Review of Descriptive Statistics with Applications in Python

08 January 2020

Overview

- Distinguishing between **S**tatistics and **s**tatistics
- Descriptive Statistics vs. Inferential Statistics
- Descriptive Statistics
    - Measures of central tendency
    - Measures of variation
- Generating numeric and visual data summaries using numpy, pandas, and seaborn

Statistics

- Statistics is a sub-field of applied mathematics and is concerned with analyzing data
- More specifically, statistics involves the following tasks
  - Collecting Data
  - Organizing Data
  - Displaying and Presenting Data
  - Interpreting Data
- Statistical methods are used to make **descriptions** and/or **inferences** about some population
- It's not surprising then that statistical methods used in data analyses are often sub-divided into two classes
  - Descriptive statistical methods
  - Inferential statistical methods

Distinguishing between **S**tatistics and **s**tatistics

- Before moving forward we need to make a clear distinction between **S**tatistics (big S) and **s**tatistics (little s)
  - **S**tatistics (big S) is a sub-field of applied mathematics and is concerned with analyzing data
  - **s**tatistics (little s) are numerical quantities calculated from a data set that provide important features about the data
- In this presentation we define a number of descriptive **s**tatistics (little s)
  - Explain what important features they provide to help us understand our data
  - Show how they are calculated
  - Demonstrate how to compute them using Python
- The big idea is that descriptive statistics allow us to reduce large data sets down to a few numerical measures - these measures give clues as to how to proceed in an analysis

Descriptive Statistics

- Descriptive statistics are numerical measures that help analysts communicate the features of a data set by giving short summaries about the **measures of central tendency** or the **measures of dispersion (variability)**

- Measures of central tendency describe the location of the center of a distribution or a data set

- Some commonly used measures of central tendency are
  - mean
  - median
  - mode

Descriptive Statistics

- Measures of variability describe how spread-out the data are
- While measures of central tendency help locate the middle of a data set, they don’t provide information about how the data are arranged (aka distributed)
- Some commonly used measures of variability include:
  - standard deviation
  - variance
  - minimum and maximum values
  - range
  - kurtosis
  - skewness

Descriptive Statistics vs. Inferential Statistics

- It’s rare that a data set contains observe from every member of a population
  - Most analyses are conducted on a representative sample taken from the population
  - Analysts make inferences about the population based on observations contained in the sample
- Inferential statistics are measures resulting from mathematical computations – help analysts **infer** trends about a population based upon the study of the sample
- Examples of inferential statistics
  - Methods to compute **Confidence intervals** that “capture” a population parameter with a specified degree of confidence
  - Methods to test claims about the population by analyzing a representative samples (**hypothesis tests**)

Descriptive Statistics using Python

- In the following slides we'll cover several descriptive statistics and show how to compute them using Python

- To do this we'll use the following Python libraries (clicking the links below will take you to the reference pages of each library)

  - **statistics** - Functions for calculating mathematical statistics of numeric (Real-valued) data
  - **numpy** - Create efficient multi-dimensional data objects for scientific computing
  - **seaborn** - High-level interface for drawing attractive and informative statistical graphics
  - **matplotlib** - Foundational 2D plotting library for Python
  - **statsmodels** - Implement statistical models, statistical tests, and statistical data exploration
  - **scipy** - Foundational software for mathematics, science, and engineering
  - **pandas** - High-performance, easy-to-use data structures and analysis tools for Python

Importing the Python Libraries

- First, we need to import the libraries into our workspace to make them available
- If you have Python installed on your machine you can copy/paste the code below
- Note that Python allow us to denote a library using a shorthand notation which I'll use in subsequent slides

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

- If you don't have Python installed click **this link** to interact with Python using Google Colaboratory – Go **here** to learn more about Google Coloaboratory,

Measures of Central Tendency - **mean**

- The mean() function from the statistics library returns the arithmetic average of a set of numeric values stored in a data object

- For the set of values $x_1, x_2, \ldots, x_N$ the mean is calculated as

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} = \frac{x_1 + x_2 + \cdots + x_N}{N}.$$

- The sample mean gives an unbiased estimate of the true population mean
  - When taken on average over all the possible samples, `mean()` converges on the true mean of the entire population
  - If the data are the entire population rather than a sample, then `mean(data)` is equivalent to calculating the true population mean μ.

Measures of Central Tendency - **mean**

- In the most simple case, we can create an array of values and find the mean on these values by passing the array to the `mean()` function.

```
nums = [-2,-4,1,2,3,5,7,9]
st.mean(nums)
## 2.625
```

- Similarly, we can create a dictionary of value:key pairs and the `mean()` function will compute the mean of the values (assuming all of the the values are numeric).

```
Dict = {1:"one",2:"two",3:"three"}

Dict
## {1: 'one', 2: 'two', 3: 'three'}
st.mean(Dict)
## 2
```

Measures of Central Tendency - **median**

- The median() function from the statistics library returns the middle value of a set of numeric values stored in a data object

- For the set of values $X = x_1, x_2, \ldots, x_N$ the median is calculated as

$$\text{median}(X) = \frac{X_{\lfloor (N+1) \div 2 \rfloor} + X_{\lceil (N+1) \div 2 \rceil}}{2}$$

- where X is an ordered list of numbers, N denotes the length of X, and $\lfloor . \rfloor$ and $\lceil . \rceil$ represent the floor and ceiling functions, respectively.

- The median is a preferred measure of central location skewed distributions and data sets, in a later slide we show how the median summarizes differently from the mean

Descriptive statistics - **median**

- As was shown when introducing the mean, we can compute the median of an array of values by passing the array to the `median()` function.

- Note that because the array contains an even number of elements the median is computed as mean of the two number in the middle - in this case 2 and 3

```
nums = [-2,-4,1,2,3,5,7,9]
st.median(nums)
## 2.5
```

- Similarly, the `median()` function will compute the median of the values in a dictionary containing value:key pairs (assuming all of the the values are numeric)

```
Dict = {1:"one",2:"two",3:"three"}


st.median(Dict)
## 2
```

Measures of Central Tendency - **mode**

- The mode() function from the statistics library returns the value that is most probable or occurs most often in a set of numeric values stored in a data object
- Note that in the array defined below there is not unique mode as each value occurs once - this results in the function throwing a `StatisticsError`

```
nums = [-2,-4,1,2,3,5,7,9]
st.mode(nums)
```

```
## StatisticsError: no unique mode; found 8 equally common values
##
## Detailed traceback:
##   File "<string>", line 1, in <module>
##   File "C:\Users\Aubur\Anaconda3\lib\statistics.py", line 506, in mode
##     'no unique mode; found %d equally common values' % len(table)
```

Measures of variation - **range**

- The range of a data set shows the span of the data
- For a sample of observations $X = x_1, x_2, \ldots, x_N$ the range of X may be found from a simple computation

$$\text{range(X)} = \max(X) - \min(X)$$

- Note - the value of the range statistic is determined by only two observations from any data set - and is easily influenced by the presence of outliers
- In Python the range statistic may be computed using the intrinsic functions `max()` and `min()`

```
max(nums) - min(nums)
## 13
```

Measures of variation - **variance**

- The variance of a data set measures how far the values are spread out from their average value (or mean)

- For a sample of observations $X = x_1, x_2, \dots, x_N$ the unbiased sample variance, denoted as s^2 is computed as

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$$

- If the data are the entire population the the population variance, denoted as σ^2 or Var$[X]$ is computed

$$\sigma^2 = \mathrm{Var}(X) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Measures of variation - **variance**

- In Python the variance of an array of numeric values can be computed using the variance function from the statistics library

```
st.variance(nums)
## 19.125
```

- The variance function can also be used to compute the variance of the values contained within a Dictionary

```
st.variance(Dict)
## 1
```

