Journal of Statistical Software
MMMMMM YYYY, Volume VV, Issue II. http://www.jstatsoft.org/

Tidy Data

Hadley Wickham
RStudio

Hadley Wickham

Hadley Wickham in 2015

Lecture 6-1: Tidy data II

Dr. James Chen, Animal Data Scientist, School of Animal Sciences

The five most common problems with messy datasets

- Column headers are values, not variable names.
- Multiple variables are stored in one column.

Week 6 - 2

- 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.

Variables Are Stored in Both Rows and Columns

Global Historical Climatology Network (GHCN)

Site: MX000017004; **YYYY**: years; **MM**: months; **ZZZZ**: {TMIN, TMAX, PRECIPITATION}

Site+YYYYMMZZZZ

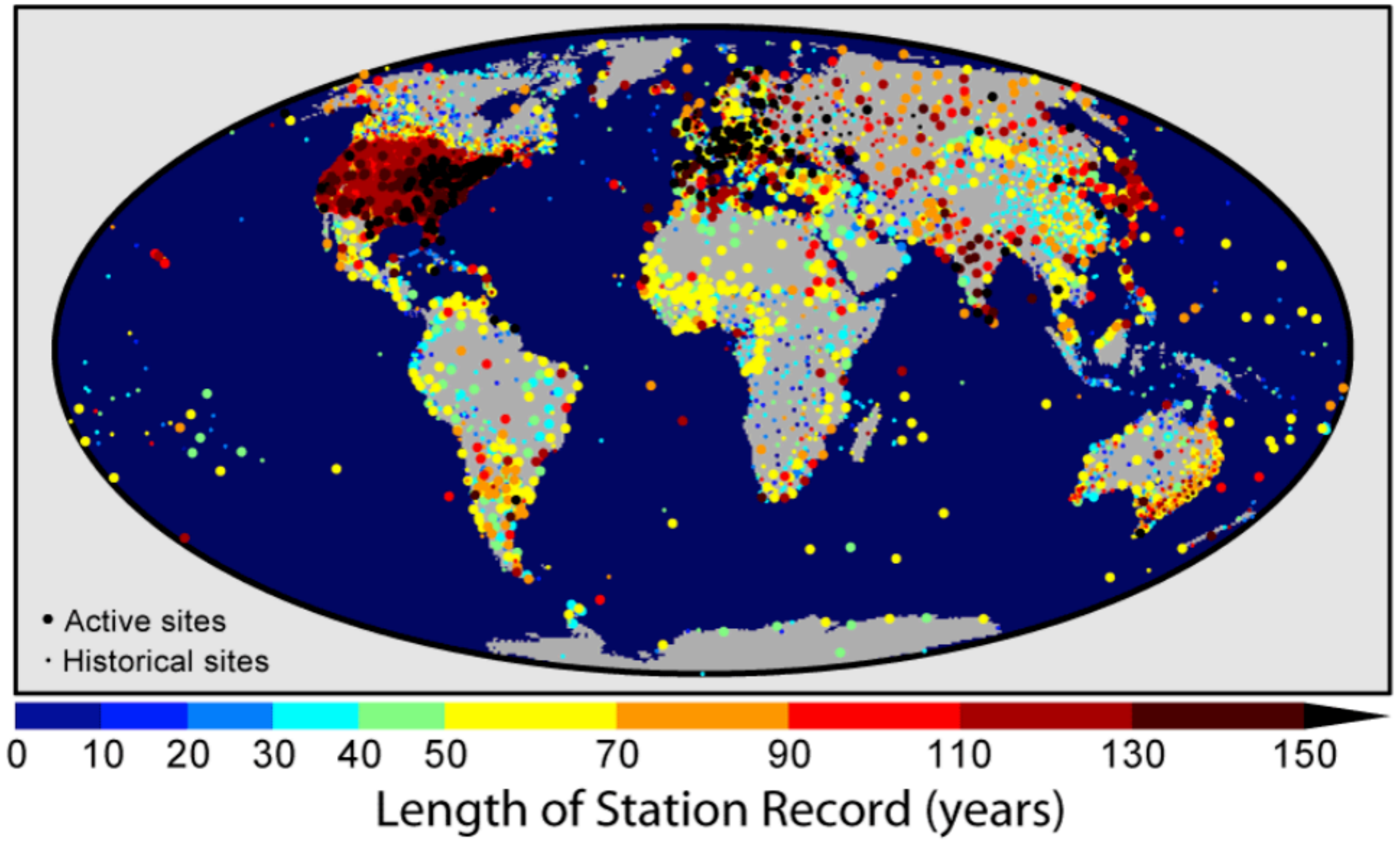
Day X (1 ~ 30 or 31)

	idx	1	2	3	4	5	6	7	8	9	...
0	MX000017004195504TMIN	150	150	160	150	160	160	160	160	160	...
1	MX000017004195504PRCP	0	0	0	0	0	0	0	0	0	...
2	MX000017004195505TMAX	310	310	310	300	300	300	310	310	310	...
3	MX000017004195505TMIN	200	160	160	150	150	150	160	160	170	...
4	MX000017004195505PRCP	0	0	0	0	0	0	0	0	0	...
..	...	...	...	...	...	...	...	...	...	...	...
994	MX000017004199004TMIN	0	147	147	150	136	0	157	0	0	...
995	MX000017004199004PRCP	0	0	0	0	23	0	0	0	0	...
996	MX000017004199005TMAX	0	350	362	348	337	0	240	0	0	...
997	MX000017004199005TMIN	168	168	167	167	170	0	132	132	0	...
998	MX000017004199005PRCP	0	0	0	0	61	0	254	20	0	...

	22	23	24	25	26	27	28	29	30	31
0	170	170	170	180	190	190	170	180	160	0
1	0	0	0	0	0	0	0	0	6	0
2	330	340	350	330	310	310	320	310	300	290
3	170	190	190	190	180	160	150	170	150	160
4	0	0	0	0	0	142	0	54	0	46
..	...	...	...	...	...	...	...	...	...	...
994	0	0	0	0	0	0	0	0	157	0
995	0	0	0	0	0	0	0	0	0	0
996	0	297	0	0	0	0	0	0	0	336
997	144	136	136	155	155	0	0	166	179	169
998	0	89	0	0	0	0	0	0	0	3

[999 rows x 32 columns]

Global Climate Network Temperature Stations



https://en.wikipedia.org/wiki/Global_Historical_Climatology_Network

Variables Are Stored in Both Rows and Columns

Extract information from a complex string

Split by positional indexing

Columns to unpivot

	idx	1	2	3	4	5	6	7	8	9	...
0	MX000017004195504TMIN	150	150	160	150	160	160	160	160	160	...
1	MX000017004195504PRCP	0	0	0	0	0	0	0	0	0	...
2	MX000017004195505TMAX	310	310	310	300	300	300	310	310	310	...
3	MX000017004195505TMIN	200	160	160	150	150	150	160	160	170	...
4	MX000017004195505PRCP	0	0	0	0	0	0	0	0	0	...
..	...	...	...	...	...	...	...	...	...	...	...
994	MX000017004199004TMIN	0	147	147	150	136	0	157	0	0	...
995	MX000017004199004PRCP	0	0	0	0	23	0	0	0	0	...
996	MX000017004199005TMAX	0	350	362	348	337	0	240	0	0	...
997	MX000017004199005TMIN	168	168	167	167	170	0	132	132	0	...
998	MX000017004199005PRCP	0	0	0	0	61	0	254	20	0	...

