Doing Data Science

Unit 5

Faizan Javed
Data Science @ SMU

Admin notes

Live Session Unit 04 assignment due today

Live Session Unit 05 assignment due next Monday, 02/12/18

Case Study 1 details will be posted by tomorrow

Main topics

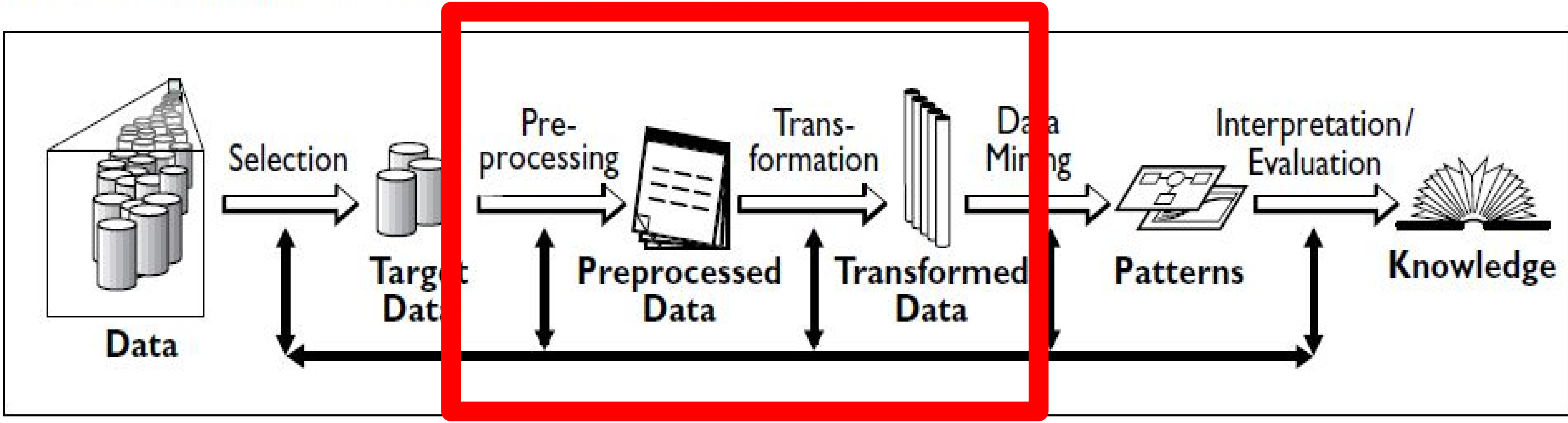
Messy data

Regular expressions

Tidy data

The Data Science Process

Figure 1. Overview of the steps constituting the KDD process



Data Munging/Cleaning - Data Janitorial tasks

Numeric/Text transformations:

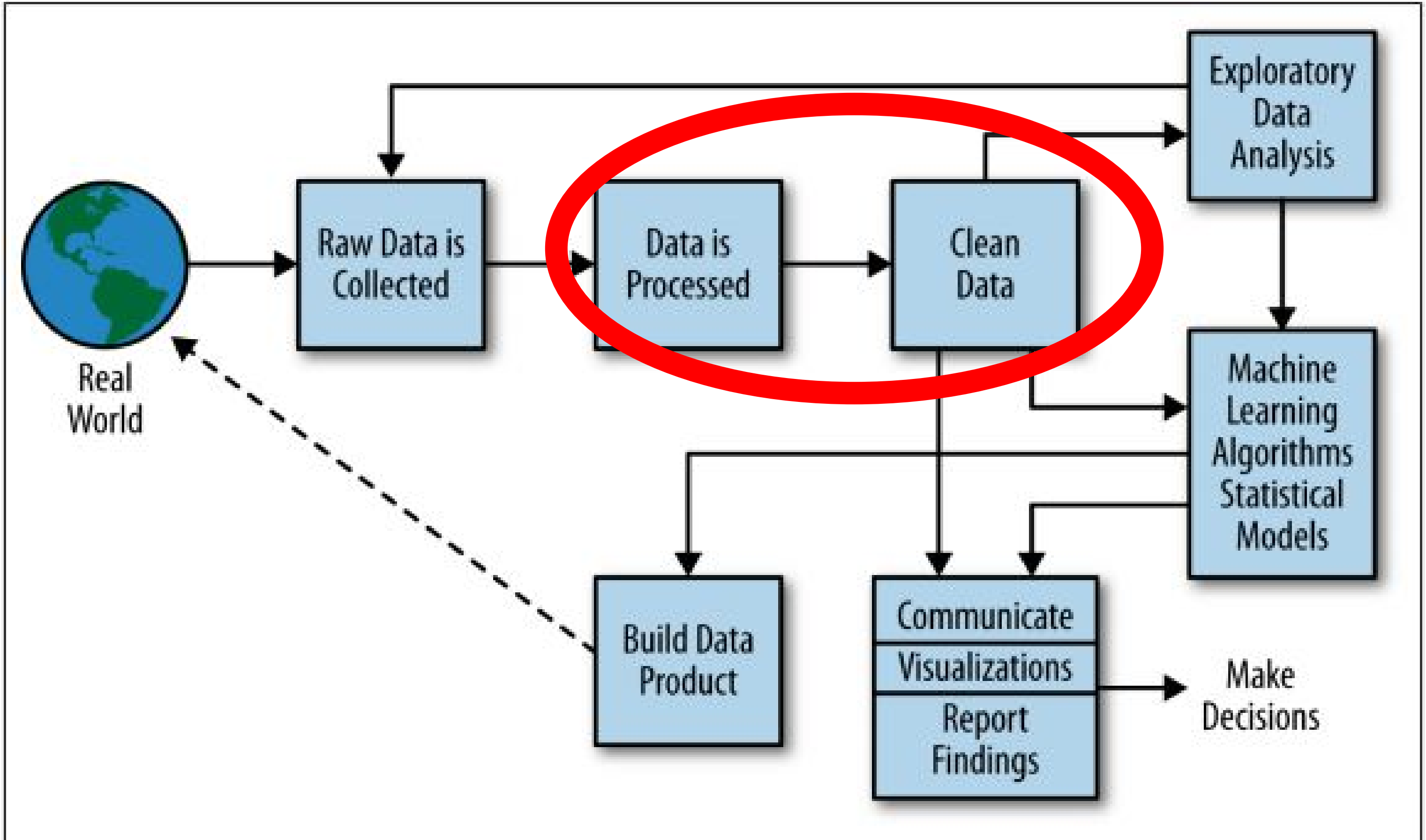
Normalization, tokenization, remove stop words, filter inconsistent values, impute missing values, numeric values binning

