Lecture 15: Cleaning data ¶

In [1]:
```
%load_ext rmagic
```

Data gathering and data cleaning

The **preparation of data** for analysis can be slip in **two steps**:

- **data gathering**
- **data cleaning**

In your class project (http://nbviewer.ipython.org/urls/db.tt/M0xXxn2j), you'll create a notebook for each of these steps:

- NB1_data_gathering.ipynb (https://drive.google.com/a/berkeley.edu/file/d/0B5rZViyRAaZqOFRoc1YwOTQydms/edit?usp=sharing)
- NB2_data_cleaning.ipynb (https://drive.google.com/a/berkeley.edu/file/d/0B5rZViyRAaZqVC0wVlc5SWlKZU0/edit?usp=sharing)

Data gathering

various data sources (XML, jason, HTML, etc.) $\longrightarrow$ **raw data tables** (csv, xls, etc.)

This involves

- **storing the raw data** into the local file system
- **loading the raw data into R** (or Python) using **library** (or **modules**) corresponding to the raw data format
- **creating data frames** containing the raw data
- **saving the data in tabular format** (as csv, xls, etc.)

R packages for data gathering

You'll find here (http://cran.r-project.org/web/packages/available_packages_by_name.html) a list of all availabe R packages.

To install the R package (if not already installed), you'll invoke the command:

```
install.packages('package_name')
```

To use the package after installation, you'll inkove the command:

```
library('package_name')
```

- Package 'XML' (http://cran.r-project.org/web/packages/XML/index.html) for **HTML** and **XML** format processing
- Package 'scrapeR' (http://cran.r-project.org/web/packages/scrapeR/index.html) for webscraping (HTML and XML processing)
- Package 'RJSONIO' (http://cran.r-project.org/web/packages/RJSONIO/index.html) for **JSON** format processing
- Package 'xlsx' (http://cran.r-project.org/web/packages/xlsx/index.html) for **xls** (Excel spreadsheets) format processing
- Package 'httpRequest' (http://cran.r-project.org/web/packages/httpRequest/index.html) for requesting HTML pages from websites
- Packate 'RWeather' (http://cran.r-project.org/web/packages/RWeather/index.html) offers very convenient ways to retrieve weather data from various sources

Python packages for data gathering

You'll find here (https://pypi.python.org/pypi) a list of all availabe Python packages (or modules).

To install a Python package the best way is througth the **Canopy package installer**. Another simple way is to use the command:

```
easy_install 'package_name'
```

To use the package after installation, you'll inkove the command:

```
import package_name
```

- module lxml (http://lxml.de) for **HTML** and **XML** scraping
- module json (http://docs.python.org/2/library/json.html) for **JSON** format processing
- module pandas (http://pandas.pydata.org/pandas-docs/dev/io.html) can read directly a variety of format directly into data frames (jason, excel, etc.).
- module request (docs.python-requests.org) for requesting HTML pages from websites
- module pandas.io.data (http://pandas.pydata.org/pandas-docs/dev/remote_data.html) offers very convenient ways to obtain data from the internet (mostly financial)

Data cleaning

raw data tables $\longrightarrow$ clean data tables ready for analysis

This involves

- **removing duplicate observations (rows)**
- **selecting/producing only relevant variables (columns)**
  - by eliminating redundant or irrelevant variables
  - by creating new variables better suited for analysis (indicators, dummy variables)

- **keeping only clean variable values**
  - by removing rows and columns with too many **missing values** (NA)
  - by making sure that the **variable values are of the right type** (dates, prices, etc.)
  - by detecting and **removing errors** and **aberrant values**
  - by making sure that **one category correspond to only one value** for categorical variable
  - by possibly **rescaling quantitative variable values**

The end goal is to prepare

- the **right set of data tables**
- with **only clean values**
- with **only relevant variables**

ready for analysis.

