RealEstateAI

Puts a price on your house...

Team Members

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

Berkay Durmuş- berkaydurmuş@posta.mu.edu.tr Data Visualation

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

Goal/Motivation

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.

Data Scraping

How Did We Scrap Our Data?

scrapy > spiders > emlajjet.py > ...

```
import scrapy


class IlanSpider(scrapy.Spider):
    name = 'ilan'
    city_name = 'istanbul'
    start_urls = ['https://www.emlakjet.com/satilik-daire/{city_name}/']
    page_number = 1
    seen_urls = set()

    def parse(self, response):
        ilan_urls = []


        for i in range(1,40):
            ilan_urls.extend(response.css(f'#listing-search-wrapper > div:nth-child({i}) > a::attr(href)').extract())

        for ilan_url in ilan_urls:
            if ilan_url not in self.seen_urls:
                self.seen_urls.add(ilan_url)
                yield scrapy.Request(url=response.urljoin(ilan_url), callback=self.parse_ilan)

        if self.page_number > 70:
            return

        self.page_number += 1
        next_page_url = f"https://www.emlakjet.com/satilik-daire/{self.city_name}/{self.page_number}/"
        yield scrapy.Request(url=next_page_url, callback=self.parse)

    def parse_ilan(self, response):
        ilan_informations = {}
        #ilan no
        ilan_no = response.css('#bilgiler > div > div._2VNNor._2eyo_P > div > div._3tH_Nw > div:nth-child(1) > div:nth-child(1) > div:nth
        if ilan_no:
            ilan_no = ilan_no.strip()
        ilan_informations['ilan no'] = ilan_no
        #price
```

Data Science

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

# 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()
```

Removal of Outliers

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

```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())
```

Categorize

We categorized our floors for performance.

Data Processing

Label Encoding

```
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"])
```

To convert categorical variables into numerical format, the LabelEncoder from the Scikit-learn library was utilized.

One-Hot Encoding

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

Some categorical variables may contain non-ordinal categories. In such cases, representing each category as a distinct feature can enhance the model's performance.

Feature Selection

```
# 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']
```

To ensure accurate and meaningful predictions, appropriate features were selected.

Machine Learning Algorithm and Parameterization

Train-Test Split

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

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.

Feature Scaling

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

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.

Model Training

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

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

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.

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.

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%

Linear Regression

```
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

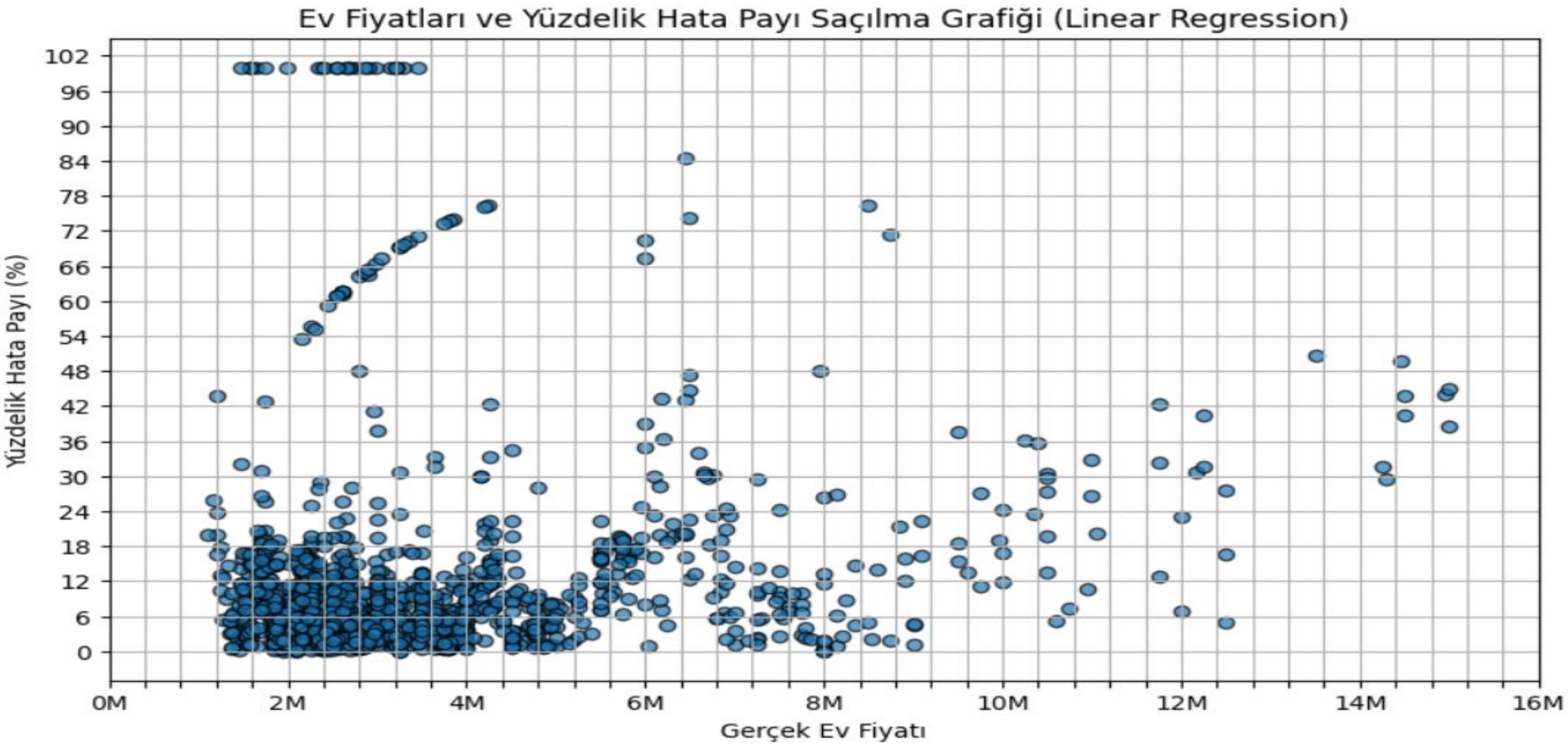
```
Mean Absolute Error: 633554.7240512016
Results saved to y_test_y_pred_results_linear_regression.csv
```

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

✓ 0.0s

```
R-squared: 0.24567263783677 14
```

Linear Regression



XGBoost

Model: XGBRegressor

Performance: Improved with hyperparameter optimization, but detailed performance results are not fully specified. 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.

```
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
```

XGBoost (continue)

```
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))
```

Linear

XGBoost

```
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%
```

```
# 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

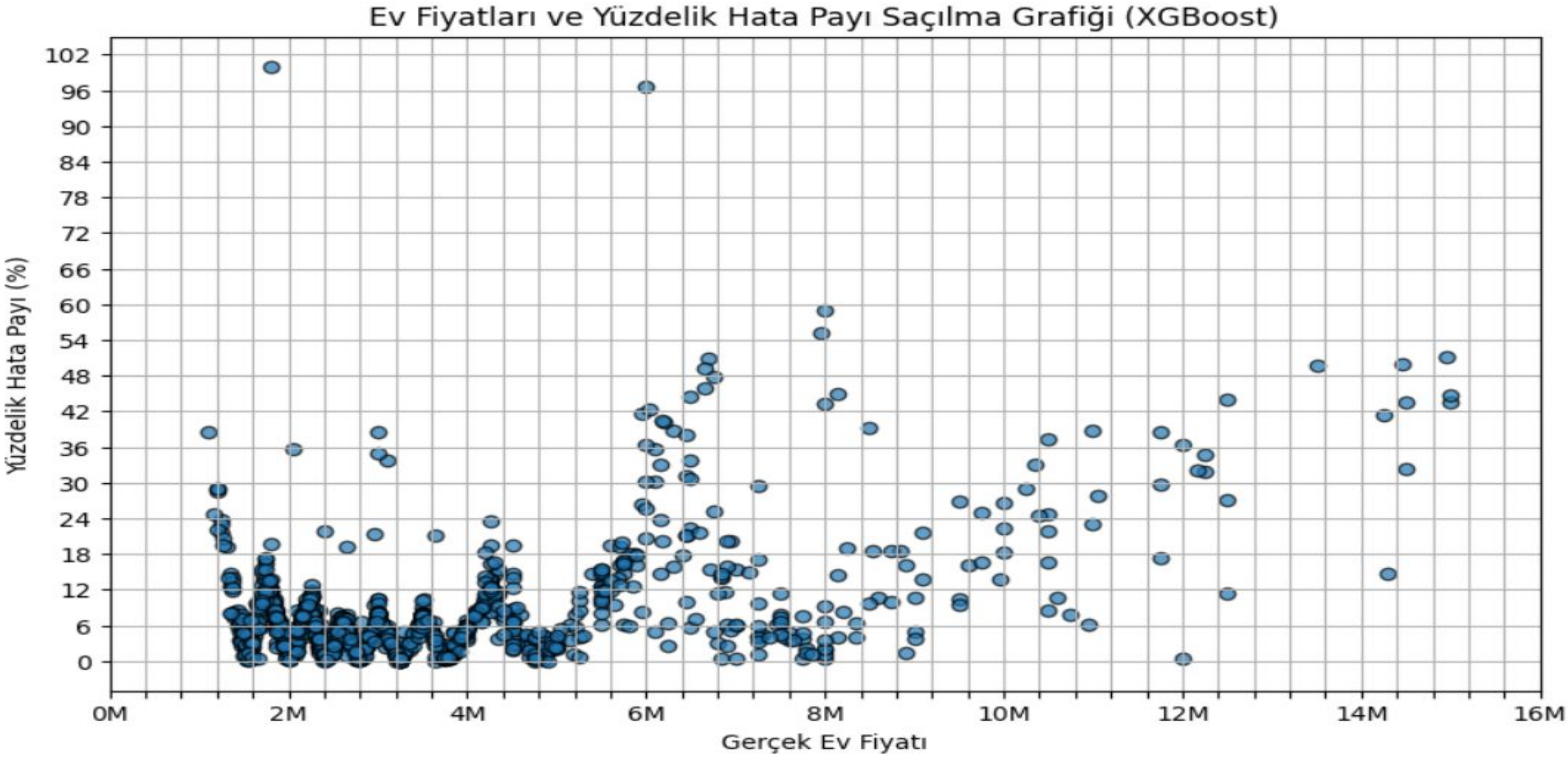
```
Mean Absolute Error: 343439.2495567376
Results saved to y_test_y_pred_results_xgboost.csv
```

```
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
```

XGBoost


Ev Fiyatları ve Yüzdelik Hata Payı Saçılma Grafiği (XGBoost)
Yüzdelik Hata Payı (%)
102
96
90
84
78
72
66
60
54
48
42
36
30
24
18
12
6
0
0M
2M
4M
6M
8M
10M
12M
14M
16M
Gerçek Ev Fiyatı

Latest Project (Random Forest)

Random Forest

Model: Random Forest Regressor

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.

Random Forest Scores

