wbs
WARWICK BUSINESS SCHOOL
THE UNIVERSITY OF WARWICK

For the Change Makers

Programming for Data Analytics

Week 6 : Data Processing
Information Systems and Management
Warwick Business School

Agenda Revisited

1) Data collection
   - ➢ Web scraping with Python; SQL and BigQuery; API and JSON
2) Data visualization
   - ➢ Matplotlib and Seaborn; Tableau
3) *Data wrangling
   - ➢ Cleaning, process, transformation (Numpy, Pandas, Regular Expression)
4) Machine learning
   - ➢ *Clustering*, classification and regression (Scikit-learn).
5) Deep learning:
   - ➢ Architecture design, network tuning (PyTorch).

Pandas

- Pandas is a Python library for data manipulation and analysis. The name Pandas comes from Python Data Analysis Library ("Panel Data" from Wiki), a bit like Excel.
- It depends on many other libraries, such as Numpy and Matplotlib.
- To some extent, it can be seen as a specialized Numpy library mostly dealing with 2-D array with many useful data manipulation functionalities.
- It is probably the most widely used data-processing library for Python.

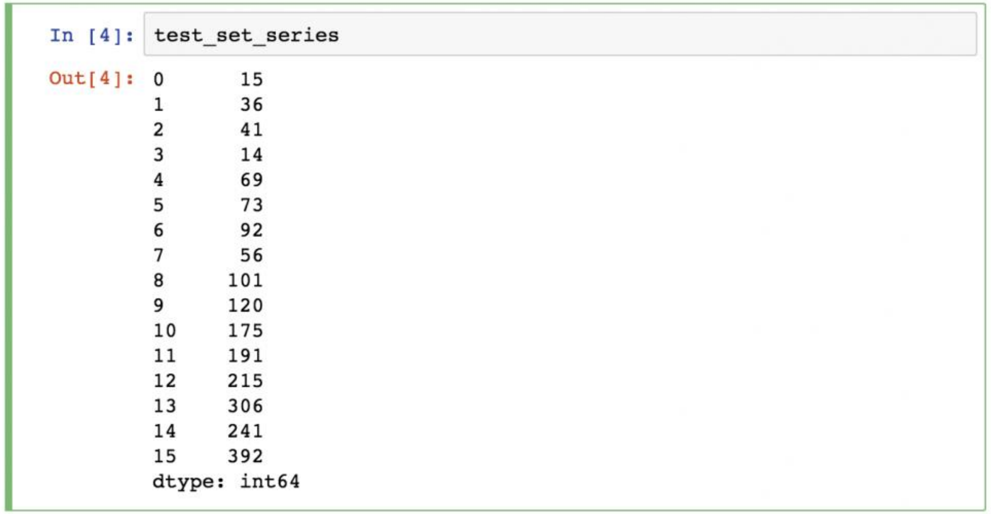
Data structure

- There are two types of data structures in pandas: Series and DataFrames.
- **Series**: one dimensional data structure ("a one dimensional ndarray"), and for every value it holds a unique **index**.
- **DataFrame:** a two dimensional data structure – basically a table with rows and columns. The columns have names and the rows have **indexes**.

1. A series can be created using pandas function Series with python **list** or numpy **1-D array** as the argument. By default, each item will receive an numeric index label starting from 0.

```
>>>s1 = pd.Series([1,2,3])
```

```
>>>s2 = pd.Series(np.array([1,2,3,4,5]))
```



```
In [4]: test_set_series

Out[4]: 0       15
        1       36
        2       41
        3       14
        4       69
        5       73
        6       92
        7       56
        8      101
        9      120
        10     175
        11     191
        12     215
        13     306
        14     241
        15     392
        dtype: int64
```

Manually creating a Series data

1. An explicit index can also be specified when creating the series by providing the index with a **list** as the second argument. This is often called label.

```
>>>s3 = pd.Series([1,2,3,'a','b','c'],
                  index=['A','B','C','D','E','F'])
```

2. When a dictionary is provided as the argument, the key will be used as the index.

```
>>> s4 = pd.Series({'A':1,'B':2,'C':3})
```

3. Each index label needs to be unique?

Indexing and slicing Series Data

1. Data in the series can be accessed similar to that in a Python list when having the default numeric index.

```
s2[2]
s2[:2]  # return a series data.
```

2. Data in the series can be accessed similar to that in a Python dictionary when having specified index label.

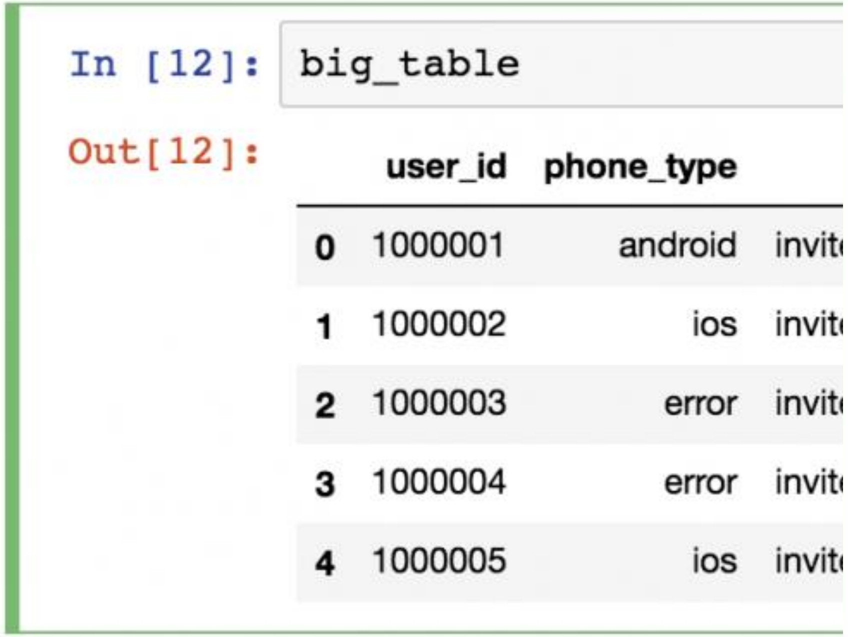
```
s3['A']
```

3. You can retrieve multiple data by providing a **list** of "keys"/labels.

```
s3[['A','B','C']]
```

Dataframe

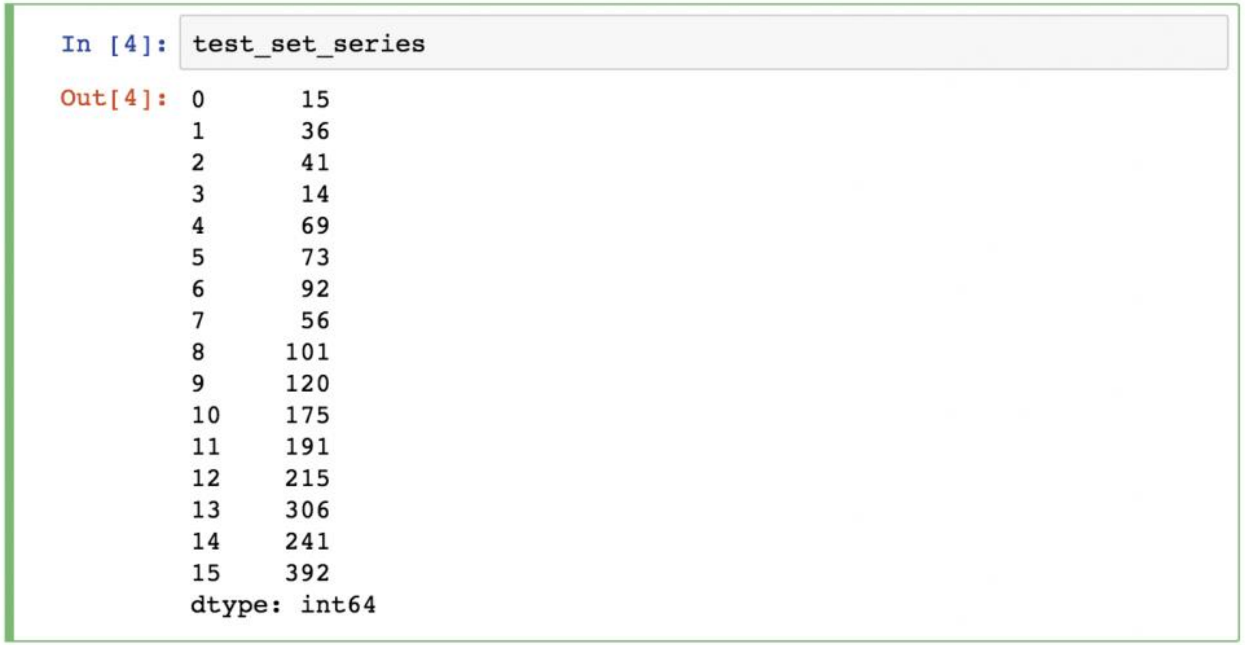
- A DataFrame has labeled axes (rows and columns) and can be created by pandas function: *DataFrame()*



```
In [12]: big_table
```

