Wide and Long Data Formats

Until now, we have only worked with dataframes where each column represents a variable and each row an observation. For example, the following table is one way of representing information.

Firstname	Lastname	Age	Account Type	Balance
Harsh	Goyal	50	Current	4000
Harsh	Goyal	50	Deposit	15000
Harsh	Goyal	50	Saving	56000
Rahul	Anand	64	Current	35000
Rahul	Anand	64	Saving	45000
Sonam	Gupta	25	Current	10000
Sonam	Gupta	25	Saving	70000

This is called the **long data format**.

But data often comes in another format - where certain variables are 'spread' into multiple columns.

For example, the same dataframe may be given to you as follows. Here, the columns 'Savings Balance', 'Current Balance' and 'Deposit Balance' separate columns (rather than observations of a single column 'Account Type'.)

This is called the **wide data format**.

Firstname	Lastname	Age	Current Balance	Saving Balance	Deposit Balance
Harsh	Goyal	50	4000	56000	15000
Rahul	Anand	64	35000	45000	NaN
Sonam	Gupta	25	10000	70000	NaN

Notice that both the tables contain the exact same data, though the *long one has more rows* than the wide one. The *wide dataframe has more columns*, and hence the names *long* and *wide*.

In this section, we will explore the two types of data frames, understand their use cases, and learn to convert dataframes from one format to the other.

Motivation: Wide and Long Data

Structure of dataframes significantly affects the speed and convenience of analysis. As shown above, the same data can be stored in two ways - wide and long. Which one is better for you as a data analyst?

The answer depends upon the purpose and type of analysis you are doing. For e.g. some machine learning algorithms such as recommender systems etc. need data to be in a wide format. On the other hand, you may find data visualisation, basic querying and analyses etc. easier in the wide-format.

Hadley Wickham from RStudio has studied this problem in his paper called **Tidy Data** (http://vita.had.co.nz/papers/tidy-data.pdf).

He defined tidy data as follows:

- Each variable forms a column
- Each observation forms a row
- Each type of observational unit forms a table (e.g. you have multiple tables in the sales dataset - customers, products, shipping, etc. rather than one master table storing everything)

Let's learn how to convert wide to long format and vice-versa.

Wide to Long Format: `pd.melt()`

Say you have a dataframe in the wide format as follows.

In [16]:
```
import numpy as np
import pandas as pd

# creating a wide dataframe
wide_df = pd.read_csv("wide.csv")
wide_df
```

Out[16]:

	Firstname	Lastname	Age	Current Balance	Saving Balance	Deposit Balance
0	Harsh	Goyal	50	4000	56000	15000.0
1	Rahul	Anand	64	35000	45000	NaN
2	Sonam	Gupta	25	10000	70000	NaN

To convert this into a long format, we can use the `pd.melt()` function. You need to identify two types of variables and provide them as arguments to `pd.melt()` :

1. *Identifier variables*: Columns used to identify a unique record (Firstname, Lastname and Age identify a unique person)
2. *Value variables*: Column(s) that are to be *melted* into a single column (Current, Saving and Deposit Balances)

For e.g., in the above dataframe, the `id_vars` are `Firstname, Lastname` and `Age` , while the other three columns are to be *melted into a single one*.

The basic syntax for *melting* value variables is as follows: `pd.melt(df, id_vars, value_vars)` .

In [17]:
```
long_df = pd.melt(wide_df,
        id_vars=['Firstname', 'Lastname', 'Age'],
        value_vars=['Current Balance', 'Saving Balance', 'Deposit Balance'])

long_df
```

Out[17]:

	Firstname	Lastname	Age	variable	value
0	Harsh	Goyal	50	Current Balance	4000.0
1	Rahul	Anand	64	Current Balance	35000.0
2	Sonam	Gupta	25	Current Balance	10000.0
3	Harsh	Goyal	50	Saving Balance	56000.0
4	Rahul	Anand	64	Saving Balance	45000.0
5	Sonam	Gupta	25	Saving Balance	70000.0
6	Harsh	Goyal	50	Deposit Balance	15000.0
7	Rahul	Anand	64	Deposit Balance	NaN
8	Sonam	Gupta	25	Deposit Balance	NaN

The dataframe given above is now in the **long format** - each column is a variable. You can now change the column names.

In [18]:
```
# renaming the newly created columns
long_df = long_df.rename(columns={'variable': 'Account Type', 'value': 'Balance'})
long_df
```

Out[18]:

	Firstname	Lastname	Age	Account Type	Balance
0	Harsh	Goyal	50	Current Balance	4000.0
1	Rahul	Anand	64	Current Balance	35000.0
2	Sonam	Gupta	25	Current Balance	10000.0
3	Harsh	Goyal	50	Saving Balance	56000.0
4	Rahul	Anand	64	Saving Balance	45000.0
5	Sonam	Gupta	25	Saving Balance	70000.0
6	Harsh	Goyal	50	Deposit Balance	15000.0
7	Rahul	Anand	64	Deposit Balance	NaN
8	Sonam	Gupta	25	Deposit Balance	NaN

Long to Wide Format: `pd.pivot_table()`

Alternatively, you sometimes need to convert long dataframes into wide ones. The `pd.pivot_table()` method can be used to do that.

Here is the long version of the dataframe.

In [19]:
```
long_df = pd.read_csv("long.csv")
long_df
```

Out[19]:

	Firstname	Lastname	Age	Account Type	Balance
0	Harsh	Goyal	50	Current	4000
1	Harsh	Goyal	50	Deposit	15000
2	Harsh	Goyal	50	Saving	56000
3	Rahul	Anand	64	Current	35000
4	Rahul	Anand	64	Saving	45000
5	Sonam	Gupta	25	Current	10000
6	Sonam	Gupta	25	Saving	70000

To convert the long dataframe into wide, you need to identify:

- the index variables (which will stay as is in the new dataframe)
- the variable which will be converted to multiple *columns*, i.e. **pivoted**, and
- the column whose *values* will be stored in the newly created cells

The `pd.pivot_table()` method is used as follows:

In [20]:
```
# Specify the index columns, the column which will be pivoted into multiple columns
wide_df = long_df.pivot_table(index=['Firstname', 'Lastname', 'Age'],
                              columns='Account Type',
                              values='Balance')
wide_df
```

Out[20]:

			Account Type	Current	Deposit	Saving
Firstname	**Lastname**	**Age**				
Harsh	**Goyal**	**50**		4000.0	15000.0	56000.0
Rahul	**Anand**	**64**		35000.0	NaN	45000.0
Sonam	**Gupta**	**25**		10000.0	NaN	70000.0

Let's now look at a real dataset to understand how wide-to-long conversion is often beneficial for analysis.

The News Popularity Dataset

The online news popularity dataset (https://archive.ics.uci.edu/ml/datasets/online+news+popularity) contains data regarding news articles published by the news publishing website mashable.com (http://mashable.com/).

Each row represents a news article and the columns represent attributes such as the URL (on which it was published), the number of words in the news article, the types of article (lifestyle, tech, business etc.), the day on which the article was published (Monday-Sunday etc.).

Say you are an analyst at a media company and want to understand which type of articles are shared more than others (lifestyle, tech, business etc.).

The original dataset is available in the **wide format**.

In [21]:
```
import numpy as np
import pandas as pd

# reading the data
df = pd.read_csv("popularity.csv")
df.head()
```

Out[21]:

	url	timedelta	n_tokens_title	n_tokens_content	n_unique_toke
0	http://mashable.com/2013/01/07/amazon-instant-...	731.0	12.0	219.0	0.6635
1	http://mashable.com/2013/01/07/ap-samsung-spon...	731.0	9.0	255.0	0.6047
2	http://mashable.com/2013/01/07/apple-40-billio...	731.0	9.0	211.0	0.5751
3	http://mashable.com/2013/01/07/astronaut-notre...	731.0	9.0	531.0	0.5037
4	http://mashable.com/2013/01/07/att-u-verse-apps/	731.0	13.0	1072.0	0.4156

5 rows × 61 columns

Since there are too many columns, jupyter notebook does not print all of them. There's a quirky solution to that.

In [22]:
```
# To display all columns of the df
from IPython.core.display import HTML
HTML(df.head().to_html())
```

Out[22]:

	url	timedelta	n_tokens_title	n_tokens_content	n_unique_toke
0	http://mashable.com/2013/01/07/amazon-instant-...	731.0	12.0	219.0	0.6635
1	http://mashable.com/2013/01/07/ap-samsung-spon...	731.0	9.0	255.0	0.6047
2	http://mashable.com/2013/01/07/apple-40-billio...	731.0	9.0	211.0	0.5751
3	http://mashable.com/2013/01/07/astronaut-notre...	731.0	9.0	531.0	0.5037
4	http://mashable.com/2013/01/07/att-u-verse-apps/	731.0	13.0	1072.0	0.4156

Have a look at the following columns - from `data_channel_is_lifestyle` till `data_channel_is_world` . Each of these columns represent the type or the 'channel' of the

article.

This is the wide data format.

In [23]:
```
df.loc[:," data_channel_is_lifestyle":" data_channel_is_world"]
```

Out[23]:

	data_channel_is_lifestyle	data_channel_is_entertainment	data_channel_is_bus	data_channe
0	0.0	1.0	0.0	
1	0.0	0.0	1.0	
2	0.0	0.0	1.0	
3	0.0	1.0	0.0	
4	0.0	0.0	0.0	
5	0.0	0.0	0.0	
6	1.0	0.0	0.0	
7	0.0	0.0	0.0	
8	0.0	0.0	0.0	
9	0.0	0.0	0.0	
10	0.0	0.0	0.0	

In the long format, there will be a single categorical variable named 'channel' whose values would be lifestyle, business, tech etc. Let's first convert these columns into integer type (they are boolean).

In [24]:
```
# convert the columns to numeric by applying as.type(int)
df.loc[:," data_channel_is_lifestyle":" data_channel_is_world"] = df.loc[:," data
df.loc[:," data_channel_is_lifestyle":" data_channel_is_world"].head()
```

Out[24]:

	data_channel_is_lifestyle	data_channel_is_entertainment	data_channel_is_bus	data_channel_is_so
0	0	1	0	
1	0	0	1	
2	0	0	1	
3	0	1	0	
4	0	0	0	

Now, say you want to conduct analyses to find:

- Compare the average number of shares for each 'channel type' (lifestyle, social media, tech etc.)
- Sort the channel types in decreasing order of average number of articles

We'll convert this dataframe into the long format and conduct the analysis.

Let's now convert these columns to the long format. The 6 columns shown above are the *value* variables, while all the rest are *identifier* variables.

In [25]:
```
# store the value and id variables in two separate arrays

# store the value variable in one Series
value_vars = df.columns[13:19]

# take the setdiff() to get the rest of the variables
id_vars = np.setdiff1d(df.columns, value_vars)

print(value_vars, "\n")
print(id_vars)
```

```
Index([' data_channel_is_lifestyle', ' data_channel_is_entertainment',
       ' data_channel_is_bus', ' data_channel_is_socmed',
       ' data_channel_is_tech', ' data_channel_is_world'],
      dtype='object')

[' LDA_00' ' LDA_01' ' LDA_02' ' LDA_03' ' LDA_04'
 ' abs_title_sentiment_polarity' ' abs_title_subjectivity'
 ' average_token_length' ' avg_negative_polarity' ' avg_positive_polarity'
 ' global_rate_negative_words' ' global_rate_positive_words'
 ' global_sentiment_polarity' ' global_subjectivity' ' is_weekend'
 ' kw_avg_avg' ' kw_avg_max' ' kw_avg_min' ' kw_max_avg' ' kw_max_max'
 ' kw_max_min' ' kw_min_avg' ' kw_min_max' ' kw_min_min'
 ' max_negative_polarity' ' max_positive_polarity' ' min_negative_polarity'
 ' min_positive_polarity' ' n_non_stop_unique_tokens' ' n_non_stop_words'
 ' n_tokens_content' ' n_tokens_title' ' n_unique_tokens' ' num_hrefs'
 ' num_imgs' ' num_keywords' ' num_self_hrefs' ' num_videos'
 ' rate_negative_words' ' rate_positive_words'
 ' self_reference_avg_sharess' ' self_reference_max_shares'
 ' self_reference_min_shares' ' shares' ' timedelta'
 ' title_sentiment_polarity' ' title_subjectivity' ' weekday_is_friday'
 ' weekday_is_monday' ' weekday_is_saturday' ' weekday_is_sunday'
 ' weekday_is_thursday' ' weekday_is_tuesday' ' weekday_is_wednesday' 'url']
```

Let's now convert the wide dataframe into a long one.

