w2_assessment

December 4, 2021

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)
```

```
In [6]: # Next, look at the 'head' of our DataFrame 'df'.
        df.head(10)

        # 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[6]:     SEQN  ALQ101  ALQ110  ALQ130  SMQ020  RIAGENDR  RIDAGEYR  RIDRETH1  \
        0  83732     1.0     NaN     1.0       1         1        62         3
        1  83733     1.0     NaN     6.0       1         1        53         3
        2  83734     1.0     NaN     NaN       1         1        78         3
        3  83735     2.0     1.0     1.0       2         2        56         3
        4  83736     2.0     1.0     1.0       2         2        42         4
        5  83737     2.0     2.0     NaN       2         2        72         1
        6  83741     1.0     NaN     8.0       1         1        22         4
        7  83742     1.0     NaN     1.0       2         2        32         1
        8  83743     NaN     NaN     NaN       2         1        18         5
        9  83744     1.0     NaN     NaN       2         1        56         4

           DMDCITZN  DMDEDUC2  DMDMARTL  DMDHHSIZ   WTINT2YR  SDMVPSU  SDMVSTRA  \
        0       1.0       5.0       1.0         2  134671.37        1       125
```

```
1        2.0        3.0        3.0         1   24328.56         1        125
2        1.0        3.0        1.0         2   12400.01         1        131
3        1.0        5.0        6.0         1  102718.00         1        131
4        1.0        4.0        3.0         5   17627.67         2        126
5        2.0        2.0        4.0         5   11252.31         1        128
6        1.0        4.0        5.0         3   37043.09         2        128
7        2.0        4.0        1.0         4   22744.36         1        125
8        1.0        NaN        NaN         3   18526.16         2        122
9        1.0        3.0        3.0         1   20395.54         2        126

   INDFMPIR  BPXSY1  BPXDI1  BPXSY2  BPXDI2  BMXWT  BMXHT  BMXBMI  BMXLEG  \
0      4.39   128.0    70.0   124.0    64.0   94.8  184.5    27.8    43.3
1      1.32   146.0    88.0   140.0    88.0   90.4  171.4    30.8    38.0
2      1.51   138.0    46.0   132.0    44.0   83.4  170.1    28.8    35.6
3      5.00   132.0    72.0   134.0    68.0  109.8  160.9    42.4    38.5
4      1.23   100.0    70.0   114.0    54.0   55.2  164.9    20.3    37.4
5      2.82   116.0    58.0   122.0    58.0   64.4  150.0    28.6    34.4
6      2.08   110.0    70.0   112.0    74.0   76.6  165.4    28.0    38.8
7      1.03   120.0    70.0   114.0    70.0   64.5  151.3    28.2    34.1
8      5.00     NaN     NaN     NaN     NaN   72.4  166.1    26.2     NaN
9      1.19   178.0   116.0   180.0   114.0  108.3  179.4    33.6    46.0

   BMXARML  BMXARMC  BMXWAIST  HIQ210
0     43.6     35.9     101.1     2.0
1     40.0     33.2     107.9     NaN
2     37.0     31.0     116.5     2.0
3     37.7     38.3     110.1     2.0
4     36.0     27.2      80.4     2.0
5     33.5     31.4      92.9     NaN
6     38.0     34.0      86.6     NaN
7     33.1     31.5      93.3     2.0
8      NaN      NaN       NaN     2.0
9     44.1     38.5     116.0     2.0
```

How many rows can you see when you don't put an argument into the previous method? 5 How many rows can you see if you use an int (n) as an argument? n rows Can you use a float as an argument? No.

```
In [7]: # Lets only consider the feature (or variable) 'BPXSY2' (Systolic Blood pressure, meas
        bp = df['BPXSY2']
```

0.1 Numerical Summaries

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

```
In [10]: # What is the mean of 'BPXSY2'?
         bp_mean = bp.mean()
         bp_mean
```

```
Out[10]: 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? By default, they are exclude (NA/null values) when computing the result.

If you used a library function, try looking up the documentation using the code:

```
help(function_you_used)
```

For example:

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

```
In [16]: help(df.mean)
```

```
Help on method mean in module pandas.core.frame:

mean(axis=None, skipna=None, level=None, numeric_only=None, **kwargs) method of pandas.core.fra
    Return the mean of the values for the requested axis.

    Parameters
    ----------
    axis : {index (0), columns (1)}
        Axis for the function to be applied on.
    skipna : bool, default True
        Exclude NA/null values when computing the result.
    level : int or level name, default None
        If the axis is a MultiIndex (hierarchical), count along a
        particular level, collapsing into a Series.
    numeric_only : bool, default None
        Include only float, int, boolean columns. If None, will attempt to use
        everything, then use only numeric data. Not implemented for Series.
    **kwargs
        Additional keyword arguments to be passed to the function.

    Returns
    -------
    mean : Series or DataFrame (if level specified)
```

.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()
```

```
In [23]: bp.head()
```

```
Out[23]: 0    124.0
         1    140.0
         2    132.0
         3    134.0
         4    114.0
         Name: BPXSY2, dtype: float64
```

0.1.2 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

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!

```
In [20]: bp.describe()
```

```
Out[20]: 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 [21]: variance = bp.std()**2
         variance
```

```
Out[21]: 343.25016328394815
```

```
In [45]: # quartis
         quartil_25 = bp.quantile(q=0.25)
         quartil_50 = bp.quantile(q=0.5)
         quartil_75 = bp.quantile(q=0.75)
         print(quartil_25, quartil_50, quartil_75)
```

```
112.0 122.0 134.0
```

```
In [46]: # intervalor interquartil
         iqr = quartil_75 - quartil_25
         iqr

Out[46]: 22.0
```

0.1.3 Example:

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

```
In [22]: # 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
         diff_by_series_method.values # only want to see the values, not the index and values

Out[22]: array([ nan,  16.,  -8., ...,  30., -40.,   8.])
```

```
In [24]: # 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, a
         # the nan we get by the Series method

Out[24]: array([ 16.,  -8.,   2., ...,  30., -40.,   8.])
```

```
In [29]: # 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
             diff_by_me.append(diff) # append to out list
         np.array(diff_by_me) # format as an np.array

Out[29]: array([ 16.,  -8.,   2., ...,  30., -40.,   8.])
```

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

```
In [30]: bp_median = bp.median()
         bp_median

Out[30]: 122.0
```

```
In [31]: bp_max = bp.max()
         bp_max

Out[31]: 238.0
```

```
In [32]: bp_min = bp.min()
         bp_min
```

```
Out[32]: 84.0
```

```
In [33]: bp_std = bp.std()
         bp_std
```

```
Out[33]: 18.527011720294997
```

```
In [34]: bp_var = bp.var()
         bp_var
```

```
Out[34]: 343.2501632839482
```

0.1.5 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 [44]: bp_iqr = stats.iqr(bp)
         bp_iqr
```

```
Out[44]: 22.0
```

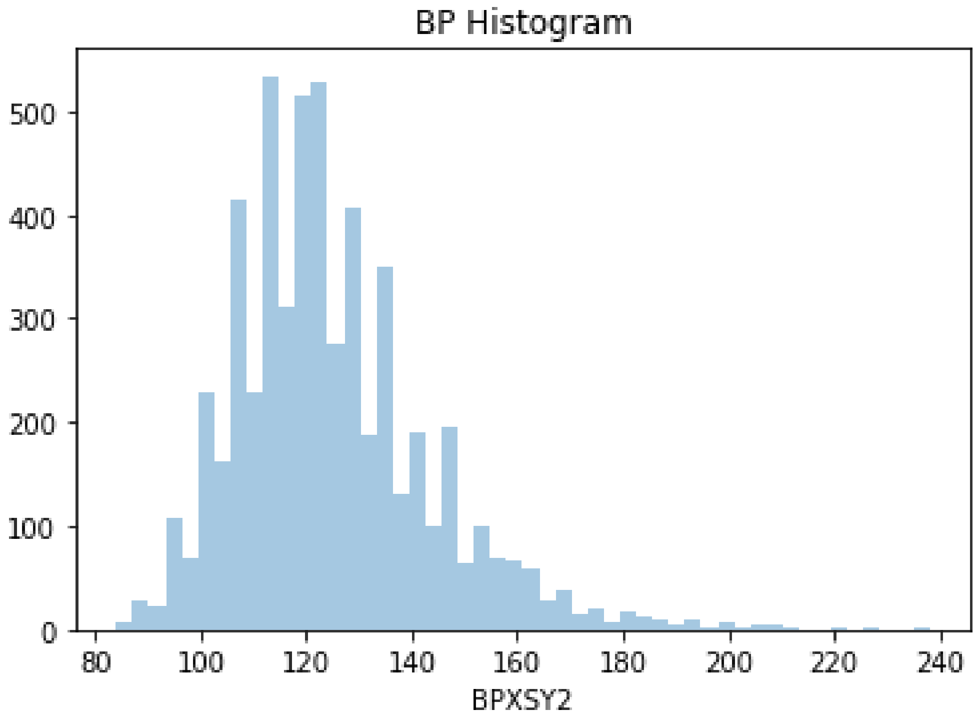
0.2 Visualizing the data

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

```
In [48]: # use the Series.describe() method to see some descriptive statistics of our Series '
         bp_descriptive_stats = bp.describe()
         bp_descriptive_stats
```

```
Out[48]: 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 [50]: # Make a histogram of our 'bp' data using the seaborn library we imported as 'sns'
         sns.distplot(bp, kde=False).set_title("BP Histogram")
         plt.show()
```



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

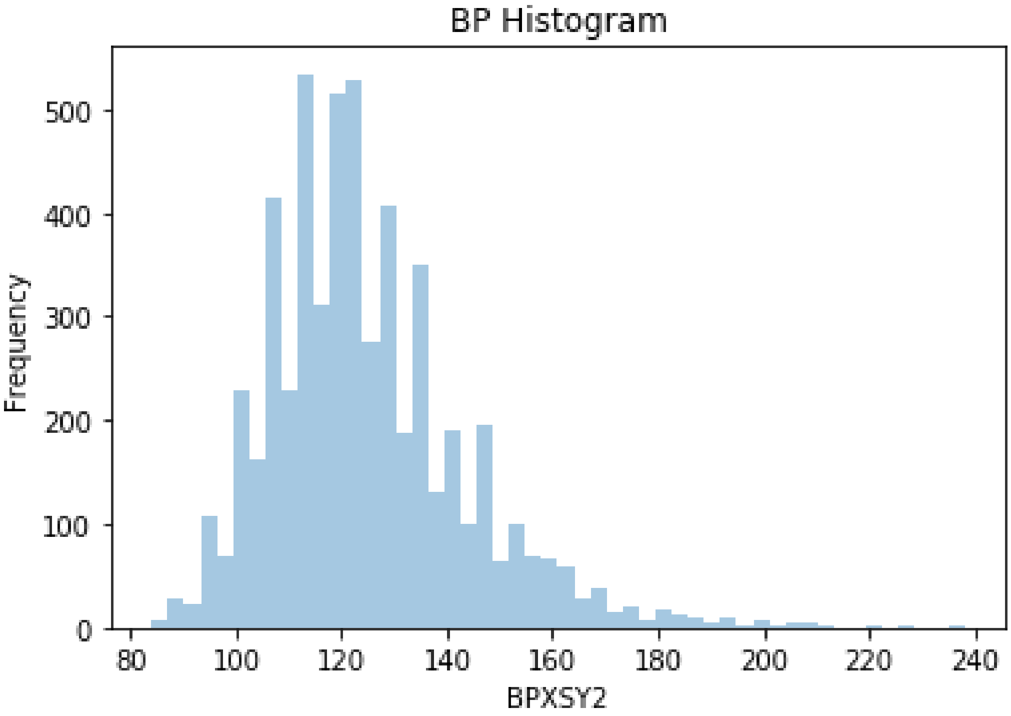
```
.set(title='your_title', xlabel='your_x_label', ylabel='your_y_label')
```

or just

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

to your graphing function

```
In [52]: # Make a boxplot of our 'bp' data using the seaborn library. Make sure it has a title
         sns.distplot(bp, kde=False).set(title="BP Histogram", ylabel='Frequency', xlabel="BPX
         plt.show()
```


BP Histogram
Frequency
0
100
200
300
400
500
80
100
120
140
160
180
200
220
240
BPXSY2