w2_assessment

October 18, 2022

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 [25]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#import scipy.stats as stats
%matplotlib inline
```

In [29]:
```
# First, you must import the data from the path given above
df = pd.read_csv('nhanes_2015_2016.csv')
df.dropna(inplace=True)
df = pd.read_csv('nhanes_2015_2016.csv')
```

In [32]:
```
# Next, look at the 'head' of our DataFrame 'df'.
df.head().T

# 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[32]:

	0	1	2	3	4
SEQN	83732.00	83733.00	83734.00	83735.0	83736.00
ALQ101	1.00	1.00	1.00	2.0	2.00
ALQ110	NaN	NaN	NaN	1.0	1.00
ALQ130	1.00	6.00	NaN	1.0	1.00
SMQ020	1.00	1.00	1.00	2.0	2.00
RIAGENDR	1.00	1.00	1.00	2.0	2.00
RIDAGEYR	62.00	53.00	78.00	56.0	42.00
RIDRETH1	3.00	3.00	3.00	3.0	4.00
DMDCITZN	1.00	2.00	1.00	1.0	1.00
DMDEDUC2	5.00	3.00	3.00	5.0	4.00
DMDMARTL	1.00	3.00	1.00	6.0	3.00
DMDHHSIZ	2.00	1.00	2.00	1.0	5.00
WTINT2YR	134671.37	24328.56	12400.01	102718.0	17627.67
SDMVPSU	1.00	1.00	1.00	1.0	2.00

```
SDMVSTRA     125.00    125.00    131.00     131.0    126.00
INDFMPIR       4.39      1.32      1.51       5.0      1.23
BPXSY1       128.00    146.00    138.00     132.0    100.00
BPXDI1        70.00     88.00     46.00      72.0     70.00
BPXSY2       124.00    140.00    132.00     134.0    114.00
BPXDI2        64.00     88.00     44.00      68.0     54.00
BMXWT         94.80     90.40     83.40     109.8     55.20
BMXHT        184.50    171.40    170.10     160.9    164.90
BMXBMI        27.80     30.80     28.80      42.4     20.30
BMXLEG        43.30     38.00     35.60      38.5     37.40
BMXARML       43.60     40.00     37.00      37.7     36.00
BMXARMC       35.90     33.20     31.00      38.3     27.20
BMXWAIST     101.10    107.90    116.50     110.1     80.40
HIQ210         2.00       NaN      2.00       2.0      2.00
```

0.1 Frequency Tables

The value_counts method can be used to determine the number of times that each distinct value of variable occurs in a data set. In statistical terms, this is the "Frequency Distribution" of the variable.

The number 1, 2, 3, 4, 5, and 9 seen below are integer codes for the 6 possible non-missing values of the DMDEDUC2 variable.

```
In [35]: df.DMDEDUC2.value_counts()
```

```
Out[35]: 4.0    1621
         5.0    1366
         3.0    1186
         1.0     655
         2.0     643
         9.0       3
         Name: DMDEDUC2, dtype: int64
```

Note that the 'value_counts' method excludes missing values. It is confirmed below that the number of observations of DMDEDUC2 are 5474 rows, and comparing this value in the data set, which is 5735. There are 261 missing values for this variable (5735-5474

```
In [38]: print(df.DMDEDUC2.value_counts().sum())
         print(1621+1366+1186+655+643+3) #manually sum the frequencies
         print(df.shape)
```

```
5474
5474
(5735, 28)
```

Another way to obtain this result is to locate all the missing values in the daya sey using 'isnull' Pandas function, and count the number of such locations.

```
In [40]: pd.isnull(df.DMDEDUC2).sum()
```

```
Out[40]: 261

In [53]: df.head(2)

Out[53]:     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

            DMDCITZN  DMDEDUC2  ...  BPXSY2  BPXDI2  BMXWT  BMXHT  BMXBMI  BMXLEG  \
         0       1.0       5.0  ...   124.0    64.0   94.8  184.5    27.8    43.3
         1       2.0       3.0  ...   140.0    88.0   90.4  171.4    30.8    38.0

            BMXARML  BMXARMC  BMXWAIST  HIQ210
         0     43.6     35.9     101.1     2.0
         1     40.0     33.2     107.9     NaN

         [2 rows x 28 columns]
```

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

0.2 Numerical Summaries

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

```
In [45]: # What is the mean of 'BPXSY2'?
         bp1 = bp.dropna()
         bp_mean = np.mean(bp1)
         bp_mean

Out[45]: 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 [25]: bp = bp.dropna()
```

0.2.2 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!

