finalProject

December 19, 2025

[103]:
```
# Initialize Otter
import otter
grader = otter.Notebook("finalProject.ipynb")
```

1 Final Project: Regression Inference & Classification

Welcome to the Final Project for Data Science for All! This is the final project for our course and with this project you will get to explore a dataset of your choice. By the end of the project, you will have some experience with:

1. Finding a dataset of interest.
2. Performing some exploratory analysis using linear regression and inference.
3. Building a k-nearest-neighbors classifier.
4. Testing a classifier on data.

1.0.1 Logistics

Rules. Don't share your code with anybody. You are welcome to discuss your project with other students, but don't share your project details or copy a project from the internet. This project should be YOUR OWN (code, etc.). If you do base your project on something you learned online or through a generative AI tool such as ChatGPT, make sure to check with your instructor before getting started and reference your sources as part of this project. The experience of solving the problems in this project will prepare you for the final exam (and life). During the final lab session, you will have a chance to share with the whole class.

Support. You are not alone! Come to lab hours, tutoring hours, office hours, and talk to your classmates. If you're ever feeling overwhelmed or don't know how to make progress, we are here to help! Don't hesitate to send an email.

Advice. Develop your project incrementally. To perform a complicated table manipulation, break it up into steps, perform each step on a different line, give a new name to each result, and check that each intermediate result is what you expect. Don't hesitate to add more names/variables or functions if this helps with your analysis or classifier development. Also, 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.

To get started, load `datascience`, `numpy`, and `plots`.

Reading:

- Inference for Regression
- Classification

[104]:
```
# Don't change this cell; just run it. If you need additional libraries for␣
 ↪your project, you can add them to this cell.

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 matplotlib import patches
from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
```

2 1. Picking a Dataset

In this project, you are exploring a dataset of your choice. The dataset should be large enough: multiple individuals (rows) with multiple attributes (columns) such that we can try to make a prediction based on the known information in this dataset using linear regression and/or classification. In this first section you will: - find a data set that you are interested in - record the source of where you found it - save it as a .csv file in the same folder as your jupyter notebook. - make sure you can read it in as a table and that your dataset represents a large enough sample for investigating the possible use of regression inference and clasification - Explore the data using visualization techniques learned in this course - Formulate what (which attributes) you would like to investigate using a linear regression model - Formulate what question you would like to answer with a classifier based on this dataset. For example: (1) Is this movie a thriller or a comedy? (2) Is this amazon order Fraudulant or not? (3) Does this patient have cancer or not? See section in the book for more details on Classification - Discuss your choice with you instructor and get approval to get started with section 2

Note 1: If you need guidance on where and how to find a dataset, ask your instructor for help!

Note 2: Your final project conclusion does not necessarily need to show that you have a good regression model or classifier to make predictions! What is important is your own analysis of its potential and limitations when investigating the dataset for making predictions using these techniques

Question 1.1 In the cell below: 1. Read in the dataset you chose as a table 2. Edit the comment to describe where you found this dataset

[105]:
```
# Edit this comment to describe where you found this dataset
# Load your dataset into a table

```

```
my_data_raw = Table().read_table("breast-cancer.csv")
my_data_raw
```

[105]:
```
ID      | Clump Thickness | Uniformity of Cell Size | Uniformity of Cell Shape |
Marginal Adhesion | Single Epithelial Cell Size | Bare Nuclei | Bland Chromatin
| Normal Nucleoli | Mitoses | Class
1000025 | 5               | 1                       | 1                        |
1                 | 2                           | 1           | 3
| 1               | 1       | 0
1002945 | 5               | 4                       | 4                        |
5                 | 7                           | 10          | 3
| 2               | 1       | 0
1015425 | 3               | 1                       | 1                        |
1                 | 2                           | 2           | 3
| 1               | 1       | 0
1016277 | 6               | 8                       | 8                        |
1                 | 3                           | 4           | 3
| 7               | 1       | 0
1017023 | 4               | 1                       | 1                        |
3                 | 2                           | 1           | 3
| 1               | 1       | 0
1017122 | 8               | 10                      | 10                       |
8                 | 7                           | 10          | 9
| 7               | 1       | 1
1018099 | 1               | 1                       | 1                        |
1                 | 2                           | 10          | 3
| 1               | 1       | 0
1018561 | 2               | 1                       | 2                        |
1                 | 2                           | 1           | 3
| 1               | 1       | 0
1033078 | 2               | 1                       | 1                        |
1                 | 2                           | 1           | 1
| 1               | 5       | 0
1033078 | 4               | 2                       | 1                        |
1                 | 2                           | 1           | 2
| 1               | 1       | 0
… (673 rows omitted)
```

[106]:
```
grader.check("q1_1")
```

[106]:
```
q1_1 results: All test cases passed!
```

Question 1.2: In the following cell, describe each of the variables in the dataset. Are they categorical or numerical? How many observations are there? Add a code cell below this to show how you found the correct number of observations programmatically.

Categorical: Gender, Smoker (boolean), Numerical: Age (years), BMI (Body Mass Index), daily steps, sleep hours, water intake (Liters).

Question 1.3: The dependent or response variable of interest is the variable that we will try to classify later in this project. This is the variable that should have two levels that will be used in classification later.

For example: (1) Is this movie a thriller or a comedy? (2) Is this amazon order Fraudulent or not? (3) Does this patient have cancer or not? See section in the book for more details on Classification.

In the code cell below, make any necessary adjustments to your data so that the variable is formatted in this way. Then assign the variable `var` to the column `label` of your table that contains the observations of this variable (note: the label should be a string, so don't forget the quotes!)

[107]:
```
# If you need to make adjustments to your data so that the variable is␣
 ↪formatted in 2 levels add the needed code below, before setting var
my_data_raw = my_data_raw.drop("ID")

def randomize_column(a):
    return a + np.random.normal(0.0, 0.09, size=len(a))

jittered = Table()
for label in my_data_raw.labels:
    data = my_data_raw.column(label)
    if (label != "Class"):
        data = randomize_column(data)
    jittered = jittered.with_column(label, data)
jittered
```

[107]:
```
Clump Thickness | Uniformity of Cell Size | Uniformity of Cell Shape | Marginal
Adhesion | Single Epithelial Cell Size | Bare Nuclei | Bland Chromatin | Normal
Nucleoli | Mitoses  | Class
4.94692         | 0.940261                | 1.10546                  | 0.894836
| 1.96044                     | 0.964895    | 2.94449         | 0.923358
| 0.884932 | 0
5.03466         | 3.9469                  | 3.91138                  | 4.915
| 7.03717                     | 10.0776     | 3.00029         | 2.03183
| 0.962331 | 0
2.96334         | 1.06211                 | 1.09925                  | 1.05349
| 1.96293                     | 2.09843     | 3.1434          | 0.926624
| 0.853773 | 0
5.91934         | 7.90119                 | 7.85439                  | 1.02937
| 2.83762                     | 3.90747     | 2.89229         | 6.92494
| 1.08516  | 0
4.03358         | 0.950115                | 0.988001                 | 2.9601
| 1.90808                     | 1.16051     | 2.96979         | 0.980237
| 0.837633 | 0
7.92992         | 10.1347                 | 9.99525                  | 7.94465
| 6.94218                     | 10.0567     | 9.02266         | 6.82906
| 0.998263 | 1
1.01959         | 1.00016                 | 1.1413                   | 0.953243
```

```
| 2.07382                    | 10.0769     | 2.98775          | 0.922442
| 1.02226  | 0
2.10282          | 1.01533                | 1.95159                  | 1.00992
| 2.1039                     | 0.883069    | 2.77381          | 1.16177
| 1.13023  | 0
2.05389          | 1.1697                 | 0.998044                 | 1.04636
| 1.98988                    | 1.18155     | 1.03942          | 1.06819
| 4.95625  | 0
4.02261          | 1.8564                 | 1.0991                   | 1.06318
| 2.12725                    | 0.945871    | 2.03874          | 1.0056
| 1.09213  | 0
… (673 rows omitted)
```

[108]:
```
var = "Class"
var
```

[108]:
```
'Class'
```

[109]:
```
grader.check("q1_3")
```

[109]:
```
q1_3 results: All test cases passed!
```

Now, we are ready to investigate our data visually!

Question 1.4: Think about the numerical variables in the dataset that might be related to each other. In the cell below, make three different scatter plots that show the relationship between different variables while also displaying how each case is classified.

