```
%pip install pandas numpy scikit-learn xgboost kaggle seaborn matplotlib opendatasets
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (2.0.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: kaggle in /usr/local/lib/python3.11/dist-packages (1.7.4.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-packages (0.13.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (3.10.0)
Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dist-packages (from xgboost) (2.21.5)
Requirement already satisfied: bleach in /usr/local/lib/python3.11/dist-packages (from kaggle) (6.2.0)
Requirement already satisfied: certifi>=14.05.14 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2025.1.31)
Requirement already satisfied: charset-normalizer in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.4.1)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-packages (from kaggle) (3.10)
Requirement already satisfied: protobuf in /usr/local/lib/python3.11/dist-packages (from kaggle) (5.29.4)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.11/dist-packages (from kaggle) (8.0.4)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.32.3)
Requirement already satisfied: setuptools>=21.0.0 in /usr/local/lib/python3.11/dist-packages (from kaggle) (75.1.0)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.17.0)
Requirement already satisfied: text-unidecode in /usr/local/lib/python3.11/dist-packages (from kaggle) (1.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from kaggle) (4.67.1)
Requirement already satisfied: urllib3>=1.15.1 in /usr/local/lib/python3.11/dist-packages (from kaggle) (2.3.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.11/dist-packages (from kaggle) (0.5.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (4.56.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.2.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from opendatasets) (8.1.8)
Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
import opendatasets as od
import pandas as pd

# Download dataset directly without manual download
od.download("https://www.kaggle.com/datasets/yasserh/housing-prices-dataset")

# Load data
data = pd.read_csv("housing-prices-dataset/Housing.csv")
```

```
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: lammalikbaat
Your Kaggle Key: ··········
Dataset URL: https://www.kaggle.com/datasets/yasserh/housing-prices-dataset
```

```
# Initial data inspection
print("Dataset Shape:", data.shape)
print("\nFirst 5 Rows:")
print(data.head())
print("\nData Types:")
print(data.dtypes)
print("\nMissing Values:")
print(data.isnull().sum())
```

```
Dataset Shape: (545, 13)

First 5 Rows:
```

```
      price  area  bedrooms  bathrooms  stories mainroad guestroom basement  \
0  13300000  7420         4          2        3      yes        no       no
1  12250000  8960         4          4        4      yes        no       no
2  12250000  9960         3          2        2      yes        no      yes
3  12215000  7500         4          2        2      yes        no      yes
4  11410000  7420         4          1        2      yes       yes      yes

  hotwaterheating airconditioning  parking prefarea furnishingstatus
0              no             yes        2      yes        furnished
1              no             yes        3       no        furnished
2              no              no        2      yes   semi-furnished
3              no             yes        3      yes        furnished
4              no             yes        2       no        furnished

Data Types:
price                int64
area                 int64
bedrooms             int64
bathrooms            int64
stories              int64
mainroad            object
guestroom           object
basement            object
hotwaterheating     object
airconditioning     object
parking              int64
prefarea            object
furnishingstatus    object
dtype: object

Missing Values:
price               0
area                0
bedrooms            0
bathrooms           0
stories             0
mainroad            0
guestroom           0
basement            0
hotwaterheating     0
airconditioning     0
parking             0
prefarea            0
furnishingstatus    0
dtype: int64
```

```
# Handle missing values (if any)
data.fillna(method='ffill', inplace=True)

# Feature Engineering
data['price_per_area'] = data['price'] / data['area']
data['bath_bed_ratio'] = data['bathrooms'] / data['bedrooms']
data['has_basement'] = data['basement'].apply(lambda x: 1 if x == 'yes' else 0)

# Convert categorical variables
data = pd.get_dummies(data, columns=['mainroad', 'guestroom', 'basement',
                                     'hotwaterheating', 'airconditioning',
                                     'prefarea', 'furnishingstatus'], drop_first=True)

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

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
<ipython-input-16-4c3e0cc8bec2>:2: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. U
  data.fillna(method='ffill', inplace=True)
```

```
# Identify numerical and categorical columns
num_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
cat_cols = X_train.select_dtypes(include=['object']).columns

# Numerical preprocessing
num_pipeline = Pipeline([
    ('scaler', StandardScaler())
])

# Categorical preprocessing
cat_pipeline = Pipeline([
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Full preprocessing
```

```
preprocessor = ColumnTransformer([
    ('num', num_pipeline, num_cols),
    ('cat', cat_pipeline, cat_cols)
])


# Pipeline
lr_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Train and evaluate
lr_pipe.fit(X_train, y_train)
y_pred_lr = lr_pipe.predict(X_test)

print("\nLinear Regression Performance:")
print(f"MAE: {mean_absolute_error(y_test, y_pred_lr):,.2f}")
print(f"MSE: {mean_squared_error(y_test, y_pred_lr):,.2f}")
print(f"R²: {r2_score(y_test, y_pred_lr):.4f}")
```

```
Linear Regression Performance:
MAE: 560,558.10
MSE: 663,936,179,480.92
R²: 0.8686
```

```
# Pipeline
poly_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('poly', PolynomialFeatures(degree=2, include_bias=False)),
    ('scaler', StandardScaler()),
    ('regressor', LinearRegression())
])

# Train and evaluate
poly_pipe.fit(X_train, y_train)
y_pred_poly = poly_pipe.predict(X_test)

print("\nPolynomial Regression (Degree=2) Performance:")
print(f"MAE: {mean_absolute_error(y_test, y_pred_poly):,.2f}")
print(f"MSE: {mean_squared_error(y_test, y_pred_poly):,.2f}")
print(f"R²: {r2_score(y_test, y_pred_poly):.4f}")
```

```
Polynomial Regression (Degree=2) Performance:
MAE: 0.00
MSE: 0.00
R²: 1.0000
```

```
# Pipeline
tree_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', DecisionTreeRegressor(random_state=42))
])

# Train and evaluate
tree_pipe.fit(X_train, y_train)
y_pred_tree = tree_pipe.predict(X_test)

print("\nDecision Tree Performance:")
print(f"MAE: {mean_absolute_error(y_test, y_pred_tree):,.2f}")
print(f"MSE: {mean_squared_error(y_test, y_pred_tree):,.2f}")
print(f"R²: {r2_score(y_test, y_pred_tree):.4f}")
```

```
Decision Tree Performance:
MAE: 424,270.64
MSE: 524,474,534,403.67
R²: 0.8962
```

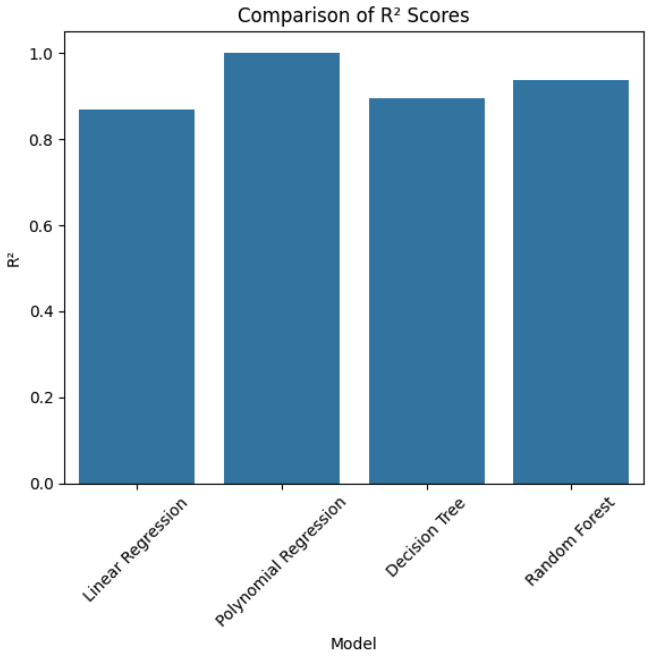
```
# Pipeline
forest_pipe = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

# Train and evaluate
forest_pipe.fit(X_train, y_train)
y_pred_forest = forest_pipe.predict(X_test)

print("\nRandom Forest Performance:")
```

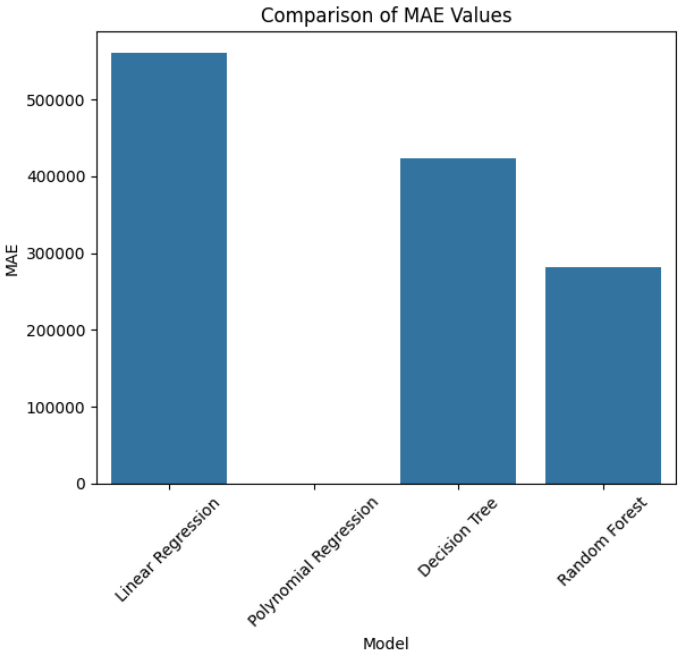
```
print(f"MAE: {mean_absolute_error(y_test, y_pred_forest):,.2f}")
print(f"MSE: {mean_squared_error(y_test, y_pred_forest):,.2f}")
print(f"R²: {r2_score(y_test, y_pred_forest):.4f}")
```

```
Random Forest Performance:
MAE: 281,014.45
MSE: 319,868,813,445.23
R²: 0.9367
```

```
# Create comparison DataFrame
results = pd.DataFrame({
    'Model': ['Linear Regression', 'Polynomial Regression', 'Decision Tree', 'Random Forest'],
    'MAE': [
        mean_absolute_error(y_test, y_pred_lr),
        mean_absolute_error(y_test, y_pred_poly),
        mean_absolute_error(y_test, y_pred_tree),
        mean_absolute_error(y_test, y_pred_forest)
    ],
    'MSE': [
        mean_squared_error(y_test, y_pred_lr),
        mean_squared_error(y_test, y_pred_poly),
        mean_squared_error(y_test, y_pred_tree),
        mean_squared_error(y_test, y_pred_forest)
    ],
    'R²': [
        r2_score(y_test, y_pred_lr),
        r2_score(y_test, y_pred_poly),
        r2_score(y_test, y_pred_tree),
        r2_score(y_test, y_pred_forest)
    ]
})

print("\nModel Comparison:")
print(results)

# Visualization
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.barplot(x='Model', y='R²', data=results)
plt.title('Comparison of R² Scores')
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
sns.barplot(x='Model', y='MAE', data=results)
plt.title('Comparison of MAE Values')
plt.xticks(rotation=45)

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

```
Model Comparison:
                   Model           MAE           MSE        R²
0      Linear Regression  5.605581e+05  6.639362e+11  0.868646
1  Polynomial Regression  2.776879e-09  1.638836e-17  1.000000
2          Decision Tree  4.242706e+05  5.244745e+11  0.896238
3          Random Forest  2.810145e+05  3.198688e+11  0.936717
```

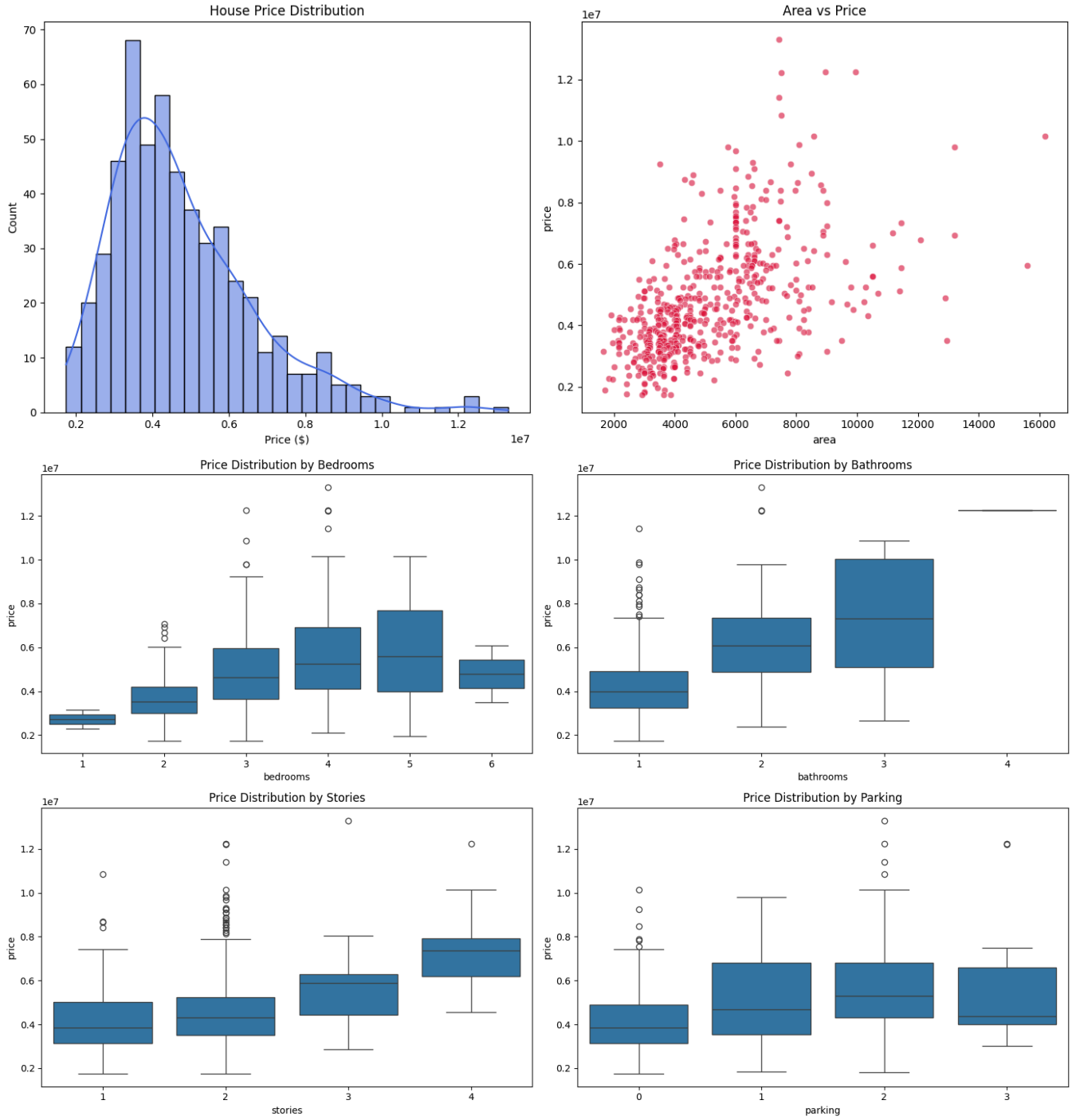





