Programming and frameworks for ML

Data Cleaning with Python

About Me

Big Data Consultant at Indra / Big Data Lecturer

- More than 20 years of experience in different environments, technologies, customers, countries ...
- Passionate data and technology
- Enthusiastic Big Data world and NoSQL

Daniel Villanueva Jiménez

BigData Developer / Lecturer

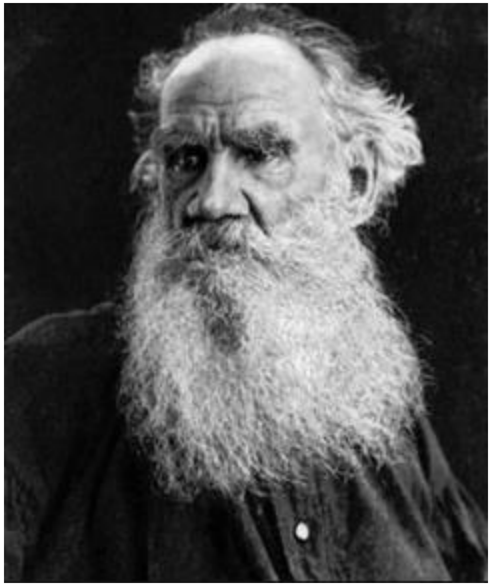
INDRA • Universidad Pontificia de Salamanca

daniel.villanueva@immune.institute

Index

Clean data

Happy families are all alike; every unhappy family is unhappy in its own way.



León Tolstói

Clean data

- A clean dataset is easy to analyze, model or visualize

Tidy datasets are all alike,
but every messy dataset is
messy in its own way.

Hadley Wickham

Definition

- A **unit of analysis** represents the entity being analysed in a study, and which contains similar features

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Definition

- An **observation** is data collected by observing behavior, events, or physical features.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Definition

- A **variable** is a property or feature that can change depending on certain factors (the person, the weather, the country, etc.)

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Definition

- A variable can take different **values,** which can be measured or observed.

country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

Rules

- Each **variable** must be in its own column
- Each **observation** should be in its own row
- Each **value** must have its own cell
- Each **unit of analysis** must be in its own table

variables observations values

Formats of a dataset

- We will display the same dataset in several formats

```
import pandas as pd
import numpy as np
```

```
table1 = pd.read_excel('tables.xlsx', 'table1')
table2 = pd.read_excel('tables.xlsx', 'table2')
table3 = pd.read_excel('tables.xlsx', 'table3')
table4a = pd.read_excel('tables.xlsx', 'table4a')
table4b = pd.read_excel('tables.xlsx', 'table4b')
table5 = pd.read_excel('tables.xlsx', 'table5')
table6 = pd.read_excel('tables.xlsx', 'table6')
```

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Formats of a dataset

- Variables such as values …

```
table2
```

	country	year	type	count
0	Afghanistan	1999	cases	745
1	Afghanistan	1999	population	19987071
2	Afghanistan	2000	cases	2666
3	Afghanistan	2000	population	20595360
4	Brazil	1999	cases	37737
5	Brazil	1999	population	172006362
6	Brazil	2000	cases	80488
7	Brazil	2000	population	174504898
8	China	1999	cases	212258
9	China	1999	population	1272915272
10	China	2000	cases	213766
11	China	2000	population	1280428583

Formats of a dataset

- A single column with several features ...

```
table3
```

	country	year	rate
0	Afghanistan	1999	745/19987071
1	Afghanistan	2000	2666/20595360
2	Brazil	1999	37737/172006362
3	Brazil	2000	80488/174504898
4	China	1999	212258/1272915272
5	China	2000	213766/1280428583

Formats of a dataset

- A feature separated into several columns…

```
table5
```

	country	century	year	rate
0	Afghanistan	19	99	745/19987071
1	Afghanistan	20	0	2666/20595360
2	Brazil	19	99	37737/172006362
3	Brazil	20	0	80488/174504898
4	China	19	99	212258/1272915272
5	China	20	0	213766/1280428583

Formats of a dataset

- A separate unit of analysis in several tables
- Values in columns instead of cells ...

```
table4a
```

	country	1999	2000
0	Afghanistan	745	2666
1	Brazil	37737	80488
2	China	212258	213766

```
table4b
```

	country	1999	2000
0	Afghanistan	19987071	20595360
1	Brazil	172006362	174504898
2	China	1272915272	1280428583

Index

- Introduction
- Widening tables
- Narrowing down tables
- Separating columns
- Joining columns
- Missing data

Widening tables

- Let's fix the 'variable as values' problem …

```
table2
```

	country	year	type	count
0	Afghanistan	1999	cases	745
1	Afghanistan	1999	population	19987071
2	Afghanistan	2000	cases	2666
3	Afghanistan	2000	population	20595360
4	Brazil	1999	cases	37737
5	Brazil	1999	population	172006362
6	Brazil	2000	cases	80488
7	Brazil	2000	population	174504898
8	China	1999	cases	212258
9	China	1999	population	1272915272
10	China	2000	cases	213766
11	China	2000	population	1280428583

Widening tables

- The **pivot_table**() function is used to distribute a key/value pair across the columns of the table

```
df
```

	column_A	column_B	column_C
0	C1	X	91
1	C1	Y	91
2	C2	X	204

```
df.pivot_table(index = "column_A",
         columns = "column_B",
         values = "column_C")
```

column_B	X	Y
column_A		
C1	91.0	91.0
C2	204.0	NaN

Widening tables

- We have to use the **first** aggregation function if the values are not numbers …

```
df
```

	column_A	column_B	column_C
0	C1	X	91
1	C1	Y	91
2	C2	X	204

```
df.pivot_table(index = "column_B",
         columns = "column_C",
         values = "column_A",
         aggfunc='first')
```

column_C	91	204
column_B		
X	C1	C2
Y	C1	NaN

Widening tables

- In the case of having a DataFrame with more than 3 columns ...

```
df
```

	column_A	column_B	column_C	column_D
0	C1	X	A	38
1	C1	X	C	67
2	C1	Y	A	50
3	C1	Y	C	59
4	C2	X	A	83
5	C2	X	B	95
6	C2	X	C	13

IMMUNE CODING INSTITUTE

Widening tables

```
df
```

	column_A	column_B	column_C	column_D
0	C1	X	A	38
1	C1	X	C	67
2	C1	Y	A	50
3	C1	Y	C	59
4	C2	X	A	83
5	C2	X	B	95
6	C2	X	C	13

```
df.pivot_table(index = ["column_B", "column_A"],
         columns = "column_C",
         values = "column_D")
```

	column_C	A	B	C
column_B	**column_A**			
X	**C1**	38.0	NaN	67.0
	C2	83.0	95.0	13.0
Y	**C1**	50.0	NaN	59.0

Widening tables

- We can reset the index of the result thanks to the **reset_index**() function

```
df
```

	column_A	column_B	column_C	column_D
0	C1	X	A	38
1	C1	X	C	67
2	C1	Y	A	50
3	C1	Y	C	59
4	C2	X	A	83
5	C2	X	B	95
6	C2	X	C	13

```
result = df.pivot_table(index = ["column_B", "column_A"],
        columns = "column_C",
        values = "column_D")
result
```

	column_C	A	B	C
column_B	**column_A**			
X	**C1**	38.0	NaN	67.0
	C2	83.0	95.0	13.0
Y	**C1**	50.0	NaN	59.0

```
result = result.reset_index()
result.columns.name = ''
result
```

	column_B	column_A	A	B	C
0	X	C1	38.0	NaN	67.0
1	X	C2	83.0	95.0	13.0
2	Y	C1	50.0	NaN	59.0

Exercise 1 (1/2)

- Load the following tables from the 'tables.xlsx' file

```
import pandas as pd

table1 = pd.read_excel('tables.xlsx', 'table1')
table2 = pd.read_excel('tables.xlsx', 'table2')
table3 = pd.read_excel('tables.xlsx', 'table3')
table4a = pd.read_excel('tables.xlsx', 'table4a')
table4b = pd.read_excel('tables.xlsx', 'table4b')
table5 = pd.read_excel('tables.xlsx', 'table5')
table6 = pd.read_excel('tables.xlsx', 'table6')
```

IMMUNE CODING INSTITUTE

Exercise 1 (2/2)

- Converts the dataset "table2" into a clean dataset, as seen in "table1"

```
table2
```

	country	year	type	count
0	Afghanistan	1999	cases	745
1	Afghanistan	1999	population	19987071
2	Afghanistan	2000	cases	2666
3	Afghanistan	2000	population	20595360
4	Brazil	1999	cases	37737
5	Brazil	1999	population	172006362
6	Brazil	2000	cases	80488
7	Brazil	2000	population	174504898
8	China	1999	cases	212258
9	China	1999	population	1272915272
10	China	2000	cases	213766
11	China	2000	population	1280428583