Example 1: Cleaning variable types

Consider the following data on the most popular movies of all times:

In [2]:
```
url = 'http://www.stat.berkeley.edu/classes/s133/data/movies.txt'
!curl $url 2>/dev/null | head -4
```

```
rank|name|box|date
1|Avatar|$759.563|December 18, 2009
2|Titanic|$600.788|December 19, 1997
3|The Dark Knight|$533.184|July 18, 2008
```

The data **tabular** with separator '|', we can directly load it into a data frame without any preprocessing:

In [3]:
```
%%R -i url

df = read.delim(url, sep='|', header=T)
print(head(df))
```

```
  rank                                name      box              date
1    1                              Avatar $759.563 December 18, 2009
2    2                             Titanic $600.788 December 19, 1997
3    3                    The Dark Knight $533.184     July 18, 2008
4    4 Star Wars: Episode IV - A New Hope $460.998      May 25, 1977
5    5                             Shrek 2 $437.212      May 19, 2004
6    6         E.T. the Extra-Terrestrial $434.975     June 11, 1982
```

Looking at the box and date variables, we see a potential mismatch in types:

- the **dollar sign** in the box column seems to indicate that the box column is

represented by a **character vector** instead of a **numeric vector**

- the date column may also be represented by a **character vector** instead of a vector containing **date objects**

Let us check the variable types of this data frame.

A **data frame** is a **class**. Since **classes in R are just enhanced lists** (containing the vectors representing our variables, or columns), we can use the **list apply** function on our data frame `df` in the following way:

```
lapply(df, class)
```

which will return a list containing the classes of our data frame columns.

For a better ouput, we will futher construct a data frame out of the return value of `lapply`:

Digression: The `factor` class

In [4]:
```
%%R

modes = data.frame(lapply(df, class))
print(modes)
```

```
     rank   name    box   date
1 integer factor factor factor
```

We see here the new class factor (http://www.stat.berkeley.edu/classes/s133/factors.html), which is used by R to store **categorical variables** in the same way that Pandas used the **Categorical** class for the same purpose.

Factors are constructed out of regular vectors using the **class constructor**:

In [5]:
```
%%R

sex = factor(c('M','F', 'F', 'F', 'M'))
```

A `factor` object stores

- the **category values** as **vector of integers**
- the **category names** as a **character vector** accessible though the function

  levels(x)

In [6]:
```
%%R

print(levels(sex))
```

```
[1] "F" "M"
```

The `print` function displays the **category values as strings**.

In [7]:
```
%%R

print(sex)
```

```
[1] M F F F M
Levels: F M
```

The `cat` function displays the **category values as integers**.

In [8]:
```
%%R

cat(sex)
```

```
2 1 1 1 2
```

By default, the family of read functions **interprets character columns as factors**.

To prevent that, one needs to set the argument `stringsAsFactors` to `FALSE`:

In [9]:
```
%%R

df    = read.delim(url, sep='|', header=T, stringsAsFactors=F)
print(head(df))
```

```
  rank                                name      box              date
1    1                             Avatar $759.563 December 18, 2009
2    2                            Titanic $600.788 December 19, 1997
3    3                     The Dark Knight $533.184     July 18, 2008
4    4 Star Wars: Episode IV - A New Hope $460.998      May 25, 1977
5    5                              Shrek 2 $437.212      May 19, 2004
6    6         E.T. the Extra-Terrestrial $434.975     June 11, 1982
```

Now, character columns are interpreted as character vectors, but the types of the "box" column and the "date" column are still wrong:

In [10]:
```
%%R
modes = data.frame(lapply(df, class))

print(modes); cat('\n\n'); print(head(df))
```

```
     rank      name       box      date
1 integer character character character


  rank                                name      box              date
1    1                             Avatar $759.563 December 18, 2009
2    2                            Titanic $600.788 December 19, 1997
```

```
3    3                    The Dark Knight $533.184     July 18, 2008
4    4 Star Wars: Episode IV - A New Hope $460.998      May 25, 1977
5    5                             Shrek 2 $437.212      May 19, 2004
6    6         E.T. the Extra-Terrestrial $434.975     June 11, 1982
```

Digression: The `date` class

R has date class (http://www.stat.berkeley.edu/classes/s133/R-5a.html) used to represent **temporal data**.

