Homework 10: Linear Regression

Reading:

- Linear Regression (https://www.inferentialthinking.com/chapters/15/2/Regression_Line.html)
- Method of Least Squares (https://www.inferentialthinking.com/chapters/15/3/Method_of_Least_Squares.html)
- Least Squares Regression (https://www.inferentialthinking.com/chapters/15/4/Least_Squares_Regression.html)

Please complete this notebook by filling in the cells provided. Before you begin, execute the following cell to load the provided tests. Each time you start your server, you will need to execute this cell again to load the tests.

Homework 10 is due **Thursday, 4/16 at 11:59pm**. You will receive an early submission bonus point if you turn in your final submission by Wednesday, 4/15 at 11:59pm. Start early so that you can come to office hours if you're stuck. Check the website for the office hours schedule. Late work will not be accepted as per the policies (http://data8.org/sp20/policies.html) of this course.

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged. Refer to the policies page to learn more about how to learn cooperatively.

For all problems that you must write our explanations and sentences for, you **must** provide your answer in the designated space. Moreover, throughout this homework and all future ones, please be sure to not re-assign variables throughout the notebook! For example, if you use `max_temperature` in your answer to one question, do not reassign it later on.

In [1]:
```
# Don't change this cell; just run it.

import numpy as np
from datascience import *

# These lines do some fancy plotting magic.
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
import warnings
warnings.simplefilter('ignore', FutureWarning)

from client.api.notebook import Notebook
ok = Notebook('hw10.ok')
```

```
=====================================================================
Assignment: Homework 10: Linear Regression
OK, version v1.14.19
=====================================================================
```

```
---------------------------------------------------------------------
----
LoadingException                          Traceback (most recent call l
ast)
<ipython-input-1-dc7635bfe511> in <module>
     13
     14 from client.api.notebook import Notebook
---> 15 ok = Notebook('hw10.ok')

/opt/anaconda3/lib/python3.7/site-packages/client/api/notebook.py in __
init__(self, filepath, cmd_args, debug, mode)
     13         ok_logger = logging.getLogger('client')   # Get top-lev
el ok logger
     14         ok_logger.setLevel(logging.DEBUG if debug else logging.
ERROR)
---> 15         self.assignment = load_assignment(filepath, cmd_args)
     16         # Attempt a login with enviornment based tokens
     17         login_with_env(self.assignment)

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in
load_assignment(filepath, cmd_args)
     22     if cmd_args is None:
     23         cmd_args = Settings()
---> 24     return Assignment(cmd_args, **config)
     25
     26 def _get_config(config):

/opt/anaconda3/lib/python3.7/site-packages/client/sources/common/core.p
y in __call__(cls, *args, **kargs)
    185                 raise ex.SerializeException('__init__() missing
expected '
    186                                             'argument {}'.format(attr))
--> 187         obj.post_instantiation()
    188         return obj
    189

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in
post_instantiation(self)
    151     def post_instantiation(self):
    152         self._print_header()
--> 153         self._load_tests()
    154         self._load_protocols()
    155         self.specified_tests = self._resolve_specified_tests(

/opt/anaconda3/lib/python3.7/site-packages/client/api/assignment.py in
_load_tests(self)
    205
    206         if not self.test_map:
--> 207             raise ex.LoadingException('No tests loaded')
    208
    209     def dump_tests(self):

LoadingException: No tests loaded
```

Exploring the PTEN Gene with Linear Regression

1. PTEN Linear Regression

This week's homework is about linear regression. The dataset we'll be using is from the Cancer Cell Line Encyclopedia -- you can read more about this database in this paper (https://www.nature.com/articles/s41586-019-1186-3) and interact with the data yourself at the online portal here (https://portals.broadinstitute.org/ccle).

The specific dataset we'll be taking a look at is expression data for the PTEN gene in around 1000 cell lines. The PTEN gene is a tumor-suppressing gene, and mutations in the PTEN gene are associated with many types of cancer. A cell line is group of cells that are kept alive and replicate indefinitely in culture (grown in petri dishes, for example).

Run the following cell to load the `pten` table. The `pten` table has four columns, a column for the specific `Cell Line`, a column for the `Copy Number`, which is how many times a copy of a portion of the PTEN gene is found in the DNA of that cell line, `mRNA Expression (Affy)`, and `mRNA Expression (RNAseq)`.

In [2]:
```
# Just run this cell
pten = Table().read_table("pten.csv")
pten.show(5)
```

Cell Line	Copy Number	mRNA Expression (Affy)	mRNA Expression (RNAseq)
DMS53_LUNG	0.1728	7.4829	2.4898
SW1116_LARGE_INTESTINE	0.191	8.15495	2.86857
NCIH1694_LUNG	-0.2372	7.99308	2.82148
P3HR1_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	-0.0129	9.96358	4.65702
HUT78_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	-1.1013	8.74116	2.75236

... (968 rows omitted)

In [3]:
```
# Just run this cell
pten.hist("Copy Number", bins = np.arange(-1, 1.5, 0.5))
```

Question 1

Looking at the histogram above, we want to check whether or not `Copy Number` is in standard units. For this question, compute the mean and the standard deviation of the values in `Copy Number` and assign these values to `copy_number_mean` and `copy_number_sd` respectively. After you calculate these values, assign `is_su` to either `True` if you think that `Copy Numbers` is in standard units or `False` if you think otherwise.

```
BEGIN QUESTION
name: q1_1
manual: false
```

In [4]:
```
copy_number = pten.column("Copy Number")
copy_number_mean = np.mean(copy_number) # SOLUTION
copy_number_sd = np.std(copy_number) # SOLUTION
is_su = False # SOLUTION
print(f"Mean: {copy_number_mean}, SD: {copy_number_sd}, Is in standard units?: {is_su}")
```

```
Mean: -0.19447913669064748, SD: 0.7462401942190691, Is in standard units?: False
```

In [5]:
```
# TEST
type(is_su) == bool
```

Out[5]:
```
True
```

In [6]:
```
# HIDDEN TEST
all([np.round(copy_number_mean, 2) == -0.19, np.round(copy_number_sd, 2)
== 0.75, is_su == False])
```

Out[6]:
```
True
```

Question 2

Create the function `standard_units` so that it converts the values in the array `arr` to standard units. We'll then use `standard_units` to create a new table, `pten_su` , that converts all the values in the table `pten` to standard units.

```
BEGIN QUESTION
name: q1_2
manual: false
```