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 1 - Solution

```python
# Carga las siguientes tablas del fichero "tables.xlsx"

import pandas as pd

table1 = pd.read_excel('tables.xlsx', 'table1')
table2 = pd.read_excel('tables.xlsx', 'table2')
table3 = pd.read_excel('tables.xlsx', 'table3')
table4a = pd.read_excel('tables.xlsx', 'table4a')
table4b = pd.read_excel('tables.xlsx', 'table4b')
table5 = pd.read_excel('tables.xlsx', 'table5')
table6 = pd.read_excel('tables.xlsx', 'table6')

# Convierte el dataset "table2" en un dataset limpio

df = table2.pivot_table(index = ["country", "year"],
               columns = "type",
               values="count").reset_index()
df.columns.name = ''
df
```

Exercise 2

- Convert the dataset "table1" into another one showing the evolution of the population by years

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	country	1999	2000
0	Afghanistan	19987071	20595360
1	Brazil	172006362	174504898
2	China	1272915272	1280428583

Exercise 2 - Solution

```
# Convierte el dataset "table2" en otro mostrando la evolución de la población por años
df = table1.pivot_table(index = ["country"],
               columns = "year",
               values="population").reset_index()
df.columns.name = ''
df
```

Index

Narrowing down tables

- Let's fix the 'Value as Column' problem …

```
table4a
```

	country	1999	2000
0	Afghanistan	745	2666
1	Brazil	37737	80488
2	China	212258	213766

Narrowing down tables

- The **melt**() function takes multiple columns and collects them into a key/value pair

```
df
```

	column_A	column_B	column_C
0	C1	X	91
1	C1	Y	91
2	C2	X	204

```
df.melt(id_vars = 'column_A')
```

	column_A	variable	value
0	C1	column_B	X
1	C1	column_B	Y
2	C2	column_B	X
3	C1	column_C	91
4	C1	column_C	91
5	C2	column_C	204

Narrowing down tables

- We can 'reserve' as much columns as we want

```python
df
```

	column_A	column_B	column_C
0	C1	X	91
1	C1	Y	91
2	C2	X	204

```python
df.melt(id_vars = ['column_A', 'column_B'])
```

	column_A	column_B	variable	value
0	C1	X	column_C	91
1	C1	Y	column_C	91
2	C2	X	column_C	204

Narrowing down tables

- We can also specify the names of the variable and value columns with the **var_name** and **value_name** parameters

```
df
```

	column_A	column_B	column_C
0	C1	X	91
1	C1	Y	91
2	C2	X	204

```
df.melt(id_vars =['column_A', 'column_B'],
              var_name = 'variable_column',
              value_name = 'value_column')
```

	column_A	column_B	variable_column	value_column
0	C1	X	column_C	91
1	C1	Y	column_C	91
2	C2	X	column_C	204

Exercise 3

- Convert the dataset "table1" into a narrow table with the following shape:

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	country	year	column	data
0	Afghanistan	1999	cases	745
1	Afghanistan	2000	cases	2666
2	Brazil	1999	cases	37737
3	Brazil	2000	cases	80488
4	China	1999	cases	212258
5	China	2000	cases	213766
6	Afghanistan	1999	population	19987071
7	Afghanistan	2000	population	20595360
8	Brazil	1999	population	172006362
9	Brazil	2000	population	174504898
10	China	1999	population	1272915272
11	China	2000	population	1280428583

Exercise 3 - Solution

```
# Convierte el dataset "table1" en una tabla estrecha con la siguiente forma

table1.melt(id_vars=['country', 'year'],
            var_name = 'column',
            value_name = 'data')
```

Exercise 4

- Converts the datasets "table4a" and "table4b" into a clean dataset, as seen in "table1"

```
table4a
```

	country	1999	2000
0	Afghanistan	745	2666
1	Brazil	37737	80488
2	China	212258	213766

```
table4b
```

	country	1999	2000
0	Afghanistan	19987071	20595360
1	Brazil	172006362	174504898
2	China	1272915272	1280428583

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

IMMUNE CODING INSTITUTE

Exercise 4 - Solution

```
# Convierte los datasets "table4a" y "table4b" en un dataset limpio, tal y como se ve en "table1"

pd.merge(
    table4a.melt(id_vars="country",
                 var_name = 'year',
                 value_name = 'cases'),
    table4b.melt(id_vars="country",
                 var_name = 'year',
                 value_name = 'population')

)
```

IMMUNE CODING INSTITUTE

Index

- Introduction
- Widening tables
- Narrowing down tables
- Separating columns
- Joining columns
- Missing data

Separating columns

- We are to fix the 'Two values in one column' problem …

```
table3
```

	country	year	rate
0	Afghanistan	1999	745/19987071
1	Afghanistan	2000	2666/20595360
2	Brazil	1999	37737/172006362
3	Brazil	2000	80488/174504898
4	China	1999	212258/1272915272
5	China	2000	213766/1280428583

Separating columns

- Another common operation is to separate the value of a column into several columns ...

```
df
```

	column_A	column_B	column_C
0	C1	X	A1
1	C1	Y	A2
2	C2	X	B1
3	C2	Y	B2

```
def parse_value(s):
  return s[-1]

df["column_C1"] = df.column_C.map(lambda s: s[0])
df["column_C2"] = df.column_C.map(parse_value)
df = df.drop('column_C', axis = 1)
df
```

	column_A	column_B	column_C1	column_C2
0	C1	X	A	1
1	C1	Y	A	2
2	C2	X	B	1
3	C2	Y	B	2

Separating columns

- Another common operation is to separate the value of a column into several columns ...

```
df
```

	column_A	column_B	column_C
0	C1	X	A:1
1	C1	Y	A:2
2	C2	X	B:1
3	C2	Y	B:2

```
def parse_value(s, separator, chunk):
  return s.split(separator)[chunk]

df["column_C1"] = df.column_C.map(lambda s: s.split(':')[0])
df["column_C2"] = df.column_C.apply(parse_value, separator = ':', chunk = 1)
df = df.drop('column_C', axis = 1)
df
```

	column_A	column_B	column_C1	column_C2
0	C1	X	A	1
1	C1	Y	A	2
2	C2	X	B	1
3	C2	Y	B	2

IMMUNE CODING INSTITUTE

Exercise 5

- Converts the dataset "table3" into a clean dataset, as seen in "table1"
- Make sure the new columns have the int datatype

```
table3
```

	country	year	rate
0	Afghanistan	1999	745/19987071
1	Afghanistan	2000	2666/20595360
2	Brazil	1999	37737/172006362
3	Brazil	2000	80488/174504898
4	China	1999	212258/1272915272
5	China	2000	213766/1280428583

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

df = table3.copy()

Exercise 5 - Solution

```
# Convierte el dataset "table3" en un dataset limpio, tal y como se ve en "table1"

def parse_value(data, separator, chunk):
  return int(data.split(separator)[chunk])

df = table3.copy()
df['cases'] = df.rate.apply(parse_value, separator = '/', chunk = 0)
df['population'] = df.rate.apply(parse_value, separator = '/', chunk = 1)
df = df.drop('rate', axis = 1)
df

# Asegurate de que las nuevas columnas son de tipo entero
df.info()
```

Index

- Introduction
- Widening tables
- Narrowing down tables
- Separating columns
- Joining columns
- Missing data

Joining columns

- We are to fix the 'Same value in two diferent columns' problem ...

```
table7
```

	country	century	year	cases	population
0	Afghanistan	19	99	745	19987071
1	Afghanistan	20	0	2666	20595360
2	Brazil	19	99	37737	172006362
3	Brazil	20	0	80488	174504898
4	China	19	99	212258	1272915272
5	China	20	0	213766	1280428583

Joining columns

- There are times when we need to join two columns into one...

```
df
```

	column_A	column_B	column_C
0	C1	X	23
1	C1	Y	33
2	C2	X	10
3	C2	Y	34

```
df["column_AB"] = df.apply(lambda row: "%s:%s" % (row['column_A'], row['column_B']), axis = 1)
df = df.drop(['column_A', 'column_B'], axis = 1)
df.columns = ["column_AB", "colum_C"]
df
```

	column_AB	colum_C
0	23	C1:X
1	33	C1:Y
2	10	C2:X
3	34	C2:Y

Exercise 6

- Converts the dataset "table5" into a clean dataset, as seen in "table1"
- Make sure the columns are the right type

```
table5
```

	country	century	year	rate
0	Afghanistan	19	99	745/19987071
1	Afghanistan	20	0	2666/20595360
2	Brazil	19	99	37737/172006362
3	Brazil	20	0	80488/174504898
4	China	19	99	212258/1272915272
5	China	20	0	213766/1280428583

