QuantUniversity, LLC
www.quantuniversity.com

Data pre-processing/Wrangling for Analytics

Presented By:
Sri Krishnamurthy, CFA, CAP
www.QuantUniversity.com
sri@quantuniversity.com

Why data preprocessing/Wrangling?

- Data stored in enterprises are typically gathered from multiple sources.
- Datasets typically have large number of records or number of variables.
- Datasets may be incomplete, noisy and inconsistent leading to low quality results when mining for information.
- If data isn't pre-processed, it won't be suitable for analytics and worse, quantitative techniques won't work
- Examples include : Division by zero, Multiplication by zero, erroneous values(text instead of numbers etc.)

Data preprocessing for analytics

- Data preprocessing for analytics include:
  - Data ingestion
  - Merging Data sources
  - Data cleansing and manipulation
  - Other data transformations
  - Data reduction

These techniques are not mutually exclusive and they may be done concurrently.

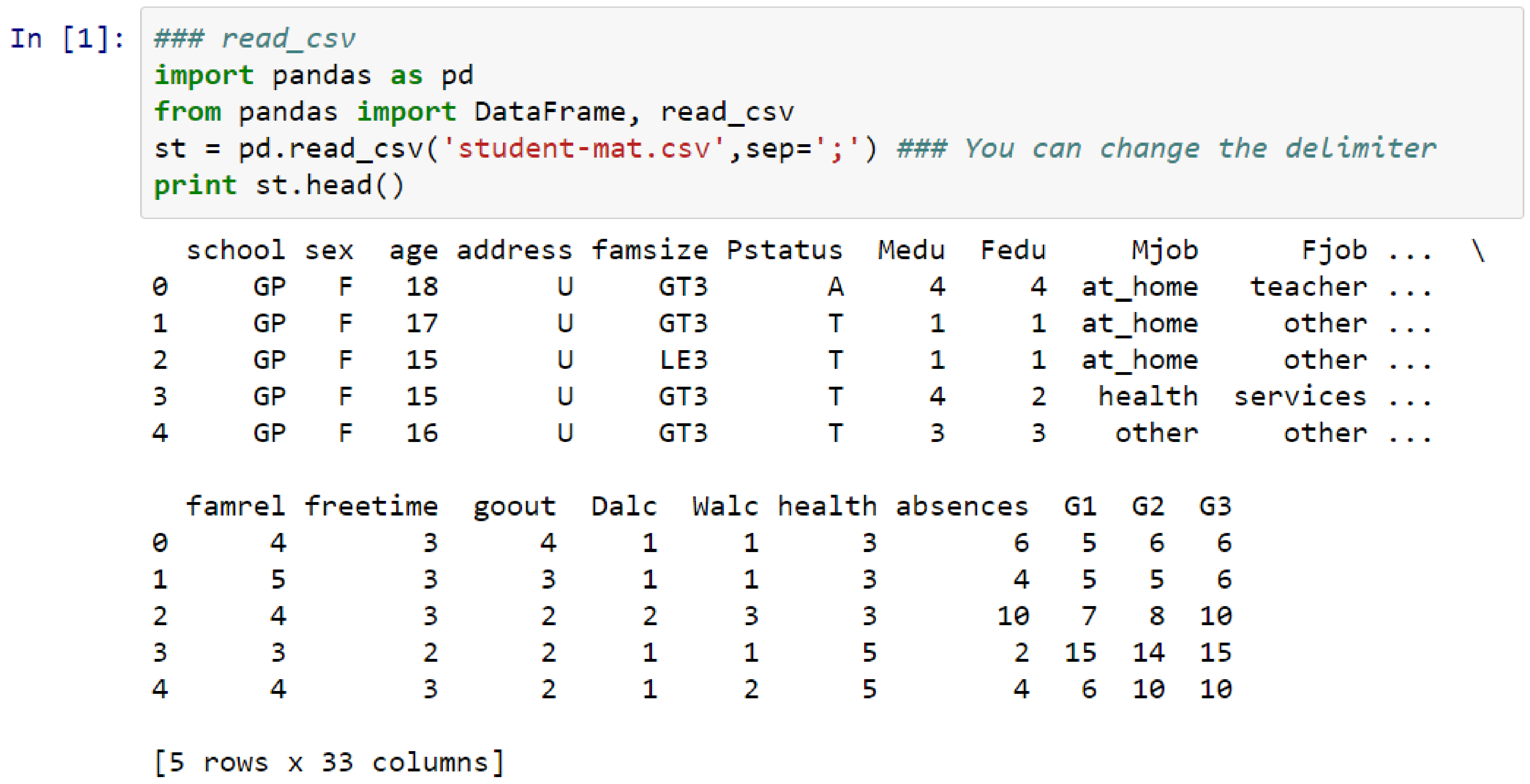
Data ingestion

- ✓ Sourcing data for processing

Accessing data (Python)

- In python pandas dataframes are commonly used for accessing and loading tabular form data.
- Like R, read_csv and read_table are usually used to load delimited data from a file, URL or file like object.
- Default delimiter of read_csv is comma while tab is known for read_table, but you can change them manually based on your file.
- Usually these function have features that are helpful for indexing, type inferences and data conversion, date time parsing, iterating of very large files and unclean data issues.
- Pandas data frame also supports Excel file and using ‘parse’ syntax, data can be read into data frame.

Accessing data (Python)

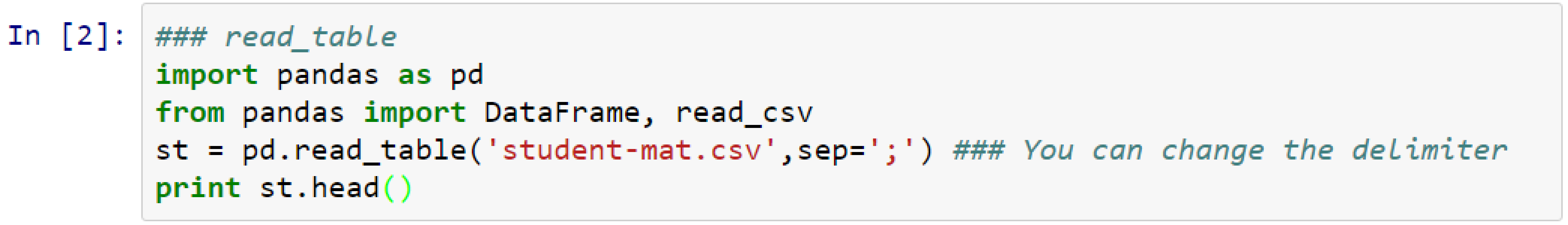


```
In [1]: ### read_csv
        import pandas as pd
        from pandas import DataFrame, read_csv
        st = pd.read_csv('student-mat.csv',sep=';') ### You can change the delimiter
        print st.head()
```

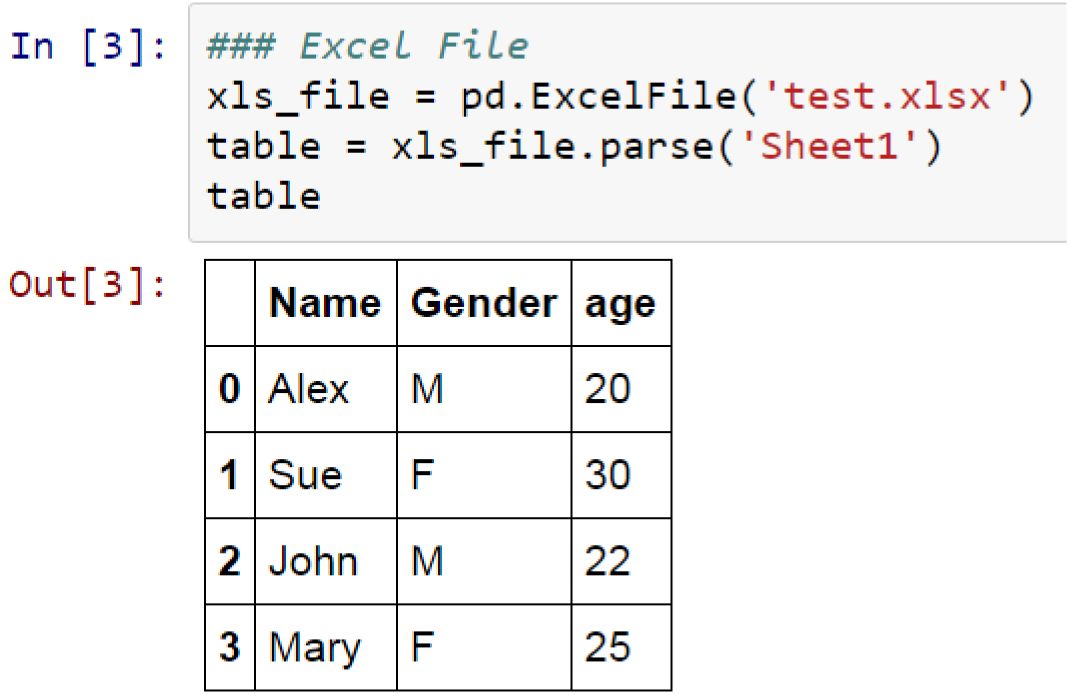
```
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob ...  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher ...
1     GP   F   17       U     GT3       T     1     1  at_home     other ...
2     GP   F   15       U     LE3       T     1     1  at_home     other ...
3     GP   F   15       U     GT3       T     4     2   health  services ...
4     GP   F   16       U     GT3       T     3     3    other     other ...

  famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0      4        3      4     1     1      3        6   5   6   6
1      5        3      3     1     1      3        4   5   5   6
2      4        3      2     2     3      3       10   7   8  10
3      3        2      2     1     1      5        2  15  14  15
4      4        3      2     1     2      5        4   6  10  10

[5 rows x 33 columns]
```

See Data_wrangling0.ipynb

Accessing data (Python)



```
In [2]: ### read_table
        import pandas as pd
        from pandas import DataFrame, read_csv
        st = pd.read_table('student-mat.csv',sep=';') ### You can change the delimiter
        print st.head()
```



```
In [3]: ### Excel File
        xls_file = pd.ExcelFile('test.xlsx')
        table = xls_file.parse('Sheet1')
        table
```

Out[3]:

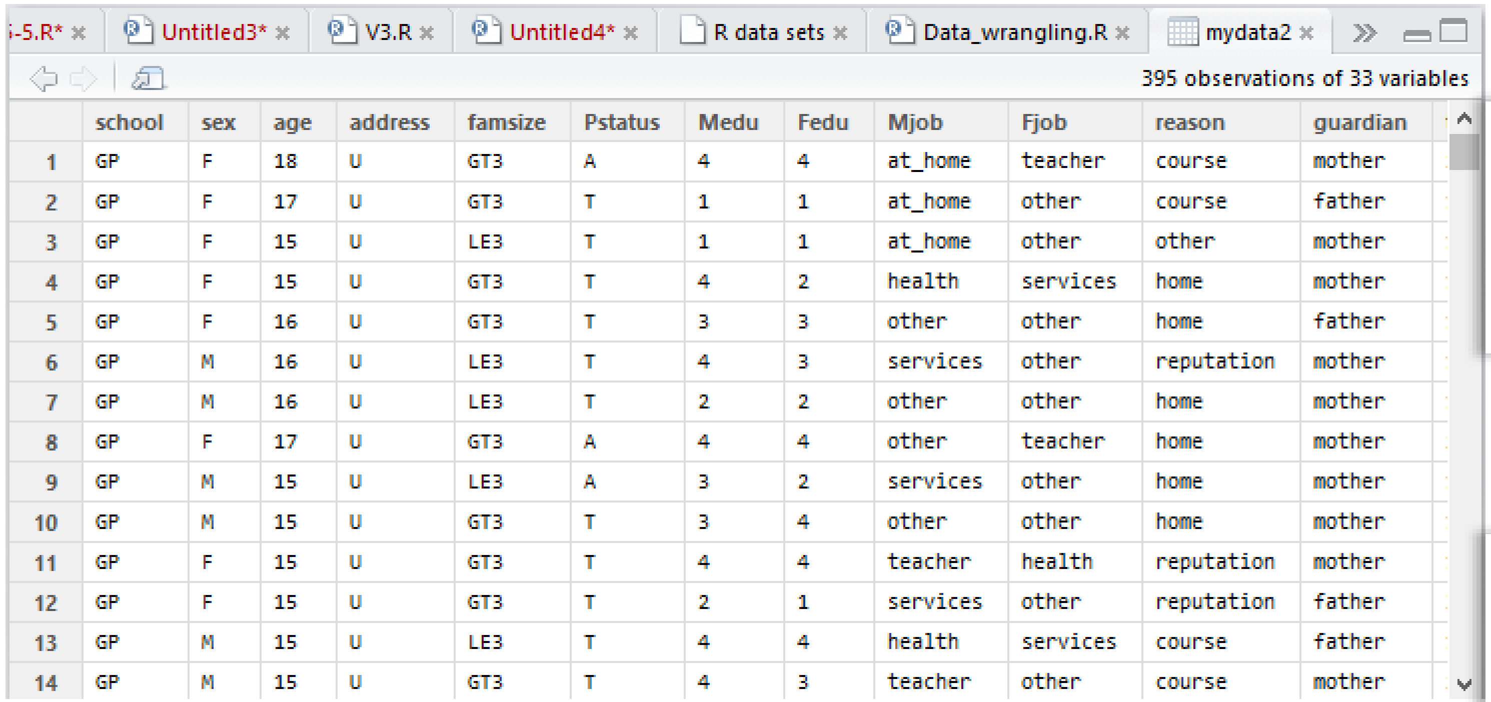
	Name	Gender	age
0	Alex	M	20
1	Sue	F	30
2	John	M	22
3	Mary	F	25

See Data_wrangling0.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Accessing data (R)

```r
### Existing local data
mydata1 <- read.csv("student-mat.csv",sep=";",header=TRUE)
head(mydata1)
mydata2 <- read.table("student-mat.csv",sep=";",header=TRUE)
head(mydata2)
```



395 observations of 33 variables

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	reason	guardian
1	GP	F	18	U	GT3	A	4	4	at_home	teacher	course	mother
2	GP	F	17	U	GT3	T	1	1	at_home	other	course	father
3	GP	F	15	U	LE3	T	1	1	at_home	other	other	mother
4	GP	F	15	U	GT3	T	4	2	health	services	home	mother
5	GP	F	16	U	GT3	T	3	3	other	other	home	father
6	GP	M	16	U	LE3	T	4	3	services	other	reputation	mother
7	GP	M	16	U	LE3	T	2	2	other	other	home	mother
8	GP	F	17	U	GT3	A	4	4	other	teacher	home	mother
9	GP	M	15	U	LE3	A	3	2	services	other	home	mother
10	GP	M	15	U	GT3	T	3	4	other	other	home	mother
11	GP	F	15	U	GT3	T	4	4	teacher	health	reputation	mother
12	GP	F	15	U	GT3	T	2	1	services	other	reputation	father
13	GP	M	15	U	LE3	T	4	4	health	services	course	father
14	GP	M	15	U	GT3	T	4	3	teacher	other	course	mother

In R, read.csv() and read.table() are usually used to access local data and from the web

In R, working with Excel package needs both Java and Perl packages.

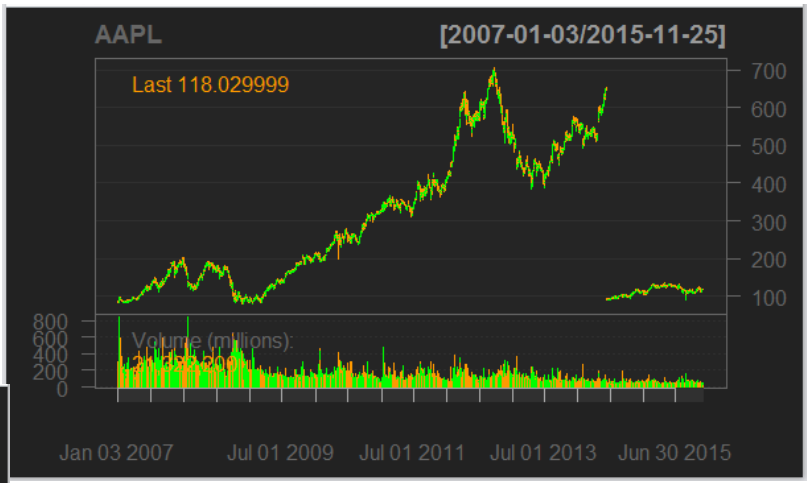
See [Data_wrangling0.R]

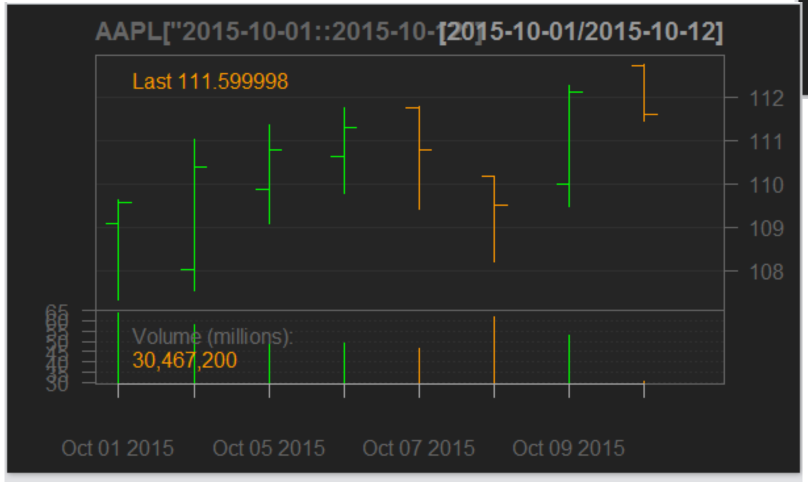
QuantUniversity, LLC
www.quantuniversity.com

Accessing data (R)

- For example, using ‘quantmod’ package in R, you can extract and analyze stock prices as well as some visualizations such as bar graphs .

```
### Help with some external data
install.packages("quantmod")
library(quantmod)
getSymbols("AAPL")
barChart(AAPL)
barChart(AAPL['2015-10-01::2015-10-12'])
```





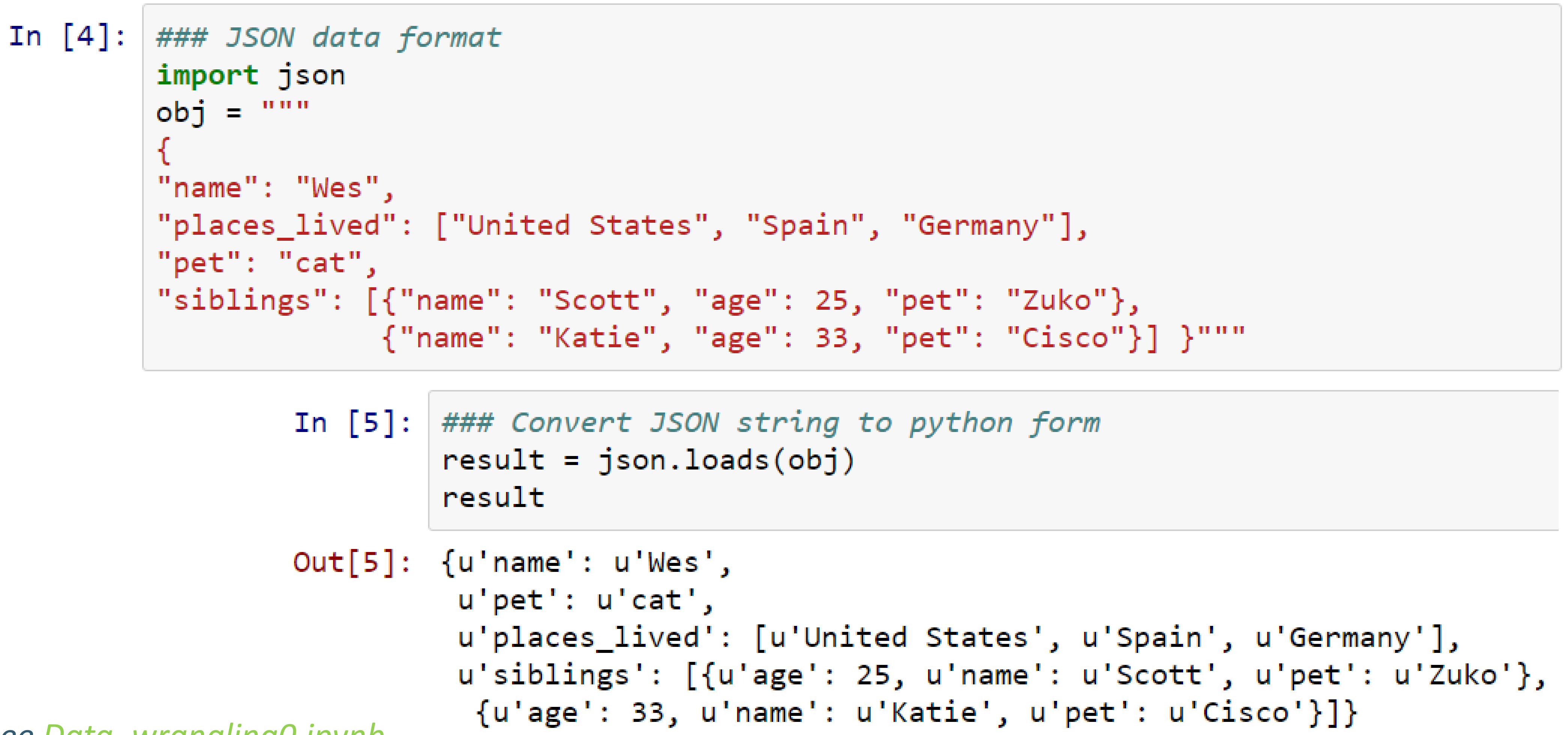
See Data_wrangling0.R

QuantUniversity, LLC
www.quantuniversity.com

Accessing data

- One of the largest directories is provided by the *Open Access Directory* which includes of scientific or research data in different areas such as energy, social sciences, computer sciences, etc.
- *CKAN* and *Quora* may help you out where you can find data on specific topic areas.
- Later on, in this presentation material we will be focused on loading and parsing of HTML structured data such as HTML tables as well as JSON data format.
- HTML tables contains small datasets published on websites.

Accessing data: (Working with JSON data format)



```
In [4]:  ### JSON data format
         import json
         obj = """
         {
         "name": "Wes",
         "places_lived": ["United States", "Spain", "Germany"],
         "pet": "cat",
         "siblings": [{"name": "Scott", "age": 25, "pet": "Zuko"},
                      {"name": "Katie", "age": 33, "pet": "Cisco"}] }"""
```

```
In [5]:  ### Convert JSON string to python form
         result = json.loads(obj)
         result

Out[5]:  {u'name': u'Wes',
          u'pet': u'cat',
          u'places_lived': [u'United States', u'Spain', u'Germany'],
          u'siblings': [{u'age': 25, u'name': u'Scott', u'pet': u'Zuko'},
           {u'age': 33, u'name': u'Katie', u'pet': u'Cisco'}]}
```

See [Data_wrangling0.ipynb]

Accessing data: (Working with JSON data format)

In [6]:
```
### Extracting some data frame from JSON data format
siblings = DataFrame(result['siblings'], columns=['name', 'age'])
siblings
```

Out[6]:

	name	age
0	Scott	25
1	Katie	33

In [7]:
```
### Back to JSON
asjson = json.dumps(result)
asjson
```

Out[7]:
```
'{"pet": "cat", "siblings": [{"pet": "Zuko", "age": 25, "name": "Scott"}, {"pe
t": "Cisco", "age": 33, "name": "Katie"}], "name": "Wes", "places_lived": ["Uni
ted States", "Spain", "Germany"]}'
```

See Data_wrangling0.ipynb

QuantUniversity, LLC
www.quantuniversity.com

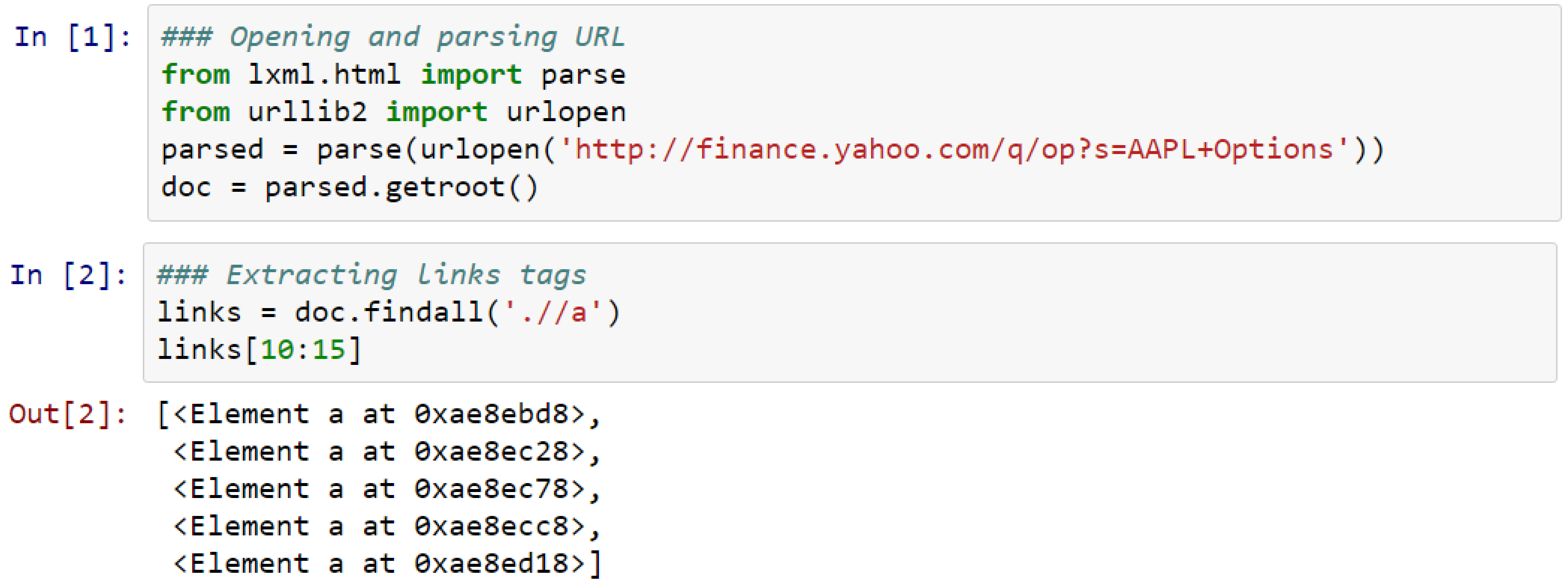
Accessing data: (Working with HTML data)

- HTML is a language for building the structure of webpage contents.
- Many websites use HTML tables to make data available. This way users can view the data by using different browsers.
- HTML elements are defined by their names as tags:
  - <html> : The whole document
  - <body> : The human-readable part of the web page
  - <table> : The frame of a table element
  - <tr> : A row in a table
  - <td> : A cell of content inside a row
  - <th> : A table header cell inside a row

Accessing data: (Working with HTML)

- Here we will show these tasks for the data in Yahoo Finance tables as an example in python step by step:
  - The first step is to open the URL and parsing the data.
  - By doing that we will be able to extract all specific tags such as table tags.
  - For instance we will show how to extract all links attached to the documents. (Links tags are “a” types in HTML document)
  - Then we should change the HTML elements to text.
  - As another example we can extract “example” table, it’s headers, rows and values inside each cell and finally convert all of these elements to usable format.

Working with HTML: Yahoo Finance links (Python)



```
In [1]: ### Opening and parsing URL
        from lxml.html import parse
        from urllib2 import urlopen
        parsed = parse(urlopen('http://finance.yahoo.com/q/op?s=AAPL+Options'))
        doc = parsed.getroot()
```

```
In [2]: ### Extracting links tags
        links = doc.findall('.//a')
        links[10:15]

Out[2]: [<Element a at 0xae8ebd8>,
         <Element a at 0xae8ec28>,
         <Element a at 0xae8ec78>,
         <Element a at 0xae8ecc8>,
         <Element a at 0xae8ed18>]
```

See Data_wrangling1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Working with HTML: Yahoo Finance links (Python)

```
In [3]: ### Changing HTML elements to text
        urls = [lnk.get('href') for lnk in doc.findall('.//a')]
        urls[10:15]

Out[3]: ['https://www.flickr.com/',
         'https://mobile.yahoo.com/',
         'http://everything.yahoo.com/',
         'https://www.yahoo.com/politics',
         'https://celebrity.yahoo.com/']
```

```
In [4]: ### Extracting table tags and assigning example to the first table
        tables = doc.findall('.//table')
        example=tables[2]
```

```
In [5]: ### Extracting all rows of example table
        rows = example.findall('.//tr')
        ### Extracting all elements of rows including headers row
        def _unpack(row, kind='td'):
            ### th kind refers to header row and td refers to other rows
            elts = row.findall('.//%s' % kind)
            return [val.text_content() for val in elts]
```

See Data_wrangling1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Working with HTML: Yahoo Finance links (Python)

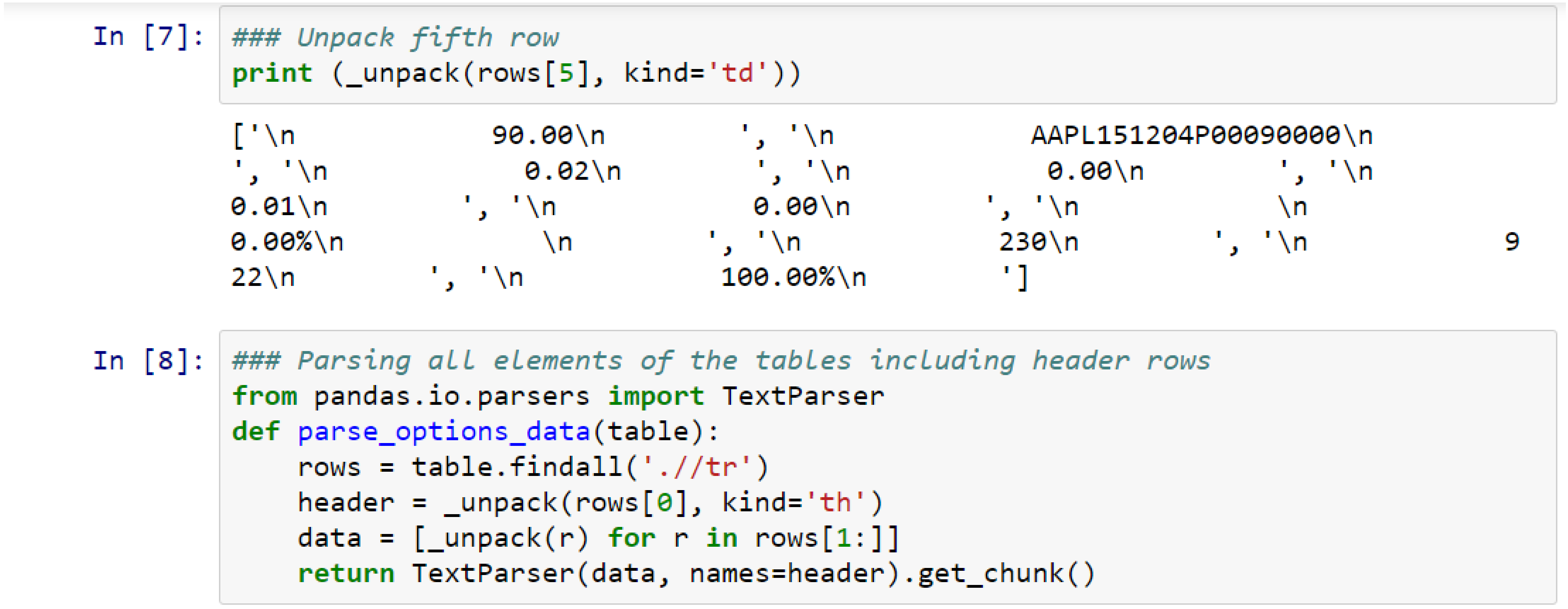
```
### Unpack header row
print (_unpack(rows[0], kind='th'))
```

```
[u'\n                \n                     Strike\n                    \n                        \ue0
04\n                        \ue002\n                    \n                \n                \u2235 Fil
ter\n            ', 'Contract Name', u'\n                \n                    Last\n
\n                        \ue004\n                        \ue002\n                    \n
\n            ', u'\n                \n                    Bid\n                    \n
\ue004\n                        \ue002\n                    \n                \n            ', u'\n
\n                    Ask\n                    \n                        \ue004\n
\ue002\n                    \n                \n            ', u'\n                \n
Change\n                    \n                        \ue004\n                        \ue002\n
\n                \n            ', u'\n                \n                    %Change\n
\n                        \ue004\n                        \ue002\n                    \n
\n            ', u'\n                \n                    Volume\n                    \n
\ue004\n                        \ue002\n                    \n                \n            ', u'\n
\n                    Open Interest\n                    \n                        \ue004\n
\ue002\n                    \n                \n            ', u'\n                \n
Implied Volatility\n                    \n                        \ue004\n                        \ue0
02\n                    \n                \n            ']
```

See Data_wrangling1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Working with HTML: Yahoo Finance links (Python)



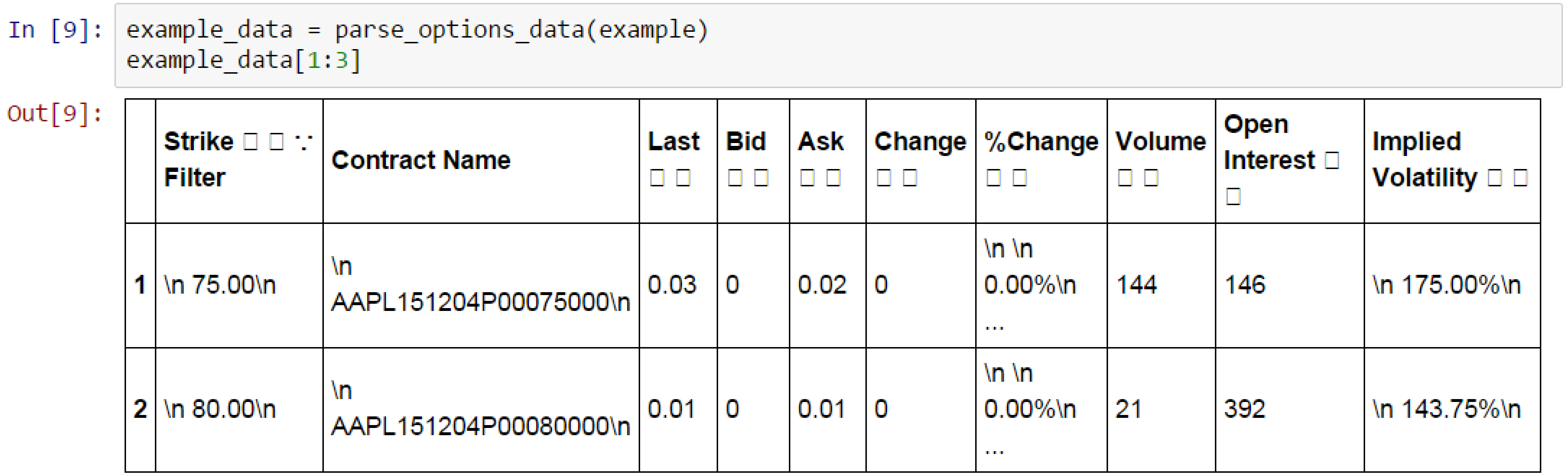
```
In [7]: ### Unpack fifth row
        print (_unpack(rows[5], kind='td'))

        ['\n            90.00\n          ', '\n            AAPL151204P00090000\n
        ', '\n            0.02\n          ', '\n            0.00\n          ', '\n
        0.01\n          ', '\n            0.00\n          ', '\n            \n
        0.00%\n            \n          ', '\n            230\n          ', '\n          9
        22\n          ', '\n            100.00%\n          ']
```

```
In [8]: ### Parsing all elements of the tables including header rows
        from pandas.io.parsers import TextParser
        def parse_options_data(table):
            rows = table.findall('.//tr')
            header = _unpack(rows[0], kind='th')
            data = [_unpack(r) for r in rows[1:]]
            return TextParser(data, names=header).get_chunk()
```

See Data_wrangling1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Working with HTML: Yahoo Finance links (Python)



```
In [9]: example_data = parse_options_data(example)
        example_data[1:3]
```

