Certificate Course in Machine Learning using Python [6 Weeks]

Data Exploration and Visualization using matplotlib and seaborn modules

Attempt: 1

Matplotlib

Matplotlib is one of the most popular Python packages used for data visualization. It is a cross-platform library for making 2D plots from data in arrays. Matplotlib is written in Python and makes use of NumPy.

There are various plots which can be created using python matplotlib. Some of them are listed below:

Installing Matplotlib

Open command prompt and type following command to install Matplotlib on your computer:

pip3 install matplotlib

t is preinstalled on Anaconda distribution.

General Concepts

A Matplotlib figure can be categorized into several parts as below:

Figure: It is a whole figure which may contain one or more than one axes (plots). You can think of a **Figure** as a canvas which contains plots.

Axes: It is what we generally think of as a plot. A **Figure** can contain many Axes. It contains two or three (in the case of 3D) **Axis** objects. Each Axes has a title, an x-label and a y-label.

Axis: They are the number line like objects and take care of generating the graph limits.

Pyplot

Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Built-in Functions of Matplotlib.pyplot

plot():

The **plot() function** in pyplot module of matplotlib library is used to make a 2D hexagonal binning plot of points x, y.

show():

The **show() function** in pyplot module of matplotlib library is used to display all figures.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([10,20,30,40])
plt.show()
```

Naming Title, X-Label and Y-Label

title(), xlabel() and ylabel() methods are used to specify title, x-label and y-label of the graph.

```
import matplotlib.pyplot as plt
import numpy as np
plt.plot([10,20,30,40])
plt.title("Sample Graph")
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.show()
```

Matplotlib Functions

Function	Description
plot( )	Plot y versus x as lines and/or markers.
xlabel( )	Set the label for the x-axis.
ylabel( )	Set the label for the y-axis.
show( )	Display a figure.
axis( )	Convenience method to get or set some axis properties.
scatter( )	A scatter plot of y vs x with varying marker size and/or color.
subplot( )	Add a subplot to the current figure.
bar( )	Make a bar plot.
suptitle( )	Add a centered title to the figure.
close( )	Close a figure window.

Formatting the style of your plot

1. For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot.
2. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string.
3. The default format string is 'b-', which is a solid blue line.
4. For example, to plot with red circles, we write

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

Note: The axis( ) command takes a list of [xmin, xmax, ymin, ymax]

character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Line Style

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker

'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker

's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker

Formatting the style of your plot

```
plt.plot( [1, 2, 3, 4], [1, 4, 9, 16], 'ro' )
plt.axis( [0, 6, 0, 20])
plt.show( )
```

Plotting with keyword strings

There are some instances where you have data in a format that lets you access particular variables with strings. For example, with **numpy.recarray** or pandas.DataFrame.

Matplotlib allows you provide such an object with the data keyword argument. If provided, then you may generate plots with the strings corresponding to these variables.

Example

```
import pandas as pd
import matplotlib.pyplot as plt
Student1={ 'Monthly': [ 'Feb' , 'Apr' , 'June', 'Sep', 'Nov', 'Dec'],
    'Eng' : [45,67,78,58,87,89],
    'Maths': [55,87,98,88,97,69]
    }
df1=pd.DataFrame(Student1)
df1['Total']= df1['Eng']+df1['Maths']
df1['PCT']=df1['Total']/2
plt.scatter('Monthly','PCT', s=50,color='r' , data=df1)
plt.xlabel('Monthly Exam')
plt.ylabel('Percentage')
plt.title('Compare Percentage of Two Students')
plt.show( )
```

Plotting with categorical variables

It is also possible to create a plot using categorical variables. Matplotlib allows you to pass categorical variables directly to many plotting functions. For example:

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]
plt.figure(figsize=(9, 3))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show( )
```

Controlling line properties

Lines have many attributes that you can set: linewidth, dash style, antialiased, etc.

There are several ways to set line properties.

1. Use **keyword args**:

```
plt.plot(x, y, linewidth=2.0)
```

2. Use **the setter methods** of a Line2D instance. plot returns a list of Line2D objects;

```
e.g., line1= plt.plot(x1, y1).
```

Use the setp( ) command.

1. The example below uses a MATLAB-style command to set multiple properties on a list of lines.
2. **setp** works transparently with a list of objects or a single object. You can either use python keyword arguments or MATLAB-style string/value pairs:

```
lines = plt.plot(x1, y1)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)
# or MATLAB style string value pairs
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

Scatter Plot

1. This type of plot shows all individual data points. Here, they aren't connected with lines.
2. Each data point has the value of the x-axis value and the value from the y-axis values.
3. This type of plot can be used to display trends or correlations.
4. In data science, it shows how 2 variables compare.
5. To make a scatter plot with Matplotlib, we can use the plt.scatter( ) function.
6. Again, the first argument is used for the data on the horizontal axis, and the second - for the vertical axis.

Example:

```
import numpy as np
import matplotlib.pyplot as plt
# Create data
Marks = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
Marks_Range=[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
colors = 'r'
circle_area = 25
contrast=0.9   # has value from 0 - 1
# Plot
plt.scatter(Marks,Marks_Range, s=circle_area, c=colors, alpha=contrast)
plt.title('Scatter plot - Marks ')
plt.xlabel('Marks Range')
plt.ylabel('Marks Obtained')
plt.show( )
```

Scatter plot with groups

Data can be classified in several groups. The code below demonstrates that:

```
import numpy as np
import matplotlib.pyplot as plt
# Create data
g1 = np.array( [ [89, 90, 70, 89, 100, 80, 90, 100, 80, 34],
      [1,2,3,4,5,6,7,8,9,10]] )
g2 = np.array([ [30, 29, 49, 48, 100, 48, 38, 45, 20, 30],
      [1,2,3,4,5,6,7,8,9,10]])
g3 = np.array([ [ 69, 90, 70, 59, 70, 80, 40, 80, 55, 24],
```

```
        [1,2,3,4,5,6,7,8,9,10]])
data = (g1, g2, g3)
colors = ("red", "green", "blue")
groups = ("Red House", "Green House", "Blue House")
# Create plot
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
x, y = g1
ax.scatter(y, x, alpha=0.8, c='red', edgecolors='none', s=30, label='Red House')
x, y = g2
ax.scatter(y, x, alpha=0.8, c='green', edgecolors='none', s=30, label='Green House')
x, y = g3
ax.scatter(y, x, alpha=0.8, c='blue', edgecolors='none', s=30, label='Blue House')
#for data, color, group in zip(data, colors, groups):
#  x, y = data
#  ax.scatter(y, x, alpha=0.8, c=color, edgecolors='none', s=30, label=group)
plt.title('Matplot scatter plot')
plt.legend(loc=2)
plt.show()
```

Bar Chart

Bar chart represents categorical data with rectangular bars. Each bar has a height corresponds to the value it represents.

To make a bar chart with Maplotlib, we'll need the **plt.bar()** function.

```
import numpy as np
import matplotlib.pyplot as plt
subjects= ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(subjects))
performance = [10,8,6,4,2,1]
plt.grid(color='y', linestyle='--', linewidth=2, axis='y', alpha=0.7)
plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, subjects)
plt.ylabel('Usage')
```

```
plt.title('Programming language usage')
plt.show()
```

Multiple Bar Chart

```
import numpy as np
import matplotlib.pyplot as plt
data1 = [23,85, 72, 43, 52]
data2 = [42, 35, 21, 16, 9]
width =0.3
plt.bar(np.arange(len(data1)), data1, color='r',width=width)
plt.bar(np.arange(len(data2))+ width, data2, color='b', width=width)
plt.show()
```

Stack Bar Chart

```
import numpy as np
import matplotlib.pyplot as plt
data1 = [23,85, 72, 43, 52]
data2 = [42, 35, 21, 16, 9]
plt.bar(range(len(data1)), data1)
plt.bar(range(len(data2)), data2, bottom=data1)
plt.show()
```

You can compare two data series using this Matplotlib code:

```
import numpy as np
import matplotlib.pyplot as plt
# data to plot
n_groups = 4
Sales_Voltas = (90, 55, 40, 65)
Sales_Samsung = (85, 62, 54, 20)
Year_Sales= ('2016', '2017', '2018', '2020')
# create plot
y_pos= np.arange(n_groups)
bar_width = 0.35
opacity = 0.8
rects1 = plt.bar(y_pos , Sales_Voltas , bar_width, alpha=opacity, color='b', label='Voltas')
```

```
rects2 = plt.bar(y_pos + bar_width, Sales_Samsung ,  bar_width, alpha=opacity, color='g', label='Samsung')

plt.xlabel('Year')

plt.ylabel('Sales')

plt.title('Sales Scores')

plt.xticks(y_pos + bar_width,  Year_Sales)

plt.legend()

plt.show()
```

Pie chart

Pie chart: a circular plot, divided into slices to show numerical proportion. They are widely used in the business world.

```
import matplotlib.pyplot as plt

labels = 'Python', 'C++', 'Ruby', 'Java'

sizes = [215, 130, 245, 210]

colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']

explode = (0.1, 0, 0, 0)  # explode 1st slice

plt.pie(sizes, explode=explode, labels=labels, colors=colors,

autopct='%1.1f%%', shadow=True, startangle=140)

plt.axis('equal')

plt.show()
```

To add a legend use the plt.legend() function:

Legend Location

- best
- upper right
- upper left
- lower left
- lower right
- right
- center left
- center right
- lower center
- upper center

- center

```
import matplotlib.pyplot as plt
labels = ['Cookies', 'Jellybean', 'Milkshake', 'Cheesecake']
sizes = [38.4, 40.6, 20.7, 10.3]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
patches, texts = plt.pie(sizes, colors=colors, shadow=True, startangle=90)
plt.legend(patches, labels, loc="best")
plt.axis('equal')
plt.tight_layout()
plt.show()
```

Next

PREVIOUS ACTIVITY
◄ Python code demo

Jump to...

NEXT ACTIVITY
Data Visualization: Python Code ►

Stay in touch

Contact Us

http://nielit.gov.in/gorakhpur/
abhinav@nielit.gov.in or ajay.verma@nielit.gov.in