# Regression Analysis_2025

August 4, 2025

## 1 Regression Analysis

There are several Python libraries which provide solid implementations of a range of machine learning algorithms. One of the best known is Scikit-Learn, a package that provides efficient versions of a large number of common algorithms. Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as by very useful and complete online documentation. A benefit of this uniformity is that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.

### 1.1 Data Description

The data in `WestRoxbury.csv` includes information on single family owner-occupied homes in WestRoxbury, a neighborhood in southwest Boston, MA in 2014. This dataset has 12 variables, and 5802 homes. The descriptions of variables are as follows:

- **TOTAL VALUE**: Total assessed value for property, in thousands of USD
- **TAX**: Tax bill amount based on total assessed value multiplied by the tax rate, in USD
- **LOT SQFT**: Total lot size of parcel in square feet
- **GROSS AREA**: Gross floor area
- **LIVING AREA**: Total living area for residential properties
- **FLOORS**: Number of floors
- **ROOMS**: Number of rooms
- **BEDROOMS**: Number of bedrooms
- **FULL BATH**: Number of full baths
- **HALF BATH**: Number of half baths
- **KITCHEN**: Number of kitchens
- **FIREPLACE**: Number of fireplaces

[1]:
```
import pandas as pd
house = pd.read_csv("WestRoxbury.csv")
house
```

[1]:

| | TOTAL VALUE | TAX | LOT SQFT | GROSS AREA | LIVING AREA | FLOORS | ROOMS | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 344.2 | 4330 | 9965 | 2436 | 1352 | 2.0 | 6 | |
| 1 | 412.6 | 5190 | 6590 | 3108 | 1976 | 2.0 | 10 | |
| 2 | 330.1 | 4152 | 7500 | 2294 | 1371 | 2.0 | 8 | |
| 3 | 498.6 | 6272 | 13773 | 5032 | 2608 | 1.0 | 9 | |
| 4 | 331.5 | 4170 | 5000 | 2370 | 1438 | 2.0 | 7 | |
| … | … | … | … | … | … | … | … | |

```
5797        404.8  5092      6762        2594         1714     2.0      9
5798        407.9  5131      9408        2414         1333     2.0      6
5799        406.5  5113      7198        2480         1674     2.0      7
5800        308.7  3883      6890        2000         1000     1.0      5
5801        447.6  5630      7406        2510         1600     2.0      7

      BEDROOMS  FULL BATH  HALF BATH  KITCHEN  FIREPLACE
0            3          1          1        1          0
1            4          2          1        1          0
2            4          1          1        1          0
3            5          1          1        1          1
4            3          2          0        1          0
…        …          …          …        …          …
5797         3          2          1        1          1
5798         3          1          1        1          1
5799         3          1          1        1          1
5800         2          1          0        1          0
5801         3          1          1        1          1

[5802 rows x 12 columns]
```

## 1.2 Data Representation for Data Mining

Machine learning is about creating models from data: for that reason, we'll start by discussing how data can be represented in order to be understood by the computer.

1. **Attribute Matrix**: a two-dimensional matrix that is most often contained in a NumPy array or a Pandas DataFrame. The samples (i.e., rows) always refer to the individual objects described by the dataset. For example, the sample might be a flower, a person, a document, an applicant, a house, or anything else you can describe with a set of quantitative measurements. The attributes (i.e., columns) always refer to the distinct properties that describe each sample in a quantitative manner.
2. **Target Array**: The target array is usually one dimensional, and is generally contained in a NumPy array.

Often one point of confusion is how the target array differs from the other attributes columns. The distinguishing feature of the target array is that it is usually the quantity we want to predict from the data. For example, in the preceding data we may wish to construct a model that can predict the species of flower based on the other measurements; in this case, the species column would be considered the target array.

### 1.2.1 Choose Predictors

Based on your business understanding and exploratory data analysis, choose predictors that might have a (linear) impact on the target variable.

For example, we choose to use **GROSS AREA**, **FLOORS** and **ROOMS** as our predictors. We are looking for a linear regression model with unknown coefficients as

$$\text{TOTAL VALUE} = \beta_0 + \beta_1 \times \text{GROSS AREA} + \beta_2 \times \text{FLOORS} + \beta_3 \times \text{ROOMS}.$$

[2]:
```
# Choose predictors to construct attribute matrix
house_X = house.loc[:, ['GROSS AREA', 'FLOORS', 'ROOMS'] ]


# Target
house_y = house.iloc[:,0]
```

### 1.3 Partition the Data into Train vs. Validation/Test

To create a train and validation set, we are going to split randomly this data set into 70% training set and 30% validation/test set. The random seed is set to 42.