Data transformations:

Convert date formats, enrich with geo data, convert currency, deduplication

Open Source tool: Google Refine http://openrefine.org/

The Data Science Process



Dirty data examples

Naming conventions: TX vs Texas

Parsing text into fields (separator issues)

Missing required field

Different representations (5 vs Five)

Redundant records (exact match etc)

Formatting issues with dates and other metrics

Outliers (e.g., 200 in age column)

Five most common problems with messy data

(Hadley, Wickham, 2014)

□Column headers are values, not variable names. □

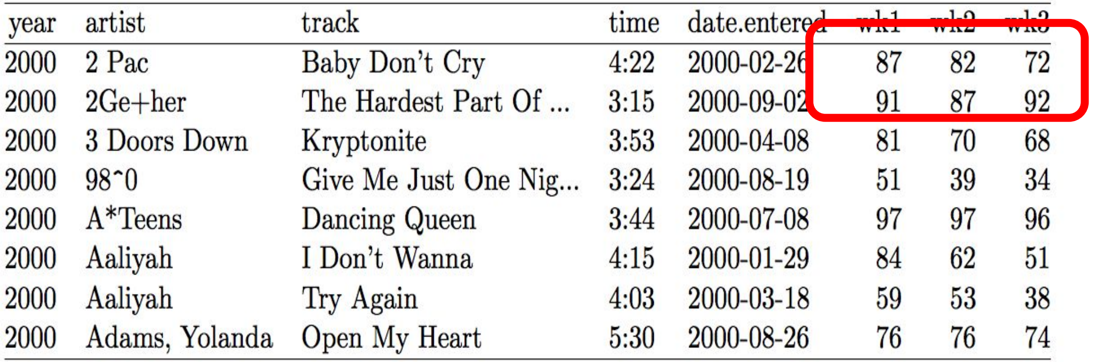
Multiple variables are stored in one column. □

Variables are stored in both rows and columns. □

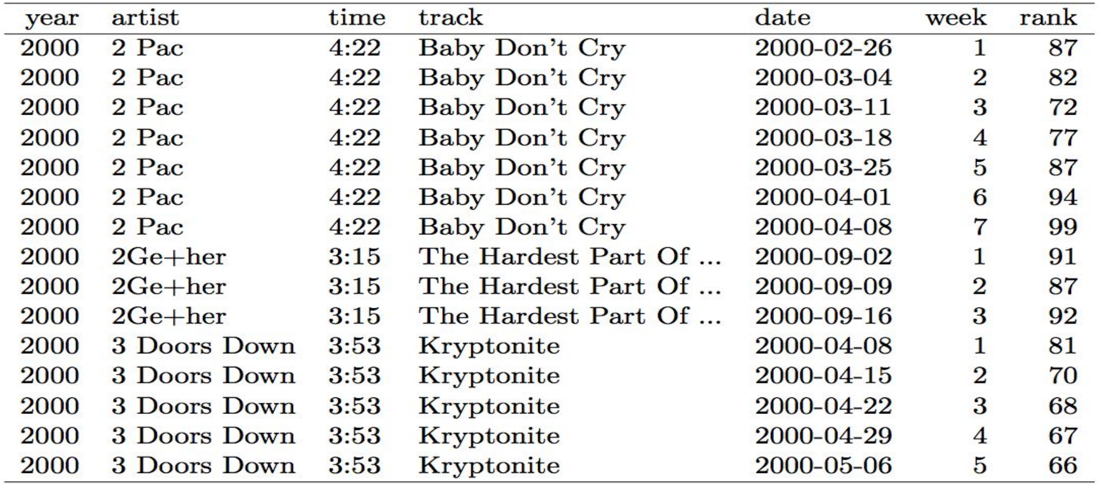
Multiple types of observational units are stored in the same table. □