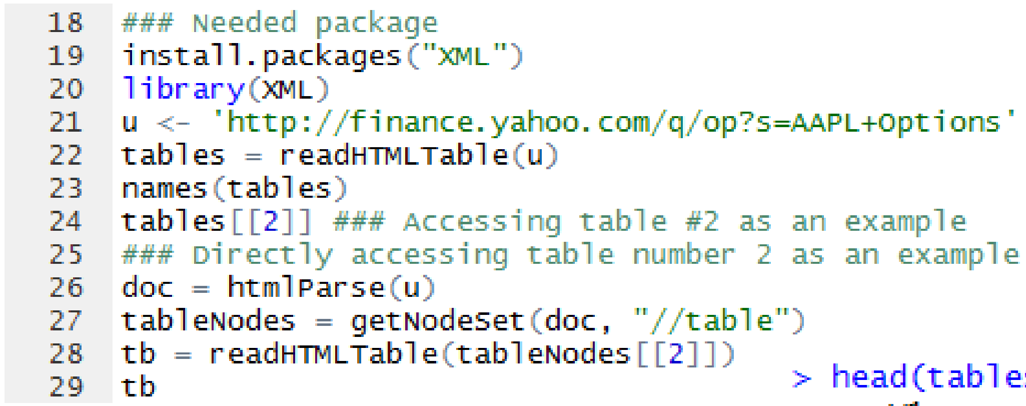
Out[9]:

	Strike □ □ ∵ Filter	Contract Name	Last □ □	Bid □ □	Ask □ □	Change □ □	%Change □ □	Volume □ □	Open Interest □ □	Implied Volatility □ □
1	\n 75.00\n	\n AAPL151204P00075000\n	0.03	0	0.02	0	\n \n 0.00%\n ...	144	146	\n 175.00%\n
2	\n 80.00\n	\n AAPL151204P00080000\n	0.01	0	0.01	0	\n \n 0.00%\n ...	21	392	\n 143.75%\n

See Data_wrangling1.ipynb

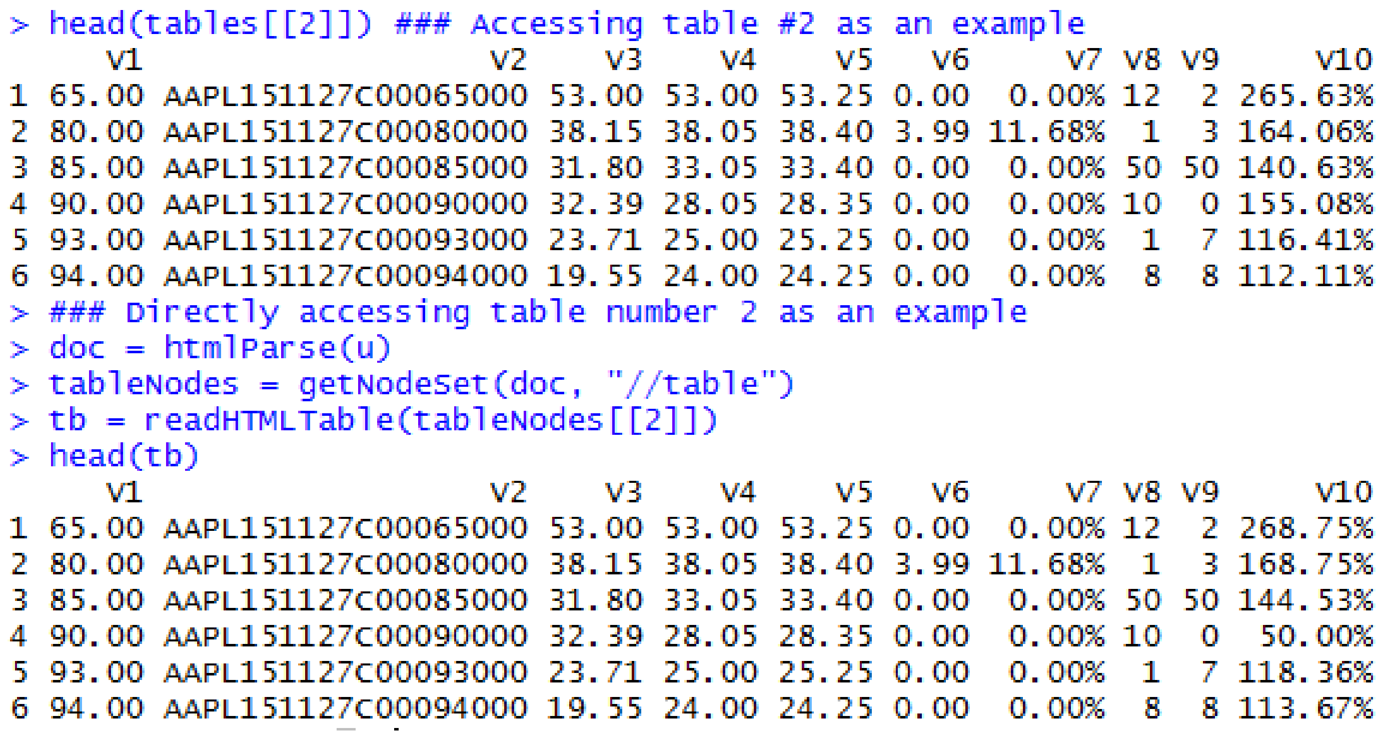
QuantUniversity, LLC
www.quantuniversity.com

Accessing data: Working with HTML (R)



```r
### Needed package
install.packages("XML")
library(XML)
u <- 'http://finance.yahoo.com/q/op?s=AAPL+Options'
tables = readHTMLTable(u)
names(tables)
tables[[2]] ### Accessing table #2 as an example
### Directly accessing table number 2 as an example
doc = htmlParse(u)
tableNodes = getNodeSet(doc, "//table")
tb = readHTMLTable(tableNodes[[2]])
tb
```

In R, 'XML' package deals with extracting and parsing HTML data



```
> head(tables[[2]]) ### Accessing table #2 as an example
     V1                 V2    V3    V4    V5   V6     V7 V8 V9     V10
1 65.00 AAPL151127C00065000 53.00 53.00 53.25 0.00  0.00% 12  2 265.63%
2 80.00 AAPL151127C00080000 38.15 38.05 38.40 3.99 11.68%  1  3 164.06%
3 85.00 AAPL151127C00085000 31.80 33.05 33.40 0.00  0.00% 50 50 140.63%
4 90.00 AAPL151127C00090000 32.39 28.05 28.35 0.00  0.00% 10  0 155.08%
5 93.00 AAPL151127C00093000 23.71 25.00 25.25 0.00  0.00%  1  7 116.41%
6 94.00 AAPL151127C00094000 19.55 24.00 24.25 0.00  0.00%  8  8 112.11%
> ### Directly accessing table number 2 as an example
> doc = htmlParse(u)
> tableNodes = getNodeSet(doc, "//table")
> tb = readHTMLTable(tableNodes[[2]])
> head(tb)
     V1                 V2    V3    V4    V5   V6     V7 V8 V9     V10
1 65.00 AAPL151127C00065000 53.00 53.00 53.25 0.00  0.00% 12  2 268.75%
2 80.00 AAPL151127C00080000 38.15 38.05 38.40 3.99 11.68%  1  3 168.75%
3 85.00 AAPL151127C00085000 31.80 33.05 33.40 0.00  0.00% 50 50 144.53%
4 90.00 AAPL151127C00090000 32.39 28.05 28.35 0.00  0.00% 10  0  50.00%
5 93.00 AAPL151127C00093000 23.71 25.00 25.25 0.00  0.00%  1  7 118.36%
6 94.00 AAPL151127C00094000 19.55 24.00 24.25 0.00  0.00%  8  8 113.67%
```

See Data_wrangling0.R

Merging data sources

- ✓ Combining and merging datasets (database style merge)
- ✓ Combining and merging datasets (Merging on index)
- ✓ Concatenating along axis
- ✓ Reshaping and pivoting
- ✓ Filtering
- ✓ Sorting

Combining and merging datasets (Database style)

- Data can be combined in different ways by using python:
  - pandas.merge: Connects rows based on one or more keys like SQL join. Merge function is pretty similar to join in SQL. So if you don't specify the key column, it will automatically consider the mutual column as key.
  - If the column names are different, you can specify them separately.
  - By doing that, keys in the result are keys in common.
  - By default merge does like an inner join. Other options could be left, right and outer join.
  - Also you may join based on multiple keys.

Database style merge (Python)

```
In [1]: import pandas as pd
        from pandas import *
        df1 = DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],'data1': range(6)})
        df2 = DataFrame({'key': ['a', 'b', 'd'],'data2': range(3)})
        print pd.merge(df1, df2)
```

```
   data1 key  data2
0      0   b      1
1      1   b      1
2      5   b      1
3      2   a      0
4      4   a      0
```

Joining on one key

```
In [2]: df3 = DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],'data1': range(7)})
        df4 = DataFrame({'rkey': ['a', 'b', 'd'],'data2': range(3)})
        pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[2]:

	data1	lkey	data2	rkey
0	0	b	1	b
1	1	b	1	b
2	6	b	1	b
3	2	a	0	a
4	4	a	0	a
5	5	a	0	a

Joining on two keys

See Data_wrangling2.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Database style merge (Python)

```
df1 = DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'], 'data1': range(6)})
df2 = DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],'data2': range(5)})
pd.merge(df1, df2, on='key', how='left')
```

	data1	key	data2
0	0	b	1
1	0	b	3
2	1	b	1
3	1	b	3
4	2	a	0
5	2	a	2
6	3	c	NaN
7	4	a	0
8	4	a	2
9	5	b	1
10	5	b	3

Left join on one key

```
left = DataFrame({'Gender': ['F', 'F', 'M'],
                  'Course': ['IE', 'IS', 'IE'],
                  'Gre': [1100, 1150, 1170]})
right = DataFrame({'Gender': ['F', 'F', 'M', 'M'],
                   'Course': ['IE', 'IE', 'IE', 'IS'],
                   'IELTS': [7.5, 7, 6.5, 8]})
pd.merge(left, right, on=['Gender', 'Course'], how='outer')
```

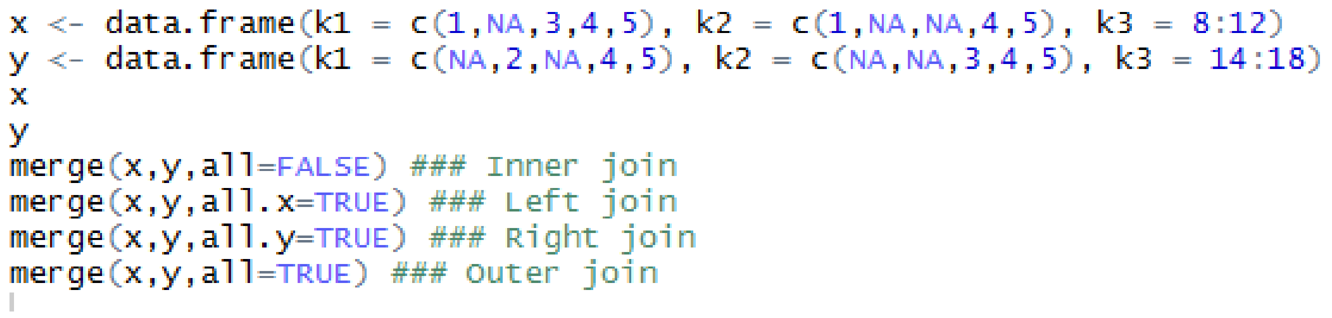
	Course	Gender	Gre	IELTS
0	IE	F	1100	7.5
1	IE	F	1100	7.0
2	IS	F	1150	NaN
3	IE	M	1170	6.5
4	IS	M	NaN	8.0

Outer join on two keys

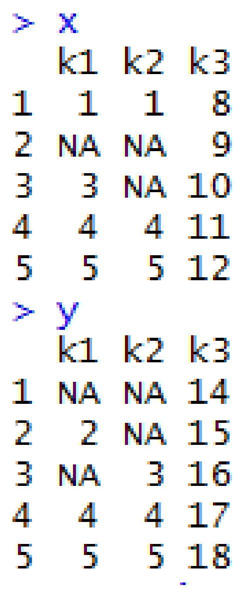
See Data_wrangling2.ipynb

QuantUniversity, LLC
www.quantuniversity.com

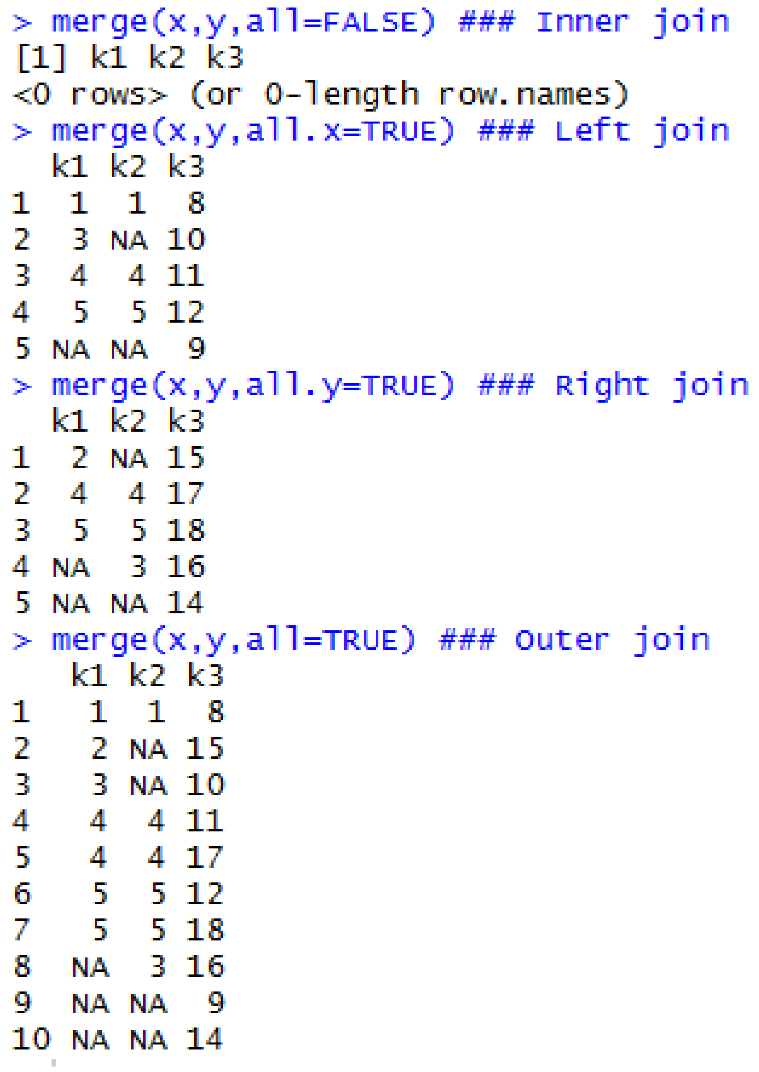
Database style merge (R)



```
x <- data.frame(k1 = c(1,NA,3,4,5), k2 = c(1,NA,NA,4,5), k3 = 8:12)
y <- data.frame(k1 = c(NA,2,NA,4,5), k2 = c(NA,NA,3,4,5), k3 = 14:18)
x
y
merge(x,y,all=FALSE) ### Inner join
merge(x,y,all.x=TRUE) ### Left join
merge(x,y,all.y=TRUE) ### Right join
merge(x,y,all=TRUE) ### Outer join
```



```
> x
  k1 k2 k3
1  1  1  8
2 NA NA  9
3  3 NA 10
4  4  4 11
5  5  5 12
> y
  k1 k2 k3
1 NA NA 14
2  2 NA 15
3 NA  3 16
4  4  4 17
5  5  5 18
```



```
> merge(x,y,all=FALSE) ### Inner join
[1] k1 k2 k3
<0 rows> (or 0-length row.names)
> merge(x,y,all.x=TRUE) ### Left join
  k1 k2 k3
1  1  1  8
2  3 NA 10
3  4  4 11
4  5  5 12
5 NA NA  9
> merge(x,y,all.y=TRUE) ### Right join
  k1 k2 k3
1  2 NA 15
2  4  4 17
3  5  5 18
4 NA  3 16
5 NA NA 14
> merge(x,y,all=TRUE) ### Outer join
   k1 k2 k3
1   1  1  8
2   2 NA 15
3   3 NA 10
4   4  4 11
5   4  4 17
6   5  5 12
7   5  5 18
8  NA  3 16
9  NA NA  9
10 NA NA 14
```

See Data_wrangling1.R

QuantUniversity, LLC
www.quantuniversity.com

Merging on index (Python)

- Some times the merge key(keys) is found in its index.

```
import pandas as pd
from pandas import *
left1 = DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],'value': range(6)})
right1 = DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
pd.merge(left1, right1, left_on='key', right_index=True)
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

Passing left/ right index to true, allows you the index from left/right data frame as key join(s)

See Data_wrangling3.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Merging on index (Python)

```python
lefth = DataFrame({'key1': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
                   'key2': [2000, 2001, 2002, 2001, 2002],
                   'data': np.arange(5.)})
righth = DataFrame(np.arange(12).reshape((6, 2)),
                   index=[['Nevada', 'Nevada', 'Ohio', 'Ohio', 'Ohio', 'Ohio'],
                          [2001, 2000, 2000, 2000, 2001, 2002]],
                   columns=['event1', 'event2'])
print lefth
print righth
```

```
   data    key1  key2
0     0    Ohio  2000
1     1    Ohio  2001
2     2    Ohio  2002
3     3  Nevada  2001
4     4  Nevada  2002
             event1  event2
Nevada 2001       0       1
       2000       2       3
Ohio   2000       4       5
       2000       6       7
       2001       8       9
       2002      10      11
```

```python
pd.merge(lefth, righth, left_on=['key1', 'key2'], right_index=True)
```

	data	key1	key2	event1	event2
0	0	Ohio	2000	4	5
0	0	Ohio	2000	6	7
1	1	Ohio	2001	8	9
2	2	Ohio	2002	10	11
3	3	Nevada	2001	0	1

See Data_wrangling3.ipynb

Merging on index (Python)

- You may also use the index for both sides:

```
left2 = DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                  index=['a', 'c', 'e'],
                  columns=['Ohio', 'Nevada'])
right2 = DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                   index=['b', 'c', 'd', 'e'],
                   columns=['Missouri', 'Alabama'])
print left2
print right2
```

```
   Ohio  Nevada
a     1       2
c     3       4
e     5       6
   Missouri  Alabama
b         7        8
c         9       10
d        11       12
e        13       14
```

```
pd.merge(left2, right2, how='outer', left_index=True, right_index=True)
```

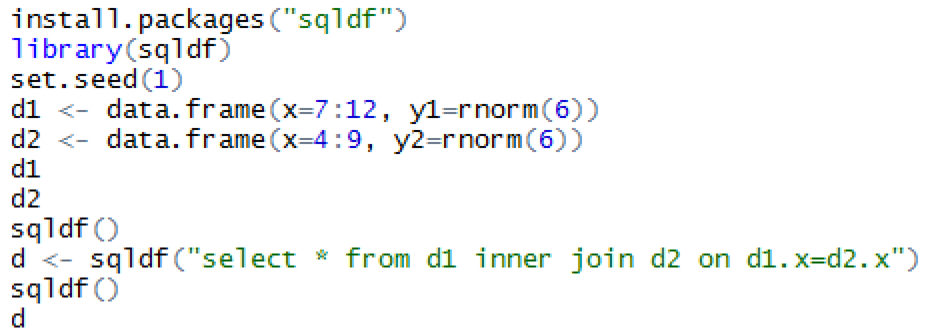
	Ohio	Nevada	Missouri	Alabama
a	1	2	NaN	NaN
b	NaN	NaN	7	8
c	3	4	9	10
d	NaN	NaN	11	12
e	5	6	13	14

See Data_wrangling3.ipynb

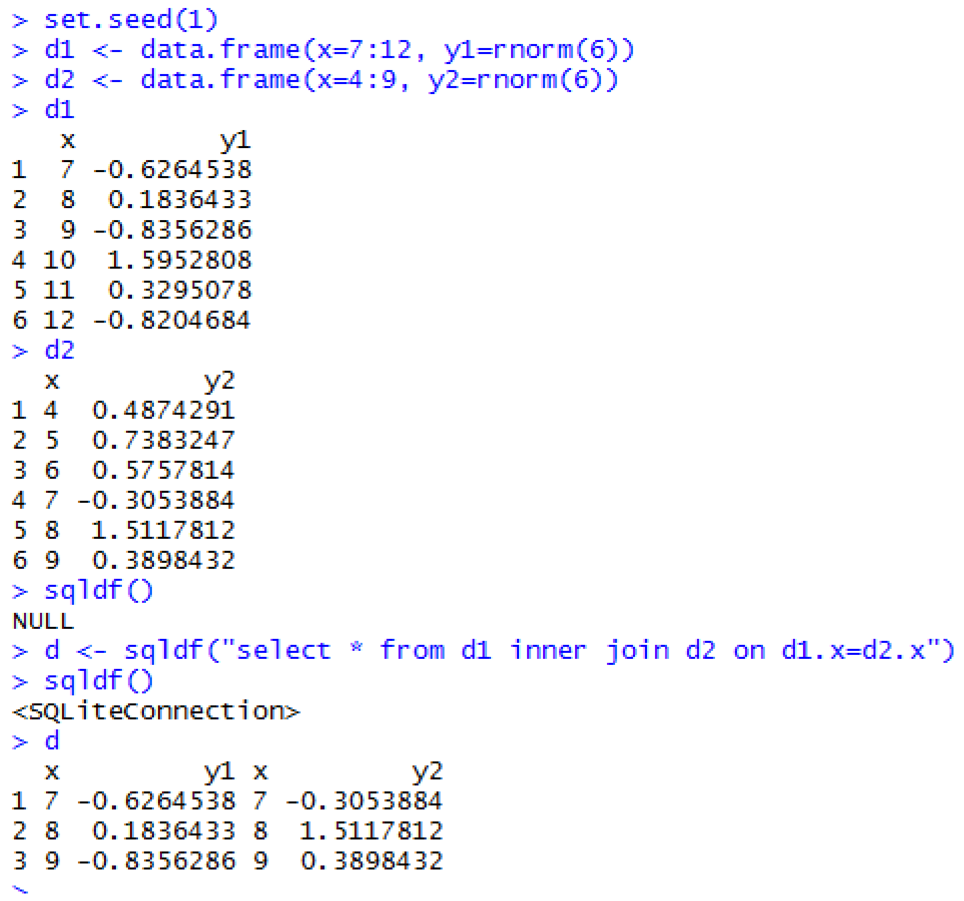
QuantUniversity, LLC
www.quantuniversity.com

Merging on index (R)

- Using “sqldf” library in R, you can perform several functions such as merging on index like SQL syntax.



```
install.packages("sqldf")
library(sqldf)
set.seed(1)
d1 <- data.frame(x=7:12, y1=rnorm(6))
d2 <- data.frame(x=4:9, y2=rnorm(6))
d1
d2
sqldf()
d <- sqldf("select * from d1 inner join d2 on d1.x=d2.x")
sqldf()
d
```



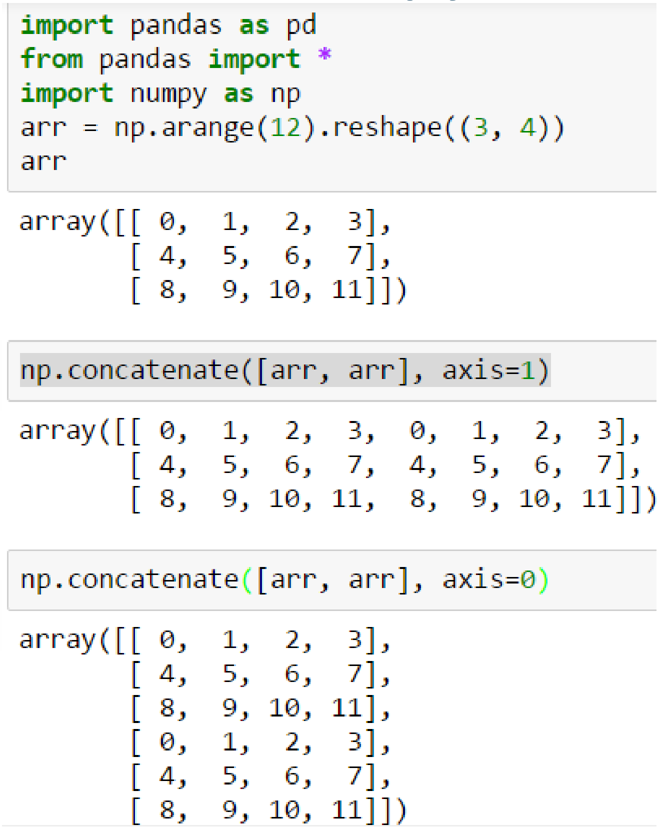
```
> set.seed(1)
> d1 <- data.frame(x=7:12, y1=rnorm(6))
> d2 <- data.frame(x=4:9, y2=rnorm(6))
> d1
   x         y1
1  7 -0.6264538
2  8  0.1836433
3  9 -0.8356286
4 10  1.5952808
5 11  0.3295078
6 12 -0.8204684
> d2
  x         y2
1 4  0.4874291
2 5  0.7383247
3 6  0.5757814
4 7 -0.3053884
5 8  1.5117812
6 9  0.3898432
> sqldf()
NULL
> d <- sqldf("select * from d1 inner join d2 on d1.x=d2.x")
> sqldf()
<SQLiteConnection>
> d
  x         y1 x         y2
1 7 -0.6264538 7 -0.3053884
2 8  0.1836433 8  1.5117812
3 9 -0.8356286 9  0.3898432
>
```

See Data_wrangling2.R

QuantUniversity, LLC
www.quantuniversity.com

Concatenating along axis (Python)

- pandas.concat: Concatenating or binding stacking could be other forms of combinations. Also numpy has this functionality.



```
import pandas as pd
from pandas import *
import numpy as np
arr = np.arange(12).reshape((3, 4))
arr
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Column wise

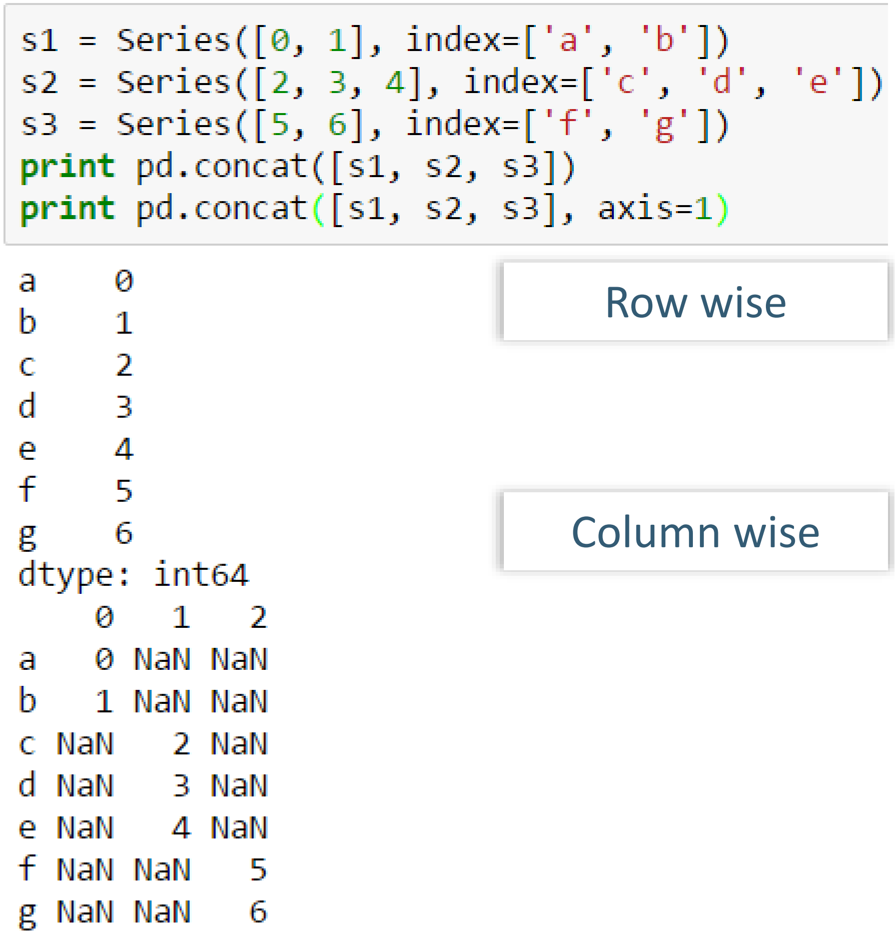
```
np.concatenate([arr, arr], axis=1)
```

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

Row wise

```
np.concatenate([arr, arr], axis=0)
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```



```
s1 = Series([0, 1], index=['a', 'b'])
s2 = Series([2, 3, 4], index=['c', 'd', 'e'])
s3 = Series([5, 6], index=['f', 'g'])
print pd.concat([s1, s2, s3])
print pd.concat([s1, s2, s3], axis=1)
```

Row wise

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

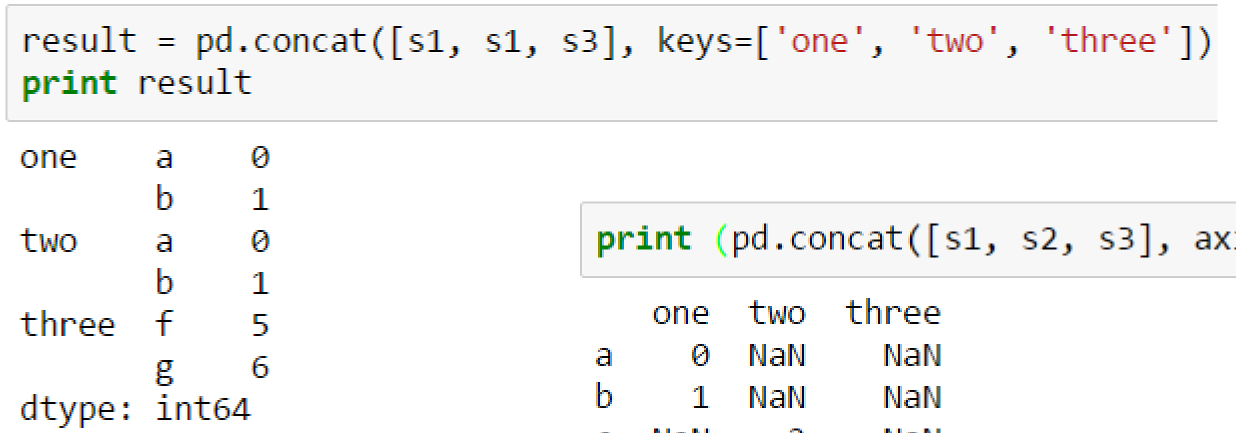
Column wise

```
     0    1    2
a    0  NaN  NaN
b    1  NaN  NaN
c  NaN    2  NaN
d  NaN    3  NaN
e  NaN    4  NaN
f  NaN  NaN    5
g  NaN  NaN    6
```

See Data_wrangling4.ipynb

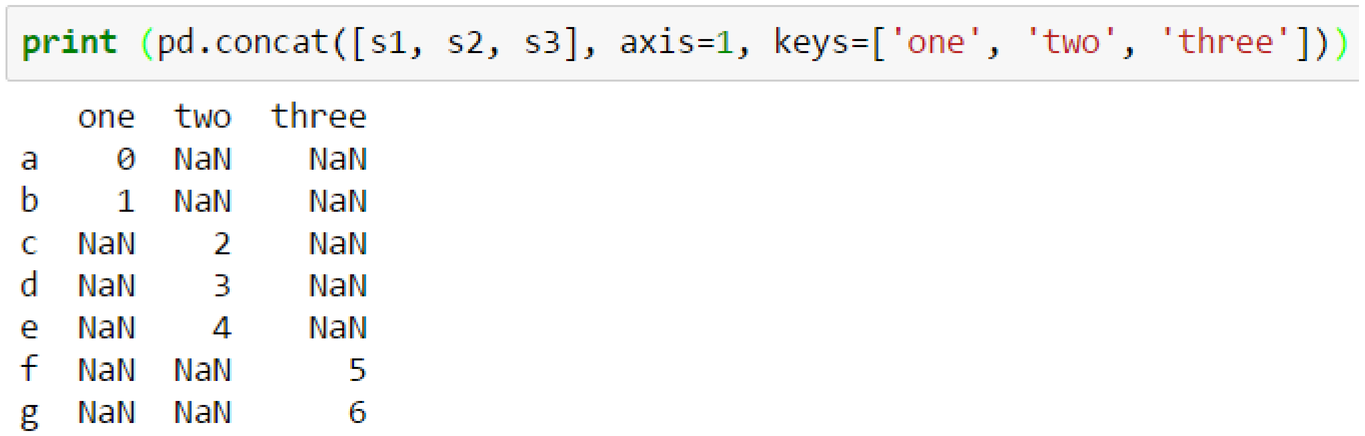
Concatenating along axis (Python)

- pandas.concat: you may also create hierarchical index by doing concatenation.



```
result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
print result
```

```
one    a    0
       b    1
two    a    0
       b    1
three  f    5
       g    6
dtype: int64
```



```
print (pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three']))
```

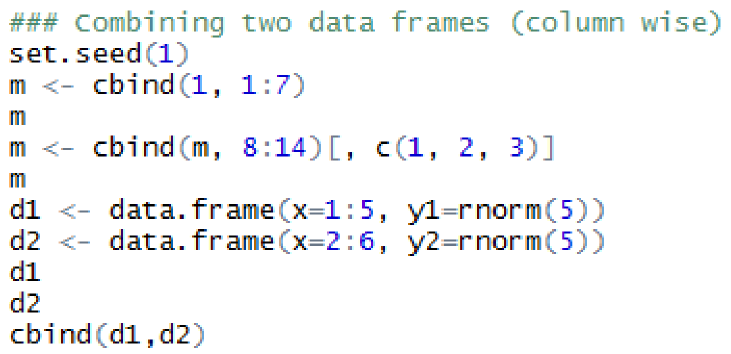
```
   one  two  three
a    0  NaN    NaN
b    1  NaN    NaN
c  NaN    2    NaN
d  NaN    3    NaN
e  NaN    4    NaN
f  NaN  NaN      5
g  NaN  NaN      6
```

See Data_wrangling4.ipynb

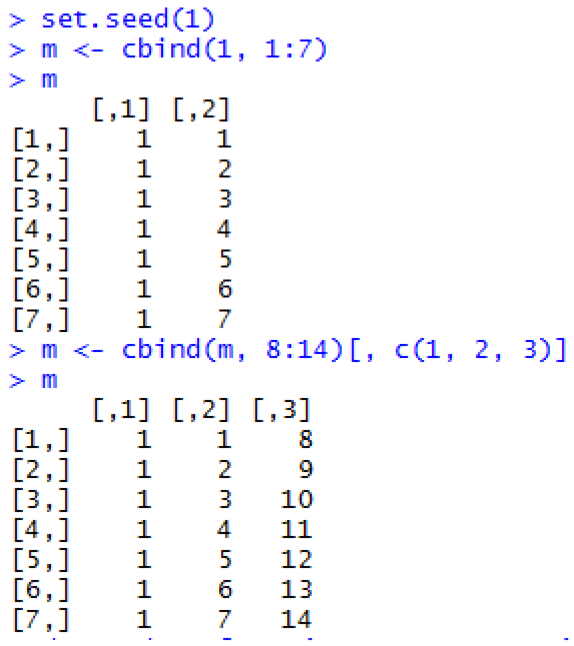
QuantUniversity, LLC
www.quantuniversity.com

Concatenating along axis (R)

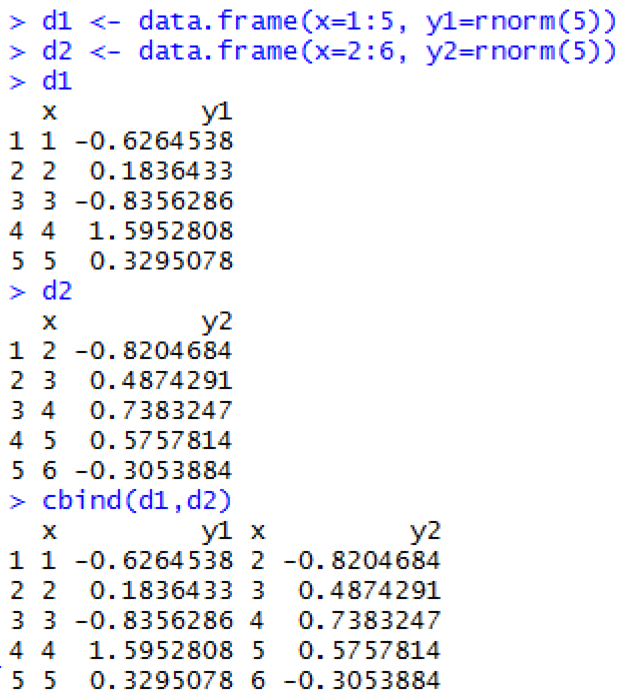
- **cbind()** and **rbind()** are two functions in R which allow you to combine data frames in R, column/row wise.



```
### Combining two data frames (column wise)
set.seed(1)
m <- cbind(1, 1:7)
m
m <- cbind(m, 8:14)[, c(1, 2, 3)]
m
d1 <- data.frame(x=1:5, y1=rnorm(5))
d2 <- data.frame(x=2:6, y2=rnorm(5))
d1
d2
cbind(d1,d2)
```



```
> set.seed(1)
> m <- cbind(1, 1:7)
> m
     [,1] [,2]
[1,]    1    1
[2,]    1    2
[3,]    1    3
[4,]    1    4
[5,]    1    5
[6,]    1    6
[7,]    1    7
> m <- cbind(m, 8:14)[, c(1, 2, 3)]
> m
     [,1] [,2] [,3]
[1,]    1    1    8
[2,]    1    2    9
[3,]    1    3   10
[4,]    1    4   11
[5,]    1    5   12
[6,]    1    6   13
[7,]    1    7   14
```

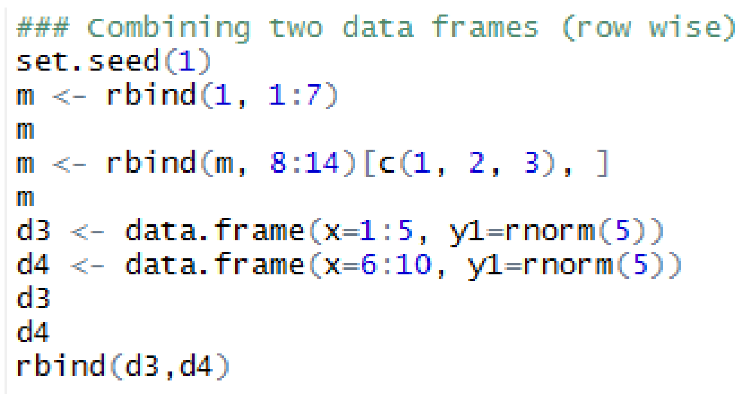


```
> d1 <- data.frame(x=1:5, y1=rnorm(5))
> d2 <- data.frame(x=2:6, y2=rnorm(5))
> d1
  x         y1
1 1 -0.6264538
2 2  0.1836433
3 3 -0.8356286
4 4  1.5952808
5 5  0.3295078
> d2
  x         y2
1 2 -0.8204684
2 3  0.4874291
3 4  0.7383247
4 5  0.5757814
5 6 -0.3053884
> cbind(d1,d2)
  x         y1 x         y2
1 1 -0.6264538 2 -0.8204684
2 2  0.1836433 3  0.4874291
3 3 -0.8356286 4  0.7383247
4 4  1.5952808 5  0.5757814
5 5  0.3295078 6 -0.3053884
```

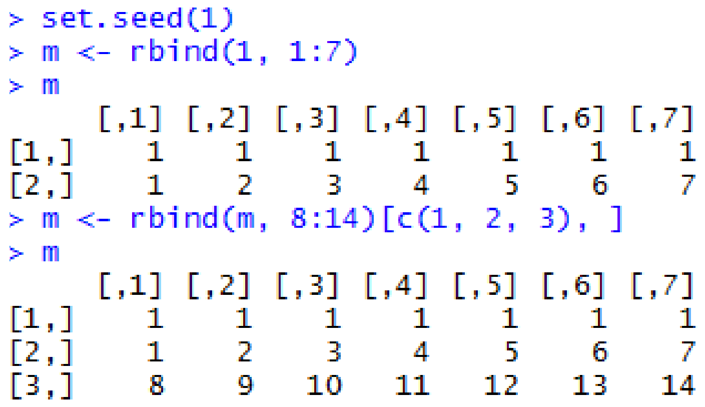
See Data_wrangling3.R

QuantUniversity, LLC
www.quantuniversity.com

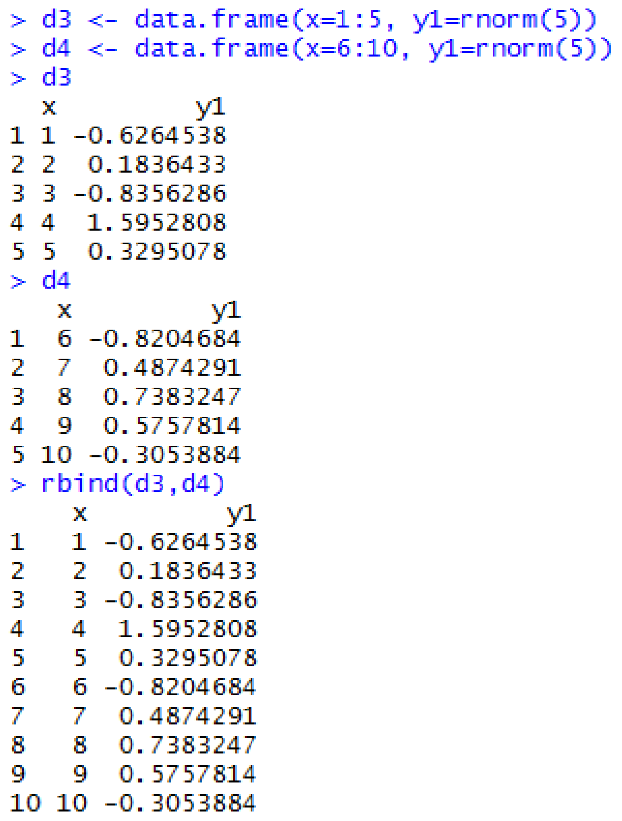
Concatenating along axis (R)



```
### Combining two data frames (row wise)
set.seed(1)
m <- rbind(1, 1:7)
m
m <- rbind(m, 8:14)[c(1, 2, 3), ]
m
d3 <- data.frame(x=1:5, y1=rnorm(5))
d4 <- data.frame(x=6:10, y1=rnorm(5))
d3
d4
rbind(d3,d4)
```



```
> set.seed(1)
> m <- rbind(1, 1:7)
> m
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    1    2    3    4    5    6    7
> m <- rbind(m, 8:14)[c(1, 2, 3), ]
> m
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]    1    1    1    1    1    1    1
[2,]    1    2    3    4    5    6    7
[3,]    8    9   10   11   12   13   14
```



```
> d3 <- data.frame(x=1:5, y1=rnorm(5))
> d4 <- data.frame(x=6:10, y1=rnorm(5))
> d3
  x         y1
1 1 -0.6264538
2 2  0.1836433
3 3 -0.8356286
4 4  1.5952808
5 5  0.3295078
> d4
   x         y1
1  6 -0.8204684
2  7  0.4874291
3  8  0.7383247
4  9  0.5757814
5 10 -0.3053884
> rbind(d3,d4)
    x         y1
1   1 -0.6264538
2   2  0.1836433
3   3 -0.8356286
4   4  1.5952808
5   5  0.3295078
6   6 -0.8204684
7   7  0.4874291
8   8  0.7383247
9   9  0.5757814
10 10 -0.3053884
```

See Data_wrangling3.R

QuantUniversity, LLC
www.quantuniversity.com

Combining data with overlaps (R)

- We may replace missing values of a data frame with values of another data frame if they have the same order or indices.

```r
### Mapping missing values of data frame with values of another data frame
### having same index

x <- data.frame(x1=c(NaN,2.5,NaN,3.5,4.5,NaN))
y <- data.frame(y1=c(1,2,3,4,5,6))
x
y
### Replacing missing values of x1 column with y1 column

for (i in 1:nrow(x)) {
  if (is.na(x[i,1])){
    x[i,1] <- y[i,1]
}}
x
```

```
   x1
1 NaN
2 2.5
3 NaN
4 3.5
5 4.5
6 NaN
> y
  y1
1  1
2  2
3  3
4  4
5  5
6  6
> for (i in 1:nrow(x)) {
+ if (is.na(x[i,1])){
+ x[i,1] <- y[i,1]
+ }}
> x
   x1
1 1.0
2 2.5
3 3.0
4 3.5
5 4.5
6 6.0
```

See Data_wrangling4.R

QuantUniversity, LLC
www.quantuniversity.com

Combining data with overlaps (Python)

- Combine_first: Can be applied when we have datasets with same indices.

```
import pandas as pd
from pandas import *
import numpy as np
from numpy import *
a = Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
            index=['f', 'e', 'd', 'c', 'b', 'a'])
print a
b = Series(np.arange(len(a), dtype=np.float64),
            index=['f', 'e', 'd', 'c', 'b', 'a'])
print b
```

```
f     NaN
e     2.5
d     NaN
c     3.5
b     4.5
a     NaN
dtype: float64
f     0
e     1
d     2
c     3
b     4
a     5
dtype: float64
```

```
print (b[:-1].combine_first(a[1:]))
```

```
a    NaN
b      4
c      3
d      2
e      1
f      0
dtype: float64
```

```
np.where(pd.isnull(a), b, a)
```

```
array([ 0. ,  2.5,  2. ,  3.5,  4.5,  5. ])
```

See Data_wrangling5.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (Python)

- Stack and unstack rotates data frame from column to row and vise versa.

```
import pandas as pd
from pandas import *
s1 = Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
s2 = Series([4, 5, 6], index=['c', 'd', 'e'])
data2 = pd.concat([s1, s2], keys=['one', 'two'])
print data2
```

```
one  a    0
     b    1
     c    2
     d    3
two  c    4
     d    5
     e    6
dtype: int64
```

```
data2.unstack()
```

	a	b	c	d	e
one	0	1	2	3	NaN
two	NaN	NaN	4	5	6

```
data2.unstack().stack()
```

```
one  a    0
     b    1
     c    2
     d    3
two  c    4
     d    5
     e    6
dtype: float64
```

See Data_wrangling6.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (Python)

- Stack and unstack rotates data frame from column to row and vise versa.

```
data = DataFrame(np.arange(6).reshape((2, 3)),
                 index=pd.Index(['Ohio', 'Colorado'], name='state'),
                 columns=pd.Index(['one', 'two', 'three'], name='number'))
data
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

```
result = data.stack()
result
```

```
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int32
```

```
result.unstack()
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

See Data_wrangling6.ipynb

QuantUniversity, LLC

www.quantuniversity.com

Reshaping and pivoting (Python)

- In python, **pivot()** syntax is used to change to long format to wide format.

```
### Changing long format to wide format
### Create a long format data set
import pandas as pd
from pandas import *
data={'type':['P1','P1','P2','P2','P3','P3'],
      'color':['Red','Blue','Red','Blue','Red','Blue'],
      'price':[100,120,140,90,110,105]}
data=DataFrame(data,columns=['type','color','price'])
data
```

	type	color	price
0	P1	Red	100
1	P1	Blue	120
2	P2	Red	140
3	P2	Blue	90
4	P3	Red	110
5	P3	Blue	105

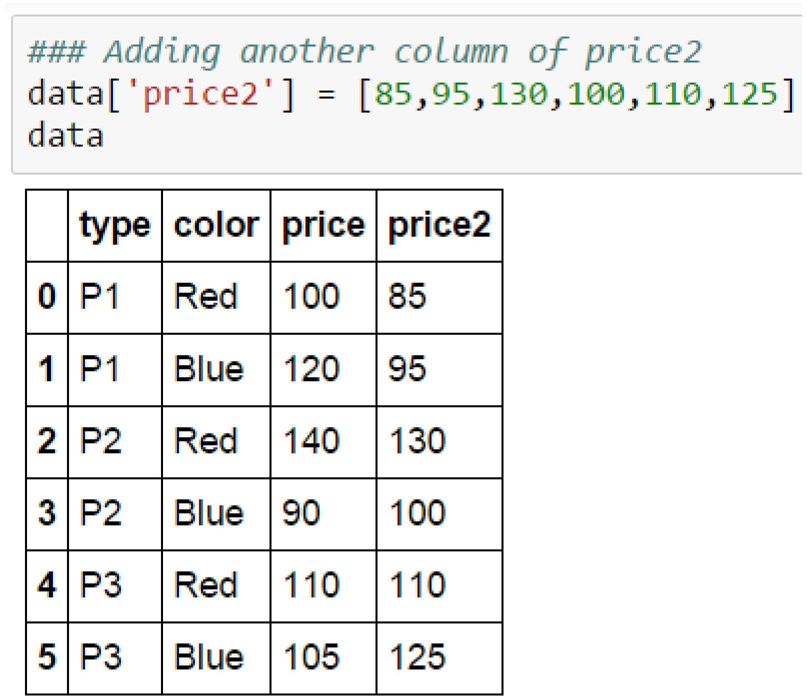
```
### Type and Price are used as row and column index
### Price is used to fill the table
pivoted = data.pivot('type','color')
pivoted
```

	price	
color	**Blue**	**Red**
type		
P1	120	100
P2	90	140
P3	105	110

See Data_wrangling6.ipynb

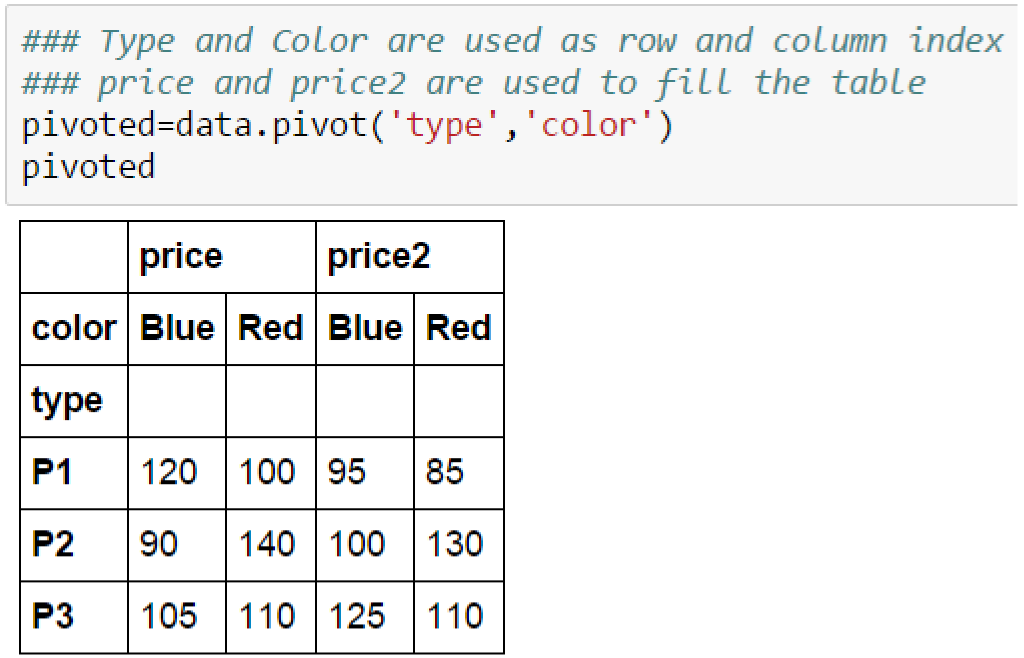
QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (Python)



```
### Adding another column of price2
data['price2'] = [85,95,130,100,110,125]
data
```

	type	color	price	price2
0	P1	Red	100	85
1	P1	Blue	120	95
2	P2	Red	140	130
3	P2	Blue	90	100
4	P3	Red	110	110
5	P3	Blue	105	125



```
### Type and Color are used as row and column index
### price and price2 are used to fill the table
pivoted=data.pivot('type','color')
pivoted
```

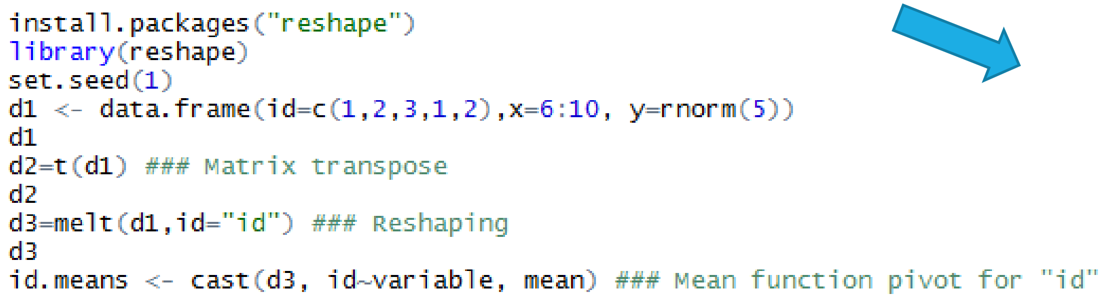
	price		**price2**	
color	**Blue**	**Red**	**Blue**	**Red**
type				
P1	120	100	95	85
P2	90	140	100	130
P3	105	110	125	110

See Data_wrangling6.ipynb

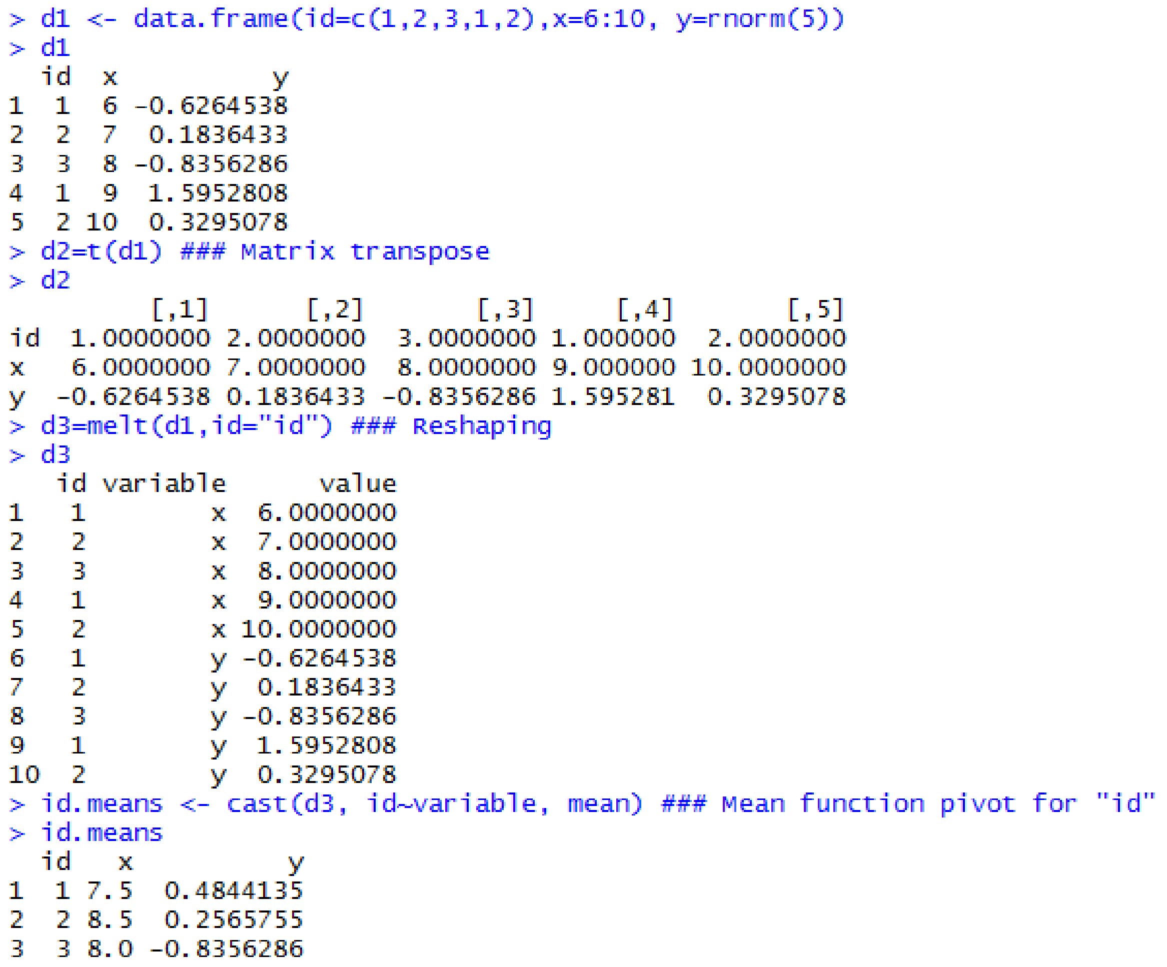
QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (R)

- Reshape package, melt and cast function deal with reshaping and pivoting data frame in R.



```r
install.packages("reshape")
library(reshape)
set.seed(1)
d1 <- data.frame(id=c(1,2,3,1,2),x=6:10, y=rnorm(5))
d1
d2=t(d1) ### Matrix transpose
d2
d3=melt(d1,id="id") ### Reshaping
d3
id.means <- cast(d3, id~variable, mean) ### Mean function pivot for "id"
```



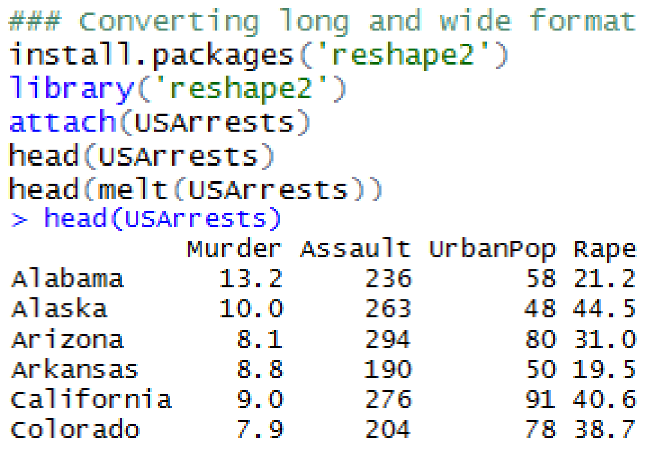
```
> d1 <- data.frame(id=c(1,2,3,1,2),x=6:10, y=rnorm(5))
> d1
  id  x          y
1  1  6 -0.6264538
2  2  7  0.1836433
3  3  8 -0.8356286
4  1  9  1.5952808
5  2 10  0.3295078
> d2=t(d1) ### Matrix transpose
> d2
         [,1]      [,2]       [,3]     [,4]       [,5]
id  1.0000000 2.0000000  3.0000000 1.000000  2.0000000
x   6.0000000 7.0000000  8.0000000 9.000000 10.0000000
y  -0.6264538 0.1836433 -0.8356286 1.595281  0.3295078
> d3=melt(d1,id="id") ### Reshaping
> d3
   id variable      value
1   1        x  6.0000000
2   2        x  7.0000000
3   3        x  8.0000000
4   1        x  9.0000000
5   2        x 10.0000000
6   1        y -0.6264538
7   2        y  0.1836433
8   3        y -0.8356286
9   1        y  1.5952808
10  2        y  0.3295078
> id.means <- cast(d3, id~variable, mean) ### Mean function pivot for "id"
> id.means
  id   x          y
1  1 7.5  0.4844135
2  2 8.5  0.2565755
3  3 8.0 -0.8356286
```

See Data_wrangling5.R

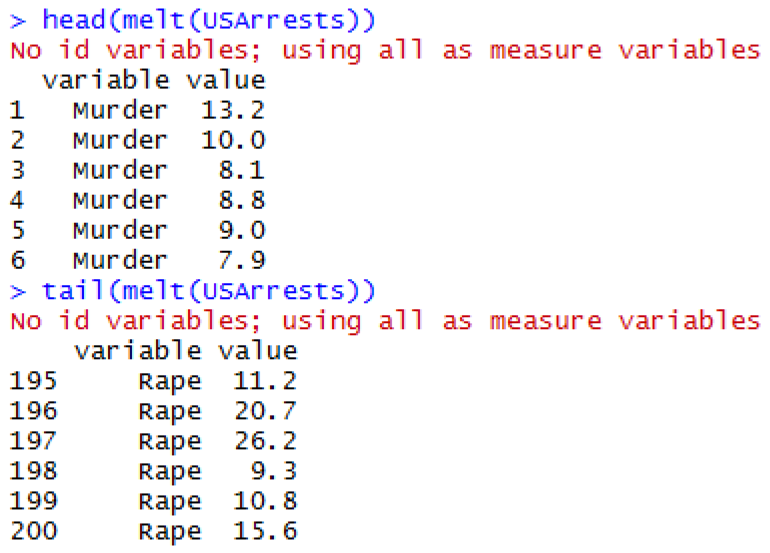
QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (R)

- In R using Reshape2 package, you can change the long and wide format to each other through melt and dcast syntax:
  - melt converts wide format to long format, while dcast changes long format data to wide one.



```
### Converting long and wide format
install.packages('reshape2')
library('reshape2')
attach(USArrests)
head(USArrests)
head(melt(USArrests))
> head(USArrests)
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

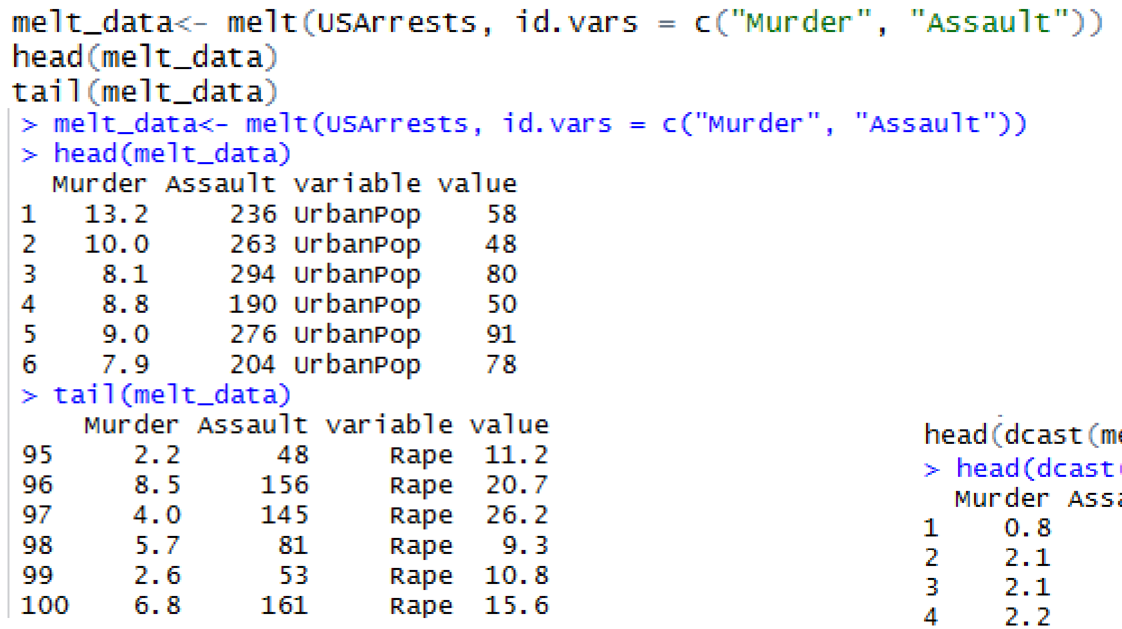


```
> head(melt(USArrests))
No id variables; using all as measure variables
  variable value
1   Murder  13.2
2   Murder  10.0
3   Murder   8.1
4   Murder   8.8
5   Murder   9.0
6   Murder   7.9
> tail(melt(USArrests))
No id variables; using all as measure variables
    variable value
195     Rape  11.2
196     Rape  20.7
197     Rape  26.2
198     Rape   9.3
199     Rape  10.8
200     Rape  15.6
```

See Data_wrangling5.R

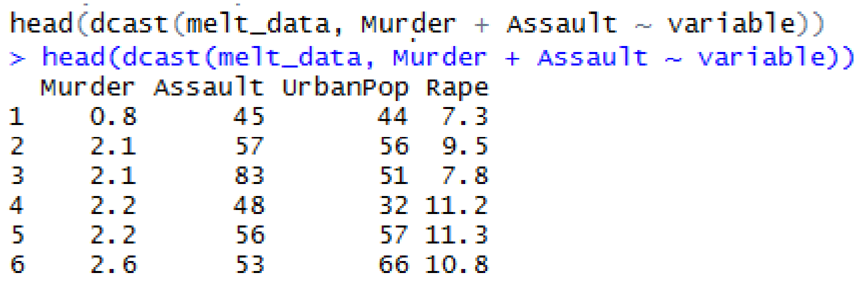
QuantUniversity, LLC
www.quantuniversity.com

Reshaping and pivoting (R)



```
melt_data<- melt(USArrests, id.vars = c("Murder", "Assault"))
head(melt_data)
tail(melt_data)
```

```
> melt_data<- melt(USArrests, id.vars = c("Murder", "Assault"))
> head(melt_data)
  Murder Assault variable value
1   13.2     236 UrbanPop    58
2   10.0     263 UrbanPop    48
3    8.1     294 UrbanPop    80
4    8.8     190 UrbanPop    50
5    9.0     276 UrbanPop    91
6    7.9     204 UrbanPop    78
> tail(melt_data)
    Murder Assault variable value
95     2.2      48     Rape  11.2
96     8.5     156     Rape  20.7
97     4.0     145     Rape  26.2
98     5.7      81     Rape   9.3
99     2.6      53     Rape  10.8
100    6.8     161     Rape  15.6
```



```
head(dcast(melt_data, Murder + Assault ~ variable))
```

```
> head(dcast(melt_data, Murder + Assault ~ variable))
  Murder Assault UrbanPop Rape
1    0.8      45       44  7.3
2    2.1      57       56  9.5
3    2.1      83       51  7.8
4    2.2      48       32 11.2
5    2.2      56       57 11.3
6    2.6      53       66 10.8
```

See Data_wrangling5.R

QuantUniversity, LLC
www.quantuniversity.com

Filtering (Python)

- Usually, slicing is used to filter out some rows/columns of a data frame based on filtering condition(s).

```
import pandas as pd
from pandas import *
data = DataFrame(np.arange(16).reshape((4, 4)),
                 index=['Ohio', 'Colorado', 'Utah', 'New York'],
                 columns=['one', 'two', 'three', 'four'])
data
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
### Filter out some columns
data[['one', 'two']]
```

	one	two
Ohio	0	1
Colorado	4	5
Utah	8	9
New York	12	13

```
### Filter out some rows
data[:2]
```

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7

See Data_wrangling6-1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Filtering (Python)

```
### Filter out rows based on condition(s) on column(s)
data[(data['four'] > 7) & (data['three'] > 7)]
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

```
### Filter out rows and columns together based on condition on column
data.ix[data.three > 5, :3]
```

	one	two	three
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

```
### Filter out rows and columns together based on condition on column
data.ix[(data['one'] > 7) & (data['two'] > 7), :3]
```

	one	two	three
Utah	8	9	10
New York	12	13	14

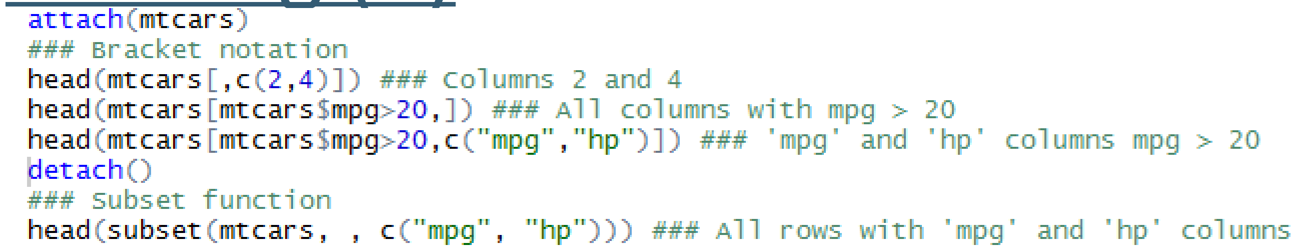
See Data_wrangling6-1.ipynb

QuantUniversity, LLC
www.quantuniversity.com

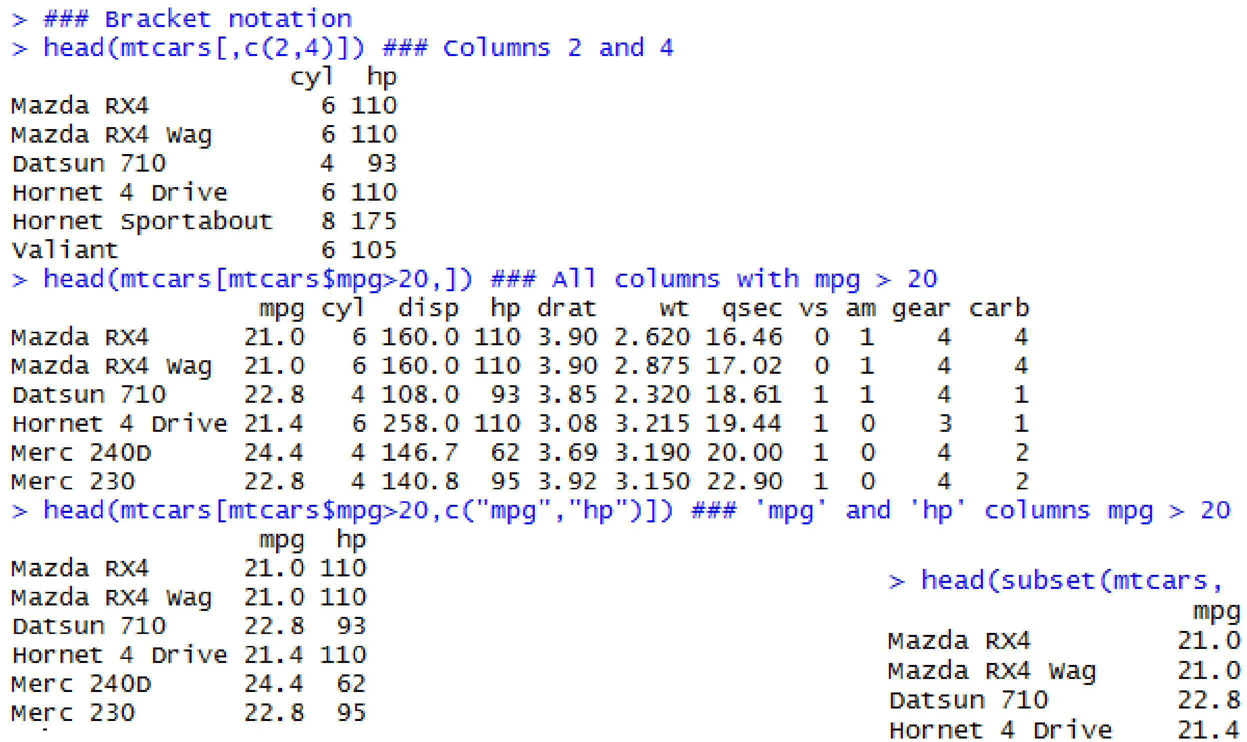
Filtering (R)

- Some times you may just need to work with some columns of data or filter some variables. You can do these tasks by :
  - Bracket notation
  - Filter, subset and select functions
- The ‘dplyr’ %>% chaining operation allows you to execute multiple command on a data frame at a same time.
- The ‘dplyr’ package, allows to manipulate data frames more faster and rational for multiple tasks.

