Data handling in Python

By

Prof. Dr. Sourav Saha

Basic Python commands

Sr #	Command	Purpose	Sample code with output
1	os.getcwd()	Getting the current working directory	>>> os.getcwd() C:\Users\amitk
2	os.chdir()	Setting the current working directory	>>> os.chdir('C:\\Python programs')
3	os.listdir()	See directory content	>>> os.listdir('C:\\Python programs') Out[16]: ['Machine_Learning_Python.py']
4	python '<<filename>>'	Compile Source File for Execution	>>> python new1.py
5	print()	Command for basic user output	>>> print("Hello") Hello

6	input ()	Command for basic user input	>>> a = input("Give input: ")
			Give input: 12
			>>> print("Your input: ", a)
			Your input: 12

7	type()	Gives the type of an object	>>> x = 5
			>>> type(x)
			Out[28]: int

8	help(<<keyword>>)	Access help related to some function.	>>> help(print) >>> help(os.listdir)

Important libraries in Python

- **scikit-learn**
  - **Numpy** – Fundamental package for scientific computing
  - **SciPy** – Package providing mathematical functions and statistical distributions
- **matplotlib** – Primary library supporting scientific plotting e.g. line diagrams, histograms, scatter plots
- **pandas** – Primary library providing data manipulation functionalities

Basic Python Libraries - NumPy

NumPy package contains functionality for multidimensional arrays, high-level mathematical functions e.g. linear algebra and Fourier transform operations, random number generators, etc.

In scikit-learn, NumPy array is the primary data structure, used to input data. Any data used needs to be converted to a NumPy array.

numpy.array(*object, dtype, copy, order, subok, ndmin*)

dtype means *data-type* i.e. the desired data-type for the array. If not given, then the type will be determined as the minimum type required to hold the objects in the sequence.

- ✓ *empty* - Return a new uninitialized array
- ✓ *full* - Return a new array of given shape filled with value
- ✓ *ones* - Return a new array setting values to one
- ✓ *zeros* - Return a new array setting values to zero

Basic Python Libraries - NumPy

```
# Defining an array variable with data ...
import numpy as np

arr1 = np.empty((2,3))

arr2 = np.array([[10,2,3], [23,45,67]])
print(var1)
[[10  2  3]
 [23 45 67]]

# Create an array of 1s ...
Arr3 = np.ones((2,3))
[[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]]

# Create an array of 0s ...
Arr4 = np.zeros((2,3),dtype=np.int)
[[0, 0, 0],
 [0, 0, 0]]

# Create an array with random numbers ...
np.random.random((2,2))
[[ 0.47448072,  0.49876875],
 [ 0.29531478,  0.48425055]]
```

Basic Python Libraries – NumPy (contd.)

```
# Defining 1-D array variable with data ...
var2 = np.empty(4)
var2[0] = 5.67
var2[1] = 2
var2[2] = 56
var2[3] = 304
print(var2)
[   5.67    2.     56.    304.  ]
print(var2.shape)  # Returns the dimension of the array ...
(4,)
print(var2.size)   # Returns the size of the array ...
4
# Defining 2-D array variable with data ...
var3 = np.empty((2,3))
var3[0][0] = 5.67
var3[0][1] = 2
var3[0][2] = 56
var3[1][0] = .09
var3[1][1] = 132
var3[1][2] = 1056
print(var3)
[[  5.67000000e+00   2.00000000e+00   5.60000000e+01]
 [  9.00000000e-02   1.32000000e+02   1.05600000e+03]]
[Note: Same result will be obtained with dtype=np.float]
print(var3.shape)
(2, 3)
```

Basic Python Libraries – NumPy (contd.)

```
# Same declaration with dtype mentioned ...
var3 = np.empty((2,3), dtype=np.int)
[[   5,    2,   56],
 [   0,  132, 1056]]
print(var3[1])    # Returns a row of an array ...
[   0  132 1056]
print(var3[[0, 1]]) # Returns multiple rows of an array ...
[[   5    2   56]
 [   0  132 1056]]
print(var3[:, 2]) # Returns a column of an array ...
[  56 1056]
print(var3[:, [1, 2]]) # Returns multiple column of an array ...
[[   2   56]
 [ 132 1056]]
print(var3[1][2]) # Returns a cell value of an array ...
1056
print(var3[1, 2]) # Returns a cell value of an array ...
1056
print(np.transpose(var3)) # Returns transpose of an array ...
[[   5    0]
 [   2  132]
 [  56 1056]]
print(var3.reshape(3,2)) # Returns a re-shaped array ...
[[   5    2]
 [  56    0]
 [ 132 1056]]
```

Basic Python Libraries – NumPy (contd.)

Create and concatenate arrays:

```
import numpy as np

arr1= np.empty((2,3), dtype=np.int)
arr1[0][0] = 5.67
arr1[0][1] = 2
arr1[0][2] = 56
arr1[1][0] = .09
arr1[1][1] = 132
arr1[1][2] = 1056

[[   5,    2,   56],
 [   0,  132, 1056]]

arr2 = np.empty((1,3), dtype=np.int)
arr2[0][0] = 37
arr2[0][1] = 2.193
arr2[0][2] = 5609

[[  37,    2, 5609]]
```

Basic Python Libraries – NumPy (contd.)

```
arr_concat = np.concatenate((arr1, arr2), axis = 0)
print(arr_concat)

[[   5    2   56]
 [   0  132 1056]
 [  37    2 5609]]


var2.min()  # Returns minimum value stored in an array ...
2.0

var2.max()  # Returns maximum value stored in an array ...
304.0

var2.cumsum() # Returns cumulative sum of the values stored in an array
...
array([   5.67,    7.67,   63.67,  367.67])

var2.mean() # Returns mean or average value stored in an array ...
91.917500000000004

var2.std() # Returns standard deviation of values stored in an array ...
124.2908299865682
```

Basic Python Libraries – NumPy (contd.)

Sr #	Command	Purpose	Sample code with output

1	sin, cos, tan, arcsin, arccos, arctan, degrees, etc.	Trigonometric functions	>>> import numpy as np
			>>> from numpy import pi
			>>> array1 = np.array([30,60,90])
			>>> np.sin(a*np.pi/180)
			array([0.5, 0.70710678, 1.])

2	around, floor, ceil	For rounding decimals to the desired precision	>>> arr2 = np.array([67.07,88.10, 34, 231.67, 0.934])
			>>> print(arr2)
			[67.07 88.1 34. 231.67 0.934]
			>>> np.around(arr2)
			array([67., 88., 34., 232., 1.])
			>>> np.around(arr2, decimals = 2)
			array([67.07, 88.1 , 34. , 231.67, 0.93])
			>>> np.floor(arr2)
			array([67., 88., 34., 231., 0.])
			>>> np.ceil(arr2)
			array([68., 89., 34., 232., 1.])

Basic Python Libraries – NumPy (contd.)

Sr #	Command	Purpose	Sample code with output

3	add, subtract, multiply, divide, power, reciprocal, mod, etc.	Basic mathematical operations on arrays	>>> arr1 = np.arange(6, dtype = np.int).reshape(2,3)
			>>> arr1
			array([[0, 1, 2],
			[3, 4, 5]])
			>>> arr2 = np.arange(4, 15, 2, dtype = np.int).reshape(2,3)
			>>> arr2
			array([[4, 6, 8],
			[10, 12, 14]])
			>>> np.add(arr1, arr2)
			array([[4, 7, 10],
			[13, 16, 19]])
			>>> np.subtract(arr1, arr2)
			array([[-4, -5, -6],
			[-7, -8, -9]])
			>>> np.multiply(arr1, arr2)
			array([[0, 6, 16],
			[30, 48, 70]])
			>>> np.divide(arr2, arr1)
			array([[inf, 6., 4.],
			[3.33333333, 3.,2.8]])
			>>> np.power(arr1,2)
			array([[0, 1, 4],
			[9, 16, 25]])
			>>> np.mod(arr2, arr1)
			array([[0, 0, 0],
			[1, 0, 4]])
			>>> np.remainder(arr2, arr1)
			array([[0, 0, 0],
			[1, 0, 4]])

Basic Python Libraries – NumPy (contd.)

Sr #	Command	Purpose	Sample code with output

4	sort, where, nonzero	For sorting and searching	>>> a = np.array([[21,7, 14],[19,40,8]])
			>>> a
			array([[21, 7, 14],
			[19, 40, 8]])
			>>> np.sort(a)
			array([[7, 14, 21],
			[8, 19, 40]])
			>>> np.sort(a, axis = 0)
			array([[19, 7, 8],
			[21, 40, 14]])
			>>> np.sort(a, axis = 1)
			array([[7, 14, 21],
			[8, 19, 40]])
			>>> np.where(a > 13)
			(array([0, 0, 1, 1]),
			array([0, 2, 0, 1]))
			>>> print(a[np.where(a > 13)])
			[21 14 19 40]

5	mean, median	Statistical functions	>>> np.mean(a)
			18.166666666666668
			>>> np.mean(a, axis = 1)
			array([14. , 22.33333333])
			>>> np.mean(a, axis = 0)
			array([20. , 23.5, 11.])
			>>> np.median(a)
			16.5
			>>> np.median(a, axis = 1)
			array([14., 19.])
			>>> np.median(a, axis = 0)
			array([20. , 23.5, 11.])
			>>> np.std(a)
			11.036555420762202
			>>> np.var(a)
			121.80555555555554

Basic Python Libraries – pandas

pandas is a Python package providing fast and flexible functionalities designed to work with "relational" or "labeled" data.

```
import pandas as pd   # "pd" is just an alias for pandas

data = pd.read_csv("auto-mpg.csv") # Uploads data from a .csv file

type(data)  # To find the type of the data set object loaded
```

pandas.core.frame.DataFrame

```
data.shape  # To find the dimensions i.e. number of rows and columns of
the data set loaded
```

(398, 9)

```
nrow_count = data.shape[0]  # To find just the number of rows
print(nrow_count)
```

398

```
ncol_count = data.shape[1]  # To find just the number of columns
print(ncol_count)
```

9

Basic Python Libraries – pandas (contd.)

```
data.columns  # To get the columns of a dataframe

Index(['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
       'acceleration', 'model year', 'origin', 'car name'],
      dtype='object')


# To change the column names of a dataframe e.g. 'mpg' in this case …

data.columns = ['miles_per_gallon', 'cylinders', 'displacement',
'horsepower', 'weight', 'acceleration', 'model year', 'origin', 'car
name']

data.columns  # To get the revised column names of the dataframe ...

Index(['miles_per_gallon', 'cylinders', ...], dtype='object')

data.rename(columns={'displacement': 'disp'}, inplace=True)
```

Basic Python Libraries – pandas (contd.)

```
data.head()  # By default displays top 5 rows

data.head(3)  # To display the top 3 rows


data.tail ()  # By default displays bottom 5 rows

data.tail (3)  # To display the bottom 3 rows

data.at[200,'cylinders']   # Will return  cell value of  the 200th row
and column 'cylinders' of the data frame
```

6

Alternatively, we can use the following code:

```
data.get_value(200,'cylinders')

data_cyl = data.loc[: , "car name"]

data_cyl.head()
```

```
0    chevrolet chevelle malibu
1            buick skylark 320
2           plymouth satellite
3                amc rebel sst
4                  ford torino
Name: car name, dtype: object
```

Basic Python Libraries – pandas (contd.)

Find missing values in a data set:

```
import numpy as np
import pandas as pd

# Creation of a data set with missing values ...
var1 = [np.nan, np.nan, np.nan, 10.1, 12, 123.14, 0.121]
var2 = [40.2, 11.78, 7801, 0.25, 34.2, np.nan, np.nan]
var3 = [1234, np.nan, 34.5, np.nan, 78.25, 14.5, np.nan]
df = pd.DataFrame({'Attr_1': var1, 'Attr_2': var2, 'Attr_3': var3})
print(df)
```

```
   Attr_1   Attr_2   Attr_3
0     NaN    40.20  1234.00
1     NaN    11.78      NaN
2     NaN  7801.00    34.50
3  10.100     0.25      NaN
4  12.000    34.20    78.25
5 123.140      NaN    14.50
6   0.121      NaN      NaN
```

```
# Find missing values in a data set
miss_val = df[df['Attr_1'].isnull()]
print(miss_val)
```

```
   Attr_1   Attr_2  Attr_3
0     NaN    40.20  1234.0
1     NaN    11.78     NaN
2     NaN  7801.00    34.5
```

