

# INT303 note

(Big Data Analytic)

## 1 Introduction [tutorial1.html](#)

### 1.1 Grades

Sequence	Method	Learning outcome s assessed	Duration	Timing	% of final mark	Resit
#1	Project 1: Data Scraping	All	See notice	S2	15%	NO
#2	Project 2: Big Data Competition	All	See notice	S2	15%	NO
#3	Written Exam	All	See notice	S2	70%	NO

### 1.2 Process

- Ask questions
  - Data Collection
  - Data Exploration
  - Data Modelling
  - Data Analysis
  - Visualization and Presentation of Results
- (Note: This process is by no means linear!)

## 2 Data [tutorial2.html](#)

### 2.1 Concepts

#### 2.1.1 What is Data?

- Collection of data **objects** and their **attributes**
- An attribute is a property or characteristic of an object
  - Examples: name, date of birth, height, occupation.
  - Attribute is also known as variable, field, characteristic, or feature
- For each object the attributes take some values.
- The collection of attribute-value pairs describes a specific object
  - Object is also known as record, point, case, sample, entity, or instance

#### 2.1.2 Relational data

- The term comes from DataBases, where we assume data is stored in a relational table with a fixed schema (fixed set of attributes)
- There are a lot of data in this form
- There are also a lot of data which do not fit well in this form
  - Sparse data: Many missing values
  - Not easy to define a fixed schema

#### 2.1.3 Types of Attributes

- **Numeric**
  - Examples: dates, temperature, time, length, value, count.
  - Discrete (counts) vs Continuous (temperature)
  - Special case: Binary/Boolean attributes (yes/no, exists/not exists)

Attributes

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Objects

Size (n): Number of objects

Dimensionality (d): Number of attributes

Density/Sparsity: Number of populated object-attribute pairs

## Categorical

- Examples: eye color, zip codes, strings, rankings (e.g, good, fair, bad), height in {tall, medium, short}
- Nominal (no order or comparison) vs Ordinal (order but not comparable)
- If data objects have the same fixed set of numeric attributes, then the data objects can be thought of as points/vectors in a multi-dimensional space, where each dimension represents a distinct attribute. Such data set can be represented by an  $n$ -by- $d$  data matrix:

	Temperature	Humidity	Pressure
O1	30	0.8	90
O2	32	0.5	80
O3	24	0.3	95

30	0.8	90
32	0.5	80
24	0.3	95

- Translate the table

Takes numerical values but it is actually categorical

ID Number	Zip Code	Age	Marital Status	Income	Income Bracket	Refund
1129842	45221	55	Single	250000	High	0
2342345	45223	25	Married	30000	Low	1
1234542	45221	45	Divorced	200000	High	0
1243535	45224	43	Single	150000	Medium	0

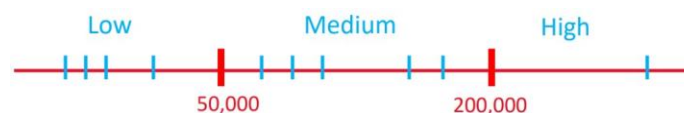
  

ID	Zip 45221	Zip 45223	Zip 45224	Age	Single	Married	Divorced	Income	Refund
1129842	1	0	0	55	0	0	0	250000	0
2342345	0	1	0	25	0	1	0	30000	1
1234542	1	0	0	45	0	0	1	200000	0
1243535	0	0	1	43	0	0	0	150000	0

## 2.2 Binning

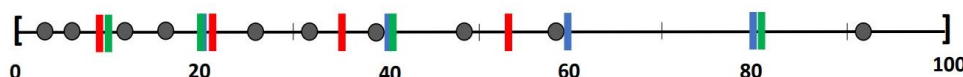
### 2.2.1 Concepts

- Idea: split the range of the domain of the numerical attribute into bins (intervals).
- Every bucket defines a categorical value



### 2.2.2 Bucketization

- Equi-width bins:** All bins have the same size
  - Example: split time into decades
  - Problem: some bins may be very sparse or empty
- Equi-size (depth) bins:** Select the bins so that they all contain the same number of elements
  - This splits data into quantiles: top-10%, second 10% etc
  - Some bins may be very small
- Equi-log bins:** log end – log start is constant
  - The size of the previous bin is a fraction of the current one
  - Better for skewed distributions
- Optimized bins:** Use a 1-dimensional clustering algorithm to create the bins



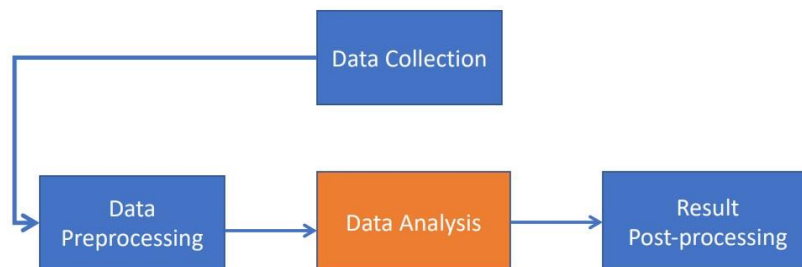
Blue: Equi-width [20,40,60,80]  
 Red: Equi-depth (2 points per bin)

Green: Equi-log (end/start = 2)

## 2.3 Types of data

- Set data: Each object is a set of values (with or without counts)
  - Sets can also be represented as binary vectors, or vectors of counts
- Dependent data:
  - Ordered sequences: Each object is an ordered sequence of values.
  - Spatial data: objects are fixed on specific geographic locations
  - Graph data: A collection of pairwise relationships
- **The data matrix:**
  - In almost all types of data we can find a way to transform the data into a matrix, where the rows correspond to different records, and the columns to numeric attributes