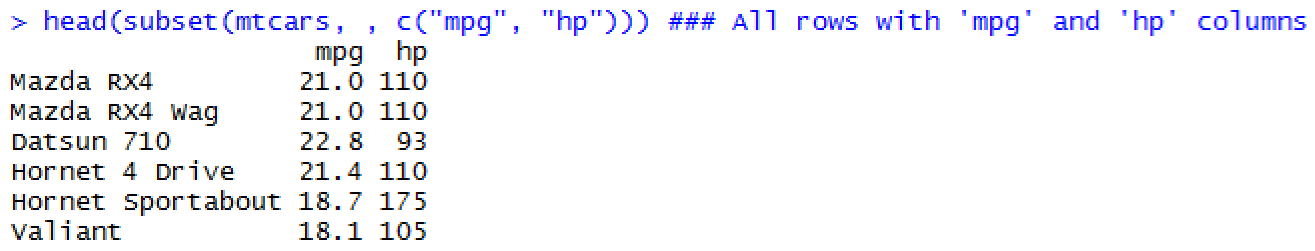
Filtering (R)



```
attach(mtcars)
### Bracket notation
head(mtcars[,c(2,4)]) ### Columns 2 and 4
head(mtcars[mtcars$mpg>20,]) ### All columns with mpg > 20
head(mtcars[mtcars$mpg>20,c("mpg","hp")]) ### 'mpg' and 'hp' columns mpg > 20
detach()
### Subset function
head(subset(mtcars, , c("mpg", "hp"))) ### All rows with 'mpg' and 'hp' columns
```



```
> ### Bracket notation
> head(mtcars[,c(2,4)]) ### Columns 2 and 4
                  cyl  hp
Mazda RX4           6 110
Mazda RX4 Wag       6 110
Datsun 710          4  93
Hornet 4 Drive      6 110
Hornet Sportabout   8 175
Valiant             6 105
> head(mtcars[mtcars$mpg>20,]) ### All columns with mpg > 20
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
> head(mtcars[mtcars$mpg>20,c("mpg","hp")]) ### 'mpg' and 'hp' columns mpg > 20
                mpg  hp
Mazda RX4      21.0 110
Mazda RX4 Wag  21.0 110
Datsun 710     22.8  93
Hornet 4 Drive 21.4 110
Merc 240D      24.4  62
Merc 230       22.8  95
```

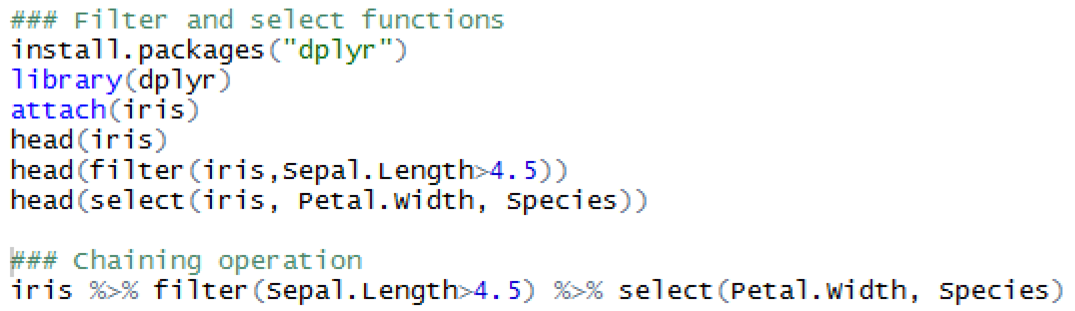


```
> head(subset(mtcars, , c("mpg", "hp"))) ### All rows with 'mpg' and 'hp' columns
                   mpg  hp
Mazda RX4         21.0 110
Mazda RX4 Wag     21.0 110
Datsun 710        22.8  93
Hornet 4 Drive    21.4 110
Hornet Sportabout 18.7 175
Valiant           18.1 105
```

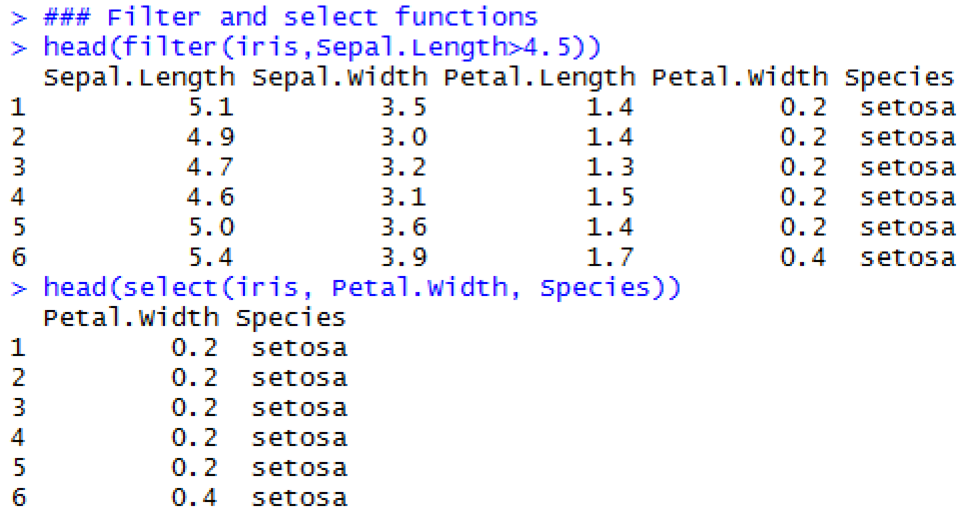
See Data_wrangling5.R

QuantUniversity, LLC
www.quantuniversity.com

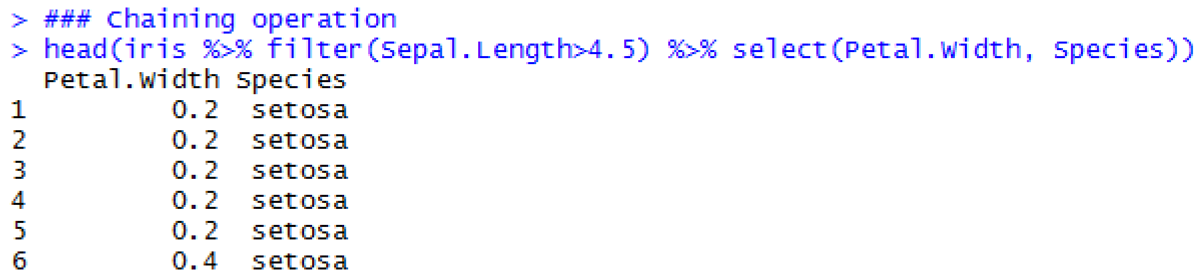
Filtering (R)



```
### Filter and select functions
install.packages("dplyr")
library(dplyr)
attach(iris)
head(iris)
head(filter(iris,Sepal.Length>4.5))
head(select(iris, Petal.Width, Species))

### Chaining operation
iris %>% filter(Sepal.Length>4.5) %>% select(Petal.Width, Species)
```



```
> ### Filter and select functions
> head(filter(iris,Sepal.Length>4.5))
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> head(select(iris, Petal.Width, Species))
  Petal.Width Species
1         0.2  setosa
2         0.2  setosa
3         0.2  setosa
4         0.2  setosa
5         0.2  setosa
6         0.4  setosa
```



```
> ### Chaining operation
> head(iris %>% filter(Sepal.Length>4.5) %>% select(Petal.Width, Species))
  Petal.Width Species
1         0.2  setosa
2         0.2  setosa
3         0.2  setosa
4         0.2  setosa
5         0.2  setosa
6         0.4  setosa
```

See Data_wrangling5.R

QuantUniversity, LLC

www.quantuniversity.com

Sorting (Python)

- In python, using sort() function you can sort on either single or multiple columns in ascending or descending from.

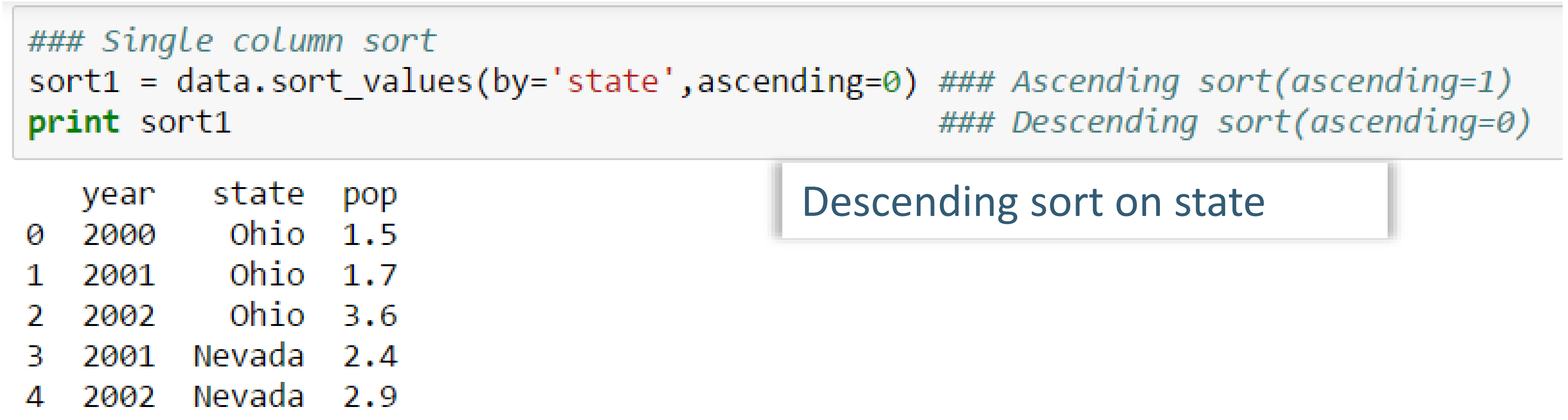
```
import pandas as pd
from pandas import *
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
data = DataFrame(data, columns=['year', 'state', 'pop'])
print data
```

```
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
```

See Data_wrangling6-2.ipynb

QuantUniversity, LLC
www.quantuniversity.com

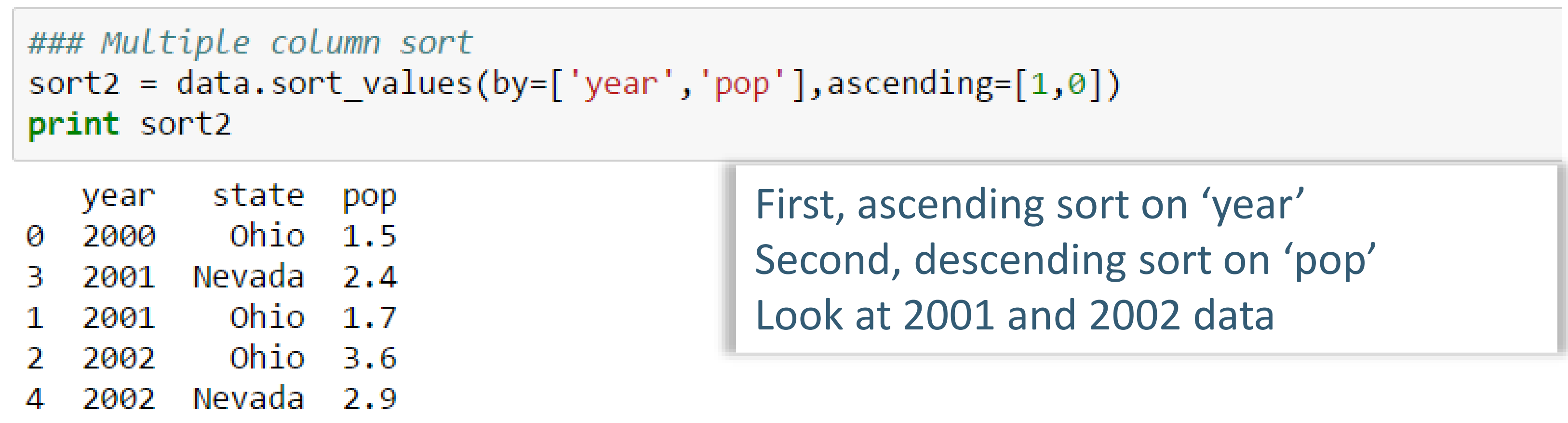
Sorting (Python)



```
### Single column sort
sort1 = data.sort_values(by='state',ascending=0) ### Ascending sort(ascending=1)
print sort1                                      ### Descending sort(ascending=0)
```

```
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
```

Descending sort on state



```
### Multiple column sort
sort2 = data.sort_values(by=['year','pop'],ascending=[1,0])
print sort2
```

```
   year   state  pop
0  2000    Ohio  1.5
3  2001  Nevada  2.4
1  2001    Ohio  1.7
2  2002    Ohio  3.6
4  2002  Nevada  2.9
```

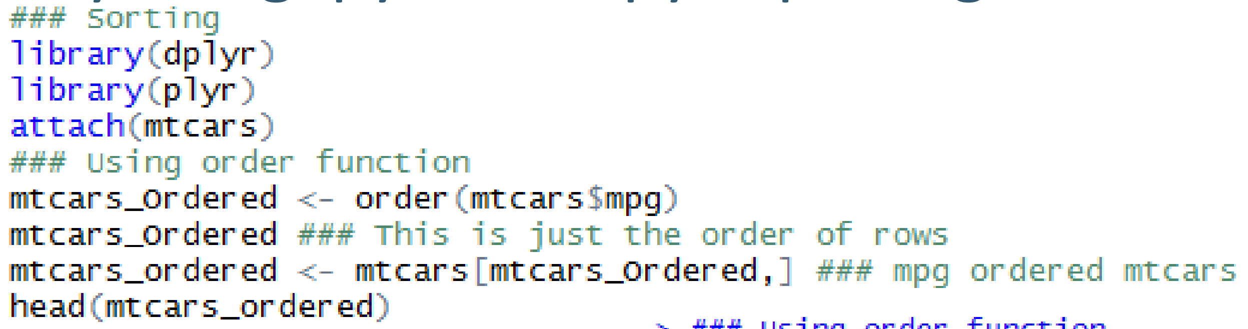
First, ascending sort on ‘year’
Second, descending sort on ‘pop’
Look at 2001 and 2002 data

See Data_wrangling6-2.ipynb

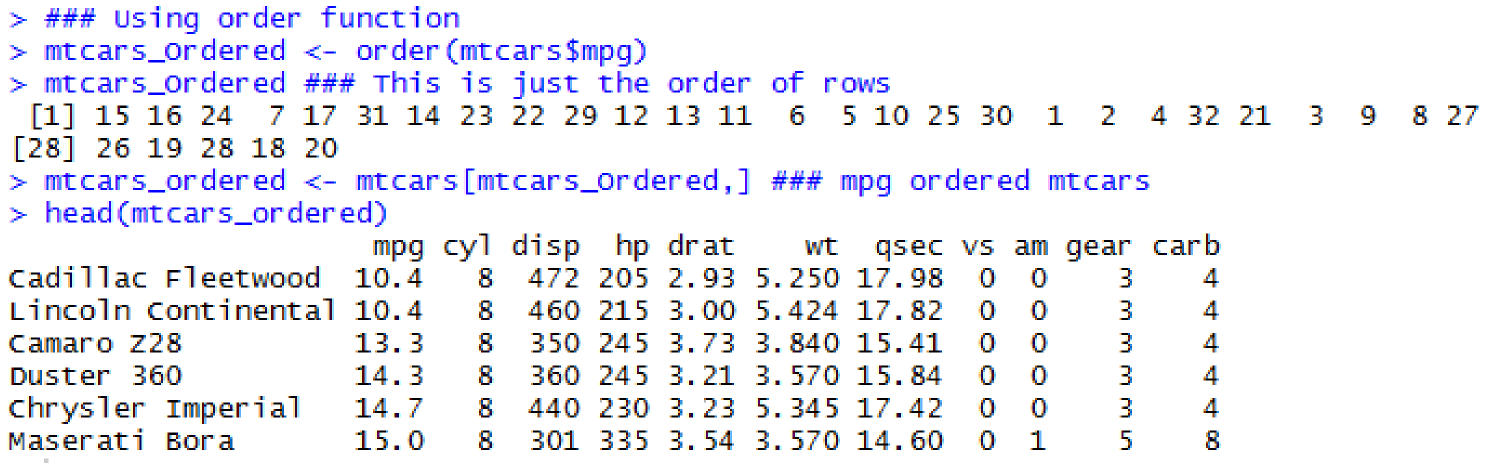
QuantUniversity, LLC
www.quantuniversity.com

Sorting (R)

- In R, you can either use built in order() function or arrange() syntax by using ‘pylr’ or ‘dpylr’ packages.



```
### Sorting
library(dplyr)
library(plyr)
attach(mtcars)
### Using order function
mtcars_Ordered <- order(mtcars$mpg)
mtcars_Ordered ### This is just the order of rows
mtcars_ordered <- mtcars[mtcars_Ordered,] ### mpg ordered mtcars
head(mtcars_ordered)
```



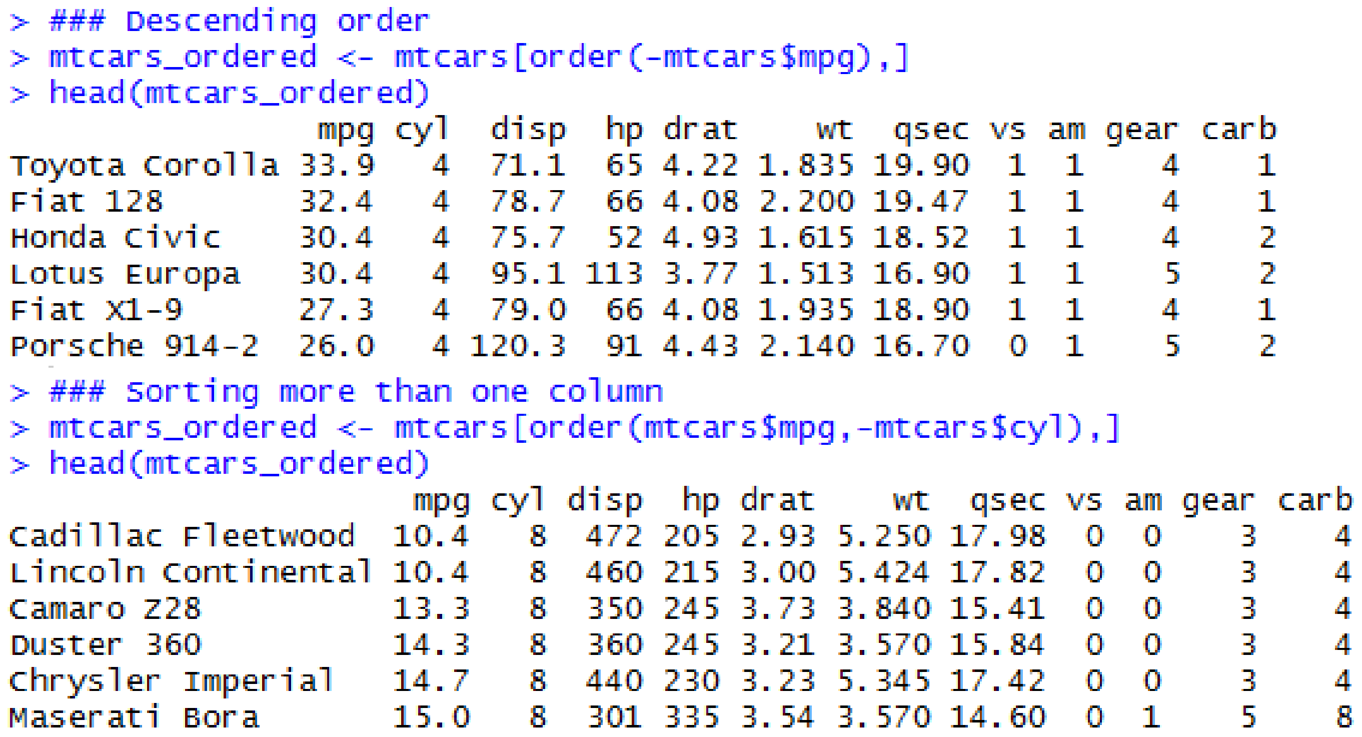
```
> ### Using order function
> mtcars_Ordered <- order(mtcars$mpg)
> mtcars_Ordered ### This is just the order of rows
 [1] 15 16 24  7 17 31 14 23 22 29 12 13 11  6  5 10 25 30  1  2  4 32 21  3  9  8 27
[28] 26 19 28 18 20
> mtcars_ordered <- mtcars[mtcars_Ordered,] ### mpg ordered mtcars
> head(mtcars_ordered)
                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

See Data_wrangling5-1.R

QuantUniversity, LLC
www.quantuniversity.com

Sorting (R)

```
### Descending order
mtcars_ordered <- mtcars[order(-mtcars$mpg),]
head(mtcars_ordered)

### Sorting more than one column
mtcars_ordered <- mtcars[order(mtcars$mpg,-mtcars$cyl),]
head(mtcars_ordered)
### or
mtcars_ordered <- mtcars[with(mtcars,order(mpg,-cyl)),]
head(mtcars_ordered)
```



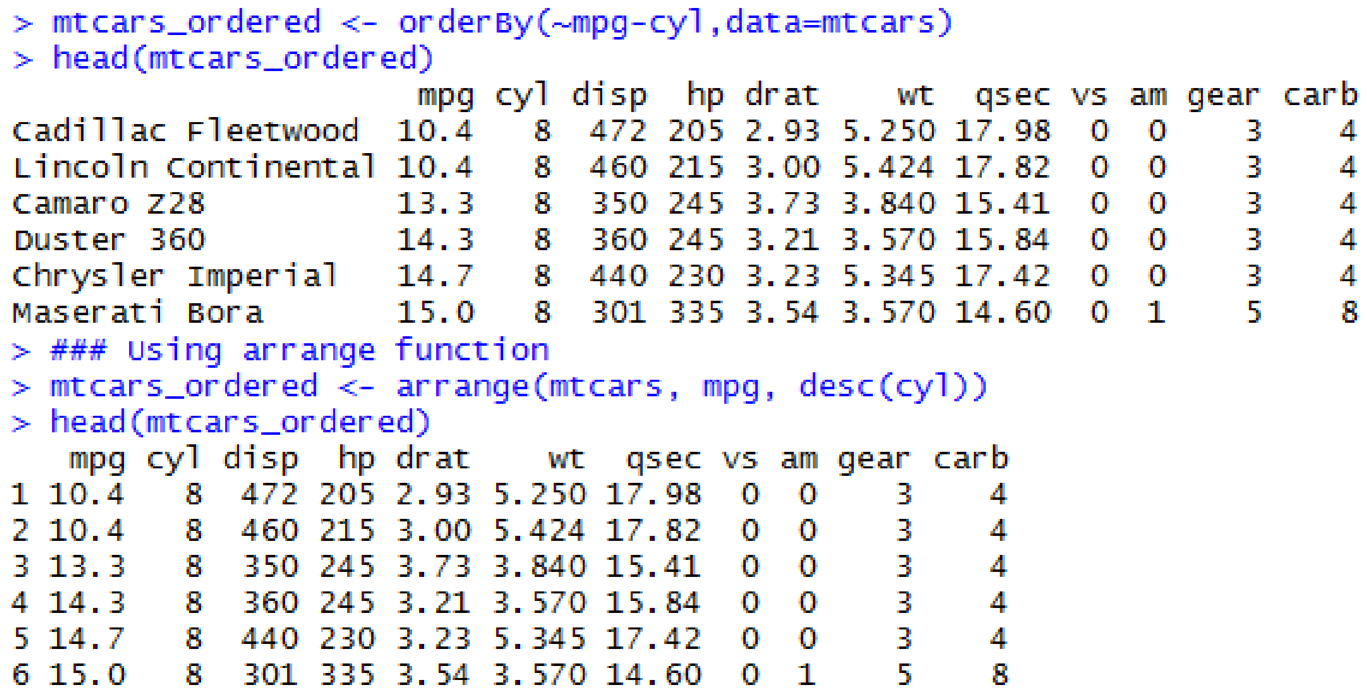
```
> ### Descending order
> mtcars_ordered <- mtcars[order(-mtcars$mpg),]
> head(mtcars_ordered)
                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
> ### Sorting more than one column
> mtcars_ordered <- mtcars[order(mtcars$mpg,-mtcars$cyl),]
> head(mtcars_ordered)
                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

See Data_wrangling5-1.R

QuantUniversity, LLC
www.quantuniversity.com

Sorting (R)

```
### Using doBy package
install.packages('doBy')
library(doBy)
mtcars_ordered <- orderBy(~mpg-cyl,data=mtcars)
head(mtcars_ordered)

### Using arrange function
mtcars_ordered <- arrange(mtcars, mpg, desc(cyl))
head(mtcars_ordered)
```



```
> mtcars_ordered <- orderBy(~mpg-cyl,data=mtcars)
> head(mtcars_ordered)
                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
Cadillac Fleetwood  10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
Lincoln Continental 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
Camaro Z28          13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
Duster 360          14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
Chrysler Imperial   14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
Maserati Bora       15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
> ### Using arrange function
> mtcars_ordered <- arrange(mtcars, mpg, desc(cyl))
> head(mtcars_ordered)
   mpg cyl disp  hp drat    wt  qsec vs am gear carb
1 10.4   8  472 205 2.93 5.250 17.98  0  0    3    4
2 10.4   8  460 215 3.00 5.424 17.82  0  0    3    4
3 13.3   8  350 245 3.73 3.840 15.41  0  0    3    4
4 14.3   8  360 245 3.21 3.570 15.84  0  0    3    4
5 14.7   8  440 230 3.23 5.345 17.42  0  0    3    4
6 15.0   8  301 335 3.54 3.570 14.60  0  1    5    8
```

See Data_wrangling5-1.R

QuantUniversity, LLC
www.quantuniversity.com

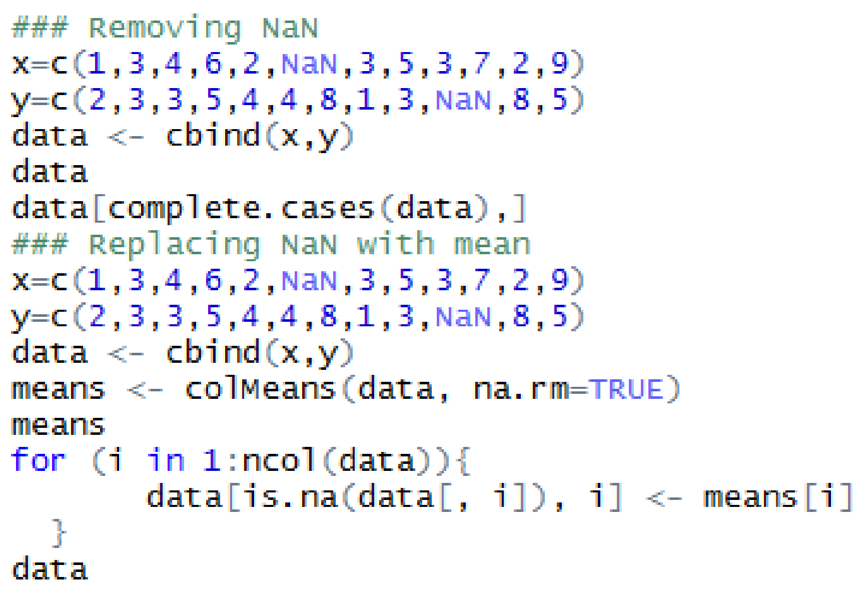
Data cleaning and manipulation

- ✓ Missing values
- ✓ Noisy data
- ✓ Removing duplicates
- ✓ Adding a new column
- ✓ Mapping
- ✓ Replacing values
- ✓ Renaming axes indexes

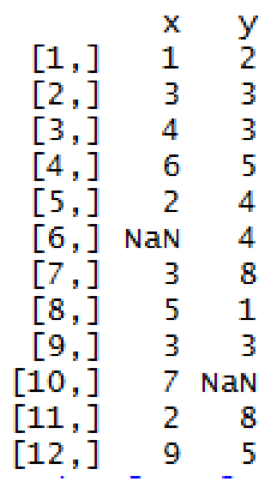
Missing values

- Data cleaning (data cleansing) deals with handling missing values, smooth out noise while identifying outliers, and correct inconsistencies of data.
- There are several ways to manage missing values such as:
  - Ignoring missing values: It is not a effective tools when we have different number of missing values per feature.
  - Fill in missing values manually, which is not feasible for large datasets with too many missing values.
  - Use a global constant to fill in the missing value such as such as “Unknown” or infinity.
  - Use the attribute mean to fill in the missing values
  - Use the attribute mean for all samples belonging to the same class.
  - Use the most probable value to fill in the missing value.

Missing values (R)

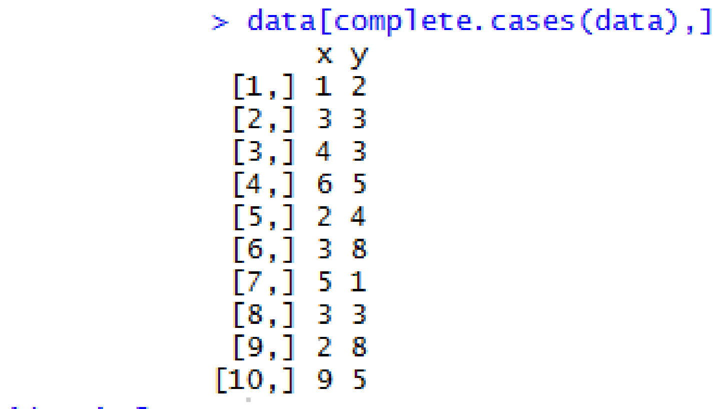


```
### Removing NaN
x=c(1,3,4,6,2,NaN,3,5,3,7,2,9)
y=c(2,3,3,5,4,4,8,1,3,NaN,8,5)
data <- cbind(x,y)
data
data[complete.cases(data),]
### Replacing NaN with mean
x=c(1,3,4,6,2,NaN,3,5,3,7,2,9)
y=c(2,3,3,5,4,4,8,1,3,NaN,8,5)
data <- cbind(x,y)
means <- colMeans(data, na.rm=TRUE)
means
for (i in 1:ncol(data)){
       data[is.na(data[, i]), i] <- means[i]
  }
data
```



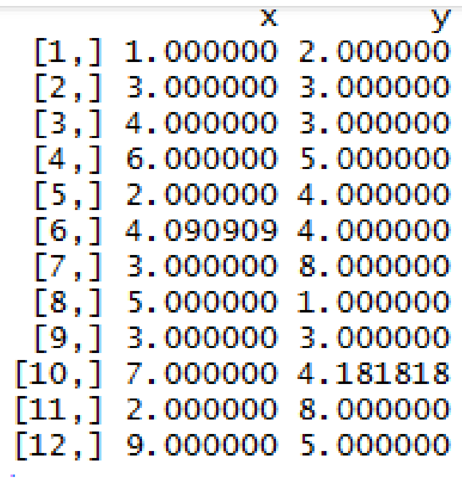
```
        x   y
 [1,]   1   2
 [2,]   3   3
 [3,]   4   3
 [4,]   6   5
 [5,]   2   4
 [6,] NaN   4
 [7,]   3   8
 [8,]   5   1
 [9,]   3   3
[10,]   7 NaN
[11,]   2   8
[12,]   9   5
```

Removing missing values



```
> data[complete.cases(data),]
      x y
 [1,] 1 2
 [2,] 3 3
 [3,] 4 3
 [4,] 6 5
 [5,] 2 4
 [6,] 3 8
 [7,] 5 1
 [8,] 3 3
 [9,] 2 8
[10,] 9 5
```

Replacing with mean



```
             x        y
 [1,] 1.000000 2.000000
 [2,] 3.000000 3.000000
 [3,] 4.000000 3.000000
 [4,] 6.000000 5.000000
 [5,] 2.000000 4.000000
 [6,] 4.090909 4.000000
 [7,] 3.000000 8.000000
 [8,] 5.000000 1.000000
 [9,] 3.000000 3.000000
[10,] 7.000000 4.181818
[11,] 2.000000 8.000000
[12,] 9.000000 5.000000
```

See Data_preprocessing1.R

QuantUniversity, LLC
www.quantuniversity.com

Missing values (Python)

```
### Working with NaN in Pandas DataFrame
import numpy as np
import pandas as pd
import scipy
x=[1,5,9,np.NaN]
x=pd.DataFrame(x,columns=['data'])
print x
print "Mean=", (scipy.mean(x['data']))
print x.dropna() ### Removing strategy
print x.fillna(scipy.mean(x['data'])) ### Mean strategy
```

```
   data
0     1
1     5
2     9
3   NaN
Mean= 5.0
   data
0     1
1     5
2     9
   data
0     1
1     5
2     9
3     5
```

Removing NaN

Replacing with mean

See Data_preprocessing2.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Noisy data

- Noise is a random error in a measured variable.
- We may be able to remove the noise by applying methods such as:
    - Smoothing by bin means / medians: Each bin value is replaced by the bin mean / median.
    - Smoothing by bin boundaries: Each bin value is replaced by the closest boundary value. (Min or max value of the boundary)
    - Regression: Data can be smoothed through fitting a function to data by applying linear regression or multiple linear regression.
    - Outliers may be detected by clustering, where similar values are organized into groups, or clusters and values outside of the clusters considered as outliers.

