Missing Values

We've seen a preview of how Pandas handles missing values using the None type and NumPy NaN values. Missing values are pretty common in data cleaning activities. And, missing values can be there for any number of reasons, and I just want to touch on a few here.

For instance, if you are running a survey and a respondant didn't answer a question the missing value is actually an omission. This kind of missing data is called **Missing at Random** if there are other variables that might be used to predict the variable which is missing. In my work when I delivery surveys I often find that missing data, say the interest in being involved in a follow up study, often has some correlation with another data field, like gender or ethnicity. If there is no relationship to other variables, then we call this data **Missing Completely at Random (MCAR)**.

These are just two examples of missing data, and there are many more. For instance, data might be missing because it wasn't collected, either by the process responsible for collecting that data, such as a researcher, or because it wouldn't make sense if it were collected. This last example is extremely common when you start joining DataFrames together from multiple sources, such as joining a list of people at a university with a list of offices in the university (students generally don't have offices).

Let's look at some ways of handling missing data in pandas.

In [1]:

```
# Lets import pandas
import pandas as pd
```

Pandas at Detecting NaN values

Pandas is pretty good at detecting missing values directly from underlying data formats, like CSV files. Although most missing valuse are often formatted as NaN, NULL, None, or N/A, **sometimes missing values are not labeled so clearly**. For example, I've worked with social scientists who regularly used the value of 99 in binary categories to indicate a missing value. The pandas `read_csv()` function has a parameter called `na_values` to let us specify the form of missing values. It allows scalar, string, list, or dictionaries to be used.

In [2]:

```
# Let's load a piece of data from a file called log.csv
df = pd.read_csv('datasets/class_grades.csv')
df.head(10)
```

Out[2]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	5	57.14	34.09	64.38	51.48	52.50
1	8	95.05	105.49	67.50	99.07	68.33
2	8	83.70	83.17	NaN	63.15	48.89
3	7	NaN	NaN	49.38	105.93	80.56
4	8	91.32	93.64	95.00	107.41	73.89
5	7	95.00	92.58	93.12	97.78	68.06
6	8	95.05	102.99	56.25	99.07	50.00
7	7	72.85	86.85	60.00	NaN	56.11
8	8	84.26	93.10	47.50	18.52	50.83
9	7	90.10	97.55	51.25	88.89	63.61

DataFrame Boolean Masks: Function `isnull( )`

In [3]:

```
# We can actually use the function .isnull() to create a boolean mask of the who
le dataframe. This effectively
# broadcasts the isnull() function to every cell of data.
mask=df.isnull()
mask.head(10)
```

Out[3]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	True	False	False
3	False	True	True	False	False	False
4	False	False	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False
7	False	False	False	False	True	False
8	False	False	False	False	False	False
9	False	False	False	False	False	False

(Temporarily) Deleting Rows using `.dropna()`

Note how the rows indexed with 2, 3, 7, and 11 are now gone.

This can be useful for processing rows based on certain columns of data. Another useful operation is to be able to drop all of those rows which have any missing data, which can be done with the dropna() function.

In [4]:

```
df.dropna().head(10)
```

Out[4]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	5	57.14	34.09	64.38	51.48	52.50
1	8	95.05	105.49	67.50	99.07	68.33
4	8	91.32	93.64	95.00	107.41	73.89
5	7	95.00	92.58	93.12	97.78	68.06
6	8	95.05	102.99	56.25	99.07	50.00
8	8	84.26	93.10	47.50	18.52	50.83
9	7	90.10	97.55	51.25	88.89	63.61
10	7	80.44	90.20	75.00	91.48	39.72
12	8	97.16	103.71	72.50	93.52	63.33
13	7	91.28	83.53	81.25	99.81	92.22

Filling NaN values with a specific value using `fillna()`

In [5]:

```
One of the handy functions that Pandas has for
# working with missing values is the filling function, fillna(). This function t
akes a number or parameters.
# You could pass in a single value which is called a scalar value to change all
 of the missing data to one
# value. This isn't really applicable in this case, but it's a pretty common use
case.

# So, if we wanted to fill all missing values with 0, we would use fillna
df.fillna(0, inplace=True)
df.head(10)
```

Out[5]:

	Prefix	Assignment	Tutorial	Midterm	TakeHome	Final
0	5	57.14	34.09	64.38	51.48	52.50
1	8	95.05	105.49	67.50	99.07	68.33
2	8	83.70	83.17	0.00	63.15	48.89
3	7	0.00	0.00	49.38	105.93	80.56
4	8	91.32	93.64	95.00	107.41	73.89
5	7	95.00	92.58	93.12	97.78	68.06
6	8	95.05	102.99	56.25	99.07	50.00
7	7	72.85	86.85	60.00	0.00	56.11
8	8	84.26	93.10	47.50	18.52	50.83
9	7	90.10	97.55	51.25	88.89	63.61

Note that the inplace attribute causes pandas to **fill the values inline** and does not return a copy of the dataframe, but instead modifies the dataframe you have.