## 2.4 The data mining pipeline



### 2.4.1 Data Collection

For many supervised learning tasks (classification), you need **labelled data**, which will be used for training and testing

### 2.4.2 Data Preprocessing

- Reducing the data: Sampling, Dimensionality Reduction

- Simple Random Sampling
- **Sampling without replacement**
- **Sampling with replacement**

*E.g., we have 100 people, 51 are women  $P(W) = 0.51$ , 49 men  $P(M) = 0.49$ . If I pick two persons what is the probability  $P(W,W)$  that both are women?*

- *Sampling with replacement:  $P(W,W) = 0.512$*
- *Sampling without replacement:  $P(W,W) = 51/100 * 50/99$*

- **Stratified sampling**

- Split the data into several groups; then draw random samples from each group.

- Biased sampling

- When sampling temporal data, we want to increase the probability of sampling recent data: **recency bias**

- **Reservoir sampling**

- For the first k numbers, we retain them all.
  - For the i-th ( $i > k$ ) number, we retain the i-th number with a probability of  $k/i$
  - Replace with any one of the previously selected k numbers with a probability of  $1/k$ .
- Every item has probability  $1/N$  to be selected after N items have been read.

- Data cleaning: deal with missing or inconsistent information

- Deal with missing values:
  - Ignore the data
  - Replace with random value
  - Replace with the mean
  - Replace with nearest neighbor value
  - Replace with cluster mean
  - Infer the value
- Deal with outliers:

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	10000K	Yes
6	No	NULL	60K	No
7	Yes	Divorced	220K	NULL
8	No	Single	85K	Yes
9	No	Married	90K	No
9	No	Single	90K	No

- Remove them
- Try to correct them using common sense
- Transform the data
- When using the data, we should be careful of cases where our results are too good to be true, or too bad to be true
- Feature extraction and selection: create a useful representation of the data by extracting useful features

### 2.4.3 Data Analysis

- Sample mean
- Sample median
- Sample range
- Sample variance
- Sample standard deviation

### 2.4.4 Result Post-processing

...

## 3 Data Grammar [tutorial3.html](https://www.tc-tea.com/tutorial3.html)

### 3.1 Data Normalization

#### 3.1.1 Column Normalization

In this data, different attributes take very different range of values. For distance/similarity the small values will disappear:

Temperature	Humidity	Pressure
30	0.8	90
32	0.5	80
24	0.3	95

So, divide (the values of a column) by the maximum value for each attribute:

Temperature	Humidity	Pressure
0.9375	1	0.9473
1	0.625	0.8421
0.75	0.375	1

(Brings everything in the [0,1] range, maximum is 1)

#### 3.1.2 Row Normalization

Some data which are similar:

	Word 1	Word 2	Word 3
Doc 1	28	50	22
Doc 2	12	25	13

Divide by the sum of values for each document (row in the matrix):

	Word 1	Word 2	Word 3
Doc 1	0.28	0.5	0.22
Doc 2	0.24	0.5	0.26

(New value = old value /  $\Sigma$  old values in the row)

### Z-score:

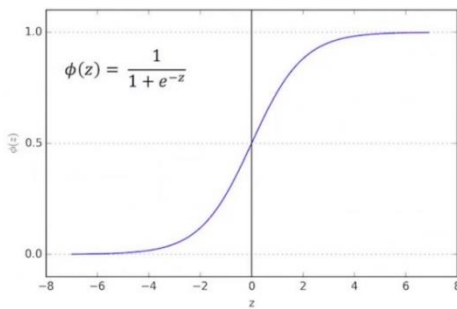
$$\text{mean}(x) = \frac{1}{N} \sum_{j=1}^N x_j$$

$$\text{std}(x) = \sqrt{\frac{\sum_{j=1}^N (x_j - \text{mean}(x))^2}{N}}$$

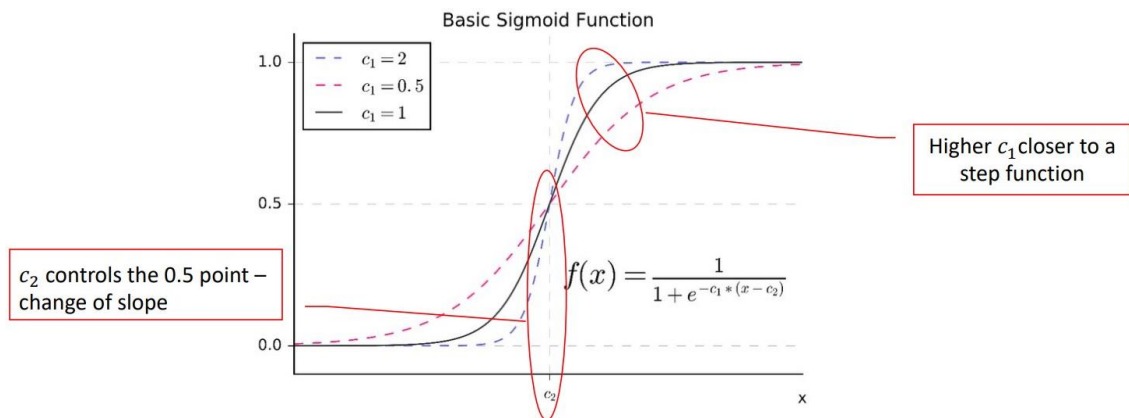
$$z_i = \frac{x_i - \text{mean}(x)}{\text{std}(x)}$$

