Final Project with Research Implementations

January 7, 2025

[930]:
```
# 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

[931]:
```
# 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

[932]:
```
#Source: This dataset was found in Data.ca.gov under the title "Short-Term␣
↪Occupational Employment Projections"
```

```
#Description: Short-term Occupational Projections for a 2-year time␣
 ↪(2023-2025)produced for the state of California.
#Purpose: Data helps to make informed decisions on individual career and␣
 ↪organizational program development.
# Load your dataset into a table

my_data_raw = Table.read_table('MyData1.csv').drop("SOC Level", "Standard␣
 ↪Occupational Classification (SOC)", "Area Type", "Area Name", "Period")
my_data_raw
```

[932]:
```
Occupational Title                                    | Base Quarter
Employment Estimate | Projected Quarter Employment Estimate | Numeric Change |
Percentage Change | Exits   | Transfers | Total Job Openings | Median Hourly
Wage | Median Annual Wage | Entry Level Education | Work Experience   | Job
Training
Total, All Occupations                                | 19920000
| 20412400                              | 492400         | 2.5              |
1981550 | 2469420   | 4943370           | 24.73             | 51450
| nan                   | nan               | nan
Management Occupations                                | 1669600
| 1705600                               | 36000          | 2.2              |
102730  | 153320    | 292050            | 63.9              | 132918
| nan                   | nan               | nan
Top Executives                                        | 335600
| 342000                                | 6400           | 1.9              |
17520   | 34380     | 58300             | 0                 | 0
| nan                   | nan               | nan
Chief Executives                                      | 51800
| 51500                                 | -300           | -0.6             |
3190    | 3590      | 6480              | 104.13            | 0
| Bachelor's degree     | 5 years or more   | nan
General and Operations Managers                       | 281500
| 288100                                | 6600           | 2.3              |
14180   | 30600     | 51380             | 57.11             | 118793
| Bachelor's degree     | 5 years or more   | nan
Legislators                                           | 2300
| 2300                                  | 0              | 0                |
140     | 190       | 330               | 0                 | 61446
| Bachelor's degree     | Less than 5 years | nan
Advertising, Marketing, Promotions, Public Relations, an … | 187100
| 189500                                | 2400           | 1.3              |
9010    | 18930     | 30340             | 0                 | 0
| nan                   | nan               | nan
Advertising and Promotions Managers                   | 5200
| 5200                                  | 0              | 0                |
210     | 690       | 900               | 65.4              | 136038
| Bachelor's degree     | Less than 5 years | nan
```

```
Marketing Managers                                          | 60300
| 61100                                | 800           | 1.3              |
2880    | 6580      | 10260             | 81.59             | 0
| Bachelor's degree     | 5 years or more   | nan
Sales Managers                                              | 108600
| 109900                               | 1300          | 1.2              |
5320    | 10500     | 17120             | 63.81             | 132734
| Bachelor's degree     | Less than 5 years | nan
… (789 rows omitted)
```

[933]:
```
grader.check("q1_1")
```

[933]:
```
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.

The variable "Occupational Title" is categorical, "Base Quarter Employment Estimate" is numerical and measures the employment for an occupation in 2023, "Projected Quarter Employment Estimate" is numerical and measures the predicted employment for an occupation in 2025, "Numeric Change" is numerical and measures the predicted change in employment between the two years for occupations, "Percentage Change" is numerical and measures the predicted percentage change in employment between the two years, "Exits" is numerical and measures the amount of people that have left an occupation, "Transfers" is numerical and measures the amount of people that left an occupation and transferred to a different one, "Total Job Openings" is numerical and measures the openings (positions) for workers entering an occupation, "Median Hourly Wage" is numerical, "Median Annual Wage" is numerical, "Entry Level Education" is categorical, "Work Experience" is categorical, and "Job Training" is categorical. There are 799 total observations.

[934]:
```
#Shows the number of observations
my_data_raw.num_rows
```

[934]:
```
799
```

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!)

[935]:
```
# 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
```

```
#Dependent variable is whether there was a growth in the number of employment␣
 ↪from the base to the projected. If "percentage change" is positive, there␣
 ↪was growth (marked by True in var)
#function used to determine the dependent variable of each row
def positiveOrNegative(array):
    if array >= 0:
        return True
    if array < 1:
        return False

posOrNeg = my_data_raw.apply(positiveOrNegative, "Percentage Change")
my_data_raw = my_data_raw.with_column("Employment Projection Growth?", posOrNeg)
var = "Employment Projection Growth?"
var
```

[935]: 'Employment Projection Growth?'

[936]:
```
grader.check("q1_3")
```

[936]: q1_3 results: All test cases passed!

[937]:
```
my_data_raw
```

[937]:
```
Occupational Title                                          | Base Quarter
Employment Estimate | Projected Quarter Employment Estimate | Numeric Change |
Percentage Change | Exits    | Transfers | Total Job Openings | Median Hourly
Wage | Median Annual Wage | Entry Level Education | Work Experience    | Job
Training | Employment Projection Growth?
Total, All Occupations                                        | 19920000
| 20412400                                | 492400          | 2.5              |
1981550 | 2469420    | 4943370             | 24.73               | 51450
| nan                     | nan                | nan           | True
Management Occupations                                        | 1669600
| 1705600                                 | 36000           | 2.2              |
102730   | 153320     | 292050              | 63.9                | 132918
| nan                     | nan                | nan           | True
Top Executives                                                | 335600
| 342000                                  | 6400            | 1.9              |
17520    | 34380      | 58300               | 0                   | 0
| nan                     | nan                | nan           | True
Chief Executives                                              | 51800
| 51500                                   | -300            | -0.6             |
3190     | 3590       | 6480                | 104.13              | 0
| Bachelor's degree       | 5 years or more    | nan           | False
General and Operations Managers                               | 281500
| 288100                                  | 6600            | 2.3              |
14180    | 30600      | 51380               | 57.11               | 118793
```

```
| Bachelor's degree     | 5 years or more   | nan          | True
Legislators                                          | 2300
| 2300                                  | 0             | 0                 |
140     | 190       | 330               | 0                 | 61446
| Bachelor's degree     | Less than 5 years | nan          | True
Advertising, Marketing, Promotions, Public Relations, an … | 187100
| 189500                                | 2400          | 1.3               |
9010    | 18930     | 30340             | 0                 | 0
| nan                   | nan               | nan          | True
Advertising and Promotions Managers                  | 5200
| 5200                                  | 0             | 0                 |
210     | 690       | 900               | 65.4              | 136038
| Bachelor's degree     | Less than 5 years | nan          | True
Marketing Managers                                   | 60300
| 61100                                 | 800           | 1.3               |
2880    | 6580      | 10260             | 81.59             | 0
| Bachelor's degree     | 5 years or more   | nan          | True
Sales Managers                                       | 108600
| 109900                                | 1300          | 1.2               |
5320    | 10500     | 17120             | 63.81             | 132734
| Bachelor's degree     | Less than 5 years | nan          | True
… (789 rows omitted)
```

[938]:
```
my_data_raw = my_data_raw.take(np.arange(1, 799, 1))
my_data_raw.show()
```

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

[939]:
```
#Some occupations had missing median hourly wage or annual wage. This cell␣
 ↪finds the missing wage using the other wage if available.
#For example: if an occupation has hourly wage, but no annual wage, we can␣
 ↪multiply the hourly wage by 2080 to find the yearly wage because the average␣
 ↪hours worked per year is 2080.
#If both annual and hourly wage are missing for an occupation, it is removed␣
 ↪from the data
def hourlyToAnnual (hourly, annual):
    if (hourly > 0):
        return (hourly * 2080)
    if (hourly == 0):
        return annual
def annualToHourly (annual):
        return (annual/2080)
annualBasedOnHour = my_data_raw.apply(hourlyToAnnual, "Median Hourly Wage",␣
 ↪"Median Annual Wage")
my_data_raw = my_data_raw.with_column("New Annual", annualBasedOnHour)
hourlyBasedOnAnnual = my_data_raw.apply(annualToHourly, "New Annual")
my_data_raw = my_data_raw.with_column("New Hourly", hourlyBasedOnAnnual)
```

```
my_data_raw.drop("Median Hourly Wage", "Median Annual Wage")
my_data_raw.relabel("New Annual", "Median Annual Wage").relabel("New Hourly",␣
 ↪"Median Hourly Wage")
my_data_raw = my_data_raw.where("Median Hourly Wage", are.above(0))
```

[940]:
```
#This cell is used to add additional useful columns to my data ("Employment %␣
 ↪Decrease", "Employment % Increase", "Exit %", "Transfer%")
def propOfExits (employees, exits):
    proportion = (exits/employees) * 100
    return proportion
def propOfTransfers (employees, transfers):
    proportion = (transfers/employees) * 100
    return proportion
def propOfOpenings (employees, openings):
    proportion = (openings/employees) * 100
    return proportion
def propOfDecrease (exits, transfers):
    return (exits + transfers)

exitProportions = my_data_raw.apply(propOfExits, "Base Quarter Employment␣
 ↪Estimate", "Exits")
my_data_raw = my_data_raw.with_column("Exit %", exitProportions)

transferProportions = my_data_raw.apply(propOfTransfers, "Base Quarter␣
 ↪Employment Estimate", "Transfers")
my_data_raw = my_data_raw.with_column("Transfer %", transferProportions)

openingProportion = my_data_raw.apply(propOfOpenings, "Base Quarter Employment␣
 ↪Estimate", "Total Job Openings")
my_data_raw = my_data_raw.with_column("Employment % Increase",␣
 ↪openingProportion)

employmentDecrease = my_data_raw.apply(propOfDecrease, "Exit %", "Transfer %")
my_data_raw = my_data_raw.with_column("Employment % Decrease",␣
 ↪employmentDecrease)
my_data_raw
```

[940]:
```
Occupational Title                  | Base Quarter Employment Estimate |
Projected Quarter Employment Estimate | Numeric Change | Percentage Change |
Exits  | Transfers | Total Job Openings | Median Hourly Wage | Median Annual
Wage | Entry Level Education | Work Experience   | Job Training | Employment
Projection Growth? | Exit %  | Transfer % | Employment % Increase | Employment %
Decrease
Management Occupations              | 1669600                           | 1705600
| 36000          | 2.2              | 102730 | 153320    | 292050            |
63.9              | 132912             | nan                   | nan
```

| nan | True | 6.15297 | 9.18304 | 17.4922 | 15.336
Chief Executives | 51800 | 51500 | -300 | -0.6 | 3190 | 3590 | 6480 | 104.13 | 216590 | Bachelor's degree | 5 years or more | nan | False | 6.1583 | 6.9305 | 12.5097 | 13.0888
General and Operations Managers | 281500 | 288100 | 6600 | 2.3 | 14180 | 30600 | 51380 | 57.11 | 118789 | Bachelor's degree | 5 years or more | nan | True | 5.0373 | 10.8703 | 18.2522 | 15.9076
Legislators | 2300 | 2300 | 0 | 0 | 140 | 190 | 330 | 29.5413 | 61446 | Bachelor's degree | Less than 5 years | nan | True | 6.08696 | 8.26087 | 14.3478 | 14.3478
Advertising and Promotions Managers | 5200 | 5200 | 0 | 0 | 210 | 690 | 900 | 65.4 | 136032 | Bachelor's degree | Less than 5 years | nan | True | 4.03846 | 13.2692 | 17.3077 | 17.3077
Marketing Managers | 60300 | 61100 | 800 | 1.3 | 2880 | 6580 | 10260 | 81.59 | 169707 | Bachelor's degree | 5 years or more | nan | True | 4.77612 | 10.9121 | 17.0149 | 15.6882
Sales Managers | 108600 | 109900 | 1300 | 1.2 | 5320 | 10500 | 17120 | 63.81 | 132725 | Bachelor's degree | Less than 5 years | nan | True | 4.89871 | 9.66851 | 15.7643 | 14.5672
Public Relations Managers | 9200 | 9500 | 300 | 3.3 | 420 | 820 | 1540 | 74.82 | 155626 | Bachelor's degree | 5 years or more | nan | True | 4.56522 | 8.91304 | 16.7391 | 13.4783
Fundraising Managers | 3800 | 3900 | 100 | 2.6 | 170 | 340 | 610 | 58.84 | 122387 | Bachelor's degree | 5 years or more | nan | True | 4.47368 | 8.94737 | 16.0526 | 13.4211
Administrative Services Managers | 37400 | 38400 | 1000 | 2.7 | 2660 | 3130 | 6790 | 51.81 | 107765 | Bachelor's degree | Less than 5 years | nan | True | 7.1123 | 8.36898 | 18.1551 | 15.4813

… (679 rows omitted)

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.

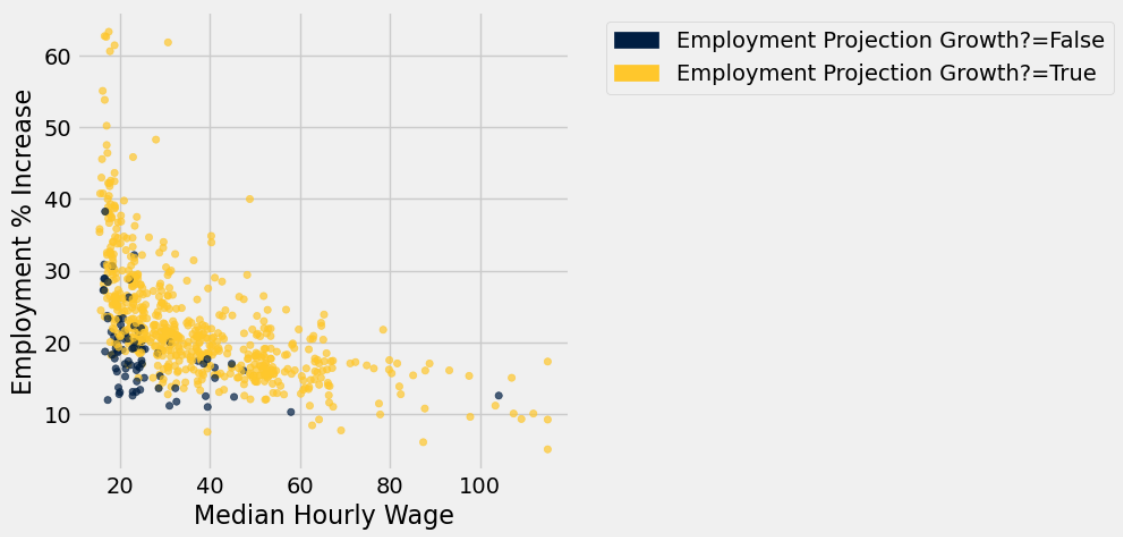
Use the following line of code:

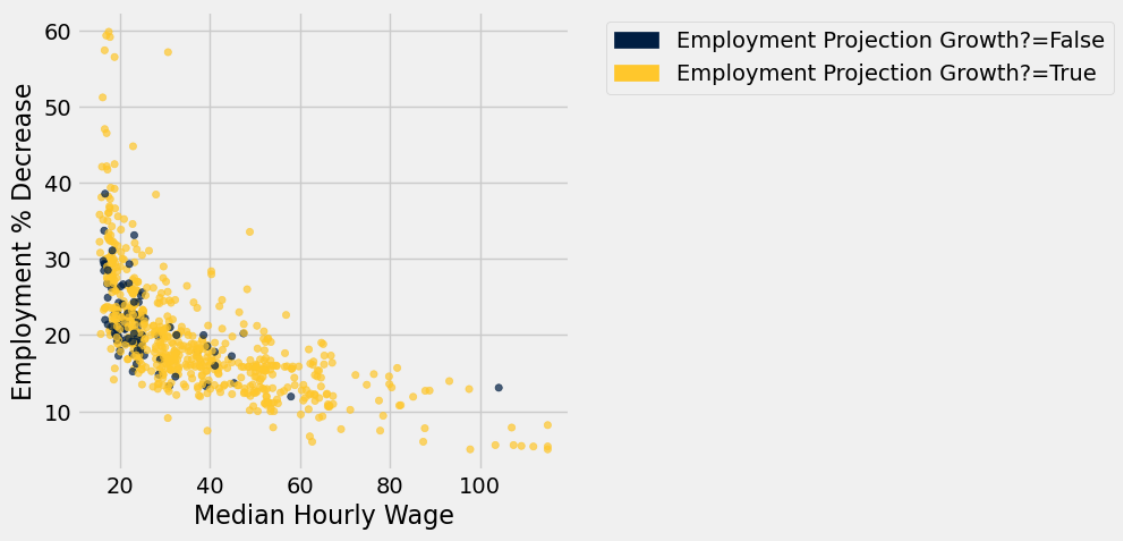
my_data.scatter(`Column 1`, `Column 2`, group=`label`)

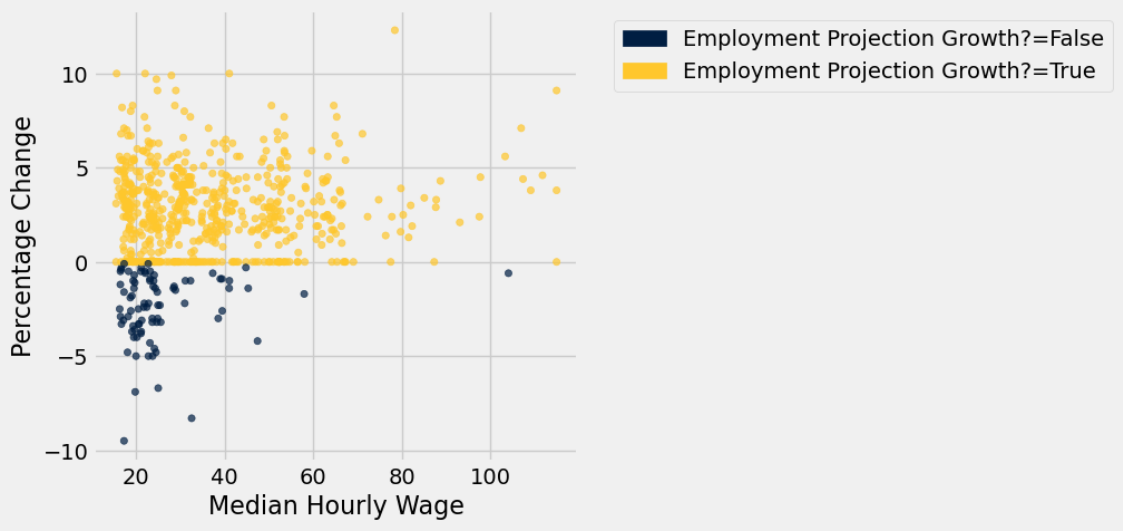
Replace `Column 1` and `Column 2` with the correct column names of numerical variables(features) you would like to investigate. Replace `label` with the column name of the categorical variable you would like to try to classsify.

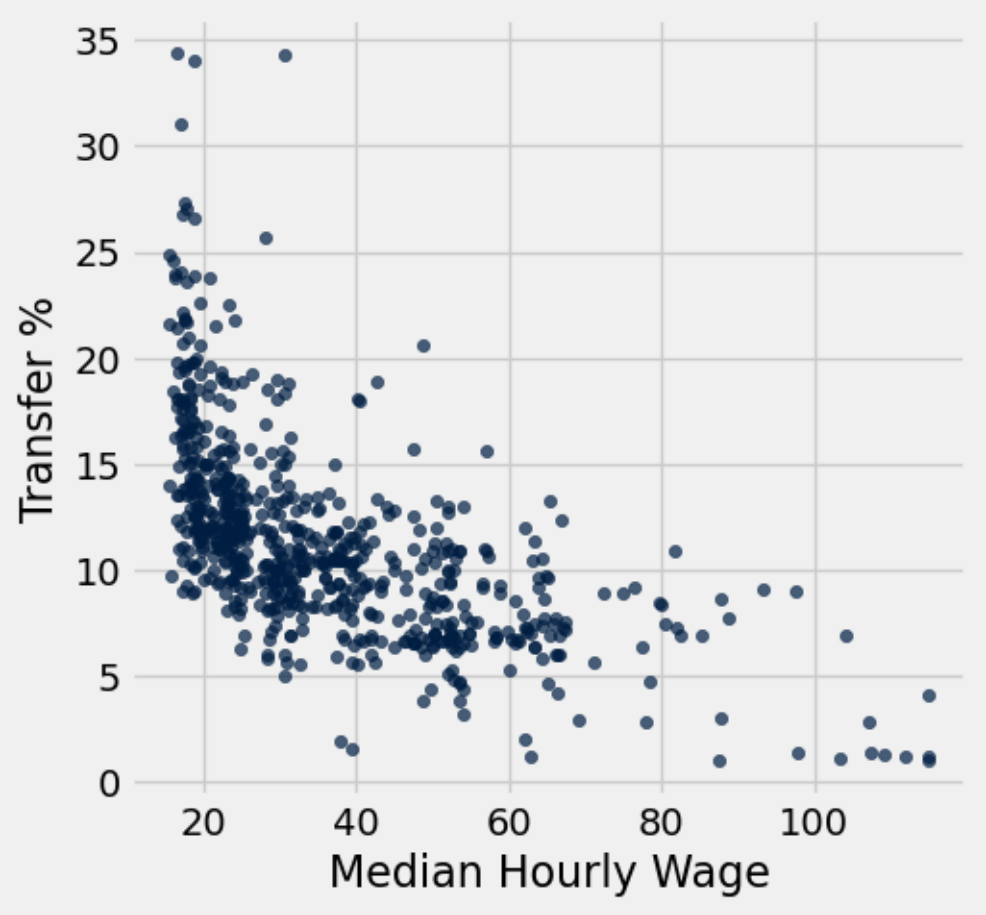
Note: The commented code in the cell below is sample code for this.

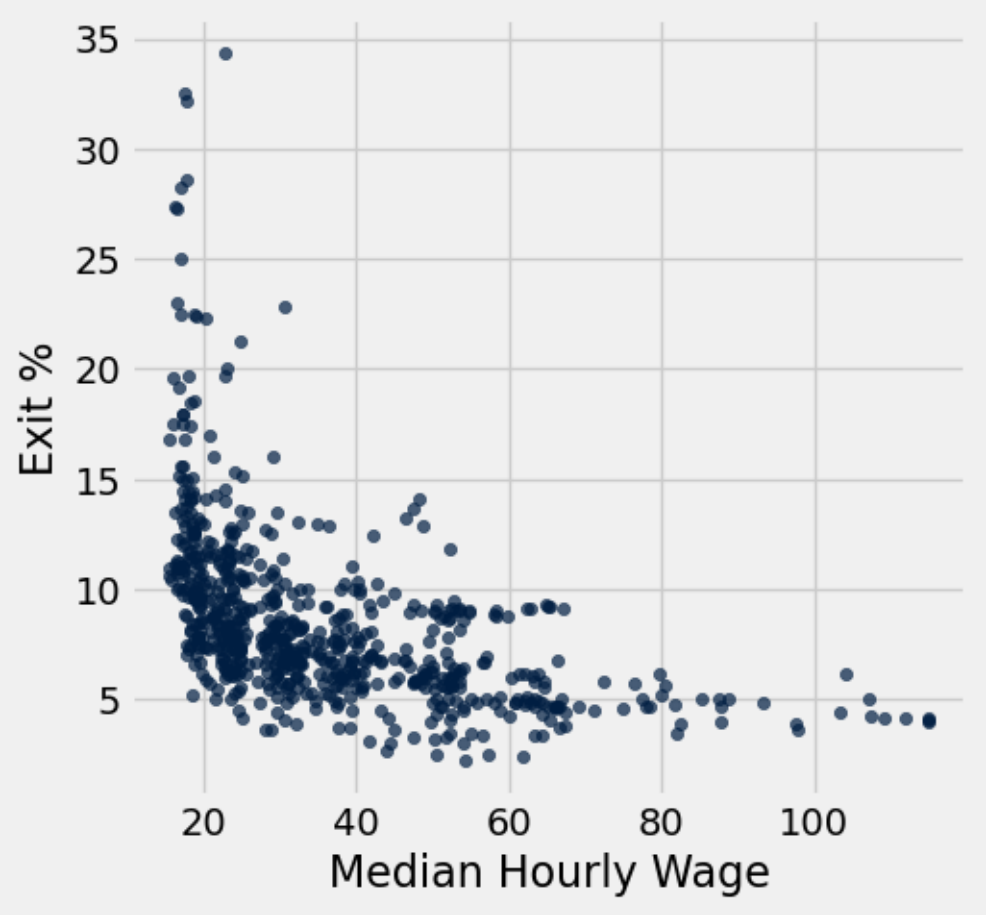
[941]:
```
# Here is the code placeholder to use, uncomment the line and adjust according␣
 ↪to the names if the columns in your dataset:
# my_data.scatter("Column 1", "Column 2", group="label")

