# Data Analysis with Python Reference Book

*Boyan Angelov and Rick Scavetta*

*2018-08-17*

# Contents

# Chapter 1

# Introductions

## 1.1 Set up your software:

Before running the commands in this book you should have the following software locally installed:

- Python from Anaconda
- Start the Anaconda Navigator and open up Jupyter Lab

# Chapter 2

# ch 1: Basic Syntax and Types

## 2.1 1.1 Data Types

- int
- float
- bool: True and False
- str

only use = for adding

```
x = 3
y = 0.5
z = True
```

Types and classes

```
print(type(x))
```

```
## <class 'int'>
```

# Chapter 3

# Differences in handling types

```
print(1 + 1)
```

```
## 2
```

```
print('1' + '1')
```

```
## 11
```

```
print('1' * 5)
```

```
## 11111
```

```
print('1' '1')
```

```
## 11
```

## 3.1 1.2 Containers - List and Dictionaries

### 3.1.1 Lists

Holds heterogeneous information (e.g., [3, 2.718, True, 'hello'])

```
l = [1, '2', True]
print(l)
```

```
## [1, '2', True]
```

```
print(l[0]) # First value, use 0
```

```
## 1
```

```
print(l[-1]) # negative values, begin at end
```

```
## True
```

```
l = [0, 1, 2, 3, 4]
print(l[0:3]) # first to third
```

```
## [0, 1, 2]
```

```
print(l[0:5:2]) # step2
# Implicit
```

```
## [0, 2, 4]
```

```
print(l[:3]) # first to third
```

```
## [0, 1, 2]
```

```
print(l[1:]) # second to end
```

```
## [1, 2, 3, 4]
```

```
print(l[::2]) # all, every second
# Explicit
```

```
## [0, 2, 4]
```

```
print(l[0:3]) #
```

```
## [0, 1, 2]
```

```
print(l[1:5]) #
```

```
## [1, 2, 3, 4]
```

```
print(l[0:5:2])
```

```
## [0, 2, 4]
```

### 3.1.2 Indexing

- Begins at zero
- Negative values go in reverse
- Left inclusive, right exclusive
- Third value in index specifies step size
- Can be implicit or explicit

### 3.1.3 Dictionaries

Lists are unlabelledm, but dictionaries provide key:value pairs using `{}` and `:`.

```
d = {'int_value':3,
     'bool_value':False,
     'str_value':'hello'}
print(d) #
```

```
## {'int_value': 3, 'bool_value': False, 'str_value': 'hello'}
```

```
print(d['str_value'])
```

```
## hello
```

Notice that the order when printing is different than in definition. Order is not guaranteed! So don't rely on the printed order of the keys.

place key in the `[]` to extract that value.

```
l = [0, 1, 2, 3, 4]
d = {'int_value':3, 'bool_value':False, 'str_value':'hello'}
print(len(l)) #
```

```
## 5
```

```
print(len(d))
```

```
## 3
```

## 3.2 1.3 Functions, Methods, and Libraries

- Calling Functions
- R and Python functions work the same
- Python: Functions and Methods

### 3.2.1 Methods vs Functions

- python is an object-oriented programming language:
  - Attributes
  - Methods
- Methods are functions that an object can call on itself
- Functions are called on an object

```
l = [0, 1, 2, 3, 4]
len(l)   # call a function
```

But not this:

```
l.len()  # not a method
```

e.g. methods are functions that an object calls, whereas you pass objects into functions.

i.e. we pass `l` a list to the function `len()`, rather than calling the method `len()` on `l` using the `.`

### 3.2.2 periods

Periods, `.`, have very specific meanings and uses

You can't use a `.` in a variable or function name.

Uses:

Append a list: Call the append method on the list `l` using the dot notation. So your not passing `l` to the `append()` function, rather you're calling the `append()` method on list `l`.

```
l = [1, "2", True]
l.append('appended value')
print(l)
```

```
## [1, '2', True, 'appended value']
```

Update a dictionary:

```
d = {'int_value':3, 'bool_value':False, 'str_value':'hello'}
d.update({'str_value':'new_value', 'new_key':'new_value'})
print(d)
```

```
## {'int_value': 3, 'bool_value': False, 'str_value': 'new_value', 'new_key': 'new_value'}
```

Methods are special functions that belong to specific objects. e.g. if you try calling append on a dictionary object, you'll get an error because the method append is not defined for a dictionary but it is defined for a list.

### 3.2.3 Libraries

Arrays and data frame are not in Python by default. They come from numby (array and matrix) and pandas (dataframe)

- Libraries provide more functionality
- Arrays and dataframes are not built into Python
- Arrays and matricies come from numpy
- Dataframes come from pandas

use import to load the library, but use an alias with the as *keyword*.

```
import numpy
# arr = numpy.loadtxt('my_file.csv', delimiter=',')
import numpy as np
# arr = np.loadtxt('my_file.csv', delimiter=',')
```

```
import pandas as pd
# df = pd.read_csv('my_file.csv')
# df.head()
```

## 3.3 Extra

Find out where your packages are. In python execute:

```
import pandas
import os
path = os.path.dirname(pandas.__file__)
print(path)
```

```
## /anaconda3/lib/python3.6/site-packages/pandas
```

```
import sys
print(sys.version)
```

```
## 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 12:06:34)
## [GCC 4.2.1 Compatible Clang 4.0.1 (tags/RELEASE_401/final)]
```

```
print(sys.executable)
```

```
## /anaconda3/bin/python
```

```
print(system("which python"))
```

```
## [1] 0
```

# Chapter 4

# ch 2

## 4.1 2.1 If statements and Loops

*Whitespace and indentation is a fundamental part of writing python code and is not optional.*

i.e. you define code blocks with indentations. Use 4 spaces to indent code

```
if 5 == 5:
    print('True')
```

```
## True
```

Everything in the indented code is executed in python. That's like `{}` in R

`elif` and `else`:

```
val = 2
if val == 1:
    print('snap')
elif val == 2:
    print('crackle')
else:
    print('pop')
```

```
## crackle
```

for loops:

```
num_val = [1, 2, 3, 4]
for value in num_val:
    print(value)
```

```
## 1
## 2
## 3
## 4
```

## 4.2 2.2 Functions

### 4.2.1 2.2.1 Writing your own functions:

use the `def` keyword in Python. The body of the function is indented. The `return` statement is necessary! The last line is not automatically returned.

```
def my_mean(x, y):
    num = x + y
    dem = 2
    return num / dem
print(my_mean(10, 20))
```

```
## 15.0
```

### 4.2.2 2.2.2 Functions can call other functions:

```
def my_sq(x):
    return x ** 2

def my_sq_mean(y, z):
    return (my_sq(y) + my_sq(z)) / 2

print(my_sq_mean(10, 12))
```

```
## 122.0
```

### 4.2.3 2.2.3 Anonymous and lambda functions

In Python, use the `lambda` keyword to make a lambda functions with the apply method (see later):

```
def add_1(x):
    return x + 1

a1_lam = lambda x: x + 1
print(a1_lam(3))
```

```
## 4
```

You can also save a lambda funcion.

## 4.3 2.3 Comprehensions

One of the most common tasks you'll perform when cleaning or summarising data is to iterate over a list, apply some function or perform some calculation on each element and then return the result as a new list.

### 4.3.1 2.3.1 List comprehensions

Comprehensions are loops:

- Iterate (loop) through a list

- Perform some function
- Append results into a new list

Open a `[]` and write the body of your `for loop` first then write the for statement to iterate over data. No `:` at the end!

`for loop`:

```
data = [1, 2, 3, 4, 5]
new = []
for x in data:
    new.append(x**2)
print(new)
```

```
## [1, 4, 9, 16, 25]
```

Comprehension:

```
data = [1, 2, 3, 4, 5]
new = [x**2 for x in data]
print(new)
```

```
## [1, 4, 9, 16, 25]
```

### 4.3.2 2.3.3 Dictionary comprehensions

Result is a dictionary, so you need to create a key:value pair. Use `{}` here

`for loop`:

```
data = [1, 2, 3, 4, 5]
new = {}
for x in data:
    new[x] = x**2
print(new)
```

```
## {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Comprehension:

```
data = [1, 2, 3, 4, 5]
new = {x: x**2 for x in data}
print(new)
```

```
## {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

### 4.3.3 2.3.4 Alternatives to for loops

- `for loop` to iterate
- R: sapply, lapply, apply functions
- Python: map function (here) and apply method

`map()` takes the name of the function as the first argument and a list of values as the second. The specified funtion is applied to all the elements in the second argument, one at a time, like `sapply()` and `lapply()` in R. The result is a `map` object, so you need to convert it to a `list` to view it.

`for loop`:

```
def sq(x):
    return x**2
l = [1, 2, 3]
for i in l:
    print(sq(i))
```

```
## 1
## 4
## 9
```

map()

```
map(sq, l)
print(list(map(sq, l)))
```

```
## [1, 4, 9]
```

# Chapter 5

# Ch 3: Selecting Data in Pandas

## 5.1 3.1.1 Data frames - 2D

You can pass a dictionary to a `pd.DataFrame()` function. Index specifies the row names.

```
df = data.frame(A = 1:3,
                B = 4:6,
                C = 7:9,
                row.names = c("x", "y", "z"))
```

```
import pandas as pd
df = pd.DataFrame({
            'A': [1, 2, 3],
            'B': [4, 5, 6],
            'C': [7, 8, 9]},
            index = ['x', 'y', 'z'])
print(df)
```

```
##    A  B  C
## x  1  4  7
## y  2  5  8
## z  3  6  9
```

Select columns:

```
df['A']
df.A
df[['A', 'B']]
```

Subset rows by index position (iloc)

```
df
df.iloc[0] # First row
df.iloc[[0, 1]] # a list of rows, the first two rows
```

```
df.iloc[0, :] # all columns, : after the comma
df.iloc[[0, 1], :]
```

Subsetting rows by label (loc)

```
df.loc['x'] # The xth row
df.loc[['x', 'y']] # The xth and yth row
```

Both rows and columns

```
df.loc['x', 'A']
df.loc[['x', 'y'], ['A', 'B']]
```

Conditional subsetting

```
df[df.A == 3]
df[(df.A == 3) | (df.B == 4)]
```

## 5.2 3.1.2 Attributes

Shape is an attribute, so no `()`:

```
df.shape
```

NOT a functon

```
df.shape()
```

## 5.3 3.2.1 Data Types

type returns the type of an object. Data frames have more info.

```
str(df)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ A: int  1 2 3
##  $ B: int  4 5 6
##  $ C: int  7 8 9
```

```
df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 3 entries, x to z
## Data columns (total 3 columns):
## A    3 non-null int64
## B    3 non-null int64
## C    3 non-null int64
## dtypes: int64(3)
## memory usage: 176.0+ bytes
```

### 5.3.1 3.2.2 Change data types:

```
df$A <- as.character(df$A)
str(df)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ A: chr  "1" "2" "3"
##  $ B: int  4 5 6
##  $ C: int  7 8 9
```

```
df['A'] = df['A'].astype(str)
df.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## Index: 3 entries, x to z
## Data columns (total 3 columns):
## A    3 non-null object
## B    3 non-null int64
## C    3 non-null int64
## dtypes: int64(2), object(1)
## memory usage: 176.0+ bytes
```

Type `object` means `string`. Access built-in string methods with str accessor

## 5.3.2 3.2.3 Accessors

### 5.3.2.1 3.2.3.1 String Accessor

e.g. strip removed leading and trailing whitespece

```
df = pd.DataFrame({'name': ['Daniel  ','  Eric', '  Julia  ']})
df['name_strip'] = df['name'].str.strip()
df
```

### 5.3.2.2 3.2.3.1 Categorical Accessor

Like factors in R

```
df = pd.DataFrame({'name': ['Daniel','Eric', 'Julia'],
                   'gender':['Male', 'Male', 'Female']})
df['gender_cat'] = df['gender'].astype('category')
```

See the categories (levels in R) by calling the `cat` accessor and the `categories` attribute on the column. Use the `codes` attribute to get the codes.

```
df['gender_cat'].cat.categories
df.gender_cat.cat.codes
```

### 5.3.2.3 3.2.3.3 Date Accessor

Dates: use the `to_datetime()` function

```
df = pd.DataFrame({'name': ['Rosaline Franklin', 'William Gosset'],
                   'born': ['1920-07-25', '1876-06-13']})
df['born_dt'] = pd.to_datetime(df['born'])
```

Similar to strings and categorical values, you can access date components with the `dt` accessor

```
df['born_dt'].dt.day
df['born_dt'].dt.month
df['born_dt'].dt.year
```

## 5.4 3.3 Advanced Pandas

### 5.4.1 3.3.1 Missing data

- `NaN` missing values from from numpy
- `np.NaN`, `np.NAN`, `np.nan` are all the same as the NA R value
- check missing with `pd.isnull`
- Check non-missing with `pd.notnull`
- `pd.isnull` is an alias for `pd.isna`

```
df = pd.DataFrame({
            'name': ["John Smith", "Jane Doe", "Mary Johnson"],
            'treatment_a': [None, 16, 3],
            'treatment_b': [2, 11, 1]})
print(df)
```

```
##            name  treatment_a  treatment_b
## 0    John Smith          NaN            2
## 1      Jane Doe         16.0           11
## 2  Mary Johnson          3.0            1
```

How to replace the `NaN` with the mean of the row?

```
a_mean = df['treatment_a'].mean()
print(a_mean)
```

```
## 9.5
```

use the `fillna` method:

```
df['a_fill'] = df['treatment_a'].fillna(a_mean)
print(df)
```

```
##            name  treatment_a  treatment_b  a_fill
## 0    John Smith          NaN            2     9.5
## 1      Jane Doe         16.0           11    16.0
## 2  Mary Johnson          3.0            1     3.0
```