One can create a date out of a *date string* in using the function:

```
as.Date(date_string, pattern)
```

where `pattern` is a string indicating how the date in `date`*`string`* *is formatted using the _date place holders*:

```
%d (day)
%m (month in decimal)
%B (month in letter)
%b (in abreviated)
%y (year: two digits)
%Y (year: four digits)
```

This function returns a `Date` object, on which we can perform numerical operations:

In [11]:
```
%%R

a = as.Date('December 18, 2009', '%B %d, %Y')
b = as.Date('January 29, 2013', '%B %d, %Y')

print(b-a)
```

```
Time difference of 1138 days
```

We can now correct the type in our date column:

In [12]:
```
%%R

df$date = as.Date(df$date, '%B %d, %Y')
print(head(df))
```

```
  rank                              name      box       date
1    1                            Avatar $759.563 2009-12-18
2    2                           Titanic $600.788 1997-12-19
3    3                    The Dark Knight $533.184 2008-07-18
4    4 Star Wars: Episode IV - A New Hope $460.998 1977-05-25
5    5                             Shrek 2 $437.212 2004-05-19
6    6         E.T. the Extra-Terrestrial $434.975 1982-06-11
```

Digression: Pattern maching and replacement

We still need to correct our "box" collum, since it contains character strings of the type:

```
$759.563
```

and we would like actual numerical values instead.

Unfortunately, we can not use the **conversion function**

```
as.numeric(x)
```

directly because of the presence of the dollar sign.

The return value would be in this case a vector of **NA values** (mising values).

In [13]:
```
%%R

print(head(as.numeric(df$box)))
```

```
[1] NA NA NA NA NA NA
```

R provides a collection of function (http://stat.ethz.ch/R-manual/R-devel/library/base/html/grep.html) to find and replace **regular expressions** in character vectors.

Here we only need to use the function

```
sub(pattern, replacement, x)
```

where

- `pattern` is regular expression to macht and replace
- `replacement` is the replacement regular expression

R uses the set of **extended regular expressions**. They are the same as we already studied in Unix:

Groups:

```
 ( ) delimits a group of characters
 |   means "either the group or character on the left or on the rig
th of |"
```

Ranges:

```
   .    means "any character"
 [...]  means "any character enclosed between the brackets"
 [^...] means "any character not enclosed between the brackets"
```

Modifiers:

```
 *      means "the previous character or group occurs zero or many
times"
 +      means "the previous character or group occurs one or many t
imes"
 ?      means "the previous character or group occurs zero or one t
ime"
 {n,m}  means "the previous character or group occurs between n or
m times"
 {n}    means "the previous character or group occurs exactly n tim
es"
 \      escape special characters
```

Positions:

```
 ^ means "at the beginning of the line"
 $ means "at the end of the line"
```

In our example, we need to remove the dollar sign from the box column.

Since the dollar sign has a special meaning as a regular expression, we will have to escape it:

In [14]:
```
%%R
pattern     = '\$'

replacement = ''

box_values = sub(pattern, replacement, df$box)

print(tail(box_values))
```

```
[1] " 57.114" " 57.059" " 57.042" " 52.823" " 56.615" " 52.581"
```

Now that the values has been stripped from the dollar sign, we can convert them into numbers, and replace the "box" column in our data frame:

In [15]:
```
%%R

df$box = as.numeric(box_values)
print(tail(df))
```

```
     rank                      name    box       date
995   995                 Beethoven 57.114 1992-04-03
996   996                     Annie 57.059 1982-05-21
997   997                   Beaches 57.042 1988-12-21
998   998      Message in a Bottle 52.823 1999-02-12
999   999 Resident Evil: Afterlife 56.615 2010-09-10
1000 1000    Kicking and Screaming 52.581 2005-05-13
```

Now we can save our data into a cleaned csv file for further use.

