Diamond Price Prediction using Regression Algorithm

In [1]:
```
# Import Libraries
import pandas as pd # to handle and manipulate data efficiently
import numpy as np # to perform mathematical operations and handle numerical data effici
import matplotlib.pyplot as plt # to visualize data and create plots
import warnings # to manage and control warning messages
import pandas.util.testing as tm # to access utility functions for testing and data gene
```

```
C:\Users\A\AppData\Local\Temp\ipykernel_8460\3197914584.py:6: FutureWarning: pandas.uti
l.testing is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm # to access utility functions for testing and data ge
neration
```

In [2]:
```
# Load the dataset
data = pd.read_csv("Diamonds.csv")
```

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

Out[3]:

	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

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

Out[4]:

	carat	cut	color	clarity	depth	table	price	x	y	z
19994	1.04	Premium	D	VS1	60.5	59.0	8532	6.62	6.58	3.99
19995	1.22	Good	G	VS2	59.9	61.0	8533	6.88	6.91	4.13
19996	1.54	Very Good	I	VS2	62.0	58.0	8537	7.38	7.43	4.59
19997	1.05	Very Good	F	VVS2	61.3	59.0	8537	6.48	6.56	4.00
19998	1.57	Very Good	J	VS2	62.0	58.0	8538	7.45	7.48	4.63

In [5]:
```
data.shape
```

Out[5]:
```
(19999, 10)
```

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

Out[6]:

	carat	depth	table	price	x	y	z
count	19999.000000	19999.000000	19999.000000	19999.000000	19999.000000	19999.000000	19999.000000
mean	0.959146	61.826316	57.766683	4530.741137	6.229036	6.228317	3.849233
std	0.299008	1.560031	2.227478	1950.501618	0.763669	0.752254	0.478376
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.820000	61.000000	56.000000	3379.000000	6.000000	6.000000	3.710000
50%	1.010000	61.900000	58.000000	4496.000000	6.390000	6.390000	3.960000
75%	1.110000	62.700000	59.000000	5851.500000	6.660000	6.650000	4.100000
max	3.010000	71.800000	70.000000	8538.000000	9.230000	9.100000	5.970000

In [7]:

```
data.isnull().sum() # to check null values in dataset
```

Out[7]:

```
carat      0
cut        0
color      0
clarity    0
depth      0
table      0
price      0
x          0
y          0
z          0
dtype: int64
```

In [8]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19999 entries, 0 to 19998
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    19999 non-null  float64
 1   cut      19999 non-null  object 
 2   color    19999 non-null  object 
 3   clarity  19999 non-null  object 
 4   depth    19999 non-null  float64
 5   table    19999 non-null  float64
 6   price    19999 non-null  int64  
 7   x        19999 non-null  float64
 8   y        19999 non-null  float64
 9   z        19999 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 1.5+ MB
```

In [9]:

```
data = data.drop(['depth','table','x','y','z'],axis = 1) # Drop the columns
```

In [10]:

```
data.head()
```

Out[10]:

	carat	cut	color	clarity	price
0	0.23	Ideal	E	SI2	326
1	0.21	Premium	E	SI1	326
2	0.23	Good	E	VS1	327
3	0.29	Premium	I	VS2	334
4	0.31	Good	J	SI2	335

In [11]:
```
data.dtypes # check the Datatype
```
Out[11]:
```
carat      float64
cut         object
color       object
clarity     object
price        int64
dtype: object
```

In [12]:
```
data['price'] = data.price.astype(float) # convert into float datatype
data.dtypes
```
Out[12]:
```
carat      float64
cut         object
color       object
clarity     object
price      float64
dtype: object
```

Data Visualization

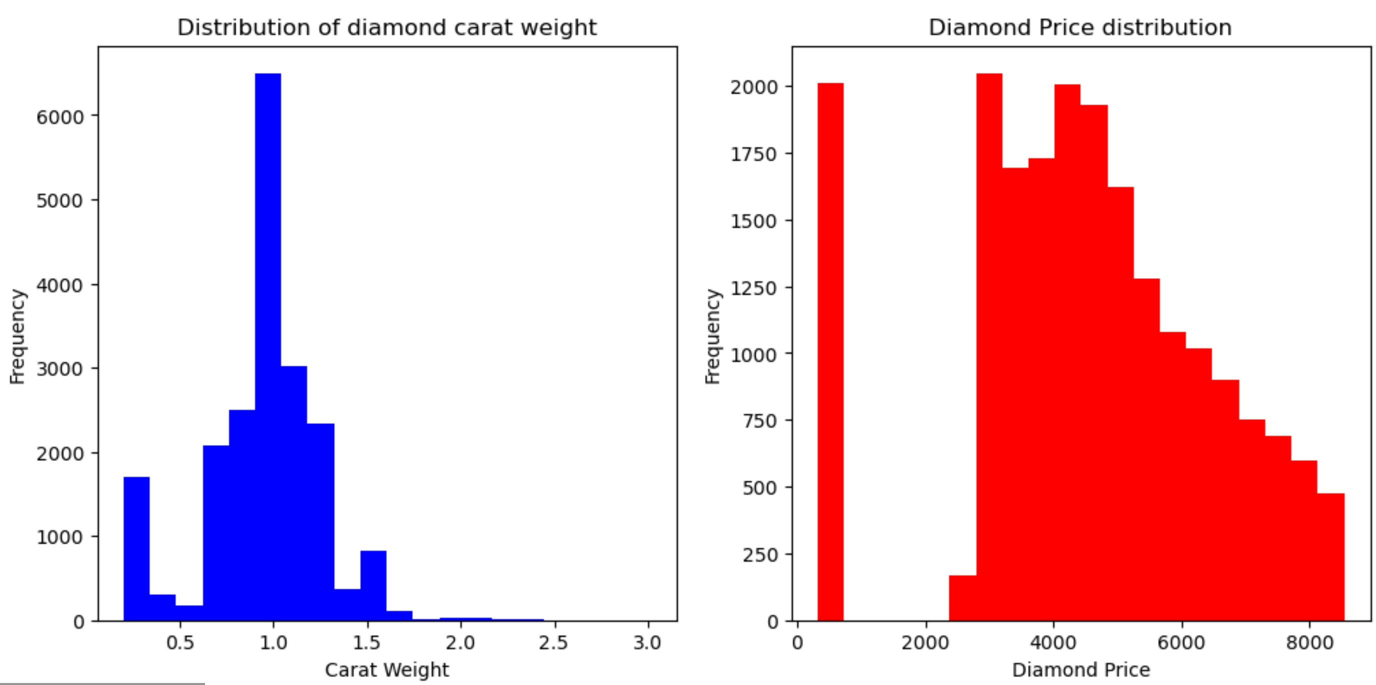
In [13]:
```
plt.figure(figsize=[12,12])
plt.subplot(221)

# carat weight distribution

plt.hist(data['carat'],bins = 20, color = 'b')
plt.xlabel('Carat Weight')
plt.ylabel('Frequency')
plt.title('Distribution of diamond carat weight')
plt.subplot(222)

# Distribution of price value

plt.hist(data['price'],bins = 20, color = 'r')
plt.xlabel('Diamond Price')
plt.ylabel('Frequency')
plt.title('Diamond Price distribution')
```
Out[13]:
```
Text(0.5, 1.0, 'Diamond Price distribution')
```



Create Independent and Dependent Variable

In [14]:
```
data.head(1)
```

Out[14]:

	carat	cut	color	clarity	price
0	0.23	Ideal	E	SI2	326.0

Label Encoder converting categorical data into numeric form

In [15]:
```
from sklearn.preprocessing import LabelEncoder
l1 = LabelEncoder()
label = l1.fit_transform(data['cut'])
l1.classes_
```

Out[15]:
```
array(['Fair', 'Good', 'Ideal', 'Premium', 'Very Good'], dtype=object)
```

In [16]:
```
label
```

Out[16]:
```
array([2, 3, 1, ..., 4, 4, 4])
```

In [17]:
```
data['cut_label'] = label
```

In [18]:
```
data.head(2)
```

Out[18]:

	carat	cut	color	clarity	price	cut_label
0	0.23	Ideal	E	SI2	326.0	2
1	0.21	Premium	E	SI1	326.0	3

In [19]:
```
l2 = LabelEncoder()
label1 = l2.fit_transform(data['clarity'])
data['clarity_label'] = label1
data.head(2)
```

Out[19]:

	carat	cut	color	clarity	price	cut_label	clarity_label
0	0.23	Ideal	E	SI2	326.0	2	3
1	0.21	Premium	E	SI1	326.0	3	2

In [20]:
```
data['color'] = data['color'].map({'D':1, 'E':2, 'F':3, 'G':4, 'H':5, 'I':6, 'J':7, 'NA'
```

In [21]:
```
data['color'].fillna(0)
```

Out[21]:
```
0        2
1        2
2        2
3        6
4        7
        ..
19994    1
19995    4
19996    6
19997    3
19998    7
Name: color, Length: 19999, dtype: int64
```

In [22]:
```
data['color'].isnull().sum() # check the null values
```

Out[22]:
```
0
```

In [23]:
```
data.head(2)
```

Out[23]:

	carat	cut	color	clarity	price	cut_label	clarity_label
0	0.23	Ideal	2	SI2	326.0	2	3
1	0.21	Premium	2	SI1	326.0	3	2

Create Dependent and independent variable

In [24]:
```
y = data['price']
y.head(1)
```

Out[24]:
```
0    326.0
Name: price, dtype: float64
```

In [25]:
```
x = data.drop(['price','cut','clarity'],axis = 1) # Drop the columns
x.head(1)
```

Out[25]:

	carat	color	cut_label	clarity_label
0	0.23	2	2	3

Training Dataset

In [26]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size = 0.8, random_state = 42
```

In [27]:
```
len(x_train)
```

Out[27]:
```
15999
```

In [28]:
```
len(y_test)
```

Out[28]:
```
4000
```

In [29]:
```
len(data)
```

Out[29]:
```
19999
```

In [30]:
```
data.head()
```

Out[30]:

	carat	cut	color	clarity	price	cut_label	clarity_label
0	0.23	Ideal	2	SI2	326.0	2	3
1	0.21	Premium	2	SI1	326.0	3	2
2	0.23	Good	2	VS1	327.0	1	4
3	0.29	Premium	6	VS2	334.0	3	5
4	0.31	Good	7	SI2	335.0	1	3

In [31]:
```
data.tail()
```

Out[31]:

	carat	cut	color	clarity	price	cut_label	clarity_label
19994	1.04	Premium	1	VS1	8532.0	3	4
19995	1.22	Good	4	VS2	8533.0	1	5
19996	1.54	Very Good	6	VS2	8537.0	4	5
19997	1.05	Very Good	3	VVS2	8537.0	4	7
19998	1.57	Very Good	7	VS2	8538.0	4	5

StandardScaler Method

In [32]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.fit_transform(x_test)
```

Linear Regression Algorithm

In [33]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [34]:
```
lr.fit(x_train,y_train)
```

Out[34]:
```
▾ LinearRegression
LinearRegression()
```

In [35]:
```
lr.coef_ # Check the coeficient
```

Out[35]:
```
array([1878.61096893, -368.96142726,   78.14312299,  502.55848673])
```

In [36]:
```
lr.intercept_  # Check the intercept
```

Out[36]:
```
4524.568223013937
```

In [37]:
```
pred = lr.predict(x_test) # predict the value
pred
```

Out[37]: array([4026.60361274, 1109.43688469, 4581.09163288, ..., 4662.46375333,
       3063.51810551, 6991.29211876])

Accuracy score of LinearRegression model

In [38]:
```
from sklearn.metrics import r2_score
lr = r2_score(y_test,pred)
print(lr)
```

0.7659609464703355

Ridge Regression Algorithm

In [39]:
```
from sklearn.linear_model import Ridge
ridreg = Ridge()
ridreg.fit(x_train,y_train)
pred4 = ridreg.predict(x_test)
```

Accuracy score of Ridge

In [40]:
```
from sklearn.metrics import r2_score
rid = r2_score(y_test,pred4)
print(rid)
```

0.7659643630604536

Lasso Regression Algorithm

In [41]:
```
from sklearn.linear_model import Lasso
lassoreg = Lasso()
lassoreg.fit(x_train,y_train)
pred4 = lassoreg.predict(x_test)
```

Accuracy score of Lasso

In [42]:
```
lasso = r2_score(y_test,pred4)
print(lasso)
```

0.7660052764314913

DecisionTree Regressor Algorithm

In [43]:
```
from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor()
reg.fit(x_train,y_train)
pred1 = reg.predict(x_test)
```

Accuracy score of DecisionTree Regressor model

In [44]:
```
from sklearn.metrics import r2_score
dtr = r2_score(y_test,pred1)
print(dtr)
```

```
0.9205864481656103
```

RandomForest Regressor Algorithm

In [45]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 50)
rf.fit(x_train,y_train)
pred2 = rf.predict(x_test)
```

Accuracy score of RandomForest Regressor model

In [46]:
```
from sklearn.metrics import r2_score
rfr = r2_score(y_test,pred2)
print(rfr)
```

```
0.9314000413801156
```

Apply K-Fold

In [47]:
```
from sklearn.model_selection import cross_val_score, KFold
```

In [48]:
```
# Perform k-fold cross-validation on the training set
k = 5  # Specify the number of folds
scores = cross_val_score(rf, x_train, y_train, cv=k)
```

In [49]:
```
# Print the accuracy scores for each fold
print("Accuracy scores for each fold:", scores)
```

```
Accuracy scores for each fold: [0.94483399 0.94323322 0.94633343 0.94506675 0.94784615]
```

In [50]:
```
# Calculate the average accuracy across all folds
average_accuracy = scores.mean()
print("Average accuracy:", average_accuracy)
```

```
Average accuracy: 0.9454627068384811
```

Overall Accuracy Score

In [51]:
```
print("LinearRegression",lr)
print("Lasso Linear model",lasso)
print("Ridge Linear model",rid)
print("DecisionTreeregression",dtr)
print("RandomForestRegressor",rfr)
print("K-Fold Average accuracy",average_accuracy)
```

```
LinearRegression 0.7659609464703355
Lasso Linear model 0.7660052764314913
Ridge Linear model 0.7659643630604536
DecisionTreeregression 0.9205864481656103
RandomForestRegressor 0.9314000413801156
K-Fold Average accuracy 0.9454627068384811
```

Loading [MathJax]/extensions/Safe.js

Conclusion

So here we apply all Regression Algorithm one-by-one on availabel data. We get better accuracy in RandomForest Regression algorithm. It's 93.18%. When we apply RandomForest Regression algorithm on model for prediction we get better prediction as compare to other algorithm. I did apply scalling method on data, because when i use scaling data for algorithm that time i getting high accuracy as compare normal data.

Prediction Part

In [52]:
```
def prediction():
    carat = (input("Enter the value of carat:"))
    cut = int(input("Enter the value of cut:"))
    clarity = int(input("Enter the value of clarity:"))
    color = int(input("Enter the value of color:"))

    price = rf.predict([[carat,cut,clarity,color]])[0]*0.1

    print("Approximately Price of Diamond is:",price, 'Rs')

predi = prediction()
predi
```

```
Enter the value of carat:0.23
Enter the value of cut:2
Enter the value of clarity:3
Enter the value of color:2
Approximately Price of Diamond is: 484.0732666666667 Rs
```

In [ ]: