# Pset0

February 1, 2018

## 1 Problem Set 0

## 2 Due Date: Not Due.

Welcome to the computation platform for 6.s077! If you are accessing this on mit-6s077.mit.edu then you have managed to get yourself set up on the platform. We provide Problem Set 0 for you to get comfortable with the platform (JupyterHub) and Jupyter notebooks.

We recommend you step through all the instructions below (line by line) even though you may already be familiar with data analysis in Python and/or Jupyter notebooks.

First note that we only support Python 3 for this platform.

Importing Libraries #This platform comes with support for common data analysis and machine learning libraries (in Python). These include the following which you will be using extensively through this course: Numpy: http://www.numpy.org/ Pandas: https://pandas.pydata.org/ Scipy: https://www.scipy.org/ Scikit-Learn: http://scikit-learn.org/stable/ Statsmodels: http://scikit-learn.org/stable/ Matplotlib: https://matplotlib.org/

0. We start by printing a quick "Hello, World!". Please execute the Run command (from the menu above this notebook) on the cell below to ensure you see no errors. You can 'select' a cell by clicking on it and then hit the Run button in the menu above. Errors, or any expected output should show up immediately below the cell. In the case below, you should see a "Hello, World!" printed below the cell.

1. You can import the libraries as shown below. Hit Run to ensure the libraries are imported successfully and without errros.

```
In [195]: print("Hello, World!")
```

```
Hello, World!
```

```
In [196]: %matplotlib inline
          from matplotlib import pyplot as plt
          import numpy as np
          import pandas as pd
          import statsmodels.api as sm
          from scipy import stats

          print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

2. Print what versions of the pandas (pd) and numpy (np) are installed on this system. For numpy, see the example below and repeat for pandas.

```
In [197]: print("Numpy {}".format(np.__version__))
          print("Pandas {}".format(pd.__version__))

          # For those unfamiliar with Python features, .format replaces {} instances
          # with the provided arguments and is very useful for easy to read formatting
          # of strings
```

```
Numpy 1.12.1
Pandas 0.19.2
```

3. You can also import a Python file within the project directory. You have been provided with a Python script (utils.py) in the same directory as this assignment notebook. Import utils and call the fib_n() function with argument n to return the n'th Fibonnaci number. Note that we restrict n <= 10 since this is function is for illustration only. Print the the result for all $0 \leq n \leq 10$ and verify that the Fibonacci number sequence is correct.

```
In [198]: import utils

          for n in range(11):
              print("The {}th Fibonacci number is {}.".format(n, utils.fib_n(n)))
```

```
The 0th Fibonacci number is 0.
The 1th Fibonacci number is 1.
The 2th Fibonacci number is 1.
The 3th Fibonacci number is 2.
The 4th Fibonacci number is 3.
The 5th Fibonacci number is 5.
The 6th Fibonacci number is 8.
The 7th Fibonacci number is 13.
The 8th Fibonacci number is 21.
The 9th Fibonacci number is 34.
The 10th Fibonacci number is 55.
```

4a. You are provided with csv file with 100 observations of daily commute times (in minutes) for an individual. The data file is called "data.csv". Read the csv as a pandas dataframe object by using the read_csv() function in pandas: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html

```
In [199]: df = pd.read_csv("data.csv")

          print("Dataframe loaded successfully!")
```

```
Dataframe loaded successfully!
```

4b. Print the total total number of observations in the dataframe. Confirm that it is 100.

```
In [200]: print("The dataframe contains {} lines of data.".format(len(df)))
```

```
The dataframe contains 100 lines of data.
```

4c. Print the column names in the dataframe. Confirm that there is only one column.

```
In [201]: df_cols = df.columns
          n_cols = len(df_cols)
          col_names = df_cols.values # list of column names
          col_names_str = ",".join(col_names)
          # joined column names into a string where each name is separated by ","
          # since this example has only one column we will not see "," in the result
          # but join() is useful when you want to nicely print arrays of length > 1

          print("The dataframe contains {} columns.".format(n_cols))
          print("The column names are: {}.".format(col_names_str))
```

```
The dataframe contains 1 columns.
The column names are: commute_time.
```

**4d. Write a function called mean(np_array) which computes and then returns the sample average of a numpy array.** Call this function on the array of observations for commute_time and print the mean.

