A. Python Crash Course

Agenda

- A.1 Installing Python & Co
- A.2 Basics
- A.3 Data Types
- A.4 Conditions
- A.5 Loops
- A.6 Functions
- A.7 I/O
- A.8 OLS with Python

A.1 Installing Python & Co

- You can download and install Python directly from https://www.python.org

- Since we're going to use several libraries for **numerical computation** (numpy), **data analysis** (pandas), **machine learning** (sklearn), and **visualization** (matplotlib), it is easier to install Anaconda, which bundles all things required https://www.continuum.io/downloads

Running Python

- Python can be used **interactively**; for that it is convenient to use `ipython`, which provides **syntax highlighting** and **auto completion**

```
Python 3.6.0 |Anaconda custom (x86_64)| (default, Dec 23 2016, 13:19:00)
Type "copyright", "credits" or "license" for more information.


IPython 5.1.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.


In [1]: x = 2


In [2]: x*2
Out[2]: 4
```

Running Python

- You can also **run a file** of python code (common suffix: **`.py`**) by giving it as an argument to **`python`** or ipython

```
your-machine:~$ python your-file.py
```

- **Most editors** (e.g., Sublime and Emacs) support **directly executing** the current file (i.e., sending it to **`python`**)

A.2 Basics

- **Comments**
  - single-line comments: `# your short comment`
  - multi-line comments: `''' your long comment '''`

- Two important differences between **Python** and **Java**
  - Python is **dynamically-typed**, i.e., the type of a variable is determined at runtime and can change during execution
  - Python uses **indentation** (i.e., spaces or tabs) instead to group statements into blocks of code

Dynamic Typing and Indentation

Dynamic typing

```
x = 2
type(x) # returns int
x = 'Hello World'
type(x) # returns str
```

Code indentation

```
if (x % 2 == 0):
    print("even")
else:
    print("odd")
```

A.3 Data Types

- Python supports, among others, the following **basic types**
  - `int` for integers (e.g., `x = 2`)
  - `float` for floating point numbers (e.g., `x = 3.14`)
  - `str` for strings (e.g., `x = ‘Hello World’`)
  - we can build a `tuple` from multiple other values (e.g., `c = (49.14, 6.58)`)

- In addition, Python supports the following **container types**
  - `list` to store multiple values in a particular order
  - `set` to store multiple without order and repetitions
  - `dict` to store key-value pairs

Lists

Lists

```
l = [] # create an empty list
l.append(2) # insert 2 at the end
l.append('x') # insert 'x' at the end
l[1] = -2 # replace 'x' by -2
l.insert(1,0) # insert 1 at position 1
l.sort() # sort l in-place

l = [1,2,3,4,5,6,7] # create new list
l.reverse() # reverse list in-place
l[:2] # returns first two elements [7,6]
l[-2:] # returns last two elements [2,1]
l[1:3] # returns second and third element [6,5]

l = [[1,2],[3,4]] # lists can be nested
l[1][0] # returns first element of second list [3]
```

Sets

- **Sets**

```
u = set([]) # create empty set u
u.add(2) # add 2 to set
u.add('x') # add 'x' to set
u.add(2) # add 2 to set -- no effect
u.remove('x') # remove 'x' from set

v = set([2,3]) # create another set

union = u | v # compute union: {2,3}
intersection = u & v # compute intersection: {2}
```

Dictionaries

Dictionaries

```
c = {} # create an empty dictionary
c[1] = 'c' # associate value 'c' with key 1
c[2] = 'b' # associate value 'b' with key 2
c[3] = 'a' # associate value 'a' with key 3

k = sorted(c.keys()) # get sorted list of keys
v = sorted(c.values()) # get sorted list of values
```

A.4 Conditions

- **Conditions** (very similar to Java)

```
if (x == 1):
  print "two"
elif (x == 2):
  print "two"
else:
  print "other"
```

- **Conditionals** (similar to `(c ? a : b)` in Java) exist

```
output = ("even" if x % 2 == 0 else "odd")
```

A.5 Loops

- `For` loops are typically used to iterate over the items in a list

```
# loop over some prime numbers
primes = [1,2,3,5,7,11,13]
for prime in primes:
    print(str(prime) + " is a prime number")

# loop over the numbers 0, 1, ..., 9
for n in range(0,10):
  print(n)
```

- `While` loops

```
b = 2  # base
e = 10 # exponent
r = 1  # result
while e > 0:
    r = r*b
    e = e-1
print(r) # prints 1024
```

A.6 Functions

- Functions can be defined using the keyword **`def`**

```
def fak(n):
  if n==1:
    return 1
  else:
    return n*fak(n-1)
```

- Functions can have **more than one return value**

```
def split_in_halves(l):
  middle = int(len(l)/2)
  left_half = l[:middle]
  right_half = l[middle:]
  return left_half, right_half

left, right = split_in_halves([1, 2, 3, 4, 5])
left # returns [1, 2]
right # returns [3, 4, 5]
```

Functions

- Function arguments can have default values

```
def greet(m='Hello'):
  print(m)

greet() # prints "Hello"
greet("Hi") # prints "Hi"
```

Math

- **Mathematical functions** (e.g., sin and cos) are provided by the module `math`, which we first need to import

```
import math

math.gcd(10,3) # greatest common divisor
math.sin(0) # sine
math.cos(0) # cosine
math.floor(3.14) # floor
math.ceil(3.14) # ceil

help(math) # more information about the math module
```

A.7 I/O

- **Reading** a text file line by line

```
import os

file = open('/path/to/your/file')
for line in file:
    print(line)

file.close()
```

- **Writing** to a text file

```
import os

file = open('/path/to/your/file','w')
for i in range(0, 100):
  file.write(str(i) + "\n")

file.close()
```

A.8 OLS with Python

- Let's now look at how we can **create the plots** from today's lecture and **apply ordinary least squares**

- Use `pandas` to read the file `autos-mpg.data`

```
import pandas as pd

cars = pd.read_csv('/path/to/autos-mpg.data', header=None, sep='\s+')
```

which gives us a `pandas.core.frame.DataFrame` containing all **rows** and **columns** from the file (similar to a table in a relational database)

Extracting the Features of Interest

- The **first** column contains the `mpg` values; the **fourth** column contains the `hp` of the car

- Let's extract those columns into **separate variables**

```
import numpy as np

mpg = cars.iloc[:,0].values # returns a numpy.ndarray
hp = cars.iloc[:,3].values # returns a numpy.ndarray
```

which gives us two `numpy.ndarray`
as `numpy`'s array data type

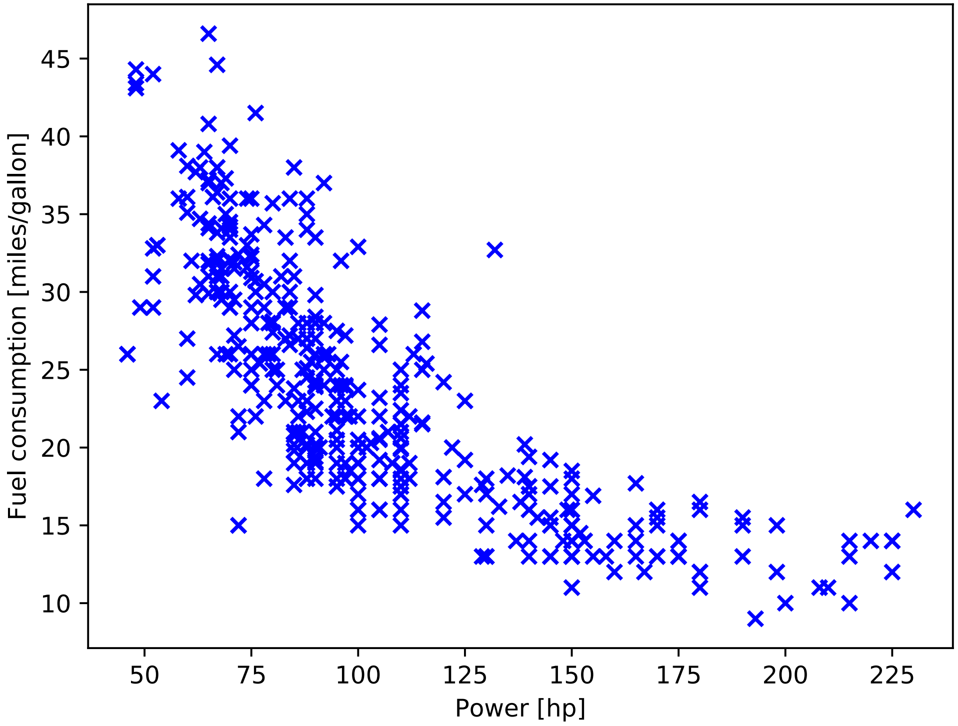
Plotting the Data

- Next, we can create a **scatter plot** of the data

```
import matplotlib.pyplot as plt

plt.scatter(hp, mpg, color='blue', marker='x')
plt.xlabel('Power [hp]')
plt.ylabel('Fuel consumption [miles/gallon]')
plt.show()
```

which gives us



Fitting a Regression Line

- Now, we can fit a **regression line** to predict **`mpg`** based on **`hp`**

```
from sklearn import linear_model

x = [] # we need an list of lists here
for i in hp:
  x.append([i])
y = mpg
reg = linear_model.LinearRegression()
reg.fit(x,y)
```

which gives us a model **`reg`** with the optimal coefficients

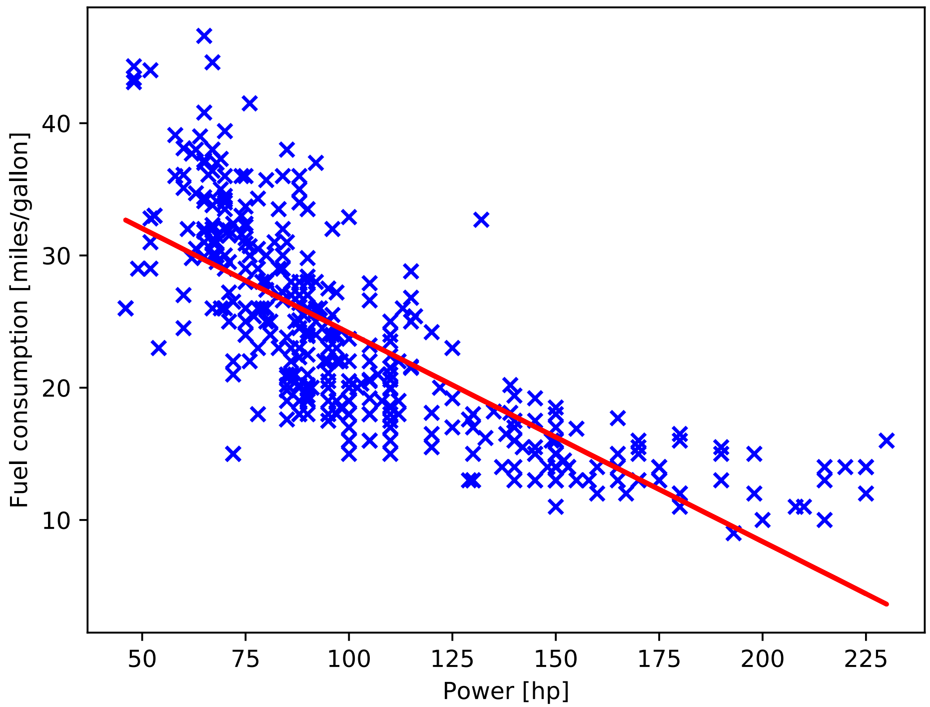
```
reg.intercept_ # returns 39.93586102117046 as the optimal w0
reg.coef_[0] # returns -0.15784473335365357 as the optimal w1
```

Plotting the Regression Line

- Finally, we plot the regression line in our scatter plot

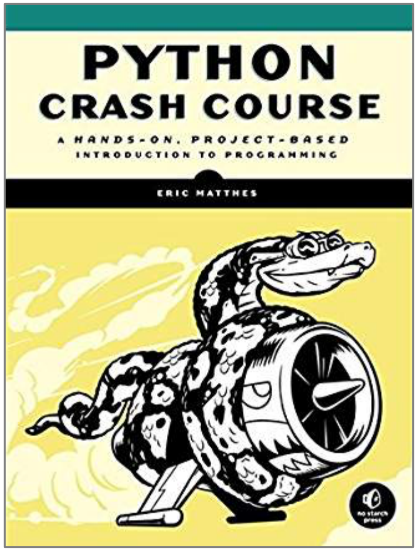
```
plt.scatter(hp, mpg, color='blue', marker='x')
plt.xlabel('Power [hp]')
plt.ylabel('Fuel consumption [miles/gallon]')
plt.plot(x, reg.predict(x), color='red', linewidth=2)
plt.show()
```

which gives us

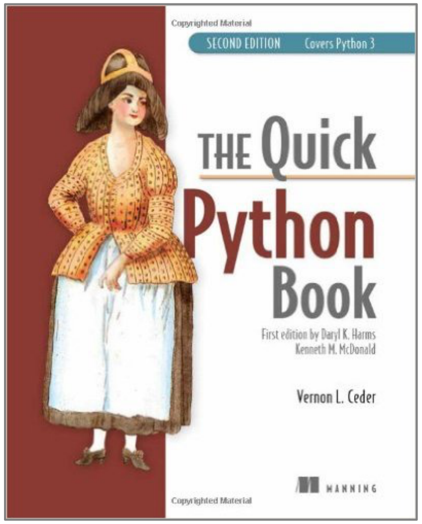


Literature

- **E. Mattes**: *Python Crash Course*, No Starch Press, 2016



- **V. L. Ceder**: *The Quick Python Book*, Manning, 2010



References

- **Python 3.6.1 Documentation**
  https://docs.python.org/3/
- **LearnPython.org**
  https://www.learnpython.org
- **NumPy**
  http://www.numpy.org
- **Python Data Analysis Library (pandas)**
  http://pandas.pydata.org
- **scikit-learn**
  http://scikit-learn.org/stable/
- **Matplotlib**
  https://matplotlib.org