In [16]:
```
%%R

write.table(df, file='movies_cleaned.table', sep='|', row.names=F)
```

In [17]:
```
%%R

movies = read.delim('movies_cleaned.table', sep='|', header=T, stringsAsFa
ctors=F)

movies$date = as.Date(movies$date)

print(head(movies))
```

```
  rank                              name     box       date
1    1                            Avatar 759.563 2009-12-18
2    2                           Titanic 600.788 1997-12-19
3    3                   The Dark Knight 533.184 2008-07-18
4    4 Star Wars: Episode IV - A New Hope 460.998 1977-05-25
5    5                            Shrek 2 437.212 2004-05-19
6    6         E.T. the Extra-Terrestrial 434.975 1982-06-11
```

In [18]:
```
%%R

print(head(data.frame(lapply(movies, class))))
```

```
     rank      name     box date
1 integer character numeric Date
```

Digression: Quantitative variable check

One way to check if the values of a categorical variable are okay is to compute the statistic summary and look for aberrant means, medians, etc.

Let's do that with the "box" variable:

In [19]:
```
%%R

print(summary(movies$box))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  52.58   70.28   93.60  117.50  134.60  759.60
```

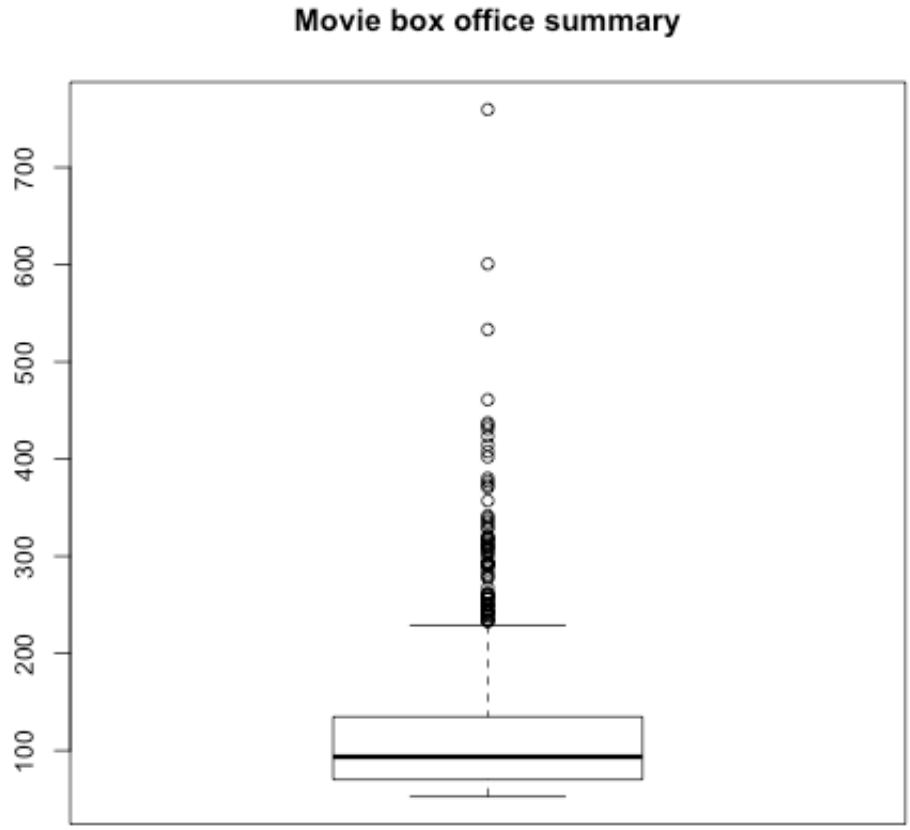
This seems okay, but still, we'd like to see how many movies are close to the max and min values.

