Diamond PRICE PREDICITON

SHAI INTERNSHIP | GROUP 7

Meet
THE TEAM

Andrew

Mahmoud Yaseen

Abdullah Alriyabi

INTRODUCTION

This report details the steps taken to develop a robust model for predicting diamond prices based on various attributes such as carat, cut, color, clarity, and dimensions.
By preprocessing the data, selecting and optimizing regression models, and employing advanced techniques like model stacking, we aimed to achieve high accuracy in our predictions.

Importing Libraries and Modules

- Pandas, Matplotlib, Seaborn, and scikit-learn for data manipulation, visualization, and machine learning.
- Specific regression models: Linear Regression, RandomForestRegressor, GradientBoostingRegressor, etc.
- XGBoost library for XGBoostRegressor.
- Preprocessing modules: OneHotEncoder, OrdinalEncoder, ColumnTransformer, Pipeline, StandardScaler.
- Performance metrics: mean_squared_error, r2_score.
- GridSearchCV for hyperparameter tuning using cross-validation.
- Filtering warnings for enhanced code readability.

```python
# Importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import warnings
warnings.filterwarnings("ignore")
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor, VotingRegressor, StackingRegressor
from sklearn.linear_model import RidgeCV
import xgboost as xgb
import numpy as np
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.linear_model import Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV
```

Reading Data

Loads the training and test datasets from CSV files into Pandas DataFrames.

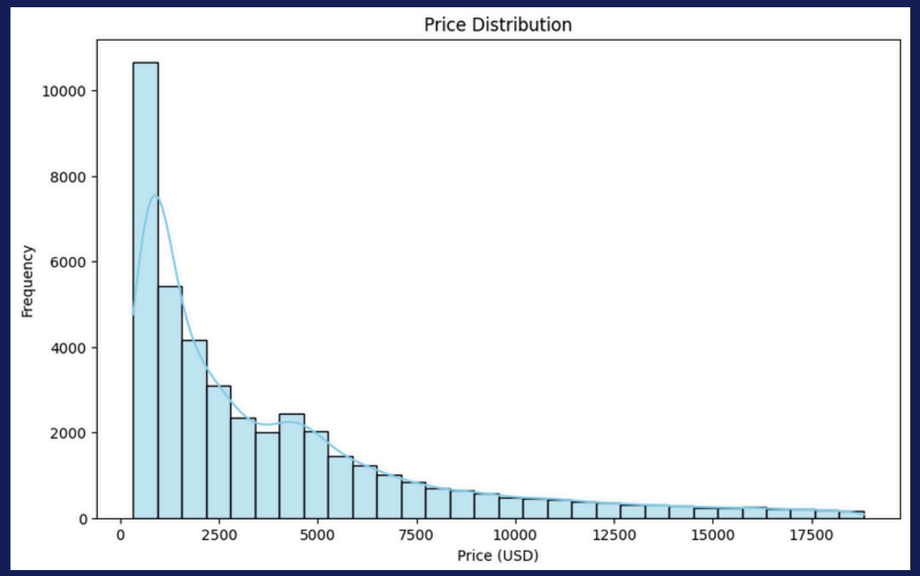
Exploratory Data Analysis (EDA)

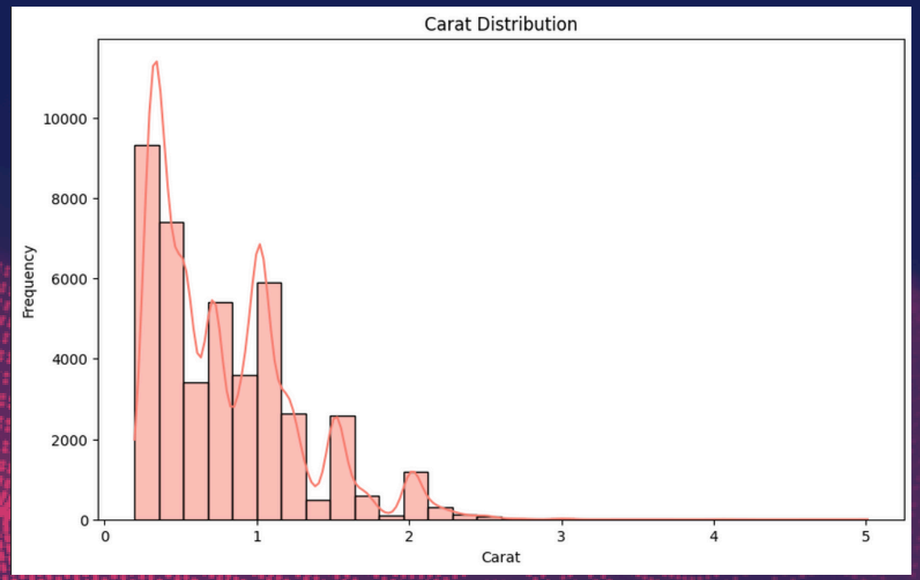
- Displays the first few rows of the training dataset to inspect its structure and contents.
- Checks for duplicated rows in the training dataset and computes the sum of duplicates.
- Checks for missing values (NaN) in the training dataset and computes the sum of missing values for each column.
- Provides information about the training dataset including the data types of columns and memory usage.

```
#EDA
df.head()
df.duplicated().sum() #zero
df.isna().sum() #zero
df.info()
```

