# Week 4 Quiz

## [Yuqiao Liu] - [yl4278]

### Due Sunday Feb 23 9:00am

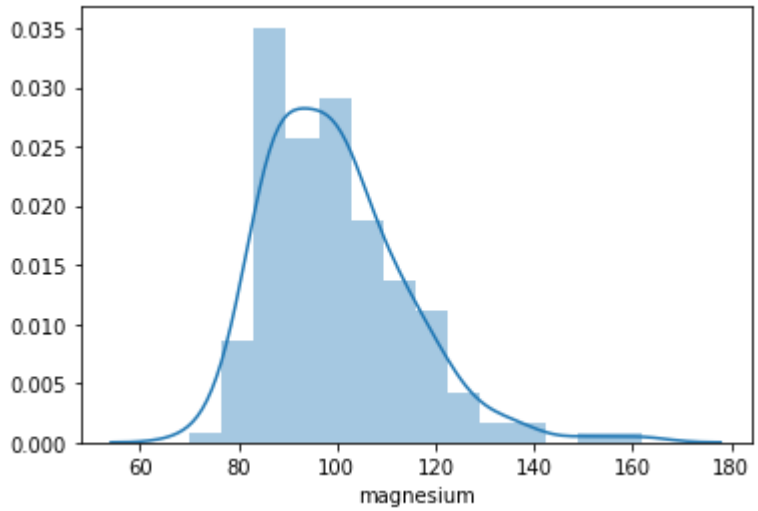
In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline
```

We're going to calculate the 95% confidence interval for the mean value of 'magnesium' from our wine dataset.

In [2]:
```
# Read in ../data/wine_dataset.csv as df
df = pd.read_csv('../data/wine_dataset.csv')
```

In [3]:
```
# Generate a distribution plot of the magnesium column.
sns.distplot(df.magnesium)
```

Out[3]: <matplotlib.axes._subplots.AxesSubplot at 0x244d0132d08>




In [4]:
```
# Assign the mean value of magnesium to variable observed_mean
observed_mean = df.magnesium.mean()

# Print the observed mean to the hundredths place
print(f'observed mean: {observed_mean:.02f}')
```

```
observed mean: 99.74
```

In [5]:
```
# generate a bootstrap sample (with the same number of values as the original
#   using pandas sample (with replacement)
#   using random_state=123
#   assign the result to sample
sample = df.magnesium.sample(random_state=123, replace=True)

# Print the mean of the sample to the hundredths place
# Note: if the sample mean is the same as the observed mean,
#       check, are you sampling with replacement?
print(sample.mean())
```

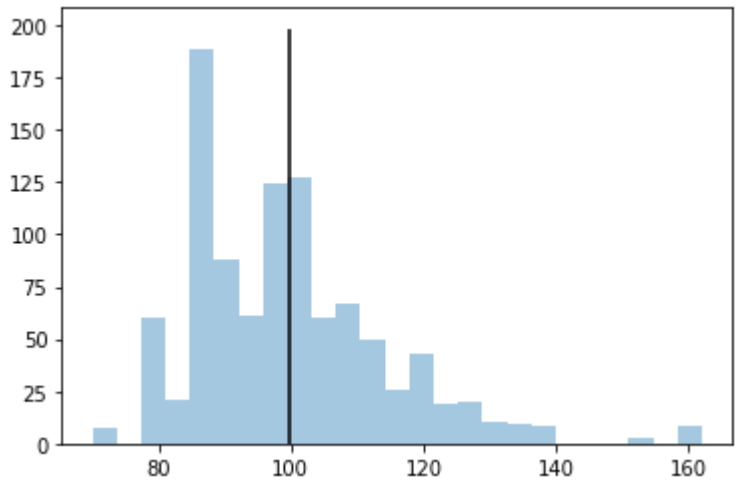
```
94.0
```

In [6]:
```
# Generate 1000 additional sample means using bootstrap sampling
#   each sample should have the same length as the original dataframe
#   store in the list sample_means
#   do not use random_state for this step (your results may differ from the c
sample_means = []
for i in range(1000):
    sample_mean = df.magnesium.sample(random_state=i, replace=True).mean()
    sample_means.append(sample_mean)

# Print the first 5 values in sample_means
sample_means[0:5]
```

Out[6]:
```
[91.0, 98.0, 105.0, 80.0, 102.0]
```

In [7]:
```
# Plot the distribution of sample means using sns.distplot
# Store the returned axis in ax.
ax = sns.distplot(sample_means, kde=False)

# Add a vertical line located at the observed mean on the x-axis using ax.vli
#     Use ax.get_ylim() to provide the y limits

ax.vlines(observed_mean,*ax.get_ylim())
```

Out[7]:
```
<matplotlib.collections.LineCollection at 0x244d36a04c8>
```



In [8]:
```
# To get the 95% confidence interval, we need want to retain the central 95%
# To do this we need to first determine how many values must be trimmed from
# For 95% CI, we want to trim 1/2 of 5% from each end.
# Calculate 2.5% of the length of sample_means and store as trim_amount.
trim_amount = 1000 * .05/2

# We want to index into our sample means, but trim_amount is a float.
# We must be first round this value and converted to an integer.
# Use np.round() to round and int() to convert to int and store the result in
trim_idx = int(np.round(trim_amount))

# Print trim_idx
trim_idx
```

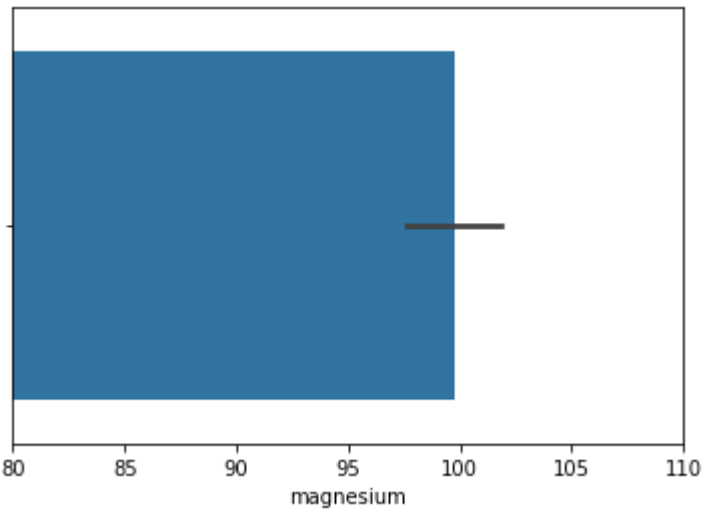
Out[8]:
```
25
```

In [9]:
```
# We can now print the 95% CI for our measure by indexing into the sorted arr
# Use np.sorted() to return a sorted numpy array
#   then use a list of the indices we want [trim_idx,-trim_idx] to index into
# Store the 95% CI values as ci
ci = np.sort(sample_means)[[trim_idx,-trim_idx]]

# Print the ci values
ci
```

Out[9]:
```
array([ 78., 136.])
```

In [11]:
```
# We can confirm these values by using seaborns barplot function.
# Call seaborn barplot on the magnesium column from the original dataframe
#   and compare the 95% CI plotted there to the values we calculated
# Capture the returned axis in x
ax = sns.barplot(df.magnesium,
            estimator=np.mean,
            ci=95,
            n_boot=1000
           )

# Modify the xaxis to zoom in between 80 and 110 using ax.set_xlim(80,110)
ax.set_xlim(80, 110)
```

Out[11]: (80, 110)




In [ ]: