Data Analysis with Pandas

Session 5

Outline

- Data structures in Pandas: Series and DataFrame
- Data loading/exporting
- Essential functionality
- Computing descriptive statistics

a collection of functions and methods that allows you to perform many actions without writing your own code

pandas

- **pandas** is a library that provides high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- To use pandas, we have to import it first.
- We use the following import conventions for pandas:

```
1  import pandas as pd
```

- Whenever we see **`pd.`** in code, it's referring to pandas.

pandas

- There are two main data structures in pandas:
  - **Series** – array-like data structure
  - **Data Frames** – spreadsheet-like data structure
- Series and data frames are used so much that it is easier to import them into the local namespace:

```
1  from pandas import Series
```

```
1  from pandas import DataFrame
```

pandas

- **Methods**: functions bound to an object

```
object.method()
object.method(parameters)
```

- **Attributes**: are like methods but, instead performing any action, they just give us more information about the object.

```
object.attribute
```

Series

Series

- A **series** is a one-dimensional object containing an array of data and an associated array of data labels, called its index.

1	obj
0	4
1	7
2	-5
3	2

Creating a Series

- *Import Series from pandas:*

```
1  from pandas import Series
```

- Create a series from a `list`

```
1  obj = Series([4,7,-5,2])
```

```
1  obj
```

```
0    4
1    7
2   -5
3    2
```

- The index is shown on the left and the values on the right.
- We can choose the index names

```
1  obj2 = Series([4,7,-5,3], index=["d", "b", "a", "c"])
```

```
1  obj2
```

```
d    4
b    7
a   -5
c    3
```

Creating a Series (cont.)

- A series can also be created from a **`dict`**.

```
1  ddata = {'Texas':71,'Oregon':16,'Utah':5}
2  sdata = Series(ddata)
```

```
1  sdata
```

```
Oregon    16
Texas     71
Utah       5
```

keys become index names

dict values become Series values

Series attributes

```
1  obj
0    4
1    7
2   -5
3    2
```

```
1  obj2
d    4
b    7
a   -5
c    3
```

- `df.values`
  Returns the values of a DataFrame

```
1  obj.values
```

```
array([ 4,  7, -5,  3])
```

```
1  obj2.values
```

```
array([ 4,  7, -5,  3])
```

- `df.index`
  Retrieves the index labels

```
1  obj.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

```
1  obj2.index
```

```
Index(['d', 'b', 'a', 'c'], dtype='object')
```

Operations on Series

```
1  obj = Series([4,7,-5,2])
```

- Indexing and slicing work for series.
  - Indexing:

```
1  obj[2]
```

```
-5
```

  - Slicing:

```
1  obj[1:3]
```

```
1    7
2   -5
```

- We can directly change elements:

```
1  obj[1] = 3
```

```
1  obj
```

```
0    4
1    3
2   -5
3    2
```

Series

Activity 1: Series

DataFrame

DataFrame

- A **DataFrame** represents a spreadsheet-like data structure containing an ordered collection of columns.
- Each column can be a different value type (numeric, string, boolean, etc.).
- A DataFrame has both a row index and a column index.
- It can be thought of as a dictionary of Series, all sharing the same index.

column index

	store	sales	skus
0	Bo	2500	100
1	Sp	1200	50
2	Ca	2200	80

row index

Creating a DataFrame

- *Import DataFrame from pandas:*

```
1  from pandas import DataFrame
```

- Constructing a DataFrame from a dictionary of equal length lists:

```
1  ddata = {'store':['Bo','Sp','Ca'],
2           'sales':[2500,1200,2200],
3           'skus':[100,50,80]}
4  data = DataFrame(ddata)
```

	sales	skus	store
0	2500	100	Bo
1	1200	50	Sp
2	2200	80	Ca

- Constructing a DataFrame from a list of lists:

```
1  ldata = [['Bo',2500,100],['Sp',1200,50],['Ca',2200,80]]
2  data = DataFrame(ldata,columns=['store','sales','skus'])
```

DataFrame attributes

	store	sales	skus
0	Bo	2500	100
1	Sp	1200	50
2	Ca	2200	80

- `df.index`

  Retrieves the index labels

- `df.columns`

  Retrieves the column names

- `df.values`

  Returns the values of a DataFrame

```
1  data.index
```

```
RangeIndex(start=0, stop=3, step=1)
```

```
1  data.columns
```

```
Index(['store', 'sales', 'skus'], dtype='object')
```

```
1  data.values
```

```
array([['Bo', 2500, 100],
       ['Sp', 1200, 50],
       ['Ca', 2200, 80]], dtype=object)
```

Operations on DataFrame

- A column can be retrieved as a Series either by **`dict`**-like notation or by attribute notation.

	sales	skus	store
0	2500	100	Bo
1	1200	50	Sp
2	2200	80	Ca

dict-like notation

```
1  data['sales']
```

```
0    2500
1    1200
2    2200
```

attribute notation

```
1  data.sales
```

```
0    2500
1    1200
2    2200
```

- Columns can be modified by assignment. Assigning a column that doesn't exist will create a new column.

```
1  data['employees'] = 10
```

	store	sales	skus	employees
0	Bo	2500	100	10
1	Sp	1200	50	10
2	Ca	2200	80	10

Operations on DataFrame (Cont.)

Converting to list:

- `df.columns.tolist()`
  Converts column names to list.
- `df['column'].tolist()`
  Converts column values to list.
- `df.values.tolist()`
  Converts all data values to list.

	store	sales	skus
0	Bo	2500	100
1	Sp	1200	50
2	Ca	2200	80

```
1  data.columns.tolist()
```

```
['store', 'sales', 'skus']
```

```
1  data['sales'].tolist()
```

```
[2500, 1200, 2200]
```

```
1  data.values.tolist()
```

```
[['Bo', 2500, 100], ['Sp', 1200, 50], ['Ca', 2200, 80]]
```

DataFrame

Activity 2: DataFrame

Data loading/exporting

Reading data from Excel

- We will be using **`read_excel`** function to read an Excel file into a pandas DataFrame.

```
pd.read_excel('path/filename.xlsx')
```

```
1 data = pd.read_excel("/Users/inma/Dropbox (MIT)/stores.xlsx")
```

- Additional parameters:
  - **`sheet_name ='sheet name'`** Sheet name, default 1st sheet
  - **`header = 0`** Column labels, default 1st row
    `None` when no header
  - **`names = ['col1', 'col2']`** List of column names to use
  - **`index_col = 'col1'`** Column to use as row index
    Default **`None`**

Reading data from other formats

- From a CSV file:

```
pd.read_csv('path/filename.csv')
```

```
1 data = pd.read_csv("/Users/inma/Dropbox (MIT)/stores.csv")
```

- From a TXT file:

```
pd.read_table('path/filename.txt')
```

```
1 data = pd.read_table("/Users/inma/Dropbox (MIT)/stores.txt", sep = ",")
```

Delimiter used to separate values

Viewing and inspecting data

- **`df.head(n)`**
  Display the first n rows
- **`df.tail(n)`**
  Display the last n rows
- **`df.shape`**
  Returns the number of rows and columns as a tuple
- **`df.info()`**
  Returns index, columns, datatypes

```
1  data.head()
```

	Name	Address	City	State	State_name	Zip	Sales	Number_emp	Empsiz
0	SCENTSIBILITIES	750 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	A
1	MODEL DAIRY	469 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	A
2	DUNKIN' DONUTS	270 SHREWSBURY ST	BOYLSTON	MA	Massachusetts	1505	840	15	C
3	MOBILE STATION	22 MAPLE AVE	SHREWSBURY	MA	Massachusetts	1545	896	4	A
4	HORTON'S SUPER MARKET	117 CLINTON ST	SHREWSBURY	MA	Massachusetts	1545	494	2	A

```
1  data.shape
```

```
(4644, 9)
```

```
1  data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4644 entries, 0 to 4643
Data columns (total 9 columns):
Name          4644 non-null object
Address       4609 non-null object
City          4644 non-null object
State         4644 non-null object
State_name    4644 non-null object
Zip           4644 non-null int64
Sales         4644 non-null int64
Number_emp    4644 non-null int64
Empsiz        4644 non-null object
dtypes: int64(3), object(6)
```

Data types:

- string — object
- int — int64
- float — float64

in pandas

Saving to a file

- We can use `to_excel` or `to_csv` functions to save a DataFrame as an Excel sheet or CSV file, respectively.