my_data_raw.scatter("Median Hourly Wage", "Employment % Increase", group =␣
 ↪"Employment Projection Growth?")
my_data_raw.scatter("Median Hourly Wage", "Employment % Decrease", group =␣
 ↪"Employment Projection Growth?")
my_data_raw.scatter("Median Hourly Wage", "Percentage Change", group =␣
 ↪"Employment Projection Growth?")
my_data_raw.scatter("Median Hourly Wage", "Transfer %")
my_data_raw.scatter("Median Hourly Wage", "Exit %")
```




Employment Projection Growth?=False
Employment Projection Growth?=True
Employment % Decrease
60
50
40
30
20
10
20
40
60
80
100
Median Hourly Wage


Employment Projection Growth?=False
Employment Projection Growth?=True
Percentage Change
10
5
0
−5
−10
20
40
60
80
100
Median Hourly Wage


Transfer %
35
30
25
20
15
10
5
0
20
40
60
80
100
Median Hourly Wage



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 first plot displays the relationship between Median Hourly Wage and the Percentage Increase of Employment (percentage of Base Quarter Employment Estimate that are total job openings). This plot shows a negative linear association. The second plot displays the relationship between the Median Hourly Wage and the Percentage Decrease of Employment (percentage of Base Quarter Employment Estimate that are transfers + the percentage that are exits). The third plot displays the relatinship between Median Hourly Wage and the Percentage Change in employment (Employment Percentage Increase - Employment Percentage Decrease). The variables in this plot does not show a linear relationship, but there seems to be a slight positive association. I will look at the plot that displays the relationship between the Median Hourly Wage and the Percentage Decrease of Employment for linear regression.

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.

Are jobs with higher wage more stable than jobs with lower wage when it comes to the eployee's employment? In other words, do people working in higher wage jobs appear to maintain that job

more than people with lower wage jobs can with their job? Can we predict whether there will be an employment projection shrinkage or growth for an occupation based on the different features of this dataset?

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

[942]:
```
instructor_signed_off = "YES"
```

[943]:
```
grader.check("q1_7")
```

[943]:
```
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.

