NUS AI SUMMER EXPERIENCE 2019

DATA CLEANING

PAN BINBIN

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

OUTLINE

01 COMMON DATA PROBLEMS

02 KNOWING YOUR DATA

03 FIXING MISSING DATA

04 NORMALISING DATA

01 COMMON DATA PROBLEMS

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

CLEAN BEFORE USE

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

SOURCES OF DATA

- Collated data in files
- Application Programming Interface (API)
  - Provided by owner of data for you to download
  - Usually with proper format of data
- Scraping the web
  - Write a program to get data off website for yourself

CLEANING NEEDED

MISSING DATA
- Common for some fields with data missing

OUTLIERS
- Data values that are very far away from the “norm”
- May be actual data or an error

ERRORS
- Numeric data expected but string or character entered
- Wrong format of data

CLEANING NEEDED

DUPLICATE ROWS

- Observations that are redundant

WRONG FORMAT

- Not errors per se
- Need to transform into correct format before processing (e.g, date-time data)

02 KNOWING YOUR DATA

INSPECT THE DATA: WHAT IS IN THERE?

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Using data of passengers in Titanic
https://www.kaggle.com/c/titanic#tutorials

Training data
Competition: Which passenger will survive?

```python
#import the pandas library
import pandas as pd

df = pd.read_csv("Titanic/train.csv")
df.info()
```

Some missing data

Different data type

INSPECT THE DATA: FIRST FEW LINES

```python
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Always output first few lines of data to see what it looks like. Can see that there are a number of missing data in the “Cabin” column)

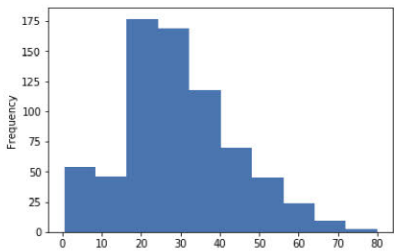
In this training dataset, we are told which passenger survived (“1”) and which did not (“0”).

Also, since there are a lot of data missing from the “Cabin” column, may be difficult to use as a predictor of who survived.

INSPECT THE DATA: VISUAL

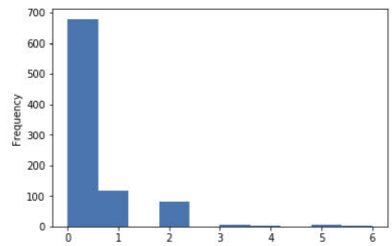
```
df.Age.plot('hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x67d45e5588>
```




```
df.Parch.plot('hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x67d7c3f8d0>
```



Plotting charts are a good way to understand your data and to see if there are any unusual data points. For example, there is nothing unusual about age data – nothing less than 0 or significantly more than 100.

- For Parch, the distribution is very different. Most are of value 0, with significant minority of value 1.

Parch: Number of children/parent aboard

INSPECT THE DATA: DUPLICATES

We only concerned about duplicates of a passenger record

```
#make a copy of original dataframe for this chcek

df_duplicateCheck = df.copy()
#create a new column.  Column will mark duplicates as True except for the first coccurence.

df_duplicateCheck["is_duplicated"]=df_duplicateCheck.duplicated('PassengerId')

#output those rows which are duplicates,i.e, where the output of the duplicate check returns True

df_duplicateCheck[df_duplicateCheck["is_duplicated"]==True]
```

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	is_duplicated

There are no duplicates

DEALING WITH DUPLICATES

First Method: Create a new dataframe without the duplicated data

```
df_nodup = df_duplicateCheck.loc[df_duplicateCheck["is_duplicated"]==False]
df_nodup  = df_duplicateCheck.drop_duplicates(subset=None, keep='first', inplace=False)
```

Copy the non-duplicated rows to df_nodup

Second Method: Drop duplicates from existing dataframe

```
df_duplicateCheck.drop_duplicates(subset=None, keep='first', inplace=True)
```

Drop the duplicated rows from the current dataframe

03 FIXING MISSING DATA

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF ISEM

MISSING DATA?

- DELETE ROWS WITH MISSING DATA
- DEFAULT VALUE FOR MISSING DATA
- DELETE COLUMNS WITH HIGH INCIDENCE OF MISSING DATA

DELETE ROWS WITH MISSING DATA?

- Missing data can mess up the analysis.
- Best to deal with it as soon as possible.
- One way to remove observations (rows) with missing data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

Before dropping

```
df1 = df.dropna().copy()
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 183 entries, 1 to 889
Data columns (total 12 columns):
PassengerId    183 non-null int64
Survived       183 non-null int64
Pclass         183 non-null int64
Name           183 non-null object
Sex            183 non-null object
Age            183 non-null float64
SibSp          183 non-null int64
Parch          183 non-null int64
Ticket         183 non-null object
Fare           183 non-null float64
Cabin          183 non-null object
Embarked       183 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 18.6+ KB
```

After dropping

- A row is dropped as long as there is one missing value in any of the columns
- Resulted in reduction from 891 observations to 183 observations

```
df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

Before dropping, first five rows

```
df1.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C103	S

After dropping, first five rows
Note those rows with Cabin = NaN are deleted and no longer in the new dataframe that contains the cleaned data

DELETE ROWS WITH SPECIFIED MISSING DATA?

- Avoid eliminating so many observations d (708 eliminated)
- Deal only with missing data on columns that are of interest
- Age, Cabin and Embarked has missing data.
- You think Age has some effect on survival but not Cabin and Embarked.
- So only deal with observations with missing data in Age

```
df_noMissingAge = df.dropna(subset=['Age']).copy()
df_noMissingAge.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 714 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    714 non-null int64
Survived       714 non-null int64
Pclass         714 non-null int64
Name           714 non-null object
Sex            714 non-null object
Age            714 non-null float64
SibSp          714 non-null int64
Parch          714 non-null int64
Ticket         714 non-null object
Fare           714 non-null float64
Cabin          185 non-null object
Embarked       712 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 72.5+ KB
```

Only observations with missing Age data dropped

714 observations left. Much more than the previous 183 observations

dropna(subset = ['Age'])
1. not dropped
2. not dropped
5. dropped
31. dropped
61. not dropped

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
31	32	1	1	Spencer, Mrs. William Augustus (Marie Eugenie)	female	NaN	1	0	PC 17569	146.5208	B78	C
61	62	1	1	Icard, Miss. Amelie	female	38.0	0	0	113572	80.0000	B28	NaN

DEFAULT VALUES WITH MISSING DATA?

- You may want to replace the missing data with default values instead of deleting the observations.
- For string data, you may want to replace it with the most likely choice for that group or even a blank" "
- For numeric data, common replacement is with the mean of that data, maximum or minimum depending on the context
- In this example, we want to replace the missing value of Age with a default value of the mean of the Age

	Name	Age
0	Braund, Mr. Owen Harris	22.0
5	Moran, Mr. James	NaN
17	Williams, Mr. Charles Eugene	NaN
19	Masselmani, Mrs. Fatima	NaN
26	Emir, Mr. Farred Chehab	NaN

Data before the replacement

```
df_AgeMissingAv.Age = df.Age.fillna(df.Age.mean())
df_AgeMissingAv.loc[[0,5,17,19,26],['Name','Age']]
```

	Name	Age
0	Braund, Mr. Owen Harris	22.000000
5	Moran, Mr. James	29.699118
17	Williams, Mr. Charles Eugene	29.699118
19	Masselmani, Mrs. Fatima	29.699118
26	Emir, Mr. Farred Chehab	29.699118

Non-missing Age value remain

All previously missing Age values set to default mean value

Same set of data after the replacement of missing age data with average value of the age

DROPPING COLUMNS FROM DATA?

- You may just want to drop the Cabin column from the dataset as it has too many NAs.
- You are unlikely to use it for analysis because of the lack of data

axis = 1 denotes that a column is being dropped.
The default is axis=0 which denotes rows

```
df_dropCabin = df_noMissingAge.drop('Cabin',axis=1).copy()
df_dropCabin.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	S

The Cabin column is not present anymore

04 NORMALISING DATA

NORMALISING DATA

- Column names meaningless
- Wrong data format

RENAMING COLUMN NAMES

- Sometimes column names are assigned by the program (e.g. col1, col2). Difficult to understand
- Simple .rename function to change to column name that is more meaningful

This operation makes the changes in the dataframe

```
df_noMissingAge.rename(columns={'SibSp':'Siblings', 'Parch':'ParentChild'},inplace=True)
df_noMissingAge.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	Siblings	ParentChild	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

The changed column names

REFORMATTING DATA TYPES (1/2)

- Easier to explain by using an example.
- Look at the dataset containing incidents of taser use.
- The Excel file contains 69 columns. Here we only read in the columns that we are interested in.

- We only want to work with these five columns
- Explicitly define the columns we want to read into our dataframe

```
#import pandas
import pandas as pd

#read in data from excel
#read in only 3 columns that we are interested in
stuns = pd.read_excel("E:/Python/Jupyter/Data_Cleaning/2015 Reported Taser Data.xlsx")[["Incident Case Number","Height","Weight","Date of Incident", "Time of Incident"]]
```


REFORMATTING DATA TYPES (2/2)

```
stuns.head()
```

	Incident Case Number	Height	Weight	Date of Incident	Time of Incident
0	NaN	NaN	NaN	NaT	NaN
1	2015-055	5'10"	220	2015-01-01	15:13:00
2	15-19	5'8"	150	2015-01-01	03:48:00
3	15-00010	5'10"	175	2015-01-01	00:32:00
4	1500006763	5'10"	150	2015-01-04	11:41:00

- This is how the data appears in dataframe
- No way to tell what is the type of data

```
stuns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 611 entries, 0 to 610
Data columns (total 5 columns):
Incident Case Number    610 non-null object
Height                  609 non-null object
Weight                  608 non-null object
Date of Incident        610 non-null datetime64[ns]
Time of Incident        609 non-null object
dtypes: datetime64[ns](1), object(4)
memory usage: 23.9+ KB
```

- .info() method - type of data
- "Date of Incident" - (datetime64)
- The rest - object type. This is string data.
- String data is the wrong data type for some of the columns.
- "Weight" and "Height" should be numeric
- "Time of Incident" should be datetime64

REFORMATTING DATA TYPES TO NUMERIC AND DATETIME TYPE

```
stuns["Weight"] = pd.to_numeric(stuns["Weight"], errors = 'coerce')
```

- "coerce" change data that cannot be converted to NaN
- Absent this, errors will be raised if non-convertible data is encountered and execution halted

```
from datetime import datetime
stuns['incidentTime'] = pd.to_datetime(stuns['Time of Incident'], format='%H:%M:%S' errors='coerce' )
```

- From http://strftime.org

%H	Hour (24-hour clock) as a zero-padded decimal number.	07
%-H	Hour (24-hour clock) as a decimal number. (Platform specific)	7
%I	Hour (12-hour clock) as a zero-padded decimal number.	07
%-I	Hour (12-hour clock) as a decimal number. (Platform specific)	7
%p	Locale's equivalent of either AM or PM.	AM
%M	Minute as a zero-padded decimal number.	06
%-M	Minute as a decimal number. (Platform specific)	6
%S	Second as a zero-padded decimal number.	05
%-S	Second as a decimal number. (Platform specific)	5

- Know what your data is like

REFORMATTING HEIGHT DATA TO NUMERIC

	Incident Case Number	Height	Weight	Date of Incident	Time of Incident
0	NaN	NaN	NaN	NaT	NaN
1	2015-055	5'10"	220	2015-01-01	15:13:00
2	15-19	5'8"	150	2015-01-01	03:48:00
3	15-00010	5'10"	175	2015-01-01	00:32:00
4	1500006763	5'10"	150	2015-01-04	11:41:00

- COMPLICATION: UNLIKE WEIGHT WHICH LOOKS LIKE NUMBERS, HEIGHT LOOKS LIKE STRINGS
- 5' 10'' – MUST IDENTIFY LOCATION OF ' AND " TO EXTRACT THE NUMBERS
- IDENTIFY NUMBERS AS FEET AND AS INCHES
- LEARN TO USE REGULAR EXPRESSIONS TO CONVERT STRINGS THAT MATCH KNOWN PATTERN

SEARCHING FOR PATTERNS

You must understand how the data in your dataset is represented

How the data may be represented	Our desired data representation (inches)
5′ 10″	70
5′ 10	70
5″	60
Any other that does not fit above three patterns (e.g. 5 10, 5′ 100)	None

REGULAR EXPRESSION (re) TO SEARCH FOR PATTERNS

- re.match(pattern,string,flags)
- pattern: regular expression to be matched
- string: searched to match pattern at beginning of pattern
- flags: modifiers (e.g, ignore cases)

REGULAR EXPRESSION

5'10"
5'10
5'

Everything inside parenthesis should be extracted as a group that we can access later

```
re.match(r"(?P<feet>[0-9]+)'(?P<inches>[0-9]{0,2}?)[\"]?\Z", prime_str)
```

Rest of expressio n in raw notation

The ?P<feet> stores the group under the name "feet"

[0-9]+ is the actual pattern to match. The numerals 0-9 in the bracket are valid match. The + signifies that there should at least be one of these values

{0,2}?means to look for up to 2 numerals; not more.

\Z matches the end of the string. No trailing characters are allowed, e.g., 5'6"LOL

prime_str is the string to be matched.

Reference: Regular expression HOWTO
https://docs.python.org/3/howto/regex.html#regex-howto

```
# import the re module
import re

#Convert prime-notation height to inches

   # Convert prime-notation height to inches
def inches(prime_str):
    ## Wrap the function in try/except
   try:
      # Extract feet and inches from the string,
     result = re.match(r"(?P<feet>[0-9]+)'(?P<inches>[0-9]{0,2}?)[\"]?\Z", prime_str)

      # Access the extracted values using the .group() method
      feet = int(result.group("feet"))
      if result.group("inches") == "":
         inches = 0
      else:
         inches = int(result.group("inches"))


   except:
      # This code will only be called if the code within the try: block
      # throws an exception
      return None

   # We won't get to this point unless the code in the try block was successful
   # Convert feet and inches to inches
    return feet * 12 + inches
```

USING APPLY FUNCTION TO CONVERT HEIGHT

```
stuns['Heights_numeric'] = stuns['Height'].apply(inches)
stuns.head(10)
```

	Incident Case Number	Height	Weight	Date of Incident	Time of Incident	Heights_numeric
0	NaN	NaN	NaN	NaT	NaN	NaN
1	2015-055	5'10"	220.0	2015-01-01	15:13:00	70.0
2	15-19	5'8"	150.0	2015-01-01	03:48:00	68.0
3	15-00010	5'10"	175.0	2015-01-01	00:32:00	70.0
4	1500006763	5'10"	150.0	2015-01-04	11:41:00	70.0
5	2015-0379	6'	153.0	2015-01-05	12:52:00	72.0
6	2015-341	5'8"	260.0	2015-01-06	18:07:00	68.0
7	15-16-AR	6'2"	200.0	2015-01-06	23:54:00	74.0
8	1500001035	5'10"	177.0	2015-01-07	09:00:00	70.0
9	15-000071	5'2"	120.0	2015-01-08	10:18:00	62.0

The new variable that is a numeric variable showing the height in inches

PAN BINBIN

BINBIN.PAN@NUS.EDU.SG

NUS AI SUMMER EXPERIENCE 2019

DATA CLEANING