

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)

LESSON 5: MISSING DATA

Lesson 5: Missing Data

One of the most common problems when performing data analysis is that there may be data missing from your data set. This problem can impact your analysis conclusions, so you need to understand the impact of missing data on your analysis based upon the reason that the data is missing. How do you handle situations where data are missing? You have several options, and your response will vary based upon the circumstances of your data set and require some subject matter expertise. You may be able to figure out the missing values based upon analysis of other available data. You might be able to interpolate missing values based upon other observations or you might be able to simply ignore the missing data.

Although missing data is never a great thing, it is preferred that there is no reason or pattern in the missing data. This is sometimes referred to as "missing at random," or MAR. If you look at the table on the right, you see that you have missing data. Compare the table on the right with the non-missing data on the left. See if you believe this is MAR:

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

44	118
46	93
48	141
51	104
51	116
54	97

44	118
46	93
48	141
51	104
51	116
54	97

Tables comparing complete data versus incomplete data

As you look at this and compare it to the table on the left, you might notice a troubling pattern: The missing IQ scores belong to anyone under 40 years of age. You are missing your younger subjects. If you were to analyze the data on the right, your results could potentially be biased toward older subjects. This missing pattern would not be considered MAR, and you should be more motivated to investigate and potentially fix this.

There are many ways to find out how much missing data you have in a data frame. Missing data itself can look different depending on your data set. Here you will see two examples, NA values in R and NaN values in Python. In addition to these, you can have NULL values, and in some data sets, you might see blank spaces, " ", "-", "_", or at times a 0 value might indicate missing data. If your data set comes with a data dictionary for the values in the columns, be sure to read that carefully as you are exploring the data.

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

```
# R Example

#Build an example dataframe
df <- data.frame(col1 = c(1:4, NA, NA, NA),
  col2 = c(1:6,NA),
  col3 = c(NA, 3.2, 5.7, 11.8 , NA, NA, NA),
  col4 = c('A','A','B','C',NA,'A',NA),
  col5 = c(2, 2, 2, NA, 3, 3,NA))

#Find where we have NAs in our dataframe
is.na(df)

#Finding the number of NAs in a single column
sum(is.na(df$col1))

#Find the rows that have an NA
which(is.na(df$col1))

#Find how many NA's are in each column
colSums(is.na(df))

#Calculate the number of NAs in a Row
apply(df, 1, function(x) sum(is.na(x)))

# R Ouput

> is.na(df)
```



Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

```
> #Finding the number of NAs in a single column
> sum(is.na(df$col1))
[1] 3
>
> #Find the rows that have an NA
> which(is.na(df$col1))
[1] 5 6 7
>
> #Find how many NA's are in each column
> colSums(is.na(df))
col1 col2 col3 col4 col5
3 1 4 2 2
>
> #Calculate the number of NAs in a Row
> apply(df, 1, function(x) sum(is.na(x)))
[1] 1 0 0 1 3 2 5
```

Find missing data using R

[Download the above R code and output \(opens new tab\)](#)

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

example, for Col 1). The which command gives you the location of the missing values. The colSums command can give you totals for all the data variables. At times, you might want to see which row has the most missing values. You can use the apply function for this, as shown above. In this example, you see that the last row has five missing values (which you can verify by looking at the original data).

Now it is time to address the different issues. First, you want to delete the final row of your data frame. There is clearly something wrong when all of the columns are NA. You should not want to replace the entire row with imputed values in this case. In the code below, create a new container of data called df2 (this name was chosen for this exercise; you could have chosen something else). In the code, remove the seventh row by using -7 in brackets. In R, rows are presented first, then columns after a comma. In this example, you do not want to do anything with the columns, so you will notice there is no value after the column.

```
#####  
  
# Dealing with NAs  
  
#####  
  
df2 <- df[-7, ]  
  
# This removes the 7th row of the data set
```

Using the R language to remove rows from a data set.

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

these. In this example, there is no reason one method was picked over the other except for column 4 (since that is categorical). The choices made in the example are mostly to illustrate options. (Note: to calculate the mode, create a "function" in R because there is no built-in option to do it.)

```
df2$col1 <- replace(df$col1, is.na(df$col1), median(df$col1, na.rm=TRUE))
df2$col3 <- replace(df$col3, is.na(df$col3), mean(df$col3, na.rm=TRUE))

#Create a mode function

mode_funct <- function(x) {
  col_tbl <- table(x)
  names(col_tbl)[which(col_tbl==max(col_tbl))]
}

df2$col4 <- replace(df$col4, is.na(df$col4), mode_funct(df$col4))
df2$col5 <- replace(df$col5, is.na(df$col5), as.numeric(mode_funct(df$col5)))
```

Replacing missing data

[Download the R code and output \(opens new tab\)](#)

Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

3	3	3	5.7	B	2
4	4	4	11.8	C	NA
5	NA	5	NA	NA	3
6	NA	6	NA	A	3
7	NA	NA	NA	NA	NA

3	3.0	3	5.7	B	2
4	4.0	4	11.8	C	2
5	2.5	5	6.9	A	3
6	2.5	6	6.9	A	3
7	2.5	NA	6.9	A	2

Table comparison depicting replacement of missing data

You can see for column 3, all the NAs are now replaced with 6.9. For column 4, the missing data are replaced with the value that occurs the most: A. For illustration purposes, column 2 was not modified.

As you can see, R has built-in functions to handle the median and mean, but there is not a base function to find the mode. This code will accomplish this in this case, but there are two points to keep in mind.

1. This code returns a string that is good for column 4, but for column 5 you need to change the data type with the `as.numeric` function.
2. This function will return multiple modes if they exist in the column, and it throws an error for the `replace` command if it occurs.

Below is how you would approach this using Python

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

In [2]: df = pd.DataFrame(data= {'col1' : [1, 2, 3, 4, np.nan, np.nan, np.nan],
                                'col2' : [1, 2, 3, 4, 5, 6, np.nan],
                                'col3' : [np.nan, 3.2, 5.7, 11.8, np.nan, np.nan, np.nan],
                                'col4' : ['A', 'A', 'B', 'C', np.nan, 'A', np.nan],
                                'col5' : [2, 2, 2, np.nan, 3, 3, np.nan]})
```



Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

```
dtype: int64
```

```
In [5]: #Find the number of NaNs in each row
df.isnull().sum(axis=1)
```

```
Out[5]: 0    1
        1    0
        2    0
        3    1
        4    3
        5    2
        6    5
dtype: int64
```

```
In [6]: # Drop the last row of the data frame
df = df.drop([6])
print(df)
```

```
   col1  col2  col3 col4  col5
0  1.0   1.0   NaN   A    2.0
1  2.0   2.0   3.2   A    2.0
2  3.0   3.0   5.7   B    2.0
3  4.0   4.0  11.8   C    NaN
4  NaN   5.0   NaN   NaN   3.0
5  NaN   6.0   NaN   A    3.0
```

```
In [7]: # Replace the NaNs in col1 with the median
df['col1'].fillna(df['col1'].median(), inplace = True)
```

```
In [8]: # Replace the NaNs in col3 with the mean
df['col3'].fillna(df['col3'].mean(), inplace = True)
```

```
In [9]: # Replace the NaNs in col5 with the mode
df['col5'].fillna(df['col5'].mode(), inplace = True)
```

```
In [10]: # Create a function to find the most frequent string
most_common_str = max(list(df['col4']), key=list(df['col4']).count)
```

```
In [11]: df['col4'].fillna(most_common_str, inplace = True)
```

```
In [12]: print(df)
```

```
   col1  col2  col3 col4  col5
0  1.0   1.0   6.9   A    2.0
1  2.0   2.0   3.2   A    2.0
2  3.0   3.0   5.7   B    2.0
3  4.0   4.0  11.8   C    NaN
4  2.5   5.0   6.9   A    3.0
```



Data Cleaning

Section 3: Missing Data, Outlier Detection, and Principal Component Analysis (PCA)



LESSON 5: MISSING DATA

here is how Python's numbering differs from R. In Python, the first value has an index of 0, and in R it is indexed as 1. Also, you will find that Python usually treats sequences as not including the last number. Below is a quick example of the difference in syntax.

```
In [1]: numbers = list([1,2,3,4,5,6,7,8])
```

```
In [2]: numbers[0:3]
```

```
Out[2]: [1, 2, 3]
```

Python code

```
# R Example  
numbers = c(1:8)
```

```
numbers[1:3]
```

```
# Output
```

```
> numbers[1:3]  
[1] 1 2 3
```

R code