	22	23	24	25	26	27	28	29	30	31
0	170	170	170	180	190	190	170	180	160	0
1	0	0	0	0	0	0	0	0	6	0
2	330	340	350	330	310	310	320	310	300	290
3	170	190	190	190	180	160	150	170	150	160
4	0	0	0	0	0	142	0	54	0	46
..	...	...	...	...	...	...	...	...	...	...
994	0	0	0	0	0	0	0	0	157	0
995	0	0	0	0	0	0	0	0	0	0
996	0	297	0	0	0	0	0	0	0	336
997	144	136	136	155	155	0	0	166	179	169
998	0	89	0	0	0	0	0	0	0	3

[999 rows x 32 columns]

site	year	month	variable
MX000017004	1955	4	TMIN
MX000017004	1955	4	PRCP
MX000017004	1955	5	TMAX
MX000017004	1955	5	TMIN
MX000017004	1955	5	PRCP
...	...	...	...
MX000017004	1990	4	TMIN
MX000017004	1990	4	PRCP
MX000017004	1990	5	TMAX
MX000017004	1990	5	TMIN
MX000017004	1990	5	PRCP

Variables Are Stored in Both Rows and Columns

Use df.melt() to turn multiple columns to one variable

```
data2 = data.melt(id_vars=["site", "year", "month", "variable"],
                  var_name="day", value_name="value")
```

id_vars

Original data

value_vars

site	year	month	variable	1	2	3	4	5	6	7	8	9
MX000017004	1955	4	TMIN	150	150	160	150	160	160	160	160	160
MX000017004	1955	4	PRCP	0	0	0	0	0	0	0	0	0
MX000017004	1955	5	TMAX	310	310	310	300	300	300	310	310	310
MX000017004	1955	5	TMIN	200	160	160	150	150	150	160	160	170
MX000017004	1955	5	PRCP	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
MX000017004	1990	4	TMIN	0	147	147	150	136	0	157	0	0
MX000017004	1990	4	PRCP	0	0	0	0	23	0	0	0	0
MX000017004	1990	5	TMAX	0	350	362	348	337	0	240	0	0
MX000017004	1990	5	TMIN	168	168	167	167	170	0	132	132	0
MX000017004	1990	5	PRCP	0	0	0	0	61	0	254	20	0

22	23	24	25	26	27	28	29	30	31
170	170	170	180	190	190	170	180	160	0
0	0	0	0	0	0	0	0	6	0
330	340	350	330	310	310	320	310	300	290
170	190	190	190	180	160	150	170	150	160
0	0	0	0	0	142	0	54	0	46
...	...	...	...	...	...	...	...	...	...
0	0	0	0	0	0	0	0	157	0
0	0	0	0	0	0	0	0	0	0
0	297	0	0	0	0	0	0	0	336
144	136	136	155	155	0	0	166	179	169
0	89	0	0	0	0	0	0	0	3

Molten data

site	year	month	variable	day	value
MX000017004	1955	4	TMIN	1	150
MX000017004	1955	4	PRCP	1	0
MX000017004	1955	5	TMAX	1	310
MX000017004	1955	5	TMIN	1	200
MX000017004	1955	5	PRCP	1	0
...	...	...	...	...	...
MX000017004	1990	4	TMIN	31	0
MX000017004	1990	4	PRCP	31	0
MX000017004	1990	5	TMAX	31	336
MX000017004	1990	5	TMIN	31	169
MX000017004	1990	5	PRCP	31	3

We still need to put this variable back to columns

Variables Are Stored in Both Rows and Columns

Use pd.pivot() to put variable values back to separate columns (variables)

pandas.DataFrame.pivot

`DataFrame.pivot(*, index=None, columns=None, values=None)` [source]

Return reshaped DataFrame organized by given index / column values.

Reshape data (produce a "pivot" table) based on column values. Uses unique values from specified *index* / *columns* to form axes of the resulting DataFrame. This function does not support data aggregation, multiple values will result in a MultiIndex in the columns. See the User Guide for more on reshaping.

Parameters: **index** : *str or object or a list of str, optional*

Column to use to make new frame's index. If None, uses existing index.

> *Changed in version 1.1.0:* Also accept list of index names.

columns : *str or object or a list of str*

Column to use to make new frame's columns.

> *Changed in version 1.1.0:* Also accept list of columns names.

values : *str, object or a list of the previous, optional*

Column(s) to use for populating new frame's values. If not specified, all remaining columns will be used and the result will have hierarchically indexed columns.

index columns values

site	year	month	variable	day	value
MX000017004	1955	4	TMIN	1	150
MX000017004	1955	4	PRCP	1	0
MX000017004	1955	5	TMAX	1	310
MX000017004	1955	5	TMIN	1	200
MX000017004	1955	5	PRCP	1	0
...	...	...	...	...	...
MX000017004	1990	4	TMIN	31	0
MX000017004	1990	4	PRCP	31	0
MX000017004	1990	5	TMAX	31	336
MX000017004	1990	5	TMIN	31	169
MX000017004	1990	5	PRCP	31	3

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html#pandas.DataFrame.pivot

Variables Are Stored in Both Rows and Columns

Use pd.pivot() to put variable values back to separate columns (variables)

```
data3 = data2.pivot(index = ["site", "year", "month", "day"],
                    columns = "variable", values="value")
```

Tidied data

index

site	year	month	day	PRCP	TMAX	TMIN
MX000017004	1955	5	1	0.0	310.0	200.0
MX000017004	1955	5	2	0.0	310.0	160.0
MX000017004	1955	5	3	0.0	310.0	160.0
MX000017004	1955	5	4	0.0	300.0	150.0
MX000017004	1955	5	5	0.0	300.0	150.0
...	...	...	...	...	...	...
MX000017004	1990	5	27	0.0	0.0	0.0
MX000017004	1990	5	28	0.0	0.0	0.0
MX000017004	1990	5	29	0.0	0.0	166.0
MX000017004	1990	5	30	0.0	0.0	179.0
MX000017004	1990	5	31	3.0	336.0	169.0

index columns values

site	year	month	variable	day	value
MX000017004	1955	4	TMIN	1	150
MX000017004	1955	4	PRCP	1	0
MX000017004	1955	5	TMAX	1	310
MX000017004	1955	5	TMIN	1	200
MX000017004	1955	5	PRCP	1	0
...	...	...	...	...	...
MX000017004	1990	4	TMIN	31	0
MX000017004	1990	4	PRCP	31	0
MX000017004	1990	5	TMAX	31	336
MX000017004	1990	5	TMIN	31	169
MX000017004	1990	5	PRCP	31	3

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html#pandas.DataFrame.pivot

Multiple Types in One Table

Billboard dataset revisit

Let's revisit the billboard dataset that we worked in the second type of messy data. Although we have tidied the dataset, we can still see many repeated values in the data. For example, if we only check records of the song "Maria, Maria", we can see that multiple columns, such as `artist.inverted` and `track`, were repeated for each week.

	year	artist.inverted	track
0	2000	Destiny's Child	Independent Women Part I
1	2000	Santana	Maria, Maria
2	2000	Savage Garden	I Knew I Loved You
3	2000	Madonna	Music
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)
...	...	...	...
24087	2000	Ghostface Killah	Cherchez LaGhost
24088	2000	Smith, Will	Freakin' It
24089	2000	Zombie Nation	Kernkraft 400
24090	2000	Eastsidaz, The	Got Beef
24091	2000	Fragma	Toca's Miracle

	time	genre	date.entered	date.peaked	week	rank
0	03:38	Rock	9/23/00	11/18/00	1	78.0
1	04:18	Rock	2/12/00	4/8/00	1	15.0
2	04:07	Rock	10/23/99	1/29/00	1	71.0
3	03:45	Rock	8/12/00	9/16/00	1	41.0
4	03:38	Rock	8/5/00	10/14/00	1	57.0
...	...	...	...	...	...	...
24087	03:04	R&B	8/5/00	8/5/00	76	NaN
24088	03:58	Rap	2/12/00	2/12/00	76	NaN
24089	03:30	Rock	9/2/00	9/2/00	76	NaN
24090	03:58	Rap	7/1/00	7/1/00	76	NaN
24091	03:22	R&B	10/28/00	10/28/00	76	NaN

Multiple Types in One Table

Repeated information

Original data

	year	artist.inverted	track
0	2000	Destiny's Child	Independent Women Part I
1	2000	Santana	Maria, Maria
2	2000	Savage Garden	I Knew I Loved You
3	2000	Madonna	Music
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)
...	...	...	...
24087	2000	Ghostface Killah	Cherchez LaGhost
24088	2000	Smith, Will	Freakin' It
24089	2000	Zombie Nation	Kernkraft 400
24090	2000	Eastsidaz, The	Got Beef
24091	2000	Fragma	Toca's Miracle

Song Data

	time	genre	date.entered	date.peaked	week	rank
0	03:38	Rock	9/23/00	11/18/00	1	78.0
1	04:18	Rock	2/12/00	4/8/00	1	15.0
2	04:07	Rock	10/23/99	1/29/00	1	71.0
3	03:45	Rock	8/12/00	9/16/00	1	41.0
4	03:38	Rock	8/5/00	10/14/00	[illegible]	[illegible]
...	...	...	...	...	...	...
24087	03:04	R&B	8/5/00	8/5/00	76	NaN
24088	03:58	Rap	2/12/00	2/12/00	76	NaN
24089	03:30	Rock	9/2/00	9/2/00	76	NaN
24090	03:58	Rap	7/1/00	7/1/00	76	NaN
24091	03:22	R&B	10/28/00	10/28/00	76	NaN

Rank Data

We need two separate datasets

The query for one specific song

```
data.query("track == 'Maria, Maria'")
```

	year	artist.inverted	track	time	genre	date.entered
1	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
318	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
635	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
952	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
1269	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
...	...	...	...	...	...	...
22508	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
22825	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
23142	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
23459	2000	Santana	Maria, Maria	04:18	Rock	2/12/00
23776	2000	Santana	Maria, Maria	04:18	Rock	2/12/00

	date.peaked	week	rank
1	4/8/00	1	15.0
318	4/8/00	2	8.0
635	4/8/00	3	6.0
952	4/8/00	4	5.0
1269	4/8/00	5	2.0
...	...	...	...
22508	4/8/00	72	NaN
22825	4/8/00	73	NaN
23142	4/8/00	74	NaN
23459	4/8/00	75	NaN
23776	4/8/00	76	NaN

Multiple Types in One Table

Obtain an unique ID for each song

index	year	artist.inverted	track	time	genre	date.entered	date.peaked
0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00
1	2000	Santana	Maria, Maria	04:18	Rock	2/12/00	4/8/00
2	2000	Savage Garden	I Knew I Loved You	04:07	Rock	10/23/99	1/29/00
3	2000	Madonna	Music	03:45	Rock	8/12/00	9/16/00
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	03:38	Rock	8/5/00	10/14/00
...	...	...	...	...	...	...	...
312	2000	Ghostface Killah	Cherchez LaGhost	03:04	R&B	8/5/00	8/5/00
313	2000	Smith, Will	Freakin' It	03:58	Rap	2/12/00	2/12/00
314	2000	Zombie Nation	Kernkraft 400	03:30	Rock	9/2/00	9/2/00
315	2000	Eastsidaz, The	Got Beef	03:58	Rap	7/1/00	7/1/00
316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00

Subset only the columns that contain song information

Drop duplicated rows (like we saw earlier from the song “Maria, Maria”

Use df.reset_index() to obtain unique IDs

Multiple Types in One Table

pd.merge() to add "index" to the original data frame by the key columns

```
pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'),
copy=True, indicator=False, validate=None)    [source]
```

Parameters: **left** : *DataFrame*

right : *DataFrame or named Series*

Object to merge with.

on : *label or list*

Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

Left (original table)

year	artist.inverted	track
2000	Destiny's Child	Independent Women Part I
2000	Santana	Maria, Maria
2000	Savage Garden	I Knew I Loved You
2000	Madonna	Music
2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)
...	...	...
2000	Ghostface Killah	Cherchez LaGhost
2000	Smith, Will	Freakin' It
2000	Zombie Nation	Kernkraft 400
2000	Eastsidaz, The	Got Beef
2000	Fragma	Toca's Miracle

time	genre	date.entered	date.peaked	week	rank
03:38	Rock	9/23/00	11/18/00	1	78.0
04:18	Rock	2/12/00	4/8/00	1	15.0
04:07	Rock	10/23/99	1/29/00	1	71.0
03:45	Rock	8/12/00	9/16/00	1	41.0
03:38	Rock	8/5/00	10/14/00	1	57.0
...	...	...	...	...	...
03:04	R&B	8/5/00	8/5/00	76	NaN
03:58	Rap	2/12/00	2/12/00	76	NaN
03:30	Rock	9/2/00	9/2/00	76	NaN
03:58	Rap	7/1/00	7/1/00	76	NaN
03:22	R&B	10/28/00	10/28/00	76	NaN

Right (song table with index)

index	year	artist.inverted	track
0	2000	Destiny's Child	Independent Women Part I
1	2000	Santana	Maria, Maria
2	2000	Savage Garden	I Knew I Loved You
3	2000	Madonna	Music
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)
...	...	...	...
312	2000	Ghostface Killah	Cherchez LaGhost
313	2000	Smith, Will	Freakin' It
314	2000	Zombie Nation	Kernkraft 400
315	2000	Eastsidaz, The	Got Beef
316	2000	Fragma	Toca's Miracle

time	genre	date.entered	date.peaked
03:38	Rock	9/23/00	11/18/00
04:18	Rock	2/12/00	4/8/00
04:07	Rock	10/23/99	1/29/00
03:45	Rock	8/12/00	9/16/00
03:38	Rock	8/5/00	10/14/00
...	...	...	...
03:04	R&B	8/5/00	8/5/00
03:58	Rap	2/12/00	2/12/00
03:30	Rock	9/2/00	9/2/00
03:58	Rap	7/1/00	7/1/00
03:22	R&B	10/28/00	10/28/00

Key columns
(The 'on' argument)

Added information
(Index)

Multiple Types in One Table

pd.merge() to add “index” to the original data frame by the key columns

```
pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None,
left_index=False, right_index=False, sort=False, suffixes=('_x', '_y'),
copy=True, indicator=False, validate=None)                          [source]
```

Parameters: **left** : *DataFrame*

right : *DataFrame or named Series*

Object to merge with.

on : *label or list*

Column or index level names to join on. These must be found in both DataFrames. If *on* is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames.

Merged table (song)

	index	year	artist.inverted	track	time	genre	date.entered	date.peaked	week	rank
0	0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00	1	78.0
1	0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00	2	63.0
2	0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00	3	49.0
3	0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00	4	33.0
4	0	2000	Destiny's Child	Independent Women Part I	03:38	Rock	9/23/00	11/18/00		
...	...	...	...	...	...	...	...	...		
24087	316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00	72	NaN
24088	316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00	73	NaN
24089	316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00	74	NaN
24090	316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00	75	NaN
24091	316	2000	Fragma	Toca's Miracle	03:22	R&B	10/28/00	10/28/00	76	NaN

Dropped Columns

Subset table (rank)

	index	week	rank
0	0	1	78.0
1	0	2	63.0
2	0	3	49.0
3	0	4	33.0
4	0	5	23.0
...	...	...	...
24087	316	72	NaN
24088	316	73	NaN
24089	316	74	NaN
24090	316	75	NaN
24091	316	76	NaN

[24092 rows x 3 columns]

Key columns (The ‘on’ argument)

Added information (Index)

Multiple Types in One Table

We can compare the number of values we need before and after the transformation.

```
size_original = data.size
size_new = data_rank.size + data_song.size

print("The number of elements (number of rows times number of columns) in data is ", size_original)
print("New data has ", size_new, "elements")
print("Compression ratio is ",   size_new / size_original)
```

```
The number of elements (number of rows times number of columns) in data is  216828
New data has  74812 elements
Compression ratio is  0.34502923976608185
```

Tidying the table can save data storage by > 65%

A Single Observational Unit Is Stored in Multiple Tables

This is a case where a single observational unit is stored in multiple tables organized by different variables. In this example, we will work on the `babynames` dataset. This dataset consists of multiple files, each of which contains the baby names and their proportions. The files are organized by year and gender in a format of `babynames_yyyy_xxx.csv`, where `yyyy` is the year and `xxx` is the gender (i.e., boy or girl).

