NYC DATA SCIENCE
ACADEMY

Data Analytics in Python
Pandas Part 1

NYC Data Science Academy

OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

Pandas

- Pandas is a large package defining several new data types, plus a variety of convenient functions for data manipulation, plotting, and web scraping.
- The *`DataFrame`* structure is inspired by the type of the same name in R, a programming language popular among statisticians and data scientists.
- Pandas is particularly strong in the area of handling missing data and, relatedly, handling time series data.
- There are four new data structures in pandas: `Series`, `DataFrame`, `time series` and `panel`. We will mainly discuss the first three.

Pandas data types

- These are the new data types introduced by pandas:
  - **Series**: 1D labeled homogeneously-typed array.
  - **Time Series**: Series with index containing datetimes.
  - **DataFrame**: General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed columns.
  - **Panel**: General 3D labeled, also size-mutable array.
- We first import the package:

```
import numpy as np
import pandas as pd
```

OVERVIEW

- Data Structure
- **Series**
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

Series

- A series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its *index*. By default, the index just consists of ordinary array indices, i.e. consecutive integers starting from zero.

```
obj = pd.Series(['a', 'b', 'c', 'd'])
obj
0   a
1   b
2   c
3   d
```

Series

- Often it will be more desirable to create a series with an index identifying each data point. Here we manually set the index from 1 to 4.

```
obj2 = pd.Series(['a', 'b', 'c', 'd'], index=[1, 2, 3, 4])
obj2
1    a
2    b
3    c
4    d
```

- We can also modify the index directly.

```
obj.index = ['A', 'B', 'C', 'D']
obj     # Check the result
```

Series

- We can access values in a series by index.

```
obj['A']
'a'
obj[['A', 'B', 'C']]
A    a
B    b
C    c
```

- The method `values` accesses all the values.

```
obj.values
array(['a', 'b', 'c', 'd'], dtype=object)
obj.values[1]
'b'
```

Series

- The Series object is similar to a dictionary, *Series.index* is like *dictionary.keys*, and *Series.values* is like *dictionary.values*. We can convert a dictionary to a Series directly:

```
dict_ = {1: 'a', 2: 'b', 3: 'c', 4: 'd'}
obj3 = pd.Series(dict_)
obj3
1    a
2    b
3    c
4    d
```

```
obj3.to_dict()          # convert Series to dict
{1: 'a', 2: 'b', 3: 'c', 4: 'd'}
```

In class lab 1: Series

- Create a pandas Series whose entries are `['analyst', 'associate', 'VP', 'analyst']`. Call it '*title*'.
- Index *series* by `['Bob', 'Sam', 'Peter', 'Jake']`.
- Create the same Series with dictionary notation. Call it '*title_2*'.
- Check if *title* equal to *title_2*. If this is NOT the case, why?
- How do we fix the problem in the last problem? Try to use the **sort_values** method. If you don't know what it is, google 'sort pandas series'.

OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

DataFrame

- A data frame represents a tabular, spreadsheet-like data structure containing an ordered collection of columns; each can be of a different value type (integers, strings, floating point numbers, Python objects, etc.), but all must be the same length.

```
# create a dictionary
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
# convert to DataFrame
df = pd.DataFrame(data)
df
```

DataFrame

- A data frame can be created with a nested list as well.

```
df_2 = pd.DataFrame([[1.5, 'Ohio', 2000],
                     [1.7, 'Ohio', 2001],
                     [3.6, 'Ohio', 2002],
                     [2.4, 'Nevada', 2001],
                     [2.9, 'Nevada', 2002]],
                      columns=['pop','state','year'])
```

- The two ways are equivalent.

DataFrame

- A DataFrame has an attribute **values**, which is of the multidimensional array type.

```
df.values
array([[1.5, 'Ohio', 2000],
       [1.7, 'Ohio', 2001],
       [3.6, 'Ohio', 2002],
       [2.4, 'Nevada', 2001],
       [2.9, 'Nevada', 2002]], dtype=object)
```

- `df_2.values` gives the same result.

DataFrame

- DataFrame v.s. Series is similar to 2D array v.s. 1D array. A data frame has column names.

```
df.columns    # column name
# here u'pop' means the string 'pop' is encoded in unicode
Index([u'pop', u'state', u'year'], dtype='object')
```

DataFrame

- Each column in a data frame can be retrieved as a Series. We have two ways to get the column: to retrieve by attribute and to retrieve by dictionary-like notation. They will give the same result.

```
df.year         # retrieve by attribute
df['year']      # retrieve by dictionary-like notation

0     2000
1     2001
2     2002
3     2001
4     2002
Name: year, dtype: int64
```

In class lab 2: DataFrame

- Create a Pandas DataFrame, 'Employee', whose columns are 'Name', 'Year' and 'Department'. The rows are supposed to be:
  - Bob has been working for IT department for a year.
  - Sam has been working for Trade department for 3 years.
  - Peter has been working for HR department for 8 years.
  - Jake has been working for IT department for 2 years.
- Now set the index of Employee to be their names. Make sure you update the DataFrame.
  - **Remark**: recording information in the index can cause problems when applying the *merge* function, as we will see this later.

In class lab 2: DataFrame

- What is the type of each column in a DataFrame?

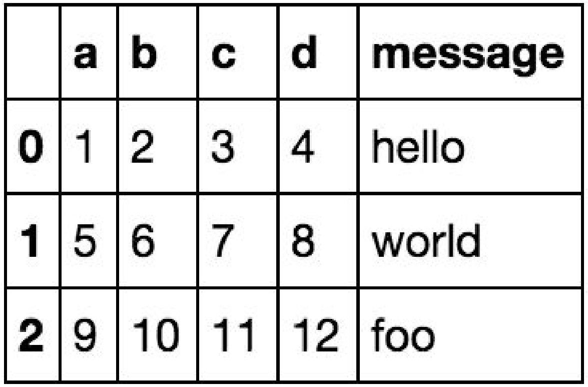
OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

DataFrame

- Pandas has a number of functions for reading tabular data as a DataFrame object.

```
pd.read_csv('foo.csv')    # use comma as the default delimiter
pd.read_table('foo.txt')  # use tab as the default delimiter
```



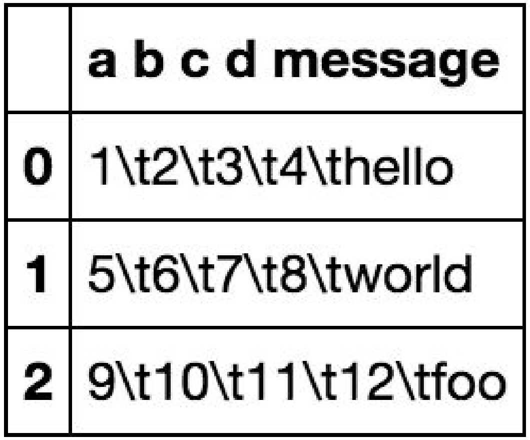
	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

- Note that both functions consider the first row as a header giving the column names, and both add incremental numbers as indices.

