Name- Bishal Mohanty

SIC- 20BCSB98

Project 1

Step 1: Importing Libraries

In [1]:

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

Step 2: Importing Dataset

In [2]:

```
dataset=pd.read_csv('investment_data.csv')
```

In [3]:

```
dataset
```

Out[3]:

	Capital investment	Employee salary	Advertisement expenditure	City	Turn over
0	165361.76	136897.80	471784.10	Kolkata	192274.39
1	162610.26	151377.59	443898.53	Bengaluru	191804.62
2	153454.07	101145.55	407934.54	Chennai	191062.95
3	144384.97	118671.85	383199.62	Kolkata	182914.55
4	142119.90	91391.77	366168.42	Chennai	166200.50
5	131889.46	99814.71	362861.36	Kolkata	157003.68
6	134628.02	147198.87	127716.82	Bengaluru	156135.07
7	130310.69	145530.06	323876.68	Chennai	155765.16
8	120555.08	148718.95	311613.29	Kolkata	152224.33
9	123347.44	108679.17	304981.62	Bengaluru	149772.52
10	101925.64	110594.11	229160.95	Chennai	146134.51
11	100684.52	91790.61	249744.55	Bengaluru	144271.96
12	93876.31	127320.38	249839.44	Chennai	141598.08
13	92004.95	135495.07	252664.93	Bengaluru	134319.91
14	119955.80	156547.42	256512.92	Chennai	132615.21
15	114536.17	122616.84	261776.23	Kolkata	129929.60
16	78025.67	121597.55	264346.06	Bengaluru	127005.49
17	94669.72	145077.58	282574.31	Kolkata	125382.93
18	91761.72	114175.79	294919.57	Chennai	124279.46
19	86432.26	153514.11	0.00	Kolkata	122789.42
20	76266.42	113867.30	298664.47	Bengaluru	118486.59
21	78402.03	153773.43	299737.29	Kolkata	111325.58
22	74007.12	122782.75	303319.26	Chennai	110364.81
23	67545.09	105751.03	304768.73	Chennai	108746.55
24	77056.57	99281.34	140574.81	Kolkata	108564.60
25	64677.27	139553.16	137962.62	Bengaluru	107416.90
26	75341.43	144135.98	134050.07	Chennai	105746.10
27	72120.16	127864.55	353183.81	Kolkata	105020.87
28	66064.08	182645.56	118148.20	Chennai	103294.94
29	65618.04	153032.06	107138.38	Kolkata	101017.20
30	62007.04	115641.28	91131.24	Chennai	99950.15
31	61148.94	152701.92	88218.23	Kolkata	97496.12
32	63421.42	129219.61	46085.25	Bengaluru	97440.40
33	55506.51	103057.49	214634.81	Chennai	96791.48
34	46438.63	157693.92	210797.67	Bengaluru	96725.36
35	46026.58	85047.44	205517.64	Kolkata	96492.07
36	28676.32	127056.21	201126.82	Chennai	90720.75

	Capital investment	Employee salary	Advertisement expenditure	City	Turn over
37	44082.51	51283.14	197029.42	Bengaluru	89961.70
38	20242.15	65947.93	185265.10	Kolkata	81241.62
39	38571.07	82982.09	174999.30	Bengaluru	81018.32
40	28766.89	118546.05	172795.67	Bengaluru	78252.47
41	27905.48	84710.77	164470.71	Chennai	77811.39
42	23653.49	96189.63	148001.11	Bengaluru	71511.05
43	15518.29	127382.30	35534.17	Kolkata	69771.54
44	22190.30	154806.14	28334.72	Bengaluru	65212.89
45	1012.79	124153.04	1903.93	Kolkata	64938.64
46	1328.02	115816.21	297114.46	Chennai	49503.31
47	12.56	135426.92	0.00	Bengaluru	42572.29
48	554.61	51743.15	0.00	Kolkata	35685.97
49	12.56	116983.80	45173.06	Bengaluru	14693.96

Step 3: Divide dataset into feature matrix(x) and dependent variable vector(y)

In [4]:

```
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,-1].values
```

In [5]:

```
x
```

Out[5]:

```
array([[165361.76, 136897.8, 471784.1, 'Kolkata'],
       [162610.26, 151377.59, 443898.53, 'Bengaluru'],
       [153454.07, 101145.55, 407934.54, 'Chennai'],
       [144384.97, 118671.85, 383199.62, 'Kolkata'],
       [142119.9, 91391.77, 366168.42, 'Chennai'],
       [131889.46, 99814.71, 362861.36, 'Kolkata'],
       [134628.02, 147198.87, 127716.82, 'Bengaluru'],
       [130310.69, 145530.06, 323876.68, 'Chennai'],
       [120555.08, 148718.95, 311613.29, 'Kolkata'],
       [123347.44, 108679.17, 304981.62, 'Bengaluru'],
       [101925.64, 110594.11, 229160.95, 'Chennai'],
       [100684.52, 91790.61, 249744.55, 'Bengaluru'],
       [93876.31, 127320.38, 249839.44, 'Chennai'],
       [92004.95, 135495.07, 252664.93, 'Bengaluru'],
       [119955.8, 156547.42, 256512.92, 'Chennai'],
       [114536.17, 122616.84, 261776.23, 'Kolkata'],
       [78025.67, 121597.55, 264346.06, 'Bengaluru'],
       [94669.72, 145077.58, 282574.31, 'Kolkata'],
       [91761.72, 114175.79, 294919.57, 'Chennai'],
       [86432.26, 153514.11, 0.0, 'Kolkata'],
       [76266.42, 113867.3, 298664.47, 'Bengaluru'],
       [78402.03, 153773.43, 299737.29, 'Kolkata'],
       [74007.12, 122782.75, 303319.26, 'Chennai'],
       [67545.09, 105751.03, 304768.73, 'Chennai'],
       [77056.57, 99281.34, 140574.81, 'Kolkata'],
       [64677.27, 139553.16, 137962.62, 'Bengaluru'],
       [75341.43, 144135.98, 134050.07, 'Chennai'],
       [72120.16, 127864.55, 353183.81, 'Kolkata'],
       [66064.08, 182645.56, 118148.2, 'Chennai'],
       [65618.04, 153032.06, 107138.38, 'Kolkata'],
       [62007.04, 115641.28, 91131.24, 'Chennai'],
       [61148.94, 152701.92, 88218.23, 'Kolkata'],
       [63421.42, 129219.61, 46085.25, 'Bengaluru'],
       [55506.51, 103057.49, 214634.81, 'Chennai'],
       [46438.63, 157693.92, 210797.67, 'Bengaluru'],
       [46026.58, 85047.44, 205517.64, 'Kolkata'],
       [28676.32, 127056.21, 201126.82, 'Chennai'],
       [44082.51, 51283.14, 197029.42, 'Bengaluru'],
       [20242.15, 65947.93, 185265.1, 'Kolkata'],
       [38571.07, 82982.09, 174999.3, 'Bengaluru'],
       [28766.89, 118546.05, 172795.67, 'Bengaluru'],
       [27905.48, 84710.77, 164470.71, 'Chennai'],
       [23653.49, 96189.63, 148001.11, 'Bengaluru'],
       [15518.29, 127382.3, 35534.17, 'Kolkata'],
       [22190.3, 154806.14, 28334.72, 'Bengaluru'],
       [1012.79, 124153.04, 1903.93, 'Kolkata'],
       [1328.02, 115816.21, 297114.46, 'Chennai'],
       [12.56, 135426.92, 0.0, 'Bengaluru'],
       [554.61, 51743.15, 0.0, 'Kolkata'],
       [12.56, 116983.8, 45173.06, 'Bengaluru']], dtype=object)
```

In [6]:

```
y
```

Out[6]:

```
array([192274.39, 191804.62, 191062.95, 182914.55, 166200.5 , 157003.68,
       156135.07, 155765.16, 152224.33, 149772.52, 146134.51, 144271.96,
       141598.08, 134319.91, 132615.21, 129929.6 , 127005.49, 125382.93,
       124279.46, 122789.42, 118486.59, 111325.58, 110364.81, 108746.55,
       108564.6 , 107416.9 , 105746.1 , 105020.87, 103294.94, 101017.2 ,
        99950.15,  97496.12,  97440.4 ,  96791.48,  96725.36,  96492.07,
        90720.75,  89961.7 ,  81241.62,  81018.32,  78252.47,  77811.39,
        71511.05,  69771.54,  65212.89,  64938.64,  49503.31,  42572.29,
        35685.97,  14693.96])
```

Step 4: Replace Missing data

In [7]:

```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(x[:,0:3])
x[:,0:3]=imputer.transform(x[:,0:3])
```

Step 5: Encoding the Data

Feature matrix using OneHotEncoding

In [8]:

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct=ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])], remainder='passt
hrough')
x=np.array(ct.fit_transform(x))
```

In [9]:

```
x
```

Out[9]:

```
array([[0.0, 0.0, 1.0, 165361.76, 136897.8, 471784.1],
       [1.0, 0.0, 0.0, 162610.26, 151377.59, 443898.53],
       [0.0, 1.0, 0.0, 153454.07, 101145.55, 407934.54],
       [0.0, 0.0, 1.0, 144384.97, 118671.85, 383199.62],
       [0.0, 1.0, 0.0, 142119.9, 91391.77, 366168.42],
       [0.0, 0.0, 1.0, 131889.46, 99814.71, 362861.36],
       [1.0, 0.0, 0.0, 134628.02, 147198.87, 127716.82],
       [0.0, 1.0, 0.0, 130310.69, 145530.06, 323876.68],
       [0.0, 0.0, 1.0, 120555.08, 148718.95, 311613.29],
       [1.0, 0.0, 0.0, 123347.44, 108679.17, 304981.62],
       [0.0, 1.0, 0.0, 101925.64, 110594.11, 229160.95],
       [1.0, 0.0, 0.0, 100684.52, 91790.61, 249744.55],
       [0.0, 1.0, 0.0, 93876.31, 127320.38, 249839.44],
       [1.0, 0.0, 0.0, 92004.95, 135495.07, 252664.93],
       [0.0, 1.0, 0.0, 119955.8, 156547.42, 256512.92],
       [0.0, 0.0, 1.0, 114536.17, 122616.84, 261776.23],
       [1.0, 0.0, 0.0, 78025.67, 121597.55, 264346.06],
       [0.0, 0.0, 1.0, 94669.72, 145077.58, 282574.31],
       [0.0, 1.0, 0.0, 91761.72, 114175.79, 294919.57],
       [0.0, 0.0, 1.0, 86432.26, 153514.11, 0.0],
       [1.0, 0.0, 0.0, 76266.42, 113867.3, 298664.47],
       [0.0, 0.0, 1.0, 78402.03, 153773.43, 299737.29],
       [0.0, 1.0, 0.0, 74007.12, 122782.75, 303319.26],
       [0.0, 1.0, 0.0, 67545.09, 105751.03, 304768.73],
       [0.0, 0.0, 1.0, 77056.57, 99281.34, 140574.81],
       [1.0, 0.0, 0.0, 64677.27, 139553.16, 137962.62],
       [0.0, 1.0, 0.0, 75341.43, 144135.98, 134050.07],
       [0.0, 0.0, 1.0, 72120.16, 127864.55, 353183.81],
       [0.0, 1.0, 0.0, 66064.08, 182645.56, 118148.2],
       [0.0, 0.0, 1.0, 65618.04, 153032.06, 107138.38],
       [0.0, 1.0, 0.0, 62007.04, 115641.28, 91131.24],
       [0.0, 0.0, 1.0, 61148.94, 152701.92, 88218.23],
       [1.0, 0.0, 0.0, 63421.42, 129219.61, 46085.25],
       [0.0, 1.0, 0.0, 55506.51, 103057.49, 214634.81],
       [1.0, 0.0, 0.0, 46438.63, 157693.92, 210797.67],
       [0.0, 0.0, 1.0, 46026.58, 85047.44, 205517.64],
       [0.0, 1.0, 0.0, 28676.32, 127056.21, 201126.82],
       [1.0, 0.0, 0.0, 44082.51, 51283.14, 197029.42],
       [0.0, 0.0, 1.0, 20242.15, 65947.93, 185265.1],
       [1.0, 0.0, 0.0, 38571.07, 82982.09, 174999.3],
       [1.0, 0.0, 0.0, 28766.89, 118546.05, 172795.67],
       [0.0, 1.0, 0.0, 27905.48, 84710.77, 164470.71],
       [1.0, 0.0, 0.0, 23653.49, 96189.63, 148001.11],
       [0.0, 0.0, 1.0, 15518.29, 127382.3, 35534.17],
       [1.0, 0.0, 0.0, 22190.3, 154806.14, 28334.72],
       [0.0, 0.0, 1.0, 1012.79, 124153.04, 1903.93],
       [0.0, 1.0, 0.0, 1328.02, 115816.21, 297114.46],
       [1.0, 0.0, 0.0, 12.56, 135426.92, 0.0],
       [0.0, 0.0, 1.0, 554.61, 51743.15, 0.0],
       [1.0, 0.0, 0.0, 12.56, 116983.8, 45173.06]], dtype=object)
```

Dependent variable vector using LabelEncoder

In [10]:

```
#No need to encode the dependent variable vector
```

Step 6: Splitting of Dataset into training and testing dataset

In [11]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=1)
```

In [12]:

```
xtrain
```

Out[12]:

```
array([[1.0, 0.0, 0.0, 63421.42, 129219.61, 46085.25],
       [1.0, 0.0, 0.0, 38571.07, 82982.09, 174999.3],
       [0.0, 0.0, 1.0, 78402.03, 153773.43, 299737.29],
       [0.0, 1.0, 0.0, 28676.32, 127056.21, 201126.82],
       [0.0, 0.0, 1.0, 86432.26, 153514.11, 0.0],
       [1.0, 0.0, 0.0, 23653.49, 96189.63, 148001.11],
       [1.0, 0.0, 0.0, 12.56, 116983.8, 45173.06],
       [0.0, 1.0, 0.0, 75341.43, 144135.98, 134050.07],
       [0.0, 1.0, 0.0, 74007.12, 122782.75, 303319.26],
       [1.0, 0.0, 0.0, 92004.95, 135495.07, 252664.93],
       [0.0, 1.0, 0.0, 27905.48, 84710.77, 164470.71],
       [0.0, 0.0, 1.0, 94669.72, 145077.58, 282574.31],
       [0.0, 0.0, 1.0, 1012.79, 124153.04, 1903.93],
       [0.0, 0.0, 1.0, 77056.57, 99281.34, 140574.81],
       [0.0, 1.0, 0.0, 67545.09, 105751.03, 304768.73],
       [0.0, 1.0, 0.0, 142119.9, 91391.77, 366168.42],
       [0.0, 1.0, 0.0, 55506.51, 103057.49, 214634.81],
       [0.0, 1.0, 0.0, 119955.8, 156547.42, 256512.92],
       [0.0, 1.0, 0.0, 62007.04, 115641.28, 91131.24],
       [0.0, 1.0, 0.0, 101925.64, 110594.11, 229160.95],
       [0.0, 1.0, 0.0, 66064.08, 182645.56, 118148.2],
       [1.0, 0.0, 0.0, 22190.3, 154806.14, 28334.72],
       [1.0, 0.0, 0.0, 46438.63, 157693.92, 210797.67],
       [0.0, 1.0, 0.0, 91761.72, 114175.79, 294919.57],
       [1.0, 0.0, 0.0, 76266.42, 113867.3, 298664.47],
       [1.0, 0.0, 0.0, 64677.27, 139553.16, 137962.62],
       [1.0, 0.0, 0.0, 134628.02, 147198.87, 127716.82],
       [0.0, 1.0, 0.0, 130310.69, 145530.06, 323876.68],
       [1.0, 0.0, 0.0, 12.56, 135426.92, 0.0],
       [1.0, 0.0, 0.0, 162610.26, 151377.59, 443898.53],
       [1.0, 0.0, 0.0, 78025.67, 121597.55, 264346.06],
       [0.0, 0.0, 1.0, 165361.76, 136897.8, 471784.1],
       [0.0, 0.0, 1.0, 114536.17, 122616.84, 261776.23],
       [0.0, 0.0, 1.0, 131889.46, 99814.71, 362861.36],
       [1.0, 0.0, 0.0, 100684.52, 91790.61, 249744.55],
       [1.0, 0.0, 0.0, 123347.44, 108679.17, 304981.62],
       [0.0, 0.0, 1.0, 120555.08, 148718.95, 311613.29],
       [0.0, 1.0, 0.0, 93876.31, 127320.38, 249839.44],
       [0.0, 0.0, 1.0, 15518.29, 127382.3, 35534.17],
       [1.0, 0.0, 0.0, 44082.51, 51283.14, 197029.42]], dtype=object)
```

