DEDA Digital Economy & Decision Analytics

Cathy Yi-Hsuan Chen

Wolfgang Karl Härdle

Ladislaus von Bortkiewicz Chair of Statistics

C.A.S.E.-Center for Applied Statistics and Economics

International Research Training Group

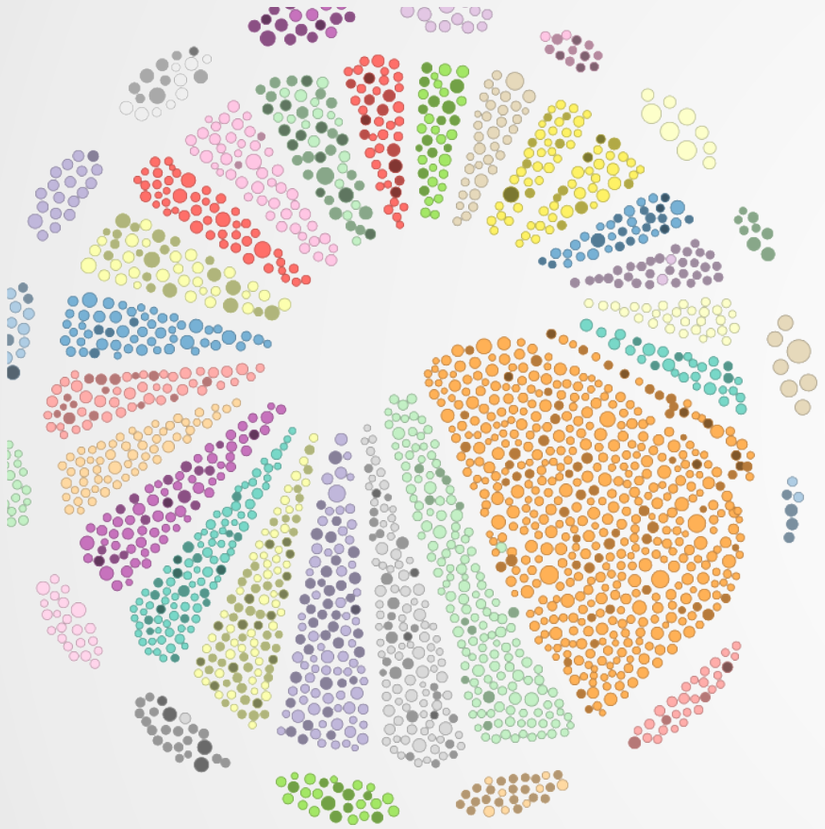
Humboldt-Universität zu Berlin

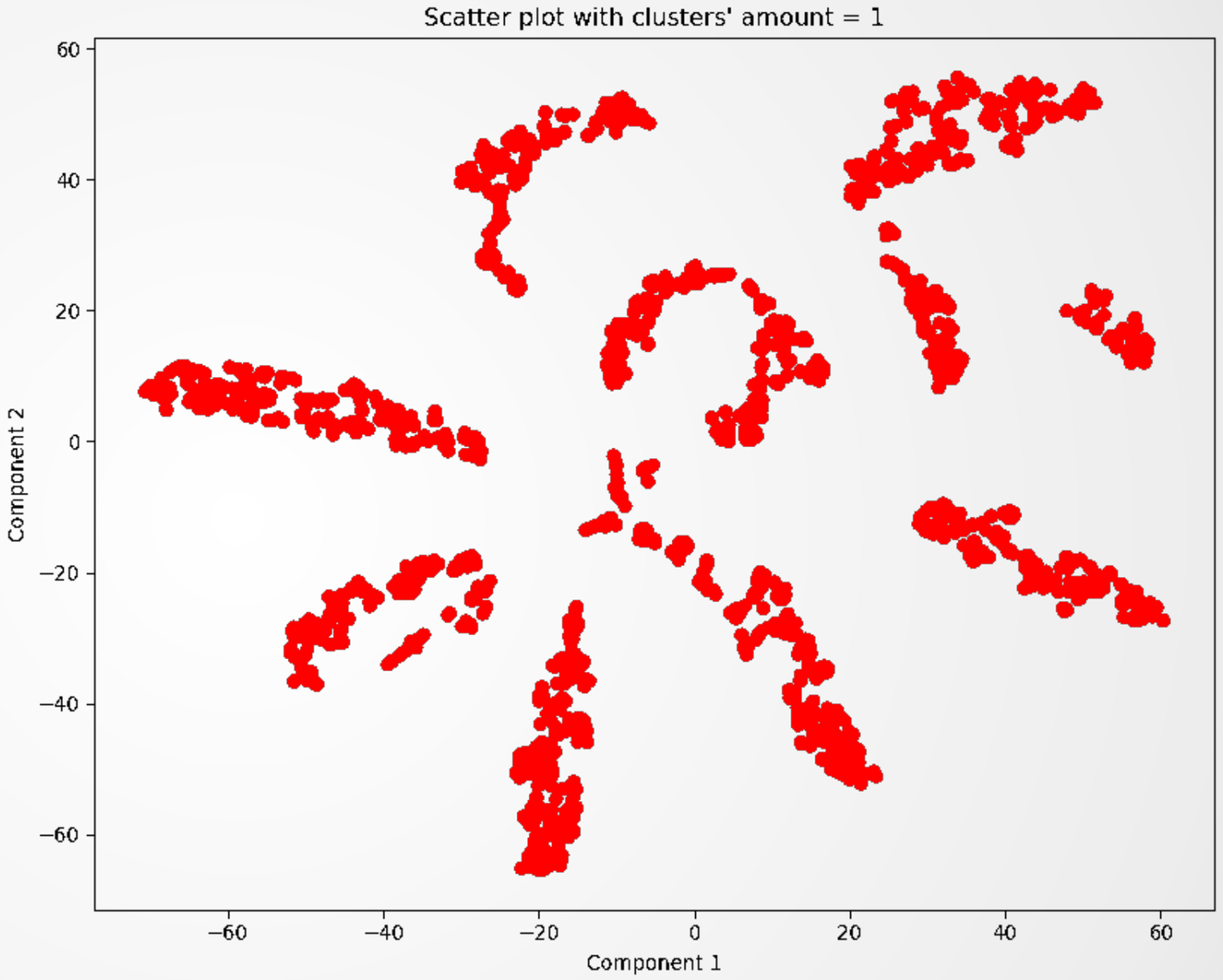
lvb.wiwi.hu-berlin.de

www.case.hu-berlin.de

irtg1792.hu-berlin.de

Smart Data Analytics





Digital Economy & Decision Analytics

Image segmentation

Original

2 Clusters

4 Clusters

8 Clusters

DEDA - outline

1. **Introduction of Python**

   Python Installation, IDE and Basic Syntax

   Data Structure and Commonly Used Operations

2. **Webpage Scraping in Python**

   Introduction of Webpage Scraping in Python

   Webpage Scraping Framework in Python

3. **Statistics and Finance in Python**

   Basic Statistics and Visualization in Python

   Python in Finance

4. **Machine Learning Methods**

   Introduction to ML Machine Learning

   Statistics and Finance

DEDA Schedule and Course Outline

C YH Chen, WK Härdle

1. Unit_1: How to install and run Python…
2. Packages, Data Structure and Commonly Used Operations
3. Unit_2 = The latter part of:

https://github.com/QuantLet/DEDA_Class_2017/tree/master/DEDA_Class_2017_Python_Introduction

+

https://github.com/QuantLet/DEDA_Class_2017/tree/master/DEDA_Class_2017_Condition%26Iteration

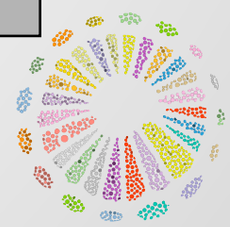
+

https://github.com/QuantLet/DEDA_Class_2017/tree/master/DEDA_Class_2017_InputOutput20181031 ….