In [7]:
```
def standard_units(arr):
    return (arr - np.mean(arr)) / np.std(arr) # SOLUTION

# DON'T DELETE OR MODIFY ANY OF THE LINES OF CODE BELOW IN THIS CELL
pten_su = Table().with_columns("Cell Line", pten.column("Cell Line"),
                               "Copy Number SU", standard_units(pten.col
umn("Copy Number")),
                               "mRNA Expression (Affy) SU", standard_uni
ts(pten.column("mRNA Expression (Affy)")),
                               "mRNA Expression (RNAseq) SU", standard_u
nits(pten.column("mRNA Expression (RNAseq)"))
                              )
pten_su.show(5)
```

Cell Line	Copy Number SU	mRNA Expression (Affy) SU	mRNA Expression (RNAseq) SU
DMS53_LUNG	0.492173	-0.925344	-0.17077
SW1116_LARGE_INTESTINE	0.516562	-0.355245	0.0992261
NCIH1694_LUNG	-0.0572481	-0.492553	0.0656589
P3HR1_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	0.243325	1.17902	1.37408
HUT78_HAEMATOPOIETIC_AND_LYMPHOID_TISSUE	-1.21519	0.142045	0.0163913

... (968 rows omitted)

In [8]:
```
# TEST
standard_units(make_array(1,2,3,4,5))
```

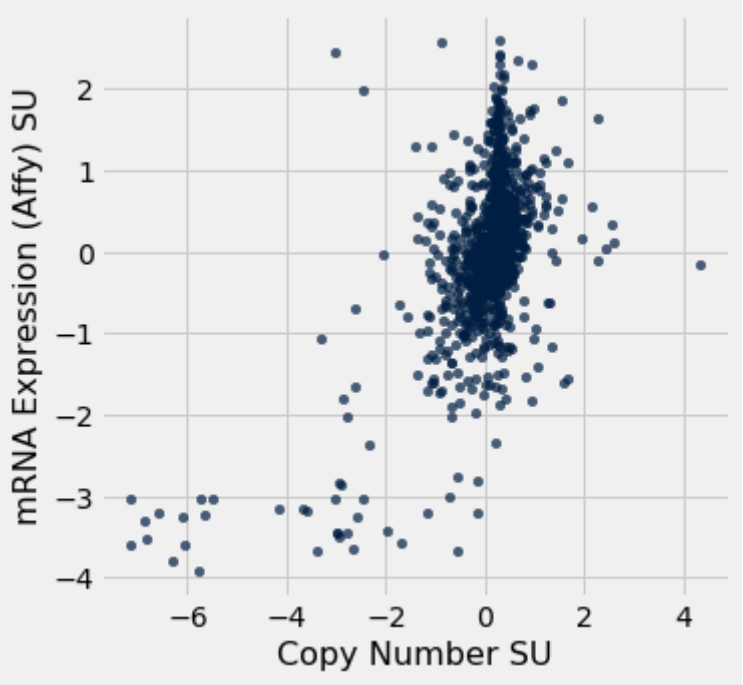
Out[8]:
```
array([-1.41421356, -0.70710678,  0.        ,  0.70710678,  1.4142135
6])
```

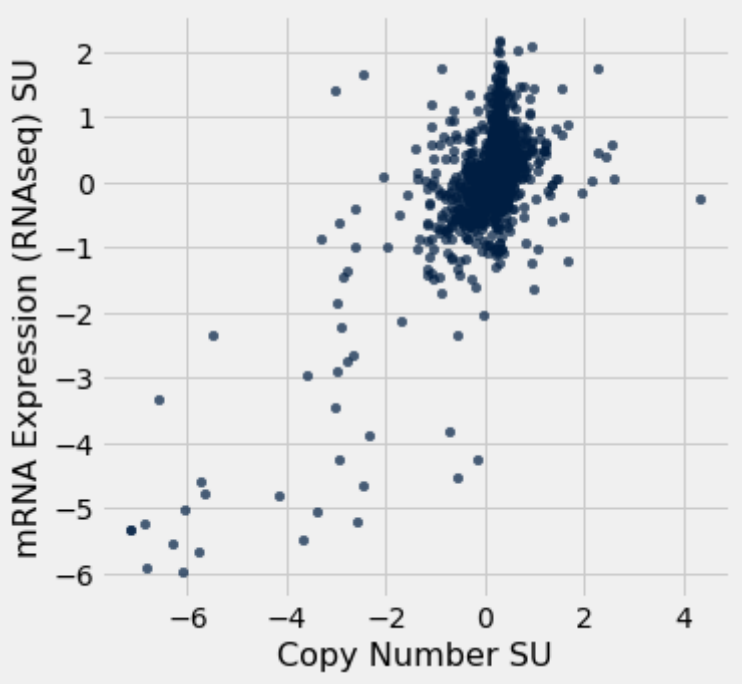
In [9]:
```
# HIDDEN TEST
standard_units(make_array(-3, -2, 1, 0, 1, 2, 3))
```

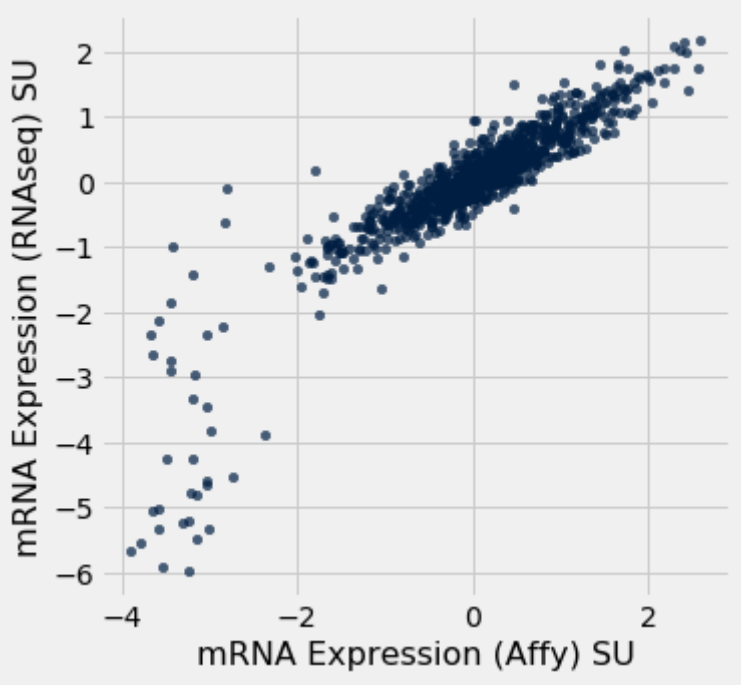
Out[9]:
```
array([-1.65988202, -1.15470054,  0.36084392, -0.14433757,  0.36084392,
        0.8660254 ,  1.37120689])
```

You should always visually inspect your data, before numerically analyzing any relationships in your dataset. Run the following cell in order to look at the relationship between the variables in our dataset.