Measures of variation - **standard deviation**

- The standard deviation of a data set, like the variance, is measure of how far the values are spread out relative to the mean

- A useful property of the standard deviation is that, unlike the variance, it is expressed in the same units as the data

- If the data are a sample the sample standard deviation, denoted by s is the square root of the sample variance

$$s = \sqrt{s^2} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

- If the data are the population the standard deviation, denoted by σ is the square root of the variance

Measures of variability - **standard deviation**

- In Python the standard deviation of an array of numeric values can be computed using the `stdev` function from the statistics library

```
st.stdev(nums)
## 4.373213921133975
```

- The `stdev` function can also be used to compute the standard deviation of the values contained within a Dictionary

```
st.stdev(Dict)
## 1.0
```

Generating numeric and visual data summaries

- In the next slides we show various ways to summarize data
- Visual summaries
  - Histograms
  - Boxplots
  - Scatterplots
- Numeric summaries
  - z-score
  - covariance
  - correlation

Generating numeric and visual data summaries

- To implement these summaries I create two numpy arrays containing pseudorandom observations generated from two different distributions
- For the first array I use numpy's `random.normal()` function **link** to generate 4000 observations from a standard normal distribution $NOR(0,1)$
- For the second array I use numpy's `random.lognormal()` function **link** to generate 4000 observations from a lognormal distribution $LOGNOR(1,0.75)$

```
N_obs = 4000

normal = np.random.normal(loc = 1, scale = 10000, size = N_obs)

lognormal = np.random.lognormal(mean = 10, sigma = .75, size = N_obs)
```

Generating numeric and visual data summaries

- Next, I create a dictionary with two keys `'normal'` and `'lognormal'` and assign the corresponding numpy arrays to these keys

```
d = {'normal': normal, 'lognormal': lognormal}
d
## {'normal': array([  1314.0900122 ,    195.34383563,  25135.95705104,
...,
##        -17298.6486693 ,   6813.87149383, -16375.90889681]),
'lognormal': array([28551.28177716, 55348.79027248, 44183.99946235, ...,
##        12221.28225576,  7869.89264355, 37439.5384024 ])}
```

- Then I use the `DataFrame()` function link from pandas to transform the dictionary into a data frame

```
df = pd.DataFrame(data = d)
```

Use `.head()` to view the first 10 rows in the data frame

```
df.head(10)
##              normal     lognormal
## 0   1314.090012  28551.281777
## 1    195.343836  55348.790272
## 2  25135.957051  44183.999462
## 3   3255.216938   6384.297271
## 4   -784.807298   5051.660669
## 5   6142.461554  63948.783707
## 6  -7715.000079  20129.411928
## 7  -2857.794677  45008.321252
## 8   3644.349437  32198.066925
## 9  12682.566722  34493.857155
```

