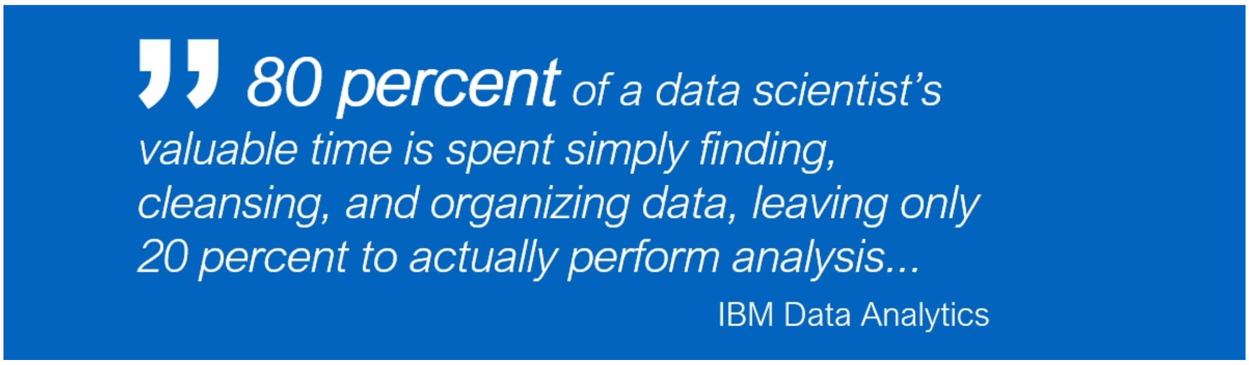
Data Cleaning with Python and Pandas: Detecting Missing Values

May 23, 2018

Data cleaning can be a tedious task.
It's the start of a new project and you're excited to apply some machine learning models.

You take a look at the **data** and quickly realize it's an **absolute mess**.

According to IBM Data Analytics you can expect to spend up to **80% of your time cleaning data**.



80 percent of a data scientist's valuable time is spent simply finding, cleansing, and organizing data, leaving only 20 percent to actually perform analysis...

IBM Data Analytics

In this post we'll walk through a number of different **data cleaning** tasks using Python's Pandas library. Specifically, we'll focus on probably the biggest data cleaning task, **missing values**.

After reading this post you'll be able to **more quickly clean data**. We all want to **spend less time cleaning data**, and more time exploring and modeling.

Click here to get the FREE Data Cleaning Cheat Sheet

Sources of Missing Values

Before we dive into code, it's important to understand the sources of missing data. Here's some typical reasons why data is missing:

- User forgot to fill in a field.
- Data was lost while transferring manually from a legacy database.
- There was a programming error.
- Users chose not to fill out a field tied to their beliefs about how the results would be used or interpreted.

As you can see, some of these sources are just simple random mistakes. Other times, there can be a deeper reason why data is missing.

It's important to understand these different types of missing data from a statistics point of view. The type of missing data will influence how you deal with filling in the missing values.

Today we'll learn how to detect missing values, and do some basic imputation. For a detailed statistical approach for dealing with missing data, check out these awesome slides from data scientist Matt Brems.

Keep in mind, imputing with a median or mean value is usually a bad idea, so be sure to check out Matt's slides for the correct approach.

Getting Started

Before you start cleaning a data set, it's a good idea to just get a general feel for the data. After that, you can put together a plan to clean the data.

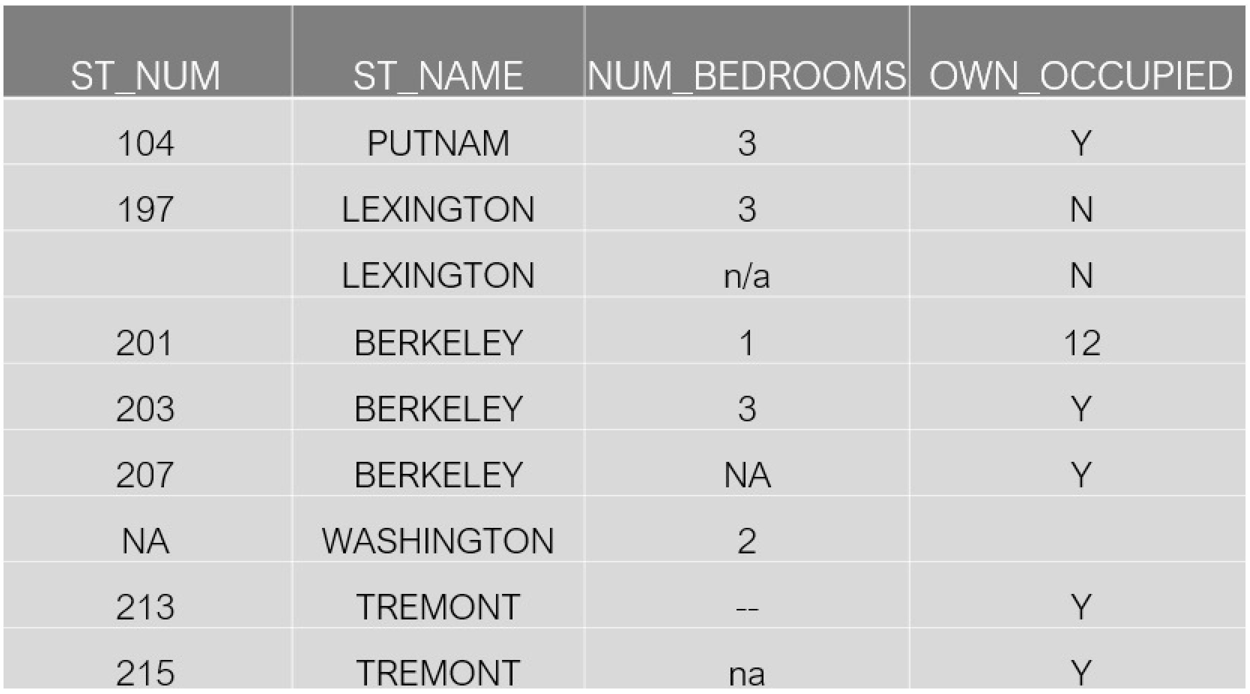
I like to start by asking the following questions:

- What are the features?
- What are the expected types (int, float, string, boolean)?
- Is there obvious missing data (values that Pandas can detect)?
- Is there other types of missing data that's not so obvious (can't easily detect with Pandas)?

To show you what I mean, let's start working through the example.

The data we're going to work with is a very small real estate dataset. Head on over to our github page to grab a copy of the csv file so that you can code along.

Here's a quick look at the data:



ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

This is a much smaller dataset than what you'll typically work with. Even though it's a small dataset, it highlights a lot of real-world situations that you will encounter on projects.

A good way to get a quick feel for the data is to take a look at the first few rows. Here's how you would do that in Pandas:

```
# Importing libraries
import pandas as pd
import numpy as np

# Read csv file into a pandas dataframe
df = pd.read_csv("property data.csv")

# Take a look at the first few rows
print df.head()
```
Copy

```
Out:
   ST_NUM    ST_NAME OWN_OCCUPIED  NUM_BEDROOMS
0   104.0     PUTNAM            Y           3.0
1   197.0  LEXINGTON            N           3.0
2     NaN  LEXINGTON            N           3.0
3   201.0   BERKELEY          NaN           1.0
4   203.0   BERKELEY            Y           3.0
```
Copy

I know that I said we'll be working with Pandas, but you can see that I also imported Numpy. We'll use this a little bit later on to rename some missing values, so we might as well import it now.

After importing the libraries we read the csv file into a Pandas dataframe. You can think of the dataframe as a spreadsheet.

With the `.head()` method, we can easily see the first few rows.

Now I can answer my original question, **what are my features?** It's pretty easy to infer the following features from the column names:

- `ST_NUM` : Street number
- `ST_NAME` : Street name
- `OWN_OCCUPIED` : Is the residence owner occupied
- `NUM_BEDROOMS` : Number of bedrooms

We can also answer, **what are the expected types?**

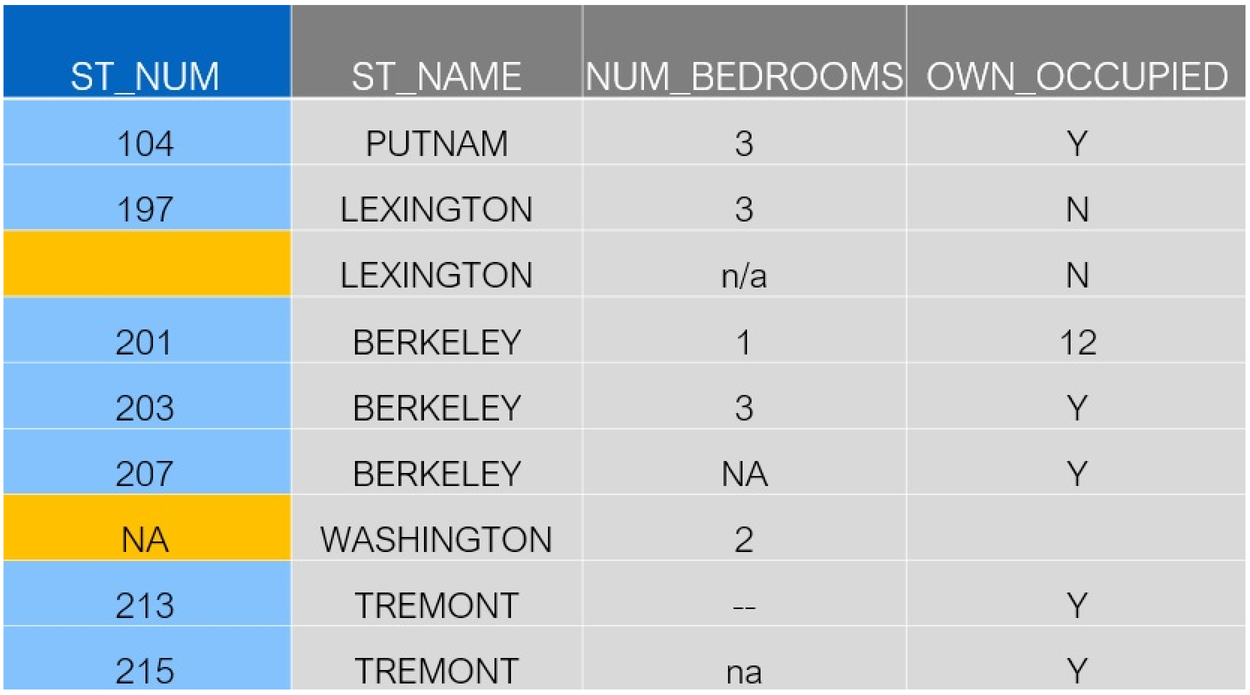
- `ST_NUM` : float or int… some sort of numeric type
- `ST_NAME` : string
- `OWN_OCCUPIED` : string… Y ("Yes") or N ("No")
- `NUM_BEDROOMS` : float or int, a numeric type

To answer the next two questions, we'll need to start getting more in-depth width Pandas. Let's start looking at examples of how to detect missing values

Standard Missing Values

So what do I mean by "standard missing values"? These are missing values that Pandas can detect.

Going back to our original data set, let's take a look at the "Street Number" column.



ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

In the third column there's an empty cell. In the seventh row there's an "NA" value.

Clearly these are both missing values. Let's see how Pandas deals with these.

```
# Looking at the ST_NUM column
print df['ST_NUM']
print df['ST_NUM'].isnull()
```
Copy

```
# Looking at the ST_NUM column
Out:
0    104.0
1    197.0
2      NaN
3    201.0
4    203.0
5    207.0
6      NaN
7    213.0
8    215.0

Out:
0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
8    False
```
Copy

