Refernce : Introduction to Machine learning with python / Andreas C.Muller

프래그래머는 일련의 데이터를 다른 데이터로 변환하는 절차를 꾸미는 사람. 당연히 그들의 도구 상자는 알고리즘으로 채워집니다. 그 알고리즘이 고도의 수학적 분석으로부터 도출되는 방정식일수도 있고, 즐겨 쓰는 언어의 표준 라이브러리에 포함된 함수 호출 규격일 수도 있습니다.

특정 부류의 알고리즘이 발달하면 장점과 한계가 명확해지고, 널리 쓰일 수 있는 것들은 필요할 때 큰 고민 없이 사용할 수 있도록 라이브러리로 만들어져 유통됩니다.

이 책은 수학보다 머신러닝 알고리즘을 실용적으로 사용하는데 초점이 맞춰 있습니다.

머신러닝 알고리즘의 수학 이론에 관심이 있다면, Trevor Hastie, Robert Tibshirani, Jerome Friedmann의 "The Element of Statistical learning"을 추천합니다.

URL : https://github.com/amueller/introduction_to_ml_with_python

저자 언급 : 원본 데이터에서 관심 있는 정보를 추출하는 데 필요한 모든 것이 해당한다고 봄.

This book will not cover the mathematical details of machine learning algorithms,

Chapter1. Introduction

Machine learning is about extracting knowledge from data. It is a research field at the intersection of statistics, artificial intelligence and computer science, which is also known as predictive analytics or statistical learning.

1. Why machine learning?

- In the early days of “intelligent” applications, many systems used hand-coded rules of “if ” and “else” decisions to process data or adjust to user input.
- Think of a spam filter whose job is to move an email to a spam folder.
- This would be an example of using an expert designed rule system to design an “intelligent” application.
- Designing kind of manual design of decision rules is feasible for some applications, in particular for those applications in which humans have a good understanding of how a decision should be made. However, using hand-coded rules to make decisions has two major disadvantages:
    - 1. The logic required to make a decision is specific to a single domain and task. Changing the task even slightly might require a rewrite of the whole system.
    - 1. Designing rules requires a deep understanding of how a decision should be madeby a human expert

1.1.1 Problems that machine learning can solve

- The most successful kind of machine learning algorithms are those that automate a decision making processes by generalizing from known examples.
- In this setting,which is known as a supervised learning setting, the user provides the algorithm with pairs of inputs and desired outputs, and the algorithm finds a way to produce the desired output given an input.
- Machine learning algorithms that learn from input-output pairs are called supervised learning algorithms because a “teacher” provides supervision to the algorithm in the form of the desired outputs for each example that they learn from.

Examples of supervised machine learning tasks include:

- Identifying the ZIP code from handwritten digits on an envelope. Here the input is a scan of the handwriting, and the desired output is the actual digits in the zip code. To create a data set for building a machine learning model, you need to collect many envelopes. Then you can read the zip codes yourself and store the digits as your desired outcomes.

- Determining whether or not a tumor is benign based on a medical image. Here the input is the image, and the output is whether or not the tumor is benign. To create a data set for building a model, you need a database of medical images. You also need an expert opinion, so a doctor needs to look at all of the images and decide which tumors are benign and which

also need an expert opinion, so a doctor needs to look at all of the images and decide which tumors are benign and which are not.

- Detecting fraudulent activity in credit card transactions. Here the input is a record of the credit card transaction, and the output is whether it is likely to be fraudulent or not. Assuming that you are the entity distributing the credit cards, collecting a dataset means storing all transactions, and recording if a user reports any transaction as fraudulent.

  ## The other type of algorithms that we will cover in this book is unsupervised algo-rithms. In unsupervised learning, only the input data is known and there is no known

- output data given to the algorithm. While there are many successful applications of these methods as well, they are usually harder to understand and evaluate.

  ### Examples of unsupervised learning include:

- Identifying topics in a set of blog posts. If you have a large collection of text data, you might want to summarize it and find prevalent themes in it. You might not know beforehand what these topics are, or how many topics there might be. Therefore, there are no known outputs.

- Segmenting customers into groups with similar preferences. Given a set of cus-tomer records, you might want to identify which customers are similar, and whether there are groups of customers with similar preferences. For a shopping site these might be “parents”, “bookworms” or “gamers”. Since you don’t know in advanced what these groups might be, or even how many there are, you have no known outputs.

- Detecting abnormal access patterns to a website. To identify abuse or bugs, it is often helpful to find access patterns that are different from the norm. Each abnormal pattern might be very different, and you might not have any recorded instances of abnormal behavior. Since in this example you only observe traffic, and you don’t know what constitutes normal and abnormal behavior, this is an unsupervised problem.

- For both supervised and unsupervised learning tasks, it is important to have a repre-sentation of your input data that a computer can understand.

- Each entity or row here is known as data point or " sample " in machine learning, while the columns, the properties that describe these entities, are called " features ".

- We will later go into more detail on the topic of building a good representation of your data, which is called feature extraction or feature engineering.

1.1.2 Knowing your data

- Quite possibly the most important part in the machine learning process is under-standing the data you are working with.
- Before you start building a model, it is important to know the answers to most of, if not all of, the following questions:

- How much data do I have? Do I need more?

- How many features do I have? Do I have too many? Do I have too few?
- Is there missing data? Should I discard the rows with missing data or handle them differently?
- What question(s) am I trying to answer? Do I think the data collected can answer that question?

The last bullet point is the most important question, and certainly is not easy to answer. Thinking about these questions will help drive your analysis.

1.2 Why python ?

- Python has become the lingua franca for many data science applications
- It combines the powers of general purpose programming languages with the ease of use of domain specific scripting languages like matlab or R.

We will focus mostly on supervised learning techniques and algorithms, as these are often the most useful ones in practice, and they are easy for beginners to use and understand.

1.3 Scikit-learn

- The scikit-learn project is constantly being developed and improved, and has a very active user community.

  ## 1.3.1 Installing Scikit-learn
- Scikit-learn depends on two other Python packages, NumPy and SciPy.
- For plottingand interactive development, you should also install matplotlib, IPython and the Jupyter notebook.

- pip install numpy scipy matplotlib ipython scikit-learn

- pip install numpy scipy matplotlib ipython scikit-learn

1.4 Essential Libraries and Tools

- Scikit-learn is built on top of the NumPy and SciPy scientific Python libraries. In addition to knowing about NumPy and SciPy, we will be using Pandas and matplotlib.

1.4.2 NumPy

- NumPy is one of the fundamental packages for scientific computing in Python
- Scikit-learn takes in data in the form of NumPy arrays.
- The core functionality of NumPy is this “ndarray”, meaning it has n dimensions, and all elements of the array must be the same type.

In [1]:

```
import numpy as np

x = np.array([ [1,2,3] , [4,5,6] ] )


type(x)
x
```

Out[1]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

1.4.3 SciPy

- SciPy is both a collection of functions for scientific computing in python. It provides, among other functionality, advanced linear algebra routines, mathematical function optimization, signal processing, special mathematical functions and statistical distri-butions.

In [2]:

```
# Sparse matri-ces are used whenever we want to store a 2d array that contains mostly zeros:

from scipy import sparse
# create a 2d numpy array with a diagonal of ones, and zeros everywhere else
eye = np.eye(4)
print("Numpy array:\n%s" % eye)
# convert the numpy array to a scipy sparse matrix in CSR format
# only the non-zero entries are stored
sparse_matrix = sparse.csr_matrix(eye)
print("\nScipy sparse CSR matrix:\n%s" % sparse_matrix)
```

```
Numpy array:
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

Scipy sparse CSR matrix:
  (0, 0) 1.0
  (1, 1) 1.0
  (2, 2) 1.0
  (3, 3) 1.0
```

1.4.4 Matplotlib

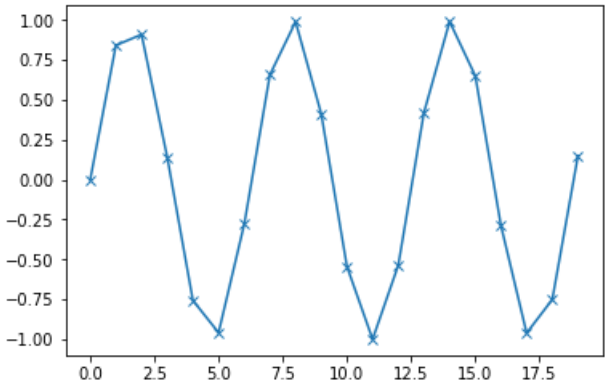
- Matplotlib is the primary scientific plotting library in Python.
  - Visualizing your data and any aspects of your analysis can give you important insights, and we will be using matplotlib for all our visualizations.

