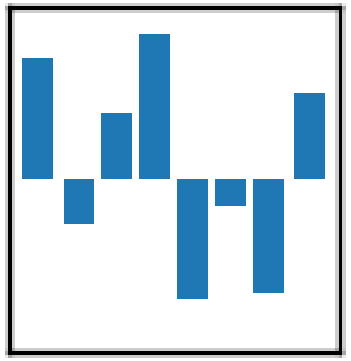
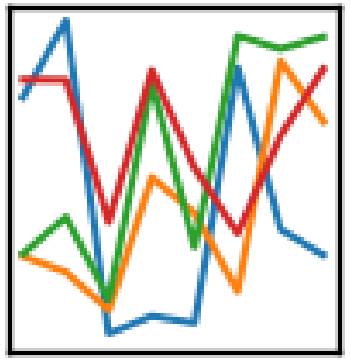
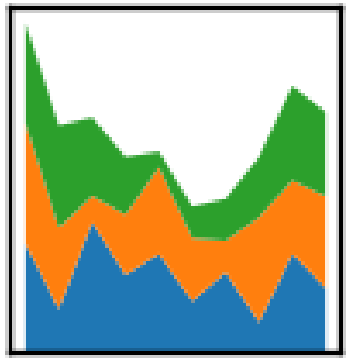
pandas

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







Topic 3 – Data Analysis with Pandas

https://pandas.pydata.org/docs/user_guide

Contents

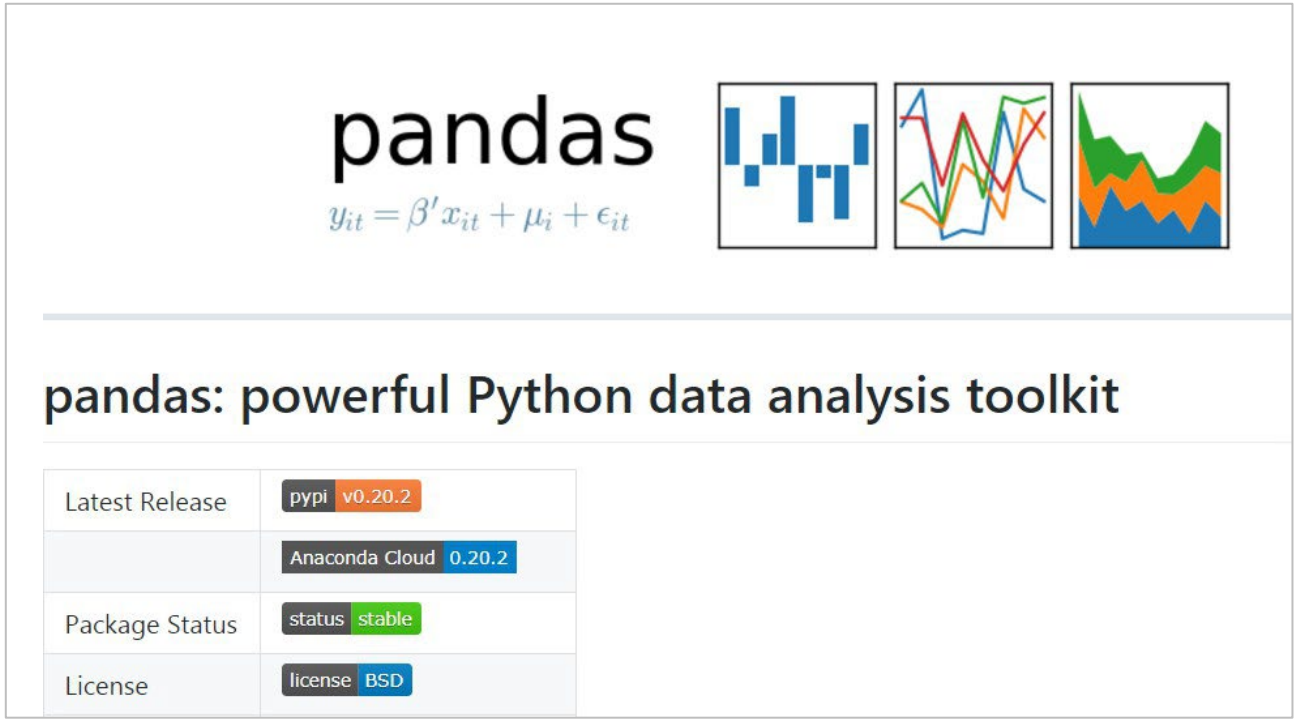
- Intro to Pandas
- A first taste of Pandas
- Pandas Data Structures
  - Series, DataFrames
- Loading and Saving Data
  - Read/Write CSV, Write Excel, SQL
- Retrieving Information
  - Basic Information, Summary
- Selections
  - Subsetting rows, Subsetting columns
- Reshaping Data
  - Dropping values, Merging, pivot, unpivot, sorting, reindexing
- Handling Missing Data
  - Dropping Missing Data, Filling Missing Data
- Group Data
  - Groupby
- Applying Functions
  - apply, lambda

Intro to Pandas

What is Pandas?

- Pandas is a high-performance open source library for data analysis in Python
- De-facto standard library for data analysis using Python

- Widespread adoption of the tool with a large community behind it (817 contributors + 15,330 commits as in Jul 2017)
- Rapid iteration, features, and enhancements continuously made
- https://github.com/pandas-dev/pandas



Key Features of Pandas

- Can process a variety of data sets in different formats: tabular heterogeneous, time series, and matrix data.
- Can load /import data from varied sources such as CSV, Excel and DB/SQL.
- Allows manipulation of datasets such as subsetting, slicing, filtering, merging, groupBy, re-ordering, and re-shaping.
- Allows handling of missing data according to rules defined by the user/developer: ignore, convert to 0, and so on.
- It can be used for parsing and munging (conversion) of data as well as modeling and statistical analysis.
- It integrates well with other Python libraries such as statsmodels, SciPy, and scikit-learn.
- It delivers fast performance and can be speeded up even more by making use of Cython (C extensions to Python).

What problem does pandas solve?

- Python has long been great for data munging and preparation, but less so for data analysis and modelling
- pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R
- pandas does not implement significant modeling functionality outside of linear and panel regression; for this, look to statsmodels and scikit-learn.

A first taste of Pandas

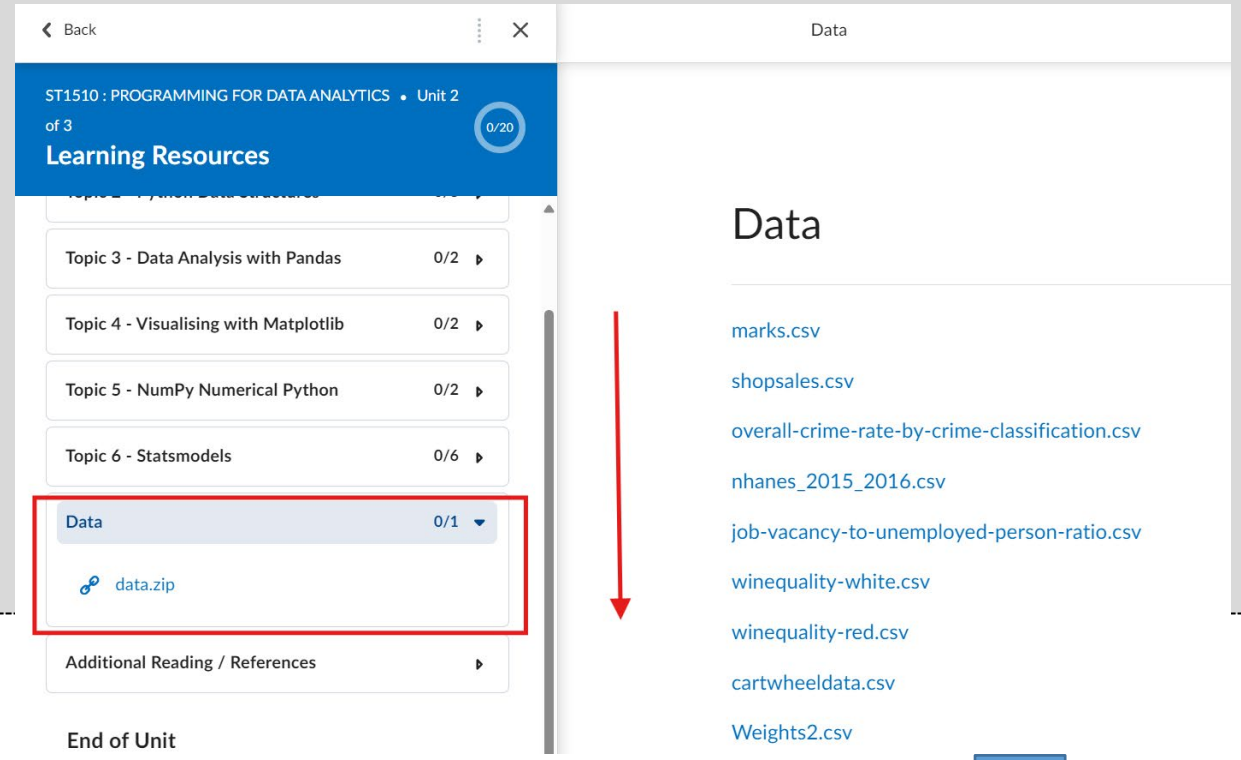
Objectives

This section will show you a simple Pandas program that you can learn techniques to:

1. Load a simple CSV file
2. Show a preview of the first n rows and last n rows of the loaded dataset
3. Display information about the loaded dataset such as:
    1. The count of how many rows and columns were loaded
    2. The column names of each row and their datatypes
4. Extract subsets of the dataset
5. Save a subset of data

Load a simple delimited file

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv', sep=',')
```



Show first/last n rows of dataset

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv', sep=',')

#  Get the 5 rows of the dataset
df.head(5)

#  Get the last 3 rows of the dataset
df.tail(3)
```

	Season	Nation	Winners	Score	Runners-up	Runner-UpNation	Venue	Attendance
0	1955–56	Spain	Real Madrid	4–3	Stade de Reims	France	Parc des Princes,Paris	38239
1	1956–57	Spain	Real Madrid	2–0	Fiorentina	Italy	Santiago Bernabéu Stadium, Madrid	124000
2	1957–58	Spain	Real Madrid	3–2	Milan	Italy	Heysel Stadium,Brussels	67000
3	1958–59	Spain	Real Madrid	2–0	Stade de Reims	France	Neckarstadion,Stuttgart	72000
4	1959–60	Spain	Real Madrid	7–3	Eintracht Frankfurt	Germany	Hampden Park,Glasgow	127621

	Season	Nation	Winners	Score	Runners-up	Runner-UpNation	Venue	Attendance
55	2010–11	Spain	Barcelona	3–1	Manchester United	England	Wembley Stadium,London	87695
56	2011–12	England	Chelsea	1–1*[K]	Bayern Munich	Germany	Allianz Arena,Munich	62500
57	2012–13	Germany	Bayern Munich	2–1	Borussia Dortmund	Germany	Wembley Stadium,London	86298

Count how many cols and rows loaded

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv', sep=',')

print(df.shape)
```

```
(58, 8)
```


Print out the columns and their datatypes

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv', sep=',')

print(df.dtypes)
```

```
Season              object
Nation              object
Winners             object
Score               object
Runners-up          object
Runner-UpNation     object
Venue               object
Attendance           int64
dtype: object
```

Extract subsets of the dataset

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv', sep=',')

#  Extract only the three columns
df[['Season', 'Nation', 'Winners']]

#  Extract only rows with attendance more than 100,000
df2 = df[df['Attendance'] > 100000]
df2
```

	Season	Nation	Winners
0	1955–56	Spain	Real Madrid
1	1956–57	Spain	Real Madrid
2	1957–58	Spain	Real Madrid
3	1958–59	Spain	Real Madrid
4	1959–60	Spain	Real Madrid
5	1960–61	Portugal	Benfica
6	1961–62	Portugal	Benfica

	Season	Nation	Winners	Score	Runners-up	Runner-UpNation	Venue	Attendance
1	1956–57	Spain	Real Madrid	2–0	Fiorentina	Italy	Santiago Bernabéu Stadium, Madrid	124000
4	1959–60	Spain	Real Madrid	7–3	Eintracht Frankfurt	Germany	Hampden Park,Glasgow	127621

Save a subset of the data

```
import pandas as pd

Df = pd.read_csv('data/euro_winners.csv', sep=',')

#  Extract only rows with population more than 100 thousands
df2 = df[df['Attendance'] > 100000]

df2.to_csv('goodattendance.csv')
```


Loading / Saving Data

Loading Data

- Pandas provides methods to load data from a variety of sources

Loading data from csv, Excel, HTML, json, SQL
csv_dataframe = pd.read_csv('my_dataset.csv', sep=',')
xls_dataframe = pd.read_excel('my_dataset.xlsx', 'Sheet1', na_values=['NA', '?'])
table_dataframe= pd.read_html('http://page.com/with/table.html')[0]
json_dataframe = pd.read_json('my_dataset.json', orient='columns')
from sqlalchemy import create_engine engine = create_engine('sqlite:///:memory:') sql_dataframe = pd.read_sql_table('my_table', engine, columns=['ColA', 'ColB']) sql_dataframe = pd.read_sql("SELECT * FROM my_table;", engine)

Saving Data

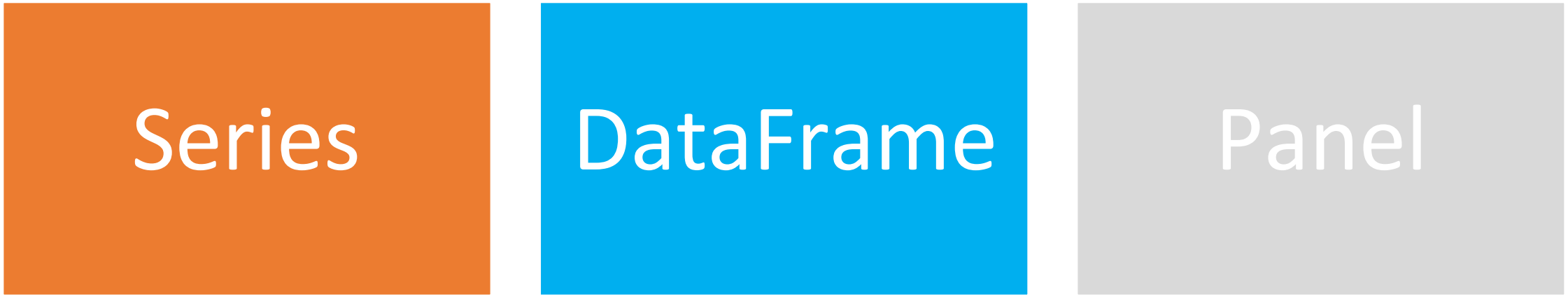
- Similarly, pandas provides methods to save data to a variety of sources

Saving data to csv, Excel, HTML, json, SQL
my_dataframe.to_csv('dataset.csv')
my_dataframe.to_excel('dataset.xlsx')
html_code = my_dataframe.to_html()
my_dataframe.to_json('dataset.json')
from sqlalchemy import create_engine engine = create_engine('sqlite:///:memory:') sql_dataframe.to_sql('my_table', engine)

Pandas Data Structures

Overview

- There are three main data structures in pandas
- In this module, we will cover **Series** and **DataFrame** only



Series

- Series is a **one-dimensional** labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects etc)
- The axis labels are collectively referred to as the **index**
- The basic method to create a Series is as follows:-

```
import pandas as pd
ser = pd.Series(data, index=idx)
```

- ***data*** can be many different things: a Python dict, an ndarray, a scalar value (like 5)
- ***index*** is a list of axis labels and are initialized differently depending on what the nature of data is (see next slides for examples)

Creating Series (using Python dictionary)

- If data is a dict, if index is passed the values in data corresponding to the labels in the index will be pulled out
- Otherwise, an index will be constructed from the sorted keys of the dict, if possible.

```
import pandas as pd

currencyDict ={
'US':'dollar', 'UK':'pound',
'Mexico':'peso', 'China':'yuan'}

currencySeries = pd.Series(currencyDict);
```

```
China      yuan
Mexico     peso
UK        pound
US       dollar
dtype: object
```

index

Creating Series (Using numpy.ndarray)

- If data is an ***ndarray***, index must be the same length as data
- If no index is passed, one will be created having values [0, ..., len(data) - 1].

```
import pandas as pd
import numpy as np

s = pd.Series(np.random.randn(5),
              index=['a', 'b', 'c', 'd', 'e'])
```

```
a   -0.231872
b    0.207976
c    0.935808
d    0.179578
e   -0.577162
```


Creating Series (using scalar values)

- If data is a scalar value, an index must be provided
- The value will be repeated to match the length of index

```
import pandas as pd

pd.Series(5,
index=['a', 'b', 'c', 'd', 'e'])
```

```
a    5
b    5
c    5
d    5
e    5
dtype: int64
```

DataFrame

- **DataFrame** is the most commonly used data structure in pandas
- DataFrame is a **2-dimensional** labeled data structure
- You can think of it like a spreadsheetor SQL table, or a dict of Series object
- The constructor accepts many different types of arguments
  - 2D NumPy array
  - Dictionary of 1D NumPy array or lists, dictionaries, or Series structures
  - Structured or record ndarray
  - Series structures
  - Another DataFrame structure

DataFrame

- The basic method to create a DataFrame is as follows:-

```
import pandas as pd
df = pd.DataFrame(data,index,columns)
```

- ***data*** : numpy ndarray, dict, Series or DataFrame
- ***index*** : Index to use for resulting frame. Default to np.arange(n)
- ***columns*** : Column labels to use for resulting frame. Default to np.arange(n)

Create DataFrame (Using 2d Numpy Array)

```
import pandas as pd
import numpy as np

# data is a numpy 10x5 ndarray
# of random numbers
data = np.random.randn(10, 5)
df = pd.DataFrame(data)

print(type(data))
print(type(df))
print(df)
```

```
import pandas as pd
import numpy as np

data = np.random.randn(10, 5)
df = pd.DataFrame(data)
print(type(data))
print(type(df))
print(df)
```

```
<class 'numpy.ndarray'>
<class 'pandas.core.frame.DataFrame'>
          0         1         2         3         4
0 -0.846432  2.010771 -0.871403  0.239019  0.822589
1  0.700545  0.234069  1.601556 -0.588629 -1.208184
2 -0.240252 -1.622228 -0.161383 -0.034586 -0.699763
3  0.117042  0.287502 -0.122274 -0.345347  1.157136
4  0.822581  0.212825 -1.487858  0.324663  2.283243
5 -0.986751  0.092294  0.233702 -1.779424 -0.965550
6  0.818944  0.671896  0.748318  1.035752  0.160825
7 -0.312808  0.352137 -1.066168  1.189180  0.539905
8 -1.814234  0.619441 -0.274704  0.416552 -0.284880
9  0.037882  0.031949 -0.480216 -0.320649  1.094990
```

Create DataFrame (dict of 1d Numpy array)

```
import pandas as pd
import numpy as np

np1 = np.array([1,2,3])
np2 = np.array([4,5,6])
np3 = np.array([7,8,9])

d = {'one' : np1,
     'two' : np2,
     'three': np3}

df1 = pd.DataFrame(d)
df2 = pd.DataFrame(d,
        index=['a', 'b', 'c'])
```

In this case, the column names are derived automatically from the dict keys

```
import pandas as pd
import numpy as np

np1 = np.array([1,2,3])
np2 = np.array([4,5,6])
np3 = np.array([7,8,9])

d = {'one' : np1,'two' : np2, 'three': np3}

df1 = pd.DataFrame(d)
df2 = pd.DataFrame(d, index=['a', 'b', 'c'])

print(df1)
print(df2)
```

```
   one  three  two
0    1      7    4
1    2      8    5
2    3      9    6
   one  three  two
a    1      7    4
b    2      8    5
c    3      9    6
```


Create DataFrame (Using 2d Numpy Array)

