MAJOR PROJECT 1

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

```
             Id  MovieId  Rating   Timestamp
     0        1        1     4.0   964982703
     1        1        3     4.0   964981247
     2        1        6     4.0   964982224
     3        1       47     5.0   964983815
     4        1       50     5.0   964982931
   ...      ...      ...     ...         ...
100831      610   166534     4.0  1493848402
100832      610   168248     5.0  1493850091
100833      610   168250     5.0  1494273047
100834      610   168252     5.0  1493846352
100835      610   170875     3.0  1493846415

100836 rows × 4 columns
```

```
df.shape
```

```
(100836, 4)
```

```
df.size
```

```
403344
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   Id         100836 non-null  int64  
 1   MovieId    100836 non-null  int64  
 2   Rating     100836 non-null  float64
 3   Timestamp  100836 non-null  int64  
```

```
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

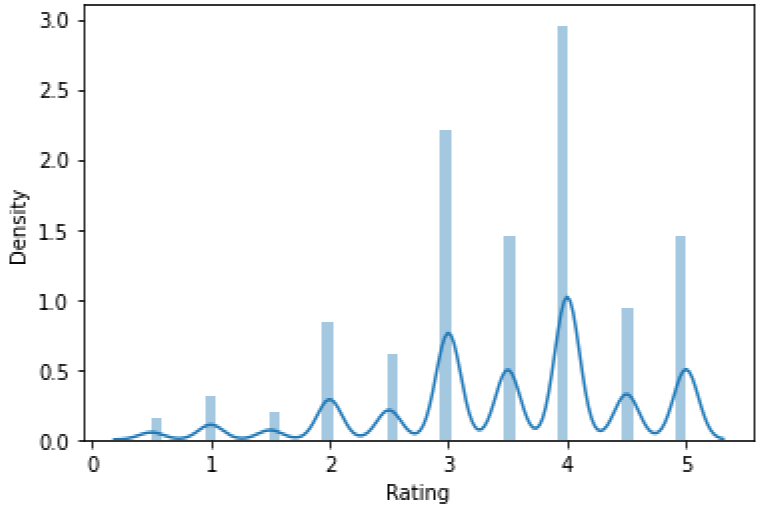
```
#2. PREPROCESSING - FILTERING OF DATA
#to remove/drop the Id column
df = df.drop(columns = 'Id')
df
```

	MovieId	Rating	Timestamp
0	1	4.0	964982703
1	3	4.0	964981247
2	6	4.0	964982224
3	47	5.0	964983815
4	50	5.0	964982931
...	...	...	...
100831	166534	4.0	1493848402
100832	168248	5.0	1493850091
100833	168250	5.0	1494273047
100834	168252	5.0	1493846352
100835	170875	3.0	1493846415

100836 rows × 3 columns

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

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `disp
  warnings.warn(msg, FutureWarning)
<matplotlib.axes._subplots.AxesSubplot at 0x7f603629ef50>
```



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

x = df.iloc[:,0:3].values
x
```

```
array([[1.00000000e+00, 4.00000000e+00, 9.64982703e+08],
       [3.00000000e+00, 4.00000000e+00, 9.64981247e+08],
       [6.00000000e+00, 4.00000000e+00, 9.64982224e+08],
       ...,
       [1.68250000e+05, 5.00000000e+00, 1.49427305e+09],
       [1.68252000e+05, 5.00000000e+00, 1.49384635e+09],
       [1.70875000e+05, 3.00000000e+00, 1.49384642e+09]])
```

```
y = df.iloc[:,2].values
y
```

```
array([ 964982703,  964981247,  964982224, ..., 1494273047, 1493846352,
       1493846415])
```

```
#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 variables,we want those allocated values to remain
#constant.By default the training variables get 75 % and
#testing variables get 25%
print("x.shape",x.shape) # 100836 rows and 3 cols
print("x_train.shape",x_train.shape) # 75627 rows and 3 cols (75%)
print("x_test.shape",x_test.shape) # 25209 rows and 3 cols (25%)
```

```
x.shape (100836, 3)
x_train.shape (75627, 3)
x_test.shape (25209, 3)
```

```
print("y.shape",y.shape) # 100836 rows and 1 col
print("y_train.shape",y_train.shape) # 75627 rows and 1 col (75%)
print("y_test.shape",y_test.shape) # 25209 rows and 1 col (25%)
```

```
y.shape (100836,)
y_train.shape (75627,)
y_test.shape (25209,)
```

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

```
array([8.58356752e+08, 1.51938155e+09, 1.21597702e+09, ...,
       9.74969489e+08, 1.39185447e+09, 1.46750650e+09])
```

```
y_test #ACTUAL VALUES
```

```
array([ 858350384, 1519235950, 1215895327, ...,  974938560, 1391735730,
       1467371826])
```

```
print(x_train[25]) #these are scaled/normalised values
```

```
[0.01997335 0.66666667 0.86682795]
```

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

```
array([1.23708294e+09])
```