Practical Data Science – COSC2670

Practical Data Science:
Data Curation

Dr. Yongli Ren

(yongli.ren@rmit.edu.au)

Computer Science & IT

School of Science

RMIT UNIVERSITY

Outline

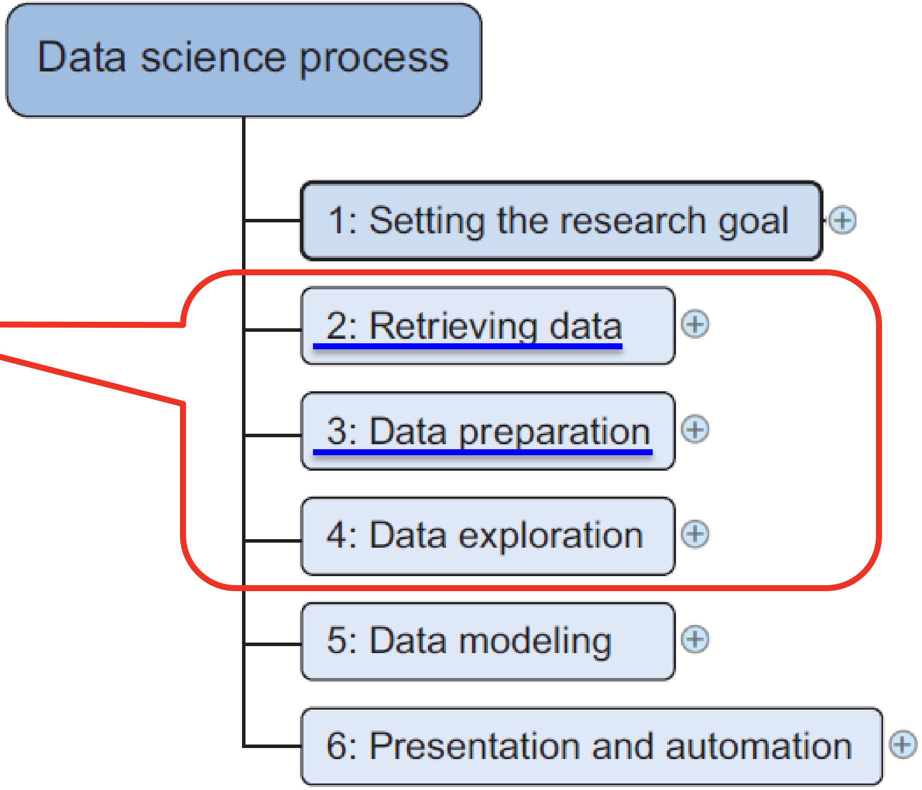
- Part 1: Overview
- Part 2: Data Retrieving
- Part 3: Data Preparation

Practical Data Science – COSC2670

PART 1: OVERVIEW

Data Curation

- Expect to spend a good portion of your project time doing data correction and cleansing
  - Sometimes up to 80%



- Data curation is done during

- The difference is in
  - the goal and
  - the depth of the investigation.

Data Curation
- Retrieving Data

- Data Retrieving:
  - This is the 1st time you inspect the data in the data science process.
  - Most of the errors here are easy to spot.
  - Focus on
    - *If the data is equal to the data in the source document and*
    - *If you have the right data types.*
  - This shouldn't take too long.
  - You stop when you have enough evidence that the data is similar to the data you find in the source document.
  - But, many hours solving data issues may be caused if you are careless here.

Data Curation
- Data Preparation

- Data Preparation
  - You do a *more elaborate* check.
  - The errors here should also be present in the source document,
    - If you did a good job during the previous step.
  - The focus is on the content of the variable:
    - Typos: USQ to USA
    - Other data entry errors
    - Missing values
    - Bad lines

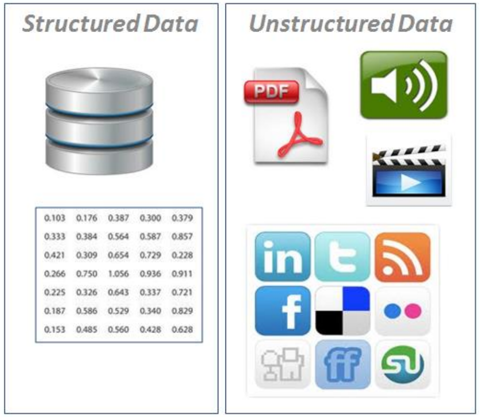
  - Bring the data to a common standard among the data sets.

