Tidy Data in Python

06 Dec 2016

I recently came across a paper named Tidy Data by Hadley Wickham. Published back in 2014, the paper focuses on one aspect of cleaning up data, tidying data: structuring datasets to facilitate analysis. Through the paper, Wickham demonstrates how any dataset can be structured in a standardized way prior to analysis. He presents in detail the different types of data sets and how to wrangle them into a standard format.

As a data scientist, I think you should get very familiar with this standardized structure of a dataset. Data cleaning is one the most frequent task in data science. No matter what kind of data you are dealing with or what kind of analysis you are performing, you will have to clean the data at some point. Tidying your data in a standard format makes things easier down the road. You can reuse a standard set of tools across your different analysis.

In this post, I will summarize some tidying examples Wickham uses in his paper and I will demonstrate how to do so using the Python pandas library.

Defining tidy data

The structure Wickham defines as tidy has the following attributes:

- Each *variable* forms a column and contains *values*

- Each *observation* forms a row
- Each type of *observational unit* forms a table

A few definitions:

- Variable: A measurement or an attribute. *Height, weight, sex, etc.*
- Value: The actual measurement or attribute. *152 cm, 80 kg, female, etc.*
- Observation: All values measure on the same unit. *Each person.*

An example of a *messy dataset*:

	Treatment A	**Treatment B**
John Smith	-	2
Jane Doe	16	11
Mary Johnson	3	1

An example of a *tidy dataset*:

Name	**Treatment**	**Result**
John Smith	a	-
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

Tidying messy datasets

Through the following examples extracted from Wickham's paper, we'll wrangle messy datasets into the tidy format. The goal here is not to analyze the datasets but rather prepare them in a standardized way prior to the analysis. These are the five types of messy datasets we'll tackle:

- 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.

Note: All of the code presented in this post is available on Github.

Column headers are values, not variable names

Pew Research Center Dataset

This dataset explores the relationship between income and religion.

Problem: The columns headers are composed of the possible income values.

```
import pandas as pd
import datetime
from os import listdir
from os.path import isfile, join
import glob
import re

df = pd.read_csv("./data/pew-raw.csv")
df
```

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k
Agnostic	27	34	60	81	76	137
Atheist	12	27	37	52	35	70
Buddhist	27	21	30	34	33	58
Catholic	418	617	732	670	638	1116
Dont know/refused	15	14	15	11	10	35
Evangelical Prot	575	869	1064	982	881	1486

Hindu	1	9	7	9	11	34
Historically Black Prot	228	244	236	238	197	223
Jehovahs Witness	20	27	24	24	21	30
Jewish	19	19	25	25	30	95

A tidy version of this dataset is one in which the income values would not be columns headers but rather values in an `income` column. In order to tidy this dataset, we need to melt it. The *pandas* library has a built-in function that allows to do just that. It "unpivots" a DataFrame from a wide format to a long format. We'll reuse this function a few times through the post.

```
formatted_df = pd.melt(df,
                       ["religion"],
                       var_name="income",
                       value_name="freq")
formatted_df = formatted_df.sort_values(by=["relig
ion"])
formatted_df.head(10)
```

This outputs a tidy version of the dataset:

religion	income	freq
Agnostic	<$10k	27
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Atheist	$40-50k	35
Atheist	$20-30k	37
Atheist	$10-20k	27
Atheist	$30-40k	52

Billboard Top 100 Dataset

This dataset represents the weekly rank of songs from

the moment they enter the Billboard Top 100 to the subsequent 75 weeks.

Problems:

- The columns headers are composed of values: the week number (`x1st.week` , ...)
- If a song is in the Top 100 for less than 75 weeks, the remaining columns are filled with missing values.

```
df = pd.read_csv("./data/billboard.csv", encoding=
"mac_latin2")
df.head(10)
```

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

		You)							
2000	Janet	Doesn't Really Matter	4:17	Rock	2000-06-17	2000-08-26	59	52.0	...
2000	Destiny's Child	Say My Name	4:31	Rock	1999-12-25	2000-03-18	83	83.0	...
2000	Iglesias, Enrique	Be With You	3:36	Latin	2000-04-01	2000-06-24	63	45.0	...
2000	Sisqo	Incomplete	3:52	Rock	2000-06-24	2000-08-12	77	66.0	...
2000	Lonestar	Amazed	4:25	Country	1999-06-05	2000-03-04	81	54.0	...

A tidy version of this dataset is one without the week's numbers as columns but rather as values of a single column. In order to do so, we'll *melt* the weeks columns into a single `date` column. We will create one row per week for each record. If there is no data for the given week, we will not create a row.

```
# Melting
id_vars = ["year",
           "artist.inverted",
           "track",
           "time",
           "genre",
           "date.entered",
           "date.peaked"]

df = pd.melt(frame=df,id_vars=id_vars, var_name="week", value_name="rank")

# Formatting
df["week"] = df['week'].str.extract('(\d+)', expand=False).astype(int)
df["rank"] = df["rank"].astype(int)

# Cleaning out unnecessary rows
df = df.dropna()
```

```
# Create "date" columns
df['date'] = pd.to_datetime(df['date.entered']) +
pd.to_timedelta(df['week'], unit='w') - pd.DateOff
set(weeks=1)

df = df[["year",
         "artist.inverted",
         "track",
         "time",
         "genre",
         "week",
         "rank",
         "date"]]
df = df.sort_values(ascending=True, by=["year","ar
tist.inverted","track","week","rank"])

# Assigning the tidy dataset to a variable for fut
ure usage
billboard = df

df.head(10)
```

A tidier version of the dataset is shown below. There is still a lot of repetition of the song details: the track name, time and genre. For this reason, this dataset is still not completely tidy as per Wickham's definition. We will address this in the next example.

year	artist.inverted	track	time	genre	week	rank	date
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	1	87	2000-02-26
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	2	82	2000-03-04
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	3	72	2000-03-11
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	4	77	2000-03-18
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	5	87	2000-03-25
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	6	94	2000-04-01
		Baby Don't					2000-

2000	2 Pac	Cry (Keep Ya Head Up II)	4:22	Rap	7	99	04-08
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	1	91	2000-09-02
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	2	87	2000-09-09
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	3	92	2000-09-16

Multiple types in one table

Following up on the Billboard dataset, we'll now address the repetition problem of the previous table.

Problems:

- Multiple observational units (the `song` and its `rank`) in a single table.

We'll first create a `songs` table which contains the details of each song:

```
songs_cols = ["year", "artist.inverted", "track",
"time", "genre"]
songs = billboard[songs_cols].drop_duplicates()
songs = songs.reset_index(drop=True)
songs["song_id"] = songs.index
songs.head(10)
```

year	artist.inverted	track	time	genre	song_id
2000	2 Pac	Baby Don't Cry (Keep Ya Head Up II)	4:22	Rap	0
2000	2Ge+her	The Hardest Part Of Breaking Up (Is Getting Ba...	3:15	R&B	1
2000	3 Doors Down	Kryptonite	3:53	Rock	2

2000	3 Doors Down	Loser	4:24	Rock	3
2000	504 Boyz	Wobble Wobble	3:35	Rap	4
2000	98	Give Me Just One Night (Una Noche)	3:24	Rock	5
2000	A*Teens	Dancing Queen	3:44	Pop	6
2000	Aaliyah	I Don't Wanna	4:15	Rock	7
2000	Aaliyah	Try Again	4:03	Rock	8
2000	Adams, Yolanda	Open My Heart	5:30	Gospel	9

We'll then create a `ranks` table which only contains the `song_id`, `date` and the `rank`.

```
ranks = pd.merge(billboard, songs, on=["year","artist.inverted", "track", "time", "genre"])
ranks = ranks[["song_id", "date","rank"]]
ranks.head(10)
```

song_id	date	rank
0	2000-02-26	87
0	2000-03-04	82
0	2000-03-11	72
0	2000-03-18	77
0	2000-03-25	87
0	2000-04-01	94
0	2000-04-08	99
1	2000-09-02	91
1	2000-09-09	87
1	2000-09-16	92

Multiple variables stored in one column

Tubercolosis Records from World Health Organization

This dataset documents the count of confirmed tuberculosis cases by country, year, age and sex.

Problems:

- Some columns contain multiple values: sex and age.
- Mixture of zeros and missing values `NaN`. This is due to the data collection process and the distinction is important for this dataset.

```
df = pd.read_csv("./data/tb-raw.csv")
df
```

country	year	m014	m1524	m2534	m3544	m4554	m5564	m65	mu	f0
AD	2000	0	0	1	0	0	0	0	NaN	Na
AE	2000	2	4	4	6	5	12	10	NaN	3
AF	2000	52	228	183	149	129	94	80	NaN	93
AG	2000	0	0	0	0	0	0	1	NaN	1
AL	2000	2	19	21	14	24	19	16	NaN	3
AM	2000	2	152	130	131	63	26	21	NaN	1
AN	2000	0	0	1	2	0	0	0	NaN	0
AO	2000	186	999	1003	912	482	312	194	NaN	24
AR	2000	97	278	594	402	419	368	330	NaN	12
AS	2000	NaN	NaN	NaN	NaN	1	1	NaN	NaN	Na

In order to tidy this dataset, we need to remove the different values from the header and unpivot them into rows. We'll first need to melt the `sex + age group` columns into a single one. Once we have that single column, we'll derive three columns from it: `sex`, `age_lower` and `age_upper`. With those, we'll be able to properly build a tidy dataset.

```
df = pd.melt(df, id_vars=["country","year"], value
_name="cases", var_name="sex_and_age")

# Extract Sex, Age lower bound and Age upper bound
group
tmp_df = df["sex_and_age"].str.extract("(\D)(\d+)(
\d{2})")

# Name columns
tmp_df.columns = ["sex", "age_lower", "age_upper"]
```

```
# Create `age`column based on `age_lower` and `age
_upper`
tmp_df["age"] = tmp_df["age_lower"] + "-" + tmp_df
["age_upper"]

# Merge
df = pd.concat([df, tmp_df], axis=1)

# Drop unnecessary columns and rows
df = df.drop(['sex_and_age',"age_lower","age_upper
"], axis=1)
df = df.dropna()
df = df.sort(ascending=True,columns=["country", "y
ear", "sex", "age"])
df.head(10)
```