[944]:
```
my_data_raw
```

[944]:
```
Occupational Title                  | Base Quarter Employment Estimate |
Projected Quarter Employment Estimate | Numeric Change | Percentage Change |
Exits  | Transfers | Total Job Openings | Median Hourly Wage | Median Annual
Wage | Entry Level Education | Work Experience   | Job Training | Employment
Projection Growth? | Exit %  | Transfer % | Employment % Increase | Employment %
Decrease
Management Occupations              | 1669600                           | 1705600
| 36000          | 2.2               | 102730 | 153320    | 292050            |
63.9               | 132912            | nan                   | nan
| nan          | True                          | 6.15297 | 9.18304    | 17.4922
| 15.336
Chief Executives                    | 51800                             | 51500
| -300           | -0.6              | 3190   | 3590      | 6480              |
104.13             | 216590            | Bachelor's degree     | 5 years or
more   | nan          | False                         | 6.1583  | 6.9305     |
12.5097               | 13.0888
General and Operations Managers     | 281500                            | 288100
| 6600           | 2.3               | 14180  | 30600     | 51380             |
57.11              | 118789            | Bachelor's degree     | 5 years or
more   | nan          | True                          | 5.0373  | 10.8703    |
18.2522               | 15.9076
Legislators                         | 2300                              | 2300
| 0              | 0                 | 140    | 190       | 330               |
29.5413            | 61446             | Bachelor's degree     | Less than 5
years | nan          | True                          | 6.08696 | 8.26087    |
14.3478               | 14.3478
```

```
Advertising and Promotions Managers | 5200                              | 5200
| 0              | 0                | 210    | 690       | 900               |
65.4               | 136032             | Bachelor's degree     | Less than 5
years | nan          | True                          | 4.03846 | 13.2692    |
17.3077              | 17.3077
Marketing Managers                  | 60300                             | 61100
| 800            | 1.3              | 2880   | 6580      | 10260             |
81.59              | 169707             | Bachelor's degree     | 5 years or
more   | nan         | True                          | 4.77612 | 10.9121    |
17.0149              | 15.6882
Sales Managers                      | 108600                            | 109900
| 1300           | 1.2              | 5320   | 10500     | 17120             |
63.81              | 132725             | Bachelor's degree     | Less than 5
years | nan          | True                          | 4.89871 | 9.66851    |
15.7643              | 14.5672
Public Relations Managers           | 9200                              | 9500
| 300            | 3.3              | 420    | 820       | 1540              |
74.82              | 155626             | Bachelor's degree     | 5 years or
more   | nan         | True                          | 4.56522 | 8.91304    |
16.7391              | 13.4783
Fundraising Managers                | 3800                              | 3900
| 100            | 2.6              | 170    | 340       | 610               |
58.84              | 122387             | Bachelor's degree     | 5 years or
more   | nan         | True                          | 4.47368 | 8.94737    |
16.0526              | 13.4211
Administrative Services Managers    | 37400                             | 38400
| 1000           | 2.7              | 2660   | 3130      | 6790              |
51.81              | 107765             | Bachelor's degree     | Less than 5
years | nan          | True                          | 7.1123  | 8.36898    |
18.1551              | 15.4813
… (679 rows omitted)
```

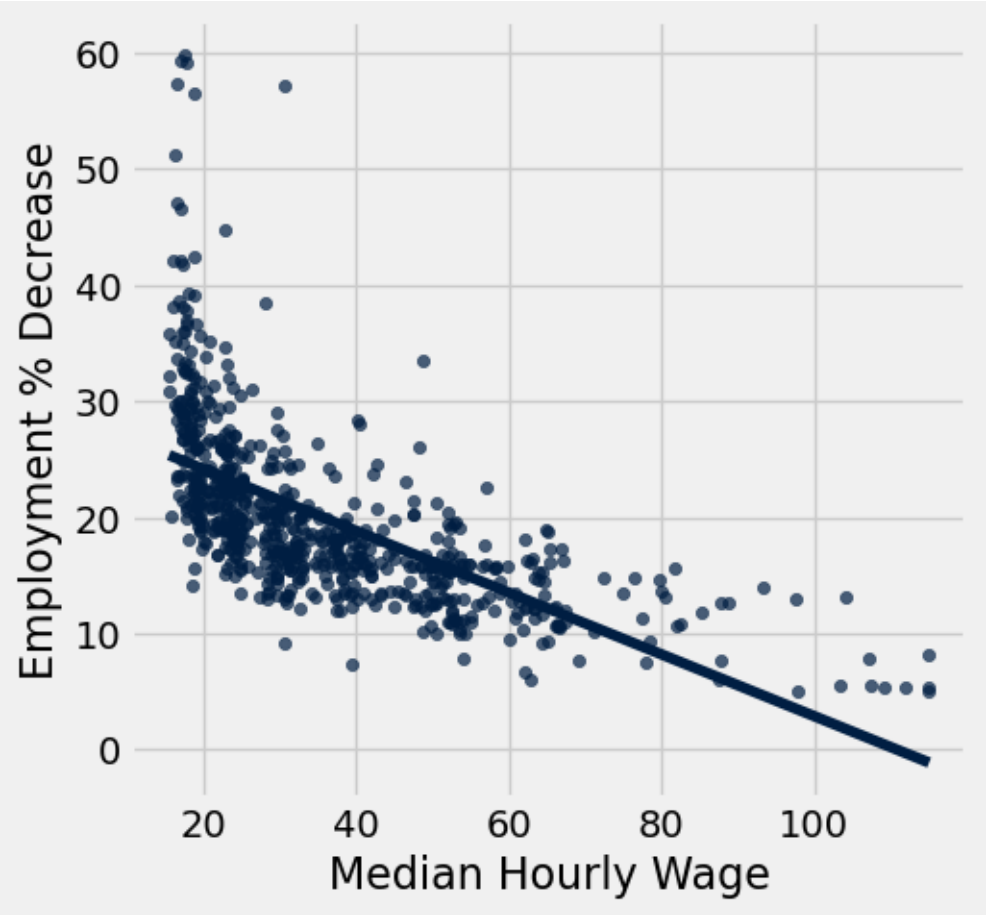
Question 2.1 Copy the cell where you loaded the dataset in section 1, reduce the table to only include the relevant data for what you would like to use in your regression model. The table should only have 2 columns of intest since this is simple linear regression.

[945]:
```
# 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
# Here, I am removing any occupations that have an employment % decrease that␣
 ↪are greater than 40. This is done to clean such outliers and improve the␣
 ↪linearity of the two variables.
```

```
my_data = my_data_raw.drop("Base Quarter Employment Estimate", "Projected␣
 ↪Quarter Employment Estimate", "Numeric Change", "Percentage Change",␣
 ↪"Exits", "Transfers", "Total Job Openings", "Median Annual Wage", "Entry␣
 ↪Level Education", "Work Experience", "Job Training", "Exit %", "Transfer %",␣
 ↪"Occupational Title", "Employment % Increase", "Employment Projection Growth?
 ↪")
my_data.show(5)
```

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

[946]:
```
# 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)
```



[947]:
```
grader.check("q2_1")
```

[947]:
```
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`.

[948]:
```
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, "Median Hourly Wage", "Employment % Decrease")
cor
```

[948]:
```
-0.62963950380787814
```

[949]:
```
grader.check("q2_2")
```

[949]:
```
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.

Null Hypothesis: There is no correlation/association between the variables, and the data values are what they are soley due to chance. Alternative Hypothesis: There is a correlation/association between the variables. Jobs with higher hourly wage will likely have employment % decrease that are changing towards one direction (higher or lower) and jobs with lower hourly wage will likely have employment % decrease that are changing towards the other direction.

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 correlation value gives the same information about the variables' assoication direction for each resample (positive or negative). Additionally, the slope is heavily dependent on the correlation value as slope = correlation * (standard deviation of y / standard deviation of x), showing their connection.

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`.

[950]:
```
def one_resample_r(tbl, col1, col2):
    resample = tbl.sample()
    resampleCorrelation = correlation(resample, col1, col2)
    return resampleCorrelation


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

one_resample = one_resample_r(my_data, "Median Hourly Wage", "Employment %␣
 ↪Decrease")
one_resample
```

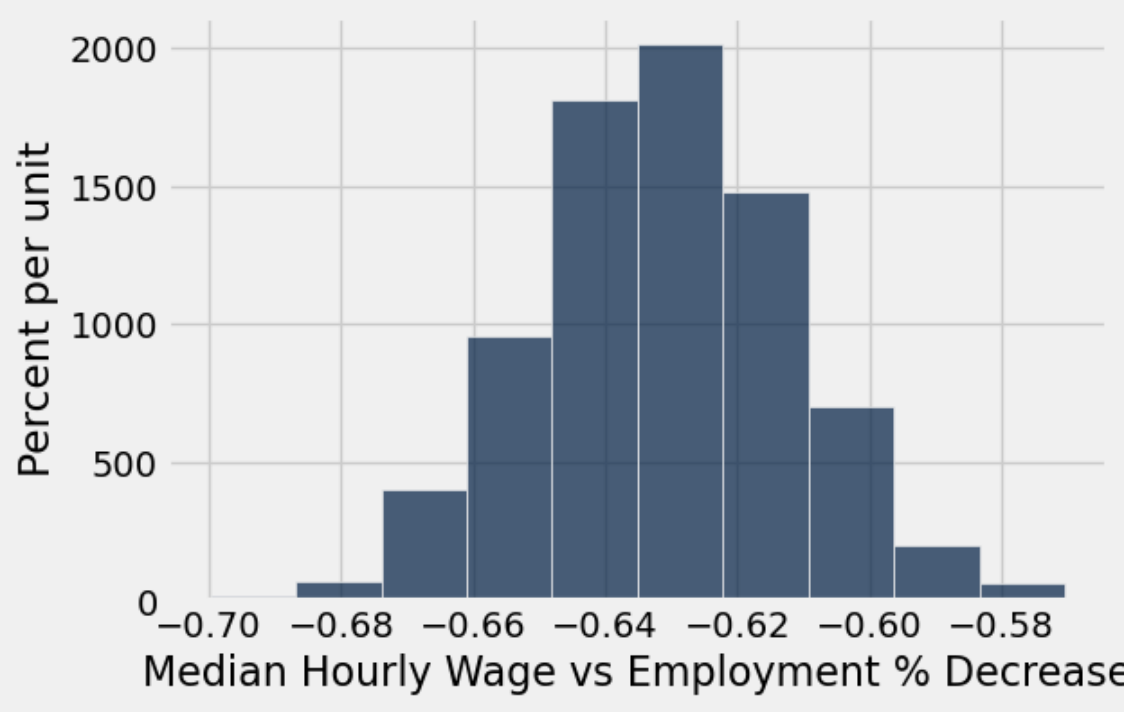
[950]:
```
-0.64675929448796265
```

[951]:
```
grader.check("q2_5")
```

[951]:
```
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.

[952]:
```
resampled_correlations = make_array()
for i in np.arange(1000):
    singleResampleR = one_resample_r(my_data, "Median Hourly Wage", "Employment␣
 ↪% Decrease")
    resampled_correlations = np.append(resampled_correlations, singleResampleR)

resampled_correlations
# Uncomment the line of code below and change column names to match your dataset
Table().with_column("Median Hourly Wage vs Employment % Decrease",␣
 ↪resampled_correlations).hist()
```



[953]:
```
grader.check("q2_6")
```

[953]:
```
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.

[954]:
```
lower_bound = percentile(2.5, resampled_correlations)
upper_bound = percentile(97.5, resampled_correlations)
reject = True


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

```
95% CI: [-0.6680314616763057, -0.5930611020927485] , Reject the null: True
```

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`****.*

[955]:
```
# 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, "Median Hourly Wage", "Employment % Decrease")).
 ↪item(0)
my_intercept = (parameters(my_data, "Median Hourly Wage", "Employment %␣
 ↪Decrease")).item(1)
print (my_slope, my_intercept)
```

```
-0.26606622101661465 29.488966578530142
```

Question 2.9:

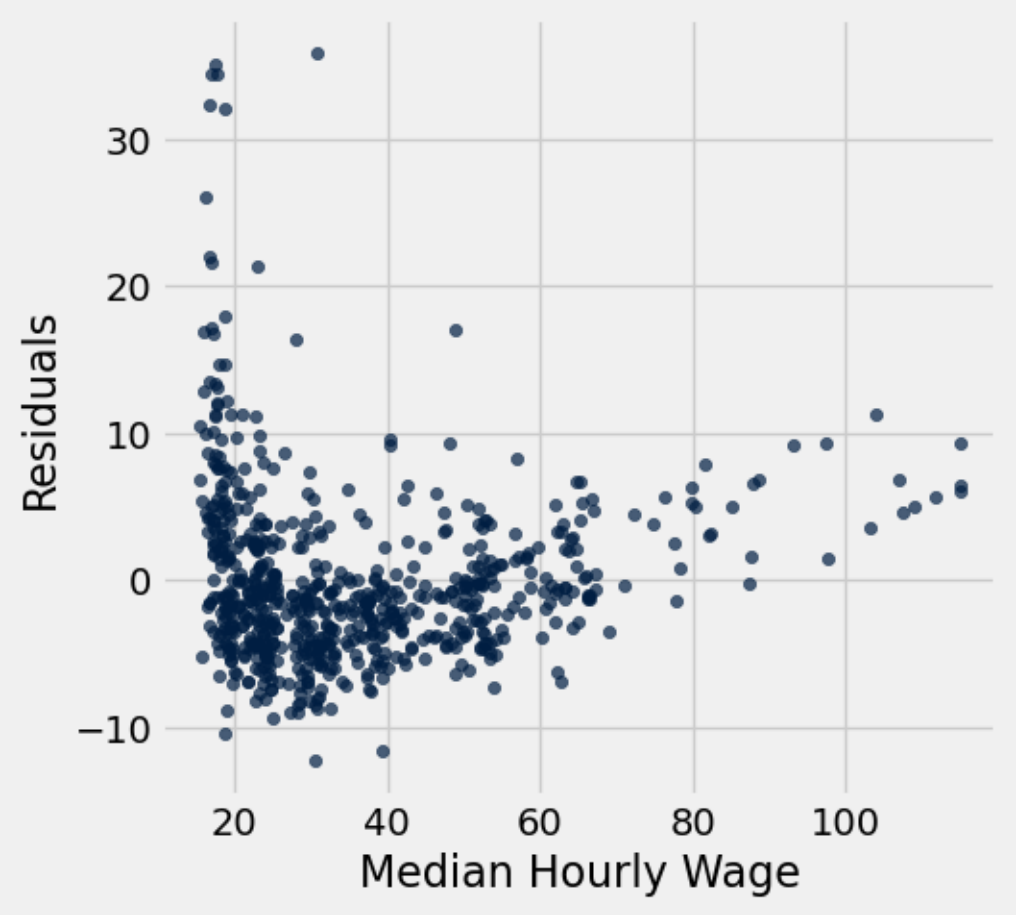
Draw a scatter plot of the residuals with the line of best fit for the 2 variables.

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!

[956]:
```
predicted_var2 = (my_slope * my_data.column("Median Hourly Wage")) +␣
 ↪my_intercept
residuals_var2 = (my_data.column("Employment % Decrease")) - predicted_var2


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


# Now generate a scatter plot of the residuals!
```

```
# Uncommment the line of code below and change "Column 1" to match variable 1␣
 ↪used in your linear regression analysis
originalTable_with_residuals.scatter("Median Hourly Wage", "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.

Due to the fact that the original data are dispersed about the line y = 0, but not evenly, linear regression may not be the best model for this case. For instance, from Median Hourly Wage values (x variables) 0 to 50, there are residual values that reach 15 and above, while there are none that reach -15 and below. In other words, the predicted y values for the x range 0 to 50 are underestimated at greater values than they are overestimated. Another observation was that past the x variable 100, there are no predicted values that are overestimated as there are no residuals that are less than 0. In tie to this observation, residual values that indicated overestimation (below 0) begin declining in amount and increasing in their value (becoming closer to 0 from the negative side) starting from the x value 40. In conclusion, a linear regression model that includes one single regression line may not be the best model for this case. However, multiple different regression lines

that vary depending on the x variable may fit the scenario better.

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

[957]:
```
correlation(originalTable_with_residuals, "Residuals", "Median Hourly Wage")
```

[957]:
```
1.8562800063398844e-16
```

[958]:
```
residual_corr = 2
#Correlation is very close to 0 from the positive side
```

[959]:
```
grader.check("q2_11")
```

[959]:
```
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.

The value of my choice for x that I want to predict y with is 40. This value is significant when it comes to the variable "Median Hourly Wage" because in California, an Hourly Wage of $40 is categorized under Top Earners according to ZipRecruiter. I want to know the predicted amounts of employee exits and transfers (employmeny % decrease) for jobs that are categorized under Top Earners. This would help me to discern whether high-end jobs tend to be more stable when it comes to employee's employment.

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*

[960]:
```
def one_resample_prediction(tbl, colx, coly, xvalue):
    resample = tbl.sample()
    resampleSlope = parameters(resample, colx, coly).item(0)
    resampleIntercept = parameters(resample, colx, coly).item(1)
    predictedY = (resampleSlope * xvalue) + resampleIntercept
    return predictedY
chosen_var1 = 40
```

```
maria_prediction = one_resample_prediction(my_data, "Median Hourly Wage",␣
 ↪"Employment % Decrease", chosen_var1)
maria_prediction
```

[960]: 18.658988190193597

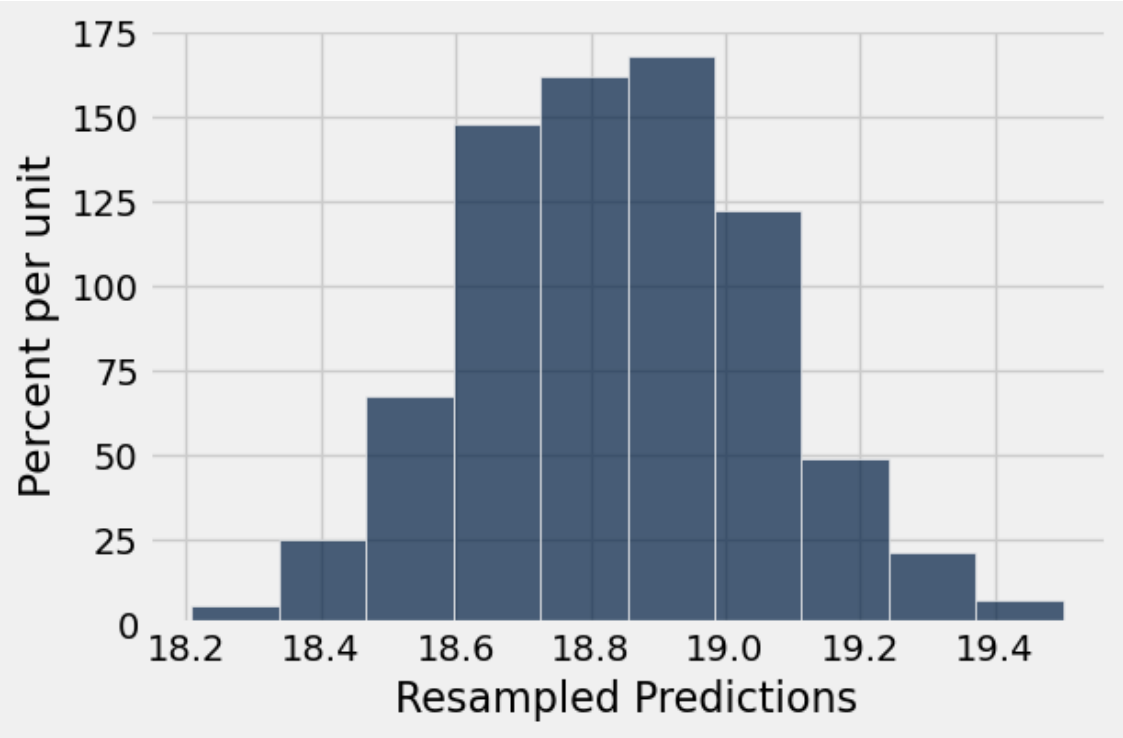
[961]:
```
grader.check("q2_13")
```

[961]: 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.

[962]:
```
resampled_predictions = make_array()

for j in np.arange(1000):
    onePrediction = one_resample_prediction(my_data, "Median Hourly Wage",␣
 ↪"Employment % Decrease", chosen_var1)
    resampled_predictions = np.append(resampled_predictions, onePrediction)

