STATISTICS FOR DATA SCIENCE

Data Cleaning

D. Uma

Department of Computer Science and Engineering

STATISTICS FOR DATA SCIENCE

Data Cleaning

D. Uma

Department of Computer Science and Engineering

Data Quality

Suppose you have a dataset/database sitting in front of you, and I ask

"**Is it a good quality dataset/database?**"

This is **about the Data** themselves, not the system in use to access it.

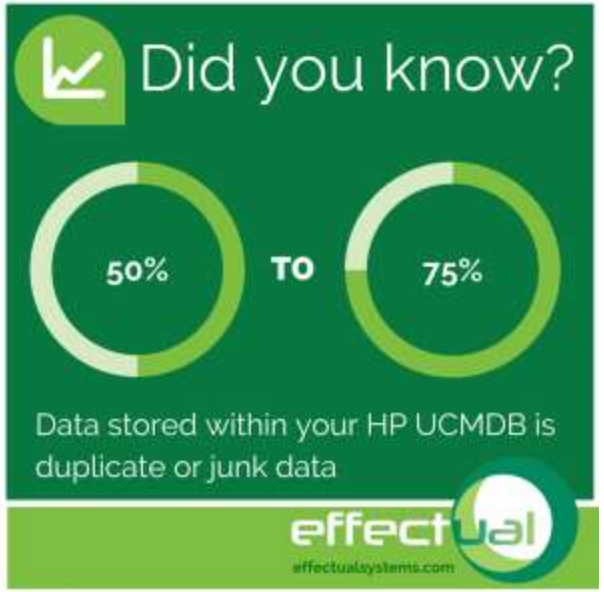
Data Quality

Data in the Real World is Dirty:
Lots of potentially incorrect data, e.g., instrument faulty, human or computer error, transmission error.

Incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data.

e.g., *Occupation*=“ ” (missing data)

Noisy: containing noise, errors, or outliers
e.g., *Salary*=“−10” (an error)



Sources: Trustinsights, Qgate, Lotame.com

Data Quality

Inconsistent: containing discrepancies in codes or names, e.g.,

Age="42", *Birthday*="03/07/2010"

Was rating "1, 2, 3", now rating "A, B, C"

discrepancy between duplicate records

Intentional (e.g., *disguised missing* data)

Jan. 1 as everyone's birthday?

Sources: Trustinsights, QGate

STATISTICS FOR DATA SCIENCE

Data Quality

Sources: Trustinsights, QGate

Data Quality

Improved data quality leads to **better decision making** across an organization.

The **more high-quality data** you have, the **more confidence** you can have in **your decisions**.

Good data **decreases risk** and can result in **consistent improvements** in **results.**

Data Cleaning

Data cleaning or cleansing is the **process** of **detecting and correcting** (or removing) corrupt or inaccurate records from a record set, table, or database.

It also refers to **identifying incomplete**, **incorrect, inaccurate or irrelevant parts** of the data and then replacing, modifying, or deleting the dirty or coarse data.

Source: towardsdatascience.com

Data Cleaning

- Makes the **data fit** for **purpose/plausible**
- **Reduces** the **negative impact of errors**
- **Improves** the **data quality**
- **Improves** the **quality of the outputs**

Source: www.youtube.com

PROCESS OF CLEANING DATA

- Detect
- Resolve
- Treat

- **Identify erroneous or suspicious data**

  - Graph or sort data - look at outliers

  - I have a **student who throws ten dice** and records the number of sixes. They recorded:

    (2, 0, 3, 12, 2, 0, 1, 1, 3, 1, 4).

  - What is wrong?
  - What do you think is the cause of it?

Source: www.youtube.com

- Consider the data points
  - 3, 4, 7, 4, 8, 3, 9, 5, 7, 6, **92**
  - **“92” is suspicious** - an **outlier**

- Outliers:
  - are potentially legitimate (correct)
  - can be data or model glitches
  - can be a data miners dream, for example, a highly profitable customer

- **Outlier - “departure from the expected”**

Source: www.questionpro.com

RESOLVE

- Deciding if **erroneous or suspicious data** should be **corrected** or amended
- Deciding on the action to "**treat**" the data

WHAT TO LOOK FOR ?

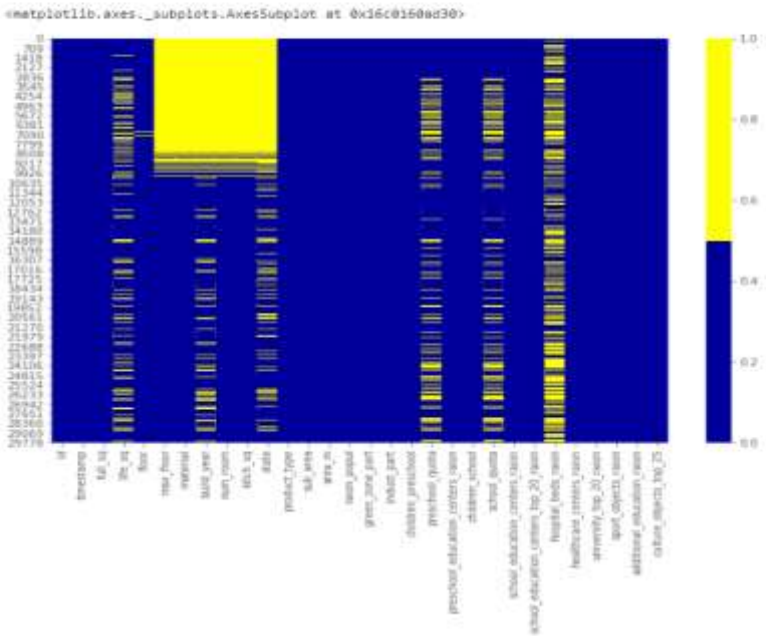
- **Non-response**
  - an item non response
  - Eg. missing data
- **Erroneous data**
  - Can negatively affect data and resulting quality
- **Suspicious data**

Data Cleaning

- **Missing Data**
- **Irregular Data (Outliers)**
- **Unnecessary Data** — Repetitive Data, Duplicates and more
- **Inconsistent Data** — Capitalization, Addresses and more

Source: towardsdatascience.com

Technique : Missing Data Heat map



The chart demonstrates the missing data patterns of the **first 30 features**.

The horizontal axis shows the feature name; the vertical axis shows the number of observations/rows.

The **yellow color** represents the missing data while the blue color otherwise.

Handling Missing Data

Technique : Missing Data Percentage List

When there are **many features** in the dataset, we can make a list of **missing data %** for each feature.

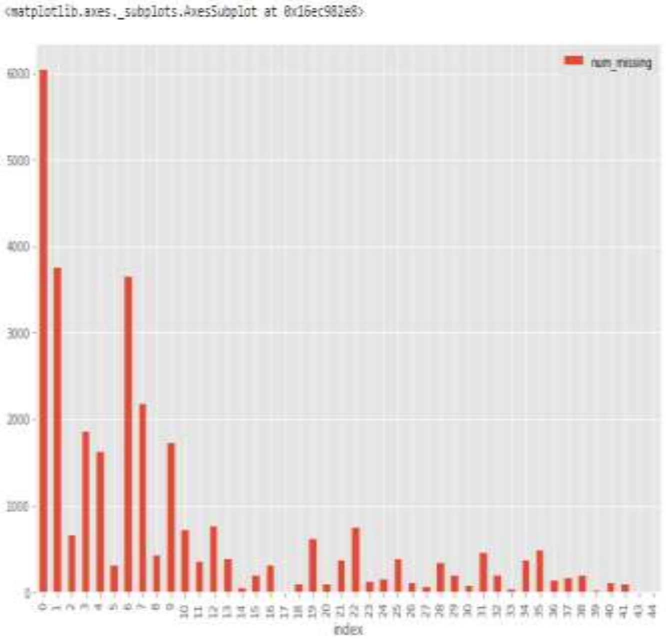
This produces a list showing the percentage of missing values for each of the features.

```
id - 0.0%
timestamp - 0.0%
full_sq - 0.0%
life_sq - 21.0%
floor - 1.0%
max_floor - 31.0%
material - 31.0%
build_year - 45.0%
num_room - 31.0%
kitch_sq - 31.0%
state - 44.0%
product_type - 0.0%
sub_area - 0.0%
area_m - 0.0%
raion_popul - 0.0%
green_zone_part - 0.0%
indust_part - 0.0%
children_preschool - 0.0%
preschool_quota - 22.0%
preschool_education_centers_raion - 0.0%
children_school - 0.0%
school_quota - 22.0%
school_education_centers_raion - 0.0%
school_education_centers_top_20_raion - 0.0%
hospital_beds_raion - 47.0%
healthcare_centers_raion - 0.0%
university_top_20_raion - 0.0%
sport_objects_raion - 0.0%
additional_education_raion - 0.0%
culture_objects_top_25 - 0.0%
```

Handling Missing Data

Technique : Missing Data Histogram

Missing data histogram is also a technique for when we have many features.



Source: towardsdatascience.com

Solution : Drop the Observation

In statistics, this method is called **the *listwise* deletion technique**.

In this solution, we drop the entire observation as long as it contains a missing value.

Only if we are sure that the **missing data** is **not informative**, we perform this. Otherwise, we should consider other solutions.

Solution : Drop the Feature

Similar to previous one, we only do this when we are confident that this feature doesn't provide useful information.

For example, from the missing data % list, we notice that hospital_beds_raion has a high missing value percentage of 47%. We may drop the entire feature.

Index	Age	Sex	Income
1	NA	M	NA
2	39	NA	75000
3	NA	NA	NA
4	28	F	50000
...	...	...	...
10000	18	F	NA

Source: towardsdatascience.com, Medium

Handling Missing Data

Solution : Impute the Missing

When the feature is a **numeric variable**, we can conduct missing data imputation.

We replace the missing values with the **average** or **median** value from the data of the same feature that is not missing.

When the feature is a **categorical variable**, we may impute the missing data by the **mode** (the most frequent value)

	col1	col2	col3	col4	col5
0	2	5.0	3.0	6	NaN
1	9	NaN	9.0	0	7.0
2	19	17.0	NaN	9	NaN

mean() →

	col1	col2	col3	col4	col5
0	2.0	5.0	3.0	6.0	7.0
1	9.0	11.0	9.0	0.0	7.0
2	19.0	17.0	6.0	9.0	7.0

Source: towardsdatascience.com

Solution : Replace the Missing

For **categorical features**, we can add a new category with a value such as "_**MISSING**_".

For **numerical features**, we can replace it with a particular value such as -**999**.

This way, we are still keeping the missing values as valuable information.

shutterstock.com • 1049623214

Irregular data (Outliers)

Outliers are data that is **distinctively different** from other observations.

They could be **real outliers** or **mistakes**.

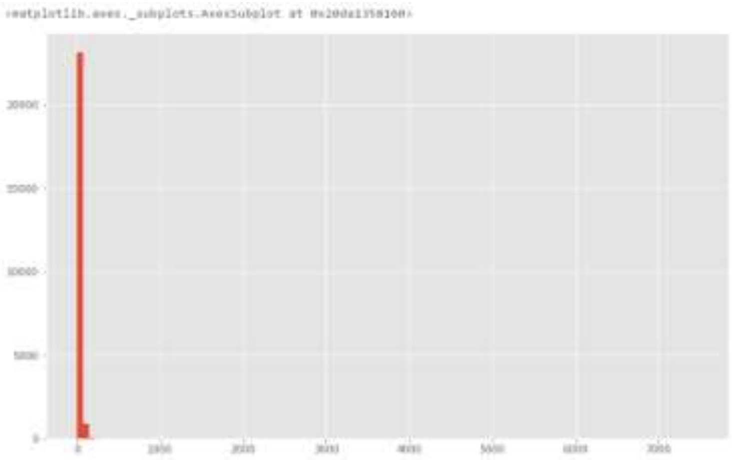
Source: towardsdatascience.com

Outliers

Technique : Histogram/Box Plot

When the feature is numeric, we can use a histogram and box plot to detect outliers.

The **data looks highly skewed** with the possible existence of outliers.

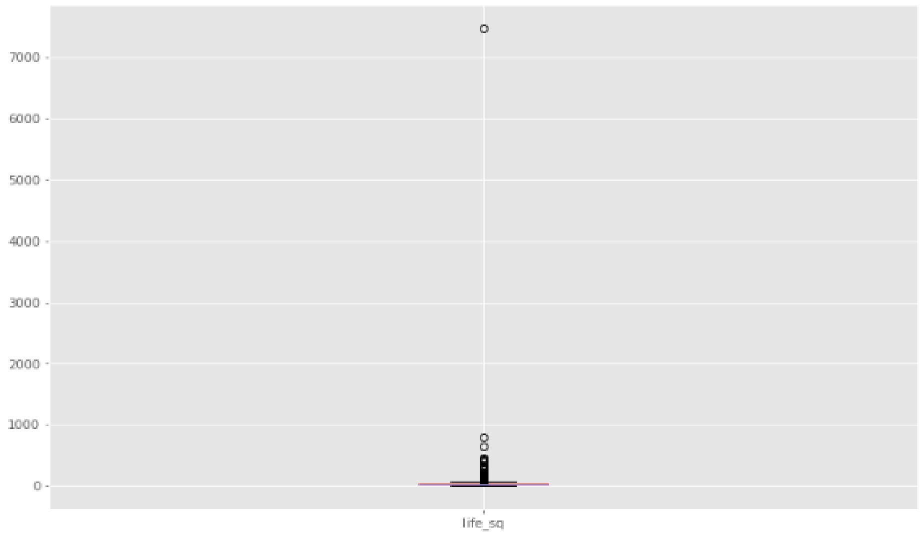


Source: towardsdatascience.com

To study the feature closer, let's make a **box plot.**

In this plot, we can see there is an outlier at a **value of over 7000**.

<matplotlib.axes._subplots.AxesSubplot at 0x20da34642b0>



Source: towardsdatascience.com

Technique : Descriptive Statistics

For **numeric features**, the outliers could be too distinct that the **box plot can't visualize** them.

Instead, we can look at their **descriptive statistics.**

For example, for the feature *life_sq* again, we can see that the **maximum value is 7478**, while the **75% quartile is only 43**.

The 7478 value is an **outlier**.

```
count    24088.000000
mean        34.403271
std         52.285733
min          0.000000
25%         20.000000
50%         30.000000
75%         43.000000
max       7478.000000
Name: life_sq, dtype: float64
```

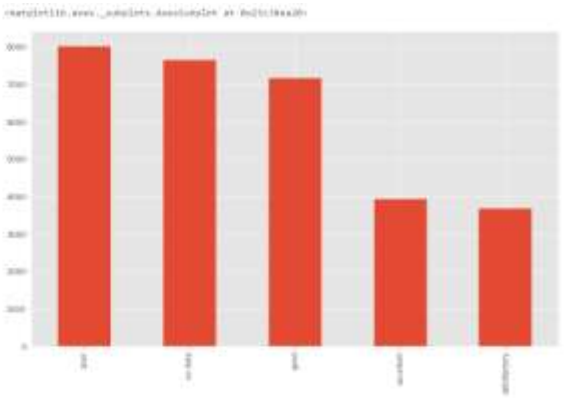
Outliers

Technique : Bar Chart

When the feature is **categorical,** we can use a bar chart to learn about its categories and distribution.

For example, the feature *ecology* has a reasonable distribution.

But if there is a category with **only one value** called “other”, then that would be an **outlier**.



Source: towardsdatascience.com

Unnecessary Data

After all the hard work done for missing data and outliers, let's look at **unnecessary data**, which is more straightforward.

The unnecessary data is when the data **doesn't add value**.

We cover three main types of unnecessary data due to different reasons.

Unnecessary Data

Unnecessary type : Uninformative / Repetitive

Sometimes one feature is uninformative because it has too many rows being the same value.

How to find out?

We can create a list of features with a high percentage of the same value.