Detecting and filtering outliers (R)

```
### Detecting outliers
### First replace missing values with zero
data <- data.frame(x=c(.1,.2,NaN,-20,.8,.9,.5,.1,1),y=c(2,NaN,.5,20,1,.3,.1,.8,.9))
data
for(i in 1:ncol(data)){
  data[is.na(data[,i]), i] <- 0
}
data
### Detecting, filtering outliers and replacing with mean of column
for(j in 1:ncol(data)){
  for (i in 1:nrow(data)){
    if ((data[i,j] < (mean(data[[j]])-(1.5)*sd(data[[j]]))|
        (data[i,j] > (mean(data[[j]])+1.5*sd(data[[j]])))))
    {data[i,j]<-mean(data[[j]])}
  }
}
data
```

Removing NaN

Replacing outlier values with mean of column

```
      x     y
1   0.1   2.0
2   0.2   NaN
3   NaN   0.5
4 -20.0  20.0
5   0.8   1.0
6   0.9   0.3
7   0.5   0.1
8   0.1   0.8
9   1.0   0.9
```

→

```
      x     y
1   0.1   2.0
2   0.2   0.0
3   0.0   0.5
4 -20.0  20.0
5   0.8   1.0
6   0.9   0.3
7   0.5   0.1
8   0.1   0.8
9   1.0   0.9
```

→

```
          x        y
1  0.100000 2.000000
2  0.200000 0.000000
3  0.000000 0.500000
4 -1.822222 2.844444
5  0.800000 1.000000
6  0.900000 0.300000
7  0.500000 0.100000
8  0.100000 0.800000
9  1.000000 0.900000
```

See Data_preprocessing2.R

QuantUniversity, LLC
www.quantuniversity.com

Detecting and filtering outliers (Python)

```
### Detecting and filtering outliers
from numpy.random import randn
from pandas import *
np.random.seed(1)
data=DataFrame(np.random.randn(10,2))
print "data:"
print  data
print "Outliers:"
print data[(np.abs(data) > 1.5).any(1)] ### finding outliers (It could be any specific value)
data[np.abs(data) > 1.5] = np.sign(data) * 1.5 ### Replcing outliers with any desirable value
print "data:"
print data
```

```
data:
          0         1
0  1.624345 -0.611756
1 -0.528172 -1.072969
2  0.865408 -2.301539
3  1.744812 -0.761207
4  0.319039 -0.249370
5  1.462108 -2.060141
6 -0.322417 -0.384054
7  1.133769 -1.099891
8 -0.172428 -0.877858
9  0.042214  0.582815
```

→

```
Outliers:
          0         1
0  1.624345 -0.611756
2  0.865408 -2.301539
3  1.744812 -0.761207
5  1.462108 -2.060141
```

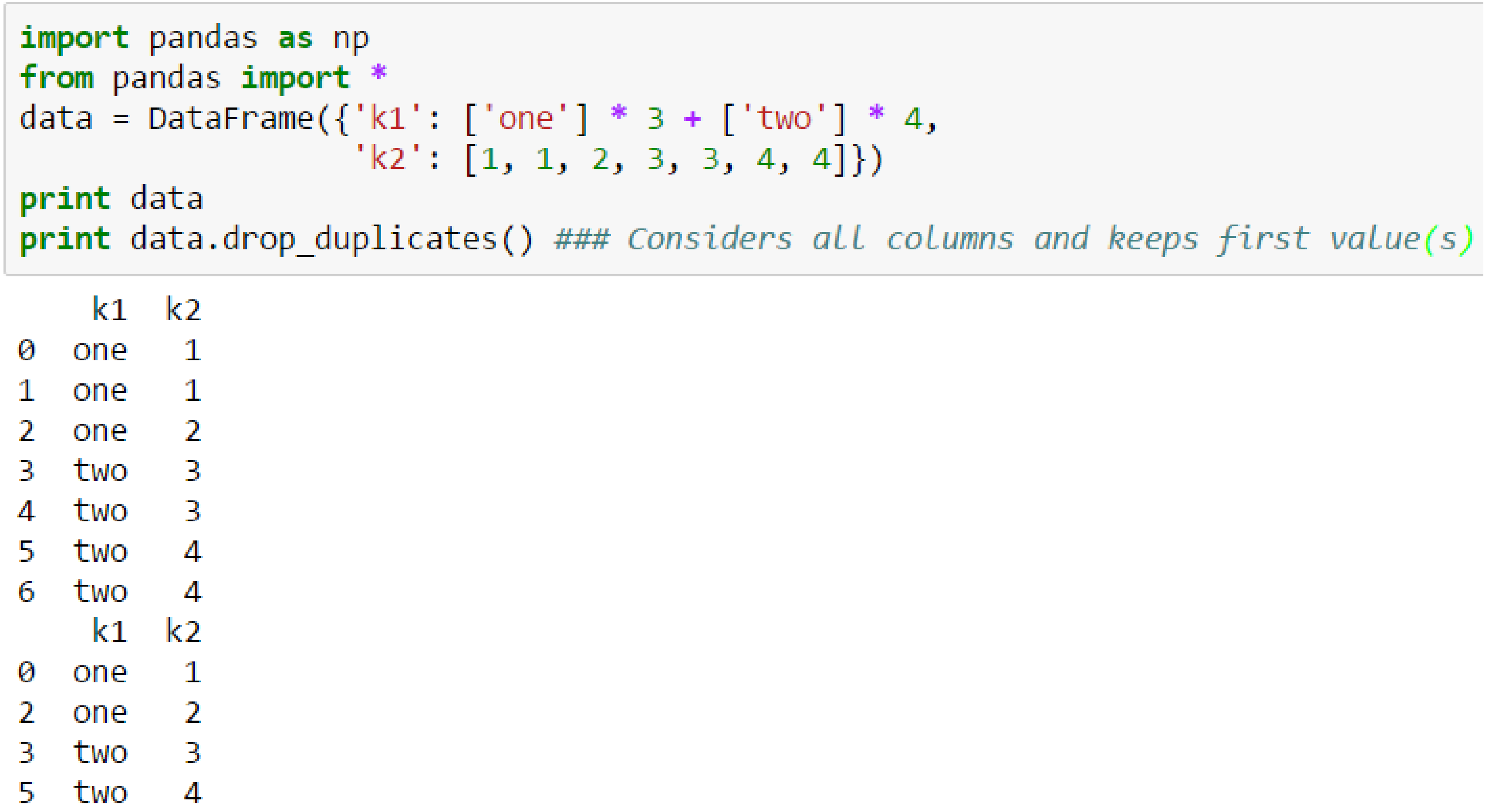
→

```
data:
          0         1
0  1.500000 -0.611756
1 -0.528172 -1.072969
2  0.865408 -1.500000
3  1.500000 -0.761207
4  0.319039 -0.249370
5  1.462108 -1.500000
6 -0.322417 -0.384054
7  1.133769 -1.099891
8 -0.172428 -0.877858
9  0.042214  0.582815
```

See Data_preprocessing2.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Removing duplicates (Python)



```python
import pandas as np
from pandas import *
data = DataFrame({'k1': ['one'] * 3 + ['two'] * 4,
                  'k2': [1, 1, 2, 3, 3, 4, 4]})
print data
print data.drop_duplicates() ### Considers all columns and keeps first value(s)
```

```
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
    k1  k2
0  one   1
2  one   2
3  two   3
5  two   4
```

See Data_wrangling7.ipynb

Removing duplicates (Python)

```
data['v1'] = range(7)
print data
print data.drop_duplicates(['k1']) ### Considers k1 and keeps first value(s)
```

```
    k1  k2  v1
0  one   1   0
1  one   1   1
2  one   2   2
3  two   3   3
4  two   3   4
5  two   4   5
6  two   4   6
    k1  k2  v1
0  one   1   0
3  two   3   3
```

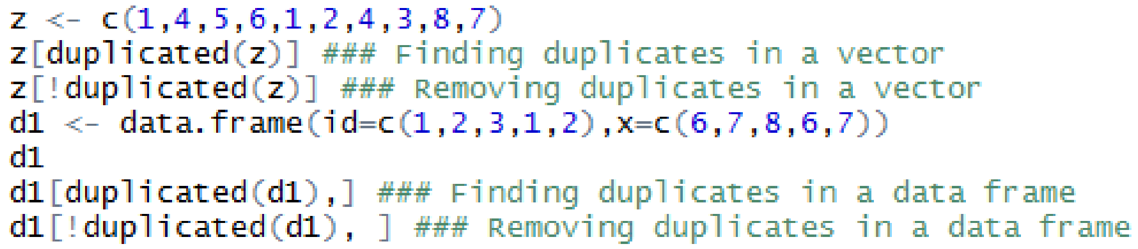
```
print data.drop_duplicates(['k1', 'k2'], keep='last') ### Considers all columns and keeps last value(s)
```

```
    k1  k2  v1
1  one   1   1
2  one   2   2
4  two   3   4
6  two   4   6
```

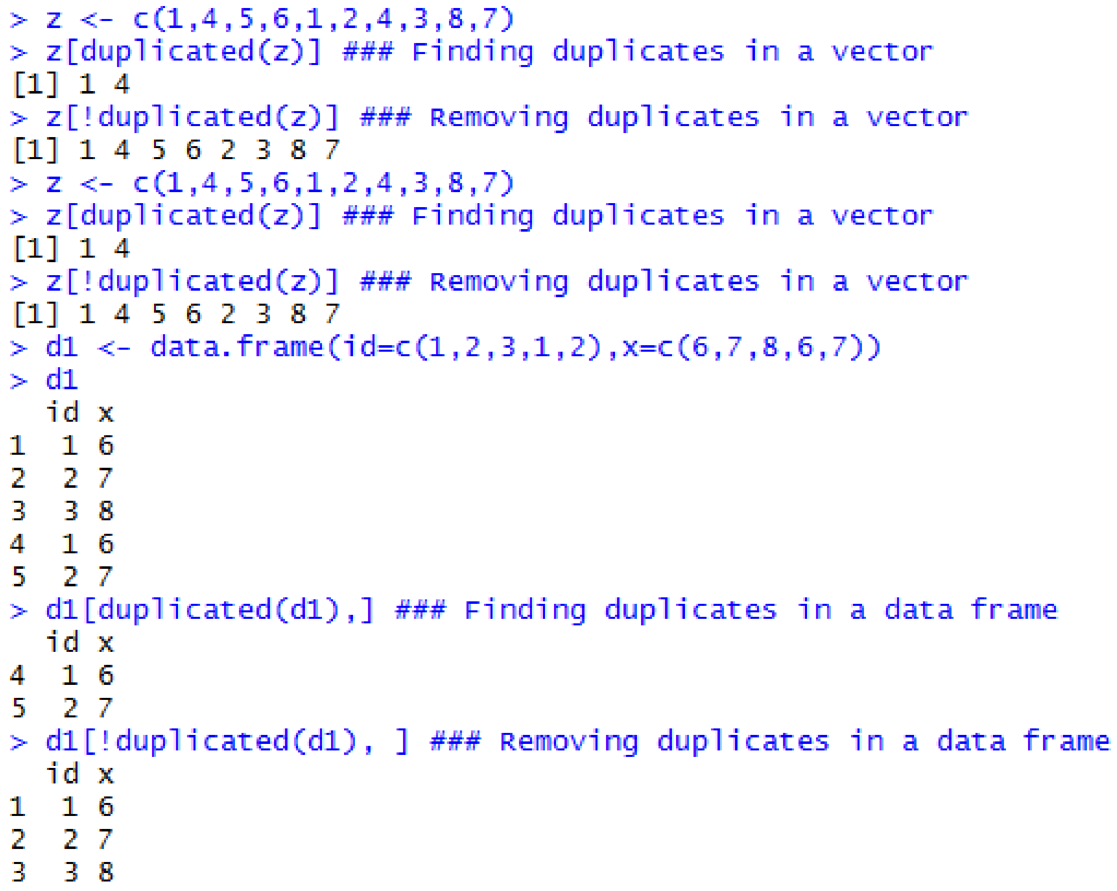
See Data_wrangling7.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Removing duplicates (R)



```
z <- c(1,4,5,6,1,2,4,3,8,7)
z[duplicated(z)] ### Finding duplicates in a vector
z[!duplicated(z)] ### Removing duplicates in a vector
d1 <- data.frame(id=c(1,2,3,1,2),x=c(6,7,8,6,7))
d1
d1[duplicated(d1),] ### Finding duplicates in a data frame
d1[!duplicated(d1), ] ### Removing duplicates in a data frame
```



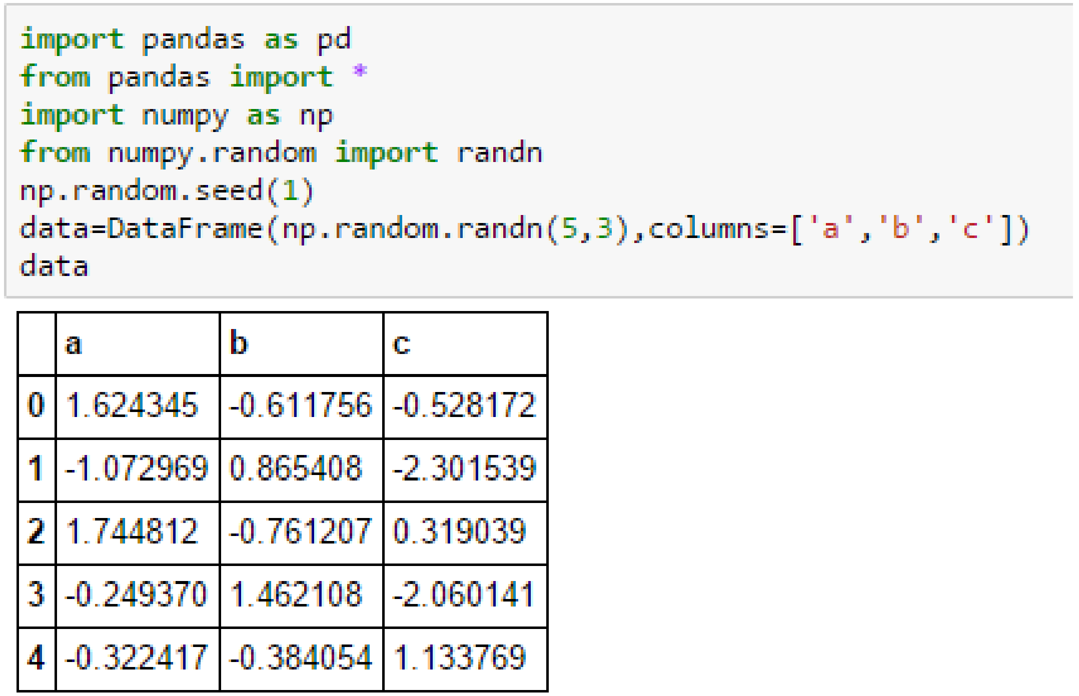
```
> z <- c(1,4,5,6,1,2,4,3,8,7)
> z[duplicated(z)] ### Finding duplicates in a vector
[1] 1 4
> z[!duplicated(z)] ### Removing duplicates in a vector
[1] 1 4 5 6 2 3 8 7
> z <- c(1,4,5,6,1,2,4,3,8,7)
> z[duplicated(z)] ### Finding duplicates in a vector
[1] 1 4
> z[!duplicated(z)] ### Removing duplicates in a vector
[1] 1 4 5 6 2 3 8 7
> d1 <- data.frame(id=c(1,2,3,1,2),x=c(6,7,8,6,7))
> d1
  id x
1  1 6
2  2 7
3  3 8
4  1 6
5  2 7
> d1[duplicated(d1),] ### Finding duplicates in a data frame
  id x
4  1 6
5  2 7
> d1[!duplicated(d1), ] ### Removing duplicates in a data frame
  id x
1  1 6
2  2 7
3  3 8
```

See Data_wrangling6.R

QuantUniversity, LLC
www.quantuniversity.com

Adding a new column (Python)

- In R, adding new column can be done by writing equation or using apply() and lambda.



```
import pandas as pd
from pandas import *
import numpy as np
from numpy.random import randn
np.random.seed(1)
data=DataFrame(np.random.randn(5,3),columns=['a','b','c'])
data
```

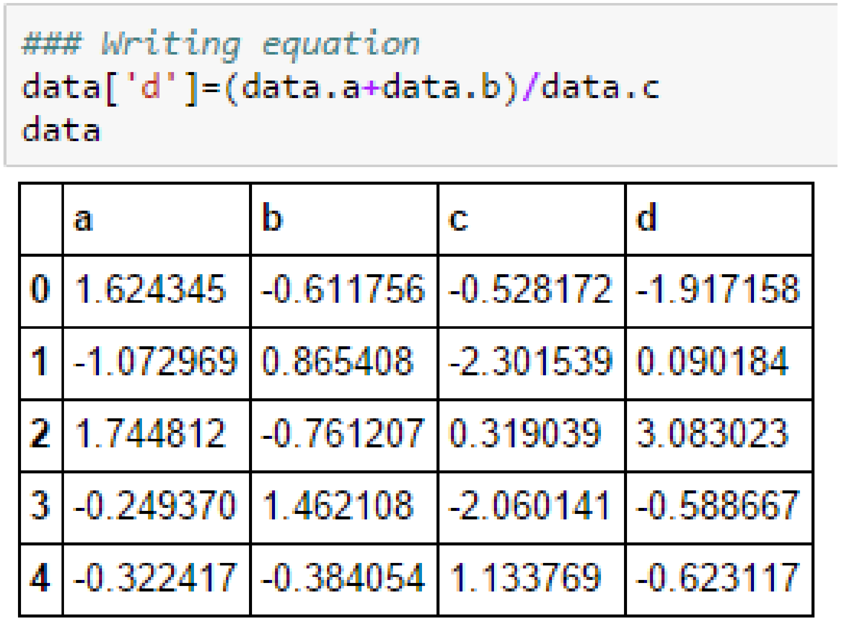
	a	b	c
0	1.624345	-0.611756	-0.528172
1	-1.072969	0.865408	-2.301539
2	1.744812	-0.761207	0.319039
3	-0.249370	1.462108	-2.060141
4	-0.322417	-0.384054	1.133769

See Data_wrangling8.ipynb

QuantUniversity, LLC
www.quantuniversity.com

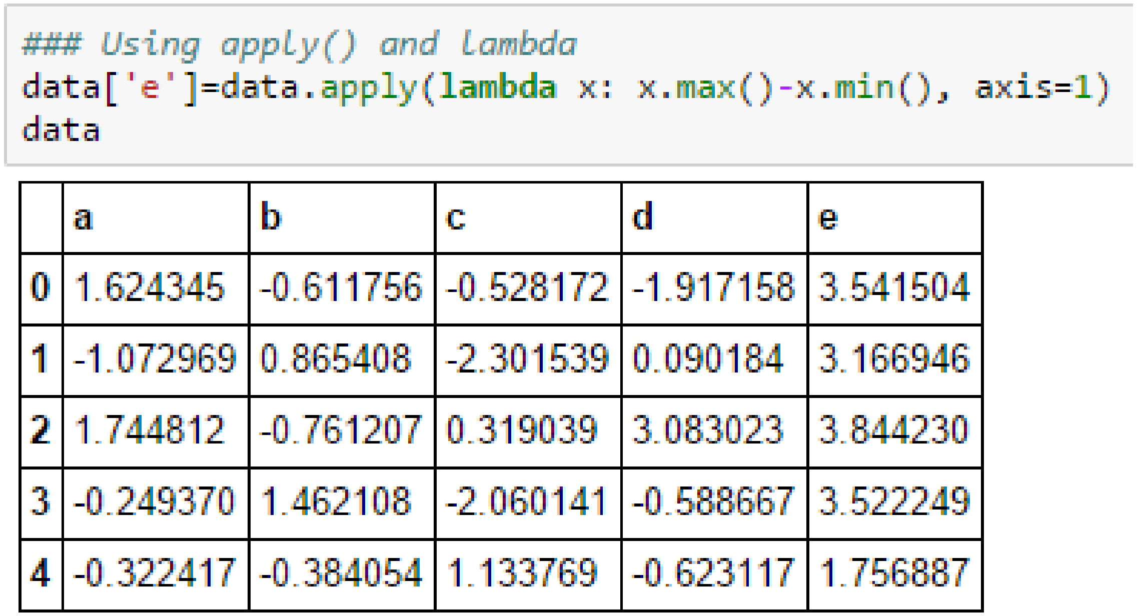
Adding a new column (Python)

- In R, adding new column can be done by writing equation or using apply() and lambda.



```
### Writing equation
data['d']=(data.a+data.b)/data.c
data
```

	a	b	c	d
0	1.624345	-0.611756	-0.528172	-1.917158
1	-1.072969	0.865408	-2.301539	0.090184
2	1.744812	-0.761207	0.319039	3.083023
3	-0.249370	1.462108	-2.060141	-0.588667
4	-0.322417	-0.384054	1.133769	-0.623117



```
### Using apply() and lambda
data['e']=data.apply(lambda x: x.max()-x.min(), axis=1)
data
```

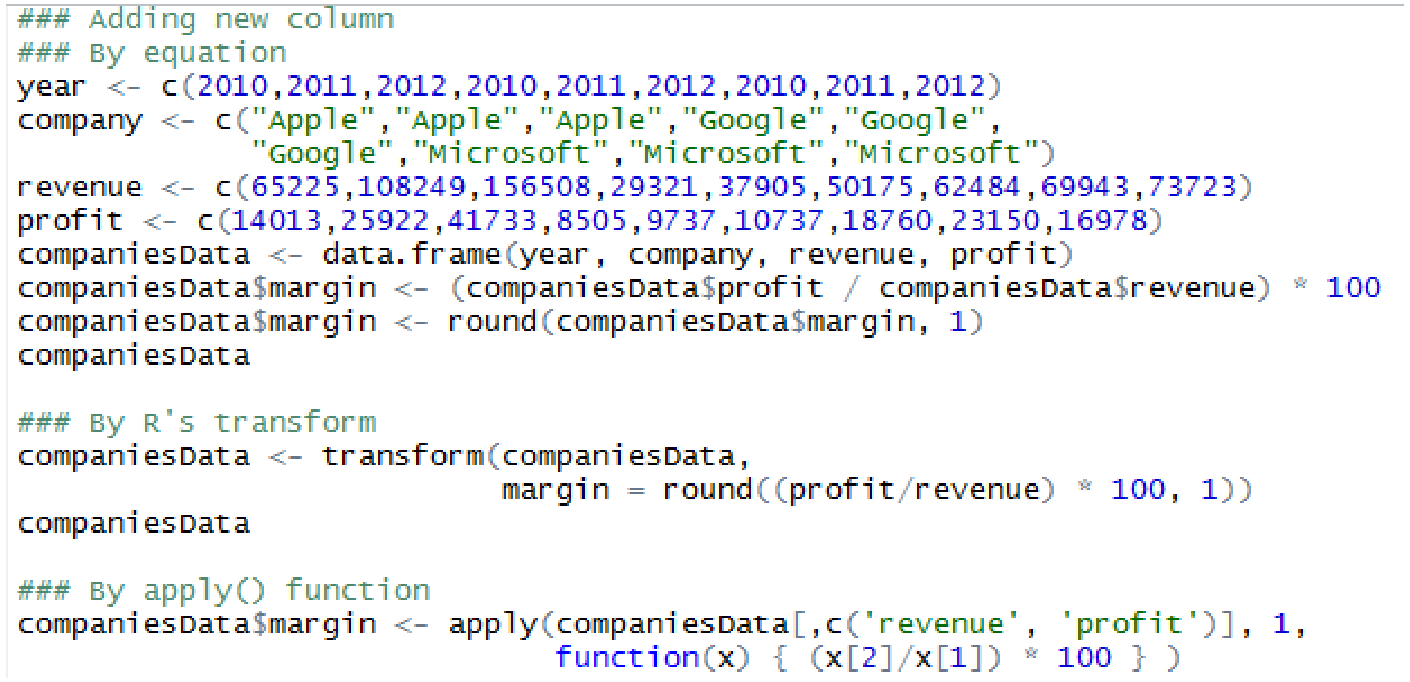
	a	b	c	d	e
0	1.624345	-0.611756	-0.528172	-1.917158	3.541504
1	-1.072969	0.865408	-2.301539	0.090184	3.166946
2	1.744812	-0.761207	0.319039	3.083023	3.844230
3	-0.249370	1.462108	-2.060141	-0.588667	3.522249
4	-0.322417	-0.384054	1.133769	-0.623117	1.756887

See Data_wrangling8.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Adding a new column (R)

- In R, adding new column can be done by writing equation, R's transform, apply() function, mapply() function and 'dplyr' function.

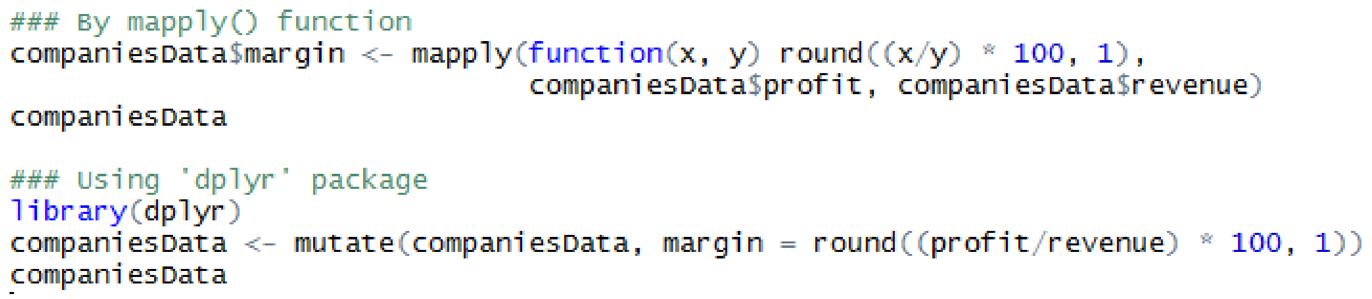


```
### Adding new column
### By equation
year <- c(2010,2011,2012,2010,2011,2012,2010,2011,2012)
company <- c("Apple","Apple","Apple","Google","Google",
             "Google","Microsoft","Microsoft","Microsoft")
revenue <- c(65225,108249,156508,29321,37905,50175,62484,69943,73723)
profit <- c(14013,25922,41733,8505,9737,10737,18760,23150,16978)
companiesData <- data.frame(year, company, revenue, profit)
companiesData$margin <- (companiesData$profit / companiesData$revenue) * 100
companiesData$margin <- round(companiesData$margin, 1)
companiesData

### By R's transform
companiesData <- transform(companiesData,
                           margin = round((profit/revenue) * 100, 1))
companiesData

### By apply() function
companiesData$margin <- apply(companiesData[,c('revenue', 'profit')], 1,
                              function(x) { (x[2]/x[1]) * 100 } )
```

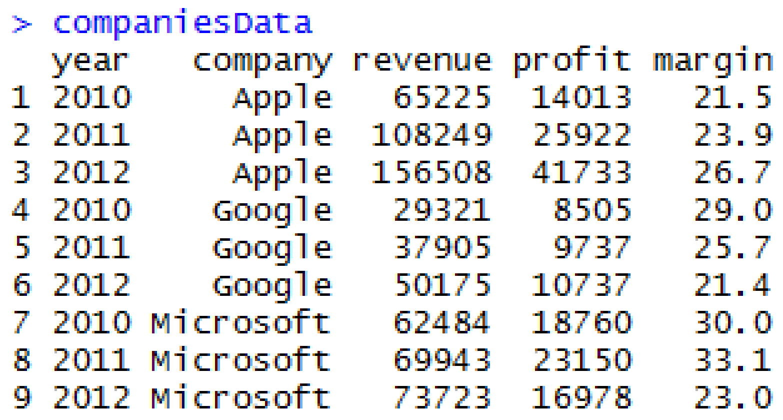
See Data_wrangling7.R

QuantUniversity, LLC
www.quantuniversity.com

Adding a new column (R)



```
### By mapply() function
companiesData$margin <- mapply(function(x, y) round((x/y) * 100, 1),
                               companiesData$profit, companiesData$revenue)
companiesData

### Using 'dplyr' package
library(dplyr)
companiesData <- mutate(companiesData, margin = round((profit/revenue) * 100, 1))
companiesData
```



```
> companiesData
  year   company revenue profit margin
1 2010     Apple   65225  14013   21.5
2 2011     Apple  108249  25922   23.9
3 2012     Apple  156508  41733   26.7
4 2010    Google   29321   8505   29.0
5 2011    Google   37905   9737   25.7
6 2012    Google   50175  10737   21.4
7 2010 Microsoft   62484  18760   30.0
8 2011 Microsoft   69943  23150   33.1
9 2012 Microsoft   73723  16978   23.0
```

See Data_wrangling7.R

Mapping values (Python)

- Mapping can be applied when you need to do transformation based on the values of a column in a data frame using map() and lambda.

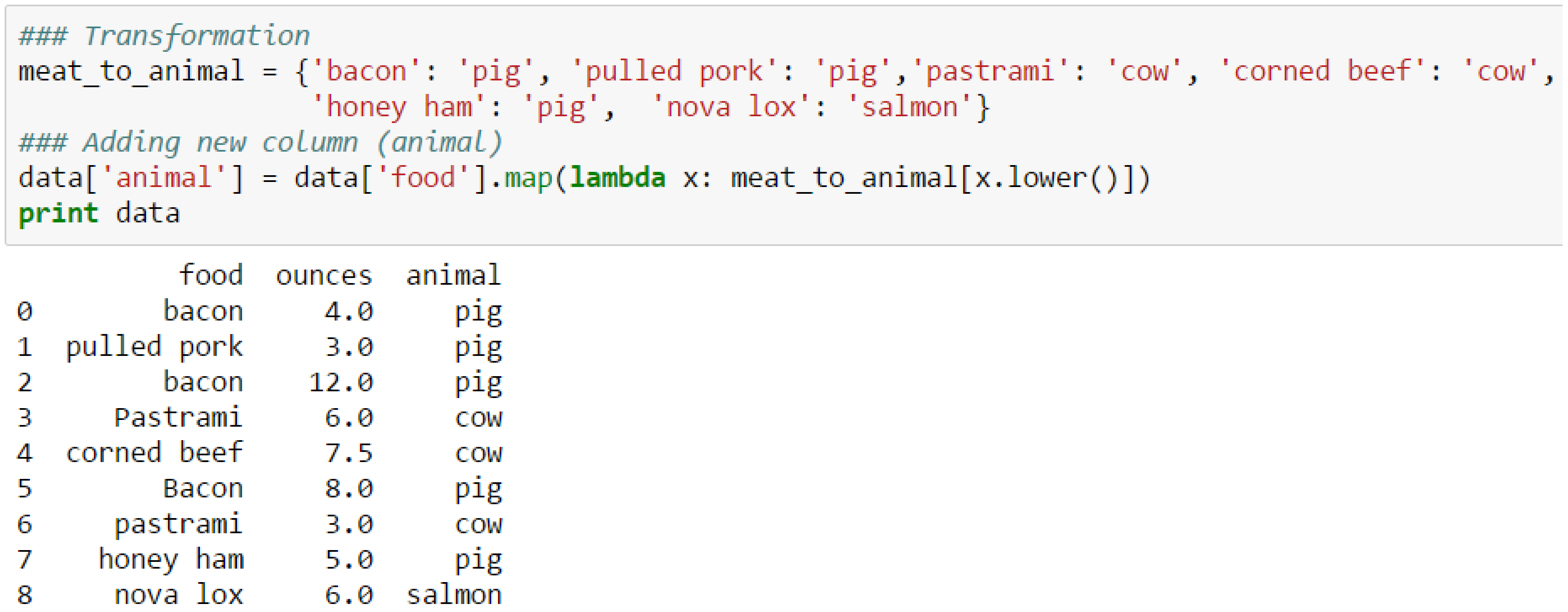
```
import pandas as pd
from pandas import *
data = DataFrame({'food': ['bacon','pulled pork','bacon','Pastrami','corned beef',
                           'Bacon','pastrami','honey ham','nova lox'],
                  'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
print data
```

```
          food  ounces
0        bacon     4.0
1  pulled pork     3.0
2        bacon    12.0
3     Pastrami     6.0
4  corned beef     7.5
5        Bacon     8.0
6     pastrami     3.0
7    honey ham     5.0
8     nova lox     6.0
```

See Data_wrangling9.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Mapping values (Python)



```
### Transformation
meat_to_animal = {'bacon': 'pig', 'pulled pork': 'pig','pastrami': 'cow', 'corned beef': 'cow',
                  'honey ham': 'pig',  'nova lox': 'salmon'}
### Adding new column (animal)
data['animal'] = data['food'].map(lambda x: meat_to_animal[x.lower()])
print data
```

```
          food  ounces  animal
0        bacon     4.0     pig
1  pulled pork     3.0     pig
2        bacon    12.0     pig
3     Pastrami     6.0     cow
4  corned beef     7.5     cow
5        Bacon     8.0     pig
6     pastrami     3.0     cow
7    honey ham     5.0     pig
8     nova lox     6.0  salmon
```

See Data_wrangling9.ipynb

Mapping values (R)

```
### Mapping values
### Defining some dictionary
dict<-data.frame(animal=c('pig','cow','salmon'),meat=c('bacon','beef','Nova lox'))
dict
### Creating a new data frame
data<-data.frame(meat=c('bacon', 'beef', 'bacon', 'Nova lox', 'bacon'))
data
### Matching meat with animal
data$animal <- dict[match(data$meat, key$meat), 'animal']
data
```

match() function deals with mapping values in R

```
> ### Mapping values
> ### Defining some dictionary
> dict<-data.frame(animal=c('pig','cow','salmon'),meat=c('bacon','beef','Nova lox'))
> dict
  animal     meat
1    pig    bacon
2    cow     beef
3 salmon Nova lox
> ### Creating a new data frame
> data<-data.frame(meat=c('bacon', 'beef', 'bacon', 'Nova lox', 'bacon'))
> data
      meat
1    bacon
2     beef
3    bacon
4 Nova lox
5    bacon
> ### Matching meat with animal
> data$animal <- dict[match(data$meat, key$meat), 'animal']
> data
      meat animal
1    bacon    pig
2     beef    cow
3    bacon    pig
4 Nova lox salmon
5    bacon    pig
```

See Data_wrangling8.R

QuantUniversity, LLC
www.quantuniversity.com

Replacing values (Python)

- Is commonly used to replace missing values with other values.

```
import numpy as np
import pandas
from pandas import *
data = Series([1., -999., 2., np.nan, -1000., 3.])
print data
```

```
0       1
1    -999
2       2
3     NaN
4   -1000
5       3
dtype: float64
```

```
print data.replace(np.nan,0)
```

```
0       1
1    -999
2       2
3       0
4   -1000
5       3
dtype: float64
```

```
print data.replace({np.nan:0})
```

```
0       1
1    -999
2       2
3       0
4   -1000
5       3
dtype: float64
```

```
import scipy as sc
from scipy import *
print data.replace({np.nan:sc.mean(data)})
```

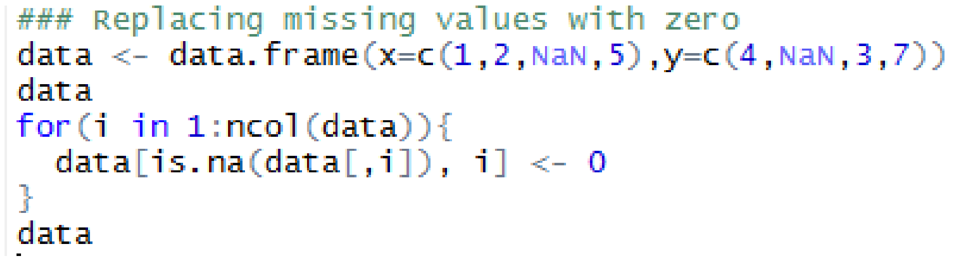
```
0       1.0
1    -999.0
2       2.0
3    -398.6
4   -1000.0
5       3.0
dtype: float64
```

See Data_wrangling10.ipynb

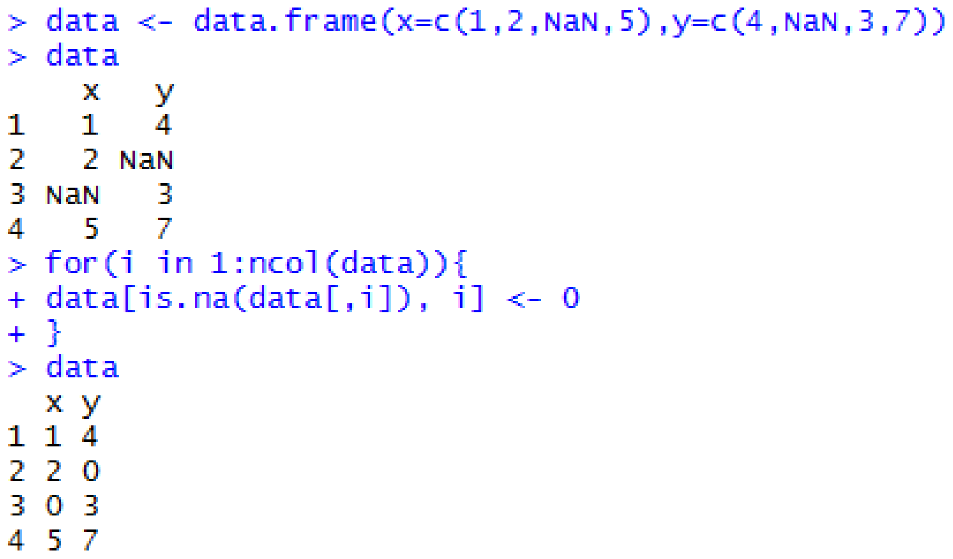
QuantUniversity, LLC
www.quantuniversity.com

Replacing values (R)

- Is commonly used to replace missing values with other values.

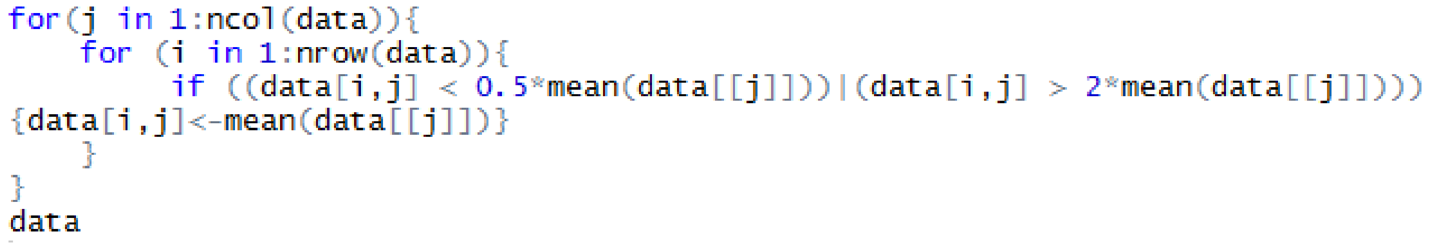


```
### Replacing missing values with zero
data <- data.frame(x=c(1,2,NaN,5),y=c(4,NaN,3,7))
data
for(i in 1:ncol(data)){
  data[is.na(data[,i]), i] <- 0
}
data
```

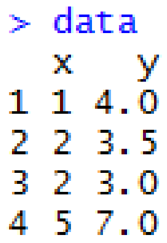


```
> data <- data.frame(x=c(1,2,NaN,5),y=c(4,NaN,3,7))
> data
    x   y
1   1   4
2   2 NaN
3 NaN   3
4   5   7
> for(i in 1:ncol(data)){
+ data[is.na(data[,i]), i] <- 0
+ }
> data
  x y
1 1 4
2 2 0
3 0 3
4 5 7
```

Replacing some specific values with mean



```
for(j in 1:ncol(data)){
    for (i in 1:nrow(data)){
         if ((data[i,j] < 0.5*mean(data[[j]]))|(data[i,j] > 2*mean(data[[j]])))
{data[i,j]<-mean(data[[j]])}
    }
}
data
```



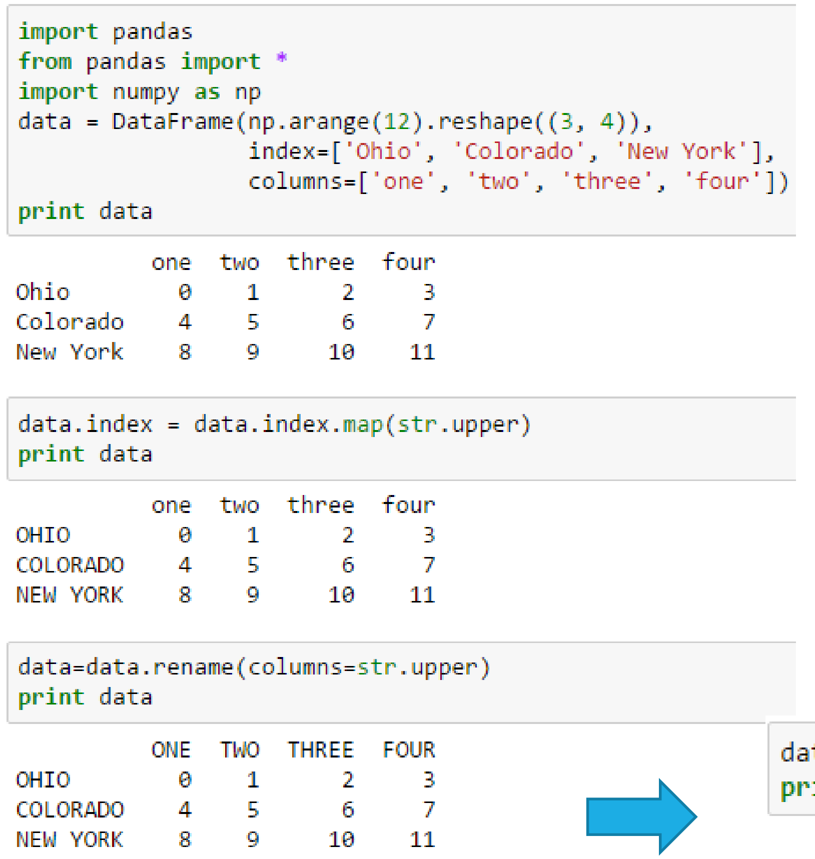
```
> data
  x   y
1 1 4.0
2 2 3.5
3 2 3.0
4 5 7.0
```

See Data_wrangling9.R

QuantUniversity, LLC
www.quantuniversity.com

Renaming axis indexes (Python)

- Labels axis can be changed or modified such as values in data frames.



```
import pandas
from pandas import *
import numpy as np
data = DataFrame(np.arange(12).reshape((3, 4)),
                 index=['Ohio', 'Colorado', 'New York'],
                 columns=['one', 'two', 'three', 'four'])
print data
```

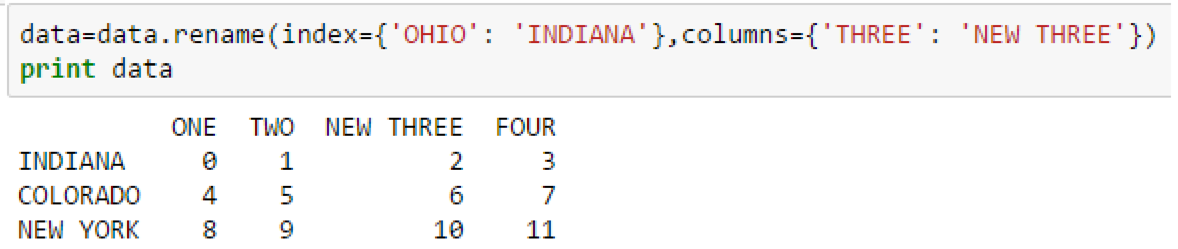
```
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
New York    8    9     10    11
```

```
data.index = data.index.map(str.upper)
print data
```

```
          one  two  three  four
OHIO        0    1      2     3
COLORADO    4    5      6     7
NEW YORK    8    9     10    11
```

```
data=data.rename(columns=str.upper)
print data
```

```
          ONE  TWO  THREE  FOUR
OHIO        0    1      2     3
COLORADO    4    5      6     7
NEW YORK    8    9     10    11
```



```
data=data.rename(index={'OHIO': 'INDIANA'},columns={'THREE': 'NEW THREE'})
print data
```

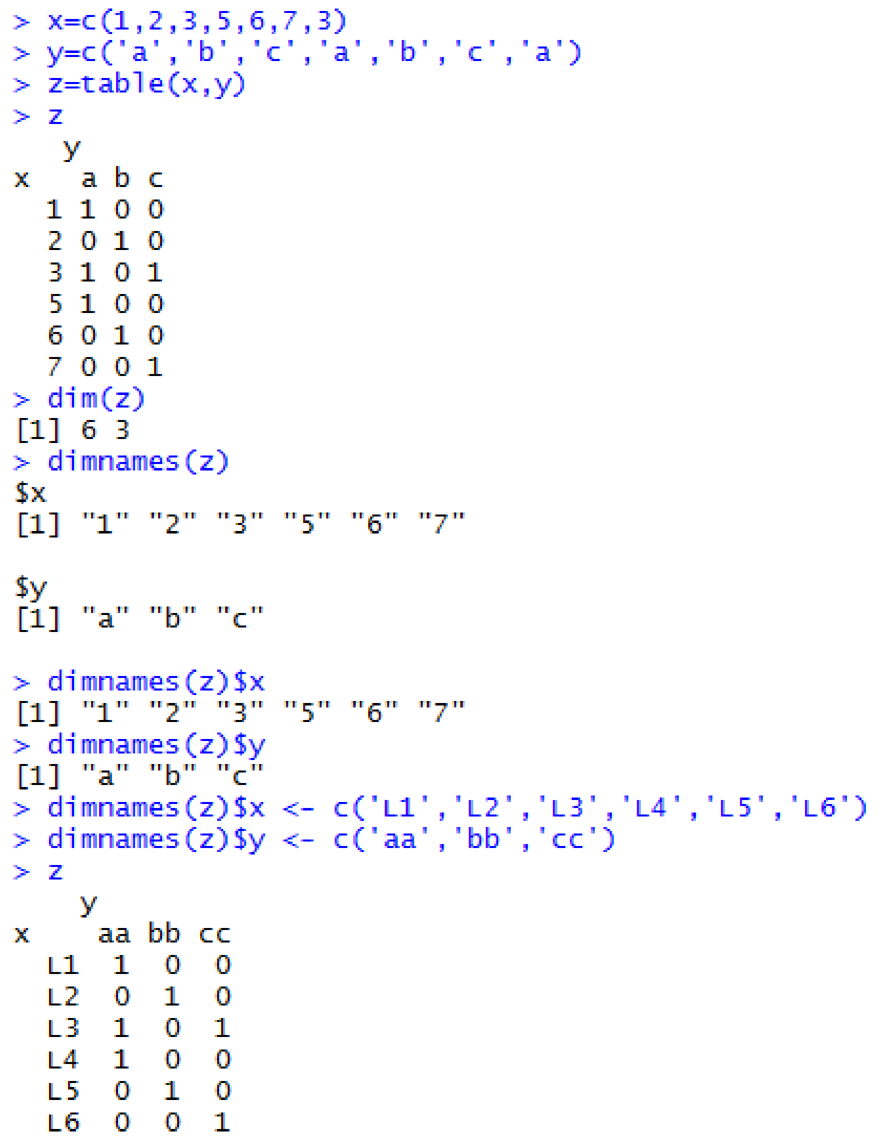
```
          ONE  TWO  NEW THREE  FOUR
INDIANA     0    1          2     3
COLORADO    4    5          6     7
NEW YORK    8    9         10    11
```

See Data_wrangling11.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Renaming axis indexes (R)

```
x=c(1,2,3,5,6,7,3)
y=c('a','b','c','a','b','c','a')
z=table(x,y)
z
dim(z)
dimnames(z)
dimnames(z)$x
dimnames(z)$y
dimnames(z)$x <- c('L1','L2','L3','L4','L5','L6')
dimnames(z)$y <- c('aa','bb','cc')
z
```



```
> x=c(1,2,3,5,6,7,3)
> y=c('a','b','c','a','b','c','a')
> z=table(x,y)
> z
   y
x   a b c
  1 1 0 0
  2 0 1 0
  3 1 0 1
  5 1 0 0
  6 0 1 0
  7 0 0 1
> dim(z)
[1] 6 3
> dimnames(z)
$x
[1] "1" "2" "3" "5" "6" "7"

$y
[1] "a" "b" "c"

> dimnames(z)$x
[1] "1" "2" "3" "5" "6" "7"
> dimnames(z)$y
[1] "a" "b" "c"
> dimnames(z)$x <- c('L1','L2','L3','L4','L5','L6')
> dimnames(z)$y <- c('aa','bb','cc')
> z
    y
x    aa bb cc
  L1  1  0  0
  L2  0  1  0
  L3  1  0  1
  L4  1  0  0
  L5  0  1  0
  L6  0  0  1
```

See Data_wrangling10.R

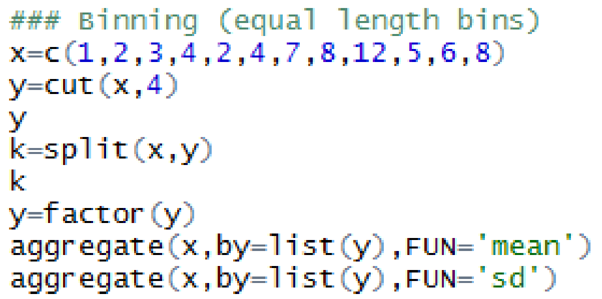
Other data transformations

- ✓ Binning
- ✓ Subgroups
- ✓ Normalization
- ✓ Dummy variables

Data transformation

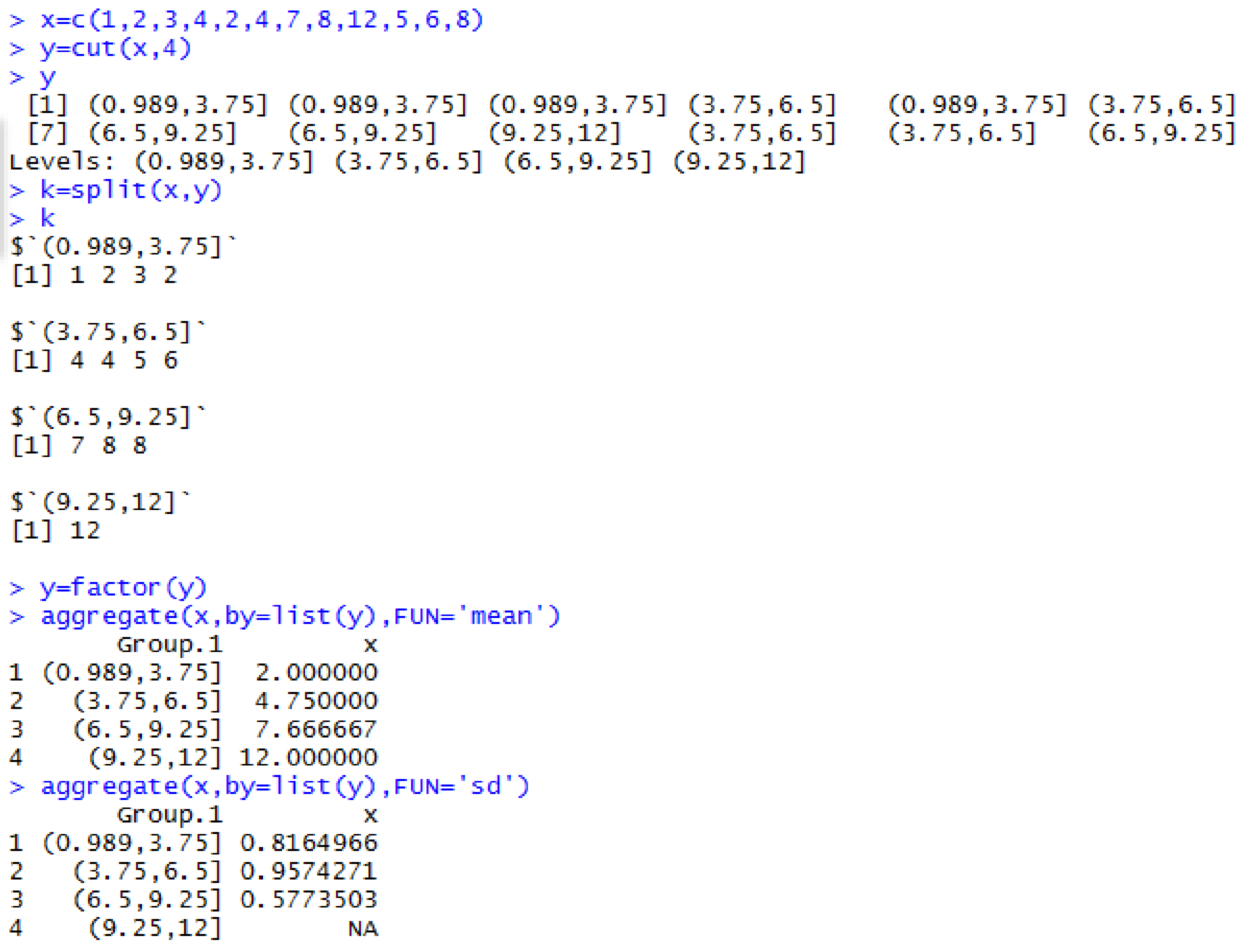
- Popular data transformation techniques include:
  - **Smoothing:** Remove noise from data by doing binning, regression or clustering.
  - **Aggregation:** Where summary operations may applied to the data. For instance daily sales may be aggregated to compute monthly and annual sales amounts.
  - **Generalization:** Where low level data are replaced by higher level data through hierarchies. For instance, categorical street feature may be changed to city or country.

Binning (R)



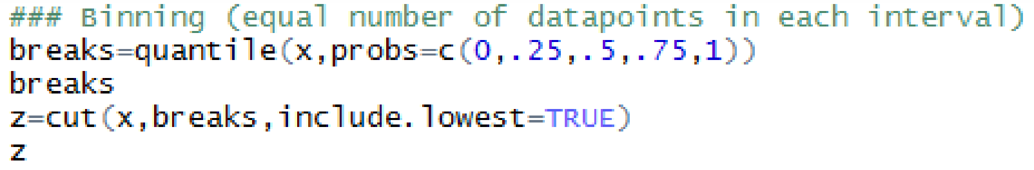
```
### Binning (equal length bins)
x=c(1,2,3,4,2,4,7,8,12,5,6,8)
y=cut(x,4)
y
k=split(x,y)
k
y=factor(y)
aggregate(x,by=list(y),FUN='mean')
aggregate(x,by=list(y),FUN='sd')
```

Binning will allows us to work with statistics of each group instead of each individual data.

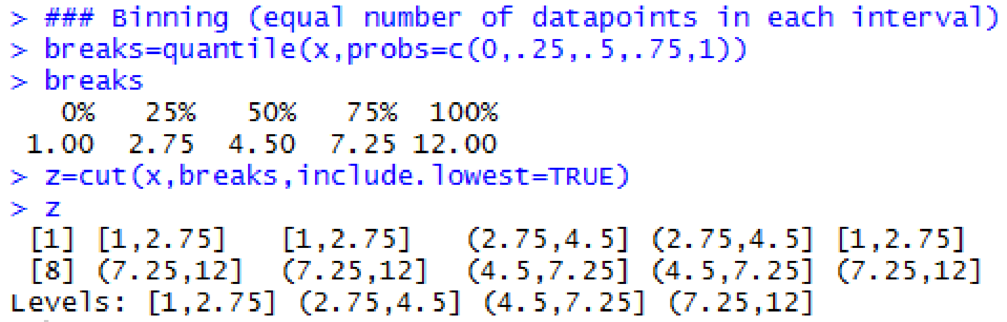


```
> x=c(1,2,3,4,2,4,7,8,12,5,6,8)
> y=cut(x,4)
> y
 [1] (0.989,3.75] (0.989,3.75] (0.989,3.75] (3.75,6.5]   (0.989,3.75] (3.75,6.5]
 [7] (6.5,9.25]   (6.5,9.25]   (9.25,12]    (3.75,6.5]   (3.75,6.5]   (6.5,9.25]
Levels: (0.989,3.75] (3.75,6.5] (6.5,9.25] (9.25,12]
> k=split(x,y)
> k
$`(0.989,3.75]`
[1] 1 2 3 2

$`(3.75,6.5]`
[1] 4 4 5 6

$`(6.5,9.25]`
[1] 7 8 8

$`(9.25,12]`
[1] 12

> y=factor(y)
> aggregate(x,by=list(y),FUN='mean')
       Group.1         x
1 (0.989,3.75]  2.000000
2   (3.75,6.5]  4.750000
3   (6.5,9.25]  7.666667
4    (9.25,12] 12.000000
> aggregate(x,by=list(y),FUN='sd')
       Group.1         x
1 (0.989,3.75] 0.8164966
2   (3.75,6.5] 0.9574271
3   (6.5,9.25] 0.5773503
4    (9.25,12]        NA
```

In R, you can bin data either in equal length or equal number of points in each interval.



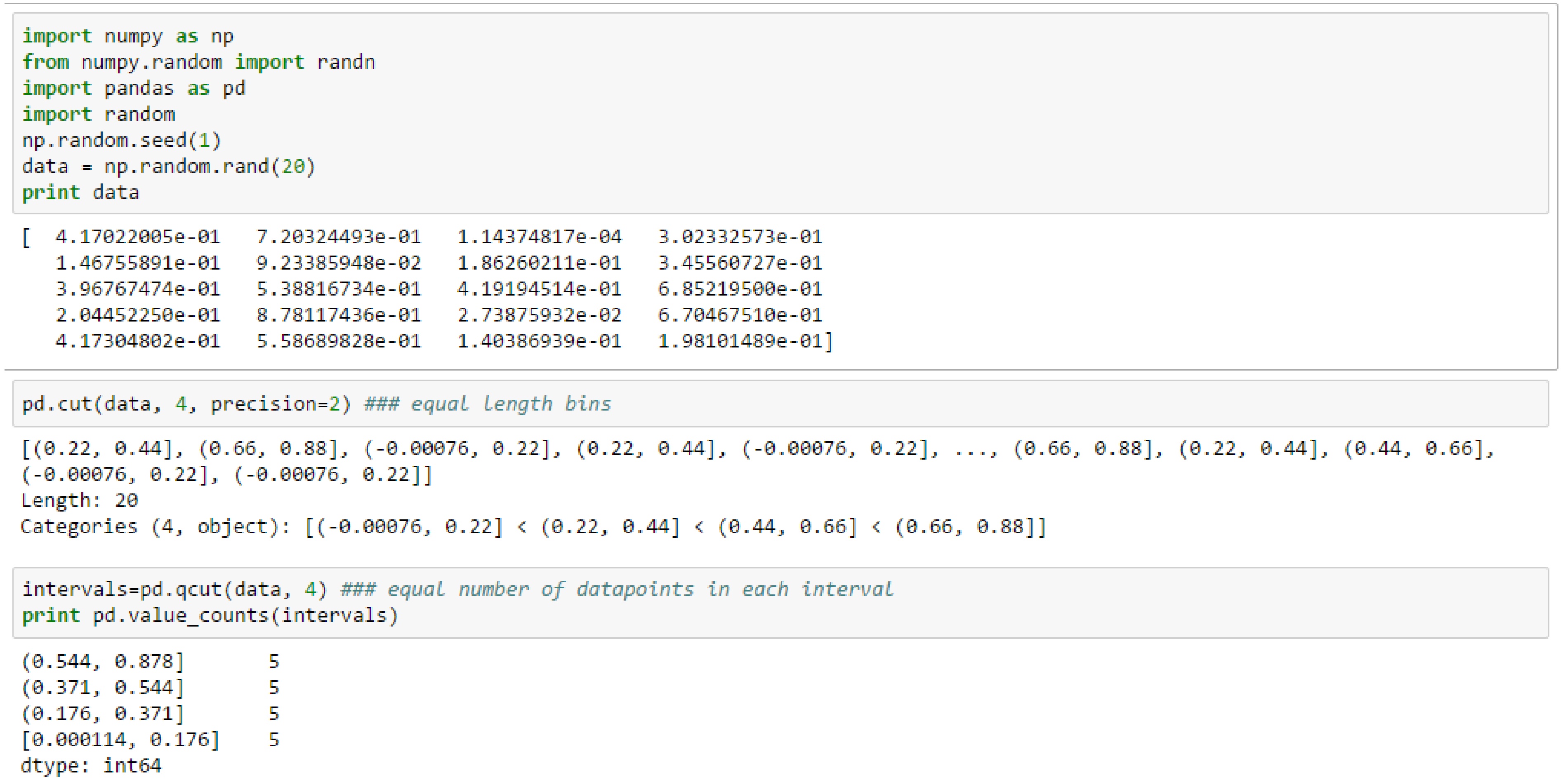
```
### Binning (equal number of datapoints in each interval)
breaks=quantile(x,probs=c(0,.25,.5,.75,1))
breaks
z=cut(x,breaks,include.lowest=TRUE)
z
```



```
> ### Binning (equal number of datapoints in each interval)
> breaks=quantile(x,probs=c(0,.25,.5,.75,1))
> breaks
   0%   25%   50%   75%  100% 
 1.00  2.75  4.50  7.25 12.00 
> z=cut(x,breaks,include.lowest=TRUE)
> z
 [1] [1,2.75]   [1,2.75]   (2.75,4.5] (2.75,4.5] [1,2.75]
 [8] (7.25,12]  (7.25,12]  (4.5,7.25] (4.5,7.25] (7.25,12]
Levels: [1,2.75] (2.75,4.5] (4.5,7.25] (7.25,12]
```

See Data_preprocessing4.R

Binning (Python)



```
import numpy as np
from numpy.random import randn
import pandas as pd
import random
np.random.seed(1)
data = np.random.rand(20)
print data
```

```
[  4.17022005e-01   7.20324493e-01   1.14374817e-04   3.02332573e-01
   1.46755891e-01   9.23385948e-02   1.86260211e-01   3.45560727e-01
   3.96767474e-01   5.38816734e-01   4.19194514e-01   6.85219500e-01
   2.04452250e-01   8.78117436e-01   2.73875932e-02   6.70467510e-01
   4.17304802e-01   5.58689828e-01   1.40386939e-01   1.98101489e-01]
```

```
pd.cut(data, 4, precision=2) ### equal length bins
```

```
[(0.22, 0.44], (0.66, 0.88], (-0.00076, 0.22], (0.22, 0.44], (-0.00076, 0.22], ..., (0.66, 0.88], (0.22, 0.44], (0.44, 0.66],
(-0.00076, 0.22], (-0.00076, 0.22]]
Length: 20
Categories (4, object): [(-0.00076, 0.22] < (0.22, 0.44] < (0.44, 0.66] < (0.66, 0.88]]
```

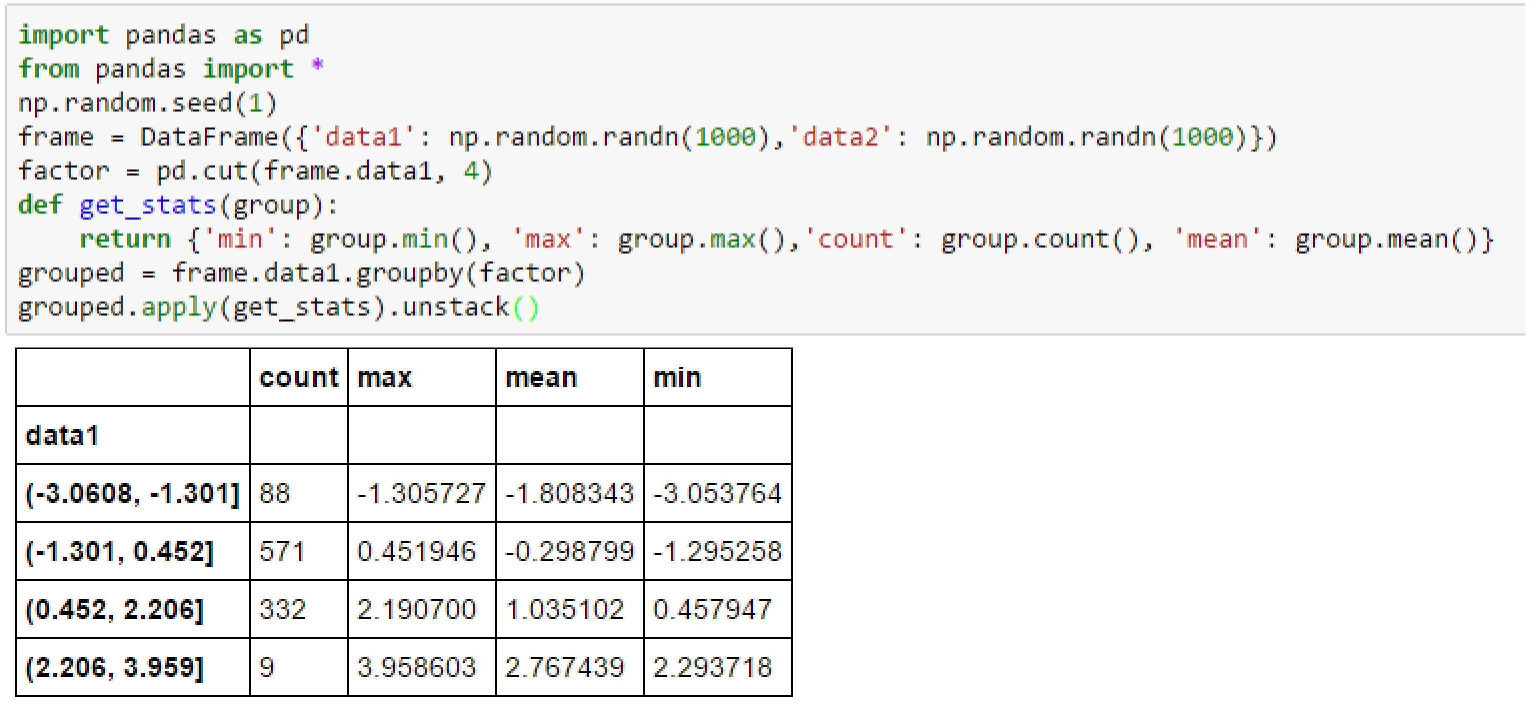
```
intervals=pd.qcut(data, 4) ### equal number of datapoints in each interval
print pd.value_counts(intervals)
```

```
(0.544, 0.878]       5
(0.371, 0.544]       5
(0.176, 0.371]       5
[0.000114, 0.176]    5
dtype: int64
```

See Data_preprocessing4.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Binning (Python)



```
import pandas as pd
from pandas import *
np.random.seed(1)
frame = DataFrame({'data1': np.random.randn(1000),'data2': np.random.randn(1000)})
factor = pd.cut(frame.data1, 4)
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),'count': group.count(), 'mean': group.mean()}
grouped = frame.data1.groupby(factor)
grouped.apply(get_stats).unstack()
```

	count	max	mean	min
data1				
(-3.0608, -1.301]	88	-1.305727	-1.808343	-3.053764
(-1.301, 0.452]	571	0.451946	-0.298799	-1.295258
(0.452, 2.206]	332	2.190700	1.035102	0.457947
(2.206, 3.959]	9	3.958603	2.767439	2.293718

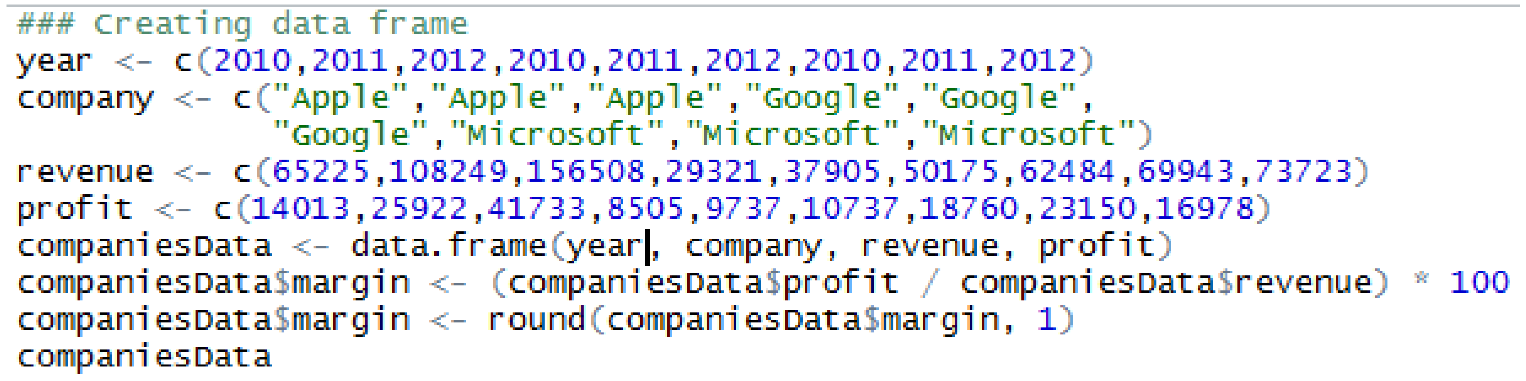
Binning will allows us to work with statistics of each group instead of each individual data.

See Data_preprocessing4.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Working with subgroups (R)

- Package ‘plyr’ is used to spilt the dataset by multiple factors and applying function:

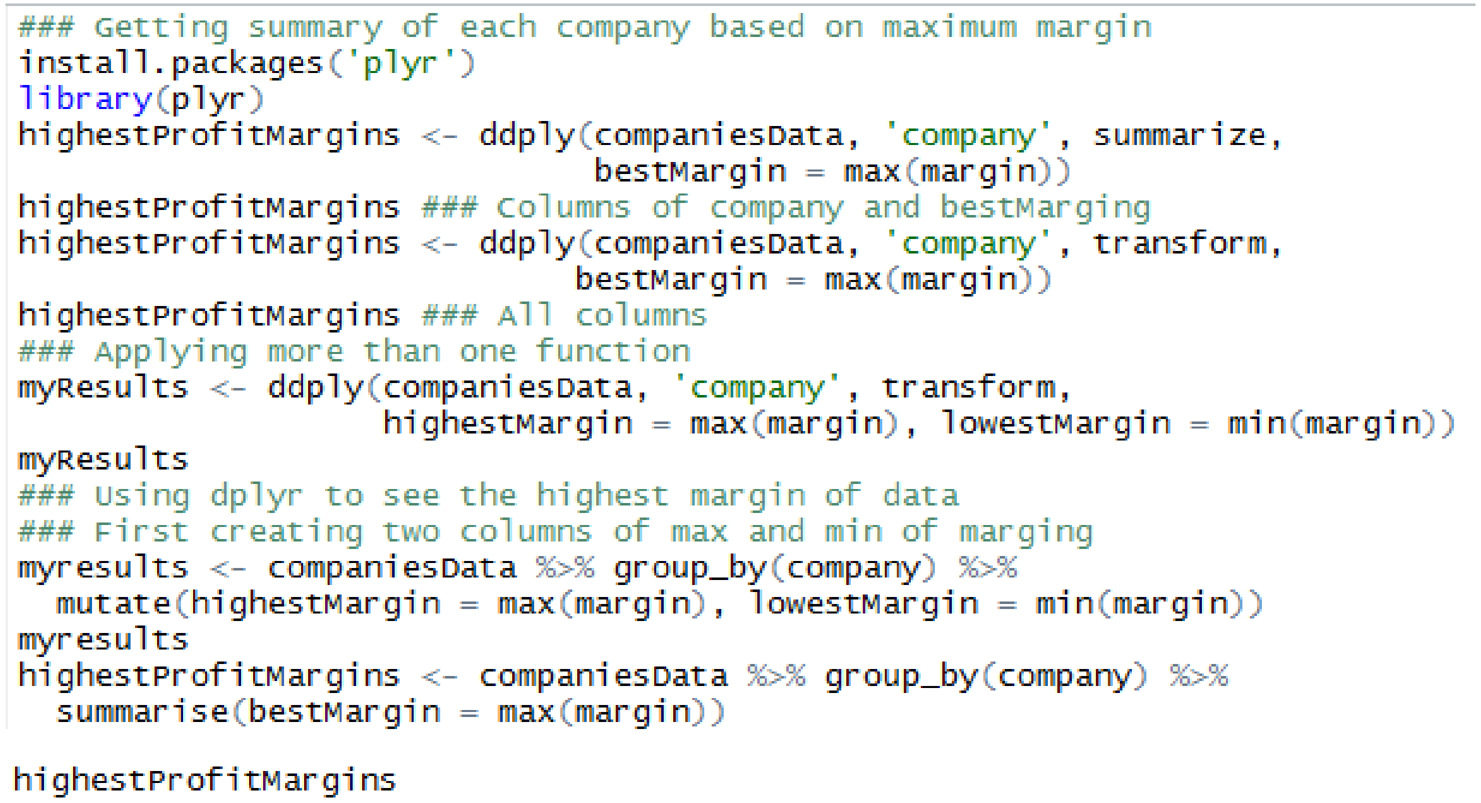


```
### Creating data frame
year <- c(2010,2011,2012,2010,2011,2012,2010,2011,2012)
company <- c("Apple","Apple","Apple","Google","Google",
             "Google","Microsoft","Microsoft","Microsoft")
revenue <- c(65225,108249,156508,29321,37905,50175,62484,69943,73723)
profit <- c(14013,25922,41733,8505,9737,10737,18760,23150,16978)
companiesData <- data.frame(year, company, revenue, profit)
companiesData$margin <- (companiesData$profit / companiesData$revenue) * 100
companiesData$margin <- round(companiesData$margin, 1)
companiesData
```

See Data_preprocessing5.R

QuantUniversity, LLC
www.quantuniversity.com

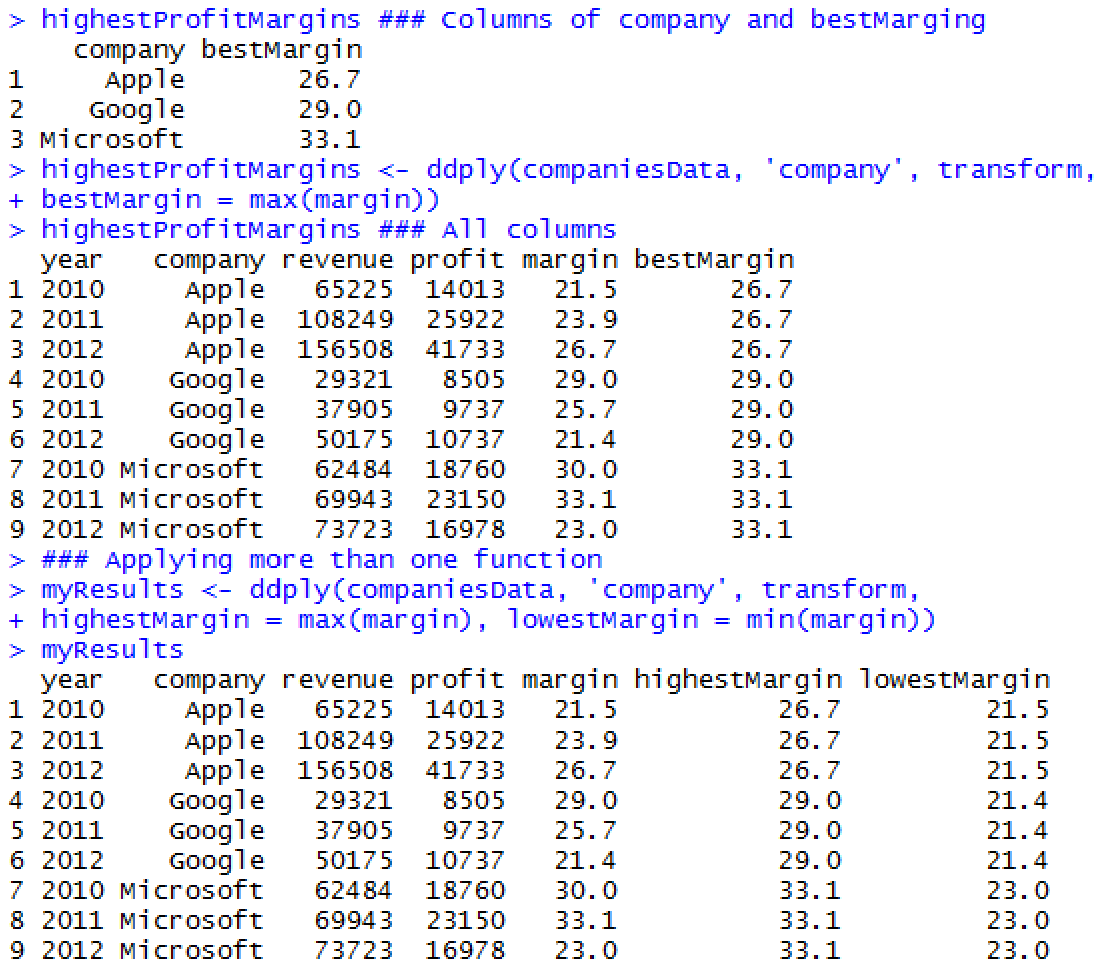
Working with subgroups (R)



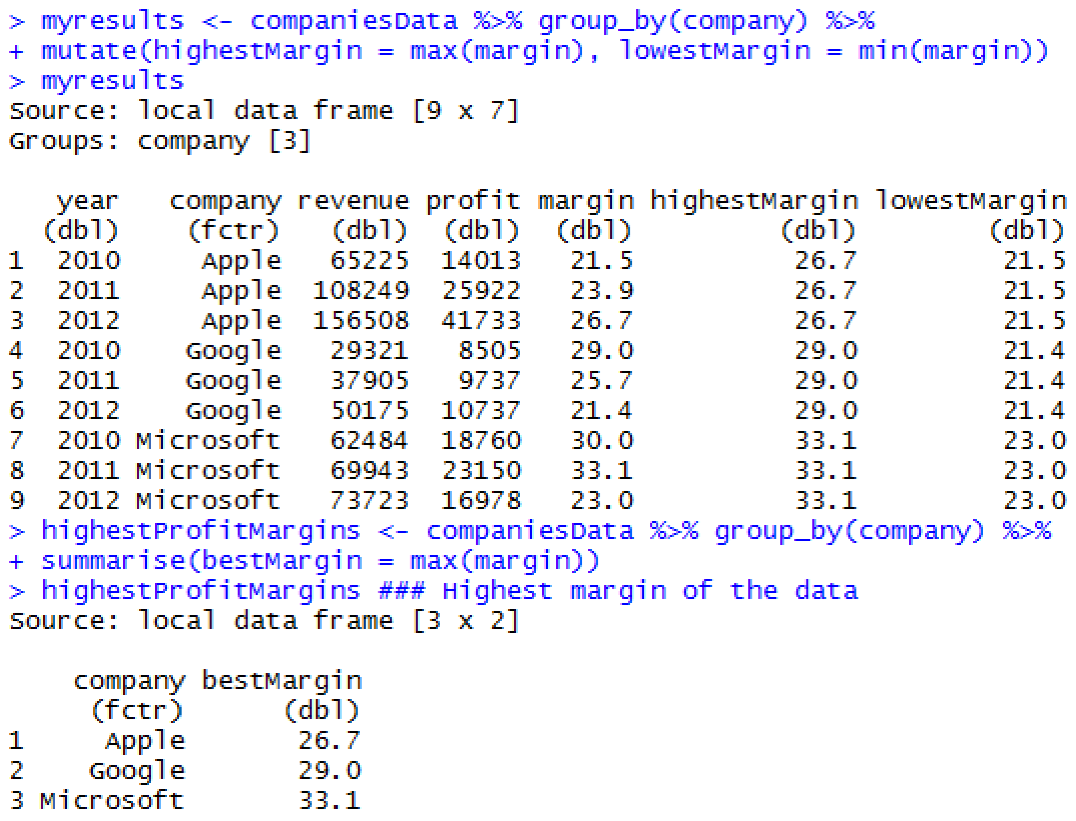
```
### Getting summary of each company based on maximum margin
install.packages('plyr')
library(plyr)
highestProfitMargins <- ddply(companiesData, 'company', summarize,
                              bestMargin = max(margin))
highestProfitMargins ### Columns of company and bestMarging
highestProfitMargins <- ddply(companiesData, 'company', transform,
                             bestMargin = max(margin))
highestProfitMargins ### All columns
### Applying more than one function
myResults <- ddply(companiesData, 'company', transform,
                   highestMargin = max(margin), lowestMargin = min(margin))
myResults
### Using dplyr to see the highest margin of data
### First creating two columns of max and min of marging
myresults <- companiesData %>% group_by(company) %>%
  mutate(highestMargin = max(margin), lowestMargin = min(margin))
myresults
highestProfitMargins <- companiesData %>% group_by(company) %>%
  summarise(bestMargin = max(margin))

highestProfitMargins
```

See Data_preprocessing5.R

Working with subgroups (R)



```
> highestProfitMargins ### Columns of company and bestMarging
    company bestMargin
1     Apple       26.7
2    Google       29.0
3 Microsoft       33.1
> highestProfitMargins <- ddply(companiesData, 'company', transform,
+ bestMargin = max(margin))
> highestProfitMargins ### All columns
  year   company revenue profit margin bestMargin
1 2010     Apple   65225  14013   21.5       26.7
2 2011     Apple  108249  25922   23.9       26.7
3 2012     Apple  156508  41733   26.7       26.7
4 2010    Google   29321   8505   29.0       29.0
5 2011    Google   37905   9737   25.7       29.0
6 2012    Google   50175  10737   21.4       29.0
7 2010 Microsoft   62484  18760   30.0       33.1
8 2011 Microsoft   69943  23150   33.1       33.1
9 2012 Microsoft   73723  16978   23.0       33.1
> ### Applying more than one function
> myResults <- ddply(companiesData, 'company', transform,
+ highestMargin = max(margin), lowestMargin = min(margin))
> myResults
  year   company revenue profit margin highestMargin lowestMargin
1 2010     Apple   65225  14013   21.5          26.7         21.5
2 2011     Apple  108249  25922   23.9          26.7         21.5
3 2012     Apple  156508  41733   26.7          26.7         21.5
4 2010    Google   29321   8505   29.0          29.0         21.4
5 2011    Google   37905   9737   25.7          29.0         21.4
6 2012    Google   50175  10737   21.4          29.0         21.4
7 2010 Microsoft   62484  18760   30.0          33.1         23.0
8 2011 Microsoft   69943  23150   33.1          33.1         23.0
9 2012 Microsoft   73723  16978   23.0          33.1         23.0
```



```
> myresults <- companiesData %>% group_by(company) %>%
+ mutate(highestMargin = max(margin), lowestMargin = min(margin))
> myresults
Source: local data frame [9 x 7]
Groups: company [3]

   year   company revenue profit margin highestMargin lowestMargin
  (dbl)    (fctr)   (dbl)  (dbl)  (dbl)         (dbl)        (dbl)
1  2010     Apple   65225  14013   21.5          26.7         21.5
2  2011     Apple  108249  25922   23.9          26.7         21.5
3  2012     Apple  156508  41733   26.7          26.7         21.5
4  2010    Google   29321   8505   29.0          29.0         21.4
5  2011    Google   37905   9737   25.7          29.0         21.4
6  2012    Google   50175  10737   21.4          29.0         21.4
7  2010 Microsoft   62484  18760   30.0          33.1         23.0
8  2011 Microsoft   69943  23150   33.1          33.1         23.0
9  2012 Microsoft   73723  16978   23.0          33.1         23.0
> highestProfitMargins <- companiesData %>% group_by(company) %>%
+ summarise(bestMargin = max(margin))
> highestProfitMargins ### Highest margin of the data
Source: local data frame [3 x 2]

    company bestMargin
     (fctr)      (dbl)
1     Apple       26.7
2    Google       29.0
3 Microsoft       33.1
```

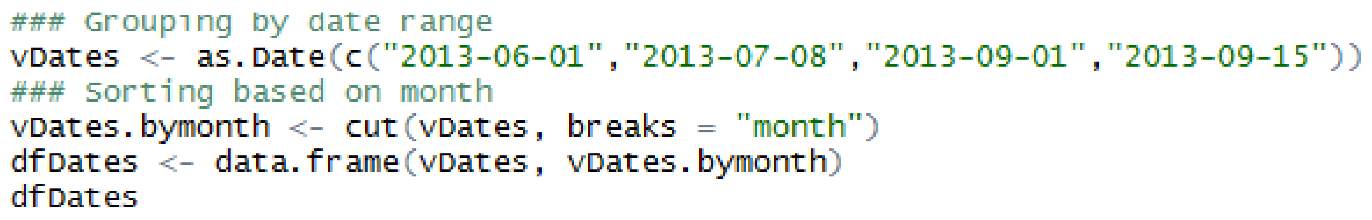
See Data_preprocessing5.R

QuantUniversity, LLC

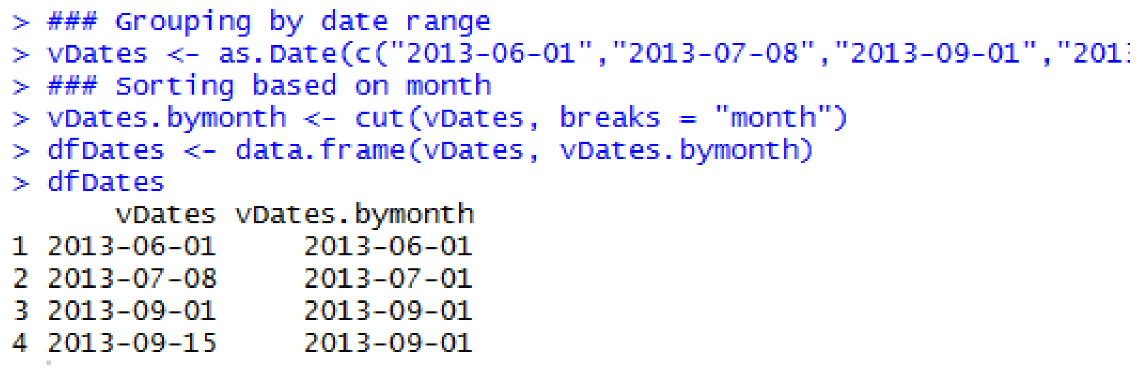
www.quantuniversity.com

Working with subgroups (R)

- Grouping by date range:



```
### Grouping by date range
vDates <- as.Date(c("2013-06-01","2013-07-08","2013-09-01","2013-09-15"))
### Sorting based on month
vDates.bymonth <- cut(vDates, breaks = "month")
dfDates <- data.frame(vDates, vDates.bymonth)
dfDates
```



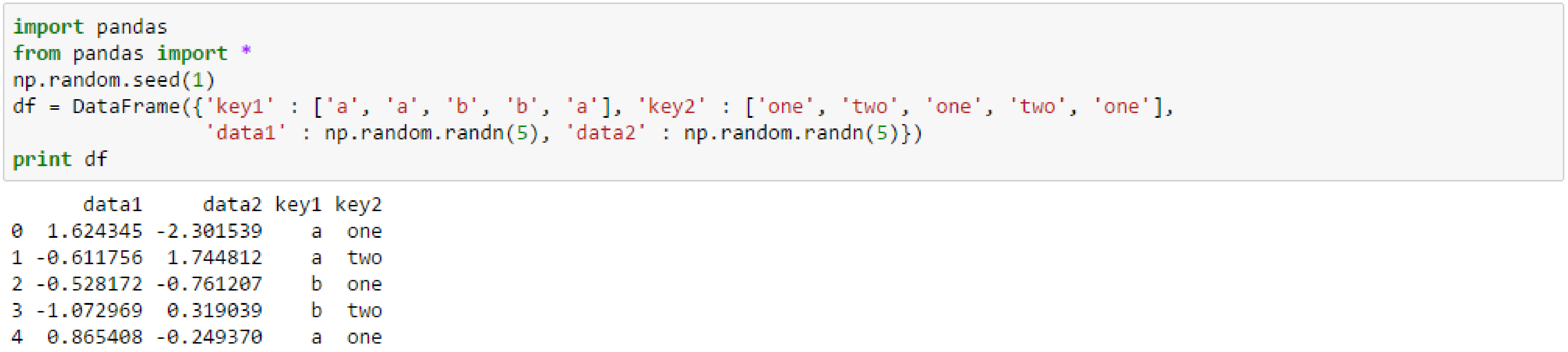
```
> ### Grouping by date range
> vDates <- as.Date(c("2013-06-01","2013-07-08","2013-09-01","201
> ### Sorting based on month
> vDates.bymonth <- cut(vDates, breaks = "month")
> dfDates <- data.frame(vDates, vDates.bymonth)
> dfDates
      vDates vDates.bymonth
1 2013-06-01     2013-06-01
2 2013-07-08     2013-07-01
3 2013-09-01     2013-09-01
4 2013-09-15     2013-09-01
```

See Data_preprocessing5.R

QuantUniversity, LLC
www.quantuniversity.com

Working with subgroups (Python)

- In python subgroups can be extracted from data frame by using ‘groupby’ function. Also you may do some functions such as mean on numerical columns for different subgroups of data.



```
import pandas
from pandas import *
np.random.seed(1)
df = DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'], 'key2' : ['one', 'two', 'one', 'two', 'one'],
                'data1' : np.random.randn(5), 'data2' : np.random.randn(5)})
print df
```

```
      data1     data2 key1 key2
0  1.624345 -2.301539    a  one
1 -0.611756  1.744812    a  two
2 -0.528172 -0.761207    b  one
3 -1.072969  0.319039    b  two
4  0.865408 -0.249370    a  one
```

See Data_preprocessing5.ipynb

Working with subgroups (Python)

```
### Means of data1 for sub-groups of key1
means = df['data1'].groupby([df['key1']]).mean()
print means
```

```
key1
a    0.625999
b   -0.800570
Name: data1, dtype: float64
```

```
### Means of data1 for sub-groups of key1 and key2
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
print means
```

```
key1  key2
a     one     1.244876
      two    -0.611756
b     one    -0.528172
      two    -1.072969
Name: data1, dtype: float64
```

```
### Means of data1 and data2 for sub-groups of key1
means=df.groupby('key1').mean()
print means
```

```
         data1     data2
key1
a     0.625999 -0.268699
b    -0.800570 -0.221084
```

See Data_preprocessing5.ipynb

```
### Quantile of data1 and data2 for sub-groups of key1
quantile=df.groupby('key1').quantile(0.9).add_prefix('quantile_')
print quantile
```

```
      quantile_data1  quantile_data2
key1
a           1.472558        1.345975
b          -0.582651        0.211014
```

```
### Mean of data1 and data2 sub-groups of key1 and key2
means=df.groupby(['key1', 'key2']).mean().add_prefix('mean_')
print means
```

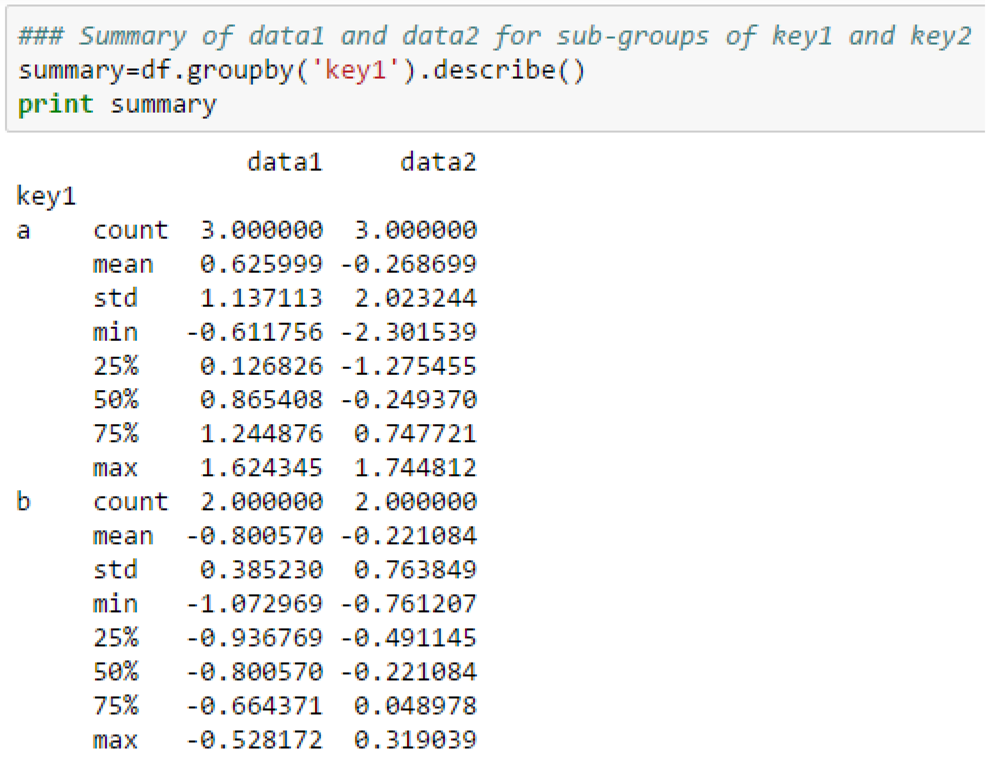
```
           mean_data1  mean_data2
key1 key2
a    one     1.244876   -1.275455
     two    -0.611756    1.744812
b    one    -0.528172   -0.761207
     two    -1.072969    0.319039
```

```
### Number of data1 and data2 pairs for sub-groups of key1 and key2
size=df.groupby(['key1', 'key2']).size()
print size
```

```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

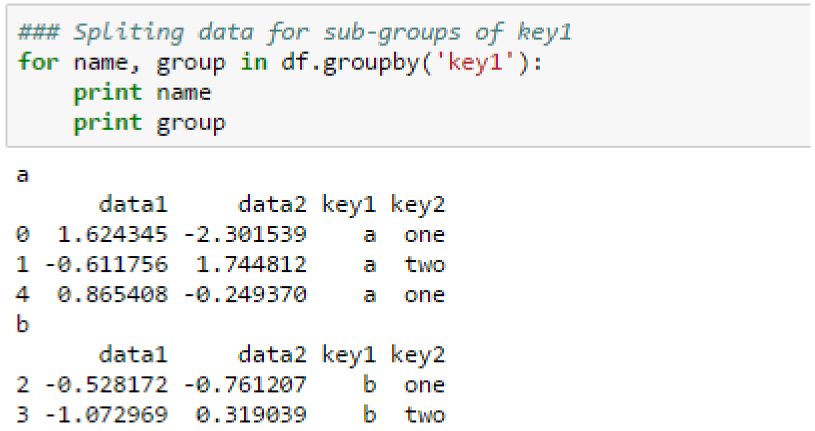
QuantUniversity, LLC
www.quantuniversity.com

Working with subgroups (Python)



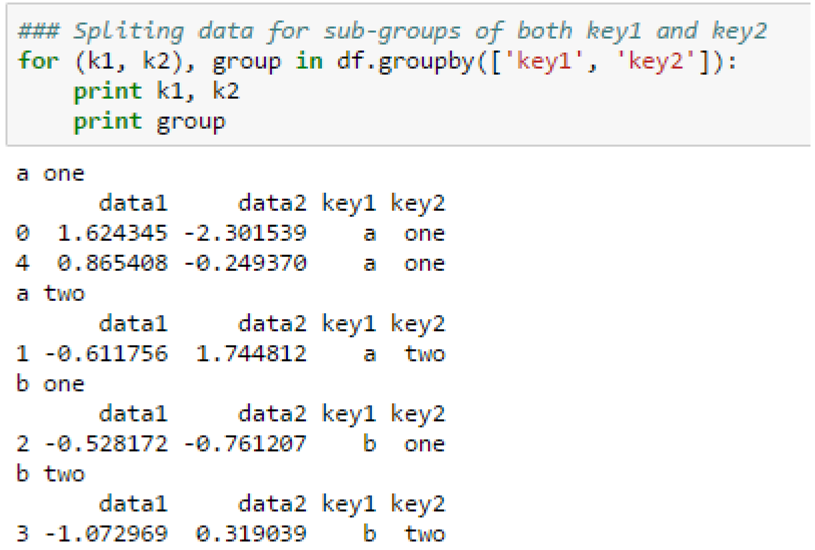
```
### Summary of data1 and data2 for sub-groups of key1 and key2
summary=df.groupby('key1').describe()
print summary
```

```
                data1     data2
key1
a    count   3.000000  3.000000
     mean    0.625999 -0.268699
     std     1.137113  2.023244
     min    -0.611756 -2.301539
     25%     0.126826 -1.275455
     50%     0.865408 -0.249370
     75%     1.244876  0.747721
     max     1.624345  1.744812
b    count   2.000000  2.000000
     mean   -0.800570 -0.221084
     std     0.385230  0.763849
     min    -1.072969 -0.761207
     25%    -0.936769 -0.491145
     50%    -0.800570 -0.221084
     75%    -0.664371  0.048978
     max    -0.528172  0.319039
```



```
### Spliting data for sub-groups of key1
for name, group in df.groupby('key1'):
    print name
    print group
```

```
a
      data1     data2 key1 key2
0  1.624345 -2.301539    a  one
1 -0.611756  1.744812    a  two
4  0.865408 -0.249370    a  one
b
      data1     data2 key1 key2
2 -0.528172 -0.761207    b  one
3 -1.072969  0.319039    b  two
```



```
### Spliting data for sub-groups of both key1 and key2
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print k1, k2
    print group
```

```
a one
      data1     data2 key1 key2
0  1.624345 -2.301539    a  one
4  0.865408 -0.249370    a  one
a two
      data1     data2 key1 key2
1 -0.611756  1.744812    a  two
b one
      data1     data2 key1 key2
2 -0.528172 -0.761207    b  one
b two
      data1     data2 key1 key2
3 -1.072969  0.319039    b  two
```

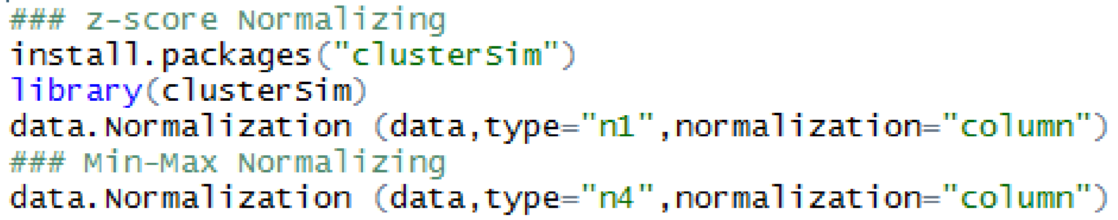
See Data_preprocessing5.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Data transformation

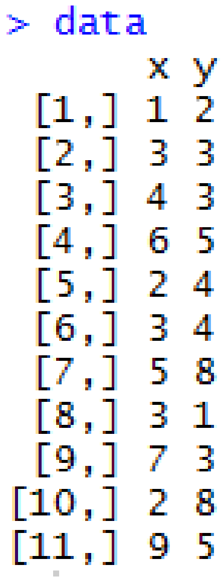
- **Normalization:** Where attribute data are scaled to a small range such as [0,1]
- **Min-Max normalization:** Each value in dataset like x_i will be changed to $\frac{x_i - Min(x)}{Max(x) - Min(x)}$
- **Z-score normalization:** Each value in a sample dataset (x_i) having specific mean and standard deviation will be changed to $\frac{x_i - Mean}{Standard\ Deviation}$
- **Decimal scaling normalization:** Each value in sample dataset is replaced by $\frac{x_i}{10^j}$ Where the Maximum absolute value of new data point is less than 1.

Normalization (R)



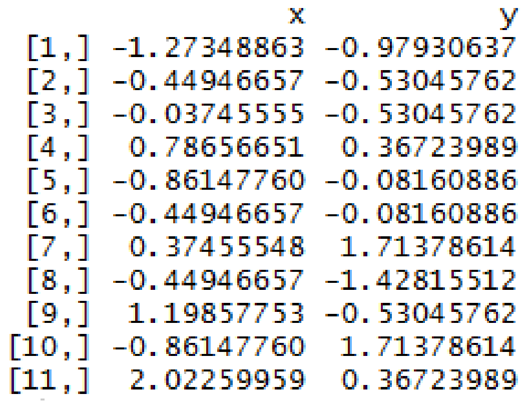
```
### z-score Normalizing
install.packages("clusterSim")
library(clusterSim)
data.Normalization (data,type="n1",normalization="column")
### Min-Max Normalizing
data.Normalization (data,type="n4",normalization="column")
```

In particular, clusterSim() package deals with normalization in R



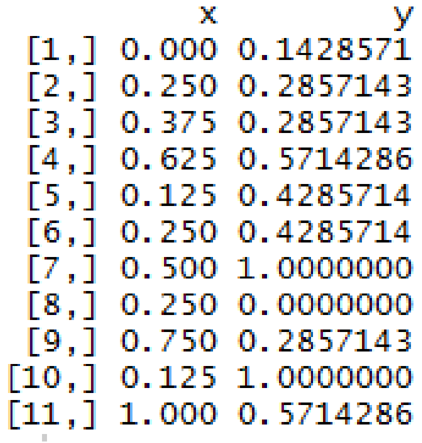
```
> data
      x y
 [1,] 1 2
 [2,] 3 3
 [3,] 4 3
 [4,] 6 5
 [5,] 2 4
 [6,] 3 4
 [7,] 5 8
 [8,] 3 1
 [9,] 7 3
[10,] 2 8
[11,] 9 5
```

Z- score



```
                x           y
 [1,] -1.27348863 -0.97930637
 [2,] -0.44946657 -0.53045762
 [3,] -0.03745555 -0.53045762
 [4,]  0.78656651  0.36723989
 [5,] -0.86147760 -0.08160886
 [6,] -0.44946657 -0.08160886
 [7,]  0.37455548  1.71378614
 [8,] -0.44946657 -1.42815512
 [9,]  1.19857753 -0.53045762
[10,] -0.86147760  1.71378614
[11,]  2.02259959  0.36723989
```

MinMax

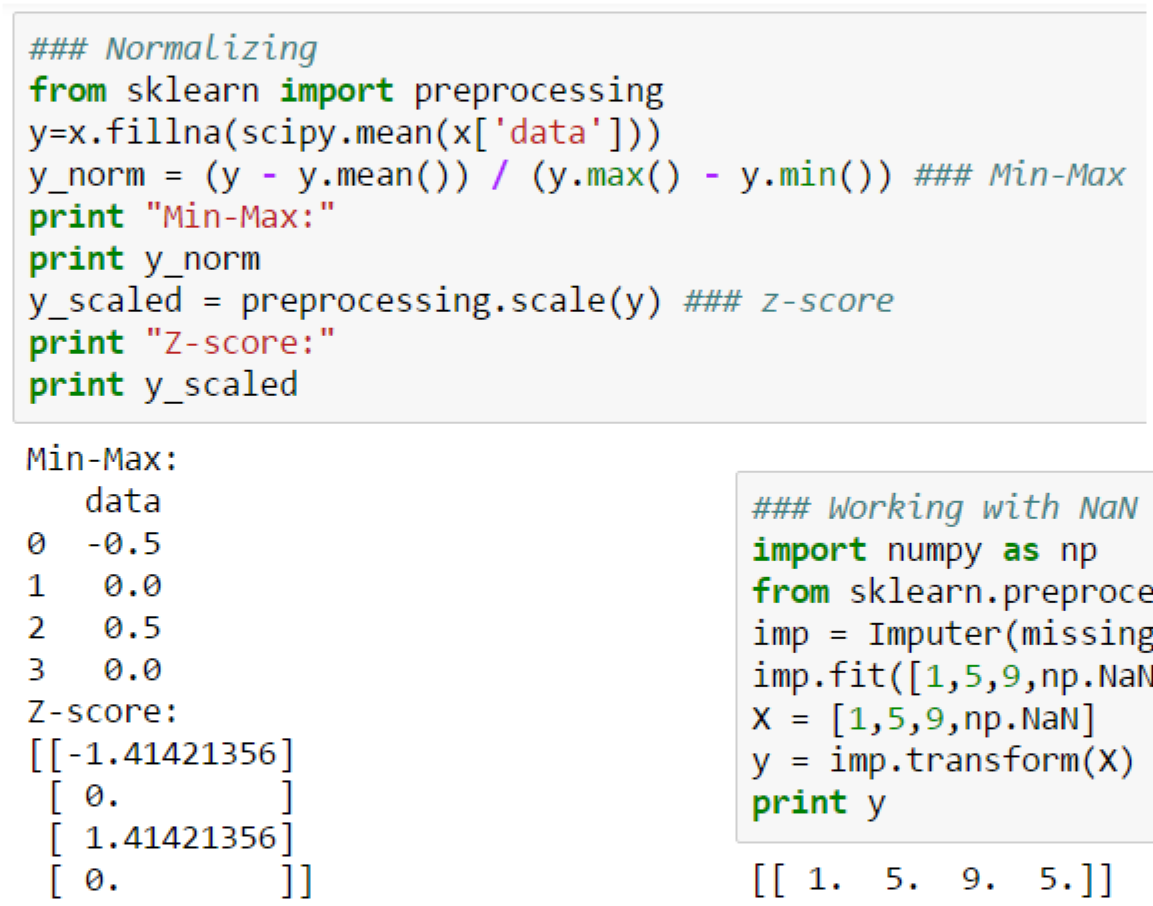


```
          x         y
 [1,] 0.000 0.1428571
 [2,] 0.250 0.2857143
 [3,] 0.375 0.2857143
 [4,] 0.625 0.5714286
 [5,] 0.125 0.4285714
 [6,] 0.250 0.4285714
 [7,] 0.500 1.0000000
 [8,] 0.250 0.0000000
 [9,] 0.750 0.2857143
[10,] 0.125 1.0000000
[11,] 1.000 0.5714286
```

See Data_preprocessing1.R

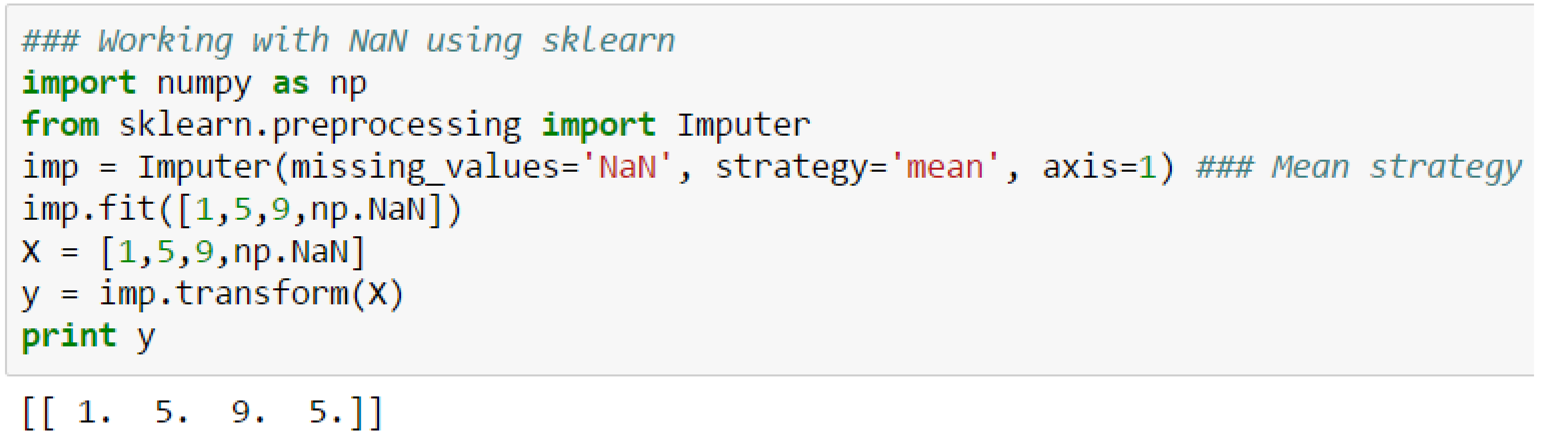
QuantUniversity, LLC
www.quantuniversity.com

Normalization (Python)



```
### Normalizing
from sklearn import preprocessing
y=x.fillna(scipy.mean(x['data']))
y_norm = (y - y.mean()) / (y.max() - y.min()) ### Min-Max
print "Min-Max:"
print y_norm
y_scaled = preprocessing.scale(y) ### z-score
print "Z-score:"
print y_scaled
```

```
Min-Max:
   data
0  -0.5
1   0.0
2   0.5
3   0.0
Z-score:
[[-1.41421356]
 [ 0.        ]
 [ 1.41421356]
 [ 0.        ]]
```



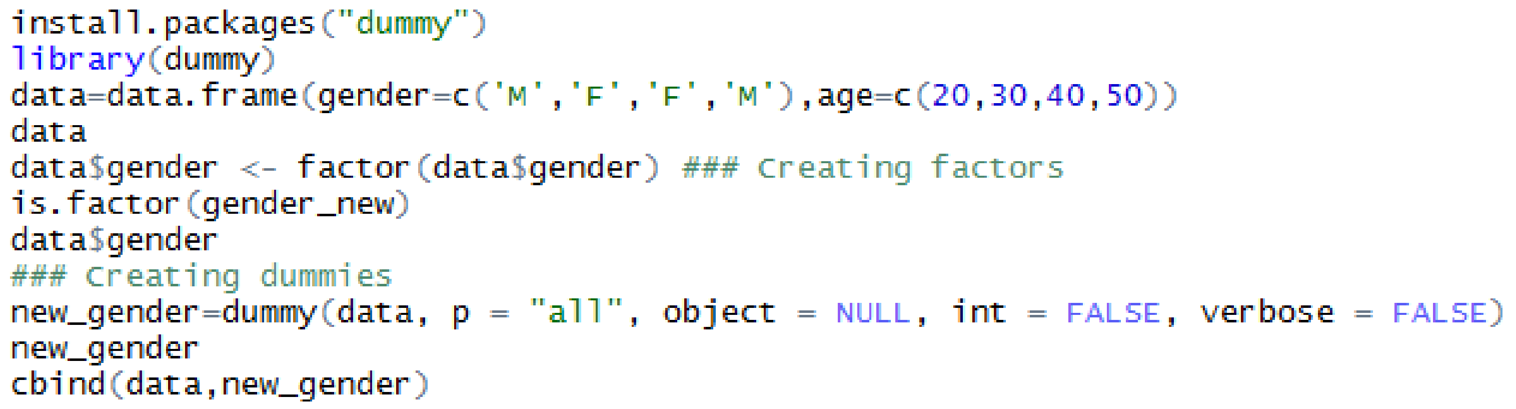
```
### Working with NaN using sklearn
import numpy as np
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values='NaN', strategy='mean', axis=1) ### Mean strategy
imp.fit([1,5,9,np.NaN])
X = [1,5,9,np.NaN]
y = imp.transform(X)
print y
```

```
[[ 1.   5.   9.   5.]]
```

See Data_preprocessing2.ipynb

Dummy variables (R)

- In R, you may either change categorical variables to factors or convert all categorical variables to binary zero and one dummies.

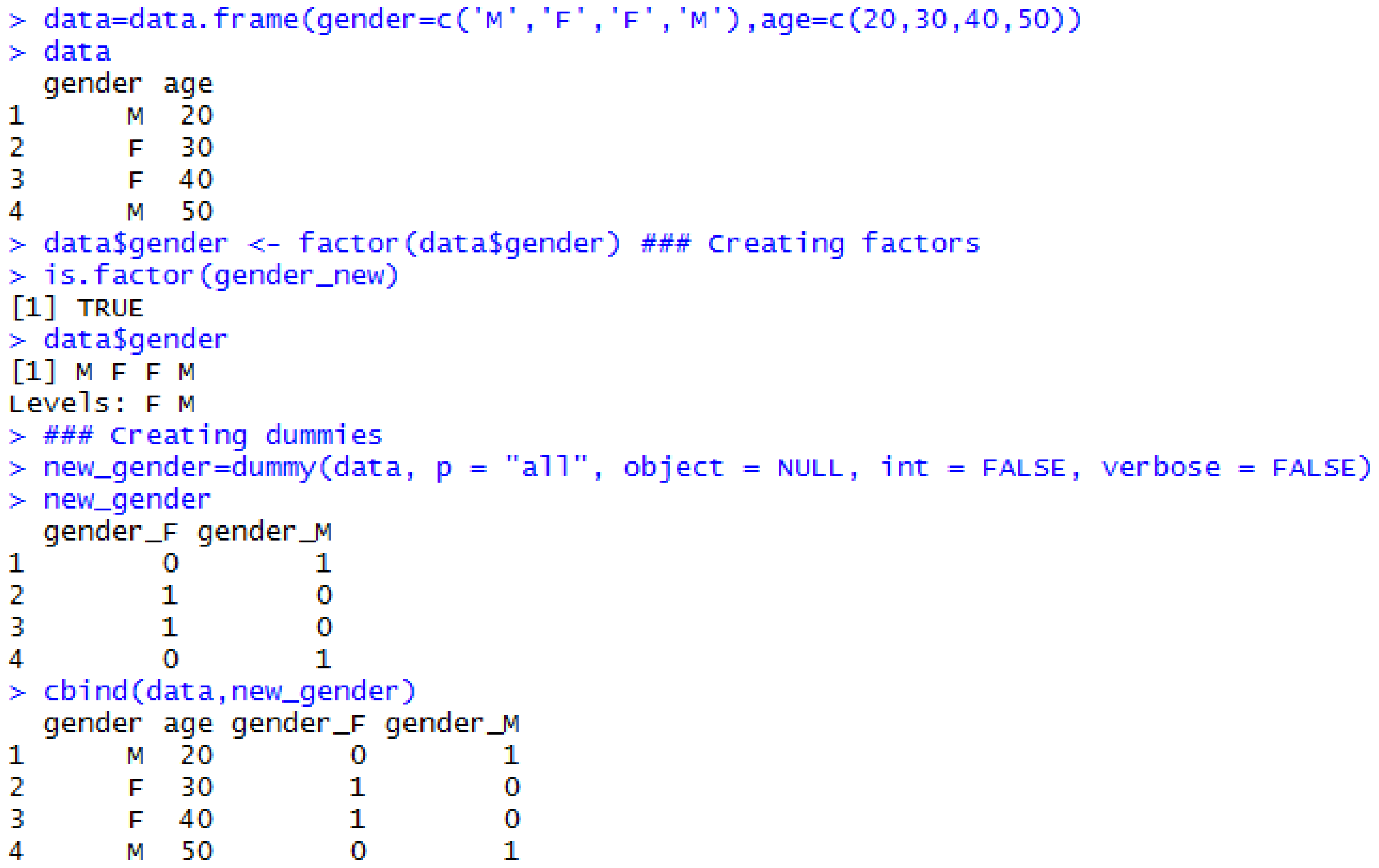


