Diamond Price

Machine Learning project

Done by :

Sondos Alarqan
Lamia Alariqi
Zaid Allwansah
Ahmad AlSheikh
Ghefar Shalhoub

TABLE OF CONTENTS

01 A bout the Data set

02 Step of the Project

03 Data Preparation and Exploration

04 Feature Engineering

05 Building Models

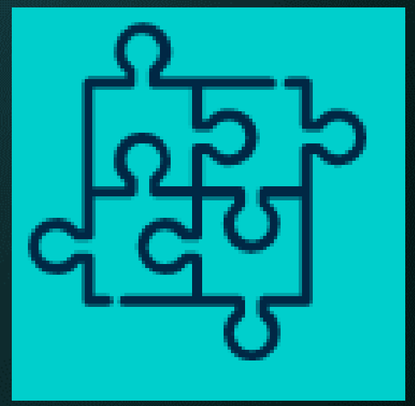
06 Testing & find predicted Prices

Describing The Dataset

The Diamond dataset is a well-known dataset that provides information about approximately 54,000 diamonds. It includes various attributes such as price ,carat weight, cut quality, color, clarity, dimensions (length, width, depth) ,and other features. This dataset is often used as a starting point for beginners in the field of data analysis and visualization.
Here is a breakdown of the attributesincluded in the dataset:

- Price: The price of the diamond in US dollars ($326—$18,823).
- Carat: The weight of the diamond ranging from 0.2 to 5.01 carats.
- Cut: The quality of the diamond's cut, categorized as Fair, Good, Very Good, Premium, or Ideal.
- Color: The diamond's color, graded from J (worst) to D (best).
- Clarity: A measure of how clear the diamond is, ranging from I1 (worst) to IF (best). It includes grades such as SI2, SI1, VS2, VS1, VVS2, VVS1.
- Dimensions: The physical dimensions of the diamond, including length (x), width (y), and depth (z) measured in millimeters (mm).
- Depth: The total depth percentage, which is calculated as the ratio of the depth (z) to the mean of length (x) and width (y). It provides information about the diamond's shape and proportions (43—79)
- Table: The width of the top of the diamond relative to its widest point (43—95).

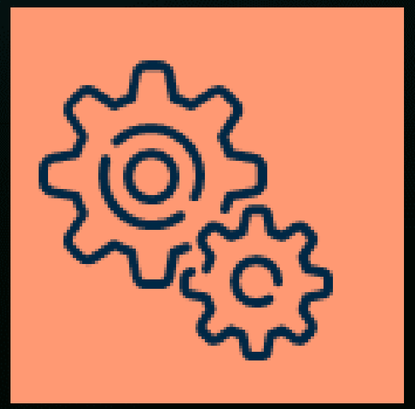
Step Of ML project



01

Data Preparation and Exploration:

a.Obtaining the data.
b.Exploring and visualizing the data structure.
c.Preparing the data for machine learning algorithms, including cleaning the data and encoding categorical features. And get rid off the outliers



02

Model Selection and Training:

a. Splitting the predictors and the label.
b. Creating a pipeline for the predictors.
c. Selecting and training different models, such as Linear Regression, Decision Tree, Random Forest, SVM,KNNand XGBRegressor

03

Model Evaluation and Fine-Tuning:

a. Evaluating the models using cross-validation
b. Fine-tuning the selected model(s) to improve performance.

Data Preparation and Exploration:

A bit of analysis before we start with the machine learning part

```
data=pd.read_csv('train.csv')
data
```

	Id	carat	cut	color	clarity	depth	table	price	x	y	z
0	1	1.06	Ideal	I	SI2	61.80	57.00	4270	6.57	6.60	4.07
1	2	1.51	Premium	G	VVS2	60.90	58.00	15164	7.38	7.42	4.51
2	3	0.32	Ideal	F	VS2	61.30	56.00	828	4.43	4.41	2.71
3	4	0.53	Ideal	G	VS2	61.20	56.00	1577	5.19	5.22	3.19
4	5	0.70	Premium	H	VVS2	61.00	57.00	2596	5.76	5.72	3.50
...	...	...	...	...	...	...	...	...	...	...	...
43147	43148	0.52	Ideal	E	VS2	61.50	56.00	1760	5.16	5.18	3.18
43148	43149	0.72	Very Good	D	VS2	62.10	59.00	3016	5.70	5.73	3.55
43149	43150	0.44	Premium	I	VVS1	61.50	58.00	990	4.95	4.87	3.02
43150	43151	0.31	Premium	E	VS1	60.20	58.00	734	4.38	4.43	2.65
43151	43152	0.33	Ideal	G	VS2	61.20	55.00	743	4.49	4.46	2.74

43152 rows × 11 columns

Data Preparation and Exploration:

Checking the Data type and handling null values :

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43152 entries, 0 to 43151
Data columns (total 18 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
 11  volume                  43152 non-null  float64
 12  density                 43152 non-null  float64
 13  surface_area            43152 non-null  float64
 14  aspect_ratio            43148 non-null  float64
 15  total_depth_percentage  43152 non-null  float64
 16  table_percentage        43152 non-null  float64
 17  symmetry                43148 non-null  float64
dtypes: float64(13), int64(2), object(3)
memory usage: 5.9+ MB
```

```python
data.isnull().sum()
```

```
Id                        0
carat                     0
cut                       0
color                     0
clarity                   0
depth                     0
table                     0
price                     0
x                         0
y                         0
z                         0
volume                    0
density                   0
surface_area              0
aspect_ratio              4
total_depth_percentage    0
table_percentage          0
symmetry                  4
dtype: int64
```

```python
[ ] data.replace([np.inf, -np.inf], np.nan, inplace=True)

    # Handle missing values (e.g., replace NaN with mean)
    data.fillna(data.median(), inplace=True)
```

Data Preparation and Exploration:

look for duplicates & drop (id) and the Correlation of the Data

```
[ ] data.duplicated().sum()
```

```
0
```

```
[ ] data=data.drop(['Id'],axis=1)
```

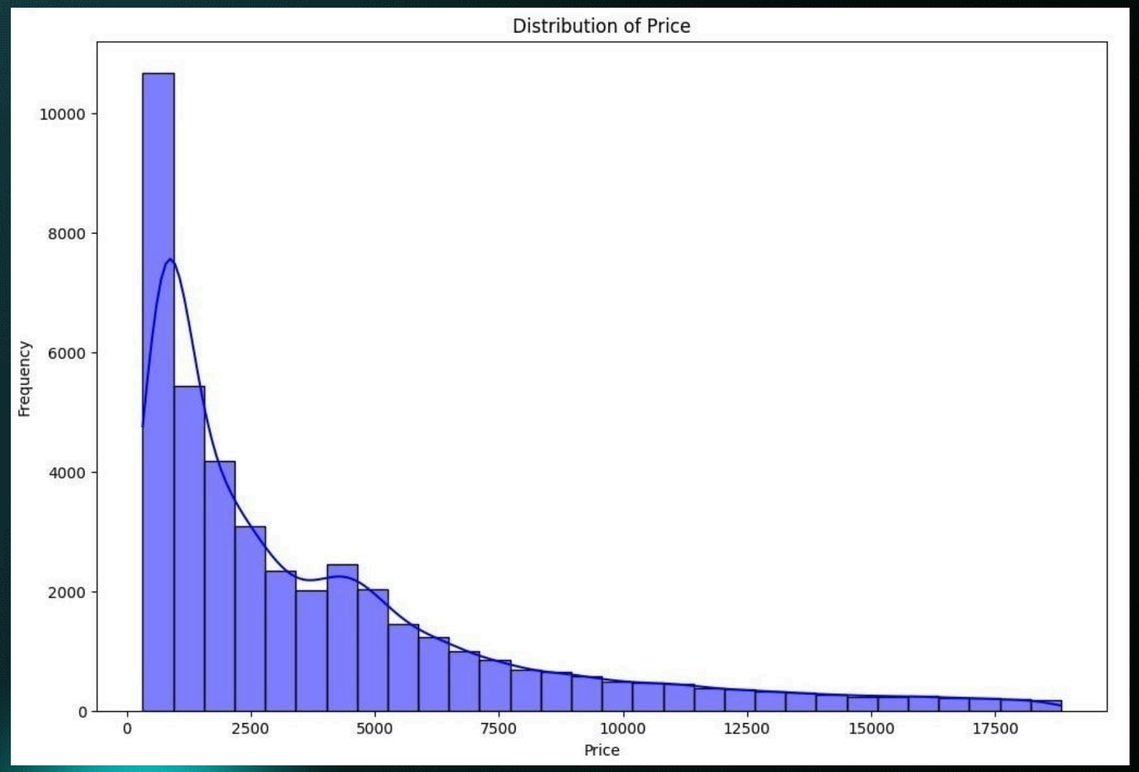
```
[ ] # correlation of the data
```

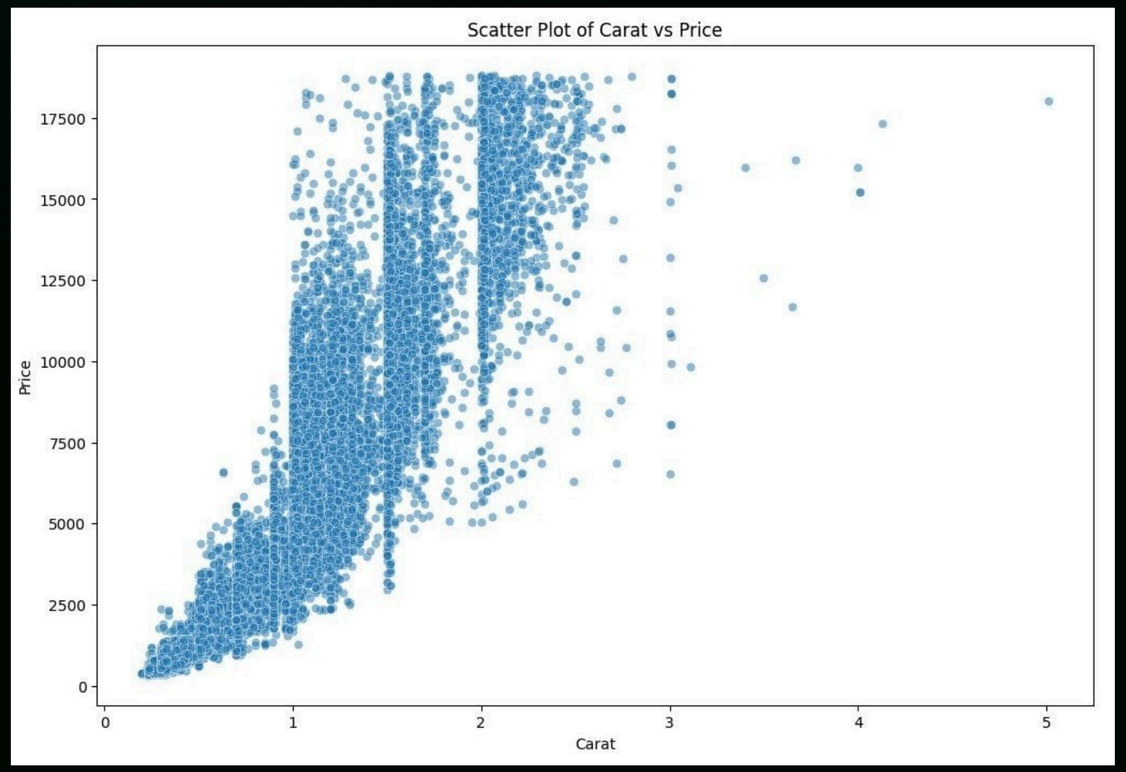
```
data.drop(['cut', 'color', 'clarity'],axis=1).corr()
```

	carat	depth	table	price	x	y	z
carat	1.000000	0.023944	0.182889	0.921911	0.975760	0.947060	0.948923
depth	0.023944	1.000000	-0.302794	-0.013137	-0.029601	-0.033354	0.090834
table	0.182889	-0.302794	1.000000	0.128501	0.197342	0.184310	0.150746
price	0.921911	-0.013137	0.128501	1.000000	0.885181	0.861354	0.857665
x	0.975760	-0.029601	0.197342	0.885181	1.000000	0.968954	0.965677
y	0.947060	-0.033354	0.184310	0.861354	0.968954	1.000000	0.942670
z	0.948923	0.090834	0.150746	0.857665	0.965677	0.942670	1.000000

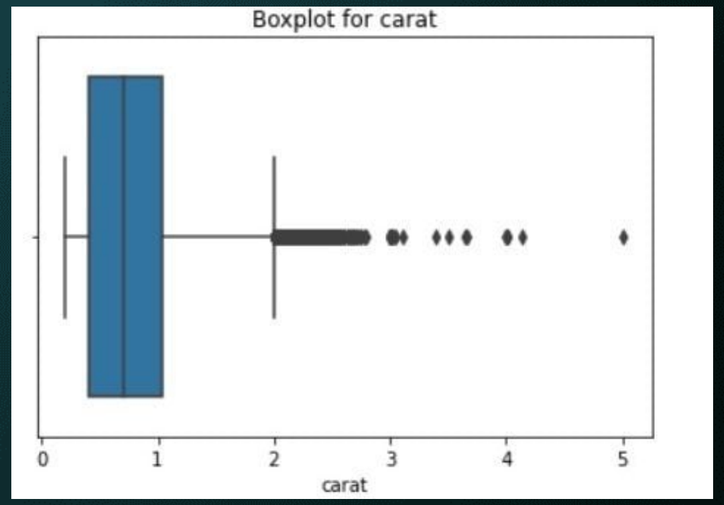
Data Preparation and Exploration:

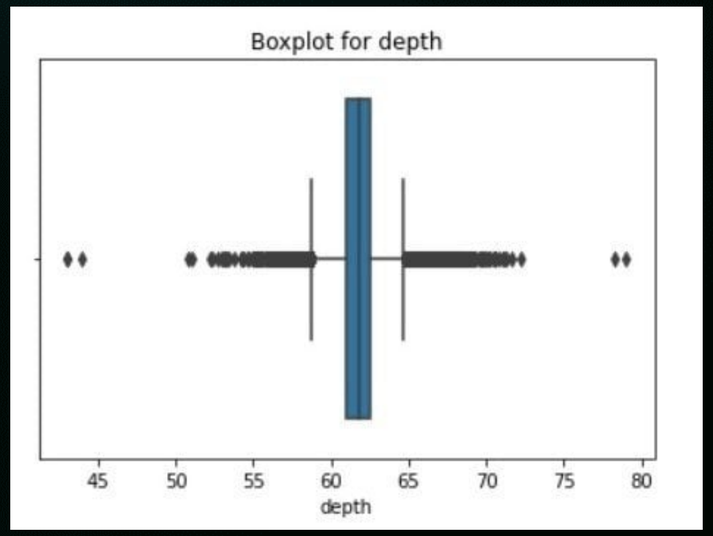
Making some Visualsations :

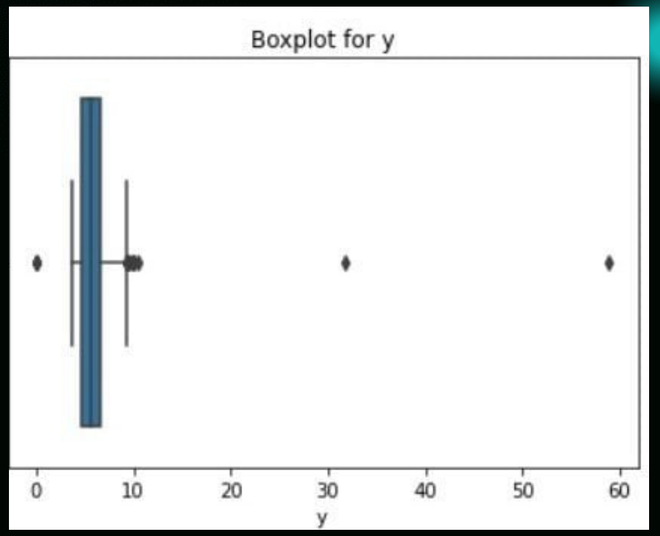


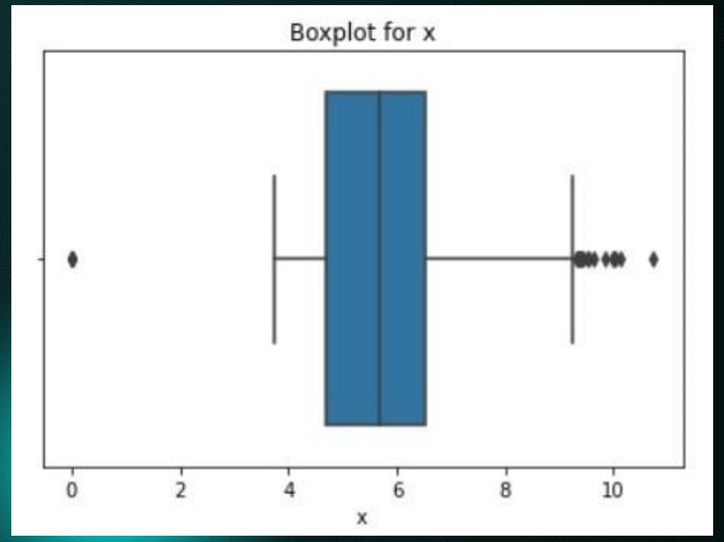


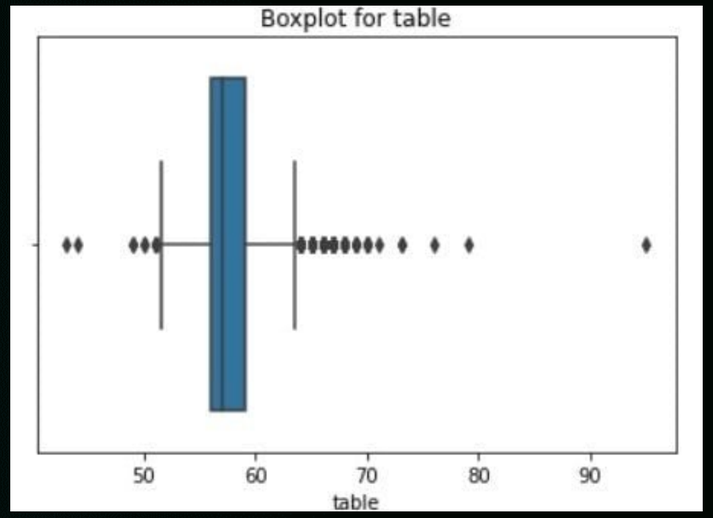
Boxplots :

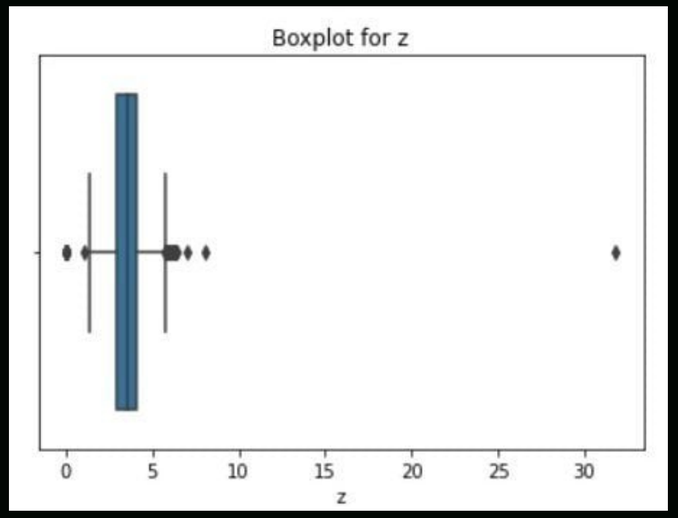

Boxplot for carat
0
1
2
3
4
5
carat


Boxplot for depth
45
50
55
60
65
70
75
80
depth


Boxplot for y
0
10
20
30
40
50
60
y


Boxplot for x
0
2
4
6
8
10
x


Boxplot for table
50
60
70
80
90
table


Boxplot for z
0
5
10
15
20
25
30
z

Feature Engineering

We've added some new features, standarized the numerical data and label encoded the categorical data so it can be trained in the model

```
data['volume'] = data['x'] * data['y'] * data['z']

# Calculate density
data['density'] = data['carat'] / data['volume']

# Calculate surface area
data['surface_area'] = 2 * (data['x'] * data['y'] + data['x'] * data['z'] + data['y'] * data['z'])

# Calculate aspect ratio
data['aspect_ratio'] = data['x'] / data['y']

# Calculate total depth percentage
data['total_depth_percentage'] = data['depth'] / data['z']

# Calculate table percentage
data['table_percentage'] = data['table'] / data[['x', 'y']].mean(axis=1)

# Calculate symmetry
data['symmetry'] = data['x'] / data['y'] - data['y'] / data['z']
```

Feature Engineering

Results After adding new features:

```
data
```

	carat	cut	color	clarity	depth	table	price	x	y	z	volume	density	surface_area	aspect_ratio	total_depth_percentage	table_percentage	symmetr
0	1.06	Ideal	I	SI2	61.8	57.0	4270	6.57	6.60	4.07	176.483340	0.006006	193.9278	0.995455	15.184275	8.656036	-0.62616
1	1.51	Premium	G	VVS2	60.9	58.0	15164	7.38	7.42	4.51	246.965796	0.006114	243.0152	0.994609	13.503326	7.837838	-0.65062
2	0.32	Ideal	F	VS2	61.3	56.0	828	4.43	4.41	2.71	52.943373	0.006044	86.9854	1.004535	22.619926	12.669683	-0.62277
3	0.53	Ideal	G	VS2	61.2	56.0	1577	5.19	5.22	3.19	86.422842	0.006133	120.5994	0.994253	19.184953	10.758886	-0.64211
4	0.70	Premium	H	VVS2	61.0	57.0	2596	5.76	5.72	3.50	115.315200	0.006070	146.2544	1.006993	17.428571	9.930314	-0.62729
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
43147	0.52	Ideal	E	VS2	61.5	56.0	1760	5.16	5.18	3.18	84.997584	0.006118	119.2200	0.996139	19.339623	10.831721	-0.63279
43148	0.72	Very Good	D	VS2	62.1	59.0	3016	5.70	5.73	3.55	115.946550	0.006210	146.4750	0.994764	17.492958	10.323710	-0.61932
43149	0.44	Premium	I	VVS1	61.5	58.0	990	4.95	4.87	3.02	72.801630	0.006044	107.5258	1.016427	20.364238	11.812627	-0.59615
43150	0.31	Premium	E	VS1	60.2	58.0	734	4.38	4.43	2.65	51.419010	0.006029	85.4998	0.988713	22.716981	13.166856	-0.68298
43151	0.33	Ideal	G	VS2	61.2	55.0	743	4.49	4.46	2.74	54.869596	0.006014	89.0968	1.006726	22.335766	12.290503	-0.62101

43152 rows × 17 columns

Replacing Null Values :

```
data.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
[ ] data.isnull().sum()
```

```
carat                     0
cut                       0
color                     0
clarity                   0
depth                     0
table                     0
price                     0
x                         0
y                         0
z                         0
volume                    0
density                  17
surface_area              0
aspect_ratio              4
total_depth_percentage   17
table_percentage          4
symmetry                 17
dtype: int64
```

```
[ ] for i in data:
        if data[i].isnull().sum()>0:
            data[i] = data[i].fillna(data[i].median())
```

Convert categorical data to numerical :

```
label_encoder = LabelEncoder()
```

```
[ ] data['color'] = label_encoder.fit_transform(data['color'])
    data['cut'] = label_encoder.fit_transform(data['cut'])
    data['clarity'] = label_encoder.fit_transform(data['clarity'])
```

```
ss=StandardScaler()
mm=MinMaxScaler()
rs=RobustScaler()
qt=QuantileTransformer()
```

```
[ ] data['carat'] =qt.fit_transform(data[['carat']])
    data['depth'] =qt.fit_transform(data[['depth']])
    data['table'] =qt.fit_transform(data[['table']])
    data['x'] =qt.fit_transform(data[['x']])
    data['y'] =qt.fit_transform(data[['y']])
    data['z'] =qt.fit_transform(data[['z']])
    data['volume'] =qt.fit_transform(data[['volume']])
    data['density'] =qt.fit_transform(data[['density']])
    data['surface_area'] =qt.fit_transform(data[['surface_area']])
    data['aspect_ratio'] =qt.fit_transform(data[['aspect_ratio']])
    data['total_depth_percentage'] =qt.fit_transform(data[['total_depth_percentage']])
    data['table_percentage'] =qt.fit_transform(data[['table_percentage']])
```

Results :

```
[ ] data
```

	carat	cut	color	clarity	depth	table	price	x	y	z	volume	density	surface_area	aspect_ratio	total_depth_percentage	table_percentage	symmetry
0	0.761762	2	5	3	0.486486	0.452953	4270	0.756757	0.772773	0.773273	0.777694	0.134591	0.774962	0.470470	0.224463	0.223789	-0.626167
1	0.905906	3	3	7	0.221722	0.627127	15164	0.909409	0.923423	0.902402	0.918462	0.487414	0.919347	0.412650	0.078116	0.086480	-0.650624
2	0.134635	2	2	5	0.320320	0.270771	828	0.156657	0.132633	0.122623	0.143576	0.237041	0.145163	0.614008	0.851137	0.843844	-0.622771
3	0.387888	2	3	5	0.290791	0.270771	1577	0.390891	0.389890	0.378879	0.395243	0.558465	0.395010	0.394394	0.603548	0.592593	-0.642111
4	0.479479	3	4	7	0.243243	0.452953	2596	0.531532	0.498999	0.489489	0.517894	0.324167	0.515809	0.756757	0.477256	0.468047	-0.627293
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
43147	0.373373	2	1	5	0.381882	0.270771	1760	0.377878	0.374875	0.371371	0.383743	0.501862	0.381810	0.510511	0.620039	0.603604	-0.632792
43148	0.532032	4	0	5	0.608609	0.769269	3016	0.504004	0.504004	0.524024	0.525526	0.799393	0.518449	0.420420	0.487865	0.523238	-0.619320
43149	0.307307	3	5	6	0.381882	0.627127	990	0.325325	0.309810	0.322823	0.323421	0.236250	0.324600	0.978504	0.674628	0.714642	-0.596156
43150	0.097598	3	1	4	0.120120	0.627127	734	0.121121	0.146146	0.049049	0.108599	0.192765	0.115493	0.067067	0.866000	0.914915	-0.682985
43151	0.161161	2	3	5	0.290791	0.122122	743	0.182182	0.164665	0.161662	0.177776	0.153803	0.176176	0.731732	0.818485	0.779771	-0.621011

43152 rows × 17 columns

Building Models

There was a lot of testing in this step, here we left some of the models we have tested and ultimately chose the best approach, which was to use a voting regressor.

