National Forensic Sciences University

Knowledge | Wisdom | Fulfilment

School of Cyber Security & Digital Forensics

"Artificial intelligence"

"Practical"

Deepanshu Patel

01230300006002014

Basic python

Program 1

```
v1 = [1,2,3,4]
v2 = [2,3,4,5]
v3 = v1 + v2
print(v3)
print(type(v3))
```

Program 2

```
import numpy as np
a = np.array([1,2,3,4])
print(a)
print(type(a))
b = np.array([2,3,4,5])
print(b)
print(type(b))
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

Program 2.1

```
import numpy as np
a = np.array([1,"a",1.5])
print(a)
print(type(a))
b = np.array([2,"b",2.4])
print(b)
print(type(b))
# print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

Program 3

```
import numpy as np
a = np.array([[1],[2],[3]])
print(type(a))
print(a)
print(a.shape)
```

Program 4

```
import numpy as np
a = np.array([[1,2,3],[4,5,6],[7,8,9]])
print(type(a))
print(a)
print(a.shape)
```

Program 5

```
import numpy as np
a = np.array ([[1,2,3],[4,5,6],[7,8,9]])
print(type(a))
print(a)
print(a.shape)
b = np.mat(a)
#b = np.matrix ([[1,2,3],[4,5,6],[7,8,9]])
print(type(b))
print(b)
print(b.shape)
```

Program 6

```
import numpy as np
a = np.matrix([[1,2,3],[4,5,6],[7,8,9]])
print(type(a))
print(a)
```

Program 7

```
import numpy as np
a=np.array([1,2,3,4,5,6,7,8,9,0])
print(a)
print("after slicing:")
b=a[0:7:3]
print(b)
b=a[1:3:1]
print(b)
```

Program 8

```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(a)
print("After Slicing:")
print("everything")
```

```
b=a[:]
print(b)
print("everything till 1st row")
b=a[:1]
print(b)
print("everything execpt 1st row")
b=a[1:]
print(b)
print("[row,column] all row till 1st column")
b=a[:,:1]
print(b)
print("all row except 1st column")
b=a[:,1:]
print(b)
print("all row & last column")
b=a[:,-1:]
print(b)
```

Program 9

```
import numpy as np
a = np.arange(6).reshape((3, 2))
print(a)
print(a.shape)
```

Program 10

```
import numpy as np
a=np.zeros(3)
print(a)
a=np.ones(3)
print(a)
a=np.random.rand(3)
print(a)
a=np.eye(3)
print(a)
```

Program 11

```
import numpy as np
a = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
b = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a+b)
print(a-b)
print(a/b)
print(a*b)
```

Program 12

```
import numpy as np
a = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a.max())
print(a.min())
print(a.prod())
print(a.mean())
print(a.std())
print(a.var())
print(a.transpose())
```

Program 13

```
import pandas as pd
import numpy as np
a=pd.Series([1,2,3,4])
print(a)
b=pd.Series([1,2.5,3,4])
print(b)
c=pd.date_range("20230801",periods=8)
print(c)
```

Program 14

```
import numpy as np
import pandas as pd
a=np.array([1,2,3,4,5])
b=pd.Series(a)
print(b)
```

Program 15

```
import numpy as np
import pandas as pd
v1 = np.matrix([[1,2,3],[4,5,6]])
a=pd.DataFrame(v1,columns=list("ABC"))
print(a)
```

Unit-1

Program 1

```
d = {'x': 10 , 'y': 2.5, 'z' : 'test'}
print(d)
print(d['x'])
print(d.values())
```

Program 2

```
a = int ( input("Enter a number:"))
if a>0:
   print("Postive")
elif a==o:
   print("zero")
else:
   print("negative")
```

Program 3

```
for i in range (1,11):
   print(i)
```

Program 4

```
for x in 'NFSU':
   print(x)
```

Program 5

```
for x in range(1,10):
   if x%2 == 0:
      continue
   print(x)
```

Program 6

```
for x in range (1,20):
   if x ==15:
      break
   print(x)
```

Program 7

```
def my_print():
   print("hello from function")
print("this is before fuction call")
my_print()
print("this is after function call")
```

Program 8

```
def my_funtion(nm):
   print("hello + nm")
my_funtion("NFSU")
```

Program 9

```
def my_funtion(nm="MTCS"):
   print("Hello" + nm)
my_funtion("NFSU")
my_funtion()
```

Program 10

```
def my_funtion(*nm):
   for i in range(0,len(nm)):
      print("Hello "+ nm[i])
my_funtion("NFSU", "Gandhinagar", "MTCS")
```

Program 11

```
def my_cities(c):
   for item in c:
      print(item)
cities = ["ahmedabad", "delhi", "mumbai"]
my_cities(cities)
```

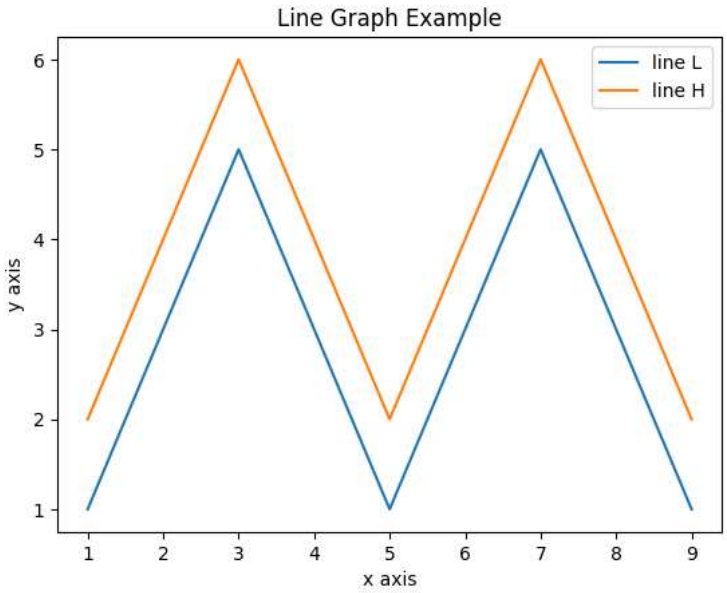
Program 12

```
def my_square(n):
   return n*n
ans = my_square(10)
print(ans)
```

DataVisualization_Ex1

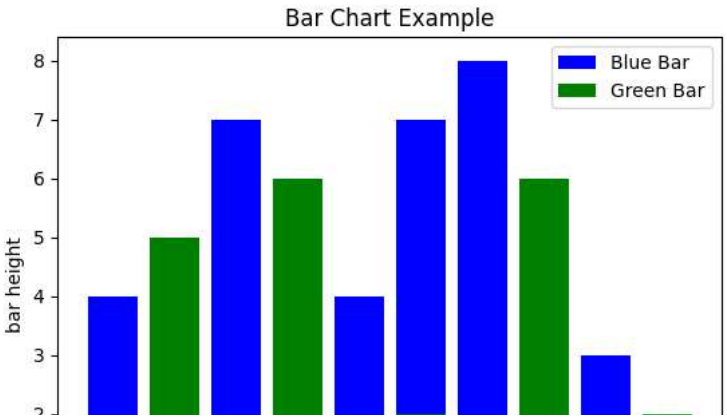
Line Plots

```
import matplotlib.pyplot as plt

x  = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y1 = [1, 3, 5, 3, 1, 3, 5, 3, 1]
y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]
plt.plot(x, y1, label="line L")
plt.plot(x, y2, label="line H")
plt.plot()

plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("Line Graph Example")
plt.legend()
plt.show()
```



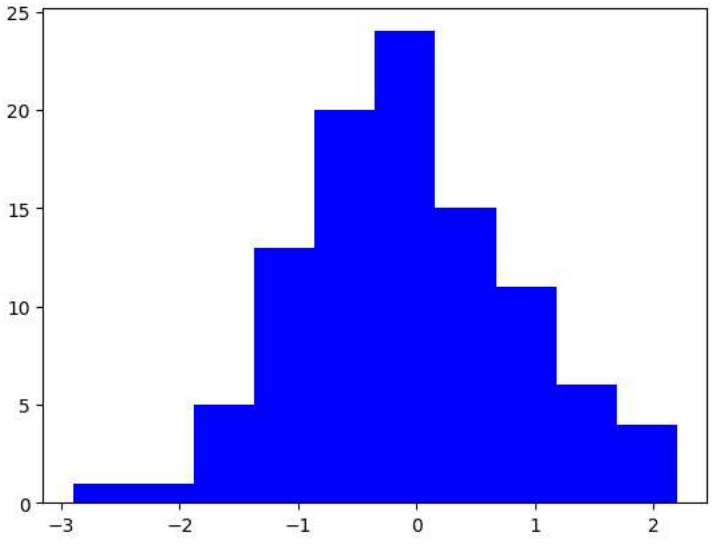
Bar Plots

```
import matplotlib.pyplot as plt

# Look at index 4 and 6, which demonstrate overlapping cases.
x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

# Colors: https://matplotlib.org/api/colors_api.html

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```



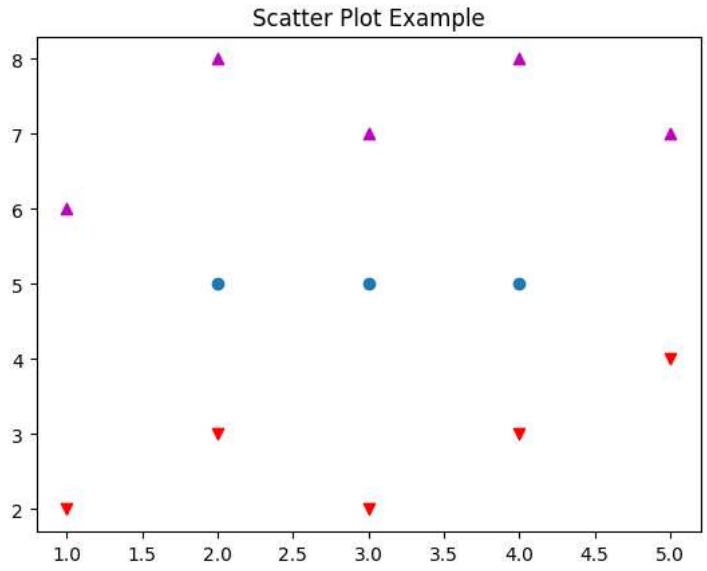
Histograms

```
import matplotlib.pyplot as plt
import numpy as np

# Use numpy to generate a bunch of random data in a bell curve around 5.
n = np.random.randn(100)

# m = [m for m in range(len(n))]
# plt.bar(m, n)
# plt.title("Raw Data")
# plt.show()
plt.hist(n, label="Blue Bar", color='b')
# plt.hist(n, bins=20)
# plt.title("Histogram")
plt.show()

# plt.hist(n, cumulative=True, bins=20)
# plt.title("Cumulative Histogram")
# plt.show()
```



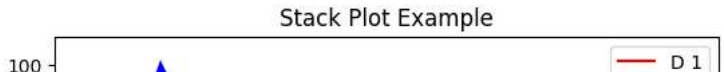
Scatter Plots

```
import matplotlib.pyplot as plt

x1 = [2, 3, 4]
y1 = [5, 5, 5]

x2 = [1, 2, 3, 4, 5]
y2 = [2, 3, 2, 3, 4]
y3 = [6, 8, 7, 8, 7]

# Markers: https://matplotlib.org/api/markers_api.html

plt.scatter(x1, y1)
plt.scatter(x2, y2, marker='v', color='r')
plt.scatter(x2, y3, marker='^', color='m')
plt.title('Scatter Plot Example')
plt.show()
```



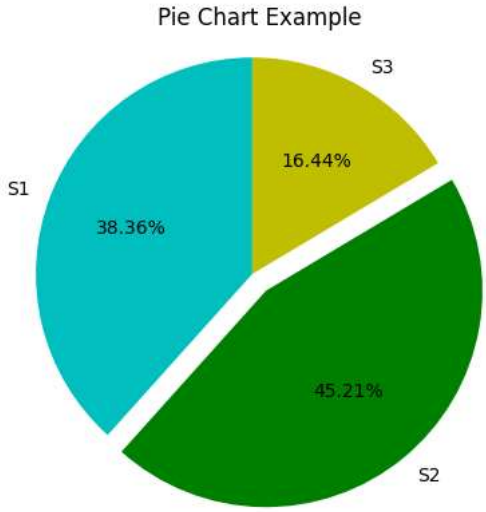
Stack Plots

```
import matplotlib.pyplot as plt

idxes = [ 1,  2,  3,  4,  5,  6,  7,  8,  9]
arr1  = [23, 40, 28, 43,  8, 44, 43, 18, 17]
arr2  = [17, 30, 22, 14, 17, 17, 29, 22, 30]
arr3  = [15, 31, 18, 22, 18, 19, 13, 32, 39]

# Adding legend for stack plots is tricky.
plt.plot([], [], color='r', label = 'D 1')
plt.plot([], [], color='g', label = 'D 2')
plt.plot([], [], color='b', label = 'D 3')

plt.stackplot(idxes, arr1, arr2, arr3, colors= ['r', 'g', 'b'])
plt.title('Stack Plot Example')
plt.legend()
plt.show()
```



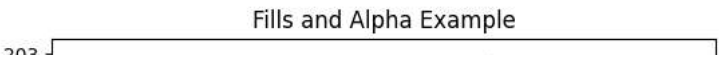
Pie Charts

```
import matplotlib.pyplot as plt

labels = 'S1', 'S2', 'S3'
sections = [56, 66, 24]
colors = ['c', 'g', 'y']

plt.pie(sections, labels=labels, colors=colors,
        startangle=90,
        explode = (0, 0.1, 0),
        autopct = '%1.2f%%')

plt.axis('equal') # Try commenting this out.
plt.title('Pie Chart Example')
plt.show()
```



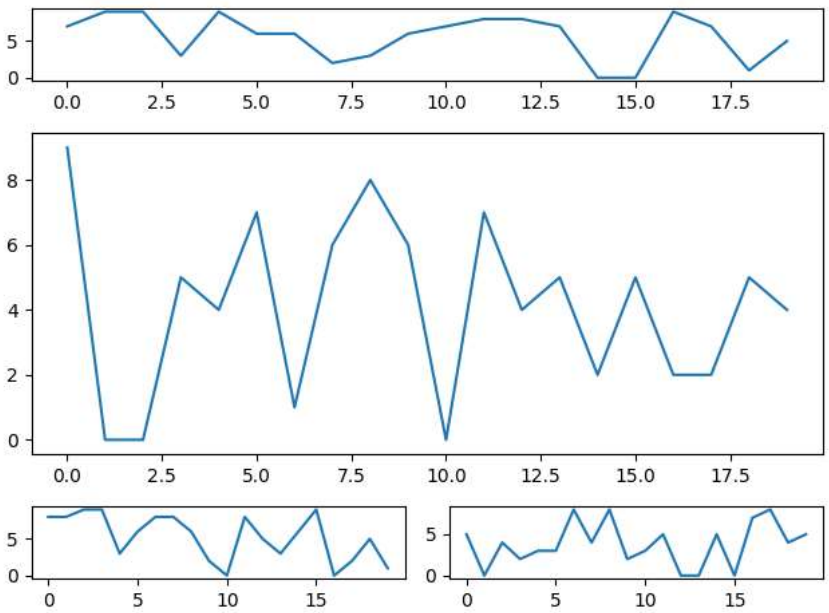
fill_between and alpha

```
import matplotlib.pyplot as plt
import numpy as np

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Fills and Alpha Example")
plt.show()
```



Subplotting using Subplot2grid

```
import matplotlib.pyplot as plt
import numpy as np

def random_plots():
  xs = []
  ys = []

  for i in range(20):
    x = i
    y = np.random.randint(10)

    xs.append(x)
    ys.append(y)

  return xs, ys

fig = plt.figure()
ax1 = plt.subplot2grid((5, 2), (0, 0), rowspan=1, colspan=2)
ax2 = plt.subplot2grid((5, 2), (1, 0), rowspan=3, colspan=2)
ax3 = plt.subplot2grid((5, 2), (4, 0), rowspan=1, colspan=1)
ax4 = plt.subplot2grid((5, 2), (4, 1), rowspan=1, colspan=1)

x, y = random_plots()
ax1.plot(x, y)

x, y = random_plots()
ax2.plot(x, y)

x, y = random_plots()
ax3.plot(x, y)

x, y = random_plots()
ax4.plot(x, y)

plt.tight_layout()
plt.show()
```



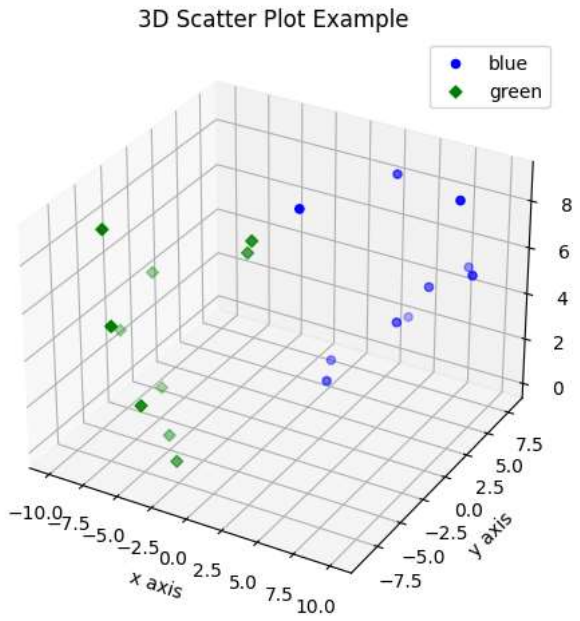
Plot styles

Colaboratory charts use Seaborn's custom styling by default. To customize styling further please see the matplotlib docs.

3D Graphs

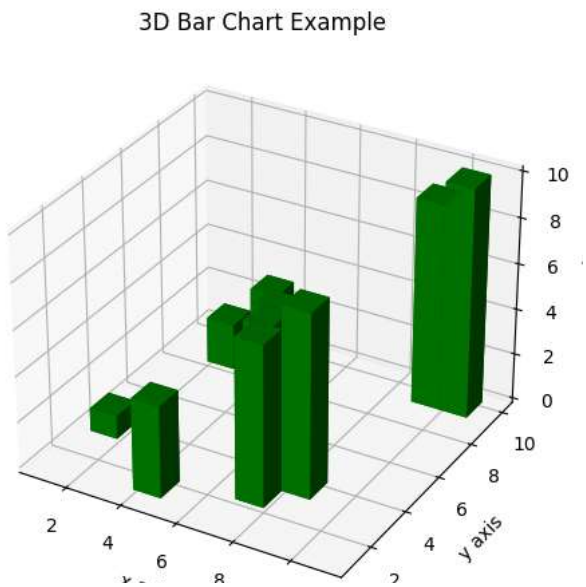
3D Scatter Plots

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import axes3d

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y1 = np.random.randint(10, size=10)
z1 = np.random.randint(10, size=10)

x2 = [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
y2 = np.random.randint(-10, 0, size=10)
z2 = np.random.randint(10, size=10)

ax.scatter(x1, y1, z1, c='b', marker='o', label='blue')
ax.scatter(x2, y2, z2, c='g', marker='D', label='green')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Scatter Plot Example")
plt.legend()
plt.tight_layout()
plt.show()
```



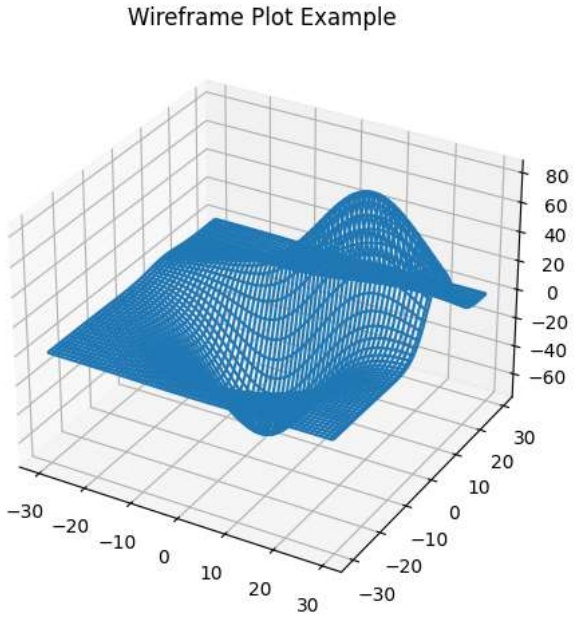
3D Bar Plots

```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.randint(10, size=10)
z = np.zeros(10)

dx = np.ones(10)
dy = np.ones(10)
dz = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

ax.bar3d(x, y, z, dx, dy, dz, color='g')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Bar Chart Example")
plt.tight_layout()
plt.show()
```



Wireframe Plots

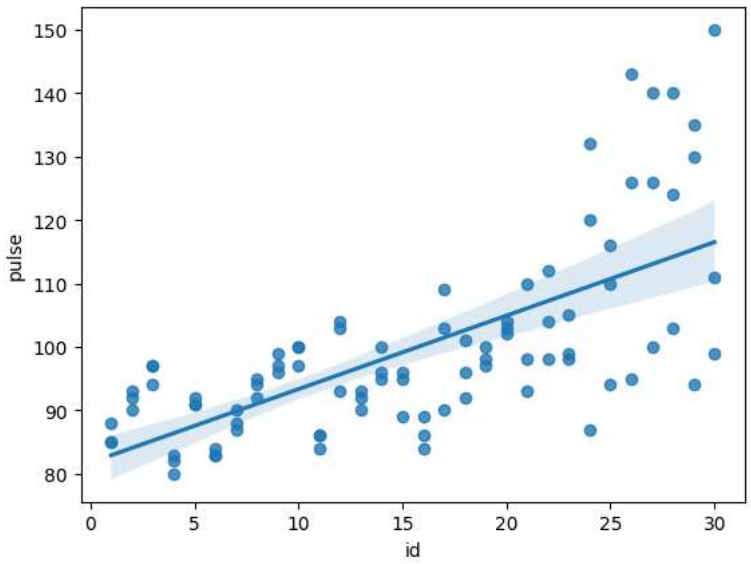
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x, y, z = axes3d.get_test_data()

ax.plot_wireframe(x, y, z, rstride = 2, cstride = 2)

plt.title("Wireframe Plot Example")
plt.tight_layout()
plt.show()
```



Seaborn

There are several libraries layered on top of Matplotlib that you can use in Colab. One that is worth highlighting is Seaborn:

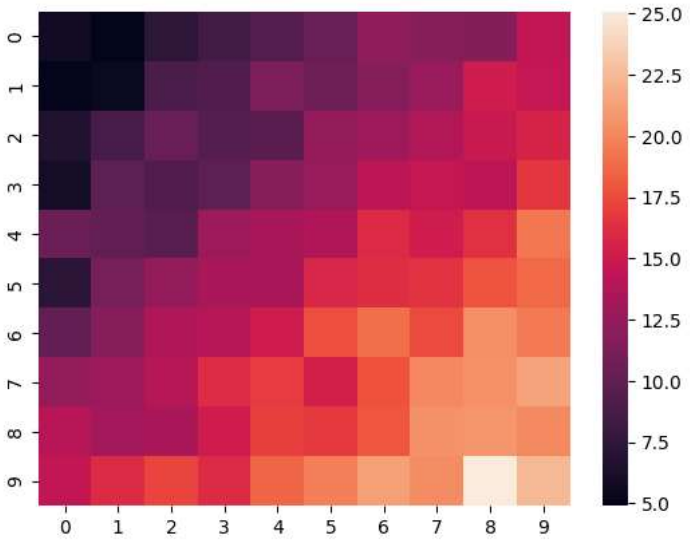
```
import seaborn as sns
import matplotlib.pyplot as plt

# loading dataset
data = sns.load_dataset("exercise")

# draw regplot
sns.regplot(x = "id",
            y = "pulse",
            data = data)

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



That's a simple scatterplot with a nice regression line fit to it, all with just one call to Seaborn's regplot.

Here's a Seaborn heatmap:

```
import matplotlib.pyplot as plt
import numpy as np

# Make a 10 x 10 heatmap of some random data
side_length = 10
# Start with a 10 x 10 matrix with values randomized around 5
data = 5 + np.random.randn(side_length, side_length)
# The next two lines make the values larger as we get closer to (9, 9)
data += np.arange(side_length)
data += np.reshape(np.arange(side_length), (side_length, 1))
# Generate the heatmap
sns.heatmap(data)
plt.show()
```

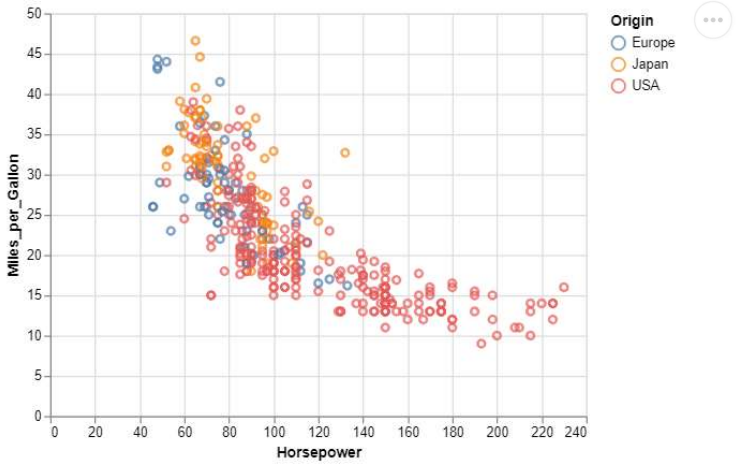


Altair

Altair is a declarative visualization library for creating interactive visualizations in Python, and is installed and enabled in Colab by default.

For example, here is an interactive scatter plot:

```
import altair as alt
from vega_datasets import data
cars = data.cars()

alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
).interactive()
```



For more examples of Altair plots, see the Altair snippets notebook or the external Altair Example Gallery.

Plotly

Sample

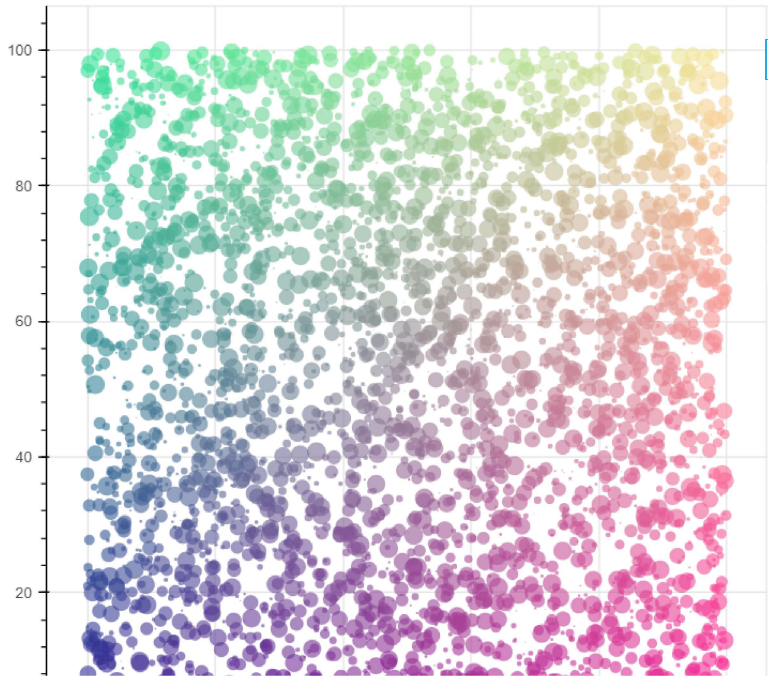
```
from plotly.offline import iplot
import plotly.graph_objs as go

data = [
    go.Contour(
        z=[[10, 10.625, 12.5, 15.625, 20],
           [5.625, 6.25, 8.125, 11.25, 15.625],
           [2.5, 3.125, 5., 8.125, 12.5],
           [0.625, 1.25, 3.125, 6.25, 10.625],
           [0, 0.625, 2.5, 5.625, 10]]
    )
]
iplot(data)
```

Bokeh

Sample

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Call once to configure Bokeh to display plots inline in the notebook.
output_notebook()
```

```
N = 4000
x = np.random.random(size=N) * 100
y = np.random.random(size=N) * 100
radii = np.random.random(size=N) * 1.5
colors = ["#%02x%02x%02x" % (r, g, 150) for r, g in zip(np.floor(50+2*x).astype(int), np.floor(30+2*y).astype(int))]

p = figure()
p.circle(x, y, radius=radii, fill_color=colors, fill_alpha=0.6, line_color=None)
show(p)
```

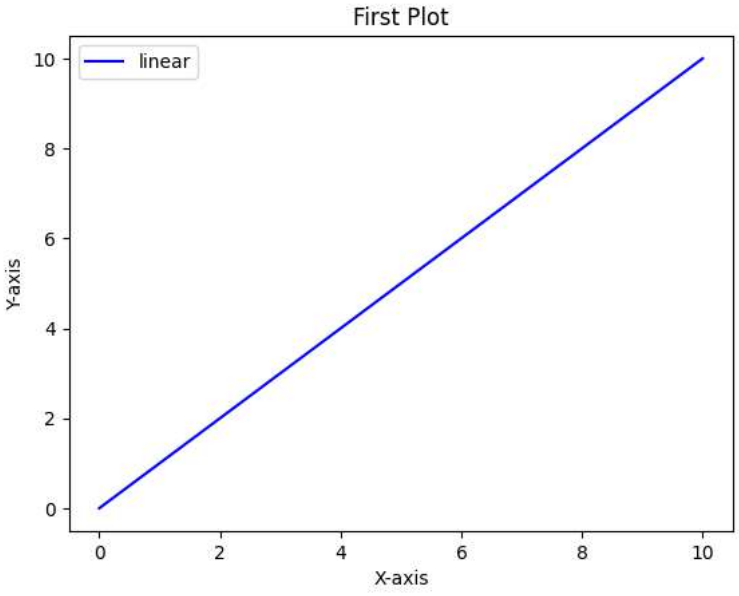


PlottingAndVisualizingData

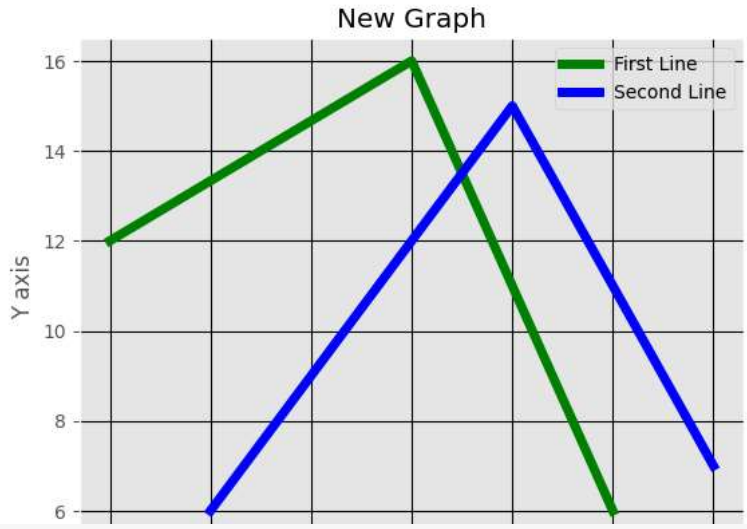
```
#import
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
#Plotting a line chart
x = np.linspace(0, 10, 100) #(start, stop, number)
print(x)
plt.plot(x, x, label='linear',color='b')
plt.legend()
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("First Plot")
plt.show()
```

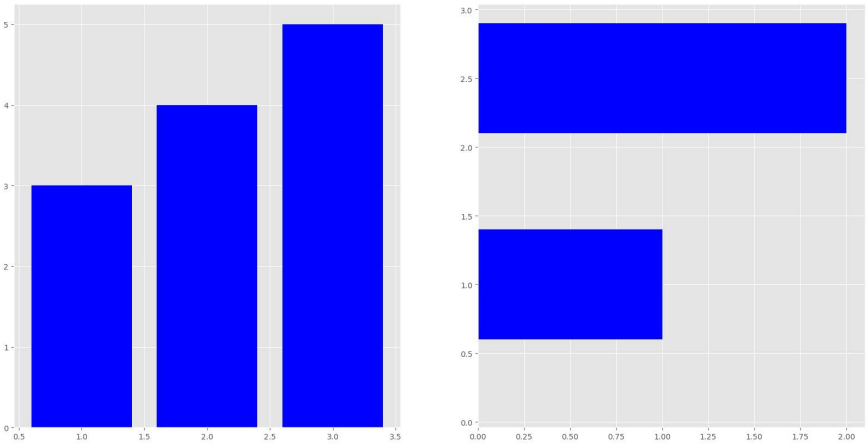
```
[ 0.          0.1010101   0.2020202   0.3030303   0.4040404   0.50505051
  0.60606061  0.70707071  0.80808081  0.90909091  1.01010101  1.11111111
  1.21212121  1.31313131  1.41414141  1.51515152  1.61616162  1.71717172
  1.81818182  1.91919192  2.02020202  2.12121212  2.22222222  2.32323232
  2.42424242  2.52525253  2.62626263  2.72727273  2.82828283  2.92929293
  3.03030303  3.13131313  3.23232323  3.33333333  3.43434343  3.53535354
  3.63636364  3.73737374  3.83838384  3.93939394  4.04040404  4.14141414
  4.24242424  4.34343434  4.44444444  4.54545455  4.64646465  4.74747475
  4.84848485  4.94949495  5.05050505  5.15151515  5.25252525  5.35353535
  5.45454545  5.55555556  5.65656566  5.75757576  5.85858586  5.95959596
  6.06060606  6.16161616  6.26262626  6.36363636  6.46464646  6.56565657
  6.66666667  6.76767677  6.86868687  6.96969697  7.07070707  7.17171717
  7.27272727  7.37373737  7.47474747  7.57575758  7.67676768  7.77777778
  7.87878788  7.97979798  8.08080808  8.18181818  8.28282828  8.38383838
  8.48484848  8.58585859  8.68686869  8.78787879  8.88888889  8.98989899
  9.09090909  9.19191919  9.29292929  9.39393939  9.49494949  9.5959596
  9.6969697   9.7979798   9.8989899  10.        ]
```



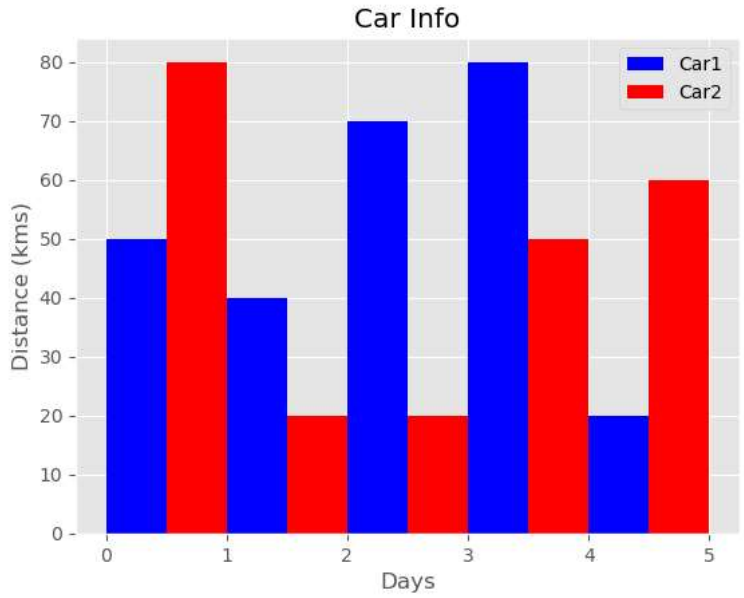

```
#plotting multi line with background
from matplotlib import style
style.use('ggplot')
x1 = [5,8,10]
y1 = [12,16,6]
x2 = [6,9,11]
y2 = [6,15,7]
plt.plot(x1,y1,'g',label='First Line', linewidth=5)
plt.plot(x2,y2,'b',label='Second Line',linewidth=5)
plt.title('New Graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.legend()
plt.grid(True,color='k')
plt.show()
```



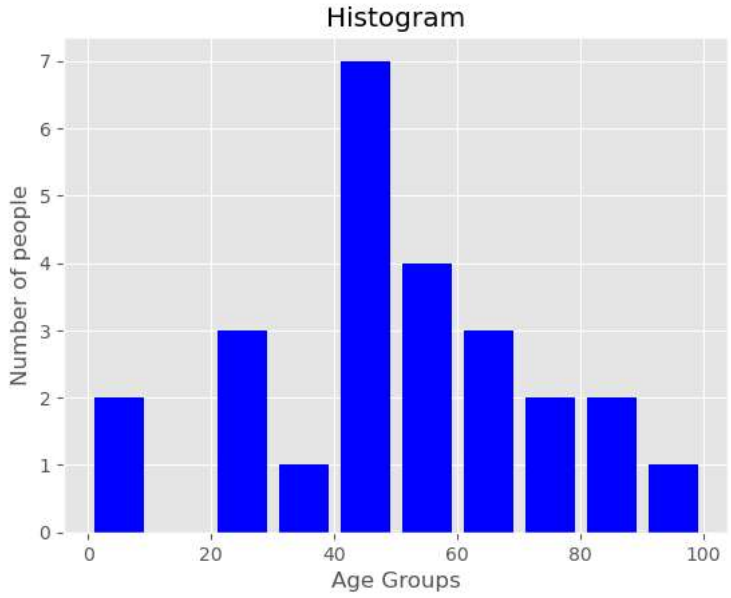

```
# plotting bar graph
fig = plt.figure(figsize=(20,10))
ax1 = fig.add_subplot(121) #1x2 grid 1st subplot
ax2 = fig.add_subplot(122) #1x2 grid 2nd subplot

ax1.bar([1,2,3],[3,4,5],color='b')
ax2.barh([0.5,1,2.5],[0,1,2],color='b')

plt.show()
```



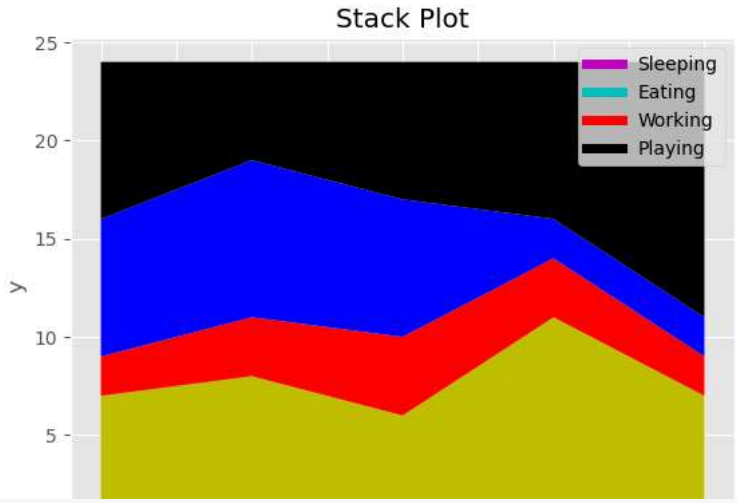
```
#plot multi bar

plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],label="Car1",color='b', width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60], label="Car2", color='r',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Car Info')
plt.show()
```



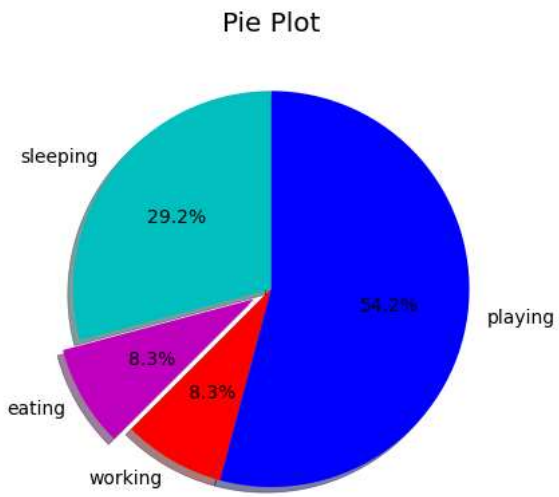

```
#plot histogram
population_age = [22,55,62,45,21,22,34,42,42,4,2,102,95,85,55,110,120,70,65,55,111,115,80,75,65,54,44,43,42,48]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar', color='b',rwidth=0.8)
plt.xlabel('Age Groups')
plt.ylabel('Number of people')
plt.title('Histogram')
plt.show()
```



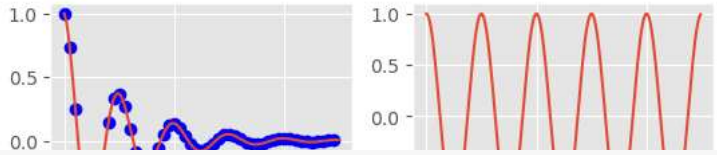

```
#plot area
days = [1,2,3,4,5]
sleeping =[7,8,6,11,7]
eating = [2,3,4,3,2]
working =[7,8,7,2,2]
playing = [8,5,7,8,13]
plt.plot([],[],color='m', label='Sleeping', linewidth=5)
plt.plot([],[],color='c', label='Eating', linewidth=5)
plt.plot([],[],color='r', label='Working', linewidth=5)
plt.plot([],[],color='k', label='Playing', linewidth=5)
plt.stackplot(days, sleeping,eating,working,playing, colors=['y','r','b','k'])
plt.xlabel('x')
plt.ylabel('y')
plt.title('Stack Plot')
plt.legend()
plt.show()
```




```
#drawing pie chart

days = [1,2,3,4,5]

sleeping =[7,8,6,11,7]
eating = [2,3,4,3,2]
working =[7,8,7,2,2]
playing = [8,5,7,8,13]
slices = [7,2,2,13]
activities = ['sleeping','eating','working','playing']
cols = ['c','m','r','b']

plt.pie(slices,
  labels=activities,
  colors=cols,
  startangle=90,
  shadow= True,
  explode=(0,0.1,0,0),
  autopct='%1.1f%%')

plt.title('Pie Plot')
plt.show()
```

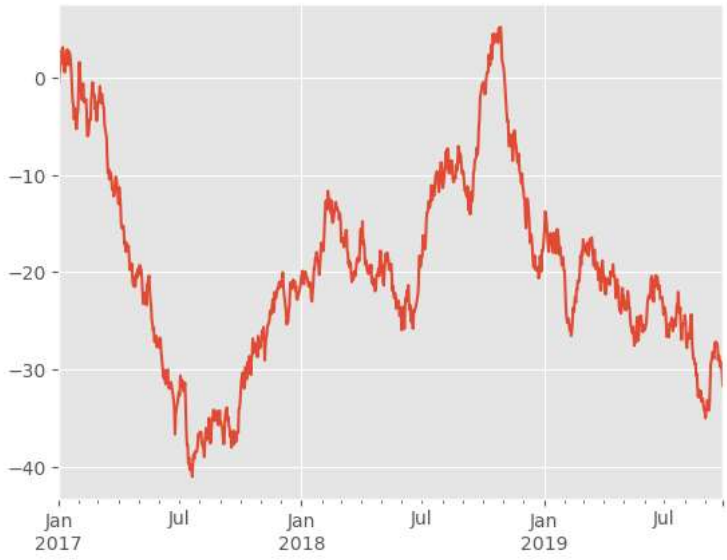



```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.subplot(221)
plt.plot(t1, f(t1), 'bo', t2, f(t2))
plt.subplot(222)
plt.plot(t2, np.cos(2*np.pi*t2))
plt.show()
```




```
#drawing with pandas
ts = pd.Series(np.random.randn(1000),index=pd.date_range('1/1/2017', periods=1000))

ts = ts.cumsum()

ts.plot()
```

```
<Axes: >
```

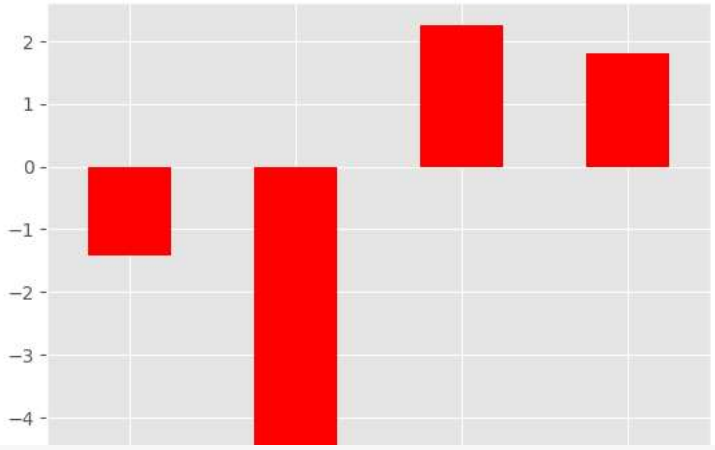



```
#drawing multiple time series
df = pd.DataFrame(np.random.randn(1000, 4),index=ts.index, columns=list('ABCD'))

df = df.cumsum()

plt.figure()

df.plot()
```

```
<Axes: >
<Figure size 640x480 with 0 Axes>
```

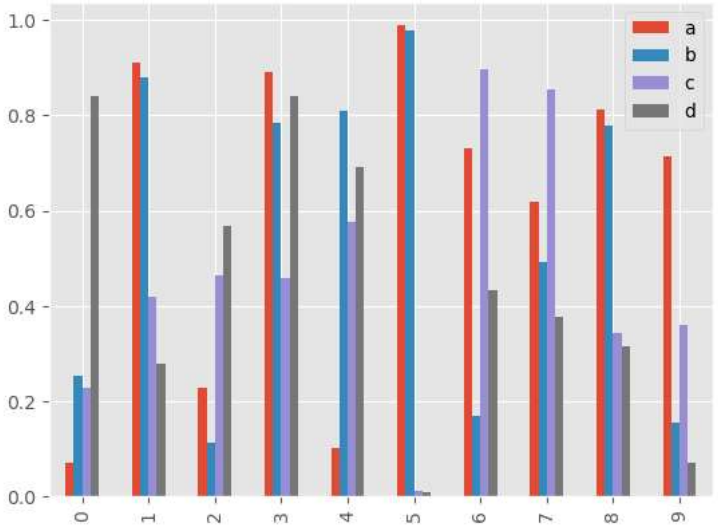

```
#plotting bar graph
plt.figure()
df.iloc[5].plot(kind='bar',color='r')
```

```
<Axes: >
```



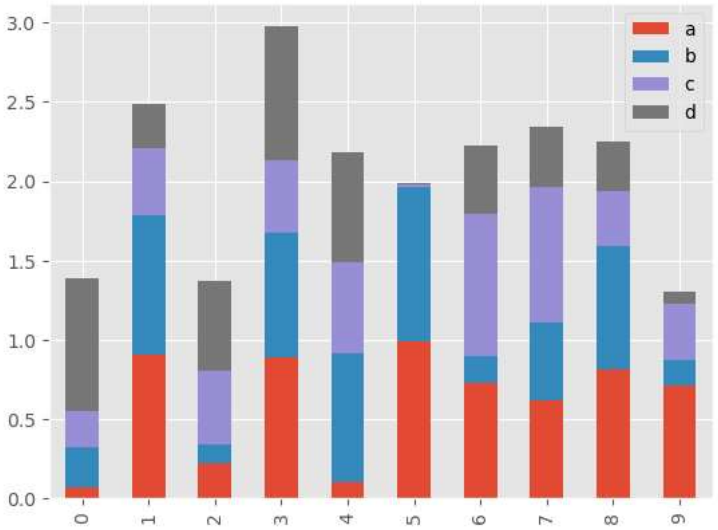
```
#plotting multiple bar graph
df2 = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])

df2.plot.bar()
```

```
<Axes: >
```



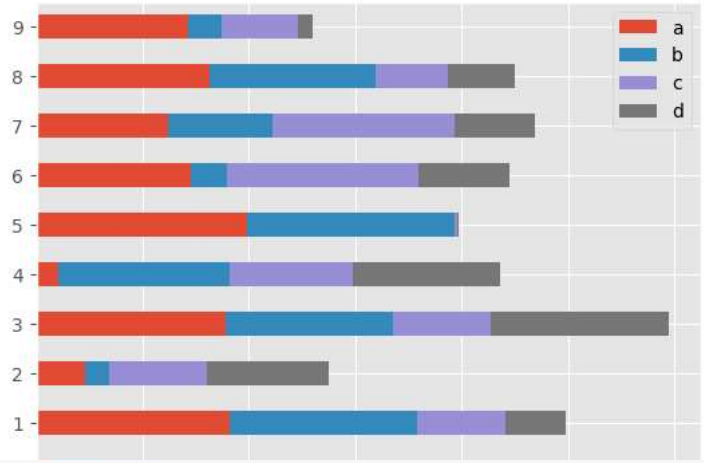

```
#plotting stacked vertical bar graph
df2.plot.bar(stacked=True)
```

```
<Axes: >
```



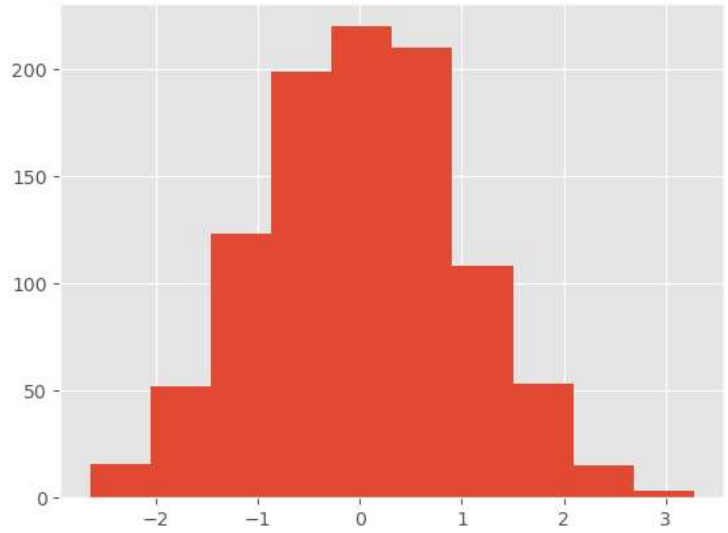

```
#plotting stacked horizontal bar graph
df2.plot.barh(stacked=True)
```

```
<Axes: >
```



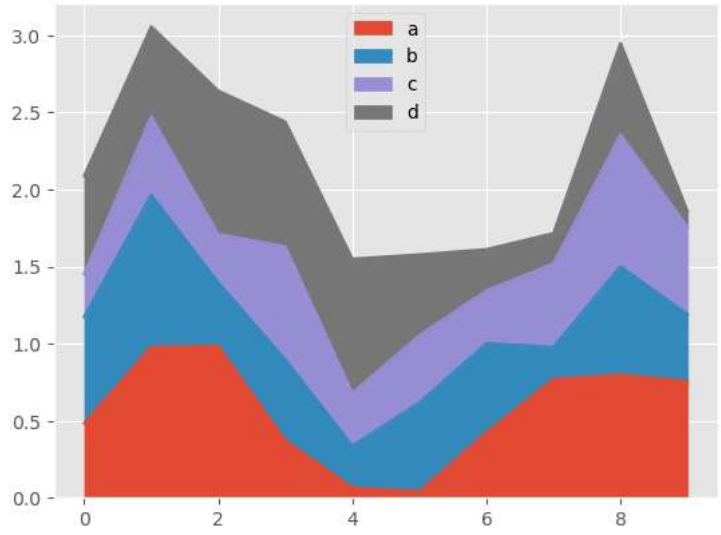

```
#plotting histogram
df['A'].diff().hist()
```

```
<Axes: >
```



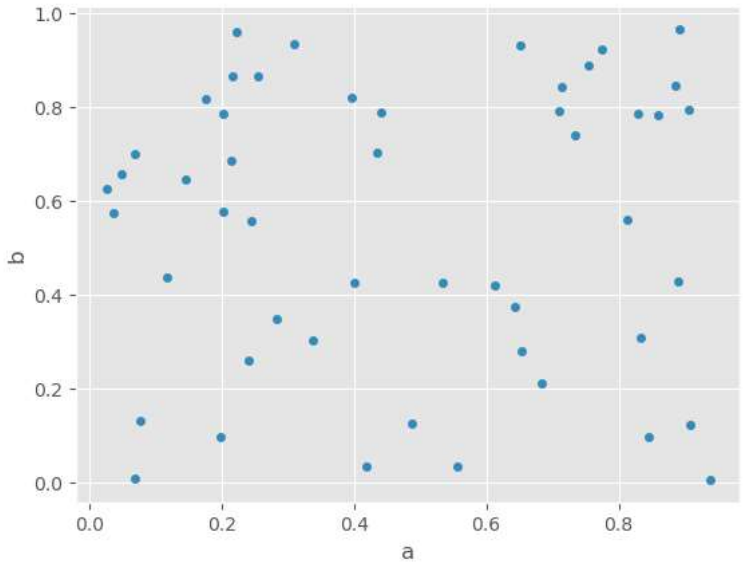

```
#plotting area
df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])

df.plot.area()
```

```
<Axes: >
```



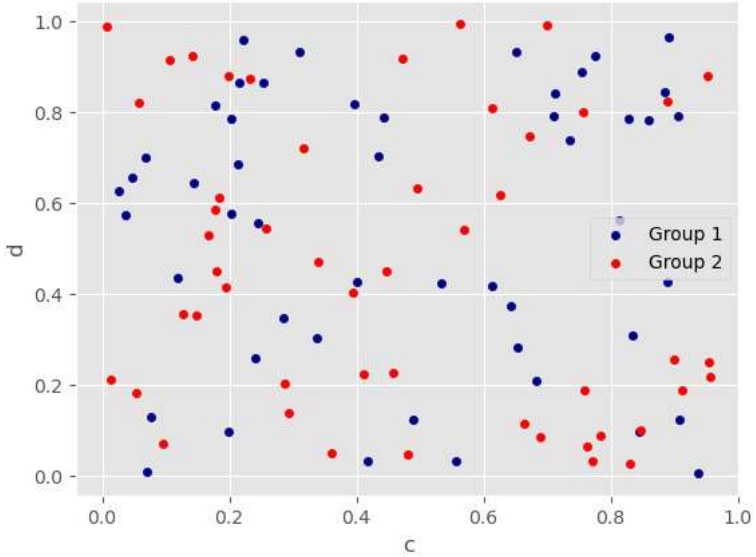

```
#plotting scatter graph
df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])

df.plot.scatter(x='a', y='b')
```

```
<Axes: xlabel='a', ylabel='b'>
```



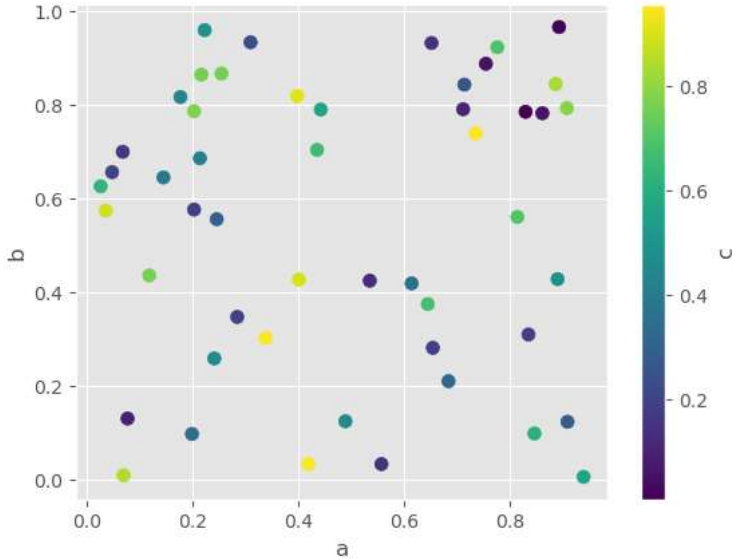

```
#plotting
ax = df.plot.scatter(x='a', y='b', color='DarkBlue', label='Group 1')
df.plot.scatter(x='c', y='d', color='Red', label='Group 2', ax=ax)
```

```
<Axes: xlabel='c', ylabel='d'>
```



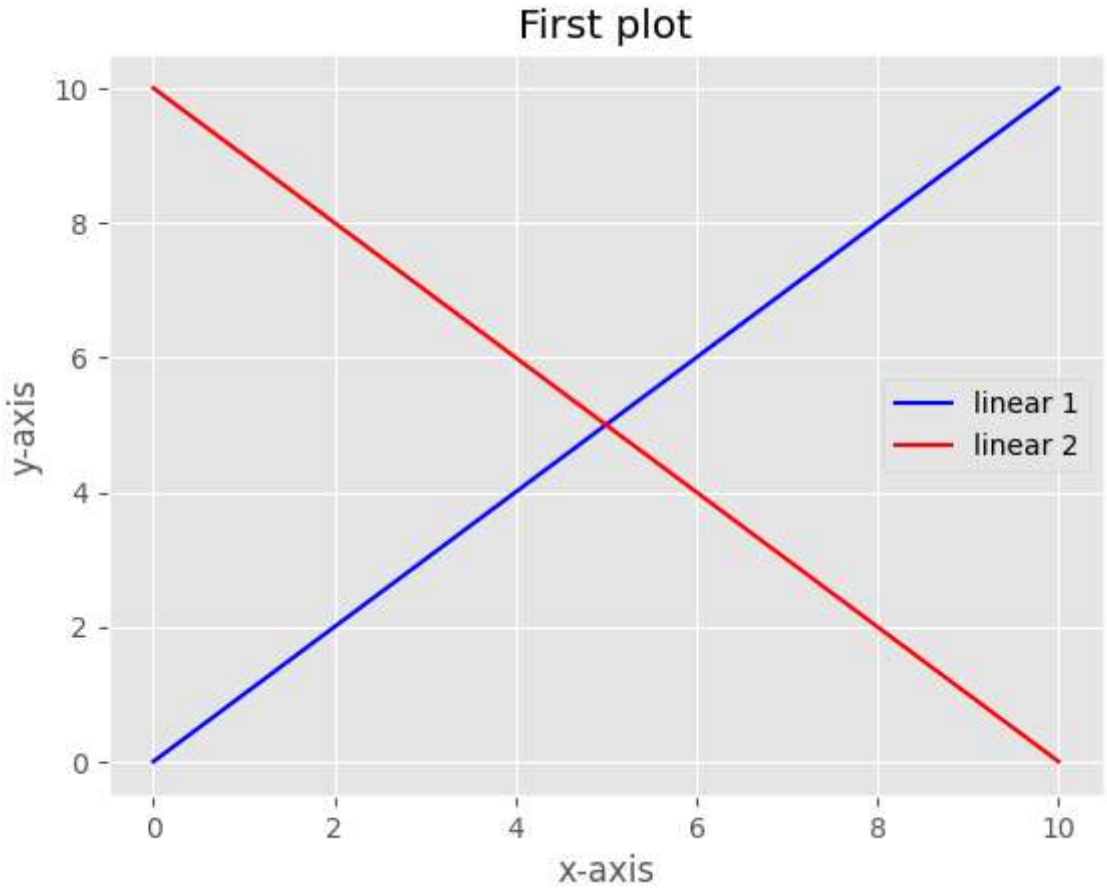

```
#plotting grayscale scatter plot
df.plot.scatter(x='a', y='b', c='c', s=50)
```

```
<Axes: xlabel='a', ylabel='b'>
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
x=np.linspace(10,0,100)
y=np.linspace(0,10,100)

plt.plot(x,x,label='linear 1',color='b')
plt.plot(x,y,label='linear 2',color='r')
plt.legend()
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.title("First plot")
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



```
import numpy as np
import pandas as pd
df = pd.read_csv("test.csv")
print(df)
print("\nhead\n")
print(df.head())
print("\ntail\n")
print(df.tail())
print("\ntail 2\n")
print(df.tail(2))
print("\nindex\n")
print(df.index)
print("\ncolumn\n")
print(df.columns)
print("\ndescribe\n")
print(df.describe)
print("\nsort\n")
print(df.sort_values(by="Period",ascending=False))
print("\nT\n")
print(df.T)
```

```
              1  Period
     1503  1505    1504
```

```
1500  1502    1501
...    ...     ...
4        6       5
3        5       4
2        4       3
1        3       2
0        2       1

[1505 rows x 2 columns]
```

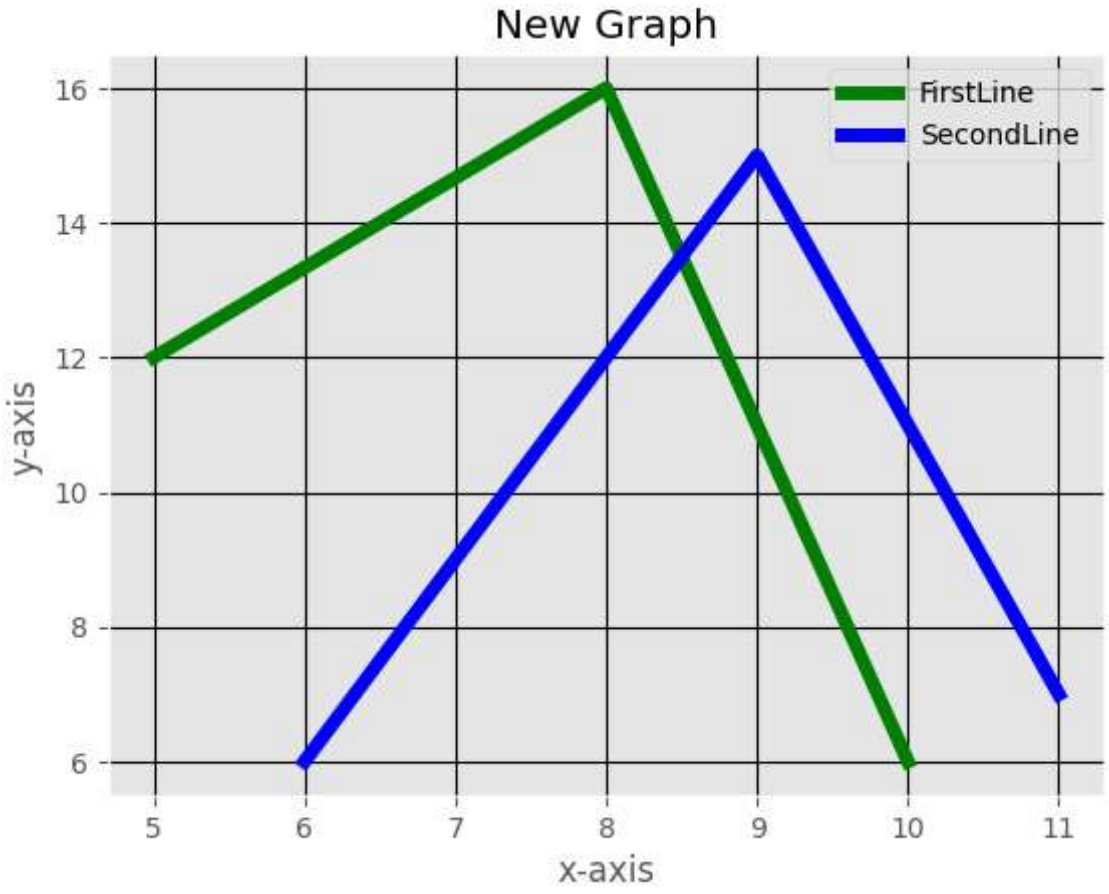
T

```
        0     1     2     3     4     5     6     7     8     9     ...  1495  \
1          2     3     4     5     6     7     8     9    10    11  ...  1497
Period     1     2     3     4     5     6     7     8     9    10  ...  1496

        1496  1497  1498  1499  1500  1501  1502  1503  1504
1       1498  1499  1500  1501  1502  1503  1504  1505  1506
Period  1497  1498  1499  1500  1501  1502  1503  1504  1505

[2 rows x 1505 columns]
```

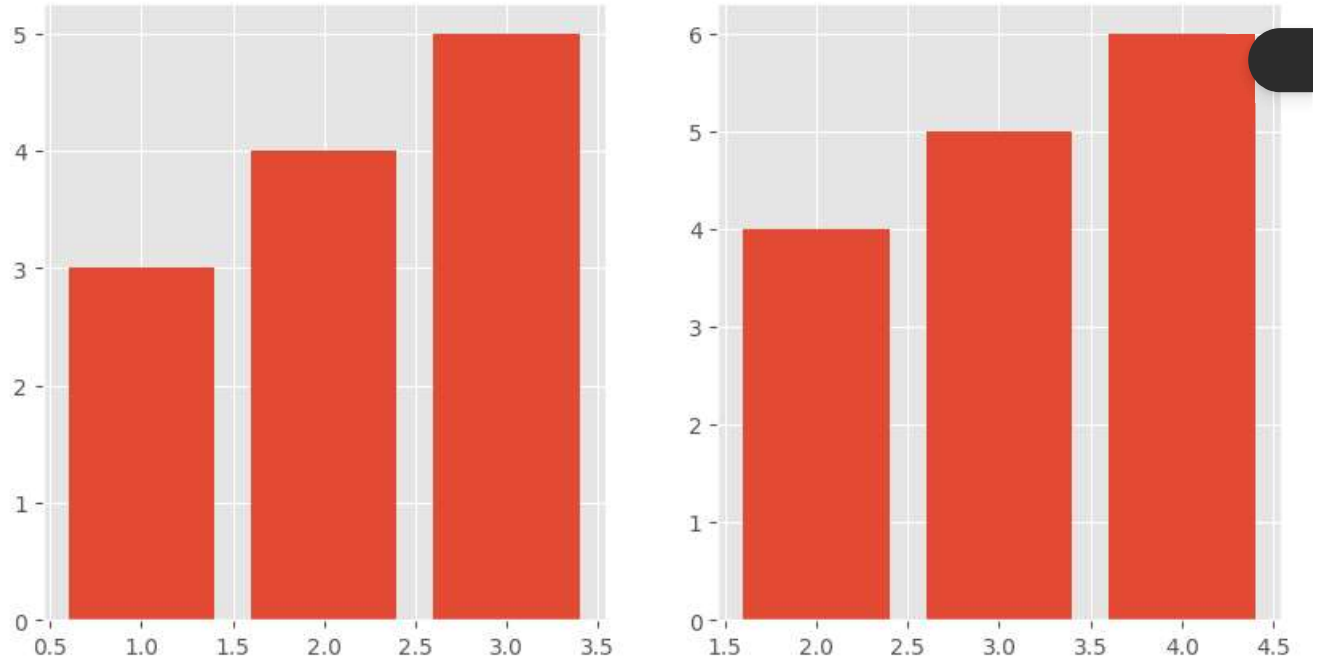
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.bar([1, 2, 3, 4, 5],[88, 4, 56, 1, 23], label="car1", color="b",width=5)
plt.bar([1, 2, 3, 4, 5], [ 5, 4, 3, 2, 1] ,label="car2",color="g",width=5)
plt.legend()
plt.title("new graph")
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.show()
```

new graph

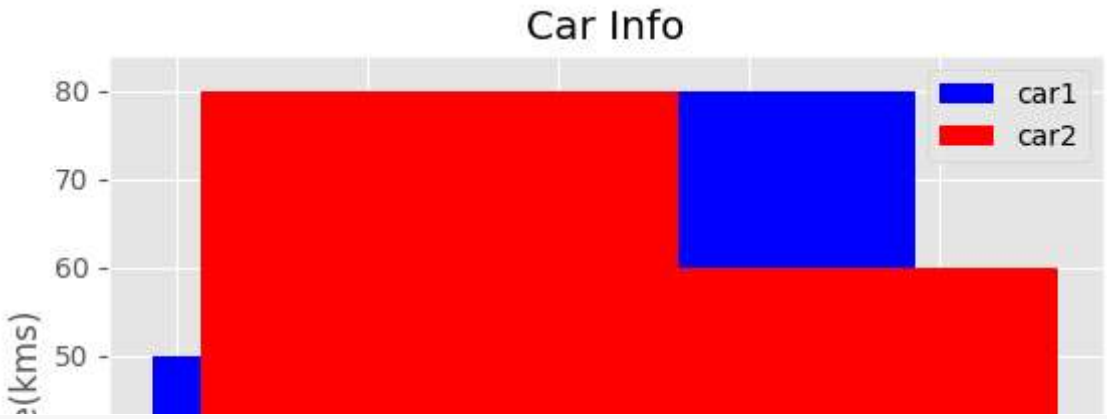
```
#18. plotting multiline with background
from matplotlib import style
style.use('ggplot')
x1=[5,8,10]
y1=[12,16,6]
x2=[6,9,11]
y2=[6,15,7]
plt.plot(x1,y1,'g',label="FirstLine",linewidth=5)
plt.plot(x2,y2,'b',label="SecondLine",linewidth=5)
plt.title("New Graph")
plt.ylabel("y-axis")
plt.xlabel("x-axis")
plt.legend()
plt.grid(True,color='k')
plt.show()
```



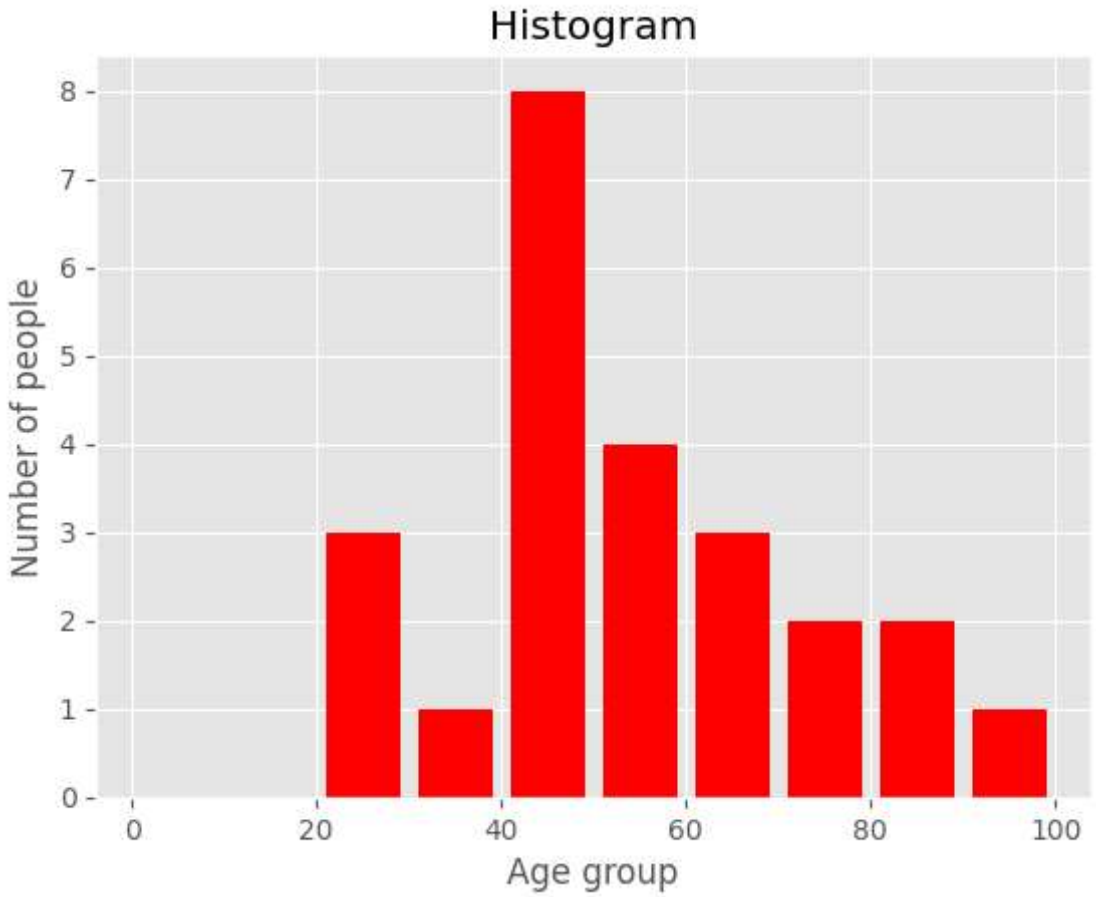
```
#19. Plotting bar graph
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10, 5))

ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.bar([1, 2, 3], [3, 4, 5])
ax2.bar([2, 3, 4], [4, 5, 6])

plt.show()
```




```
#20. Plot Multibar
plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],label='car1',color='b',width=5)
plt.bar([0.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],label='car2',color='r',width=5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance(kms)')
plt.title('Car Info')
plt.show()
```




```
#21. Plot histogram
population_age=[22,55,62,45,21,22,34,42,42,42,102,95,85,55,110,120,70,65,55,111,115,80,75
bins=[0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age,bins,histtype='bar',color='r',rwidth=0.8)
plt.xlabel("Age group")
plt.ylabel("Number of people")
plt.title('Histogram')
plt.show()
```



ML Model housing

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

```
df = pd.read_csv("/content/housing.csv")
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-122.23	37.88	41.0	880.0	129.0	322.0
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0
2	-122.24	37.85	52.0	1467.0	190.0	496.0
3	-122.25	37.85	52.0	1274.0	235.0	558.0
4	-122.25	37.85	52.0	1627.0	280.0	565.0

```
from sklearn.model_selection import train_test_split
df.dropna(inplace=True)
train_set,test_set=train_test_split(df,test_size=0.2,random_state=42)
X_train=train_set.iloc[:,:-2]
X_train
# y1_train=train_set.iloc[:,-2:]
# y1_train
y_train=train_set.iloc[:,-2:-1]
y_train
X_test=test_set.iloc[:,:-2]
X_test
# y1_test=test_set.iloc[:,-2:]
# y1_test
y_test=test_set.iloc[:,-2:-1]
y_test
```

	median_house_value
14425	80100.0
16398	500001.0
7721	352100.0
1411	187500.0
1336	361000.0
...	...
8285	163100.0
6264	229200.0
2999	327500.0
13452	58300.0
14809	500001.0

4087 rows × 1 columns

```
# from sklearn.model_selection import train_test_split
# df.dropna(inplace=True)
# train_set,test_set=train_test_split(df,test_size=0.2,random_state=42)
# X_train=train_set.iloc[:,:-2]
# X_train
# y1_train=train_set.iloc[:,-2:]
# y1_train
# y_train=y1_train.iloc[:,:1]
# y_train
# X_test=test_set.iloc[:,:-2]
# X_test
# y1_test=test_set.iloc[:,-2:]
# y1_test
# y_test=y1_test.iloc[:,:1]
# y_test
```

```
from sklearn.linear_model import LinearRegression
#print(train_X)
model=LinearRegression()
model.fit(X_train,y_train)
```

▾ LinearRegression

LinearRegression()

```
#testing model
y_pred=model.predict(X_test)
print(y_pred)
```

```
[[ 62720.63712627]
 [414459.89124757]
 [238028.6841184 ]
 ...
 [282366.97531626]
 [ 77825.94834712]
 [387233.21922771]]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
70025.94402055604
```

```
X_sample=[[-122.23,37.88,41.0,880.0,129.0,322.0,126.0,8.3252]]
y_pred=model.predict(X_sample)
print(y_pred)
```

```
[[411052.90996556]]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not
  warnings.warn(
```

```
# longitude = input("longitude\n")
# latitude = input("latitude\n")
# housing_median_age = input("housing_median_age\n")
# total_rooms = input("total_rooms\n")
# total_bedrooms = input("total_bedrooms\n")
# population = input("population\n")
# households = input("households\n")
# median_income = input("median_income\n")
# X_sample1=[[longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population
# y_pred=model.predict(X_sample)
# print("house price =")
# print(y_pred)
```

```
#checking Decision tree Regressor

from sklearn.tree import DecisionTreeRegressor
model2 = DecisionTreeRegressor()
model2.fit(X_train,y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
#testing model
y_pred=model2.predict(X_test)
print(y_pred)
```

```
[ 78500. 500001. 235500. ... 342500.  60900. 374000.]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
69501.52492301987
```

```
#Using Cross Validation

from sklearn.model_selection import cross_val_score
model2_rmses = -cross_val_score(model2, X_train, y_train, scoring = "neg_root_mean_square
model2_rmses
pd.Series(model2_rmses).describe()
```

```
count       10.000000
mean     70278.875928
std       1891.063101
min      66722.422826
25%      69387.047113
50%      70306.525198
75%      71507.897823
max      73235.984580
dtype: float64
```

```
#checking Random Forest Regressor

from sklearn.ensemble._forest import RandomForestRegressor
model3 = RandomForestRegressor()
model3.fit(X_train,y_train)
```

```
<ipython-input-250-aee46ef53514>:5: DataConversionWarning: A column-vector y was pass
  model3.fit(X_train,y_train)
▾ RandomForestRegressor
RandomForestRegressor()
```

```
#testing model
y_pred=model3.predict(X_test)
print(y_pred)
```

```
[ 71916.   475829.58 253150.   ... 296846.05  73683.   432412.32]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
49636.368617575325
```

```
#using Cross Validation
from sklearn.model_selection import cross_val_score
model3_rmses = -cross_val_score(model2, X_train, y_train, scoring = "neg_root_mean_square
model3_rmses
pd.Series(model3_rmses).describe()
```

```
count       10.000000
mean     69720.629159
std       1472.164911
min      66317.550887
25%      69340.766212
50%      70004.823119
75%      70800.943073
max      71196.083567
dtype: float64
```

ML Model for Salary Dataset

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

```
df = pd.read_csv("/content/Salary_dataset.csv")
df.head()
```

	Unnamed: 0	YearsExperience	Salary
0	0	1.2	39344.0
1	1	1.4	46206.0
2	2	1.6	37732.0
3	3	2.1	43526.0
4	4	2.3	39892.0

```
from sklearn.model_selection import train_test_split
df.dropna(inplace=True)
train_set,test_set=train_test_split(df,test_size=0.2,random_state=42)
X_train=train_set.iloc[:,:-1]
X_train
y1_train=train_set.iloc[:,-2:]
y1_train
y_train=y1_train.iloc[:,:1]
y_train
X_test=test_set.iloc[:,-2:-1]
X_test
y1_test=test_set.iloc[:,-2:]
y1_test
y_test=y1_test.iloc[:,:1]
y_test
```

	YearsExperience
27	9.7
15	5.0
23	8.3
17	5.4
8	3.3
9	3.8

```
df.shape
```

```
(30, 3)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Unnamed: 0       30 non-null     int64  
 1   YearsExperience  30 non-null     float64
 2   Salary           30 non-null     float64
dtypes: float64(2), int64(1)
memory usage: 848.0 bytes
```

```
df.describe()
```

	Unnamed: 0	YearsExperience	Salary
count	30.000000	30.000000	30.000000
mean	14.500000	5.413333	76004.000000
std	8.803408	2.837888	27414.429785
min	0.000000	1.200000	37732.000000
25%	7.250000	3.300000	56721.750000
50%	14.500000	4.800000	65238.000000
75%	21.750000	7.800000	100545.750000
max	29.000000	10.600000	122392.000000

```
df.isnull().sum()
```

```
Unnamed: 0          0
YearsExperience     0
Salary              0
dtype: int64
```

```
df[['YearsExperience','Salary']].cov()
```

	YearsExperience	Salary
YearsExperience	8.053609	7.610630e+04
Salary	76106.303448	7.515510e+08

```
X=df.drop('Salary',axis=1)
```

```
y=df.Salary
```

```
X.head()
```

	Unnamed: 0	YearsExperience
0	0	1.2
1	1	1.4
2	2	1.6
3	3	2.1
4	4	2.3

```
y.head()
```

```
0    39344.0
1    46206.0
2    37732.0
3    43526.0
4    39892.0
Name: Salary, dtype: float64
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=0,test_size=0.30)
print(X_train.shape)
X_test.shape
```

```
(21, 2)
(9, 2)
```

```
from sklearn.linear_model import LinearRegression
#print(train_X)
model=LinearRegression()
model.fit(X_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
#testing model
y_pred=model.predict(X_test)
print(y_pred)
```

```
[ 41068.96469651 124994.63666745  63523.55124173  63316.24056634
 117108.4126212  109222.18857494 117564.91702613  63772.74497127
  74647.18917665]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
4929.255926250668
```

```
y_pred=model.predict(X_test)
print(y_pred)
```

```
[ 41068.96469651 124994.63666745  63523.55124173  63316.24056634
 117108.4126212  109222.18857494 117564.91702613  63772.74497127
  74647.18917665]
```

```
#checking Decision tree Regressor

from sklearn.tree import DecisionTreeRegressor
model2 = DecisionTreeRegressor()
model2.fit(X_train,y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
#testing model
y_pred=model2.predict(X_test)
print(y_pred)
```

```
[ 46206. 121873.  56958.  57190. 105583. 105583. 105583.  56958.  66030.]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
8131.063091625842
```

```
#Using Cross Validation

from sklearn.model_selection import cross_val_score
model2_rmses = -cross_val_score(model2, X_train, y_train, scoring = "neg_root_mean_square
model2_rmses
pd.Series(model2_rmses).describe()
```

```
count       10.000000
mean      7702.111486
std       3476.187428
min       3232.645743
25%       4618.423468
50%       7631.134269
75%      10460.976139
max      12499.022842
dtype: float64
```

```
#checking Random Forest Regressor

from sklearn.ensemble._forest import RandomForestRegressor
model3 = RandomForestRegressor()
model3.fit(X_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
#testing model
y_pred=model3.predict(X_test)
print(y_pred)
```

```
[ 44241.59 116046.6   58549.96  58206.66 109943.8  108748.42 110508.
  57779.96  69409.95]
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
6283.461087346685
```

```
#using Cross Validation
from sklearn.model_selection import cross_val_score
model3_rmses = -cross_val_score(model2, X_train, y_train, scoring = "neg_root_mean_square
model3_rmses
pd.Series(model3_rmses).describe()
```

```
count       10.000000
mean      7424.631240
std       3097.082095
min       2893.969938
25%       5828.508358
50%       6886.732808
75%       9625.547162
max      12499.022842
dtype: float64
```

weight-height

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

```
df = pd.read_csv("/content/weight-height.csv")
df
```

	Gender	Height	Weight

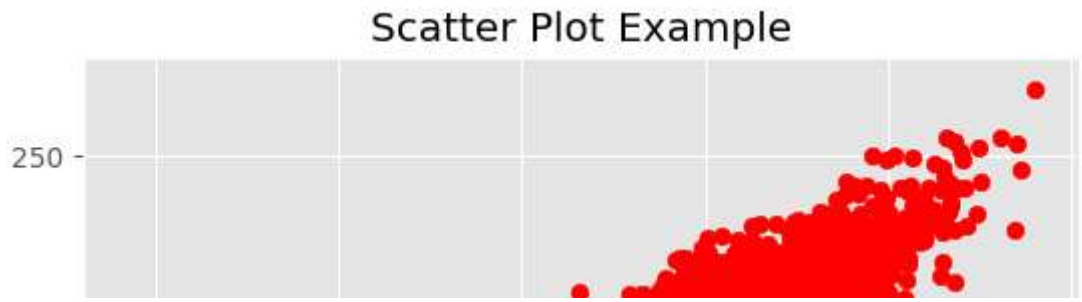
```
x = df['Height']
#x = df.iloc[:1]
x
```

```
0       73.847017
1       68.781904
2       74.110105
3       71.730978
4       69.881796
          ...
9995    66.172652
9996    67.067155
9997    63.867992
9998    69.034243
9999    61.944246
Name: Height, Length: 10000, dtype: float64
```

```
y = df['Weight']
#y = df.iloc[:2]
y
```

```
0       241.893563
1       162.310473
2       212.740856
3       220.042470
4       206.349801
           ...
9995    136.777454
9996    170.867906
9997    128.475319
9998    163.852461
9999    113.649103
Name: Weight, Length: 10000, dtype: float64
```

```
plt.scatter(x, y, marker = 'o', color = 'r')
plt.title("Scatter Plot Example")
plt.show()
```



Binary Classification using Logistic Reggression

```
#import
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

```
dataset = pd.read_csv('/content/diabetes2.csv')
dataset.head()
dataset.isnull().any()
dataset=dataset.fillna(method='ffill')
```

```
#dividing data into x & y
X = dataset.drop(columns = ['Outcome'])
y = dataset['Outcome']
```

```
#split data into test & train dataset
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=0)
```

```
logreg = LogisticRegression(solver='liblinear')
```

```
#fitting model with data
logreg.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression(solver='liblinear')
```

```
#predict for X_test
y_pred=logreg.predict(X_test)
```

```
from sklearn.metrics import mean_squared_error
rmse=mean_squared_error(y_pred,y_test,squared=False)
rmse
```

```
0.4264014327112209
```

```
#check the model accuracy
from sklearn import metrics
print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
```

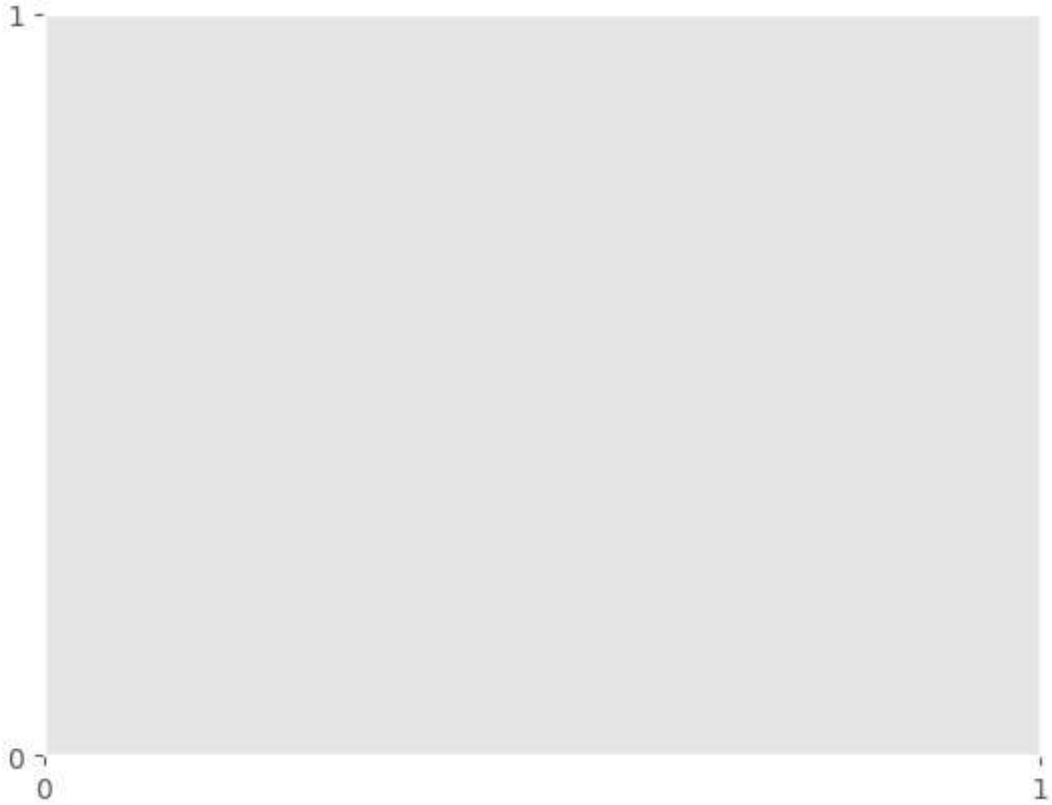
```
Accuracy: 0.8181818181818182
```

```
confusion_matrix=metrics.confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

```
[[98  9]
 [19 28]]
```

```
class_names=[0 , 1]
fig,ax=plt.subplots()
ticks_marks =  np.arange(len(class_names))
plt.xticks(ticks_marks,class_names)
plt.yticks(ticks_marks,class_names)
```

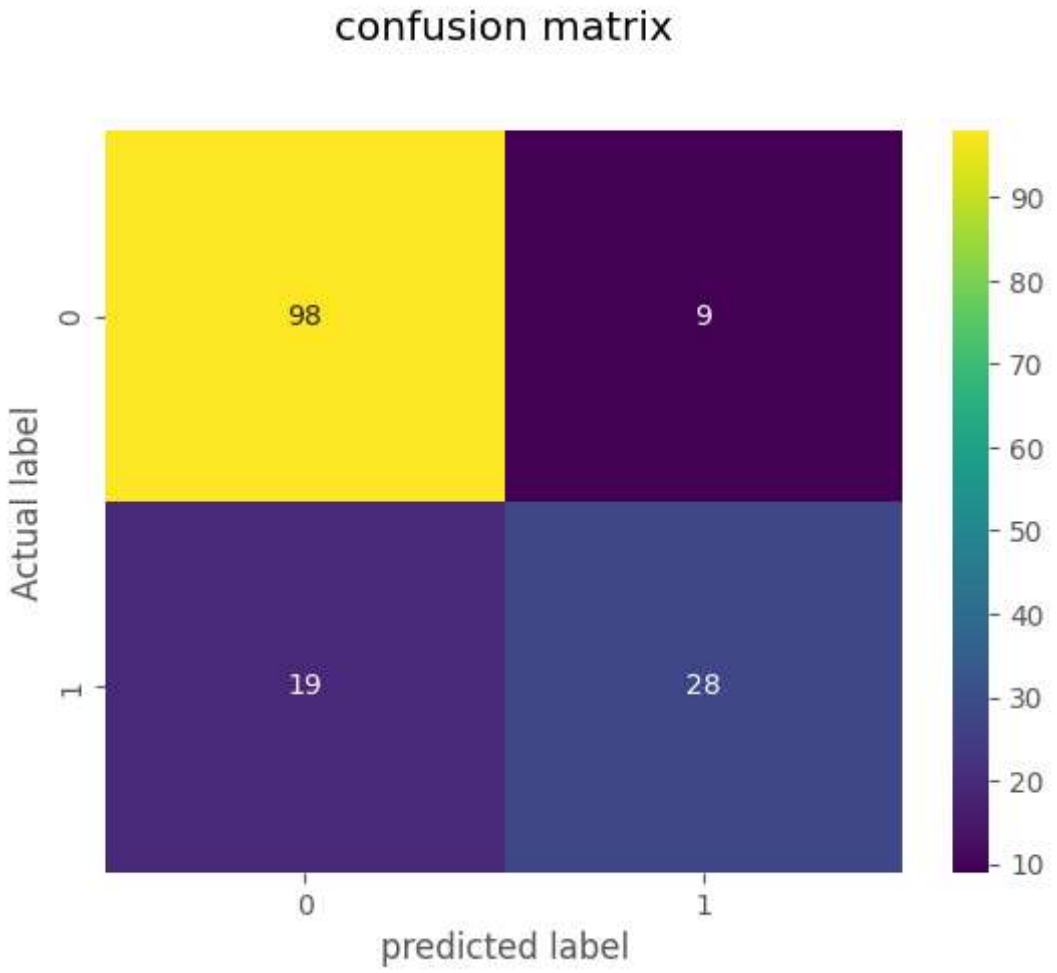
```
([<matplotlib.axis.YTick at 0x79e5446a8910>,
  <matplotlib.axis.YTick at 0x79e5446a8c70>],
 [Text(0, 0, '0'), Text(0, 1, '1')])
```




```
#create heatmap
sns.heatmap(pd.DataFrame(confusion_matrix),annot=True,cmap="viridis",fmt='g')
```

```
ax.xaxis.set_label_position("bottom")
plt.title('confusion matrix',y=1.1)
plt.ylabel('Actual label')
plt.xlabel('predicted label')
```

```
Text(0.5, 23.52222222222222, 'predicted label')
```



SVM Multicast

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn import datasets
from sklearn import svm
```

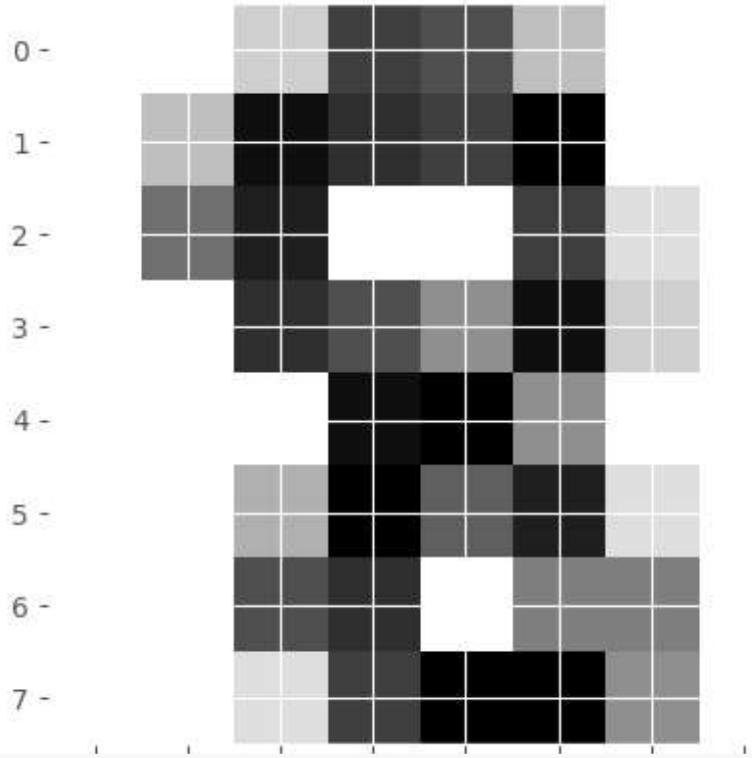
```
#load datasets
digits = datasets.load_digits()
dir(digits)
print(digits.data.shape)
print(digits.DESCR)
```

```
(1797, 64)
.. _digits_dataset:

Optical recognition of handwritten digits dataset
--------------------------------------------------
```

```
**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

```
#examine data
x = digits.images.reshape((len(digits.images), -1))
x.shape
y = digits.target
y_names = digits.target_names
print(y_names)
idx = 722
print(digits.images[idx])
print(digits.target[idx])
plt.imshow(digits.images[idx],cmap='binary')
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9]
[[ 0.  0.  3. 12. 11.  4.  0.  0.]
 [ 0.  4. 15. 13. 12. 16.  0.  0.]
 [ 0.  9. 14.  0.  0. 12.  2.  0.]
 [ 0.  0. 13. 11.  7. 15.  3.  0.]
 [ 0.  0.  0. 15. 16.  7.  0.  0.]
 [ 0.  0.  5. 16. 10. 14.  2.  0.]
 [ 0.  0. 11. 13.  0.  8.  8.  0.]
 [ 0.  0.  2. 12. 16. 16.  7.  0.]]
8
```




```
from scipy.sparse import random
X_train,x_test,y_train,y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
from sklearn.multiclass import OneVsRestClassifier
model = OneVsRestClassifier(svm.SVC())
```

```
model = svm.SVC(gamma = 0.001)
```

```
model.fit(X_train, y_train)
```

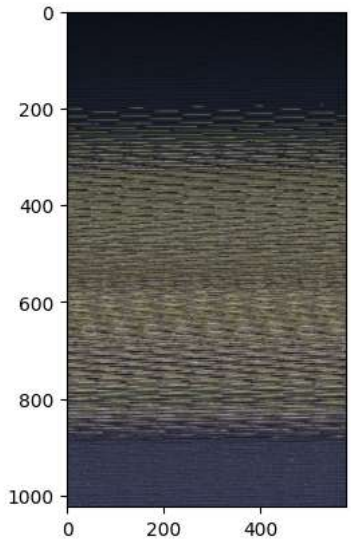
```
▾ SVC
SVC(gamma=0.001)
```

```
predictions = model.predict(x_test)
```

KMeans_EX

```
#import section
from matplotlib.image import imread
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import os
import numpy as np
```

```
# Load the image
image = Image.open("/content/pexels-pixabay-70083.jpg")
image = image.resize((1024, 576))
image = np.array(image)
image = image.reshape(1024, 576, 3)
plt.imshow(image / 255)
plt.show()
```

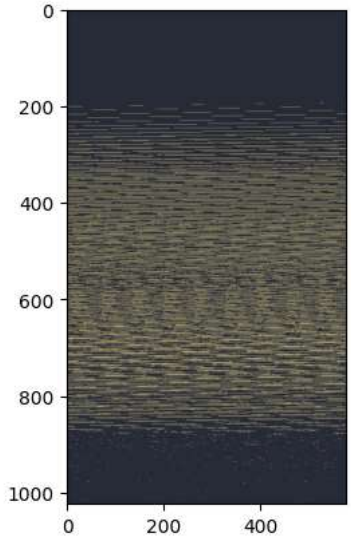



```
#Get X from the image
X = image.reshape(-1,3)
```

```
#use KMeans
kmeans = KMeans(n_clusters = 2).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```

```
#show segmented image
plt.imshow(segmented_img/255)
```

```
<matplotlib.image.AxesImage at 0x7c02702ffe80>
```



Binary Classification_Sk_NN

```
#import section
import sklearn
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
```

```
#load database
data = load_breast_cancer()
```

```
#examine data
x_names = data['feature_names']
y_names = data['target_names']
x = data['data']
y = data['target']
print(y_names)
print(x_names)
```

```
['malignant' 'benign']
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2,random_state=42)
```

```
mlp = MLPClassifier(hidden_layer_sizes=(200,), activation='logistic', alpha = 0.01, solve
```

```
mlp.fit(x_train,y_train)
```

```
Iteration 1, loss = 0.66423476
Iteration 2, loss = 0.62794413
Iteration 3, loss = 0.60989763
Iteration 4, loss = 0.59967550
Iteration 5, loss = 0.59427097
Iteration 6, loss = 0.58521329
Iteration 7, loss = 0.55965895
Iteration 8, loss = 0.55196990
Iteration 9, loss = 0.53932907
Iteration 10, loss = 0.52514628
Iteration 11, loss = 0.51722911
Iteration 12, loss = 0.50711408
Iteration 13, loss = 0.49787412
Iteration 14, loss = 0.49050556
Iteration 15, loss = 0.48752876
Iteration 16, loss = 0.47673948
Iteration 17, loss = 0.47993544
Iteration 18, loss = 0.46797245
Iteration 19, loss = 0.46441103
Iteration 20, loss = 0.45171874
Iteration 21, loss = 0.44151621
Iteration 22, loss = 0.43654456
Iteration 23, loss = 0.42940712
Iteration 24, loss = 0.42571736
Iteration 25, loss = 0.42065474
Iteration 26, loss = 0.41780376
Iteration 27, loss = 0.41532171
Iteration 28, loss = 0.40574464
Iteration 29, loss = 0.40541116
Iteration 30, loss = 0.40079164
Iteration 31, loss = 0.39621897
Iteration 32, loss = 0.38990156
Iteration 33, loss = 0.38724137
Iteration 34, loss = 0.38653433
Iteration 35, loss = 0.38053886
Iteration 36, loss = 0.37144850
Iteration 37, loss = 0.37419820
Iteration 38, loss = 0.36781577
Iteration 39, loss = 0.36358413
Iteration 40, loss = 0.38588840
Iteration 41, loss = 0.35720477
Iteration 42, loss = 0.35326649
Iteration 43, loss = 0.35139777
Iteration 44, loss = 0.35413033
Iteration 45, loss = 0.34662249
Iteration 46, loss = 0.34465118
Iteration 47, loss = 0.35473041
Iteration 48, loss = 0.33575499
Iteration 49, loss = 0.32968173
Iteration 50, loss = 0.33003552
Iteration 51, loss = 0.32563593
Iteration 52, loss = 0.32142269
Iteration 53, loss = 0.33429541
Iteration 54, loss = 0.32404276
Iteration 55, loss = 0.32392734
Iteration 56, loss = 0.32152923
Iteration 57, loss = 0.31658237
Iteration 58, loss = 0.30715736
Iteration 59, loss = 0.30706361
Iteration 60, loss = 0.31783154
```

```
Iteration 61, loss = 0.30688478
Iteration 62, loss = 0.30540081
Iteration 63, loss = 0.30700401
Iteration 64, loss = 0.30176876
Iteration 65, loss = 0.30186032
Iteration 66, loss = 0.30562930
Iteration 67, loss = 0.30408710
Iteration 68, loss = 0.29969768
Iteration 69, loss = 0.29255798
Iteration 70, loss = 0.30184832
Iteration 71, loss = 0.29473848
Iteration 72, loss = 0.30223574
Iteration 73, loss = 0.31478976
Iteration 74, loss = 0.28871198
Iteration 75, loss = 0.32356931
Iteration 76, loss = 0.29239020
Iteration 77, loss = 0.28854308
Iteration 78, loss = 0.31323790
Iteration 79, loss = 0.28649529
Iteration 80, loss = 0.28229284
Iteration 81, loss = 0.28659529
Iteration 82, loss = 0.29547102
Iteration 83, loss = 0.28898333
Iteration 84, loss = 0.27582272
Iteration 85, loss = 0.27657120
Iteration 86, loss = 0.27553380
Iteration 87, loss = 0.27425651
Iteration 88, loss = 0.27275920
Iteration 89, loss = 0.27612397
Iteration 90, loss = 0.27600615
Iteration 91, loss = 0.27009301
Iteration 92, loss = 0.27514190
Iteration 93, loss = 0.26935071
Iteration 94, loss = 0.26879087
Iteration 95, loss = 0.28248209
Iteration 96, loss = 0.27654794
Iteration 97, loss = 0.27641710
Iteration 98, loss = 0.26993826
Iteration 99, loss = 0.27454449
Iteration 100, loss = 0.27336509
Iteration 101, loss = 0.27683239
Iteration 102, loss = 0.26768910
Iteration 103, loss = 0.31394857
Iteration 104, loss = 0.27696649
Iteration 105, loss = 0.28943155
Iteration 106, loss = 0.26700490
Iteration 107, loss = 0.26183720
Iteration 108, loss = 0.26543810
Iteration 109, loss = 0.26998549
Iteration 110, loss = 0.26144614
Iteration 111, loss = 0.26681276
Iteration 112, loss = 0.26515997
Iteration 113, loss = 0.26467451
Iteration 114, loss = 0.25689668
Iteration 115, loss = 0.25912646
Iteration 116, loss = 0.27503798
Iteration 117, loss = 0.33879276
Iteration 118, loss = 0.27424440
Iteration 119, loss = 0.25668529
Iteration 120, loss = 0.29455947
```

```
Iteration 121, loss = 0.27781186
Iteration 122, loss = 0.31046773
Iteration 123, loss = 0.27691942
Iteration 124, loss = 0.26822355
Iteration 125, loss = 0.25596092
Iteration 126, loss = 0.25405997
Iteration 127, loss = 0.30208047
Iteration 128, loss = 0.26172731
Iteration 129, loss = 0.25153075
Iteration 130, loss = 0.27497685
Iteration 131, loss = 0.26104151
Iteration 132, loss = 0.26144125
Iteration 133, loss = 0.25332222
Iteration 134, loss = 0.27731017
```

```
predictions = mlp.predict(x_test)
```

```
Iteration 137, loss = 0.25555127
```

```
metrics.accuracy_score(y_test,predictions)
```

```
0.9210526315789473
```

MLPClassifier

```
MLPClassifier(activation='logistic', alpha=0.01, hidden_layer_sizes=(200,),
              random_state=1, solver='sgd', verbose=True)
```

KerasMultiClass_MNIST_Fashion

```
#import keras
import numpy as np
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.models import Sequential
from keras.optimizers import Adam
```

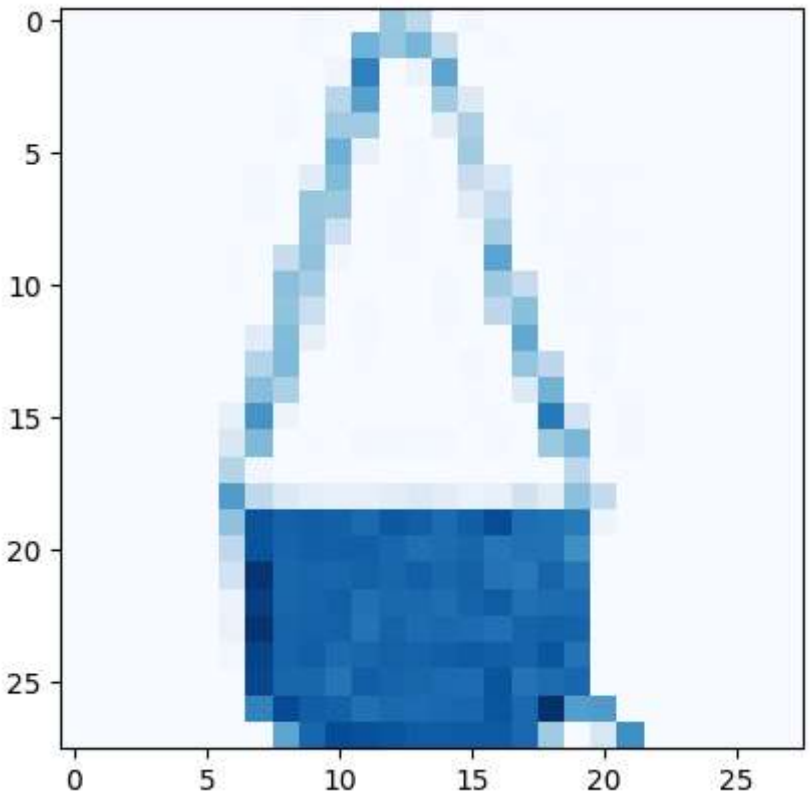
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train)
```

```
(60000, 28, 28)
(10000, 28, 28)
[9 0 0 ... 3 0 5]
```

```
import matplotlib.pyplot as plt
image_index = 1248
print(y_train[image_index])
plt.imshow(x_train[image_index], cmap = 'Blues')
plt.show()
```

```
8
```



```
model = Sequential([
    Flatten(input_shape=(28,28)),
    Dense(2048, activation = 'relu'),
    Dense(1024, activation = 'relu'),
    Dense(1024, activation = 'relu'),
    Dense(512, activation = 'relu'),
    Dense(10, activation = 'softmax'),
])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_5 (Dense)             (None, 2048)              1607680

 dense_6 (Dense)             (None, 1024)              2098176

 dense_7 (Dense)             (None, 1024)              1049600

 dense_8 (Dense)             (None, 512)               524800

 dense_9 (Dense)             (None, 10)                5130

=================================================================
Total params: 5285386 (20.16 MB)
Trainable params: 5285386 (20.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy', metrics = ['a
```

```
history = model.fit(x_train, y_train, epochs = 25, verbose = 2)
print(history.epoch, history.history['accuracy'][-1])
```

```
Epoch 1/25
1875/1875 - 6s - loss: 0.2475 - accuracy: 0.9105 - 6s/epoch - 3ms/step
Epoch 2/25
1875/1875 - 6s - loss: 0.2617 - accuracy: 0.9076 - 6s/epoch - 3ms/step
Epoch 3/25
1875/1875 - 5s - loss: 0.2470 - accuracy: 0.9115 - 5s/epoch - 3ms/step
Epoch 4/25
1875/1875 - 6s - loss: 0.2444 - accuracy: 0.9126 - 6s/epoch - 3ms/step
Epoch 5/25
1875/1875 - 6s - loss: 0.2481 - accuracy: 0.9109 - 6s/epoch - 3ms/step
Epoch 6/25
1875/1875 - 6s - loss: 0.2421 - accuracy: 0.9127 - 6s/epoch - 3ms/step
Epoch 7/25
1875/1875 - 5s - loss: 0.2613 - accuracy: 0.9104 - 5s/epoch - 3ms/step
Epoch 8/25
1875/1875 - 6s - loss: 0.2397 - accuracy: 0.9158 - 6s/epoch - 3ms/step
Epoch 9/25
```

```
1875/1875 - 6s - loss: 0.2315 - accuracy: 0.9167 - 6s/epoch - 3ms/step
Epoch 10/25
1875/1875 - 6s - loss: 0.2549 - accuracy: 0.9135 - 6s/epoch - 3ms/step
Epoch 11/25
1875/1875 - 6s - loss: 0.2432 - accuracy: 0.9162 - 6s/epoch - 3ms/step
Epoch 12/25
1875/1875 - 6s - loss: 0.2255 - accuracy: 0.9191 - 6s/epoch - 3ms/step
Epoch 13/25
1875/1875 - 6s - loss: 0.2285 - accuracy: 0.9182 - 6s/epoch - 3ms/step
Epoch 14/25
1875/1875 - 5s - loss: 0.2257 - accuracy: 0.9184 - 5s/epoch - 3ms/step
Epoch 15/25
1875/1875 - 6s - loss: 0.2320 - accuracy: 0.9173 - 6s/epoch - 3ms/step
Epoch 16/25
1875/1875 - 6s - loss: 0.2205 - accuracy: 0.9214 - 6s/epoch - 3ms/step
Epoch 17/25
1875/1875 - 6s - loss: 0.2510 - accuracy: 0.9160 - 6s/epoch - 3ms/step
Epoch 18/25
1875/1875 - 5s - loss: 0.2150 - accuracy: 0.9220 - 5s/epoch - 3ms/step
Epoch 19/25
1875/1875 - 6s - loss: 0.2153 - accuracy: 0.9215 - 6s/epoch - 3ms/step
Epoch 20/25
1875/1875 - 6s - loss: 0.2656 - accuracy: 0.9105 - 6s/epoch - 3ms/step
Epoch 21/25
1875/1875 - 5s - loss: 0.2111 - accuracy: 0.9244 - 5s/epoch - 3ms/step
Epoch 22/25
1875/1875 - 6s - loss: 0.2209 - accuracy: 0.9217 - 6s/epoch - 3ms/step
Epoch 23/25
1875/1875 - 5s - loss: 0.2091 - accuracy: 0.9249 - 5s/epoch - 3ms/step
Epoch 24/25
1875/1875 - 6s - loss: 0.2151 - accuracy: 0.9241 - 6s/epoch - 3ms/step
Epoch 25/25
1875/1875 - 5s - loss: 0.4296 - accuracy: 0.9174 - 5s/epoch - 3ms/step
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23
```

```
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.5325 - accuracy: 0.8
[0.5325161814689636, 0.8888000249862671]
```