```
In [202]: def mean(np_array):
              n = len(np_array)
              return sum(np_array) / n
          # For those accustomed to Python 2.7, '/' in Python 3.0 returns a float while '//'
          # returns the integer part e.g. 3 / 2 = 1.5 but 3 // 2 = 1

          print("Implemented mean()!")
```

```
Implemented mean()!
```

4e. Now, print the sample average of the commute_time observations using numpy's mean() function. Verify that your own mean() function produces the same sample average as numpy's mean().

```
In [203]: times = df.commute_time.values
          print("Our mean function returns {}.".format(mean(times)))
          print("Numpy's mean function returns {}.".format(np.mean(times)))
```

```
Our mean function returns 15.066961950150123.
Numpy's mean function returns 15.066961950150128.
```

4f. Write a function called sd(np_array) which computes and then returns the population standard deviation of a numpy array. Call this function on the array of observations for commute_time and print the population standard deviation.

```
In [204]: def sd(np_array):
              n = len(np_array)
              avg = mean(np_array)
              deviations = [(n - avg) ** 2 for n in np_array]
              # This is known as a 'list comprehension', read about it below
              # https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions
              sum_deviations = sum(deviations)
              return np.sqrt(sum_deviations / n)

          print("Implemented sd()!")
```

```
Implemented sd()!
```

4g. Now, print the population standard deviation of the commute_time observations using numpy's std() function. Confirm that your own std() function produces the same population average as numpy's std(). Note: even though we do not require it here, you should read the np.std() documentation to understand how to compute the sample (unbiased) standard deviation (instead of the population standard deviation).

```
In [205]: print("Our sd function returns {}.".format(sd(times)))
          print("Numpy's sd function returns {}.".format(np.std(times)))
```

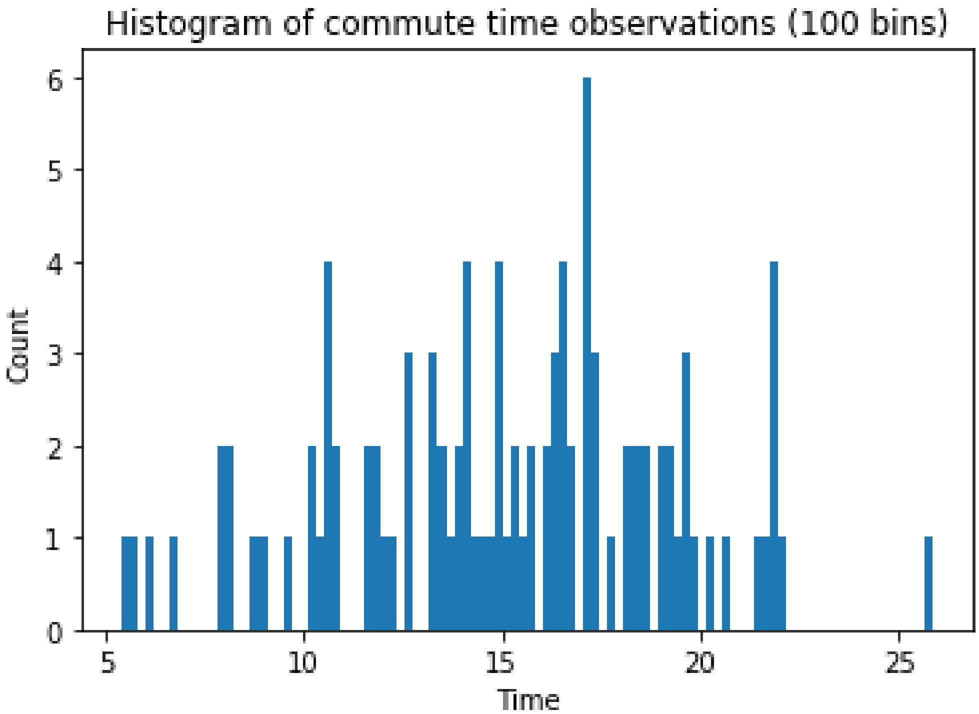
```
Our sd function returns 4.148359733799695.
Numpy's sd function returns 4.148359733799694.
```

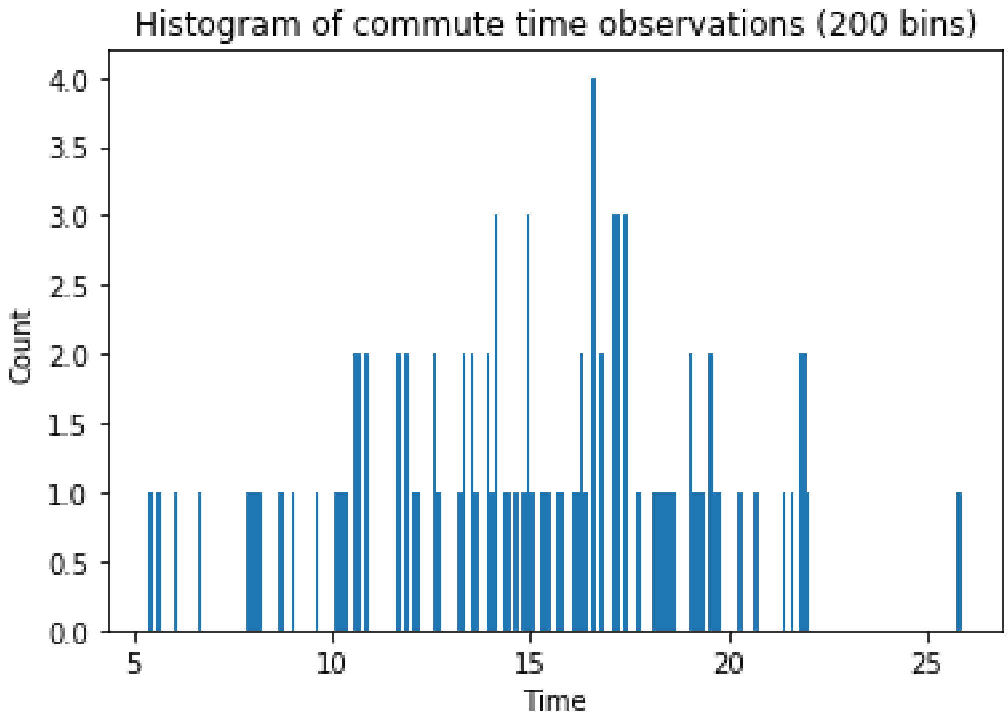
4h. Produce a histogram of the commute_time observations using the hist() function provided by matplotlib.pyplot: https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.hist.html Produce two histograms: one with 100 bins and 20 bins. Title your histograms appropriately.

