WEEK-02: DATA ANALYSIS_AND_VISUALIZATION_WITH_PYTHON

Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Basic plots in Matplotlib :

Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations. They are typically instruments for reasoning about quantitative information. Some of the sample plots are covered here.

Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

import matplotlib.pyplot as plt or from matplotlib import pyplot as plt

The plot() function is used to draw points (markers) in a diagram. By default, the plot() function draws a line from point to point. The function takes parameters for specifying points in the diagram. Parameter 1 is an array containing the points on the x-axis. Parameter 2 is an array containing the points on the y-axis.

The plt.show() command interacts with our system's interactive graphical backend. plt.show() command should be used only once per Python session, and is most often seen at the very end of the script. Multiple show() commands can lead to unpredictable backend-dependent behavior, and should mostly be avoided.

1. Line plot :

Example 1

```
# importing matplotlib module
#import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot
plt.plot(x,y)
# function to show the plot
plt.show()
```

2. Bar plot:

With pyplot, we can use the bar() function to draw bar graphs. The bar() function takes arguments that describes the layout of the bars. The categories and their values are represented by the first and second argument as arrays.

Example 2

```
# importing matplotlib module
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot the bar
plt.bar(x,y)
```

```
# function to show the plot
plt.show()
```

3. Hist plot:

A histogram is basically used to represent data provided in a form of some groups. It is accurate method for the graphical representation of numerical data distribution. It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

Creating a Histogram

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

In Matplotlib, we use the hist() function to create histograms. The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

The hist() function will read the array and produce a histogram:

You can read from the histogram that there are approximately:
2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

Simple histogram Template:

```
import matplotlib.pyplot as plt

x = [value1, value2, value3,....]

plt.hist(x, bins=number of bins)

plt.show()
```

Attribute	parameter
x	array or sequence of array
bins	optional parameter contains integer or sequence or strings

Step 1: Collect the data for the histogram

Assume the following data about the age of 100 individuals is available:

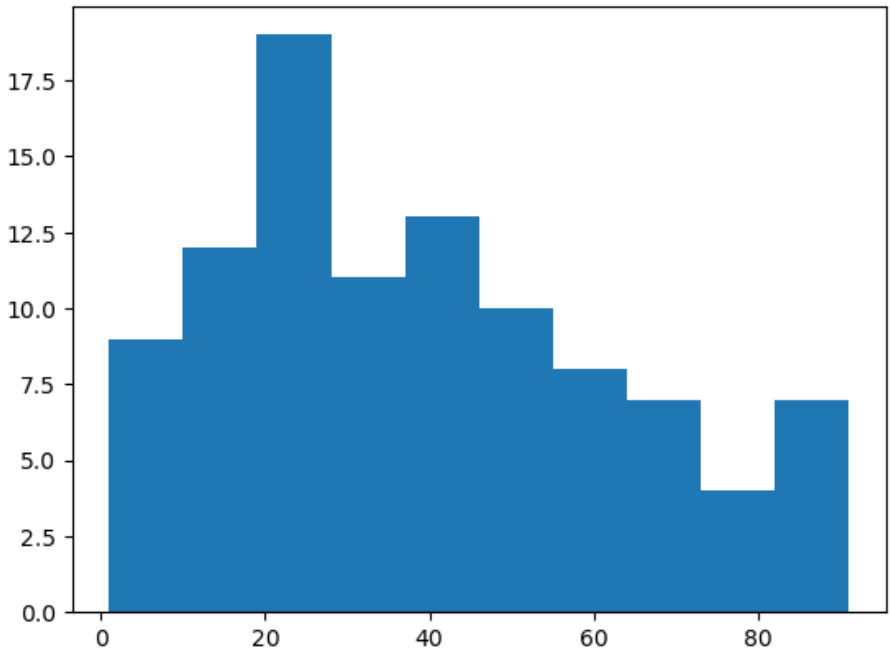
Age
1, 1, 2, 3, 3, 5, 7, 8, 9, 10, 10, 11, 11, 13, 13, 15, 16, 17, 18, 18, 18, 19, 20, 21, 21, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27, 27, 29, 30, 30, 31, 33, 34, 34, 34, 35, 36, 36, 37, 37, 38, 38, 39, 40, 41, 41, 42, 43, 44, 45, 45, 46, 47, 48, 48, 49, 50, 51, 52, 53, 54, 55, 55, 56, 57, 58, 60, 61, 63, 64, 65, 66, 68, 70, 71, 72, 74, 75, 77, 81, 83, 84, 87, 89, 90, 90, 91

Step 2: Determine the number of bins

Next, determine the number of bins to be used for the histogram. For simplicity, set the number of bins to 10. At the end we will see another way to derive the bins.