DataFrame

- Parsing can't be done properly with a bad delimiter.

```
# read_csv reads a \t separated file
pd.read_csv('foo.txt')
```



	a b c d message
0	1\t2\t3\t4\thello
1	5\t6\t7\t8\tworld
2	9\t10\t11\t12\tfoo

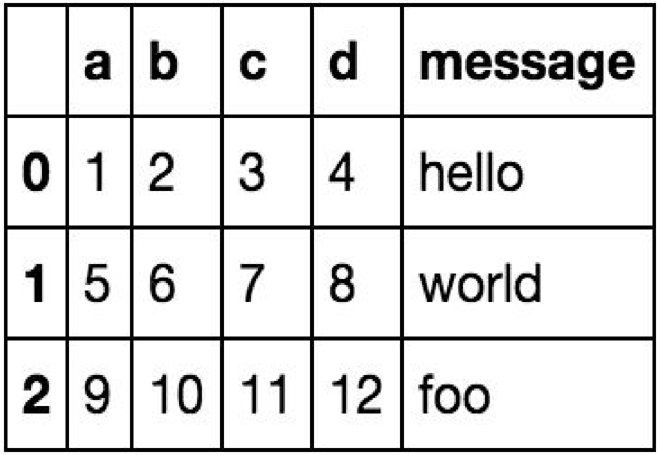
- We see the DataFrame becomes messy with a bad delimiter.

DataFrame

- The problem will be fixed by passing `sep = '\t'` to read_csv.

```
# read_csv reads a \t separated file
pd.read_csv('foo.txt', sep='\t')
```

.



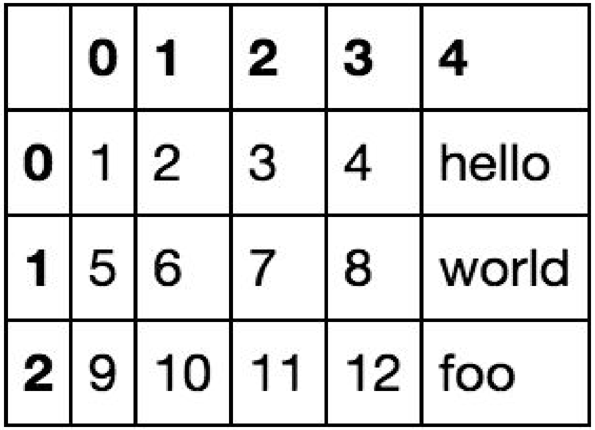
	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

DataFrame

- In some cases, there is no header in the file. With argument `header = None`, the column names will be filled with incremental numbers.

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

.



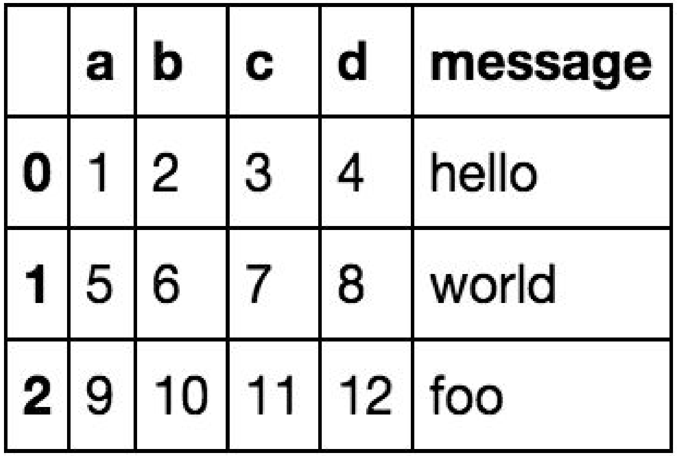
	0	1	2	3	4
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

DataFrame

- But we can manually set the names of the columns by passing the list of column names.

```
# Set the names manually
pd.read_csv('foo_noheader.csv',
            names=['a', 'b', 'c', 'd', 'message'])
```

.



	a	b	c	d	message
0	1	2	3	4	hello
1	5	6	7	8	world
2	9	10	11	12	foo

In class lab 3: I/O tools

So far we covered only importing a file. With this exercise we first demonstrate how exporting is done.

- Write the data frame, `Employee`, to a file, Employee.csv. Use function [`to_csv`].
- Read the csv file back as a data frame and call it `Employee2`.
- Check if `Employee == Employee2`. If not, change the way you read the csv file to fix it.

In class lab 3: I/O tools

- Read the file Employee.txt into a data frame.

OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- **Data Manipulation in Pandas**
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

Data manipulation in pandas

- Like numpy, pandas defines many broadcast operations, as well as numerous methods of manipulating data.

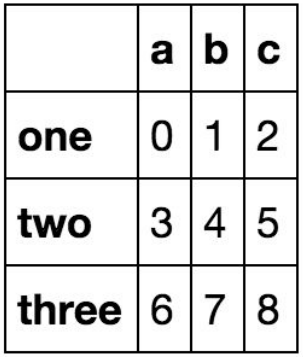
OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - **concat**
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

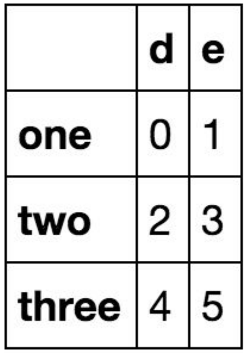
Data manipulation in pandas: concat

- Pandas DataFrames can be expanded in both directions. Let’s create two data frames first.

```
df1 = pd.DataFrame(np.arange(9).reshape((3, 3)),
                   columns=['a', 'b', 'c'],
                   index=['one', 'two', 'three'])
df2 = pd.DataFrame(np.arange(6).reshape((3, 2)),
                   columns=['d','e'],
                   index=['one', 'two', 'three'])
```



	a	b	c
one	0	1	2
two	3	4	5
three	6	7	8

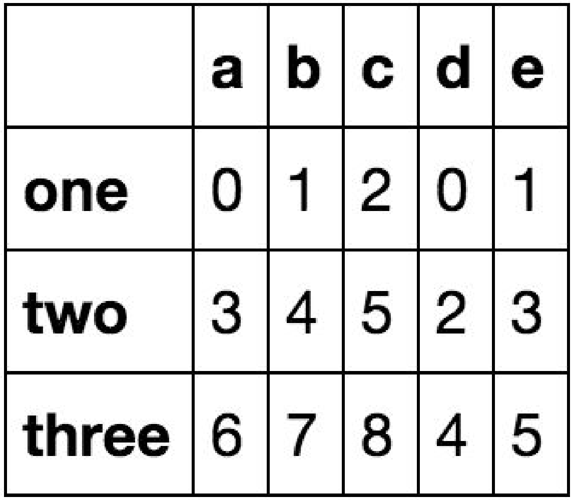


	d	e
one	0	1
two	2	3
three	4	5

Data manipulation in pandas: concat

- Since the two data frames have exactly the same rows, it is natural that we can combine them "horizontally".

```
pd.concat([df1, df2], axis = 1)
```



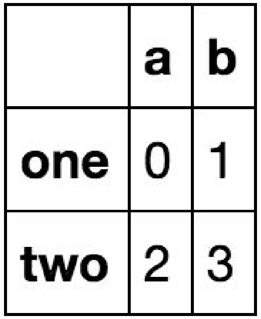
	a	b	c	d	e
one	0	1	2	0	1
two	3	4	5	2	3
three	6	7	8	4	5

- The argument "axis = 1" means expanding along the column indices.

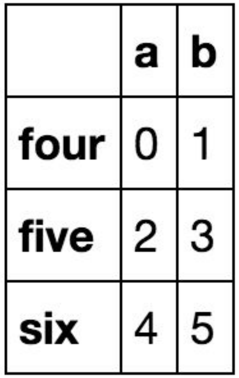
Data manipulation in pandas: concat

- Sometimes, we would like to extend the Pandas DataFrames in a vertical direction. Let’s create two data frames first.

```
df1 = pd.DataFrame(np.arange(4).reshape((2, 2)),
                   columns=['a', 'b'],
                   index=['one', 'two'])
df2 = pd.DataFrame(np.arange(6).reshape((3, 2)),
                   columns=['a','b'],
                   index=['four', 'five', 'six'])
```



	a	b
one	0	1
two	2	3

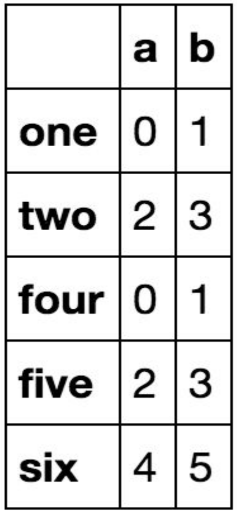


	a	b
four	0	1
five	2	3
six	4	5

Data manipulation in pandas: concat

- We can still use concat

```
pd.concat([df1, df2], axis = 0)
pd.concat([df1, df2])
```



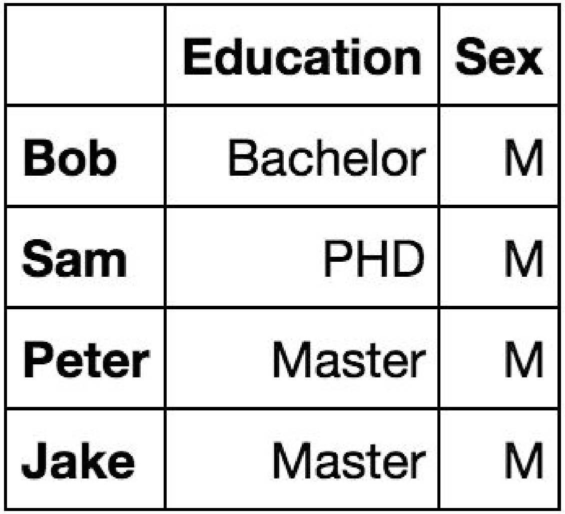
	a	b
one	0	1
two	2	3
four	0	1
five	2	3
six	4	5

- The argument 'axis =0' expands the data frames along the row indices. This is actually the default setting, so the second line of code performs the same task.

In class lab 4: concat

- Before we concatenate multiple DataFrames, let's consider an easier case. Recall that we created a Series, *title*. Combine it with our `Employee` data frame.

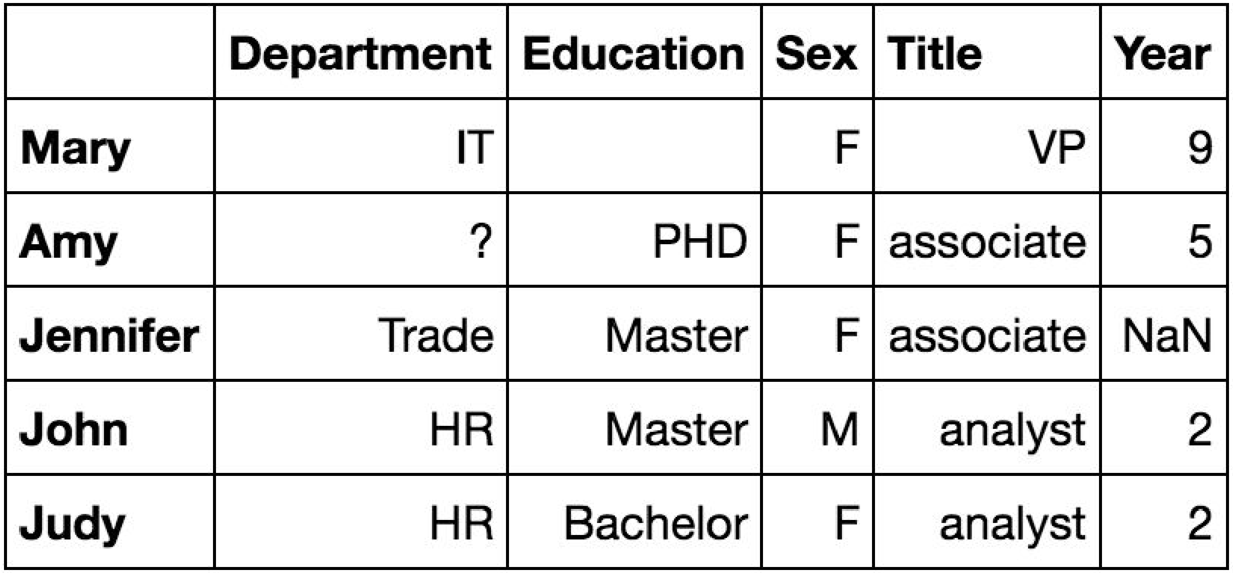
- In the iPython notebook, we created the data frame below. How should we combine it with the old `Employee`? Observe that this is a data frame with new features.



	Education	Sex
Bob	Bachelor	M
Sam	PHD	M
Peter	Master	M
Jake	Master	M

In class lab 4: concat

- In the iPython notebook, we created the data frame below. How should we combine it with the old `Employee`? Observe that this is a data frame with new observations.



	Department	Education	Sex	Title	Year
Mary	IT		F	VP	9
Amy	?	PHD	F	associate	5
Jennifer	Trade	Master	F	associate	NaN
John	HR	Master	M	analyst	2
Judy	HR	Bachelor	F	analyst	2

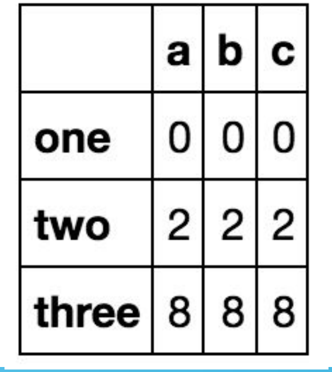
OVERVIEW

- ❖ Data Structure
- ❖ Series
- ❖ Data Frame
  - ➢ I/O tool
- ❖ Data Manipulation in Pandas
  - ➢ concat
  - ➢ **merge**
- ❖ More on Data Manipulation
  - ➢ arithmetic, drop, apply and describe
  - ➢ selection and filter
- ❖ Handling missing values

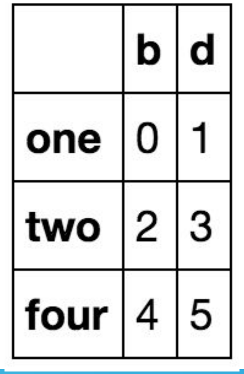
Data manipulation in pandas: merge

- Merging is the most common way to combine multiple data frames. Let’s create two data frames first.

```
df1 = pd.DataFrame(np.array([0,0,0,2,2,2,8,8,8]).\
                   reshape((3, 3)),columns=['a','b','c'],
                   index=['one', 'two', 'three'])
df2 = pd.DataFrame(np.arange(6).reshape((3, 2)),
                   columns=['b', 'd'],
                   index=['one', 'two', 'four'])
```



	a	b	c
one	0	0	0
two	2	2	2
three	8	8	8

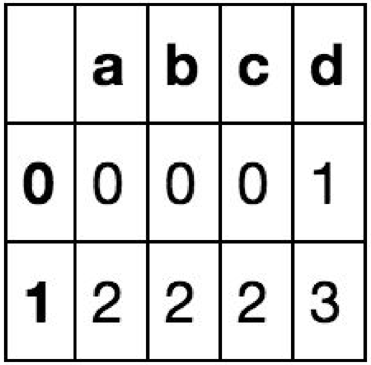


	b	d
one	0	1
two	2	3
four	4	5

Data manipulation in pandas: merge

- The code identifies the column 'b' from both data frames. The argument 'inner' means it only keeps rows occur in both data frames.

```
pd.merge(df1, df2, how='inner', on ='b')
```



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

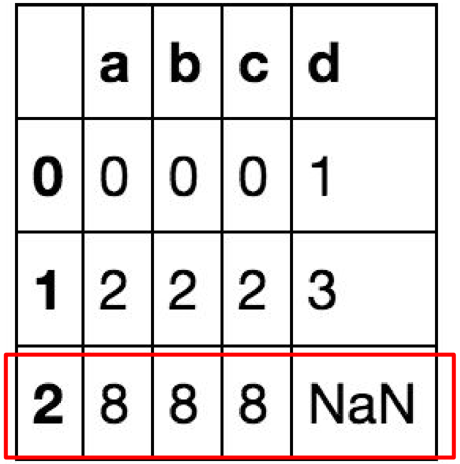
- The 'how' argument defaults to 'inner'. So the following code performs the same task as above.

```
pd.merge(df1, df2, on ='b')
```

Data manipulation in pandas: merge

- If we want to keep every row in df1, then we can specify how = "left".

```
pd.merge(df1, df2, how='left', on ='b')
```



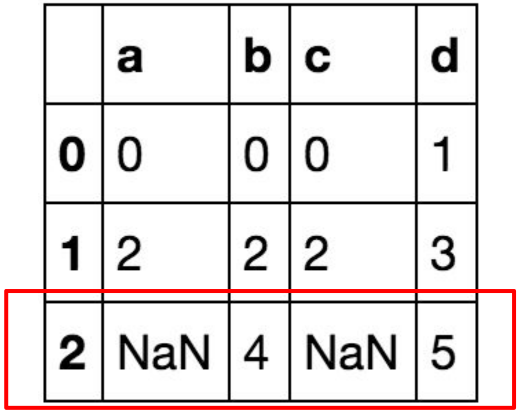
	a	b	c	d
0	0	0	0	1
1	2	2	2	3
2	8	8	8	NaN

- Since df2 does not have a row with b=8, pandas leaves NaN for column d.

Data manipulation in pandas: merge

- If we want to keep every row in df2, then we can specify how = "right".

```
pd.merge(df1, df2, how='right', on ='b')
```



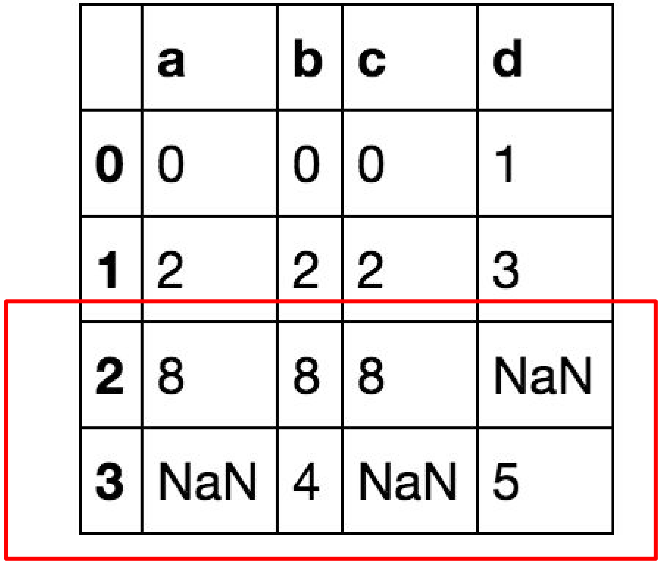
	a	b	c	d
0	0	0	0	1
1	2	2	2	3
2	NaN	4	NaN	5

- Since df1 does not have the row with b=4, pandas leaves NaN for columns a and c.

Data manipulation in pandas: merge

- If we want to keep all rows from both df1 and df2, then we can specify how = "outer".

```
pd.merge(df1, df2, how='outer', on ='b')
```



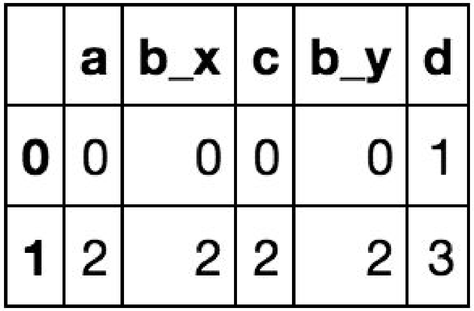
	a	b	c	d
0	0	0	0	1
1	2	2	2	3
2	8	8	8	NaN
3	NaN	4	NaN	5

- All the rows are kept this way.

Data manipulation in pandas: merge

- We can also merge on columns with different names.

```
pd.merge(df1, df2, right_on='b', left_on='a')
```

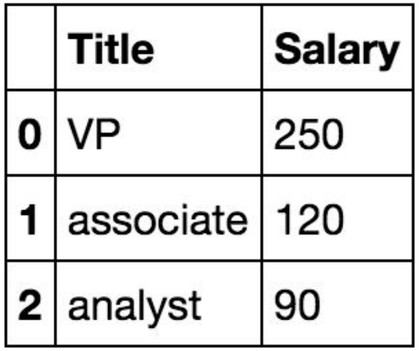


	a	b_x	c	b_y	d
0	0	0	0	0	1
1	2	2	2	2	3

- Since we have a row with a=0 in df1 and a row with b=0 in df2, they are identified. Similarly the row with a=2 in df1 and the row with b=2 in df2 are identified. Since the inner merge is default, and there is no row with a=b=4 nor a=b=8, so those two rows are discarded. Since this time the column b from the two data frames are not identified, there are still two after merging, namely, **b_x** and **b_y**.

In class lab 5: merge

- Run the code provided in iPython notebook to create a data frame, 'Salary'. How is this data frame related to `Employee`? Why do we separate this piece of information into another data frame?
- How should we combine the two data frames in a meaningful way?
  - **Caution:** We mentioned that having the information 'Name' in the index might cause problem when merging. Pay attention to the indices after merging.



	Title	Salary
0	VP	250
1	associate	120
2	analyst	90

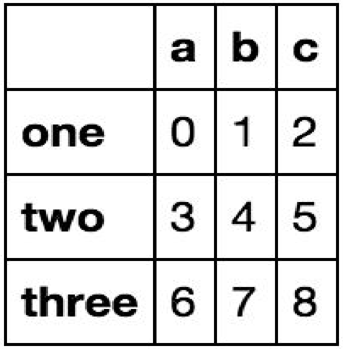
OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

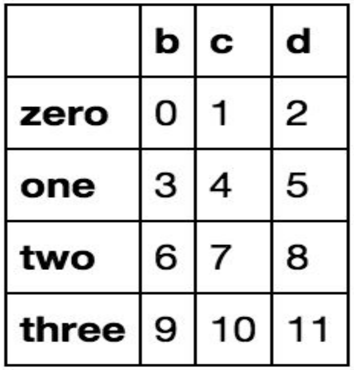
Data manipulation in pandas

- One of the most important pandas features is the behavior of arithmetic between objects with different indices. Let’s create two data frames first.

```
df1 = pd.DataFrame(np.arange(9).reshape((3, 3)),
                   columns=['a', 'b', 'c'],
                   index=['one', 'two', 'three'])
df2 = pd.DataFrame(np.arange(12).reshape((4, 3)),
                   columns=['b', 'c', 'd'],
                   index=['zero', 'one', 'two', 'three'])
```



	a	b	c
one	0	1	2
two	3	4	5
three	6	7	8

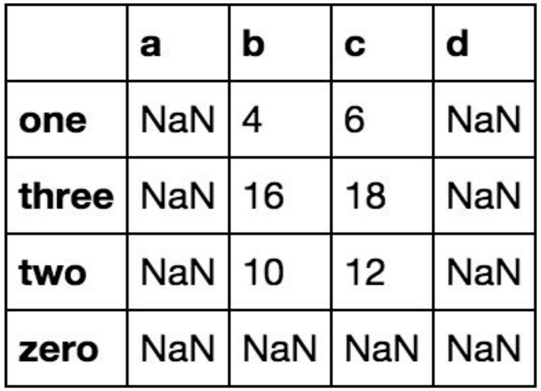


	b	c	d
zero	0	1	2
one	3	4	5
two	6	7	8
three	9	10	11

Data manipulation in pandas: arithmetic

- We can easily add df1 and df2 by using +.

```
df3 = df1 + df2
df3
# returns a DataFrame whose index and columns are
# the unions of the ones in each DataFrame
```



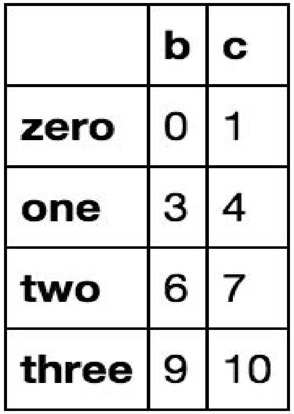
	a	b	c	d
one	NaN	4	6	NaN
three	NaN	16	18	NaN
two	NaN	10	12	NaN
zero	NaN	NaN	NaN	NaN

- Oops! Seems like the concatenation operations produces some NaNs. We will see how to fix it later.