Data Visualization

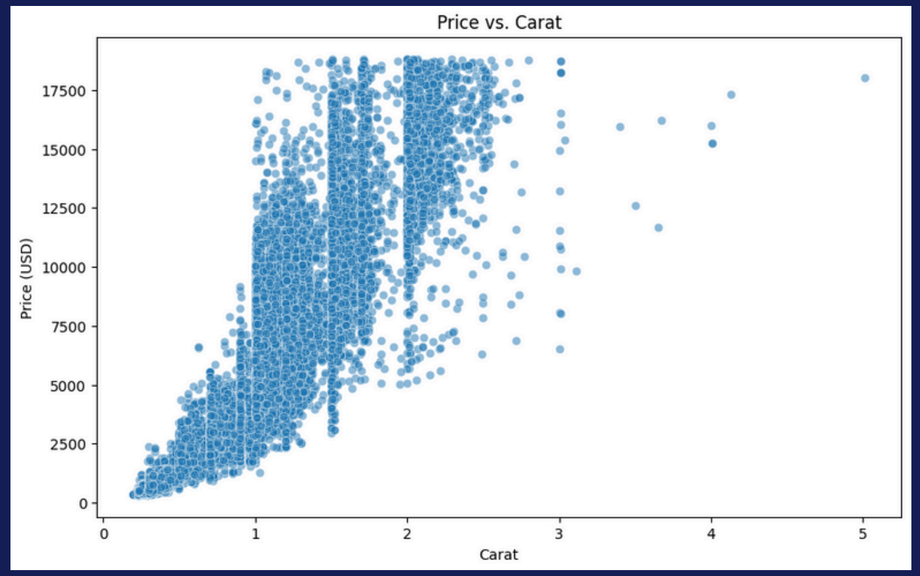
- Price Distribution: A histogram showing the distribution of diamond prices reveals a right-skewed distribution with potential outliers.
- Carat Distribution: A histogram displaying the distribution of diamond carats shows a similar right-skewed pattern with possible outliers.

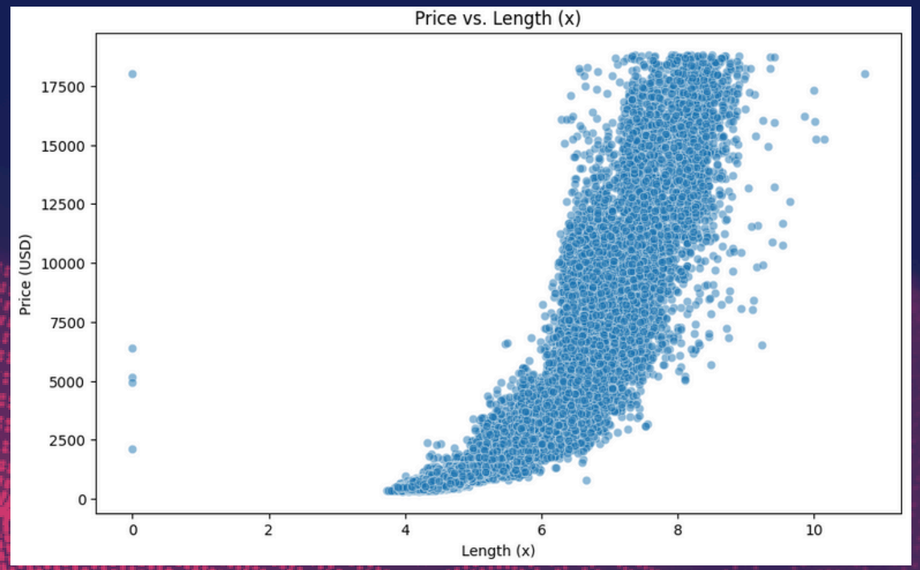




Data Visualization

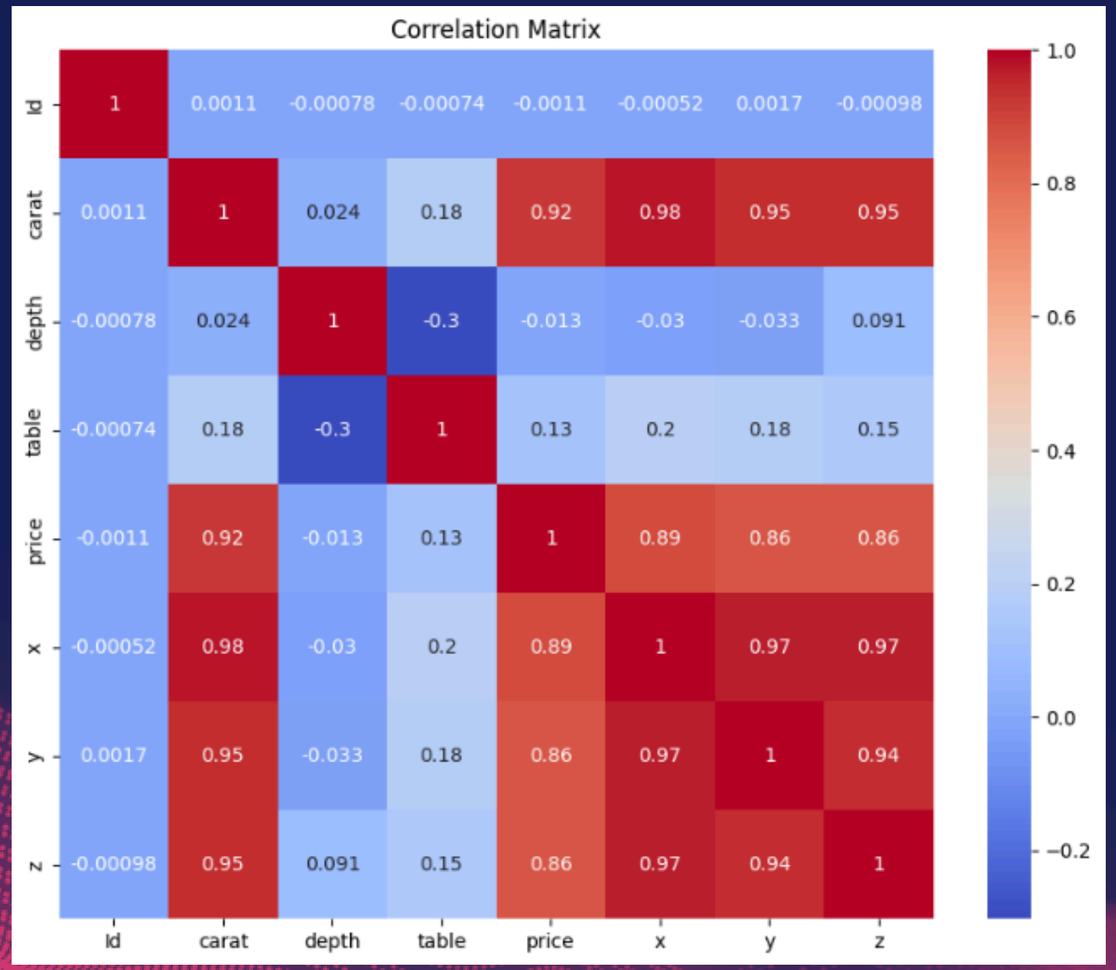
- Price vs. Carat: A scatter plot illustrating the relationship between diamond price and carat, highlighting how price tends to increase with carat size, shows a clear positive correlation.
- A scatter plot depicting the relationship between diamond price and its length (x dimension), indicating how length might influence price, shows a clear positive correlation.





Data Visualization

A heatmap displaying the correlation matrix between numerical features in the dataset, showing the strength and direction of relationships between variables like carat, depth, table, and dimensions (x, y, z).



Linear Regression Model Training and Evaluation

- Splits the dataset into features (X) and target variable (y) for training.
- Divides the data into training and testing sets using train_test_split with a test size of 20% and a random state of 42.
- Instantiates and trains a Linear Regression model on the training data.
- Makes predictions on the test data.
- Computes the root mean squared error (RMSE) and R-squared (R2) score to evaluate the model's performance on the test set.
- Prints the RMSE and R2 score for assessment

```python
# Select features and target variable
X = df[['carat', 'depth', 'table', 'x', 'y', 'z']]  # Features
y = df['price']  # Target variable

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initialize the Linear Regression model
model = LinearRegression()

# Fit the model to the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, y_pred, squared=False)

# Calculate the R^2 Score
r2 = r2_score(y_test, y_pred)

# Print the evaluation metrics
print(f"Root Mean Squared Error: {rmse}")
print(f"R^2 Score: {r2}")
```

```
Root Mean Squared Error: 1449.813312868884
R^2 Score: 0.8642238789032652
```

Model Training and Evaluation with Encoding Techniques

- Splits the dataset into features (X) and target variable (y) for training.
- Divides the data into training and testing sets using train_test_split with a test size of 20% and a random state of 42.
- Instantiates and trains a Linear Regression model on the training data.
- Makes predictions on the test data.
- Computes the root mean squared error (RMSE) and R-squared (R2) score to evaluate the model's performance on the test set.
- Prints the RMSE and R2 score for assessment

```python
# Define column transformer for encoding categorical columns
# One-Hot Encoding for 'cut', 'color', and 'clarity'
# Ordinal Encoding for 'cut', 'color', and 'clarity'
one_hot_transformer = ColumnTransformer(
    transformers=[
        ('one_hot', OneHotEncoder(drop='first'), ['cut', 'color', 'clarity'])
    ],
    remainder='passthrough'
)

ordinal_transformer = ColumnTransformer(
    transformers=[
        ('ordinal', OrdinalEncoder(), ['cut', 'color', 'clarity'])
    ],
    remainder='passthrough'
)

# Linear regression pipeline with One-Hot Encoding
one_hot_pipeline = Pipeline([
    ('encoder', one_hot_transformer),
    ('regressor', LinearRegression())
])

# Linear regression pipeline with Ordinal Encoding
ordinal_pipeline = Pipeline([
    ('encoder', ordinal_transformer),
    ('regressor', LinearRegression())
])
```

```
Metrics with One-Hot Encoding:
Mean Squared Error: 1081.018079959619
R^2 Score: 0.9245142201937553

Metrics with Ordinal Encoding:
Root Mean Squared Error: 1305.2489562044825
R^2 Score: 0.8899510406968214
```

Cross-Validation for Model Evaluation

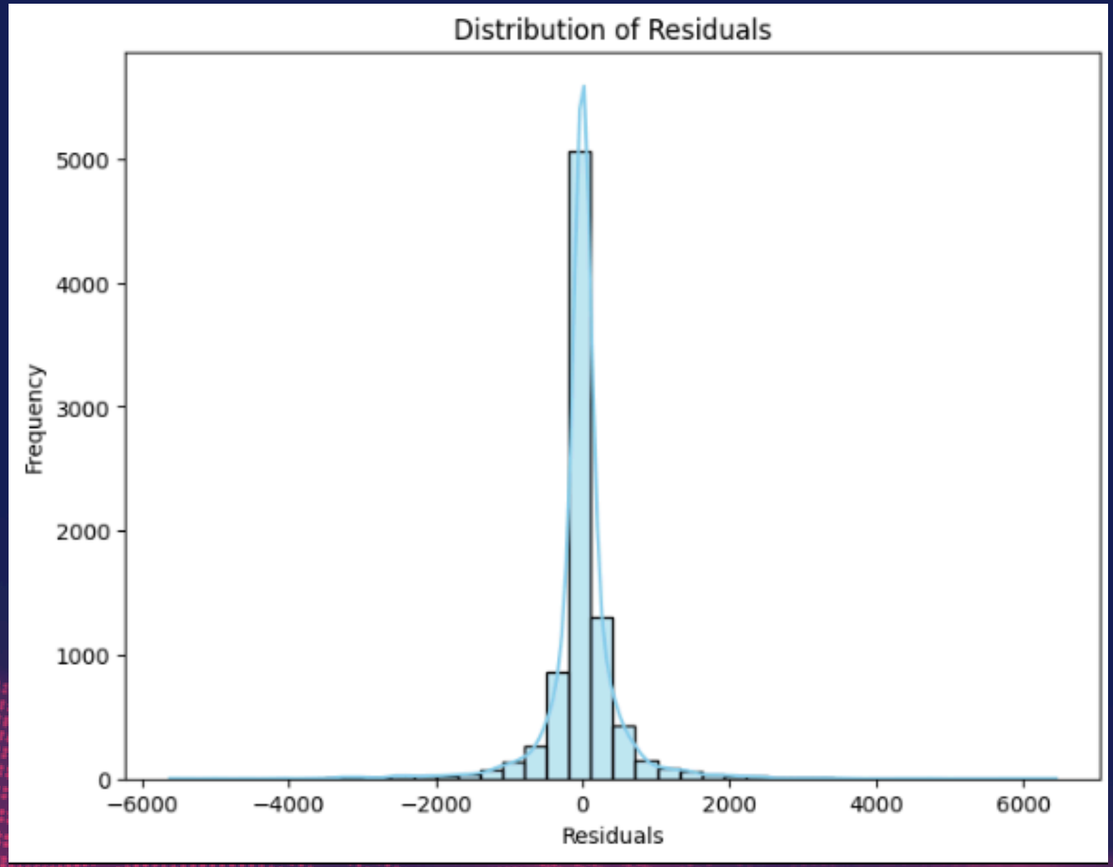
- Defines features (X) and target variable (y) from the dataset.
- Defines a ColumnTransformer for one-hot encoding categorical columns ('cut', 'color', 'clarity') and standard scaling numerical columns ('carat', 'depth', 'table', 'x', 'y', 'z').
- Constructs a pipeline with preprocessing steps including one-hot encoding and standard scaling followed by linear regression.
- Performs cross-validation with 5 folds using the pipeline and computes the negative root mean squared error (RMSE) scores.
- Prints the cross-validation RMSE scores and the mean RMSE score for model evaluation.

```python
# Define features (X) and target variable (y)
X = df[['carat', 'cut', 'color', 'clarity', 'depth', 'table', 'x', 'y', 'z']]
y = df['price']

# Define column transformer for encoding categorical columns and scaling numerical columns
# One-Hot Encoding for 'cut', 'color', and 'clarity'
# Standard Scaling for numerical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('one_hot', OneHotEncoder(drop='first'), ['cut', 'color', 'clarity']),
        ('scaler', StandardScaler(), ['carat', 'depth', 'table', 'x', 'y', 'z'])
    ],
    remainder='passthrough'
)

# Linear regression pipeline with One-Hot Encoding and Standard Scaling
pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Perform cross-validation
cv_scores = cross_val_score(pipeline, X, y, cv=5, scoring='neg_root_mean_squared_error')

# Print cross-validation scores
print("Cross-Validation rmse Scores:", -cv_scores)
print("Mean rmse Score:", -cv_scores.mean())
```

```
Mean rmse Score: 1166.173866673296
```

Residual Analysis

- Aligns the indices of y_test and y_pred_test to ensure correct mapping.
- Calculates residuals by subtracting predicted values from actual values.
- Visualizes the distribution of residuals using a histogram plot with 40 bins and kernel density estimation (KDE) for smoother visualization.
- The plot helps in understanding the distribution and pattern of errors (residuals) made by the model.
- A bell curve in the plot indicates a good model.



Model Evaluation with Various Regression Models

- Defines preprocessing steps including standard scaling for numerical features and one-hot encoding for categorical features.
- Defines a dictionary of regression models to evaluate, including Linear Regression, Ridge Regression, Lasso Regression, ElasticNet Regression, Decision Tree Regression, Random Forest Regression, Gradient Boosting Regression, and k-Nearest Neighbors Regression.
- Constructs pipelines for each model with preprocessing and the respective regression model.
- Performs cross-validation with 5 folds for each model and computes the negative root mean squared error (RMSE) scores.
- Prints the mean RMSE score for each model to evaluate their performance.

```python
# Define preprocessing steps
# One-Hot Encoding for categorical features, Standard Scaling for numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['carat', 'depth', 'table', 'x', 'y', 'z']),
        ('cat', OneHotEncoder(drop='first'), ['cut', 'color', 'clarity'])
    ],
    remainder='passthrough'
)

# Define regression models to evaluate
models = {
    'Linear Regression': LinearRegression(),
    'Ridge Regression': Ridge(),
    'Lasso Regression': Lasso(),
    'ElasticNet Regression': ElasticNet(),
    'Decision Tree Regression': DecisionTreeRegressor(),
    'Random Forest Regression': RandomForestRegressor(),
    'Gradient Boosting Regression': GradientBoostingRegressor(),
    'k-Nearest Neighbors Regression': KNeighborsRegressor()
}

# Perform cross-validation for each model
for name, model in models.items():
    # Create a pipeline with preprocessing and the regression model
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])

    # Perform cross-validation and calculate mean R-squared score
    cv_scores = cross_val_score(pipeline, X, y, cv=5,
scoring='neg_root_mean_squared_error')
    print(f"{name}: Mean rmse Score: {-cv_scores.mean()}")
```

```
Linear Regression: Mean rmse Score: 1166.1738666732965
Ridge Regression: Mean rmse Score: 1164.1623428595144
Lasso Regression: Mean rmse Score: 1133.3546587132353
ElasticNet Regression: Mean rmse Score: 1733.6167735134873
Decision Tree Regression: Mean rmse Score: 853.0849311264152
Random Forest Regression: Mean rmse Score: 641.569144591658
Gradient Boosting Regression: Mean rmse Score: 865.7676767006933
k-Nearest Neighbors Regression: Mean rmse Score: 864.3019570050171
```

Feature Engineering and Model Evaluation

- Calculates a new feature 'volume' by multiplying the dimensions 'x', 'y', and 'z'.
- Selects numerical features 'carat' and 'volume' along with categorical features 'cut', 'color', and 'clarity' as predictors.
- Splits the data into training and testing sets.
- Defines preprocessing steps including standard scaling for numerical features and one-hot encoding for categorical features.
- Defines evaluation metrics to compute: Mean Squared Error, Root Mean Absolute Error, and R-squared Score.
- Performs cross-validation for each top model with 5 folds and calculates evaluation metrics using cross_val_predict.
- Prints the evaluation metrics for each model to assess their performance after feature engineering.

```python
# Feature engineering: calculate volume
df['volume'] = df['x'] * df['y'] * df['z']

# Select features and target variable
numerical_features = ['carat', 'volume']  # Use volume instead of x, y, z
categorical_features = ['cut', 'color', 'clarity']
target_variable = 'price'

# Prepare the data
X = df[numerical_features + categorical_features]
y = df[target_variable]

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), ['carat', 'volume']),
        ('cat', OneHotEncoder(drop='first'), ['cut', 'color', 'clarity'])
    ],
    remainder='passthrough'
)

# Define evaluation metrics to compute
evaluation_metrics = ['Mean Squared Error', 'Root Mean Absolute Error', 'R^2 Score']

# Perform cross-validation and calculate evaluation metrics for each top model
for name, model in top_models.items():
    # Create a pipeline with preprocessing and the regression model
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('model', model)])

    # Perform cross-validation
    cv_predictions = cross_val_predict(pipeline, X, y, cv=5)
```

```
Random Forest Regression:
        Mean Squared Error: 414143.2452265944
        Root Mean Absolute Error: 643.5396221108646
        R^2 Score: 0.9739271226176316
```

Feature Selection and Model Training

- Defines preprocessing steps for numerical features (no scaling necessary for tree-based models) and categorical features using OneHotEncoder.
- Bundles preprocessing for both numerical and categorical features using ColumnTransformer.
- Defines models including Linear Regression and Random Forest.
- Appends each model to the preprocessing pipeline and trains it on the training data.
- Predicts on the test set and calculates the root mean squared error (RMSE) for each model to evaluate their performance.

```
# Preprocessing for numerical features (scaling not necessary for tree-based models)
numerical_transformer = 'passthrough'

# Preprocessing for categorical features
categorical_transformer = OneHotEncoder(handle_unknown='ignore')

# Bundle preprocessing for numerical and categorical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Define models
model = RandomForestRegressor(random_state=42)


# Append model to preprocessing pipeline
model_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                 ('model', model)])
# Train the model
model_pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = model_pipeline.predict(X_test)

# Calculate RMSE
rmse = mean_squared_error(y_test, y_pred, squared=False)
print(f"RMSE for {name}: {rmse}")
```

```
RMSE for Random Forest: 562.777415609872
```

Stacking Regressor Model with Custom Hyperparameters

- Sets up preprocessing steps for numerical features (no transformation) and categorical features using OneHotEncoder.
- Defines base models including RandomForestRegressor, GradientBoostingRegressor, and XGBRegressor with custom hyperparameters.
- Initializes the StackingRegressor with the defined base models and a RidgeCV as the final estimator with 5-fold cross-validation.
- Creates a pipeline including preprocessing and the stacking regressor model.
- Fits the model on the training data.
- Makes predictions on the testing data.
- Calculates the root mean squared error (RMSE) and R-squared score of the model to evaluate its performance.

```
# Define the base models
base_models = [
    ('rf', RandomForestRegressor(n_estimators=300, min_samples_split=5, max_depth=200,
random_state=42)),
    ('gb', GradientBoostingRegressor(n_estimators=200, max_depth=5, learning_rate=0.1,
random_state=42)),
    ('xgb', xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1,
random_state=42))
]

# Initialize the Stacking Regressor
model = StackingRegressor(estimators=base_models, final_estimator=RidgeCV(), cv=5)

# Create a pipeline
pipeline_stacking = Pipeline(steps=[('preprocessor', preprocessor),
                                    ('model', model)])

# Fit the model on the training data
pipeline_stacking.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = pipeline_stacking.predict(X_test)

# Calculate RMSE of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'The RMSE of the Stacking Regressor model is {rmse}')

# Calculate R-squared score of the model
r2 = r2_score(y_test, y_pred)

print(f'The R-squared score of the Stacking Regressor model is {r2}')
```

```
The RMSE of the Stacking Regressor model is 590.752833593846
```

Final Stacking Regressor Model

- Encodes categorical variables using one-hot encoding.
- Splits the data into training and testing sets.
- Defines base models including RandomForestRegressor, GradientBoostingRegressor, and XGBRegressor with optimized hyperparameters.
- Initializes the StackingRegressor with the defined base models and a RidgeCV as the final estimator with 5-fold cross-validation.
- Fits the model on the training data.
- Makes predictions on the testing data.
- Calculates the root mean squared error (RMSE) and R-squared score of the model to evaluate its performance.
- This model represents the final trained and evaluated model for predicting diamond prices.

```
df_encoded = pd.get_dummies(df)
X = df_encoded.drop(['Id', 'price'], axis=1)
y = df_encoded['price']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Define the base models with the best parameters
base_models = [
    ('rf', RandomForestRegressor(n_estimators=300, min_samples_split=5, max_depth=200,
random_state=42)),
    ('gb', GradientBoostingRegressor(n_estimators=200, max_depth=5, learning_rate=0.1,
subsample=0.75, random_state=42)),
    ('xgb', xgb.XGBRegressor(n_estimators=200, max_depth=5, learning_rate=0.1, gamma=0,
random_state=42))
]

# Initialize the Stacking Regressor
model = StackingRegressor(estimators=base_models, final_estimator=RidgeCV(), cv=5)

# Fit the model on the training data
model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = model.predict(X_test)

# Calculate RMSE of the model
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f'The RMSE of the Stacking Regressor model is {rmse}')

# Calculate R-squared score of the model
r2 = r2_score(y_test, y_pred)

print(f'The R-squared score of the Stacking Regressor model is {r2}')
```

```
The RMSE of the Stacking Regressor model is 516.4786
```

SUMMARY

This project involved predicting diamond prices using various regression models. Key steps included data preparation, feature engineering, and model evaluation. A stacking ensemble model combining Random Forest, Gradient Boosting, and XGBoost provided the best performance. Residual analysis confirmed the model's accuracy, and final predictions were prepared for submission.