Step 3: Plot the histogram in Python using matplotlib

```
import matplotlib.pyplot as plt
x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
    10, 11, 11, 13, 13, 15, 16, 17, 18, 18,
    18, 19, 20, 21, 21, 23, 24, 24, 25, 25,
    25, 25, 26, 26, 26, 27, 27, 27, 27, 27,
    29, 30, 30, 31, 33, 34, 34, 34, 35, 36,
    36, 37, 37, 38, 38, 39, 40, 41, 41, 42,
    43, 44, 45, 45, 46, 47, 48, 48, 49, 50,
    51, 52, 53, 54, 55, 55, 56, 57, 58, 60,
    61, 63, 64, 65, 66, 68, 70, 71, 72, 74,
    75, 77, 81, 83, 84, 87, 89, 90, 90, 91
    ]
plt.hist(x, bins=10)
plt.show()
```



Additional way to determine the number of bins

Originally, we set the number of bins to 10 for simplicity.Alternatively, we may derive the bins using the following formulas:

n = number of observations

Range = maximum value – minimum value

Number of intervals = $\sqrt{n}$

Width of intervals = Range / (Number of intervals)

These formulas can then be used to create the frequency table followed by the histogram.

Recall that our dataset contained the following 100 observations:

Age
1, 1, 2, 3, 3, 5, 7, 8, 9, 10, 10, 11, 11, 13, 13, 15, 16, 17, 18, 18, 18, 19, 20, 21, 21, 23, 24, 24, 25, 25, 25, 25, 26, 26, 26, 27, 27, 27, 27, 27, 29, 30, 30, 31, 33, 34, 34, 34, 35, 36, 36, 37, 37, 38, 38, 39, 40, 41, 41, 42, 43, 44, 45, 45, 46, 47, 48, 48, 49, 50, 51, 52, 53, 54, 55, 55, 56, 57, 58, 60, 61, 63, 64, 65, 66, 68, 70, 71, 72, 74, 75, 77, 81, 83, 84, 87, 89, 90, 90, 91

Using our formulas:

n = number of observations = 100

Range = maximum value – minimum value = 91 – 1 = 90

Number of intervals = $\sqrt{n} = \sqrt{100} = 10$

Width of intervals = Range / (Number of intervals) = 90/10 = 9

Based on this information, the frequency table would look like this:

Intervals (bins)	Frequency
0-9	9
10-19	13
20-29	19
30-39	15
40-49	13
50-59	10
60-69	7
70-79	6
80-89	5
90–99	3

Note that the starting point for the first interval is 0, which is very close to the minimum observation of 1 in our dataset. If, for example, the minimum observation was 20, then the starting point for the first interval should be 20, rather than 0.

For the *bins* in the Python code below, we need to specify the values highlighted in blue, rather than a particular number (such as 10, which we used before). Also, we need to include the last value of 99.

This is how the Python code would look like:

```
import matplotlib.pyplot as plt

x = [1, 1, 2, 3, 3, 5, 7, 8, 9, 10,
    10, 11, 11, 13, 13, 15, 16, 17, 18, 18,
    18, 19, 20, 21, 21, 23, 24, 24, 25, 25,
    25, 25, 26, 26, 26, 27, 27, 27, 27, 27,
    29, 30, 30, 31, 33, 34, 34, 34, 35, 36,
    36, 37, 37, 38, 38, 39, 40, 41, 41, 42,
    43, 44, 45, 45, 46, 47, 48, 48, 49, 50,
    51, 52, 53, 54, 55, 55, 56, 57, 58, 60,
```

```
61, 63, 64, 65, 66, 68, 70, 71, 72, 74,
75, 77, 81, 83, 84, 87, 89, 90, 90, 91
]

plt.hist(x, bins=[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 99])
plt.show()
```

Example 3

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)
plt.hist(x)
plt.show()
```

Ref: https://www.w3schools.com/python/matplotlib_histograms.asp

Example 4