Some erroneous outlier values may have crept in, and we may see that by plotting

- a **boxplot**
- a **histogram**

of the variable values to spot outliers visualy.

In [20]:
```
%%R

boxplot(movies$box, main="Movie box office summary")
```



It seems that there are quite a bunch of outliers. We may try to check the values by

- retrieving the movie names
- comparing their high success with our expectations

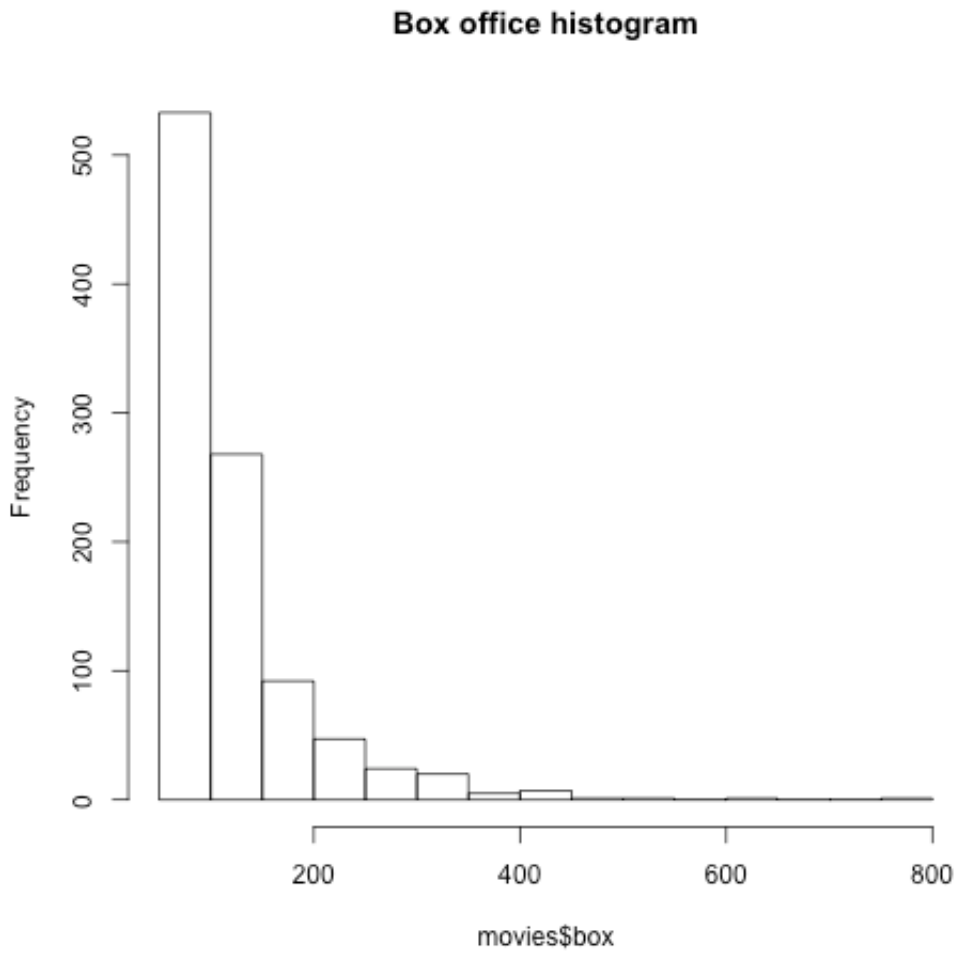
In [21]:
```
%%R

limit = 400

outliers = movies[movies$box > limit,]

print(dim(outliers))
print(outliers)
```

```
[1] 11  4
   rank                                        name     box       date
1     1                                      Avatar 759.563 2009-12-18
2     2                                     Titanic 600.788 1997-12-19
3     3                             The Dark Knight 533.184 2008-07-18
4     4         Star Wars: Episode IV - A New Hope 460.998 1977-05-25
```

```
5     5                                    Shrek 2 437.212 2004-05-19
6     6                 E.T. the Extra-Terrestrial 434.975 1982-06-11
7     7  Star Wars: Episode I - The Phantom Menace 431.088 1999-05-19
8     8 Pirates of the Caribbean: Dead Man's Chest 423.416 2006-07-07
9     9                                Toy Story 3 414.638 2010-06-18
10   10                                 Spider-Man 407.681 2002-05-03
11   11        Transformers: Revenge of the Fallen 402.077 2009-06-24
```

