Diamonds Prices Prediction Project

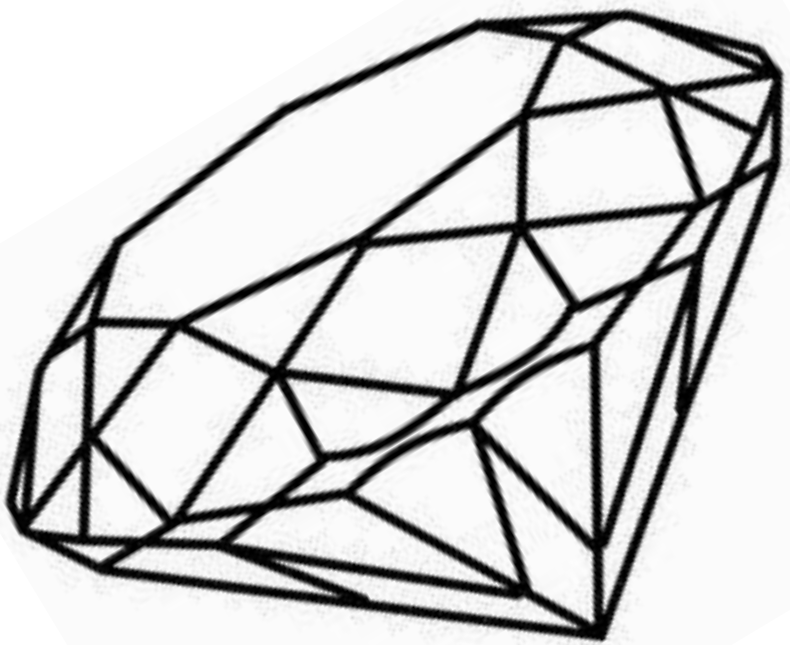
shAI training : 1st project

Group members:
Reem Abu-Farah
Yu Fi (Mabrouka Salmi)
Tarek Al-Mosa

Problem statement

Predict diamonds prices using a dataset with almost 54000 diamonds and 10 variables:

`'cut', 'carat', 'clarity' 'color', 'depth', 'table', 'x', 'y', 'z', 'price'`



Data description

carat = weight of the diamond (0.2--5.01).

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

color = from D (best) to J (worst).

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

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).

price = the Price of the Diamond, (326 --18823).

x = length (0--10.74 mm).

y = width (0--58.9 mm).

z = depth (0--31.8 mm).

Dataset head

```
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

The project process:

1. Get insights from the data
2. Discover and visualize the data
3. Prepare the data
4. Model selection
5. fine-tune the model

Get insights from the data

- The data contains the target variable `'price'` which is continuous value, it is a **supervised** learning and specifically a **regression** problem.
- It can either be **instance or model based learning** depending on the algorithm used to predict the diamond prices, and because the data is not updated frequently so the system doesn't need to learn sequentially and incrementally it is a **batch learning** task.

Discover and visualize data

- We used methods: info, describe to understand the data.
- For visualization those were chosen:

Pairplot, **relplot, hist**: to visualize numerical variables distributions and the relationships between them.

Boxplot: to get hint about outliers existence.

Heatmap: to show the correlation between the variables.

Discover and visualize data (.info method)

The data contains:

- 3 ordinal-categorical variables : 'cut', 'color', 'clarity'.
- 7 numerical variables: 'carat', 'price', 'depth', 'table', 'x', 'y', 'z'.

There are no missing values.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
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: 4.1+ MB
```

Discover and visualize data (.describe method)

- The describe method reveals the existence of **corrupted data**, some observations have values of zeros in: `'x'`, `'y'`, `'z'` variables.
- Which is also clear in the pairplot.

