DATA GATHERING & PREPARATION DATA SCRAPING AND USING APIs

Gates, ANLY503
Georgetown University

TOPICS

1) Data Gathering

2) Data Cleaning/Processing (Data Wrangling)

3) Adding Features

4) Data Discretization

5) Scraping and APIs

6) Python 3 and Pandas

7) Data Reduction Techniques will be the next lecture

STEPS IN DATA PREPARATION

1) Often, data starts in a common spreadsheet such as Excel or csv.

2) Data can range from very clean, such as data collected from government sites, to very dirty and disorganized, such as data scraped from a web site.

3) All data must be cleaned, organized, and prepared before it can be used in visualization.

 - In many cases, remapping (like binning), transformation (like normalization), and reduction (like feature selection, PCA, …) are part of the process.

4) It is often suggested that gathering and preparing data is significantly more time consuming then visualizing the data.

STEPS CONTINUED

5) **Data preparation may also require many steps** such as determining what to do with

- outliers,
- missing values,
- possible false values,
- improperly formatted values, etc.

- There are several mathematical methods for trying to determine if a data value is an outlier or is simply a new piece of important information that was unexpected.
- Similarly, there are many options for replacing (or removing) missing data value.

Altering data (or collecting poor data) will affect the information and visualizations generated by that data.

DATA PREPROCESSING

Data in the real world is dirty

- incomplete: lacking attribute/feature values, lacking certain features of interest, or containing only aggregate data
- noisy: containing errors or outliers
- inconsistent: containing discrepancies in codes or names

Poor quality data = poor results (AKA garbage in garbage out)

Data Quality Measure: A multi-dimensional **measure** of data quality includes the following considerations:

- accuracy
- completeness
- consistency
- timeliness
- believability
- value added
- interpretability
- accessibility

COMMON TASKS IN DATA PREPROCESSING

Data cleaning

- Fill in missing values, smooth noisy data, identify or remove outliers, and resolve inconsistencies

Data integration

- Integration of multiple datasets

Data transformation

- Normalization (scaling to a specific range)
- Aggregation
- Binning
- Feature Creation
- Dimension Reduction
- Discretation

ANY PROBLEMS?

Customer Id	Zip	Gender	Income	Age	Marital Status	Transaction Amount
1001	10048	M	75000	C	M	5000
1002	J2S7K7	F	-40000	40	W	4000
1003	90210		10000000	45	S	7000
1004	6263	M	50000	0	S	1000
1005	55101	F	99999	30	D	3000

LAROSE example

MISSING DATA

Data is not always available

- E.g., many tuples have no recorded value for one or more attributes/variables, such as customer income in sales data, age, race, etc.

Missing data may be due to

- equipment malfunction
- inconsistent with other recorded data and thus deleted
- data not entered - due to misunderstanding or privacy
- certain data may not be considered important at the time of entry

Missing data may need to be inferred or removed.

Removal creates data loss and inference can alter the data information.

What is best to do – in which cases – and why?

HOW TO HANDLE MISSING DATA?

- Ignore the tuple: (Remove the row) This can severely reduce the dataset in some cases.
  - This can also affect or skew the data validity – why?
- Fill in the missing value manually: tedious + often infeasible.
  - What are other/better options for this?
- Use a global constant to fill in the missing value.
  - Why is the good and why is it not good?
- Use the attribute mean or median to fill in the missing value.
- Use the most probable value to fill in the missing value:
  - inference-based such as regression, Bayesian formula, decision tree, etc.

NOISY DATA

What is noise?

Noise is a type of **random error** in a measured variable.

Incorrect/noisey attribute values may be due to

- faulty data collection instruments (such as imprecision of readings)
- data entry problems (and human error)
- data transmission problems
- technology limitation

Inconsistency and Duplicates

- Duplicate records (repeats) – are these purposeful or accidental?
- Inconsistency in naming convention
  - such as FL, Florida, Fla, etc
  - such as $20000, 20,000, 20000.00, 20K, etc.

HOW TO HANDLE NOISY DATA?

Smoothing via Binning:

- sort data and partition into (equi-depth) bins
- smooth bins by mean, median, boundaries, etc.

Binning for data transformation from continuous/quantitative to categorical:

- Choose the number of bins (classes/categories)
- Place data into bins

Clustering - detect and remove outliers

Semi-automated method: combined computer and human inspection

- detect suspicious values and check manually

Regression

- smooth by fitting the data into regression functions

SIMPLE DISCRETIZATION METHODS: BINNING

Equal-width (distance) partitioning:

- It divides the range into *N* intervals of equal size: uniform grid
  - width = (*MAX*-*MIN*)/*N*.
- Most straightforward
- Outliers may dominate presentation
- Skewed data is not handled well.

Equal-depth (frequency) partitioning:

- It divides the range into *N* intervals, each containing approximately **same number of samples**
- Good data scaling
- Managing categorical attributes can be tricky.

EXAMPLE: EQUAL-WIDTH BINNING

Dataset: 4, 8, 9, 15, 21, 21, 24, 25, 26, 28, 29, 34

Suppose I want 5 bins

Max value = 34

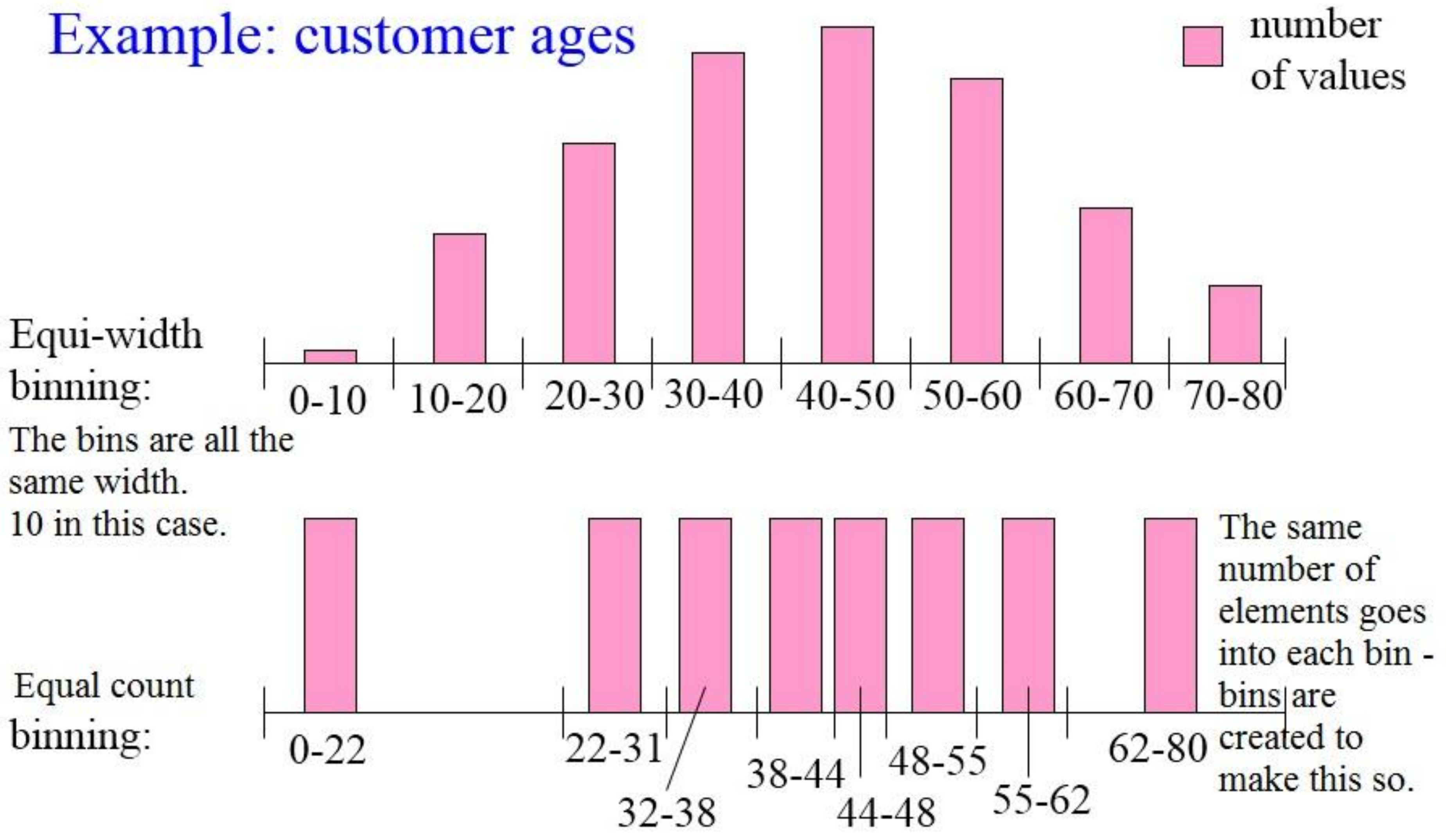
Min value = 4

Data width = 34 – 4 = 30

Bin width = (max – min)/#bins = 30 / 5 = 6

Bin 1 = 4 – 10:	**4, 8, 9**
Bin 2 = 10 – 16:	**15**
Bin 3 = 16 – 22:	**21, 21**
Bin 4 = 22 – 28:	**24, 25, 26, 28**
Bin 5 = 28 – 34:	**29, 34**

