Python 101

Agenda

- Day 1: Python, Pandas & Pep8
- Day 2: Matplotlib, Seaborn & Machine Learning
- Day 3: Linear regression, tree models and capstone overview
- Day 4: Openai, Capstone part 2, Tree model, Technical indicators
- Day 5: Classification tree, class, backtesting, prediction
- Day 6: Yahoo finance, LSTM

Source: Type source image link here

Day 1 Agenda

- We will be covering the following topics
  - Python
  - Pandas
  - Pep 8

Source: Type source image link here

Python 101

Python 101

- We will be covering the following topics
  - Fundamental Data Types
  - Comparison and Logical Operators
  - Control Flow Statements
  - Functions
  - Built-in Python Expressions

Source: Type source image link here

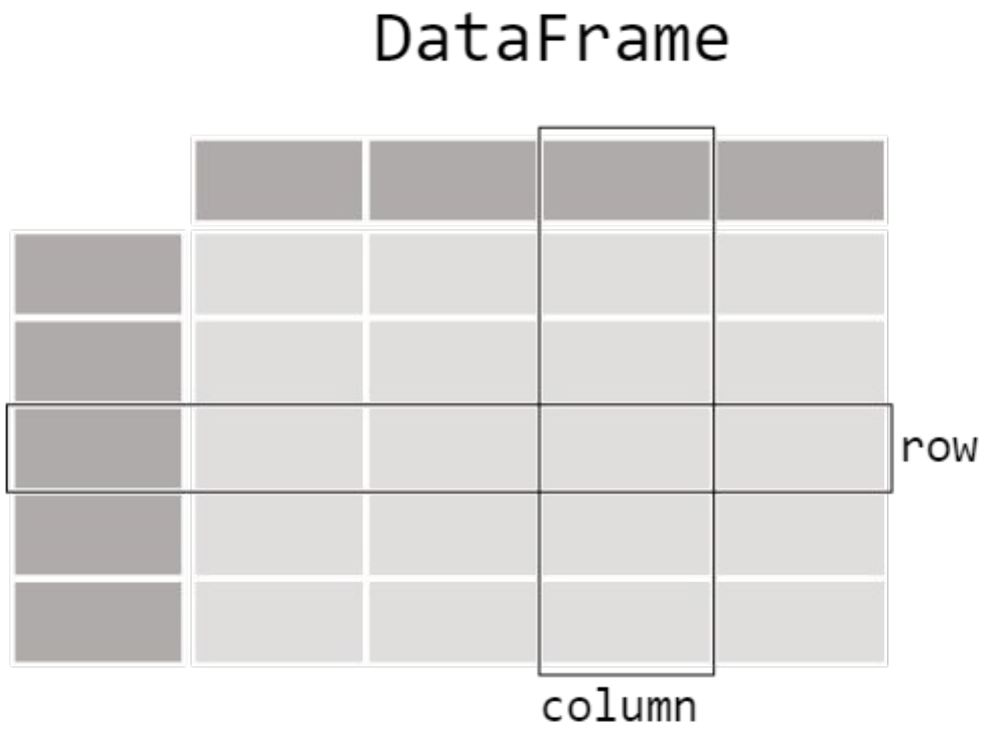
Pandas

Pandas 101

- We will be covering the following topics
  - Series
  - DataFrame
  - Groupby

Source: Type source image link here

Pandas: Data Table Representation

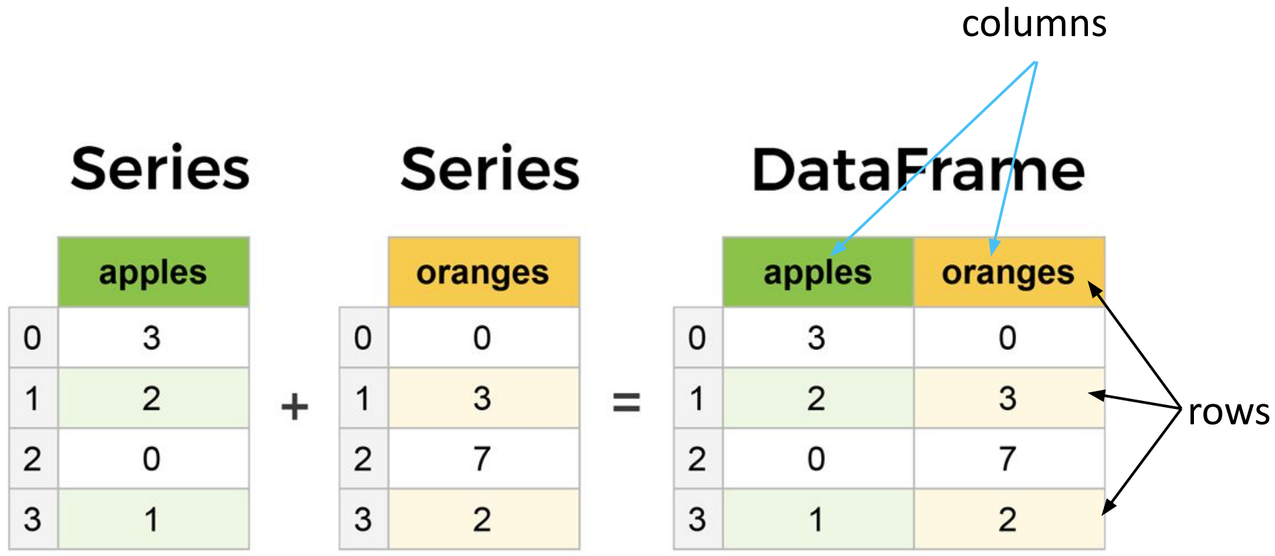


Source: Type source image link here

Core components of pandas: Series & DataFrames

- The primary two components of pandas are the Series and DataFrame.
  - Series is essentially a column, and
  - DataFrame is a multi-dimensional table made up of a collection of Series.
- DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.

- Features of DataFrame
  - Potentially columns are of different types
  - Size – Mutable
  - Labeled axes (*rows* and *columns*)
  - Can Perform Arithmetic operations on rows and columns



Source: Type source image link here

pandas.DataFrame

pandas.DataFrame(data, index , columns , dtype)

- data: data takes various forms like *ndarray*, *series*, *map*, *lists*, *dict*, constants and also another *DataFrame*.
- index: For the **row labels**, that are to be used for the resulting frame, Optional, Default is *np.arrange(n)*if no index is passed.
- columns: For **column labels**, the optional default syntax is - *np.arrange(n)*. This is only true if no index is passed.
- dtype: Data type of each column.

- **Create DataFrame**
  - A pandas DataFrame can be created using various inputs like −
    - Lists
    - dict
    - Series
    - Numpy ndarrays
    - Another DataFrame

Creating a DataFrame from scratch

- There are many ways to create a DataFrame from scratch, but a great option is to just use a simple dict. But first you must import pandas.

```
import pandas as pd
```

- Let's say we have a fruit stand that sells apples and oranges. We want to have a column for each fruit and a row for each customer purchase. To organize this as a dictionary for pandas we could do something like:

```
data = { 'apples':[3, 2, 0, 1] , 'oranges':[0, 3, 7, 2] }
```

- And then pass it to the pandas DataFrame constructor:

```
df = pd.DataFrame(data)
```

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

How did that work?

- Each (key, value) item in data corresponds to a column in the resulting DataFrame.
- The Index of this DataFrame was given to us on creation as the numbers **0-3**, but we could also create our own when we initialize the DataFrame.
- E.g. if you want to have customer names as the index:

```
df = pd.DataFrame(data, index=['Ahmad', 'Ali', 'Rashed', 'Hamza'])
```

	apples	oranges
Ahmad	3	0
Ali	2	3
Rashed	0	7
Hamza	1	2

- So now we could locate a customer's order by using their names:

```
df.loc['Ali']
```

```
apples      2
oranges     3
Name: Ali, dtype: int64
```

Loading dataset

- We're loading this dataset from a CSV and designating the movie titles to be our index.

```
movies_df = pd.read_csv("movies.csv", index_col="title")
```

Viewing your data

- The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with **.head()**:

```
movies_df.head()
```

- .head() outputs the first five rows of your DataFrame by default, but we could also pass a number as well: **movies_df.head(10)** would output the top ten rows.

