The aim of this lab is to introduce DATA and FEATURES.

Let's get started.

Make a copy before running the cells

Extracting features from data

FMML Module 1, Lab 2

Module Coordinator : amit.pandey@research.iiit.ac.in

```
! pip install wikipedia

import wikipedia
import nltk
from nltk.util import ngrams
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import re
import unicodedata
import plotly.express as px
import pandas as pd
```

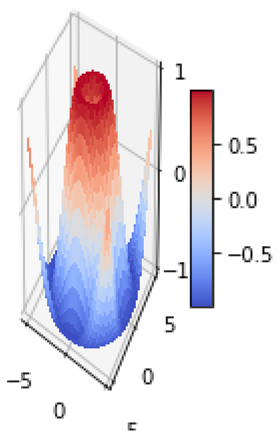
```
Collecting wikipedia
  Downloading wikipedia-1.4.0.tar.gz (27 kB)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: requests<3.0.0,>=2.0.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-pac
Building wheels for collected packages: wikipedia
  Building wheel for wikipedia (setup.py) ... done
  Created wheel for wikipedia: filename=wikipedia-1.4.0-py3-none-any.whl size=11695 s
  Stored in directory: /root/.cache/pip/wheels/15/93/6d/5b2c68b8a64c7a7a04947b4ed6d89
Successfully built wikipedia
Installing collected packages: wikipedia
Successfully installed wikipedia-1.4.0
```

What are features?

features are individual independent variables that act like a input to your system

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

from mpl_toolkits.mplot3d.axes3d import get_test_data


# set up a figure twice as wide as it is tall
fig = plt.figure(figsize=plt.figaspect(0.9))

# =============
# First subplot
# =============
# set up the axes for the first plot
ax = fig.add_subplot(1, 2, 1, projection='3d')

# plot a 3D surface like in the example mplot3d/surface3d_demo
X = np.arange(-5, 5, 0.25) # feature 1
Y = np.arange(-5, 5, 0.25) # feature 2
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R) #output
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap=cm.coolwarm,
                       linewidth=0.4, antialiased=False)
ax.set_zlim(-1.01, 1.01)
fig.colorbar(surf, shrink=0.5, aspect=10)
```

```
     <matplotlib.colorbar.Colorbar at 0x7fb6a46d0c10>
```



Part 2: Features of text

How do we apply machine learning on text? We can't directly use the text as input to our algorithms. We need to convert them to features. In this notebook, we will explore a simple way of converting text to features.

Let us download a few documents off Wikipedia.

```
topic1 = 'Giraffe'
topic2 = 'Elephant'
wikipedia.set_lang('en')
eng1 = wikipedia.page(topic1).content
eng2 = wikipedia.page(topic2).content
wikipedia.set_lang('fr')
fr1 = wikipedia.page(topic1).content
fr2 = wikipedia.page(topic2).content
```

This is what the text looks like:

```
fr2
```

```
'Les éléphants sont des mammifères proboscidiens de la famille des Éléphantidés. Ils
à trois espèces réparties en deux genres distincts. L\'Éléphant de savane d\'Afrique
\'Afrique, autrefois regroupés sous la même espèce d\'« Éléphant d\'Afrique », appar
tandis que l\'Éléphant d\'Asie, anciennement appelé « éléphant indien », appartient
férencient par certaines caractéristiques anatomiques, les éléphants d\'Asie étant e
es oreilles plus petites, ou encore une différence du bout de la trompe. Ces espèces
l\'objet de programmes ou de projets de réintroduction et de protection.\nLe mot fra
mot latin elephantus qui tire son origine du grec ἐλέφας signifiant « ivoire » ou «
```

We need to clean this up a bit. Let us remove all the special characters and keep only 26 letters and space. Note that this will remove accented characters in French also. We are also removing all the numbers and spaces. So this is not an ideal solution.

```
def cleanup(text):
  text = text.lower()  # make it lowercase
  text = re.sub('[^a-z]+', '', text) # only keep characters
  return text
```

```
eng1 = cleanup(eng1)
eng2 = cleanup(eng2)
fr1 = cleanup(fr1)
fr2 = cleanup(fr2)
```

```
print(eng1)
```

```
thegiraffeisatallafricanmammalbelongingtothegenusgiraffaspecificallyitisaneventoedun
```

Instead of directly using characters as the features, to understand a text better, we may consider group of tokens i.e. ngrams as features.

for this example let us consider that each character is one word, and let us see how n-grams work.

nltk library provides many tools for text processing, please explore them.

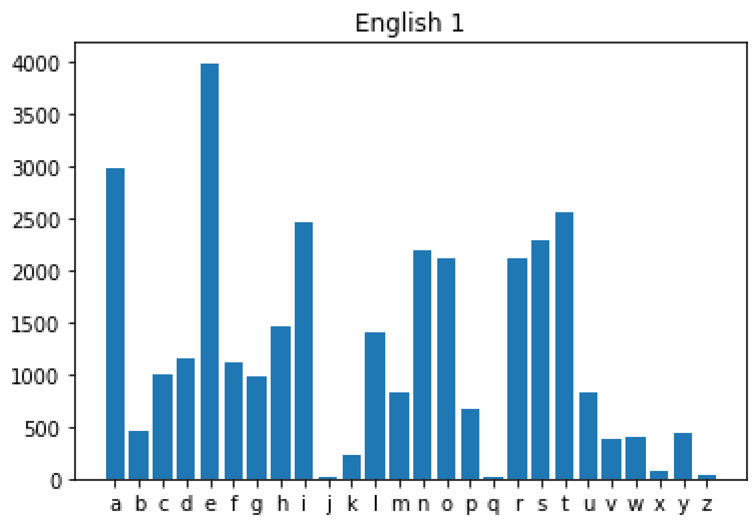
Now let us calculate the frequency of the character n-grams. N-grams are groups of characters of size n. A unigram is a single character and a bigram is a group of two characters and so on.

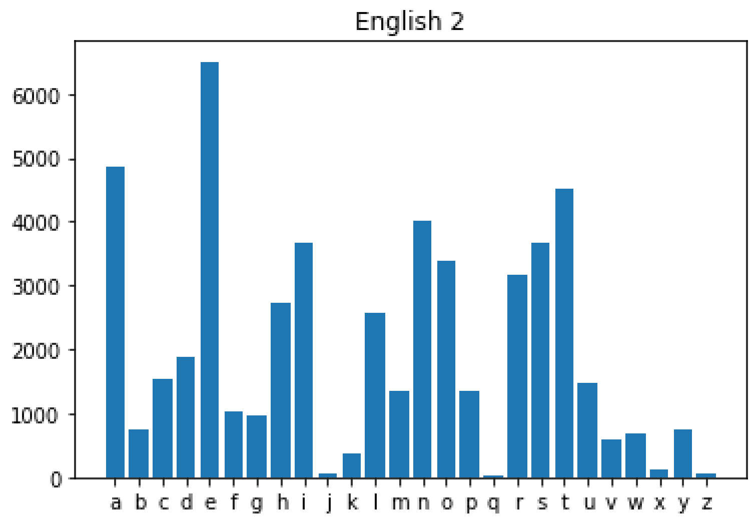
Let us count the frequency of each character in a text and plot it in a histogram.

```
# convert a tuple of characters to a string
def tuple2string(tup):
  st = ''
  for ii in tup:
    st = st + ii
  return st

# convert a tuple of tuples to a list of strings
def key2string(keys):
  return [tuple2string(i) for i in keys]

# plot the histogram
def plothistogram(ngram):
  keys = key2string(ngram.keys())
  values = list(ngram.values())

  # sort the keys in alphabetic order
  combined = zip(keys, values)
  zipped_sorted = sorted(combined, key=lambda x: x[0])
  keys, values = map(list, zip(*zipped_sorted))
  plt.bar(keys, values)
```

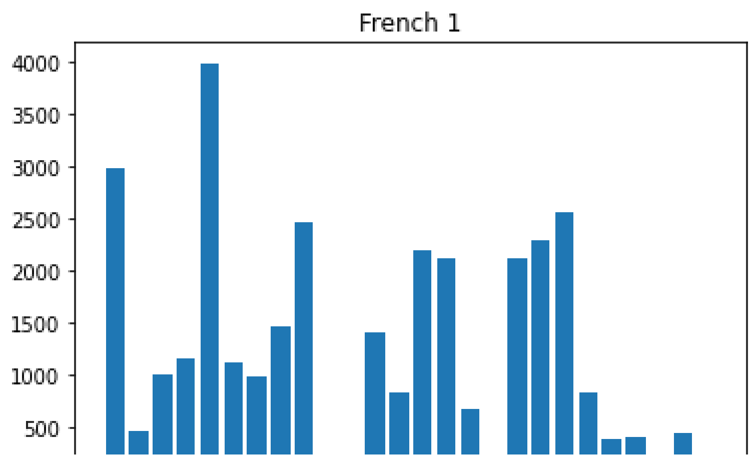
Let us compare the histograms of English pages and French pages. Can you spot a difference?

```
## we passed ngrams 'n' as 1 to get unigrams. Unigram is nothing but single token (in this

unigram_eng1 = Counter(ngrams(eng1,1))
plothistogram(unigram_eng1)
plt.title('English 1')
plt.show()
unigram_eng2 = Counter(ngrams(eng2,1))
plothistogram(unigram_eng2)
plt.title('English 2')
plt.show()
```





