1)Preprocessing

Fetching the Dataset

```
import numpy as np
import pandas as pd

dataset=pd.read_csv('/content/drive/MyDrive/ML/ML_Lab/Data.csv')
```

```
Dataset
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	NaN	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	NaN	No
9	France	37.0	67000.0	Yes

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

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

```
X=dataset.iloc[:,:-1].values   #---taking all the columns except the label column
y=dataset.iloc[:,-1].values #---taking the values of the label column print(X)
print(y)
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' nan 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 nan]
 ['France' 37.0 67000.0]]
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Handling missing data

```
from sklearn.impute import SimpleImputer

#---create an instance of the class SimpleImputer imputer=SimpleImputer(missing_values=np.nan, strategy= 'mean') #---
missing_values=np.nan, means all the empty values in the data
#---strategy=mean, median, most_frequent, constant

#---connect the imputer object with the data imputer.fit(X[:,1:3]) #---the column with
numerical data is only fitted to imputer object

#----transform the data(replace the missing data with the mean)

X[:,1:3]=imputer.transform(X[:,1:3]) print(X) #------no missing

values now in the X
```

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 39.875 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 61375.0]
 ['France' 35.0 58000.0]
 ['Spain' 39.875 52000.0]
 ['France' 48.0 79000.0]
```

```
['Germany' 50.0 61375.0]
['France' 37.0 67000.0]]
```

----Arguments----

SimpleImputer(missing_values=nan, strategy='mean', ll_value=None, verbose=0, copy=True, add_indicator=False,) copy-->If

True, a copy of X will be created. If False, imputation will be done in-place whenever possible. ll_value--> When strategy ==

"constant", ll_value is used to replace all occurrences of missing_values.default=None

Encoding Categorical data

Suppose the data is categorical in the dataset and we need to convert it into the numeric format

Here we are having "Purchased" & "Country" column as categorical data

'Country' has 3 values--> France, Spain, Germany, If we assign 0, 1 and 2 numbers respectively to each value then it will be considered as some kind of priority given to each value(0,1 & 2)

To do this, we can do one hot encoding(create different columns for each unique values)

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

#---- Object of Column Transformer Class ct= ColumnTransformer(transformers=
[('encoder',OneHotEncoder(),[0])], remainder='passthrough')
#---3 things in 'transformers'= 1st--> kind of transformation/encoding
#---                            2nd--> Class of encoder
#---                            3rd--> index of the columns that we want to apply the OnHot encoding,here [0] for country column #-
--remainder=passthrough --> that means we want to keep all other features/columns in feature matrix without transformation
#---- {'drop','passthrough'}

print(type(X))
#---- Connect the data with the 'ct' object,here we will use 'fit_transform' which will fit and transform together in 1 step
X=ct.fit_transform(X) #---- update the encoded column to the same feature matrix X
X
```

```
<class 'numpy.ndarray'>
array([[1.0, 0.0, 0.0, 44.0, 72000.0],
[0.0, 0.0, 1.0, 27.0, 48000.0],
       [0.0, 1.0, 0.0, 39.875, 54000.0],
       [0.0, 0.0, 1.0, 38.0, 61000.0],
       [0.0, 1.0, 0.0, 40.0, 61375.0],
       [1.0, 0.0, 0.0, 35.0, 58000.0],
       [0.0, 0.0, 1.0, 39.875, 52000.0],
       [1.0, 0.0, 0.0, 48.0, 79000.0],
       [0.0, 1.0, 0.0, 50.0, 61375.0],
       [1.0, 0.0, 0.0, 37.0, 67000.0]], dtype=object)
```

Encoding the dependent Variable

```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder() y=le.fit_transform(y) #---parameter as
dependent variable print(y)
```

```
[0  1 0 0 1 1 0 1 0 1]
```

Splitting Dataset

```
from sklearn.model_selection import train_test_split
#---returns the 4 matrices-2 for training and 2 for testing


X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)  #--random_state=seed value for splitting order #-
--X stands for feature matrix and y stands for  target matrix


print(X_train)
```

```
[[0.0 0.0 1.0 39.875 52000.0]
 [0.0 1.0 0.0 40.0 61375.0]
 [1.0 0.0 0.0 44.0 72000.0]
 [0.0 0.0 1.0 38.0 61000.0]
 [0.0 0.0 1.0 27.0 48000.0]
 [1.0 0.0 0.0 48.0 79000.0]
 [0.0 1.0 0.0 50.0 61375.0]
 [1.0 0.0 0.0 35.0 58000.0]]
```

```
print(X_test)
```

```
[[0.0 1.0 0.0 39.875 54000.0]
 [1.0 0.0 0.0 37.0 67000.0]]
```

```
print(y_train)
```

```
[0 1 0 0 1 1 0 1]
```

```
print(y_test)
```

```
[0 1]
```

Feature Scaling

Feature scaling is done to prevent over dominate some feature values just based upon their magnitues even if they were not important

2 Methods of Feature Scaling

- Standardization (scales between -3 and +3)

x(stand)=[x-mean(x)]/standard deviation(x) •

Normalization (scales between 0 and +1) x

(norm)=[x-min(x)]/max(x)-min(x)

Normalization is recomended when you have a normal distribution in features Standardization is

used in general conditions

```
from sklearn.preprocessing import StandardScaler

sc=StandardScaler()
X_train[:,3:]=sc.fit_transform(X_train[:,3:])  #---specify the range, takes all rows and selected columns from 3 upto last column
#---fit method will only compute the mean and standard deviation of the feature,then transform method will apply the formula

#------we apply the same scaler to tes set, thats why we use 'transform' method only X_test[:,3:]=sc.transform(X_test[:,3:])
```

```
print(X_train)
```

```
[[0.0 0.0 1.0 -0.05231070414657415 -1.0245464314521104]
 [0.0 1.0 0.0 -0.03411567661733096 -0.023360993551025323]
 [1.0 0.0 0.0 0.5481252043184508 1.1113158360702047]
 [0.0 0.0 1.0 -0.3252361170852219 -0.06340841106706872]
 [0.0 0.0 1.0 -1.9263985396586218 -1.4517188849565736]
 [1.0 0.0 0.0 1.1303660852542325 1.858867629703015]
 [0.0 1.0 0.0 1.4214865257221234 -0.023360993551025323]
 [1.0 0.0 0.0 -0.7619167777870582 -0.38378775119541597]]
```

```
print(X_test)
```

```
[[0.0 1.0 0.0 -0.05231070414657415 -0.8109602046998791]
 [1.0 0.0 0.0 -0.4707963373191673 0.5773502691896258]]
```

Note : We need to apply the same scaler of training data on the test data, otherwise it will compute the different values of mean and standard deviation for the test set and the results would be affected. So that we will get the same transformation on the test set

Do we have to apply the scaling to the dummy variables/ one hot encoders? No, as the scaling takes the values between -3 and +3, the values in dummy varibales lie between 0 and 1 which fall in this range.

Apply feature scaling to numerical values only, not to dummy variables.

Feature scaling should be done after splitting the data because the test set should be totally unseen, and feature scaling scales the features into one scale so that no fetaure dominates others just on the basis of magnitude

Feature scaling nds the mean or standard deviation of the feature in order to perform scaling

Doing feature scaling before, will do some information leakage and the test data will not totally remain unseen

2) SCIKIT LEARN

Importing Datasets from sklearn package

```
import sklearn
from sklearn import datasets
dir(datasets)  #---displays all the datasets in the 'dataset' package of sklearn
```

```
['__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_base',
 '_california_housing',
 '_covtype',
 '_kddcup99',
 '_lfw',
 '_olivetti_faces',
 '_openml',
 '_rcv1',
 '_samples_generator',
 '_species_distributions',
 '_svmlight_format_fast',
 '_svmlight_format_io',
 '_twenty_newsgroups',
 'clear_data_home',
 'dump_svmlight_file',
 'fetch_20newsgroups',
 'fetch_20newsgroups_vectorized',
 'fetch_california_housing',
 'fetch_covtype',
 'fetch_kddcup99',
 'fetch_lfw_pairs',
 'fetch_lfw_people',
 'fetch_olivetti_faces',
 'fetch_openml',
 'fetch_rcv1',
 'fetch_species_distributions',
 'get_data_home',
 'load_boston',
 'load_breast_cancer',
 'load_diabetes',
 'load_digits',
 'load_files',
 'load_iris',
 'load_linnerud',
 'load_sample_image',
 'load_sample_images',
 'load_svmlight_file',
 'load_svmlight_files',
 'load_wine',
 'make_biclusters',
 'make_blobs',
 'make_checkerboard',
 'make_circles',
 'make_classification',
 'make_friedman1',
 'make_friedman2',
 'make_friedman3',
 'make_gaussian_quantiles',
 'make_hastie_10_2',
 'make_low_rank_matrix',
 'make_moons',
 'make_multilabel_classification',
 'make_regression',
 'make_s_curve',
 'make_sparse_coded_signal',
 'make_sparse_spd_matrix',
 'make_sparse_uncorrelated',
 'make_spd_matrix',
 'make_swiss_roll']
```

Load Dataset

```
i=datasets.load_iris() #---Load the dataset
print(type(i)) print(i)
```

```
<class 'sklearn.utils.Bunch'>
{'data': array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4],
       [4.6, 3.4, 1.4, 0.3],
       [5. , 3.4, 1.5, 0.2],
       [4.4, 2.9, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.1],
       [5.4, 3.7, 1.5, 0.2],
       [4.8, 3.4, 1.6, 0.2],
       [4.8, 3. , 1.4, 0.1],
       [4.3, 3. , 1.1, 0.1],
       [5.8, 4. , 1.2, 0.2],
       [5.7, 4.4, 1.5, 0.4],
       [5.4, 3.9, 1.3, 0.4],
       [5.1, 3.5, 1.4, 0.3],
       [5.7, 3.8, 1.7, 0.3],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
       [4.7, 3.2, 1.6, 0.2],
       [4.8, 3.1, 1.6, 0.2],
       [5.4, 3.4, 1.5, 0.4],
       [5.2, 4.1, 1.5, 0.1],
       [5.5, 4.2, 1.4, 0.2],
       [4.9, 3.1, 1.5, 0.2],
       [5. , 3.2, 1.2, 0.2],
       [5.5, 3.5, 1.3, 0.2],
       [4.9, 3.6, 1.4, 0.1],
       [4.4, 3. , 1.3, 0.2],
       [5.1, 3.4, 1.5, 0.2],
       [5. , 3.5, 1.3, 0.3],
       [4.5, 2.3, 1.3, 0.3],
       [4.4, 3.2, 1.3, 0.2],
       [5. , 3.5, 1.6, 0.6],
       [5.1, 3.8, 1.9, 0.4],
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
```

Feature names/column names of the dataset

```
features=i.feature_names #---fetch the feature names or the column names
print(features)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

Print the loaded dataset

```
print(i.data) #---print the loaded dataset feature matrix
```

```
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
```

```
target=i.target  #---gets the labels associated with the data points
print(target)
```

```
[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 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

```
print(i.target_names)  #--displays the target names associated with values 0 and 1 and 2
```

```
['setosa' 'versicolor' 'virginica']
```

```
print(i.DESCR)  #--gives all the detailed description about the dataset
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa. ov)
```

```
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day. (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant. One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other. .. topic:: References

- Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
Mathematical Statistics" (John Wiley, NY, 1950).
- Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
- Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments". IEEE Transactions on Pattern Analysis and Machine
     I t lli         V l PAMI 2 N 1 67 71
```

Download any dataset from openml repository

```
from sklearn.datasets import fetch_openml
mice=fetch_openml(name='miceprotein',version=4)
mice
```

```
{'data':       DYRK1A_N   ITSN1_N    BDNF_N     NR1_N    NR2A_N    pAKT_N   pBRAF_N  \
0     0.503644  0.747193  0.430175  2.816329  5.990152  0.218830  0.177565
1     0.514617  0.689064  0.411770  2.789514  5.685038  0.211636  0.172817
2     0.509183  0.730247  0.418309  2.687201  5.622059  0.209011  0.175722
3     0.442107  0.617076  0.358626  2.466947  4.979503  0.222886  0.176463
4     0.434940  0.617430  0.358802  2.365785  4.718679  0.213106  0.173627
      ...        ...       ...       ...       ...       ...       ...
      ...
1075  0.254860  0.463591  0.254860  2.092082  2.600035  0.211736  0.171262
1076  0.272198  0.474163  0.251638  2.161390  2.801492  0.251274  0.182496
1077  0.228700  0.395179  0.234118  1.733184  2.220852  0.220665  0.161435
1078  0.221242  0.412894  0.243974  1.876347  2.384088  0.208897  0.173623
      1079  0.302626  0.461059  0.256564  2.092790  2.594348  0.251001
      0.191811

       pCAMKII_N   pCREB_N    pELK_N  ...     SHH_N     BAD_N  BCL2_N  \
0      2.373744  0.232224  1.750936  ...  0.188852  0.122652     NaN
1      2.292150  0.226972  1.596377  ...  0.200404  0.116682     NaN
2      2.283337  0.230247  1.561316  ...  0.193685  0.118508     NaN
3      2.152301  0.207004  1.595086  ...  0.192112  0.132781     NaN
4      2.134014  0.192158  1.504230  ...  0.205604  0.129954     NaN    ...
       ...        ...       ...  ...       ...       ...     ...
1075   2.483740  0.207317  1.057971  ...  0.275547  0.190483     NaN
1076   2.512737  0.216339  1.081150  ...  0.283207  0.190463     NaN
1077   1.989723  0.185164  0.884342  ...  0.290843  0.216682     NaN
1078   2.086028  0.192044  0.922595  ...  0.306701  0.222263     NaN    1079
       2.361816  0.223632  1.064085  ...  0.292330  0.227606     NaN

          pS6_N   pCFOS_N     SYP_N  H3AcK18_N    EGR1_N  H3MeK4_N    CaNA_N
0     0.106305  0.108336  0.427099   0.114783  0.131790  0.128186  1.675652
1     0.106592  0.104315  0.441581   0.111974  0.135103  0.131119  1.743610
2     0.108303  0.106219  0.435777   0.111883  0.133362  0.127431  1.926427
3     0.103184  0.111262  0.391691   0.130405  0.147444  0.146901  1.700563
4     0.104784  0.110694  0.434154   0.118481  0.140314  0.148380  1.839730
      ...        ...       ...       ...        ...       ...       ...
      ...
1075  0.115806  0.183324  0.374088   0.318782  0.204660  0.328327  1.364823
1076  0.113614  0.175674  0.375259   0.325639  0.200415  0.293435  1.364478
1077  0.118948  0.158296  0.422121   0.321306  0.229193  0.355213  1.430825
1078  0.125295  0.196296  0.397676   0.335936  0.251317  0.365353  1.404031
      1079  0.118899  0.187556  0.420347   0.335062  0.252995  0.365278
      1.370999
 [1080 rows x 77 columns], 'target': 0       c-CS-m
1       c-CS-m
2       c-CS-m
3       c-CS-m
4       c-CS-m          ...   1075    t-SC-s
1076    t-SC-s
1077    t-SC-s
1078    t-SC-s
1079    t-SC-s
 Name: class, Length: 1080, dtype: category
 Categories (8, object): ['c-CS-m', 'c-CS-s', 'c-SC-m', 'c-SC-s', 't-CS-m', 't-CS-s', 't-SC-m',
                          't-SC-s'], 'frame':       DYRK1A_N   ITSN1_N    BDNF_N     NR1_N    NR2A_N    pAKT_N   pBRAF_N  \
0     0.503644  0.747193  0.430175  2.816329  5.990152  0.218830  0.177565
1     0.514617  0.689064  0.411770  2.789514  5.685038  0.211636  0.172817
2     0.509183  0.730247  0.418309  2.687201  5.622059  0.209011  0.175722
3     0.442107  0.617076  0.358626  2.466947  4.979503  0.222886  0.176463
```

```
4     0.434940  0.617430  0.358802  2.365785  4.718679  0.213106  0.173627
```

Suppose data is of Excel, Jason, SQL, CSV le-then its best to use PANDAS library

If the le is binary like .mat then its recomended to use scipy library

Polynomial data-numpy array

Image and video-load it in numpy array using skimage library

3) Linear Regression

In the below example, we will be working on a housing dataset trying to create a model to predict housing prices based upon the existing features

```
import pandas as pd
import numpy as np

df=pd.read_csv('/content/drive/MyDrive/ML LAB/USA_Housing.csv')
df.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820

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

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

```
df.describe()  #---gives an account of statistical numbers of the dataframe
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

Divide the dataset into data and labels

```
X=df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
y=df['Price']
```

Divide the dataset into train and test dataset

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=101)
#---random_state gives us the randomness in the split of the dataset
```

Create and Train the Model

