```
from google.colab import drive
drive.mount('/content/drive/')
```

```
import os
os.chdir('/content/drive/My Drive/MakineSon')
!pwd
```

```
#kütüphanelerin tanımlanması

# Pandas kütüphanesini veri manipülasyonu için kullanıyoruz.
import pandas as pd
import xgboost as xgb
# Veri görselleştirmesi için Seaborn kütüphanesini kullanıyoruz.
import seaborn as sns

# Verileri görselleştirmek için Matplotlib kütüphanesini
kullanıyoruz.
import matplotlib.pyplot as plt

# Matematiksel işlemler yapmak için NumPy kütüphanesini
kullanıyoruz.
import numpy as np

# Regresyon modelleri için gerekli olan XGBoost kütüphanesini
kullanıyoruz.
from xgboost import XGBRegressor

# Regresyon modelleri için gerekli olan LightGBM kütüphanesini
kullanıyoruz.
from lightgbm import LGBMRegressor

# Regresyon modelleri için gerekli olan RandomForestRegressor'ı
kullanıyoruz.
from sklearn.ensemble import RandomForestRegressor

# Regresyon modelleri için gerekli olan
GradientBoostingRegressor'ı kullanıyoruz.
from sklearn.ensemble import GradientBoostingRegressor

from sklearn.preprocessing import StandardScaler

# Doğrusal regresyon modeli için gerekli olan LinearRegression'ı
kullanıyoruz.
from sklearn.linear_model import LinearRegression
```

```
# Veri setini eğitim ve test olarak bölme ve performans
metriklerini hesaplamak için kullanıyoruz.
from sklearn.model_selection import cross_val_score,
cross_validate, train_test_split, GridSearchCV


# Performans metriklerini hesaplamak için kullanıyoruz.
from sklearn.metrics import mean_absolute_error,
mean_squared_error, accuracy_score, r2_score
```

```
df_2 = pd.read_csv(r"istanbulson2.csv")
df = df_2.copy()
```

```
# "fiyat" sütununu bağımlı değişken olarak (y) belirleyin
y = df["fiyat"]

# "fiyat" sütununu hariç tutarak geri kalan tüm sütunları
bağımsız değişkenler (X) olarak kullanın
X = df.drop(["fiyat"], axis=1)
```

```
# Veri kümesini eğitim ve test setlerine böler
# X: bağımsız değişkenler, y: bağımlı değişkenler
# test_size: test setinin oranı (0.25 = %25), random_state: veri
setinin karıştırılması için kullanılan rastgele durumun
sabitlenmesi
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.25, random_state=144)
```

```
# XGBoost algoritması için hiperparametrelerin tanımlanması
params = {
    'subsample': 0.8,
    'n_estimators': 2500,
    'max_depth': 7,
    'learning_rate': 0.03,
    'gamma': 0,
    'colsample_bytree': 0.5
}
```

```
# XGBoost Regresyon modeli oluşturma
xgb_model = XGBRegressor()
```

```
# Grid Search Cross Validation ile en iyi hiperparametreleri
bulma
grid = GridSearchCV(xgb_model, params, cv=10, n_jobs=-1,
verbose=2)
grid.fit(X_train, y_train)

# En iyi hiperparametreleri yazdırma
print(grid.best_params_)
```

```
# Belirli hiperparametre değerleriyle XGBoost Regresyon modeli
oluşturma
xgb1 = XGBRegressor(colsample_bytree=0.5, learning_rate=0.09,
max_depth=4, n_estimators=2000)
```

```
# XGBoost Regresyon modelini eğitme
model_xgb = xgb1.fit(X_train, y_train)
```

```
# Test seti üzerinde tahminler yapma ve belirtilen aralıktaki
tahminleri döndürme
predictions = model_xgb.predict(X_test)[15:20]
```

```
# Test setindeki gerçek hedef değerlerini döndürme
 y_test[15:20]
```

```
# Eğitilmiş XGBoost Regresyon modelinin test seti üzerindeki
doğruluk skorunu hesaplama
model_xgb.score(X_test, y_test)
```

```
# Eğitilmiş XGBoost Regresyon modelinin eğitim seti üzerindeki
doğruluk skorunu hesaplama
model_xgb.score(X_train, y_train)
```

```
# Eğitilmiş XGBoost Regresyon modelinin çapraz doğrulama
yöntemiyle ortalama kök ortalama kare hatasını hesaplama
np.sqrt(-1 * (cross_val_score(model_xgb, X_test, y_test, cv=10,
scoring='neg_mean_squared_error'))).mean()
```

```
# Özelliklerin önem derecelerini içeren bir veri çerçevesi
oluşturma
importance = pd.DataFrame({"Importance":
model_xgb.feature_importances_}, index=X_train.columns)
importance
```

```
# Öncelikle matplotlib kütüphanesinden pyplot modülünü içe
aktarıyoruz
from matplotlib import pyplot as plt

# Özelliklerin önem derecelerini içeren DataFrame'i çizgi grafiği
olarak görselleştiriyoruz
importance['Importance'].plot(kind='line', figsize=(8, 4),
title='Importance')

# Grafiğin sağ ve üst kenarlarının görünürlüğünü kaldırıyoruz
plt.gca().spines[['top', 'right']].set_visible(False)
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score

# Test seti üzerinde tahmin yapma
y_pred = model_xgb.predict(X_test)

# R^2 skoru hesaplama
r2 = r2_score(y_test, y_pred)

# Ortalama mutlak hata (MAE) hesaplama
mae = mean_absolute_error(y_test, y_pred)

# Ortalama kare hatası (MSE) hesaplama
mse = mean_squared_error(y_test, y_pred)

# Kök ortalama kare hatası (RMSE) hesaplama
rmse = np.sqrt(mse)

# Sonuçları yazdırma
print(f"R² Skoru: {r2:.4f}")
print(f"Ortalama Mutlak Hata (MAE): {mae:.4f}")
print(f"Ortalama Kare Hatası (MSE): {mse:.4f}")
print(f"Kök Ortalama Kare Hatası (RMSE): {rmse:.4f}")

# Gerçek vs Tahmin Değerleri Grafiği
```

```
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolors=(0, 0, 0))
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'k--', lw=4)
plt.xlabel('Gerçek Değerler')
plt.ylabel('Tahmin Edilen Değerler')
plt.title('Gerçek vs Tahmin Edilen Değerler')
plt.show()

# Hata Dağılım Grafiği
errors = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.hist(errors, bins=50, edgecolor='black')
plt.xlabel('Hata Değerleri')
plt.ylabel('Frekans')
plt.title('Hata Dağılım Grafiği')
plt.show()
```

```
# Sürekli özellikler ve fiyat arasındaki ilişkiyi görselleştirme
continuous_features = ["oda_sayısı", "brüt_metrekare",
"binanın_yaşı", "binanın_kat_sayısı"]

for feature in continuous_features:
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, x=feature, y='fiyat')
    plt.title(f'Fiyat vs {feature}')
    plt.show()
```

```
# Kategorik özellikler ve fiyat arasındaki ilişkiyi kutu grafiği
ile görselleştirme
categorical_features = ["ısıtma_tipi", "krediye_uygunluk",
"yapı_durumu", "eşya_durumu", "site_içerisinde",
                        "tipi", "kullanım_durumu",
"yatırıma_uygunluk", "ilce"]

for feature in categorical_features:
    plt.figure(figsize=(12, 6))
    sns.boxplot(data=df, x=feature, y='fiyat')
    plt.title(f'Fiyat vs {feature}')
    plt.xticks(rotation=45)
    plt.show()
```

```
# Sürekli özellikler arasındaki ve fiyat ile olan korelasyonu
görselleştirme
plt.figure(figsize=(14, 10))
correlation_matrix = df[continuous_features + ['fiyat']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Korelasyon Matrisi')
plt.show()
```

```
# Sürekli özellikler arasındaki ve fiyat ile olan ilişkiyi
çiftler halinde görselleştirme
sns.pairplot(df[continuous_features + ['fiyat']])
plt.show()
```

```
# Sürekli özelliklerin dağılımını görselleştirme
for feature in continuous_features:
    plt.figure(figsize=(10, 6))
    sns.histplot(df[feature], kde=True)
    plt.title(f'{feature} Dağılımı')
    plt.show()
```

```
# Gerçek ve tahmin edilen fiyat değerleri arasındaki ilişkiyi
gösteren scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolors=(0, 0, 0))
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'k--', lw=4)
plt.xlabel('Gerçek Değerler')
plt.ylabel('Tahmin Edilen Değerler')
plt.title('Gerçek vs Tahmin Edilen Fiyat Değerleri')
plt.show()
```

```
# Tahmin hatalarının dağılımını görselleştirme
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
sns.histplot(residuals, kde=True)
plt.title('Tahmin Hataları Dağılımı')
plt.xlabel('Hata Değerleri')
plt.ylabel('Frekans')
plt.show()
```