```
# MAJOR PROJECT 1 - Choose any dataset of ur choice and apply suitable REGRESSOR/CLASSIFIE
#Dataset - '/content/CO2_emission.csv'


#1.Take a dataset and create dataframe
import pandas as pd
df = pd.read_csv("/content/CO2_emission.csv")
df
```

	Model_Year	Make	Model	Vehicle_Class	Engine_Size	Cylinders	Transmission	F
0	2021	Acura	ILX	Compact	2.4	4	AM8	
1	2021	Acura	NSX	Two-seater	3.5	6	AM9	
2	2021	Acura	RDX SH-AWD	SUV: Small	2.0	4	AS10	
3	2021	Acura	RDX SH-AWD A-SPEC	SUV: Small	2.0	4	AS10	
4	2021	Acura	TLX SH-AWD	Compact	2.0	4	AS10	
...	...	...	...	...	...	...	...	
930	2021	Volvo	XC40 T5 AWD	SUV: Small	2.0	4	AS8	
931	2021	Volvo	XC60 T5 AWD	SUV: Small	2.0	4	AS8	
932	2021	Volvo	XC60 T6 AWD	SUV: Small	2.0	4	AS8	
933	2021	Volvo	XC90 T5 AWD	SUV: Standard	2.0	4	AS8	
934	2021	Volvo	XC90 T6 AWD	SUV: Standard	2.0	4	AS8	

935 rows × 12 columns

```
#to display the information present in the table
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 935 entries, 0 to 934
Data columns (total 12 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   Model_Year                              935 non-null    int64  
 1   Make                                    935 non-null    object 
 2   Model                                   935 non-null    object 
 3   Vehicle_Class                           935 non-null    object 
 4   Engine_Size                             935 non-null    float64
 5   Cylinders                               935 non-null    int64  
 6   Transmission                            935 non-null    object 
 7   Fuel_Consumption_in_City(L/100 km)      935 non-null    float64
 8   Fuel_Consumption_in_City_Hwy(L/100 km)  935 non-null    float64
 9   Fuel_Consumption_comb(L/100km)          935 non-null    float64
 10  CO2_Emissions                           935 non-null    int64  
 11  Smog_Level                              935 non-null    int64  
dtypes: float64(4), int64(4), object(4)
memory usage: 87.8+ KB
```

```
df.shape # 935 rows and 13 columns
```

```
(935, 12)
```

```
df.size # total number of elements
```

```
11220
```

```
#to check the number to null values present
df.isnull()
```

	Model_Year	Make	Model	Vehicle_Class	Engine_Size	Cylinders	Transmission	F
0	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	
930	False	False	False	False	False	False	False	
931	False	False	False	False	False	False	False	
932	False	False	False	False	False	False	False	
933	False	False	False	False	False	False	False	
934	False	False	False	False	False	False	False	

935 rows × 12 columns

```
# To display 1st 5 row  indexes
df.head()
```

	Model_Year	Make	Model	Vehicle_Class	Engine_Size	Cylinders	Transmission	Fue
0	2021	Acura	ILX	Compact	2.4	4	AM8	
1	2021	Acura	NSX	Two-seater	3.5	6	AM9	
2	2021	Acura	RDX SH-AWD	SUV: Small	2.0	4	AS10	
3	2021	Acura	RDX SH-AWD A-SPEC	SUV: Small	2.0	4	AS10	
4	2021	Acura	TLX SH-AWD	Compact	2.0	4	AS10	

```
#To display last 5 row indexes
df.tail()
```

	Model_Year	Make	Model	Vehicle_Class	Engine_Size	Cylinders	Transmission	F
930	2021	Volvo	XC40 T5 AWD	SUV: Small	2.0	4	AS8	
931	2021	Volvo	XC60 T5 AWD	SUV: Small	2.0	4	AS8	
932	2021	Volvo	XC60 T6 AWD	SUV: Small	2.0	4	AS8	
933	2021	Volvo	XC90 T5 AWD	SUV: Standard	2.0	4	AS8	
934	2021	Volvo	XC90 T6 AWD	SUV: Standard	2.0	4	AS8	

```
#2.Preprocessing – Filtering of Data(to remove model_year column)
df_numeric = df_numeric.drop(['Model_Year'],axis = 1)#axis = 1 -column,axis = 0 - row
df_numeric
```

	Engine_Size	Cylinders	Fuel_Consumption_in_City(L/100 km)	Fuel_Consumption_in_Cit
0	2.4	4	9.9	
1	3.5	6	11.1	
2	2.0	4	11.0	
3	2.0	4	11.3	
4	2.0	4	11.2	
...	...	...	...	
930	2.0	4	10.7	
931	2.0	4	11.1	
932	2.0	4	11.7	
933	2.0	4	11.5	
934	2.0	4	12.1	

935 rows × 7 columns

```
#We want to consider only the numeric data
#So we will create a new dataframe with only numeric data
df_numeric = df.select_dtypes(include = ['float64','int64'])
df_numeric
```

	Model_Year	Engine_Size	Cylinders	Fuel_Consumption_in_City(L/100 km)	Fuel_Consum
0	2021	2.4	4	9.9	
1	2021	3.5	6	11.1	
2	2021	2.0	4	11.0	
3	2021	2.0	4	11.3	
4	2021	2.0	4	11.2	
...	...	...	...	...	
930	2021	2.0	4	10.7	
931	2021	2.0	4	11.1	
932	2021	2.0	4	11.7	
933	2021	2.0	4	11.5	
934	2021	2.0	4	12.1	

935 rows × 8 columns

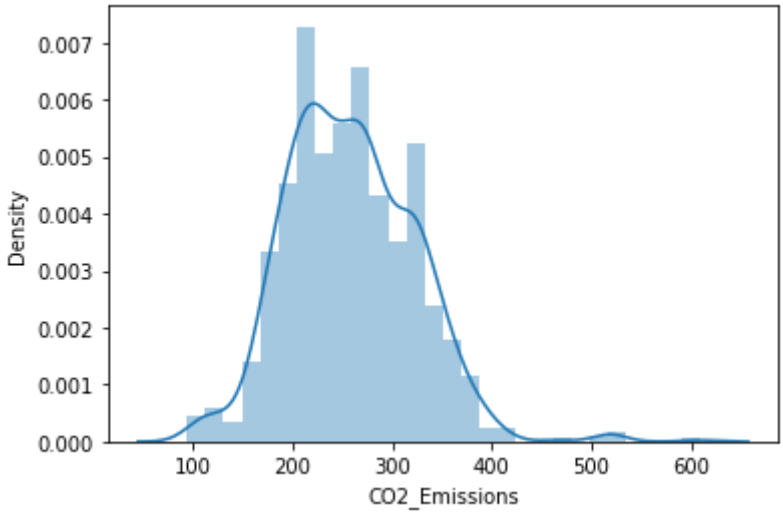
```
#to display the table information which contains only numeric data
df_numeric.info()
```

```
df_numeric.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 935 entries, 0 to 934
Data columns (total 7 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   Engine_Size                             935 non-null    float64
 1   Cylinders                               935 non-null    int64  
 2   Fuel_Consumption_in_City(L/100 km)      935 non-null    float64
 3   Fuel_Consumption_in_City_Hwy(L/100 km)  935 non-null    float64
 4   Fuel_Consumption_comb(L/100km)          935 non-null    float64
 5   CO2_Emissions                           935 non-null    int64  
 6   Smog_Level                              935 non-null    int64  
dtypes: float64(4), int64(3)
memory usage: 51.3 KB
```

```
#3.VISUALIZATION
import seaborn as sns
sns.distplot(df['CO2_Emissions']) # distribution plot
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f4b1a616410>
```




```
#4.divide the data into i/p and o/p
#output - Smog_Level
#input - All the columns except the Smog_Level column
```

```
x = df_numeric.iloc[:,0:6].values
x
```

```
array([[  2.4,   4. ,   9.9,   7. ,   8.6, 199. ],
       [  3.5,   6. ,  11.1,  10.8,  11. , 256. ],
       [  2. ,   4. ,  11. ,   8.6,   9.9, 232. ],
       ...,
       [  2. ,   4. ,  11.7,   8.6,  10.3, 240. ],
       [  2. ,   4. ,  11.5,   8.4,  10.1, 236. ],
       [  2. ,   4. ,  12.1,   8.5,  10.5, 245. ]])
```

```
y = df_numeric.iloc[:,6]
y
```

```
0      3
1      3
2      6
3      6
4      7
      ..
930    5
931    5
932    7
933    5
934    7
Name: Smog_Level, Length: 935, dtype: int64
```

```
#5.TRAIN and TEST VARIABLES
#sklearn.model_selection - package , train_test_split - library
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 0)

#Whatever data splitting /data allocation happens to the xtrain,x_test,ytrain,ytest variab
#By default the training variables get 75 % and testing variables get 25%


print(x.shape) # 935 rows and 16 cols
print(x_train.shape) # 935 rows and 16 cols (75%)
print(x_test.shape) # 935 rows and 6 cols (25%)
```

```
(935, 6)
(701, 6)
(234, 6)
```

```
print(y.shape) # 935 rows and 6 col s
print(y_train.shape) # 935 rows and 6 cols (75%)
print(y_test.shape) # 935 rows and 6 cols (25%)
```

```
(935,)
(701,)
(234,)
```

```
#6.SCALING or NORMALISATION -DONE ONLY FOR INPUTS
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)


#7.RUN a CLASSIFIER/REGRESSOR/CLUSTERER
from sklearn.linear_model import LinearRegression
model = LinearRegression()


#8.MODEL FITTING
model.fit(x_train,y_train)
```

```
LinearRegression()
```

```
#9.PREDICT THE OUTPUT
y_pred = model.predict(x_test)#By taking the input testing data , we predict the output
y_pred #PREDICTED VALUES
```

```
array([ 5.03280346,  5.60236537,  5.90871566,  4.27895557,  2.09829152,
        2.13895317,  4.75142172,  5.79303749,  3.02294949,  3.38021619,
        0.58545191,  6.6905652 ,  4.25165964,  4.40134534,  5.55785873,
        4.93826861,  3.52132741,  4.88727142,  4.57343103,  3.22249974,
        3.20967148,  5.22087635,  4.37314986,  3.65104634,  4.68989912,
        2.19673097,  3.76610878,  5.48844628,  4.29705295,  5.7494009 ,
        3.74350064,  4.89116343,  4.70804526,  7.26930534,  2.43418722,
        3.76960145,  3.68255503,  4.46710958,  5.72965351,  7.31130157,
        3.81556038,  4.27869683,  4.01949527,  2.37845392,  1.93938136,
        3.87672782,  4.69937159,  2.8697577 ,  4.9982898 ,  3.67171002,
        3.7131786 ,  5.27386344,  3.52132741,  4.90666228,  5.54912635,
        0.63126567,  5.79601488,  2.8697577 ,  4.61574324,  3.94991553,
        4.88851614,  4.31257509,  3.13413653,  4.30718411,  5.19608474,
        4.72910047,  5.05972428,  6.70961455,  4.37094101,  5.15038308,
        5.26952539,  5.09845412,  4.27855622,  5.47104076,  4.94551399,
        5.60977415,  3.68255503,  4.63205346,  4.30643513,  3.88273091,
        4.96272534,  3.71148783,  4.58333237, -2.47737066,  3.59385227,
        4.99384516,  4.41238297,  4.01021227,  3.63539349,  5.93453236,
        4.11810814,  5.41971966,  5.63761171,  4.17157046,  5.32928654,
        4.94037958,  3.85738667,  5.69973664,  3.27433989,  4.48627932,
        5.11109431,  5.66257326,  4.61266503,  5.37367123,  4.68640645,
        4.18115923,  3.46762623,  4.07946918,  3.35799981,  3.4678225 ,
        7.102476  ,  4.29768835,  4.77541601,  5.25137096,  6.77086104,
        2.43961928,  4.24016021,  5.15923576,  3.88161702,  4.07121293,
        4.2064738 ,  4.01195994,  4.40983426,  5.54496887,  5.46815742,
        5.26326226,  3.64500135,  4.74579984,  4.88767076,  5.40898441,
        3.25037656,  4.47340447,  4.75205711,  3.27397465,  3.57473957,
        3.59131612,  6.95573366,  2.92362812,  3.88533787,  2.28823243,
        5.76635212,  4.66645597,  5.13981625,  3.76960145,  4.59923056,
        4.68138603,  2.43418722,  4.34964207,  2.89134076,  2.89485672,
        4.68652003,  1.36856634,  3.91471086,  4.34014631,  3.32533778,
        2.53738004,  5.06996685,  1.38213382,  1.36856634,  4.71573911,
        4.35487289,  5.15027097,  2.03694481,  4.71573911,  4.24851879,
        2.69957221,  3.08064696,  0.85637764,  4.14641195,  5.37665176,
        3.7812347 ,  2.79492506,  5.40270975,  3.41340349,  4.73182087,
        4.6299998 ,  1.54087092,  3.25599076,  3.93804043,  2.99721636,
        2.45576052,  4.48627932,  4.55426218,  2.52120879,  4.65337813,
        4.9395109 ,  4.21739725,  5.08306738,  5.05972428,  5.1711887 ,
        4.89245485,  2.79492506,  4.76295787,  3.37768947,  3.71148783,
        4.25372179,  4.17157046,  4.75264002,  5.53919207,  5.67460143,
        3.98300881,  1.41499796,  3.59670126,  6.84170247,  3.63539349,
        5.7494009 ,  5.73343798,  4.72007716,  3.21651995,  4.26824211,
        5.59671122,  3.65104634,  5.58300697,  4.84286255,  5.49052803,
        1.83047394,  3.12339977,  3.71023964,  3.81799814,  2.33904958,
        5.4940207 ,  2.43418722,  3.29304729,  3.95298666,  5.92633533,
        4.65643839,  3.8168442 ,  4.25499121,  2.18938168,  4.1619108 ,
        4.61574324,  3.98283888,  4.31124468,  4.33302036])
```

```
y_test #ACTUAL VALUES
```

```
689    6
236    5
738    3
766    1
266    1
      ..
736    5
172    6
492    5
735    3
363    5
Name: Smog_Level, Length: 234, dtype: int64
```

```
print(x_train[10]) #these are scaled/normalised values
```

```
[0.18571429 0.07692308 0.36121673 0.57647059 0.43438914 0.43579767]
```

```
#INDIVIDUAL PREDICTION
model.predict([x_train[10]])
```

```
array([4.91500536])
```