In [10]:
```
# Just run this cell
pten_su.scatter("Copy Number SU", "mRNA Expression (Affy) SU")
pten_su.scatter("Copy Number SU", "mRNA Expression (RNAseq) SU")
pten_su.scatter("mRNA Expression (Affy) SU", "mRNA Expression (RNAseq) S
U")
```


mRNA Expression (Affy) SU
Copy Number SU
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2
4


mRNA Expression (RNAseq) SU
Copy Number SU
2
1
0
−1
−2
−3
−4
−5
−6
−6
−4
−2
0
2
4



Question 3

Which of the following relationships do you think has the highest correlation (i.e. highest absolute value of `r`)? Assign `highest_correlation` to the number corresponding to the relationship you think has the highest correlation.

1. Copy Number vs. mRNA Expression (Affy)
2. Copy Number vs. mRNA Expression (RNAseq)
3. mRNA Expression (Affy) vs. mRNA Expression (RNAseq)

```
BEGIN QUESTION
name: q1_3
manual: false
```

In [11]:
```
highest_correlation = 3 # SOLUTION
```

In [12]:
```
# TEST
type(highest_correlation) == int
```

Out[12]: True

In [13]:
```
# HIDDEN TEST
highest_correlation == 3
```

Out[13]: True

Question 4

Now, using the `standard units` function, define the function `correlation` which computes the correlation between `arr1` and `arr2` .

```
BEGIN QUESTION
name: q1_4
manual: false
```

In [14]:
```
def correlation(arr1, arr2):
    return np.mean(standard_units(arr1) * standard_units(arr2)) # SOLUTI
ON

# This computes the correlation between the different variables in pten
copy_affy = correlation(pten.column("Copy Number"), pten.column("mRNA Ex
pression (Affy)"))
copy_rnaseq = correlation(pten.column("Copy Number"), pten.column("mRNA
 Expression (RNAseq)"))
affy_rnaseq = correlation(pten.column("mRNA Expression (Affy)"), pten.co
lumn("mRNA Expression (RNAseq)"))

print(f" \
      Copy Number vs. mRNA Expression (Affy) Correlation: {copy_affy}, 
\n \
      Copy Number vs. mRNA Expression (RNAseq) Correlation: {copy_rnase
q}, \n \
      mRNA Expression (Affy) vs. mRNA Expression (RNAseq) Correlation: 
{affy_rnaseq}")
```

```
       Copy Number vs. mRNA Expression (Affy) Correlation: 0.5819516653
311988,
       Copy Number vs. mRNA Expression (RNAseq) Correlation: 0.69541960
09651351,
       mRNA Expression (Affy) vs. mRNA Expression (RNAseq) Correlation:
0.9000764746535077
```

In [15]:
```
# TEST
correlation([1,2,3], [4,5,6])
```

Out[15]:
```
0.9999999999999999
```

In [16]:
```
# HIDDEN TEST
correlation([-3, 0, 3], [-3, 0, 3])
```

Out[16]:
```
1.0000000000000002
```

Question 5

If we switch what we input as arguments to `correlation`, i.e. found the correlation between `mRNA Expression (Affy)` vs. `Copy Number` instead of the other way around, would the correlation change? Assign `correlation_change` to either `True` if you think yes, or `False` if you think no.

```
BEGIN QUESTION
name: q1_5
manual: false
```

In [17]:
```
correlation_change = False # SOLUTION
```

In [18]:
```
# TEST
type(correlation_change) == bool
```

Out[18]: True

In [19]:
```
# HIDDEN TEST
correlation_change == False
```

Out[19]: True

Question 6

Looking at both the scatter plots after Question 2 and the correlations computed in Question 4, describe a pattern you see in the relationships between the variables.

```
BEGIN QUESTION
name: q1_6
manual: true
```

SOLUTION: Any of the following patterns are sufficient: all the relationships have a positive association, all the relationships have a fairly high correlation.

Question 7

Let's look at the relationship between mRNA Expression (Affy) vs. mRNA Expression (RNAseq) only. Define a function called `regression_parameters` that returns the parameters of the regression line as a two-item array containing the slope and intercept of the regression line as the first and second elements respectively. The function `regression_line` takes in two arguments, an array of `x` values, and an array of `y` values.

```
BEGIN QUESTION
name: q1_7
manual: false
```

In [20]:
```
def regression_parameters(x, y):
    # BEGIN SOLUTION
    x_mean = np.mean(x)
    y_mean = np.mean(y)
    x_sd = np.std(x)
    y_sd = np.std(y)
    r = correlation(x, y)
    # END SOLUTION
    slope = r * y_sd / x_sd # SOLUTION
    intercept = y_mean - (slope * x_mean) # SOLUTION
    return make_array(slope, intercept)

parameters = regression_parameters(pten.column("mRNA Expression (Affy)"
), pten.column("mRNA Expression (RNAseq)"))
parameters
```

Out[20]:
```
array([ 1.07113964, -6.45428385])
```

In [21]:
```
# TEST
# The returned array should be of length 2
len(parameters) == 2
```

Out[21]:
```
True
```

In [22]:
```
# HIDDEN TEST
np.allclose(parameters, [1.07113964, -6.45428385])
```

Out[22]:
```
True
```

Question 8

If we switch what we input as arguments to `regression_parameters`, i.e. found the parameters for the regression line for `mRNA Expression (RNAseq)` vs. `mRNA Expression (Affy)` instead of the other way around, would the regression parameters change (would the slope and/or intercept change)? Assign `parameters_change` to either `True` if you think yes, or `False` if you think no.

```
BEGIN QUESTION
name: q1_8
manual: false
```

In [23]:
```
parameters_change = True # SOLUTION
```

In [24]:
```
# TEST
type(parameters_change) == bool
```

Out[24]:
```
True
```

In [25]:
```
# HIDDEN TEST
parameters_change == True
```

Out[25]:
```
True
```

Question 9

Now, let's look at how the regression parameters look like in standard units. Use the table `pten_su` and the function `regression_parameters`, and assign `parameters_su` to a two-item array containing the slope and the intercept of the regression line for mRNA Expression (Affy) in standard units vs. mRNA Expression (RNAseq) in standard units.

```
BEGIN QUESTION
name: q1_9
manual: false
```