0.2.3 Example:

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

```
In [46]: # 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[46]: array([ nan,  16.,  -8., ...,  30., -40.,   8.])

In [47]: # 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[47]: array([ 16.,  -8.,   2., ...,  30., -40.,   8.])

In [28]: # 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[28]: array([ 16.,  -8.,   2., ...,  30., -40.,   8.])
```

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

```
In [29]: bp_median = np.median(bp)
         bp_median
```

```
Out[29]: 122.0

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

Out[31]: 238.0

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

Out[32]: 84.0

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

Out[33]: 18.525338021233786

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

Out[34]: 343.1881488009701
```

0.2.5 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'

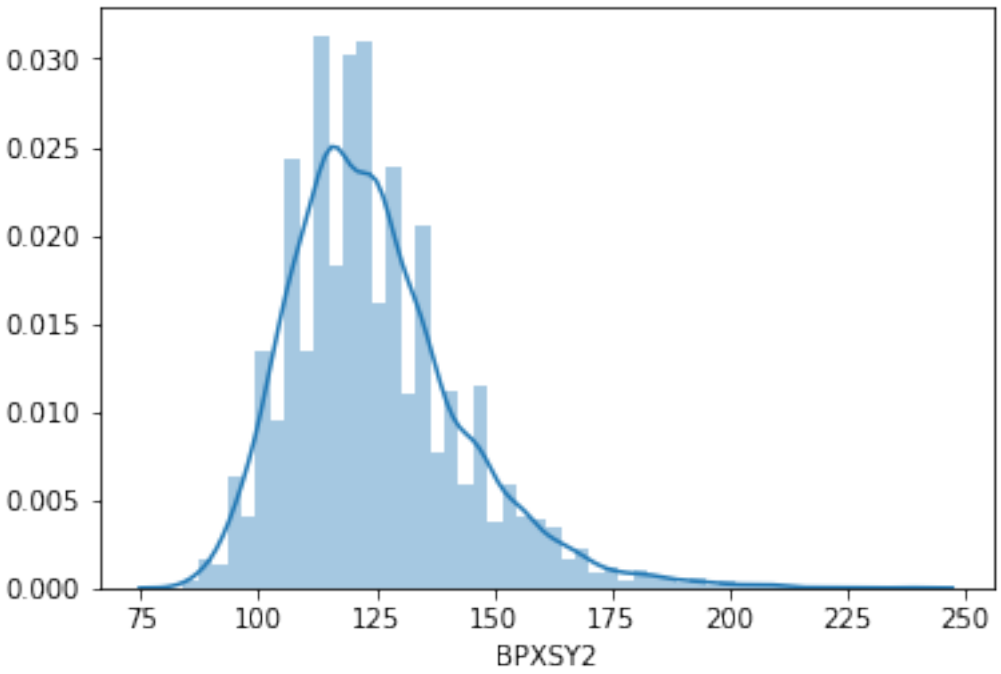
```
In [35]: bp_iqr = stats.iqr(bp)
         bp_iqr

Out[35]: 22.0
```

0.3 Visualizing the data

A quick way to get a set of numerical summaries for a quantitative variable is with the "describe" data frame method. Below we demonstrate how to do this using the second systolic blood pressure measurements variable(BPXSY2). As with many surveys, some data values are missing, so we explicitly drop the missing cases using the 'dropna' method before generating the summaries.