# 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.

[963]:
```
lower_bound_maria = percentile(0.5, 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: [18.31199416701492, 19.42242448514902]
```

[964]:
```
grader.check("q2_15")
```

[964]:
```
q2_15 results:
    q2_15 - 1 result:
         Test case failed
        Trying:
            all([type(lower_bound_maria) in set([float, np.float32,
np.float64]),
        Expecting nothing
        **********************************************************************
        Line 1, in q2_15 0
        Failed example:
            all([type(lower_bound_maria) in set([float, np.float32,
np.float64]),
        Exception raised:
            Traceback (most recent call last):
              File "/opt/conda/lib/python3.10/doctest.py", line 1350, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest q2_15 0[0]>", line 1
                all([type(lower_bound_maria) in set([float, np.float32,
np.float64]),
                    ^
            SyntaxError: '[' was never closed
        Trying:
            type(upper_bound_maria) in set([float, np.float32, np.float64])])
        Expecting:
            True
        **********************************************************************
        Line 2, in q2_15 0
        Failed example:
            type(upper_bound_maria) in set([float, np.float32, np.float64])])
        Exception raised:
            Traceback (most recent call last):
```

```
              File "/opt/conda/lib/python3.10/doctest.py", line 1350, in __run
                exec(compile(example.source, filename, "single",
              File "<doctest q2_15 0[1]>", line 1
                type(upper_bound_maria) in set([float, np.float32,
np.float64])])
                                                                              ^
            SyntaxError: unmatched ']'
```

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)

[965]:
```
# 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, "Median Hourly Wage", "Employment %␣
 ↪Decrease")
    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("Median Hourly Wage")),␣
 ↪max(originalTable_with_residuals.column("Median Hourly Wage"))]
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("Employment % Decrease"); # You can change the label here to be more␣
 ↪descriptive
plt.xlabel("Median Hourly Wage"); # 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.

One bias is the fact that occupations with greater than or equal to 40% employment decrease are not accounted for. This is because I cleaned my data to only include occupations with an employment % decrease that is less than 40 (to remove outliers). This causes my regression model to not fully represent every occupations from my raw data set. Additionally, the rows in my data are not conventional individual beings, but rather different occupations. Thus, I cannot weigh the rows in my data simply using median hourly salary to determine the employment % decrease. Every occupations have various different factors. One might have a lower end wage and have a higher emplyment % decrease while another occupation with similar wage may have a significantly lower employment % decrease. Some factors that might cause this are: How new an occupation is, how common an occupation position is, or how popular the occupation field is.

4 3. Classification

Recommended Reading:

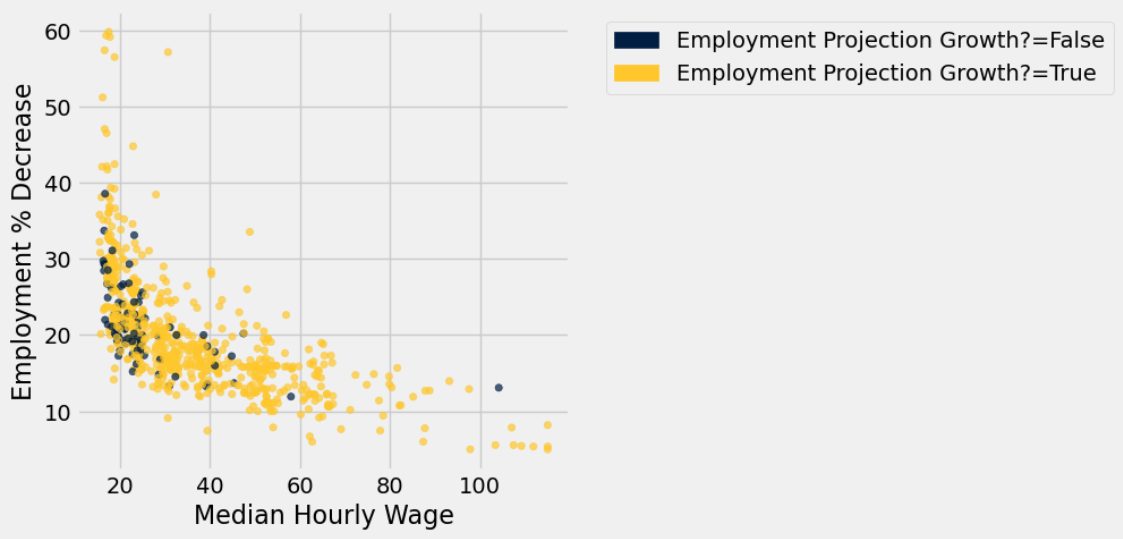
- Classification

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.

[966]:
```
my_data_raw.scatter("Median Hourly Wage", "Employment % Decrease", group =␣
↪"Employment Projection Growth?")
my_data_raw.num_rows
```

[966]: 689



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.

[967]:
```
def distance(arr1, arr2):
    euc_dist = np.sqrt(sum ((arr1 - arr2) ** 2))
    return euc_dist
```

```
# 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
```

[967]: 5.196152422706632

[968]:
```
grader.check("q3_1")
```

[968]: 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*

[969]:
```
my_data_raw
```

[969]:
```
Occupational Title                  | Base Quarter Employment Estimate |
Projected Quarter Employment Estimate | Numeric Change | Percentage Change |
Exits  | Transfers | Total Job Openings | Median Hourly Wage | Median Annual
Wage | Entry Level Education | Work Experience   | Job Training | Employment
Projection Growth? | Exit %  | Transfer % | Employment % Increase | Employment %
Decrease
Management Occupations              | 1669600                          | 1705600
| 36000          | 2.2              | 102730 | 153320    | 292050             |
63.9               | 132912             | nan                    | nan
| nan          | True                          | 6.15297 | 9.18304    | 17.4922
| 15.336
Chief Executives                   | 51800                            | 51500
| -300           | -0.6             | 3190   | 3590      | 6480               |
```

```
104.13              | 216590              | Bachelor's degree     | 5 years or
more    | nan           | False                          | 6.1583  | 6.9305     |
12.5097               | 13.0888
General and Operations Managers     | 281500                          | 288100
| 6600            | 2.3               | 14180  | 30600     | 51380              |
57.11               | 118789              | Bachelor's degree     | 5 years or
more    | nan           | True                           | 5.0373  | 10.8703    |
18.2522               | 15.9076
Legislators                         | 2300                            | 2300
| 0               | 0                 | 140    | 190       | 330                |
29.5413             | 61446               | Bachelor's degree     | Less than 5
years | nan           | True                           | 6.08696 | 8.26087    |
14.3478               | 14.3478
Advertising and Promotions Managers | 5200                            | 5200
| 0               | 0                 | 210    | 690       | 900                |
65.4                | 136032              | Bachelor's degree     | Less than 5
years | nan           | True                           | 4.03846 | 13.2692    |
17.3077               | 17.3077
Marketing Managers                  | 60300                           | 61100
| 800             | 1.3               | 2880   | 6580      | 10260              |
81.59               | 169707              | Bachelor's degree     | 5 years or
more    | nan           | True                           | 4.77612 | 10.9121    |
17.0149               | 15.6882
Sales Managers                      | 108600                          | 109900
| 1300            | 1.2               | 5320   | 10500     | 17120              |
63.81               | 132725              | Bachelor's degree     | Less than 5
years | nan           | True                           | 4.89871 | 9.66851    |
15.7643               | 14.5672
Public Relations Managers           | 9200                            | 9500
| 300             | 3.3               | 420    | 820       | 1540               |
74.82               | 155626              | Bachelor's degree     | 5 years or
more    | nan           | True                           | 4.56522 | 8.91304    |
16.7391               | 13.4783
Fundraising Managers                | 3800                            | 3900
| 100             | 2.6               | 170    | 340       | 610                |
58.84               | 122387              | Bachelor's degree     | 5 years or
more    | nan           | True                           | 4.47368 | 8.94737    |
16.0526               | 13.4211
Administrative Services Managers    | 37400                           | 38400
| 1000            | 2.7               | 2660   | 3130      | 6790               |
51.81               | 107765              | Bachelor's degree     | Less than 5
years | nan           | True                           | 7.1123  | 8.36898    |
18.1551               | 15.4813
… (679 rows omitted)
```

[970]:
```
my_data_raw = my_data_raw.where("Entry Level Education", are.
↪not_equal_to("nan"))
```