```
unigram_fr1 = Counter(ngrams(fr1,1))
plothistogram(unigram_eng1)
plt.title('French 1')
plt.show()
unigram_fr2 = Counter(ngrams(fr2,1))
plothistogram(unigram_fr2)
plt.title('French 2')
plt.show()
```

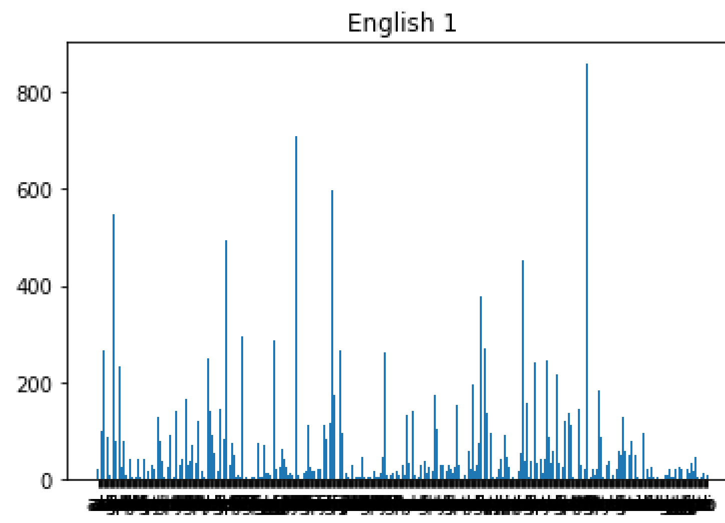


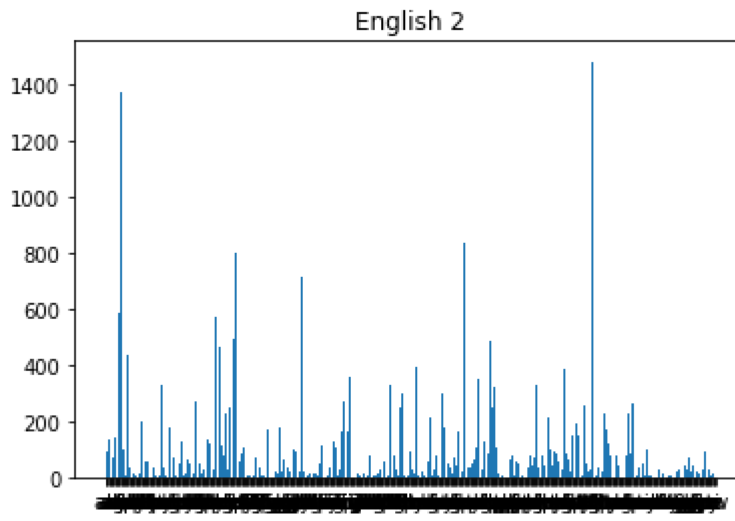
A good feature is one that helps in easy prediction and classification.

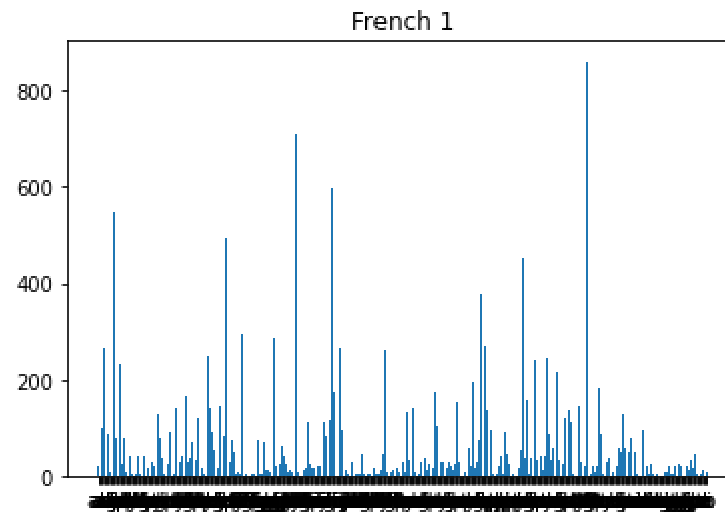
for ex : if you wish to differentiate between grapes and apples, size can be one of the useful features.

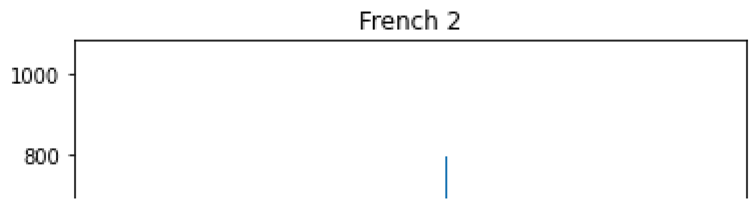
We can see that the unigrams for French and English are very similar. So this is not a good feature if we want to distinguish between English and French. Let us look at bigrams.

```
## Now instead of unigram, we will use bigrams as features, and see how useful bigrams are

bigram_eng1 = Counter(ngrams(eng1,2)) # bigrams
plothistogram(bigram_eng1)
plt.title('English 1')
plt.show()

bigram_eng2 = Counter(ngrams(eng2,2))
plothistogram(bigram_eng2)
plt.title('English 2')
plt.show()

bigram_fr1 = Counter(ngrams(fr1,2))
plothistogram(bigram_eng1)
plt.title('French 1')
plt.show()

bigram_fr2 = Counter(ngrams(fr2,2))
plothistogram(bigram_fr2)
plt.title('French 2')
plt.show()
```







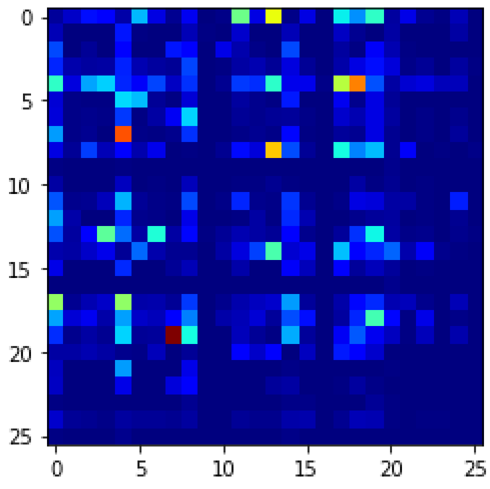


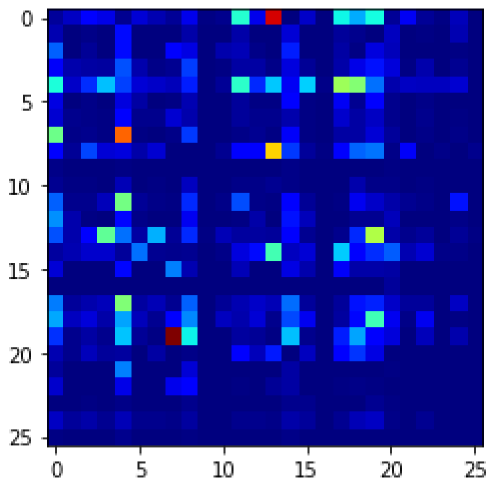
Another way to visualize bigrams is to use a 2-dimensional graph.

```
## lets have a lot at bigrams.
```

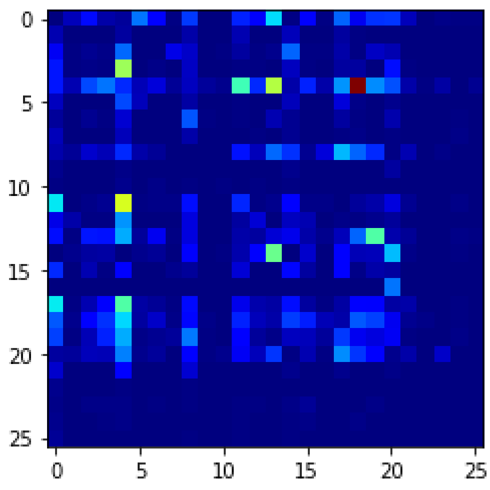
bigram_eng1

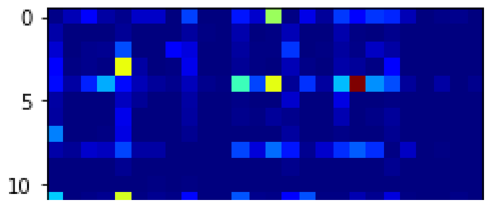
```
Counter({('a', 'a'): 21,
         ('a', 'b'): 56,
         ('a', 'c'): 119,
         ('a', 'd'): 99,
         ('a', 'e'): 32,
         ('a', 'f'): 265,
         ('a', 'g'): 77,
         ('a', 'h'): 16,
         ('a', 'i'): 85,
         ('a', 'j'): 2,
         ('a', 'k'): 9,
         ('a', 'l'): 416,
         ('a', 'm'): 76,
         ('a', 'n'): 548,
         ('a', 'o'): 7,
         ('a', 'p'): 77,
         ('a', 'q'): 1,
         ('a', 'r'): 312,
         ('a', 's'): 232,
         ('a', 't'): 350,
         ('a', 'u'): 26,
         ('a', 'v'): 79,
         ('a', 'w'): 14,
         ('a', 'x'): 6,
         ('a', 'y'): 43,
         ('a', 'z'): 1,
         ('b', 'a'): 41,
         ('b', 'b'): 1,
         ('b', 'c'): 3,
         ('b', 'd'): 1,
         ('b', 'e'): 125,
         ('b', 'f'): 4,
         ('b', 'g'): 1,
         ('b', 'i'): 41,
         ('b', 'j'): 4,
         ('b', 'l'): 59,
         ('b', 'm'): 1,
         ('b', 'n'): 1,
         ('b', 'o'): 42,
         ('b', 'p'): 1,
         ('b', 'r'): 52,
         ('b', 's'): 16,
         ('b', 't'): 1,
         ('b', 'u'): 28,
         ('b', 'y'): 28,
         ('c', 'a'): 171,
         ('c', 'c'): 20,
         ('c', 'e'): 105,
         ('c', 'g'): 1,
         ('c', 'h'): 127,
         ('c', 'i'): 109,
         ('c', 'k'): 80,
         ('c', 'l'): 35,
         ('c', 'm'): 8,
         ('c', 'n'): 3,
         ('c', 'o'): 171,
         ('c', 'p'): 1,
            ('c', 'r'): 26,

def plotbihistogram(ngram):
  freq = np.zeros((26,26))
  for ii in range(26):
    for jj in range(26):
      freq[ii,jj] = ngram[(chr(ord('a')+ii), chr(ord('a')+jj))]
  plt.imshow(freq, cmap = 'jet')
  return freq


bieng1 = plotbihistogram(bigram_eng1)
plt.show()
bieng2 = plotbihistogram(bigram_eng2)
```





```
bifr1 = plotbihistogram(bigram_fr1)
plt.show()
bifr2 = plotbihistogram(bigram_fr2)
```





Let us look at the top 10 ngrams for each text.

