Data Manipulation

Fabrice Rossi

CEREMADE
Université Paris Dauphine

2021

Data Manipulation

In this course

- tabular data
- elementary extension to multiple-table data
- data transformation
  - wrangling
  - filtering
  - ordering
- data aggregation and summary
- tidy data and reshaping

In other courses

- database management system
- data models
- relational data
- unstructured data

Data Model

In this course

- a data set is
  - a (finite) set of entities (a.k.a. objects, instances, subjects)
  - each entity is described by its values with respect to a fix set of variables (a.k.a. attributes)
- in practice a data set is a *table* with
  - a row per entity
  - a column per variable

Extension

- multiple-table data
- a data set = several tables

Example

	age	job	marital	education	default	balance	housing
1	30	unemployed	married	primary	no	1787	no
2	33	services	married	secondary	no	4789	yes
3	35	management	single	tertiary	no	1350	yes
4	30	management	married	tertiary	no	1476	yes
5	59	blue-collar	married	secondary	no	0	yes
6	35	management	single	tertiary	no	747	no
7	36	self-employed	married	tertiary	no	307	yes
8	39	technician	married	secondary	no	147	yes
9	41	entrepreneur	married	tertiary	no	221	yes
10	43	services	married	primary	no	-88	yes
11	39	services	married	secondary	no	9374	yes
12	43	admin.	married	secondary	no	264	yes
13	36	technician	married	tertiary	no	1109	no
14	20	student	single	secondary	no	502	no
15	31	blue-collar	married	secondary	no	360	yes
16	40	management	married	tertiary	no	194	no
17	56	technician	married	secondary	no	4073	no
18	37	admin.	single	tertiary	no	2317	yes
19	25	blue-collar	single	primary	no	-221	yes
20	31	services	married	secondary	no	132	no

Variable types

Numerical

- essentially “physical” measurements
- integer or decimal
- easier to handle than the other types

Categorical

- a.k.a. Nominal (factors and levels in R)
- finite number of values (called categories or modalities)
- might be ordered

Dates and times

- very important in numerous applications
- notoriously difficult to handle
- use specific libraries!

Short texts

- a.k.a. strings
- could be handled as categorical data
- specific processing in some cases
- do not confuse them with full texts

Example

Bank dataset

- sources
  - https://archive.ics.uci.edu/ml/datasets/Bank%2BMarketing
  - http://hdl.handle.net/1822/14838
- data types
  - age: integer
  - balance: integer
  - education: categorical semi ordered
  - most of the others: categorical with some binary

Data manipulation software

- typical examples: R with tidyverse or python with pandas
- limited automatic support for enforcing complex data models
  - declarative support for broad types
  - constraints can be checked explicitly
    - ☺ very complex constraints can be enforced
    - ☹ error/bug prone
    - ☹ difficult to read
- documentation is needed

Outline

Introduction

Data transformation

Data grouping and summarizing

Tidy data

Multiple data tables

Subsetting

Working on a subpopulation

- a.k.a by “removing” rows
- several motivations
  - speed (for large data sets)
  - robustness (removing outliers)
  - modeling
- generally called *filtering*
- declarative approach in R and python
  - give me the subset of the data that fulfills some conditions
  - supported by comparison and Boolean operators

Example

Bank data set

Married clients with secondary education level in their thirties (age between 30 and 39 included)

Example

Bank data set

Married clients with secondary education level in their thirties (age between 30 and 39 included)

Python (pandas)

```
bank[ (bank.marital == 'married') &
      (bank.education == 'secondary') &
      (bank.age >= 30) &
      (bank.age < 40) ]
```

R (dplyr)

```
bank %>% filter(marital == "married",
                education == "secondary",
                age >= 30,
                age < 40)
```

Computational considerations

Running time

- filtering is a row oriented operation
- naive implementation
  - browse the data row by row
  - keep a row if the conditions are fulfilled
- run time proportional to the number of rows in the data
- can be improved in some cases via *indexing*

Computational considerations

Running time

- filtering is a row oriented operation
- naive implementation
  - browse the data row by row
  - keep a row if the conditions are fulfilled
- run time proportional to the number of rows in the data
- can be improved in some cases via *indexing*

Do not program it yourself!

- far less efficient
- less readable

Dropping Variables

Column oriented subsetting

- two main motivations
  - to restrict the data set to variable types compatible with some technique
  - to restrict the data set to meaningful variables for an automated analysis (e.g. clustering or predictive modeling)
- simple declarative approach

Example

Bank data set

Keep some numerical variables

Example

Bank data set

Keep some numerical variables

Python (pandas)

```
bank[['age', 'balance', 'day', 'duration']]
```

R (dplyr)

```
bank %>% select(age, balance, day, duration)
```

Ordering

Sorting

- standard sorting feature
- multiple criteria

Python

```
bank.sort_values(by=['age',
                     'balance'])
```

R

```
bank %>% arrange(age, balance)
```

Transformation of Variables

Variable Operations

- modifying a variable
- adding new variables from other sources
- computing new variables (based on existing ones)

Data Wrangling

- low level transformation
- recoding
- extraction and merging
- etc.

Data Management

- context variables
- enforcing data model

Preparing Analysis

- e.g. recoding categorical to numerical
- or quantifying numerical variables
- scaling, normalization
- merging categories

Computed variables

Principle

- row oriented calculation
- create a new variable using the existing ones
- e.g. duration from starting and ending times
- combines nicely with aggregation/summary functions

Support

- numerous statistical summary functions (column oriented)
- column oriented arithmetic (e.g. sum of columns)
- column oriented logical operations (e.g. comparison)
- function application (e.g. to each row)

Example

Bank data set

Binary variable telling whether some client has some characteristics

- more than average mean annual balance
- at least one loan

Python

```
bank['moreavg'] = bank['balance'] > bank['balance'].mean()
bank['oneormoreloan'] = (bank['loan'] == 'yes') | (bank['housing'] == 'yes')
```

R

```
bank <- bank %>% mutate(moreavg = balance > mean(balance))
bank <- bank %>% mutate(oneormoreloan = loan=="yes" | housing=="yes")
```

Example

One Hot Encoding

- typical preparatory transformation
- categorical variable turn into a set of binary variables
- e.g. Gender=Male or Female transformed into GenderMale and GenderFemale (binary)

Python

```
bank.join(pd.get_dummies(bank['education']))
bank.join(pd.get_dummies(bank['education'])).drop('education',axis=1)
```

R

```
bank %>%
    bind_cols(as.data.frame(model.matrix(~education-1,data=bank))) %>%
    select(-education)
```

Example

	age	job	marital	education	default	balance	housing
1	30	unemployed	married	primary	no	1787	no
2	33	services	married	secondary	no	4789	yes
3	35	management	single	tertiary	no	1350	yes
4	30	management	married	tertiary	no	1476	yes
5	59	blue-collar	married	secondary	no	0	yes
6	35	management	single	tertiary	no	747	no
7	36	self-employed	married	tertiary	no	307	yes
8	39	technician	married	secondary	no	147	yes
9	41	entrepreneur	married	tertiary	no	221	yes
10	43	services	married	primary	no	-88	yes
11	39	services	married	secondary	no	9374	yes
12	43	admin.	married	secondary	no	264	yes
13	36	technician	married	tertiary	no	1109	no
14	20	student	single	secondary	no	502	no
15	31	blue-collar	married	secondary	no	360	yes
16	40	management	married	tertiary	no	194	no
17	56	technician	married	secondary	no	4073	no
18	37	admin.	single	tertiary	no	2317	yes
19	25	blue-collar	single	primary	no	-221	yes
20	31	services	married	secondary	no	132	no

Example

	age	job	marital	primary	secondary	tertiary	unknown
1	30	unemployed	married	1	0	0	0
2	33	services	married	0	1	0	0
3	35	management	single	0	0	1	0
4	30	management	married	0	0	1	0
5	59	blue-collar	married	0	1	0	0
6	35	management	single	0	0	1	0
7	36	self-employed	married	0	0	1	0
8	39	technician	married	0	1	0	0
9	41	entrepreneur	married	0	0	1	0
10	43	services	married	1	0	0	0
11	39	services	married	0	1	0	0
12	43	admin.	married	0	1	0	0
13	36	technician	married	0	0	1	0
14	20	student	single	0	1	0	0
15	31	blue-collar	married	0	1	0	0
16	40	management	married	0	0	1	0
17	56	technician	married	0	1	0	0
18	37	admin.	single	0	0	1	0
19	25	blue-collar	single	1	0	0	0
20	31	services	married	0	1	0	0

Improving Representation

Data Management

- convert values to proper types, e.g.
  - integer only to language supported integer
  - data as string to language supported date
  - string to semantic content
- proprer encoding of missing data
- add diagnostic data

Nominal data

- important particular case
- frequently represented by strings
  - loss of efficiency
  - cannot leverage automatic handling (in R in particular)

Example

Data Representation

- make sure that nominal variables are recognized as such
- yes/no nominal variables can be encoded as logical variables

Python

```
for myvar in ['job', 'marital', 'education']:
    bank[myvar] = bank[myvar].astype('category')
for myvar in ['default', 'housing', 'loan']:
    bank[myvar] = bank[myvar] == 'yes'
```

R

```
bank <- bank %>% mutate_at(vars(job,marital,education), ~(as.factor(.)))
bank <- bank %>% mutate_at(vars(default,housing,loan), ~(. == "yes"))
```

Example

Unknown values

- specific “unknown” category in the bank dataset (for several variables)
- not considered “special” by the software while it should be

Python

Numpy provides a special value `nan` for not available

```
bank.replace("unknown",np.nan)
```

R

Similar special value **`NA`**

```
bank %>%  mutate_all(~replace(., . == "unknown", NA))
```

Outline

Conditional analysis

Finding dependencies and links

One of the main goal of data analysis, e.g.

- predictive models: links between target variables and explanatory variables
- frequent patterns: variables that are frequently non zero at the same time
- etc.

Conditional summaries

- chose one or more variables
- for each possible combination of the values of the chosen variables
  - find all corresponding objects in the data set
  - compute a summary of the other variables on this subset

Mechanism

X	Y
2	A
1	B
4	B
3	B
1	B
2	C
1	A
3	C
3	A
2	A
2	A
3	A
3	B
1	A
1	A

Split

Y	X
A	2
A	1
A	3
A	2
A	2
A	3
A	1
A	1

Y	X
B	1
B	4
B	3
B	1
B	3

Y	X
C	2
C	3

Apply

Y	Sum
A	15

Y	Sum
B	12

Y	Sum
C	5

Combine

Y	Sum
A	15
B	12
C	5

Example

Bank dataset

- balance conditioned on the response to the marketing campaign
- age conditioned on marital status and education level

Python

```
bank.groupby('y')['balance'].median()
bank.groupby(['marital', 'education'])['age'].mean()
```

R

```
bank %>% group_by(y) %>% summarise(median_balance = median(balance),
    .groups = "drop_last")
bank %>% group_by(marital, education) %>%
    summarise(mean_age = mean(age), .groups = "drop_last")
```

Example

Balance versus marketing

y	median_balance
no	419.50
yes	710.00

Age versus marital and education

marital	education	mean_age
divorced	primary	51.39
divorced	secondary	43.50
divorced	tertiary	45.15
divorced	missing	50.38
married	primary	47.51
married	secondary	42.40
married	tertiary	41.78
married	missing	48.44
single	primary	37.01
single	secondary	33.05
single	tertiary	34.51
single	missing	34.65

Pivot Table

Conditioning by two variables

- useful special case
- we can leverage standard tabular representation
  - compute a standard aggregated table with two conditioning variables and a single aggregate
  - “pivot” the table
    - remove one of conditioning variable and the aggregate
    - creates as many columns as they are values of the removed variable
    - use the aggregate to populate cells

Mechanism

X	Y	Z
2	B	U
2	A	V
2	C	V
1	B	V
4	A	V
3	A	U
1	C	U
4	B	U
3	B	V
2	C	V
2	C	U
2	B	U
2	B	V
3	B	V
4	B	V

S-A-C

Y	Z	Sum
A	U	3
A	V	6
B	U	8
B	V	13
C	U	3
C	V	4

Pivot

Y/Z	U	V
A	3	6
B	8	13
C	3	4

Example

Age versus marital and education

marital	education	mean_age
divorced	primary	51.39
divorced	secondary	43.50
divorced	tertiary	45.15
divorced	missing	50.38
married	primary	47.51
married	secondary	42.40
married	tertiary	41.78
married	missing	48.44
single	primary	37.01
single	secondary	33.05
single	tertiary	34.51
single	missing	34.65

marital/education	primary	secondary	tertiary	missing
divorced	51.39	43.50	45.15	50.38
married	47.51	42.40	41.78	48.44
single	37.01	33.05	34.51	34.65

Howto

Python

Specific support for Pivot tables

```
bank.pivot_table('age', index = 'marital', columns = 'education')
```

R

A particular case of table reshaping

```
bank %>% group_by(marital, education) %>%
    summarise(mean_age = mean(age), .groups = "drop_last") %>%
    spread(key = education, value = mean_age)
```

Multidimensional analysis (MDA)

Pivot (hyper)cube

- a pivot table with more than 2 “dimensions” (e.g. a pivot cube)
- specific vocabulary:
  - a dimension: a variable with a finite set of possible values
  - a measure: a numerical variable
  - a cell contains aggregate values for objects with given values for the dimension
- ☹ a very convoluted way of presenting conditional analysis
- ☺ rich possibilities when the set of values of a “dimension” is structured (e.g. postcodes)
- ☺ rich support with specific OLAP software (not in this part of the course)

Example

Bank data set

- Possible dimensions: job, marital, education, housing, loan
- Possible measures: age and balance
- A cell (such as unemployed, married, primary education, no housing and no loan) contains the average age and the median balance for the persons with the specified values on the dimensions

Tabular point of view

job	marital	education	housing	loan	mean_age	median_balance
admin.	divorced	primary	FALSE	FALSE	57.00	1.00
admin.	divorced	primary	FALSE	TRUE	56.00	0.00
admin.	divorced	primary	TRUE	FALSE	57.00	179.00
admin.	divorced	secondary	FALSE	FALSE	41.80	432.00
admin.	divorced	secondary	FALSE	TRUE	45.83	175.00

and 337 additional rows...

Example

Tabular view

y	housing	loan	n
no	FALSE	FALSE	1394
no	FALSE	TRUE	267
no	TRUE	FALSE	1958
no	TRUE	TRUE	381
yes	FALSE	FALSE	283
yes	FALSE	TRUE	18
yes	TRUE	FALSE	195
yes	TRUE	TRUE	25

MDA view

y="no"

housing/loan	FALSE	TRUE
FALSE	1394	267
TRUE	1958	381

y="yes"

housing/loan	FALSE	TRUE
FALSE	283	18
TRUE	195	25

arranged as a cube!

Outline

Introduction

Data transformation

Data grouping and summarizing

Tidy data

Multiple data tables

Tidy Data

Concept introduced by Hadley Wickham the Tidy Data paper

Definition

A data set made of several data tables is tidy if

1. each variable forms a column of a table
2. each observation forms a row of a table
3. each type of observational unit forms a table

Observational unit

- a particular type of observations in a data set
- e.g.
  - persons
  - daily behavior of persons
  - etc.

Example

Bank data set

- mixes person information and marketing campaign information (and economic variables in an extended version!)
- marketing campaign data
  - last contact data
  - contacts during the campaign
  - summary of previous campaigns!
- clearly untidy
  - violates property 3
  - multiple types of observational unit in a single table

Example

Pivot table

housing/loan	FALSE	TRUE
FALSE	1677	285
TRUE	2153	406

- intrinsically untidy
- columns are not variables but variable values!

Example

Pivot table

housing/loan	FALSE	TRUE
FALSE	1677	285
TRUE	2153	406

- intrinsically untidy
- columns are not variables but variable values!

Summary table

loan	housing	n
FALSE	FALSE	1677
FALSE	TRUE	2153
TRUE	FALSE	285
TRUE	TRUE	406

- tidy data
- new observational unit: groups of observations!

Splitting or Joining

- observational unit level
- splitting
  - separates different observational unit into several tables
  - filtering/selecting + linking
- joining
  - merges several tables about the same observational unit
  - specific tools (see the last part of this course)

Tidying Data

Gathering or Spreading

- variable/observation level
- specific tools
- gathering
  - reduces the number of columns
  - merges several columns in a single one that corresponds to a proper variable
- spreading
  - increases the number of columns
  - splits a column into several ones that correspond to proper variables

Example (Splitting)

Splitting bank marketing data

Steps:

1. add an identifier to each row (to identify clients)
2. select variables associated to each observational unit, keeping the id
   - 2.1 persons
   - 2.2 current campaign (i.e. last contact)
   - 2.3 previous campaigns
3. clean the tables (remove useless rows, e.g. when no previous campaign is available)

Example (Splitting)

Python

- pandas data frames have always row identifiers
- splitting and cleaning

```
bank_persons = bank[['age', 'job', 'marital', 'education', 'default',
                     'balance', 'housing', 'loan']]
bank_current = bank[['contact', 'day', 'month', 'duration',
                     'campaign']]
bank_previous = bank[bank['pdays'] != -1][['pdays', 'previous',
                                           'poutcome']]
```

Example (Splitting)

