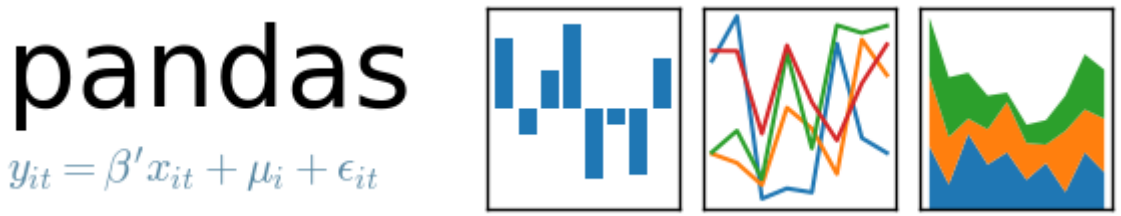
Sane's STATS Academy of Statistics

Package pandas

Working with Data Frames

Package pandas

- pandas is an open source, library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- Built by Wes McKinney based on package numpy



DataFrame Objects

- We are going to learn firstly two ways of creating a DataFrame object:
    1. From Dictionary
    2. From CSV file

Creating Data Frame from Dictionary

- Data frame object can be created from the dictionary using the method Data Frame on object of pandas

```
demog = { 'India': { 'area':3287263, 'population':1339180127 },
          'China': { 'area':9596961, 'population':1409517397 },
          'US': { 'area':9833520, 'population':324459463 },
          'Indonesia': { 'area':1904569, 'population':263991379 } }

import pandas as pd

topPop = pd.DataFrame(demog)
```

Index	China	India	Indonesia	US
area	9596961	3287263	1904569	9833520
population	1409517397	1339180127	263991379	324459463

Reading from CSV files

CSV file can be read using function read_csv() called on pandas object

Syntax : read_csv(“filepath”, sep=“,”,index_col,…)

Where sep : separator / delimiter

index_col : Column to use as the row labels of the DataFrame

```
iris = pd.read_csv("F:/Python Material/Python Course/Datasets/iris.csv")
```

Accessing Columns in DataFrame

- We can access columns by
  - Square brackets [ ]
  - Advanced Methods like **loc** and **iloc**

Square Brackets [ ] : Column Access

```
In [5]: topPop
Out[5]:
                China       India  Indonesia         US
area          9596961     3287263    1904569    9833520
population  1409517397  1339180127  263991379  324459463

In [6]: topPop["India"]
Out[6]:
area             3287263
population    1339180127
Name: India, dtype: int64

In [7]: topPop[["India"]]
Out[7]:
                 India
area           3287263
population  1339180127
```

```
In [8]: type(topPop["India"])
Out[8]: pandas.core.series.Series

In [9]: type(topPop[["India"]])
Out[9]: pandas.core.frame.DataFrame


In [10]: topPop[["India","China"]]
Out[10]:
                 India       China
area           3287263     9596961
population  1339180127  1409517397
```

- The class `pandas.core.series.Series` is equivalent to 1D array

Sane's STATS Academy of Statistics

Square Brackets [ ] : Row Access

```
In [11]: iris[2:4]
Out[11]:
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
```

```
In [17]: iris[:3]
Out[17]:
   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
```

Sane's STATS Academy of Statistics

Using loc : Row Access

- Using loc we can access the data by label

```
In [19]: topPop.loc["area"]
Out[19]:
China        9596961
India        3287263
Indonesia    1904569
US           9833520
Name: area, dtype: int64
```

```
In [20]: topPop.loc[["area"]]
Out[20]:
          China    India  Indonesia       US
area  9596961  3287263    1904569  9833520
```

```
In [21]: type(topPop.loc["area"])
Out[21]: pandas.core.series.Series
```

```
In [22]: type(topPop.loc[["area"]])
Out[22]: pandas.core.frame.DataFrame
```

```
In [4]: topPop.loc[["area","population"]]
Out[4]:
                 China       India  Indonesia       US
area           9596961     3287263    1904569  9833520
population  1409517397  1339180127  263991379  324459463
```

Sane's STATS Academy of Statistics

Using loc : Row & Column Access

```
In [5]: topPop.loc[["area","population"],["India","US"]]
Out[5]:
                India         US
area           3287263    9833520
population  1339180127  324459463
```

```
In [6]: topPop.loc[:,["India","US"]]
Out[6]:
                India         US
area           3287263    9833520
population  1339180127  324459463
```

Sane's STATS Academy of Statistics

Using iloc: Row & Column Access

- Using iloc, we need to specify the indices for rows and columns

```
In [9]: topPop
Out[9]:
                China       India  Indonesia         US
area          9596961     3287263    1904569    9833520
population  1409517397  1339180127  263991379  324459463
```

```
In [7]: topPop.iloc[[1],[1,3]]
Out[7]:
                India         US
population  1339180127  324459463

In [8]: topPop.iloc[:,[1,3]]
Out[8]:
                India         US
area          3287263    9833520
population  1339180127  324459463
```

Sane's STATS Academy of Statistics

Subsetting the DataFrames

- You can create a Boolean expression and pass the Boolean expression inside the [ ] of the DataFrame object to sub set it.

```
In [5]: iris[iris["Sepal.Width"] > 3.9]
Out[5]:
    Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
14           5.8          4.0           1.2          0.2  setosa
15           5.7          4.4           1.5          0.4  setosa
32           5.2          4.1           1.5          0.1  setosa
33           5.5          4.2           1.4          0.2  setosa
```

Applying logical NOT, AND and OR to columns

- We can apply logical NOT, AND and OR with `logical_not()`, `logical_and()` and `logical_or()` functions respectively from package `numpy`.

```
## Doesn't work in Python
iris["Sepal.Width"] > 3.5 & iris["Sepal.Length"] > 5.2

## Works
import numpy as np
np.logical_not(iris["Species"]=="setosa")
np.logical_and(iris["Sepal.Width"] > 3.5 , iris["Sepal.Length"] > 5.2)
np.logical_or(iris["Sepal.Width"] > 3.5 , iris["Sepal.Length"] > 5.2)
```

Handling Column Names

- For displaying the column names, we can call `.columns` attribute on the pandas data frame object

```
In [46]: iris.columns
Out[46]:
Index(['Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width',
       'Species'],
      dtype='object')
```

- For renaming the columns, we can call `.rename()` method on the pandas data frame object

```
In [47]: iris.rename(columns={'Sepal.Length': 'Sepal Length',
    ...:                      'Sepal.Width': 'Sepal Width',
    ...:                      'Petal.Length': 'Petal Length',
    ...:                      'Petal.Width':'Petal Width'},inplace=True)

In [48]: iris.columns
Out[48]:
Index(['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width',
       'Species'],
      dtype='object')
```

Sane's STATS Academy of Statistics

Knowing internal structure of Data Frame

- For knowing the internal structure of the pandas data frame, we require `.info()` method
- This is similar to `str()` in R

```
In [64]: print(iris.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Sepal.Length    150 non-null float64
Sepal.Width     150 non-null float64
Petal.Length    150 non-null float64
Petal.Width     150 non-null float64
Species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

Changing the column type

- We can change the column type using the method `.astype()`

```
In [65]: iris['Sepal.Width'] = iris['Sepal.Width'].astype(str)

In [66]: print(iris.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
Sepal.Length    150 non-null float64
Sepal.Width     150 non-null object
Petal.Length    150 non-null float64
Petal.Width     150 non-null float64
Species         150 non-null object
dtypes: float64(3), object(2)
memory usage: 5.9+ KB
```

category data type

- Python has a datatype category for handling categorical values (similar to factor in R)
- The method `reorder_categories()` can be used to re-order the categorical values

```
In [15]: iris['Species'] = iris['Species'].astype('category')

In [17]: iris['Species'].cat.categories
Out[17]: Index(['setosa', 'versicolor', 'virginica'], dtype='object')

In [18]: iris.Species.cat.reorder_categories(['virginica', 'setosa', 'versicolor' ], ordered=True)
    ...: iris['Species'].cat.categories
Out[18]: Index(['setosa', 'versicolor', 'virginica'], dtype='object')

In [21]: iris['Species'].cat.codes.unique()
Out[21]: array([0, 1, 2], dtype=int64)
```

Sane's STATS Academy of Statistics

Changing string to dates

- We can use the function `pandas.to_datetime()`

```
In [30]: ords['Order Date'] = pd.to_datetime(ords['Order Date'],format="%d-%b-%y")
```

- We can also directly parse date column with the function `pandas.read_csv(parse_dates=['column name'])`

```
In [32]: ords = pd.read_csv("G:/Statistics (Python)/Datasets/Orders.csv",parse_dates=['Order Date'])

In [33]: ords.dtypes
Out[33]:
Order ID                      object
Order Date            datetime64[ns]
Place of Shipment             object
Payment Terms                 object
dtype: object
```

Sane's STATS Academy of Statistics

Extracting the components of date

- Components of date can be extracted using `dt` attribute

```
In [4]: ords['year'] = ords['Order Date'].dt.year
   ...: ords['month'] = ords['Order Date'].dt.month
   ...: ords['day'] = ords['Order Date'].dt.day
   ...:

In [5]: ords.dtypes
Out[5]:
Order ID                      object
Order Date           datetime64[ns]
Place of Shipment             object
Payment Terms                 object
year                           int64
month                          int64
day                            int64
dtype: object

In [6]: ords.head()
Out[6]:
     Order ID Order Date Place of Shipment Payment Terms  year  month  day
0  32 90 001 2010-12-31              Pune         Cheque  2010     12   31
1  32 90 002 2011-01-06             Nasik         Online  2011      1    6
2  32 90 003 2011-01-14        Ahmednagar           Cash  2011      1   14
3  32 90 004 2011-02-18            Nanded         Cheque  2011      2   18
4  32 90 005 2011-02-19          Kolhapur           Cash  2011      2   19
```

Formatting Dates

Abbreviation	Specification
%d	Day as a number (01 - 31)
%a	Abbreviated weekday (Mon, Tue)
%A	Unabbreviated weekday (Monday, Tuesday, Wednesday)
%w	Weekday (0-6) 0-Sunday, 1-Monday
%W	Week (00-53) with Monday as first day of the week
%m	Month (01 – 12)
%b	Abbreviated month (Jan, Feb)
%B	Unabbreviated month (January, February)
%y	2 digit year
%Y	4 digit year

List Comprehensions

- With comprehensions, we can easily create lists or dictionaries using the feature of loop
- Comprehension is the best way to address a common programming task

```
In [50]: nums = [2,4,6,1,9]
    ...: for i in nums:
    ...:     print(i**2)
4
16
36
1
81

In [51]: Squares = [i**2 for i in nums]
    ...: print(Squares)
[4, 16, 36, 1, 81]
```

Sane's STATS Academy of Statistics

Dictionary comprehensions

```
In [54]: twitter_followers = [["@fchollet",105350],["@robjhyndman",7951],
    ...:                      ["@AndrewYNg",326407],["@teoliphant",19479]]
    ...: # Write a dict comprehension
    ...: tf_dict = {key:value for key,value in twitter_followers}
    ...:
    ...: # Print tf_dict
    ...: print(tf_dict)
{'@fchollet': 105350, '@robjhyndman': 7951, '@AndrewYNg': 326407, '@teoliphant':
19479}

In [55]: type(tf_dict)
Out[55]: dict
```

Sane's STATS Academy of Statistics

Control Structures

Decision and Loop

If else Structure

- All the control structures in Python rely heavily on indentation.
- It is necessary to indent the code which you want to put into code block.
- In R, Java, C, C++ etc., we apply { }. But in Python, we need to indent on the next line after putting a ":"

```
In [46]: rating1_5 = 4
    ...:
    ...: # if statement
    ...: if rating1_5 < 0:
    ...:     print("error")
    ...:
    ...: # elif statement
    ...: elif rating1_5 < 4:
    ...:     print("Not so Good")
    ...: # else statement
    ...: else:
    ...:     print('Excellent')
Excellent
```

If else Structure

- Python provides the following syntax for if and else structures

Syntax :

```
if condition1 :
        statements
elif condition2 :
        statements
elif condition3 :
        ….
else :
        statements
```

Examples

```
In [12]: x = 56
    ...: if x < 30 :
    ...:     print("Less")
    ...: elif x < 40 :
    ...:     print("Medium")
    ...: else :
    ...:     print("High")
    ...:
    ...:
High
```

```
In [11]: x = 34
    ...: if x < 30 :
    ...:     print("Less")
    ...: elif x < 40 :
    ...:     print("Medium")
    ...: else :
    ...:     print("High")
    ...:
    ...:
Medium
```

```
In [17]: x = 20
    ...: if x < 30 :
    ...:     print("Less")
    ...:     y = x+56
    ...: elif x < 40 :
    ...:     print("Medium")
    ...:     y = x+96
    ...: else :
    ...:     print("High")
    ...:     y = x-90
    ...:
    ...:
Less

In [18]: y
Out[18]: 76
```

Sane's STATS Academy of Statistics

Loop Control

Syntax :

while condition:

statements

- The statements in the while loop continue to execute so long as the condition remains true

```
In [20]: f = 97
    ...: while f < 100:
    ...:     print("Increasing..")
    ...:     f = f + 1
    ...:     print(f)
    ...:
    ...: print("Loop Over")
Increasing..
98
Increasing..
99
Increasing..
100
Loop Over
```

Loop Control

Syntax :

for var in seq:

statements/expressions

- The statements in the for loop continue to execute for the whole sequence of seq
- With a function enumerate(), we can handle the indices/iterators of the loop

```
In [4]: fam = [12.3,34.5,23.5,20.4]
   ...: for i in fam:
   ...:     print(i*i)
   ...:
   ...:
151.29000000000002
1190.25
552.25
416.15999999999997
```

```
In [9]: for index, height in enumerate(fam) :
   ...:     print("index " + str(index+1) + ": " + str(height))
   ...:
   ...:
index 1: 12.3
index 2: 34.5
index 3: 23.5
index 4: 20.4
```

Sane's STATS Academy of Statistics

Loop with List

```
In [12]: house = [["Hallway", 10.25],
    ...:          ["Kitchen", 19.0],
    ...:          ["Living Room", 20.0],
    ...:          ["Bedroom", 9.75],
    ...:          ["Bathroom", 9.55]]
    ...:
    ...: # Build a for loop from scratch
    ...: for x in house:
    ...:     print("the " + str(x[0]) + " is " + str(x[1]) + " sqm")
    ...:
    ...:
the Hallway is 10.25 sqm
the Kitchen is 19.0 sqm
the Living Room is 20.0 sqm
the Bedroom is 9.75 sqm
the Bathroom is 9.55 sqm
```

Sane's STATS Academy of Statistics

Loop with list

- Whenever, we want repeated execution of a code snippet, we use loop

```
In [47]: customers = [5, 4, 3, 3, 3, 5, 6, 10]
    ...:
    ...: # Write a for loop
    ...: for rating in customers:
    ...:     # if/else statement
    ...:     if rating < 4:
    ...:         print('Not So Good')
    ...:     else:
    ...:         print('Excellent')
Excellent
Excellent
Not So Good
Not So Good
Not So Good
Excellent
Excellent
Excellent
```

Looping on Dictionaries

- For looping over the dictionaries, we require items() method

```
In [18]: demog = { 'India': { 'area':3287263, 'population':1339180127 },
    ...:           'China': { 'area':9596961, 'population':1409517397 },
    ...:           'US': { 'area':9833520, 'population':324459463 },
    ...:           'Indonesia': { 'area':1904569, 'population':263991379 } }

In [19]: for key, value in demog.items() :
    ...:     print(key + " has " + str(value))
    ...:
    ...:
India has {'area': 3287263, 'population': 1339180127}
China has {'area': 9596961, 'population': 1409517397}
US has {'area': 9833520, 'population': 324459463}
Indonesia has {'area': 1904569, 'population': 263991379}
```

Looping on pandas data frame

- For iterating through rows in the pandas data frame, we can call function iterrows() on it

topPop - DataFrame

Index	China	India	Indonesia	US
area	9596961	3287263	1904569	9833520
population	1409517397	1339180127	263991379	324459463

```
In [20]: for col,row in topPop.iterrows():
    ...:     print(col + ": " + str(row["India"]))
area: 3287263
population: 1339180127
```

```
In [18]: for i in topPop:
    ...:     print(i)
China
India
Indonesia
US
```

```
In [19]: for col,row in topPop.iterrows():
    ...:     print(col)
    ...:     print(row)
area
China          9596961
India          3287263
Indonesia      1904569
US             9833520
Name: area, dtype: int64
population
China          1409517397
India          1339180127
Indonesia       263991379
US              324459463
Name: population, dtype: int64
```

Sane's STATS Academy of Statistics

Adding New Column Using Loop

topPop - DataFrame

Index	China	India	Indonesia	US
area	9596961	3287263	1904569	9833520
population	1409517397	1339180127	263991379	324459463

```
In [21]: for col,row in topPop.iterrows():
    ...:     topPop.loc[col,"TotalPop"] = row["China"]+row["India"]
+row["Indonesia"]+row["US"]
```

topPop - DataFrame

Index	China	India	Indonesia	US	TotalPop
area	9596961	3287263	1904569	9833520	2.46223e+07
population	1409517397	1339180127	263991379	324459463	3.33715e+09

Sane's STATS Academy of Statistics

Sane's STATS Academy of Statistics

Questions ?