4. more to come …

Data Structure _ list

```
# Using list makes Python code simple.
natr_language = ['English', 'German', 'Chinese']
prog_language = ['C++', 'Java', 'C#']

# how many elements in a list?
len(natr_language)  # 3
# how many elements in a string?
len(natr_language[0])  # 7

# adding elements into the list
# append() allows you to add 1 element at the end of the list
natr_language.append('Spanish')  # ['English', 'German', 'Chinese', 'Spanish']

# insert() allows you to add 1 element at arbitrary place
prog_language.insert(0, 'Python')  # ['Python', 'C++', 'Java', 'C#']

# extend() allows you to add multiple elements at the end of the list
python = ['python 2.7', 'python 3.6']
prog_language.append(python)  # ['Python', 'C++', 'Java', 'C#', ['python 2.7', 'python 3.6']]
more_languages = ['Japanese', 'Korean']
natr_language.extend(more_languages)
# ['English', 'German', 'Chinese', 'Spanish', 'Japanese', 'Korean']
```



Data Structure _ list

```
# Using list makes Python code simple.
# remove elements
prog_language.remove(python)  # ['Python', 'C++', 'Java', 'C#']
natr_language.pop(-1)
# 'Korean' pops out. ['English', 'German', 'Chinese', 'Spanish', 'Japanese']
del prog_language[-1]  # ['Python', 'C++', 'Java']

# reorder in list
numbers = [43, 11, 32, 95, 22]
numbers.reverse()  # [22, 95, 32, 11, 43]
numbers.sort()  # [11, 22, 32, 43, 95]
numbers.sort(reverse=True)  # [95, 43, 32, 22, 11]
sorted_number = sorted(numbers)
# sorted function can return a new list instead of altering the original list
# basic search in list
min_num = min(numbers)  # 11
max_num = max(numbers)  # 95
sum_num = sum(numbers)  # 203
index_num = numbers.index(32)  # 2
```

Data Structure _ list

```
# Using list makes Python code simple.
# iterate in list
for lang in prog_language:
    print(lang)
# iterate in list with index -> 0.English, 1.German, .....
for num, lang in enumerate(natr_language):
    print(f'{num}. {lang}')

# list to string
natr_language_str = ', '.join(natr_language)
# 'Spanish, Japanese, German, English, Chinese'
natr_language_new = natr_language_str.split(', ')
# ['Spanish', 'Japanese', 'German', 'English', 'Chinese']

# looping in the list. The basic syntax is:
# [func(ele) for func(ele) in a_list if func(ele)]
# For example: (Can you change the cmds below from ese to e ? Or even h ?)
people = [language + ' People' for language in natr_language_new if
language.endswith('ese')]
# format number in list to 2 digit
num_seq = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
num_seq_db = [format(num, '02d') for num in num_seq]
```

Data Structure _ tuple and set

```
# tuple and set are two basic structures with different features to list.

# tuple is very alike list, but tuple is immutable
person_list = ['Jon Dow', '06-04-2000', 'Male', 'U.S.A']
person_tuple = ('Jon Dow', '06-04-2000', 'Male', 'U.S.A')
person_list[0] = 'Allan Lee'  # ['Allan Lee', '06-04-2000', 'Male', 'U.S.A']
person_tuple[0] = 'Allan Lee'  # tuple' object does not support item assignment

# set
countries_1 = {'China', 'Korea', 'Japan', 'Turkey', 'Singapore', 'Russia', 'Japan'}
countries_2 = {'UK', 'Germany', 'France', 'Spain', 'Italy', 'Russia', 'Turkey'}

print(countries_1)  # sets will drop duplicated elements automatically
count_inter = countries_1.intersection(countries_2)  # {'Russia', 'Turkey'}
count_1_diff = countries_1.difference(countries_2)
# {'China', 'Japan', 'Korea', 'Singapore'}
count_2_diff = countries_2.difference(countries_1)
# {'France', 'Germany', 'Italy', 'Spain', 'UK'}
countries_new = countries_1.union(countries_2)
# merge two sets into 1 and without duplicates
```

Data Structure _ dict

```
# Like a real dictionary, dict type is form by 2 parts, a unique key and values for a key.

# initial a dict
profile = {'name': 'Anna', 'birth': '10-05-2000', 'gender': 'female', 'height': 1.70}
# find the value by using the key
print(profile['name'])  # 'Anna'
# get all the keys and put into a list
keys = list(profile.keys())  # ['name', 'birth', 'gender', 'height']
# get all values and put into a list
values = list(profile.values())  # ['Anna', '10-05-2000', 'female', 1.7]
# get all key-value pairs and put into a list
key_value = list(profile.items())
# add a key with value
profile['weight'] = 55
# change value of a key
profile['name'] = 'Yoki'
# alter multiple keys and values at a time
profile.update({'birth': '01-10-2001', 'tel': '123-312'})

# loop with key and value
for key, value in profile.items():
    print(f'{key}: {value}')
```

Conditional Execution

```
"""
Condition
"""
# if() takes 1 argument to evaluate if the argument is True or False
bool_seq = [True, False, 0, 1, -1, 100, '', (), [], {}, [False], [True]]
# False, 0, empty sequence will be evaluated as False
for bool_smb in bool_seq:
    if bool_smb:
        print(bool_smb, ": This is True")
    elif not bool_smb:
        print(bool_smb, ": This is False")

# True, False, and, or
print(True or False)  # True
print(True and False)  # False
print(True or True)  # True
print(True and True)  # True
print(False and False)  # False
print(False or False)  # False
```

Conditional Execution

```
"""
Condition
"""
# operator "is" is not "=="
# "is" is used to evaluate if two 2 variables pointed to the same object, "==" is used to evaluate the equivalence of 2 variables

bool_seq_diff = [True, False, 0, 1, -1, 100, '', (), [], {}, [False], [True]]
print(bool_seq == bool_seq_diff)  # True
print(bool_seq is bool_seq_diff)  # False
bool_seq_same = bool_seq
print(bool_seq == bool_seq_same)  # True
print(bool_seq is bool_seq_same)  # True

# 2 variables pointed to the same object will vary simultaneously
bool_seq.append(1)
print(bool_seq)
print(bool_seq_same)
```

Conditional Execution

```
"""
Condition
"""


# what if I want to create different object but don't want to copy the values manually?
# The answer is deep copy, different data structure may provide different methods.
# Further more see: https://docs.python.org/3/library/copy.html
bool_seq_2 = bool_seq.copy()
print(bool_seq_2 is bool_seq)  # False

# using id(), you can check the memory address of a variable
id(bool_seq)
# the same object will have the same memory address
```

Iteration

```
"""
Iteration
"""
iterate_object = ['\t', 'Digital', 'Economy', 'and', 'Decision', 'Analytics', '\n', 'in',
'Python']

"""
for loop
"""
# Important: Try not using index in the loop while do it in a pythonic way
# This is a NOT recommended way to iterate, because it's ugly and less efficient.
print("\n\t===Example 1===")
for i in range(len(iterate_object)):
    print(i)
    print(iterate_object[i])

# Using any name you want in the for loop
print("\n\t===Example 2===")
for word in iterate_object:
    print(word)
```

Iteration

```
"""
Iteration
"""


# what if I want to use the index?
print("\n\t===Example 3===")
for idx, word in enumerate(iterate_object):
    print(idx, word)


# using the keyword "continue" to skip a loop and key word "break" to stop whole loop.
print("\n\t===Example 4===")
for word in iterate_object:
    if word == '\n':
        continue
    elif word == 'in':
        print("****break in here****")
        break
    else:
        print(word)
```

Iteration

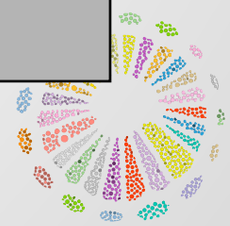
```
"""
Iteration
"""


# a loop embed in another loop
print("\n\t===Example 5===")
for word in iterate_object:
    if word == 'in':
        break
    else:
        for letter in word:
            if letter.isupper() is True:
                print(letter)

# infinite loop
num = 0
while True:
    print(num)
    num +=1
```

Import Packages

```
# Import packages

# Using "import" can input packages (modules), .py files from defined paths
# 2 ways to import
import os
path_direct = os.getcwd()
os.chdir(path_direct + '/DEDA_Class_2017_InputOutput')
# You can check, create, delete, rename your files and directories
# Hint: path.exists(), listdir(), mkdir(), makedirs(), remove(), removedirs(), rename(),
walk()

# Give the package an alias
import numpy as np
np.power(2, 10)  # 1024

# Instead of importing whole package, import only 1 method in the package
from pandas import DataFrame
some_info = {'name': ['Alice', 'Bob', 'Clark', 'Douglas'],
             'age': [5, 10, 3, 22]}
df = DataFrame(some_info)
```

Read & Write Files

```
"""
Read and write file
"""
# Using build-in function, open(), to open the file and using close() to close the file
shakespeare = open('shakespeare.txt', 'r', encoding='utf-8')
for string in shakespeare:
    print(string)
shakespeare.close()
# You will get 10 segments of the shakespeare text
# In python, we usually use syntax "with open() as container_name" to load the content
# There are 3 basic containers here:
with open('shakespeare.txt', 'r') as shakespeare_read:
# read(n) method will put n characters into a string.  Below you get 10 blanks! why?
    shakespeare_string_10 = shakespeare_read.read(10)
    shakespeare_string = shakespeare_read.read()

with open('shakespeare.txt', 'r') as shakespeare_read:
    # readline() method will read one line once.
    print(shakespeare_read.readline(), end='*')
    print(shakespeare_read.readline(), end='*')
    print(shakespeare_read.readline(), end='*')
```