R

1. identifier

```
bank.tidy <- bank %>% mutate(key = 1:nrow(bank))
```

2. selection with cleanup

```
bank.persons <- bank.tidy %>% select(key, age, job, marital,
    education, default, balance, housing, loan)
bank.current <- bank.tidy %>% select(key, contact, day, month,
    duration, campaign)
bank.previous <- bank.tidy %>% filter(pdays != -1) %>% select(key,
    pdays, previous, poutcome)
```

Spreading data

Replace variable encoded on rows by columns

- operates on two variables in the original table: a key and a value
- each value taken by the key becomes a column
- the value variable is used to fill the column

Original table

X	Y	Z
1	A	2
1	B	3
2	A	4
2	B	5

Spread table

Y is the key, Z is the value

X	A	B
1	2	3
2	4	5

Example (Spreading)

CallIt2 dataset

- flow (in number of persons) in and out a building
- direction encoded in the flow column

flow	date	time	count
7	2005-07-24	00:00:00	0
9	2005-07-24	00:00:00	0
7	2005-07-24	00:30:00	1
9	2005-07-24	00:30:00	0
7	2005-07-24	01:00:00	0
9	2005-07-24	01:00:00	0
7	2005-07-24	01:30:00	0
9	2005-07-24	01:30:00	0
7	2005-07-24	02:00:00	0
9	2005-07-24	02:00:00	0

- untidy: flow is not a variable!
- spreading is needed

Example (Spreading)

CallIt2 dataset

- flow (in number of persons) in and out a building
- tidy version

date	time	entering	leaving
2005-07-24	00:00:00	0	0
2005-07-24	00:30:00	1	0
2005-07-24	01:00:00	0	0
2005-07-24	01:30:00	0	0
2005-07-24	02:00:00	0	0
2005-07-24	02:30:00	2	0
2005-07-24	03:00:00	0	0
2005-07-24	03:30:00	0	0
2005-07-24	04:00:00	0	0
2005-07-24	04:30:00	0	0

Example (Spreading)

Python

- spreading can be done by leveraging the indexing system
- hierarchical indexing in this case

```
calit.set_index(['date', 'time'], inplace = True)
tcalit = calit.pivot(columns='flow')
tcalit.columns = tcalit.columns.to_flat_index()
tcalit.rename(columns={('count', 7):'entering',
                       ('count', 9): 'leaving'},
              inplace=True)
tcalit.reset_index(inplace = True)
```

- this is somewhat convoluted

Example (Spreading)

Python

- spreading can be also be done with `pivot_table`

```
tcalit = calit.pivot_table(values='count',
                           index=['date', 'time'],
                           columns='flow')
tcalit.rename(columns={7: 'entering', 9: 'leaving'},
              inplace=True)
tcalit.reset_index(inplace=True)
```

- a bit simpler

R

Standard use of the spread function

```
calit %>% spread(flow, count) %>% rename(entering = `7`, leaving = `9`)
```

Gathering data

Gather

- gathering is the reverse of spreading
- it reduces the number of columns by encoding them as a series of rows and two new columns/variables
- the new key variable encode the gathered columns while the new value variable contains the original value

Original table

Gather X, Y and Z

W	X	Y	Z
a	1	2	3
b	5	6	7

Gathered table

W	K	V
a	X	1
a	Y	2
a	Z	3
b	X	5
b	Y	6
b	Z	7

Example (Gathering)

Sales transaction data set

- weekly product sales
- one row per product, one column per week: product view

Product_Code	W0	W1	W2	W3	W4	W5
P1	11.00	12.00	10.00	8.00	13.00	12.00
P2	7.00	6.00	3.00	2.00	7.00	1.00
P3	7.00	11.00	8.00	9.00	10.00	8.00
P4	12.00	8.00	13.00	5.00	9.00	6.00
P5	8.00	5.00	13.00	11.00	6.00	7.00
P6	3.00	3.00	2.00	7.00	6.00	3.00
P7	4.00	8.00	3.00	7.00	8.00	7.00
P8	8.00	6.00	10.00	9.00	6.00	8.00
P9	14.00	9.00	10.00	7.00	11.00	15.00
P10	22.00	19.00	19.00	29.00	20.00	16.00

with 53 columns and 811 rows

Example (Gathering)

Gather the weeks

- gather all the columns except the product one
- one observation: (product, week)

Product_Code	Week	Quantity
P1	1	11
P2	1	7
P3	1	7
P4	1	12
P5	1	8
P6	1	3
P7	1	4
P8	1	8
P9	1	14
P10	1	22

with 42162 more rows

Example (Gathering)