In [13]:

```
ytrain
```

Out[13]:

```
array([ 97440.4 ,  81018.32, 111325.58,  90720.75, 122789.42,  71511.05,
        14693.96, 105746.1 , 110364.81, 134319.91,  77811.39, 125382.93,
        64938.64, 108564.6 , 108746.55, 166200.5 ,  96791.48, 132615.21,
        99950.15, 146134.51, 103294.94,  65212.89,  96725.36, 124279.46,
       118486.59, 107416.9 , 156135.07, 155765.16,  42572.29, 191804.62,
       127005.49, 192274.39, 129929.6 , 157003.68, 144271.96, 149772.52,
       152224.33, 141598.08,  69771.54,  89961.7 ])
```

In [14]:

```
xtest
```

Out[14]:

```
array([[0.0, 0.0, 1.0, 72120.16, 127864.55, 353183.81],
       [0.0, 0.0, 1.0, 46026.58, 85047.44, 205517.64],
       [1.0, 0.0, 0.0, 28766.89, 118546.05, 172795.67],
       [0.0, 0.0, 1.0, 20242.15, 65947.93, 185265.1],
       [0.0, 1.0, 0.0, 153454.07, 101145.55, 407934.54],
       [0.0, 0.0, 1.0, 144384.97, 118671.85, 383199.62],
       [0.0, 0.0, 1.0, 554.61, 51743.15, 0.0],
       [0.0, 0.0, 1.0, 65618.04, 153032.06, 107138.38],
       [0.0, 1.0, 0.0, 1328.02, 115816.21, 297114.46],
       [0.0, 0.0, 1.0, 61148.94, 152701.92, 88218.23]], dtype=object)
```

In [15]:

```
ytest
```

Out[15]:

```
array([105020.87,  96492.07,  78252.47,  81241.62, 191062.95, 182914.55,
        35685.97, 101017.2 ,  49503.31,  97496.12])
```

Step 7: Feature Scaling

In [16]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xtrain[:,3:]=sc.fit_transform(xtrain[:,3:])
xtest[:,3:]=sc.fit_transform(xtest[:,3:])
```

In [17]:

```
xtrain
```

Out[17]:

```
array([[1.0, 0.0, 0.0, -0.3213303849622652, 0.18700266744239274,
        -1.3700842498314485],
       [1.0, 0.0, 0.0, -0.8955888632549122, -1.6137113933587277,
        -0.28442172627772067],
       [0.0, 0.0, 1.0, 0.024851552213493347, 1.1432479184934816,
        0.7660716916733844],
       [0.0, 1.0, 0.0, -1.1242433521488782, 0.102749342455734,
        -0.06438621193773587],
       [0.0, 0.0, 1.0, 0.21041946816397133, 1.133148735616741,
        -1.758195776958278],
       [1.0, 0.0, 0.0, -1.2403142611108307, -1.0993455063779112,
        -0.5117896748200481],
       [1.0, 0.0, 0.0, -1.786624654615268, -0.2895193240547929,
        -1.377766348852932],
       [0.0, 1.0, 0.0, -0.04587483589220653, 0.7679187112767512,
        -0.6292797489632387],
       [0.0, 1.0, 0.0, -0.07670896217898561, -0.06367998831873588,
        0.7962376091160044],
       [1.0, 0.0, 0.0, 0.3391969091012882, 0.43139962169069607,
        0.36964711927627536],
       [0.0, 1.0, 0.0, -1.1420564374940387, -1.546388180897815,
        -0.3730892965608864],
       [0.0, 0.0, 1.0, 0.40077619330452474, 0.8045891979215861,
        0.6215319446209713],
       [0.0, 0.0, 1.0, -1.7635106720056608, -0.010314226695379883,
        -1.7421616406599485],
       [0.0, 0.0, 1.0, -0.006240235717204621, -0.9789392720271195,
        -0.5743310006000313],
       [0.0, 1.0, 0.0, -0.2260378655739653, -0.7269780562866471,
        0.8084444652190854],
       [0.0, 1.0, 0.0, 1.4972866312465332, -1.2861975253390332,
        1.3255280755068437],
       [0.0, 1.0, 0.0, -0.5042334093632597, -0.8318776129957924,
        0.049372672462672844],
       [0.0, 1.0, 0.0, 0.9851038113375158, 1.2512805897423231,
        0.4020533506685055],
       [0.0, 1.0, 0.0, -0.35401482226506187, -0.3418035867251681,
        -0.9907249543987474],
       [0.0, 1.0, 0.0, 0.5684508368332148, -0.5383649504456615,
        0.17170600858774815],
       [0.0, 1.0, 0.0, -0.26026203340001963, 2.2676691775981626,
        -0.7631989324302334],
       [1.0, 0.0, 0.0, -1.2741266324799672, 1.1834666719588485,
        -1.5195721077818902],
       [1.0, 0.0, 0.0, -0.7137800342655379, 1.2959308796088318,
        0.01705781542708258],
       [0.0, 1.0, 0.0, 0.33357618787703514, -0.398876899674553,
        0.725498782413492],
       [1.0, 0.0, 0.0, -0.024499549434031147, -0.41089100165529335,
        0.7570368310936499],
       [1.0, 0.0, 0.0, -0.2923093647366418, 0.5894413964873636,
        -0.5963298190950038],
       [1.0, 0.0, 0.0, 1.3241592665071586, 0.8872025545643478,
        -0.682615845201753],
       [0.0, 1.0, 0.0, 1.2243915225483335, 0.8222109712829453,
        0.9693639710211176],
       [1.0, 0.0, 0.0, -1.786624654615268, 0.4287455290135026,
        -1.758195776958278],
       [1.0, 0.0, 0.0, 1.9707915498198545, 1.0499422486515493,
```

```
        1.9801399458088698],
       [1.0, 0.0, 0.0, 0.01615437406951465, -0.10983744453311507,
        0.46802092006648527],
       [0.0, 0.0, 1.0, 2.0343750487654315, 0.4860287547953159,
        2.2149810534843644],
       [0.0, 0.0, 1.0, 0.8598633830655122, -0.07014133118114546,
        0.4463788405343934],
       [0.0, 0.0, 1.0, 1.2608747937205333, -0.9581672483846423,
        1.297677340035081],
       [1.0, 0.0, 0.0, 0.539770207281576, -1.270664754636584,
        0.34505284797275304],
       [1.0, 0.0, 0.0, 1.0634800927708346, -0.6129420338097887,
        0.8102373395771508],
       [0.0, 0.0, 1.0, 0.9989523736664655, 0.946401867796169,
        0.8660866095952406],
       [0.0, 1.0, 0.0, 0.38244154533127, 0.11303740793679416,
        0.3458519735673645],
       [0.0, 0.0, 1.0, -1.4283078939009661, 0.11544887409338739,
        -1.4589412218916487],
       [1.0, 0.0, 0.0, -0.7682268282090787, -2.8482227810289933,
        -0.09889287450872049]], dtype=object)
```

In [18]:

```
xtest
```

Out[18]:

```
array([[0.0, 0.0, 1.0, 0.25133340117623953, 0.5889703053597811,
        1.0279045230053547],
       [0.0, 0.0, 1.0, -0.2628037665160427, -0.7514947213553398,
        -0.11208849177028128],
       [1.0, 0.0, 0.0, -0.6028815991418, 0.29723825358317,
        -0.36470435652588457],
       [0.0, 0.0, 1.0, -0.7708495697657275, -1.3494385149087984,
        -0.2684394970917044],
       [0.0, 1.0, 0.0, 1.8539033101246853, -0.24751504007341874,
        1.450583933962291],
       [0.0, 0.0, 1.0, 1.6752094939630824, 0.3011766441088446,
        1.2596286459566892],
       [0.0, 0.0, 1.0, -1.1587647601853064, -1.7941441665088809,
        -1.6986988441155333],
       [0.0, 0.0, 1.0, 0.1232183031024855, 1.3768835242693027,
        -0.8715831627487197],
       [0.0, 1.0, 0.0, -1.1435258076588655, 0.2117758053112525,
        0.595045274722486],
       [0.0, 0.0, 1.0, 0.0351609949012508, 1.3665479102140872,
        -1.0176480253946951]], dtype=object)
```

Build Multiple Regression Model

Step 1: Training the Linear Model

In [19]:

```
from sklearn.linear_model import LinearRegression
LR=LinearRegression()
LR.fit(xtrain,ytrain)
```

Out[19]:

```
LinearRegression()
```

Step 2: Testing the Linear Model

In [20]:

```
yestimate=LR.predict(xtest)
```

In [21]:

```
yestimate
```

Out[21]:

```
array([126373.64824823, 105556.07466397,  92765.95892989,  88140.21796932,
       182037.91922702, 174951.41361381,  70338.24649832, 115367.10257703,
        78548.73483006, 111917.35895451])
```

In [22]:

```
ytest
```

Out[22]:

```
array([105020.87,  96492.07,  78252.47,  81241.62, 191062.95, 182914.55,
        35685.97, 101017.2 ,  49503.31,  97496.12])
```

In [23]:

```
yestimate.reshape(len(yestimate),1)
ytest.reshape(len(ytest),1)
```

Out[23]:

```
array([[105020.87],
       [ 96492.07],
       [ 78252.47],
       [ 81241.62],
       [191062.95],
       [182914.55],
       [ 35685.97],
       [101017.2 ],
       [ 49503.31],
       [ 97496.12]])
```

In [24]:

```
np.concatenate((yestimate.reshape(len(yestimate),1),ytest.reshape(len(ytest),1)),1)
```

Out[24]:

```
array([[126373.64824823, 105020.87      ],
       [105556.07466397,  96492.07      ],
       [ 92765.95892989,  78252.47      ],
       [ 88140.21796932,  81241.62      ],
       [182037.91922702, 191062.95      ],
       [174951.41361381, 182914.55      ],
       [ 70338.24649832,  35685.97      ],
       [115367.10257703, 101017.2       ],
       [ 78548.73483006,  49503.31      ],
       [111917.35895451,  97496.12      ]])
```

Coefficients of the regression

In [26]:

```
LR.coef_
```

Out[26]:

```
array([-2.85177769e+02,  2.97560876e+02, -1.23831070e+01,  3.35087290e+04,
       -2.42488847e+02,  3.43382663e+03])
```

In [27]:

```
LR.intercept_
```

Out[27]:

```
114577.34132777256
```

In [30]:

```
#According to given data in the question
```

In [ ]:

```
print(LR.predict([[1.0,0.0,0.0,84592,193348,45363]]))
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
'''
Project 2:

For given “logistic_data.csv” dataset, determine the classification model.
a)      Using logistic regression algorithm.
b)      Using KNN algorithm
c)      Compare (a) and (b) and state which gives better performance in terms of metric parameter such as accuracy score, precision score, recall.

'''

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score,precision_score
from sklearn.neighbors import KNeighborsClassifier

# Import data set
dataset=pd.read_csv('../Data/Logistic Data.csv')


# To create feature matrix and dependent variable vector
a=dataset.iloc[:,:-1].values
b=dataset.iloc[:,-1].values

# Replace the missing data

imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
imputer.fit(a[:,:])
a[:,:]=imputer.transform(a[:,:])

# Spiliting of data set into training and testing set

atrain,atest,btrain,btest=train_test_split(a,b,test_size=0.2,random_state=1)

# Feature scaling

sc=StandardScaler()
atrain=sc.fit_transform(atrain)
atest=sc.fit_transform(atest)


# Using logistic regression algorithm.
# Training the classification model

LoR=LogisticRegression(random_state=0)
LoR.fit(atrain,btrain)

# Testing the linear model
bestimated=LoR.predict(atest)

# Performance matrix
cm=confusion_matrix(btest,bestimated)
print("Logistic regression :")
print(f"Accuracy score : {accuracy_score(btest,bestimated)}")
```

```
print(f"Precision score : {precision_score(btest,bestimated)}")

error_rate=[]
for i in range(1,30):
    KC=KNeighborsClassifier(n_neighbors=i)
    KC.fit(atrain,btrain)
    bpred_i=KC.predict(atest)
    error_rate.append(np.mean(bpred_i!=btest))


fig = plt.figure()
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

ax1.set_title("Logistic Regression")
ax1.plot(range(1,30),error_rate,marker='o',markerfacecolor='red',markersize=5)
ax1.set_xlabel('K value')
ax1.set_ylabel('Error rate')

# By using KNN Algorithm
# Build my KNN classification model
# Training the classification model

KC=KNeighborsClassifier(n_neighbors=7,weights='uniform',p=2)
KC.fit(atrain,btrain)

# Testing the linear model
bestimated=KC.predict(atest)

# Performance matrix

cm=confusion_matrix(btest,bestimated)
print("\nK Nearest Neighbours : ")
print(f"Accuracy score : {accuracy_score(btest,bestimated)}")
print(f"Precision score : {precision_score(btest,bestimated)}")
print("\n\n")

error_rate=[]
for i in range(1,30):
    KC=KNeighborsClassifier(n_neighbors=i)
    KC.fit(atrain,btrain)
    bpred_i=KC.predict(atest)
    error_rate.append(np.mean(bpred_i!=btest))


ax2.set_title("K Nearest Neighbours")
ax2.plot(range(1,30),error_rate,marker='o',markerfacecolor='red',markersize=5)
ax2.set_xlabel('K value')
ax2.set_ylabel('Error rate')


fig.tight_layout()
plt.show()



'''
According to the performance metrics, The KNN algorithm provides a better accuracy scor
e
and precision score than that of Logistic Regression .
```

```
Logistic regression :
Accuracy score : 0.8125
Precision score : 0.823232323232323

K Nearest Neighbours :
Accuracy score : 0.9
Precision score : 0.84444444444442

'''
```

In [ ]:

In [ ]:

PROJECT : 3
Data Preprocessing

Step1 : Importing the libraries

In [2]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
```

Step 2: Import data set

In [3]:

```
1 dataset=pd.read_csv('captia_income.csv')
```

In [4]:

```
1 dataset
```

Out[4]:

	year	per capita income (US$)
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392
7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681
10	1980	8355.968120
11	1981	9434.390652
12	1982	9619.438377
13	1983	10416.536590
14	1984	10790.328720
15	1985	11018.955850
16	1986	11482.891530
17	1987	12974.806620
18	1988	15080.283450
19	1989	16426.725480
20	1990	16838.673200
21	1991	17266.097690
22	1992	16412.083090
23	1993	15875.586730
24	1994	15755.820270
25	1995	16369.317250
26	1996	16699.826680
27	1997	17310.757750
28	1998	16622.671870
29	1999	17581.024140
30	2000	18987.382410
31	2001	18601.397240
32	2002	19232.175560
33	2003	22739.426280

	year	per capita income (US$)
34	2004	25719.147150
35	2005	29198.055690
36	2006	32738.262900
37	2007	36144.481220
38	2008	37446.486090
39	2009	32755.176820
40	2010	38420.522890
41	2011	42334.711210
42	2012	42665.255970
43	2013	42676.468370
44	2014	41039.893600
45	2015	35175.188980
46	2016	34229.193630

Step3: To create feature matrix and dependent variable vector

In [5]:

```
a=dataset.iloc[:,:-1].values
b=dataset.iloc[:,-1].values
```

In [6]:

```
1 a
```

Out[6]:

```
array([[1970],
       [1971],
       [1972],
       [1973],
       [1974],
       [1975],
       [1976],
       [1977],
       [1978],
       [1979],
       [1980],
       [1981],
       [1982],
       [1983],
       [1984],
       [1985],
       [1986],
       [1987],
       [1988],
       [1989],
       [1990],
       [1991],
       [1992],
       [1993],
       [1994],
       [1995],
       [1996],
       [1997],
       [1998],
       [1999],
       [2000],
       [2001],
       [2002],
       [2003],
       [2004],
       [2005],
       [2006],
       [2007],
       [2008],
       [2009],
       [2010],
       [2011],
       [2012],
       [2013],
       [2014],
       [2015],
       [2016]], dtype=int64)
```

In [7]:

```
1 b
```

Out[7]:

```
array([ 3399.299037,  3768.297935,  4251.175484,  4804.463248,
        5576.514583,  5998.144346,  7062.131392,  7100.12617 ,
        7247.967035,  7602.912681,  8355.96812 ,  9434.390652,
        9619.438377, 10416.53659 , 10790.32872 , 11018.95585 ,
       11482.89153 , 12974.80662 , 15080.28345 , 16426.72548 ,
       16838.6732  , 17266.09769 , 16412.08309 , 15875.58673 ,
       15755.82027 , 16369.31725 , 16699.82668 , 17310.75775 ,
       16622.67187 , 17581.02414 , 18987.38241 , 18601.39724 ,
       19232.17556 , 22739.42628 , 25719.14715 , 29198.05569 ,
       32738.2629  , 36144.48122 , 37446.48609 , 32755.17682 ,
       38420.52289 , 42334.71121 , 42665.25597 , 42676.46837 ,
       41039.8936  , 35175.18898 , 34229.19363 ])
```

Step4: Replace missing data

In [8]:

```
1 from sklearn.impute import SimpleImputer
2 imputer=SimpleImputer(missing_values=np.nan,strategy='mean')
3 imputer.fit(a[:,:])
4 a[:,:]=imputer.transform(a[:,:])
```

In [9]:

```
1 a
```

Out[9]:

```
array([[1970],
       [1971],
       [1972],
       [1973],
       [1974],
       [1975],
       [1976],
       [1977],
       [1978],
       [1979],
       [1980],
       [1981],
       [1982],
       [1983],
       [1984],
       [1985],
       [1986],
       [1987],
       [1988],
       [1989],
       [1990],
       [1991],
       [1992],
       [1993],
       [1994],
       [1995],
       [1996],
       [1997],
       [1998],
       [1999],
       [2000],
       [2001],
       [2002],
       [2003],
       [2004],
       [2005],
       [2006],
       [2007],
       [2008],
       [2009],
       [2010],
       [2011],
       [2012],
       [2013],
       [2014],
       [2015],
       [2016]], dtype=int64)
```

Step5: Encoding(not required)

step6 : spiliting of data set into training and testing set

In [10]:

```
from sklearn.model_selection import train_test_split
atrain,atest,btrain,btest=train_test_split(a,b,test_size=0.2,random_state=1)
```

In [11]:

```
atrain
```

Out[11]:

```
array([[1989],
       [2006],
       [2016],
       [2003],
       [1993],
       [2004],
       [1997],
       [1991],
       [1983],
       [2008],
       [1987],
       [2012],
       [1974],
       [1998],
       [1984],
       [1980],
       [2011],
       [2000],
       [2001],
       [2010],
       [1990],
       [1988],
       [1995],
       [1976],
       [1977],
       [2014],
       [1971],
       [1986],
       [1970],
       [1985],
       [1975],
       [1981],
       [1979],
       [1978],
       [1982],
       [2013],
       [2007]], dtype=int64)
```

In [12]:

```
1 btrain
```

Out[12]:

```
array([16426.72548 , 32738.2629  , 34229.19363 , 22739.42628 ,
       15875.58673 , 25719.14715 , 17310.75775 , 17266.09769 ,
       10416.53659 , 37446.48609 , 12974.80662 , 42665.25597 ,
        5576.514583, 16622.67187 , 10790.32872 ,  8355.96812 ,
       42334.71121 , 18987.38241 , 18601.39724 , 38420.52289 ,
       16838.6732  , 15080.28345 , 16369.31725 ,  7062.131392,
        7100.12617 , 41039.8936  ,  3768.297935, 11482.89153 ,
        3399.299037, 11018.95585 ,  5998.144346,  9434.390652,
        7602.912681,  7247.967035,  9619.438377, 42676.46837 ,
       36144.48122 ])
```

step7 : Feature scaling (not required)

Part B: build my first linear model

step 1: training the model

In [14]:

```
1 from sklearn.linear_model import LinearRegression
2 LR=LinearRegression()
3 LR.fit(atrain,btrain)
```

Out[14]:

```
LinearRegression()
```

step 2: testing the linear model

In [15]:

```
1 yestimated=LR.predict(atest)
```

In [16]:

```
1 yestimated
```

Out[16]:

```
array([20349.94572643, 18613.49135581, 33373.35350612, 29900.44476487,
        1248.94764955,  2117.17483487, 24691.081653  , 27295.76320894,
       38582.716618  , 22086.40009706])
```

In [17]:

```
1 btest
```

Out[17]:

```
array([15755.82027 , 16412.08309 , 32755.17682 , 29198.05569 ,
        4251.175484,  4804.463248, 17581.02414 , 19232.17556 ,
       35175.18898 , 16699.82668 ])
```

step 3: visualising the data's

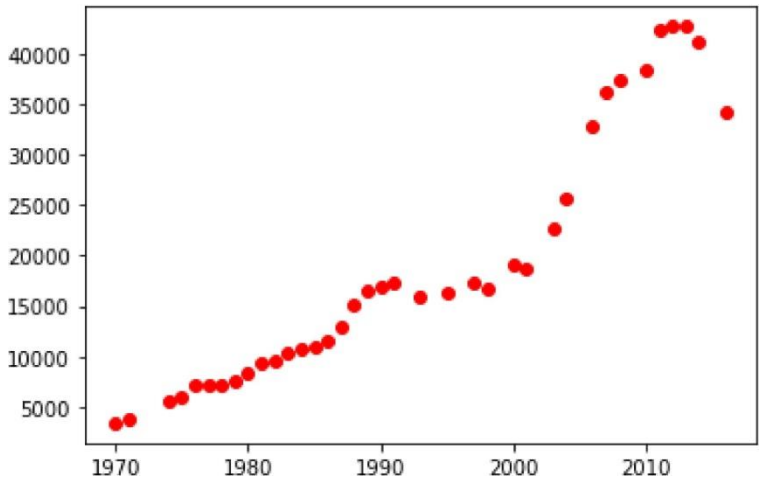
step a: training data

In [18]:

```
1 plt.scatter(atrain,btrain,color='red')
```

Out[18]:

```
<matplotlib.collections.PathCollection at 0x13a97e60e50>
```

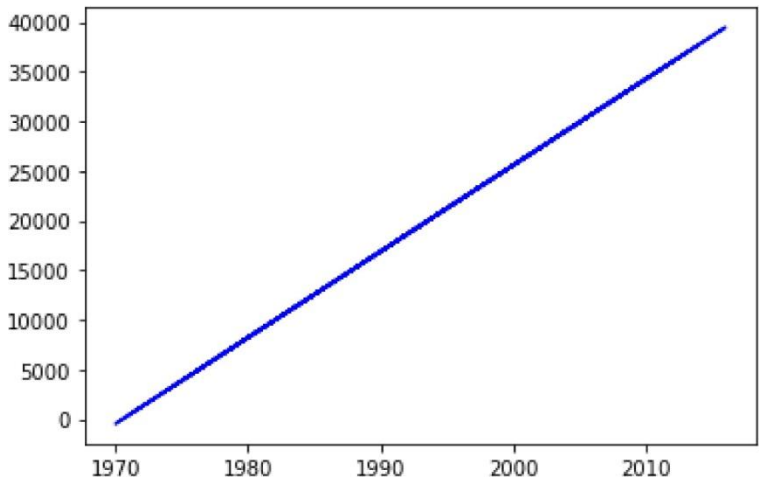


In [19]:

```
1 plt.plot(atrain,LR.predict(atrain),color='blue')
```

Out[19]:

```
[<matplotlib.lines.Line2D at 0x13a97f49c70>]
```

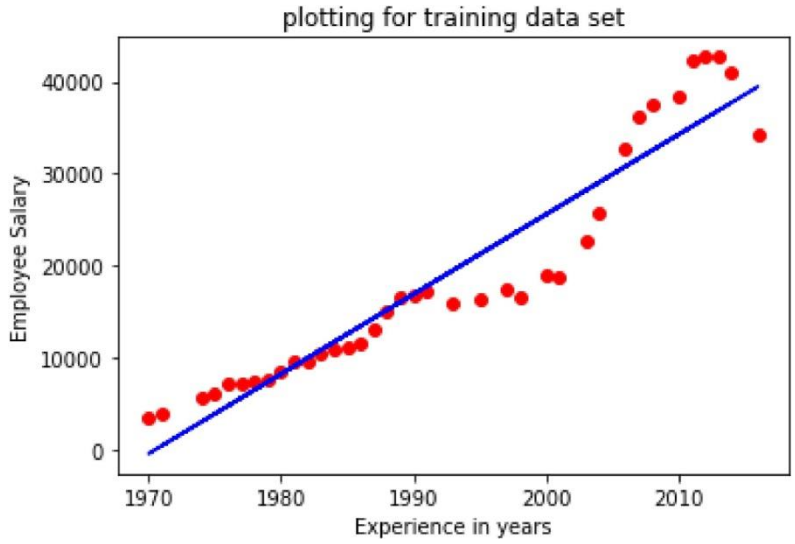


In [20]:

```
plt.scatter(atrain,btrain,color='red')
plt.plot(atrain,LR.predict(atrain),color='blue')
plt.xlabel('Experience in years')
plt.ylabel('Employee Salary')
plt.title('plotting for training data set')
```

Out[20]:

```
Text(0.5, 1.0, 'plotting for training data set')
```



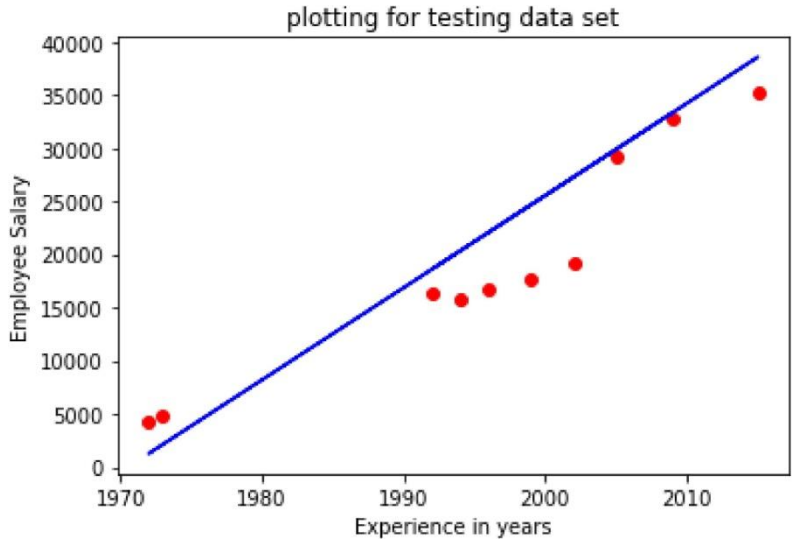
step b: testing data

In [21]:

```
plt.scatter(atest,btest,color='red')
plt.plot(atest,LR.predict(atest),color='blue')
plt.xlabel('Experience in years')
plt.ylabel('Employee Salary')
plt.title('plotting for testing data set')
```

Out[21]:

```
Text(0.5, 1.0, 'plotting for testing data set')
```



In [22]:

```
LR.coef_
```

Out[22]:

```
array([868.22718531])
```

In [23]:

```
LR.intercept_
```

Out[23]:

```
-1710895.0617872213
```

For the given predict the per capitia income for Canadian citizens in year 2021.

In [24]:

```
1 LR.predict([[5]])
```

Out[24]:

```
array([-1706553.92586066])
```

In [ ]:

```
1
```