**Housing Prices**

* This program allows you to predict housing prices using Python. We will be using the California housing dataset for this problem. This dataset is provided by the sklearn module, hence accessing the dataset will employ means different from what we normally use for pandas.
* We will be using preprocessing package that gives us the scale function (as the name suggests, we will use it to scale the dataset).
* We use polynomial features in this problem because linear models with only single degree might lead to a bad fit. Hence we use multiple degrees. We use degree of value 2 in the program to make the model more accurate.
* We follow the standard procedure of splitting the dataset into a training set and a test set. This step is important when we employ the use of linear regression. This operation can be done using train\_test\_split.
* x\_train and y\_train are used to store the training examples. Test\_size can be used to denote what percentage of the set we need.
* After that, we can fit our regression model to the given data. model.fit(X\_train, y\_train) is used to fit the data to the linear regression model.
* We can now use our test set to test the model after having trained it on the previous set of values.
* predictionTestSet = model.predict(X\_test) is the instruction used wherein predictionTestSet is used to store the prediction value. model.predict() takes x\_test as a parameter in order to predict the result.
* We can also analyse the percentage of error by using mean\_squared\_error()

**Documentation of various modules used in this problem :**

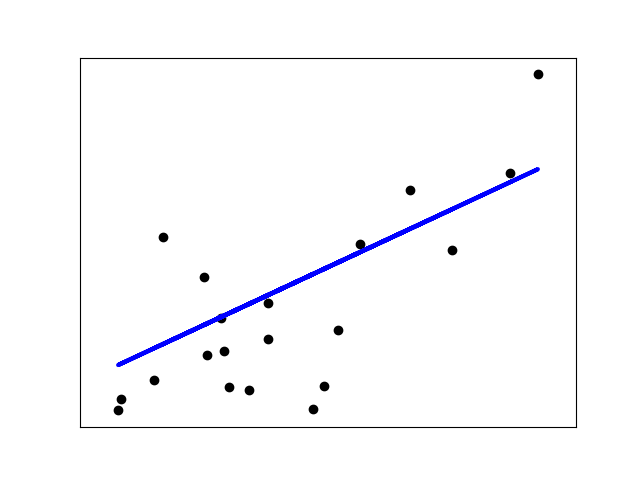
**Matplotlib :**

* Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and [IPython](http://ipython.org/) shells, the [Jupyter](http://jupyter.org/) notebook, web application servers, and four graphical user interface toolkits.
* Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the [sample plots](https://matplotlib.org/tutorials/introductory/sample_plots.html) and [thumbnail gallery](https://matplotlib.org/gallery/index.html).
* For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

**Sklearn :**

* The main module that we employ in this problem is sklearn. The documentation and explanation for this module has been taken from the official documentation website. These are some important excerpts :
  + Scikit-learn is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines. (Taken from Wikipedia).
  + [**LinearRegression**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) fits a linear model with coefficients w=(w1,...,wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

min||Xw-y||

[](https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html)

* [**LinearRegression**](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression) will take in its fit method arrays X, y and will store the coefficients w of the linear model in its coef\_member:

>>>

**>>> from** **sklearn** **import** linear\_model

**>>>** reg = linear\_model.LinearRegression()

**>>>** reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])

**...**

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=None,

normalize=False)

**>>>** reg.coef\_

array([0.5, 0.5])

The coefficient estimates for Ordinary Least Squares rely on the independence of the features. When features are correlated and the columns of the design matrix X have an approximate linear dependence, the design matrix becomes close to singular and as a result, the least-squares estimate becomes highly sensitive to random errors in the observed target, producing a large variance. This situation of *multicollinearity* can arise, for example, when data are collected without an experimental design.