BINNING OPTIONS



BINNING PART 2

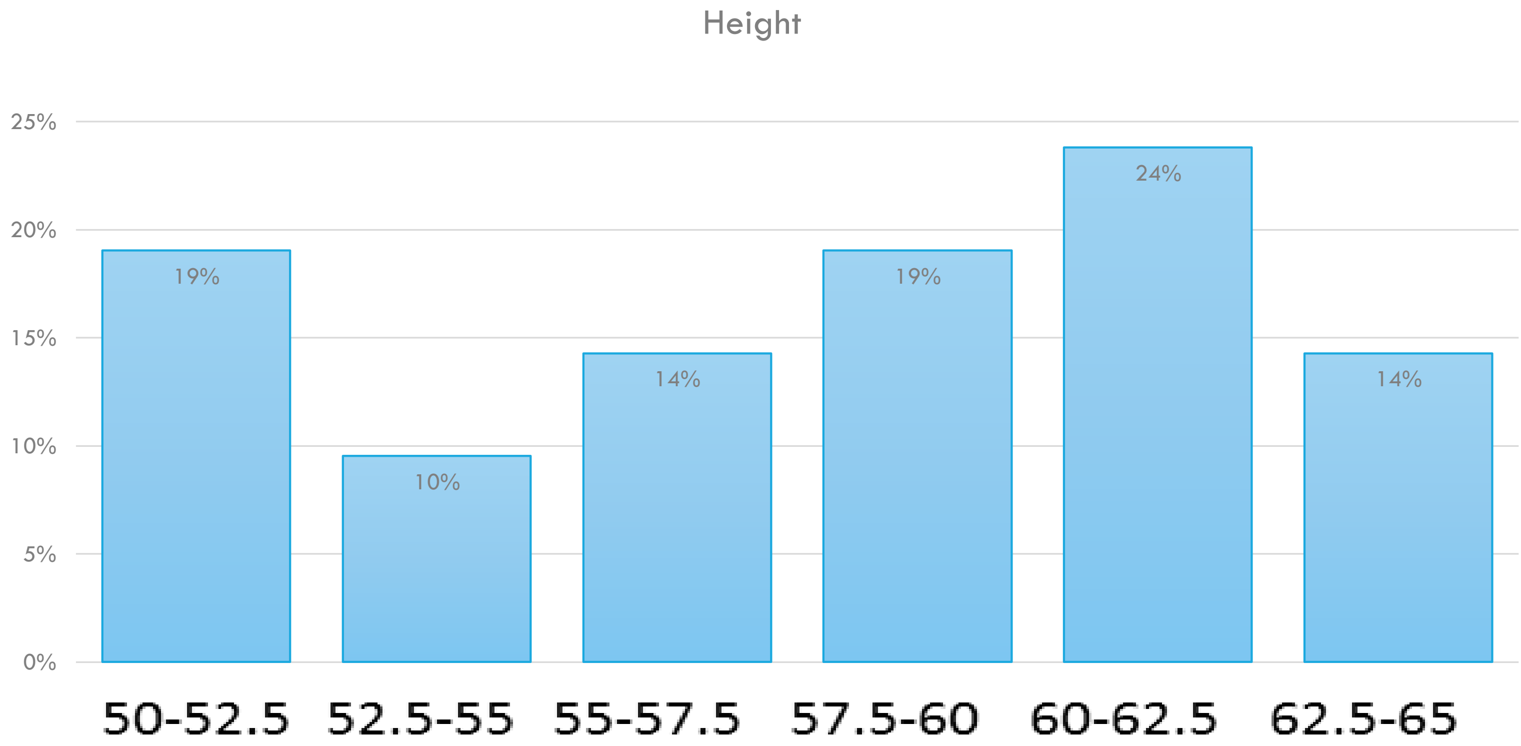
54.876985
63.345339
52.478001
62.047462
59.692849
63.116668
52.394022
60.539069
56.14605
56.553081
57.899076
59.550554
61.852781
55.549742
50.020629
62.877525
60.620873
54.269527
50.999137
59.763285
61.565763

Example Dataset: height in inches

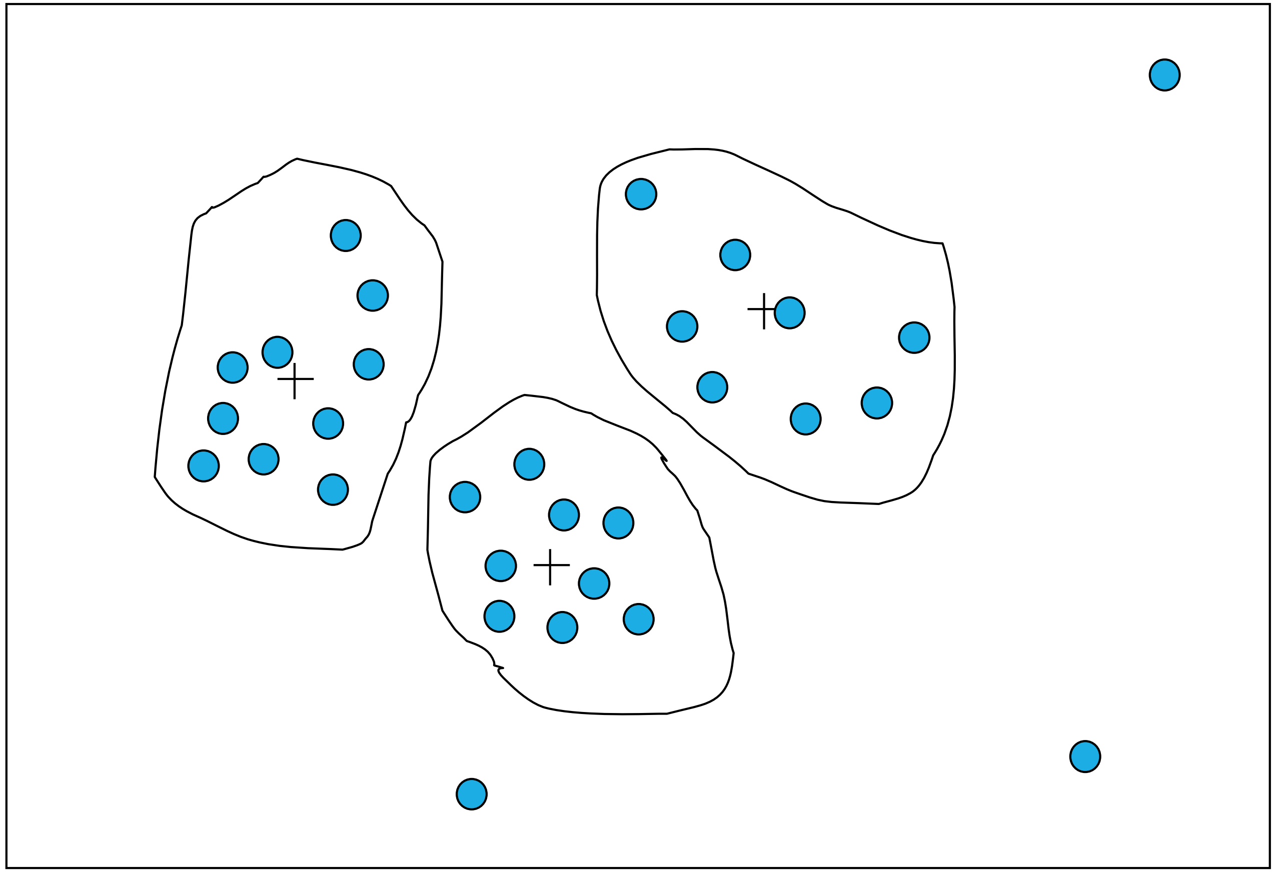
(65 – 50) /6 = 15/6 = 2.5 **So, the WIDTH of each bin is 2.5.**

Height	Range	Absolute Frequency	Relative Frequency	Cumulative Frequency
Bin 1	50 – 52.5	4	4/21=19.1%	19.1%
Bin 2	52.5 - 55	2	2/21=9.5%	19.1%+9.5%
Bin 3	55 – 57.5	3	3/21=14.3%	19.1%+9.5%+14.3%
Bin 4	57.5 - 60	4	4/21=19.1%	19.1%+9.5%+14.3%+19.1%
Bin 5	60 – 62.5	5	5/21=23.8%	19.1%+9.5%+14.3%+19.1%+23.8%
Bin 6	62.5 - 65	3	3/21=14.3%	19.1%+9.5%+14.3%+19.1%+23.8%+ 14.3%
Total		**21**	100%	100%