We can also plot an histogram of the variable:

In [22]:
```
%%R

hist(movies$box, main='Box office histogram')
```



The situation seems to correspond to what we expect: A lot of movies in the same range, and a few with enormous box office.

Adding variables

Since we have a temporal information, we maybe interested in patterns in the time variable.

We may want to add a variable for our analysis, for instance the day of the week that a movie was released. The function

```
weekdays(date)
months(date)
```

will return the corresponding day of the week for a data.

Let's use is to create another variable and add it as a factor:

In [23]:
```
%%R

days = c('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday
', 'Sunday')

movies$weekday = factor(weekdays(movies$date), levels=days)

print(tail(movies))
```

```
     rank                      name    box       date   weekday
995   995                 Beethoven 57.114 1992-04-03    Friday
996   996                     Annie 57.059 1982-05-21    Friday
997   997                   Beaches 57.042 1988-12-21 Wednesday
998   998      Message in a Bottle 52.823 1999-02-12    Friday
999   999 Resident Evil: Afterlife 56.615 2010-09-10    Friday
1000 1000    Kicking and Screaming 52.581 2005-05-13    Friday
```

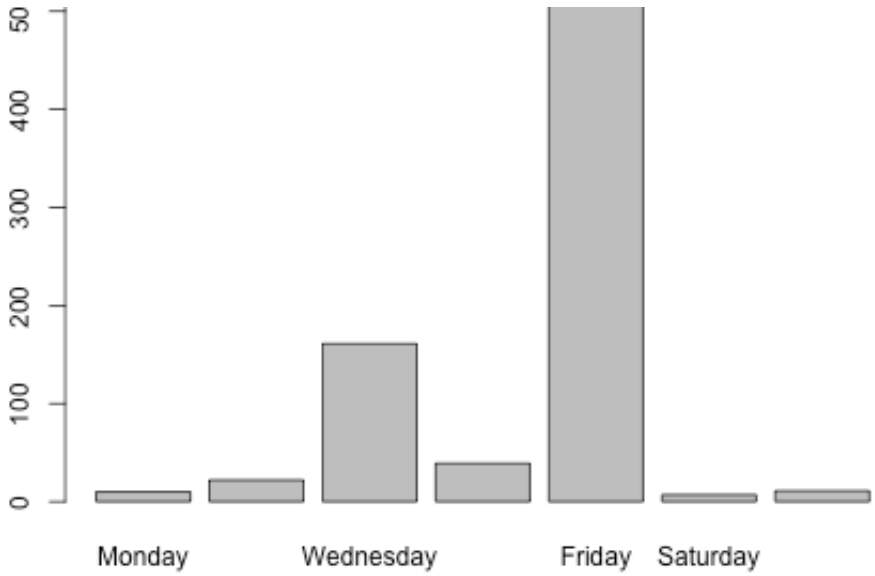
We can now compute a frequency table for this new categorical variable, and diplay it as a barplot:

In [24]:
```
%%R

release_days = table(movies$weekday)
print(release_days)
```

```
   Monday   Tuesday Wednesday  Thursday    Friday  Saturday    Sunday
       10        22       161        39       750         7        11
```

There seems to be a pattern emerging. Let see it with a plot:

In [25]:
```
%%R

barplot(release_days)
```



In []: