I

Introduction to Pandas

Data Science Academy

Importing Pandas Library

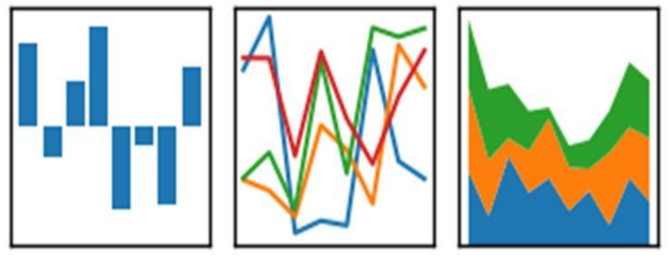
- Press Shift+Enter to execute the *jupyter* cell

```
In [ ]:  #Import Pandas Library
         import pandas as pd
```

Pandas

- Pandas is Python library that allows high-performance, easy-to-use data structures and data analysis tools.
- It’s an invaluable tool for data scientist and analysts
- The name stems from the term ‘panel data’ an econometrics term for multidimensional structured data sets.
- Pandas allows us to manipulate data frames (think of excel sheets or tables of data) and produce useful data outputs.

$$y_{it} = \beta' x_{it} + \mu_i + \varepsilon_{it}$$



Reading Data Using Pandas

```
Read csv file
df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

- There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

Table Without Pandas

- Simple Example – Imagine a 1000s of rows of data like the table below

Name	DOB	Subject	Exam scores
Lenord, robin	2001-02-22	Mathematis	71
Lenord, robin		Physics	64
⋮	⋮	⋮	⋮
khan, imran	2002-08-19	Spanish	76

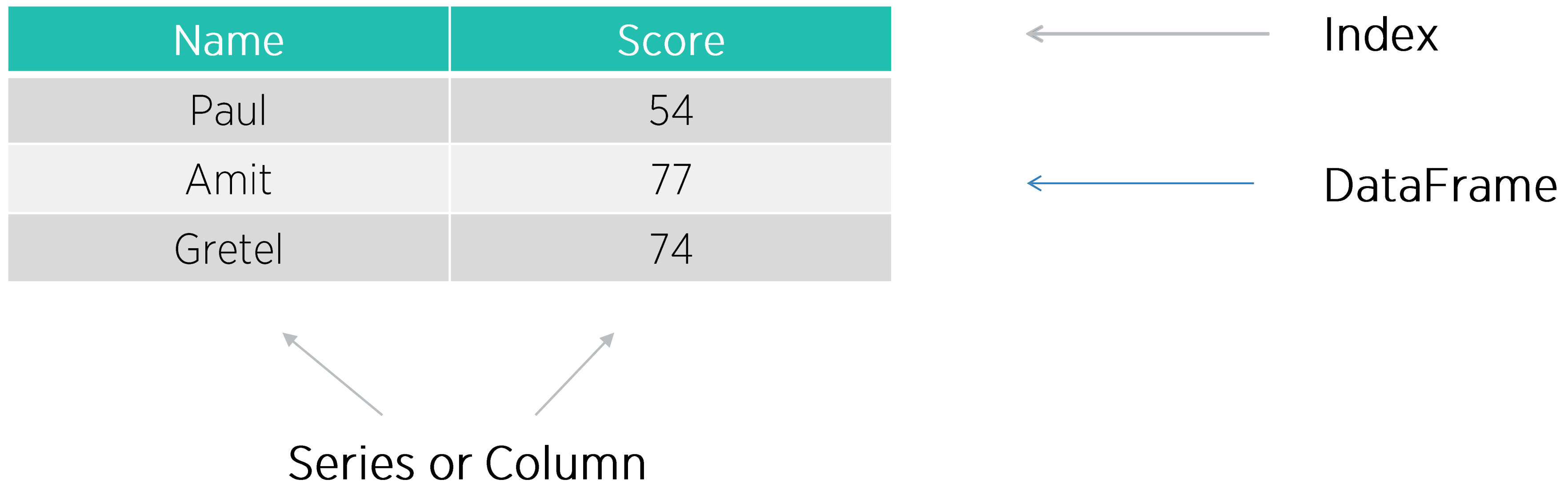
Table with pandas

- We can use pandas to produce this:

First Name	Age	General Subject Area	Overall Grade	Average Mark
Robin	18	Sciences	B+	65
Imran	17	Languages	B	62
⋮	⋮	⋮	⋮	⋮

Pandas

- Understanding Pandas Data Frames



Name	Score
Paul	54
Amit	77
Gretel	74

Selecting a column in a Data Frame

- *Method 1:* Subset the data frame using column name:

```
df[‘Age’]
```

- *Method 2*: Use the column name as an attribute:

```
df.age
```

II

Data Mining with Pandas

Data Science Academy

Data Frames attributes

- Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the column
columns	list the columns names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensioality
values	numpy representation of the data

Data Frames methods

- Unlike attributes, python methods have *parenthesis.*
- All attributes and methods can be listed with a *dir()* function: dir(df)

df.method()	description
head ([n]), tail([n])	first/last n rows
describe	generate descriptive statisitics (for numeric columns only
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Data Frames group by method

- Using method we can:
  - Split the dat "group by" a into groups based on some criteria
  - Calculate statistics (or apply a function) to each group
  - Similar to dplyr() function in R

```
                 phd    service         salary
rank
AssocProf  15.076923  11.307692   91786.230769
 AsstProf   5.052632   2.210526   81362.789474
     Prof  27.065217  21.413043  123624.804348
```

