Mohs Hardness Regression Project Results

Phase	Description	Outcome
Data Exploration	Exploring and analyzing the dataset	Insights into key features and patterns
□ Model Development	Building and training regression models	Predictive models for Mohs hardness
Performance Evaluation	Assessing model accuracy and reliability	Quantitative evaluation metrics
Model Optimization	Fine-tuning and improving model performance	Enhanced prediction accuracy

Project Overview

The Mohs Hardness Regression project aimed to predict the Mohs hardness of minerals based on various features. The project involved data exploration, model development, performance evaluation, and model optimization to achieve accurate predictions.

Project Objectives:

1. **Data Exploration:**
   - Conducted in-depth exploration and analysis of the dataset to understand feature distributions and relationships.
   - Identified key factors influencing Mohs hardness through visualizations and statistical analysis.
2. **Model Development:**
   - Implemented regression models to predict Mohs hardness using relevant features.
   - Utilized machine learning techniques for feature selection and model training.
3. **Performance Evaluation:**
   - Evaluated model performance using metrics such as Mean Absolute Error (MAE) and R-squared to gauge prediction accuracy.
   - Analyzed model strengths and limitations through visualizations and statistical assessments.
4. **Model Optimization:**
   - Fine-tuned hyperparameters and conducted feature engineering to optimize model performance.
   - Iteratively improved models to achieve higher accuracy in Mohs hardness prediction.

Data Set Description:

The dataset consisted of mineral samples with various features such as chemical composition, crystalline structure, and density. The target variable was the Mohs hardness rating.

Information	Details
Number of Samples	[Number]
Features	[List of Features]
Target Variable	Mohs Hardness

Project Steps:

Data Exploration:

- Explored and visualized feature distributions, correlations, and potential outliers.
- Identified significant features affecting Mohs hardness.

Model Development:

- Implemented regression models such as Linear Regression, Decision Trees, and Random Forest for prediction.
- Split the dataset into training and testing sets for model training and evaluation.

Performance Evaluation:

- Utilized metrics like MAE and R-squared to evaluate model accuracy.
- Visualized predicted vs. actual Mohs hardness values for qualitative assessment.

Model Optimization:

- Fine-tuned hyperparameters and conducted cross-validation for optimal model configuration.
- Addressed overfitting or underfitting issues through regularization techniques.

Technologies Used:

- Python (Pandas, NumPy, Matplotlib, Seaborn)
- Scikit-Learn for Machine Learning
- Jupyter Notebooks for Analysis and Visualization

The successful implementation of the Mohs Hardness Regression project provides valuable insights into predicting mineral hardness, with applications in geology, material science, and industry.

1. Data Loading and Exploration

In [ ]:
```
# Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

import time

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.discriminant_analysis import StandardScaler

from scipy.stats import zscore

# Models
from catboost import CatBoostRegressor
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.neural_network import MLPRegressor

from keras.models import Sequential
from keras.layers import Dense
from keras.callbacks import EarlyStopping

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# Load Data
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
sample_submission = pd.read_csv('data/sample_submission.csv')
```

In [ ]:
```
start_time = time.time()
```

In [ ]:
```
# Pandas setting to display more dataset rows and columns
pd.set_option('display.max_rows', 150)
pd.set_option('display.max_columns', 600)
pd.set_option('display.max_colwidth', None)
pd.set_option('display.float_format', lambda x: '%.5f' % x)
```

In [ ]:
```
print('\nTrain Dataset:')
train.head()
```

```
Train Dataset:
```

Out[ ]:

	id	allelectrons_Total	density_Total	allelectrons_Average	val_e_Average	atomicweight_Average	ionenergy_Average	el_neg_chi
0	0	100.00000	0.84161	10.00000	4.80000	20.61253	11.08810	
1	1	100.00000	7.55849	10.00000	4.80000	20.29889	12.04083	
2	2	76.00000	8.88599	15.60000	5.60000	33.73926	12.08630	
3	3	100.00000	8.79530	10.00000	4.80000	20.21335	10.94850	
4	4	116.00000	9.57800	11.60000	4.80000	24.98813	11.82448	

In [ ]:
```
print('\nTest Dataset:')
test.head()
```

```
Test Dataset:
```

Out[ ]:

	id	allelectrons_Total	density_Total	allelectrons_Average	val_e_Average	atomicweight_Average	ionenergy_Average	el_neg_
0	10407	884.00000	121.42000	35.36000	5.28000	82.56124	9.37038	
1	10408	90.00000	9.93196	18.00000	5.60000	39.56806	12.08630	
2	10409	116.00000	7.76799	11.60000	4.80000	23.23182	11.02384	
3	10410	100.00000	9.10800	10.00000	4.80000	20.29889	12.08630	
4	10411	55.00000	4.03000	11.00000	4.00000	22.97767	11.28095	

In [ ]:
```
print('\nSubmission Dataset:')
sample_submission.head()
```

```
Submission Dataset:
```

Out[ ]:

	id	Hardness
0	10407	4.64700
1	10408	4.64700
2	10409	4.64700
3	10410	4.64700
4	10411	4.64700

In [ ]:
```
print('\nTrain Dataset İnfo:')
train.info()
```

```
Train Dataset İnfo:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10407 entries, 0 to 10406
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     10407 non-null  int64  
 1   allelectrons_Total     10407 non-null  float64
 2   density_Total          10407 non-null  float64
 3   allelectrons_Average   10407 non-null  float64
 4   val_e_Average          10407 non-null  float64
 5   atomicweight_Average   10407 non-null  float64
 6   ionenergy_Average      10407 non-null  float64
 7   el_neg_chi_Average     10407 non-null  float64
 8   R_vdw_element_Average  10407 non-null  float64
 9   R_cov_element_Average  10407 non-null  float64
 10  zaratio_Average        10407 non-null  float64
 11  density_Average        10407 non-null  float64
 12  Hardness               10407 non-null  float64
dtypes: float64(12), int64(1)
memory usage: 1.0 MB
```

In [ ]:
```
print('\nTest Dataset İnfo:')
test.info()
```

```
Test Dataset İnfo:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6939 entries, 0 to 6938
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     6939 non-null   int64  
 1   allelectrons_Total     6939 non-null   float64
 2   density_Total          6939 non-null   float64
 3   allelectrons_Average   6939 non-null   float64
 4   val_e_Average          6939 non-null   float64
 5   atomicweight_Average   6939 non-null   float64
 6   ionenergy_Average      6939 non-null   float64
 7   el_neg_chi_Average     6939 non-null   float64
 8   R_vdw_element_Average  6939 non-null   float64
 9   R_cov_element_Average  6939 non-null   float64
 10  zaratio_Average        6939 non-null   float64
 11  density_Average        6939 non-null   float64
dtypes: float64(11), int64(1)
memory usage: 650.7 KB
```

In [ ]:
```
print('\nSubmission Dataset İnfo:')
sample_submission.info()
```

```
Submission Dataset İnfo:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6939 entries, 0 to 6938
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   id        6939 non-null   int64
 1   Hardness  6939 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 108.5 KB
```

In [ ]:
```
print('\nTrain Dataset Describe:')
train.describe()
```

Train Dataset Describe:

Out[ ]:

	id	allelectrons_Total	density_Total	allelectrons_Average	val_e_Average	atomicweight_Average	ionenergy_Averag
count	10407.00000	10407.00000	10407.00000	10407.00000	10407.00000	10407.00000	10407.0000
mean	5203.00000	128.05352	14.49134	17.03322	4.54679	37.50770	10.9383
std	3004.38646	224.12378	15.97288	10.46873	0.69086	26.01231	1.4082
min	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
25%	2601.50000	68.00000	7.55849	10.00000	4.00000	20.29889	10.5906
50%	5203.00000	100.00000	10.65000	12.60000	4.71429	26.20383	11.2027
75%	7804.50000	131.00000	16.67700	22.00000	4.80000	48.71950	11.6707
max	10406.00000	15300.00000	643.09380	67.00000	6.00000	167.40000	15.2458

In [ ]:
```
print('\nTest Dataset Describe:')
test.describe()
```

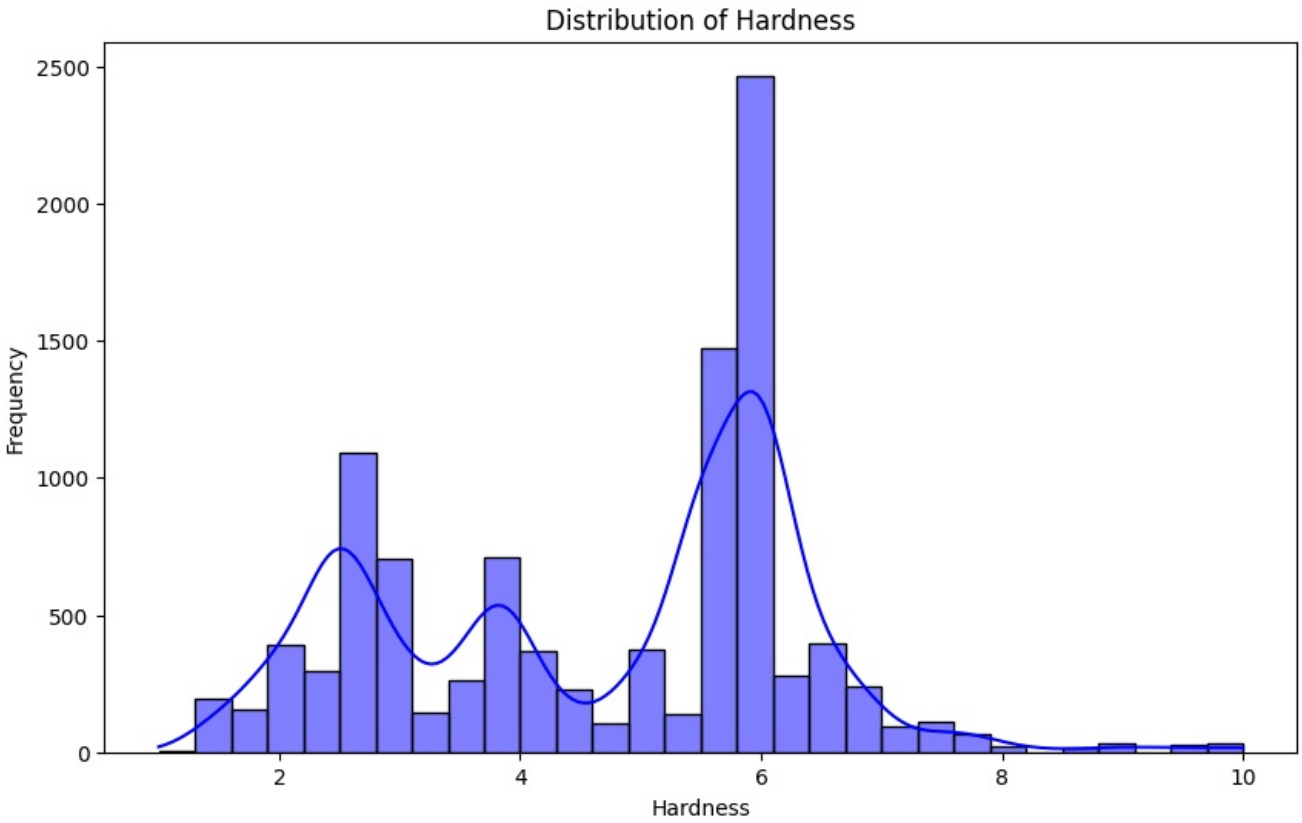
Test Dataset Describe:

Out[ ]:

	id	allelectrons_Total	density_Total	allelectrons_Average	val_e_Average	atomicweight_Average	ionenergy_Averag
count	6939.00000	6939.00000	6939.00000	6939.00000	6939.00000	6939.00000	6939.0000
mean	13876.00000	126.46013	14.79402	17.40619	4.54685	38.42279	10.9215
std	2003.26109	207.56450	18.98245	10.99609	0.68316	27.34435	1.3789
min	10407.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.0000
25%	12141.50000	68.00000	7.55849	10.00000	4.00000	20.29889	10.5843
50%	13876.00000	100.00000	10.65000	12.66667	4.75000	26.20383	11.2027
75%	15610.50000	128.00000	16.60133	22.00000	4.80000	48.71950	11.6455
max	17345.00000	10116.00000	643.09380	67.00000	6.00000	167.40000	15.2458

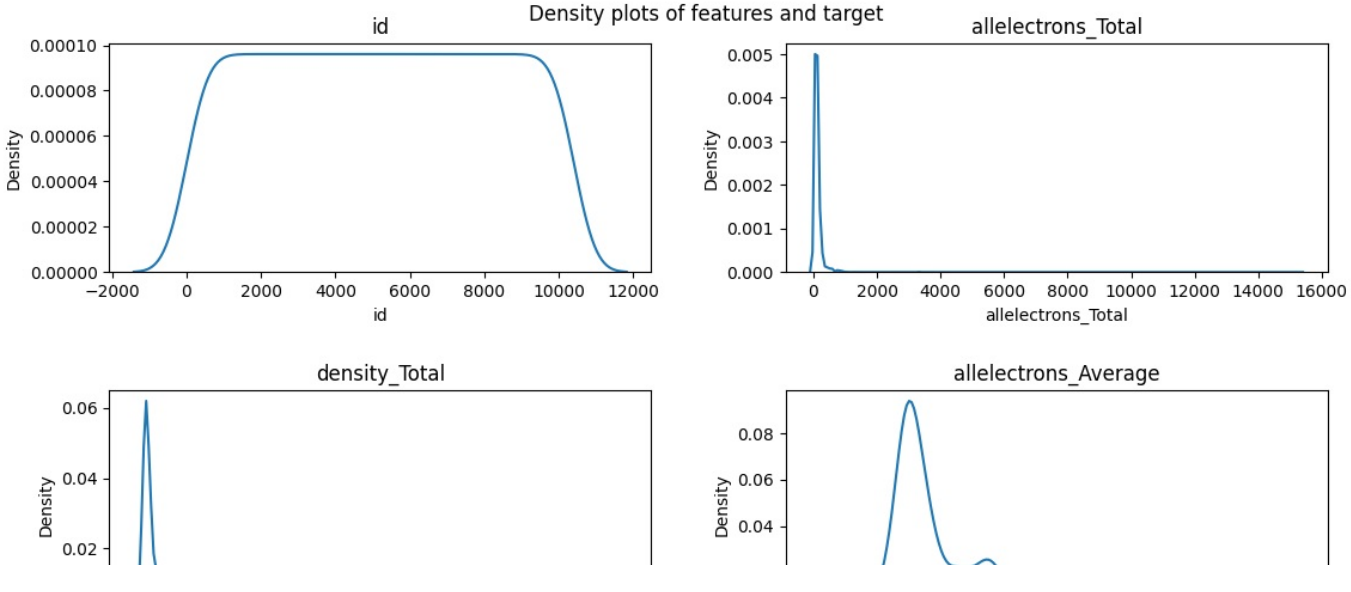
2. Exploratory Data Analysis (EDA)

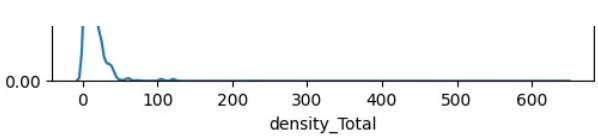
In [ ]:
```
# Visualize the distribution of the target variable
plt.figure(figsize=(10, 6))
sns.histplot(train['Hardness'], bins=30, kde=True, color='blue')
plt.title('Distribution of Hardness')
plt.xlabel('Hardness')
plt.ylabel('Frequency')
plt.show()
```

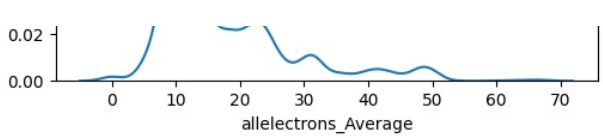


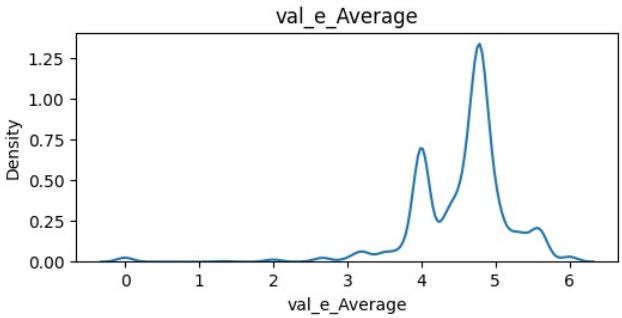
In [ ]:

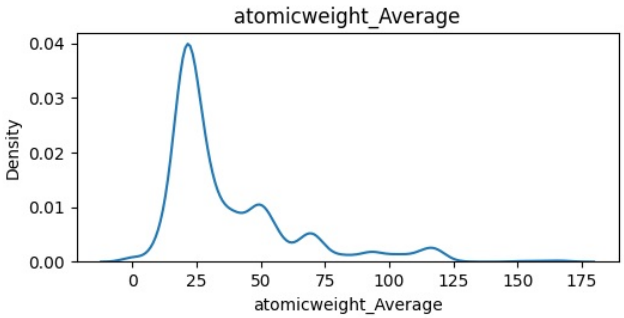
```python
def generate_density_plots(data, title):
    # Calculate the number of rows and columns based on the number of features
    num_features = data.shape[1]
    num_rows = (num_features + 1) // 2  # Add 1 to include the target variable
    num_cols = 2

    # Create subplots
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(12, 3 * num_rows))
    fig.tight_layout(pad=5.0)
    fig.suptitle(title)

    # Flatten the axes in case num_rows is greater than 1
    axes = axes.flatten()

    # Loop through each feature and the target variable
    for i, ax in enumerate(axes[:-1]):  # Exclude the last subplot for the target variable
        col = data.columns.tolist()[i]
        sns.kdeplot(data=data, ax=ax, x=col)
        ax.set_title(col)

    # Add a density plot for the target variable in the last subplot
    target_col = data.columns.tolist()[-1]
    sns.kdeplot(data=data, ax=axes[-1], x=target_col, color='red')
    axes[-1].set_title(target_col)

# Example usage for your train dataset
generate_density_plots(data=train, title='Density plots of features and target')
plt.show()
```