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Execise 6 - Solution

```python
# Convierte el dataset "table5" en un dataset limpio, tal y como se ve en "table1"

def parse_value(data, separator, chunk):
  return int(data.split(separator)[chunk])

def join_columns(row):
  return row['century'] + row['year']

df = table5.copy()
df['cases'] = df.rate.apply(parse_value, separator = '/', chunk = 0)
df['population'] = df.rate.apply(parse_value, separator = '/', chunk = 1)
df['year'] = df.apply(join_columns, axis = 1)
df = df.drop(['century', 'rate'], axis = 1)
df

# Asegurate de que las columnas del dataset tienen el tipo correcto
df.info()
```

Index

- Introduction
- Widening tables
- Narrowing down tables
- Separating columns
- Joining columns
- Missing data

Missing Data

We can have two different strategies to treat missing data:

- Removing the data that is null
- Filling the voids

Removing nulls

- The **dropna**() function removes all rows that contain any null value

```
df
```

	column_A	column_B	column_C	column_D
0	NaN	NaN	A	23.0
1	C1	NaN	A	33.0
2	C2	X	B	10.0
3	NaN	NaN	NaN	NaN

```
df.dropna()
```

	column_A	column_B	column_C	column_D
2	C2	X	B	10.0

Removing nulls

- The '**how**' parameter allows to specify if all the columns have to be null in order to delete the row

```
df
```

	column_A	column_B	column_C	column_D
0	NaN	NaN	A	23.0
1	C1	NaN	A	33.0
2	C2	X	B	10.0
3	NaN	NaN	NaN	NaN

```
df.dropna(how = 'all')
```

	column_A	column_B	column_C	column_D
0	NaN	NaN	A	23.0
1	C1	NaN	A	33.0
2	C2	X	B	10.0

Removing nulls

- The '**subset**' parameter allows you to specify the columns that must be set to zero to delete the row

```
df
```

	column_A	column_B	column_C	column_D
0	NaN	NaN	A	23.0
1	C1	NaN	A	33.0
2	C2	X	B	10.0
3	NaN	NaN	NaN	NaN

```
df.dropna(subset = ['column_A', 'column_B'], how = 'all')
```

	column_A	column_B	column_C	column_D
1	C1	NaN	A	33.0
2	C2	X	B	10.0

Filling in the voids

- The **fillna**() function replaces the nulls with the values specified in each column

```
df
```

	column_A	column_B	column_C	column_D
0	C1	NaN	A	23.0
1	C0	Y	A	33.0
2	NaN	X	B	NaN

```
values = {'column_A' : 'C1', 'column_B' : 'X'}
df.fillna(values)
```

	column_A	column_B	column_C	column_D
0	C1	X	A	23.0
1	C0	Y	A	33.0
2	C1	X	B	NaN

Filling in the voids

- We could fill in the nulls of a column with their mean value

```
df
```

	column_A	column_B	column_C	column_D
0	C1	NaN	A	23.0
1	C0	Y	A	33.0
2	NaN	X	B	NaN

```
df.fillna({'column_D' : df.column_D.mean()})
```

	column_A	column_B	column_C	column_D
0	C1	NaN	A	23.0
1	C0	Y	A	33.0
2	NaN	X	B	28.0

Filling in the voids

- **fillna**() provides a ‘**method**’ parameter to fill in the nulls with the previous value …

```
df
```

	column_A	column_B	column_C	column_D
0	C1	X	A	23
1	C0	Y	A	33
2	NaN	X	B	54

```
df.fillna(method = 'ffill')
```

	column_A	column_B	column_C	column_D
0	C1	X	A	23
1	C0	Y	A	33
2	C0	X	B	54

Exercise 7

- Turn the dataset table6 into a clean dataset

```
table6
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	NaN	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	NaN	2000	80488	174504898
4	China	1999	212258	1272915272
5	NaN	2000	213766	1280428583

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

Exercise 7 - Solution

```
# Convierte el dataset "table7" en un dataset limpio

table6.fillna(method='ffill')
```

Exercise 8

- Convert the dataset "table1" into a narrow table with the following shape:

```
table1
```

	country	year	cases	population
0	Afghanistan	1999	745	19987071
1	Afghanistan	2000	2666	20595360
2	Brazil	1999	37737	172006362
3	Brazil	2000	80488	174504898
4	China	1999	212258	1272915272
5	China	2000	213766	1280428583

	country	cases_1999	cases_2000	population_1999	population_2000
0	Afghanistan	745	2666	19987071	20595360
1	Brazil	37737	80488	172006362	174504898
2	China	212258	213766	1272915272	1280428583

THANKS FOR YOUR ATTENTION

Daniel Villanueva Jiménez
daniel.villanueva@immune.institute
@dvillaj