Let's bbserve the file naming pattern first

```
os.listdir("tidy_6_babynames")
```

```
['babynames_1887_boy.csv',
 'babynames_1897_boy.csv',
 'babynames_1959_girl.csv',
 'babynames_1958_girl.csv',
 'babynames_1946_boy.csv',
 'babynames_1956_boy.csv',
 'babynames_1982_girl.csv',
 ...
 'babynames_1931_boy.csv',
 'babynames_1921_boy.csv',
 'babynames_1974_girl.csv',
 'babynames_1975_girl.csv',
 'babynames_1988_boy.csv',
 'babynames_1998_boy.csv']
```

We can try to open one of them to inspect the data.

```
pd.read_csv(os.path.join("tidy_6_babynames", "babynames_1887_boy.csv"))
```

```
         name   percent
0        John  0.074181
1     William  0.068344
2       James  0.043617
3      George  0.039190
4     Charles  0.036875
..        ...       ...
995    Jessee  0.000046
996     Jewel  0.000046
997     Jodie  0.000046
998      Lars  0.000046
999    Laurel  0.000046

[1000 rows x 2 columns]
```

No **gender** or **year** info in the table itself

Need to extract the info from the **filename**

A Single Observational Unit Is Stored in Multiple Tables

You will find that there is no year or gender information in the file, we need to extract them from the file name. Let's start with a single file to experiment the extraction process.

```
filename = "babynames_1887_boy.csv"
year = re.findall(r"\d+", filename)[0]
gender = re.findall(r"[a-z]+\.", filename)[0].replace(".", "")
data = pd.read_csv(os.path.join("tidy_6_babynames", filename))
data["year"] = year
data["gender"] = gender
data
```

Try to understand the code

```
        name   percent  year gender
0       John  0.074181  1887    boy
1    William  0.068344  1887    boy
2      James  0.043617  1887    boy
3     George  0.039190  1887    boy
4    Charles  0.036875  1887    boy
..       ...       ...   ...    ...
995   Jessee  0.000046  1887    boy
996    Jewel  0.000046  1887    boy
997    Jodie  0.000046  1887    boy
998     Lars  0.000046  1887    boy
999   Laurel  0.000046  1887    boy

[1000 rows x 4 columns]
```

Obtained from the **filename**

A Single Observational Unit Is Stored in Multiple Tables

Now, we can iteratively extract needed information from each file and store them into one csv file. We can create an empty file with only the header defined.

```
FILE_OUT = "tidy_6_babynames.csv"
with open(FILE_OUT, "w") as f:
    f.write("name,percent,year,gender\n")
```

Create a new file with the header written

Next, let's test the extraction process on the first five files. Try to define constant variables instead of hard-coding the values every time.

```
DIR_DATA = "tidy_6_babynames"
ls_files = os.listdir(DIR_DATA)
for filename in ls_files:
    year = re.findall(r"\d+", filename)[0]
    gender = re.findall(r"[a-z]+\.", filename)[0].replace(".", "")
    data = pd.read_csv(os.path.join(DIR_DATA, filename))
    data["year"] = year
    data["gender"] = gender
    data.to_csv(FILE_OUT, mode="a", header=False, index=False)
```

Case Study - How To Use a Tidy Dataset?

We will use a case study to illustrate the advantages of tidying data. The dataset `tidy_X.csv` contains the individual-level mortality from Mexico.

The columns include the following:

- `sex` : the gender of the deceased
- `age` : the age of the deceased
- `yod` : the year of death
- `mod` : the month of death
- `dod` : the day of death
- `hod` : the hour of death
- `cod` : the cause of death

```
data = pd.read_csv("tidy_X.csv")
data
```

```
        sex  age   yod  mod  dod  hod  cod
0         1   90  2008    1    7   20  F17
1         1   72  2008    1   13   14  I05
2         1   49  2008    1   12   20  K65
3         2   79  2008    1   20   10  I38
4         1   15  2008    1    1   15  N18
...     ...  ...   ...  ...  ...  ...  ...
528323    1    1  2008   10    6   12  P22
528324    2   20  2008   10   18   20  Q24
528325    2    3  2008   11   11   19  P22
528326    1   24  2008    9   25   12  P22
528327    1    2  2008    9   22   16  P26

[528328 rows x 7 columns]
```

The goal is to find causes of death with unusual temporal patterns within a day.

Case Study - Inspect the Data

Data

```
data = pd.read_csv("tidy_X.csv")
data
```

```
        sex  age   yod  mod  dod  hod  cod
0         1   90  2008    1    7   20  F17
1         1   72  2008    1   13   14  I05
2         1   49  2008    1   12   20  K65
3         2   79  2008    1   20   10  I38
4         1   15  2008    1    1   15  N18
...     ...  ...   ...  ...  ...  ...  ...
528323    1    1  2008   10    6   12  P22
528324    2   20  2008   10   18   20  Q24
528325    2    3  2008   11   11   19  P22
528326    1   24  2008    9   25   12  P22
528327    1    2  2008    9   22   16  P26

[528328 rows x 7 columns]
```

Data types

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 528328 entries, 0 to 528327
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   sex     528328 non-null  int64
 1   age     528328 non-null  int64
 2   yod     528328 non-null  int64
 3   mod     528328 non-null  int64
 4   dod     528328 non-null  int64
 5   hod     528328 non-null  int64
 6   cod     528328 non-null  object
dtypes: int64(6), object(1)
memory usage: 28.2+ MB
```

Distributions

```
data.describe()
```

```
                 sex            age            yod            mod
count  528328.000000  528328.000000  528328.000000  528328.000000
mean        1.443461      61.246593    2007.950735       6.490457
std         0.496794      24.611694       2.881578       3.554002
min         1.000000       1.000000       0.000000       0.000000
25%         1.000000      48.000000    2008.000000       3.000000
50%         1.000000      67.000000    2008.000000       6.000000
75%         2.000000      80.000000    2008.000000      10.000000
max         2.000000      99.000000    2008.000000      12.000000

                 dod            hod
count  528328.000000  528328.000000
mean       15.738475      11.701500
std         8.826922       6.763691
min         0.000000       0.000000
25%         8.000000       6.000000
50%        16.000000      12.000000
75%        23.000000      17.000000
max        31.000000      23.000000
```

Case Study - Summarize the Data

Data

```
data = pd.read_csv("tidy_X.csv")
data
```

```
        sex  age   yod  mod  dod  hod  cod
0         1   90  2008    1    7   20  F17
1         1   72  2008    1   13   14  I05
2         1   49  2008    1   12   20  K65
3         2   79  2008    1   20   10  I38
4         1   15  2008    1    1   15  N18
...     ...  ...   ...  ...  ...  ...  ...
528323    1    1  2008   10    6   12  P22
528324    2   20  2008   10   18   20  Q24
528325    2    3  2008   11   11   19  P22
528326    1   24  2008    9   25   12  P22
528327    1    2  2008    9   22   16  P26

[528328 rows x 7 columns]
```

6.2 Count the number of deaths in each hour

We need to tidy the data by `hod` and `cod` to get the number of deaths in each hour. Before we do that, we can use `df.query()` to check what result we will get. For example, there should be five records of hod=0, and cod=A06.

```
data.query("hod == 0 and cod == 'A06'")
```

```
        sex  age   yod  mod  dod  hod  cod
19640     2   88  2008    1   10    0  A06
298031    1   56  2008   12   28    0  A06
395562    2   22  2008   12   20    0  A06
412156    1   83  2008    1    5    0  A06
502497    1    2  2008    7   19    0  A06
```

Then use `df.groupby()` to group the data by `hod` and `cod` and aggregate the data by `size` to count the number of deaths. We should see "5" in the `freq_by_hodcod` column as we queried above.