Histogram of normal observations

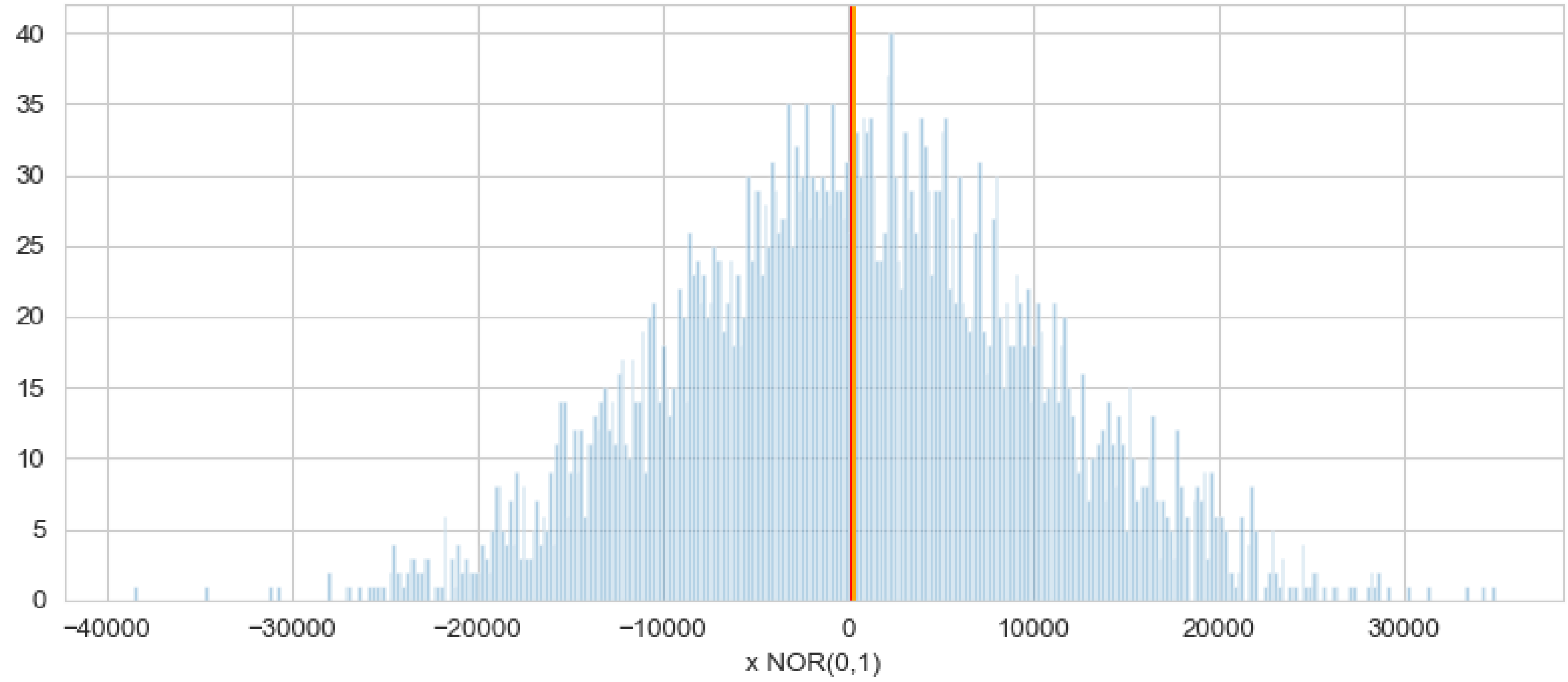
- This code creates a histogram for the normal data showing that mean and median are nearly the same for symmetrically distributed data (i.e. have low skewness values)

```
# Settings
sb.set_style("whitegrid")


# Create histogram
plot = sb.distplot(df['normal'],
                   kde = False,
                   bins = int(N_obs / 10),
                   axlabel = "x NOR(0,1)")


# add vertical line showing the location of the mean, median
plot = plt.axvline(df['normal'].mean(),   0,1, color = 'red')
plot = plt.axvline(df['normal'].median(), 0,1, color = 'orange')


plt.show(plot)
```



Histogram of pseudorandom observations from a standard normal distribution

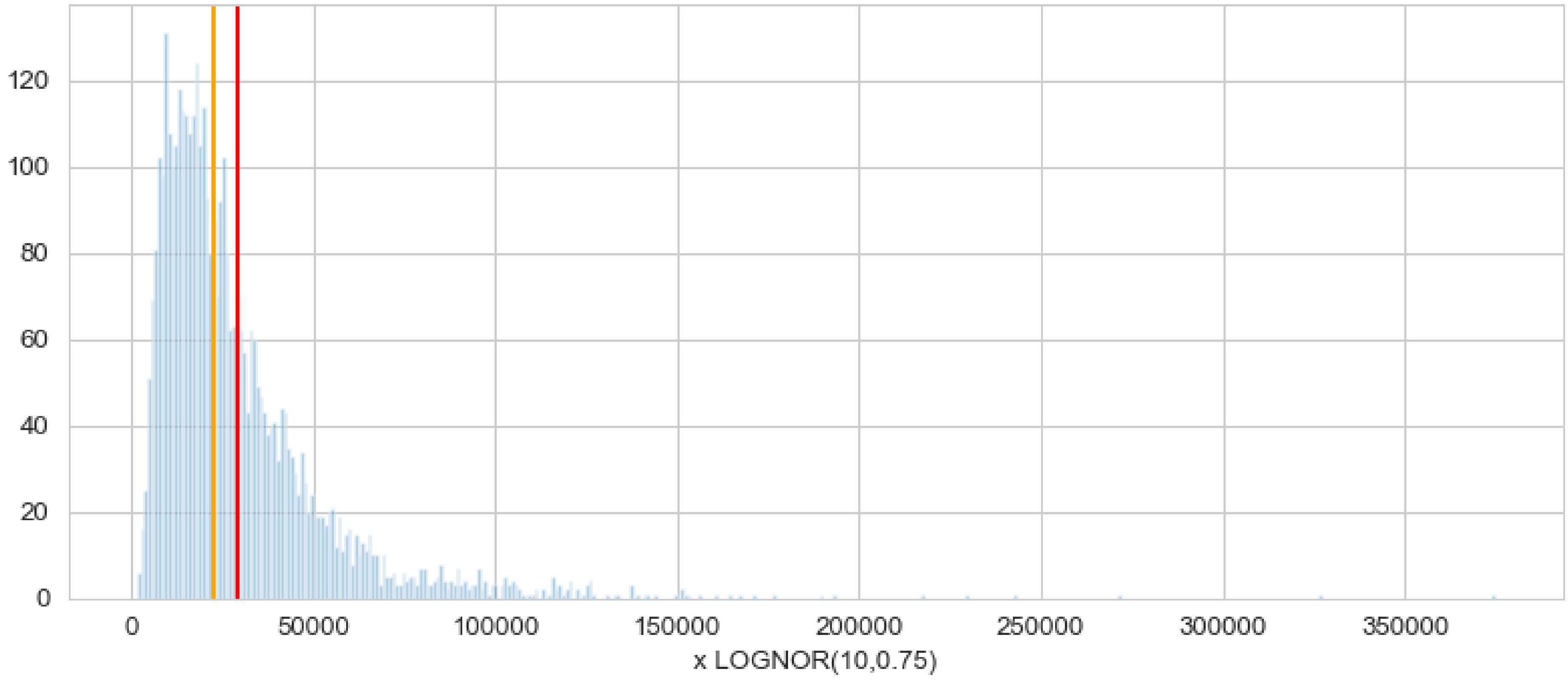
Histogram of lognormal observations

- This code creates a histogram for the lognormal data and shows how the mean and median separate when the data are not symmetrically distributed (i.e. have larger skewness values)

```
# Create histogram
plot = sb.distplot(df['lognormal'],
                   kde = False,
                   bins = int(N_obs / 10),
                   axlabel = "x LOGNOR(10,0.75)")


# add vertical line showing the location of the mean, median
plot = plt.axvline(df['lognormal'].mean(),   0,1, color = 'red')
plot = plt.axvline(df['lognormal'].median(), 0,1, color = 'orange')


plt.show(plot)
```



Histogram of pseudorandom observations from a standard normal distribution

Box plots

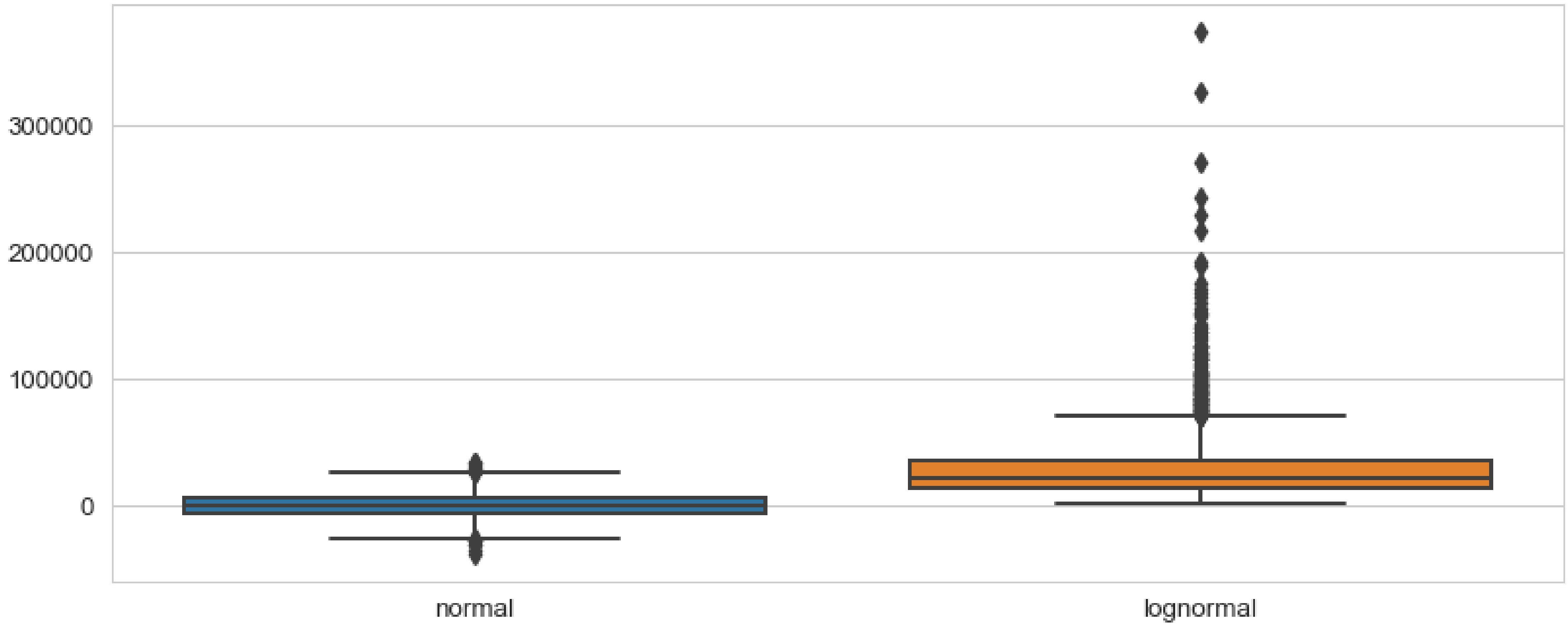
- A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable.

- The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

- The code below generates a seaborn boxplot displaying both columns of the DataFrame

```
plot = sb.boxplot(data = df)

plt.show(plot)
```

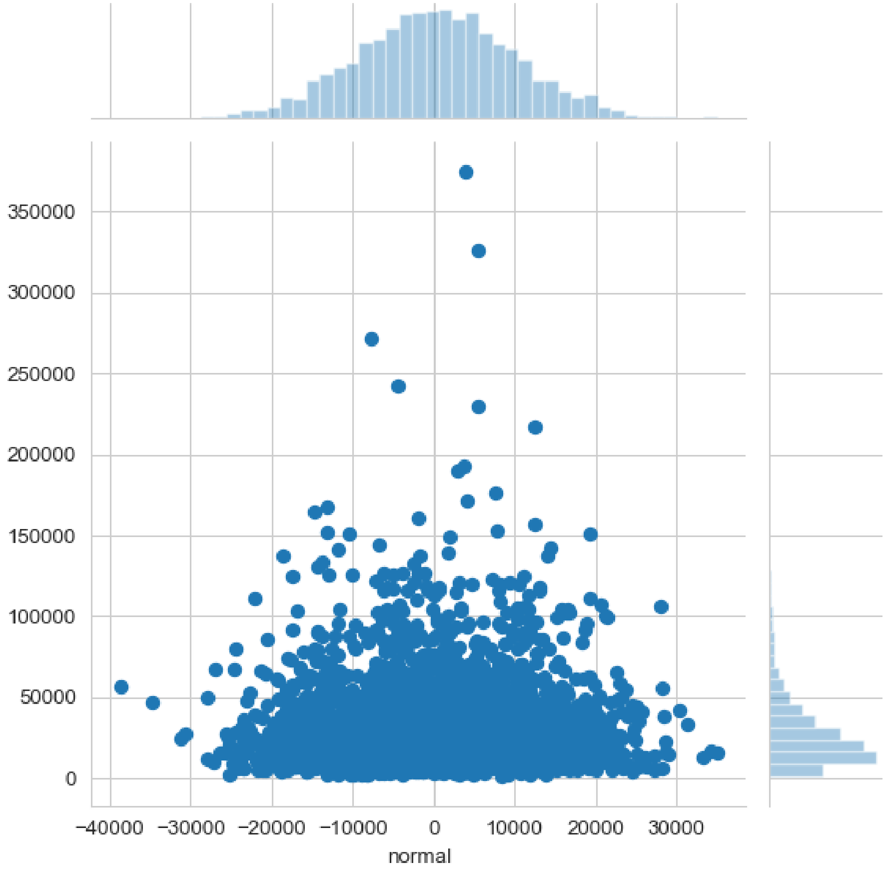


Boxplot comparing the normal and lognormal data

Jointplots

- A joint plot combines a histogram with scatter plot
- Scatter plots are useful for visualizing the relationship between variables
- The scatter chart created by the code below shows no evidence of a linear relationship between the normally distributed observations and the lognormally distributed observations

```
sb.jointplot("normal","lognormal", data = df)
## <seaborn.axisgrid.JointGrid object at 0x0000000036292A48>
plt.show()
```



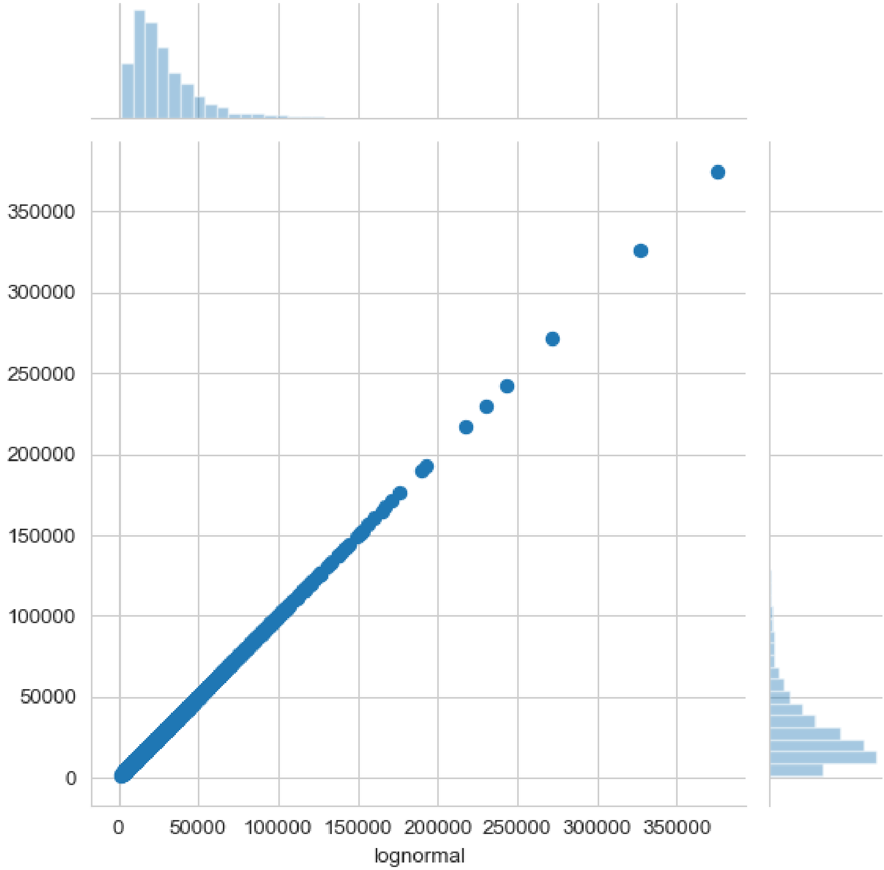
Jointplot showing the relationship between the normal and lognormal observations

