LGT3109
Introduction to Coding for Business with Python
(week 10)

Xiaoyu Wang

Dept. of LMS, The HK PolyU

Summary of Week 9

- Pathlib , os , shutil modules
- Path and file name
- Get and change working directory: Path.cwd () and os.chdir()
- Absolute path: Path.resolve()
- Relative path: dot and dot-dot folders
- Create new folder: os.makedirs()
- Check path validity: exists(), is_file (), is_dir () of a Path object
- Get folder content: os.listdir (), glob() of a Path object

Summary of Week 9

- Copy files and folders: shutil.copy (), shutil.copytree()
- Move and rename files and folders: shutil.move()
- Delete files and folders: os.unlink (), os.rmdir (), shutil.rmtree()
- Walk a directory: os.walk()
- Read ZIP files: zipfile.ZipFile (), namelist (), getinfo()
- Extract ZIP files: extractall (),
- Write ZIP files: ‘w’ mode, write()
- Close ZIP files: close()

Basics of Data Analytics

- Acquiring data

➢Access data from csv files

➢Data cleaning and munging

- Exploring data

➢Summary statistics for data

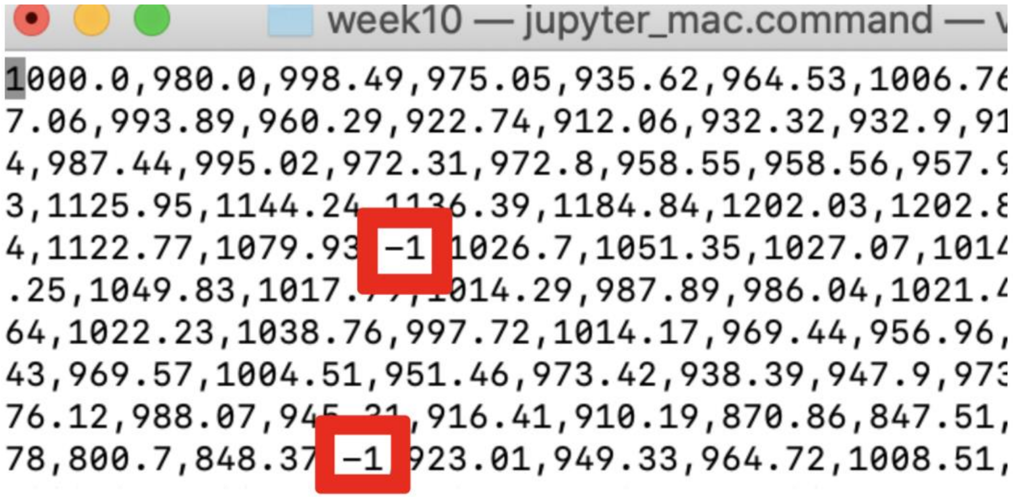
➢Charts for data

Motivation Case-Tasks

- Foxconn explores shipping rates stored in a text file (rates.csv).

➢One line of 300 float numbers.

➢Rates may be missing for some days.



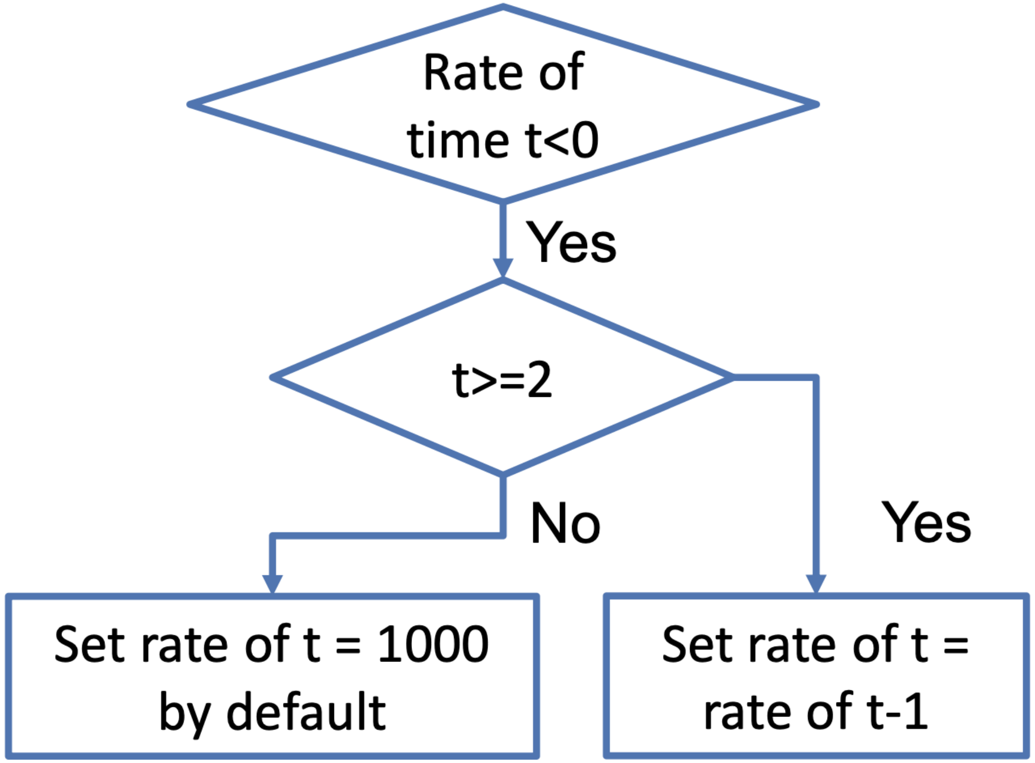
```
1000.0,980.0,998.49,975.05,935.62,964.53,1006.76
7.06,993.89,960.29,922.74,912.06,932.32,932.9,91
4,987.44,995.02,972.31,972.8,958.55,958.56,957.9
3,1125.95,1144.24,1136.39,1184.84,1202.03,1202.8
4,1122.77,1079.93 -1 1026.7,1051.35,1027.07,1014
.25,1049.83,1017.77,1014.29,987.89,986.04,1021.4
64,1022.23,1038.76,997.72,1014.17,969.44,956.96,
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,
```

- Foxconn needs to:

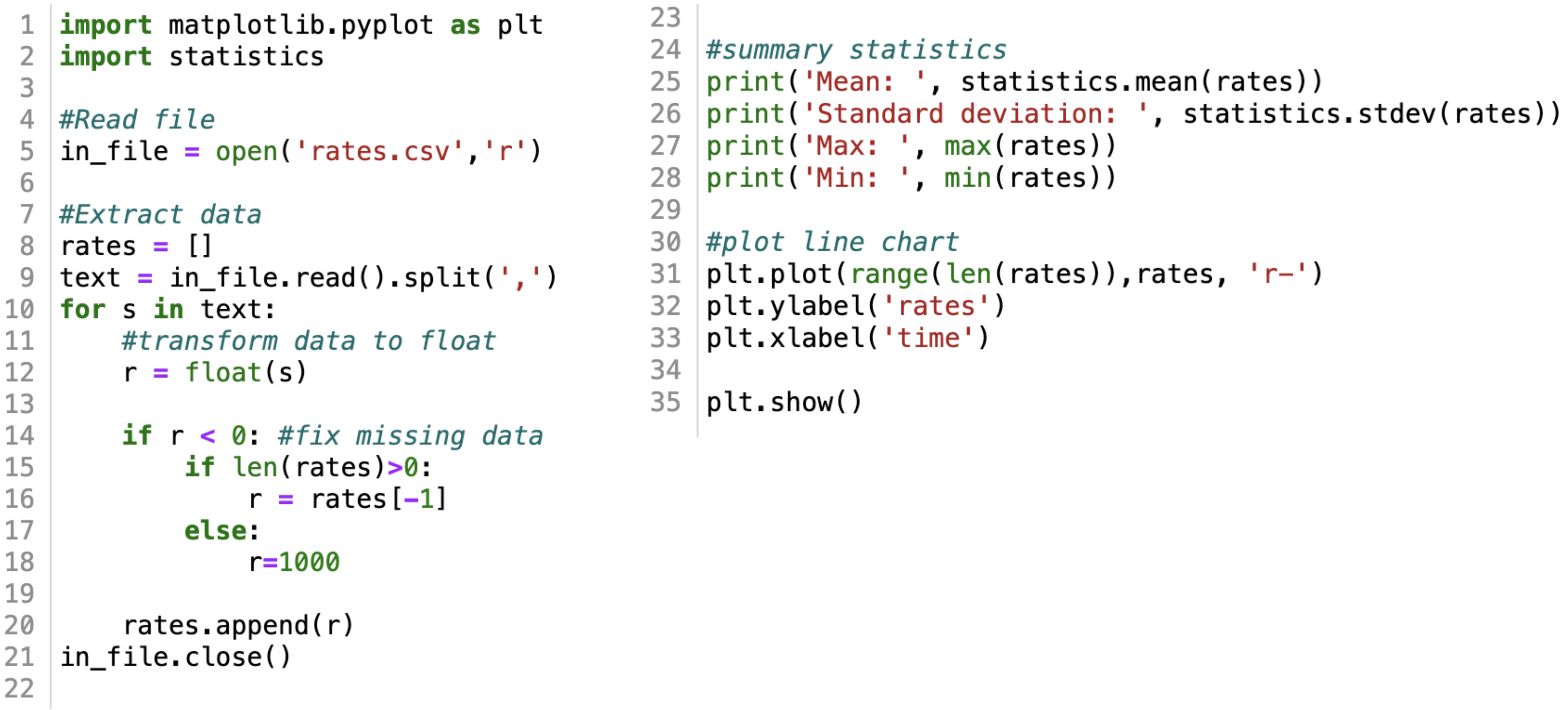
➢Access the data; fix missing data; Compute summary statistics; Plot a line chart to observe trends.

Motivation Case-Fixing Data

How to fix missing data?



Motivation Case-Code



```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:
    #transform data to float
    r = float(s)

    if r < 0: #fix missing data
        if len(rates)>0:
            r = rates[-1]
        else:
            r=1000

    rates.append(r)
in_file.close()


#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()
```

Basics of Data Analytics

- Acquiring data
    - Access data from csv files
    - Data cleaning and munging

- Exploring data
    - Summary statistics for data
    - Charts for data

Acquiring Data from CSV Files

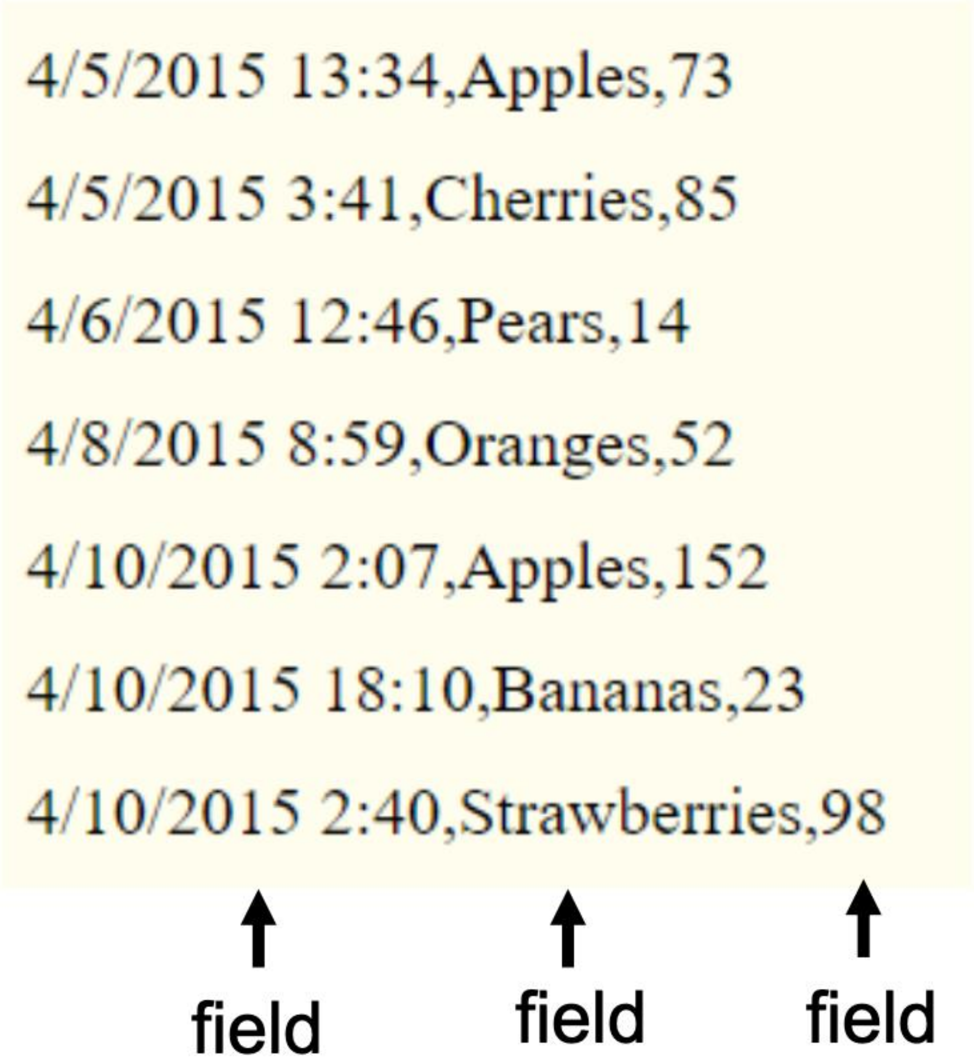
- Text files for data analysis are often comma or tab separated.

➢Commas or tabs separate different field in a line.

- Data often in CSV (comma-separated values) format.

➢Each line in a CSV file represents a row in the spreadsheet.

➢Commas separate the cells in the row.

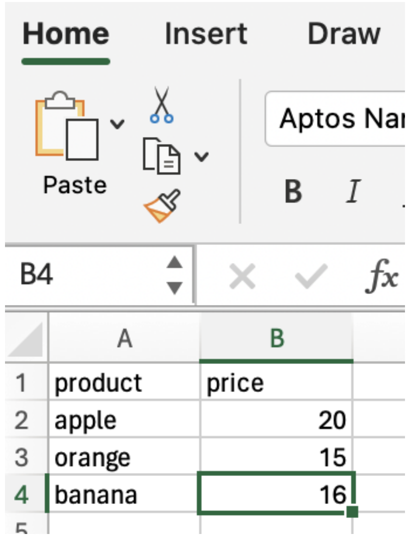


```
4/5/2015 13:34,Apples,73
4/5/2015 3:41,Cherries,85
4/6/2015 12:46,Pears,14
4/8/2015 8:59,Oranges,52
4/10/2015 2:07,Apples,152
4/10/2015 18:10,Bananas,23
4/10/2015 2:40,Strawberries,98
```

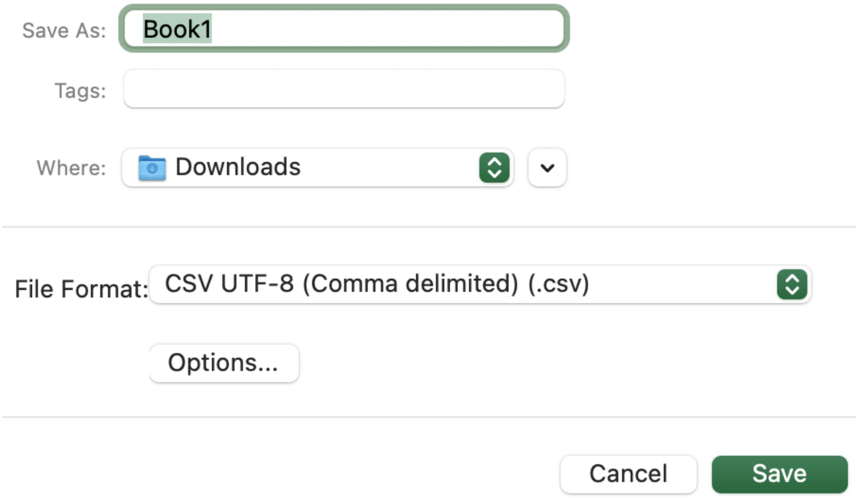
field
field
field

Create a CSV File from Excel

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



	A	B
1	product	price
2	apple	20
3	orange	15
4	banana	16
5		

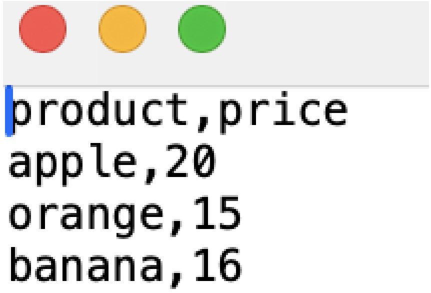


Save As: Book1

Tags:

Where: Downloads

File Format: CSV UTF-8 (Comma delimited) (.csv)



```
product,price
apple,20
orange,15
banana,16
```

Features of CSV Flies

- CSV files are simple, but compared with Excel spreadsheet:

➢Don’t have types for values: everything is a string.

➢Don’t have multiple worksheets

➢Can’t specify cell widths, heights, font size, and color.

➢Can’t merge cells.

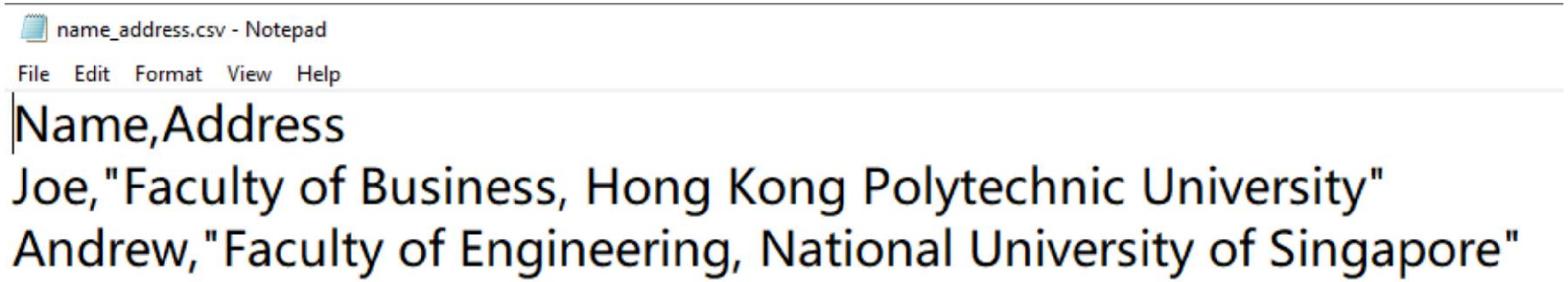
➢No images or charts in CSV.

Parsing CVS Files Using split() of Strings

- CSV files are just text files. Why not read them as a string and process with what 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
- Create reader object to read CSV file with the csv module.
- Reader object enables to iterate over lines (rows) of data in the CSV file.
- Import csv module; open csv file as a text file; create reader object

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

Reading Data from Reader Object as a List

- Convert data to a plain Python list using list().

➢Each element in the list is a list representing a row of data.

➢Can 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 Loop

- For large CSV files, use reader object to loop over lines of data.

➤This avoids loading the entire 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']
```

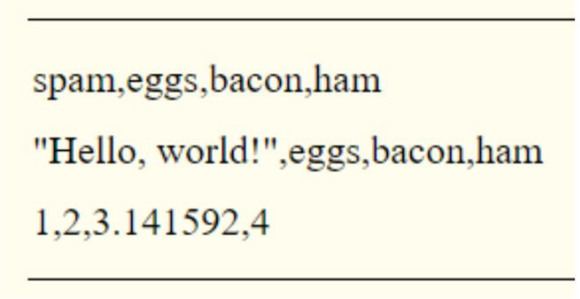
An attribute of the reader indicating the last line that has been read

Using csv Module: Write Objects

- A writer object lets you write data to a CSV file.

➢Use csv.writer() to create a writer object.

➢Use writerow() and each value in the list is placed in its own cell in the output CSV file.



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

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

Delimiter and Line Terminator

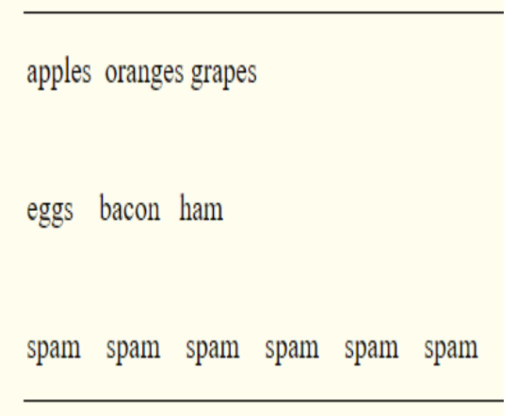
- The delimiter is the character between cells on a row.

➢By default, the delimiter for a CSV file is a comma.

- The line terminator is the character at the end of a row.

➢By default, the line terminator is a newline.

```
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()
```

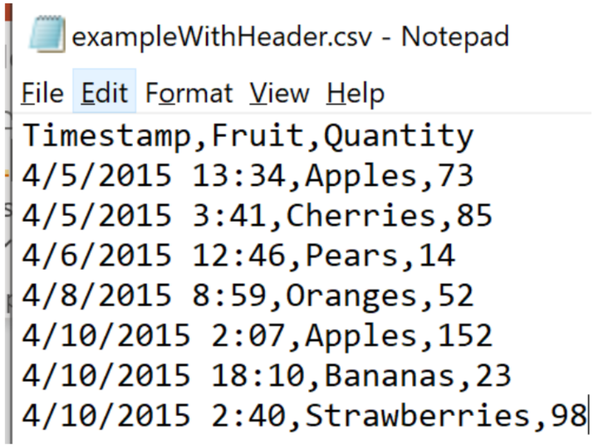


```
apples  oranges grapes

eggs    bacon   ham

spam    spam    spam    spam    spam    spam
```

DictReader and DictWriter CSV Objects (1)

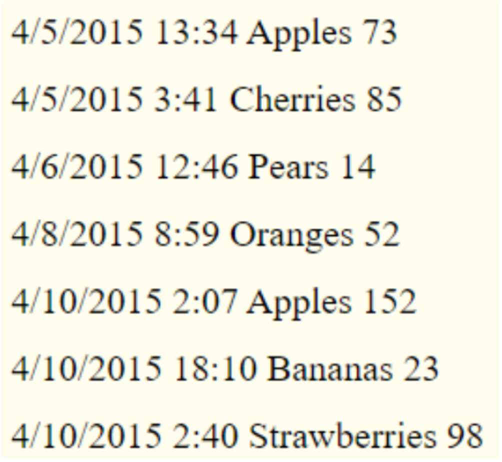
- DictReader and DictWriter CSV objects perform the same functions but use dictionaries instead of lists.

➢ 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
```

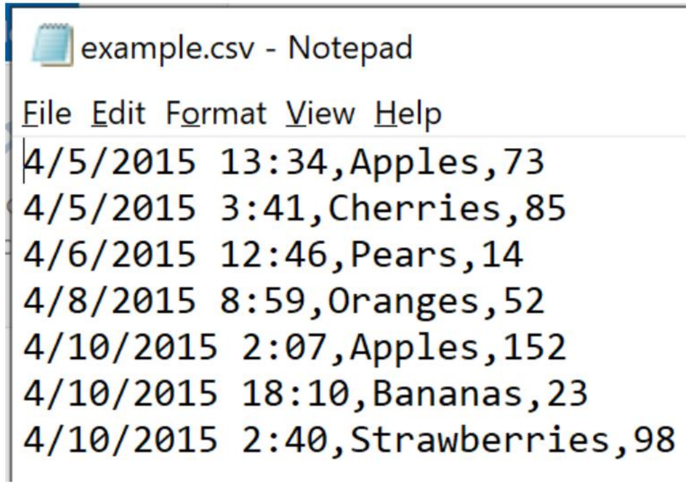
```
>>> import csv
>>> exampleFile = open('exampleWithHeader.csv')
>>> exampleDictReader = csv.DictReader(exampleFile)
>>> for row in exampleDictReader:
... print(row['Timestamp'], row['Fruit'], row['Quantity'])
```



```
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
```

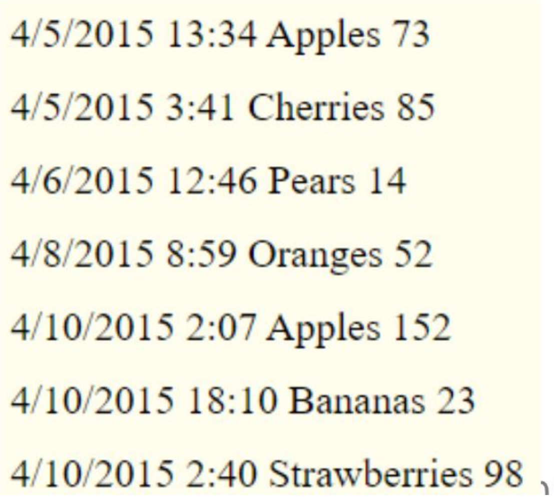
DictReader and DictWriter CSV Objects (2)

- If CSV files do not contain header rows, use DictReader() with a second argument containing made-up header names.



```
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'])
```



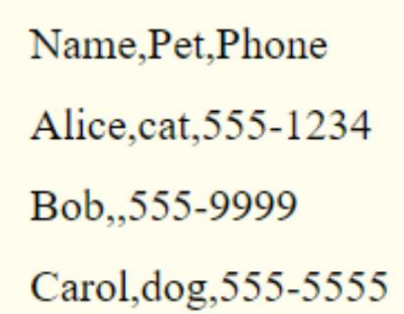
```
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
4/6/2015 12:46 Pears 14
4/8/2015 8:59 Oranges 52
4/10/2015 2:07 Apples 152
4/10/2015 18:10 Bananas 23
4/10/2015 2:40 Strawberries 98
```

DictReader and DictWriter CSV Objects (3)

- To contain a header row, write that row using writeheader().

➢Missing keys will be empty in the csv file.

- DictWriter objects use dictionaries to create CSV files: writerow()



```
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'})
>>> outputDictWriter.writerow({'Name': 'Bob', 'Phone': '555-9999'})
>>> outputDictWriter.writerow({'Phone': '555-5555', 'Name': 'Carol', 'Pet': 'dog'})
>>> outputFile.close()
```

Basics of Data Analytics

- Acquiring data

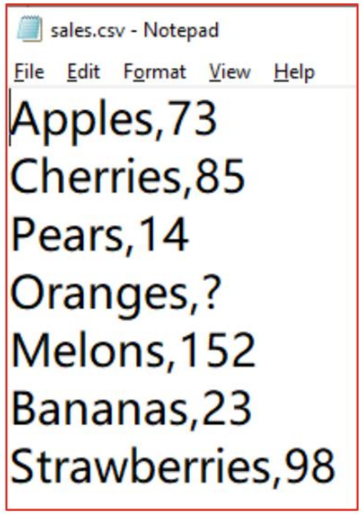
  - Access data from csv files
  - Data cleaning and munging

- Exploring data

  - Summary statistics for data
  - Charts for data

Data Cleaning and Munging

- Real-world data is need to be cleaned before using.
- Data Cleaning for missing values and bad data:
  - E.g., missing sales (‘?’) to default sales (20).
- Data Munging for wrongly formatted data:
  - E.g., string value (‘152’) to 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)
```

Basics of Data Analytics

- Acquiring data
  - Access data from csv files
  - Data cleaning and munging

- Exploring data
  - Summary statistics for data
  - Charts for data

Exploring Data

- Data exploration is the very first step in data analysis

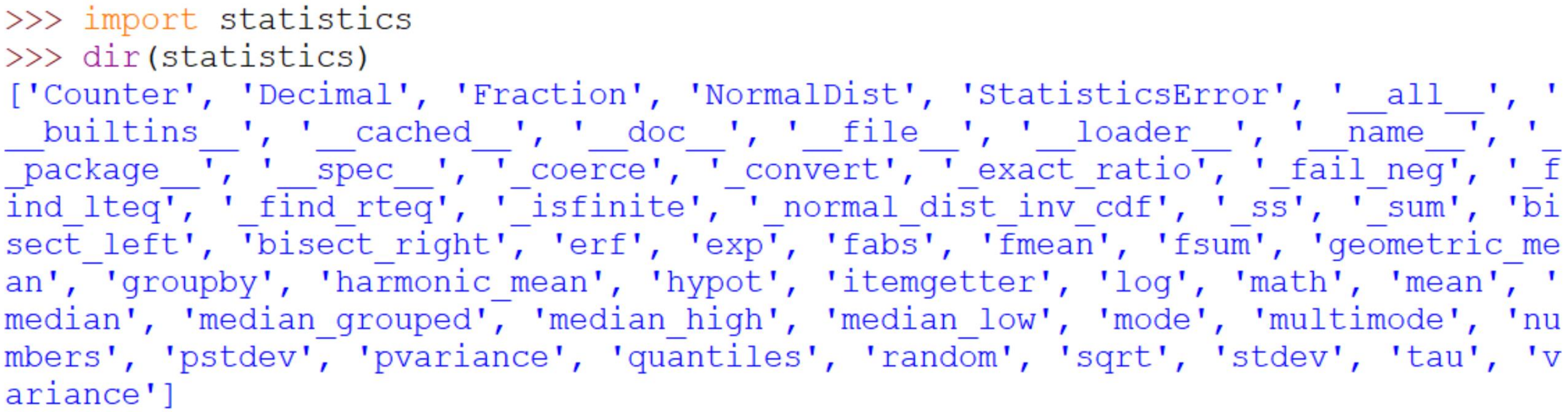
- Looking for patterns, characteristics, or points of interest.

➢Summary statistics: statistics, numpy, scipy, pandas modules.

➢Summary charts: matplotlib module.

Summary Statistics by Statistics Module (1)

- Python’s statistics module is a built-in Python library for descriptive statistics.



```
>>> 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)

- 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)
47.265209192385896
>>> max(x)
152.0
>>> min(x)
14.0
```

Basics of Data Analytics

- Acquiring data

➢Access data from csv files
➢Data cleaning and munging

- Exploring data

➢Summary statistics for data
➢Charts for data

Summary Charts by Matplotlib Module

- A fundamental part of the data analytics is data visualization.

- matplotlib contains a variety of tools for visualizing data.

➢Installation: pip install matplotlib

- pyplot is a useful submodule contained in matplotlib

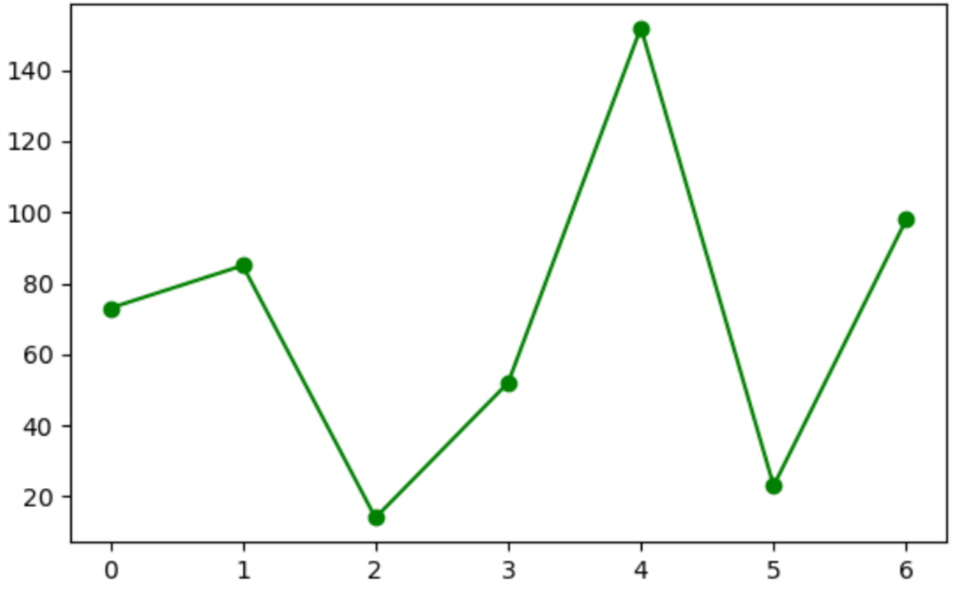
➢We often import matplotlib.pyplot using the alias plt:

➢import matplotlib.pyplot as plt

Making a Chart Using pyplot

- You can save the chart with savefig() method or display with show() method.

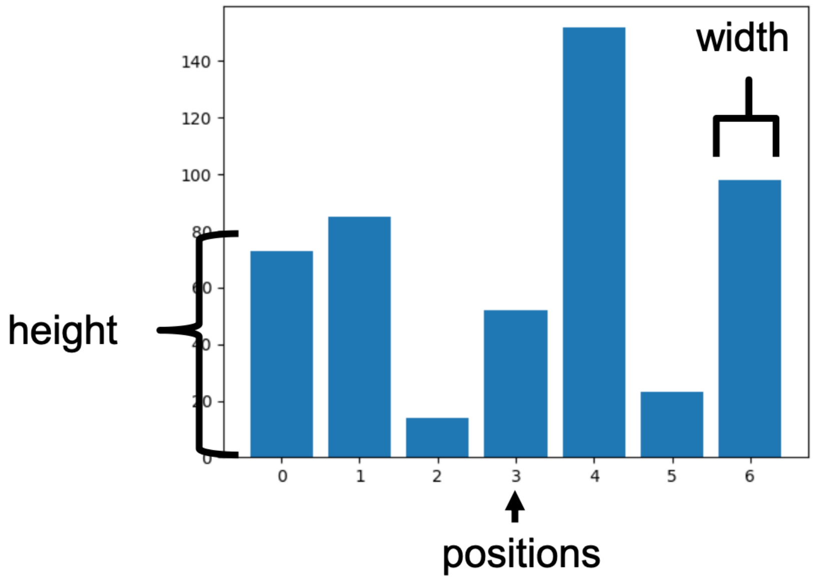
```
>>>import matplotlib.pyplot as plt
>>>y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
>>>plt.plot(y, color='green', marker='o',
linestyle='solid’)
>>>plt.savefig('plot.png')
>>>plt.show()
```



Bar Charts

- Use bar chart when you want to show how quantity varies among discrete set of items.
- matplotlib.pyplot.bar(positions, height, width=0.8) method.

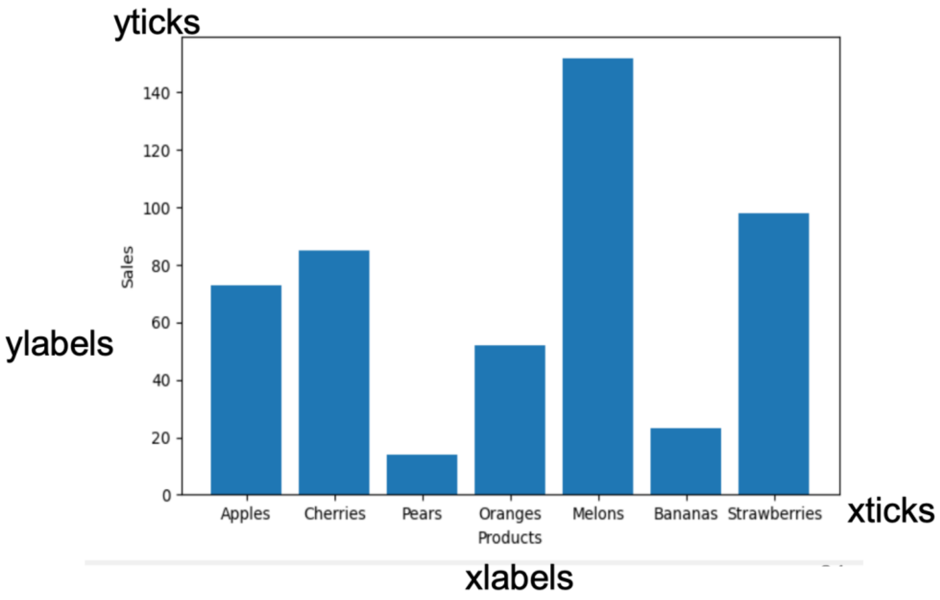
```
>>>import matplotlib.pyplot as plt
>>>y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
>>>x = list(range(len(y)))
>>>plt.bar(x,y)
>>>plt.show()
```



Add Labels and Ticks

- Labels: xlabel(text), ylabel(text)
- Ticks: xticks(ticks, labels), yticks(ticks, labels)

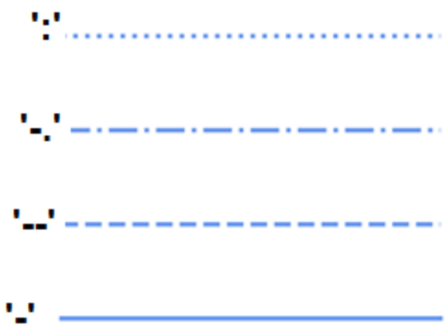
```
import matplotlib.pyplot as plt
y = [73.0, 85.0, 14.0, 52.0, 152.0, 23.0, 98.0]
products=['Apples', 'Cherries', 'Pears',
'Oranges', 'Melons', 'Bananas', 'Strawberries']
plt.bar(range(len(y)),y)
plt.ylabel('Sales')
plt.xlabel('Products')
plt.xticks(range(len(products)),products)
plt.show()
```



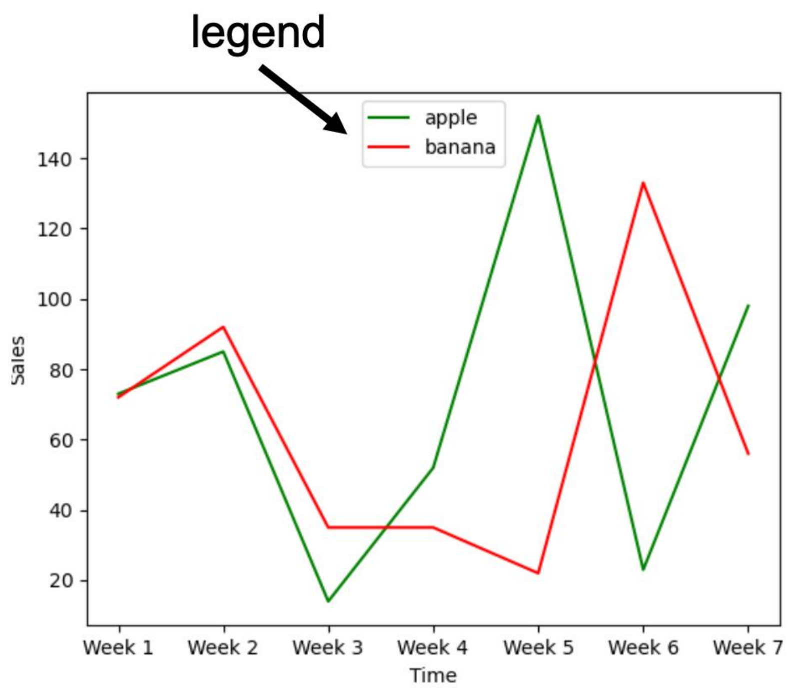
Line Charts

- Line chart for showing trends. matplotlib.pyplot.plot(x,y,fmt,label):
- Plot y versus x as lines and/or markers; fmt specifies the format of the line; label specifies the label of the line.

```
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()
```



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

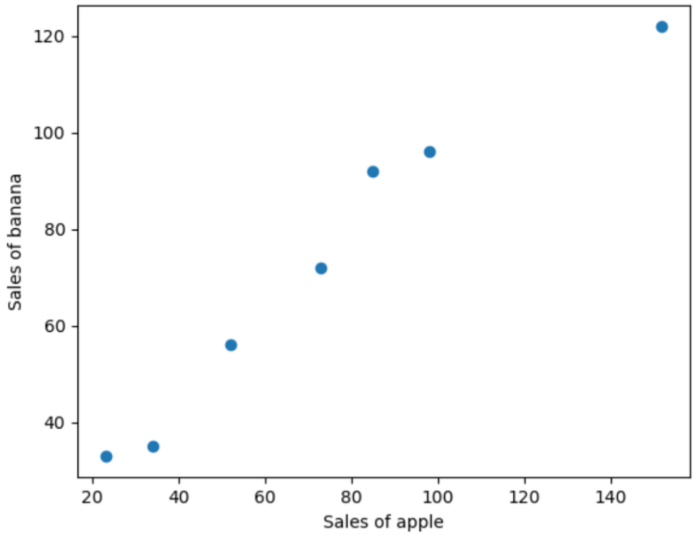


Scatterplots

- Scatterplot for visualizing the relationship between two paired sets of data.
- 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()
```



Acknowledgement

- Acknowledgements / Contributions
- These slides are Copyright 2010-Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.
- Initial Development: Charles Severance, University of Michigan School of Information
- Further Development: Zhou Xu, Hong Kong Polytechnic University
- Continuous development: Xiaoyu Wang, Hong Kong Polytechnic University