DSCI-631, Applied Machine Learning

Title : Diamond Price Prediction

Winter 2020-21

Team Members: Himanshu Jat, Richard Hong, Manoj Venkatachalaiah

Overview

We have divided our overall problem into:

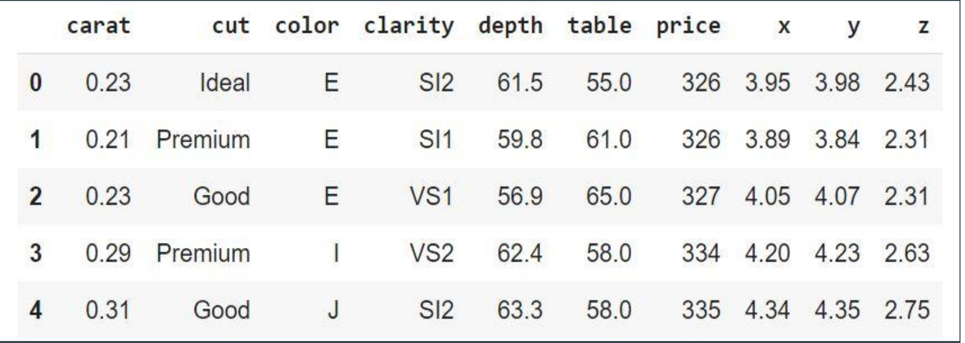
1) A Regression Problem : Where we have predicted the retail price of diamonds using several features.
2) A DBSCAN and regression Problem : Where we have predicted the retail price of diamonds belonging to a certain class.

Data Sources : Kaggle

Link : https://www.kaggle.com/shivam2503/diamonds

Data Description (Raw data)

Sample data:



	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

Attributes and description:

price(target attribute): 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 D (best) to J (worst)

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)

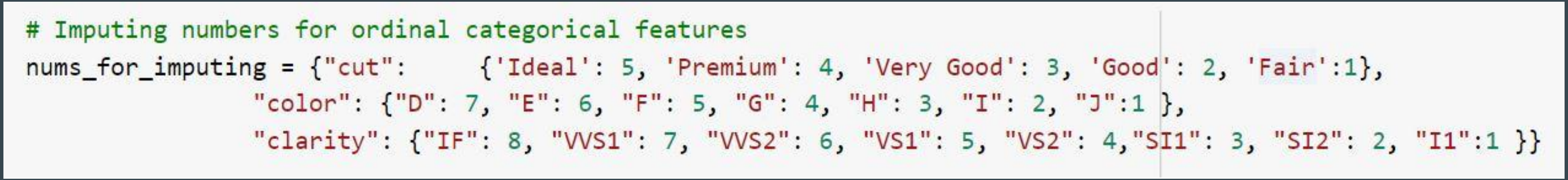
depth: total depth percentage = z / mean(x, y) = 2 * z / (x + y) (43--79)

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

Preprocessing

Encoding the categorical features:

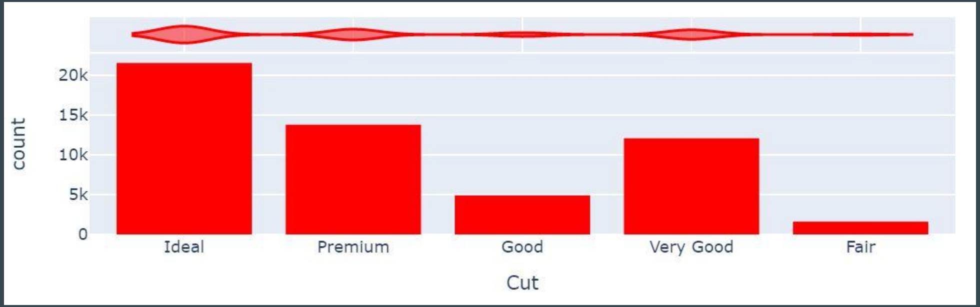
We used the following encoding scheme for the categorical variables color, cut and clarity