### Logistic function:

$$\phi(x) = \frac{1}{1 + e^{-x}}$$



- Maps reals to the  $[0,1]$  range
- Mimics the step function
- In the class of sigmoid functions



### Softmax function:

If we want to transform the scores into probabilities that sum to one, but we capture the single selection of the user.

$$\frac{e^{x_i}}{\sum_i e^{x_i}}$$

	Restaurant 1	Restaurant 2	Restaurant 3
User 1	5	2	3
User 2	1	3	4

→

	Restaurant 1	Restaurant 2	Restaurant 3
User 1	0.72	0.10	0.18
User 2	0.07	0.31	0.62

### Logarithm function:

- Sometimes a data row/column may have a very wide range of values. Normalizing in this case will obliterate small values.
- We can bring the values to the same scale by applying a logarithm to the column values.

User id	Income	Log Income
1	6000	3.778151
2	6500	3.812913
3	7000	3.845098
4	4000	3.60206
5	8000	3.90309
6	9000	3.954243
7	18000	4.255273
8	36000	4.556303
9	600000	5.778151
10	1000000	6

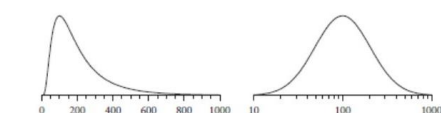


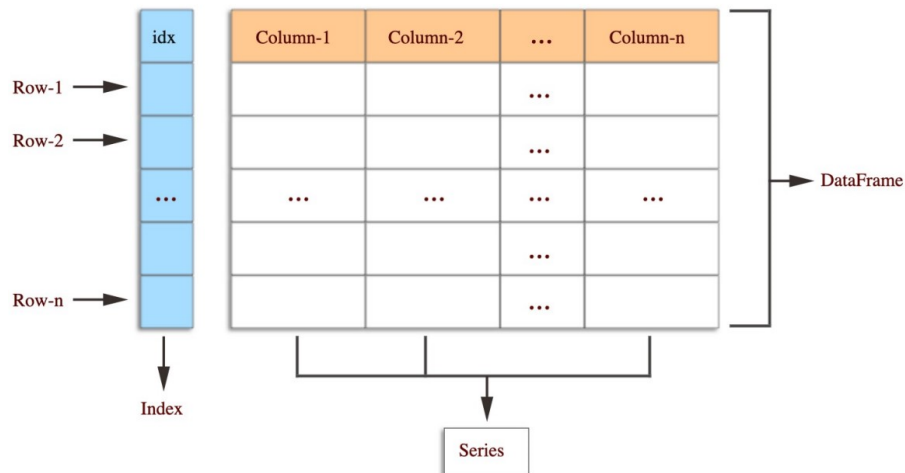
Figure 11.2. Lognormal distribution.  
(Left) Lognormal distribution. The distribution appears Gaussian when plotted on a logarithmic axis (right) or when all values are transformed to their logarithm.

## 3.2 Exploratory Data Analysis (EDA)

1. Store data in data structure(s) that will be convenient for exploring/processing (Memory is fast. Storage is slow)
2. Clean/format the data so that:
  - Each row represents a single object/observation/entry
  - Each column represents an attribute/property/feature of that entry
  - Values are numeric whenever possible
  - Columns contain atomic properties that cannot be further decomposed\*
3. Explore global properties: use histograms, scatter plots, and aggregation functions to summarize the data
4. Explore group properties: group like-items together to compare subsets of the data

### 3.2.1 Pandas (Python package) [User Guide — pandas 2.1.1 documentation \(pydata.org\)](https://pandas.pydata.org/docs/user_guide/1.html)

#### Pandas Data structure



Import pandas library code:

```
import pandas as pd
dataframe = pd.read_csv("yourfile.csv")
```

Common Pandas functions:

- `head()` – first N observations
- `tail()` – last N observations
- `columns()` – names of the columns
- `describe()` – statistics of the quantitative data
- `dtypes()` – the data types of the columns
- `df["column_name"]`
- `Df.column_name`
- `.max()`, `.min()`, `.idxmax()`, `.idxmin()`
- `<dataframe> <conditional statement>`
- `.loc[]` – label-based accessing
- `.iloc[]` – index-based accessing
- `.sort_values()`
- `.isnull()`, `.notnull()`
- `groupby()`, `.get_groups()`
- `.merge()`
- `.concat()`
- `.aggregate()`
- `.append()`

```
# input
state = np.random.RandomState(100)
ser = pd.Series(state.normal(10, 5, 25))
```

```
# using pandas
ser.describe()
```

```
count    25.000000
mean      10.435437
std        4.253118
min        1.251173
25%        7.709865
50%       10.922593
75%       13.363604
max       18.094908
dtype: float64
```



## Rebuild Dataset:

```
data.isna().sum()
```

```
+
```

```
DataFrame.fillna(value=None, method=None, axis=None, inplace=False, limit=None,
downcast=None)
```

```
+
```

```
DataFrame.drop_duplicates(subset=None, keep='first', inplace=False,
ignore_index=False)
```

## GroupBy Dataset:

```
In [28]: dfcwc.groupby("state").sum()
```

```
Out[28]:
```

	zip	amount	candidate_id
state			
AK	2985459621	1210.00	111
AR	864790	14200.00	192
AZ	860011121	120.00	37
CA	14736360720	-5013.73	600
CO	2405477834	-5823.00	111
CT	68901376	2300.00	35
DC	800341853	-1549.91	102
FL	8970626520	-4050.00	803

