SSE 691: Intro to Data Science Project 3

Jonathan Alfred
9-28-2016

Contents

Introduction

In this chapter, we learn of statistical analysis using Python. The chapter goes over several features of statistics, coding them using basic libraries within Python. These were then recoded using Pandas, along with some extended code on Pandas functionality.

Chapter 5

Each section of the chapter contained code that needed to be redone in Pandas. Both the initial code and the new code are displayed and discussed in each section, highlighting key difference between the normal Python framework and Pandas series.

Describing a Single Set of Data

The section goes over how to describe data in ways that are meaningful for data analysis. It explains that simply displaying all the data will not assist in understanding the data at all. Instead, the text suggests putting the data into a histogram, for easily displaying whatever data is necessary. This may not work in all cases, but, for the example of a data science social networking site, this works well in organizing the number of people who have a certain number of friends. The code to display the histogram is seen below, along with the histogram output in Figure 1.

```
friends_counts = Counter(num_friends)
xs = range(101)
ys = [friends_counts[x] for x in xs]
plt.bar(xs, ys)
plt.axis([0, 101, 0, 25])
plt.title("Histogram of Friend Counts")
plt.xlabel("# of friends")
plt.ylabel("# of people")
plt.show()
```

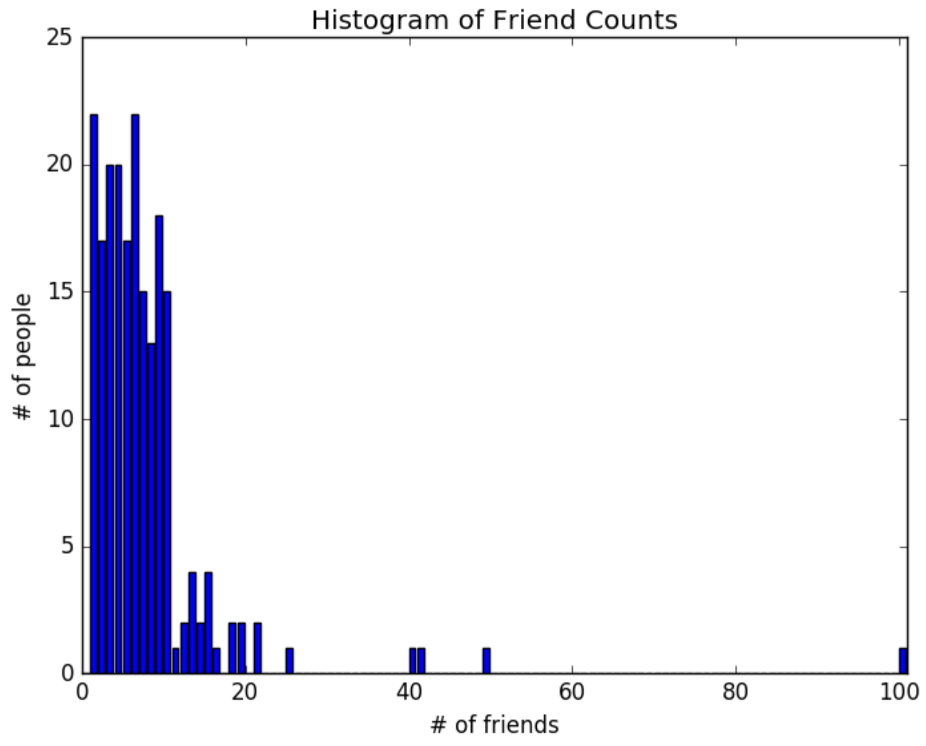


Figure 1 - Histogram output using matplotlib

Changing this code to Pandas was very quick and easy. Pandas architecture allows the user to convert lists into the Pandas type Series. Another datatype that could be used would be Dataframes, but for the remainder of this project, Series were used. One issue that was noted in the coversion was modifying the x ticks so that not ever number was displayed at the bottom. This, however, could not be done easily, and was left the way it was. The code for the new histogram is seen below, along with the histogram output in Figure 2.

```
pd_num_friends = pd.Series(num_friends)
pd_xs =
pd_num_friends.value_counts(sort=True,bins=range(101)).sort_index()
plt.figure()
pd_plot = pd_xs.plot.bar(title="Histogram of Friend Counts")
pd_plot.set_xlabel("# of friends")
pd_plot.set_ylabel("# of people")
plt.show()
```