In [26]:
```
parameters_su = regression_parameters(pten_su.column("mRNA Expression (Affy) SU"), pten_su.column("mRNA Expression (RNAseq) SU")) # SOLUTION
parameters_su
```

Out[26]:
```
array([9.00076475e-01, 1.16123028e-16])
```

In [27]:
```
# TEST
# The returned array should be of length 2
len(parameters_su) == 2
```

Out[27]:
```
True
```

In [28]:
```
# HIDDEN TEST
np.allclose(parameters_su, [9.00076475e-01, 1.16123028e-16])
```

Out[28]:
```
True
```

Question 10

Looking at the array `parameters_su`, what do you notice about the slope and intercept values specifically? Relate them to another value we already calculated in a previous question, as well as relate them to an equation.

```
BEGIN QUESTION
name: q1_10
manual: true
```

SOLUTION: The slope is 0.9 which matches the correlation we computed for mRNA Expression (Affy) vs. mRNA Expression (RNAseq) in question 4. The intercept is 0. Both these values correspond to the regression line calculated using standard units: $y_{su} = r * x_{su} + 0$

Question 11

The oldest and most commonly used cell line in Biology is the HeLa cell line, named after Henrietta Lacks, whose cervical cancer cells were taken without her consent in 1951 to create this cell line. The issue of data privacy and consent is very important to data science! You can read more about this topic here (https://www.hopkinsmedicine.org/henriettalacks/).

The HeLa cell line is missing from our dataset. If we know that the HeLa mRNA Expression (Affy) value is 8.2, what is the predicted mRNA Expression (RNAseq) value? Use the values in `parameters` that we derived in Question 1.7, and assign the result to `hela_rnaseq`.

```
BEGIN QUESTION
name: q1_11
manual: false
```

In [29]:
```
hela_rnaseq = parameters.item(0) * 8.2 + parameters.item(1) # SOLUTION
hela_rnaseq
```

Out[29]:
```
2.32906120281946
```

In [30]:
```
# TEST
type(hela_rnaseq) in set([float, np.float32, np.float64])
```

Out[30]:
```
True
```

In [31]:
```
# HIDDEN TEST
np.round(hela_rnaseq, 3) == 2.329
```

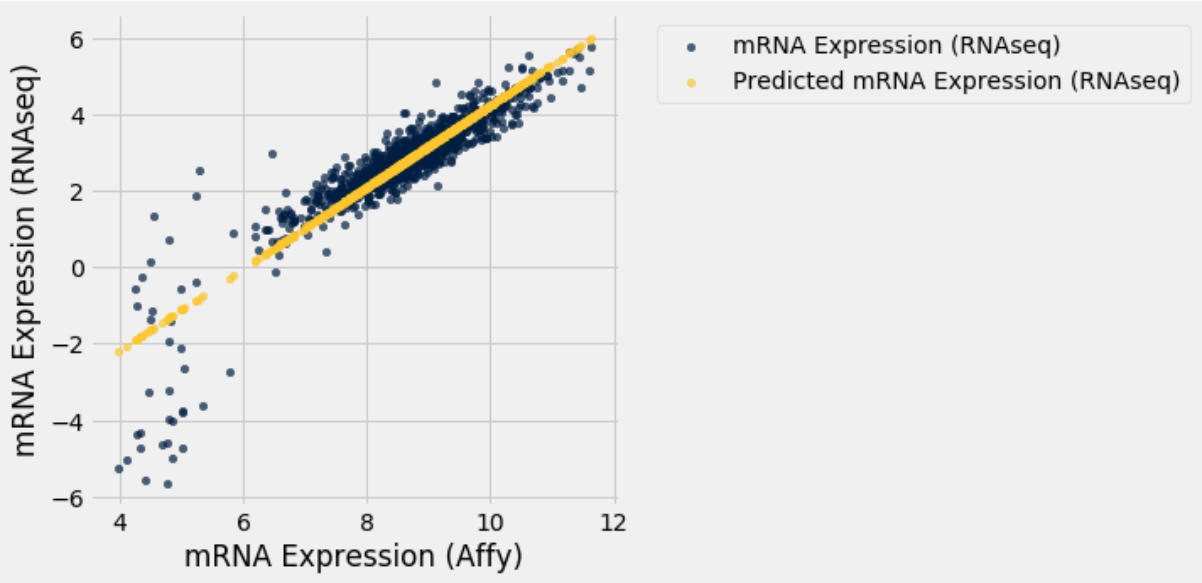
Out[31]:
```
True
```

Question 12

Compute the predicted mRNA Expression (RNAseq) values from the mRNA Expression (Affy) values in the `pten` table. Use the values in the `parameters` array from Question 1.7, and assign the result to `predicted_rnaseq`. We'll plot your computed regression line with the scatter plot from after question 1.2 of mRNA Expression (Affy) vs. mRNA Expression (RNAseq).

```
BEGIN QUESTION
name: q1_12
manual: true
```

In [32]:
```
predicted_rnaseq = parameters.item(0) * pten.column("mRNA Expression (Affy)") + parameters.item(1) # SOLUTION

# DON'T CHANGE/DELETE ANY OF THE BELOW CODE IN THIS CELL
(pten.with_column("Predicted mRNA Expression (RNAseq)", predicted_rnaseq)
 .select("mRNA Expression (Affy)", "mRNA Expression (RNAseq)", "Predicted mRNA Expression (RNAseq)")
 .scatter("mRNA Expression (Affy)"))
plt.ylabel("mRNA Expression (RNAseq)");
```



Fitting a least-squares regression line

Recall that the least-square regression line is the unique straight line that minimizes root mean squared error (RMSE) among all possible fit lines. Using this property, we can find the equation of the regression line by finding the pair of slope and intercept values that minimize root mean squared error.

Question 13

Define a function called `RMSE`. It should take two arguments:

1. the slope of a line (a number)
2. the intercept of a line (a number).

It should return a number that is the root mean squared error (RMSE) for a line defined with the arguments slope and intercept used to predict mRNA Expression (RNAseq) values from mRNA Expression (Affy) values for each row in the `pten` table.

Hint: Errors are defined as the difference between the actual `y` values and the predicted `y` values.

Note: if you need a refresher on RMSE, here's the link (https://www.inferentialthinking.com/chapters/15/3/Method_of_Least_Squares.html#Root-Mean-Squared-Error) from the textbook

```
BEGIN QUESTION
name: q1_13
manual: false
```

In [33]:
```
def RMSE(slope, intercept):
    affy = pten.column("mRNA Expression (Affy)")
    rnaseq = pten.column("mRNA Expression (RNAseq)")
    predicted_rnaseq = slope * affy + intercept # SOLUTION
    return np.sqrt(np.mean((rnaseq - predicted_rnaseq) ** 2)) # SOLUTION

# DON'T CHANGE THE FOLLOWING LINES BELOW IN THIS CELL
rmse_example = RMSE(0.5, 6)
rmse_example
```

Out[33]:
```
7.612008179226994
```

In [34]:
```
# TEST
5 < rmse_example < 10
```

Out[34]:
```
True
```

In [35]:
```
# HIDDEN TEST
np.round(rmse_example, 3) == 7.612
```

Out[35]:
```
True
```

Question 14

What is the RMSE of a line with slope 0 and intercept of the mean of `y` equal to?

Hint 1: The line with slope 0 and intercept of mean of `y` is just a straight horizontal line at the mean of `y`

Hint 2: What does the formula for RMSE become if we input our predicted `y` values in the formula. Try writing it out on paper! It should be a familiar formula.

```
BEGIN QUESTION
name: q1_14
manual: true
```

SOLUTION: The RMSE of a line with slope 0 and intercept of the mean of `y` is equal to the standard deviation of `y`. We know that the predicted line is just going to be a straight horizontal line with the mean of y as the y values. When we plug that in to the formula for RMSE we get:

$$\sqrt{mean((y_{actual} - y_{mean})^2)}$$

which is exactly equal to the standard deviation of `y`.

Question 15

Find the parameters that minimizes RMSE of the regression line for mRNA Expression (Affy) vs. mRNA Expression (RNAseq). Assign the result to `minimized_parameters`.

If you haven't tried to use the `minimize` function (http://data8.org/sp20/python-reference.html) yet, now is a great time to practice. Here's an example from the textbook (https://www.inferentialthinking.com/chapters/15/3/Method_of_Least_Squares.html#numerical-optimization).

Hint: Use the `RMSE` function in Question 1.13

NOTE: When you use the minimize function, please pass in `smooth=True` as the second argument to this function. You'll need to do this, otherwise, your answer will be incorrect

```
BEGIN QUESTION
name: q1_15
manual: false
```

In [36]:
```
minimized_parameters = minimize(RMSE, smooth=True) # SOLUTION
minimized_parameters
```

Out[36]:
```
array([ 1.07113932, -6.4542811 ])
```

In [37]:
```
# TEST
len(minimized_parameters) == 2
```

Out[37]:
```
True
```

In [38]:
```
# TEST
# Make sure to input `smooth=True` as the second argument to the `minimi
ze` function
# i.e. minimize(..., smooth=True)
(1 <= minimized_parameters.item(0) <= 2) and (-7 <= minimized_parameters
.item(1) <= -6)
```

Out[38]: True

In [39]:
```
# HIDDEN TEST
np.allclose(minimized_parameters, [1.07113932, -6.4542811])
```

Out[39]: True

Question 16

The slope and intercept pair you found in Question 1.15 should be very similar to the values that you found in Question 1.7. Why were we able to minimize RMSE to find the same slope and intercept from the previous formulas?

```
BEGIN QUESTION
name: q1_16
manual: true
```

SOLUTION: The regression line is the unique straight line (in other words, the unique slope/intercept pair) that minimizes RMSE. Therefore, we can also find the regression line by finding the slope and intercept values that minimize RMSE.

Question 17

If we had instead minimized mean squared error (MSE), would we have gotten the same slope and intercept of the minimized root mean squared error (RMSE) results? Assign `same_parameters` to either `True` if you think yes, or `False` if you think no.

```
BEGIN QUESTION
name: q1_17
manual: false
```

In [40]:
```
same_parameters = True # SOLUTION
same_parameters
```

Out[40]: True

In [41]:
```
# TEST
type(same_parameters) == bool
```

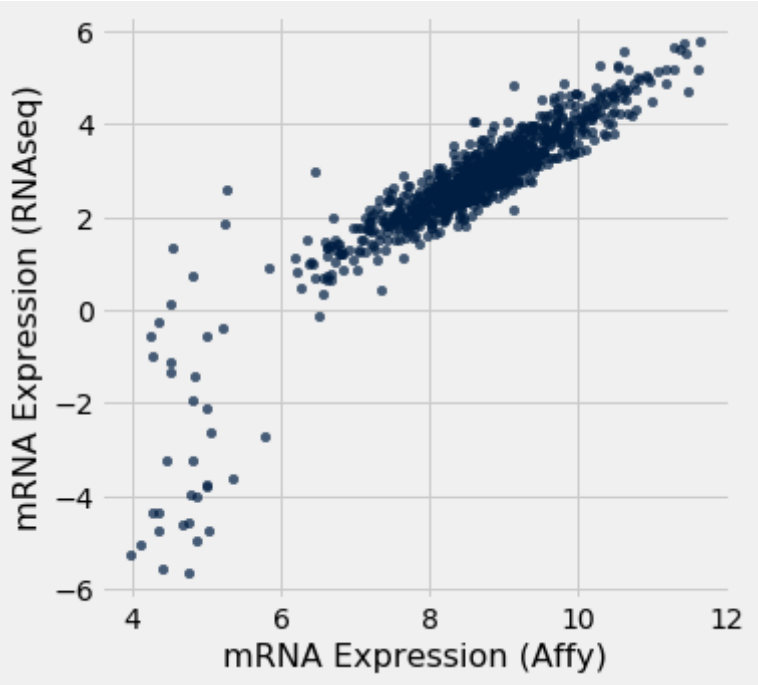
Out[41]: True

In [42]:
```
# HIDDEN TEST
same_parameters == True
```

Out[42]: True

Let's look at the scatter plot of the relationship between mRNA Expression (Affy) and mRNA Expression (RNAseq) again:

In [43]:
```
pten.scatter("mRNA Expression (Affy)", "mRNA Expression (RNAseq)")
```



Question 18

Using a linear regression model, would we be able to obtain accurate predictions for most of the points? Explain why or why not.

```
BEGIN QUESTION
name: q1_18
manual: true
```

SOLUTION: Yes, using linear regression to fit this data is valid because the correlation is high (0.9) as we found, and when we plotted the regression line/line of best fit, it crossed through a majority of the data points.

Alternatively, if you answered "No, ...", the only valid explanation for why using linear regression to fit this data would be invalid is because of the heteroscedasticity (meaning uneven spread in the errors seen with the fitted values vs actual values) seen with the errors of the fitted line vs actual values of the data points with an mRNA Expression (Affy) values between 4 and 6. You can read about heteroscedasticity in section 15.5 of the textbook (https://www.inferentialthinking.com/chapters/15/5/Visual_Diagnostics.html#Detecting-Heteroscedasticity).