Jointplots

- But, what if we modified the code to produce a jointplot of the lognormal with itself?
- Do so gives the expected result, the plot shows that the lognormal data has a perfect linear relationship with itself
- Can we express this numerically?

```
plot = sb.jointplot("lognormal","lognormal", data = df)

plt.show(plot)
```



Jointplot showing the relationship between the lognormal observations at itself

Z-Score

- The z-score (aka the standard score)
  - A measure computed for each value in a data set
  - Returns the number of standard deviations below or above the mean
  - Usually assumes that the data are normally distributed
- The Z-score for a sample is computed as

$$z_i = \frac{x_i - \overline{x}}{s_x}$$

Z-Score

- If you’re comparing multiple samples that may contain a different number of elements, the Z-score for each sample is computed as

$$z_i = \frac{x_i - \overline{x}}{s_x / \sqrt{n}}$$

- Where the n term is used to account for potentially different sample sizes

z-score

- In Python we can compute the z-score for the normally distributed data using the `zscore()` function from scipy

```
z1 = sp.stats.zscore(df['normal'])

# Print first 10 elements in z1

z1[0:9]
## array([ 0.10782604, -0.00644843,  2.54111351,  0.30610268, -0.10656608,
##          0.60102063, -0.81445146, -0.31831163,  0.34585066])
```

z-score

- Or we can compute it using `pandas` functions

```
(df['normal'] - df['normal'].mean()) / df['normal'].std()
## 0       0.107813
## 1      -0.006448
## 2       2.540796
## 3       0.306064
## 4      -0.106553
##           ...
## 3995    1.730497
## 3996   -1.282631
## 3997   -1.793150
## 3998    0.669518
## 3999   -1.698909
## Name: normal, Length: 4000, dtype: float64
```

Covariance

- **Covariance** is a descriptive statistic used to measure the linear association between two variables

- The sample covariance between variables X and Y is computed as

$$S_{XY} = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{N - 1}$$

- The population covariance is computed as

$$\sigma_{XY} = \frac{\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)}{N - 1}$$

Covariance

- To compute the covariances for the DataFrame `df` we created earlier we can use the `cov()` function from the pandas library

```
cov1 = df.cov()
cov1
```

```
##                    normal     lognormal
## normal     9.586793e+07  4.206186e+06
## lognormal  4.206186e+06  6.203347e+08
```

Correlation

- **Correlation** is a descriptive statistic used to measure the linear association between two variables

- The correlation (or correlation coefficient) is a measure defined between -1 and 1

- Is a dimensionless quantity that is not affected by the units of measurement for X and Y

- The sample correlation between variables X and Y is computed as

$$r_{XY} = \frac{S_{XY}}{S_X S_Y}$$

Correlation

- To compute the correlations for the DataFrame `df` we created earlier we can use the `corr()` function from the pandas library

```
df.corr()
##               normal  lognormal
## normal      1.000000   0.017248
## lognormal   0.017248   1.000000
```

Covariance and Correlation

- Finally, what if we wanted to compute the covariance ourselves?
- The code below computes the covariance as well as the difference between this value and the value found from using the `cov()` function from pandas

```
X = df['normal']
Y = df['lognormal']
X_diff = X - st.mean(X)
Y_diff = Y - st.mean(Y)
prod = X_diff * Y_diff

cov2 = sum(prod) / (len(X) - 1)

cov1['lognormal'][0] - cov2
## 1.30385160446167e-08
```