```
data_grp = data.groupby(["hod", "cod"]).agg(freq_by_hodcod=("hod", "size")).reset_index()
data_grp
```

```
       hod  cod  freq_by_hodcod
0        0  A02               1
1        0  A04               6
2        0  A06               5
3        0  A09              87
4        0  A15               7
...    ...  ...             ...
16171   23  Y34              28
16172   23  Y57               3
16173   23  Y83               7
16174   23  Y86              16
16175   23  Y89               5

[16176 rows x 3 columns]
```

Case Study - Summarize the Data

data_grp

```
       hod  cod  freq_by_hodcod
0        0  A02               1
1        0  A04               6
2        0  A06               5
3        0  A09              87
4        0  A15               7
...    ...  ...             ...
16171   23  Y34              28
16172   23  Y57               3
16173   23  Y83               7
16174   23  Y86              16
16175   23  Y89               5

[16176 rows x 3 columns]
```

merge

6.3 The proportion of deaths in each cause by hour

Now, let's continue with the `data_grp` to compute proportion of deaths of each `cod` given `hod`. We need to determine the denominator of the proportion first. The denominator is the total number of deaths in each `cod` across all `hod`.

Again, we can use `df.query()` to preview what we will get.

```
data_grp.query("cod == 'A03'")
```

```
       hod  cod  freq_by_hodcod
5257     8  A03               1
5922     9  A03               1
6611    10  A03               1
8014    12  A03               1
11503   17  A03               1
12204   18  A03               1
14210   21  A03               1
```

Then use `df.groupby()` to group the data by `cod` and aggregate the data by `sum` to get the total number of deaths in each `cod`.

```
sum_cod = data_grp.groupby(["cod"]).agg(sum_by_cod=("freq_by_hodcod", "sum")).reset_index()
sum_cod
```

```
      cod  sum_by_cod
0     A01          51
1     A02          62
2     A03           7
3     A04         144
4     A05          20
...   ...         ...
1192  Y85           4
1193  Y86         363
1194  Y87           2
1195  Y88           5
1196  Y89          39

[1197 rows x 2 columns]
```

Case Study - Summarize the Data

data_grp2

```
data_grp2 = pd.merge(data_grp, sum_cod)
data_grp2
```

```
       hod  cod  freq_by_hodcod  sum_by_cod
0        0  A02               1          62
1        1  A02               3          62
2        2  A02               9          62
3        3  A02               1          62
4        4  A02               3          62
...    ...  ...             ...         ...
16171   22  Y52               2           2
16172   23  D24               1           1
16173   23  N88               1           1
16174   23  O67               1           1
16175   23  V33               1           1

[16176 rows x 4 columns]
```

Then, with the columns `freq_by_hodcod` as the numerator and `sum_cod` as the denominator, we can compute the proportion of deaths in each `hod` given `cod`.

```
data_grp2["prop_by_hodcod"] = data_grp2["freq_by_hodcod"] / data_grp2["sum_by_cod"]
data_grp2
```

```
       hod  cod  freq_by_hodcod  sum_by_cod  prop_by_hodcod
0        0  A02               1          62        0.016129
1        1  A02               3          62        0.048387
2        2  A02               9          62        0.145161
3        3  A02               1          62        0.016129
4        4  A02               3          62        0.048387
...    ...  ...             ...         ...             ...
16171   22  Y52               2           2        1.000000
16172   23  D24               1           1        1.000000
16173   23  N88               1           1        1.000000
16174   23  O67               1           1        1.000000
16175   23  V33               1           1        1.000000

[16176 rows x 5 columns]
```

Case Study - Summarize the Data

data_grp2

	hod	cod	freq_by_hodcod	sum_by_cod	prop_by_hodcod
0	0	A02	1	62	0.016129
1	1	A02	3	62	0.048387
2	2	A02	9	62	0.145161
3	3	A02	1	62	0.016129
4	4	A02	3	62	0.048387
...	...	...	...	...	...
16171	22	Y52	2	2	1.000000
16172	23	D24	1	1	1.000000
16173	23	N88	1	1	1.000000
16174	23	O67	1	1	1.000000
16175	23	V33	1	1	1.000000

[16176 rows x 5 columns]

Merge

Add ‘prop_by_hod’ based on ‘hod’

6.4 The proportion of deaths in each hour

Next, to know if a cause of death has unusual temporal patterns, we need to compare the proportion of deaths in each hour with the proportion of deaths in each hour across all causes of death. We can use the `data_grp` to compute the sum of deaths in each hour first.

```
sum_hod = data_grp2.groupby(["hod"]).agg(sum_by_hod=("freq_by_hodcod", "sum")).reset_index()
sum_hod
```

	hod	sum_by_hod
0	0	20072
1	1	20248
2	2	18806
3	3	19532
4	4	20069
5	5	21883
6	6	23536
7	7	21619
8	8	21713
9	9	22223
10	10	24093
11	11	23627
12	12	23172
13	13	23058
14	14	22786
15	15	23047
16	16	23622
17	17	23395
18	18	24093
19	19	22681
20	20	22702
21	21	20813
22	22	20298
23	23	21240

```
sum_hod["sum"] = sum_hod["sum_by_hod"].sum()
sum_hod
```

	hod	sum_by_hod	sum
0	0	20072	528328
1	1	20248	528328
2	2	18806	528328
3	3	19532	528328
4	4	20069	528328
5	5	21883	528328
6	6	23536	528328
7	7	21619	528328
8	8	21713	528328
9	9	22223	528328
10	10	24093	528328
11	11	23627	528328
12	12	23172	528328
13	13	23058	528328
14	14	22786	528328
15	15	23047	528328
16	16	23622	528328
17	17	23395	528328
18	18	24093	528328
19	19	22681	528328
20	20	22702	528328
21	21	20813	528328
22	22	20298	528328
23	23	21240	528328

```
sum_hod["prop_by_hod"] = sum_hod["sum_by_hod"] / sum_hod["sum"]
sum_hod = sum_hod.loc[:, ["hod", "sum_by_hod", "prop_by_hod"]]
sum_hod
```

	hod	sum_by_hod	prop_by_hod
0	0	20072	0.037992
1	1	20248	0.038325
2	2	18806	0.035595
3	3	19532	0.036969
4	4	20069	0.037986
5	5	21883	0.041419
6	6	23536	0.044548
7	7	21619	0.040920
8	8	21713	0.041098
9	9	22223	0.042063
10	10	24093	0.045602
11	11	23627	0.044720
12	12	23172	0.043859
13	13	23058	0.043643
14	14	22786	0.043129
15	15	23047	0.043623
16	16	23622	0.044711
17	17	23395	0.044281
18	18	24093	0.045602
19	19	22681	0.042930
20	20	22702	0.042970
21	21	20813	0.039394
22	22	20298	0.038419
23	23	21240	0.040202

we can then compute the proportion by hod

Case Study - Calculate the Deviation Between Exp. And Obs.

data_grp3

	hod	cod	freq_by_hodcod	sum_by_cod	prop_by_hodcod	sum_by_hod
0	0	A02	1	62	0.016129	20072
1	0	A04	6	144	0.041667	20072
2	0	A06	5	88	0.056818	20072
3	0	A09	87	3111	0.027965	20072
4	0	A15	7	209	0.033493	20072
...	...	...	...	...	...	...
16171	23	N95	1	2	0.500000	21240
16172	23	D24	1	1	1.000000	21240
16173	23	N88	1	1	1.000000	21240
16174	23	O67	1	1	1.000000	21240
16175	23	V33	1	1	1.000000	21240

Observed

	prop_by_hod
0	0.037992
1	0.037992
2	0.037992
3	0.037992
4	0.037992
...	...
16171	0.040202
16172	0.040202
16173	0.040202
16174	0.040202
16175	0.040202

Expected
(Average)

6.5 Deviation from the expected proportion

Finally, we can compute the deviation from the expected proportion (`prop_by_hod`)

```
data_grp3["diff_prop"] = (data_grp3["prop_by_hodcod"] - data_grp3["prop_by_hod"])**2
data_grp3.head()
```

	hod	cod	freq_by_hodcod	sum_by_cod	prop_by_hodcod	sum_by_hod	prop_by_hod \
0	0	A02	1	62	0.016129	20072	0.037992
1	0	A04	6	144	0.041667	20072	0.037992
2	0	A06	5	88	0.056818	20072	0.037992
3	0	A09	87	3111	0.027965	20072	0.037992
4	0	A15	7	209	0.033493	20072	0.037992

	diff_prop
0	0.000478
1	0.000014
2	0.000354
3	0.000101
4	0.000020

Case Study - Calculate the Deviation Between Exp. And Obs.

data_grp3

hod	cod	freq_by_hodcod	sum_by_cod	prop_by_hodcod	sum_by_hod	prop_by_hod
0	A02	1	62	0.016129	20072	0.037992
0	A04	6	144	0.041667	20072	0.037992
0	A06	5	88	0.056818	20072	0.037992
0	A09	87	3111	0.027965	20072	0.037992
0	A15	7	209	0.033493	20072	0.037992

diff_prop
0.000478
0.000014
0.000354
0.000101
0.000020

Observed

Expected (Average)

Deviation

And we can follow the same process as the tidy paper to get the distance from the expected proportion.