```
In [206]: TITLE_F = "Histogram of commute time observations ({} bins)"
          TITLE_1 = TITLE_F.format(100)
          TITLE_2 = TITLE_F.format(200)

          plt.figure(1)
          plt.hist(times, bins = 100);
          plt.title(TITLE_1);
          plt.xlabel("Time");
          plt.ylabel("Count");
```

```
plt.figure(2)
plt.hist(times, bins = 200);
plt.title(TITLE_2);
plt.xlabel("Time");
plt.ylabel("Count");
```






5a. We wish to generate a sample consisting of n randomly drawn records from our data set. Write a function sample_n(np_array, n) which produces a random sample of length n (with replacement) of from the numpy array (np_array). In each step of the iteration you draw an index at random, and all records in np_array are equally likely to be picked. This is called 'sampling with replacement': each record may get selected more than once. At each drawing, you need to pick a random index (in the range 0: n-1, both inclusive) and pull the corresponding entry from the data array. You may find numpy's random.randint(n) to be useful in randomly selecting indices between 0 and n-1. For more info check: https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.random.randint.html Produce a random sample of size N=50 and then report the mean and population standard deviation of the selected sample. Use the commute_time array for input data

```
In [207]: import random

          def sample_n(np_array, n):
              return [times[random.randrange(n)] for i in range(n)]

          N = 50
          samples = sample_n(times, N)
          sample_mean = mean(samples)
          sample_sd = sd(samples)
          print("The sample mean is {}".format(sample_mean))
          print("The sample sd is {}".format(sample_sd))
```

```
The sample mean is 15.849075679304088
```

```
The sample sd is 3.2865321019142226
```

5b. Now write a function repeat(np_array, N, n) which will do the following: It calls sample_n() N times, each producing a sample (with replacement) of length n from np_array. For each of the N samples, compute the mean and population standard deviation and store them in two arrays. Return both the mean and standard deviation arrays (each of length = N). Call the function repeat() with N = 500 and n = 100, and np_array = commute_time array. Store the array of means as mean_samples and the array of standard deviations as sd_samples.

```
In [208]: def sample_mean_and_sd(times, n):
              sample = sample_n(times, n)
              return mean(sample), sd(sample)

          def repeat(np_array, N, n):
              sample_stats = [sample_mean_and_sd(times, n) for i in range(N)]
              # each element of sample_stats is a tuple containing
              # the mean and sd for the corresponding sample
              mean_samples = [el[0] for el in sample_stats] # first elements are the means
              sd_samples = [el[1] for el in sample_stats] # second elements are the sds
              return mean_samples, sd_samples


          # We'll also provide a more straightforward solution below that ignores list
          # comprehensions. There is no need to use all the neat features Python
          # provides but some familiarity can greatly improve the readability and
          # debugability of your code at times :) For this example the task is simple
          # enough so there is no obvious preference for either solution

          def simple_repeat(np_array, N, n):
              mean_samples = []
              sd_samples = []
              for i in range(N):
                  sample = sample_n(times, n)
                  mean_samples.append(mean(sample))
                  sd_samples.append(sd(sample))
              return mean_samples, sd_samples

          mean_samples, sd_samples = repeat(times, 500, 100)
```

5c. Print the mean of mean_samples and sd_samples.

```
In [209]: print("The mean of mean_samples is {}.".format(mean(mean_samples)))
          print("The mean of sd_samples is {}.".format(mean(sd_samples)))
```

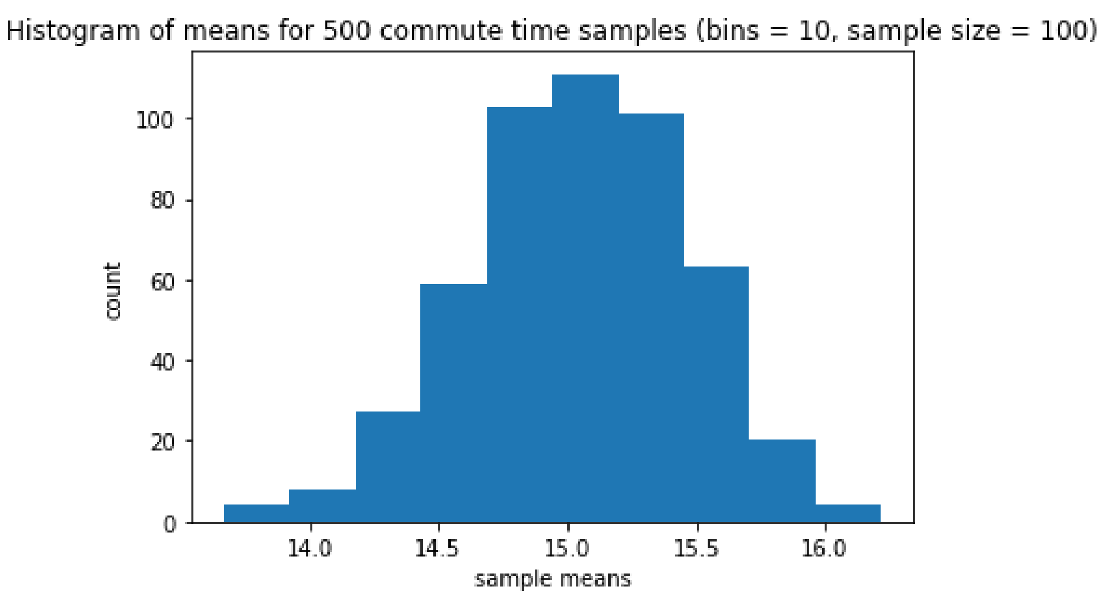
```
The mean of mean_samples is 15.049273548401.
The mean of sd_samples is 4.142268168034169.
```

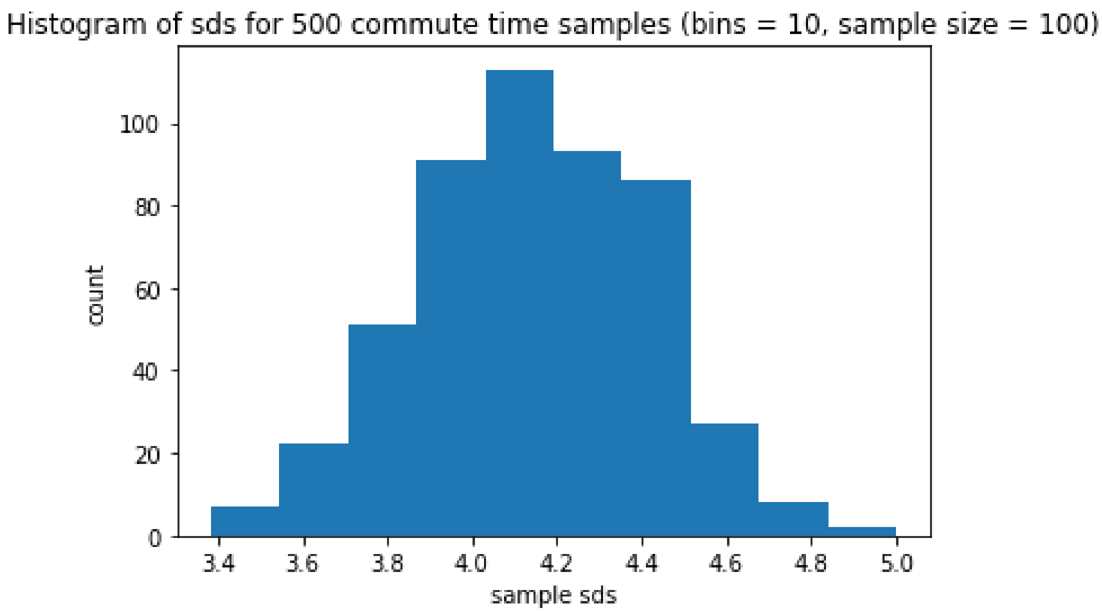
5d. Produce separate histograms of the mean_samples and sd_samples. Title your plots appropriately. Set the number of bins = 10.

In [210]:
```
SAMPLE_TITLE_F = ("Histogram of {stat} for {n_samples} commute time "
                  "samples (bins = {bins}, sample size = {samp_size})")
MEAN_TITLE = SAMPLE_TITLE_F.format(stat="means", n_samples=500,
                                   bins=10, samp_size=100)
SD_TITLE = SAMPLE_TITLE_F.format(stat="sds", n_samples=500,
                                 bins=10, samp_size=100)
# You can provide labels inside {} to provide names for your formatting
# arguments to make it easier to read

