Data Mining

Intro. Python

Santiago Alonso-Díaz

Tecnólogico de Monterrey
EGADE, Business School

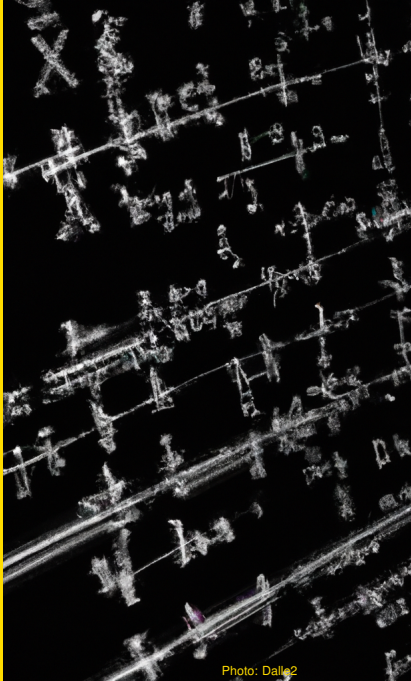


Photo: Dalle2

Why should someone interested in businesses learn/understand code?

- Excel is great but ...
- Data science
- Execute non-trivial models

Example of non-trivial models

Which city has more population?

- Lima
- Buenos Aires

Lee and Wagenmakers, 2014

Example of non-trivial models

Memory of	Country Population	Capital	GDP	World cup wins
Lima	¯_(ツ)_/¯	Yes	¯_(ツ)_/¯	No
Buenos Aires	¯_(ツ)_/¯	Yes	¯_(ツ)_/¯	Yes

Lee and Wagenmakers, 2014

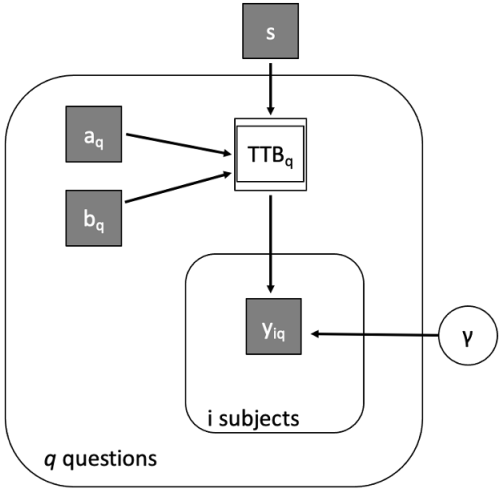
Example of non-trivial models

Memory of	Country Population	Capital	GDP	World cup wins
Lima	¯_(ツ)_/¯	Yes	¯_(ツ)_/¯	No
Buenos Aires	¯_(ツ)_/¯	Yes	¯_(ツ)_/¯	Yes

Lee and Magenmakers, 2014

Example of non-trivial models

Graphical representation of take-the-best (TTB)



Observed:

- a_q,b_q: vector of cues for option a or b
- s: order of cues (sorted by validity)
- y_{iq} = decision made by subject i to question q

Latent / parameters:

- TTB_q = TTB decision (a or b)
- γ = Probability of reporting the TTB decision (Bernoulli)

Lee and Wagenmakers, 2014

Example of non-trivial models

Algorithmic representation of take-the-best (TTB)

```
model <- "
data {
  //load data
}
transformed data {
  // add some transformations to the data
}
parameters {
  // Choose TTB Decision With Probability Gamma, or Guess
  ttb[1] <- 1 - gamma;
  ttb[2] <- .5;
  ttb[3] <- gamma;
}
model {
 for (q in 1:nq)
    for (i in 1:ns)
      y[i,q] ~ bernoulli(ttb[t[q]]);
}"
```

Lee and Wagenmakers, 2014

Table of contents

Python basics

Basics

First, a Jupyter notebook premier on Google's Collab

Basics

Variable types:

- Strings
- Integers
- Floats
- Lists
- Dictionaries
- Arrays (from numpy)
- Data frame (from pandas)
- Others

Try in Python:

- type("hello world")
- type(17)
- type(6.28)
- type(6,28) Why an error?
- type(["hello world", 17, 6.28])
- type(np.array([[1,2],[3,4]]))
- type(pd.DataFrame(np.array([[1,2],[3,4]])))

Basics

Variables:

- x = 2
- y = "2"
- my_var = 3*4
- my.var = 3*4; WRONG, dot in name
- my_data = pd.read_csv("data_folder\shipments.csv")

Never use these for variable names (core Python routines):
and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try, nonlocal

Basics

Try in Python:
principal = 30000
rate = 0.1
principal*Rate; Why an error?