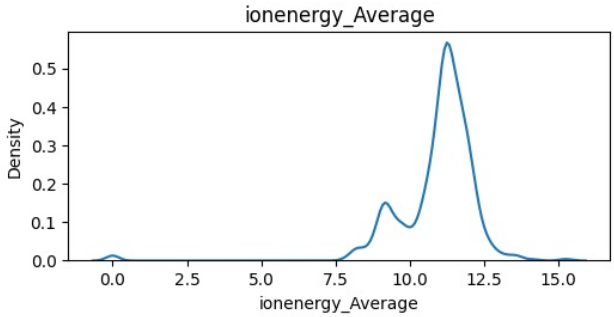


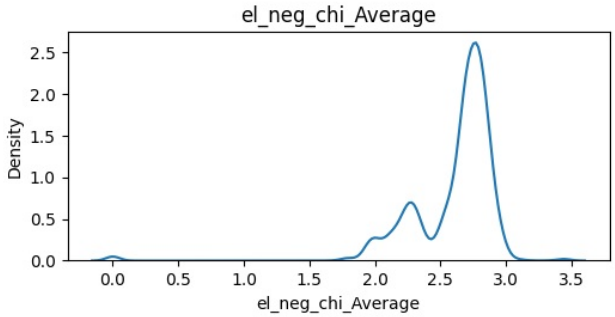


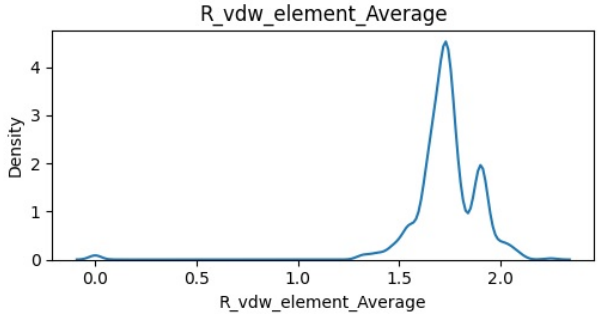


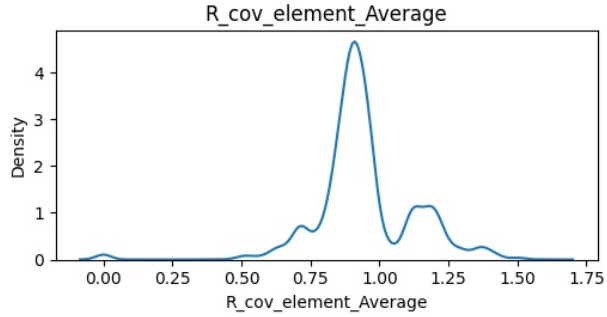


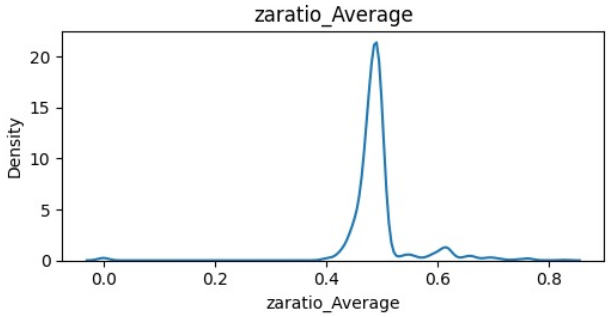


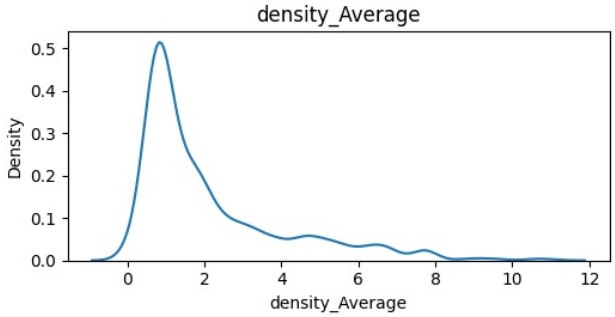


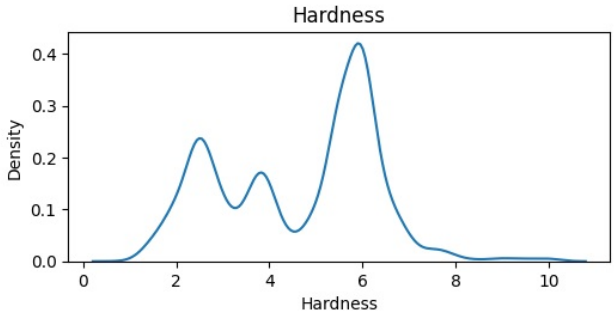


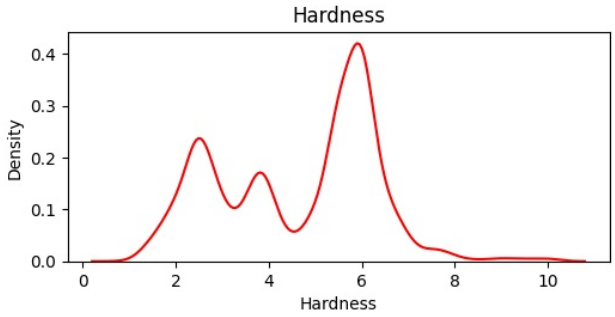




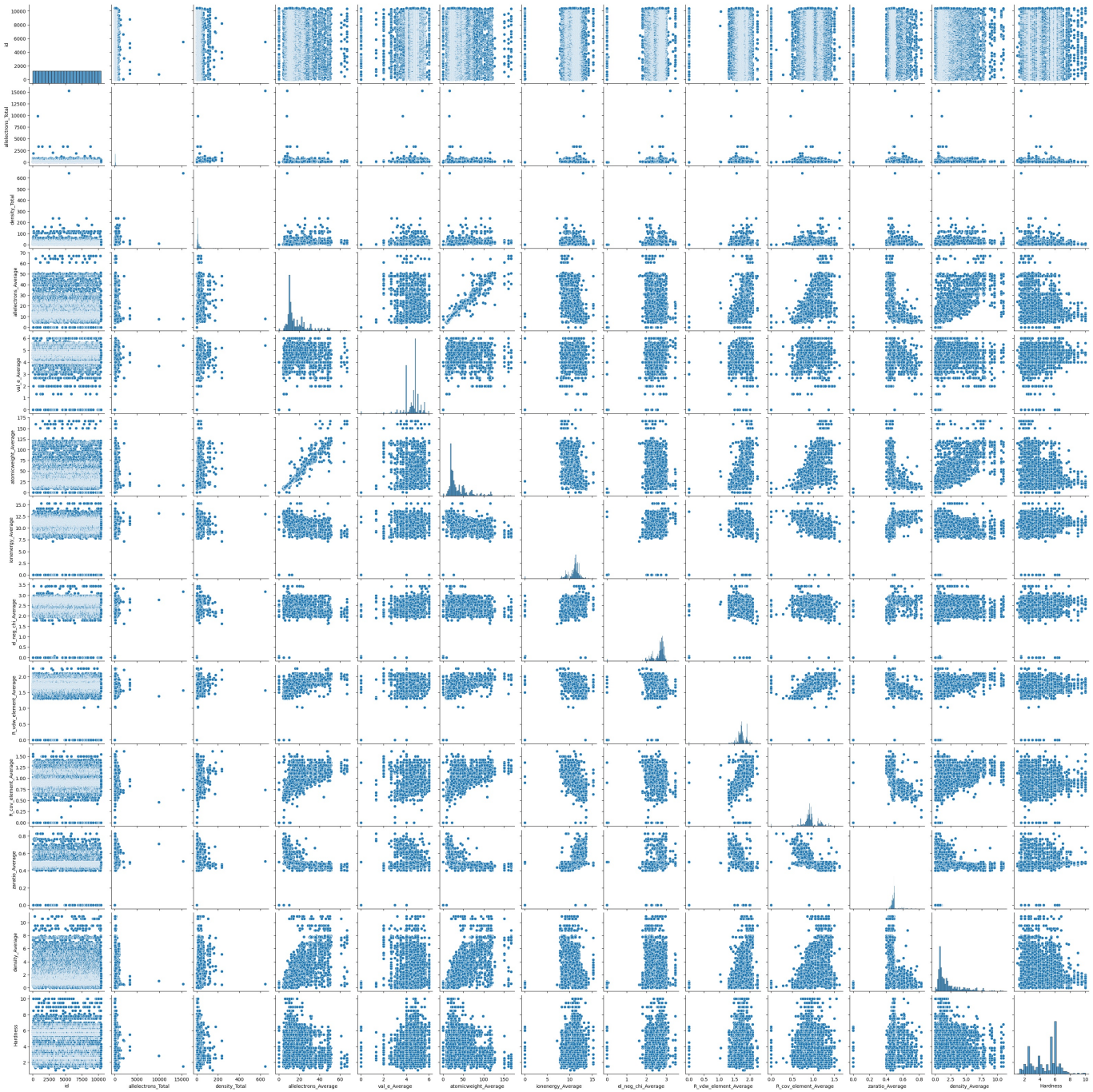




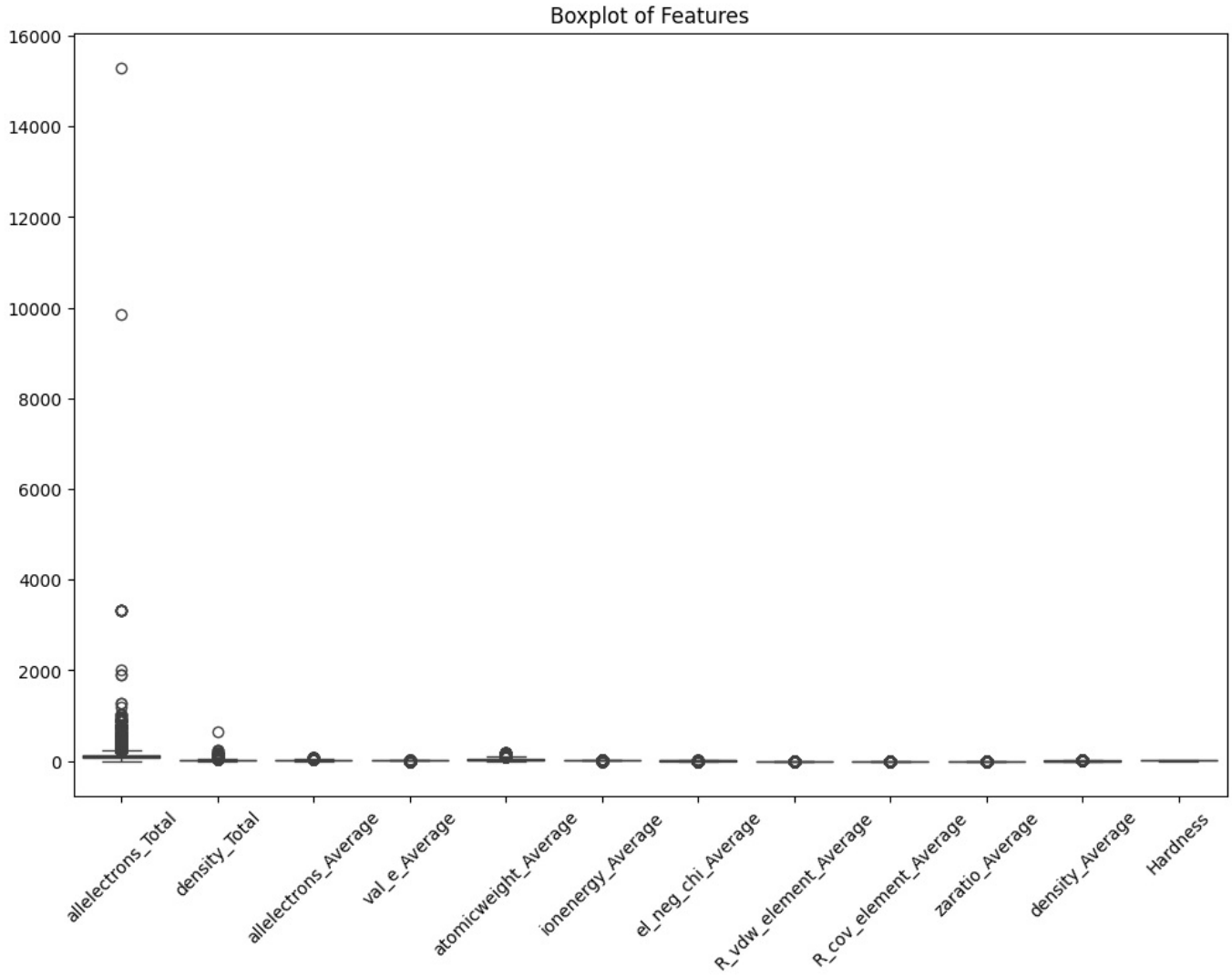




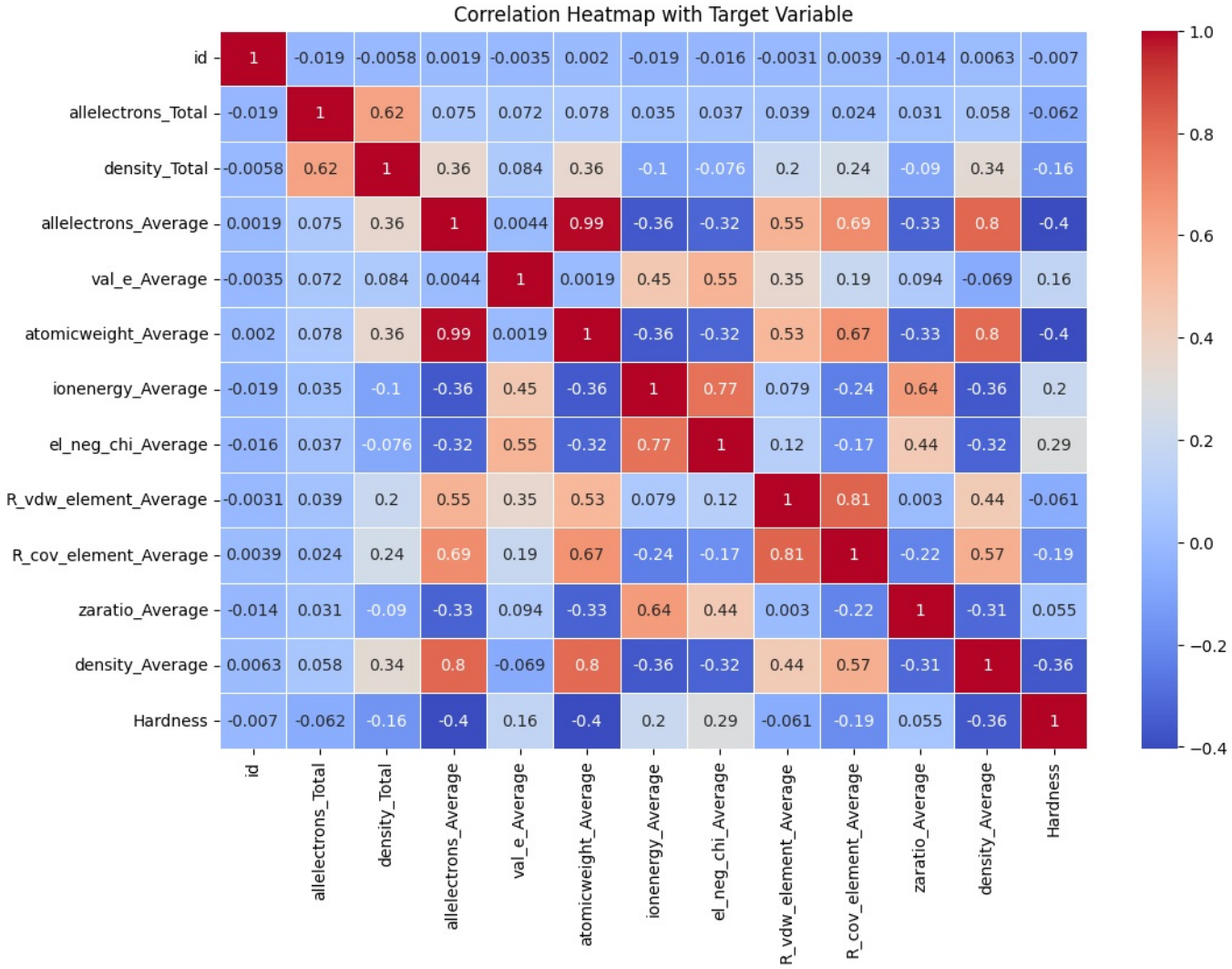
In [ ]:
```
# Pairplot for pairwise relationships
sns.pairplot(train)
plt.suptitle('Pairplot of Features', y=1.02)
plt.show()
```



In [ ]:
```
# Boxplot for each feature to identify outliers
plt.figure(figsize=(12, 8))
sns.boxplot(data=train.drop('id', axis=1))
plt.title('Boxplot of Features')
plt.xticks(rotation=45)
plt.show()
```

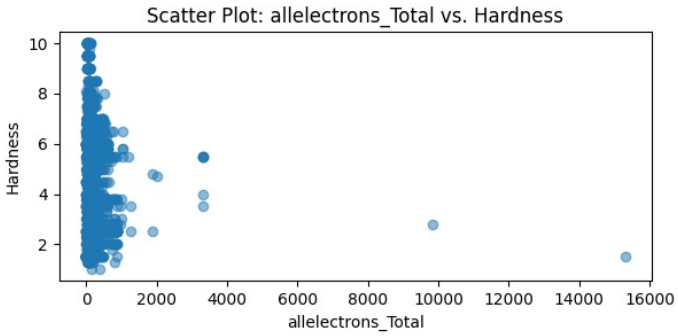


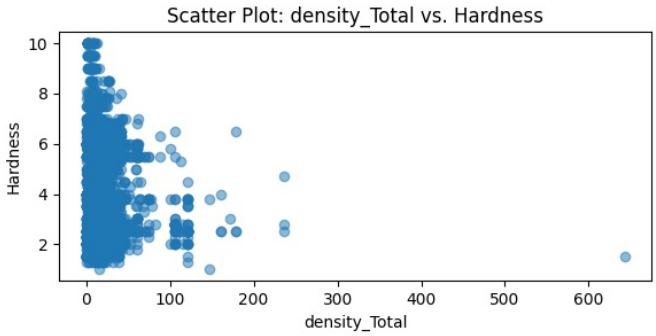
In [ ]:
```
# Correlation matrix heatmap with target variable
plt.figure(figsize=(12, 8))
sns.heatmap(train.corr(), annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Heatmap with Target Variable')
plt.show()
```

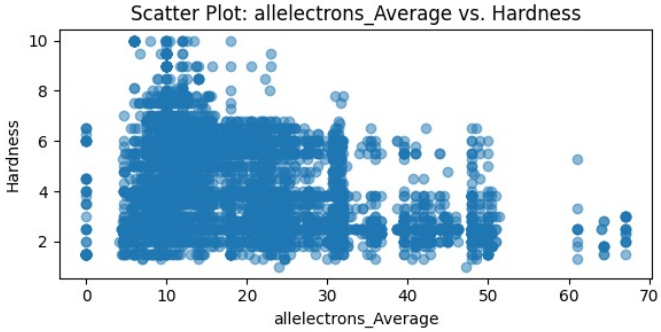


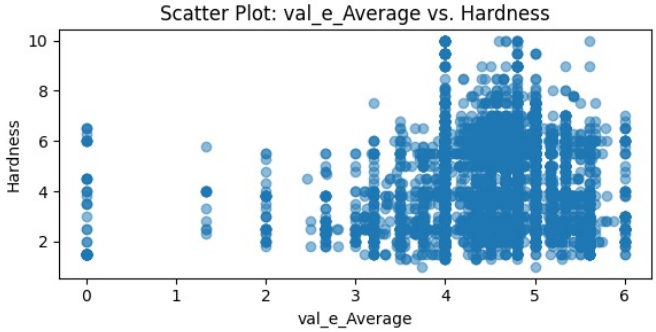
In [ ]:

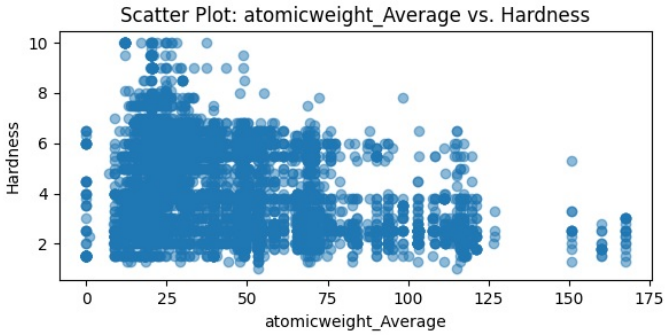
```python
# Scatter plot of the target variable against each feature
num_features = len(train.columns) - 2  # Exclude 'id' and 'Hardness' columns
num_rows = (num_features + 1) // 2  # Add 1 to include the target variable
num_cols = 2

plt.figure(figsize=(12, 3 * num_rows))
for i, col in enumerate(train.columns[1:]):  # Exclude 'id' column
    plt.subplot(num_rows, num_cols, i+1)
    plt.scatter(train[col], train['Hardness'], alpha=0.5)
    plt.title(f'Scatter Plot: {col} vs. Hardness')
    plt.xlabel(col)
    plt.ylabel('Hardness')

plt.tight_layout()
plt.show()
```

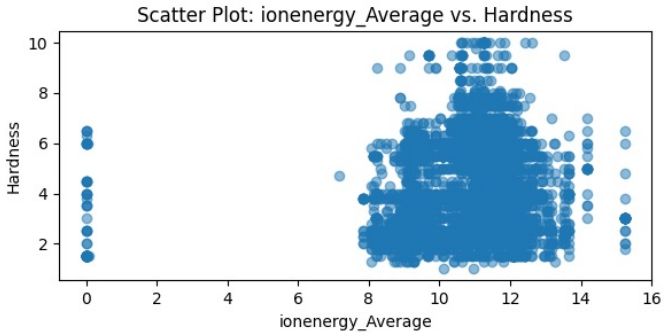


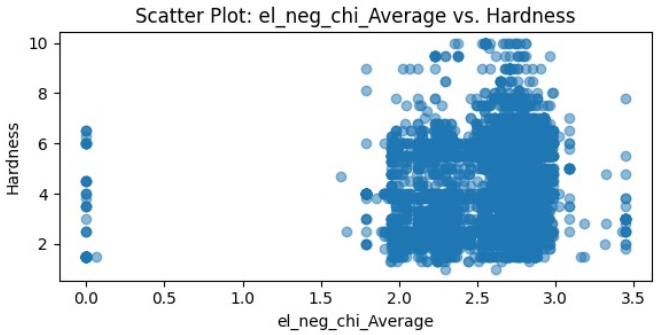


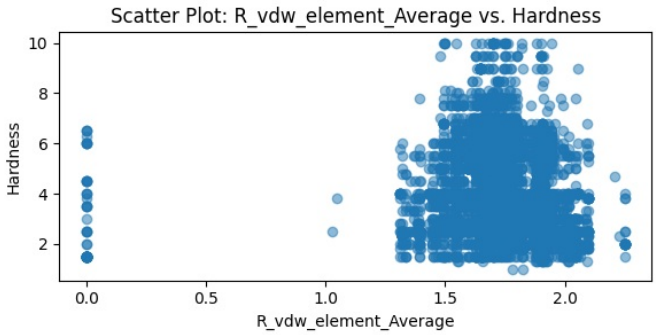


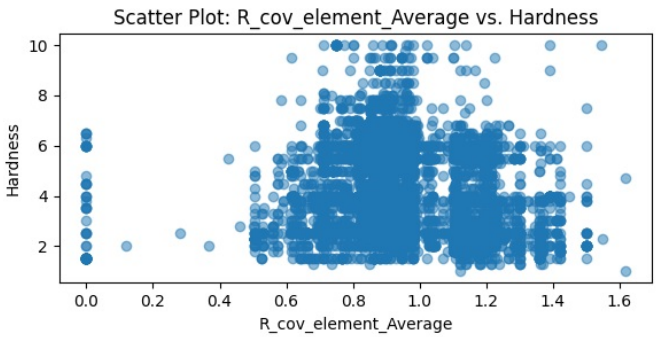


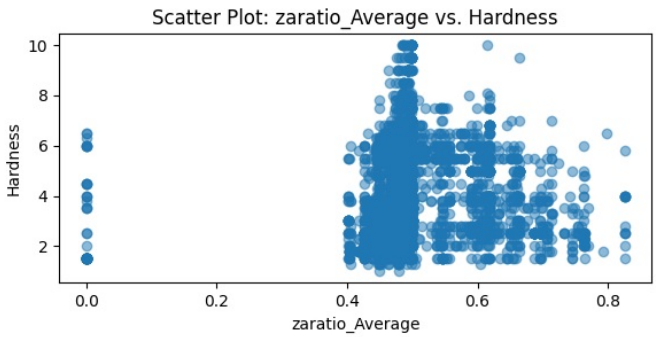


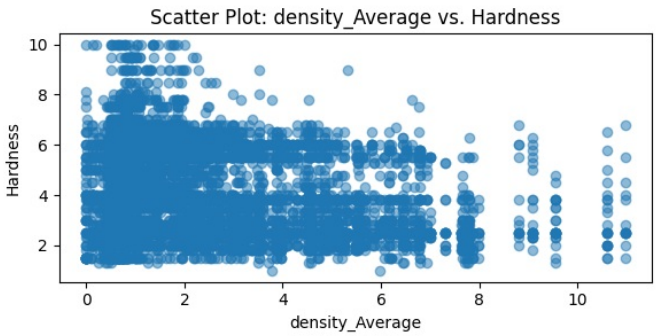


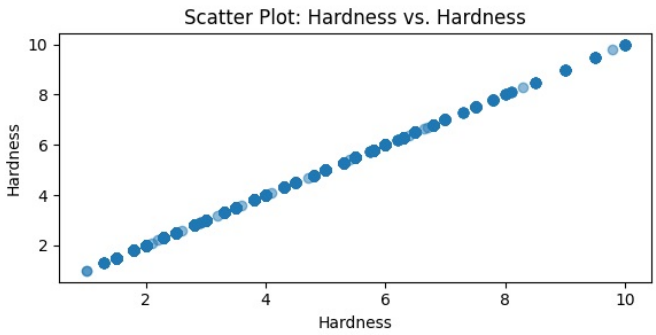












In [ ]:
```
# Bar plot for categorical features (if any)
categorical_cols = train.select_dtypes(include='object').columns
if len(categorical_cols) > 0:
    num_cols = min(2, len(categorical_cols))  # Adjust the number of columns based on the number of categorical
```

```
    num_rows = (len(categorical_cols) + num_cols - 1) // num_cols

    plt.figure(figsize=(12, 4 * num_rows))
    for i, col in enumerate(categorical_cols):
        plt.subplot(num_rows, num_cols, i+1)
        sns.countplot(data=train, x=col)
        plt.title(f'Bar Plot: {col}')
        plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
else:
    print("No categorical features in the dataset.")
```

```
No categorical features in the dataset.
```

3. Data Preprocessing

1. Missing Values:

Check for missing values in the dataset and handle them appropriately. Depending on the extent of missingness, you can either impute missing values or drop the corresponding rows or columns.

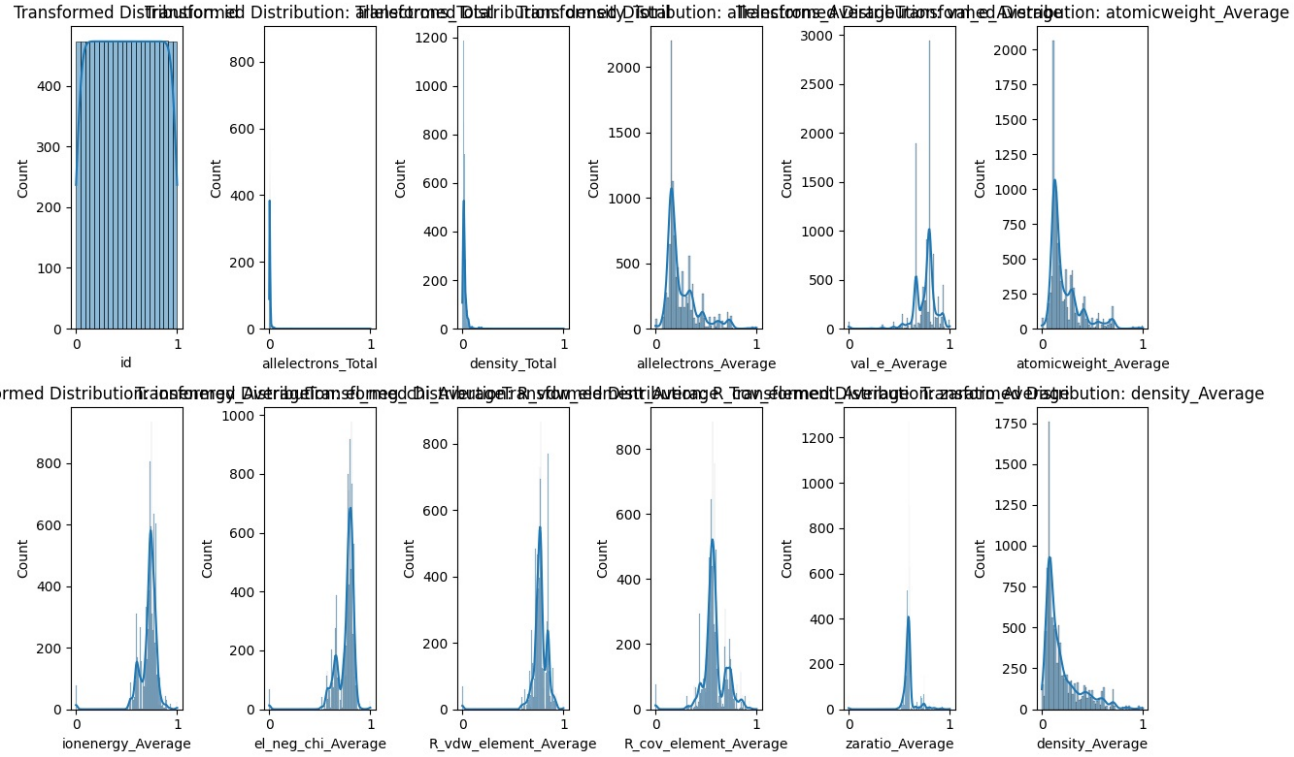
In [ ]:
```
# Check for missing values
missing_values = train.isnull().sum()
# Handle missing values based on your strategy
```

2. Feature Scaling:

Scale numerical features if they are on different scales. Common scaling methods include Min-Max scaling or Standardization (z-score normalization).

In [ ]:
```
# Identify numerical features
numerical_features = train.select_dtypes(include=['float64', 'int64']).columns

# Exclude the target variable 'Hardness'
numerical_features = numerical_features.drop('Hardness', errors='ignore')

# Example of Min-Max scaling
scaler = MinMaxScaler()
train_scaled = pd.DataFrame(scaler.fit_transform(train[numerical_features]), columns=numerical_features)
test_scaled = pd.DataFrame(scaler.transform(test[numerical_features]), columns=numerical_features)
```

In [ ]:
```
# Visualize the distribution of transformed features
plt.figure(figsize=(12, 8))
for i, feature in enumerate(numerical_features):
    plt.subplot(2, len(numerical_features)//2, i+1)
    sns.histplot(train_scaled[feature], kde=True)
    plt.title(f'Transformed Distribution: {feature}')
    plt.xlabel(feature)

plt.tight_layout()
plt.show()
```



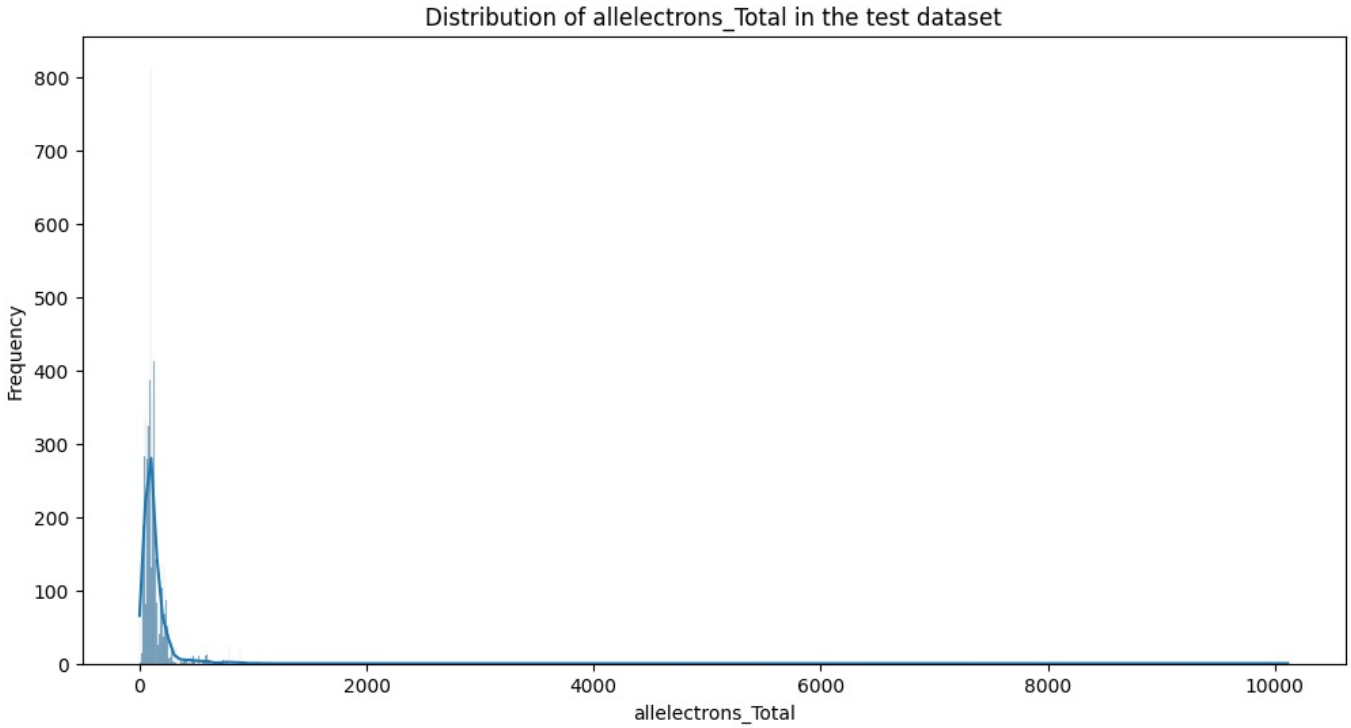
Data Transformation:

1. Skewness Correction:

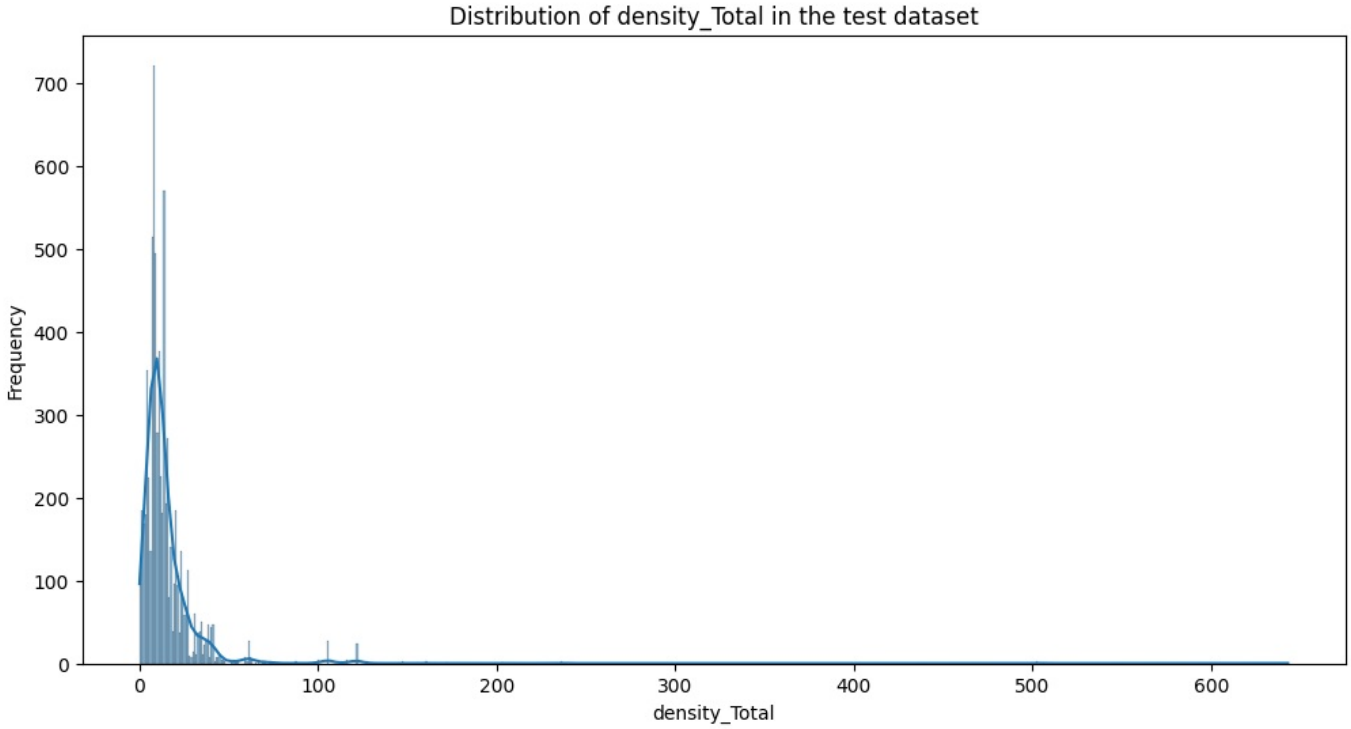
You mentioned that some features are skewed. Consider applying log transformation to right-skewed features and squaring for left-skewed features.