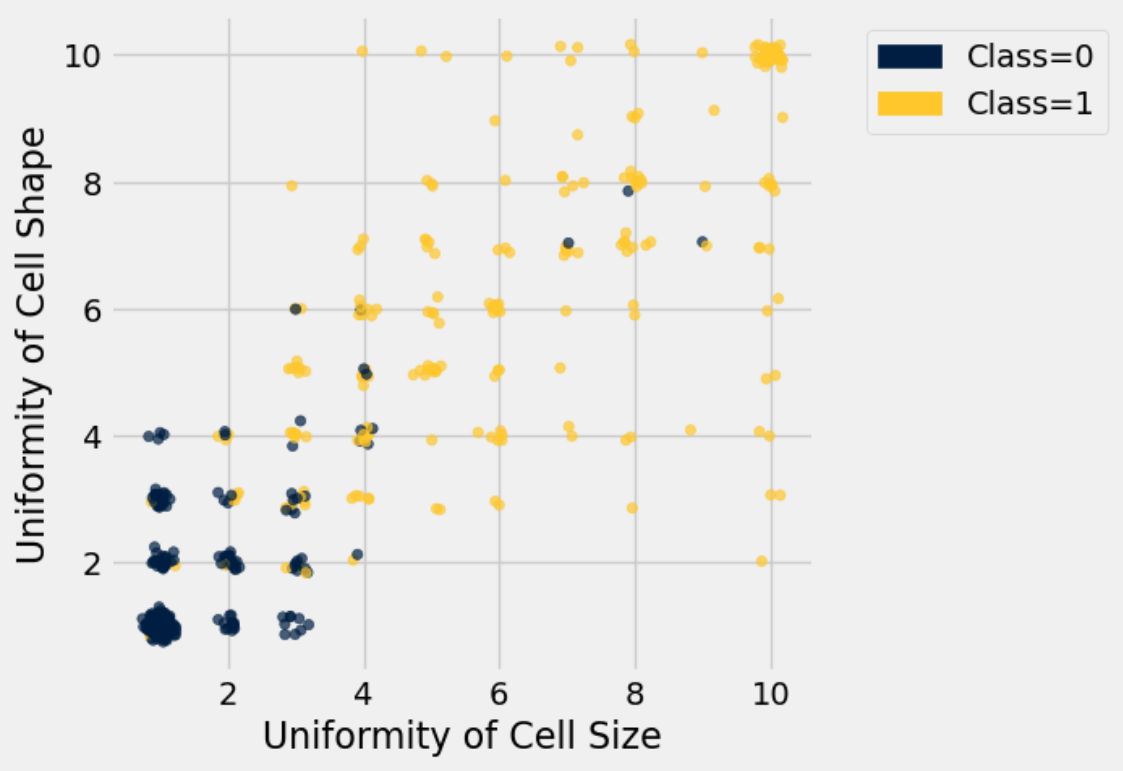
Use the following line of code:

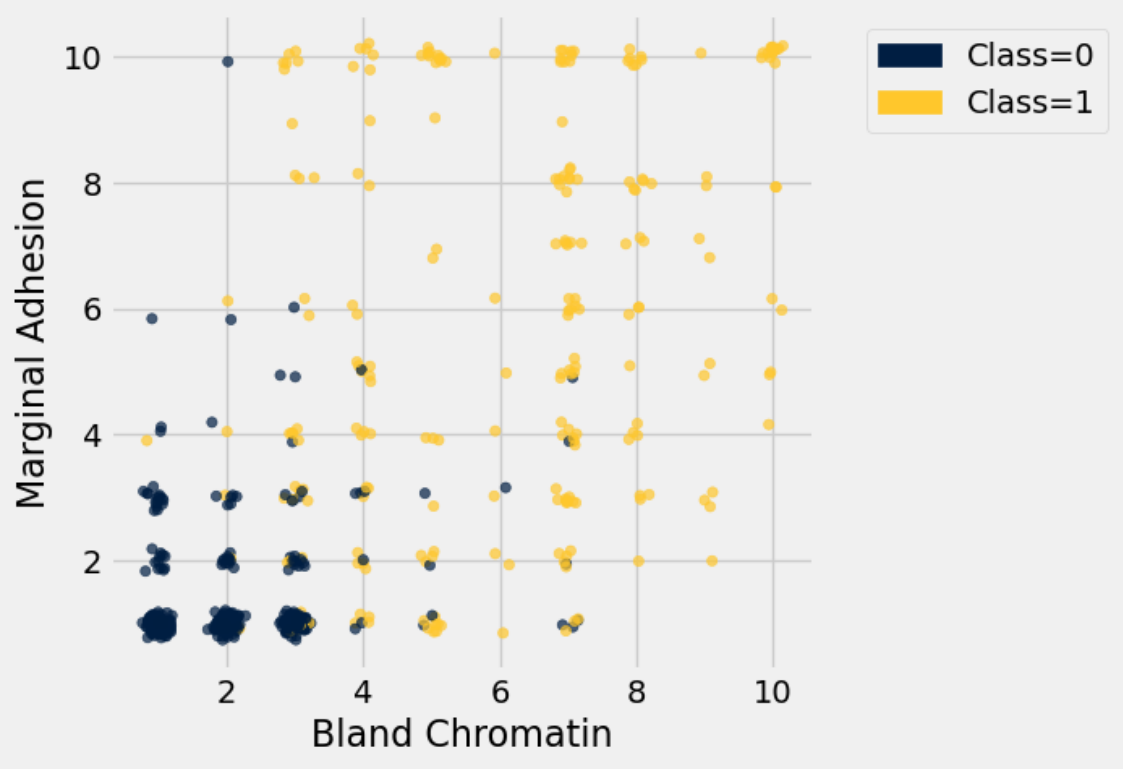
my_data.scatter(`Column 1`, `Column 2`, group=`label`)

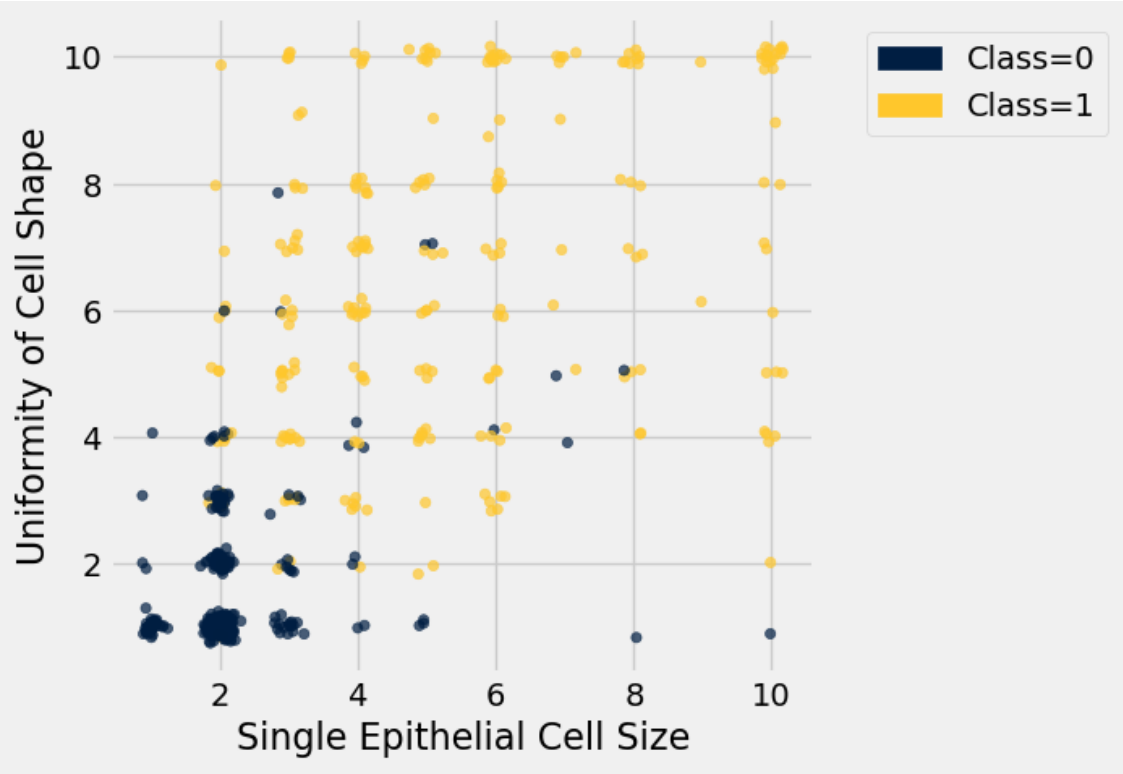
Replace `Column 1` and `Column 2` with the correct column names of numerical variables(features) you would like to investigate. Replace `label` with the column name of the categorical variable you would like to try to classsify.

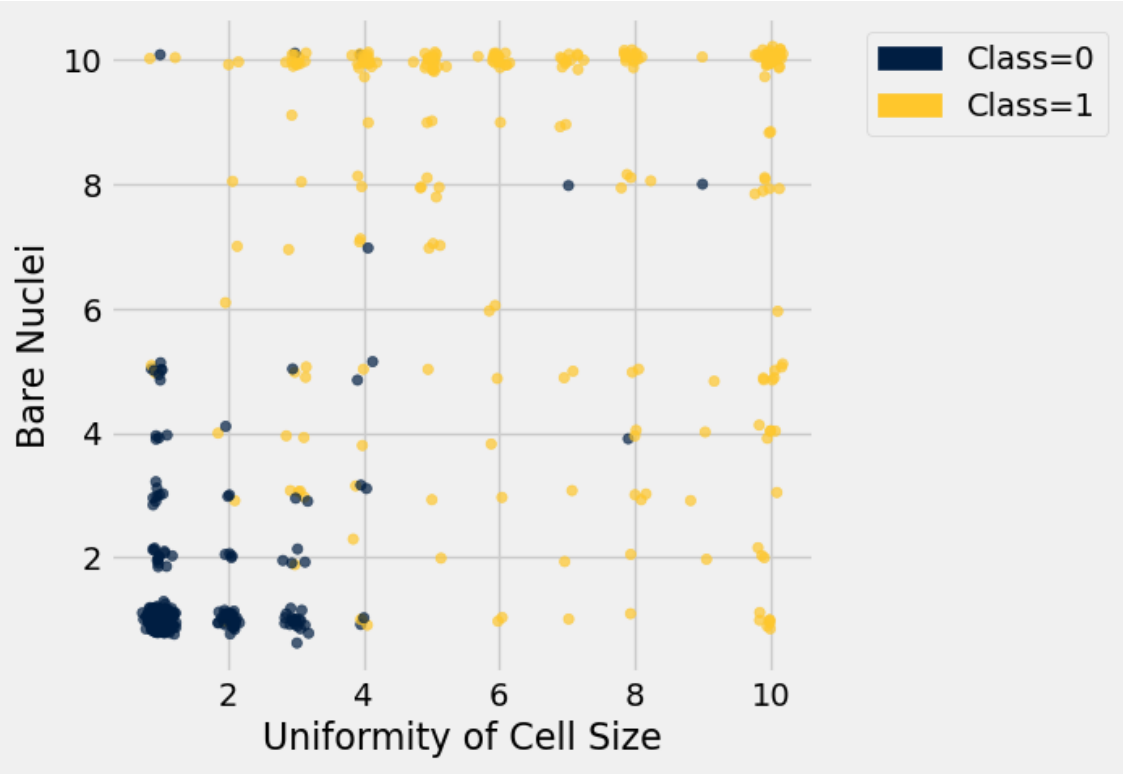
Note: The commented code in the cell below is sample code for this.