Python

Using the `melt` function

```
prodlong = pd.melt(prodperweek, 'Product_Code')
prodlong.rename(columns={'variable': 'Week',
                         'value': 'Quantity'},
                inplace=True)
prodlong['Week'] = pd.to_numeric(prodlong['Week'].str[1:])+1
```

R

Standard use of the gather function

```
prodperweek %>% gather(Week, Quantity, -Product_Code) %>%
    mutate(Week = parse_number(Week) + 1)
```

Tidy Data

Modeling assumptions

- data are tidy only with respect to some modeling assumptions
- what is an observation?
  - maybe the most important assumption
  - very frequently associated to an independence assumption

Practical aspects

- the data format must be adapted to the tool
- data mining and machine learning
  - generally limited to a single table
  - one might need to merge tables (e.g. bank data set)
- column oriented software (R)
  - have limited row oriented capabilities
  - might need a specific untidy representation

Tidy Data

Examples

- bank marketing: several object types or only one (person + marketing action)
- flow in/out a building
  - the half-hour bidirectional flow is an observation
  - or a day is an observation
- weekly sales
  - product point of view (original data set)
  - product $\times$ week point of view
  - week point of view

Outline

Multiple data tables

Tidy data

- one table per observational unit
- complex data
  - multiple observational units (e.g. persons and products)
  - multiple tables!

Difficulties

- the vast majority of data analysis methods are limited to single tables
- complex real world data use multiple table!
- a core data manipulation task: join multiple tables into data analysis oriented tables

Example

Loan application data set

- https://relational.fit.cvut.cz/dataset/Financial
- 8 tables including
  - client table
  - account table
  - credit card table
  - loan table
  - etc.
- open ended data set: no specific goal

Associations and keys

Relational data

- tables must be related one to another
- in the relational model a table is a relation
- some relations describe entities while others describe links between entities

Keys

- a key is a (set of) variable(s) that uniquely identifies an entity
- a primary key does that in the relation/table that describe the entity
- a foreign key does that in another relation/table

Loan application data set

Client table

client_id	gender	birth_date	district_id
1	F	1970-12-13	18
2	M	1945-02-04	1
3	F	1940-10-09	1
4	M	1956-12-01	5
5	F	1960-07-03	5

Keys

- primary key `client_id`
- foreign key `district_id`

Account table

account_id	district_id	date
1	18	1995-03-24
2	1	1993-02-26
3	5	1997-07-07
4	12	1996-02-21
5	15	1997-05-30

Keys

- primary key `account_id`
- foreign key `district_id`

Loan application data set

Disposition table

disp_id	client_id	account_id	type
1	1	1	OWNER
2	2	2	OWNER
3	3	2	DISPONENT
4	4	3	OWNER
5	5	3	DISPONENT

Keys

- primary key `disp_id`
- foreign keys `client_id` and `account_id`

Link table

- the disposition table/relation is a typical example of a link table
- it associates clients with accounts
- the link is also an entity as it has characteristics

Loan application data set

District table

district_id	Name	Region	Inhabitants
1	Hl.m. Praha	Prague	1204953
2	Benesov	central Bohemia	88884
3	Beroun	central Bohemia	75232
4	Kladno	central Bohemia	149893

Direct links

- no link table from accounts and clients to the district table
- `district_id` is used as a foreign key in the account and client tables

Joining tables

Main operations

- we need to build unique tables that gather information from separate ones
- this is done via *join* operations
  - identifying matching entities in two different tables
  - generating tables which combine variables from said tables for matched entities

Example

- joining client information with account information
- joining client information with district information

Basic case

Principle

- input
  - two tables A and B
  - A contains a variable V which is a foreign key
  - B contains the same variable V as its primary key
- result
  - a table with all the variables in A and B (no repeat)
  - such that each entity in A is merged with the entity in B referenced by the value of V

Example

Client with district information

- join the client table with the district table
- `district_id`: foreign key in the client table, primary key in the district table
- (part of the) result

client_id	gender	birth_date	district_id	Name	Region	Inhabitants
1	F	1970-12-13	18	Pisek	south Bohemia	70699
2	M	1945-02-04	1	Hl.m. Praha	Prague	1204953
3	F	1940-10-09	1	Hl.m. Praha	Prague	1204953
4	M	1956-12-01	5	Kolin	central Bohemia	95616
5	F	1960-07-03	5	Kolin	central Bohemia	95616

Python

```
pd.merge(client, district)
```

R

```
client %>% inner_join(district)
```

Common semantics: natural join

- common columns/variables are considered as a key

Join types

Missing keys

- missing foreign keys
- unreferenced foreign keys
- wrong foreign keys

How to build the join table?