### 5.4.2 3.3.2 Applying custom functions

Use all the following: - Built-in functions - Custom functions

Use: `apply` method and pass in an axis

```
df <- data.frame(a = 1:3,
                 b = 4:6)

apply(df, 2, mean)
```

```
## a b
## 2 5
```

```
apply(df, 1, mean)
```

```
## [1] 2.5 3.5 4.5
```

```
import numpy as np
df = pd.DataFrame({'A': [1, 2, 3],
```

```
                   'B': [4, 5, 6]})
df.apply(np.mean, axis=0) # Column-wise
df.apply(np.mean, axis=1) # Row-wise
```

### 5.4.3 3.3.4 Tidy data

Reshaping and tidying our data (Tidy Data Paper)[http://vita.had.co.nz/papers/tidy-data.pdf]

- Each row is an observation
- Each column is a variable
- Each type of observational unit forms a table

Tidy Melt

```
df = pd.DataFrame({
            'name': ["John Smith", "Jane Doe", "Mary Johnson"],
            'treatment_a': [None, 16, 3],
            'treatment_b': [2, 11, 1]})
print(df)
```

```
##            name  treatment_a  treatment_b
## 0    John Smith          NaN            2
## 1      Jane Doe         16.0           11
## 2  Mary Johnson          3.0            1
```

```
df_melt = pd.melt(df, id_vars='name')
print(df_melt)
```

```
##            name     variable  value
## 0    John Smith  treatment_a    NaN
## 1      Jane Doe  treatment_a   16.0
## 2  Mary Johnson  treatment_a    3.0
## 3    John Smith  treatment_b    2.0
## 4      Jane Doe  treatment_b   11.0
## 5  Mary Johnson  treatment_b    1.0
```

Tidy pivot_table

```
df_melt_pivot = pd.pivot_table(df_melt,
                               index='name',
                               columns='variable',
                               values='value')

print(df_melt_pivot)
```

```
## variable      treatment_a  treatment_b
## name
## Jane Doe             16.0         11.0
## John Smith            NaN          2.0
## Mary Johnson          3.0          1.0
```

This results in a hierarchial index, so to get a regular flat data frame, call

```
df_melt_pivot.reset_index()
print(df_melt_pivot)
```

```
## variable      treatment_a  treatment_b
```

```
## name
## Jane Doe             16.0         11.0
## John Smith            NaN          2.0
## Mary Johnson          3.0          1.0
```

### 5.4.4 3.3.5 groupby operations

- groupby: split-apply-combine
  - *split* data into separate partitions
  - *apply* a function on each partition
  - *combine* the results

```
print(df_melt)
```

```
##            name     variable  value
## 0    John Smith  treatment_a    NaN
## 1      Jane Doe  treatment_a   16.0
## 2  Mary Johnson  treatment_a    3.0
## 3    John Smith  treatment_b    2.0
## 4      Jane Doe  treatment_b   11.0
## 5  Mary Johnson  treatment_b    1.0
```

```
df_melt.groupby('name')['value'].mean()
```

# Chapter 6

# ch 4: Data Visualization with Pandas

## 6.1 4.1.1 Plot Methods

- `plot()` method

Works on the pandas `DataFrame` and `Series` objects

Pass plot the `kind` argument: - 'line' : line plot (default) - 'bar' : vertical bar plot - 'barh' : horizontal bar plot - 'hist' : histogram - 'box' : boxplot - 'kde' : Kernel Density Estimation plot - 'density' : same as 'kde' - 'area' : area plot - 'pie' : pie plot - 'scatter' : scatter plot - 'hexbin' : hexbin plot