[110]:
```
# Here is the code placeholder to use, uncomment the line and adjust according␣
↪to the names if the columns in your dataset:
# my_data.scatter("Column 1", "Column 2", group="label")
jittered.scatter("Uniformity of Cell Size", "Uniformity of Cell Shape",␣
↪group="Class")
jittered.scatter(6, 3, group="Class")
jittered.scatter(4, 2, group="Class")
jittered.scatter(1, 5, group="Class")
#my_data_raw.sample(500).scatter("water intake (liters)", "daily steps", group␣
↪= "Healthy BMI?")
#my_data_raw.sample(500).scatter("daily steps", "cholesterol", group="Healthy␣
↪BMI?")
```


Class=0
Class=1
Uniformity of Cell Shape
Uniformity of Cell Size
2
4
6
8
10


Class=0
Class=1
Marginal Adhesion
Bland Chromatin
2
4
6
8
10


Class=0
Class=1
Uniformity of Cell Shape
Single Epithelial Cell Size
2
4
6
8
10


Class=0
Class=1
Bare Nuclei
Uniformity of Cell Size
2
4
6
8
10

Question 1.5 Describe the three plots from the last question. For each plot, note whether the relationship appears to be linear and whether it is a positive or negative association. Which of the three plots will you look at for linear regression?

The relationships are subtle, but all four scatterplots seem to have a slight positive linear association. The first scatterplot has the clearest positive linear relationship. For linear regression, I would use the first plot comparing uniformity of cell size and uniformity of cell shape.

Question 1.6 In the cell below, formulate the question you would like to try to answer with a classifier that you plan to build.

Does this person have breast cancer based on these traits?

Question 1.7 Set the variable **instructor_signed_off** to 'YES' if you have checked in with your instructor during lab hours.

[111]:
```
instructor_signed_off = "YES"
```

[112]:
```
grader.check("q1_7")
```

[112]:
```
q1_7 results: All test cases passed!
```

3 2. Regression Inference

To get started, reduce the table with relevant data that you would like to use to evaluate a linear regression model to make a prediction. In this section, you will evaluate this model and set up a hypothesis test to check if there is true correlation/linear association. You will analyze residuals, confidence interval and prediction lines of best fit.

Question 2.1 Copy the cell where you loaded the dataset in section 1, reduce the table to only include the relevant data for what you would like to use in your regression model. The table should only have 2 columns of intest since this is simple linear regression.

[113]:
```
# Copy the cell where you loaded the dataset in section 1 and reduce the table␣
↪to only include the relevant data for linear regression
# You may have to clean your data to get rid of outliers

my_data = Table().read_table("breast-cancer.csv")
my_data = my_data.select("Uniformity of Cell Size", "Uniformity of Cell Shape")
my_data = my_data.where(0, are.below(10))
my_data = my_data.where(1, are.below(10))
my_data.show(5)
```

```
<IPython.core.display.HTML object>
```

[114]:
```
# As usual, let's investigate our data visually before analyzing it numerically.
↪
```

```
# Just run this cell to plot the relationship between the 2 attribute/columns.
# The scatter plot should look similar to the one you plotted for 1.4.
my_data.scatter(0, 1, fit_line=True)
```

[115]:
```
grader.check("q2_1")
```

[115]:
```
q2_1 results: All test cases passed!
```

Question 2.2:

Use the functions given to assign the correlation between the 2 attributes to the variable `cor`.

The function `correlation` takes in three arguments, a table `tbl` and the labels of the columns you are finding the correlation between, `col1` and `col2`.

[116]:
```
def standard_units(arr):
    return (arr- np.mean(arr)) / np.std(arr)

def correlation(tbl, col1, col2):
```

```
    r = np.mean(standard_units(tbl.column(col1)) * standard_units(tbl.
 ↪column(col2)))
    return r

cor = correlation(my_data, "Uniformity of Cell Size", "Uniformity of Cell␣
 ↪Shape")
cor
```

[116]: 0.88388960272282335

[117]:
```
grader.check("q2_2")
```

[117]: q2_2 results: All test cases passed!

Can you see a correlation between the 2 variables? If in this sample, we found a linear relation between the two variables, would the same be true for the population? Would it be exactly the same linear relation? Could we predict the response of a new individual who is not in our sample?

Question 2.3: Writing Hypotheses.

Suppose you think the slope of the true line of best fit for the 2 variables is not zero: that is, there is some correlation/association between them. To test this claim, we can run a hypothesis test! Define the null and alternative hypothesis for this test.

The null hypothesis would be that Uniformity of cell size and shape are correlated due to random chance. The alternate hypothesis is that they are correlated due to some other factor(s).

Question 2.4:

Maria says that instead of finding the slope for each resample, we can find the correlation instead, and that we will get the same result. Why is she correct? What is the relationship between slope and correlation?

Maria is correct because the slope and the correlation are directly proportional to each other when the variables being compared have fixed spreads.

Question 2.5: Define the function `one_resample_r` that performs a bootstrap and finds the correlation between the 2 variables in the resample. `one_resample_r` should take three arguments, a table `tbl` and the labels of the columns you are finding the correlation between, `col1` and `col2`.

[118]:
```
def one_resample_r(tbl, col1, col2):
    boot_data = my_data.sample()
    return correlation(boot_data, col1, col2)


# Uncomment the line of code below and change `Column 1` and `Column 2` to␣
 ↪match your dataset.

one_resample = one_resample_r(my_data, "Uniformity of Cell Size", "Uniformity␣
 ↪of Cell Shape")
one_resample
```

[118]:
```
0.89755838652264375
```

[119]:
```
grader.check("q2_5")
```

[119]:
```
q2_5 results: All test cases passed!
```

Question 2.6: Generate 1000 bootstrapped correlations for the 2 variables, store your results in the array `resampled_correlations`, and plot a histogram of your results.

[120]:
```
resampled_correlations = make_array()
for i in np.arange(1000):
    one_resample = one_resample_r(my_data, "Uniformity of Cell Size",␣
 ↪"Uniformity of Cell Shape")
    resampled_correlations = np.append(resampled_correlations, one_resample)

table = Table().with_column("Uniformity of Cell Size vs Uniformity of Cell␣
 ↪Shape", resampled_correlations)
table.hist()
resampled_correlations
```