```
devi = data_grp3.groupby(["cod"]).agg(n=("freq_by_hodcod", lambda x: x.sum()),
                                       dist=("diff_prop", lambda x: x.mean())).reset_index()
devi = devi.query("n > 50")
devi
```

```
      cod     n      dist
0     A01    51  0.000958
1     A02    62  0.000733
3     A04   144  0.000185
5     A06    88  0.000360
8     A09  3111  0.000030
...   ...   ...       ...
1172  Y33    60  0.000627
1173  Y34   780  0.000068
1183  Y57   111  0.000284
1190  Y83   174  0.000203
1193  Y86   363  0.000094

[450 rows x 3 columns]
```

Case Study - Visualization Between Avg. Deviation and Case Count

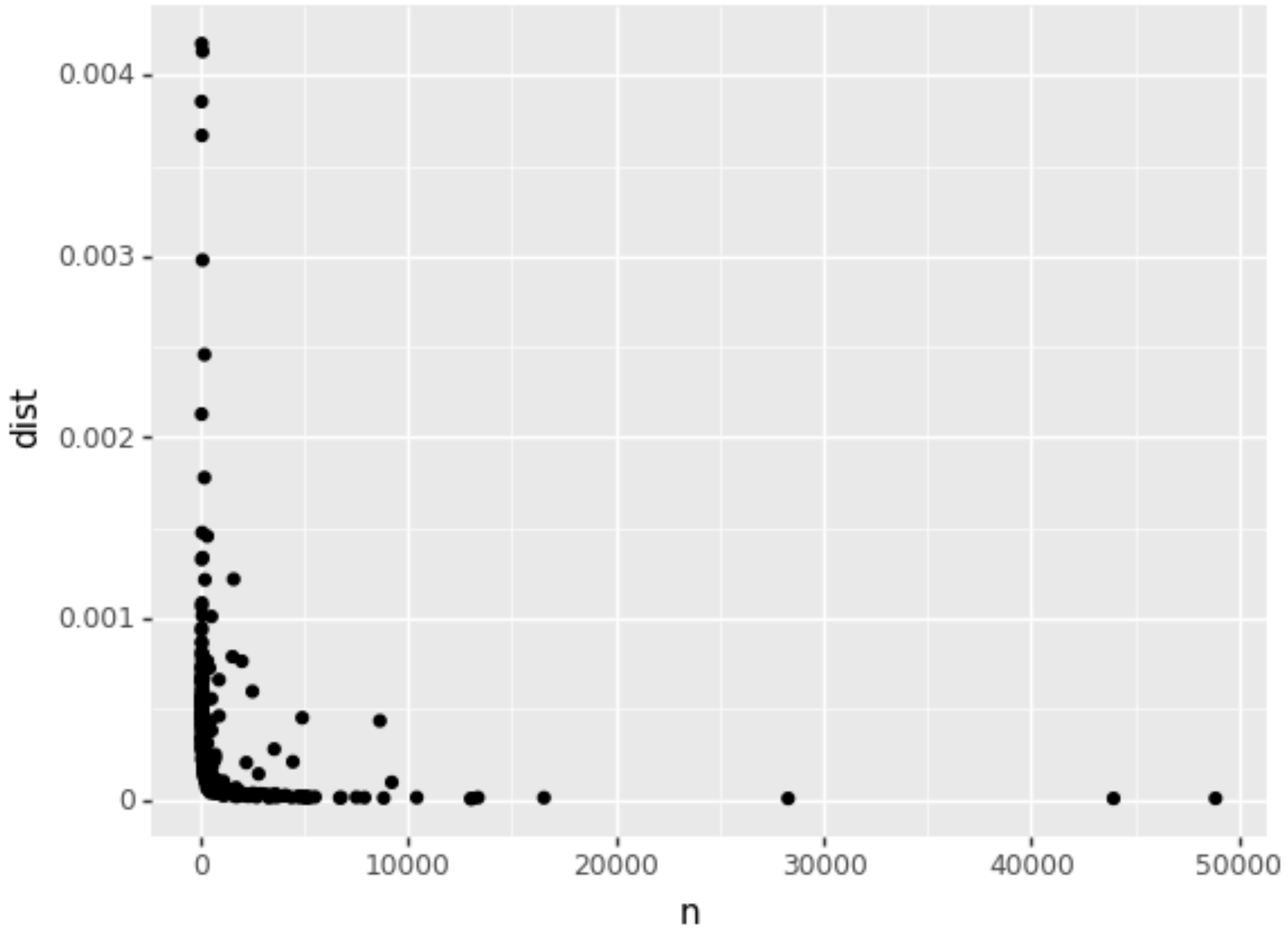
devi

```
      cod     n      dist
0     A01    51  0.000958
1     A02    62  0.000733
3     A04   144  0.000185
5     A06    88  0.000360
8     A09  3111  0.000030
...   ...   ...       ...
1172  Y33    60  0.000627
1173  Y34   780  0.000068
1183  Y57   111  0.000284
1190  Y83   174  0.000203
1193  Y86   363  0.000094

[450 rows x 3 columns]
```

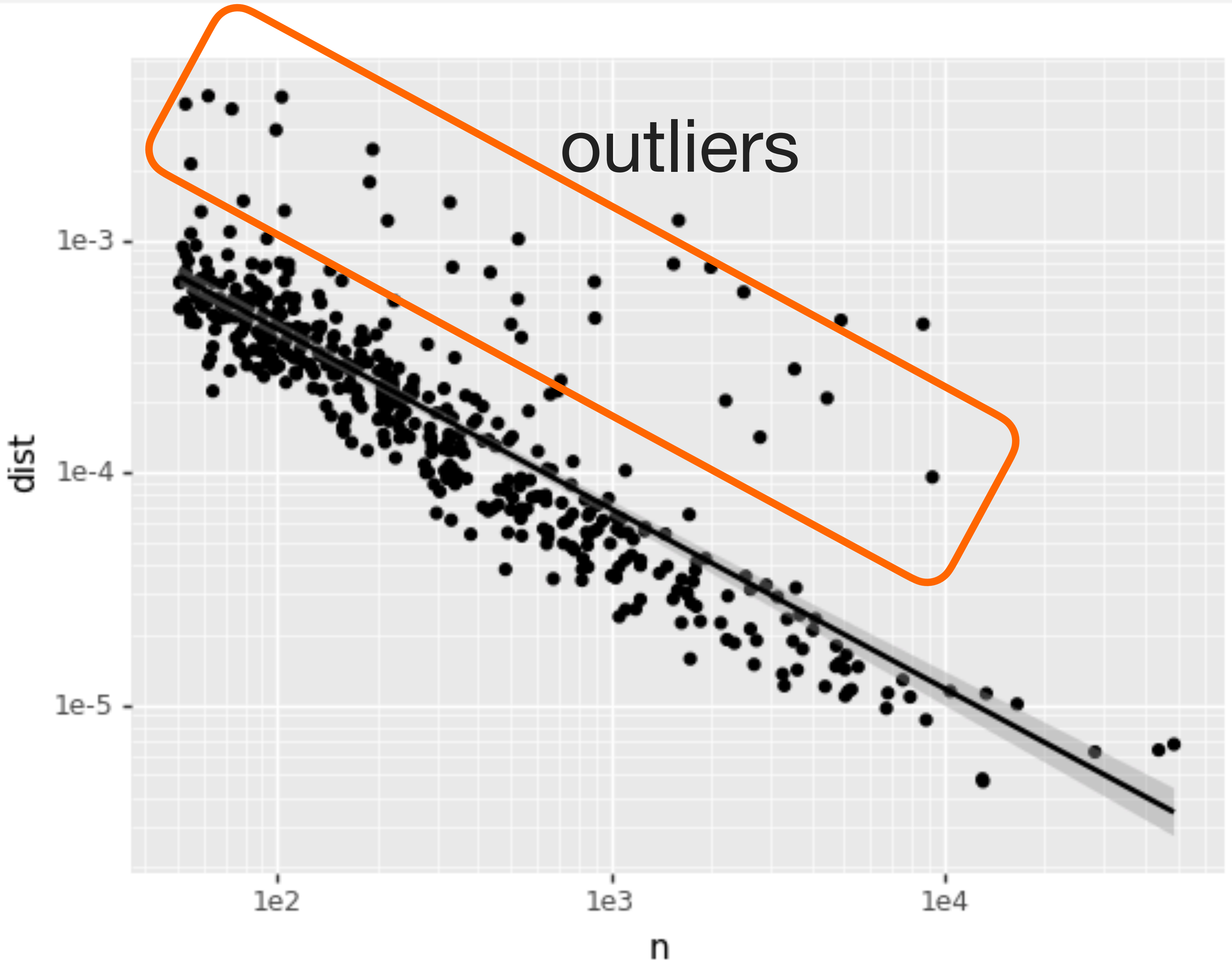
Linear

```
ggplot(devi, aes(x = 'n', y = 'dist')) + geom_point()
```



Log-transformed

```
ggplot(devi, aes(x = 'n', y = 'dist')) + geom_point()+\
scale_x_log10() +\
scale_y_log10() +\
geom_smooth(method = "lm")
```



Each dot represents a COD

Case Study - Find the Residuals

6.6 Use a residual plot to find unusual temporal patterns

We can use how well the data fit a linear model to find unusual temporal patterns. We need `statsmodels` to fit a linear model. As we learned from the previous visualization, the data are linearly related when the variables are log-transformed. Hence, we can directly fit a linear model on the log-transformed data.

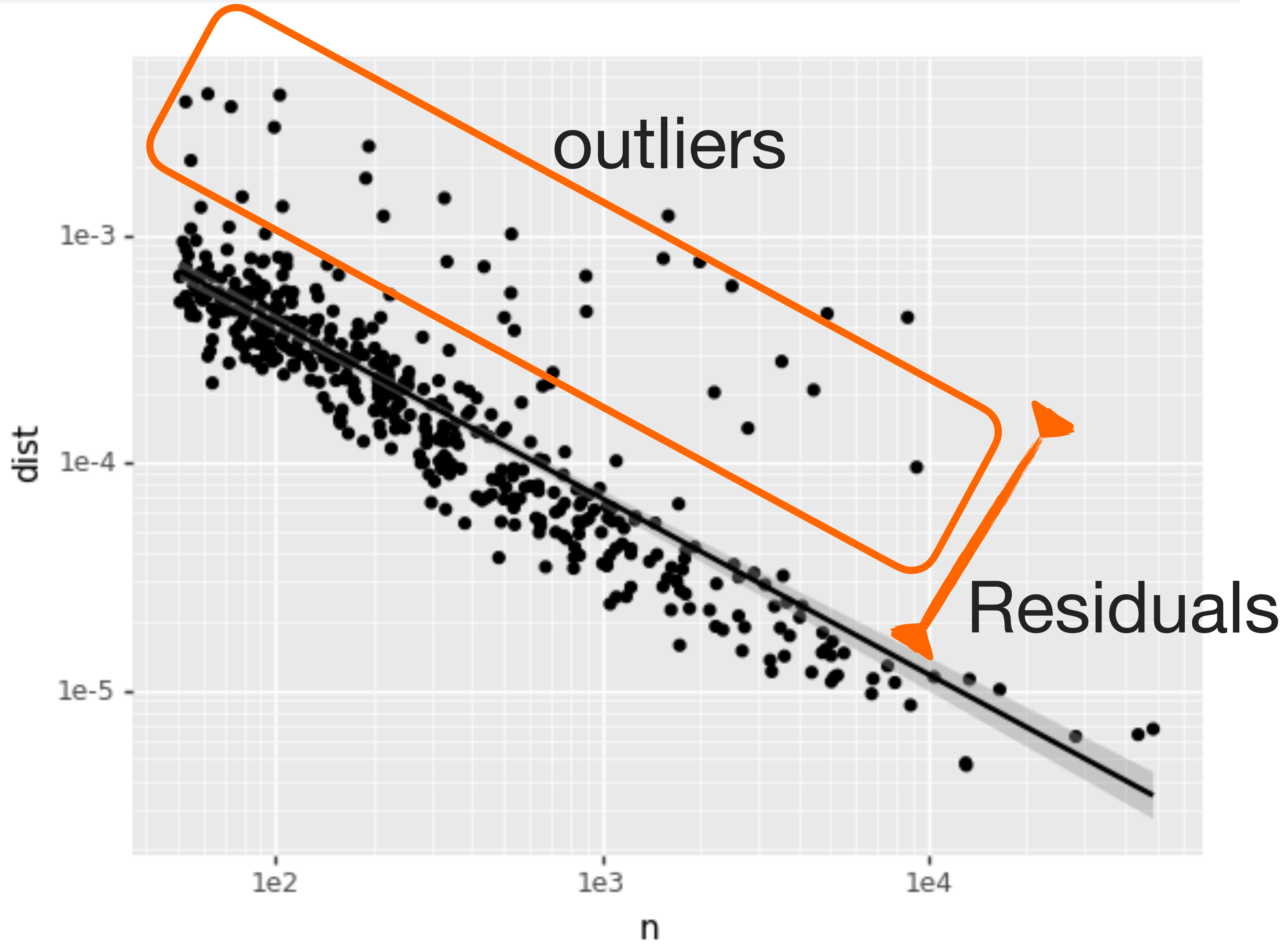
```
# fit a linear model
import statsmodels.api as sm
import numpy as np
# log transformation
devi["log_n"] = np.log(devi["n"])
devi["log_dist"] = np.log(devi["dist"])
# fit a linear model
model = sm.OLS.from_formula("log_dist ~ log_n", data=devi)
result = model.fit()
result.summary()
```

OLS Regression Results

Dep. Variable:	log_dist	**R-squared:**	0.751
Model:	OLS	**Adj. R-squared:**	0.750
Method:	Least Squares	**F-statistic:**	1348.
Date:	Mon, 20 Feb 2023	**Prob (F-statistic):**	3.74e-137
Time:	17:54:58	**Log-Likelihood:**	-436.72
No. Observations:	450	**AIC:**	877.4
Df Residuals:	448	**BIC:**	885.7
Df Model:	1		
Covariance Type:	nonrobust		

Log-transformed