- To see the last five rows use **.tail()** that also accepts a number, and in this case we printing the bottom two rows.:

```
movies_df.tail(2)
```

Getting info about your data

- **.info()** should be one of the very first commands you run after loading your data
- **.info()** provides the essential details about your dataset, such as the number of rows and columns, the number of non-null values, what type of data is in each column, and how much memory your DataFrame is using.
-

```
movies_df.info()
```

```
movies_df.shape
```

OUT:

```
<class 'pandas.core.frame.DataFrame'>
Index: 1000 entries, Guardians of the Galaxy to Nine Lives
Data columns (total 11 columns):
Rank                    1000 non-null int64
Genre                   1000 non-null object
Description             1000 non-null object
Director                1000 non-null object
Actors                  1000 non-null object
Year                    1000 non-null int64
Runtime (Minutes)       1000 non-null int64
Rating                  1000 non-null float64
Votes                   1000 non-null int64
Revenue (Millions)      872 non-null float64
Metascore               936 non-null float64
dtypes: float64(3), int64(4), object(4)
memory usage: 93.8+ KB
```

OUT:

```
(1000, 11)
```

Understanding your variables

- Using **.describe()** on an entire DataFrame we can get a summary of the distribution of continuous variables:

```
movies_df.describe()
```

OUT:

	rank	year	runtime	rating	
count	1000.000000	1000.000000	1000.000000	1000.000000	1.0(
mean	500.500000	2012.783000	113.172000	6.723200	1.69
std	288.819436	3.205962	18.810908	0.945429	1.88
min	1.000000	2006.000000	66.000000	1.900000	6.1(
25%	250.750000	2010.000000	100.000000	6.200000	3.6
50%	500.500000	2014.000000	111.000000	6.800000	1.10
75%	750.250000	2016.000000	123.000000	7.400000	2.3
max	1000.000000	2016.000000	191.000000	9.000000	1.79

- **.describe()** can also be used on a categorical variable to get the count of rows, unique count of categories, top category, and freq of top category:

```
movies_df['genre'].describe()
```

OUT:

```
count                          1000
unique                          207
top       Action,Adventure,Sci-Fi
freq                             50
Name: genre, dtype: object
```

- This tells us that the genre column has 207 unique values, the top value is Action/Adventure/Sci-Fi, which shows up 50 times (freq).

Pandas 102

- Datetime
- Timeshift
- Rolling & Expansion
- Time Resampling

Source: Type source image link here

Style

PEP 8

- Readability counts!

Source: Type source image link here

Code Layout

- Indentation
- Tabs or spaces
- Maximum length lines
- Blank lines
- Imports

Source: Type source image link here

PEP8

- Indentation

```
 4  # Aligned with opening delimiter.
 5  foo = long_function_name(var_one, var_two,
 6                           var_three, var_four)
 7
 8  # More indentation included to distinguish this from the rest.
 9  def long_function_name(
10          var_one, var_two, var_three,
11          var_four):
12      print(var_one)
13
14  # Hanging indents should add a level.
15  foo = long_function_name(
16      var_one, var_two,
17      var_three, var_four)
18
```

PEP8

- Indentation

```
34  if (this_is_one_thing and
35          that_is_another_thing):
36          do_something()
37
```

```
39  # Add a comment, which will provide some distinction in editors
40  # supporting syntax highlighting.
41  if (this_is_one_thing and
42          that_is_another_thing):
43          # Since both conditions are true, we can frobnicate.
44          do_something()
45
46  # Add some extra indentation on the conditional continuation line.
47  if (this_is_one_thing
48              and that_is_another_thing):
49          do_something()
```

PEP8

Indentation

```
56  my_list = [
57      1, 2, 3,
58      4, 5, 6,
59      ]
60  # Versus
61  my_list = [
62      1, 2, 3,
63      4, 5, 6,
64  ]
```

Equivalent, no specific recommendation

Maximum Length

- Maximum line length?
  - Coding lines? Keep it to 79 characters
  - Most editors can show you the line position
  - E.g. vim, Sublime
- Comments & doc strings?
  - 72 characters
- Why? My monitor is big!
  - Open two files side by side?
  - Some teams choose to use a different max
- Python core library is 79/72

Line Spacing

- Line spacing
  - Two blank lines around top level functions
  - Two blank lines around classes

- One blank line between functions in a class
- One blank line between logical groups in a function (sparingly)
- Extra blank lines between groups of groups of related functions (why are they in the same file?)

Imports

- Imports
    - *Some discussion of this already*
    - Imports go at the top of a file after any comments
    - Imports for separate libraries go on separate lines

```
71  import os
72  import sys
73  # Versus
74  import os, sys
```

Imports

- Imports should be grouped with a blank line separating each group in the following order:
  - Standard library imports
  - os, sys, …
  - Related third party imports
    - matplotlib, seaborn, numpy, etc…
  - Local application / library specific imports
    - knn_utils

```
1  import os
2  import sys
3
4  import pandas as pd
5  import numpy as np
6
7  from phylodist.constants import TAXONOMY_HIERARCHY, PHYLODIST_HEADER
8
```

Whitespace

- No trailing spaces at end of a line
  - Do not pad ([{ with spaces, e.g.

```
78   spam(ham[1], {eggs: 2})
79   spam( ham[ 1 ], { eggs: 2 } )
```

```
82   if x == 4: print x, y; x, y = y, x
83   if x == 4 : print x , y ; x , y = y , x
```

  - Do not pad before : ; , , e.g.

Whitespace

- Always surround =, +=, -=, == , < , > , != , <> , <= , >= , in , not in , is , is not, and, or, not with a single space

```
86   i = i + 1
87   submitted += 1
88   x = x*2 - 1
89   hypot2 = x*x + y*y
90   c = (a+b) * (a-b)
```

```
93   i=i+1
94   submitted +=1
95   x = x * 2 - 1
96   hypot2 = x * x + y * y
97   c = (a + b) * (a - b)
```

Whitespace

- Never surround = with a space as a function parameter argument

```
101   def complex(real, imag=0.0):
102       return magic(r=real, i=imag)
103
104
105   def complex(real, imag = 0.0):
106       return magic(r = real, i = imag)
```

Naming convention

- How you name functions, classes, and variables can have a huge impact on readability

```
1  from statistics import mean
2  import numpy.random as nprnd
3  from statistics import stdev
4  def MyFuNcTiOn(ARGUMENT):
5      m = mean(ARGUMENT)
6      s = stdev(ARGUMENT)
7      gt3sd = 0
8      lt3sd = 0
9      for m in ARGUMENT:
10         if m > m + (s * 2):
11             gt3sd += 1
12         elif m < m - (s * 2):
13             lt3sd += 1
14     return(gt3sd,lt3sd)
15 def AnotherFunction(anumber, anothernumber):
16     l = nprnd.randint(anothernumber, size = anumber)
17     return(MyFuNcTiOn(l))
18 a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19 print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```

Naming convention

- Avoid the following variable names:
  - Lower case L (l)
  - Upper case O (O)
  - Upper case I (I)

- There are unacceptable, terrible, and awful. Why?
  - Can be confused with 1 and 0 in some fonts
  - Can be confused with each other (i.e. I and l)

Naming convention

- Module names should be short, lowercase
  - Underscores are OK if it helps with readability

- Package names should be short, lowercase
  - Underscores are frowned upon and people will speak disparagingly behind your back if you use them

Naming convention

- Class names should be in CapWords
  - SoNamedBecauseItUsesCapsForFirstLetterInEachWord
  - Also known as CamelCase

- Notice no underscore!
- Much hate on the internet for
  - Camel_Case

Naming convention

- Functions
  - Lowercase, with words separated by underscores as necessary to improve readability
  - mixedCase is permitted if that is the prevailing style (some legacy pieces of Python used this style)
  - Easy habit to fall into… Very common in style guides for other languages, e.g. R
  - If this is your thing, then be consistent

Comments

- Shell script
  - #
- Python
  - #

Comments

- Good comments
  - Make the comments easy to read
  - Write the comments in English
  - Discuss the function parameters and results

