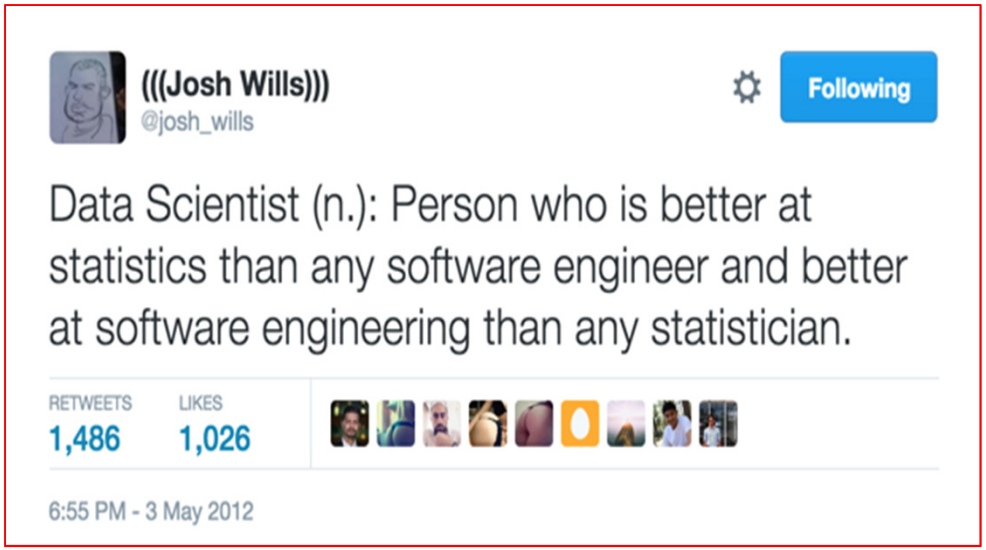
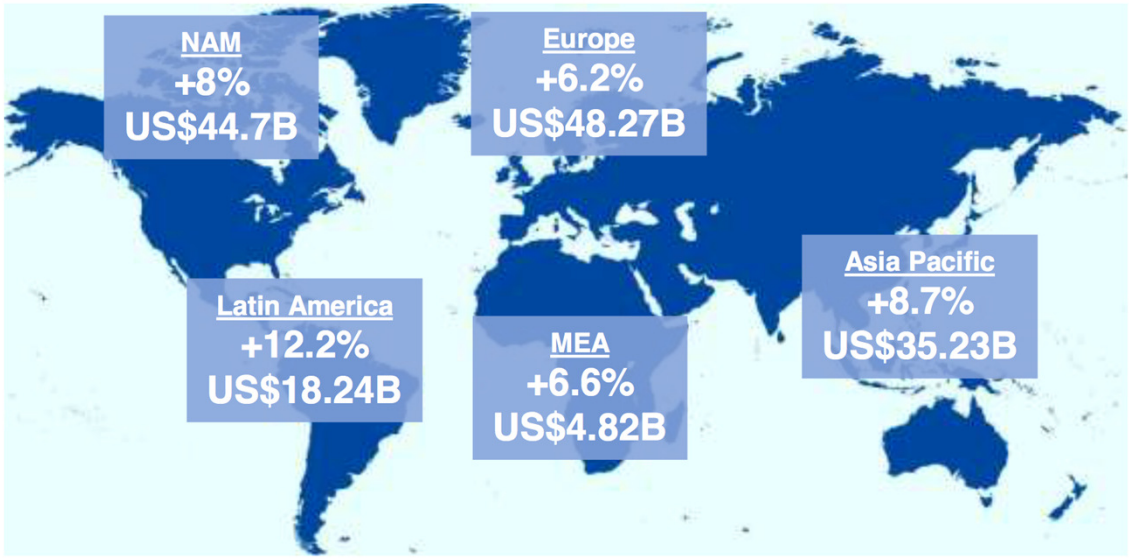
Data Preparation and Preprocessing

Recap from Week 1

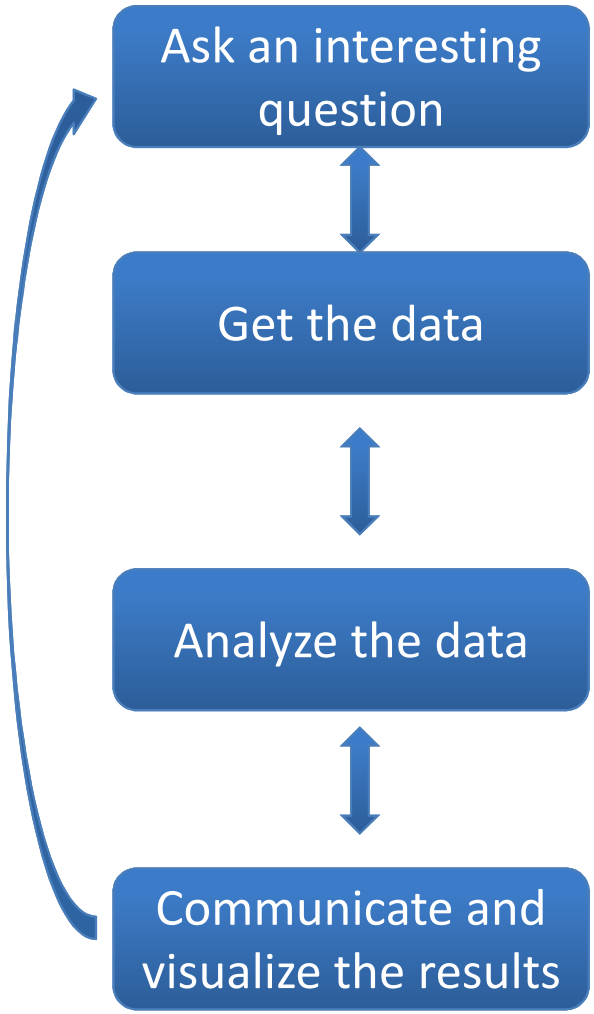


Josh Wills, Data Scientist at Slack



Recap from Week 1

Data Science pipeline

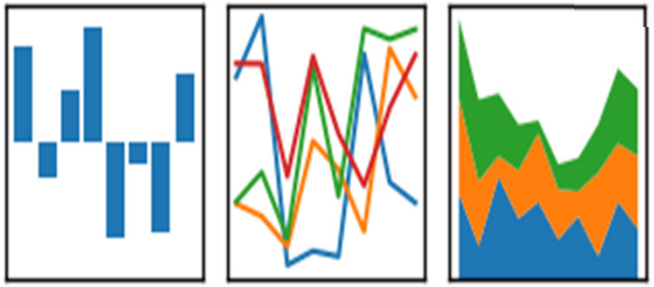


Python libraries for Data Analytics

https://seaborn.pydata.org/

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



- Numpy: great for handling numbers, vectors, matrices
- Scipy: great for numerical optimizations
- Pandas: great for handling tabular/relational data
- Scikit Learn: great for data analytics techniques

3803ICT course structure

W1. Introduction to Data Analytics

Data Preparation and Preprocessing

W2. Data Preparation and Preprocessing

Data Analysis and Interpretation

W3. Exploratory Data Analytics

W4. Statistical Data Analytics

W5. Predictive Data Analytics

Visualization

W6. Data Visualization

Analysis of special types of data

W7. Time Series

W8. Textual Data

W9. Graph Data

Analysis with big data infrastructure

W10. Cloud Computing

W11. Spark

W12. Revision

Learning Outcomes

- At the end of this lecture students will be able to know:
  - Relational data storage with Pandas
  - Different storage models
  - Data normalisation
  - Data cleaning

Outline

1. Data Storage
2. Data Normalisation
3. Data Cleaning

1. Data storage: Pandas library

- Pandas is an open source library
  - Process relational data in memory
  - Support SQL-like query
- Rich relation data tool built on top of NumPy
  - Excellent performance
  - Easy-to-use, highly consistent API
- A foundation for data analysis in Python
  - It also has built-in visualization features
  - It can work with data from a wide variety of sources
  - Takes data preparation and preprocessing to the next level

https://pandas.pydata.org/

Pandas vs. SQL implementations

Advantages:

+ Pandas is lightweight and fast.

+ Natively Python, i.e., full SQL expressiveness plus the expressiveness of Python, especially for function evaluation.

+ Integration with plotting functions like Matplotlib.

Disadvantages:

- Tables must fit into memory.

- No post-load indexing functionality: indices are built when a table is created.

- No transactions, journaling, etc.

- Large, complex joins are slower.

Pandas: features

- Series
- DataFrames → Tables
- Operations
  - GroupBy
  - Merging, Joining, and Concatening
- Data Input and Output

Pandas: Series

- Series: a named, ordered dictionary
  - Easy data search and retrieval.

- One-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.).
  - The keys of the dictionary are the indexes
  - Built on NumPy's **ndarray**
  - Values can be any NumPy data type object

- Why use Series?
  - Better for single-valued entries of the same type (Integer, String, etc.).
  - Easy and efficient search using index.

Pandas: Series

- Create a series:

```
ser1 = pd.Series([1,2,3,4],index = ['USA', 'Germany','USSR', 'Japan'])
```

```
ser1
```

```
USA         1
Germany     2
USSR        3
Japan       4
dtype: int64
```

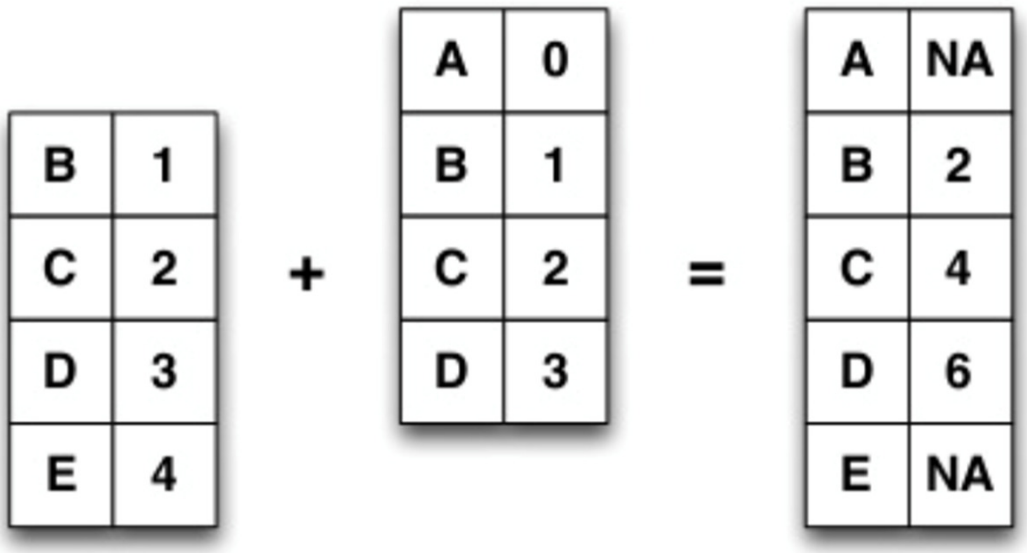
- Access an element → Search by the element's Index.
- Easy to search for elements in big series.

```
ser1['USA']
```

```
1
```

Pandas: Series

- Operations are then done based on the index.



B	1
C	2
D	3
E	4

+

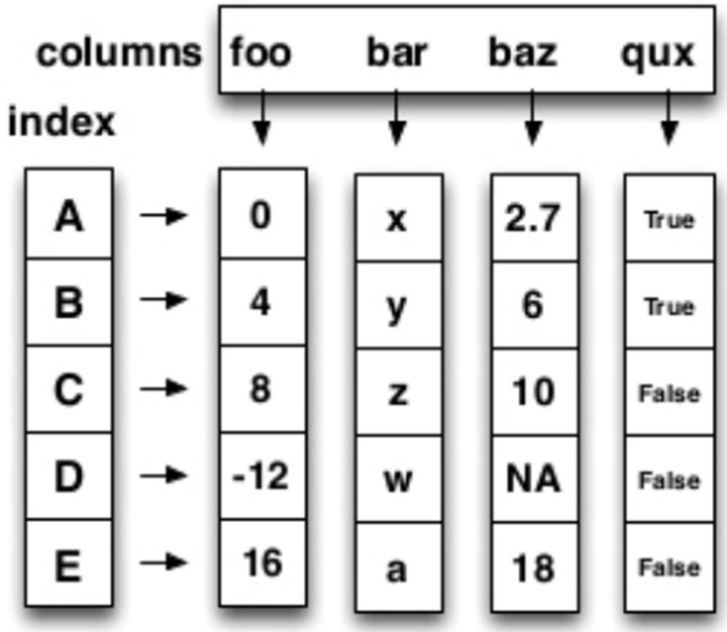
A	0
B	1
C	2
D	3

=

A	NA
B	2
C	4
D	6
E	NA

Pandas: DataFrame

- DataFrame: a table with named columns (like in the relational model)
  - Represented as a dictionary
    - columnName -> series
  - Each Series object represents a column
  - Each column can have a different type
  - Row and column indices
  - Size mutable: insert and delete columns



- Why Use DataFrames?
  - better for series with multiples attributes of different types.
  - Easy and efficient search elements by index.

Pandas: DataFrame

- Create a data frame:

```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.split())
```

```
df
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

- Selection and Indexing: you can select columns by their names.
- Columns of DataFrame are just Series.

```
df['W']
```

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

```
type(df['W'])
```

```
pandas.core.series.Series
```

```
# Pass a list of column names
df[['W','Z']]
```

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

Pandas: DataFrame

- Conditional Selection: An important feature of pandas is conditional selection → The result is also a DataFrame.
  - Filter rows based on a given condition.
  - Useful if you want to do analytics on a specific set of entries.

```
df>0
```

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

Condition table

```
df[df>0]
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

Filter every cell

```
df[df['W']>0]
```

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

Filter a column

Pandas: DataFrame

- Conditional Selection: for two conditions you can use | and & with parenthesis.
  - Filtering rows based on multiple conditions.

```
df[(df['W']>0) & (df['Y'] > 1)]
```

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

Pandas: DataFrame Multi-index

- Multi-index and index hierarchy: create a multi-index DataFrame.
  - Data can be organized and searched using any of the index levels.

```
# Index Levels
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

```
hier_index
```

```
MultiIndex(levels=[['G1', 'G2'], [1, 2, 3]],
           labels=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])
```

```
df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df
```

		A	B
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

Pandas: Missing Data

- Data is not perfect → There may be missing values
- Example:

```
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
```

```
df
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

Pandas: Removing missing data

- Drop all rows with missing values:

```
df.dropna()
```

	A	B	C
0	1.0	5.0	1

- Drop all columns with missing values:

```
df.dropna(axis=1)
```

	C
0	1
1	2
2	3

- Drop rows with 2 or more missing values

```
df.dropna(thresh=2)
```

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

Pandas: Missing data

- Fill in the missing values
  - Basic step of data completion.

```
df.fillna(value='FILL VALUE')
```

	A	B	C
0	1	5	1
1	2	FILL VALUE	2
2	FILL VALUE	FILL VALUE	3

With arbitrary value

```
df['A'].fillna(value=df['A'].mean())
```

```
0    1.0
1    2.0
2    1.5
Name: A, dtype: float64
```

With calculated value

Pandas: Find Null Values

- Example:

```
df = pd.DataFrame({'col1':[1,2,3,np.nan],
                   'col2':[np.nan,555,666,444],
                   'col3':['abc','def','ghi','xyz']})
df.head()
```

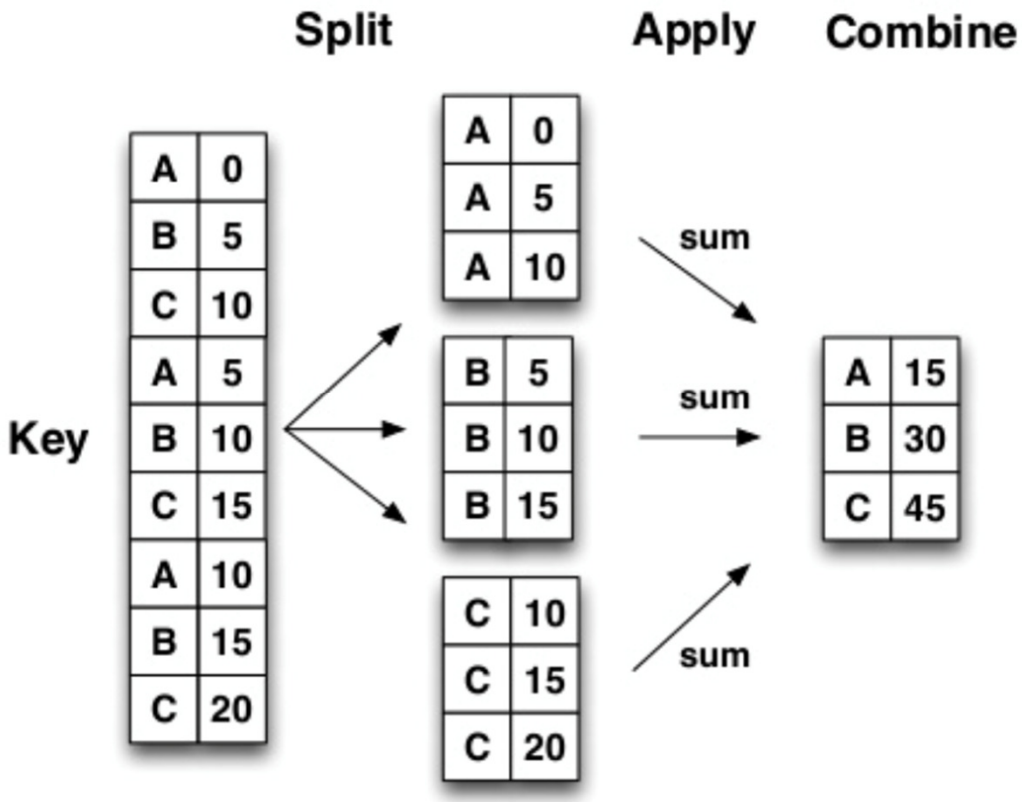
	col1	col2	col3
0	1.0	NaN	abc
1	2.0	555.0	def
2	3.0	666.0	ghi
3	NaN	444.0	xyz

```
df.isnull()
```

	col1	col2	col3
0	False	True	False
1	False	False	False
2	False	False	False
3	True	False	False

Pandas: Groupby

- **GroupBy** allows you to group together rows based on a column and perform an aggregate function on them.
  - Ex: sum students by grading (HD, D, C, P, F).



Pandas: Groupby

- Create a DataFrameGroupBy object and apply call aggregate methods.

```
# Create dataframe
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
        'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
        'Sales':[200,120,340,124,243,350]}
```

```
df = pd.DataFrame(data)
```

```
df
```

	Company	Person	Sales
0	GOOG	Sam	200
1	GOOG	Charlie	120
2	MSFT	Amy	340
3	MSFT	Vanessa	124
4	FB	Carl	243
5	FB	Sarah	350

```
df.groupby('Company').mean()
```

	Sales
Company	
FB	296.5
GOOG	160.0
MSFT	232.0

Pandas: Groupby

- More aggregate methods:
  - Standard deviation
  - Minimum
  - Maximum
  - Count

```
by_comp.std()
```

	Sales
Company	
FB	75.660426
GOOG	56.568542
MSFT	152.735065

```
by_comp.min()
```

	Person	Sales
Company		
FB	Carl	243
GOOG	Charlie	120
MSFT	Amy	124

```
by_comp.max()
```

	Person	Sales
Company		
FB	Sarah	350
GOOG	Sam	200
MSFT	Vanessa	340

```
by_comp.count()
```

	Person	Sales
Company		
FB	2	2
GOOG	2	2
MSFT	2	2

Pandas: Combining Data Frame

- There are 3 main ways of combining DataFrame:
  - Concatenating
  - Merging
  - Joining
- Example: Suppose you have 3 DataFrames and you want to combine them:

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

df3

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Pandas: Concatenation

- **Concatenation**: basically glues together DataFrames rows.
  - Keep in mind that dimensions should match along the axis (column) you are concatenating on.
- You can use **pd.concat** and pass in a list of DataFrames to concatenate together:

```
pd.concat([df1,df2,df3])
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

Pandas: Merging

- **Merge**: allows you to merge DataFrames together using a similar logic as merging SQL Tables → Merge columns.
- Example:

```
left
```

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

```
right
```

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

```
pd.merge(left,right,how='inner',on='key')
```

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

Pandas: Joining

- **Join**: Joining is a convenient method for combining the columns of two potentially differently-indexed DataFrames into a single result DataFrame.
- Example:

```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                      'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

```
left.join(right)
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2

```
left.join(right, how='outer')
```

	A	B	C	D
K0	A0	B0	C0	D0
K1	A1	B1	NaN	NaN
K2	A2	B2	C2	D2
K3	NaN	NaN	C3	D3

Pandas: UNIQUE and NUNIQUE

❖ Example:

```
import pandas as pd
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666,444],'col3':['abc','def','ghi','xyz']})
df.head()
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

```
df['col2'].unique()
```

```
array([444, 555, 666])
```

```
df['col2'].nunique()
```

```
3
```

```
df['col2'].value_counts()
```

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

Pandas: Sorting a DataFrame

- Example:
  - Sort the entire rows by a given column.

```
df
```

	col2	col3
0	444	abc
1	555	def
2	666	ghi
3	444	xyz

```
df.sort_values(by='col2') #inplace=False by default
```

	col2	col3
0	444	abc
3	444	xyz
1	555	def
2	666	ghi

Pandas: Pivot Table - Reshape

❖ Example:

```
data = {'A':['foo','foo','foo','bar','bar','bar'],
     'B':['one','one','two','two','one','one'],
       'C':['x','y','x','y','x','y'],
       'D':[1,3,2,5,4,1]}

df = pd.DataFrame(data)
```

```
df
```

	A	B	C	D
0	foo	one	x	1
1	foo	one	y	3
2	foo	two	x	2
3	bar	two	y	5
4	bar	one	x	4
5	bar	one	y	1

```
df.pivot_table(values='D',index=['A', 'B'],columns=['C'])
```

	C	x	y
A	**B**		
bar	**one**	4.0	1.0
	two	NaN	5.0
foo	**one**	1.0	3.0
	two	2.0	NaN

Pandas: Data Input and Output

- Raw data is organized in different structures for the purpose of distribution and processing.
- How to read raw data into Pandas:
  - CSV
  - Excel
  - HTML
  - SQL
- Installation (using Anaconda Prompt):

```
conda install sqlalchemy
conda install lxml
conda install html5lib
conda install BeautifulSoup4
```

Optional: SQL

- The `pandas.io.sql` module provides a collection of query wrappers to both facilitate data retrieval and to reduce dependency on DB-specific API. Database abstraction is provided by `SQLAlchemy` if installed. In addition you will need a driver library for your database. E.g. for `SQLite` database this is included in Python's standard library by default.

Optional: SQL

- The key functions are:

  - `read_sql_table(table_name, con[, schema, ...])`

    Read SQL database table into a DataFrame.

  - `read_sql_query(sql, con[, index_col, ...])`

    Read SQL query into a DataFrame.

  - `read_sql(sql, con[, index_col, ...])`

    Read SQL query or database table into a DataFrame.

  - `DataFrame.to_sql(name, con[, flavor, ...])`

    Write records stored in a DataFrame to a SQL database.