A single observational unit is stored in multiple tables. *E.g, a separate table of an individual’s medical history for each year of their life.*

Column headers are values, not variable names



year	artist	track	time	date.entered	wk1	wk2	wk3
2000	2 Pac	Baby Don't Cry	4:22	2000-02-26	87	82	72
2000	2Ge+her	The Hardest Part Of ...	3:15	2000-09-02	91	87	92
2000	3 Doors Down	Kryptonite	3:53	2000-04-08	81	70	68
2000	98ˆ0	Give Me Just One Nig...	3:24	2000-08-19	51	39	34
2000	A*Teens	Dancing Queen	3:44	2000-07-08	97	97	96
2000	Aaliyah	I Don't Wanna	4:15	2000-01-29	84	62	51
2000	Aaliyah	Try Again	4:03	2000-03-18	59	53	38
2000	Adams, Yolanda	Open My Heart	5:30	2000-08-26	76	76	74



year	artist	time	track	date	week	rank
2000	2 Pac	4:22	Baby Don't Cry	2000-02-26	1	87
2000	2 Pac	4:22	Baby Don't Cry	2000-03-04	2	82
2000	2 Pac	4:22	Baby Don't Cry	2000-03-11	3	72
2000	2 Pac	4:22	Baby Don't Cry	2000-03-18	4	77
2000	2 Pac	4:22	Baby Don't Cry	2000-03-25	5	87
2000	2 Pac	4:22	Baby Don't Cry	2000-04-01	6	94
2000	2 Pac	4:22	Baby Don't Cry	2000-04-08	7	99
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-02	1	91
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-09	2	87
2000	2Ge+her	3:15	The Hardest Part Of ...	2000-09-16	3	92
2000	3 Doors Down	3:53	Kryptonite	2000-04-08	1	81
2000	3 Doors Down	3:53	Kryptonite	2000-04-15	2	70
2000	3 Doors Down	3:53	Kryptonite	2000-04-22	3	68
2000	3 Doors Down	3:53	Kryptonite	2000-04-29	4	67
2000	3 Doors Down	3:53	Kryptonite	2000-05-06	5	66

Table 8: First fifteen rows of the tidied Billboard dataset. The `date` column does not appear in the original table, but can be computed from `date.entered` and `week`.

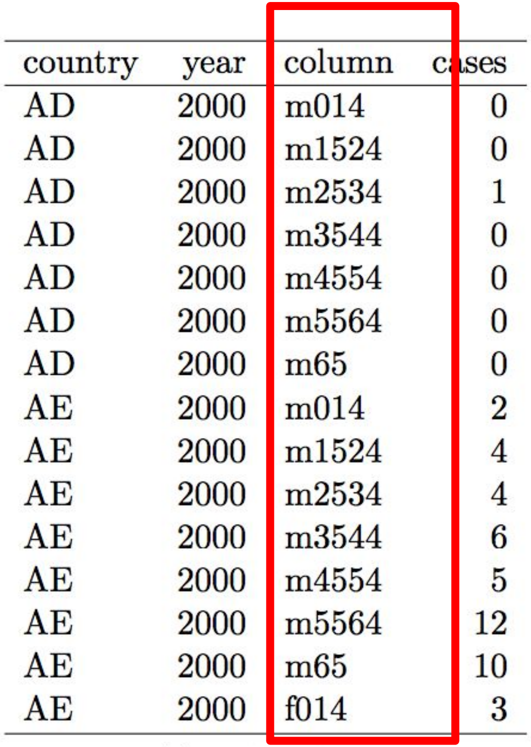
Stack/Melt: turn column into rows

Long vs Wide formats

Tidy data version results in fewer columns but increased duplication.

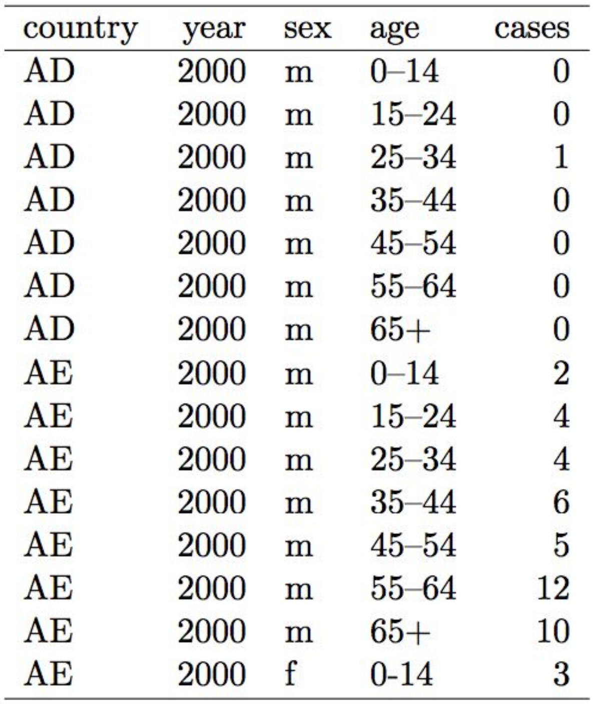
SQL analogy → data normalization vs denormalization

Multiple variables stored in one column



country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data



country	year	sex	age	cases
AD	2000	m	0–14	0
AD	2000	m	15–24	0
AD	2000	m	25–34	1
AD	2000	m	35–44	0
AD	2000	m	45–54	0
AD	2000	m	55–64	0
AD	2000	m	65+	0
AE	2000	m	0–14	2
AE	2000	m	15–24	4
AE	2000	m	25–34	4
AE	2000	m	35–44	6
AE	2000	m	45–54	5
AE	2000	m	55–64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

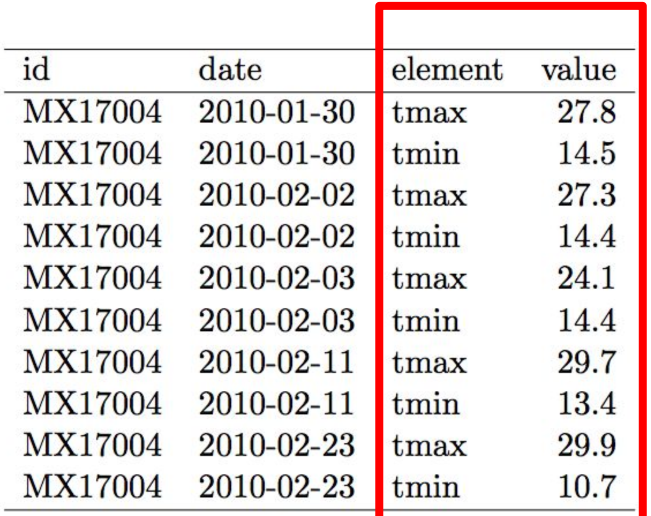
(b) Tidy data

Column is split into sex and age

E.g, M014 → m & 0-14

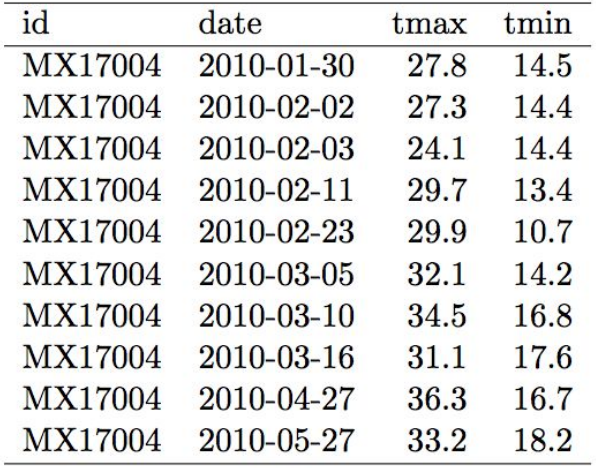
Usually occurs after a melting operation.

Variables are stored in both rows and columns



id	date	element	value
MX17004	2010-01-30	tmax	27.8
MX17004	2010-01-30	tmin	14.5
MX17004	2010-02-02	tmax	27.3
MX17004	2010-02-02	tmin	14.4
MX17004	2010-02-03	tmax	24.1
MX17004	2010-02-03	tmin	14.4
MX17004	2010-02-11	tmax	29.7
MX17004	2010-02-11	tmin	13.4
MX17004	2010-02-23	tmax	29.9
MX17004	2010-02-23	tmin	10.7

(a) Molten data



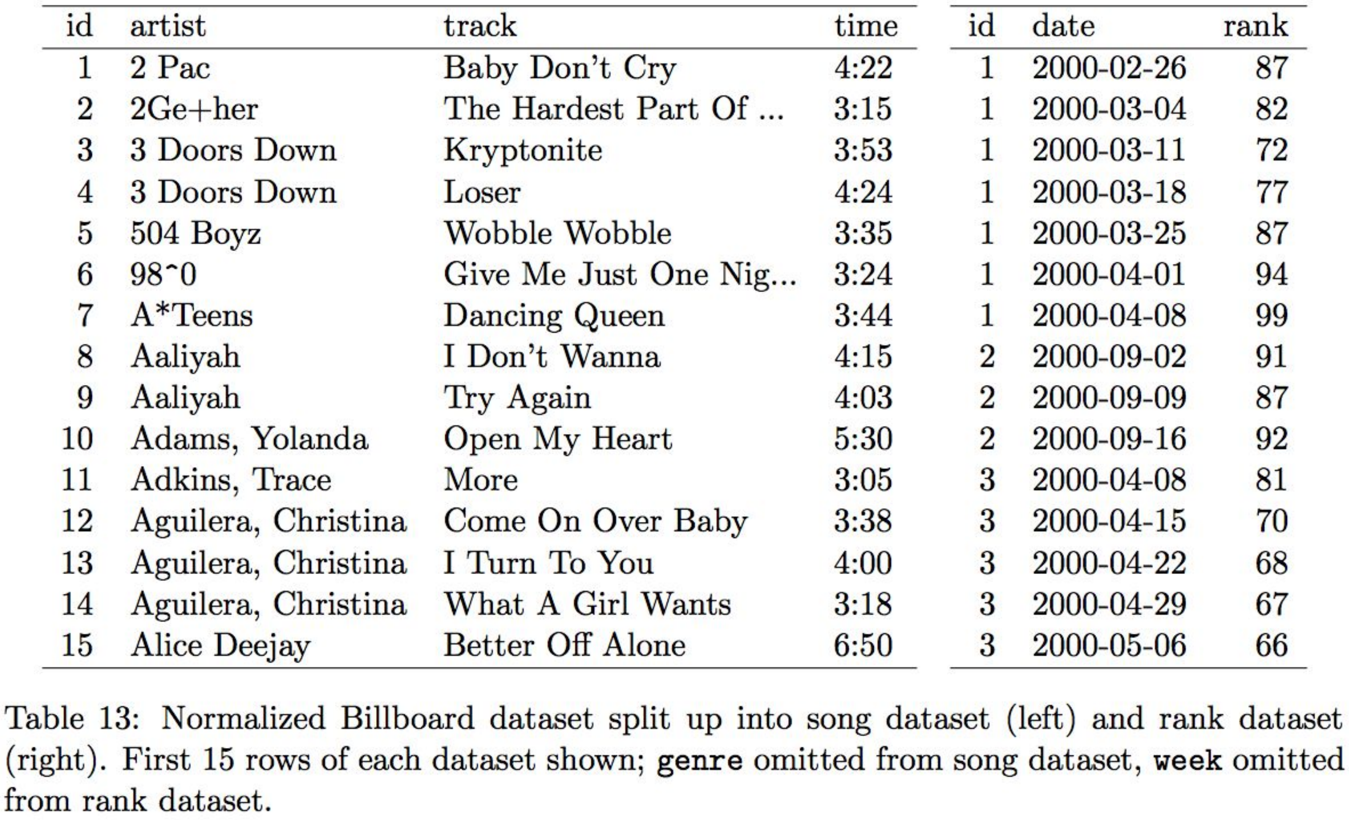
id	date	tmax	tmin
MX17004	2010-01-30	27.8	14.5
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-02-11	29.7	13.4
MX17004	2010-02-23	29.9	10.7
MX17004	2010-03-05	32.1	14.2
MX17004	2010-03-10	34.5	16.8
MX17004	2010-03-16	31.1	17.6
MX17004	2010-04-27	36.3	16.7
MX17004	2010-05-27	33.2	18.2

(b) Tidy data

Table in a) is “almost” tidy, two variables stored in rows tmin and tmax

Replace element and value with tmax and tmin

Multiple types of observational units are stored in the same table.



id	artist	track	time
1	2 Pac	Baby Don't Cry	4:22
2	2Ge+her	The Hardest Part Of ...	3:15
3	3 Doors Down	Kryptonite	3:53
4	3 Doors Down	Loser	4:24
5	504 Boyz	Wobble Wobble	3:35
6	98^0	Give Me Just One Nig...	3:24
7	A*Teens	Dancing Queen	3:44
8	Aaliyah	I Don't Wanna	4:15
9	Aaliyah	Try Again	4:03
10	Adams, Yolanda	Open My Heart	5:30
11	Adkins, Trace	More	3:05
12	Aguilera, Christina	Come On Over Baby	3:38
13	Aguilera, Christina	I Turn To You	4:00
14	Aguilera, Christina	What A Girl Wants	3:18
15	Alice Deejay	Better Off Alone	6:50

id	date	rank
1	2000-02-26	87
1	2000-03-04	82
1	2000-03-11	72
1	2000-03-18	77
1	2000-03-25	87
1	2000-04-01	94
1	2000-04-08	99
2	2000-09-02	91
2	2000-09-09	87
2	2000-09-16	92
3	2000-04-08	81
3	2000-04-15	70
3	2000-04-22	68
3	2000-04-29	67
3	2000-05-06	66

Table 13: Normalized Billboard dataset split up into song dataset (left) and rank dataset (right). First 15 rows of each dataset shown; `genre` omitted from song dataset, `week` omitted from rank dataset.

Related to DB normalization.

Relational databases.

Reduce duplication.

Compare Table 13 to Table 8 on slide 9.

Table 8 stores song details and ranking in a single table.

One type in multiple tables

Observations spread over multiple files or tables (e.g., a separate table of an individual’s medical history for each year of their life.)

If the format is consistent, just combine the data into a single table using the **plyr** package.

Perform additional tidying as needed.

More complex example: https://github.com/hadley/data-fuel-economy

EPA fuel economy data for 50,000 cars from 1978-2008.

Dataset structure also changes over time (inconsistent format)

Requires tidying each file individually before combining into a single dataset.

Regular expressions in R

Origins: Automata theory, theoretical computer science and formal languages

Pattern matching in text editors & lexical analysis in compilers.

Problem: For a dataset of products, find all products which have MX or US as the last two characters of their serial number.

In **R**: use **grep**, syntax: **grep ([regex], [input vector], ..)**

Other options: **grep, grepl, regexpr, gregexpr and regexec (differ in format of and amount of detail in the results)**

see: https://www.regular-expressions.info/rlanguage.html

Supports many character classes such as [0-9], [a-z], [A-Z], non-digits [^0-9] and many others

Quantifiers:

*****: matches at least 0 times.

+: matches at least 1 times.

?: matches at most 1 times.

{n}: matches exactly n times.

{n,}: matches at least n times.

{n,m}: matches between n and m times.

value = True returns matches instead of indices

```
(strings <- c("a", "ab", "acb", "accb", "acccb", "accccb"))

grep("ac*b", strings, value = TRUE)

## [1] "ab"     "acb"    "accb"   "acccb"  "accccb"

grep("ac+b", strings, value = TRUE)

## [1] "acb"    "accb"   "acccb"  "accccb"

grep("ac?b", strings, value = TRUE)

## [1] "ab"  "acb"

grep("ac{2}b", strings, value = TRUE)

## [1] "accb"

grep("ac{2,}b", strings, value = TRUE)   what will this return ?

grep("ac{2,3}b", strings, value = TRUE)  what will this return ?
```

Pattern positions:

^: matches the start of the string.

$: matches the end of the string.

There are a few more operators but we are focusing on these two

```
(strings <- c("abcd", "cdab", "cabd", "c abd", "cabdd"))

## [1] "abcd"  "cdab"  "cabd"  "c abd"  "cbdd"

grep("ab", strings, value = TRUE)

## [1] "abcd"  "cdab"  "cabd"  "c abd"  "cbdd"

grep("^ab", strings, value = TRUE)

## [1] "abcd"

grep("ab$", strings, value = TRUE)

## [1] "cdab"

grep("(^c)d?", strings, value = TRUE) → what will this return?

grep("(^c)d+", strings, value = TRUE) → what will this return?
```

Operators:

. : matches any single character,

[...] : a character list, matches any one of the characters inside the square brackets.

[^...] : an inverted character list, similar to [...], but matches any characters except those inside the square brackets.

\ : suppress the special meaning of metacharacters

| : an “or” operator, matches patterns on either side of the |.

```
(strings <- c("^ab", "ab", "abc", "abd", "abe", "ab 12"))

grep("ab.", strings, value = TRUE)

## [1] "abc"   "abd"   "abe"   "ab 12"

grep("ab[c-e]", strings, value = TRUE)

## [1] "abc" "abd" "abe"

grep("ab[^c]", strings, value = TRUE)

## [1] "abd"   "abe"   "ab 12"

grep("^ab", strings, value = TRUE)

## [1] "ab"    "abc"   "abd"   "abe"   "ab 12"

grep("\\^ab", strings, value = TRUE)          ## [1] "^ab"

grep("abc|abd", strings, value = TRUE)  → what will this return?
```

Gathering & preparing data

Support for importing from other stat packages such as SPSS, SAS, or Stata

```
library (foreign)
```

```
StataData ← read.dta(file=”Data1.dta”)
```

Data from Secure (https) URLs

```
library(repmis)
```

On github every file has a SHA-1 hash which will change if the file is changed → ensures that end users of data are aware of changes.

Use ***tidyr’s* gather** function to transform a dataframe from wide to long format (also matrix transpose **(t)**[2] and **reshape** functions in base R)

Use ***dplyr’s* rename** function to rename variables:

```
GatheredFert ← rename(GatheredFert, year = Year, FertilizerConsumption = Fert)
```

and **arrange** function to order data (similar to sort order):

```
GatheredFert ← arrange(GatheredFert, country, year)
```

Getting subsets of a dataframe using the **subset** function:

```
FertOutliers ← subset(x = GatheredFert, FertilizerConsumption > 1000)
```

Recoding values:

```
# Recode country == "Korea, Rep."" to "South Korea"
GatheredFertSub$country[GatheredFertSub$country == "Korea, Rep."] ←"South Korea"
```

**note that normalization/classification machine learning approaches are usually used for these in large scale projects*

Creating new variables from existing variables:

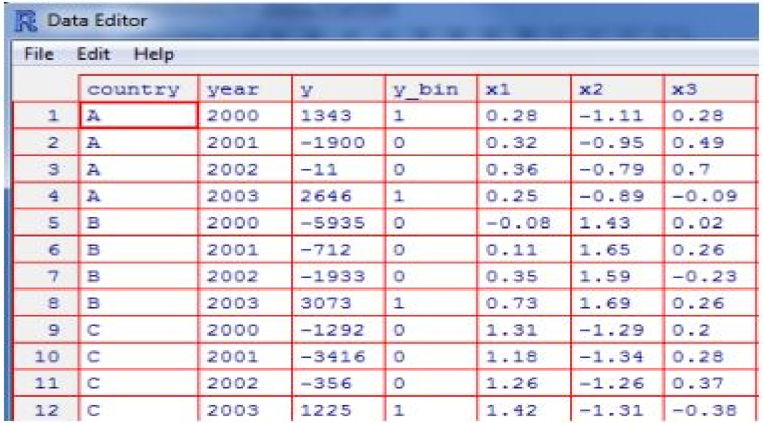
```
GatheredFertSub$logFertConsumption ← log( GatheredFertSub$FertilizerConsumption)
```

(note: may result in -Inf or Inf values for values such as 0.

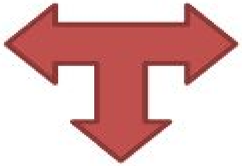
Solution: recode zeros as 0.001)

Merge (very similar to a database join): datasets should have at least one variable in common. Parameter **all=FALSE** (only matching rows are returned) ref. princeton

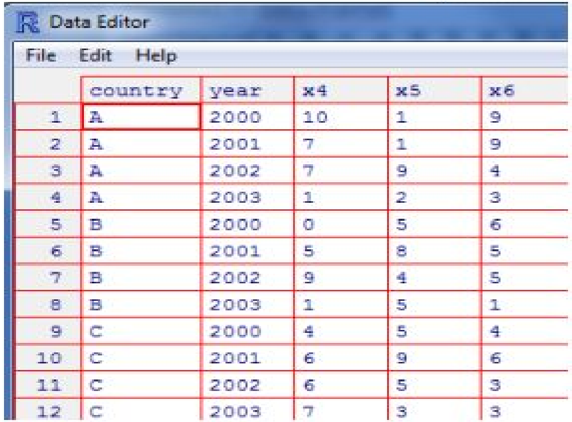
mydata1



	country	year	y	y_bin	x1	x2	x3
1	A	2000	1343	1	0.28	-1.11	0.28
2	A	2001	-1900	0	0.32	-0.95	0.49
3	A	2002	-11	0	0.36	-0.79	0.7
4	A	2003	2646	1	0.25	-0.89	-0.09
5	B	2000	-5935	0	-0.08	1.43	0.02
6	B	2001	-712	0	0.11	1.65	0.26
7	B	2002	-1933	0	0.35	1.59	-0.23
8	B	2003	3073	1	0.73	1.69	0.26
9	C	2000	-1292	0	1.31	-1.29	0.2
10	C	2001	-3416	0	1.18	-1.34	0.28
11	C	2002	-356	0	1.26	-1.26	0.37
12	C	2003	1225	1	1.42	-1.31	-0.38



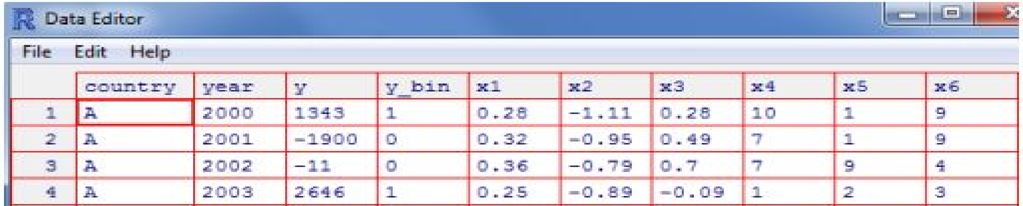
mydata2



	country	year	x4	x5	x6
1	A	2000	10	1	9
2	A	2001	7	1	9
3	A	2002	7	9	4
4	A	2003	1	2	3
5	B	2000	0	5	6
6	B	2001	5	8	5
7	B	2002	9	4	5
8	B	2003	1	5	1
9	C	2000	4	5	4
10	C	2001	6	9	6
11	C	2002	6	5	3
12	C	2003	7	3	3

```
mydata <- merge(mydata1, mydata2, by=c("country","year"))

edit(mydata)
```

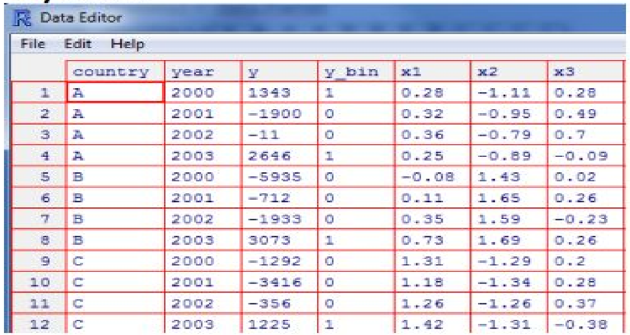


	country	year	y	y_bin	x1	x2	x3	x4	x5	x6
1	A	2000	1343	1	0.28	-1.11	0.28	10	1	9
2	A	2001	-1900	0	0.32	-0.95	0.49	7	1	9
3	A	2002	-11	0	0.36	-0.79	0.7	7	9	4
4	A	2003	2646	1	0.25	-0.89	-0.09	1	2	3

