National University of Computer and Emerging Sciences



Lab Manual 05

CL461-Artificial Intelligence Lab

|  |  |
| --- | --- |
| Course Instructor | Dr. Mubasher Baig |
| Lab Instructor (s) | Usman Anwer  Mahmood Hussain |
| Section | B |
| Semester | Spring 2021 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

Table of Contents

[**Objectives**](#_heading=h.30j0zll) **3**

[**Task Distribution**](#_heading=h.1fob9te) **3**

[**Pandas**](#_heading=h.4d34og8) **3**

[3.1 Installation:](#_heading=h.yb6bfaq9y65s) 3

[3.2 Getting Started With Pandas:](#_heading=h.u8e67eqqfxgu) 3

[3.3 DataFrame:](#_heading=h.2s8eyo1) 4

[3.3.1 Defining a Dataframe:](#_heading=h.4p975l6ibl1y) 4

[3.3.2 Selecting Elements:](#_heading=h.aq80lwir8ql9) 4

[3.3.3 loc vs iloc](#_heading=h.1gka67g0mg4p) 5

[3.3.4 Dropping Rows and Columns](#_heading=h.cdlifchsqyr6) 6

[3.3.5 Filling Null Values:](#_heading=h.y24ze4lsat1s) 6

[3.4 Reading data in Files:](#_heading=h.yw7cz4wy6gyk) 7

[3.5 Writing data to Files:](#_heading=h.fxqnjhxaa09a) 7

[3.6 Data Manipulation:](#_heading=h.l7h36csat9fq) 7

[3.6.1 Merging:](#_heading=h.x9dw2ne9tlby) 7

[3.6.2 DataFrame Concatenation:](#_heading=h.7rxnib9lwx4v) 8

[3.7 Data Transformation](#_heading=h.uz1mnwbuyysg) 8

[3.7.1 Removing Duplicates:](#_heading=h.6euitsltk5l) 9

[**4. Data Visualizations:**](#_heading=h.oemcsjtmq6qq) **9**

[4.1 Matplotlib](#_heading=h.mrh7r8cp8vot) 9

[4.1.1 Installation of Matplotlib:](#_heading=h.laqro39d432r) 9

[4.1.2 Scripting Layer (pyplot)](#_heading=h.zigcsa2s9h92) 9

[4.1.3 Matplotlib and Numpy:](#_heading=h.g1mgghy2vwvj) 10

[4.1.4 Working with Multiple Figures and Axes](#_heading=h.ljtract6fcaq) 11

[4.1.4 Handling Date Values:](#_heading=h.8qk2vntmbpb) 12

[4.2 Chart Topologies:](#_heading=h.2ihezr9fd8as) 14

[4.2.1 LinePlot:](#_heading=h.ribe4if2dqwp) 15

[4.2.2 Histograms:](#_heading=h.lbsr7bhmzstx) 16

[4.2.3 Bar Plot:](#_heading=h.d2erl6gu9dsu) 17

[4.2.4 Pie Plot:](#_heading=h.r3uu5ibnz54c) 17

[**Exercise (25 marks)**](#_heading=h.v2i0jw8banbo) **18**

[5.1 Analyze Covid19 in India (10)](#_heading=h.ih2djklye1sj) 18

[5.2 Find States having most Covid19 Cases (10)](#_heading=h.6kph05ke2eni) 19

[5.3 Visualize the Higly infected States via barplot (5)](#_heading=h.oq8uj9bb8sl1) 19

[**Submission Instructions**](#_heading=h.1pxezwc) **19**

# Objectives

After performing this lab, students shall be able to understand Python data analysis and visualizations, which include:

* Python Pandas
* Python Matplotlib

# Task Distribution

|  |  |
| --- | --- |
| **Total Time** | **170 Minutes** |
| Data Frames | 30 Minutes |
| Data Manipulation and Transformation | 20 Minutes |
| Data Visualization | 20 Minutes |
| Exercise | 90 Minutes |
| Online Submission | 10 Minutes |
|  |  |

# Pandas

Pandas is the library used with Python for Data Analysis.

## 3.1 Installation:

Pandas come installed with Anaconda distributions of Jupyter, Spyder. If you are using Google Colab, then also not required to be installed.

One can install it using *conda* package.

conda install pandas

Pandas can also be installed by PyPI:

pip install pandas

## 3.2 Getting Started With Pandas:

To use the pandas, import the pandas first, where *pd* is the alias to pandas :

import pandas as pd

## 3.3 DataFrame:

Dataframe is the primary data structure on which all transactions, which are generally

made during the analysis of data, are centralized. In fact, the DataFrame consists of an ordered collection of columns each of which can contain a value of different type (numeric, string, Boolean, etc.). Dataframe has two indexed arrays, The first, associated with the lines, The second array instead contains a series of labels, each associated with a particular column.

### 3.3.1 Defining a Dataframe:

The most common way to create a new DataFrame is precisely to pass a dict object to the DataFrame() constructor. This dict object contains a key for each column that we want to define, with an array of values for each of them.

data = {'color' : ['blue','green','yellow','red','white'],

'object' : ['ball','pen','pencil','paper','mug'],

'price' : [1.2,1.0,0.6,0.9,1.7]}

Second, pass this dictionary object to DataFrame() constructors.

import pandas as pd

frame = pd.DataFrame(data)

To get first 5 entries of dataframe, head() function is to be used of DataFrame

frame.head()

### 3.3.2 Selecting Elements:

To get all the column names of a DataFrame, *columns* attribute of DataFrame is used.

frame.columns

To select the contents of a column, the label of the column can be used as an index.

frame['color']

To select the multiple columns, an array of column names can be used.

frame[['object','price']]

To select the specified range of rows, *Slicing* can be used.

frame[1:] # select all the rows from index 1

frame[0:2] # select all the rows upto index 2

frame[:4] # select all the rows upto index 3

frame[:-2] # select all the rows rather then last two

frame[-2:] # select last two rows

### 

### 3.3.3 loc vs iloc

loc and iloc are the two functions of a DataFrame which are widely used to select the data.

**loc** is label-based, which means that we have to specify the name of the rows and columns that we need to filter out.

**iloc** is integer index-based. So here, we have to specify rows and columns by their integer index.

To select the first three rows of ‘color’ and ‘object’ columns, we may use loc. loc allows selection of both rows and columns in a statement, but column selection is label-based.

frame.loc[0:2,['color','object']]

To select the rows based on a condition, to make a condition (conditional operators >, <,>=,<=, !=,==) are used. To acquire the resulting rows based on condition, we can use *loc*.

frame.loc[frame['price']>=1]

# 

*loc* can return results based on multiple conditions.

# 

frame.loc[(frame['price']>=1) & (frame['price']<=1.5)]

Furthermore, to select the specified set of columns instead of whole, based on a condition we can also use *loc.* In this example, we selected only columns ‘object’ and ‘color’ when the price is more than 1 dollar.

frame.loc[frame['price']>1, ['object','color']]

On the other hand, *loc* is indexed based selection, we can select both rows and columns based on their index. This example selects the first two occurrences of both rows and columns from DataFrame.

frame.iloc[0:2,0:2]

### 3.3.4 Dropping Rows and Columns

DataFrames has a *drop* function, which can be used to drop both rows and columns. To drop the rows, we can use the indexof that particular row.

#drop a row with index 1

frame.drop(1)

#drop rows with index 1 and 2

frame.drop([1,2])

To drop columns, *axis* parameter is to be used. for pandas, axis=1 is used to select columns.

# drop one column

frame.drop(['color'],axis=1)

# drop multiple columns

frame.drop(['color','object'],axis=1)

### 3.3.5 Filling Null Values:

Missing data is one of the important factors in data science. A data file can have a lot of missing values. To apply machine learning algorithms on data , it is mandatory to remove the missing values from data first. Pandas equip DataFrame with enough functions to deal with such null values.

To Understand Null values, it is required to have a DataFrame first having null values init.