In [3]:

```
%matplotlib notebook
%matplotlib inline
import matplotlib.pyplot as plt
# Generate a sequence of integers
x = np.arange(20)
# create a second array using sinus
y = np.sin(x)
# The plot function makes a line chart of one array against another
plt.plot(x, y, marker="x")
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x1543842f9e8>]
```



1.4.5 Pandas

- Pandas is a Python library for data wrangling and analysis. It is built around a data structure called DataFrame, that is modeled after the R DataFrame.
- Simply put, a Pandas Pandas DataFrame is a table, similar to an Excel Spreadsheet.
- Another valuable tool provided by Pandas is its ability to ingest from a great variety of file formats and databases, like SQL, Excel files and comma separated value (CSV) files

In [4]:

```
# Here is a small example of creating a DataFrame using a dictionary

import pandas as pd
# create a simple dataset of people
data = {'Name': ["John", "Anna", "Peter", "Linda"],
 'Location' : ["New York", "Paris", "Berlin", "London"],
 'Age' : [24, 13, 53, 33]
 }

#data
data_pandas = pd.DataFrame(data)

data_pandas
```

Out[4]:

	Name	Location	Age
0	John	New York	24
1	Anna	Paris	13
2	Peter	Berlin	53
3	Linda	London	33

In [5]:

```
# Age > 30
display( data_pandas[data_pandas.Age > 30] )
```

	Name	Location	Age
2	Peter	Berlin	53
3	Linda	London	33

1.5 Python2 versus Python3

There are two major versions of Python that are widely used at the moment: Python2(more precisely 2.7) and Python3 (with the latest release being 3.5 at the time of writ-ing), which sometimes leads to some confusion. Python2 is no longer actively devel-oped, but because Python3 contains major changes, Python2 code does usually notrun without changes on Python3. If you are new to Python, or are starting a new project from scratch, we highly recommend using the latests version of Python3.

1.6 Versions Used in this Book

In [6]:

```
import pandas as pd
print("pandas version: %s" % pd.__version__)

import matplotlib
print("matplotlib version: %s" % matplotlib.__version__)

import numpy as np
print("numpy version: %s" % np.__version__)

import IPython
print("IPython version: %s" % IPython.__version__)

import sklearn
print("scikit-learn version: %s" % sklearn.__version__)
```

```
pandas version: 0.23.0
matplotlib version: 2.2.2
numpy version: 1.14.3
IPython version: 6.4.0
scikit-learn version: 0.19.1
```

While it is not important to match these versions exactly, you should have a version of scikit-learn that is as least as recent as the one we used.

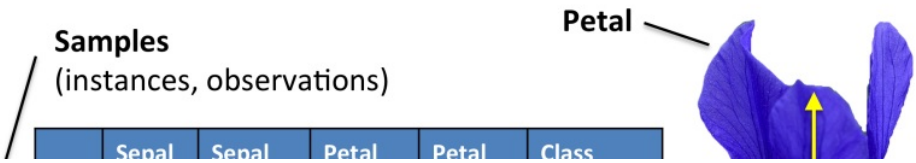
1.7 A First Application: Classifying iris species

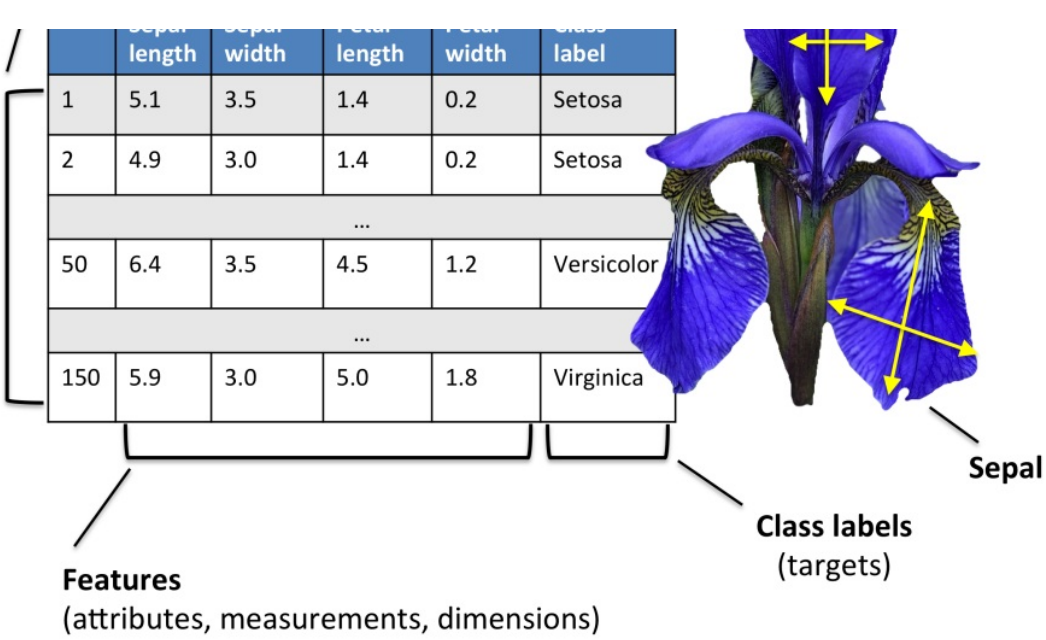
- Let's assume that a hobby botanist is interested in distinguishing what the species is of some iris flowers that she found. She has collected some measurements associated with the iris: the length and width of the petals, and the length and width of the sepal, all measured in centimeters.