```
In [ ]:  #Group data using rank
         df_rank = df.groupby (['rank'])
```

```
In [ ]:  #Calculate mean value for each numeric column per each group
         df_rank.mean ()
```

Data Frames group by method

- Once groupby object is create we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:
        df_groupby(‘rank’)[[‘salary’]].mean()
```

```
                 salary
rank
AssocProf   91786.230769
 AsstProf   81362.789474
     Prof  123624.804348
```

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames group by method

- *groupby* performance notes:
  - No grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
  - by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

```
In [ ]:  #Calculate mean salary for each professor rank:
         df_groupby (('rank'), sort=False )[['salary']].mean()
```

Data Frame: Filtering

- To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 12000 ]
```

- Any Boolean operator can be used to subset the data:

> greater;	>= greater or equal;
< less;	<= less or equal;
== equal;	!= not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female']
```

Data Frame: Slicing

- There are a number of ways to subset the Data Frame:
    - one or more columns
    - one or more rows
    - a subset of rows and columns

- Rows and columns can be selected by their position or label

Data Frame: Slicing

- When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

```
In [ ]:  #Select column salary:
         df[‘salary’]
```

- **When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:**

```
In [ ]:  #Select column salary:
         df[[‘rank’,‘salary’]]
```

Data Frames: Selecting rows

- If we need to select a range of rows, we can specify the range using ":"

```
In [ ]: #Select rows by their position:
        df[10:20]
```

- Notice that the first row has a position 0, and the last value in the range is omitted:

So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

III
Working with Real Data

Data Science Academy

Data Frames: method loc

- If we need to select a range of rows, using their labels we can use method loc:

```
In [ ]:  #Select rows by their labels:
         df_sub.loc[10:20, [‘rank’, ‘sex’, ‘salary’]]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method loc

- If we need to select a range of rows and/or columns, using their positions we can use method iloc:

```
In [ ]:  #Select rows by their labels:
         df_sub.iloc[10:20, [0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0] # First row of a data frame
df.iloc[i] # (i+1) th row
df.iloc[-1]# Last row
```

```
df.iloc[:, 0]   # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]              #First 7 row
df.iloc[:, 0:2]           #First 2 columns
df.iloc[1:3, 0:2]         #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]] #1st and 6th rows and 2nd and 4th columns
```

Data Frame: Sorting

- We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

```
In [ ]:
# Create a new daa frame from the original sorted by the column Salary
df_sorted = df.sort_values(by = ‘service’)
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frame: Sorting

- We can sort the data using 2 or more columns:

In [ ]:

```
df_sorted = df.sort_values (by = [‘service’, ‘salary’], ascending = [True, False]
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have skipna option to control if missing data should be excluded . This value is set to True by default (unlike R)

Missing Values

- Missing values are marked as NaN

```
In [ ]:  # Read a dataset with missing values
         flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis
         /flights.csv")
```

```
In [ ]:  # lect the columns that have at least one missing value
         fligts[flights.isnull().any(axis=1)].head()
```

```
Out[ ]:
     year  month  day  dep_time  dep_delay  arr_time  arr_delay  carrier  tailnum  flight  origin  dest  air_time  distance  hour  minute
330  2013      1    1    1807.0       29.0    2251.0        NaN       UA   N31412    1228     EWR   SAN       NaN      2425  18.0     7.0
403  2013      1    1       NaN        NaN       NaN        NaN       AA   N3EHAA     791     LGA   DFW       NaN      1389   NaN     NaN
404  2013      1    1       NaN        NaN       NaN        NaN       AA   N3EVAA    1925     LGA   MIA       NaN      1096   NaN     NaN
855  2013      1    2    2145.0       16.0       NaN        NaN       UA   N12221    1299     EWR   RSW       NaN      1068  21.0    45.0
858  2013      1    2       NaN        NaN       NaN        NaN       AA      NaN     133     JFK   LAX       NaN      2475   NaN     NaN
```

Missing Values

- There are a number of methods to deal with missing values in the data frame:

df.method()	description
head ([n]), tail([n])	Drop missing observations
dropna	Drop observations where all cells is NA
dropna(how=‘all’)	Drop row if all the values are missing
dropna(axis=1, how=‘all’)	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull	returns True if the value is missing
notnull()	Returns True for non-missing values

Aggregation Functions in Pandas

- Aggregation - computing a summary statistic about each group, i.e.
  - compute group sums or means
  - compute group sizes/counts

- Common aggregation functions:
  - min, max
  - count, sum, prod
  - mean, median, mode
  - std, var

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
corr	Correlation between columns
skew	Sample skewness
kurt	Kurtosis

Aggregation Functions in Pandas

- agg() method are useful when multiple statistics are computed per column:

```
In [ ]: Flight [['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Data Frame: Sorting

In []:

```
#Sort dataset for salary
df.sort_values("salary")
```

Out[]:

	rank	discipline	phd	service	sex	salary
9	Prof	A	51	51	Male	57800
54	AssocProf	A	25	22	Female	62884
66	AsstProf	A	7	6	Female	63100
71	AssocProf	B	12	9	Female	71065
57	AsstProf	A	3	1	Female	72500
...	...	...	...	...	...	...
31	Prof	B	22	21	Male	155750
27	Prof	A	45	43	Male	155865
72	Prof	B	24	15	Female	161101
13	Prof	B	35	33	Male	162200
0	Prof	B	56	49	Male	186960

78 rows × 6 columns