2. Properties of Binary Distributions

Binary distributions arise in regular everyday life, and as data scientists you will encounter them constantly. A binary distribution is a distribution across two categories: such as voting in support of a proposition or voting against it on your local ballot, flipping heads or tails, having heart disease or not having heart disease. Generally we represent 'yes' or `True` as 1, and 'no' or `False` as 0. Binary distributions have some special properties that make working with them especially easy!

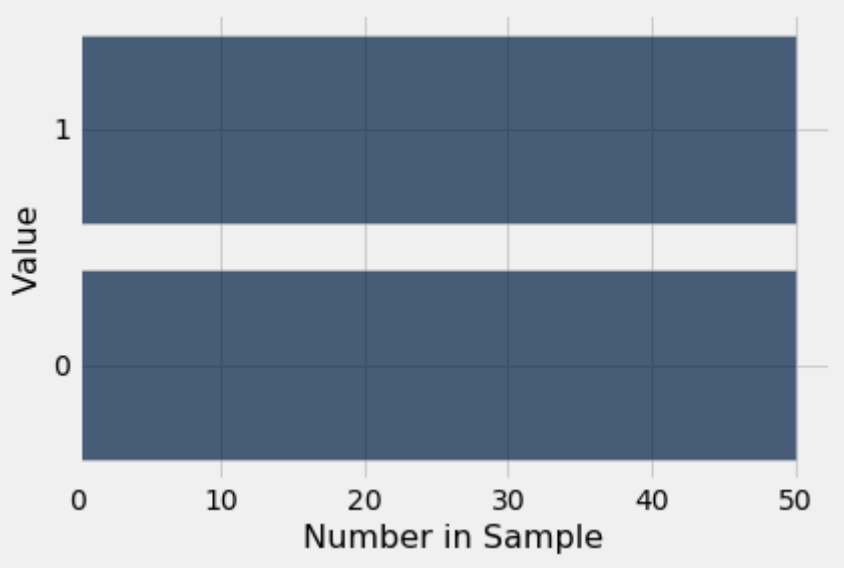
The intent of this section of the homework is to walk you through these properties, so we decided to make all of the tests for this section public (i.e. there are no hidden tests to worry about for this section only).

Question 1

Let's generate a random binary distribution of 0's and 1's. Assign `binary_options` to the correct array of possible values in a binary distribution (i.e. look at the previous sentence).

```
BEGIN QUESTION
name: q2_1
manual: false
```

In [44]:
```
binary_options = make_array(0, 1) # SOLUTION

# DON'T DELETE/MODIFY ANY OF THE CODE IN THIS CELL BELOW
sample_size = 100
binary_sample = np.random.choice(binary_options, sample_size)

# Run this to see a histogram of this random distribution.
Table().with_columns("Value", make_array(1, 0), "Number in Sample", make
_array(sum(binary_sample), sample_size - sum(binary_sample))).barh("Valu
e")
```



In [45]:
```
# TEST
# Make sure you assigned `binary_options` to an array
type(binary_options) == np.ndarray
```

Out[45]: True

In [46]:
```
# TEST
# Should be a two element array of a binary distribution
sorted(set(binary_options)) == sorted(set([0, 1]))
```

Out[46]: True

Question 2

The first property you should note is that the proportion of ones in a binary distribution is equal to the mean of the distribution. Think about why this is true (https://www.inferentialthinking.com/chapters/14/1/Properties_of_the_Mean.html#Proportions-are-Means). Complete the following cell to show that this is the case for your `binary_sample`. Assign `number_of_ones` and `number_of_zeros` to the number of `1`'s and the number of `0`'s respectively from your `binary_sample`.

```
BEGIN QUESTION
name: q2_2
manual: false
```

In [47]:
```
number_of_ones = np.count_nonzero(binary_sample == 1) # SOLUTION
number_of_zeros = np.count_nonzero(binary_sample == 0) # SOLUTION


# DON'T DELETE/MODIFY ANY OF THE CODE BELOW IN THIS CELL
number_values = len(binary_sample)
sum_of_binary_sample = sum(binary_sample)
# Remember that the mean is equal to the sum divided by the number of it
ems
mean_binary_sample = sum_of_binary_sample / number_values

# Don't change this!
print(f"In your binary sample there were {number_of_ones} ones and {numb
er_of_zeros} zeros. 1*{number_of_ones} + 0*{number_of_zeros} = {number_o
f_ones}")
print(f"The sum of values in your sample was {sum_of_binary_sample}, div
ided by the number of items, {number_values}, gives us a mean of {mean_b
inary_sample}")
print(f"The proportion of ones in your sample was {number_of_ones} ones,
divided by the number of items, {number_values}, gives us a value of {me
an_binary_sample}" )
print('Those values are equal!')
```

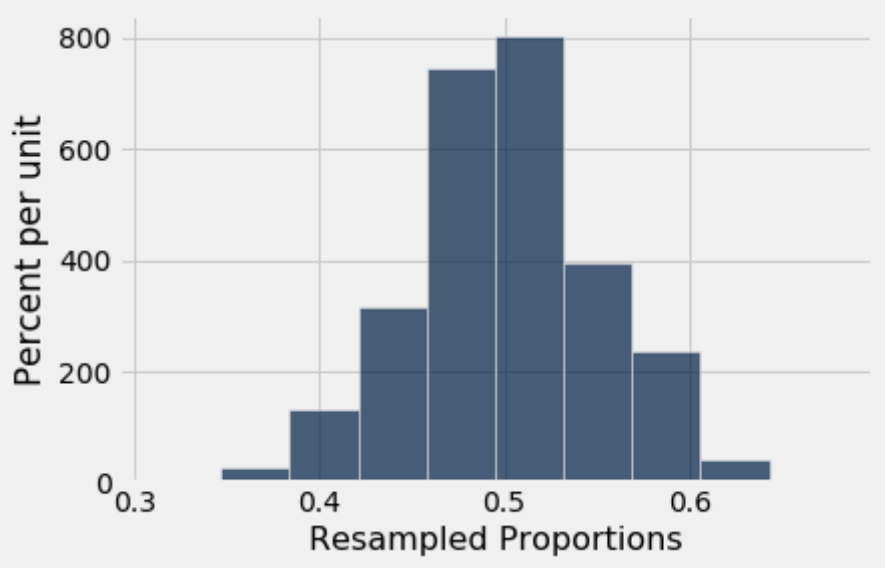
```
In your binary sample there were 50 ones and 50 zeros. 1*50 + 0*50 = 50
The sum of values in your sample was 50, divided by the number of item
s, 100, gives us a mean of 0.5
The proportion of ones in your sample was 50 ones, divided by the numbe
r of items, 100, gives us a value of 0.5
Those values are equal!
```

In [48]:
```
# TEST
# Number of ones and number of zeros in your `binary_sample` should sum
 to 100
int(number_of_ones + number_of_zeros) == 100
```

Out[48]:
```
True
```

Since the proportion of ones is the same as the mean, the Central Limit Theorem applies! That is, if we resample our sample a lot of times, the distribution of the proportion of ones in our resamples will be roughly normal, with a predictable center and spread!