Data manipulation in pandas: drop

- The *drop* method can be used to drop some columns and rows.

```
# drop column 'd'
# axis = 1 means drop column
df2 = df2.drop('d', axis=1)
df2
```

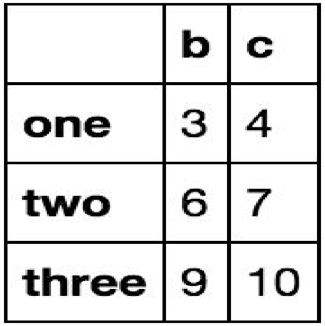


	b	c
zero	0	1
one	3	4
two	6	7
three	9	10

Data manipulation in pandas: drop

- If we set the *axis* parameter to 0, we will delete the specific row instead of the column.

```
# drop row 'zero'
# axis = o means drop row
df2 = df2.drop('zero', axis=0)
df2
```



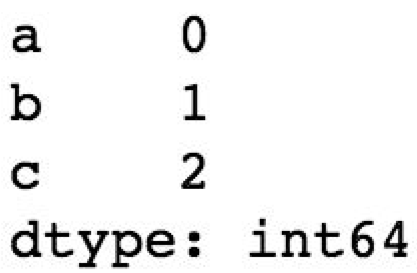
	b	c
one	3	4
two	6	7
three	9	10

Data manipulation in pandas: apply

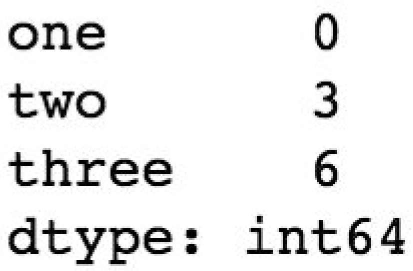
- DataFrame's *apply* method applies a function on 1D arrays to each column or row.

```
df1.apply(min, axis=0)
# minimum number in each column
```

```
df1.apply(min, axis=1)
# minimum number in each row
```



```
a      0
b      1
c      2
dtype: int64
```

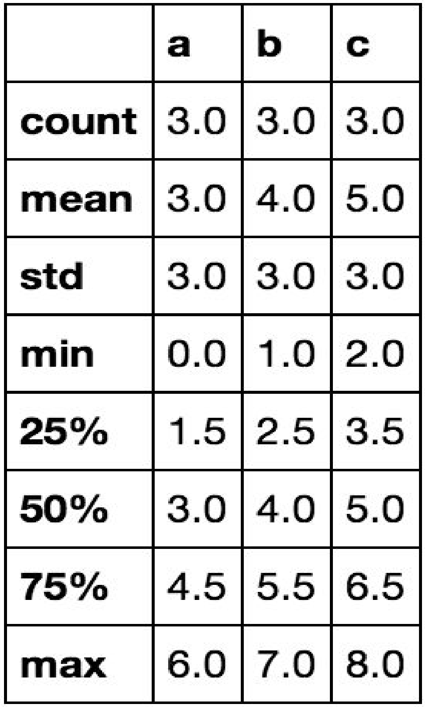


```
one      0
two      3
three    6
dtype: int64
```

Data manipulation in pandas: describe

- The *describe* method computes a set of summary statistics for a Series or for each data frame column.

```
df1.describe()
```



	a	b	c
count	3.0	3.0	3.0
mean	3.0	4.0	5.0
std	3.0	3.0	3.0
min	0.0	1.0	2.0
25%	1.5	2.5	3.5
50%	3.0	4.0	5.0
75%	4.5	5.5	6.5
max	6.0	7.0	8.0

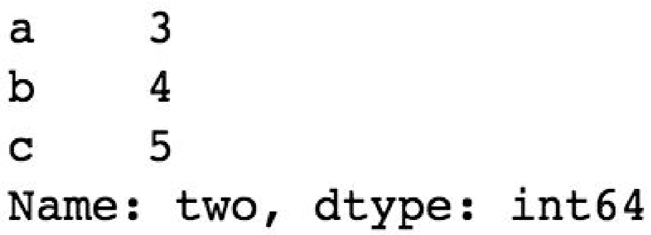
OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- Handling missing values

Data manipulation in pandas: selection

- The *loc* method provides purely label (index/columns)-based indexing. This method only allows you do selection from a data frame by its index and columns. For example:

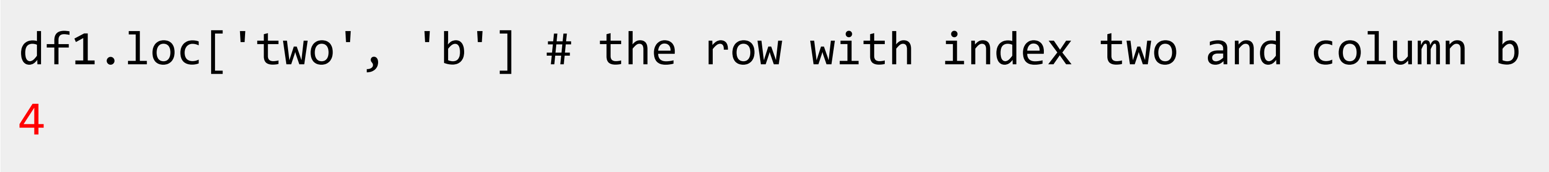
```
df1.loc['two'] # the row that has index two
```



```
a     3
b     4
c     5
Name: two, dtype: int64
```

Data manipulation in pandas: selection

- You can also pass a second parameter to *loc* to specify which column you want to choose. For example:

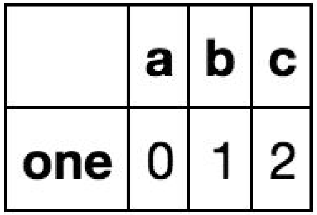


```
df1.loc['two', 'b'] # the row with index two and column b
4
```

Data manipulation in pandas: filter

- Fancy indexing as in Numpy can be done with *loc* in pandas as well. We may select a row with a condition:

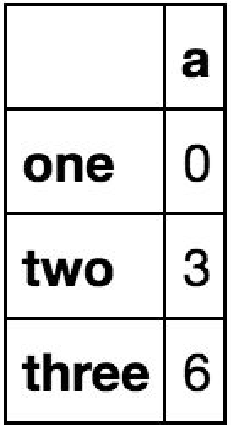
```
df1.loc[df1.a==0,:]
```



	a	b	c
one	0	1	2

- We may select columns in a similar way:

```
df1.loc[:,df1.loc['one']==0]
```



	a
one	0
two	3
three	6

Data manipulation in pandas: selection

- Note: loc only accepts labels as input. If you try to use numbers, it will give you an error. For example:

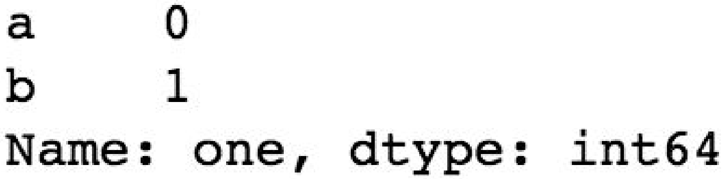
```
df1.loc[1, 2]
KeyError: 'the label [1] is not in the [index]'
```

Data manipulation in pandas: selection

- If you want to select data by number, you need the help of *iloc*. The *iloc* method provides a purely position based indexing.

```
df1.iloc[1, 2]
# select as a matrix
# row 2, col 3
5
```

```
# first row, first two columns
# return a Series
row1 = df1.iloc[0, :2]
row1
```



```
a     0
b     1
Name: one, dtype: int64
```

In class lab 6: More on Data Manipulation

- From the *df1* we created, what should we do if we want to access the elements greater than 4?
  - **Remark:** Is it possible to keep the data frame structure after filtering?
- Give VPs a 5% raise!
- Apply the method `describe` on Employee. How many columns are there? Why?
- Find the sum of the two columns Salary and Year.
- For each row, sum up the Salary and Year.

OVERVIEW

- Data Structure
- Series
- Data Frame
  - I/O tool
- Data Manipulation in Pandas
  - concat
  - merge
- More on Data Manipulation
  - arithmetic, drop, apply and describe
  - selection and filter
- **Handling missing values**

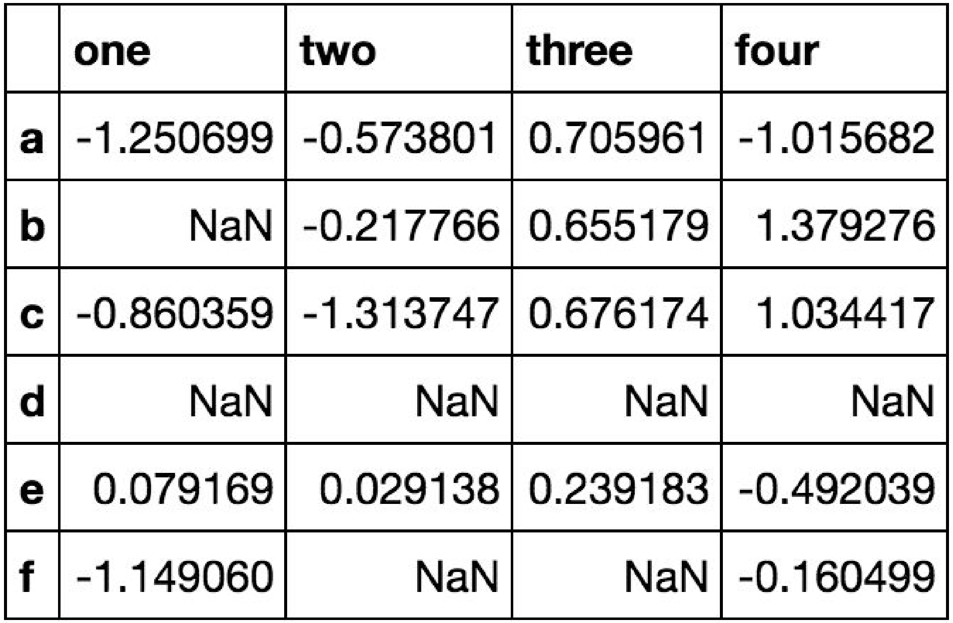
Handling missing data

- Missing - or, what amounts to the same thing, corrupt - data is an unavoidable fact of life in dealing with large quantities of data. There are many ways of dealing with it, depending upon the circumstances:
  - Discard it, and all related data.
  - Interpolate values from surrounding data
  - Isolate it and analyze it separately
- Whatever approach is chosen - and this is a scientific, not a computational, question - pandas has methods to make it simpler to carry out.

Handling missing data

- FIrst, let’s read a csv file that contains NaNs. Note here we set *index_col* to 0 which means we are using the first column as the index.

```
df = pd.read_csv('missing.csv', index_col = 0)
df
```

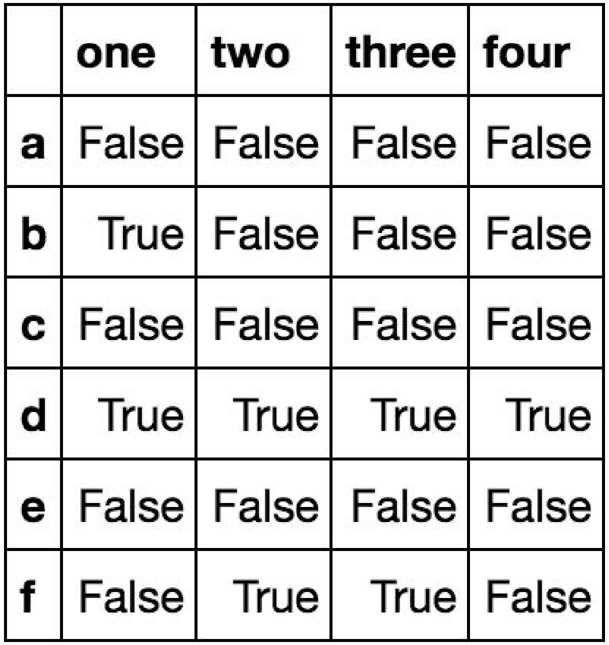


	one	two	three	four
a	-1.250699	-0.573801	0.705961	-1.015682
b	NaN	-0.217766	0.655179	1.379276
c	-0.860359	-1.313747	0.676174	1.034417
d	NaN	NaN	NaN	NaN
e	0.079169	0.029138	0.239183	-0.492039
f	-1.149060	NaN	NaN	-0.160499

Handling missing data: isnull

- If we have no idea about what the dataset looks like, the first thing we want to do is to figure out where the missing data is. We can use the *isnull* method.

```
df.isnull()
```

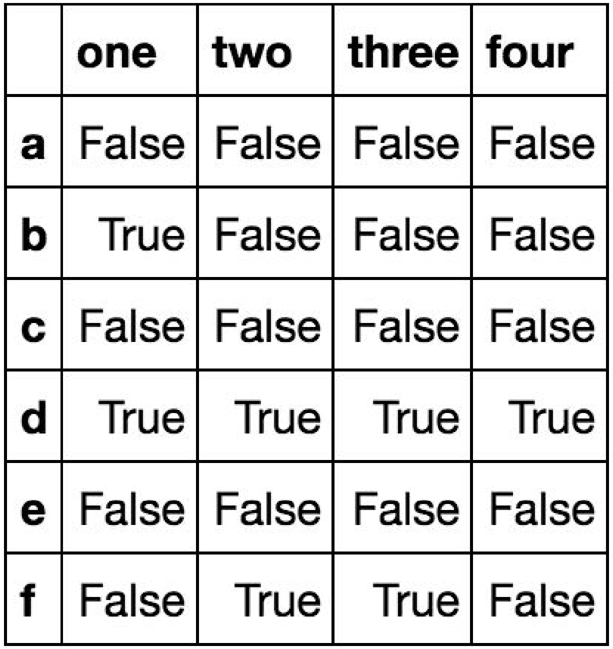


	one	two	three	four
a	False	False	False	False
b	True	False	False	False
c	False	False	False	False
d	True	True	True	True
e	False	False	False	False
f	False	True	True	False

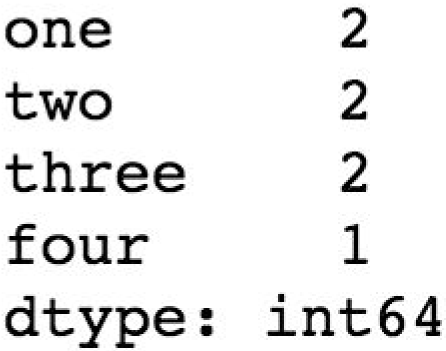
Handling missing data: isnull

- Also we can sum up the boolean array to see how many missing values each column has:

```
np.sum(df.isnull())
```



	one	two	three	four
a	False	False	False	False
b	True	False	False	False
c	False	False	False	False
d	True	True	True	True
e	False	False	False	False
f	False	True	True	False

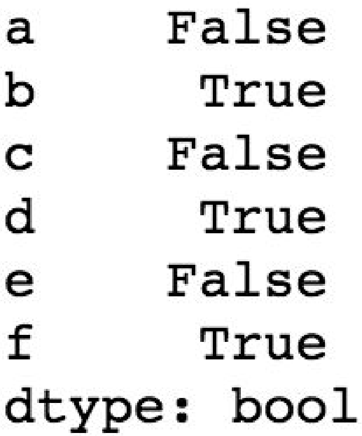


```
one      2
two      2
three    2
four     1
dtype: int64
```

Handling missing data: isnull

- Sometimes we need a close look at those NaNs, so we want to find which rows contain NaNs. To do that , we aggregate the data frame with boolean value, `df.isnull()`, by the function any. *axis=1* indicates rows.

```
df.isnull().any(axis=1)
```

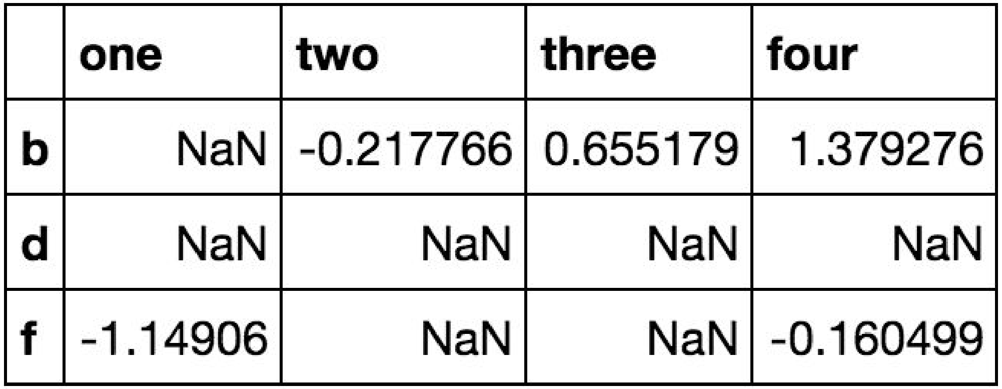


```
a        False
b         True
c        False
d         True
e        False
f         True
dtype: bool
```

Handling missing data: isnull

- Passing the boolean Series to the first position of the *loc* method of the data frame selects the rows:

```
df.loc[df.isnull().any(axis=1),:]
```



	one	two	three	four
b	NaN	-0.217766	0.655179	1.379276
d	NaN	NaN	NaN	NaN
f	-1.14906	NaN	NaN	-0.160499

In class lab 7: Handling Missing Values

- We now deal with the missing values. `Employee` is a very small data frame, so let's just print it out; how many missing values do we have? **Remark:** Some of the missing values are not in the form you might expect.
- Now, we learn that Amy works for a department called 'Trade'. Fill it in.
- Look up the replace method to replace the empty strings in the Employee data frame by *np.nan*. Make sure you update the data frame.

In class lab 7: Handling Missing Values

- We have now replaced all missing values that weren’t NaNs by NaNs. Write code to find out how many NaNs we have in each row. In each column?
- Print the rows with NaNs.
- Print the columns with NaNs.