```
from sklearn.linear_model import LinearRegression
```

```
lm=LinearRegression()
lm.fit(X_train,y_train)
```

```
LinearRegression()
```

Evaluating the Model

```
print(lm.intercept_)
#---returns the intercept of the model
```

```
-2640159.7968526958
```

```
lm.coef_
#---returns the coefficient of each feature
```

```
array([2.15282755e+01, 1.64883282e+05, 1.22368678e+05, 2.23380186e+03,
       1.51504200e+01])
```

```
X_train.columns
```

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population'],
      dtype='object')
```

create the dataframe of the coefficients and their corresponding features

```
cdf=pd.DataFrame(lm.coef_,X.columns,columns=['Coeff'])
cdf
```

	Coeff
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Predictions

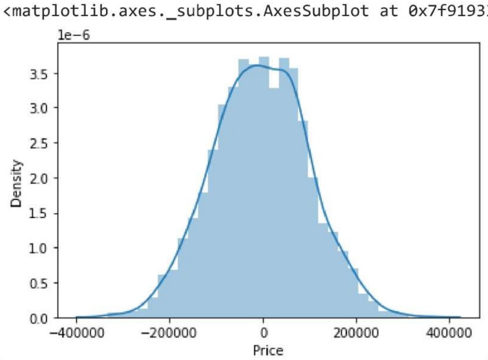
```
predictions=lm.predict(X_test)
print(predictions)
#----prints the predicted prices of the test data
```

```
[1260960.70567627  827588.75560329 1742421.24254344 ...  372191.40626917
 1365217.15140898 1914519.5417888 ]
```

```
import seaborn as sns
```

```
sns.distplot((y_test-predictions))
#---Plotting the residuals i.e, |Original values -predicted values|
#---If you have normally distributed residuals then the model is correct choice
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a de
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f9193336700>
```



Evaluating Metrics

Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

```
from sklearn import metrics

MAE=metrics.mean_absolute_error(y_test,predictions)
MSE=metrics.mean_squared_error(y_test,predictions)
RMSE=np.sqrt(MSE)
print('MAE=',MAE)
print('MSE=',MSE)
print('RMSE=',RMSE)
```

```
MAE= 82288.22251914942
MSE= 10460958907.208977
RMSE= 102278.82922290897
```

2) Polynomial Regression

Comparing Linear Vs polynomial

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

dataset = pd.read_csv('/content/drive/MyDrive/ML LAB/Position_Salaries.csv')
print(dataset.head())
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values
```

```
            Position  Level  Salary
0   Business Analyst      1   45000
1  Junior Consultant      2   50000
2  Senior Consultant      3   60000
3            Manager      4   80000
4    Country Manager      5  110000
```

Training the Linear Regression model on the whole dataset

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

```
LinearRegression()
```

Training the Polynomial Regression model on the whole dataset

```
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, y)
```

```
LinearRegression()
```

-----Arguments----

PolynomialFeatures(degree=2 (---degree of polynomial features),

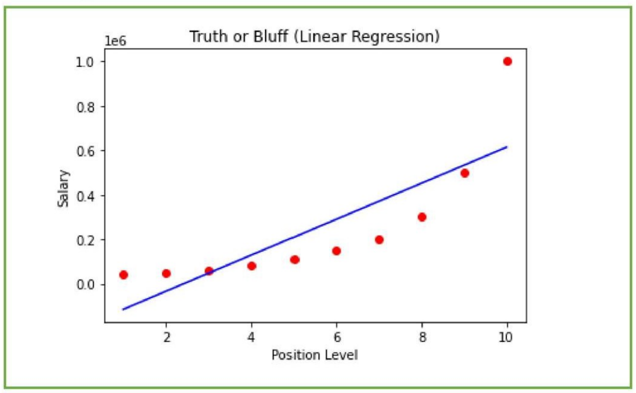
`interaction_only=False(produces interaction features if true, ignores x[1] ^2 or x[1]^3),`

`include_bias=True(---If True (default),then include a bias column, the feature in which all polynomial powers are zero),`

)

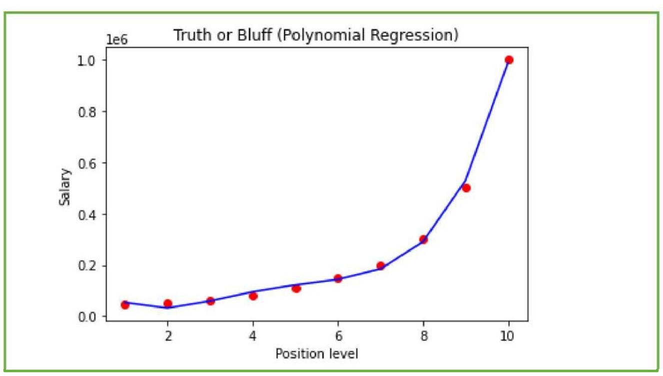
Visualising the Linear Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



Visualising the Polynomial Regression results

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluff (Polynomial Regression)')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```



4) Logistic Regression

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

dataset = pd.read_csv('/content/drive/MyDrive/ML LAB/Position_Salaries.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].values

from sklearn.svm import SVR
regressor = SVR(kernel = 'rbf')
regressor.fit(X, y)
regressor.predict([[6.5]])
```

```
array([130001.82883924])array([130001.82883924])
```

-----Arguments----

SVR(kernel='rbf' (---'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable.default='rbf'),

```
degree=3(--- If we are using 'poly' kernel then it has to be specified, otherwise ignored by other kernels),

gamma='scale'(---If gamma is large, then variance is small implying the support vector does not have wide-spread influence. large gamma leads to h
{'scale', 'auto'} ,Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.
If gamma='scale' then it uses 1 / (n_features * X.var()) and gamma='auto the it uses 1 / n_features),

coef0=0.0 (---Independent term in kernel function.Significant for 'poly' and 'sigmoid'),

tol=0.001 (---Tolerance for stopping criterion.default=1e-3),

C=1.0 (---C is the parameter for the soft margin cost function, which controls the influence of each individual support vector; this process invol

epsilon=0.1 (---It controls the width of the ε-insensitive zone, used to fit the training data. The value of ε can affect the number of support ve

shrinking=True,

cache_size=200 (---Specify the size of the kernel cache (in MB)),

verbose=False,

max_iter=-1 (---Hard limit on iterations within solver, or -1 for no limit.default=-1),
```

)

0s completed at 10:06 PM

2) LOGISTIC REGRESSION

```
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).
```

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

df=pd.read_csv('/content/drive/MyDrive/ML Lab/HCP/kyphosis.csv')


print(df.head())
from sklearn.model_selection import train_test_split

X=df.drop('Kyphosis',axis=1)
y=df['Kyphosis']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)


#---Dataset represents the patients condition on kyphosis(situation on spinal condition) #---
After surgery whether the kyphosis condition was present or absent in the patient
#---Age=age is the person in months
#---Number- is the number of vetebrae involved in operation
#---start is the top most vertebare operated in the operation
```

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

```
from sklearn.linear_model import LogisticRegression

logmodel=LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
LogisticRegression()
```

LogisticRegression(penalty='l2'(used to specify the penalty : {'l1', 'l2', 'elasticnet', 'none'}, default='l2'),

```
dual=False(---Dual or primal formulation. Dual formulation is only implemented for l2 penalty with liblinear solver. Prefer dual=False when n_samp

tol=0.0001 (---Tolerance for stopping criteria,default=False),

C=1.0 (---its inverse of regularization strength,smaller values means stronger regularization,default=1.0),

fit_intercept=True (---Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function),

intercept_scaling=1,

class_weight=None (---Weights associated with classes in the form {class_label: weight}. If not specified than class all weights are 1.),

random_state=None (---The seed of the pseudo random number generator to use when shuffling the data.),

solver='lbfgs' (---Algorithm to use in the optimization problem,solver={'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}each has its own uniquene

max_iter=100 (---Maximum number of iterations taken for the solvers to converge.),

multi_class='auto'(--- {'auto', 'ovr', 'multinomial'}, default='auto'. 'ovr' for binary classification),

verbose=0,
```

```
warm_start=False,

n_jobs=None (---Number of CPU cores used when parallelizing over classes if multi_class='ovr'".),

l1_ratio=None (---its a combiation of l1 and l2),
```

```
predictions=logmodel.predict(X_test)
print(predictions)
```

```
['absent' 'absent' 'absent' 'absent' 'absent' 'absent' 'absent' 'absent'
 'absent' 'absent' 'present' 'present' 'absent' 'absent' 'absent' 'absent'
 'absent' 'absent' 'absent' 'present' 'absent' 'absent' 'present'
 'present' 'absent']
```

Evaluating Metrics

Classi cation Report