Out[12]:

	user_id	phone_type	
0	1000001	android	invit
1	1000002	ios	invit
2	1000003	error	invit
3	1000004	error	invit
4	1000005	ios	invit



```
In [4]: test_set_series

Out[4]: 0        15
        1        36
        2        41
        3        14
        4        69
        5        73
        6        92
        7        56
        8       101
        9       120
        10      175
        11      191
        12      215
        13      306
        14      241
        15      392
        dtype: int64
```

Create DataFrame from a dictionary

- You can create a DataFrame from **dictionary** of narrays/lists/series. Keys will be used as the column labels by default. Values become the columns corresponding to the key. Handy for dealing with JSON data.

```
>>>d1 = pd.DataFrame({'A':[1,2,3],'B':[2,3,4]})
```

- You may also specify index label for the rows with argument index.

```
>>>d2 = pd.DataFrame({'A':[1,2,3],'B':[2,3,4]},index=['X','Y','Z'])
```

```
>>>d3 = pd.DataFrame({'A':np.array([1,2,3]),
                     'B':[2,3,4]}, index=['X','Y','Z'])
```

```
>>> d4 = pd.DataFrame({'A':[1,2,3],'B':s1})#s1 is a series
```

Create DataFrame from a dictionary

- Note: Items in the dictionary must have the same length unless they are all series.

```
>>>d5 = pd.DataFrame({'A' : [1,2,3], 'B' :[2,3,4,5]}) #error
>>>d6 = pd.DataFrame({'A' : s1, 'B' :[1,2]}) #error
```

- When series have different length, Python will try to match their index to create the dataframe and NaN (Not a Number) is appended in missing areas.

```
>>>d7 = pd.DataFrame({'A' : s1, 'B' :s2}) #using default numeric index
>>>d8 = pd.DataFrame({'A' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'B' : pd.Series([1, 2, 3, 4], index=['b', 'c', 'd', 'e'])})#using specified index
```

Create DataFrame from a list

- A DataFrame can be created using a single list or a list of lists.

```
>>> d9 = pd.DataFrame([1,2,3,'a','b','c']) #compare with s1.
>>> d10 = pd.DataFrame([[1,2,3],[2,3,4],[3,4,5]])
```

- Numeric labels will be created for row and column by default. You can also specify the labels for columns and index (row).

```
>>> d11 = pd.DataFrame([[1,2,3],[2,3,4],[3,4,5]],columns=['A','B','C'])
>>> d12=pd.DataFrame([[1,2,3],[2,3,4],[3,4,5]], columns=['A','B','C'],
                      index=['X','Y','Z'])
```

Create DataFrame from a list

- You can create a DataFrame from a list of dictionaries. Keys will be used as the column labels by default.

```
>>>d13 = pd.DataFrame([{'a': 1, 'b': 2},{'a': 5, 'b': 10}])
```

```
>>>d14 = pd.DataFrame([{'a': 1, 'b': 2},{'a': 5, 'b': 10}],index=['A','B'])
```

- Each item in the list is like a row in a table. Items in the list can have different length.

List of elements with different lengths

- When no specific column label is provided, Python will match the default labels (number index or keys) to create the dataframe and NaN is appended in missing areas.

```
>>> d15 = pd.DataFrame([[1,2],[2,3],[3,4,5]])
```

```
>>> d16= pd.DataFrame([{'a': 1, 'b': 2},{'b': 5, 'c':
10,'d':15}])
```

- When column labels are specified, Python will create DataFrame based on the column labels and try to match keys with the labels. Values with non-match keys will be ignored.

```
>>>d17 = pd.DataFrame([{'a': 1, 'b': 2},{'b': 5,
'c':10,'d':15}], columns=['b','d','e'])
```

Create DataFrame from CSV files

- Pandas can read data directly from a wide range of file formats, such as csv, Excel, JSON, SQL database, Stata, SAS, etc. We will focus on csv files in this class.
- Use *read_csv()* function. Filename is the only required argument.

```
df_tips = pd.read_csv('tips.csv')
```