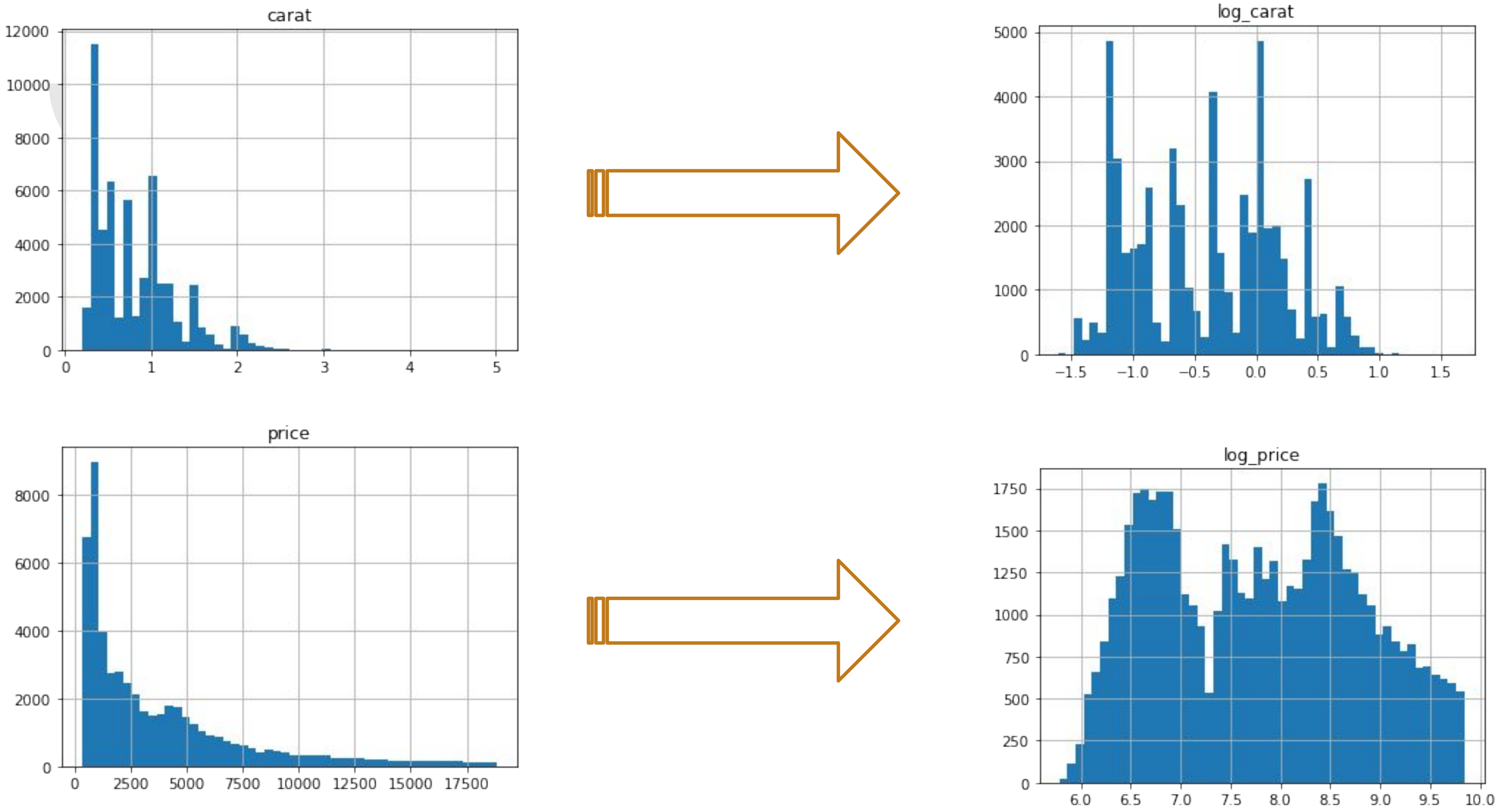
	x	y	z
count	53940.000000	53940.000000	53940.000000
mean	5.731157	5.734526	3.538734
std	1.121761	1.142135	0.705699
min	0.000000	0.000000	0.000000
25%	4.710000	4.720000	2.910000
50%	5.700000	5.710000	3.530000
75%	6.540000	6.540000	4.040000
max	10.740000	58.900000	31.800000

Pairplot + Hist + Relplot:

At the diagonal of the pairplot we see the distribution of the variables. Otherwise it shows scatter plots between each pair of variables.

We notice :

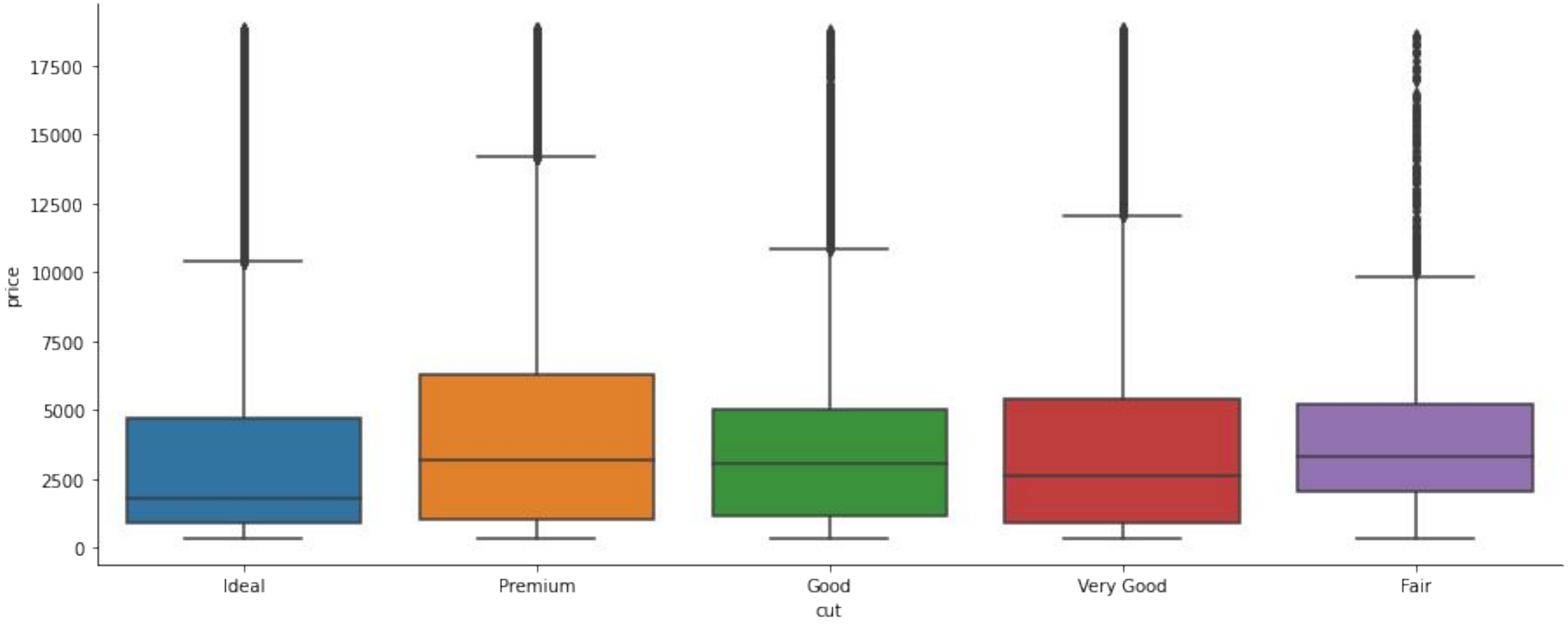
- The variables `'carat'` and `'price'` have a right skewed distribution which is clear in the hist plots. Where we used log to deal with this, shown in next slide.
- Scatter plots show a significant positive correlation between 'price' and: {`'x'`, `'y'`, `'z'`, `'carat'`}.
- Also some scatter plots indicate almost no linear correlation between 'price' and: {`'table'`, `'depth'`}.