Feature-engine

```
pip install feature-engine
```

```
Collecting feature-engine
  Downloading feature_engine-1.6.2-py2.py3-none-any.whl (328 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 328.9/328.9 kB 2.1 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.23.5)
Requirement already satisfied: pandas>=1.0.3 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.5.3)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.2.2)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (1.11.4)
Requirement already satisfied: statsmodels>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from feature-engine) (0.14.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.3->feature-engine
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.3->feature-engine) (2023.3
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->feature-engine)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->feature-en
Requirement already satisfied: patsy>=0.5.4 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.11.1->feature-engine) (0
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.11.1->feature-engine
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.4->statsmodels>=0.11.1->feature-engin
Installing collected packages: feature-engine
Successfully installed feature-engine-1.6.2
```

```
import pandas as pd
import numpy as np
from feature_engine.imputation import MeanMedianImputer
from feature_engine.imputation import ArbitraryNumberImputer
```

```
X = pd.DataFrame(dict(
       x1 = [np.nan,1,1,0,np.nan],
       x2 = ["a", np.nan, "b", np.nan, "a"],
       ))
mmi = MeanMedianImputer(imputation_method='median')
mmi.fit(X)
mmi.transform(X)
```

	x1	x2
0	1.0	a
1	1.0	NaN
2	1.0	b
3	0.0	NaN
4	1.0	a

```
#linear regression Feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
```

```
#define dataset
X,y = make_regression(n_samples = 1000, n_features = 10, n_informative  = 5, random_state=1)
print(X)
print(y)
```

```
-1.85522340e+02 -1.46659525e+02  1.90527552e+02 -1.78451438e+02
 5.74887734e+01  6.22650526e+01 -2.92632921e+02  8.97522801e+01
-1.91354522e+01 -1.64150085e+02 -1.87321662e+02 -7.44857575e+01
-8.04354162e+01  6.86452731e+01 -1.55766601e+01  9.18264690e+01
 6.67782077e+01  9.07662401e+01  1.05390570e+02 -1.75013182e+02
-6.60974494e+01 -9.06738270e+01  1.57725742e+02 -1.65912914e+02
-5.64399414e+01 -6.65081460e+01 -7.26406181e+01 -5.10926612e+01
-3.49734549e+01  3.55960072e+01  1.61116562e+02  5.67937636e+01
-2.35856803e+02  1.22714706e+01 -7.60215593e+01  3.29527602e+02
-1.12163956e+02  4.74364226e+01  1.41043812e+02  1.20192624e+02
 3.17254522e+01 -8.06723696e+01  1.67454591e+02 -4.48007186e+01
 8.06408356e+00  1.41211761e+02 -1.41676026e+02 -1.13483950e+01
 2.47618814e+01  9.12720109e+01  8.60001871e+01  1.75376031e+02
-5.83973677e+00  2.03129861e+02  1.55768727e+02  1.50890858e+02
 7.56363249e+00 -1.22252092e+01 -9.23299302e+01  6.44178620e+01
 1.27934359e+02  3.22574232e+02  9.25429377e+01 -8.50578660e+01
-4.21236353e+01  8.07969166e+01  1.76883323e+02  5.96441341e+01
-1.86196788e+02  1.98198986e+02  9.29990604e+01 -9.66954765e+01
-1.12997672e+02 -1.47062667e+02 -1.44604461e+01 -5.55209941e+01
-1.53948970e+02 -8.20590622e+01 -5.20716713e+01 -1.10074497e+02
 2.24021702e+02 -1.66007119e+01  2.24193193e+02  1.92250976e+01
 2.73847938e+02 -7.32042966e+01 -8.52798932e+01 -9.48820083e+01
 2.21717791e+01 -1.61220482e+02  1.45967873e+02 -3.09994276e+02
-2.02028034e+02 -1.12219977e+02 -2.08661308e+02  1.08242338e+02
-1.47533564e+02 -1.29558556e+02  1.71483047e+01 -5.99546433e+01
-6.60454409e+01 -8.08894870e+01 -2.17055691e+02  1.44635884e+02
 1.08663240e+02 -3.62222925e+00 -1.21494745e+02 -5.61510065e+01
-1.94774346e+02  1.05574500e+02  1.69434490e+02  2.53386403e+01
-2.49879691e+02  5.47612021e+01 -3.89924018e+00  5.98817745e+01
-5.35324973e+01  1.36424075e+01  4.36980268e+01  8.68697845e+01
 1.41802193e+02  1.56101208e+02  8.02513759e+01 -1.14317614e+01
 1.23067318e+02 -4.83287115e+01 -2.21956103e+01 -7.48781802e+01
 2.30816006e+02  4.51240930e+01 -6.15724583e+01 -3.50107772e+02
 7.33796770e+01  7.97852933e+01  1.32517589e+01 -6.21098841e-02
-1.00649480e+02  1.44254869e+02  2.35376670e+01  1.17149745e+01
-1.49851426e+02  2.47278908e+02  5.35253152e+01  1.12843191e+02
```

```
# define the model
model = LinearRegression()
```

```
#fit the model
model.fit(X,y)
```

```
LinearRegression()
```

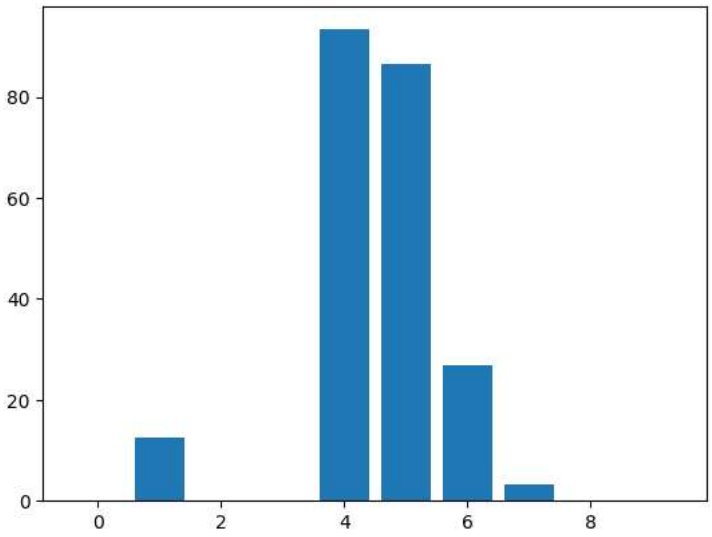
```
#get importance
importance  = model.coef_
```

```
#summarize feature importance
for i,v in enumerate(importance):
  print('Feature: %0d, Score: %.5f' % (i,v))
```

```
Feature: 0, Score: 0.00000
Feature: 1, Score: 12.44483
Feature: 2, Score: -0.00000
Feature: 3, Score: -0.00000
Feature: 4, Score: 93.32225
Feature: 5, Score: 86.50811
Feature: 6, Score: 26.74607
Feature: 7, Score: 3.28535
Feature: 8, Score: -0.00000
Feature: 9, Score: 0.00000
```

```
#plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
```

```
<BarContainer object of 10 artists>
```



PCA

```
from numpy import array
from numpy import mean
from numpy import cov
from numpy.linalg import eig
```

```
#define matrix
A = array([[1,2],[3,4],[5,6]])
print(A)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

```
#Calculate the mean of each column
M = mean(A.T, axis=1)
print(M)
print(A.T)
```

```
[3. 4.]
[[1 3 5]
 [2 4 6]]
```

```
#center columns by substracting column means
C = A-M
print(C)
```

```
[[-2. -2.]
 [ 0.  0.]
 [ 2.  2.]]
```

```
#calculate covariance matrix of centerred matrix
V = cov(C.T)
print(V)
```

```
[[4. 4.]
 [4. 4.]]
```

```
#eigender composition of covariance matrix
values, vectors = eig(V)
print(vectors)
print(values)
```

```
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
[8. 0.]
```

```
#project data
P = vectors.T.dot(C.T)
print(P.T)
```

```
[[-2.82842712  0.        ]
 [ 0.          0.        ]
 [ 2.82842712  0.        ]]
```

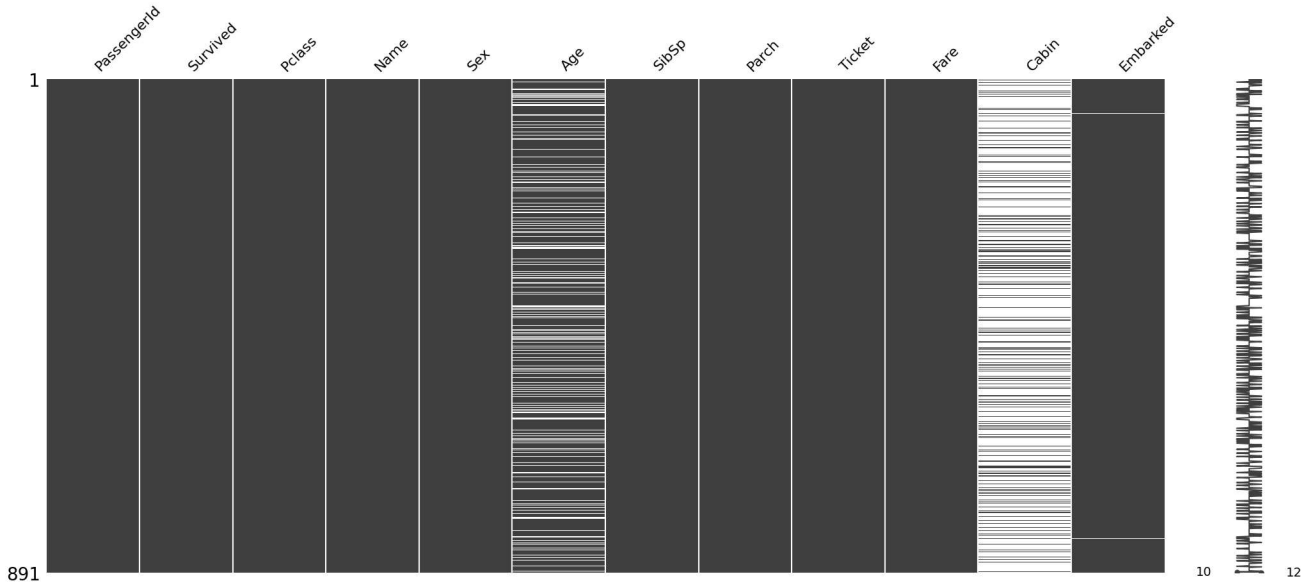
7 Ways to Handle Missing Values

```
from sklearn.linear_model import LinearRegression
import pandas as pd
import numpy as np
import missingno as msno
from feature_engine.imputation import MeanMedianImputer
```

```
data = pd.read_csv("train.csv")
```

```
msno.matrix(data)
```

```
<Axes: >
```




```
# print(data.isnull().sum())
# print(data.shape)
```

```
# data.dropna(inplace=True)
# print(data.isnull().sum())
# print(data.shape)
```

```
# data["Age"][:20]
```

```
# data["Age"] = data["Age"].replace(np.NaN, data["Age"].mean())
# print(data["Age"][:20])
```

```
data.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
```

```
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin         687
Embarked        2
dtype: int64
```

```
data["Cabin"] = data["Cabin"].fillna('U')
```

```
data.isnull().sum()
```

```
PassengerId     0
Survived        0
Pclass          0
Name            0
Sex             0
Age           177
SibSp           0
Parch           0
Ticket          0
Fare            0
Cabin           0
Embarked        2
dtype: int64
```

```
print(data["Age"][:20])
```

```
0     22.0
1     38.0
2     26.0
3     35.0
4     35.0
5      NaN
6     54.0
7      2.0
8     27.0
9     14.0
10     4.0
11    58.0
12    20.0
13    39.0
14    14.0
15    55.0
16     2.0
17     NaN
18    31.0
19     NaN
Name: Age, dtype: float64
```

```
data["Age"] = data["Age"].fillna(method='ffill')
```

```
print(data["Age"][:20])
```

```
0     22.0
1     38.0
2     26.0
3     35.0
4     35.0
5     35.0
6     54.0
7      2.0
8     27.0
9     14.0
10     4.0
11    58.0
12    20.0
13    39.0
14    14.0
15    55.0
16     2.0
17     2.0
18    31.0
19    31.0
Name: Age, dtype: float64
```

```
data["Age"] = data["Age"].interpolate(method='linear', limit_direction='forward', axis=0)
```

```
data.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
Sex            0
Age            0
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin          0
Embarked       2
dtype: int64
```

```
# data["Sex"] = [1 if x=="male" else 0 for x in data["Sex"]]
# print(data)
```

```
# test_data = data[data["Age"].isnull()]
# print(test_data)
# data.dropna(inplace=True)
```

```
# y_train = data["Age"]
# print(y_train)
# X_train = data.drop("Age", axis=1)
# X_test = test_data.drop("Age", axis=1)
```

```
# model = LinearRegression()
# model.fit(X_train, y_train)
```

```
# y_pred = model.predict(X_test)
# print(y_pred)
```

```
pip install feature-engine
```

```
Requirement already satisfied: feature-engine in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pandas>=1.0.3 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: statsmodels>=0.11.1 in /usr/local/lib/python3.10/di
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from p
```

```
import pandas as pd
import numpy as np
from feature_engine.imputation import MeanMedianImputer
```

```
X = pd.DataFrame(dict(
       x1 = [np.nan,1,1,0,np.nan],
       x2 = ["a", np.nan, "b", np.nan, "a"],
       ))
mmi = MeanMedianImputer(imputation_method='median')
mmi.fit(X)
mmi.transform(X)
```

	x1	x2
0	1.0	a
1	1.0	NaN
2	1.0	b
3	0.0	NaN
4	1.0	a

```
mmi = MeanMedianImputer(imputation_method='median', variables=['Age'])
data['Age'] = mmi.fit_transform(data[['Age']])
print(data.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
```

```
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500     U        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250     U        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500     U        S
```

DimensionalityReduction

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import StandardScaler
```

```
#load the data
iris = datasets.load_iris()
X=iris.data
Y=iris.target
```

```
#Z-score the features
scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
```

```
#PCA model
pca = PCA(n_components=2) #estimate only 2 pcs
X_new = pca.fit_transform(X) #projets the original data into PCA space
print(abs(pca.components_))
```

```
[[0.52106591 0.26934744 0.5804131  0.56485654]
 [0.37741762 0.92329566 0.02449161 0.06694199]]
```

ANN_Binary_Diabeties

```
from numpy import loadtxt
import tensorflow as tf
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense
```

```
dataset = pd.read_csv('/content/diabetes2.csv')
```

```
dataset.isnull().any()
dataset = dataset.fillna(method='ffill')
```

```
X = dataset[['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']].values
y = dataset['Outcome'].values
```

```
model = Sequential([
        Dense(12, input_shape=(8,),activation='relu'),
        Dense(8, activation='relu'),
        Dense(1,activation='sigmoid')])
```

```
model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
```

```
history = model.fit(X,y,validation_split=0.33, epochs =300, batch_size=5, verbose= 1)
```

```
Epoch 1/300
103/103 [==============================] - 1s 6ms/step - loss: 3.2927 - accuracy: 0.4650 - val_loss: 1.5679 - val_accuracy: 0.476
Epoch 2/300
103/103 [==============================] - 0s 4ms/step - loss: 1.4333 - accuracy: 0.5700 - val_loss: 1.0660 - val_accuracy: 0.618
Epoch 3/300
103/103 [==============================] - 0s 5ms/step - loss: 1.0426 - accuracy: 0.5953 - val_loss: 0.8291 - val_accuracy: 0.673
Epoch 4/300
103/103 [==============================] - 0s 5ms/step - loss: 0.9710 - accuracy: 0.6167 - val_loss: 1.0742 - val_accuracy: 0.618
Epoch 5/300
103/103 [==============================] - 0s 4ms/step - loss: 0.8846 - accuracy: 0.6284 - val_loss: 0.7509 - val_accuracy: 0.566
Epoch 6/300
103/103 [==============================] - 0s 5ms/step - loss: 0.8827 - accuracy: 0.6226 - val_loss: 0.7372 - val_accuracy: 0.673
Epoch 7/300
103/103 [==============================] - 0s 4ms/step - loss: 0.7597 - accuracy: 0.6712 - val_loss: 0.6667 - val_accuracy: 0.661
Epoch 8/300
103/103 [==============================] - 0s 4ms/step - loss: 0.7668 - accuracy: 0.6595 - val_loss: 0.7299 - val_accuracy: 0.618
Epoch 9/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6851 - accuracy: 0.6732 - val_loss: 0.7036 - val_accuracy: 0.574
Epoch 10/300
103/103 [==============================] - 0s 5ms/step - loss: 0.7076 - accuracy: 0.6848 - val_loss: 0.6597 - val_accuracy: 0.637
Epoch 11/300
103/103 [==============================] - 0s 4ms/step - loss: 0.6692 - accuracy: 0.6984 - val_loss: 0.7209 - val_accuracy: 0.696
Epoch 12/300
103/103 [==============================] - 0s 5ms/step - loss: 0.7291 - accuracy: 0.6615 - val_loss: 0.6558 - val_accuracy: 0.637
Epoch 13/300
103/103 [==============================] - 0s 4ms/step - loss: 0.7041 - accuracy: 0.6576 - val_loss: 0.7056 - val_accuracy: 0.622
Epoch 14/300
103/103 [==============================] - 0s 5ms/step - loss: 0.7374 - accuracy: 0.6498 - val_loss: 0.6219 - val_accuracy: 0.720
Epoch 15/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6628 - accuracy: 0.6887 - val_loss: 0.6579 - val_accuracy: 0.633
Epoch 16/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6554 - accuracy: 0.7140 - val_loss: 0.6022 - val_accuracy: 0.712
Epoch 17/300
103/103 [==============================] - 0s 4ms/step - loss: 0.6694 - accuracy: 0.6732 - val_loss: 0.6665 - val_accuracy: 0.696
Epoch 18/300
103/103 [==============================] - 0s 5ms/step - loss: 0.7099 - accuracy: 0.6615 - val_loss: 0.6332 - val_accuracy: 0.712
Epoch 19/300
103/103 [==============================] - 1s 6ms/step - loss: 0.7219 - accuracy: 0.6654 - val_loss: 0.8562 - val_accuracy: 0.563
Epoch 20/300
103/103 [==============================] - 1s 6ms/step - loss: 0.7787 - accuracy: 0.6615 - val_loss: 0.7454 - val_accuracy: 0.673
Epoch 21/300
103/103 [==============================] - 1s 6ms/step - loss: 0.6601 - accuracy: 0.6907 - val_loss: 0.6935 - val_accuracy: 0.618
Epoch 22/300
103/103 [==============================] - 1s 6ms/step - loss: 0.6835 - accuracy: 0.6984 - val_loss: 0.6343 - val_accuracy: 0.661
Epoch 23/300
103/103 [==============================] - 1s 5ms/step - loss: 0.6443 - accuracy: 0.7179 - val_loss: 0.6480 - val_accuracy: 0.665
Epoch 24/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6209 - accuracy: 0.6887 - val_loss: 0.6359 - val_accuracy: 0.689
Epoch 25/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6602 - accuracy: 0.6829 - val_loss: 0.6228 - val_accuracy: 0.700
Epoch 26/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6233 - accuracy: 0.7082 - val_loss: 0.6269 - val_accuracy: 0.657
Epoch 27/300
103/103 [==============================] - 0s 5ms/step - loss: 0.6395 - accuracy: 0.6887 - val_loss: 0.8517 - val_accuracy: 0.696
```