```
from IPython.core.debugger import set_trace

def ind2tup(ind):
  ind = int(ind)
  i = int(ind/26)
  j = int(ind%26)
  return (chr(ord('a')+i), chr(ord('a')+j))

def ShowTopN(bifreq, n=10):
  f = bifreq.flatten()
  arg = np.argsort(-f)
  for ii in range(n):
    print(f'{ind2tup(arg[ii])} : {f[arg[ii]]}')



print('\nEnglish 1:')
ShowTopN(bieng1)
print('\nEnglish 2:')
ShowTopN(bieng2)
print('\nFrench 1:')
ShowTopN(bifr1)
print('\nFrench 2:')
ShowTopN(bifr2)
```

```
     English 1:
     ('t', 'h') : 861.0
     ('h', 'e') : 709.0
     ('e', 's') : 664.0
     ('i', 'n') : 599.0
     ('a', 'n') : 548.0
     ('e', 'r') : 493.0
```

```
('r', 'a') : 455.0
('r', 'e') : 454.0
('a', 'l') : 416.0
('n', 'd') : 399.0

English 2:
('t', 'h') : 1479.0
('a', 'n') : 1368.0
('h', 'e') : 1190.0
('i', 'n') : 1014.0
('n', 't') : 833.0
('e', 'r') : 800.0
('e', 's') : 752.0
('r', 'e') : 750.0
('l', 'e') : 724.0
('h', 'a') : 716.0

French 1:
('e', 's') : 645.0
('l', 'e') : 394.0
('e', 'n') : 367.0
('d', 'e') : 349.0
('o', 'n') : 309.0
('n', 't') : 289.0
('r', 'e') : 289.0
('e', 'l') : 277.0
('r', 'a') : 230.0
('l', 'a') : 228.0

French 2:
('e', 's') : 1032.0
('n', 't') : 794.0
('d', 'e') : 656.0
('e', 'n') : 649.0
('l', 'e') : 625.0
('a', 'n') : 556.0
('o', 'n') : 484.0
('r', 'e') : 481.0
('e', 'l') : 439.0
('s', 'e') : 392.0
```

At times, we need to reduce the number of features. We will discuss this more in the upcoming sessions, but a small example has been discussed here. Instead of using each unique token (a word) as a feature, we reduced the number of features by using 1-gram and 2-gram of characters as features.

We observe that the bigrams are similar across different topics but different across languages. Thus, the bigram frequency is a good feature for distinguishing languages, but not for distinguishing topics.

Thus, we were able to convert a many-dimensional input (the text) to 26 dimesions (unigrams) or 26*26 dimensions (bigrams).

A few ways to explore:

1. Try with different languages.
2. The topics we used are quite similar, wikipedia articles of 'elephant' and 'giraffe'. What happens if we use very different topics? What if we use text from another source than Wikipedia?
3. How can we use and visualize trigrams and higher n-grams?

Features of Images.

Images in digital format are stored as numeric values, and hence we can use these values as features. for ex : a black and white (binary) image is stored as an array of 0 and 255 or 0 and 1.

Part 2: Written numbers

We will use a subset of the MNIST dataset. Each input character is represented in a 28*28 array. Let us see if we can extract some simple features from these images which can help us distinguish between the digits.

Load the dataset:

```
from keras.datasets import mnist

#loading the dataset
(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

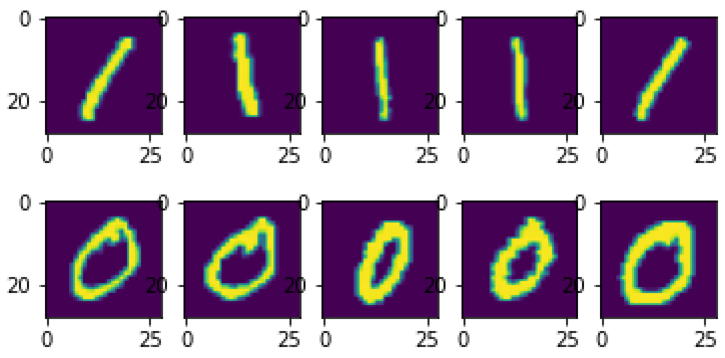
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mn
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

Extract a subset of the data for our experiment:

```
no1 = train_X[train_y==1,:,:] ## dataset corresponding to number = 1.
no0 = train_X[train_y==0,:,:] ## dataset corresponding to number = 0.
```

Let us visualize a few images here:

```
for ii in range(5):
  plt.subplot(1, 5, ii+1)
  plt.imshow(no1[ii,:,:])
plt.show()
for ii in range(5):
  plt.subplot(1, 5, ii+1)
  plt.imshow(no0[ii,:,:])
plt.show()
```



We can even use value of each pixel as a feature. But let us see how to derive other features.

Now, let us start with a simple feature: the sum of all pixels and see how good this feature is.

```
## sum of pixel values.

sum1 = np.sum(no1>0, (1,2)) # threshold before adding up
sum0 = np.sum(no0>0, (1,2))
```

Let us visualize how good this feature is: (X-axis is mean, y-axis is the digit)

```
plt.hist(sum1, alpha=0.7);
plt.hist(sum0, alpha=0.7);
```

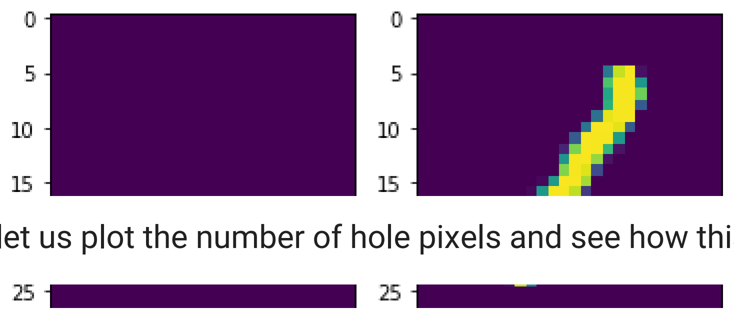
We can already see that this feature separates the two classes quite well.