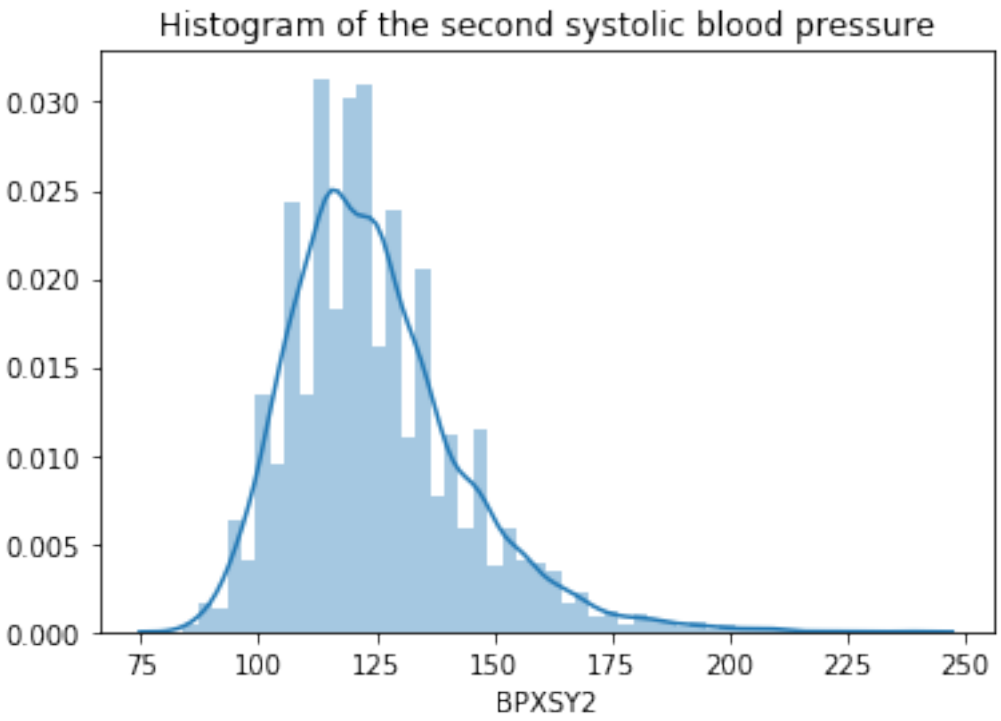
```
In [51]: # use the Series.describe() method to see some descriptive statistics of our Series '
         df.BPXSY2.dropna().describe()

Out[51]: 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 [52]: # Make a histogram of our 'bp' data using the seaborn library we imported as 'sns'
         sns.distplot(df.BPXSY2.dropna())

Out[52]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8135deef28>
```



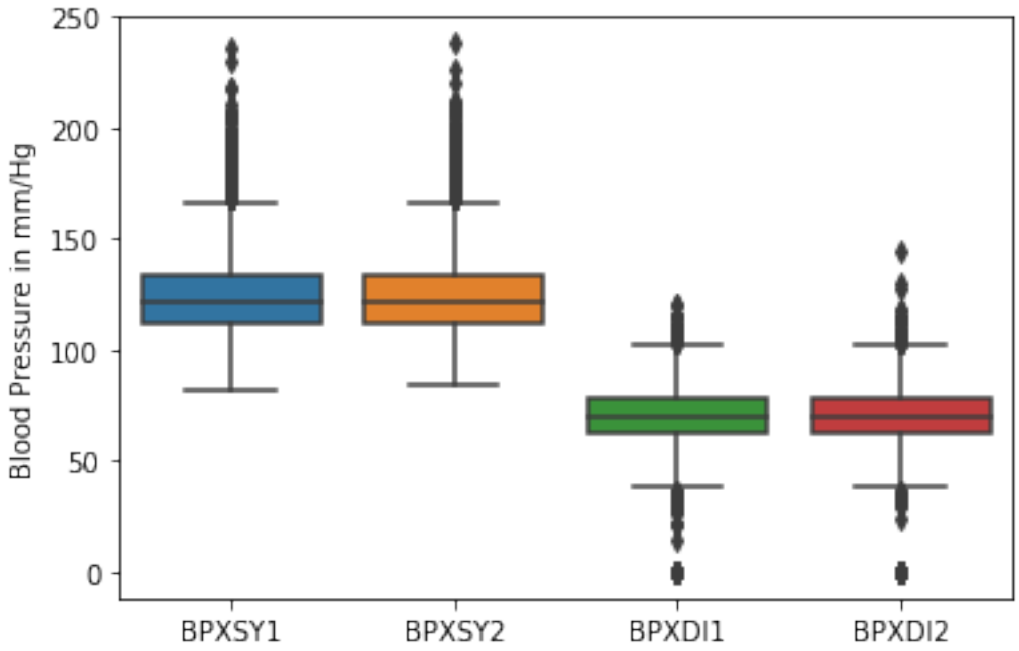
Another way to visualize our data as histogram

```
In [56]: sns.distplot(df.BPXSY2.dropna()).set(title="Histogram of the second systolic blood pr

Out[56]: [Text(0.5,1,'Histogram of the second systolic blood pressure')]
```



To compare several distributions, we can use side-by-side boxplots. Below we compare the distributions of the first and second systolic blood pressure measurements (BPXSY1 and BPXSY2); The first and second blood pressure measurements (BPXDI1 and BPXDI2).

```
In [48]: # Make a boxplot of our 'bp' data using the seaborn library. Make sure it has a title
         bp = sns.boxplot(data=df.loc[:,["BPXSY1", "BPXSY2","BPXDI1", "BPXDI2"]])
         _=bp.set_ylabel("Blood Pressure in mm/Hg")
```



As it can be seen, diastolic measurements are substantially lower than systolic measurements.