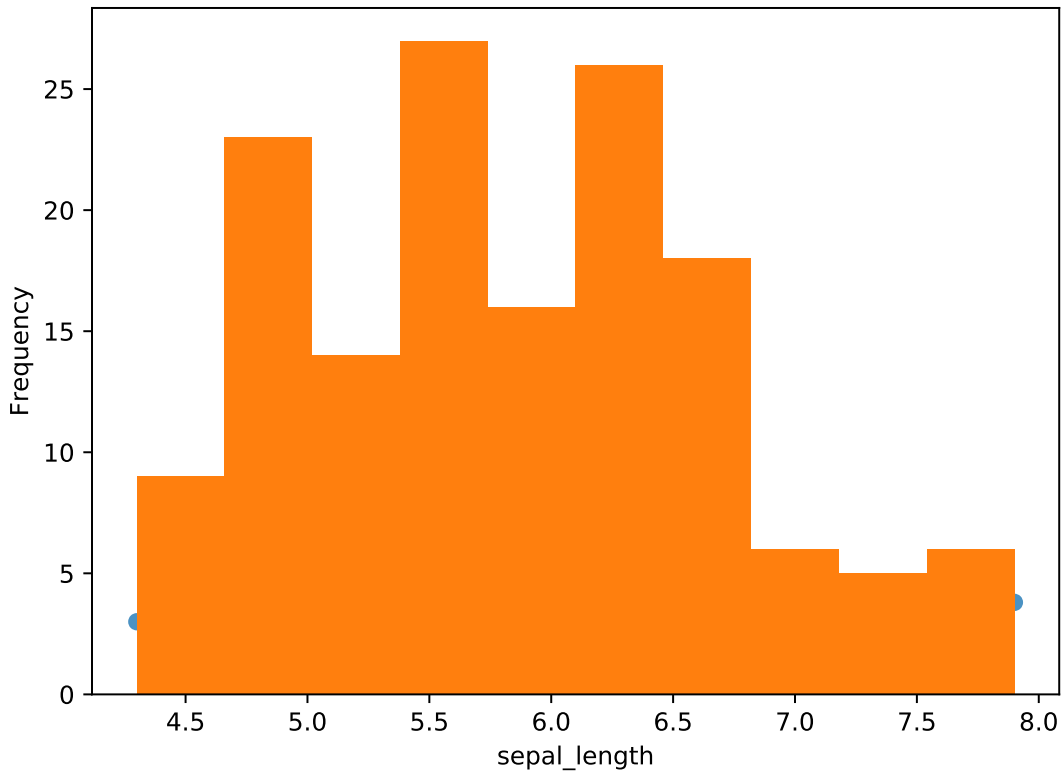
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
iris = datasets.load_iris()
```

Obtain data

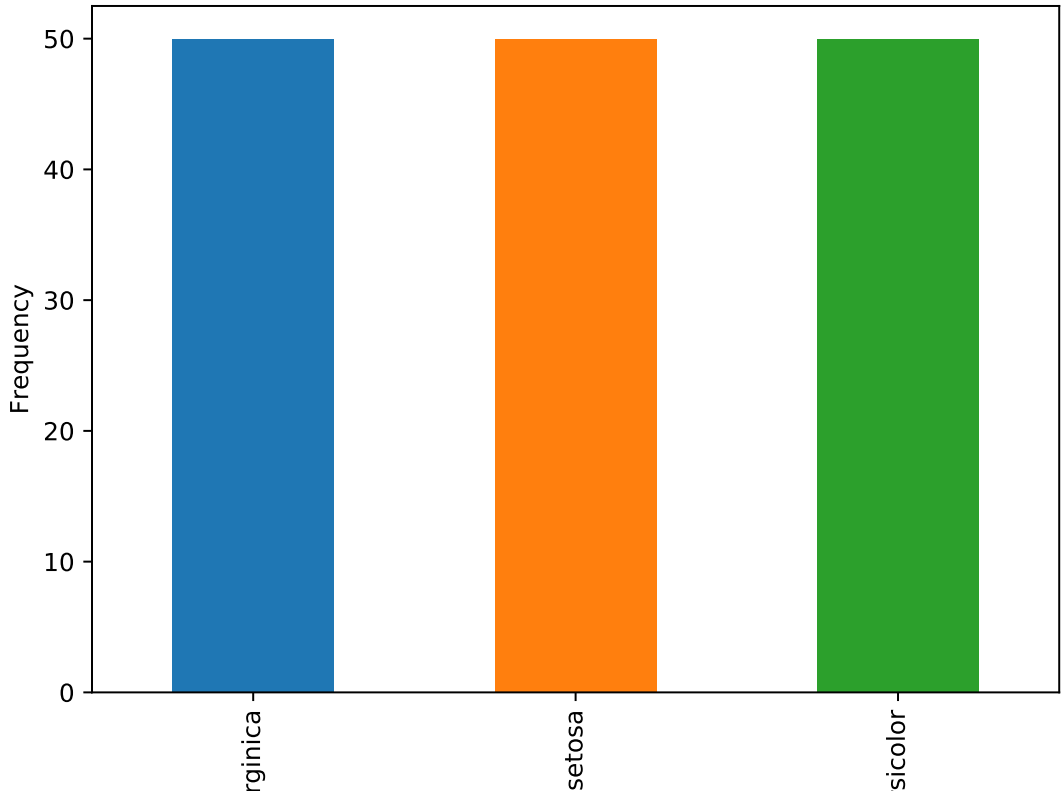
```
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']
iris = pd.read_csv(url, names=names)
```

Histogram

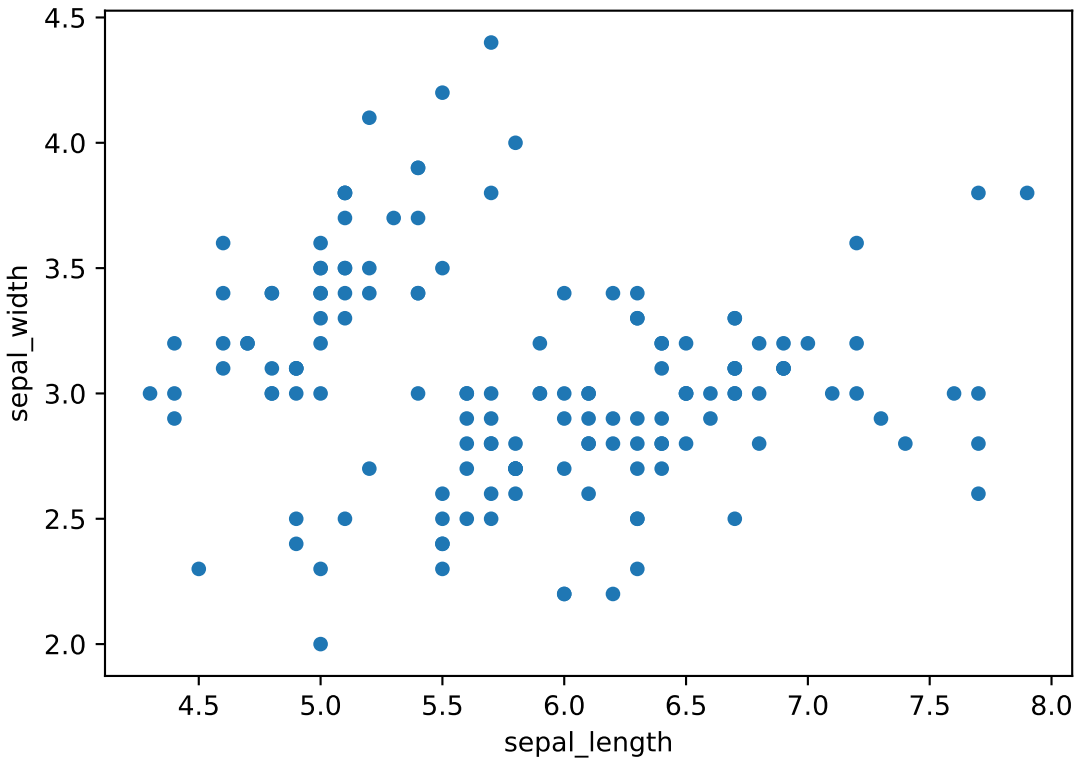
```
import matplotlib.pyplot as plt
iris['sepal_length'].plot(kind='hist')
plt.show()
```



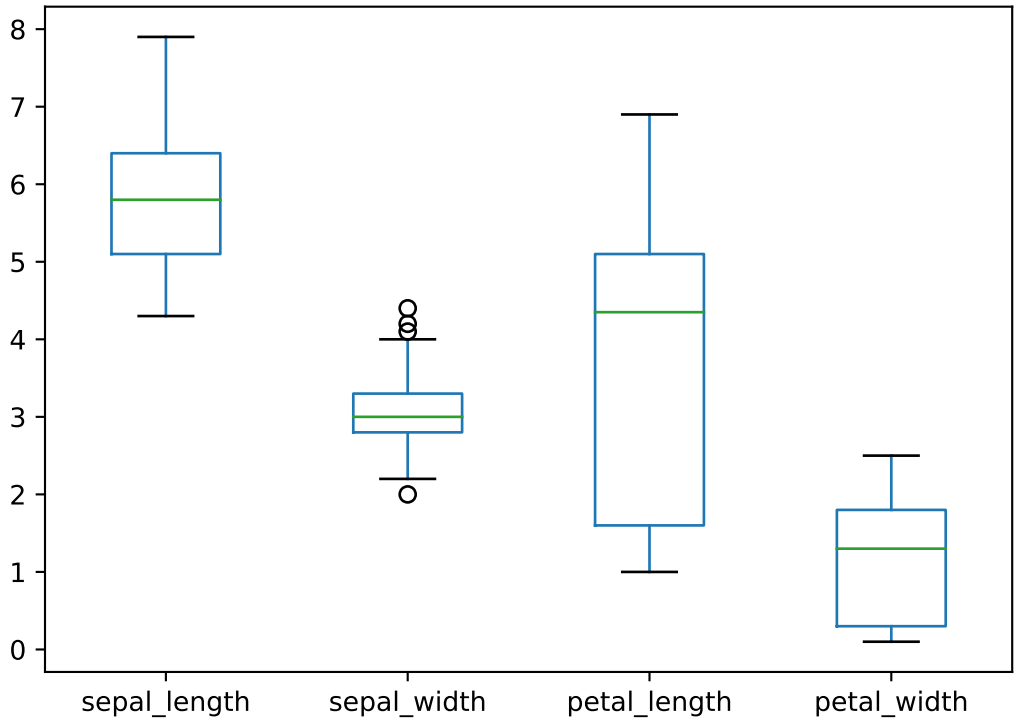

```
cts = iris['species'].value_counts()
cts.plot(kind='bar')
plt.show()
```




```
iris.plot(kind='scatter', x='sepal_length', y='sepal_width')
plt.show()
```



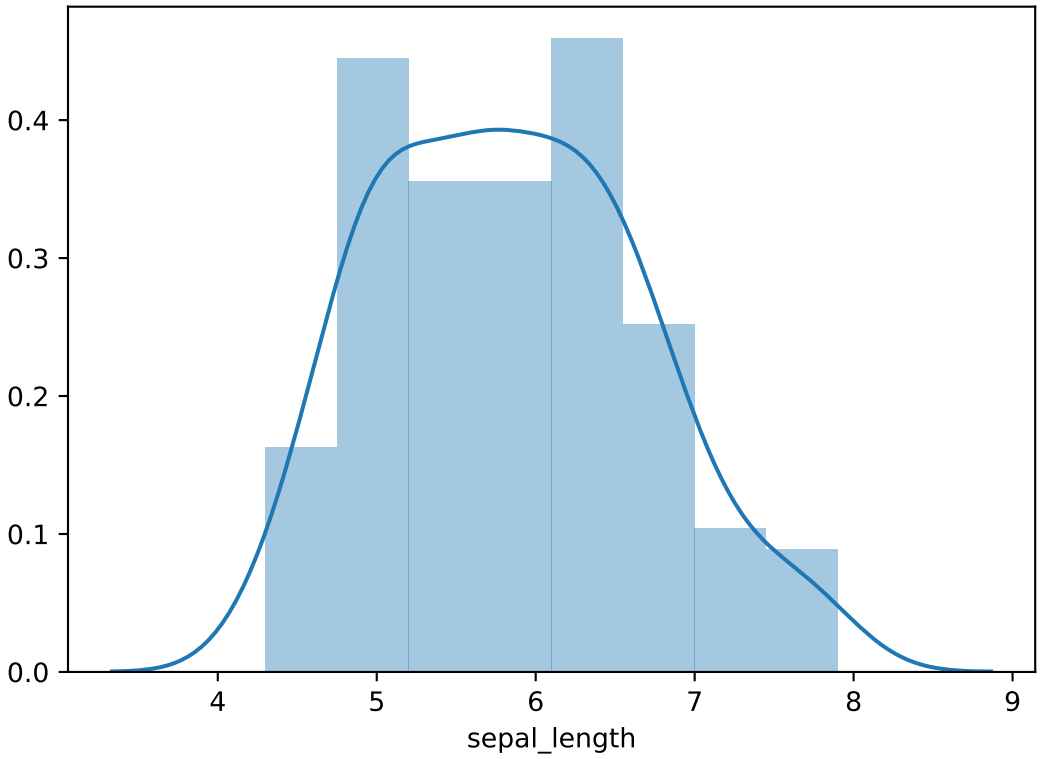

```
iris.plot(kind='box')
plt.show()
```




```
iris.boxplot(by='species', column='sepal_length')
plt.show()
```

## 6.2 4.2 Data Visualization with Seaborn

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.clf()
sns.distplot(iris['sepal_length'])
plt.show()
```

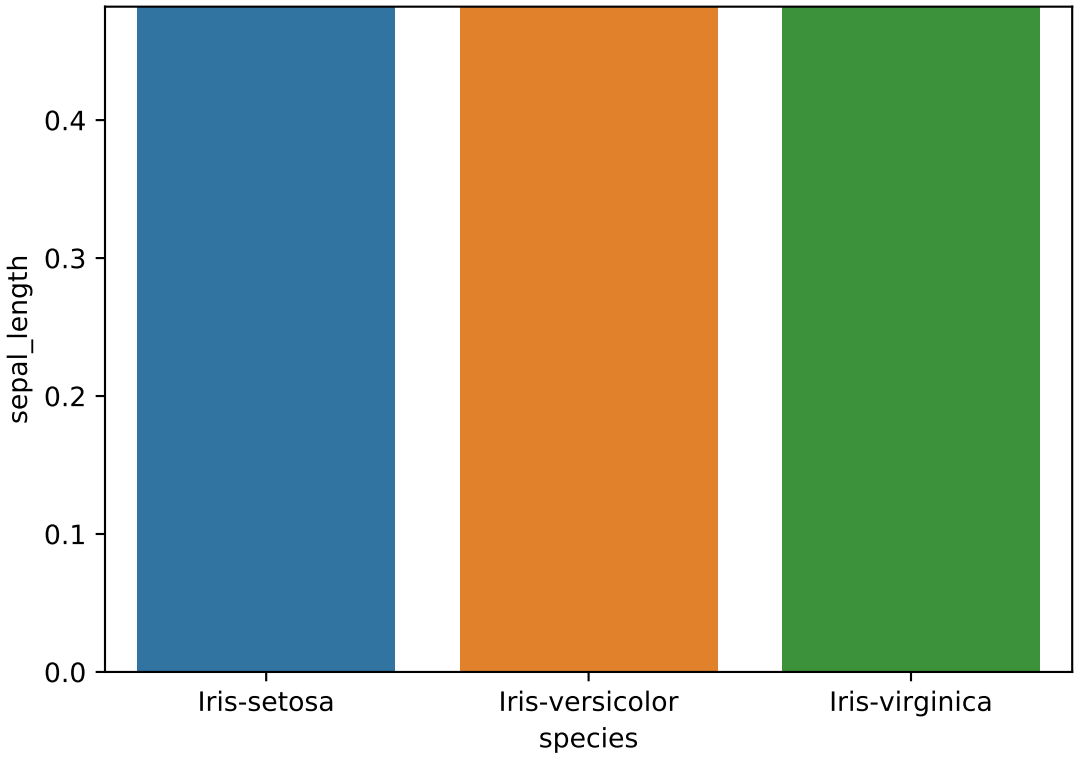



Bar plot:No need to pretabulate the data:

```
sns.countplot('species', data=iris)
plt.show()
```


Box plots:

```
sns.boxplot(x='species', y='sepal_length', data=iris)
plt.show()
```




regplot:

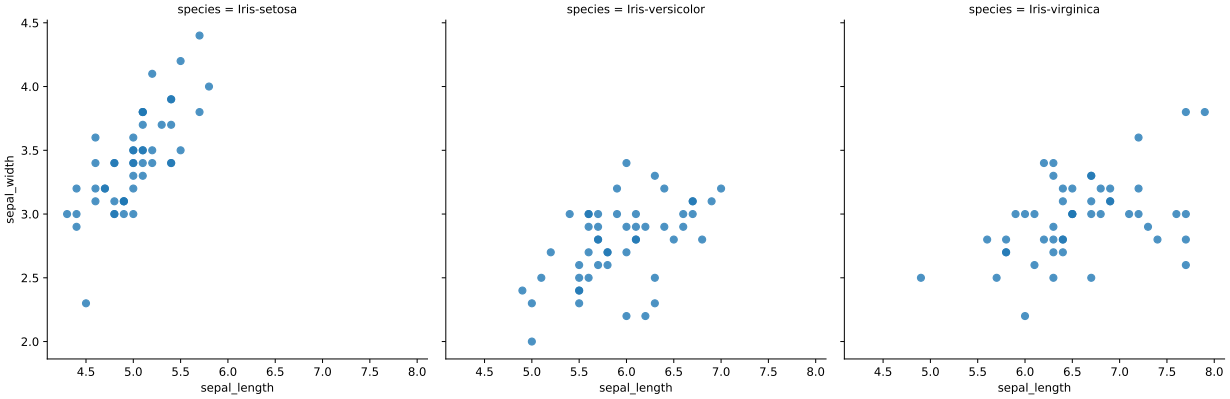
```
sns.regplot(x='sepal_length', y='sepal_width', data=iris)
plt.show()
```


No regression line:

```
sns.regplot(x='sepal_length', y='sepal_width', data=iris,
            fit_reg=False)
plt.show()
```

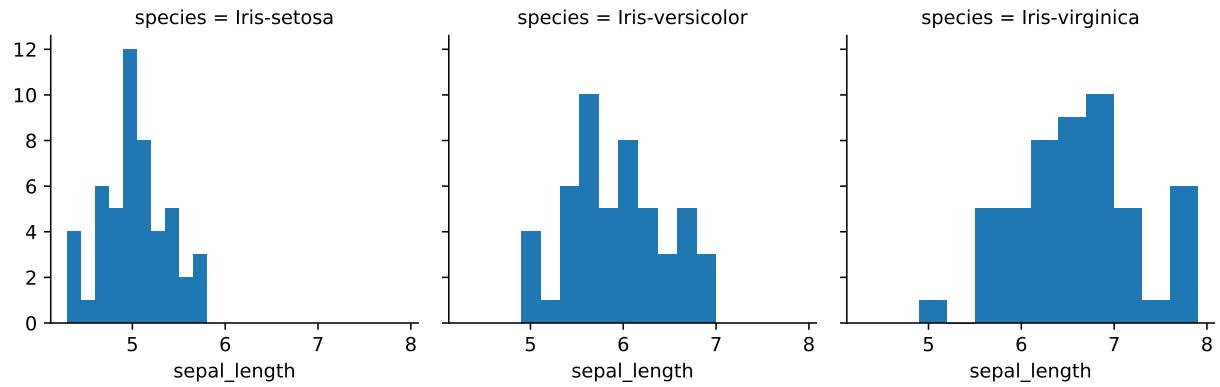

Specify row and col arguments

```
sns.lmplot(x='sepal_length', y='sepal_width', data=iris,
           fit_reg=False,
           col='species')
plt.show()
```




Create a facet grid object and then map the plot into that.

```
g = sns.FacetGrid(iris, col="species")
g = g.map(plt.hist, "sepal_length")
plt.show()
```

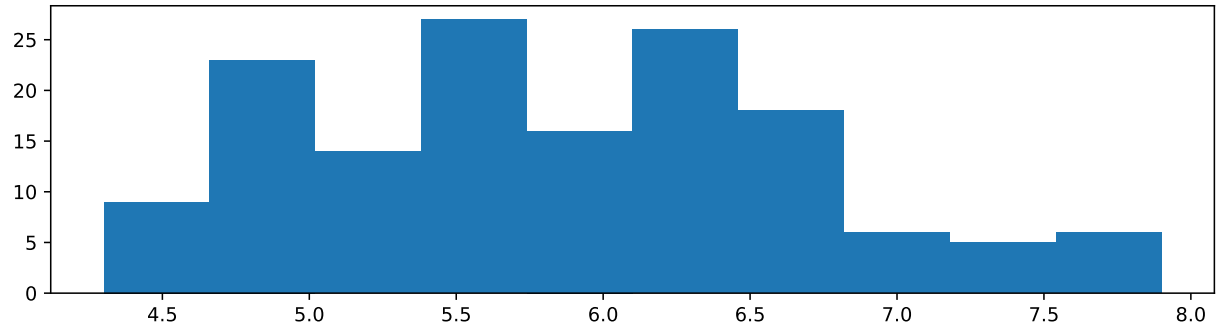



g

## 6.3 4.3 Data Visualization with Matplotlib

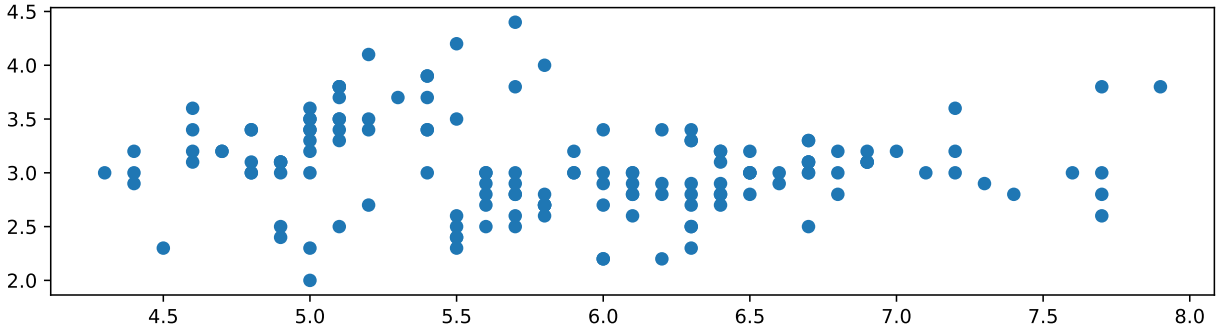
Histogram

```
import matplotlib.pyplot as plt
plt.clf()
plt.hist(iris['sepal_length'])
plt.show()
```

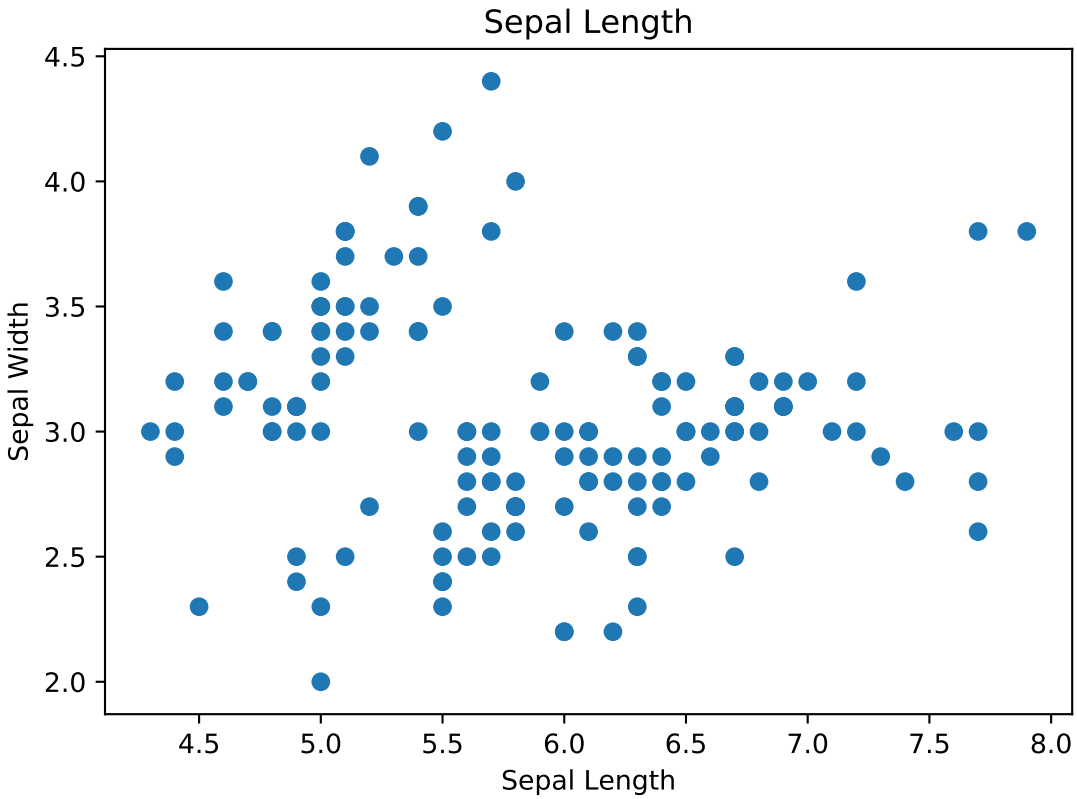


Scatter plot

```
plt.clf()
plt.scatter(iris['sepal_length'], iris['sepal_width'])
plt.show()
```

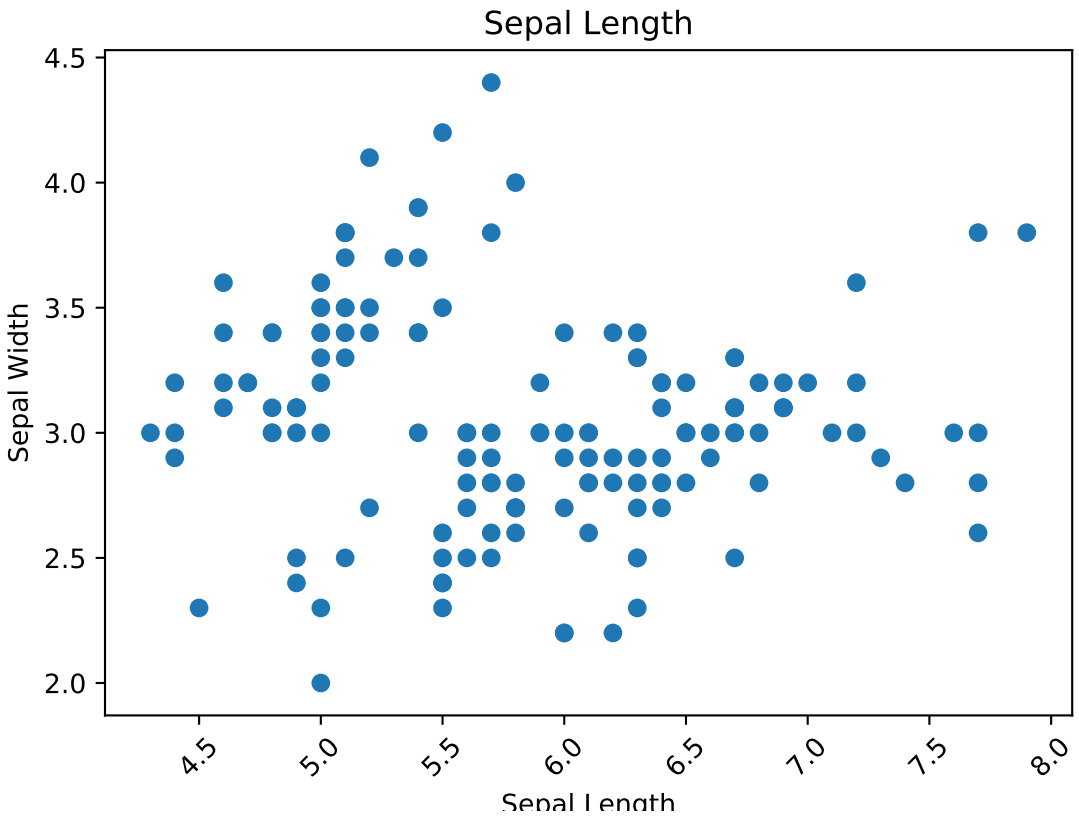



```
fig, ax = plt.subplots()
ax.scatter(iris['sepal_length'], iris['sepal_width'])
ax.set_title('Sepal Length')
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
plt.show()
```




```
fig, ax = plt.subplots()
ax.scatter(iris['sepal_length'], iris['sepal_width'])
ax.set_title('Sepal Length')
ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
plt.xticks(rotation=45) # rotate the x-axis ticks
```

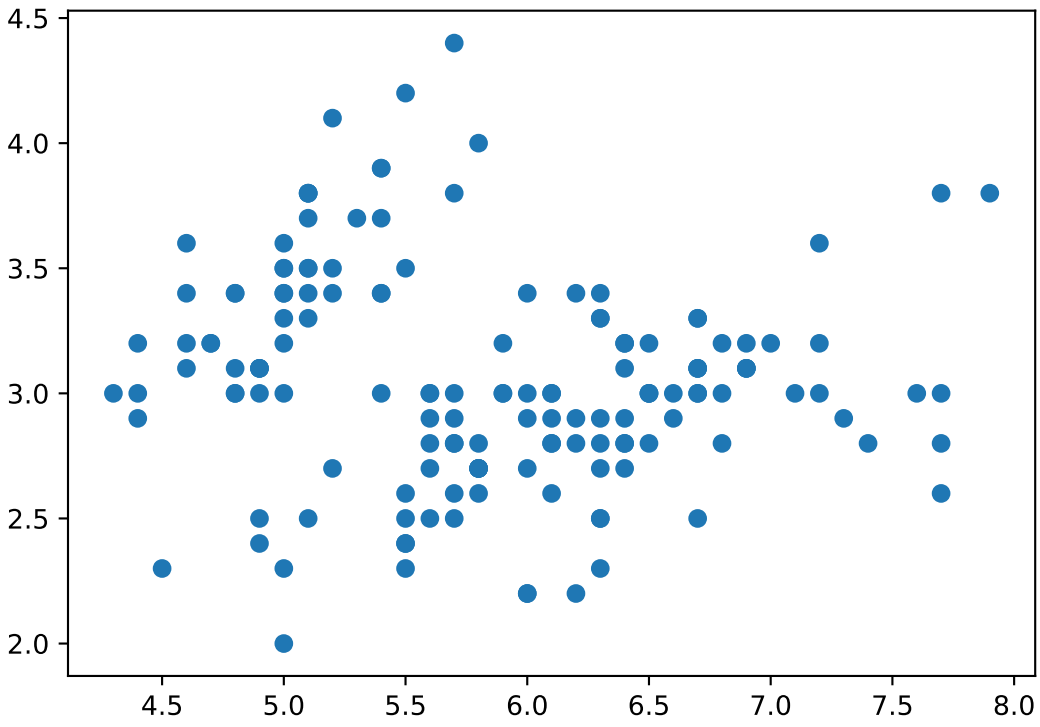
```
plt.show()
```




The figure‘ `fig`` is an entire image, but an Axes,`ax‘, is an individual plot in it, i.e. sub plots.

