Data Science

Week 3

Data Manipulation

Course Schedule

F: face to face class

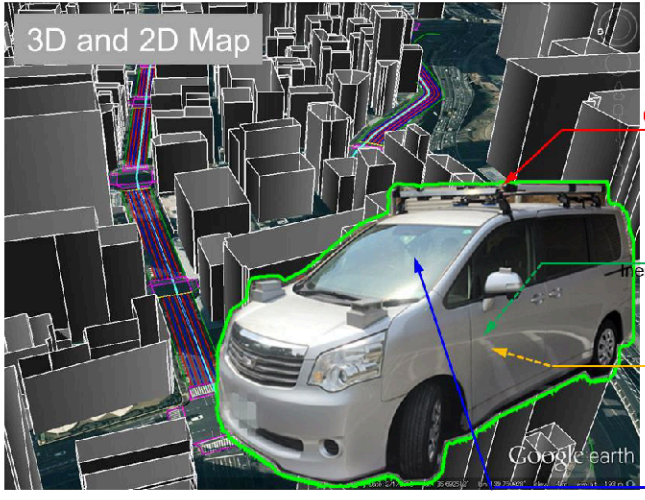
Week 1: Introduction
Week 2: Real-World Problems and Data Science Solutions
Week 3: Getting Start with Data Manipulation
Week 4: Exploratory Data Analytics (1) - Statistics
F Week 5: Test 1 (tentative) and Exploratory Data Analytics (1) – Visualization
F Week 6: Introduction to Predictive Modeling
Week 7: Fitting a Model to Data (1) - Classification
Week 8: Fitting a Model to Data (2) - Neural Networks
Week 9: Course Review and Q&A
F Week 10: Test 2 (tentative) and Fitting a Model to Data (3) - Regression
F Week 11: Fitting a Model to Data (4) – Clustering
Week 12: Decision Analytic Thinking: What Is a Good Model?
Week 13: Visualizing Data and Model Performance
Week 14: Representing and Mining Text
F Week 15: Course Summary and Final test

Building a Dataset from Scratch

- In my experience, we could not find the open dataset for most of projects, and we need to collect data by ourselves.
- Data collection is the process of gathering quantitative and qualitative information on specific variables. Good data collection requires a clear process to ensure the data you collect is clean, consistent, and reliable.
- Establishing that process involves taking stock of your objectives (limitations if any), identifying your data requirements, and finally organizing a data collection plan that synthesizes the most important aspects of your project.
- The following 5 examples (3 projects conducted by the instructor of this class) demonstrate how to perform the data collection for the projects with specific objectives.

Example 1: Multi-sensor based vehicle self-localization project

- Objective: using GNSS receiver, IMU (Inertial Measurement Unit) sensor, Speedometer, Camera, Map information for vehicle positioning



GNSS Receiver

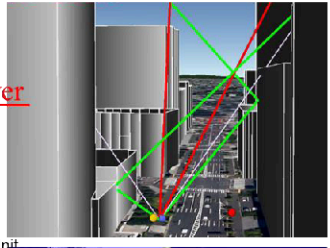
IMU sensor

Inertial Measurement Unit

Speedometer

Camera

3D-GNSS



Lane keeping detection

Example 2: Smartphone Lifelog project

Smartphone-based Lifelog automatically annotates the users' daily experience (position & activity) from multisensory streams on smartphones.

- Objective: using multisensory streams (GPS receiver, accelerometer, gyroscope, barometer, compass sensor, WiFi receiver) to calculate the position and activity of user.
- After survey, we found that there is NO open dataset can be used for this research.

Example 3: Stock performance prediction project

- Objective: Using historical data of stocks to predict the future of stock performance
- After survey, we found that there is NO open dataset (completed data) can be used for this research.
- We found that Thomson Reuters database (not free) can provide the data, and it is possible to use Datastream to access the financial data.
- We prepare a simple program to acquire the data automatically by batch processing.

Example 4: Data collection from internet-Wikipedia

- Using Wikipedia API and obtain the data from Wikipedia by python.

```
install wikipedia
```

```
import wikipedia

wikipedia.set_lang("en")
page = wikipedia.page('Ritsumeikan', auto_suggest=False)
print (page.content)
```

There are many useful functions wikipedia.*, please check the instruction of wikipedia lib when you use it.

Example 5: Data collection from internet-Website

- “Web scrapers” automatically collect information and data that's usually only accessible by visiting a website in a browser.
- By doing this autonomously, web scraping scripts open up a world of possibilities in data mining, data analysis, statistical analysis, and much more.

The copyright issue is existing when you use Web scrapers, please do NOT violate the **copyright laws** when you use Web scrapers

Outline

1. Data Loading and File Formats
2. Data Cleaning and Preprocessing

Getting Started with pandas

- Pandas adopts many coding idioms from NumPy, the biggest difference is that
  - Pandas is designed for working with tabular or heterogeneous (different types of) data.
  - NumPy, by contrast, is best suited for working with homogeneous numerical array data.

```
import pandas as pd
```

Data Loading function in pandas

Function	Description
`read_csv`	Load delimited data from a file, URL, or file-like object; use comma as default delimiter
`read_table`	Load delimited data from a file, URL, or file-like object; use tab (`'\t'`) as default delimiter
`read_fwf`	Read data in fixed-width column format (i.e., no delimiters)
`read_clipboard`	Version of `read_table` that reads data from the clipboard; useful for converting tables from web pages
`read_excel`	Read tabular data from an Excel XLS or XLSX file
`read_hdf`	Read HDF5 files written by pandas
`read_html`	Read all tables found in the given HTML document
`read_json`	Read data from a JSON (JavaScript Object Notation) string representation
`read_msgpack`	Read pandas data encoded using the MessagePack binary format
`read_pickle`	Read an arbitrary object stored in Python pickle format
`read_sas`	Read a SAS dataset stored in one of the SAS system's custom storage formats
`read_sql`	Read the results of a SQL query (using SQLAlchemy) as a pandas DataFrame
`read_stata`	Read a dataset from Stata file format
`read_feather`	Read the Feather binary file format

Comma-separated (CSV) text file

1) Use read_csv to read it into a DataFrame

2) A file will not always have a header row, we should use header=None to avoid losing the first row in this case

3) Give the column name (header) manually

4) Use one column to be the index of the returned DataFrame

Files used for programming

ex1.csv

	A	B	C	D	E
1	a	b	c	d	message
2	1	2	3	4	hello
3	5	6	7	8	world
4	9	10	11	12	foo

ex2.csv

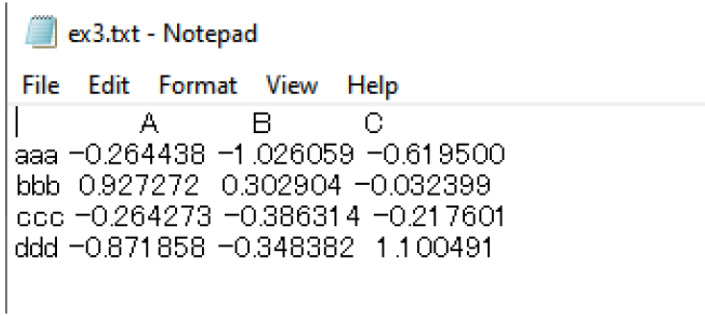
	A	B	C	D	E
1	1	2	3	4	hello
2	5	6	7	8	world
3	9	10	11	12	foo