Data Curation
- Data Exploration

- Data Exploration
  - Focus on *what you can learn from the data*.
  - Now, you assume the data to be clean and look at the statistical properties
    - E.g. distributions,
    - correlations, and
    - *outliers*

- You will often iterate over these three steps to do data curation to prevent problems later. E.g.
  - If you discover outliers in the exploratory phase, they can point to a data entry error.

Data Source and Data Type



- Data Source
  - Files on the disk
  - Tables in a database, data warehouse, other data repositories.
- Structured vs Unstructured Data
  - Structured Data: is information, usually text files, displayed in titled columns and rows which can easily be ordered and processed by data mining tools.
    - This could be visualized as a perfectly organized filing cabinet where everything is identified, labeled and easy to access.
  - Unstructured data: usually has no identifiable internal structure.
    - Even if the data has some form of structure but this is not helpful to the processing task at hand, it may still be characterized as "unstructured."
    - E.g. Emails, Word Processing Files, PDF files, Spreadsheets, Digital Images, Video, Audio, Social Media Posts.

http://sherpasoftware.com/blog/structured-and-unstructured-data-what-is-it/
https://www.laserfiche.com/ecmblog/4-ways-to-manage-unstructured-data-with-ecm/

Data Source and Data Type

- Data Source Document
  - There might be no “Types” for data, e.g. data in CSV files.
  - Different levels of measurement
    - Nominal
    - Ordinal
    - Interval/ratio
      - Discrete/continuous
    - https://www.youtube.com/watch?v=hZxnzfnt5v8
- Data Types in Programming
  - Variables
  - Array
  - String

Variable Assignment

- A *variable* is a paired *name* and *storage location*
  - The storage location contains a *value* (some information)
- Variables are set with the = character
  - `$ count = 30`
- Conceptually, we've attached the name "count" to a storage location, and that location now holds the value 30
- More formally, two things happen:
  - Memory is allocated and an instance of an object of type *int* is created (since 30 is an integer value)
  - In the current namespace, a label is created for the new instance, binding the value into the namespace with the name "count"
- https://www.youtube.com/watch?v=aeoGGabJhAQ

Garbage Collection

- Python keeps track of the number of references to an object
  - `count = 30`
  - `range = count`
- While a reference to the *int* object exists, it will stay alive and allocated
- Once there are no references, the object is freed, and the python garbage collection process will reclaim the resources
  - `del count`
  - `del range`

Data Types

- The following basic types are available
  - **Integers** - Positive or negative whole numbers (3, -177)
  - **Floats** - Real numbers (-2.34, 1.23e-4)
  - **Strings** - Strings of single byte (8-bit) characters ("And now for something")
  - **Boolean** - True or False

Numbers

- 12 * 4
- 5 + 2
- 12 / 3

Strings

- Can be surrounded with single or double quotes
  - “Lucis”
  - ‘Nifelheim’
- Special characters are escaped with backslash
  - ‘Don\’t mess with Ifrit’
  - “Don’t mess with Ifrit”
- \n - newline
- \t – tab
- \\ - backslash

- The string concatenation operator is +
  - “do we have to” + “ go in there?”

Data Typing in Python

- Python is a *strongly typed* language
  - A variable can point to data of any type
    - level = 10
    - type(level)
    - class = “mage”
    - type(class)
  - You can point a variable to new data of a different type
    - level = “ten”
    - type(level)

Data Typing in Python…

- Trying to perform operations on incompatible types results in a runtime error

```
In [1]: a = 1
In [2]: type(a)
Out[2]: int
In [3]: b = 'two'
In [4]: type(b)
Out[4]: str
In [5]: a + b
------------------------------------------------------------
---------------
TypeError                                 Traceback (most
recent call last)
<ipython-input-5-f96fb8f649b6> in <module>()
----> 1 a + b
TypeError: unsupported operand type(s) for +: 'int' and
'str'
```

Python Tools
- Pandas

- Thanks to its specific object data structures, *DataFrames* and *Series*, pandas allows you to handle complex tables of data of different types and time series.
- Thanks to Wes McKinney's creation, you will be able to easily and smoothly load data from a variety of sources.
- You can then slice, dice, handle missing elements, add, rename, aggregate, reshape, and finally visualize this data at your will.

