Table of Contents

[Converting Strings to Numbers 1](#_Toc29281739)

[Try – Except 2](#_Toc29281740)

[Comprehensions Introduction 3](#_Toc29281741)

[Visualizations 5](#_Toc29281742)

[Scatter Plots 7](#_Toc29281743)

[Bar Plots 8](#_Toc29281744)

[Grouped Bar Plots 9](#_Toc29281745)

[Frequency Histograms 10](#_Toc29281746)

[Displaying Multiple Graphs 11](#_Toc29281747)

[Scatter Matrix 14](#_Toc29281748)

[Correlation Heatmap 16](#_Toc29281749)

[SQLAlchemy 17](#_Toc29281750)

[Filtered Queries 17](#_Toc29281751)

[Aggregate Queries 17](#_Toc29281752)

[Creating Database Table from a DataFrame 18](#_Toc29281753)

[Creating an In-Memory Database 20](#_Toc29281754)

[Reading from a Database 20](#_Toc29281755)

**Note:**

You are encouraged to work with others and you can discuss your solutions over-the-shoulder as well but DO NOT SHARE COPIES of YOUR WORK ELECTONICALLY OR IN ANY OTHER FORM. Keep good karma with your career network and protect your reputation – do not plagiarize.

## Converting Strings to Numbers

When you need to convert strings to numbers you can with *int()* and *float()*.

Example 1: Converting Strings to Numbers

This example demonstrates how to convert string values to a whole number and a floating-point number.

|  |
| --- |
| unformattedInt = '233'  wholeNumber = int(unformattedInt)  print(wholeNumber)  unformattedFloat = '12.99'  print(unformattedFloat)  floatNumber = float(unformattedFloat)  print(wholeNumber + floatNumber) |

The output verifies that the conversions from strings to numbers has been successful:

|  |
| --- |
| ﻿233  12.99  245.99 |

## Try – Except

Try except is really helpful when parsing data when you do not know if the inputs are delivered in the expected format. Try-Except blocks allow you to handle erroneous conversions gracefully so your application does not crash while running.

Without a try-except block the following lines of code would crash the program:

|  |
| --- |
| ﻿wholeNumber = int("abc")  print(wholeNumber) |

The output when running these two instructions leads to the error below:

|  |
| --- |
| File "<ipython-input-138-47997d7f31af>", line 1, in <module>  wholeNumber = int("abc")  ValueError: invalid literal for int() with base 10: 'abc' |

Example 2: Try-Except

This example demonstrates how to perform string to number conversions which handle invalid inputs without crashing the program:

|  |
| --- |
| # Converts string to float & returns it as a value rounded to 2 decimal places.  # If conversion fails then 'None' is returned.  def convertStringToFloat(str):  try:  floatNumber = float(str)  formattedFloat = round(floatNumber, 2)  print("The value has successfully been converted.")  print(formattedFloat)  return formattedFloat  except:  print("An error occurred during the conversion.")  return None # None represents a null object in Python.  formattedValue = convertStringToFloat("23.233342")  if(formattedValue != None):  print("The converted value is: ", str(formattedValue))    print("")  formattedValue = convertStringToFloat("abc")  if(formattedValue == None):  print("'abc' could not be converted to a float.") |

The output after running the program shows a more user-friendly result even when the input is invalid:

|  |
| --- |
| ﻿The value has successfully been converted.  23.23  The converted value is: 23.23  An error occurred during the conversion.  'abc' could not be converted to a float. |

Exercise 1 (1 mark)

Starting with the following code shell, modify the *convertStringToFloat()* function so it performs division safely within a try-except block.

|  |
| --- |
| # This function performs division. It does not print anything.  def convertStringToFloat(numerator, denominator):  # your code goes here.  def showResult(result):  if(result != None):  print("The value is " + str(result))  else:  print("An error occurred during the division.")    result = convertStringToFloat(15, 5)  showResult(result)  print("")  result = convertStringToFloat(15,0)  showResult(result)  print("")  result = convertStringToFloat(21,3)  showResult(result) |

If successful your output should look like the following:

|  |
| --- |
| The value is 3.0  An error occurred during the division.  The value is 7.0 |

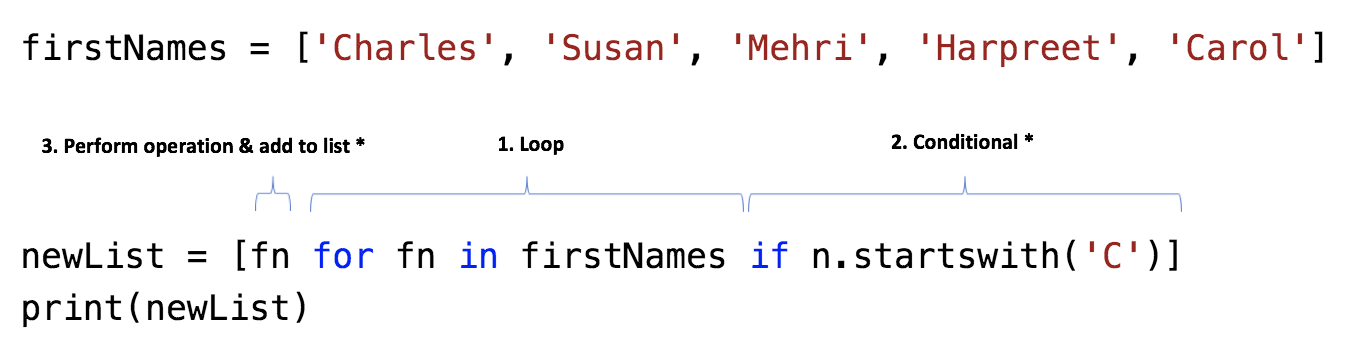
Show your revised program here:

|  |
| --- |
| # This function performs division. It does not print anything. def convertStringToFloat(numerator, denominator):  try:  return float(numerator/denominator)  except:  return None  def showResult(result):  if (result != None):  print("The value is " + str(result))  else:  print("An error occurred during the division.") |

## Comprehensions Introduction

Comprehensions allow us to loop through lists, perform operations on elements and add the resulting values to a new list. Figure 1 shows the three sections of a comprehension. The loop section is mandatory but the conditional and operations section are optional \*. In this example no operation is performed on the list element but the value is appended to the new list.

Figure 1: Comprehension Introduction



The output shows the contents of the new list after the comprehension has executed:

﻿ ['Charles', 'Carol']

Exercise 2 (1 mark)

Revise Example 4 so the comprehension generates a list that contains the first letter of every name. Hint: You can extract the first letter of every name by referencing the string with its first index (fn[0]). The output should look like the following:

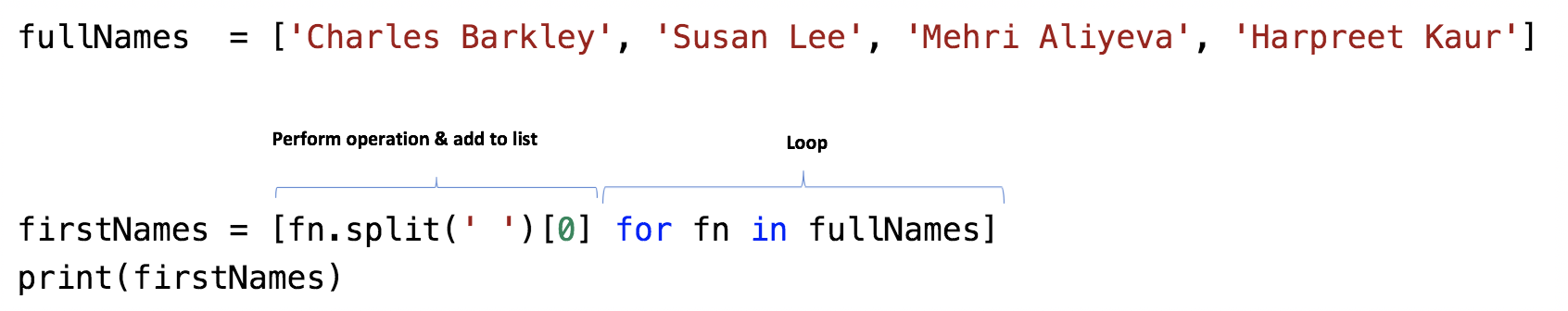
﻿ ['C', 'S', 'M', 'H', 'C']

Show your new program here. (No screenshots please)

|  |
| --- |
| first\_names = ['Charles', 'Susan', 'Mehri', 'Harpreet', 'Carol']  first\_letter\_list = [f\_letter[0] for f\_letter in first\_names] print(first\_letter\_list) |

The comprehension in Figure 2 iterates through a list of full names, splits the into first and last names into an array, and then adds the first name to a list.

Figure 2: Advanced Operation with Comprehension



The output from these instructions in Figure 2 shows the list of resulting first names:

﻿ ['Charles', 'Susan', 'Mehri', 'Harpreet']

Exercise 3 (2 marks)

Starting with the following code:

﻿ dates = ['March 1', 'March 5', 'March 8', 'March 9']

Use a comprehension to extract the day number. Your output should show an array of day values after they have been converted to integers. See Example 1 for information on how to perform the conversion to an integer:

﻿ [1, 5, 8, 9]

Show your code here:

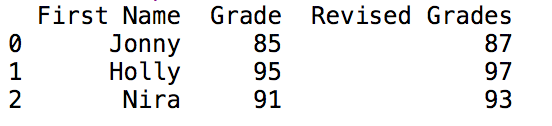
|  |
| --- |
| dates = ['March 1', 'March 5', 'March 8', 'March 9'] delimited\_dates = [int(word.split(' ')[1]) for word in dates] print(delimited\_dates) |

Example 3: Comprehensions with DataFrame Columns

Recalling that DataFrame columns are made from lists, this example shows how a comprehension can be used to update or create new DataFrame columns. For this case, the comprehension is used to update the value in the DataFrame ‘Grade’ column.

|  |
| --- |
| import pandas as pd  # Create data set.  dataSet = {'First Name': ['Jonny','Holly','Nira'], 'Grade': [85,95,91] }  # Create dataframe with data set and named columns.  df = pd.DataFrame(dataSet, columns= ['First Name', 'Grade'])  # Add 2 grades to the original score with a comprehension.  revisedGrades = [grade + 2 for grade in df['Grade']]  df['Revised Grades'] = revisedGrades  # Show adjusted DataFrame  print(df) |

The output shows a new column with adjusted grades has been added to the DataFrame.

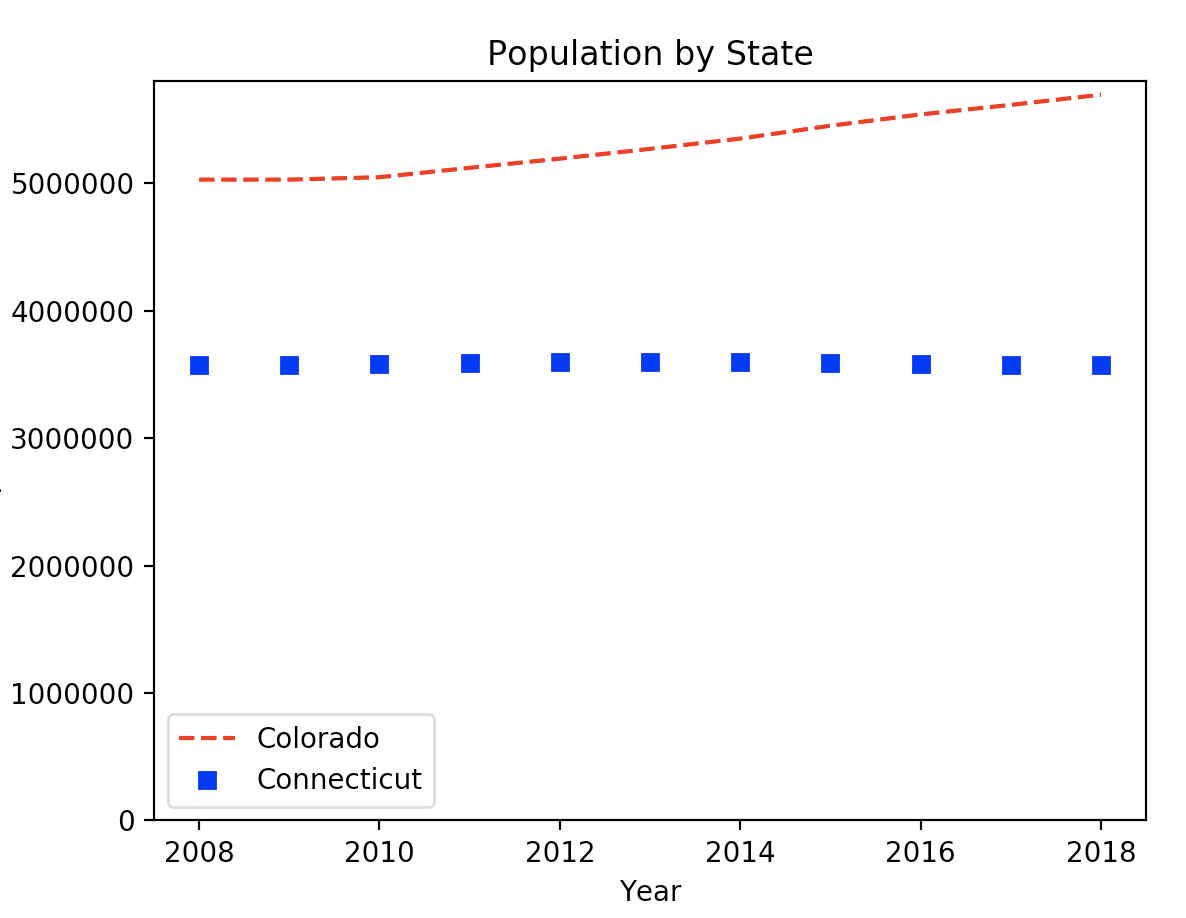


## Visualizations

This section will discuss some basic but essential ways to visualize statistical data. We will then apply these techniques towards exploratory data analysis to help understand the potential and limitations of a data set. You may want to refer back to this document – if you do be sure to use the table of contents and side bar navigation to find the resources that you need from it.

Example 4: Line Graphs

This example demonstrates how to plot two different lines that show changes to population for Connecticut and Colorado. Notice how the legend labels, titles, axis labels and line styles are set – you may want to refer back to this example to see how these features are implemented. This is the output:



Here is the code:

|  |
| --- |
| **import** numpy **as** np **import** matplotlib.pyplot **as** plt  years = [2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018] colorado = [5029196,5029316,5048281,5121771,5193721,5270482,5351218,5452107,  5540921,5615902,5695564] connecticut = [3574097,3574147,3579125,3588023,3594395,3594915,3594783,3587509,  3578674,3573880,3572665]  *# red dashes, blue squares and green triangles* plt.plot(years, colorado, **"--"**, color=**'red'**, label=**"Colorado"**) plt.plot(years,connecticut, **"s"**, color=**'blue'**, label=**"Connecticut"**) *# legend # https://matplotlib.org/users/legend\_guide.html* plt.ylim(ymin=0) *# Set's y axis start to 0.* plt.legend(loc=0) plt.xlabel(**"Year"**) plt.ylabel(**"Population"**) plt.title(**'Population by State'**)  plt.show() |

Exercise 4 (2 marks)

See if you can figure out how to adjust your code in Example 4 to include a third line for Delaware. The population Delaware between 2008 and 2018 is included in this array.

delaware = [897934,897934,899595,907316,915188,923638,932596,941413,949216,  
 957078,967171]

You can make the line a straight line by not including a style parameter. “--" and “s” were used as style parameters previously. Try not including a style parameter to draw a solid line which is the default. Or if you prefer to add line formatting, you can look up ***linestyle*** at <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.lines.Line2D.html#matplotlib.lines.Line2D>

Figure out how to make the legend appear at the center left of the graph. See <https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.legend.html> to look up ***location code***.

Show your revised program here:

|  |
| --- |
| import numpy as np import matplotlib.pyplot as plt  years = [2008,2009,2010,2011,2012,2013,2014,2015,2016,2017,2018] colorado = [5029196,5029316,5048281,5121771,5193721,5270482,5351218,5452107,  5540921,5615902,5695564] connecticut = [3574097,3574147,3579125,3588023,3594395,3594915,3594783,3587509,  3578674,3573880,3572665] delaware = [897934,897934,899595,907316,915188,923638,932596,941413,949216,  957078,967171]   # red dashes, blue squares and green triangles plt.plot(years, colorado, "--", color='red', label="Colorado") plt.plot(years,connecticut, "s", color='blue', label="Connecticut") plt.plot(years, delaware, color='green', label="Delware") # legend # https://matplotlib.org/users/legend\_guide.html plt.ylim(ymin=0) # Set's y axis start to 0. plt.legend(loc=0) plt.xlabel("Year") plt.ylabel("Population") plt.title('Population by State') plt.show() |

Show a screenshot of your chart with three lines in it here:

|  |
| --- |
|  |

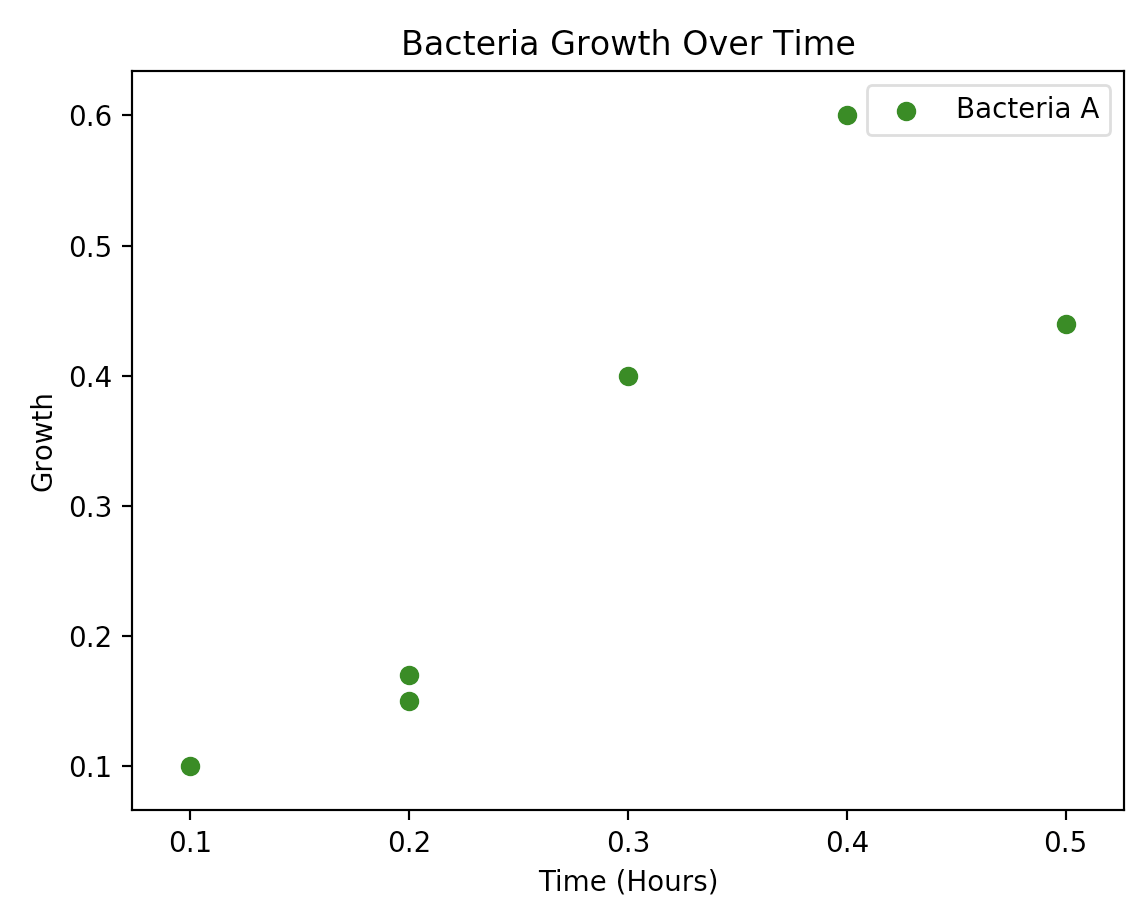
## Scatter Plots

Scatter plots show how one variable is correlated to another for multiple samples.