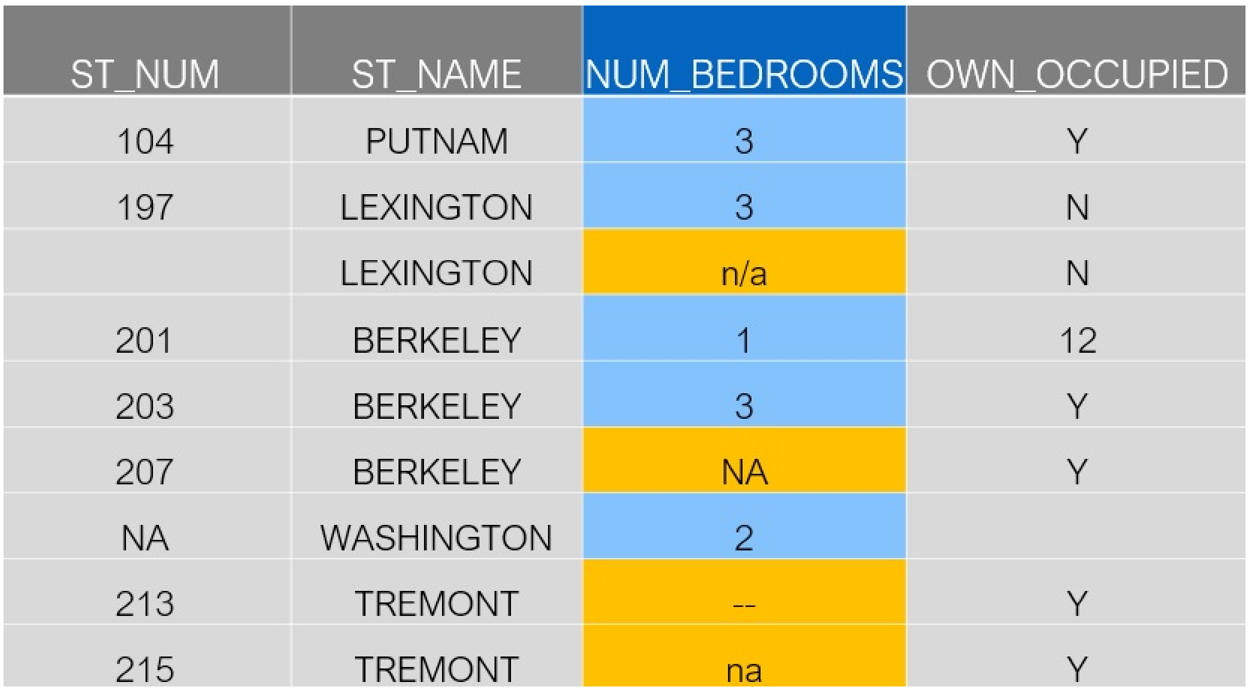
Taking a look at the column, we can see that Pandas filled in the blank space with “NA”. Using the `isnull()` method, we can confirm that both the missing value and “NA” were recognized as missing values. Both boolean responses are `True`.

This is a simple example, but highlights an important point. Pandas will recognize both empty cells and “NA” types as missing values. In the next section, we’ll take a look at some types that Pandas won’t recognize.

Non-Standard Missing Values

Sometimes it might be the case where there’s missing values that have different formats.

Let’s take a look at the “Number of Bedrooms” column to see what I mean.



ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

In this column, there's four missing values.

- n/a
- NA
- —
- na

From the previous section, we know that Pandas will recognize "NA" as a missing value, but what about the others? Let's take a look.

```
# Looking at the NUM_BEDROOMS column
print df['NUM_BEDROOMS']
print df['NUM_BEDROOMS'].isnull()
```
Copy

```
Out:
0      3
1      3
2    n/a
3      1
4      3
5    NaN
6      2
7     --
8     na

Out:
0    False
1    False
2    False
3    False
4    False
5     True
6    False
7    False
8    False
```
Copy

Just like before, Pandas recognized the “NA” as a missing value. Unfortunately, the other types weren’t recognized.

If there’s multiple users manually entering data, then this is a common problem. Maybe i like to use “n/a” but you like to use “na”.

An easy way to detect these various formats is to put them in a list. Then when we import the data, Pandas will recognize them right away. Here’s an example of how we would do that.

```
# Making a list of missing value types
missing_values = ["n/a", "na", "--"]
df = pd.read_csv("property data.csv", na_values = missing_values)
```
Copy

Now let’s take another look at this column and see what happens.

```
# Looking at the NUM_BEDROOMS column
print df['NUM_BEDROOMS']
print df['NUM_BEDROOMS'].isnull()
```
Copy

```
Out:
0    3.0
1    3.0
2    NaN
3    1.0
4    3.0
5    NaN
6    2.0
7    NaN
8    NaN

Out:
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7     True
8     True
```
Copy

This time, all of the different formats were recognized as missing values.