```
# importing matplotlib module
from matplotlib import pyplot as plt
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot histogram
plt.hist(y)
# Function to show the plot
plt.show()
```

Ref: https://www.geeksforgeeks.org/plotting-histogram-in-python-using-matplotlib/

4. Scatter plot:

A scatter plot is a diagram where each value in the data set is represented by a dot. The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis.

Example 5

```
# importing matplotlib module
from matplotlib import pyplot as plt
# x-axis values
x = [5, 2, 9, 4, 7]
# Y-axis values
y = [10, 5, 8, 4, 2]
# Function to plot scatter
plt.scatter(x, y)
# function to show the plot
plt.show()
```

SciPy

What is SciPy?

- SciPy is a scientific computation library that uses NumPy underneath.
- SciPy stands for Scientific Python.
- It provides more utility functions for optimization, stats and signal processing.
- Like NumPy, SciPy is open source so we can use it freely.
- SciPy was created by NumPy's creator Travis Olliphant.

Why Use SciPy?

- If SciPy uses NumPy underneath, why can we not just use NumPy?
- SciPy has optimized and added functions that are frequently used in NumPy and Data Science.

Which Language is SciPy Written in?

- SciPy is predominantly written in Python, but a few segments are written in C.

Import SciPy

- Once SciPy is installed, import the SciPy module(s) you want to use in your applications by adding the from scipy import module statement:

  from scipy import constants

constants: SciPy offers a set of mathematical constants, one of them is liter which returns 1 liter as cubic meters.

Unit Categories

- The units are placed under these categories:
- Metric
- Binary
- Mass
- Angle
- Time
- Length
- Pressure
- Volume
- Speed
- Temperature
- Energy
- Power
- Force

Example 6

```
from scipy import constants
print(constants.peta)    #1000000000000000.0
print(constants.tera)    #1000000000000.0
print(constants.giga)    #1000000000.0
print(constants.mega)    #1000000.0
print(constants.kilo)    #1000.0
print(constants.hecto)   #100.0
print(constants.deka)    #10.0
print(constants.deci)    #0.1
print(constants.centi)   #0.01
print(constants.milli)   #0.001
print(constants.micro)   #1e-06
print(constants.nano)    #1e-09
print(constants.pico)    #1e-12
```

Sparse Data

- Sparse data is data that has mostly unused elements (elements that don't carry any information).
- It can be an array like this one: [1, 0, 2, 0, 0, 3, 0, 0, 0, 0, 0, 0]
- **Sparse Data:** is a data set where most of the item values are zero.
- **Dense Array:** is the opposite of a sparse array: most of the values are *not* zero.
- In scientific computing, when we are dealing with partial derivatives in linear algebra we will come across sparse data.

Work With Sparse Data

SciPy has a module, scipy.sparse that provides functions to deal with sparse data.

- There are primarily two types of sparse matrices that we use:
- CSC - Compressed Sparse Column. For efficient arithmetic, fast column slicing.
- CSR - Compressed Sparse Row. For fast row slicing, faster matrix vector products We will use the CSR matrix.

CSR Matrix

We can create CSR matrix by passing an arrray into function scipy.sparse.csr_matrix().

Ref: w3schools.com

Example 7

Create a CSR matrix from an array:

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([0, 0, 0, 0, 0, 1, 1, 0, 2])
print(csr_matrix(arr))
```

```
The example above returns:
  (0, 5) 1
  (0, 6) 1
  (0, 8) 2
```

```
From the result we can see that there are 3 items with value.
The 1. item is in row 0 position 5 and has the value 1.
The 2. item is in row 0 position 6 and has the value 1.
The 3. item is in row 0 position 8 and has the value 2.
```

Sparse Matrix Methods

Viewing stored data (not the zero items) with the data property:

Example 8

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
print(csr_matrix(arr).data)
```

```
[1 1 2]
```

Converting from csr to csc with the tocsc() method:

Example 9

```
import numpy as np
from scipy.sparse import csr_matrix
arr = np.array([[0, 0, 0], [0, 0, 1], [1, 0, 2]])
newarr = csr_matrix(arr).tocsc()
print(newarr)
```