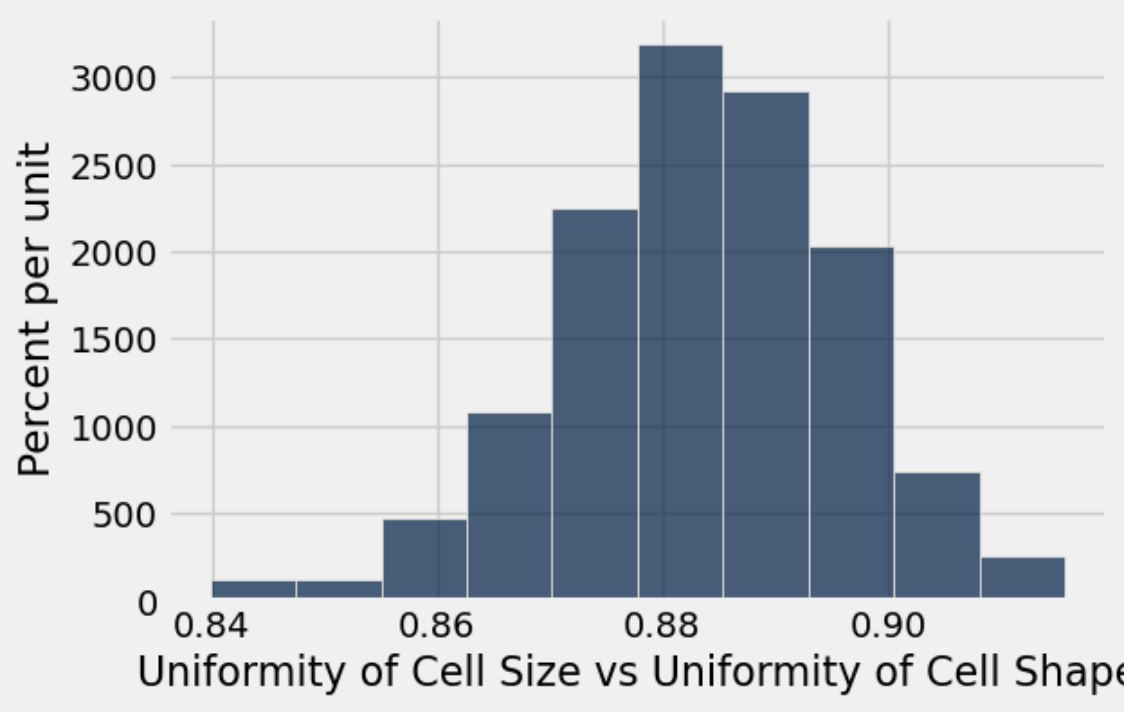
[120]:
```
array([ 0.87151422,  0.8944222 ,  0.87785347,  0.89740507,  0.89095046,
        0.90642658,  0.89959897,  0.8844851 ,  0.8922539 ,  0.86777161,
        0.88975323,  0.88447735,  0.88034824,  0.88011668,  0.87686392,
        0.88106897,  0.88500982,  0.86079759,  0.86396913,  0.90021901,
        0.8807195 ,  0.87487917,  0.89256796,  0.88603553,  0.885899  ,
        0.87495317,  0.88113714,  0.89009426,  0.88637434,  0.87699252,
        0.87088622,  0.89646003,  0.87600125,  0.88415095,  0.87985165,
        0.88220893,  0.89677947,  0.87039238,  0.90379374,  0.87517501,
        0.87693339,  0.91316082,  0.88902476,  0.90810158,  0.88878789,
        0.90799712,  0.89053813,  0.88430137,  0.88010724,  0.87857374,
        0.8694099 ,  0.8833243 ,  0.89840207,  0.87085294,  0.87204674,
        0.9099361 ,  0.87433067,  0.88702543,  0.88938498,  0.86905337,
        0.89954792,  0.86926532,  0.88427808,  0.89876618,  0.89320696,
        0.88657722,  0.86819294,  0.8788051 ,  0.88147659,  0.90492979,
        0.89665886,  0.88921653,  0.88792096,  0.88503656,  0.89310694,
        0.89247784,  0.88320747,  0.88877798,  0.87876419,  0.87179964,
        0.87612477,  0.88427607,  0.87892406,  0.87091803,  0.88576072,
        0.8924459 ,  0.88613957,  0.87670546,  0.85613702,  0.87647344,
        0.88701036,  0.8937458 ,  0.85969802,  0.87746526,  0.89888648,
        0.90083304,  0.87730978,  0.88269235,  0.8742082 ,  0.87362972,
        0.90045926,  0.88395523,  0.89066563,  0.89424443,  0.87683822,
        0.88131572,  0.86573623,  0.87108803,  0.87418993,  0.85825666,
        0.87183056,  0.90405201,  0.90182421,  0.89436302,  0.89777813,
        0.88647651,  0.88291999,  0.89465982,  0.88119017,  0.87415137,
        0.89460483,  0.86920053,  0.8838141 ,  0.88155188,  0.87156527,
        0.88929789,  0.88190356,  0.8874756 ,  0.89082552,  0.88658136,
        0.86738414,  0.87089895,  0.87546406,  0.88863444,  0.87807522,
```

```
0.87513579,  0.87105862,  0.85618918,  0.86694548,  0.87600324,
0.87640104,  0.88239008,  0.90463442,  0.87428868,  0.87666355,
0.88970055,  0.87983876,  0.89844309,  0.87475045,  0.89439141,
0.88417277,  0.87065726,  0.9035536 ,  0.87322706,  0.87210367,
0.89293909,  0.87698023,  0.88490403,  0.90902738,  0.8869977 ,
0.8804077 ,  0.89085666,  0.88372422,  0.88857573,  0.87396899,
0.88424981,  0.89576122,  0.89738878,  0.87771262,  0.87191494,
0.8797358 ,  0.88135403,  0.91092381,  0.8542033 ,  0.87430186,
0.90484963,  0.88658195,  0.88310032,  0.89137347,  0.90003278,
0.90444278,  0.88395256,  0.88551674,  0.85258045,  0.86685493,
0.8657152 ,  0.89388265,  0.88653579,  0.89921387,  0.89282739,
0.89763108,  0.87582903,  0.90034771,  0.87697407,  0.86329757,
0.89607141,  0.87767276,  0.90359221,  0.88085933,  0.8882255 ,
0.88410848,  0.89674085,  0.88584558,  0.87367178,  0.88116317,
0.85177967,  0.89391548,  0.85074317,  0.88700263,  0.87997021,
0.89201877,  0.87900007,  0.88830531,  0.85818188,  0.88480151,
0.88023253,  0.89618373,  0.87413757,  0.88690721,  0.87521154,
0.88534981,  0.88181597,  0.88110758,  0.86240963,  0.87091739,
0.88713283,  0.90631846,  0.86640763,  0.89759533,  0.88159501,
0.87856761,  0.87859192,  0.8747535 ,  0.87495101,  0.88491117,
0.87580508,  0.87433609,  0.87540224,  0.88453161,  0.88509673,
0.89849582,  0.87466257,  0.88790159,  0.86161094,  0.88213601,
0.90344959,  0.89143945,  0.84577548,  0.86832308,  0.88179371,
0.88533666,  0.8578328 ,  0.87855226,  0.88788747,  0.88965831,
0.86279919,  0.88059294,  0.88537504,  0.87540341,  0.89324299,
0.89866793,  0.88306747,  0.89286673,  0.91101736,  0.87993165,
0.86372667,  0.8923827 ,  0.9001491 ,  0.87999334,  0.88836241,
0.8673789 ,  0.89647021,  0.88322064,  0.8769175 ,  0.88987076,
0.88610103,  0.87716878,  0.8886348 ,  0.8677805 ,  0.87225241,
0.85691069,  0.89954302,  0.85561258,  0.8585057 ,  0.88690969,
0.88673544,  0.87719783,  0.89723173,  0.89011741,  0.87891807,
0.87866792,  0.86614832,  0.88460337,  0.85698226,  0.87794426,
0.90080659,  0.86312566,  0.89349133,  0.88568171,  0.88258931,
0.88432486,  0.88445873,  0.87936007,  0.89623216,  0.88270686,
0.86692327,  0.90048413,  0.86523084,  0.88586706,  0.88497253,
0.88596552,  0.90262414,  0.87562965,  0.86634504,  0.8821297 ,
0.89658246,  0.89914744,  0.8905695 ,  0.88318477,  0.88540981,
0.89632125,  0.89682961,  0.872598  ,  0.88093359,  0.89800501,
0.88588089,  0.8665087 ,  0.90180511,  0.89482993,  0.91099956,
0.88580698,  0.87212935,  0.88362592,  0.87736215,  0.8817479 ,
0.88809409,  0.89241834,  0.87320496,  0.89414599,  0.8865333 ,
0.88794173,  0.87553424,  0.88790888,  0.87814677,  0.89902373,
0.89119093,  0.86242534,  0.89044003,  0.87956263,  0.84548099,
0.89199254,  0.86727179,  0.88259557,  0.87884296,  0.87537583,
0.88320501,  0.86974583,  0.86435157,  0.89965178,  0.87660424,
0.87800348,  0.87590462,  0.8809122 ,  0.88340933,  0.89128701,
0.89398912,  0.86247741,  0.89011683,  0.86463089,  0.87565582,
```

```
0.89468974,  0.88744673,  0.88875089,  0.90211241,  0.8828351 ,
0.89376695,  0.88148963,  0.88278847,  0.87947105,  0.88282841,
0.89660617,  0.88208651,  0.88070055,  0.8680763 ,  0.90180444,
0.87672204,  0.91307588,  0.90954423,  0.88953381,  0.8905619 ,
0.8775376 ,  0.89262957,  0.88499415,  0.89183986,  0.89219869,
0.87260188,  0.88553205,  0.89088818,  0.90249291,  0.89244196,
0.8737803 ,  0.87514944,  0.8743179 ,  0.88601902,  0.86780802,
0.8738654 ,  0.8991614 ,  0.87439974,  0.88197852,  0.8850892 ,
0.85788506,  0.91182128,  0.84791784,  0.88461275,  0.88740724,
0.89509632,  0.91529459,  0.8784171 ,  0.86322925,  0.88557519,
0.87690813,  0.8972898 ,  0.88144247,  0.88014055,  0.87771249,
0.87717008,  0.90160213,  0.91322024,  0.89573073,  0.87873435,
0.8998906 ,  0.90113334,  0.87976373,  0.88445651,  0.85921453,
0.87595849,  0.87647667,  0.89654814,  0.87670157,  0.89996333,
0.88573912,  0.85700278,  0.87484413,  0.85064762,  0.90334998,
0.89298678,  0.88342998,  0.90652679,  0.88542229,  0.89550023,
0.90232365,  0.87422752,  0.89207597,  0.89891982,  0.8779296 ,
0.87800763,  0.89590526,  0.89116277,  0.85414543,  0.87589321,
0.88591056,  0.89129972,  0.8565912 ,  0.89081545,  0.88980956,
0.89161322,  0.89427929,  0.85616058,  0.89478345,  0.90303763,
0.84729187,  0.87278472,  0.8694925 ,  0.87994863,  0.91021282,
0.89473016,  0.85478165,  0.89531619,  0.88827298,  0.89973589,
0.87812514,  0.86272748,  0.87952147,  0.88829168,  0.88782467,
0.8926275 ,  0.87494043,  0.86931997,  0.88897846,  0.88537574,
