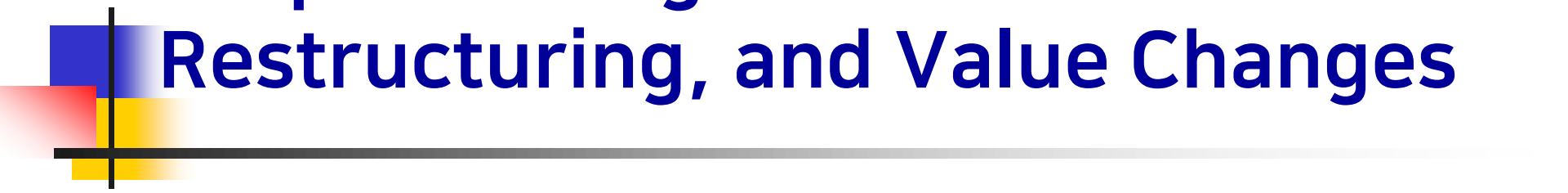


Data Science: Preprocessing – Data Restructuring, and Value Changes

Won Kim & Woong-Kee Loh

2024

Roadmap: Big Data End-to-End Process

- 1. Objective Setting
- 2. Data Curation
- 3. Data Inspection
- 4. Data Preprocessing
- 5. Data Analysis
- 6. Evaluation
- 7. Deployment

Data Preprocessing

- Also called “Data Preparation”
- 80% of the time and efforts needed for the entire end-to-end process.
- Include four major steps.
- Some require human experts who use software tools.
- Some require understanding of basic statistics.

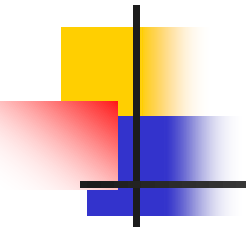
Acknowledgments

- https://www.slideshare.net/TonyNguyen197/data-preprocessing-61425960
- https://slideplayer.com/slide/12686164/
- http://www.life.illinois.edu/ib/372/labs2010/IB372_FA10_Lab01_Intro_statistics_presentation.ppt
- https://www.slideshare.net/marinasantini1/lecture-2-preliminaries-understanding-and-preprocessing-data

Roadmap: Data Preprocessing

- Data Restructuring
- Data Value Changes
- Feature Engineering
- Data Reduction

- ** Just as software development, the steps above are not strictly sequential, and also they iterate.



Data Restructuring

- Table Decomposition
  - horizontal, vertical
- Table Merge
  - denormalization

Note: Terminology

- Data science (machine learning) is an interdisciplinary field, with database, machine learning, mathematics, economics, medicine, etc. all contributing.
- As a consequence, often there are 3-4 terms for the same concept.
- Those studying data science must be familiar with all these terms; we will use different terms interchangeably in this course.
- table=Excel worksheet=dataset=Data Frame
- table row=record=sample=observation
- table column=attribute=feature=variable

Example

- In the following “People” table, the “Name” column is the primary key (by which a record can be uniquely identified.)

Name	Age	Job	Salary	Car	Hobby
Peter	25	engineer	4500	Avante	K-pop
Paul	30	marketing	3000	Matiz	game
Mary	35	marketing	5000	Grandeur	music
Neil	27	SW engineer	6000	BMW	golf

Horizontal Decomposition

- Decompose (split) a table into two or more tables (keeping all the columns)
  - A condition is used for grouping the rows.
- * (This is similar to one of the techniques used in Data Reduction. *)

Example

- Split a table into two (using Age<30 as the splitting condition)
- One or both tables may be used for analysis

Name	Age	Job	Salary	Car	Hobby
Peter	25	engineer	4500	Avante	K-pop
Neil	27	SW engineer	6000	BMW	golf

Name	Age	Job	Salary	Car	Hobby
Paul	30	marketing	3000	Matiz	game
Mary	35	marketing	5000	Grandeur	music

Vertical Decomposition

- Decompose (split) a table into two or more tables (dropping some columns, but keeping all the rows)
- The split tables all retain the same primary key.
- * (This is similar to one of the techniques used in Data Reduction. *)

Example

- Split a table into two, and use only one for the analysis
- One or both tables may be used for analysis

Name	Age	Job	Salary
Peter	25	engineer	4500
Paul	30	marketing	3000
Mary	35	marketing	5000
Neil	27	SW engineer	6000

Name	Car	Hobby
Peter	Avante	K-pop
Paul	Matiz	game
Mary	Grandeur	music
Neil	BMW	golf

Table Merge

- Combining two or more tables into a single table.
  - Machine learning algorithms can take only one dataset (table). If there is more than one dataset, they must be merged into one.
- The tables must have a shared column in the form of a primary key or a foreign key.

Example

- Merge the following two tables into a single table.
- The two tables have a shared column: Name

Name	Age	Job	Salary
Peter	25	engineer	4500
Paul	30	marketing	3000
Mary	35	marketing	5000
Neil	27	SW engineer	6000

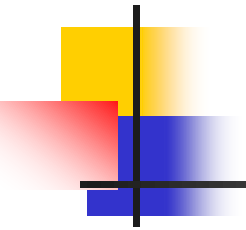
Name	Car	Hobby
Peter	Avante	K-pop
Paul	Matiz	game
Mary	Grandeur	music
Neil	BMW	golf

Roadmap: Data Preprocessing

- Data Restructuring
- Data Value Changes
- Feature Engineering
- Data Reduction

Roadmap: Data Value Changes

- Cleaning dirty data
- Text preprocessing
- Data discretization
- Data normalization/standardization
- Encoding for data mining algorithms



Roadmap: Cleaning Dirty Data

- Introduction
- Missing Data
- Wrong Data
- Unusable Data
- Outliers

Dirty Data

- Many (30+) types
  - missing data
  - wrong data (not missing, but)
  - unusable data (not missing, not wrong, but)
- Important Note
  - The datasets you will download from the Internet for homework contain some dirty data, but they are pretty well sanitized.
  - Don't let this make you think dirty data is not a serious problem. When datasets are first created and explored, they are quite problematic.

Examples of Dirty Data

- missing data
- wrong data
- unusable data (inconsistent, redundant, non-standard)

name	age	height	hobby
John	27	180	
John	277	180	game
John	27	1.8	game
John	25	175	music

Roadmap: Cleaning Dirty Data

- Introduction
- Missing Data
- Wrong Data
- Unusable Data
- Outliers

Missing Data (1/2)

- Presence of lots of missing data may make the entire dataset unusable
  - (because of the statistical significance issue)
  - (e.g., Sampling only 10 people after election cannot give a correct prediction of who will win.)
- Many database systems treat missing data as having NULL value.
- Two types of missing data
  - data missing because there really is no value (i.e., Not Applicable); e.g., “income” for a child.
  - data missing because the user purposely or accidentally provides no value; e.g., “income”

Missing Data (2/2)

- Python NumPy and other libraries focus mostly on missing data.
- Ways to deal with missing data
  - drop it from the dataset
  - replace it with some default value
    - median, maximum, minimum, or some default value
    - (e.g., if the “age” value for “John” in a “Students” table is missing, the “median” age of all Students may be used.)

Illustration: Ignore Missing Data

	Price	Country	Reliability	Mileage	Type	Weight	Disp.	HP
Hyundai Sonata 4	9999	Korea	NA	23	Medium	2885	143	110
Mazda 929 V6	23300	Japan	5	21	Medium	3480	180	158
Nissan Maxima V6	17899	Japan	5	22	NA	3200	180	160
Oldsmobile Cutlass Ciera 4	13150	USA	2	21	Medium	2765	151	110
Oldsmobile Cutlass Supreme V6	14495	NA	1	21	Medium	3220	189	135
Toyota Cressida 6	21498	Japan	3	23	Medium	3480	180	190
Buick Le Sabre V6	16145	USA	3	23	Large	3325	231	165
Chevrolet Caprice V8	14525	USA	1	18	Large	3855	305	170
Ford LTD Crown Victoria V8	17257	USA	3	20	Large	3850	302	150
Chevrolet Lumina APV V6	13995	USA	NA	18	Van	3195	151	110
Dodge Grand Caravan V6	15395	USA	3	18	Van	3735	202	150

- Ignore the feature (column) (vertical decomposition)
  - pro: simple, typically not biased
  - con: may be a very useful feature
- Ignore the sample (row) (horizontal decomposition)
  - pro: simple, all features are kept
  - con: removed samples may be biased
  - con: data may become small

Illustration: Replace Missing Data (1/2)

- Simple data imputation: mean, median, mode

	Price	Country	Reliability	Mileage	Type	Weight	Disp.	HP
Hyundai Sonata 4	9999	Korea	NA	23	Medium	2885	143	110
Mazda 929 V6	23300	Japan	5	21	Medium	3480	180	158
Nissan Maxima V6	17899	Japan	5	22	NA	3200	180	160
Oldsmobile Cutlass Ciera 4	13150	USA	2	21	Medium	2765	151	110
Oldsmobile Cutlass Supreme V6	14495	NA	1	21	Medium	3220	189	135
Toyota Cressida 6	21498	Japan	3	23	Medium	3480	180	190
Buick Le Sabre V6	16145	USA	3	23	Large	3325	231	165
Chevrolet Caprice V8	14525	USA	1	18	Large	3855	305	170
Ford LTD Crown Victoria V8	17257	USA	3	20	Large	3850	302	150
Chevrolet Lumina APV V6	13995	USA	NA	18	Van	3195	151	110
Dodge Grand Caravan V6	15395	USA	3	18	Van	3735	202	150

Mean (Reliability): (5+5+2+1+3+3+1+3+3)/9 = **2.88**
Median (Reliability): 1 1 2 3 **3** 3 3 5 5
Mode (Country): **USA = 6**, Japan = 3, Korea = 1.

Illustration: Replace Missing Data (1+/2)

- Use attribute mean, median or mode based on similarity (same class, nearest neighbors, etc.)
- Example: same class

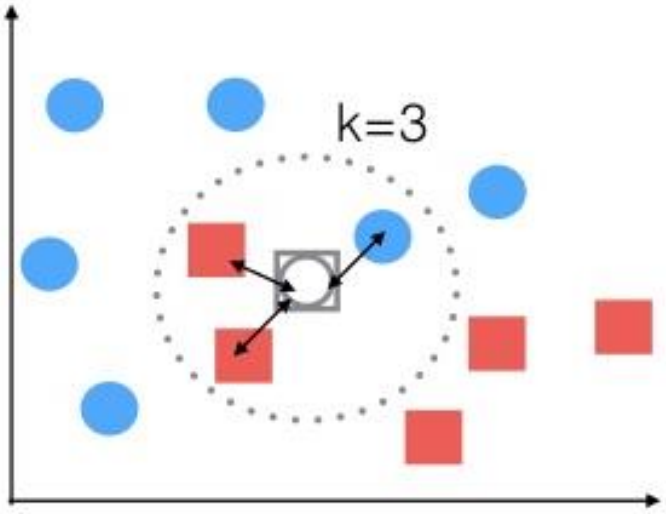
	Price	Country	Reliability	Mileage	Type	Weight	Disp.	HP	Class
Hyundai Sonata 4	9999	Korea	NA	23	Medium	2885	143	110	A
Mazda 929 V6	23300	Japan	5	21	Medium	3480	180	158	A
Nissan Maxima V6	17899	Japan	5	22	NA	3200	180	160	A
Oldsmobile Cutlass Ciera 4	13150	USA	2	21	Medium	2765	151	110	A
Oldsmobile Cutlass Supreme V6	14495	NA	1	21	Medium	3220	189	135	B
Toyota Cressida 6	21498	Japan	3	23	Medium	3480	180	190	B
Buick Le Sabre V6	16145	USA	3	23	Large	3325	231	165	B
Chevrolet Caprice V8	14525	USA	1	18	Large	3855	305	170	B
Ford LTD Crown Victoria V8	17257	USA	3	20	Large	3850	302	150	C
Chevrolet Lumina APV V6	13995	USA	NA	18	Van	3195	151	110	C
Dodge Grand Caravan V6	15395	USA	3	18	Van	3735	202	150	C

Hyundai.**Mean** (Reliability): (5+5+2)/3 = 4
Hyundai.**Median** (Reliability): 2 5 5
Oldsmobile cutlass supreme.**Mode** (Country): USA = 2, Japan = 1

Illustration: Replace Missing Data (2/2)

- Using statistics and data mining algorithms
  - Computing the most probable value
  - Others (will be studied later in this course)
  - clustering
  - sampling
  - regression
  - inference (decision tree)

Computing the Most Probable Value (a peek into k-nearest neighbors)



- Similarity based, clustering based approach
- Distance metric required
- Fill in the missing value using the (median, mean, mode) of the K-nearest neighbors.
- Con: can be very time-consuming when there are very many attributes

Handling Missing Values with Pandas

- https://towardsdatascience.com/handling-missing-values-with-pandas-b876bf6f008f

Pandas, DataFrame

- Import Pandas and Numpy
- Create a DataFrame (2-d array) with missing values.

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

```
df  =  pd.DataFrame({'column_a':[1,2,4,4,np.nan,np.nan,6],
                     'column_b':[1.2,1.4,np.nan,6.2,None,1.1,4.3],
                     'column_c':['a','?','c','d','--',np.nan,'d'],
                     'column_d':[True,True,np.nan,None,False,True,False]})
```

Display the DataFrame

```
df
```

	column_a	column_b	column_c	column_d
0	1.0	1.2	a	True
1	2.0	1.4	?	True
2	4.0	NaN	c	NaN
3	4.0	6.2	d	None
4	NaN	NaN	--	False
5	NaN	1.1	NaN	True
6	6.0	4.3	d	False

- np.nan, None, NA ('not applicable'), and NaT (for datetime64[ns] types) are standard missing value for Pandas.
- np.nan and None are for floats; NA is for integers.

Add a Column to the DataFrame

- Add column_e (‘series’ is a 1-d array)

```
new = pd.Series([1, 2, np.nan, 4, np.nan, 5], dtype=pd.Int64Dtype())
df['column_e'] = new
```

```
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1
1	2.0	1.4	?	True	2
2	4.0	NaN	c	NaN	NaN
3	4.0	6.2	d	None	4
4	NaN	NaN	--	False	NaN
5	NaN	1.1	NaN	True	5
6	6.0	4.3	d	False	NaN

isna() function (1/2)

- Pandas provides isnull(), isna() functions to detect missing values. Both of them do the same thing.
- df.isna() returns the Dataframe with boolean values (True, False) indicating missing values. cf. page 31
- notna() is the opposite of isna().

```
df.isna()
```

	column_a	column_b	column_c	column_d	column_e
0	False	False	False	False	False
1	False	False	False	False	False
2	False	True	False	True	True
3	False	False	False	True	False
4	True	True	False	False	True
5	True	False	True	False	False
6	False	False	False	False	True

isna() Function (2/2)

- df.isna().any() returns a boolean value for each column.
  If there is at least one missing value in that column,
  the result is True. cf. page 31
- df.isna().sum() returns
  the number of missing values
  in each column.

```
df.isna().any()
```

```
column_a    True
column_b    True
column_c    True
column_d    True
column_e    True
dtype: bool
```

```
df.isna().sum()
```

```
column_a    2
column_b    2
column_c    1
column_d    2
column_e    3
dtype: int64
```

Treating Values as Missing Values (1/2)

- Finding and treating non-informative values as 'missing' values (e.g., '?', '- -')
- Use na_values parameter.

```
missing_values = ["?","--"]
df_test = pd.read_csv("dataset.csv", na_values = missing_values)
```

Treating Values as Missing Values (2/2)

- Use Pandas replace() function cf. page 31
- inplace parameter saves the changes in the DataFrame.
- Default value for inplace is False; If it is set to True, changes are saved.

```
df.replace({"?": np.nan, "--": np.nan}, inplace=True)
```

```
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1
1	2.0	1.4	NaN	True	2
2	4.0	NaN	c	NaN	NaN
3	4.0	6.2	d	None	4
4	NaN	NaN	NaN	False	NaN
5	NaN	1.1	NaN	True	5
6	6.0	4.3	d	False	NaN

dropna() Function

- Drop a row or column with missing values using dropna() function.
- The 'how' parameter is used to set the condition to drop.
  - how='any' : drop if there is any missing value
  - how='all' : drop if all values are missing
- Using the 'thresh' parameter, we can set a threshold for the number of non-NA values in order for a row/column to be dropped.

Illustration (1/3)

- dropna(axis=0, how='all', inplace=True) cf. page 31
- axis parameter is used to select row (0) or column (1).

```
df.dropna(axis=0,  how='all',  inplace=True)
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1
1	2.0	1.4	NaN	True	2
2	4.0	NaN	c	NaN	NaN
3	4.0	6.2	d	None	4
4	NaN	NaN	NaN	False	NaN
5	NaN	1.1	NaN	True	5
6	6.0	4.3	d	False	NaN

- Our DataFrame does not have a row full of missing values, so setting how='all' does not drop any row.

Illustration (2/3)

- The default value for the 'how' parameter is 'any'.
- Rows with at least one missing value can be dropped.

cf. page 31

```
df.dropna(axis=0,  inplace=True)
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1

Illustration (3/3)

- Setting thresh parameter to 3 will drop rows with less than 3 non-NA values. cf. page 31

```
df.dropna(axis=0,  thresh=3,  inplace=True)
df
```

	column_a	column_b	column_c	column_d	column_e
0	1.0	1.2	a	True	1
1	2.0	1.4	NaN	True	2
3	4.0	6.2	d	None	4
5	NaN	1.1	NaN	True	5
6	6.0	4.3	d	False	NaN

fillna() Function

- Replacing missing values
- Using Pandas fillna() function, missing values can be replaced by a special scalar value or an aggregate value such as mean, median, etc.
- Further, missing values can be replaced with the value before or after it. (This is pretty useful for time-series datasets.)

Illustration (1/4)

- Replace missing values with a scalar.

```
df.fillna(25)
```

cf. page 30

	column_a	column_b	column_c	column_d
0	1.0	1.2	a	True
1	2.0	1.4	25	True
2	4.0	25.0	c	25
3	4.0	6.2	d	25
4	25.0	25.0	25	False
5	25.0	1.1	25	True
6	6.0	4.3	d	False

Illustration (2/4)

- fillna() can also be used on a particular column

cf. page 30

```
mean = df['column_a'].mean()
df['column_a'].fillna(mean)
```

```
0    1.0
1    2.0
2    4.0
3    4.0
4    3.4
5    3.4
6    6.0
Name: column_a, dtype: float64
```

Illustration (3/4)

- Using the 'method' parameter, missing values can be replaced with the values before or after them.
- ffill (forward fill) replaces missing values with the values in the previous row. cf. page 30
- bfill (backward fill) is the opposite.

```
df.fillna(axis=0,  method='ffill')
```

	column_a	column_b	column_c	column_d
0	1.0	1.2	a	True
1	2.0	1.4	a	True
2	4.0	1.4	c	True
3	4.0	6.2	d	True
4	4.0	6.2	d	False
5	4.0	1.1	d	True
6	6.0	4.3	d	False

ffill and bfill: verify

- What if the first row or last row contains NaN?
- What if the replacement row also contains NaN?

Illustration (4/4)

- If there are consecutive missing values in a column or row, using the 'limit' parameter, you can limit the number of missing values to be forward or backward filled.
- Below, the 'limit' parameter is set to 1; so only one missing value is forward filled.

```
df.fillna(axis=0,  method='ffill',  limit=1)
```

cf. page 30

	column_a	column_b	column_c	column_d
0	1.0	1.2	a	True
1	2.0	1.4	a	True
2	4.0	1.4	c	True
3	4.0	6.2	d	None
4	4.0	6.2	d	False
5	NaN	1.1	NaN	True
6	6.0	4.3	d	False

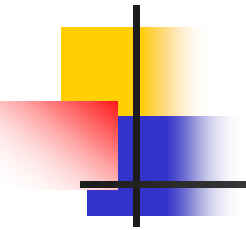
Important Note

- NumPy, Pandas, Scikit-Learn each has many library functions.
- Each function has several parameters.
- It is not possible or even necessary to explain all or most of these in detail in class.
- You will need to study most of them on your own doing homework and projects.

Exercise (Finish in 30 minutes)

- Create a Pandas (4, 4) DataFrame from the following NumPy array

  (3. ? 2. 5. * 4. 5. 6. + 3. 2. & 5. ? 7. !)
- Display the DataFrame
- Replace any non-numeric value with NaN.
- Display the DataFrame
- Apply the following functions one at a time in sequence to the DataFrame, and display the DataFrame after applying each function.
  - isna with any, and sum
  - dropna with how any, how all, thresh 1, thresh 2
  - fillna with 100, mean, median
  - ffill, bfill



Roadmap: Cleaning Dirty Data

- Introduction
- Missing Data
- Wrong Data
- Unusable Data
- Outliers

Wrong Data (1/2)

- Wrong data that a DBMS can prevent, but occur because
  - Users do not specify integrity constraints
    - data type, unique, primary key, foreign key, check, trigger
  - Users use DBMS that does not support ACID transaction management capabilities that can prevent
    - lost update, dirty read, unrepeatable read, lost transaction

Wrong Data (2/2)

- Wrong data that a DBMS canNOT prevent (see the next 3 pages)
  - Wrong non-primitive data
  - Many wrong data involving one entity
  - Many wrong data involving more than one entity

Wrong Non-Primitive Data

- DBMS does not support integrity constraints on
  - semi-structured data (e.g., email), and unstructured data (e.g., free-form text)
  - compound data (e.g., point(x,y))
  - time-series data
  - categorical data (e.g., weather = sunny, rain, cloudy, snow,…)

Wrong Data Involving One Entity

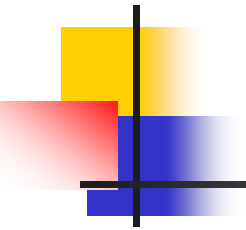
- Wrong entry (by the user or input device)
  - "age": 255 (instead of 25)
  - "name": Sillion Valy (instead of Silicon Valley)
- Switched attribute values (by the user)
  - "age, height": "180, 25"
  - "university, hobby": "tennis, Gachon"

Wrong Data Involving More Than One Entity

- Example
  - There are “Workers” table and “Department” table. The “Department” table has “Number of Workers” column.
  - The total number of workers from the two tables may differ.

Homework: DBMS-preventable wrong data (cf. page 49)

- RDBMSs support various integrity constraints (both on columns and on the table). For each
  - describe it, and
  - provide an example use case of how it works (using a very tiny table)
- RDBMS transaction management guarantees ACID properties.
  - briefly explain what ACID is, and
  - describe each of the (lost update, dirty read, unrepeatable read, lost transaction) problems with a simple example scenario



Roadmap: Cleaning Dirty Data

- Introduction
- Missing Data
- Wrong Data
- Unusable Data
- Outliers

Roadmap: Unusable Data

- Due to ambiguous meanings
- Due to multiple valid representations
- Due to non-conformance to standards
- Due to database integration
- Redundant data

Data with Ambiguous Meanings

- Due to lack of context/full semantics
  - Miami (city in the state of Florida or Ohio)
    광주 (경기도, 전라도)

Data with Multiple Valid Representations (* not really 'unusable')

- Due to use of different versions
  - table 1: Naver; table 2: NHN Corp.
- Due to use of abbreviations
  - Microsoft, MS
  - KakaoTalk, KaTalk
- People's names
  - Acronyms William Clinton, Bill Clinton, President Clinton
  - Hong Gildong, Gildong Hong, Gil-Dong Hong
- Abbreviations
  - St. Paul, Saint Paul,…
  - DJ, 김대중; YS 김영삼
- Due to data integration
  - John's pay is 200 in DB-1, John's pay is 50 in DB-2 (both are correct, since John works part time for two companies)

Data That Do Not Conform to Standards

- Due to use of different data types
  - (credit card number: table 1 uses integer with no dashes, table 2 uses string with dashes)
- Incomplete compound data
  - concatenated data – national ID card, address, phone number, credit card number, etc.
  - hierarchical data – address (province, city, gu, dong, street address, apartment number)

Mismatch Due To Database Integration

- (different) Data types
  - integer, real, char, varchar
- Representation formats
  - decimal, hexadecimal
- Date and time formats
  - 2013.08,31 (Korea), 31/08/2013 (France), 08/31/2013 (US)
- Measurement units
  - weight (ounce/pound/gallon vs. gram/kg/liter)
  - length (inch/foot/yard vs. cm/meter)
  - area (acre/sq ft vs. 평),…
- Monetary units
  - dollar, euro, ruble, 원, 엔, 위안,…

Redundant Data

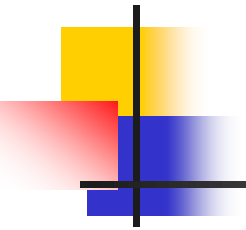
- Redundant data for the same object
  - due to integration of data from multiple data sources
  - due to entry error
- Need to keep only one copy
- Correlation analysis can help detect redundant data

Two Steps in Dealing with Dirty Data

- (1) Prevent dirty data from getting into the database (before insert/update/delete) – (* This is limited *)
  - type checking
  - integrity constraints
  - use of triggers
- (2) Clean the dirty data that got through – (* It takes a lot of work *)
  - Using data cleaning tools, if available
    - commercial tools
    - national directory of names, addresses, phone numbers, e-mail addresses
  - Manual cleaning (using a data browser tool)

Exercise

- Browse through your computer folders and identify the following:
  - redundant data
  - outdated data
  - wrong data



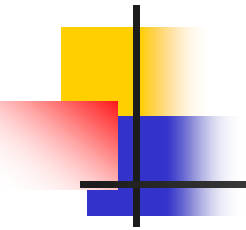
Roadmap: Cleaning Dirty Data

- Introduction
- Missing Data
- Wrong Data
- Unusable Data
- Outliers

Outliers (Anomaly, Noise Data)

- An outlier (also known as anomaly, or noisy data) is data that does not belong in a group of "similar" data.
  - "An observation point that is distant from other observations."
- If it is statistically not significant, it may be best to remove it from analysis.