## Merge Dataset:

```
DataFrame.merge(right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'), copy=True,
indicator=False, validate=None)
```

```
In [40]: cols_wanted=['last_name_x', 'first_name_x', 'candidate_id', 'id', 'last_name_y']
dfcwc.merge(dfccand, left_on="candidate_id", right_on="id")[cols_wanted]
```

```
Out[40]:
```

	last_name_x	first_name_x	candidate_id	id	last_name_y
0	Agee	Steven	16	16	Huckabee
1	Akin	Charles	16	16	Huckabee
2	Akin	Mike	16	16	Huckabee
3	Akin	Rebecca	16	16	Huckabee
4	Aldridge	Brittini	16	16	Huckabee

## 3.3 Data Mining

### 3.3.1 Similarity

- Numerical measure of how alike two data objects are.
  - A function that maps pairs of objects to real values
  - Higher when objects are more alike.
- Often falls in the range  $[0,1]$ , sometimes in  $[-1,1]$
- Desirable properties for similarity
  - $s(p, q) = 1$  (or maximum similarity) only if  $p = q$ . (Identity)
  - $s(p, q) = s(q, p)$  for all  $p$  and  $q$ . (Symmetry)
  - $s(p, q) = 0.5$  and  $s(p, x) = 0.5$ , then  $s(q, x) = 0.5$
- The **Jaccard similarity (Jaccard coefficient)** of two sets  $S_1, S_2$  is the size of their intersection divided by the size of their union.

$$JSim(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

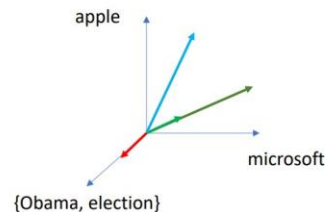
- Extreme behavior:
  - $\text{JSim}(X, Y) = 1$ , iff  $X = Y$
  - $\text{JSim}(X, Y) = 0$ , iff  $X, Y$  have no elements in common
- JSim is symmetric
- Similarity between vectors:

document	Apple	Microsoft	Obama	Election
D1	10	20	0	0
D2	30	60	0	0
D3	60	30	0	0
D4	0	0	10	20

Documents D1, D2 are in the "same direction"

Document D3 is on the same plane as D1, D2

Document D4 is orthogonal to the rest



### • Cosine Similarity

- $\text{Sim}(X, Y) = \cos(X, Y)$
- If the vectors are aligned (correlated) angle is zero degrees and  $\cos(X, Y) = 1$
- If the vectors are orthogonal (no common coordinates) angle is 90 degrees and  $\cos(X, Y) = 0$
- If  $x$  and  $y$  are two vectors, then:

$$\cos(x, y) = \frac{x \cdot y}{\|x\| \|y\|}$$

Example:

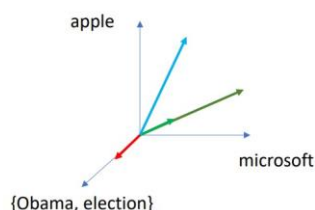
$$\begin{aligned}
 x &= 3 \ 2 \ 0 \ 5 \ 0 \ 0 \ 2 \ 0 \ 0 \\
 y &= 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 2 \\
 x \cdot y &= 3 \cdot 1 + 2 \cdot 0 + 0 \cdot 0 + 5 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 + 2 \cdot 1 + 0 \cdot 0 + 0 \cdot 0 = 12 \\
 \|x\| &= \sqrt{3^2 + 2^2 + 0^2 + 5^2 + 0^2 + 0^2 + 2^2 + 0^2 + 0^2} = \sqrt{42} = 6.481 \\
 \|y\| &= \sqrt{1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 2^2} = \sqrt{6} = 2.245 \\
 \cos(x, y) &= 0.315
 \end{aligned}$$

document	Apple	Microsoft	Obama	Election
D1	10	20	0	0
D2	30	60	0	0
D3	60	30	0	0
D4	0	0	10	20

$$\cos(D1, D2) = 1$$

$$\cos(D3, D1) = \cos(D3, D2) = 4/5$$

$$\cos(D4, D1) = \cos(D4, D2) = \cos(D4, D3) = 0$$



### • Correlation Coefficient

- If we have observations (vectors)  $X = (x_1, \dots, x_n)$  and  $Y = (y_1, \dots, y_n)$  is defined as:

$$\text{CorrCoeff}(X, Y) = \frac{\sum_i (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_i (x_i - \mu_X)^2} \sqrt{\sum_i (y_i - \mu_Y)^2}}$$

- The correlation coefficient takes values in  $[-1, 1]$   
 (-1 negative correlation, +1 positive correlation, 0 no correlation)



Normalized vectors

document	Apple	Microsoft	Obama	Election
D1	-5	+5	0	0
D2	-15	+15	0	0
D3	+15	-15	0	0
D4	0	0	-5	+5

$$\text{CorrCoeff}(X, Y) = \frac{\sum_i (x_i - \mu_X)(y_i - \mu_Y)}{\sqrt{\sum_i (x_i - \mu_X)^2} \sqrt{\sum_i (y_i - \mu_Y)^2}}$$

$$\text{CorrCoeff}(\text{D1}, \text{D2}) = 1$$

$$\text{CorrCoeff}(\text{D1}, \text{D3}) = \text{CorrCoeff}(\text{D2}, \text{D3}) = -1$$

$$\text{CorrCoeff}(\text{D1}, \text{D4}) = \text{CorrCoeff}(\text{D2}, \text{D4}) = \text{CorrCoeff}(\text{D3}, \text{D4}) = 0$$

