Team Members

Göktuğ KURNAZ - goktugkurnaz@posta.mu.edu.tr Model Developer

Berkay DURMUŞ - berkaydurmus@posta.mu.edu.tr Data Visualition

Reyhan DUYGU - reyhanduygu@posta.mu.edu.tr Data Scraping

Goal/ Motivation

Increasing population and economic growth cause a continuous increase in housing markets. Migration to cities and the perception of housing as an investment instrument are among the factors that increase the demand for housing. In this context, determining housing prices is an important issue. We aim to predict house prices using data obtained from property platforms such as **emlakjet.com** . This study aims to support decision-making processes in the property market by providing more accurate and reliable price forecasts to home buyers and sellers.

Methods - What are collection, processing, machine learning algorithms and their parameterization, performance evaluation …steps?

Data Source

The dataset used in this project is collected from **emlakjet.com**. This file contains real estate listings with various features such as price, city, town, neighborhood, total square meters, number of rooms, number of floor, floor of home, heating and whether credit is accepted."

Data Loading

The data is loaded into a pandas DataFrame for further analysis and preprocessing using pd.read_csv() as shown in Figure 1.

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt

[2]: # Uploading csv
     df = pd.read_csv('C:\\Users\\sasas\\OneDrive\\Desktop\\ilanlarson3.csv')

[3]: #First 10 rows of data
     df.head(10)
```

Figure 1. Data Loading

Data Cleaning

Missing Values: The dataset is checked for missing values using df.isnull().sum() .

Price Column: Dots are removed, and the column is converted to numeric format using df['Price'].str.replace('.', '').astype(float) as shown in Figure 2.

```
[7]: # Remove dots and convert to numeric format
     df['Price'] = df['Price'].str.replace('.', '').astype(float)
```

Figure 2. Data Cleaning

Removal of Outliers

Values in the price that are unlikely to occur in real life were removed.

Districts with fewer than 5 observations and neighborhoods with fewer than 2 observations were removed.

Values in square meters that are unlikely to occur in real life were removed.

Abnormal values were detected and removed in the number of rooms and floor of the house columns

Price (shown in Figure 3)

```
# Get unique values
unique_values = df['Price'].unique()

anormal_deger = []


for price in unique_values:
    if price < 1100000 or price > 15000000:
        anormal_deger.append(price)


# Remove rows containing abnormal values ••from the data set
df = df[~df['Price'].isin(anormal_deger)]

# Check updated dataset
unique_values = df['Price'].unique()
```

Figure 3. Removing Price Outliers

Total Square of Meter (shown in Figure 4)

```
anormal_deger = []
for meter in unique_values:


    if meter < 100 or meter > 500 :
        anormal_deger.append(meter)


# Removing rows containing abnormal values ••from the data set
df = df[~df['Total Square of Meter'].isin(anormal_deger)]

# Checking updated dataset
unique_values = df['Total Square of Meter'].unique()
print(unique_values)
```

Figure 4. Removing Total Square of Meters Outliers

Number of Room (shown in Figure 5)

```
# List of anormal values ••to use to delete rows containing anormal values
anormal_degerler = [12, 10, 1, 2.5,9 ,5.5, 8, 3.5, 4.5, 7]

# Removing rows containing abnormal values ••from the data set
df = df[~df['Number of room'].isin(anormal_degerler)]

# Checking the updated dataset
unique_values = df['Number of room'].unique()
print(unique_values)


df['Number of room'].value_counts()
```

Figure 5. Removing Number of Room Outliers

Floor Of Home (shown in Figure 6)

```
# List of anormal values ••to use to delete rows containing anormal values
anormal_degerler = ['Müstakil', 'Villa Tipi', '10', '-3', '9', '-2', '8', 'Bahçe Dublex']

# Removing rows containing abnormal values ••from the data set
df = df[~df['Floor of home'].isin(anormal_degerler)]


# Checking the updated dataset
unique_values = df['Floor of home'].unique()
print(unique_values)


df['Floor of home'].value_counts()
```

Figure 6. Removing Floor of Home Outliers

Credi Accepting (shown in Figure 7)

```
# Changing 'Unknown' to 'Yes' in the 'Credit Accepting' column
df['Credi Accepting'] = df['Credi Accepting'].replace('Bilinmiyor', 'Evet')

# printing unique values ••and numbers again
unique_values = df['Credi Accepting'].unique()
print(unique_values)
print(df['Credi Accepting'].value_counts())
```

Figure 7. Removing Credi Accepting Outliers

Categorical Encoding

Floor Category (shown in Figure 8):

```python
# Function that categorizes floors
def categorize_floor(floor):
    try:
        floor = int(floor)
        if floor < 0:
            return 'Zemin Altı'
        elif floor == 0:
            return 'Giriş Katı'
        elif floor <= 2:
            return 'Alt Katlar'
        elif floor <= 5:
            return 'Orta Katlar'
        else:
            return 'Üst Katlar'
    except ValueError:
        return 'Özel Değer'

# Categorizing the 'Floor of home' column
df['Floor Category'] = df['Floor of home'].apply(categorize_floor)

# Check for changes
print(df['Floor Category'].value_counts())

# Converting 'Floor Category' column with one-hot encoding
df = pd.get_dummies(df, columns=['Floor Category'], prefix='Floor')

# View the updated data frame
print(df.head())
```

Figure 8. Categorizing Floor

Label Encoding

To convert categorical variables into numerical format, the LabelEncoder from the Scikit-learn library was utilized. Categorical variables such as City, District, Neighborhood, Floor of the House, Credit Acceptance Status, and Number of Rooms were transformed into numerical values using the le.fit_transform method as shown in Figure 9.

```python
df["City"] = le.fit_transform(df.City)
df["Neighbourhood"] = le.fit_transform(df.Neighbourhood)
df["Floor of home"] = le.fit_transform(df["Floor of home"])
df["Credi Accepting"] = le.fit_transform(df["Credi Accepting"])
df["Number of room"] = le.fit_transform(df["Number of room"])
```

Figure 9. Label Encoding

One-Hot Encoding

Some categorical variables may contain non-ordinal categories. In such cases, representing each category as a distinct feature can enhance the model's performance. We applied one-hot encoding using **pd.get_dummies()** for features like Floor Category, City, Town, Neighbourhood, Credit Acceptance, and Kombi Doğalgaz Heating as shown in Figure 10.

```
# Encode categorical variables
data = pd.get_dummies(df, columns=['City', 'Town', 'Neighbourhood', 'Credi Accepting', 'Kombi Doğalgaz Heating'], drop_first=True)
```

Figure 10. One-Hot Encoding

.Feature Selection

To ensure accurate and meaningful predictions, appropriate features were selected. The chosen features include Total Square of Meter, Number of Rooms, Number of Floors, Floor of Home, and all one-hot encoded categorical variables as shown in Figure 11. These features represent the data used by the model for making predictions.

```
# Feature selection
features = ['Total Square of Meter', 'Number of room', 'Number of floor', 'Floor of home'] + [col for col in data.columns if col.startswith(('City', 'Town
X = data[features]
y = data['Price']
```

Figure 11. Feature Selection

Machine Learning Algorithm and Parameterization:

Train-Test Split

To evaluate the model's performance on unseen data and prevent overfitting, the dataset was split into 80% training and 20% testing sets using train_test_split() with a random state of 50 to ensure reproducibility as shown in Figure 12.

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=50)
```

Figure 12. Train-Test Split

Feature Scaling

To ensure that all features contribute equally to the model's performance, feature scaling was applied using StandardScaler from the Scikit-learn library, which standardizes the features by removing the mean and scaling to unit variance as shown in Figure 13.

```
# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 13. Feature Scaling

Model Training

To build a robust model capable of effectively handling both numerical and categorical data, a Random Forest Regressor with 210 estimators and a random state of 50 for reproducibility was selected as shown in Figure 14.

```
# Model training
model = RandomForestRegressor(n_estimators=210, random_state=50)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)
```

Figure 14. Model Training

First Project (Linear Regression and XGBoost)

Linear Regression

Model: Linear Regression

Performance: Mean Absolute Percentage Error (MAPE): 18.71%

Advantages: Simple and fast. Useful for initial data analysis and modeling.

Disadvantages: Accuracy is low in more complex datasets.

```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# MAPE calculating
mape = mean_absolute_percentage_error(y_test, y_pred)
print(f'Mean Absolute Percentage Error (MAPE): {mape:.2f}%')
```

✓ 0.0s

```
Mean Absolute Percentage Error (MAPE): 18.71%
```

```
# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')

# Save actual and predicted prices to a new CSV file
results = pd.DataFrame({'Actual Price': y_test, 'Predicted Price': y_pred})

# CSV dosyasına yazma
results.to_csv('y_test_y_pred_results_linear_regression.csv', index=False)
print("Results saved to y_test_y_pred_results_linear_regression.csv")
```

✓ 0.0s

```
Mean Absolute Error: 633554.7240512016
Results saved to y_test_y_pred_results_linear_regression.csv
```

```
from sklearn.metrics import r2_score
# Calculating R-square
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

✓ 0.0s

```
R-squared: 0.2456726378367714
```

Figure 15. Linear Regression Results

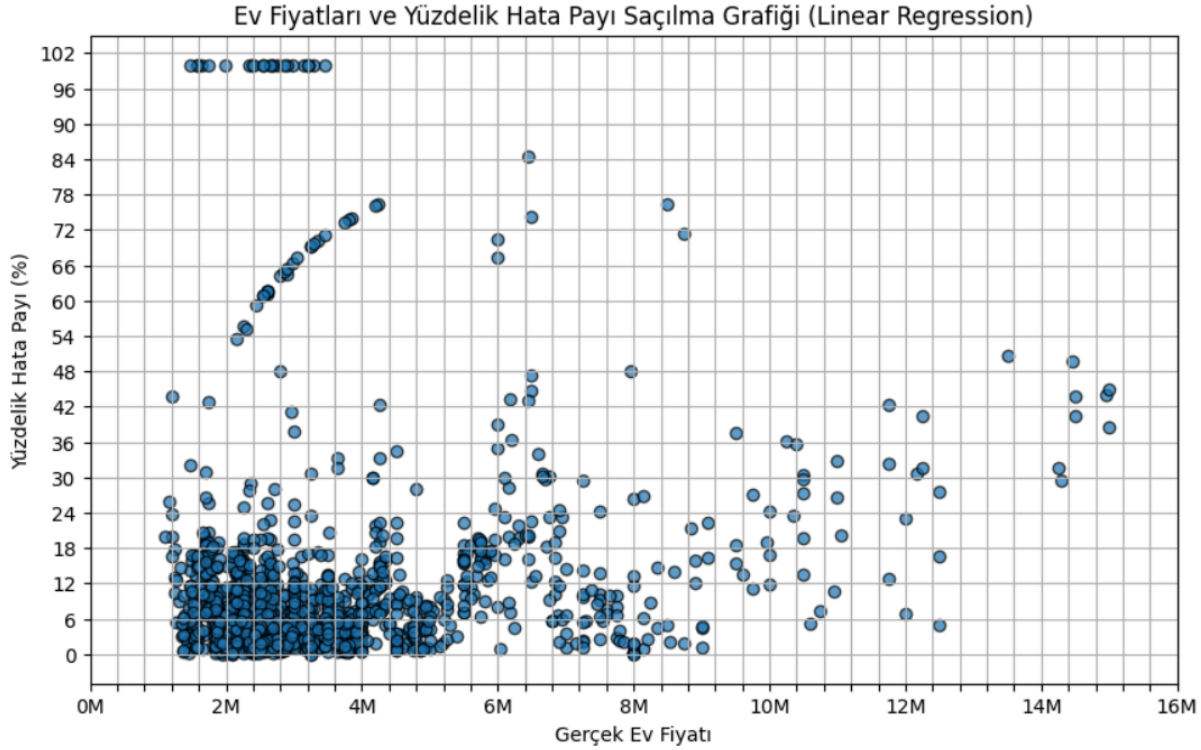

Figure 16. The Price vs. Percentage Error Scatter Plot (Linear Regression)

Price Range	<10% Error	<20% Error	<30% Error	<40% Error	<50% Error
----------Linear Regression----------					
0-2M	64.50%	92.64%	95.67%	96.54%	97.40%
2-4M	75.33%	92.83%	94.17%	94.65%	94.90%
4-6M	59.18%	92.35%	97.45%	98.47%	98.98%
6-8M	41.38%	65.52%	81.61%	89.66%	95.40%
8-10M	47.22%	77.78%	91.67%	94.44%	94.44%
10-12M	14.29%	42.86%	71.43%	95.24%	100.00%
12-14M	12.50%	25.00%	50.00%	75.00%	87.50%

Figure 17. Error Comparison Table (Linear Regression)

XGBoost

Model: XGBRegressor

Performance: Improved with hyperparameter optimization, but detailed performance results are not fully specified for the first version. For second version: Mean Absolute Percentage Error (MAPE): 7.19%

Advantages: Powerful and high accuracy. Effective in large and complex datasets.

Disadvantages: Complex and requires more computational resources.

```python
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
def select_columns(df):
    X = df.drop(columns=['Price'])
    y = df['Price']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    model = XGBRegressor()
    model.fit(X_train, y_train)

    # Özellik önemlerini alma
    importances = model.feature_importances_
    indices = np.argsort(importances)[::-1]

    avg_imp = sum(importances) / len(importances)

    selected_columns = []
    for f in range(X_train.shape[1]):
        if importances[indices[f]] > 0:
            selected_columns.append(X_train.columns[indices[f]])
    return selected_columns
```

Figure 18. XGBoost (First Version)

```python
import pandas as pd
import xgboost as xgb
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
from sklearn import model_selection

print(len(df.columns))
selected_columns = select_columns(df)
selected_columns.append('Price')
df = df[selected_columns]

print(len(df.columns))
X = df.drop(columns=['Price'])
y = df['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# GridSearchCV
params = {"colsample_bytree":[0.2,0.5,0.8],
          "learning_rate":[0.1,0.01],
          "max_depth":[100,500,1000],
          "n_estimators":[100,500,2000]}


xgb = XGBRegressor()


grid = GridSearchCV(xgb, params, cv = 3, verbose = 2)

grid.fit(X_train, y_train)

print(grid.best_params_)

xgb1 = XGBRegressor(colsample_bytree = grid.best_params_['colsample_bytree'], learning_rate = grid.best_params_['learning_rate'], max_depth = grid.best_p

model_xgb = xgb1.fit(X_train, y_train)

y_pred = model_xgb.predict(X_test)

y_dif = abs(y_test - y_pred)
y_dif = y_dif / y_test
print(sum(y_dif) / len(y_dif))
```

Figure 19. XGBoost (First Version Continue)

```python
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler



# Encode categorical variables
data = pd.get_dummies(df, columns=['City', 'Town', 'Neighbourhood', 'Credi Accepting', 'Kombi Doğalgaz Heating'], drop_first=True)

# Feature selection
features = ['Total Square of Meter', 'Number of room', 'Number of floor', 'Floor of home'] + [col for col in data.columns if col.startswith(
    ('City', 'Town', 'Neighbourhood', 'Credi Accepting', 'Kombi Doğalgaz Heating'))]
X = data[features]
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=50)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# XGBoost model training
xgb_model = XGBRegressor(n_estimators=210, random_state=50)
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred = xgb_model.predict(X_test)
```

Figure 20. XGBoost (Last Version)

```python
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# MAPE calculating
mape = mean_absolute_percentage_error(y_test, y_pred)
print(f'Mean Absolute Percentage Error (MAPE): {mape:.2f}%')
```

✓ 0.0s

```
Mean Absolute Percentage Error (MAPE): 7.19%
```

```python
# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')

# Save actual and predicted prices to a new CSV file
results = pd.DataFrame({'Actual Price': y_test, 'Predicted Price': y_pred})

# CSV dosyasına yazma
results.to_csv('y_test_y_pred_results_xgboost.csv', index=False)
print("Results saved to y_test_y_pred_results_xgboost.csv")
```

✓ 0.0s

```
Mean Absolute Error: 343439.2495567376
Results saved to y_test_y_pred_results_xgboost.csv
```

```python
from sklearn.metrics import r2_score
# Calculating R-square
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

✓ 0.0s

```
R-squared: 0.845317373669032
```

Figure 21. XGBoost Results

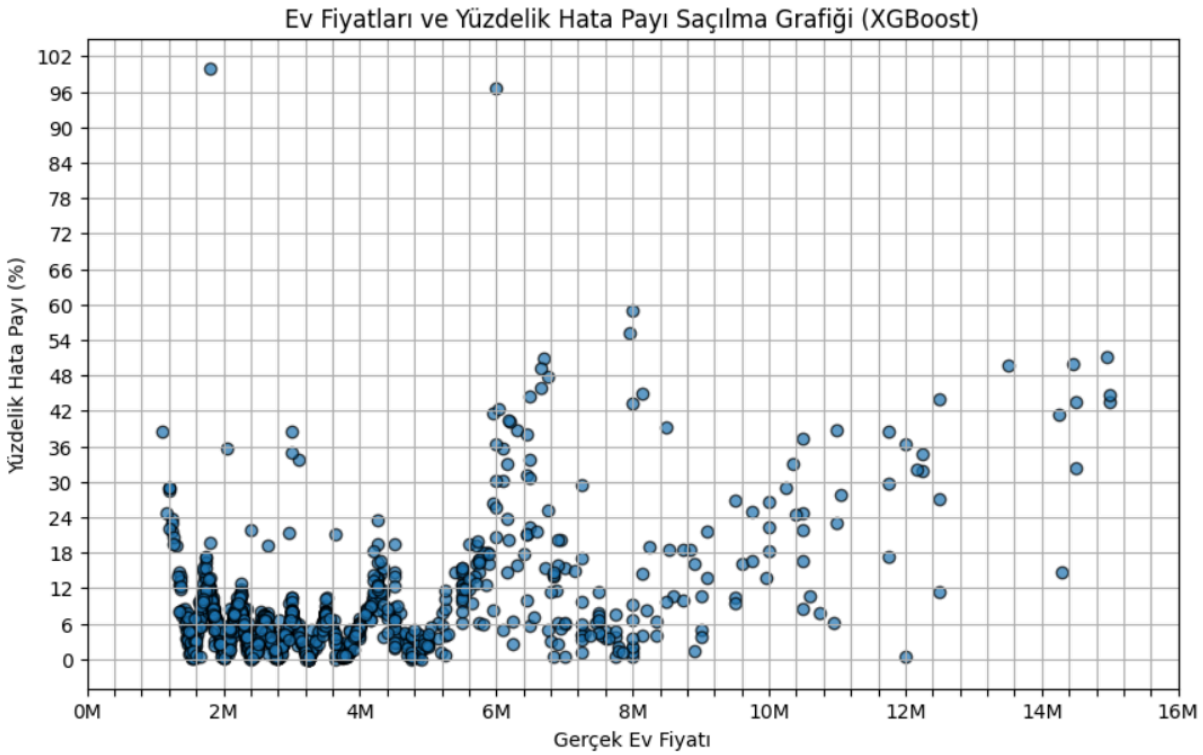

Figure 22. The Price vs. Percentage Error Scatter Plot (XGBoost)

Price Range	<10% Error	<20% Error	<30% Error	<40% Error	<50% Error
		XGBoost			
0-2M	74.46%	95.24%	99.13%	99.57%	99.57%
2-4M	97.08%	99.15%	99.51%	100.00%	100.00%
4-6M	62.76%	98.47%	99.49%	99.49%	100.00%
6-8M	47.13%	63.22%	77.01%	88.51%	96.55%
8-10M	44.44%	80.56%	88.89%	91.67%	97.22%
10-12M	19.05%	38.10%	80.95%	100.00%	100.00%
12-14M	0.00%	12.50%	25.00%	75.00%	100.00%

Figure 23. Error Comparison (XGBoost)

Latest Project (Random Forest)

Random Forest

Model: RandomForestRegressor

Performance: Mean Absolute Percentage Error (MAPE): 7.05%, Mean Absolute Error (MAE): 337,268, R-squared: 0.857

Advantages: Works well with both numerical and categorical data. Provides high accuracy. Prevents overfitting.

Disadvantages: The complexity and computational cost of the model can be high.

```python
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

# MAPE calculating
mape = mean_absolute_percentage_error(y_test, y_pred)
print(f'Mean Absolute Percentage Error (MAPE): {mape:.2f}%')
```

✓ 0.0s

```
Mean Absolute Percentage Error (MAPE): 7.05%
```

```python
# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
print(f'Mean Absolute Error: {mae}')

# Save actual and predicted prices to a new CSV file
results = pd.DataFrame({'Actual Price': y_test, 'Predicted Price': y_pred})

# CSV dosyasına yazma
results.to_csv('y_test_y_pred_results_random_forest.csv', index=False)
print("Results saved to y_test_y_pred_results_random_forest.csv")
```

✓ 0.0s

```
Mean Absolute Error: 337268.92979957437
Results saved to y_test_y_pred_results_random_forest.csv
```

```python
from sklearn.metrics import r2_score
# Calculating R-square
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

✓ 0.0s

```
R-squared: 0.8577197327139315
```

Figure 24. Random Forest

Evaluation and Conclusion

First Project

Linear Regression: Simple and fast but has low performance on complex datasets.