- ** "Outliers" is often included as part of "Dirty Data"; However, it may also be included as part of "Data Reduction".



Examples

- A person whose weight is 200 kg.
- A person whose height is 210 cm.
- A student who scores 1000 points in the Data Science course, and a student who scores 50 points in the same course.

How to Deal with Outliers

- If they are minor, remove them
- If they are likely not very important data, re-weight them or smooth them
  - give them lower weight than other samples
- If they are the data we are looking for, use them for deeper analysis (and remove other data).
  - examples: fraud detection, Corona virus infection, cancer cells in medical imaging

Preprocessing
Using Pandas and Scikit-Learn

Acknowledgments

- https://www.datacamp.com/community/tutorials/k-means-clustering-python

Titanic Tragedy

- (April 15, 1912) 1502 died out of 2224 passengers and crew
- Some groups of people (such as women, children, and the upper-class passengers) survived better than others.

Titanic Dataset

- 12 Features
  - passengerID, name, sex, age, ticket, fare, passengerClass, cabin, embarked, parch(parents/children aboard), sibsp(siblings/spouse aboard)
  - survived
- Labeled dataset: passengers who survived, and who did not.

(Self-Study Outside the Class)

Import Class Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

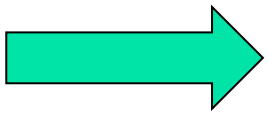
Import Data

```
# Load the train and test datasets to create two
DataFrames

train_url =
"http://s3.amazonaws.com/assets.datacamp.com/course/
Kaggle/train.csv"
train = pd.read_csv(train_url)
test_url =
"http://s3.amazonaws.com/assets.datacamp.com/course/
Kaggle/test.csv"
test = pd.read_csv(test_url)
```

Peek at the Data (1/3)

```
print("***** Train_Set *****")
print(train.head())       # print top n rows, n=5 default
print("₩n")
print("***** Test_Set *****")
print(test.head())
```

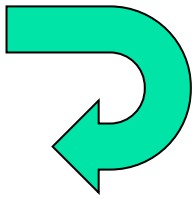
```
***** Train_Set *****
   PassengerId  Survived  Pclass

0          1         0       3
1          2         1       1
2          3         1       3
3          4         1       1
4          5         0       3
```



continued to next page

Peek at the Data (2/3)

	Name	Sex	Age	SibSp
0	Braund, Mr. Owen Harris	male	22.0	1
1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1
2	Heikkinen, Miss. Laina	female	26.0	0
3	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1
4	Allen, Mr. William Henry	male	35.0	0



continued

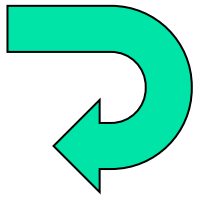
	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

Port of Embarkation
(C = Cherbourg; Q = Queenstown; S = Southampton)

Peek at the Data (3/3)

***** Test_Set *****

	PassengerId	Pclass	Name	Sex	₩
0	892	3	Kelly, Mr. James	male	
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	
2	894	2	Myles, Mr. Thomas Francis	male	
3	895	3	Wirz, Mr. Albert	male	
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	



continued

	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	34.5	0	0	330911	7.8292	NaN	Q
1	47.0	1	0	363272	7.0000	NaN	S
2	62.0	0	0	240276	9.6875	NaN	Q
3	27.0	0	0	315154	8.6625	NaN	S
4	22.0	1	1	3101298	12.2875	NaN	S

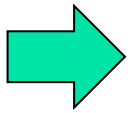
Get Statistics on the Data (1/3)

describe() # calculates statistical data of the numerical values of the Series or DataFrame

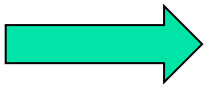
```
print("***** Train_Set *****")
print(train.describe())
print("₩n")
print("***** Test_Set *****")
print(test.describe())
```

```
***** Train_Set *****
       PassengerId    Survived      Pclass         Age       SibSp  ₩
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000
```

continued to next page



Get Statistics on the Data (2/3)



	Parch	Fare
count	891.000000	891.000000
mean	0.381594	32.204208
std	0.806057	49.693429
min	0.000000	0.000000
25%	0.000000	7.910400
50%	0.000000	14.454200

Get Statistics on the Data (3/3)

***** Test_Set *****

	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

List Feature Names

```
print(train.columns.values)

['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age'
'SibSp' 'Parch' 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

Check for Missing Values

```
# For the train set
train.isna().head()

# For the test set
test.isna().head()
```

Get Statistics on the Missing Values (1/2)

```
print("*****In the train set*****")
print(train.isna().sum())
print("₩n")
print("*****In the test set*****")
print(test.isna().sum())
```

```
*****In the train set*****
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2

dtype: int64
```

Get Statistics on the Missing Values (2/2)

*****In the test set*****

PassengerId	0
Pclass	0
Name	0
Sex	0
Age	86
SibSp	0
Parch	0
Ticket	0
Fare	1
Cabin	327
Embarked	0

dtype: int64

Fill Missing Values

```
# Fill missing values with mean column values in the train set
train.fillna(train.mean(), inplace=True)

# Fill missing values with mean column values in the test set
test.fillna(test.mean(), inplace=True)
```

Check for Missing Values in Training Data (after filling them)

```
print(train.isna().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
```

Check for Missing Values in Test Data

```
print(test.isna().sum())
```

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0

dtype: int64
```

Why Are There Still Missing Data?

- "Cabin" and "Embarked" columns have non-numeric values.

- Data Types for the Columns

categorical: Survived, Sex, Embarked
ordinal: Pclass
continuous: Age, Fare
discrete: SibSp, Parch
mix of numeric and alphanumeric: Ticket, Cabin

Find Survival Count of Passengers wrt Features: Pclass

```
train[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

Find Survival Count of Passengers wrt Features: Sex

```
train[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	Sex	Survived
0	female	0.742038
1	male	0.188908

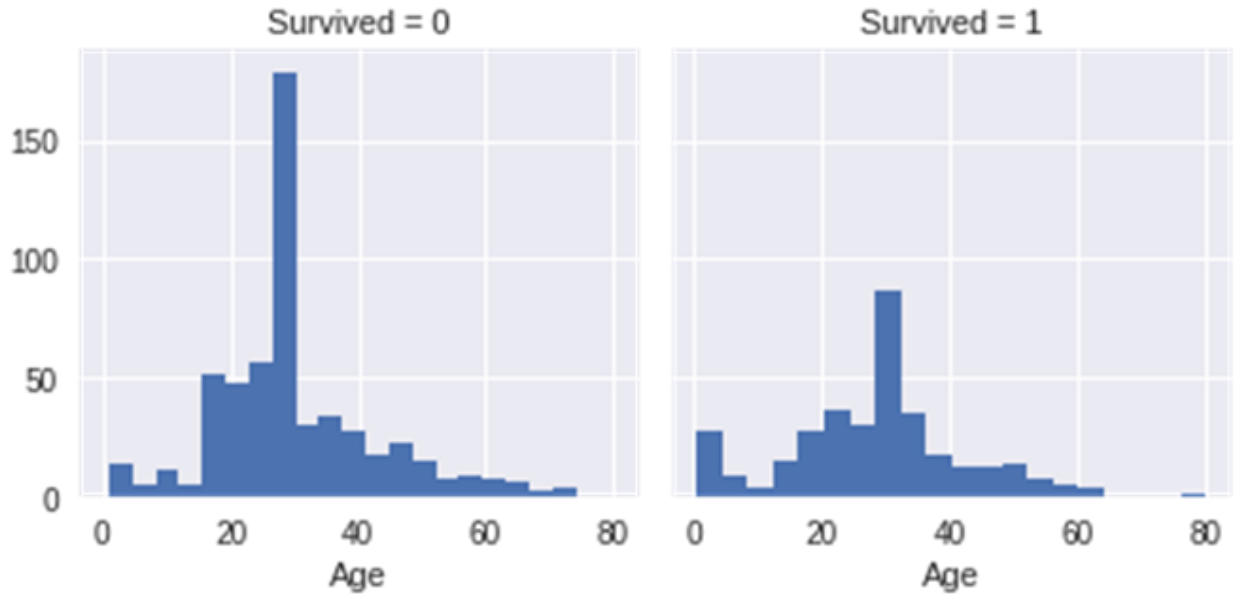
Find Survival Count of Passengers wrt Features: SibSp

```
train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

	SibSp	Survived
0	0	0.345395
1	1	0.535885
2	2	0.464286
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

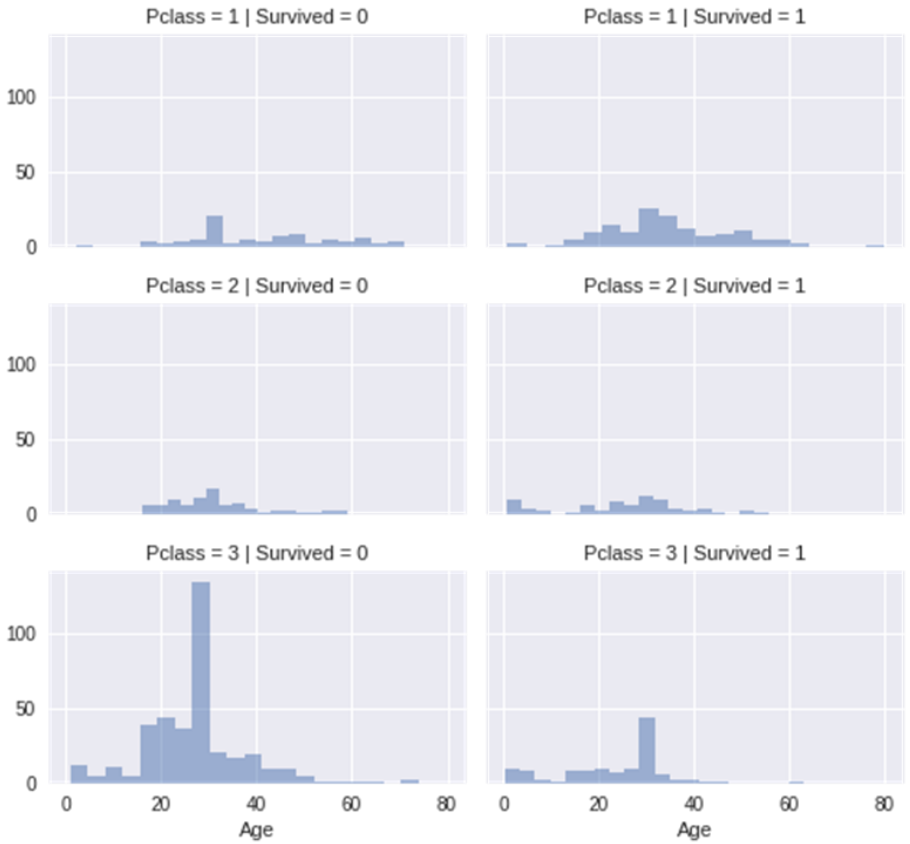
Plot the Data

```
g = sns.FacetGrid(train, col='Survived')
g.map(plt.hist, 'Age', bins=20)
<seaborn.axisgrid.FacetGrid at 0x7fa990f87080>
```



Plot the Data

```
grid = sns.FacetGrid(train, col='Survived',
        row='Pclass', size=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```



Show Data Types for the Features

```
train.info()
```

info() returns information about the DataFrame including the data types of each feature and memory usage of the DataFrame

```
<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             891 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
```

Feature Engineering

- Features like Name, Ticket, Cabin, and Embarked do not have any impact on the survival status of the passengers.
- Remove them from consideration.

```
train = train.drop(['Name','Ticket','Cabin','Embarked'], axis=1)
test = test.drop(['Name','Ticket','Cabin','Embarked'], axis=1)
```

Convert ‘Sex’ Feature to Numeric Values

LabelEncoder encodes labels of a feature to a value between 0 and n_classes-1, where n is the number of distinct labels.

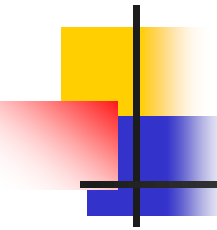
(e.g.,’sex’ has 2 labels: ‘f’ and ‘m’
Label Encoder encodes them to ‘o’ and ‘1’

```
labelEncoder = LabelEncoder()
labelEncoder.fit(train['Sex'])
labelEncoder.fit(test['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])
test['Sex'] = labelEncoder.transform(test['Sex'])
```

Converted Data Types

```
train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Sex            891 non-null int64
Age            891 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Fare           891 non-null float64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
```

```
test.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 7 columns):
PassengerId  418 non-null int64
Pclass       418 non-null int64
Sex          418 non-null int64
Age          418 non-null float64
SibSp        418 non-null int64
Parch        418 non-null int64
Fare         418 non-null float64

dtypes: float64(2), int64(5)
```



Roadmap: Data Value Changes

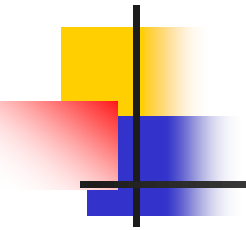
- Cleaning dirty data
- Text preprocessing
- Data discretization
- Data normalization/standardization
- Encoding for data mining algorithms

Text Data Processing

- Text data is very important, as most of world's knowledge is in text: books, news articles, advertising,…
- The focus of data science and machine learning has been on numerical data.
- Just as images have to be converted to numerical data, to automate text processing, text has to be converted to numerical data.

Text Data Preprocessing

- Just like numerical data, text data needs to undergo preprocessing.
- Text preprocessing includes
  - noise removal
  - tokenization/segmentation/lexical analysis
  - normalization



Noise Removal

- When collecting a corpus from the Web
  - Remove text file headers, footers.
  - Remove markup and metadata from HTML, XML, JSON, etc. documents

Tokenization of Text Documents

- Split longer strings of text into smaller pieces (segments and tokens)
  - (a book) into chapters - into paragraphs – into sentences – into words
- Punctuation marks, etc. make it difficult to identify segment ending symbols.
- Examples
  - "Dr. Ford did not ask Col. Mustard the name of Mr. Smith's dog." (First 3 '.' are not sentence ending symbols.)
  - "The student isn't living in on-campus housing, and she's not wanting to visit Hawai'i." (Each apostrophe is different.)

Normalization of Text Elements (1/2)

- Put all elements of a text on a common ground, so that text similarity, word frequency count, etc. can be reasonably computed.
- Three major tasks
  - word stemming
    - obtain word stem by removing prefix, suffix, etc.
    - running → run, ended → end, unhappy → happy, lightly → light
  - word lemmatization
    - convert word to canonical form
    - better → good, ran → run, finest → fine

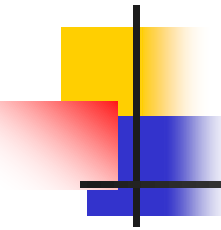
Normalization of Text Elements (2/2)

- Three major tasks (cont'd)
  - everything else
    - convert to the same case (upper, lower)
    - remove punctuation marks
    - convert numbers to words (7 → seven)
    - remove stop words ("a", "the", "is", "are",….)

Exercise

- Do tokenization and normalization on the following Nigerian fraud spam text.

- "Dear Sir, SEEKING YOUR IMMEDIATE ASSISTANCE. Please permit me to make your acquaintance in so informal a manner. My name is. DAN PATRICK of the Democratic Republic of Congo and One of the close aides to the former President of the Democratic Republic of Congo LAURENT KABILA of blessed memory, may his soul rest in peace."



Roadmap: Data Value Changes

- Cleaning dirty data
- Text preprocessing
- Data discretization
- Data normalization/standardization
- Encoding for data mining algorithms

Data Discretization (Binning)

- Discretization Method
  - Sort data and partition into (usually equal-frequency) bins

    (e.g., change numerical data, such as age values, to categorical data, such as 20, 30, 40,…)
  - Data in each bin may be smoothed by bin means, bin median, or bin boundaries (the minimum and maximum values)

Roadmap: Data Value Changes

- Cleaning dirty data
- Text preprocessing
- Data discretization
- Data normalization/standardization
- Encoding for data mining algorithms

Data Scaling (Normalizing)

- Data scaling = data normalization = data standardization
- Data scaling means
  - either applying a transformation to make the data roughly normally distributed,
  - or scaling data to fall within a small, specified range (e.g., between 0 and 1, or between -1 and +1)
  - example: height 180 cm scaled to 0.5.

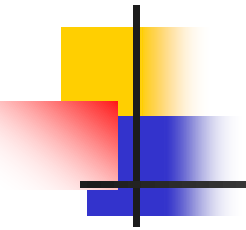
Why Scale (Normalize) Data?

- Machine learning algorithms may assign bigger importance to a feature with much larger numbers (e.g., height, 180 cm) than a feature with much lower numbers (e.g., weight, 60 kg).
- Many machine learning algorithms perform better or converge faster when features are on a similar scale and/or close to normally distributed.
- Examples of such algorithms
  - linear and logistic regression
  - nearest neighbors
  - neural networks
  - support vector machines with radial basis kernel functions
  - principal component analysis
  - linear discriminant analysis

Scaling Data Using Scikit-Learn

```
import pandas as pd
from sklearn import preprocessing
scaler = preprocessing.XXXScaler()
scaled_df = scaler.fit_transform(df)
```

XXX = MinMax (or Standard, or Robust or Normalizer)



Roadmap: Data Scaling

- min-max scaling
- z-score (standard) scaling
- decimal scaling
- log normalization
- robust scaling
- vector normalization

Min-Max Scaling

- Fit the data within unity - all the data will have a value between o and 1.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- (e.g.) height max: 210, min: 150, new height: 180

  normalized height:

  (180-150)/(210-150)=30/60=0.5

Z-Score (Standard) Scaling

- Standardization (or Z-score normalization) is the process of rescaling the features
  - so that they'll have the properties of a Gaussian distribution with μ=0 and σ=1
  - where μ is the mean and σ is the standard deviation from the mean
- Converts all data to a common scale
  - reflects how many standard deviations from the mean the data falls
- The number of standard deviations from the mean is called "standard score", "sigma", or "z-score'.

Computing Standardization

$$z = \frac{x - \mu}{\sigma}$$

- where μ is the mean and σ is the standard deviation from the mean

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

Robust Scaling

- Robust scaling is similar to min-max scaling, but uses the interquartile range rather than the min-max.
- It is robust to outliers (because outliers are not considered).

$$x_{new} = \frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

$Q_1(x)$ is the 1st quartile (25%) value of x.

$Q_3(x)$ is the 3rd quartile (75%) value of x.

Computing the Quartiles (1/2)

- Sort the list of N numbers, and apply the formula.

Quartile Formula

The Quartile Formula For Q1	$=$	$\frac{1}{4}(n+1)^{th}$ term
The Quartile Formula For Q3	$=$	$\frac{3}{4}(n+1)^{th}$ term
The Quartile Formula For Q2	$=$	Q3 – Q1 (Equivalent to Median)

- Example: 5, 7, 4, 4, 6, 2, 8 (N=7)
  - Sort the list: 2, 4, 4, 5, 6, 7, 8
  - Quartile 1 (Q1) = ¼ * 8 = 2 (2^{nd} term = 4)
  - Quartile 3 (Q3) = ¾ * 8 = 6 (6^{th} term = 7)
  - Quartile 2 (Q2) = median = Q3-Q1= 4 (4^{th} term = 5)
  - ** Q3-Q1 is Inter-Quartile Range (IQR)

Computing the Quartiles (2/2) Another Example

list: 4, 17, 7, 14, 18, 12, 3, 16, 10, 4, 4, 11 (N=12)

- Sort the list:

  3, 4, 4, 4, 7, 10, 11, 12, 14, 16, 17, 18

- In this case all the quartiles are between two numbers:

  Quartile 1 (Q1) = ¼ * 13

  = 3.25 (between the 3rd and 4th terms)

  take an average of the two terms: (4+4)/2=4

  Quartile 3 (Q3) = ¾ * 13

  = 9.75 (between the 9th and 10th terms)

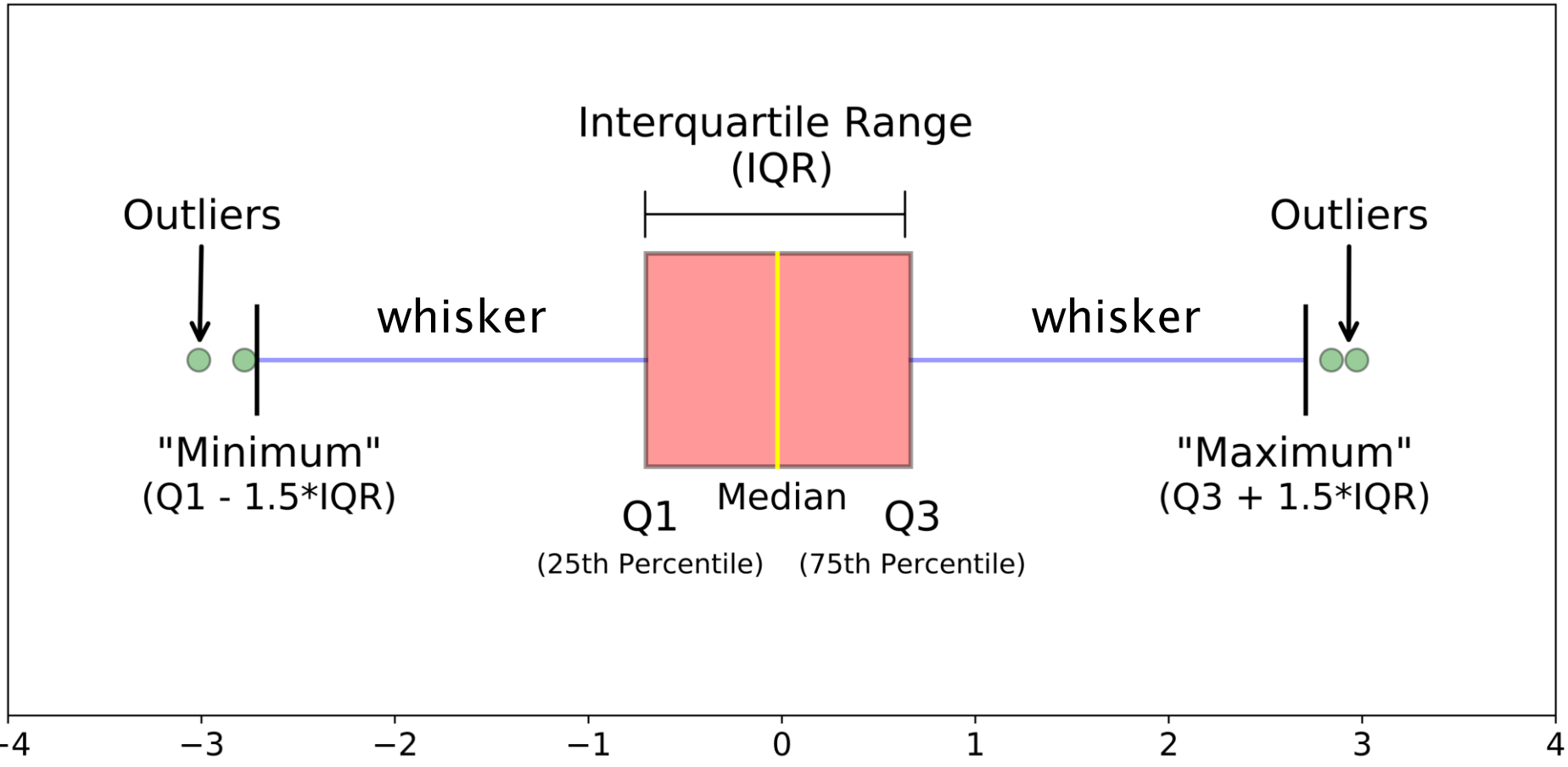
  take an average of the two terms: (14+16)/2=15

  Quartile 2 (Q2) = 2/4 * 13

  = 6.5 (between the 6th and 7th terms)

  take an average of the two terms: (10+11)/2=10.5

Quartile (Box and Whisker Plot)



Vector Normalizer (Direction Cosine)

- Vector normalizer (direction cosine, coordinate normalizer) scales each value by dividing it by its magnitude in n-dimensional space for n number of features.
- For example, if the features are x, y and z Cartesian co-ordinates, the scaled values for x, y, z would be:

$$\alpha = \cos a = \frac{\mathbf{v} \cdot \mathbf{e}_x}{\|\mathbf{v}\|} = \frac{v_x}{\sqrt{v_x^2 + v_y^2 + v_z^2}},$$

$$\beta = \cos b = \frac{\mathbf{v} \cdot \mathbf{e}_y}{\|\mathbf{v}\|} = \frac{v_y}{\sqrt{v_x^2 + v_y^2 + v_z^2}},$$

$$\gamma = \cos c = \frac{\mathbf{v} \cdot \mathbf{e}_z}{\|\mathbf{v}\|} = \frac{v_z}{\sqrt{v_x^2 + v_y^2 + v_z^2}}.$$

- Each point is now within 1 unit of the origin on this Cartesian co-ordinate system.

Comparing the Scalers

- http://benalexkeen.com/feature-scaling-with-scikit-learn/

```
import pandas as pd
import numpy as np

np.random.seed(1)
df = pd.DataFrame({
   'x1': np.random.normal(0, 2, 10000),
   'x2': np.random.normal(5, 3, 10000),
   'x3': np.random.normal(-5, 5, 10000)
})
```

numpy.random.normal(loc = 0.0, scale = 1.0, size = None) : creates an array of specified shape and fills it with random values which is actually a part of Normal Distribution.

"loc" is mean of the distribution

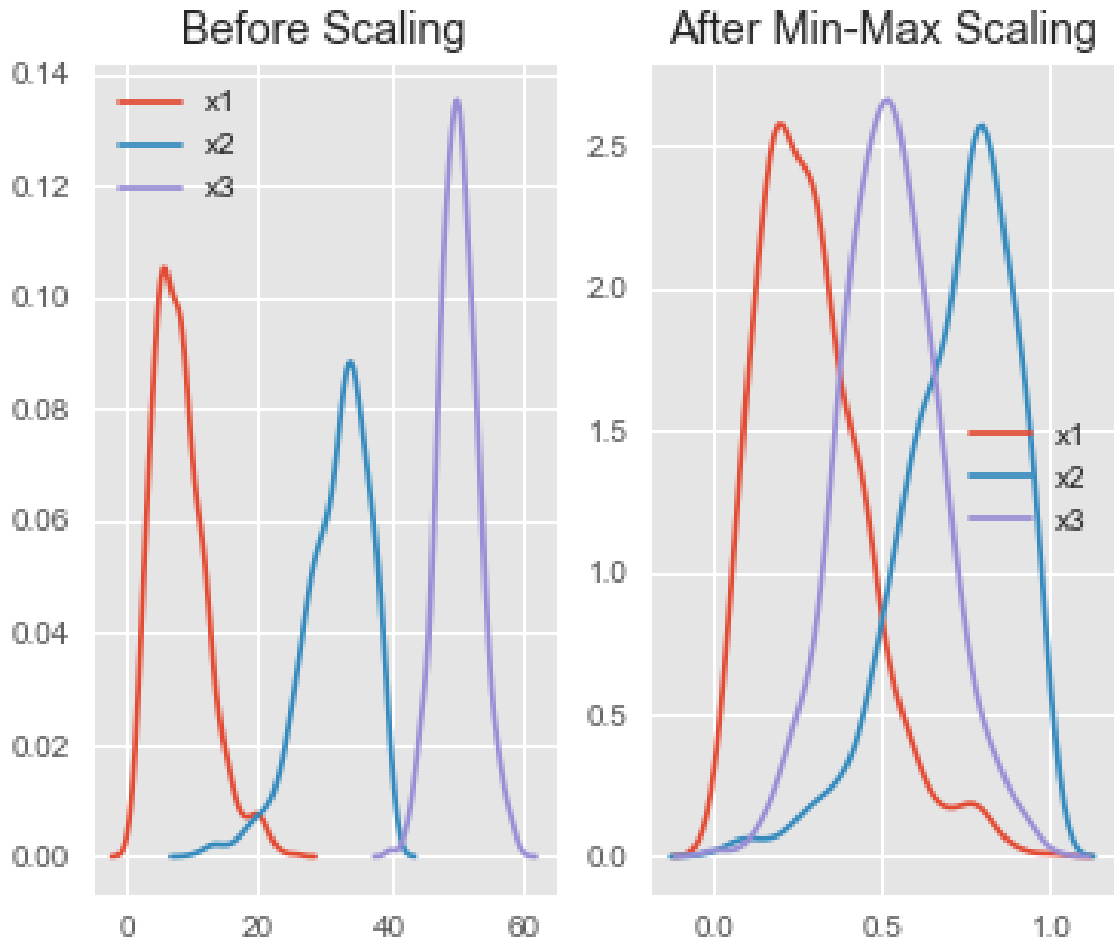
"scale" is standard deviation,

"size" is the shape of the distribution

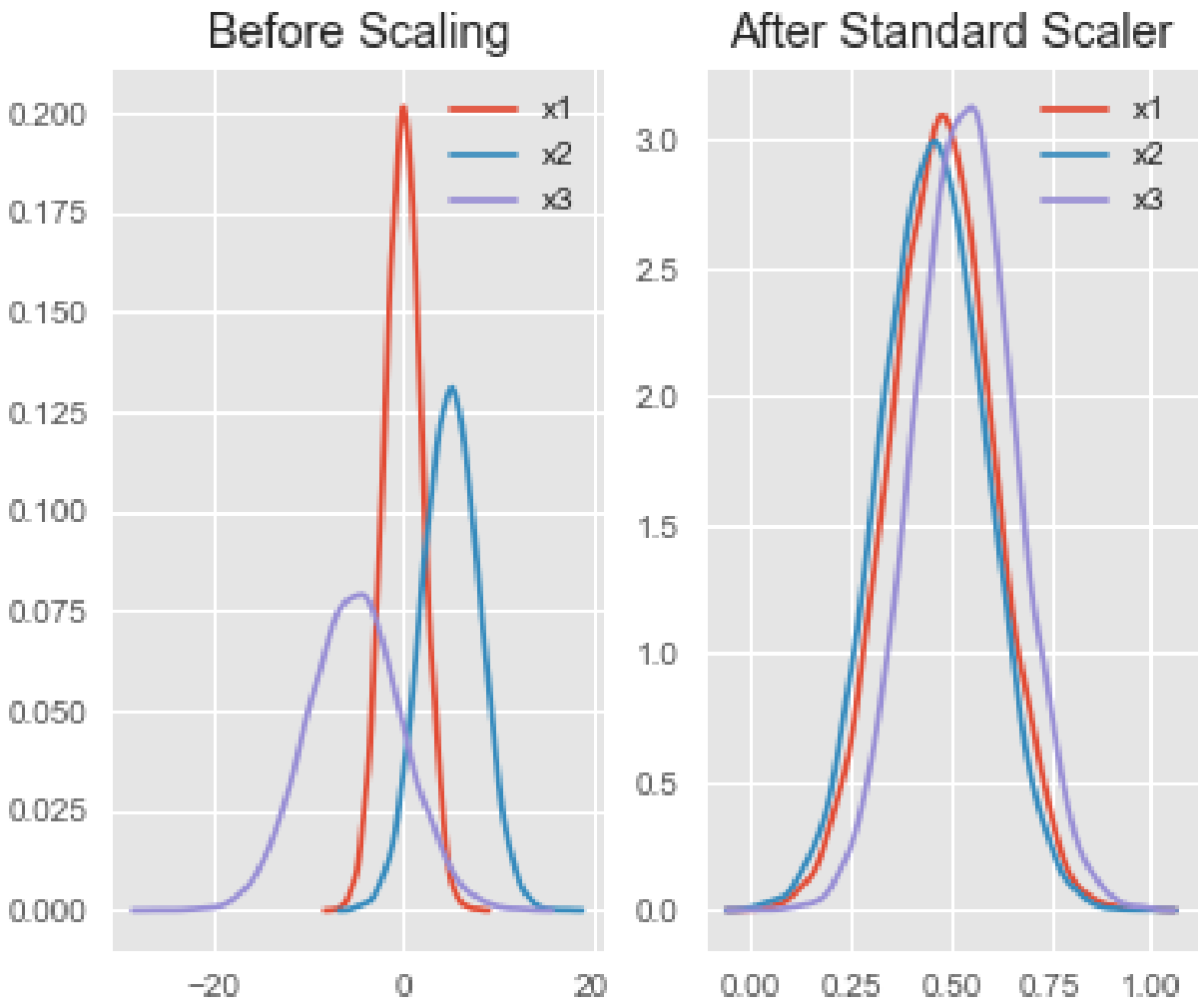
(preparing to plot the data -- just for completeness)

```
scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df,
columns=['x1', 'x2', 'x3'])

fig, (ax1, ax2) = plt.subplots(ncols=2,
figsize=(6, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)
ax2.set_title('After Standard Scaler')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
plt.show()
```

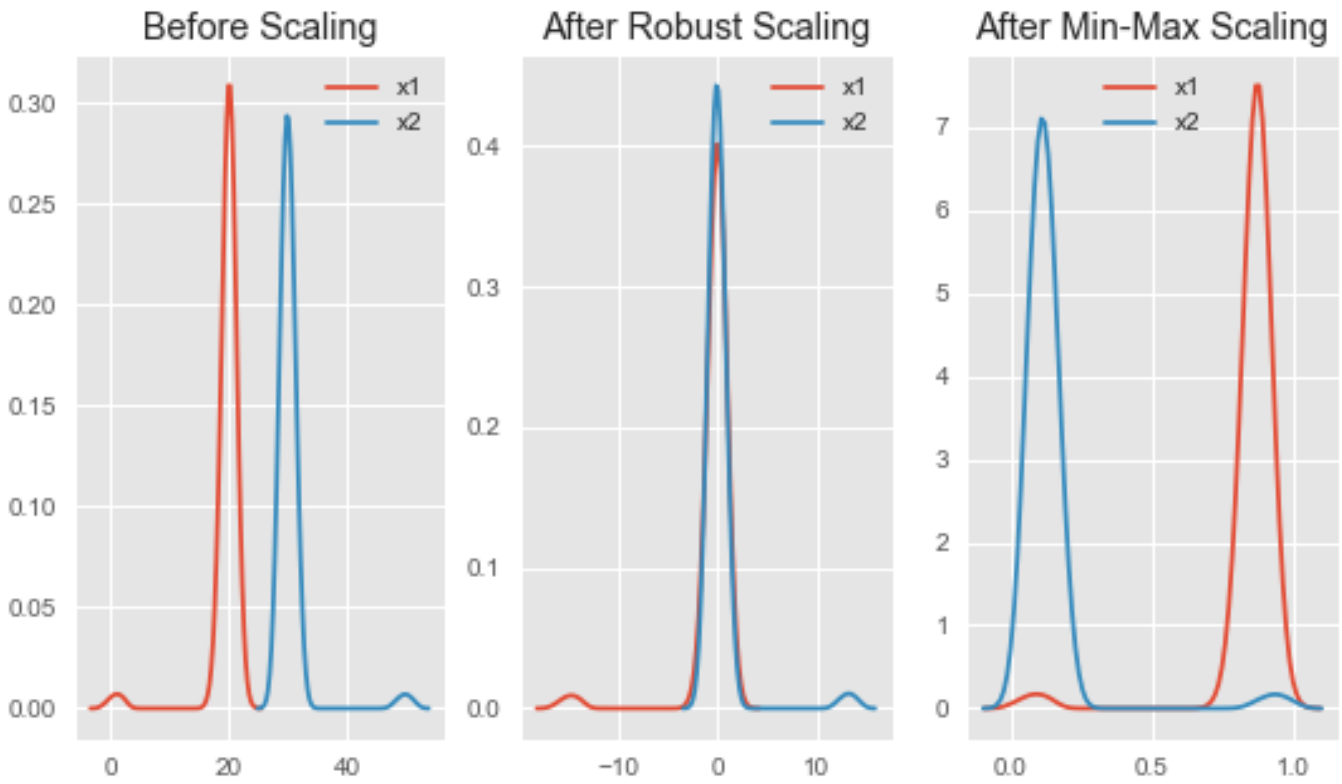
Effect of the Min-MaxScaler



Effect of the Standard Scaler

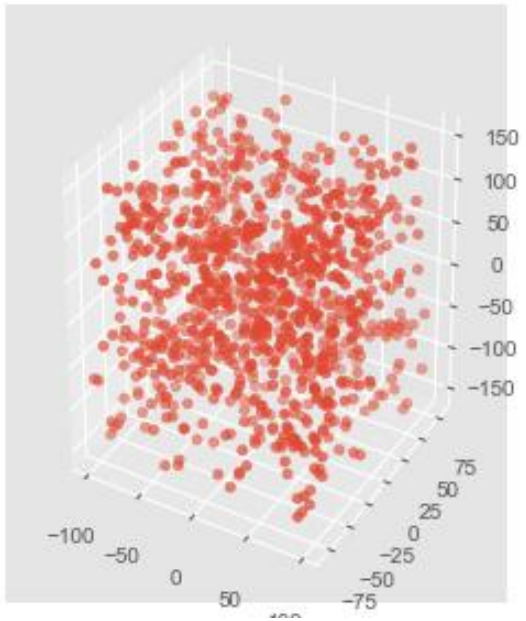


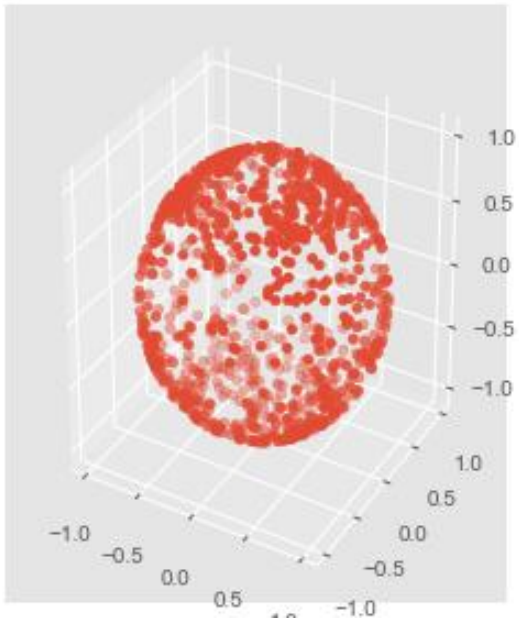
Effect of the Robust Scaler



Effect of the Vector Normalizer

```
df = pd.DataFrame({
    'x1': np.random.randint(-100, 100, 1000).astype(float),
    'y1': np.random.randint(-80, 80, 1000).astype(float),
    'z1': np.random.randint(-150, 150, 1000).astype(float),
})
xxx = preprocessing.Normalizer()
```





Example Standard Scaling (Exercise: confirm correctness)

- Student test scores (total 60 points)

  20, 15, 26, 32, 18, 28, 35, 14, 26, 22, 17

  (most have less than 30 points)
- The mean: 23
- The standard deviation: 6.6
- The standard scores:

  -0.45, -1.21, 0.45, 1.36, -0.76, 0.76,

  1.82, -1.36, 0.45, -0.15, -0.91
- The student scores 15 and 14 stand out to receive an F !!

Exercise: Standard Scaling

- Student test scores (total 60 points)

  28, 35, 26, 32, 28, 28, 35, 34, 46, 42, 37
- The mean: ??
- The standard deviation: ??
- The standard scores: ??
- Which scores should receive an F?

Decimal Scaling

- Suppose there are numbers which range from -986 to 917.
- To normalize by decimal scaling
  - Find the largest number in the given range
  - Count the number of digits in the largest number (i.e., j = 3)
  - Divide each number by 10^j (i.e., $10^3 = 1000$)
  - The normalized values for -986 to 917 are -0.986 to 0.917

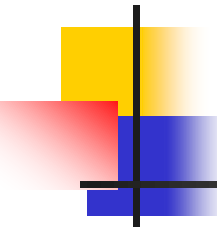
Log Normalization

- Used when values are ranged over several orders of magnitude.

$$X' = \log_b(X) \quad /* \ b=2, 10, e \ */$$

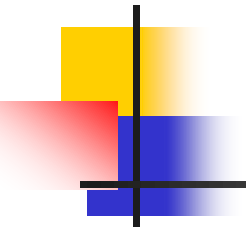
Choosing a Normalization Method

- large range of data: (i.e. $4 to $120,000,000)
  - log transformation is often good
- skewed data (often large range)
  - log transformation is often good
- high entropy (highly random data)
  - usually normalizing to [-1,1] is good
- low entropy (less random data)
  - often z-standardization is good
- near normally distributed
  - z-standardization is good



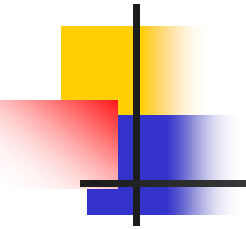
Roadmap: Data Value Changes

- Cleaning dirty data
- Text preprocessing
- Data discretization
- Data normalization/standardization
- Encoding for data mining algorithms



Encoding for Data Mining Algorithms

- Map data to an array for processing by data mining algorithms
- Convert non-numeric data to numeric data
  - categorical data to numeric data
  - text to numeric data
  - images (pixels) to array of numeric data
- Convert data to be in [0, 1] for processing by neural networks



End of Class