Preparation of Feature Matrix and Target Vector

Slicing of Pandas DataFrame

- Pandas DataFrame can be Sliced using the following methods

- *loc method*

- *iloc method*

pandas.DataFrame.loc

property `DataFrame.loc`

Access a group of rows and columns by label(s) or a boolean array.

`.loc[]` is primarily label based, but may also be used with a boolean array.

Allowed inputs are:

- A single label, e.g. `5` or `'a'`, (note that `5` is interpreted as a *label* of the index, and **never** as an integer position along the index).
- A list or array of labels, e.g. `['a', 'b', 'c']`.
- A slice object with labels, e.g. `'a':'f'`.

> **Warning**
>
> Note that contrary to usual python slices, **both** the start and the stop are included

- A boolean array of the same length as the axis being sliced, e.g. `[True, False, True]`.
- A `callable` function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above)

https://pandas.pydata.org

pandas.DataFrame.iloc

property `DataFrame.iloc`

Purely integer-location based indexing for selection by position.

`.iloc[]` is primarily integer position based (from `0` to `length-1` of the axis), but may also be used with a boolean array.

Allowed inputs are:

- An integer, e.g. `5`.
- A list or array of integers, e.g. `[4, 3, 0]`.
- A slice object with ints, e.g. `1:7`.
- A boolean array.
- A `callable` function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above). This is useful in method chains, when you don't have a reference to the calling object, but would like to base your selection on some value.

`.iloc` will raise `IndexError` if a requested indexer is out-of-bounds, except *slice* indexers which allow out-of-bounds indexing (this conforms with python/numpy *slice* semantics).

https://pandas.pydata.org

Feature Matrix

- Two-dimensional numerical array or matrix
- By convention, this features matrix is often stored in a variable named X.
- The features matrix is assumed to be two-dimensional, with shape [n_samples, n_features], and is most often contained in a NumPy array or a Pandas DataFrame
- The features (i.e., columns) always refer to the distinct observations that describe each sample in a quantitative manner.
- Features are generally real-valued, but may be Boolean or discrete-valued in some cases.

Target Vector

- Label or target array, which by convention we will usually call y.
- The target array is usually one dimensional, with length n_samples, and is generally contained in a NumPy array or Pandas Series.
- The target array may have continuous numerical values, or discrete classes/labels.

Note

- While some Scikit-Learn estimators do handle multiple target values in the form of a two-dimensional, [n_samples, n_targets] target array

Feature Matrix vs Target Vector

The distinguishing feature of the target array is that,

It is usually the quantity we want to predict from the data: in statistical terms, it is the dependent variable.

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.

Preparing Feature Matrix

```
1  X=iris.iloc[:,0:-1]
2  X.head()
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Preparing Target Vector

Since the Species name is in Textual format, we can use the Label Enconder to convert it to numeric values

```
1 from sklearn import preprocessing
2 le = preprocessing.LabelEncoder()
```

```
1 y=le.fit_transform(iris["species"])
2 print(y)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```