DASC 500: Introduction to Data Analytics

Assignment #3

16 Feb 2020

Each of the questions below involve the `advertising` data set. The data set has been provided as part of this assignment in Canvas and may also be accessed by clicking **this link**.

Problem 1

Focusing on the `newspaper` column, show how you would generate a random sample of size N - where $N \in \mathbb{I}_{1,200}$ (N is any positive integer between 1 and 200).

If you use Python, your answer should include the code used to generate the sample, if you use Excel provide a list of the commands used to generate the sample.

Using Python, we generate a random sample by first importing the necessary libraries and the `advertising` data set.

```
import random
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import scipy as sp
import statsmodels.api as sm
import statistics as st
```

```
data_url = "http://faculty.marshall.usc.edu/gareth-james/ISL/Advertising.csv"
```

```
data_url = r.data_url
```

```
df2 = pd.read_csv(data_url,
                  usecols = { "sales","TV","radio","newspaper" })
```

With the data imported we can use the function below, called `samp()` to generate a random sample of size N

```
# define function to extract the sample
def samp(df, N):

  rows = random.sample(range(df2.shape[0]), N)
  samp = df2.iloc[rows]

  return(samp)
```

This question was included to demonstrate the importance (and challenge) of ensuring reproducibility in your work. It's important to ensure that, if required, someone can verify the methods you used to produce one or more results. Those of you who used Excel saw this is made more challenging when using GUI-based software where you have to type out every step. When you use command line tools reproducibility is much easier in that you just have to provide the commands used.

Problem 2

Use the method described in Exercise 1 to generate a sample of size $N = 30$ - provide a histogram of the observations.

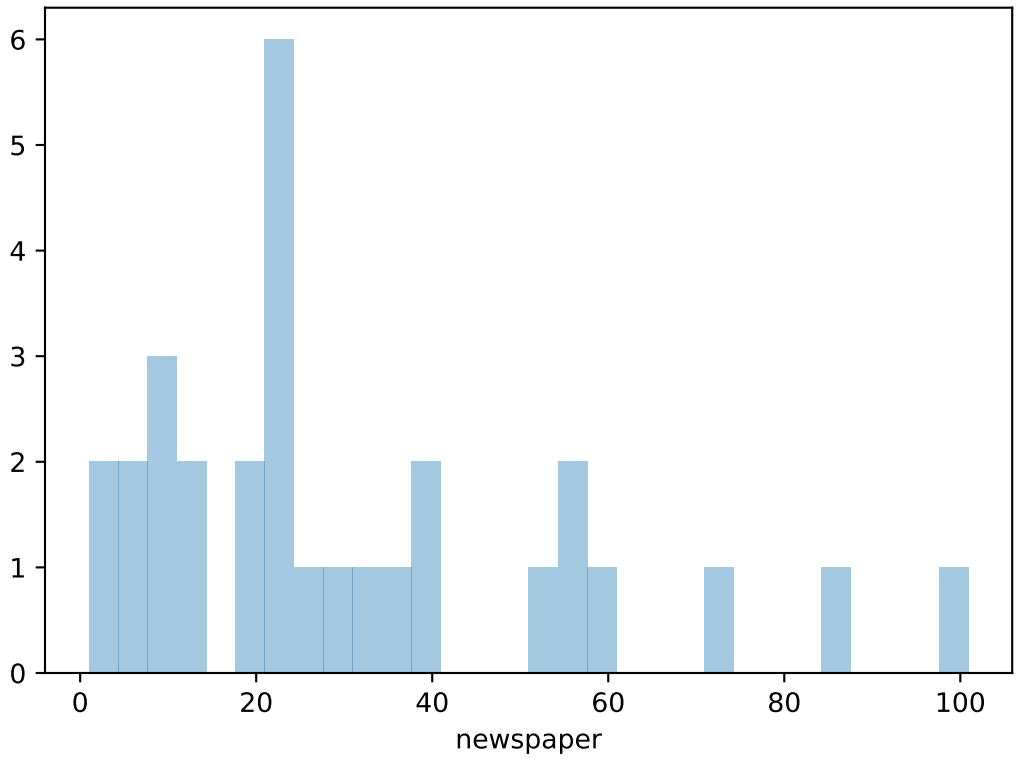
With the function `samp()` defined, we now use it to generate the random sample and store it as an object named `out`

```
# use the function for N = 5
out = samp(df = df2, N = 30)
```