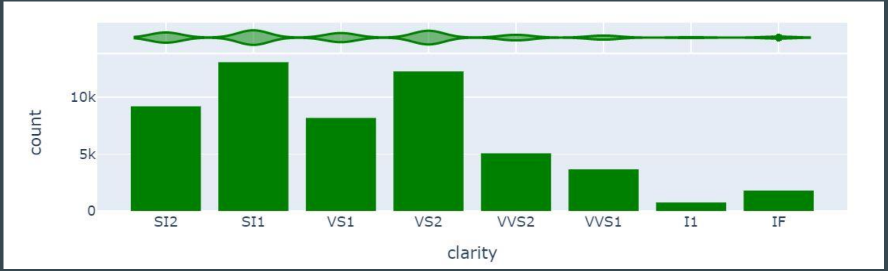


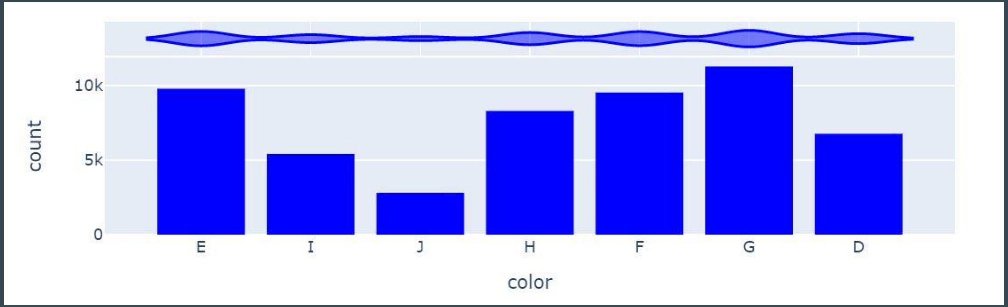
```
# Imputing numbers for ordinal categorical features
nums_for_imputing = {"cut":     {'Ideal': 5, 'Premium': 4, 'Very Good': 3, 'Good': 2, 'Fair':1},
                "color": {"D": 7, "E": 6, "F": 5, "G": 4, "H": 3, "I": 2, "J":1 },
                "clarity": {"IF": 8, "VVS1": 7, "VVS2": 6, "VS1": 5, "VS2": 4,"SI1": 3, "SI2": 2, "I1":1 }}
```

The above encoding scheme was designed based on the logic that a better cut, color and clarity should have a higher score than other values

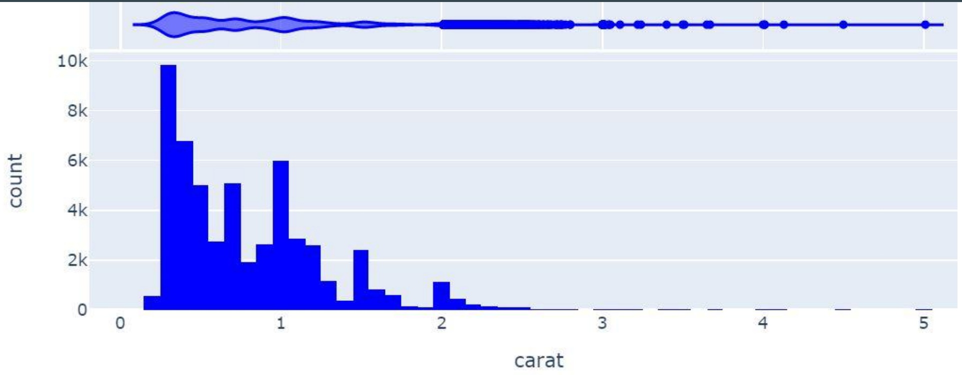
Exploratory Data Analysis
Categorical features: Cut, Clarity and Color

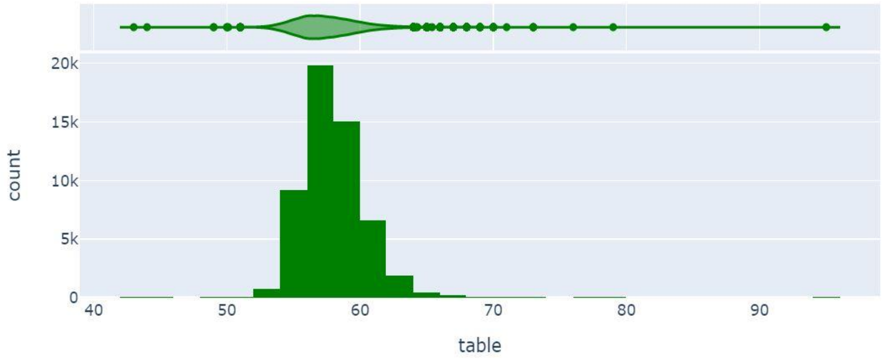


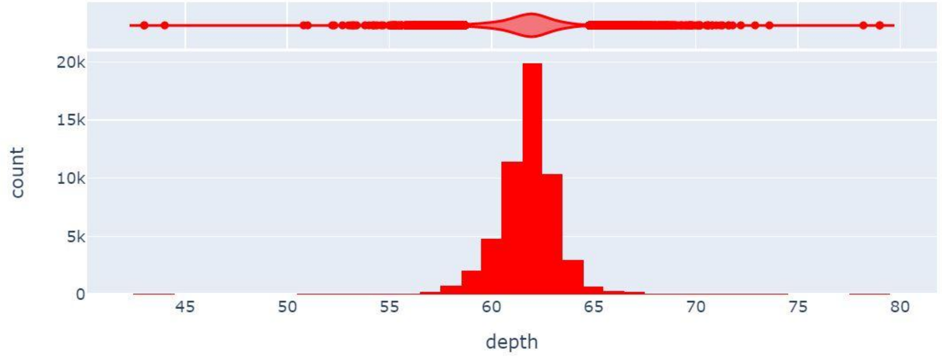


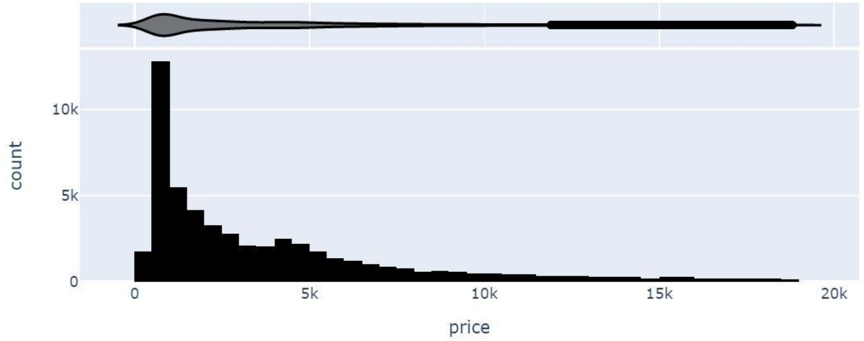


Exploratory Data Analysis
Numerical features: Carat, Depth, Table, Price (target attribute)







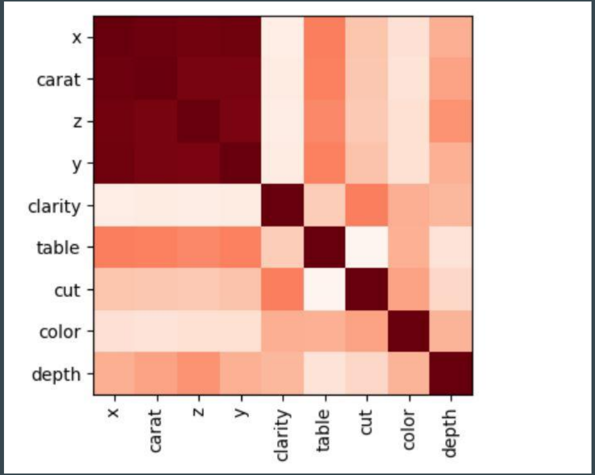


Exploratory Data Analysis

Correlation analysis

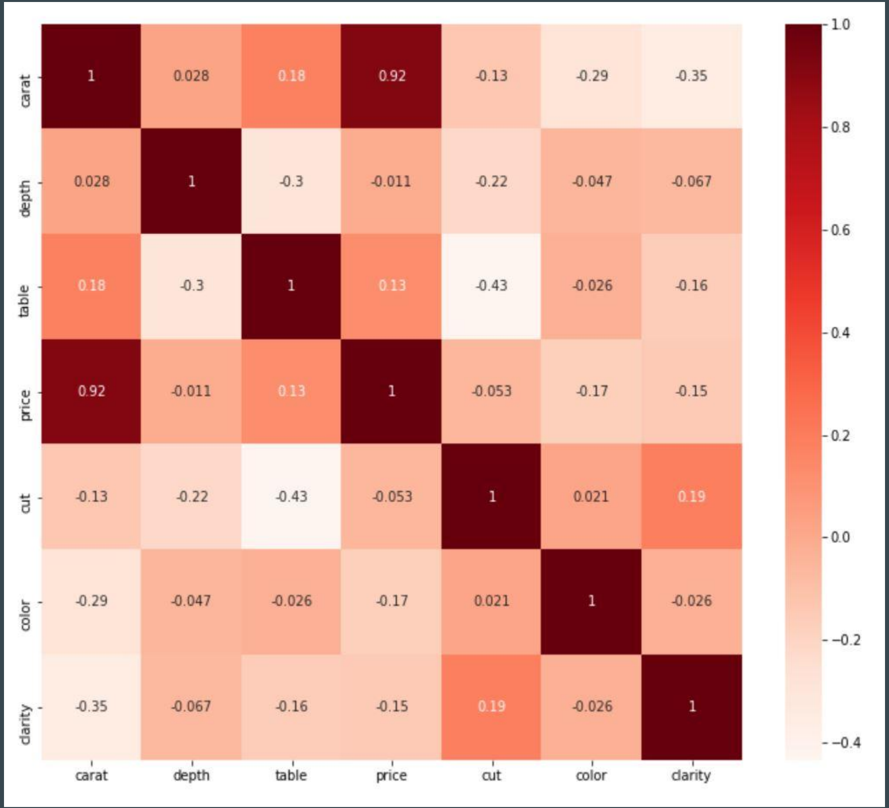
1) Unsupervised feature selection

Applying correlation analysis without the target value:



Since the attributes carat, x, y and z have a high correlativity, we decided the drop the columns x, y and z

2) Applying correlation analysis wit the target value:



	carat	depth	table	price	cut	color	clarity
carat	1	0.028	0.18	0.92	-0.13	-0.29	-0.35
depth	0.028	1	-0.3	-0.011	-0.22	-0.047	-0.067
table	0.18	-0.3	1	0.13	-0.43	-0.026	-0.16
price	0.92	-0.011	0.13	1	-0.053	-0.17	-0.15
cut	-0.13	-0.22	-0.43	-0.053	1	0.021	0.19
color	-0.29	-0.047	-0.026	-0.17	0.021	1	-0.026
clarity	-0.35	-0.067	-0.16	-0.15	0.19	-0.026	1

Predictive modeling: Regression

Train/ test split = 80/20

Scaling method = Standard scalar

Datasets: a) Entire dataset
b) 0.25% outlier removed dataset

Procedure = for each dataset:

a) Get r2 scores by implementing 10 fold cross validation for Linear Regression,
Decision Tree Regressor, Random Forest Regressor, K Neighbors Regressor and XGB Regressor algorithms

b) Perform hyperparameter tuning using GridSearchCv on the algorithm that gives the best r2 score in step a) and get results using the test dataset

Predictive modeling: Regression results

Entire dataset:

R2 scores using cross-validation:

1) Linear Regression : 0.9040
2) Decision Tree Regressor : 0.9661
3) Random Forest Regressor : 0.9806
4) K Neighbors Regressor : 0.9583
5) XGB Regressor : 0.9759

Since the Random forest regressor gave us the best performance, we implemented hyperparameter tuning on it and evaluated it against the test data

Hyper-parameter tuning using GridSearchCv

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
```

R2 score : 0.9818

Predictive modeling: Regression results

0.25% outlier removed dataset:

R2 scores using cross-validation:

1) Linear Regression : 0.9071
2) Decision Tree Regressor : 0.9702
3) *Random Forest Regressor : 0.9825*
4) K Neighbors Regressor : 0.9630
5) XGB Regressor : 0.9790

Since the Random forest regressor gave us the best performance, we implemented hyperparameter tuning on it and evaluated it against the test set

Hyper-parameter tuning using GridSearchCv

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
```

R2 score : 0.9829

Predictive modeling: Regression results

Performance comparison:

Entire Dataset r2 scores :

1) Linear Regression : 0.9040
2) Decision Tree Regressor : 0.9661
3) Random Forest Regressor : 0.9806
4) K Neighbors Regressor : 0.9583
5) XGB Regressor : 0.9759

After hyperparameter tuning :
R2 score : 0.9818

10% data removed!!

0.25% outlier of standard deviation removed dataset r2 scores:

1) Linear Regression : 0.9071
2) Decision Tree Regressor : 0.9702
3) Random Forest Regressor : 0.9825
4) K Neighbors Regressor : 0.9630
5) XGB Regressor : 0.9790

After hyperparameter tuning :
R2 score : 0.9829

Predictive modeling: DBSCAN and Regression

Train/ test split = 80/20

Scaling method = Minmax scalar

Datasets: a) 4 cluster data
b) 2 cluster data

Procedure :

1) Create 4 clusters with DBSCAN algorithm using the features 'carat', 'depth', 'table', 'cut', 'color', 'clarity'. Create labels for each instance based on the cluster it belongs to. Use Linear Regression, Decision Tree Regressor, Random Forest Regressor, K-Neighbors Regressor, and XGB Regressor algorithms to predict each class individually (One Vs All classification). Perform hyper-parameter tuning on the best performing algorithm

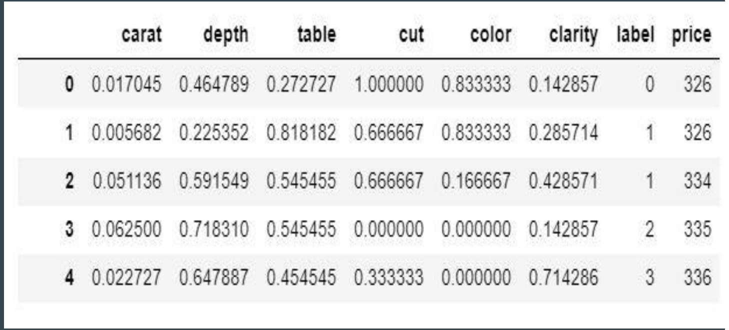
2) Repeat step 1 for 2 clusters. Compare results.

Predictive modeling: DBSCAN and Regression

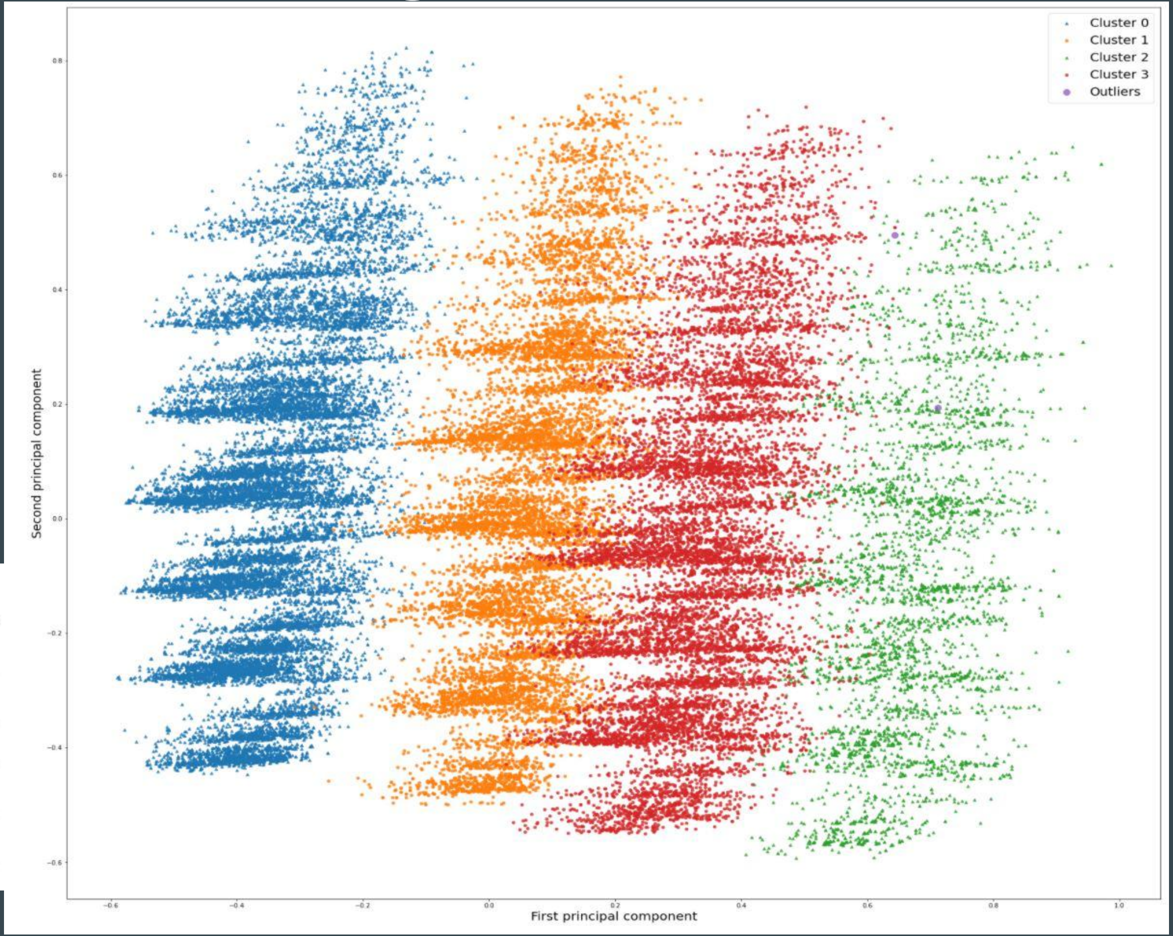
```
DBSCAN(eps=0.3333333, min_samples=5)
```

Creating the clusters : 4 cluster

```
The shape of Cluster 0(Blue)
(20711, 8)
The shape of Cluster 1(Orange)
(12633, 8)
The shape of Cluster 2(Green)
(3941, 8)
The shape of Cluster 3(Red)
(11383, 8)
```



	carat	depth	table	cut	color	clarity	label	price
0	0.017045	0.464789	0.272727	1.000000	0.833333	0.142857	0	326
1	0.005682	0.225352	0.818182	0.666667	0.833333	0.285714	1	326
2	0.051136	0.591549	0.545455	0.666667	0.166667	0.428571	1	334
3	0.062500	0.718310	0.545455	0.000000	0.000000	0.142857	2	335
4	0.022727	0.647887	0.454545	0.333333	0.000000	0.714286	3	336



Predictive modeling: DBSCAN and Regression

4 cluster data:
R2 scores using cross-validation for different clusters:

Cluster 0:
1) Linear Regression : 0.9094
2) Decision Tree Regressor : 0.9707
3) ***Random Forest Regressor : 0.9825***
4) K Neighbors Regressor : 0.9642
5) XGB Regressor : 0.9804

Cluster 1:
1) Linear Regression : 0.9075
2) Decision Tree Regressor : 0.9631
3) ***Random Forest Regressor : 0.9779***
4) K Neighbors Regressor : 0.9494
5) XGB Regressor : 0.9768

Cluster 2:
1) Linear Regression : 0.9045
2) Decision Tree Regressor : 0.9578
3) ***Random Forest Regressor : 0.9756***
4) K Neighbors Regressor : 0.9364
5) XGB Regressor : 0.9743

Cluster 3:
1) Linear Regression : 0.9155
2) Decision Tree Regressor : 0.9708
3) ***Random Forest Regressor : 0.9830***
4) K Neighbors Regressor : 0.9607
5) XGB Regressor : 0.9825

Hyper-parameter tuning using GridSearchCv:

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
```

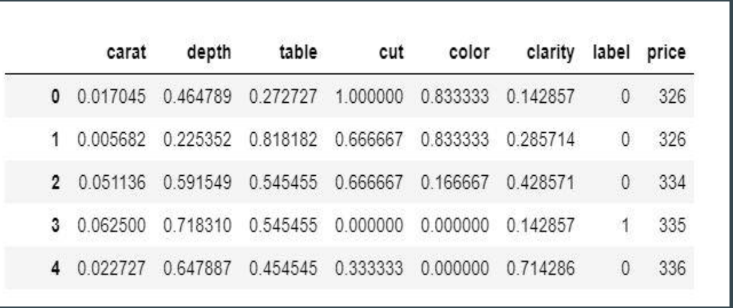
R2 score of testing data:
Cluster 0: 0.9826
Cluster 1: 0.9786
Cluster 2: 0.9668 (3941 instances, the lowest r2 score)
Cluster 3: 0.9838 (11383 instances)

Predictive modeling: DBSCAN and Regression

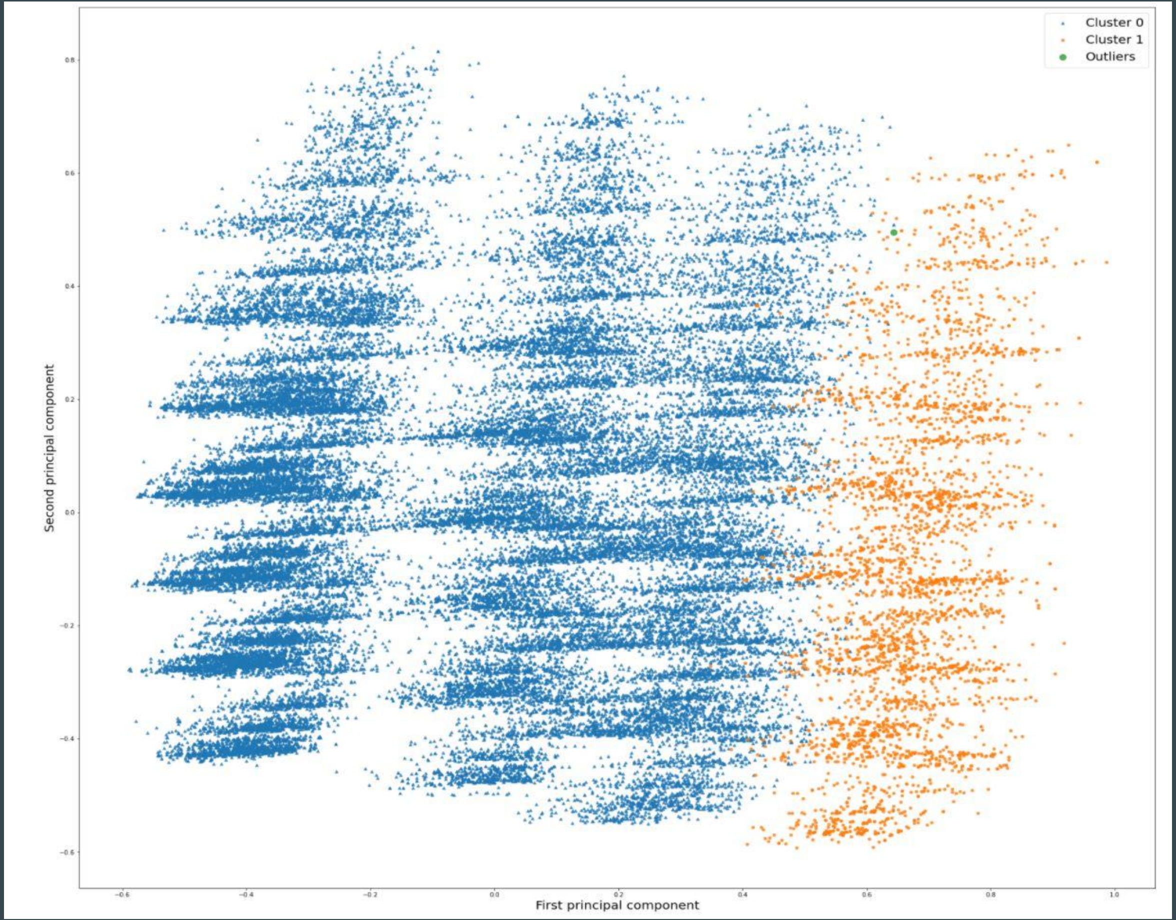
```
dbscan = DBSCAN(eps=1/3, min_samples=2)
```

Creating the clusters : 2 cluster

```
The shape of Cluster 0
(44727, 8)
The shape of Cluster 1
(3942, 8)
```



	carat	depth	table	cut	color	clarity	label	price
0	0.017045	0.464789	0.272727	1.000000	0.833333	0.142857	0	326
1	0.005682	0.225352	0.818182	0.666667	0.833333	0.285714	0	326
2	0.051136	0.591549	0.545455	0.666667	0.166667	0.428571	0	334
3	0.062500	0.718310	0.545455	0.000000	0.000000	0.142857	1	335
4	0.022727	0.647887	0.454545	0.333333	0.000000	0.714286	0	336



Predictive modeling: DBSCAN and Regression

2 cluster data:
R2 scores using cross-validation for different clusters:

Cluster 0:

1) Linear Regression : 0.9088
2) Decision Tree Regressor : 0.9699
3) ***Random Forest Regressor : 0.9824***
4) K Neighbors Regressor : 0.9626
5) XGB Regressor : 0.9792

Cluster 1:

1) Linear Regression : 0.9044
2) Decision Tree Regressor : 0.9591
3) ***Random Forest Regressor : 0.9743***
4) K Neighbors Regressor : 0.9356
5) XGB Regressor : 0.9740

Hyper-parameter tuning using GridSearchCv:

```
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
```

R2 score of testing data :
Cluster 0: 0.9838 (44727 instances)
Cluster 1: 0.9703

Conclusion

Summary

1) Regression

Without hyperparameter tuning: 0.9825
With hyperparameter tuning: 0.9828

2) DBSCAN and Regression

Without hyperparameter tuning:
Cluster 0: 0.9825
Cluster 1: 0.9743

With hyperparameter tuning:
Cluster 0: 0.9838
Cluster 1: 0.9703

Random forest algorithms gave us the best results in both stages of the project, although the other algorithms weren’t behind my a lot.