# numpy is used to get null values

import numpy as np

frame3 = pd.DataFrame([[6,np.nan,6],[np.nan,np.nan,np.nan],[2,np.nan,5]],

index = ['blue','green','red'],

columns = ['ball','mug','pen'])

To fill all the null values with 0, we can use *fillna* function.

frame3.fillna(0)

To replace values differently following columns, *fillna* function also supports column-wise data assignment.

frame3.fillna({'ball':1,'mug':0,'pen':99})

## 3.4 Reading data in Files:

From common experience, the most common operation for a person approaching data analysis is to read the data contained in a file, the file can be a CSV, html , excel file or at least in a text file.

To read the data from the csv file, *read\_csv* function is to be used.

*read\_csv* function reads the data from a csv file, and returns the content in DataFrame format.

import pandas as pd

#read csv file

df=pd.read\_csv('Iris.csv')

#read html file

web\_frames = pd.read\_html('myFrame.html')

#read excel file

excel\_data=pd.read\_excel('data.xls')

it is also possible to directly assign a list of column names while reading the csv file.

df=pd.read\_csv('ch05\_02.csv', names=['white','red','blue','green','animal'])

## 3.5 Writing data to Files:

DataFrame can be written into CSV, Excel, HTML or Xml file format. To save a DataFrame into CSV file format, use *to\_csv* function.

For example,

frame3.to\_csv('ch05\_07.csv')

## 3.6 Data Manipulation:

Once having data in DataFrame format, they are ready to be manipulated. The manipulation of the data has the purpose of preparing the data so that they can be more easily subjected to analysis.

### 3.6.1 Merging:

The merging operation, which corresponds to the JOIN operation for those who are familiar with SQL, consists of a combination of data through the connection of rows using one or more keys. During merging, rows having the same index in both DataFrames are combined.

Lets say, we have two dataframes, where row indexes are Strings,

#import the libraries

import pandas as pd

#dataframe one

frame1 = pd.DataFrame( {'id':['ball','pencil','pen','mug','ashtray'],

'price': [12.33,11.44,33.21,13.23,33.62]})

#dataframe two

frame2 = pd.DataFrame( {'id':['pencil','pencil','ball','pen'],

'color': ['white','red','red','black']})

To merge both DataFrames , *merge* function of Pandas is used.

#dataframes merging

pd.merge(frame1,frame2)

The *merge* function combines both DataFrames to one, This merge is column-wise where indexes are similar.

### **3.6.2 DataFrame Concatenation:**

Concatenation is a bit different from merging, while concatenation one can decide either to combine both DataFrames either by rows or columns.

In the first example, *concat* function of pandas is used to merge row-wise. By default the function combines both DataFrames by rows, Meaning the contents of frame2 will be appearing underneath of frame1, given column names are the same. For example,

pd.concat([frame1,frame2])

In the second example, *concat* function is used to combine both DataFrames column-wise. The contents of frame2 will be appearing to the right of frame1. For column-wise concatenation, *axis=1* is used.

pd.concat([frame1,frame2],axis=1)

## 3.7 Data Transformation

The second stage of data manipulation is called Data Transformation. After acquiring the appropriate data into DataFrame it is important to transform their values. First of its type operation is called Removing Duplicates.

### 3.7.1 Removing Duplicates:

First of all, let’s create a DataFrame.

dframe = pd.DataFrame({ 'color': ['white','white','red','red','white'],

'value': [2,1,3,3,2]})

The DataFrame has two columns, color and value. Both have duplicate values. First of all to check the existence of duplicate values, either present in DataFrame or not. we can use *duplicated* function.

dframe[dframe.duplicated()]

and to remove these duplicate values from DataFrame, *drop\_duplicates* function of DataFrame is to be used.

dframe.drop\_duplicates()

Pandas is a library made of data analysis in python. It is widely used in industry by data scientists to analyse the data before applying machine learning.

# 4. Data Visualizations:

Data visualization is an aspect too often underestimated in data analysis, but it is actually a very

important factor because an incorrect or inefficient data representation can ruin an

otherwise-excellent analysis.

## 4.1 Matplotlib

Matplotlib is a Python library specializing in the development of two-dimensional charts

(including 3D charts); in recent years, it has been widespread in scientific and engineering circles.

### 4.1.1 Installation of Matplotlib:

Installation of Matplotlib is quite simple, it is not required when one is using Jupyter, Spyder distributions of Anaconda. In case anyone is using Standalone Python, Conda distribution of linux can also be used to install it.

conda install matplotlib

### 4.1.2 Scripting Layer (pyplot)

This layer consists of an interface called pyplot. Which is used with Python to interface matplotlib. To import the matplotlib, following statement is to be used in Python

import matplotlib.pyplot as plt

To draw a simple plot in matplotlib, *plot* function is used. A simple plot is given,

plt.plot([1,2,3,4],[1,4,9,16])

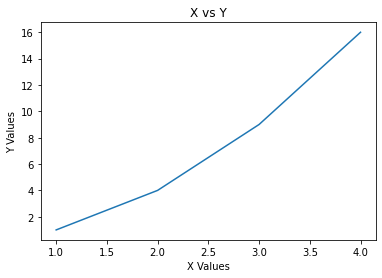
plt.xlabel('X Values')

plt.ylabel('Y Values')

plt.title('X vs Y')

To draw a simple plot, x and y values have to be specified.

The resultant graph is given:



### 4.1.3 Matplotlib and Numpy:

Even the Matplot library, despite being a fully graphical library, has its foundation the NumPy library. In fact, you have seen so far how to pass lists as arguments, both to represent the data and to set the extremes of the axes. Actually, these lists have been converted internally in NumPy arrays.

Therefore, you can directly enter NumPy arrays as input data. This array of data, which have been processed by pandas, can be directly used with matplotlib without further processing.

import math

import numpy as np

# generates an array containing 10 points

x = np.arange(1,11)

y = 2 \* x + 5

plt.title("Matplotlib demo")

plt.xlabel("x axis caption")

plt.ylabel("y axis caption")

plt.plot(x,y,"ob")

plt.show()

### 4.1.4 Working with Multiple Figures and Axes

While working with pyplot, one can divide the figure into multiple subplots. *Subplot*  function is to be used for this purpose. The argument

of the subplot() function is composed of three integers. The first number defines how many parts the figure

is split into vertically. The second number defines how many parts the figure is divided into horizontally.

The third issue selects which is the current subplot on which you can direct commands.

Now you will display two sinusoidal trends (sine and cosine) and the best way to do that is to divide the canvas vertically in two horizontal subplot. So the numbers to pass as an argument are ‘211’ and ‘212’.

t = np.arange(0,5,0.1)

y1 = np.sin(2\*np.pi\*t)

y2 = np.sin(2\*np.pi\*t)

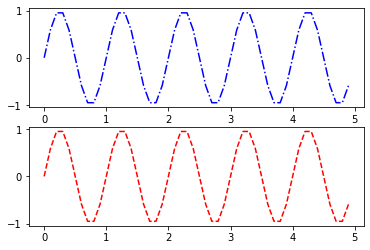
plt.subplot(211)

plt.plot(t,y1,'b-.')

plt.subplot(212)

plt.plot(t,y2,'r--')

The resultant image is:



### 4.1.4 Handling Date Values:

One of the most common problems encountered when doing data analysis is handling data of the

date-time type. Displaying them along an axis (normally the x axis) can be really problematic especially for the management of ticks.

Take for example the display of a linear chart with a data set of eight points in which you have to

represent date values on the x axis with the following format: day-month-year.

import datetime

import numpy as np

import matplotlib.pyplot as plt

#making list of DataTime values

events = [datetime.date(2015,1,23),datetime.date(2015,1,28),datetime.

date(2015,2,3),datetime.date(2015,2,21),datetime.date(2015,3,15),datetime.

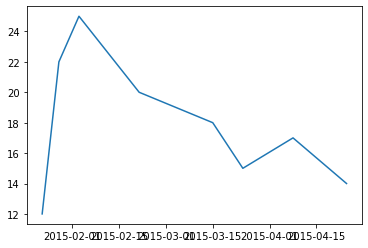
date(2015,3,24),datetime.date(2015,4,8),datetime.date(2015,4,24)]

#y list

readings = [12,22,25,20,18,15,17,14]

plt.plot(events,readings)

The resultant Image for the graph is:



To manage the dates therefore it is advisable to define a time scale with appropriate objects. First you need to import matplotlib.dates, a module specialized for the management of this type of data. Then you define the scales of the times, as in this case, a scale of days and one of the months, through the functions MonthLocator() and DayLocator(). In these cases, the formatting is also very important, and to avoid overlap or unnecessary references, you have to limit the tick labels to the essential, which in this case is year-month. This format can be passed as an argument to DateFormatter() function.

After you defined the two scales, one for the days and one for the months, you can set two different kinds of ticks on the x axis, using the functions set\_major\_locator() and set\_minor\_locator() on the x-axis object. Instead, to set the text format of the tick labels referred to the months you have to use the set\_major\_ formatter() function.

import datetime

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.dates as mdates

months = mdates.MonthLocator()

# getting object of DayLocator

days = mdates.DayLocator()

# Date formatting

timeFmt = mdates.DateFormatter('%Y-%m')

# Generating time series data (xaxis)

events = [datetime.date(2015,1,23),datetime.date(2015,1,28),datetime.

date(2015,2,3),datetime.date(2015,2,21),datetime.date(2015,3,15),datetime.

date(2015,3,24),datetime.date(2015,4,8),datetime.date(2015,4,24)]

#getting y axis data

readings = [12,22,25,20,18,15,17,14]

fig, ax = plt.subplots()

plt.plot(events,readings)

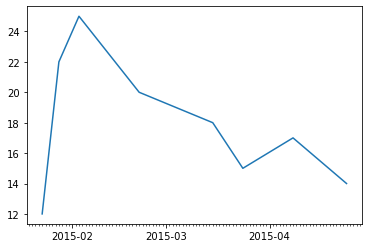
#Adjusting the dates

ax.xaxis.set\_major\_locator(months)

ax.xaxis.set\_major\_formatter(timeFmt)

ax.xaxis.set\_minor\_locator(days)

The Output is:



## 4.2 Chart Topologies:

When it comes to the variety of plots to work with, Matplotlib is one of the most popular library to get hands-on. It has many varieties of graphs such as

1. Line Plot
2. Histogram
3. BarPlot
4. Pie Plot

Lets create an understanding level with them.

### 

### 4.2.1 LinePlot:

Among all the types of chart the linear chart is the simplest. A line chart is a sequence of data points connected by a line. Each data point consists of a pair of values (x,y), which will be reported in the chart according to the scale of values of the two axes (x and y).

A simple Example is to draw a sinusidal wave, the data is generated through *numpy*:

import matplotlib.pyplot as plt

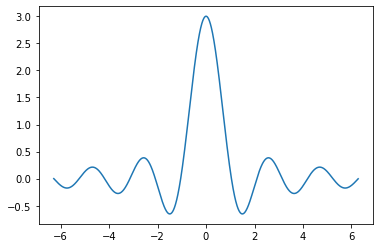
import numpy as np

x = np.arange(-2\*np.pi,2\*np.pi,0.01)

y = np.sin(3\*x)/x

plt.plot(x,y)

The Resultant Image is:



It is also possible to draw multiple graphs in a figure. Lets say, we generated three different y sequences through numpy.

import matplotlib.pyplot as plt

import numpy as np

x = np.arange(-2\*np.pi,2\*np.pi,0.01)

y = np.sin(4\*x)/x

y2 = np.sin(2\*x)/x

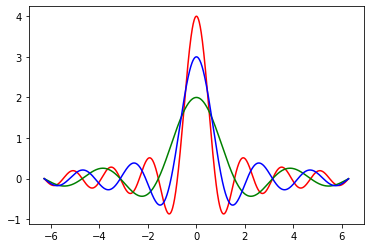
y3 = np.sin(3\*x)/x

plt.plot(x,y,'r')

plt.plot(x,y2,'g')

plt.plot(x,y3,'b')

The Resultant Image is:



### 4.2.2 Histograms:

A histogram consists of adjacent rectangles erected on the x axis, split into discrete intervals called bins, and with an area proportional to the frequency of the occurrences for that bin. This kind of visualization is commonly used in statistical studies about distribution of samples.