You might not be able to catch all of these right away. As you work through the data and see other types of missing values, you can add them to the list.

It's important to recognize these non-standard types of missing values for purposes of summarizing and transforming missing values. If you try and count the number of missing values before converting these non-standard types, you could end up missing a lot of missing values.

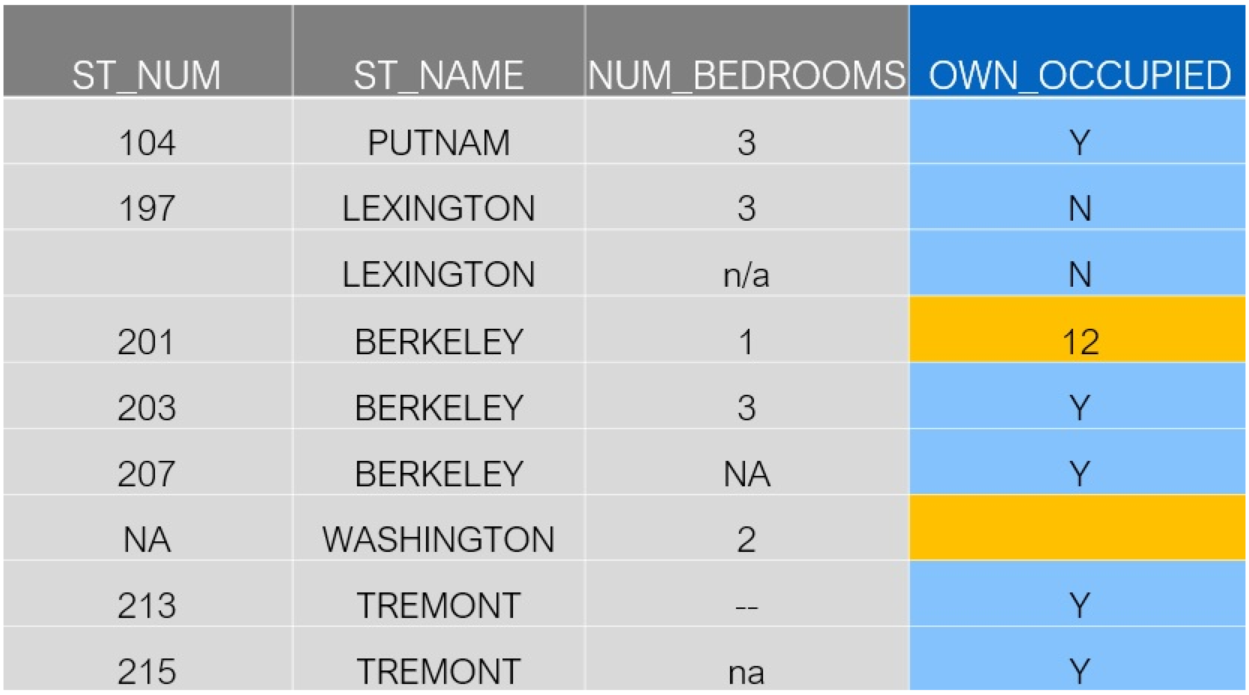
In the next section we'll take a look at a more complicated, but very common, type of missing value.

Unexpected Missing Values

So far we've seen standard missing values, and non-standard missing values. What if we an unexpected type?

For example, if our feature is expected to be a string, but there's a numeric type, then technically this is also a missing value.

Let's take a look at the "Owner Occupied" column to see what I'm talking about.



ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y

From our previous examples, we know that Pandas will detect the empty cell in row seven as a missing value. Let's confirm with some code.

```
# Looking at the OWN_OCCUPIED column
print df['OWN_OCCUPIED']
print df['OWN_OCCUPIED'].isnull()
```
Copy

```
# Looking at the ST_NUM column
Out:
0       Y
1       N
2       N
3      12
4       Y
5       Y
6     NaN
7       Y
8       Y

Out:
0     False
1     False
2     False
3     False
4     False
5     False
6      True
7     False
8     False
```
Copy

In the fourth row, there's the number 12. The response for Owner Occupied should clearly be a string (Y or N), so this numeric type should be a missing value.

This example is a little more complicated so we'll need to think through a strategy for detecting these types of missing values. There's a number of different approaches, but here's the way that I'm going to work thorugh this one.

1. Loop through the OWN_OCCUPIED column
2. Try and turn the entry into an integer
3. If the entry can be changed into an integer, enter a missing value
4. If the number can't be an integer, we know it's a string, so keep going

Let's take a look at the code and then we'll go through it in detail.

```
# Detecting numbers
cnt=0
for row in df['OWN_OCCUPIED']:
    try:
        int(row)
        df.loc[cnt, 'OWN_OCCUPIED']=np.nan
    except ValueError:
        pass
    cnt+=1
```
Copy

In the code we're looping through each entry in the "Owner Occupied" column. To try and change the entry to an integer, we're using `int(row)`.

If the value can be changed to an integer, we change the entry to a missing value using Numpy's `np.nan`.

On the other hand, if it can't be changed to an integer, we `pass` and keep going.

You'll notice that I used `try` and `except ValueError`. This is called exception handling, and we use this to handle errors.

If we were to try and change an entry into an integer and it couldn't be changed, then a `ValueError` would be returned, and the code would stop. To deal with this, we use exception handling to recognize these errors, and keep going.

Another important bit of the code is the `.loc` method. This is the preferred Pandas method for modfiying entries in place. For more info on this you can check out the Pandas documentation.

Now that we've worked through the different ways of detecting missing values, we'll take a look at summarizing, and replacing them.

Summarizing Missing Values

After we've cleaned the missing values, we will probably want to summarize them. For instance, we might want to look at the total number of missing values for each feature.

```
# Total missing values for each feature
print df.isnull().sum()
```
Copy

```
Out:
ST_NUM          2
ST_NAME         0
OWN_OCCUPIED    2
NUM_BEDROOMS    4
```
Copy

Other times we might want to do a quick check to see if we have any missing values at all.

```
# Any missing values?
print df.isnull().values.any()
```
Copy

```
Out:
True
```
Copy

We might also want to get a total count of missing values.

```
# Total number of missing values
print df.isnull().sum().sum()
```
Copy

```
Out:
8
```
Copy

Now that we've summarized the number of missing values, let's take a look at doing some simple replacements.

Replacing

Often times you'll have to figure out how you want to handle missing values.

Sometimes you'll simply want to delete those rows, other times you'll replace them.

As I mentioned earlier, this shouldn't be taken lightly. We'll go over some basic imputations, but for a detailed statistical approach for dealing with missing data, check out these awesome slides from data scientist Matt Brems.

That being said, maybe you just want to fill in missing values with a single value.

```
# Replace missing values with a number
df['ST_NUM'].fillna(125, inplace=True)
```
Copy

More likely, you might want to do a location based imputation. Here's how you would do that.

```
# Location based replacement
df.loc[2,'ST_NUM'] = 125
```
Copy

A very common way to replace missing values is using a median.

```
# Replace using median
median = df['NUM_BEDROOMS'].median()
df['NUM_BEDROOMS'].fillna(median, inplace=True)
```
Copy

We’ve gone over a few simple ways to replace missing values, but be sure to check out Matt’s slides for the proper techniques.

Conclusion

Dealing with messy data is inevitable. Data cleaning is just part of the process on a data science project.

In this article we went over some ways to detect, summarize, and replace missing values.

For even more resources about data cleaning, check out these data science books.

Armed with these techniques, you’ll spend less time data cleaning, and more time exploring and modeling.

Leave a comment below and let me know which technique you find most useful!

Click here to get the FREE Data Cleaning Cheat Sheet

Learn Data Science Fast

Learn how to get a data science job with techniques that I used to land a job in under 3 months.

Sign Up

Get Access to the top Data Science books in 2018.
Get Access Now