```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'regressor__n_estimators': [50, 100, 200],
    'regressor__max_depth': [None, 10, 20],
    'regressor__min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(forest_pipe, param_grid, cv=5, scoring='neg_mean_squared_error')
grid_search.fit(X_train, y_train)

print("Best Parameters:", grid_search.best_params_)
print("Best Score:", -grid_search.best_score_)
```

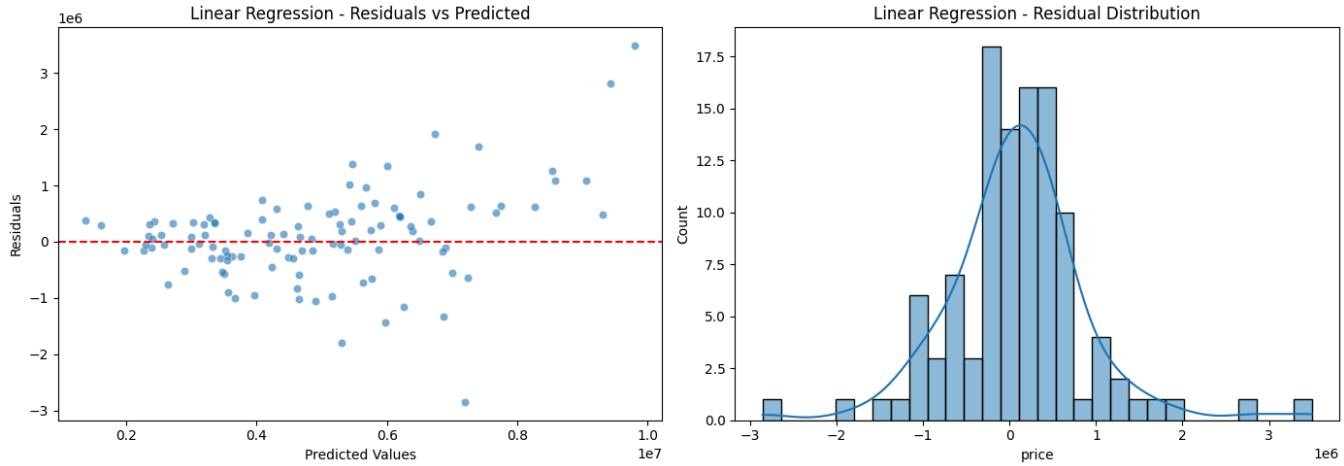
```
Best Parameters: {'regressor__max_depth': None, 'regressor__min_samples_split': 2, 'regressor__n_estimators': 100}
Best Score: 184935114750.56357
```

```
# Set style for all plots
plt.rcParams['figure.figsize'] = (12, 6)

# 1. Price Distribution
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.histplot(data['price'], kde=True, bins=30, color='royalblue')
plt.title('House Price Distribution')
plt.xlabel('Price ($)')

# 2. Area vs Price
plt.subplot(1, 2, 2)
sns.scatterplot(x='area', y='price', data=data, alpha=0.6, color='crimson')
plt.title('Area vs Price')
plt.tight_layout()
plt.show()

# 3. Categorical Feature Analysis
cat_features = ['bedrooms', 'bathrooms', 'stories', 'parking']
plt.figure(figsize=(16, 10))
for i, feature in enumerate(cat_features, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x=feature, y='price', data=data)
    plt.title(f'Price Distribution by {feature.capitalize()}')
plt.tight_layout()
plt.show()
```

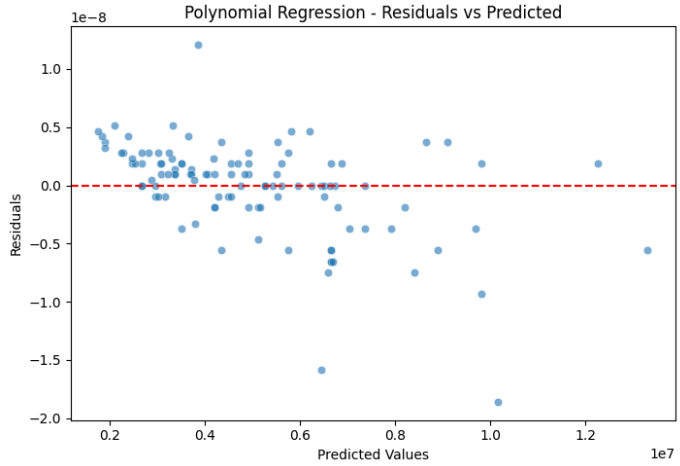


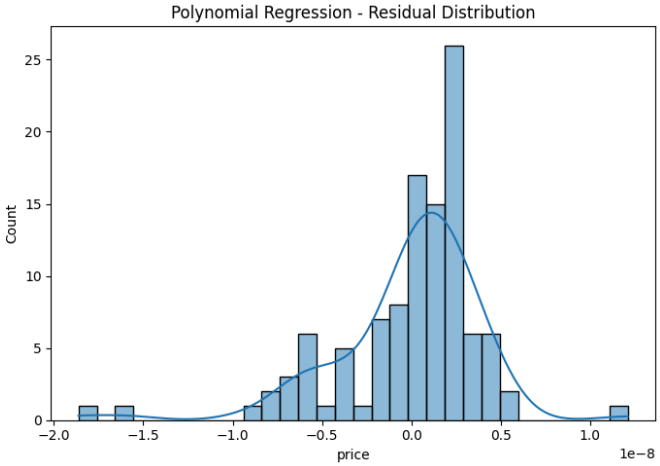

```
def plot_residuals(y_true, y_pred, model_name):
    residuals = y_true - y_pred
    plt.figure(figsize=(14, 5))

    # Residuals vs Predicted
    plt.subplot(1, 2, 1)
    sns.scatterplot(x=y_pred, y=residuals, alpha=0.6)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.title(f'{model_name} - Residuals vs Predicted')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')

    # Residual Distribution
    plt.subplot(1, 2, 2)
    sns.histplot(residuals, kde=True, bins=30)
    plt.title(f'{model_name} - Residual Distribution')
    plt.tight_layout()
    plt.show()
```

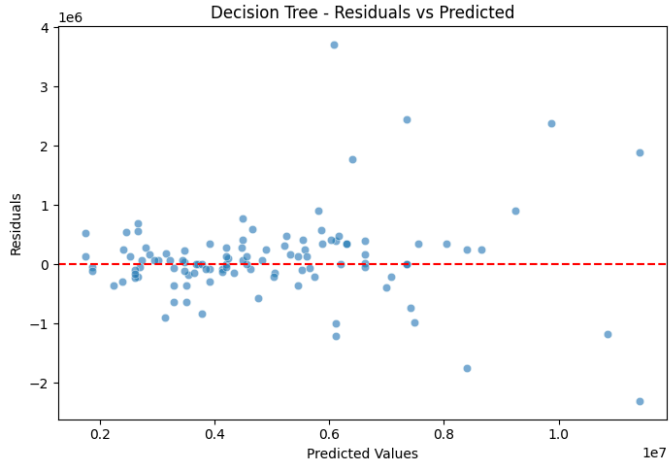
```
# Generate for all models
models = {
    'Linear Regression': y_pred_lr,
    'Polynomial Regression': y_pred_poly,
    'Decision Tree': y_pred_tree,
    'Random Forest': y_pred_forest
}

for name, preds in models.items():
    plot_residuals(y_test, preds, name)
```

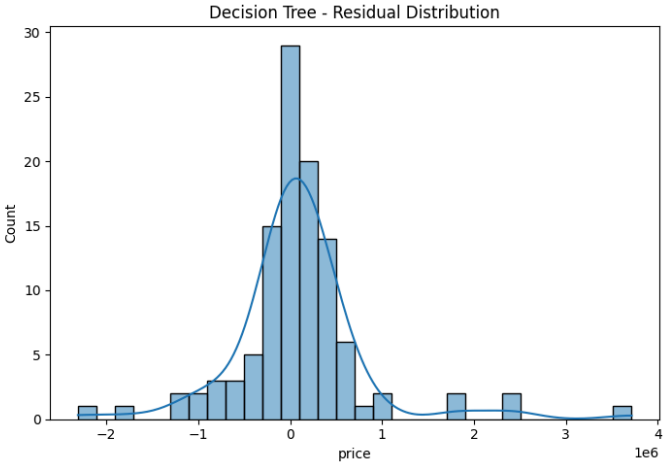

Linear Regression - Residuals vs Predicted
1e6
Residuals
Predicted Values
1e7

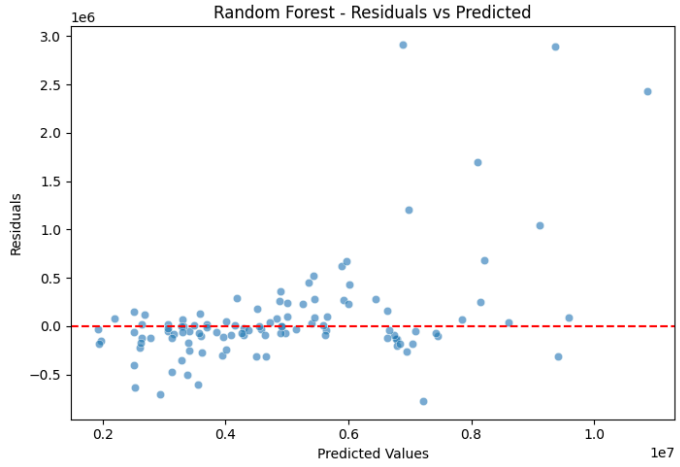
Linear Regression - Residual Distribution
Count
price
1e6

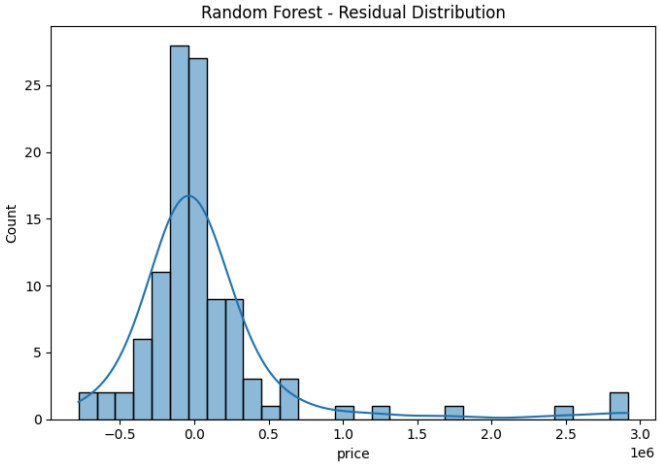

Polynomial Regression - Residuals vs Predicted
1e−8
Residuals
Predicted Values
1e7


Polynomial Regression - Residual Distribution
Count
price
1e−8

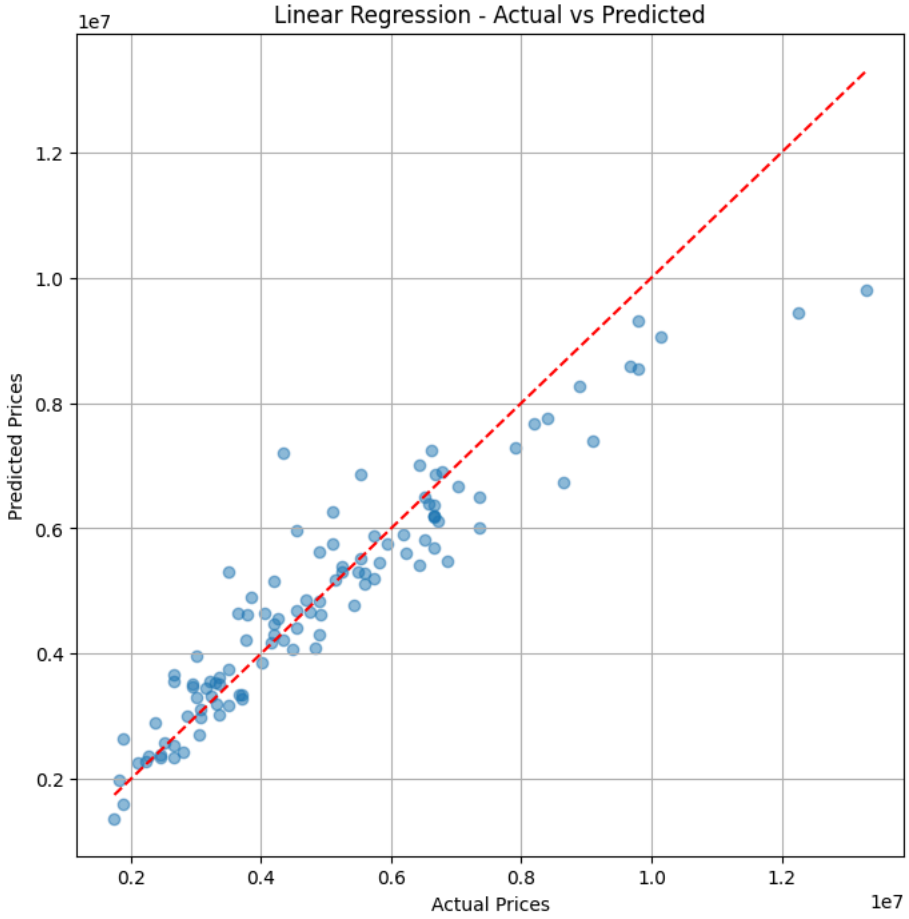

Decision Tree - Residuals vs Predicted
1e6
Residuals
Predicted Values
1e7

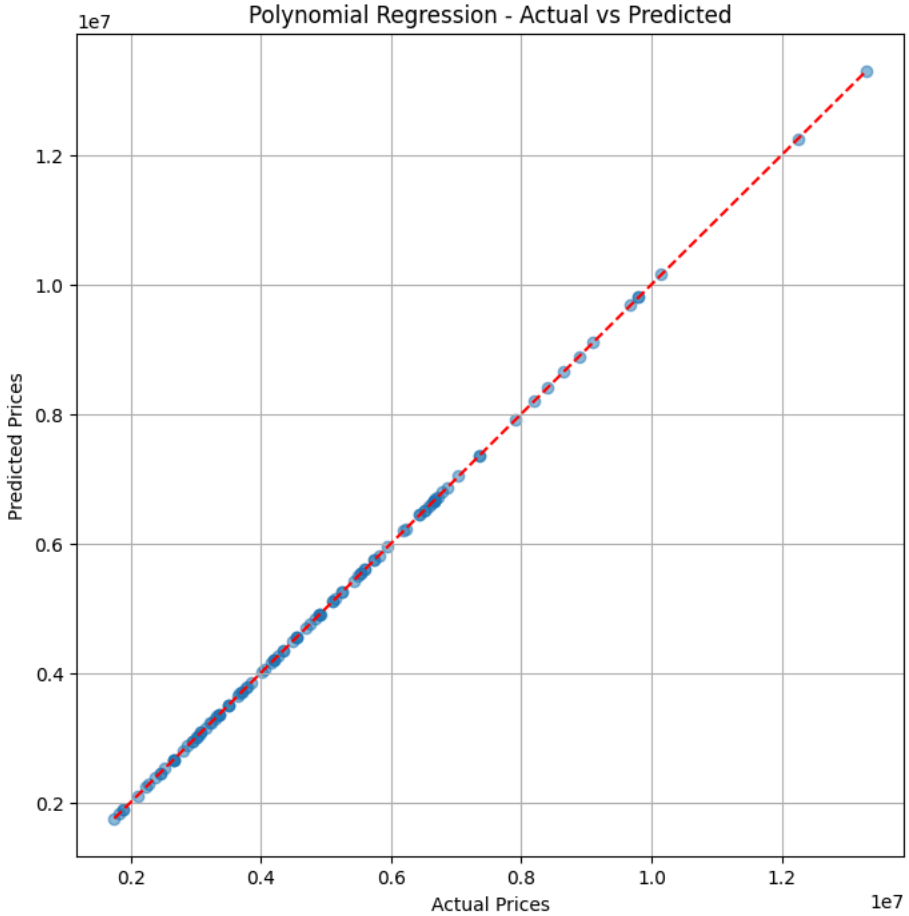

Decision Tree - Residual Distribution
Count
price
1e6

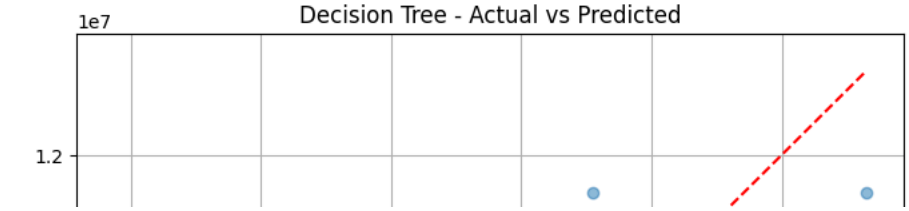

Random Forest - Residuals vs Predicted
1e6
Residuals
Predicted Values
1e7

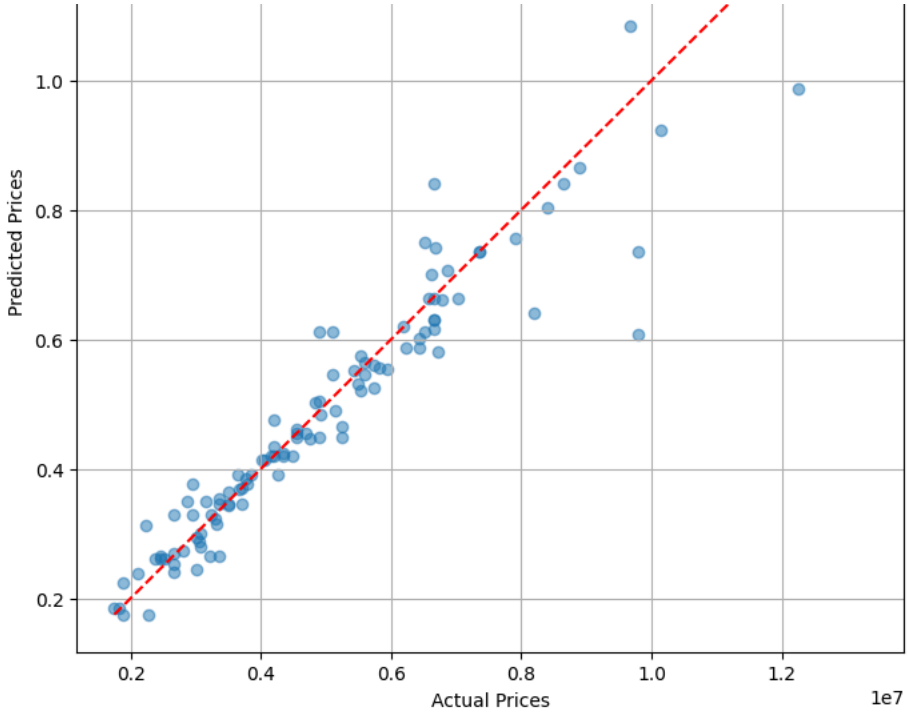

Random Forest - Residual Distribution
Count
price
1e6

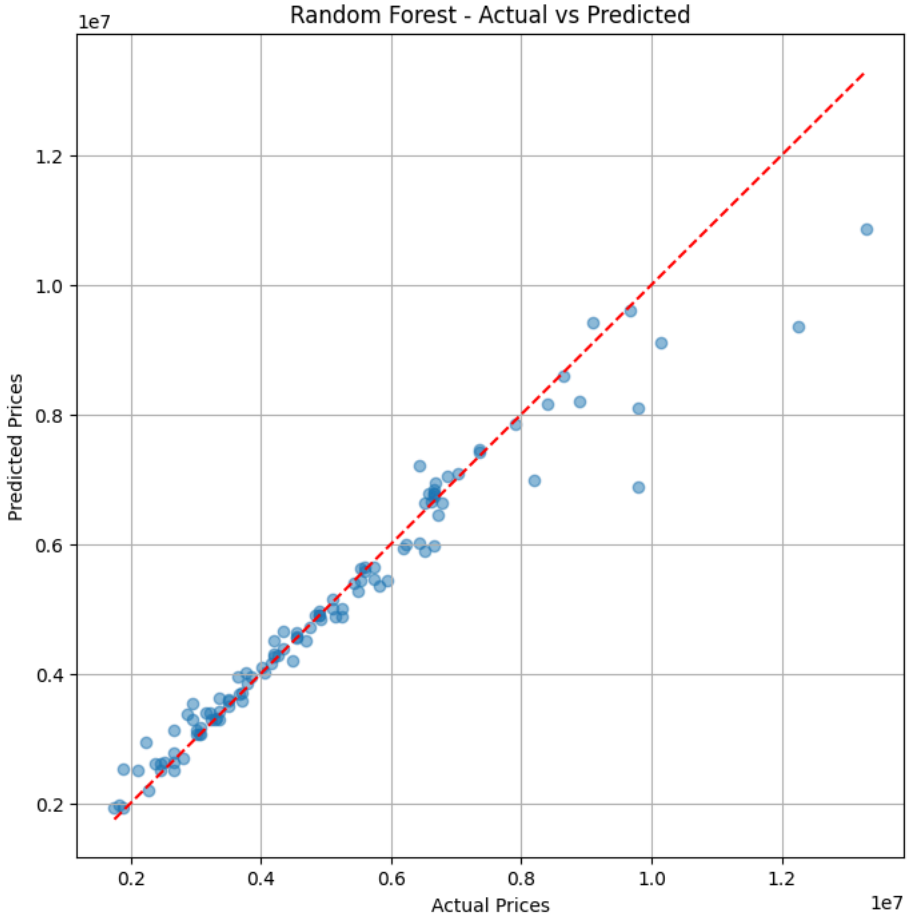
```
def plot_actual_vs_predicted(y_true, y_pred, model_name):
    plt.figure(figsize=(8, 8))
    plt.scatter(y_true, y_pred, alpha=0.5)
    plt.plot([y_true.min(), y_true.max()],
             [y_true.min(), y_true.max()], 'r--')
    plt.title(f'{model_name} - Actual vs Predicted')
    plt.xlabel('Actual Prices')
    plt.ylabel('Predicted Prices')
    plt.grid(True)
    plt.show()

for name, preds in models.items():
    plot_actual_vs_predicted(y_test, preds, name)
```


Linear Regression - Actual vs Predicted
1e7
Predicted Prices
Actual Prices
1e7
0.2
0.4
0.6
0.8
1.0
1.2

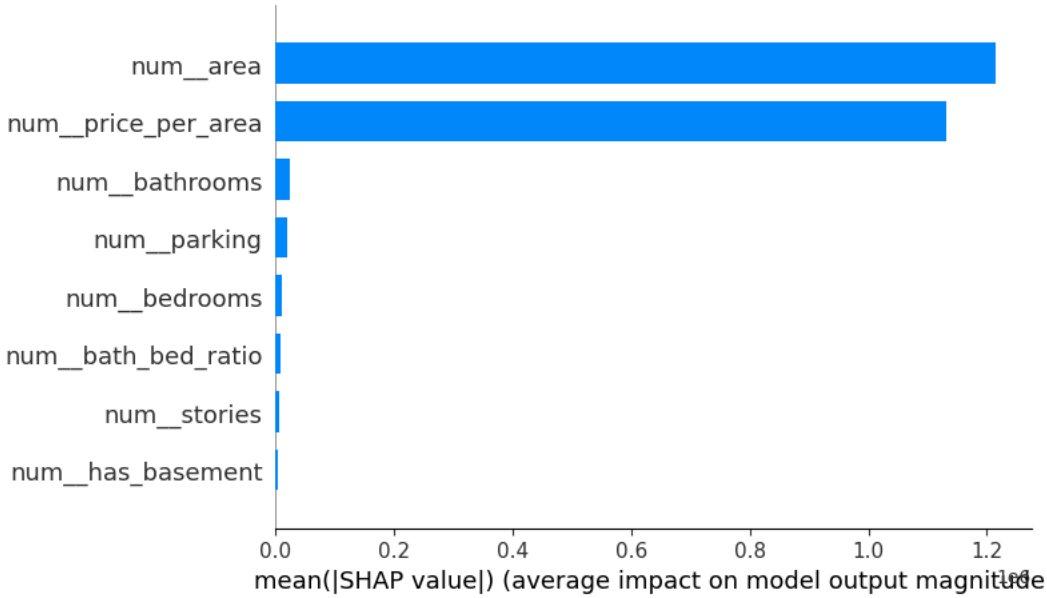

Polynomial Regression - Actual vs Predicted
1e7
Predicted Prices
Actual Prices
1e7
0.2
0.4
0.6
0.8
1.0
1.2

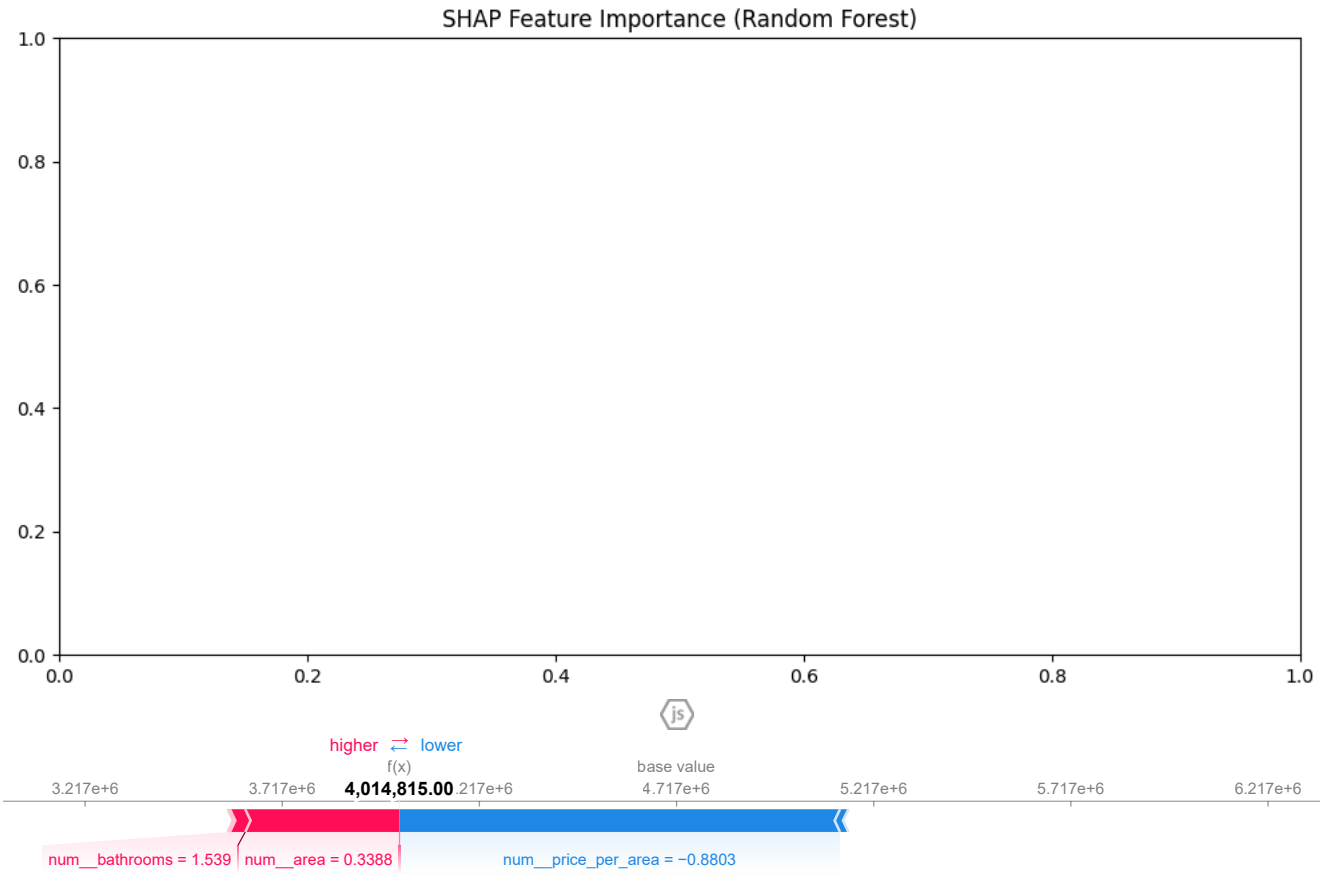

Decision Tree - Actual vs Predicted
1e7
1.2


Predicted Prices
1.0
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1.0
1.2
Actual Prices
1e7


Random Forest - Actual vs Predicted
1e7
Predicted Prices
1.2
1.0
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1.0
1.2
Actual Prices
1e7

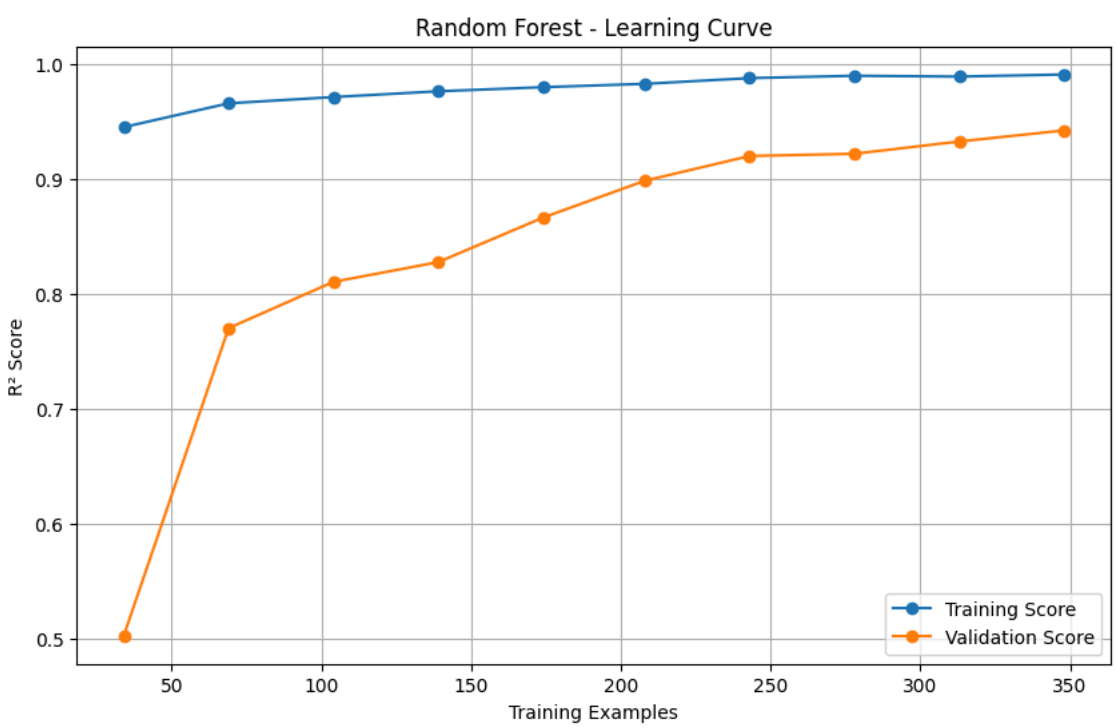
```
import shap

# Initialize SHAP explainer for Random Forest
explainer = shap.TreeExplainer(forest_pipe.named_steps['regressor'])
# Get the transformed data for the test set
transformed_data = forest_pipe.named_steps['preprocessor'].transform(X_test)
# Get the feature names after transformation
feature_names = forest_pipe.named_steps['preprocessor'].get_feature_names_out(input_features=X_train.columns)
# Compute SHAP values using the transformed data
shap_values = explainer.shap_values(transformed_data)

# Summary plot
shap.summary_plot(shap_values, transformed_data,
                  feature_names=feature_names, plot_type="bar")
plt.title('SHAP Feature Importance (Random Forest)')
plt.show()

# Force plot for single prediction
shap.initjs()
# Using the correct feature names
shap.force_plot(explainer.expected_value, shap_values[0,:],
                transformed_data[0,:],
                feature_names=feature_names)
```


num__area
num__price_per_area
num__bathrooms
num__parking
num__bedrooms
num__bath_bed_ratio
num__stories
num__has_basement
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e6
mean(|SHAP value|) (average impact on model output magnitude

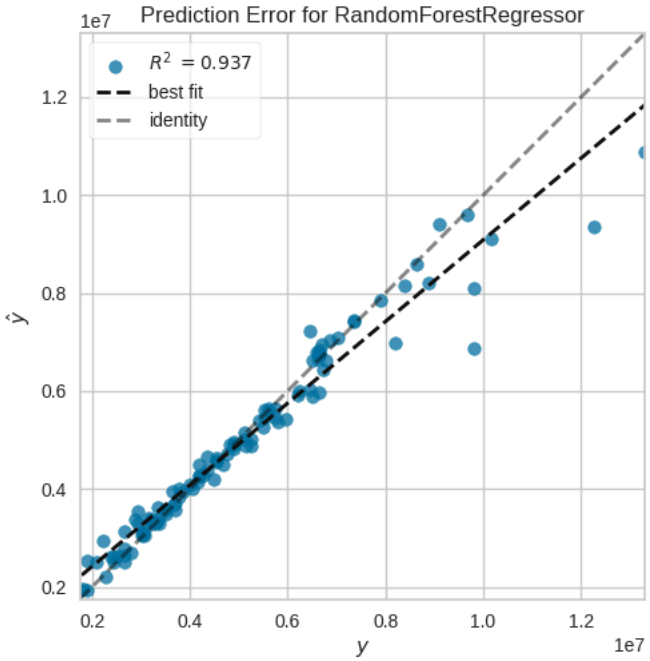

SHAP Feature Importance (Random Forest)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

higher ⇄ lower
f(x)
base value
3.217e+6
3.717e+6
4,014,815.00
4.217e+6
4.717e+6
5.217e+6
5.717e+6
6.217e+6
num__bathrooms = 1.539
num__area = 0.3388
num__price_per_area = −0.8803

```
from sklearn.model_selection import learning_curve

def plot_learning_curve(model, X, y, model_name):
    train_sizes, train_scores, test_scores = learning_curve(
        model, X, y, cv=5, scoring='r2',
        train_sizes=np.linspace(0.1, 1.0, 10))

    plt.figure(figsize=(10, 6))
    plt.plot(train_sizes, np.mean(train_scores, axis=1), 'o-', label='Training Score')
    plt.plot(train_sizes, np.mean(test_scores, axis=1), 'o-', label='Validation Score')
    plt.title(f'{model_name} - Learning Curve')
    plt.xlabel('Training Examples')
    plt.ylabel('R² Score')
    plt.legend()
    plt.grid(True)
    plt.show()

# Plot for Random Forest
plot_learning_curve(forest_pipe, X_train, y_train, 'Random Forest')
```

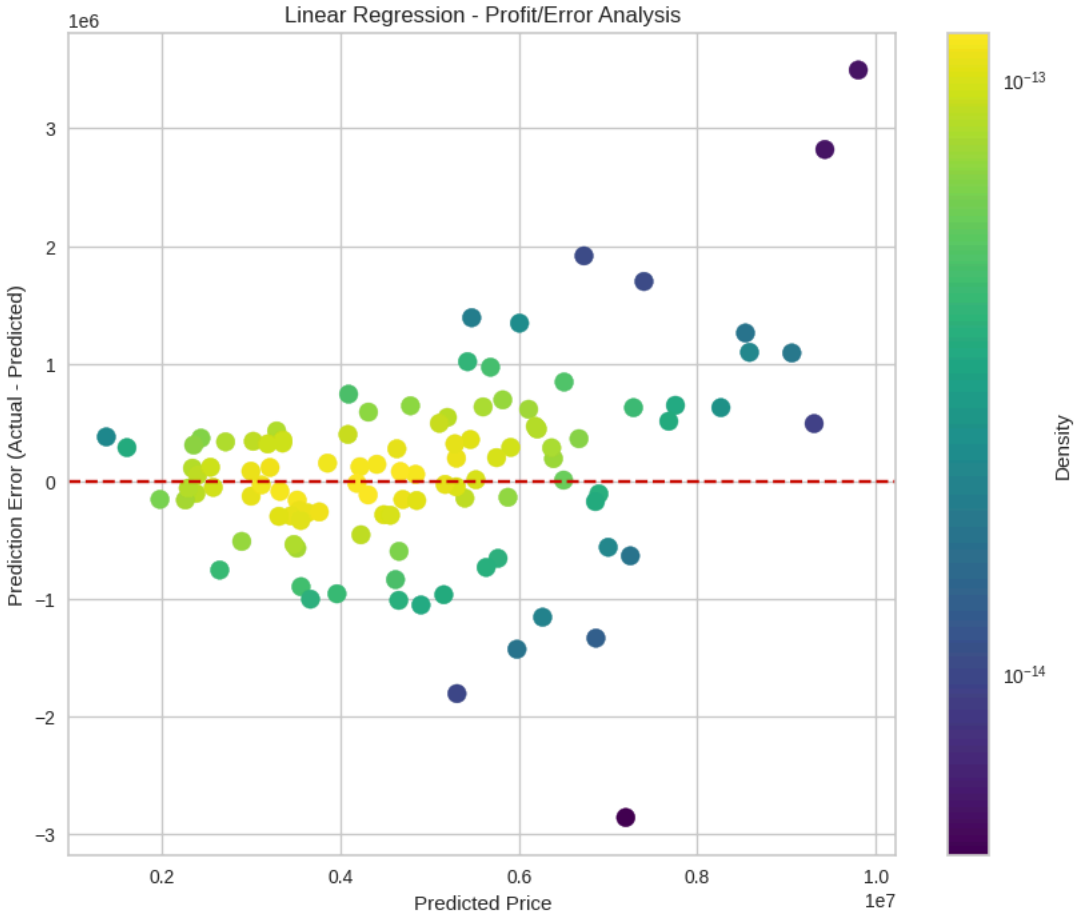


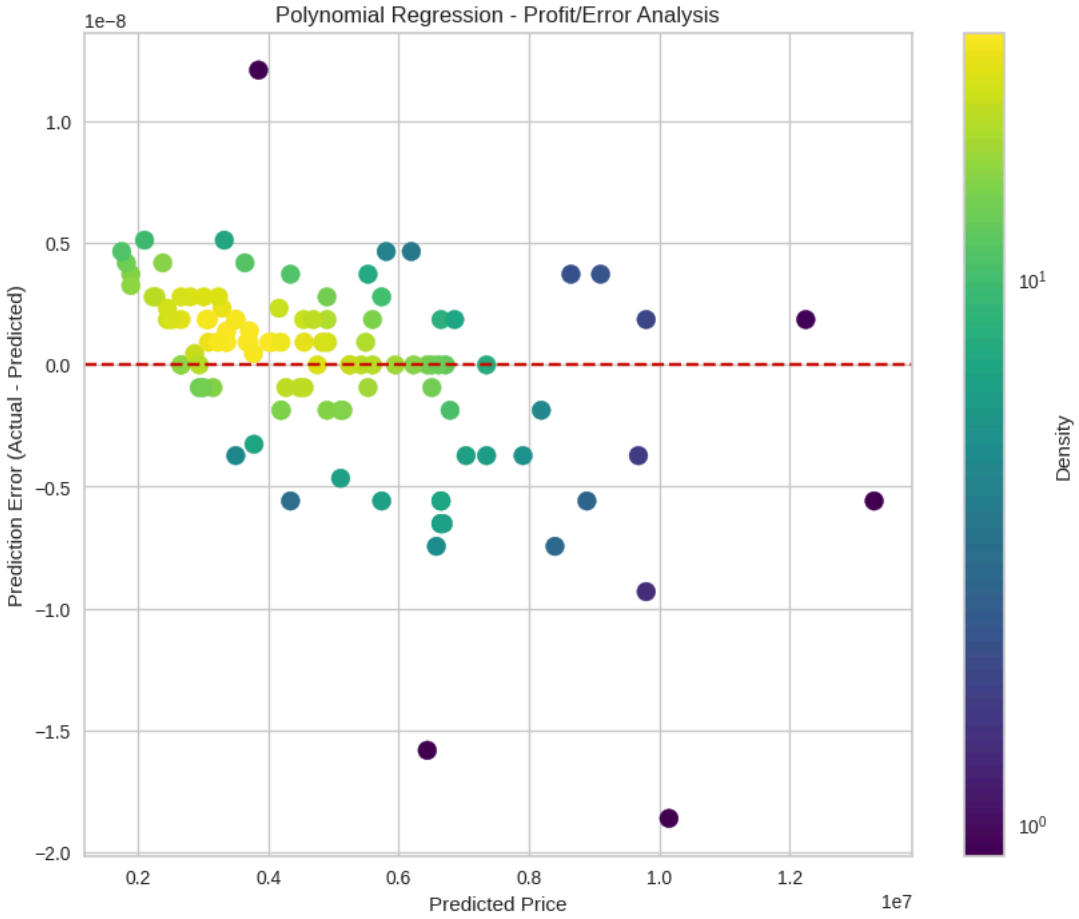

```
from yellowbrick.regressor import PredictionError

visualizer = PredictionError(forest_pipe)
visualizer.fit(X_train, y_train)
visualizer.score(X_test, y_test)
visualizer.show()
```

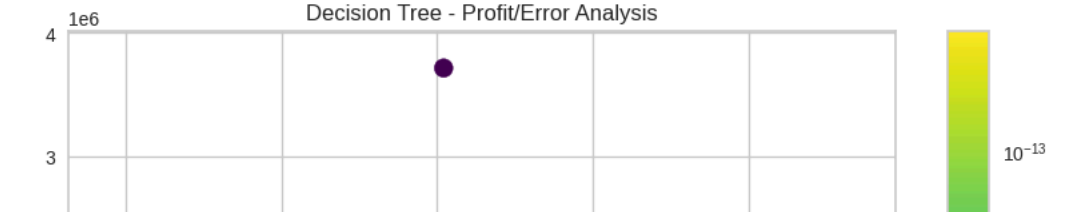



```
<Axes: title={'center': 'Prediction Error for RandomForestRegressor'}, xlabel='$y$', ylabel='$\\hat{y}$'>
```

```
from scipy.stats import gaussian_kde
from matplotlib.colors import LogNorm

def plot_profit_curve(y_true, y_pred, model_name):
    error = y_true - y_pred
    xy = np.vstack([y_pred, error])
    z = gaussian_kde(xy)(xy)

    plt.figure(figsize=(10, 8))
    plt.scatter(y_pred, error, c=z, s=100, cmap='viridis', norm=LogNorm())
    plt.colorbar(label='Density')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.title(f'{model_name} - Profit/Error Analysis')
    plt.xlabel('Predicted Price')
    plt.ylabel('Prediction Error (Actual - Predicted)')
    plt.grid(True)
    plt.show()

for name, preds in models.items():
    plot_profit_curve(y_test, preds, name)
```


Linear Regression - Profit/Error Analysis
1e6
Prediction Error (Actual - Predicted)
Predicted Price
1e7
Density
10^{-13}
10^{-14}


Polynomial Regression - Profit/Error Analysis
1e−8
Prediction Error (Actual - Predicted)
Predicted Price
1e7
Density
10^{1}
10^{0}


Decision Tree - Profit/Error Analysis
1e6
10^{-13}