MSBA Workshop: Brief Review of Logarithms

Logarithms

- A **logarithm** is an exponent.
- If $y = b^x$ then $\log_b(y) = x$.
- b is called the base.
- A logarithm answers the question: “Compute the power (x) that a base (b) must be raised to in order to return a certain value (y).”
- For example, since $16 = 4^2$ then $\log_4(16) = 2$, and since $16 = 2^4$ then $\log_4(16) = 2$.

1. $\log_{10}(100)$

2. $\log_{10}(1000)$

3. $\log_{10}(10000)$

4. $\log_{10}(10)$

5. $\log_{10}(1)$

6. $\log_8(64)$

7. $\log_2(64)$

- Any number to the power of 0 is 1: $b^0 = 1$.
- So $\log_b(1) = 0$ for any base b.

Logarithms in Python

The libraries `math` and `numpy` include functions for computing logarithms.

```python
import math

math.log(100, 10)
```

```
2.0
```

```python
math.log(64, 8)
```

```
2.0
```

```
math.log(64, 2)
```

```
6.0
```

The number e

Invest 1 dollar at an annual interest rate of 100%. How much money do you have at the end of 1 year if interest is compounded:

1. Annually
2. Semi-annually
3. Monthly
4. Daily
5. n times in the year
6. Continuously

$$e = \lim_{n\to\infty}\left(1 + \frac{1}{n}\right)^n \approx 2.7183$$

```
import numpy as np

np.exp(1)
```

```
np.float64(2.718281828459045)
```

Exponential function

How much money do you have at the end of 2 years if interest is compounded continuously?

```
import numpy as np

np.exp(2)
```

```
np.float64(7.38905609893065)
```

Invest 1 dollar at an annual interest rate of 10%. How much money do you have at the end of 1 year if interest is compounded:

1. Annually
2. Semi-annually
3. Monthly
4. Daily
5. n times in the year
6. Continuously

$$e^x = \lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n$$

Natural logarithm

The base-e logarithm is called the **natural logarithm**. In math, it is usually denoted ln.

$$\ln(x) = \log_e(x)$$

However, in data science and in most software packages, Python included, log or `log` refers to "natural log".

```
np.log(np.exp(1))
```

```
np.float64(1.0)
```

```
np.log(np.exp(1) ** 2)
```

```
np.float64(2.0)
```

```
np.log(10)
```

```
np.float64(2.302585092994046)
```

The `numpy` function for base-10 logarithms is `log10`.

```
np.log10(10)
```

```
np.float64(1.0)
```

```
np.log10(100)
```

```
np.float64(2.0)
```

Log Transformations

Log transformations of variables are common in data science. Let's look at the lot area variable from the Ames housing data.

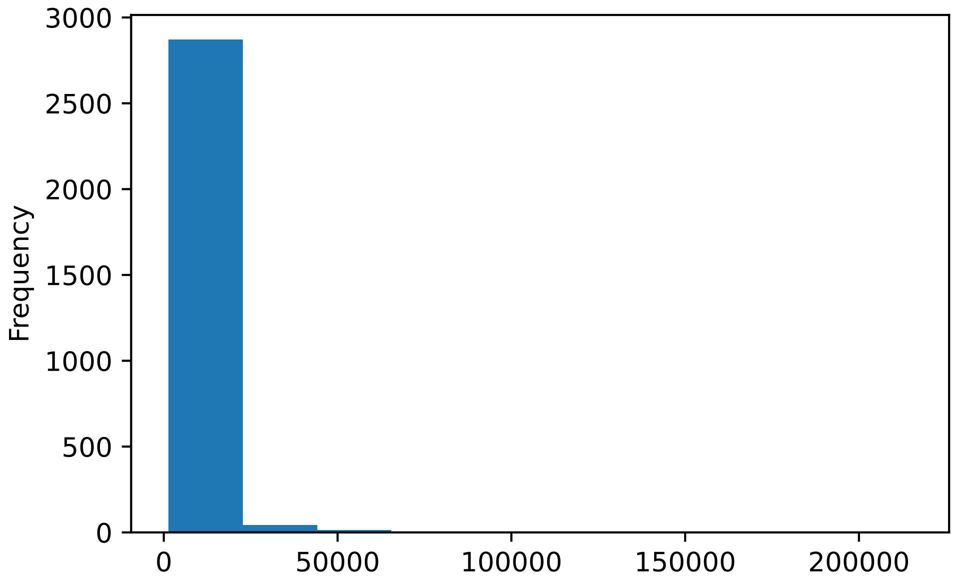
```
import pandas as pd
```

```
df_ames = pd.read_csv("https://raw.githubusercontent.com/kevindavisross/data
301/main/data/AmesHousing.txt", sep="\t")
df_ames
```