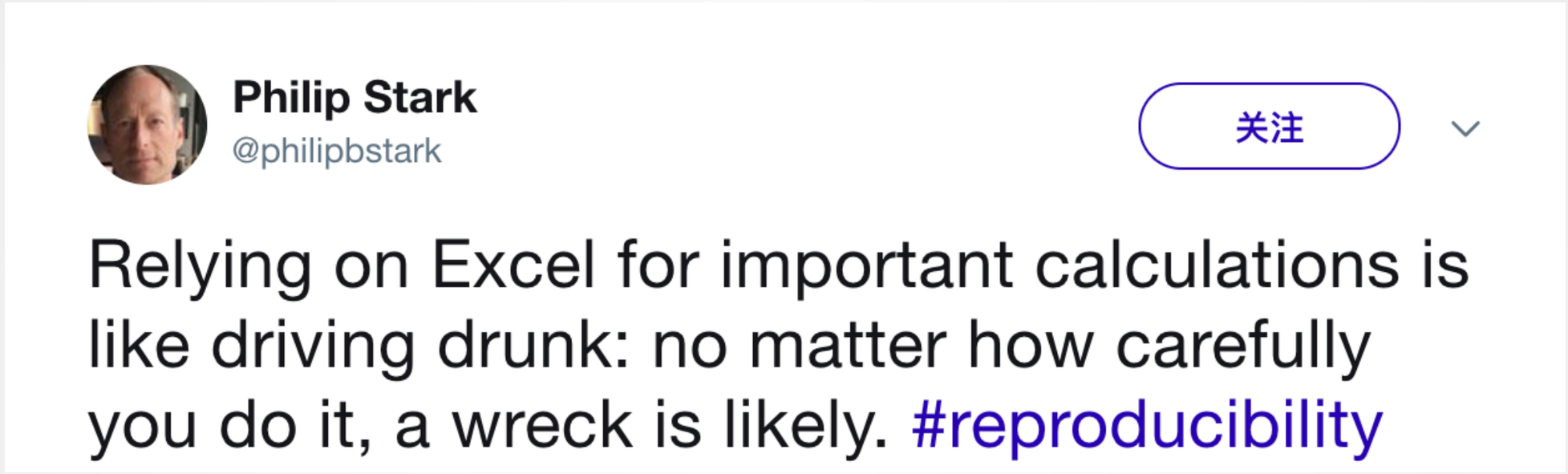


Read & Write Files

```
# Read and write file
# readlines() method will put content into a list, every line is a string in the list
with open('shakespeare.txt', 'r') as shakespeare_read:
    shakespeare_lines = shakespeare_read.readlines()
    print(shakespeare_lines)

for line in shakespeare_lines:
    print(line)

# Using pickle to serialize data, save as binary format
# Create a standard normal distribution data set
temp_data = np.random.normal(size=100000)
temp_data = list(temp_data)
import matplotlib.pyplot as plt
plt.hist(temp_data, 100)    # nr of bins = 100 in the plt.hist method
import pickle
# If you are writing plain text, you can use 'w'
# If you want to save as binary format, you should use 'wb'
with open('temp.pkl', 'wb') as temp_file:
    pickle.dump(temp_data, temp_file)
```

Structured Data I/O

If you are still using Excel to process data, you should stop it now.



Structured Data I/O

```
# Input and output structured data

import pandas as pd
# Pandas supports most of the common structured data formats
# The read_csv method can take more arguments to satisfy your need
# For example, you can specify the delimiter and decimal style
# further more see: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

apple_stock = pd.read_csv('AAPL.csv', index_col='date', parse_dates=True)
# Pandas will read files as DataFrame type
# This is a very powerful data structure that you can do almost everything to the data.

type(apple_stock)
# For example, easily slicing rows and selecting columns
apple_stock_2013 = apple_stock.loc[apple_stock.index.year == 2013, ['low', 'high', 'open', 'close', 'volume']]

# shape of DataFrame
print(apple_stock_2013.shape)
```

Structured Data I/O

```
# sorting by value
apple_stock_2013.sort_values(by='volume', ascending=False, inplace=True)
print(apple_stock_2013)

# check monotonicity increasing through time
print(apple_stock_2013.index.is_monotonic_increasing)

# sorting by index
apple_stock_2013.sort_index(axis=0, ascending=True, inplace=True)

# reset index as numeric
apple_stock_2013.reset_index(drop=False, inplace=True)

# add a row : may be you have to import datetime…
new_row_1 = {'date': dt.datetime(2013, 8, 31), 'low': 500, 'high': 510}
apple_stock_2013 = apple_stock_2013.append(new_row_1, ignore_index=True)

# Check null values
nan_rows = apple_stock_2013[apple_stock_2013['open'].isna()]

# remove null values
apple_stock_2013.dropna(axis=0, how='any', subset=['volume', 'close'],
inplace=True)
```

Structured Data I/O

```
# duplicate a row
new_row = {'date': dt.datetime(2013, 8, 30)}
apple_stock_2013 = apple_stock_2013.append(new_row, ignore_index=True)
apple_stock_2013.fillna(method='ffill', inplace=True)

# set a column as index
apple_stock_2013.set_index(keys='date', drop=True, append=False, inplace=True)

# duplicates timestamp operations
print(apple_stock_2013.index.has_duplicates)
apple_stock_2013.index.duplicated()

# operation
# way1, potential risk
way1 = apple_stock_2013.drop_duplicates(keep='first', subset=['low', 'high',
'open', 'close', 'volume'])
# way2, drop by index, row operation. ~, take inverse
apple_stock_2013 = apple_stock_2013[~ (apple_stock_2013.index.duplicated())]

# a simple build-in plot function of pandas
apple_stock_2013['open'].plot()

# Save the new data as json format
apple_stock_2013.to_json('AAPL_2013.json')
apple_stock_2013.to_csv('test.csv')
```

DEDA Digital Economy & Decision Analytics

Cathy Yi-Hsuan Chen

Wolfgang K. Härdle

Ladislaus von Bortkiewicz Chair of Statistics

C.A.S.E.-Center for Applied Statistics and Economics

International Research Training Group

Humboldt-Universität zu Berlin

lvb.wiwi.hu-berlin.de

www.case.hu-berlin.de

irtg1792.hu-berlin.de