```
211  % parameters:
212  %   sequence = character string of nucleotide letters (ATCG)
213  % returns:
214  %   geneStarts = vector of start index into sequence of start codon
215  %   geneEnds = vector of stop index into sequence of stop codons
216  %                value is the first base of the stop codon
217  function [ geneStarts, geneEnds ] = callGenesFromSequence(sequence)
218      ...
219  end
```

Comments

- These comments should also describe side effects
  - Any global variables that might be altered
  - Plots that are generated
  - Output that is puked

Comments

- Don’t insult the reader

```
240   % compute the square root of the square of the distance
241   distance = sqrt(distanceSquared);
```

- If they are reading your code… they aren’t that dumb
- Corollary: don’t comment every line!

```
187   # Find the square of the 3D distance.
188   distance_squared = (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2
189   # Compute the square root of the square of the distance.
190   distance = math.sqrt(distance_squared);
191   # Make sure the distance is less than 3.5 Angstroms or error.
192   if distance < 3.5:
193           # Throw an error.
194           error('interatomic distance is less than 3.5 Angstroms')
195   else:
196           # Add the distance to the list of distances.
197           distances.append(distance)
```

Comments

- Problems with this code
  - Randint – A list of random int [0 to anumber], and length of anumber

```
1  from statistics import mean
2  import numpy.random as nprnd
3  from statistics import stdev
4  def MyFuNcTiOn(ARGUMENT):
5      m = mean(ARGUMENT)
6      s = stdev(ARGUMENT)
7      gt3sd = 0
8      lt3sd = 0
9      for m in ARGUMENT:
10         if m > m + (s * 2):
11             gt3sd += 1
12         elif m < m - (s * 2):
13             lt3sd += 1
14     return(gt3sd,lt3sd)
15 def AnotherFunction(anumber, anothernumber):
16     l = nprnd.randint(anothernumber, size = anumber)
17     return(MyFuNcTiOn(l))
18 a,b=AnotherFunction(anumber = 1000, anothernumber = 1000)
19 print('found %d random values greather than 2 * sd and %d less than 2 * sd' % (a, b))
```

Comments

- Problems with this code (other than excessive comments?)
- What happens if I want to change the cutoff distance
  - I have to change the code (in 2 places)
  - I have to change the comment

```
187   # Find the square of the 3D distance.
188   distance_squared = (x2-x1)^2 + (y2-y1)^2 + (z2-z1)^2
189   # Compute the square root of the square of the distance.
190   distance = math.sqrt(distance_squared);
191   # Make sure the distance is less than 3.5 Angstroms or error.
192   if distance < 3.5:
193           raise Exception('distance violation',
194                           'interatomic distance is less than 3.5 Angstroms')
195   else:
196           # Add the distance to the list of distances.
197           distances.append(distance)
```

Docstrings

- They are triple quoted strings
  - What kind of quotes to use?

- They can be processed by the docutils package into HTML, LaTeX, etc. for high quality code documentation (that makes you look smart).

- They should be phrases (end in period).

Docstrings

- Multiline docstrings are more of the norm

```
"""
A multiline doc string begins with a single line summary (<72 chars).

The summary line may be used by automatic indexing tools; it is
important that it fits on one line and is separated from the rest of
the docstring by a blank line.
"""

import sys
```