This results in a *tidy dataset*.

country	year	cases	sex	age
AD	2000	0	m	0-14
AD	2000	0	m	15-24
AD	2000	1	m	25-34
AD	2000	0	m	35-44
AD	2000	0	m	45-54
AD	2000	0	m	55-64
AE	2000	3	f	0-14
AE	2000	2	m	0-14
AE	2000	4	m	15-24
AE	2000	4	m	25-34

Variables are stored in both rows and columns

Global Historical Climatology Network Dataset

This dataset represents the daily weather records for a weather station (MX17004) in Mexico for five months in 2010.

Problems:

- Variables are stored in both rows (`tmin` , `tmax`) and columns (`days`).

```
df = pd.read_csv("./data/weather-raw.csv")
df
```

id	year	month	element	d1	d2	d3	d4	d5	d6	d7	d8
MX17004	2010	1	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MX17004	2010	1	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MX17004	2010	2	tmax	NaN	27.3	24.1	NaN	NaN	NaN	NaN	NaN
MX17004	2010	2	tmin	NaN	14.4	14.4	NaN	NaN	NaN	NaN	NaN
MX17004	2010	3	tmax	NaN	NaN	NaN	NaN	32.1	NaN	NaN	NaN
MX17004	2010	3	tmin	NaN	NaN	NaN	NaN	14.2	NaN	NaN	NaN
MX17004	2010	4	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MX17004	2010	4	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MX17004	2010	5	tmax	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
MX17004	2010	5	tmin	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In order to make this dataset tidy, we want to move the three misplaced variables (`tmin` , `tmax` and `days`) as three individual columns: `tmin` . `tmax` and `date` .

```
# Extracting day
df["day"] = df["day_raw"].str.extract("d(\d+)", ex
pand=False)
df["id"] = "MX17004"

# To numeric values
df[["year","month","day"]] = df[["year","month","d
ay"]].apply(lambda x: pd.to_numeric(x, errors='ign
ore'))

# Creating a date from the different columns
def create_date_from_year_month_day(row):
    return datetime.datetime(year=row["year"], mon
th=int(row["month"]), day=row["day"])

df["date"] = df.apply(lambda row: create_date_from
_year_month_day(row), axis=1)
df = df.drop(['year',"month","day", "day_raw"], ax
is=1)
df = df.dropna()

# Unmelting column "element"
df = df.pivot_table(index=["id","date"], columns="
element", values="value")
df.reset_index(drop=False, inplace=True)
df
```

id	date	tmax	tmin
MX17004	2010-02-02	27.3	14.4
MX17004	2010-02-03	24.1	14.4
MX17004	2010-03-05	32.1	14.2

One type in multiple tables

Dataset: Illinois Male Baby Names for the year 2014/2015.

Problems:

- The data is spread across multiple tables/files.
- The "Year" variable is present in the file name.

In order to load those different files into a single DataFrame, we can run a custom script that will append the files together. Furthermore, we'll need to extract the "Year" variable from the file name.

```
def extract_year(string):
    match = re.match(".+(\d{4})", string)
    if match != None: return match.group(1)

path = './data'
allFiles = glob.glob(path + "/201*-baby-names-illi
nois.csv")
frame = pd.DataFrame()
df_list= []
for file_ in allFiles:
    df = pd.read_csv(file_,index_col=None, header=
0)
    df.columns = map(str.lower, df.columns)
    df["year"] = extract_year(file_)
    df_list.append(df)

df = pd.concat(df_list)
df.head(5)
```

rank	name	frequency	sex	year
1	Noah	837	Male	2014
2	Alexander	747	Male	2014

3	William	687	Male	2014
4	Michael	680	Male	2014
5	Liam	670	Male	2014

Final Thoughts

In this post, I focused on one aspect of Wickham's paper, the data manipulation part. My main goal was to demonstrate the data manipulations in Python. It's important to mention that there is a significant section of his paper that covers the tools and visualizations from which you can benefit by tidying your dataset. I did not cover those in this post.

Overall, I enjoyed preparing this post and wrangling the datasets into a streamlined format. The defined format makes it easier to query and filter the data. This approach makes it easier to reuse libraries and code across analysis. It also makes it easier to share a dataset with other data analysts.

First Name

Email Address

Subscribe

Learn How to Get Started in Data Science

Subscribe to get a weekly email that will get you one step forward to becoming a data scientist.

Every week, you will get fresh content in your inbox. I cover the topics of Data Analysis in Python, Applied Statistics, Machine Learning, SQL etc.

First Name

Email Address

Subscribe

Related Posts

Profiling a Dataset of Craft Beers 23 Apr 2017

What is a Data Scientist? 06 Feb 2017

Scraping for Craft Beers 17 Jan 2017