Getting Started With Python

Akash Perni, Ganesh Pendala, Samba Sivesh Kurra
Maryland Applied Graduate Engineering
University of Maryland
Collge Park, MD, United States
aperni@umd.edu, spendela@umd.edu, shiv1818@umd.edu

***Abstract*—This Paper explores data analysis and programming in Python. It discusses the foundational ideas of variable manipulation and draws attention to the distinctions between text strings and numerical variables. The study also examines the list and fundamental operations of numpy arrays, including statistical analysis and reshaping mathematical procedures. The Pandas library is also used to read a CSV file and convert it into a data frame for actual data management. Finally, a Histogram is generated from the data frame using the Matplotlib tool.**

I. Introduction

This paper covers essential data manipulation and analysis tasks, including text and numerical variable handling, list and dictionary operations, array transformations, and CSV data extraction. The paper explores text manipulation, differentiates string and numerical variables, works with lists, covers array operations, and introduces a CSV data reading function for analyzing names data with histograms.

II. Methodology

A. Variables and its Operations

At the core of Python programming, variables serve as data containers, enabling various operations. This section delves into the fundamental aspects of Python's variables, data types, and basic functions. Below are the examples of the operations on the variables

```
In [1]: a="Unsupervised learning needs a target"
        a=a.upper()
        print(a)

UNSUPERVISED LEARNING NEEDS A TARGET
```

Fig. 1. Converting String to Upper Case

Fig. 2. Variable Inspectors for Numerical Variables

The difference between changes to string variables and numerical variables is that we can assign numbers, letters, characters, symbols, and words to a string variable, as in numerical variables, we can only assign the numerical values.

B. Data Structures and its Operations

In Python, a data structure is a method for efficiently arranging and storing data, enabling convenient retrieval and manipulation of the information. Python provides four different types of predefined data structures for other purposes.

In Python, a list and a dictionary are two fundamental data structures for organizing and storing data.

```
l1=["akash","priya","ganesh","sivesh","virat"]
l2=[22,21,22,23,34]

l1.append("vijay")
print(l1)

l1.pop()
print(l1)

for i in range(len(l1)):
    print(l1[i],l2[i])


l1.extend(l2)
print(l1)

l2.append(45)
print(l2)

l2.pop()
print(l2)
```

```
['akash', 'priya', 'ganesh', 'sivesh', 'virat', 'vijay']
['akash', 'priya', 'ganesh', 'sivesh', 'virat']
akash 22
priya 21
ganesh 22
sivesh 23
virat 34
['akash', 'priya', 'ganesh', 'sivesh', 'virat', 22, 21, 22, 23, 34]
[22, 21, 22, 23, 34, 45]
[22, 21, 22, 23, 34]
```

Fig. 3. List Properties

The append() method pushes the elements to the end of the list, and the pop() method returns the list's last element or any element at the specified index.

A dictionary is one of the built-in data structures in Python that allows to store and retrieve the data in a key-value pair format. A dictionary can be declared using the dict() method or a set of flower braces.

```
d1={}
l1=["akash","priya","ganesh","sivesh","virat"]
l2=[22,21,22,23,34]

for i in range(len(l1)):
    d1[l1[i]]=l2[i]+1
print(d1)
```

```
{'akash': 23, 'priya': 22, 'ganesh': 23, 'sivesh': 24, 'virat': 35}
```

Fig. 4. Dictionaries in Python

Set is a built-in data type representing the unordered collection of unique elements in Python. A set can be declared using a set() method or curly braces.

```
l2=list(set(l2))
print(sorted(l2))
```

```
[21, 22, 23, 34]
```

Fig. 5. Set in Python

User-defined functions in Python are custom-made blocks of reusable code that perform specific tasks. You create functions using the def keyword, and they can accept parameters and return values, making your code easy to maintain. In contrast, list comprehensions are concise ways to create lists in Python. They allow you to generate new lists by applying a condition or expression to each element in an existing iterable and optionally filtering the items based on a situation.

```
l3=[]
def mean_(lst):
    return sum(l2)//len(l2)
x=mean_(l2)

for i in l2:
    l3.append(i-x)
print(l3)

#using list comprehension
l4=[x-(sum(l2)//len(l2)) for x in l2]
print(l4)
```

```
[9, -4, -3, -2]
[9, -4, -3, -2]
```

Fig. 6. Functions and List comprehensions in Python

C. NumPy Data Structures

NumPy (Numerical Python) is an open-source library in Python that supports working with large, multi-dimensional arrays and matrices and provides a range of mathematical operations that can be executed on these arrays.