```
import pandas as pd
import numpy as np

data = np.random.randn(5, 3)
df = pd.DataFrame(data,
                  columns=['i','ii','iii'],
                index=['a','b','c','d','e']
     )
```

You can specify the column names and index references when creating the DataFrame object

```
import pandas as pd
import numpy as np

data = np.random.randn(5, 3)
df = pd.DataFrame(data,
                        columns=['i','ii','iii'],
                        index=['a','b','c','d','e'])
df
```

	i	ii	iii
a	0.835674	1.563080	0.101055
b	-0.657617	0.863878	0.989823
c	0.637169	-0.971556	-1.322144
d	0.533673	0.089607	0.666361
e	-0.542542	-0.972537	-1.055490

Create DataFrame (dict of lists)

```
import pandas as pd

d = {'weights' :[50,70.5,85.3,43.1],
     'heights' : [1.54,1.73,1.82,1.6]}

df1 = pd.DataFrame(d)
df2 = pd.DataFrame(d,
      index=['Mary', 'John',
             'Robert', 'Christine'])
```

In this case, the column names are derived automatically from the dict keys

	heights	weights
Mary	1.54	50.0
John	1.73	70.5
Robert	1.82	85.3
Christine	1.60	43.1

Create DataFrame (dict of Series)

```
import pandas as pd

d = {
     'one' : pd.Series([1, 2, 3],
     index=['a', 'b', 'c']),

     'two' : pd.Series([1, 2, 3, 4],
     index=['a', 'b', 'c', 'd'])
}

df = pd.DataFrame(d)
```

In this case, the column names are derived automatically from the dict keys

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

Create DataFrame (dict of Series)

```
import pandas as pd

stockSummaries={
'AMZN':pd.Series([346.15, 589.8,158.88],
     index=['Closing price','P/E','Market Cap(B)']),
'GOOG':pd.Series([1133.43, 380.64],
     index=['Closing price','Market Cap(B)']),
'FB':pd.Series([61.48, 150.92],
      index=['Closing price','Market Cap(B)'])}

stockDF = pd.DataFrame(stockSummaries)
```

	AMZN	FB	GOOG
Closing price	346.15	61.48	1133.43
Market Cap(B)	158.88	150.92	380.64
P/E	589.80	NaN	NaN

In this case, the column names are derived automatically from the dict keys

Create DataFrame (structured record/array)

```
import pandas as pd
data = [(1,2,'Hello'), (2,3,"World")]

df = pd.DataFrame(data)
df2 = pd.DataFrame(data,
                   index=['first', 'second'],
                   columns=['a','b','c'])
```

df

	0	1	2
0	1	2	Hello
1	2	3	World

df2

	a	b	c
first	1	2	Hello
second	2	3	World

Create DataFrame (list of dicts)

```
import pandas as pd

data = [{'a': 1, 'b': 2},
        {'a': 5, 'b': 10, 'c': 20}
       ]

df = pd.DataFrame(data)
```

df

	a	b	c
0	1	2	NaN
1	5	10	20.0

In this case, the column names are derived automatically from the dict keys

Retrieving Information

Basic Information

Method	Description
df.shape	Returns (rows, columns)
df.index	Describe index
df.columns	Describe DataFrame column s
df.count()	Number of non-NA values
df.info()	Info on DataFrame
df.dtypes	Data types

Basic Information

df.shape

```
In [2]: import pandas as pd
        df = pd.read_csv('data/gapminder.tsv', sep='\t')
        df.shape

Out[2]: (1704, 6)
```

df.index

```
In [3]: import pandas as pd
        df = pd.read_csv('data/gapminder.tsv', sep='\t')
        df.index

Out[3]: RangeIndex(start=0, stop=1704, step=1)
```

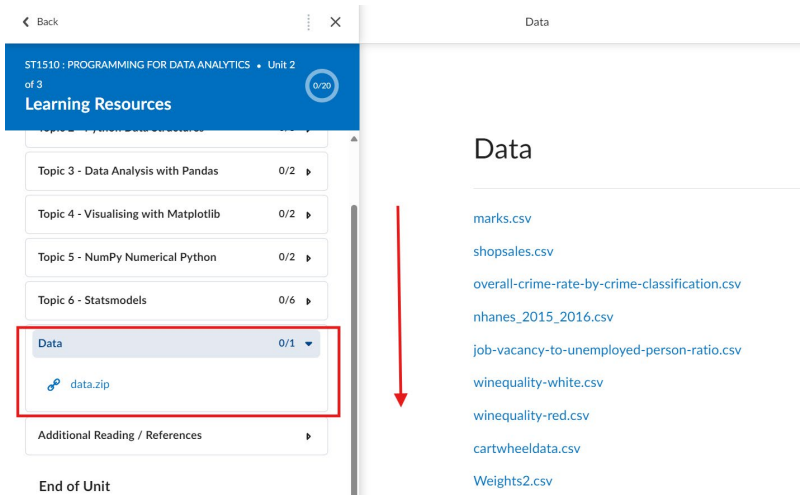
Basic Information

```
In [4]: import pandas as pd
        df = pd.read_csv('data/gapminder.tsv', sep='\t')
        df.columns

Out[4]: Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

df.columns

```
In [6]: import pandas as pd
        df = pd.read_csv('data/gapminder.tsv', sep='\t')
        df.count()

Out[6]: country      1704
        continent    1704
        year         1704
        lifeExp      1704
        pop          1704
        gdpPercap    1704
        dtype: int64
```

df.count



Basic Information

```
In [5]: import pandas as pd
        df = pd.read_csv('data/gapminder.tsv', sep='\t')
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
country      1704 non-null object
continent    1704 non-null object
year         1704 non-null int64
lifeExp      1704 non-null float64
pop          1704 non-null int64
gdpPercap    1704 non-null float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
```

df.info

Basic Information

df.dtypes

```
In [1]:
1 import pandas as pd
2 df = pd.read_csv('data/gapminder.tsv',sep='\t')
3 df.dtypes
```

```
Out[1]: country      object
        continent    object
        year          int64
        lifeExp     float64
        pop           int64
        gdpPercap   float64
        dtype: object
```

Summary

Method	Description
df.sum()	Sum of values
df.cumsum()	Cumulative sum of values
df.min(), df.max()	Minimum / Maximum values
df.idxmin(), df.idxmax()	Minimum / Maximum Index Value
df.describe	Summary Statistics
df.mean()	Mean of values
df.median()	Median of values

Summary

Original dataset - df

In [11]:
```
import pandas as pd
df = pd.read_csv('data/Weights.csv')
df
```

Out[11]:

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

Summary

df.sum

```
In [9]: import pandas as pd
        df = pd.read_csv('data/Weights.csv')
        df['weight'].sum()

Out[9]: 328.8
```

df.cumsum

```
In [8]: import pandas as pd
        df = pd.read_csv('data/Weights.csv')
        df['weight'].cumsum()

Out[8]: 0     45.5
        1    110.5
        2    160.5
        3    235.5
        4    285.5
        5    328.8
        Name: weight, dtype: float64
```

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

Summary

df.min()

```
In [10]: import pandas as pd
         df = pd.read_csv('data/Weights.csv')
         df['weight'].min()

Out[10]: 43.299999999999997
```

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

df.max()

```
In [12]: import pandas as pd
         df = pd.read_csv('data/Weights.csv')
         df['weight'].max()

Out[12]: 75.0
```


Summary

df.idxmin()

df.idxmax()

```
: 1 df = pd.read_csv('classdata/weights.csv', sep=',')
  2 print(df)
  3 print()
  4
  5 print(df['weight'].idxmin())
  6 print(df['weight'].idxmax())
```

```
   observation  weight gender
0            1    45.5      f
1            2    65.0      m
2            3    50.0      f
3            4    75.0      m
4            5    50.0      m
5            6    43.3      f

5
3
```


Summary

df.describe()

```
In [17]: import pandas as pd
         df = pd.read_csv('data/Weights.csv')
         df['weight'].describe()

Out[17]: count      6.000000
         mean      54.800000
         std       12.465151
         min       43.300000
         25%       46.625000
         50%       50.000000
         75%       61.250000
         max       75.000000
         Name: weight, dtype: float64
```

Summary

df.mean()

```
In [13]: import pandas as pd
         df = pd.read_csv('data/Weights.csv')
         df['weight'].mean()

Out[13]: 54.800000000000004
```

df.median()

```
In [16]: import pandas as pd
         df = pd.read_csv('data/Weights.csv')
         df['weight'].median()

Out[16]: 50.0
```

Subsetting columns

Method	Description
df['width'] or df.width	Subset a single column by column name
df[['width','length','species']]	Subset multiple columns by column names
df.loc[:, 'A':'C']	Subset a range of columns by column names using **loc**
df.iloc[:,2]	Subset a single column by its index using **iloc**
df.iloc[:, [0, -1]]	Subset multiple columns by their indices using **iloc**
df.iloc[:, 0:2]	Subset a range of columns by index using **iloc**
df.loc[:,'pop'] > 100000	Create derived columns by using Boolean logic
re = '^customer' df.filter(regex=re)	Subset columns whose names match a regular expression, e.g. where data contains 'customer'

Subsetting columns (by column name)

```
Index(['Season', 'Nation', 'Winners', 'Score', 'Runners-up', 'Runner-UpNation',
       'Venue', 'Attendance'],
      dtype='object')
```

```
import pandas as pd

df = pd.read_csv('data/euro_winners.csv',
                 sep=',')
print(df.columns)

Subset a single column by column name
df1 = df['Season']
df2 = df.Season

# Subset multiple columns by column names
df3 = df[['Season', 'Winners', 'Score']]

# Subset a range of columns using loc
df4 = df.loc[:, 'Season':'Score']
```

df3

	Season	Winners	Score
0	1955–56	Real Madrid	4–3
1	1956–57	Real Madrid	2–0
2	1957–58	Real Madrid	3–2
3	1958–59	Real Madrid	2–0
4	1959–60	Real Madrid	7–3
5	1960–61	Benfica	3–2
6	1961–62	Benfica	5–3
7	1962–63	Milan	2–1

df4

	Season	Nation	Winners	Score
0	1955–56	Spain	Real Madrid	4–3
1	1956–57	Spain	Real Madrid	2–0
2	1957–58	Spain	Real Madrid	3–2
3	1958–59	Spain	Real Madrid	2–0
4	1959–60	Spain	Real Madrid	7–3

Subsetting columns (by index)

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')

# Subset a single column by its index
df0 = df.iloc[:,0]

# Subset multiple columns by their indices
df2 = df.iloc[:,[0,-1]]

# Subset a range of columns using iloc
df3 = df.iloc[:, 0:3]
```

Subsetting columns by index

df2

	Season	Attendance
0	1955–56	38239
1	1956–57	124000
2	1957–58	67000
3	1958–59	72000
4	1959–60	127621

df3

	Season	Nation	Winners
0	1955–56	Spain	Real Madrid
1	1956–57	Spain	Real Madrid
2	1957–58	Spain	Real Madrid
3	1958–59	Spain	Real Madrid
4	1959–60	Spain	Real Madrid

Subsetting columns (by boolean logic)

```
import pandas as pd

df = pd.read_csv('gapminder.tsv', sep='\t')

# Boonlean expression for more than
mask = df.loc[:,'Attendance'] > 100000

df[mask][['Season', 'Venue', 'Attendance']]
```

	Season	Venue	Attendance
1	1956–57	Santiago Bernabéu Stadium, Madrid	124000
4	1959–60	Hampden Park,Glasgow	127621

Subsetting columns by Boolean logic

Subsetting columns (by reg expression)

```
import pandas as pd

df = pd.read_csv('classdata/euro_winners.csv', sep=',')

re1 = 'er' # Match strings containing a 'er'
re2 = 'on$' # Match strings ending with 'on'
re3 = '^S' # Match strings starting with S
re4 = '[n]+' #Match strings with at least one 'n' (one or more 'n')

df2 = df.filter(regex=re1)
```

Subsetting columns by regular expressions

Subsetting rows

Method	Description
df.loc['a':'c']	Select rows by label
df.iloc[10:20,:]	Select rows by index
df[df.Length > 7]	Select rows by Boolean logic
df.head(n)	Select first *n* rows
df.tail(n)	Select last *n* rows
df.sample(frac=0.5)	Randomly select fraction of rows.
df.sample(n=10)	Randomly select *n* rows
df.nlargest(n, 'value')	Select and order top *n* entries
df.nsmallest(n, 'value')	Select and order bottom *n* entries
df.drop_duplicates()	Select unique rows only (duplicates removed)

Subsetting rows (by label)

Subsetting rows by label

```
import pandas as pd

df = pd.read_csv('studentsdataset.csv',
                 index_col='StudentID')

df1 = df.loc[1516045]
df2 = df.loc[[1516045,1532537]]
```

df

	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
StudentID					
1566159	Andrea	Haynes	DVEMG	Female	2A02
1570238	Lynda	Snyder	DDA	Female	2B22
1507382	Jackie	West	DMAT	Female	1B21
1569201	Verna	Houston	DVEMG	Female	2B21
1532537	Catherine	Payne	DISM	Male	3A01
1533390	Leonard	Vasquez	DMAT	Female	2B22

df1

```
StudentFirstName     Lynda
StudentLastName     Snyder
StudentCourse          DDA
StudentGender       Female
StudentClass          2B22
Name: 1570238, dtype: object
```

df2

	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
StudentID					
1570238	Lynda	Snyder	DDA	Female	2B22
1569201	Verna	Houston	DVEMG	Female	2B21

Subsetting rows (by label)

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.randn(6, 4),
                  index=list('abcdef'),
                  columns=list('ABCD'))

df1 = df.loc['c':'f']
```

Subsetting rows by label

	A	B	C	D
a	-0.787947	0.862498	-0.445377	1.029100
b	-0.645390	0.409603	-0.457001	0.373289
c	0.042265	0.847289	-0.514286	1.787222
d	1.177871	0.867652	0.840295	-0.973243
e	0.223607	0.653171	-0.156444	-0.405972
f	1.071169	1.281024	0.038239	1.343728

df

	A	B	C	D
c	0.170510	0.134285	1.810408	0.631950
d	0.243851	-0.666696	0.792390	-0.540487
e	0.205696	1.101297	0.301214	0.995170
f	0.026745	-1.220182	0.021192	-0.843796

df1

Subsetting rows (by index)

Subsetting rows by index

```
import pandas as pd

df = pd.read_csv('studentsdataset.csv')

df1 = df.iloc[2]
df2 = df.iloc[[2,4,6]]
```

df

	StudentID	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
0	1566159	Andrea	Haynes	DVEMG	Female	2A02
1	1570238	Lynda	Snyder	DDA	Female	2B22
2	1507382	Jackie	West	DMAT	Female	1B21
3	1569201	Verna	Houston	DVEMG	Female	2B21
4	1532537	Catherine	Payne	DISM	Male	3A01
5	1533390	Leonard	Vasquez	DMAT	Female	2B22
6	1509165	Lynne	Vega	DVEMG	Female	3A01

df1

```
StudentID            1507382
StudentFirstName      Jackie
StudentLastName         West
StudentCourse           DMAT
StudentGender         Female
StudentClass            1B21
Name: 2, dtype: object
```

df2

	StudentID	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
2	1507382	Jackie	West	DMAT	Female	1B21
4	1532537	Catherine	Payne	DISM	Male	3A01
6	1509165	Lynne	Vega	DVEMG	Female	3A01

Subsetting rows (by boolean logic)

Subsetting rows by Boolean logic

```
import pandas as pd

df = pd.read_csv('studentsdataset.csv')

# Select only rows with col course = DBIT
df1 = df[df.StudentCourse == 'DBIT']
df2 = df[df.StudentCourse.isin(
['DBIT','DIT'])]
```

df

	StudentID	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
0	1566159	Andrea	Haynes	DVEMG	Female	2A02
1	1570238	Lynda	Snyder	DDA	Female	2B22
2	1507382	Jackie	West	DMAT	Female	1B21
3	1569201	Verna	Houston	DVEMG	Female	2B21
4	1532537	Catherine	Payne	DISM	Male	3A01
5	1533390	Leonard	Vasquez	DMAT	Female	2B22
6	1509165	Lynne	Vega	DVEMG	Female	3A01

df1

	StudentID	StudentFirstName	StudentLastName	StudentCourse	StudentGender	StudentClass
15	1516045	Kay	Burton	DBIT	Female	2B21
21	1557545	Mercedes	Doyle	DBIT	Male	1A02
23	1587345	Catherine	Padilla	DBIT	Female	3A01
26	1567542	Miguel	Mcdaniel	DBIT	Male	2A02
39	1598256	Darrel	Phelps	DBIT	Female	3B01

Subsetting rows (by regex matching)

Subsetting rows by regular expression matching

```
import pandas as pd

df = pd.read_csv('rainfall.csv')

# Regular expression - starts with 2017
re2017 = '^2017'
# Extract dataset with months from 2017
df2017 = df[df['month'].str.contains(re2017)]
```

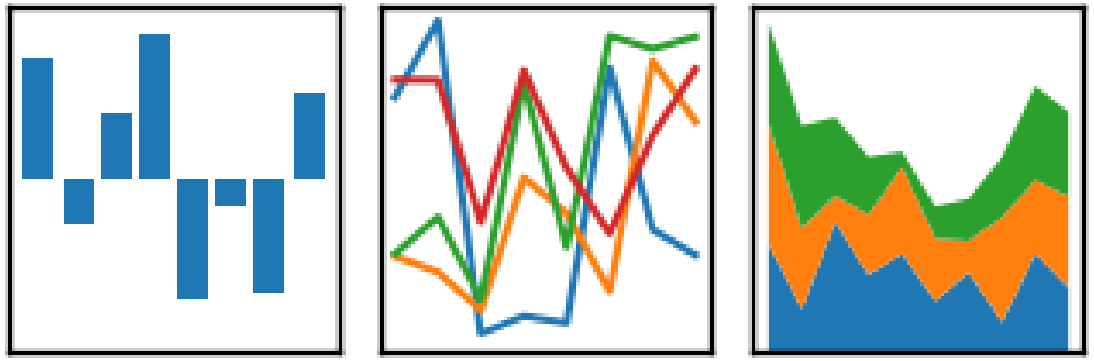
month	total_rainfall
2017-01	197.6
2017-02	158.4
2017-03	136.2
2017-04	208.6
2017-05	190.0
2017-06	106.0
2017-07	79.6
2017-08	84.2
2017-09	124.4

Topic 3 part 2

Data Analysis with Pandas

pandas

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



Reshaping Data

Change the layout of a data set

Change the layout of datasets

Method	Description
df.drop(['Length','Height'], axis=1)	Drop columns from a DataFrame
pd.concat	Append rows to DataFrames, axis=0
pd.concat	Append columns to DataFrames, axis=1
df.pivot	Spread rows into columns
pd.melt	Gather columns into rows
df.sort_values('mpg')	Sort DataFrame by column values
df.sort_index()	Sort DataFrame by index values
df.reindex()	Conform DataFrame to a new index
df.reset_index()	Reset the index of a DataFrame
df.rename(columns = {'y':'year'})	Rename the columns of a DataFrame

Drop columns from a DataFrame (drop)

	Season	Nation	Winners	Score	Runners-up	Runner-UpNation	Venue	Attendance
0	1955–56	Spain	Real Madrid	4–3	Stade de Reims	France	Parc des Princes,Paris	38239
1	1956–57	Spain	Real Madrid	2–0	Fiorentina	Italy	Santiago Bernabéu Stadium, Madrid	124000
2	1957–58	Spain	Real Madrid	3–2	Milan	Italy	Heysel Stadium,Brussels	67000
3	1958–59	Spain	Real Madrid	2–0	Stade de Reims	France	Neckarstadion,Stuttgart	72000
4	1959–60	Spain	Real Madrid	7–3	Eintracht Frankfurt	Germany	Hampden Park,Glas	

df

	Season	Nation	Winners	Score
0	1955–56	Spain	Real Madrid	4–3
1	1956–57	Spain	Real Madrid	2–0
2	1957–58	Spain	Real Madrid	3–2
3	1958–59	Spain	Real Madrid	2–0
4	1959–60	Spain		

df2

```
import pandas as pd
df =
pd.read_csv("data/euro_winners.csv"
)
df2= df.drop(['Runners-up',
              'Runner-UpNation',
'Venue', 'Attendance'], axis=1)
```

Append rows to DataFrames (concat)

```
import pandas as pd
df1 = pd.read_csv("data/Weights1.csv",
                  index_col=['name'])
df2 = pd.read_csv("data/Weights2.csv",
                  index_col=['name'])

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

df1

	weight	gender
name		
Mary	45.5	f
John	65.0	m
Anna	50.0	f
Bryan	75.0	m
Rob	50.0	m
Kay	43.3	f

df2

	weight	gender
name		
Calvin	89.3	m
Peter	55.1	m
Julia	41.1	f

result

	weight	gender
name		
Mary	45.5	f
John	65.0	m
Anna	50.0	f
Bryan	75.0	m
Rob	50.0	m
Kay	43.3	f
Calvin	89.3	m
Peter	55.1	m
Julia	41.1	f

Append columns to DataFrames (concat)

```
import pandas as pd
df1 = pd.read_csv("data/Weights1.csv",
                  index_col=['name'])
df2 = pd.read_csv("data/Weights2.csv",
                  index_col=['name'])

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

df1

	weight	gender
name		
Mary	45.5	f
John	65.0	m
Anna	50.0	f
Bryan	75.0	m
Rob	50.0	m
Kay	43.3	f

df2

	weight	gender
name		
Calvin	89.3	m
Peter	55.1	m
Julia	41.1	f

result

	weight	gender	weight	gender
name				
Mary	45.5	f	NaN	NaN
John	65.0	m	NaN	NaN
Anna	50.0	f	NaN	NaN
Bryan	75.0	m	NaN	NaN
Rob	50.0	m	NaN	NaN
Kay	43.3	f	NaN	NaN
Calvin	NaN	NaN	89.3	m
Peter	NaN	NaN	55.1	m
Julia	NaN	NaN	41.1	f

Append rows to DataFrames (concat)

```
import pandas as pd
df1 = pd.read_csv("data/Weights1.csv",index_col=['name'])
df3 = pd.read_csv("data/Heights2.csv",index_col=['name'])

result = pd.concat([df1,df3])
```

df1

	weight	gender
name		
Mary	45.5	f
John	65.0	m
Anna	50.0	f
Bryan	75.0	m
Rob	50.0	m
Kay	43.3	f

df3

	height	gender
name		
Calvin	1.88	m
Peter	1.67	m
Julia	1.51	f

result

	weight	gender	height
name			
Mary	45.5	f	NaN
John	65.0	m	NaN
Anna	50.0	f	NaN
Bryan	75.0	m	NaN
Rob	50.0	m	NaN
Kay	43.3	f	NaN
Calvin	NaN	m	1.88
Peter	NaN	m	1.67
Julia	NaN	f	1.51

Concat – One More

In [29]:
```
import pandas as pd
df1 = pd.read_csv("pandas_data/Weights1.csv", index_col=['name'])
df2 = pd.read_csv("pandas_data/Weights2.csv", index_col=['name'])

df3 = pd.read_csv("pandas_data/Heights2.csv",index_col=['name'])
df3.loc['Mary'] = {'height': 1.63, 'gender':'f'}

print(df3)
```

name	height	gender
Calvin	1.88	m
Peter	1.67	m
Julia	1.51	f
Mary	1.63	f

Note: One row is inserted to the df3

```
result1 = pd.concat([df1,df3], axis=0)
print(result1)
```

name	weight	gender	height
Mary	45.5	f	NaN
John	65.0	m	NaN
Anna	50.0	f	NaN
Bryan	75.0	m	NaN
Rob	50.0	m	NaN
Kay	43.3	f	NaN
Calvin	NaN	m	1.88
Peter	NaN	m	1.67
Julia	NaN	f	1.51
Mary	NaN	f	1.63

```
result2 = pd.concat([df1,df3], axis=1)

print(result2)
```

name	weight	gender	height	gender
Mary	45.5	f	1.63	f
John	65.0	m	NaN	NaN
Anna	50.0	f	NaN	NaN
Bryan	75.0	m	NaN	NaN
Rob	50.0	m	NaN	NaN
Kay	43.3	f	NaN	NaN
Calvin	NaN	NaN	1.88	m
Peter	NaN	NaN	1.67	m
Julia	NaN	NaN	1.51	f

Spread rows into columns (pivot)

```
import pandas as pd
df = pd.read_csv("data/Weights.csv")
df2= df.pivot(index='observation',columns='gender',values='weight')
```

index

Values of 'gender'

Before pivot

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

After pivot

gender	f	m
observation		
1	45.5	NaN
2	NaN	65.0
3	50.0	NaN
4	NaN	75.0
5	NaN	50.0
6	43.3	NaN

Gather columns into rows (melt)

```
import pandas as pd
df = pd.read_csv("data/Weights.csv")

df3 = pd.melt(df, id_vars=['observation','gender'])
```

Before melt

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

After melt

	observation	gender	variable	value
0	1	f	weight	45.5
1	2	m	weight	65.0
2	3	f	weight	50.0
3	4	m	weight	75.0
4	5	m	weight	50.0
5	6	f	weight	43.3

Sort DataFrame by column values

DataFrame.sort_values(by, axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')[source]

```
import pandas as pd
df = pd.read_csv("data/Weights.csv")
df2 = df.sort_values(by="weight")
```

Before sort

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

After sort

	observation	weight	gender
5	6	43.3	f
0	1	45.5	f
2	3	50.0	f
4	5	50.0	m
1	2	65.0	m
3	4	75.0	m

Sort DataFrame by index values

```
import pandas as pd
import numpy as np

dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df2= df.sort_index(ascending=False)
```

	A	B	C	D
2013-01-01	0.349165	-0.163410	-0.886826	0.344683
2013-01-02	0.448764	-1.191895	-0.434969	-0.167275
2013-01-03	1.287675	-1.576437	-1.001160	-0.719710
2013-01-04	0.323716	1.004806	2.013644	-0.226959
2013-01-05	-0.590850	-0.897190	-1.151655	0.965096
2013-01-06	-1.364636	1.976835	0.957803	-0.796417

df

	A	B	C	D
2013-01-06	-1.364636	1.976835	0.957803	-0.796417
2013-01-05	-0.590850	-0.897190	-1.151655	0.965096
2013-01-04	0.323716	1.004806	2.013644	-0.226959
2013-01-03	1.287675	-1.576437	-1.001160	-0.719710
2013-01-02	0.448764	-1.191895	-0.434969	-0.167275
2013-01-01	0.349165	-0.163410	-0.886826	0.344683

df2

Sort DataFrame by index values (2)

```
import pandas as pd
df=pd.read_csv('data/stock_index_prices.csv')
df2=df.set_index(['TradingDate','PriceType'])
df3=df2.sort_index(ascending=False,level="PriceType")
```

	TradingDate	PriceType	Nasdaq	S&P 500	Russell 2000
0	2014/02/21	open	4282.17	1841.07	1166.25
1	2014/02/21	close	4263.41	1836.25	1164.63
2	2014/02/21	high	4284.85	1846.13	1168.43
3	2014/02/24	open	4273.32	1836.78	1166.74
4	2014/02/24	close	4292.97	1847.61	1174.55
5	2014/02/24	high	4311.13	1858.71	1180.29
6	2014/02/25	open	4298.48	1847.66	1176.00
7	2014/02/25	close	4287.59	1845.12	1173.95
8	2014/02/25	high	4307.51	1852.91	1179.43
9	2014/02/26	open	4300.45	1845.79	1176.11

df

		Nasdaq	S&P 500	Russell 2000
TradingDate	PriceType			
2014/02/21	open	4282.17	1841.07	1166.25
	close	4263.41	1836.25	1164.63
	high	4284.85	1846.13	1168.43
2014/02/24	open	4273.32	1836.78	1166.74
	close	4292.97	1847.61	1174.55
	high	4311.13	1858.71	1180.29

df2

		Nasdaq	S&P 500	Russell 2000
TradingDate	PriceType			
2014/02/28	open	4323.52	1855.12	1189.19
2014/02/27	open	4291.47	1844.90	1179.28
2014/02/26	open	4300.45	1845.79	1176.11
2014/02/25	open	4298.48	1847.66	1176.00
2014/02/24	open	4273.32	1836.78	1166.74
2014/02/21	open	4282.17	1841.07	1166.25
2014/02/28	high	4342.59	1867.92	1193.50
2014/02/27	high	4322.46	1854.53	1187.94
2014/02/26	high	4316.82	1852.65	1188.06
2014/02/25	high	4307.51	1852.91	1179.43
2014/02/24	high	4311.13	1858.71	1180.29
2014/02/21	high	4284.85	1846.13	1168.43
2014/02/28	close	4308.12	1859.45	1183.03
2014/02/27	close	4318.93	1854.29	1187.94

df3

Reindex a DataFrame

```
import pandas as pd
index = ['Firefox', 'Chrome', 'Safari', 'IE10','Konqueror']
df = pd.DataFrame({
     'http_status': [200,200,404,404,301],
     'response_time': [0.04, 0.02, 0.07, 0.08, 1.0]}, index=index)

new_index= ['Safari', 'Iceweasel', 'Comodo Dragon', 'IE10','Chrome']
df2 = df.reindex(new_index)
```

	http_status	response_time
Firefox	200	0.04
Chrome	200	0.02
Safari	404	0.07
IE10	404	0.08
Konqueror	301	1.00

Before reindexing

	http_status	response_time
Safari	404.0	0.07
Iceweasel	NaN	NaN
Comodo Dragon	NaN	NaN
IE10	404.0	0.08
Chrome	200.0	0.02

After reindexing (df2)

Reset the index of a DataFrame

```
import pandas as pd
df=pd.read_csv('data/stock_index_prices.csv')
df = df.set_index(['TradingDate','PriceType'])

df= df.reset_index()
```

		Nasdaq	S&P 500	Russell 2000
TradingDate	**PriceType**			
2014/02/21	**open**	4282.17	1841.07	1166.25
	close	4263.41	1836.25	1164.63
	high	4284.85	1846.13	1168.43
2014/02/24	**open**	4273.32	1836.78	1166.74
	close	4292.97	1847.61	1174.55
	high	4311.13	1858.71	1180.29

Before reset index

	TradingDate	PriceType	Nasdaq	S&P 500	Russell 2000
0	2014/02/21	open	4282.17	1841.07	1166.25
1	2014/02/21	close	4263.41	1836.25	1164.63
2	2014/02/21	high	4284.85	1846.13	1168.43
3	2014/02/24	open	4273.32	1836.78	1166.74
4	2014/02/24	close	4292.97	1847.61	1174.55
5	2014/02/24	high	4311.13	1858.71	1180.29
6	2014/02/25	open	4298.48	1847.66	1176.00

After reset index

Rename the columns of a DataFrame

```
import pandas as pd
import pandas as pd
df = pd.read_csv("data/Weights.csv")

df=df.rename(columns={"observation":"reading"})
```

Before rename

	observation	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

After rename

	reading	weight	gender
0	1	45.5	f
1	2	65.0	m
2	3	50.0	f
3	4	75.0	m
4	5	50.0	m
5	6	43.3	f

Handling Missing Data

When / why does data become missing?

- Many data sets simply arrive with missing data, either because it exists and was not collected or it never existed
- For example, in a collection of financial time series, some of the time series might start on different dates. Thus, values prior to the start date would generally be marked as missing

E.g how reindexing causes missing data

```
df1 = pd.DataFrame(np.random.randn(3, 3), index=['a', 'b', 'c',
                   columns=['one', 'two', 'three'])

df2 = df1.reindex(['a', 'b', 'c', 'd'])
```

	one	two	three
a	-0.192006	-0.424708	-0.415116
b	-0.973682	1.882696	-0.755095
c	-0.368496	0.456729	-0.240106

	one	two	three
a	-0.192006	-0.424708	-0.415116
b	-0.973682	1.882696	-0.755095
c	-0.368496	0.456729	-0.240106
d	NaN	NaN	NaN

Handling Missing Data

Code Example	Description
isnull	Detect missing values (NaN in numeric arrays, None/NaN in object arrays)
notnull	Replacement for numpy.isfinite / -numpy.isnan which is suitable for use on object arrays
df.dropna	Drop rows with any column having NA/null data
df.fillna	Replace all NA/null data with value
df.interpolate	Uses interpolation to ‘best guess’ missing numeric values

Handling Missing Data in Pandas

- isnull() and notnull() functions in pandas may be used on both Series and DataFrame objects to detect missing values

```
import pandas as pd
df = pd.read_csv('data/kidney_disease.csv')
print(df.loc[0:4,'id':'pcc']) # print first few columns and rows
print(pd.isnull(df['rbc']).head()) # check if 'rbc' has null values
df["rbc"].isnull().sum() # counts number of null values in 'rbc'
```

	id	age	bp	sg	al	su	rbc	pc	pcc
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent

```
0     True
1     True
2    False
3    False
4    False
```

Values considered missing

- As data comes in many shapes and forms, pandas aims to be flexible with regards to handling missing data
- While **NaN** is the default missing value marker for reasons of computational speed and convenience, we need to be able to easily detect this value with data of different types: floating point, integer, boolean, and general object
- In many cases, however, the Python **None** will arise and we wish to also consider that “missing” or “null”.

Handling Missing Data

- There are many ways to handle missing data. Some common methods include:
  - Drop them! Simple and will not introduce errors through imputation

  Univariate Imputation
  - Fill with fixed value or derived from ‘neighbouring’ records
  - Mean imputation/Mode for categorical
  - Interpolation

  Multivariate Imputation
  - Regression
  - KNN etc

- Do note that imputation introduces errors! You are using values to estimate the missing value, then using that value again for the model. Limit to 5%.

dropna

```
import pandas as pd

df = pd.read_csv('data/kidney_disease.csv')
df2 = df.dropna()
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

Original dataset with NaN values. See the next slide for comparison after using dropna

dropna

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classificatio
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29	12100	3.7	yes	yes	no	poor	no	yes	ckd
11	11	63.0	70.0	1.010	3.0	0.0	abnormal	abnormal	present	notpresent	...	32	4500	3.8	yes	yes	no	poor	yes	no	ckd
14	14	68.0	80.0	1.010	3.0	2.0	normal	abnormal	present	present	...	16	11000	2.6	yes	yes	yes	poor	yes	no	ckd
20	20	61.0	80.0	1.015	2.0	0.0	abnormal	abnormal	notpresent	notpresent	...	24	9200	3.2	yes	yes	yes	poor	yes	yes	ckd
22	22	48.0	80.0	1.025	4.0	0.0	normal	abnormal	notpresent	notpresent	...	32	6900	3.4	yes	no	no	good	no	yes	ckd
27	27	69.0	70.0	1.010	3.0	4.0	normal	abnormal	notpresent	notpresent	...	37	9600	4.1	yes	yes	yes	good	yes	no	ckd

fillna (scalar values)

- The fillna function can “fill in” NA values with non-null data in a few ways:

scalar values

fill gaps forwards

fill gaps backwards

fillna (scalar values)

```
import pandas as pd

df = pd.read_csv('data/kidney_disease.csv')

# Replace NA with a scalar value
df2 = df.fillna(0)
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

Original dataset with NaN values. See the next slide for comparison after using fillna

fillna (scalar values)

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classificatio
0	0	48.0	80.0	1.020	1.0	0.0	0	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	0	normal	notpresent	notpresent	...	38	6000	0	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	0	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	0	0	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

fillna (fill gaps forward)

```
import pandas as pd

df = pd.read_csv('data/kidney_disease.csv')

# Fill gaps forwards
df2 = df.fillna(method='ffill')
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

Original dataset with NaN values. See the next slide for comparison after using fillna

fillna (fill gaps forward)

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classificatio
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	5.2	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	5.2	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	normal	normal	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

fillna (fill gaps backward)

```
import pandas as pd

df = pd.read_csv('data/kidney_disease.csv')

# Fill gaps forwards
df2 = df.fillna(method='bfill')
```

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

Original dataset with NaN values. See the next slide for comparison after using fillna

fillna (fill gaps backward)

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classificatio
0	0	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	...	38	6000	3.9	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	3.9	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	normal	normal	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

fillna (mean)

```
import pandas as pd

df = pd.read_csv('data/kidney_disease.csv')

df.rc = pd.to_numeric(df.rc, errors='coerce')
# Replace NA with the mean of the values of the column
df.rc =df['rc'].fillna(df['rc'].mean())
```

Why is this step necessary?

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

Original dataset with NaN values. See the next slide for comparison after using fillna

fillna (mean)

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	NaN	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	NaN	normal	notpresent	notpresent	...	38	6000	NaN	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	NaN	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	NaN	NaN	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd

	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm
0	0	48.0	80.000000	1.020000	1.000000	0.000000	NaN	normal	notpresent	notpresent	...	44	7800.000000	5.200000	yes	yes
1	1	7.0	50.000000	1.020000	4.000000	0.000000	NaN	normal	notpresent	notpresent	...	38	6000.000000	4.707435	no	no
2	2	62.0	80.000000	1.010000	2.000000	3.000000	normal	normal	notpresent	notpresent	...	31	7500.000000	4.707435	no	yes
3	3	48.0	70.000000	1.005000	4.000000	0.000000	normal	abnormal	present	notpresent	...	32	6700.000000	3.900000	yes	no
4	4	51.0	80.000000	1.010000	2.000000	0.000000	normal	normal	notpresent	notpresent	...	35	7300.000000	4.600000	no	no
5	5	60.0	90.000000	1.015000	3.000000	0.000000	NaN	NaN	notpresent	notpresent	...	39	7800.000000	4.400000	yes	yes
6	6	68.0	70.000000	1.010000	0.000000	0.000000	NaN	normal	notpresent	notpresent	...	36	8406.122449	4.707435	no	no

Handling Missing Data – fillna (mode)

- Mean cannot be computed for categorical variables. You can use mode.

```
df = pd.read_csv('pandas_data/kidney_disease.csv')

count1 = df['rbc'].isnull().sum()

print('Number of Nan values in rbc:', count1)

df2= df.fillna(df.mode().iloc[0]) # Replace all columns NaN with mode vlaue

count2 = df2["rbc"].isnull().sum() # Number of Nan values in rbc 0

print('Number of Nan values in rbc:', count2)

df2.head(10)
```

Handling Missing Data – fillna (mode)

```
Number of Nan values in rbc: 152
Number of Nan values in rbc: 0
```

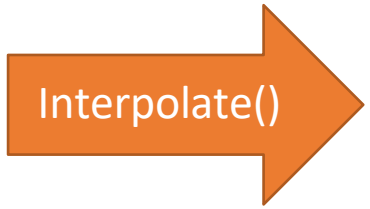
	id	age	bp	sg	al	su	rbc	pc	pcc	ba	...	pcv	wc	rc	htn	dm	cad	appet	pe	ane	classification
0	0	48.0	80.0	1.020	1.0	0.0	normal	normal	notpresent	notpresent	...	44	7800	5.2	yes	yes	no	good	no	no	ckd
1	1	7.0	50.0	1.020	4.0	0.0	normal	normal	notpresent	notpresent	...	38	6000	5.2	no	no	no	good	no	no	ckd
2	2	62.0	80.0	1.010	2.0	3.0	normal	normal	notpresent	notpresent	...	31	7500	5.2	no	yes	no	poor	no	yes	ckd
3	3	48.0	70.0	1.005	4.0	0.0	normal	abnormal	present	notpresent	...	32	6700	3.9	yes	no	no	poor	yes	yes	ckd
4	4	51.0	80.0	1.010	2.0	0.0	normal	normal	notpresent	notpresent	...	35	7300	4.6	no	no	no	good	no	no	ckd
5	5	60.0	90.0	1.015	3.0	0.0	normal	normal	notpresent	notpresent	...	39	7800	4.4	yes	yes	no	good	yes	no	ckd
6	6	68.0	70.0	1.010	0.0	0.0	normal	normal	notpresent	notpresent	...	36	9800	5.2	no	no	no	good	no	no	ckd
7	7	24.0	80.0	1.015	2.0	4.0	normal	abnormal	notpresent	notpresent	...	44	6900	5	no	yes	no	good	yes	no	ckd
8	8	52.0	100.0	1.015	3.0	0.0	normal	abnormal	present	notpresent	...	33	9600	4.0	yes	yes	no	good	no	yes	ckd
9	9	53.0	90.0	1.020	2.0	0.0	abnormal	abnormal	present	notpresent	...	29	12100	3.7	yes	yes	no	poor	no	yes	ckd

Interpolate

- Interpolation is a mathematical method that adjusts a function to your data and uses this function to extrapolate the missing data.
- The simplest type of interpolation is the linear interpolation, that makes a mean between the values before the missing data and the value after.

```
df2 =df.interpolate() # default linear method for numeric
```

	...	pcv	wc	rc
0	...	44	7800	5.2
1	...	38	6000	NaN
2	...	31	7500	NaN
3	...	32	6700	3.9
4	...	35	7300	4.6



	...	pcv	wc	rc
0	...	44	7800	5.200000
1	...	38	6000	4.766667
2	...	31	7500	4.333333
3	...	32	6700	3.900000
4	...	35	7300	4.600000

Combine Data Sets

Combine Data Sets

Code Example	Description
pd.merge(adf, bdf, how='left', on='x1')	Get all rows from adf and only those rows from bdf that have a match with adf on the column x1
pd.merge(adf, bdf, how='right', on='x1')	Get all rows from bdf and only those rows from adf that have a match with bdf on the column x1
pd.merge(adf, bdf, how='inner', on='x1')	Join matching rows from both adf and bdf
pd.merge(adf, bdf, how='outer', on='x1')	Join all rows from adf and bdf
adf[adf.x1.isin(bdf.x1)]	All rows in adf that have a match in bdf.
adf[~adf.x1.isin(bdf.x1)]	All rows in adf that do not have a match in bdf

merge (how='left')

```
import pandas as pd
df1 = pd.read_csv('data/mergedataset1s.csv')
df2 = pd.read_csv('data/mergedataset2s.csv')
df = pd.merge(df1, df2, how='left', on='customerid')
```

```
transactionid,amount,customerid,datetransaction
1,85,101,1 Jan 2017
9,401,105,11 Jan 2017
11,202,106,11 Jan 2017
12,823,102,12 Jan 2017
13,720,107,12 Jan 2017
14,399,104,13 Jan 2017
51,344,109,31 Jan 2017
52,190,110,31 Jan 2017
```

df1- mergedataset1s.csv

```
customerid,name,region
100,Mary,north
101,John,south
102,Brenda,south
103,Chad,central
104,Tanya,west
105,Simon,east
106,Leonardo,north
```

df2 - mergedataset2s.csv

Original datasets before merging. See the next slide for comparison after merging.

merge (how='left')

```
Merge type = LEFT

(8, 6)

   transactionid  amount  customerid datetransaction      name region
0              1      85         101      1 Jan 2017      John  south
1              9     401         105     11 Jan 2017     Simon   east
2             11     202         106     11 Jan 2017  Leonardo  north
3             12     823         102     12 Jan 2017    Brenda  south
4             13     720         107     12 Jan 2017       NaN    NaN
5             14     399         104     13 Jan 2017     Tanya   west
6             51     344         109     31 Jan 2017       NaN    NaN
7             52     190         110     31 Jan 2017       NaN    NaN
```

merge (how='right')

```
import pandas as pd
df1 = pd.read_csv('data/mergedataset1s.csv')
df2 = pd.read_csv('data/mergedataset2s.csv')
df = pd.merge(df1, df2, how='right', on='customerid')
```

```
transactionid,amount,customerid,datetransaction
1,85,101,1 Jan 2017
9,401,105,11 Jan 2017
11,202,106,11 Jan 2017
12,823,102,12 Jan 2017
13,720,107,12 Jan 2017
14,399,104,13 Jan 2017
51,344,109,31 Jan 2017
52,190,110,31 Jan 2017
```

df1- mergedataset1s.csv

```
customerid,name,region
100,Mary,north
101,John,south
102,Brenda,south
103,Chad,central
104,Tanya,west
105,Simon,east
106,Leonardo,north
```

df2 - mergedataset2s.csv

Original datasets before merging. See the next slide for comparison after merging.

merge (how='right')

```
Merge type = RIGHT

(7, 6)

   transactionid  amount  customerid datetransaction      name   region
0            1.0    85.0         101       1 Jan 2017      John    south
1            9.0   401.0         105      11 Jan 2017     Simon     east
2           11.0   202.0         106      11 Jan 2017  Leonardo    north
3           12.0   823.0         102      12 Jan 2017    Brenda    south
4           14.0   399.0         104      13 Jan 2017     Tanya     west
5            NaN     NaN         100             NaN      Mary    north
6            NaN     NaN         103             NaN      Chad  central
```


merge (how='inner')

```
import pandas as pd
df1 = pd.read_csv('data/mergedataset1s.csv')
df2 = pd.read_csv('data/mergedataset2s.csv')
df = pd.merge(df1, df2, how='inner', on='customerid')
```

```
transactionid,amount,customerid,datetransaction
1,85,101,1 Jan 2017
9,401,105,11 Jan 2017
11,202,106,11 Jan 2017
12,823,102,12 Jan 2017
13,720,107,12 Jan 2017
14,399,104,13 Jan 2017
51,344,109,31 Jan 2017
52,190,110,31 Jan 2017
```

df1- mergedataset1s.csv

```
customerid,name,region
100,Mary,north
101,John,south
102,Brenda,south
103,Chad,central
104,Tanya,west
105,Simon,east
106,Leonardo,north
```

df2 - mergedataset2s.csv

Original datasets before merging. See the next slide for comparison after merging.

merge (how='inner')

```
Merge type = INNER

(5, 6)

   transactionid  amount  customerid datetransaction      name region
0              1      85         101       1 Jan 2017      John  south
1              9     401         105      11 Jan 2017     Simon   east
2             11     202         106      11 Jan 2017  Leonardo  north
3             12     823         102      12 Jan 2017    Brenda  south
4             14     399         104      13 Jan 2017     Tanya   west
```

isin

```
import pandas as pd
df1 = pd.read_csv('data/mergedataset1s.csv')
df2 = pd.read_csv('data/mergedataset2s.csv')
df = df1[df1.customerid.isin(df2.customerid)]
```

```
transactionid,amount,customerid,datetransaction
1,85,101,1 Jan 2017
9,401,105,11 Jan 2017
11,202,106,11 Jan 2017
12,823,102,12 Jan 2017
13,720,107,12 Jan 2017
14,399,104,13 Jan 2017
51,344,109,31 Jan 2017
52,190,110,31 Jan 2017
```

df1- mergedataset1s.csv

```
customerid,name,region
100,Mary,north
101,John,south
102,Brenda,south
103,Chad,central
104,Tanya,west
105,Simon,east
106,Leonardo,north
```

df2 - mergedataset2s.csv

Original datasets before merging. See the next slide for comparison after merging.

isin

```
isin

(5, 4)

   transactionid  amount  customerid datetransaction
0              1      85         101     1 Jan 2017
1              9     401         105    11 Jan 2017
2             11     202         106    11 Jan 2017
3             12     823         102    12 Jan 2017
5             14     399         104    13 Jan 2017
```

~isin

```
import pandas as pd
df1 = pd.read_csv('data/mergedataset1s.csv')
df2 = pd.read_csv('data/mergedataset2s.csv')
df = df1[~df1.customerid.isin(df2.customerid)]
```

```
transactionid,amount,customerid,datetransaction
1,85,101,1 Jan 2017
9,401,105,11 Jan 2017
11,202,106,11 Jan 2017
12,823,102,12 Jan 2017
13,720,107,12 Jan 2017
14,399,104,13 Jan 2017
51,344,109,31 Jan 2017
52,190,110,31 Jan 2017
```

df1- mergedataset1s.csv

```
customerid,name,region
100,Mary,north
101,John,south
102,Brenda,south
103,Chad,central
104,Tanya,west
105,Simon,east
106,Leonardo,north
```

df2 - mergedataset2s.csv

Original datasets before merging. See the next slide for comparison after merging.

~isin

```
~ isin

(3, 4)

   transactionid  amount  customerid datetransaction
4             13     720         107      12 Jan 2017
6             51     344         109      31 Jan 2017
7             52     190         110      31 Jan 2017
```

Group Data

Group data

Code Example	Description
df1 = df.groupby(by='col)	Return a GroupBy object, grouped by values in column named 'col'
df1 = df.groupby(level='ind')	Return a GroupBy object, grouped by values in index level named 'ind'
df1.groups	The dictionary of groups
len(df1.groups)	Number of groups in the dictionary
size()	Size of each group
agg(function)	Aggregate group using function

Study the partial dataset below which stores the results of the European club soccer championship since 1955

	Season	Nation	Winners	Score	Runners-up	Runner-UpNation	Venue	Attendance
0	1955–56	Spain	Real Madrid	4–3	Stade de Reims	France	Parc des Princes,Paris	38239
1	1956–57	Spain	Real Madrid	2–0	Fiorentina	Italy	Santiago Bernabéu Stadium, Madrid	124000
2	1957–58	Spain	Real Madrid	3–2	Milan	Italy	Heysel Stadium,Brussels	67000
3	1958–59	Spain	Real Madrid	2–0	Stade de Reims	France	Neckarstadion,Stuttgart	72000
4	1959–60	Spain	Real Madrid	7–3	Eintracht Frankfurt	Germany	Hampden Park,Glasgow	127621
5	1960–61	Portugal	Benfica	3–2	Barcelona	Spain	Wankdorf Stadium,Bern	26732
6	1961–62	Portugal	Benfica	5–3	Real Madrid	Spain	Olympisch Stadion,Amsterdam	61257
7	1962–63	Italy	Milan	2–1	Benfica	Portugal	Wembley Stadium,London	45715
8	1963–64	Italy	Internazionale	3–1	Real Madrid	Spain	Prater Stadium,Vienna	71333
9	1964–65	Italy	Internazionale	1–0	Benfica	Portugal	San Siro, Milan	89000
10	1965–66	Spain	Real Madrid	2–1	Partizan	Yugoslavia	Heysel Stadium,Brussels	46745
11	1966–67	Scotland	Celtic	2–1	Internazionale	Italy	Estádio Nacional,Lisbon	45000
12	1967–68	England	Manchester United	4–1	Benfica	Portugal	Wembley Stadium,London	92225

Suppose I want to find the countries who won the championship the most number of times. We can rank the nations by the number of European club championships they have won

We can achieve this by using the **groupby** function to group the data by **Nation**.

We can then use **size()** to count how many times each Nation appeared in the dataset (indicating a win for that nation). This returns a **Series**, with **Nation** as the index.

```
Spain          13
Italy          12
England        12
Germany         7
Netherlands     6
Portugal        4
Yugoslavia      1
Scotland        1
Romania         1
France          1
dtype: int64
```


Grouping Data

```
import pandas as pd
df = pd.read_csv("data/euro_winners.csv")

# Returns a dictionary of DataFrameGroupBy objects
nationsGrp = df.groupby(['Nation'])

# size() returns a Series, indexed by "Nation"
nationWins = nationsGrp.size()
nationWins.sort_values(ascending=False)
```

```
Nation
Spain          13
Italy          12
England        12
Germany         7
Netherlands     6
Portugal        4
Yugoslavia      1
Scotland        1
Romania         1
France          1
dtype: int64
```


Let's say we want to find not the best country, but the best clubs! We can find out the total wins, not only by country alone, but by club as well.

We can achieve this by using multicolumn groupby function in Pandas.

Nation	Winners	
Spain	Real Madrid	9
Italy	Milan	7
Germany	Bayern Munich	5
England	Liverpool	5
Spain	Barcelona	4
Netherlands	Ajax	4
England	Manchester United	3
Italy	Internazionale	3
	Juventus	2
Portugal	Porto	2
	Benfica	2
England	Nottingham Forest	2
	Chelsea	1
France	Marseille	1
Yugoslavia	Red Star Belgrade	1
Germany	Borussia Dortmund	1
	Hamburg	1

Grouping Data

```
import pandas as pd
df = pd.read_csv("data/euro_winners.csv")

# To do a further breakup by country and club
# apply a multicolumn groupby function
winnersGrp = df.groupby(['Nation','Winners'])

# a dataframe is returned indexed by both
# "Nation" and "Winners"
clubWins=winnersGrp.size()
clubWins.sort_values(ascending=False)
```

```
Nation       Winners
Spain        Real Madrid          9
Italy        Milan                7
Germany      Bayern Munich        5
England      Liverpool            5
Spain        Barcelona            4
Netherlands  Ajax                 4
England      Manchester United    3
Italy        Internazionale       3
             Juventus             2
Portugal     Porto                2
             Benfica              2
England      Nottingham Forest    2
             Chelsea              1
France       Marseille            1
Yugoslavia   Red Star Belgrade    1
Germany      Borussia Dortmund    1
             Hamburg              1
Netherlands  Feyenoord            1
             PSV Eindhoven        1
Romania      Steaua Bucure?ti     1
Scotland     Celtic               1
England      Aston Villa          1
dtype: int64
```

Knowing the best teams, I wish to sum up all the attendance for the matches. Are the matches where the best teams win also the most popular (highest attendance)?

To do this we need to aggregate the data in the Attendance column by summing up all the attendance based on matching Nation and Winners.

We can then sort it if we want.

Nation	Winners	Attendance
Spain	Real Madrid	654604
Italy	Milan	427312
Germany	Bayern Munich	340583
England	Liverpool	331631
Spain	Barcelona	300599
Netherlands	Ajax	283747
England	Manchester United	249780
Italy	Internazionale	233823
	Juventus	128000
Portugal	Porto	110553
England	Nottingham Forest	108500
Portugal	Benfica	87989
Germany	Hamburg	73500
Romania	Steaua Bucure?ti	70000
Netherlands	PSV Eindhoven	68000

Aggregate Grouped Data

```
import pandas as pd
df = pd.read_csv("data/euro_winners.csv")

# We can aggregate data by using sum
# on the Attendance column
att = df.groupby(['Nation','Winners'])
                    [['Attendance']].sum()
```

Nation	Winners	Attendance
England	Aston Villa	46000
	Chelsea	62500
	Liverpool	331631
	Manchester United	249780
	Nottingham Forest	108500
France	Marseille	64400
Germany	Bayern Munich	340583
	Borussia Dortmund	59000
	Hamburg	73500
Italy	Internazionale	233823
	Juventus	128000
	Milan	427312
Netherlands	Ajax	283747
	Feyenoord	53187
	PSV Eindhoven	68000

Aggregate Grouped Data

```
import pandas as pd
df = pd.read_csv("data/euro_winners.csv")

# We can aggregate data by using sum
# on the Attendance column
att = df.groupby(['Nation','Winners'])
                    [['Attendance']].sum()


# You can choose to sort by Attenance
att.sort_values(ascending=False,
                           by='Attendance')
```

		Attendance
Nation	Winners	
Spain	Real Madrid	654604
Italy	Milan	427312
Germany	Bayern Munich	340583
England	Liverpool	331631
Spain	Barcelona	300599
Netherlands	Ajax	283747
England	Manchester United	249780
Italy	Internazionale	233823
	Juventus	128000
Portugal	Porto	110553
England	Nottingham Forest	108500
Portugal	Benfica	87989
Germany	Hamburg	73500
Romania	Steaua Bucure?ti	70000
Netherlands	PSV Eindhoven	68000

With the Series of Wins and the DataFrame of Attendance, we can now join this two data to produce the final table on the right.

Do you see a trend for the Attendance and the number of Wins?

		Attendance	Wins
Nation	Winners		
Spain	Real Madrid	654604	9
Italy	Milan	427312	7
Germany	Bayern Munich	340583	5
England	Liverpool	331631	5
Spain	Barcelona	300599	4
Netherlands	Ajax	283747	4
England	Manchester United	249780	3
Italy	Internazionale	233823	3
	Juventus	128000	2
Portugal	Porto	110553	2
England	Nottingham Forest	108500	2
Portugal	Benfica	87989	2
Germany	Hamburg	73500	1
Romania	Steaua Bucure?ti	70000	1
Netherlands	PSV Eindhoven	68000	1

Aggregate Grouped Data

```
import pandas as pd
df = pd.read_csv("euro_winners.csv")

winnersGrp = df.groupby(['Nation','Winners'])
clubWins=winnersGrp.size()

att = df.groupby(['Nation','Winners'])
                        [['Attendance']].sum()

clubWins.name = "Wins" # Name the Series
attendance = att.join(clubWins)
attendance.sort_values(ascending=False,
                                by='Attendance')
```

Nation	Winners	Attendance	Wins
Spain	Real Madrid	654604	9
Italy	Milan	427312	7
Germany	Bayern Munich	340583	5
England	Liverpool	331631	5
Spain	Barcelona	300599	4
Netherlands	Ajax	283747	4
England	Manchester United	249780	3
Italy	Internazionale	233823	3
	Juventus	128000	2
Portugal	Porto	110553	2
England	Nottingham Forest	108500	2
Portugal	Benfica	87989	2
Germany	Hamburg	73500	1
Romania	Steaua Bucure?ti	70000	1
Netherlands	PSV Eindhoven	68000	1

apply (for panda Series)

```
import pandas as pd
import numpy as np

series = pd.Series([20, 21, 12],
index=['London','New York','Helsinki'])

def toFahrenheit(x):
    return x*32

series = series.apply(toFahrenheit)
```

```
BEFORE apply
London      20
New York    21
Helsinki    12
dtype: int64

AFTER apply
London      640
New York    672
Helsinki    384
dtype: int64
```

Datasets before and after applying the function

apply (for panda DataFrame)

Applies function along input axis of DataFrame

```
import pandas as pd
import numpy as np

data = np.random.randint(1,10,(3,2))
df = pd.DataFrame(data,
   index=['Student 1', 'Student 2',
'Student 3'],
    columns=['Reward 1', 'Reward 2'])

def multiply(x):
    return x*2

df = df.apply(multiply)
```

BEFORE apply

	Reward 1	Reward 2
Student 1	5	6
Student 2	4	1
Student 3	5	8

Dataset before applying the function

AFTER apply

	Reward 1	Reward 2
Student 1	10	12
Student 2	8	2
Student 3	10	16

Dataset after applying the function

apply (for panda DataFrame)

Applies function along input axis of DataFrame

```
data = {'name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
        'year': [1989, 1990, 1994, 1979, 1975]}
df = pd.DataFrame(data)


# Create a capitalization lambda function
capitalizer = lambda x: x.upper()

# Apply the capitalizer function over the column 'name'
df['name'].apply(capitalizer)
```

Dataset before applying the function

```
BEFORE apply
    name  year
0  Jason  1989
1  Molly  1990
2   Tina  1994
3   Jake  1979
4    Amy  1975
```

Dataset after applying the function

```
AFTER apply
    name  year
0  JASON  1989
1  MOLLY  1990
2   TINA  1994
3   JAKE  1979
4    AMY  1975
```

The End