0.84347327,  0.86290692,  0.91202165,  0.89589044,  0.89409902,
0.89113066,  0.88277311,  0.87901163,  0.84313321,  0.86512569,
0.87224312,  0.87547775,  0.89487733,  0.89067328,  0.86974477,
0.86351136,  0.88319191,  0.84480089,  0.88153986,  0.88071527,
0.89244651,  0.87686771,  0.88796471,  0.88922813,  0.87224872,
0.88150648,  0.88464039,  0.86062994,  0.87487515,  0.89904085,
0.90616187,  0.90144945,  0.90332276,  0.87961037,  0.88807621,
0.90160434,  0.86504798,  0.87732946,  0.8885499 ,  0.90237021,
0.89086028,  0.89390623,  0.87633755,  0.87604029,  0.88521244,
0.90221708,  0.87079656,  0.88256329,  0.87627128,  0.89117673,
0.89669177,  0.89207351,  0.89373351,  0.86389495,  0.88450038,
0.90752301,  0.89163027,  0.89386078,  0.88123943,  0.88845023,
0.8826566 ,  0.87980142,  0.89821951,  0.88790756,  0.883404  ,
0.89789826,  0.86880293,  0.87978988,  0.86400973,  0.89109067,
0.89895573,  0.88323497,  0.86577183,  0.89407744,  0.88749561,
0.87287   ,  0.87291017,  0.87332951,  0.88362514,  0.88350591,
0.88422384,  0.86774529,  0.88225923,  0.88372334,  0.88343193,
0.88942959,  0.89688075,  0.87293114,  0.88125203,  0.88915639,
0.89103059,  0.87360525,  0.89462694,  0.88296882,  0.8798918 ,
0.89794389,  0.90186834,  0.88969891,  0.87968534,  0.89713935,
0.90599062,  0.85999678,  0.86743461,  0.88157099,  0.88887885,
0.88799715,  0.86511356,  0.89318644,  0.88253299,  0.91570769,
0.88298395,  0.89411531,  0.90057053,  0.87699334,  0.88147093,
```

```
0.87264342,  0.88941011,  0.89253334,  0.88193022,  0.86662778,
0.89149161,  0.8645792 ,  0.86360004,  0.89420841,  0.88027608,
0.86944206,  0.89754999,  0.89833823,  0.86937725,  0.89329454,
0.89867974,  0.89111414,  0.88593952,  0.88294939,  0.89066777,
0.89175387,  0.88918448,  0.89174745,  0.86594069,  0.89662847,
0.88913544,  0.88460075,  0.8942192 ,  0.87660325,  0.87599121,
0.90254154,  0.85807969,  0.84101758,  0.86074333,  0.88618027,
0.89271549,  0.89446222,  0.87944159,  0.88500419,  0.90132118,
0.88368265,  0.87979622,  0.88888372,  0.88660731,  0.87465471,
0.88162214,  0.86940458,  0.89063596,  0.86882257,  0.90748201,
0.88667561,  0.88148519,  0.91514578,  0.87617294,  0.8742565 ,
0.87914661,  0.87455424,  0.88780687,  0.89534383,  0.88321281,
0.90231991,  0.87893506,  0.89709115,  0.87693294,  0.8999796 ,
0.87861683,  0.8711308 ,  0.8804089 ,  0.91039879,  0.87416496,
0.87550696,  0.9067662 ,  0.89467168,  0.90289631,  0.86538298,
0.88806054,  0.87587467,  0.86685192,  0.87089882,  0.87834934,
0.88047025,  0.86094802,  0.85797179,  0.86711976,  0.88358582,
0.88598673,  0.88880204,  0.88796906,  0.87813632,  0.88143654,
0.8859377 ,  0.87081798,  0.87768499,  0.85883524,  0.8819143 ,
0.89280781,  0.86463153,  0.87680336,  0.88547686,  0.89020414,
0.88583271,  0.89736092,  0.86383108,  0.88654972,  0.87862666,
0.87949389,  0.89150702,  0.88414443,  0.89765099,  0.87131677,
0.88035862,  0.87791267,  0.89988499,  0.88633139,  0.89049543,
0.86961433,  0.89888558,  0.86860981,  0.86656631,  0.89343737,
0.88311039,  0.88917037,  0.88441864,  0.88754468,  0.89756978,
0.88277719,  0.88971518,  0.89223906,  0.9017518 ,  0.8791548 ,
0.86805404,  0.88313034,  0.87256148,  0.8917142 ,  0.88774516,
0.88216292,  0.87379895,  0.89305609,  0.87863676,  0.8976889 ,
0.89098437,  0.90052917,  0.86814226,  0.87854946,  0.8882113 ,
0.88762278,  0.88642985,  0.88776264,  0.9033216 ,  0.8874066 ,
0.87389171,  0.89674821,  0.88272625,  0.86178357,  0.88692913,
0.8746193 ,  0.89609707,  0.88605078,  0.88816007,  0.86097535,
0.86496679,  0.87868847,  0.88870234,  0.89896376,  0.89043653,
0.87331542,  0.87913618,  0.89827004,  0.87227934,  0.89565149,
0.87525145,  0.87062302,  0.87239229,  0.85600442,  0.88550452,
0.87145848,  0.89211502,  0.90353299,  0.88038848,  0.88622139,
0.88060793,  0.89626723,  0.89685886,  0.86967513,  0.89617828,
0.88664145,  0.89214936,  0.8858469 ,  0.87129332,  0.89092253,
0.86044519,  0.88449875,  0.88372771,  0.88387527,  0.87268479,
0.87833091,  0.89789563,  0.86338514,  0.8928176 ,  0.89621637,
0.88930738,  0.89897238,  0.88070246,  0.88985677,  0.89508511,
0.89030611,  0.88706102,  0.88661394,  0.89569634,  0.88976855,
0.89985859,  0.88699833,  0.88419724,  0.87730109,  0.8802179 ,
0.86071848,  0.8907923 ,  0.88950499,  0.87044137,  0.89308041,
0.88387997,  0.86910723,  0.88558055,  0.86974757,  0.88046642,
0.87158917,  0.86907395,  0.86966534,  0.87179956,  0.89448872,
0.86827365,  0.87370754,  0.86535403,  0.87897945,  0.8580772 ,
```

```
0.88223723,  0.88242499,  0.88387621,  0.8818806 ,  0.8884393 ,
0.87926438,  0.88824343,  0.88364269,  0.89651344,  0.88596669,
0.8840118 ,  0.88127919,  0.88187057,  0.87270304,  0.90351452,
0.88415491,  0.89050218,  0.89030268,  0.87403116,  0.89428613,
0.87712373,  0.8810284 ,  0.87258531,  0.87457821,  0.89554921,
0.88614288,  0.88381755,  0.87805034,  0.88829936,  0.88861411,
0.88299894,  0.8894302 ,  0.87489143,  0.90584966,  0.87947389,
0.89163007,  0.88975453,  0.88755939,  0.8813833 ,  0.88566862,
0.87273851,  0.89207715,  0.89089525,  0.88052825,  0.8796019 ,
0.87906245,  0.89412904,  0.85803389,  0.8931633 ,  0.89295912,
0.88376351,  0.8708306 ,  0.89688498,  0.88136853,  0.88200163,
0.87802689,  0.89080264,  0.87540915,  0.88359375,  0.84254117,
0.88799398,  0.91449702,  0.90848537,  0.86881791,  0.88153094,
0.87708768,  0.87462782,  0.87627057,  0.89156815,  0.88696801,
0.87978771,  0.87002141,  0.88491754,  0.87219016,  0.89647978,
0.89056693,  0.88422943,  0.8713721 ,  0.88235747,  0.88277027,
0.88854963,  0.88309916,  0.87635689,  0.87511613,  0.85849792,
0.88083333,  0.88159383,  0.8792472 ,  0.87491143,  0.87604765,
0.85284813,  0.89572857,  0.88738044,  0.87425733,  0.89579944,
0.86620621,  0.87177899,  0.89266351,  0.87160812,  0.87521956,
0.87987531,  0.90363285,  0.89908021,  0.8997783 ,  0.87231731,
0.89938727,  0.8993446 ,  0.8877759 ,  0.8931118 ,  0.89269036,
0.89713606,  0.88427952,  0.86697663,  0.87224142,  0.90422342,
0.88228967,  0.90161454,  0.88568644,  0.88644399,  0.89388369,
0.89521834,  0.83981484,  0.87410934,  0.89707074,  0.87471277,
0.8953357 ,  0.86593054,  0.88984636,  0.88006714,  0.87498499,
0.88506588,  0.9021746 ,  0.88288792,  0.85732405,  0.8988819 ,
0.89419397,  0.87933451,  0.88236605,  0.90427593,  0.8844895 ,
0.89409033,  0.88940611,  0.87930693,  0.86509281,  0.89541295,
0.89116471,  0.89427369,  0.89505632,  0.86474297,  0.86756572,
0.90172706,  0.87985176,  0.90893768,  0.90415545,  0.88282328,
0.88941824,  0.87656499,  0.89366753,  0.88504171,  0.87384821])
```



[121]:
```
grader.check("q2_6")
```

[121]:
```
q2_6 results: All test cases passed!
```

Question 2.7: Calculate a 95% confidence interval for the resampled correlations and assign either `True` or `False` to `reject` if we can reject the null hypothesis or if we cannot reject the null hypothesis using a 5% p-value cutoff.

[122]:
```
lower_bound = percentile(2.5, table.column("Uniformity of Cell Size vs␣
 ↪Uniformity of Cell Shape"))
upper_bound = percentile(97.5, table.column("Uniformity of Cell Size vs␣
 ↪Uniformity of Cell Shape"))
reject = False


# Don't change this!
print(f"95% CI: [{lower_bound}, {upper_bound}] , Reject the null: {reject}")
```

```
95% CI: [0.8569106917340795, 0.9064265834778792] , Reject the null: False
```

3.1 Analyzing Residuals

Next, we want to make a prediction for one variable (call this your y variable, or var2) based on the the other (call this your x variable, or var1). First, let's investigate how effective our predictions are.

Question 2.8:

Calculate the slope and intercept for the line of best fit for the 2 variables. Assign these values to `my_slope`, and `my_intercept`respectively. The function `parameters` returns a two-item array containing the slope and intercept of a linear regression line.

Hint 1: Use the ***`parameters`*** *function with the arguments specified!*

Hint 2: Remember we're predicting the 2nd variable ***based off*** *a first variable. That should tell you what the* ***`colx`*** *and* ***`coly`*** *arguments you should specify when calling* ***`parameters`****.*

[123]:
```
# DON'T EDIT THE PARAMETERS FUNCTION
def parameters(tbl, colx, coly):
    x = tbl.column(colx)
    y = tbl.column(coly)

    r = correlation(tbl, colx, coly)

    x_mean = np.mean(x)
    y_mean = np.mean(y)
    x_sd = np.std(x)
    y_sd = np.std(y)

    slope = (y_sd / x_sd) * r
    intercept = y_mean - (slope * x_mean)
    return make_array(slope, intercept)

my_slope = parameters(my_data, "Uniformity of Cell Size", "Uniformity of Cell␣
 ↪Shape").item(0)
my_intercept = parameters(my_data, "Uniformity of Cell Size", "Uniformity of␣
 ↪Cell Shape").item(1)
```

Question 2.9:

Draw a scatter plot of the residuals with the line of best fit for the 2 variables.

Hint: We want to get the predictions for every data point in the dataset

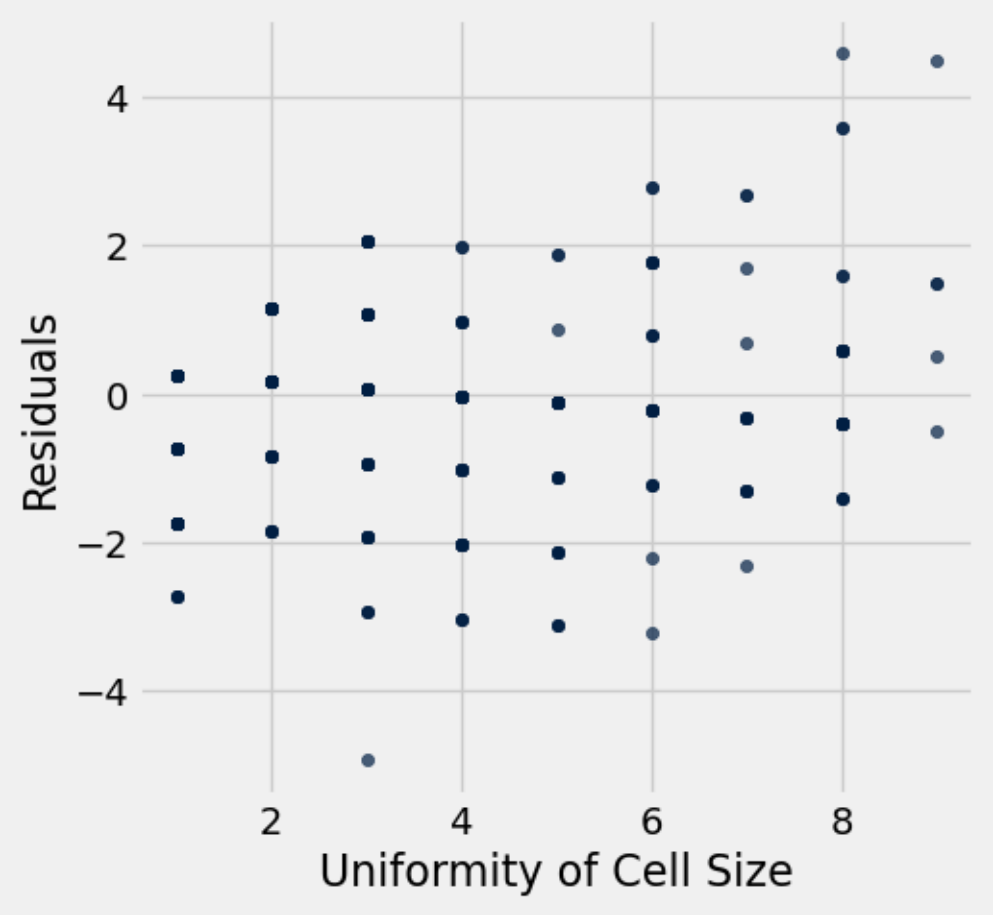
Hint 2: This question is really involved, try to follow the skeleton code!

[124]:
```
predicted_var2 = my_slope * my_data.column("Uniformity of Cell Size") +␣
 ↪my_intercept
residuals_var2 = predicted_var2 - my_data.column("Uniformity of Cell Shape")


originalTable_with_residuals = my_data.with_columns("Residuals", residuals_var2)


# Now generate a scatter plot of the residuals!
# Uncomment the line of code below and change "Column 1" to match variable 1␣
 ↪used in your linear regression analysis
```

```
originalTable_with_residuals.scatter("Uniformity of Cell Size", "Residuals")
```



Here's a link to properties of residuals in the textbook that could help out with some questions.

Question 2.10 :

Based on the plot of residuals, do you think linear regression is a good model in this case? Explain.

Yes this is a good linear regression model because there is little to no visible correlation between the residual points.

Question 2.11 Is the correlation between the residuals and your predictor positive, zero, or negative? Assign `residual_corr` to either 1, 2 or 3 corresponding to whether the correlation between the residuals and your predictor is positive, zero, or negative. Hint: it is ok to check this with Python before answering!

1. Positive
2. Zero
3. Negative

[125]:
```
residual_corr = 2
```

[126]:
```
grader.check("q2_11")
```

[126]:
```
q2_11 results: All test cases passed!
```

3.2 Prediction Intervals

Now, Maria wants to predict the 2nd variable based on a chosen first variable x.

Question 2.12: First, let's identify a value of your choice for x that you want to predict y with and explain in your own words why you chose that value.

I chose 6 because it's in the range.

Question 2.13:

Define the function `one_resample_prediction` that generates a bootstrapped sample from the `tbl` argument, calculates the line of best fit for `ycol` vs `xcol` for that resample, and predicts a value based on `xvalue`. Then assign the value you chose for x in Question 2.12 to `chosen_var1`.

Hint: Remember you defined the ***`parameters`*** *function earlier*

[127]:
```
def one_resample_prediction(tbl, colx, coly, xvalue):
    resample = tbl.sample()
    best_fit = parameters(resample, colx, coly)
    bfline = best_fit.item(0) * xvalue + best_fit.item(1)
    return bfline

chosen_var1 = 6

maria_prediction = one_resample_prediction(my_data, "Uniformity of Cell Size",␣
 ↪"Uniformity of Cell Shape", 6)
maria_prediction
```

[127]:
```
5.687342410489158
```

[128]:
```
grader.check("q2_13")
```

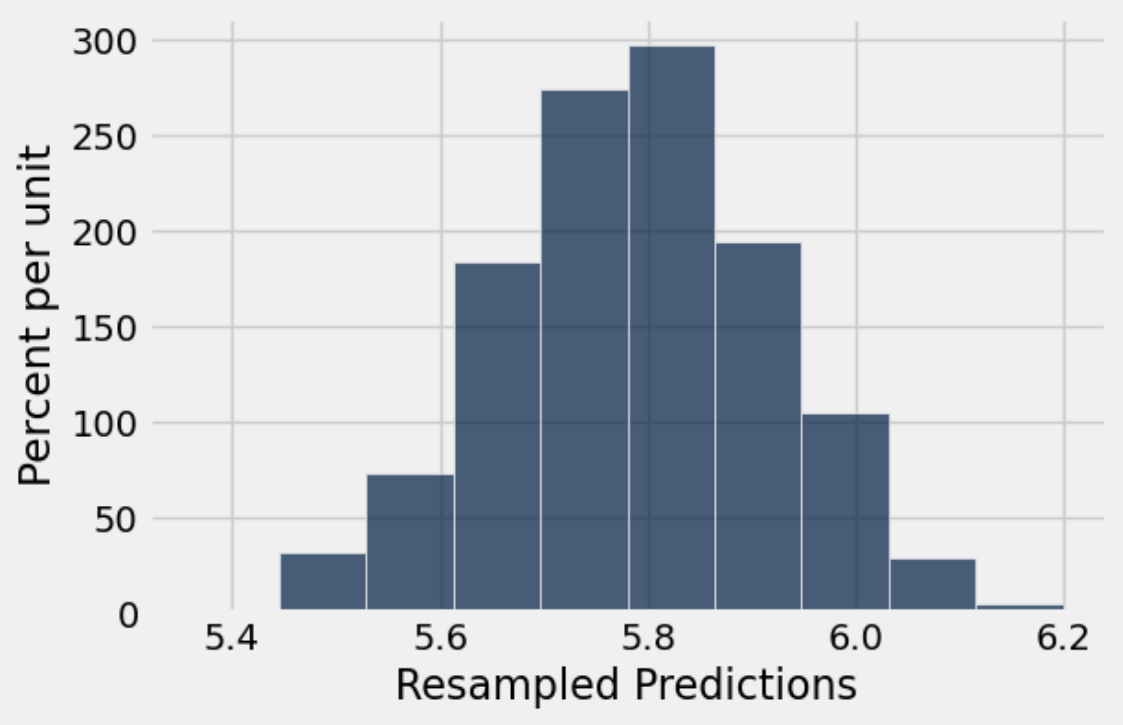
[128]:
```
q2_13 results: All test cases passed!
```

Question 2.14:

Assign `resampled_predictions` to be an array that will contain 1000 resampled predictions for the 2nd variable based on `chose_var1` that you picked, and then generate a histogram of it.