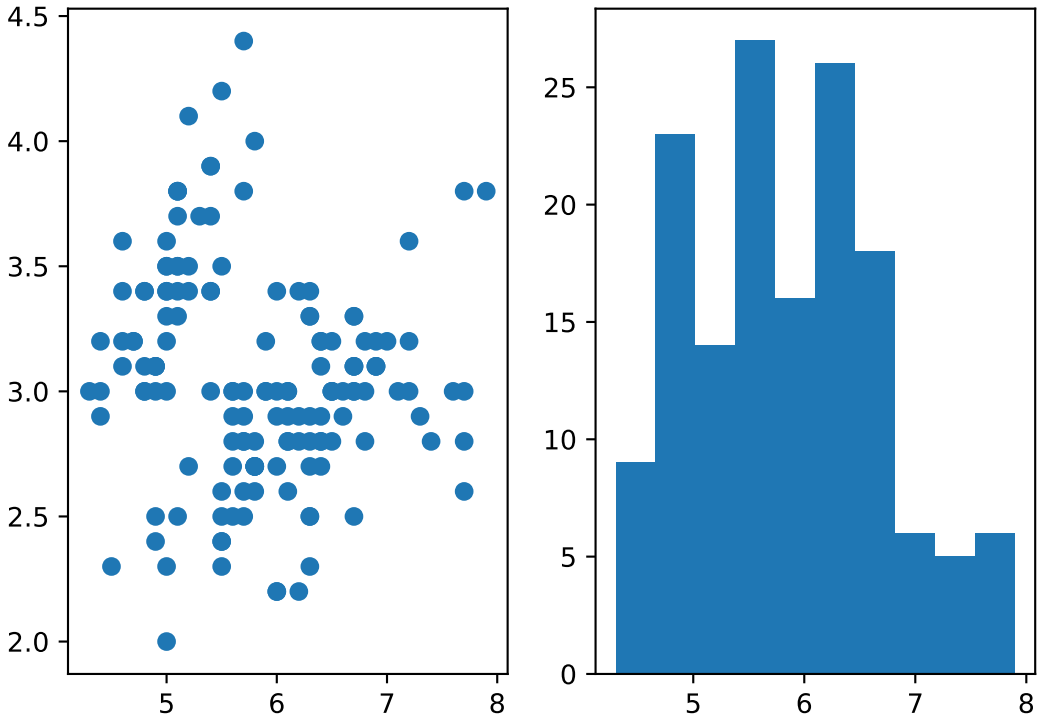
we can call `scatter()` on the ax object.

```
fig, ax = plt.subplots()
ax.scatter(iris['sepal_length'], iris['sepal_width'])
plt.show()
```



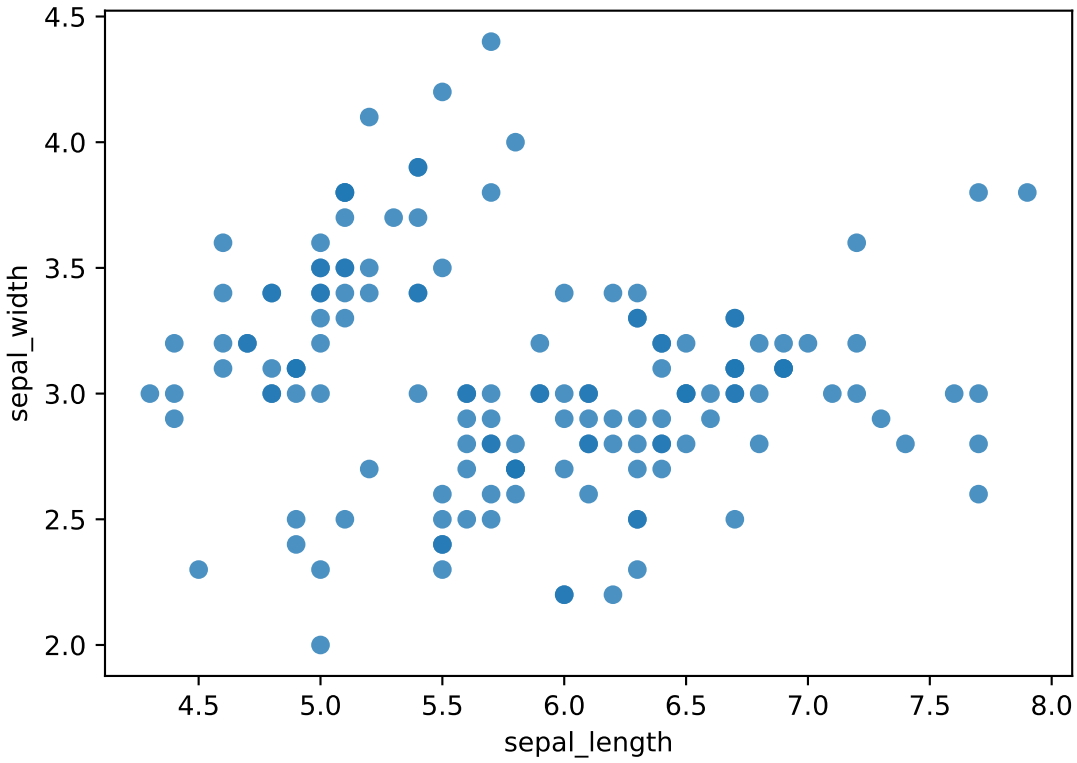

Here we create two axes:

```
fig, (ax1, ax2) = plt.subplots(1, 2)
ax1.scatter(iris['sepal_length'], iris['sepal_width'])
ax2.hist(iris['sepal_length'])
plt.show()
```




You can mix functions form different plots: `ax = ax`

```
import seaborn as sns
fig, ax = plt.subplots()
sns.regplot(x='sepal_length', y='sepal_width',
            data=iris, fit_reg=False, ax=ax)
plt.show()
```




Prevent plots from overlapping by clearing the figure with `plt.clf()`.

## 6.4 Libraries

### 6.4.1 Numpy

Adds Python support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

### 6.4.2 SciPy

A collection of mathematical algorithms and convenience functions built on the Numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.

### 6.4.3 Pandas

Software library written for data manipulation and analysis in Python. Offers data structures and operations for manipulating numerical tables and time series.

### 6.4.4 Scikit-learn

A Python module for machine learning built on top of SciPy and distributed under the 3-Clause BSD license.

# Chapter 7

# Ch 5: virtualenv

## 7.1 Setting up your virtual env

```
virtualenv ~/ds_intro -p /anaconda3/bin/python3.6
python -V

source ~/ds_intro/bin/activate
pip install pandas
pip install sklearn
pip install scipy
pip install jupyter

pip install seaborn
atom
pip install mpl_toolkits
pip install sympy
deactivate
```

Connect to shiny server

```
ssh -i .ssh/Rick2018.pem ubuntu@ec2-18-185-131-255.eu-central-1.compute.amazonaws.com
```

on server

```
sudo -i
```

As root user, install packages

```
sudo su - -c "R -e \"install.packages('shiny', repos='http://cran.rstudio.com/')\""
```