NumPy arrays, also known as ndarrays, are a fundamental data structure in the NumPy library for Python. They are used to store and manipulate large sets of data efficiently. Numpy arrays can declared in many ways. The numpy package is imported to work with numpy. The np. arrange () function creates a one-dimensional array. The np.reshape is used to change the shape of an existing NumPy array while keeping the same data. It allows you to convert a array from one-dimensional to multi-dimensional or vice versa.

```
import numpy as np
l5=np.arange(0,50)
print(l5)
n=l5.size
print(n)
m=5 #no of columns
z=np.reshape(l5,(10,m))
print(z)
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49]
50
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]
```

Fig. 7. Numpy arrays

For accessing multi-dimensional arrays, you can use multiple indices or slices to access elements along different dimensions.

```
print(z[:,2])
```

```
[ 2  7 12 17 22 27 32 37 42 47]
```

Fig. 8. Acessing multi dimensional arrays

The np.ndim() function returns the number of dimensions in a numpy array, and the np.square() function doubles all the elements of the numpy array.

```
z1=np.square(z)
print(z1)
print(z1.ndim)
```

```
[[   0    1    4    9   16]
 [  25   36   49   64   81]
 [ 100  121  144  169  196]
 [ 225  256  289  324  361]
 [ 400  441  484  529  576]
 [ 625  676  729  784  841]
 [ 900  961 1024 1089 1156]
 [1225 1296 1369 1444 1521]
 [1600 1681 1764 1849 1936]
 [2025 2116 2209 2304 2401]]
2
```

Fig. 9. Operations on numpy array

```
x=2 #scalar value
print(z)
z2=z*2
print(z2)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]
 [25 26 27 28 29]
 [30 31 32 33 34]
 [35 36 37 38 39]
 [40 41 42 43 44]
 [45 46 47 48 49]]
[[ 0  2  4  6  8]
 [10 12 14 16 18]
 [20 22 24 26 28]
 [30 32 34 36 38]
 [40 42 44 46 48]
 [50 52 54 56 58]
 [60 62 64 66 68]
 [70 72 74 76 78]
 [80 82 84 86 88]
 [90 92 94 96 98]]
```

Fig. 10. Multiplying a scalar value by every member of the array

To find all the numbers greater than ten and their sum, the two-dimensional array converts into a one-dimensional array, and the np.sum() function calculates the sum of the array.

```
one_dimensional=np.reshape(z2,(1,50))
print(one_dimensional)
numbers_greater_than=[]
for i in one_dimensional[0]:
    if i>10:
        numbers_greater_than.append(i)
print(numbers_greater_than)

print(np.sum(numbers_greater_than))
```

Fig. 11. Creating a new array with only numbers greater than 10 and finding their sum

The output of the above code is:

[[0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98]]

[12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98]

2420

NumPy offers various statistical functions and methods that allow you to perform multiple statistical operations on NumPy arrays. Minimum and Maximum, Variance, Standard Deviation, Median, Mode, and Mean are some statistics performed on numpy arrays. The axis parameter specifies the axis on which the operation is performed, and it is often used to deal with multi-dimensional arrays, mainly to perform the statistic operation on specific row and column.

```
print(z2)
print("-----------")
print("np.median(z2)",np.median(z2))
print("np.median(z2,axis=0)", np.median(z2,axis=0))
print("np.median(z2,axis=1)", np.median(z2,axis=1))

print("-----------")
print(np.mean(z2))
print(np.mean(z2,axis=0))
print(np.mean(z2,axis=1))

print("-----------")
print(np.std(z2))
print(np.std(z2,axis=0))
print(np.std(z2,axis=1))

print("-----------")
print(np.var(z2))
print(np.var(z2,axis=0))
print(np.var(z2,axis=1))


print("-----------")
print(np.average(z2))
print(np.average(z2,axis=0))
print(np.average(z2,axis=1))

print("-----------")
print(np.ptp(z2))
print(np.ptp(z2,axis=0))
print(np.ptp(z2,axis=1))
```

```
[[ 0  2  4  6  8]
 [10 12 14 16 18]
 [20 22 24 26 28]
 [30 32 34 36 38]
 [40 42 44 46 48]
 [50 52 54 56 58]
 [60 62 64 66 68]
 [70 72 74 76 78]
 [80 82 84 86 88]
 [90 92 94 96 98]]
-----------
np.median(z2) 49.0
np.median(z2,axis=0) [45. 47. 49. 51. 53.]
np.median(z2,axis=1) [ 4. 14. 24. 34. 44. 54. 64. 74. 84. 94.]
-----------
49.0
[45. 47. 49. 51. 53.]
[ 4. 14. 24. 34. 44. 54. 64. 74. 84. 94.]
-----------
28.861739379323623
[28.72281323 28.72281323 28.72281323 28.72281323 28.72281323]
[2.82842712 2.82842712 2.82842712 2.82842712 2.82842712 2.82842712
 2.82842712 2.82842712 2.82842712 2.82842712]
-----------
833.0
[825. 825. 825. 825. 825.]
[8. 8. 8. 8. 8. 8. 8. 8. 8. 8.]
-----------
49.0
[45. 47. 49. 51. 53.]
[ 4. 14. 24. 34. 44. 54. 64. 74. 84. 94.]
-----------
98
[90 90 90 90 90]
[8 8 8 8 8 8 8 8 8 8]
```