    - **Website**: `http://pandas.pydata.org/`
    - **Version at the time of print**: 0.15.2
    - **Suggested install command**: `pip install pandas`

Conventionally, pandas is imported as pd:

```
import pandas as pd
```

Import/load *pandas* package into memory, and give it a nickname "*pd*"

Python Tools
- Matplotlib

- Originally developed by John Hunter
- Matplotlib is the library that contains all the building blocks that are required to create quality plots from arrays and to visualize them interactively.
- You can find all the MATLAB-like plotting frameworks inside the pylab module.
  - **Website**: `http://matplotlib.org/`
  - **Version at the time of print**: 1.4.2
  - **Suggested install** command: `pip install matplotlib`

You can simply import what you need for your visualization purposes with the following command:

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

Import/load *matplotlib.pyplot* package into memory, and give it a nickname "*plt*"

Practical Data Science – COSC2670

PART 2:
DATA RETRIEVING

Data Retrieving

- Sometimes
  - You need to go into the field and design a data collection process yourself
- But most of the time
  - You won't be involved in this step.
- Many companies will have already collected and stored the data for you
- What they don't have can often by bought from third parties.
  - Don't be afraid to look outside your organization for data
    - because more and more organizations are making even high-quality data freely available for public and commercial use.

Data Retrieving

- Data can be stored in many forms
  - Ranging from simple text files to tables in a database.
    - E.g. csv, MS excel, SQL, JSON, HTML.
- Data retrieving is a critical part,
  - Especially when facing novel challenges.
- However, we will just briefly touch upon this aspect by
  - *Offering the basic tools to get your data into your computer memory by using either a textual file present on your hard disk or the Web.*

Fast and Easy Data Loading

- Let's start with a CSV file and pandas

Import/load *pandas* package into memory, and give it a nickname "*pd*"

Define a *variable* "iris_filename", and assign ("=") to it the value of "datasets-uci-iris.csv"

```
import pandas as pd
iris_filename = 'datasets-uci-iris.csv'
iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,
names= ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'target'])
```

Define a *variable* "iris", and assign to it the value returned by "pd.read_csv", which is the "read_csv" function in package pandas. The following bracket includes the *arguments* of the "read_csv" function: "(iris_filename, sep=',', decimal='.', header=None,names = […])"

Fast and Easy Data Loading

- Let's start with a CSV file and pandas

```
import pandas as pd
iris_filename = 'datasets-uci-iris.csv'
iris = pd.read_csv(iris_filename, sep=',', decimal='.', header=None,
names= ['sepal_length', 'sepal_width', 'petal_length', 'petal_width',
'target'])
```

- The arguments include:
  - The name of the file (*iris_filename*)
  - The character used as a separator (*sep,* default *sep=',’*)
  - The character used for the decimal placeholder (*decimal,* default *decimal='.'*)
  - The character used if there is a header (*header*)
  - The variables names (using *names* and a list)

  (The separator and the decimal may be different from the default)
- Let's open 'datasets-uci-iris.csv' to have a look.

http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Fast and Easy Data Loading

- If the dataset is not available in the online folder, you can follow these steps to download if from the **Internet**:

Define a variable “iris_p”, and assign it the value in the quotation marks, a String

```
import pandas as pd
iris_p = 'http://aima.cs.berkeley.edu/data/iris.csv'
iris_other = pd.read_csv(iris_p, sep=',', decimal='.', header=None,
names=['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'target'])
```

“iris_p” and “iris_other” are defined in a similar way

Fast and Easy Data Loading

- The resulting object, named iris, is a pandas DataFrame.
- To get a rough idea of its content, you can do:

```
In: iris.head()
Out:
   sepal_length  sepal_width  petal_length  petal_width      target
0           5.1          3.5           1.4          0.2      setosa
1           4.9          3.0           1.4          0.2      setosa
2           4.7          3.2           1.3          0.2      setosa
3           4.6          3.1           1.5          0.2      setosa
4           5.0          3.6           1.4          0.2      setosa


In: iris.head(2)


In: iris.tail()
[...]
```

Fast and Easy Data Loading

- To get the names of columns, you can simply use the following code:

```
In: iris.columns
Out: Index([u'sepal_length', u'sepal_width', u'petal_length',
  u'petal_width', u'target'], dtype='object')
```

- To extract the ‘target’ column, you do the following:

```
In: Y = iris['target']
Y
Out:
0           setosa
1           setosa
2           setosa
3           setosa
...
149          virginica
Name: target, Length: 150, dtype: object
```

Fast and Easy Data Loading

- To extract more than one column:

```
In: X = iris[['sepal_length', 'sepal_width']]
X
Out:
     sepal_length  sepal_width
0             5.1          3.5
1             4.9          3.0
2             4.7          3.2
...
147           6.5          3.0
148           6.2          3.4
149           5.9          3.0
[150 rows x 2 columns]
```