Optional: SQL

❖ Example:

```
from sqlalchemy import create_engine
```

```
engine = create_engine('sqlite:///:memory:')
```

```
df.to_sql('data', engine)
```

```
sql_df = pd.read_sql('data',con=engine)
```

```
sql_df
```

	index	a	b	c	d
0	0	0	1	2	3
1	1	4	5	6	7
2	2	8	9	10	11
3	3	12	13	14	15

Other: SQL Pandas functions for summary

map() functions

filter (apply predicate to rows)

sort/group by

aggregate: sum, count, average, max, min

Pivot or reshape

Relational:

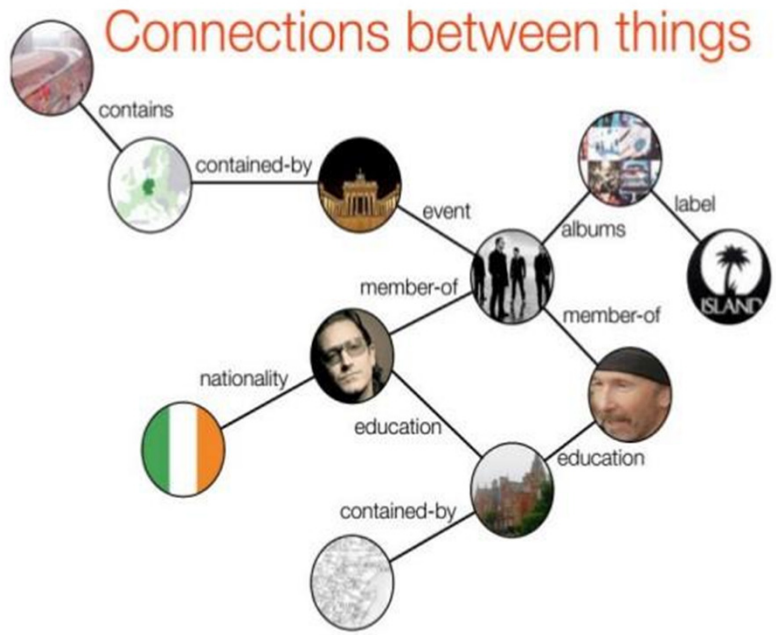
union, intersection, difference, cartesian product (CROSS JOIN), select/filter, project, join: natural join (INNER JOIN), theta join, semi-join, etc.

Optional: Other data models

- Document model (e.g. XML)

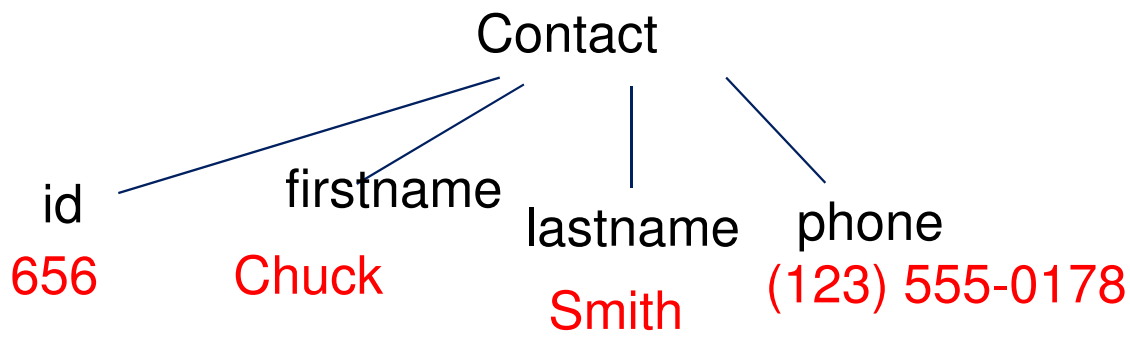
```
<contact>
   <id>656</id>
   <firstname>Chuck</firstname>
   <lastname>Smith</lastname>
   <phone>(123) 555-0178</phone>
</contact>
```

- Network model (e.g. graph DB)



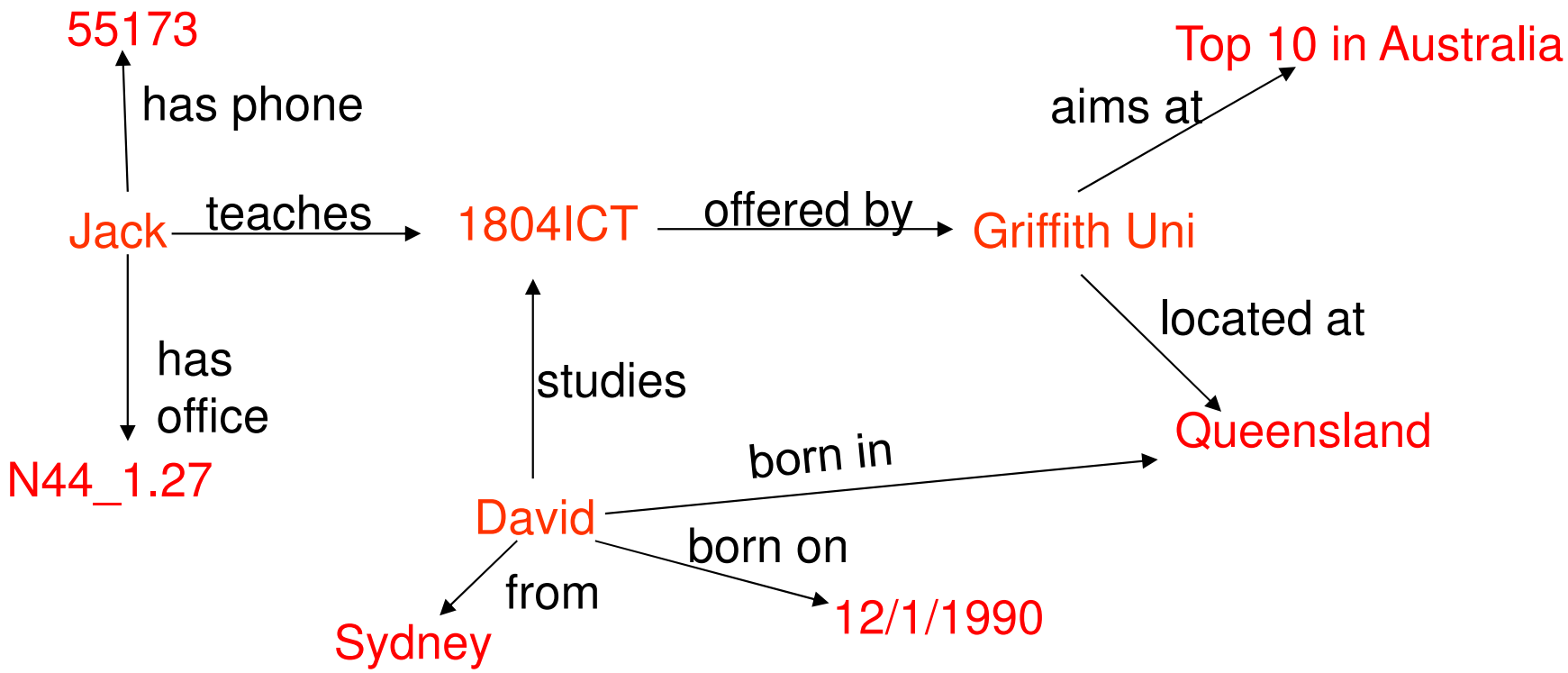
Optional: Document model

- Use hierarchical representation



Optional: Network model

- Use graph representation
- Example data:
  - entities: Jack, David, Griffith University, 1804ICT…
  - attributes: phone number, birthday, location…
  - relationships: Jack teaches 1804ICT at Griffith …



2. Data Normalization

- Why:
  - Non-normalized data might affect analysis results.
  - Data from different sources may be in different scales, ranges, and units → Meters and Feet.
  - Data may be biased/skewed.
- When:
  - Data should be in the same scale
  - Bias data into a specific range/size for specific purposes

Scale

- If your model is based on several numerical attributes – are they comparable?
  - Example: ages may range from 0-100, and incomes from 0-billions.
  - Some models may not perform well when different attributes are on very different scales.
  - It can result in some attributes counting more than others.
  - Bias/Skew in the attributes can also be a problem.

How to normalize

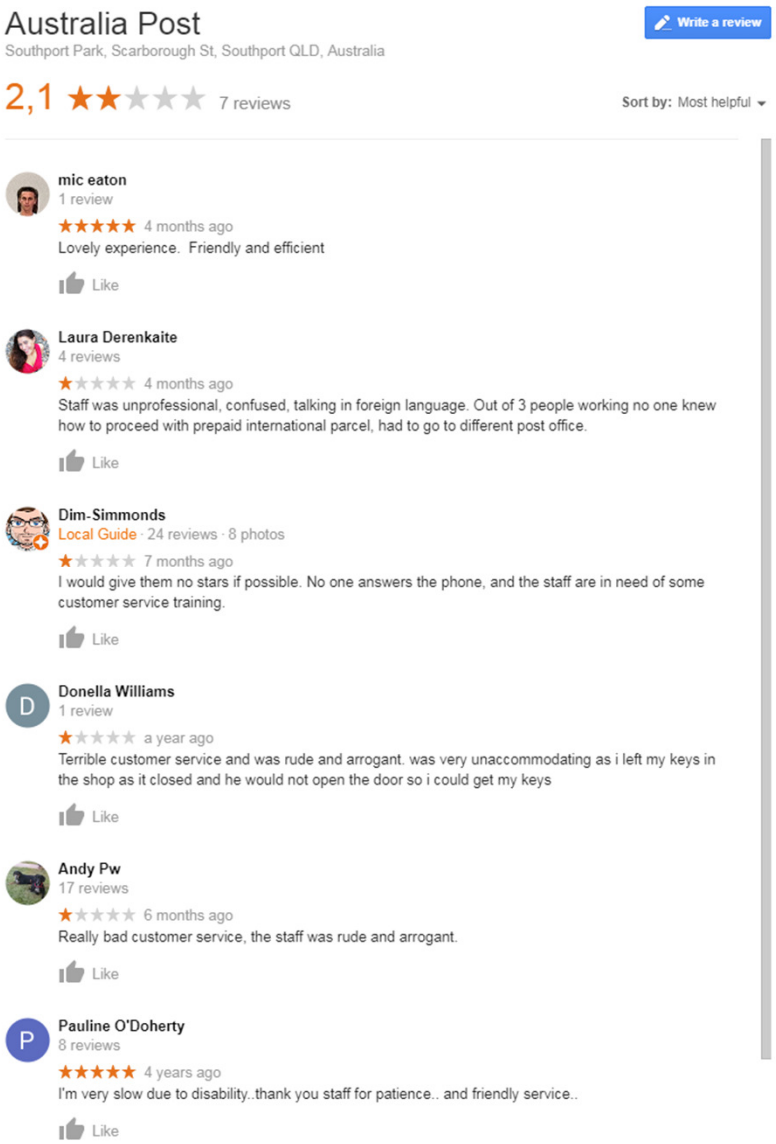
- Read documentation of tools:
  - Scikit-learn's PCA implementation has a "whiten" option
  - Sciki-learn has a preprocessing module with handy normalize and scale functions
  - Convert textual data to numerical/ordinal data might be convenient for classification:
    - E.g. Your data may have "yes" and "no" that needs to be converted to "1" and "0".

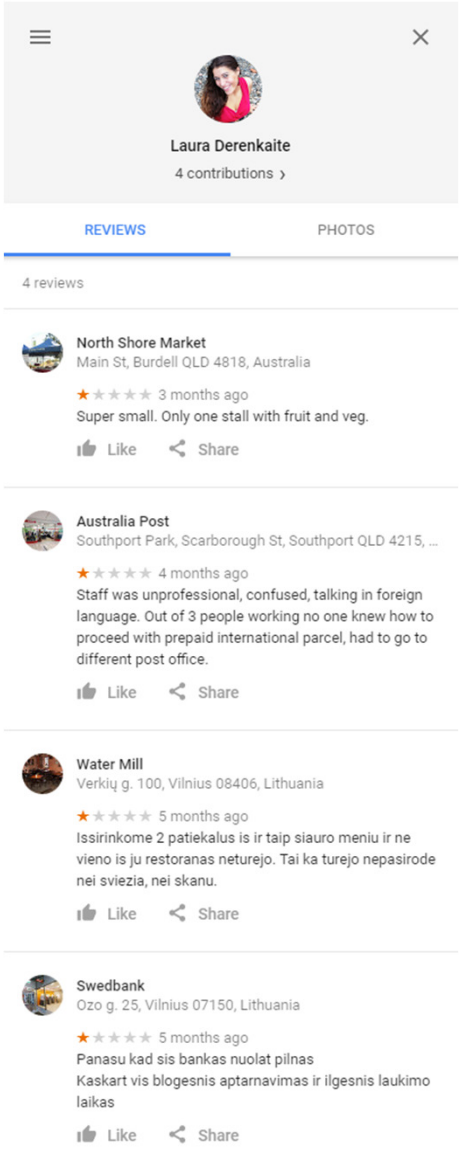
- Don't forget to rescale/unbiased the results → correct interpretation

3. Data cleaning

- Data cleaning: very time-consuming step in data analytics
- Why data cleaning?
  - Make analytical result reliable
    - Result could be skewed/bias → mistakes, wrong business decisions (e.g. fake reviews lead to wrong conclusion of customer opinion)
  - Reduce the effort of data analytics:
    - Simple models on clean data can outperform complex models on dirty data
- Challenges:
  - Too many sources produce dirty data
  - No generic framework for data cleaning → handle case by case
  - Need background knowledge → human involvement→ costly, time consuming, not scalable

Why data cleaning? – Bad reviews



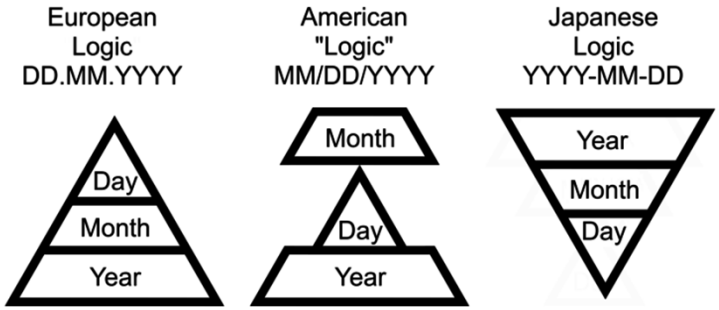


Types of “dirty” data

- Formatting
- Missing data
- Erroneous data
- Irrelevant data
- Inconsistent data
- Malicious data
- Outliers

Formatting

- **What:** the same entity can be inconsistently formatted
- **Why:** different standards, different platforms
- **Examples:**
  - Dates (US format …)
  - Phone numbers (parentheses, dashes)
- **How to handle:**
  - Identify all standard formats
  - Convert to the same format



Missing data

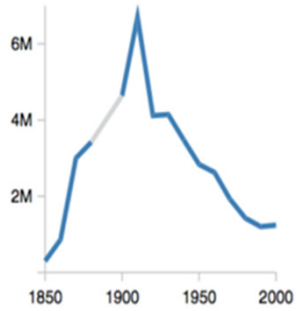
- **What:** Some of the data are not there
- **Why:**
  - Invalid data and ingestion
    - Invalid city
    - Missing state
    - Missing zipcode
  - Unexpected incident
    - Server crash

Customer	Address Line 1	Address Line 2	City	State	Zip
Antony Mc James IV	123 Untidy Cir	Suite# 234	Cumbersome	TX	76849
Miller Johnson	1102 Messy Data St	Middletow	OH		
Betty Flyier 6483 Phew Lane	Apt A4	FixMe	CA	91103	

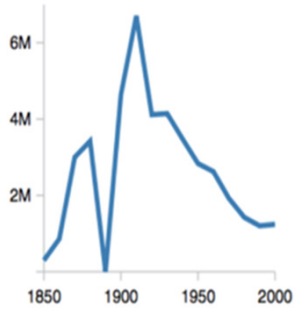
Dealing with missing data

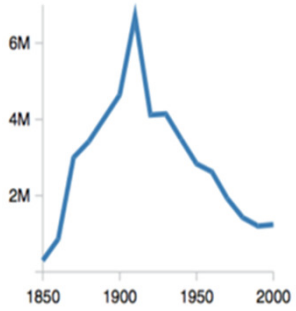
Knowledge about domain and data collection should drive your choice!

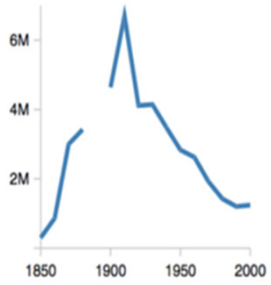
- Create new classification (e.g. set values to zero)
- Interpolate based on existing data
- Omit missing data



U.S. census counts of people working as ‘‘Farm Laborers’’; values from 1890 are **missing due to records being burned in a fire**







Erroneous data

- **What:** recurring error data in a particular case
- **Why:**
  - Software bug
  - Interrupted process that generates data: e.g. user is disconnected during a Web survey session
- **How to handle:**
  - Dig into unreasonable data
  - Look at things aren't being combined properly: e.g. sum, product

Irrelevant data

- **What:** data whose non-existence does not affect your results
- **Why:** redundant data crawled from the Web
- **Example:** you are interested in only data from Gold Coast city → all data from the rest of the world are irrelevant
- **How to handle:**
  - Simply throw all irrelevant data (or select the subset of data that are relevant)
    - Define the rules to filter out (or manually)
    - Be careful! Over-cleaning irrelevant data might become missing data

Inconsistent data

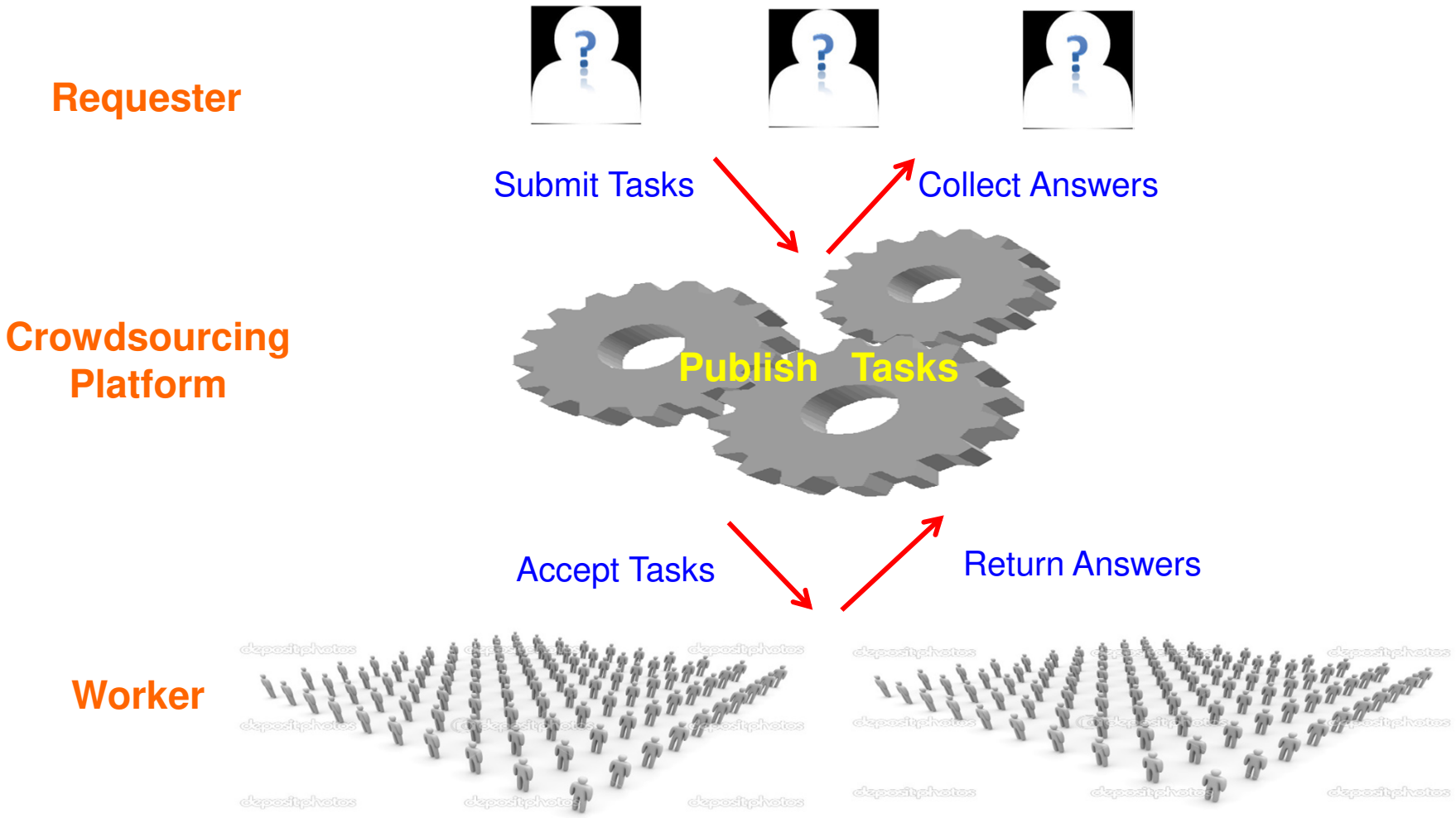
- **What:** the same data can be represented in different ways
- **Why:**
  - Different platforms
  - Different input preferences
- **Examples:**
  - Same address can be written in many different ways (street abbreviation, order between number and street, zip).
  - Movies/Books might have different names in different countries
- **How to handle:**
  - List all variations and data representations
  - Normalize/combine them all together to get the correct results

Malicious data

- **What:** data is intended to cause undesired effects
- **Why:**
  - People trying to game/trick/cheat your system
  - Malicious attacks: e.g. spam
- **How to deal:**
  - Identify the attacks
  - Filter them out from the results

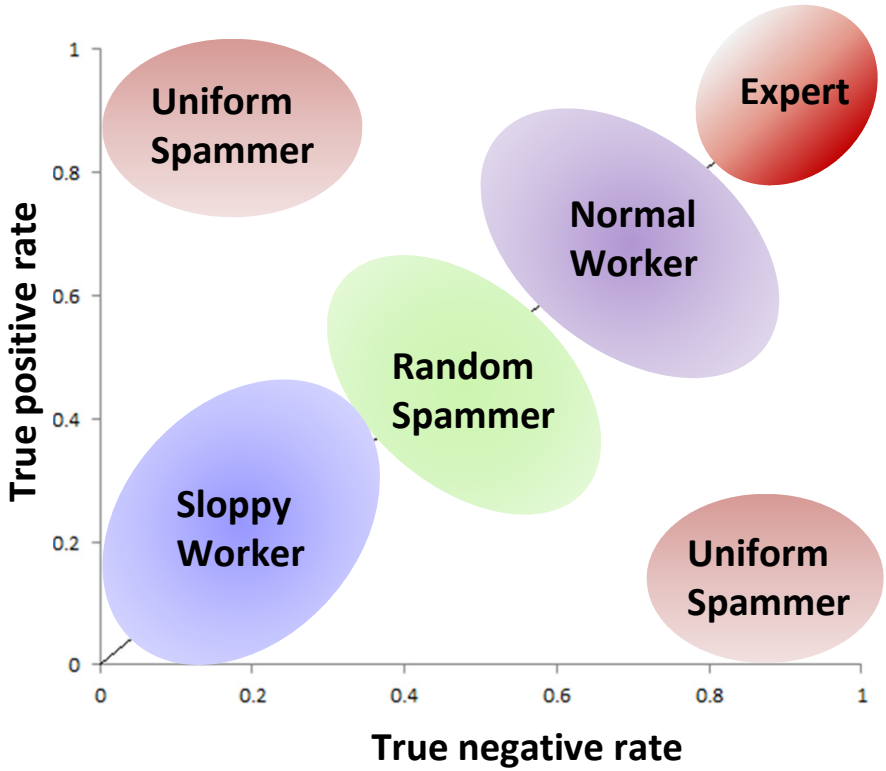
Malicious Data: Example

- The Case of Crowdsourcing



Malicious Data: Example

- Truthful workers
  - Expert
  - Normal
- Untruthful workers
  - Sloppy
  - Uniform spammer
  - Random spammer



Malicious Data: Example

- Who is spammer?