Basic Python Libraries – pandas (contd.)

```
>>> np.mean(data[["mpg"]])
23.514573
>>> np.median(data[["mpg"]])
23.0
>>> np.var(data[["mpg"]])
60.936119
>>> np.std(data[["mpg"]])
7.806159
```

Basic Python Libraries – matplotlib

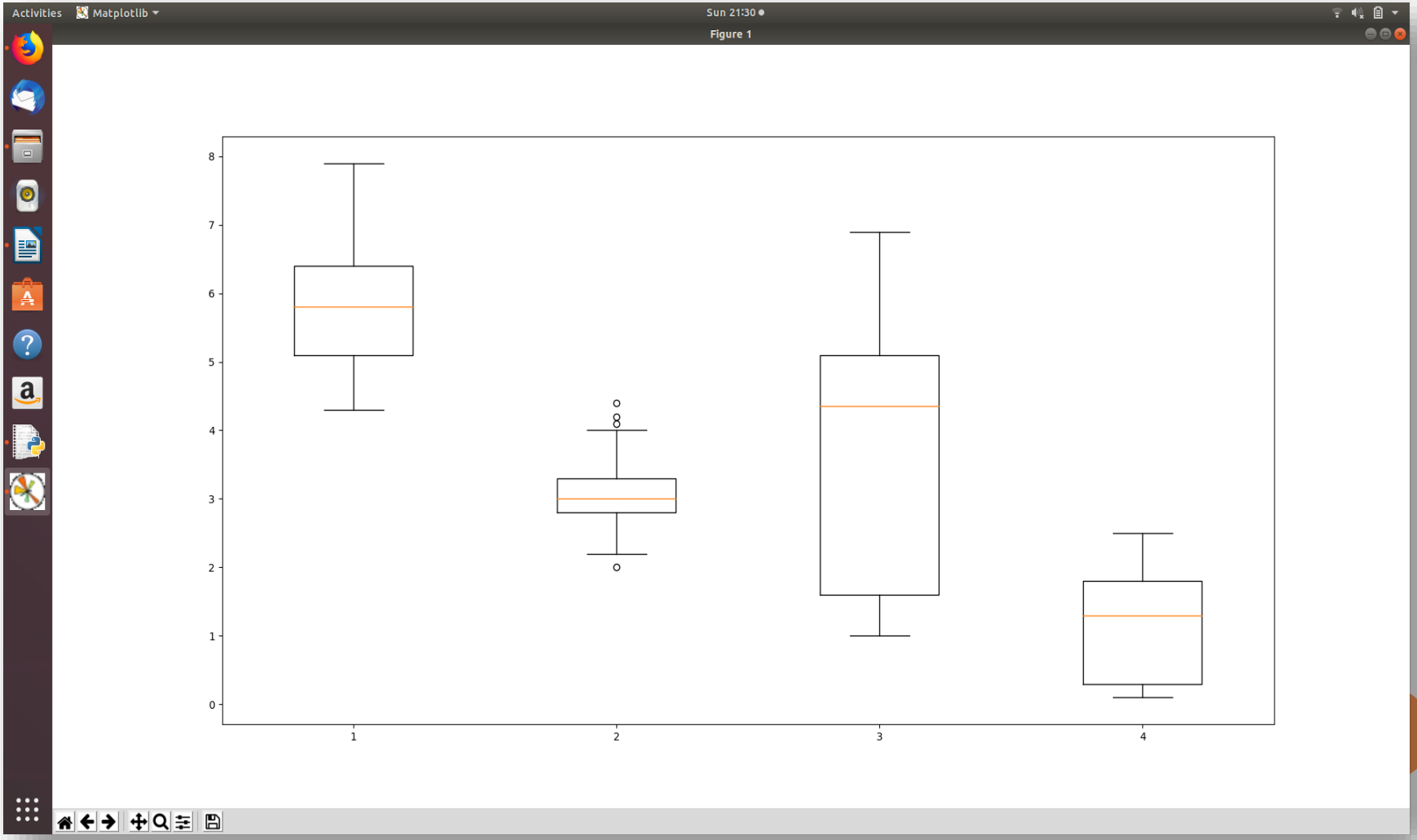
Constructing Box plot for Iris data set

- ✓ Popular data set in the machine learning
- ✓ Consists of 3 different types of iris flower - Setosa, Versicolour, and Virginica
- ✓ 4 columns - Sepal Length, Sepal Width, Petal Length and Petal Width
- ✓ First have to import the Python library ***datasets***

```
>>> from sklearn import datasets
# import some data to play with
>>> iris = datasets.load_iris()
>>> import matplotlib.pyplot as plt
>>> X = iris.data[:, :4]
>>> plt.boxplot(X)
>>> plt.show()
```

Basic Python Libraries – matplotlib (contd.)