In [ ]:
```
# Identify numerical features for log transformation
numerical_features = train.select_dtypes(include=[np.number]).columns

# Apply log transformation to skewed numerical features
train[numerical_features] = train[numerical_features].apply(lambda x: np.log1p(x) if x.skew() > 0 else np.squar
```

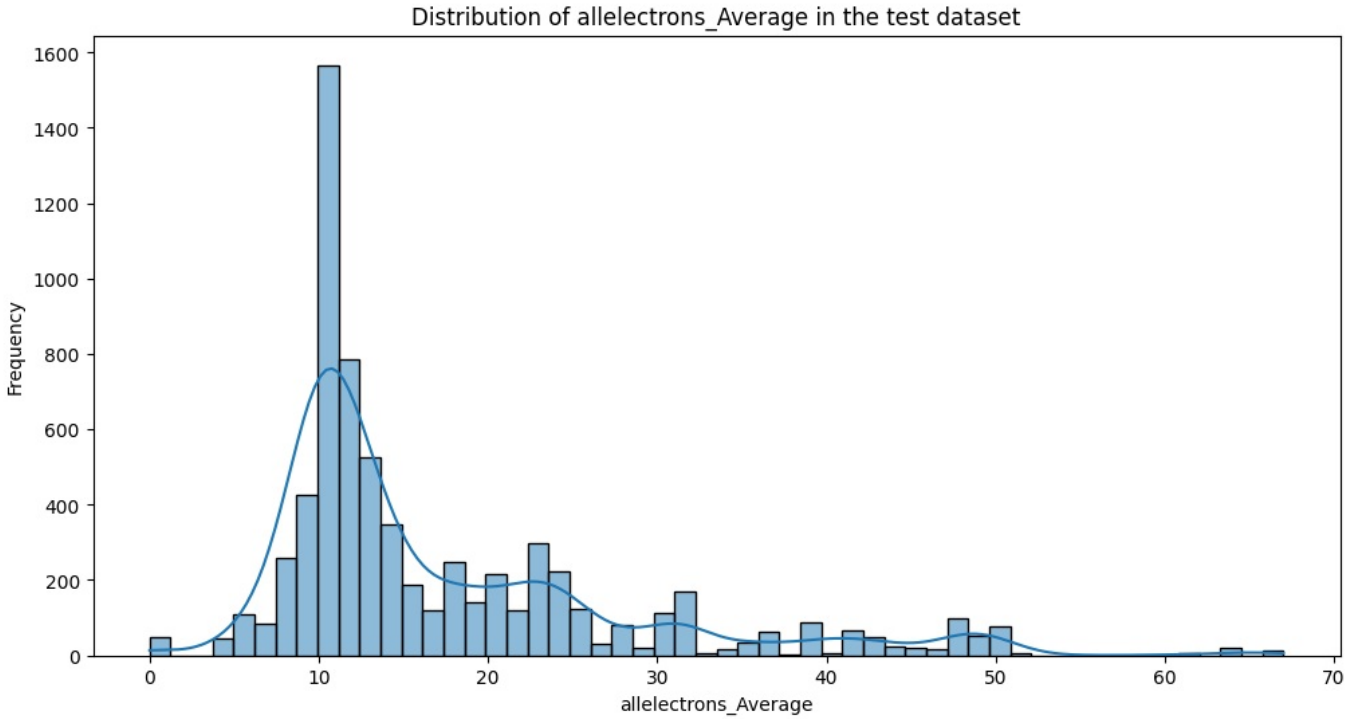
In [ ]:
```
# Display the distribution of each feature
for col in test.columns[1:]:
    plt.figure(figsize=(12, 6))

    # Plot a histogram for numerical features
    if test[col].dtype != 'object':
        sns.histplot(test[col], kde=True)
        plt.title(f'Distribution of {col} in the test dataset')
        plt.xlabel(col)
        plt.ylabel('Frequency')
        plt.show()
    else:
        # Plot a count plot for categorical features
        sns.countplot(x=col, data=test)
        plt.title(f'Count of {col} in the test dataset')
        plt.xlabel(col)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()

    # Print length and missing values information
    print(f'Length of {col} in test:', len(test[col]))
    print(f'Number of missing values in {col} in test:', test[col].isnull().sum())
    print('\n' + '='*50 + '\n')
```



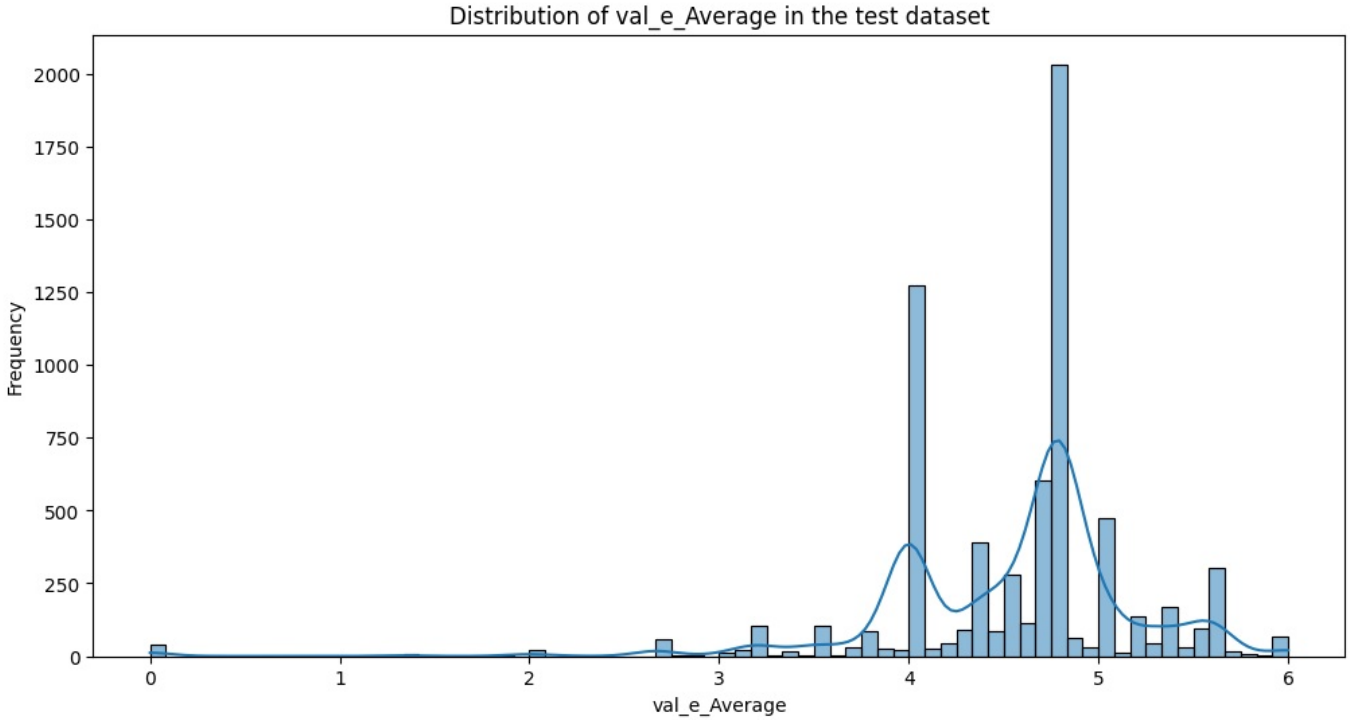

```
Length of allelectrons_Total in test: 6939
Number of missing values in allelectrons_Total in test: 0

==================================================
```



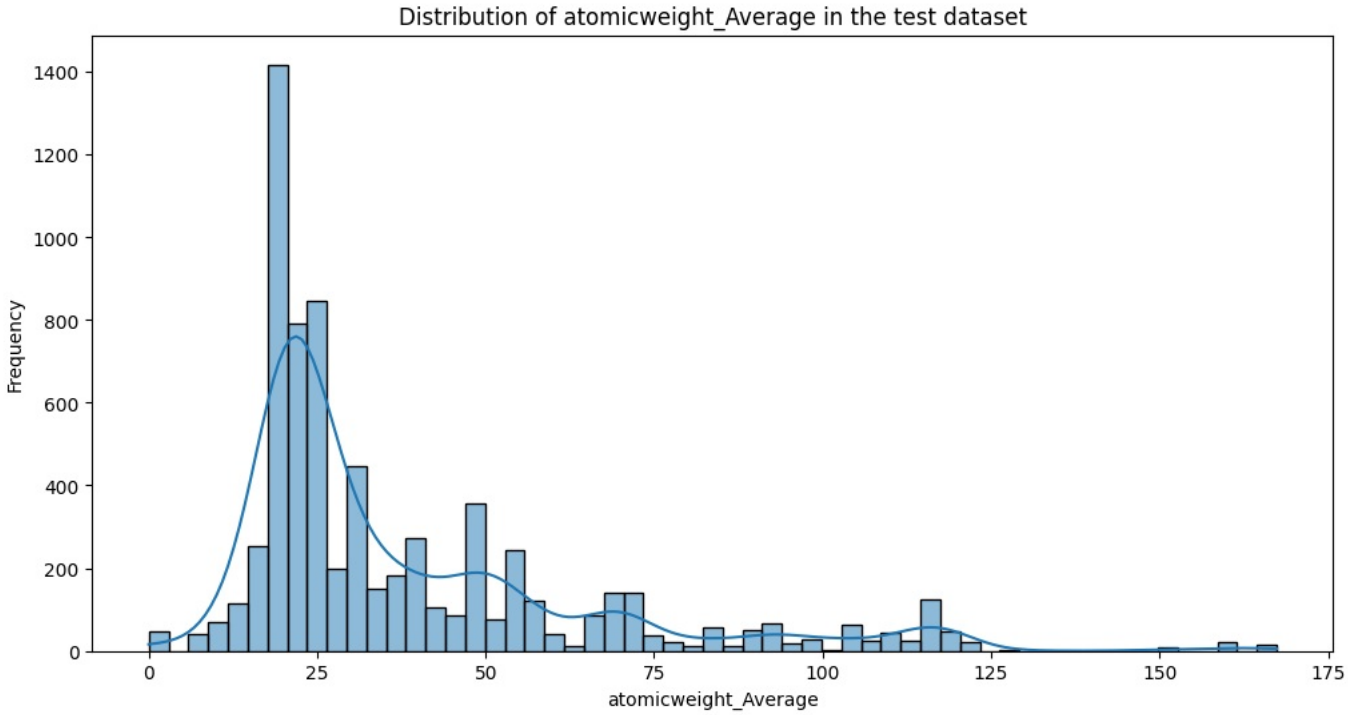

```
Length of density_Total in test: 6939
Number of missing values in density_Total in test: 0

==================================================
```



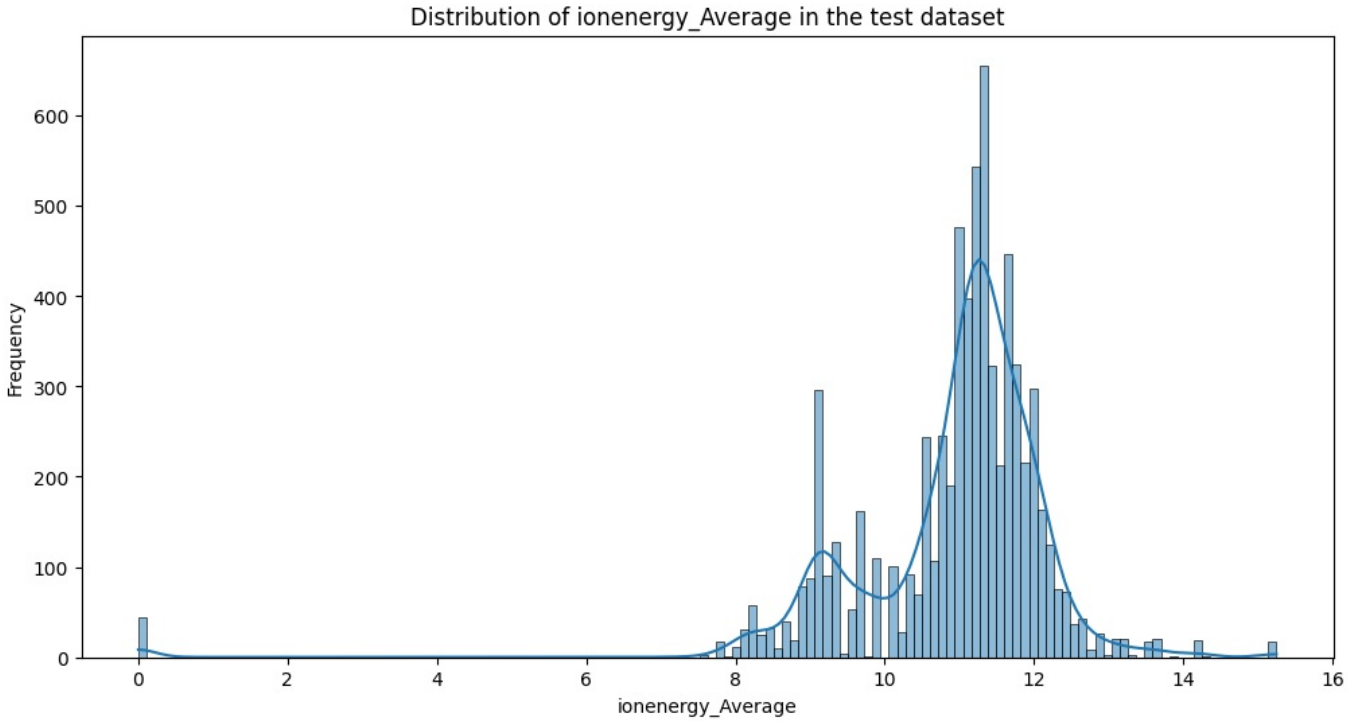

```
Length of allelectrons_Average in test: 6939
Number of missing values in allelectrons_Average in test: 0

================================================
```



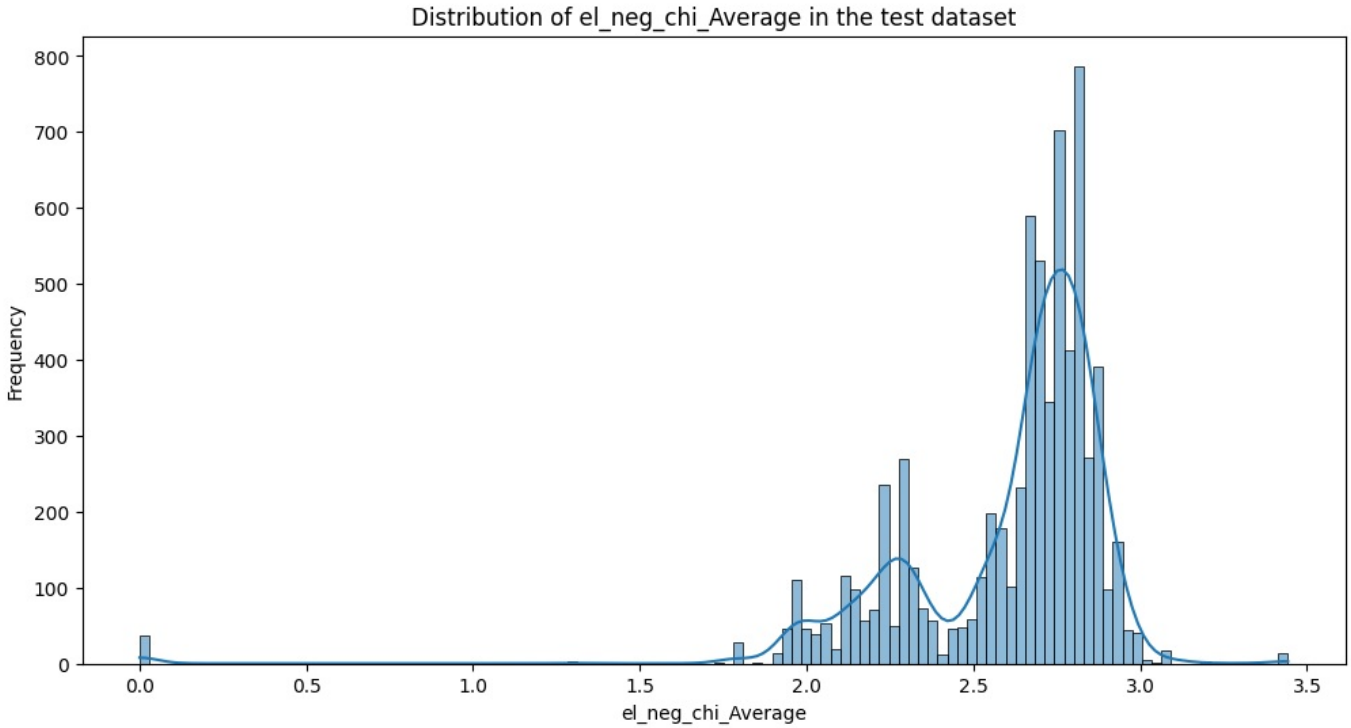

```
Length of val_e_Average in test: 6939
Number of missing values in val_e_Average in test: 0

================================================
```



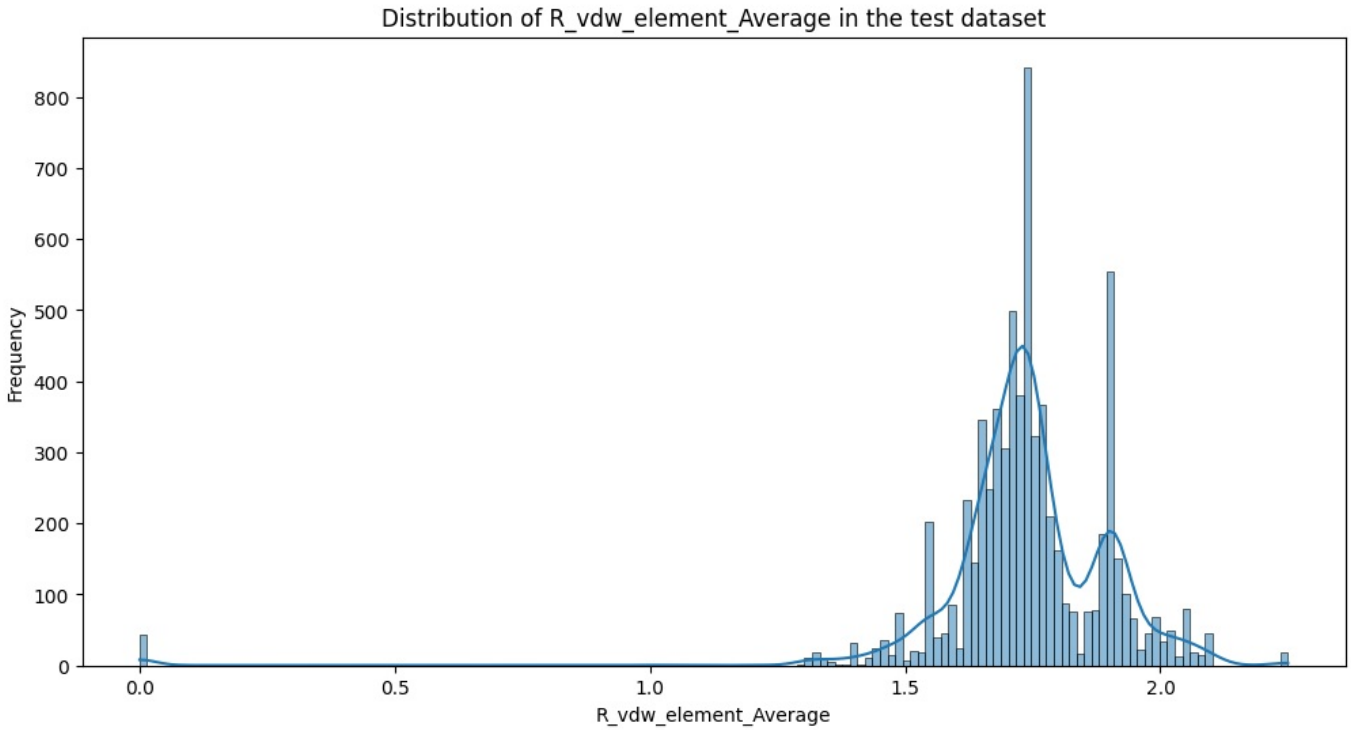

```
Length of atomicweight_Average in test: 6939
Number of missing values in atomicweight_Average in test: 0

==================================================
```



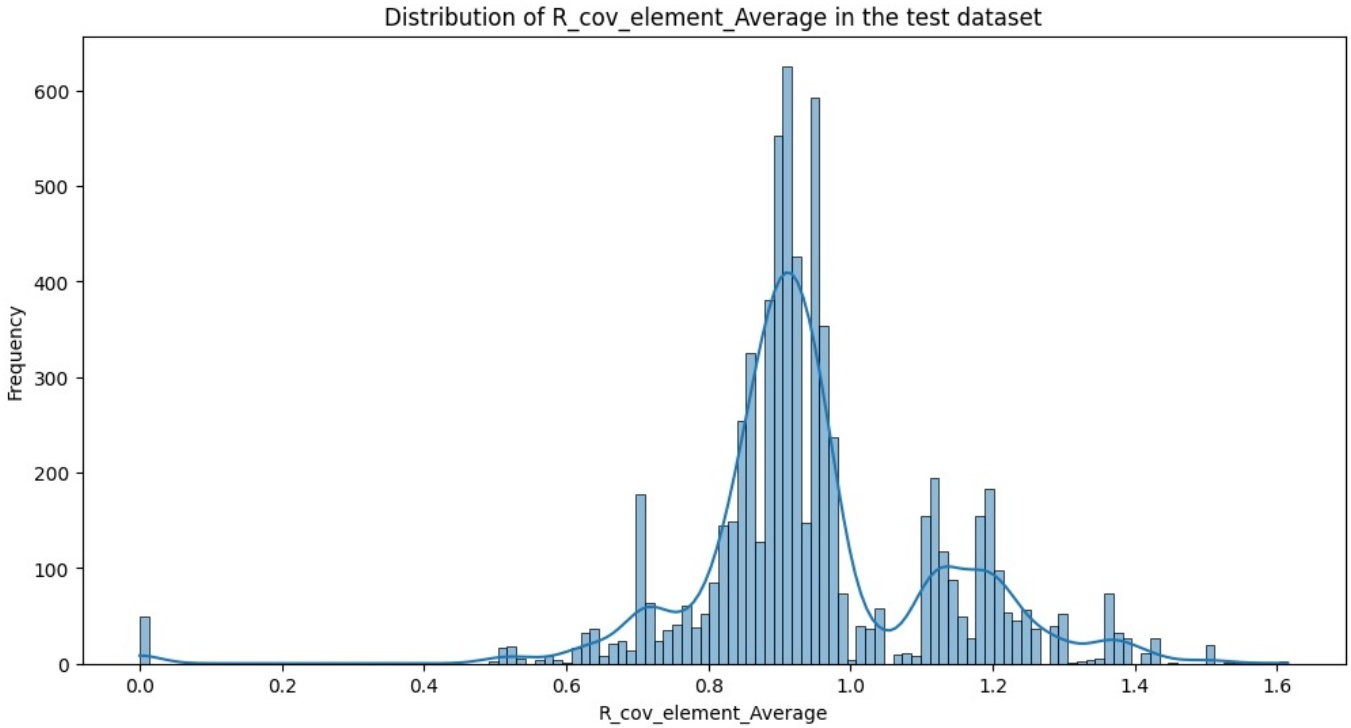

```
Length of ionenergy_Average in test: 6939
Number of missing values in ionenergy_Average in test: 0

==================================================
```



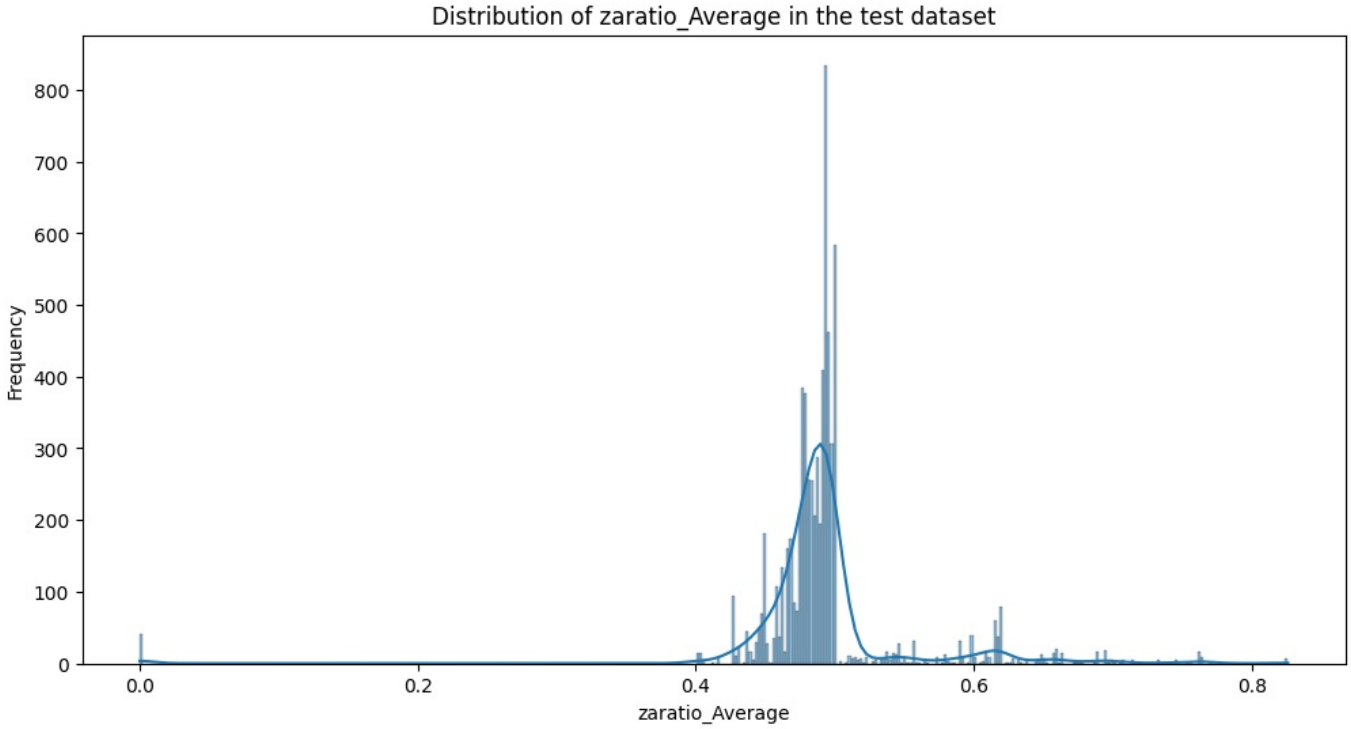

```
Length of el_neg_chi_Average in test: 6939
Number of missing values in el_neg_chi_Average in test: 0

==================================================
```



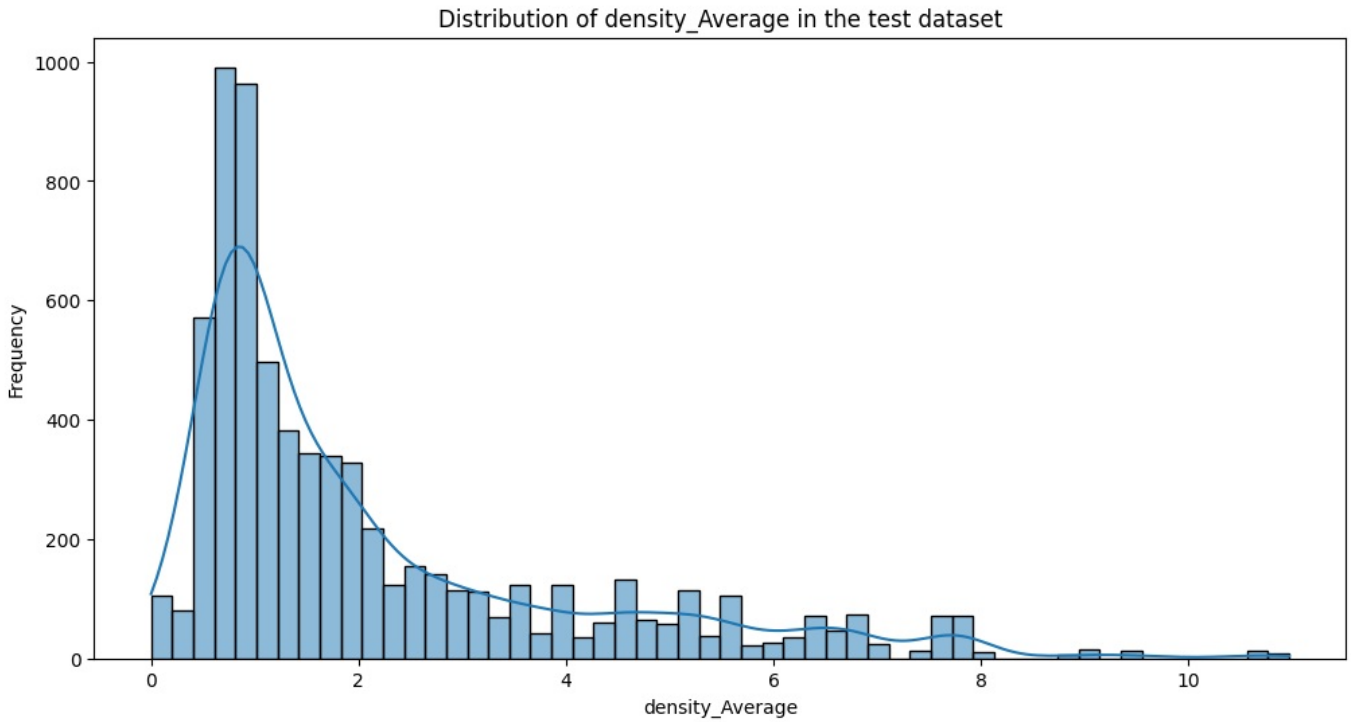

```
Length of R_vdw_element_Average in test: 6939
Number of missing values in R_vdw_element_Average in test: 0

==================================================
```




```
Length of R_cov_element_Average in test: 6939
Number of missing values in R_cov_element_Average in test: 0

==================================================
```




```
Length of zaratio_Average in test: 6939
Number of missing values in zaratio_Average in test: 0

==================================================
```




```
Length of density_Average in test: 6939
Number of missing values in density_Average in test: 0

==================================================
```

1. Scaling:

It seems there was an issue with the scaling part. Ensure that you have defined numerical_features before using it in the Min-Max scaling code.

In [ ]:
```
# Replace 'feature1', 'feature2', ... with actual column names of numerical features
numerical_features = ['allelectrons_Total', 'density_Total', 'allelectrons_Average', 'val_e_Average',
                      'atomicweight_Average', 'ionenergy_Average', 'el_neg_chi_Average',
                      'R_vdw_element_Average', 'R_cov_element_Average', 'zaratio_Average', 'density_Average']

scaler = MinMaxScaler()
```

In [ ]:
```
# Scale the numerical features in the training dataset
train_scaled = pd.DataFrame(scaler.fit_transform(train[numerical_features]), columns=numerical_features)
```

In [ ]:
```
# Scale the numerical features in the test dataset
test_scaled = pd.DataFrame(scaler.transform(test[numerical_features]), columns=numerical_features)
```

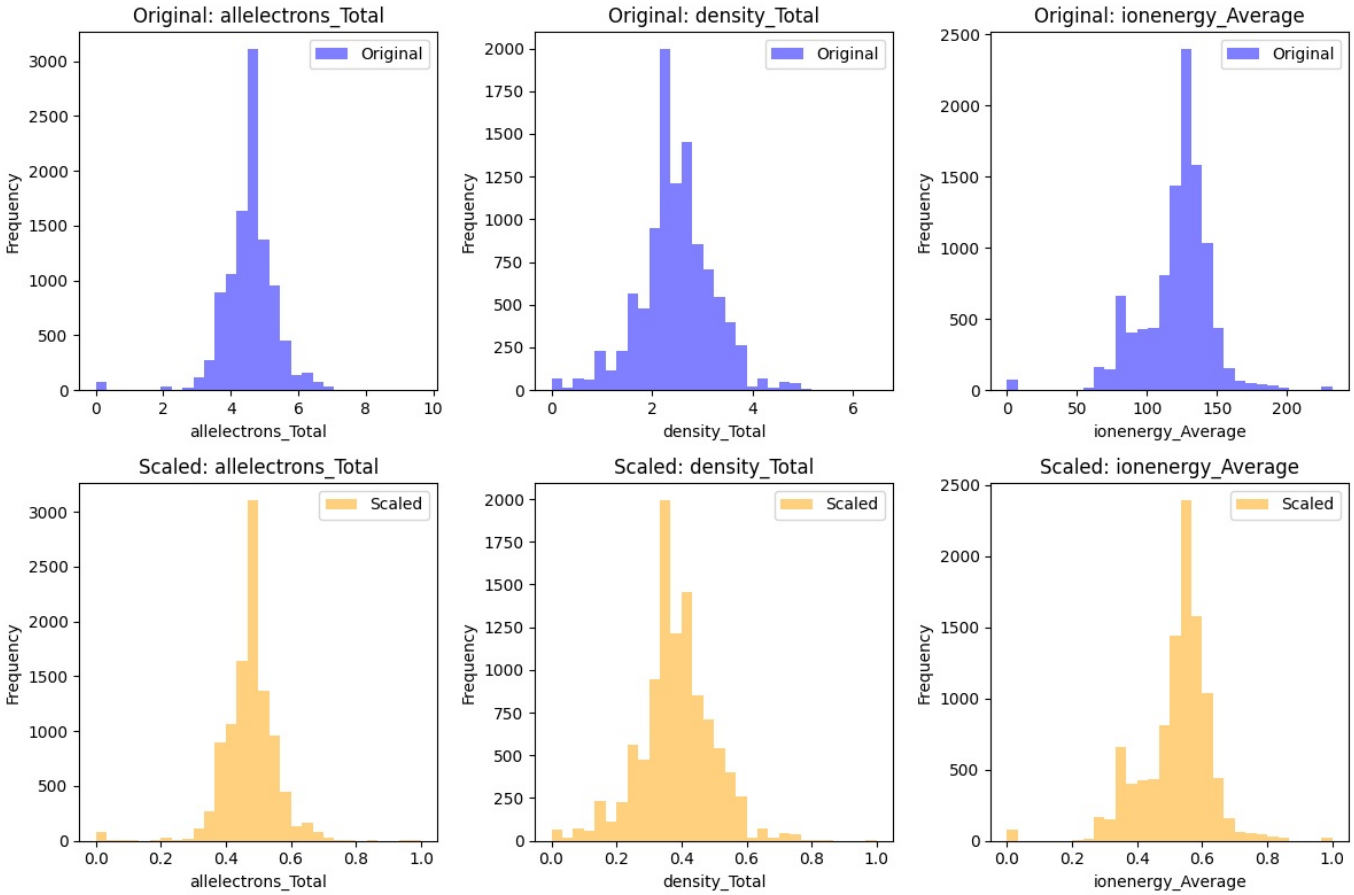
In [ ]:
```
# Display the first few rows of the scaled training dataset
print("Scaled Training Dataset:")
print(train_scaled.head())

# Display the first few rows of the scaled test dataset
print("\nScaled Test Dataset:")
print(test_scaled.head())
```

```
Scaled Training Dataset:
   allelectrons_Total  density_Total  allelectrons_Average  val_e_Average  \
0             0.47896        0.09441               0.56829        0.64000
1             0.47896        0.33194               0.56829        0.64000
2             0.45080        0.35423               0.66581        0.87111
3             0.47896        0.35281               0.56829        0.64000
4             0.49422        0.36469               0.60047        0.64000

   atomicweight_Average  ionenergy_Average  el_neg_chi_Average  \
0               0.59951            0.52895             0.64540
1               0.59665            0.62375             0.64028
2               0.69209            0.62847             0.67466
3               0.59587            0.51571             0.59151
4               0.63547            0.60154             0.64540

   R_vdw_element_Average  R_cov_element_Average  zaratio_Average  \
0                0.59256                0.28327          0.36069
1                0.52546                0.31717          0.35584
2                0.63150                0.28591          0.33978
3                0.52225                0.33555          0.35087
4                0.55884                0.30748          0.35586

   density_Average
0          0.26164
1          0.21790
2          0.37013
3          0.23440
4          0.42399

Scaled Test Dataset:
   allelectrons_Total  density_Total  allelectrons_Average  val_e_Average  \
0            91.74242       18.77287               8.38012        0.14667
1             9.34029        1.53559               4.26590        0.15556
2            12.03860        1.20102               2.74914        0.13333
3            10.37810        1.40820               2.36994        0.13333
4             5.70796        0.62308               2.60694        0.11111

   atomicweight_Average  ionenergy_Average  el_neg_chi_Average  \
0              16.10529            0.04031             0.19385
1               7.71858            0.05200             0.24194
2               4.53185            0.04743             0.22304
3               3.95972            0.05200             0.23856
4               4.48227            0.04853             0.20710

   R_vdw_element_Average  R_cov_element_Average  zaratio_Average  \
0                0.37673                0.45960          0.67701
1                0.32632                0.33092          0.69797
2                0.35437                0.36768          0.73215
3                0.32830                0.30334          0.72670
4                0.34568                0.34215          0.69782

   density_Average
0          0.72292
1          0.56878
2          0.31757
3          0.48528
4          0.37647
```

In [ ]:

```python
# Print the column names
print(train.columns)

# Replace 'feature1', 'feature2', 'feature3' with actual column names
selected_features = ['allelectrons_Total', 'density_Total', 'ionenergy_Average']

# The rest of the visualization code remains the same
plt.figure(figsize=(12, 8))
for i, feature in enumerate(selected_features):
    plt.subplot(2, len(selected_features), i + 1)
    plt.hist(train[feature], bins=30, color='blue', alpha=0.5, label='Original')
    plt.title(f'Original: {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.legend()

    plt.subplot(2, len(selected_features), i + 1 + len(selected_features))
    plt.hist(train_scaled[feature], bins=30, color='orange', alpha=0.5, label='Scaled')
    plt.title(f'Scaled: {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.legend()

plt.tight_layout()
plt.show()
```

```
Index(['id', 'allelectrons_Total', 'density_Total', 'allelectrons_Average',
       'val_e_Average', 'atomicweight_Average', 'ionenergy_Average',
       'el_neg_chi_Average', 'R_vdw_element_Average', 'R_cov_element_Average',
       'zaratio_Average', 'density_Average', 'Hardness'],
      dtype='object')
```



4. Feature Engineering:

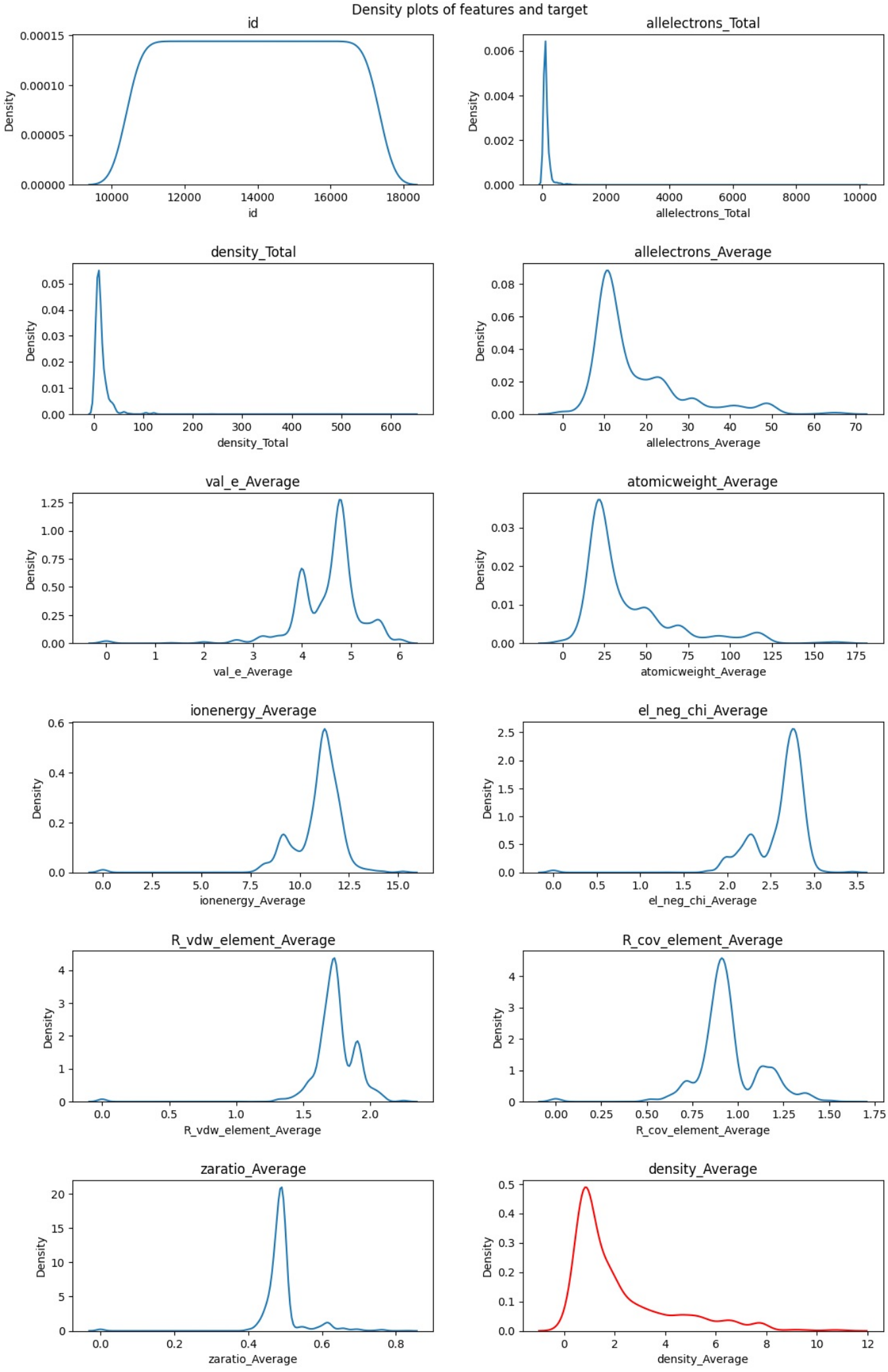
Create new features that might be informative for the prediction task. For example, you can derive features from existing ones or use domain knowledge to generate meaningful features.

Data Exploration and Visualization:

4.1 Density Plots:

You can create density plots for the numerical features in your training dataset using the generate_density_plots function you shared earlier. Make sure to adjust the code to match the column names in your dataset.

In [ ]:
```
generate_density_plots(data=test, title='Density plots of features and target')
```



4.2 Outlier Detection and Handling:

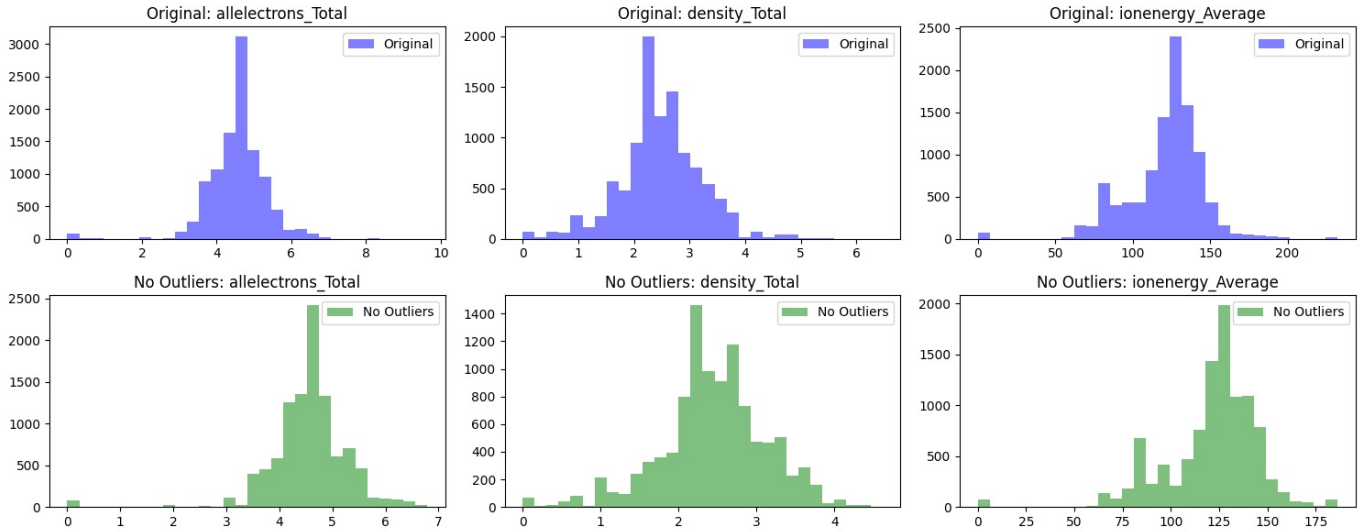
Identify and handle outliers in the data. You can use visualization techniques or statistical methods to detect outliers and decide whether to remove or transform them.

In [ ]:
```
# Define the numerical features
numerical_features = ['allelectrons_Total', 'density_Total', 'ionenergy_Average']

# Calculate z-scores
z_scores = zscore(train[numerical_features])

# Create a boolean mask for outliers (values with z-score greater than 3)
outlier_mask = (z_scores < 3).all(axis=1)

# Create a new DataFrame without outliers
train_no_outliers = train[outlier_mask]

# Print or visualize the results
print("Original Shape:", train.shape)
print("Shape after removing outliers:", train_no_outliers.shape)

# You can also display the rows with outliers if needed
outliers = train[~outlier_mask]
print("Outliers:")
print(outliers)
```

```
Original Shape: (10407, 13)
Shape after removing outliers: (10254, 13)
Outliers:
             id  allelectrons_Total  density_Total  allelectrons_Average  \
21          441             6.78559        5.08317               3.59778
65         4225             7.54697        3.25894               2.19103
119       14161             6.78559        4.80746               3.59347
138       19044             6.67330        4.66828               3.70130
329      108241             6.67330        4.66828               3.70130
...         ...                 ...            ...                   ...
10018  100360324             5.31321        3.17022               3.05400
10102  102050404             6.78559        4.80746               3.59347
10295  105987025             4.67283        2.44296               3.06805
10305  106193025             5.35186        4.80746               3.59347
10403  108222409             3.43399        1.00911               2.39790

       val_e_Average  atomicweight_Average  ionenergy_Average  \
21          10.06421               4.39533           77.19695
65          19.26235               2.84553          131.50407
119         27.87840               4.42558           87.80410
138         31.36000               4.55161           86.00439
329         27.04000               4.55161          115.75866
...              ...                   ...                ...
10018       36.00000               3.87775          232.43472
10102       27.87840               4.42558           87.80410
10295       25.00000               3.85954          232.43472
10305       27.87840               4.42558           87.80410
10403       28.44444               3.08039          200.61701

       el_neg_chi_Average  R_vdw_element_Average  R_cov_element_Average  \
21                4.68826                3.64264                1.35522
65                7.22714                2.38531                0.54104
119               5.28080                3.63741                1.44000
138               5.19384                3.63741                1.47501
329               7.04902                3.28176                1.05678
...                   ...                    ...                    ...
10018            11.85425                2.76224                0.69890
10102             5.28080                3.63741                1.44000
10295             7.78410                2.79726                0.91202
10305             5.28080                3.63741                1.44000
10403             9.54810                2.42321                0.75111

       zaratio_Average  density_Average  Hardness
21             0.21484          1.96900   6.25000
65             0.36193          0.43798   6.25000
119            0.21335          1.87183   6.25000
138            0.20899          1.80980   7.84000
329            0.20060          1.79228   9.00000
...                ...              ...       ...
10018          0.21160          1.19755   4.00000
10102          0.21335          2.05105   6.25000
10295          0.21025          1.58207  42.25000
10305          0.21160          1.58373  12.25000
10403          0.23077          0.59598  25.00000

[153 rows x 13 columns]
```

In [ ]:
```
# Visualize the distribution of each numerical feature before and after removing outliers
plt.figure(figsize=(15, 6))
```

```
for i, feature in enumerate(numerical_features):
    plt.subplot(2, len(numerical_features), i + 1)
    plt.hist(train[feature], bins=30, color='blue', alpha=0.5, label='Original')
    plt.title(f'Original: {feature}')
    plt.legend()

    plt.subplot(2, len(numerical_features), i + len(numerical_features) + 1)
    plt.hist(train_no_outliers[feature], bins=30, color='green', alpha=0.5, label='No Outliers')
    plt.title(f'No Outliers: {feature}')
    plt.legend()

plt.tight_layout()
plt.show()
```



4.3 Advanced Feature Analysis:

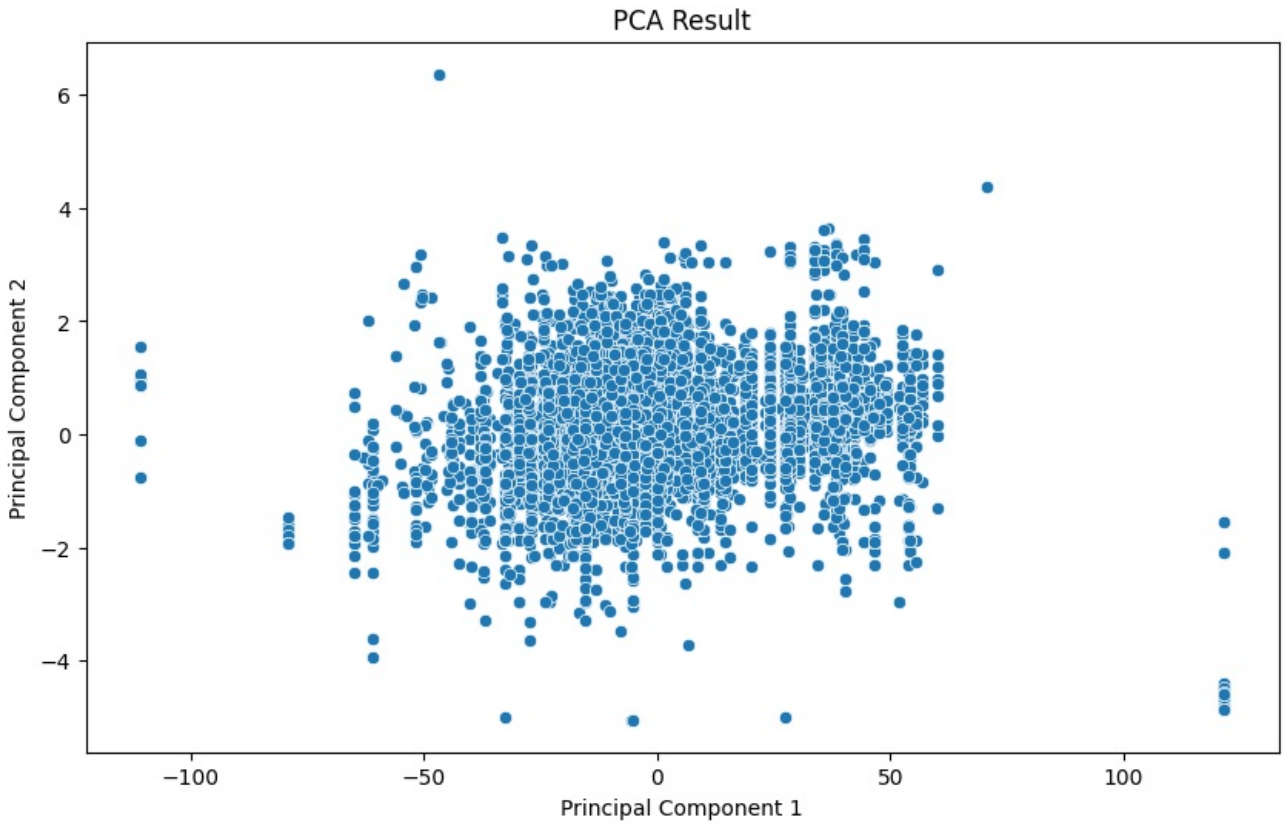
Explore relationships between features using advanced techniques like PCA (Principal Component Analysis) or LDA (Linear Discriminant Analysis) for dimensionality reduction.

In [ ]:
```
# Example of PCA
pca = PCA(n_components=2)
pca_result = pca.fit_transform(train[numerical_features])
```

In [ ]:
```
# Visualize the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_
print(f'Explained Variance Ratio: {explained_variance_ratio}')
```

```
Explained Variance Ratio: [0.99825643 0.00154358]
```

In [ ]:
```
# Scatter plot of the PCA result
plt.figure(figsize=(10, 6))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1])
plt.title('PCA Result')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



5. Modeling

In [ ]:
```
# First, identify independent variables (X) and the dependent variable (y)
X = train[numerical_features].values
y = train['Hardness'].values
```

In [ ]:
```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
# Choose a model (e.g., a simple linear regression model)
model = LinearRegression()
```

In [ ]:
```
# Train the model
model.fit(X_train, y_train)
```

Out[ ]:
```
▾ LinearRegression
LinearRegression()
```

In [ ]:
```
# Make predictions on the training set
train_predictions = model.predict(X_train)
```

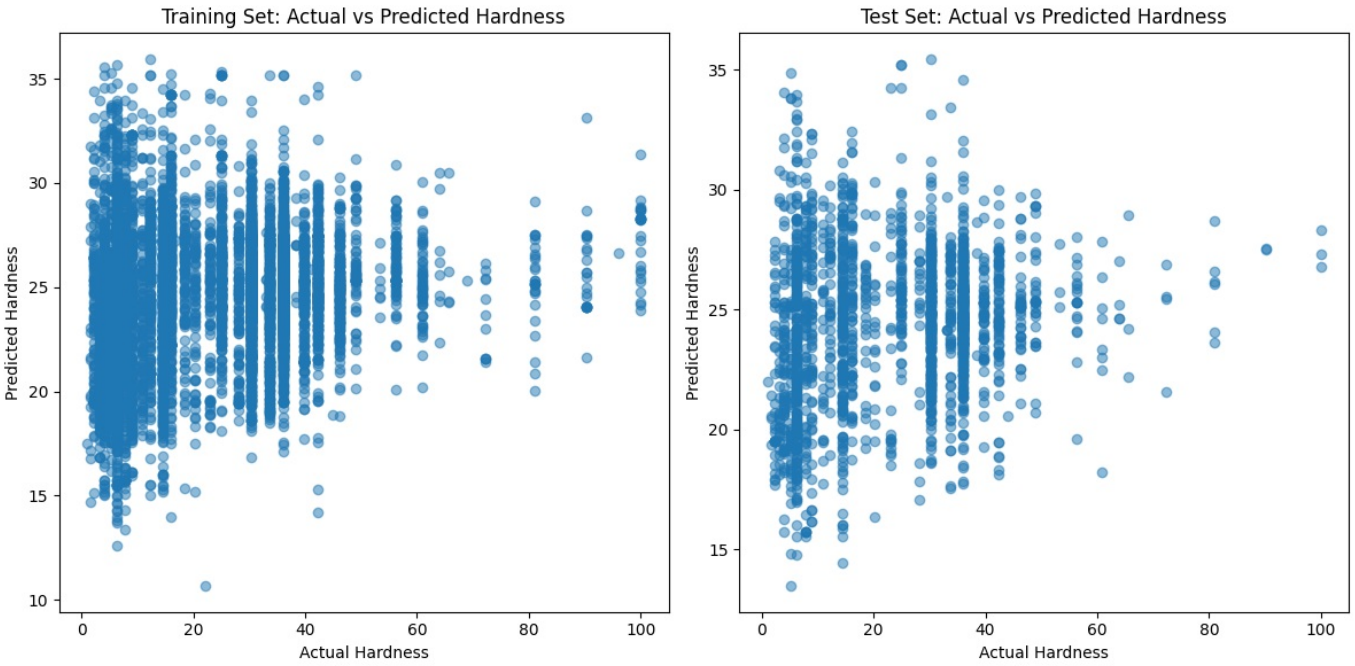
In [ ]:
```
# Make predictions on the test set
test_predictions = model.predict(X_test)
```

In [ ]:
```
# Evaluate the model's performance
train_rmse = np.sqrt(mean_squared_error(y_train, train_predictions))
test_rmse = np.sqrt(mean_squared_error(y_test, test_predictions))
r2 = r2_score(y_test, test_predictions)

print(f'Training RMSE: {train_rmse}')
print(f'Test RMSE: {test_rmse}')
print(f'R^2 Score: {r2}')
```

```
Training RMSE: 15.38778690173805
Test RMSE: 14.81396104307172
R^2 Score: 0.030257195213989507
```

In [ ]:
```
# Scatter plot for training set
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(y_train, train_predictions, alpha=0.5)
plt.title('Training Set: Actual vs Predicted Hardness')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')
```

```
# Scatter plot for test set
plt.subplot(1, 2, 2)
plt.scatter(y_test, test_predictions, alpha=0.5)
plt.title('Test Set: Actual vs Predicted Hardness')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')

plt.tight_layout()
plt.show()
```



Model Performance Metrics:

R-Squared (R2) Value:

R-Squared measures the explanatory power of the model. The closer it is to 1, the better the model explains the data. R-Squared is calculated as follows:

[R^2 = 1 - \frac{\text{MSE}}{\text{Var}(y)}]

- (R^2): R-Squared value
- (\text{MSE}): Mean Squared Error
- (\text{Var}(y)): Variance of the actual values

Mean Absolute Error (MAE):

MAE measures how far the model's predictions are from the actual values. It is calculated by taking the average of the absolute differences between each prediction and its corresponding actual value:

[MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|]

- (n): Number of observations
- (y_i): Actual value
- (\hat{y}_i): Model's prediction

Mean Squared Error (MSE):

MSE calculates the average of the squared differences between predictions and actual values. MSE emphasizes larger errors, as it squares the differences. The formula is as follows:

[MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2]

- (n): Number of observations
- (y_i): Actual value
- (\hat{y}_i): Model's prediction

These metrics are used to evaluate the performance of the model and understand how close the predictions are to the actual values.

In [ ]:
```
# Train set performance
train_r2 = r2_score(y_train, train_predictions)
train_mae = mean_absolute_error(y_train, train_predictions)
train_mse = mean_squared_error(y_train, train_predictions)
```

In [ ]:
```
# Test set performance
test_r2 = r2_score(y_test, test_predictions)
test_mae = mean_absolute_error(y_test, test_predictions)
test_mse = mean_squared_error(y_test, test_predictions)
```

In [ ]:
```
print(f'Train R2: {train_r2:.4f}, MAE: {train_mae:.4f}, MSE: {train_mse:.4f}')
print(f'Test R2: {test_r2:.4f}, MAE: {test_mae:.4f}, MSE: {test_mse:.4f}')
```