In [49]:
```
# Just run this cell
resampled_proportion_of_ones = make_array()

for i in np.arange(5000):
    resample = Table().with_column("Value", binary_sample).sample()
    resample_proportion_ones = resample.where("Value", 1).num_rows / res
ample.num_rows
    resampled_proportion_of_ones = np.append(resampled_proportion_of_one
s, resample_proportion_ones)

Table().with_column('Resampled Proportions', resampled_proportion_of_one
s).hist()
```



Let's generate a table where each row has a different number of ones and zeros that we'll use for the following parts.

In [50]:
```
# Just run this cell
possible_number_ones = np.arange(sample_size + 1)
possible_number_zeros = sample_size - possible_number_ones

possibilities_table = Table().with_columns("Values of One", possible_num
ber_ones, "Values of Zero", possible_number_zeros)
possibilities_table.show(5)
```

Values of One	Values of Zero
0	100
1	99
2	98
3	97
4	96

... (96 rows omitted)

Question 3

The second important property of binary distributions is that the standard deviation of every binary distribution is equal to:

$$\sqrt{\text{proportion\_ones} * \text{proportion\_zeros}}$$

While this property is useful in some cases, a more useful extension of this property is that it tells us that the maximum standard deviation for a binary distribution is 0.5!

Let's explore why that is the case!

Complete the `binary_std_formula` function below so that it returns the standard deviation of a binary distribution according to the formula above.

```
BEGIN QUESTION
name: q2_3
manual: false
```

In [51]:
```
def binary_std_formula(row):
    num_ones = row.item("Values of One")
    num_zeros = row.item("Values of Zero")

    sum_ones_and_zeros = num_ones + num_zeros # SOLUTION
    prop_ones = num_ones / sum_ones_and_zeros # SOLUTION
    prop_zeros = num_zeros / sum_ones_and_zeros # SOLUTION
    return np.sqrt(prop_ones * prop_zeros) # SOLUTION

# DON'T DELETE/MODIFY ANY OF THE LINES BELOW IN THIS CELL
possibilities_table = possibilities_table.with_column("Formula SD", poss
ibilities_table.apply(binary_std_formula))
possibilities_table.show(5)
```

Values of One	Values of Zero	Formula SD
0	100	0
1	99	0.0994987
2	98	0.14
3	97	0.170587
4	96	0.195959

... (96 rows omitted)

In [52]:
```
# TEST
np.round(possibilities_table.column("Formula SD").item(4), 3) == 0.196
```

Out[52]: True

Here's another function that takes in a row object from a table, generates a sample that has the same number of ones and zeros as the row specifies, and then returns the standard deviation of that table. You should be able to understand exactly what this function does! It also does the same thing as above, where we return the standard deviation, but we just use `np.std` for this function.

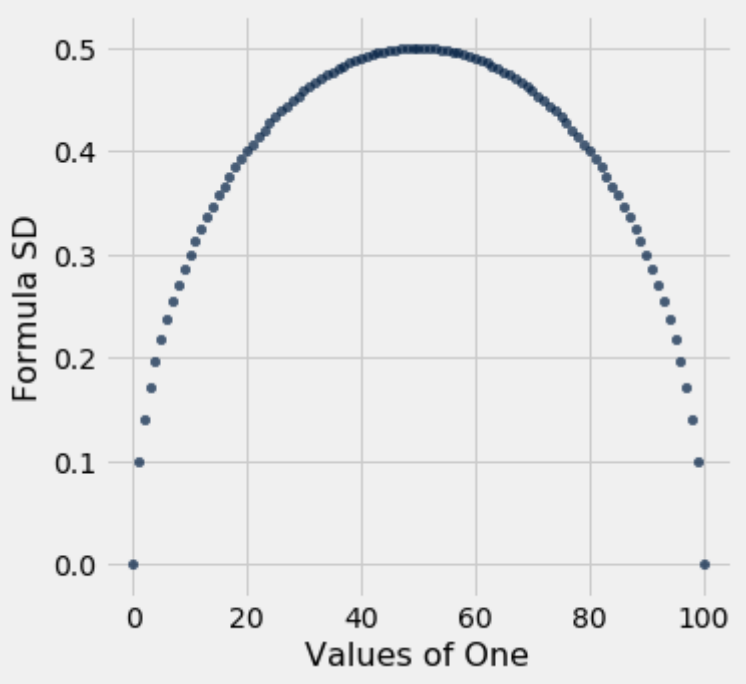
In [53]:
```
# Just run this cell
def binary_std(row):
    values = make_array()
    for i in np.arange(row.item("Values of One")):
        values = np.append(values, 1)
    for i in np.arange(row.item("Values of Zero")):
        values = np.append(values, 0)
    return np.std(values)

possibilities_table = possibilities_table.with_column("Empirical SD", po
ssibilities_table.apply(binary_std))
possibilities_table.show(5)
```

Values of One	Values of Zero	Formula SD	Empirical SD
0	100	0	0
1	99	0.0994987	0.0994987
2	98	0.14	0.14
3	97	0.170587	0.170587
4	96	0.195959	0.195959

... (96 rows omitted)

All the values are the same! Let's see what this formula means!

In [54]:
```
# Just run this cell
possibilities_table.scatter("Values of One", "Formula SD")
```



What a beautiful curve!

Looking at that curve, we can see that maximum value is 0.5, which occurs in the middle of the distribution, when the two categories have equal proportions (proportion of ones = proportion of zeros = $\frac{1}{2}$).

(OPTIONAL, NOT IN SCOPE) Logarithmic Plots

A kind of visualization you will frequently encounter as a data scientist is a scatter plot or line plot that uses a logarithmic scale. This **Optional** section will cover how to read and generate logarithmic plots. Since this is optional, there is no autograded/free response questions for these sections. Just read, run cells, and explore.

