Price Prediction Of Diamonds: Regression

Importing Libraries

```
#All the libralies used in this project

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from xgboost import XGBRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn import metrics
```

Loading Data

```
data = pd.read_csv("diamonds.csv")
data.head()
```

	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

```
data.shape
```

```
(53940, 11)
```

Data Preprocessing

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 11 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  53940 non-null  int64
 1   carat       53940 non-null  float64
 2   cut         53940 non-null  object
 3   color       53940 non-null  object
 4   clarity     53940 non-null  object
 5   depth       53940 non-null  float64
 6   table       53940 non-null  float64
 7   price       53940 non-null  int64
 8   x           53940 non-null  float64
 9   y           53940 non-null  float64
 10  z           53940 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 4.5+ MB
```

The first column is an index ("Unnamed: 0") and thus we are going to remove it.

```
#The first column seems to be just index
data = data.drop(["Unnamed: 0"], axis=1)
data.describe()
```

	carat	depth	table	price	x	y	z
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	61.749405	57.457184	3932.799722	5.731157	5.734526	3.538734
std	0.474011	1.432621	2.234491	3989.439738	1.121761	1.142135	0.705699
min	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	0.400000	61.000000	56.000000	950.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5324.250000	6.540000	6.540000	4.040000
max	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

In [7]:

```
#Dropping dimentionless diamonds
data = data.drop(data[data["x"]==0].index)
data = data.drop(data[data["y"]==0].index)
data = data.drop(data[data["z"]==0].index)
data.shape
```

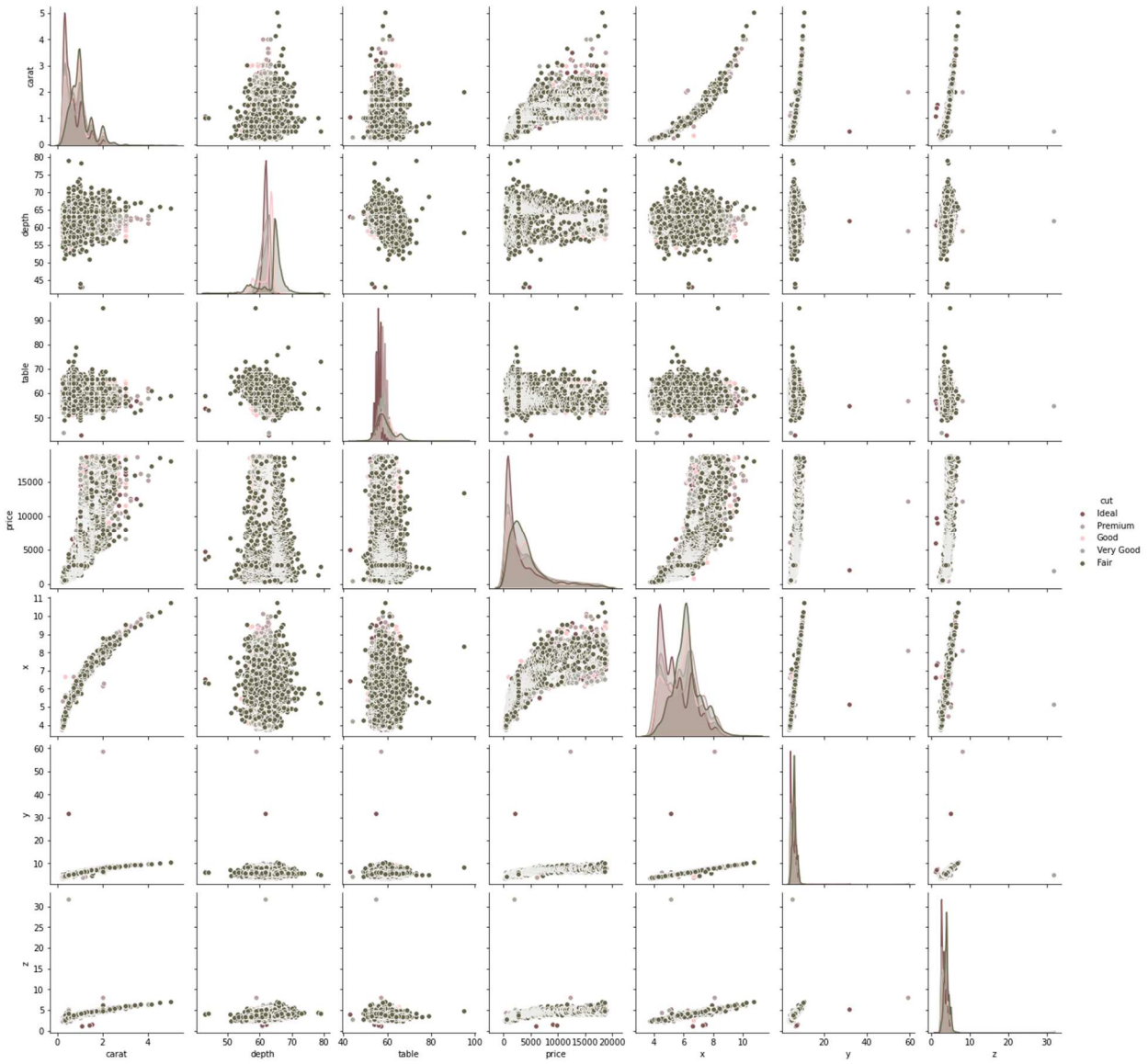
Out[7]:

```
(53920, 10)
```

Pairplot Of Data

In [8]:

```
#Let's have a look at data
shade = ["#835656", "#baa0a0", "#ffc7c8", "#a9a799", "#65634a"]#shades for
hue
ax = sns.pairplot(data, hue= "cut",palette=shade)# I chose "cut" as hue. We
can also examine other attributes in hue with less value count.
```

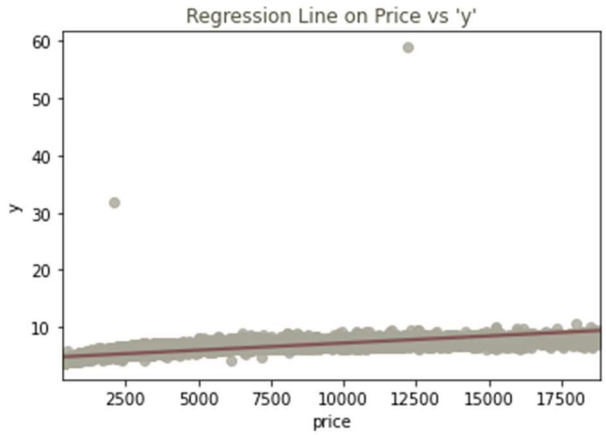


In [9]:

```
ax = sns.regplot(x="price", y="y", data=data, fit_reg=True,
scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'y'", color="#4e4c39")
```

Out[9]:

```
Text(0.5, 1.0, "Regression Line on Price vs 'y'")
```

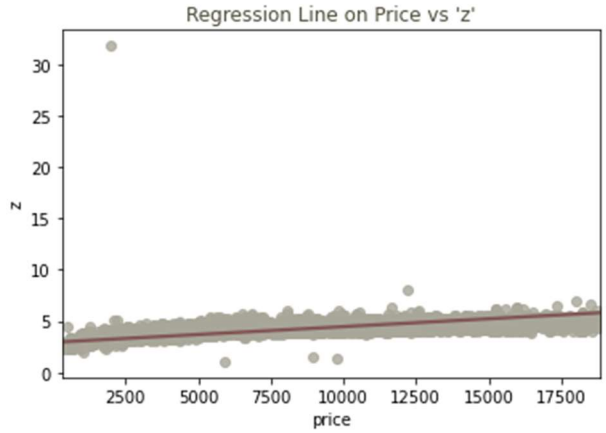


In [10]:

```
ax= sns.regplot(x="price", y="z", data=data, fit_reg=True,
scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs 'z'", color="#4e4c39")
```

Out[10]:

```
Text(0.5, 1.0, "Regression Line on Price vs 'z'")
```

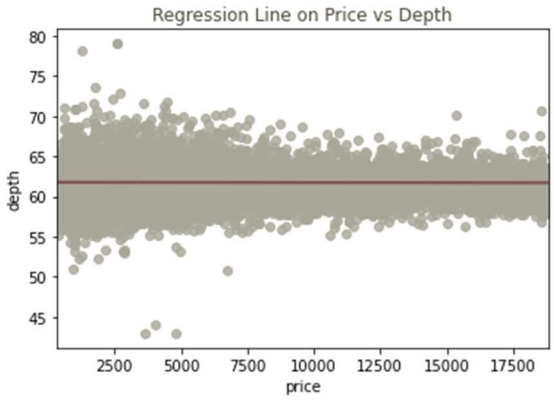


In [11]:

```
ax= sns.regplot(x="price", y="depth", data=data, fit_reg=True,
scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Depth", color="#4e4c39")
```

Out[11]:

```
Text(0.5, 1.0, 'Regression Line on Price vs Depth')
```

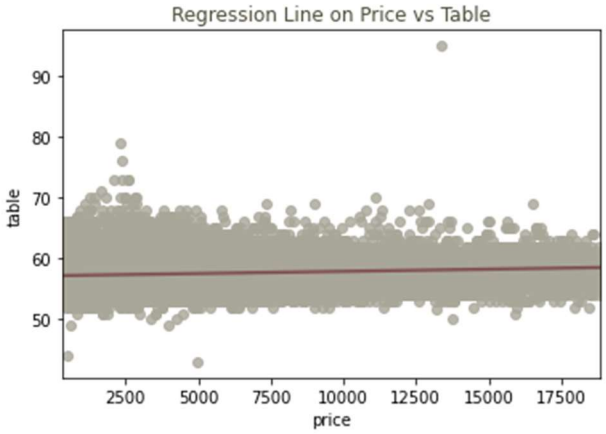


In [12]:

```
ax=sns.regplot(x="price", y="table", data=data, fit_reg=True,
scatter_kws={"color": "#a9a799"}, line_kws={"color": "#835656"})
ax.set_title("Regression Line on Price vs Table", color="#4e4c39")
```

Out[12]:

```
Text(0.5, 1.0, 'Regression Line on Price vs Table')
```



We can clearly spot outliers in these attributes. Next up, we will remove these data points.

In [13]:

```
#Dropping the outliers.
data = data[(data["depth"]<75)&(data["depth"]>45)]
data = data[(data["table"]<80)&(data["table"]>40)]
data = data[(data["x"]<30)]
data = data[(data["y"]<30)]
data = data[(data["z"]<30)&(data["z"]>2)]
data.shape
```

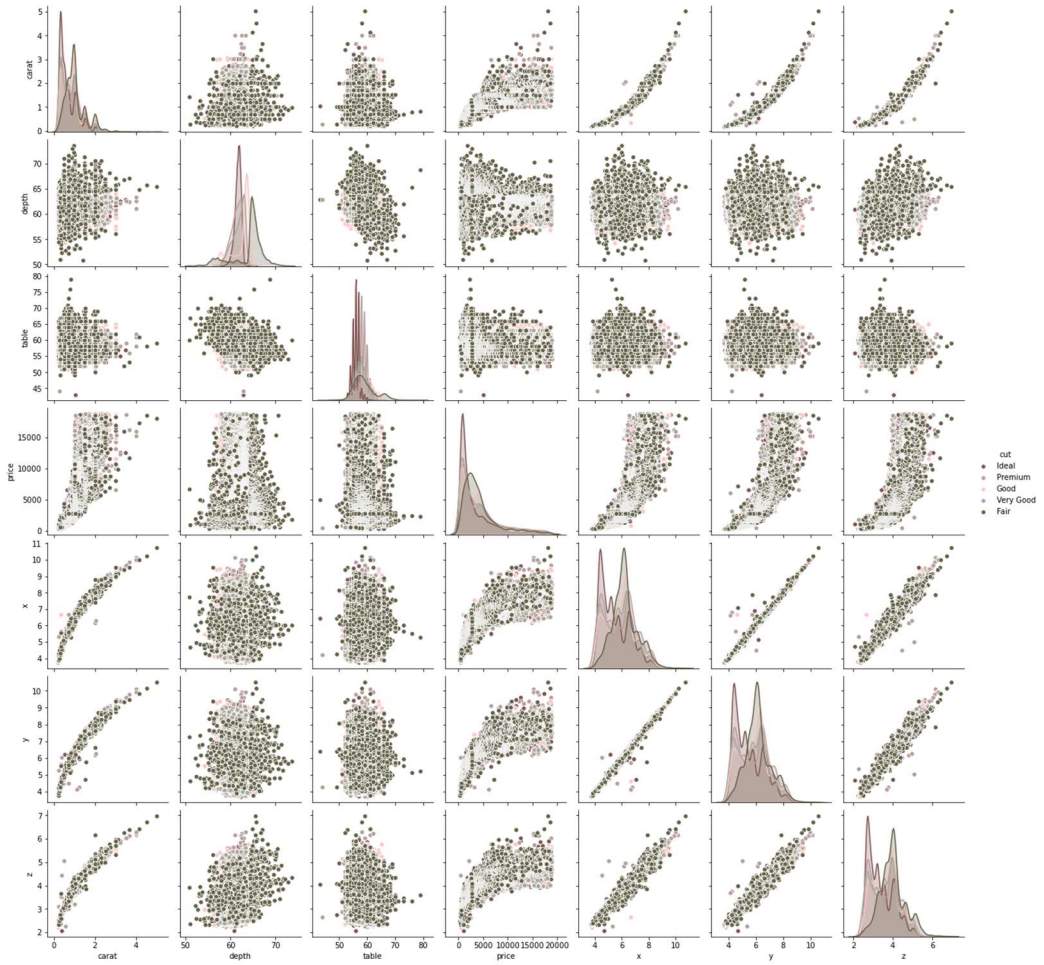
Out[13]:

```
(53907, 10)
```

We can clearly spot outliers in these attributes. Next up, we will remove these data points.Now that we have removed regression outliers, let us have a look at the pair plot of data in our hand.

In [14]:

```
ax=sns.pairplot(data, hue= "cut",palette=shade)
```



That's a much cleaner dataset. Next, we will deal with the categorical variables.

In [15]:

```
# Get list of categorical variables
s = (data.dtypes =="object")
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)
Categorical variables:
['cut', 'color', 'clarity']
```

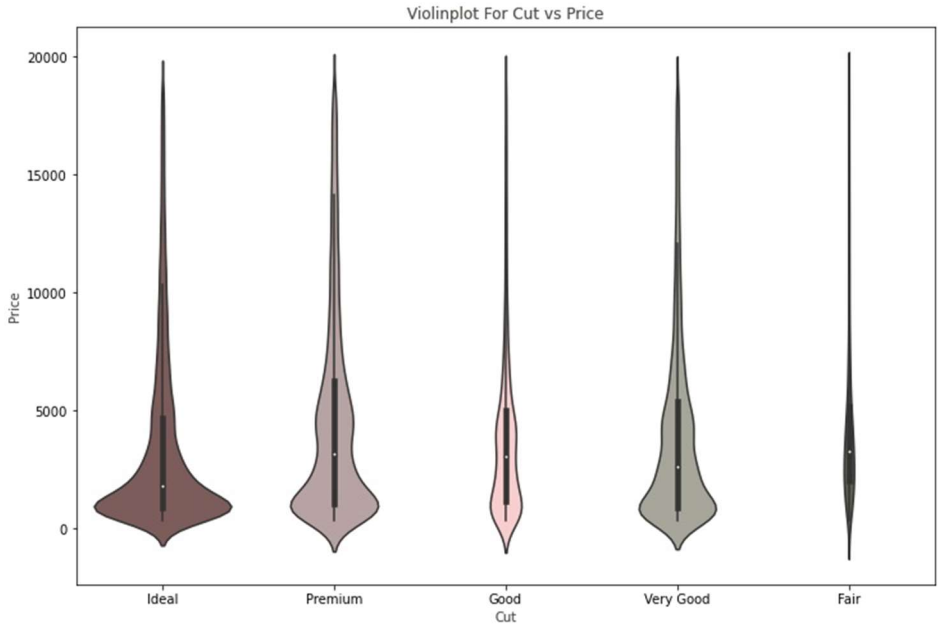
We have three categorical variables. Let us have a look at them.

In [16]:

```
plt.figure(figsize=(12,8))
ax = sns.violinplot(x="cut",y="price", data=data, palette=shade,scale=
"count")
ax.set_title("Violinplot For Cut vs Price", color="#4e4c39")
ax.set_ylabel("Price", color="#4e4c39")
ax.set_xlabel("Cut", color="#4e4c39")
```

Out[16]:

```
Text(0.5, 0, 'Cut')
```

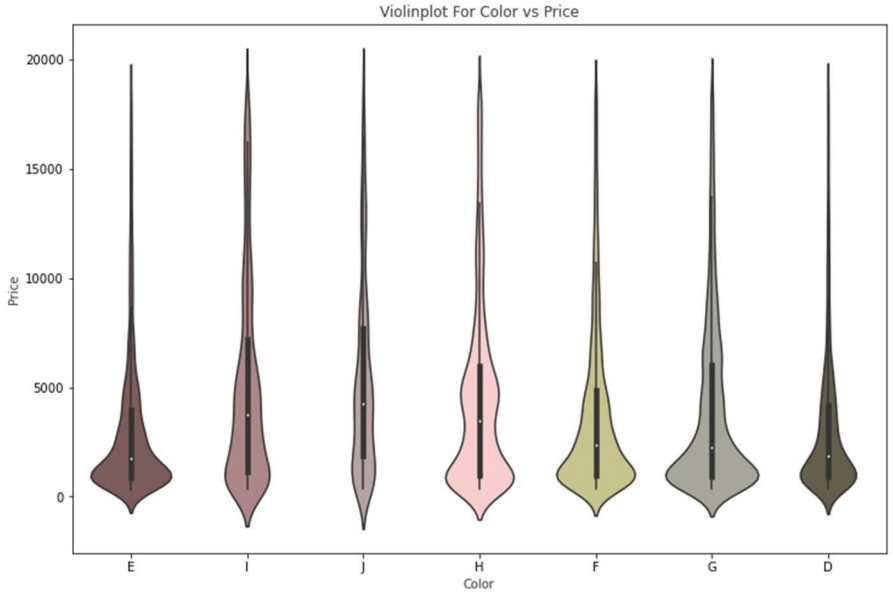


In [17]:

```
plt.figure(figsize=(12,8))
shade_1 = ["#835656","#b38182", "#baa0a0","#ffc7c8","#d0cd85", "#a9a799",
"#65634a"]
ax = sns.violinplot(x="color",y="price", data=data, palette=shade_1,scale=
"count")
ax.set_title("Violinplot For Color vs Price", color="#4e4c39")
ax.set_ylabel("Price", color="#4e4c39")
ax.set_xlabel("Color", color="#4e4c39")
```

Out[17]:

```
Text(0.5, 0, 'Color')
```

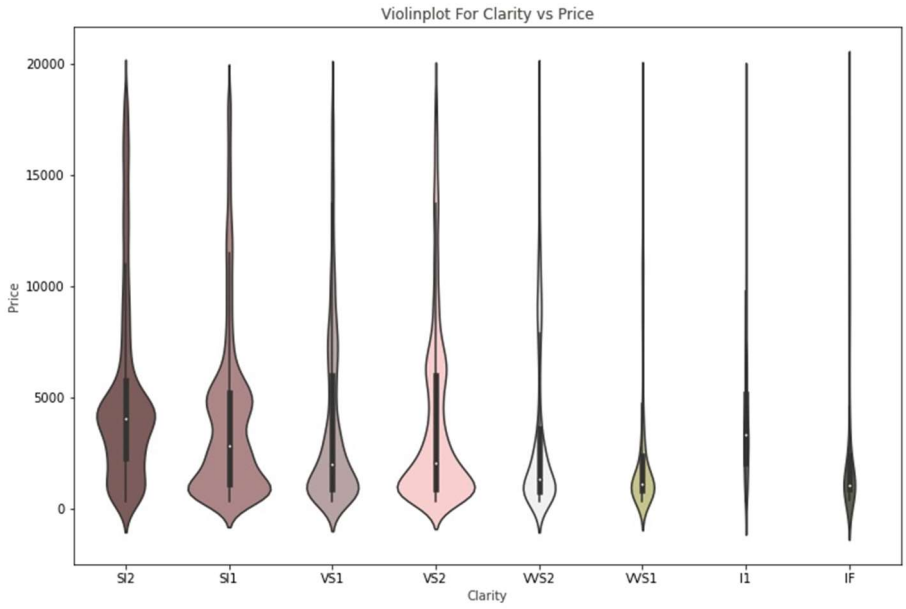


In [18]:

```
plt.figure(figsize=(12,8))
shade_2 = ["#835656","#b38182", "#baa0a0","#ffc7c8","#f1f1f1","#d0cd85",
"#a9a799", "#65634a"]
ax = sns.violinplot(x="clarity",y="price", data=data, palette=shade_2,scale=
"count")
ax.set_title("Violinplot For Clarity vs Price", color="#4e4c39")
ax.set_ylabel("Price", color="#4e4c39")
ax.set_xlabel("Clarity", color="#4e4c39")
```

Out[18]:

```
Text(0.5, 0, 'Clarity')
```



Lable encoding the data to get rid of object dtype.

In [19]:

```
# Make copy to avoid changing original data
label_data = data.copy()

# Apply label encoder to each column with categorical data
label_encoder = LabelEncoder()
for col in object_cols:
    label_data[col] = label_encoder.fit_transform(label_data[col])
label_data.head()
```

Out[19]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	2	1	3	61.5	55.0	326	3.95	3.98	2.43
1	0.21	3	1	2	59.8	61.0	326	3.89	3.84	2.31
2	0.23	1	1	4	56.9	65.0	327	4.05	4.07	2.31
3	0.29	3	5	5	62.4	58.0	334	4.20	4.23	2.63
4	0.31	1	6	3	63.3	58.0	335	4.34	4.35	2.75

```
data.describe()
```

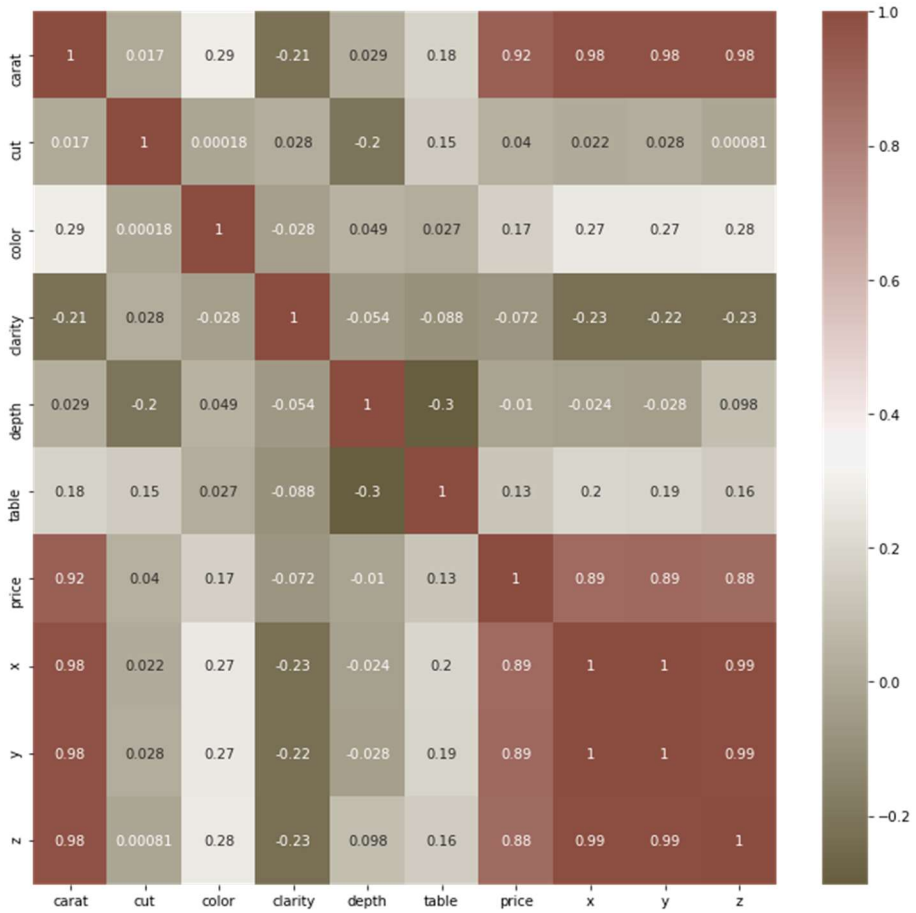
Out[20]:

	carat	depth	table	price	x	y	z
count	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000	53907.000000
mean	0.797628	61.749741	57.455948	3930.584470	5.731463	5.733292	3.539441
std	0.473765	1.420119	2.226153	3987.202815	1.119384	1.111252	0.691434
min	0.200000	50.800000	43.000000	326.000000	3.730000	3.680000	2.060000
25%	0.400000	61.000000	56.000000	949.000000	4.710000	4.720000	2.910000
50%	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	1.040000	62.500000	59.000000	5322.000000	6.540000	6.540000	4.040000
max	5.010000	73.600000	79.000000	18823.000000	10.740000	10.540000	6.980000

In [21]:

```
#correlation matrix
cmap = sns.diverging_palette(70,20,s=50, l=40, n=6,as_cmap=True)
corrmat= label_data.corr()
f, ax = plt.subplots(figsize=(12,12))
sns.heatmap(corrmat,cmap=cmap,annot=True, )
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd4c1d18c10>
```



Model Building

```
# Assigning the features as X and target as y
X= label_data.drop(["price"],axis =1)
y= label_data["price"]
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.25,
random_state=7)
# Building pipelines of standard scaler and model for various repressors.

pipeline_lr=Pipeline([("scalar1",StandardScaler()),
                      ("lr_classifier",LinearRegression())])

pipeline_dt=Pipeline([("scalar2",StandardScaler()),
                      ("dt_classifier",DecisionTreeRegressor())])

pipeline_rf=Pipeline([("scalar3",StandardScaler()),
                      ("rf_classifier",RandomForestRegressor())])


pipeline_kn=Pipeline([("scalar4",StandardScaler()),
                      ("rf_classifier",KNeighborsRegressor())])


pipeline_xgb=Pipeline([("scalar5",StandardScaler()),
                      ("rf_classifier",XGBRegressor())])

# List of all the pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_rf, pipeline_kn,
pipeline_xgb]

# Dictionary of pipelines and model types for ease of reference
pipe_dict = {0: "LinearRegression", 1: "DecisionTree", 2: "RandomForest",3:
"KNeighbors", 4: "XGBRegressor"}

# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, y_train)
```

In [24]:

```
cv_results_rms = []
for i, model in enumerate(pipelines):
    cv_score = cross_val_score(model,
X_train,y_train,scoring="neg_root_mean_squared_error", cv=10)
    cv_results_rms.append(cv_score)
    print("%s: %f " % (pipe_dict[i], cv_score.mean()))
```

Out[24]:

```
LinearRegression: -1348.811824
DecisionTree: -752.832514
RandomForest: -548.170470
KNeighbors: -823.649442
XGBRegressor: -545.458107
```

Testing the Model with the best score on the test set

In the above scores, XGBClassifier appears to be the model with the best scoring on negative root mean square error. Let's test this model on a test set and evaluate it with different parameters.

In [25]:

```
# Model prediction on test data
pred = pipeline_xgb.predict(X_test)
```

In [26]:

```
# Model Evaluation
print("R^2:",metrics.r2_score(y_test, pred))
print("Adjusted R^2:",1 - (1-metrics.r2_score(y_test, pred))*(len(y_test)-
1)/(len(y_test)-X_test.shape[1]-1))
print("MAE:",metrics.mean_absolute_error(y_test, pred))
print("MSE:",metrics.mean_squared_error(y_test, pred))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, pred)))
```

Out[26]:

```
R^2: 0.98108479806778
Adjusted R^2: 0.981072157032851
MAE: 278.09339997286685
MSE: 296738.36382521846
RMSE: 544.7369675588563
```