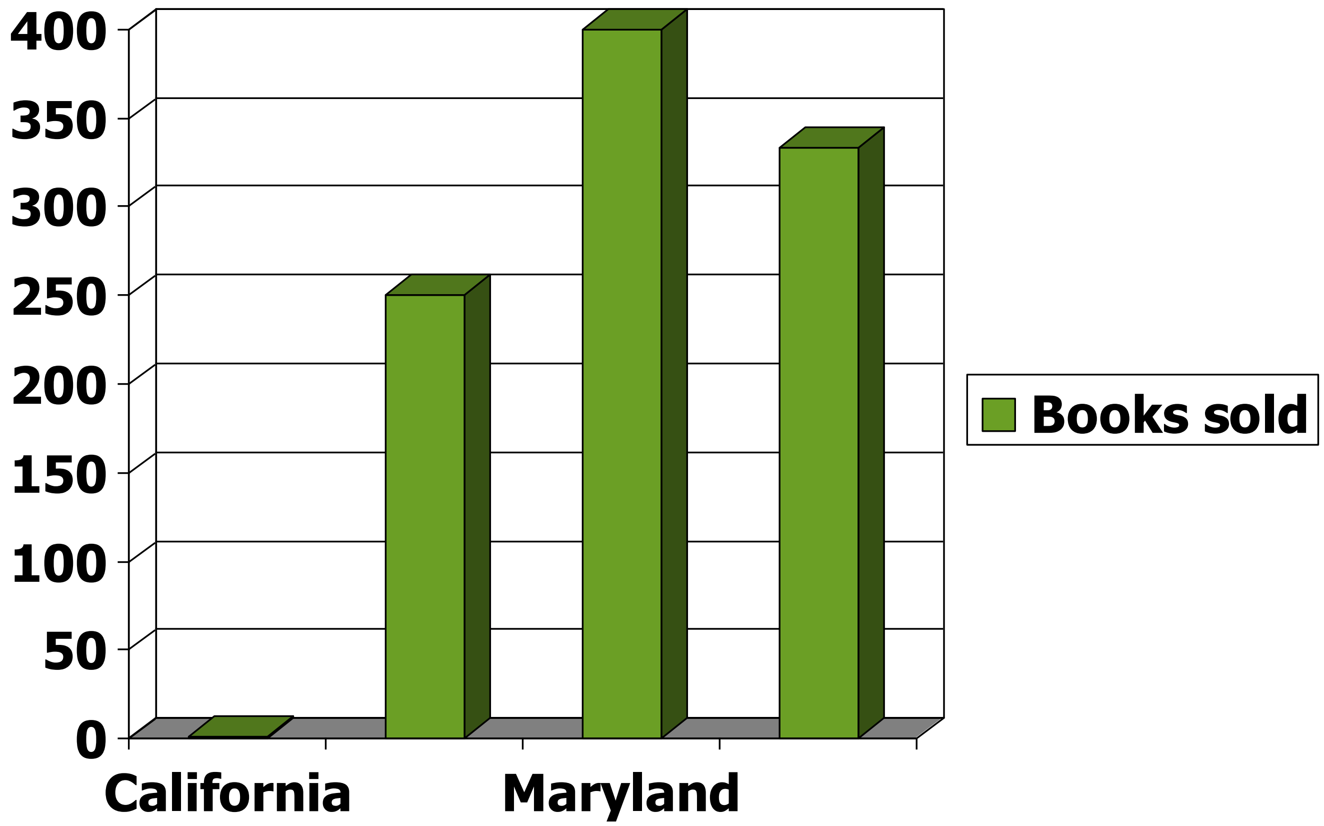
BAR GRAPH OF BINNED DATA



HANDLING OUTLIERS: CLUSTER ANALYSIS



OUTLIER DETECTION USING HISTOGRAM



OUTLIER DETECTION USING IQR

The IQR is defined as the difference between the upper quartile and the lower quartile of a data set, sometimes written as Q3 - Q1.

A data value is an outlier if:

- It is 1.5*IQR (or more) **below** Q1
- It is 1.5*IQR (or more) **above** Q3

This same concept can also be done using standard deviations when data is generally normally distributed.

OUTLIERS NOT REMOVED

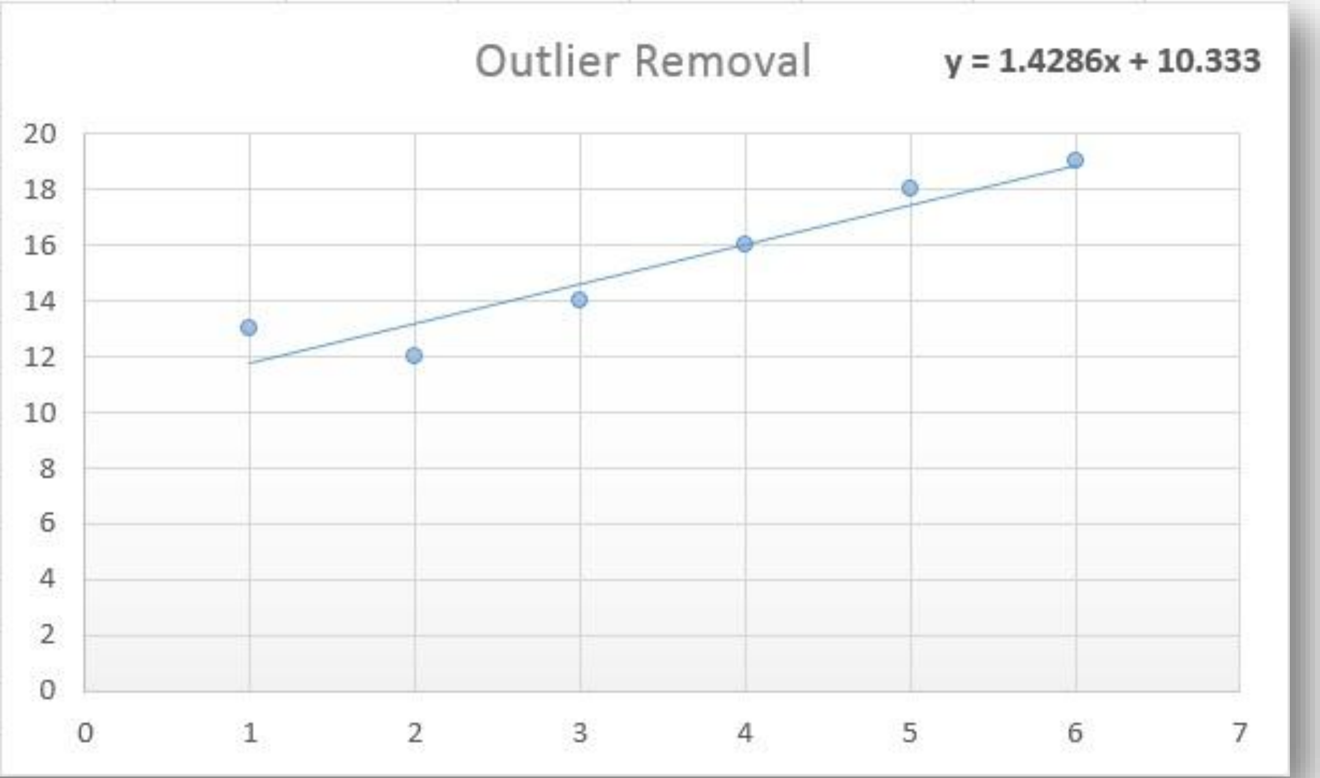
Here the rate of change (slope) of the line is 1.83

x	y
1	13
2	12
3	14
4	30
5	18
6	19

OUTLIER REMOVED (30 REMOVED)

Here the rate of change (slope) of the line is 1.43

x	y
1	13
2	12
3	14
4	16
5	18
6	19



HOW TO HANDLE INCONSISTENT DATA?

Determining that some data is inconsistent or incorrect:

1) **Visually** (this is a starting place)
2) **Writing and applying a program** that checks the data
   - Such programs can check for correct labeling, correcting naming, expected data ranges or categories, etc.
   - For example, an INCOME attribute should not be “0” (this would actually be a missing value), should not be negative, and should be within an expected range given the sample.

Programs and regular expressions (along with creativity and some knowledge of the data) can be created to find and correct data issues.

DATA TRANSFORMATION

Smoothing: remove noise from data (binning, clustering, regression)

Aggregation: summarization of data – including averages, totals, medians, other summary statistics, dataset merging, etc.

Generalization: concept hierarchy - organizes concepts (i.e., attribute values) hierarchically and is usually associated with each dimension in a dataset

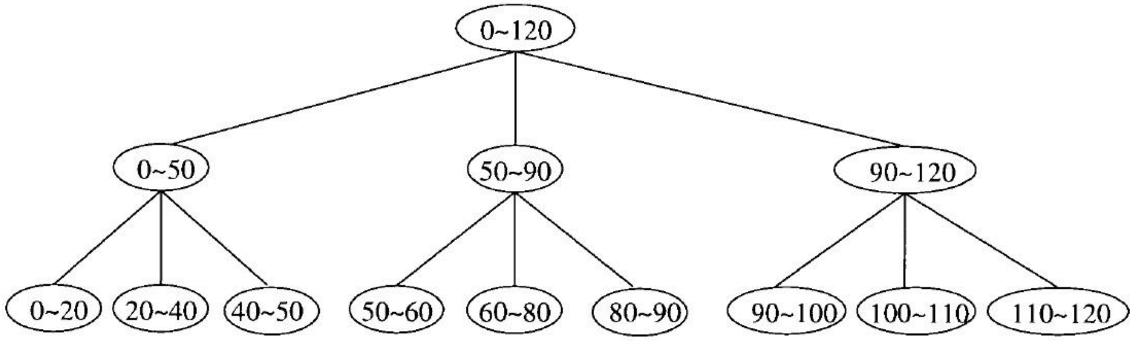
Normalization: scaled to fall within a small, specified range

- min-max normalization
- z-score normalization
- normalization by decimal scaling

Attribute/feature construction

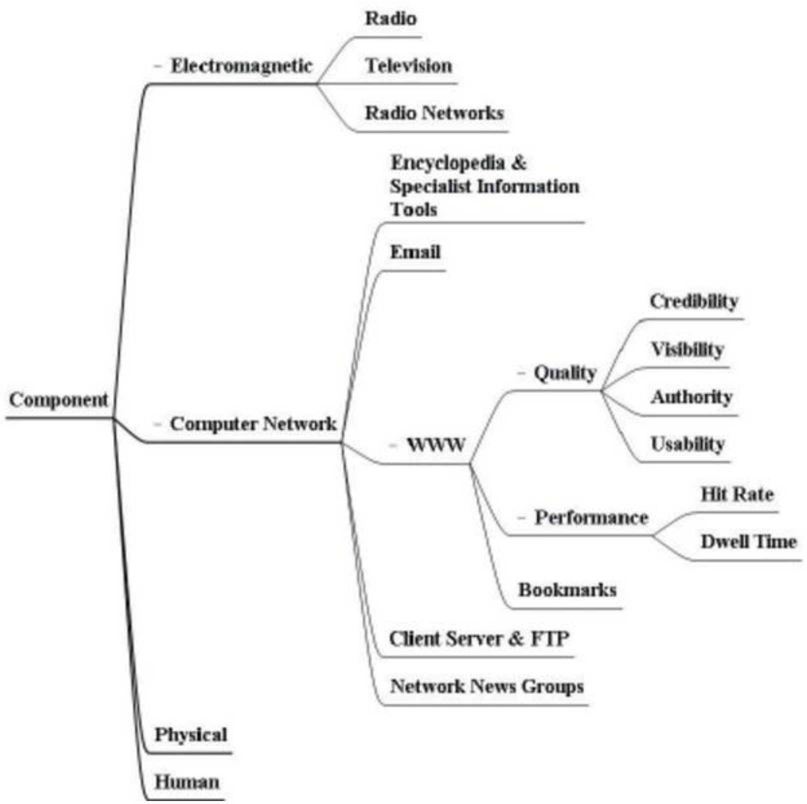
- New attributes constructed from the given ones

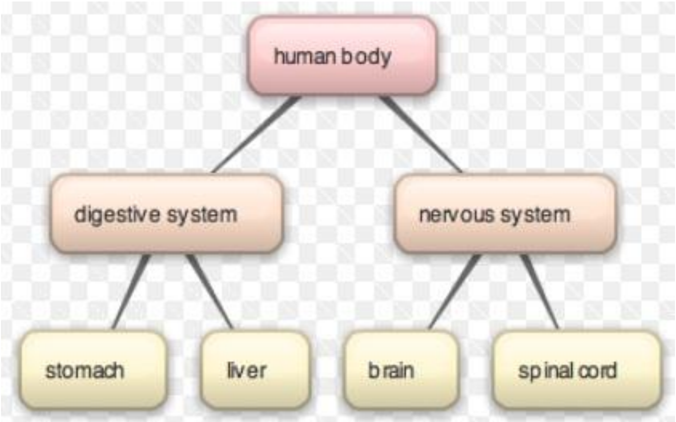
EXAMPLES OF CONCEPT HIERARCHIES



From: summit.sfu.ca/system/files/iritems1/7984/b18914287.pdf

Figure 4.2: A concept hierarchy for attribute A generated by algorithm AGHF.





DATA TRANSFORMATION: NORMALIZATION

Particularly useful for classification (distance measurements, NN classification, etc)

min-max normalization

$$v' = \frac{v - min_A}{max_A - min_A}(newmax_A - newmin_A) + newmin_A$$

z-score normalization

$$v' = \frac{v - mean_A}{stdev_A}$$

normalization by decimal scaling

$$v' = \frac{v}{10^j}$$

Where j is the smallest integer such that Max($|v'|$)<1

WHAT DOES IT MEAN TO "NORMALIZE DATA" ?

Most census data are **counts** - the number of people (or households, families, housing units, etc.) who reside within a particular census geography (e.g., a census block group) and meet some criterion (e.g., have at least a high school education).

To judge whether that count is 'high' or 'low' we need an **appropriate basis for comparison** (e.g., the count of all people residing in that geography who are adults with a known education level)….

Normalizing census data allows you to interpret data variables **relative to the universe** in which they exist (by dividing the counts by the total count for the appropriate universe).

http://web.mit.edu/11.520/www/labs/lab5/normalize.html

CONSIDER THIS EXAMPLE :

Suppose you wanted to visualize the **fraction of rents less than $500 in each block group**.

Why normalize the data?

Comparing the raw numbers of housing units per block group may be deceiving, as the total number of renter-occupied housing units will vary from one block group to the next.

By **dividing** the number housing units with rent less than $500 by the total number of housing units, we obtain a fraction of the total occupied housing units with rents under $500.

This fraction may then be compared fairly **among block groups**. (Rate per Block)

OTHER EXAMPLES FOR NORMALIZATION

1) Suppose you are looking at counts of lung cancer for each state and you want to compare the states to each other. Should you normalize? How and why?

2) Suppose you want to look at the amount of funding high schools get in each county. Should you normalize per student?

3) Suppose you want to compare food recalls for each state or county. What should you normalize by?

When should you not normalize data?

SCRAPING, APIs AND CLEANING DATA

Reminder Slides, Gates

URLLIB VERSUS REQUESTS

Web Gathering Examples

Ami Gates

WHAT IS DATA SCRAPING?

1) Data scraping generally refers to “grabbing” or gathering data from the Web.

2) However, data can be scraped from a database, etc.

3) Scraping can be done “by hand” – such as writing Python code and using regular expressions (RE), parsing, and appropriate HTML (Get and Post).

4) Scraping can be done using an API. For example, Twitter, AirNow, and other Sites offer API and KEY options to access data from their sites.

I will post a few API and RE scrape examples (in Python 3) on the class Site.

EXAMPLE

Using the AirNow API to collect air pollution data.

1) The API and related information is here:

https://docs.airnowapi.org/

2) You will need to register so that you can get a KEY

3) Most sites that offer APIs to access their data also **limit** the number of calls to their dataset per unit of time.

4) AirNow Web Services to view request URLs etc.

https://docs.airnowapi.org/webservices

5) Seeing what a query looks like

https://docs.airnowapi.org/forecastsbyzip/query

CREATE THE BASE URL AND PARAMETERS

```
BaseURL="http://www.airnowapi.org/aq/observation/latLong/historical/"
    # Example complete URL for AirNow
    #http://www.airnowapi.org/aq/observation/latLong/historical/?
    #format=application/json&latitude=38.3651&longitude=-114.4141&
    #date=2016-09-10T00-0000&distance=25&API_KEY=867C-XXXX

    URLPost = {'API_KEY': 'D9AA91E7-070D-4221-867C-EFF5EXXXXXXXXXXXX',
                'latitude': 38.9,
                'longitude':-77.3,
                'date': '2016-09-10T00-0000',
                'distance': '5',
                'format': 'application/json'}
```

USING THE “REQUESTS” LIBRARY

```
import requests
    response=requests.get(BaseURL, URLPost)
    jsontxt = response.json()
    #print(jsontxt, "\n")

    for list in jsontxt:
        AQIType = list['ParameterName']
        City=list['ReportingArea']
        AQIValue=list['AQI']
        print("For Location ", City, " the AQI for ", AQIType, "is ", AQIValue, "\n")
```

USING THE “URLLIB” LIBRARY

```
import urllib
from urllib.request import urlopen
import json
URL=BaseURL + "?"+ urllib.parse.urlencode(URLPost)
WebURL=urlopen(URL)
data=WebURL.read()
encoding = WebURL.info().get_content_charset('utf-8')
jsontxt = json.loads(data.decode(encoding))
# print(jsontxt, "\n")
for list in jsontxt:
    AQIType = list['ParameterName']
    City=list['ReportingArea']
    AQIValue=list['AQI']
    print("For Location ", City, " the AQI for ", AQIType, "is ", AQIValue, "\n")
```

THE OUTPUT IN BOTH CASES

For Location Northern Virginia the AQI for OZONE is 51

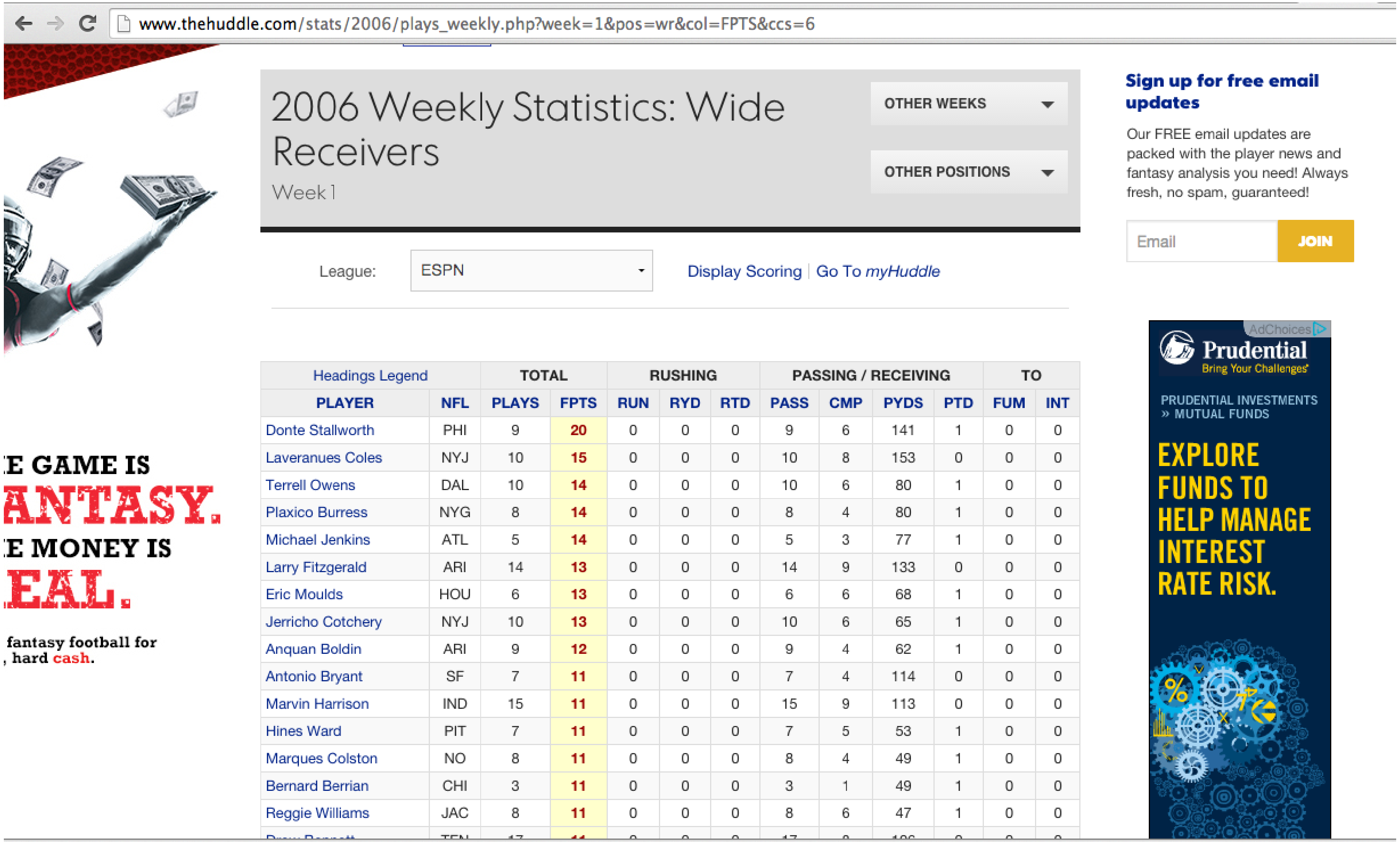
For Location Northern Virginia the AQI for PM2.5 is 55

For Location Northern Virginia the AQI for PM10 is 20

GETTING DATA THAT IS LISTED ON THE WEB

```
#GetDiabetesData.py

#Gates

import pandas


url = "https://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"

attributeNames = ['times-pregnant', 'plasma-glucose-concentration', 'diastolic-blood-pressure', 'skin-fold-thickness', 'serum-insulin', 'bmi', 'ped-function', 'age', 'class']

myData = pandas.read_csv(url, names=attributeNames)

#####################################################

# print some of the data – you can also write data to a file, etc.

#####################################################

# First few rows

print(myData.head(20))
```

EXAMPLE 2 – HTML TABLE



www.thehuddle.com/stats/2006/plays_weekly.php?week=1&pos=wr&col=FPTS&ccs=6

2006 Weekly Statistics: Wide Receivers

Week 1

OTHER WEEKS

OTHER POSITIONS

League: ESPN

Display Scoring | Go To *myHuddle*

Headings Legend		TOTAL		RUSHING			PASSING / RECEIVING				TO	
PLAYER	NFL	PLAYS	FPTS	RUN	RYD	RTD	PASS	CMP	PYDS	PTD	FUM	INT
Donte Stallworth	PHI	9	20	0	0	0	9	6	141	1	0	0
Laveranues Coles	NYJ	10	15	0	0	0	10	8	153	0	0	0
Terrell Owens	DAL	10	14	0	0	0	10	6	80	1	0	0
Plaxico Burress	NYG	8	14	0	0	0	8	4	80	1	0	0
Michael Jenkins	ATL	5	14	0	0	0	5	3	77	1	0	0
Larry Fitzgerald	ARI	14	13	0	0	0	14	9	133	0	0	0
Eric Moulds	HOU	6	13	0	0	0	6	6	68	1	0	0
Jerricho Cotchery	NYJ	10	13	0	0	0	10	6	65	1	0	0
Anquan Boldin	ARI	9	12	0	0	0	9	4	62	1	0	0
Antonio Bryant	SF	7	11	0	0	0	7	4	114	0	0	0
Marvin Harrison	IND	15	11	0	0	0	15	9	113	0	0	0
Hines Ward	PIT	7	11	0	0	0	7	5	53	1	0	0
Marques Colston	NO	8	11	0	0	0	8	4	49	1	0	0
Bernard Berrian	CHI	3	11	0	0	0	3	1	49	1	0	0
Reggie Williams	JAC	8	11	0	0	0	8	6	47	1	0	0

USING PYTHON REQUESTS AND BEAUTIFULSOUP (BS)

```
#Sports Data with BS and CV

#Author Gates and Singh

# Additional Ref: “Web Scraping with Python, Mitchell”

#SportsUsingRequests.py

from bs4 import BeautifulSoup

import requests

import csv

def Sports():

    csvName = "SportsData.csv“

url="http://www.thehuddle.com/stats/2006/plays_weekly.php?we
ek=1&pos=wr&col=FPTS&ccs=6"

    page=requests.get(url)

    soup = BeautifulSoup(page.text, "lxml")
```

```
 table=soup.findAll("table")[0]
    All_TR=table.findAll("tr")
    csvFile=open(csvName, "wt")
    playerwriter = csv.writer(csvFile, delimiter=',')
    playerwriter.writerow(['Player', 'Team', 'Plays',
'Fpts', 'Run', 'Ryd', 'RunTD', 'Pass', 'Cmp',
    'Pyds', 'PTD', 'Fum', 'Int'  ])

 for nextTR in All_TR:
        csvRow=[]
        for nextTD in nextTR.findAll("td"):
            csvRow.append(nextTD.text.strip())
        playerwriter.writerow(csvRow)
        ## TO SEE WHAT IS WRITING:
            ##print(csvRow)
    csvFile.close()

Sports()
```

RESULTS IN CSV FILE

Player,Team,Plays,Fpts,Run,Ryd,RunTD,Pass,Cmp,Pyds,PTD,Fum,Int

Donte Stallworth,PHI,9,20,0,0,0,9,6,141,1,0,0

Laveranues Coles,NYJ,10,15,0,0,0,10,8,153,0,0,0

Terrell Owens,DAL,10,14,0,0,0,10,6,80,1,0,0

Plaxico Burress,NYG,8,14,0,0,0,8,4,80,1,0,0

Michael Jenkins,ATL,5,14,0,0,0,5,3,77,1,0,0

Larry Fitzgerald,ARI,14,13,0,0,0,14,9,133,0,0,0

…

COMPARISON: URLLIB VS REQUESTS

urllib	**requests**
Standard Python 3 library	Standard in Anaconda – but must be installed from some Python 3 installations
can request data across web	can request data across web
can handle cookies	can handle cookies
change meta data such as headers and user-agent	**allows complete customization of headers**

Useful reference: Pages 178 – 182, “Web Scraping with Python”, Mitchell

GETTING WEB DATA: GET AND POST

```
import requests
# GET REQUEST
myRequest = requests.get('https://api.github.com', auth=('user', 'pass'))

# POST REQUEST
my Request = requests.post('http://myurl.com', data='data to post')

myRequest.status_code
myRequest.headers['content-type']'
```

EXAMPLE USING GET TO SCRAPE DATA

```
import requests
response=requests.get("http://www.foxnews.com/")
txt = response.text
print(txt)
```

EXAMPLE USING GET TO ANSWER SPECIFIC QUERY

```
# Find Google results for data science

import requests
response=requests.get("https://www.google.com/?q=data+science")

txt = response.text
print(txt)
```

CLEANER VERSION OF EXAMPLE

```
# Find Google results for data science

import requests
myUrl = 'http://google.com'
query = ‘data+science’

response = requests.get(myUrl, query)
txt = response.text
print(txt)
```

APIs – APPLICATION PROGRAMMING INTERFACE

It is a contract that specifies **how a program interacts with an application**.

Web site APIs specify the process for authentication, important classes, URLs, etc.

Application Program[ing] Interface

Many sites offer an API that allows for gathering data from their site

Google has many APIs

Twitter, Wikipedia, ESPN, … have APIs

Most API usage requires registering for the API use and gaining a "key".

Ref: Gates Chapter 13 Python Book (see class Site)

API ENDPOINTS AND KEYS

API-defined URL or location where particular data is stored.

You can retrieve data from an API by making a **URL request to an endpoint** using **specific query parameters.**

Most Web-based APIs require a key.

The key is unique and identifies each user.

The key is associated with each request.

XML AND JSON

Return values from the APIs are in XML format or JSON format – you choose.

XML looks like html markup

<student>

<firstName>Jane</firstName>

<lastName>Mouse</lastName

>

</student>

JSON is not tagged the same way.

"students": [

{

"firstName": "Jane",

"lastName": "Mouse"

} ,

{

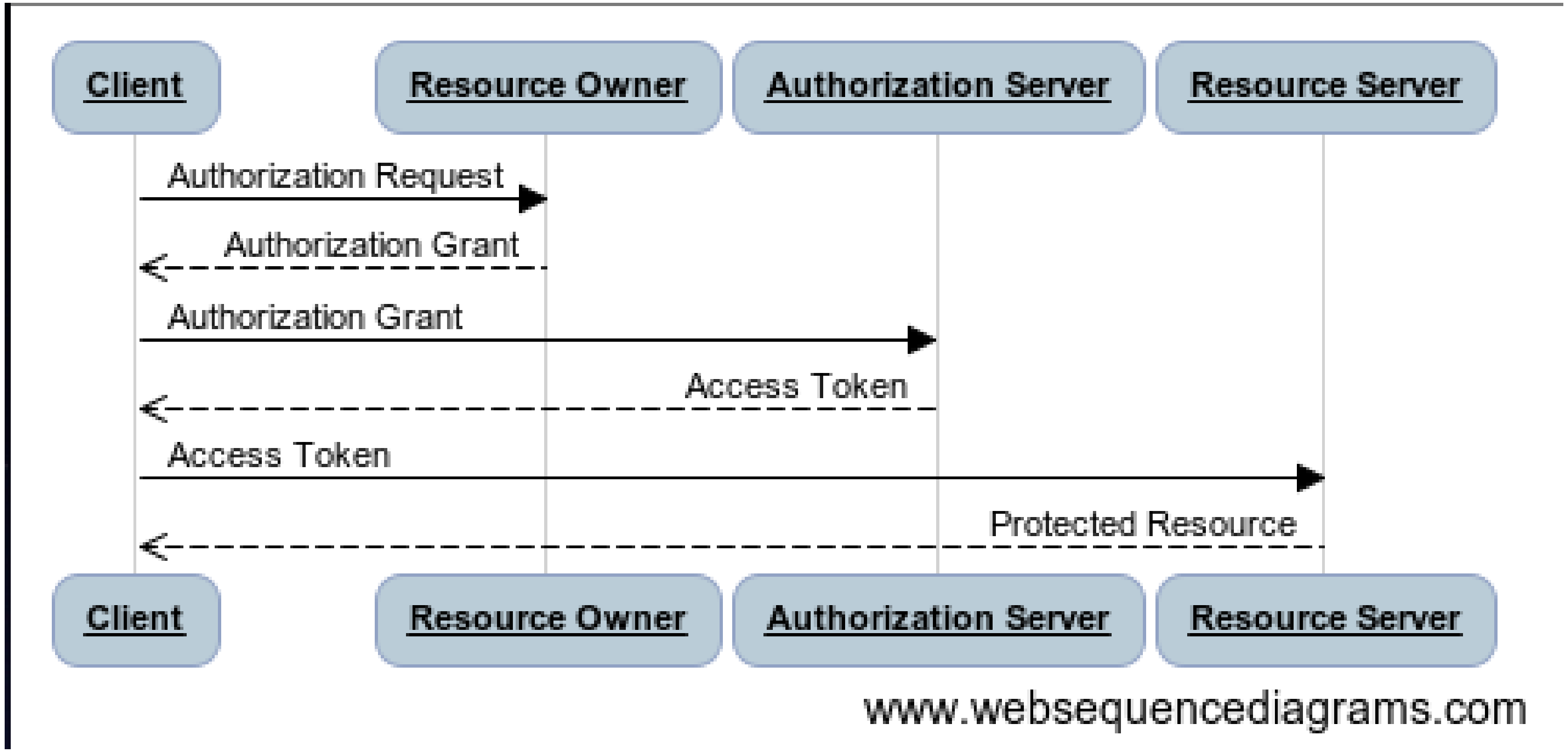
"firstName": "Jack",

"lastName": "Dog"

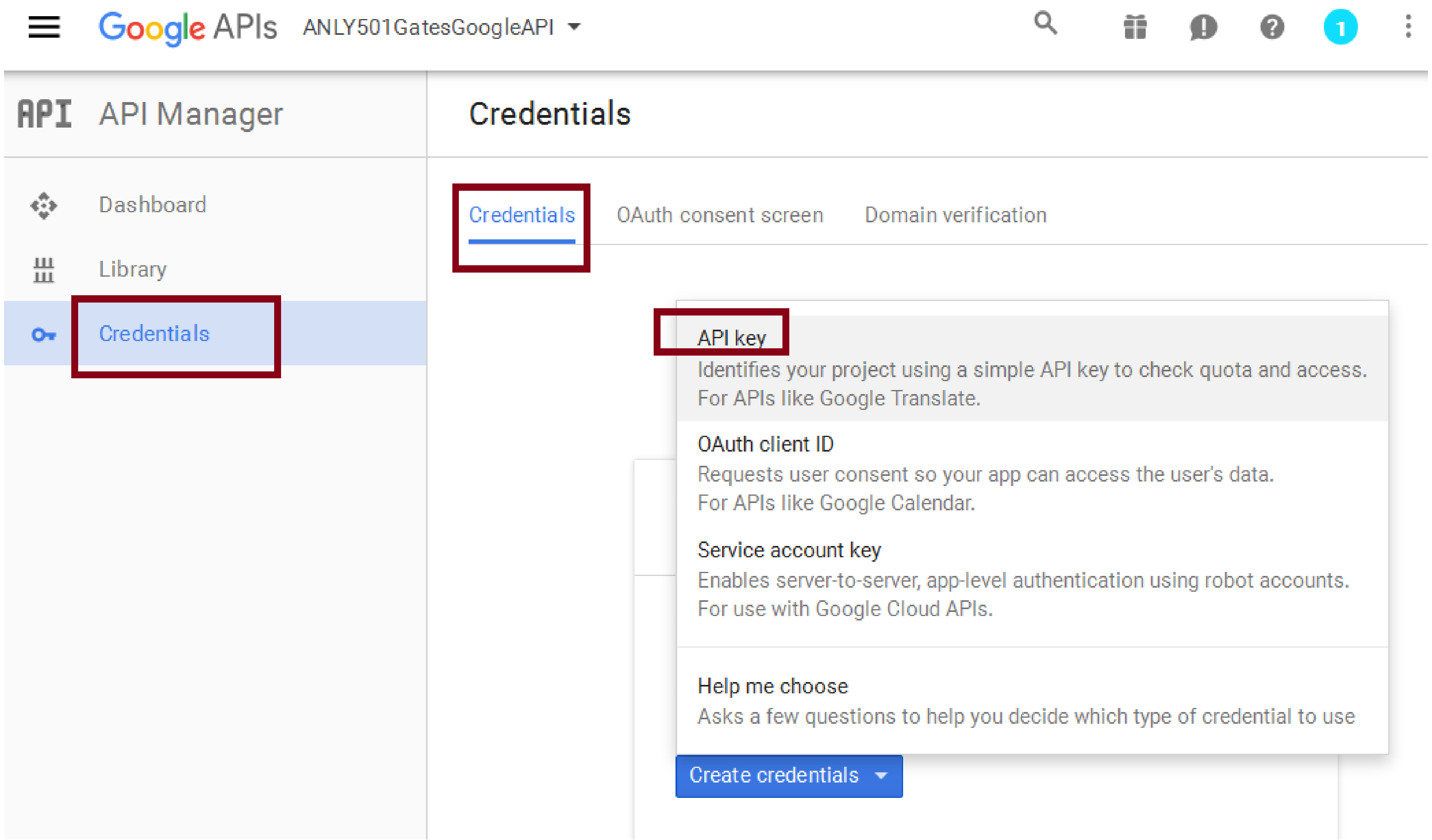
}

]

