UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Python - Refresher

Machine Learning

Get python

- Exclusively python 3
  - Any version should be fine, so why not get the latest (3.9.1)
  - Python 2 no longer supported
- Install from:
  - https://www.python.org/ — pip3 is package manager
  - https://anaconda.org/ — conda package manager
- Jupyter Notebooks
  - Get after installing python 3, on command line (for example):

```
pip3 install jupyter
```

  - … and test

```
. ~.bash_profile; jupyter notebook
```

Basic control flow

- Python blocks are indented, control ends with a colon(:) character
- Commands operate more or less like C/C++/Java:

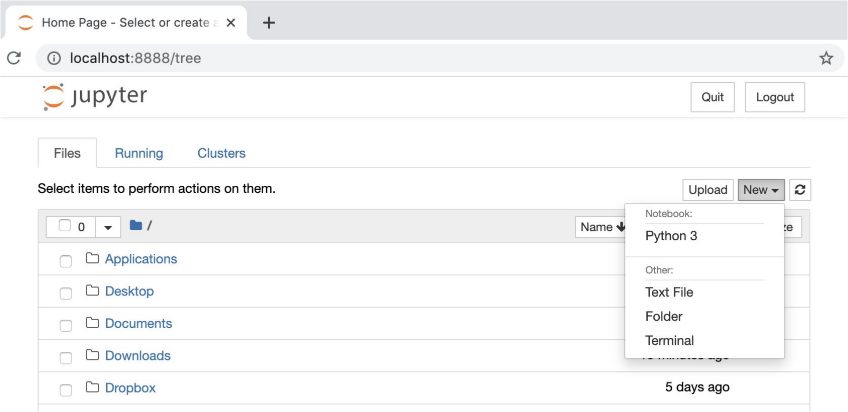
```
if / if… elif … else

for

range(s, t) # Returns a sequence from s to (t-1) ...
```

- Functions

```
def fib(n):
```

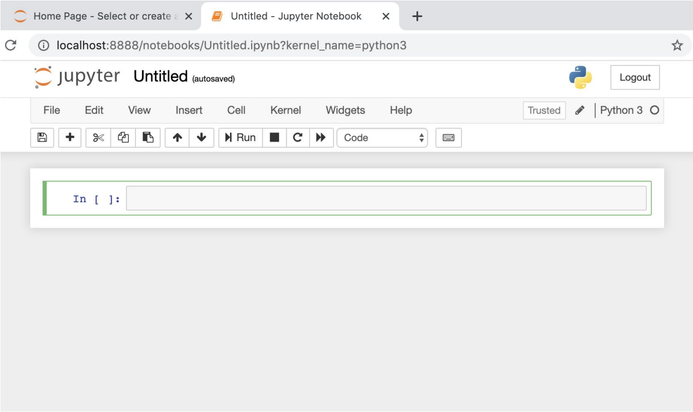
Jupyter

- Interactive environment for languages in Julia, Python, R and others
- Run code snippets immediately, get output immediately
- To get a new notebook, select: `New > Python 3`



localhost:8888/tree

Select items to perform actions on them.



localhost:8888/notebooks/Untitled.ipynb?kernel_name=python3

Jupyter shortcuts

Action	Shortcut
Run cell	`Ctrl-Enter`
Run cell, select below	`Shift-Enter`
Convert cell to Markdown	`M`
Convert cell to code	`Y`
Insert cell above	`A`
Insert cell below	`B`
Cut selected cell	`X`
Delete selected cell	`D, D`
Merge with cell below	`Shift-M`

Python basics (01) - variables

- Comments start with #; block comments start & end with ''' / """
- Syntax is C/C++/Java-esque; eg.

```
x = 1 # Integer instance; float equivalent: 1. -OR- 1.0

y = 'hello world' # String literals with single / double quotes
```

- Variables are not statically typed — eg.

```
x = 1

x = 'hello world'
```

Python basics (02) - variable types

- We can determine the type of a variable with the “type” command

```
x = 1.

type(x)

x = 'hello world'

type(x)
```

- Basic types:
  - str — string
  - bool (True / False) — note uppercase “T” and “F”
  - int / float
  - None — Equivalent to `null` / `NULL` in C++/Java

Python basics (03) - list basics

- Akin to vectors / arrays (without the need to declare size)
- Can mix types (floats, strings, other lists, etc.)

```
x = ['1st', 2, '3rd']
```

- Can be indexed from back or front

```
x[0] # '1st'

x[-1]  # '3rd'
```

- Lists are mutable
  - Tuples — closely related — are immutable
  - Tuples declared with round brackets

Python basics (04) - list slicing

- Lists can be “sliced” to generate a subset
- Slice an array using a `:` to separate the start from end
- When slicing a list, from start:end, start is included; end is excluded

```
x = ['1st', 2, '3rd']

a = x[1:-1]

# What does a contain?

b = x[-2:]

# What does b contain?

c = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ] # Lists of lists
```

Python basics (05) - list operations

Operation	Explanation	Example / Usage
`[]`	Create an empty list	`x = []`
`len`	Return the length of the list	`len(x)`
`append`	Add a single element to the end of the list	`x.append([-2, -1])`
`insert`	Insert an element to a given position	`x.insert(0, '1st')`
`del`	Remove a list element (or slice)	`del(x[0])`
`remove`	Search for and remove a given value	`x.remove('1st')`
`reverse`	Reverse a list (in place)	`x.reverse()`

Python basics (06) - list operations

Operation	Explanation	Example / Usage
`sort`	Sort a list in place	`x.sort()`
`+`	Add two lists together	`x + y`
`*`	Return a list “n” times larger, elements copied	`x = ['y'] * 3`
`min`	Return the smallest element in a list	`min(x)`
`max`	Return the largest element in a list	`max(x)`
`index`	Return the position of a value in a list	`x.index('1st')`
`count`	Count the number of times a value occurs in a list	`x.count(19)`

Python basics (07) - list operations

Operation	**Explanation**	**Example / Usage**
`extend`	Add multiple elements to the end of the list	`x.extend([-2, -1])`
`in`	Return True if item is in list; False otherwise	`'1st' in x`

Python basics (08) - dictionaries

- Akin to hash tables / associative arrays
  - For list, key is index +/- (0.. N-1), and items are ordered
  - For dictionary, key must be explicitly declared, and items are unordered
- Examples

```
eng_to_french = {}

eng_to_french['blue'] = 'bleu'

eng_to_french['red'] = 'rouge'

print('In French, red is', eng_to_french['red'])
```

Python basics (09) - dictionaries

Operation	Explanation	Example / Usage
`{}`	Create an empty dictionary	`x = {}`
`len`	Return number of items in dictionary	`len(x)`
`keys`	Return all keys in dictionary	`x.keys()`
`values`	Return all values in dictionary	`x.values()`
`items`	Return all items in dictionary (as tuples)	`x.items()`
`del`	Remove an entry from dictionary	`del(x['red'])`
`in`	Return True if key exists in dictionary’s keys	`'blue' in x`

Python basics (10) - dictionaries

Operation	Explanation	Example / Usage
`get`	Return the value of a key (or default)	`x.get('green', None)`
`setdefault`	Set the value to the default if key does not exist; return the value	`x.setdefault('y', None)`
`copy`	Make a copy of dictionary	`y = x.copy()`
`update`	Add entries from another dictionary instance	`x.update(y)`

numpy (01)

- Numerical Python (numpy)
  - Designed for high-performance analysis
  - Fast, vectorised array operations
  - Where possible, use numpy operations instead of python loops... because: speed
- Must be imported, ala the below (“as np” is convention):

```
import numpy as np
```

- Numpy arrays must have homogeneous type (all int, for example)

```
x = np.array([[1, 2, 3], [-99, -98, -97]])

print(x.shape)   # Tuple of array dimensions

print(x)    # How is this different to a list?
```

numpy (02) — not a list, but...

- Convert to a python list:

```
x = np.array([[1, 2, 3], [-99, -98, -97]])

y = x.tolist()
```

- What is the result of the below? (How are they different?)

```
print(x + x)

print(y + y)
```

numpy (03) — Some operations

Operation	Explanation	Example / Usage

`dtype`	Return type of numpy array	`x.dtype`
		`  => dtype('int64')`

`zeros`	Create an n-dimensional array with all instances = 0	`x = np.zeros(5)`
		`  => [0, 0, 0, 0, 0]`

`empty`	Create an n-dimensional array, randomly initialised	`x = np.empty(15)`

`arange`	Create an array from 0 to parameter	`x = np.arange(5)`
		`  => [0, 1, 2, 3, 4]`

`[start:end]`	Slice array	`x[1:3] = 5`
		`  => [0, 5, 5, 3, 4]`

numpy (04) — Operators

Operation	Explanation	Example / Usage
`sum`	Return the sum of numpy array	`x.sum()`
`mean`	Return the mean of numpy array	`x.mean()`

`>, <, etc.`	Return array of pairwise comparison between numpy arrays	`x > z`
		` => [True, False, … ]`

Pandas Overview

- “Panel Data” = package for manipulating tabular data
- Must be imported, ala

```
import pandas as pd
```

- Two main data structures:
    - Series — 1-dimensional column-vector, is an extension of ndarray object in numpy, with additional features that facilitate data analysis
    - DataFrame — spreadsheet–like collection of Series objects

Pandas Series (01)

- A Series object can be created and initialized by passing either a scalar, a numpy array, a list or a dictionary
- What is the result of the below?

```
series = pd.Series(15) # Also try: series = pd.Series(np.arange(5))

print(series)
```

- Note that series has 2 parts:
  - Scalar value (eg. 3)
  - Index / row label (eg. 0) — which we will use for analysis

Pandas Series (02)

- Importantly, a Series can be initialised with a named index
- Index “names” must be list of string, int, etc. instances

```
series1 = pd.Series([10, 9, 8], index = ['colour', 'size', 'wgt'])

series2 = pd.Series([900, 19, 31], index = ['size', 'price', 'r'])
```

- Operations are performed according to named index
- Consider result of:

```
series1 + series2
```

- Caveat: a Series may have duplicate indices, and that may act… strangely

Pandas Series (03)

Operation	Explanation	Example / Usage
`index`	Return index ranges	`series.index`
`values`	Return series values as numpy array	`series.values`
`loc`	Return value based on named index	`x.loc['blue']`

`iloc`	Return value based on index	`x = pd.Series(np.arange(1, 4))`
		`x.iloc[2]`

Pandas DataFrame (01)

- A DataFrame is a collection of Series instances aligned according to named label
- Each column in a DataFrame instance has homogenous data
- Each row in a DataFrame can composed from heterogeneous data
- Create a DataFrame in many ways, eg.:

```
df1 = pd.DataFrame([[111, 222], ['a', 'b']])

df2 = pd.DataFrame(np.array([[111, 222], ['a', 'b']]))

df3 = pd.DataFrame([pd.Series([111, 222]),pd.Series(['a', 'b'])])
```

Pandas DataFrame (02)

- DataFrame instances are commonly created from CSV / JSON files

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

df = pd.read_json('data.json')
```

- Other useful functions

```
head() # Shows the first n rows (n = 5?)  “tail()” shows last n rows

describe() # Shows counts, min, max, interquartile ranges, etc.
```

- Try it
  - ○ Download titanic.csv
  - ○ Explain the data

Pandas DataFrame (03)

- DataFrame instances can also be index and sliced; using titanic data:

```
df.Name

df.Name[890]

df.Fare[500:]
```

- Combining naming, slicing:

```
df['Name'].head(2)

df[['Survived', 'Fare']].head(2)
```

Pandas DataFrame (04)

- Your friends: loc, iloc
- Index on [row:row, col:col]

```
df.loc[:, 'Name'].head()   # What type / values does this return?

df.iloc[:, 2].head(2)      # Equivalent to: df.iloc[:2, 2]
```

- Add columns by naming and assigning values:

```
df['age_squared'] = df.Age**2
```

- Delete columns with “del” operation or “drop” function:

```
del df['age_squared'] # -OR- df = df.drop(['age_squared'], axis =
1)\ -OR- df.drop(['age_squared'], axis = 1, inplace = True)
```

Pandas DataFrame (04)

Operation	Explanation	Example / Usage
`mean() / median() / mode () / var() / min()/ max()`	Functional of column (mean, median, mode, variance, etc.)	`df.Fare.mean()`
`count()`	Number of instances (rows) in a dataframe	`df.count()`
`unique()`	All unique values of a column	`df.Survived.unique()`
`value_counts()`	Number of instances by value	`df.Survived.value_counts()`

Pandas DataFrame (05)

- Missing values — represented by NaN (not a number), but you may see "NA" or other values

```
df = pd.concat([pd.Series([1, 2, 3]).rename('mycol1'),
                pd.Series([4, np.nan, 6]).rename('mycol2'),
                pd.Series([7, np.nan, 17]).rename('mycol3')],
axis=1)
```

- Often pose problems in data, DataFrames, etc.
- Detect them in pandas with ".isnull()" / ".notnull()"

```
df.isnull()
```

- Combine with other functions

```
df.isnull().sum()
```

Pandas DataFrame (06)

- Replace NaN values with “fillna()” function

```
df.fillna(23)
```

- Perhaps better to use “interpolate()” function

```
df.mycol2.interpolate()
```

- These functions do not change the values in the DataFrame instance

Pandas DataFrame (07)

- Use groupby() function to split data according to values

```
df = pd.read_csv('titanic.csv')  # I’m cynical about titanic data
df.groupby('Pclass').mean()
```