- ❖ Many optional arguments can be passed when importing data.
- ❖ https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Key parameters of read_csv()

- **delimiter**: comma by default, set when necessary.
- **header**: row to be used as the column headers, and the start of data. 0 by default (the first row). Set to None if no header.

```
df_tips = pd.read_csv('tips.csv',header=None)
```

- **names**: a list of column names to be used instead header.

```
df_tips = pd.read_csv('tips.csv',header=None,names=[1,2])
```

- **index_col**: specify a column to be used for row index.

```
df_tips = pd.read_csv('tips.csv',index_col=0)
```

Row index can also be specified with column name

```
df_tips = pd.read_csv('tips.csv',index_col='tips')
```

Key parameters of read_csv()

- **usecols**: import selected columns by passing a list of column **index or names**.

```
columns = [1,2,3] #columns = ['tip', 'sex', 'smoker']
df_tips = pd.read_csv('tips.csv',usecols=columns)
```

- **skiprows**: specify the number of first n rows to skip.
- **nrows**: specify the total number of rows to read. Useful when reading large files.

```
df_tips = pd.read_csv('tips.csv',skiprows=3, nrows=10)
```

Key parameters of read_csv()

- na_values: list of **strings** to be treated as NaN. Most common ones can be detected automatically, such as #N/A, n/a, null, etc.

```
missing = ['not available', 'missing']
df_tips = pd.read_csv('tips.csv',na_values=missing)
```

Create DataFrame from JSON files

- JSON data can also be imported into DataFrame directly with .read_json().
- It works best if your JSON data doesn't have complex structure.
- It does not handle NaN value well.

Conversion between DataFrame and ndarray

DataFrame and ndarray can be easily converted.

- From ndarray to DataFrame with DataFrame()

```
df_nd = pd.DataFrame(ndarray)
```

You may also optionally provide labels and index

```
df_nd = pd.DataFrame(ndarray, columns=['a','b'],
index = [1, 2])
```

Caution: ndarray is uni-typed.

Conversion between DataFrame and ndarray

- From DataFrame to ndarray with .to_numpy() method.

```
ndarray2 = df_nd.to_numpy()
```

- Labels and index will be ignored, data types will be unified.

Basic operations

- Basic arithmetic and Boolean operations with scalar data are element-wise.
- `df_tips * 2` #broadcasting
- `df_tips.add(2)`
- `df_tips >2`

Python Operator	Pandas Method(s)
+	add()
-	sub(), subtract()
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

More operations

- Arithmetic and Boolean operations with another list or Series will be performed based on matching labels (columns).

```
d10 = pd.DataFrame([[1,2,3],[3,4,5],[5,6,7]])
d10 - [1,2,3] #default to compare by column.
d10 > [3,3,3]
d10 - [1,2]#error, different length
d10 - pd.Series([1,2]) # NaN for no-match column.
```

Column selection and deletion

- Column selection using the column label:

```
>>>print(df_tips['tip']) #column label as the key, return a series.
>>>print(df_tips[['tip']]) #return a dataframe.
```

- Add new column with label, similar as adding new item to a dictionary:

```
>>> df_tips['f'] = pd.Series([10,10]) #series/list/narray
```

- New column can be added by calculating existing columns:

```
>>> df_tips['total'] = df_tips['total_bill'] + df_tips['tip']
#NaN if one cell is Nan.
```

- del to delete a column:

```
>>>del df_tips['f']
```

Row Selection

- Row selection by passing row **labels** to loc[] method. The row will be returned as a series or a dataframe:

```
>>> df_tips.loc[1] #column labels will be used as row index.
```

- Multiple rows can be selected:

```
>>> df_tips.loc[[1,2,3]] #a list of row indexes/labels, returns a dataframe
>>> df_tips.loc[1:3] #slice, both start and end included.
```

- Column labels can be provided to filter the results:

```
>>> df_tips.loc[[1,2,3],'tip']
```

- Select rows with Boolean list indicating whether to be selected:

```
>>> df_tips.loc[[True,False,False,True,False]]#same length as #row.
```

- Select rows with Boolean expression passed as series:

```
>>> df_tips.loc[df_tips['tips'] > 2]
>>> df_tips.loc[df_tips['tips'] > 2,'total']# only display column 'total'
```

You may also use:

```
>>> df_tips[df_tips['tip'] > 2]
```

Exercise

- Create a DataFrame using the data file 'all_games.csv', make sure you use the data header as the column labels and convert irregular data into NaN.