```
[ ] regressors = [
        LinearRegression(),
        GammaRegressor(),
        SVR(),
        DecisionTreeRegressor(),
        RandomForestRegressor(),
        MLPRegressor(),
        AdaBoostRegressor(),
        PassiveAggressiveRegressor(),
        KNeighborsRegressor(),
        HistGradientBoostingRegressor(),
        LGBMRegressor(),
        ExtraTreeRegressor(),
        XGBRegressor()
    ]
    n=5
    acclist=[]
    acclist1=[]
    accs=dict()
    accs1=dict()
    for regressor in regressors:
        acclist=[]
        acclist1=[]
        for i in range(n):
            X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
            regressor.fit(X_train, y_train)
            y_pred = regressor.predict(X_test)
            acc = r2_score(y_test, y_pred)
            acc1 = np.sqrt(mean_squared_error(y_test, y_pred))
            acclist.append(acc)
            acclist1.append(acc1)
        accs[regressor]=[np.mean(acclist),np.std(acclist)]
        accs1[regressor]=[np.mean(acclist1),np.std(acclist1)]
    display(accs)
    display(accs1)
```

```
{LinearRegression(): [0.7245139160373613, 0.002275141898168109],
 GammaRegressor(): [0.39242531053324614, 0.007577460333722839],
 SVR(): [0.16891702227432506, 0.008344775291540393],
 DecisionTreeRegressor(): [0.964173633645594, 0.0019367712051872217],
 RandomForestRegressor(): [0.9815679226487216, 0.0009318016690780432],
 MLPRegressor(): [0.7598568495465536, 0.03910663827689101],
 AdaBoostRegressor(): [0.8865985110697956, 0.005303611988884473],
 PassiveAggressiveRegressor(): [0.6452622615678798, 0.009597352340553698],
 KNeighborsRegressor(): [0.9086784102185158, 0.003856604989844193],
 HistGradientBoostingRegressor(): [0.9817083002186389, 0.0007674388163625641],
 LGBMRegressor(): [0.9814124755178666, 0.0006607936415550459],
 ExtraTreeRegressor(): [0.9587493003094879, 0.002047285421774092],
 XGBRegressor(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...): [0.980392615312376,
  0.001639196264653788]}
{LinearRegression(): [2109.3213449300224, 25.16473104382102],
 GammaRegressor(): [3115.96642190468, 50.67546809537252],
 SVR(): [3648.206230711995, 53.01319841779426],
 DecisionTreeRegressor(): [749.9139701590295, 18.05055254633971],
 RandomForestRegressor(): [540.8699078314022, 14.847269918622299],
 MLPRegressor(): [1947.0518133476476, 167.4206650054957],
 AdaBoostRegressor(): [1343.5502625488855, 35.848485459327485],
 PassiveAggressiveRegressor(): [2381.153241317889, 43.2270439957962],
 KNeighborsRegressor(): [1210.2165761370932, 21.235600662079914],
 HistGradientBoostingRegressor(): [533.596784017983, 14.090643307570371],
 LGBMRegressor(): [546.150216190382, 12.454780519841597],
 ExtraTreeRegressor(): [815.1862457325749, 14.745531764453395],
 XGBRegressor(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...): [558.7203606304972,
  22.580283313517963]}
```

Building Models

```
param_grid = {
    'n_estimators': np.arange(600, 700, 20),
    'learning_rate': np.arange(0.04, 0.045, 0.001),
    'max_depth': np.arange(5, 8, 1),
    #'num_leaves': np.arange(20, 41, 5),
    #'min_child_samples': np.arange(10, 31, 5),
    #'reg_alpha': [0, 0.1, 0.5, 1.0],
    #'reg_lambda': [0, 0.1, 0.5, 1.0]
}
lgbm = LGBMRegressor()
grid_search = GridSearchCV(lgbm, param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(X_train1, y_train1)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'learning_rate': 0.041, 'max_depth': 7, 'n_estimators': 600}
```

```
[ ] param_grid = {
        'learning_rate': np.arange(0.1, 1, 0.1),
        'max_depth': np.arange(9, 19, 1),
    }
    gb = HistGradientBoostingRegressor()
    grid_search = GridSearchCV(gb, param_grid, cv=3, scoring='neg_mean_squared_error')
    grid_search.fit(X_train1, y_train1)
    best_params = grid_search.best_params_
    print("Best Parameters:", best_params)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 13}
```

```
param_grid = {
    'learning_rate': np.arange(0.1, 1, 0.1),
    'max_depth': np.arange(1, 10, 1),
}
xg = XGBRegressor()
grid_search = GridSearchCV(xg, param_grid, cv=3, scoring='neg_mean_squared_error')
grid_search.fit(X_train1, y_train1)
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 7}
```

Building Models

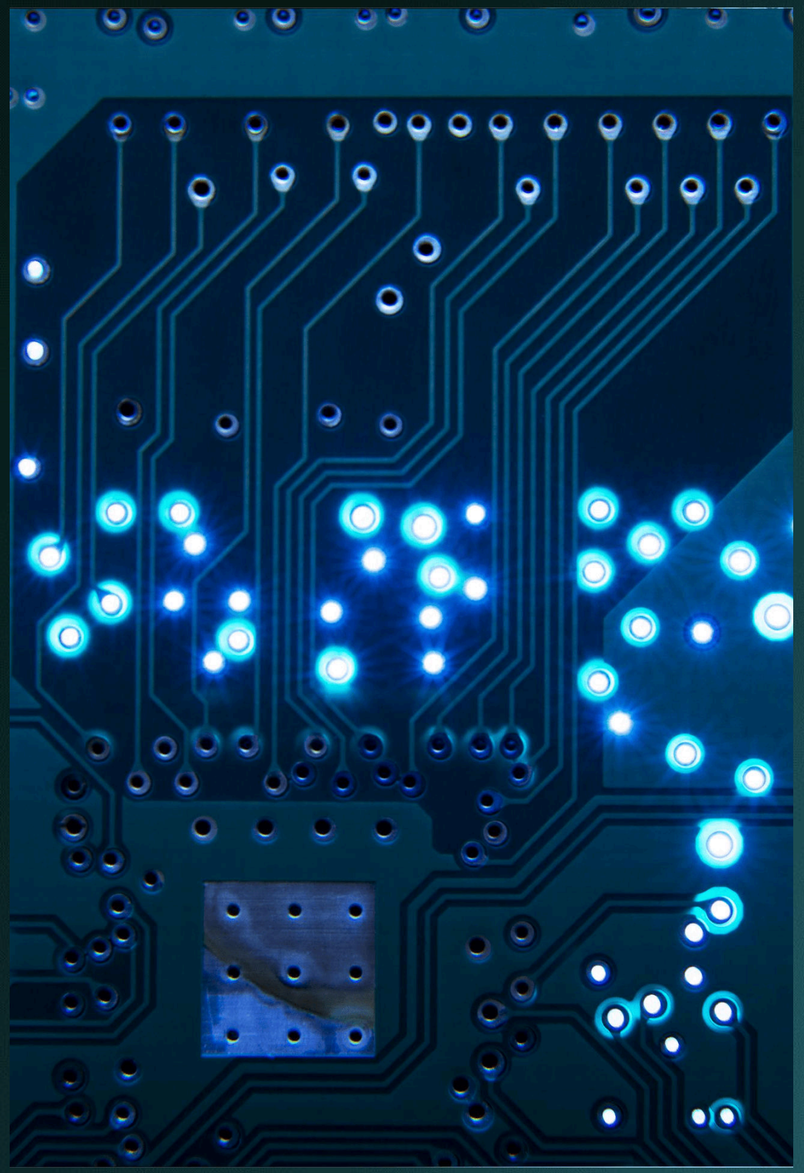
This is the final model we've used, we have added an extra model to increase the voting regressor's accuracy :

```
model1 = RandomForestRegressor()
model2 = HistGradientBoostingRegressor(learning_rate= 0.1,max_depth=9)
model3 = LGBMRegressor(learning_rate=0.041,max_depth=7, n_estimators=600)
model4 = XGBRegressor(learning_rate=0.1, max_depth= 7)
model5 = ExtraTreesRegressor(n_estimators=600)
voting_regressor = VotingRegressor([('rf', model1), ('gb', model2), ('lg', model3),('xg', model4),('etr',model5)])

# Step 4: Train the ensemble model
voting_regressor.fit(X_train1, y_train1)

# Step 5: Evaluate the model performance
y_pred = voting_regressor.predict(X_test1)
mse = mean_squared_error(y_test1, y_pred)
display(r2_score(y_pred,y_test1))
print("Mean Squared Error:", np.sqrt(mse))
```

Testing



Finally we will implement the model into a testing dataset

And adding the features we've added in the training data to the testing data :

```
[ ] test=pd.read_csv('test.csv')
    test
```

	Id	carat	cut	color	clarity	depth	table	x	y	z
0	1	0.34	Ideal	G	VVS2	61.1	57.0	4.52	4.48	2.75
1	2	0.71	Premium	E	VS2	62.7	58.0	5.74	5.68	3.58
2	3	0.44	Very Good	I	VS1	62.8	56.0	4.83	4.88	3.05
3	4	0.81	Premium	E	SI2	60.1	59.0	6.09	6.03	3.65
4	5	0.40	Ideal	G	VVS1	61.2	56.0	4.74	4.80	2.92
...	...	...	...	...	...	...	...	...	...	...
10783	10784	0.57	Ideal	H	VS1	60.9	56.0	5.34	5.36	3.26
10784	10785	1.05	Ideal	G	VS2	60.8	57.0	6.65	6.58	4.02
10785	10786	0.71	Ideal	E	VVS1	62.3	55.0	5.68	5.72	3.55
10786	10787	1.11	Premium	E	SI2	61.0	60.0	6.68	6.66	4.07
10787	10788	2.00	Premium	F	SI2	61.5	59.0	8.03	7.95	4.92

10788 rows × 10 columns

```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10788 entries, 0 to 10787
Data columns (total 17 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      10788 non-null  int64
 1   carat                   10788 non-null  float64
 2   cut                     10788 non-null  object
 3   color                   10788 non-null  object
 4   clarity                 10788 non-null  object
 5   depth                   10788 non-null  float64
 6   table                   10788 non-null  float64
 7   x                       10788 non-null  float64
 8   y                       10788 non-null  float64
 9   z                       10788 non-null  float64
 10  volume                  10788 non-null  float64
 11  density                 10788 non-null  float64
 12  surface_area            10788 non-null  float64
 13  aspect_ratio            10785 non-null  float64
 14  total_depth_percentage  10788 non-null  float64
 15  table_percentage        10788 non-null  float64
 16  symmetry                10785 non-null  float64
dtypes: float64(13), int64(1), object(3)
```

Replacing Nulls

```
[ ] test.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
[ ] for i in test:
        if test[i].isnull().sum()>0:
            test[i] = test[i].fillna(data[i].median())
```

```
test.isnull().sum()
```

```
Id                        0
carat                     0
cut                       0
color                     0
clarity                   0
depth                     0
table                     0
x                         0
y                         0
z                         0
volume                    0
density                   0
surface_area              0
aspect_ratio              0
total_depth_percentage    0
table_percentage          0
symmetry                  0
dtype: int64
```

```
test
```

	Id	carat	cut	color	clarity	depth	table	x	y	z	volume	density	surface_area	aspect_ratio	total_depth_percentage	table_percentage	symmetry
0	1	0.34	Ideal	G	VVS2	61.1	57.0	4.52	4.48	2.75	55.686400	0.006106	89.9992	1.008929	22.218182	12.666667	-4.758494
1	2	0.71	Premium	E	VS2	62.7	58.0	5.74	5.68	3.58	116.719456	0.006083	146.9736	1.010563	17.513966	10.157618	-5.315323
2	3	0.44	Very Good	I	VS1	62.8	56.0	4.83	4.88	3.05	71.889720	0.006120	106.3718	0.989754	20.590164	11.534501	-37.855824
3	4	0.81	Premium	E	SI2	60.1	59.0	6.09	6.03	3.65	134.037855	0.006043	161.9214	1.009950	16.465753	9.735974	-15.140103
4	5	0.40	Ideal	G	VVS1	61.2	56.0	4.74	4.80	2.92	66.435840	0.006021	101.2176	0.987500	20.958904	11.740042	-8.889328
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10783	10784	0.57	Ideal	H	VS1	60.9	56.0	5.34	5.36	3.26	93.309024	0.006109	127.0088	0.996269	18.680982	10.467290	-7.883731
10784	10785	1.05	Ideal	G	VS2	60.8	57.0	6.65	6.58	4.02	175.903140	0.005969	193.8832	1.010638	15.124378	8.616780	-7.902474
10785	10786	0.71	Ideal	E	VVS1	62.3	55.0	5.68	5.72	3.55	115.338080	0.006156	145.9192	0.993007	17.549296	9.649123	-40.116921
10786	10787	1.11	Premium	E	SI2	61.0	60.0	6.68	6.66	4.07	181.069416	0.006130	197.5652	1.003003	14.987715	8.995502	-8.136200
10787	10788	2.00	Premium	F	SI2	61.5	59.0	8.03	7.95	4.92	314.085420	0.006368	284.9202	1.010063	12.500000	7.384230	-12.636071

10788 rows × 17 columns

preprocessing

```
test['color'] = label_encoder.fit_transform(test['color'])
test['cut'] = label_encoder.fit_transform(test['cut'])
test['clarity'] = label_encoder.fit_transform(test['clarity'])
test['carat'] =qt.fit_transform(test[['carat']])
test['depth'] =qt.fit_transform(test[['depth']])
test['table'] =qt.fit_transform(test[['table']])
test['x'] =qt.fit_transform(test[['x']])
test['y'] =qt.fit_transform(test[['y']])
test['z'] =qt.fit_transform(test[['z']])
test['volume'] =qt.fit_transform(test[['volume']])
test['density'] =qt.fit_transform(test[['density']])
test['surface_area'] =qt.fit_transform(test[['surface_area']])
test['aspect_ratio'] =qt.fit_transform(test[['aspect_ratio']])
test['table_percentage'] =qt.fit_transform(test[['table_percentage']])
test['total_depth_percentage'] =qt.fit_transform(test[['total_depth_percentage']])
test['symmetry'] =qt.fit_transform(test[['symmetry']])
```

```
[ ] # using the model on the testing data
```

```
x=data.drop(['price','x','y','z'],axis=1)
y=data.price
model1 = RandomForestRegressor(n_estimators=600)
model2 = HistGradientBoostingRegressor(learning_rate=0.1,max_depth=13)
model3 = LGBMRegressor(learning_rate=0.041,max_depth=7, n_estimators=600)
model4 = XGBRegressor()
model6 = ExtraTreesRegressor(n_estimators=600)
voting_regressor = VotingRegressor([('rf', model1), ('gb', model2), ('lg', model3),('xg', model4),('etr',model6)])

# Step 4: Train the ensemble model
voting_regressor.fit(x,y)
test['price']=voting_regressor.predict(test.drop(['id','price','x','y','z'],axis=1))
test.rename(columns={'Id': 'id'}, inplace=True)
test50=test[['id','price']]
test50
```

Finally we got the Predicted diamond prices

	id	price
0	1	967.65
1	2	2878.67
2	3	805.20
3	4	2845.54
4	5	1106.48
...	...	...
10783	10784	1735.72
10784	10785	6499.96
10785	10786	4500.65
10786	10787	4734.08
10787	10788	13891.51

10788 rows × 2 columns