String Manipulation and Data Visualizations

Johnny Zhang

String Manipulation

- String Manipulation
  - Cleaning and preparing data
- Finding Initial Position of a String
  - find(): the find function is used to find the first occurrence of the specified value.
  - The find() method return -1 if the value is not found
  - Syntax: string.find(value, start, end)

```
 # Searches for match in an array of characters and starts count at 0.
sentence = "A lazy dog jumped over a log"
print("Starting position: " + str(sentence.find('dog')))

# When item not found, -1 is returned.
sentence = "A lazy dog jumped over a log"
print("Starting position: " + str(sentence.find('cat')))
```

String Slicing (1)

- Slice: span of items taken from a sequence (List, Tuple, Array, & String), known as substring
- To access an individual character in a string:
  - for loop
    - Ex. for character in string:
    - iterate over the whole string (ex. Count the occurrences of a specific character)
  - Use indexing
    - Each character has an index specifying its position in the string, starting at 0 (How about list?)
    - Format: character = myString[i]

```
str1='Hello COMP2454'
indexArr=[]
for index in range(0, len(str1)):
    if str1[index] =='o':
        indexArr.append(index)
print(indexArr)
```

```
[4]
```

```
str1='Hello COMP2454'

for index in range(0, len(str1)):
    print(str(index)+'-->'+str1[index])
```

```
0-->H
1-->e
2-->l
3-->l
4-->o
5--> 
6-->C
7-->O
8-->M
9-->P
10-->2
11-->4
12-->5
13-->4
```

```
str1='Hello COMP2454'
str1[1]
```

```
'e'
```

```
str1='Hello COMP2454'
str1[0]
```

```
'H'
```

String Slicing (2)

- Slicing
  - Slicing format: string[*start* : *end*]
    - Expression will return a string containing a copy of the characters from *start* upto, but not including, *end*
    - If *start* not specified, *0* is used for start index
    - If *end* not specified, *len(string)* is used for end index
  - Slicing expressions can include a step value and negative indexes relative to end of string.

```
fullName      = "Bob Jones"
spacePosition = fullName.find(" ")
startPosition = 0
endPosition   = spacePosition
firstName     = fullName[startPosition:endPosition]
print(firstName)
```

Output?

Exercise 1 & 2
5 Mins

Replacing Strings

- The replace() function offers a nice simple way to exchange old text and with new text.

Example 4: Replacing Strings

This code replaces 'dog' with 'cat' and prints the result.

```
text = "A lazy dog jumped over a log."
newText = text.replace("dog", "cat")
print(newText)
```

The output shows a successful replacement:

```
A lazy cat jumped over a log.
```

Exercise 3 (1 mark)

Figure out how to use the *replace()* function to change the sentence in Example 3 to "A dog jumped over a log". (The answer may be simpler than you think) Show your code here:

split()

- The *split()* function separates a string based on a preferred delimiter. This function is really helpful when reading from a comma separated list of values.

This example shows how all words can be extracted from a sentence and stored in an array.

```
word = "A lazy dog jumped over a log."
sentenceArray = word.split(' ')
print(sentenceArray)

for counter in range(0, len(sentenceArray)):
    print(sentenceArray[counter])
```

The output confirms that sentence has successfully been stored in an array:

```
['A', 'lazy', 'dog', 'jumped', 'over', 'a', 'log.']
A
lazy
dog
jumped
over
a
log.
```

join()

- The *join()* function performs in an opposite manner compared to the *split()* function.
- The *join()* function converts an array back into a single string.
- This function is especially helpful when writing comma separated value content to a file.

This example shows how an array is converted back into one string with a single space between each word.

```
sentenceArray = ['A', 'lazy', 'dog', 'jumped', 'over', 'a', 'log.']
delimiter     = ' '
newSentence   = delimiter.join(sentenceArray)
print(newSentence)
```

The output shows that all elements have been concatenated into one string where all words are separated by an empty space:

```
A lazy dog jumped over a log.
```

Exercise 4 (1 mark)

Modify Example 6 so the string created contains all words in the original array separated by a comma. Show your code here.

Exercise 4 (3Mins)

Scope and Precedence

- Scope refers to the region where a variable is defined.
- A variable only exists in the region where it is defined.
- When two variables have the same name, the variable with the most localized scope is the one that is recognized. This bias to recognize local variables over global variables is called precedence.

```
def displayContent(x): x=a; x->100
    a = 25 # The number is born here.
    print('Inside the function the variable \'a\'=' + str(a))
    print('The parameter x=' + str(x))
    return # Locally declared variables die here when the function exits.

a = 100
displayContent(a)
print("The global value for \'a\'=" + str(a))
```

Exercise 5

Visualization

- Data visualization is the process of converting raw data into easily understood pictures of information that enable fast and effective decisions.
- Three functions of visualization
  - ✓Record: store information
  - ✓Analyze: support reasoning about information
  - ✓Communicate: convey information to other

Matplotlib Error

AttributeError: module 'backend_interagg' has no attribute 'FigureCanvas'. Did you mean: 'FigureCanvasAgg'?

Solution:

import matplotlib

modify the backend configuration

Matplotlib.use('TKAGG')

import matplotlib
matplotlib.use('TkAgg')

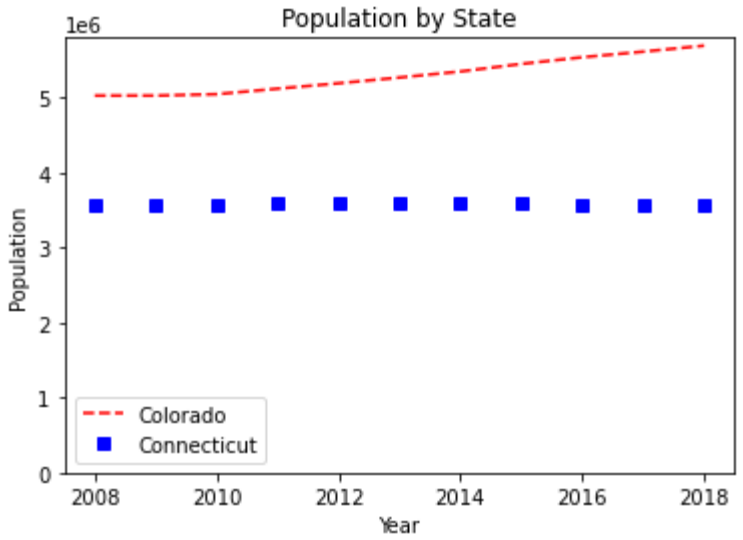
Line Graphs

A line graph connects individual data points with lines. It's usually used to show trends over a continuous variable, most often time.

```
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()
```

Legend(loc=0)?
Legend(loc=1)?
Legend(loc=2)?
Legend(loc=3)?
Legend(loc=4)?

Exercise 6



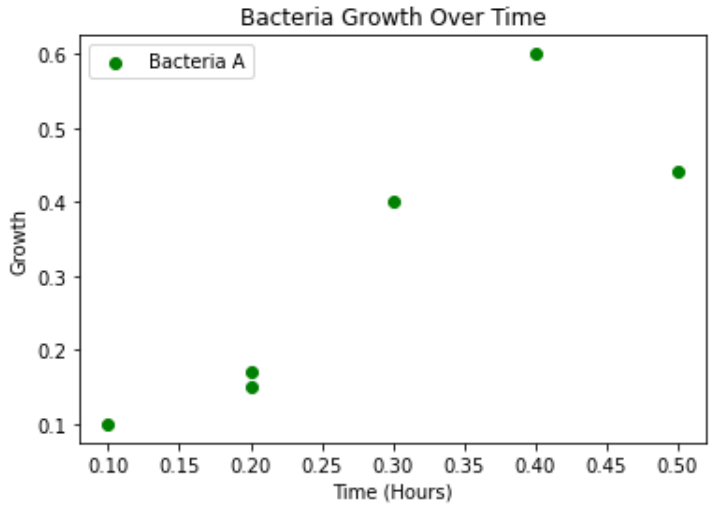
- Trends over time: stock prices, temperature, sales, etc.
- Continuous variables: when the x-axis is numeric or ordered.
- Comparisons: show multiple datasets in the same plot.
- Highlight patterns: rising/falling trends, peaks, or drops.

Scatter Plots

- Scatter plots show how one variable is correlated to another for multiple samples.
- You use scatter plots when you want to visualize the relationship between two numeric variables or show the distribution of individual data points

Exercise 7-8

```
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(loc=0)
plt.xlabel("Time (Hours)")
plt.ylabel("Growth")
plt.title('Bacteria Growth Over Time')
plt.show()
```

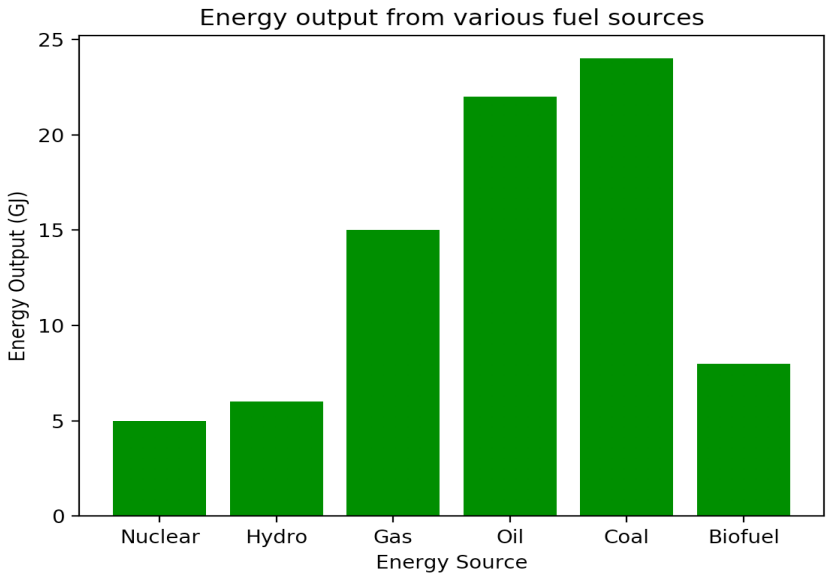


Use case	Scatter Plot Advantage
Correlation	See trends/patterns between two numeric variables
Clustering	Spot natural groups in data
Outliers	Identify unusual data points
Distribution	Visualize spread along x and y
Multivariate	Use size/color to encode extra variables

Bar Plots

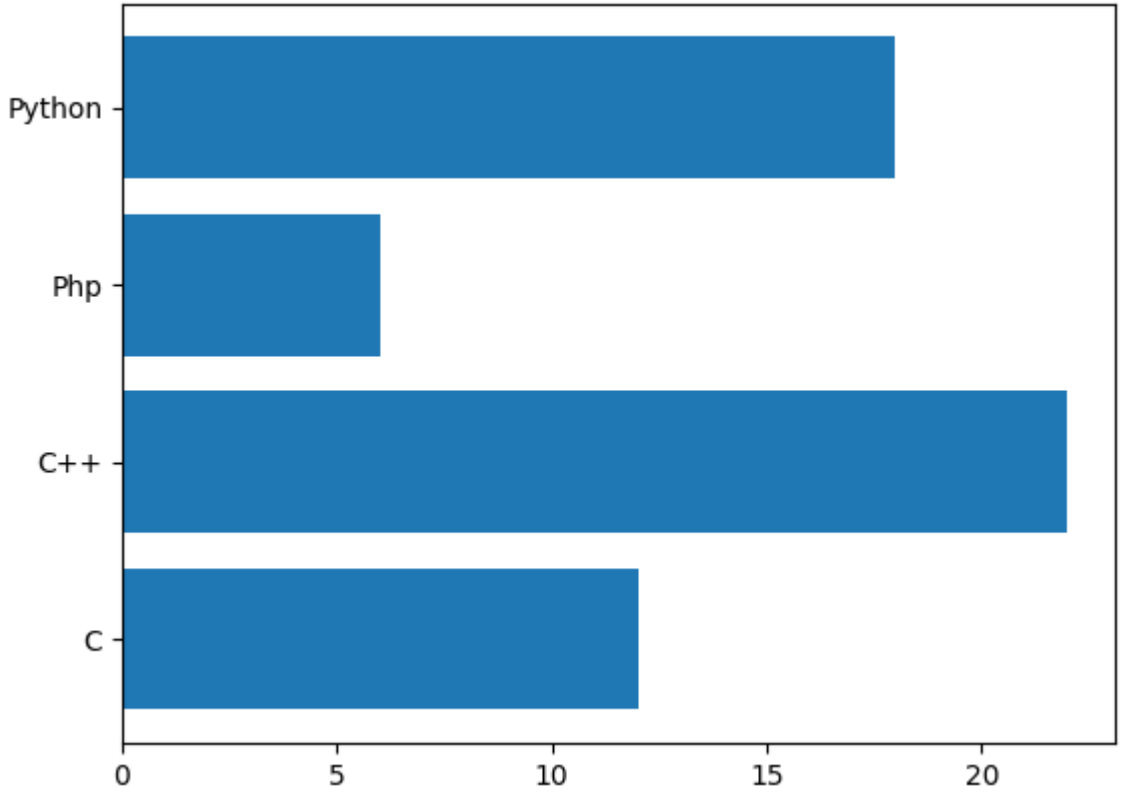
A bar plot is a plot that presents categorical data with rectangular bars with lengths proportional to the values that they represent. A bar plot shows comparisons among discrete categories. One axis of the plot shows the specific categories being compared, and the other axis represents a measured value.

```
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()
```



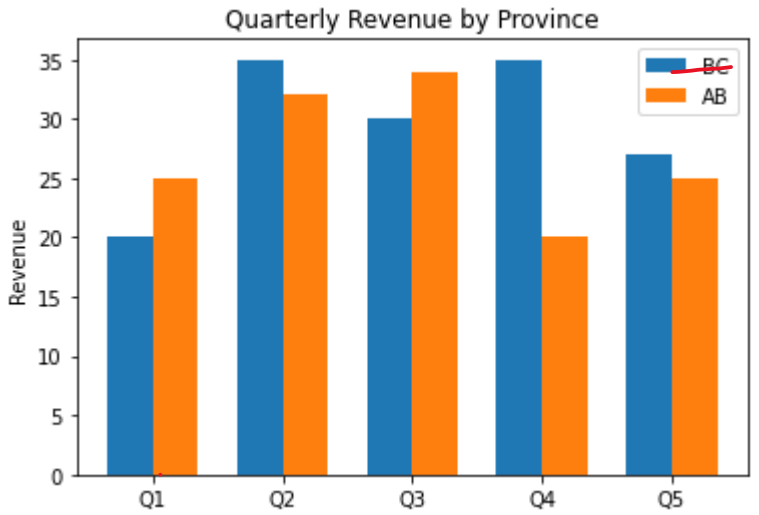
barh(): Make a horizontal bar plot

```
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt

import numpy as np
x = np.array(["C", "C++","Php", "Python"])
y = np.array([12, 22, 6, 18])

plt.barh(x,y)
plt.show()
```

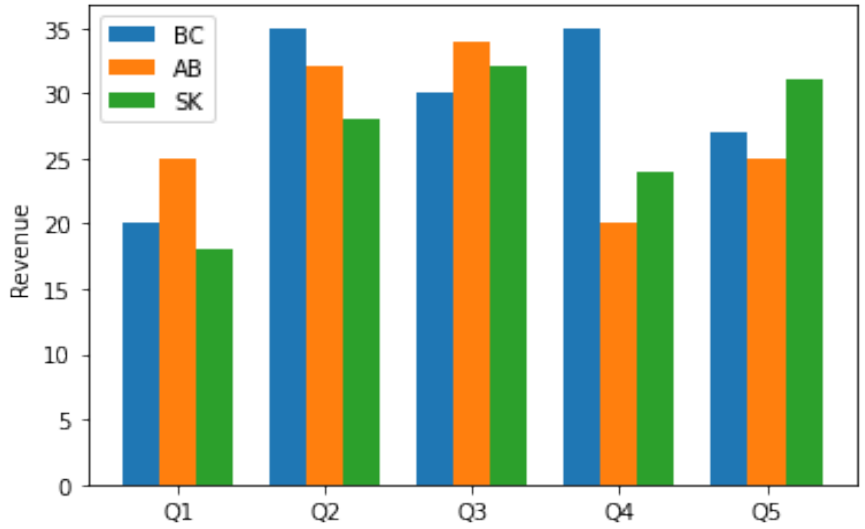


Grouped Bar Plots

```
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()
```



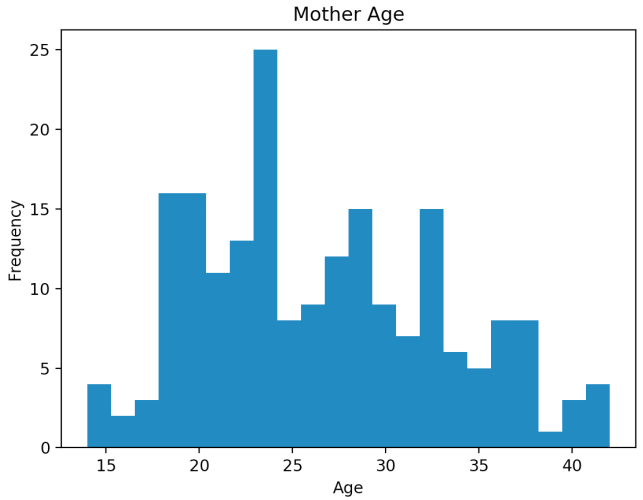
Exercise 9 (4 marks)



Frequency Histograms

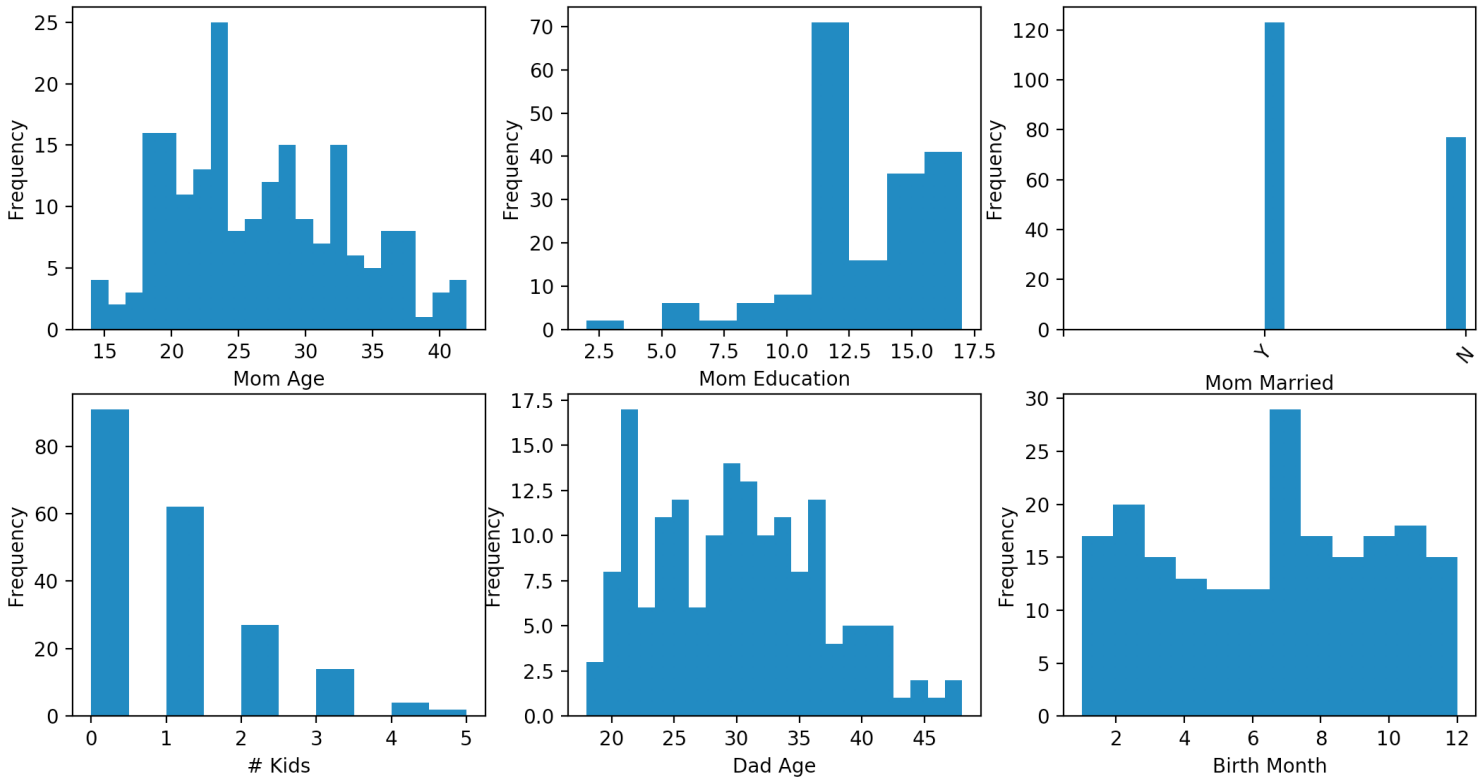
Having tables with precise frequency and ranges is helpful. However, many numbers in a summary report can be overwhelming so histograms offer a quick but effective overview of frequency, range and distribution

```
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()
```



Display 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



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

```
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.

```
plt.subplot(2, 3, 1)
```

```
import pandas as pd
import matplotlib.pyplot as plt

# Import data into a DataFrame.
path = "/Users/pm/Desktop/DayDocs/2019_2020/PythonForDataAnalytics/workingData/ba
bysamp-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'))
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")
```

```
(<Figure size 1008x504 with 6 Axes>,
 array([[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
        [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]], dtype=object))
```