Binary Questions: evaluate the sentiment of a given sentence (1: positive, 0: negative)

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
W1	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0	1	0
W2	1	0	1	1	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0
W3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W4	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	0	1	0
W5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
W6	1	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	1
W7	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0
W8	0	1	1	0	1	0	1	0	1	0	0	0	1	0	1	1	1	0	1	0

Workers

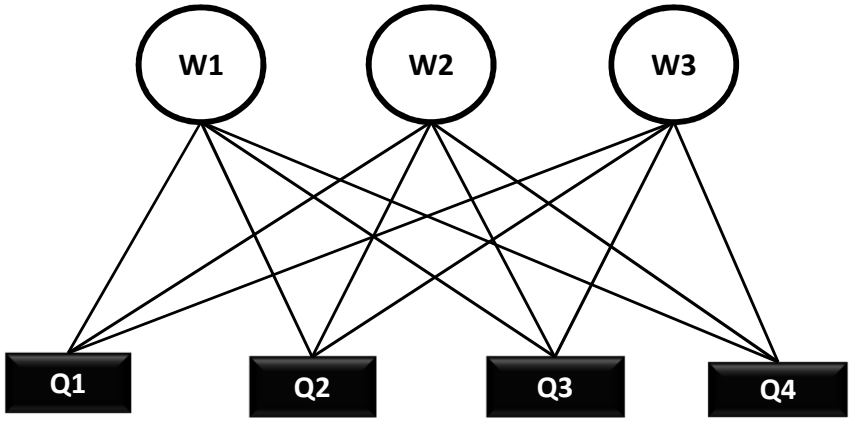
Malicious Data: Example

- Aggregated result

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
W1	1	1	1	0	0	0	1	1	1	0	0	0	1	0	1	0	1	0	1	0
W2	1	0	1	1	1	1	1	0	0	0	1	0	1	1	1	0	1	0	1	0
W3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W4	1	0	0	0	1	1	1	0	1	1	1	0	1	0	0	0	1	0	1	0
W5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
W6	1	0	1	1	1	0	1	0	1	0	1	0	0	0	1	1	1	0	1	1
W7	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0
W8	0	1	1	0	1	0	1	0	1	0	0	0	1	0	1	1	1	0	1	0
R	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

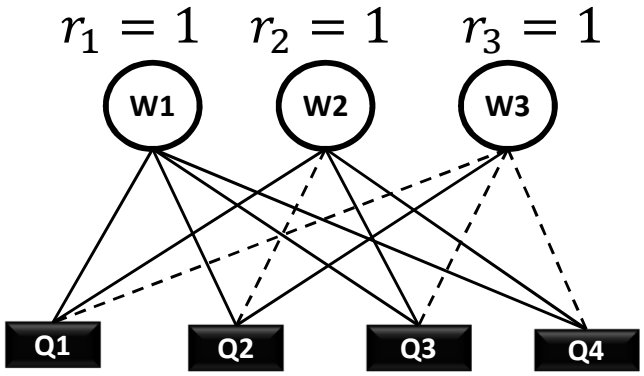
Malicious Data: Example

- How to detect malicious users:
  - Expectation Maximization (EM)
- How it works:
  - Workers who provide similar answers should have high reliability
  - Answers which are provided by reliable workers should have high quality



Expectation Maximization: example

- **Step 1:**
  - Assume reliability of each worker is 1
  - Aggregated result: Q1 = 1, Q2 = 1, Q3 = 1, Q4 = 1
- **Step 2:**
  - Update worker reliability: $r_1 = 1, r_2 = 0.75, r_3 = 0.25$
  - Aggregated result: Q1=1, Q2 = 1, Q3=1, Q4=1
- **Stop** as the result is converged



EM: Math

- Iterates in two steps
  1. **E-Step:** estimate the labels from the answers of workers
  2. **M-Step:** estimate the reliability of workers from the consistency of answers

EM: Math (Optional)

- **(E) step:** estimate answer for each question as $P(Q_j = \ell)$

$$P(Q_j = \ell) = \frac{1}{\sum_{i=1}^{N} r_i} \sum_{i=1}^{N} (r_i \mid a_i(Q_j) = \ell)$$

- **(M) step:** update the reliability as r_i

$$r_i = \frac{1}{M} \sum_{j=1}^{M} (1 \mid a_i(Q_j) = \operatorname{argmax}_{\ell} P(Q_j = \ell))$$

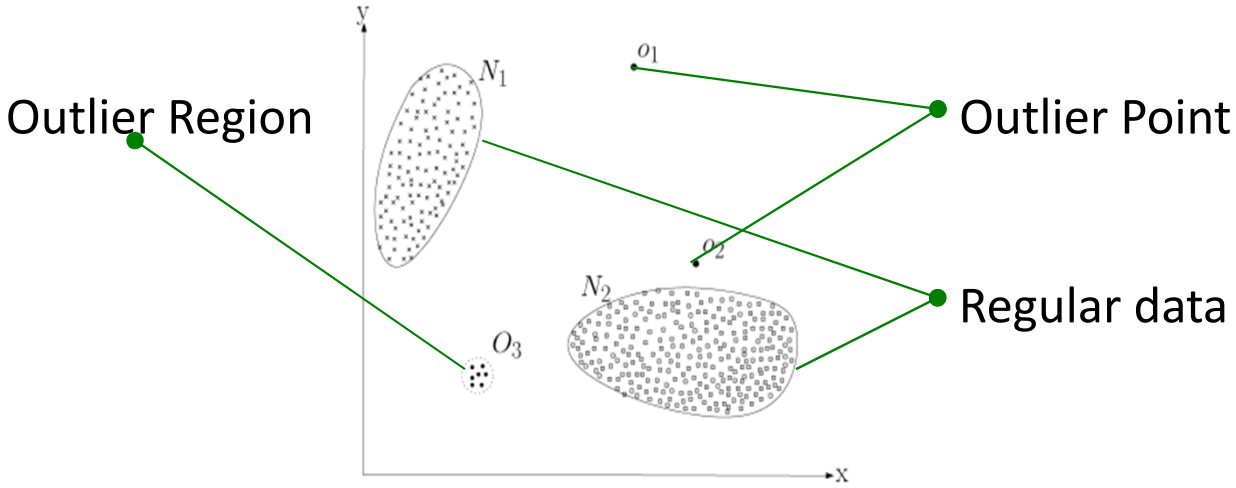
r_i reliability of worker *i*

M number of questions to answer

N number of workers

Outliers

- **Outlier:** an observation point that is distant from other observations (a.k.a. noises, anomalies)
  - E.g. malicious data are often outliers
  - But non-malicious data can still be outlier



A simple example of anomalies in a two-dimensional data set [Chandola et al. 2009].

Source of Outliers

- **Human errors**
  - Due to measurements (measure wrongly)
  - By accidents (spelling, typos, data entry errors, etc.)
  - Due to human bias (e.g., coding disease code in hospital), etc.
- **Errors due to machines**
  - Software bugs
  - Data crawling errors
  - Data integration errors, etc.
- **Others**
  - Errors may be deliberate, duplicates, stale data (artifact of caching, not being refreshed).

How to handle outliers

- Use standard deviation
  - e.g. Find data points more than some “multiple” of a standard deviation of data
- Use statistical functions describing distribution: e.g. histogram
  - e.g. Points that lie in the tails are outliers
- Use scatter plots:
  - e.g. Points that far away from others are outliers

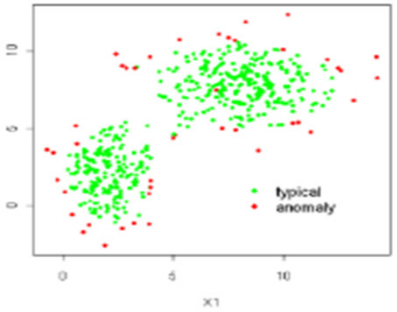
How to handle outliers (cont'd)

- Exclude them in the analysis
  - Sometimes it's appropriate to remove outliers
    - But do this responsibly (understand why)
    - Do in it a principled manner (outlier detection)
- Sometimes we need to treat them as points of interest
  - E.g. calculating the mean income of US citizens: don't toss out billionaires.