Let us look at another, more complicated feature. We will count the number black pixels that are surrounded on four sides by non-black pixels, or "hole pixels".

```
def cumArray(img):
  img2 = img.copy()
  for ii in range(1, img2.shape[1]):
    img2[ii,:] = img2[ii,:] + img2[ii-1,:]  # for every row, add up all the rows above it.
  #print(img2)
  img2 = img2>0
  #print(img2)
  return img2

def getHolePixels(img):
  im1 = cumArray(img)
  im2 = np.rot90(cumArray(np.rot90(img)), 3) # rotate and cumulate it again for differnt d
  im3 = np.rot90(cumArray(np.rot90(img, 2)), 2)
  im4 = np.rot90(cumArray(np.rot90(img, 3)), 1)
  hull =  im1 & im2 & im3 & im4 # this will create a binary image with all the holes fille
  hole = hull & ~ (img>0) # remove the original digit to leave behind the holes
  return hole
```

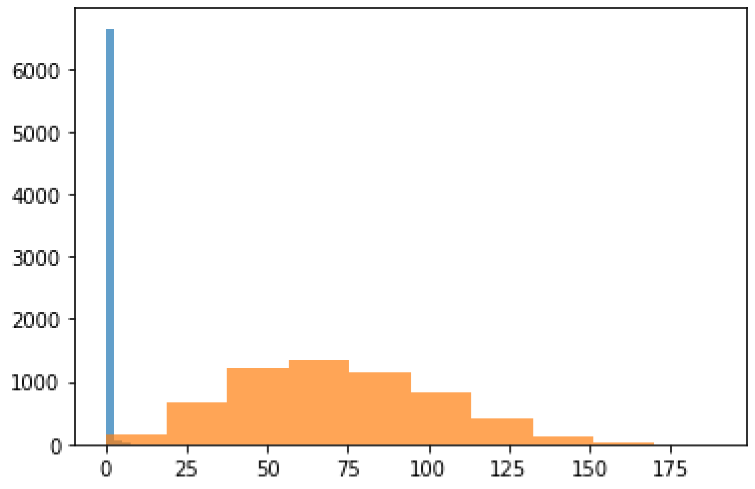
Visualize a few:

```
imgs = [no1[456,:,:],  no0[456,:,:]]
for img in imgs:
  plt.subplot(1,2,1)
  plt.imshow(getHolePixels(img))
  plt.subplot(1,2,2)
  plt.imshow(img)
  plt.show()
```



Now let us plot the number of hole pixels and see how this feature behaves

```
hole1 = np.array([getHolePixels(i).sum() for i in no1])
hole0 = np.array([getHolePixels(i).sum() for i in no0])

plt.hist(hole1, alpha=0.7);
plt.hist(hole0, alpha=0.7);
```

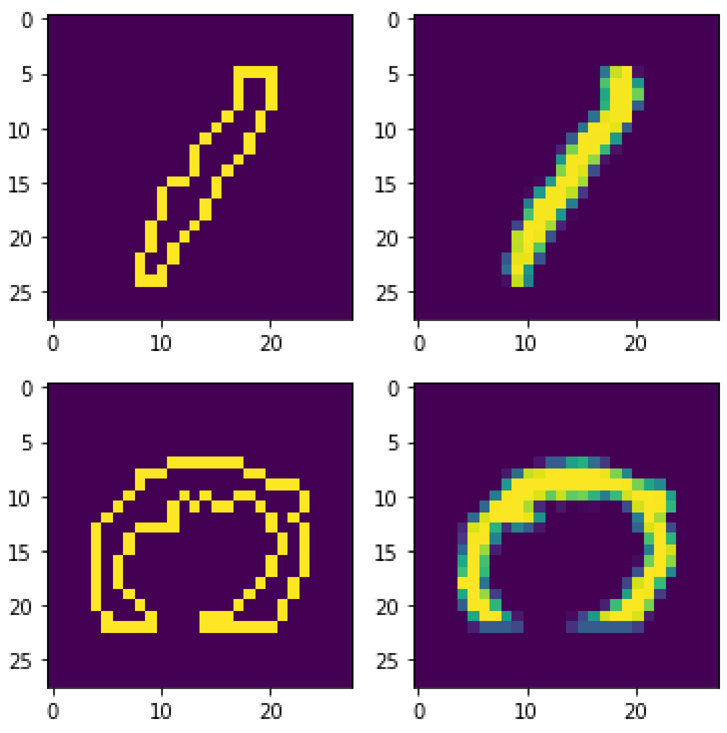


This feature works even better to distinguish between one and zero.

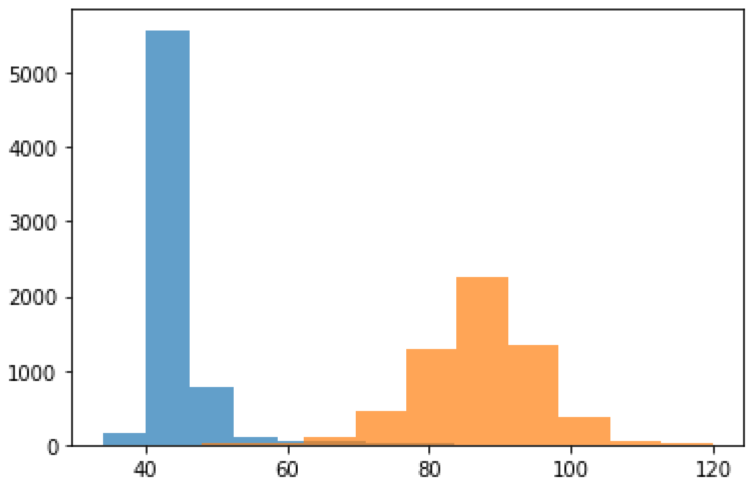
Now let us try the number of pixels in the 'hull' or the number with the holes filled in:

Let us try one more feature, where we look at the number of boundary pixels in each image.

```
def minus(a, b):
  return a & ~ b

def getBoundaryPixels(img):
  img = img.copy()>0  # binarize the image
  rshift = np.roll(img, 1, 1)
  lshift = np.roll(img, -1 ,1)
  ushift = np.roll(img, -1, 0)
  dshift = np.roll(img, 1, 0)
  boundary = minus(img, rshift) | minus(img, lshift) | minus(img, ushift) | minus(img, dsh
  return boundary


imgs = [no1[456,:,:],  no0[456,:,:]]
for img in imgs:
  plt.subplot(1,2,1)
  plt.imshow(getBoundaryPixels(img))
  plt.subplot(1,2,2)
  plt.imshow(img)
  plt.show()
```



```
bound1 = np.array([getBoundaryPixels(i).sum() for i in no1])
bound0= np.array([getBoundaryPixels(i).sum() for i in no0])

plt.hist(bound1, alpha=0.7);
plt.hist(bound0, alpha=0.7);
```



What will happen if we plot two features together?

Feel free to explore the above graph with your mouse.

We have seen that we extracted four features from a 28*28 dimensional image.

Some questions to explore:

1. Which is the best combination of features?

2. How would you test or visualize four or more features?
3. Can you come up with your own features?
4. Will these features work for different classes other than 0 and 1?
5. What will happen if we take more that two classes at a time?

Features from CSV file

```
import pandas as pd

df = pd.read_csv('/content/sample_data/california_housing_train.csv')
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0
2	-114.56	33.69	17.0	720.0	174.0	333.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0

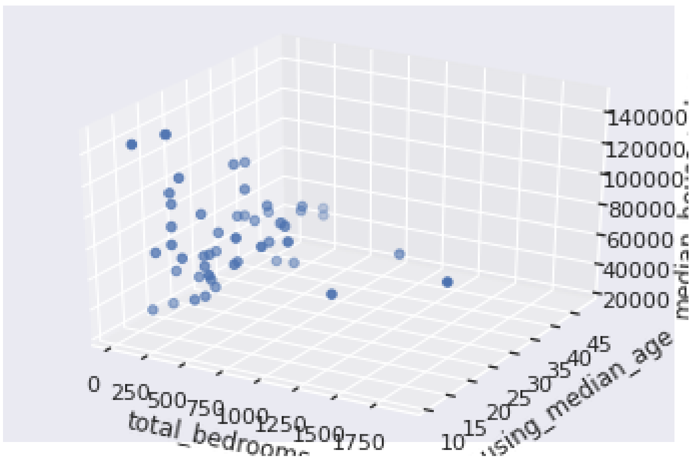
```
df.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

```
df = df.rename(columns={'oldName1': 'newName1', 'oldName2': 'newName2'})
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D


sns.set(style = "darkgrid")

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

x = df['total_bedrooms'][:50]
```

```
y = df['housing_median_age'][:50]
z = df['median_house_value'][:50]

ax.set_xlabel("total_bedrooms")
ax.set_ylabel("housing_median_age")
ax.set_zlabel("median_house_value")

ax.scatter(x, y, z)

plt.show()
```



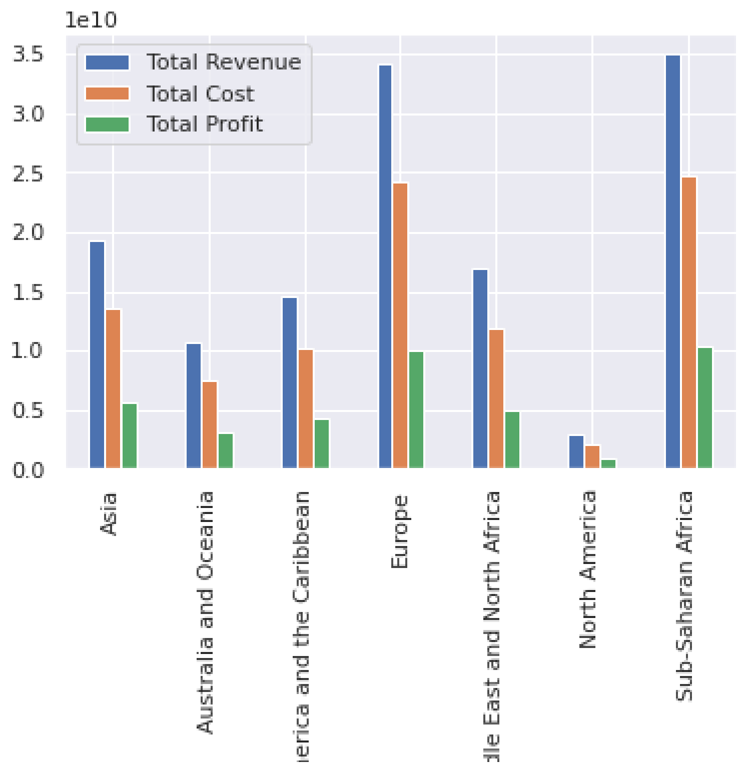
```
## Task :
## Download a CSV file from the internet, upload it to your google drive.
## Read the CSV file and plot graphs using different combination of features and write you
## Ex : IRIS flower datasaet


import pandas as pd
from google.colab import files


files.upload()
```