Example 5: Simple Scatter Plot

Figure 3 shows a simplified scatter plot which illustrates how the level of bacteria A growth is related to time in days for six different samples.

Figure 3: Bacteria A growth over time

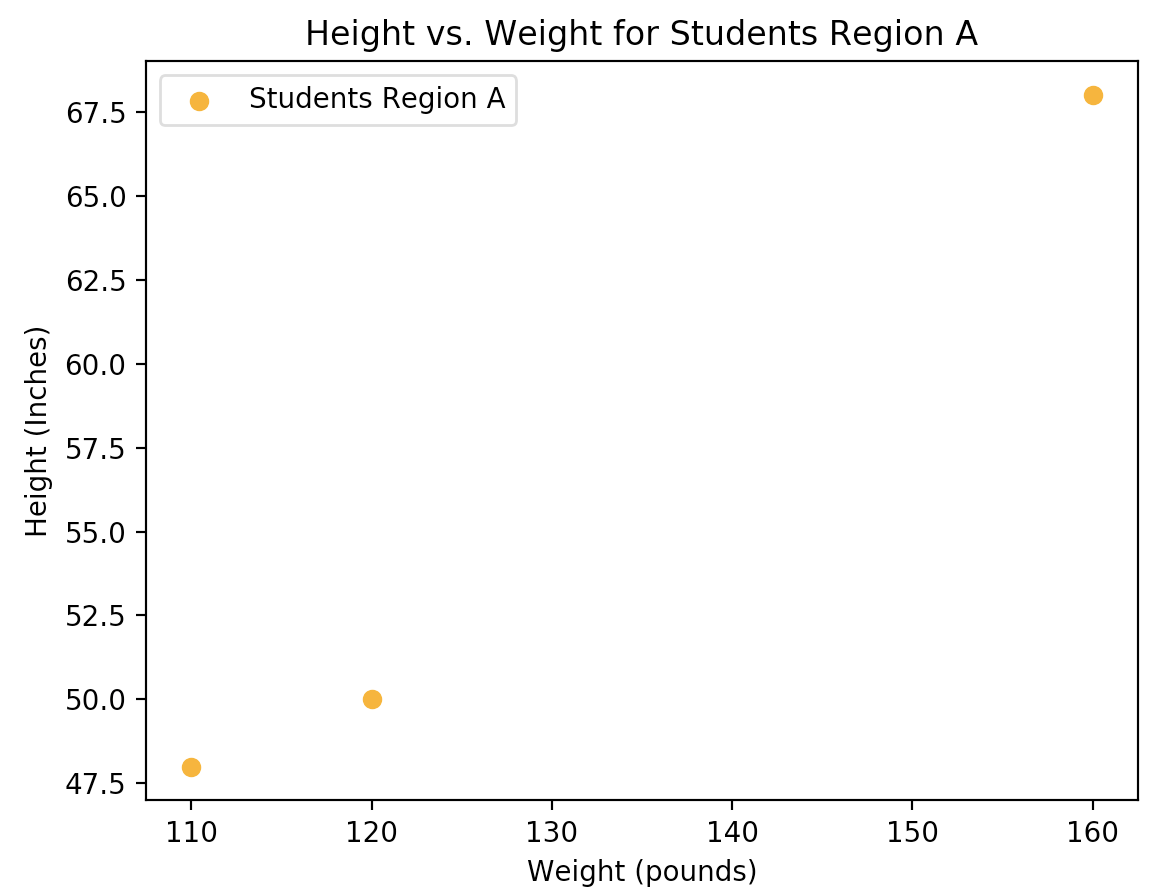


This is the code that is needed to draw the scatter plot. Notice how the x and y values are supplied as arrays to the scatter() function to provide data for the graph. Also notice how the axis, title and legend items are labelled.

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  *# Plot scatter of x and y coordinates.* time\_X = [0.1, 0.2, 0.3, 0.4, 0.5, 0.2] growth\_Y = [0.1, 0.15, 0.4, 0.6, 0.44, 0.17] plt.scatter(time\_X, growth\_Y, color=**'green'**, label=**'Bacteria A'**)  *# Add a legend, axis labels, and title.* plt.legend() plt.xlabel(**"Time (Hours)"**) plt.ylabel(**"Growth"**) plt.title(**'Bacteria Growth Over Time'**)  plt.show() |

Exercise 5 (2 marks)

Draw a scatter plot that looks exactly like the following one.



These are the data arrays:

pounds = [120, 110, 160]  
inches = [50, 48, 68]

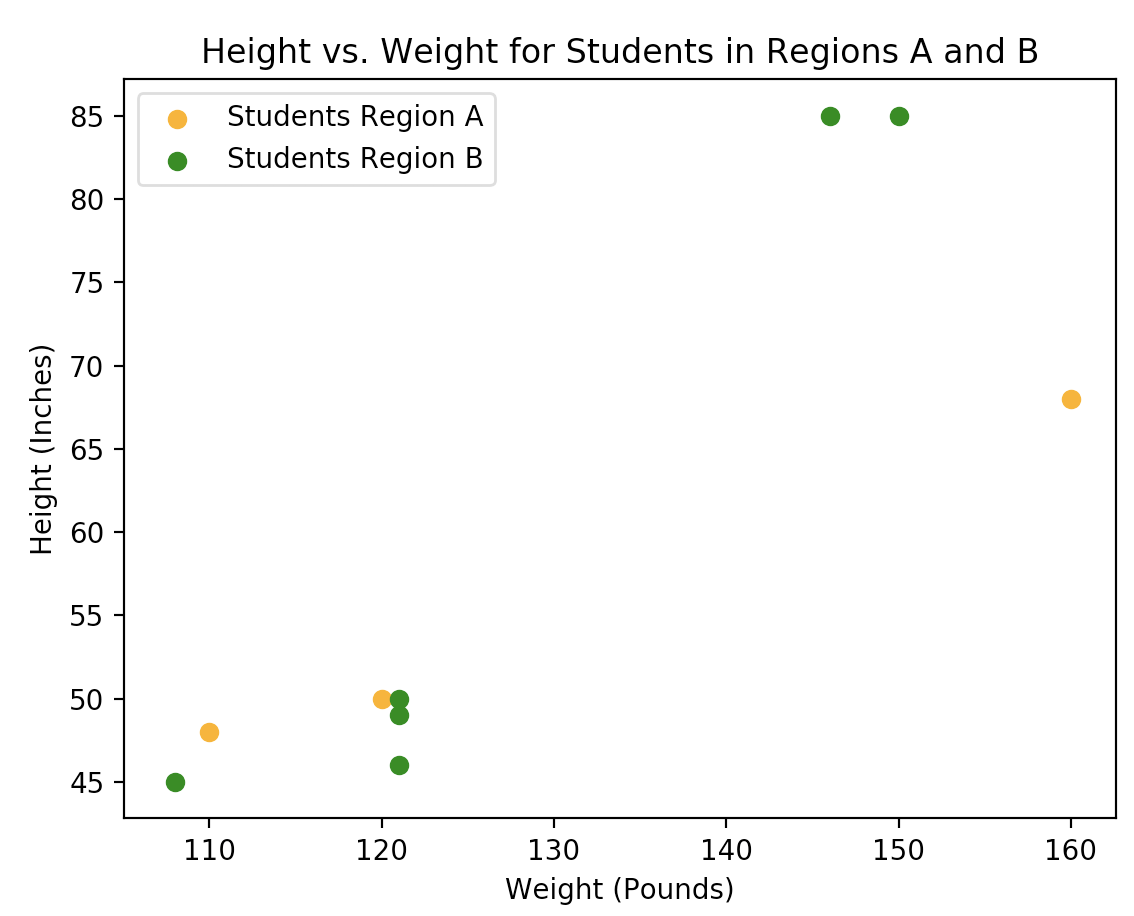
Show your program here:

|  |
| --- |
| import matplotlib.pyplot as plt  # Plot scatter of x and y coordinates. pounds = [120, 110, 160] inches = [50, 48, 68] plt.scatter(pounds, inches, color='yellow', label='Students Region A')  # Add a legend, axis labels, and title. plt.legend() plt.xlabel("Weight (Pounds)") plt.ylabel("Height (Inches)") plt.title('Heights vs. Weight for Students of Region A')  plt.show() |

Exercise 6 (2 marks)

Starting with your solution from Exercise 5, add in code to plot a second scatter group as shown in Figure 4. Keep your new code above the plt.show() instruction.

Figure 4: Comparing Height and Weight Correlations of Two Sample Groups



These are the data sets:

poundsA = [120, 110, 160]  
inchesA = [50, 48, 68]

poundsB = [121, 108, 150, 121, 121, 146]  
inchesB = [49, 45, 85, 46, 50, 85]

Show your program here:

|  |
| --- |
|  |

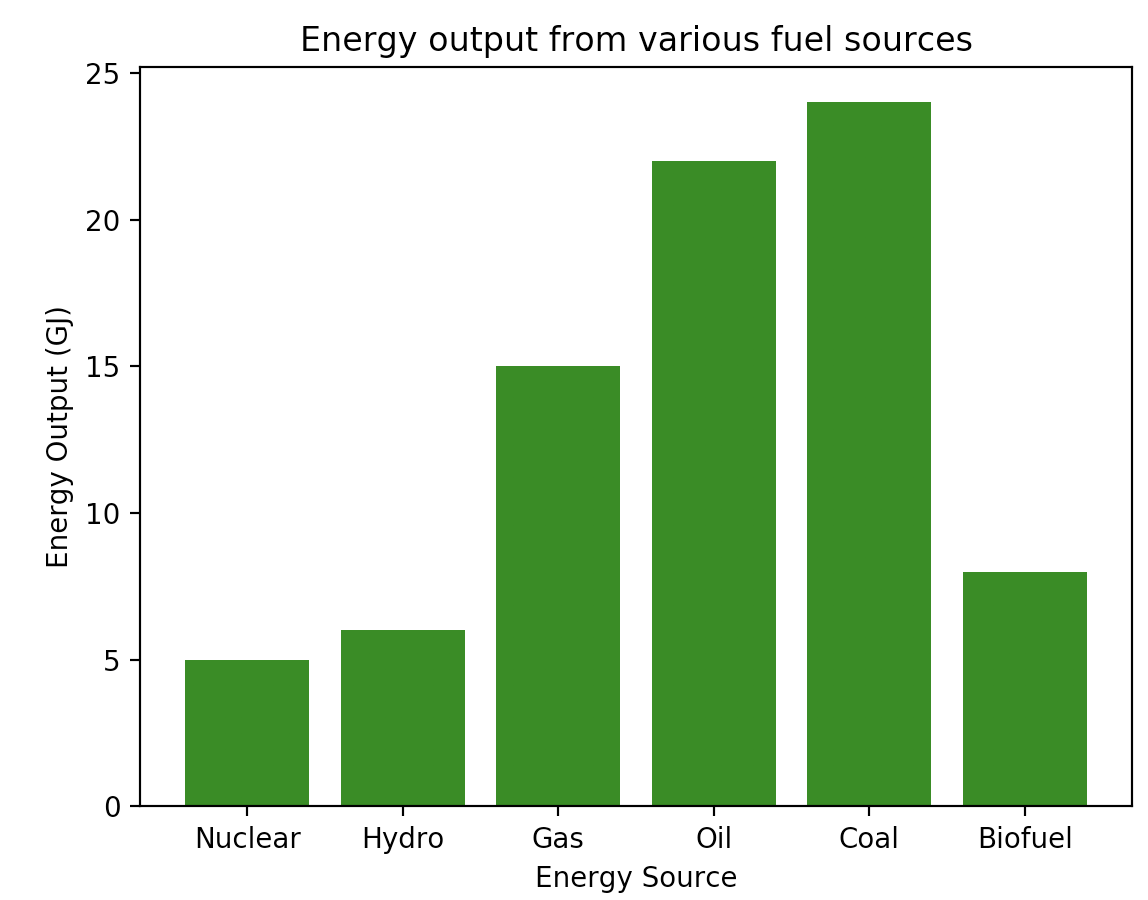
## Bar Plots

Bar plot documentation can be viewed here:

<https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.bar.html>

Example 6: Bar Plots

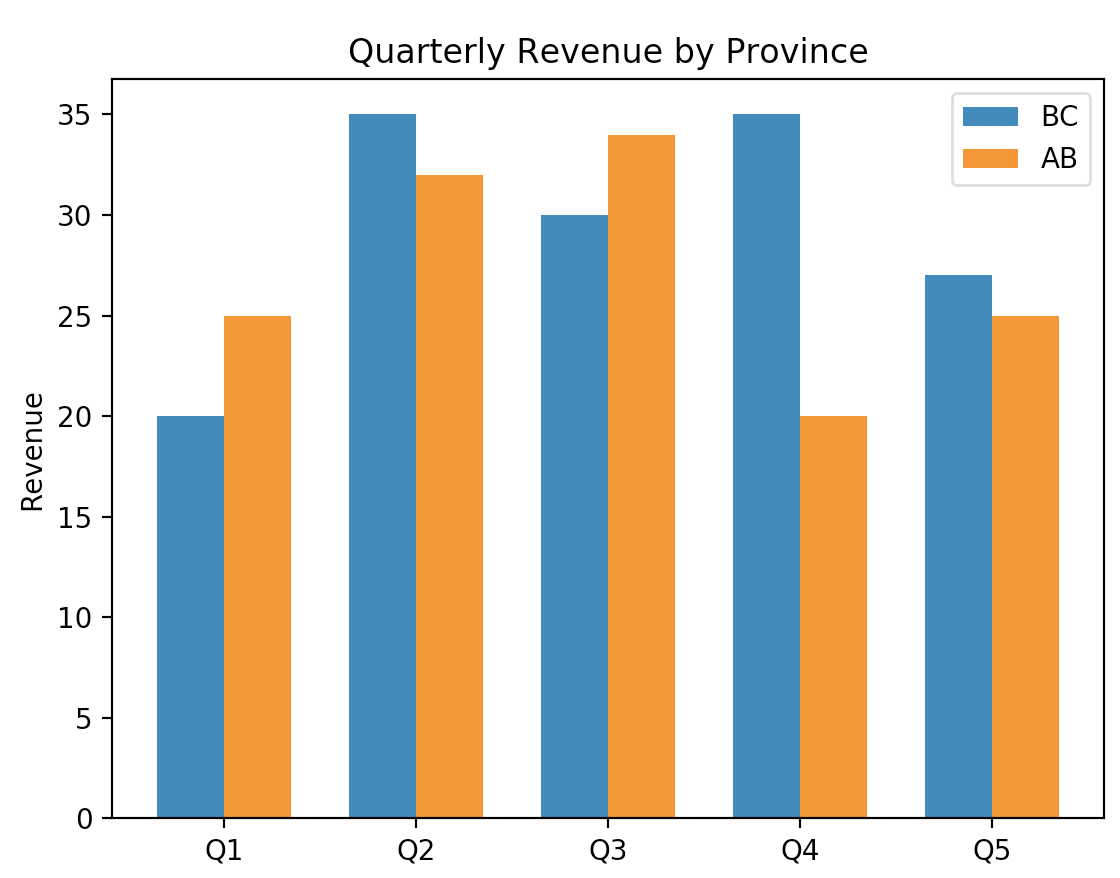
This example shows how to create a basic bar plot.



|  |
| --- |
| **import** matplotlib.pyplot **as** plt  x = [**'Nuclear'**, **'Hydro'**, **'Gas'**, **'Oil'**, **'Coal'**, **'Biofuel'**] energy = [5, 6, 15, 22, 24, 8]  plt.bar(x, energy, color=**'green'**) plt.xlabel(**"Energy Source"**) plt.ylabel(**"Energy Output (GJ)"**) plt.title(**"Energy output from various fuel sources"**)  plt.xticks(x, x) plt.show() |

## Grouped Bar Plots

Example 7: Grouped Bar Plots



|  |
| --- |
| **import** matplotlib.pyplot **as** plt **import** numpy **as** np  NUM\_MEANS = 5 NUM\_GROUPS = 2 bc\_means = [20, 35, 30, 35, 27] alberta\_means = [25, 32, 34, 20, 25]  *# This generates indices from 0 to 4 in a format that is accepted for # plotting bar charts.* ind = np.arange(NUM\_MEANS) print(ind) width = 0.35 plt.bar(ind, bc\_means, width, label=**'BC'**) plt.bar(ind + width, alberta\_means, width, label=**'AB'**)  plt.ylabel(**'Revenue'**) plt.title(**'Quarterly Revenue by Province'**)  plt.xticks(ind + width / NUM\_GROUPS, (**'Q1'**, **'Q2'**, **'Q3'**, **'Q4'**, **'Q5'**)) plt.legend(loc=**'best'**) plt.show() |

Exercise 7 (2 marks)

Modify the code in Example 7 to show groupings for BC, Alberta and Saskatchewan but for the first four quarters only. Customize the color of the bars. You may need to reduce the width to make the bars fit together.

saskatchewan\_means = [18, 28, 32, 24, 31]

Show your code here:

|  |
| --- |
| import matplotlib.pyplot as plt import numpy as np  NUM\_MEANS = 5 NUM\_GROUPS = 3 bc\_means = [20, 35, 30, 35, 27] alberta\_means = [25, 32, 34, 20, 25] saskatchewan\_means = [18, 28, 32, 24, 31]  # This generates indices from 0 to 4 in a format that is accepted for # plotting bar charts. ind = np.arange(NUM\_MEANS) print(ind) width = 0.10 plt.bar(ind, bc\_means, width, label='BC', color="magenta") plt.bar(ind + width, alberta\_means, width, label='AB', color="purple") plt.bar(ind + width, saskatchewan\_means, width, label="SK", color="black")  plt.ylabel('Revenue') plt.title('Quarterly Revenue by Province')  plt.xticks(ind + width / NUM\_GROUPS, ('Q1', 'Q2', 'Q3', 'Q4')) plt.legend(loc='best') plt.show() |

Show a screenshot of your graph here:

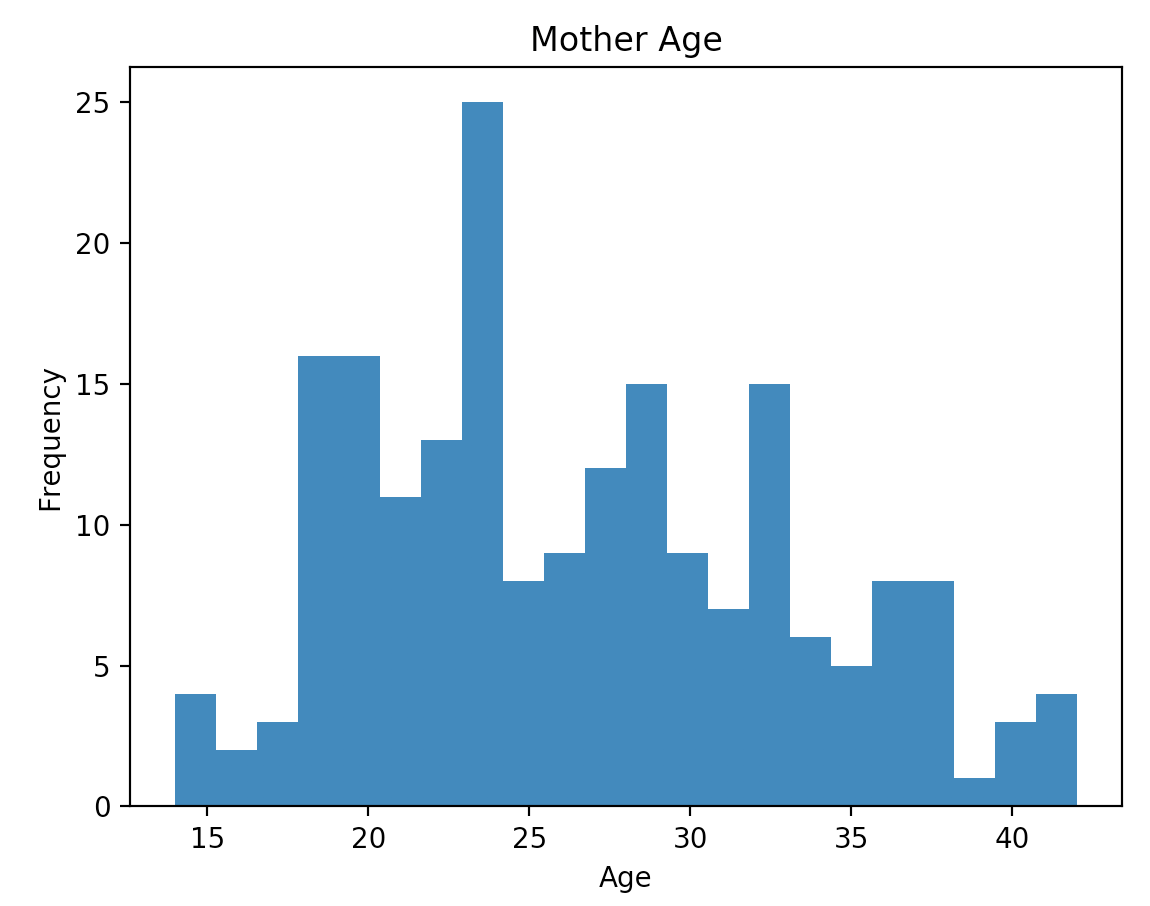
|  |
| --- |
|  |

## Frequency Histograms

Having tables with precise frequency and ranges is helpful. However, many numbers in a summary report can be overwhelming so histograms offer a quick but effective overview of frequency, range and distribution.

Example 8: Histogram Introduction

When examining a histogram of the mother age we can observe a distribution that looks a bit like a normal distribution with a peak around 24 years of age. Some other spikes may also be significant.



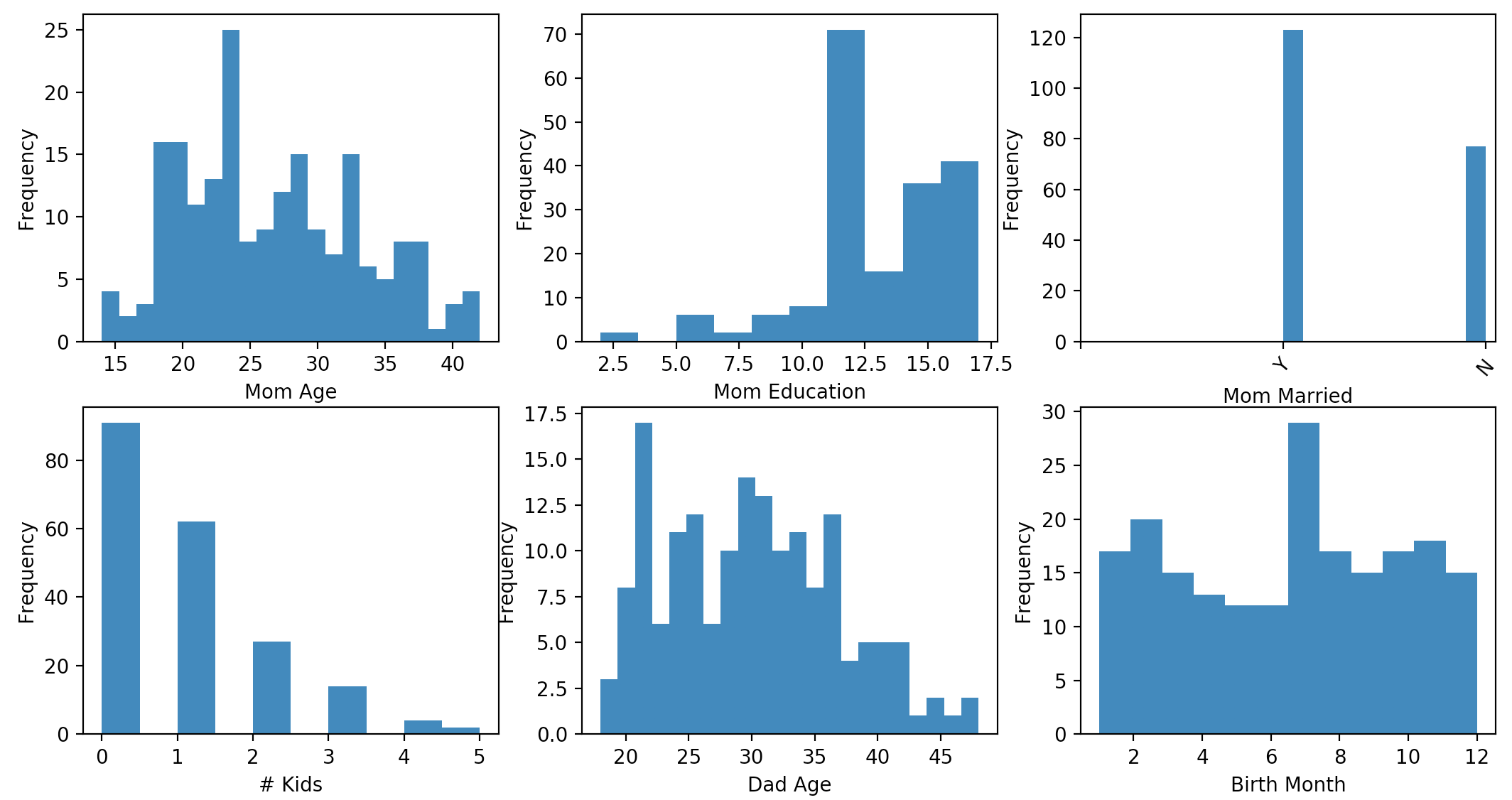
Here is the code used to generate the histogram for the Mother’s age.

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  # Import data into a DataFrame.  path = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/babysamp-98.txt"  df = pd.read\_csv(path, skiprows=1,                     sep='\t',                     names=('MomAge', 'DadAge', 'MomEduc', 'MomMarital', 'numlive',                            "dobmm", 'gestation', 'sex', 'weight', 'prenatalstart',                            'orig.id', 'preemie'))  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  plt.hist(df["MomAge"], bins=22)  plt.xlabel("Age")  plt.ylabel("Frequency")  plt.title('Mother Age')  plt.show() |

## Displaying Multiple Graphs

During exploratory data analysis (EDA) and reporting you will often need to present summaries about many variables. You can adjust graphics output so more than one chart appears in the same row (See Figure 5).

Figure 5: Plotting Multiple Charts in One Row



Example 9: Drawing Multiple Graphs on One Line

This example shows how to output multiple graphs in the same row-column layout as Figure 5. Placing the following line before instructions to draw all graphs allows you to control the size of each of the graphs. I have not perfected the sizing so I use the width and height parameters in the *figsize* attribute as relative measures and experiment with them until getting the proper size.

*# This line allows us to set the figure size supposedly in inches.*

*# When rendered in the IDE the output often does not translate to inches.*

plt.subplots(nrows=2, ncols=3, figsize=(3,4))

Placing a line like this one above each chart sets the position of the chart in the matrix. The first two parameters represent the number of rows and columns of the output. The final number is the position in the matrix where charts are rendered in a clock-wise manner. I find the syntax to be a bit odd but it works. Each time you plot a new chart in the series you state the number of rows and columns in the series as well as the position of the current plot within it.