[129]:
```
resampled_predictions = make_array()
for i in np.arange(1000):
    one = one_resample_prediction(my_data, "Uniformity of Cell Size",␣
 ↪"Uniformity of Cell Shape", 6)
    resampled_predictions = np.append(resampled_predictions, one)
```

```
# Don't change/delete the code below in this cell, just run to visualize the␣
↪distribution
Table().with_column("Resampled Predictions", resampled_predictions).hist()
```



Question 2.15:

Using `resampled_predictions` from Question 2.14, generate a 99% confidence interval for Maria's prediction.

[130]:
```
lower_bound_maria = percentile(.5, resampled_predictions)
upper_bound_maria = percentile(99.5, resampled_predictions)


# Don't delete/modify the code below in this cell
print(f"99% CI: [{lower_bound_maria}, {upper_bound_maria}]")
```

```
99% CI: [5.469285118827983, 6.1001410530506295]
```

[131]:
```
grader.check("q2_15")
```

[131]:
```
q2_15 results: All test cases passed!
```

Question 2.16: Uncomment and change the 2 lines of code underneath the TODOs, with the correct `Column 1` and `Column 2`. Then run the following cell to see a few bootstrapped regression lines, and the predictions they make for your chosen value for `chosen_var1` (picked in question

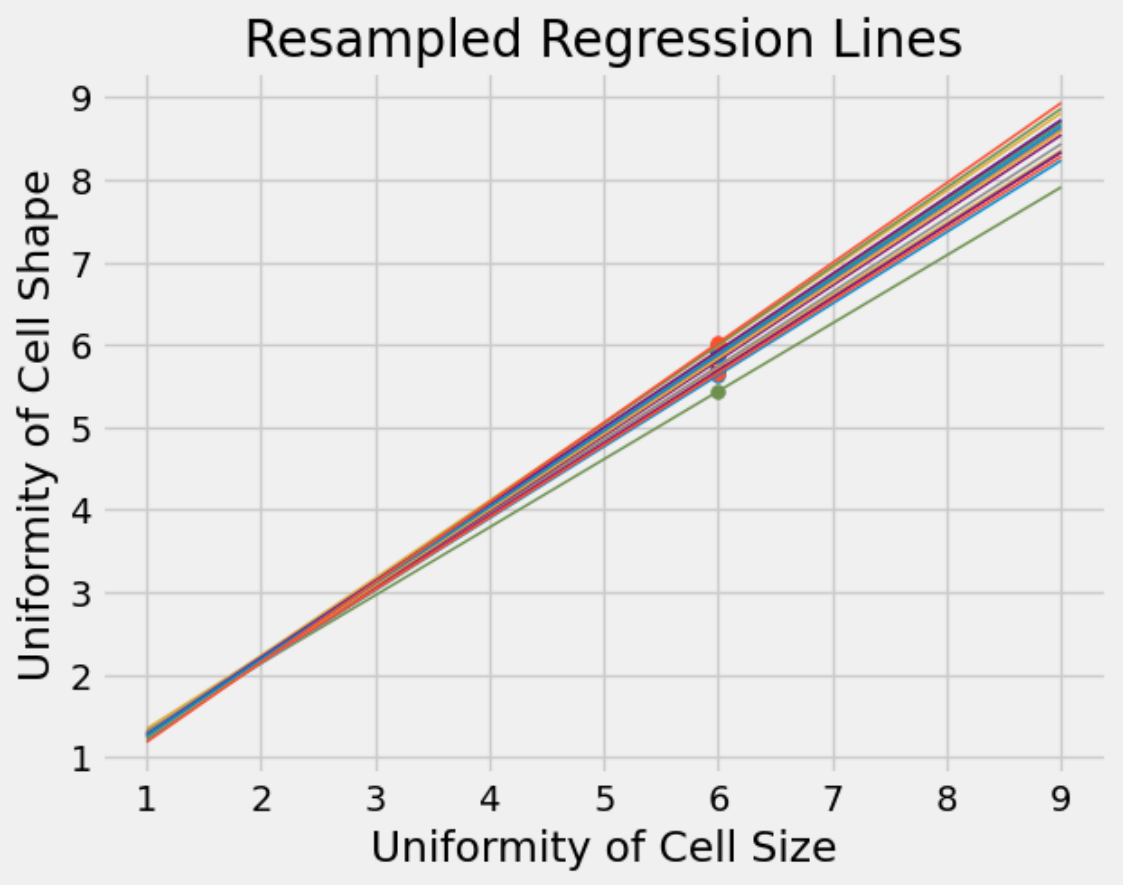
2.13)

[132]:
```
# You don't need to understand all of what it is doing but you should recognize␣
 ↪a lot of the code!
lines = Table(['slope','intercept'])

x=chosen_var1 # This is the value you picked in question 2.14

for i in np.arange(20):
    resamp = originalTable_with_residuals.sample(with_replacement=True)
    # TODO: change Column 1 and Column 2 in the line below and uncomment
    resample_pars = parameters(resamp, "Uniformity of Cell Size", "Uniformity␣
 ↪of Cell Shape")
    slope = resample_pars.item(0)
    intercept = resample_pars.item(1)
    lines.append([slope, intercept])

lines['prediction at x='+str(x)] = lines.column('slope')*x + lines.
 ↪column('intercept')
# TODO: change Column 1 in the line below and uncomment
xlims = [min(originalTable_with_residuals.column("Uniformity of Cell Size")),␣
 ↪max(originalTable_with_residuals.column("Uniformity of Cell Size"))]
left = xlims[0]*lines[0] + lines[1]
right = xlims[1]*lines[0] + lines[1]
fit_x = x*lines['slope'] + lines['intercept']
for i in range(20):
    plt.plot(xlims, np.array([left[i], right[i]]), lw=1)
    plt.scatter(x, fit_x[i], s=30)
plt.ylabel("Uniformity of Cell Shape"); # You can change the label here to be␣
 ↪more descriptive
plt.xlabel("Uniformity of Cell Size"); # You can change the label here to be␣
 ↪more descriptive
plt.title("Resampled Regression Lines");
```



Question 2.17

What are some biases in this dataset that may have affected our analysis? Some questions you can ask yourself are: "is our sample a simple random sample?" or "what kind of data are we using/what variables are we dealing with: are they categorical, numerical, or both (both is something like ordinal data)?".

Hint: you might want to revisit the beginning of this assignment to reread where your data came from and how the table was generated.

Possible biases for my data is how the data was collected, where in the world the data was collected, and possible gender biases when collecting the data.

4 3. Classification

Recommended Reading:

- Classification

This part of the project is about k-Nearest Neighbors classification (kNN), and the purpose is to reinforce the basics of this method. You will be using the same dataset you picked in section one to complete this part.

We will try to classify our data in 2 classes/groups (labels) based on other variables (features) in our dataset. Go back to question 1.4 and review your answer and your visualization. If it helps copy the code for the visualization below.

4.1 3.1 Splitting the Dataset

Question 3.1. Let's begin implementing the k-Nearest Neighbors algorithm. Define the `distance` function, which takes in two arguments: an array of numerical features (`arr1`), and a different array of numerical features (`arr2`). The function should return the Euclidean distance between the two arrays. Euclidean distance is often referred to as the straight-line distance formula that you may have learned previously.

[133]:
```
def distance(arr1, arr2):
    return np.sqrt(np.sum((arr1 - arr2) ** 2))

# Don't change/delete the code below in this cell
distance_example = distance(make_array(1, 2, 3), make_array(4, 5, 6))
distance_example
```

[133]:
```
5.196152422706632
```

[134]:
```
grader.check("q3_1")
```

[134]:
```
q3_1 results: All test cases passed!
```

4.1.1 Splitting the Dataset

We'll do two different kinds of things with the dataset:

1. We'll build a classifier using the data for which we know the associated label; this will teach it to recognize labels of similar coordinate values. This process is known as *training*.
2. We'll evaluate or *test* the accuracy of the classifier we build on data we haven't seen before.

As discussed in Section 17.2, we want to use separate datasets for training and testing. As such, we split up our one dataset into two.

Question 3.2. Next, let's split our dataset into a training set and a test set. We will start with the full dataset `my_data_raw` (not the one with just the columns used for regression). The table should contain the variable that will be used in classification, it should look like the table after question 1.3.

Now, let's create a training set with the first 75% of the dataset and a test set with the remaining 25% (e.g. if your dataset has 100 rows, 75 rows will be the training set, 25 rows will be the test set). Remember that assignment to each group should be random, so we should shuffle the table first.