### 3.3.2 Distance

- Distances for real vectors:

- Vectors  $x = (x_1, \dots, x_d)$  and  $y = (y_1, \dots, y_d)$

- $L_p$ -norms or **Minkowski** distance:

$$L_p(x, y) = [|x_1 - y_1|^p + \dots + |x_d - y_d|^p]^{1/p}$$

- $L_2$ -norm: **Euclidean** distance:

$$L_2(x, y) = \sqrt{|x_1 - y_1|^2 + \dots + |x_d - y_d|^2}$$

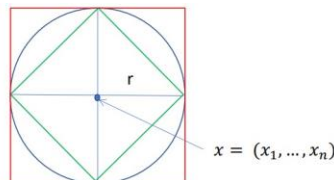
- $L_1$ -norm: **Manhattan** distance:

$$L_1(x, y) = |x_1 - y_1| + \dots + |x_d - y_d|$$

- $L_\infty$ -norm:

$$L_\infty(x, y) = \max\{|x_1 - y_1|, \dots, |x_d - y_d|\}$$

- The limit of  $L_p$  as  $p$  goes to infinity.



**Green:** All points  $y$  at distance  $L_1(x, y) = r$  from point  $x$

**Blue:** All points  $y$  at distance  $L_2(x, y) = r$  from point  $x$

**Red:** All points  $y$  at distance  $L_\infty(x, y) = r$  from point  $x$

- Similarities into distances:

- Jaccard distance:**  $JDist(X, Y) = 1 - JSim(X, Y)$
- Cosine distance:**  $Dist(X, Y) = 1 - \cos(X, Y)$

- Hamming distance:

- Hamming distance is the number of positions in which bit-vectors differ.

- Example:

- $p_1 = 10101$
- $p_2 = 10011$
- $d(p_1, p_2) = 2$  because the bit-vectors differ in the 3<sup>rd</sup> and 4<sup>th</sup> positions.
- The  $L_1$  norm for the binary vectors

- Variational distance:

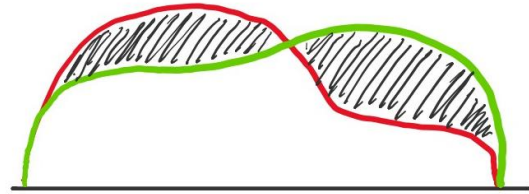
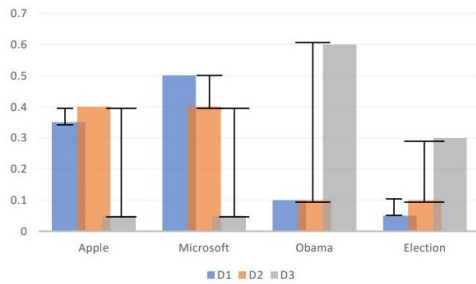
- Variational distance is the  $L_1$  distance between the distribution vectors

document	Apple	Microsoft	Obama	Election
D1	0.35	0.5	0.1	0.05
D2	0.4	0.4	0.1	0.1
D3	0.05	0.05	0.6	0.3

$$\text{Dist}(D1, D2) = 0.05 + 0.1 + 0.05 = 0.2$$

$$\text{Dist}(D2, D3) = 0.35 + 0.35 + 0.5 + 0.2 = 1.4$$

$$\text{Dist}(D1, D3) = 0.3 + 0.45 + 0.5 + 0.25 = 1.5$$



#### • Information theoretic distances:

- **KL-divergence (Kullback-Leibler)** for distributions P,Q:

$$D_{KL}(P||Q) = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

- KL-divergence is **asymmetric**. We can make it symmetric by taking the average of both sides:

$$\frac{1}{2} (D_{KL}(P||Q) + D_{KL}(Q||P))$$

- **JS-divergence (Jensen-Shannon)**:

$$JS(P, Q) = \frac{1}{2} D_{KL}(P||M) + \frac{1}{2} D_{KL}(Q||M)$$

$$M = \frac{1}{2} (P + Q)$$

#### • Ranking distances:

- The input in this case is two rankings/orderings of the same n items.

For example:

$$R_1 = \langle x, y, z, w \rangle$$

$$R_2 = \langle y, w, z, x \rangle$$

	x	y	z	w
R <sub>1</sub>	1	2	3	4
R <sub>2</sub>	4	1	3	2

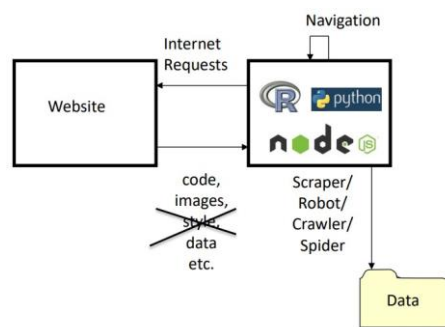
- Spearman rank distance: L<sub>1</sub> distance between the ranks

$$SR(R_1, R_2) = |1 - 4| + |2 - 1| + |3 - 3| + |4 - 2| = 6$$

## 4 Web Scraping [tutorial4.html](https://www.tc-tea.com/tutorial4.html)

### 4.1 Role

Instead of people repeatedly querying the browser for website data, use tools like python to quickly get data directly from the website.



### HTTP STATUS CODES:

#### **200 OK:**

Means that the server did whatever the client wanted it to, and all is well.

#### **400: Bad request**

The request sent by the client didn't have the correct syntax.

**401: Unauthorized**

Means that the client is not allowed to access the resource. This may change if the client retries with an authorization header.

**403: Forbidden**

The client is not allowed to access the resource and authorization will not help.

**404: Not found**

It means that the server has not heard of the resource and has no further clues as to what the client should do about it. In other words: dead link.

**500: Internal server error**

Something went wrong inside the server.

**501: Not implemented**

The request method is not supported by the server.

**4.2 Python data scraping****4.2.1 Legal**



Respect the Robots Exclusion Protocol also known as the `[robots.txt](#)` (add this to the website url tail to see which element you can scrap)

e.g.: <http://google.com/robots.txt>

**4.2.2 Step 1: Inspect Your Data Source**

Example website for exercise: [Fake Python \(realpython.github.io\)](https://realpython.github.io/fake-jobs/) <https://realpython.github.io/fake-jobs/>

Developer tool:

- Mac: 
- Windows/Linux: 

**4.2.3 Step 2: Scrape Html Content From A Page**

Python

```
import requests

URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)

print(page.text)
```

**4.2.4 Step 3: Parse Html Code With BeautifulSoup**

Python

```
import requests
from bs4 import BeautifulSoup

URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")
```

- Find elements by ids:

HTML

```
<div id="ResultsContainer">
  <!-- all the job listings -->
</div>
```

- BeautifulSoup allows you to find that specific HTML element by its ID:

Python

```
results = soup.find(id="ResultsContainer")
```

**find VS findAll**

```
import bs4
## get bs4 object
soup = bs4.BeautifulSoup(source)
## all a tags
soup.findAll('a')
## first a
soup.find('a')
## get all links in the page
link_list = [l.get('href') for l in soup.findAll('a')]
```

- Find elements by HTML class name:

Python

```
job_elements = results.find_all("div", class_="card-content")
```

Python

```
for job_element in job_elements:
    print(job_element, end="\n"*2)
```

Python

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element)
    print(company_element)
    print(location_element)
    print()
```

HTML

```
<h2 class="title is-5">Senior Python Developer</h2>
<h3 class="subtitle is-6 company">Payne, Roberts and Davis</h3>
<p class="location">Stewartbury, AA</p>
```

- What if we just want the text content of the extract data? Use the `.text` script and you can `.strip()` the superfluous whitespace:

Python

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    print()
```

- Find elements by class name and text content:

Python

```
python_jobs = results.find_all("h2", string="Python")
```

- Use a lambda function to filter the data by substring "python" and `<h2>` tag:

Python

```
python_jobs = results.find_all(
    "h2", string=lambda text: "python" in text.lower()
)
```

- HTML is a tree

```

• tree = bs4.BeautifulSoup(source)

• ## get html root node
• root_node = tree.html
• ## get head from root using contents
• head = root_node.contents[0]
• ## get body from root
• body = root_node.contents[1]
• ## could directly access body
• tree.body

```

- Access parent elements:

Python

```

python_jobs = results.find_all(
    "h2", string=lambda text: "python" in text.lower()
)

python_job_elements = [
    h2_element.parent.parent.parent for h2_element in python_jobs
]

```

- Extract attributes from HTML elements

Python

```

for job_element in python_job_elements:
    # -- snip --
    links = job_element.find_all("a")
    for link in links:
        link_url = link["href"]
        print(f"Apply here: {link_url}\n")

```

(Start by fetching all the <a> elements in a job card. Then, extract the value of their href attributes using square-bracket notation)

## 4.3 Training website

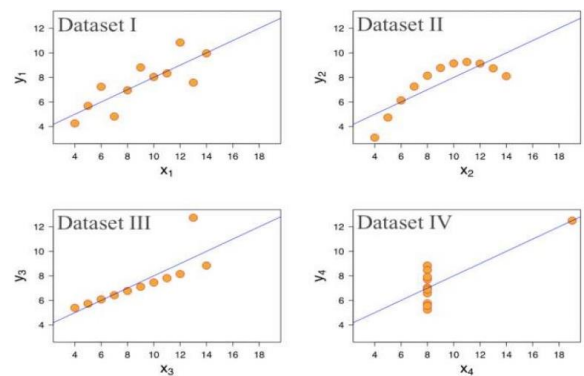
Any API: <https://any-api.com/>

## 5 Data Visualization [tutorial5.html](#)

### 5.1 Visualization motivation

Although some datasets have same statistic value, but their feature are different:

	Dataset I		Dataset II		Dataset III		Dataset IV	
	x	y	x	y	x	y	x	y
	10	8.04	10	9.14	10	7.46	8	6.58
	8	6.95	8	8.14	8	6.77	8	5.76
	13	7.58	13	8.74	13	12.74	8	7.71
	9	8.81	9	8.77	9	7.11	8	8.84
	11	8.33	11	9.26	11	7.81	8	8.47
	14	9.96	14	8.1	14	8.84	8	7.04
	6	7.24	6	6.13	6	6.08	8	5.25
	4	4.26	4	3.1	4	5.39	19	12.5
	12	10.84	12	9.13	12	8.15	8	5.56
	7	4.82	7	7.26	7	6.42	8	7.91
	5	5.68	5	4.74	5	5.73	8	6.89
Sum:	99.00	82.51	99.00	82.51	99.00	82.51	99.00	82.51
Avg:	9.00	7.50	9.00	7.50	9.00	7.50	9.00	7.50
Std:	3.32	2.03	3.32	2.03	3.32	2.03	3.32	2.03



- Identify hidden patterns and trends
- Formulate/test hypotheses
- Communicate any modeling results
  - Present information and ideas succinctly
  - Provide evidence and support
  - Influence and persuade
- Determine the next step in analysis/modeling

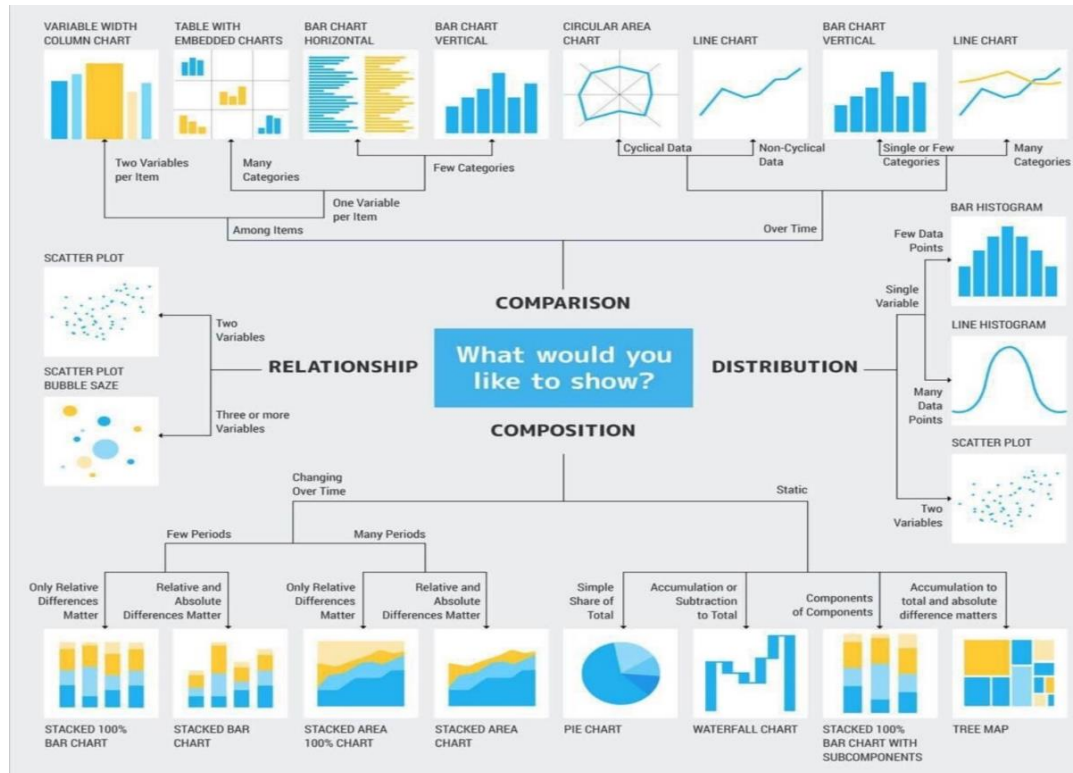
### 5.2 Principle of Visualization

1. Maximize data to ink ratio: show the data
2. Don't lie with scale: minimize

3. Minimize chart-junk: show data variation, not designvariation

4. Clear, detailed and thorough labeling

### 5.3 Types of Visualization



- Distribution: how a variable or variables in the dataset distribute over a range of possible values.
- Relationship: how the values of multiple variables in the dataset relate
- Composition: how a part of your data compares to the whole.
- Comparison: how trends in multiple variable or datasets compare

Some examples: [treevis.net](https://treevis.net)

## 6 Infrastructure that supports Big Data processing

### 6.1 Large-scale computing

**Issue 1:** Copying data over a network takes time.

Idea:

- Bring computation to data
- Store files multiple times for reliability

Spark/Hadoop address these problems

- Storage Infrastructure – File system
- Google: GFS. Hadoop: HDFS

Programming model: MapReduce

**Issue 2:** If nodes fail, how to store data persistently?

Idea:

- Distributed File System
- Provides global file namespace

Typical usage pattern:

- Huge files (100s of GB to TB)
- Data is rarely updated in place
- Reads and appends are common

### 6.2 Distributed file system

Chunk servers:

- File is split into contiguous chunks



- Typically, each chunk is 16-64MB
- Each chunk replicated (usually 2x or 3x)
- Try to keep replicas in different racks

Master node:

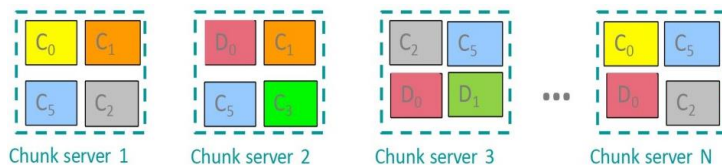
- a.k.a. Name Node in Hadoop's HDFS
- Stores metadata about where files are stored
- Might be replicated

Client library for file access:

- Talks to master to find chunk servers
- Connects directly to chunk servers to access data

Reliable distributed file system:

- Data kept in "chunks" spread across machines
- Each chunk replicated on different machines
- Seamless recovery from disk or machine failure



### 6.3 MapReduce: Distributed computing programming model

MapReduce is a style of programming designed for:

- Easy parallel programming
- Invisible management of hardware and software failures
- Easy management of very-large-scale data

3-steps of MapReduce:

1. Map

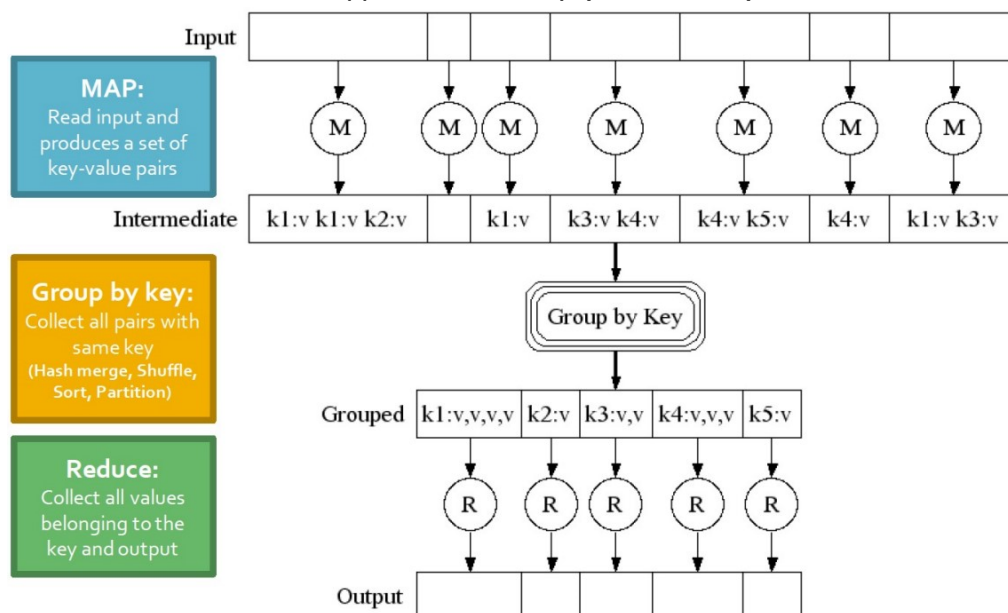
- Apply a user-written Map function to each input element
  - Mapper applies the Map function to a single element
  - Many mappers grouped in a Map task(the unit of parallelism)
- The output of the Map function is a set of 0, 1, or more key-value pairs.

2. Group by key: Sort and shuffle

- System sorts all the key-value pairs by key, and outputs key-(list of values) pairs

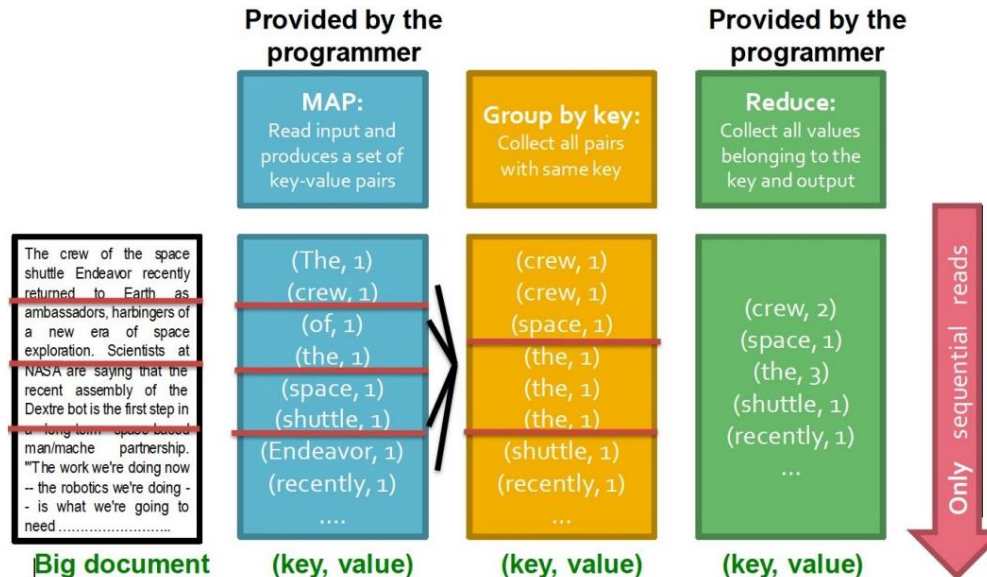
3. Reduce

- User-written Reduce functions applied to each key-(list of values)



The inputs to reducers are grouped by key

## EXAMPLE: WORD COUNTING



## 6.4 Spark: Extends MapReduce

## Hadoop

- Disk-based computation where the results of each step are written to disk.

RDD [Difference Between Hadoop and Spark - GeeksforGeeks](http://Difference Between Hadoop and Spark - GeeksforGeeks)

- Distributed computing in memory
- Greatly improving the speed of data processing

Consider a dataset of text documents that you want to process to count the frequency of each word across all documents. In a MapReduce framework, this is typically done in two steps:

1. Mapping Phase:
  1. Input: Text documents
  2. Output: Key-value pairs where the key is a word and the value is 1 (indicating one occurrence of the word).
2. Reducing Phase:
  1. Input: Key-value pairs from the mapping phase
  2. Output: Key-value pairs where the key is a word and the value is the total count of that word across all documents.

In **Apache Spark**, you could create an RDD from the text documents and then use Spark operations to achieve the same word count task:

```
wordsList = ['cat', 'elephant', 'rat', 'rat', 'cat']
wordsRDD = sc.parallelize(wordsList, 3)
wordCountsCollected = (wordsRDD
    .map(lambda w: (w, 1))
    .reduceByKey(lambda x,y: x+y)
    .collect())
[('rat', 2), ('elephant', 1), ('cat', 2)]
```

FILE RDD —flatMap—>[list of words]—map—>[(word,1),...]  
—reduceByKey (map combiner + reduce transformation) —>[(word, count),...] --save Action-->File

```
text_file = spark.textFile("hdfs://...")
counts = (text_file.flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a + b)
)
counts.saveAsTextFile("hdfs://...")
```

