Saurabh Kumar Mittal

9818251855

1. What are Eigenvectors and Eigenvalues? How are they relevant in the field of Data Science/Machine Learning?

Ans. Eigenvectors and Eigenvalues are the beautiful concepts of the Linear Algebra. Eigenvalues and their corresponding eigenvectors give us extremely important information about the long-term behavior of a linear model. In statistics, eigenvalue/ eigenvector computations are used in principal components and factor analysis; in multivariate analysis of variance (MANOVA); and in discriminant analysis. The elements of an eigenvector can be transformed into weights that can be used to combine either raw scores or z scores (to form a new weighted linear combination). Elements of an eigenvector are also scaled to represent "factor loadings" in principal components analysis. These weighted linear combinations may be created in a way that maximizes the degree to which a correlation matrix or variance / covariance matrix can be reproduced (in principal components); or maximizes the differences of this weighted linear combination across groups and minimize variance within groups (in MANOVA or discriminant analysis).

Eigenvalue/ eigenvector solutions are ways of handling relatively large numbers of variables, at least some of which may be thought to "measure the same thing" or "be part of some pattern". A transformation of the corresponding eigenvalue provides information about proportion of explained variance (or ratio of between/ within group variance) for each eigenvector.

Eigenvalues and eigenvectors come in pairs.

The number of eigenvalue/eigenvector solutions for a problem may vary depending on things such as the number of variables or the number of groups. So, by using the eigenvalues/eigen vectors, we can find the main axes of our data. The first main axes are also called principal component, is the axis in which the data varies the most. The second main axis is called the second principal components, is the axis with the second largest variations and so on.
SOME OF THE FUNCATIONS AND LIBRARY THAT IS USED IN PYTHON FOR THE CALCULATION OF EIGENVALUES/EIGENVECTORS:

```
Import numpy as np
N=np.cov (X, rowvar = False)
Eigenvalues, Eigenvectors = np. Linalg.eig(c)
```

The eigenvectors show us the direction of our main axes (principal components) of our data. The greater the eigenvalue, the greater the variation along this axis. So, the eigenvector with the largest eigenvalue corresponds to the axis with the most variance.

2. What do you understand about Imbalanced Data? How are these issues usually resolved?

Ans. Imbalanced data refers to a situation, primarily in classification machine learning, where one target class represents a significant portion of observations. Imbalanced data frequently occurs in real-world problems, so it's a situation data scientist often have to deal with.

For example:

Resting heart rate	Age	Illness present
100	65	1
65	60	0
75	40	0
50	30	0
80	35	0
85	45	0
60	55	0

As in the data, there are 9 observations, of which one is +ve case, meaning that 10% of our example are positive class. This would commonly classified as imbalanced datasets. So, it is very challenging for model training and evaluations. So, to solve this type of problems, we required assumptions according to various related tables and relate the data sets and after changing with the observations, we send our data sets for further training.

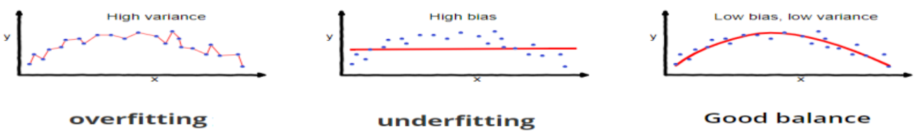
3.Define bias-variance trade-off?

Ans. Bias and variance are compliments if each other's. Increase of one in result of decrease of other vice-versa.

- Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
- Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.

In supervised learning, **underfitting** happens when a model unable to capture the underlying pattern of the data. These models usually have high bias and low variance. It happens when we have very less amount of data to build an accurate model or when we try to build a linear model with a nonlinear data. Also, these kinds of models are very simple to capture the complex patterns in data like Linear and logistic regression.

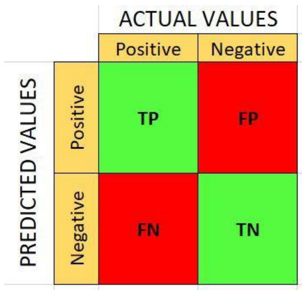
Models with high variance pays a lot of attentions to the training data and does not generalize on the data which it hasn't seen before. As, a result such model performs very well in training data but has high error in the test data.



4. Define the confusion matrix?

Ans. In the field of machine learning and specially the problem of statistical classification, confusion matrix is also known as error matrix. It is NxN matrix used for evaluating the performance of classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by machine learning model.

- A **good model** is one which has high TP and TN rates, while low FP and FN rates.
- If you have an imbalanced dataset to work with, it's always better to use confusion matrix as your evaluation criteria for your machine learning model.

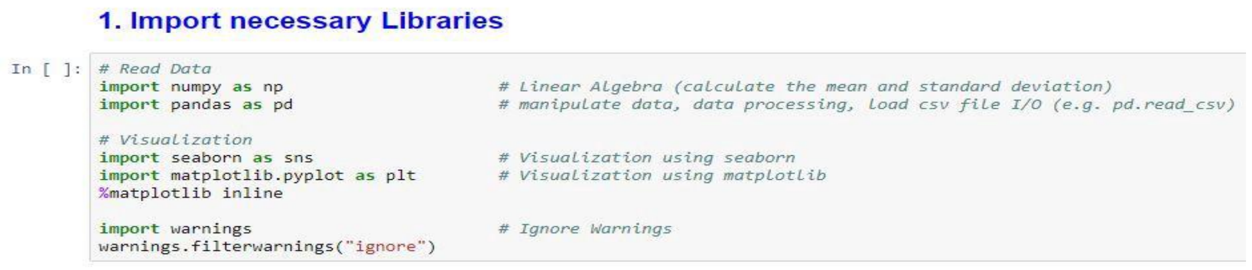


Accuracy = TP+TN/100, Error rate = 1-accuracy, precision = TP/Predicted yes, Recall = TP/actual yes.

For ex. If we take: X= predicted value, Y= Actual value

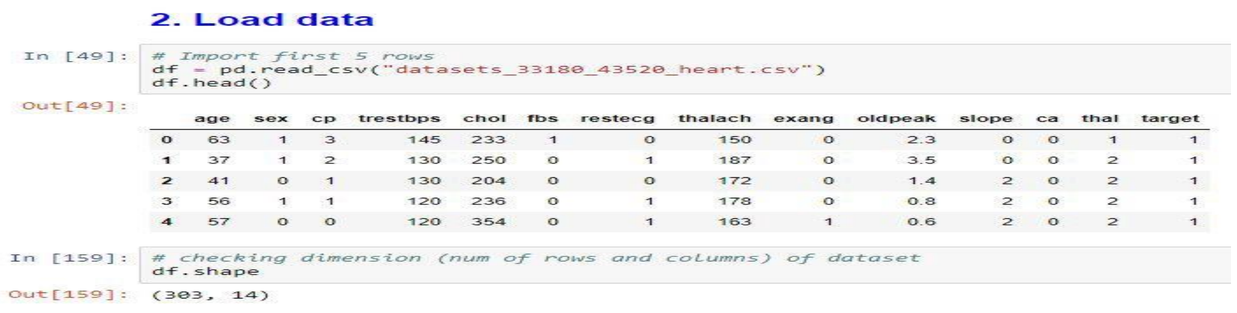
165(SAMPLES)	NO	YES	
NO	TN (50)	FP (10)	60
YES	FN (5)	TP (100)	105
	55	110	

Accuracy = 100+50/100 = 165 = 0.91, ERROR RATE = 1-0.91 = 0.09, Precision = 100/110 = 0.64, Recall = 100/105 = 0.95