- intersection approach
- missing data approach

Join types

Missing keys

- missing foreign keys
- unreferenced foreign keys
- wrong foreign keys

How to build the join table?

- intersection approach
- missing data approach

Inner join

- most common solution
- keeps only full rows
- discard rows that would have missing values

Outer joins

- produce tables with missing data
- full or asymmetric (left or right) joins

All cases

Left table

x	y
-3	1
4	2
5	NA

Missing foreign key

Right table

y	z
1	a
2	b
3	c

Unreferenced key

All cases

Left table

x	y
-3	1
4	2
5	NA

Missing foreign key

Right table

y	z
1	a
2	b
3	c

Unreferenced key

Inner join

x	y	z
-3	1	a
4	2	b

Only full rows

All cases

Left table

x	y
-3	1
4	2
5	NA

Missing foreign key

Right table

y	z
1	a
2	b
3	c

Unreferenced key

Inner join

x	y	z
-3	1	a
4	2	b

Only full rows

Full outer join

x	y	z
-3	1	a
4	2	b
5	NA	NA
NA	3	c

All combinations

All cases

Left table

x	y
-3	1
4	2
5	NA

Missing foreign key

Right table

y	z
1	a
2	b
3	c

Unreferenced key

Inner join

x	y	z
-3	1	a
4	2	b

Only full rows

Full outer join

x	y	z
-3	1	a
4	2	b
5	NA	NA
NA	3	c

All combinations

Left outer join

x	y	z
-3	1	a
4	2	b
5	NA	NA

All rows from the left table

Right outer join

x	y	z
-3	1	a
4	2	b
NA	3	c

All rows from the right table

Implementation

Python

- `merge` function
- `pd.merge(left, right)`: inner join
- parameters
  - `how`: join type among `'left'`, `'right'`, `'outer'`, `'inner'`
  - `on`: column name(s) for the join
- many others

R

- `merge` in base R
- `dplyr`:
  - several functions in with explicit names: `inner_join`, `full_join`, `left_join`, `right_join`
  - `by` parameter: column name(s) for the join

```
left %>% full_join(right)
```

Multiple joins

Loan data

- table with clients and accounts
- difficulties
  - link table
  - duplicate variable name `district_id`

Solution

- two joins
- columns renaming

Multiple joins

Loan data

- table with clients and accounts
- difficulties
  - link table
  - duplicate variable name `district_id`

Solution

- two joins
- columns renaming

Python

```
da = pd.merge(disposition, account)
da.rename(columns={
        'district_id':
            'acc_district_id'},
          inplace=True)
fulldata = pd.merge(da, client)
```

R

```
disposition %>% inner_join(account) %>%
    rename(acc_district_id=
               district_id) %>%
    inner_join(client)
```

Result

disp_id	client_id	account_id	type
1	1	1	OWNER
2	2	2	OWNER
3	3	2	DISPONENT
4	4	3	OWNER
5	5	3	DISPONENT
6	6	4	OWNER
7	7	5	OWNER
8	8	6	OWNER
9	9	7	OWNER
10	10	8	OWNER
11	11	8	DISPONENT
12	12	9	OWNER
13	13	10	OWNER
14	14	11	OWNER
15	15	12	OWNER
16	16	12	DISPONENT
17	17	13	OWNER
18	18	13	DISPONENT
19	19	14	OWNER
20	20	15	OWNER

account_id	district_id	date
1	18	1995-03-24
2	1	1993-02-26
3	5	1997-07-07
4	12	1996-02-21
5	15	1997-05-30
6	51	1994-09-27
7	60	1996-11-24
8	57	1995-09-21
9	70	1993-01-27
10	54	1996-08-28
11	76	1995-10-10
12	21	1997-04-15
13	76	1997-08-17
14	47	1996-11-27
15	70	1993-10-02
16	12	1997-09-23
17	1	1997-01-08
18	43	1993-05-26
19	21	1995-04-07
20	74	1996-08-24

Result

disp_id	client_id	account_id	type	acc_district_id	date
1	1	1	OWNER	18	1995-03-24
2	2	2	OWNER	1	1993-02-26
3	3	2	DISPONENT	1	1993-02-26
4	4	3	OWNER	5	1997-07-07
5	5	3	DISPONENT	5	1997-07-07
6	6	4	OWNER	12	1996-02-21
7	7	5	OWNER	15	1997-05-30
8	8	6	OWNER	51	1994-09-27
9	9	7	OWNER	60	1996-11-24
10	10	8	OWNER	57	1995-09-21
11	11	8	DISPONENT	57	1995-09-21
12	12	9	OWNER	70	1993-01-27
13	13	10	OWNER	54	1996-08-28
14	14	11	OWNER	76	1995-10-10
15	15	12	OWNER	21	1997-04-15
16	16	12	DISPONENT	21	1997-04-15
17	17	13	OWNER	76	1997-08-17
18	18	13	DISPONENT	76	1997-08-17
19	19	14	OWNER	47	1996-11-27
20	20	15	OWNER	70	1993-10-02

Result

client_id	type	acc_district_id	date
1	OWNER	18	1995-03-24
2	OWNER	1	1993-02-26
3	DISPONENT	1	1993-02-26
4	OWNER	5	1997-07-07
5	DISPONENT	5	1997-07-07
6	OWNER	12	1996-02-21
7	OWNER	15	1997-05-30
8	OWNER	51	1994-09-27
9	OWNER	60	1996-11-24
10	OWNER	57	1995-09-21
11	DISPONENT	57	1995-09-21
12	OWNER	70	1993-01-27
13	OWNER	54	1996-08-28
14	OWNER	76	1995-10-10
15	OWNER	21	1997-04-15
16	DISPONENT	21	1997-04-15
17	OWNER	76	1997-08-17
18	DISPONENT	76	1997-08-17
19	OWNER	47	1996-11-27
20	OWNER	70	1993-10-02

client_id	gender	birth_date	district_id
1	F	1970-12-13	18
2	M	1945-02-04	1
3	F	1940-10-09	1
4	M	1956-12-01	5
5	F	1960-07-03	5
6	M	1919-09-22	12
7	M	1929-01-25	15
8	F	1938-02-21	51
9	M	1935-10-16	60
10	M	1943-05-01	57
11	F	1950-08-22	57
12	M	1981-02-20	40
13	F	1974-05-29	54
14	F	1942-06-22	76
15	F	1918-08-28	21
16	M	1919-02-25	21
17	M	1934-10-13	76
18	F	1931-04-05	76
19	M	1942-12-28	47
20	M	1979-01-04	46

Result

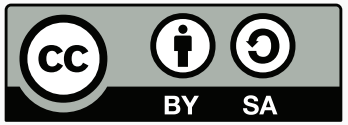
disp_id	client_id	account_id	type	acc_district_id	date	gender	birth_date	district_id
1	1	1	OWNER	18	1995-03-24	F	1970-12-13	18
2	2	2	OWNER	1	1993-02-26	M	1945-02-04	1
3	3	2	DISPONENT	1	1993-02-26	F	1940-10-09	1
4	4	3	OWNER	5	1997-07-07	M	1956-12-01	5
5	5	3	DISPONENT	5	1997-07-07	F	1960-07-03	5
6	6	4	OWNER	12	1996-02-21	M	1919-09-22	12
7	7	5	OWNER	15	1997-05-30	M	1929-01-25	15
8	8	6	OWNER	51	1994-09-27	F	1938-02-21	51
9	9	7	OWNER	60	1996-11-24	M	1935-10-16	60
10	10	8	OWNER	57	1995-09-21	M	1943-05-01	57
11	11	8	DISPONENT	57	1995-09-21	F	1950-08-22	57
12	12	9	OWNER	70	1993-01-27	M	1981-02-20	40
13	13	10	OWNER	54	1996-08-28	F	1974-05-29	54
14	14	11	OWNER	76	1995-10-10	F	1942-06-22	76
15	15	12	OWNER	21	1997-04-15	F	1918-08-28	21
16	16	12	DISPONENT	21	1997-04-15	M	1919-02-25	21
17	17	13	OWNER	76	1997-08-17	M	1934-10-13	76
18	18	13	DISPONENT	76	1997-08-17	F	1931-04-05	76
19	19	14	OWNER	47	1996-11-27	M	1942-12-28	47
20	20	15	OWNER	70	1993-10-02	M	1979-01-04	46

Advanced join topics

Support for real world issues

- key selection
- variable renaming
- filtering join (dplyr R only)
- enforcing/checking unicity of keys (python only)
- index based join (python only)
- etc.

Licence



Version

Last git commit: 2021-01-19
By: Fabrice Rossi (Fabrice.Rossi@apiacoa.org)
Git hash: a623238c82efeb5372d8b821e0e946cfd8c918cc

Changelog

- November 2019: added multiple table data
- October 2019: initial version