Fast and Easy Data Loading

- If you want to know the size of the problem, you need to know the size of the dataset.
- Typically,
  - For each observation, we count a line;
  - For each feature, a column

```
In:  X.shape
Out: (150, 2 )
In:   Y.shape
Out: (150,)
```

Practical Data Science – COSC2670

PART 3:
DATA PREPARATION

Data Preparation

- The data received from the data retrieval phase is likely to be
  - "*A Diamond in the Rough*"
- Your task now is to sanitize and prepare it for use in the modelling and reporting step.
- Doing so is *tremendously important* because your models will perform better and you'll lose less time trying to fix strange output.
- Remember:
  - *Garbage in equals garbage out.*

Data Preparation
- Cleansing Data

- Focus on
  - Removing errors in your data so that your data becomes a true and consistent representation of the processes it originates from.

Data Preparation
- Cleansing Data

An overview of common errors

General solution	
Try to fix the problem early in the data acquisition chain or else fix it in the program.	
Error description	**Possible solution**
Errors pointing to false values within one data set	
Mistakes during data entry	Manual overrules
Redundant white space	Use string functions
Impossible values	Manual overrules
Missing values	Remove observation or value
Outliers	Validate and, if erroneous, treat as missing value (remove or insert)

Data Preparation
- Data Entry Error

- Data collection and data entry are error-prone processes.
- *Errors can arise from human sloppiness, whereas others are due to machine or hardware failure.*
  - They often require human intervention,
    - Because humans are only human, they may make typos or lose their concentration for a second and introduce an error into the chain.
  - But, data collected by machines or computers isn’t free from errors either.
- For small data sets you can check every value *by hand*.
- Detecting data errors when the variables you study don’t have many classes can be done by *tabulating the data with counts*.
  - When you have a variable that can take only two values:
    - “Good” and “Bad”.
  - You can create a frequency table and see if those are truly the only two values present.

Data Preparation
- Data Entry Error

Detecting outliers on simple variables with a frequency table

Value	Count
Good	1598647
Bad	1354468
Godo	15
Bade	1

```
iris['target'].value_counts()
```

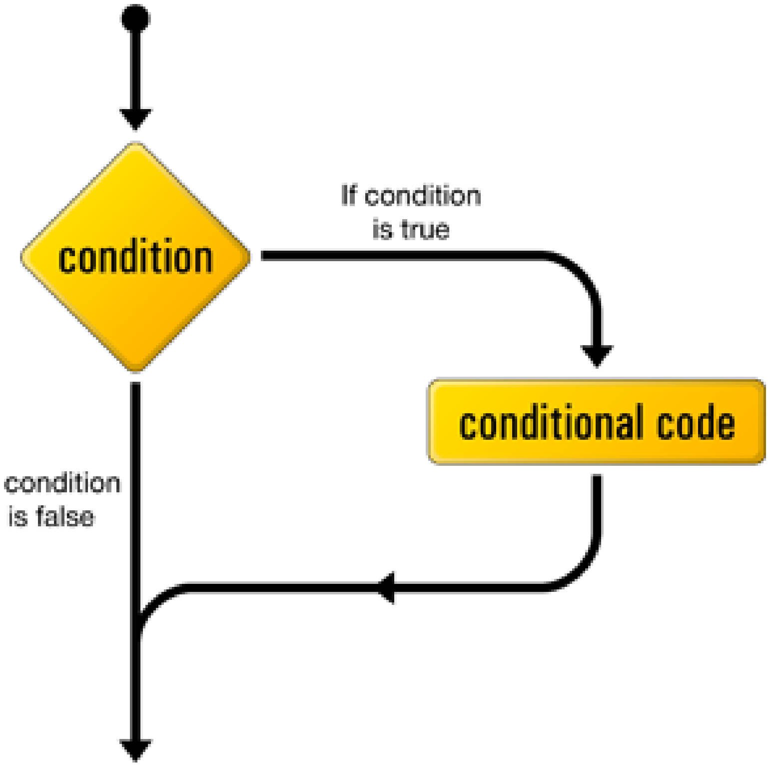
Data Preparation
- Data Entry Error

- Most errors of this type are easy to fix with simple assignment statements and if-then-else rules.

```
if x == “Godo”:
    x = “Good”
if x == “Bade”:
    x = “Bad”
```

“==“ Equal to (a comparison operator) - True if both operands are equal.

Comparison operators are used to compare values. It either returns True or False according to the condition.