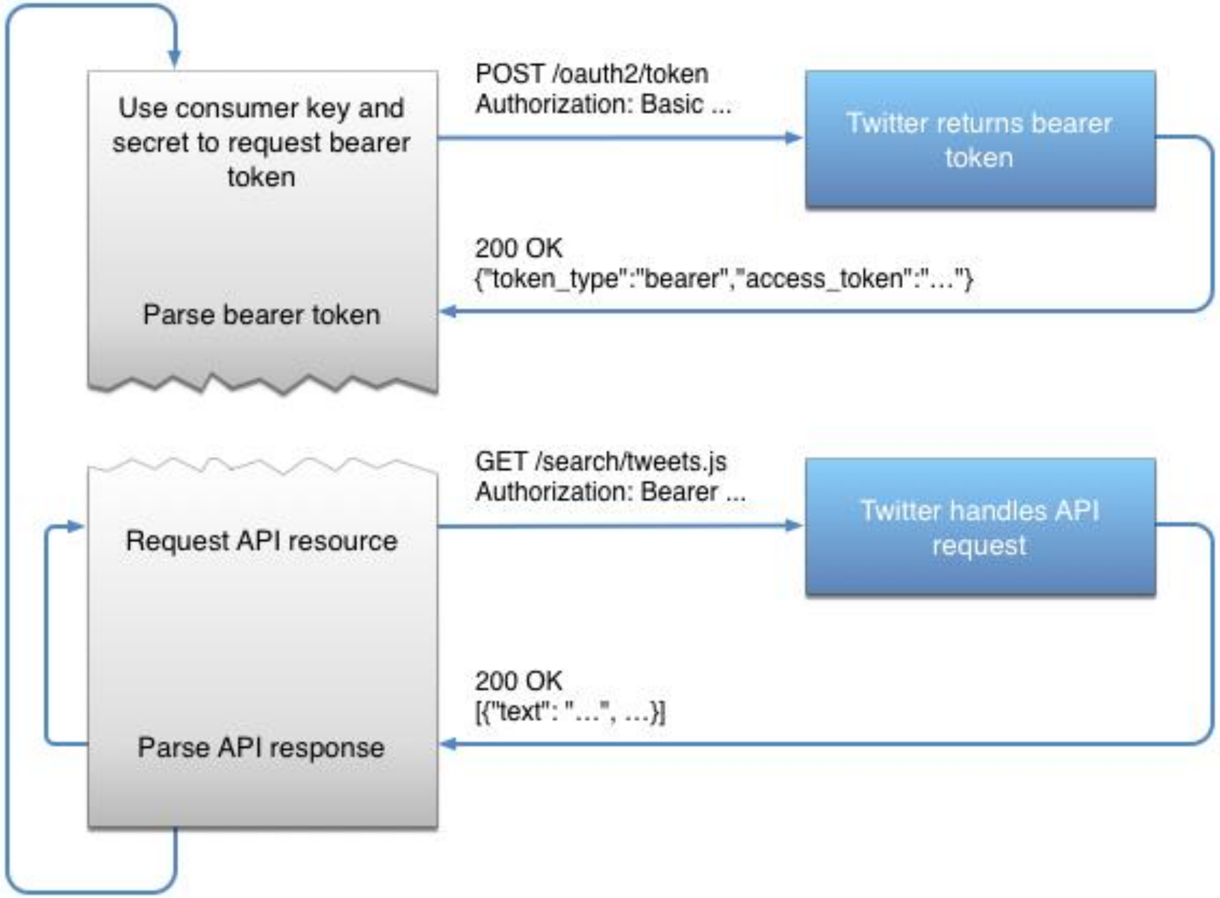
OAUTH – ABSTRACT FLOW



GOOGLE API, OAUTH AND KEY



TWITTER’S VERSION



CLASS ACTIVITY: USING WEB REQUESTS AND APIs

Gates

PRACTICE SCRAPING: IN CLASS ACTIVITY

Instructions: (see the next few slides)

1) Go to the AirNow site.

https://docs.airnowapi.org/

Create a log in and get a KEY.

2) Then, review the syntax of an API query

https://docs.airnowapi.org/webservices

For example, a possible query that can be built is

http://www.airnowapi.org/aq/forecast/zipCode/?format=text/csv&zipCode=20002&date=2017-02-17&distance=25&API_KEY=D9AA91E7-xxxx-xxxx-xxxx-xxxxxxx

CONTINUED

3) Break down the URL query:

Example:

http://www.airnowapi.org/aq/forecast/zipCode/?format=text/csv&zipCode=20002&date=2017-02-17&distance=25&API_KEY=D9AA91E7-xxxx-xxxx-xxxx-xxxxxxx

breaks down to:

BaseURL = “http://www.airnowapi.org/aq/forecast/zipCode/”

format=“text/csv”

zipCode=“20002”

date=“2017-02-17

distance=“25”

API_KEY=“xxxxxxxxxxxxxxxxx”

CONTINUED

Use this example:

https://drive.google.com/file/d/0B4RXVYeUUKitV3ViZzFDLXFtclU/view?usp=sharing

(This link is also on the class website under the Homework Assignment for this week)

Complete the following tasks:

1) Using your AirNow login and key, update and run the code from above. Look at the differences between “request” and “urllib”. Also look at the use of JSON.

2) Update the code to use 4 different date, a different distance, and zip codes rather than lat and long.

PANDAS AND CLEANING

This is a review of Python 3 and pandas – it is not part of the lecture – just FYI

TEXT FILES: BASIC WRITING

```
TextFile="MyTextFile.txt"

File1=open(TextFile, "w")

DictText='{"ID":"D1234", "Firstname":"John", "Car":["BMW", "Honda","Kia"]}\n'

File1.write(DictText)

DictText='{"ID":"D5555", "Firstname":"Benny", "Car":["Ford", "Mazda","Kia"]}\n'

File1.write(DictText)

File1.close()
```

TEXT FILES: BASIC READING

File1=**open**(TextFile, "r")

data=File1.**read**()

print("Entire File Contents ", data)

File1.seek(0)

data=File1.**readline**(11)

print("Read first line and first 11 chars ", data)

File1.seek(0)

data=File1.**readlines**()

print("Reads until the End of File (EOF) ", data)

TEXT FILES: APPENDING

```
TextFile="MyTextFile.txt"
File1=open(TextFile, "a")
DictText='{"ID":"D7878", "Firstname":"Paul", "Car":["Chevy"]}\n'
File1.write(DictText)
DictText='{"ID":"D9199", "Firstname":"Alan", "Car":["Mazda","Kia"]}\n'
File1.write(DictText)
File1.close()
File1=open(TextFile, "r")
data=File1.read()
print(data)
print(len(data))
print(type(data))
File1.close()
```

CSV FILES (COMMA SEPARATED VALUES): BASIC WRITING

```
csvFile="MyCSVFile.csv"
File2=open(csvFile, "w", newline='')
CSVList=(["FirstName", "John", "Lastname", "Smith", "Car", "BMW", "Zipcode", "20001"])
Fwriter=csv.writer(File2, delimiter=",")
Fwriter.writerow((CSVList[0],CSVList[2],CSVList[4],CSVList[6]))
Fwriter.writerow((CSVList[1],CSVList[3],CSVList[5],CSVList[7]))
File2.close()
```

CSV: BASIC READING

```
File2=open(csvFile, newline='')
Freader=csv.reader(File2, delimiter=",")
for row in Freader:
    print(row)
File2.close()
```

CSV: READING AND WRITING WITH DICTREADER AND DICTWRITER

```
import csv
columnNames=['Firstname', 'Car']
Filename="csvFile2.csv"
CSV1=open(Filename,"w", newline='')
writer=csv.DictWriter(CSV1, fieldnames=columnNames)
writer.writeheader()
writer.writerow({"Firstname":"Sally", "Car":"Honda"})
writer.writerow({"Firstname":"Pam", "Car":"Buick"})
CSV1.close()
```

```
with open(Filename) as CSV2:

reader=csv.DictReader(CSV2)
    for line in reader:
        print(line["Car"])

CSV2.close()
```

The Output:

```
Honda
Buick
```

INTRODUCTION TO PYTHON 3 PANDAS

Python pandas (http://pandas.pydata.org/) is an open source library that offers excellent data structures, such as the pandas **dataframe**, as well as a number of analysis tools.