```
(2, 0)          1
 (1, 2) 1
 (2, 2) 2
```

Pandas

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Use of Pandas

Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science.

Data Science: is a branch of computer science where we study how to store, use and analyze data for deriving information from it.

Pandas gives you answers about the data. Like:

- Is there a correlation between two or more columns?
- What is average value?
- Max value?
- Min value?

Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called *cleaning* the data.

Once Pandas is installed, import it in your applications by adding the import keyword:

import pandas

Example 10

```
import pandas
mydataset = {   'cars': ["BMW", "Volvo", "Ford"],
 'passings': [3, 7, 2] }
myvar = pandas.DataFrame(mydataset)
print(myvar)
```

```
cars  passings
0    BMW        3
1  Volvo        7
2   Ford        2
```

Create an alias with the as keyword while importing:

import pandas as pd

Now the Pandas package can be referred to as pd instead of pandas.

Example 11

```
import pandas as pd
mydataset = {
 'cars': ["BMW", "Volvo", "Ford"],
 'passings': [3, 7, 2]
}
myvar = pd.DataFrame(mydataset)
print(myvar)
```

Pandas Series:

A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type. Series data structure is same as the NumPy array data structure but only one difference that is arrays indices are integers and starts with 0, whereas in series, the index can be anything even strings. The labels do not need to be unique but they must be of hashable type.

Indices in a pandas series:

- A pandas series is similar to a list, but differs in the fact that a series associates a label with each element. This makes it look like a dictionary.
- If an index is not explicitly provided by the user, pandas creates a RangeIndex ranging from 0 to N-1.
- Each series object also has a data type.

Example 12

```
import pandas as pd
new_series= pd.Series([5,6,7,8,9,10])
print(new_series)
```

```
0    5
1    6
2    7
3    8
4    9
5   10
dtype: int64
```

- As you may suspect by this point, a series has ways to extract all of the values in the series, as well as individual elements by index.

Example 13

```
import pandas as pd
new_series= pd.Series([5,6,7,8,9,10])
print(new_series.values)
print('_____________')
print(new_series[4])
```

- You can also provide an index manually using index argument. When you have created labels, you can access an item by referring to the label.

Example 14

```
import pandas as pd
new_series= pd.Series([5,6,7,8,9,10], index=['a','b','c','d','e','f'])
print(new_series)
print('_____________')
print(new_series.values)
print('_____________')
print(new_series['f'])
```

- It is easy to retrieve several elements of a series by their indices or make group assignments. Since the index values are customized, they are used to access the values in the series like series_name[‘index_name’]. When multiple index values need to be accessed, they are first specified in a list and then series indexing can be used to access these values. Note − Observe the two ‘[[‘ in the code.

Example 15

```
import pandas as pd
new_series= pd.Series([5,6,7,8,9,10], index=['a','b','c','d','e','f'])
print(new_series)
print('_____________')
print(new_series[['b', 'c', 'f']])
```

- Filtering and maths operations are easy with Pandas as well.

Example 16

```
import pandas as pd
new_series= pd.Series([5,6,7,8,9,10], index=['a','b','c','d','e','f'])
new_series2= new_series[new_series>0]  # check with 7 instead of 0
print(new_series2)
print('_____________')
```

```
new_series2= new_series[new_series>0]  * 2
print(new_series2)
```

DataFrame:

- A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns. Series is like a column, a DataFrame is the whole table.
- Simplistically, a data frame is a table, with rows and columns.
- Each column in a data frame is a series object.
- Rows consist of elements inside series.

Case ID	*Variable one*	*Variable two*	*Variable 3*
1	*123*	*ABC*	*10*
2	*456*	*DEF*	*20*
3	*789*	*XYZ*	*30*

- Pandas data frames can be constructed using Python dictionaries.

Example 17

Create a simple Pandas DataFrame:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns. Pandas use the loc attribute to return one or more specified row(s)

Example 18

Return row 0:

```
#refer to the row index:
print(df.loc[0])
```

This example returns a Pandas Series

Named Indexes

With the index argument, you can name your own indexes.

Example 19

Add a list of names to give each row a name:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

- You can also create a data frame from a list. We can update the auto-created column name/labels by sending a list of values to the DataFrame.columns attribute “df.columns”.

Example 20

```
import pandas as pd
```

```
# create a Pandas DataFrame

list2=[[0,1,2],[3,4,5],[6,7,8]]
df= pd.DataFrame(list2)
print("DataFrame:")
print(df)