carat
12000
10000
8000
6000
4000
2000
0
0 1 2 3 4 5
log_carat
5000
4000
3000
2000
1000
0
−1.5 −1.0 −0.5 0.0 0.5 1.0 1.5
price
8000
6000
4000
2000
0
0 2500 5000 7500 10000 12500 15000 17500
log_price
1750
1500
1250
1000
750
500
250
0
6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0

Boxplots

- Boxplots of `'price'` show clearly the presence of outliers. which was later treated using `'log_price'` (log(price)).
- Boxplots of `'log_price'` demonstrate a huge difference comparing to boxplots of `'price'`.
- Distribution of `'log_price'` seems to have a more normal distribution.
- Distribution of `'log_carat'` also turned to be more normal.

Boxplot between price and cut.



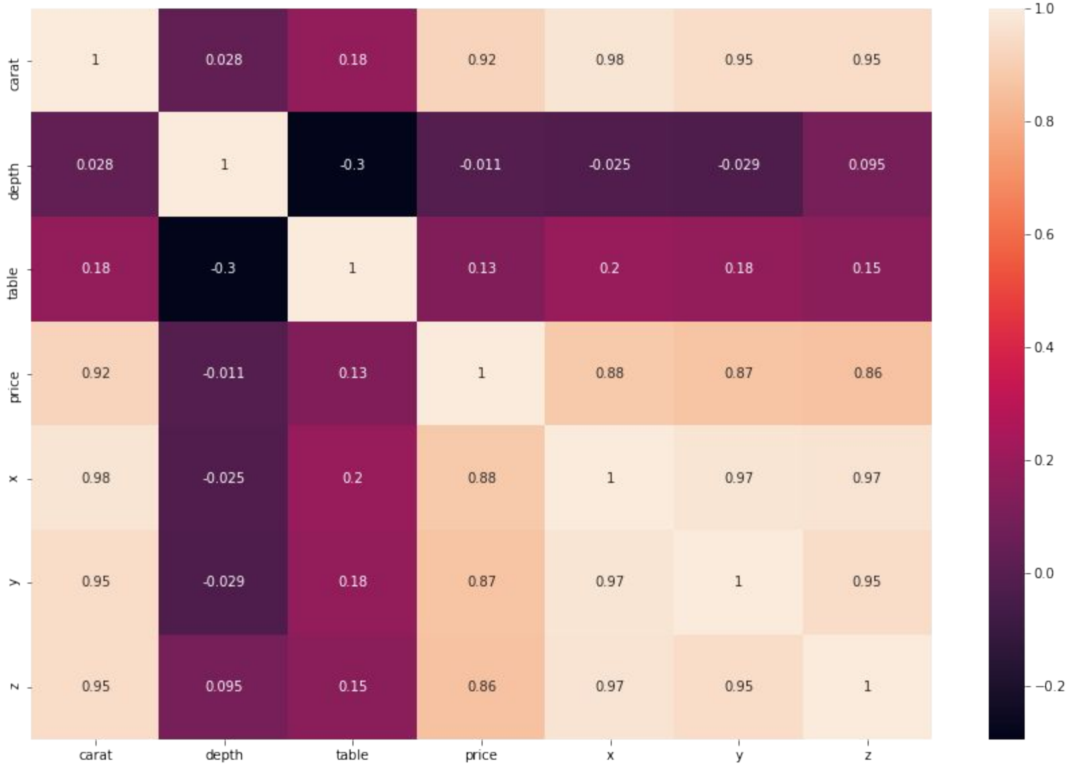
Correlation with Heatmap

- Most variables are positively correlated with the target variable `'price'`.
- Almost all the variable are significantly correlated (>0.8) except `'table'` and `'depth'`. are slightly correlated with the target variable: `'price'`.

Results of data.corr():

```
price     1.000000
carat     0.921591
x         0.884435
y         0.865421
z         0.861249
table     0.127134
depth    -0.010647
Name: price, dtype: float64
```

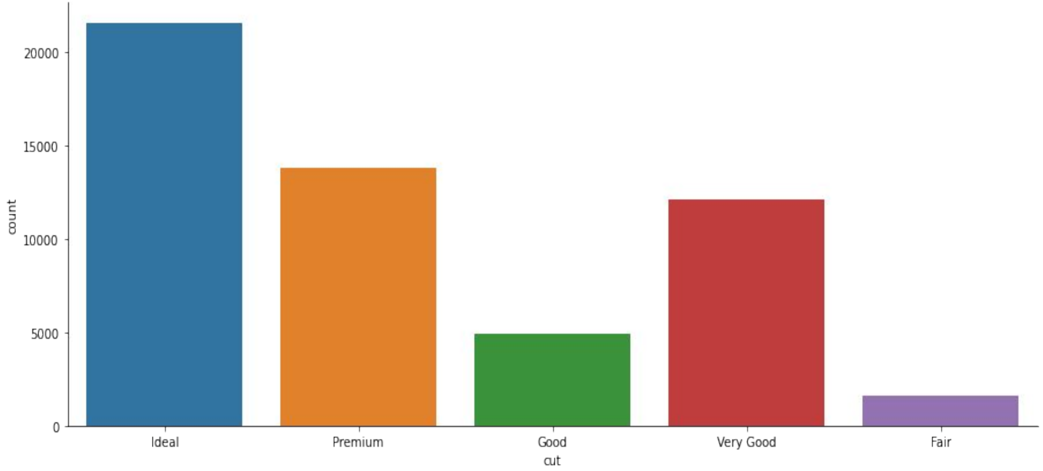
Correlation with Heatmap



Bar charts

- We used the bar charts to help us visualize the different categorical features `'cut'`, `'clarity'`, `'color'`, to find out which features were most common in the dataset, and whether there is a correlation between importance of these features and how frequently they occur in the dataset.

- For example the bar chart on the right shows that a better cut diamond is more commonly sold, which is what we expected initially.



Data preparation

Features extraction

1. Extracting 'table_width' and 'z_depth' (other characteristics of diamond shape) from 'table' and 'depth' (as ratios) respectively, by using these formulas:

```
table_width = table * x / 100

z_depth    = depth * z / 100
```

2. Also we extracted the volume of the diamond feature from: 'x' , 'y' and 'z' features, as follows:

```
Volume = x*y*z
```

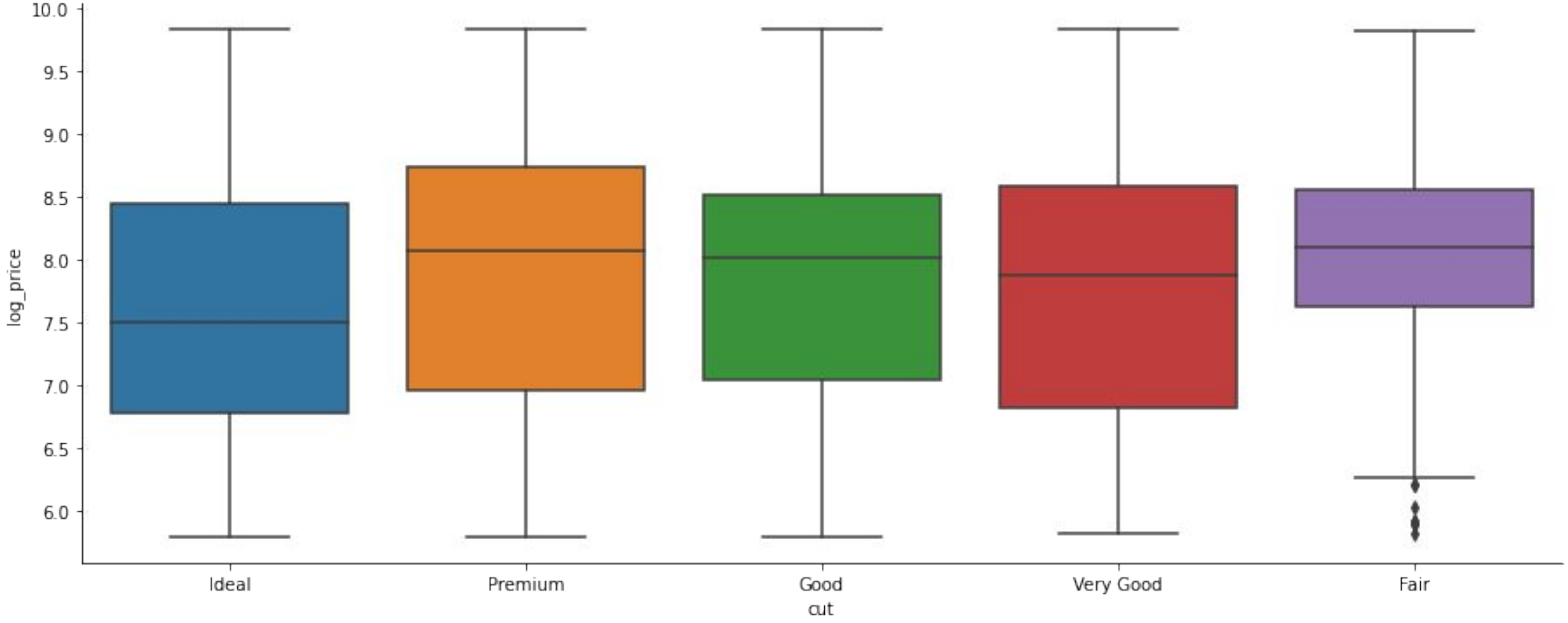
Features engineering

3. Since `'carat'`, `'price'`, and `'volume'` features are positively skewed, we used '**log**' from the numpy library to make them more normal (symmetrical distribution).

```
log_carat    = log(carat)
log_price    = log (price)
Log_volume   = log (volume)
```

****By replacing these features, the correlations become better.**

Boxplot between log_price and cut (Ref. slide 13)



Handling numerical features

1. Are there any missing values? No, clean :)

2. The min values of `'x'`, `'y'` and `'z'` equal to zero, it doesn't make any sense to have either for Length or Width or Height to be zero. Since there are only 20 samples of corrupted data, we can drop them as it seems like a better choice instead of filling them with any of Mean or Median.

```
data.isna().sum() #Clean :)
```

```
carat          0
cut            0
color          0
clarity        0
depth          0
table          0
price          0
x              0
y              0
z              0
table_width    0
z_depth        0
log_price      0
log_carat      0
dtype: int64
```

Handling categorical features

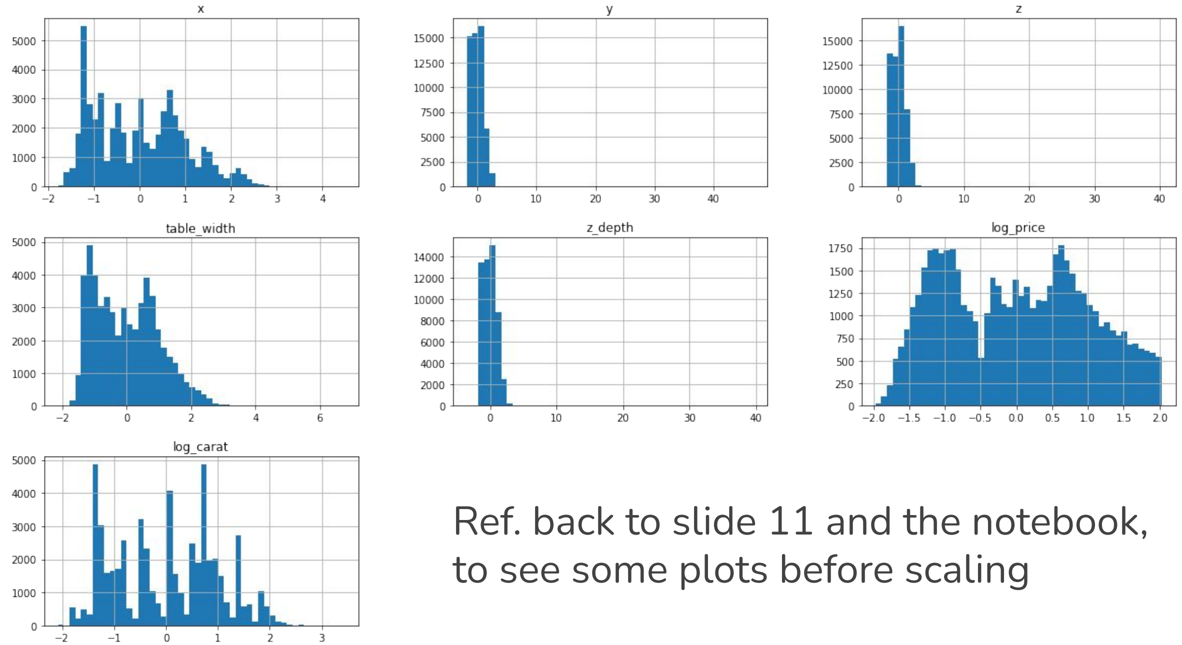
- After adding the new features and dropped `'carat'`, `'depth'`, `'table'`, and `'price'`. It's time to handle our categorical features.
- By using the idea of an ordinal encoder, we made our code. For example the feature 'Cut' contains `['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']` we replaced them by the numbers [1, 2, 3, 4, 5] respectively. Same thing for 'Clarity' and 'Color' features according the grade/scale of each feature.

Scaling features

- Our data has now become numerical, no categorical values! But we need to scale or normalize our data.
- The question is, which one should we use? Standardization or Normalization?
- Since there are some outliers in our data, so the best solution is using the Standardization technique.

Scaling features

- After scaling the data, we can see from the final result that there is a difference of measures and range.



Ref. back to slide 11 and the notebook, to see some plots before scaling

Feature Selection

In the 3 proposed approach, we used three different subsets of features:

- This subset of features: `{'cut', 'carat', 'clarity', 'color', 'depth', 'table', 'x', 'y', 'z', 'price', 'table_width', 'z_depth'}`
- The same subset adding: `'volume'`
- And the same subset adding: `'log_volume'`

Model selection

We chose to train the following regressors with and without Cross-Validation:

1.Random Forest Regressor 2.KNeighbors Regressor 3.Ridge Regressor 4.Bayesian Ridge

5.SGD Regressor 6.Lasso Regressor 7.Linear Regression 8.Support Vector Machine

9. Decision Tree Regressor 10.Gaussian Process Regression

Hence, we approached the problem in 4 ways by selecting three features subset and training the model with and without Cross-Validation. For model evaluation our selected metric is RMSE

1. Random Forest Regressor (first approach)

- RMSE = 0.08967

- Execution time = 173

2. KNeighbors Regressor (first approach)

- RMSE = 0.11113
- Execution time = 2.74s

3. Ridge Regressor (first approach)

- RMSE = 0.14454
- Execution time = 0.02s

4. Bayesian Ridge (first approach)

- RMSE = 0.14454

- Execution time = 0.017s

5. SGDRegressor (first approach)

- RMSE = 0.14845

- Execution time = 1.44s

6. Lasso Algorithm (first approach)

- RMSE = 0.14552

- Execution time = 0.9s

7. Linear Regression
(first approach)

- RMSE = 0.14454

- Execution time = 0.61s

8. Support Vector Machines (first approach)

- RMSE = 0.09985

- Execution time = 443s

9. Decision Tree Regressor (first approach)

- RMSE = 0.12356

- Execution time = 3.33s

10. Gaussian Process Regression (first approach)

The process couldn't be finished using colab.

Comparative Analysis

Model	*1st Approach* ***RMSE without CrossVal and without volume feature***	*2nd Approach* ***RMSE with CrossVal and with volume feature***	*3rd Approach* ***RMSE with CrossVal and without volume Feature***	*4th Approach* ***RMSE with CrossVal and with log_volume Feature***
Random Forest	**0.088016**	**0.105453**	**0.089668**	**0.031580**
K-Neighbors Regressor	0.088714	0.116059	0.111131	0.052686
Ridge Regression	0.144502	0.144752	0.144539	0.040014
Bayesian Ridge	0.144502	0.144751	0.144539	0.040007
SDG Regressor	0.143607	0.146769	0.148448	0.045405
Lasso Regressor	0.142013	0.145329	0.145524	0.041246
Linear Regression	0.141744	0.144751	0.144539	0.040007
SV-Regressor	0.097632	0.108312	0.099848	0.044036
Decision Tree Regressor	0.121775	0.140146	0.123556	0.037755

Model Fine-Tuning

- For each and every approach(without cross validation, with cross validation and without volume feature, with cross validation and with volume feature, and with cross validation and log_volume), our selected model is **Random Forest** which is the best among the build models based on RMSE. We used both **GridSearchCV** and **RandomizedSearch** to fine-tune the random forest regressor.
- Where after 5 very_very long successive attempts (based on the results of previous fine-tuning) that took us more than 15 minutes each time, just one attempt improved the RMSE slightly, however the rest made it worst.
- In the next slide, the results of fine-tuning are shown

Model Fine-Tuning Results

Finally, for each approach used, here is the best estimator with its parameters:

- Without cross validation and without volume feature:

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, **max_features=6,** max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators=200,** n_jobs=None, oob_score=False,random_state=42, verbose=0, warm_start=False)

- With cross validation and without volume feature:

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, **max_features=6,** max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators=200,** n_jobs=None, oob_score=False, random_state=42, verbose=0, warm_start=False)

Model Fine-Tuning Results

- With cross validation and with volume feature:

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, **max_features=4,** max_leaf_nodes=None, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators=200,** n_jobs=None, oob_score=False, random_state=42, verbose=0, warm_start=False)

- With cross validation and with log_volume feature:

RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse', max_depth=None, **max_features=2,** max_leaf_nodes=Non, max_samples=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, **n_estimators=200,** n_jobs=None, oob_score=False, random_state=42, verbose=0, warm_start=False)

^_^


My model on training data
My model on test dataset

Comparison between all the models

Model	*RMSE without CrossVal and without volume feature*	*RMSE with CrossVal and with volume feature*	*RMSE with CrossVal and without volume Feature*	*RMSE with CrossVal and with log_volume Feature*
Fine-tuned Model	**0.085602**	**0.101468**	**0.085602**	**0.028368**
Random Forest	**0.088016**	**0.105453**	**0.089668**	**0.031580**
K-Neighbors Regressor	0.088714	0.116059	0.111131	0.052686
Ridge Regression	0.144502	0.144752	0.144539	0.040014
Bayesian Ridge	0.144502	0.144751	0.144539	0.040007
SDG Regressor	0.143607	0.146769	0.148448	0.045405
Lasso Regressor	0.142013	0.145329	0.145524	0.041246
Linear Regression	0.141744	0.144751	0.144539	0.040007
SV-Regressor	0.097632	0.108312	0.099848	0.044036
Decision Tree Regressor	0.121775	0.140146	0.123556	0.037755

- **Extracted features**: `'table_width'` and `'z_depth'` highly improved the results, However `'volume'` degraded the performance of the regressors. For that the first subset of features is the best in this project.

- **Log transformation** applied to `'price'` and `'carat'` made their distribution more normal and lessened significantly the effect of outliers on regression

- **Random Forest** model was the best model using the 3 different approaches.

- **Cross-validation** increases slightly the rmse comparing to the results of the approach without cross-validation.

- **Log-Log regression** were used and evaluated with RMSE, however, comparing our the results of different is difficult using it.

Conclusions: findings of the project


THE END :)