melbourne-train

July 30, 2024

1 Data Analysis Project

1.1 Student Information

Name:	Ramesh Chandra Soren
Enrollment No:	2022CSB086
Department:	Computer Science and Technology

1.2 Project: Overfitting and Underfitting Analysis using Regression Models

1.2.1 Dataset: Melbourne Housing Snapshot

This notebook demonstrates an analysis of overfitting and underfitting issues using various regression models on the Melbourne Housing Snapshot Dataset.

1.2.2 Table of Contents

---

2 Overfitting and Underfitting Analysis using Regression Models

This notebook demonstrates an analysis of overfitting and underfitting issues using various regression models on the Melbourne Housing Snapshot Dataset.

2.1 1. Import Libraries

First, we'll import the necessary libraries for our analysis.

```
[ ]: 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, learning_curve
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

2.2 2. Load and Explore the Dataset

We'll load the Melbourne Housing Snapshot Dataset and take a look at its structure.

```
[ ]: # Load the dataset
# Make sure you've uploaded the melbourne.csv file to your Colab environment
df = pd.read_csv('melb_data.csv')

# Display the first few rows and dataset info
print(df.head())
print("\nDataset Information:")
print(df.info())
```

```
       Suburb           Address  Rooms Type      Price Method SellerG  \
0  Abbotsford      85 Turner St      2    h  1480000.0      S  Biggin
1  Abbotsford   25 Bloomburg St      2    h  1035000.0      S  Biggin
2  Abbotsford      5 Charles St      3    h  1465000.0     SP  Biggin
3  Abbotsford  40 Federation La      3    h   850000.0     PI  Biggin
4  Abbotsford       55a Park St      4    h  1600000.0     VB  Nelson

        Date  Distance  Postcode  …  Bathroom  Car  Landsize  BuildingArea  \
0  3/12/2016       2.5    3067.0  …       1.0  1.0     202.0           NaN
1  4/02/2016       2.5    3067.0  …       1.0  0.0     156.0          79.0
2  4/03/2017       2.5    3067.0  …       2.0  0.0     134.0         150.0
3  4/03/2017       2.5    3067.0  …       2.0  1.0      94.0           NaN
4  4/06/2016       2.5    3067.0  …       1.0  2.0     120.0         142.0

   YearBuilt  CouncilArea Lattitude  Longtitude             Regionname  \
0        NaN        Yarra  -37.7996    144.9984  Northern Metropolitan
1     1900.0        Yarra  -37.8079    144.9934  Northern Metropolitan
2     1900.0        Yarra  -37.8093    144.9944  Northern Metropolitan
3        NaN        Yarra  -37.7969    144.9969  Northern Metropolitan
4     2014.0        Yarra  -37.8072    144.9941  Northern Metropolitan

  Propertycount
0        4019.0
1        4019.0
```

```
2        4019.0
3        4019.0
4        4019.0

[5 rows x 21 columns]

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Suburb         13580 non-null  object
 1   Address        13580 non-null  object
 2   Rooms          13580 non-null  int64
 3   Type           13580 non-null  object
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object
 6   SellerG        13580 non-null  object
 7   Date           13580 non-null  object
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
None
```

2.3 3. Data Preprocessing

In this step, we'll check for missing values, select features and target variable, and normalize the features.

[ ]:
```
# Check for missing values
print("Missing values:")
print(df.isnull().sum())

# Select features and target
# Adjust these based on the actual columns in your dataset
```

```python
features = ['Rooms', 'Distance', 'Bathroom', 'Car', 'Landsize', 'BuildingArea',
 ↪'YearBuilt']
target = 'Price'

X = df[features]
y = df[target]

# Handle missing values
X = X.fillna(X.mean())

# Normalize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
 ↪random_state=42)

print("\nShape of training set:", X_train.shape)
print("Shape of testing set:", X_test.shape)
```

```
Missing values:
Suburb              0
Address             0
Rooms               0
Type                0
Price               0
Method              0
SellerG             0
Date                0
Distance            0
Postcode            0
Bedroom2            0
Bathroom            0
Car                62
Landsize            0
BuildingArea     6450
YearBuilt        5375
CouncilArea      1369
Lattitude           0
Longtitude          0
Regionname          0
Propertycount       0
dtype: int64

Shape of training set: (10864, 7)
Shape of testing set: (2716, 7)
```

2.4 4. Model Development and Evaluation

We'll create a function to train and evaluate our models, then apply it to Linear Regression, Decision Tree, and Random Forest regressors.

[ ]:
```
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    train_mse = mean_squared_error(y_train, y_train_pred)
    test_mse = mean_squared_error(y_test, y_test_pred)
    train_r2 = r2_score(y_train, y_train_pred)
    test_r2 = r2_score(y_test, y_test_pred)

    print(f"Train MSE: {train_mse:.2f}")
    print(f"Test MSE: {test_mse:.2f}")
    print(f"Train R2: {train_r2:.4f}")
    print(f"Test R2: {test_r2:.4f}")

    return model, y_train_pred, y_test_pred

# Linear Regression
print("Linear Regression:")
lr_model, lr_train_pred, lr_test_pred = 
 ↪train_evaluate_model(LinearRegression(), X_train, X_test, y_train, y_test)

# Decision Tree
print("\nDecision Tree:")
dt_model, dt_train_pred, dt_test_pred = 
 ↪train_evaluate_model(DecisionTreeRegressor(random_state=42), X_train, 
 ↪X_test, y_train, y_test)

# Random Forest
print("\nRandom Forest:")
rf_model, rf_train_pred, rf_test_pred = 
 ↪train_evaluate_model(RandomForestRegressor(random_state=42), X_train, 
 ↪X_test, y_train, y_test)
```

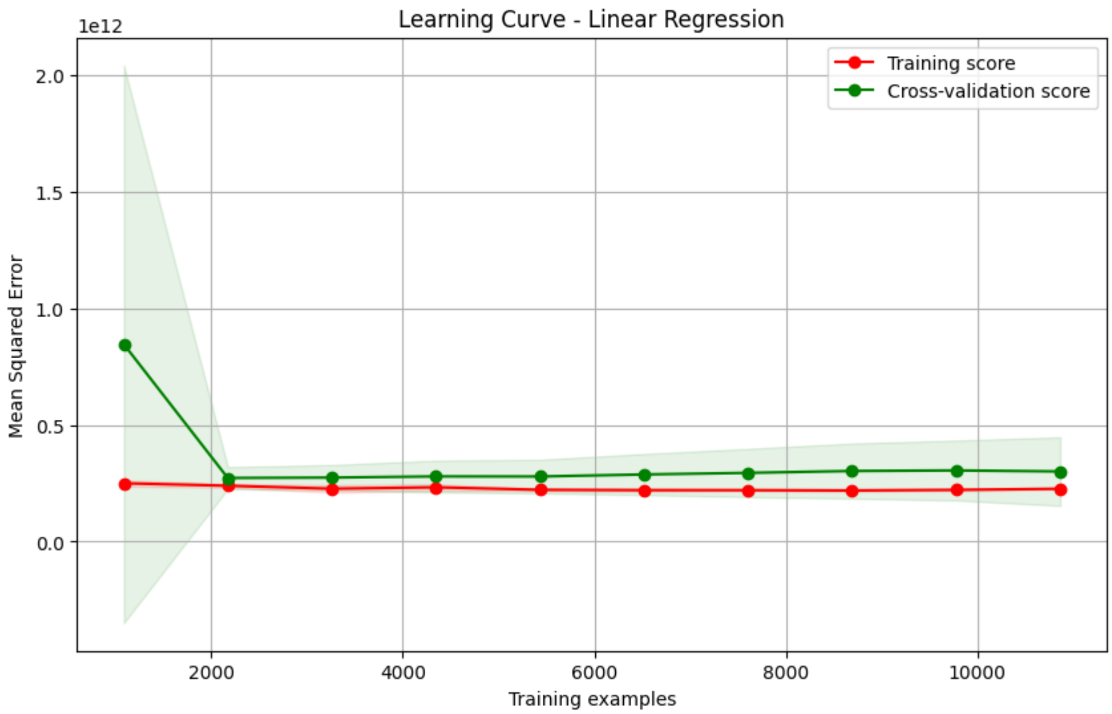
```
Linear Regression:
Train MSE: 229734929577.10
Test MSE: 223129520570.14
Train R2: 0.4418
Test R2: 0.4383

Decision Tree:
Train MSE: 743128918.30
Test MSE: 233234371329.80
```

```
Train R2: 0.9982
Test R2: 0.4128

Random Forest:
Train MSE: 21554415326.57
Test MSE: 132467864596.03
Train R2: 0.9476
Test R2: 0.6665
```

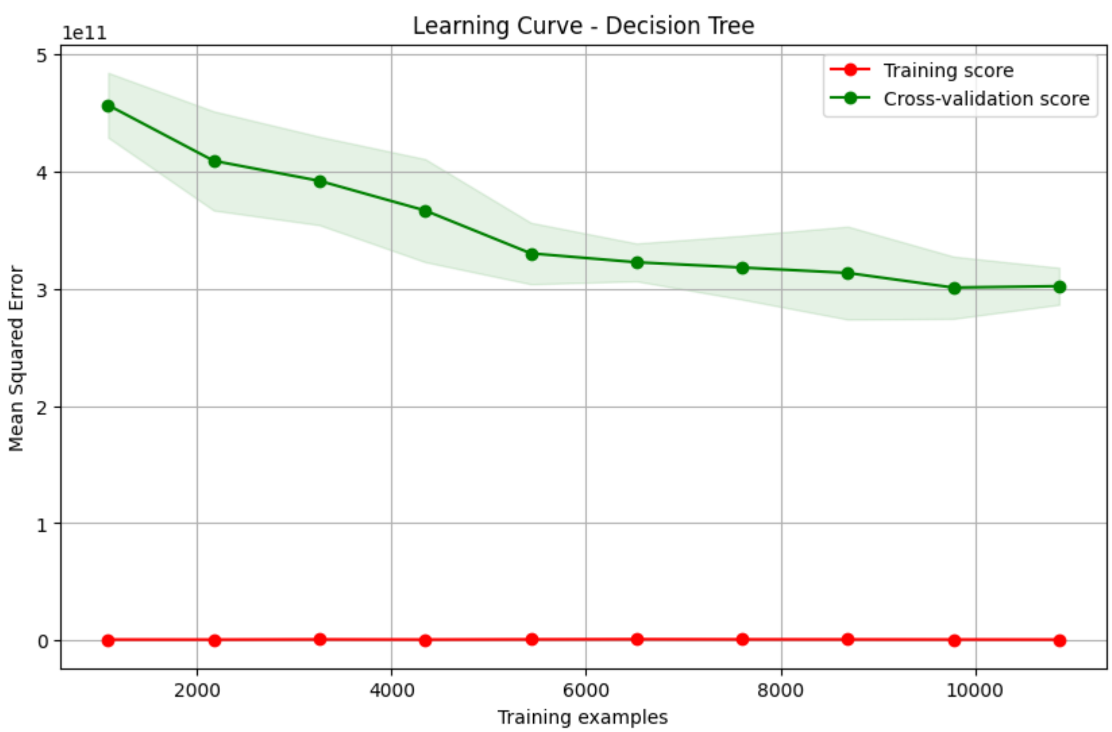
2.5 5. Learning Curves for Overfitting/Underfitting Analysis

We'll plot learning curves to visualize how the model's performance changes with increasing amounts of training data.

[ ]:
```python
def plot_learning_curve(model, X, y, title):
    train_sizes, train_scores, val_scores = learning_curve(
        model, X, y, cv=5, n_jobs=-1, train_sizes=np.linspace(0.1, 1.0, 10),
        scoring='neg_mean_squared_error')

    train_scores_mean = -np.mean(train_scores, axis=1)
    train_scores_std = np.std(train_scores, axis=1)
    val_scores_mean = -np.mean(val_scores, axis=1)
    val_scores_std = np.std(val_scores, axis=1)

    plt.figure(figsize=(10, 6))
    plt.title(title)
    plt.xlabel("Training examples")
    plt.ylabel("Mean Squared Error")
    plt.grid()

    plt.fill_between(train_sizes, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.1, color="r")
    plt.fill_between(train_sizes, val_scores_mean - val_scores_std,
                     val_scores_mean + val_scores_std, alpha=0.1, color="g")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r", label="Training␣
↪score")
    plt.plot(train_sizes, val_scores_mean, 'o-', color="g",␣
↪label="Cross-validation score")
    plt.legend(loc="best")
    plt.show()

plot_learning_curve(lr_model, X_scaled, y, "Learning Curve - Linear Regression")
plot_learning_curve(dt_model, X_scaled, y, "Learning Curve - Decision Tree")
plot_learning_curve(rf_model, X_scaled, y, "Learning Curve - Random Forest")
```



Learning Curve - Linear Regression
1e12
Training score
Cross-validation score
2.0
1.5
1.0
0.5
0.0
Mean Squared Error
2000
4000
6000
8000
10000
Training examples

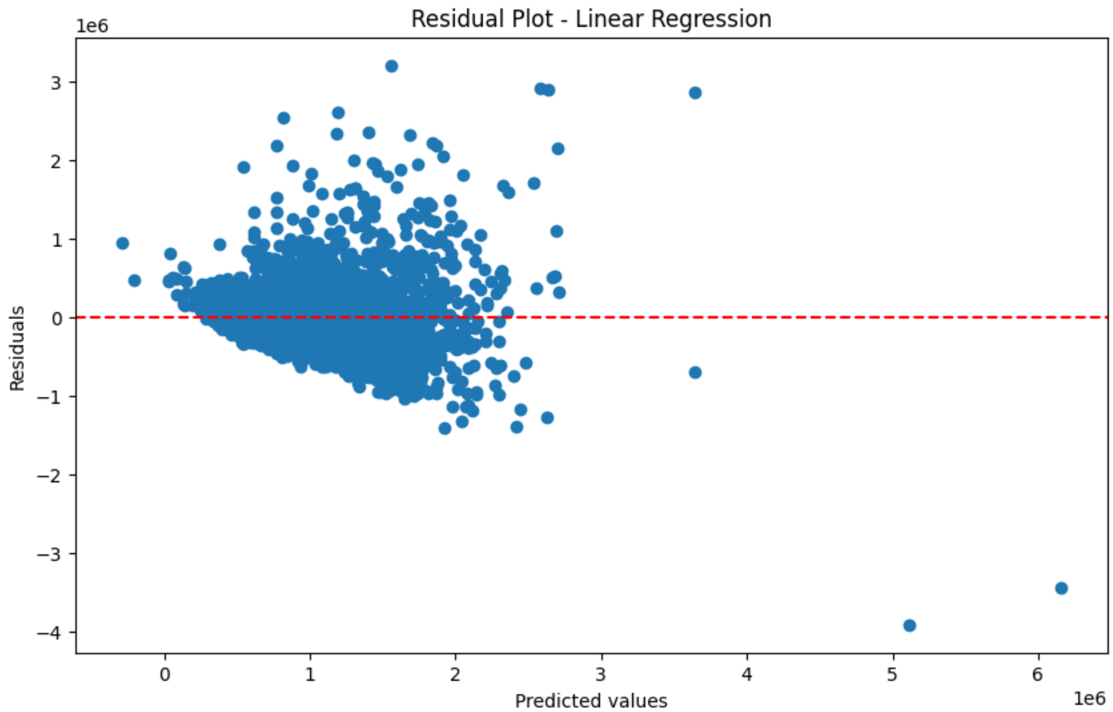


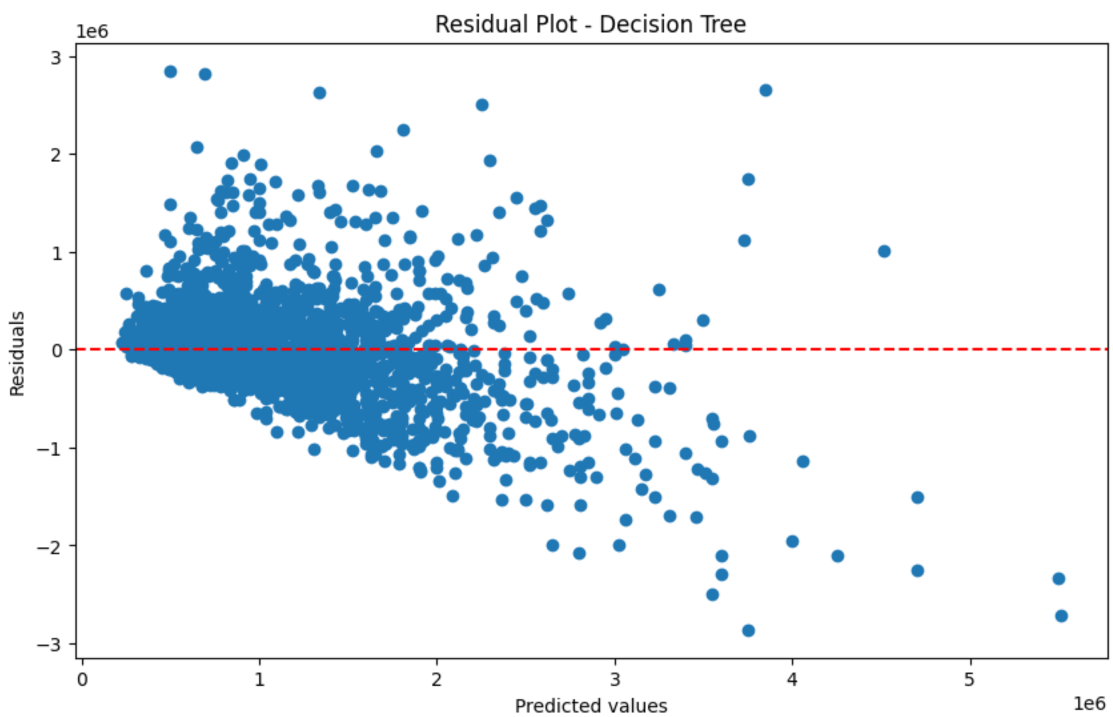
Learning Curve - Decision Tree
1e11
Training score
Cross-validation score
5
4
3
2
1
0
Mean Squared Error
2000
4000
6000
8000
10000
Training examples

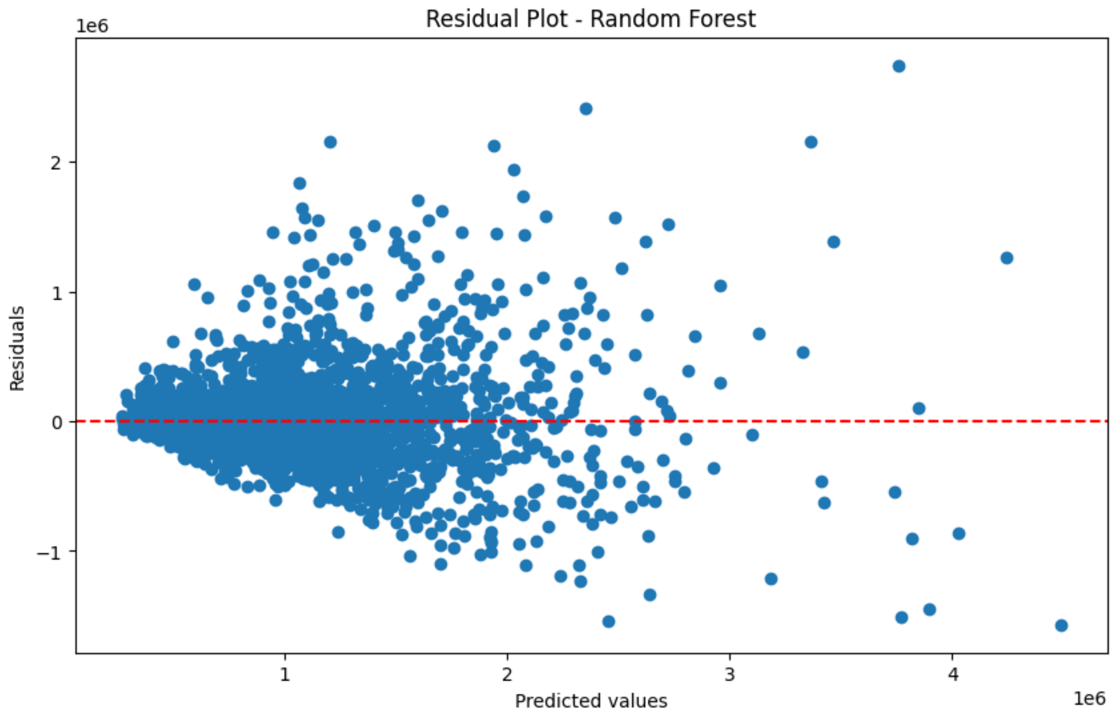
2.6 6. Residual Plots

We'll create residual plots to visualize the performance of our models.

[ ]:
```
def plot_residuals(y_true, y_pred, title):
    residuals = y_true - y_pred
    plt.figure(figsize=(10, 6))
    plt.scatter(y_pred, residuals)
    plt.title(title)
    plt.xlabel('Predicted values')
    plt.ylabel('Residuals')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.show()

plot_residuals(y_test, lr_test_pred, "Residual Plot - Linear Regression")
plot_residuals(y_test, dt_test_pred, "Residual Plot - Decision Tree")
plot_residuals(y_test, rf_test_pred, "Residual Plot - Random Forest")
```


Residual Plot - Linear Regression
1e6
3
2
1
0
−1
−2
−3
−4
Residuals
0
1
2
3
4
5
6
Predicted values
1e6

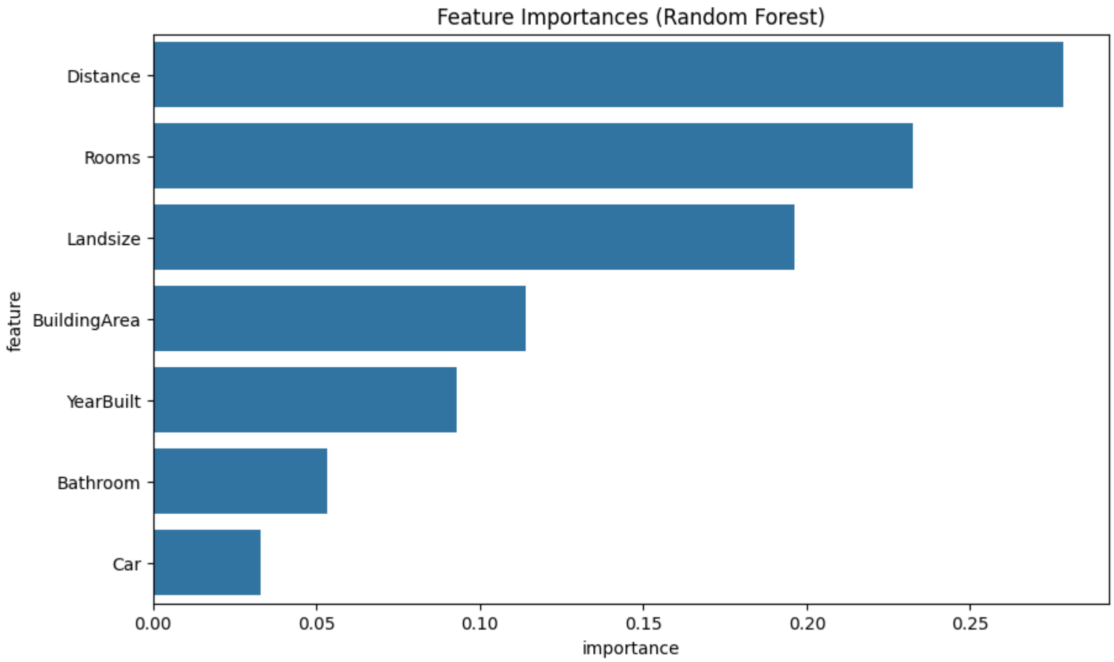

Residual Plot - Decision Tree
1e6
3
2
1
0
−1
−2
−3
Residuals
0
1
2
3
4
5
Predicted values
1e6



2.7 7. Feature Importance

We'll examine feature importance for the Random Forest model to understand which features are most influential in the prediction.

[ ]:
```
feature_importance = pd.DataFrame({
    'feature': features,
    'importance': rf_model.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Feature Importances (Random Forest)')
plt.show()
```



2.8 8. Summary and Analysis

Based on our analysis, we can draw the following conclusions and make recommendations.

[ ]:
```
print("Summary and Analysis:")
print("1. Linear Regression:")
print("   - Baseline performance")
print("   - May be underfitting if the relationship between features and target␣
↪is complex")

print("\n2. Decision Tree:")
print("   - Better performance on training data, but lower on test data")
print("   - Likely overfitting due to high complexity")
print("   - Consider pruning or setting max_depth to reduce overfitting")

print("\n3. Random Forest:")
print("   - Best overall performance")
print("   - Less prone to overfitting compared to Decision Tree")
print("   - Good balance between training and test performance")

print("\nRecommendations:")
print("1. Feature engineering: Create new features or transform existing ones␣
↪to capture non-linear relationships")
print("2. Hyperparameter tuning: Use techniques like GridSearchCV to find␣
↪optimal parameters")
```

```python
print("3. Regularization: Apply regularization to Linear Regression (Lasso,␣
↪Ridge, or Elastic Net)")
print("4. Ensemble methods: Explore other ensemble methods like Gradient␣
↪Boosting")
print("5. Cross-validation: Use k-fold cross-validation for more robust␣
↪performance estimation")
print("6. Handle outliers: Investigate and potentially remove or transform␣
↪outliers in the dataset")
```

```
Summary and Analysis:
1. Linear Regression:
   - Baseline performance
   - May be underfitting if the relationship between features and target is
complex

2. Decision Tree:
   - Better performance on training data, but lower on test data
   - Likely overfitting due to high complexity
   - Consider pruning or setting max_depth to reduce overfitting

3. Random Forest:
   - Best overall performance
   - Less prone to overfitting compared to Decision Tree
   - Good balance between training and test performance

Recommendations:
1. Feature engineering: Create new features or transform existing ones to
capture non-linear relationships
2. Hyperparameter tuning: Use techniques like GridSearchCV to find optimal
parameters
3. Regularization: Apply regularization to Linear Regression (Lasso, Ridge, or
Elastic Net)
4. Ensemble methods: Explore other ensemble methods like Gradient Boosting
5. Cross-validation: Use k-fold cross-validation for more robust performance
estimation
6. Handle outliers: Investigate and potentially remove or transform outliers in
the dataset
```