plt.figure(1)
plt.hist(mean_samples, bins = 10);
plt.title(MEAN_TITLE);
plt.xlabel("sample means");
plt.ylabel("count");

plt.figure(2)
plt.hist(sd_samples, bins = 10);
plt.title(SD_TITLE);
plt.xlabel("sample sds");
plt.ylabel("count");
```





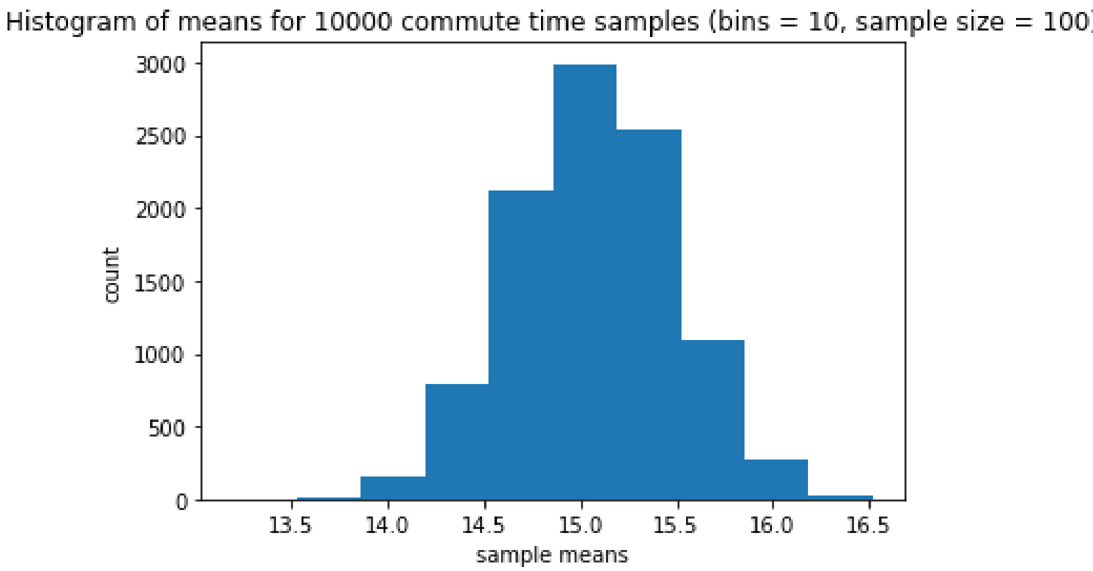

5e. Repeat the same exercise (plot histograms of means and standard deviations) as above for N=10000 and n= 100. What do you notice?

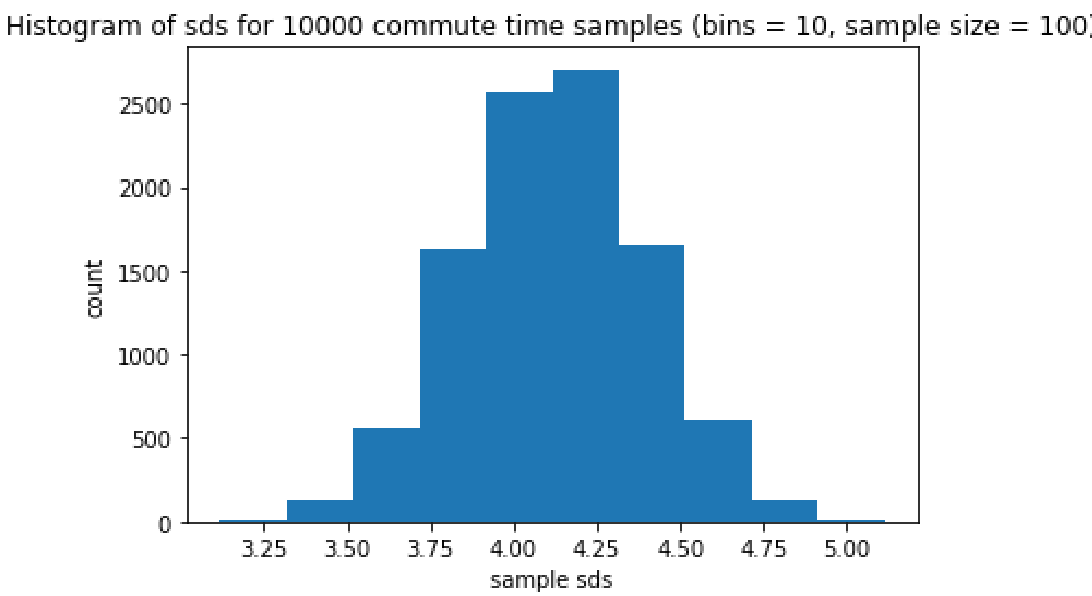
```
In [211]: mean_samples, sd_samples = repeat(times, 10000, 100)

          MEAN_TITLE = SAMPLE_TITLE_F.format(stat="means", n_samples=10000,
                                             bins=10, samp_size=100)
          SD_TITLE = SAMPLE_TITLE_F.format(stat="sds", n_samples=10000,
                                           bins=10, samp_size=100)

          plt.figure(1)
          plt.hist(mean_samples, bins = 10);
          plt.title(MEAN_TITLE);
          plt.xlabel("sample means");
          plt.ylabel("count");

          plt.figure(2)
          plt.hist(sd_samples, bins = 10);
          plt.title(SD_TITLE);
          plt.xlabel("sample sds");
          plt.ylabel("count");
```




(Type your observation here and then press Run. Note that this “cell” is designated as Markdown instead of “code” so that you can type freely).

The distributions start to localize around the true mean and standard deviation as the number of sampling procedures increase.

5f. Now write a function sample_n2(np_array, n, with_replacement) which can produce a sample of length n np_array but produces a sample without replacement when with_replacement = False. It should default to with_replacement = True (where you can just call sample_n(np_array, n) from earlier). Call the sample_n2() function on the with n = 50 and with_replacement=True and separately for with_replacement=False. Print the mean and standard deviation for both cases. Use the commute_time array for input data.

```
In [212]: def sample_n2(np_array,n,with_replacement):
              if with_replacement:
                  return sample_n(np_array, n)
              else:
                  return random.sample(list(times), n)

          unique_samples = sample_n2(times, 50, False)
          sample_mean = mean(unique_samples)
          sample_sd = sd(unique_samples)

          print("The without replacement sample mean is {}.".format(sample_mean))
          print("The without replacement sample sd is {}.".format(sample_sd))
```

```
The without replacement sample mean is 15.055373561071436.
The without replacement sample sd is 4.229319937386925.
```

6. Submission You will submit all computational parts of problem sets, and the mini-projects, in the following manner: 1. Complete the Assignment on this portal; 2. Save your work (File > Save and Checkpoint); 3. Download the Notebook (File > Download as > Notebook); 4. Download the pdf (File > Download as > pdf via LaTex); 5. Submit both the .ipynb and .pdf files on Stellar.