This classic dataset contains the prices and other attributes of almost 54,000 diamonds. Perform single neuron regression to predict the diamond price.

price price in US dollars (\$326--\$18,823)

carat weight of the diamond (0.2--5.01)

cut quality of the cut (Fair, Good, Very Good, Premium, Ideal)

color diamond colour, from J (worst) to D (best)

clarity a measurement of how clear the diamond is (I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best))

x length in mm (0--10.74)

y width in mm (0--58.9)

z depth in mm (0--31.8)

depth total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)

table width of the top of the diamond relative to widest point (43--95)

In [20]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

In [3]:

```
df=pd.read_csv("diamonds.csv")
```

In [4]:

```
df
```

Out[4]:

	Unnamed: 0	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...	...
53935	53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 11 columns

In [5]:

```
df.drop(['Unnamed: 0'],axis=1,inplace=True)
```

In [6]:

```
df
```

Out[6]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53935	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [7]:

```
df.insert(0, 'ID', range(1, 1 + len(df)))
df.set_index("ID",inplace=True)
```

In [8]:

```
df
```

Out[8]:

ID	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
4	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
5	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...	...	...	...
53936	0.72	Ideal	D	SI1	60.8	57.0	2757	5.75	5.76	3.50
53937	0.72	Good	D	SI1	63.1	55.0	2757	5.69	5.75	3.61
53938	0.70	Very Good	D	SI1	62.8	60.0	2757	5.66	5.68	3.56
53939	0.86	Premium	H	SI2	61.0	58.0	2757	6.15	6.12	3.74
53940	0.75	Ideal	D	SI2	62.2	55.0	2757	5.83	5.87	3.64

53940 rows × 10 columns

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53940 entries, 1 to 53940
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object
 2   color    53940 non-null  object
 3   clarity  53940 non-null  object
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.5+ MB
```

In [10]:

```
df_num=df.select_dtypes(['int64','float64'])
```

In [11]:

```
df_cat=df.select_dtypes(['object'])
```

In [12]:

```
from scipy.stats import skew
```

In [13]:

```
# Skew = 3 * (Mean – Median) / Standard Deviation.
for i in df_num:
    print(i,skew(df_num[i]))

#data skewness > -1 and < 1 that means data is normally distributed
```

```
carat 1.1166148681277797
depth -0.08229173779627727
table 0.7968736878796518
price 1.6183502776053016
x 0.3786658120772097
y 2.4340990250113648
z 1.5223802221853722
```

In [14]:

```
from sklearn.preprocessing import LabelEncoder
```

In [15]:

```
le=LabelEncoder()
for col in df_cat:
    df_cat[col]=le.fit_transform(df_cat[col])
```

In [16]:

```
df_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 53940 entries, 1 to 53940
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   cut      53940 non-null  int32
 1   color    53940 non-null  int32
 2   clarity  53940 non-null  int32
dtypes: int32(3)
memory usage: 1.0 MB
```

In [17]:

```
df_cat
```

Out[17]:

	cut	color	clarity
ID			
1	2	1	3
2	3	1	2
3	1	1	4
4	3	5	5
5	1	6	3
...	...	...	...
53936	2	0	2
53937	1	0	2
53938	4	0	2
53939	3	4	3
53940	2	0	3

53940 rows × 3 columns

In [18]:

```
df_new=pd.merge(df_num,df_cat,on="ID")
```

In [19]:

```
df_new
```

Out[19]:

ID	carat	depth	table	price	x	y	z	cut	color	clarity
1	0.23	61.5	55.0	326	3.95	3.98	2.43	2	1	3
2	0.21	59.8	61.0	326	3.89	3.84	2.31	3	1	2
3	0.23	56.9	65.0	327	4.05	4.07	2.31	1	1	4
4	0.29	62.4	58.0	334	4.20	4.23	2.63	3	5	5
5	0.31	63.3	58.0	335	4.34	4.35	2.75	1	6	3
...	...	...	...	...	...	...	...	...	...	...
53936	0.72	60.8	57.0	2757	5.75	5.76	3.50	2	0	2
53937	0.72	63.1	55.0	2757	5.69	5.75	3.61	1	0	2
53938	0.70	62.8	60.0	2757	5.66	5.68	3.56	4	0	2
53939	0.86	61.0	58.0	2757	6.15	6.12	3.74	3	4	3
53940	0.75	62.2	55.0	2757	5.83	5.87	3.64	2	0	3

53940 rows × 10 columns

In [21]:

```
x=df_new.drop("price",axis=1)
y=df_new["price"]
```

In [42]:

```
# Scaling the data
x_mean = np.mean(x)
```

In [43]:

```
x_mean
```

Out[43]:

```
carat       0.797940
depth      61.749405
table      57.457184
x           5.731157
y           5.734526
z           3.538734
cut         2.553003
color       2.594197
clarity     3.835150
dtype: float64
```

In [44]:

```
x_std = np.std(x)
```

In [45]:

```
x_std
```

Out[45]:

```
carat      0.474007
depth      1.432608
table      2.234470
x          1.121750
y          1.142124
z          0.705692
cut        1.027698
color      1.701089
clarity    1.724575
dtype: float64
```

In [48]:

```
x_x = (x-x_mean) / x_std
```

In [49]:

```
x_x
```

Out[49]:

ID	carat	depth	table	x	y	z	cut	color
1	-1.198168	-0.174092	-1.099672	-1.587837	-1.536196	-1.571129	-0.538099	-0.937163
2	-1.240361	-1.360738	1.585529	-1.641325	-1.658774	-1.741175	0.434949	-0.937163
3	-1.198168	-3.385019	3.375663	-1.498691	-1.457395	-1.741175	-1.511147	-0.937163
4	-1.071587	0.454133	0.242928	-1.364971	-1.317305	-1.287720	0.434949	1.414272
5	-1.029394	1.082358	0.242928	-1.240167	-1.212238	-1.117674	-1.511147	2.002131
...	...	...	...	...	...	...	...	...
53936	-0.164427	-0.662711	-0.204605	0.016798	0.022304	-0.054888	-0.538099	-1.525021
53937	-0.164427	0.942753	-1.099672	-0.036690	0.013548	0.100988	-1.511147	-1.525021
53938	-0.206621	0.733344	1.137995	-0.063434	-0.047741	0.030135	1.407998	-1.525021
53939	0.130927	-0.523105	0.242928	0.373383	0.337506	0.285204	0.434949	0.826413
53940	-0.101137	0.314528	-1.099672	0.088115	0.118616	0.143499	-0.538099	-1.525021

53940 rows × 9 columns

In [50]:

```
x_train, x_test, y_train, y_test = train_test_split(x_x, y, test_size=0.3)
```

In [51]:

```
model = tf.keras.Sequential([tf.keras.layers.Dense(1, input_shape=(9,))])
```

In [52]:

```
model.compile(optimizer="sgd", loss="mse")
```

In [53]:

```
trained_model = model.fit(x_train, y_train, epochs=50)
```

```
Epoch 1/50
1180/1180 [==============================] - 1s 567us/step - loss: 4436778.4
041
Epoch 2/50
1180/1180 [==============================] - 1s 563us/step - loss: 1911444.9
264
Epoch 3/50
1180/1180 [==============================] - 1s 533us/step - loss: 1936684.9
781
Epoch 4/50
1180/1180 [==============================] - 1s 536us/step - loss: 1818138.3
090
Epoch 5/50
1180/1180 [==============================] - 1s 563us/step - loss: 1833496.5
618
Epoch 6/50
1180/1180 [==============================] - 1s 578us/step - loss: 1823795.9
470
Epoch 7/50
1180/1180 [==============================] - 1s 567us/step - loss: 1746740.7
430
Epoch 8/50
1180/1180 [==============================] - 1s 624us/step - loss: 1834848.8
347
Epoch 9/50
1180/1180 [==============================] - 1s 580us/step - loss: 1784011.5
848
Epoch 10/50
1180/1180 [==============================] - 1s 621us/step - loss: 1831464.1
083
Epoch 11/50
1180/1180 [==============================] - 1s 567us/step - loss: 1785456.6
213
Epoch 12/50
1180/1180 [==============================] - 1s 577us/step - loss: 1887061.7
530
Epoch 13/50
1180/1180 [==============================] - 1s 567us/step - loss: 1870181.1
311
Epoch 14/50
1180/1180 [==============================] - 1s 607us/step - loss: 1791370.8
515
Epoch 15/50
1180/1180 [==============================] - 1s 533us/step - loss: 1825887.4
636
Epoch 16/50
1180/1180 [==============================] - 1s 543us/step - loss: 1769436.5
620
Epoch 17/50
1180/1180 [==============================] - 1s 536us/step - loss: 1830362.8
243
Epoch 18/50
1180/1180 [==============================] - 1s 543us/step - loss: 1880613.8
246
Epoch 19/50
1180/1180 [==============================] - 1s 556us/step - loss: 1825285.1
```

```
722
Epoch 20/50
1180/1180 [==============================] - 1s 543us/step - loss: 1807044.7
478
Epoch 21/50
1180/1180 [==============================] - 1s 529us/step - loss: 1811000.4
961
Epoch 22/50
1180/1180 [==============================] - 1s 584us/step - loss: 1812599.9
052
Epoch 23/50
1180/1180 [==============================] - 1s 570us/step - loss: 1769915.1
559
Epoch 24/50
1180/1180 [==============================] - 1s 550us/step - loss: 1856408.4
255
Epoch 25/50
1180/1180 [==============================] - 1s 570us/step - loss: 1851703.5
313
Epoch 26/50
1180/1180 [==============================] - 1s 577us/step - loss: 1791187.7
117
Epoch 27/50
1180/1180 [==============================] - 1s 553us/step - loss: 1819278.1
862
Epoch 28/50
1180/1180 [==============================] - 1s 536us/step - loss: 1739057.5
928
Epoch 29/50
1180/1180 [==============================] - 1s 594us/step - loss: 1879389.5
278
Epoch 30/50
1180/1180 [==============================] - 1s 612us/step - loss: 1761444.4
372
Epoch 31/50
1180/1180 [==============================] - 1s 584us/step - loss: 1793397.4
790
Epoch 32/50
1180/1180 [==============================] - 1s 624us/step - loss: 1845452.3
546
Epoch 33/50
1180/1180 [==============================] - 1s 570us/step - loss: 1884488.8
819
Epoch 34/50
1180/1180 [==============================] - 1s 604us/step - loss: 1814700.3
371
Epoch 35/50
1180/1180 [==============================] - 1s 556us/step - loss: 1786090.4
794
Epoch 36/50
1180/1180 [==============================] - 1s 539us/step - loss: 1860553.4
079
Epoch 37/50
1180/1180 [==============================] - 1s 543us/step - loss: 1898835.5
630
Epoch 38/50
1180/1180 [==============================] - 1s 587us/step - loss: 1852524.2
324
Epoch 39/50
1180/1180 [==============================] - 1s 536us/step - loss: 1808252.7
326
```

```
Epoch 40/50
1180/1180 [==============================] - 1s 621us/step - loss: 1815245.8
864
Epoch 41/50
1180/1180 [==============================] - 1s 550us/step - loss: 1850231.7
570
Epoch 42/50
1180/1180 [==============================] - 1s 539us/step - loss: 1886358.5
570
Epoch 43/50
1180/1180 [==============================] - 1s 543us/step - loss: 1834787.6
779
Epoch 44/50
1180/1180 [==============================] - 1s 536us/step - loss: 1760127.2
574
Epoch 45/50
1180/1180 [==============================] - 1s 546us/step - loss: 1882329.5
883
Epoch 46/50
1180/1180 [==============================] - 1s 546us/step - loss: 1900655.9
566
Epoch 47/50
1180/1180 [==============================] - 1s 597us/step - loss: 1866726.5
197
Epoch 48/50
1180/1180 [==============================] - 1s 621us/step - loss: 1755582.2
546
Epoch 49/50
1180/1180 [==============================] - 1s 607us/step - loss: 1744955.9
602
Epoch 50/50
1180/1180 [==============================] - 1s 590us/step - loss: 1828544.4
806
```

In [54]:

```
trained_model
```

Out[54]:

```
<tensorflow.python.keras.callbacks.History at 0x236b99fae50>
```

In [55]:

```
trained_model.history
```

Out[55]:

```
{'loss': [2686543.75,
  1915283.0,
  1865870.625,
  1846135.875,
  1831995.125,
  1833070.75,
  1830074.375,
  1829699.625,
  1828531.125,
  1829416.5,
  1827082.625,
  1830909.875,
  1830014.625,
  1829468.375,
  1830789.625,
  1829044.875,
  1829045.5,
  1829282.625,
  1828484.375,
  1829961.5,
  1828917.0,
  1827854.375,
  1829776.625,
  1831058.5,
  1827825.75,
  1827868.75,
  1828065.0,
  1829183.25,
  1828250.375,
  1831711.5,
  1828974.5,
  1828545.625,
  1829152.875,
  1828933.125,
  1828154.0,
  1828641.25,
  1829051.375,
  1829966.0,
  1828489.375,
  1830411.875,
  1829867.5,
  1830227.75,
  1827748.25,
  1827374.0,
  1829355.75,
  1828244.625,
  1831807.625,
  1832429.0,
  1825595.0,
  1828953.75]}
```

In [56]:

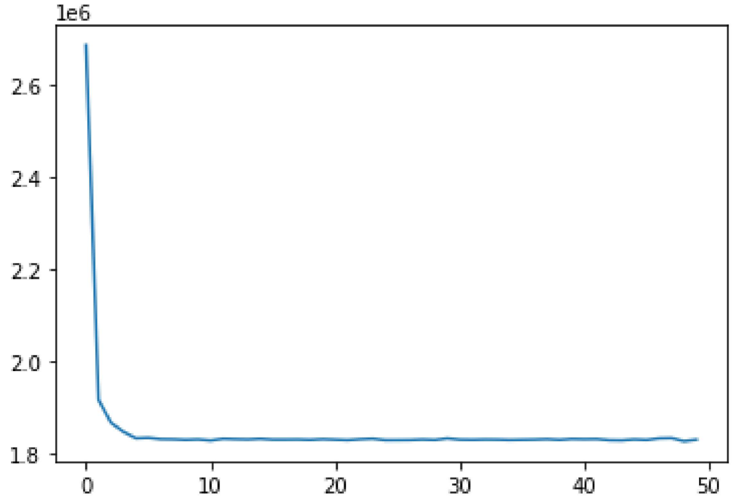
```
trained_model.history['loss']
```

Out[56]:

```
[2686543.75,
 1915283.0,
 1865870.625,
 1846135.875,
 1831995.125,
 1833070.75,
 1830074.375,
 1829699.625,
 1828531.125,
 1829416.5,
 1827082.625,
 1830909.875,
 1830014.625,
 1829468.375,
 1830789.625,
 1829044.875,
 1829045.5,
 1829282.625,
 1828484.375,
 1829961.5,
 1828917.0,
 1827854.375,
 1829776.625,
 1831058.5,
 1827825.75,
 1827868.75,
 1828065.0,
 1829183.25,
 1828250.375,
 1831711.5,
 1828974.5,
 1828545.625,
 1829152.875,
 1828933.125,
 1828154.0,
 1828641.25,
 1829051.375,
 1829966.0,
 1828489.375,
 1830411.875,
 1829867.5,
 1830227.75,
 1827748.25,
 1827374.0,
 1829355.75,
 1828244.625,
 1831807.625,
 1832429.0,
 1825595.0,
 1828953.75]
```

In [57]:

```
plt.plot(trained_model.history['loss'])
plt.show()
```



In [58]:

```
y_hat = model.predict(x_test)
```

In [59]:

```
y_hat
```

Out[59]:

```
array([[ 461.8523 ],
       [ 876.7219 ],
       [6185.9927 ],
       ...,
       [5807.62   ],
       [ 776.26196],
       [-553.1023 ]], dtype=float32)
```

In [60]:

```
from sklearn.metrics import r2_score
```

In [61]:

```
r2_score(y_test, y_hat)
```

Out[61]:

```
0.8823440920404613
```

In [ ]:

```

```