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

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

```
import numpy as np
import pandas as pd
```

```
dataset = pd.read_csv("/content/drive/MyDrive/50_Startups.csv")
```

```
dataset
```

	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
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	NaN	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	NaN	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37
18	91749.16	114175.79	294919.57	Florida	124266.90
19	86419.70	153514.11	0.00	New York	122776.86
20	76253.86	113867.30	298664.47	California	118474.03
21	78389.47	153773.43	299737.29	New York	111313.02
22	73994.56	122782.75	303319.26	Florida	110352.25
23	67532.53	105751.03	304768.73	Florida	108733.99
24	77044.01	99281.34	140574.81	New York	108552.04
25	64664.71	139553.16	137962.62	California	107404.34
26	75328.87	144135.98	134050.07	Florida	105733.54
27	72107.60	127864.55	353183.81	New York	105008.31

```
dataset.isnull().any()
```

```
R&D Spend          False
Administration      True
Marketing Spend     True
State              False
Profit             False
dtype: bool
```

```
dataset.isnull().sum()
```

```
R&D Spend          0
Administration     1
Marketing Spend    1
State              0
Profit             0
dtype: int64
```

```
dataset.dtypes
```

```
R&D Spend          float64
Administration     float64
Marketing Spend    float64
State               object
Profit             float64
dtype: object
```

```
dataset.columns = map(str.lower,dataset.columns)
```

```
dataset.dtypes
```

```
r&d spend          float64
administration     float64
marketing spend    float64
state               object
profit             float64
dtype: object
```

```
dataset["administration"].fillna(dataset["administration"].mean(),inplace = True)
dataset["marketing spend"].fillna(dataset["marketing spend"].mean(),inplace = True)
```

```
dataset.isnull().any()
```

```
r&d spend          False
administration     False
marketing spend    False
state              False
profit             False
dtype: bool
```

```
x=dataset.iloc[:,0:4]
```

```
y=dataset.iloc[:,4:]
```

```
x.head()
```

	r&d spend	administration	marketing spend	state
0	165349.20	136897.80	471784.10	New York
1	162597.70	151377.59	443898.53	California
2	153441.51	101145.55	407934.54	Florida
3	144372.41	118671.85	383199.62	New York
4	142107.34	91391.77	366168.42	Florida

```
y.head()
```

	profit
0	192261.83
1	191792.06
2	191050.39
3	182901.99
4	166187.94

```
dataset["state"].unique()
```

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

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
dataset["state"]=le.fit_transform(dataset["state"])
```

```
dataset["state"].head()
```

```
0    2
1    0
2    1
3    2
4    1
Name: state, dtype: int64
```

```
x=dataset.iloc[:,0:4].values
y=dataset.iloc[:,4:].values
```

```
print(type(x))
print(type(y))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test= train_test_split(x,y,test_size=0.3,random_state=1)
```

```
print(x_train.shape)
print(y_train.shape)
```

```
(35, 4)
(35, 1)
```

```
print(x_test.shape)
print(y_test.shape)
```

```
(15, 4)
(15, 1)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = StandardScaler().fit_transform(x_train)
x_test = StandardScaler().fit_transform(x_test)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
Regression_model = Sequential()
```

```
Regression_model.add(Dense(units=156,kernel_initializer="normal",activation="relu"))
```

```
Regression_model.add(Dense(units=180,kernel_initializer="normal",activation="relu"))
```

```
Regression_model.add(Dense(units=10,kernel_initializer="normal",activation="linear"))
```

```
Regression_model.compile(optimizer="adam",loss="mean_squared_error",metrics=['mean_squared_error'])
```

```
history=Regression_model.fit(x_train,y_train,epochs=100,batch_size=16,validation_data=(x_test,y_test))
```

```
3/3 [==============================] - 0s 16ms/step - loss: 15071994880.0000 - mean_squared_error: 15071994880.0000 - val_loss: 11845028864.0000 - val_mean_squared_error: 1184
Epoch 41/100
3/3 [==============================] - 0s 14ms/step - loss: 15068483584.0000 - mean_squared_error: 15068483584.0000 - val_loss: 11841739776.0000 - val_mean_squared_error: 1184
Epoch 42/100
3/3 [==============================] - 0s 14ms/step - loss: 15064713216.0000 - mean_squared_error: 15064713216.0000 - val_loss: 11838366720.0000 - val_mean_squared_error: 1183
Epoch 43/100
```

```
3/3 [==============================] - 0s 18ms/step - loss: 15060819968.0000 - mean_squared_error: 15060819968.0000 - val_loss: 11834776576.0000 - val_mean_squared_error: 1183
Epoch 44/100
3/3 [==============================] - 0s 15ms/step - loss: 15056397312.0000 - mean_squared_error: 15056397312.0000 - val_loss: 11831014400.0000 - val_mean_squared_error: 1183
Epoch 45/100
3/3 [==============================] - 0s 17ms/step - loss: 15052041216.0000 - mean_squared_error: 15052041216.0000 - val_loss: 11827042304.0000 - val_mean_squared_error: 1182
Epoch 46/100
3/3 [==============================] - 0s 18ms/step - loss: 15047549952.0000 - mean_squared_error: 15047549952.0000 - val_loss: 11822758912.0000 - val_mean_squared_error: 1182
Epoch 47/100
3/3 [==============================] - 0s 13ms/step - loss: 15042450432.0000 - mean_squared_error: 15042450432.0000 - val_loss: 11818211328.0000 - val_mean_squared_error: 1181
Epoch 48/100
3/3 [==============================] - 0s 13ms/step - loss: 15036965888.0000 - mean_squared_error: 15036965888.0000 - val_loss: 11813439488.0000 - val_mean_squared_error: 1181
Epoch 49/100
3/3 [==============================] - 0s 14ms/step - loss: 15031589888.0000 - mean_squared_error: 15031589888.0000 - val_loss: 11808419840.0000 - val_mean_squared_error: 1180
Epoch 50/100
3/3 [==============================] - 0s 13ms/step - loss: 15025733632.0000 - mean_squared_error: 15025733632.0000 - val_loss: 11803257856.0000 - val_mean_squared_error: 1180
Epoch 51/100
3/3 [==============================] - 0s 12ms/step - loss: 15019418624.0000 - mean_squared_error: 15019418624.0000 - val_loss: 11797902336.0000 - val_mean_squared_error: 1179
Epoch 52/100
3/3 [==============================] - 0s 13ms/step - loss: 15013342208.0000 - mean_squared_error: 15013342208.0000 - val_loss: 11792229376.0000 - val_mean_squared_error: 1179
Epoch 53/100
3/3 [==============================] - 0s 13ms/step - loss: 15006846976.0000 - mean_squared_error: 15006846976.0000 - val_loss: 11786216448.0000 - val_mean_squared_error: 1178
Epoch 54/100
3/3 [==============================] - 0s 23ms/step - loss: 14999581696.0000 - mean_squared_error: 14999581696.0000 - val_loss: 11780056064.0000 - val_mean_squared_error: 1178
Epoch 55/100
3/3 [==============================] - 0s 28ms/step - loss: 14992604160.0000 - mean_squared_error: 14992604160.0000 - val_loss: 11773652992.0000 - val_mean_squared_error: 1177
Epoch 56/100
3/3 [==============================] - 0s 17ms/step - loss: 14984464384.0000 - mean_squared_error: 14984464384.0000 - val_loss: 11766841344.0000 - val_mean_squared_error: 1176
Epoch 57/100
3/3 [==============================] - 0s 14ms/step - loss: 14976399360.0000 - mean_squared_error: 14976399360.0000 - val_loss: 11759242240.0000 - val_mean_squared_error: 1175
Epoch 58/100
3/3 [==============================] - 0s 13ms/step - loss: 14968087552.0000 - mean_squared_error: 14968087552.0000 - val_loss: 11751190528.0000 - val_mean_squared_error: 1175
Epoch 59/100

3/3 [==============================] - 0s 14ms/step - loss: 14958507008.0000 - mean_squared_error: 14958507008.0000 - val_loss: 11743009792.0000 - val_mean_squared_error: 1174
Epoch 60/100
3/3 [==============================] - 0s 12ms/step - loss: 14948954112.0000 - mean_squared_error: 14948954112.0000 - val_loss: 11734777856.0000 - val_mean_squared_error: 1173
Epoch 61/100
3/3 [==============================] - 0s 13ms/step - loss: 14938909696.0000 - mean_squared_error: 14938909696.0000 - val_loss: 11726498816.0000 - val_mean_squared_error: 1172
Epoch 62/100
3/3 [==============================] - 0s 14ms/step - loss: 14929488896.0000 - mean_squared_error: 14929488896.0000 - val_loss: 11717782528.0000 - val_mean_squared_error: 1171
Epoch 63/100
3/3 [==============================] - 0s 13ms/step - loss: 14919208960.0000 - mean_squared_error: 14919208960.0000 - val_loss: 11708840960.0000 - val_mean_squared_error: 1170
Epoch 64/100
3/3 [==============================] - 0s 14ms/step - loss: 14909065216.0000 - mean_squared_error: 14909065216.0000 - val_loss: 11699847168.0000 - val_mean_squared_error: 1169
Epoch 65/100
3/3 [==============================] - 0s 18ms/step - loss: 14897938432.0000 - mean_squared_error: 14897938432.0000 - val_loss: 11690575872.0000 - val_mean_squared_error: 1169
Epoch 66/100
3/3 [==============================] - 0s 16ms/step - loss: 14886890496.0000 - mean_squared_error: 14886890496.0000 - val_loss: 11680691200.0000 - val_mean_squared_error: 1168
Epoch 67/100
3/3 [==============================] - 0s 14ms/step - loss: 14875598848.0000 - mean_squared_error: 14875598848.0000 - val_loss: 11670521856.0000 - val_mean_squared_error: 1167
Epoch 68/100
3/3 [==============================] - 0s 12ms/step - loss: 14863248384.0000 - mean_squared_error: 14863248384.0000 - val_loss: 11660068864.0000 - val_mean_squared_error: 1166
Epoch 69/100
```

```
Regression_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 156)               780
_________________________________________________________________
dense_4 (Dense)              (None, 180)               28260
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1810
=================================================================
Total params: 30,850
Trainable params: 30,850
Non-trainable params: 0
_________________________________________________________________
```

```
prediction_y = Regression_model.predict(x_test)
```

```
prediction_y
```

```
array([[5640.5366, 5598.917 , 5743.7275, 5674.1665, 5611.9854, 5612.293 ,
        5599.4463, 5711.595 , 5612.8496, 5707.1733],
       [2063.8413, 2048.388 , 2101.4497, 2075.9644, 2052.9604, 2053.309 ,
        2048.872 , 2089.8965, 2053.6042, 2087.8157],
       [1570.4669, 1559.3727, 1599.6472, 1580.0753, 1562.5659, 1562.7133,
        1559.7443, 1590.6194, 1563.5194, 1589.276 ],
       [1468.232 , 1457.3815, 1495.1155, 1476.8384, 1460.3895, 1460.7404,
        1457.7534, 1486.9294, 1461.1282, 1485.313 ],
       [8886.886 , 8821.874 , 9049.692 , 8940.237 , 8842.247 , 8842.669 ,
        8822.508 , 8999.097 , 8843.717 , 8992.257 ],
       [8929.038 , 8863.262 , 9092.409 , 8982.295 , 8883.929 , 8884.391 ,
        8864.1   , 9041.567 , 8885.3545, 9034.624 ],
       [1320.2966, 1310.8057, 1344.7772, 1328.016 , 1313.2533, 1313.5596,
        1311.1556, 1337.3893, 1314.388 , 1335.8953],
       [2600.6174, 2581.3882, 2648.3257, 2616.068 , 2587.3804, 2587.5842,
        2581.72  , 2633.4338, 2587.9802, 2631.4114],
       [1680.1183, 1668.0358, 1711.1207, 1690.3022, 1671.5919, 1671.7229,
        1668.2927, 1701.5444, 1672.2946, 1700.095 ],
       [2242.8315, 2226.231 , 2284.003 , 2256.1565, 2231.3657, 2231.571 ,
        2226.5579, 2271.1497, 2231.9626, 2269.3745],
       [1560.3646, 1549.3193, 1589.3849, 1569.8431, 1552.5092, 1552.6611,
        1549.6826, 1580.291 , 1553.4868, 1579.0424],
       [1604.1743, 1592.6697, 1633.8501, 1613.9689, 1595.9529, 1596.1029,
        1593.264 , 1624.8297, 1596.979 , 1623.2715],
       [5620.486 , 5579.0776, 5723.421 , 5654.0435, 5592.149 , 5592.4126,
        5579.555 , 5691.326 , 5592.977 , 5687.0684],
       [1666.5636, 1654.5983, 1697.3627, 1676.6582, 1658.1544, 1658.2747,
        1654.8368, 1687.7953, 1658.8656, 1686.4479],
       [2179.9685, 2163.8235, 2220.0027, 2192.9043, 2168.8   , 2169.028 ,
        2164.1914, 2207.5098, 2169.4438, 2205.7688]], dtype=float32)
```

```
sc = StandardScaler()
sc.fit(x)
x = sc.transform(x)
```

```
x = sc.transform(x)
yp = Regression_model.predict(sc.transform([[165349.20, 136897.80,  471784.10,  2]]))
```

```
print(yp)
```

```
[[10703.729 10624.798 10899.571 10767.504 10649.656 10650.171 10625.83
  10838.596 10651.324 10830.326]]
```

```
sc.transform([[165349.20, 136897.80,  471784.10,  2]])
```

```
array([[2.01641149, 0.55369219, 2.15031508, 1.21267813]])
```

✓ 0s completed at 18:33