```
from sklearn.metrics import classification_report

print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

      absent       0.90      0.82      0.86        22
     present       0.20      0.33      0.25         3

    accuracy                           0.76        25
   macro avg       0.55      0.58      0.55        25
weighted avg       0.82      0.76      0.78        25
```

Confusion Metrics

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test,predictions)
```

```
array([[18,  4],
       [ 2,  1]])
```

5) KNN Classifier

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

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

```
import pandas as pd
import numpy as np

df=pd.read_csv('/content/drive/MyDrive/ML Lab/HCP/Classified Data',index_col=0)

print(df.head()) from sklearn.model_selection import train_test_split

X=df.drop('TARGET
CLASS',axis=1)
```
2

```
0  0.643798  0.879422  1.231409
2  1.154483  0.957877  1.285597          0  3  1.380003  1.522692  1.153093
4  0.646691  1.463812  1.419167
```
2
2
2

```
y=df['TARGET CLASS']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=100)
```

```
        WTT       PTI       EQW       SBI       LQE       QWG       FDJ  \
0 0.913917  1.162073  0.567946  0.755464  0.780862  0.352608  0.759697
1  0.635632  1.003722  0.535342  0.825645  0.924109  0.648450  0.675334
2  0.721360  1.201493  0.921990  0.855595  1.526629  0.720781  1.626351
3  1.234204  1.386726  0.653046  0.825624  1.142504  0.875128  1.409708
4  1.279491  0.949750  0.627280  0.668976  1.232537  0.703727  1.115596

        PJF       HQE       NXJ  TARGET CLASS
1  1.013546  0.621552  1.492702             0
```

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train) pred=knn.predict(X_test)
```

----Arguments----

KNeighborsClassi er(

```
n_neighbors=5,

weights='uniform'(---'uniform' or 'callable'),

algorithm='auto'({'auto', 'ball_tree', 'kd_tree', 'brute'},Algorithm used to compute the nearest neighbors),
leaf_size=30,
p=2(---Power parameter. When p = 1, this is
equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2),
metric='minkowski',
metric_params=None(---the distance metric to use for the tree),
n_jobs=None,
)
```

```
from sklearn.metrics import classification_report,confusion_matrix

print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
[[98 12]
 [ 8 82]]
              precision    recall  f1-score   support

0       0.92      0.89      0.91       110
1       0.87      0.91      0.89        90
    accuracy                          0.90       200
macro avg       0.90      0.90      0.90       200
weighted avg       0.90      0.90      0.90       200
```

KNN using Standard Scaler

1) Split the Dataset

```
#------Here we are not knowing that what are the features so how to group the data points?
#---- If the values of some features are higher than it is required to do the feature scaling otherwise such features will show a very majo
#---- it will have much effect on the distance between the features
```

```
import pandas as pd
import numpy as np

df=pd.read_csv('/content/drive/MyDrive/ML Lab/HCP/Classified Data',index_col=0)

print(df.head()) from sklearn.model_selection import train_test_split

X=df.drop('TARGET CLASS',axis=1)

y=df['TARGET CLASS']

X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=100)
```

```
        WTT       PTI       EQW       SBI       LQE       QWG       FDJ  \
0 0.913917  1.162073  0.567946  0.755464  0.780862  0.352608  0.759697
1  0.635632  1.003722  0.535342  0.825645  0.924109  0.648450  0.675334
2  0.721360  1.201493  0.921990  0.855595  1.526629  0.720781  1.626351
3  1.234204  1.386726  0.653046  0.825624  1.142504  0.875128  1.409708
4  1.279491  0.949750  0.627280  0.668976  1.232537  0.703727  1.115596

        PJF       HQE       NXJ  TARGET CLASS
0  0.643798  0.879422  1.231409             1
1  1.013546  0.621552  1.492702             0
2  1.154483  0.957877  1.285597             0
3  1.380003  1.522692  1.153093             1
4  0.646691  1.463812  1.419167             1
```

2) Scale the Splitted dataset

1st Fit the data

2nd Transform the data

```
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()
scaler.fit(X_train) #--- It will drop the target class as we dont want to scale the labels
```

```
StandardScaler()
```

```
scaled_features_X_train=scaler.transform(X_train)
scaled_features_X_test=scaler.transform(X_test)
```

3) Apply KNN Model on the scaled dataset