We can also use the **na_filter option** to turn off white space filtering, if white space is an actual value of interest. But in practice, this is pretty rare. In data without any NAs, passing `na_filter=False` , can improve the performance of reading a large file.

Case Study: Logs of Online Learning Systems

In these systems it's common for the player for have a heartbeat functionality where playback statistics are sent to the server every so often, maybe every 30 seconds. These heartbeats can get big as they can carry the whole state of the playback system such as where the video play head is at, where the video size is, which video is being rendered to the screen, how loud the volume is.

In [7]:

```
# In addition to rules controlling how missing values might be loaded, it's some
times useful to consider
# missing values as actually having information. I'll give an example from my ow
n research.  I often deal with
# logs from online learning systems. I've looked at video use in lecture capture
systems.

# If we load the data file log.csv, we can see an example of what this might loo
k like.
df = pd.read_csv("datasets/log.csv")
df.head(20)
```

Out[7]:

	time	user	video	playback position	paused	volume
0	1469974424	cheryl	intro.html	5	False	10.0
1	1469974454	cheryl	intro.html	6	NaN	NaN
2	1469974544	cheryl	intro.html	9	NaN	NaN
3	1469974574	cheryl	intro.html	10	NaN	NaN
4	1469977514	bob	intro.html	1	NaN	NaN
5	1469977544	bob	intro.html	1	NaN	NaN
6	1469977574	bob	intro.html	1	NaN	NaN
7	1469977604	bob	intro.html	1	NaN	NaN
8	1469974604	cheryl	intro.html	11	NaN	NaN
9	1469974694	cheryl	intro.html	14	NaN	NaN
10	1469974724	cheryl	intro.html	15	NaN	NaN
11	1469974454	sue	advanced.html	24	NaN	NaN
12	1469974524	sue	advanced.html	25	NaN	NaN
13	1469974424	sue	advanced.html	23	False	10.0
14	1469974554	sue	advanced.html	26	NaN	NaN
15	1469974624	sue	advanced.html	27	NaN	NaN
16	1469974654	sue	advanced.html	28	NaN	5.0
17	1469974724	sue	advanced.html	29	NaN	NaN
18	1469974484	cheryl	intro.html	7	NaN	NaN
19	1469974514	cheryl	intro.html	8	NaN	NaN

In this data the first column is a timestamp in the Unix epoch format. The next column is the user name followed by a web page they're visiting and the video that they're playing. Each row of the DataFrame has a playback position. And we can see that as the playback position increases by one, the time stamp increases by about 30 seconds.

Except for user Bob. It turns out that Bob has paused his playback so as time increases the playback position doesn't change. Note too how difficult it is for us to try and derive this knowledge from the data, because it's not sorted by time stamp as one might expect. This is actually not uncommon on systems which have a high degree of parallelism. There are a lot of missing values in the paused and volume columns. It's not efficient to send this information across the network if it hasn't changed. So this articular system just inserts null values into the database if there's no changes.

Forward Filling `ffill` & Backward Filling `bfill`

Next up is the method parameter(). The two common fill values are ffill and bfill. ffill is for forward filling and it updates an na value for a particular cell with the value from the previous row. bfill is backward filling, which is the opposite of ffill. It fills the missing values with the next valid value. It's important to note that your data needs to be sorted in order for this to have the effect you might want. Data which comes from traditional database management systems usually has no order guarantee, just like this data. So be careful

Setting & Resetting indexes

In [9]:

```
# In Pandas we can sort either by index or by values. Here we'll just promote th
e time stamp to an index then
# sort on the index.
df = df.set_index('time')
df = df.sort_index()
df.head(20)
```

Out[9]:

	user	video	playback position	paused	volume
time					
1469974424	cheryl	intro.html	5	False	10.0
1469974424	sue	advanced.html	23	False	10.0
1469974454	cheryl	intro.html	6	NaN	NaN
1469974454	sue	advanced.html	24	NaN	NaN
1469974484	cheryl	intro.html	7	NaN	NaN
1469974514	cheryl	intro.html	8	NaN	NaN
1469974524	sue	advanced.html	25	NaN	NaN
1469974544	cheryl	intro.html	9	NaN	NaN
1469974554	sue	advanced.html	26	NaN	NaN
1469974574	cheryl	intro.html	10	NaN	NaN
1469974604	cheryl	intro.html	11	NaN	NaN
1469974624	sue	advanced.html	27	NaN	NaN
1469974634	cheryl	intro.html	12	NaN	NaN
1469974654	sue	advanced.html	28	NaN	5.0
1469974664	cheryl	intro.html	13	NaN	NaN
1469974694	cheryl	intro.html	14	NaN	NaN
1469974724	cheryl	intro.html	15	NaN	NaN
1469974724	sue	advanced.html	29	NaN	NaN
1469974754	sue	advanced.html	30	NaN	NaN
1469974824	sue	advanced.html	31	NaN	NaN

