Preparation for Lecture and Tutorial

- Start Anaconda Navigator (Remotely or Locally)
  - Follow Part 1.1 of instructions for Tutorial 1
- Download files of the lecture from Blackboard
  - Extract and store them in E:\LGT3109: rates.csv, input.csv, name_address.csv, example.csv, exampleWithHeader.csv, sales.csv
- Download files of the tutorial from Blackboard
  - Follow Part 1.2 of instructions for Tutorial 1
  - For today's tutorial, download the zip file from Tutorial 10 in the Blackboard, and extract the zip file in working directory E:\LGT3109\T10
- Launch Python IDLE from Anaconda Navigator
  - Follow Part 2.1 of instructions for Tutorial 1
- Launch Jupyter Notebook from Anaconda Navigator
  - Follow Part 3.1 of instructions for Tutorial 1

The session will be recorded!

LGT3109
Introduction to Coding for Business with Python
(Week 10)
Acquiring and Exploring Data -
Basics of Data Analytics

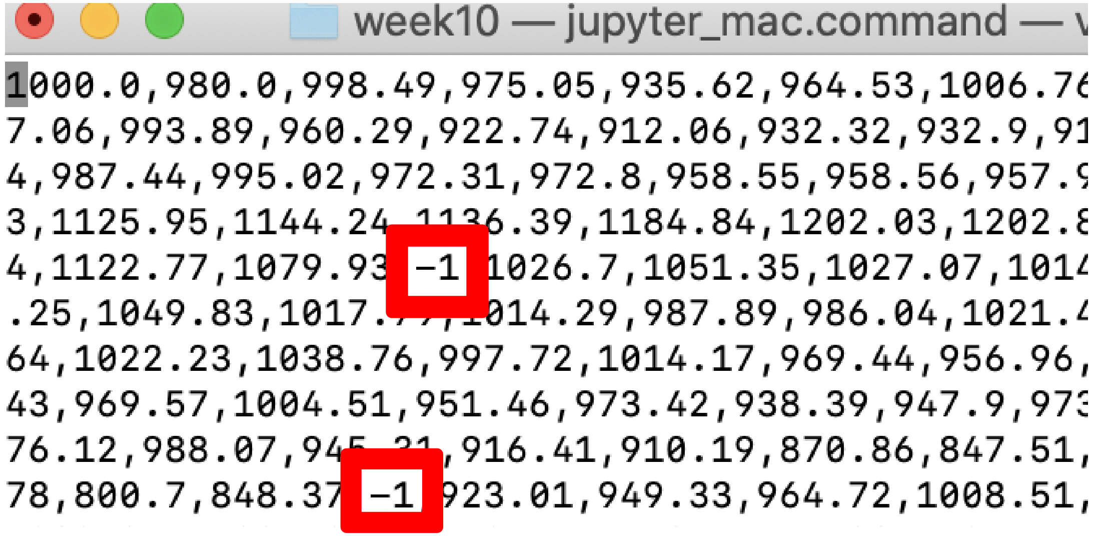
Professor Zhou Xu

Dept. LMS, PolyU

CASE STUDY: EXPLORE SHIPPING RATES

Case Study: Explore shipping rates

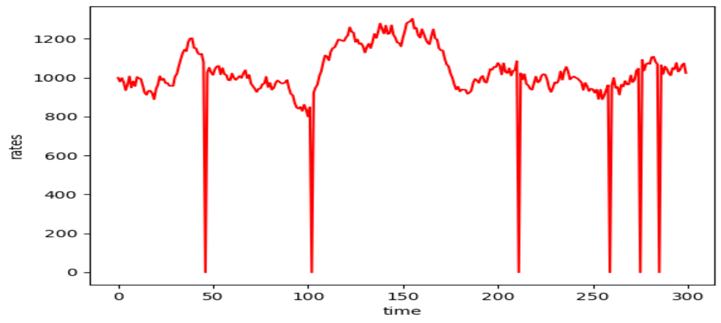
- Company FE needs to explore the trend of shipping rates based on the past 300-day historical data stored in a text file (rates.csv)
  - The text file has one line of 300 float numbers separated by commas
  - Rates may be missing for some days, where -1 are placed
- Company FE needs to
  - Access the data
  - Fix missing data
  - Compute summary statistics
  - Plot a line chart to observe trends



```
1000.0,980.0,998.49,975.05,935.62,964.53,1006.76
7.06,993.89,960.29,922.74,912.06,932.32,932.9,91
4,987.44,995.02,972.31,972.8,958.55,958.56,957.9
3,1125.95,1144.24,1136.39,1184.84,1202.03,1202.8
4,1122.77,1079.93,-1,1026.7,1051.35,1027.07,1014
.25,1049.83,1017.77,1014.29,987.89,986.04,1021.4
64,1022.23,1038.76,997.72,1014.17,969.44,956.96,
43,969.57,1004.51,951.46,973.42,938.39,947.9,973
76.12,988.07,945.21,916.41,910.19,870.86,847.51,
78,800.7,848.37,-1,923.01,949.33,964.72,1008.51,
```

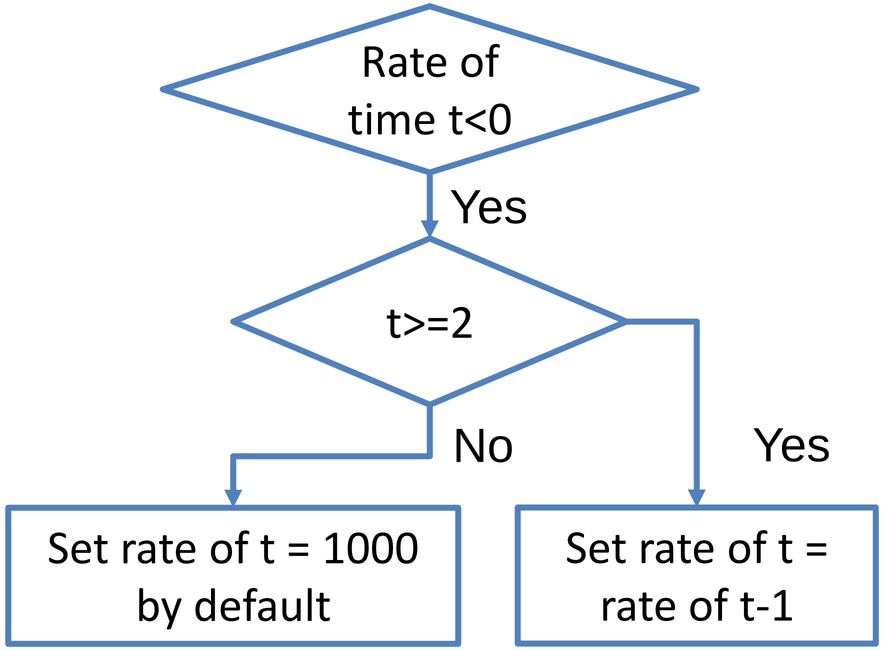
Case Study: Algorithm and Data Structure

- Company FE needs to
  1. Access/acquire the data
     - open(), read(), strip()
     - split(), float()
  2. **Fix missing data**
  3. Compute summary statistics
     - Mean, min, max, standard deviation
  4. Plot a line chart to observe trends
- Data structure
  – rates: list of rates
- Let’s work on Tasks 1-3 as warmup



```
Mean:  1038.9854666666668
Standard deviation:  104.98354623805555
Max:  1303.5
Min:  800.7
```

How to fix missing data?



Rate of time t<0

Yes

t>=2

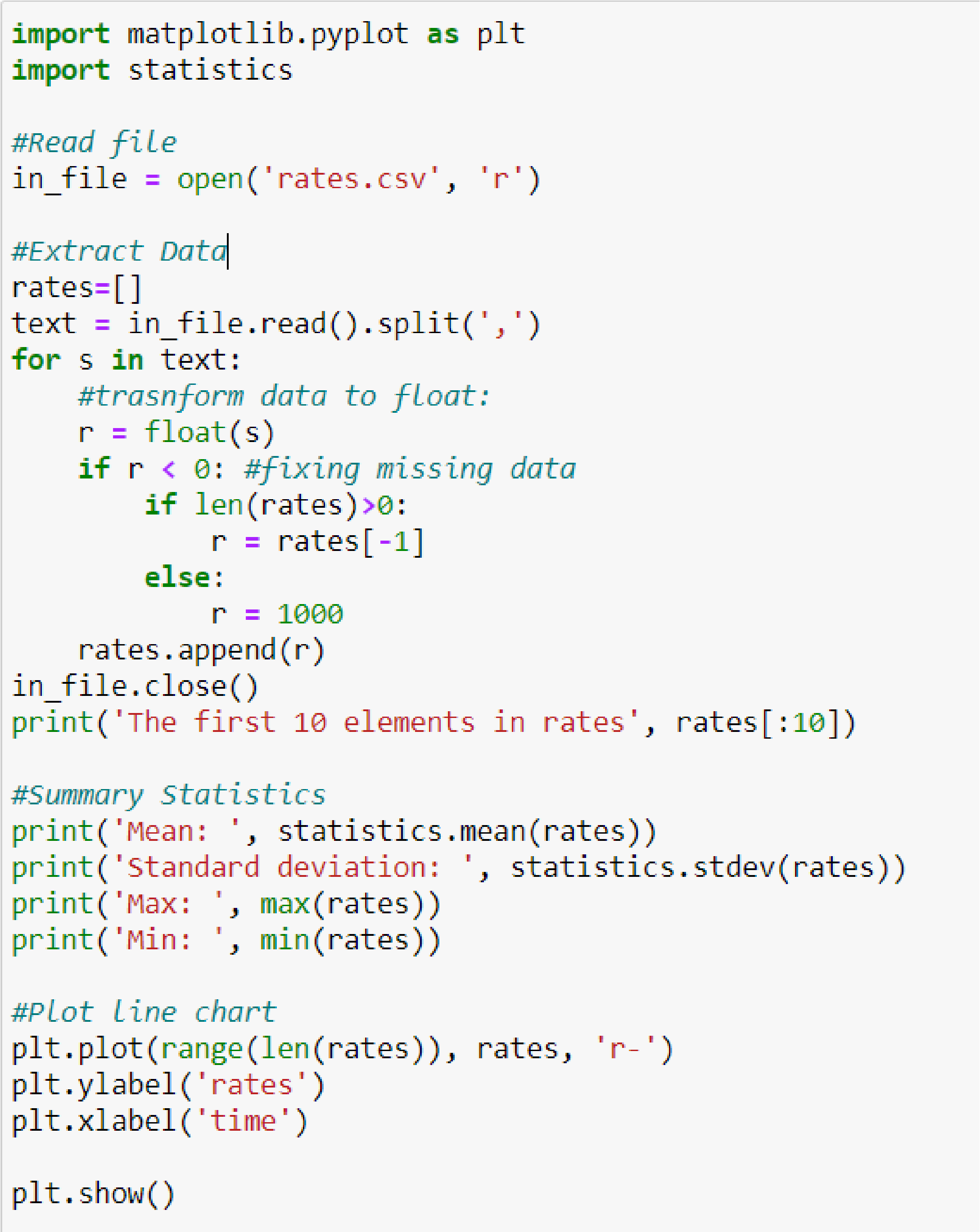
No

Yes

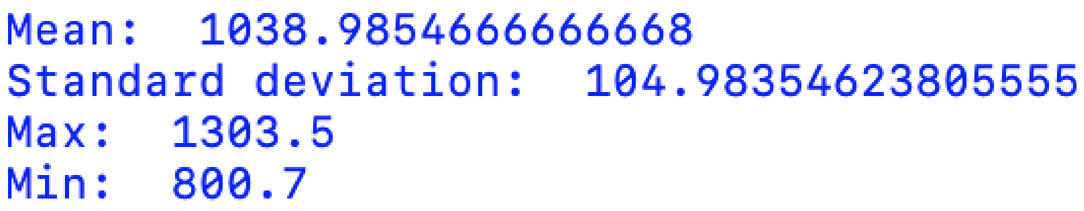
Set rate of t = 1000 by default

Set rate of t = rate of t-1

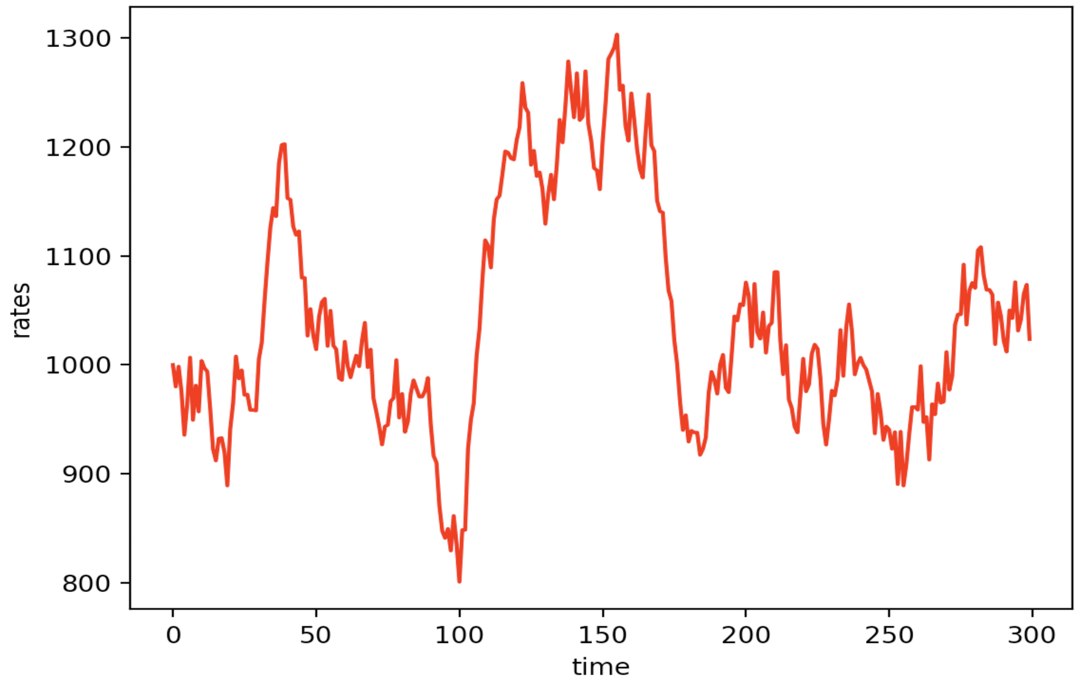
Case Study:
Code and Results



```python
import matplotlib.pyplot as plt
import statistics

#Read file
in_file = open('rates.csv', 'r')

#Extract Data
rates=[]
text = in_file.read().split(',')
for s in text:
    #trasnform data to float:
    r = float(s)
    if r < 0: #fixing missing data
        if len(rates)>0:
            r = rates[-1]
        else:
            r = 1000
    rates.append(r)
in_file.close()
print('The first 10 elements in rates', rates[:10])

#Summary Statistics
print('Mean: ', statistics.mean(rates))
print('Standard deviation: ', statistics.stdev(rates))
print('Max: ', max(rates))
print('Min: ', min(rates))

#Plot line chart
plt.plot(range(len(rates)), rates, 'r-')
plt.ylabel('rates')
plt.xlabel('time')

plt.show()
```



```
Mean:  1038.9854666666668
Standard deviation:  104.9835462380555
Max:  1303.5
Min:  800.7
```



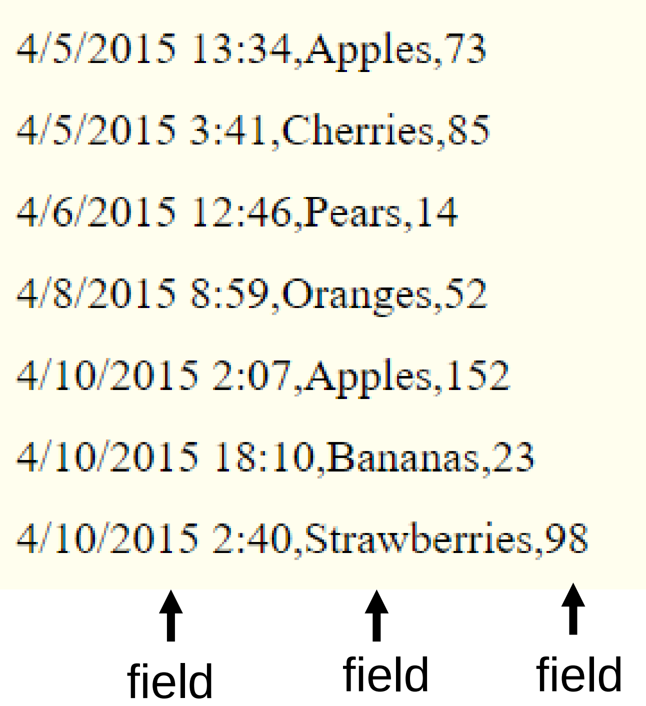
ACQUIRING DATA

Acquiring data

- Acquiring data from data sources is necessary before data analysis
- There are many sources to access data
  - Input
    - module: sys
  - Text files
    - module: csv, pandas
  - Excel files
    - module: openpyxl, pandas
  - HTML files
    - modules: requests, bs4, pandas
  - Database
    - modules: sqlite3, pandas
  - JSON (JavaScript Object Notation)
    - module: json, pandas
  - API (application programming interface) – called to obtain data from internet (e.g., X …)
    - modules: requests, json

Acquiring data from Delimited Files

- Text files for data analysis are very often either comma-separated or tab-separated
  - Each line has several fields, with a comma or a tab indicating where one field ends and the next field starts
- Such delimited files are often in CSV (comma-separated values) format
  - Each line in a CSV file represents a row in the spreadsheet, and commas separate the cells in the row.

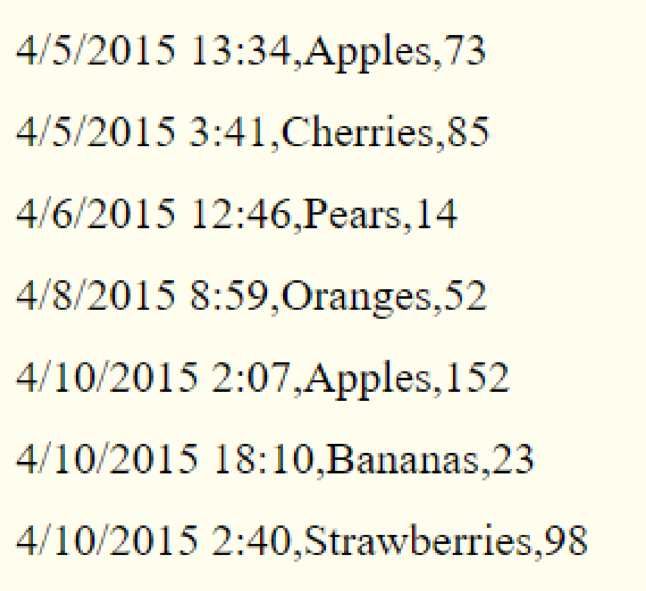


```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

field
field
field

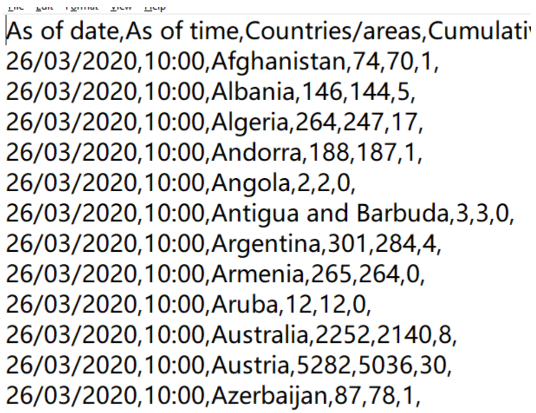
Advantages of CSV Flies

- CSV files are simple, but lacking many of the features of an Excel spreadsheet. For example, CSV files:
- Don’t have types for their values—everything is a string
- Don’t have settings for font size or color
- Don’t have multiple worksheets
- Can’t specify cell widths and heights
- Can’t have merged cells
- Can’t have images or charts embedded in them



```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

Covid-19 data

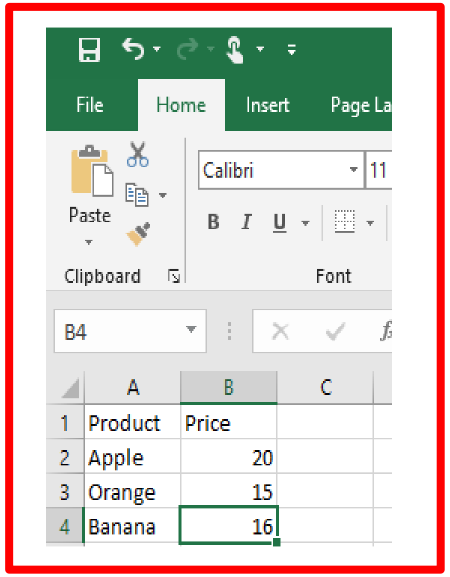


```
As of date,As of time,Countries/areas,Cumulati
26/03/2020,10:00,Afghanistan,74,70,1,
26/03/2020,10:00,Albania,146,144,5,
26/03/2020,10:00,Algeria,264,247,17,
26/03/2020,10:00,Andorra,188,187,1,
26/03/2020,10:00,Angola,2,2,0,
26/03/2020,10:00,Antigua and Barbuda,3,3,0,
26/03/2020,10:00,Argentina,301,284,4,
26/03/2020,10:00,Armenia,265,264,0,
26/03/2020,10:00,Aruba,12,12,0,
26/03/2020,10:00,Australia,2252,2140,8,
26/03/2020,10:00,Austria,5282,5036,30,
26/03/2020,10:00,Azerbaijan,87,78,1,
```

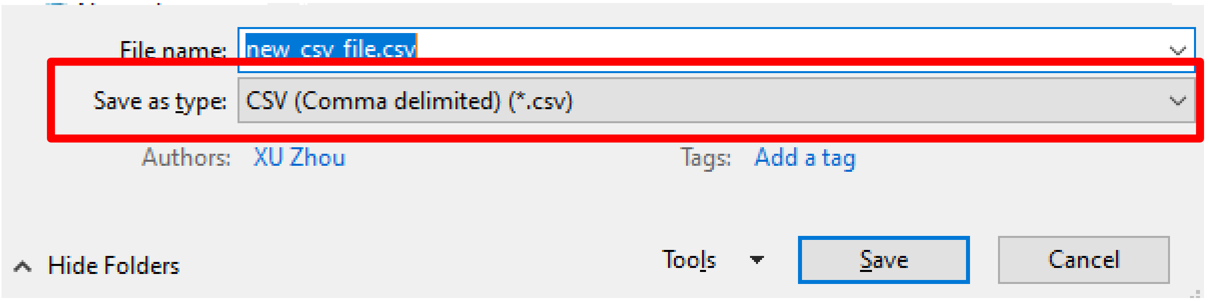
https://data.gov.hk/en-data/dataset/hk-dh-chpsebcddr-novel-infectious-agent

Create a CSV file from Excel

- In Excel, create a new workbook
- Edit the workbook
- When saving the workbook, choose “CVS (Comma delimited)” as “Save as type”
- Use notepad to open the newly saved CSV file



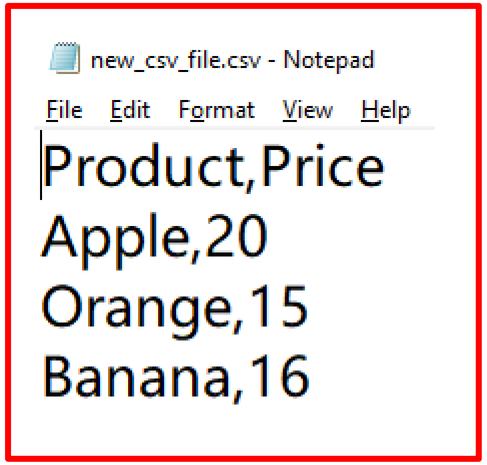
	A	B	C
1	Product	Price	
2	Apple	20	
3	Orange	15	
4	Banana	16	


File name: new_csv_file.csv

Save as type: CSV (Comma delimited) (*.csv)

Authors: XU Zhou

Tags: Add a tag



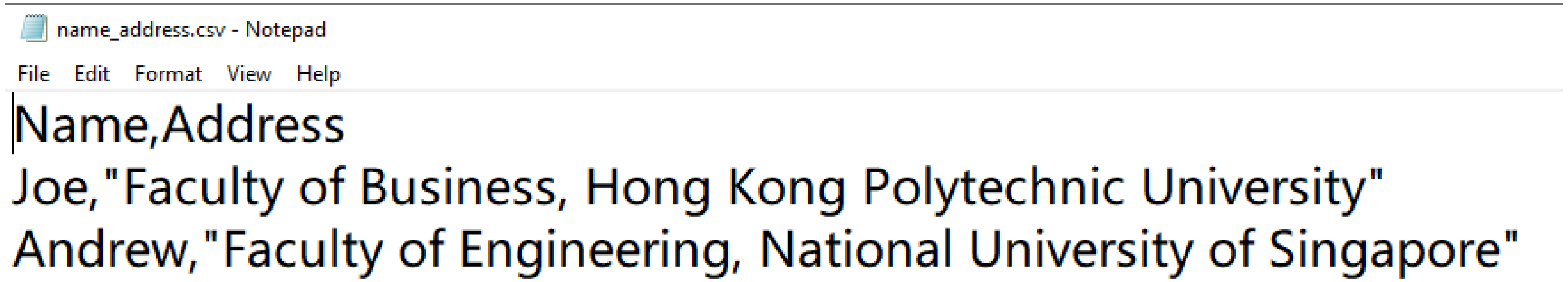
```
Product,Price
Apple,20
Orange,15
Banana,16
```

Parsing CVS files using split() of strings

- Since CSV files are just text files, you might be tempted to read them in as a string and then process that string using the techniques you learned in Lecture 6

```
file = open('input.csv', 'r')
for line in file:
    product, price, quantity = line.split(sep=',')
    print(product, price, quantity, end='')
file.close()
```

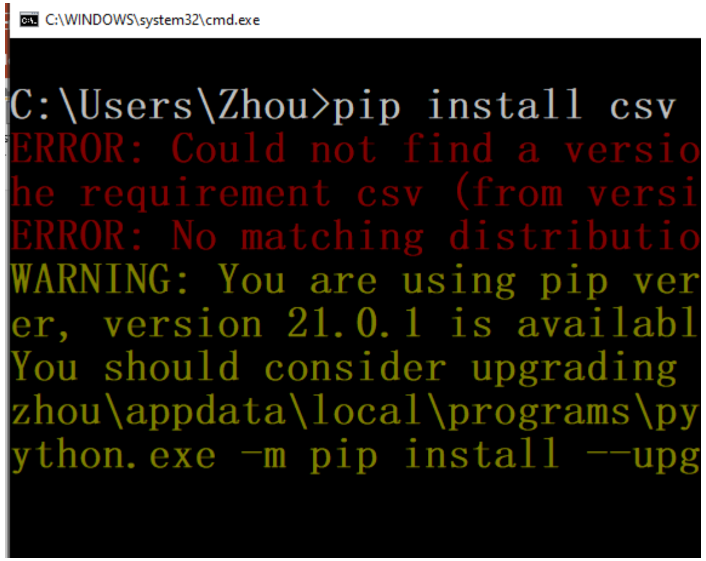
- It can be more complicated: what if the field contains comma?



```
Name,Address
Joe,"Faculty of Business, Hong Kong Polytechnic University"
Andrew,"Faculty of Engineering, National University of Singapore"
```

Using csv Module: Reader Objects

- Installation: pip install csv
- To read data from a CSV file with the csv module, you need to create a reader object.
- A reader object lets you iterate over lines (rows) of data in the CSV file.
  - import csv module; open csv file as a text file; create reader object



```
C:\Users\Zhou>pip install csv
ERROR: Could not find a versio
he requirement csv (from versi
ERROR: No matching distributio
WARNING: You are using pip ver
er, version 21.0.1 is availabl
You should consider upgrading
zhou\appdata\local\programs\py
ython.exe -m pip install --upg
```

```
>>> import csv
>>> exampleFile = open('example.csv', 'r')
>>> exampleReader = csv.reader(exampleFile)
```

Reading data from reader object as a list

- The most direct way to access the values in the reader object is to convert it to a plain Python list by passing it to list()
  - Each element in the list is a list representing a row of data
  - you can then access the value at a particular row and column from the list

```
>>> exampleData = list(exampleReader)
>>> exampleData
[['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries', '85'],
   ['4/6/2015 12:46', 'Pears', '14'], ['4/8/2015 8:59', 'Oranges', '52'],
   ['4/10/2015 2:07', 'Apples', '152'], ['4/10/2015 18:10', 'Bananas', '23'],
   ['4/10/2015 2:40', 'Strawberries', '98’]]
>>> exampleData[0][0]
'4/5/2015 13:34’
>>> exampleData[6][1]
'Strawberries'
```

Reading data from reader object in a for loop

- For large CSV files, you'll want to use the reader object in a for loop to iterate over lines (rows) of data
  - This avoids loading the entire file into memory at once, but row by row

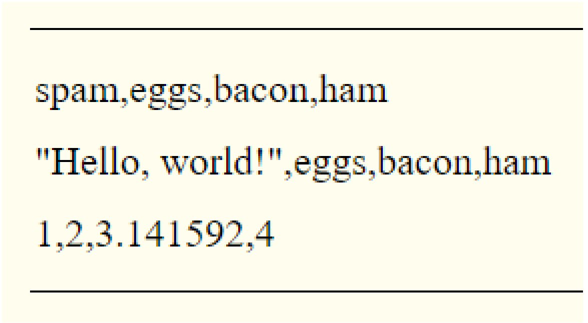
```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
        print('Row #' + str(exampleReader.line_num) + ' ' + str(row))

Row #1 ['4/5/2015 13:34', 'Apples', '73']
Row #2 ['4/5/2015 3:41', 'Cherries', '85']
Row #3 ['4/6/2015 12:46', 'Pears', '14']
Row #4 ['4/8/2015 8:59', 'Oranges', '52']
Row #5 ['4/10/2015 2:07', 'Apples', '152']
Row #6 ['4/10/2015 18:10', 'Bananas', '23']
Row #7 ['4/10/2015 2:40', 'Strawberries', '98']
```

An attribute of the reader indicating the last line that has been read

Using csv Module: Write Objects

- A writer object lets you write data to a CSV file.
- To create a writer object, you use the csv.writer() function.
  - On Windows, you’ll also need to pass a blank string for the open() function’s newline keyword argument.
- The writerow() method takes a list argument. Each value in the list is placed in its own cell in the output CSV file.
  - The return value is the number of characters written to the file for that row

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')
>>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])
21
>>> outputWriter.writerow(['Hello, world!', 'eggs', 'bacon', 'ham'])
32
>>> outputFile.close()
```

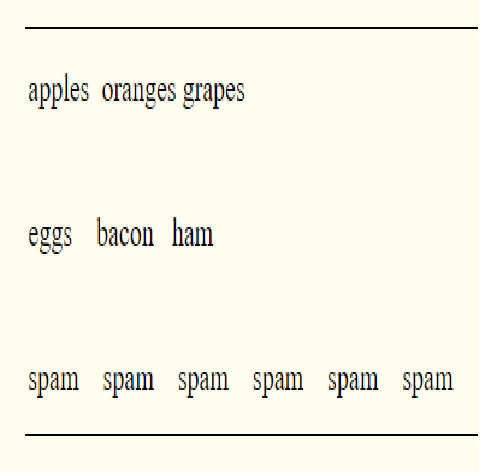


```
spam,eggs,bacon,ham
"Hello, world!",eggs,bacon,ham
1,2,3.141592,4
```

The delimiter and lineterminator Keyword Arguments

- The *delimiter* is the character that appears between cells on a row.
  - By default, the delimiter for a CSV file is a comma.
- The *lineterminator* is the character that comes at the end of a row.
  - By default, the line terminator is a newline.
- You can change characters to different values by using the *delimiter* and *lineterminator* keyword arguments with csv.writer().

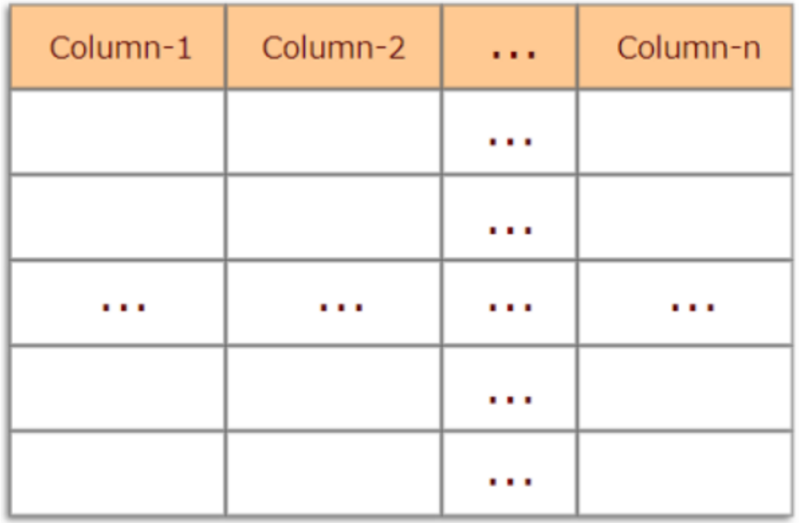
```
import csv
csvFile = open('output2.csv', 'w', newline='')
csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
csvWriter.writerow(['apples', 'oranges', 'grapes'])
csvWriter.writerow(['eggs', 'bacon', 'ham'])
csvWriter.writerow(['spam'] * 6)
csvFile.close()
```



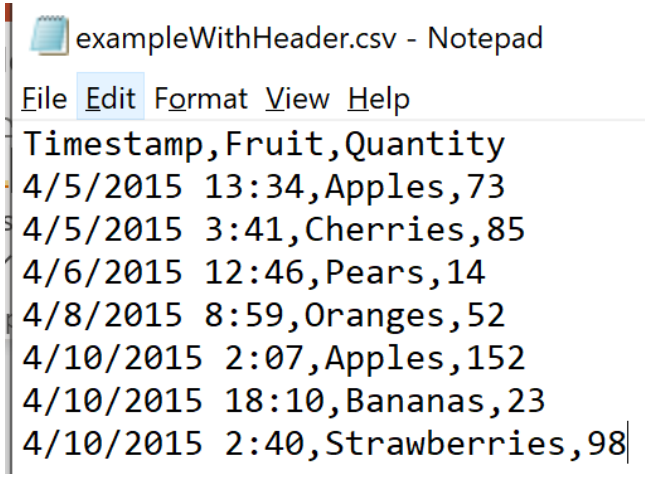
```
apples  oranges grapes


eggs    bacon   ham


spam    spam    spam    spam    spam    spam
```

Tabular Data

- Rows
- Columns
  - Column Names
- Cells
- How to represent a row
  - List : [‘1A’,’1B’]
  - Dictionary: {‘Column Name A’: ‘1A’, ‘Column Name B’: ‘1B’, … }



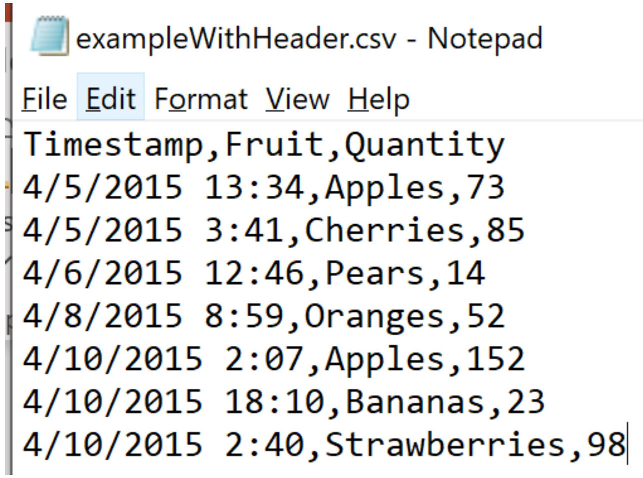
Column-1	Column-2	…	Column-n
		…	
		…	
…	…	…	…
		…	
		…	



```
Timestamp,Fruit,Quantity
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

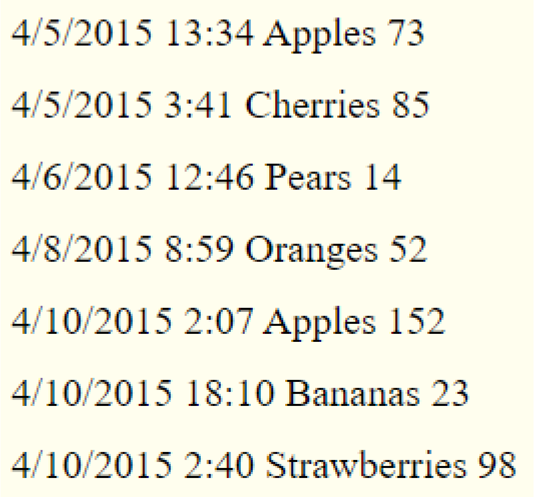
DictReader and DictWriter CSV Objects (1)

- For CSV files that contain header rows, it’s often more convenient to work with DictReader and DictWriter objects
- DictReader and DictWriter CSV objects perform the same functions but use dictionaries instead of lists
  - Using the first row of the CSV file as the keys of these dictionaries.



```
Timestamp,Fruit,Quantity
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

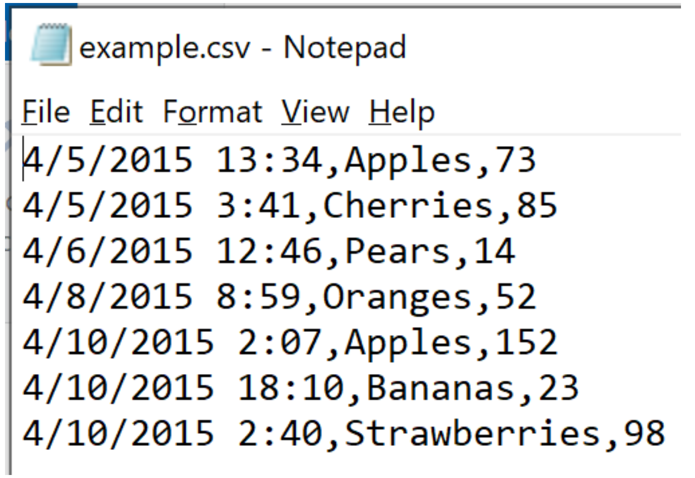
```
>>> import csv
>>> exampleFile = open('exampleWithHeader.csv')
>>> exampleDictReader = csv.DictReader(exampleFile)
>>> for row in exampleDictReader:
...     print(row['Timestamp'], row['Fruit'], row['Quantity'])
```



```
4/5/2015 13:34 Apples 73

4/5/2015 3:41 Cherries 85

4/6/2015 12:46 Pears 14

4/8/2015 8:59 Oranges 52

4/10/2015 2:07 Apples 152

4/10/2015 18:10 Bananas 23

4/10/2015 2:40 Strawberries 98
```

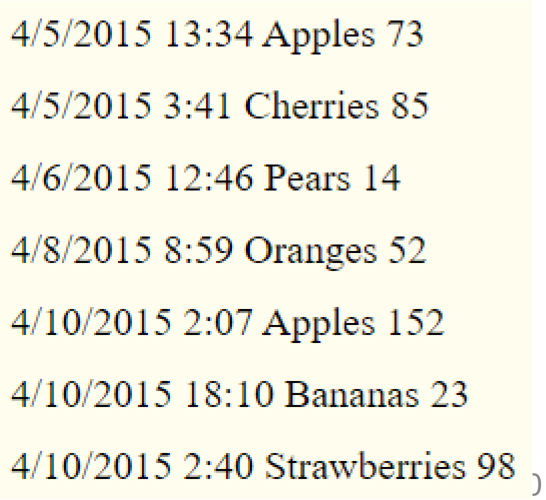
DictReader and DictWriter CSV Objects (2)

- For CSV files that do not contain header rows, you can supply the DictReader() function with a second argument containing made-up header names, which can be used as keys



```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

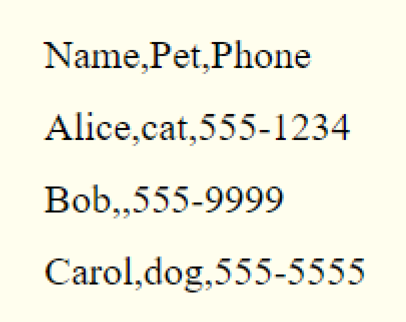
```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleDictReader = csv.DictReader(exampleFile,
['time', 'name','amount'])
>>> for row in exampleDictReader:
...     print(row['time'], row['name'], row['amount'])
```



```
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
4/6/2015 12:46 Pears 14
4/8/2015 8:59 Oranges 52
4/10/2015 2:07 Apples 152
4/10/2015 18:10 Bananas 23
4/10/2015 2:40 Strawberries 98
```

DictReader and DictWriter CSV Objects (3)

- DictWriter objects use dictionaries to create CSV files.
  - Using writerow() method
- If you want your file to contain a header row, write that row by calling writeheader() method
  - missing keys will be empty in the csv file.

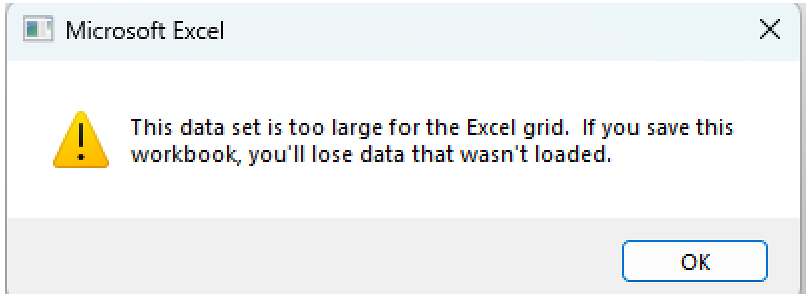


```
Name,Pet,Phone
Alice,cat,555-1234
Bob,,555-9999
Carol,dog,555-5555
```

```
>>> import csv
>>> outputFile = open('output3.csv', 'w', newline='')
>>> outputDictWriter = csv.DictWriter(outputFile, ['Name', 'Pet', 'Phone'])
>>> outputDictWriter.writeheader()
>>> outputDictWriter.writerow({'Name': 'Alice', 'Pet': 'cat', 'Phone': '555-1234'})
20
>>> outputDictWriter.writerow({'Name': 'Bob', 'Phone': '555-9999'})
15
>>> outputDictWriter.writerow({'Phone': '555-5555', 'Name': 'Carol', 'Pet': 'dog'})
20
>>> outputFile.close()
```

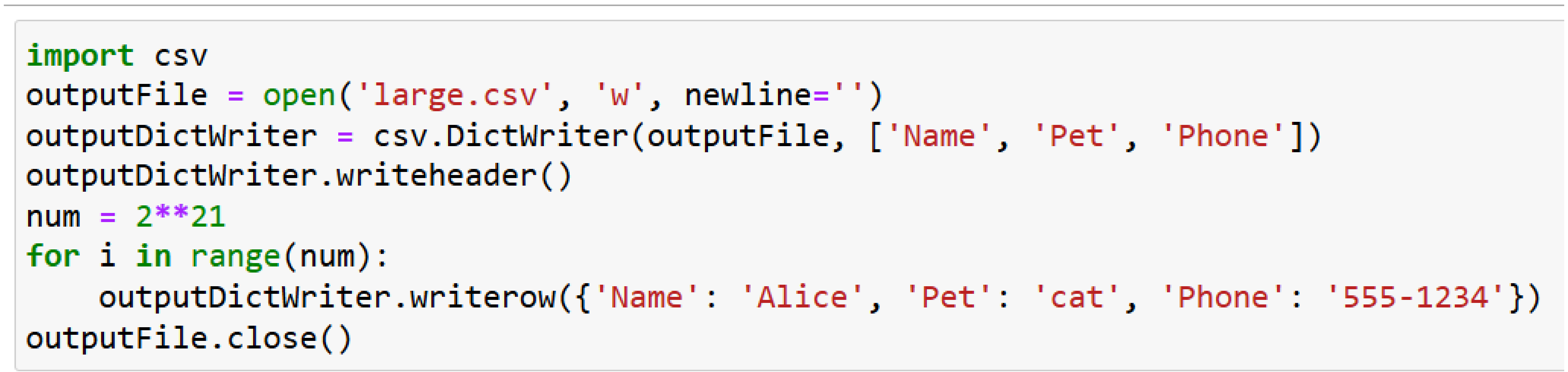
What’s the maximum number of rows in an Excel Worksheet?

- 2^{20}: about 1 million
- Create a large .csv file of about 2 million rows
- Use Excel to open this large .csv file



This data set is too large for the Excel grid. If you save this workbook, you'll lose data that wasn't loaded.

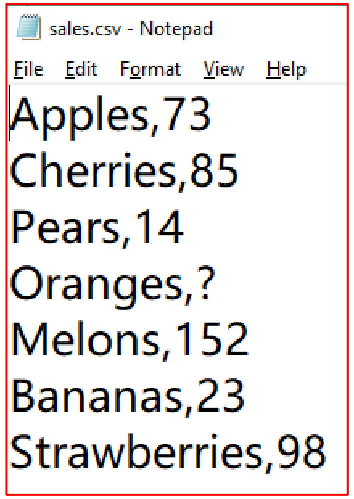
Write a large csv file



```
import csv
outputFile = open('large.csv', 'w', newline='')
outputDictWriter = csv.DictWriter(outputFile, ['Name', 'Pet', 'Phone'])
outputDictWriter.writeheader()
num = 2**21
for i in range(num):
    outputDictWriter.writerow({'Name': 'Alice', 'Pet': 'cat', 'Phone': '555-1234'})
outputFile.close()
```

Data Cleaning and Munging

- Real-world data is dirty.
- You’ll have to do some work on the data before using it:
  - Data Cleaning for missing values and bad data
    - E.g. missing sales (‘?’)→ default sales (20)
  - Data Munging for wrongly formatted data
    - E.g. for sales: string value (‘152’)→ float values (152.0)



```
Apples,73
Cherries,85
Pears,14
Oranges,?
Melons,152
Bananas,23
Strawberries,98
```

```
import csv
exampleFile = open('sales.csv')
exampleReader = csv.reader(exampleFile)

default_sales = 20
sales={}
for line in exampleReader:
    col2 = line[1]
    if col2 == '?': #Fixing missing data
        sales[line[0]] = default_sales
    else: #Converting string to float
        sales[line[0]] = float(col2)
print(sales)
```

Acquiring Data

- Acquiring data from data sources is necessary before data analysis
- There are many sources to access data
  - Input
    - module: sys
  - Text files
    - module: csv, pandas
  - Excel files
    - module: openpyxl, pandas
  - HTML files
    - modules: requests, bs4, pandas
  - Database
    - modules: sqlite3, pandas
  - JSON (JavaScript Object Notation)
    - module: json, pandas
  - API (application programming interface) – called to obtain data from internet (e.g., twitter …)
    - modules: requests, json

Summary

- Various sources / formats for acquiring data
- Access data from csv files
  - csv files
  - csv module for reading and writing data
- Data cleaning and munging
  - Parse → extract → edit

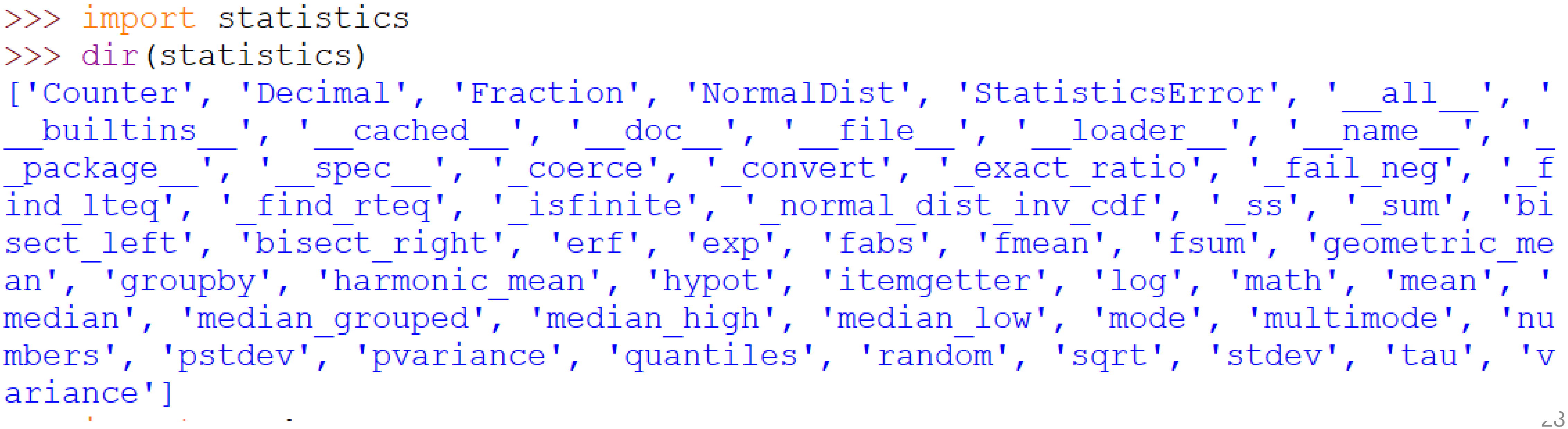
EXPLORING DATA

Exploring Data

- Data exploration is the very first step in data analysis
  - It all begins with exploring a large set of unstructured data while looking for patterns, characteristics, or points of interest.
  - Summarizing the size, accuracy and initial patterns in the data is key to enabling deeper analysis.
    - Summary statistics: statistics, numpy, scipy, pandas modules
    - Summary charts: matplotlib module
  - The purpose is meant to help create a broader picture of potential trends or points to look for upon further analysis to refine the data.

Summary Statistics by statistics module (1)

- Python's **statistics** module is a built-in Python library for descriptive statistics.
  - You can use it if your datasets are not too large or if you can't rely on importing other libraries.



```
>>> import statistics
>>> dir(statistics)
['Counter', 'Decimal', 'Fraction', 'NormalDist', 'StatisticsError', '__all__', '
__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '_
_package__', '__spec__', '_coerce', '_convert', '_exact_ratio', '_fail_neg', '_f
ind_lteq', '_find_rteq', '_isfinite', '_normal_dist_inv_cdf', '_ss', '_sum', 'bi
sect_left', 'bisect_right', 'erf', 'exp', 'fabs', 'fmean', 'fsum', 'geometric_me
an', 'groupby', 'harmonic_mean', 'hypot', 'itemgetter', 'log', 'math', 'mean', '
median', 'median_grouped', 'median_high', 'median_low', 'mode', 'multimode', 'nu
mbers', 'pstdev', 'pvariance', 'quantiles', 'random', 'sqrt', 'stdev', 'tau', 'v
ariance']
```

Summary Statistics by statistics module (2)

- Python’s statistics module is a built-in Python library for *descriptive statistics*
  - You can use it if your datasets are not too large or if you can’t rely on importing other libraries.
  - mean() function: returns average value
  - stdev() function: returns standard deviation

```
>>> x = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
>>> statistics.mean(x)
71.0
>>> statistics.stdev(x)
47.26520919238589б
>>> max(x)
152.0
>>> min(x)
14.0
```

Summary Charts by matplotlib module

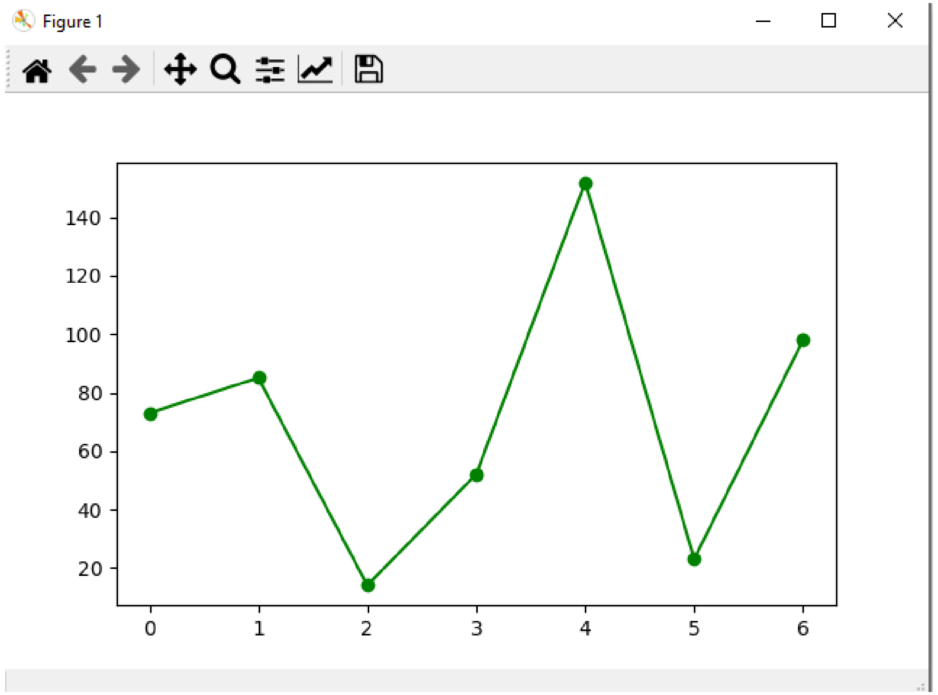
- A fundamental part of the data analytics is data visualization
- **matplotlib** module contains a wide variety of tools for visualizing data, and is widely used
  - Installation: pip install matplotlib
- **pyplot** is a useful submodule contained in matplotlib and is frequently used.
  - We often import matplotlib.pyplot using the alias plt:

```
import matplotlib.pyplot as plt
```

Making a chart using pyplot

- pyplot maintains an internal state in which we can build up a visualization step by step
- Once you are done, you can save it with savefig() method or display with show() method

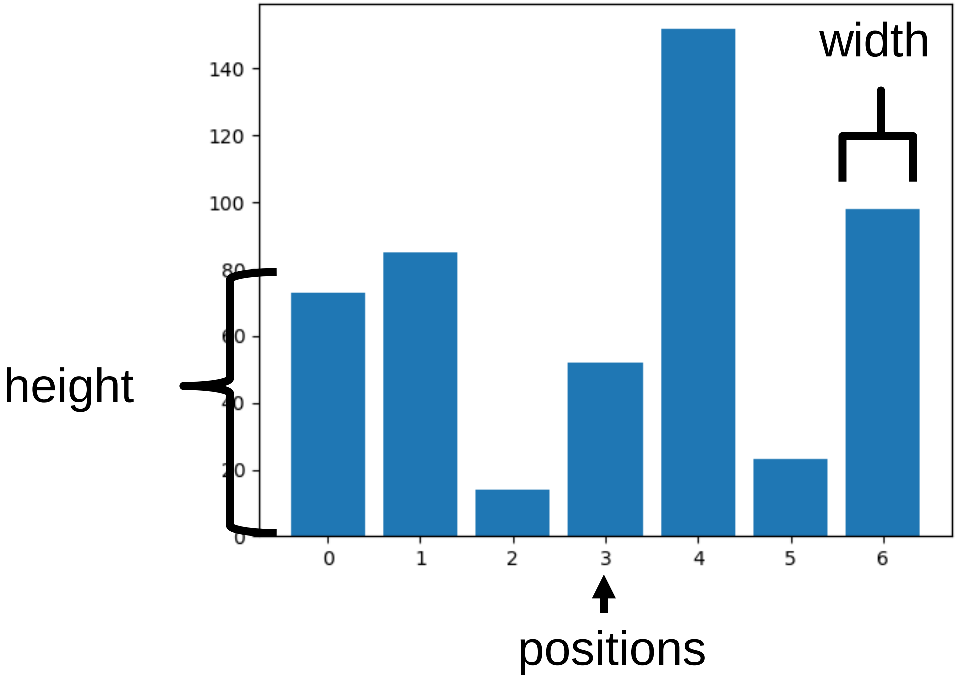
```
>>>import matplotlib.pyplot as plt
>>>y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
>>>plt.plot(y, color='green', marker='o',
linestyle='solid’)
>>>plt.savefig('plot.png')
>>>plt.show()
```



Bar Charts

- A bar chart is a good choice when you want to show how some quantity varies among some discrete set of items
  - For example, sales quantities of different products
- matplotlib.pyplot.bar(positions, height, width=0.8) method
  - The bars are positioned at points in *positions* with sizes given by *height* and *width*

```
>>>import matplotlib.pyplot as plt
>>>y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
>>>x = list(range(len(y)))
>>>plt.bar(x,y)
>>>plt.show()
```

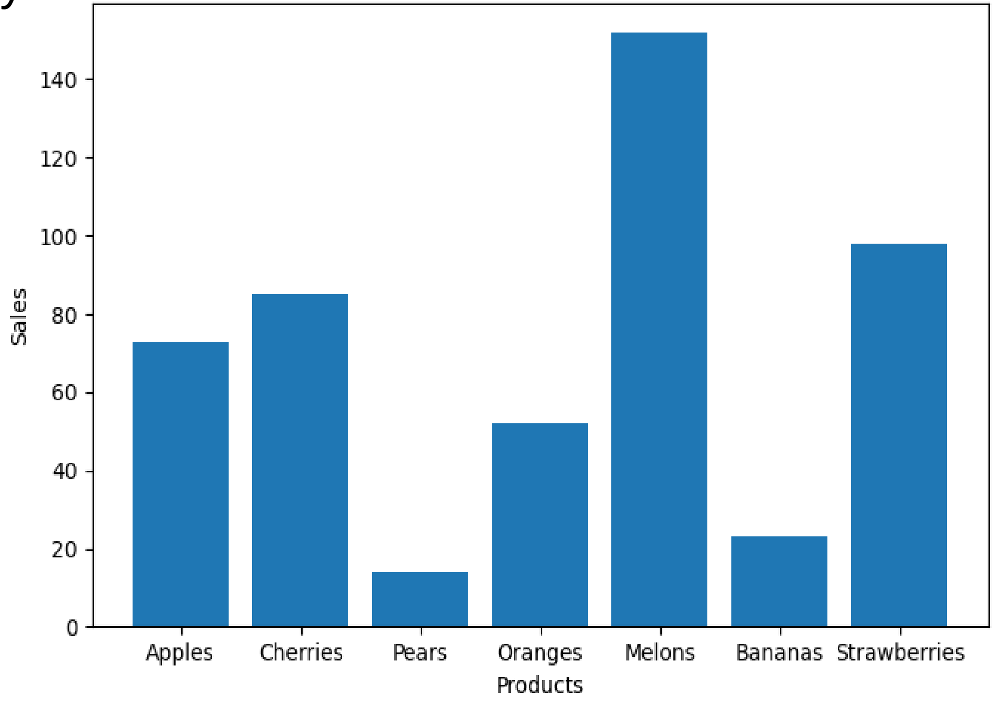


Add Labels and Ticks

- Labels: Get or set the current labels of the x-axis and y-axix
  - – xlabel(text), ylabel(text)
- Ticks: Get or set the current ticks of the x-axis and y-axix
  - – xticks(ticks, labels), yticks(ticks, labels)

```
import matplotlib.pyplot as plt

y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
products=['Apples', 'Cherries', 'Pears',
'Oranges', 'Melons', 'Bananas', 'Strawberries']

plt.bar(range(len(y)),y)
plt.ylabel('Sales')
plt.xlabel('Products')

plt.xticks(range(len(products)),products)
plt.show()
```

yticks



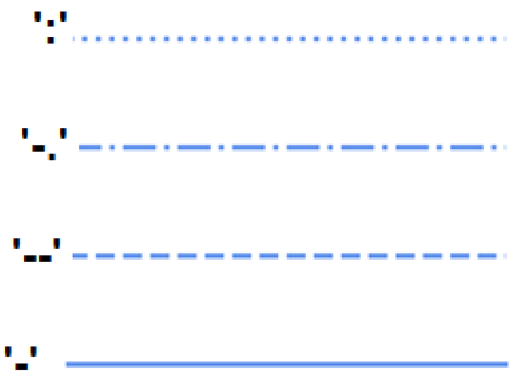
ylabels

xticks

xlabels

Line Charts

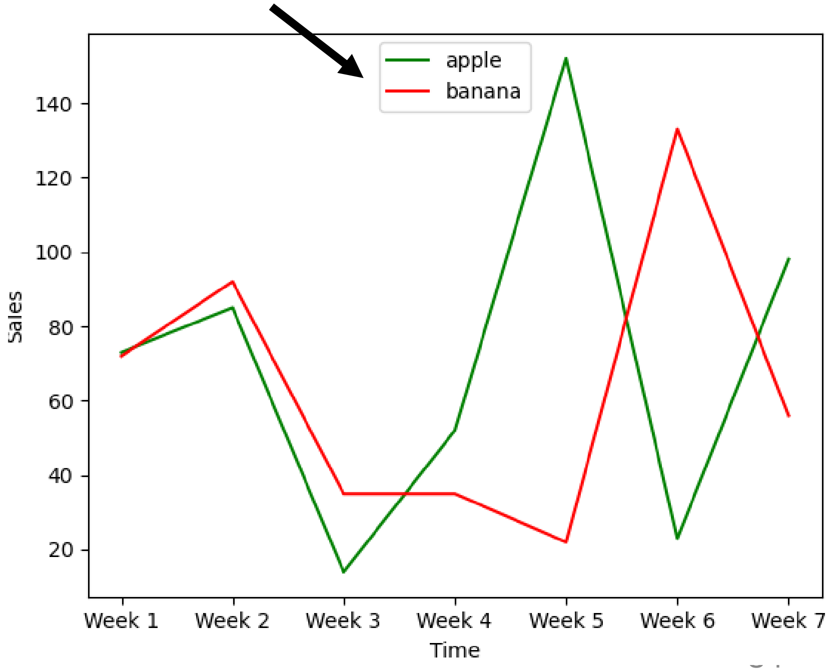
- A line chart is for showing trends
  - E.g., sales of the same product for different weeks
- matplotlib.pyplot.plot(x,y,fmt,label):
  - Plot y versus x as lines and/or markers.
  - fmt is a parameter specifying the format of the line
  - label is a parameter specifying the label of the line



'b', 'g', 'r', 'c', 'm', 'y', 'k' , ’w’ for blue, green, red, cyan, magenta, yellow, black, white.

```
import matplotlib.pyplot as plt
y1 = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
y2 = [72.0, 92.0, 35.0, 35.0, 22.0, 133.0, 56.0]
weeks=['Week 1', 'Week 2', 'Week 3', 'Week 4', 'Week 5',
'Week 6', 'Week 7']
plt.plot(weeks,y1,'g-', label='apple') #green solid line
plt.plot(weeks,y2,'r-', label='banana') #red dot-dahsed
line
plt.legend(loc=9) #loc=9 means upper center
plt.ylabel('Sales')
plt.xlabel('Time')
plt.show()
```

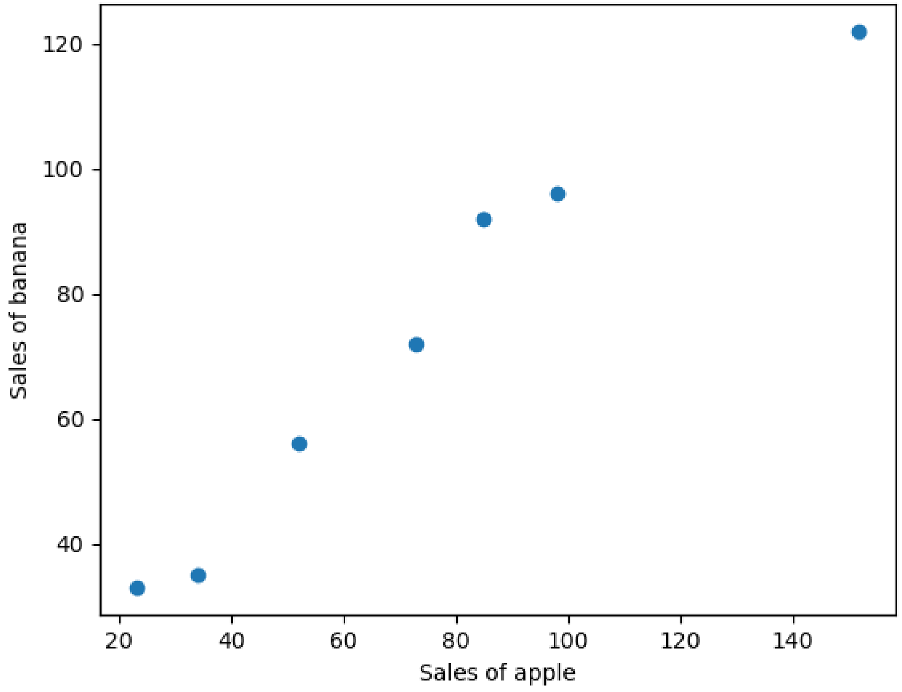
legend



Scatterplots

- A scatterplot is for visualizing the relationship between two paired sets of data
  - For example, relationship between sales of two products
- matplotlib.pyplot.scatter(data1, data2): A scatter plot of data1 vs. data2, where data1 and data2 are of the same size

```
import matplotlib.pyplot as plt

y1 = [73.0, 85.0, 34.0, 52.0, 152.0, 23.0, 98.0]
y2 = [72.0, 92.0, 35.0, 56.0, 122.0, 33.0, 96.0]
plt.scatter(y1,y2)
plt.xlabel('Sales of apple')
plt.ylabel('Sales of banana')
plt.show()
```

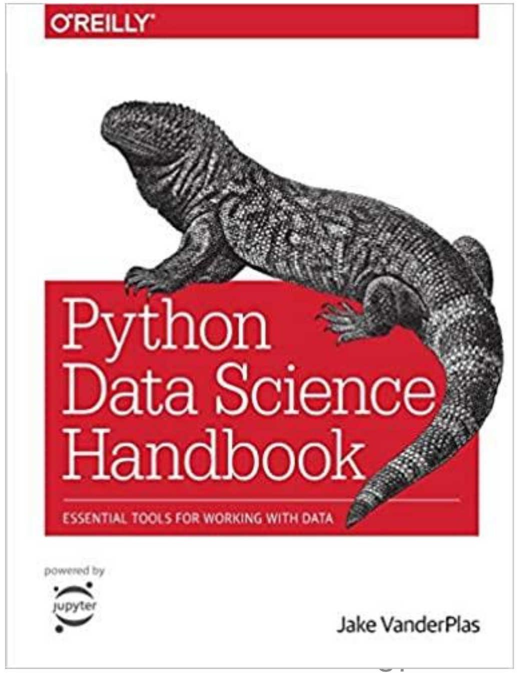


Future Study: Data Visualization by Python

- The matplotlib Gallery (https://matplotlib.org/stable/gallery/index.html) give examples about what can be done with matplotlib (and how to do them)
  - A more extensive introductory-level tutorial can be found at https://matplotlib.org/stable/tutorials/index.html
- The seaborn package (https://seaborn.pydata.org/) is a built on top of matplotlib and allows you to easily produce prettier (and more complex) visualization

Future Study: Data Analytics by Python

- Data Acquiring
  - Pandas
- Summary Statistics
  - Scipy
- Data Manipulation
  - Pandas
- Statistical Hypothesis Testing
  - Scipy
- Time Series Analysis
  - Pandas
- Machine Learning
  - Scikit-Learn
- Data Visualization
  - Matplotlib


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA
powered by
jupyter
Jake VanderPlas

Summary

- Various sources / formats for acquiring data
- Access data from csv files
  - csv files
  - csv module for reading and writing data
- Data cleaning and munging
  - Parse → extract → edit

- Summary statistics for data
  - Use of statistics module
- Charts for data
  - Use of matplotlib.pyplot module
  - Line, bar, scatterplot

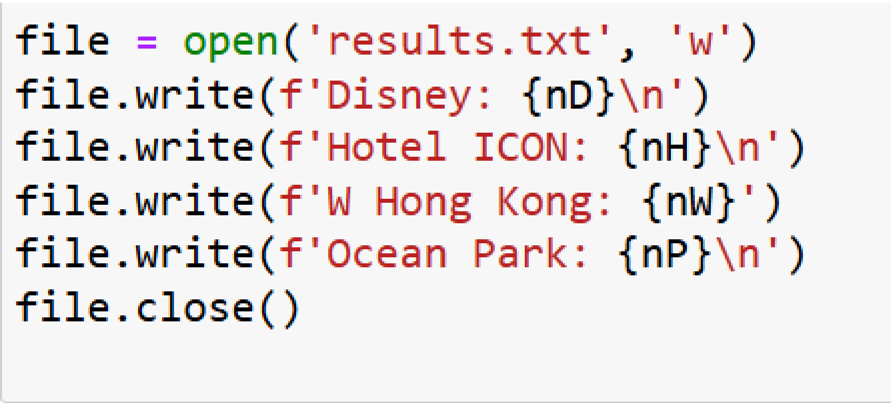
Preparation for Tutorials

If you are not able to make these preparation for our tutorials, please contact me

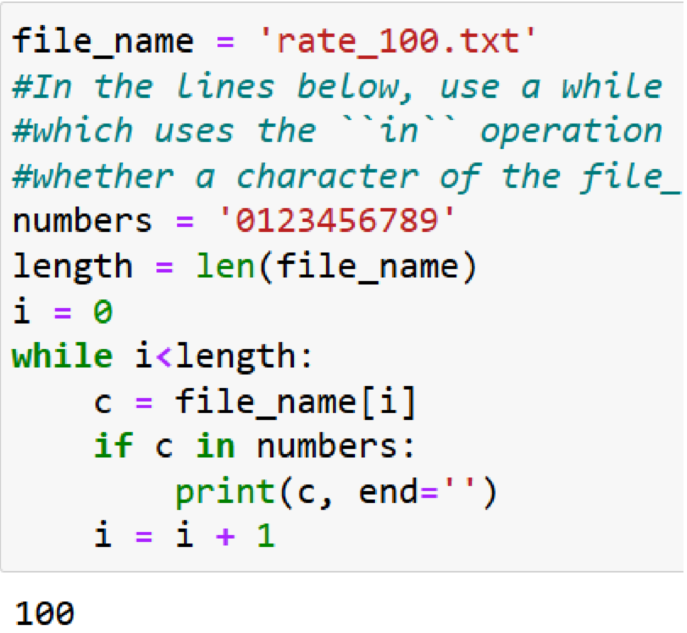
- Start Anaconda Navigator (Remotely or Locally)
  - Follow Part 1.1 of instructions for Tutorial 1
- Download files of the tutorials from Blackboard
  - Follow Part 1.2 of instructions for Tutorial 1
  - For today's tutorial, use T10 as the working folder's name, download the zip file from Tutorial 10 in the Blackboard, and extract the zip file in working directory E:\LGT3109\T10
- Launch Python IDLE from Anaconda Navigator
  - Follow Part 2.1 of instructions for Tutorial 1
- Launch Jupyter Notebook from Anaconda Navigator
  - Follow Part 3.1 of instructions for Tutorial 1

About Tutorial 6

- Feedbacks and Sample Solutions are available at Blackboard
- 1.3(c)Use c in '0123456789' to indicate whether c is a number.
- 2.1: help(''.isnumeric)
- 4.1: The results need to be written to the file



```
file = open('results.txt', 'w')
file.write(f'Disney: {nD}\n')
file.write(f'Hotel ICON: {nH}\n')
file.write(f'W Hong Kong: {nW}')
file.write(f'Ocean Park: {nP}\n')
file.close()
```



```
file_name = 'rate_100.txt'
#In the lines below, use a while
#which uses the ``in`` operation
#whether a character of the file_
numbers = '0123456789'
length = len(file_name)
i = 0
while i<length:
    c = file_name[i]
    if c in numbers:
        print(c, end='')
    i = i + 1
```

```
100
```

Tutorial 10: Question 1.4

REMOVING THE HEADER FROM CSV FILES

- Ideas for Similar Programs
  - Compare data between different rows in a CSV file or between multiple CSV files.
  - Copy specific data from a CSV file to a text file in different format, or vice versa.
  - Check for invalid data or formatting mistakes in CSV files and alert the user to these errors.
  - Read data from a CSV file as input for your Python programs.

Schedule

- TODAY: Lecture & Tutorial 10: Basic Data management
  - Due: Tutorial 8
- 4 April: Ching Ming Festival (No class)
  - Due: Individual Assignment
- 11 April: Lecture: Testing, Debugging, Course Review
  - Due: Tutorial 9
- 18 April: Easter Holidays (No class)
  - Due: Tutorial 10
- 24 April (12:30 – 15:40): Q&A (No attendance)
  - MN102c
  - Make-up class for 18 April
- 13 May (15:15-18:15): Final Exam
  - MN102a and MN102c
  - Computer-based, Blackboard, 2 sided A4 paper, sample solutions and lecture slides shared on Blackboard

Reminders

- Feel free to contact the instructor if you have problems
  - Post/reply to threads in “Discussion Forum” on the Blackboard
  - By email: lgtzx@polyu.edu.hk
  - Call: 3400-3624
  - Office hour: Friday afternoon (15:45-17:15), open door, or by appointment
  - No office hour today due to other appointsments