Data Preparation
- Redundant Whitespace

- *Whitespaces* tend to be hard to detect but cause errors like other redundant characters would.
- *A common example: extra whitespace at the end of a string*
- If you know to watch out for them, fixing redundant whitespace is luckily easy enough in most programming languages.
- A string function that will remove the leading and trailing whitespaces
  - strip()

```
$ name = “your name”
$ name2 = “your name   ”
$ name == name2
$ name.strip() == name2.strip()
```

1) You ask the program to join two keys and notice that observations are missing from the output file.
2) After looking for days through the code, you finally find the *bug*.
3) Then comes the hardest part: explaining the delay to the project stakeholders.

   The cleaning phase wasn’t well executed, and keys in one table contained a whitespace at the end of a string.

   This caused a mismatch of keys such as “FR ” – “FR”, dropping the observations that couldn’t be matched.

Data Preparation
- Capital Letter Mismatches

- Capital letter mismatches are common.
- Most programming languages make a distinction between “Brazil” and “brazil”.
- In this case you can solve the problem by applying a function
  - that returns both strings in lowercase, such as *.lower()* in Python. “Brazil”.lower() == “brazil”.lower()
  - should result in true.

Data Preparation
- Impossible Values and Sanity Checks

- Sanity checks are another valuable type of data check.
- Here you check the value against physically or theoretically impossible values
  - such as people taller than 3 meters
  - or someone with an age of 299 years.
- Sanity checks can be directly expressed with rules:

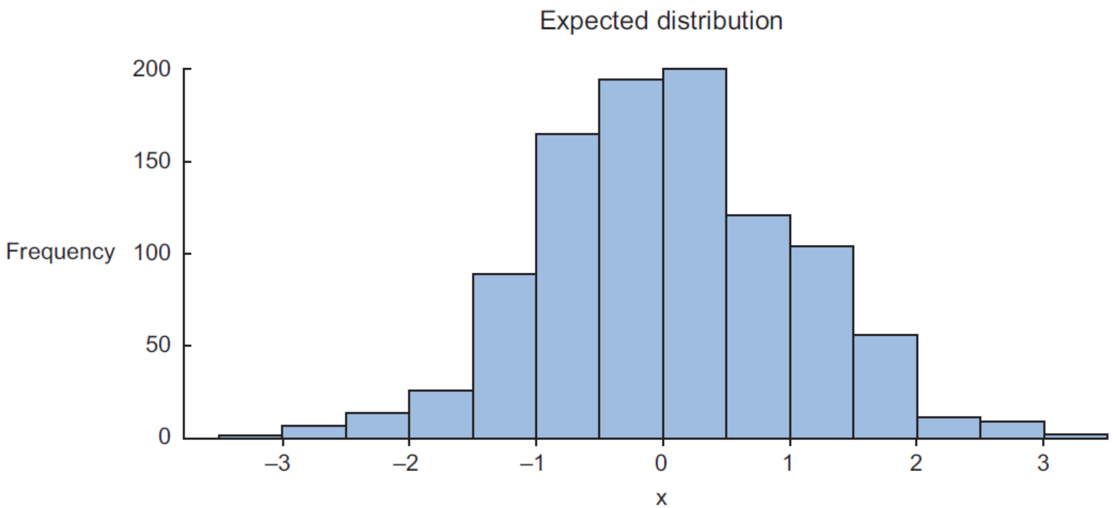
```
check = 0 <= age <= 120
```

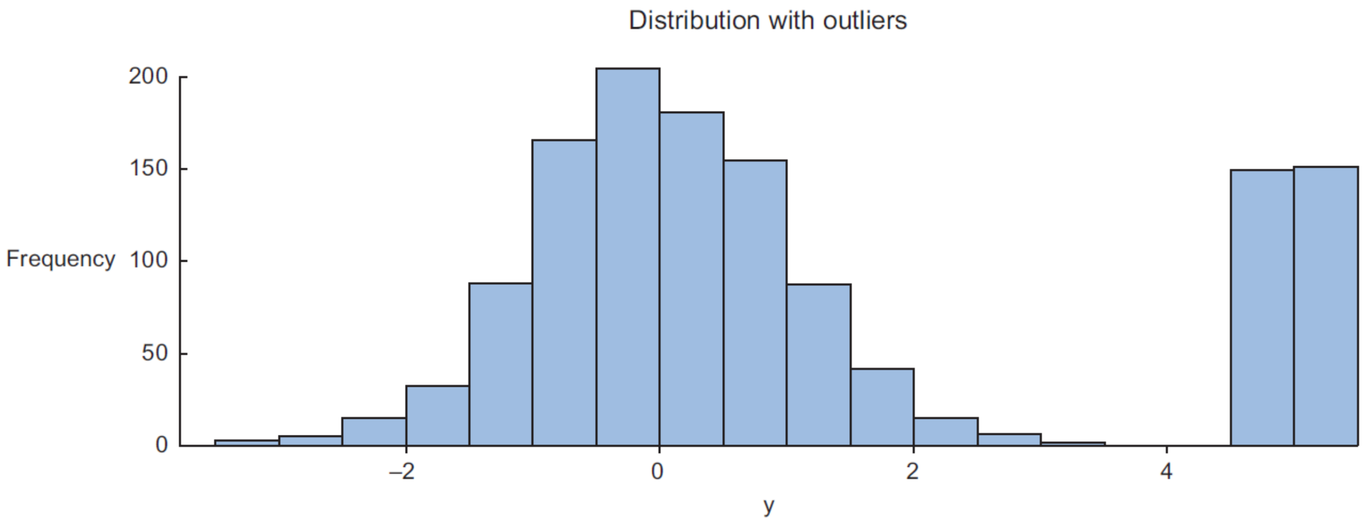
"<=" Less than or equal to (a comparison operator) - True if left operand is less than or equal to the right.