```
install.packages("dummy")
library(dummy)
data=data.frame(gender=c('M','F','F','M'),age=c(20,30,40,50))
data
data$gender <- factor(data$gender) ### Creating factors
is.factor(gender_new)
data$gender
### Creating dummies
new_gender=dummy(data, p = "all", object = NULL, int = FALSE, verbose = FALSE)
new_gender
cbind(data,new_gender)
```

See Data_preprocessing6.R

QuantUniversity, LLC
www.quantuniversity.com

Dummy variables (R)



```
> data=data.frame(gender=c('M','F','F','M'),age=c(20,30,40,50))
> data
  gender age
1      M  20
2      F  30
3      F  40
4      M  50
> data$gender <- factor(data$gender) ### Creating factors
> is.factor(gender_new)
[1] TRUE
> data$gender
[1] M F F M
Levels: F M
> ### Creating dummies
> new_gender=dummy(data, p = "all", object = NULL, int = FALSE, verbose = FALSE)
> new_gender
  gender_F gender_M
1        0        1
2        1        0
3        1        0
4        0        1
> cbind(data,new_gender)
  gender age gender_F gender_M
1      M  20        0        1
2      F  30        1        0
3      F  40        1        0
4      M  50        0        1
```

See Data_preprocessing6.R

Dummy variables (Python)

```python
import pandas as pd
from pandas import *
df = DataFrame({'key': ['b', 'b', 'a', 'c'], 'data1': range(4)})
df
```

	data1	key
0	0	b
1	1	b
2	2	a
3	3	c

```python
pd.get_dummies(df['key'])
```

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1

```python
dummies = pd.get_dummies(df['key'], prefix='key')
df_with_dummy = df[['data1']].join(dummies)
print df_with_dummy
```

```
   data1  key_a  key_b  key_c
0      0      0      1      0
1      1      0      1      0
2      2      1      0      0
3      3      0      0      1
```

See Data_preprocessing6.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Data reduction

- ✓ Data cube aggregation
- ✓ Attribute subset selection
- ✓ Dimensionality reduction
- ✓ Numerosity reduction
- ✓ Discretization and concept hierarchy generation

Data reduction

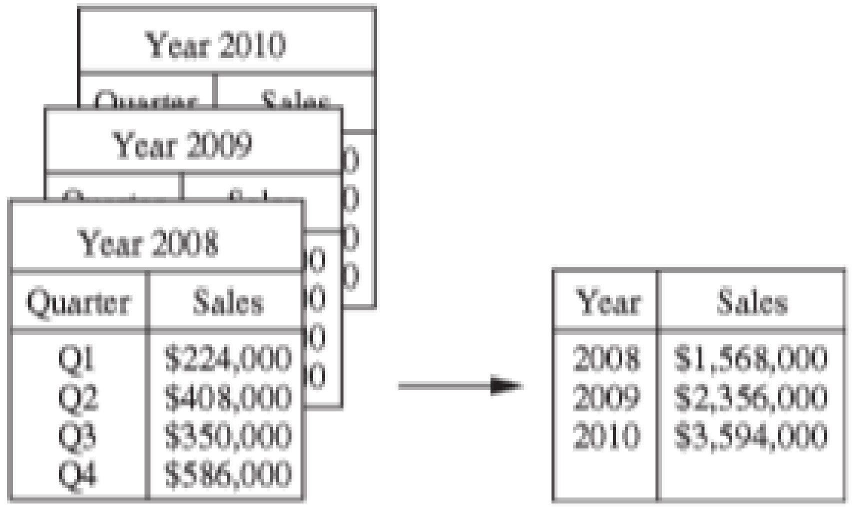
- Data reduction techniques attempt to reduce the representation of the data, while keeping the integrity of the data under consideration.
- Data reduction strategies includes of:
  - Data cube aggregation
  - Attribute subset selection
  - Dimensionality reduction
  - Numerosity reduction
  - Discretization and concept hierarchy generation

Data cube aggregation

- Data can be aggregated in multi dimensional way as cubes.
- Data cubes helps us to access pre-computed and summarized data, very fast.
- We may apply hierarchies for each attribute to allow analysis of the data at multiple levels.
- A cube at highest level is apex cuboid and just gives us a high level understanding of the data.
- A cube at lowest level is base cuboid which is usable for data analysis.

Numerosity reduction: Data cube aggregation

- Data cubes stores data in multidimensional data. The cube created as lowest level is referred to the base cuboid and the one at highest level as apex cuboid. Below figures show two and three dimensional cubes.

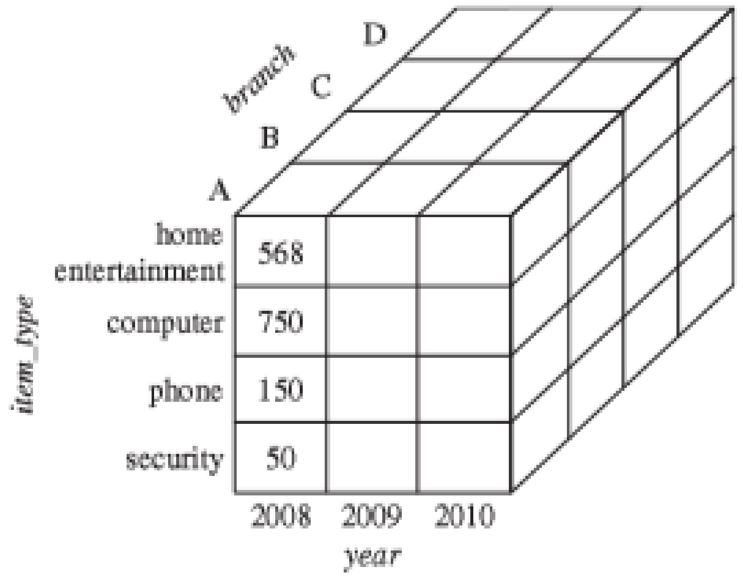


Year 2010
Year 2009

Year 2008

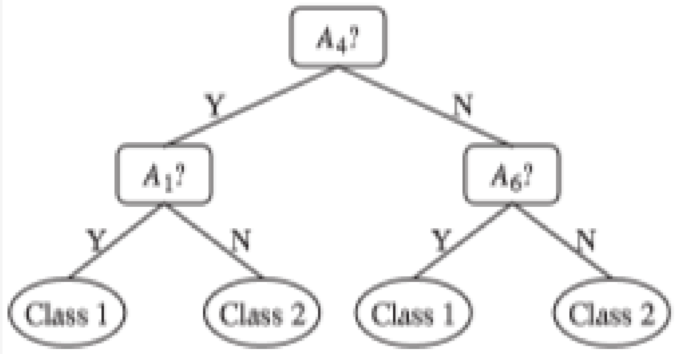
Quarter	Sales
Q1	$224,000
Q2	$408,000
Q3	$350,000
Q4	$586,000

Year	Sales
2008	$1,568,000
2009	$2,356,000
2010	$3,594,000



Attribute subset selection

- Datasets may include features which may be irrelevant or redundant. Attribute subset selection reduces the number of variables by removing redundant and irrelevant ones.

Forward selection	Backward elimination	Decision tree induction
Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ Initial reduced set: $\{\}$ => $\{A_1\}$ => $\{A_1, A_4\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_3, A_4, A_5, A_6\}$ => $\{A_1, A_4, A_5, A_6\}$ => Reduced attribute set: $\{A_1, A_4, A_6\}$	Initial attribute set: $\{A_1, A_2, A_3, A_4, A_5, A_6\}$  A_4? Y N A_1? A_6? Y N Y N Class 1 Class 2 Class 1 Class 2 => Reduced attribute set: $\{A_1, A_4, A_6\}$

Greedy (heuristic) methods for attribute subset selection.

Attribute subset selection

- **Forward selection:** Here the model adds one predictor at a time and continues until the time that adding another predictor is no longer statistically significant.
- **Backward selection:** It is the opposite of forward selection and all variables are included in the model to start with and variables are dropped one at a time till only the statistically significant variables remain.
- **Stepwise regression:** It combines both Forward and Backward eliminations and drops/adds variables based on their statistical significance.

Dimensionality reduction

- Dimension reduction can be categorized into two main groups of **variable selection and variable reduction.**
- The goal of dimension reduction is having few number of variables which capture the meaningful information of the data instead of massive number of variables.
- Variable subset selection methods are defined as choosing the best features of the dataset while variable reduction may change the origin of the variables and transform them to a new form such as linear combination.

Dimensionality reduction

- When variables are highly correlated, PCA can be used to transform a large set of variables into a smaller set of variables that have the predictive power of the original variable set.
- The new variables are a weighted linear combination of the original variables and are uncorrelated.
- The first few components capture most of the variability observed in the original dataset.
- Works well for sparse data.

Numerosity reduction

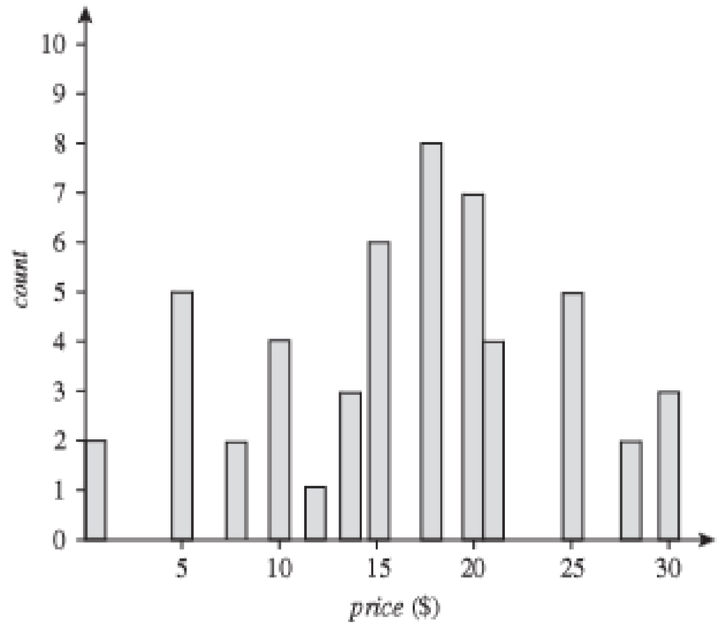
- Numerosity reduction techniques try to find smaller forms of data and includes of both **parametric** and **nonparametric** methods.
  - For parametric methods a model is utilized to estimate the data, so that the parameters of the model needs to be stored instead of actual data. Linear models: In linear regression a random variable y can be modeled by a linear function of another random variable x, as y=wx+b considering assumptions of linearity, randomness of error and equality of variance for y.

Numerosity reduction

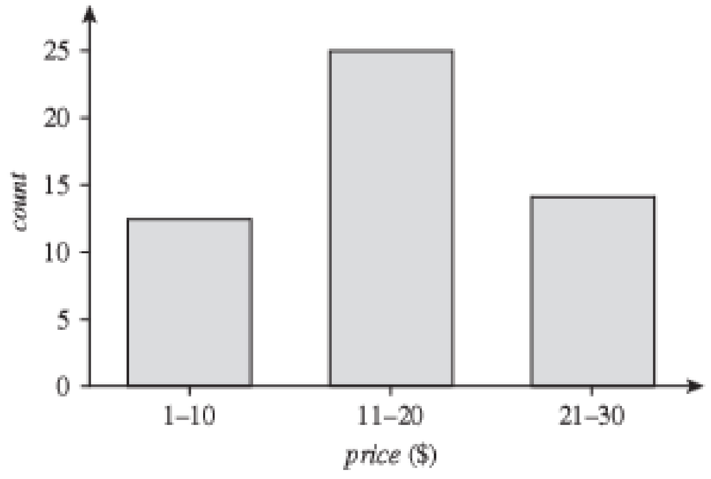
- Nonparametric methods includes of :
  - **Histograms:** Approximates data distribution by applying binning methods.
  - **Clustering:** Categorizes data into different clusters or groups by defining level of similarity of data objects, such a way each cluster is representative all included data objects. It allows us to have a simple random from any desirable cluster.
  - **Sampling:** Allows you to have a smaller number of data as a random dataset instead of working with a large dataset. By doing sampling method we may reduce the cost and complexity of working with huge number of records and variables.

Numerosity reduction: Histogram

- In histogram if each bin shows only a single attribute-value frequency pair the buckets called singleton buckets.



A histogram for *price* using singleton buckets—each bucket represents one price–value/frequency pair.



An equal-width histogram for *price*, where values are aggregated so that each bucket has a uniform width of $10.

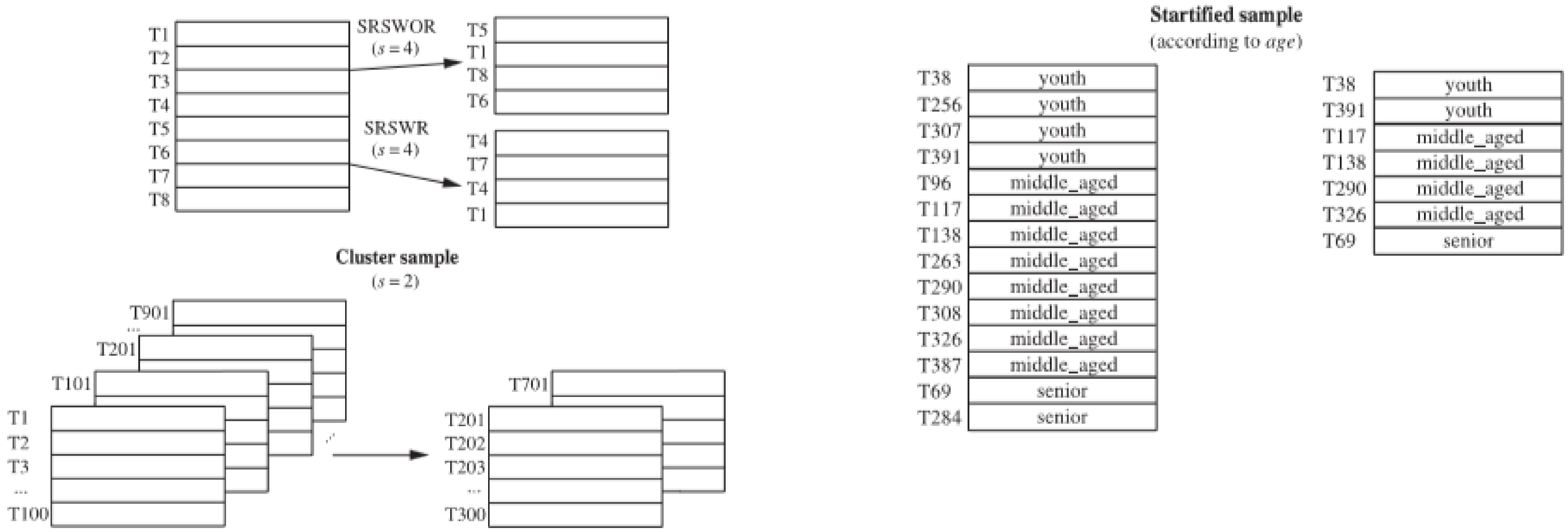
QuantUniversity, LLC
www.quantuniversity.com

Numerosity reduction: Sampling

- There are simple and complex sampling method as:
- **Simple random sampling without replacement**: After an object being selected, it will not be replaced to population
- **Simple random sampling with replacement**: After an object chosen in sample it will be replaced back to population.
- **Stratified sampling**: If data comes in groups sample should include data from all stratums or groups.
- **Cluster sampling**: If data includes clusters or blocks, some of these clusters will be selected randomly.

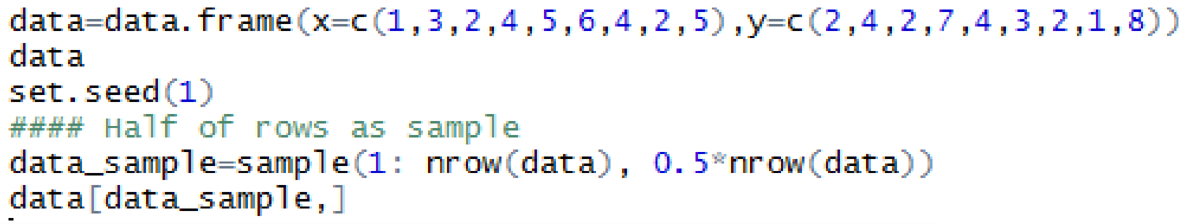
Numerosity reduction: Sampling

- Below figure show a schematic of sampling methods:



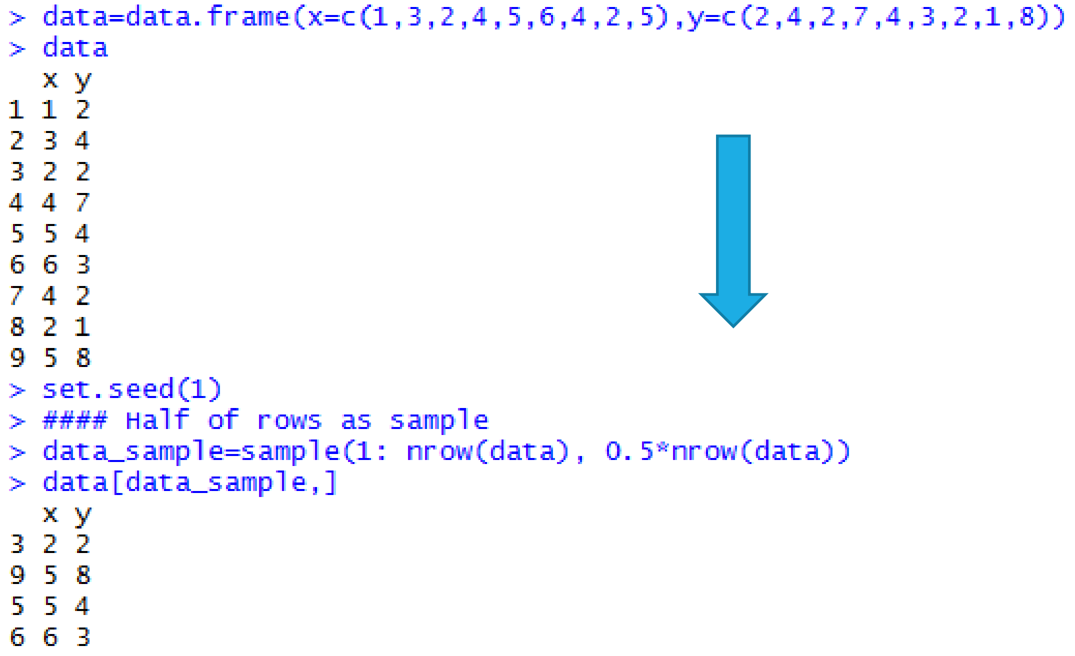
QuantUniversity, LLC
www.quantuniversity.com

Sampling (R)



```
data=data.frame(x=c(1,3,2,4,5,6,4,2,5),y=c(2,4,2,7,4,3,2,1,8))
data
set.seed(1)
#### Half of rows as sample
data_sample=sample(1: nrow(data), 0.5*nrow(data))
data[data_sample,]
```

Sample() function and nrow(data) allows you to take a random sample of records from a dataset, the third parameter, 0.5*nrow(data) is the sample size and it can be any specific number.



```
> data=data.frame(x=c(1,3,2,4,5,6,4,2,5),y=c(2,4,2,7,4,3,2,1,8))
> data
  x y
1 1 2
2 3 4
3 2 2
4 4 7
5 5 4
6 6 3
7 4 2
8 2 1
9 5 8
> set.seed(1)
> #### Half of rows as sample
> data_sample=sample(1: nrow(data), 0.5*nrow(data))
> data[data_sample,]
  x y
3 2 2
9 5 8
5 5 4
6 6 3
```

See Data_preprocessing7.R

QuantUniversity, LLC
www.quantuniversity.com

Sampling (Python)

```
import pandas
from pandas import *
import numpy as np
```

```
df = DataFrame(np.arange(20).reshape(5, 4))
df
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

df.take() and random.permutation are two key components of random sampling without replacement from a data frame.

```
df.take(np.random.permutation(len(df))[:2]) ### Two sample rows (without replacement)
```

	0	1	2	3
2	8	9	10	11
1	4	5	6	7

```
bag = np.array([5, 7, -1, 6, 4])
np.random.seed(1)
sampler = np.random.randint(0, len(bag), size=10)
print sampler
```

```
[3 4 0 1 3 0 0 1 4 4]
```

```
draws = bag.take(sampler) ### 10 repeated samples
print draws
```

```
[6 4 5 7 6 5 5 7 4 4]
```

Defining sampler() parameter and bag.take() can be used to take a random sample with replacement in python.

See Data_preprocessing7.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Discretization and concept hierarchy generation

- Data discretization techniques reduce the number of values for continuous variable by dividing them to intervals.
- This enables replacing actual values with interval labels and work with small number of labels instead of original data.
- Techniques for discrete numerical variables include binning, histogram analysis, cluster analysis, interval merging by chi-square analysis , entropy–based discretization and intuitive partitioning.
- Concept hierarchy generation may be used for categorical data with too many outcomes and no ordering such as geographic data, job category and item type.

Data transformation strategies overview

- Smoothing: Attempts to remove noises from data.(binning, regression and clustering are the techniques)
- Attribute construction: New attributes will be added to mine data better.
- Aggregation: Applying aggregation function such as daily data to monthly and annually data.
- Normalization: Scaling data to a range of [0,1] or [-1,1].
- Discretization: Transforming numerical data to categorical ones.
- Concept hierarchy generation for nominal data: Categorical data such as street or city are aggregated to higher level such as state or country.

Discretization of numerical variables

- **Discretization by binning**: This method splits numbers to bins.
- Bins can be equal width or equal frequent values.
- Each bin value can be smoothed by bin mean or median as smoothing by mean or median.
- This method can be applied to generate concept hierarchies.
- **Discretization by histogram**: Like binning, histogram groups data into bins. Bins or buckets can have the same range or contain equal number of data.
- Histogram analysis can be applied recursively to each partition to reach a multilevel concept hierarchy.
- A minimum interval size should be used to control this recursive procedure.

Discretization of numerical variables

- **Discretization by cluster analysis**: Clustering algorithm partitions the value of numerical variable into groups to create a high quality discretization results.
- The closeness of data points are taken into accounts in clustering as well as distribution of data.
- Clusters may include sub-clusters to form a low level hierarchy.
- **Discretization by decision tree:** Decision trees uses class information to employ a top-down splitting approach.
- The idea behind splitting is creating partitions which contains as many tuple of the same class.

QuantUniversity, LLC
www.quantuniversity.com

Discretization of numerical variables

- To do that, decision trees apply entropy measure such that the splitting point results minimum entropy.
- **Discretization by correlation:** ChiMerge is a chi-squared based discretization method.
- Correlation method applies a bottom up approach to find the best neighboring intervals to merge data and form larger intervals.
- First each distinct data point considered to as an interval, then among all pairs of adjacent intervals the one with lowest chi-squared will be selected.
- The lower chi-squared greater level of similar class distribution.

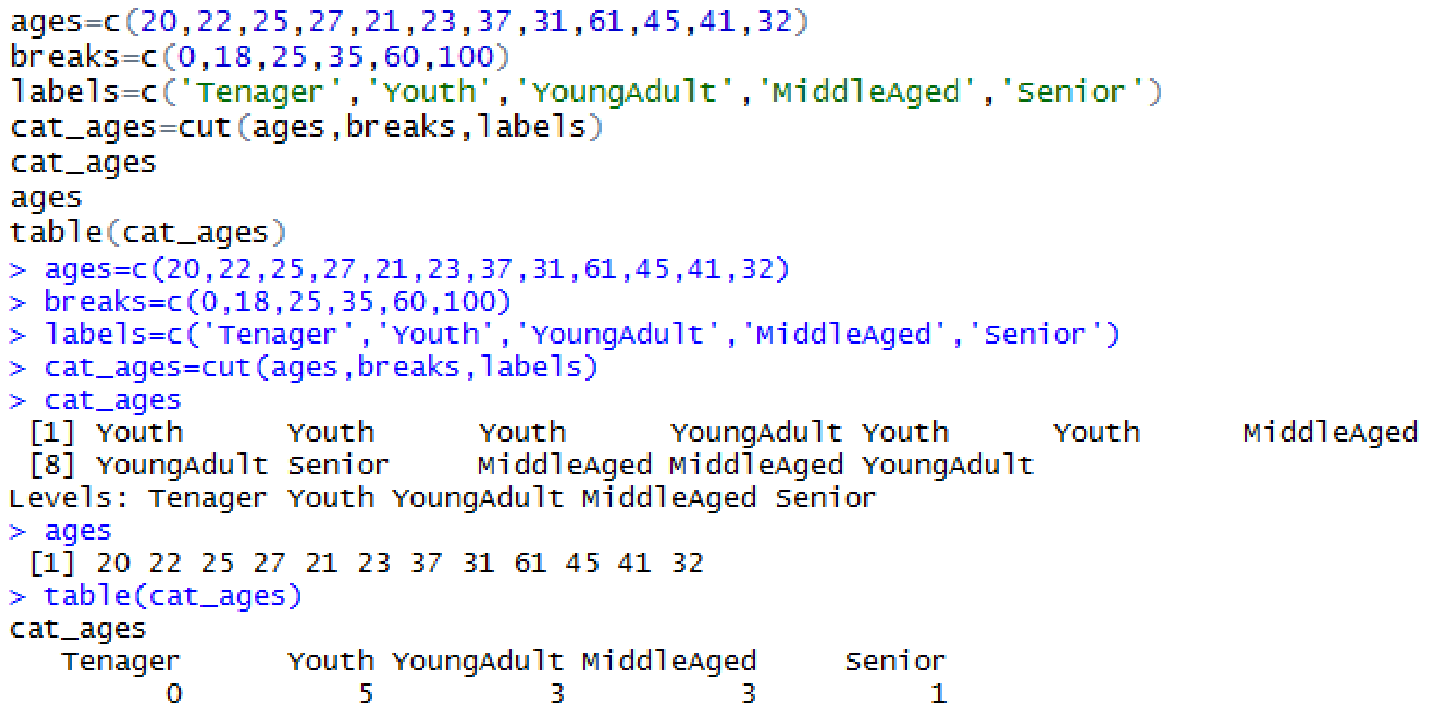
Concept hierarchy generation for categorical data

- There are four main methods of generating concept hierarchies for categorical data.
- **Defining a by partial ordering:** One may define an order to define a concept hierarchy, for example street < city < state < country can be used as an order
- **Defining an explicit data grouping:** We can define explicit grouping for a small portion of intermediate level data.
- For example after defining states we can specify some intermediate level as {Massachusetts, New Hampshire, Rhode Island, Connecticut and Vermont} as “New England” states.

Concept hierarchy generation for categorical data

- **Defining a set of attributes but not of their partial ordering:** A concept hierarchy can be generated based on the number of distinct possible outcomes per categorical variable in a set of categorical features.
- The variables with lowest number of possible outcomes placed at the highest level of hierarchy.
- **Using pre-specified semantic connection:** One may define a set of variables together as they are very related to a higher level variable. For instance {city, street, state} are semantically linked regarding of location.

Discretization (R)



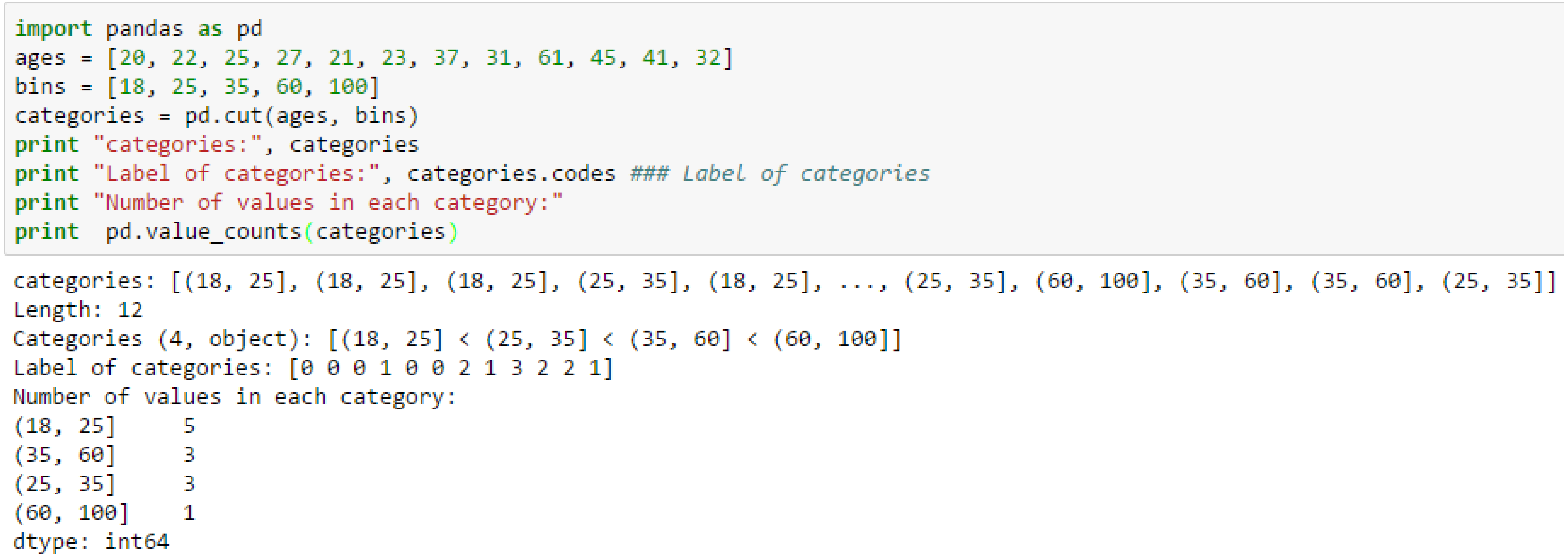
```
ages=c(20,22,25,27,21,23,37,31,61,45,41,32)
breaks=c(0,18,25,35,60,100)
labels=c('Tenager','Youth','YoungAdult','MiddleAged','Senior')
cat_ages=cut(ages,breaks,labels)
cat_ages
ages
table(cat_ages)
```

```
> ages=c(20,22,25,27,21,23,37,31,61,45,41,32)
> breaks=c(0,18,25,35,60,100)
> labels=c('Tenager','Youth','YoungAdult','MiddleAged','Senior')
> cat_ages=cut(ages,breaks,labels)
> cat_ages
 [1] Youth      Youth      Youth      YoungAdult Youth      Youth      MiddleAged
 [8] YoungAdult Senior     MiddleAged MiddleAged YoungAdult
Levels: Tenager Youth YoungAdult MiddleAged Senior
> ages
 [1] 20 22 25 27 21 23 37 31 61 45 41 32
> table(cat_ages)
cat_ages
   Tenager      Youth YoungAdult MiddleAged     Senior 
         0          5          3          3          1 
```

See Data_preprocessing8.R

QuantUniversity, LLC
www.quantuniversity.com

Discretization (Python)

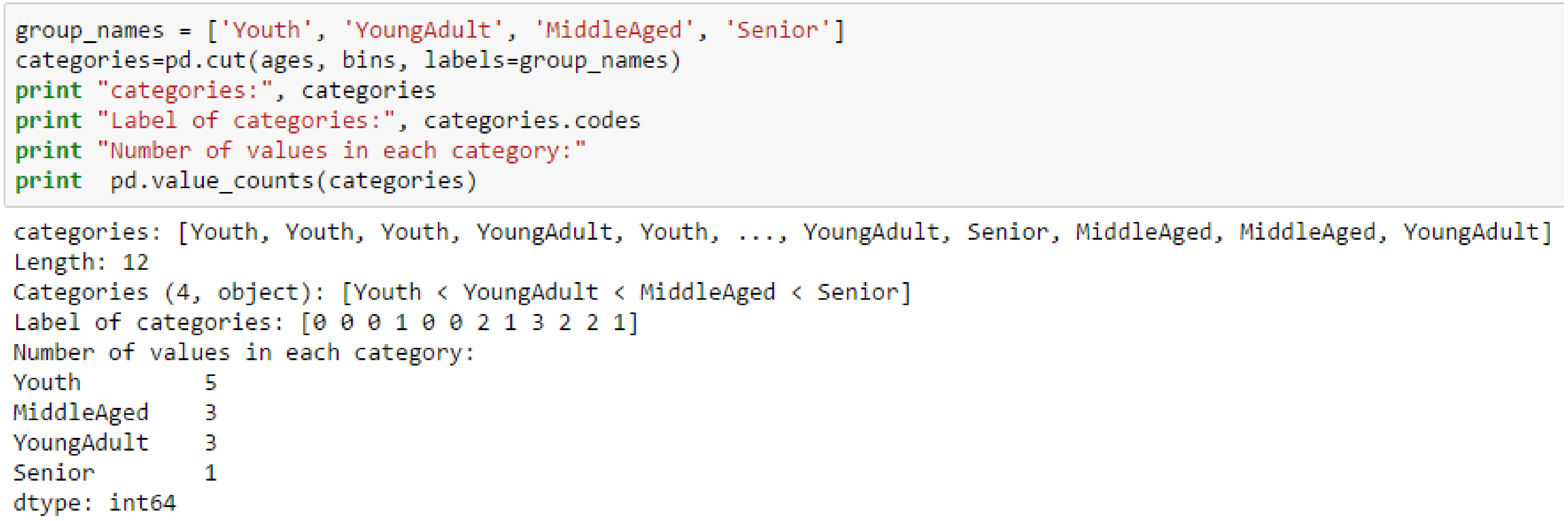


```python
import pandas as pd
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
categories = pd.cut(ages, bins)
print "categories:", categories
print "Label of categories:", categories.codes ### Label of categories
print "Number of values in each category:"
print  pd.value_counts(categories)
```

```
categories: [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, object): [(18, 25] < (25, 35] < (35, 60] < (60, 100]]
Label of categories: [0 0 0 1 0 0 2 1 3 2 2 1]
Number of values in each category:
(18, 25]     5
(35, 60]     3
(25, 35]     3
(60, 100]    1
dtype: int64
```

See Data_preprocessing8.ipynb

Discretization (Python)



```python
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
categories=pd.cut(ages, bins, labels=group_names)
print "categories:", categories
print "Label of categories:", categories.codes
print "Number of values in each category:"
print  pd.value_counts(categories)
```

```
categories: [Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior, MiddleAged, MiddleAged, YoungAdult]
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
Label of categories: [0 0 0 1 0 0 2 1 3 2 2 1]
Number of values in each category:
Youth         5
MiddleAged    3
YoungAdult    3
Senior        1
dtype: int64
```

See Data_preprocessing8.ipynb

QuantUniversity, LLC
www.quantuniversity.com

Summary

We have covered	Data preprocessing
Introduction	✓ Why data preprocessing and what is data preprocessing?
Descriptive data summarization	✓ Measuring central tendency and dispersion of data (Mean, median, variance, quintiles, sign of skewness) for numerical data and counting factors for categorical data ✓ Graphic display tools of descriptive data summarization (categorical-numerical: side-by-side box plots, numerical-numerical: scatterplot and cross tabs for categorical-categorical data types)
Data cleaning	✓ To deal with missing values, ignoring or replacing them with other values ✓ Detecting outliers and replacing them with mean of column
Data integration and transformation	✓ Difficulties and issues of data integration, different transformation techniques and methods (Min-Max, Z-score and decimal scaling normalization, transforming categorical variables by using factors or dummies) ✓ Correlation coefficient and correlation matrix ✓ Splitting data into subgroups and working with them (Getting summaries, statistics and doing function on them)
Data reduction	✓ Data cube aggregation ✓ Feature subset selection (Forward, backward, stepwise elimination) ✓ Dimensionality reduction (PCA) ✓ Numerosity reduction methods such as binning and sampling (with/without replacement) ✓ Discretization and concept hierarchy generation

Summary

We have covered	Data wrangling
Introduction	✓ The steps and definitions of a data project ✓ Different types of data as categorical, numerical, qualitative and quantitative ✓ How we can access the data and make a story about the data ✓ Accessing existing local data (read as csv, table, Excel and JSON formats) ✓ Loading and parsing external HTML data such as links and tables
Accessing and combining data	✓ Merging datasets like SQL type including inner, left, right and outer joins, joining on one key or more than one key ✓ Concatenating datasets on indexes, rows and columns ✓ How to reshape, transpose and pivot the data, changing long format to wide ✓ Filtering, splitting, and sorting data
Data transformations	✓ Dropping duplicated values of a dataset ✓ Different methods for adding a new column to dataset ✓ How to create some new values by mapping on other values ✓ Replacing some values of data such as missing values with mean or any specific value ✓ Renaming datasets along indexes, rows and columns