Then, we use the `distplot()` function from the `seaborn` library to produce the histogram plot.

```
plot = sb.distplot(out["newspaper"],
                   bins = int(out.shape[0]),
                   kde = False)

plt.show(plot)
```



Problem 3

Compute the mean and variance of the sample you generated "by hand" (i.e. type up the formulae used to compute these values yourself in the software tool of choice). Then, use an implicit function to compute these values.

This question is essentially asking you to compute the mean and variance yourself and then

compare your values to those that are returned by the native Python/Excel function that has been provided to compute these for you.

The results should essentially be the same plus/minus some rounding error. If this is not the case can you suggest a reason why?

The code in the chunk below computes the mean of the `newspaper` column from our random sample and then compares this value to the mean computed using the `numpy` and `statistics` modules. The results show that all three return the same value for the mean.

```
my_mean = (1/out.shape[0]) * sum(out["newspaper"])
np_mean = np.mean(out["newspaper"])
st_mean = st.mean(out["newspaper"])

print("my mean    = "+str(my_mean))
```

```
## my mean    = 30.96333333333333
```

```
print("numpy mean = "+str(np_mean))
```

```
## numpy mean = 30.963333333333328
```

```
print("stats mean = "+str(st_mean))
```

```
## stats mean = 30.963333333333335
```

The code in the chunk below computes the variance of the `newspaper` column from our random sample and then compares this value to the variance computed using the `numpy` and `statistics` modules. The results show that all three do not return the same value for the variance. We see that the variance that was computed by hand is the same as the value returned by the `statistics` library as both assume that the adjusted sample variance is desired. By contrast the `numpy` implementation assumes that the un-adjusted sample variance is desired

```
my_var = 1/(out.shape[0] - 1) * np.sum((out["newspaper"] - np.mean(out["newspaper"]))**2)
np_var = np.var(out["newspaper"])
st_var = st.variance(out["newspaper"])

print("my variance    = "+str(my_var))
```

```
## my variance    = 617.6265402298851
```

```
print("numpy variance = "+str(np_var))
```

```
## numpy variance = 597.038988888889
```

```
print("stats variance = "+str(st_var))
```

```
## stats variance = 617.626540229885
```

Problem 4

Using the values you computed in Exercise 3 construct a 95% confidence interval for the mean μ based on the sample you generated.

Because the sample you generated contains 30 observations, we can invoke the Central Limit Theorem.

The desired confidence interval is computed using the code in the chunk below

```
# import the normal module from scipy
from scipy.stats import norm

# Set the alpha level
alpha = 0.05

# Find the sample standard deviation
my_std = np.sqrt(my_var)

# Find z value at alpha / 2
z = norm.ppf(1 - alpha / 2, loc = 0, scale = 1)

# compute and print the confidence limits
mu_l = my_mean - z * my_std / np.sqrt(len(out))
mu_h = my_mean + z * my_std / np.sqrt(len(out))

mu_l, my_mean, mu_h
```

```
## (22.070289383404663, 30.96333333333333, 39.856377283262)
```

Problem 5

Suppose that we knew that the true value of the mean $\mu_0 = 30.55$, perform a 0.05-level hypothesis where

$$H_O : \mu = \mu_0$$

$$H_A : \mu \neq \mu_0.$$

Based on the sample you generated what conclusion do you reach? Be sure to include the value of your test statistic W in your answer.

```
alpha = 0.05
mu_0 = 31.55

W = (my_mean - mu_0) / (my_std) / np.sqrt(out.shape[0])

z = norm.ppf(alpha / 2, loc = 0, scale = 1)

W ; z ; abs(W) <= z
```

```
## -0.004309906124647981
## -1.9599639845400545
## False
```

Problem 6

Repeat the steps performed in Exercises 2-5 three times, that is

- **Generate three random samples where $N = 30$**
- **Compute the means and variances of the sample**
- **Construct a 95% confidence interval on the mean μ for each sample**
- **Perform a 0.05-level hypothesis where $H_O : \mu = \mu_0$ and $H_A : \mu \neq \mu_0$ for each sample**

For each of the three iterations provide the following results:

- **The value of the sample mean $\bar{X}_n = \hat{\mu}$**
- **The value of the adjusted sample variance $s^2 = \hat{\sigma}$**
- **The upper and lower limits of the 95% confidence interval**
- **The value of the test statistic W**

Whenever you need to repeat a process - that's a clear indication that you should write a function! Below I create an example function, called `sampler()` that can be used to repeat the process described in the previous problems. To use this function we need to paste it into our Python console to define it and make it available for use.

Note that when defining a function in Python every line after the colon must be indented. Python is one of several languages where white space is important. The way that knows where the function ends is by finding the line where you stop indenting.

```
def sampler(df, colname,N, alpha, mu_0):

  rows = random.sample(range(df.shape[0]), N)
  samp = df.iloc[rows]
```

```
  col = samp[colname]

  my_mean = (1/len(col)) * sum(col)
  my_var = 1/(len(col) - 1) * np.sum((col - np.mean(col))**2)
  my_std = np.sqrt(my_var)

  # Find z value at alpha / 2
  z = norm.ppf(1-alpha / 2, loc = 0, scale = 1)

  mu_l, mu_h = my_mean - z * my_std / np.sqrt(len(col)), my_mean + z * my_std / np.sqrt(len(col))

  W = (my_mean - mu_0) / (my_std) / np.sqrt(len(col))

  Test = abs(W) <= z

  out = dict({"mean": my_mean,
              "var": my_var,
              "std_dev":my_std,
              "mu_low": mu_l,
              "mu_high": mu_h,
              "Test stat": W,
              "Reject H_O": Test})

  return(out)
```

With the function defined, we need only run the function three times and display the results for each run.

```
res1 = sampler(df = df2,colname="newspaper",N = 30, alpha = 0.05, mu_0= 31.55)
res2 = sampler(df = df2,colname="newspaper",N = 30, alpha = 0.05, mu_0= 31.55)
res3 = sampler(df = df2,colname="newspaper",N = 30, alpha = 0.05, mu_0= 31.55)
```

	mean	var	std_dev	mu_low	mu_high	Test stat	Reject H_O
Res1	30.66333	534.6672	23.12287	22.38907	38.93759	-0.0070010	1
Res2	31.18667	757.2391	27.51798	21.33967	41.03367	-0.0024106	1
Res3	33.16667	492.7313	22.19755	25.22352	41.10981	0.0132970	1

Problem 7

Repeat the steps performed in the Exercise 6 three times, but this time set $N = 100$

For this problem we can simply use the `sampler()` function again, noting that we need to change the value of the argument N to 100. This is shown below along with the results.

```
res1b = sampler(df = df2,colname="newspaper",N = 100, alpha = 0.05, mu_0= 31.55)
res2b = sampler(df = df2,colname="newspaper",N = 100, alpha = 0.05, mu_0= 31.55)
res3b = sampler(df = df2,colname="newspaper",N = 100, alpha = 0.05, mu_0= 31.55)
```

	mean	var	std_dev	mu_low	mu_high	Test stat	Reject H_O
Res1b	31.380	417.6228	20.43582	27.37465	35.38535	-0.0008319	1
Res2b	33.027	553.0091	23.51614	28.41792	37.63608	0.0062808	1
Res3b	30.398	480.3226	21.91626	26.10249	34.69351	-0.0052564	1

Problem 8

What are your observations based on the results from Exercises 6 and 7?

Observing the values produced in Problem 6 and those produced in Problem 7 we see that much more consistent results were produced when the sample size is larger. Specifically, we see that the value of the mean and upper and lower limits of the confidence intervals fluctuated a lot when $N = 30$ but were more consistent when $N = 100$.