DIAMOND PRICE PREDICTION

Sherin Ahmad && Roaa mamoun

Agenda

- Problem Definition
- Overview of Dataset
- Technical Explanation
  - Exploratory Data Analysis
  - Data Preprocessing
  - Model Building and Evaluation
- Solution

PROBLEM DEFINITION

The main aim of this problem statement is to predict diamond price. This problem comes under **supervised Machine Learning Regression**.

Overview of Dataset

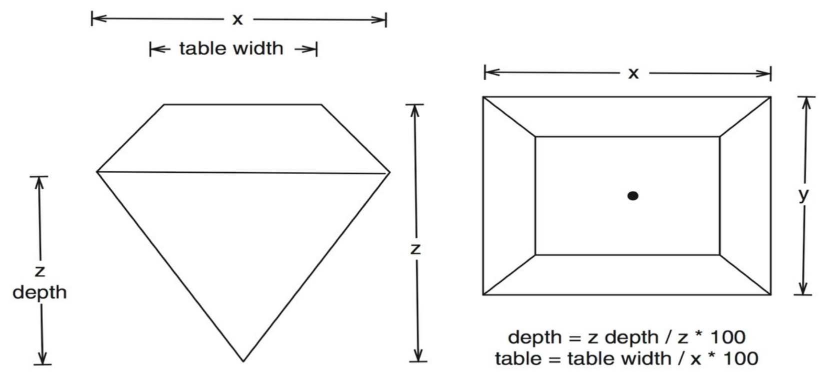


Fig. 3.1 How the variables x, y, z, table and depth are measured

Data Description

This classic dataset contains the prices and other attributes of almost **54,000** diamonds. It's a great dataset for beginners learning to work with data analysis and visualization.

Features

carat weight of the diamond
cut quality of the cut
color diamond color
clarity clear the diamond
x length
y width
z depth
depth total depth percentage = z / mean (x, y)
table width of top of diamond

Target

price price of the diamond

Exploratory Data Analysis

Exploratory Data Analysis is one of the important steps in the data analysis process.

Here, the focus is on making sense of the data in hand — things like formulating the correct questions to ask to your data set, how to manipulate the data sources to get the required answers, and others

Basic data exploration

head

	Id	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	1.06	Ideal	I	SI2	61.8	57.0	4270	6.57	6.60	4.07
1	2	1.51	Premium	G	VVS2	60.9	58.0	15164	7.38	7.42	4.51
2	3	0.32	Ideal	F	VS2	61.3	56.0	828	4.43	4.41	2.71
3	4	0.53	Ideal	G	VS2	61.2	56.0	1577	5.19	5.22	3.19
4	5	0.70	Premium	H	VVS2	61.0	57.0	2596	5.76	5.72	3.50

info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43152 entries, 0 to 43151
Data columns (total 11 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Id       43152 non-null  int64
 1   carat    43152 non-null  float64
 2   cut      43152 non-null  object
 3   color    43152 non-null  object
 4   clarity  43152 non-null  object
 5   depth    43152 non-null  float64
 6   table    43152 non-null  float64
 7   price    43152 non-null  int64
 8   x        43152 non-null  float64
 9   y        43152 non-null  float64
 10  z        43152 non-null  float64
dtypes: float64(6), int64(2), object(3)
memory usage: 3.6+ MB
```

describe

	Id	carat	depth	table	price	x	y	z
count	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000	43152.000000
mean	21576.500000	0.797855	61.747177	57.458347	3929.491912	5.731568	5.735018	3.538568
std	12457.053745	0.473594	1.435454	2.233904	3985.527795	1.121279	1.148809	0.708238
min	1.000000	0.200000	43.000000	43.000000	326.000000	0.000000	0.000000	0.000000
25%	10788.750000	0.400000	61.000000	56.000000	947.750000	4.710000	4.720000	2.910000
50%	21576.500000	0.700000	61.800000	57.000000	2401.000000	5.700000	5.710000	3.530000
75%	32364.250000	1.040000	62.500000	59.000000	5312.000000	6.540000	6.540000	4.040000
max	43152.000000	5.010000	79.000000	95.000000	18823.000000	10.740000	58.900000	31.800000

nunique

```
Id          43152
carat         266
cut             5
color           7
clarity         8
depth         179
table         121
price       10640
x             546
y             543
z             368
dtype: int64
```

isnull().sum()

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

duplicated().sum()

```
0
```

Categorical variables:

- cut

['Fair', 'Good', 'Very Good’, 'Ideal', 'Premium']

- Color

['J', 'I', 'H','G', 'F', 'E', 'D']

- Clarity

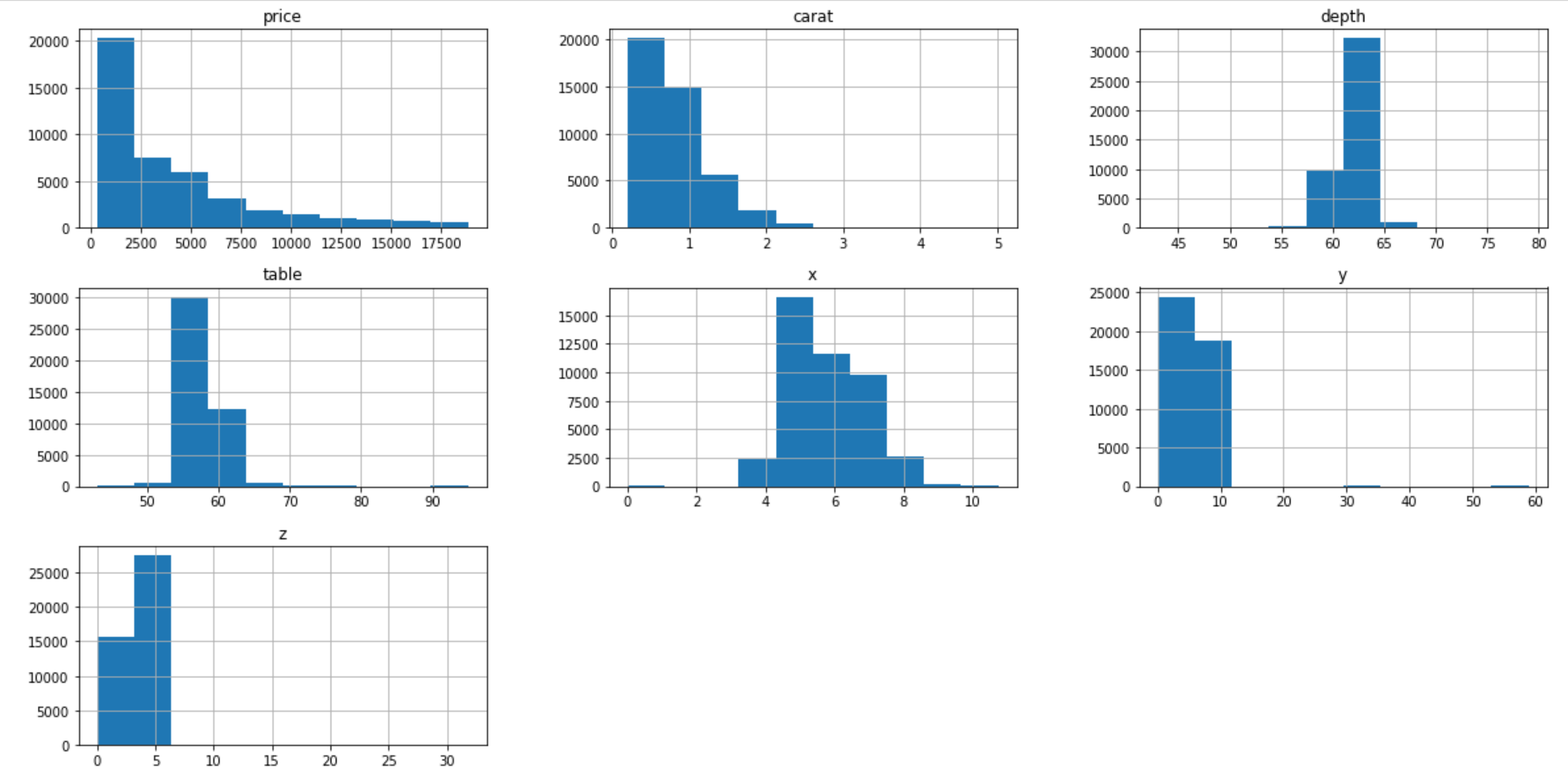
['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF']

Numerical variables:

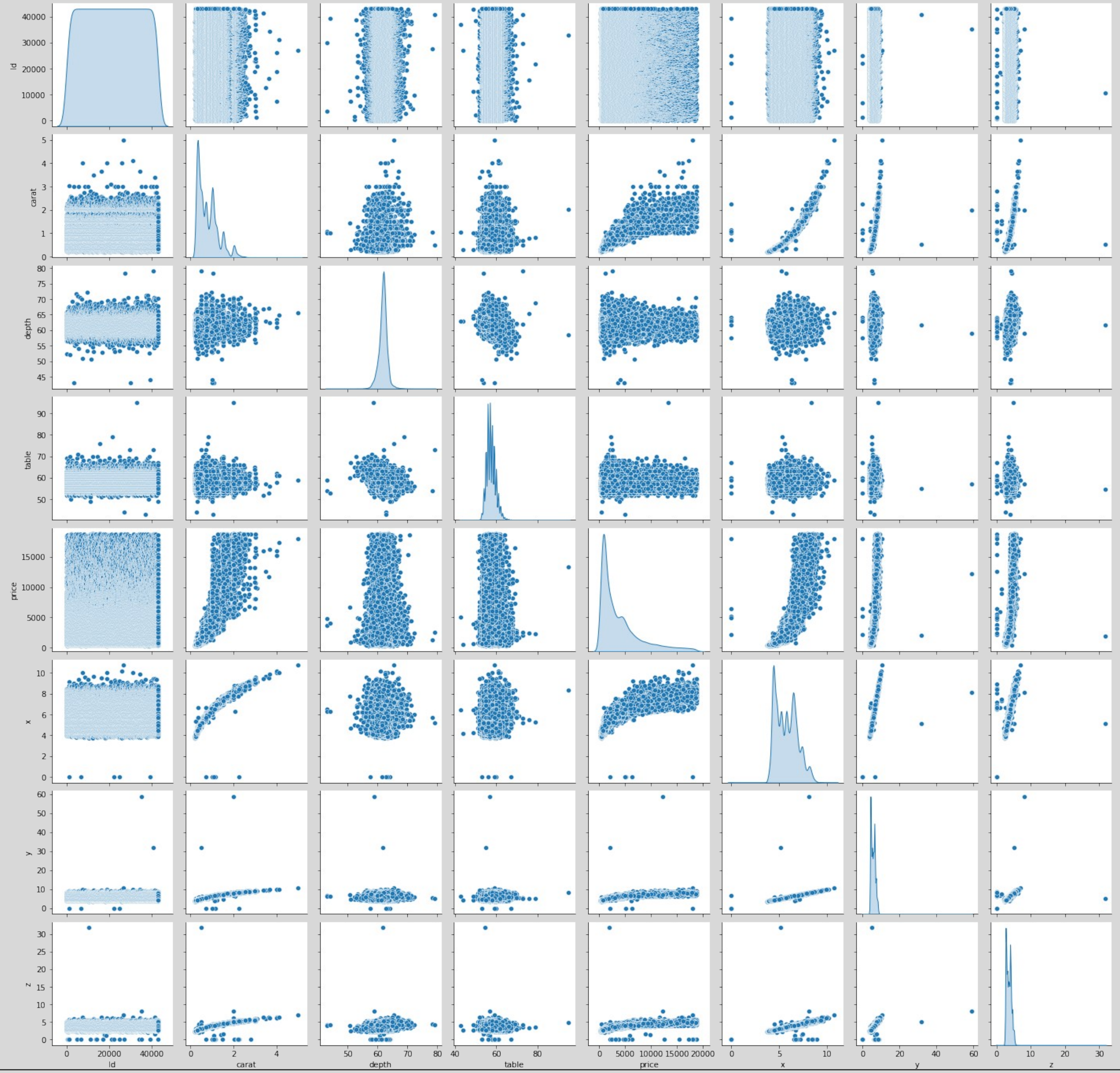
- Id
- Carat
- Depth
- Table
- Price
- X
- Y
- z

Visual Exploratory

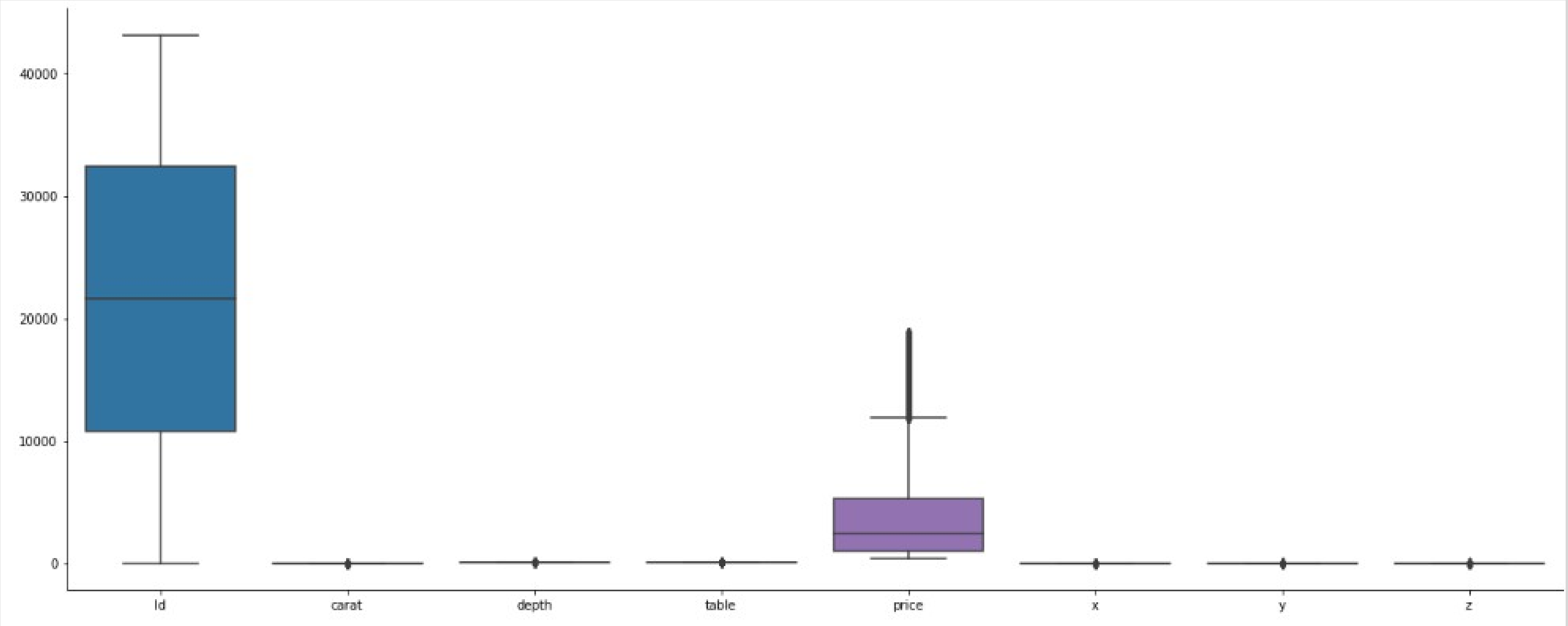
- Visualize distribution of all the Continuous Predictor variables in the data using histograms



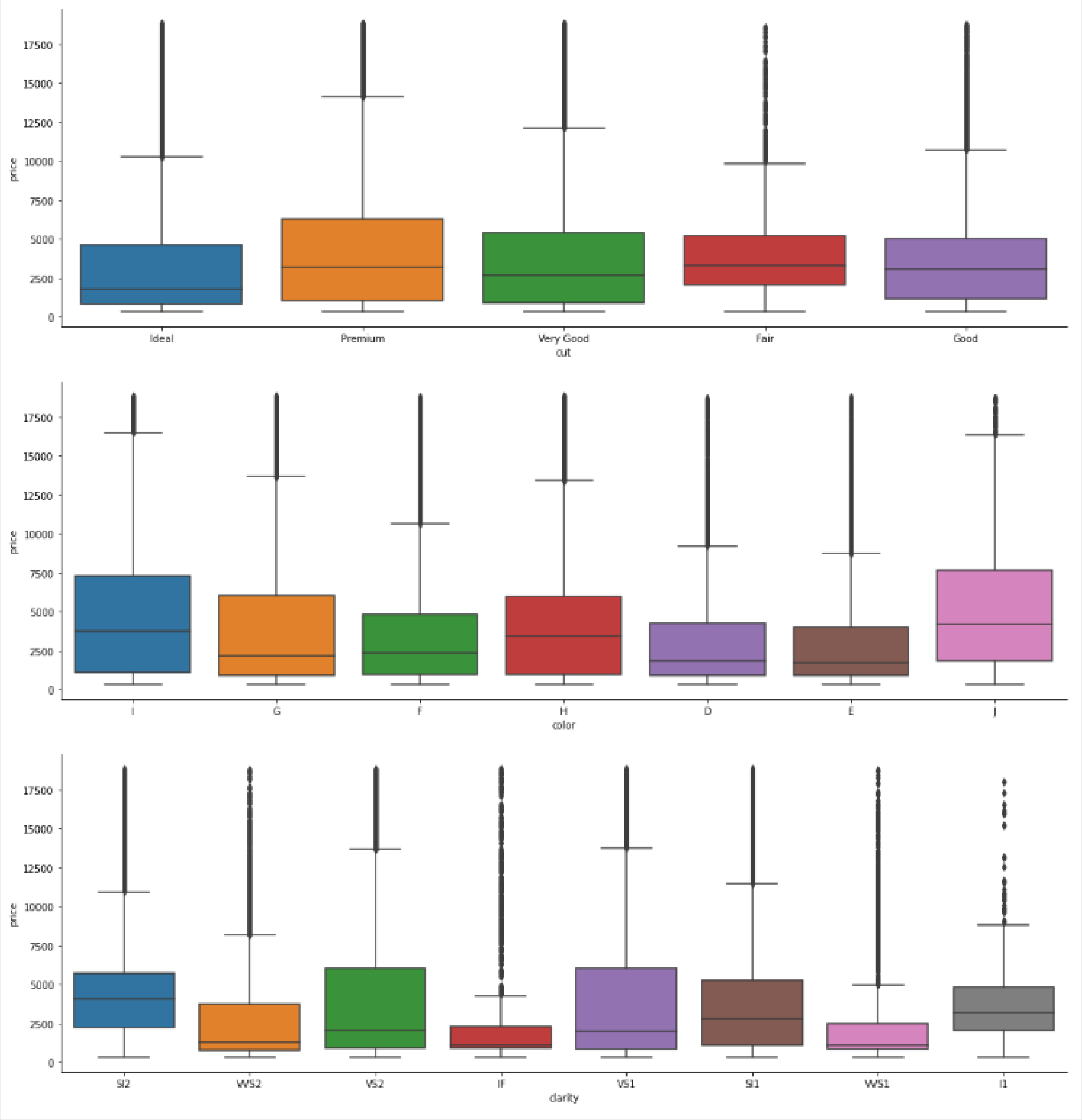
Let’s look at the pair plot of the dataset. Pair plot allows us to see both the distribution of variables and also the relationships between two variables



Let's have a sense of **all** features with respect to target (price) variable by using box plots



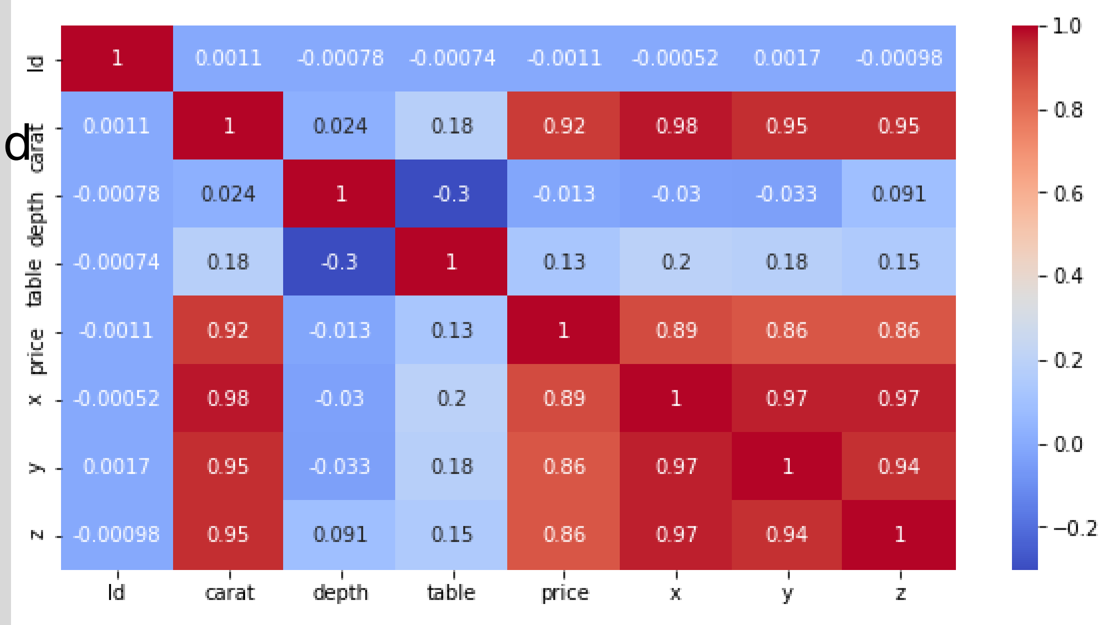
Let's have a sense of **categorical** features with respect to target (price) variable by using box plots



let's quantify that correlation
by using .corr()

	Id	carat	depth	table	price	x	y	z
Id	1.000000	0.001141	-0.000776	-0.000739	-0.001111	-0.000519	0.001660	-0.000981
carat	0.001141	1.000000	0.023944	0.182889	0.921911	0.975760	0.947060	0.948923
depth	-0.000776	0.023944	1.000000	-0.302794	-0.013137	-0.029601	-0.033354	0.090834
table	-0.000739	0.182889	-0.302794	1.000000	0.128501	0.197342	0.184310	0.150746
price	-0.001111	0.921911	-0.013137	0.128501	1.000000	0.885181	0.861354	0.857665
x	-0.000519	0.975760	-0.029601	0.197342	0.885181	1.000000	0.968954	0.965677
y	0.001660	0.947060	-0.033354	0.184310	0.861354	0.968954	1.000000	0.942670
z	-0.000981	0.948923	0.090834	0.150746	0.857665	0.965677	0.942670	1.000000

visualize the same using sns.heatmap() method

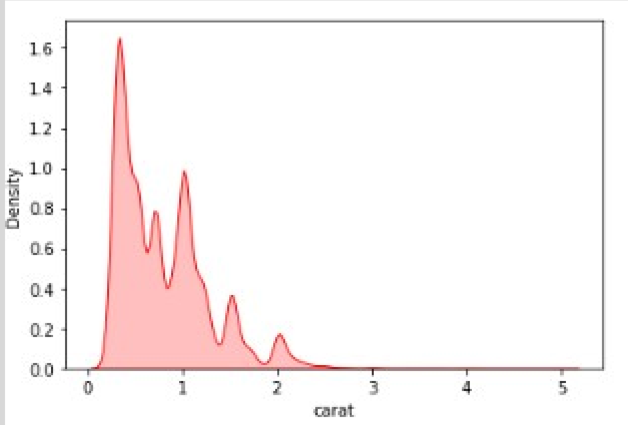


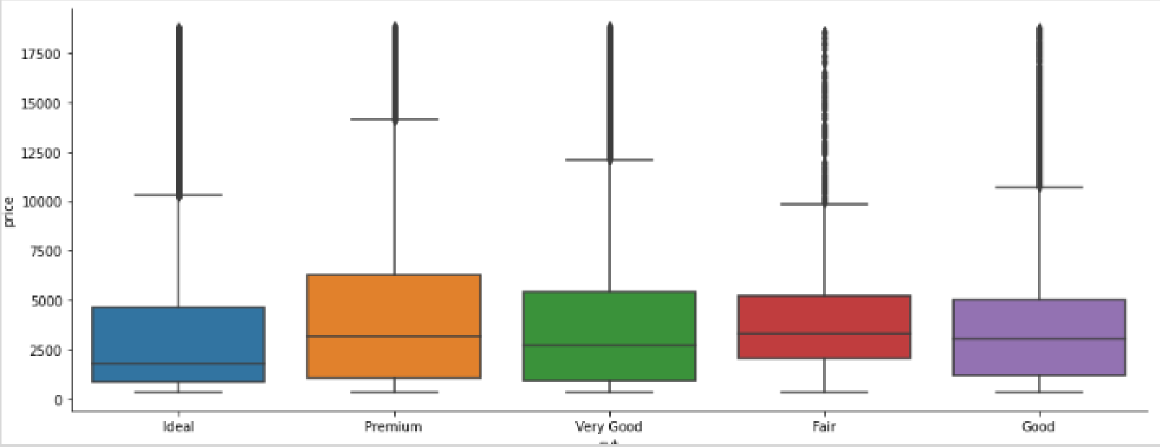
Let’s sort values with absolute correlation with target

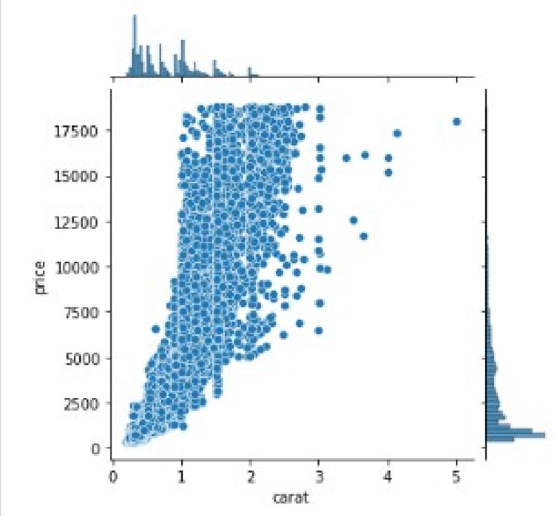
```
price    1.000000
carat    0.921911
x        0.885181
y        0.861354
z        0.857665
table    0.128501
depth    0.013137
Id       0.001111
Name: price, dtype: float64
```

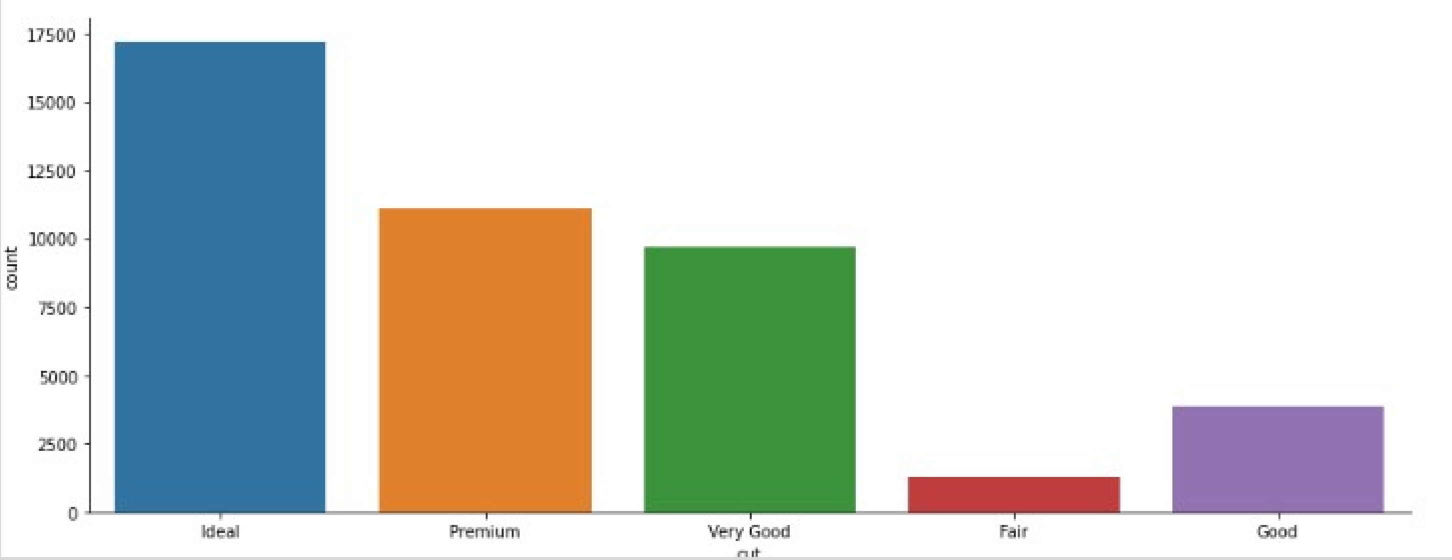
Visualization every feature

- **carat**

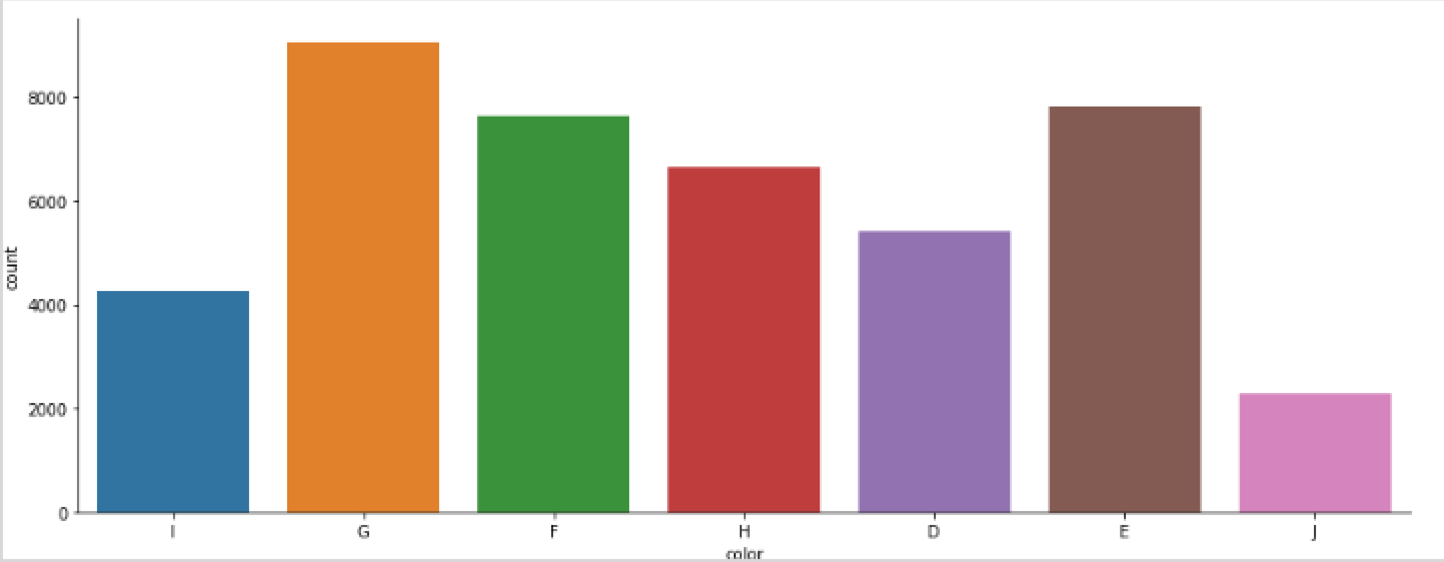




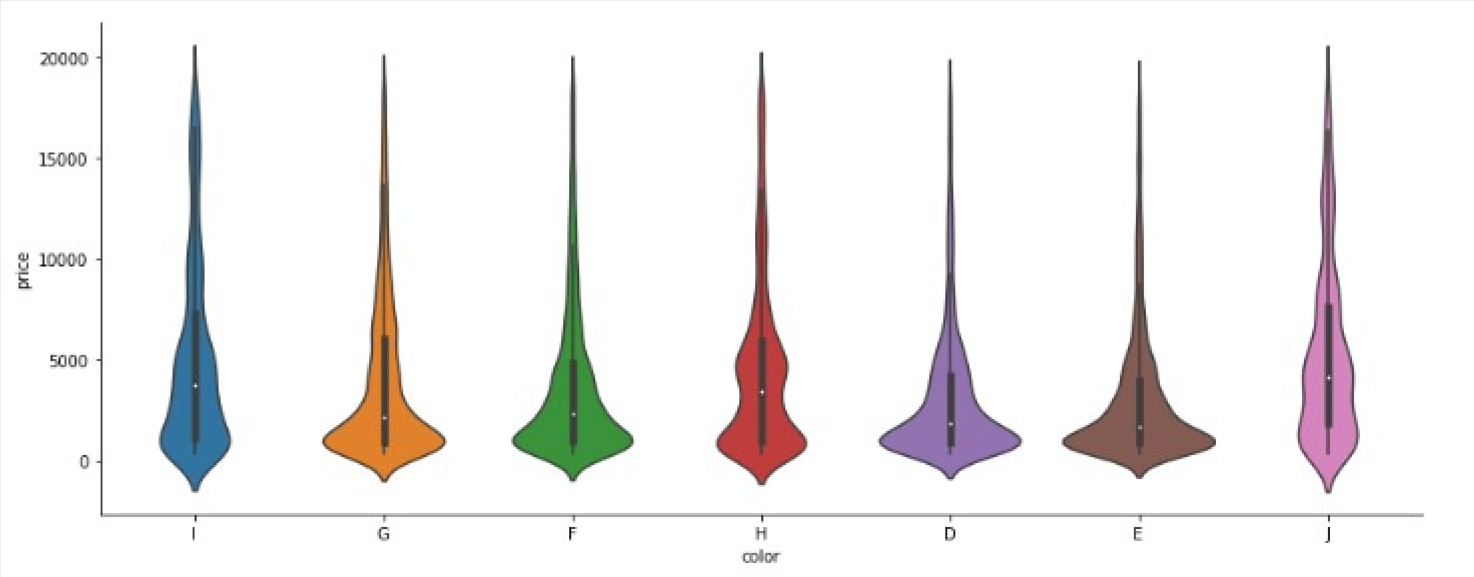




- **Color**

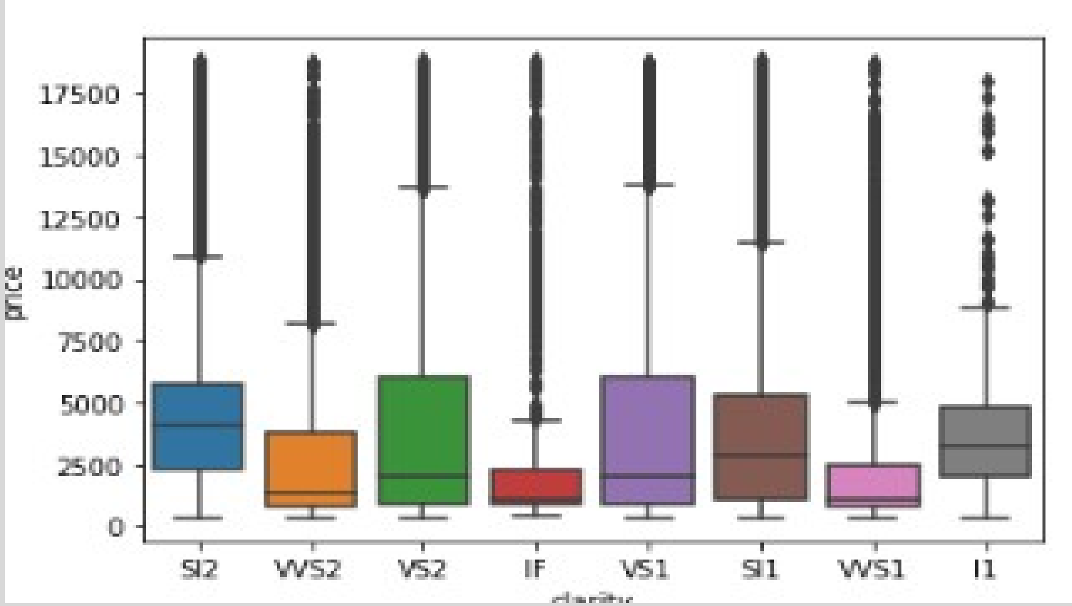


8000
6000
4000
2000
0
count
I
G
F
H
D
E
J
color

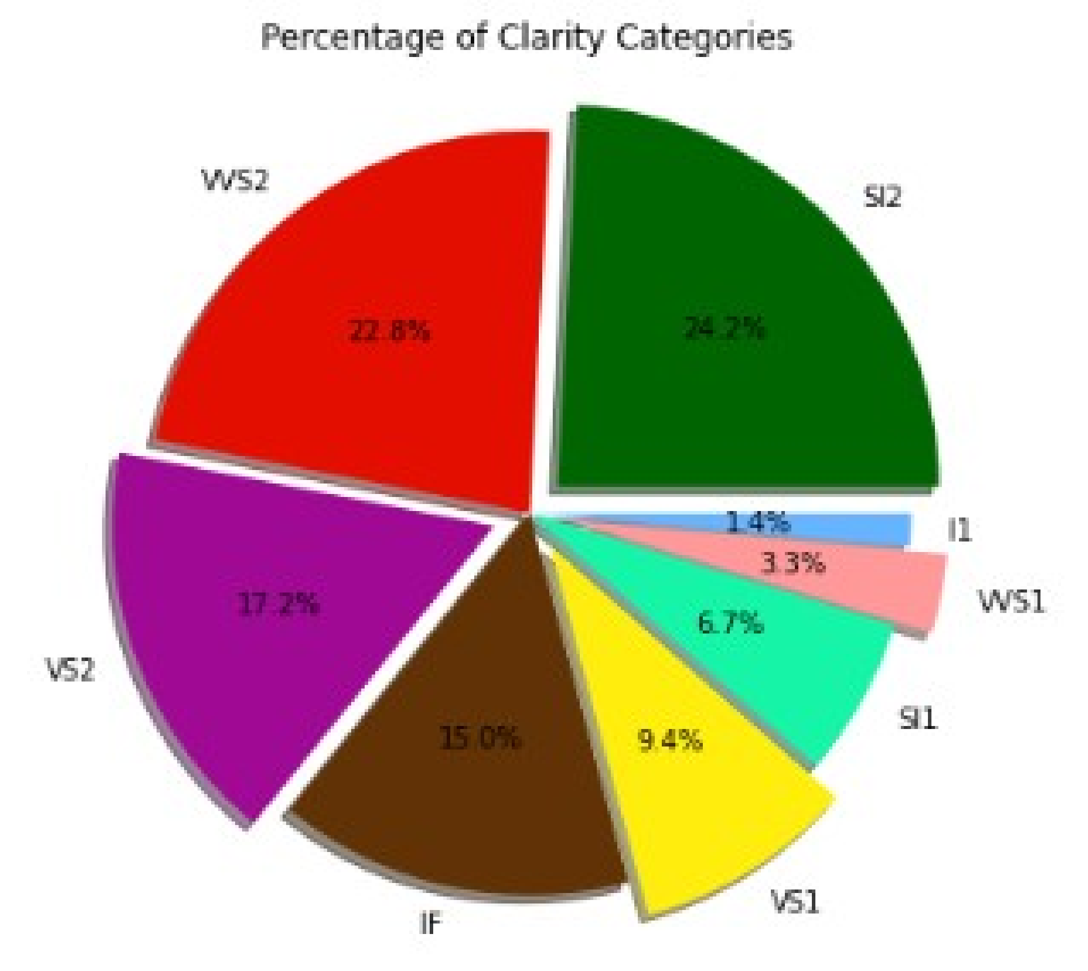


20000
15000
10000
5000
0
price
I
G
F
H
D
E
J
color

- clarity

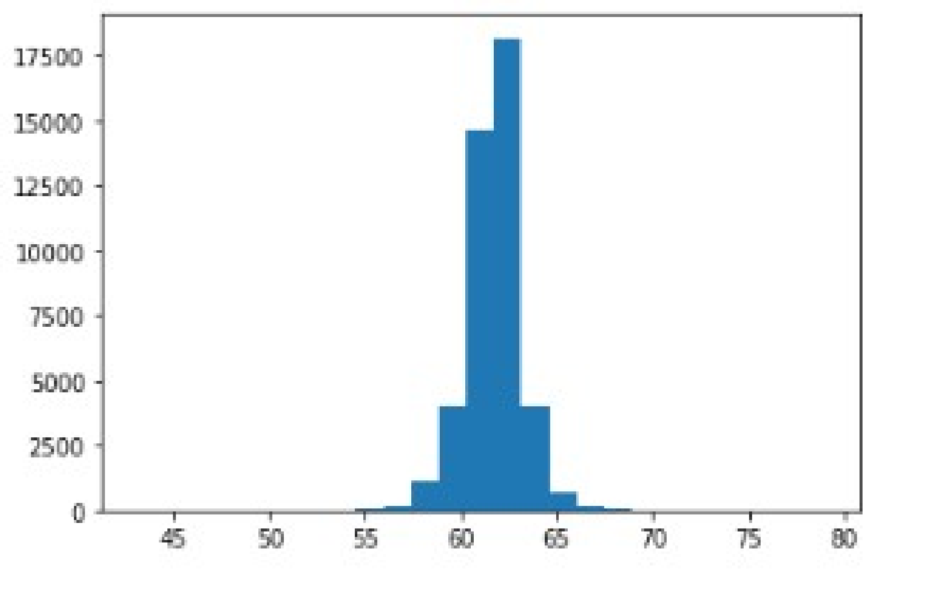


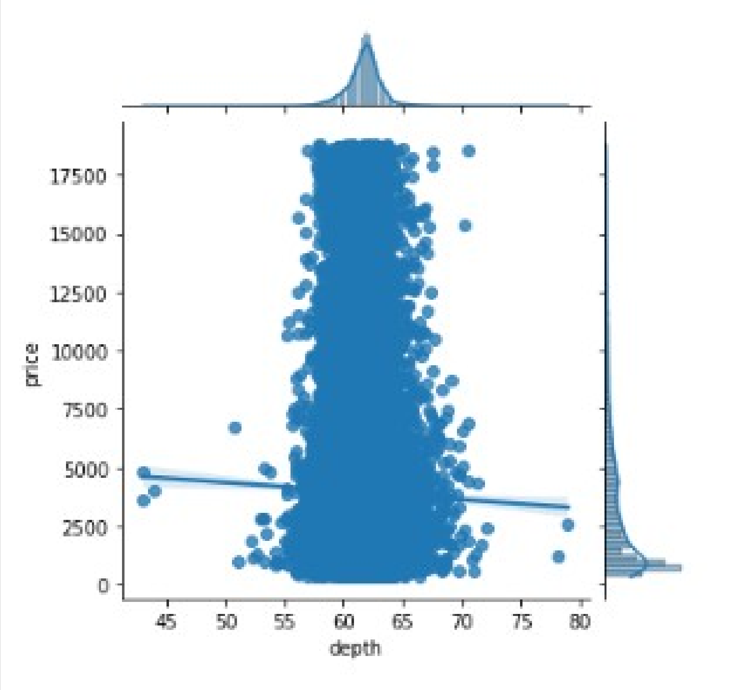
17500
15000
12500
10000
7500
5000
2500
0
price
SI2
VVS2
VS2
IF
VS1
SI1
VVS1
I1



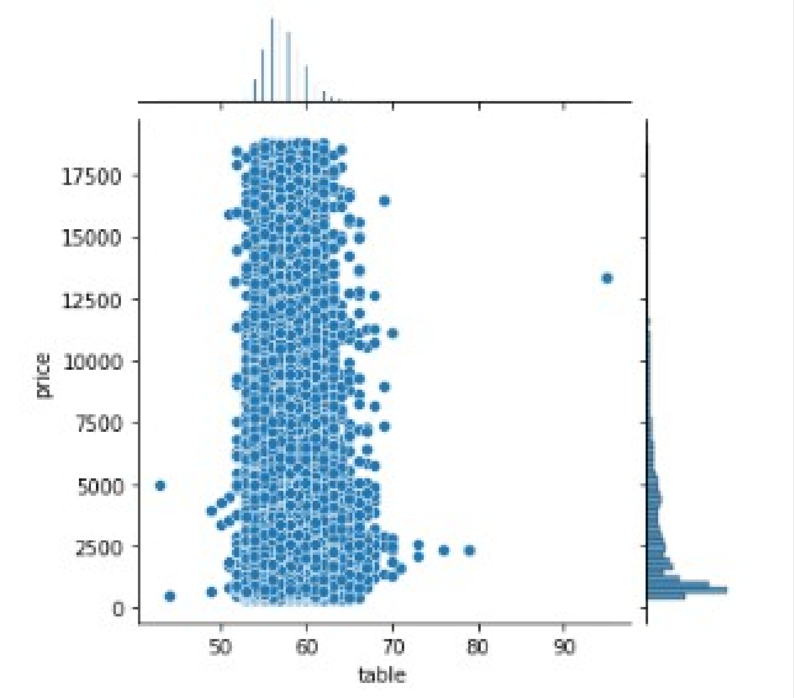
Percentage of Clarity Categories
VVS2
SI2
22.8%
24.2%
1.4%
I1
3.3%
VVS1
17.2%
6.7%
VS2
SI1
15.0%
9.4%
IF
VS1

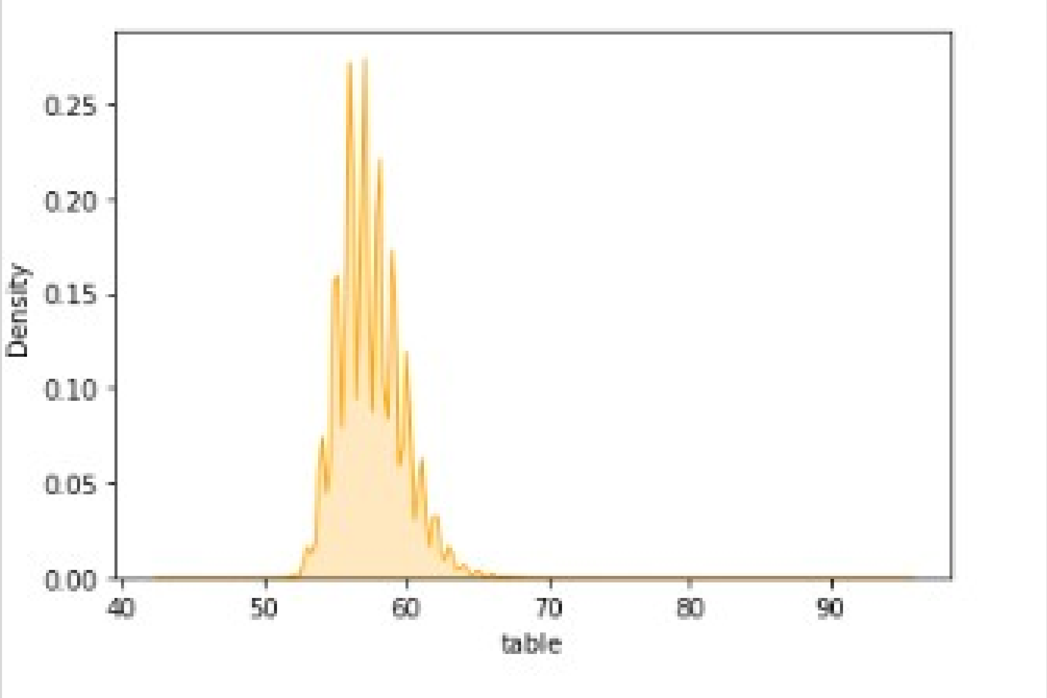
- **depth**


17500
15000
12500
10000
7500
5000
2500
0
45
50
55
60
65
70
75
80


17500
15000
12500
10000
7500
5000
2500
0
price
45
50
55
60
65
70
75
80
depth

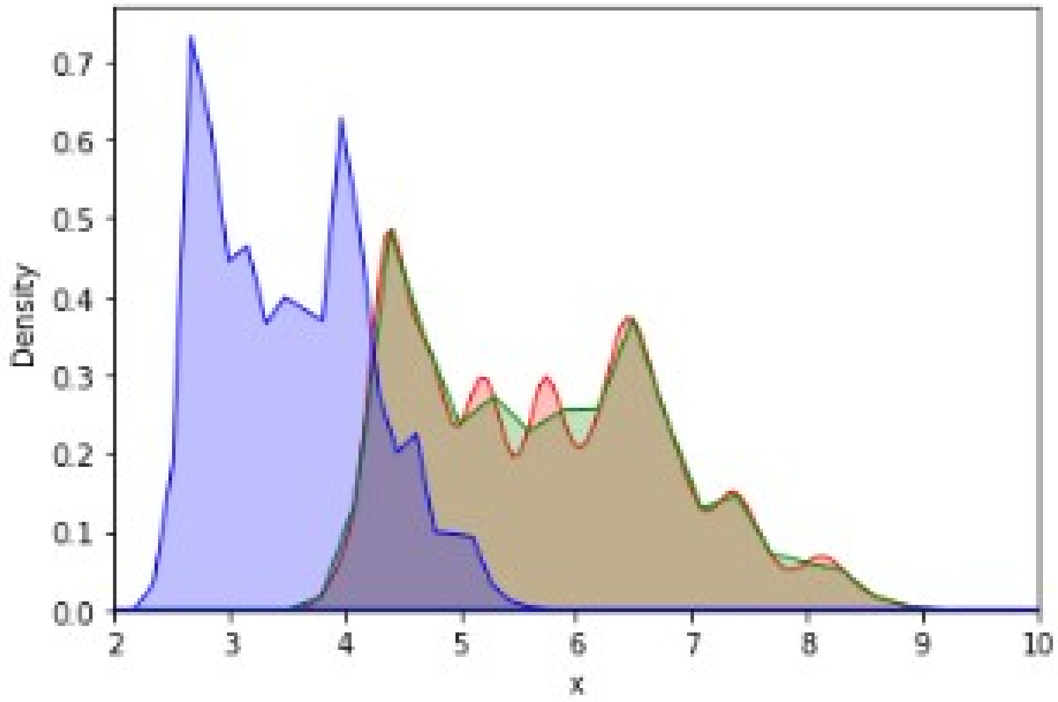
table


price
17500
15000
12500
10000
7500
5000
2500
0
50
60
70
80
90
table


Density
0.25
0.20
0.15
0.10
0.05
0.00
40
50
60
70
80
90
table

```
sns.kdeplot(train_df['x'] ,shade=True , color='r' )
sns.kdeplot(train_df['y'] , shade=True , color='g' )
sns.kdeplot(train_df['z'] , shade= True , color='b')
plt.xlim(2,10)
```

```
(2.0, 10.0)
```



Data preprocessing

Before we begin to building the model

- **Split** train and test :

    X_train, X_test, y_train, y_test = train_test_split(x,y, test_size=0.2, random_state=42)

- **Encoder:** it is good to convert the categorical data to numerical data, so I use **OrdinalEncoder** to encoder categorical column because the values have ordinal

    Cut: worst -> best ['Fair', 'Good', 'Very Good’, 'Ideal', 'Premium’]

    clarity: worst -> best ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2', 'VVS1', 'IF’]

    color: worst -> best ['J', 'I', 'H','G', 'F', 'E', ‘D’]

- **Feature engineering**

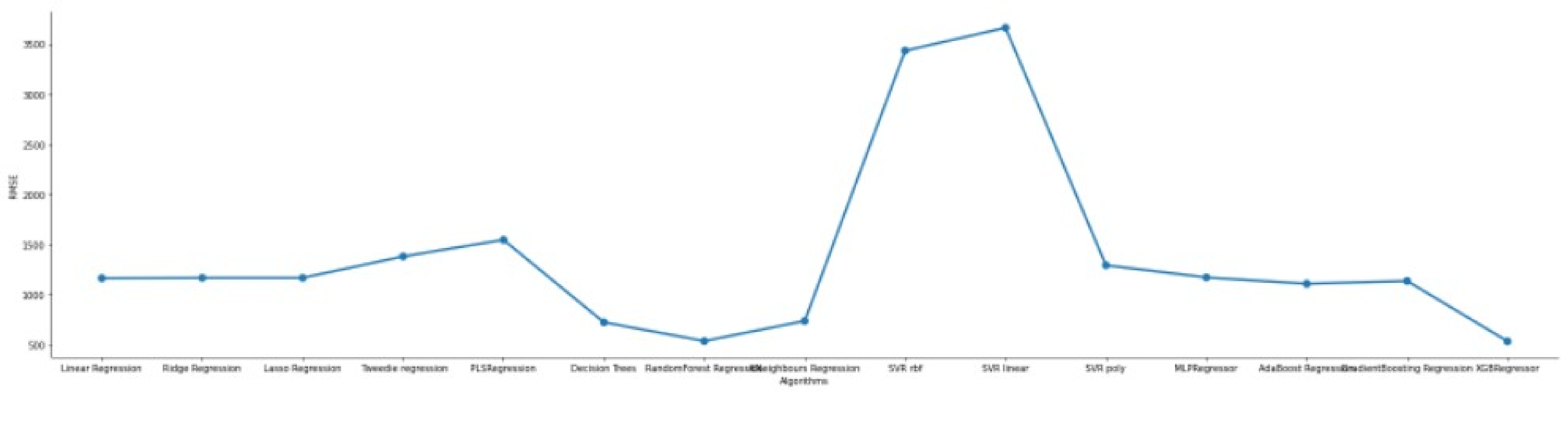
    volume by mult x*y*z drop x,y,z and cut and clarity and color divide carat and drop id

- **Outliers x, y, z == 0 replace min value**
- **Scaling:** MinMaxScaler on features

Model Building and Evaluation

Evaluation Metric
The evaluation metric for this competition is Root Mean Squared Error (**RMSE**). The RMSE is a commonly used measure of the differences between predicted values provided by a model and the actual observed values.

	Algorithms	RMSE
14	XGBRegressor	535.458363
6	RandomForest Regression	538.147085
5	Decision Trees	725.135447
7	KNeighbours Regression	738.401309
12	AdaBoost Regression	1109.842224
13	GradientBoosting Regression	1137.612469
0	Linear Regression	1165.498813
1	Ridge Regression	1168.673320
2	Lasso Regression	1168.673320
11	MLPRegressor	1173.080574
10	SVR poly	1294.400789
3	Tweedie regression	1381.930373
4	PLSRegression	1547.372337
8	SVR rbf	3434.285106
9	SVR linear	3662.365926



3500
3000
2500
2000
1500
1000
500
RMSE
Linear Regression
Ridge Regression
Lasso Regression
Tweedie regression
PLSRegression
Decision Trees
RandomForest Regression
KNeighbours Regression
SVR rbf
SVR linear
SVR poly
MLPRegressor
AdaBoost Regression
GradientBoosting Regression
XGBRegressor
Algorithms

cross_val_score

- **XGBRegressor**

  scores: [542.10441822 537.3111314 545.35761334 573.37097948 520.88732879 550.87423178 570.89884977 607.40190688 605.8569732 520.4258874]

  Mean: **557.4489320264947**

- **RandomForestRegressor**

  scores: [543.9899158 553.47443877 563.56207965 567.31613669 547.85045644 543.03828822 572.21165078 570.69609686 588.28727917 507.56818154]

  Mean: **555.7994523932296**

- **DecisionTreeRegressor**

  scores: [726.50682336 792.08543152 740.27272233 787.22690403 731.27000077 762.31251461 749.93688113 754.61602695 756.93475506 691.04230209]

  Mean: **749.2204361856111**

Fine tuning

XGBRegressor

```
parameters =
{
   'nthread':[x for x in range(1,6)],
   'objective':['reg:squarederror'],
   'learning_rate': [.01,.03, 0.05,0.02],
    'max_depth': [x for x in range(4,10)],
    'min_child_weight': [4,3,5,6],
    'subsample': [0.7],
    'colsample_bytree': [0.7],
     'n_estimators': [500,700,200,400,800]
}
```

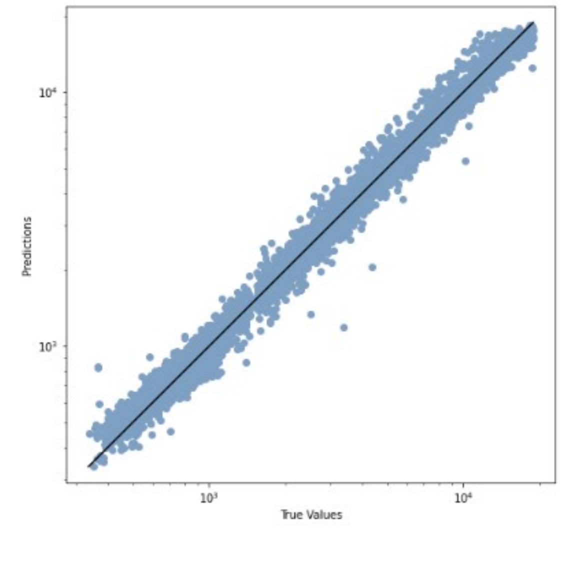
RandomForestRegressor

```
parameters =
{
       'bootstrap': [False , True],
       "criterion":["squared_error"],
       'n_estimators': [x for x in
range(1,600,50)],
       'max_features': [x for x in
range(1,NUM_F)],
       "max_depth":[x for x in range(1,10)]
  }
```

Finally

XGBRegressor(colsample_bytree= 0.7,learning_rate= 0.03,max_depth= 7,min_child_weight= 5,n_estimators= 500,nthread= 1,objective= 'reg:squarederror',subsample= 0.7)

RMSE on test = 510



RandomForestRegressor(bootstrap= True,
criterion= 'squared_error',
max_depth= 8,
max_features= 6,
n_estimators= 500)

RMSE on test = 620.7198339071548

Choose **XGBRegressor model** with best parameter and train on all data, then predict test data and submission

- RMSE on puplic data = 520.56555 score
- RMSE on private data = 528.48284 score

#	△	Team	Members	Score	Entries	Last	Code
1	▲ 1	sherin & roaa		528.48284	24	2d	

THANKS

Any questions