How to handle outliers (Optional)

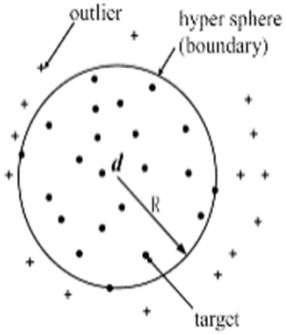
Distance-based

Based on well-defined distance metrics to compute the dissimilarity between two data points



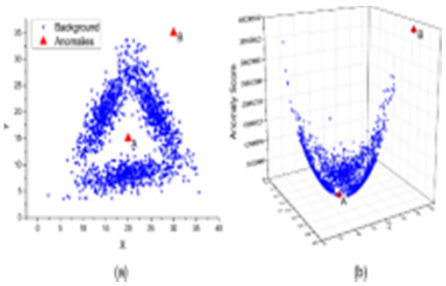
Domain-based

Learn domain of normality that characterises normal data



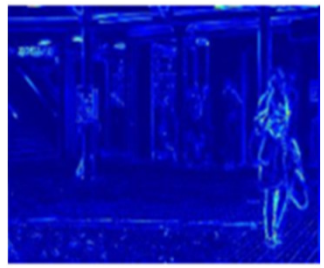
Information theoretic

Based on the information content of data to decide (e.g., entropy, mutual information, KL divergence)



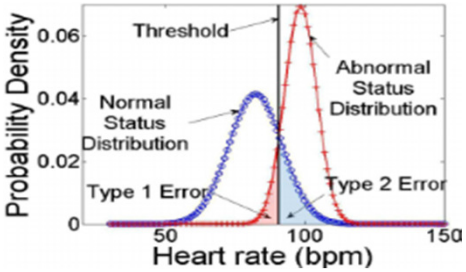
Reconstruction-based

Based on reconstruction error to decide (e.g., PCA, auto-encoder)



Probabilistic

Based on probabilistic likelihood to decide (e.g., parametric and non-parametric mixture model)



Data Cleaning: Summary

- Look at your data and examine it
- Question your results:
  - Look for weird things
  - Suspect "good" things even if you like your results
- Best practices:
  - Check frequencies of continuous and categorical variables for detection of unexpected values. For continuous variables, look into data "clumps" and "gaps".
  - Check the type or numeric variables: decimal, integer, and date.
  - Check the meanings of misinformative values, e.g., "NA", the blank, "", the number '0', the letter 'O', the dash, "-", and the dot "."
  - Check for out-of-range data: Values "far out" from the fences" of the data.
  - Check for outliers: Values "outside" the fences of the data.
  - Check for missing values, and the meanings of their coded values, e.g, the varied string of "9s", the number "0", the letter "O", the dash "-", and the dot ".".
  - Check the logic of data, e.g., response rates cannot be 110%, and weigh contradictory values, along with conflict resolution rules, e.g., duplicate DOB: 12/22/56 and 12/22/65.
  - Last but not least, check for the typos.

Data cleaning: Usecase

- Review data from Amazon
- Import data

```
ds_folder = "../dataspace/amazon-reviews/"
dataset = getDF(ds_folder + 'reviews_Cell_Phones_and_Accessories_5.json.gz')
dataset.head()
```

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime
0	A30TL5EWN6DFXT	120401325X	christina	[0, 0]	They look good and stick good! I just don't li...	4.0	Looks Good	1400630400	05 21, 2014
1	ASY55RVNIL0UD	120401325X	emily l.	[0, 0]	These stickers work like the review says they ...	5.0	Really great product.	1389657600	01 14, 2014
2	A2TMXE2AFO7ONB	120401325X	Erica	[0, 0]	These are awesome and make my phone look so st...	5.0	LOVE LOVE LOVE	1403740800	06 26, 2014
3	AWJ0WZQYMYFQ4	120401325X	JM	[4, 4]	Item arrived in great time and was in perfect ...	4.0	Cute!	1382313600	10 21, 2013
4	ATX7CZYFXI1KW	120401325X	patrice m rogoza	[2, 3]	awesome! stays on, and looks great. can be use...	5.0	leopard home button sticker for iphone 4s	1359849600	02 3, 2013

Usecase: Cleaning Amazon reviews

- Formatting
  - Convert data to a standard format, so it can be manipulated easily.

reviewTime is not datetime type so we have to convert it to datetime type

```
df['reviewTime_convert']=pd.to_datetime(df.reviewTime)
```

```
df[['reviewTime','reviewTime_convert']].head(5)
```

	reviewTime	reviewTime_convert
0	05 21, 2014	2014-05-21
1	01 14, 2014	2014-01-14
2	06 26, 2014	2014-06-26
3	10 21, 2013	2013-10-21
4	02 3, 2013	2013-02-03

Usecase: Cleaning Amazon reviews

- Formatting
  - Data of same type should be in the same format

Change the type of unixReviewTime to datetime format.

```
df['unixReviewTime_convert'] = pd.to_datetime(df['unixReviewTime'],unit='s')
df[['unixReviewTime','unixReviewTime_convert']].head()
```

	unixReviewTime	unixReviewTime_convert
0	1400630400	2014-05-21
1	1389657600	2014-01-14
2	1403740800	2014-06-26
3	1382313600	2013-10-21
4	1359849600	2013-02-03

Usecase: Cleaning Amazon reviews

- Formatting

Change overall to integer type

```
df['overall']=df['overall'].astype(int)
df['overall'].head()
```

```
0    4
1    5
2    5
3    4
4    5
Name: overall, dtype: int64
```

Usecase: Cleaning Amazon reviews

- Check for missing values
  - Displays the number of missing values in each column.
  - Should we remove the *reviewerName* column?

```
df.isnull().sum()
```

```
reviewerID                  0
asin                        0
reviewerName             3519
helpful                     0
reviewText                  0
overall                     0
summary                     0
unixReviewTime              0
reviewTime                  0
reviewTime_convert          0
unixReviewTime_convert      0
year                        0
length_review               0
helpfulness_rate            0
dtype: int64
```

Usecase: Cleaning Amazon reviews

- **Dealing with missing values:** We notice that some of the reviewer names are missing but since the reviewer Id are available anyway, we can replace missing values with “Unknown”.

```
df.fillna('Unknown',inplace=True)
```

```
df.isnull().sum()
```

```
reviewerID                0
asin                      0
reviewerName              0
helpful                   0
reviewText                0
overall                   0
summary                   0
unixReviewTime            0
reviewTime                0
reviewTime_convert        0
unixReviewTime_convert    0
year                      0
length_review             0
helpfulness_rate          0
dtype: int64
```

Usecase: Cleaning Amazon reviews

- Check for invalid data
  - **overall** must be between **0 – 5**

```
df.loc[(df['overall']<0) | (df['overall']>5)]
```

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime	reviewTime_convert	unixReviewTime_con

No invalid data!

Usecase: Cleaning Amazon reviews

- Check for invalid data
  - **reviewTime** must be between **May 1996 – July 2014**

```
df.loc[(df['reviewTime_convert']<'1996-05-01') |
       (df['reviewTime_convert']>'2014-07-31')]
```

	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime	reviewTime_convert	unixReviewTime_con

No invalid data!

Usecase: Cleaning Amazon reviews

- **Cleaning irrelevant data**: The field *unixReviewTime* has the same meaning of *reviewTime* in the dataset, thus we can remove this column.

```
del df['unixReviewTime']
```

Use case: Cleaning Amazon reviews

- **Cleaning inconsistent data**: two different reviewers should not have the same ID.

```
group = df.groupby('reviewerID')['reviewerName'].unique()
group[group.apply(lambda x: len(x)>1)].head(10)
```

```
reviewerID
A102TCMSQAJIDR                                  [crmorris, Unknown]
A103CDLPIN70O9                                  [Patricia, Unknown]
A103D3GYGFHS96                          [Daisy Cartagena, Unknown]
A103EFN0LE0PPT                                      [Unknown, tmar]
A10436JZZW87TE                                    [Hector, Unknown]
A104R1UFTJNBKK                          [Amazon Customer, Unknown]
A105K765Z5SX1A                          [Springs Shopper, Unknown]
A105S56ODHGJEK    [Peace Daddy "Eclectic ReflectionZ", Unknown]
A10DHJK4D0QFKR                                      [stu, Unknown]
A10DHVWCLL3EA3                                [Bassinator, Unknown]
Name: reviewerName, dtype: object
```

References

[1] https://www.slideshare.net/AshwiniKuntamukkala/data-wrangling-62017599

[2] https://www.slideshare.net/hhamalai/python-for-data-science

[3] https://www.slideshare.net/hassass15/data-cleaning-and-screening-58372141

[4] https://machinelearningmastery.com/handle-missing-data-python/

[5] https://www.slideshare.net/AshwiniKuntamukkala/data-wrangling-62017599

[6] https://www.slideshare.net/hhamalai/python-for-data-science

[7] https://www.slideshare.net/hafidztio/resampling-methods

[8] https://www.slideshare.net/AjinkyaMore3/python-resampling

[9] http://kindsonthegenius.blogspot.com.au/2017/12/dimensionality-reduction-and-principal.html

[10] https://www.udemy.com/data-science-and-machine-learning-with-python-hands-on

[11] https://www.youtube.com/watch?v=NWldkpR2sGA