Check the code by yourself

```
# (1) read standard CSV
# Code Example 1
print ("--------------------")
df = pd.read_csv('ch06/ex1.csv')
print (df)
print ("--------------------")
# (2 )A file will not always have a header row. Consider this file:
df2 = pd.read_csv('ch06/ex2.csv')
print (df2)
print ("--------------------")
# should use header=None to avoid lossing the first row
df3 = pd.read_csv('ch06/ex2.csv', header=None)
print (df3)
print ("--------------------")
# (3) give the column name manually
df4 = pd.read_csv('ch06/ex2.csv', names=['a', 'b', 'c', 'd', 'message'])
print (df4)
print ("--------------------")
# (4) use message column to be the index of the returned DataFrame
names = ['a', 'b', 'c', 'd', 'message']
df5 = pd.read_csv('ch06/ex2.csv', names=names, index_col='message')
print (df5)
print ("--------------------")
```

Whitespace-separated text file

- If the data are separated by the whitespace, we can pass a regular expression as a delimiter for `read_csv`. This can be expressed by the regular expression `\s+`:



```
# read other type Text
# Code Example 2
#In some cases, a table might not have a fixed delimiter, using
whitespace or some
#other pattern to separate fields. Consider a text file that looks
like this:
list(open('ch06/ex3.txt'))
result = pd.read_csv('ch06/ex3.txt')
print ("--------------------")
print (result)
print ("--------------------")
print (result.iloc[0,[0]])

result1 = pd.read_csv('ch06/ex3.txt', sep='\s+')
print ("--------------------")
print (result1)
print ("--------------------")
print (result1.iloc[0,[0]])
print ("--------------------")
print (result1.loc['aaa',['A']])
print ("--------------------")
```

File with missing data

ex5.csv

	A	B	C	D	E	F
1	somethin	a	b	c	d	message
2	one	1	2	3	4	all
3	two	5	6		8	world
4	three	9	10	11	12	foo

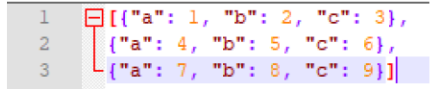
```
# read a file with missing data
# Code Example 3
result = pd.read_csv('ch06/ex5.csv')
print ("--------------------")
print (result)
print ("--------------------")
```

JSON Data

- Short for JavaScript Object Notation
- JSON has become one of the standard formats for sending data by HTTP request between web browsers and other applications.
- It is a much more free-form data format than a tabular text form like CSV.

Load JSON Data

JSON_example.json



- The pandas.read_json can automatically convert JSON datasets in specific arrangements into a Series or DataFrame.
- The default options for pandas.read_json assume that each object in the JSON array is a row in the table

Check the code by yourself

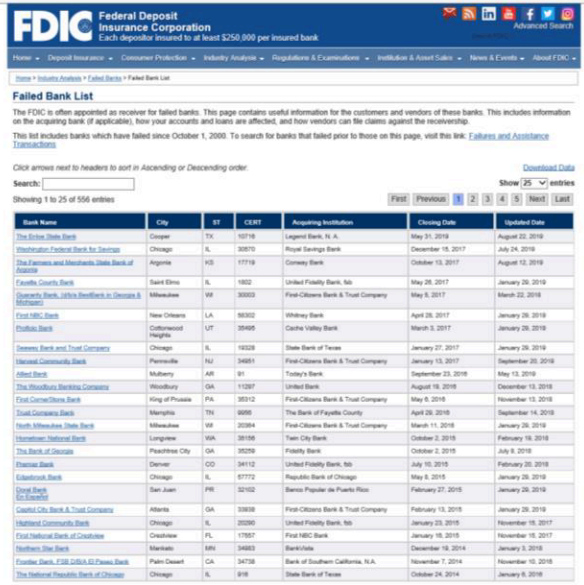
```
# read a json file
data = pd.read_json('ch06/JSON_example.json')
print ("--------------------")
print (data)
print ("--------------------")
print (data.to_json())
print ("--------------------")open
```

XML and HTML files

Extensible Markup Language and
Hypertext Markup Language

- Python has many libraries for reading and writing data in HTML and XML formats.
- Examples include lxml, Beautiful Soup, and html5lib.
- pandas has a built-in function, read_html, which uses libraries like lxml and Beautiful Soup to automatically parse tables out of HTML files as DataFrame objects.
- First, you must install some additional libraries used by read_html

Downloaded an HTML file from the United States FDIC government agency showing bank failures.



