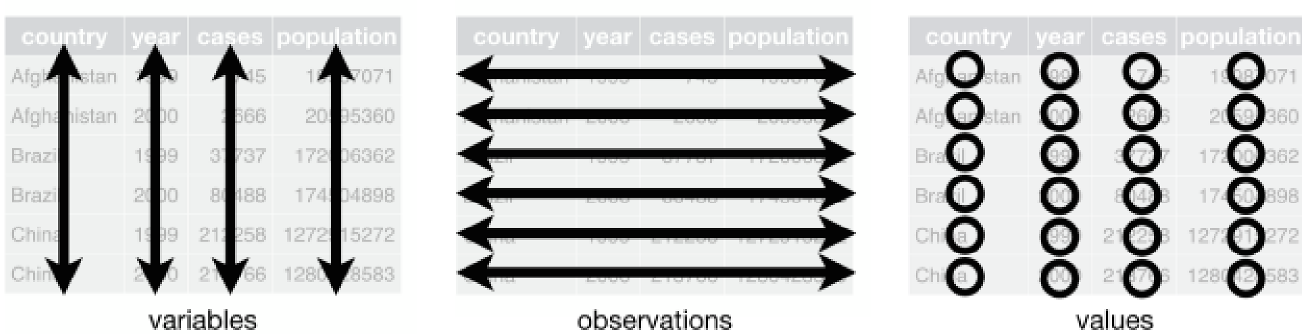
Published in Towards Data Science

Rodrigo Marino Follow

Jan 15, 2020 · 7 min read · · Listen

What's Tidy Data?

How to organize messy datasets in Python with Melt and Pivotable functions



Source: R for Data Science (Hadley Wickham & Garrett Grolemund)

Data scientists spend about 80% of their time cleaning and organizing the data. Tidy Data is a way of structuring datasets to facilitate analysis.

In 2014, Hadley Wickham published an awesome paper named Tidy Data, that describes the process of tidying a dataset in R. My goal with this article is to summarize these steps and show the code in Python.

2. Each observation must have its own row.

3. Each type of observational unit forms a table.

Messy data is any other arrangement of the data.

Messy Data

There are 5 examples of messy data we will explore here:

- 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.

Of course, there are more types of messiness that are not shown above, but they can be tidied in a similar way.

Column headers are values, not variable names.

For this example, we will use the dataset "relinc.csv" which explores the relationship between income and religion. Note, despite being messy, this arrangement can be useful in some cases, so we will learning how to tidy and untidy it.

```
import pandas as pd

df_relinc=pd.read_csv("relinc.csv")
df_relinc.head()
```

1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	refused	15	14	15	11	10	35	21	17	18	116

There are three variables in this dataset: religion, income, and frequency. The column headers are values, not variable names, so we need to turn the variables in the columns (income) into rows. We will use Panda's function **Melt.**

```
# Applying melt (to a long format)

df_relinc=df_relinc.melt(id_vars=["religion"],var_name=
["income"],value_name="frequency")

df_relinc.head()
```

	religion	income	frequency
0	Agnostic	10-20k	34
1	Atheist	10-20k	27
2	Buddhist	10-20k	21
3	Catholic	10-20k	617
4	Evangelical Prot	10-20k	869

The output above is our tidy version of the dataset.

To return the dataset to a wide format we will use Panda's function **pivot_table**.

```
# Applying pivot_table (to a wide format)

df_relinc=(df_relinc.pivot_table(index = "religion", columns =
"income", values = "frequency")
    .reset_index()
```

0	Agnostic	34	109	60	81	76	137	122	27	84	96
1	Atheist	27	59	37	52	35	70	73	12	74	76
2	Buddhist	21	39	30	34	33	58	62	27	53	54
3	Catholic	617	792	732	670	638	1116	949	418	633	1489
4	Evangelical Prot	869	723	1064	982	881	1486	949	575	414	1529

Multiple variables stored in one column.

Now we will explore the tuberculosis dataset from the World Health Organisation. The records show the count of tuberculosis cases by country, year and demographic group.

The demographic groups are broken down by sex (m, f) and age (0–14, 15–24, 25–34, 35–44, 45–54, 55–64, 65+, unknown).

```
df_tb=pd.read_csv('tb.csv')
df_tb.columns
```

```
Index(['iso2', 'year', 'm014', 'm1524', 'm2534', 'm3544', 'm4554', 'm5564',
       'm65', 'mu', 'f014', 'f1524', 'f2534', 'f3544', 'f4554', 'f5564', 'f65',
       'fu'],
      dtype='object')
```

```
df_tb.tail()
```

	iso2	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f014	f1524	f2534	f3544	f4554	f5564	f65	fu
5764	ZW	2004	187.0	833.0	2908.0	2298.0	1056.0	366.0	198.0	NaN	225.0	1140.0	2858.0	1565.0	622.0	214.0	111.0	NaN
5765	ZW	2005	210.0	837.0	2264.0	1855.0	762.0	295.0	656.0	NaN	269.0	1136.0	2242.0	1255.0	578.0	193.0	603.0	NaN
5766	ZW	2006	215.0	736.0	2391.0	1939.0	896.0	348.0	199.0	NaN	237.0	1020.0	2424.0	1355.0	632.0	230.0	96.0	NaN
5767	ZW	2007	138.0	500.0	3693.0	0.0	716.0	292.0	153.0	NaN	185.0	739.0	3311.0	0.0	553.0	213.0	90.0	NaN
5768	ZW	2008	127.0	614.0	0.0	3316.0	704.0	263.0	185.0	0.0	145.0	840.0	0.0	2890.0	467.0	174.0	105.0	0.0

First, we will gather up the non-variable columns, by moving the age range and sex to a single column. To do so, we will use the **Melt.**

```
# Applying melt (to a long format)

df_tb=df_tb.melt(id_vars=["iso2","year"],var_name=
["demographic"],value_name="cases")

df_tb.sample(5)
```

	iso2	year	demographic	cases
37765	MA	1994	m3544	NaN
112546	LC	2003	f65	1.0
44942	SC	2007	m4554	NaN
101567	MT	1986	f4554	NaN
47341	CR	1984	m5564	NaN

Now we need to split the column demographic to get two columns for the variables sex and age.

```
# Creating new columns for sex and age

df_tb=(df_tb.assign(

sex = lambda x: x.demographic.str[0].astype(str),
age = lambda x: x.demographic.str[1:].astype(str))
      .drop("demographic",axis=1))

df_tb.sample(5)
```

69752	BG	2001	5.0	f	4554
41676	CY	2007	0.0	m	u
16985	VC	1986	NaN	m	2534
62726	TH	2008	1513.0	f	2534

Now, each observation has its own row and each variable has its own column. We just tidied our dataset! Before go ahead, let's clean the data.

```
# Styling the dataset

df_tb.update(pd.DataFrame({"age":[age[:2]+'-'+age[2:] if len(age) == 4
else (age) for age in df_tb["age"]]}))

df_tb=(df_tb.replace(to_replace =["m","f","014","65","u"],value =
["Male","Female","0-14","65+","unknown"])
              .dropna())

df_tb.sample(10)
```

	iso2	year	cases	sex	age
31225	IQ	1995	900.0	Male	55-64
67332	NO	1996	5.0	Female	35-44
37026	IR	1998	579.0	Male	65+
78850	NL	2005	1.0	Female	55-64
7932	HN	2005	238.0	Male	15-24
34992	BA	1997	74.0	Male	65+
60899	MD	1998	20.0	Female	25-34
76935	GN	1995	37.0	Female	55-64
11751	AO	2006	3049.0	Male	25-34
74722	VE	2004	184.0	Female	45-54

Variables are stored in both rows and columns.

We will use the data from the Global Historical Climatology Network that represents with fewer than 31 days have structural missing values for the last day(s) of the month. The columns d9 to d31 have been omitted for better visualization.

```
import datetime

df_weather = pd.read_csv('weather-raw.csv')

df_weather.sample(5)
```

	id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
5	MX17004	2010	3	tmin	NaN	NaN	NaN	NaN	14.2	NaN	NaN	NaN
4	MX17004	2010	3	tmax	NaN	NaN	NaN	NaN	32.1	NaN	NaN	NaN
8	MX17004	2010	5	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	MX17004	2010	4	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
0	MX17004	2010	1	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

As seen above, the dataset is messy. Variables are stored in both rows (tmin, tmax) and columns (days). Let's start by working on d1, d2, d3… columns.

We will apply **melt** to create a row for each record for the day variable.

```
# Applying melt (to a long format)

df_weather=df_weather.melt(id_vars=
["id","year","month","element"],var_name=["day"],value_name="temp")
df_weather.update(pd.DataFrame({"day":[day[1:] for day in
df_weather["day"]]}))

df_weather.sample(5)
```

34	MX17004	2010	3	tmax	4	NaN
37	MX17004	2010	4	tmin	4	NaN
58	MX17004	2010	5	tmax	6	NaN
53	MX17004	2010	2	tmin	6	NaN

Now, we will use **pivot_table** function to create new columns for the tmin and tmax, once they are variables.

```
# applying pivot_table to create columns for tmin and tmax

df_weather=(df_weather.pivot_table(index =
["year","month","day","id"], columns = "element", values = "temp")
       .reset_index().rename_axis(None, axis = 1))

df_weather
```

	year	month	day	id	tmax	tmin
0	2010	2	2	MX17004	NaN	14.4
1	2010	2	2	MX17004	27.3	NaN
2	2010	2	3	MX17004	NaN	14.4
3	2010	2	3	MX17004	24.1	NaN
4	2010	3	5	MX17004	32.1	14.2

The dataset looks better but we still need to improve it. Let's create a column for the dates and group it.

```
# Creating a date column

df_weather=(df_weather.assign(date = lambda x: x.year.astype("str")
+"/"+ x.month.astype("str").str.zfill(2) +"/"+
x.day.astype("str").str.zfill(2))
            .drop(["year", "month","day"],axis=1))
df_weather['date'] =  pd.to_datetime(df_weather['date'],
format='%Y/%m/%d')

# Grouping by date
```

Open in app Get started

```
df_weather
```

	date	tmax	tmin
0	2010-02-02	27.3	14.4
1	2010-02-03	24.1	14.4
2	2010-03-05	32.1	14.2

We finally tidied our dataset.

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

The dataset shows the Billboard top hits for 2000. This dataset records the date a song first entered the Billboard Top 100. It has variables for artist, track, date entered, date peaked, genre, time, rank and week.

```
import pandas as pd
import re
import numpy as np
import datetime

df_bill = pd.read_csv('billboard.csv',header=0,encoding =
'unicode_escape')

df_bill.head()
```

	year	artist.inverted	track	time	genre	date.entered	date.peaked	x1st.week	x2nd.week	x3rd.week
0	2000	Destiny's Child	Independent Women Part I	3:38	Rock	2000-09-23	2000-11-18	78	63.0	49.0
1	2000	Santana	Maria, Maria	4:18	Rock	2000-02-12	2000-04-08	15	8.0	6.0
2	2000	Savage Garden	I Knew I Loved You	4:07	Rock	1999-10-23	2000-01-29	71	48.0	43.0
3	2000	Madonna	Music	3:45	Rock	2000-08-12	2000-09-16	41	23.0	18.0
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is	3:38	Rock	2000-08-05	2000-10-14	57	47.0	45.0

are filled with NaN.

```
df_bill.columns
```

```
Index(['year', 'artist.inverted', 'track', 'time', 'genre', 'date.entered',
       'date.peaked', 'x1st.week', 'x2nd.week', 'x3rd.week', 'x4th.week',
       'x5th.week', 'x6th.week', 'x7th.week', 'x8th.week', 'x9th.week',
       'x10th.week', 'x11th.week', 'x12th.week', 'x13th.week', 'x14th.week',
       'x15th.week', 'x16th.week', 'x17th.week', 'x18th.week', 'x19th.week',
       'x20th.week', 'x21st.week', 'x22nd.week', 'x23rd.week', 'x24th.week',
       'x25th.week', 'x26th.week', 'x27th.week', 'x28th.week', 'x29th.week',
       'x30th.week', 'x31st.week', 'x32nd.week', 'x33rd.week', 'x34th.week',
       'x35th.week', 'x36th.week', 'x37th.week', 'x38th.week', 'x39th.week',
       'x40th.week', 'x41st.week', 'x42nd.week', 'x43rd.week', 'x44th.week',
       'x45th.week', 'x46th.week', 'x47th.week', 'x48th.week', 'x49th.week',
       'x50th.week', 'x51st.week', 'x52nd.week', 'x53rd.week', 'x54th.week',
       'x55th.week', 'x56th.week', 'x57th.week', 'x58th.week', 'x59th.week',
       'x60th.week', 'x61st.week', 'x62nd.week', 'x63rd.week', 'x64th.week',
       'x65th.week', 'x66th.week', 'x67th.week', 'x68th.week', 'x69th.week',
       'x70th.week', 'x71st.week', 'x72nd.week', 'x73rd.week', 'x74th.week',
       'x75th.week', 'x76th.week'],
      dtype='object')
```

This dataset contains observations on two types of observational units: the song and its rank in each week. As a consequence of it, the artist and time are repeated for every song in each week. Before break the Billboard dataset into two, we need to tidy it. Let's start by gathering all the week columns.

```
# Applying melt (to a long format)
df_bill=(df_bill.melt(id_vars=
["year","artist.inverted","track","genre","date.entered","date.peaked"
,"time"],var_name=["week"],value_name="rank"))

# Week to number
df_bill.update(pd.DataFrame({"week": np.ravel([list(map(int,
re.findall(r'\d+', i))) for i in df_bill["week"]])}))

df_bill.head()
```

1	2000	Santana	Maria, Maria	Rock	2000-02-12	2000-04-08	4:18	1	15.0
2	2000	Savage Garden	I Knew I Loved You	Rock	1999-10-23	2000-01-29	4:07	1	71.0
3	2000	Madonna	Music	Rock	2000-08-12	2000-09-16	3:45	1	41.0
4	2000	Aguilera, Christina	Come On Over Baby (All I Want Is You)	Rock	2000-08-05	2000-10-14	3:38	1	57.0

It looks better! Now we have a column for the variable week. By the way, we can use the information from the date entered and the week to create a new column, which will be the date column.

```
# creating a date column from date.entered and week

df_bill['date.entered'] =  pd.to_datetime(df_bill['date.entered'], format='%Y/%m/%d')

df_bill=(df_bill.assign(date= [df_bill['date.entered'][i]+datetime.timedelta(weeks = df_bill["week"][i]-1) for i in range(len(df_bill["week"]))])
         .drop(['date.entered','date.peaked','week'], axis=1)
         .sort_values('artist.inverted', ascending=True)
         .reset_index(drop=True))

df_bill.head()
```

	year	artist.inverted	track	genre	time	rank	date
0	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-11-25
1	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2001-07-21
2	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-09-16
3	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2001-02-24
4	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-12-16

Now, we will create an id from the track. Each song must have a unique id number. To do so, we will use Panda's function **factorize.**

```
# creating an id column from track
```

	year	artist.inverted	track	genre	time	rank	date	id
0	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-11-25	1
1	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2001-07-21	1
2	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-09-16	1
3	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2001-02-24	1
4	2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	Rap	4:22	NaN	2000-12-16	1

Finally, we will break our dataset into two datasets: the song dataset and the rank dataset.

```
# creating a new dataframe for rank

df_rank=df_bill.filter(["id", "date", "rank"]).dropna()
df_rank=df_rank.sort_values(by=['id','date']).reset_index(drop=True)

# creating a new dataframe for song

df_song=df_bill.filter(["id", "artist.inverted", "track","time"])
df_song=df_song.drop_duplicates('id').reset_index(drop=True)

df_rank.head(10)
df_song.head()
```

	id	date	rank
0	1	2000-02-26	87.0
1	1	2000-03-04	82.0
2	1	2000-03-11	72.0
3	1	2000-03-18	77.0
4	1	2000-03-25	87.0
5	1	2000-04-01	94.0
6	1	2000-04-08	99.0
7	2	2000-09-02	91.0
8	2	2000-09-09	87.0
9	2	2000-09-16	92.0

Open in app Get started

1	2	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15
2	3	3 Doors Down	Loser	4:24
3	4	3 Doors Down	Kryptonite	3:53
4	5	504 Boyz	Wobble Wobble	3:35

We just tackled the problem of multiple types of observational units stored in the same table!

A single observational unit is stored in multiple tables.

This problem uses to be easy to fix. We basically need to read the tables, to add a new column that records the original file name and finally combine all tables into a single one.

```
import pandas as pd

df_baby14 = pd.read_csv("2014-baby-names-illinois.csv")
df_baby15 = pd.read_csv("2015-baby-names-illinois.csv")

df_baby14.head()
```

	rank	name	frequency	sex
0	1	Noah	837	Male
1	2	Alexander	747	Male
2	3	William	687	Male
3	4	Michael	680	Male
4	5	Liam	670	Male

```
df_baby15.head()
```

1	2	Liam	709	Male
2	3	Alexander	703	Male
3	4	Jacob	650	Male
4	5	William	618	Male

Let's create a column year in each dataset according to the file name. Finally, we will apply Panda's **concat** function to concatenate the data frames.

```
# Creating a column for the year
df_baby14["year"]="2014"
df_baby15["year"]="2015"

# Concatenating the datasets
df_baby = pd.concat([df_baby14, df_baby15]).sort_values(by=['rank'])

(df_baby.set_index('rank', inplace=True))

df_baby.head()
```

rank	name	frequency	sex	year
1	Noah	837	Male	2014
1	Noah	863	Male	2015
2	Alexander	747	Male	2014
2	Liam	709	Male	2015
3	William	687	Male	2014

Final Comments

The goal of this article was to explain the concept of Tidy Data, by covering the five most commons types of messy data, as well as how to organize and clean these datasets in Python.

If you find any mistakes, please don't hesitate to contact me! I started to surf the data science world recently and in spite of loving it, I am a dummy.

more information about Tidy Data: https://github.com/hadley
