4. Data Assembly

4.1 Introduction

By now, you should be able to load data into `Pandas` and do some basic visualizations. This part of the book focuses on various data cleaning tasks. We begin with assembling a data set for analysis by combining various data sets together.

Concept Map

1. Prior knowledge

a. loading data

b. subsetting data

c. functions and class methods

Objectives

This chapter will cover:

1. Tidy data
2. Concatenating data
3. Merging data sets

4.2 Tidy Data

Hadley Wickham,[1] one of the more prominent members of the R community, talks about the idea of *tidy* data. In fact, he's written a paper about this concept in the *Journal of Statistical Software*.[2] Tidy data is a framework to structure data sets so they can be easily analyzed. It is mainly used as a goal one should aim for when cleaning data. Once you understand what tidy data is, that knowledge will make data collection much easier.

---

1. Hadley Wickham's homepage: http://hadley.nz

---

2. Tidy data paper: http://vita.had.co.nz/papers/tidy-data.pdf

So what is *tidy* data? Hadley Wickham's paper defines it as meeting the following criteria:

- Each row is an observation.
- Each column is a variable.
- Each type of observational unit forms a table.

4.2.1 Combining Data Sets

We begin with Hadley Wickham's last tidy data point: "Each type of observational unit forms a table." When data is tidy, you need to combine various tables together to answer a question. For example, there may be a separate table holding company information and another table holding stock prices. If we want to look at all the stock prices within the tech industry, we may first have to find all the tech companies from the company information table, and then combine that data with the stock price data to get the data we need for our question. The data may have been split up into separate tables to reduce the amount of redundant information (we don't need to store the company information with each stock price entry), but this arrangement means we as data analysts must combine the relevant data ourselves to answer our question.

At other times, a single data set may be split into multiple parts. For example, with time-series data, each date may be in a separate file. In another case, a file may have been split into parts to make the individual files smaller. You may also need to combine data from multiple sources to answer a question (e.g., combine latitudes and longitudes with zip codes). In both cases, you will need to combine data into a single dataframe for analysis.

4.3 Concatenation

One of the (conceptually) easier ways to combine data is with concatenation. Concatenation can be thought of appending a row or column to your data. This approach is possible if your data was split into parts or if you performed a calculation that you want to append to your existing data set.

Concatenation is accomplished by using the `concat` function from Pandas.

4.3.1 Adding Rows

Let's begin with some example data sets so you can see what is actually happening.

Click here to view code image

```
import pandas as pd


df1 = pd.read_csv('../data/concat_1.csv')
df2 = pd.read_csv('../data/concat_2.csv')
df3 = pd.read_csv('../data/concat_3.csv')


print(df1)
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
```

```
print(df2)
```

```
    A   B   C   D
0  a4  b4  c4  d4
1  a5  b5  c5  d5
```

```
2  a6  b6  c6  d6

3  a7  b7  c7  d7
```

```
print(df3)
```

```
     A    B    C    D

0   a8   b8   c8   d8

1   a9   b9   c9   d9

2  a10  b10  c10  d10

3  a11  b11  c11  d11
```

Stacking the dataframes on top of each other uses the `concat` function in `Pandas`. All of the dataframes to be concatenated are passed in a `list`.

Click here to view code image

```
row_concat = pd.concat([df1, df2, df3])

print(row_concat)
```

```
     A    B    C    D

0   a0   b0   c0   d0

1   a1   b1   c1   d1

2   a2   b2   c2   d2

3   a3   b3   c3   d3

0   a4   b4   c4   d4
```

```
1   a5   b5   c5   d5

2   a6   b6   c6   d6

3   a7   b7   c7   d7

0   a8   b8   c8   d8

1   a9   b9   c9   d9

2  a10  b10  c10  d10

3  a11  b11  c11  d11
```

As you can see, `concat` blindly stacks the dataframes together. If you look at the row names (i.e., the row indices), they are also simply a stacked version of the original row indices. If we apply the various subsetting methods from Table 2.3, the table will be subsetted as expected.

Click here to view code image

```
# subset the fourth row of the concatenated dataframe

print(row_concat.iloc[3,])
```

```
A    a3

B    b3

C    c3

D    d3

Name: 3, dtype: object
```

QUESTION

What happens when you use `loc` or `ix` to subset the new dataframe?

Section 2.2.1 showed the process for creating a `Series`. However, if we create a new series to append to a dataframe, it does not append correctly.

Click here to view code image

```
# create a new row of data
new_row_series = pd.Series(['n1', 'n2', 'n3', 'n4'])
print (new_row_series)
```

```
0    n1
1    n2
2    n3
3    n4
dtype: object
```

```
# attempt to add the new row to a dataframe
print(pd.concat([df1, new_row_series]))
```

```
     A    B    C    D    0
0   a0   b0   c0   d0  NaN
1   a1   b1   c1   d1  NaN
2   a2   b2   c2   d2  NaN
```

```
3   a3   b3   c3   d3  NaN

0  NaN  NaN  NaN  NaN   n1

1  NaN  NaN  NaN  NaN   n2

2  NaN  NaN  NaN  NaN   n3

3  NaN  NaN  NaN  NaN   n4
```

The first things you may notice are the `NaN` values. This is simply Python's way of representing a "missing value" (see Chapter 5, "Missing Data"). We were hoping to append our new values as a row, but that didn't happen. In fact, not only did our code not append the values as a row, but it also created a new column completely misaligned with everything else.

If we pause to think about what is happening here, we can see that the results actually make sense. First, if we look at the new indices that were added, we notice that they are very similar to the results we obtained when we concatenated dataframes earlier. The indices of the `new_row series` object are analogs to the row numbers of the dataframe. Also, since our series did not have a matching column, our `new_row` was added to a new column.

To fix this problem, we can turn our series into a dataframe. This data frame contains one row of data, and the column names are the ones the data will bind to.

Click here to view code image

```
                          # note the double brackets
new_row_df = pd.DataFrame([['n1', 'n2', 'n3', 'n4']],
                          columns=['A', 'B', 'C', 'D'])
print (new_row_df)
```

```
    A   B   C   D
0  n1  n2  n3  n4
```

```
print(pd.concat([df1, new_row_df]))
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
0  n1  n2  n3  n4
```

`concat` is a general function that can concatenate multiple things at once. If you just needed to append a single object to an existing dataframe, the `append` function can handle that task.

Using a `DataFrame`:

Click here to view code image

```
print(df1.append(df2))
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
```

```
3  a3  b3  c3  d3

0  a4  b4  c4  d4

1  a5  b5  c5  d5

2  a6  b6  c6  d6

3  a7  b7  c7  d7
```

Using a single-row `DataFrame` :

```
print(df1.append(new_row_df))
```

```
    A   B   C   D

0  a0  b0  c0  d0

1  a1  b1  c1  d1

2  a2  b2  c2  d2

3  a3  b3  c3  d3

0  n1  n2  n3  n4
```

Using a Python dictionary:

```
data_dict = {'A': 'n1',

             'B': 'n2',

             'C': 'n3',

             'D': 'n4'}



print(df1.append(data_dict, ignore_index=True))
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
4  n1  n2  n3  n4
```

4.3.1.1 Ignoring the Index

In the last example, when we added a `dict` to a dataframe, we had to use the `ignore_index` parameter. If we look closer, you can see that the row index was also incremented by 1, and did not repeat a previous index value.

If we simply want to concatenate or append data together, we can use the `ignore_index` parameter to reset the row index after the concatenation.

Click here to view code image

```
row_concat_i = pd.concat([df1, df2, df3], ignore_index=True)
print (row_concat_i)
```

```
      A    B    C    D
0    a0   b0   c0   d0
1    a1   b1   c1   d1
2    a2   b2   c2   d2
```

```
3    a3   b3   c3   d3

4    a4   b4   c4   d4

5    a5   b5   c5   d5

6    a6   b6   c6   d6

7    a7   b7   c7   d7

8    a8   b8   c8   d8

9    a9   b9   c9   d9

10  a10  b10  c10  d10

11  a11  b11  c11  d11
```

4.3.2 Adding Columns

Concatenating columns is very similar to concatenating rows. The main difference is the `axis` parameter in the `concat` function. The default value of `axis` is `0`, so it will concatenate data in a row-wise fashion. However, if we pass `axis=1` to the function, it will concatenate data in a column-wise manner.

Click here to view code image

```
col_concat = pd.concat([df1, df2, df3], axis=1)

print (col_concat)
```

```
    A   B   C   D   A   B   C   D    A    B    C    D

0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8

1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9

2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10
```

```
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11
```

If we try to subset data based on column names, we will get a similar result when we concatenated row-wise and subset by row index.

Click here to view code image

```
print(col_concat['A'])
```

```
    A   A    A
0  a0  a4   a8
1  a1  a5   a9
2  a2  a6  a10
3  a3  a7  a11
```

Adding a single column to a dataframe can be done directly without using any specific Pandas function. Simply pass a new column name the vector you want assigned to the new column.

Click here to view code image

```
col_concat['new_col_list'] = ['n1', 'n2', 'n3', 'n4']
print (col_concat)
```

```
    A   B   C   D   A   B   C   D    A    B    C    D new_col_list
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8           n1
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9           n2
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10           n3
```

```
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11           n4
```

```
col_concat['new_col_series'] = pd.Series(['n1', 'n2', 'n3', 'n4'])
print (col_concat)
```

```
    A   B   C   D   A   B   C   D    A   B     C    D new_col_series
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8            n1
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9            n2
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10            n3
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11            n4
```

Using the `concat` function still works, as long as you pass it a dataframe. This approach does require a bit more unnecessary code.

Finally, we can reset the column indices so we do not have duplicated column names.

Click here to view code image

```
print(pd.concat([df1, df2, df3], axis=1, ignore_index=True))
```

```
   0   1   2   3   4   5   6   7    8    9    10   11
0  a0  b0  c0  d0  a4  b4  c4  d4   a8   b8   c8   d8
1  a1  b1  c1  d1  a5  b5  c5  d5   a9   b9   c9   d9
2  a2  b2  c2  d2  a6  b6  c6  d6  a10  b10  c10  d10
3  a3  b3  c3  d3  a7  b7  c7  d7  a11  b11  c11  d11
```

4.3.3 Concatenation With Different Indices

The examples shown so far have assumed we are performing a simple row or column concatenation. They also assume that the new row(s) had the same column names or the column(s) had the same row indices.

This section addresses what happens when the row and column indices are not aligned.

4.3.3.1 Concatenate Rows With Different Columns

Let's modify our dataframes for the next few examples.

Click here to view code image

```
df1.columns = ['A', 'B', 'C', 'D']

df2.columns = ['E', 'F', 'G', 'H']

df3.columns = ['A', 'C', 'F', 'H']

print (df1)
```

```
    A   B   C   D

0  a0  b0  c0  d0

1  a1  b1  c1  d1

2  a2  b2  c2  d2

3  a3  b3  c3  d3
```

```
print (df2)
```

```
    E   F   G   H

0  a4  b4  c4  d4
```

```
1  a5  b5  c5  d5

2  a6  b6  c6  d6

3  a7  b7  c7  d7
```

```
print (df3)
```

```
     A    C    F    H

0   a8   b8   c8   d8

1   a9   b9   c9   d9

2  a10  b10  c10  d10

3  a11  b11  c11  d11
```

If we try to concatenate these dataframes as we did in Section 4.3.1, the dataframes now do much more than simply stack one on top of the other. The columns align themselves, and `NaN` fills in any missing areas.

Click here to view code image

```
row_concat = pd.concat([df1, df2, df3])

print (row_concat)
```

```
     A    B    C    D    E    F    G    H

0   a0   b0   c0   d0  NaN  NaN  NaN  NaN

1   a1   b1   c1   d1  NaN  NaN  NaN  NaN

2   a2   b2   c2   d2  NaN  NaN  NaN  NaN

3   a3   b3   c3   d3  NaN  NaN  NaN  NaN
```

```
0  NaN  NaN  NaN  NaN   a4   b4   c4   d4

1  NaN  NaN  NaN  NaN   a5   b5   c5   d5

2  NaN  NaN  NaN  NaN   a6   b6   c6   d6

3  NaN  NaN  NaN  NaN   a7   b7   c7   d7

0   a8  NaN   b8  NaN  NaN   c8  NaN   d8

1   a9  NaN   b9  NaN  NaN   c9  NaN   d9

2  a10  NaN  b10  NaN  NaN  c10  NaN  d10

3  a11  NaN  b11  NaN  NaN  c11  NaN  d11
```

One way to avoid the inclusion of `NaN` values is to keep only those columns that are shared in common by the list of objects to be concatenated. A parameter named `join` accomplishes this. By default, it has a value of `'outer'`, meaning it will keep all the columns. However, we can set `join='inner'` to keep only the columns that are shared among the data sets.

If we try to keep only the columns from all three dataframes, we will get an empty dataframe, since there are no columns in common.

Click here to view code image

```
print(pd.concat([df1, df2, df3], join='inner'))
```

```
Empty DataFrame

Columns: []

Index: [0, 1, 2, 3, 0, 1, 2, 3, 0, 1, 2, 3]
```

If we use the dataframes that have columns in common, only the columns that all of them share will be returned.

Click here to view code image

```
print(pd.concat([df1,df3], ignore_index=False, join='inner'))
```

```
     A    C
0   a0   c0
1   a1   c1
2   a2   c2
3   a3   c3
0   a8   b8
1   a9   b9
2  a10  b10
3  a11  b11
```

4.3.3.2 Concatenate Columns With Different Rows

Let's take our dataframes and modify them again so that they have different row indices. Here, we are building on the same dataframe modifications from Section 4.3.3.1.

Click here to view code image

```
df1.index = [0, 1, 2, 3]
df2.index = [4, 5, 6, 7]
df3.index = [0, 2, 5, 7]
```

```
print(df1)
```

```
    A   B   C   D
0  a0  b0  c0  d0
1  a1  b1  c1  d1
2  a2  b2  c2  d2
3  a3  b3  c3  d3
```

```
print(df2)
```

```
    E   F   G   H
4  a4  b4  c4  d4
5  a5  b5  c5  d5
6  a6  b6  c6  d6
7  a7  b7  c7  d7
```

```
print(df3)
```

```
     A    C    F    H
0   a8   b8   c8   d8
2   a9   b9   c9   d9
5  a10  b10  c10  d10
7  a11  b11  c11  d11
```

When we concatenate along `axis=1`, we get the same results from concatenating along `axis=0`. The new dataframes will be added in a column-wise fashion and matched against their respective row indices. Missing values indicators appear in the areas where the indices did not align.

Click here to view code image

```
col_concat = pd.concat([df1, df2, df3], axis=1)

print (col_concat)
```

```
     A    B    C    D    E    F    G    H    A    C    F    H
0   a0   b0   c0   d0  NaN  NaN  NaN  NaN   a8   b8   c8   d8
1   a1   b1   c1   d1  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
2   a2   b2   c2   d2  NaN  NaN  NaN  NaN   a9   b9   c9   d9
3   a3   b3   c3   d3  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN   a4   b4   c4   d4  NaN  NaN  NaN  NaN
5  NaN  NaN  NaN  NaN   a5   b5   c5   d5  a10  b10  c10  d10
6  NaN  NaN  NaN  NaN   a6   b6   c6   d6  NaN  NaN  NaN  NaN
7  NaN  NaN  NaN  NaN   a7   b7   c7   d7  a11  b11  c11  d11
```

Just as we did when we concatenated in a row-wise manner, we can choose to keep the results only when there are matching indices by using `join='inner'`.

Click here to view code image

```
print(pd.concat([df1, df3], axis=1, join='inner'))
```

```
    A   B   C   D   A   C   F   H
0  a0  b0  c0  d0  a8  b8  c8  d8
2  a2  b2  c2  d2  a9  b9  c9  d9
```

4.4 Merging Multiple Data Sets

The previous section alluded to a few database concepts. The `join='inner'` and the default `join='outer'` parameters come from working with databases when we want to merge tables.

Instead of simply having a row or column index that you want to use to concatenate values, sometimes you may have two or more dataframes that you want to combine based on common data values. This task is known in the database world as performing a "join."

Pandas has a `pd.join` command that uses `pd.merge` under the hood. `join` will merge dataframe objects based on an index, but the `merge` command is much more explicit and flexible. If you are planning to merge dataframes by the row index, for example, you might want to look into the `join` function.[3]

3. Pandas `DataFrame join` function: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.join.html

We will be using sets of survey data in this series of examples.

