We will be using the **Scikit Learn** package.

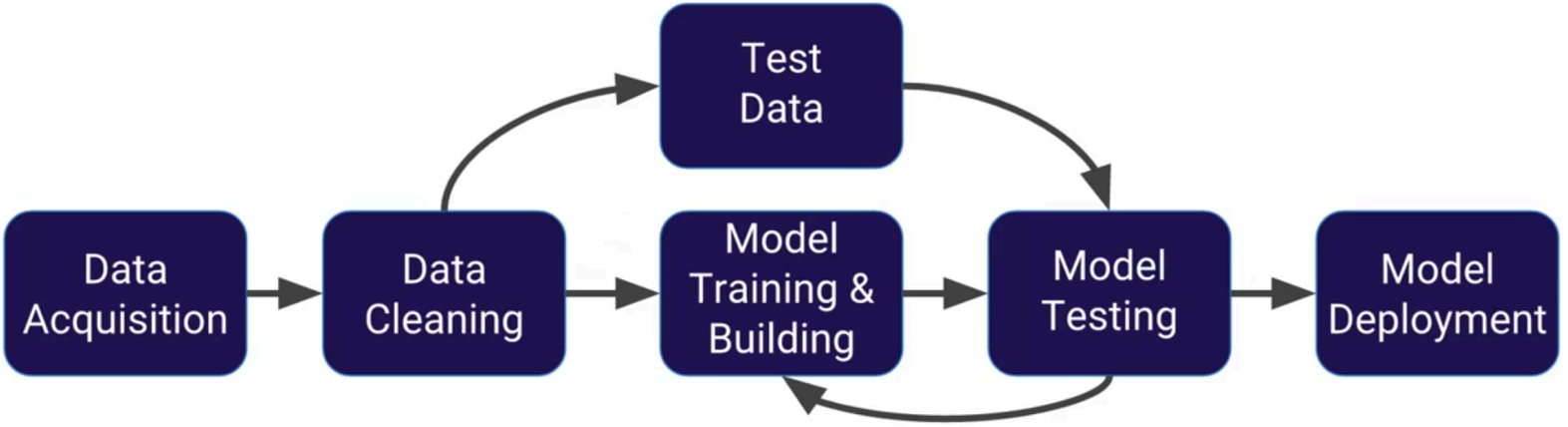
It's the most popular machine learning package for Python and has a lot of algorithms built-in!

You'll need to install it using:

conda install scikit-learn

or

pip install scikit-learn


Test Data
Data Acquisition
Data Cleaning
Model Training & Building
Model Testing
Model Deployment

- Now let's go over an example of the process to use SciKit Learn.
- Don't worry about memorizing any of this, we'll get plenty of practice and review when we actually start coding in subsequent lectures!

Every algorithm is exposed in scikit-learn via an "Estimator"

First you'll import the model, the general form is:

```
from sklearn.family import Model
```

For example:

```
from sklearn.linear_model import LinearRegression
```

Estimator parameters: All the parameters of an estimator can be set when it is instantiated, and have suitable default values.

You can use Shift+tab in jupyter to check the possible parameters.

For example:

```
model = LinearRegression(normalize=True)
print(model)
```

```
LinearRegression(copy_X=True, fit_intercept=True,
normalize=True)
```

Once you have your model created with your parameters, it is time to fit your model on some data!

But remember, we should split this data into a training set and a test set.

```
>>> import numpy as np
>>> from sklearn.cross_validation import train_test_split
>>> X, y = np.arange(10).reshape((5, 2)), range(5)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> list(y)
[0, 1, 2, 3, 4]
```

```
>>> X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3)
>>> X_train
array([[4, 5],
       [0, 1],
       [6, 7]])
>>> y_train
[2, 0, 3]
>>> X_test
array([[2, 3],
       [8, 9]])
>>> y_test
[1, 4]
```

Now that we have split the data, we can train/fit our model on the training data.

This is done through the model.fit() method:

```
model.fit(X_train,y_train)
```

- Now the model has been fit and trained on the training data.

- The model is ready to predict labels or values on the test set!

We get predicted values using the predict method:

```
predictions = model.predict(X_test)
```

We can then evaluate our model by comparing our predictions to the correct values.

The evaluation method depends on what sort of machine learning algorithm we are using (e.g. Regression,Classification, Clustering, etc.)

Scikit-learn strives to have a uniform interface across all methods, and we'll see examples of these below.

Given a scikit-learn *estimator* object named model, the following methods are available...

- Available in **all Estimators**
    - model.fit() : fit training data.
    - For supervised learning applications, this accepts two arguments: the data X and the labels y (e.g. model.fit(X, y)).
    - For unsupervised learning applications, this accepts only a single argument, the data X (e.g. model.fit(X)).

Available in **supervised estimators**

- model.predict() : given a trained model, predict the label of a new set of data. This method accepts one argument, the new data X_new (e.g. model.predict(X_new)), and returns the learned label for each object in the array.

Available in **supervised estimators**

- model.predict_proba() : For classification problems, some estimators also provide this method, which returns the probability that a new observation has each categorical label. In this case, the label with the highest probability is returned by model.predict().

Available in **supervised estimators**

- model.score() : for classification or regression problems, most estimators implement a score method. Scores are between 0 and 1, with a larger score indicating a better fit.

Available in **unsupervised estimators**

- model.predict() : predict labels in clustering algorithms.

Available in **unsupervised estimators**

- model.transform() : given an unsupervised model, transform new data into the new basis. This also accepts one argument X_new, and returns the new representation of the data based on the unsupervised model.

Available in **unsupervised estimators**

- model.fit_transform() : some estimators implement this method, which more efficiently performs a fit and a transform on the same input data.