Diamond Price Prediction

Group :

- Mohamed Gamal
- Amira Djaiz
- Aya Abu Dabbur
- Mai Serry
- Dyaa Dwekat

Supervisor:

- Saad Altamari

Content

- Introduction
- Data Loading and Exploration
- Data Visualization
- Data Cleaning
- Feature Engineering and Data Preparation
- Preparing Model
- Fine-Tuning the Model
- Submission

Structure

- Step 01 — Data Loading and Exploration
- Step 02 — Data Visualization
- Step 03 — Data Cleaning
- Step 04 — Feature Engineering
- Step 05 — Preparing Model
- Step 06 — Fine-Tuning the Model
- Step 07 — Evalute Final Model
- Step 08 — Submission

Introduction

Introduction

Background

- Understanding the complexities of diamond pricing, which depends on the 4Cs (Carat, Cut, Color, Clarity), is essential. These attributes interact in complex ways, making accurate price prediction a challenging task.

Objective

- Our goal is to develop a robust machine-learning model that can predict diamond prices accurately. This involves thorough data analysis and selecting the best predictive techniques.

Data Loading and Exploration

Data Loading

The first step of the project involved loading the training data.

```
df = pd.read_csv('/kaggle/input/diamond-price-prediciton-2024/train.csv')
```

Data Exploration

df.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

df.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
```

Data Preprocessing

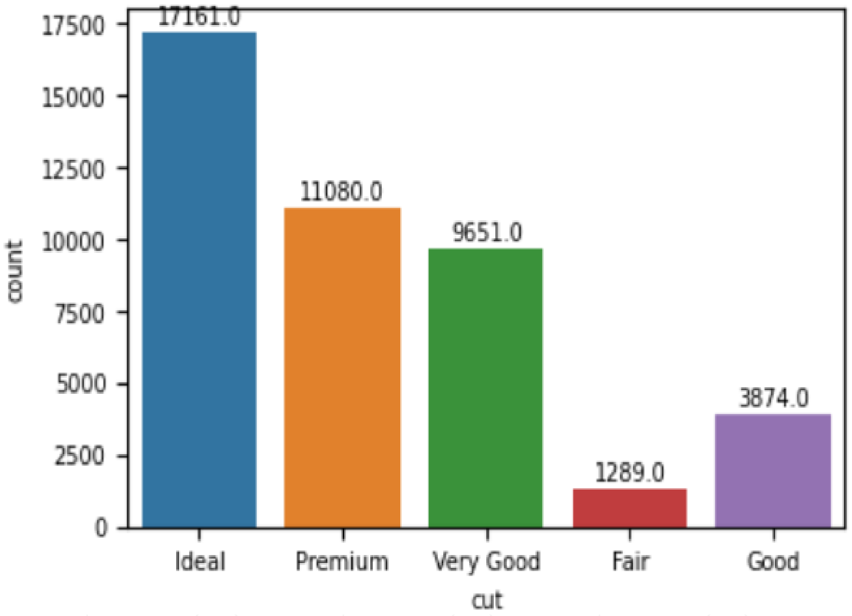
Removing Irrelevant Columns: The 'Id' column was determined to be unnecessary for our analysis and was thus removed from the dataset.

```
df_no_id = df.drop("Id", axis=1)
duplicates = df_no_id.duplicated()
num_duplicates = duplicates.sum()
num_duplicates
```

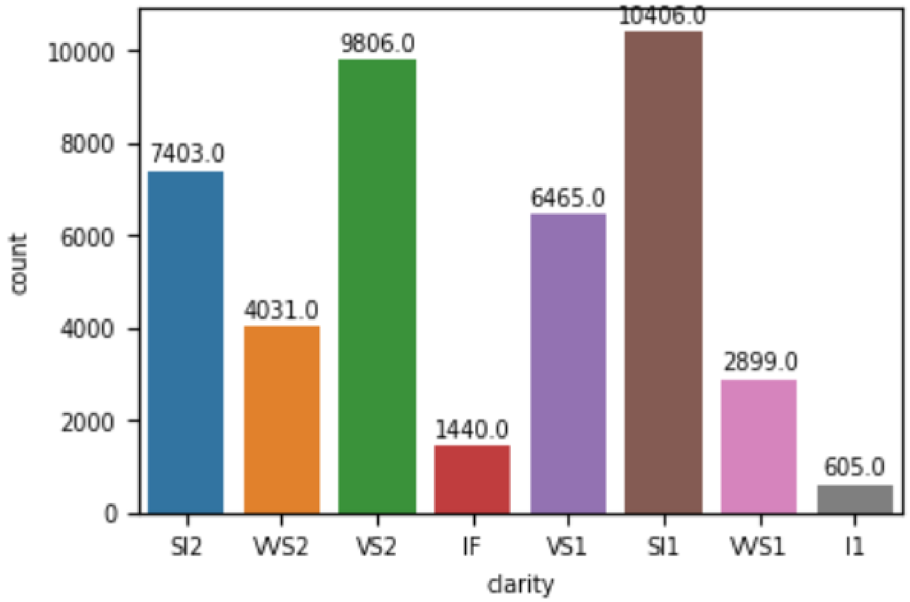
Data Visualization

Data Visualization

- **Countplot for Cut**

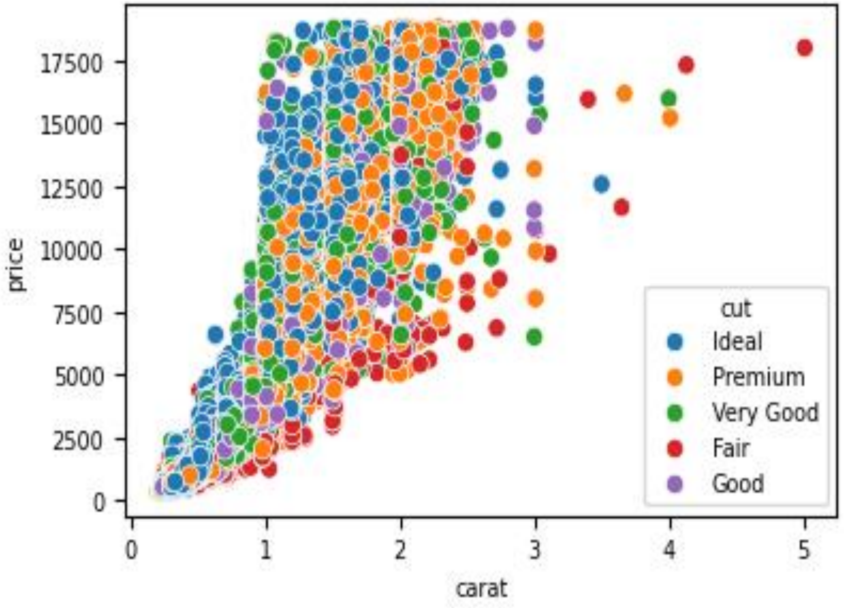


- **Countplot for Clarity**

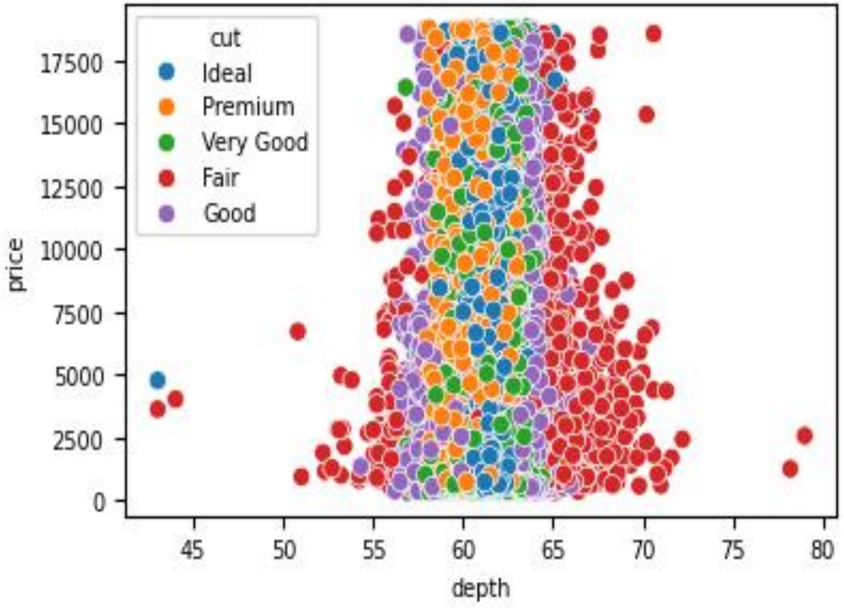


Data Visualization

- **Scatterplot for Carat vs. Price**:

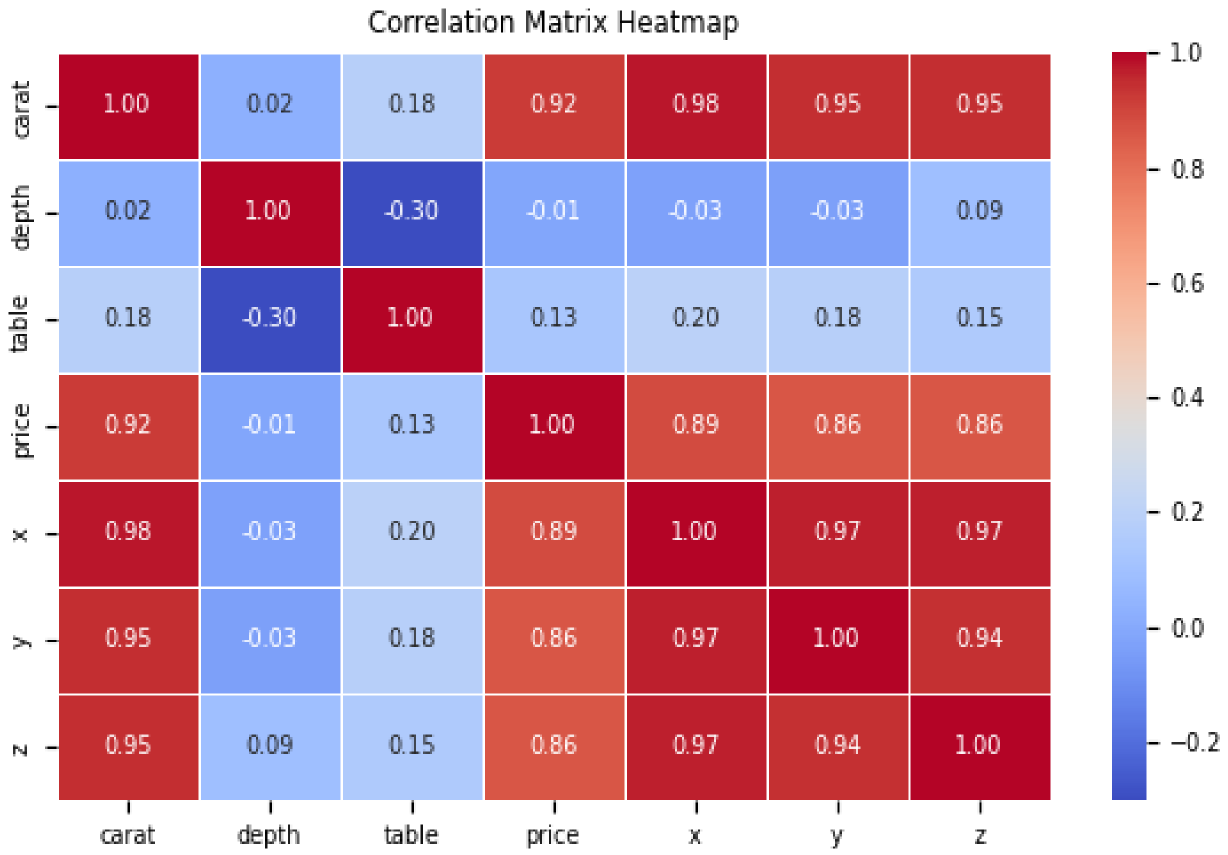


- **Scatterplot for Depth vs. Price**:



Data Visualization

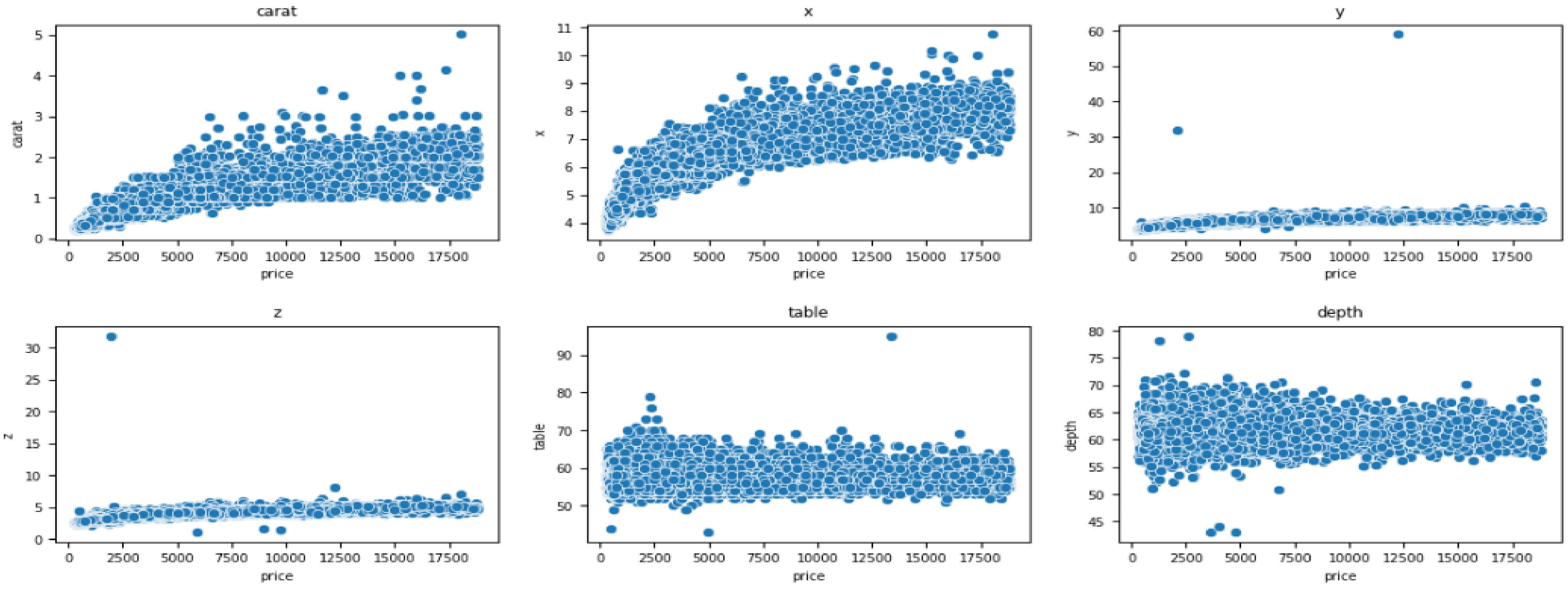
Heatmap for Feature Correlation
(for numeric features)



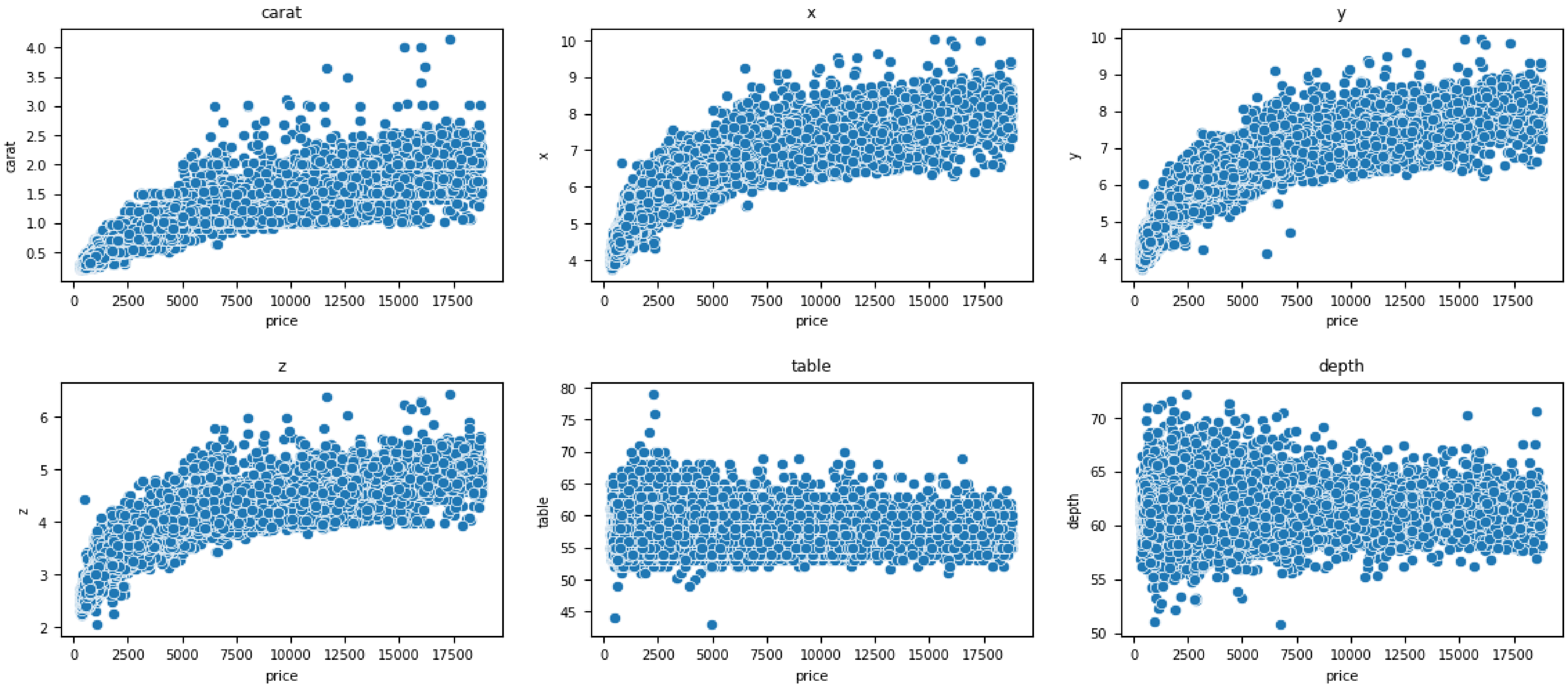
Data Cleaning

Data Cleaning

Handling Outliers



Data Cleaning



Feature Engineering and Data Preparation

Feature Engineering

01 Feature Engineering

02 Adding Volume Feature

03 Dropping Columns

04 Ordinal Encoding

Feature Engineering

enc_processed_df.corr()

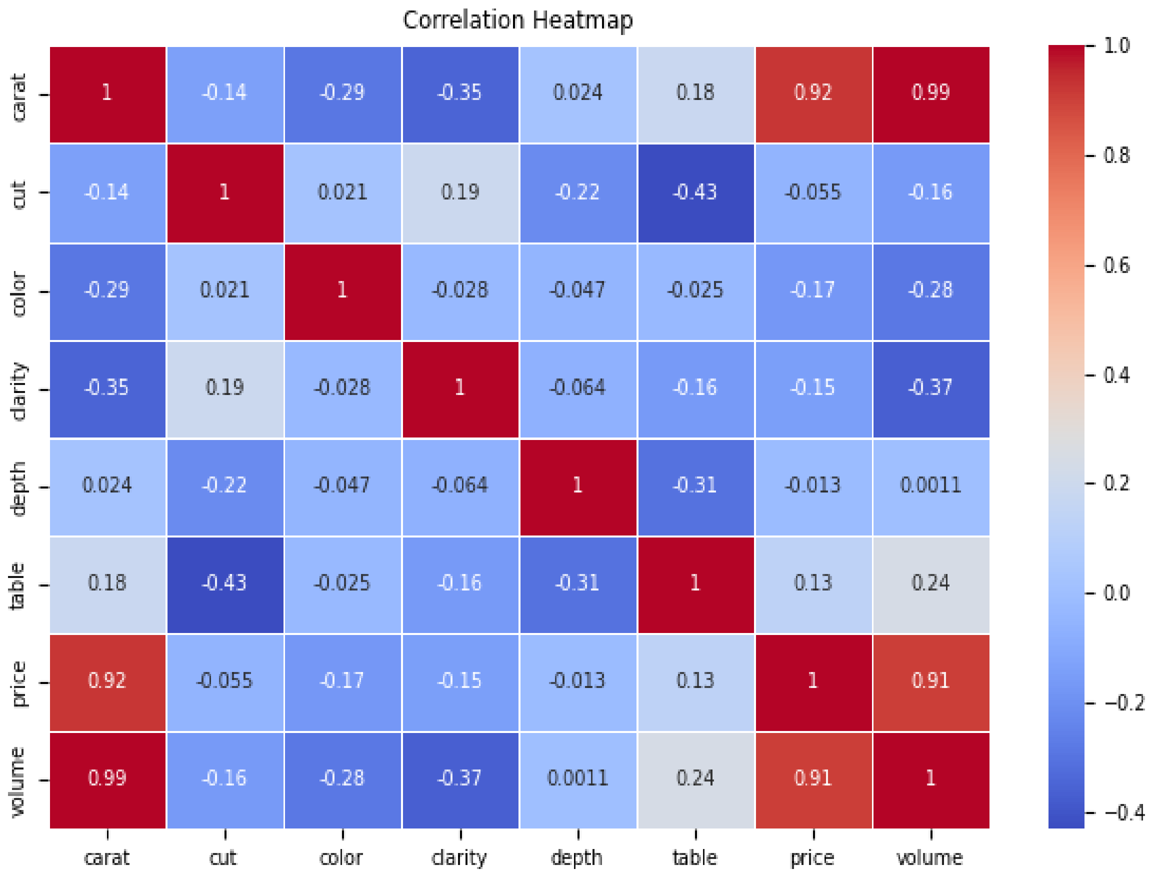
```
price       1.000000
carat       0.922247
volume      0.909367
table       0.127605
depth      -0.013130
cut        -0.055436
clarity    -0.146704
color      -0.170819
Name: price, dtype: float64
```

enc_processed_df.head()

	carat	cut	color	clarity	depth	table	price	volume
0	1.06	4.0	1.0	1.0	61.8	57.0	4270	740.430017
1	1.51	3.0	3.0	5.0	60.9	58.0	15164	957.036622
2	0.32	4.0	4.0	3.0	61.3	56.0	828	304.412538
3	0.53	4.0	3.0	3.0	61.2	56.0	1577	432.648888
4	0.70	3.0	2.0	5.0	61.0	57.0	2596	542.129615

Data Exploration

Correlation Matrix: Calculated the correlation matrix to examine the relationships between different features.



Preparing Models

Preparing Model

- **Custom Transformer for Feature Engineering**

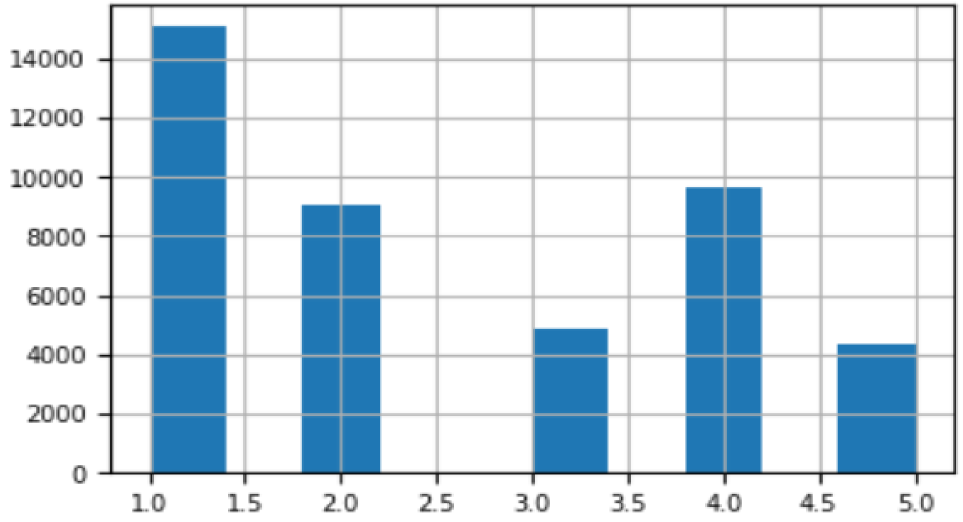
We define a custom transformer CombinedAttributesAdder to generate a new feature, volume, based on the diamond dimensions:

```
from sklearn.base import BaseEstimator, TransformerMixin

x_ix, y_ix, z_ix, table_ix = [
    list(df.drop(["cut", "color", "clarity", "price"], axis=1).columns).index(col
    for col in ("x", "y", "z", "table")]

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
```

Preparing Model

Splitting the Data

```
df.loc[:, "carat_cat"] = pd.cut(df["carat"],
                                bins=[0.19, 0.5, 0.75, 1, 1.5, np.inf],
                                labels=[1, 2, 3, 4, 5])
df["carat_cat"].hist();
```




```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=10, test_size=0.2, random_state=42)

for train_index, test_index in split.split(df, df["carat_cat"]):
    train_set = df.iloc[train_index]
    test_set = df.iloc[test_index]
```

Preparing Model

Numerical Pipeline

```
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('attribs_adder', CombinedAttributesAdder()),
    ('std_scaler', StandardScaler()),
])
```

Full Pipeline

```
full_pipeline = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", OrdinalEncoder(categories=[colors_cats, cuts, clarities]), cat_attribs)
])
```

Data Transformation

```
x_train_prepared = full_pipeline.fit_transform(x_train)
x_test_prepared = full_pipeline.transform(x_test)
```

```python
from sklearn.linear_model import LinearRegression
linear_model=LinearRegression()
linear_model.fit(x_train_prepared,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```python
linear_model.score(x_train_prepared, y_train)
```

```
0.9186449950483981
```

Linear Regression model

We evaluate the model's performance on the training set.

Linear Regression model

Calculating RMSE on Training Data.

```
from sklearn.metrics import mean_squared_error

lin_predic = linear_model.predict(x_train_prepared)
lin_rmse = mean_squared_error(
    y_train,
    lin_predic,
    squared=False  #RMSE
)
lin_rmse
```

```
1133.7829679831411
```

Linear Regression model

use cross-validation to evaluate the model's performance.

```python
from sklearn.model_selection import cross_val_score

lin_scores = cross_val_score(
  linear_model,
  x_train_prepared,
  y_train,
  scoring ="neg_mean_squared_error",
  cv = 10
)
lin_rmse_scores = np.sqrt(-lin_scores)
print("Scores: ", lin_rmse_scores)
print("Mean: ", lin_rmse_scores.mean())
print("Standard Deviation: ", lin_rmse_scores.std())
```

```
Scores:  [1145.17612289 1137.04199626 1101.15552983 1208.70921887 1142.37399683
 1110.04639488 1143.66724006 1144.35294896 1098.31134199 1148.04160739]
Mean:  1137.887639796304
Standard Deviation:  29.93097691246276
```

```python
from sklearn.tree import DecisionTreeRegressor
tree_model=DecisionTreeRegressor(random_state = 42)
tree_model.fit(x_train_prepared,y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor(random_state=42)
```

```python
tree_model.score(x_train_prepared, y_train)
```

```
0.9999958224864528
```

Decision Tree Regressorion Model

We evaluate the model's performance on the training set.

Decision Tree Regressorion Model

Calculating RMSE on Training Data.

```
tree_predic = tree_model.predict(x_train_prepared)
tree_rmse = mean_squared_error(
    y_train,
    tree_predic,
    squared=False
)
tree_rmse
```

```
8.142529435502995
```

```
tree_scores = cross_val_score(
  tree_model,
  x_train_prepared,
  y_train,
  scoring ="neg_mean_squared_error",
  cv = 10
)
tree_rmse_scores = np.sqrt(-tree_scores)
print("Scores: ", tree_rmse_scores)
print("Mean: ", tree_rmse_scores.mean())
print("Standard Deviation: ", tree_rmse_scores.std())
```

```
Scores:  [742.84381883 703.8662325  764.20661659 697.67372586 772.21417935
 757.46766405 754.37957042 720.71381911 695.87127516 730.35974988]
Mean:  733.9596651750455
Standard Deviation:  26.99672946198288
```

Decision Tree Regression Model

use cross-validation to evaluate the model's performance.

```python
from sklearn.ensemble import RandomForestRegressor

forest_model = RandomForestRegressor(n_estimators=100, random_state=42)
forest_model.fit(x_train_prepared, y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```python
forest_model.score(x_train_prepared, y_train)
```

```
0.9972230352470094
```

RandomForest Regressor Model

We evaluate the model's performance on the training set.

RandomForest Regressor Model

Calculating RMSE on Training Data.

```
forest_predic = forest_model.predict(x_train_prepared)
forest_rmse = mean_squared_error(
    y_train,
    forest_predic,
    squared=False
)
forest_rmse
```

```
204.63439946095136
```

```python
forest_scores = cross_val_score(
  forest_model,
  x_train_prepared,
  y_train,
  scoring ="neg_mean_squared_error",
  cv = 10
)
forest_rmse_scores = np.sqrt(-forest_scores)
print("Scores: ", forest_rmse_scores)
print("Mean: ", forest_rmse_scores.mean())
print("Standard Deviation: ", forest_rmse_scores.std())
```

```
Scores:  [563.51686332 536.22530208 575.45252782 536.89516107 558.59890765

 563.49796513 558.90275627 519.22794338 527.196818   563.58505227]

Mean:  550.3099296989064

Standard Deviation:  17.82669031675242
```

RandomForest Regressor Model

use cross-validation to evaluate the model's performance.

Fine-Tuning the Model

Fine-Tuning the Model

```
from sklearn.model_selection import RandomizedSearchCV

param_distribs = {
  'n_estimators': np.random.randint(1, 200, 10),
  'max_features': np.random.randint(1, 8, 10),
}

forest_model_fine = RandomForestRegressor(random_state=42)
rnd_search = RandomizedSearchCV(
  forest_model_fine,
  param_distributions=param_distribs,
  n_iter=10,
  cv=5,
  scoring='neg_mean_squared_error',
  random_state=42
)
rnd_search.fit(x_train_prepared, y_train)
```

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(random_state=42),
,                  param_distributions={'max_features': array([7, 7, 2, 4, 1, 2, 1, 4, 1, 2]),
,                                       'n_estimators': array([ 72, 108,  92,  24,  68,   1,  22,  46,  62,  85])},
,                  random_state=42, scoring='neg_mean_squared_error')
```

```python
cvres = rnd_search.cv_results_
for mean_score, params in zip(cvres["mean_test_score"], cvres["params"]):
    print(np.sqrt(-mean_score), params)
```

```
666.0574858358463 {'n_estimators': 24, 'max_features': 1}
581.8702387052098 {'n_estimators': 24, 'max_features': 2}
548.3524475625106 {'n_estimators': 72, 'max_features': 4}
1059.9121520738188 {'n_estimators': 1, 'max_features': 1}
644.1725377061641 {'n_estimators': 68, 'max_features': 1}
547.5641124850662 {'n_estimators': 85, 'max_features': 4}
564.4709532633339 {'n_estimators': 92, 'max_features': 2}
643.6625766840698 {'n_estimators': 72, 'max_features': 1}
555.139398111733 {'n_estimators': 72, 'max_features': 7}
555.139398111733 {'n_estimators': 72, 'max_features': 7}
```

```python
final_model = rnd_search.best_estimator_
final_model
```

```
RandomForestRegressor(max_features=4, n_estimators=85, random_state=42)
```

Fine-Tuning the Model

The best-performing set of parameters is {'n_estimators': 72, 'max_features': 4}which resulted in the lowest RMSE of 548.35.

Evalute Final Model

```python
# final_predictions = final_model.predict(x_test_prepared)
final_predictions = final_model.predict(x_train_prepared)


# final_mse = mean_squared_error(y_test, final_predictions)
final_mse = mean_squared_error(y_train, final_predictions)
final_rmse = np.sqrt(final_mse)
final_rmse
```

```
202.77888794641905
```

```python
from scipy import stats

confidence = 0.95
squared_errors = (final_predictions - y_train) ** 2
np.sqrt(stats.t.interval(confidence, len(squared_errors) - 1,
                         loc=squared_errors.mean(),
                         scale=stats.sem(squared_errors)))
```

```
array([198.18512292, 207.27086589])
```

Evalute Final Model

Calculate the confidence interval for the Root Mean Square Error (RMSE) of the final model's predictions.

Submission

Submission

the first few rows of our submission file:

```
data={'Id':Id,'price':price}
sub=pd.DataFrame(data)
filename=f'mg_sub_{int(round(final_rmse))}.csv'
print(filename)
print(sub)
sub.to_csv(filename, index=False)
```

Id	price
1	839.1597
2	2905.8908
3	875.2437
4	2753.8571
5	1054.3025
...	...
10786	4347.8571
10787	4786.2605
10788	13905.1765