```
ggplot(devi, aes(x = 'n', y = 'dist')) + geom_point()+\
scale_x_log10() +\
scale_y_log10() +\
geom_smooth(method = "lm")
```



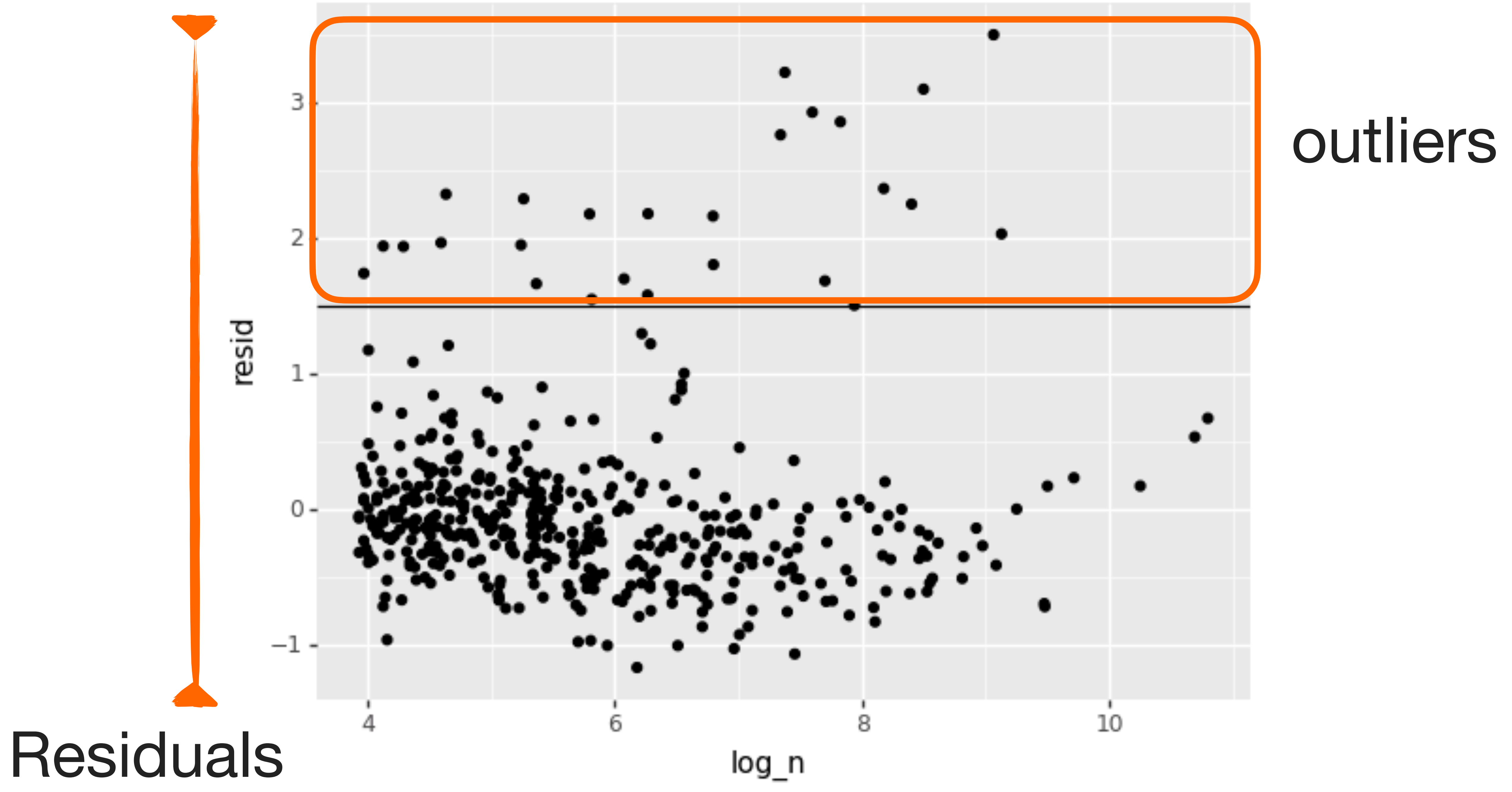
Each dot represents a COD

Case Study - Find the Residuals

devi

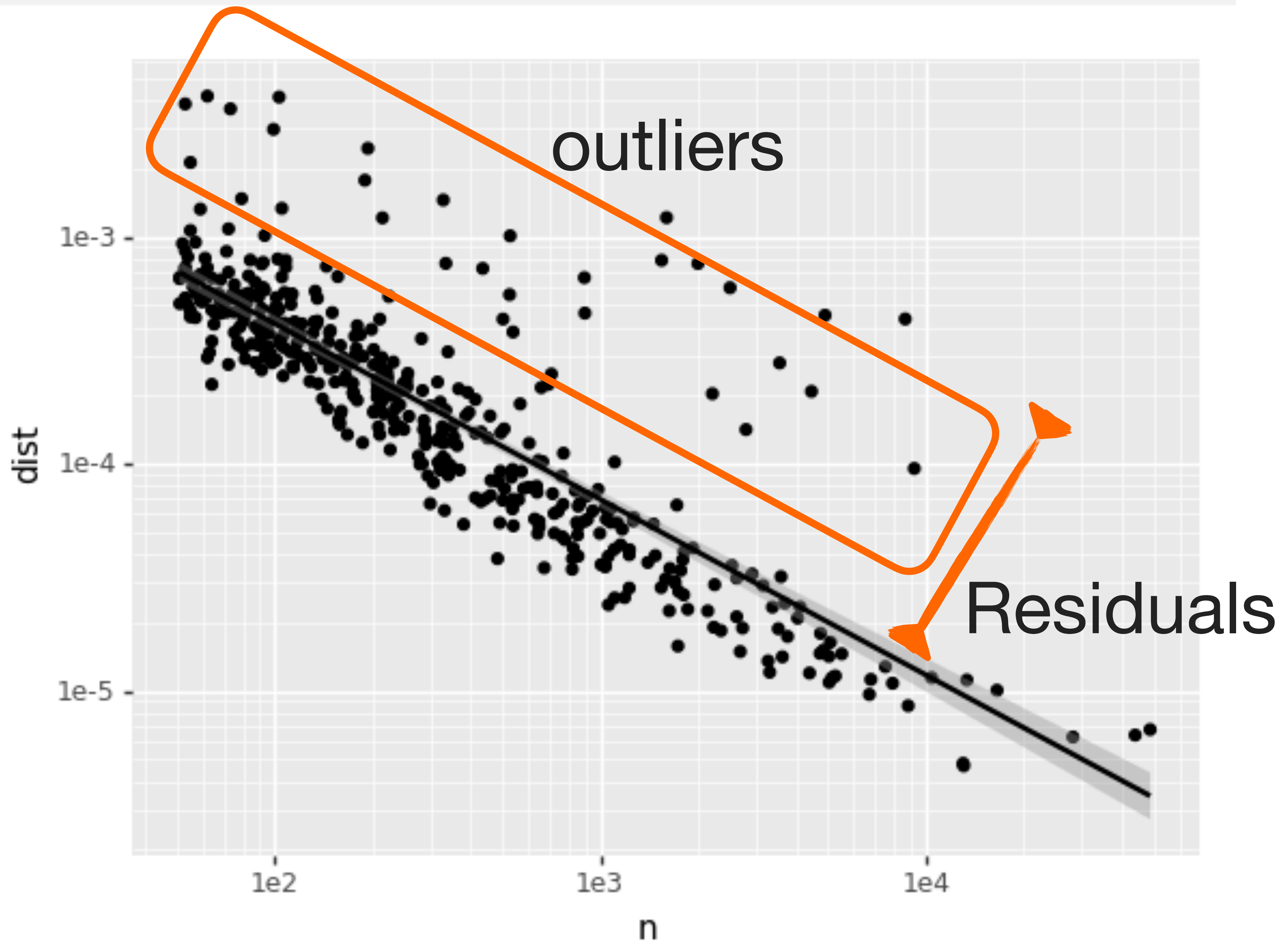
	cod	n	dist	log_n	log_dist	resid
0	A01	51	0.000958	3.931826	-6.950290	0.290330
1	A02	62	0.000733	4.127134	-7.218939	0.178910
3	A04	144	0.000185	4.969813	-8.596625	-0.520395
5	A06	88	0.000360	4.477337	-7.929282	-0.249510
8	A09	3111	0.000030	8.042699	-10.422575	0.127419
...	...	...	...	...	...	...
1172	Y33	60	0.000627	4.094345	-7.373953	-0.002500
1173	Y34	780	0.000068	6.659294	-9.596346	-0.160034
1183	Y57	111	0.000284	4.709530	-8.165902	-0.299207
1190	Y83	174	0.000203	5.159055	-8.501431	-0.272856
1193	Y86	363	0.000094	5.894403	-9.270435	-0.449883

```
ggplot(devi, aes(x = 'log_n', y = 'resid')) + geom_point() + geom_hline(yintercept = 1.5)
```



Log-transformed

```
ggplot(devi, aes(x = 'n', y = 'dist')) + geom_point()+\
scale_x_log10() +\
scale_y_log10() +\
geom_smooth(method = "lm")
```



Each dot represents a COD

Case Study - Visualization of the Outliers