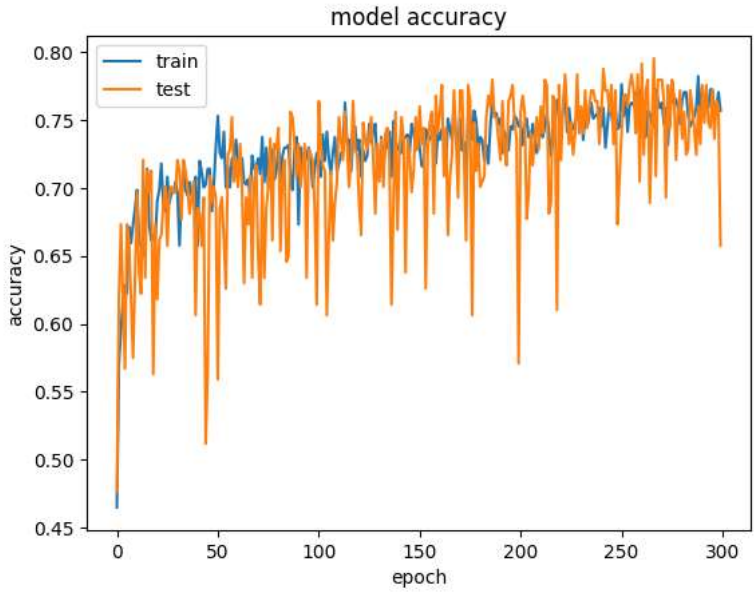
```
Epoch 28/300
103/103 [==============================] - 0s 4ms/step - loss: 0.6890 - accuracy: 0.6965 - val_loss: 0.6170 - val_accuracy: 0.700
```

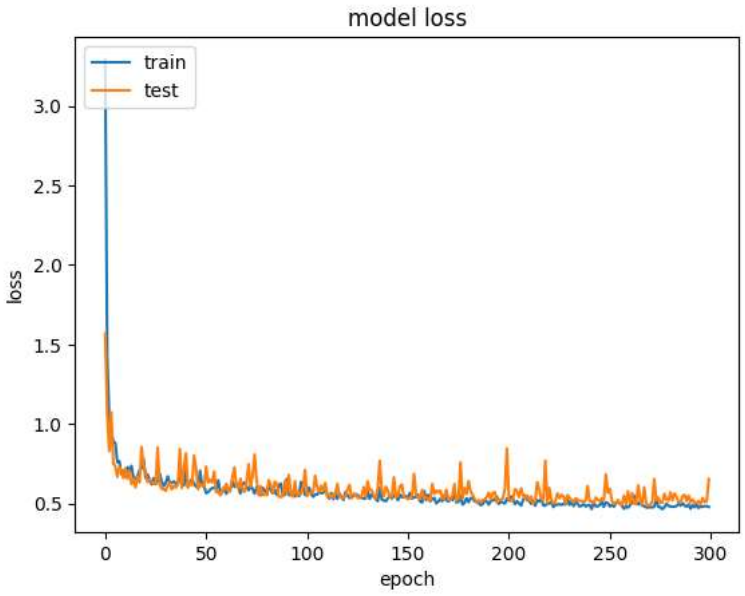
```
_, accuracy = model.evaluate(X,y)
print('Accuracy: %.2f'%(accuracy*100))
```

```
24/24 [==============================] - 0s 3ms/step - loss: 0.6108 - accuracy: 0.7018
Accuracy: 70.18
```

```
import matplotlib.pyplot as plt
print(history.history.keys())
#sumarise history from accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
#sumarise history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train','test'],loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





CNN_Convolution_n_Pooling

```
#import
import cv2
import numpy as np
from scipy import misc
i = misc.ascent()
#i = misc.face
```

```
<ipython-input-8-81001d7af789>:5: DeprecationWarning: scipy.misc.ascent has been deprecated in SciPy v1.10.0; and will be complet
  i = misc.ascent()
```

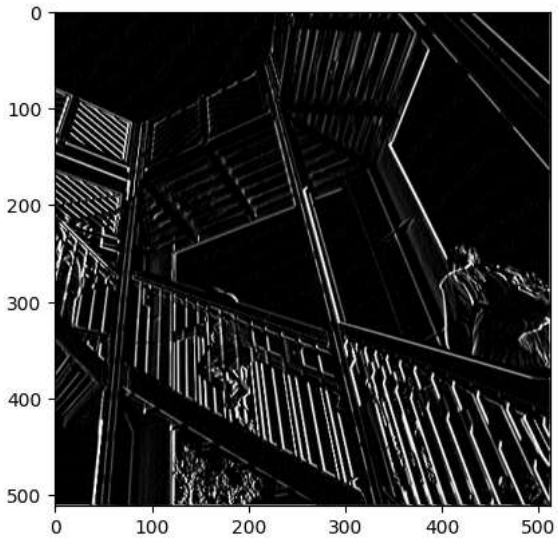
```
i_transformed = np.copy(i)
size_x = i_transformed.shape[0]
size_y = i_transformed.shape[1]
```

```
#plot
import matplotlib.pyplot as plt
plt.grid(False)
plt.gray()
plt.axis('off')
plt.imshow(i)
plt.show()
```

```
#filter1
#filter = [[0,1,0],[1,-4,1],[0,1,0]]

#filter2
filter = [[-1,-2,-1],[0,0,0],[1,2,1]]
#filter = [[-1,0,1],[-2,0,2],[-1,0,1]]
weight = 1
```

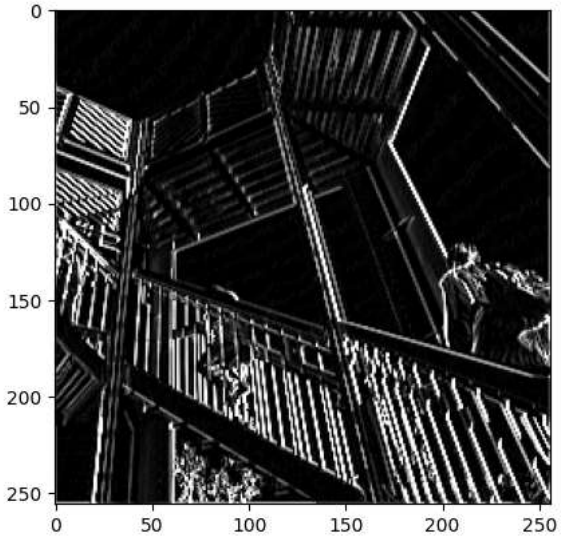
```
for x in range(1,size_x-1):
  for y in range(1,size_y-1):
    convolution = 0.0
    convolution = convolution + (i[x - 1,y-1]*filter[0][0])
    convolution = convolution + (i[x,y-1]*filter[0][1])
    convolution = convolution + (i[x + 1,y-1]*filter[0][2])
    convolution = convolution + (i[x - 1,y]*filter[1][0])
    convolution = convolution + (i[x,y]*filter[1][1])
    convolution = convolution + (i[x + 1,y]*filter[1][2])
    convolution = convolution + (i[x - 1,y+1]*filter[2][0])
    convolution = convolution + (i[x,y+1]*filter[2][1])
    convolution = convolution + (i[x + 1,y+1]*filter[2][2])
    convolution = convolution * weight
    if(convolution<0):
      convolution=0
    if(convolution>255):
      convolution=255
    i_transformed[x,y] = convolution
```

```
#let's draw transformed image
plt.gray()
plt.grid(False)
plt.imshow(i_transformed)
plt.show()
```




```
#pooling example
new_x = int(size_x/2)
new_y = int(size_y/2)
newImage = np.zeros((new_x, new_y))
for x in range(0, size_x,2):
  for y in range(0, size_y,2):
    pixels = []
    pixels.append(i_transformed[x,y])
    pixels.append(i_transformed[x+1,y])
    pixels.append(i_transformed[x,y+1])
    pixels.append(i_transformed[x+1,y+1])
    newImage[int(x/2),int(y/2)] = max(pixels)

print(newImage.shape)
#plot the image. Note the size of the axess --new 256 pixels instead of 512
plt.grid(False)
plt.gray()
#plt.axis('off')
plt.imshow(newImage)
plt.show()
```

```
(256, 256)
```



CNNEx1_CatsDogs

Source:https://www.kaggle.com/c/dogs-vs-cats/data

```
!wget --no-check-certificate \
  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip \
  -O /tmp/cats_and_dogs_filtered.zip
```

```
--2023-12-18 05:46:32--  https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
Resolving storage.googleapis.com (storage.googleapis.com)... 172.217.212.207, 142.250.128.207, 172.253.114.207, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|172.217.212.207|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 68606236 (65M) [application/zip]
Saving to: ‘/tmp/cats_and_dogs_filtered.zip’

/tmp/cats_and_dogs_ 100%[===================>]  65.43M   187MB/s    in 0.3s

2023-12-18 05:46:33 (187 MB/s) - ‘/tmp/cats_and_dogs_filtered.zip’ saved [68606236/68606236]
```

```
#extract files from zip to /tmp
import os
import zipfile

local_zip = '/tmp/cats_and_dogs_filtered.zip'

zip_ref = zipfile.ZipFile(local_zip, 'r')

zip_ref.extractall('/tmp')
zip_ref.close()
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

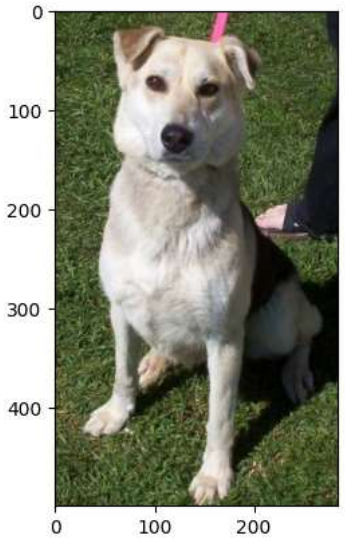
```
#create necessary directories to split data
base_dir = '/tmp/cats_and_dogs_filtered'

train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'validation')

# train directories
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# test directories
test_cats_dir = os.path.join(test_dir, 'cats')
test_dogs_dir = os.path.join(test_dir, 'dogs')
```

```
#Check data shape
print('Training (Cat) :', len(os.listdir(train_cats_dir ) ))
print('Training (Dog) :', len(os.listdir(train_dogs_dir ) ))

print('Testing (Cat) :', len(os.listdir(test_cats_dir ) ))
print('Testing (Dog) :', len(os.listdir(test_dogs_dir ) ))
#Expected ouput
#1000
#1000
#500
#500
```

```
Training (Cat) : 1000
Training (Dog) : 1000
Testing (Cat) : 500
Testing (Dog) : 500
```

```
#See the cats and dogs
%matplotlib inline

import matplotlib.image as mpimg
import matplotlib.pyplot as plt
#img_path=os.path.join(train_cats_dir, 'cat.40.jpg')
img_path=os.path.join(train_dogs_dir, 'dog.90.jpg')
img = mpimg.imread(img_path)
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7d35a4161a20>
```




```
#Generate the data on the fly
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# reshape the image
train_datagen = ImageDataGenerator( rescale = 1.0/255. )
test_datagen  = ImageDataGenerator( rescale = 1.0/255. )

#batch_size = 20 and target_size = 150x150
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

test_generator =  test_datagen.flow_from_directory(test_dir,
                                                   batch_size=20,
                                                   class_mode  = 'binary',
                                                   target_size = (150, 150))
```

```
Found 2000 images belonging to 2 classes.
Found 1000 images belonging to 2 classes.
```

```
#define the model
import tensorflow as tf
model = tf.keras.models.Sequential([
    #Conv2D adds a convulation layer, 16: number of filters(https://lodev.org/cgtutor/filtering.html), (3,3) convolution
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)), #images are 150x150 RGB(3)
    tf.keras.layers.MaxPooling2D(2,2),                   #MaxPooling2D will reduce the dimension
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'), #another conv layer with 32 filters
    tf.keras.layers.MaxPooling2D(2,2),                   #another pooling layer
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'), #another conv layer with 64 filters
    tf.keras.layers.MaxPooling2D(2,2),                   #anotehr pooling layers
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),        #512 neurons
    tf.keras.layers.Dense(1, activation='sigmoid')        #0 for cats and 1 for dogs (Binary classifier)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2  (None, 74, 74, 16)        0
 D)
```

```
 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 512)               9470464

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 9494561 (36.22 MB)
Trainable params: 9494561 (36.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

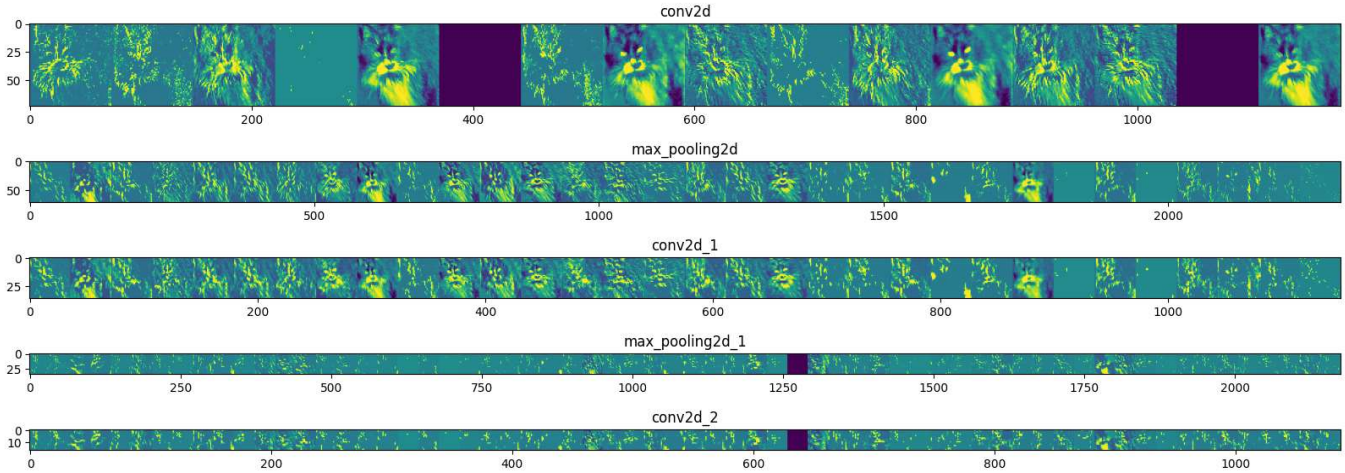
```
#compile with RMSprop: RMS = Root Mean Squared
from tensorflow.keras.optimizers import RMSprop

model.compile(optimizer=RMSprop(learning_rate=0.001),
              loss='binary_crossentropy',
              metrics = ['accuracy'])
```

```
#Fit the model with 15 epochs
history = model.fit(train_generator,
                              validation_data=test_generator,
                              steps_per_epoch=100,
                              epochs=15,
                              validation_steps=50,
                              verbose=2)
```

```
Epoch 1/15
100/100 - 5s - loss: 9.7066e-07 - accuracy: 1.0000 - val_loss: 2.7263 - val_accuracy: 0.7410 - 5s/epoch - 45ms/step
Epoch 2/15
100/100 - 6s - loss: 8.6776e-07 - accuracy: 1.0000 - val_loss: 2.7479 - val_accuracy: 0.7410 - 6s/epoch - 61ms/step
Epoch 3/15
100/100 - 4s - loss: 7.8723e-07 - accuracy: 1.0000 - val_loss: 2.7655 - val_accuracy: 0.7430 - 4s/epoch - 45ms/step
Epoch 4/15
100/100 - 4s - loss: 7.2243e-07 - accuracy: 1.0000 - val_loss: 2.7792 - val_accuracy: 0.7430 - 4s/epoch - 45ms/step
Epoch 5/15
100/100 - 6s - loss: 6.6388e-07 - accuracy: 1.0000 - val_loss: 2.7947 - val_accuracy: 0.7430 - 6s/epoch - 61ms/step
Epoch 6/15
100/100 - 4s - loss: 6.1564e-07 - accuracy: 1.0000 - val_loss: 2.8087 - val_accuracy: 0.7430 - 4s/epoch - 45ms/step
Epoch 7/15
100/100 - 9s - loss: 5.8168e-07 - accuracy: 1.0000 - val_loss: 2.8193 - val_accuracy: 0.7430 - 9s/epoch - 92ms/step
Epoch 8/15
100/100 - 5s - loss: 5.4558e-07 - accuracy: 1.0000 - val_loss: 2.8311 - val_accuracy: 0.7400 - 5s/epoch - 45ms/step
Epoch 9/15
100/100 - 6s - loss: 5.1336e-07 - accuracy: 1.0000 - val_loss: 2.8411 - val_accuracy: 0.7390 - 6s/epoch - 60ms/step
Epoch 10/15
100/100 - 4s - loss: 4.7973e-07 - accuracy: 1.0000 - val_loss: 2.8551 - val_accuracy: 0.7430 - 4s/epoch - 45ms/step
Epoch 11/15
100/100 - 6s - loss: 4.6241e-07 - accuracy: 1.0000 - val_loss: 2.8623 - val_accuracy: 0.7390 - 6s/epoch - 61ms/step
Epoch 12/15
100/100 - 4s - loss: 4.4097e-07 - accuracy: 1.0000 - val_loss: 2.8706 - val_accuracy: 0.7390 - 4s/epoch - 44ms/step
Epoch 13/15
100/100 - 4s - loss: 4.2050e-07 - accuracy: 1.0000 - val_loss: 2.8817 - val_accuracy: 0.7390 - 4s/epoch - 45ms/step
Epoch 14/15
100/100 - 6s - loss: 4.0507e-07 - accuracy: 1.0000 - val_loss: 2.8901 - val_accuracy: 0.7390 - 6s/epoch - 61ms/step
Epoch 15/15
100/100 - 5s - loss: 3.8955e-07 - accuracy: 1.0000 - val_loss: 2.8989 - val_accuracy: 0.7390 - 5s/epoch - 45ms/step
```

```
#The internal process
import numpy as np
import random
from   tensorflow.keras.preprocessing.image import img_to_array, load_img


successive_outputs = [layer.output for layer in model.layers[1:]]

visualization_model = tf.keras.models.Model(inputs = model.input, outputs = successive_outputs)

cat_img_files = [os.path.join(train_cats_dir, f) for f in ['cat.0.jpg','cat.10.jpg','cat.25.jpg','cat.40.jpg'
                                                           ,'cat.100.jpg']]
dog_img_files = [os.path.join(train_dogs_dir, f) for f in ['dog.0.jpg','dog.10.jpg','dog.25.jpg','dog.40.jpg'
                                                           ,'dog.100.jpg']]

img_path = random.choice(cat_img_files + dog_img_files)
img = load_img(img_path, target_size=(150, 150))

x   = img_to_array(img)
x   = x.reshape((1,) + x.shape)

x /= 255.0

successive_feature_maps = visualization_model.predict(x)

layer_names = [layer.name for layer in model.layers]


for layer_name, feature_map in zip(layer_names, successive_feature_maps):

  if len(feature_map.shape) == 4:

    n_features = feature_map.shape[-1]
    size       = feature_map.shape[ 1]

    display_grid = np.zeros((size, size * n_features))

    for i in range(n_features):
      x  = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std ()
      x *=  64
      x += 128
      x  = np.clip(x, 0, 255).astype('uint8')
      display_grid[:, i * size : (i + 1) * size] = x # Tile each filter into a horizontal grid

    scale = 20. / n_features
    plt.figure( figsize=(scale * n_features, scale) )
    plt.title ( layer_name )
    plt.grid  ( False )
    plt.imshow( display_grid, aspect='auto', cmap='viridis' )
```

```
1/1 [==============================] - 0s 307ms/step
<ipython-input-11-5fdf8221a19d>:41: RuntimeWarning: invalid value encountered in divide
  x /= x.std ()
```



```
#Let's test with random images
import numpy as np

from google.colab import files
from keras.preprocessing import image

uploaded = files.upload()

for fn in uploaded.keys():
  path='/content/' + fn       #Save the image to content folder
  img=image.load_img(path, target_size=(150, 150))    #load the image

  x=image.img_to_array(img)
  x=np.expand_dims(x, axis=0)
  images = np.vstack([x])

  classes = model.predict(images, batch_size=10)    #predict the label for the image

  print(classes[0])     #Print the label, remember it will be either one or zero

  if classes[0]>0:
    print(fn + " is a dog")     #print human readable label

  else:
    print(fn + " is a cat")     #print human readable label
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

VGG16

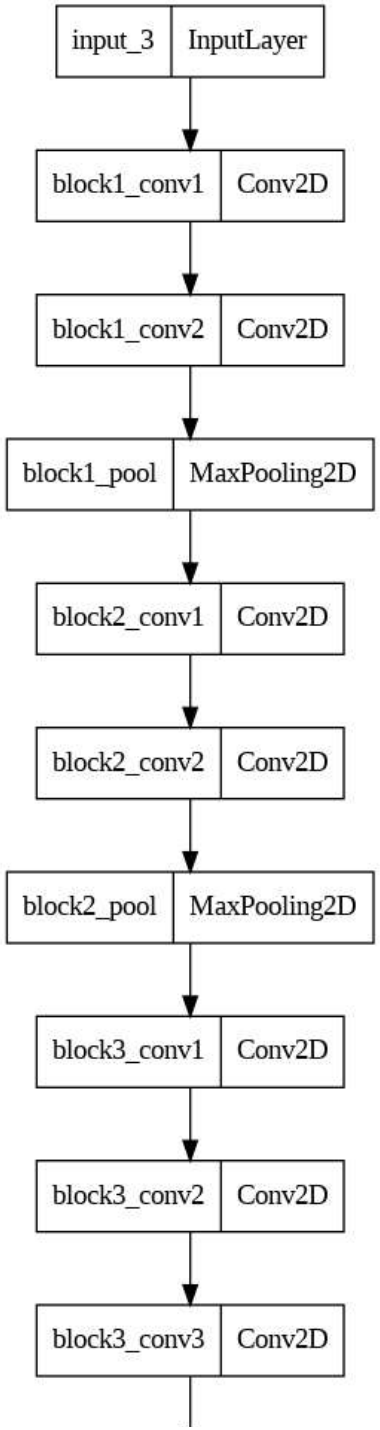
```
from keras.applications.vgg16 import VGG16
model=VGG16()
```

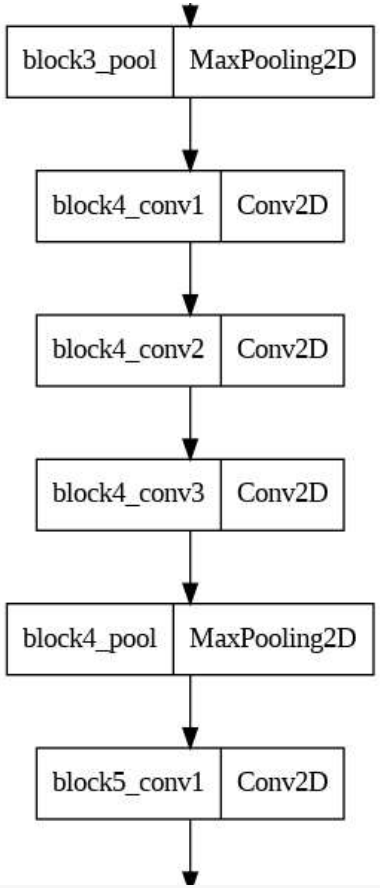
```
print(model.summary())
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0
```

```
 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
from tensorflow.keras.utils import plot_model
plot_model(model,to_file='vgg.png')
```


input_3 InputLayer
block1_conv1 Conv2D
block1_conv2 Conv2D
block1_pool MaxPooling2D
block2_conv1 Conv2D
block2_conv2 Conv2D
block2_pool MaxPooling2D
block3_conv1 Conv2D
block3_conv2 Conv2D
block3_conv3 Conv2D




```
# from keras.preprocessing.image import load_img
# #load an image from file
# image = load_img('test1.jpg', target_size =(224,224))
```

```
from keras.preprocessing.image import load_img
from keras.preprocessing import image
from google.colab import files
#load an image from file
uploaded = files.upload()
# image = load_img('uploaded', target_size =(224,224))
for fn in uploaded.keys():
  path='/content/' + fn       #Save the image to content folder
  image=image.load_img(path, target_size=(224, 224))    #load the image
```

Choose files pexels-pixabay-70083.jpg

- **pexels-pixabay-70083.jpg**(image/jpeg) - 419539 bytes, last modified: 25/12/2023 - 100% done

```
Saving pexels-pixabay-70083.jpg to pexels-pixabay-70083.jpg
```

```
from keras.preprocessing.image import img_to_array
#convert the image pixels to numpy array
image = img_to_array(image)
```

```
#reshape data from the model
image = image.reshape(1,image.shape[0],image.shape[1],image.shape[2])
```

```
from keras.applications.vgg16 import preprocess_input
#prepare the image from the VGG model
image = preprocess_input(image)
```

```
#predict the probility across all output class
yhat = model.predict(image)
```

```
1/1 [==============================] - 0s 481ms/step
```

```
from keras.applications.vgg16 import decode_predictions
#connect the probabilities to class labels
label = decode_predictions(yhat)
#retrieve the most likely result, e.g hihest probility
label = label[0][0]
#predict the classification
print('%s(%.2f%%)'%(label[1],label[2]*100))
```

```
tree_frog(97.23%)
```

Transfer Learning using vgg16

```
from keras.applications import vgg16

# MobileNet was designed to work on 224 x 224 pixel input images sizes
img_rows, img_cols = 224, 224

# Re-loads the MobileNet model without the top or FC layers
vgg = vgg16.VGG16(weights = 'imagenet',
                  include_top = False,
                  input_shape = (img_rows, img_cols, 3))

# Here we freeze the last 4 layers
# Layers are set to trainable as True by default
for layer in vgg.layers:
    layer.trainable = False

# Let's print our layers
for (i,layer) in enumerate(vgg.layers):
    print(str(i) + " "+ layer.__class__.__name__, layer.trainable)


def lw(bottom_model, num_classes):
    """creates the top or head of the model that will be
    placed ontop of the bottom layers"""

    top_model = bottom_model.output
    top_model = GlobalAveragePooling2D()(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(1024,activation='relu')(top_model)
    top_model = Dense(512,activation='relu')(top_model)
    top_model = Dense(num_classes,activation='softmax')(top_model)
    return top_model


from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
from keras.layers import BatchNormalization
from keras.models import Model

# Set our class number to 3 (Young, Middle, Old)
num_classes = 2

FC_Head = lw(vgg, num_classes)

model = Model(inputs = vgg.input, outputs = FC_Head)

print(model.summary())
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0
```

```
block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

global_average_pooling2d_2  (None, 512)               0
 (GlobalAveragePooling2D)

dense_8 (Dense)             (None, 1024)              525312

dense_9 (Dense)             (None, 1024)              1049600

dense_10 (Dense)            (None, 512)               524800

dense_11 (Dense)            (None, 2)                 1026

=================================================================
Total params: 16815426 (64.15 MB)
Trainable params: 2100738 (8.01 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
None
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
from keras.preprocessing.image import ImageDataGenerator

train_data_dir = '/content/drive/My Drive/Ai_dataset//train'
validation_data_dir = '/content/drive/My Drive/Ai_dataset/validate'

# Let's use some data augmentaiton
train_datagen = ImageDataGenerator(
      rescale=1./255,
      rotation_range=45,
      width_shift_range=0.3,
      height_shift_range=0.3,
      horizontal_flip=True,
      fill_mode='nearest')

validation_datagen = ImageDataGenerator(rescale=1./255)

# set our batch size (typically on most mid tier systems we'll use 16-32)
batch_size = 32

train_generator = train_datagen.flow_from_directory(
        train_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='categorical')
```

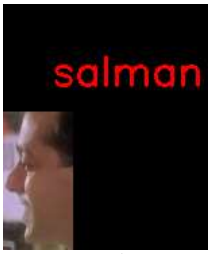
```
Found 539 images belonging to 2 classes.
Found 303 images belonging to 2 classes.
```

```
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, EarlyStopping


checkpoint = ModelCheckpoint("face recognisation.h5",
                             monitor="val_loss",
                             mode="min",
                             save_best_only = True,
                             verbose=1)

earlystop = EarlyStopping(monitor = 'val_loss',
                          min_delta = 0,
                          patience = 3,
                          verbose = 1,
                          restore_best_weights = True)

# we put our call backs into a callback list
callbacks = [earlystop, checkpoint]

# We use a very small learning rate
model.compile(loss = 'categorical_crossentropy',
              optimizer = RMSprop(lr = 0.001),
              metrics = ['accuracy'])

# Enter the number of training and validation samples here
nb_train_samples = 108
nb_validation_samples = 52

# We only train 5 EPOCHS
epochs = 5
batch_size = 16

history = model.fit_generator(
    train_generator,
    steps_per_epoch = nb_train_samples // batch_size,
    epochs = epochs,
    callbacks = callbacks,
    validation_data = validation_generator,
    validation_steps = nb_validation_samples // batch_size)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers
<ipython-input-10-2587e57752a8>:33: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
  history = model.fit_generator(
Epoch 1/5
6/6 [==============================] - ETA: 0s - loss: 1.4823 - accuracy: 0.4948
Epoch 1: val_loss improved from inf to 0.73278, saving model to face recognisation.h5
6/6 [==============================] - 47s 7s/step - loss: 1.4823 - accuracy: 0.4948 - val_loss: 0.7328 - val_accuracy: 0.4688
Epoch 2/5
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
6/6 [==============================] - ETA: 0s - loss: 0.7050 - accuracy: 0.4920
Epoch 2: val_loss did not improve from 0.73278
6/6 [==============================] - 37s 7s/step - loss: 0.7050 - accuracy: 0.4920 - val_loss: 0.7668 - val_accuracy: 0.4479
Epoch 3/5
6/6 [==============================] - ETA: 0s - loss: 0.6865 - accuracy: 0.5365
Epoch 3: val_loss improved from 0.73278 to 0.69035, saving model to face recognisation.h5
6/6 [==============================] - 16s 3s/step - loss: 0.6865 - accuracy: 0.5365 - val_loss: 0.6903 - val_accuracy: 0.4688
Epoch 4/5
6/6 [==============================] - ETA: 0s - loss: 0.7033 - accuracy: 0.4740
Epoch 4: val_loss did not improve from 0.69035
6/6 [==============================] - 10s 2s/step - loss: 0.7033 - accuracy: 0.4740 - val_loss: 0.7093 - val_accuracy: 0.5104
Epoch 5/5
6/6 [==============================] - ETA: 0s - loss: 0.6936 - accuracy: 0.5260
Epoch 5: val_loss did not improve from 0.69035
6/6 [==============================] - 9s 2s/step - loss: 0.6936 - accuracy: 0.5260 - val_loss: 0.6961 - val_accuracy: 0.5104
```

```
from keras.models import load_model

classifier = load_model('/content/face recognisation.h5')

import os
import cv2
from google.colab.patches import cv2_imshow
import numpy as np
from os import listdir
from os.path import isfile, join

monkey_breeds_dict = {"[0]": " salman ",
                      "[1]": "aamir "}

monkey_breeds_dict_n = {"n0": " salman",
                      "n1": "aamir "}

def draw_test(name, pred, im):
    monkey = monkey_breeds_dict[str(pred)]
    BLACK = [0,0,0]
    expanded_image = cv2.copyMakeBorder(im, 80, 0, 0, 100 ,cv2.BORDER_CONSTANT,value=BLACK)
    cv2.putText(expanded_image, monkey, (20, 60) , cv2.FONT_HERSHEY_SIMPLEX,1, (0,0,255), 2)
    cv2_imshow(expanded_image)

def getRandomImage(path):
    """function loads a random images from a random folder in our test path """
    folders = list(filter(lambda x: os.path.isdir(os.path.join(path, x)), os.listdir(path)))
    random_directory = np.random.randint(0,len(folders))
    path_class = folders[random_directory]
    print("Class - " + monkey_breeds_dict_n[str(path_class)])
    file_path = path + path_class
    file_names = [f for f in listdir(file_path) if isfile(join(file_path, f))]
    random_file_index = np.random.randint(0,len(file_names))
    image_name = file_names[random_file_index]
    return cv2.imread(file_path+"/"+image_name)

for i in range(0,10):
    input_im = getRandomImage("/content/drive/My Drive/Ai_dataset/validate/")
    input_original = input_im.copy()
    input_original = cv2.resize(input_original, None, fx=0.5, fy=0.5, interpolation = cv2.INTER_LINEAR)

    input_im = cv2.resize(input_im, (224, 224), interpolation = cv2.INTER_LINEAR)
    input_im = input_im / 255.
    input_im = input_im.reshape(1,224,224,3)

    # Get Prediction
    res = np.argmax(classifier.predict(input_im, 1, verbose = 0), axis=1)

    # Show image with predicted class
    draw_test("Prediction", res, input_original)
    cv2.waitKey(0)

cv2.destroyAllWindows()
```

Class - salman

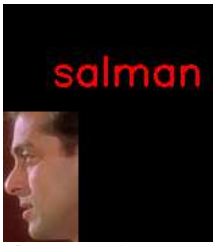


Class - aamir

Class - salman

Class - aamir

Class - salman



Class - aamir

Class - aamir

NLP_intro

```
import tensorflow as tf
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
sentences = [
              'i do not like dogs',
              'do you like dogs ?',
              'His cat is black',
              'This white cat is very cute',
              'gandhinagar गांधीनगर'
]
```

```
tokenizer = Tokenizer(oov_token='<oov>')
tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)
```

```
{'<oov>': 1, 'do': 2, 'like': 3, 'dogs': 4, 'cat': 5, 'is': 6, 'i': 7, 'not': 8, 'you': 9, 'his': 10, 'black': 11, 'this': 12, 'whit
```

```
sequences = tokenizer.texts_to_sequences(sentences)
print(sequences[0])
```

```
[7, 2, 8, 3, 4]
```

```
padded = pad_sequences(sequences, padding='post')
print(padded[0])
print(padded[1])
```

```
[7 2 8 3 4 0]
[2 9 3 4 0 0]
```

IMDB_review_EX1

```
#import
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import Sequential
from keras.layers import Embedding,LSTM,Dense,Dropout
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
```

```
#load imdb data and split it intp training & testing datasets
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = vocabulary_size)
# print('loaded dataset with {}training samples, {} test sample'.format((len(X_train),(len(X_test)))
print('loaded dataset with {}training samples, {} test sample'.format(str(len(X_train)),str(len(X_test))))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 [==============================] - 2s 0us/step
loaded dataset with 25000training samples, 25000 test sample
```

```
print("review:", X_train[0])
print("Label:", y_train[0])
```

```
review: [1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9,
Label: 1
```

```
#decode the sentences to see the reviews as test
word_index = imdb.get_word_index()
#print(word_index)
reverse_word_index = dict( [(value,key) for (key,value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i-3,'?')for i in X_train[0]])
print(X_train[0])
print(decoded_review)
decoded_review = ' '.join([reverse_word_index.get(i-3,'?')for i in X_train[2]])
print(X_train[2])
print(decoded_review)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 [==============================] - 1s 1us/step
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 48
? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could
[1, 14, 47, 8, 30, 31, 7, 4, 249, 108, 7, 4, 2, 54, 61, 369, 13, 71, 149, 14, 22, 112, 4, 2401, 311, 12, 16, 3711, 33, 75, 43, 1829,
? this has to be one of the worst films of the ? when my friends i were watching this film being the target audience it was aimed at
```

```
#pad the sequences
max_words =500
X_train = sequence.pad_sequences(X_train, maxlen = max_words)
X_test = sequence.pad_sequences(X_test, maxlen = max_words)
print(X_train[0])
```

```
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    1   14   22   16   43  530  973 1622 1385   65  458 4468
   66 3941    4  173   36  256    5   25  100   43  838  112   50  670
    2    9   35  480  284    5  150    4  172  112  167    2  336  385
   39    4  172 4536 1111   17  546   38   13  447    4  192   50   16
    6  147 2025   19   14   22    4 1920 4613  469    4   22   71   87
   12   16   43  530   38   76   15   13 1247    4   22   17  515   17
   12   16  626   18    2    5   62  386   12    8  316    8  106    5
```

```
   4 2223    2   16  480   66 3785   33    4  130   12   16   38  619
   5   25  124   51   36  135   48   25 1415   33    6   22   12  215
  28   77   52    5   14  407   16   82    2    8    4  107  117    2
  15  256    4    2    7 3766    5  723   36   71   43  530  476   26
 400  317   46    7    4    2 1029   13  104   88    4  381   15  297
  98   32 2071   56   26  141    6  194    2   18    4  226   22   21
 134  476   26  480    5  144   30    2   18   51   36   28  224   92
  25  104    4  226   65   16   38 1334   88   12   16  283    5   16
4472  113  103   32   15   16    2   19  178   32]
```

```
_#inrialize the model
embedding_size = 32
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(vocabulary_size,embedding_size, input_length = max_words))
model.add(tf.keras.layers.LSTM(100))
model.add(tf.keras.layers.Dense(1,activation = 'sigmoid'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           160000

 lstm (LSTM)                 (None, 100)               53200

 dense (Dense)               (None, 1)                 101

=================================================================
Total params: 213301 (833.21 KB)
Trainable params: 213301 (833.21 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
#compile model
model.compile( loss = 'binary_crossentropy',
              optimizer = 'adam',
               metrics = ['accuracy'])
```

```
#divide X_train in train and validation datasets
batch_size= 64
X_vaild , y_valid = X_train[:batch_size], y_train[:batch_size]
X_train_partial, y_train_partial = X_train[:batch_size:], y_train[:batch_size:]
```

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self , epoch , logs ={}):
    DESIRED_ACC = 0.9
    if(logs.get('val_accuracy')>= DESIRED_ACC):
      print("\n stopping training as validation accuracy is reached to %.2f!" % DESIRED_ACC)
    self.model.stop_training= True
callbacks = myCallback()
```

```
#fit the model
num_epochs = 5
history = model.fit(X_train_partial, y_train_partial, validation_data=(X_vaild, y_valid), batch_size= batch_size, epochs = num_epochs, c
```

```
Epoch 1/5
1/1 [==============================] - 4s 4s/step - loss: 0.6935 - accuracy: 0.4531 - val_loss: 0.6909 - val_accuracy: 0.6250
```

```
#test the model and print test accuracy score
scores =  model.evaluate (X_test, y_test)
print = ('Test accuracy:', scores[1])
```

```
782/782 [==============================] - 8s 10ms/step - loss: 0.6932 - accuracy: 0.4993
```

IMDBReviewEx2

```
#import
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras import Sequential
from keras.layers import Embedding, LSTM, Dense, Dropout
import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences

# Load IMDB data and split it into training & testing datasets
vocabulary_size = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=vocabulary_size)
print('Loaded dataset with {} training samples, {} test samples'.format(len(X_train), len(X_test)))

# Decode the sentences to see the reviews as test
word_index = imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?') for i in X_train[0]])
print(X_train[0])
print(decoded_review)

# Pad the sequences
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
print(X_train[0])

# Initialize the model
embedding_size = 32
model = tf.keras.Sequential()
model.add(tf.keras.layers.Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(tf.keras.layers.LSTM(100, return_sequences=True))  # Return sequences for deeper LSTM layers
model.add(tf.keras.layers.Dropout(0.2))  # Adding dropout for regularization
model.add(tf.keras.layers.LSTM(100))
model.add(tf.keras.layers.Dropout(0.2))  # Adding dropout for regularization
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Split X_train into train and validation datasets
batch_size = 64
X_valid, y_valid = X_train[:batch_size], y_train[:batch_size]
X_train_partial, y_train_partial = X_train[batch_size:], y_train[batch_size:]

# Define a callback for early stopping
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        DESIRED_ACC = 0.94
        if logs.get('val_accuracy') >= DESIRED_ACC:
            print("\nStopping training as validation accuracy reached %.2f!" % DESIRED_ACC)
            self.model.stop_training = True

callbacks = myCallback()

# Fit the model
num_epochs = 20
history = model.fit(X_train_partial, y_train_partial, validation_data=(X_valid, y_valid),
                    batch_size=batch_size, epochs=num_epochs, callbacks=[callbacks])

# Test the model and print the test accuracy score
scores = model.evaluate(X_test, y_test)
print('Test accuracy:', scores[1])
```

```
Epoch 1/20
390/390 [==============================] - 62s 145ms/step - loss: 0.4656 - accuracy: 0.7760 - val_loss: 0.2540 - val_accuracy: 0.
Epoch 2/20
390/390 [==============================] - 31s 81ms/step - loss: 0.3009 - accuracy: 0.8802 - val_loss: 0.3115 - val_accuracy: 0.8
Epoch 3/20
390/390 [==============================] - 22s 58ms/step - loss: 0.3290 - accuracy: 0.8621 - val_loss: 0.6736 - val_accuracy: 0.6
Epoch 4/20
390/390 [==============================] - 18s 47ms/step - loss: 0.6532 - accuracy: 0.5833 - val_loss: 0.6986 - val_accuracy: 0.4
Epoch 5/20
390/390 [==============================] - 19s 50ms/step - loss: 0.4580 - accuracy: 0.7771 - val_loss: 0.2095 - val_accuracy: 0.9
Epoch 6/20
390/390 [==============================] - 19s 48ms/step - loss: 0.2701 - accuracy: 0.8960 - val_loss: 0.2122 - val_accuracy: 0.8
Epoch 7/20
390/390 [==============================] - 18s 45ms/step - loss: 0.2280 - accuracy: 0.9143 - val_loss: 0.1988 - val_accuracy: 0.9
Epoch 8/20
390/390 [==============================] - 18s 47ms/step - loss: 0.2092 - accuracy: 0.9202 - val_loss: 0.1672 - val_accuracy: 0.
Epoch 9/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1871 - accuracy: 0.9312 - val_loss: 0.1952 - val_accuracy: 0.9
Epoch 10/20
390/390 [==============================] - 17s 44ms/step - loss: 0.1749 - accuracy: 0.9360 - val_loss: 0.1799 - val_accuracy: 0.9
Epoch 11/20
390/390 [==============================] - 17s 43ms/step - loss: 0.1603 - accuracy: 0.9427 - val_loss: 0.1950 - val_accuracy: 0.9
Epoch 12/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1421 - accuracy: 0.9502 - val_loss: 0.2386 - val_accuracy: 0.9
Epoch 13/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1244 - accuracy: 0.9593 - val_loss: 0.2205 - val_accuracy: 0.9
Epoch 14/20
390/390 [==============================] - 16s 41ms/step - loss: 0.1065 - accuracy: 0.9660 - val_loss: 0.2581 - val_accuracy: 0.9
Epoch 15/20
390/390 [==============================] - 15s 40ms/step - loss: 0.0921 - accuracy: 0.9722 - val_loss: 0.3507 - val_accuracy: 0.9
Epoch 16/20
390/390 [==============================] - 16s 40ms/step - loss: 0.0753 - accuracy: 0.9786 - val_loss: 0.3434 - val_accuracy: 0.9
Epoch 17/20
390/390 [==============================] - 16s 41ms/step - loss: 0.0676 - accuracy: 0.9808 - val_loss: 0.3732 - val_accuracy: 0.9
Epoch 18/20
390/390 [==============================] - 17s 44ms/step - loss: 0.0638 - accuracy: 0.9824 - val_loss: 0.2970 - val_accuracy: 0.9
Epoch 19/20
390/390 [==============================] - 15s 39ms/step - loss: 0.0505 - accuracy: 0.9880 - val_loss: 0.4462 - val_accuracy: 0.8
Epoch 20/20
390/390 [==============================] - 15s 39ms/step - loss: 0.0513 - accuracy: 0.9869 - val_loss: 0.3978 - val_accuracy: 0.8
782/782 [==============================] - 12s 15ms/step - loss: 0.5335 - accuracy: 0.8609
Test accuracy: 0.8608800172805786
```