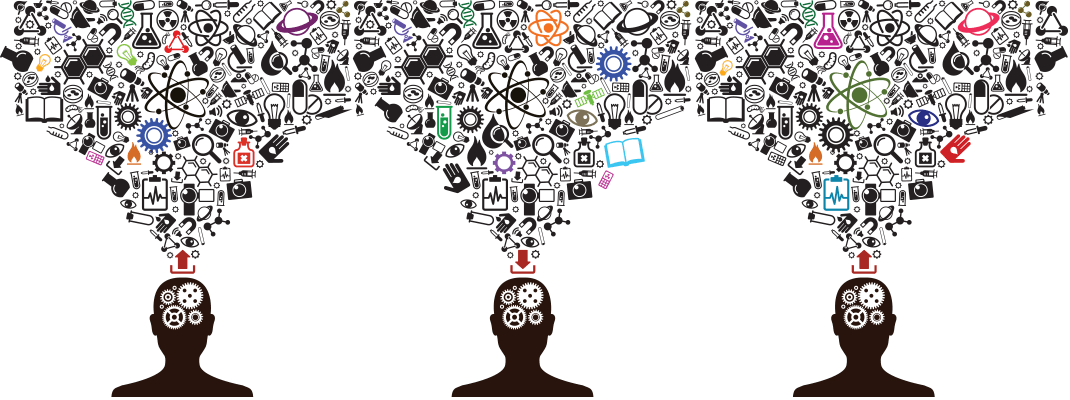
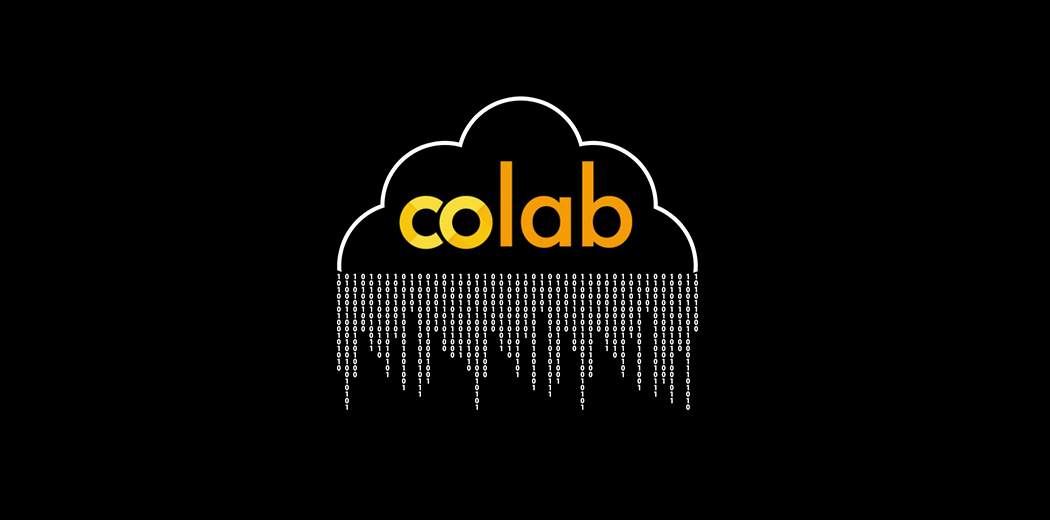
It’s an excellent idea. I believe that **every developer should have a blog on their projects. I know the fact how much we coders hate to write but, writing a blog will help the newbies understand the code better and even a non-coder could understand the code.**



In this blog, I am going to assume that you are an absolute beginner who has never built a python code before. This blog will teach you exactly how to build your program step-by-step, starting from choosing a platform to write your code as in my case I chose **Google Colab**, The project I have chosen is small and it’s about how to get the information about the cars for sale in our locality.



So, let’s begin by importing **pandas**. It’s a software library written for the **Python** programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. but rather than using the name **pandas**, it's instructed to use the name **pd** instead (# import pandas as pd). As we have a great start let’s move to the next step of gathering all the information on google colab into one colab notebook that we are using (# from google.colab import files). Next step we will upload the file directly in the Colab notebook by using (# uploaded = files.upload()) which gives you a traditional upload button that allows you to move files from your computer into to the Colab environment. Now that we have made a suitable environment for our program next step is to choose the file which we already have saved in our computers as in my case I chose a file of data base which was already present in **kaggel.** Which I had downloaded and kept ready as the file being in excel format, I had to convert it into the notepad kind of format using (txt.) and saving the file again and once it was done it was easy to upload the file easy to understand it.



Once the file was uploaded the next step was to make the notebook read the uploaded file and see if the right file is being upload and how the file would look in the colab environment for which we used the following (# df =pd.read table), this helps colab read the file and in this case the file was in txt. Format, the name of the file and txt. So as to display and read the file, we display the uploaded file by giving the command of df which prints the data in the file. Then we will write (# df.describe()), which will describe how many rows and column are present in the file and the details present in the file in it.

Next, we choose a unique identity present in the file as which would help us differentiate between this and different columns and rows as in this case we us Car\_Name and the code (# fname = df['Car\_Name'].unique()) which makes the car name as the unique feature of this code. And then we will display the unique label by typing fname in the dialog box so as to display the all the items present in the file. The next step is to group the data that is being provided we do this by using (# fsize = df.groupby(df['Car\_Name'],sort = False ).size()), this groups the data and sorts the data as its being given cause we are giving sort as false. Then we will plot the graph which will help us in understanding the data better for this first we have to import the graph library and plot library (# import matplotlib.pyplot as plt) this will import all the things we will need for the graph and the plotting of it. Then we give the command as to how we want the graph to look and other details.

(# plt.bar(fname,fsize,color = ['r','b','c','m'])

plt.title('cars vs count')

plt.xlabel('cars')

plt.ylabel('count')

plt.show())

this command will make a graph for you all you have to do is to select the labels for x-axis and y-axis and the name you want to give the graph and the color of the bars which will display the different things for your graph.

Next, we take the next step in which Pandas provide a unique method to retrieve rows from a Data frame.( **Dataframe.iloc[] ),** method is used when the index label of a data frame is something other than numeric series of 0, 1, 2, 3….n or in case the user doesn’t know the index label. Rows can be extracted using an imaginary index position which isn’t visible in the data frame using

(# x = df.iloc[:,[2,3,4]].values

y = df.iloc[:,7].values

print (x,y) )

which gives a long and detail description of the lists of items that were in the row.

As shown the next step is sklearn and X-train and Y-train and other things similar with the axis so, what it is actually?

Let’s make a background to what is it all. So, let’s begin by seeing first what is sklearn (or Scikit-learn). It is a [Python library](https://www.bitdegree.org/tutorials/python-libraries/) that offers various features for **data processing** that can be used for classification, clustering, and model selection. So now what is model selection, it is a method for setting a **blueprint** to analyze data and then using it to measure new data. Selecting a proper model allows you to generate **accurate results** when making a prediction, well hope that makes you understand a lot about sklearn and model selection, so now let’s get back to explain what is that test and train that we have used. It’s called train\_test\_split, it is a function in **Sklearn model selection** for splitting data arrays into **two subsets**: for training data and for testing data. With this function, you don't need to divide the dataset manually. By default, Sklearn **train\_test\_split** will make random partitions for the two subsets. However, you can also specify a random state for the operations as in this case I have chosen random state to be 6

(# from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,random\_state = 6) )

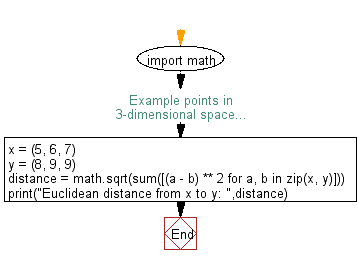
further giving the print command we, all know for what.

Moving on (# from sklearn.preprocessing import MinMaxScaler # Range 0 to 1)

class **sklearn**.**preprocessing**. **MinMaxScaler** (feature\_range = (**0**, **1**)). Transform features by scaling each feature to a given **range**. This estimator scales and translates each feature individually such that it is in the given **range** on the training set, e.g. between zero and one, as we have chosen. The following steps are just making the model fit to scaler scale.

The next major step is to using the classification so as to get the required answer for which we use (# from sklearn.neighbors import KNeighborsClassifier). the [**KNeighborsClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier) implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. The optimal choice of the value k is highly data-dependent in general makes the classification boundaries less distinct.

As for the term **Euclidean**. It is a mathematical term, the **Euclidean** distance or **Euclidean** metric is the straight-line distance between two points in **Euclidean** space



With this distance, **Euclidean** space becomes a metric space.

The next step is preparing the model for to fit and the following step is to making the model to predict by using (# y\_pred = model.predict(x\_test)), y pred is the conversion of the numpy array X\_test into a tensor. The tensor **y\_pred** is the data predicted (calculated, output) by your model. Usually, both y\_true and y**\_pred** have exactly the same shape. A few of the losses, such as the sparse ones, may be accepted.

Next we will test the Y\_test so to see if all the above steps have been done as we were plaining them once we are sura about the displayed we will finally move to the last step for which we had to make an environment to get the prediction of the dataset we had taken in the first place. So, the final step is

(# from sklearn.metrics import accuracy\_score

accuracy\_score(y\_test,y\_pred)\*100)

The [accuracy\_score](https://docs.w3cub.com/scikit_learn/modules/generated/sklearn.metrics.accuracy_score/#sklearn.metrics.accuracy_score) function computes the [accuracy](https://en.wikipedia.org/wiki/Accuracy_and_precision), either the fraction (default) or the count of correct predictions.

In multilabel classification, the function returns the subset accuracy. If the entire set of predicted right for a sample strictly, it matches with the true set of data, then the subset accuracy is 1.0, otherwise it is 0.0.

Here we are done with the program and if everything goes write the prediction will be right but the best prediction is considered to be the maximum times the outcome will be useful and right, but it should be less than 100 because if the prediction is 100 there is some mistake and the prediction should be on the positive side mostly crossing 50 % only then the program might have a right prediction and be helpful for the purpose it is being made.