```
from sklearn.neighbors import KNeighborsClassifier

knn=KNeighborsClassifier(n_neighbors=1) #---means k=1
knn.fit(scaled_features_X_train,y_train)
pred_1=knn.predict(scaled_features_X_test) pred
```

```
array([0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0,
1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0,
0, 1])
```

4) Find the Classi cation Report for KNN =1 using scaled Data

```
from sklearn.metrics import classification_report,confusion_matrix

 print(confusion_matrix(y_test,pred_1))
print(classification_report(y_test,pred_1))

#---Here you can see that the number of Misclassifications(17) in scaled dataset is more as compared to unscaled dataset(20).
```

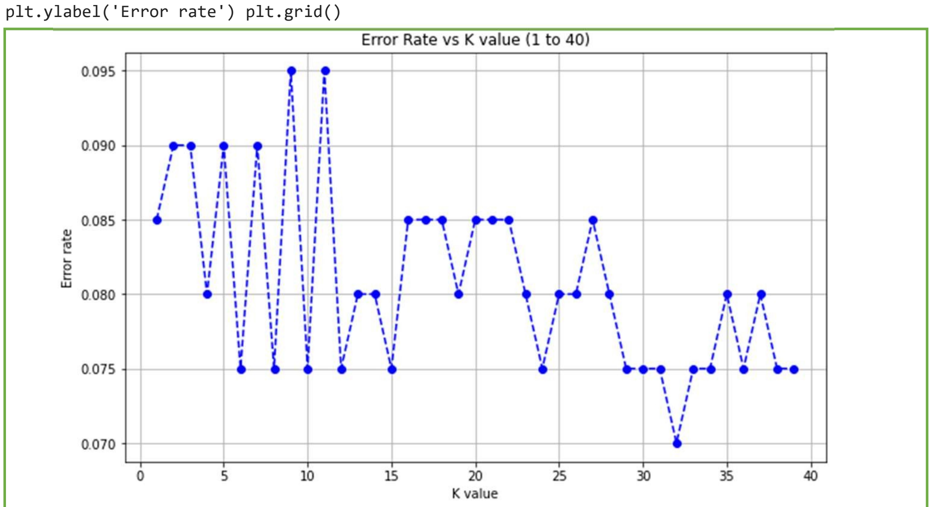
```
[[98 12] [ 5 85]]
              precision    recall  f1-score   support

           0       0.95      0.89      0.92       110
1       0.88      0.94      0.91        90

    accuracy                           0.92       200
macro avg       0.91      0.92      0.91       200
weighted avg       0.92      0.92      0.92       200
```

'Elbow' method to nd correct value of 'k'

```
#------Use elbow method to choose correct value of k
#------Use the model with different values of 'k' and plot the error rate
#---- and observe which one has minimum error rate

error_rate=[]  # ----empty list

for i in range(1,40):
    knn=KNeighborsClassifier(n_neighbors=i)
knn.fit(scaled_features_X_train,y_train)
pred_i=knn.predict(scaled_features_X_test)
error_rate.append(np.mean(pred_i != y_test))
    # ----taking the mean of all prediction and actual labels which are not equal

print(error_rat)
```

```
[0.085, 0.09, 0.09, 0.08, 0.09, 0.075, 0.09, 0.075, 0.095, 0.075, 0.095, 0.075, 0.08, 0.08, 0.075, 0.085, 0.085, 0.085, 0.08, 0.085
```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='--',marker='o')
plt.title('Error Rate vs K value (1 to 40)') plt.xlabel('K value')
plt.ylabel('Error rate') plt.grid()
```




```
knn=KNeighborsClassifier(n_neighbors=32)
```

```
knn.fit(scaled_features_X_train,y_train)
pred_28=knn.predict(scaled_features_X_test)

print(confusion_matrix(y_test,pred_28))
print('\n')
print(classification_report(y_test,pred_28))

#---Compare the confusion matrix for k=1 and for k=28 , it has better classsification

#---Misclassifications without Scaled dataset : 20
#---Misclassifications with Scaled dataset :17
#---Misclassification with better 'k' value and scaled dataset :16
```

```
[[99 11]
 [ 3 87]]


              precision    recall  f1-score   support

           0       0.97      0.90      0.93       110
           1       0.89      0.97      0.93        90

    accuracy                           0.93       200
   macro avg       0.93      0.93      0.93       200
weighted avg       0.93      0.93      0.93       200
```