# Machine Learning –Linear Regression and House Prices

In this practical, you will use **Pandas** and get to know **Scikit-learn**.  
You will also learn about **linear regression** with NE (**Normal Equation**) and SGD (Stochastic **Gradient Descent**).

## Seaborn[¶](#Seaborn)

Link: <https://seaborn.pydata.org/index.html>  
Seaborn is a Python data visualization library based on matplotlib.  
It provides a high-level interface for drawing attractive and informative statistical graphics.

## Scikit-learn[¶](#Scikit-learn)

Link: <https://scikit-learn.org/stable/>

Simple and efficient tools for predictive data analysis.  
Built on NumPy, SciPy, and Matplotlib.  
Open source, commercially usable - BSD license.  
With this package we can do machine learning tasks such as:

### Preprocessing[¶](#Preprocessing)

Link: <https://scikit-learn.org/stable/modules/preprocessing.html#preprocessing>

The Preprocessing sub-package contains methods to do data preparation.  
Example for an applications in this sub-package is transforming textual data for use in machine learning algorithms.

### Dimensionality Reduction[¶](#Dimensionality-Reduction)

Link: <https://scikit-learn.org/stable/modules/decomposition.html#decompositions>

The Dimensionality Reduction sub-package contains methods that reduce the number of random variables for ML.  
Application examples of this sub-package are visualization and increasing efficiency.

### Regression[¶](#Regression)

Link: <https://scikit-learn.org/stable/modules/linear_model.html#ordinary-least-squares>

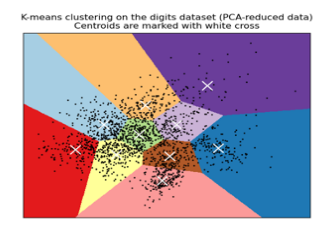
The Regression sub-package contains methods for predicting a continuous-valued attribute associated with an object.  
Application examples of this sub-package are predicting drug response and stock prices.

### Classification[¶](#Classification)

Link: <https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression>

The Classification sub-package, contains methods that identify the category an object belongs to.  
Applications examples of this sub-package are spam detection and image recognition.

### Clustering[¶](#Clustering)

  
Link: <https://scikit-learn.org/stable/modules/clustering.html#clustering>

The Clustering sub-package contains methods for automatic grouping of similar objects into sets.  
Application examples of this sub-package are customer segmentation and grouping experiment outcomes.

### Model Selection[¶](#Model-Selection)

Link: <https://scikit-learn.org/stable/model_selection.html#model-selection>

In the Model Selection sub-package, there are methods to do comparing, validating, and choosing parameters and models.  
An application example of this sub-package is improving accuracy via parameter tuning.

Let’s get going!

Data Investigation and Preprocessing

## As always, start with Imports and Definitions, let’s set them together:

In [1]:

# import numpy, matplotlib, etc.

import math

import numpy as np

import matplotlib.pyplot as plt

# define plt settings

plt.rcParams["font.size"] = 20

plt.rcParams["axes.labelsize"] = 20

plt.rcParams["xtick.labelsize"] = 20

plt.rcParams["ytick.labelsize"] = 20

plt.rcParams["legend.fontsize"] = 20

plt.rcParams["figure.figsize"] = (20,10)

We shall use the Boston House Prices Dataset for this regression task.

Start exploring the data simply by printing the keys of this dataset.

You should get a result of dictionary with five keys:

['data', 'target', 'feature\_names', 'DESCR', 'filename']

where,

1. DESCR - the description of the dataset.
2. filename - the location of the CSV file that contains the dataset.
3. data - the NumPy array with the data (the samples we want to learn from).
4. target - the list with the labels (the values we want to predict).
5. feature\_names - the names of the features (the values of each sample).

Use print to examine each key:

Printing the data should inform you that of its shape: a matrix of (506, 13), with 506 samples/datapoints – which are for this set the [suburbs](https://en.wikipedia.org/wiki/Greater_Boston#Largest_cities_and_towns) of [Boston](https://en.wikipedia.org/wiki/Neighborhoods_in_Boston#List_of_places_and_squares_within_neighborhood_areas) and 13 features for each suburb.  
Printing the labels should give you a list of 506 numeric values that represent the median value of homes in each [suburb](https://en.wikipedia.org/wiki/Suburb).  
You can display the dataset with both data and targets in a convenient table form by using Pandas DataFrame (table).   
You can read the data directly from the CSV file or from the loaded data and target.

The Table should look like this:

|  | **CRIM** | **ZN** | **INDUS** | **CHAS** | **NOX** | **RM** | **AGE** | **DIS** | **RAD** | **TAX** | **PTRATIO** | **B** | **LSTAT** | **MEDV** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0.00632 | 18.0 | 2.31 | 0.0 | 0.538 | 6.575 | 65.2 | 4.0900 | 1.0 | 296.0 | 15.3 | 396.90 | 4.98 | 24.0 |
| **1** | 0.02731 | 0.0 | 7.07 | 0.0 | 0.469 | 6.421 | 78.9 | 4.9671 | 2.0 | 242.0 | 17.8 | 396.90 | 9.14 | 21.6 |
| **2** | 0.02729 | 0.0 | 7.07 | 0.0 | 0.469 | 7.185 | 61.1 | 4.9671 | 2.0 | 242.0 | 17.8 | 392.83 | 4.03 | 34.7 |
| **3** | 0.03237 | 0.0 | 2.18 | 0.0 | 0.458 | 6.998 | 45.8 | 6.0622 | 3.0 | 222.0 | 18.7 | 394.63 | 2.94 | 33.4 |
| **4** | 0.06905 | 0.0 | 2.18 | 0.0 | 0.458 | 7.147 | 54.2 | 6.0622 | 3.0 | 222.0 | 18.7 | 396.90 | 5.33 | 36.2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **501** | 0.06263 | 0.0 | 11.93 | 0.0 | 0.573 | 6.593 | 69.1 | 2.4786 | 1.0 | 273.0 | 21.0 | 391.99 | 9.67 | 22.4 |
| **502** | 0.04527 | 0.0 | 11.93 | 0.0 | 0.573 | 6.120 | 76.7 | 2.2875 | 1.0 | 273.0 | 21.0 | 396.90 | 9.08 | 20.6 |
| **503** | 0.06076 | 0.0 | 11.93 | 0.0 | 0.573 | 6.976 | 91.0 | 2.1675 | 1.0 | 273.0 | 21.0 | 396.90 | 5.64 | 23.9 |
| **504** | 0.10959 | 0.0 | 11.93 | 0.0 | 0.573 | 6.794 | 89.3 | 2.3889 | 1.0 | 273.0 | 21.0 | 393.45 | 6.48 | 22.0 |
| **505** | 0.04741 | 0.0 | 11.93 | 0.0 | 0.573 | 6.030 | 80.8 | 2.5050 | 1.0 | 273.0 | 21.0 | 396.90 | 7.88 | 11.9 |

With 506 rows × 14 columns

Data preparation always includes checking if there are missing values in the data.  
If there are, you will need to decide what to do with them: insert some constant value in place of the missing values, Or put there a median or a mean of the feature, Or delete every row with a missing value (not recommended. on small datasets, every sample counts).  
You can read about these ways and a few more in lecture 6 or in : [7 Ways to Handle Missing Values in Machine Learning](https://towardsdatascience.com/7-ways-to-handle-missing-values-in-machine-learning-1a6326adf79e).

Are there missing values in this dataset?

Now explore the data: how do the targets depend on each feature? and how can you detect the most meaningful features in this dataset?  
To answer these questions - start with Visuals: make a scatter plot (df.plot.scatter) for each feature values vs. their target values (MEDV), to observe their relations.

To have a “full picture” plot all the features vs. the target values in one graph containing the sub-plots (plt.subplots) for each feature. Make the graph pretty by using different colors.

Using Seaborn, you can now plot both the regression line and histograms together ([jointplot](https://seaborn.pydata.org/generated/seaborn.jointplot.html" \l "seaborn-jointplot)).

You’ll find that the joint plot is hard to implement in a subplot, so create another subplots graph with only the scatter plot with the regression line ([regplot](https://seaborn.pydata.org/generated/seaborn.regplot.html" \l "seaborn-regplot)).  
Use a different [collor pallet](https://seaborn.pydata.org/generated/seaborn.color_palette.html), for better visualization of the regression line.

Which are the features that have the best correlation with the target values?

Instead of many subplots, you can use Seaborn’s heatmap (heatmap) that show in a color-coded matrix all the correlations between all the features.

Let's start with machine learning!

First, you need some preparation for the data to fit the structure the machine learning expects.

Start with separate the target values from the input features, so that the input data table should now have only 13 columns and the target are in a separate vector.

Two more preparation steps:

1. Split the data to two sets: a train set and a test set with Scikit-learn [train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html).
2. Import model\_selection from sklearn

Implement the Regression leaning:

There are two options for learning the target values from the data features (which mathematicians call “solving the regression problem”).

1. Use Normal Equation (NE)
2. Use Gradient Descent (GD)

In Scikit-learn, the [LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) model uses NE.  
There is no implementation for regular GD regressor, only [SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html).  
See [GD vs. SGD](https://datascience.stackexchange.com/a/36451) for more details.

Try fitting (fit) linear regression (linear\_model, LinearReression) with NE on the training set of input matrix and target vector.

In [18]:

# import linear\_model and train with NE

from sklearn import linear\_model

NE\_reg = linear\_model.LinearRegression().fit(X\_train, t\_train)

To assess how good your model performed on the train and test sets, use a new metric, called [R2 score](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression.score). Here are its equations, in comparison to the other metrics we discussed in class:

The R2 score is based on the MSE loss function.  
The R2 values are in range (-infinity, 1].  
The highest the score, the better the model.  
It determines how well the regression predictions approximate the real data points (See more in [Importance of Model Evaluation](https://www.datacourses.com/evaluation-of-regression-models-in-scikit-learn-846/)).

Calculate the R2 scores for the train set and on the test set (NE\_reg.score).

To calculate the MSE or RMSE, we can use Scikit-learn [mean\_squared\_error](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html).

Now try a regression model with SGD (linear\_model.SGDRegressor, SGD\_reg.predict).

You’ll see that the model performance is far from good... The reason is that the features are not standardized and normalized:  
To demonstrate the problem look at the picture below:  
  
Our case is case a. The model is having a hard time finding the minimum.  
We need to standardize the features to have case b, so it will be easier for the model to find the minimum.  
You can do it with Scikit-learn [StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html).

The model should now perform better, closer to the performance of NE.

And lastly: are all the features really necessary? You found the two best features from the Visuals of the data : Try to do the prediction using only these best two features.

You should see that it is worse than using all the features. It means that some other features are helping the prediction.

You will learn more about feature selection in future practices.

## More Information[¶](#More-Information)

Scikit-learn toy datasets.  
[Toy Datasets](https://scikit-learn.org/stable/datasets/index.html?highlight=boston%20housing%20price#toy-datasets)

The difference between isna and isnull pandas.DataFrame methods  
[Difference between isna() and isnull() in pandas](https://datascience.stackexchange.com/a/37879)

Documentation of plt.rcParams:  
[A sample matplotlibrc file](https://matplotlib.org/3.3.2/tutorials/introductory/customizing.html#a-sample-matplotlibrc-file)

Documentation of matplotlib.pyplot.axes:  
[matplotlib.pyplot.axes](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.axes.html#matplotlib-pyplot-axes)

Guide for multi-output regression models:  
[How to Develop Multi-Output Regression Models with Python](https://machinelearningmastery.com/multi-output-regression-models-with-python/)

How Scikit-learn implements GD and NE:  
[Linear Regression and Gradient Descent in Scikit learn/Pandas?](https://stackoverflow.com/a/34470001)

The differences between normalization and standardization:  
[Normalization vs Standardization](https://towardsdatascience.com/normalization-vs-standardization-cb8fe15082eb)

Explanation on Seaborn color palettes:  
[Choosing color palettes](https://seaborn.pydata.org/tutorial/color_palettes.html)

How to create a custom score in Scikit-learn:  
[Custom Loss vs Custom Scoring](https://kiwidamien.github.io/custom-loss-vs-custom-scoring.html)

How to create a custom loss in Scikit-learn:  
[Fitting Linear Models with Custom Loss Functions and Regularization in Python](https://alex.miller.im/posts/linear-model-custom-loss-function-regularization-python/)

How to use MAE as a loss function in Scikit-learn SGDRegressor:  
[Training Linear Models with MAE using sklearn in Python](https://stackoverflow.com/a/50394085)

Guide for using Seaborn:  
[An introduction to seaborn](https://seaborn.pydata.org/introduction.html)