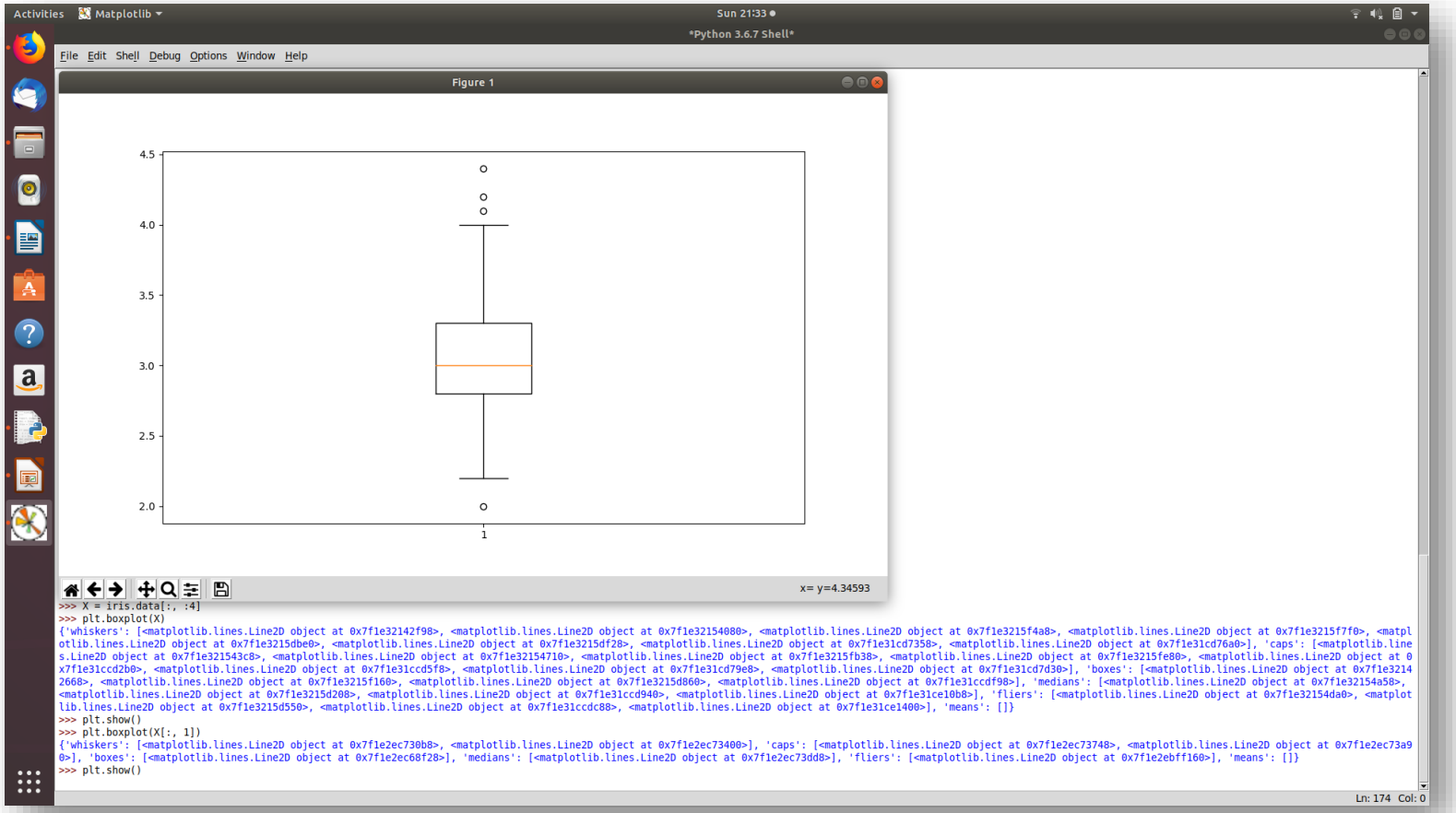
Box plot for Iris data set (all features):