Basics

Operators:

- + (sum)
- - (minus)
- * (times)
- / (division)
- ** (power)
- % (modulo or placeholder for strings: 'My %d camels from %s' %(3, "Dubai")

The data type determines the existence and effect of a given operator. For instance, try in Python 3*"A" or "Car" + "toon"

Basics

Functions (native in Python):

- type(32)
- int("32")
- int("hello") ... why an error?
- int(-2.3)
- float(32)
- ... (many more)

Try this in Python:

```
prompt = "Cual es la velocidad de un bus de transmilenio? "
speed = input(prompt)
print("La velocidad es: ", str(speed))
```

Basics

Functions (from packages):

```
import math
print(math.pi)
print(math.log(math.e))
print(math.factorial(6))
y = 20
x = math.exp(math.log(y+1))
print(x)
```

Basics

Functions (yours):

```
def print_lyrics():
    print (“I’m a lumberjack, and I’m okay.”)
    print (“I sleep all night and I work all day.”)
def print_lyrics()

def print_twice(text):
    print (text)
    print (text)
print_twice(math.pi)
print_twice("hello world")
```

NOTE THE INDENTATIONS. PYTHON CARES ABOUT THEM.

Basics

Variables in functions are local:

```
def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)
    return cat #COMMENT: returns this value
line1 = 'Bing tiddle '
line2 = 'tiddle bang.'
cat_twice(line1, line2)
```

Basics

Classes:

```
class Point(object):
    "Represents a point in 2-D space."
class Rectangle(object):
    "Represents the dimensions of a rectangle."
blank = Point() #object of the class Point
blank_rect = Rectangle() #object of the class Rectangle

blank.x = 3.0 #attribute for the object
blank.y = 4.0
blank_rect.width = 20
blank_rect.height = 50
print(blank.x, hasattr(p, 'x'), hasattr(p, 'z'))
distance = math.sqrt(blank.x**2 + blank.y**2)
```

Basics

Basics

For this and more, let's move to Python now.
Open: Python \ Python Tutorial \ python-intro - Eng.ipynb

Python data objects

Numpy

Loosely based on scipy.org lectures

What is Numpy?

Python package to build and compute on arrays of any dimension (e.g. 1D vectors, 2D matrices, 3D coordinates, 4D fMRI data).

Numpy

In Python do:

1D array

```
a = np.array([0, 1, 2, 3])
print(a, a.ndim, a.shape)
```

2D array

```
b = np.array([[0, 1], [2, 3]])
print(b, b.ndim, b.shape)
```

3D array

```
c = np.array([[[1,2], [2,3]], [[3,4], [4,5]]])
print(c, c.ndim, c.shape)
```

Numpy

Not just real numbers:

Complex numbers

```
a = np.array([1+2j, 3+4j, 5+6*1j])
```

Booleans

```
b = np.array([True, False, False, True])
```

Strings

```
c = np.array(['Bonjour', 'Hello', 'Hallo'])
```

Numpy

Numpy has many routines/functions. Do in Python:

```
a = np.arange(1, 9, 2)
b = np.linspace(0, 1, 6)
c = np.ones((3, 3))
d = np.zeros((2, 2))
e = np.random.rand(4)
f = np.random.randn(4,4)
g = np.sqrt(33)
h = np.sort([3,2,1])
i = np.round(np.pi, 4)
```

Numpy

Linear algebra functions:

Dot product

```
a = [[1, 0], [0, 1]]
b = [[4, 1], [2, 2]]
np.dot(a, b)
```

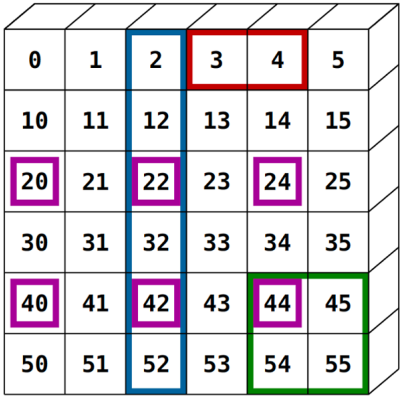
Eigenvalues

```
matrix = np.array([[1,2],[3,4]])
np.linalg.eigvals(matrix)
```

Numpy

Indexing:

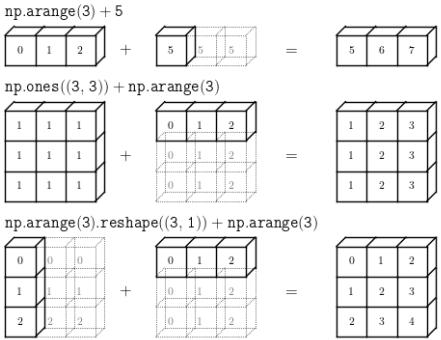
```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 45],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])

>>> a[2::2, ::2]
array([[20, 22, 24],
       [40, 42, 44]])
```



Source: https://lectures.scientific-python.org/

Numpy

Broadcasting:



Source: https://numpy.org/doc/stable/user/basics.broadcasting.html

Pandas

Check Molin tutorial on Github

What is Pandas?

Python package based on Numpy but with more data science functionalities.

Polars

Check Polars documentation

What is Polars?

Python package inspired in Pandas but faster.

References


Lee, M. D., & Wagenmakers, E.-J. (2014). *Bayesian cognitive modeling: A practical course*. Cambridge university press.