Fig. 12. Statistics on numpy array

D. Operations on Files and Histogram

The Pandas library gets imported when pandas as pd is used to import it. To read the text file, utilize the pd read csv function. Pandas's "read" mode is the default for reading a CSV or text file. Here the data from the yob1880.txt file is transformed into a data frame using pandas library. Also the names of the three columns are assigned as Name, Gender, Count. Also, to print the top 5 rows in a data frame , df.head() method is used. Refer fig.13 for the code.

```
import csv
import pandas as pd
file=open( "yob1880.txt", "r")
df=pd.read_csv("yob1880.txt",header=None)
df.rename(columns={0: 'Name', 1: 'Gender', 2:'Count'}, inplace=True)
df.head()
```

	Name	Gender	Count
0	Mary	F	7065
1	Anna	F	2604
2	Emma	F	2003
3	Elizabeth	F	1939
4	Minnie	F	1746

Fig. 13. Transforming a text file to a data frame

Here each column in the data frame is assigned to an individual lists. The list() function is used to create a new list from an iterable. Refer the Fig.14 for the code.

```
names=list(df.Name)
gender=list(df.Gender)
count=list(df.Count)
```

Fig. 14. assigning the columns to individual lists

To plot an histogram based on data frame columns, we need to import pyplot from the matplotlib library. so according to the assignment question, only the top 10 names and their counts from the Names and Counts column are used to plot the histogram. So to separate the top 10 rows from a data frame, the df.iloc[:10] function is used. Below you can see the histogram based on the Names of the persons with respect to their name counts. For the X label the ”Names” column is assigned and for the Y label the ”Count” column is assigned. Also, the the figure size and title can be assigned using plt.figure(figsize=?) function and plt.title() function. Refer the Fig.15 for the histogram code.

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


df2 = df.iloc[:10]

# Create a histogram
plt.figure(figsize=(12, 6))
plt.bar(df2["Name"], df2["Count"], color='blue')
plt.xlabel('Names')
plt.ylabel('Count')
plt.title('histogram of names wrt count')

plt.show()
```

Fig. 15. Plotting the histogram based on Names and their Counts

Below is the output for the histogram code.

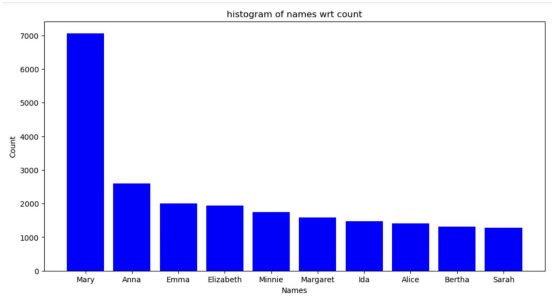

Fig. 16. Histogram based on names and their count

III. CONCLUSION

Python is very useful for beginners because of its simplicity and easy-to-write syntax. The concepts of variables, data structures including lists, dictionaries, NumPy one dimensional and two dimensional arrays, opening and accessing data in files, and plotting using matplotlib are basic useful topics in Python.

REFERENCES

[1] https://www.w3schools.com/python/python-variables.asp
[2] https://www.w3schools.com/python/python-variables.asp
[3] https://www.analyticsvidhya.com/blog/2021/06/working-with-lists-dictionaries-in-python/
[4] https://www.w3schools.com/python/matplotlib-pyplot.asp
[5] https://numpy.org/doc/stable/reference/generated/numpy.reshape.html
[6] https://www.geeksforgeeks.org/numpy-ndarray-ndim-method-python/