# set the column labels
 df.columns = ['C1','C2','C3']
print("Column names are Updated:")
print(df)
```

Example 21

```
import pandas as pd
df=pd.DataFrame({
    'Country': ['Kazakhstan','Russia','Belarus','Ukraine'],
    'Population': [17.04,143.5,9.5,45.5],
    'Square':[2724902, 17125191,207600,603628]
   })
print(df)
```

- DataFrame.dtypes: Return the dtypes in the DataFrame. This returns a Series with the data type of each column. Columns with mixed types are stored with the object dtype.
- print(df.dtypes): check the data type of all columns in the DataFrame
- Use DataFrame.dtypes[‘C’] attribute to find out the data type (dtype) of column ‘C’ in the given Dataframe. Or equivalently use print(df['Country'].dtypes)
- print(df.dtypes['Country']) #print(df['Country'].dtypes)
- A Pandas data frame object has two indices; a column index and row index. Again, if you do not provide one, Pandas will create a RangeIndex from 0 to N-1.
- index- The DataFrame attribute index returns the row index
- columns - attribute returns the column indexes.

Example 22

```
import pandas as pd
df=pd.DataFrame({
    ‘Country’: [‘Kazakhstan’,’Russia’,’Belarus’,’Ukraine’],
    ‘Population’: [17.04,143.5,9.5,45.5]
    ‘Square’:[2724902, 17125191,207600,603628]
   })
print(df.columns)
print(‘_____’)
print(df.index)
```

- There are numerous ways to provide row indices explicitly.
- For example, you could provide an index when creating a data frame:

Example 23

```
import pandas as pd
df=pd.DataFrame({
   'Country': ['Kazakhstan','Russia','Belarus','Ukraine'],
   'Population': [17.04,143.5,9.5,45.5],
    'Square':[2724902,17125191,207600,603628],
    }, index = ['KZ','RU','BY','UA'])
print(df)
```

- or do it during runtime.Here, also name the index ‘country code’.

Example 24

```
import pandas as pd
df=pd.DataFrame({
```

```
    'Country': ['Kazakhstan','Russia','Belarus','Ukraine'],
    'Population': [17.04,143.5,9.5,45.5],
     'Square':[2724902,17125191,207600,603628],
     })
print(df)
print('_____')
df.index = ['KZ','RU','BY','UA']
df.index.name = 'Country Code'
print(df)
```

- Row access using index can be performed in several ways.
- First, you could use .loc() and provide an index label.

Example 25

```
print(df.loc['KZ'])
```

- Second, you could use .iloc() and provide an index number. Dataframe.iloc[] method is used when the index label of a data frame is something other than numeric series of 0, 1, 2, 3….n or in case the user does not know the index label. Rows can be extracted using an imaginary index position that is not visible in the Dataframe.

```
print(df.iloc[0])
```

- A selection of particular rows and columns can be selected this way.

```
print(df.loc[['KZ','RU'],'Population'])
```

- You can feed .loc() two arguments, index list and column list, slicing operation is supported as well. We can slice it in a number of ways. The Pandas DataFrame syntax takes the first set of indexes in the operator for row slicing and the second set for column slicing. Here is the general rule:
- df.loc[row slicing, column slicing] or df.iloc[startrow:endrow, startcolumn:endcolumn]
- When data is indexed using labels or integers, the best approach is typically to use the loc function.
- When data is indexed using integers only, the best approach is typically to use the iloc function.

```
import numPy as np
import pandas as pd
df = pd.DataFrame(np.arange(20).reshape(5,4), columns=["A", "B", "C", "D"])
print(df)
```

- This line tells NumPy to create a list of integers from 0 to 19 values in the "shape" of 5 rows x 4 columns and to label the columns A through D.

```
A B C D
0 0 1 2 3
1 4 5 6 7
2 8 9 10 11
3 12 13 14 15
4 16 17 18 19
```

Slicing Rows and Columns by Label

The following is an example of how to slice both rows and columns by label using the loc function: df.loc[:, "B":"D"] This line uses the slicing operator to get DataFrame items by label. The first slice [:] indicates to return all rows. The second slice specifies that only columns B, C, and D should be returned.

```
B C D
0 1 2 3
1 5 6 7
2 9 10 11
3 13 14 15
4 17 18 19
```

Slicing Rows and Columns by Index Position

The following is an example of how to slice both rows and columns by index position using the iloc attribute, focusing on row slicing:

df.iloc[0:2, :]
With the iloc function, you specify the range and even the steps while slicing columns and rows. In this case, the first slice [0:2] is requesting only rows 0 through 1of the DataFrame. When slicing by index position in Pandas, the start index is included in the output, but the stop index is one step beyond the row you want to select. So the slice return row 0 and row 1, but does not return row 2. The second slice [:] indicates that all columns are required. The results are shown below.

Output:

A B C D
0 0 1 2 3
1 4 5 6 7

Slicing Columns using the iloc attribute

The following is another example of how to slice using the iloc attribute, focusing on column slicing:

df.iloc[:, 1:3]
In this case, the first operator [:] requires all rows be returned. The second operator [1:3] yields columns B and C only. This is because our DataFrame contains column A (index 0) through column D (index 3). Recall that when slicing a DataFrame by index we specify the stop bound one column beyond what we want. In this case, we are telling pandas to slice from the column with index 1 (B) through index 3 (D), but do not include index 3 (D). This returns column indexed 1 and 2 only. The results are shown below.

Output:

B C
0 1 2
1 5 6
2 9 10
3 13 14
4 17 18

Slicing Specific Rows and Columns using iloc attribute

This next example shows how to slice a specific set of rows and columns using the iloc attribute:

df.iloc[1:2, 1:3]
The first operator [1:2] is requesting only rows 1 through 2. This yields 1 row only because you are actually telling pandas to start at index 1 and select index 2, but exclude 2 because it is the last index. The second operator [1:3] has yielded columns B through C only. Remember that our DataFrame has column A (index 0) through column D (index 3). We get these results because we are telling pandas to slice from columns with index 1 through 3, excluding index 3. This yields columns indexed 1 and 2 only. The results are as shown below.

Output:
B C
1 5 6

Alternative to Slicing Specific Rows and Columns using iloc attribute

Finally, an alternative example of how to slice a specific set of rows and columns using the iloc attribute:

df.iloc[:2, :2]
The first operator [:2] requires the data from the start of the rows to the second row (i.e, row 0, and row 1). The second operator [:2] requires data from the start of the columns to column 2 (i.e, A, and B). The output is shown below.

Output:
A B
0 0 1
1 4 5

```
print(df.loc[['KZ':'BY', :])
```

Ref: https://levelup.gitconnected.com/how-to-slice-a-dataframe-in-pandas-884bd8b298a6

Filtering

- Filtering is performed using so-called Boolean arrays.

Example 26

```
print([[df.Population > 10],'Country','Square'])
```

Deleting columns

You can delete a column using the drop() function. The drop() method removes the specified row or column. By specifying the column axis (axis='columns'), the drop() method removes the specified column. By specifying the row axis (axis='index'), the drop() method removes the specified row.

```
print(df)
df = df.drop(['Population'], axis = 'columns')
print(df)
```

Combining all operations:

```
import pandas as pd
df=pd.DataFrame({
   'Country': ['Kazakhstan','Russia','Belarus','Ukraine'],
   'Population': [17.04,143.5,9.5,45.5],
    'Square':[2724902,17125191,207600,603628],
    })
print(df)
print('1_____')
df.index = ['KZ','RU','BY','UA']
df.index.name = 'Country Code'
print(df)
print('_2_____')
print(df.loc['KZ'])
print('3_____')
print(df.iloc[0])
print('4_____')
print(df.loc[['KZ','RU'],'Population'])
print('5_____')
print(df.loc['KZ':'UA', :])
print('6_____')
print([[df.Population > 10],'Country','Square'])
print('7_____')
print(df)
print('8_____')
df = df.drop(['Population'], axis = 'columns')
print(df)
print('9_____')
```

```
Country  Population    Square
0  Kazakhstan       17.04   2724902
1      Russia      143.50  17125191
2     Belarus        9.50    207600
3     Ukraine       45.50    603628
1_____
              Country  Population    Square
Country Code
```

```
KZ          Kazakhstan       17.04   2724902
RU              Russia      143.50  17125191
BY             Belarus        9.50    207600
UA             Ukraine       45.50    603628
_2_____________________
Country       Kazakhstan
Population         17.04
Square           2724902
Name: KZ, dtype: object
3_____________________
Country       Kazakhstan
Population         17.04
Square           2724902
Name: KZ, dtype: object
4_____________________
Country Code
KZ     17.04
RU    143.50
Name: Population, dtype: float64
5_____________________
              Country  Population    Square
Country Code
KZ          Kazakhstan       17.04   2724902
RU              Russia      143.50  17125191
BY             Belarus        9.50    207600
UA             Ukraine       45.50    603628
6_____________________
[[Country Code
KZ     True
RU     True
BY    False
UA     True
Name: Population, dtype: bool], 'Country', 'Square']
7_____________________
              Country  Population    Square
Country Code
KZ          Kazakhstan       17.04   2724902
RU              Russia      143.50  17125191
BY             Belarus        9.50    207600
UA             Ukraine       45.50    603628
8_____________________
              Country    Square
Country Code
KZ          Kazakhstan   2724902
RU              Russia  17125191
BY             Belarus    207600
UA             Ukraine    603628
9_____________________
```

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example 27

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv')
print(df)
```

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files). CSV files contains plain text and is a well know format that can be read by everyone including Pandas. In our examples we will be using a CSV file called 'data.csv'.

Example 28

Load the CSV into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.to_string())  # use to_string() to print the entire DataFrame.
print(df)   #Print the DataFrame without the to_string() method
```

Data Cleaning

Data cleaning means fixing bad data in your data set.

Bad data could be:

- Empty cells
- Data in wrong format
- Wrong data
- Duplicates

Plotting

Pandas uses the plot() method to create diagrams. We can use Pyplot, a submodule of the Matplotlib library to visualize the diagram on the screen.

Example 29

Import pyplot from Matplotlib and visualize our DataFrame:

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('data.csv')
df.plot()
plt.show()
```

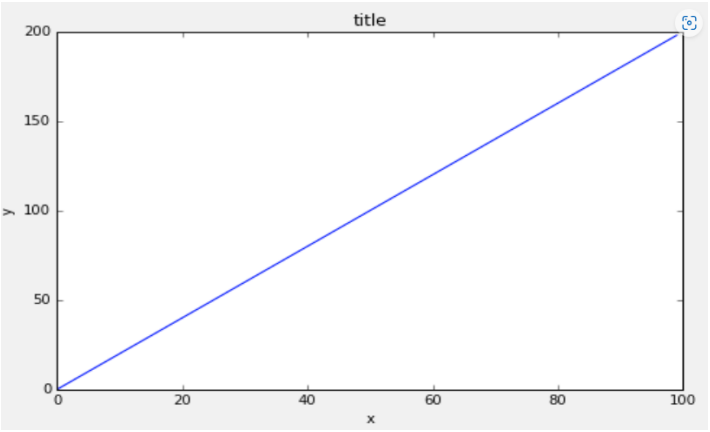
Questions

1. Follow along with these steps:

a) Create a figure object called fig using plt.figure()

b) Use add_axes to add an axis to the figure canvas at [0,0,1,1]. Call this new axis ax.

c) Plot (x,y) on that axes and set the labels and titles to match the plot below:



2. Create a figure object and put two axes on it, ax1 and ax2. Located at [0,0,1,1] and [0.2,0.5,.2,.2] respectively. Now plot (x,y) on both axes. And call your figure object to show it.

3. Use the company sales dataset csv file, read Total profit of all months and show it using a line plot

Total profit data provided for each month. Generated line plot must include the following properties: –

a. X label name = Month Number
b. Y label name = Total profit

4. Use the company sales dataset csv file, get total profit of all months and show line plot with the following Style properties. Generated line plot must include following Style properties: –

a. Line Style dotted and Line-color should be red
b. Show legend at the lower right location.
c. X label name = Month Number
d. Y label name = Sold units number
e. Add a circle marker.
f. Line marker color as read
g. Line width should be 3

Additional Questions

1. Use the company sales dataset csv file, read all product sales data and show it using a multiline plot. Display the number of units sold per month or each product using multiline plots. (i.e., Separate Plotline for each product).
2. Use the company sales dataset csv file, calculate total sale data for last year for each product and show it using a Pie chart.
   Note: In Pie chart display Number of units sold per year for each product in percentage.