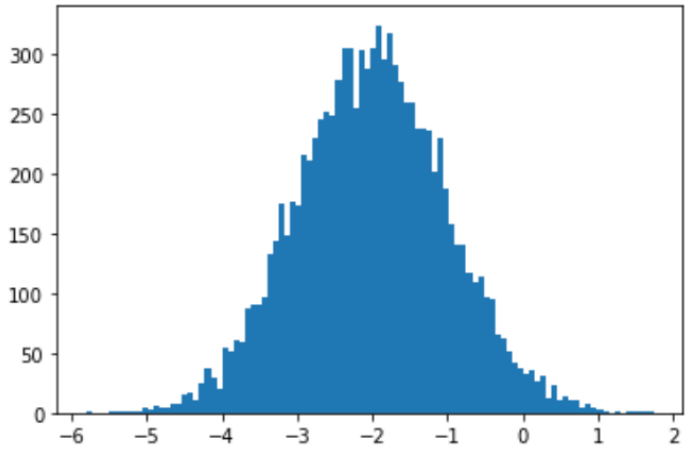
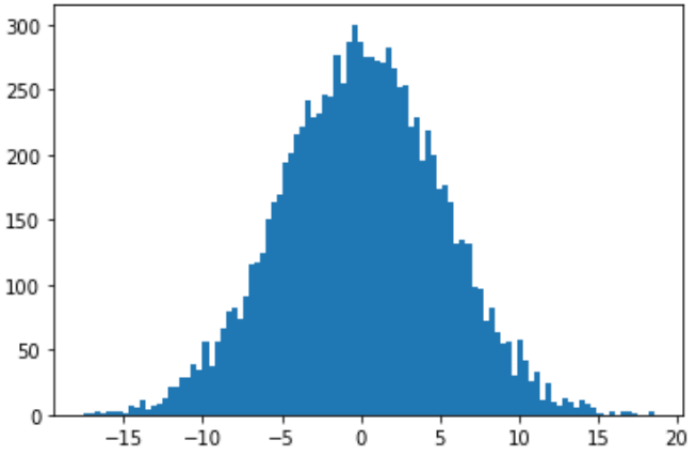
Andrew Nalundasan
OMSBA 5067 – ML for Business
April 3, 2021
Lab 1

Step 5a:
mu = -2 and sigma = 1, obs = 10000

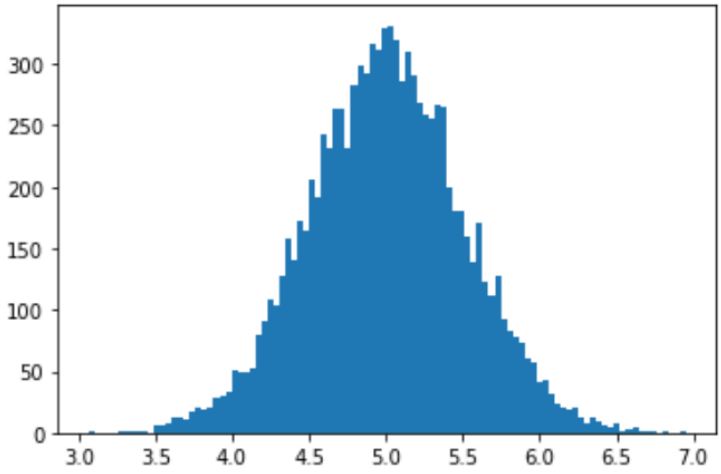


Step 5b:
mu = 0 and sigma = 5, obs = 10000



Step 5c:

mu = 5 and sigma = 0.5, obs = 10000



Step 7a:

```
In [7]: df_from_dict
Out[7]:
   Label  Fixed  Variable
0    big      1      2000
1  small      1         5
2    big      1      3000
3  small      1         4

In [8]: df_from_dict.T
Out[8]:
             0      1     2      3
Label      big  small   big  small
Fixed        1      1     1      1
Variable  2000      5  3000      4
```

I observed that 'df_from_dict.T' transposed the rows and columns in the data frame.

Step 7b:

```
In [19]: df_from_dict.describe()
Out[19]:
       Fixed     Variable
count    4.0     4.000000
mean     1.0  1252.250000
std      0.0  1497.500223
min      1.0     4.000000
25%      1.0     4.750000
50%      1.0  1002.500000
75%      1.0  2250.000000
max      1.0  3000.000000
```

I observed that 'df_from_dict.describe()' provided summary statistics of the data frame.

Step 9:

```
# Generates three sets of numpy normal random numbers with means of -2, 0,
# and +2, and with standard deviations equal to 1. Each set should contain
# 2000 random numbers.

R_neg2 = np.random.normal(-2, 5, 2000)
R_zero = np.random.normal(0, 5, 2000)
R_pos2 = np.random.normal(2, 5, 2000)

# Puts these arrays in a dictionary with these keys: "R_neg2", "R_zero", and "R_pos2".

step_9_b = dict()
step_9_b["R_neg2"] = np.array(R_neg2)
step_9_b["R_zero"] = np.array(R_zero)
step_9_b["R_pos2"] = np.array(R_pos2)

# Converts the dictionary to a DataFrame, name it as "df"

df = pd.DataFrame(step_9_b)
```

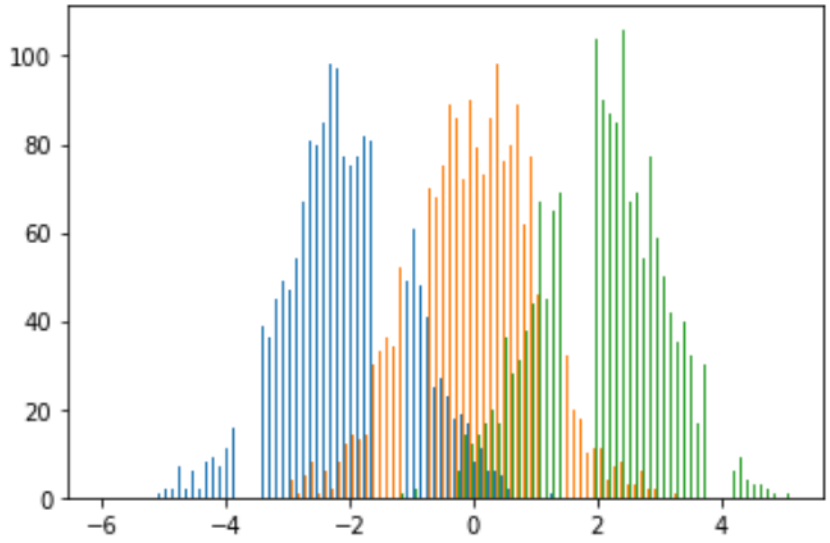
```
In [66]: df
Out[66]:
        R_neg2    R_zero    R_pos2
0    -2.960242  1.520506  0.553563
1    -1.717066  0.503570  2.089423
2    -1.615100  0.565199  1.185042
3    -1.683619  0.062338  1.821417
4    -1.834068  1.703496  0.995718
...        ...       ...       ...
1995 -3.212412  0.390735  1.612982
1996 -2.096782 -0.263948  3.066223
1997 -1.428307 -2.495172  1.601978
1998 -1.297502  0.530318  1.388933
1999 -1.218693 -0.209089  1.539741

[2000 rows x 3 columns]
```

```
# Calculates the statistics of each column (using describe())
df.describe()
```

```
In [65]: df.describe()
Out[65]:
            R_neg2       R_zero       R_pos2
count  2000.000000  2000.000000  2000.000000
mean     -2.038380     0.005726     2.008784
std       1.006312     0.996874     1.018358
min      -5.540003    -3.251841    -1.808037
25%      -2.723157    -0.679556     1.304424
50%      -2.056097    -0.030348     2.023008
75%      -1.363859     0.685595     2.705594
max       1.459809     3.349805     5.001620
```

# Plots the histogram of the three sets of numbers on a same graph.
plt.hist([R_neg2, R_zero, R_pos2], 100)
plt.show



all_low = np.where((df["R_neg2"]<=-2) & (df["R_zero"]<=-2) & (df["R_pos2"]<=-2))
all_high = np.where((df["R_neg2"]>=2) & (df["R_zero"]>=2) & (df["R_pos2"]>=2))
print("All Low Count: ",len(all_low[0])," All High Count: ",len(all_high[0]))

	std. dev = 1	std. dev = 2	std. dev = 3	std. dev = 4	std. dev = 5
all_low	0	1	18	48	81
all_high	0	4	25	43	75

As we increase the number of standard deviations, we see that both numbers for all_low and all_high increase along with the standard deviations. This occurs because when the number of standard deviations increase, the distribution widens, increasing the chances of including values lower than -2 and higher that 2. We see that with 5 standard deviations, this yields the highest amount of all_low and all_high. The counts of these numbers will be different each time we run the code since we are working with a random number generator.