In this notebook, we'll ask you to find numerical summaries for a certain set of data. You will use the values of what you find in this assignment to answer questions in the quiz that follows (we've noted where specific values will be requested in the quiz, so that you can record them.)

We'll also ask you to create some of the plots you have seen in previous lectures.

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import scipy.stats as stats
%matplotlib inline
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', 100)

path = "nhanes_2015_2016.csv"
```

In [2]:

```
# First, you must import the data from the path given above
df = pd.read_csv(path) # using pandas, read in the csv data found at the url defined by 'path'
```

In [7]:

```
# Next, look at the 'head' of our DataFrame 'df'.
df.head()

# If you can't remember a function, open a previous notebook or video as a reference
# or use your favorite search engine to look for a solution
```

Out[7]:

| | SEQN | ALQ101 | ALQ110 | ALQ130 | SMQ020 | RIAGENDR | RIDAGEYR | RIDRETH1 | DMDCITZN | DMDEDUC2 | DMDMARTL | DMDHHSIZ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 83732 | 1.0 | NaN | 1.0 | 1 | 1 | 62 | 3 | 1.0 | 5.0 | 1.0 | 2 |
| **1** | 83733 | 1.0 | NaN | 6.0 | 1 | 1 | 53 | 3 | 2.0 | 3.0 | 3.0 | 1 |
| **2** | 83734 | 1.0 | NaN | NaN | 1 | 1 | 78 | 3 | 1.0 | 3.0 | 1.0 | 2 |
| **3** | 83735 | 2.0 | 1.0 | 1.0 | 2 | 2 | 56 | 3 | 1.0 | 5.0 | 6.0 | 1 |
| **4** | 83736 | 2.0 | 1.0 | 1.0 | 2 | 2 | 42 | 4 | 1.0 | 4.0 | 3.0 | 5 |

How many rows can you see when you don't put an argument into the previous method?
How many rows can you see if you use an int as an argument?
Can you use a float as an argument?

In [8]:

```
# 5
# the number the int value passed in
# no - bombs
```

In [9]:

```
# Lets only consider the feature (or variable) 'BPXSY2'
bp = df['BPXSY2']
```

## Numerical Summaries

### Find the mean (note this for the quiz that follows)

```
In [16]: # What is the mean of 'BPXSY2'?
         bp_mean = bp.mean()
         bp_mean

Out[16]: 124.78301716350497
```

In the method you used above, how are the rows of missing data treated?
Are the excluded entirely? Are they counted as zeros? Something else? If you used a library function, try looking up the documentation using the code:

```
help(function_you_used)
```

For example:

```
help(np.sum)
```

#### .dropna()

To make sure we know that we aren't treating missing data in ways we don't want, lets go ahead and drop all the nans from our Series 'bp'

```
In [17]: bp = bp.dropna()
         bp.mean()

Out[17]: 124.78301716350497
```

## Find the:

- Median
- Max
- Min
- Standard deviation
- Variance

You can implement any of these from base python (that is, without any of the imported packages), but there are simple and intuitively named functions in the numpy library for all of these. You could also use the fact that 'bp' is not just a list, but is a pandas.Series. You can find pandas.Series attributes and methods here (https://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.Series.html)

A large part of programming is being able to find the functions you need and to understand the documentation formatting so that you can implement the code yourself, so we highly encourage you to search the internet whenever you are unsure!

## Example:

Find the difference of an element in 'bp' compared with the previous element in 'bp'.

```
In [12]: # Using the fact that 'bp' is a pd.Series object, can use the pd.Series method diff()
         # call this method by: pd.Series.diff()
         diff_by_series_method = bp.diff()
         # note that this returns a pd.Series object, that is, it had an index associated with it
         diff_by_series_method.values # only want to see the values, not the index and values
```

```
Out[12]: array([ nan,  16.,  -8., ...,  30., -40.,   8.])
```

In [13]:

```
# Now use the numpy library instead to find the same values
# np.diff(array)
diff_by_np_method = np.diff(bp)
diff_by_np_method
# note that this returns an 'numpy.ndarray', which has no index associated with it, and therefore ignores
# the nan we get by the Series method
```

Out[13]:

```
array([ 16.,  -8.,   2., ...,  30., -40.,   8.])
```

In [14]:

```
# We could also implement this ourselves with some looping
diff_by_me = [] # create an empty list
for i in range(len(bp.values)-1): # iterate through the index values of bp
    diff = bp.values[i+1] - bp.values[i] # find the difference between an element and the previous element
    diff_by_me.append(diff) # append to out list
np.array(diff_by_me) # format as an np.array
```

Out[14]:

```
array([ 16.,  -8.,   2., ...,  30., -40.,   8.])
```

## Your turn (note these values for the quiz that follows)

In [15]:

```
bp_median = bp.median()
bp_median
```

Out[15]:

```
122.0
```

In [18]:

```
bp_max = bp.max()
bp_max
```

Out[18]:

```
238.0
```

```
In [19]: bp_min = bp.min()
         bp_min

Out[19]: 84.0
```

```
In [20]: bp_std = bp.std()
         bp_std

Out[20]: 18.527011720294997
```

```
In [21]: bp_var = bp.var()
         bp_var

Out[21]: 343.2501632839482
```

```
In [22]: bp_std**2

Out[22]: 343.25016328394815
```

## How to find the interquartile range (note this value for the quiz that follows)

This time we need to use the scipy.stats library that we imported above under the name 'stats'

```
In [23]: bp_iqr = stats.iqr(bp)
         bp_iqr

Out[23]: 22.0
```

## Visualizing the data

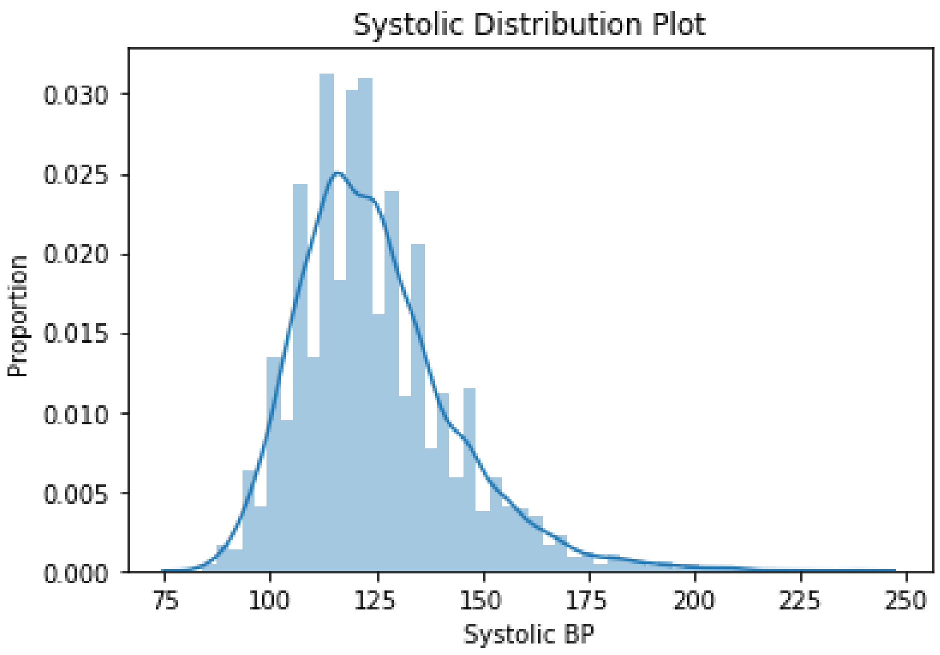
Next we'll use what you have learned from the *Tables, Histograms, Boxplots in Python* video

```
In [24]: # use the Series.describe() method to see some descriptive statistics of our Series 'bp'
         bp_descriptive_stats = bp.describe()
         bp_descriptive_stats
```

```
Out[24]: count    5535.000000
         mean      124.783017
         std        18.527012
         min        84.000000
         25%       112.000000
         50%       122.000000
         75%       134.000000
         max       238.000000
         Name: BPXSY2, dtype: float64
```

```
In [28]: # Make a histogram of our 'bp' data using the seaborn library we imported as 'sns'
         sns.distplot(bp.dropna()).set(title='Systolic Distribution Plot', xlabel='Systolic BP', ylabel='Proportion')
```

```
Out[28]: [Text(0,0.5,'Proportion'),
          Text(0.5,0,'Systolic BP'),
          Text(0.5,1,'Systolic Distribution Plot')]
```



Is your histogram labeled and does it have a title? If not, try appending

```
.set(title='your_title', xlabel='your_x_label', ylabel='your_y_label')
```

or just

```
.set(title='your_title')
```

to your graphing function

In [0]:

```
# Make a boxplot of our 'bp' data using the seaborn library. Make sure it has a title and labels!
# Just did
```