Basic Python Libraries – matplotlib (contd.)

```
>>> plt.boxplot(X[:, 1])

>>> plt.show()
```

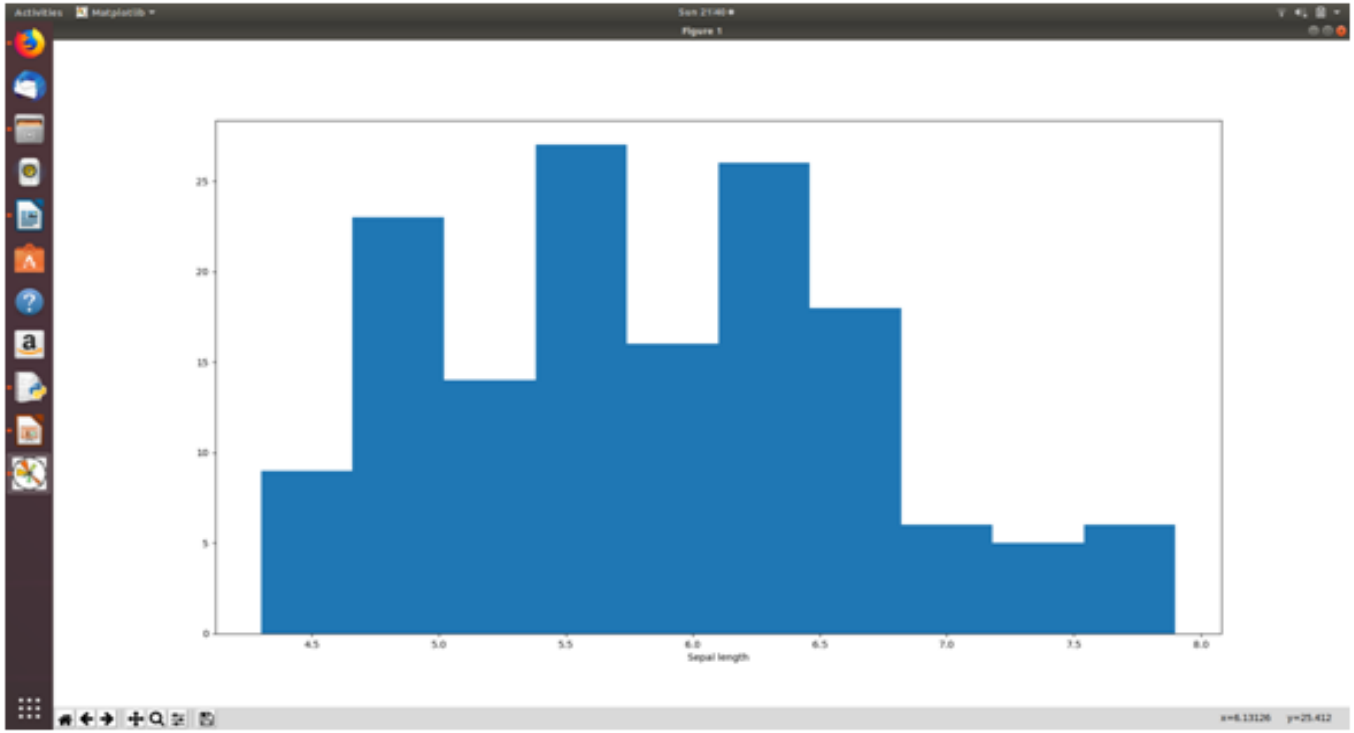


Box plot for Iris data set (single feature)

Basic Python Libraries – matplotlib (contd.)

```
>>> import matplotlib.pyplot as plt
>>> X = iris.data[:, :1]
>>> plt.hist(X)
>>> plt.xlabel('Sepal length')
>>> plt.show()
```

Histogram



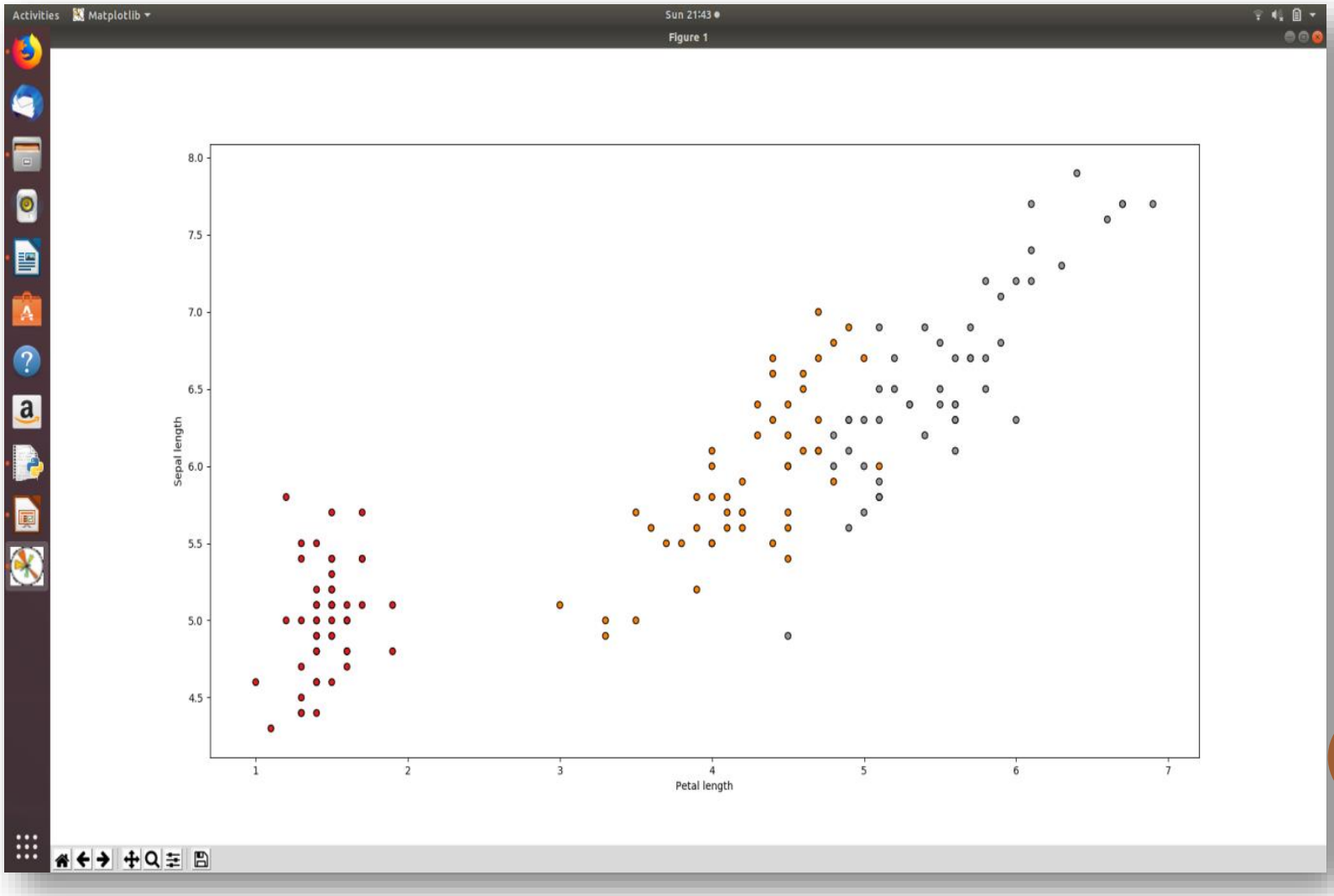
Basic Python Libraries – matplotlib (contd.)

Scatterplot of Iris data set : Sepal length vs. Petal length

```
>>> X = iris.data[:, :4]  # We take the first 4 features
>>> y = iris.target
>>> plt.scatter(X[:, 2], X[:, 0], c=y, cmap=plt.cm.Set1,
edgecolor='k')
>>> plt.xlabel('Petal length')
>>> plt.ylabel('Sepal length')
>>> plt.show()
```

Basic Python Libraries – matplotlib (contd.)