To create a simple histogram we incorporated numpy to generate random data.

import matplotlib.pyplot as plt

import numpy as np

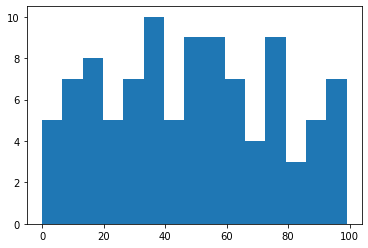
# random data array are created using numpy,

# the values count is 100 ranging from 0-100

pop = np.random.randint(0,100,100)

plt.hist(pop,bins=15)

The output is:



### 4.2.3 Bar Plot:

Another very common type of chart is the bar chart. It is very similar to a histogram but in this case the x axis is not used to reference numerical values but categories. The realization of the bar chart is very simple with matplotlib using the bar() function.

import matplotlib.pyplot as plt

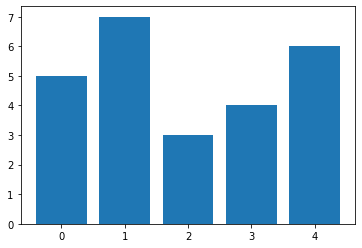
index = [0,1,2,3,4]

values = [5,7,3,4,6]

# bar plot

plt.bar(index,values)

The output is:



### 4.2.4 Pie Plot:

An alternative way to display data to the bar charts is the pie chart, easily obtainable using the pie() function.

A simple example is:

import matplotlib.pyplot as plt

# three arrays are defined

# for labels

labels = ['Nokia','Samsung','Apple','Lumia']

# for values incorporated percentage

values = [10,30,45,15]

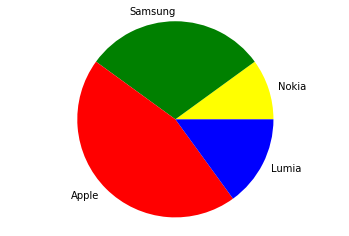
# array contain color values

colors = ['yellow','green','red','blue']

#plot

plt.pie(values,labels=labels,colors=colors)

plt.axis('equal')



# Exercise (25 marks)

## 5.1 Analyze Covid19 in India (10)

Find the total number of confirmed cases, deaths and recovered covid19 patients in Indian States Kerala, Odisha, Rajasthan and Tamil Nadu from April 27,2020 to May 27,2020. (data file is attached).

Your output should be similar to this table.

|  |  |  |  |
| --- | --- | --- | --- |
| State | Confirmed | Recovered | Deaths |
| Kerala | 27000 | 24400 | 2000 |
| Odisha | #### | #### | #### |
| Rajasthan | #### | #### | #### |
| Tamil Nadu | #### | #### | #### |

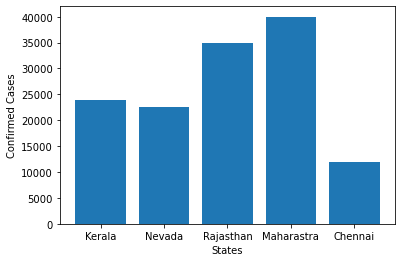
## 5.2 Find States having most Covid19 Cases (10)

Find the top 5 states of india, having highest number of covid19 confirmed cases. List the states, along number of confirmed cases in descending order.

## 5.3 Visualize the Higly infected States via barplot (5)

Visualize the result got in 5.2, via barplot.

Output should be similar to this.



# Submission Instructions

Always read the submission instructions carefully.

* Rename your Jupyter notebook to your roll number and download the notebook as **.ipynb** extension.
* To download the required file, go to **File->Download .ipynb**
* Only submit the **.ipynb** file. DO NOT **zip** or **rar** your submission file
* Submit this file on Google Classroom under the relevant assignment.
* Late submissions will not be accepted
* Data File for Exercise is attached.