We will use the `sklearn.model_selection.train_test_split()` function. Detailed documentation can be found at

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

Specifically,

1. **test_size**: If float, should be between 0.0 and 1.0 and represent the proportion of the dataset to include in the validation split. If int, represents the absolute number of validation samples. If None, the value is set to the complement of the train size.
2. **random_state**: an int as the seed for reproducing the random partition.

[3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(house_X, house_y,
                                                    test_size=0.3,
                                                    random_state=42)
```

### 1.4 Fit the Model to your Training Data

We first need to import the model from Scikit-Learn and then create the linear regression model instance.

[4]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

There are many other models available in Scikit-Learn. After you import the desired model, the following pipelines and API's are consistent. It can be easily changed from LinearRegression to another model. For more information, please visit the documentation of Scikit-Learn. For example, you can also do

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
```

to use a different machine learning model called Gaussian naive Bayes.

After we initialize the linear regression model, now it is time to apply our model to data. This can be done with the `fit()` method of the model:

[7]:
```
#import numpy as np

# Train the model using the training set
model.fit(X_train, y_train)

# Print out the fitted coefficients
# np.set_printoptions(precision=4, suppress=True) # set precision for numpy␣
 ↪arrays
print(model.coef_)
print( model.intercept_ )
# print('%.4f' % model.intercept_ )  # set precision for floating points
```

```
[ 0.07365711 52.30357956  6.42323096]
43.63251892929566
```

Let's write down the fitted predictive equation:

$$\text{TOTAL VALUE} = 43.6325 + 0.0737 \times \text{GROSS AREA} + 52.3036 \times \text{FLOORS} + 6.4232 \times \text{ROOMS}$$

## 1.5 Make Predictions for Any Dataset

Once the model is trained, the main task of regression is to evaluate it based on what it says about new data that was not part of the training set. In Scikit-Learn, this can be done using the predict() method.

For performance evaluation purposes, you need to get predictions for the **test** set and compare the predicted values to their true values. You can also make predictions for the **training** set (by changing the input argument of the predict() from `X_test` to `X_train` ), or **any** "new" sample.

[9]:
```
# make predictions for validation set
y_pred = model.predict(X_test)
```

### 1.5.1 For interested readers: Construct DataFrame for any new houses

Make prediction for a new house with GROSS AREA = 2655, FLOORS = 1.5, ROOMS = 6, by constructing a DataFrame.

[7]:
```
# Example house with GROSS AREA, FLOORS, and ROOMS
example_houses = {
    'GROSS AREA': [2655],
    'FLOORS': [1.5],
    'ROOMS': [6]
}

# Convert the example house features to a DataFrame
example_house_df = pd.DataFrame(example_houses)
```

```
example_house_df
```

[7]:
```
   GROSS AREA  FLOORS  ROOMS
0        2655     1.5      6
```

`example_houses` is a **Python Dictionary** (i.e. one data structure). A dictionary is a collection of key-value pairs. Think of it like a real-world dictionary where you look up a word (key) and get its definition (value).

**Key Characteristics** + Keys: Unique identifiers (like words in a dictionary). + Values: Information associated with keys (like definitions).

**Example of a dictionary:** `student_grades = { "Alice": 90, "Bob": 85, "Charlie": 98 }` + "Alice", "Bob", "Charlie" are keys. + 90, 85, 98 are their corresponding values.

**Accessing Values** Use keys to get values: `print(student_grades["Alice"])`

[8]:
```
student_grades = { "Alice": 90, "Bob": 85, "Charlie": 98 }
print( student_grades["Alice"] )
```

```
90
```

[9]:
```
y_new_pred = model.predict( example_house_df )
print(y_new_pred)
```

```
[356.18689946]
```

### 1.6 Performance Evaluation

Recall the error or residual is defined as $e_i = Y_i - \hat{Y}_i$, and difference accuracy measures are:

$$ME = \frac{1}{n}\sum_{i=1}^{n} e_i$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |e_i|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

Let's compute the mean absolute error of the **test** set.

[10]:
```
e = y_test - y_pred
# compute MAE
mae = np.mean( np.abs(e))
print( mae)
```

```
40.61120379680034
```

We can also compute the mean absolute error of the **training** set.

[11]:
```
# make predictions on training set
y_pred_train = model.predict(X_train)

e = y_train - y_pred_train
# compute MAE
mae_train = np.mean( np.abs(e) )
print(mae_train)
```

```
39.88844892000601
```