Deal with missing values

- Take a look at the data
- See how many missing data points
- Figure out why the data is missing
- Drop missing values
- Filling in missing values

Take a look at the data

In [1]:
```
import pandas as pd
import numpy as np

nfl_data = pd.read_csv("../input/nflplaybyplay2009to2016/NFL Play by Play 2009-2017 (v4).csv")
sf_permits = pd.read_csv("../input/building-permit-applications-data/Building_Permits.csv")

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

```
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2698: DtypeWarning: Columns (25,51) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
/opt/conda/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2698: DtypeWarning: Columns (22,32) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

The first thing I do when I receive a new dataset is take a look at some of its contents. This allows me to verify whether it has been read in correctly and to gain an initial understanding of the data. Specifically, I check for any missing values, typically represented as 'NaN' or 'None'.

In [2]:
```
# look at a few rows of the nfl_data file. see a handful of missing data.
nfl_data.sample(5)
```

Out[2]:

	Date	GameID	Drive	qtr	down	time	TimeUnder	TimeSecs	PlayTimeDiff	SideofField	...	yacEPA	Home_WP_pre	Away_WI
244485	2014-10-26	2014102607	18	3	1.0	00:39	1	939.0	12.0	TB	...	1.240299	0.225647	0.77
115340	2011-11-20	2011112000	22	4	1.0	06:47	7	407.0	44.0	OAK	...	NaN	0.056036	0.94
68357	2010-11-14	2010111401	8	2	NaN	00:23	1	1823.0	0.0	CLE	...	NaN	0.365307	0.63
368377	2017-09-24	2017092405	24	4	1.0	08:48	9	528.0	8.0	CLE	...	1.075660	0.935995	0.06
384684	2017-11-05	2017110505	11	2	1.0	09:15	10	2355.0	0.0	DEN	...	NaN	0.928474	0.07

5 rows × 102 columns

it looks like there's some missing values.

See how many missing data points we have

In [4]:
```
# get the number of missing data points per column
missing_values_count = nfl_data.isnull().sum()

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

Out[4]:
```
Date               0
GameID             0
Drive              0
qtr                0
down           61154
time             224
TimeUnder          0
TimeSecs         224
PlayTimeDiff     444
SideofField      528
dtype: int64
```

It might be helpful to determine the percentage of missing values in our dataset to gain a better sense of the scale of this problem:

In [5]:
```
# 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
(total_missing/total_cells) * 100
```

Out[5]:
```
24.87214126835169
```

Almost a quarter of the cells in this dataset are empty.

Figure out why the data is missing

This marks the point where we delve into a facet of data science I like to term 'data intuition'—the practice of deeply examining your data to comprehend why it presents as it does and how it influences your analysis. It can prove challenging, particularly for newcomers in the field, lacking substantial experience.

When handling missing values, leveraging your intuition becomes pivotal to discern why a particular value is absent. One crucial question to aid in this determination is:

'Is this value missing because it was unrecorded, or because it doesn't exist?'

If a value is missing because it simply doesn't exist (for instance, the height of a non-existent child for a person without children), attempting to conjecture its value becomes futile. These values should likely be retained as NaN (Not a Number). Conversely, if a value is absent due to being unrecorded, you can endeavor to estimate it based on other values within that column and row—a process termed 'imputation.'

Let's illustrate this with an example. Upon inspecting the nfl_data dataframe for missing values, I observe that the 'TimesSec' column contains numerous missing entries.

In [7]:
```
# look at the missing points in the first ten columns
missing_values_count[0:10]
```

Out[7]:
```
Date               0
GameID             0
Drive              0
qtr                0
down           61154
time             224
TimeUnder          0
TimeSecs         224
PlayTimeDiff     444
SideofField      528
dtype: int64
```

If you're conducting meticulous data analysis, this marks the juncture where you'd meticulously examine each column independently to determine the optimal approach for addressing those missing values. Throughout the remainder of this notebook, we'll explore some 'quick and dirty' techniques. While these methods can aid in handling missing values, they might inadvertently remove valuable information or introduce noise to your dataset.

Drop missing values

If you're pressed for time or lack a compelling reason to investigate the reasons behind missing values, one option available is to simply eliminate any rows or columns containing these missing entries. (Please note: I typically don't advocate this approach for significant projects! Taking the time to thoroughly examine each column with missing values can be invaluable in understanding your dataset.)

Should you opt to remove rows with missing values, pandas offers a convenient function, dropna(), to facilitate this task. Let's apply it to our NFL dataset.

In [8]:
```
# remove all the rows that contain a missing value
nfl_data.dropna()
```

Out[8]:

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

0 rows × 102 columns

In [9]:
```
# remove all columns with at least one missing value
columns_with_na_dropped = nfl_data.dropna(axis=1)
columns_with_na_dropped.head()
```

Out[9]:

	Date	GameID	Drive	qtr	TimeUnder	ydstogo	ydsnet	PlayAttempted	Yards.Gained	sp	...	Timeout_Indicator	Timeout_Team	post
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	

5 rows × 41 columns

In [10]:
```
# 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 attempt to fill in the missing values. For this upcoming section, I'm extracting a small subsection of the NFL data to ensure it prints legibly.

In [13]:
```
# get a small subset of the NFL dataset
subset_nfl_data = nfl_data.loc[:, 'EPA':'Season'].head()
subset_nfl_data
```

Out[13]:

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA	yacWPA
0	2.014474	NaN	NaN	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	NaN	NaN
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244	0.036899
2	-1.402760	NaN	NaN	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	NaN	NaN
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663	-0.156239
4	2.097796	NaN	NaN	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	NaN	NaN

We can employ Panda's fillna() function to replace missing values within a dataframe. One approach is specifying the value we want to use as a replacement for NaN values. In this instance, I'm indicating that I'd like to replace all the NaN values with 0.

In [14]:
```
# replace all NA's with 0
subset_nfl_data.fillna(0)
```

Out[14]:

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA	yacWPA
0	2.014474	0.000000	0.000000	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	0.000000	0.000000
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244	0.036899
2	-1.402760	0.000000	0.000000	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	0.000000	0.000000
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663	-0.156239
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	0.000000	0.000000

An alternative strategy is to replace missing values with the succeeding value in the same column, a method especially effective for datasets where observations adhere to a logical order.

In [15]:
```
# replace all NA's value that comes directly after it in the same column,
# then replace all the reamining na's with 0
subset_nfl_data.fillna(method = 'bfill', axis=0).fillna(0)
```

Out[15]:

	EPA	airEPA	yacEPA	Home_WP_pre	Away_WP_pre	Home_WP_post	Away_WP_post	Win_Prob	WPA	airWPA	yacWPA
0	2.014474	-1.068169	1.146076	0.485675	0.514325	0.546433	0.453567	0.485675	0.060758	-0.032244	0.036899
1	0.077907	-1.068169	1.146076	0.546433	0.453567	0.551088	0.448912	0.546433	0.004655	-0.032244	0.036899
2	-1.402760	3.318841	-5.031425	0.551088	0.448912	0.510793	0.489207	0.551088	-0.040295	0.106663	-0.156239
3	-1.712583	3.318841	-5.031425	0.510793	0.489207	0.461217	0.538783	0.510793	-0.049576	0.106663	-0.156239
4	2.097796	0.000000	0.000000	0.461217	0.538783	0.558929	0.441071	0.461217	0.097712	0.000000	0.000000

Filling in missing values is also known as "imputation".