python™ 2 programming

The Way Ahead

Content

- Useful python packages overview

- Python has hundreds of modules covering almost all areas
  - GUI
  - Scientific
  - Big data
  - Gaming
  - Web and Networking
  - ….

Web python packages overview

- Client side
  - urllib2
  - httplib
  - requests

- Server side
  - Django
  - Flask
  - Pyramid
  - Tornado

The requests package

- “Requests: HTTP for Humans”
- REST Requests

```
import requests
r = requests.get('https://api.github.com/events')
r = requests.post('http://httpbin.org/post', data = {'key':'value'})
r = requests.put('http://httpbin.org/put', data = {'key':'value'})
r = requests.delete('http://httpbin.org/delete')
r = requests.head('http://httpbin.org/get')
r = requests.options('http://httpbin.org/get')
```

Http Get

```
import  requestsr = requests.get('http://127.0.0.1:8000/data')

t = r.content

j = r.json()
```

```
args = {'key1': 'value1', 'key2': 'value2'}

r = requests.get('http://127.0.0.1:8000/data',
params=args)

print(r.url)
```

```
headers = {'user-agent': 'my-app/0.0.1','Content-Type':
'application/json'}

r = requests.get('http://127.0.0.1:8000/data',
headers=headers)

print r.json()
```

Http Post

```
args = {'key1': 'value1', 'key2': 'value2'}

r = requests.post("http://127.0.0.1:8000/data/",data=args)

print(r.text)


import json

args = {'key1': 'value1', 'key2': 'value2'}

strr = requests.post("http://127.0.0.1:8000/data/",
json=json.dumps(args))

print(r.text)
```

```
url = 'http://httpbin.org/post'>>> files = {'file':
open('report.xls', 'rb')}

r = requests.post(url, files=files)

r.text
```

Raw Response Content

```
r = requests.get('https://api.github.com/events', stream=True)

r.raw

#<requests.packages.urllib3.response.HTTPResponse object at
0x101194810>

r.raw.read(10)

#'\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x03‘

r = requests.get('https://api.github.com/events', stream=True)

with open(filename, 'wb') as fd:

    for chunk in r.iter_content(chunk_size=128):

        fd.write(chunk)
```

Cookies

- **Get**

```
url =
'http://example.com/some/cookie/setting/url‘
r = requests.get(url)
r.cookies['example_cookie_name']
```

- **Send**

```
url = 'http://httpbin.org/cookies‘
cookies = dict(cookies_are='working')
r = requests.get(url, cookies=cookies)
```

Timeouts, Errors and Exceptions

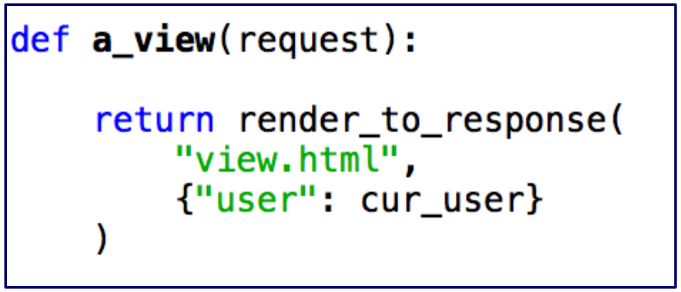
```
requests.get('http://github.com', timeout=0.001)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
requests.exceptions.Timeout:
```

- **In the event of a network problem**
  - **DNS failure,**
  - **refused connection,**
  - **etc**

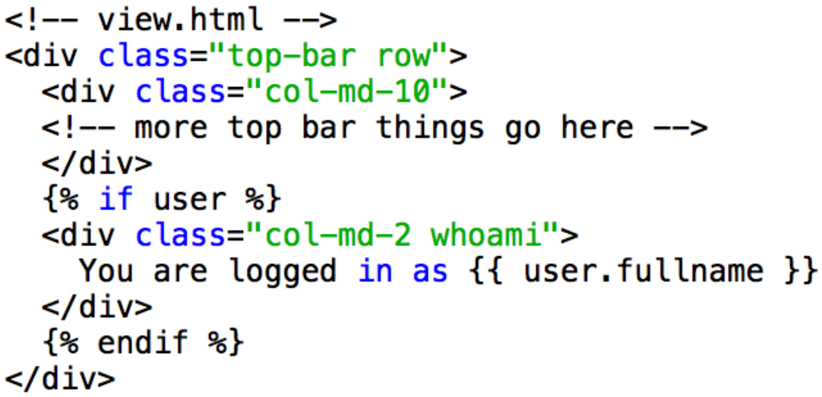
- **Requests will raise a ConnectionError** exception

Django

- https://www.djangoproject.com
- The most popular
- Templating, forms, routing, authentication, basic database administration, and more



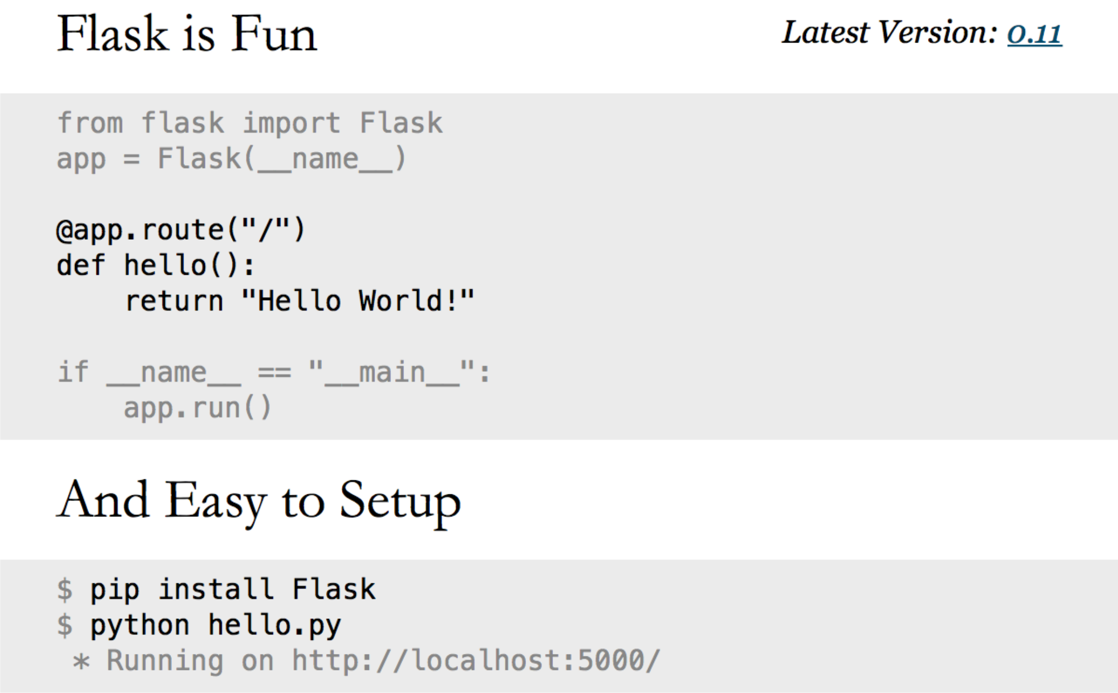
```python
def a_view(request):

    return render_to_response(
        "view.html",
        {"user": cur_user}
    )
```



```html
<!-- view.html -->
<div class="top-bar row">
  <div class="col-md-10">
  <!-- more top bar things go here -->
  </div>
  {% if user %}
  <div class="col-md-2 whoami">
    You are logged in as {{ user.fullname }}
  </div>
  {% endif %}
</div>
```

Flask

- http://flask.pocoo.org
- Microframework



Flask is Fun

Latest Version: 0.11

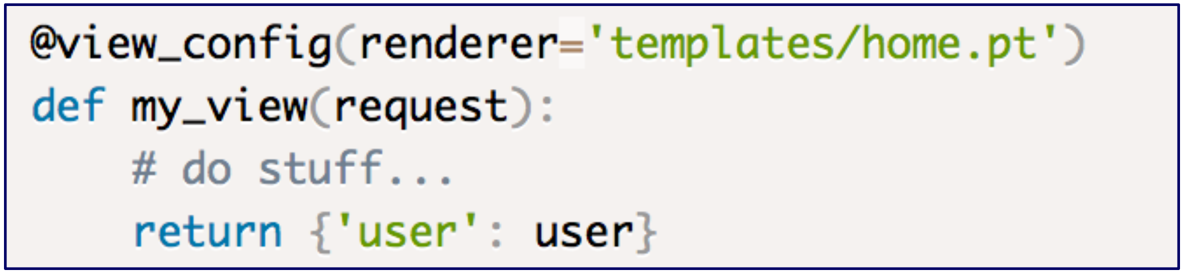
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

And Easy to Setup

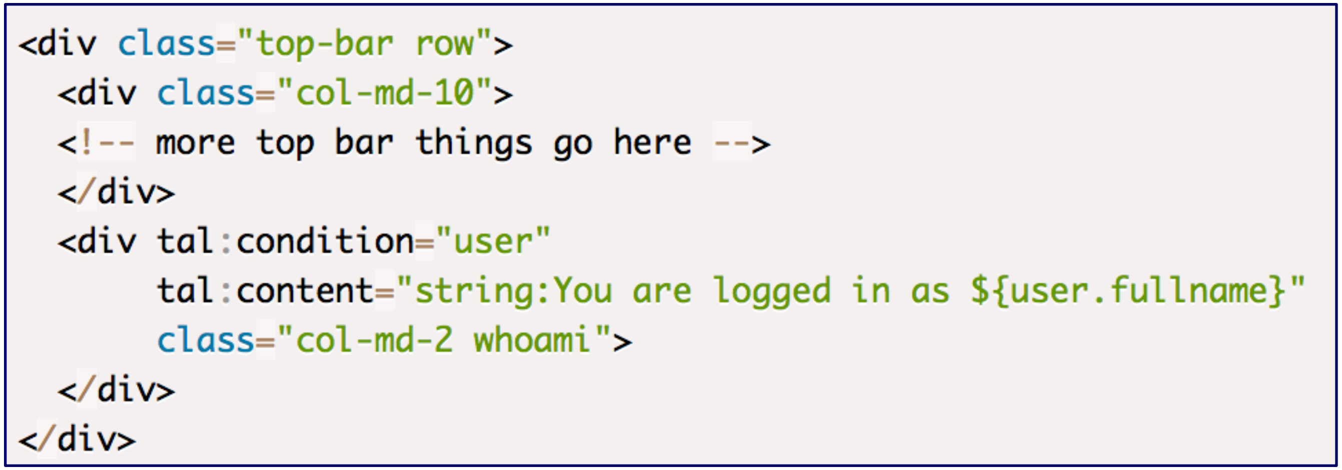
```
$ pip install Flask
$ python hello.py
 * Running on http://localhost:5000/
```

Pyramid

- **https://trypyramid.com**



```python
@view_config(renderer='templates/home.pt')
def my_view(request):
    # do stuff...
    return {'user': user}
```



```html
<div class="top-bar row">
  <div class="col-md-10">
  <!-- more top bar things go here -->
  </div>
  <div tal:condition="user"
       tal:content="string:You are logged in as ${user.fullname}"
       class="col-md-2 whoami">
  </div>
</div>
```

Introduction to GUI with Python

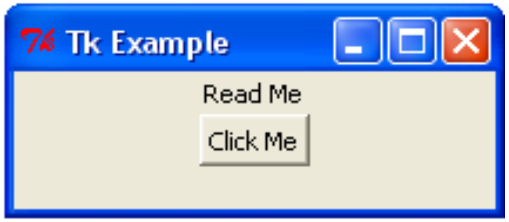
- Python supports various GUI extensions
  - X11
  - Win32
  - Macintosh
  - Gtk (X specific)
  - Tk
  - Qt
- Tk is most commonly used
  - Tkinter on Python 2, tkinter on Python 3
  - Portable across UNIX and Windows
  - Based on tcl/Tk toolkit
  - Object-oriented interface

Introduction to Tk

- Introduction to Tk
- Overview of Python Tk
- Requirements for Python Tk
  - Named arguments
  - Subroutine references
  - Closures
- Tk design
  - Event-driven
  - Widget hierarchy
  - Dynamic widget size & position

A simple example

- Create objects
  - Main window
  - Labels and buttons
  - Define call back
  - Invoke main loop



```
from Tkinter import *

def button_proc():
    print('Call-back function for button')

root = Tk()
root.title('Tk Example')
Label(root, text='Read Me').pack()
Button(root, text='Click Me',
       command=button_proc).pack()
root.mainloop()
```

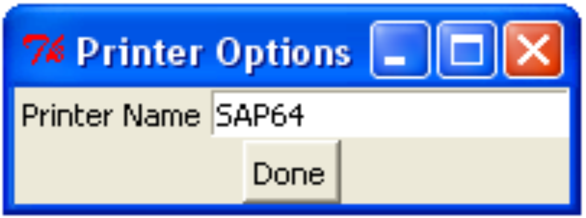
elements.py

Elements of Python Tkinter

- Event-driven
  - Create objects, then run a main loop
  - Main loop (indirectly) invokes callback subroutines
- Methods use named arguments
  - Default values exist for most arguments
- Events invoke callback subroutines
  - Reference to subroutine
  - Anonymous subroutine
  - Closure
  - Object + method name

User input is hidden

- Data entry objects handle input
  - Stored in a StringVar, IntVar variable
  - No need to query state


Printer Options

Printer Name SAP64

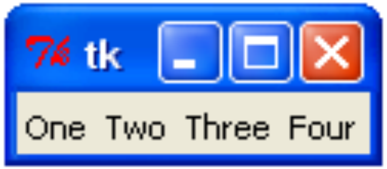
```
from Tkinter import *

def button_proc():
    print('Call-back function, printer is now',ent.get())

root = Tk()
root.title('Printer Options')

but = Button(root, text='Done', command=button_proc)
but.pack(side = 'bottom')
Label(text='Printer Name').pack(side='left')

printer = StringVar()
ent=Entry(root,textvariable=printer)
ent.pack(side='left')

printer.set('SAP64')
root.mainloop()
```

printer.py

Widgets are hierarchical

- **Widgets inside widgets inside…**
  - **Normally indicated through order of creation and choice of parent**

```
from Tkinter import *

root = Tk()

topf = Frame(root).pack()
botf = Frame(root).pack()

Label(topf,text ='One').pack(side ='left')
Label(topf,text ='Two').pack(side ='left')

Label(botf,text ='Three').pack(side ='left')
Label(botf,text ='Four' ).pack(side ='left')

root.mainloop()
```



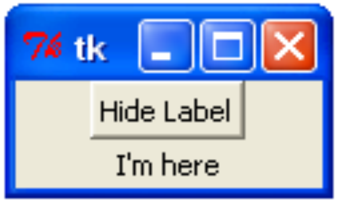
hierarchy.py

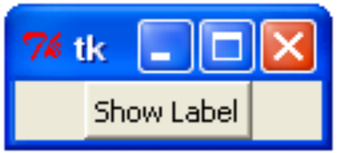
Widgets are dynamic

- Widgets may be added or removed while running
  - All widgets dynamically sized

Initial size is computed and resized when window size changes

```
from Tkinter import *

root = Tk()
button = Button(root)
button.pack()
label  = Label(root,text ="I'm here")

def hide_label():
    label.forget();
    button.configure(text ="Show Label",
                     command =show_label)

def show_label():
    label.pack()
    button.configure(text='Hide Label',
                     command=hide_label)

show_label()
root.mainloop()
```



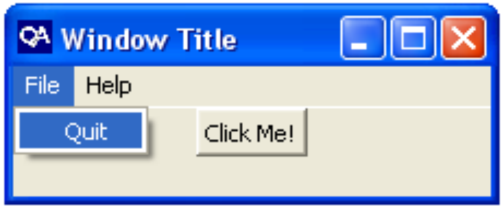


dynamic.py

Building a main window

- A main window also requires title, icon, and menus

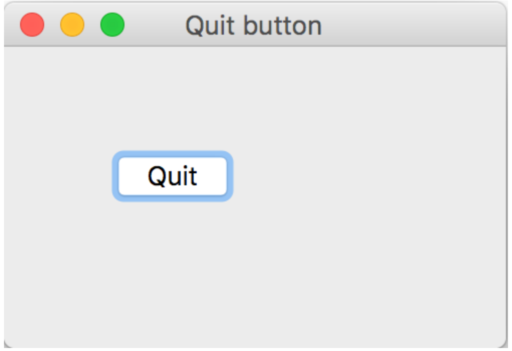
```
from Tkinter import *
root = Tk()
root.title('Window Title')
root.wm_iconbitmap('qa.ico')

mbar = Menu(root)
filemenu = Menu(mbar, tearoff=0)
filemenu.add_command(label ="Quit",
                     command =lambda: root.destroy())
helpmenu = Menu(mbar, tearoff=0)
helpmenu.add_command(label='RTFM')

mbar.add_cascade(label='File',menu=filemenu)
mbar.add_cascade(label='Help',menu=helpmenu)
root.config(menu=mbar)

bt = Button(root,text ='Click Me!',command=button_proc)
bt.pack()

root.mainloop()
```



mainwindow.py

PyQT Example

```
import sys
from PyQt5.QtWidgets import QWidget, QPushButton,
QApplication
from PyQt5.QtCore import QCoreApplication

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
    def initUI(self):
        qbtn = QPushButton('Quit', self)
        qbtn.clicked.connect(QCoreApplication.instance().quit)
        qbtn.resize(qbtn.sizeHint())
        qbtn.move(50, 50)
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('Quit button')
        self.show()


if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())
```



Mobile Applications

- https://kivy.org/#home
- https://www.blender.org
- http://pyzia.com
- …

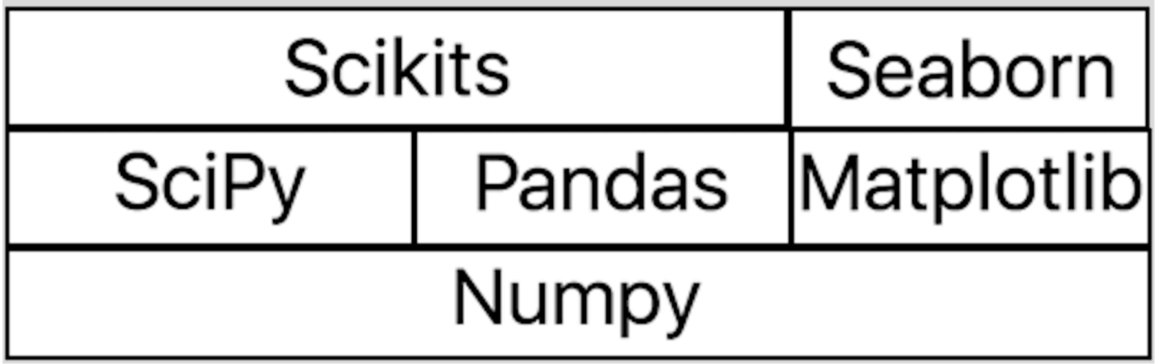
Summary

- TK/QT is portable
  - Normally bundled with the base
- Widgets are hierarchical
- Widgets are dynamic
- Start with a main window

Embedded and IOT

- It is very easy to integrate python interpreter on OS based Embedded Systems
  - Embedded Linux
  - Android
- IOT
  - Zerynth
  - MicroPython
  - PyMCU

Data Packages



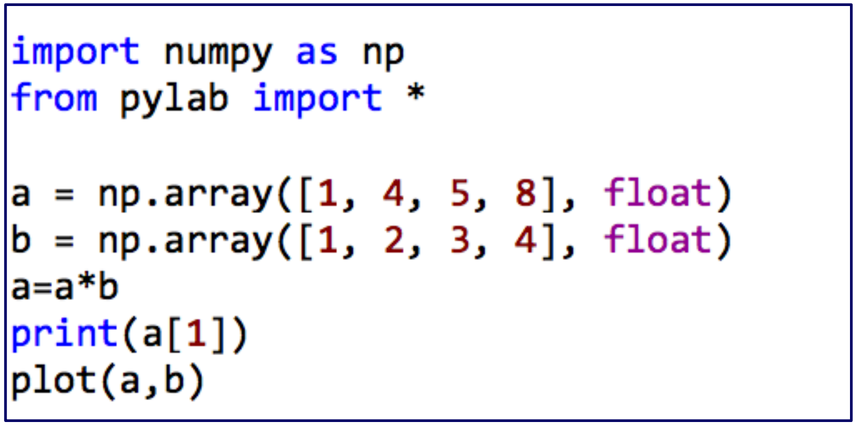
Scikits
Seaborn
SciPy
Pandas
Matplotlib
Numpy

NumPy Overview

- Python is a fabulous language
  - Easy to extend
  - Great syntax which encourages easy to write and maintain code
  - Incredibly large standard-library and third-party tools
- No built-in multi-dimensional array (but it supports the needed syntax for extracting elements from one)
- NumPy provides a fast built-in object (ndarray) which is a multi-dimensional array of a homogeneous data-type.
- https://www.scipy.org

N-D Array

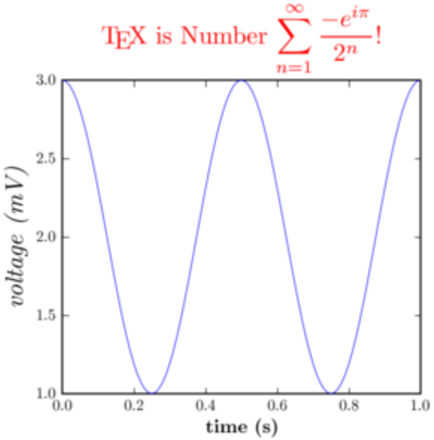
- N-dimensional array of rectangular data
- Element of the array can be C-structure or simple data-type.
- Fast algorithms on machine data-types (int, float, etc.)



```
import numpy as np
from pylab import *

a = np.array([1, 4, 5, 8], float)
b = np.array([1, 2, 3, 4], float)
a=a*b
print(a[1])
plot(a,b)
```

Matplotlib

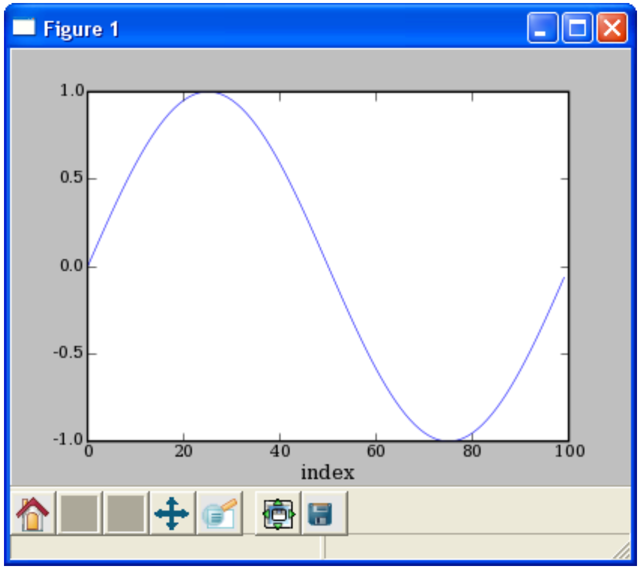
- Requires NumPy extension. Provides powerful plotting commands.
- http://matplotlib.sourceforge.net



Line Plots

PLOT AGAINST INDICES

```
>>> x = arange(50)*2*pi/50.
>>> y = sin(x)
>>> plot(y)
>>> xlabel('index')
```



MULTIPLE DATA SETS

```
>>> plot(x,y,x2,y2)
>>> xlabel('radians')
```

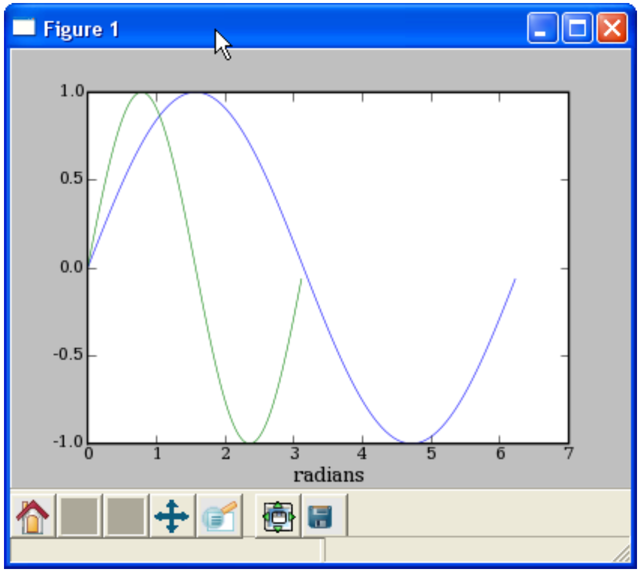
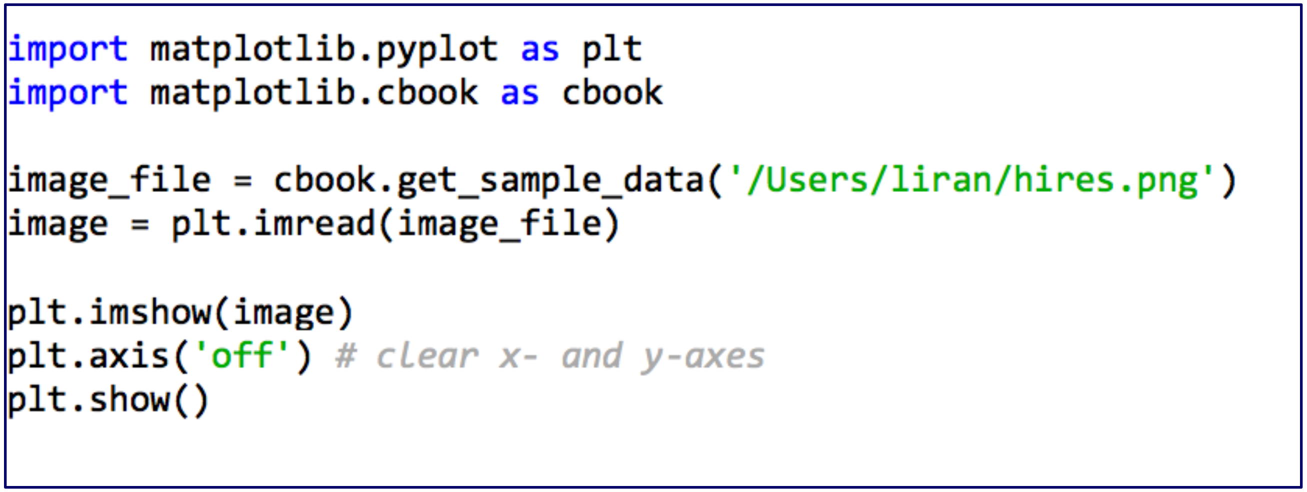
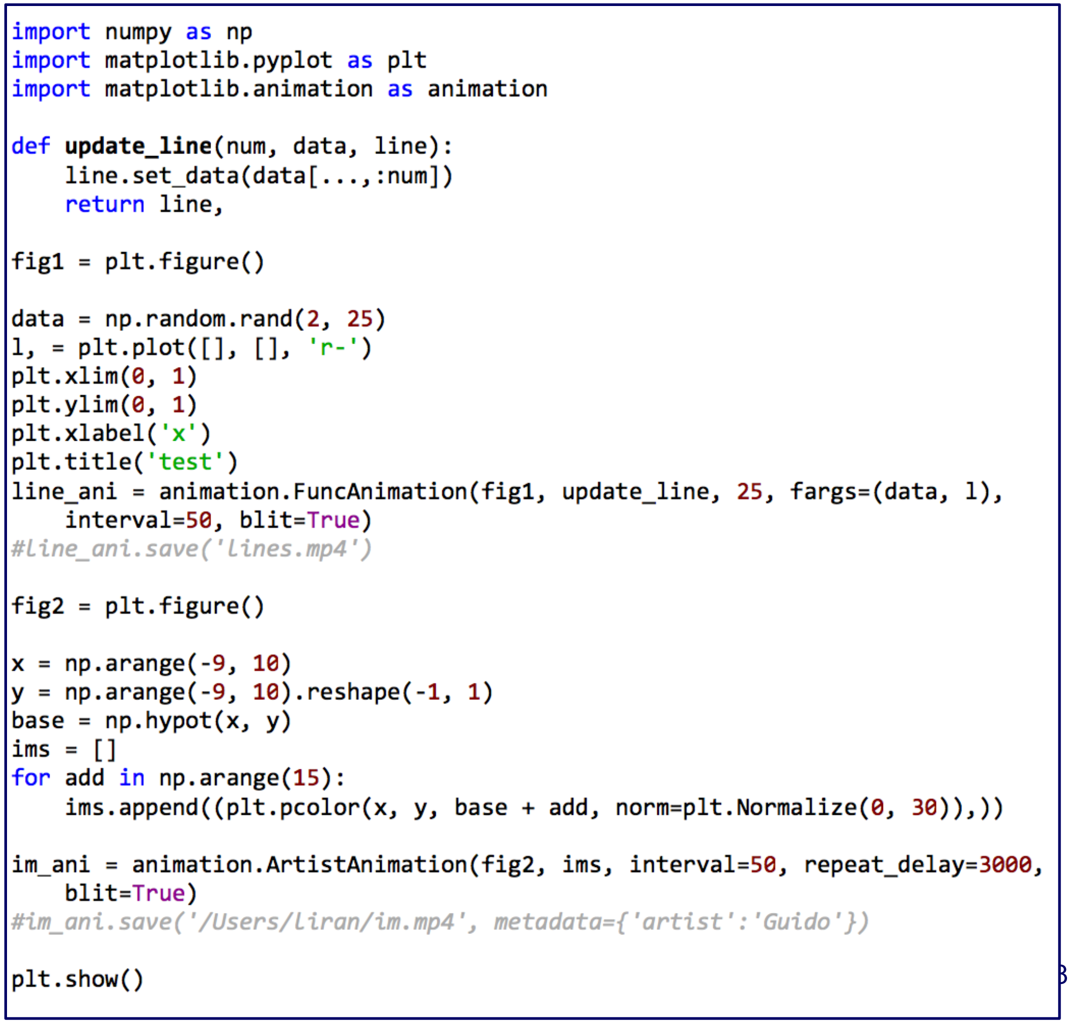


Image demo



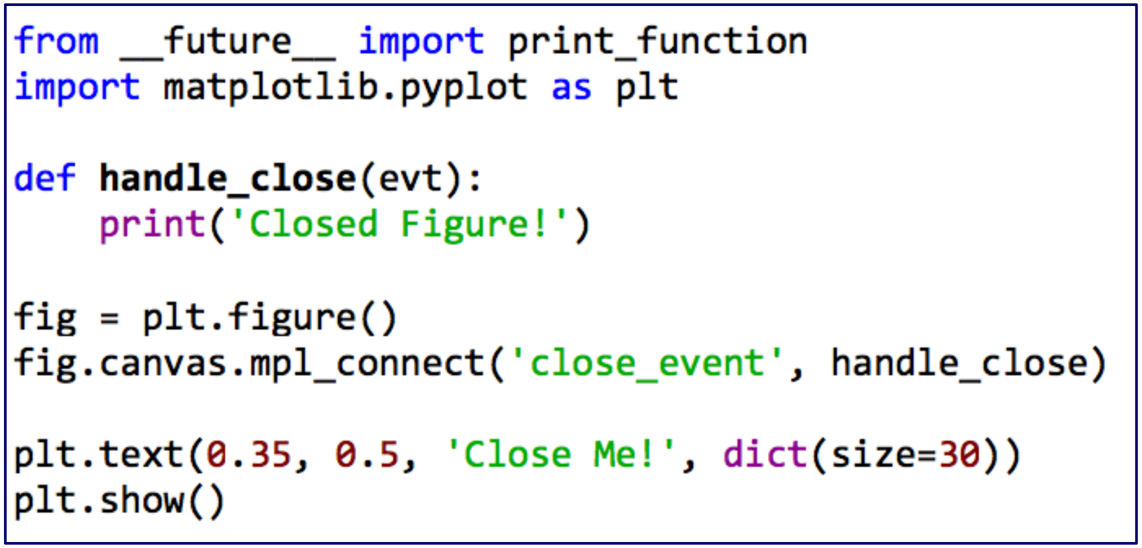
```
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook

image_file = cbook.get_sample_data('/Users/liran/hires.png')
image = plt.imread(image_file)

plt.imshow(image)
plt.axis('off') # clear x- and y-axes
plt.show()
```

Animation



```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

def update_line(num, data, line):
    line.set_data(data[...,:num])
    return line,

fig1 = plt.figure()

data = np.random.rand(2, 25)
l, = plt.plot([], [], 'r-')
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel('x')
plt.title('test')
line_ani = animation.FuncAnimation(fig1, update_line, 25, fargs=(data, l),
    interval=50, blit=True)
#line_ani.save('lines.mp4')

fig2 = plt.figure()

x = np.arange(-9, 10)
y = np.arange(-9, 10).reshape(-1, 1)
base = np.hypot(x, y)
ims = []
for add in np.arange(15):
    ims.append((plt.pcolor(x, y, base + add, norm=plt.Normalize(0, 30)),))

im_ani = animation.ArtistAnimation(fig2, ims, interval=50, repeat_delay=3000,
    blit=True)
#im_ani.save('/Users/liran/im.mp4', metadata={'artist':'Guido'})

plt.show()
```

Events



```python
from __future__ import print_function
import matplotlib.pyplot as plt

def handle_close(evt):
    print('Closed Figure!')

fig = plt.figure()
fig.canvas.mpl_connect('close_event', handle_close)

plt.text(0.35, 0.5, 'Close Me!', dict(size=30))
plt.show()
```

SciPy Overview

- Available at www.scipy.org
- Open Source BSD Style License
- Over 30 svn “committers” to the project

CURRENT PACKAGES

- Special Functions (scipy.special)
- Signal Processing (scipy.signal)
- Image Processing (scipy.ndimage)
- Fourier Transforms (scipy.fftpack)
- Optimization (scipy.optimize)
- Numerical Integration (scipy.integrate)
- Linear Algebra (scipy.linalg)

- Input/Output (scipy.io)
- Statistics (scipy.stats)
- Fast Execution (scipy.weave)
- Clustering Algorithms (scipy.cluster)
- Sparse Matrices (scipy.sparse)
- Interpolation (scipy.interpolate)
- More (e.g. scipy.odr, scipy.maxentropy)

1D Spline Interpolation

```
from scipy.interpolate import interp1d
from pylab import plot, axis, legend
from numpy import linspace

# sample values
x = linspace(0,2*pi,6)
y = sin(x)

# Create a spline class for interpolation.
# kind=5 sets to 5th degree spline.
# kind=0 -> zeroth order hold.
# kind=1 or ‘linear’ -> linear interpolation
# kind=2 or
spline_fit = interp1d(x,y,kind=5)
xx = linspace(0,2*pi, 50)
yy = spline_fit(xx)

# display the results.
plot(xx, sin(xx), 'r-', x,y,'ro',xx,yy, 'b--',linewidth=2)
axis('tight')
legend(['actual sin', 'original samples', 'interpolated curve'])
```

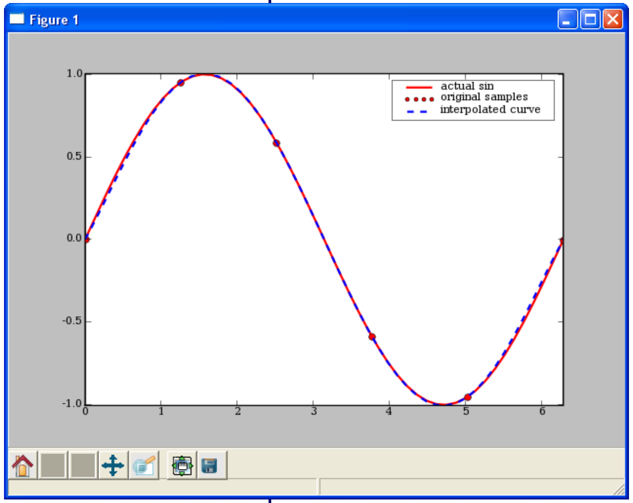
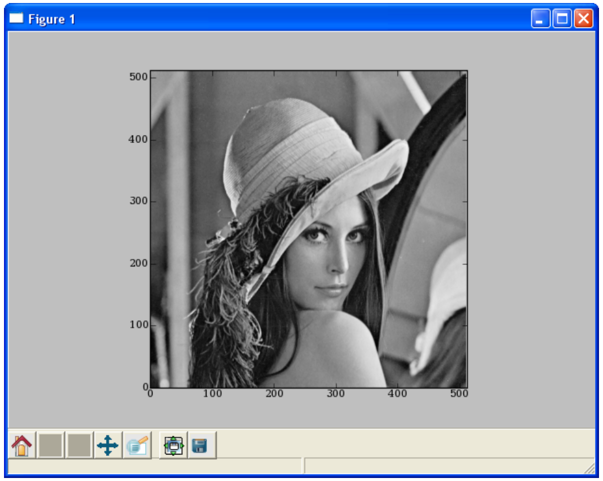


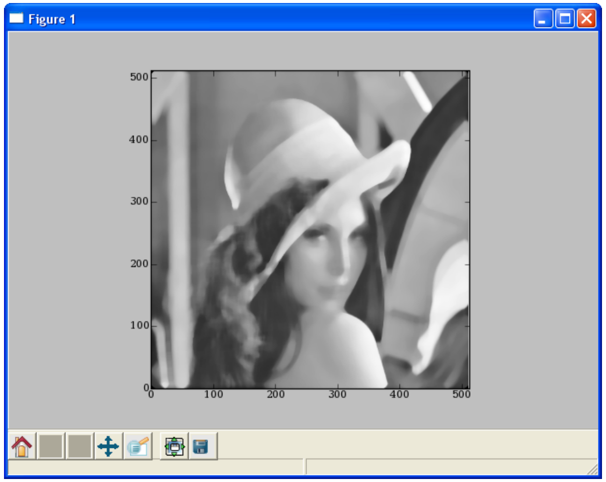
Image Processing

```
# The famous lena image is packaged with scipy
>>> from scipy import misc.lena, signal
>>> lena = lena().astype(float32)
>>> imshow(lena, cmap=cm.gray)
# Blurring using a median filter
>>> fl = signal.medfilt2d(lena, [15,15])
>>> imshow(fl, cmap=cm.gray)
```

LENA IMAGE



MEDIAN FILTERED IMAGE



Pandas

- Python Library to provide data analysis features similar to : R, MATLB, SAS

- Rich data structures and functions to make working with data structure fast, easy and expressive.

- It is built on top of NumPy which provides it agility

- Key components provided by Pandas :

  Two new data structures to Python

  - Series
  - DataFrame

Pandas is well suited for:

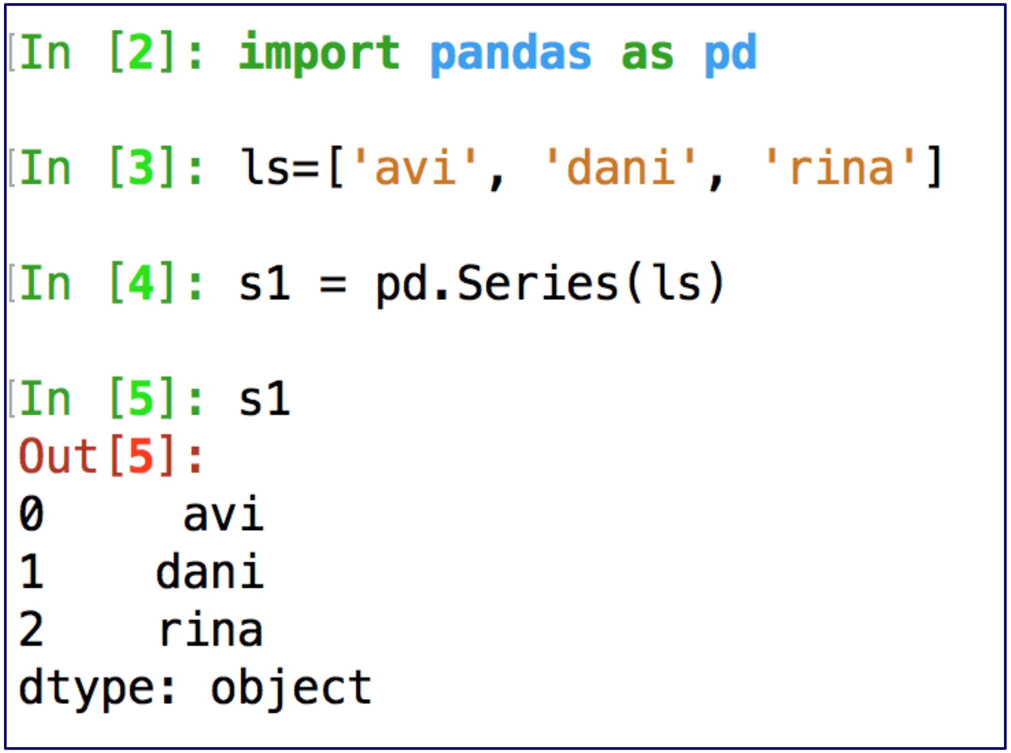
- Tabular Data (SQL table & Excel spreadsheet)
- Ordered and Unordered Time Series Data
- Arbitrary Matrix Data
- Any other form of observational / statistical data sets

Series

- One dimensional array like object like an array or list
- It contains array of data (of any NumPy data type) with associated indexes.
- By default ,

  the series will get indexing from 0 to N where N = size -1

Series



```
In [2]: import pandas as pd

In [3]: ls=['avi', 'dani', 'rina']

In [4]: s1 = pd.Series(ls)

In [5]: s1
Out[5]:
0     avi
1    dani
2    rina
dtype: object
```

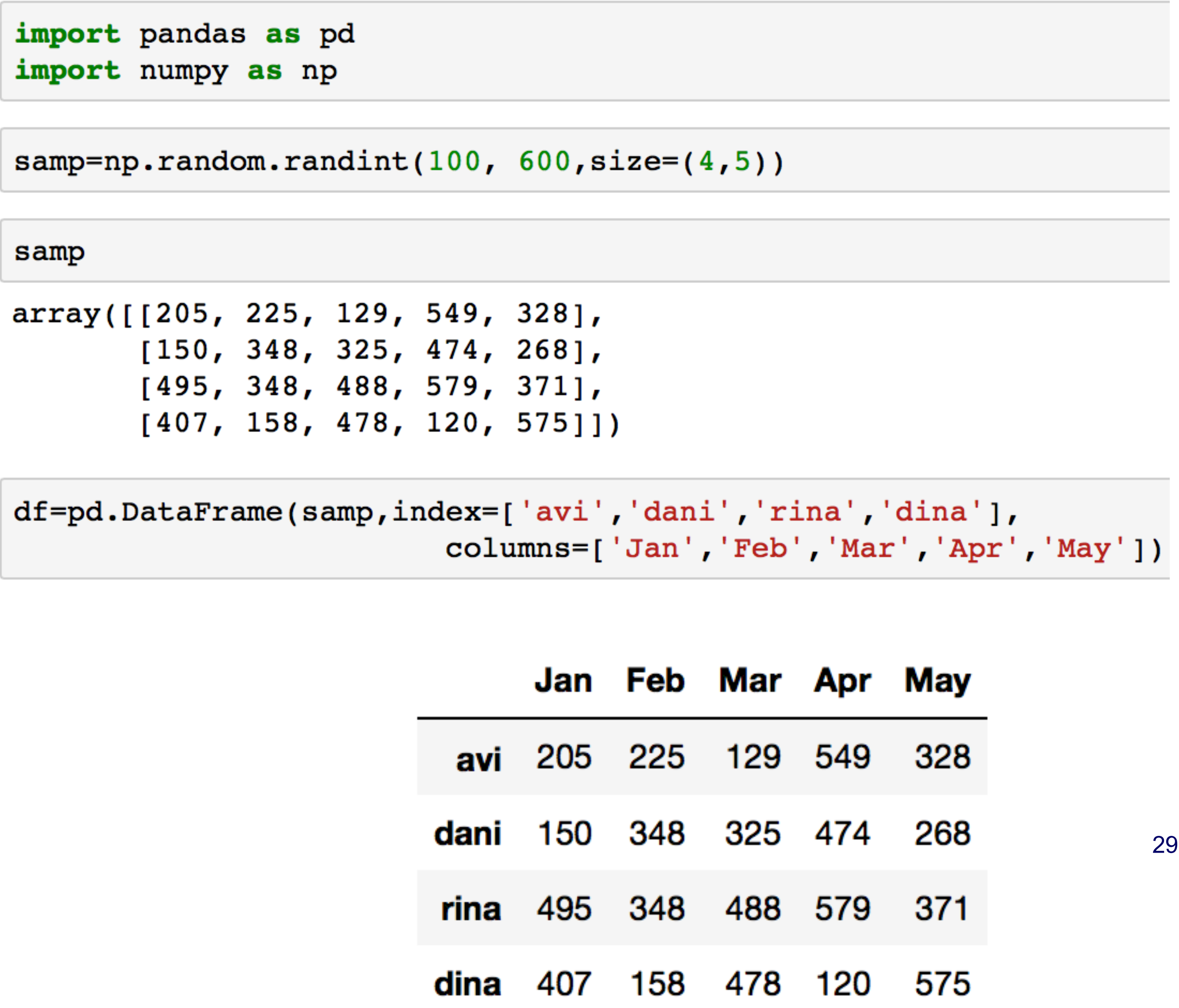
Dataframe

- A dataFrame is a tabular data structure comprised of rows and columns, akin to a spreadsheet or database table.
- It can be treated as a series of objects sharing common index

Operations

- Filtering
- Summarizing
- Group by – split apply combine
- Merge, join, aggregate
- Time series/ Data functionality
- Plotting with Matplotlib and many more…

DataFrame From NumPy Array



```
import pandas as pd
import numpy as np
```

```
samp=np.random.randint(100, 600,size=(4,5))
```

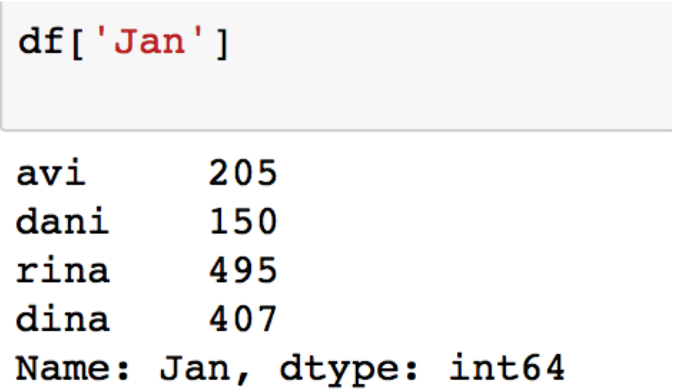
```
samp
```

```
array([[205, 225, 129, 549, 328],
       [150, 348, 325, 474, 268],
       [495, 348, 488, 579, 371],
       [407, 158, 478, 120, 575]])
```

```
df=pd.DataFrame(samp,index=['avi','dani','rina','dina'],
                        columns=['Jan','Feb','Mar','Apr','May'])
```

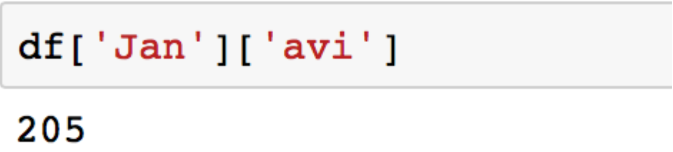
	Jan	Feb	Mar	Apr	May
avi	205	225	129	549	328
dani	150	348	325	474	268
rina	495	348	488	579	371
dina	407	158	478	120	575

Selecting and Indexing



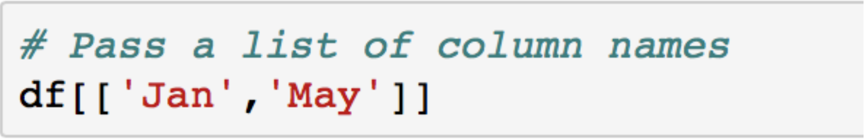
```
df['Jan']
```

```
avi     205
dani    150
rina    495
dina    407
Name: Jan, dtype: int64
```

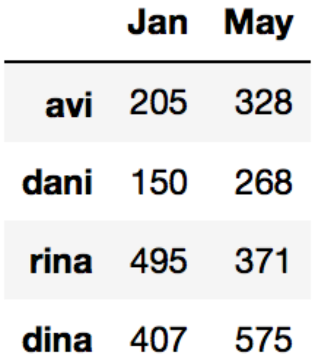


```
df['Jan']['avi']
```

```
205
```

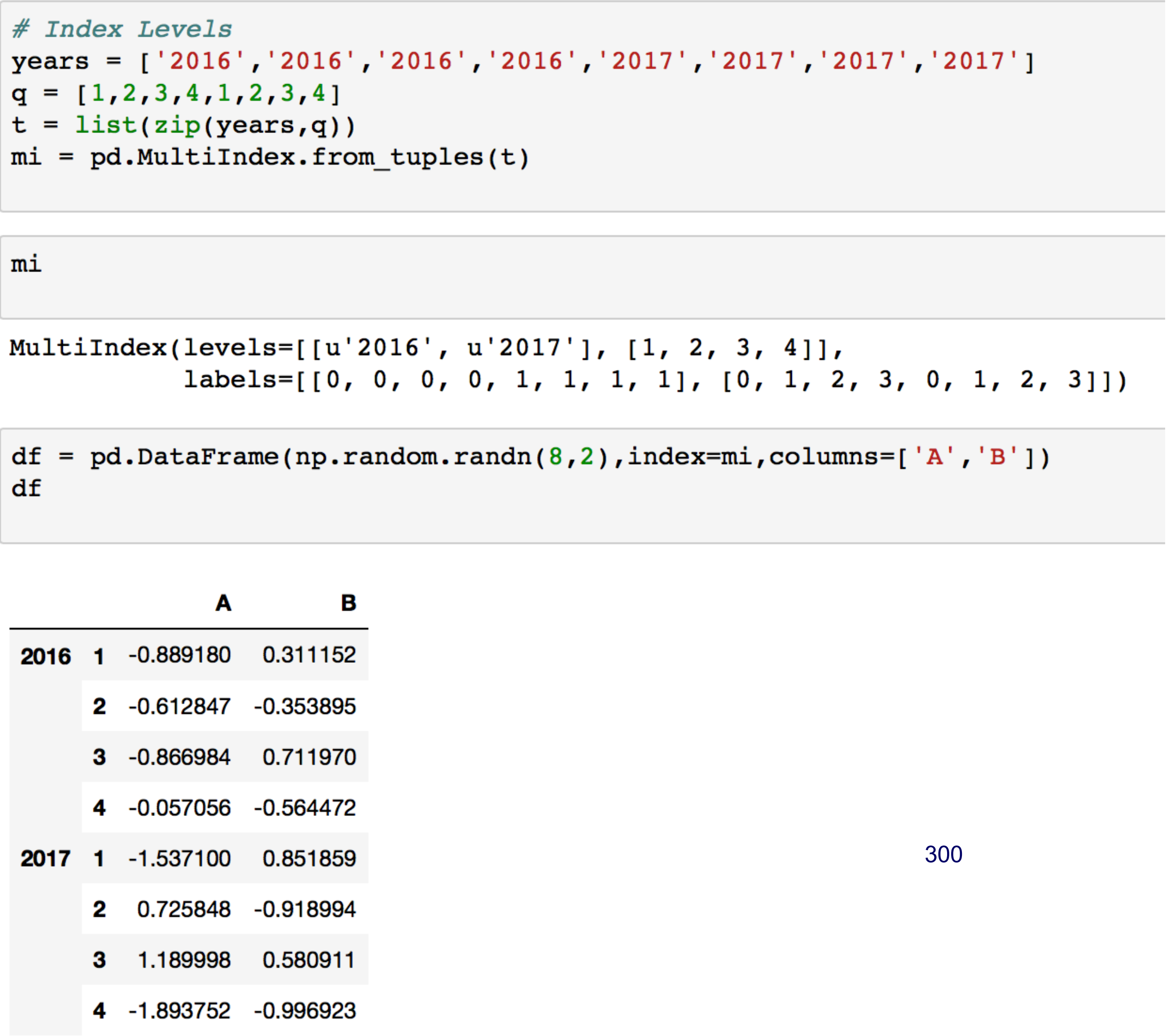


```
# Pass a list of column names
df[['Jan','May']]
```



	Jan	May
avi	205	328
dani	150	268
rina	495	371
dina	407	575

Multi-index



```
# Index Levels
years = ['2016','2016','2016','2016','2017','2017','2017','2017']
q = [1,2,3,4,1,2,3,4]
t = list(zip(years,q))
mi = pd.MultiIndex.from_tuples(t)
```

```
mi
```

```
MultiIndex(levels=[[u'2016', u'2017'], [1, 2, 3, 4]],
           labels=[[0, 0, 0, 0, 1, 1, 1, 1], [0, 1, 2, 3, 0, 1, 2, 3]])
```

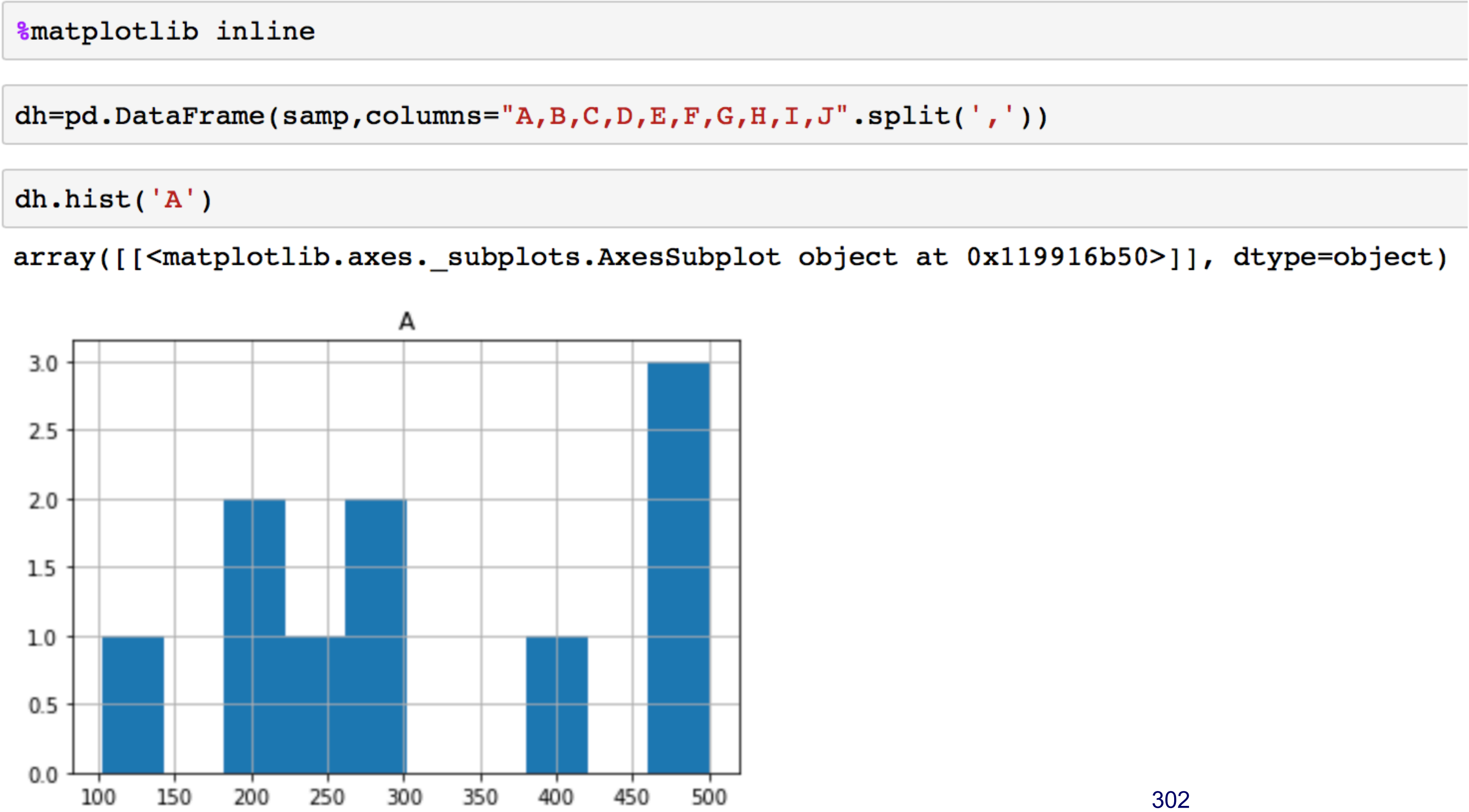
```
df = pd.DataFrame(np.random.randn(8,2),index=mi,columns=['A','B'])
df
```

		A	B
2016	**1**	-0.889180	0.311152
	2	-0.612847	-0.353895
	3	-0.866984	0.711970
	4	-0.057056	-0.564472
2017	**1**	-1.537100	0.851859
	2	0.725848	-0.918994
	3	1.189998	0.580911
	4	-1.893752	-0.996923

Working With Files

- pd.read_csv(’cust.csv')
- df.to_csv(cust.csv',index=False)
- pd.read_excel(‘samp.xslx’)
- df.to_excel(’samp.xlsx',sheet_name=’cust')
- Also supported:
  - HTML
  - JSON
  - …

Visualization



```
%matplotlib inline
```

```
dh=pd.DataFrame(samp,columns="A,B,C,D,E,F,G,H,I,J".split(','))
```

```
dh.hist('A')
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x119916b50>]], dtype=object)
```

SQL Servers

- The pandas.io.sql module provides a collection of query wrappers to both facilitate data retrieval and to reduce dependency on DB-specific API.
- Database abstraction is provided by SQLAlchemy if installed.
- In addition you will need a driver library for your database.
- Examples of such drivers are
  - psycopg2 for PostgreSQL
  - pymysql for MySQL.
  - SQLite - included in Python’s standard library by default.

SQL - Function

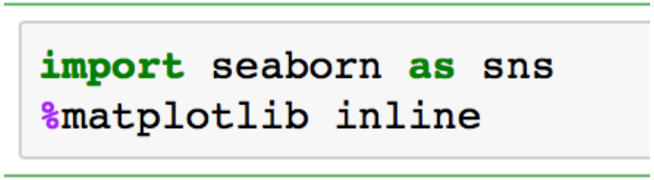
- read_sql_table(table_name, con[, schema, ...])
  - Read SQL database table into a DataFrame.
- read_sql_query(sql, con[, index_col, ...])
  - Read SQL query into a DataFrame.
- read_sql(sql, con[, index_col, ...])
  - Read SQL query or database table into a DataFrame.
- DataFrame.to_sql(name, con[, flavor, ...])
  - Write records stored in a DataFrame to a SQL database

Seaborn

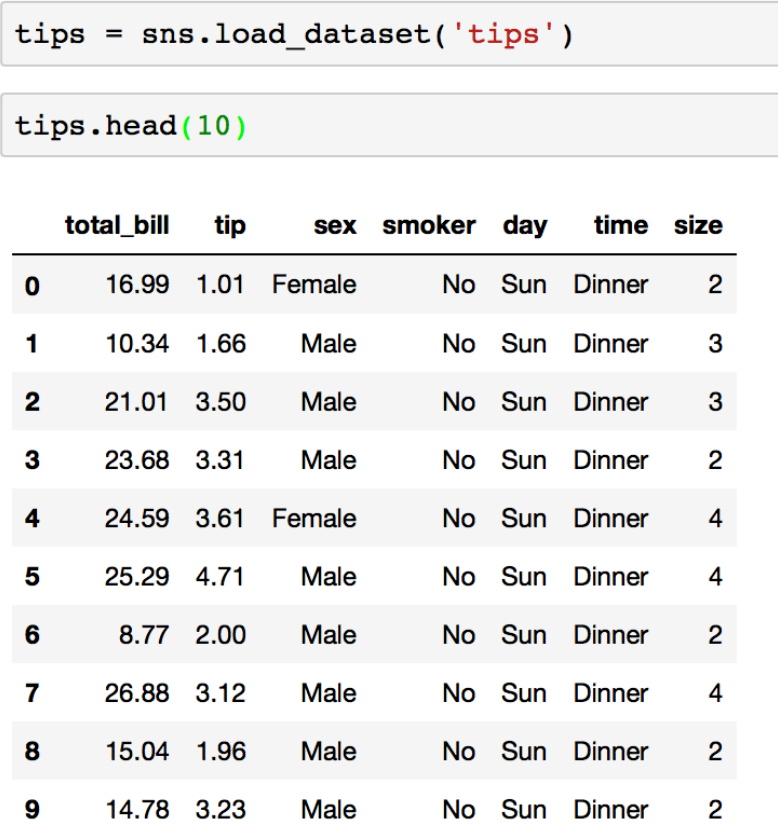
- Statistical plotting library
- Great styles
- Works great with NumPy arrays and Pandas Dataframes

Seaborn

- **Some built in datasets**



```
import seaborn as sns
%matplotlib inline
```

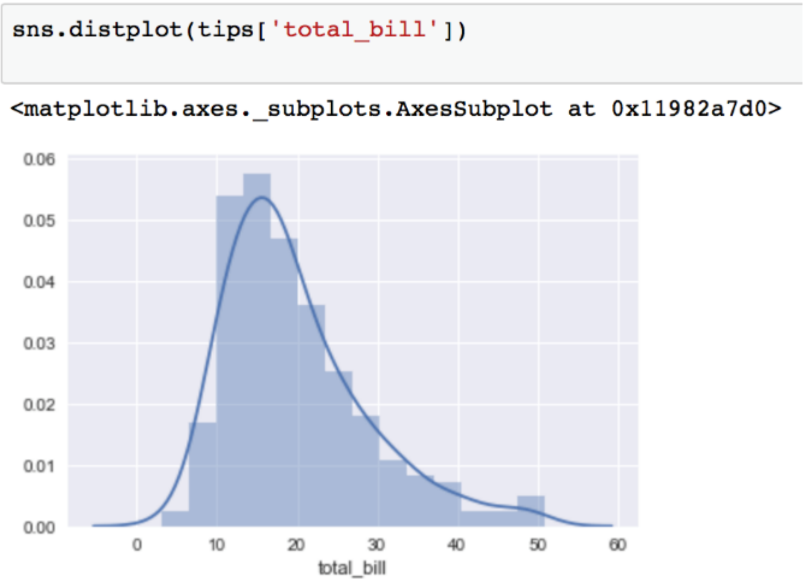


```
tips = sns.load_dataset('tips')
```

```
tips.head(10)
```

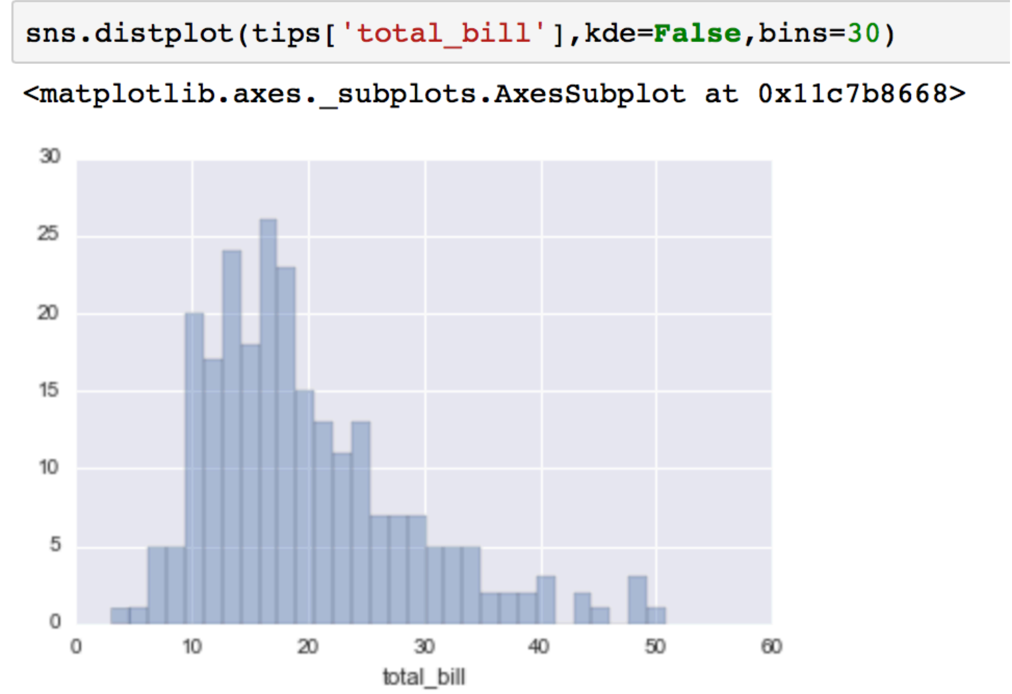
	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2

Distribution Plot



```
sns.distplot(tips['total_bill'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11982a7d0>
```

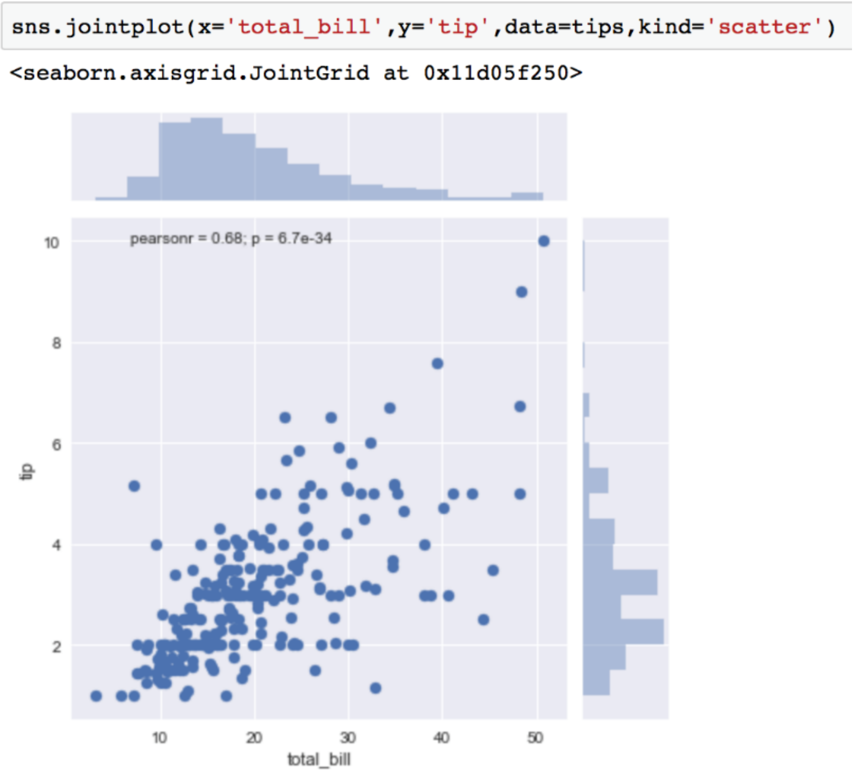


```
sns.distplot(tips['total_bill'],kde=False,bins=30)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x11c7b8668>
```

JointPlot

- **jointplot() allows you to basically match up two distplots() for bivariate data. With your choice of what kind parameter to compare with:**
  - **scatter**
  - **reg**
  - **resid**
  - **kde**
  - **hex**

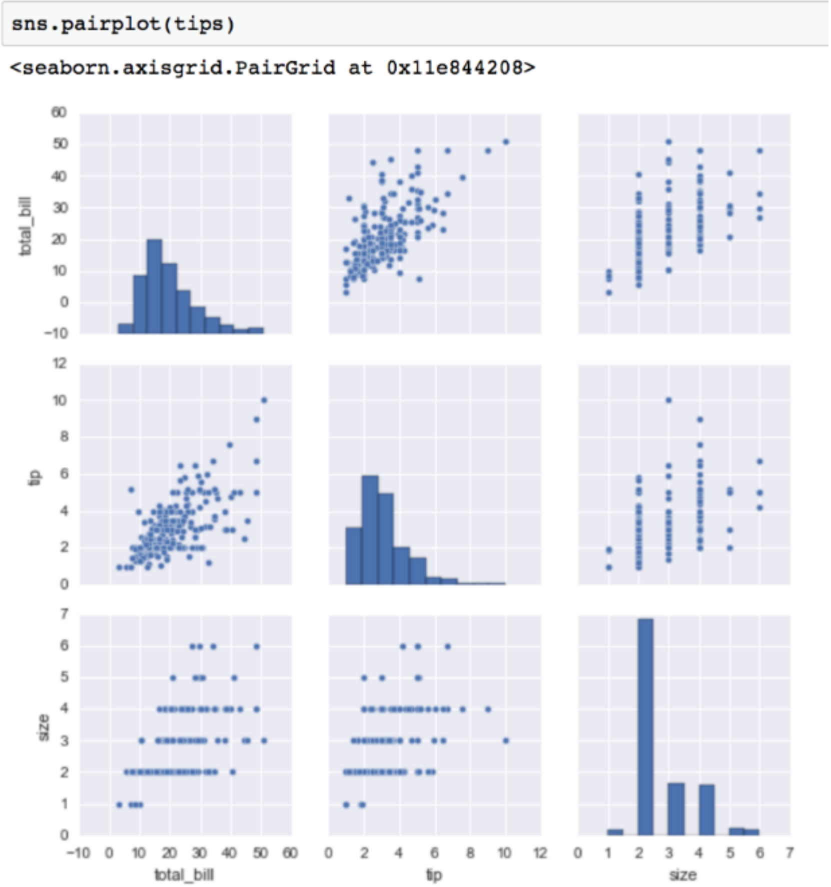


```
sns.jointplot(x='total_bill',y='tip',data=tips,kind='scatter')
```

```
<seaborn.axisgrid.JointGrid at 0x11d05f250>
```

Pairplot

- pairplot will plot pairwise relationships across an entire dataframe (for the numerical columns)
- supports a color hue argument (for categorical columns)



```
sns.pairplot(tips)
```

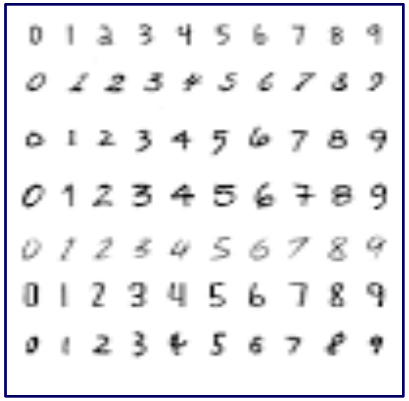
```
<seaborn.axisgrid.PairGrid at 0x11e844208>
```

Machine Learning with Python

- Overview
  - Machine Learning is the science of programming computers so they can *learn from data*
  - *“field of study that gives computers the ability to learn without being explicitly programmed”* (Arthur Samuel, 1959)
  - Example: spam filter - Machine Learning program that can learn to flag spam given examples of spam emails

Why Learn?

- Learn it when you can’t code it
  - Complex tasks where deterministic solution don’t suffice
  - e.g. speech recognition, handwriting recognition



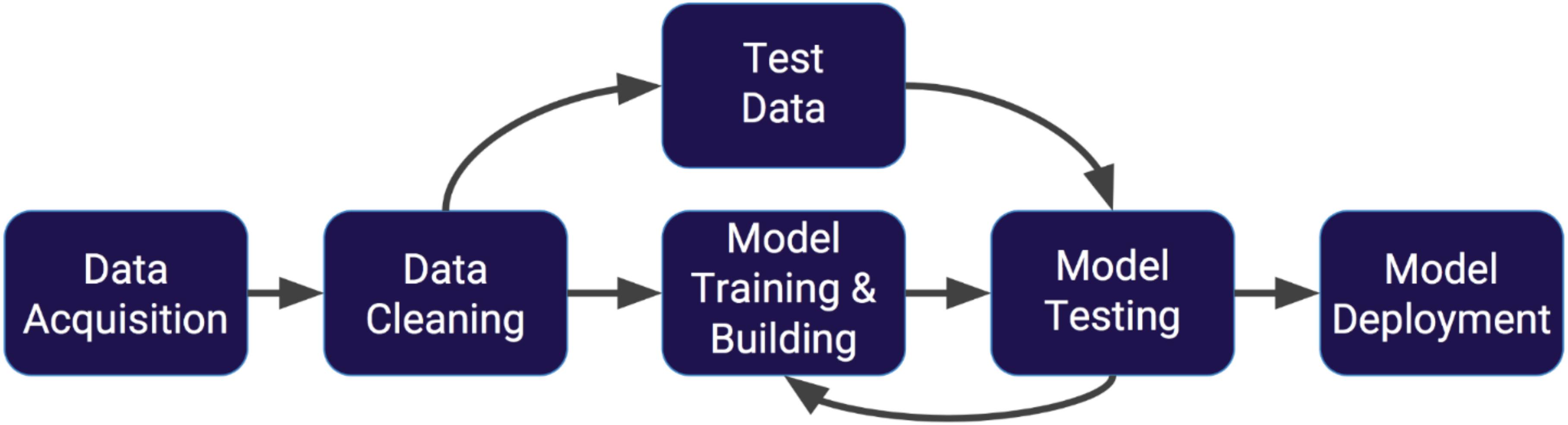
- Learn it when you can’t scale it
  - Repetitive task needing human-like expertise (e.g. recommendations, spam & fraud detection)
  - Speed, scale of data, number of data points

- Learn it when you need to adapt/personalize
  - e.g., personalized product recommendations, stock predictions

Applications

- Fraud detection.
- Web search results.
- Real-time ads on web pages
- Credit scoring and next-best offers.
- Prediction of equipment failures.
- New pricing models.
- Network intrusion detection.
- Recommendation Engines
- Customer Segmentation
- Text Sentiment Analysis
- Predicting Customer Churn
- Pattern and image recognition.
- Email spam filtering.
- Financial Modeling

Machine Learning Process

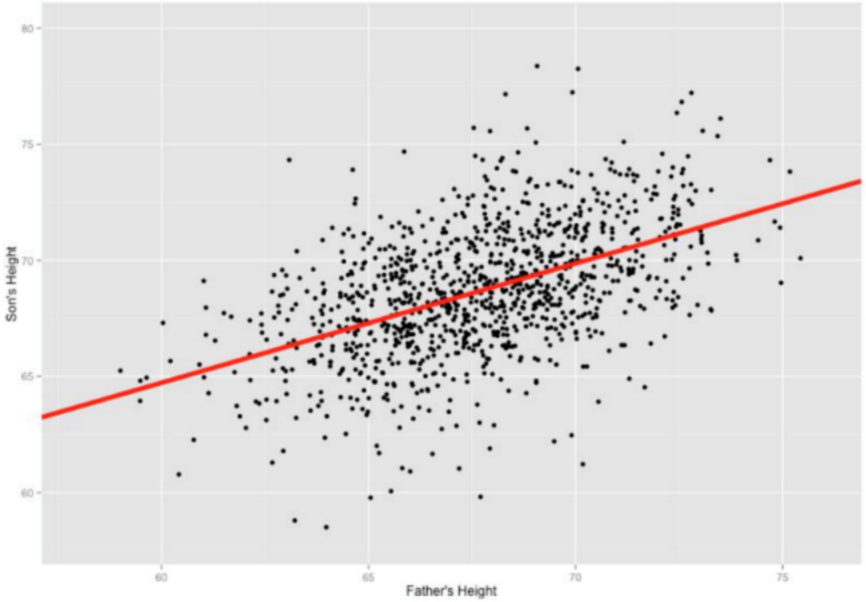


Scikit Learn

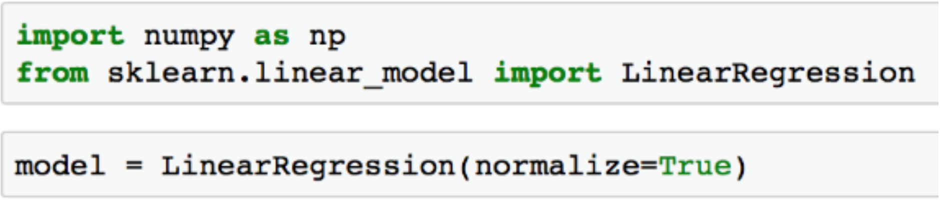
- Every algorithm is exposed in scikit-learn via an "Estimator"
- import the algorithm:
  - from sklearn.linear_model import LinearRegression
- The process for each algorithm depends on its type

Linear Regression

- **Our goal with linear regression is to minimize the vertical distance between all the data points and our line.**
- **So in determining the best line, we are attempting to minimize the distance between all the points and their distance to our line.**

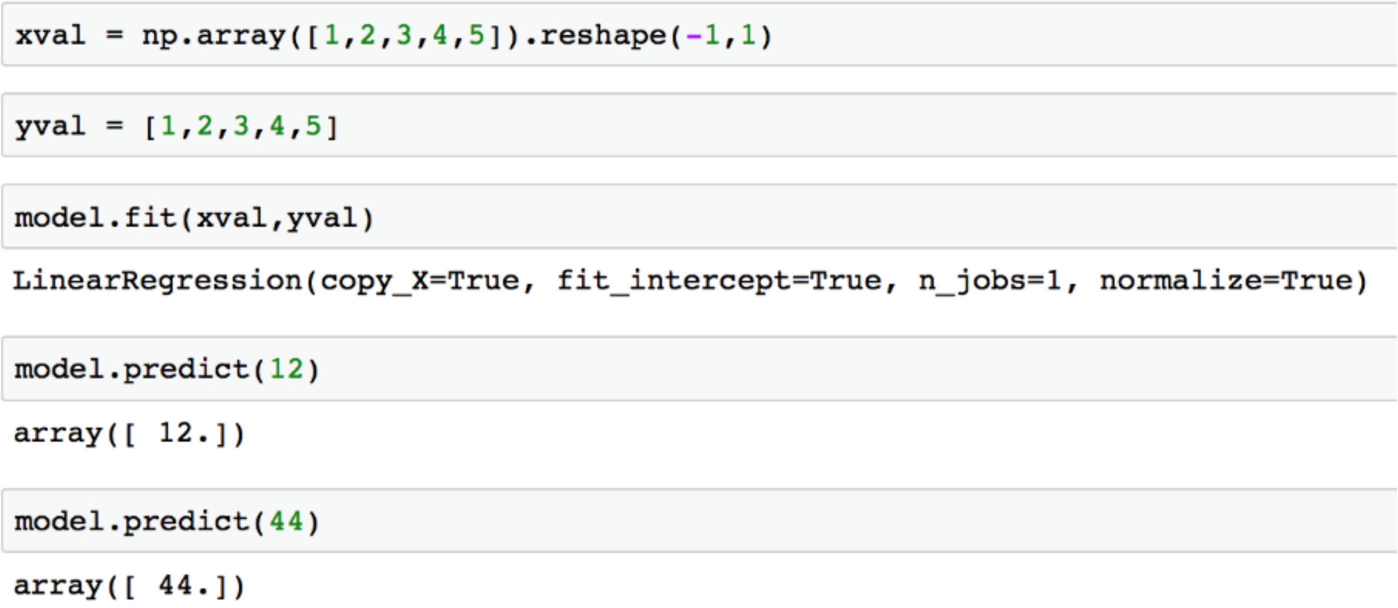


Simple Example - Linear Regression



```
import numpy as np
from sklearn.linear_model import LinearRegression
```

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



```
xval = np.array([1,2,3,4,5]).reshape(-1,1)
```

```
yval = [1,2,3,4,5]
```

```
model.fit(xval,yval)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
```

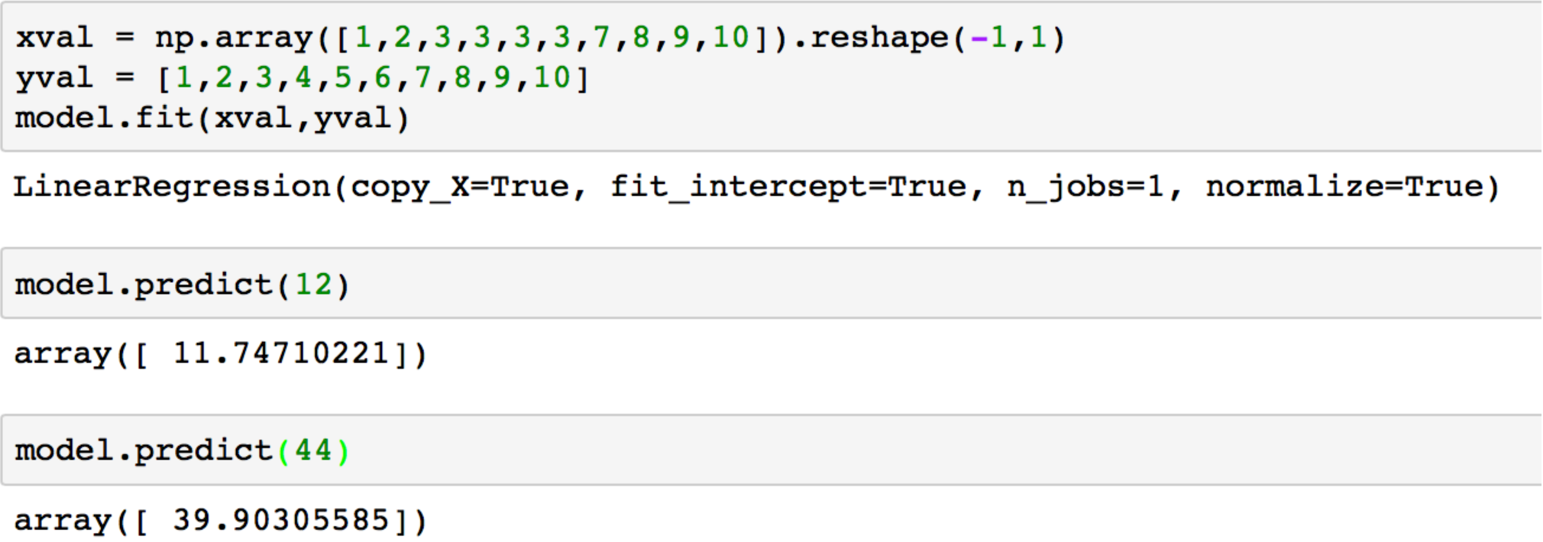
```
model.predict(12)
```

```
array([ 12.])
```

```
model.predict(44)
```

```
array([ 44.])
```

Example



```
xval = np.array([1,2,3,3,3,3,7,8,9,10]).reshape(-1,1)
yval = [1,2,3,4,5,6,7,8,9,10]
model.fit(xval,yval)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=True)
```

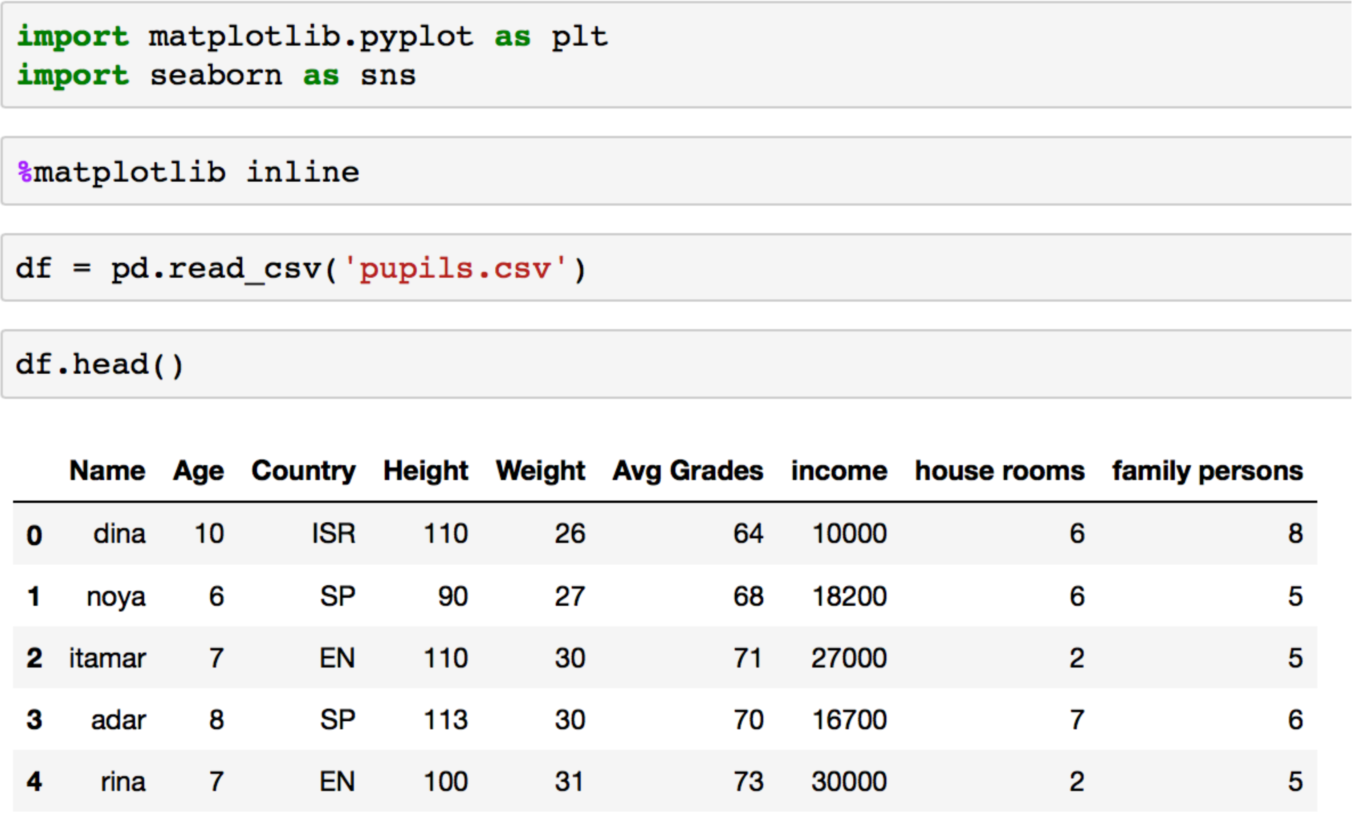
```
model.predict(12)
```

```
array([ 11.74710221])
```

```
model.predict(44)
```

```
array([ 39.90305585])
```

Using Datasets



```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
%matplotlib inline
```

```
df = pd.read_csv('pupils.csv')
```

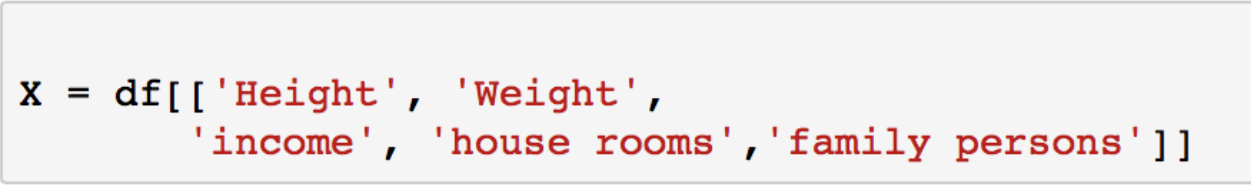
```
df.head()
```

	Name	Age	Country	Height	Weight	Avg Grades	income	house rooms	family persons
0	dina	10	ISR	110	26	64	10000	6	8
1	noya	6	SP	90	27	68	18200	6	5
2	itamar	7	EN	110	30	71	27000	2	5
3	adar	8	SP	113	30	70	16700	7	6
4	rina	7	EN	100	31	73	30000	2	5

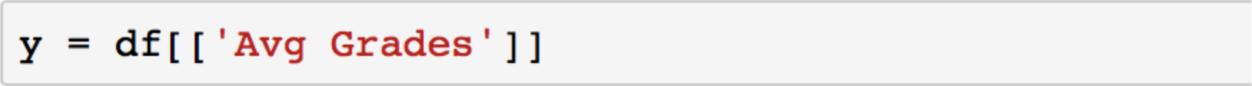
Pupils data - we want to predict average grades

Build the model

- **First we need to define our features and the target we want to predict**



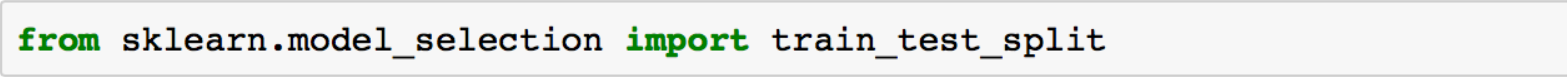
```
X = df[['Height', 'Weight',
       'income', 'house rooms','family persons']]
```



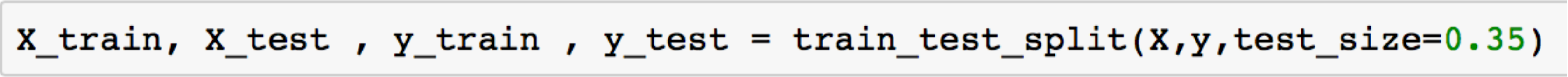
```
y = df[['Avg Grades']]
```

Test our Model

- We have a data and we want to build a model to predict targets
- How do we know that the model (algorithm) is working
- The solution is to split the data – Train and Test
  - For example 70% train and 30% test
- Then run the model on the Train data, and then test on the Test and see if the results are close to the real values.