When all=TRUE (include all data from both datasets)

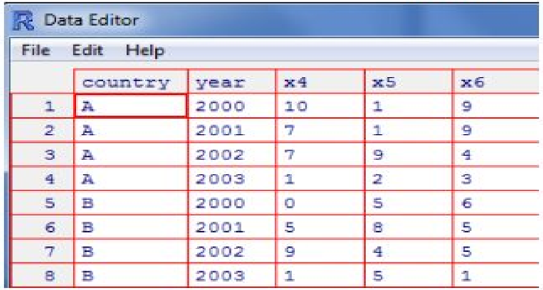
MERGE – EXAMPLE 2 (cont.) – including all data from both datasets

mydata1



	country	year	y	y_bin	x1	x2	x3
1	A	2000	1343	1	0.28	-1.11	0.28
2	A	2001	-1900	0	0.32	-0.95	0.49
3	A	2002	-11	0	0.36	-0.79	0.7
4	A	2003	2646	1	0.25	-0.89	-0.09
5	B	2000	-5935	0	-0.08	1.43	0.02
6	B	2001	-712	0	0.11	1.65	0.26
7	B	2002	-1933	0	0.35	1.59	-0.23
8	B	2003	3073	1	0.73	1.69	0.26
9	C	2000	-1292	0	1.31	-1.29	0.2
10	C	2001	-3416	0	1.18	-1.34	0.28
11	C	2002	-356	0	1.26	-1.26	0.37
12	C	2003	1225	1	1.42	-1.31	-0.38

mydata3

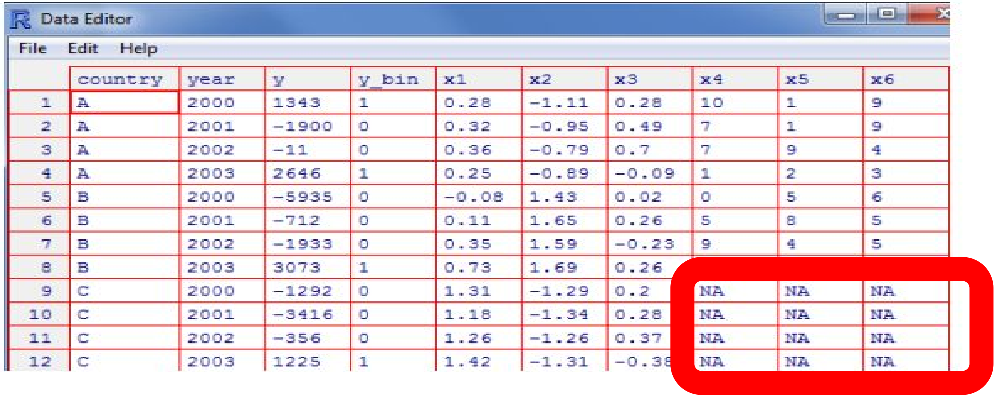


	country	year	x4	x5	x6
1	A	2000	10	1	9
2	A	2001	7	1	9
3	A	2002	7	9	4
4	A	2003	1	2	3
5	B	2000	0	5	6
6	B	2001	5	8	5
7	B	2002	9	4	5
8	B	2003	1	5	1

Adding the option “`all=TRUE`” includes all cases from both datasets.

```
mydata <- merge(mydata1, mydata3, by=c("country","year"), all=TRUE)
```

```
edit(mydata)
```



	country	year	y	y_bin	x1	x2	x3	x4	x5	x6
1	A	2000	1343	1	0.28	-1.11	0.28	10	1	9
2	A	2001	-1900	0	0.32	-0.95	0.49	7	1	9
3	A	2002	-11	0	0.36	-0.79	0.7	7	9	4
4	A	2003	2646	1	0.25	-0.89	-0.09	1	2	3
5	B	2000	-5935	0	-0.08	1.43	0.02	0	5	6
6	B	2001	-712	0	0.11	1.65	0.26	5	8	5
7	B	2002	-1933	0	0.35	1.59	-0.23	9	4	5
8	B	2003	3073	1	0.73	1.69	0.26			
9	C	2000	-1292	0	1.31	-1.29	0.2	NA	NA	NA
10	C	2001	-3416	0	1.18	-1.34	0.28	NA	NA	NA
11	C	2002	-356	0	1.26	-1.26	0.37	NA	NA	NA
12	C	2003	1225	1	1.42	-1.31	-0.38	NA	NA	NA

What did you learn today?

Questions?