FDIC Federal Deposit Insurance Corporation
Each depositor insured to at least $250,000 per insured bank

Advanced Search

Home | Deposit Insurance | Consumer Protection | Industry Analysis | Regulations & Examinations | Institution & Asset Sales | News & Events | About FDIC

Home > Industry Analysis > Failed Banks > Failed Bank List

Failed Bank List

The FDIC is often appointed as receiver for failed banks. This page contains useful information for the customers and vendors of these banks. This includes information on the acquiring bank (if applicable), how your accounts and loans are affected, and how vendors can file claims against the receivership.

This list includes banks which have failed since October 1, 2000. To search for banks that failed prior to those on this page, visit this link: Failures and Assistance Transactions

Click arrows next to headers to sort in Ascending or Descending order.

Download Data

Search:

Show 25 entries

Showing 1 to 25 of 556 entries

First Previous 1 2 3 4 5 Next Last

Bank Name	City	ST	CERT	Acquiring Institution	Closing Date	Updated Date
The Enloe State Bank	Cooper	TX	10716	Legend Bank, N. A.	May 31, 2019	August 22, 2019
Washington Federal Bank for Savings	Chicago	IL	30870	Royal Savings Bank	December 15, 2017	July 24, 2019
The Farmers and Merchants State Bank of Argonia	Argonia	KS	17719	Conway Bank	October 13, 2017	August 12, 2019
Fayette County Bank	Saint Elmo	IL	1802	United Fidelity Bank, fsb	May 26, 2017	January 29, 2019
Guaranty Bank, (d/b/a BestBank in Georgia & Michigan)	Milwaukee	WI	30003	First-Citizens Bank & Trust Company	May 5, 2017	March 22, 2018
First NBC Bank	New Orleans	LA	58302	Whitney Bank	April 28, 2017	January 29, 2019
Proficio Bank	Cottonwood Heights	UT	35495	Cache Valley Bank	March 3, 2017	January 29, 2019
Seaway Bank and Trust Company	Chicago	IL	19328	State Bank of Texas	January 27, 2017	January 29, 2019
Harvest Community Bank	Pennsville	NJ	34951	First-Citizens Bank & Trust Company	January 13, 2017	September 20, 2018
Allied Bank	Mulberry	AR	91	Today's Bank	September 23, 2016	May 13, 2019
The Woodbury Banking Company	Woodbury	GA	11297	United Bank	August 19, 2016	December 13, 2018
First Cornerstone Bank	King of Prussia	PA	35312	First-Citizens Bank & Trust Company	May 6, 2016	November 13, 2018
Trust Company Bank	Memphis	TN	9956	The Bank of Fayette County	April 29, 2016	September 14, 2018
North Milwaukee State Bank	Milwaukee	WI	20364	First-Citizens Bank & Trust Company	March 11, 2016	January 29, 2019
Hometown National Bank	Longview	WA	35156	Twin City Bank	October 2, 2015	February 19, 2018
The Bank of Georgia	Peachtree City	GA	35259	Fidelity Bank	October 2, 2015	July 9, 2018
Premier Bank	Denver	CO	34112	United Fidelity Bank, fsb	July 10, 2015	February 20, 2018
Edgebrook Bank	Chicago	IL	57772	Republic Bank of Chicago	May 8, 2015	January 29, 2019
Doral Bank En Español	San Juan	PR	32102	Banco Popular de Puerto Rico	February 27, 2015	January 29, 2019
Capitol City Bank & Trust Company	Atlanta	GA	33938	First-Citizens Bank & Trust Company	February 13, 2015	January 29, 2019
Highland Community Bank	Chicago	IL	20290	United Fidelity Bank, fsb	January 23, 2015	November 18, 2017
First National Bank of Crestview	Crestview	FL	17557	First NBC Bank	January 16, 2015	November 18, 2017
Northern Star Bank	Mankato	MN	34983	BankVista	December 19, 2014	January 3, 2018
Frontier Bank, FSB D/B/A El Paseo Bank	Palm Desert	CA	34738	Bank of Southern California, N.A.	November 7, 2014	November 10, 2015
The National Republic Bank of Chicago	Chicago	IL	916	State Bank of Texas	October 24, 2014	January 8, 2016

Load <table> data of HTML

- The pandas.read_html function has a number of options, but by default it searches for and attempts to parse all tabular data contained within <table> tags.
- The result is a list of DataFrame objects.

```
# read a HTML file
tables = pd.read_html('ch06/FDICFailedBankList.html')
print (tables)
```

Check the code by yourself

HDF5 Format

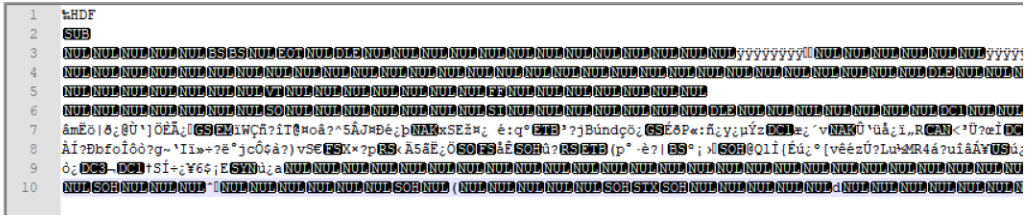
- The “HDF” in HDF5 stands for hierarchical data format.
- HDF5 is a well-regarded file format intended for storing large quantities of scientific array data.
- Each HDF5 file can store multiple datasets and supporting metadata.
- Compared with simpler formats, HDF5 supports compression modes, enabling data with repeated patterns to be stored more efficiently.
- HDF5 can be a good choice for working with very large datasets that don’t fit into memory, as you can efficiently read and write small sections of much larger arrays.

Load HDF5 File

Check the code by yourself

```
# read a HDF file
frame = pd.DataFrame({'a': np.random.randn(100)})
frame.to_hdf('ch06/mydata.h5', key='frame', mode='w')
frame2 = pd.read_hdf('ch06/mydata.h5')
```

mydata.h5



Microsoft Excel Files

- pandas also supports reading tabular data stored in Excel 2003 (and higher) files using pandas.read_excel function.
- Internally these tools use the add-on packages xlrd and openpyxl to read XLS and XLSX files, respectively.
- You may need to install these manually with pip or conda.

Interacting with Web APIs

- Many websites have public APIs providing data feeds via JSON or some other format.
- There are a number of ways to access these APIs from Python.
- One easy-to-use method is the requests package.

Example for Web APIs

- To find the last 30 GitHub issues for pandas on GitHub, we can make a GET HTTP request using the add-on requests library:

Check the code by yourself

```
# find the last 30 GitHub issues for pandas on GitHub
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)
print (resp)
data = resp.json()

print (data[0]['title'])
issues = pd.DataFrame(data, columns=['number', 'title'])
print (issues)
```

Example for Web APIs

- The Response object's json method will return a dictionary containing JSON parsed into native Python objects.

- Each element in data is a dictionary containing all of the data found on a GitHub issue page (except for the comments). We can pass data directly to DataFrame and extract fields of interest

Image file and image processing lib

- Image Processing is a field of knowledge that falls in Computer Vision.
- The premises of machine learning were first laid down by computer vision theory, applying a whole set of techniques to process and analyze imagery data to extract valuable information that computers and machines may use for a wide range of applications.
- Some of the commonly used image processing techniques leveraging a very popular Computer Vision library, OpenCV. We can use the functions of OpenCV lib load image files.

Example for Opencv

```
# load and show image using opencv
import cv2

img = cv2.imread('opencv.png',cv2.IMREAD_GRAYSCALE)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Data Cleaning and Preparation

- During the course or research of doing data analysis and modeling, a significant amount of time is spent on data preparation: loading, cleaning, transforming, and rearranging.
- Such tasks are often reported to take up 80% or more of an analyst's time.
- Sometimes the way that data is stored in files or databases is not in the right format for a particular task.

Handling Missing Data

- Missing data occurs commonly in many data analysis applications. One of the goals of pandas is to make working with missing data as painless as possible.
- For example, all of the statistics on pandas objects exclude missing data by default.
- For numeric data, pandas uses the floating-point value NaN (Not a Number) to represent missing data.

Discussion

- Your machine learning algorithm may report error if you import the data with missing data (nan) to it.

- How should we deal with the missing data by considering the characteristics of the project or data?

Specify NaN and Check it

```
2 import pandas as pd
3 import numpy as np
4 #---------------------------------------
5 # Handling Missing Data
6 string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
7 print ("----------------------")
8 print (string_data)
9 print ("----------------------")
10 print (string_data.isnull())
11 print ("----------------------")
```

```
----------------------
0     aardvark
1    artichoke
2          NaN
3      avocado
dtype: object
----------------------
0    False
1    False
2     True
3    False
dtype: bool
----------------------
```

Filtering Out Missing Data

- There are a few ways to filter out missing data. While you always have the option to do it by hand using pandas.isnull and boolean indexing to filter our the missing data.
- You also can use dropna to filter our the missing data.

```
19 # filter out missing data for series
20 data = pd.Series([1, np.nan, 3.5, np.nan, 7])
21 print("----------------------")
22 print(data)
23 print("----------------------")
24 print(data.dropna())
25 print("----------------------")
```

```
----------------------
0    1.0
1    NaN
2    3.5
3    NaN
4    7.0
dtype: float64
----------------------
0    1.0
2    3.5
4    7.0
dtype: float64
----------------------
```

Filling In Missing Data

- fillna method is a function to fill in the “holes” .

```
39 # filling in missing data
40 df = pd.DataFrame(np.random.randn(7, 3))
41 df.iloc[1:4, 1] = np.nan
42 df.iloc[1:3, 2] = np.nan
43 print("----------------------")
44 print(df)
45 print("----------------------")
46 df1=df.fillna(0)
47 print(df1)
48 print("----------------------")
```

```
----------------------
          0         1         2
0  0.605347  0.148541 -0.283900
1 -0.808930       NaN       NaN
2 -0.179342       NaN       NaN
3 -1.107136       NaN  0.945894
4 -0.861679  1.143359  0.131292
5  0.705228 -0.360142  0.912524
6 -0.099595  0.163215 -0.857075
----------------------
          0         1         2
0  0.605347  0.148541 -0.283900
1 -0.808930  0.000000  0.000000
2 -0.179342  0.000000  0.000000
3 -1.107136  0.000000  0.945894
4 -0.861679  1.143359  0.131292
5  0.705228 -0.360142  0.912524
6 -0.099595  0.163215 -0.857075
----------------------
```

Filling In Missing Data

```
----------------------
          0         1         2
0  0.605347  0.148541 -0.283900
1 -0.808930       NaN       NaN
2 -0.179342       NaN       NaN
3 -1.107136       NaN  0.945894
4 -0.861679  1.143359  0.131292
5  0.705228 -0.360142  0.912524
6 -0.099595  0.163215 -0.857075
----------------------
```

→

```
----------------------
          0         1         2
0  0.605347  0.148541 -0.283900
1 -0.808930  0.148541 -0.283900
2 -0.179342  0.148541 -0.283900
3 -1.107136  0.148541  0.945894
4 -0.861679  1.143359  0.131292
5  0.705228 -0.360142  0.912524
6 -0.099595  0.163215 -0.857075
----------------------
```

Exercise 1:

- How to use fillna to get the result like above

Removing Duplicates

```
53 #---------------------------------------
54 #Removing Duplicates
55 data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
56                       'k2': [1, 1, 2, 3, 3, 4, 4]})
57 print("---------------------")
58 print(data)
59 print("---------------------")
60 print(data.duplicated())
61 print("---------------------")
62 print(data.drop_duplicates())
63 print("---------------------")
64 print(data.drop_duplicates(['k1']))
65
```

```
---------------------
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
6  two   4
---------------------
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
---------------------
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
---------------------
    k1  k2
0  one   1
1  two   1
```

Combining and Merging Datasets

- pandas.merge connects rows in DataFrames based on one or more keys.

- pandas.concat concatenates or “stacks” together objects along an axis.

pandas.merge

```
52 #---------------------------------------
53 #Combining and Merging Datasets
54 # many to one
55 df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
56                     'data1': range(7)})
57 df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
58                     'data2': range(3)})
59 print("-----------------------------")
60 print(df1)
61 print("-----------------------------")
62 print(df2)
63 print("-----------------------------")
64 df3 = pd.merge(df1, df2)
65 print(df3)
66 print("-----------------------------")
67 df4 = pd.merge(df1, df2, on='key')
68 print(df4)
69 print("-----------------------------")
70 df5 = pd.merge(df1, df2, how='right')
71 print(df5)
72 print("-----------------------------")
73 df6 = pd.merge(df1, df2, how='left')
74 print(df6)
75 print("-----------------------------")
76
```

```
-----------------------------
   data1 key
0      0   b
1      1   b
2      2   a
3      3   c
4      4   a
5      5   a
6      6   b
-----------------------------
   data2 key
0      0   a
1      1   b
2      2   d
-----------------------------
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
-----------------------------
   data1 key  data2
0      0   b      1
1      1   b      1
2      6   b      1
3      2   a      0
4      4   a      0
5      5   a      0
-----------------------------
   data1 key  data2
0    0.0   b      1
1    1.0   b      1
2    6.0   b      1
3    2.0   a      0
4    4.0   a      0
5    5.0   a      0
6    NaN   d      2
-----------------------------
   data1 key  data2
0      0   b    1.0
1      1   b    1.0
2      2   a    0.0
3      3   c    NaN
4      4   a    0.0
5      5   a    0.0
6      6   b    1.0
-----------------------------
```

pandas.merge connects rows in DataFrames based on one or more keys (shared column name).

Concatenating Along an Axis

- Another kind of data combination operation is referred to interchangeably as concatenation, or stacking.

- Calling concat with these objects in a list glues together the values and indexes
- By default concat works along axis=0, producing another Series. If you pass axis=1, the result will instead be a DataFrame (axis=1 is the columns)

Concatenating Along an Axis

```
77 s1 = pd.Series([0, 1], index=['a', 'b'])
78 s2 = pd.Series([2, 3, 4], index=['c', 'd', '
79 s3 = pd.Series([5, 6], index=['f', 'g'])
80 s4=pd.concat([s1, s2, s3])
81 print("----------------------------")
82 print(s1)
83 print("----------------------------")
84 print(s2)
85 print("----------------------------")
86 print(s3)
87 print("----------------------------")
88 print(s4)
89 print("----------------------------")
90 s5=pd.concat([s1, s2, s3], axis=1)
91 print(s5)
92 print("----------------------------")
93
```

=0