Hint: as a first step we can ***shuffle*** *all the rows, then use the* `tbl.take` *function to split up the rows for each table*

[135]:
```
round(75* my_data_raw.num_rows/100,0)
```

[135]:
```
512.0
```

[136]:
```
shuffled_table = my_data_raw.sample(with_replacement = False)
train = shuffled_table.take(np.arange(512))
test = shuffled_table.take(np.arange(512, my_data_raw.num_rows))

print("Training set:\t",   train.num_rows, "examples")
print("Test set:\t",       test.num_rows, "examples")
train.show(5), test.show(5);
```

```
Training set:    512 examples
Test set:        171 examples
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[137]:
```
grader.check("q3_2")
```

[137]:
```
q3_2 results: All test cases passed!
```

4.2 3.2 K-Nearest Neighbors

K-Nearest Neighbors (k-NN) is a classification algorithm. Given some numerical *attributes* (also called *features*) of an unseen example, it decides whether that example belongs to one or the other of two categories based on its similarity to previously seen examples. Predicting the category of an example is called *labeling*, and the predicted category is also called a *label*.

Question 3.3. Assign `chosen_features` to an array of column names (strings) of the features (column labels) from the dataset.

Hint: Which of the column names in the table are the features, and which of the column names correspond to the class we're trying to predict?

Hint: No need to modify any tables, just manually create an array of the feature names!

[138]:
```
chosen_features = make_array('Clump Thickness', "Uniformity of Cell Size",␣
↪"Uniformity of Cell Shape", "Marginal Adhesion", "Single Epithelial Cell␣
↪Size", "Bare Nuclei")
chosen_features
```

[138]:
```
array(['Clump Thickness', 'Uniformity of Cell Size',
       'Uniformity of Cell Shape', 'Marginal Adhesion',
       'Single Epithelial Cell Size', 'Bare Nuclei'],
      dtype='<U27')
```

Question 3.4. Now define the `classify` function. This function should take in a `test_row` from a table like `test` and classify in using the k-Nearest Neighbors based on the correct `features` and the data in `train`. A refresher on k-Nearest Neighbors can be found here.

Hint 1: The `distance` function we defined earlier takes in arrays as input, so use the `row_to_array` function we defined for you to convert rows to arrays of features.

Hint 2: The skeleton code we provided iterates through each row in the training set.

[139]:
```
def row_to_array(row, features):
    """Converts a row to an array of its features."""
    arr = make_array()
    for feature in features:
        arr = np.append(arr, row.item(feature))
    return arr

def classify(test_row, k, train, features):
    test_row_features_array = row_to_array(test_row, features)
    distances = make_array()
    for train_row in train.rows:
        train_row_features_array = row_to_array(train_row, features)
        row_distance = distance(train_row_features_array,␣
 ↪test_row_features_array)
        distances = np.append(distances, row_distance)
    train_with_distances = train.with_column("Distances", distances)
    nearest_neighbors = train_with_distances.sort("Distances").take(np.
 ↪arange(k))
    most_common_label = nearest_neighbors.group("Class")
    return most_common_label.sort("count", descending = True).column(0).item(0)

# Don't modify/delete the code below
first_test = classify(test.row(0), 5, train, chosen_features)
first_test
```

[139]: 0

4.2.1 Evaluating your classifier

Now that we have a way to use this classifier, let's focus on the 3 Nearest Neighbors and see how accurate it is on the whole test set.

Question 3.5. Define the function `three_classify` that takes a `row` from `test` as an argument and classifies the row based on using 3-Nearest Neighbors. Use this function to find the `accuracy` of a 3-NN classifier on the `test` set. `accuracy` should be a proportion (not a percentage) of the test data that were correctly predicted.

Hint: You should be using a function you just created!

Note: Usually before using a classifier on a test set, we'd classify first on a "validation" set, which we then can modify our training set again if need be, before actually testing on the test set. You don't need to do that for this question, but please keep this in mind for future courses.

[140]:
```
def three_classify(row):
    return classify(row, 3, train, chosen_features)

predictions =  test.apply(three_classify)
test_with_prediction = test.with_columns("Predictions", predictions)
labels_correct = test.column("Class")
accuracy = np.count_nonzero(predictions == labels_correct) / test.num_rows
accuracy
```

[140]:
```
0.9707602339181286
```

Question 3.6. An important part of evaluating your classifiers is figuring out where they make mistakes. Assign the name `test_correctness` to the test_with_prediction table with an additional column `'Was correct'`. The last column should contain `True` or `False` depending on whether or not our classifier classified correctly. *Note:* You can either include all of the columns from the test_with_prediction table or just the columns representing the features used by the classifier.

[141]:
```
# Feel free to use multiple lines of code
# but make sure to assign test_correctness to the proper table!
test_correctness = test_with_prediction.with_column("Was correct", (predictions
 ↪== labels_correct))
test_correctness.sort('Was correct')
```

[141]:
```
Clump Thickness | Uniformity of Cell Size | Uniformity of Cell Shape | Marginal
Adhesion | Single Epithelial Cell Size | Bare Nuclei | Bland Chromatin | Normal
Nucleoli | Mitoses | Class | Predictions | Was correct
5               | 2                       | 2                       | 4
| 2                       | 4           | 1               | 1
| 1       | 0     | 1           | False
3               | 4                       | 4                       | 10
| 5                       | 1           | 3               | 3
| 1       | 1     | 0           | False
8               | 4                       | 4                       | 5
| 4                       | 7           | 7               | 8
| 2       | 0     | 1           | False
5               | 3                       | 4                       | 3
| 4                       | 5           | 4               | 7
| 1       | 0     | 1           | False
5               | 7                       | 4                       | 1
| 6                       | 1           | 7               | 10
| 3       | 1     | 0           | False
2               | 3                       | 1                       | 1
| 5                       | 1           | 1               | 1
| 1       | 0     | 0           | True
4               | 5                       | 5                       | 10
| 4                       | 10          | 7               | 5
| 8       | 1     | 1           | True
```

```
5               | 3                     | 5                       | 5
| 3                         | 3           | 4               | 10
| 1       | 1     | 1           | True
4               | 8                     | 6                       | 3
| 4                         | 10          | 7               | 1
| 1       | 1     | 1           | True
8               | 4                     | 10                      | 5
| 4                         | 4           | 7               | 10
| 1       | 1     | 1           | True
… (161 rows omitted)
```

Question 3.7. Do you see a pattern in the rows that your classifier misclassifies? In two sentences or less, describe any patterns you see in the results or any other interesting findings from the table above.

some patterns I noticed were that my classifier seems to misclassify rows with high number of uniformity of cell shape, and higher numbers of single epithelial cell size.

Question 3.8. Why do we divide our data into a training and test set? What is the point of a test set, and why do we only want to use the test set once? Explain your answer in 3 sentences or less.

Hint: Check out this section in the textbook.

We use a training set to test on data that we do know, so that it can be more accurate when testing data that we do not know. The test set shows whether the model generalizes instead of just memorizing the training data. We use the test set only once because repeatedly checking it can leak information and make the evaluation less accurate.

Question 3.9. Why do we use an odd-numbered `k` in k-NN? Explain.

We use an odd number for k so that the vote doesn't end in a tie. This helps the model make a clear decision when choosing a class.

At this point, you've gone through one cycle of classifier design. Let's summarize the steps: 1. From available data, select test and training sets. 2. Choose an algorithm you're going to use for classification. 3. Identify some features. 4. Define a classifier function using your features and the training set. 5. Evaluate its performance (the proportion of correct classifications) on the test set.

4.3 4. Explorations

Now that you know how to evaluate a classifier, it's time to build a better one.

Question 4.1:

Develop a classifier with better test-set accuracy than `three_classify`. Your new function should have the same arguments as `three_classify` and return a classification. Name it `another_classifier`. Then, check your accuracy using code from earlier.

You can use more or different features, or you can try different values of `k`. (Of course, you still have to use `train` as your training set!)

Make sure to create new variable names where needed, don't reassign any previously used variables here, such as `accuracy` from the section 3.

[142]:
```
# Run this cell to remember what your accuracy was in the first attempt.

accuracy
```

[142]: 0.9707602339181286

[143]:
```
# Feel free to add or change this array to improve your classifier
# Note that you can either use the original chosen_features or create a new␣
 ↪list below

new_features = make_array("Bland Chromatin", "Normal Nucleoli", "Mitoses")

def another_classifier(row):
    return classify(row, 3, train, new_features)

new_test_with_prediction = test.apply(another_classifier)
new_labels_correct = test.column("Class")
new_accuracy = np.count_nonzero(new_test_with_prediction == new_labels_correct)␣
 ↪/ test.num_rows
new_accuracy
```

[143]: 0.96491228070175439

[144]:
```
# Now that we looked at the accuracy, let's analyze correctness of your new␣
 ↪classifier
# Use this coding cell to explore your data/ this cell will not be graded
# You are free to use as many lines of code as you would like
# you could look at a sorted table again just like in 3.6!
```

Question 4.2

Did your new classifier work better? Do you see a pattern in the mistakes your new classifier makes? What about in the improvement from your first classifier to the second one? Describe in two sentences or less.

Hint: You may not be able to see a pattern.

My new classifier does not work better, this suggests that the new features I added do not contribute to testing whether or not this person has breast cancer.

Question 4.3

Briefly describe what you tried to improve your classifier. Any other ideas on how you could make a better classifier?

I tried to add the rest of the columns (besides class). I could make a better classifier if I had more data relating to things that could possibly contribute to breast cancer. Maybe even other health/lifestyle data such as diet and family history.

Question 4.4: Misclassification and errors in classifiers happens all the time and can really affect an individual. When applying machine learning and building classifiers, we all need to do our best to minimize misclassification and make sure we are transparent about the accuracies of what we built. Have you ever experienced something like this in real life before, where something was classified incorrectly? If not, can you think of an example where misclassification could really affect an individual?

I have not experienced this in real life, but in this case it could very dangerously affect an individual if something such as breast cancer was misclassified.

Question 4.5: We hope you enjoyed the project! You made it to the concluding question.

In a couple of sentences, share what you learned about your dataset while exploring its potential for prediction and classification.

I learned more anatomy, as well as many terms I hadn't heard of relating to breast cancer. This was actually a very valuable thing to learn about. Thanks!

Congratulations: You're DONE with the final project notebook! Nice work. Time to submit.