In [7]:

```
from IPython.display import Image
Image(filename ='iris.png')
```

Out[7]:





She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species Setosa, Versicolor or Virginica.

For these measurements, she can be certain of which species each iris belongs to. Let's assume that these are the only species our hobby botanist will encounter in the wild.

Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.

Since we have measurements for which we know the correct species of iris, this is a supervised learning problem. In this problem, we want to predict one of several options (the species of iris). This is an example of a classification problem. The possi-ble outputs (different species of irises) are called classes.

The desired output for a single data point (an iris) is the species of this flower. For a particular data point, the species it belongs to is called its label.

1.7 Meet the data

The data we will use for this example is the iris dataset, a classical dataset in machine learning an statistics.

It is included in scikit-learn in the dataset module.

In [8]:

```
from sklearn.datasets import load_iris
iris = load_iris()

iris.keys()
```

Out[8]:

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

The value to the key DESCR is a short description of the dataset. We show the begin-ning of the description here.

In [9]:

```
print(iris['DESCR'][:193] + "\n...")
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
```

```
    :Number of Attributes: 4 numeric, predictive att
...
```

In [10]:

```
type(iris.keys())
```

Out[10]:

```
dict_keys
```

The value with key target_names is an array of strings, containing the species of flower that we want to predict:

In [11]:

```
iris['target_names']
```

Out[11]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

In [12]:

```
type('target_names')
```

Out[12]:

```
str
```

The feature_names are a list of strings, giving the description of each feature:

In [13]:

```
iris['feature_names']
```

Out[13]:

```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

The data itself is contained in the target and data fields. The data contains the numeric measurements of sepal length, sepal width, petal length, and petal width in a numpy array:

In [14]:

```
type(iris['data'])
```

Out[14]:

```
numpy.ndarray
```

The rows in the data array correspond to flowers, while the columns represent the four measurements that were taken for each flower:

In [15]:

```
iris['data'].shape
```

Out[15]:

```
(150, 4)
```

Remember that the individual items are called "samples" in machine learning, and their properties are called "features".

This is a convention in scikit-learn, and your data will always be assumed to be in this shape.

feature values for the first five samples:

In [16]:

```
iris['data'][:5]
```

Out[16]:

```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

The target array contains the species of each of the flowers that were measured, also as a numpy array:

In [17]:

```
iris['target'][:5]
```

Out[17]:

```
array([0, 0, 0, 0, 0])
```

In [18]:

```
iris['target'].shape
```

Out[18]:

```
(150,)
```

The target is a one-dimensional array, with one entry per flower:

In [19]:

```
iris['target']
```

Out[19]:

```
array([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, 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])
```

The species are encoded as integers from 0 to 2:

The meaning of the numbers are given by the iris['target_names'] array: 0 means Setosa, 1 means Versicolor and 2 means Virginica.