What is a logarithm? A logarithm helps us find the inverse of an equation that uses exponentials. Specifically, if

$$a^y = x$$

Then

$$\log_a x = y$$

The most commonly used a, which is known as the base of the logarithm, is e, which is equivalent to about 2.718, or 10 (for powers of 10).

We can use `numpy` to take logs in Python! By default, np.log uses a base of e.

In [55]:
```
make_array(np.log(np.e), np.log(np.e**2), np.log(100))
```

Out[55]:
```
array([1.        , 2.        , 4.60517019])
```

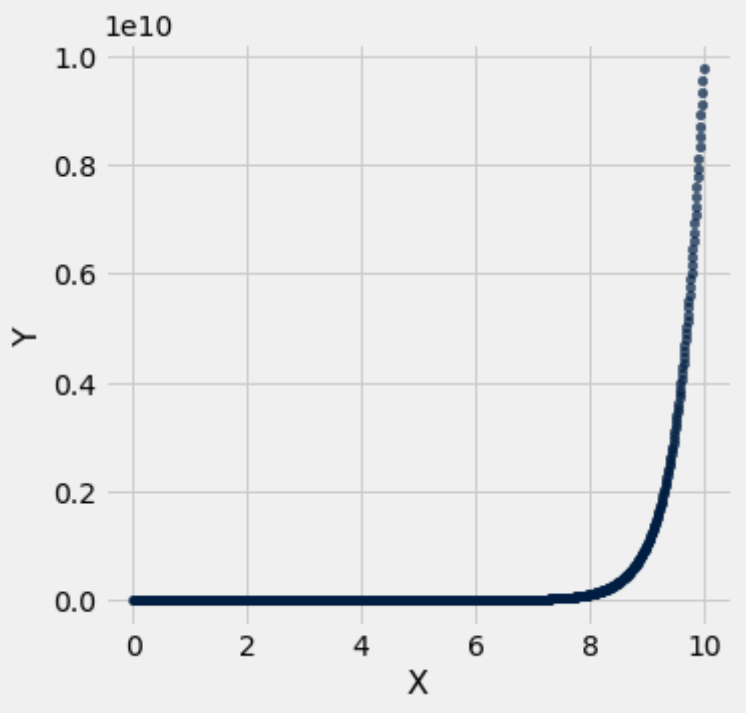
Back to the visualization: when we are plotting trends that grow exponentially, such as the line

$$y = e^x$$

our y-axis needs to have a large range of values, which makes it difficult to understand.

Let's see what this looks like:

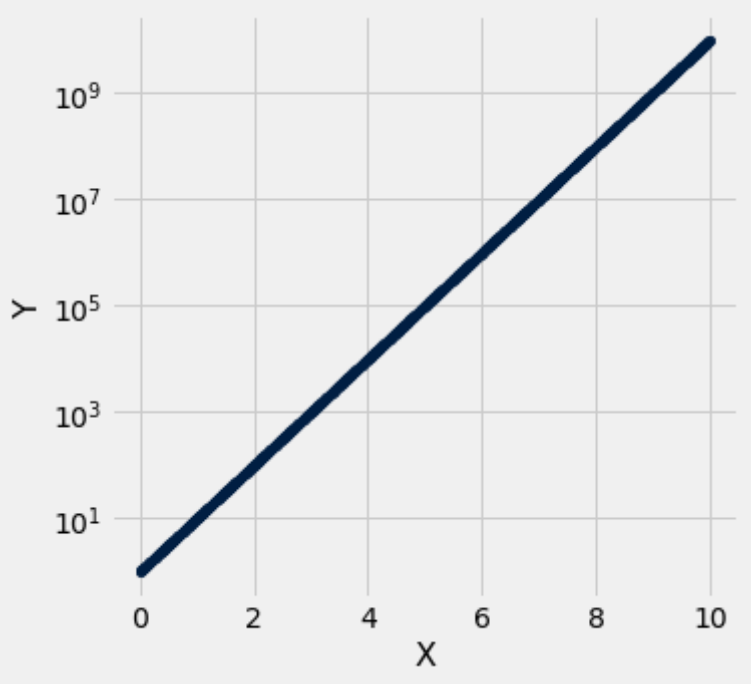
In [56]:
```
x = np.arange(0, 10, 1/100)
y = 10 ** x

Table().with_columns("X", x, "Y", y).scatter(0,1)
```



Note that since 10^{10} is so big, we can't really see what's happening at all to the y values when they have x values below 8.

One solution to this to change our y and/or x axis so that instead of having even spaces between the tick marks, our marks grow by an uneven factor. We do this by making the tick marks go on a logarithmic scale, and we'll then be able to understand our data better!

In [57]:
```
Table().with_columns("X", x, "Y", y).scatter(0,1)
plt.yscale("log")
```



Now we can tell what's happening to the y values for every x value!

Note how the y values start at $10^0 = 1$, and increase by a *factor* of 10 each mark - the next mark is $10^1 = 10$, then $10^2 = 100$.

You still read this plot like a normal plot, so at a value of $x = 5$, $y = 10^5 = 10000$.

How do you calculate intermediate values?

At a value like $x = 2.5$ it looks like the y value is somewhere in-between 10^1 and 10^3. In this graph with a logarithmic scale, you would say that $y = 10^{2.5} \approx 316$.

When visualizing data about the spread of diseases, you will commonly run into plots with logarithmic scales, such as this example from the New York Times. Make sure to always know what the scales of the data are!

Image is from https://www.nytimes.com/2020/03/20/health/coronavirus-data-logarithm-chart.html (https://www.nytimes.com/2020/03/20/health/coronavirus-data-logarithm-chart.html)

3. Submission

Once you're finished, select "Save and Checkpoint" in the File menu and then execute the `submit` cell below. The result will contain a link that you can use to check that your assignment has been submitted successfully. If you submit more than once before the deadline, we will only grade your final submission. If you mistakenly submit the wrong one, you can head to okpy.org (https://okpy.org/) and flag the correct version. To do so, go to the website, click on this assignment, and find the version you would like to have graded. There should be an option to flag that submission for grading!

In [58]:
```
_ = ok.submit()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-58-cc46ca874451> in <module>
----> 1 _ = ok.submit()

NameError: name 'ok' is not defined
```

In [59]:
```
# For your convenience, you can run this cell to run all the tests at once!
import os
print("Running all tests...")
_ = [ok.grade(q[:-3]) for q in os.listdir("tests") if q.startswith('q') and len(q) <= 10]
print("Finished running all tests.")
```

```
Running all tests...
Finished running all tests.
```