AI and ML Project Laboratory Report

A report submitted in partial fulfilment of requirements for the award of the degree of

MASTER OF COMPUTER APPLICATION

BY

SOWMYA R

1DS22MC100

Under the Guidance of

Dr . Chandrika M

Assistant Professor,

Dept of MCA, DSCE

DEPARTMENT OF MASTER OF COMPUTER APPLICATIONS
DAYANANDA SAGAR COLLEGE OF ENGINEERING SHAVIGE
MALLESHWARA HILLS, KUMARASWAMY LAYOUT,
BANGALORE-560078

October 2023

DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institute to VTU, Belagavi, Approved by AICTE & ISO 9001-2008 Certified)

Accredited by National Assessment & Accreditation Council (NAAC) with 'A' grade, Shavige Malleshwara hills, Kumaraswamy layout, Bangalore-560078

CERTIFICATE

This is to certify that the Laboratory Report entitled **" AI and ML Project Laboratory "** carried out by Sowmya R (1DS22MC100) in partial fulfilment for the award of Degree of **Master of Computer Applications** of Dayananda Sagar College of Engineering, An Autonomous Institute affiliated to VTU, Belagavi during the year **2022-2023**, It is certified that all suggestions or correction indicated for internal assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements with respect to the work prescribed for the Master of Computer Applications.

Signature of Internal Guide:

Dr. Chandrika M
Assistant Professor,
Dept of MCA, DSCE, Bangalore.

Signature of HOD :

Dr. Samitha Khaiyum
Prof & HOD,
Dept. of MCA, DSCE, Bangalore.

Examiners:

Name of the Examiner	**Signature with Date**
1. …….	1. …….
2.…….	2. …….

Table of Contents

1. DATA PREPROCESSING

PART A

Data Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So for this, we use data preprocessing task.

Data Preprocessing involves the following steps:

1. Creating the dataset
2. Importing libraries
3. Importing datasets
4. Finding Missing Data
5. Encoding Categorical Data

Step 1 : Importing necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Step 2 : Creating the dataset

In [2]:
```
#Create dataset from pandas Series
pop = pd.Series({'Karnataka':61095297, 'Tamil Nadu':72147030, 'Andhra Pradesh':None
area = pd.Series({'Karnataka':191791, 'Tamil Nadu':130060, 'Andhra Pradesh':162968,
states = pd.DataFrame({'Population':pop,'Area':area})
states
```

Out[2]:

	Population	Area
Karnataka	61095297.0	191791.0
Tamil Nadu	72147030.0	130060.0
Andhra Pradesh	NaN	162968.0
Kerala	33406061.0	NaN

Understanding the data

In [3]:
```
states.columns
```

Out[3]:
```
Index(['Population', 'Area'], dtype='object')
```

In [4]:
```
states.index
```

Out[4]:
```
Index(['Karnataka', 'Tamil Nadu', 'Andhra Pradesh', 'Kerala'], dtype='object')
```

In [5]:
```
states.count()
```

Out[5]:
```
Population    3
Area          3
dtype: int64
```

In [6]:
```
states.sum()
```

Out[6]:
```
Population    166648388.0
Area             484819.0
dtype: float64
```

In [7]:
```
states.mean()
```

Out[7]:
```
Population    5.554946e+07
Area          1.616063e+05
dtype: float64
```

In [8]:
```
states.median()
```

Out[8]:
```
Population    61095297.0
Area            162968.0
dtype: float64
```

In [9]:
```
states.mode()
```

Out[9]:

	Population	Area
0	33406061.0	130060.0
1	61095297.0	162968.0
2	72147030.0	191791.0

In [10]:
```
states.std()
```

Out[10]:
```
Population    1.995703e+07
Area          3.088802e+04
dtype: float64
```

In [11]:
```
states.min()
```

Out[11]:
```
Population    33406061.0
Area            130060.0
dtype: float64
```

In [12]:
```
states.max()
```

Out[12]:
```
Population    72147030.0
Area            191791.0
dtype: float64
```

In [13]:
```
states.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, Karnataka to Kerala
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Population  3 non-null      float64
 1   Area        3 non-null      float64
dtypes: float64(2)
memory usage: 96.0+ bytes
```

In [14]:
```
states.describe()
```

Out[14]:

	Population	Area
count	3.000000e+00	3.000000
mean	5.554946e+07	161606.333333
std	1.995703e+07	30888.018589
min	3.340606e+07	130060.000000
25%	4.725068e+07	146514.000000
50%	6.109530e+07	162968.000000
75%	6.662116e+07	177379.500000
max	7.214703e+07	191791.000000

Step 4 : Handling Null values

Missing data are not rare in real data sets. The chance that at least one data point is missing increases as the data set size increases. Missing data can occur any number of ways, some of which include the following.
Merging of source data sets
Random events
Failures of measurement

In [15]:
```
#Handling Missing Values
states.isnull()
```

Out[15]:

	Population	Area
Karnataka	False	False
Tamil Nadu	False	False
Andhra Pradesh	True	False
Kerala	False	True

In [16]:
```
#Columns having missing values
missing_columns = [col for col in states.columns if states[col].isnull().sum() > 0]
missing_columns
```

Out[16]:
```
['Population', 'Area']
```

In [17]:
```
states.isnull().any()
```

Out[17]:
```
Population    True
Area          True
dtype: bool
```

In [18]:
```
states.isnull().sum()
```

Out[18]:
```
Population    1
Area          1
dtype: int64
```

Methods to handle missing data

As we Know if our data has missing values than our model will not train except few models which can tolerate them like some tree based models but the point is we want to handle this and how can we handle them. Following are some of the techniques :-
Deletion of Data
Encoding Missingness
Imputation Methods

Deletion of Data

The simplest approach for dealing with missing values is to remove entire attribute or sample that contain missing values. However, one must carefully consider a number of aspects of the data prior to taking this approach.

In [19]:
```
states.dropna()
```

Out[19]:

	Population	Area
Karnataka	61095297.0	191791.0
Tamil Nadu	72147030.0	130060.0

Filling Missing Data

In [20]:
```
states.fillna(0)
```

Out[20]:

	Population	Area
Karnataka	61095297.0	191791.0
Tamil Nadu	72147030.0	130060.0
Andhra Pradesh	0.0	162968.0
Kerala	33406061.0	0.0

In [21]:
```
states.fillna(method="ffill")
```

Out[21]:

	Population	Area
Karnataka	61095297.0	191791.0
Tamil Nadu	72147030.0	130060.0
Andhra Pradesh	72147030.0	162968.0
Kerala	33406061.0	162968.0

In [22]:
```
states.fillna(method="bfill")
```

Out[22]:

	Population	Area
Karnataka	61095297.0	191791.0
Tamil Nadu	72147030.0	130060.0
Andhra Pradesh	33406061.0	162968.0
Kerala	33406061.0	NaN

In [23]:
```
mean_pop=states['Population'].mean()
mean_area=states['Area'].mean()
states['Population'].fillna(value=mean_pop, inplace=True)
states['Area'].fillna(value=mean_area, inplace=True)
```

In [24]:
```
print(states)
```

```
                Population           Area
Karnataka       6.109530e+07  191791.000000
Tamil Nadu      7.214703e+07  130060.000000
Andhra Pradesh  5.554946e+07  162968.000000
Kerala          3.340606e+07  161606.333333
```

In [25]:
```
# Save the data into a csv file
states.to_csv('states.csv')
```

In [26]:
```
df=pd.read_csv('states.csv')
df
```

Out[26]:

	Unnamed: 0	Population	Area
0	Karnataka	6.109530e+07	191791.000000
1	Tamil Nadu	7.214703e+07	130060.000000
2	Andhra Pradesh	5.554946e+07	162968.000000
3	Kerala	3.340606e+07	161606.333333

In [ ]:
```

```

2. LINEAR REGRESSION

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
year=pd.Series((1.1,1.3,1.5,2.0,2.2,2.9,3.0,3.2,3.2,3.7,3.9,4.0,4.0,4.1,4.5,4.9,5.1
sal=pd.Series((39343,46205,37731,43525,39891,56642,60150,54445,64445,5718,63218,557
salary=pd.DataFrame({'yearexperience':year,'salary':sal})
print(salary)
```

```
    yearexperience  salary
0              1.1   39343
1              1.3   46205
2              1.5   37731
3              2.0   43525
4              2.2   39891
5              2.9   56642
6              3.0   60150
7              3.2   54445
8              3.2   64445
9              3.7    5718
10             3.9   63218
11             4.0   55794
12             4.0   56957
13             4.1   57081
14             4.5   61111
15             4.9   67938
16             5.1   66029
17             5.3   83088
18             5.9   81363
19             6.0   93940
20             6.8   91738
21             7.1   98273
22             7.9  101302
23             8.2  113812
24             8.7  109431
25             9.0  105582
26             9.5  116969
27             9.6  112635
28            10.3  122391
29            10.5  121872
```

In [3]:
```
salary.to_csv('salary.csv',index=False)
```

In [4]:
```
data=pd.read_csv('salary.csv')
data
```

Out[4]:

	yearexperience	salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891
5	2.9	56642
6	3.0	60150
7	3.2	54445
8	3.2	64445
9	3.7	5718
10	3.9	63218
11	4.0	55794
12	4.0	56957
13	4.1	57081
14	4.5	61111
15	4.9	67938
16	5.1	66029
17	5.3	83088
18	5.9	81363
19	6.0	93940
20	6.8	91738
21	7.1	98273
22	7.9	101302
23	8.2	113812
24	8.7	109431
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

In [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   yearexperience  30 non-null     float64
 1   salary          30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
```

In [6]:
```
data.head()
```

Out[6]:

	yearexperience	salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

In [7]:
```
data.tail()
```

Out[7]:

	yearexperience	salary
25	9.0	105582
26	9.5	116969
27	9.6	112635
28	10.3	122391
29	10.5	121872

In [8]:
```
data.shape
```

Out[8]:
```
(30, 2)
```

In [9]:
```
data.describe()
```

Out[9]:

	yearexperience	salary
count	30.000000	30.000000
mean	5.313333	74287.300000
std	2.837888	30110.534121
min	1.100000	5718.000000
25%	3.200000	56006.000000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

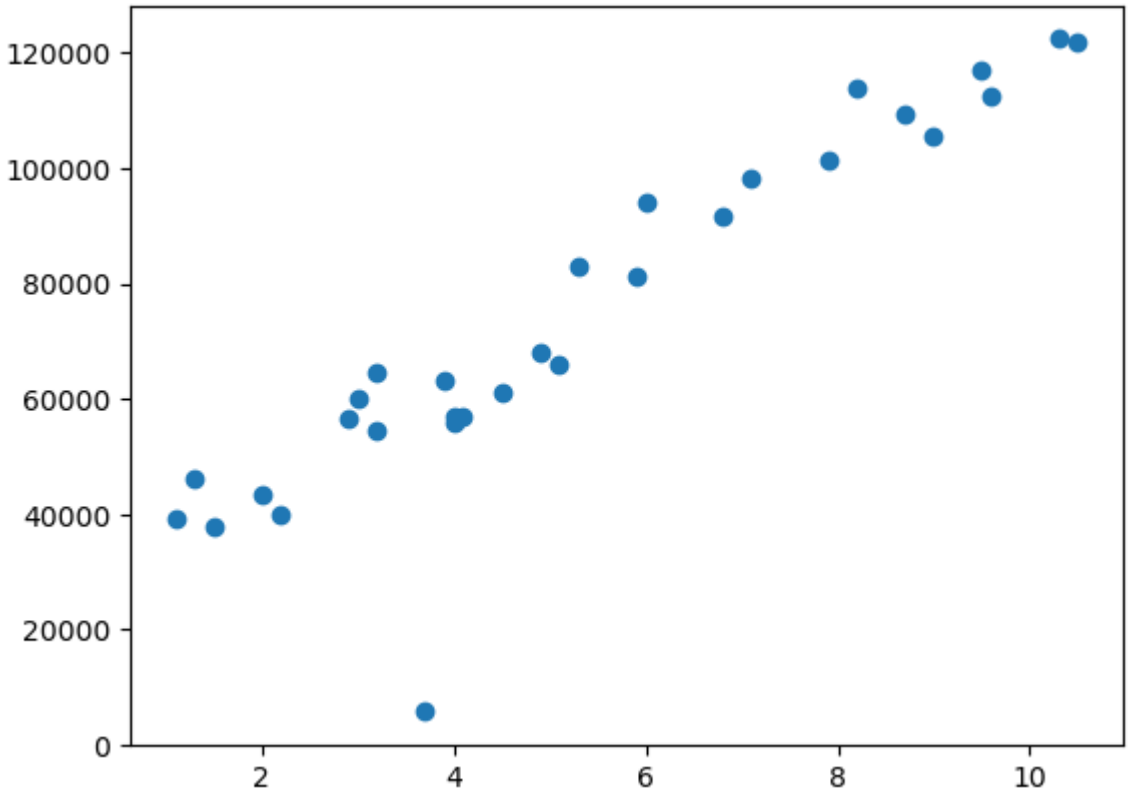
In [10]:
```
#data pre-processing
data.isnull().any()
```

Out[10]:
```
yearexperience    False
salary            False
dtype: bool
```

In [11]:
```
#data visuaization
import matplotlib.pyplot as plt
```

In [12]:
```
plt.scatter(data.yearexperience, data.salary)
```

Out[12]:
```
<matplotlib.collections.PathCollection at 0x1e1699d8b90>
```



In [13]:
```
x=data.iloc[:,:1]
x
```

Out[13]:

	yearexperience
0	1.1
1	1.3
2	1.5
3	2.0
4	2.2
5	2.9
6	3.0
7	3.2
8	3.2
9	3.7
10	3.9
11	4.0
12	4.0
13	4.1
14	4.5
15	4.9
16	5.1
17	5.3
18	5.9
19	6.0
20	6.8
21	7.1
22	7.9
23	8.2
24	8.7
25	9.0
26	9.5
27	9.6
28	10.3
29	10.5

In [14]:
```
y=data.iloc[:,1:2]
y
```

Out[14]:

	salary
0	39343
1	46205
2	37731
3	43525
4	39891
5	56642
6	60150
7	54445
8	64445
9	5718
10	63218
11	55794
12	56957
13	57081
14	61111
15	67938
16	66029
17	83088
18	81363
19	93940
20	91738
21	98273
22	101302
23	113812
24	109431
25	105582
26	116969
27	112635
28	122391
29	121872

In [15]:
```
type(x)
```

Out[15]:
```
pandas.core.frame.DataFrame
```

In [16]:
```
type(y)
```

Out[16]:
```
pandas.core.frame.DataFrame
```

In [17]:
```
np.shape(x)
```

Out[17]:
```
(30, 1)
```

In [18]:
```
np.shape(y)
```

Out[18]:
```
(30, 1)
```

In [19]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size = 0.2 , random_state=
```

In [21]:
```
print(x_train.shape)
```

```
(24, 1)
```

In [22]:
```
print(y_train.shape)
```

```
(24, 1)
```

In [23]:
```
print(x_test.shape)
```

```
(6, 1)
```

In [24]:
```
print(y_test.shape)
```

```
(6, 1)
```

In [25]:
```
from sklearn.linear_model import LinearRegression
```

In [26]:
```
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[26]:
```
▾ LinearRegression
LinearRegression()
```

In [27]:
```
y_pred=lr.predict(x_test)
y_pred
```

Out[27]:
```
array([[ 37050.61048599],
       [122843.74239885],
       [ 62398.58127843],
       [ 60448.73737132],
       [115044.36677041],
       [107244.99114197]])
```

In [28]:
```
y_test
```

Out[28]:

	salary
2	37731
28	122391
13	57081
10	63218
26	116969
24	109431

In [29]:
```
from sklearn.metrics import r2_score
```

In [30]:
```
acc=r2_score(y_pred ,y_test)
acc
```

Out[30]:
```
0.9927710342518399
```

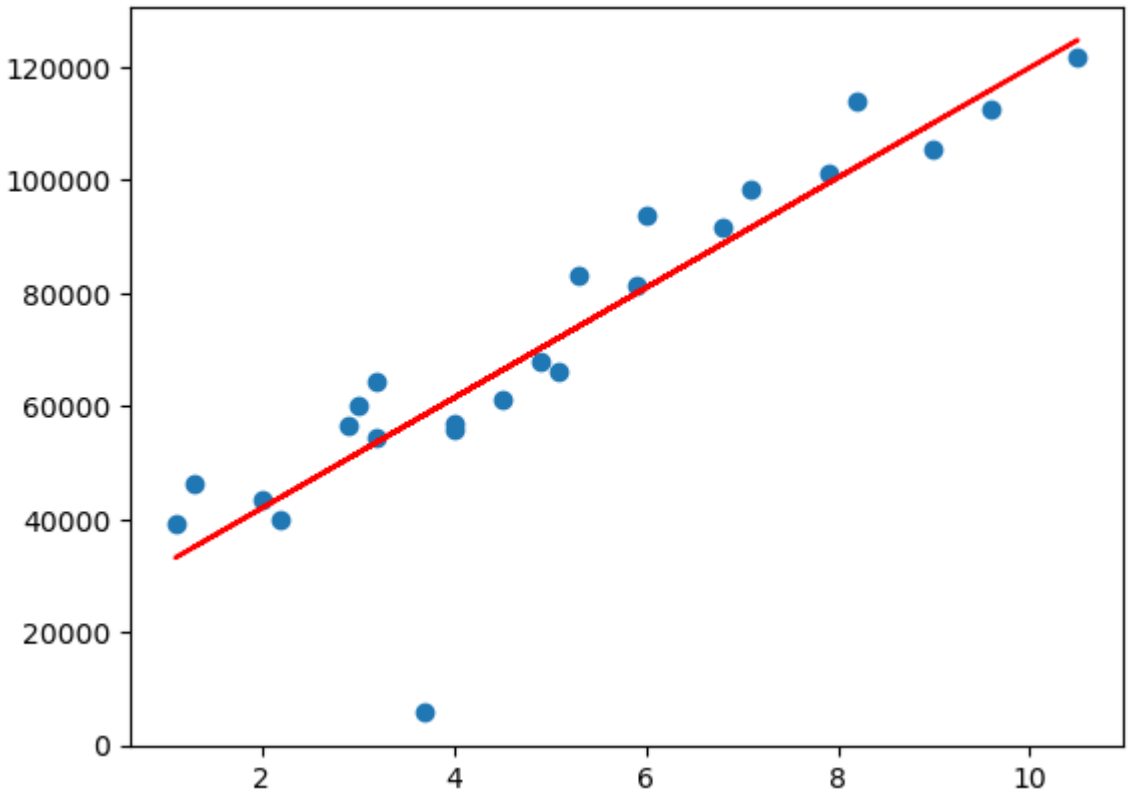
In [31]:
```
sal = lr.predict([[15]])
sal
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does n
ot have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Out[31]:
```
array([[168665.07421594]])
```

In [32]:
```
plt.scatter(x_train,y_train)
plt.plot(x_train, lr.predict(x_train),'r')
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x1e16add38d0>]
```



3. MULTI-LINEAR REGRESSION

Multi Linear Regression (MLR)

Multiple linear regression (MLR) is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. It is an extension of linear regression that uses just one explanatory variable. Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable.

In [1]:
```
#required libraries
import numpy as np
import pandas as pd
```

In [2]:
```
#load the dataset into pandas dataframe
data=pd.read_csv("C:\\Users\\soumy\\Downloads\\Startups.csv")
data.head()
```

Out[2]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108 entries, 0 to 107
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   R&D Spend        108 non-null    float64
 1   Administration   108 non-null    float64
 2   Marketing Spend  108 non-null    float64
 3   State            108 non-null    object 
 4   Profit           108 non-null    float64
dtypes: float64(4), object(1)
memory usage: 4.3+ KB
```

Data Preprocessing

In [4]:
```
data.isnull().any()
```

Out[4]:
```
R&D Spend          False
Administration     False
Marketing Spend    False
State              False
Profit             False
dtype: bool
```

LabelEncoder- Converts text value into numeric value by following the alphabetical order.

