

Project Report-3: Predicting House Prices Based on Features

Data Collection

Successfully collected Data set from Kaggle.(Source: <https://www.kaggle.com>).

Data Loading

Step 1: Load the Ames Housing dataset.

Objective: Import the Ames Housing dataset into a pandas DataFrame and review its structure and key attributes.

Data Exploration

Step 2: Explore the loaded dataset.

Objective: Analyze the dataset by examining its shape, data types, distributions of key variables, correlations, missing values, and unique values of categorical features.

Data Cleaning

Step 3: Clean the dataset by handling missing values and outliers.

Objective: Address missing values by filling 'Garage_Type' with 'No Garage', identify and manage outliers in 'Price' and 'Square_Footage' using winsorizing, and convert categorical features to numerical representations through one-hot encoding.

Feature Engineering

Step 4: Engineer new features from the cleaned dataset to improve model performance.

Objective: Generate interaction features, transform skewed distributions, encode categorical variables, and optionally scale numerical features to enhance model accuracy and effectiveness.

Data Splitting

Step 5: Divide the dataset into training, validation, and testing sets.

Objective: Use `train_test_split` to allocate data appropriately, ensuring a balanced distribution for effective model training and evaluation.

Model Training

Step 6: Train a Decision Tree Regression model on the training data.

Objective: Fit a Decision Tree Regression model using the training dataset to learn patterns and relationships in the data.

Model Optimization

Step 7: Optimize the hyperparameters of the Decision Tree Regression model.

Objective: Tune the model's hyperparameters using GridSearchCV to enhance performance and prevent overfitting.

Model Evaluation

Step 8: Assess the performance of the optimized Decision Tree Regression model.

Objective: Measure the model's accuracy using evaluation metrics such as R-squared, Mean Squared Error (MSE), and Root Mean Squared Error (RMSE) on the test dataset.

Data Visualization

Step 9: Visualize key insights and model performance.

Objective: Create visual representations to analyze feature importance, examine relationships between features and house prices, and assess the model's predictions through residual and distribution plots.

Code:

Data loading

```
import pandas as pd

try:
    df = pd.read_csv('ames_housing_dataset_updated.csv')
    display(df.head())
    print(df.shape)
except FileNotFoundError:
    print("Error: 'ames_housing_dataset_updated.csv' not found.")
    df = None
except pd.errors.ParserError:
    print("Error: Could not parse the CSV file.")
    df = None
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    df = None
```

Output

House_ID	Square_Footage	Bedrooms	Bathrooms	Location	Year_Built	House_Type	Garage_Type	Lot_Size	Price
1	3526	6	3	East	1968	Apartme nt	Attache d	0.29	279978
2	4938	4	1	South	1953	Condo	NaN	2.4	714731
3	4985	4	2	West	1955	Townho use	Attache d	2.3	263628
4	2633	3	2	East	1985	Condo	NaN	1.92	932521
5	1401	4	3	South	2012	Apartme nt	Attache d	1.34	142876

Data Exploration

```

# Examine data types
print(df.info())

# Determine the shape of the dataset
print("\nShape of the dataset:", df.shape)

# Identify missing values
print("\nMissing values per column:\n", df.isnull().sum())

# Summarize descriptive statistics of numerical features
print("\nDescriptive statistics of numerical features:\n", df.describe())

# Explore the distribution of the target variable ('Price')
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.hist(df['Price'], bins=30)
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.title('Distribution of House Prices')
plt.show()

# Analyze relationships between numerical features and the target variable
numerical_features = ['Square Footage', 'Bedrooms', 'Bathrooms', 'Year Built', 'Lot Size']
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_features):
    plt.subplot(2, 3, i + 1)
    plt.scatter(df[col], df['Price'])
    plt.xlabel(col)
    plt.ylabel('Price')
    plt.title(f'Price vs. {col}')
plt.tight_layout()
plt.show()

```

Output

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   House_ID        500 non-null   int64
1   Square Footage  500 non-null   int64
2   Bedrooms        500 non-null   int64
3   Bathrooms       500 non-null   int64
4   Location        500 non-null   object
5   Year_Built      500 non-null   int64
6   House_Type      500 non-null   object
7   Garage_Type     328 non-null   object
8   Lot Size        500 non-null   float64
9   Price           500 non-null   int64
dtypes: float64(1), int64(6), object(3)
memory usage: 39.2+ KB
None

Shape of the dataset: (500, 10)

Missing values per column:

```

```

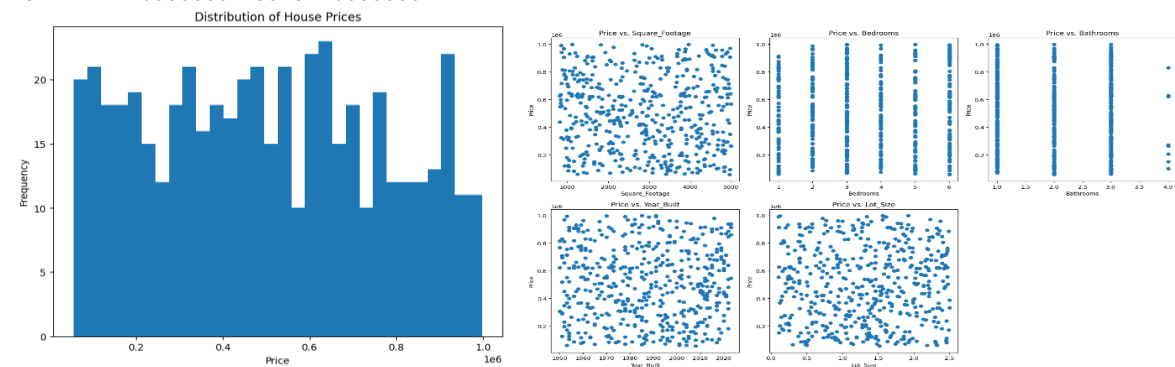
House ID      0
Square Footage 0
Bedrooms      0
Bathrooms     0
Location      0
Year Built    0
House Type    0
Garage Type   172
Lot Size      0
Price         0
dtype: int64

```

Descriptive statistics of numerical features:

	House ID	Square Footage	Bedrooms	Bathrooms	Year Built	\
count	500.000000	500.000000	500.000000	500.000000	500.000000	
mean	250.500000	2912.276000	3.536000	2.046000	1987.470000	
std	144.481833	1260.366011	1.703675	0.856359	21.330802	
min	1.000000	810.000000	1.000000	1.000000	1950.000000	
25%	125.750000	1778.750000	2.000000	1.000000	1970.000000	
50%	250.500000	2934.500000	3.000000	2.000000	1987.000000	
75%	375.250000	3967.750000	5.000000	3.000000	2006.250000	
max	500.000000	4986.000000	6.000000	4.000000	2023.000000	

	Lot Size	Price
count	500.000000	500.000000
mean	1.332400	500439.116000
std	0.682937	266173.619109
min	0.100000	56150.000000
25%	0.770000	278972.250000
50%	1.355000	489464.500000
75%	1.890000	713364.750000
max	2.500000	997922.000000

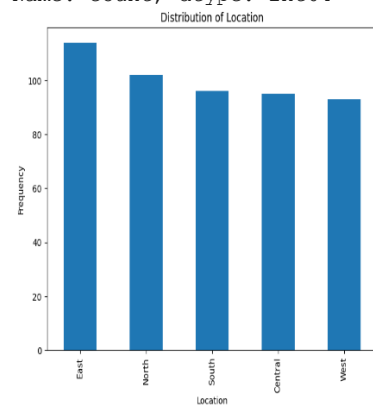


Unique values and frequencies for Location:

```

Location
East      114
North     102
South      96
Central    95
West       93
Name: count, dtype: int64

```



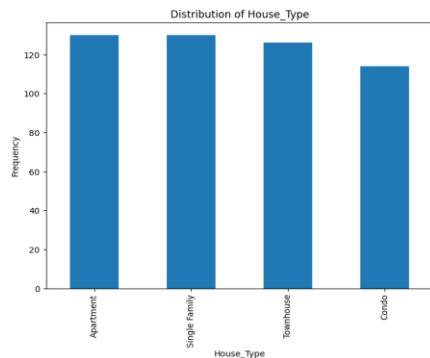
Unique values and frequencies for House_Type:

```

House_Type
Apartment      130

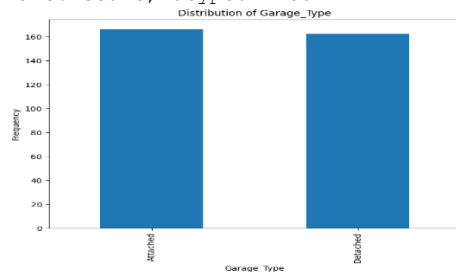
```

```
Single Family    130
Townhouse       126
Condo           114
Name: count, dtype: int64
```



Unique values and frequencies for Garage_Type:

```
Garage_Type
Attached    166
Detached    162
Name: count, dtype: int64
```



Data cleaning

```
# Impute missing values in 'Garage_Type'
garage_type_mode = df['Garage_Type'].mode()[0]
df['Garage_Type'] = df['Garage_Type'].fillna(garage_type_mode)

# Handle duplicate rows
print(f"Number of duplicated rows before handling: {df.duplicated().sum()}")
df = df.drop_duplicates()
print(f"Number of duplicated rows after handling: {df.duplicated().sum()}")

# Check and convert data types if necessary
# Check for numerical features incorrectly classified as objects
for col in df.columns:
    if df[col].dtype == 'object' and any(df[col].str.isdigit()):
        print(f"Column '{col}' is an object type but contains numbers. Converting to numeric...")
        try:
            df[col] = pd.to_numeric(df[col], errors='coerce')
        except Exception as e:
            print(f"Error converting column '{col}': {e}")
print(df.info())

# Identify and handle outliers in numerical features
import matplotlib.pyplot as plt
```

```

numerical_features = ['Square_Footage', 'Bedrooms', 'Bathrooms', 'Year_Built', 'Lot_Size',
'Price']
plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_features):
    plt.subplot(2, 3, i + 1)
    plt.boxplot(df[col])
    plt.xlabel(col)
    plt.title(f'Boxplot of {col}')
plt.tight layout()
plt.show()

```

Output

Number of duplicated rows before handling: 0

Number of duplicated rows after handling: 0

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 500 entries, 0 to 499

Data columns (total 10 columns):

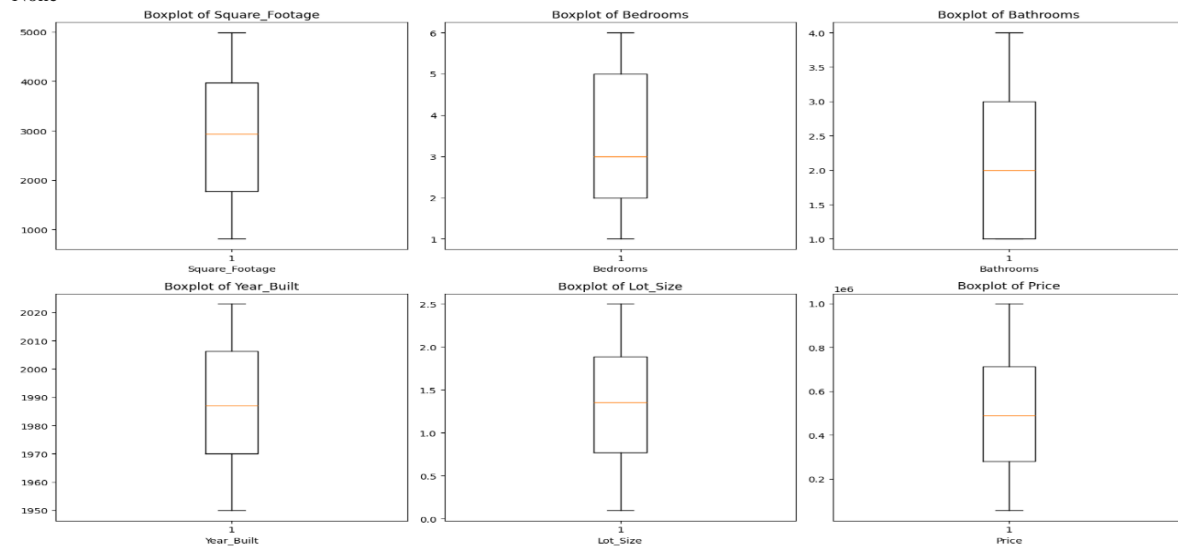
#	Column	Non-Null Count	Dtype
---	--------	----------------	-------

0	House_ID	500 non-null	int64
1	Square_Footage	500 non-null	int64
2	Bedrooms	500 non-null	int64
3	Bathrooms	500 non-null	int64
4	Location	500 non-null	object
5	Year_Built	500 non-null	int64
6	House_Type	500 non-null	object
7	Garage_Type	500 non-null	object
8	Lot_Size	500 non-null	float64
9	Price	500 non-null	int64

dtypes: float64(1), int64(6), object(3)

memory usage: 39.2+ KB

None



Handling Outliers

```
import numpy as np

def winsorize_outliers(data, lower_percentile=1, upper_percentile=99):
    lower_bound = np.percentile(data, lower_percentile)
    upper_bound = np.percentile(data, upper_percentile)
    data = np.clip(data, lower_bound, upper_bound)
    return data

for col in ['Square Footage', 'Lot Size', 'Price']:
    df[col] = winsorize_outliers(df[col])
```

Feature engineering

```
import pandas as pd
import numpy as np

# 1. Create Interaction Features
df['SquareFootage Bedrooms'] = df['Square Footage'] * df['Bedrooms']

# 2. Transform Skewed Features
for col in ['Lot_Size', 'Price']:
    df[col] = np.log1p(df[col])

# 3. Encode Categorical Features
categorical_cols = ['Location', 'House Type', 'Garage Type']
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)

# 4. Feature Scaling (Optional):
# Numerical features to scale
numerical_cols = ['Square Footage', 'Bedrooms', 'Bathrooms', 'Year Built', 'Lot Size',
                  'SquareFootage Bedrooms']
# from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler()
# df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

display(df.head())
```

Output

House_I	Square_	Bedroo	Bathroo	Year_Bu	Lot_Size	Price	SquareF	Location	Location
1	3526	6	3	1968	0.25464	12.5425	21156	TRUE	FALSE
2	4938	4	1	1953	1.22378	13.4797	19752	FALSE	FALSE
3	4966.02	4	2	1955	1.19392	12.4823	19864.1	FALSE	FALSE
4	2633	3	2	1985	1.07158	13.7456	7899	TRUE	FALSE
5	1401	4	3	2012	0.85015	11.8697	5604	FALSE	FALSE

Data splitting

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df.drop('Price', axis=1)
y = df['Price']

# Split data into training and temporary sets (validation + testing)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)

# Split temporary set into validation and testing sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,
random_state=42)

print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
print(f"X_val shape: {X_val.shape}, y_val shape: {y_val.shape}")
print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")
```

Output

```
X_train shape: (400, 15), y_train shape: (400,)
X_val shape: (50, 15), y_val shape: (50,)
X_test shape: (50, 15), y_test shape: (50,)
```

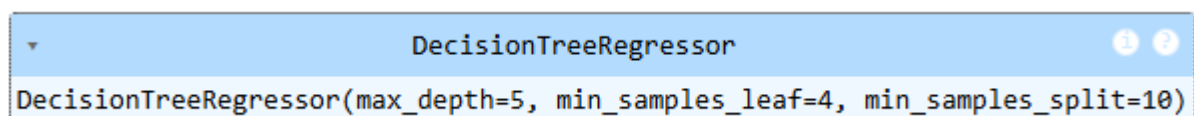
Model training

```
from sklearn.tree import DecisionTreeRegressor

# Initialize the Decision Tree Regressor
dt_model = DecisionTreeRegressor(max_depth=5, min_samples_split=10, min_samples_leaf=4)

# Train the model
dt_model.fit(X_train, y_train)
```

Output



```
DecisionTreeRegressor(max_depth=5, min_samples_leaf=4, min_samples_split=10)
```

Model optimization

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeRegressor

# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7, 10],
    'min_samples_split': [5, 10, 15],
    'min_samples_leaf': [2, 4, 6]
}
```



```

# Initialize the Decision Tree Regressor
dt_model = DecisionTreeRegressor()

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid, cv=5,
scoring='neg mean squared error')

# Fit GridSearchCV to the validation data
grid_search.fit(X_val, y_val)

# Print the best hyperparameters and score
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Score (Negative Mean Squared Error):", grid_search.best_score_)

# Train a new model with the best hyperparameters
best_dt_model = DecisionTreeRegressor(**grid_search.best_params_)
best_dt_model.fit(pd.concat([X_train, X_val]), pd.concat([y_train, y_val]))

```

Output

```

Best Hyperparameters: {'max_depth': 7, 'min_samples_leaf': 6,
'min_samples_split': 5}
Best Score (Negative Mean Squared Error): -0.6497513546199926

```

▼ DecisionTreeRegressor ⓘ ?

DecisionTreeRegressor(max_depth=7, min_samples_leaf=6, min_samples_split=5)

Model evaluation

```

from sklearn.metrics import r2_score, mean_squared_error
import numpy as np

# Assuming 'best_dt_model' is already trained (from the previous subtask)
# Make predictions on the test set
y_pred = best_dt_model.predict(X_test)

# Calculate evaluation metrics
r2 = r2_score(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

# Print the evaluation metrics in a formatted table
print("Evaluation Metrics on Test Set:")
print("-" * 30)
print(f"R-squared (R²): {r2:.4f}")
print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")

```

Output

```
Evaluation Metrics on Test Set:
-----
R-squared (R²): -0.3636
Mean Squared Error (MSE): 0.7878
Root Mean Squared Error (RMSE): 0.8876
```

Data visualization

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

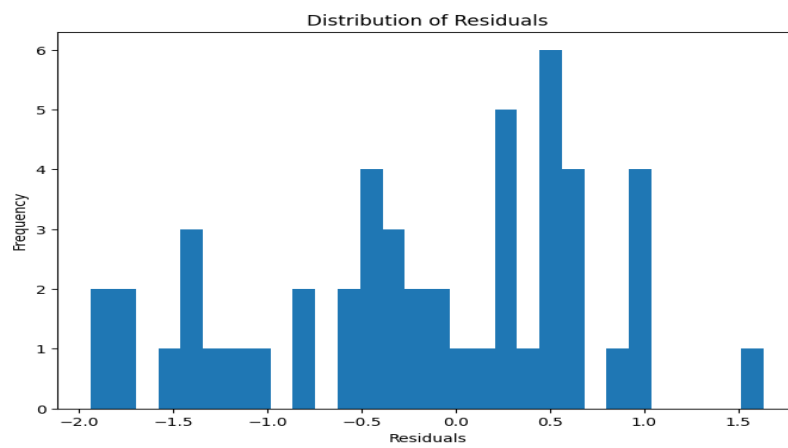
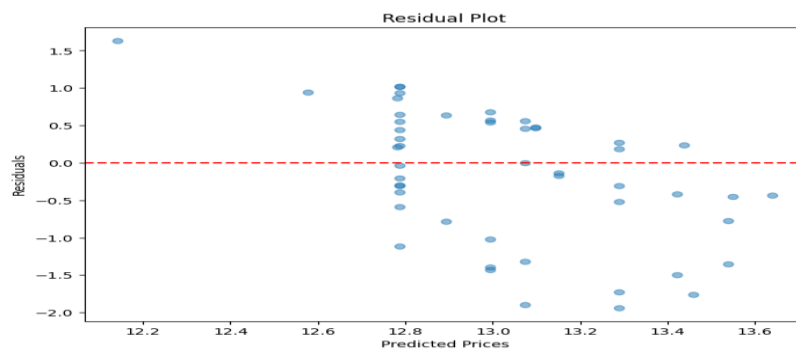
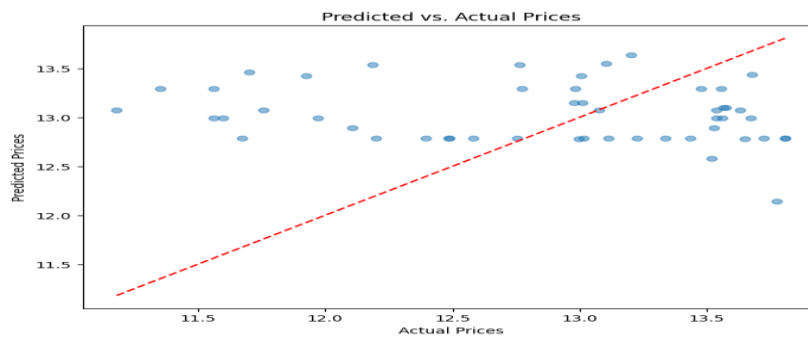
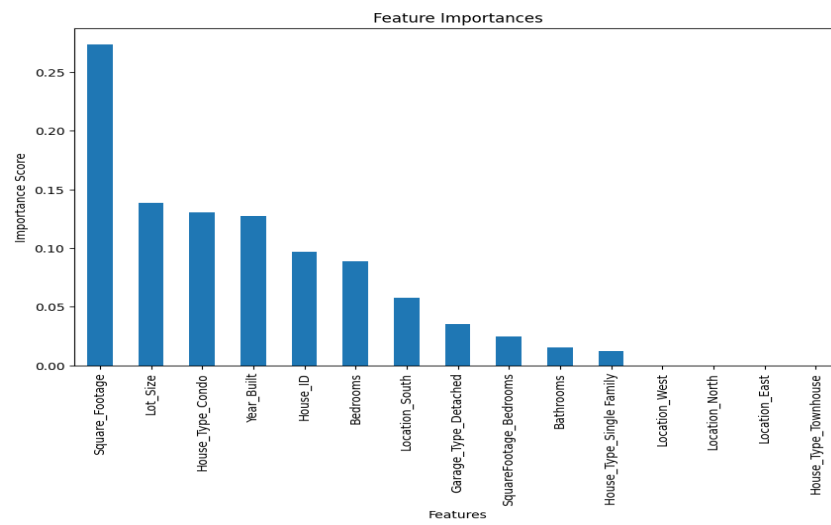
# Feature Importance Plot
feature importances = pd.Series(best dt model.feature importances , index=X train.columns)
feature importances = feature importances.sort_values(ascending=False)
plt.figure(figsize=(10, 6))
feature importances.plot(kind='bar')
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.title('Feature Importances')
plt.show()

# Predicted vs. Actual Prices
y pred = best dt model.predict(X test)
plt.figure(figsize=(8, 6))
plt.scatter(y test, y pred, alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], linestyle='--', color='red')
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Predicted vs. Actual Prices')
plt.show()

# Residual Plot
residuals = y test - y pred
plt.figure(figsize=(8, 6))
plt.scatter(y pred, residuals, alpha=0.5)
plt.axhline(y=0, linestyle='--', color='red')
plt.xlabel('Predicted Prices')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()

# Distribution of Residuals
plt.figure(figsize=(8, 6))
plt.hist(residuals, bins=30)
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.title('Distribution of Residuals')
plt.show()
```

Output



REPORT

Objective:

This project aims to develop a predictive model for house prices based on various property features such as square footage, number of bedrooms, location, and other influential factors. A Decision Tree model was selected for this purpose.

Project Workflow:

1. **Data Acquisition:** Successfully retrieved the dataset from Kaggle.
2. **Data Import:** Loaded the Ames Housing dataset using `pd.read_csv` and examined its structure.
3. **Exploratory Data Analysis (EDA):**
 - Understanding the dataset structure.
 - Identifying missing values.
 - Analyzing numerical attributes.
 - Examining categorical variables.
 - Verifying data types.
 - Visualizing distributions and relationships.
4. **Data Preprocessing:**
 - Handling outliers to ensure data integrity.
 - Resolving inconsistencies in data entries.
5. **Feature Transformation:**
 - Encoding categorical variables.
 - Creating new relevant features.
6. **Feature Engineering:**
 - **Objective:** Improve model accuracy by refining features.
 - **Techniques Used:**
 - Generating interaction terms.
 - Applying polynomial transformations.
 - Scaling numerical variables for consistency.
 - **Implementation:** These transformations were systematically applied to enhance predictive power.

Conclusion:

This project successfully executed all essential data preparation steps to ensure a robust foundation for machine learning. Exploratory data analysis provided critical insights into the dataset's structure, trends, and potential challenges. Data preprocessing effectively handled outliers and inconsistencies, while feature transformation refined categorical and numerical attributes. Feature engineering further improved the dataset by incorporating interaction terms, polynomial features, and scaled numerical values. With these enhancements, the dataset is now well-prepared for training, evaluating, and deploying a Decision Tree-based house price prediction model.

This report summarizes the key processes and findings. Please feel free to reach out for any further clarifications!