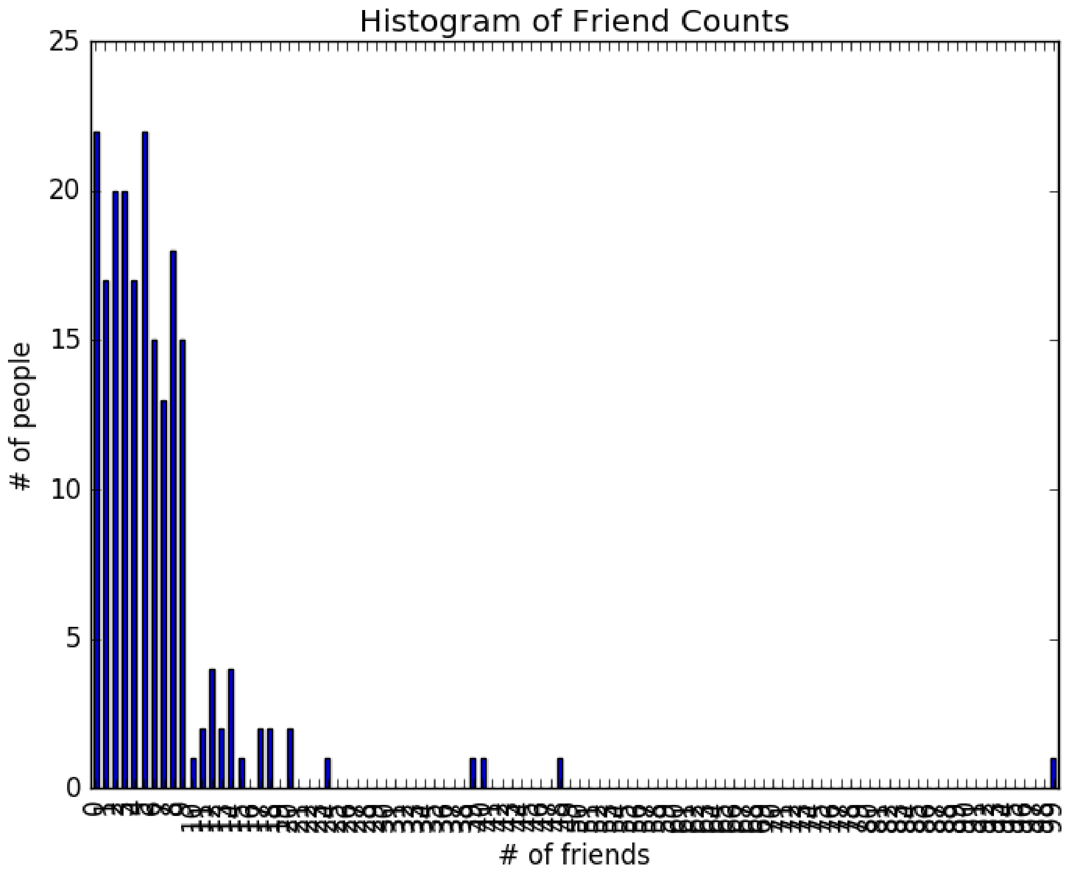


Figure 2 - Histogram using Pandas and matplotlib

While this code does display the data, this cannot be used in simple conversation, so other basic statistics could be taken from the dataset. This could be anything from the largest few values to the number of data points within a data set. Below shows a list displaying all these values.

```
num_points = len(num_friends)
largest_value = max(num_friends)
smallest_value = min(num_friends)
sorted_values = sorted(num_friends)
second_smallest_value = sorted_values[1]
second_largest_value = sorted_values[-2]

#Output
Num points: 204
Largest value: 100
Smallest value: 1
Sorted values: [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3,
3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,
6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,
9, 9, 9, 9, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 10, 11, 12, 12, 13, 13, 13, 13, 14, 14, 15, 15, 15, 15,
16, 18, 18, 19, 19, 21, 21, 25, 40, 41, 49, 100]
Second smallest: 1
Second largest: 49
```

Pandas is also able to do these with its Series datatype, using different keywords and built in functions to the Series datatype. Getting the second largest and smallest data points, however, was much simpler to do with lists. Asides from that, the code was basically the same, with edits simply for the Series datatype. The outputs were not included, as they were the same as the vanilla python values.

```
pd_num_points = pd_num_friends.size
pd_largest_value = pd_num_friends.max()
pd_smallest_value = pd_num_friends.min()
pd_sorted_values = pd_num_friends.sort_values()
print("Num points: {}".format(pd_num_points))
print("Largest value: {}".format(pd_largest_value))
print("Smallest value: {}".format(pd_smallest_value))
print("Sorted values: {}".format(pd_sorted_values))
```

Central Tendencies

Learning more about our data can often include finding where the central point of the data set is located. This is done by searching for the mean value of the set. A function was defined to do this within Python, seen below.

```
def mean(x):
    return sum(x)/len(x)

print(mean(num_friends))

#Output
#7.333333333333333
```

This code is much easier to do with a Pandas Series. The functionality for means are already programmed in as a method for the container, not requiring any extra coding. The code for displaying the mean value is seen below.

```
print(pd_num_friends.mean())

#Output
# 7.333333333333333
```

The median is another statistics that shows the approximate center of the data, searching through the number of values and returning the value directly in the center of the data set. The code for this in vanilla python is seen below:

```
def median(v):
    n = len(v)
    sorted_v = sorted(v)
    midpoint = n // 2
    if n%2 == 1:
        return(sorted_v[midpoint])
    else:
        lo = midpoint - 1
        hi = midpoint
        return (sorted_v[lo] + sorted_v[hi])/2

print(median(num_friends))

# 6.0
```

Again, Pandas has median built in to its Series objects, rather the needing to have code written for it. This code is seen below:

```
print(pd_num_friends.median())
```

Quantiles are useful for looking at the value at which a certain percentage of the data set lies below. An example of this would be looking at what value 50% of the population’s income lies below. The code for this in vanilla Python is seen below.

```
def quantile(x, p):
    p_index = int(p*len(x))
    return sorted(x)[p_index]
```

```
print(quantile(num_friends, .10))
print(quantile(num_friends, .25))
print(quantile(num_friends, .75))
print(quantile(num_friends, .90))

#1
#3
#9
#13
```

This is a defined function with Pandas. The extra flag ensures that a data point is selected, rather than interpolating and finding the exact number for which the population lies below. Nearest causes the quantile to search for the closest value to the exact number and select that as the quantile. The code is seen below.

```
print(pd_num_friends.quantile(.10, 'nearest'))
print(pd_num_friends.quantile(.25, 'nearest'))
print(pd_num_friends.quantile(.75, 'nearest'))
print(pd_num_friends.quantile(.90, 'nearest'))
```

Lastly, the mode was discussed. The mode is the most common item seen within a dataset. This can easily be done with a counter object within vanilla python. Code for this is seen below.

```
def mode(x):
    counts = Counter(x)
    max_count = (max(counts.values()))
    return [x_i for x_i, count in counts.items() if count ==
max_count]

print(mode(num_friends))

[1, 6]
```

Once again, Pandas has this functionality built in to the Series object, requiring it to just be called for the mode to be displayed. This is seen below. It is important to note that this returns another Series with all modal values inside of it.

```
print(pd_num_friends.mode())
```

Dispersion

Dispersion is important to look at within data sets. This refers to how spread out the data points are. Typically, larger values denote a greater spread of data, while smaller values denote a narrower spread.

Data ranges were the first type of dispersion discussed. This is the difference between the largest value in the data set and the smallest value in the data set. This can sometimes be misleading, especially if there are many outliers within the data set. The code for this is seen within vanilla python:

```
def data_range(x):
    return max(x) - min(x)

print(data_range(num_friends))

#99
```

This is a built in function to the Pandas Series datatype. The function is called ptp(), which made it a little more difficult to find. It does print out the data range though, returning 99 as the function above did. The code for displaying this is seen below.

```
print(pd_num_friends.ptp())
```

Variance is an important factor to look at, and a much closer view of dispersion than data range. This checks to see how far apart values are from the average of the dataset. This is often used for probability distribution. The variability of the data from the average can help display risk within making a specific decision. To code this, a few old snippets of code from prior chapters were used in the vanilla python version, resulting in the four functions below to create a variance.

```
def dot(v, w):
    return sum(v_i*w_i for v_i, w_i in zip(v, w))

def sum_of_squares(v):
    return dot(v, v)

def de_mean(x):
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]
```

```
def variance(x):
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations)/(n-1)

print(variance(num_friends))

#81.54351395730716
```

Pandas makes this much more simplified by included code with Series to calculate the variance without the need for multiple functions. This function is seen below and produces the same result as the code above, with one single line rather than twelve.

```
print(pd_num_friends.var())
```

Standard deviation is a similar measure to variance. Whereas variance returns a square of the original values, this takes the square root of the values, creating a smaller deviation within the actual units of the original data values. This seems to be used more often than variance. Code for this is seen below:

```
def standard_deviation(x):
    return math.sqrt(variance(x))

print(standard_deviation(num_friends))

#9.03014473623248
```

Standard deviation is a built in function, as variance was. This returns the same value as above. Code for the function is seen below.

```
print(pd_num_friends.std())
```

Correlation

Correlation intends to look at multiple datasets together and see how they line up. This can help users come to conclusions based off of different statistics, such as covariance and correlation. It can also be misleading, which will be discussed in the two following sections.

Covariance is a measure of how two different datasets vary from their means, together. When covariance is large, both tend to be larger than their means. When covariance is small, either one tends to be smaller while the other tends to be larger. This prevents the covariance from growing. The code for this is seen below, making use of code defined earlier within the paper.

```
def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y))/(n-1)

print(covariance(num_friends, daily_minutes))
```

Pandas has this functionality built into its framework, allowing for ease of use. In order to do so, the function must be called with a parameter of the Series that is used within the covariance. The code below shows a new Series being made, along with the covariance calculated.

```
pd_daily_minutes = pd.Series(daily_minutes)
print(pd_num_friends.cov(pd_daily_minutes))
```

Correlation shows how close the two data sets match. It shows the strength and the direction of the linear relationship between the variables of a scatter plot. The closer they are to one, the closer the values are to having perfect correlation. The closer to negative one, the closer the values are to having perfect anti-correlation. Code for correlation is seen below, along with a graph of the scatter plot being checked, in Figure 3.

```
def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y)/stdev_x/stdev_y
    else:
        return 0

print(correlation(num_friends, daily_minutes))
plt.scatter(num_friends, daily_minutes)
plt.title("Outlier")
plt.axis([0, 101, 0, 101])
plt.xlabel("# of friends")
plt.ylabel("# of minutes")
plt.show()

#0.24736957366478218
```

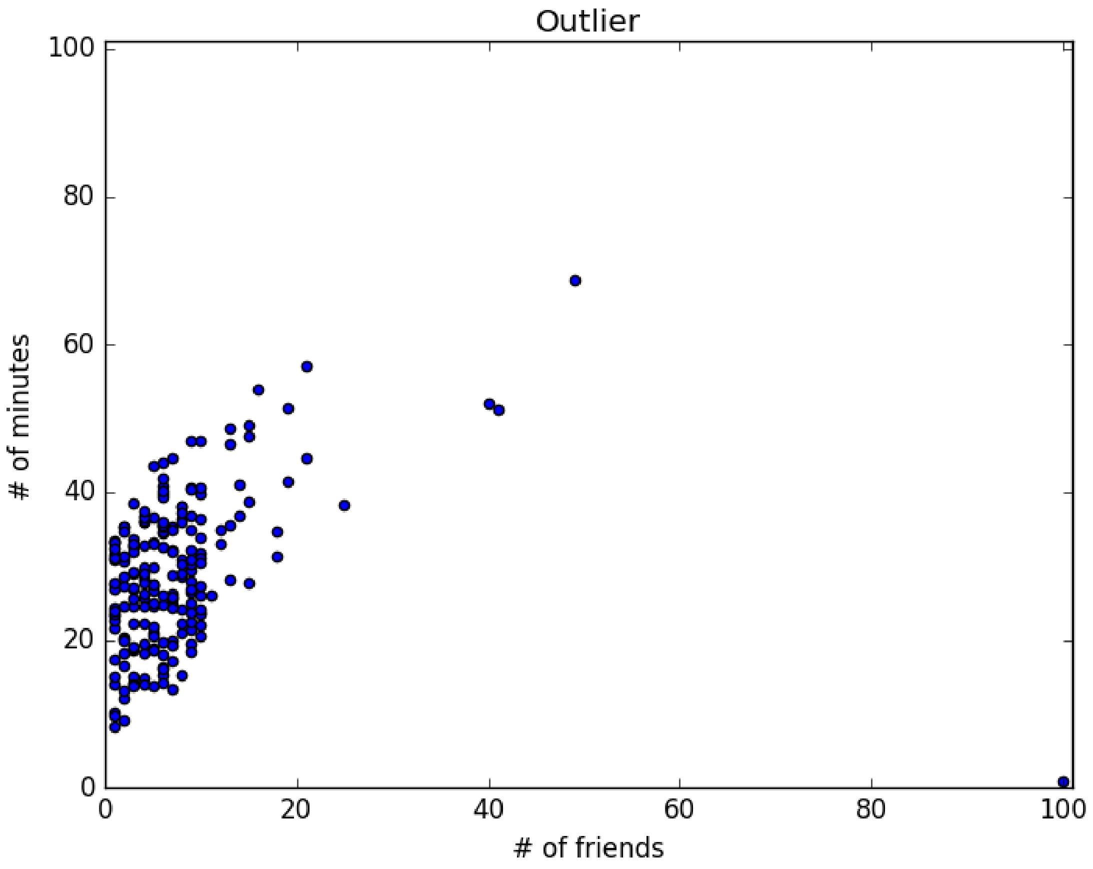


Figure 3 - Correlation with the outlier included

Pandas has this functionality built in to its Series variable. However, plotting a scatterplot is not possible with a series variable, and would be best with a dataframe object. Code for printing the correlation is seen below.

```
print(pd_num_friends.corr(pd_daily_minutes))
```

Correlation can be thrown off with the outlier, causing misleading correlations. Removing the outlier created a much stronger correlation between the values, seen below, along with a figure of the new scatterplot.

```
outlier = num_friends.index(100)
num_friends_good = [x
                   for i, x in enumerate(num_friends)
                   if i != outlier]
daily_minutes_good = [x
                     for i, x in enumerate(daily_minutes)
                     if i != outlier]
print(correlation(num_friends_good, daily_minutes_good))
plt.scatter(num_friends_good, daily_minutes_good)
plt.title("Outlier removed")
plt.axis([0, 101, 0, 101])
plt.xlabel("# of friends")
plt.ylabel("# of minutes")
plt.show()
#0.5736792115665573
```

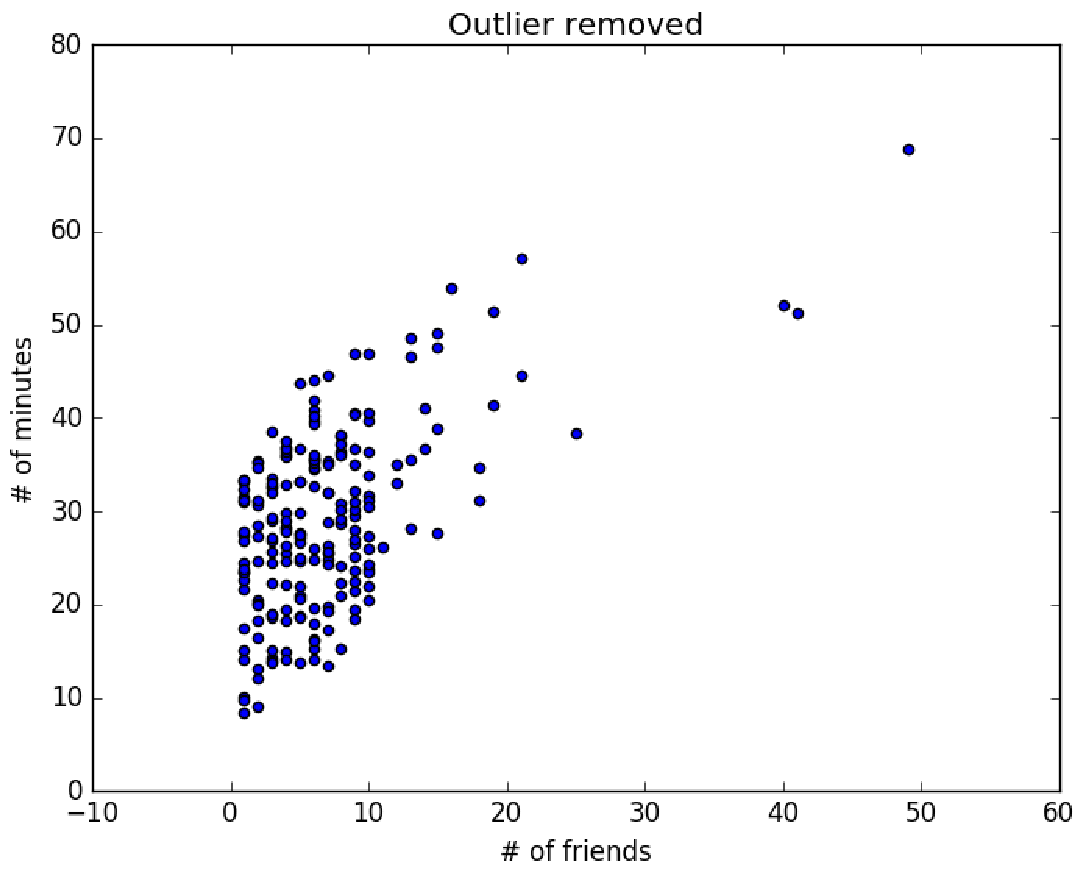


Figure 4 - Scatterplot with outlier removed

Simpson's Paradox

This section did not have any code to change. It instead went over the issues with correlations that ignore confounding variables. An example is shown, talking about whether data scientists of the East Coast or the West Coast have more friends. Overall, the average pointed to the West Coast having more friends. However, when splitting the data scientists into what degree they have, the new subsets showed that the East Coast subsets each had more friends than their West Coast counterparts. This is meant to highlight an important assertation that is often taken when using conduction many different statistical analyses: "all else being equal". This means that, when comparing the relationship between two variables, an assumption that there is nothing else that matters asides from these two variables is made. This can be fine with random assignments, but often times there are much deeper patterns that require more thorough investigations than this assumption allows for.

Some Other Correlational Caveats

Statistical analysis will only see what it is designed to look for. Sometimes obvious relationships can be overlooked simply because there was nothing actually searching for it. Examples of this can be seen in the following:

```
X = [-2, -1, 0, 1, 2]
Y = [2, 1, 0, 1, 2]
```

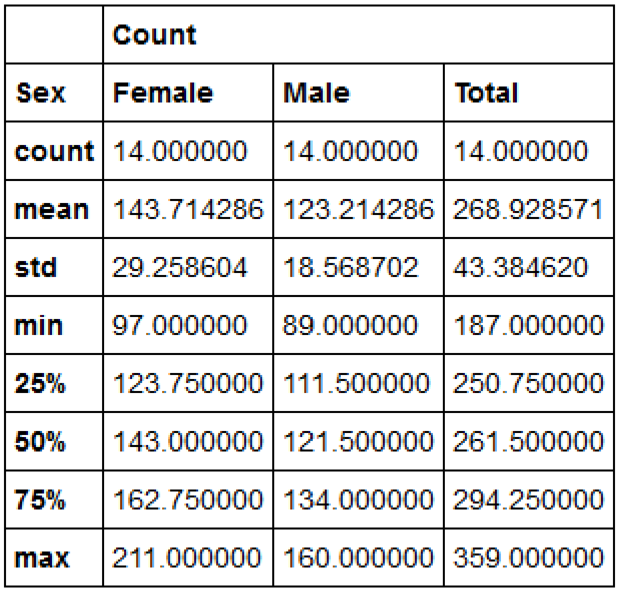
While it is obvious to us that Y is the absolute value of everything with X and Y, X and Y will have zero correlation. These are simple things that should not be overlooked when using statistical analysis.

Correlation and Causation

This is often said about data. While two sets of data may correlate, it does not necessarily mean that once causes the other. They both could be caused by separate things entirely and simply happen to correlate. They could be both caused by the same thing, which may explain while they correlate. There are many possibilities for correlations, and they do not always link back to causation.

Discussion

Pandas is a useful tool for statistical analysis. In past project, I have used dataframes to hold large amounts of data. All of the functions seen coded here within Python are easily available with Pandas. Rather than code in these functions, it would be easier to use Pandas. While I might have used series here, it may have been to my advantage to use a dataframe. However, most functionalities are shared between the two different datatypes. Dataframes have a little more functionality.



	Count		
Sex	**Female**	**Male**	**Total**
count	14.000000	14.000000	14.000000
mean	143.714286	123.214286	268.928571
std	29.258604	18.568702	43.384620
min	97.000000	89.000000	187.000000
25%	123.750000	111.500000	250.750000
50%	143.000000	121.500000	261.500000
75%	162.750000	134.000000	294.250000
max	211.000000	160.000000	359.000000

Figure 5 - Dataframe example, showing off multiple levels of a table with several statistics as well

I, as a student, have used many of these statistics before. However, my knowledge on when to use them and what conclusions to draw are rather limited. I know how to use them; I simply do not know when to use them. Hopefully throughout the course, I will have more opportunities to figure out how to utilize these tools that I have.

Indirect Logs

Date	Time	Activity
9/24/2016	100	Read through the chapter.
9/26/2016	120	Began coding.
9/27/2016	120	Continued coding.
9/28/2016	290	Continued coding. Drafted and completed document.
Total	630	

Code

```
import matplotlib.pyplot as plt
import matplotlib
import pandas as pd
import numpy as np
from collections import Counter
import py
import random as rand
import math
num_friends =
[100,49,41,40,25,21,21,19,19,18,18,16,15,15,15,15,14,14,13,13,13,13,12
,12,11,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,9,9,9,9,9,9,9,9,9,
9,9,9,9,9,9,9,9,9,8,8,8,8,8,8,8,8,8,8,8,8,8,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3,3,3,3,3,3,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
daily_minutes =
[1,68.77,51.25,52.08,38.36,44.54,57.13,51.4,41.42,31.22,34.76,54.01,38
.79,47.59,49.1,27.66,41.03,36.73,48.65,28.12,46.62,35.57,32.98,35,26.0
7,23.77,39.73,40.57,31.65,31.21,36.32,20.45,21.93,26.02,27.34,23.49,46
.94,30.5,33.8,24.23,21.4,27.94,32.24,40.57,25.07,19.42,22.39,18.42,46.
96,23.72,26.41,26.97,36.76,40.32,35.02,29.47,30.2,31,38.11,38.18,36.31
,21.03,30.86,36.07,28.66,29.08,37.28,15.28,24.17,22.31,30.17,25.53,19.
85,35.37,44.6,17.23,13.47,26.33,35.02,32.09,24.81,19.33,28.77,24.26,31
.98,25.73,24.86,16.28,34.51,15.23,39.72,40.8,26.06,35.76,34.76,16.13,4
4.04,18.03,19.65,32.62,35.59,39.43,14.18,35.24,40.13,41.82,35.45,36.07
,43.67,24.61,20.9,21.9,18.79,27.61,27.21,26.61,29.77,20.59,27.53,13.82
,33.2,25,33.1,36.65,18.63,14.87,22.2,36.81,25.53,24.62,26.25,18.21,28.
08,19.42,29.79,32.8,35.99,28.32,27.79,35.88,29.06,36.28,14.1,36.63,37.
49,26.9,18.58,38.48,24.48,18.95,33.55,14.24,29.04,32.51,25.63,22.22,19
,32.73,15.16,13.9,27.2,32.01,29.27,33,13.74,20.42,27.32,18.23,35.35,28
.48,9.08,24.62,20.12,35.26,19.92,31.02,16.49,12.16,30.7,31.22,34.65,13
.13,27.51,33.2,31.57,14.1,33.42,17.44,10.12,24.42,9.82,23.39,30.93,15.
03,21.67,31.09,33.29,22.61,26.89,23.48,8.38,27.81,32.35,23.84]

friends_counts = Counter(num_friends)
xs = range(101)
ys = [friends_counts[x] for x in xs]
plt.bar(xs, ys)
plt.axis([0, 101, 0, 25])
plt.title("Histogram of Friend Counts")
plt.xlabel("# of friends")
plt.ylabel("# of people")
plt.show()

pd_num_friends = pd.Series(num_friends)
pd_xs =
pd_num_friends.value_counts(sort=True,bins=range(101)).sort_index()
```

```
plt.figure()
pd_plot = pd_xs.plot.bar(title="Histogram of Friend Counts")
pd_plot.set_xlabel("# of friends")
pd_plot.set_ylabel("# of people")
plt.show()

num_points = len(num_friends)
largest_value = max(num_friends)
smallest_value = min(num_friends)
sorted_values = sorted(num_friends)
second_smallest_value = sorted_values[1]
second_largest_value = sorted_values[-2]
print("Num points: {}".format(num_points))
print("Largest value: {}".format(largest_value))
print("Smallest value: {}".format(smallest_value))
print("Sorted values: {}".format(sorted_values))
print("Second smallest: {}".format(second_smallest_value))
print("Second largest: {}".format(second_largest_value))

pd_num_points = pd_num_friends.size
pd_largest_value = pd_num_friends.max()
pd_smallest_value = pd_num_friends.min()
pd_sorted_values = pd_num_friends.sort_values()
print("Num points: {}".format(pd_num_points))
print("Largest value: {}".format(pd_largest_value))
print("Smallest value: {}".format(pd_smallest_value))
print("Sorted values: {}".format(pd_sorted_values))

def mean(x):
    return sum(x)/len(x)

print(mean(num_friends))

print(pd_num_friends.mean())

def median(v):
    n = len(v)
    sorted_v = sorted(v)
    midpoint = n // 2
    if n%2 == 1:
        return(sorted_v[midpoint])
    else:
        lo = midpoint - 1
        hi = midpoint
        return (sorted_v[lo] + sorted_v[hi])/2

print(median(num_friends))

print(pd_num_friends.median())

def quantile(x, p):
    p_index = int(p*len(x))
    return sorted(x)[p_index]

print(quantile(num_friends, .10))
print(quantile(num_friends, .25))
print(quantile(num_friends, .75))
print(quantile(num_friends, .90))

print(pd_num_friends.quantile(.10, 'nearest'))
print(pd_num_friends.quantile(.25, 'nearest'))
print(pd_num_friends.quantile(.75, 'nearest'))
print(pd_num_friends.quantile(.90, 'nearest'))

def mode(x):
    counts = Counter(x)
    max_count = (max(counts.values()))
    return [x_i for x_i, count in counts.items() if count ==
max_count]

print(mode(num_friends))

print(pd_num_friends.mode())

def data_range(x):
    return max(x) - min(x)

print(data_range(num_friends))

print(pd_num_friends.ptp())

def dot(v, w):
    return sum(v_i*w_i for v_i, w_i in zip(v, w))

def sum_of_squares(v):
    return dot(v, v)

def de_mean(x):
    x_bar = mean(x)
    return [x_i - x_bar for x_i in x]

def variance(x):
    n = len(x)
    deviations = de_mean(x)
    return sum_of_squares(deviations)/(n-1)

print(variance(num_friends))
print(pd_num_friends.var())

def standard_deviation(x):
    return math.sqrt(variance(x))

print(standard_deviation(num_friends))
print(pd_num_friends.std())

def covariance(x, y):
    n = len(x)
    return dot(de_mean(x), de_mean(y))/(n-1)

print(covariance(num_friends, daily_minutes))

pd_daily_minutes = pd.Series(daily_minutes)
print(pd_num_friends.cov(pd_daily_minutes))

def correlation(x, y):
    stdev_x = standard_deviation(x)
    stdev_y = standard_deviation(y)
    if stdev_x > 0 and stdev_y > 0:
        return covariance(x, y)/stdev_x/stdev_y
    else:
        return 0

print(correlation(num_friends, daily_minutes))
plt.scatter(num_friends, daily_minutes)
plt.title("Outlier")
plt.axis([0, 101, 0, 101])
plt.xlabel("# of friends")
plt.ylabel("# of minutes")
plt.show()

print(pd_num_friends.corr(pd_daily_minutes))

outlier = num_friends.index(100)
num_friends_good = [x
                   for i, x in enumerate(num_friends)
                   if i != outlier]
daily_minutes_good = [x
                     for i, x in enumerate(daily_minutes)
                     if i != outlier]
print(correlation(num_friends_good, daily_minutes_good))
plt.scatter(num_friends_good, daily_minutes_good)
plt.title("Outlier removed")
plt.axis([0, 101, 0, 101])
plt.xlabel("# of friends")
plt.ylabel("# of minutes")
plt.show()
```