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
0	1	526301100	20	RL	141.0	31770	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	NaN	0	5	2010	WD	Normal	215000
1	2	526350040	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	NaN	0	6	2010	WD	Normal	105000
2	3	526351010	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	...	0	NaN	NaN	Gar2	12500	6	2010	WD	Normal	172000
3	4	526353030	20	RL	93.0	11160	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2010	WD	Normal	244000
4	5	527105010	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	...	0	NaN	MnPrv	NaN	0	3	2010	WD	Normal	189900
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2925	2926	923275080	80	RL	37.0	7937	Pave	NaN	IR1	Lvl	...	0	NaN	GdPrv	NaN	0	3	2006	WD	Normal	142500
2926	2927	923276100	20	RL	NaN	8885	Pave	NaN	IR1	Low	...	0	NaN	MnPrv	NaN	0	6	2006	WD	Normal	131000
2927	2928	923400125	85	RL	62.0	10441	Pave	NaN	Reg	Lvl	...	0	NaN	MnPrv	Shed	700	7	2006	WD	Normal	132000

	Order	PID	MS SubClass	MS Zoning	Lot Frontage	Lot Area	Street	Alley	Lot Shape	Land Contour	...	Pool Area	Pool QC	Fence	Misc Feature	Misc Val	Mo Sold	Yr Sold	Sale Type	Sale Condition	SalePrice
2928	2929	924100070	20	RL	77.0	10010	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	4	2006	WD	Normal	170000
2929	2930	924151050	60	RL	74.0	9627	Pave	NaN	Reg	Lvl	...	0	NaN	NaN	NaN	0	11	2006	WD	Normal	188000

```
df_ames["Lot Area"].plot.hist()
```



There are a few homes with such extreme lot areas that we get virtually no resolution at the lower end of the distribution. Over 95% of the observations are in a single bin of this histogram. In other words, the distribution of this variable is extremely *skewed*.

One way to improve this histogram is to use more bins. But this does not solve the fundamental problem: we need more resolution at the lower end of the scale and less resolution at the higher end. One way to spread out the values at the lower end of a distribution and to compress the values at the higher end is to take the logarithm (provided that the values are all positive). Log transformations are particularly effective at dealing with data that is skewed to the right.

```
df_ames["log(Lot Area)"] = np.log(df_ames["Lot Area"])
df_ames[["Lot Area", "log(Lot Area)"]]
```

	Lot Area	log(Lot Area)
0	31770	10.366278
1	11622	9.360655
2	14267	9.565704
3	11160	9.320091
4	13830	9.534595
…	…	…
2925	7937	8.979291
2926	8885	9.092120
2927	10441	9.253496
2928	10010	9.211340
2929	9627	9.172327

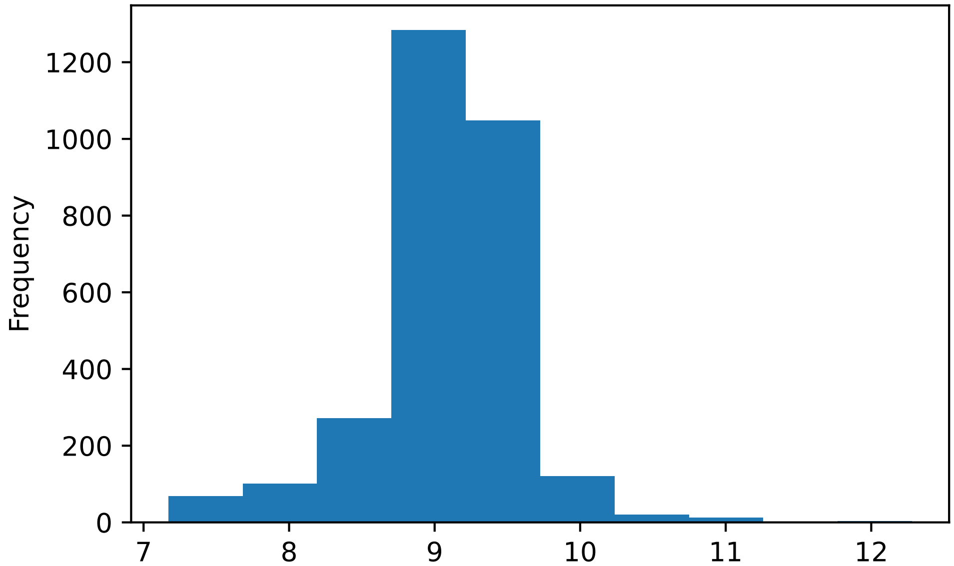
Remember "log" usually refers to "natural log", and that's what we have computed (`np.log`). We could also transform using base-10 log (`np.log10`).

```
df_ames["log10(Lot Area)"] = np.log10(df_ames["Lot Area"])
df_ames[["Lot Area", "log(Lot Area)", "log10(Lot Area)"]]
```

	Lot Area	log(Lot Area)	log10(Lot Area)
0	31770	10.366278	4.502017
1	11622	9.360655	4.065281
2	14267	9.565704	4.154333
3	11160	9.320091	4.047664
4	13830	9.534595	4.140822
…	…	…	…
2925	7937	8.979291	3.899656
2926	8885	9.092120	3.948657
2927	10441	9.253496	4.018742
2928	10010	9.211340	4.000434
2929	9627	9.172327	3.983491

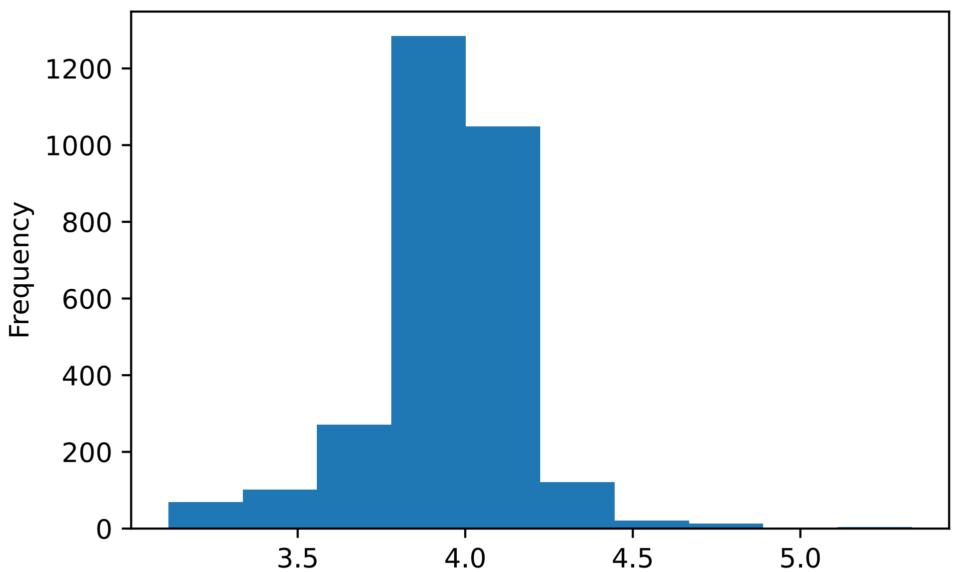
These numbers are not very interpretable on their own, but if we make a histogram of these values, we see that the lower end of the distribution is now more spread out, and the extreme values are not as far out on the logarithmic scale.

```
df_ames["log(Lot Area)"].plot.hist()
```



The effect is the same if we take natural log or base-10 log; the only difference is the scale.

```
df_ames["log10(Lot Area)"].plot.hist()
```



> ⚠ Warning
>
> Do NOT interpret symmetric as “good” and skewed as “bad”. The shape of the data is what it is; there is no right or wrong.

- From an analysis perpesctive, the fact that Lot Area was skewed just meant that the histogram —which uses bins of equal width—did not adequately visualize the sample data.
- It is often more natural to view values that span multiple orders of magnitude on a logarithmic/ multiplicative scale (“how many *times* bigger?”) rather than a linear/absolute scale.
- Taking a log transformation allows us to differentially zoom in to different ranges of the data, achieving better resolution when the distribution is skewed.

Here is a graph from the *NY Times* with GDP per capita on a logarithmic scale.

Useful properties of logarithms

Roughly, logarithms turn multiplication into addition. The logarithm of a product is the sum of the logarithms.

$$\log_b(xy) = \log_b(x) + \log_b(y)$$

Roughly, logarithms turn exponentiation (powers) into products.

$$\log_b(x^y) = y \log_b(x)$$