In [26]:
```
# convert into long
long_df = pd.melt(df,
        id_vars=list(id_vars),
        value_vars=list(value_vars))

long_df.head()
```

Out[26]:

	LDA_00	LDA_01	LDA_02	LDA_03	LDA_04	abs_title_sentiment_polarity	abs_title_subjectiv
0	0.500331	0.378279	0.040005	0.041263	0.040123	0.187500	0.0000
1	0.799756	0.050047	0.050096	0.050101	0.050001	0.000000	0.5000
2	0.217792	0.033334	0.033351	0.033334	0.682188	0.000000	0.5000
3	0.028573	0.419300	0.494651	0.028905	0.028572	0.000000	0.5000
4	0.028633	0.028794	0.028575	0.028572	0.885427	0.136364	0.0454

5 rows × 57 columns

The last two columns, variable and value, now store the variable "channel_type" and its value.

Note that you do not need the rows where the value is 0 (they are redundant rows), so let's remove them. Also, let's rename the columns.

In [27]:
```
# rename the columns
long_df = long_df.rename(columns={'variable': 'Channel', 'value': 'Value'})
long_df.head()

# filter by value=1
long_df = long_df.loc[(long_df.Value == 1),:]
long_df.head()
```

Out[27]:

	LDA_00	LDA_01	LDA_02	LDA_03	LDA_04	abs_title_sentiment_polarity	abs_title_subject
6	0.020082	0.114705	0.020024	0.020015	0.825173	0.000000	0.500
11	0.028628	0.028573	0.028596	0.028715	0.885488	0.000000	0.500
18	0.437374	0.200363	0.033457	0.033403	0.295403	0.714286	0.357
28	0.020041	0.020031	0.020015	0.020008	0.919905	0.000000	0.500
34	0.211470	0.025507	0.025141	0.025081	0.712801	0.000000	0.500

5 rows × 57 columns

Now since the entire column `Value` contains the value 1, you can simply drop that.

In [28]:
```
# del long_df['Value']
long_df.head()
```

Out[28]:

	LDA_00	LDA_01	LDA_02	LDA_03	LDA_04	abs_title_sentiment_polarity	abs_title_subject
6	0.020082	0.114705	0.020024	0.020015	0.825173	0.000000	0.500
11	0.028628	0.028573	0.028596	0.028715	0.885488	0.000000	0.500
18	0.437374	0.200363	0.033457	0.033403	0.295403	0.714286	0.357
28	0.020041	0.020031	0.020015	0.020008	0.919905	0.000000	0.500
34	0.211470	0.025507	0.025141	0.025081	0.712801	0.000000	0.500

5 rows × 57 columns

Let's try doing the analyses we started with:

- Compare the average number of shares for each 'channel type' (lifestyle, social media, tech etc.)
- Sort the channel types in decreasing order of average number of articles

In [29]:
```
# Compare the average number of shares for each 'channel type' (lifestyle, social
avg_shares = long_df.groupby('Channel')[' shares'].mean()
avg_shares
```

Out[29]:
```
Channel
 data_channel_is_bus              3063.018536
 data_channel_is_entertainment    2970.487034
 data_channel_is_lifestyle        3682.123392
 data_channel_is_socmed           3629.383125
 data_channel_is_tech             3072.283283
 data_channel_is_world            2287.734069
Name:  shares, dtype: float64
```

In [30]:
```
# Sort the channel types in decreasing order of average number of articles
avg_shares.sort_values(ascending=False)
```

Out[30]:
```
Channel
 data_channel_is_lifestyle        3682.123392
 data_channel_is_socmed           3629.383125
 data_channel_is_tech             3072.283283
 data_channel_is_bus              3063.018536
 data_channel_is_entertainment    2970.487034
 data_channel_is_world            2287.734069
Name:  shares, dtype: float64
```

Similarly, the columns weekday_is_monday, weekday_is_tuesday etc. are also in a wide format. You can try converting them into long as an exercise.

Additional Stuff to Read

Paper on tidy data by Hadley Wickham: http://vita.had.co.nz/papers/tidy-data.pdf (http://vita.had.co.nz/papers/tidy-data.pdf)