For example, we specify below to show features with over 95% rows being the same value.

```
oil_chemistry_raion: 99.02858%
no     30175
yes      296
Name: oil_chemistry_raion, dtype: int64

railroad_terminal_raion: 96.27187%
no     29335
yes     1136
Name: railroad_terminal_raion, dtype: int64

nuclear_reactor_raion: 97.16780%
no     29608
yes      863
Name: nuclear_reactor_raion, dtype: int64

big_road1_1line: 97.43691%
no     29690
yes      781
Name: big_road1_1line, dtype: int64

railroad_1line: 97.06934%
no     29578
yes      893
Name: railroad_1line, dtype: int64

cafe_count_500_price_high: 97.25641%
0    29635
1      787
2       38
3       11
Name: cafe_count_500_price_high, dtype: int64

mosque_count_500: 99.51101%
0    30322
1      149
Name: mosque_count_500, dtype: int64

cafe_count_1000_price_high: 95.52689%
0    29108
1     1104
2      145
3       51
4       39
5       15
6        8
7        1
Name: cafe_count_1000_price_high, dtype: int64

mosque_count_1000: 98.08342%
0    29887
1      584
Name: mosque_count_1000, dtype: int64

mosque_count_1500: 96.21936%
0    29319
1     1152
Name: mosque_count_1500, dtype: int64
```

Source: towardsdatascience.com

Unnecessary Type :

- **Irrelevant**
- **Duplicates**

```
timestamp   full_sq  life_sq  floor  build_year  num_room  price_doc
2014-12-09  40       -999.0   17.0   -999.0       1.0      4607265     2
2014-04-15  134       134.0   1.0     0.0         3.0      5798496     2
2013-08-30  40       -999.0   12.0   -999.0       1.0      4462000     2
2012-09-05  43       -999.0   21.0   -999.0      -999.0    6229540     2
2013-12-05  40       -999.0   5.0    -999.0       1.0      4414080     2
2014-12-17  62       -999.0   9.0    -999.0       2.0      6552000     2
2013-05-22  68       -999.0   2.0    -999.0      -999.0    5406690     2
2012-08-27  59       -999.0   6.0    -999.0      -999.0    4506800     2
2013-04-03  42       -999.0   2.0    -999.0      -999.0    3444000     2
2015-03-14  62       -999.0   2.0    -999.0       2.0      6520500     2
2014-01-22  46        28.0    1.0     1968.0      2.0      3000000     2
2012-10-22  61       -999.0   18.0   -999.0      -999.0    8248500     2
2013-09-23  85       -999.0   14.0   -999.0       3.0      7725974     2
2013-06-24  40       -999.0   12.0   -999.0      -999.0    4112800     2
2015-03-30  41        41.0    11.0    2016.0      1.0      4114580     2
2013-12-18  39       -999.0   6.0    -999.0       1.0      3700946     2
2013-08-29  58        58.0    13.0    2013.0      2.0      5764128     1
            50        33.0    2.0     1972.0      2.0      8150000     1
            52        30.0    9.0     2006.0      2.0      10000000    1
2013-08-30  38        17.0    15.0    2004.0      1.0      6400000     1
Name: id, dtype: int64
```

There are 16 duplicates based on this set of key features.

Inconsistent : Capitalization

What to do?

To avoid this, we can put all letters to lower cases (or upper cases).

```
Poselenie Sosenskoe                       1776
Nekrasovka                                1611
Poselenie Vnukovskoe                      1372
Poselenie Moskovskij                       925
Poselenie Voskresenskoe                    713
                                          ...
Molzhaninovskoe                              3
Poselenie Kievskij                           2
Poselenie Shhapovskoe                        2
Poselenie Mihajlovo-Jarcevskoe               1
Poselenie Klenovskoe                         1
Name: sub_area, Length: 146, dtype: int64
```

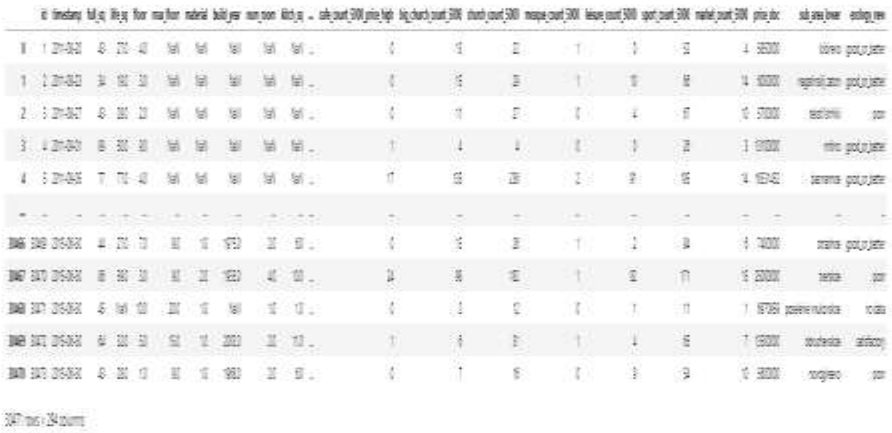
Source: towardsdatascience.com

Inconsistent Data

Inconsistent Type : Formats

Another standardization we need to perform is the data formats.

One example is to convert the feature from string to DateTime format.



How to find out?

The feature *timestamp* is in string format while it represents dates.

Inconsistent Data

What to do?

We can convert it and extract the date or time values by using the code below. After this, it’s easier to analyze the transaction volume group by either year or month.

```
2014      13662
2013       7978
2012       4839
2015       3239
2011        753
Name: year, dtype: int64

12      3400
4       3191
3       2972
11      2970
10      2736
6       2570
5       2496
9       2346
2       2275
7       1875
8       1831
1       1809
Name: month, dtype: int64
```

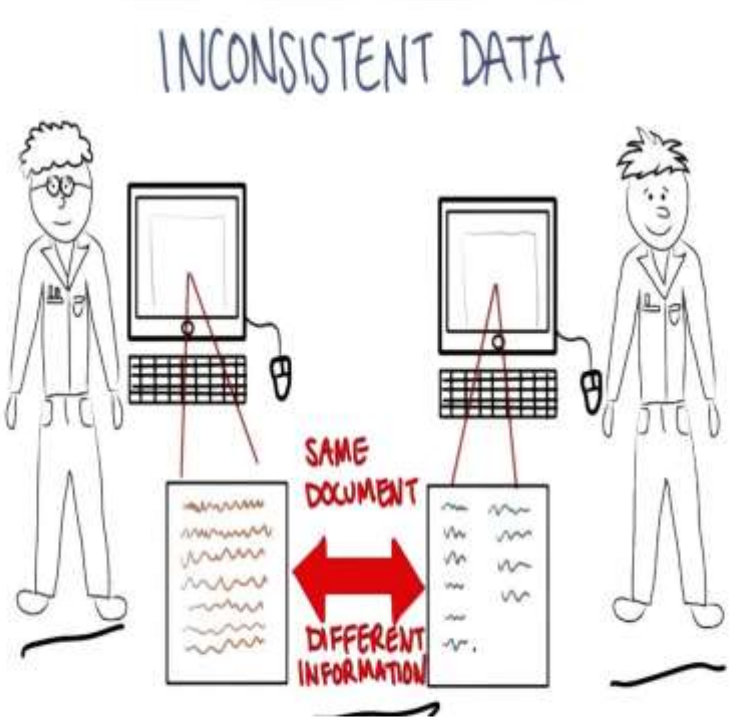
Source: towardsdatascience.com

Inconsistent Data

Inconsistent Type : Categorical Values

Inconsistent categorical values are the last inconsistent type we cover.

A categorical feature has a limited number of values. Sometimes there may be other values due to reasons such as typos.



How to find out?

For instance, the value of *city* was typed by mistakes as “torontoo” and “tronto”. But they both refer to the correct value “toronto”.

Source: towardsdatascience.com

Inconsistent Type : Addresses

The address feature could be a headache for many of us. Because people entering the data into the database often *don't* follow a standard format.

	address
0	123 MAIN St Apartment 15
1	123 Main Street Apt 12
2	543 FirSt Av
3	876 FIRst Ave.

How to find out?

We can find messy address data by looking at it. Even though sometimes we can't spot any issues, we can still run code to standardize them.

	address	address_std
0	123 MAIN St Apartment 15	123 main st apt 15
1	123 Main Street Apt 12	123 main st apt 12
2	543 FirSt Av	543 first ave
3	876 FIRst Ave.	876 first ave

Source: towardsdatascience.com

THANK YOU

D. Uma
Department of Computer Science and Engineering
umaprabha@pes.edu