The pandas library is installed with Anaconda and can be used by including the following import statement:

```
import pandas as pd
```

PANDAS: SERIES

```
import numpy as np
import pandas as pd
#Create an array from 0 to 4
Mydata=np.arange(5)
#Note the index (row) value names
indexvalue=["C1", "C2", "C3", "C4", "C5"]
MySeries=pd.Series(Mydata, index=indexvalue)
print(MySeries)
```

The output:

```
C1    0
C2    1
C3    2
C4    3
C5    4
```

PANDAS: SERIES AND DICTIONARIES

```
MyDict={"Name":"Bob", "Age":29, "Degree":"MS"}
print(pd.Series(MyDict))
```

The Output:

```
Age        29
Degree     MS
Name       Bob
```

PANDAS: SERIES, NUMPY ARRAYS, FUNCTIONS

```
MyDict2={"Grade1":90.1, "Grade2":88.5,
"Grade3":93.6}

MySeries=pd.Series(MyDict2)

print(MySeries)

print("Grade 2 is: ", MySeries[1])

print("The mean of the grades: ",MySeries.mean())

print("Grades plus 5 points added is:\n", MySeries+5)

print("Grade 1 is: ", MySeries.get("Grade1"))
```

The Output:

```
Grade1    90.1
Grade2    88.5
Grade3    93.6

Grade 2 is:  88.5

The mean of the grades:
90.73
Grades plus 5 points added
is:
Grade1    95.1
Grade2    93.5
Grade3    98.6

Grade 1 is:  90.1
```

PANDAS: DATAFRAME

```
import pandas as pd
Gradebook={"Student1": pd.Series([89.3, 78.7, 92.2], index=['Grade1', 'Grade2', 'Grade3']),
                    "Student2": pd.Series([77.3, 83.4, 91.8], index=['Grade1', 'Grade2', 'Grade3']),
                    "Student3": pd.Series([97.1, 88.6, 98.5], index=['Grade1', 'Grade2', 'Grade3'])
          }
GradeBookDF=pd.DataFrame(Gradebook)
print(GradeBookDF)
```

OUTPUT: DATA FRAME

Output:

	Student1	Student2	Student3
Grade1	89.3	77.3	97.1
Grade2	78.7	83.4	88.6
Grade3	92.2	91.8	98.5

PANDAS DF: CREATE EMPTY DF AND ADD VALUE

#Create an empty dataframe

Gradebook2 = pd.DataFrame(Gradebook, index=['G1', 'G2', 'G3'], columns=['Bob Smith', 'Sandy Stern'])

print(Gradebook2)

#Fill in values

Gradebook2.**ix**["G1","Bob Smith"]=98.1

print(Gradebook2)

The Output:

	Bob Smith	Sandy Stern
G1	NaN	NaN
G2	NaN	NaN
G3	NaN	NaN

	Bob Smith	Sandy Stern
G1	**98.1**	NaN
G2	NaN	NaN
G3	NaN	NaN

PANDAS DF: ADD NEW COLUMN

```
#Create an empty dataframe
Gradebook2 = pd.DataFrame(Gradebook, index=['G1', 'G2', 'G3'], columns=['Bob Smith', 'Sandy Stern'])
print(Gradebook2)
#Create a new column
Gradebook2["NewColumn"]="NaN"
print(Gradebook2)
```

The Output

```
      Bob Smith   Sandy Stern    NewColumn
G1          NaN           NaN          NaN
G2          NaN           NaN          NaN
G3          NaN           NaN          NaN
```

PANDAS DF: ADD VALUES

```
import random
for i in range(len(Gradebook2.BobSmith)):
    Gradebook2.ix[i,"BobSmith"]=random.randint(50,100)
print(Gradebook2)
```

The Output:

	BobSmith	SandyStern	NewColumn
G1	**91**	NaN	NaN
G2	**56**	NaN	NaN
G3	**63**	NaN	NaN

PANDAS DF: CONVERT DICT AND ADD

MyDict=[{"Name":"Bob", "Age":29, "Degree":"MS"}, {"Name":"Rob", "Age":34, "Degree":"PhD"}]

DictDF=**pd.DataFrame.from_dict**(MyDict)

DictDF.**insert**(2, '**NewColumn**', [20007, 23604])

print(DictDF)

The Output:

```
     Age  Degree   NewColumn  Name
  0   29      MS       20007   Bob
  1   34     PhD       23604   Rob
```

PANDAS DF: DROPPING ROWS AND COLUMNS

```
MyDict=[{"Name":"Bob", "Age":29, "Degree":"MS"}, {"Name":"Rob", "Age":34, "Degree":"PhD"}]
DictDF=pd.DataFrame.from_dict(MyDict)
DictDF.insert(2, 'NewColumn', [20007, 23604])
#REMOVE the "Degree" column

DictDF=DictDF.drop("Degree", axis=1)
#axis=1 is the column, axis=0 is the row

#Remove the first row (row 0)
DictDF=DictDF.drop(0)
print(DictDF)
```

READ CSV TO PANDAS DF

```
csvFile="MyCSVFile3.csv"
File2=open(csvFile, "w", newline='')
Header=(["FirstName", "Lastname", "Grade1", "Grade2", "Grade3"])
Data1=(["John", "Smith", 90.3, 87.5, 77.2])
Data2=(["Bob", "Benson", 88.8, 77.7, 66.6])
Fwriter=csv.writer(File2)
Fwriter.writerow(Header)
Fwriter.writerow(Data1)
Fwriter.writerow(Data2)
File2.close()

csvDataFrame=pd.read_csv(csvFile)
print(csvDataFrame)
```

The Output:

```
  FirstName Lastname  Grade1  Grade2  Grade3
0      John    Smith    90.3    87.5    77.2
1       Bob   Benson    88.8    77.7    66.6
```

PANDAS DF: ADDING A NEW FEATURE PART 1

```
import pandas as pd
import csv
csvFile="MyCSVFile4.csv"
File2=open(csvFile, "w", newline='')
Header=(["FirstName", "Lastname", "Grade1", "Grade2", "Grade3"])
Data1=(["John", "Smith", 90.3, 97.5, 97.2])
Data2=(["Bob", "Benson", 88.8, 77.7, 66.6])
Data3=(["Sally", "Sue", 78.8, 71.7, 76.6])
Data4=(["Annie", "Apple", 58.8, 67.7, 69.6])
Fwriter=csv.writer(File2)
Fwriter.writerow(Header)
for i in [Data1, Data2, Data3, Data4]:
    Fwriter.writerow(i)
File2.close()
csvDataFrame=pd.read_csv(csvFile)
```

PANDAS DF: ADDING A NEW FEATURE PART 2

```
csvDataFrame["NewFeature"]="NaN"

for i in range(len(csvDataFrame.Grade1)):
    Avg=mean([csvDataFrame.ix[i,"Grade1"], csvDataFrame.ix[i,"Grade2"]])
    if Avg > 89.9:
        csvDataFrame.ix[i,"NewFeature"]="A"
    elif 79.9 < Avg < 90:
        csvDataFrame.ix[i,"NewFeature"]="B"
    elif 69.9 < Avg < 80:
        csvDataFrame.ix[i,"NewFeature"]="C"
    else:
        csvDataFrame.ix[i,"NewFeature"]="D"

print(csvDataFrame)
```

OUTPUT

	FirstName	Lastname	Grade1	Grade2	Grade3	**NewFeature**
0	John	Smith	90.3	97.5	97.2	A
1	Bob	Benson	88.8	77.7	66.6	C
2	Sally	Sue	78.8	81.7	86.6	B
3	Annie	Apple	58.8	67.7	69.6	D

PANDAS DF:
ADDING A NEW FEATURE PART 1

```
import pandas as pd
import csv
csvFile="MyCSVFile4.csv"
File2=open(csvFile, "w", newline='')
Header=(["FirstName", "Lastname", "Grade1", "Grade2", "Grade3"])
Data1=(["John", "Smith", 90.3, 97.5, 97.2])
Data2=(["Bob", "Benson", 88.8, 77.7, 66.6])
Data3=(["Sally", "Sue", 78.8, 71.7, 76.6])
Data4=(["Annie", "Apple", 58.8, 67.7, 69.6])
Fwriter=csv.writer(File2)
Fwriter.writerow(Header)
for i in [Data1, Data2, Data3, Data4]:
    Fwriter.writerow(i)
File2.close()
csvDataFrame=pd.read_csv(csvFile)
```

PANDAS DF:
ADDING A NEW FEATURE PART 2

```
csvDataFrame["NewFeature"]="NaN"
for i in range(len(csvDataFrame.Grade1)):
   Avg=mean([csvDataFrame.ix[i,"Grade1"], csvDataFrame.ix[i,"Grade2"]])
   if Avg > 89.9:
      csvDataFrame.ix[i,"NewFeature"]="A"
   elif 79.9 < Avg < 90:
      csvDataFrame.ix[i,"NewFeature"]="B"
   elif 69.9 < Avg < 80:
      csvDataFrame.ix[i,"NewFeature"]="C"
   else:
      csvDataFrame.ix[i,"NewFeature"]="D"
print(csvDataFrame)
```

OUTPUT

	FirstName	Lastname	Grade1	Grade2	Grade3	**NewFeature**
0	John	Smith	90.3	97.5	97.2	A
1	Bob	Benson	88.8	77.7	66.6	C
2	Sally	Sue	78.8	81.7	86.6	B
3	Annie	Apple	58.8	67.7	69.6	D