plt.subplot(2, 3, 1) *# Specfies total rows, columns and image #*

*# where images are drawn clockwise.*

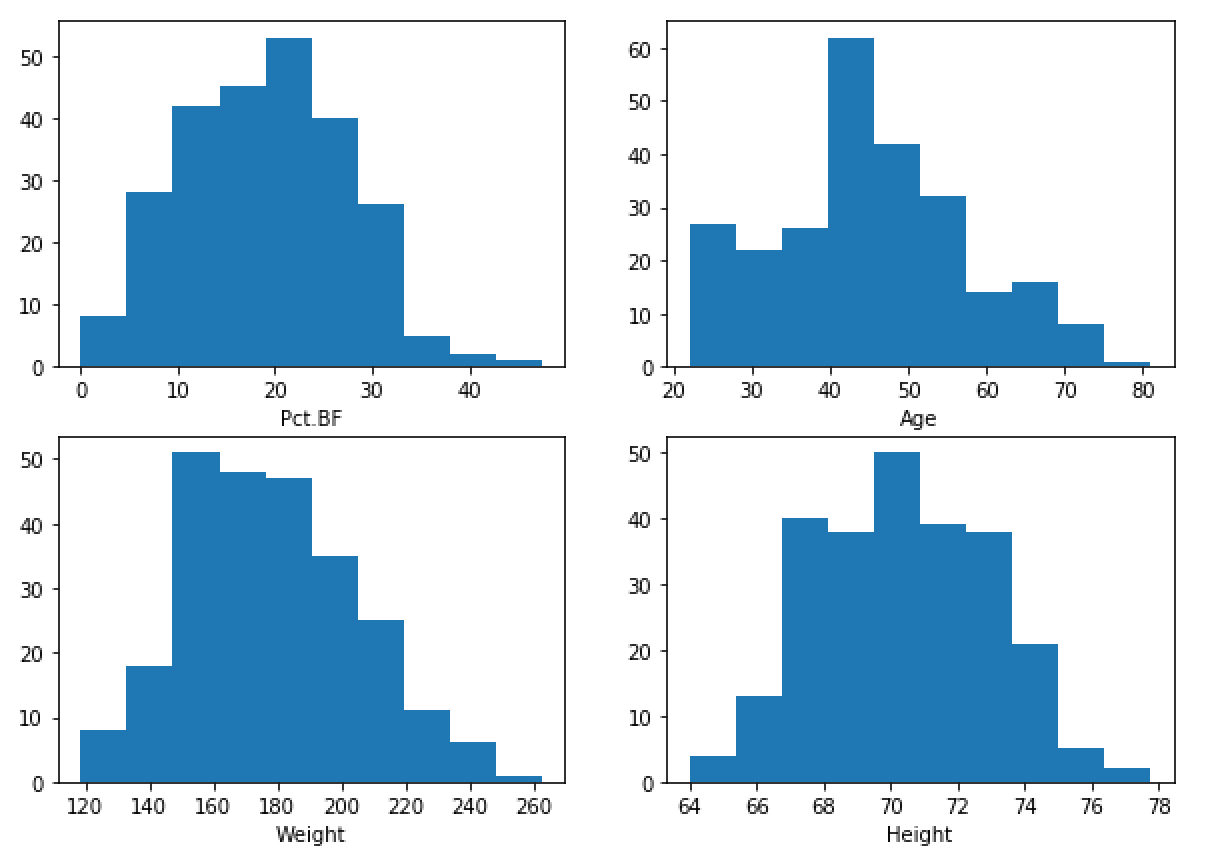
|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  # Import data into a DataFrame.  path = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/babysamp-98.txt"  df = pd.read\_csv(path, skiprows=1,                     sep='\t',                     names=('MomAge', 'DadAge', 'MomEduc', 'MomMarital', 'numlive',                            "dobmm", 'gestation', 'sex', 'weight', 'prenatalstart',                            'orig.id', 'preemie'))    # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  # This line allows us to set the figure size supposedly in inches.  # When rendered in the IDE the output often does not translate to inches.  plt.subplots(nrows=2, ncols=3,  figsize=(14,7))  plt.subplot(2, 3, 1) # Specfies total rows, columns and image #                       # where images are drawn clockwise.  plt.hist(df["MomAge"], bins=22)  plt.xlabel("Mom Age")  plt.ylabel("Frequency")  #plt.title('Mother Age')  plt.subplot(2, 3, 2)  plt.hist(df["MomEduc"], bins=10)  plt.xlabel("Mom Education")  plt.ylabel("Frequency")  # 1 is married  # 2 is unmarried  plt.subplot(2, 3, 3)  plt.hist(df["MomMarital"], bins=10)  t11 = ['', 'Y', 'N']  plt.xticks(range(len(t11)), t11,  rotation=50)  #plt.xticks(['Mar', '2'], rotation=50)  plt.xlabel("Mom Married")  plt.ylabel("Frequency")  # 1 is married. 2 is unmarried.  plt.subplot(2, 3, 4)  plt.hist(df["numlive"], bins=10)  plt.xlabel("# Kids")  plt.ylabel("Frequency")  plt.subplot(2, 3, 5) # of rows, # of columns, # plots.  plt.hist(df["DadAge"], bins=22)  plt.xlabel("Dad Age")  plt.ylabel("Frequency")  plt.subplot(2, 3, 6)  plt.hist(df["dobmm"], bins=12)  plt.xlabel("Birth Month")  plt.ylabel("Frequency")  plt.show() |

Exercise 8 (2 marks)

Starting with the following code;

|  |
| --- |
| ﻿import pandas as pd  import matplotlib.pyplot as plt  # Import data into a DataFrame.  path = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/bodyfat.txt"  df = pd.read\_csv(path, skiprows=1,                     sep='\t',                     names=('Density', 'Pct.BF', 'Age', 'Weight', 'Height',                             'Neck', 'Chest', 'Abdomen', 'Waist', 'Hip', 'Thigh',                            'Ankle', 'Knee', 'Bicep', 'Forearm', 'Wrist')) |

Make the following histograms by drawing 2 per row with 10 bins each.



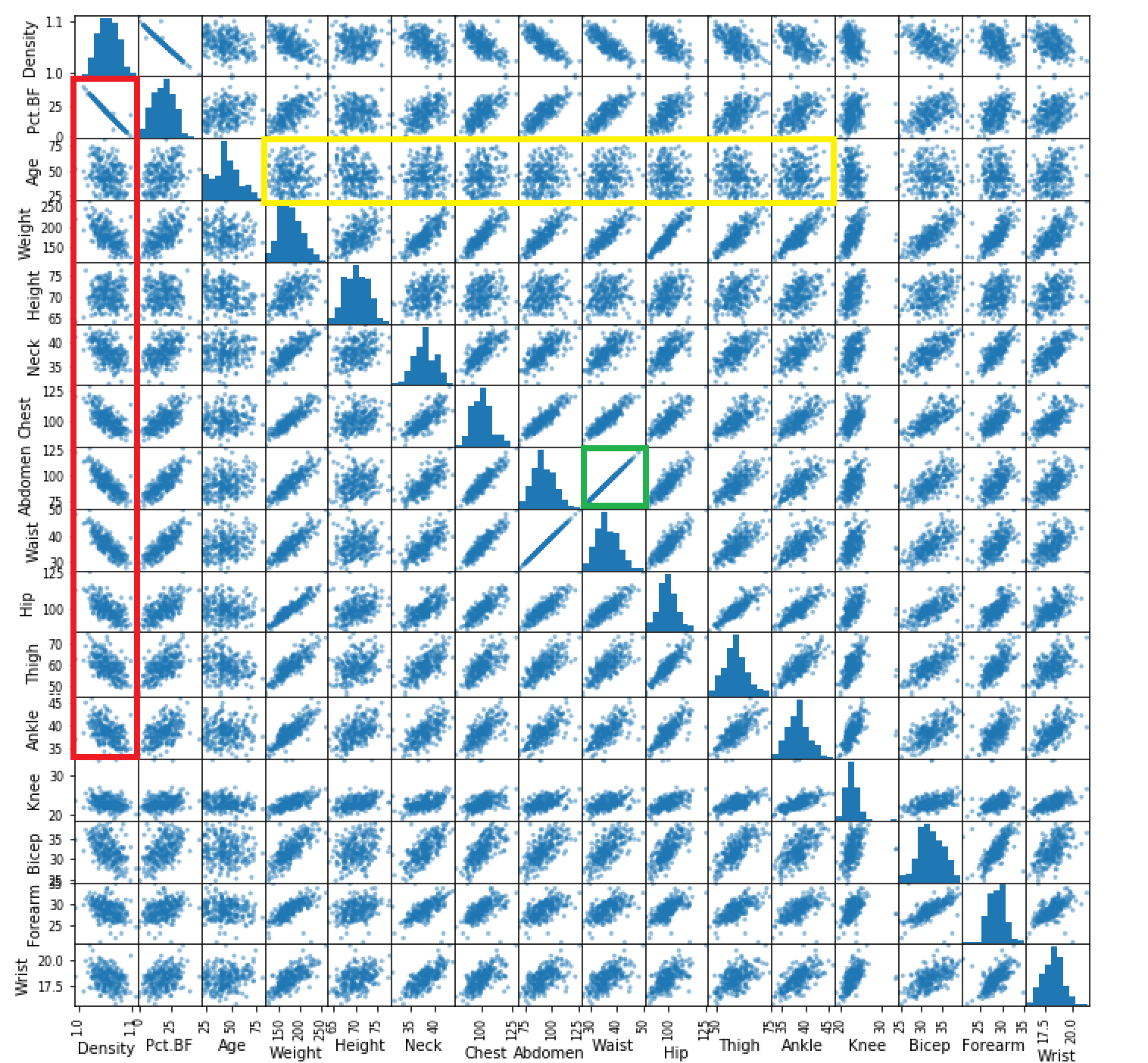
Show your code here:

|  |
| --- |
| import pandas as pd import matplotlib.pyplot as plt  # Import data into a DataFrame.  path = "../dataset/bodyfat.txt" df = pd.read\_csv(path, skiprows=1,  sep='\t',  names=('Density', 'Pct.BF', 'Age', 'Weight', 'Height',  'Neck', 'Chest', 'Abdomen', 'Waist', 'Hip', 'Thigh',  'Ankle', 'Knee', 'Bicep', 'Forearm', 'Wrist'))  pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000) plt.subplots(nrows=2, ncols=2, figsize=(14,7)) plt.subplot(2, 2, 1) plt.hist(df['Pct.BF'], bins=10) plt.xlabel('Pct.BF') plt.ylabel('')  plt.subplot(2, 2, 2) plt.hist(df['Age'], bins=10) plt.xlabel('Age') plt.ylabel('')  plt.subplot(2, 2, 3) plt.hist(df['Weight'], bins=10) plt.xlabel('Weight') plt.ylabel('')  plt.subplot(2, 2, 4) plt.hist(df['Height'], bins=10) plt.xlabel('Height') plt.ylabel('')  plt.show() |

## Scatter Matrix

Scatter matrices are helpful for showing correlation between variable pairs. The red outline in Figure 6 indicates very little correlation. The green outline suggests very high correlation between waist and abdomen. Density seems to be negatively correlated to many of the variables as outlined in red. Age does not seem correlated with many weights and measurements as outlined in yellow. There does appear to be a strong correlation between abdomen size and waist as outlined in green.

Figure 6: Scatter Matrix



Here is the code:

|  |
| --- |
| ﻿import pandas as pd  from pandas.plotting import scatter\_matrix  # Import data into a DataFrame.  path = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/bodyfat.txt"  df = pd.read\_csv(path, skiprows=1,                     sep='\t',                     names=('Density', 'Pct.BF', 'Age',   'Weight', 'Height',                            'Neck', 'Chest','Abdomen', 'Waist', 'Hip', 'Thigh',                            'Ankle', 'Knee', 'Bicep',                            'Forearm',                            'Wrist'))  scatter\_matrix(df, figsize=(12,12)) |

Exercise 9 (3 marks)`

It is not possible to run a scatter matrix non-numeric data and some categorical variables. To isolate the numeric data, use the sample from babysamp-98.txt file to create a sub-DataFrame with MomAge, gestation and weight columns only. Then draw a scatter matrix with this subset. Show your program here:

|  |
| --- |
| import pandas as pd from pandas.plotting import scatter\_matrix  # Import data into a DataFrame. path = "../dataset/babysamp-98.txt" df = pd.read\_csv(path, skiprows=1,  sep='\t',  names=('MomAge',  'gestation',  'weight'))  scatter\_matrix(df) |

Show a screenshot of your scatter matrix here:

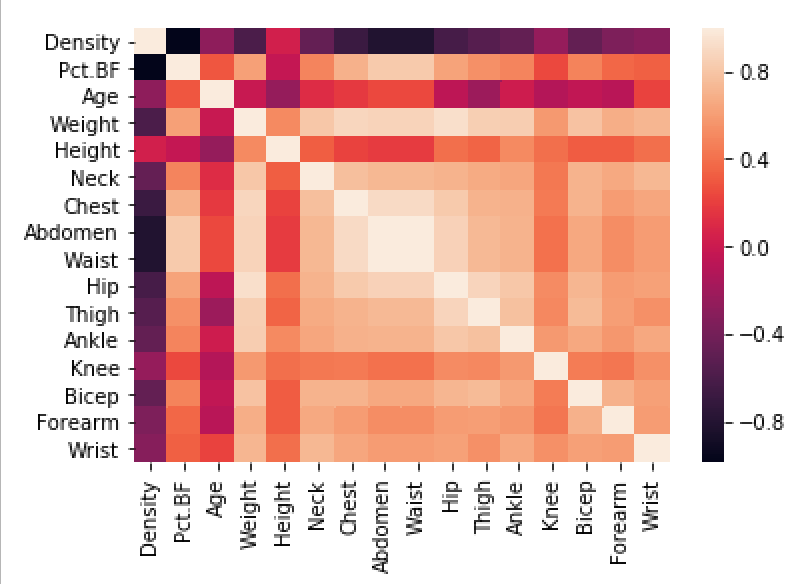
|  |
| --- |
| This is not working in Python 3.8. I tried the sample code example for scatter matrix which didn’t output anything but ran with no error.  Error message: |

Are there any visible trends (or lack of trends) that appear in terms of correlation?

|  |
| --- |
| Cannot answer because the visualization isn’t working. |

## Correlation Heatmap

Another way to show correlation is with the correlation heat map. The correlation heatmap offers a more condensed view of correlation between variables but it does not show the distribution of correlation.



Example 10: Correlation Heatmap

This example demonstrates how to draw a correlation heatmap with the body fat sample.

|  |
| --- |
| ﻿import numpy as np  import seaborn as sns  import matplotlib.pylab as plt  from string import ascii\_letters  import numpy as np  import pandas as pd  import seaborn as sns  # Import data into a DataFrame.  path = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/bodyfat.txt"  df = pd.read\_csv(path, skiprows=1,                     sep='\t',                     names=('Density', 'Pct.BF', 'Age',   'Weight', 'Height',                            'Neck', 'Chest','Abdomen', 'Waist', 'Hip', 'Thigh',                            'Ankle', 'Knee', 'Bicep',                            'Forearm',                            'Wrist'))  # Compute the correlation matrix  corr = df.corr()  # plot the heatmap  sns.heatmap(corr,          xticklabels=corr.columns,          yticklabels=corr.columns) |

Exercise 10 (3 marks)

As mentioned previously, it is not possible to run a scatter matrix non-numeric data and some categorical variables. To isolate the numeric data, use the sample from babysamp-98.txt file to create a sub-DataFrame with MomAge, gestation and weight columns only. Then draw a correlation heatmap with this subset. Show your program here:

|  |
| --- |
| import numpy as np import seaborn as sns import matplotlib.pylab as plt from string import ascii\_letters import numpy as np import pandas as pd import seaborn as sns  # Import data into a DataFrame. path = "../dataset/babysamp-98.txt" df = pd.read\_csv(path, skiprows=1,  sep='\t',  names=('MomAge','gestation','weight')) # Compute the correlation matrix corr = df.corr()  # plot the heatmap sns.heatmap(corr,  xticklabels=corr.columns,  yticklabels=corr.columns) plt.show() |

Show a screenshot of your correlation heatmap here:

|  |
| --- |
|  |

## SQLAlchemy

This section assumes that you know how to perform aggregate queries using SQL.

SqlAlchemy provides an interface that allows you to query DataFrames with SQL. SqlAlchemy can also be used to create databases and to perform queries on databases.

### Filtered Queries

With SqlAlchemy standard query syntax can be used to query a DataFrame:

SELECT \* FROM brazilForest WHERE state = 'Rio' OR state='Sao Paulo' ORDER BY date

### Aggregate Queries

If you are used to writing SQL you may find performing SQL-based aggregate queries on a DataFrame is much easier with SqlAlchemy when compared to using the built-in DataFrame summary functions. If you are familiar with aggregate queries you will know it is only possible to show column values that are contained in the GROUP BY clause as well as summarized values when writing aggregate queries.

SELECT state as 'State', Round(SUM(number),2) AS 'Burn Units' FROM brazilForest

GROUP BY state HAVING SUM(number)>30000 Order by SUM(number)

### Creating Database Table from a DataFrame

The code needed to create a database table from a DataFrame is very condensed. A connection must first be created. Then the table is created and given data with the DataFrame instruction *to\_sql()*.

# Create the database at the specified path.

DB\_FILE = 'forestFire.db'

engine = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False)

connection = engine.connect()

# Store data in database in a table named 'brazilForest'.

df.to\_sql(name='table\_name', con=connection, if\_exists='replace', index=False)

Example 11: Creating and Querying a Database

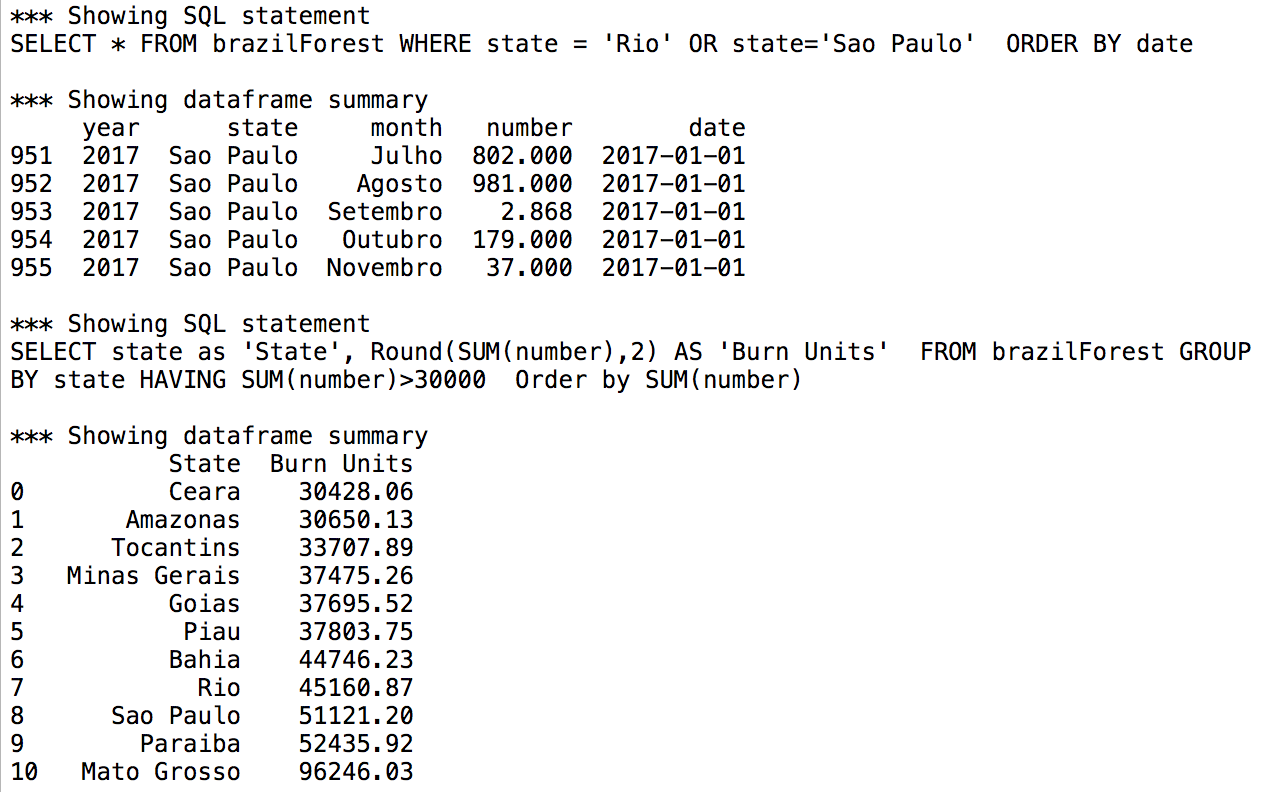
This example demonstrates how to create a Sqlite database and insert data from a DataFrame into the database. First, you will need to add in your file path and the name of your file. Then you can run this code and explore the data.

For this example, we are creating a database named forestFire.db and a table called brazilForest. Be sure to modify your query as needed.

You will also notice that SQL query is executed inside a function so the code can easily be re-used without a duplication of effort.

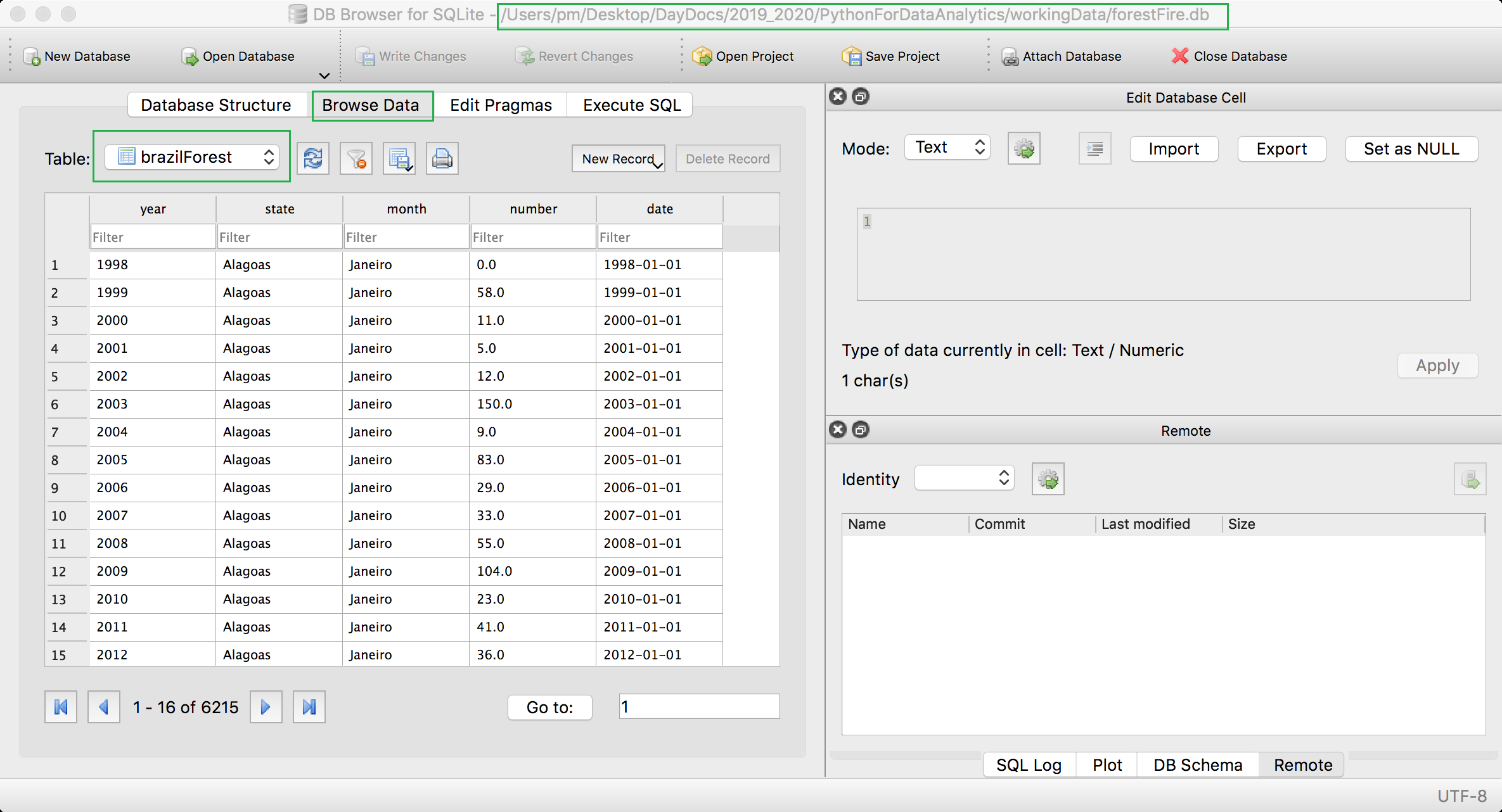
|  |
| --- |
| import pandas as pd  from sqlalchemy import create\_engine  # The data file path and file name need to be configured.  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  CSV\_DATA = "brazil\_forestFires.csv"  # Note this has a comma separator.  df = pd.read\_csv(PATH + CSV\_DATA, skiprows=1, encoding = "ISO-8859-1", sep=',',  names=('year', 'state', 'month', 'number','date', ))  df.describe()  df['state'].value\_counts()  print(df.tail())  # Create the database at the specified path.  DB\_FILE = 'forestFire.db'  engine = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False)  connection = engine.connect()  # Store data in database in a table named 'brazilForest'.  df.to\_sql(name='brazilForest', con=connection, if\_exists='replace', index=False)  # Placed query in this function to enable code re-usuability.  def showQueryResult(sql):  print("\n\*\*\* Showing SQL statement")  print(sql)  # Perform query  subDf = pd.read\_sql(sql, connection)  print("\n\*\*\* Showing dataframe summary")  return subDf  # Get DataFrame contents for 'Rio' and 'Sao Paulo' only.  sql = "SELECT \* FROM " + "brazilForest" \  + " WHERE state = 'Rio' OR state='Sao Paulo' " \  + " ORDER BY date"  print(showQueryResult(sql).tail())  # Get total 'Burn Units' by State where total units > 3000.  # Order the results by the total 'Burn Units'.  sql = "SELECT state as 'State', Round(SUM(number),2) AS 'Burn Units' " + \  " FROM brazilForest GROUP BY state HAVING SUM(number)>30000 " + \  " Order by SUM(number)"  print(showQueryResult(sql)) |

The output demonstrates that we were able to filter our results with SQL by *state* equalling ‘Rio’:



Exercise 11 (4 marks)

To inspect the database, download **DB Brower for Sqlite** from <https://sqlitebrowser.org/>. Then navigate to your file and browse the data.



Using **DB Browser for SQLite**, scroll to the end of the table and show a screenshot of the window here:

|  |
| --- |
|  |

### Creating an In-Memory Database

You can actually create an in-memory database rather than storing the data in a db file.

Example 12: Creating an In-Memory Database

To do this replace these two lines in Example 11.

|  |
| --- |
| DB\_FILE    = 'forestFire.db'  engine     = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False) |

with:

|  |
| --- |
| engine     = create\_engine('sqlite://', echo=False) |

### Reading from a Database

The code needed to read from an existing database is also very condensed. We need our data connection and once we have it we can execute SQL.

engine = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False)

connection = engine.connect()

subdf = pd.read\_sql(sql, connection)

Example 13: Reading from a Database

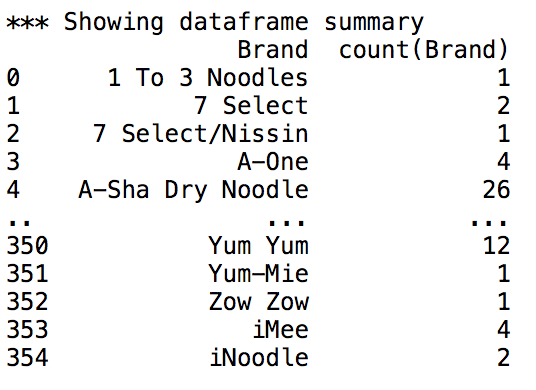
This example demonstrates how to read from a *Review* table within a *ramenReviews.db* database. The content happens to be in a SQLite database but the content could easily be read from MS SQL, MySQL, Oracle, PostGreSQL etc.

|  |
| --- |
| import pandas as pd  from sqlalchemy import create\_engine  # The data file path and file name need to be configured.  PATH = "/Users/pm/Desktop/DayDocs/2019\_2020/PythonForDataAnalytics/workingData/"  # Create the database at the specified path.  DB\_FILE = 'ramenReviews.db'  engine = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False)  connection = engine.connect()  # Show all columns.  pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line.  pd.set\_option('display.width', 1000)  # Placed query in this function to enable code re-usuability.  def showQueryResult(sql):  print("\n\*\*\* Showing SQL statement")  print(sql)  # Perform query  subDf = pd.read\_sql(sql, connection)  print("\n\*\*\* Showing dataframe summary")  return subDf  # Get ramen reviews.  sql = "SELECT \* FROM Review WHERE \"Top Ten\" is not null"  print(showQueryResult(sql).head()) |

The output shows that we have a table with 2579 rows in it.

Exercise 12 (2 marks)

Using SqlAlchemy, count the number of instances per Brand in the ramenReviews.db. Show Brand in your output as well. Your output look like the following:



Show your code here:

|  |
| --- |
| import pandas as pd from sqlalchemy import create\_engine  # The data file path and file name need to be configured. PATH = "../dataset/"  # Create the database at the specified path. DB\_FILE = 'ramenReviews.db'  engine = create\_engine('sqlite:///' + PATH + DB\_FILE, echo=False) connection = engine.connect()  # Show all columns. pd.set\_option('display.max\_columns', None)  # Increase number of columns that display on one line. pd.set\_option('display.width', 1000)  # Placed query in this function to enable code re-usuability. def showQueryResult(sql):  print("\n\*\*\* Showing SQL statement")  print(sql)  # Perform query  subDf = pd.read\_sql(sql, connection)  print("\n\*\*\* Showing dataframe summary")  return subDf  # Get ramen reviews. sql = "SELECT Brand, *count*(Brand) FROM Review GROUP BY Brand" print(showQueryResult(sql).head()) |