8: Introduction to Machine Learning Part 1

*"We are drowning in information and starving for knowledge." - Rutherford D. Roger*

## Introduction

The next two modules are designed to give an introduction to Machine Learning (ML).

## Learning Outcomes

In this module you will learn and practice:

- Different types of machine learning problems
- Supervised and unsupervised learning
- Regression as a central tool in supervised ML
- How to build prediction models
- How to evaluate models
- Statistical meaning of regression analysis

## Readings and Resources

The majority of the notebook content draws from the recommended readings. We invite you to further supplement this notebook with the following recommended texts:

Geron, A. (2017) *Hands-on Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media.

Witten, I. H. and Frank, E. (2005) *Data Mining. Practical Machine Learning Tools and Techniques* (2nd edition). Elsevier.

`statsmodels` Documentation can be found at https://www.statsmodels.org/dev/index.html.

`scikit-learn` Documentation can be found at http://scikit-learn.org/stable/documentation.html.

## Table of Contents

# Introduction to Machine Learning

## Creating, Testing, Validating, Evaluating and Deploying the Model

This image shows the stages of creating a model from start to finish. (Course Authors, 2018)
*This image shows the stages of creating a model from start to finish.*

Throughout this course, you have gained experience working with Python and its libraries, and you should now have an understanding of the analytics methodology we use to build models.

Now that you are familiar with data preparation methods and forecasting, in this module we will introduce Machine Learning. Development of machine learning models involves activities which span the final 5 phases of the methodology:

1. Create the Model
2. Test the Model

3. Validate the Model
4. Evaluate the Model
5. Deploy the Model

Prepared datasets are used to train a model and test its accuracy. We will also discuss model validation and evaluation, as these are critical steps to complete before you deploy a model for business use. In theory, we would like to use machine learning models to help us predict outcomes so we can make better business decisions. This is why we need to ensure our models are not biased, or skewed, and that they make appropriate assumptions based on the data.

Nowadays, machine learning algorithms are ubiquitous. They are literally conquering the industry. Here are some examples:

- **Medical diagnosis**: Determine a diagnosis, given features such as test results, symptoms, and treatments
- **Loan applications**: Develop an algorithm to decide whether to grant a loan by analyzing input values such as credit scores, income, education level and marital status
- **Churn prediction**: In this class of problems, we use machine learning to help us define a strategy for keeping customers. For example, changes in usage patterns are analyzed.
- **Product consumption**: Predict product consumption given features such as promotions, advertising, socio-demographics, and concurrent activity
- **Rank problems**: Online stores use product ranking. When clients can score items, a rank problem is a prediction of the item rank prior to purchase. To achieve this, client preferences and item rankings are analyzed. Netflix also uses this type of algorithm.
- **Market basket analysis**: Analyze which products are purchased together to inform shelving and promotion planning decisions
- **Stock prices**: Predict the price of a stock from company performance measures and economic data
- **Image recognition**: Identify the digits in a handwritten postal code from a digital image
- **Risk factors**: In this class of problems, risk factors for disease are identified from clinical and demographic variables

This list is by no means exhaustive. Think of other big classes that you may be aware of such as natural language processing.

## Definitions of Machine Learning

Even though most of us have heard about machine learning and have an idea what machine learning is, it is a bit difficult to give a strict definition. The overview we just explored suggests it's about learning from data using an algorithm. Usually we have outcome measurements, either:

1. Quantitative
    - i.e. Product rank, stock price
2. Categorical
    - i.e. Yes or No for granting a loan, or for risk factors for getting disease

We want to *predict* these outcomes based on a set of *features* (e.g. credit scores, household income, etc.).

Also, we should not expect machines to learn in the same way that humans do. For humans, learning implies thinking and purpose. See Chapter 1 of (Witten, 2005) for more discussion. However, some aspects of machine learning are similar to human learning. For computers, *training* might be a more appropriate term, but learning is already a widely used term.

There are two popular definitions of machine learning (Ng, 2018):

- Arthur Samuel describes machine learning as: "the field of study that gives computers the ability to learn without being explicitly programmed."
- A more formal and modern definition is provided by Tom Mitchell, "A computer program is said to **learn** from the **experience E** with respect to some class of **tasks T** and **performance measure P**, if its performance at tasks T, as measured by P, improves with experience E."

Let's use the example of playing checkers to illustrate Mitchell's definition:

| *Variable* | *Problem Definition* |
|---:|---|
| **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 |

## Machine learning model: Predictor and target

Each example we have discussed can be described as a set of labeled data. Each instance is a pair, that we can denote as:

$$ \{x^{(i)}, y^{(i)}\} $$
where $ x^{(i)} $ are **input variables**; also called input **features** or **predictors**, and $ y^{(i)} $ are **output** or **target** variables, that we can learn how to predict.

Usually, we are solving problems with multiple variables, so $ x^{(i)} $ is a vector or set of values. Actually, all examples above are multi-variable problems. In the loan application problem, the predictor variables are credit scores, income, applicant's education level and marital status. The target is the loan status (approved or not).

Sometimes, there might be a single variable problem where there is only one x variable (predictor variable, P) which needs to be considered. For example, consider a simplified problem to determine how house price depends on the size (square footage) of the house. Then, the square footage, $ x^{(i)} $, is used to predict the house price, $ y^{(i)} $. $x^{(i)}$ (the size if the house) is called the **predictor** variable and $ y^{(i)} $ (the house price) is the **target**. A pair $ ( x^{(i)}, y^{(i)}) $ is called a **training** example. The list of all available training examples forms a training dataset. Note that the superscript $ (i) $ is used as an index in the training dataset.

With such defined variables, the more formal definition of the machine learning problem can be formulated as a *goal* to learn a function, $ y = F(x) $, on a given training dataset. Ultimately, the function, $ F(x) $, will be used to predict the corresponding value, $ y $. So, in our example for a new house of known size on the market, we can predict its price using the $ F(x) $ function. The function $ F(x) $ is also called the **hypothesis** and the goal of machine learning in this case is to find the relationship $ F(x) $ between predictors $ x $ and responses $ y $.

## Supervised and unsupervised learning

Generally, machine learning problems are classified into two broad categories: **supervised** and **unsupervised** learning.

In **supervised** learning, the output (target) values are given in the training dataset. Most of the examples considered so far are supervised learning examples: the function $ y = F(x) $ is learned from labeled pairs $ (x^{(i)}, y^{(i)})$. When we are solving stock price value, we have a dataset with $ N $ examples of stock prices versus company performance measures and economic data, or we are given a set of pairs:

( (company performance measures, economic data), stock price )

> **NOTE:** $ x^{(i)} $ is multi-variable — 2-dimensional in this case. It comprises both the company performance measures as well as the economic data.

This type of learning is called supervised, because the learning process operates as though under supervision — a set of given examples of outcomes show what correct output should look like.

In **unsupervised** learning, the problem is approached with little or no idea of what outcome results should look like, data isn't labeled, or there is no target variable. This method aims to create groups of data points. Structural patterns in data can be found by grouping or **clustering** the data. Clustering is based on the relationships among the variables in the data. Data that belongs in the same cluster are *close* to each other, while data that belong to different clusters are in general apart. Identification of the numbers in a handwritten ZIP code is an example of unsupervised learning.

For example, the MNIST dataset (https://en.wikipedia.org/wiki/MNIST_database) is a database of handwritten digits and is used as a training dataset for various image processing systems. The dataset was obtained by scanning thousands of handwritten digits where each scan is characterized by brightness of pixels. The machine learning problem involves clustering or grouping data into 10 groups where each group corresponds to a certain digit. This is unsupervised learning since each scan is not labeled and there is no prior knowledge that a certain pattern of the pixel brightness on 2-dimensional area will correspond to a certain digit (0, 1 or any other).

Other examples of unsupervised learning algorithms are:

- **Visualization algorithms**: They produce a 2- or 3-dimensional representation from complex and unlabeled data and their outputs can be easily plotted
- **Dimensionality reduction**: Algorithms of this type simplify data without losing too much information
- **Anomaly detection**: They detect outliers or observations that are very different form the rest of dataset

For further reading: https://en.wikipedia.org/wiki/Unsupervised_learning#Approaches.

## Supervised learning: Regression and Classification

Unsupervised learning methods will be considered in later courses. In this module, we will continue with supervised learning.

You may have noticed that some target values are numeric (continuous), such as house prices. Thus, we are trying to learn a *continuous* function $ F(x) $ that maps input variables. This category of supervised learning problems is called **regression**. Learning house price as a function of the size of the house, and stock price as a function of economic data coupled with company performance measures are examples of regression problems or *regressions*, for short.

Other supervised learning problems involve predicting a **discrete** result as an output also known as a *categorical* response. This type of learning task is called **classification**. Classification is different from clustering. Clustering is an unsupervised learning algorithm where unlabeled data are grouped into clusters by similarity. In classification problems, the output is known in the training dataset and the goal is to learn the predicting function $ F(x) $ to map input variables into **discrete** categories. In the loan application problem the output is a grant status *Yes* or *No*; "Yes" can be represented by 1 and "No" by 0, or alternatively 1 for "Yes" and -1 for "No."

In this course, we will cover the most common supervised learning algorithms:

- Linear Regression
- Logistic Regression
- k-Nearest Neighbors (kNN)
- Support Vector Machines (SVM)
- Decision Trees and Random Forests

## Linear Regression

Linear regression is an algorithm for finding the linear relationship between predictors and responses. It is applicable when the response is a numeric variable. Linear regression models the relationship between a scalar dependent variable y (the output) and one or more independent variables x (the inputs). The objective is to find the **line of best fit**:

$$ y = w_0 + \sum_{i=1}^L w_i x_i $$

where

- $ x_i $ are features
- $ w_i $ are the coefficients or weights of each feature
- $ w_0 $ is an intercept, or the value of the response when all $ x_i $ are equal to 0

Quite often, the formula is written in a more compact form, since the intercept $ w_0 $ can be viewed as the coefficient for a constant variable equal to one (i.e. $ x_i^0 $), and thus the formula can be rewritten as:

$$ y = \sum_{i=0}^L w_i x_i $$

In the case of one-variable prediction, it takes a simpler form:

$$ y = w_0 + w_1 x_1 $$

Let us consider an example of linear regression. In Python, there are two big libraries with linear regression tools:

- `statsmodels`
- Scikit-learn ( `sklearn` )

First, we will use `statsmodels`. The example we will explore is adapted from Connor Johnson's blog (http://connor-johnson.com/2014/02/18/linear-regression-with-python/) and we will be analyzing the correlation between tobacco and alcohol purchases in different regions of the United Kingdom.

**NOTE:** The data used in this article can be found on the DASL website: the **Data And Story Library**.

In [1]:
```
import numpy as np
import pandas
#import pandas as pd
from pandas import DataFrame, Series
import statsmodels.formula.api as sm
from sklearn.linear_model import LinearRegression
import scipy, scipy.stats
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
'''
The data is given as a multi-line string
and describes alcohol and tobacco consumption in UK regions.
'''

data_str = '''Region\tAlcohol\tTobacco
North\t6.47\t4.03
Yorkshire\t6.13\t3.76
Northeast\t6.19\t3.77
East Midlands\t4.89\t3.34
West Midlands\t5.63\t3.47
East Anglia\t4.52\t2.92
Southeast\t5.89\t3.20
Southwest\t4.79\t2.71
Wales\t5.27\t3.53
Scotland\t6.08\t4.51
Northern Ireland\t4.02\t4.56'''
```

In [3]:
```
'''
In order to prepare the data for a Pandas DataFrame,
we will split the data into a list of lists.
'''

#First we will make list of rows
d = data_str.split('\n')

#Next, each row can be split by 'tab' into list
d = [ i.split('\t') for i in d ]

d
```

Out[3]:
```
[['Region', 'Alcohol', 'Tobacco'],
 ['North', '6.47', '4.03'],
 ['Yorkshire', '6.13', '3.76'],
 ['Northeast', '6.19', '3.77'],
 ['East Midlands', '4.89', '3.34'],
 ['West Midlands', '5.63', '3.47'],
 ['East Anglia', '4.52', '2.92'],
 ['Southeast', '5.89', '3.20'],
 ['Southwest', '4.79', '2.71'],
 ['Wales', '5.27', '3.53'],
 ['Scotland', '6.08', '4.51'],
 ['Northern Ireland', '4.02', '4.56']]
```

In [4]:
```
'''
This list of lists is ready for upload into a DataFrame.
But first, we change the type from numerical strings to floats.
'''

for i in range( len( d ) ):
    for j in range( len( d[0] ) ):
        try:
            d[i][j] = float( d[i][j] )
        except:
            pass
```

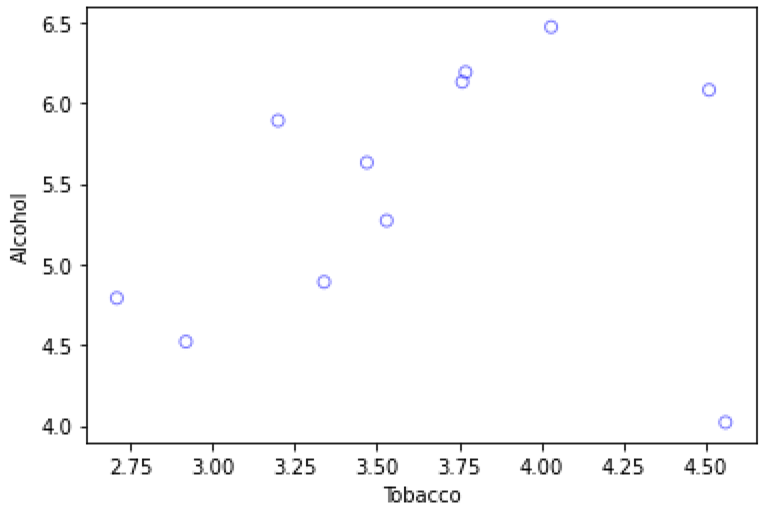
In [5]:
```
'''And finally create the DataFrame'''

df = DataFrame( d[1:], columns=d[0] )
```

In [6]:
```
df
```

Out[6]:

| | Region | Alcohol | Tobacco |
|---|---|---|---|
| **0** | North | 6.47 | 4.03 |
| **1** | Yorkshire | 6.13 | 3.76 |
| **2** | Northeast | 6.19 | 3.77 |
| **3** | East Midlands | 4.89 | 3.34 |
| **4** | West Midlands | 5.63 | 3.47 |
| **5** | East Anglia | 4.52 | 2.92 |
| **6** | Southeast | 5.89 | 3.20 |
| **7** | Southwest | 4.79 | 2.71 |
| **8** | Wales | 5.27 | 3.53 |
| **9** | Scotland | 6.08 | 4.51 |
| **10** | Northern Ireland | 4.02 | 4.56 |

In [7]:
```
plt.scatter( df.Tobacco, df.Alcohol,
             marker='o',
         edgecolor='b',
         facecolor='none',
         alpha=0.5 )
plt.xlabel('Tobacco')
plt.ylabel('Alcohol')
None
```



This scatter plot shows that one observation falls far from the rest of data or "cloud" of points. Such points are called **outliers** and they can have a strong influence on the regression line.

First, linear regression will be run without outliers. Since it happened to be the last row in the `Dataframe` it is very easy to exclude.

In [15]:
```
# For the intercept coefficient, we add new column of 1
df['Eins'] = np.ones((len(df), ))

# Define X and Y variables for the regression.
Y = df.Alcohol[:-1]
X = df[['Tobacco','Eins']][:-1]
```

In [16]:
```
df
```

Out[16]:

| | Region | Alcohol | Tobacco | Eins |
|---|---|---|---|---|
| **0** | North | 6.47 | 4.03 | 1.0 |
| **1** | Yorkshire | 6.13 | 3.76 | 1.0 |
| **2** | Northeast | 6.19 | 3.77 | 1.0 |
| **3** | East Midlands | 4.89 | 3.34 | 1.0 |
| **4** | West Midlands | 5.63 | 3.47 | 1.0 |
| **5** | East Anglia | 4.52 | 2.92 | 1.0 |
| **6** | Southeast | 5.89 | 3.20 | 1.0 |
| **7** | Southwest | 4.79 | 2.71 | 1.0 |
| **8** | Wales | 5.27 | 3.53 | 1.0 |
| **9** | Scotland | 6.08 | 4.51 | 1.0 |
| **10** | Northern Ireland | 4.02 | 4.56 | 1.0 |

In [18]:

```
'''
This will run the fit, here we are going to use the Ordinary Least Squares (OLS) al
'''
import statsmodels.api as sm
# This will create an instance of Linear Regression
result_no_outlier = sm.OLS( Y, X).fit()
```

In [19]:

```
# To get the summary of the regression
result_no_outlier.summary()
```

```
/Users/mlsimon/opt/anaconda3/lib/python3.8/site-packages/scipy/stats/stats.py:1603:
UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=10
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Out[19]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Alcohol | **R-squared:** | 0.615 |
| **Model:** | OLS | **Adj. R-squared:** | 0.567 |
| **Method:** | Least Squares | **F-statistic:** | 12.78 |
| **Date:** | Wed, 02 Dec 2020 | **Prob (F-statistic):** | 0.00723 |
| **Time:** | 10:53:19 | **Log-Likelihood:** | -4.9998 |
| **No. Observations:** | 10 | **AIC:** | 14.00 |
| **Df Residuals:** | 8 | **BIC:** | 14.60 |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **Tobacco** | 1.0059 | 0.281 | 3.576 | 0.007 | 0.357 | 1.655 |
| **Eins** | 2.0412 | 1.001 | 2.038 | 0.076 | -0.268 | 4.350 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 2.542 | **Durbin-Watson:** | 1.975 |
| **Prob(Omnibus):** | 0.281 | **Jarque-Bera (JB):** | 0.904 |
| **Skew:** | -0.014 | **Prob(JB):** | 0.636 |
| **Kurtosis:** | 1.527 | **Cond. No.** | 27.2 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

## Interpreting Regression Results

We will take a closer look at some of the statistical output we just generated.

The method `summary()` gives a full statistical evaluation of the regression. We will cover these parameters in detail in the next course. For now, we will focus on the second block of summary output, which contains:

- The ( `R-squared` ) value articulates what percentage of the variability in Y is explained by the model. This value will always be between 0 and 1; if R-squared = 1, then the model perfectly explains variability in Y, while if R-squared = 0, there is no linear relationship.

- The standard errors ( `std err` ). This is the standard error of the coefficients.

- `p-values`, column `P > |t|`. This column refers to probability and will be explained in detail in the next course. Just note that to compute probability, it is assumed that the coefficient is 0 (there is no dependence). Under this assumption, the probability to find the fitted value is `P > |t|`. A small value of `P` means that the assumption about the coefficient being zero is not reasonable. Attributes with a `p-value` (column `P > |t|`) greater than 0.05 are not significant for this model. Attributes with values between 0 and 0.05 are statistically significant predictors of the response. In the given example, `P > |t|` for Tobacco is 0.007. It has a value much smaller than 0.05 so we accept the fit of the regression line.

- 95% confidence interval. The two last columns ( `[0.025 0.975]` ) give the confidence interval, that is, with 95% confidence the coefficient is between these limits.

- The coefficients ( `coef` ) for 'Tobacco' and 'Eins' features. The coefficient for 'Eins' is an intercept of the line.

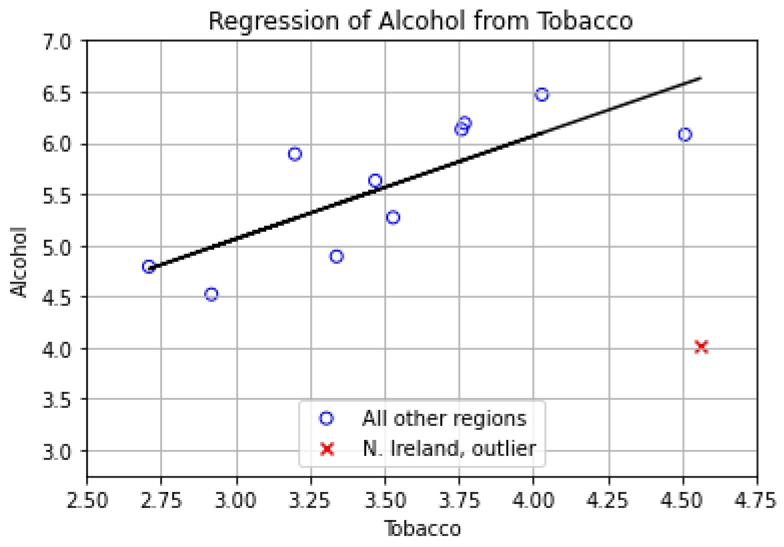
We can get the coefficients from the `params` attribute. Since it tells us the slope and the intercept, this will allow us to plot the "line of best fit." The formula for a line is notated by $ y = mx+b $, with $m$ representing the slope of the line (in this case, the coefficient for Tobacco), and $b$ representing the intercept (the coefficient for Eins). This line will help us determine the nature of the relationship between the two variables.

In [20]:
```
result_no_outlier.params
```

Out[20]:
```
Tobacco     1.005896
Eins        2.041223
dtype: float64
```

In [21]:
```
'''Let's assign these parameters to variables slope and intercept.'''

slope, intercept = result_no_outlier.params
```

In [22]:
```
'''Let's plot the line over the scatter plot'''

plt.scatter( df.Tobacco[:-1], df.Alcohol[:-1],
         marker='o', facecolors='none', edgecolors='b',
         label='All other regions')

plt.scatter( df.Tobacco[-1:], df.Alcohol[-1:],
         marker='x', color='r',
         label='N. Ireland, outlier')

plt.plot( df.Tobacco, intercept + slope*df.Tobacco, 'k' )
plt.xlabel('Tobacco') ;
plt.axis([2.5,4.75,2.75,7.0])
plt.ylabel('Alcohol')
plt.title('Regression of Alcohol from Tobacco') ;
plt.grid() ;
plt.legend(loc='lower center')
plt.show()
```



## Evaluating the Regression Model

As can be seen from the plot, the line fits the observation reasonably well. But how did the chosen algorithm perform this fit, and how were parameters like slope and intercept calculated? The performance measure should be selected for a machine learning algorithm. A typical performance measure for a regression algorithm is the **Mean Square Error (MSE)**.

For each observation, the **residual**, which is the difference between the predicted value $ f(x_i) $ and the response $ y_i $, is calculated. Mean Square Error, also known as the **cost function**, is an average of the squared residuals:

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

where $ N $ is the number of observations in the dataset.

Linear regression models are often fitted by minimizing MSE. This is the known as the **least squares** approach. We used this approach in the example above when the `OLS` method from the `statsmodels` library was called.

Another useful metric is absolute error. In a similar fashion, the **Mean Absolute Error (MAE)** is defined as an average of absolute values of the residuals.

$$ MAE = \frac{1}{N} \sum_{i=1}^N |f(x_i) - y_i| $$

For linear regression models, Mean Squared Error is the preferred performance measure. It gives higher weight for large errors, but more importantly, it is a quadratic function and therefore is suitable for minimization.

## Gradient Descent

Generally, a function that defines how good the model describes the data is called a **cost function**. For linear regression problems, the cost function measures the distance between the linear model's predictions and the training instances.

The mean square error (MSE) and mean absolute error (MAE) are examples of a cost function.

To find the best fit, the *cost function* must be *minimized*. In other words, we need to find coefficient values, $ w_i $, that will minimize the cost function.

**Gradient descent** algorithms provide methods that can be used to calculate coefficients for minimizing the cost function.

In gradient descent, coefficients, $ w_i $, are changed iteratively by small step values in order to ultimately arrive at the minimal cost function. At each iteration, the derivative of the cost function is calculated to find the direction of the next step in order to "descend" to the minimum.

There are two main versions of Gradient Descent methods:

1. **Batch Gradient Descent**: On each iteration, the sum of residuals over all observations is calculated. For large datasets this might be very slow.
2. **Stochastic Gradient Descent**: On each iteration, only one residual is calculated for a random sample of observations from the dataset. This makes the algorithm much faster and suitable for very large datasets.

## Outliers and leverage

Let's get back to our regression example and run another fit, this time including the outlier.

In [22]:
```
X_o = df[['Tobacco','Eins']]
Y_o = df.Alcohol

result_with_outlier = sm.OLS( Y_o, X_o ).fit()
```

In [23]:
```
result_with_outlier.summary()
```

```
/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1394: UserWarning: kurto
sistest only valid for n>=20 ... continuing anyway, n=11
  "anyway, n=%i" % int(n))
```

Out[23]:

OLS Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Alcohol | **R-squared:** | 0.050 |
| **Model:** | OLS | **Adj. R-squared:** | -0.056 |
| **Method:** | Least Squares | **F-statistic:** | 0.4735 |
| **Date:** | Thu, 06 Sep 2018 | **Prob (F-statistic):** | 0.509 |
| **Time:** | 01:06:03 | **Log-Likelihood:** | -12.317 |
| **No. Observations:** | 11 | **AIC:** | 28.63 |
| **Df Residuals:** | 9 | **BIC:** | 29.43 |
| **Df Model:** | 1 | | |
| **Covariance Type:** | nonrobust | | |

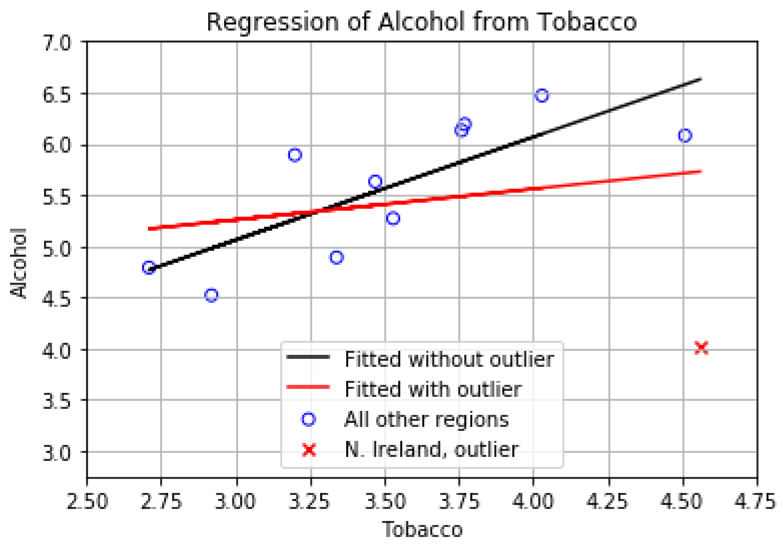
| | **coef** | **std err** | **t** | **P>\|t\|** | **[0.025** | **0.975]** |
|---|---|---|---|---|---|---|
| **Tobacco** | 0.3019 | 0.439 | 0.688 | 0.509 | -0.691 | 1.295 |
| **Eins** | 4.3512 | 1.607 | 2.708 | 0.024 | 0.717 | 7.986 |

| | | | |
|---|---|---|---|
| **Omnibus:** | 3.123 | **Durbin-Watson:** | 1.655 |
| **Prob(Omnibus):** | 0.210 | **Jarque-Bera (JB):** | 1.397 |
| **Skew:** | -0.873 | **Prob(JB):** | 0.497 |
| **Kurtosis:** | 3.022 | **Cond. No.** | 25.5 |

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

In [24]:
```
slope_o, intercept_o = result_with_outlier.params
```

In [25]:
```
'''Let's compare the results of the two fits'''

plt.scatter( df.Tobacco[:-1], df.Alcohol[:-1],
         marker='o', facecolors='none', edgecolors='b',
         label='All other regions')

plt.scatter( df.Tobacco[-1:], df.Alcohol[-1:],
         marker='x', color='r',
         label='N. Ireland, outlier')

plt.plot( df.Tobacco, intercept + slope*df.Tobacco, 'k', label='Fitted without outl
plt.plot( df.Tobacco, intercept_o + slope_o * df.Tobacco , 'r-', label='Fitted with
plt.xlabel('Tobacco') ;
plt.axis([2.5,4.75,2.75,7.0])
plt.ylabel('Alcohol')
plt.title('Regression of Alcohol from Tobacco') ;
plt.grid() ;
plt.legend(loc='lower center')
plt.show()
```



In this example, the influence of the outlier is rather significant. It appears that an outlier point has high influence on the slope of the line. Points like these are called **influential points**; we can say that these points "pull on the line" and are points with **high leverage**.

**NOTE:** It might seem that removing outliers will help us produce better prediction models and it is tempting to discard outliers as insignificant. But outliers must not be removed without thorough analysis. For example, in the analysis of financial data, ignoring the largest market changes — the outliers — may be very costly for the company.

## Linear regression with scikit-learn

Scikit-learn ( `sklearn` ) is a Python machine learning library. In this section we will use Linear Regression from `sklearn` . Let us see how linear regression can be used.

The dataset used here can be downloaded from Kaggle (https://www.kaggle.com/c/bike-sharing-demand). Save the dataset to your machine in the same folder as this notebook with the name 'bikes_sharing.csv'. Alternatively, update the code below to reflect the name and location of the dataset.

This dataset was generated by a bicycle sharing system which is part of the Capital Bikeshare program in Washington, D.C. These systems allow people to rent a bike from one location and return it to a different location on an as-needed basis. The process of membership, rental, and bike return is automated via a network of kiosks located throughout the city. The dataset contains information such as the duration of travel, departure location, arrival location, the time elapsed, etc.

In [17]:
```
import sklearn
```

```
'''We will also use numpy for calculations and Pandas for dataset manipulations'''

import numpy as np
import pandas as pd

%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [18]:
```
'''Let's read data into Pandas dataframe'''
bike_sharing = pd.read_csv('bikes_sharing.csv', header = 0, sep = ',')
```

In [19]:
```
bike_sharing.head(5)
```

Out[19]:

| | datetime | season | holiday | workingday | weather | temp | atemp | humidity | windspeed |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-01-01 00:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 81 | 0.0 |
| 1 | 2011-01-01 01:00:00 | 1 | 0 | 0 | 1 | 9.02 | 13.635 | 80 | 0.0 |
| 2 | 2011-01-01 02:00:00 | 1 | 0 | 0 | 1 | 9.02 | 13.635 | 80 | 0.0 |
| 3 | 2011-01-01 03:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 75 | 0.0 |
| 4 | 2011-01-01 04:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 75 | 0.0 |

Here is a brief description of the data fields:

| Field | Description |
|---|---|
| datetime | hourly date + timestamp |
| season | `1` = spring, `2` = summer, `3` = fall, `4` = winter |
| holiday | whether the day is considered a holiday |
| workingday | whether the day is not a weekend / holiday |
| weather | `1` = Clear, Few clouds, Partly cloudy, Partly cloudy; `2` = Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist; `3` = Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds; `4` = Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog |
| temp | temperature in Celsius |
| atemp | "feels like" temperature in Celsius |

| Field | Description |
|---|---|
| humidity | relative humidity |
| windspeed | wind speed |
| casual | number of non-registered user rentals initiated |
| registered | number of registered user rentals initiated |
| count | number of total rentals |

```
In [20]: '''Before proceeding with regression, first check
         if there are any missing values'''
         bike_sharing.isnull().values.any()
```

```
Out[20]: False
```

```
In [21]: '''And the size of dataframe'''
         bike_sharing.shape
```

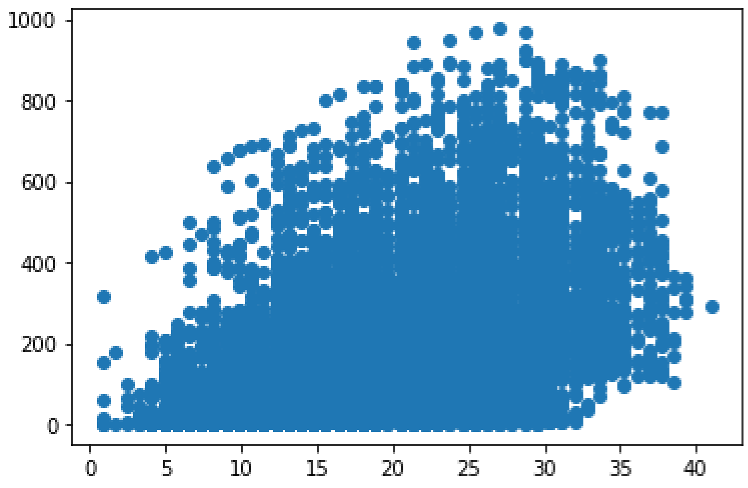
```
Out[21]: (10886, 12)
```

The dataset has no missing values and contains 10,886 observations (rows) and 12 columns.

Let's see if the number of total rentals depends on the air temperature.

```
In [22]: plt.scatter(x=bike_sharing["temp"], y=bike_sharing["count"])
```

```
Out[22]: <matplotlib.collections.PathCollection at 0x1a21199590>
```



This graph does show an upward trend, the number of rentals is increasing with temperature. But since data is collected every hour, it would be more informative to show data at a given hour of a day. Before that, we shall create two new columns *month* and *hour*.

```
In [23]: bike_sharing.datetime = bike_sharing.datetime.apply(pd.to_datetime)
```

```
In [24]: bike_sharing['month'] = bike_sharing.datetime.apply(lambda x : x.month)
         bike_sharing['hour'] = bike_sharing.datetime.apply(lambda x: x.hour)
```

```
bike_sharing.head(5)
```

Out[24]:

| | datetime | season | holiday | workingday | weather | temp | atemp | humidity | windspeed |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2011-01-01 00:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 81 | 0.0 |
| 1 | 2011-01-01 01:00:00 | 1 | 0 | 0 | 1 | 9.02 | 13.635 | 80 | 0.0 |
| 2 | 2011-01-01 02:00:00 | 1 | 0 | 0 | 1 | 9.02 | 13.635 | 80 | 0.0 |
| 3 | 2011-01-01 03:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 75 | 0.0 |
| 4 | 2011-01-01 04:00:00 | 1 | 0 | 0 | 1 | 9.84 | 14.395 | 75 | 0.0 |

In [25]:
```
# Here we fetch only row with hour = 15
bike_sharing_15 = bike_sharing.loc[bike_sharing['hour'] == 15]
```

In [26]:
```
bike_sharing_15.head(5)
```

Out[26]:

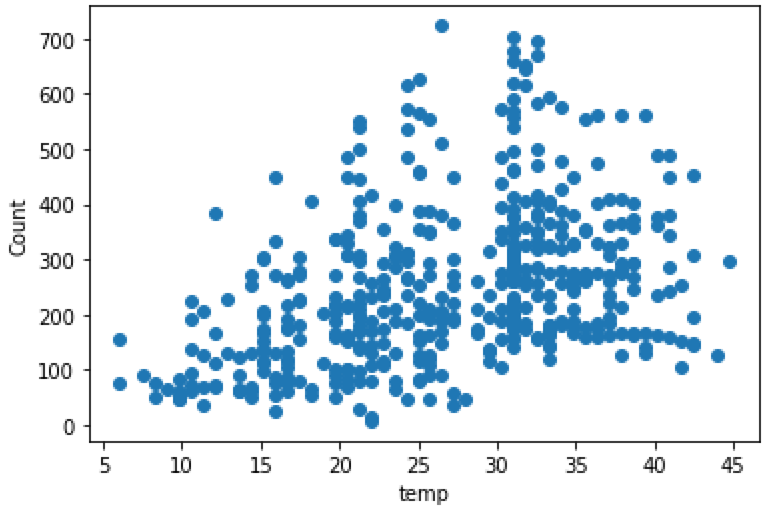
| | datetime | season | holiday | workingday | weather | temp | atemp | humidity | windspeed |
|---|---|---|---|---|---|---|---|---|---|
| 15 | 2011-01-01 15:00:00 | 1 | 0 | 0 | 2 | 18.04 | 21.970 | 77 | 19.9995 |
| 38 | 2011-01-02 15:00:00 | 1 | 0 | 0 | 3 | 13.94 | 16.665 | 81 | 11.0014 |
| 60 | 2011-01-03 15:00:00 | 1 | 0 | 1 | 1 | 10.66 | 12.120 | 30 | 16.9979 |
| 83 | 2011-01-04 15:00:00 | 1 | 0 | 1 | 1 | 11.48 | 13.635 | 52 | 16.9979 |
| 106 | 2011-01-05 15:00:00 | 1 | 0 | 1 | 1 | 12.30 | 14.395 | 28 | 12.9980 |

In [27]:
```
bike_sharing_15.shape
```

```
Out[27]: (456, 14)
```

```
In [28]: plt.scatter(x=bike_sharing_15["atemp"], y=bike_sharing_15["count"])
         plt.xlabel("temp")
         plt.ylabel('Count')
```

```
Out[28]: Text(0, 0.5, 'Count')
```



```
In [29]: # scikit-learn needs the data organized as numpy vectors
         Y = bike_sharing_15["count"]
         X = bike_sharing_15["temp"].values.reshape(-1, 1)

         # the model to do the fit
         model = sklearn.linear_model.LinearRegression().fit(X, Y)
         print(model.coef_)
         print(model.intercept_)
```

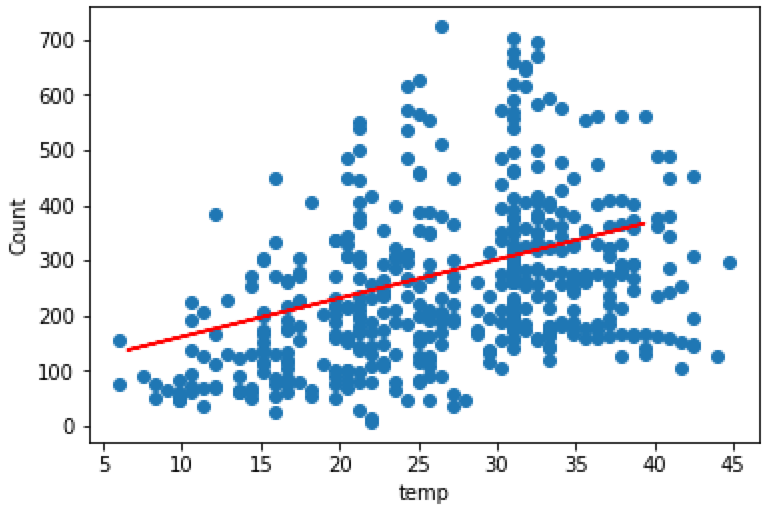
```
[6.99568077]
91.42560375076249
```

```
In [50]: b=bike_sharing_15['temp'].values.reshape(-1,1)
```

```
In [39]: '''Let's calculate predicted value of count,
         using the result of our model with one variable X'''

         lin_predicted_one = model.predict(X)
```

```
In [40]: '''The graph below shows how well the model (line) fits the data'''
         plt.scatter(x=bike_sharing_15["atemp"], y=bike_sharing_15["count"])
         plt.plot(X, model.predict(X), 'r-')
         plt.xlabel("temp")
         plt.ylabel('Count')
```

```
Out[40]: Text(0,0.5,'Count')
```



In [49]:
```
'''As a mesure of "goodness" the Root Mean Square Error is calculated.
The actual data (Y column) and predicted values are compared.'''

from sklearn.metrics import mean_squared_error, r2_score

lin_one_mse= mean_squared_error(Y, lin_predicted_one)
lin_one_rmse = np.sqrt(lin_one_mse)

print(lin_one_rmse)
```

```
133.06710533692635
```

For the best fit, the smallest possible mean square error (RMSE) is expected, but we should not expect zero for a real dataset (zero would mean all observations fall on the straight line). The computed value of RMSE should be compared with the data spread. The graph shows that the calculated values are scattered in wide range between 0 and 700, so that an RSME value of 133.067 seems to be reasonable.

Other useful metrics:

- `explained_variance_score` : Compares the fit to the variance of data points; best possible score is 1.0, lower values are worse
- `mean_absolute_error` : Returns mean absolute error

All `sklearn` metrics can be imported from `sklearn.metrics` , the meaning of other metrics will be explained in the next course.

In [51]:
```
X_temp_hum = bike_sharing_15[["temp", "humidity"]].values.reshape(-1,2)
```

In [52]:
```
'''This is an example of multi-variable fit, in this case 2 variables: temp and hum
'''scikit-learn needs the data organized as numpy vectors'''

Y = bike_sharing_15["count"]
X_temp_hum = bike_sharing_15[["temp", "humidity"]]
model_two_variables = sklearn.linear_model.LinearRegression().fit(X_temp_hum, Y)
```

```
print(model_two_variables.coef_)
print(model_two_variables.intercept_)
```

```
[ 6.69169846 -1.26172547]
159.1625510018755
```

In [51]:
```
'''Note that the intercept and temp coefficient changed when another variable was a

lin_predicted_two = model_two_variables.predict(X_temp_hum)

lin_two_mse= mean_squared_error(Y, lin_predicted_two)
lin_two_rmse = np.sqrt(lin_two_mse)
print(lin_two_rmse)
```

```
131.11213598724154
```

**EXERCISE 1**. Build a regression prediction model for the Tobacco-Alcohol dataset from Connor Johnson blog using Scikit-learn Linear Regression. **NOTE:** The dataset was prepared in Linear regression section of this module. Evaluate the results by RMSE metrics.

In [ ]:
```
# YOUR CODE HERE.
```

**EXERCISE 2**. Use Scikit-learn Linear Regression to build prediction models for bike 'count' from the bike_sharing_15 dataset with different combinations of predictors available in the dataset ('temp', humidity', 'windspeed', 'registered', 'workingday'). Again, evaluate results by computing RMSE.

In [53]:
```
# YOUR CODE HERE.
```

## K-Nearest Neighbors (kNN)

The **k-Nearest Neighbors**, or k-NN, algorithm is one of the most widely used classification techniques. It is a very simple yet powerful algorithm. It is part of the supervised machine learning family of algorithms.

The KNN algorithm analyses the entire dataset and determines the class of a new data point based on similarity measures (e.g. distance function). Classification is done by a majority vote of its neighbors. A new data point is assigned to the class which has the most nearest neighbors around this new data point. `k` is the number of neighbors that the algorithm considers in the calculations. Usually, `k` is a small positive integer. If `k = 1`, then the object is simply assigned to the class of that single nearest neighbor.

Here is a simple example:

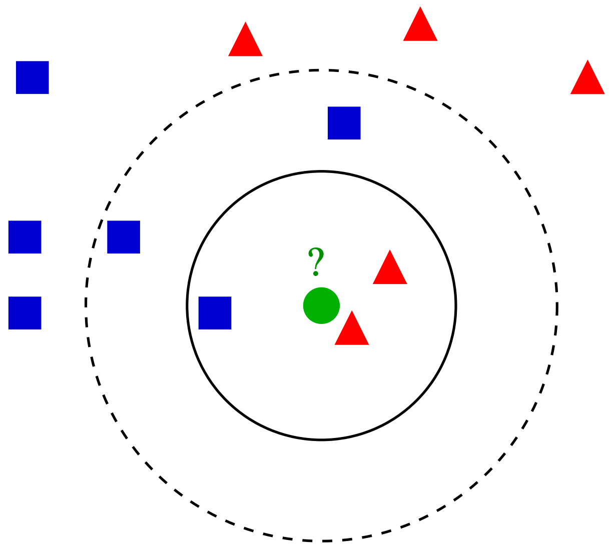


**Image source:** Wikimedia, Antti Ajanki AnAj - Own work, CC BY-SA 3.0

In this example, we are trying to determine the class of a green dot. Do we need to classify it as a blue square or a red triangle? If we use the kNN algorithm with `k = 3`, we will find 3 existing objects close to the green dot as indicated by the solid line circle. There are 2 red triangles and only 1 blue square inside the solid line circle. Hence, the green dot will be classified as a **red triangle**. If `k = 5` (dashed line circle), the green dot becomes a **blue square** as there are 3 squares vs. 2 triangles inside the dashed circle.

The kNN algorithm can also be used to solve **regression** problems. In this case, the prediction is based on the mean or the median of the `k`-most similar instances.

The kNN algorithm is easy to implement from scratch. This article, Tutorial To Implement k-Nearest Neighbors in Python From Scratch, is a good example of a kNN implementation. **NOTE:** This implementation is for Python 2.7.

However, as you can expect, you don't need to write a kNN implementation from scratch as the `scikit-learn` library contains an implementation of the nearest neighbors classifier, `KNeighborsClassifier`, which you will use in one of the assignments for this course.

**End of Module**

You have reached the end of this module.

If you have any questions, please reach out to your peers using the discussion boards. If you and your peers are unable to come to a suitable conclusion, do not hesitate to reach out to your instructor on the designated discussion board.

When you are comfortable with the content, and have practiced to your satisfaction, you may proceed to any related assignments, and to the next module.

## References

Johnson, C. (2014). blog post Feb 18, 2014 - *Linear Regression with Python* (http://connor-johnson.com/2014/02/18/linear-regression-with-python/).

MNIST database, n.d. Wikipedia, accessed Sept 7, 2018. (https://en.wikipedia.org/wiki/MNIST_database).

Ng, A. (2018) *Machine Learning Yearning* (electronic book in progress; draft copies available http://www.mlyearning.org/).

Unsupervised Learning. n.d. Wikipedia, accessed Sept 7, 2018. (https://en.wikipedia.org/wiki/Unsupervised_learning#Approaches).

Witten, I.H, Frank, E. (2005) *Data Mining. Practical Machine Learning Tools and Techniques* (2nd edition). Elsevier.

```
In [ ]:
```