```
my_data_raw
```

[970]:
```
Occupational Title                   | Base Quarter Employment Estimate |
Projected Quarter Employment Estimate | Numeric Change | Percentage Change |
Exits | Transfers | Total Job Openings | Median Hourly Wage | Median Annual Wage
| Entry Level Education | Work Experience    | Job Training | Employment
Projection Growth? | Exit %  | Transfer % | Employment % Increase | Employment %
Decrease
Chief Executives                     | 51800                            | 51500
| -300           | -0.6              | 3190  | 3590      | 6480               |
104.13             | 216590            | Bachelor's degree     | 5 years or
more   | nan           | False                          | 6.1583  | 6.9305     |
12.5097               | 13.0888
General and Operations Managers      | 281500                           | 288100
| 6600           | 2.3               | 14180 | 30600     | 51380              |
57.11              | 118789            | Bachelor's degree     | 5 years or
more   | nan           | True                           | 5.0373  | 10.8703    |
18.2522               | 15.9076
Legislators                          | 2300                             | 2300
| 0              | 0                 | 140   | 190       | 330                |
29.5413            | 61446             | Bachelor's degree     | Less than 5
years | nan           | True                           | 6.08696 | 8.26087    |
14.3478               | 14.3478
Advertising and Promotions Managers  | 5200                             | 5200
| 0              | 0                 | 210   | 690       | 900                |
65.4               | 136032            | Bachelor's degree     | Less than 5
years | nan           | True                           | 4.03846 | 13.2692    |
17.3077               | 17.3077
Marketing Managers                   | 60300                            | 61100
| 800            | 1.3               | 2880  | 6580      | 10260              |
81.59              | 169707            | Bachelor's degree     | 5 years or
more   | nan           | True                           | 4.77612 | 10.9121    |
17.0149               | 15.6882
Sales Managers                       | 108600                           | 109900
| 1300           | 1.2               | 5320  | 10500     | 17120              |
63.81              | 132725            | Bachelor's degree     | Less than 5
years | nan           | True                           | 4.89871 | 9.66851    |
15.7643               | 14.5672
Public Relations Managers            | 9200                             | 9500
| 300            | 3.3               | 420   | 820       | 1540               |
74.82              | 155626            | Bachelor's degree     | 5 years or
more   | nan           | True                           | 4.56522 | 8.91304    |
16.7391               | 13.4783
Fundraising Managers                 | 3800                             | 3900
| 100            | 2.6               | 170   | 340       | 610                |
58.84              | 122387            | Bachelor's degree     | 5 years or
more   | nan           | True                           | 4.47368 | 8.94737    |
```

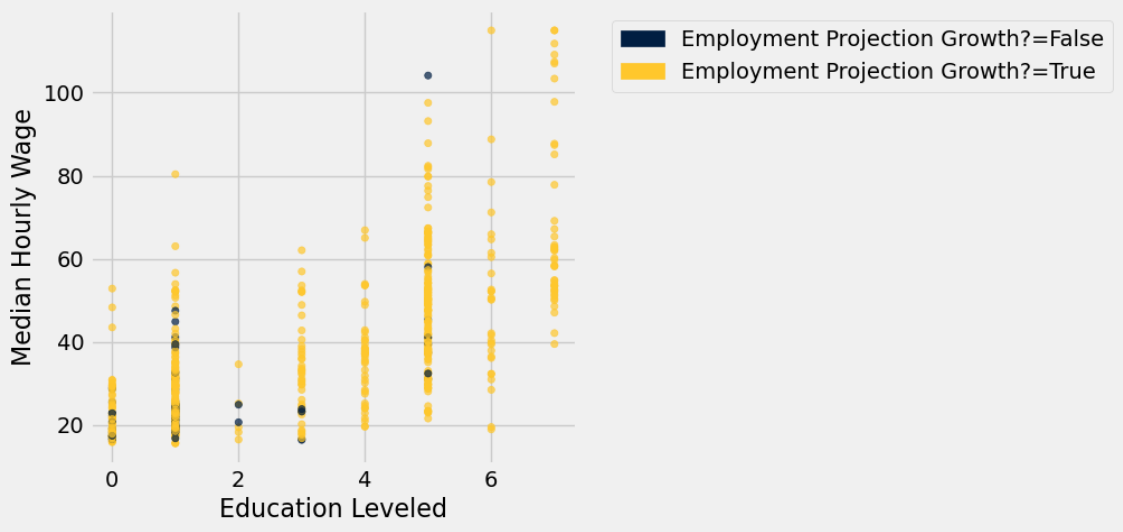
```
16.0526               | 13.4211
Administrative Services Managers    | 37400                              | 38400
| 1000           | 2.7               | 2660  | 3130      | 6790               |
51.81              | 107765            | Bachelor's degree     | Less than 5
years | nan          | True                           | 7.1123  | 8.36898    |
18.1551               | 15.4813
Facilities Managers                 | 20700                              | 21300
| 600            | 2.9               | 1350  | 1800      | 3750               |
50.09              | 104187            | Bachelor's degree     | Less than 5
years | nan          | True                           | 6.52174 | 8.69565    |
18.1159               | 15.2174
… (657 rows omitted)
```

5 "Education Leveled" + Correlation between Education and Income (Implemented Section)

[971]:
```
#I ranked the attributes of "Entry Level Education" according to prestige (7 -␣
↪highest prestiege and 0 - lowest prestiege)

levelArray = make_array()
def levelEd (education):
    if (education == "No formal educational credential"):
        return 0
    if (education == "High school diploma or equivalent"):
        return 1
    if (education == "Some college, no degree"):
        return 2
    if (education == "Postsecondary non-degree award"):
        return 3
    if (education == "Associate's degree"):
        return 4
    if (education == "Bachelor's degree"):
        return 5
    if (education == "Master's degree"):
        return 6
    if (education == "Doctoral or professional degree"):
        return 7
levelArray = my_data_raw.apply(levelEd, "Entry Level Education")
my_data_raw = my_data_raw.with_column("Education Leveled", levelArray)
my_data_raw.show()
```

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

[972]:
```
my_data_raw.scatter("Education Leveled", "Median Hourly Wage", group =␣
↪"Employment Projection Growth?")
```



[973]:
```
my_data_raw.where("Median Hourly Wage", are.above(60)).num_rows
```

[973]: 70

[974]:
```
my_data_raw.num_rows
```

[974]: 667

[975]:
```
70/667
```

[975]: 0.10494752623688156

[992]:
```
correlation(my_data_raw, "Education Leveled", "Median Hourly Wage")
```

[992]: 0.69950735988783552

[976]:
```
trainNum = 667 * 0.75
testNum = 667 * 0.25
print (trainNum, testNum)
#round 500.25 training set down to 500 and round 166.75 testing set up to 167
```

```
500.25 166.75
```

6 K Fold Cross Validation (Implemented Section)

[977]:
```
#k fold cross validation

shuffled_table = my_data_raw.sample(with_replacement = False)
test1 = (np.arange(133))
```

```
train1 = (np.arange(133, 667))

test2 = (np.arange(133, 266))
firstRange2 = np.arange(133)
secondRange2 = np.arange(266, 667)
train2 = (np.concatenate((firstRange2, secondRange2)))

test3 = (np.arange(266, 399))
firstRange3 = np.arange(266)
secondRange3 = np.arange(399, 667)
train3 = (np.concatenate((firstRange3, secondRange3)))

test4 = (np.arange(399, 532))
firstRange4 = np.arange(399)
secondRange4 = np.arange(532, 667)
train4 = (np.concatenate((firstRange4, secondRange4)))

test5 = (np.arange(532, 667))
train5 = (np.arange(532))


#print("Training set 1:\t",   train1.num_rows, "examples")
#print("Test set 1:\t",       test1.num_rows, "examples")

#print("Training set 2:\t",   train2.num_rows, "examples")
#print("Test set 2:\t",       test2.num_rows, "examples")

#print("Training set 3:\t",   train3.num_rows, "examples")
#print("Test set 3:\t",       test3.num_rows, "examples")

#print("Training set 4:\t",   train4.num_rows, "examples")
#print("Test set 4:\t",       test4.num_rows, "examples")

#print("Training set 5:\t",   train5.num_rows, "examples")
#print("Test set 5:\t",       test5.num_rows, "examples")
```

[978]:
```
grader.check("q3_2")
```

[978]:
```
q3_2 results:
    q3_2 - 1 result:
         Test case failed
        Trying:
            train.num_rows == round(75* my_data_raw.num_rows/100,0)
        Expecting:
            True
        **********************************************************************
        Line 2, in q3_2 0
```

```
        Failed example:
            train.num_rows == round(75* my_data_raw.num_rows/100,0)
        Expected:
            True
        Got:
            False

    q3_2 - 2 result:
         Test case failed
        Trying:
            test.num_rows == round(25* my_data_raw.num_rows/100,0)
        Expecting:
            True
        **********************************************************************
        Line 2, in q3_2 1
        Failed example:
            test.num_rows == round(25* my_data_raw.num_rows/100,0)
        Expected:
            True
        Got:
            False

    q3_2 - 3 result:
         Test case passed
```

6.1 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!

[979]:
```
chosen_features = make_array("Base Quarter Employment Estimate", "Projected␣
↪Quarter Employment Estimate", "Numeric Change", "Exits", "Transfers", "Total␣
↪Job Openings", "Median Hourly Wage", "Median Annual Wage", "Exit %",␣
↪"Transfer %", "Employment % Increase", "Employment % Decrease", "Education␣
↪Leveled")
chosen_features
```

[979]:
```
array(['Base Quarter Employment Estimate',
       'Projected Quarter Employment Estimate', 'Numeric Change', 'Exits',
       'Transfers', 'Total Job Openings', 'Median Hourly Wage',
       'Median Annual Wage', 'Exit %', 'Transfer %',
       'Employment % Increase', 'Employment % Decrease',
       'Education Leveled'],
      dtype='<U37')
```

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.

[980]:
```
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("Distance", distances)
    nearest_neighbors = train_with_distances.sort("Distance").take(np.arange(k))
    most_common_label = nearest_neighbors.group("Employment Projection Growth?
 ↪").sort("count", descending = True).column("Employment Projection Growth?").
 ↪item(0)
    return most_common_label

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

[980]:
```
False
```

6.1.1 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.