XGBoost: Higher accuracy and performance. Improved with hyperparameter optimization.

Latest Project

Random Forest: Provided the best results. The Mean Absolute Percentage Error (MAPE) is 7.05%, and the R-squared value of 0.857 indicates that the model is robust and accurate.

Recommendations

First Project

Good as an initial attempt. Starting with simple models and then moving to more complex ones is a valid approach.

Latest Project

The Random Forest model showed the best performance for our data. It can be confidently used in business or research projects.

Visualization and Interpretation of Results for Best Model

Histogram

The Percentage Error Histogram shows that most errors are low, indicating that the model generally makes accurate predictions as shown in Figure 25.

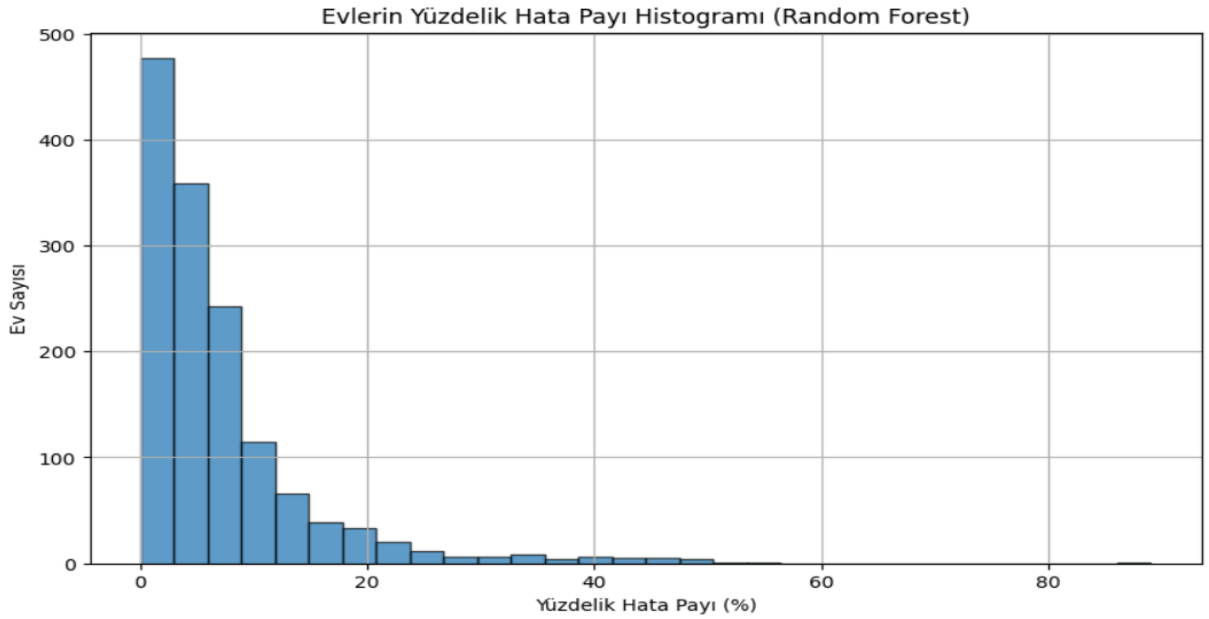

Figure 25. Percentage Error Histogram of Houses (Random Forest)

Scatter Plot

The Price vs. Percentage Error Scatter Plot shows that the error rate increases for higher-priced houses as shown in Figure 26.

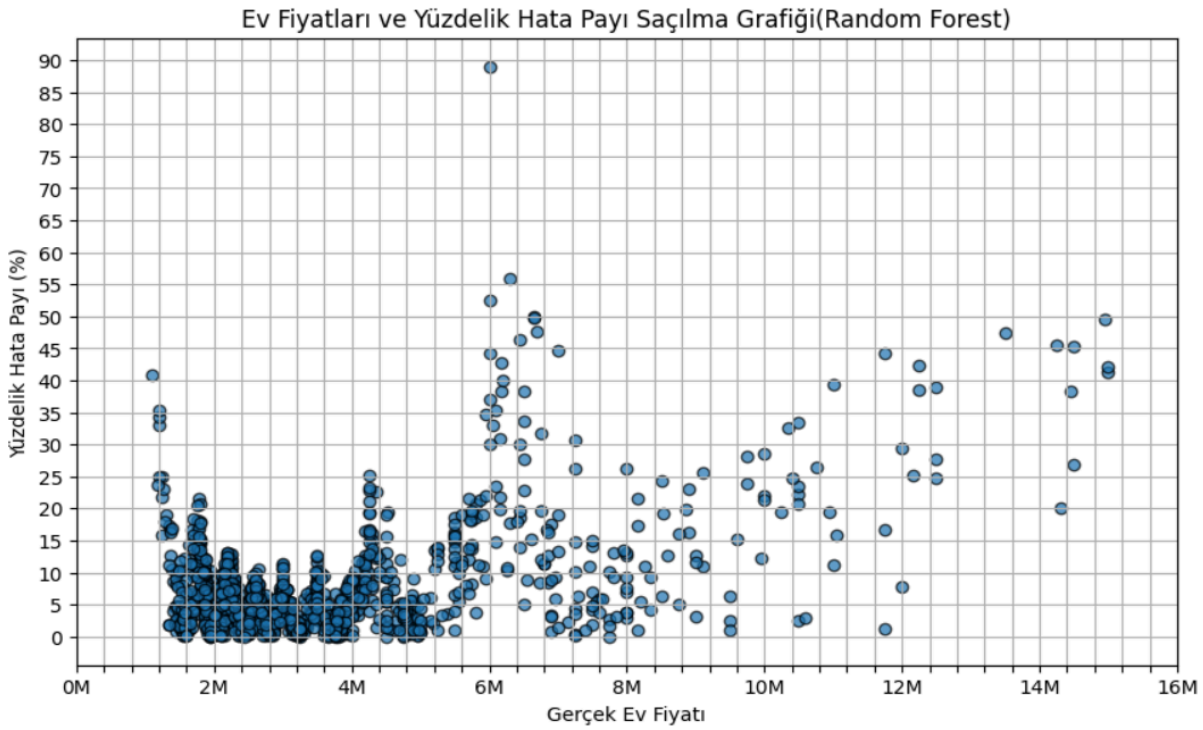

Figure 26. The Price vs. Percentage Error Scatter Plot (Random Forest)

Error Table

The table where we compare the error percentages in order to see the evaluation statistics of houses according to their price ranges is shown in Figure 27.

Price Range	<10% Error	<20% Error	<30% Error	<40% Error	<50% Error
----------Random Forest----------					
0-2M	69.70%	94.81%	98.27%	99.57%	100.00%
2-4M	96.11%	100.00%	100.00%	100.00%	100.00%
4-6M	62.24%	94.90%	99.49%	100.00%	100.00%
6-8M	35.63%	68.97%	77.01%	88.51%	96.55%
8-10M	44.44%	80.56%	100.00%	100.00%	100.00%
10-12M	19.05%	42.86%	80.95%	95.24%	100.00%
12-14M	0.00%	0.00%	50.00%	75.00%	100.00%

Figure 27. Error Comparison Table (Random Forest)

Conclusion

Objectives Met

Our goal was to reduce the error rate below 10%, and we successfully achieved this objective.

Effort and Time Investment

We dedicated as much time as possible to our project. However, machine learning projects often require a lot of trial and error. With more trials, we could have achieved better results, but this also requires more time.

Current Result

Despite the challenges we faced, the result we obtained is promising. There is always room for improvement, but we are satisfied with the current outcome.

Value Proposition

The results of our project demonstrate a strong potential for practical application. This level of accuracy indicates that the model can be effectively used in business and research contexts. This accuracy level could support the establishment of a startup company or further investment into the project. However, due to the intense competition in this field, continuous improvement and further validation are necessary.

URLS

You can find all codes from these repos (scrapy things and machine learning codes.)
https://github.com/missreyyo/HouseFinder

https://github.com/goktugkurnaz/Machine_learning

https://github.com/U190709047/AML_PROJECT