Click here to view code image

```
person = pd.read_csv('../data/survey_person.csv')
site = pd.read_csv('../data/survey_site.csv')
survey = pd.read_csv('../data/survey_survey.csv')
visited = pd.read_csv('../data/survey_visited.csv')
```

```
print(person)
```

```
      ident   personal    family
0      dyer    William      Dyer
1        pb      Frank   Pabodie
2      lake   Anderson      Lake
3       roe  Valentina   Roerich
4  danforth      Frank  Danforth
```

```
print(site)
```

```
    name    lat    long
0   DR-1 -49.85 -128.57
1   DR-3 -47.15 -126.72
2  MSK-4 -48.87 -123.40
```

```
print(visited)
```

```
   ident   site       dated
0    619   DR-1  1927-02-08
1    622   DR-1  1927-02-10
2    734   DR-3  1939-01-07
```

```
3    735   DR-3  1930-01-12
4    751   DR-3  1930-02-26
5    752   DR-3         NaN
6    837  MSK-4  1932-01-14
7    844   DR-1  1932-03-22
```

```
print (survey)
```

```
    taken person quant  reading
0     619   dyer   rad     9.82
1     619   dyer   sal     0.13
2     622   dyer   rad     7.80
3     622   dyer   sal     0.09
4     734     pb   rad     8.41
5     734   lake   sal     0.05
6     734     pb  temp   -21.50
7     735     pb   rad     7.22
8     735    NaN   sal     0.06
9     735    NaN  temp   -26.00
10    751     pb   rad     4.35
11    751     pb  temp   -18.50
12    751   lake   sal     0.10
```

```
13    752   lake   rad    2.19

14    752   lake   sal    0.09

15    752   lake  temp  -16.00

16    752    roe   sal   41.60

17    837   lake   rad    1.46

18    837   lake   sal    0.21

19    837    roe   sal   22.50

20    844    roe   rad   11.25
```

Currently, our data is split into multiple parts, where each part is an observational unit. If we wanted to look at the dates at each site along with the latitude and longitude information for that site, we would have to combine (and merge) multiple dataframes. We can do this with the `merge` function in Pandas. `merge` is actually a `DataFrame` method.

When we call this method, the dataframe that is called will be referred to the one on the `'left'`. Within the `merge` function, the first parameter is the `'right'` dataframe. The next parameter is `how` the final merged result looks. Table 4.1 provides more details. Next, we set the `on` parameter. This specifies which columns to match on. If the left and right columns do not have the same name, we can use the `left_on` and `right_on` parameters instead.

Table 4.1 How the Pandas how Parameter Relates to SQL

Pandas	SQL	Description
left	left outer	Keep all the keys from the left
right	right outer	Keep all the keys from the right
outer	full outer	Keep all the keys from both left and right
inner	inner	Keep only the keys that exist in both left and right

4.4.1 One-to-One Merge

In the simplest type of merge, we have two dataframes where we want to join one column to another column, and where the columns we want to join do not contain any duplicate values.

For this example, we will modify the `visited` dataframe so there are no duplicated `site` values.

Click here to view code image

```
visited_subset = visited.loc[[0, 2, 6], ]
```

We can perform our one-to-one merge as follows:

Click here to view code image

```
# the default value for 'how' is 'inner'

# so it doesn't need to be specified

o2o_merge = site.merge(visited_subset,
```

```
                                left_on='name', right_on='site')

print(o2o_merge)
```

```
   name    lat    long  ident   site       dated
0  DR-1 -49.85 -128.57    619   DR-1  1927-02-08
1  DR-3 -47.15 -126.72    734   DR-3  1939-01-07
2 MSK-4 -48.87 -123.40    837  MSK-4  1932-01-14
```

As you can see, we have now created a new dataframe from two separate dataframes where the rows were matched based on a particular set of columns. In SQL-speak, the columns used to match are called "keys."

4.4.2 Many-to-One Merge

If we choose to do the same merge, but this time without using the subsetted `visited` dataframe, we would perform a many-to-one merge. In this kind of merge, one of the dataframes has key values that repeat. The dataframes that contains the single observations will then be duplicated in the merge.

Click here to view code image

```
m2o_merge = site.merge(visited, left_on='name', right_on='site')

print(m2o_merge)
```

```
    name    lat    long  ident   site       dated
0   DR-1 -49.85 -128.57    619   DR-1  1927-02-08
1   DR-1 -49.85 -128.57    622   DR-1  1927-02-10
2   DR-1 -49.85 -128.57    844   DR-1  1932-03-22
```

```
3   DR-3 -47.15 -126.72    734   DR-3  1939-01-07

4   DR-3 -47.15 -126.72    735   DR-3  1930-01-12

5   DR-3 -47.15 -126.72    751   DR-3  1930-02-26

6   DR-3 -47.15 -126.72    752   DR-3         NaN

7  MSK-4 -48.87 -123.40    837  MSK-4  1932-01-14
```

As you can see, the `site` information (`name`, `lat`, and `long`) were duplicated and matched to the `visited` data.

4.4.3 Many-to-Many Merge

Lastly, there will be times when we want to perform a match based on multiple columns. As an example, suppose we have two dataframes that come from `person` merged with `survey`, and another dataframe that comes from `visited` merged with `survey`.

Click here to view code image

```
ps = person.merge(survey, left_on='ident', right_on='person')

vs = visited.merge(survey, left_on='ident', right_on='taken')

print (ps)
```

```
    ident   personal   family  taken person quant reading

0   dyer    William     Dyer    619   dyer   rad    9.82

1   dyer    William     Dyer    619   dyer   sal    0.13

2   dyer    William     Dyer    622   dyer   rad    7.80

3   dyer    William     Dyer    622   dyer   sal    0.09

4     pb      Frank  Pabodie    734     pb   rad    8.41
```

```
5     pb      Frank  Pabodie    734     pb  temp  -21.50
6     pb      Frank  Pabodie    735     pb   rad    7.22
7     pb      Frank  Pabodie    751     pb   rad    4.35
8     pb      Frank  Pabodie    751     pb  temp  -18.50
9   lake   Anderson     Lake    734   lake   sal    0.05
10  lake   Anderson     Lake    751   lake   sal    0.10
11  lake   Anderson     Lake    752   lake   rad    2.19
12  lake   Anderson     Lake    752   lake   sal    0.09
13  lake   Anderson     Lake    752   lake  temp  -16.00
14  lake   Anderson     Lake    837   lake   rad    1.46
15  lake   Anderson     Lake    837   lake   sal    0.21
16   roe  Valentina  Roerich    752    roe   sal   41.60
17   roe  Valentina  Roerich    837    roe   sal   22.50
18   roe  Valentina  Roerich    844    roe   rad   11.25
```

```
print(vs)
```

```
    ident   site       dated  taken person quant  reading
0     619   DR-1  1927-02-08    619   dyer   rad     9.82
1     619   DR-1  1927-02-08    619   dyer   sal     0.13
2     622   DR-1  1927-02-10    622   dyer   rad     7.80
3     622   DR-1  1927-02-10    622   dyer   sal     0.09
```

```
4     734   DR-3  1939-01-07    734     pb   rad     8.41
5     734   DR-3  1939-01-07    734   lake   sal     0.05
6     734   DR-3  1939-01-07    734     pb  temp   -21.50
7     735   DR-3  1930-01-12    735     pb   rad     7.22
8     735   DR-3  1930-01-12    735    NaN   sal     0.06
9     735   DR-3  1930-01-12    735    NaN  temp   -26.00
10    751   DR-3  1930-02-26    751     pb   rad     4.35
11    751   DR-3  1930-02-26    751     pb  temp   -18.50
12    751   DR-3  1930-02-26    751   lake   sal     0.10
13    752   DR-3         NaN    752   lake   rad     2.19
14    752   DR-3         NaN    752   lake   sal     0.09
15    752   DR-3         NaN    752   lake  temp   -16.00
16    752   DR-3         NaN    752    roe   sal    41.60
17    837  MSK-4  1932-01-14    837   lake   rad     1.46
18    837  MSK-4  1932-01-14    837   lake   sal     0.21
19    837  MSK-4  1932-01-14    837    roe   sal    22.50
20    844   DR-1  1932-03-22    844    roe   rad    11.25
```

We can perform a many-to-many merge by passing the multiple columns to match on in a Python list.

Click here to view code image

```
ps_vs = ps.merge(vs,
                 left_on=['ident', 'taken', 'quant', 'reading'],
                 right_on=['person', 'ident', 'quant', 'reading'])
```

Let's look at just the first row of data.

```
print(ps_vs.loc[0, ])
```

```
ident_x           dyer
personal       William
family            Dyer
taken_x             619
person_x          dyer
quant              rad
reading           9.82
ident_y             619
site              DR-1
dated       1927-02-08
taken_y             619
person_y          dyer
Name: 0, dtype: object
```


Pandas will automatically add a suffix to a column name if there are collisions in the name. In the output, the `_x` refers to values from the left

dataframe, and the `_y` suffix comes from values in the right dataframe.

4.5 Conclusion

Sometimes you may need to combine various parts or data or multiple data sets depending on the question you are trying to answer. Keep in mind, however, that the data you need for analysis does not necessarily equate to the best shape of data for storage.

The survey data used in the last example came in four separate parts that needed to be merged together. After we merged the tables a lot of redundant information appeared across the rows. From a data storage and data entry point of view, each of these duplications can lead to errors and data inconsistency. This is what Hadley meant by saying that in tidy data, “each type of observational unit forms a table.”