DSC 201: Data Analysis & Visualization

Python Basics

Dr. David Koop
Jupyter Notebook

- Display rich representations and text
- Uses Web technology
- Cell-based
- Built-in editor
- GitHub displays notebooks
JupyterLab

Open a CSV file using Pandas

In [5]:
1. import pandas
2. df = pandas.read_csv('data/iris.csv')
3. df.head(10)

Out[5]:
<table>
<thead>
<tr>
<th>sepal_length</th>
<th>sepal_width</th>
<th>petal_length</th>
<th>petal_width</th>
<th>species</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td><strong>setosa</strong></td>
</tr>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.7</td>
<td>3.2</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.1</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.6</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>4.6</td>
<td>3.4</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.4</td>
<td>1.5</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.4</td>
<td>2.0</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>3.1</td>
<td>1.5</td>
<td>0.1</td>
<td>setosa</td>
</tr>
<tr>
<td>4.4</td>
<td>3.0</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

JupyterLab Demo

JupyterLab: The next generation user interface for Project Jupyter

https://github.com/jupyter/jupyterlab

It has been a collaboration between:
- Project Jupyter
- Bloomberg
- Anaconda

1) Building blocks of interactive computing

[JupyterLab Documentation]
Jupyter Notebook & JupyterLab @ UMassD

• https://rps.cscvr.umassd.edu:8000/
• Just log in and everything should be setup
• For JupyterLab, click the corresponding button after you log in
• Advanced:
  - Can add your own conda environments in your user directory
Assignment 1

• Link
• Analyze Atlantic hurricane data
• You need to manipulate lists and strings to extract the answers
• Find:
  - Number of unique names
  - Hurricane with top sustained wind
• Code to read data is provided
• Code to check answers is provided
• Due Tuesday, Sept. 25 at 11:59pm
• Get started!
Python Compared to Java

• Dynamic Typing
  - A variable does not have a fixed type
  - Example: \( a = 1; \ a = "abc" \)

• Indentation
  - Braces define blocks in Java, good style is to indent but not required
  - Indentation is critical in Python
  - Example:
    
    ```python
    z = 20
    if x > 0:
        if y > 0:
            z = 100
        else:
            z = 10
    ```
Print function

• `print("Hello World")`

• Can also print variables:

  ```python
  name = "Jane"
  print("Hello," , name)
  ```
Python Variables and Types

• No type declaration necessary

• Variables are names, not memory locations
  
  ```python
  a = 0
  a = "abc"
  a = 3.14159
  ```

• Don't worry about types, but think about types

• Strings are a type

• Integers are as big as you want them

• Floats can hold large numbers, too (double-precision)
Python Math and String "Math"

- Standard Operators: +, -, *, /, %
- Division "does what you want" (new in v3)
  - 5 / 2 = 2.5
  - 5 // 2 = 2 # use // for integer division
- Shortcuts: +=, -=, *=
- No ++, --
- Exponentiation (Power): **
- Order of operations and parentheses:
  - 4 - 3 - 1
  - 4 - (3 - 1)
- "abc" + "def"
- "abc" * 3
Python Strings

• Strings can be delimited by single or double quotes
  - "abc" and 'abc' are exactly the same thing
  - Easier use of quotes in strings: "Joe's" or 'He said "Stop!"'

• String concatenation: "abc" + "def"

• Repetition: "abc" * 3

• Special characters: \n \t like Java

• Indexing:
  a = "abcdef"
  a[0]

• Slicing: a[1:3]

• Format: name = "Jane"

  print("Hello, {}".format(name))
Exercise

• Given variables x and y, print the long division answer of x divided by y with the remainder.

• Examples:
  - x = 11, y = 4 should print "2R3"
  - x = 15, y = 2 should print "7R1"
Loops

• while <condition>:
  <indented block>
  # end of while block (indentation done)

• Remember the colon!

• a = 5
  while a > 0:
    print(a)
    a -= 2

• a > 0 is the condition

• Python has standard boolean operators (<, >, <=, >=, ==, !=)
  - What does a boolean operation return?
  - Linking boolean comparisons (and, or)
break and continue

- **break** stops the execution of the loop
- **continue** skips the rest of the loop and goes to the next iteration
Conditionals

- *if, else*
  - Again, indentation is required
- *elif*
  - Shorthand for `else: if:
- *Same type of boolean expressions (and or)*
True and False

• True and False (captialized) are defined values in Python
• v == 0 will evaluate to either True or False
Quiz

• Suppose I want to write Python code to print the numbers from 1 to 100. What errors do you see?

```python
// print the numbers from 1 to 100
int counter = 1
while counter < 100 {
    print counter
    counter++
}
```
Functions

• Calling functions is as expected:
  
  \texttt{mul(2,3)} \# \texttt{computes 2*3 (mul from operator package)}

- Values passed to the function are parameters
- May be variables!

  \begin{verbatim}
  a = 5
  b = 7
  mul(a,b)
  \end{verbatim}

• \texttt{print} is a function

  \begin{verbatim}
  print("This line doesn't end", end="")
  print("See it continues")
  \end{verbatim}

- \texttt{end} is also a parameter, but this has a different syntax
- Keyword argument!
Why do we create and use functions?
Defining Functions

- **def** keyword

- Arguments have names but **no types**

```python
def hello(name):
    print("Hello {}".format(name))
```

- Can have defaults:

```python
def hello(name="Jane Doe"):  
    print("Hello {}".format(name))
```

- With defaults, we can skip the parameter

```python
hello() or hello("John")
```

- Also can pick and choose arguments:

```python
def hello(name1="Joe", name2="Jane"):  
    print("Hello {} and {}".format(name1, name2))

hello(name2="Mary")
```
Return statement

• Return statement gives back a value:
  
  ```python
  def mul(a,b):
      return a * b
  ```

• Variables changed in the function won't be updated:
  
  ```python
  def increment(a):
      a += 1
      return a
  
b = 12
  c = increment(b)
  print(b,c)
  ```
Python Containers

• Container: store more than one value
• Mutable versus immutable: Can we update the container?
  - Yes → mutable
  - No → immutable
    - Lists are mutable, tuples are immutable
• Lists and tuples may contain values of different types:
  • List: [1, "abc", 12.34]
  • Tuple: (1, "abc", 12.34)
• You can also put functions in containers!
  • len function: number of items: len(l)