```
Train R2: 0.0394, MAE: 12.9016, MSE: 236.7840
Test R2: 0.0303, MAE: 12.7000, MSE: 219.4534
```

Model Hyperparameter Tuning:

- You can improve the performance of the model by adjusting its hyperparameters using techniques such as Grid Search or Random Search.
- For example, using GridSearchCV:

Assuming you have X_train and y_train as your features and target variable

In [ ]:
```
# Perform cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='neg_mean_squared_error')
```

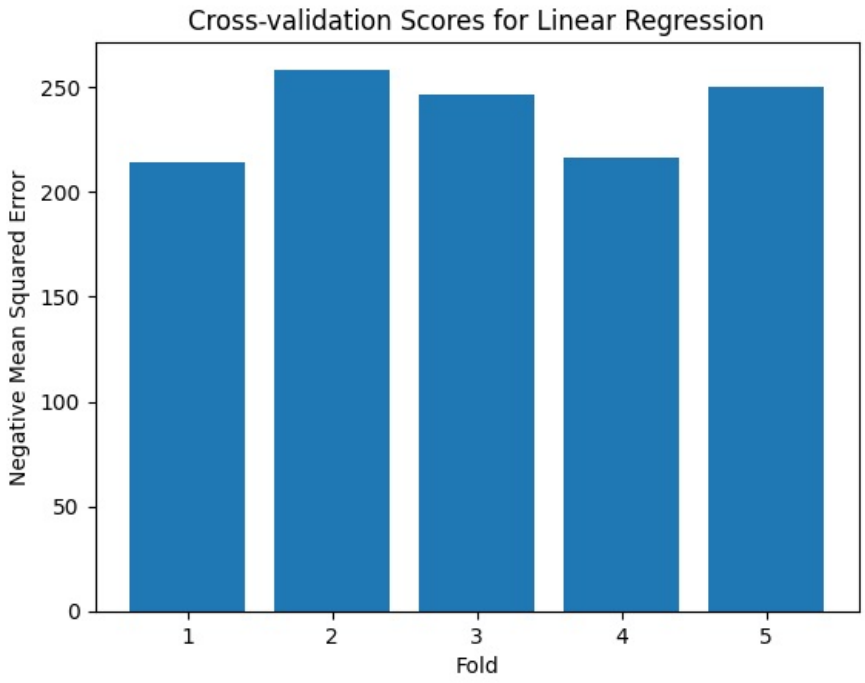
In [ ]:
```
# Display the cross-validation scores
print("Cross-validation Scores (negative mean squared error):", cv_scores)
```

```
Cross-validation Scores (negative mean squared error): [-214.08808939 -258.53350372 -246.56738888 -216.79821938
-249.92110254]
```

In [ ]:
```
# Calculate the mean of the cross-validation scores
mean_cv_score = np.mean(cv_scores)
print("Mean Cross-validation Score:", mean_cv_score)
```

```
Mean Cross-validation Score: -237.18166078529626
```

In [ ]:
```
# Visualize cross-validation scores
plt.bar(range(1, len(cv_scores) + 1), -cv_scores)
plt.xlabel('Fold')
plt.ylabel('Negative Mean Squared Error')
plt.title('Cross-validation Scores for Linear Regression')
plt.show()
```



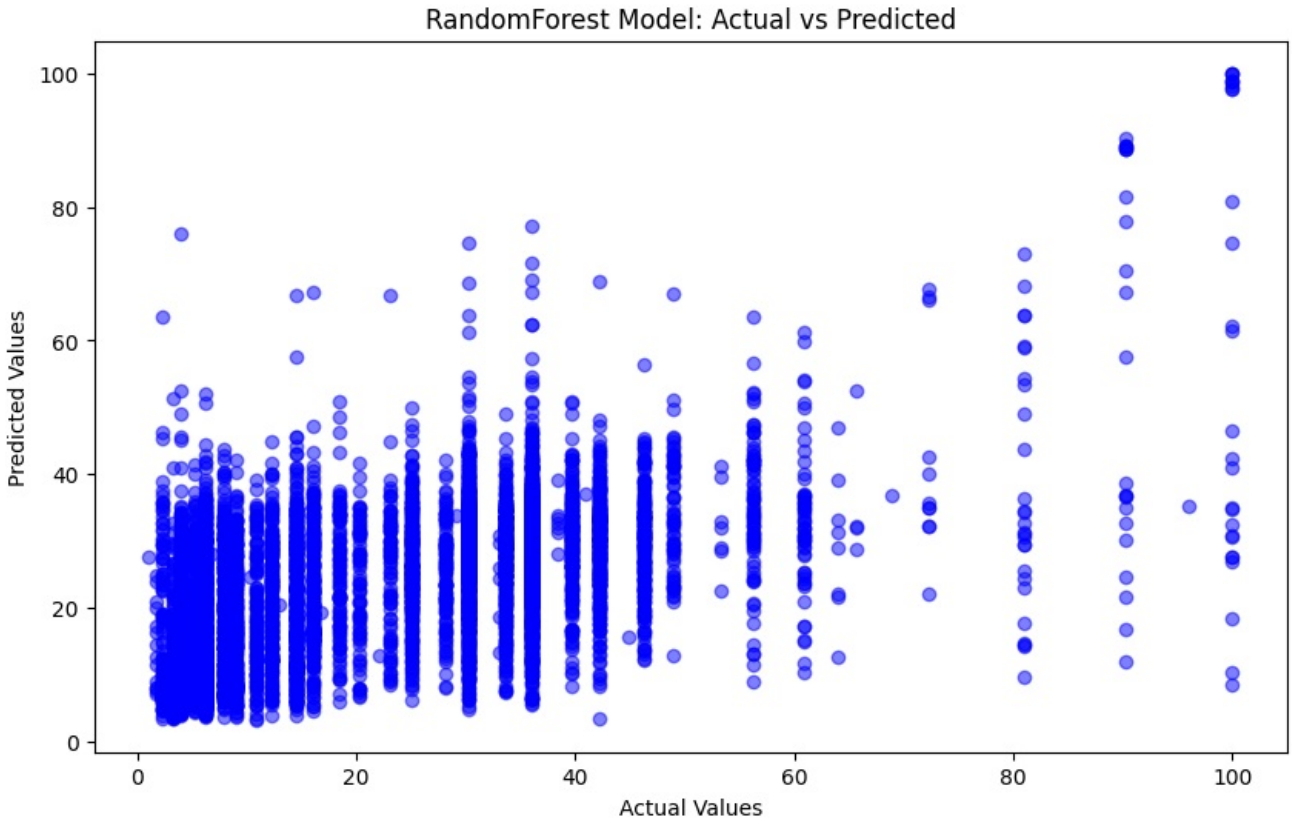
5.1 Baseline RandomForest Model

In [ ]:
```
rf_model = RandomForestRegressor(random_state=42)
rf_preds = cross_val_predict(rf_model, X_train, y_train, cv=5)
rf_mse = mean_squared_error(y_train, rf_preds)
```

```
print(f'RandomForest Mean Squared Error: {rf_mse}')
```

```
RandomForest Mean Squared Error: 174.46564543909415
```

In [ ]:
```
# Scatter plot for RandomForest predictions vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_train, rf_preds, color='blue', alpha=0.5)
plt.title('RandomForest Model: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

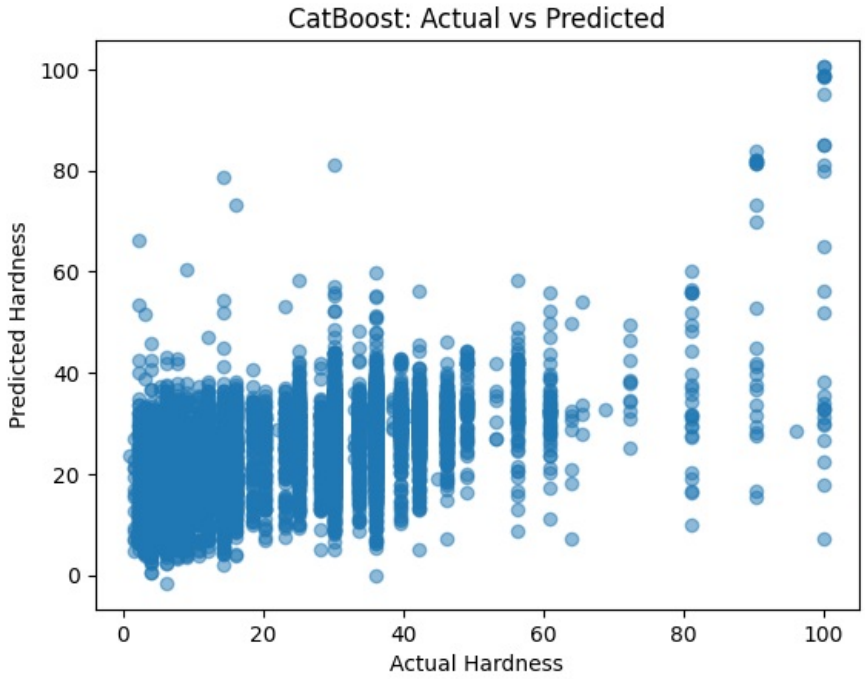


5.2 Baseline CatBoost Model

In [ ]:
```
catboost_model = CatBoostRegressor(random_state=42, verbose=0)
catboost_preds = cross_val_predict(catboost_model, X_train, y_train, cv=5)
catboost_mse = mean_squared_error(y_train, catboost_preds)
print(f'CatBoost Mean Squared Error: {catboost_mse}')
```

```
CatBoost Mean Squared Error: 160.00326742325356
```

In [ ]:
```
# Scatter plot for CatBoost predictions vs actual values
plt.scatter(y_train, catboost_preds, alpha=0.5)
plt.title('CatBoost: Actual vs Predicted')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')
plt.show()
```

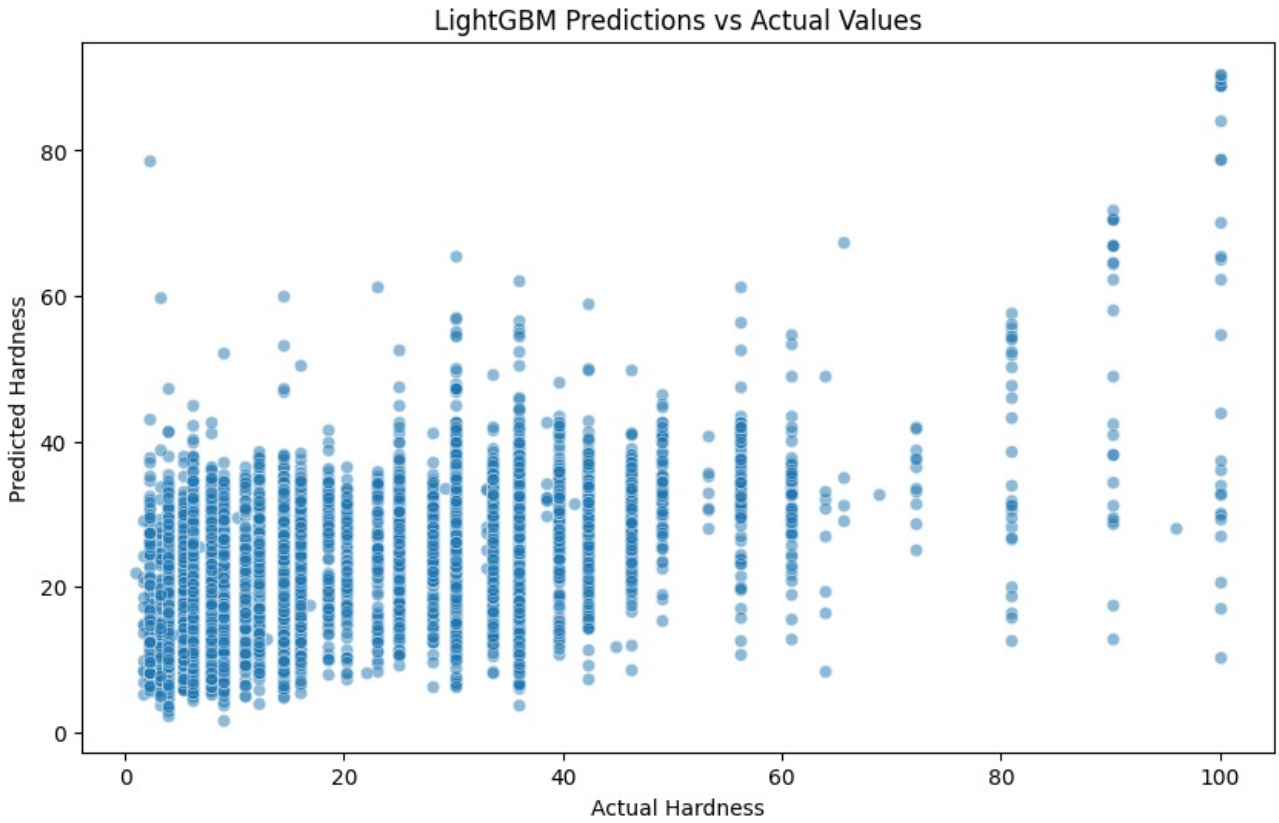


5.3 LGBM Model

In [ ]:
```
lgbm_model = LGBMRegressor(random_state=42)
lgbm_preds = cross_val_predict(lgbm_model, X_train, y_train, cv=5)
lgbm_mse = mean_squared_error(y_train, lgbm_preds)
print(f'LGBM Mean Squared Error: {lgbm_mse}')
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000546 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 711
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.663579
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000298 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.220146
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000142 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 708
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.307672
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.001020 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.638069
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000310 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.541583
LGBM Mean Squared Error: 160.67157757707898
```

In [ ]:
```
# Scatter plot for LightGBM predictions vs actual values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_train, y=lgbm_preds, alpha=0.5)
plt.title('LightGBM Predictions vs Actual Values')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')
plt.show()
```

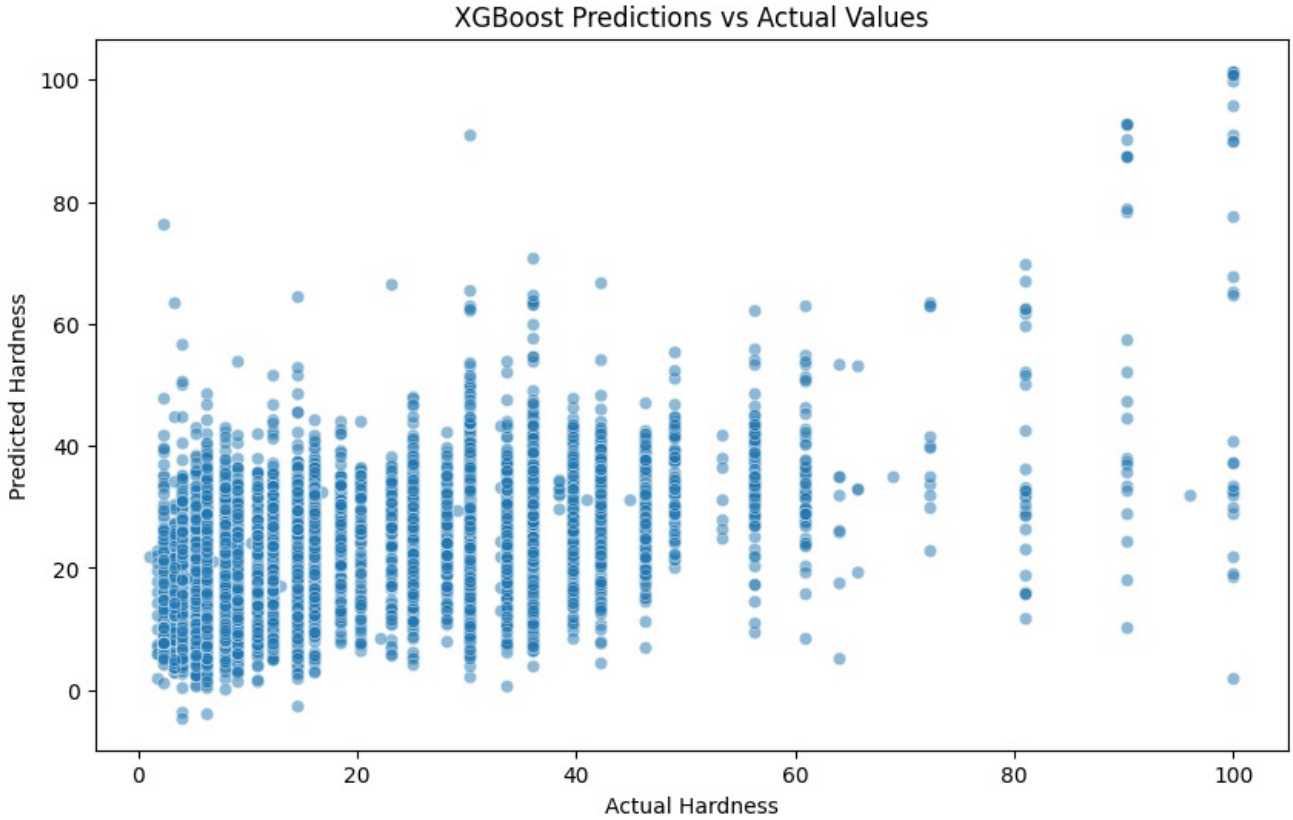


5.4 XGB Model

In [ ]:
```
xgb_model = XGBRegressor(random_state=42)
xgb_preds = cross_val_predict(xgb_model, X_train, y_train, cv=5)
xgb_mse = mean_squared_error(y_train, xgb_preds)
print(f'XGB Mean Squared Error: {xgb_mse}')
```

```
XGB Mean Squared Error: 170.5667454845043
```

In [ ]:
```
# Scatter plot for XGBoost predictions vs actual values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_train, y=xgb_preds, alpha=0.5)
plt.title('XGBoost Predictions vs Actual Values')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')
plt.show()
```

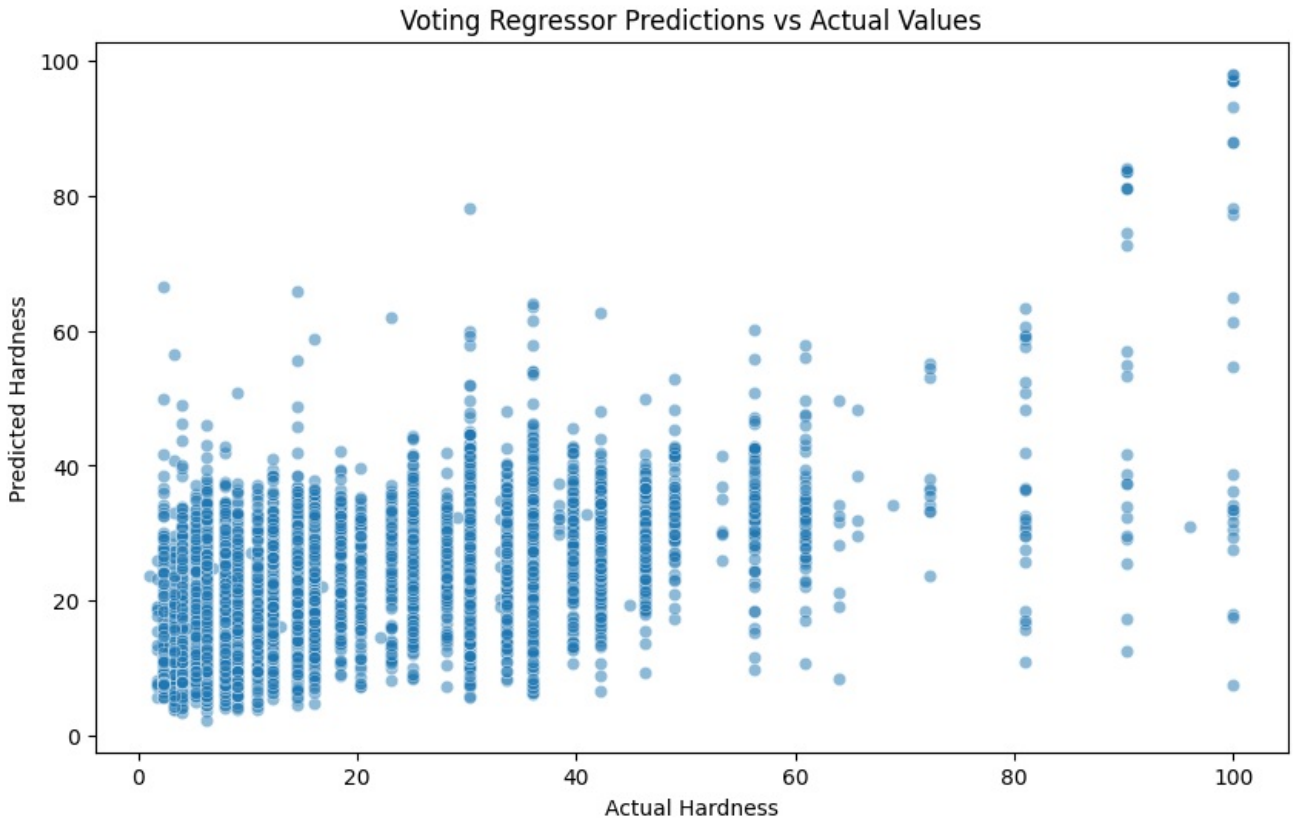


5.5 Voting Regressor

In [ ]:
```
voting_model = VotingRegressor([('rf', rf_model), ('catboost', catboost_model), ('lgbm', lgbm_model), ('xgb', xg
voting_preds = cross_val_predict(voting_model, X_train, y_train, cv=5)
voting_mse = mean_squared_error(y_train, voting_preds)
print(f'Voting Regressor Mean Squared Error: {voting_mse}')
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000246 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 711
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.663579
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000220 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.220146
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000276 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 708
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.307672
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000617 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.638069
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000269 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 704
[LightGBM] [Info] Number of data points in the train set: 6660, number of used features: 3
[LightGBM] [Info] Start training from score 24.541583
Voting Regressor Mean Squared Error: 159.49297067751152
```

In [ ]:
```
# Scatter plot for Voting Regressor predictions vs actual values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_train, y=voting_preds, alpha=0.5)
plt.title('Voting Regressor Predictions vs Actual Values')
plt.xlabel('Actual Hardness')
plt.ylabel('Predicted Hardness')
plt.show()
```



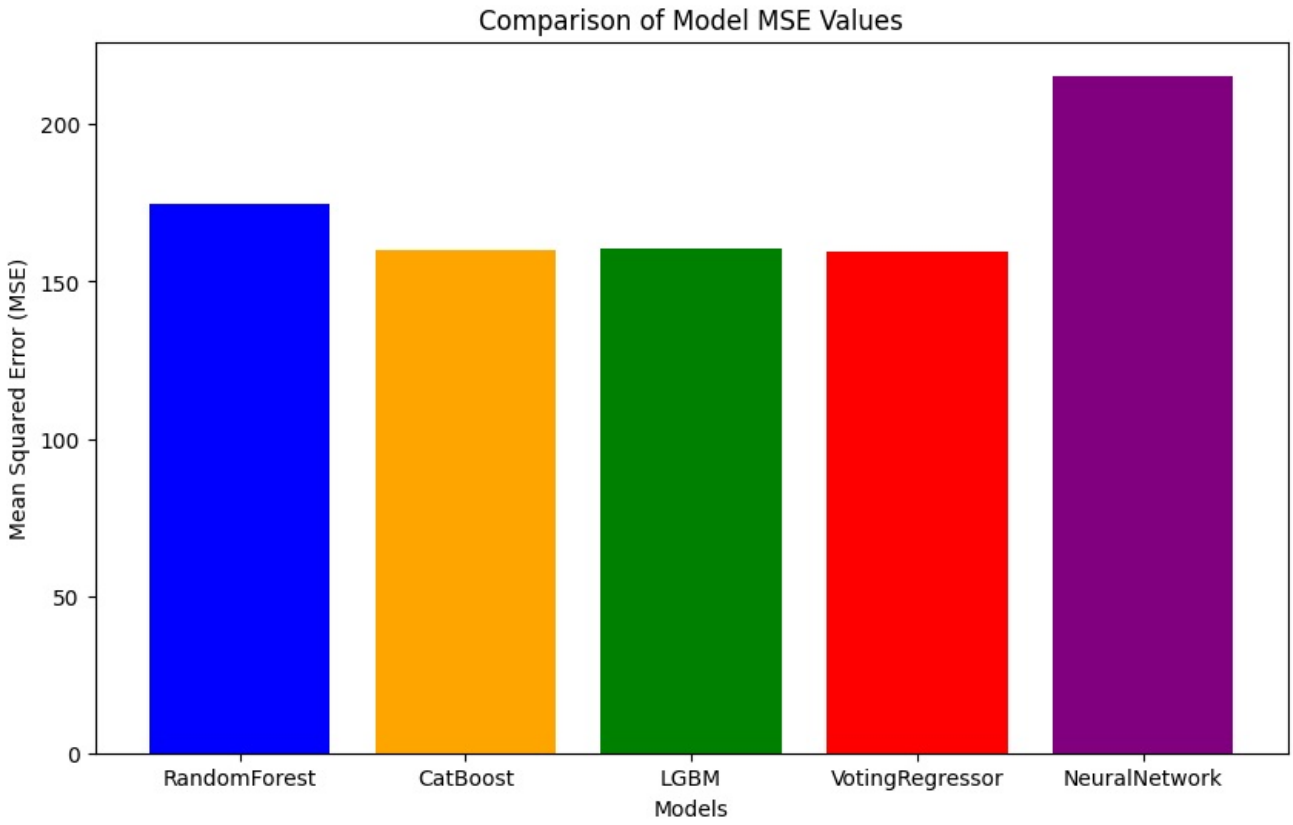
5.6 Neural Network based on VR Preds

In [ ]:
```
# Assuming X_train is your original feature set
nn_model = MLPRegressor(random_state=42, max_iter=1000)
nn_preds = cross_val_predict(nn_model, X_train, y_train, cv=5)
nn_mse = mean_squared_error(y_train, nn_preds)
print(f'Neural Network Mean Squared Error: {nn_mse}')
```

```
Neural Network Mean Squared Error: 215.46265007000378
```

In [ ]:
```
# Define models and their MSE values
models = ['RandomForest', 'CatBoost', 'LGBM', 'VotingRegressor', 'NeuralNetwork']
mse_values = [rf_mse, catboost_mse, lgbm_mse, voting_mse, nn_mse]
```

In [ ]:
```
# Create a bar chart
plt.figure(figsize=(10, 6))
plt.bar(models, mse_values, color=['blue', 'orange', 'green', 'red', 'purple'])
plt.xlabel('Models')
plt.ylabel('Mean Squared Error (MSE)')
plt.title('Comparison of Model MSE Values')
plt.show()
```



6. Artificial Neural Network

In [ ]:
```
# Assuming you have a DataFrame named 'train' with features and target
# Adjust the column names and target variable as needed
features = train.drop(columns=['Hardness'])
target = train['Hardness']
```

In [ ]:
```
# Define the model
model = Sequential()
```

In [ ]:
```
# Add layers to the model
model.add(Dense(units=64, activation='relu', input_dim=features.shape[1]))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=1, activation='linear'))
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

In [ ]:
```
# Display the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                832

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                832

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 2945 (11.50 KB)
Trainable params: 2945 (11.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

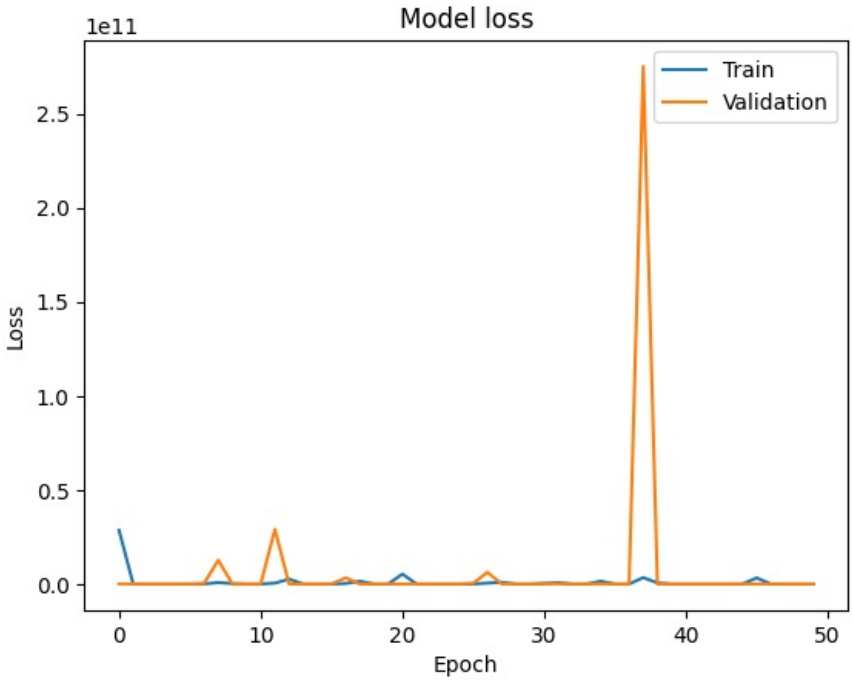
In [ ]:
```python
# Train the model
history = model.fit(
    features, target,
    epochs=50,  # You can adjust the number of epochs
    batch_size=32,
    validation_split=0.2  # You can adjust the validation split
)
```

```
Epoch 1/50
261/261 [==============================] - 3s 4ms/step - loss: 28574820352.0000 - mae: 41948.3555 - val_loss: 10
5341904.0000 - val_mae: 10180.7949
Epoch 2/50
261/261 [==============================] - 1s 3ms/step - loss: 356053.8750 - mae: 179.9982 - val_loss: 1061.0830
- val_mae: 28.4924
Epoch 3/50
261/261 [==============================] - 1s 2ms/step - loss: 328.1065 - mae: 14.4948 - val_loss: 1215.6849 - v
al_mae: 31.7661
Epoch 4/50
261/261 [==============================] - 1s 3ms/step - loss: 402.0723 - mae: 15.8554 - val_loss: 1778.3379 - v
al_mae: 39.3887
Epoch 5/50
261/261 [==============================] - 1s 2ms/step - loss: 504.9778 - mae: 17.0441 - val_loss: 972.6118 - va
l_mae: 27.8770
Epoch 6/50
261/261 [==============================] - 1s 3ms/step - loss: 407.7424 - mae: 15.8446 - val_loss: 3061.4956 - v
al_mae: 52.8480
Epoch 7/50
261/261 [==============================] - 1s 3ms/step - loss: 49530780.0000 - mae: 1471.5763 - val_loss: 258360
800.0000 - val_mae: 15943.8604
Epoch 8/50
261/261 [==============================] - 1s 2ms/step - loss: 802155264.0000 - mae: 9011.1084 - val_loss: 12669
692928.0000 - val_mae: 111650.6562
Epoch 9/50
261/261 [==============================] - 1s 3ms/step - loss: 246810176.0000 - mae: 6867.8901 - val_loss: 6980.
9336 - val_mae: 81.3372
Epoch 10/50
261/261 [==============================] - 1s 2ms/step - loss: 838.1118 - mae: 20.6745 - val_loss: 32504.2773 -
val_mae: 178.0654
Epoch 11/50
261/261 [==============================] - 1s 2ms/step - loss: 71767.9531 - mae: 129.6533 - val_loss: 15184.7051
- val_mae: 121.1589
Epoch 12/50
261/261 [==============================] - 1s 3ms/step - loss: 479338752.0000 - mae: 4152.7354 - val_loss: 29079
259136.0000 - val_mae: 169148.9688
Epoch 13/50
261/261 [==============================] - 1s 3ms/step - loss: 2607742720.0000 - mae: 21574.3770 - val_loss: 136
01386.0000 - val_mae: 3658.2847
Epoch 14/50
261/261 [==============================] - 1s 3ms/step - loss: 1012159.6250 - mae: 404.0286 - val_loss: 9655.636
7 - val_mae: 96.1232
Epoch 15/50
261/261 [==============================] - 1s 3ms/step - loss: 3389.6597 - mae: 37.7983 - val_loss: 8239.7324 -
val_mae: 88.6036
Epoch 16/50
261/261 [==============================] - 1s 3ms/step - loss: 1384730.3750 - mae: 409.6948 - val_loss: 5691289.
0000 - val_mae: 2366.4263
Epoch 17/50
261/261 [==============================] - 1s 3ms/step - loss: 311456896.0000 - mae: 9213.5537 - val_loss: 33012
```

```
16512.0000 - val_mae: 56992.0156
Epoch 18/50
261/261 [==============================] - 1s 3ms/step - loss: 1493726208.0000 - mae: 14453.1445 - val_loss: 354
2697.5000 - val_mae: 1867.0504
Epoch 19/50
261/261 [==============================] - 1s 3ms/step - loss: 25446.8828 - mae: 61.1508 - val_loss: 20568.8828
- val_mae: 141.3112
Epoch 20/50
261/261 [==============================] - 1s 3ms/step - loss: 24160882.0000 - mae: 1582.8875 - val_loss: 873044
7.0000 - val_mae: 2930.9287
Epoch 21/50
261/261 [==============================] - 1s 3ms/step - loss: 5298008576.0000 - mae: 29608.8730 - val_loss: 150
818.6719 - val_mae: 385.0266
Epoch 22/50
261/261 [==============================] - 1s 3ms/step - loss: 2774.0776 - mae: 30.4392 - val_loss: 1378.6515 -
val_mae: 33.5491
Epoch 23/50
261/261 [==============================] - 1s 3ms/step - loss: 622.5258 - mae: 18.9125 - val_loss: 340.9146 - va
l_mae: 14.4352
Epoch 24/50
261/261 [==============================] - 1s 3ms/step - loss: 482.0749 - mae: 17.0217 - val_loss: 25956.4121 -
val_mae: 158.9438
Epoch 25/50
261/261 [==============================] - 1s 3ms/step - loss: 17027.1504 - mae: 66.4865 - val_loss: 8518626.000
0 - val_mae: 2894.9363
Epoch 26/50
261/261 [==============================] - 1s 3ms/step - loss: 954613.0000 - mae: 407.9008 - val_loss: 359851040
.0000 - val_mae: 18816.3965
Epoch 27/50
261/261 [==============================] - 1s 3ms/step - loss: 490556704.0000 - mae: 10084.5537 - val_loss: 6209
686528.0000 - val_mae: 78165.2188
Epoch 28/50
261/261 [==============================] - 1s 3ms/step - loss: 932598528.0000 - mae: 10737.2949 - val_loss: 4113
4120.0000 - val_mae: 6361.7427
Epoch 29/50
261/261 [==============================] - 1s 3ms/step - loss: 3261847.2500 - mae: 830.2858 - val_loss: 107138.1
016 - val_mae: 324.3263
Epoch 30/50
261/261 [==============================] - 1s 3ms/step - loss: 10899.3525 - mae: 56.2966 - val_loss: 1074.2535 -
val_mae: 29.5173
Epoch 31/50
261/261 [==============================] - 1s 3ms/step - loss: 317966752.0000 - mae: 5108.0029 - val_loss: 13136
3400.0000 - val_mae: 11368.8164
Epoch 32/50
261/261 [==============================] - 1s 3ms/step - loss: 699770176.0000 - mae: 14668.0146 - val_loss: 1585
88192.0000 - val_mae: 12491.4287
Epoch 33/50
261/261 [==============================] - 1s 3ms/step - loss: 4710064.0000 - mae: 1087.8447 - val_loss: 17933.9
277 - val_mae: 131.8887
Epoch 34/50
261/261 [==============================] - 1s 4ms/step - loss: 7312274.0000 - mae: 845.3128 - val_loss: 65376296
.0000 - val_mae: 8020.2642
Epoch 35/50
261/261 [==============================] - 1s 3ms/step - loss: 1504430976.0000 - mae: 16567.2188 - val_loss: 486
85848.0000 - val_mae: 6921.1357
Epoch 36/50
261/261 [==============================] - 1s 3ms/step - loss: 2669877.7500 - mae: 755.4864 - val_loss: 2702.190
7 - val_mae: 49.4695
Epoch 37/50
261/261 [==============================] - 1s 3ms/step - loss: 9067.4014 - mae: 53.7884 - val_loss: 1339.7797 -
val_mae: 33.6020
Epoch 38/50
261/261 [==============================] - 1s 3ms/step - loss: 3424433920.0000 - mae: 13714.1719 - val_loss: 275
167051776.0000 - val_mae: 520326.7500
Epoch 39/50
261/261 [==============================] - 1s 3ms/step - loss: 597291904.0000 - mae: 7777.2827 - val_loss: 18434
6.2344 - val_mae: 425.6353
Epoch 40/50
261/261 [==============================] - 1s 3ms/step - loss: 42470.9805 - mae: 81.2814 - val_loss: 1230.9634 -
val_mae: 31.9602
Epoch 41/50
261/261 [==============================] - 1s 3ms/step - loss: 375.2551 - mae: 15.2474 - val_loss: 580.9504 - va
l_mae: 20.1312
Epoch 42/50
261/261 [==============================] - 1s 3ms/step - loss: 458.9619 - mae: 16.6820 - val_loss: 1483.6414 - v
al_mae: 34.9154
Epoch 43/50
261/261 [==============================] - 1s 3ms/step - loss: 1022.9634 - mae: 21.2240 - val_loss: 6894.6685 -
val_mae: 80.9050
Epoch 44/50
261/261 [==============================] - 1s 3ms/step - loss: 62350.0352 - mae: 138.2564 - val_loss: 1185687.62
50 - val_mae: 1080.0190
Epoch 45/50
```

```
261/261 [==============================] - 1s 3ms/step - loss: 28321580.0000 - mae: 2743.8364 - val_loss: 259267
06.0000 - val_mae: 5050.6431
Epoch 46/50
261/261 [==============================] - 1s 3ms/step - loss: 3246844160.0000 - mae: 21711.2031 - val_loss: 245
954.0312 - val_mae: 491.7550
Epoch 47/50
261/261 [==============================] - 1s 3ms/step - loss: 1746.0422 - mae: 23.3808 - val_loss: 853.4942 - v
al_mae: 25.6873
Epoch 48/50
261/261 [==============================] - 1s 3ms/step - loss: 477.9806 - mae: 16.8923 - val_loss: 250.0475 - va
l_mae: 12.9778
Epoch 49/50
261/261 [==============================] - 1s 3ms/step - loss: 596.9636 - mae: 18.5321 - val_loss: 4194.9043 - v
al_mae: 62.3607
Epoch 50/50
261/261 [==============================] - 1s 3ms/step - loss: 1256.5946 - mae: 23.9821 - val_loss: 14718.3906 -
val_mae: 119.3891
```

In [ ]:
```
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```



7. Creating 'submission.csv'

In [ ]:
```
SPLITS = 5  # Replace 5 with the actual value you intend to use
REPEATS = 3  # Replace 3 with the actual value you intend to use

# Assuming public_work is a DataFrame or a variable with the necessary data
public_work = pd.DataFrame({
    'id': [1, 2, 3],  # Replace with your actual data
    'Hardness': [10.0, 15.0, 20.0]  # Replace with your actual data
})
# Other code...

sample_submission["Hardness"] = sample_submission["Hardness"] / (SPLITS * REPEATS)
sample_submission["Hardness"] = 0.5 * public_work["Hardness"] + 0.5 * sample_submission["Hardness"]
sample_submission.to_csv("submission.csv", index=False, header=True)
```