```
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%
```

```
# 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
```

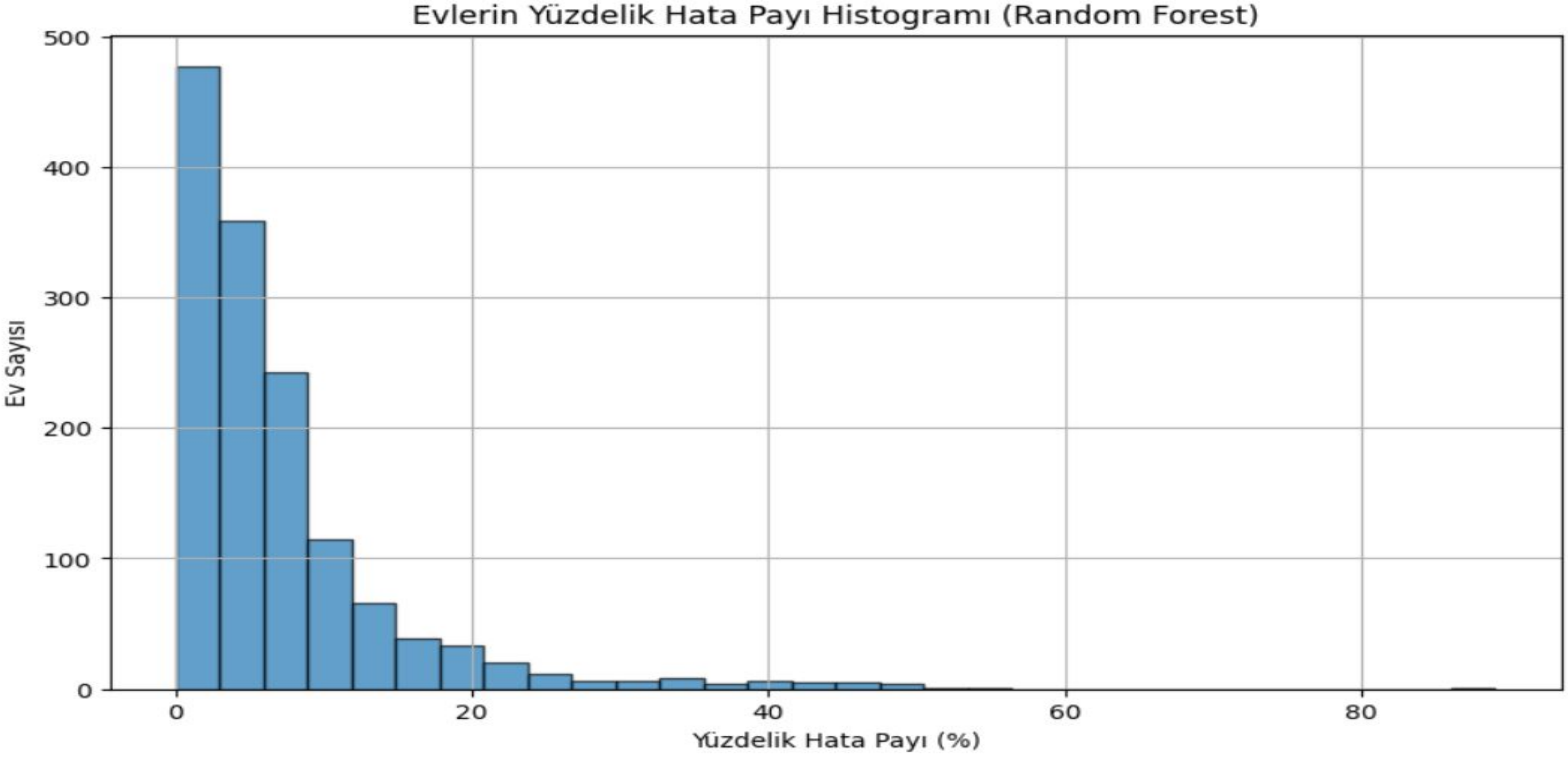
```
from sklearn.metrics import r2_score
# Calculating R-square
r2 = r2_score(y_test, y_pred)
print("R-squared:", r2)
```

✓ 0.0s

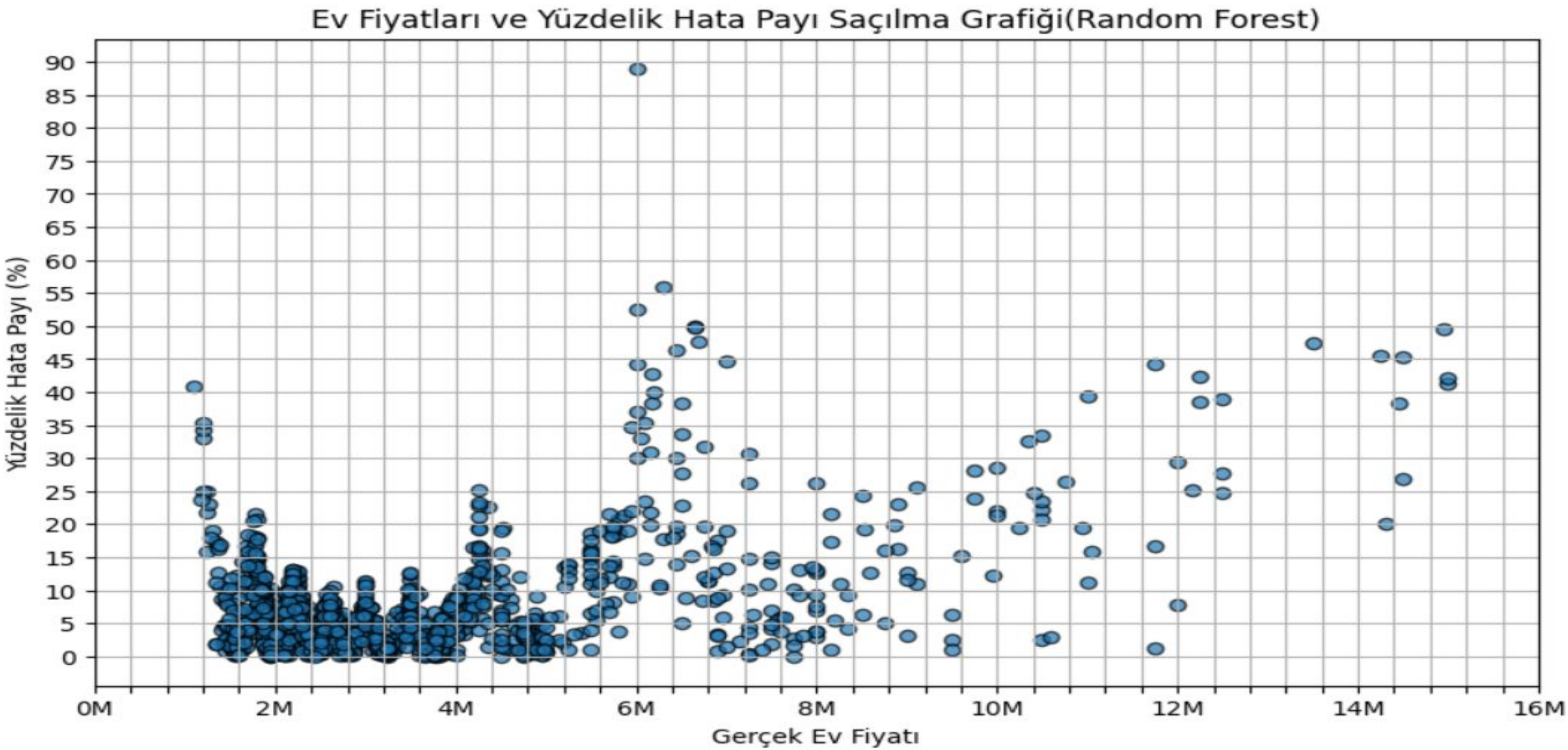
```
R-squared: 0.8577197327139315
```

Visualization and Interpretation of Results

The Percentage Error Histogram



The Price vs. Percentage Error Scatter Plot



Error Comparison

Price Range	MAE
----------Random Forest-------------	
0-2M	126849.56
2-4M	112885.57
4-6M	420763.51
6-8M	1219794.80
8-10M	1068153.72
10-12M	2224506.76
12-14M	4289075.71
14-16M	5658692.74

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%

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.

Thank you