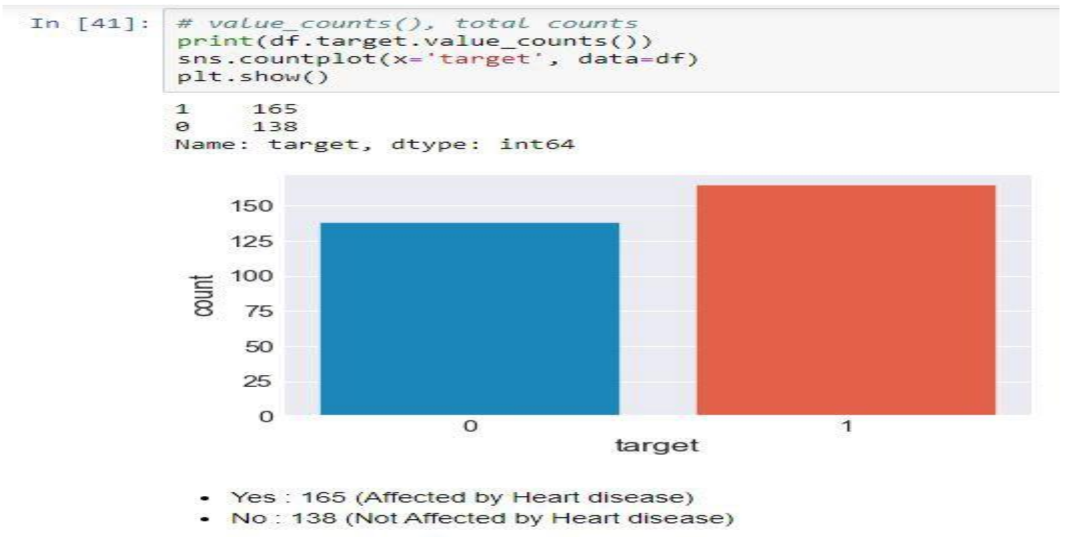
1. Import necessary Libraries

```
In [ ]: # Read Data
        import numpy as np                   # Linear Algebra (calculate the mean and standard deviation)
        import pandas as pd                  # manipulate data, data processing, load csv file I/O (e.g. pd.read_csv)

        # Visualization
        import seaborn as sns                # Visualization using seaborn
        import matplotlib.pyplot as plt      # Visualization using matplotlib
        %matplotlib inline

        import warnings                      # Ignore Warnings
        warnings.filterwarnings("ignore")
```



2. Load data

```
In [49]: # Import first 5 rows
         df = pd.read_csv("datasets_33180_43520_heart.csv")
         df.head()

Out[49]:
            age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal  target
         0   63    1   3       145   233    1        0      150      0      2.3      0   0     1       1
         1   37    1   2       130   250    0        1      187      0      3.5      0   0     2       1
         2   41    0   1       130   204    0        0      172      0      1.4      2   0     2       1
         3   56    1   1       120   236    0        1      178      0      0.8      2   0     2       1
         4   57    0   0       120   354    0        1      163      1      0.6      2   0     2       1

In [159]: # checking dimension (num of rows and columns) of dataset
          df.shape

Out[159]: (303, 14)
```



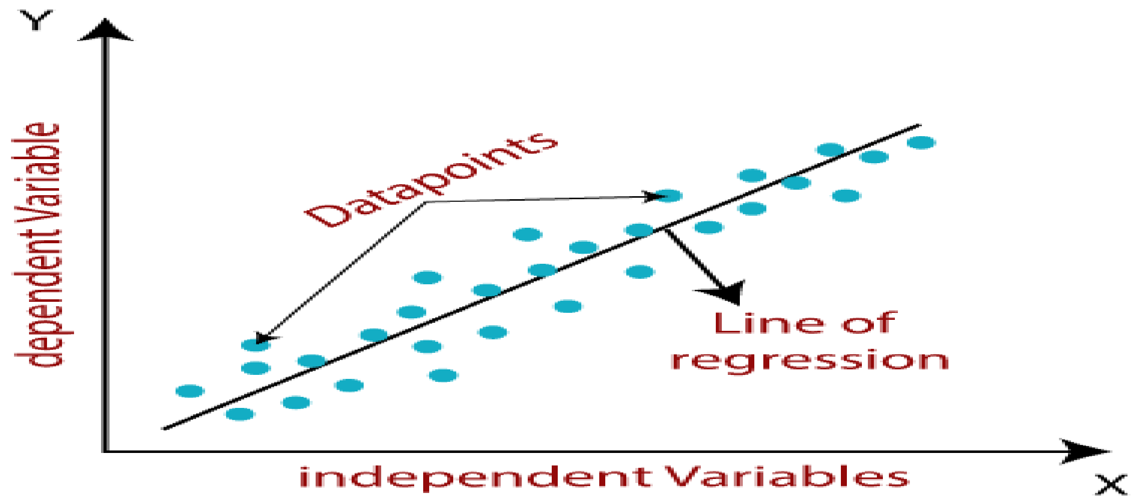
```
In [41]: # value_counts(), total counts
         print(df.target.value_counts())
         sns.countplot(x='target', data=df)
         plt.show()

1    165
0    138
Name: target, dtype: int64
```

- Yes : 165 (Affected by Heart disease)
- No : 138 (Not Affected by Heart disease)

After that model building and evaluation takes place. We can go for SVM, RANDOM FOREST ETC.

5. What is Linear Regression? What are some of the major drawbacks of the linear model? State an example where you have recently used linear regression.

Ans. In a data set, there is two variable, independent and dependent, for ex: if we want to find the annual sales from the company data sets, then first we have to look for the relation of columns, what depends on what. The dependent variable should be must in continuous in nature. And both X & Y should be linear in nature, for ex. If the value of X will increase then the value of Y should also increase. There are two types of linear regression: 1. Simple linear regression and 2. Multiple linear regression.



The model finds the best fit line between independent and dependent variable point. Algorithm depends upon the datasets. Y=(MX+C) ------ Linear Regression.

In a simple linear regression, uses traditional slope -intercept form. Where X represents our input and Y represents our predictions.

In a Multiple linear regression, F (x, y, z) = w1*x+w2*y+w3*z, where x, y, z represents the attributes, or a distinct piece of information's.

Main limitation of Linear Regression is the assumption of linearity between the dependent variable and the independent variables. In the real world, the data is rarely linearly separable. It assumes that there is a straight-line relationship between the dependent and independent variables which is incorrect many times.

6. What is a Gradient and Gradient Descent? How is it used? Show with an example.

Ans. Gradient descent (GD) is used to find stationary points (hopefully local maxima) on functions/curves. Often, we cannot find these stationary points analytically, that is find exact solutions using pen and paper, and so we resort to numerical algorithms like GD to do this for us. This is the case when we try to find optimal parameters/weights in artificial neural networks and many other machine learning models. There are other reasons why GD might be preferred over numerical optimization algorithms in specific problems, but given your question, this is the gist of it.

When we're building machine learning models, one approach to training them (apart from using established closed form solutions) is iteratively searching for the best combination of coefficients.

BY TAKING ONE EXAMPLE: imagine the path of a river originating from top of a mountain.

If a river starts from the top hill of the mountains then, it will reach the base point of where the river come with the level, so the during the flow path, if there is an obstacle, so minimize the obstacles, that we are reducing the local minima path of the river. For that in machine learning algorithms we use various algorithms to optimize the gradient descents (e.g., lasagne's, caffe's and kera's documentations). These algorithms, however, are often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

7. What are Support Vectors in SVM (Support Vector Machine)? Explain with a use case.

Ans. SVM, is a sets of supervised learning methods use for classification, regression, and outlier's detection. All of these are common tasks in machine learning.

There are several types of SVM for particular machine learning problems, like support vectors regressor which is extension of support vector classification. These are the math equation, which turns to gives the answers more accurately and fast.

A simple linear SVM classifier works by making a straight line between two classes. That means all of the data points on one side of the line will represent a category and the data points on the other side of the line will be put into a different category. This means there can be an infinite number of lines to choose from. like k-nearest neighbors, is that it chooses the best line to classify your data points. It chooses the line that separates the data and is the furthest away from the closet data points as possible.

Types of SVMs

There are two different types of SVMs, each used for different things:

- Simple SVM: Typically used for linear regression and classification problems.
- Kernel SVM: Has more flexibility for non-linear data because you can add more features to fit a hyperplane instead of a two-dimensional space.
- SVMs are used in applications like handwriting recognition, intrusion detection, face detection, email classification, gene classification, and in web pages. This is one of the reasons we use SVMs in machine learning. It can handle both classification and regression on linear and non-linear data.
- Another reason we use SVMs is because they can find complex relationships between your data without you needing to do a lot of transformations on your own. It's a great option when you are working with smaller datasets that have tens to hundreds of thousands of features. They typically find more accurate results when compared to other algorithms because of their ability to handle small, complex datasets.

Examples with datasets

To show how SVMs work in practice, we'll go through the process of training a model with it using the Python Scikit-learn library. This is commonly used on all kinds of machine learning problems and works well with other Python libraries.
Here are the steps regularly found in machine learning projects:

- Import the dataset
- Explore the data to figure out what they look like
- Pre-process the data
- Split the data into attributes and labels
- Divide the data into training and testing sets
- Train the SVM algorithm
- Make some predictions
- Evaluate the results of the algorithm

Linear SVM Example

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import svm
```

For a simple linear example, we'll just make some dummy data and that will act in the place of importing a dataset.

```
# linear data
X = np.array([1, 5, 1.5, 8, 1, 9, 7, 8.7, 2.3, 5.5, 7.7, 6.1])
y = np.array([2, 8, 1.8, 8, 0.6, 11, 10, 9.4, 4, 3, 8.8, 7.5])
```

The reason we're working with numpy arrays is to make the matrix operations faster because they use less memory than Python lists.

```
# show unclassified data
plt.scatter(X, y)
plt.show()
# train the model
clf.fit(training_X, training_y)
# show the plot visually
plt.scatter(training_X[:, 0], training_X[:, 1], c=training_y)
plt.legend()
plt.show()
```

Non-Linear SVM Example

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn import svm
# non-linear data
circle_X, circle_y = datasets.make_circles(n_samples=300, noise=0.05), just considering the values for example only.
# show raw non-linear data
plt.scatter(circle_X[:, 0], circle_X[:, 1], c=circle_y, marker='.')
plt.show()
# make non-linear algorithm for model
nonlinear_clf = svm.SVC(kernel='rbf', C=1.0)
# training non-linear model
nonlinear_clf.fit(circle_X, circle_y)
# plot data and decision boundary
plt.scatter(circle_X[:, 0], circle_X[:, 1], c=circle_y, s=50)
plot_decision_boundary(nonlinear_clf)
```

plt.scatter(nonlinear_clf.support_vectors_[:, 0], nonlinear_clf.support_vectors_[:, 1], s=50, lw=1, facecolors='none')
plt.show()
NOTE: SIMPLY ASSUMING ALL THE VALUES

8. How would you handle a dataset with missing values of more than 30%?
Ans. After import the data sets, first we will check the data set by writing one function i.e., print(data.isna().sum()), we can use this function to check the missing values insides the data sets.
If the data sets are very large, then we can drop the values or columns. And for categorical value missing, then by using mode function, we can fill the value.
For numerical values, we use mean and median values.
If there is any missing value, then by using simple imputer.
from sklearn.impute import SimpleImputer
SI=SimpleImputer (missing_values=np.nan, strategy=" median"), we use this code to fill the missing values inside the data sets.

9. Why is data cleaning crucial? How do you clean the data?
Ans. Data cleaning is the process of fixing incorrect, incomplete, missing, and duplications data in the data sets. It helps to get the more accuracy about the data sets, by using the several machine learning algorithms.

```
 import pandas as pd
data=pd.read_csv("C:/Users/kiit/OneDrive - Brandscapes Consultancy Pvt. Ltd/Desktop/L&B data sets/50_Startups.csv")
print(data.isna().sum())
from sklearn.impute import SimpleImputer
SI=SimpleImputer()

SI_R__D=SI.fit(data[['R&D Spend']])
data['R&D Spend']=SI_R__D.transform(data[['R&D Spend']])



SI_Administration=SI.fit(data[['Administration']])
data['Administration']=SI_Administration.transform(data[['Administration']])



SI_Marketing_Spend=SI.fit(data[['Marketing Spend']])
```

```
data['Marketing Spend']=SI_Marketing_Spend.transform(data[['Marketing Spend']])


SI_State=SimpleImputer(strategy='most_frequent')
SI_State=SI_State.fit(data[['State']])
data['State']=SI_State.transform(data[['State']])
print(data.isna().sum())

print(data.corr()['Profit'])

from sklearn.preprocessing import LabelEncoder
LB=LabelEncoder()
data['State']=LB.fit_transform(data['State'])

X=data.iloc[:,[0,2]].values
Y=data.iloc[:,-1].values

from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X=sc.fit_transform(X)


from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=.2,random_state=2)

print("PLOY,LR,KNN,SVR,DTR")

from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=2)
x_trans=poly.fit_transform(X_train)

from sklearn.linear_model import LinearRegression
Lr=LinearRegression()
Lr.fit(x_trans,Y_train)
x_trans_test=poly.fit_transform(X_test)
y1_pred=Lr.predict(x_trans_test)
```

```
from sklearn.metrics import r2_score
score1=r2_score(Y_test,y1_pred)
print(score1*100,'%')

from sklearn.neighbors import KNeighborsRegressor
KNN=KNeighborsRegressor(n_neighbors=3)
KNN.fit(X_train,Y_train)
y2_pred=KNN.predict(X_test)
from sklearn.metrics import r2_score
score2=r2_score(Y_test,y2_pred)
print(score2*100,'%')

from sklearn.svm import SVR
SVR=SVR(kernel='rbf',degree=2)
SVR.fit(X_train,Y_train)
y3_pred=SVR.predict(X_test)
from sklearn.metrics import r2_score
score3=r2_score(Y_test,y3_pred)
print(score3*100,'%')

from sklearn.tree import DecisionTreeRegressor
DTR=DecisionTreeRegressor(random_state=0,max_depth=4)
DTR.fit(X_train,Y_train)
y4_pred=DTR.predict(X_test)
from sklearn.metrics import r2_score
score4=r2_score(Y_test,y4_pred)
print(score4*100,'%')
```