Comparison operators are used to compare values. It either returns True or False according to the condition.

Data Preparation
- Outliers

- An outlier is an observation that seems to be distant from other observations.
  - More specifically, one observation that follows a different logic or generative process than the other observations.
- The easiest way to find outliers is to use a plot or a table with the minimum and maximum values.
- Here is an example.





Distribution plots are helpful in detecting outliers and helping you understand the variable.

Data Preparation
- Missing Values

- Missing values aren't necessarily wrong,
  - but you still need to handle them separately;
- Certain modelling techniques can't handle missing values.
- They might be an indicator that
  - Something went wrong in your data collection or
  - that an error happened in the processing phase.
- Common techniques data scientists use are listed as follows.

An overview of techniques to handle missing data

Technique	Advantage	Disadvantage
Omit the values	Easy to perform	You lose the information from an observation
Set value to `null`	Easy to perform	Not every modeling technique and/or implementation can handle `null` values
Impute a static value such as 0 or the mean	Easy to perform You don't lose information from the other variables in the observation	Can lead to false estimations from a model
Impute a value from an estimated or theoretical distribution	Does not disturb the model as much	Harder to execute You make data assumptions
Modeling the value (nondependent)	Does not disturb the model too much	Can lead to too much confidence in the model Can artificially raise dependence among the variables Harder to execute You make data assumptions

Data Preparation
- Missing Values

- Let's see what happens if the CSV file contains
  - a header, and some missing values and dates.
- For example, to make things very easy and clear, let's imagine the situation of a travel agency.
  - *According to the temperature of three popular destinations,*
  - *they record whether the user picks the first, second, or the third destination.*

```
Date,Temperature_city_1,Temperature_city_2,Temperature_city_3,
  Which_destination
20140910,80,32,40,1
20140911,100,50,36,2
20140912,102,55,46,1
20140912,60,20,35,3
20140914,60,,32,3
20140914,,57,42,2
```

Data Preparation
- Missing Values

- In this case, all the numbers are integers and the header is in the file.
- In our first attempt to load this dataset, we can give the following command:

```
In: import pandas as pd

In: fake_dataset = pd.read_csv('a_loading_example_1.csv', sep=',')
fake_dataset

Out:

  Date   Temperature_city_1  Temperature_city_2  Temperature_city_3
    Which_destination

0  20140910  80   32   40  1

1  20140911  100  50   36  2

2  20140912  102  55   46  1

3  20140913  60   20   35  3

4  20140914  60   NaN  32  3

5  20140915  NaN  57   42  2
```

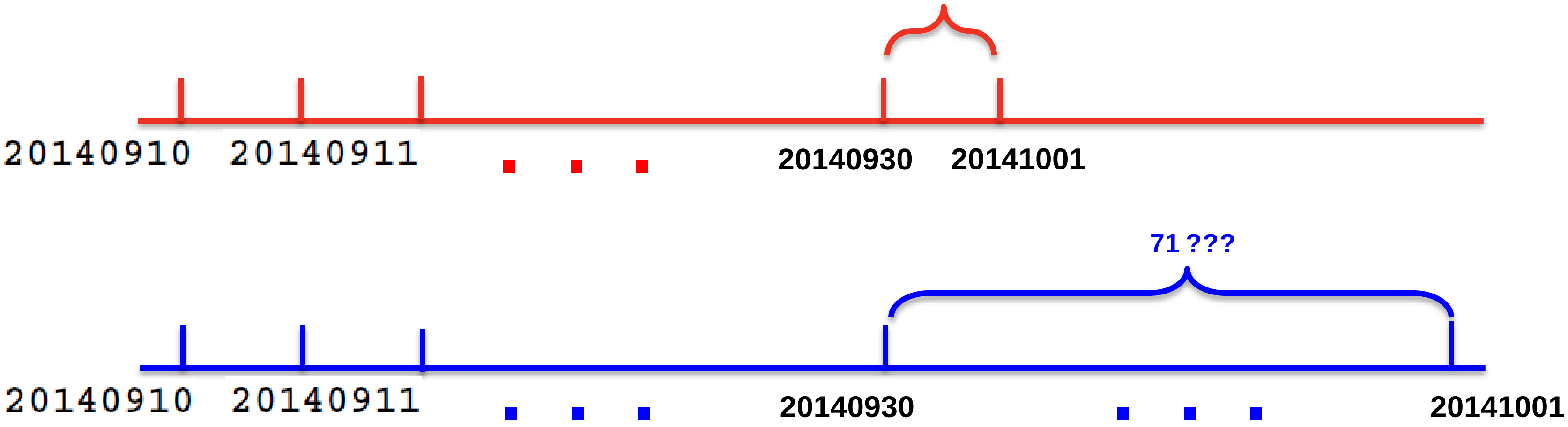
Data Preparation
- Missing Values

- That's a *great* achievement!
- pandas automatically named the columns with their actual name by taking the same from the first data row.
- We first detect a problem: all the data, even the dates, has been parsed as integers (or, in other cases, as string).

```
In: import pandas as pd

In: fake_dataset = pd.read_csv('a_loading_example_1.csv', sep=',')
fake_dataset

Out:

  Date   Temperature_city_1  Temperature_city_2  Temperature_city_3
    Which_destination

0  20140910  80   32   40  1

1  20140911  100  50   36  2
```

Date: 2014-Sep-10 or the number 20,140,910 ???

Data Preparation
- Missing Values




```
In: import pandas as pd

In: fake_dataset = pd.read_csv('a_loading_example_1.csv', sep=',')
fake_dataset

Out:

  Date   Temperature_city_1  Temperature_city_2  Temperature_city_3
    Which_destination

0  20140910  80   32   40  1

1  20140911  100  50   36  2
```

Date: 2014-Sep-10 or the number 20,140,910 ???

Data Preparation
- Missing Values

- If the format of the dates is not very strange, you can try the *autodetection* routines that specify the column that contains the date data.
- In this example, it works well with the following arguments:

```
In: fake_dataset = pd.read_csv('a_loading_example_1.csv',
parse_dates=[0])
fake_dataset

Out:

  Date   Temperature_city_1  Temperature_city_2  Temperature_city_3
Which_destination
0   2014-09-10   80   32   40   1

1   2014-09-11   100   50   36   2

2   2014-09-12   102   55   46   1

3   2014-09-13   60   20   35   3

4   2014-09-14   60   NaN   32   3

5   2014-09-15   NaN   57   42   2
```

Data Preparation
- Missing Values

- Now, to get rid of the missing data that is indicated as *NaN*,
  - replace them with a more meaningful number (let's say 30 Fahrenheit, for example).
- We can do this in the following way:

```
In: fake_dataset.fillna(30)

Out:

Date   Temperature_city_1  Temperature_city_2  Temperature_city_3  Which_
destination
0   2014-09-10   80   32   40   1

1   2014-09-11   100   50   36   2

2   2014-09-12   102   55   46   1

3   2014-09-13   60   20   35   3

4   2014-09-14   60   30   32   3

5   2014-09-15   30   57   42   2
```

Data Preparation
- Missing Values

- Here, all the missing data disappears.
- Treating missing data can require different approaches.
- As an alternative to the previous command, values can be replaced by a negative constant value to mark the fact that they are different from others (and leave the guess for the learning algorithm):

```
In: fake_dataset.fillna(-1)
```

- *NaN* values can also be replaced by the column mean or median value as a way to minimize the guessing error:

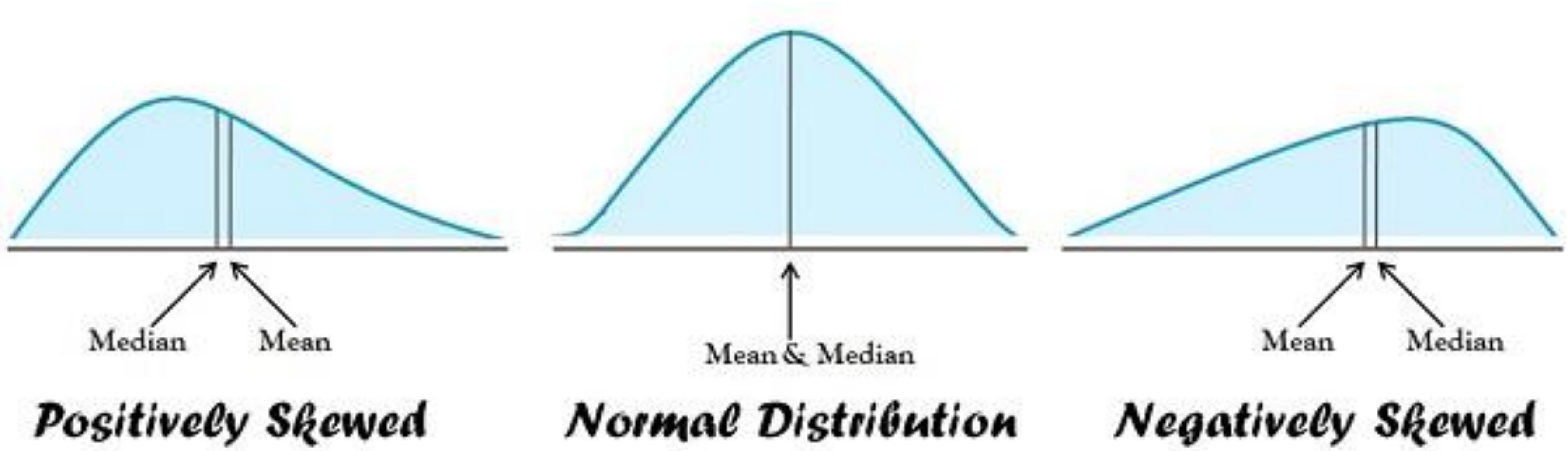
```
In: fake_dataset.fillna(fake_dataset.mean(axis=0))
```

Data Preparation
- Missing Values

```
In: fake_dataset.fillna(fake_dataset.mean(axis=0))
```

- The *.mean* method calculates the mean of the specified axis.
- Please note that *axis= 0* implies the calculation of the means span the rows, so obtained means extend column-wise.
- Instead, *axis=1* spans columns and therefore, row-wise results are obtained.
- This works in the same way for all other methods that require the axis parameter, both in pandas and NumPy.
- The *.median* method is analogous to *.mean* , but it computes the median value, which is useful in case the mean is not so well representative given too skewed data.

Data Preparation
- Missing Values



http://keydifferences.com/difference-between-mean-and-median.html

Data Preparation
- Missing Values

```
  Date   Temperature_city_1  Temperature_city_2  Temperature_city_3
Which_destination
0  2014-09-10  80   32   40   1

1  2014-09-11  100   50   36   2

2  2014-09-12  102   55   46   1

3  2014-09-13  60   20   35   3

4  2014-09-14  60   NaN   32   3

5  2014-09-15  NaN   57   42   2
```

Data Preparation
- Missing Values

Items. e.g. films

Users	Spider-Man	2012	The Godfather	The Social Network
Bob	3	2	4	5
Cindy	ø	3	5	4
Paul	2	3	4	ø
David	3	ø	4	1

rating

No rating

David did not watch '2012'?

David does not like '2012'?

Data Preparation
- Bad Lines

- Another possible problem when handling real world datasets is the loading of a dataset with errors or bad lines.
- In this case, the default behavior of the *load_csv* method is to stop and raise an exception.
- A possible workaround, which is not always feasible, is to ignore this line.
- In many cases, such a choice has the sole implication of training the machine learning algorithm without any observation.
- Let's say that you have a badly formatted dataset and you want to load just all the good lines and ignore the badly formatted ones.

Data Preparation
- Bad Lines

- Here is what you can do with the *error_bad_lines* option:

```
Val1,Val2,Val3
0,0,0
1,1,1
2,2,2,2
3,3,3
```

```
In: bad_dataset = pd.read_csv('a_loading_example_2.csv',
error_bad_lines=False)


bad_dataset
Skipping line 4: expected 3 fields, saw 4
Out:
   Val1  Val2  Val3
0     0     0     0
1     1     1     1
2     3     3     3
```

References and Further Reading

- A. Boschetti and L. Massaron, *Python Data Science Essentials*, Chapters 2
- D. Cielen and A. Meysman and M. Ali, *Introducing Data Science*, Chapter 2
- Pandas read_csv:
  - http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

Data Science

Thanks!