In [10]:

```
# If we look closely at the output though we'll notice that the index
# isn't really unique. Two users seem to be able to use the system at the same
# time. Again, a very common case. Let's reset the index, and use some
# multi-level indexing on time AND user together instead,
# promote the user name to a second level of the index to deal with that issue.

df = df.reset_index()
df = df.set_index(['time', 'user'])
df
```

Out[10]:

time	user	video	playback position	paused	volume
1469974424	**cheryl**	intro.html	5	False	10.0
	sue	advanced.html	23	False	10.0
1469974454	**cheryl**	intro.html	6	NaN	NaN
	sue	advanced.html	24	NaN	NaN
1469974484	**cheryl**	intro.html	7	NaN	NaN
1469974514	**cheryl**	intro.html	8	NaN	NaN
1469974524	**sue**	advanced.html	25	NaN	NaN
1469974544	**cheryl**	intro.html	9	NaN	NaN
1469974554	**sue**	advanced.html	26	NaN	NaN
1469974574	**cheryl**	intro.html	10	NaN	NaN
1469974604	**cheryl**	intro.html	11	NaN	NaN
1469974624	**sue**	advanced.html	27	NaN	NaN
1469974634	**cheryl**	intro.html	12	NaN	NaN
1469974654	**sue**	advanced.html	28	NaN	5.0
1469974664	**cheryl**	intro.html	13	NaN	NaN
1469974694	**cheryl**	intro.html	14	NaN	NaN
1469974724	**cheryl**	intro.html	15	NaN	NaN
	sue	advanced.html	29	NaN	NaN
1469974754	**sue**	advanced.html	30	NaN	NaN
1469974824	**sue**	advanced.html	31	NaN	NaN
1469974854	**sue**	advanced.html	32	NaN	NaN
1469974924	**sue**	advanced.html	33	NaN	NaN
1469977424	**bob**	intro.html	1	True	10.0
1469977454	**bob**	intro.html	1	NaN	NaN
1469977484	**bob**	intro.html	1	NaN	NaN
1469977514	**bob**	intro.html	1	NaN	NaN
1469977544	**bob**	intro.html	1	NaN	NaN
1469977574	**bob**	intro.html	1	NaN	NaN
1469977604	**bob**	intro.html	1	NaN	NaN
1469977634	**bob**	intro.html	1	NaN	NaN
1469977664	**bob**	intro.html	1	NaN	NaN
1469977694	**bob**	intro.html	1	NaN	NaN
1469977724	**bob**	intro.html	1	NaN	NaN

Applying `ffill`

In [11]:

```
# Now that we have the data indexed and sorted appropriately, we can fill the mi
ssing datas using ffill. It's
# good to remember when dealing with missing values so you can deal with individ
ual columns or sets of columns
# by projecting them. So you don't have to fix all missing values in one comman
d.

df = df.fillna(method='ffill')
df.head(10)
```

Out[11]:

		video	playback position	paused	volume
time	**user**				
1469974424	**cheryl**	intro.html	5	False	10.0
	sue	advanced.html	23	False	10.0
1469974454	**cheryl**	intro.html	6	False	10.0
	sue	advanced.html	24	False	10.0
1469974484	**cheryl**	intro.html	7	False	10.0
1469974514	**cheryl**	intro.html	8	False	10.0
1469974524	**sue**	advanced.html	25	False	10.0
1469974544	**cheryl**	intro.html	9	False	10.0
1469974554	**sue**	advanced.html	26	False	10.0
1469974574	**cheryl**	intro.html	10	False	10.0

Customised Fill-In with `replace()`

In [12]:

```
# We can also do customized fill-in to replace values with the replace() functio
n. It allows replacement from
# several approaches: value-to-value, list, dictionary, regex Let's generate a s
imple example
df = pd.DataFrame({'A': [1, 1, 2, 3, 4],
                   'B': [3, 6, 3, 8, 9],
                   'C': ['a', 'b', 'c', 'd', 'e']})
df
```

Out[12]:

	A	B	C
0	1	3	a
1	1	6	b
2	2	3	c
3	3	8	d
4	4	9	e

In [13]:

```
# We can replace 1's with 100, let's try the value-to-value approach
df.replace(1, 100)
```

Out[13]:

	A	B	C
0	100	3	a
1	100	6	b
2	2	3	c
3	3	8	d
4	4	9	e

In [14]:

```
# How about changing two values? Let's try the list approach For example, we wan
t to change 1's to 100 and 3's
# to 300
df.replace([1, 3], [100, 300])
```

Out[14]:

	A	B	C
0	100	300	a
1	100	6	b
2	2	300	c
3	300	8	d
4	4	9	e

Using RegEx To Aid Customised Fill-Ins

In [15]:

```
# What's really cool about pandas replacement is that it supports regex too!
# Let's look at our data from the dataset logs again
df = pd.read_csv("datasets/log.csv")
df.head(20)
```

Out[15]:

	time	user	video	playback position	paused	volume
0	1469974424	cheryl	intro.html	5	False	10.0
1	1469974454	cheryl	intro.html	6	NaN	NaN
2	1469974544	cheryl	intro.html	9	NaN	NaN
3	1469974574	cheryl	intro.html	10	NaN	NaN
4	1469977514	bob	intro.html	1	NaN	NaN
5	1469977544	bob	intro.html	1	NaN	NaN
6	1469977574	bob	intro.html	1	NaN	NaN
7	1469977604	bob	intro.html	1	NaN	NaN
8	1469974604	cheryl	intro.html	11	NaN	NaN
9	1469974694	cheryl	intro.html	14	NaN	NaN
10	1469974724	cheryl	intro.html	15	NaN	NaN
11	1469974454	sue	advanced.html	24	NaN	NaN
12	1469974524	sue	advanced.html	25	NaN	NaN
13	1469974424	sue	advanced.html	23	False	10.0
14	1469974554	sue	advanced.html	26	NaN	NaN
15	1469974624	sue	advanced.html	27	NaN	NaN
16	1469974654	sue	advanced.html	28	NaN	5.0
17	1469974724	sue	advanced.html	29	NaN	NaN
18	1469974484	cheryl	intro.html	7	NaN	NaN
19	1469974514	cheryl	intro.html	8	NaN	NaN

To replace using a regex we make the first parameter to replace the regex pattern we want to match, the second parameter the value we want to emit upon match, and then we pass in a third parameter "regex=True".

Take a moment to pause this video and think about this problem: imagine we want to detect all html pages in the "video" column, lets say that just means they end with ".html", and we want to overwrite that with the keyword "webpage". How could we accomplish this?

In [19]:

```
pattern = "[^\s]*[.]html$"
df.replace(to_replace = pattern,value ="webpage", regex=True)
```

Out[19]:

	time	user	video	playback position	paused	volume
0	1469974424	cheryl	webpage	5	False	10.0
1	1469974454	cheryl	webpage	6	NaN	NaN
2	1469974544	cheryl	webpage	9	NaN	NaN
3	1469974574	cheryl	webpage	10	NaN	NaN
4	1469977514	bob	webpage	1	NaN	NaN
5	1469977544	bob	webpage	1	NaN	NaN
6	1469977574	bob	webpage	1	NaN	NaN
7	1469977604	bob	webpage	1	NaN	NaN
8	1469974604	cheryl	webpage	11	NaN	NaN
9	1469974694	cheryl	webpage	14	NaN	NaN
10	1469974724	cheryl	webpage	15	NaN	NaN
11	1469974454	sue	webpage	24	NaN	NaN
12	1469974524	sue	webpage	25	NaN	NaN
13	1469974424	sue	webpage	23	False	10.0
14	1469974554	sue	webpage	26	NaN	NaN
15	1469974624	sue	webpage	27	NaN	NaN
16	1469974654	sue	webpage	28	NaN	5.0
17	1469974724	sue	webpage	29	NaN	NaN
18	1469974484	cheryl	webpage	7	NaN	NaN
19	1469974514	cheryl	webpage	8	NaN	NaN
20	1469974754	sue	webpage	30	NaN	NaN
21	1469974824	sue	webpage	31	NaN	NaN
22	1469974854	sue	webpage	32	NaN	NaN
23	1469974924	sue	webpage	33	NaN	NaN
24	1469977424	bob	webpage	1	True	10.0
25	1469977454	bob	webpage	1	NaN	NaN
26	1469977484	bob	webpage	1	NaN	NaN
27	1469977634	bob	webpage	1	NaN	NaN
28	1469977664	bob	webpage	1	NaN	NaN
29	1469974634	cheryl	webpage	12	NaN	NaN
30	1469974664	cheryl	webpage	13	NaN	NaN
31	1469977694	bob	webpage	1	NaN	NaN
32	1469977724	bob	webpage	1	NaN	NaN

One last note on missing values. When you use statistical functions on DataFrames, these functions typically ignore missing values. For instance if you try and calculate the mean value of a DataFrame, the underlying NumPy function will ignore missing values. This is usually what you want but you should be aware that values are being excluded. Why you have missing values really matters depending upon the problem you are trying to solve. It might be unreasonable to infer missing values, for instance, if the data shouldn't exist in the first place.