```
df.to_excel('path/filename.xlsx')
```

```
1  data.to_excel('Data_stores.xlsx')
```

- The location on the Excel file will be the same folder as the notebook, unless specified otherwise.
- Additional parameters:
  - `sheet_name = 'sheet1'` Sheet name
  - `index = False` Writes row index, default `True`

Data loading and exporting

Activity 3: Data loading/exporting

Essential functionality

Indexing and selection

Indexing allows to select specific columns and/or rows.

- `df['col']`
  Returns column *col* as Series.
- `df[['col1','col2']]`
  Returns columns as a new DataFrame.
- `df.iloc[i,:]`
  Returns row *i*.
- `df.iloc[i,j]`
  Returns data value of row *i* and column *j*.

```
1  data['Sales']
```

```
1  data[['Name','Sales']]
```

```
1  data.iloc[3,:]
```

```
1  data.iloc[4,3]
```

Filtering

- `df[df['col']>x]`

  Returns rows where column *col* is greater than x.

```
1  data[data['Sales']>2000]
```

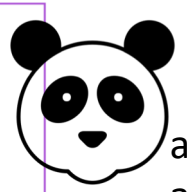
- `df[(df['col1']>x) & (df['col2']<y)]`

  Returns rows where column *col1* is > x and *col2* is < y.

```
1  data[(data['Sales']>2000) & (data['Number_emp']>100)]
```

Operators:

>	greater than	<=	less or equal to	&	and
>=	greater or equal to	==	equal	|	or
<	less than	!=	not equal		



and, or operators are different than usual

Adding/deleting columns

- We can create new columns based on other columns:

```
1  data['Sales_per_employee'] = data['Sales']/data['Number_emp']
```

	Name	Address	City	State	State_name	Zip	Sales	Number_emp	Empsiz	Sales_per_employee
0	SCENTSIBILITIES	750 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	A	115.0
1	MODEL DAIRY	469 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	A	115.0
2	DUNKIN' DONUTS	270 SHREWSBURY ST	BOYLSTON	MA	Massachusetts	1505	840	15	C	56.0

- The `del` keyword will delete columns:

```
1  del data['Empsiz']
```

	Name	Address	City	State	State_name	Zip	Sales	Number_emp	Sales_per_employee
0	SCENTSIBILITIES	750 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	115.0
1	MODEL DAIRY	469 MAIN ST	BOYLSTON	MA	Massachusetts	1505	115	1	115.0
2	DUNKIN' DONUTS	270 SHREWSBURY ST	BOYLSTON	MA	Massachusetts	1505	840	15	56.0

Indexing, selection and filtering

jupyter

Activity 4: Indexing, selection, and filtering

Sorting and ranking

- `df.sort_values('col')`
  Sorts values by column *col* in ascending order.

```
1  data.sort_values('Sales')
```

- `df.sort_values('col', ascending=False)`
  Sorts values by column *col* in descending order.

```
1  data.sort_values('Sales',ascending=False)
```

- `df.sort_values(['col1','col2'],ascending=[True,False])`
  Sorts values by column *col1* in asc order then *col2* in desc order.

```
1  data.sort_values(['Sales','Number_emp'],ascending=[True,False])
```

Sorting and ranking

Activity 5: Sorting and ranking

Computing Descriptive Statistics

Statistical functions

- pandas is equipped with a set of common statistical functions.
- General statistical functions:
  - `df['col'].count()`
    Returns the number of non null-values in column *col*
  - `df['col'].sum()`
    Returns the sum of values of numeric column *col*

Statistical functions (Cont.)

- Measures of central tendency:
  - `df['col'].mean()`
    Returns the mean of numeric column *col*
  - `df['col'].median()`
    Returns the median of numeric column *col*
  - `df['col'].mode()`
    Returns the mode of column *col*

Statistical functions (Cont.)

- Measures of dispersion:
  - `df['col'].min()/df['col'].max()`
    Returns the lowest/highest value in numeric column *col*
  - `df['col'].idxmin()/df['col'].idxmax()`
    Returns the row (index) at which min/max is obtained
  - `df['col'].var()`
    Returns the sample variance of values of numeric column *col*
  - `df['col'].std()`
    Returns the sample standard deviation of values of column *col*

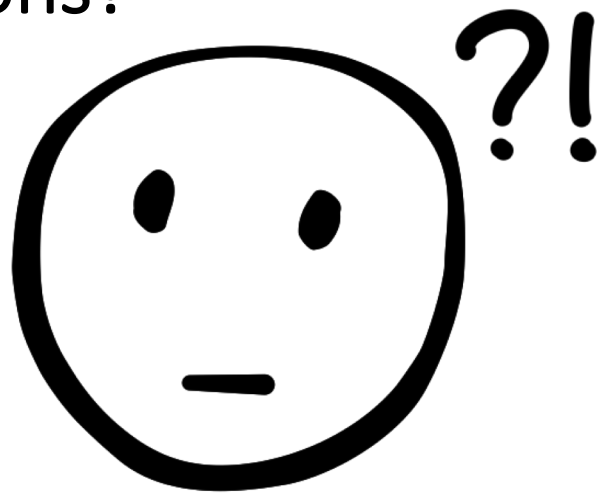
Descriptive statistics

Activity 6: Computing descriptive statistics

Today we learned:

- Data structures in Pandas
- Data loading/exporting
- Essential functionality
- Computing descriptive statistics

Any questions?



References

McKinney, W. (2013). Python for data analysis. O'Reilly Media, Inc.

Thanks!

Inma Borrella, Ph.D.
inma@mit.edu

Problem 5

HSBD Bank has hired Mario, a data analyst, to do some data analysis to help the company to better understand the customers' demographic characteristics. The upper management has some questions, and Mario must find the answer by analyzing the data available.

To start, please run the cell below to get the data provided by the bank (it might take a few seconds to finish running).

```
# Run this cell but DO NOT MODIFY
# Do not modify this part, it belongs to the question
#-------------------------------------------------------------------------------
import pandas as pd
url='https://raw.githubusercontent.com/juliencohensolal/BankMarketing/master/rawData/bank-a
data = pd.read_csv(url,sep=";") # use sep="," for coma separation.
Age = data['age']
data = data[['age', 'job', 'marital']].copy()
#-------------------------------------------------------------------------------
```

5.1

The series *Age* (defined in the cell above) contains information about the customers' age.

Write a script that prints the customers' age every 5,000th customers. This means printing the value associated to the 5,000th element in the series, the 10,000th element, the 15,000th element, and so on.

Tip: Use a loop.

The output should have the following format:

The age of the 5000 th customer is 44

The age of the 10000 th customer is 28

The age of the 15000 th customer is 34

...etc...

5.2

The dataframe *data* (defined in the cell above) contains information regarding customers' age, job, and marital status.

- Print the name of all the columns in the dataframe
- Print the shape of the dataframe to understand the size of *data*
- In one line of code, print the average, the median, the maximum and the minimum age of the customers

```
1 # Print the name of all the columns in the dataframe
2
3 # Print the shape of the dataframe to understand the size of data
4
5 # In one line of code, print the average, the median, the maximum and the minimum age of th
```

5.3

- Using the maximum age of the customers, create a new column called normalized_age that contains the age of each customer divided by the maximum age.
- Print the job information for all customers between index 500 and index 515 (including 500 and 515).
- Delete the marital column from the dataframe
- Dave the dataframe as an excel file to your local computer and name it Problem5.

Upload both this Jupyter notebook and the excel file you just saved to Stellar.

```
1  # Create a new column called normalized_age that contains the age of each customer divided
2
3  # Print the job information for all customers between index 500 and index 515 (including 50
4
5  # Delete the marital column from the dataframe
6
7  # Save the dataframe as an excel file to your local computer and name it Problem5.
```