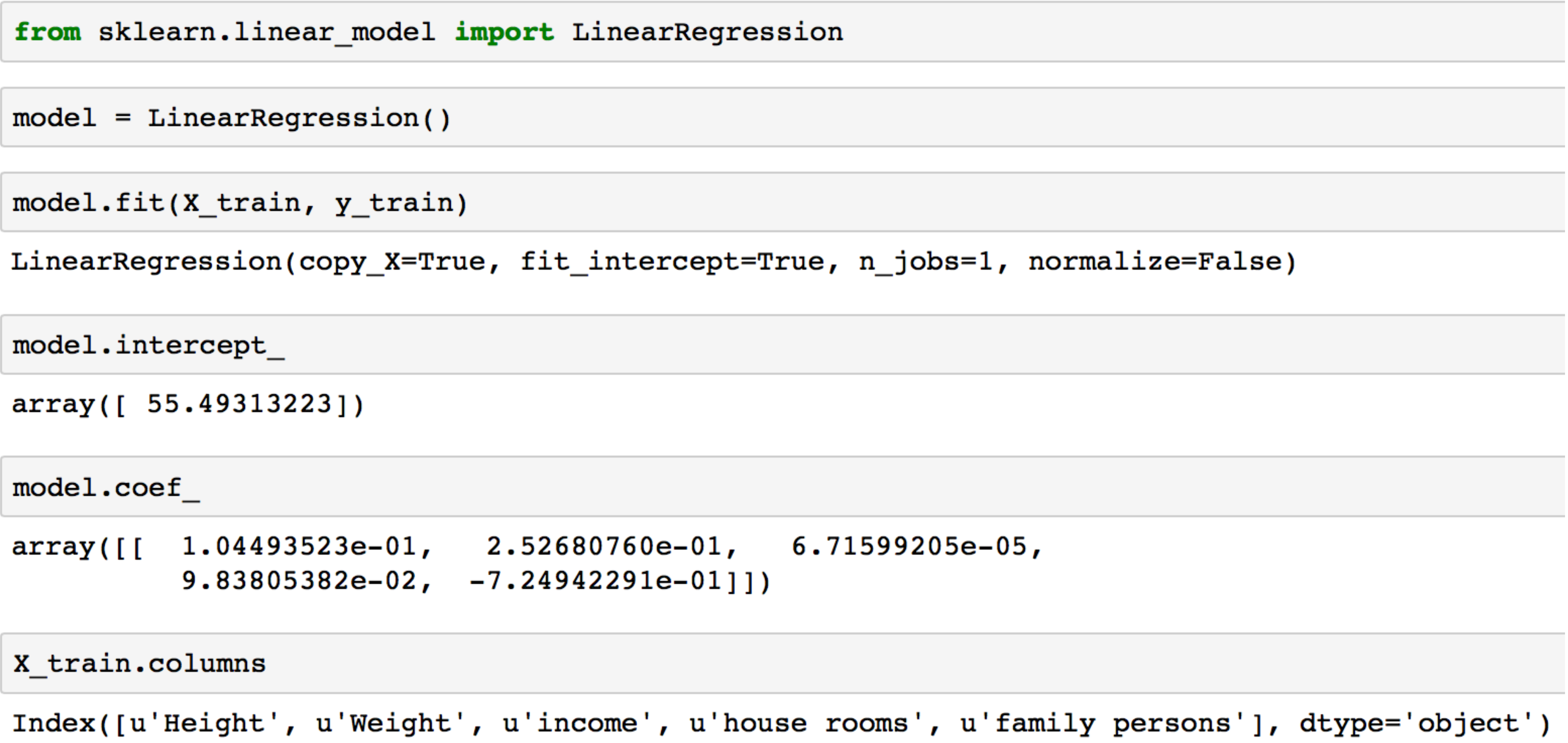


```
from sklearn.model_selection import train_test_split
```



```
X_train, X_test , y_train , y_test = train_test_split(X,y,test_size=0.35)
```

Train the model



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

```
model = LinearRegression()
```

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

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
model.intercept_
```

```
array([ 55.49313223])
```

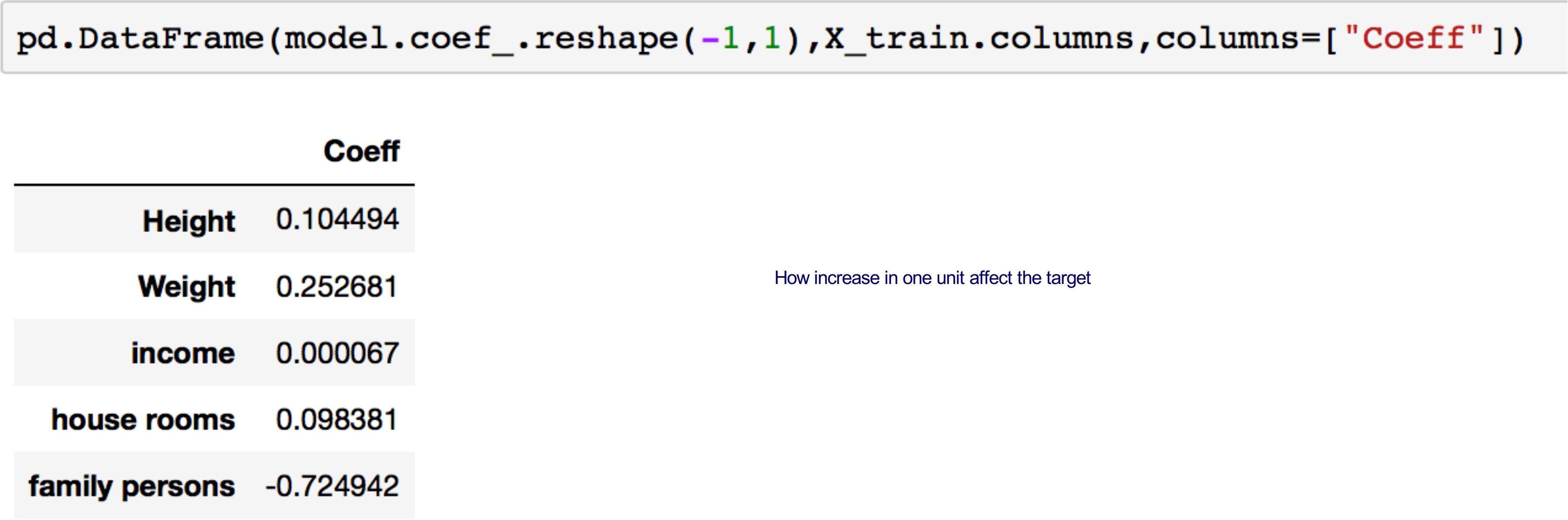
```
model.coef_
```

```
array([[  1.04493523e-01,   2.52680760e-01,   6.71599205e-05,
          9.83805382e-02,  -7.24942291e-01]])
```

```
X_train.columns
```

```
Index([u'Height', u'Weight', u'income', u'house rooms', u'family persons'], dtype='object')
```

Model Coefficient

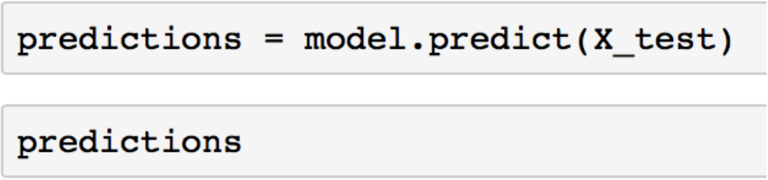


```
pd.DataFrame(model.coef_.reshape(-1,1),X_train.columns,columns=["Coeff"])
```

	Coeff
Height	0.104494
Weight	0.252681
income	0.000067
house rooms	0.098381
family persons	-0.724942

How increase in one unit affect the target

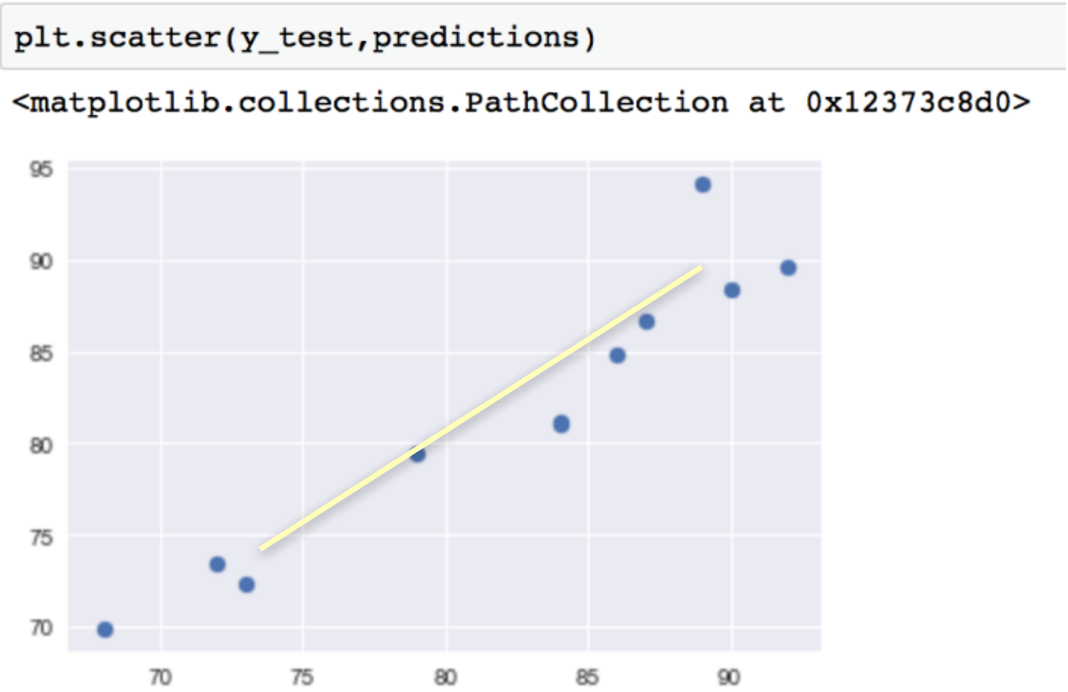
Using the Test Data



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

```
predictions
```

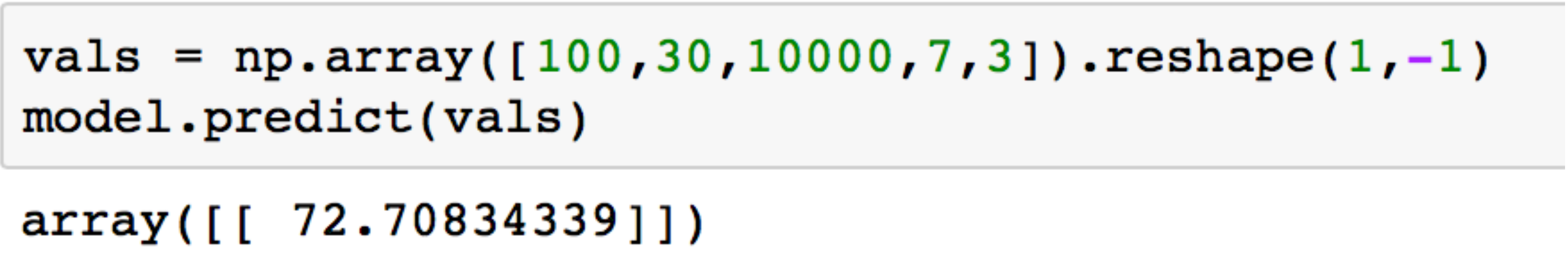
```
array([[ 81.14011874],
       [ 80.96265737],
       [ 88.40090084],
       [ 72.36243529],
       [ 94.07724114],
       [ 84.87499382],
       [ 73.48435341],
       [ 86.71113673],
       [ 89.61635687],
       [ 69.90781211],
       [ 79.46572653]])
```



```
plt.scatter(y_test,predictions)
```

```
<matplotlib.collections.PathCollection at 0x12373c8d0>
```

Predict the target



```
vals = np.array([100,30,10000,7,3]).reshape(1,-1)
model.predict(vals)
```

```
array([[ 72.70834339]])
```