Handling Missing Values

Drop missing values, or fill them in with an automated workflow.

Tutorial Data

Learn Tutorial
Data Cleaning

Course step
1 of 5

Welcome to the **Data Cleaning** course on Kaggle Learn!

Data cleaning is a key part of data science, but it can be deeply frustrating. Why are some of your text fields garbled? What should you do about those missing values? Why aren't your dates formatted correctly? How can you quickly clean up inconsistent data entry? In this course, you'll learn why you've run into these problems and, more importantly, how to fix them!

In this course, you'll learn how to tackle some of the most common data cleaning problems so you can get to actually analyzing your data faster. You'll work through five hands-on exercises with real, messy data and answer some of your most commonly-asked data cleaning questions.

In this notebook, we'll look at how to deal with missing values.

Take a first look at the data

The first thing we'll need to do is load in the libraries and dataset we'll be using.

For demonstration, we'll use a dataset of events that occured in American Football games. In the **following exercise**, you'll apply your new skills to a dataset of building permits issued in San Francisco.

]:

```
# modules we'll use
import pandas as pd
import numpy as np

# read in all our data
nfl_data = pd.read_csv("../input/nflplaybyplay2009to2016/NFL Play by Play 2009-2017 (v4).csv")

# set seed for reproducibility
np.random.seed(0)
```

The first thing to do when you get a new dataset is take a look at some of it. This lets you see that it all read in correctly and gives an idea of what's going on with the data. In this case, let's see if there are any missing values, which will be reprsented with `NaN` or `None`.

In [2]:
```
# look at the first five rows of the nfl_data file.
# I can see a handful of missing data already!
nfl_data.head()
```

Out[2]:

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Awa
0	2009-09-10	2009091000	1	1	NaN	15:00	15	3600.0	0.0	TEN	...	NaN	0.485675	0.5
1	2009-09-10	2009091000	1	1	1.0	14:53	15	3593.0	7.0	PIT	...	1.146076	0.546433	0.4
2	2009-09-10	2009091000	1	1	2.0	14:16	15	3556.0	37.0	PIT	...	NaN	0.551088	0.4
3	2009-09-10	2009091000	1	1	3.0	13:35	14	3515.0	41.0	PIT	...	-5.031425	0.510793	0.4
4	2009-09-10	2009091000	1	1	4.0	13:27	14	3507.0	8.0	PIT	...	NaN	0.461217	0.5

5 rows × 102 columns

Yep, it looks like there's some missing values.

How many missing data points do we have?

Ok, now we know that we do have some missing values. Let's see how many we have in each column.

In [3]:
```
# get the number of missing data points per column
missing_values_count = nfl_data.isnull().sum()

# look at the # of missing points in the first ten columns
missing_values_count[0:10]
```

Out[3]:
```
Date               0
GameID             0
Drive              0
qtr                0
down           61154
time             224
TimeUnder          0
TimeSecs         224
PlayTimeDiff     444
SideofField      528
dtype: int64
```

That seems like a lot! It might be helpful to see what percentage of the values in our dataset were missing to give us a better sense of the scale of this problem:

In [4]:
```
# how many total missing values do we have?
total_cells = np.product(nfl_data.shape)
total_missing = missing_values_count.sum()

# percent of data that is missing
percent_missing = (total_missing/total_cells) * 100
print(percent_missing)
```

```
24.87214126835169
```

Wow, almost a quarter of the cells in this dataset are empty! In the next step, we're going to take a closer look at some of the columns with missing values and try to figure out what might be going on with them.

Figure out why the data is missing

This is the point at which we get into the part of data science that I like to call "data intution", by which I mean "really looking at your data and trying to figure out why it is the way it is and how that will affect your analysis". It can be a frustrating part of data science, especially if you're newer to the field and don't have a lot of experience. For dealing with missing values, you'll need to use your intution to figure out why the value is missing. One of the most important questions you can ask yourself to help figure this out is this:

> **Is this value missing because it wasn't recorded or because it doesn't exist?**

If a value is missing becuase it doesn't exist (like the height of the oldest child of someone who doesn't have any children) then it doesn't make sense to try and guess what it might be. These values you probably do want to keep as `NaN`. On the other hand, if a value is missing because it wasn't recorded, then you can try to guess what it might have been based on the other values in that column and row. This is called **imputation**, and we'll learn how to do it next! :)

Let's work through an example. Looking at the number of missing values in the `nfl_data` dataframe, I notice that the column "TimesSec" has a lot of missing values in it:

In [5]:
```
# look at the # of missing points in the first ten columns
missing_values_count[0:10]
```

Out[5]:
```
Date               0
GameID             0
Drive              0
qtr                0
down           61154
time             224
TimeUnder          0
TimeSecs         224
PlayTimeDiff     444
SideofField      528
dtype: int64
```

By looking at the documentation, I can see that this column has information on the number of seconds left in the game when the play was made. This means that these values are probably missing because they were not recorded, rather than because they don't exist. So, it would make sense for us to try and guess what they should be rather than just leaving them as NA's.

On the other hand, there are other fields, like "PenalizedTeam" that also have lot of missing fields. In this case, though, the field is missing because if there was no penalty then it doesn't make sense to say *which* team was penalized. For this column, it would make more sense to either leave it empty or to add a third value like "neither" and use that to replace the NA's.

> **Tip**: This is a great place to read over the dataset documentation if you haven't already! If you're working with a dataset that you've gotten from another person, you can also try reaching out to them to get more information.

If you're doing very careful data analysis, this is the point at which you'd look at each column individually to figure out the best strategy for filling those missing values. For the rest of this notebook, we'll cover some "quick and dirty" techniques that can help you with missing values but will probably also end up removing some useful information or adding some noise to your data.

Drop missing values

If you're in a hurry or don't have a reason to figure out why your values are missing, one option you have is to just remove any rows or columns that contain missing values. (Note: I don't generally recommend this approch for important projects! It's usually worth it to take the time to go through your data and really look at all the columns with missing values one-by-one to really get to know your dataset.)

If you're sure you want to drop rows with missing values, pandas does have a handy function, `dropna()` to help you do this. Let's try it out on our NFL dataset!

```
# remove all the rows that contain a missing value
nfl_data.dropna()
```

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Away_WP_pre	H

Oh dear, it looks like that's removed all our data! 😱 This is because every row in our dataset had at least one missing value. We might have better luck removing all the *columns* that have at least one missing value instead.

```
# remove all columns with at least one missing value
columns_with_na_dropped = nfl_data.dropna(axis=1)
columns_with_na_dropped.head()
```

:

	Date	GameID	Drive	qtr	TimeUnder	ydstogo	ydsnet	PlayAttempted	Yards.Gained	sp	...	Timeout_Indicator	Timeout_Team
0	2009-09-10	2009091000	1	1	15	0	0	1	39	0	...	0	None
1	2009-09-10	2009091000	1	1	15	10	5	1	5	0	...	0	None
2	2009-09-10	2009091000	1	1	15	5	2	1	-3	0	...	0	None
3	2009-09-10	2009091000	1	1	14	8	2	1	0	0	...	0	None
4	2009-09-10	2009091000	1	1	14	8	2	1	0	0	...	0	None

```
# just how much data did we lose?
print("Columns in original dataset: %d \n" % nfl_data.shape[1])
print("Columns with na's dropped: %d" % columns_with_na_dropped.shape[1])
```

```
Columns in original dataset: 102

Columns with na's dropped: 41
```

We've lost quite a bit of data, but at this point we have successfully removed all the `NaN`'s from our data.

Filling in missing values automatically

Another option is to try and fill in the missing values. For this next bit, I'm getting a small sub-section of the NFL data so that it will print well.

```
# get a small subset of the NFL dataset
subset_nfl_data = nfl_data.loc[:, 'EPA':'Season'].head()
subset_nfl_data
```

:

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA
0	2.014474	NaN	NaN	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	NaN
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244
2	-1.402760	NaN	NaN	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	NaN
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663
4	2.097796	NaN	NaN	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	NaN

We can use the Panda's `fillna()` function to fill in missing values in a dataframe for us. One option we have is to specify what we want the `NaN` values to be replaced with. Here, I'm saying that I would like to replace all the `NaN` values with 0.

```
# replace all NA's with 0
subset_nfl_data.fillna(0)
```

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA
0	2.014474	0.000000	0.000000	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	0.000000
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244
2	-1.402760	0.000000	0.000000	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	0.000000
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	0.000000

I could also be a bit more savvy and replace missing values with whatever value comes directly after it in the same column. (This makes a lot of sense for datasets where the observations have some sort of logical order to them.)

```
# replace all NA's the value that comes directly after it in the same column,
# then replace all the remaining na's with 0
subset_nfl_data.fillna(method='bfill', axis=0).fillna(0)
```

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA
0	2.014474	-1.068169	1.146076	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	-0.032244
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244
2	-1.402760	3.318841	-5.031425	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	0.106663
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	0.000000