Scatterplot of Iris data set : Sepal length vs. Petal length



Data pre-processing

Mainly deals with two things –

- ❑ Handling outliers
- ❑ Remediating missing values

Primary measures for remediating outliers and missing values are:

- ✓ Removing specific rows containing outliers / missing values
- ✓ Imputing the value (i.e. outlier / missing value) with a standard statistical measure e.g. mean or median or mode for that attribute
- ✓ Estimate the value (i.e. outlier / missing value) based on value of the attribute in similar records and replace with the estimated value.
- ✓ Cap the values within 1.5 X IQR limits

Data pre-processing (contd.)

```
>>> df = pd.read_csv("auto-mpg.csv")
```

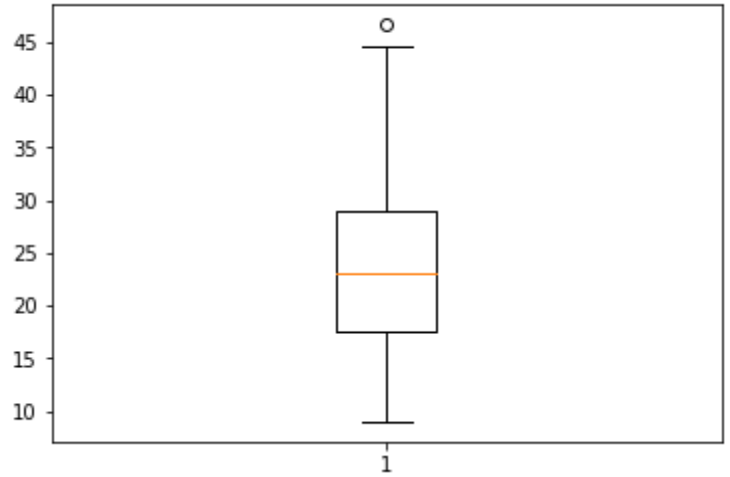
Finding missing values in a data set:

```
>>> miss_val = df[df['horsepower'].isnull()]
>>> print(miss_val)
```

Data pre-processing (contd.)

Finding Outliers (Option 1) :

```
>>> import matplotlib.pyplot as plt
>>> X = data["mpg"]
>>> plt.boxplot(X)
>>> plt.show()
```




```
>>> outliers = plt.boxplot(X[:, ])["fliers"][0].get_data()[1]
>>> outliers
array([ 46.6])
```

Data pre-processing (contd.)

Finding Outliers (Option 2) :

```
def find_outlier(ds, col):
    quart1 = ds[col].quantile(0.25)
    quart3 = ds[col].quantile(0.75)
    IQR = quart3 - quart1 #Inter-quartile range
    low_val  = quart1 - 1.5*IQR
    high_val = quart3 + 1.5*IQR
    ds = ds.loc[(ds[col] < low_val) | (ds[col] > high_val)]
    return ds
>>> outliers = find_outlier(data, "mpg")
>>> outliers
```

```
     mpg  cylinders  displacement  horsepower  weight  acceleration  \
322  46.6          4          86.0        65.0    2110          17.9

     model year  origin   car name
322          80       3  mazda glc
```

Data pre-processing (contd.)

Removing records with missing values / outliers:

We can drop the rows / columns with missing values using the code below.

```
>>> data.dropna(axis=0, how='any')
```

In a similar way, outlier values can be removed.

```
def remove_outlier(ds, col):
    quart1 = ds[col].quantile(0.25)
    quart3 = ds[col].quantile(0.75)
    IQR = quart3 - quart1 #Interquartile range
    low_val  = quart1 - 1.5*IQR
    high_val = quart3 + 1.5*IQR
    df_out = ds.loc[(ds[col] > low_val) & (ds[col] <
 high_val)]
    return df_out

>>> data = remove_outlier(data, "mpg")
```

Data pre-processing (contd.)

Imputing standard values:

Only the affected rows are identified and the value of the attribute is transformed to the mean value of the attribute.

```
>>> hp_mean = np.mean(data['horsepower'])
>>> imputedrows = data[data['horsepower'].isnull()]
>>> imputedrows = imputedrows.replace(np.nan, hp_mean)
```

Then the portion of the data set not having any missing row is kept apart.

```
>>> missval_removed_rows  =  data.dropna(subset=['horsepower'])
```

Then join back the imputed rows and the remaining part of the data set.

```
>>> data_mod = missval_removed_rows.append(imputedrows,
ignore_index=True)
```

In a similar way, outlier values can be imputed.

Thank You & Stay Tuned!