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.

```
Saving 100000 Sales Records.csv to 100000 Sales Records.csv
{'100000 Sales Records.csv': b"Region,Country,Item Type,Sales Channel,Order Priority,
```

```
sales_data = pd.read_csv("/content/100000 Sales Records.csv")
trimmed_sales_data = sales_data[['Region', 'Total Revenue', 'Total Cost', 'Total Profit']]
regionwise = trimmed_sales_data.groupby('Region').sum()


regionwise.plot.bar() #plotting data of the total revenue, total cost and total profit gen
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb634314c50>
```

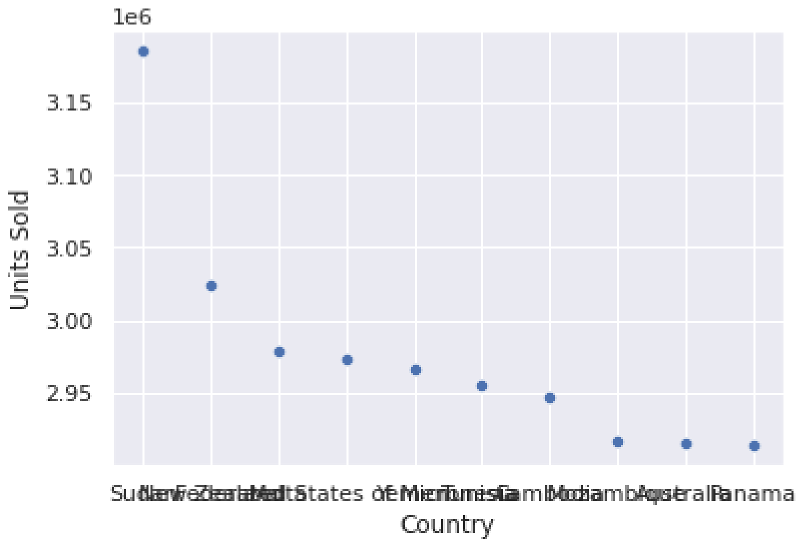


```
trim_country = sales_data[['Country', 'Units Sold']]
countrywise = trim_country.groupby('Country').sum()
```

Region

```
countrywise = countrywise.sort_values(by="Units Sold")
countrywise = countrywise[-1:-11:-1]
sns.scatterplot(y='Units Sold', x='Country', data=countrywise) #Country-wise plot of units
```

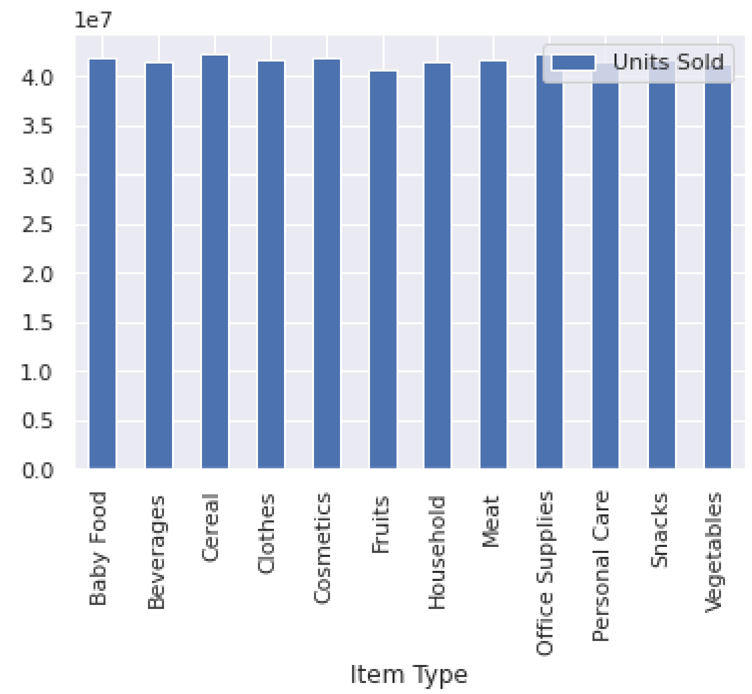
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb6347b85d0>
```



```
item_type = sales_data[['Item Type', 'Units Sold']]
item_type = item_type.groupby('Item Type').sum()
```

```
item_type.plot.bar() # Plot of units sold of each Item type
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb634588410>
```


1e7
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Units Sold
Baby Food
Beverages
Cereal
Clothes
Cosmetics
Fruits
Household
Meat
Office Supplies
Personal Care
Snacks
Vegetables
Item Type