1.7.2 Measuring Success: Training and testing data

- Before we can apply our model to new measurements, we need to know whether our model actually works, that is whether we should trust its predictions.
  - Unfortunately, we can not use the data we use to build the model to evaluate it. This is because our model can always simply remember the whole training set, and will therefore always predict the correct label for any point in the training set. This “remembering” does not indicate to us whether our model will generalize well, in other words whether it will also perform well on new data. So before we apply our model to new measurements, we will want to know whether we can trust its predictions.
- To assess the models’ performance, we show the model new data (that it hasn’t seenbefore) for which we have labels. This is usually done by splitting the labeled data wehave collected (here our 150 flower measurements) into two parts.The part of

the data is used to build our machine learning model, and is called thetraining data or training set. The rest of the data will be used to access how well the model works and is called "test data", "test set" or "hold-out set".
- This function extracts 75% of the rows in the data as the training set, together withthe corresponding labels for this data. The remaining 25% of the data, together with the remaining labels are declared as the test set.

In [20]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'],random_state=0)

# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
```

The train_test_split function shuffles the dataset using a pseudo random number generator before making the split.

- data points are sorted by the label
- Using a tests set containing only one ofthe three classes would not tell us much about how well we generalize, so we shuffle our data, to make sure the test data contains data from all classes.
- To make sure that we will get the same output if we run the same function several times, we provide the pseudo random number generator with a fixed seed using therandom_state parameter. This will make the outcome deterministic, so this line willalways have the same outcome. We will always fix the random_state in this way when using randomized procedures in this book.

In [21]:

```
X_train.shape
```

Out[21]:

```
(112, 4)
```

In [22]:

```
X_test.shape
```

Out[22]:

```
(38, 4)
```

1.7.3 First things first: Look at your data

Before building a machine learning model, it is often a good idea to inspect the data, to see if the task is easily solvable without machine learning, or if the desired information might not be contained in the data.

One of the best ways to inspect data is to visualize it. One way to do this is by using a scatter plot.

A scatter plot of the data puts one feature along the x-axis, one feature along the yaxis, and draws a dot for each data point.

Unfortunately, computer screens have only two dimensions, which allows us to only plot two (or maybe three) features at a time. It is difficult to plot datasets with more than three features this way.

One way around this problem is to do a pair plot, which looks at all pairs of two features.

Here is a pair plot of the features in the training set. The data points are colored according to the species the iris belongs to:

In [23]:

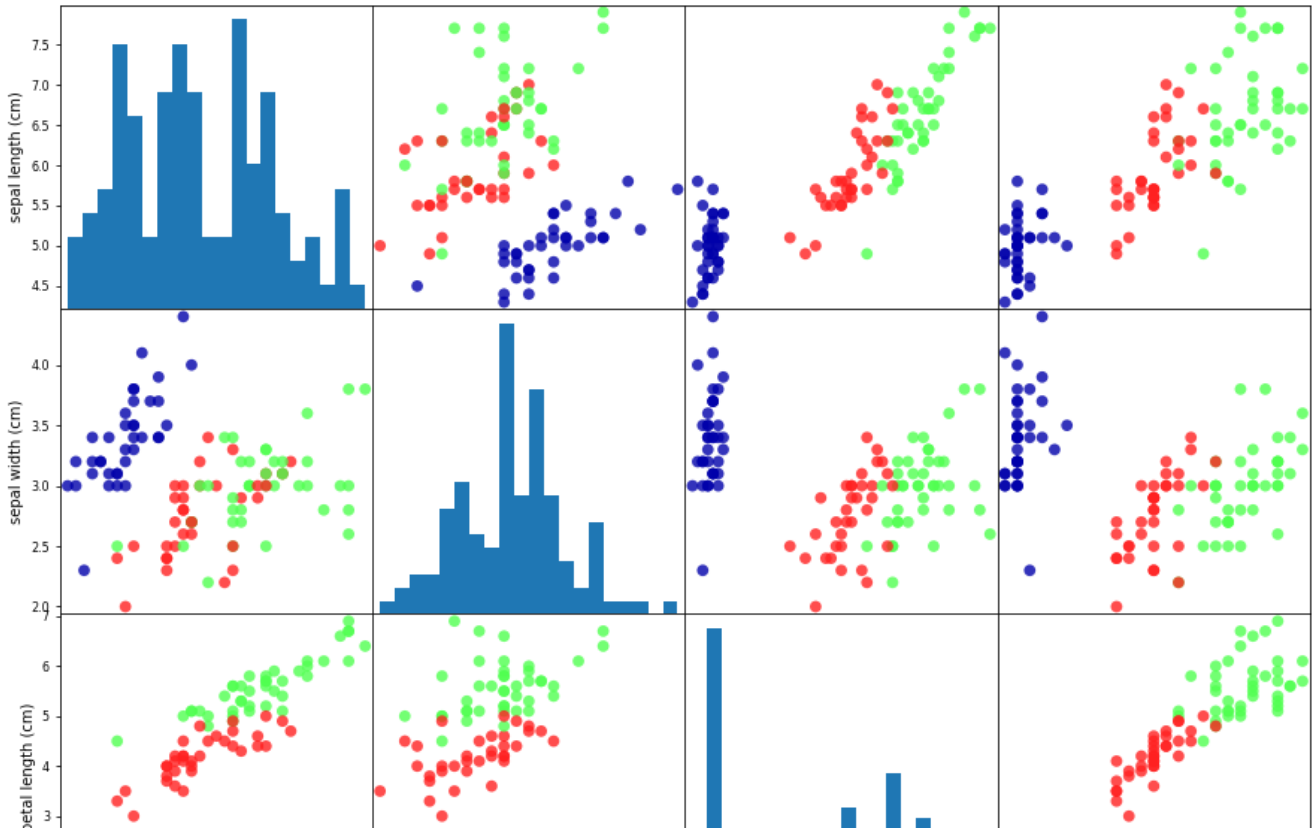
```
!pip install mglearn
import mglearn
# create dataframe from data in X_train
# label the columns using the strings in iris_dataset.feature_names
iris_dataframe = pd.DataFrame(X_train, columns=iris.feature_names)
# create a scatter matrix from the dataframe, color by y_train
pd.plotting.scatter_matrix(iris_dataframe, c=y_train, figsize=(15, 15),
                           marker='o', hist_kwds={'bins': 20}, s=60,
                           alpha=.8, cmap=mglearn.cm3)
```

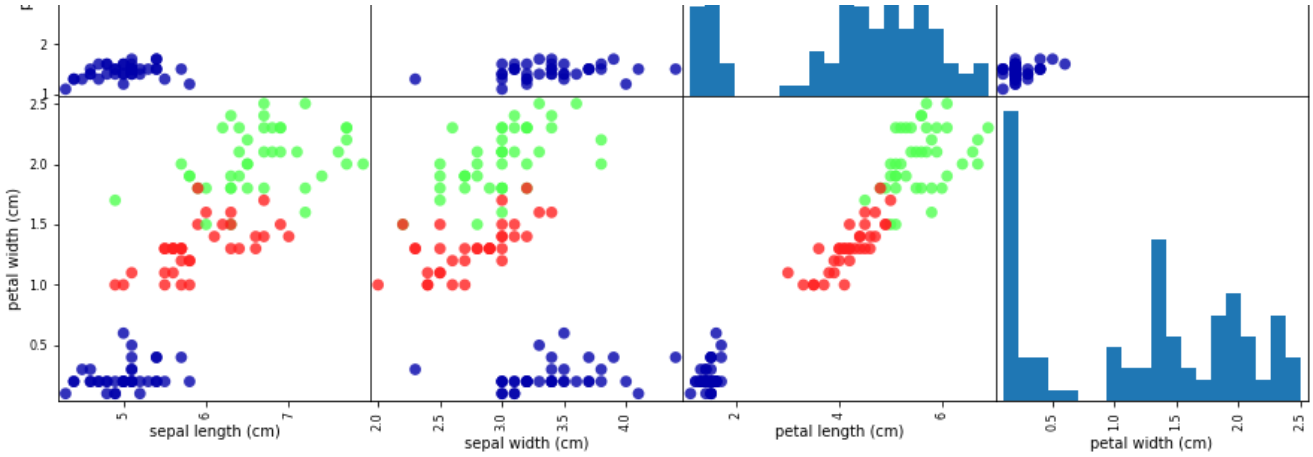
```
Requirement already satisfied: mglearn in c:\programdata\anaconda3\lib\site-packages (0.1.7)
Requirement already satisfied: cycler in c:\programdata\anaconda3\lib\site-packages (from mglearn)
(0.10.0)
```

```
(0.10.0)
Requirement already satisfied: imageio in c:\programdata\anaconda3\lib\site-packages (from
mglearn) (2.3.0)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (from
mglearn) (0.19.1)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-packages (from
mglearn) (2.2.2)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site-packages (from mglearn)
(5.1.0)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packages (from mglearn)
(0.23.0)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (from mglearn)
(1.14.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from cycler-
>mglearn) (1.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib->mglearn) (2.2.0)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib->mglearn) (2.7.3)
Requirement already satisfied: pytz in c:\programdata\anaconda3\lib\site-packages (from
matplotlib->mglearn) (2018.4)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\lib\site-packages
(from matplotlib->mglearn) (1.0.1)
Requirement already satisfied: setuptools in c:\programdata\anaconda3\lib\site-packages (from
kiwisolver>=1.0.1->matplotlib->mglearn) (39.1.0)
```

Out[23]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001543B8B1198>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B8DA4E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B901B70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B934240>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001543B95A8D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B95A908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B9B6630>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543B9DDCC0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001543BA11390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BA38A20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BA690F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BA8F780>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001543BAB9E10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BAEA4E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BB12B70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001543BB46240>]],
      dtype=object)
```





From the plots, we can see that the three classes seem to be relatively well separated using the sepal and petal measurements. This means that a machine learning model will likely be able to learn to separate them.

1.7.4 Building your first model: k nearest neighbors

Contents : https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

math : https://booking.ai/k-nearest-neighbours-from-slow-to-fast-thanks-to-maths-bec682357ccd

- Here we will use a k nearest neighbors classifier, which is easy to understand.
- To make a prediction for a new data point, the algorithm finds the point in the training set that is closest to the new point. Then, it and assigns the label of this closest data training point to the new data point.
  - The k in k nearest neighbors stands for the fact that instead of using only the closest neighbor to the new data point, we can consider any fixed number k of neighbors in the training (for example, the closest three or five neighbors).
- All machine learning models in scikit-learn are implemented in their own class, which are called Estimator classes. The k nearest neighbors classification algorithm is implemented in the KNeighborsClassifier class in the neighbors module.

In [24]:

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

It will also hold the information the algorithm has extracted from the training data. In the case of KNeighborsClassifier, it will just store the training set.

To build the model on the training set, we call the fit method of the knn object, which takes as arguments the numpy array X_train containing the training data and the numpy array y_train of the corresponding training labels:

In [25]:

```
knn.fit(X_train, y_train)
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs
=1, n_neighbors=1, p=2,weights='uniform')

# https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
```

Out[25]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

1.7.5 Making predictions

We can now make predictions using this model on new data, for which we might not know the correct labels.

Imagine we found an iris in the wild with a sepal length of 5cm, a sepal width of 2.9cm, a petal length of 1cm and a petal width of 0.2cm. What species of iris would this be?

We can put this data into a numpy array, again with the shape number of samples (one) times number of features (four):

In [26]:

```
X_new = np.array([[5, 2.9, 1, 0.2]])
```

In [27]:

```
X_new.shape
```

Out[27]:

```
(1, 4)
```

To make prediction we call the predict method of the knn object:

In [28]:

```
prediction = knn.predict(X_new)
prediction
```

Out[28]:

```
array([0])
```

In [29]:

```
iris['target_names'][prediction]
```

Out[29]:

```
array(['setosa'], dtype='<U10')
```

Our model predicts that this new iris belongs to the class 0, meaning its species is Setosa.

But how do we know whether we can trust our model? We don't know the correct species of this sample, which is the the whole point of building the model!

1.7.6 Evaluating the model

This is where the test set that we created earlier comes in. This data was not used to build the model, but we do know what the correct species are for each iris in the test set.

We can make a prediction for an iris in the test data, and compare it against its label (the known species). We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted:

In [30]:

```
y_pred = knn.predict(X_test)
y_pred
```

Out[30]:

```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])
```

In [31]:

```
print("Test set accuracy : {:.2f}".format(np.mean(y_pred == y_test)))
```

```
Test set accuracy : 0.97
```

We can also use the score method of the knn object, which will compute the test set accuracy for us:

In [32]:

```
print("Test set accuracy : {:.2f}".format(knn.score(X_test, y_test)))
```

```
Test set accuracy : 0.97
```

1.8 Summary

We started off formulating a task of predicting which species of iris a particular flower belongs to by using physical measurements of the flower.

making this a supervised learning task.

task. There were three possible species, Setosa, Versicolor or Virginica, which made the task a three-class classification problem.

The possible species are called classes in the classification problem, and the species of a single iris is called its label.

We split our dataset into a training set, to build our model, and a test set, to evaluate how well our model will generalize to new, unseen data.

In [33]:

```
X_train, X_test, y_train, y_test = train_test_split(iris['data'], iris['target'],
random_state=0)
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn.score(X_test, y_test)
```

Out[33]:

```
0.9736842105263158
```