```
----------------------------
a    0
b    1
dtype: int64
----------------------------
c    2
d    3
e    4
dtype: int64
----------------------------
f    5
g    6
dtype: int64
----------------------------
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
----------------------------
     0    1    2
a  0.0  NaN  NaN
b  1.0  NaN  NaN
c  NaN  2.0  NaN
d  NaN  3.0  NaN
e  NaN  4.0  NaN
f  NaN  NaN  5.0
g  NaN  NaN  6.0
----------------------------
```

Time Series

- Time series data is an important form of structured data in many different fields, such as finance, economics, neuroscience, and physics.
- Anything that is observed or measured at many points in time forms a time series.
- Many time series are *fixed frequency*, which is to say that data points occur at regular intervals according to some rule, such as every 15 seconds, every 5 minutes, or once per month.
- Time series can also be *irregular* without a fixed unit of time or offset between units. How you mark and refer to time series data depends on the application.

Time Series

- *Timestamps*, specific instants in time
- Fixed *periods*, such as the month January 2007 or the full year 2010
- *Intervals* of time, indicated by a start and end timestamp. Periods can be thought of as special cases of intervals

The simplest and most widely used kind of time series are those indexed by timestamp.

Python standard library

- The Python standard library includes data types for date and time data, as well as calendar-related functionality. The datetime, time, and calendar modules are the main places to start.

```
#----------------------------------------
now = datetime.now()
print("----------------------------")
print(now)
print("----------------------------")
print(now.year)
print(now.month)
print(now.weekday()) #0-6
print("----------------------------")
dateFormat = datetime(2017, 9, 25, 14, 5, 52, 72973)
print(dateFormat)
print("----------------------------")
delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)
print(delta)
print("----------------------------")
#----------------------------------------
start = datetime(2011, 1, 7)
newTime = start + timedelta(12)
print(newTime)
print("----------------------------")
start = datetime(2011, 1, 7,0,0,0)
newTime = start + timedelta(12)
print(newTime)
#----------------------------------------
```

```
----------------------------
2019-10-12 20:30:48.461000
----------------------------
2019
10
5
----------------------------
2017-09-25 14:05:52.072973
----------------------------
926 days, 15:45:00
----------------------------
2011-01-19 00:00:00
----------------------------
2011-01-19 00:00:00
```

Exercise

```python
import pandas as pd

# Load customer data
customer_master = pd.read_csv('customer_master.csv')
print(customer_master.head())

# Load product item data
item_master = pd.read_csv('item_master.csv')
print(item_master.head())

# Load transaction data
transaction_1 = pd.read_csv('transaction_1.csv')
print(transaction_1.head())

# Load transaction details 1
transaction_detail_1 = pd.read_csv('transaction_detail_1.csv')
print(transaction_detail_1.head())

# Load transaction data 2
transaction_2 = pd.read_csv('transaction_2.csv')
print(transaction_2.head())

# Load transaction details 2
transaction_detail_2 = pd.read_csv('transaction_detail_2.csv')
print(transaction_detail_2.head())

# Q(1): use one-line code to combine two transaction data
print(transaction.head())


# Q(2): use one-line code to combine two transaction_detail data

print(transaction_detail.head())

# Q(3): use one-line code to combine transaction and transaction_detail
# the generated data should be indexed based on the index of transaction_detail,
# and the "payment_date", "customer_id" should be integrated with the information
# in transaction_detail

print(join_data.head())
```

Reference

- Devin Pickell, Structured vs Unstructured Data – What's the Difference?, https://learn.g2.com/structured-vs-unstructured-data.
- 21 Places to Find Free Datasets for Data Science Projects, https://www.dataquest.io/blog/free-datasets-for-projects/
- Will Koehrsen, Wikipedia Data Science: Working with the World's Largest Encyclopedia, https://towardsdatascience.com/wikipedia-data-science-working-with-the-worlds-largest-encyclopedia-c08efbac5f5c
- McKinney, Wes. Python for data analysis: Data wrangling with Pandas, NumPy, and IPython. " O'Reilly Media, Inc.", 2012.
- Provost, Foster, and Tom Fawcett. Data Science for Business: What you need to know about data mining and data-analytic thinking. O'Reilly Media, Inc., 2013.

Data Science

Week 4

Statistics for Data Science