10. An in-depth analysis of an organization's ideal customers is referred to as a Customer Personality Analysis. Businesses can use it to better understand their customers and modify products according to their preferences, behavior, and concerns. Suppose you have to do an analysis that should help a business to modify its product based on its target customers from different types of customer segments. For example, instead of spending money to market a new product to every customer in the company's database, a company can analyze which customer segment is most likely to buy the product and then market the product only on that particular segment. Explain in detail your approach how you would solve this problem for the company - beginning from the data collection to model selection to deployment.

Ans. At first, we have to collect the data sets, and then check it contains relevant information like, customer AGE, NUMBER OF TIME VISIT OR ENQUIRY FOR THE PRODUCT, INVESTMENT, SALARIED OR BUSSINESS CUSTOMER, CUSTOMER ID, YEAR OF BIRTH, EDUCATION, MARITAL STATUS, DIFFERENT CATEGORIES OF PRODUCT THEY PIRCHASED PREVIOUSLY.

There are many sources of pulling the data, we can pull from internet, directly ask from the particular company about the data.

After that, we can see the data on excel file or whatever file we get, then check the data manually without using any algorithms, like which columns is required and which is independent and which is dependent, or which column is dependent upon which column.

STEP 1: WE HAVE TO IMPORT OR LOAD THE DATA SETS INTO PYTHON BY IMPORTING REQUIRED LIB

```
Import numpy as np
Import pandas as pd
```

Then, after running the codes, we will check the data sets, is it uploaded properly or not.

The, we will check for the missing values insides the datasets.

Then, if there are any missing value, then we will fill those values by using python functions.

Then, if there are no any missing values, then we will check the correlation between the columns with required column.

```
import pandas as pd
data=pd.read_csv("")
print(data.isna().sum())
print(data.head())
X_corr=data.corr()
print(X_corr)
```

if corr., between the table is nearer to 1, either it is in + or -, then we will pick that by using tuple slicing.

```
x=data.iloc[:,[2,5,7,8,9,10,11,12]].values
y=data.iloc[:,-1].values
y=y.reshape(y.shape[0],1)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc=sc.fit(x)
x=sc.transform(x) # we use this to make the data In range.
 from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=.2,random_state=23) # we will go for
training and testing of our model.
```

After that, we will select our model, which we will select.

If we want to select only two columns, then we will go for linear model and if we want to find the relationship between more than one column, then we will go for classification model.

```
print("PLOY,LR,KNN,SVR,DTR")

from sklearn.preprocessing import PolynomialFeatures
poly=PolynomialFeatures(degree=2)
x_trans=poly.fit_transform(X_train)

from sklearn.linear_model import LinearRegression
Lr=LinearRegression()
Lr.fit(x_trans,Y_train)
x_trans_test=poly.fit_transform(X_test)
y1_pred=Lr.predict(x_trans_test)
from sklearn.metrics import r2_score
score1=r2_score(Y_test,y1_pred)
print(score1*100,'%')

from sklearn.neighbors import KNeighborsRegressor
KNN=KNeighborsRegressor(n_neighbors=3)
KNN.fit(X_train,Y_train)
y2_pred=KNN.predict(X_test)
from sklearn.metrics import r2_score
score2=r2_score(Y_test,y2_pred)
print(score2*100,'%')

from sklearn.svm import SVR
SVR=SVR(kernel='rbf',degree=2)
SVR.fit(X_train,Y_train)
y3_pred=SVR.predict(X_test)
from sklearn.metrics import r2_score
score3=r2_score(Y_test,y3_pred)
print(score3*100,'%')

from sklearn.tree import DecisionTreeRegressor
DTR=DecisionTreeRegressor(random_state=0,max_depth=4)
DTR.fit(X_train,Y_train)
```

```
y4_pred=DTR.predict(X_test)
from sklearn.metrics import r2_score
score4=r2_score(Y_test,y4_pred)
print(score4*100,'%')
```