Author: David Luna Naranjo

Date: 21-08-2023

Libraries

In [...

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Intro

> This analysis focuses on conducting a comprehensive exploratory data analysis (EDA) of the provided dataset. The goal is to inform decision-making and recommendations before entering the modeling phase. Through this EDA, we aim to uncover data patterns, relationships, and anomalies that could impact subsequent modeling. The insights gained will guide preprocessing steps to improve model performance.

Data

In this case, it is a database of quality and price of zircons (precious stones)

▸ Variable Information

Load Data

In [...

```
df = pd.read_excel('../data/raw/diamonds.xlsx',sheet_na
df.head(5)
```

Out[…

	carat	cut	color	clarity	depth	table	price	x	y
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2

Descriptive Statistics

Dataframe missing value check.

In […

```
df.info()
```

```
<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
```

Summary statistics of the numeric variables (mean, median, standard deviation, etc.).

In […

```
df.describe().T.sort_values(by='std', ascending=False)
```

Out[…

	count	mean	std	min	25%	50%
price	53940.0	3932.799722	3989.439738	326.0	950.00	2401.00
table	53940.0	57.457184	2.234491	43.0	56.00	57.00
depth	53940.0	61.749405	1.432621	43.0	61.00	61.80
y	53940.0	5.734526	1.142135	0.0	4.72	5.71
x	53940.0	5.731157	1.121761	0.0	4.71	5.70
z	53940.0	3.538734	0.705699	0.0	2.91	3.53
carat	53940.0	0.797940	0.474011	0.2	0.40	0.70

Min - Max and Count

In […

```
# Get top 5 value counts for each column in `df`.
counts = pd.Series({ft: [df[ft].value_counts().round(3)

# Extract min and max values for each numeric column.
min_max = df.describe().T[['min', 'max']]
# Merge the min, max, and top 5 value counts data.
stats_pivot = pd.concat([min_max, counts], axis=1)
stats_pivot.style.background_gradient()
```

Out[…

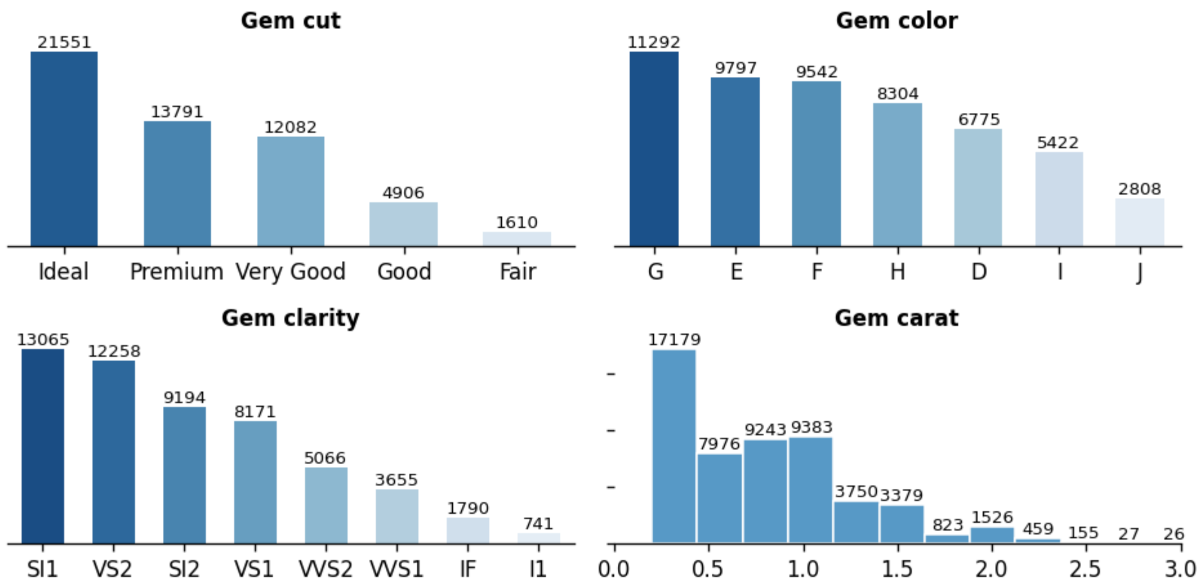
	min	max	Top 5
carat	0.200000	5.010000	[{0.3: 2604, 0.31: 2249, 1.01: 2242, 0.7: 1981, 0.32: 1840}]
depth	43.000000	79.000000	[{62.0: 2239, 61.9: 2163, 61.8: 2077, 62.2: 2039, 62.1: 2020}]
table	43.000000	95.000000	[{56.0: 9881, 57.0: 9724, 58.0: 8369, 59.0: 6572, 55.0: 6268}]
price	326.000000	18823.000000	[{605: 132, 802: 127, 625: 126, 828: 125, 776: 124}]
x	0.000000	10.740000	[{4.37: 448, 4.34: 437, 4.33: 429, 4.38: 428, 4.32: 425}]
y	0.000000	58.900000	[{4.34: 437, 4.37: 435, 4.35: 425, 4.33: 421, 4.32: 414}]
z	0.000000	31.800000	[{2.7: 767, 2.69: 748, 2.71: 738, 2.68: 730, 2.72: 697}]
cut	nan	nan	[{'Ideal': 21551, 'Premium': 13791, 'Very Good': 12082, 'Good': 4906, 'Fair': 1610}]
color	nan	nan	[{'G': 11292, 'E': 9797, 'F': 9542, 'H': 8304, 'D': 6775}]
clarity	nan	nan	[{'SI1': 13065, 'VS2': 12258, 'SI2': 9194, 'VS1': 8171, 'VVS2': 5066}]

4Cs of Diamond Quality (Cut, Color, Clarity, Carat)

In […

```
# List of categorical variables
categories = ['cut', 'color', 'clarity']

# Create subplots with 2 rows and 2 columns
fig, ax = plt.subplots(2, 2, figsize=(8, 4), dpi=120)
ax = ax.ravel()

# Iterate through categories and create count plots
```

```python
for i, category in enumerate(categories):
    # Create a count plot for the current category
    s = sns.countplot(data=df, x=category, order=df[cat

    # Set title, labels, and styling for the count plot
    ax[i].set_title(f'Gem {category}', ha='center', for
    ax[i].set_yticks([])
    for container in s.containers:
        s.bar_label(container, c='black', size=8)
        s.set_ylabel('')
        s.spines['top'].set_visible(False)
        s.set_xlabel('')
        s.spines['right'].set_visible(False)
        s.spines['left'].set_visible(False)
        plt.tick_params(labelleft=False)

# Create a histogram for the 'carat' variable
s = sns.histplot(data=df, x='carat', bins=20, ax=ax[3],
ax[3].set_title(f'Gem carat', ha='center', fontweight='
for container in s.containers:
    s.bar_label(container, c='black', size=8)
    s.set_ylabel('')
    s.spines['top'].set_visible(False)
    s.set_xlabel('')
    s.spines['right'].set_visible(False)
    s.spines['left'].set_visible(False)
    plt.tick_params(labelleft=False)
ax[3].set_xlim(0, 3)

# Adjust layout
fig.tight_layout()
```



Findings:

- A significant proportion of cubic zirconia gems in the dataset weigh less than 1 carat.
- The dataset is dominated by cubic zirconia gems that exhibit a colorless or nearly colorless appearance.
- Cubic zirconia gems with an Ideal or Premium cut are widely represented in the dataset.
- The most prevalent clarity grades among the cubic zirconia gems in the dataset are SI1, VS1, and VS2. This suggests that the gems in the dataset are generally of high quality.

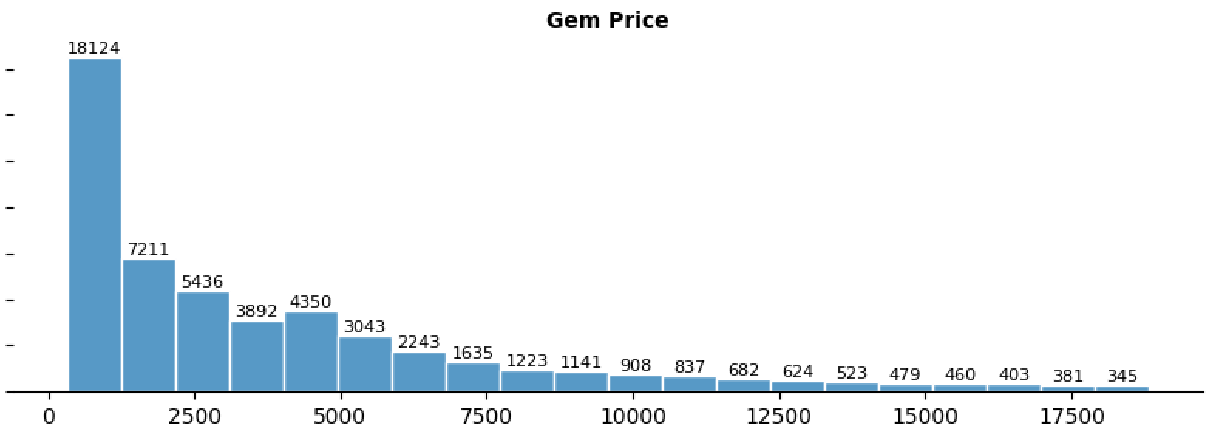
Data visualization

Price distribution

In [...

```
# Create a single subplot
fig, ax = plt.subplots(1, 1, figsize=(8, 3))

# Create a histogram for the 'price' variable
s = sns.histplot(data=df, x='price', bins=20, ax=ax, ec
ax.set_title(f'Gem Price', ha='center', fontweight='bol
for container in s.containers:
    s.bar_label(container, c='black', size=8)
```

```
    s.set_ylabel('')
    s.spines['top'].set_visible(False)
    s.set_xlabel('')
    s.spines['right'].set_visible(False)
    s.spines['left'].set_visible(False)
    plt.tick_params(labelleft=False)

# Adjust layout
fig.tight_layout()
```



In [...

```
pd.crosstab(df.color, df.clarity).style.background_grac
```

Out[...

clarity	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2
color								
D	42	73	2083	1370	705	1697	252	553
E	102	158	2426	1713	1281	2470	656	991
F	143	385	2131	1609	1364	2201	734	975
G	150	681	1976	1548	2148	2347	999	1443
H	162	299	2275	1563	1169	1643	585	608
I	92	143	1424	912	962	1169	355	365
J	50	51	750	479	542	731	74	131

In [...

```
pd.crosstab(df.color, df.cut).style.background_gradient
```

Out[...

color \ cut	Fair	Good	Ideal	Premium	Very Good
D	163	662	2834	1603	1513
E	224	933	3903	2337	2400
F	312	909	3826	2331	2164
G	314	871	4884	2924	2299
H	303	702	3115	2360	1824
I	175	522	2093	1428	1204
J	119	307	896	808	678

In [...

```
pd.crosstab(df.clarity, df.cut).style.background_gradie
```

Out[...

clarity \ cut	Fair	Good	Ideal	Premium	Very Good
I1	210	96	146	205	84
IF	9	71	1212	230	268
SI1	408	1560	4282	3575	3240
SI2	466	1081	2598	2949	2100
VS1	170	648	3589	1989	1775
VS2	261	978	5071	3357	2591
VVS1	17	186	2047	616	789
VVS2	69	286	2606	870	1235

Observations:

- The majority of gems in the database are priced below $2500.
- The color J is the least represented in the database, while the colors E, F, and G are the most prevalent, with notable relationships to clarity and cut.

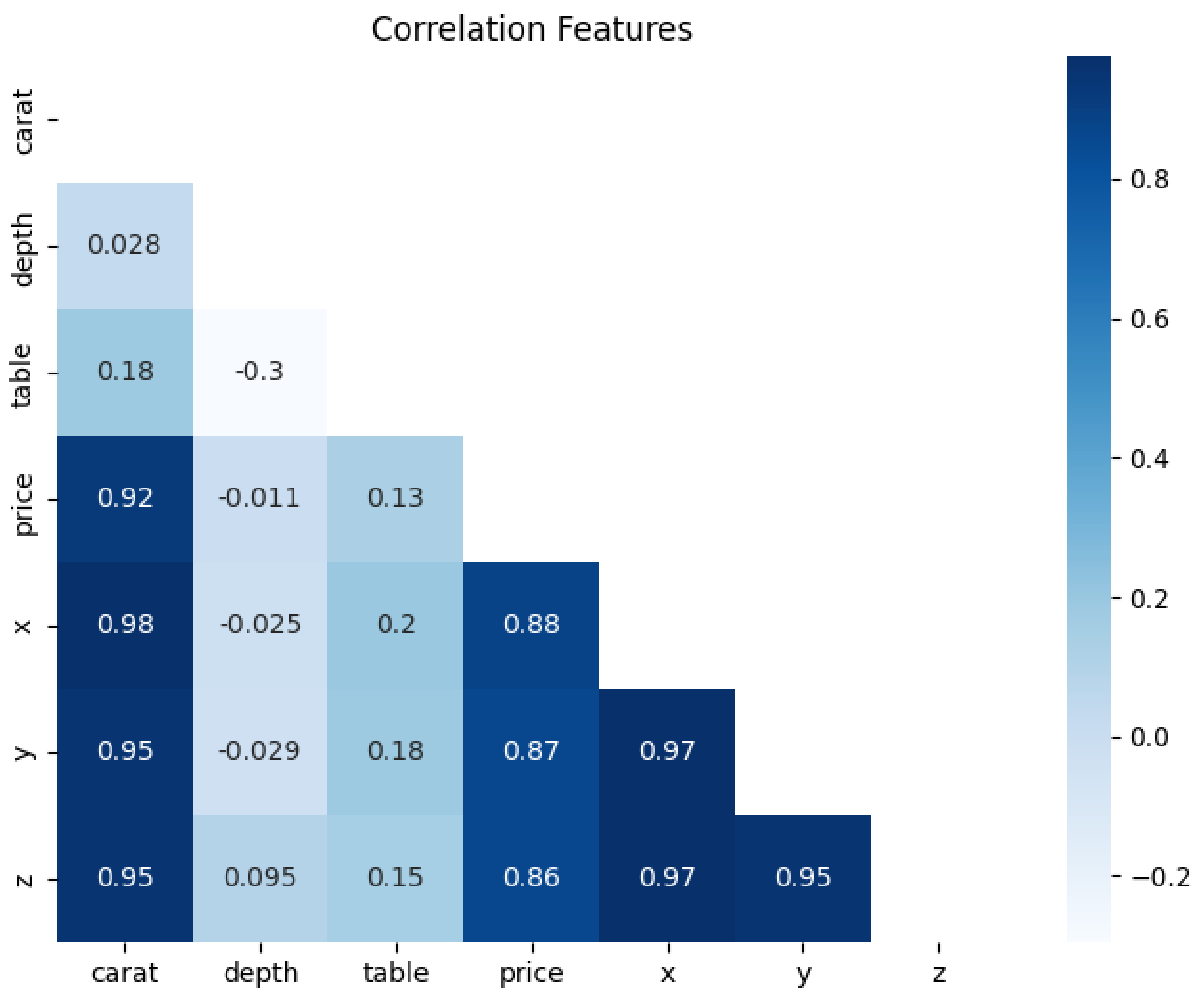
- There is no discernible pattern between clarity and cut.

Correlations

Calculate the correlation matrix and visualize a heatmap.

In […

```
# Select numeric columns from the dataframe
numeric_columns = df.select_dtypes(include=[np.number])

# Calculate the correlation matrix for numeric columns
correlation_matrix = numeric_columns.corr()

# Create a mask to hide the upper triangle of the heatm
mask = np.triu(np.ones_like(correlation_matrix, dtype=b

# Create a figure and plot the heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues
plt.title('Correlation Features')
plt.show()
```



Observations:

- Size-related variables exhibit a strong positive correlation with the gem's price.
- The variables "table" and "depth" show weak correlations with both the price and other variables.
- High correlation among different size measurements indicates the shape of the gem.

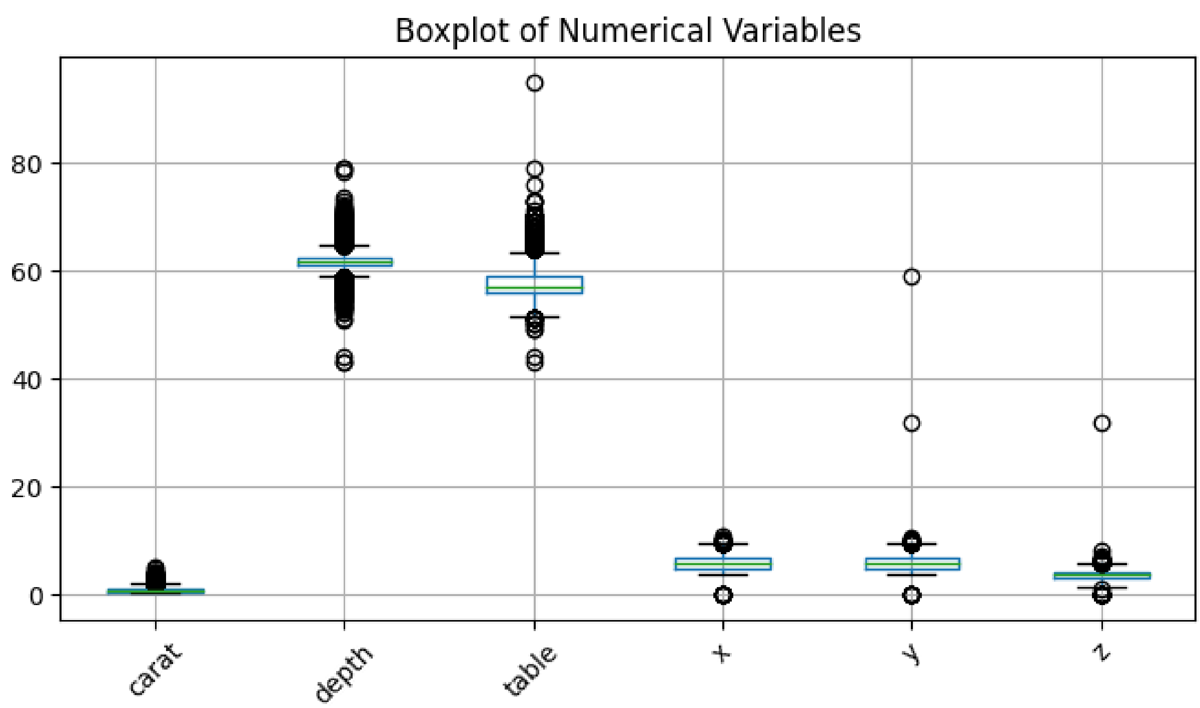
Outlier and Anomaly Analysis

Generate box plots for the numeric variables and identify potential outlier values.

In [...

```
# Create a figure with a specified size
plt.figure(figsize=(8, 4))
```

```
# Create a boxplot for selected numerical variables
df[['carat', 'depth', 'table', 'x', 'y', 'z']].boxplot(

# Set title and rotate x-axis labels for better visibil
plt.title('Boxplot of Numerical Variables')
plt.xticks(rotation=45)

# Display the plot
plt.show()
```



Observations:

- All features exhibit low variability.
- The "depth" and "table" variables have a high number of outliers; however, it is not advisable to correct them as these could correspond to rare gems that might exist.
- The outliers of the "carat," "x," "y," and "z" variables are very close to the interquartile range.

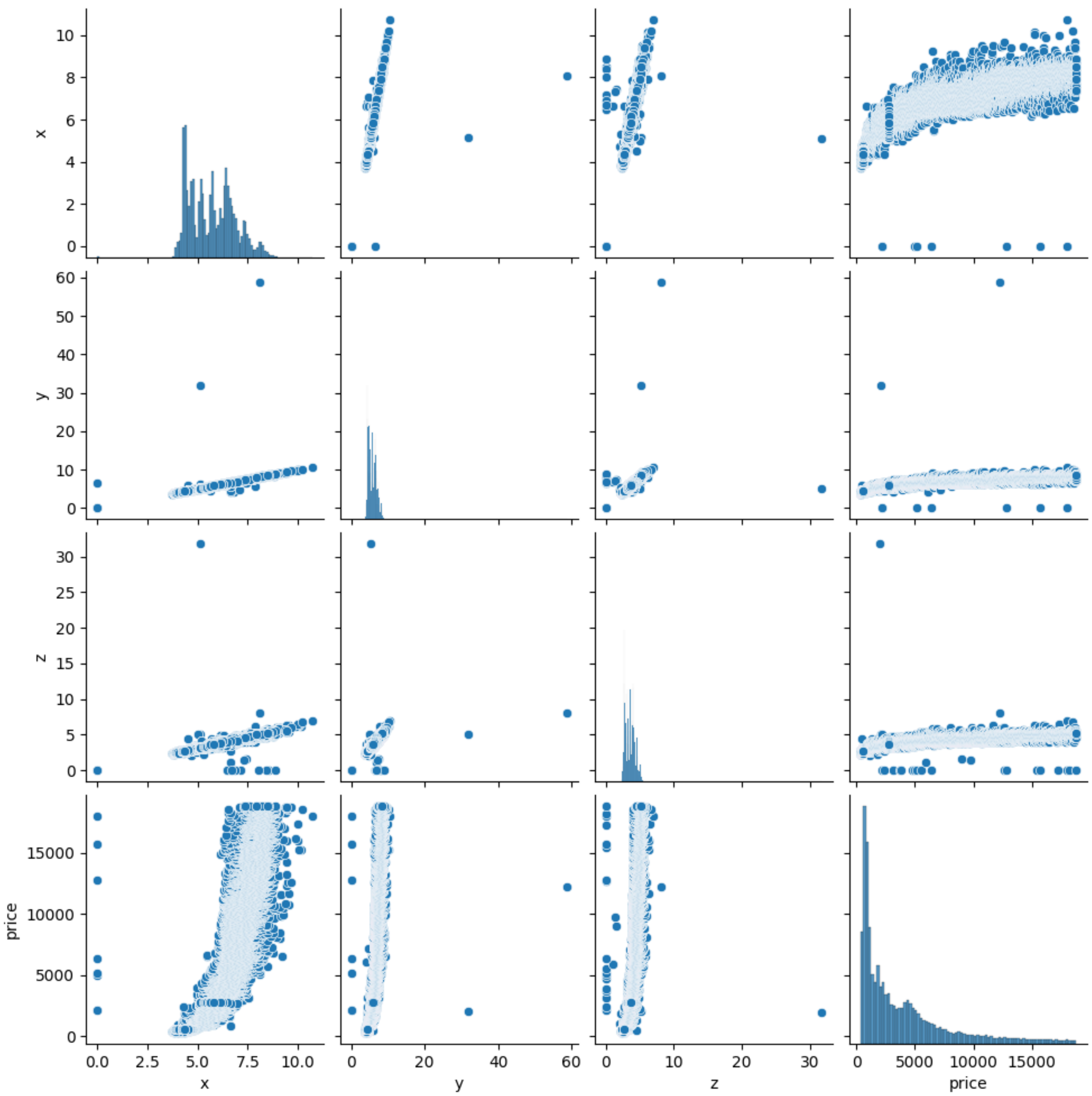
Relationships Between Variables

Create scatter plots to examine relationships between numeric variables.

Size Features

In [...

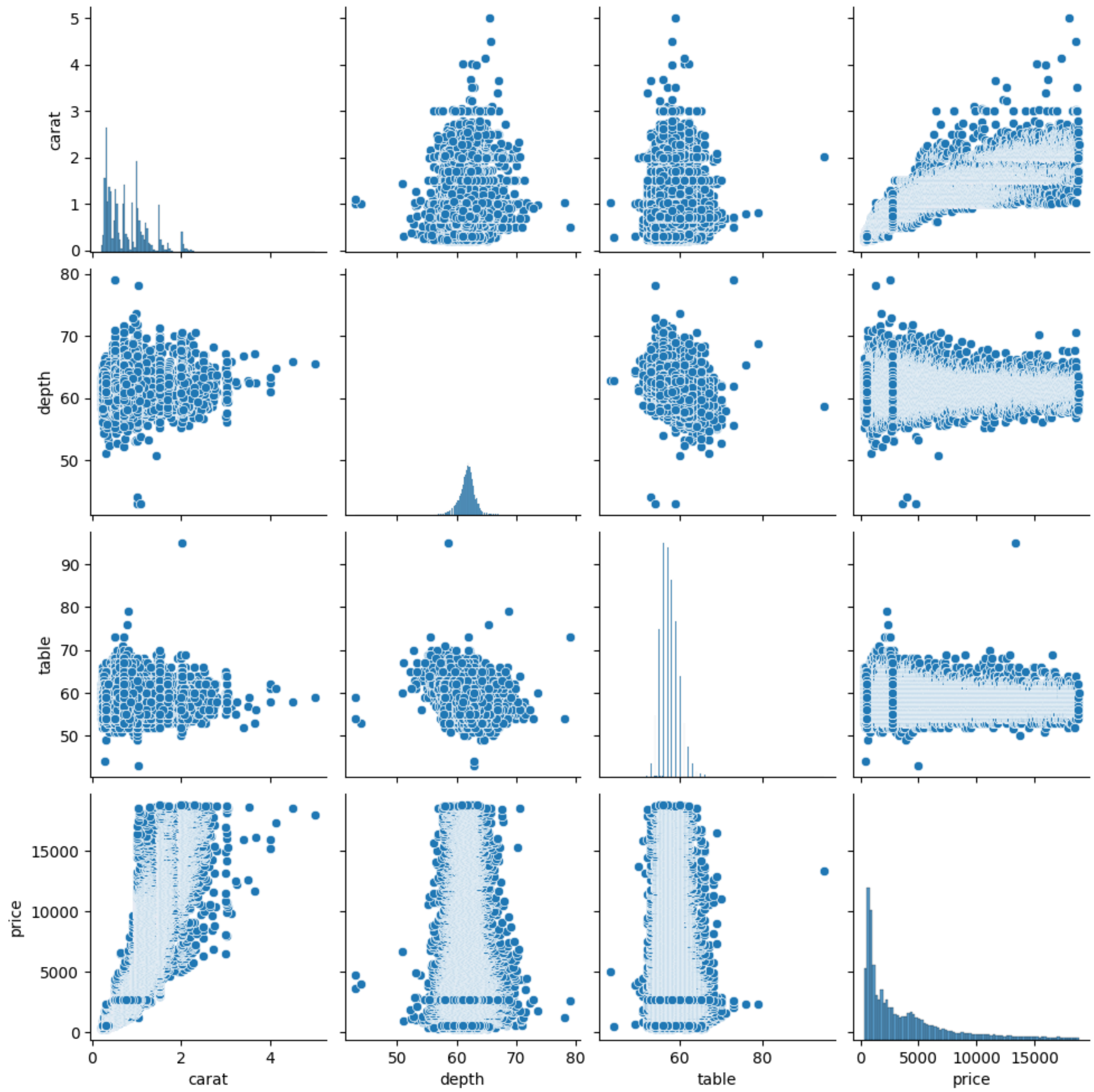
```
# Create a pair plot for size variables
s = sns.pairplot(df[['x', 'y', 'z', 'price']])
plt.show()
```



Propierties

In [...

```
# Create a pair plot for selected variables
s = sns.pairplot(df[['carat', 'depth', 'table', 'price'
plt.show()
```



Observations:

- The relationships between the size variables x, y, and z are pseudo-linear among themselves.
- The size variables x, y, and z show low variability with respect to the price.
- The depth and table variables exhibit high variability with respect to the price and are unrelated to each other.
- The carat variable displays high variability with respect to the price; however, it also shows some linear trend with the price for certain points.

Key Insights from Exploratory Data Analysis

The exploratory analysis of the dataset yielded the following key insights:

1. **Size and Characteristics:**
    - Gemstones with weights under 1 carat constitute the majority.
    - Gemstones of color J are underrepresented, while E, F, and G colors dominate.
    - Pseudo-linear correlations exist among size variables (x, y, z).
2. **Price and Features:**
    - Size-related attributes are positively correlated with gemstone prices.
    - 'Depth' and 'table' features show weak correlations with price.
    - 'Carat' demonstrates significant variability in relation to price.
3. **Outliers and Variability:**
    - Numeric attributes generally display low variability.
    - 'Depth' and 'table' exhibit pronounced outliers, likely indicating unique gemstones.
    - 'Carat', 'x', 'y', and 'z' outliers cluster near the interquartile range.
4. **Category Imbalance:**
    - Imbalanced dataset observed for the 'cut' feature.
    - Dominance of the 'Ideal' label, with limited representation for 'Good' and 'Fair'.
5. **Data Visualization:**

- Utilized various visualizations, including bar plots and box plots, to explore data relationships and distributions.

These insights collectively provide a deeper understanding of the dataset's composition and the unique attributes of gemstones. These findings underscore the need for further analysis and modeling considerations.

Libraries

In [...

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

Load Data

In [...

```
df = pd.read_excel('../data/raw/diamonds.xlsx',sheet_na
df.head(5)
```

Out[...

	carat	cut	color	clarity	depth	table	price	x	y	
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2

Analyze dataset balance for the 'cut' feature

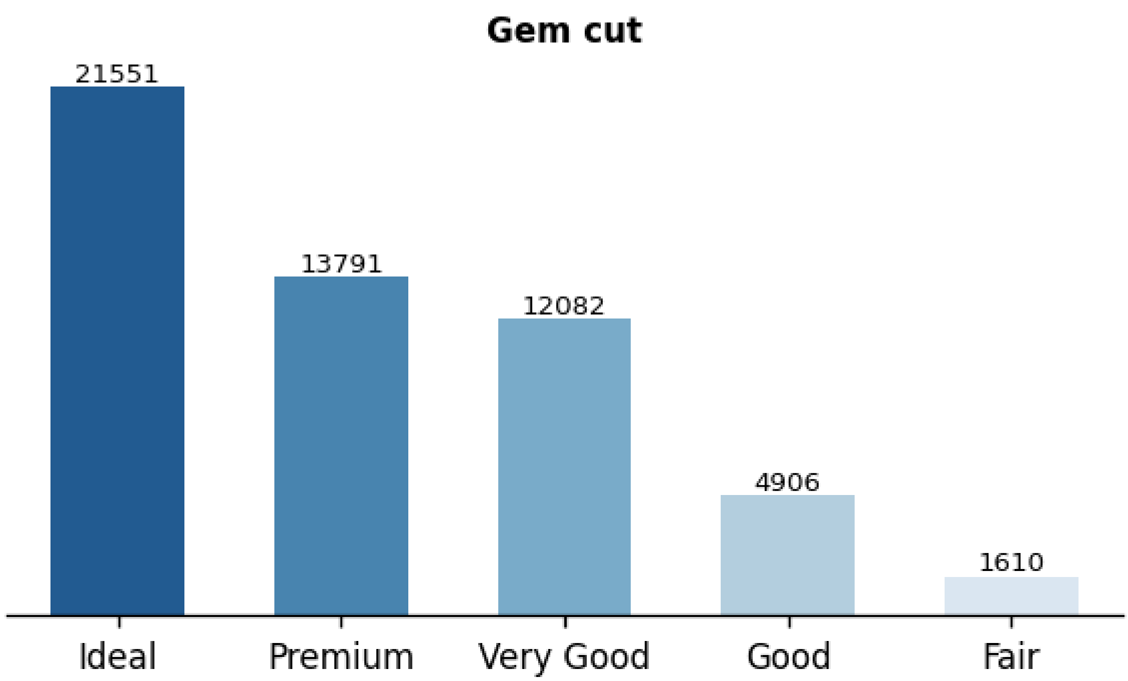
In [...

```
cut_counts = df['cut'].value_counts()
print(cut_counts)
```

```
cut
Ideal         21551
Premium       13791
Very Good     12082
Good           4906
Fair           1610
Name: count, dtype: int64
```

In […

```python
# List of categorical variables
category = 'cut'

# Create subplots with 2 rows and 2 columns
fig, ax = plt.subplots(1, 1, figsize=(6, 3), dpi=120)

# Create a count plot for the current category
s = sns.countplot(data=df, x=category, order=df[categor

# Set title, labels, and styling for the count plot
ax.set_title(f'Gem {category}', ha='center', fontweight
ax.set_yticks([])
for container in s.containers:
    s.bar_label(container, c='black', size=8)
    s.set_ylabel('')
    s.spines['top'].set_visible(False)
    s.set_xlabel('')
    s.spines['right'].set_visible(False)
    s.spines['left'].set_visible(False)
    plt.tick_params(labelleft=False)
```



Comments:

It can be observed that the database is unbalanced for the 'cut' feature, as there is a significantly reduced number of samples for the labels 'Good' and 'Fair'. Furthermore, the 'Ideal' label holds the majority of the samples, aligning with the preliminary analysis conducted during the exploratory data analysis.

As requested in the technical test, we will now propose a systematic method to balance the dataset in terms of this feature.

Synthetic data balancing

We will use the SMOTE (Synthetic Minority Over-sampling Technique) technique to address the imbalance in the dataset.

> **Justification for Choosing SMOTE:**
>
> 1. In the context of imbalanced datasets, where certain classes are underrepresented compared to others, SMOTE (Synthetic Minority Over-sampling Technique) stands out as an effective technique for generating

synthetic samples. This technique addresses the imbalance by creating synthetic examples for the minority class, thereby mitigating the class imbalance problem.

2. The primary reasons for choosing SMOTE are as follows:

3. Maintaining Diversity: SMOTE not only oversamples the minority class but also generates samples along the decision boundary, preserving the diversity within the class. This helps prevent overfitting and maintains the representativeness of the minority class.

4. Addressing Overfitting: By creating synthetic samples based on the feature space distribution, SMOTE avoids direct replication of existing examples. This reduces the risk of overfitting that might occur with simpler oversampling techniques.

5. Algorithmic Simplicity: SMOTE is a relatively simple and intuitive technique to implement. It involves identifying nearest neighbors and creating synthetic examples based on the interpolation of feature vectors. This simplicity makes it accessible and adaptable to various scenarios.

6. Compatibility with Algorithms: The generated synthetic samples from SMOTE can be seamlessly integrated into machine learning algorithms without any modification. This makes it convenient for downstream modeling tasks.

7. Effective Handling of Minority Class: SMOTE has shown promising results in addressing the class imbalance issue, leading to improved classification

performance and better generalization on imbalanced datasets.

8. Given the unbalanced distribution of the 'cut' feature in the dataset, using SMOTE to generate synthetic samples for the minority classes ('Fair' and 'Good') can help improve the balance of the dataset and potentially enhance the overall performance of predictive models that might be trained on it.

In [...

```
# Copy the original DataFrame
df_encoded = df.copy()

# Encode categorical variables using LabelEncoder
label_encoders = {}
for col in ['color', 'clarity']:
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])
    label_encoders[col] = le

# Separate features and target
X = df_encoded.drop('cut', axis=1)
y = df_encoded['cut']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,

# Apply SMOTE to balance classes
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resamp

# Verify the new class distribution
print(y_train_resampled.value_counts())
```

```
cut
Good         17259
Very Good    17259
Premium      17259
Ideal        17259
Fair         17259
Name: count, dtype: int64
```

In [...

```
X_train_resampled['cut'] = y_train_resampled
X_train_resampled.head(5)
```

Out[...

	carat	color	clarity	depth	table	price	x	y	z	
0	2.01	2	3	58.1	64.0	16231	8.23	8.19	4.77	G
1	1.01	1	3	60.0	60.0	4540	6.57	6.49	3.92	\ G
2	1.10	4	5	62.5	58.0	5729	6.59	6.54	4.10	Prem
3	1.50	1	3	61.5	65.0	6300	7.21	7.17	4.42	G
4	1.52	3	4	62.1	57.0	12968	7.27	7.32	4.53	\ G

Save balanced data

In [...

```
X_train_resampled.to_excel('../data/processed/diamonds_
```

Comments:

Despite having performed class balancing using the SMOTE technique, it is crucial to validate the impact of the class imbalance in the 'cut' feature on algorithmic predictions during model training. If necessary, further balancing procedures should be considered. Additionally, exploring alternative techniques, such as undersampling or hybrid methods, might be beneficial to comprehensively address the imbalance challenge. Careful monitoring and adjustment of class balance are imperative for ensuring robust and accurate predictive modeling. This becomes especially pertinent when the balanced feature is relevant for regression tasks or represents the target label to be predicted.

Libraries

In [...

```
import sys

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder

sys.path.append('../src/features')
from build_features import ZirconsDataProcessor
```

Load Data

In [...

```
df = pd.read_excel('../data/raw/diamonds.xlsx')
df.head(5)
```

Out[...

	carat	cut	color	clarity	depth	table	price	x	y
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2

Feature Engineering

> New Features Explanation
>
> 1. Volume (`volume`): Represents the calculated volume of the cubic zirconia stone based on its dimensions

(`x` , `y` , and `z`). This volume metric provides insight into the spatial extent of the stone, which could be indicative of its physical size and overall presence.

2. Density (`density`): Feature is derived from the carat weight of the stone (`carat`) and its calculated volume. This metric helps to quantify the mass-to-volume ratio of the stone. Density might be relevant in identifying denser or less dense stones, which could potentially relate to their material composition.

3. Depth per Volume (`depth_per_volume`): Expresses the depth (`depth`) of the stone in relation to its calculated volume. This ratio offers a measure of how deep the stone is relative to its overall size. It could provide insights into the stone's proportions and whether its depth is proportional to its volume.

4. Depth per Density (`depth_per_density`): Signifies the depth (`depth`) of the stone relative to its calculated density. This ratio gives an indication of how the stone's depth compares to its density, which might help identify cases where the depth is not aligned with the expected density.

5. Depth per Table (`depth_per_table`): Reflects the depth (`depth`) of the stone in relation to its table width (`table`). This ratio aids in understanding whether the stone's depth is balanced with its table width, which could influence its visual appearance and proportions.

6. Ratio of Length to Width (`ratio_xy`), Ratio of Length to Height (`ratio_xz`), Ratio of Width to Height (`ratio_yz`): These three features represent different ratios between the dimensions of the stone

> (`x` , `y` , and `z`). The `ratio_xy` gives insight into the stone's overall shape, while the `ratio_xz` and `ratio_yz` ratios provide information about how the dimensions are distributed along the height axis. These ratios could be useful for understanding the stone's geometric attributes and potential aesthetic aspects.

Add new features

In [...
```
processor = ZirconsDataProcessor(df)
data, price = processor.data_processor()
```

In [...
```
data.head()
```

Out[...

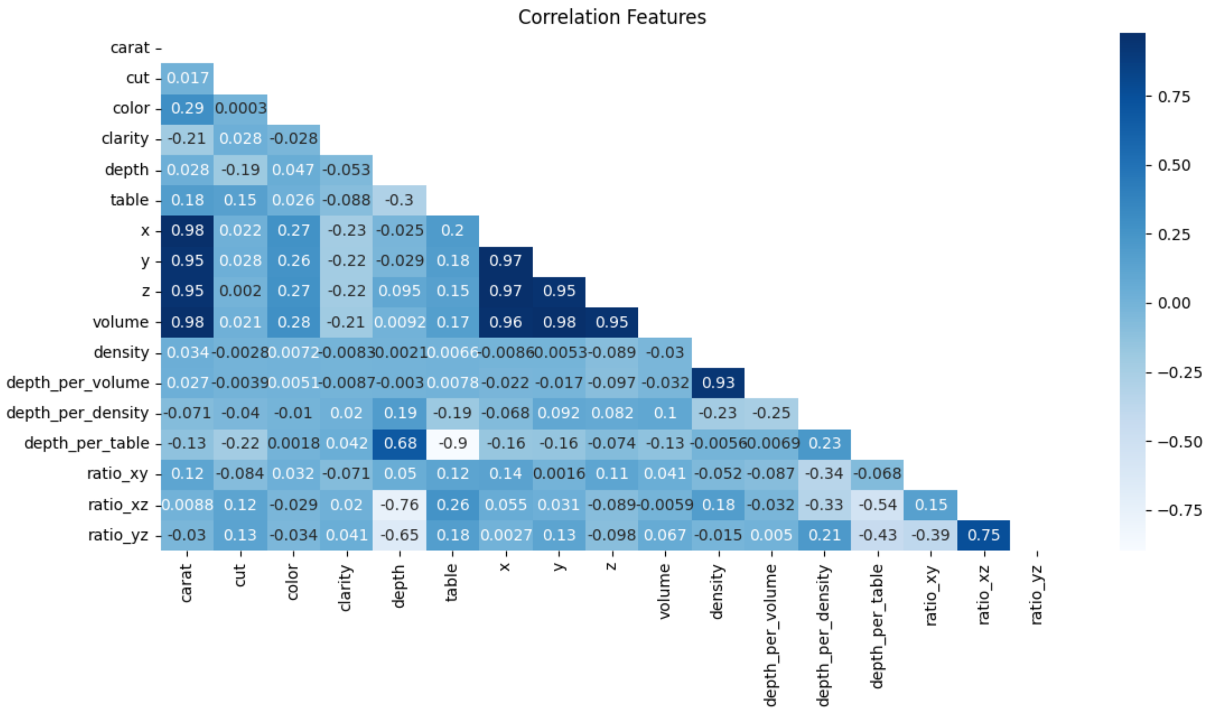
	carat	cut	color	clarity	depth	table	x	
0	0.23	2.0	1.0	3.0	-0.174092	-1.099672	-1.587837	-1.536
1	0.21	3.0	1.0	2.0	-1.360738	1.585529	-1.641325	-1.658
2	0.23	1.0	1.0	4.0	-3.385019	3.375663	-1.498691	-1.457
3	0.29	3.0	5.0	5.0	0.454133	0.242928	-1.364971	-1.317
4	0.31	1.0	6.0	3.0	1.082358	0.242928	-1.240167	-1.212

Correlations

Calculate the correlation matrix and visualize a heatmap.

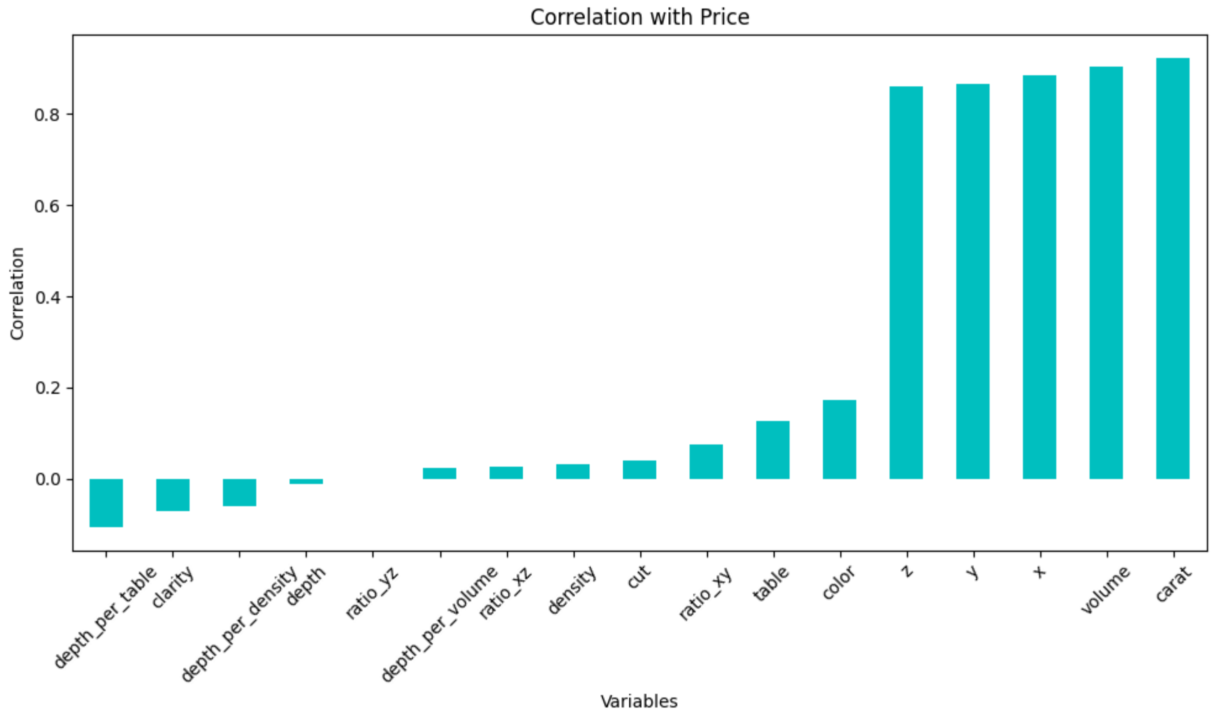
In [...
```
# Select numeric columns from the dataframe
numeric_columns = data.select_dtypes(include=[np.number

# Calculate the correlation matrix for numeric columns
correlation_matrix = numeric_columns.corr()

# Create a mask to hide the upper triangle of the heatm
mask = np.triu(np.ones_like(correlation_matrix, dtype=b
```

```python
# Create a figure and plot the heatmap
plt.figure(figsize=(13, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='Blues
plt.title('Correlation Features')
plt.show()
```



In [...

```python
data['price'] = price
correlations = data.corr()['price']

plt.figure(figsize=(10, 6))
correlations.drop('price').sort_values().plot(kind='bar
plt.title('Correlation with Price')
plt.ylabel('Correlation')
plt.xlabel('Variables')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



These newly created features introduce additional dimensions of information to the dataset, potentially enriching the insights that can be derived from the data. By incorporating these metrics, you can explore different aspects of the cubic zirconia stones beyond their basic characteristics.

Libraries

In [...

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.metrics import (
    mean_squared_error,
    mean_absolute_error
)
import matplotlib.pyplot as plt

import sys
sys.path.append('../src/')

from models.train_model import Train
from data.partitions import split_data
from models.predict_model import Predict
from features.build_features import ZirconsDataProcesso
```

Load Data

In [...

```
df_raw = pd.read_excel('../data/raw/diamonds.xlsx')
df_bal = pd.read_excel('../data/processed/diamonds_bala
df_raw.head(5)
```

Out[...

	carat	cut	color	clarity	depth	table	price	x	y	
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2

Unbalanced Data

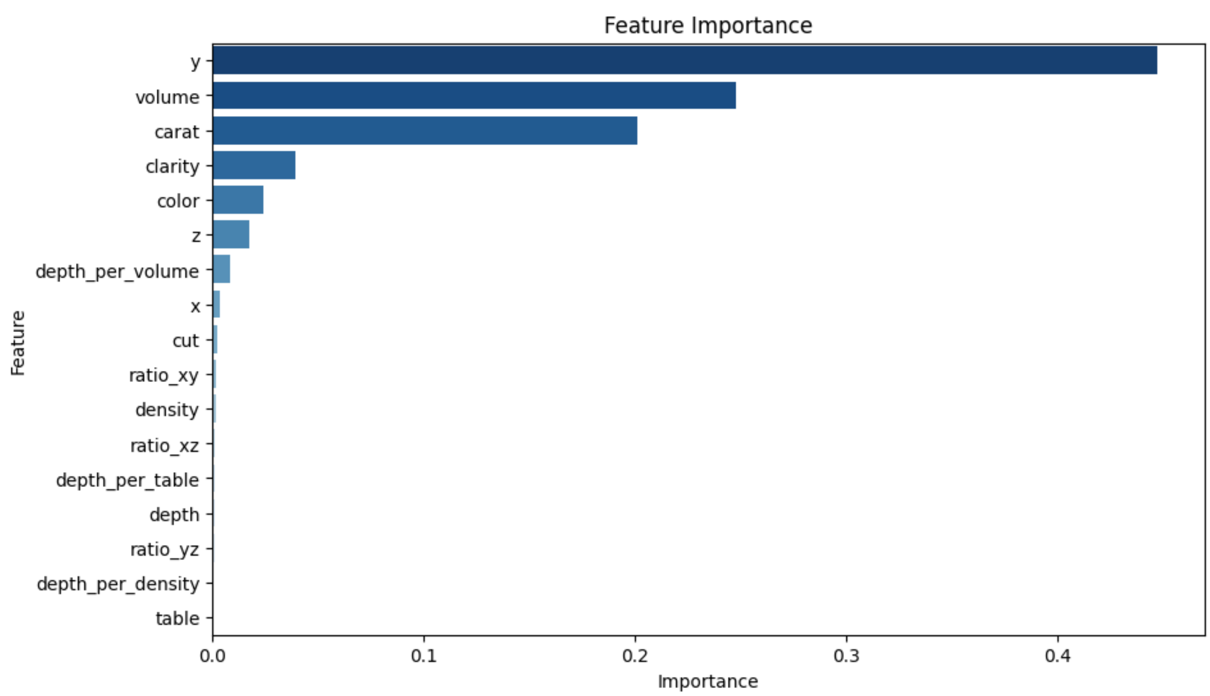
Train model

```
In [...
processor = ZirconsDataProcessor(df_raw, test=False)
data, price = processor.data_processor()

X_train, X_test, y_train, y_test = split_data(data, pri

trainer = Train(X_train, y_train, 'best_unbalanced_mode
best_model, feature_relevance, top_score = trainer.trai
```

```
Fitting 3 folds for each of 27 candidates, totalling 81
fits
```

Feature relevance

```
In [...
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_r
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')
plt.show()
```



Best train scores

In [...
```
top_score.head(5)
```
Out[...

	params	mean_test_score
8	{'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	-547.787477
7	{'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	-548.616606
6	{'learning_rate': 0.1, 'max_depth': 5, 'n_esti...	-558.259833
5	{'learning_rate': 0.1, 'max_depth': 4, 'n_esti...	-561.492331
4	{'learning_rate': 0.1, 'max_depth': 4, 'n_esti...	-567.850369

Test model

In [...
```
predictor = Predict('../models/best_unbalanced_model.pk
y_pred = predictor.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  =  mean_absolute_error(y_test, y_pred)

print(f'rmse test: {rmse}')
print(f'mae test: {mae}')
```
```
rmse test: 521.7446558565866
mae test: 268.2372914730228
```

Balanced Data

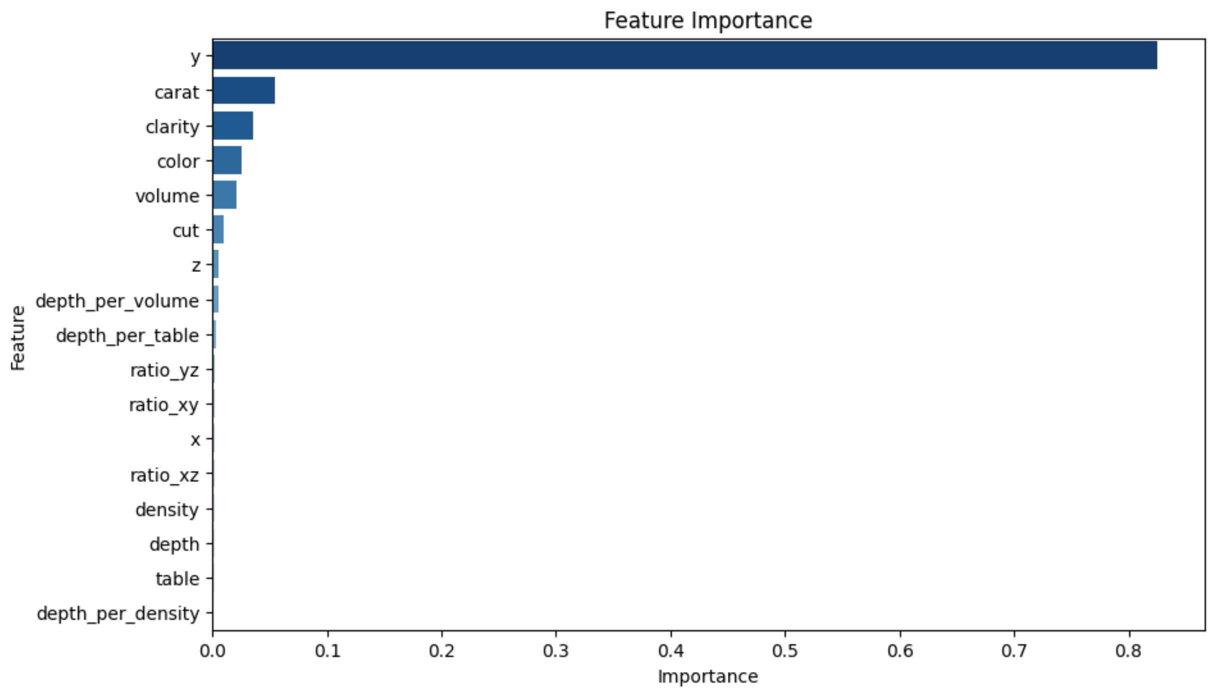
Train model

In [...
```
processor = ZirconsDataProcessor(df_bal, test=False)
data, price = processor.data_processor()

X_train, X_test, y_train, y_test = split_data(data, pri

trainer = Train(X_train, y_train,'best_balanced_model')
best_model, feature_relevance, top_score = trainer.trai
```

```
Fitting 3 folds for each of 27 candidates, totalling 81
fits
```

Feature relevance

```
In [...  plt.figure(figsize=(10, 6))
         sns.barplot(x='Importance', y='Feature', data=feature_r
         plt.xlabel('Importance')
         plt.ylabel('Feature')
         plt.title('Feature Importance')
         plt.show()
```



Test model

```
In [...  predictor = Predict('../models/best_balanced_model.pkl'
         y_pred = predictor.predict(X_test)

         rmse = np.sqrt(mean_squared_error(y_test, y_pred))
         mae  =  mean_absolute_error(y_test, y_pred)

         print(f'rmse test: {rmse}')
         print(f'mae test: {mae}')
```

```
rmse test: 685.3129627895665
mae test: 369.48215099450977
```

Conclusions:

- The model was trained using 70% of the data and tested on the remaining 30%.
- The most relevant features were determined, with the 'y' size feature being the most influential, followed by volume, carat, and clarity.
- Despite showing significant variability in the initial exploration, color proved to have relevance for predicting the model.
- The model was trained both with balanced and unbalanced data for the 'cut' feature.
- However, no substantial improvement was observed, as 'cut' is a less relevant feature for prediction. Therefore, the unbalanced version of the model will be used for exposition, eliminating the need for additional balancing steps.