In [5]:
```
#Get the unique entries of states column
data['State'].unique()
```

Out[5]:
```
array(['New York', 'California', 'Florida'], dtype=object)
```

In [6]:
```
#label encoder assigns values in the alphabetical order
from sklearn.preprocessing import LabelEncoder
```

In [7]:
```
le=LabelEncoder()
data['State']=le.fit_transform(data['State'])
data
```

Out[7]:

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	2	192261.83
1	162597.70	151377.59	443898.53	0	191792.06
2	153441.51	101145.55	407934.54	1	191050.39
3	144372.41	118671.85	383199.62	2	182901.99
4	142107.34	91391.77	366168.42	1	166187.94
...	...	...	...	...	...
103	119943.24	156547.42	256512.92	1	132602.65
104	114523.61	122616.84	261776.23	2	129917.04
105	78013.11	121597.55	264346.06	0	126992.93
106	94657.16	145077.58	282574.31	2	125370.37
107	91749.16	114175.79	294919.57	1	124266.90

108 rows × 5 columns

We now can see the numeric value in the State column with values:
California -> 0
Florida -> 1
Newyork -> 2

In [8]:
```
x=data.iloc[:,0:4].values
y=data.iloc[:,4:5].values
```

In [9]:
```
x.shape
```

Out[9]:
```
(108, 4)
```

In [10]:
```
y.shape
```

Out[10]:
```
(108, 1)
```

In [11]:
```
#applying onehotEncoder()
from sklearn.preprocessing import OneHotEncoder
```

In [12]:
```
one=OneHotEncoder()
z=one.fit_transform(x[:,3:4]).toarray()
```

In [13]:
```
z.shape
```

Out[13]:
```
(108, 3)
```

In [14]:
```
z
```

Out[14]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
```

```
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

In [15]:
```
x=np.delete(x,3,axis=1)
```

In [16]:
```
x=np.concatenate((z,x),axis=1)
```

Split this data into training and testing data set

In [17]:
```
from sklearn.model_selection import train_test_split
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_stat
```

In [19]:
```
x_train.shape
```

Out[19]: (86, 6)

In [20]:
```
x_test.shape
```

Out[20]: (22, 6)

In [21]:
```
y_train.shape
```

Out[21]: (86, 1)

In [22]:
```
y_test.shape
```

Out[22]: (22, 1)

Build the model

In [23]:
```
from sklearn.linear_model import LinearRegression
```

In [24]:
```
mlr= LinearRegression()
mlr.fit(x_train, y_train)
```

Out[24]:
```
▾ LinearRegression
LinearRegression()
```

Test the model

In [25]:
```
y_pred= mlr.predict(x_test)
y_pred
```

Out[25]:
```
array([[ 48466.75433417],
       [134429.39628447],
       [ 76093.94745793],
       [181201.94746512],
       [113069.45110036],
       [134397.50337433],
       [128842.89457941],
       [159664.83856911],
       [116952.09718497],
       [ 46372.47771261],
       [102406.37641003],
       [115207.87076438],
       [ 48466.75433417],
       [119345.51846303],
       [ 88717.92542864],
       [127301.6808492 ],
       [127301.6808492 ],
       [ 91145.83032982],
       [ 58337.13774601],
       [146474.04747225],
       [149038.44713346],
       [152718.63782934]])
```

In [26]:
```
y_test
```

Out[26]:
```
array([[ 64926.08],
       [146121.95],
       [ 90708.19],
       [191050.39],
       [108552.04],
       [144259.4 ],
       [124266.9 ],
       [155752.6 ],
       [126992.93],
       [ 42559.73],
       [101004.64],
       [110352.25],
       [ 64926.08],
       [111313.02],
       [ 89949.14],
       [134307.35],
       [134307.35],
       [ 96712.8 ],
       [ 49490.75],
       [129917.04],
       [132602.65],
       [152211.77]])
```

Checking r2_score (the evaluation metric)

In [27]:
```
from sklearn.metrics import r2_score
```

In [28]:
```
acc=r2_score(y_test,y_pred)
acc
```

Out[28]: 0.9262629717101495

To predict the profit by giving some new value as input to the model

In [29]:
```
mlr.predict([[1,0,0,1234,1345,2345]])
```

Out[29]: array([[48734.87469882]])

In [ ]:
```

```

4. POLYNOMIAL REGRESSION

In [1]:
```
#Libraries
import numpy as np
import pandas as pd
```

In [2]:
```
data=pd.read_csv('Position_Salaries.csv')
data.head()
```

Out[2]:

	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

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   Position  10 non-null     object
 1   Level     10 non-null     int64 
 2   Salary    10 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 372.0+ bytes
```

In [4]:
```
data
```

Out[4]:

	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
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

In [5]:
```
x=data.iloc[:,1:2]
x
```

Out[5]:

	Level
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

In [6]:
```
y=data.iloc[:,2:3]
y
```

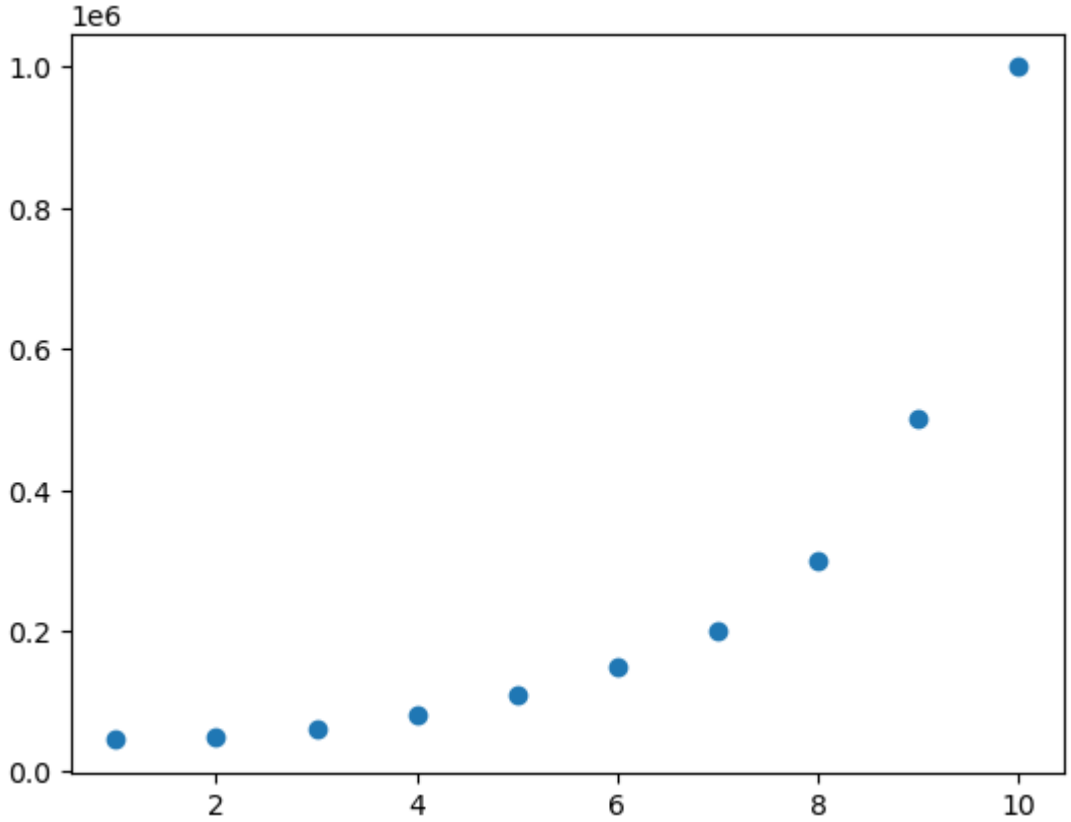
Out[6]:

	Salary
0	45000
1	50000
2	60000
3	80000
4	110000
5	150000
6	200000
7	300000
8	500000
9	1000000

In [7]:
```
import matplotlib.pyplot as plt
```

In [8]:
```
plt.scatter(x,y)
```

Out[8]:
```
<matplotlib.collections.PathCollection at 0x2cf3ecb8e50>
```



In [9]:
```
from sklearn.preprocessing import PolynomialFeatures
```

In [10]:
```
pf=PolynomialFeatures(degree=4)
xp=pf.fit_transform(x)
xp
```

Out[10]:
```
array([[1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 1.000e+00],
       [1.000e+00, 2.000e+00, 4.000e+00, 8.000e+00, 1.600e+01],
       [1.000e+00, 3.000e+00, 9.000e+00, 2.700e+01, 8.100e+01],
       [1.000e+00, 4.000e+00, 1.600e+01, 6.400e+01, 2.560e+02],
       [1.000e+00, 5.000e+00, 2.500e+01, 1.250e+02, 6.250e+02],
       [1.000e+00, 6.000e+00, 3.600e+01, 2.160e+02, 1.296e+03],
       [1.000e+00, 7.000e+00, 4.900e+01, 3.430e+02, 2.401e+03],
       [1.000e+00, 8.000e+00, 6.400e+01, 5.120e+02, 4.096e+03],
       [1.000e+00, 9.000e+00, 8.100e+01, 7.290e+02, 6.561e+03],
       [1.000e+00, 1.000e+01, 1.000e+02, 1.000e+03, 1.000e+04]])
```

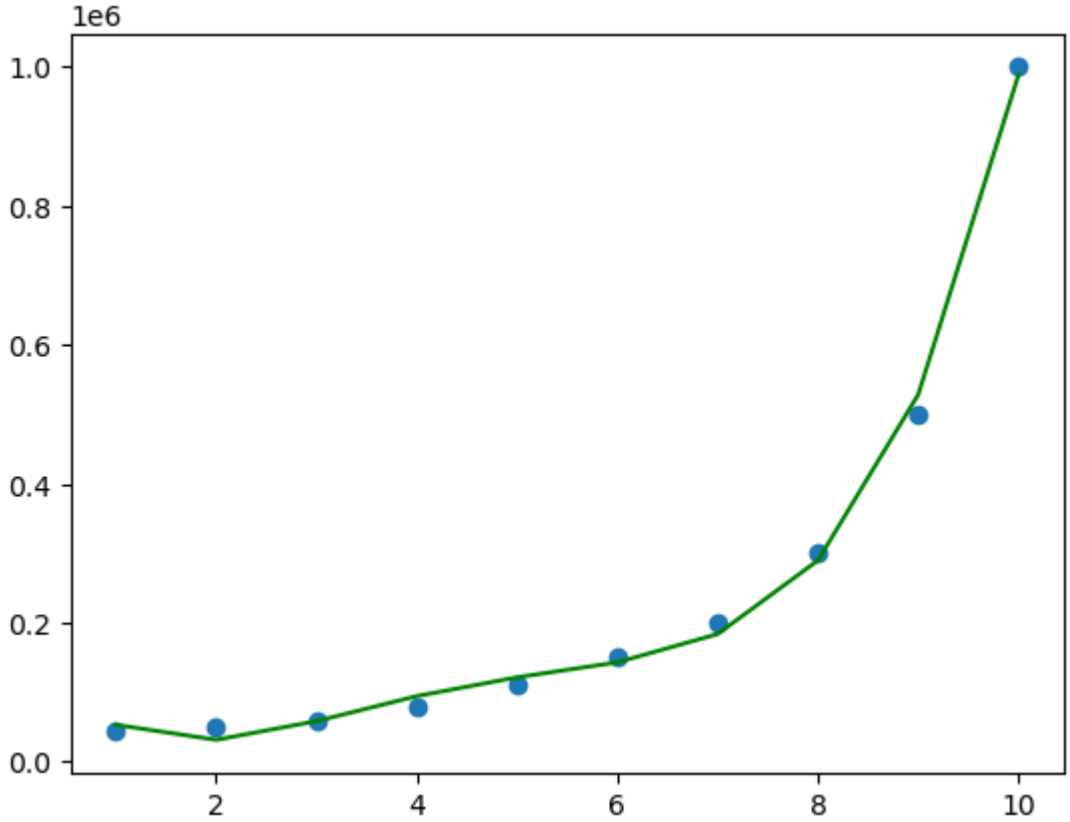
In [11]:
```
from sklearn.linear_model import LinearRegression
```

In [12]:
```
lr=LinearRegression()
lr.fit(xp,y)
```

Out[12]:
```
▾ LinearRegression
LinearRegression()
```

In [13]:
```
plt.scatter(x,y)
plt.plot(x,lr.predict(xp),'g')
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x2cf41b39b50>]
```


1e6
1.0
0.8
0.6
0.4
0.2
0.0
2
4
6
8
10

5. CLASSIFICATION

In [46]:
```
import numpy as np
import pandas as pd
```

In [47]:
```
df=pd.read_csv('loan_prediction.csv')
df.head()
```

Out[47]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	LP001002	Male	No	0.0	Graduate	No	5849	
1	LP001003	Male	Yes	1.0	Graduate	No	4583	
2	LP001005	Male	Yes	0.0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0.0	Not Graduate	No	2583	
4	LP001008	Male	No	0.0	Graduate	No	6000	

In [48]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    float64
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(5), int64(1), object(7)
memory usage: 62.5+ KB
```

In [49]:
```
df.describe()
```

Out[49]:

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credi
count	599.000000	614.000000	614.000000	592.000000	600.00000	56
mean	0.762938	5403.459283	1621.245798	146.412162	342.00000	
std	1.015216	6109.041673	2926.248369	85.587325	65.12041	
min	0.000000	150.000000	0.000000	9.000000	12.00000	
25%	0.000000	2877.500000	0.000000	100.000000	360.00000	
50%	0.000000	3812.500000	1188.500000	128.000000	360.00000	
75%	2.000000	5795.000000	2297.250000	168.000000	360.00000	
max	3.000000	81000.000000	41667.000000	700.000000	480.00000	

In [50]:
```
#data processing
df.isnull().any()
```

Out[50]:
```
Loan_ID              False
Gender                True
Married               True
Dependents            True
Education            False
Self_Employed         True
ApplicantIncome      False
CoapplicantIncome    False
LoanAmount            True
Loan_Amount_Term      True
Credit_History        True
Property_Area        False
Loan_Status          False
dtype: bool
```

In [51]:
```
df.isnull().sum()
```

Out[51]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

In [52]:
```
df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)
df['Married'].fillna(df['Married'].mode()[0], inplace=True)
df['Dependents'].fillna(df['Dependents'].mode()[0], inplace=True)
df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean(), inplace=True)
df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)
```

In [53]:
```
df.isnull().sum()
```

Out[53]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [54]:
```
#handling text data
from sklearn.preprocessing import LabelEncoder
```

In [55]:
```
le=LabelEncoder()
df['Gender']=le.fit_transform(df['Gender'].astype(str))
df['Married']=le.fit_transform(df['Married'].astype(str))
df['Education']=le.fit_transform(df['Education'].astype(str))
df['Self_Employed']=le.fit_transform(df['Self_Employed'].astype(str))
df['Property_Area']=le.fit_transform(df['Property_Area'].astype(str))
df['Loan_Status']=le.fit_transform(df['Loan_Status'].astype(str))
```

In [56]:
```
df.head()
```

Out[56]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Coapplic
0	LP001002	1	0	0.0	0	0	5849	
1	LP001003	1	1	1.0	0	0	4583	
2	LP001005	1	1	0.0	0	1	3000	
3	LP001006	1	1	0.0	1	0	2583	
4	LP001008	1	0	0.0	0	0	6000	

In [57]:
```
x=df.drop(columns=['Loan_ID','Loan_Status']).values
x
```

Out[57]:
```
array([[  1.,   0.,   0., ..., 360.,   1.,   2.],
       [  1.,   1.,   1., ..., 360.,   1.,   0.],
       [  1.,   1.,   0., ..., 360.,   1.,   2.],
       ...,
       [  1.,   1.,   1., ..., 360.,   1.,   2.],
       [  1.,   1.,   2., ..., 360.,   1.,   2.],
       [  0.,   0.,   0., ..., 360.,   0.,   1.]])
```

In [58]:
```
x.shape
```

Out[58]:
```
(614, 11)
```

In [59]:
```
y=df['Loan_Status'].values
y
```

Out[59]:
```
array([1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1,
       1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0])
```

In [60]:
```
y.shape
```

Out[60]:
```
(614,)
```

In [61]:
```
from sklearn.preprocessing import OneHotEncoder
```

In [62]:
```
one=OneHotEncoder()
z=one.fit_transform(x[:,10:11]).toarray()
z
```

Out[62]:
```
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 1., 0.]])
```

In [63]:
```
x=np.delete(x,10,axis=1)
```

In [64]:
```
x=np.concatenate((z,x),axis=1)
```

In [65]:
```
x.shape
```

Out[65]:
```
(614, 13)
```

In [66]:
```
from sklearn.model_selection import train_test_split
```

In [67]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [68]:
```
x_train.shape
```

Out[68]:
```
(491, 13)
```

In [69]:
```
x_test.shape
```

Out[69]:
```
(123, 13)
```

In [70]:
```
x
```

Out[70]:
```
array([[  0.        ,   0.        ,   1.        , ..., 146.41216216,
        360.        ,   1.        ],
       [  1.        ,   0.        ,   0.        , ..., 128.        ,
        360.        ,   1.        ],
       [  0.        ,   0.        ,   1.        , ...,  66.        ,
        360.        ,   1.        ],
       ...,
       [  0.        ,   0.        ,   1.        , ..., 253.        ,
        360.        ,   1.        ],
       [  0.        ,   0.        ,   1.        , ..., 187.        ,
        360.        ,   1.        ],
       [  0.        ,   1.        ,   0.        , ..., 133.        ,
        360.        ,   0.        ]])
```

In [71]:
```
from sklearn.preprocessing import StandardScaler
```

In [72]:
```
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.fit_transform(x_test)
```

In [73]:
```
from sklearn.linear_model import LogisticRegression
```

In [74]:
```
log=LogisticRegression()
log.fit(x_train,y_train)
```

Out[74]:
```
▾ LogisticRegression

LogisticRegression()
```

In [75]:
```
logprediction=log.predict(x_test)
logprediction
```

Out[75]:
```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1])
```

In [76]:
```
from sklearn.metrics import accuracy_score
```

In [77]:
```
logacc=accuracy_score(y_test,logprediction)
logacc
```

Out[77]:
```
0.8373983739837398
```

In [78]:
```
from sklearn.metrics import confusion_matrix
```

In [79]:
```
logcm=confusion_matrix(logprediction ,y_test)
logcm
```

Out[79]:
```
array([[15,  2],
       [18, 88]], dtype=int64)
```

In [80]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [81]:
```
dtc= DecisionTreeClassifier(criterion='entropy')
dtc.fit(x_train,y_train)
```

Out[81]:
```
DecisionTreeClassifier(criterion='entropy')
```

In [82]:
```
dtcprediction=dtc.predict(x_test)
dtcprediction
```

Out[82]:
```
array([1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0])
```

In [83]:
```
dtcacc=accuracy_score(dtcprediction ,y_test)
dtcacc
```

Out[83]:
```
0.6666666666666666
```

In [84]:
```
dtccm=confusion_matrix(dtcprediction,y_test)
dtccm
```

Out[84]:
```
array([[20, 28],
       [13, 62]], dtype=int64)
```

In [85]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [86]:
```
rfc=RandomForestClassifier(criterion='entropy')
rfc.fit(x_train,y_train)
```

Out[86]:
```
RandomForestClassifier(criterion='entropy')
```

In [87]:
```
rfc_predict=rfc.predict(x_test)
rfc_predict
```

Out[87]:
```
array([1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0])
```

In [88]:
```
y_test
```

Out[88]:
```
array([1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1])
```

In [89]:
```
rfcacc=accuracy_score(y_test,rfc_predict)
rfcacc
```

Out[89]:
```
0.7479674796747967
```

In [90]:
```
rfccm=confusion_matrix(y_test,rfc_predict)
rfccm
```

Out[90]:
```
array([[16, 17],
       [14, 76]], dtype=int64)
```

PART – B:

COMMENT COUNT ANALYSIS FOR

T-SERIES YOUTUBE DATASET

INTRODUCTION

OVERVIEW OF THE DATASET

On July 11, 1983, Gulshan Kumar established the Indian music record label and motion picture production firm T-Series. It is most recognized for creating Indi-pop songs and Hindi film soundtracks. With up to a 35% market share as of September 7, 2023, It is the biggest music record label in India.

T-Series also owns and runs the most popular YouTube channel, with over 248 million subscribers and 232 billion views overall as of September 7, 2023 12. TIt is most known as a music label, but it has also had considerable success as a production firm for movies. T-Series was initially created by Kumar, a fruit juice vendor in Delhi, to sell stolen Hindi songs before the business eventually started creating fresh music.

Their breakthrough came with the soundtrack to the Aamir Khan and Juhi Chawla-starring 1988 Bollywood blockbuster Qayamat Se Qayamat Tak, written by Majrooh Sultanpuri and composed by Anand-Milind. The album went on to become one of the best-selling Indian music albums of the 1980s, selling more than 8 million copies. With the release of Nadeem-Shravan's Aashiqui (1990), which sold 20 million copies and went on to become the best-selling Indian soundtrack album of all time, they eventually rose to prominence in the music industry.

The dataset used is the youtube t-series dataset with the following columns:

videoId: This column contains unique identifiers or codes for each video. These are alphanumeric strings that serve as identifiers for each video.

publishedTime: This column represents the date and time when each video was published. The date and time are in the format "DD-MM-YYYY HH:MM."

durationSec: This column indicates the duration of each video in seconds. It shows how long each video lasts.

viewCount: This column displays the number of views each video has received. It represents the total count of how many times each video has been watched.

likeCount: This column shows the number of likes (or thumbs up) that each video has received. It represents the count of users who have liked the video.

dislikeCount: This column indicates the number of dislikes (or thumbs down) that each video has received. It represents the count of users who have disliked the video.

commentCount: This column displays the number of comments that have been posted on each video. It represents the count of user comments associated with each video.

Each row in the dataset represents a data for a specific date and time related to the T-Series YouTube channel. The dataset appears to capture changes in the number of subscribers, the total video count, and the total view count over time. This data could be useful for analyzing trends and patterns in the growth of the T-Series channel on Youtube

TOOLS AND TECHNOLOGIES

Multiple tools and technologies are commonly used in multilinear regression analysis. These tools and technologies help researchers and data analysts perform regression analysis efficiently and effectively. Here tools and technologies used in this project on multilinear regression:

Python: Python is statistical tool and high level interpreted programming language that has libraries like NumPy, pandas, and scikit-learn that provide tools for data manipulation, linear regression modeling and statistical analysis.

Jupyter Notebooks: Jupyter notebooks are interactive documents that allows us to combine code, explanations, and visualizations in a single environment. They are often used for data exploration and regression analysis.

Data Visualization Tools: Matplotlib is a popular Python library for creating static, animated, and interactive visualizations.

Feature Engineering Tools: Tools like scikit-learn provide utilities for feature selection, scaling, and transformation, which are important in preparing data for regression analysis.

Data Preprocessing Tools: Data preprocessing libraries like pandas and scikit-learn help with data cleaning, handling missing values, and encoding categorical variables.

Reporting and Documentation Tools: Tools like Jupyter Notebook, is used for creating reports, and documentation of the analysis. These are the combination of these tools used to conduct a thorough and effective regression analysis

IMPLEMENTATION LOGIC

In this project to perform comment count analysis using multiple linear regression (MLR), we have used comment count as the dependent variable and other variables like view count, like count, dislike count, and duration as independent variables. MLR allows us to estimate the relationship between these independent variables and the comment count.

Steps used for MLR are as follows:

1. **Data Preprocessing**: Ensure that your dataset is clean, handle any missing or incorrect values, and format the data types appropriately. Here are the key details of data preprocessing in MLR:

   Handling Missing Data: Check for missing values in the dataset using functions like isnull() or info(). Decide on a strategy to handle missing values. Common approaches include removing rows with missing values, imputing missing values with the mean, median, or a custom value, or using advanced imputation techniques like regression imputation.

   Data Type Conversion: Ensure that the data types of each column are appropriate. For MLR, numerical data types are usually required for both the dependent and

independent variables. Convert date or time-related columns to appropriate date/time data types if needed.

Handling Categorical Variables: If your dataset contains categorical variables (e.g., video categories, video IDs), you'll need to encode them into numerical values. Common techniques include one-hot encoding or label encoding, depending on the nature of the categorical variable.

2. **Feature Selection**: Choose the independent variables (explanatory variables) that you believe may influence the comment count. In this case, view count, like count, dislike count, and duration are relevant.

   Correlation Analysis: Calculate the correlation coefficients between each independent variable (view count, like count, dislike count, duration) and the dependent variable (comment count). You can use methods like Pearson's correlation coefficient. High positive correlation values (close to 1) suggest a strong positive relationship, while high negative correlation values (close to -1) suggest a strong negative relationship. Select variables with reasonably strong correlations with comment count. Variables with higher absolute correlation values are often more influential

3. **Model Building**: Use a Python library like scikit-learn to build your MLR model. Fit the model using selected independent variables to predict the comment count.

   Split the dataset into training and testing sets to evaluate MLR model's performance.

   Create an instance of the LinearRegression model and fit it to the training data.

   Make Predictions: Use the trained MLR model to make predictions on the testing data.

4. **Model Evaluation:** Assess the model's performance by looking at metrics like R-squared to understand how well the independent variables explain the variation in the comment count.
5. **Prediction:** You can use the trained MLR model to make predictions for comment counts for new videos.

IPYTHON NOTEBOOK

Comment Count Analysis for T-Series YouTube Dataset

Multiple linear regression (MLR) is a statistical technique that uses several explanatory variables to predict the outcome of a response varia ble. It is an extension of linear regression that uses just one explanatory variable. Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable.

In [1]:
```
#required libraries
import numpy as np
import pandas as pd
```

In [2]:
```
#load the dataset into pandas dataframe
data=pd.read_csv("C:\\Users\\soumy\\OneDrive\\Desktop\\T-Series.csv")
data
```

Out[2]:

	videoId	publishedTime	durationSec	viewCount	likeCount	dislikeCount	com
0	O5X3bnTI1Go	17-05-2020 12:00	235	320360	13976.0	25966.0	
1	bTPt1voMr8g	17-05-2020 09:07	189	277845	33600.0	1645.0	
2	#NAME?	17-05-2020 10:39	376	62515	2334.0	373.0	
3	g4Xa7WwOFw0	17-05-2020 05:30	140	117328	6043.0	604.0	
4	tXIHXccXcCg	16-05-2020 08:31	20	996968	50636.0	2307.0	
...	...	...	...	...	...	...	...
831	G1D0n68lOS8	23-10-2019 08:30	243	135862	1874.0	128.0	
832	tDHQPHCLd6c	23-10-2019 09:30	278	45847	525.0	98.0	
833	B88D1SGN0Ig	23-10-2019 10:15	168	80220	898.0	136.0	
834	frTgD-MC5pI	25-10-2019 05:30	153	2943938	39888.0	2534.0	
835	UHS1nCK3Smc	22-10-2019 10:36	23	581443	15215.0	1074.0	

836 rows × 7 columns

In [3]:
```
data.head()
```

Out[3]:

	videoId	publishedTime	durationSec	viewCount	likeCount	dislikeCount	comm
0	O5X3bnTI1Go	17-05-2020 12:00	235	320360	13976.0	25966.0	
1	bTPt1voMr8g	17-05-2020 09:07	189	277845	33600.0	1645.0	
2	#NAME?	17-05-2020 10:39	376	62515	2334.0	373.0	
3	g4Xa7WwOFw0	17-05-2020 05:30	140	117328	6043.0	604.0	
4	tXIHXccXcCg	16-05-2020 08:31	20	996968	50636.0	2307.0	

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 836 entries, 0 to 835
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   videoId        836 non-null    object 
 1   publishedTime  836 non-null    object 
 2   durationSec    836 non-null    int64  
 3   viewCount      836 non-null    int64  
 4   likeCount      834 non-null    float64
 5   dislikeCount   834 non-null    float64
 6   commentCount   836 non-null    int64  
dtypes: float64(2), int64(3), object(2)
memory usage: 45.8+ KB
```

Data Preprocessing

In [5]:
```
data.isnull().any()
```

Out[5]:
```
videoId          False
publishedTime    False
durationSec      False
viewCount        False
likeCount         True
dislikeCount      True
commentCount     False
dtype: bool
```

In [6]:
```
data['likeCount'].fillna(data['likeCount'].mean(),inplace=True)
data['dislikeCount'].fillna(data['dislikeCount'].mean(),inplace=True)
```

In [7]:
```
data.isnull().any()
```

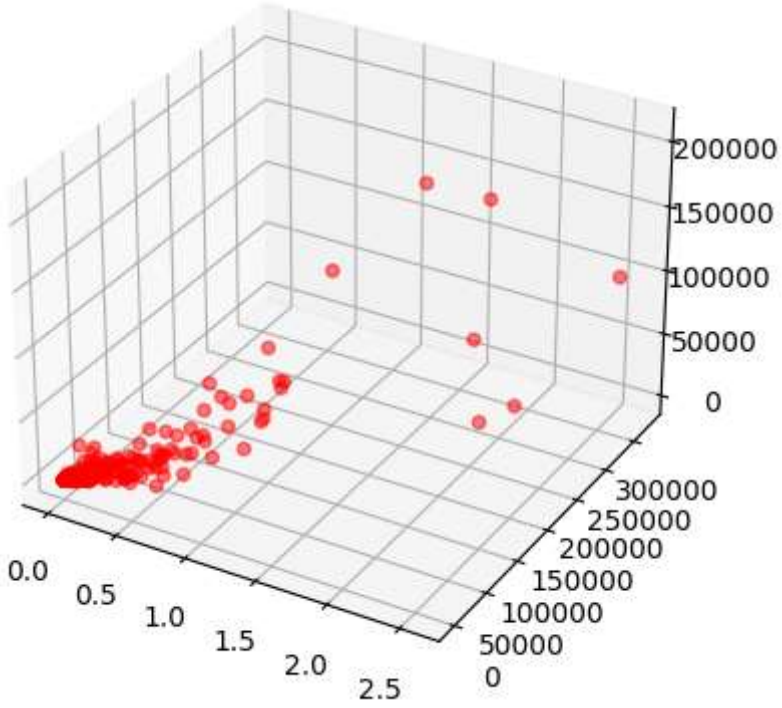
Out[7]:
```
videoId          False
publishedTime    False
durationSec      False
viewCount        False
likeCount        False
dislikeCount     False
commentCount     False
dtype: bool
```

Vizualizing the Data

In [8]:
```
import matplotlib.pyplot as plt
```

In [9]:
```
fig=plt.figure()
ax=fig.add_subplot(111,projection='3d')
ax.scatter(data['likeCount'],data['dislikeCount'],data['commentCount'],c='red',alph
```

Out[9]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1a874be0290>
```



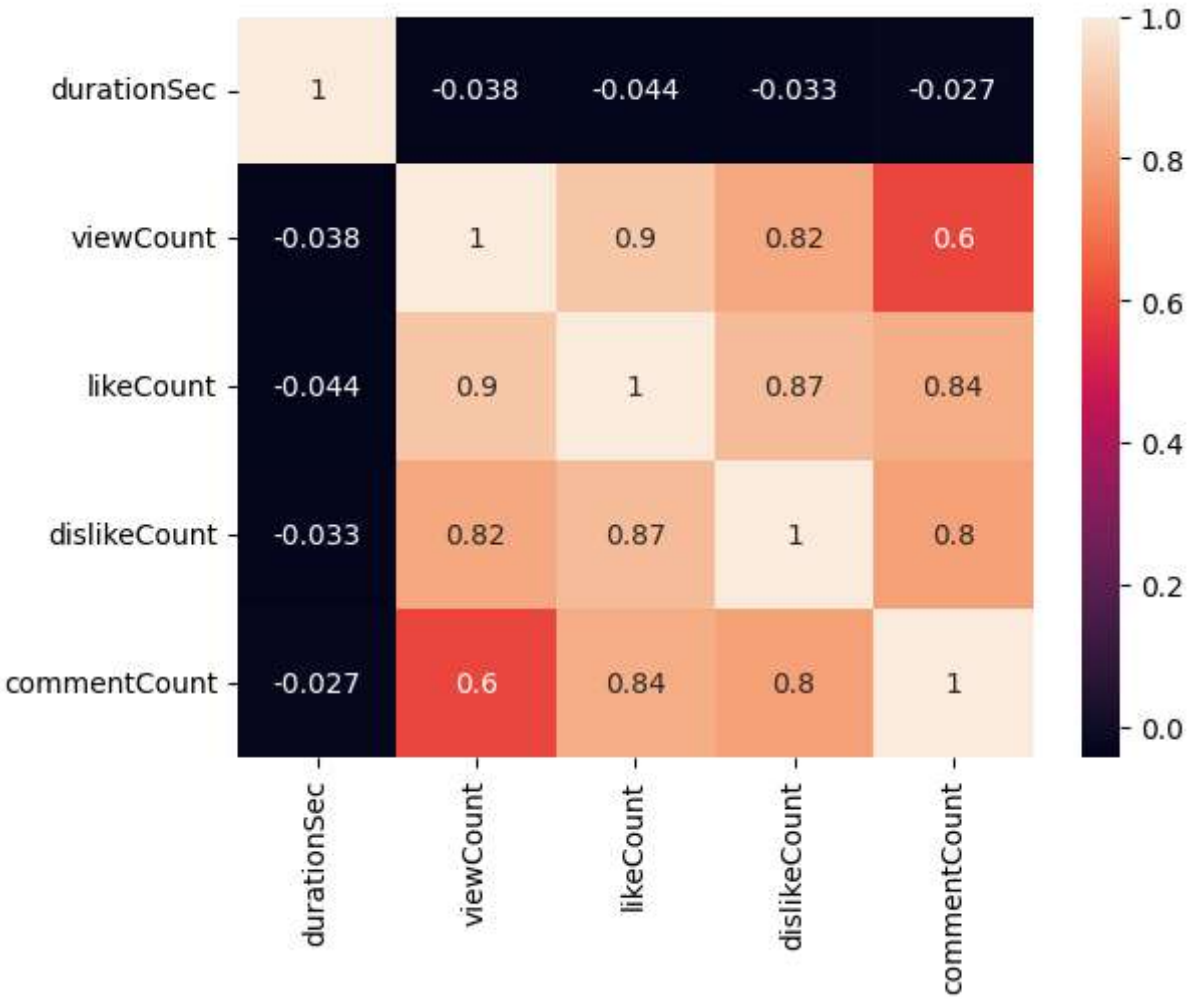
In [10]:
```
#correlation analysis
corr=data.iloc[:,2:7].corr()
corr
```

Out[10]:

	durationSec	viewCount	likeCount	dislikeCount	commentCount
durationSec	1.000000	-0.037876	-0.043883	-0.033498	-0.026743
viewCount	-0.037876	1.000000	0.900141	0.819763	0.598791
likeCount	-0.043883	0.900141	1.000000	0.872930	0.835181
dislikeCount	-0.033498	0.819763	0.872930	1.000000	0.802999
commentCount	-0.026743	0.598791	0.835181	0.802999	1.000000

In [11]:
```
import seaborn as sns
sns.heatmap(corr,annot=True)
```

Out[11]: <Axes: >



Depenedent and Independent Variables

In [12]:
```
x=data.iloc[:,2:6].values
x
```

Out[12]:
```
array([[2.350000e+02, 3.203600e+05, 1.397600e+04, 2.596600e+04],
       [1.890000e+02, 2.778450e+05, 3.360000e+04, 1.645000e+03],
       [3.760000e+02, 6.251500e+04, 2.334000e+03, 3.730000e+02],
       ...,
       [1.680000e+02, 8.022000e+04, 8.980000e+02, 1.360000e+02],
       [1.530000e+02, 2.943938e+06, 3.988800e+04, 2.534000e+03],
       [2.300000e+01, 5.814430e+05, 1.521500e+04, 1.074000e+03]])
```

In [13]:
```
y=data.iloc[:,6:].values
y
```

Out[13]:
```
array([[  3175],
       [  3313],
       [   291],
       [   685],
       [  2712],
       [   736],
       [   780],
       [  2689],
       [  1523],
       [   876],
       [   241],
       [   894],
       [   842],
       [   822],
       [   332],
       [  1203],
       [  1303],
       [  1267],
       [ 16037],
       [  1062],
       [   930],
       [  2580],
       [   406],
       [  1698],
       [  1396],
       [  1022],
       [  1040],
       [  2089],
       [   467],
       [  1336],
       [   942],
       [  1194],
       [  1038],
       [  1346],
       [   884],
       [   333],
       [  3558],
       [ 24355],
       [   272],
       [   659],
       [  1199],
       [  1397],
       [  1521],
       [  1207],
       [  4075],
       [  1247],
       [  2815],
       [  8024],
       [   462],
       [   501],
       [  1080],
       [  1007],
       [   119],
       [   361],
       [   624],
       [   809],
```

```
[   723],
[  8834],
[   797],
[   459],
[   969],
[   551],
[   522],
[   752],
[   993],
[  1694],
[  1953],
[   999],
[  1810],
[   649],
[   676],
[   738],
[   537],
[   414],
[  1323],
[   792],
[   778],
[   756],
[   467],
[  1256],
[  9728],
[   882],
[  1045],
[  1120],
[  1120],
[   939],
[   568],
[   647],
[   618],
[100808],
[  1105],
[  5389],
[  1490],
[   620],
[  1305],
[ 11105],
[   457],
[   822],
[   932],
[   668],
[  1253],
[  1024],
[102947],
[  1085],
[  1075],
[   582],
[  1235],
[   205],
[   762],
[  1386],
[  1173],
[  1454],
```

```
[  7948],
[  5761],
[   885],
[  2108],
[   944],
[ 26970],
[   331],
[  2073],
[   936],
[  3470],
[  1093],
[  1032],
[   997],
[  1229],
[  1351],
[   266],
[   775],
[   553],
[   869],
[   938],
[   479],
[  1038],
[   758],
[   596],
[   998],
[ 52650],
[  1158],
[   647],
[  1267],
[   261],
[  1096],
[  7518],
[   791],
[  1638],
[   664],
[  3023],
[  2502],
[   977],
[  1078],
[  1152],
[   931],
[  1136],
[   801],
[  1649],
[  1366],
[  1041],
[   820],
[  2202],
[   721],
[   871],
[   952],
[  1121],
[  2363],
[  2020],
[   955],
[  1699],
```

```
[  2665],
[  1322],
[   779],
[   730],
[   329],
[   540],
[   502],
[   863],
[   736],
[   329],
[  1076],
[   249],
[   528],
[  4227],
[   512],
[130152],
[  1234],
[   772],
[   212],
[   701],
[   806],
[   395],
[   328],
[   195],
[   599],
[  3889],
[  2836],
[   506],
[  1395],
[  1311],
[   323],
[  3378],
[   752],
[   662],
[  1225],
[  1024],
[  9349],
[ 18267],
[  1735],
[   454],
[   465],
[  7443],
[210408],
[  1748],
[  1051],
[   443],
[   605],
[   446],
[   612],
[   269],
[   421],
[   229],
[   662],
[   209],
[ 20790],
[  2805],
```

```
[   490],
[   240],
[   828],
[ 32514],
[   294],
[   866],
[   538],
[  1878],
[  1752],
[   508],
[   275],
[   370],
[   468],
[   550],
[   557],
[   576],
[   200],
[  2180],
[   619],
[   996],
[   312],
[   129],
[   393],
[    69],
[  3414],
[  1087],
[  1289],
[   491],
[   544],
[  2176],
[   330],
[  6731],
[  1027],
[  2460],
[ 15764],
[   482],
[   818],
[   289],
[   630],
[   366],
[   741],
[   430],
[   638],
[ 70160],
[   592],
[  1689],
[  4638],
[  1342],
[  1262],
[   232],
[   648],
[   996],
[  8472],
[   935],
[   829],
[   704],
```

```
[   665],
[   603],
[   275],
[   567],
[   450],
[ 44301],
[   215],
[   216],
[   521],
[ 16248],
[  3429],
[  1025],
[   656],
[  5473],
[ 27652],
[   358],
[   641],
[   443],
[   435],
[   415],
[   151],
[   127],
[   164],
[   346],
[   368],
[   348],
[   480],
[   840],
[   288],
[  9089],
[   469],
[   125],
[  7298],
[  8842],
[   576],
[   244],
[   555],
[   493],
[   445],
[   367],
[   487],
[   139],
[   273],
[    92],
[  1255],
[   316],
[   482],
[   448],
[   132],
[   549],
[   328],
[  3424],
[  1437],
[   817],
[   287],
[   163],
```

```
[ 15189],
[  9802],
[  1558],
[   446],
[   289],
[   505],
[  8181],
[   587],
[   760],
[  1367],
[   318],
[  1609],
[   155],
[  9978],
[  5938],
[   788],
[  2675],
[   990],
[  1558],
[ 38982],
[  1946],
[   264],
[  2023],
[  1617],
[   878],
[ 13525],
[   646],
[  2032],
[   557],
[   176],
[  4766],
[ 10467],
[   650],
[  1872],
[   262],
[   519],
[ 11024],
[   237],
[   589],
[  8162],
[   475],
[   215],
[   729],
[   378],
[   306],
[   245],
[   992],
[   513],
[   328],
[  6841],
[   699],
[  3406],
[   473],
[  3063],
[ 33525],
[   591],
```

```
[   622],
[   382],
[   258],
[  3332],
[   536],
[   917],
[   348],
[  2469],
[  3637],
[   154],
[  1501],
[  1662],
[ 13896],
[   444],
[ 33686],
[   594],
[   619],
[  1319],
[   312],
[    75],
[    61],
[   104],
[    86],
[   180],
[   163],
[   169],
[   482],
[   263],
[  2155],
[  1817],
[   365],
[   399],
[   880],
[   324],
[   830],
[   744],
[ 17514],
[  1310],
[   214],
[  1330],
[ 16955],
[   663],
[   723],
[   956],
[   678],
[ 20056],
[  1366],
[  1510],
[  1024],
[   857],
[   810],
[   681],
[   336],
[   176],
[   995],
[   260],
```

```
[   475],
[   514],
[   669],
[   310],
[ 35454],
[   362],
[   412],
[   361],
[   429],
[   714],
[   796],
[  4285],
[   212],
[  3189],
[ 19991],
[  1707],
[   297],
[  2905],
[  2436],
[  2169],
[  1210],
[   221],
[   497],
[   588],
[   197],
[   648],
[   717],
[   258],
[  2656],
[   331],
[  2250],
[   348],
[  2595],
[  1182],
[   923],
[   887],
[   734],
[   612],
[   382],
[   704],
[  5779],
[  1920],
[  2271],
[   609],
[   348],
[   442],
[   387],
[   407],
[   532],
[  1372],
[  1378],
[ 66172],
[ 11287],
[   130],
[   947],
[  1144],
```

```
[   369],
[   837],
[   687],
[ 30886],
[  1342],
[  1166],
[   447],
[  2694],
[   955],
[   975],
[   797],
[  1194],
[   284],
[   519],
[   531],
[   488],
[   260],
[  1719],
[  1223],
[ 59420],
[   669],
[  1569],
[   405],
[ 12756],
[  4888],
[   346],
[  1142],
[   385],
[ 35744],
[ 15785],
[  1300],
[   944],
[ 19692],
[   753],
[    85],
[   419],
[   383],
[  1433],
[   722],
[   164],
[   271],
[  1525],
[  8182],
[  2515],
[  2912],
[  2154],
[   575],
[  2707],
[   676],
[   773],
[   251],
[  2001],
[ 10256],
[   553],
[  2863],
[   167],
```

```
[   507],
[   107],
[   698],
[  3182],
[  1134],
[   137],
[   432],
[   821],
[   461],
[   223],
[   638],
[   316],
[   159],
[   190],
[   546],
[   399],
[   852],
[   321],
[   350],
[  9496],
[   168],
[   129],
[   536],
[  1259],
[  3194],
[   563],
[  1073],
[   196],
[   812],
[  2897],
[  1375],
[  1001],
[   178],
[  2052],
[  1681],
[  4658],
[ 13874],
[ 15343],
[  1096],
[   983],
[ 41063],
[   190],
[   213],
[   208],
[    94],
[    85],
[    69],
[    83],
[  2259],
[   252],
[   522],
[  1012],
[   264],
[   145],
[   433],
[   422],
```

```
[   677],
[   208],
[  4092],
[  4517],
[  2563],
[   132],
[   507],
[   895],
[   532],
[   470],
[   207],
[   561],
[  3801],
[   399],
[   228],
[   169],
[  1672],
[   474],
[   640],
[  2939],
[   763],
[ 21485],
[  6820],
[   779],
[   297],
[   968],
[  1108],
[  3164],
[  1851],
[   106],
[    81],
[   445],
[   184],
[  3893],
[ 43933],
[  8909],
[ 18231],
[   935],
[   590],
[   161],
[   829],
[   344],
[   600],
[   530],
[   329],
[  7971],
[ 15821],
[   411],
[   160],
[    99],
[ 20933],
[   510],
[   181],
[   213],
[   261],
[  1401],
```

```
[   861],
[ 25354],
[    94],
[   669],
[   488],
[   244],
[   103],
[   329],
[  3674],
[   676],
[   554],
[   391],
[   311],
[  3021],
[  1447],
[ 40064],
[   377],
[   447],
[   241],
[ 86762],
[   120],
[   142],
[   232],
[   431],
[  1091],
[  1235],
[  3913],
[104673],
[   831],
[  1917],
[   687],
[  4978],
[   388],
[   371],
[   656],
[  2292],
[   387],
[  1088],
[   674],
[  2421],
[  1928],
[   342],
[   255],
[  1264],
[  1916],
[   122],
[   290],
[ 20060],
[  1365],
[   299],
[  5203],
[  1070],
[   306],
[ 17690],
[  4578],
[    89],
```

```
[   148],
[    90],
[   366],
[ 13571],
[ 15238],
[  1812],
[  1613],
[  7057],
[ 12903],
[  3877],
[   672],
[   698],
[   502],
[   309],
[   218],
[   538],
[   374],
[   247],
[  4113],
[  9210],
[   434],
[   202],
[  4534],
[   207],
[   161],
[    81],
[   228],
[   213],
[   133],
[ 11956],
[  1227],
[  1490],
[  1715],
[   396],
[   418],
[  1317],
[   798],
[   676],
[   373],
[  1351],
[ 27771],
[  2398],
[   637],
[   322],
[   449],
[  1880],
[  1101],
[  1259],
[  2398],
[ 23707],
[   531],
[   331],
[   208],
[   105],
[   138],
[   247],
```

```
       [   568],
       [   454],
       [   286],
       [   370],
       [   132],
       [    83],
       [  5896],
       [   373],
       [  3049],
       [   588],
       [   110],
       [   566],
       [   257],
       [ 15498],
       [    76],
       [   256],
       [    66],
       [   133],
       [  1381],
       [  7310],
       [  2007],
       [   337],
       [   172],
       [   617],
       [  1227],
       [  9805],
       [   573],
       [ 16552],
       [   596],
       [   112],
       [   216],
       [  1074],
       [   642],
       [   731],
       [ 16940],
       [   904],
       [   742],
       [   220],
       [   610],
       [   366],
       [   298],
       [   115],
       [   213],
       [   202],
       [   239],
       [   136],
       [   193],
       [   146],
       [    83],
       [   100],
       [  1266],
       [   623]], dtype=int64)
```

In [14]:
```
x.shape
```

Out[14]: (836, 4)

In [15]:
```
y.shape
```

Out[15]: (836, 1)

Split this data into training and testing data set

In [16]:
```
from sklearn.model_selection import train_test_split
```

In [17]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2,random_stat
```

In [18]:
```
x_train.shape
```

Out[18]: (668, 4)

In [19]:
```
x_test.shape
```

Out[19]: (168, 4)

In [20]:
```
y_train.shape
```

Out[20]: (668, 1)

In [21]:
```
y_test.shape
```

Out[21]: (168, 1)

Build the model

In [22]:
```
from sklearn.linear_model import LinearRegression
```

In [23]:
```
mlr = LinearRegression()
mlr.fit(x_train, y_train)
```

Out[23]: ▾ LinearRegression

LinearRegression()

Test the model

In [24]:
```
y_pred = mlr.predict(x_test)
y_pred
```

Out[24]:
```
array([[   238.91251038],
       [  1422.0023255 ],
       [   299.81434734],
       [   836.9545628 ],
       [  4670.69496035],
       [  1174.52247327],
       [   246.74220261],
       [  1027.65987267],
       [  1039.84569115],
       [ 21263.51246526],
       [  1261.76179968],
       [   915.94504225],
       [   495.06629344],
       [  1814.03707863],
       [   239.00785889],
       [   720.14415727],
       [   566.84111325],
       [   651.64516928],
       [   342.86220443],
       [   273.45036346],
       [   447.15673851],
       [   663.01102196],
       [  1084.97894865],
       [  7178.84363682],
       [  3763.43105188],
       [   223.96240972],
       [   373.17996413],
       [  1051.50538957],
       [   491.81924464],
       [  1212.07507672],
       [  1085.96432679],
       [   412.52545909],
       [119222.23451575],
       [   561.75709584],
       [  1027.40339944],
       [   872.25232969],
       [  1447.95187503],
       [  1166.59157904],
       [  2838.27581629],
       [ 37224.76709023],
       [   229.28360577],
       [   345.89377095],
       [   437.17313601],
       [ -8124.06691363],
       [  1107.63019377],
       [   382.48616899],
       [   522.72709265],
       [ -4060.4717921 ],
       [   367.14160596],
       [   646.84696407],
       [   493.05695076],
       [  4602.14718531],
       [   286.74090216],
       [   716.52311004],
       [  4621.17340918],
       [  1518.84464686],
```

```
[   657.86354699],
[  1616.71577801],
[   620.89592389],
[ -9165.26677838],
[  1727.85773572],
[ 55721.09895321],
[   213.88613829],
[   656.0025609 ],
[   494.70003154],
[   809.55616222],
[   264.47284311],
[   417.61376737],
[   644.92762263],
[  1840.76624596],
[   260.94226466],
[   467.35860432],
[   835.81213625],
[   310.48445603],
[   703.46978852],
[  2182.26339994],
[   474.45352519],
[   727.3245829 ],
[  1854.90563389],
[   325.61707946],
[  1024.53420623],
[ 43519.59819123],
[ 54297.39432816],
[   390.79608146],
[ 14316.88741531],
[  7956.28446687],
[  1915.5590053 ],
[  2978.36237947],
[   805.93102501],
[   405.74428169],
[  1353.07250856],
[  1248.5121617 ],
[   413.75681709],
[   458.49497936],
[  1013.85662428],
[  1310.67240828],
[   663.45059694],
[   491.67049413],
[   625.76008557],
[  1610.18830419],
[   530.4145344 ],
[  1093.31674392],
[   795.36785776],
[   626.41054977],
[  5018.51441311],
[   343.51535509],
[  1114.85706594],
[  2192.25762389],
[  1642.63893502],
[  9398.21376307],
[   276.89185132],
[  1011.55600017],
```

```
[   470.5877086 ],
[   644.40424352],
[ 67227.66536009],
[   806.50312702],
[   627.75837186],
[   446.94714669],
[  1896.31166772],
[   581.41909822],
[  1113.87698649],
[  2306.4977046 ],
[ -1074.40357863],
[   578.32005857],
[   735.07814967],
[  4017.42812694],
[  6249.93148898],
[   409.90949097],
[   525.54632765],
[  1419.28966831],
[   449.11070019],
[  4160.49521017],
[  1532.10574814],
[  3601.22628032],
[   507.07538859],
[  3909.15828766],
[  2646.72266218],
[   392.10147044],
[ 29894.97627712],
[   950.65342102],
[  3274.19706784],
[ 64939.56682082],
[  1254.3038573 ],
[   323.64304993],
[   597.81647987],
[ 21984.68015511],
[   426.01379091],
[   507.4230885 ],
[  4320.71996868],
[   976.23752639],
[  1865.73648169],
[  5270.21031079],
[   715.02630731],
[   394.46412053],
[  1503.73313289],
[   343.68037769],
[   400.43356634],
[  1608.17250246],
[  3487.26852388],
[   384.83418832],
[   591.01064039],
[  6102.40204943],
[   361.86120999],
[   261.64461771],
[   505.99388221],
[   441.9620236 ],
[   480.01226012],
[   325.76676289]])
```

In [25]:
```
y_test
```

Out[25]:
```
array([[   104],
       [   880],
       [   161],
       [  1267],
       [  2250],
       [   820],
       [   125],
       [   729],
       [   600],
       [ 11024],
       [  1523],
       [  1096],
       [   405],
       [  1266],
       [   130],
       [  1024],
       [   373],
       [   538],
       [   228],
       [   127],
       [    99],
       [   612],
       [  1501],
       [  8909],
       [  6731],
       [    66],
       [   261],
       [   791],
       [   640],
       [   672],
       [  1194],
       [   290],
       [210408],
       [   329],
       [   656],
       [   553],
       [  1087],
       [   704],
       [  2089],
       [ 27771],
       [   107],
       [   220],
       [   480],
       [  3889],
       [  1510],
       [   332],
       [   329],
       [  4517],
       [   252],
       [   809],
       [   468],
       [  2656],
       [   200],
       [  1085],
       [  2176],
       [  1880],
```

```
[   391],
[  1091],
[   605],
[ 15821],
[   756],
[ 86762],
[    69],
[   590],
[   488],
[   752],
[    83],
[   454],
[   469],
[  1262],
[    85],
[   365],
[   676],
[   324],
[   369],
[  1024],
[   446],
[   566],
[  1342],
[   331],
[   596],
[ 40064],
[ 44301],
[   223],
[  6820],
[ 13874],
[  2169],
[  3424],
[  1199],
[   366],
[  1108],
[  5761],
[   213],
[   348],
[   952],
[  1638],
[   642],
[   442],
[   544],
[  1365],
[   522],
[  1256],
[   744],
[   863],
[  2694],
[   202],
[   589],
[  1319],
[  1259],
[ 15343],
[    94],
[   857],
```

```
[   493],
[   939],
[ 66172],
[   792],
[   449],
[   321],
[  1225],
[   367],
[  1142],
[  3313],
[   136],
[   736],
[   638],
[  5203],
[  2863],
[   184],
[   555],
[  1366],
[  2675],
[  1351],
[  1395],
[  2398],
[   387],
[  3063],
[  1078],
[  2469],
[ 33525],
[   646],
[  2436],
[ 59420],
[   788],
[   154],
[   659],
[ 16037],
[   258],
[   445],
[  2259],
[  1080],
[  1381],
[  3674],
[  1093],
[   190],
[  1672],
[   241],
[   348],
[  1810],
[  2052],
[   218],
[   411],
[  9349],
[   132],
[    94],
[   491],
[   291],
[   239],
[   160]], dtype=int64)
```

Checking r2_score (the evaluation metric)

In [26]:
```
from sklearn.metrics import r2_score
```

In [27]:
```
acc=r2_score(y_test,y_pred)
acc
```

Out[27]:
```
0.8286560931566199
```

Predicting the comment count by giving some new value as input to the model

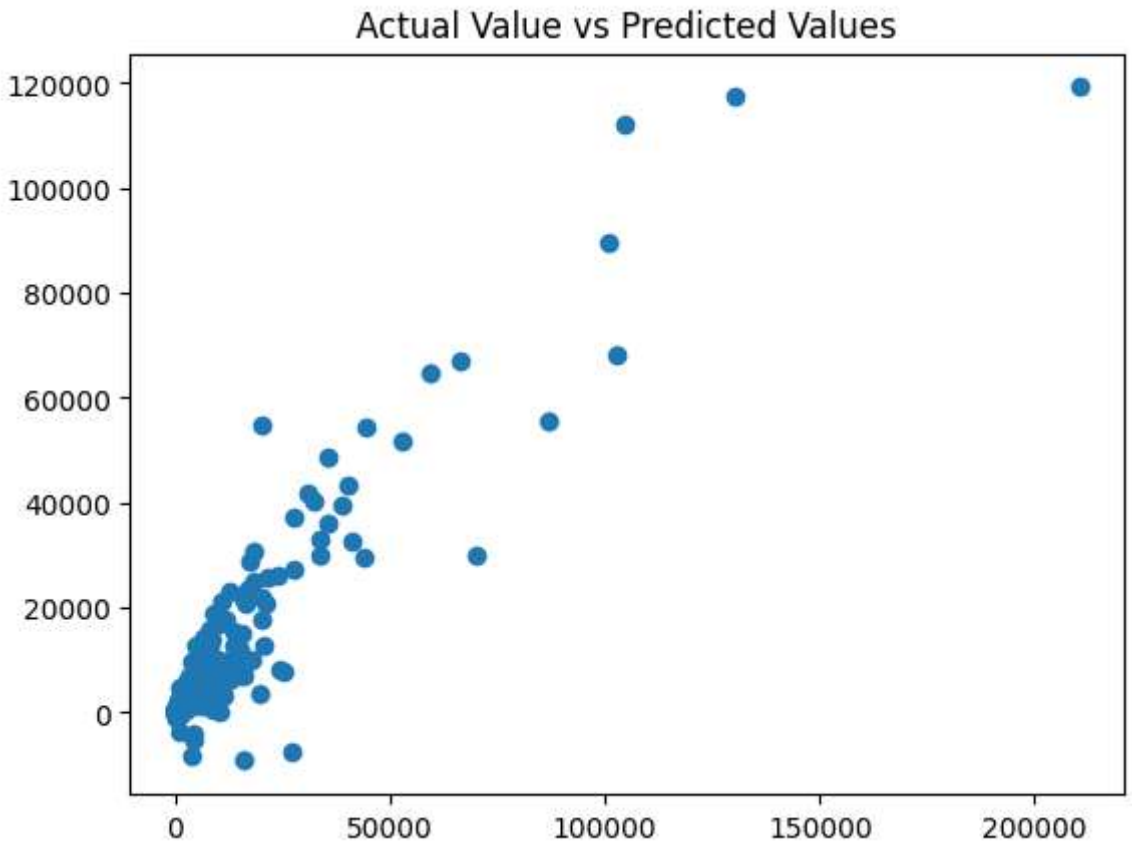
In [29]:
```
mlr.predict([[2,300,23,2]])
```

Out[29]:
```
array([[171.64014801]])
```

Visualizing Actual Y(commentCount) with Predicted Y(commentCount)

In [30]:
```
plt.scatter(y,mlr.predict(x))
plt.title("Actual Value vs Predicted Values")
```

Out[30]:
```
Text(0.5, 1.0, 'Actual Value vs Predicted Values')
```



In [ ]:
```

```

RESULTS AND CONCLUSION

In this MLR model, we have analyzed the Comment count of the YouTube T-Series dataset using the following variables, view count, like count, dislike count and duration. The R2 score of this Multiple Linear Regression (MLR) model is approximately 82% for the Comment count analysis of the T-Series YouTube dataset. By this we conclude that we can analyze the comment counts using the selected independent variables (view count, like count, dislike count). 82% R2 score is promising, it's essential to keep in mind that there may be other unmeasured factors influencing comment counts that the model doesn't capture.

The coefficients of the independent variables provide insights into their impact on comment counts:

Positive coefficients for view count and like count indicate that higher view counts and more likes are associated with more comments. A negative coefficient for duration indicates how the length of a video affects comment counts, that it is associated with fewer comments.

The model performs well, it's important to note that it may not capture all factors influencing comment counts. There could be other unmeasured variables and external factors at play. This is the main disadvantage of this model.

In conclusion, an 82% R2 score indicates a reasonably good fit of MLR model to the data, that is significant portion of the variation in comment counts can be explained by the selected independent variables. Additionally, the accuracy of model can be assessed using cross-validation, because there may be other unmeasured factors that influence comment counts on YouTube videos.

BIBLIOGRAPHY

1. Stuart Russel, Peter Norvig: Artificial Intelligence A Modern Approach, 2nd Edition, Pearson Education, 2003.
2. Elaine Rich, Kevin Knight: Artificial Intelligence, 3rd Edition, Tata McGraw Hill, 2009.
3. Nils J. Nilsson: Principles of Artificial Intelligence, Elsevier, 1980.
4. Hands-On Machine Learning with Scikit-Learn, Keras & tenserflow: Concepts, Tools and techniques to build Intelligent systems. Second edition, O'REILLY. ISBN-13: 978-1492032649, ISBN-10: 1492032646
5. Machine Learning, Tom M Mitchel, McGraw Hill publications, ISBN-0070428077
6. Machine Learning, Saikat Dutt, Subramanian Chandramouli, Amit Kumar Das, 1st Edition, 2019, Pearson Publications, , ISBN 978-93-530-6669-7
7. Su X, Y. X. (2012 May). Linear regression. Wiley Interdisciplinary Reviews:.
8. Tom M Mitchel. (n.d.). Machine Learning, . McGraw Hill publications,.