[981]:
```
def three_classify(row):
    prediction = classify(row, 3, train, chosen_features)
    return prediction

def getLabels(testData):
    predictionsArray = make_array()
    for eachRow in (np.arange(testData.num_rows)):
        testRow = testData.row(eachRow)
        eachPrediction = three_classify(testRow)
        predictionsArray = np.append(predictionsArray, eachPrediction)
    return predictionsArray == 1

labels = getLabels(test)

test_with_prediction = test.with_column("Prediction", labels)

def checkIfCorrect (testingData, actual, predicted):
    numCorrect = 0
    for rowEach in (np.arange(testingData.num_rows)):
        if (testingData.column(actual).item(rowEach) == testingData.
 ↪column(predicted).item(rowEach)):
            numCorrect = numCorrect + 1
    return numCorrect

accuracy = (checkIfCorrect(test_with_prediction, "Employment Projection Growth?
 ↪", "Prediction"))/test_with_prediction.num_rows
accuracy
```

[981]:
```
0.8740740740740741
```

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.

[982]:
```
# Feel free to use multiple lines of code
# but make sure to assign test_correctness to the proper table!
def correctnessColumn (dataTesting, actualC, predictedC):
    correctness = make_array()
    for oneRow in (np.arange(dataTesting.num_rows)):
        if ((dataTesting.column(actualC)).item(oneRow) == (dataTesting.
 ↪column(predictedC)).item(oneRow)):
            result = True
        if ((dataTesting.column(actualC)).item(oneRow) != (dataTesting.
 ↪column(predictedC)).item(oneRow)):
            result = False
        correctness = np.append(correctness, result)
    return correctness

correctnessArray = correctnessColumn(test_with_prediction, "Employment␣
 ↪Projection Growth?", "Prediction")
test_correctness = test_with_prediction.with_column("Was correct",␣
 ↪correctnessArray == 1)
test_correctness.sort('Was correct', descending = False).show(15)
```

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

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.

Majority of the occupations in the results table are projected to have an employment shrinkage (being marked with 'False' for the column 'Employment Projection Growth?')(11 out of the first 15 rows). With the classifier failing to predict correctly for such occupations (despite there being significantly less occupations that are projected to shrink) we can assume that the classifier is specifically biased against occupations projected to have employment shrinkage.

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.

The K nearest neighbor algorithm uses the classification data of rows that were incoporated onto it (training set). It is able to predict the occupation projections (shrinkage/growth) of a certain occupation using the classification of those incorporated rows that have similar attibutes. Thus, attempting to predict the label of a training set would mean that there is aready an incorporated row that has the exact same attributions, causing an inaccurate leaning towards one label. To prevent this, we have a separate test set that are not one of the incorporated data, allowing an accurate prediction towards one label.

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

If the K was even numbered, there is a chance that a testing set row results in k nearest neighbors in which half are one label, and the other half are the other label. This would mean that there is an equal number for both labels out of the k nearest neighbors. In this case, the algorithm is not

able to know which label is more common out of the k nearest neighbors, failing to give a single label prediction for the testing set.

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.

7 Paired T Test (Implemented Section)

7.1 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.

[984]:
```
my_data_raw.show()
```

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

[985]:
```
# 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
#shuffled_table = my_data_raw.sample(with_replacement = False)
withoutWage = make_array()
trainArray = make_array(train1, train2, train3, train4, train5)
testArray = make_array(test1, test2, test3, test4, test5)
for x in np.arange(5):
    train = shuffled_table.take(trainArray[x])
    test = shuffled_table.take(testArray[x])

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

    different_features = make_array("Base Quarter Employment Estimate",␣
 ↪"Projected Quarter Employment Estimate", "Numeric Change", "Exits",␣
 ↪"Transfers", "Total Job Openings", "Exit %", "Transfer %", "Employment %␣
 ↪Increase", "Employment % Decrease", "Education Leveled")
```

```
    def another_classifier(newRow):
        newPrediction = classify(newRow, 5, train, different_features)
        return newPrediction

    def newGetLabels(testDataNew):
        predictionsArrayNew = make_array()
        for eachRowNew in (np.arange(testDataNew.num_rows)):
            testRowNew = testDataNew.row(eachRowNew)
            eachPredictionNew = another_classifier(testRowNew)
            predictionsArrayNew = np.append(predictionsArrayNew,␣
 ↪eachPredictionNew)
        return predictionsArrayNew == 1

    new_labels = newGetLabels(test)
    new_test_with_prediction = test.with_column("New Prediction", new_labels)
    new_labels_correct = checkIfCorrect(new_test_with_prediction,"Employment␣
 ↪Projection Growth?", "New Prediction")
    new_accuracy = new_labels_correct/new_test_with_prediction.num_rows
    withoutWage = np.append(withoutWage, new_accuracy)
withoutWage
```

[985]:
```
array([ 0.87969925,  0.89473684,  0.89473684,  0.87218045,  0.91111111])
```

[986]:
```
# 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
#shuffled_table = my_data_raw.sample(with_replacement = False)
withWage = make_array()
trainArray = make_array(train1, train2, train3, train4, train5)
testArray = make_array(test1, test2, test3, test4, test5)
for x in np.arange(5):
    train = shuffled_table.take(trainArray[x])
    test = shuffled_table.take(testArray[x])

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

    different_features = make_array("Base Quarter Employment Estimate",␣
 ↪"Projected Quarter Employment Estimate", "Numeric Change", "Exits",␣
 ↪"Transfers", "Total Job Openings", "Median Hourly Wage", "Median Annual␣
 ↪Wage", "Exit %", "Transfer %", "Employment % Increase", "Employment %␣
 ↪Decrease", "Education Leveled")

    def another_classifier(newRow):
        newPrediction = classify(newRow, 5, train, different_features)
        return newPrediction
```

```
    def newGetLabels(testDataNew):
        predictionsArrayNew = make_array()
        for eachRowNew in (np.arange(testDataNew.num_rows)):
            testRowNew = testDataNew.row(eachRowNew)
            eachPredictionNew = another_classifier(testRowNew)
            predictionsArrayNew = np.append(predictionsArrayNew,␣
 ↪eachPredictionNew)
        return predictionsArrayNew == 1

    new_labels = newGetLabels(test)
    new_test_with_prediction = test.with_column("New Prediction", new_labels)
    new_labels_correct = checkIfCorrect(new_test_with_prediction,"Employment␣
 ↪Projection Growth?", "New Prediction")
    new_accuracy = new_labels_correct/new_test_with_prediction.num_rows
    withWage = np.append(withWage, new_accuracy)
withWage
```

[986]:
```
array([ 0.84962406,  0.85714286,  0.87969925,  0.85714286,  0.85185185])
```

[996]:
```
#Paired T Test
```

[1005]:
```
withoutWage-withWage
```

[1005]:
```
array([ 0.03007519,  0.03759398,  0.01503759,  0.01503759,  0.05925926])
```

[1003]:
```
avg = np.average(withoutWage - withWage)
avg
```

[1003]:
```
0.031400724032302985
```

[997]:
```
std = np.std(withoutWage - withWage)
```

[999]:
```
#standard Error
ste = std/(np.sqrt(5))
```

[1007]:
```
t_value = avg/ste
t_value
```

[1007]:
```
4.2703847897928062
```

[1009]:
```
#Degrees of Freedom (df)
df = 5-1
#P value at df = 4 and t value = 4.2703847897928062 is 0.013
```

[1010]:
```
#Null Hyp: Mean of the difference of the two models is 0 -> adding wage does␣
 ↪not have a significant effect on the model
```

```
#Alt Hyp: Mean of the difference of the two models is != 0 -> adding wage does␣
 ↪have a significant effect on the model

#P-value 0.013 < 0.05. This means that we can reject the null hyp. Addition of␣
 ↪wages (hourly + annual) as features does have a significant effect on the␣
 ↪performance of the KNN model
```

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 has about a 6% (0.06) increase in accuracy from my previous classifier. Similar to the old classifier, majority of the incorrect predictions are made for jobs projected to face employment shrinkage, but the new classifier was able to predict correctly more of such jobs (9 out of the first 15 rows had "false" for "Employment Projection Growth?" column)(compared to 11 out of the first 15 for the previous classifier).

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 increasing the K from the previous value of 3, and learned that at ranges 11 to 17 (odd only), we get the highest accuracy. For the features, I removed all attributes from the first classifier except for "Median Hourly Wage", "Employment % Increase", and "Employment % Decrease." Due to the fact that the result in "Employment Projection Growth?" is dependent on how many individuals left and joined an occupation during a certain year, attributes like "Employment % Increase"/"Employment % Decrease", and "Exits" + "Transfers"/"Total Job Openings" (that measure the joining/leaving of employees) can be used to improve the classifier. K values 11 - 17 had the most consistency in highest accuracy, and thus, using a value from that range can also improve the classifier.

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? During my senior year in high school, I took the class "AP Art History" because previous students often classified it as an "easy AP class." However, I found the class work load and exams to be pretty challenging. In my case, the misclassification of AP Art History's difficulty level, led me to taking the class with wrong expectations and struggling in it. However, the class was also seen by some other students in my year to be easy, meaning that the previous students' classification is biased against individuals (like me) that are not as experienced or gifted in the subject matter. **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.

By conducting regression inference on my dataset, I learned that variables like 'income level' and 'occupation employment projection' are not as closely correlated as I had expected. I initially chose my research topic being curious about whether technological advancements have really shrunk the employment numbers of lower end jobs compared to higher end jobs. However, the correlation level was moderately strong, and such correlation was only achieved because I had removed occupations that had more than 40 in 'Employment % Decrease' as they were seen as outliers. Through the classification section, I learned that one can make descently accurate predictions about occupation employment projection by using transfers %, exits %, and job openings % to derive variables like "Employement % Decrease" and "Employment % Increase" and using them as attributes to a classification algorithm together with the hourly wage. I also learned that using hourly wage as attributes is better than using annual wage and employment % increase as attributes is better than using employment % decrease at classifying accurate employment projections.

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