Kira Tahminleme Sistemi - Teknik Rapor

- Bu çalışmada, konut kiralarının tahmin edilmesine yönelik bir makine öğrenmesi projesi gerçekleştirilmiştir. Gerçek dünya verileri kullanılarak bir regresyon problemi çözülmüş ve farklı algoritmaların performansları karşılaştırılmıştır.

Analiz İçerği

Veri Seti Tanıtımı

Bu veri seti Türkiye'deki farklı şehirlerdeki ev fiyatları hakkında bilgi içerir. Metrekare büyüklüğü, oda sayısı, kat seviyesi, binanın yaşı, ısıtma türü ve bir evin fiyatını etkileyebilecek diğer nitelikler gibi çeşitli özellikleri içerir.

- Net_Metrekare: Evin net metrekare alanı.
- Brüt_Metrekare: Evin brüt metrekare alanı.
- Oda_Sayısı: Evdeki oda sayısı.
- Bulunduğu_Kat: Evin taban seviyesi.

- Eşya_Durumu: Evin eşyalı olup olmadığını gösterir.
- Binanın_Yaşı: Binanın yaşı (örneğin; "0 (Yeni)", "5-10", "21 ve üzeri").
- Isıtma_Tipi: Isıtma türü (örn. "Kombi Doğalgaz", "Klimalı").
- Fiyat: Evin Türk Lirası (TRY) cinsinden fiyatı. (Hedef)
- Şehir: Evin bulunduğu şehir.
- Binanın_Kat_Sayısı: Binanın toplam kat sayısı.
- Kullanım_Durumu: Evin dolu ya da boş olduğunu gösterir.
- Yatırıma_Uygunluk: Evin yatırıma uygun olup olmadığı.
- Site_İçerisinde: Evin bir konut kompleksi içerisinde olup olmadığı.
- Takas: Evin başka bir gayrimenkul ile takas edilip edilemeyeceğini gösterir.
- Krediye_Uygunluk: Evin ipotek için uygun olup olmadığı.
- Tapu_Durumu: Mülkiyet unvanının hukuki durumu (örneğin, "Kat Mülkiyeti").
- Banyo_Sayısı: Evdeki banyo sayısı.

Kaynak: https://www.kaggle.com/datasets/cabbar14ylnce/trkiye-home-price-data

Python Kütüphaneleri

In [42]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
import warnings
warnings.filterwarnings("ignore")
```

Veri Yükleme

In [43]:
```
df=pd.read_csv("home_price.csv") #veri setini yüklüyoruz
```

In [44]:
```
df.head() # ilk 5 verinin gösterimi
```

Out[44]:

	Net_Metrekare	Brüt_Metrekare	Oda_Sayısı	Bulunduğu_Kat	Eşya_Durumu	Binanın_Y
0	120	150.0	4.0	4.Kat	Eşyalı	21 Ve Üz
1	100	125.0	4.0	3.Kat	Boş	
2	89	95.0	3.0	4.Kat	Boş	0 (Ye
3	40	55.0	2.0	6.Kat	Boş	0 (Ye
4	140	150.0	4.0	Düz Giriş (Zemin)	Boş	5-

Veri Seti Analizi

In [45]:
```
df.info() #veri setimizde temel bilgilerini görürüz TYPE(Tipini) ve NON-Null Cou
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20326 entries, 0 to 20325
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Net_Metrekare       20326 non-null  int64  
 1   Brüt_Metrekare      20326 non-null  float64
 2   Oda_Sayısı          20066 non-null  float64
 3   Bulunduğu_Kat       18029 non-null  object 
 4   Eşya_Durumu         12928 non-null  object 
 5   Binanın_Yaşı        20326 non-null  object 
 6   Isıtma_Tipi         20326 non-null  object 
 7   Fiyat               20326 non-null  float64
 8   Şehir               20326 non-null  object 
 9   Binanın_Kat_Sayısı  20326 non-null  int64  
 10  Kullanım_Durumu     20326 non-null  object 
 11  Yatırıma_Uygunluk   14750 non-null  object 
 12  Takas               14431 non-null  object 
 13  Tapu_Durumu         9760 non-null   object 
 14  Banyo_Sayısı        20280 non-null  float64
dtypes: float64(4), int64(2), object(9)
memory usage: 2.3+ MB
```

In [46]:
```
df.describe() #Numerik verilerimizin temel istatiklerini tanımlıyoruz
```

Out[46]:

	Net_Metrekare	Brüt_Metrekare	Oda_Sayısı	Fiyat	Binanın_Kat_Sayısı	I
count	20326.000000	2.032600e+04	20066.00000	2.032600e+04	20326.000000	2
mean	151.964528	1.437683e+09	3.68940	4.649510e+06	5.185969	
std	1137.304466	2.049686e+11	1.11206	7.136120e+07	3.051288	
min	0.000000	2.000000e+00	1.00000	2.000000e+04	1.000000	
25%	90.000000	1.000000e+02	3.00000	1.640000e+06	3.000000	
50%	120.000000	1.350000e+02	4.00000	2.300000e+06	4.000000	
75%	150.000000	1.700000e+02	4.00000	3.300000e+06	6.000000	
max	110000.000000	2.922222e+13	12.00000	7.500000e+09	15.000000	

Eksik Veri Kontrolü

In [47]:
```
df.isnull().sum()
```

Out[47]:
```
Net_Metrekare             0
Brüt_Metrekare            0
Oda_Sayısı              260
Bulunduğu_Kat          2297
Eşya_Durumu            7398
Binanın_Yaşı              0
Isıtma_Tipi               0
Fiyat                     0
Şehir                     0
Binanın_Kat_Sayısı        0
Kullanım_Durumu           0
Yatırıma_Uygunluk      5576
Takas                  5895
Tapu_Durumu           10566
Banyo_Sayısı             46
dtype: int64
```

Veri setinde hem sayısal hem de kategorik sütunlar yer almaktadır. Eksik veriler yukarıdaki sütunlarda gözlemlenmiştir

Eksik veriler şu şekilde ele alınmıştır:

- **Sayısal sütunlar** olan `Oda_Sayısı` ve `Banyo_Sayısı`, merkezi eğilimi daha iyi temsil etmesi ve uç değerlerden etkilenmemesi açısından **medyan** ile doldurulmuştur.
- **Kategorik sütunlar**, eksik değerlerin hangi sınıfa ait olabileceği net olarak bilinemediğinden dolayı `"Bilinmiyor"` etiketi ile doldurulmuştur. Bu yöntem:
  - Modelin bu eksikliği bir sinyal olarak görmesini sağlar,
  - Veri kaybını önler,
  - Kodlamada basitlik sunar.

- `Tapu_Durumu` sütunu, eksik oranı %50'den fazla olmasına rağmen model için **anlamlı bir ayırt edici öznitelik** olabileceği düşünülerek yine `"Bilinmiyor"` etiketiyle korunmuştur. Çünkü tapu türü (örneğin "Kat Mülkiyeti", "Kat İrtifakı") konutun değerini doğrudan etkileyebilir ve bu bilgi modelin kira tahmin doğruluğunu artırabilir.

In [48]:
```
data = df.copy()

data['Oda_Sayısı'].fillna(data['Oda_Sayısı'].median(), inplace=True)
data['Banyo_Sayısı'].fillna(data['Banyo_Sayısı'].median(), inplace=True)

categorical_na_cols = ['Bulunduğu_Kat', 'Eşya_Durumu', 'Yatırıma_Uygunluk', 'Tak
for col in categorical_na_cols:
    data[col].fillna("Bilinmiyor", inplace=True)

label_cols = ['Bulunduğu_Kat', 'Eşya_Durumu', 'Binanın_Yaşı', 'Isıtma_Tipi',
              'Şehir', 'Kullanım_Durumu', 'Yatırıma_Uygunluk', 'Takas', 'Tapu_Du


data.head()
```

Out[48]:

	Net_Metrekare	Brüt_Metrekare	Oda_Sayısı	Bulunduğu_Kat	Eşya_Durumu	Binanın_Y
0	120	150.0	4.0	4.Kat	Eşyalı	21 Ve Üz
1	100	125.0	4.0	3.Kat	Boş	
2	89	95.0	3.0	4.Kat	Boş	0 (Ye
3	40	55.0	2.0	6.Kat	Boş	0 (Ye
4	140	150.0	4.0	Düz Giriş (Zemin)	Boş	5-

In [49]:
```
data.isnull().sum()
```

Out[49]:
```
Net_Metrekare         0
Brüt_Metrekare        0
Oda_Sayısı            0
Bulunduğu_Kat         0
Eşya_Durumu           0
Binanın_Yaşı          0
Isıtma_Tipi           0
Fiyat                 0
Şehir                 0
Binanın_Kat_Sayısı    0
Kullanım_Durumu       0
Yatırıma_Uygunluk     0
Takas                 0
Tapu_Durumu           0
Banyo_Sayısı          0
dtype: int64
```

Yukarda göründüğü gibi eksik verilerimizi doldurduk

Çoğaltılmış (Duplicate) Verilerin Kontrolü

- Veri setinde aynı özelliklere sahip birden fazla kayıt (duplicate) olup olmadığı kontrol edilmiştir. Bu tür kayıtlar, modelin aynı örneği birden çok kez öğrenmesine neden olabileceği için modelin genellenebilirliğini olumsuz etkileyebilir.

In [50]:
```
duplicate_count = data.duplicated().sum()
print(f"Toplam {duplicate_count} adet tekrar eden (duplicate) satır bulundu.")
```

```
Toplam 198 adet tekrar eden (duplicate) satır bulundu.
```

In [51]:
```
data = data.drop_duplicates()
```

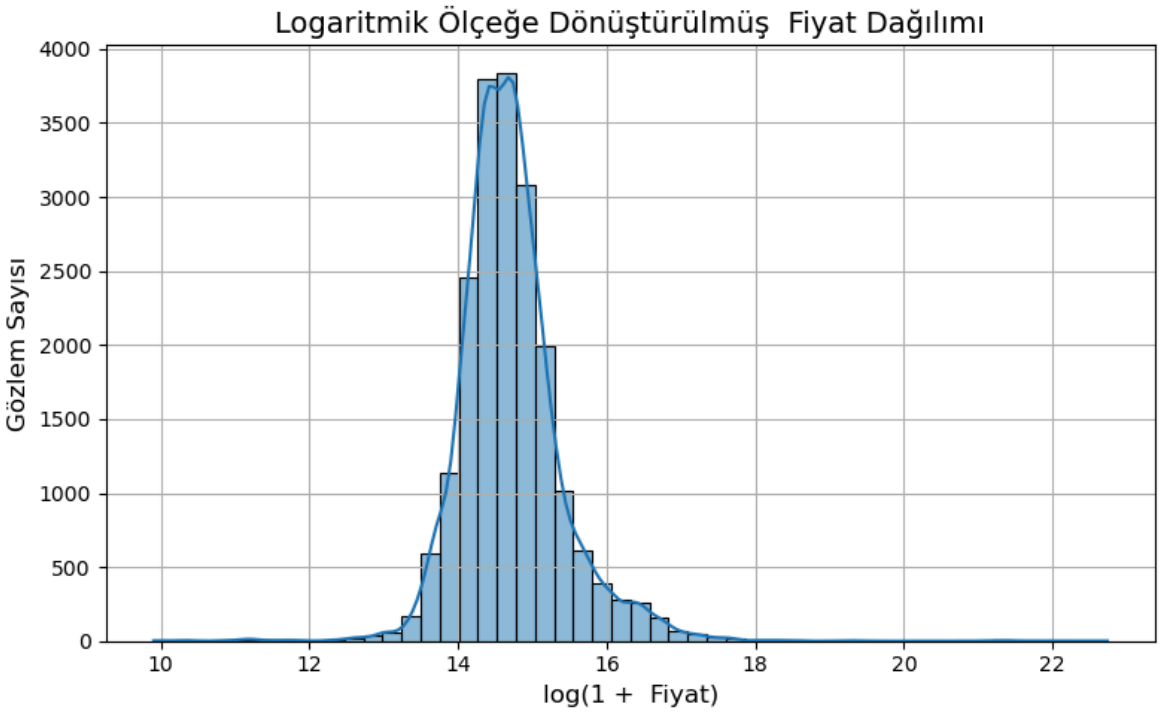
Veri setinde yapılan inceleme sonucunda **198 adet tekrarlayan kayıt** tespit edilmiştir. Aynı verilere sahip bu kayıtların model eğitimine dahil edilmesi, modelin aynı bilgiyi birden çok kez öğrenmesine neden olarak overfitting riskini artırabilir.

Bu nedenle, veri setinden bu satırlar çıkartılmış

Veri Görselleştirme

- Veri setinin temel özelliklerini daha iyi anlamak amacıyla çeşitli görselleştirmeler yapılmıştır. Bu görselleştirmeler, hem hedef değişken olan `Fiyat` ın dağılımını hem de diğer özniteliklerle olan ilişkisini ortaya koymak açısından önemlidir.

In [52]:
```
#Fiyat Dağılımı - Histogram
plt.figure(figsize=(8, 5))
sns.histplot(np.log1p(data['Fiyat']), bins=50, kde=True)
plt.title("Logaritmik Ölçeğe Dönüştürülmüş  Fiyat Dağılımı", fontsize=14)
plt.xlabel("log(1 +  Fiyat)", fontsize=12)
plt.ylabel("Gözlem Sayısı", fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```



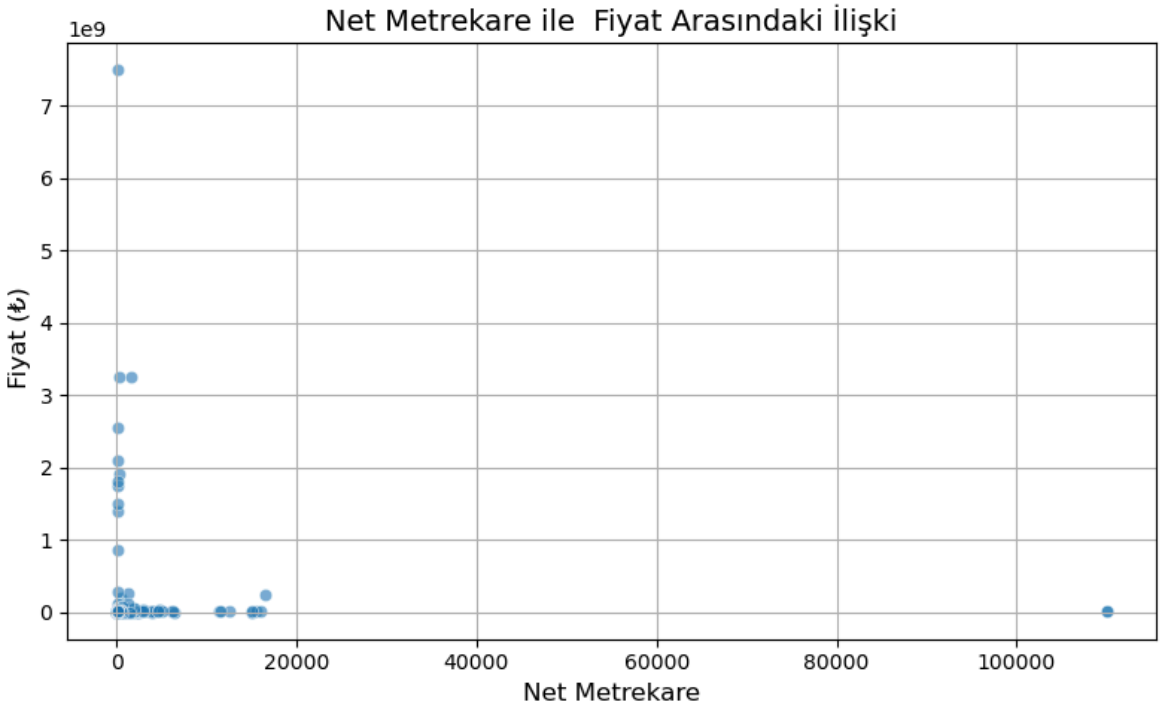
Ev fiyatları değişkeni uç değerler içermekte ve orijinal halinde sağa çarpık bir dağılım sergilemektedir. Bu durum, veri analizi ve bazı makine öğrenmesi algoritmaları açısından problem teşkil edebilir.

Bu nedenle `log(1 + Fiyat)` dönüşümü uygulanarak, dağılım daha simetrik hale getirilmiştir

In [53]:
```
#Net Metrekare ile Kira Fiyatı Arasındaki İlişki
plt.figure(figsize=(8, 5))

filtered_data = data[data['Net_Metrekare'] <= 500000]

sns.scatterplot(x='Net_Metrekare', y='Fiyat', data=filtered_data, alpha=0.6)
plt.title('Net Metrekare ile  Fiyat Arasındaki İlişki', fontsize=14)
plt.xlabel('Net Metrekare', fontsize=12)
plt.ylabel(' Fiyat (₺)', fontsize=12)
plt.grid(True)
plt.tight_layout()
plt.show()
```



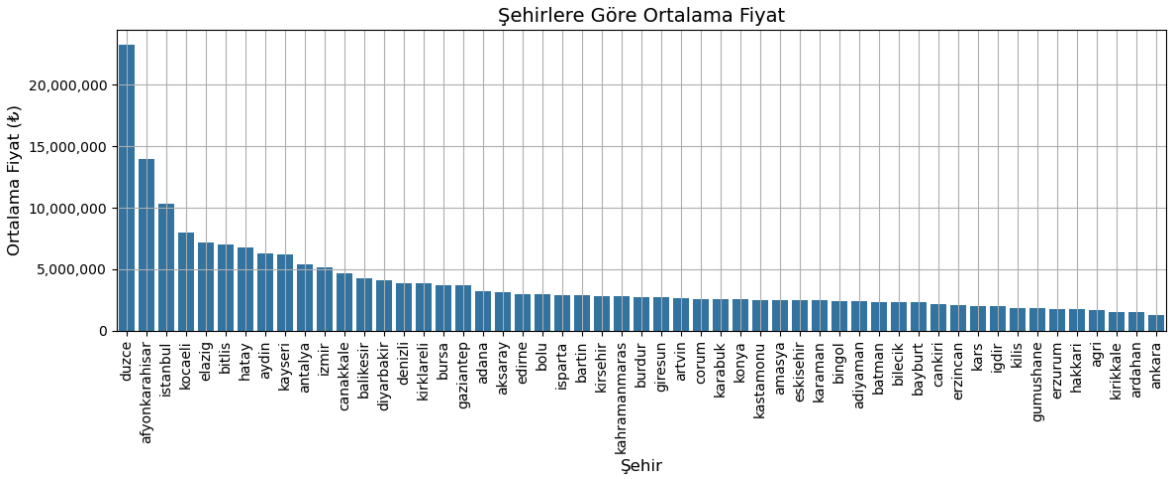
Grafik bize metrekare ile kira arasında mantıklı bir ilişki olduğunu, ancak bu ilişkiyi bozabilecek aykırı gözlemler olduğunu gösteriyor. Bu nedenle:

Modeli eğitirken bu uç değerleri dikkate almak önemli,

Gerekirse bu değerleri filtrelemek modelin doğruluğunu artırabilir.

In [54]:
```
import matplotlib.ticker as mticker

plt.figure(figsize=(12, 5))
mean_prices = df.groupby('Şehir')['Fiyat'].mean().sort_values(ascending=False)
sns.barplot(x=mean_prices.index, y=mean_prices.values)
plt.title("Şehirlere Göre Ortalama Fiyat", fontsize=14)
plt.xlabel("Şehir", fontsize=12)
plt.ylabel("Ortalama Fiyat (₺)", fontsize=12)
plt.xticks(rotation=90)
plt.gca().yaxis.set_major_formatter(mticker.StrMethodFormatter('{x:,.0f}'))
plt.grid(True)
plt.tight_layout()
plt.show()
```



Şehirler bazında ortalama fiyat değerlerinin karşılaştırıldığı bu grafikte dikkat çeken bazı anomaliler mevcuttur. Özellikle Düzce ve Afyonkarahisar gibi şehirlerin, İstanbul'dan daha yüksek ortalama fiyat değerine sahip olması, verideki aykırı değerlerin etkisini açıkça göstermektedir.

Bu durum, bazı şehirlerdeki ilan sayısının az olmasından veya çok yüksek fiyatlı birkaç kaydın ortalamayı bozmasından kaynaklanıyor olabilir. Bu nedenle şehir bilgisi, fiyat tahmini için önemli bir değişken olmakla birlikte, modellemeden önce bu tür değerlerin gözden geçirilmesi önemlidir.

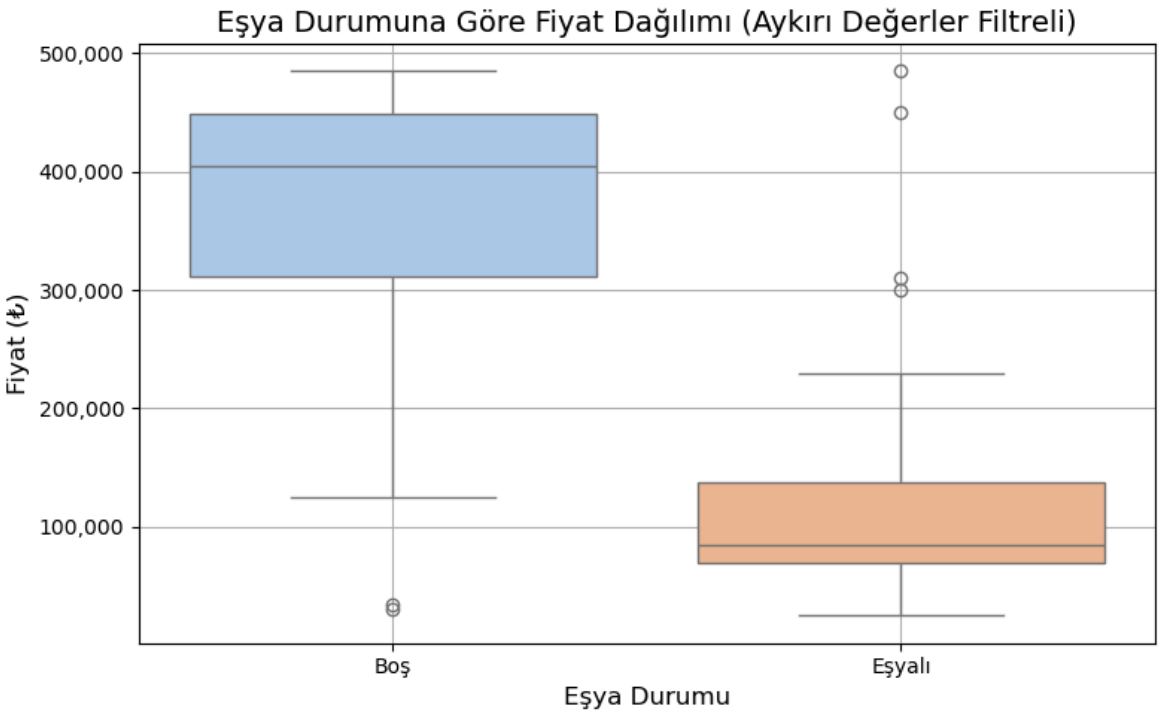
Gelecekte, ortalama yerine **medyan fiyat** gibi aykırı değerlerden daha az etkilenen ölçüler kullanılabilir.

In [55]:
```
filtered_df = df[df['Fiyat'] < 500000]

plt.figure(figsize=(8, 5))
sns.boxplot(data=filtered_df, x='Eşya_Durumu', y='Fiyat', palette="pastel")
plt.title("Eşya Durumuna Göre Fiyat Dağılımı (Aykırı Değerler Filtreli)", fontsi
plt.xlabel("Eşya Durumu", fontsize=12)
plt.ylabel("Fiyat (₺)", fontsize=12)
plt.grid(True)

# Sayısal biçimi düzelt
plt.gca().yaxis.set_major_formatter(mticker.StrMethodFormatter('{x:,.0f}'))

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



Evin eşyalı olup olmamasının fiyata etkisi bu boxplot ile analiz edilmiştir. Aykırı değerler filtrelendikten sonra görselleştirme daha net hale gelmiştir.

Grafikte dikkat çeken bir durum, "Boş" evlerin "Eşyalı" evlere kıyasla daha yüksek Fiyatlı değerlerine sahip olmasıdır. Bu durum genel beklentinin tersidir ve verideki dağılımın

altındaki nedenlerin sorgulanmasını gerektirir.

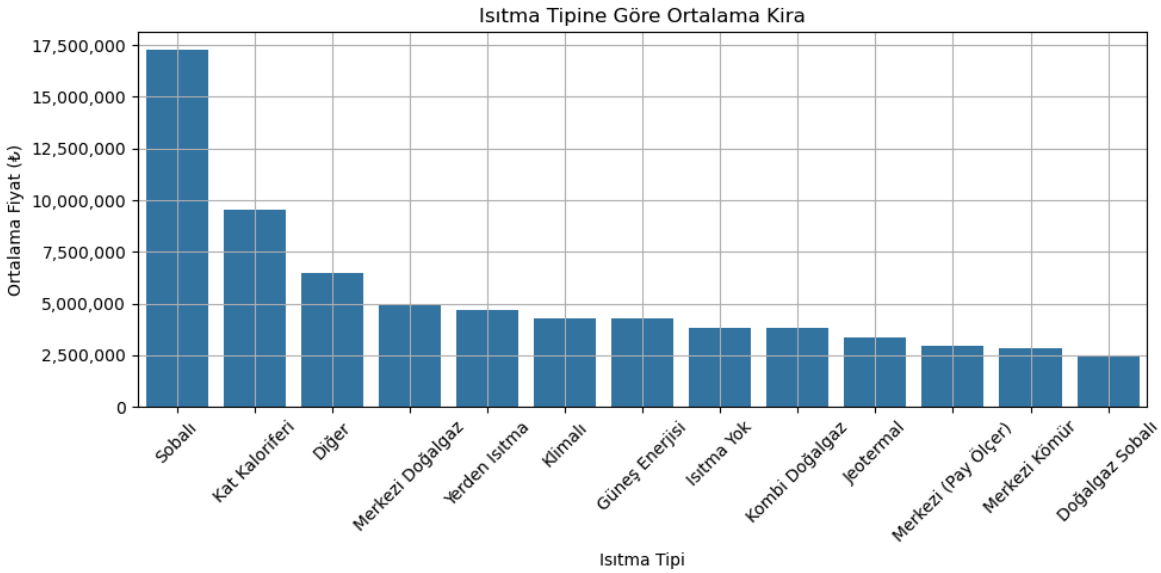
Bu farkın nedeni, boş evlerin genellikle daha büyük veya lüks olması ya da eşyalı evlerin daha küçük, öğrenciler için kiralanan daireler olması olabilir. Alternatif olarak, etiketleme hataları da bu dağılımı bozmuş olabilir.

Sonuç olarak, eşya durumu önemli bir değişken olmakla birlikte, diğer değişkenlerle birlikte analiz edilerek modelleme sürecine dahil edilmelidir.

In [56]:
```
plt.figure(figsize=(10, 5))
mean_heating = df.groupby('Isıtma_Tipi')['Fiyat'].mean().sort_values(ascending=F
sns.barplot(x=mean_heating.index, y=mean_heating.values)
plt.title("Isıtma Tipine Göre Ortalama Kira")
plt.xlabel("Isıtma Tipi")
plt.ylabel("Ortalama Fiyat (₺)")
plt.xticks(rotation=45)
plt.grid(True)

plt.gca().yaxis.set_major_formatter(mticker.StrMethodFormatter('{x:,.0f}'))

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



Bu grafikte konutların ısıtma türlerine göre ortalama kira fiyatları karşılaştırılmıştır. Ancak sonuçlar bazı anomaliler içermektedir.

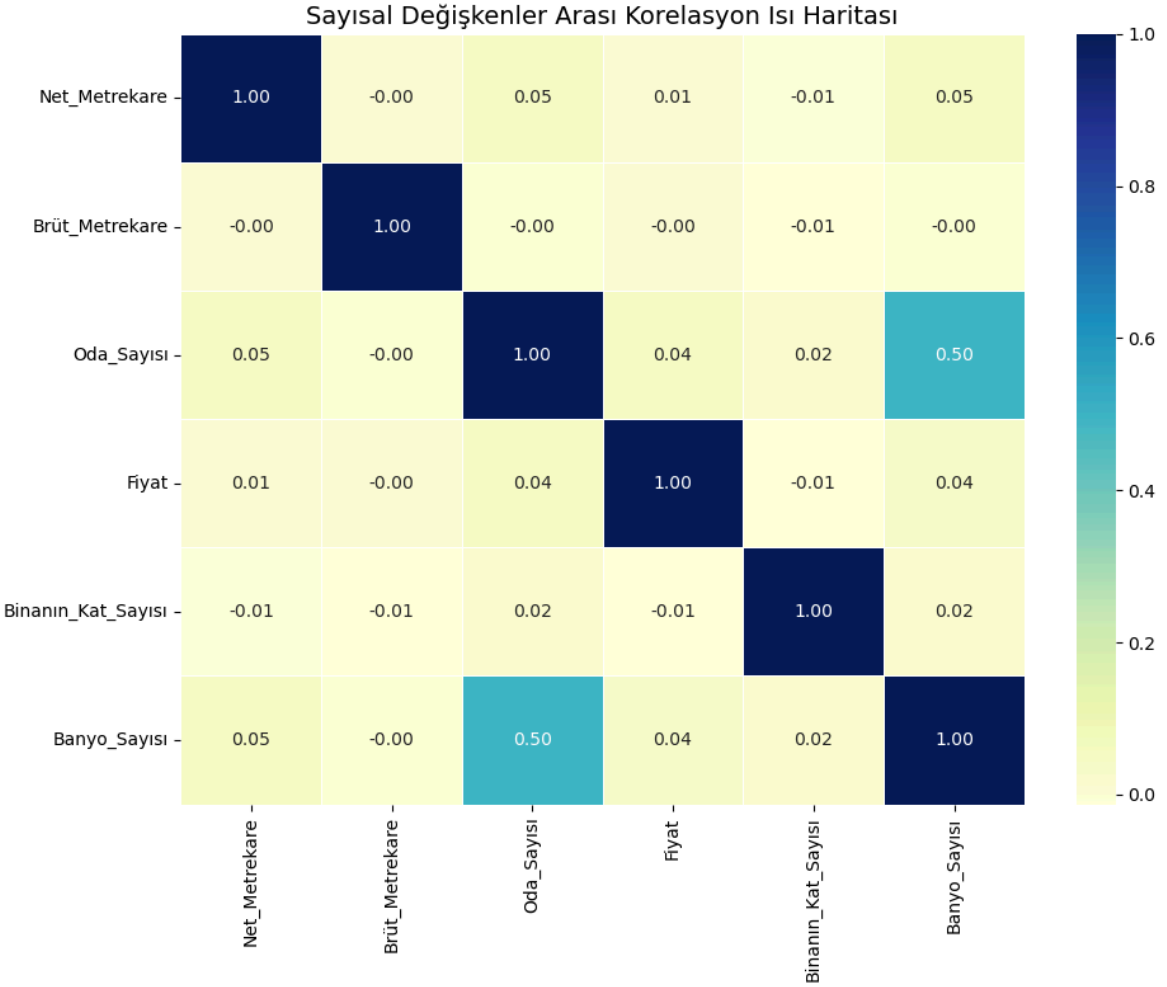
Örneğin, soba ile ısıtılan evlerin ortalama kira fiyatı beklenmeyen şekilde en yüksek çıkmıştır. Oysa ki soba genellikle daha düşük fiyatlı konutlarda tercih edilen bir ısıtma türüdür. Bu durum, veri setindeki aykırı değerlerin veya az sayıda verinin ortalamayı bozmasının bir sonucu olabilir.

Modern ısıtma sistemlerine sahip evlerin (örneğin kombi veya merkezi doğalgaz) daha düşük ortalama fiyatlara sahip olması da dikkat çekicidir. Bu nedenle bu özelliğin modele dahil edilmeden önce gruplanarak sadeleştirilmesi ya da aykırı değerlerden arındırılması faydalı olabilir.

In [57]:
```
# Sadece sayısal sütunları seç
numerical_df = data.select_dtypes(include=['int64', 'float64'])

# Korelasyon matrisini oluştur
corr_matrix = numerical_df.corr()

# Heatmap çizimi
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap="YlGnBu", fmt=".2f", linewidths=0.5)
plt.title("Sayısal Değişkenler Arası Korelasyon Isı Haritası", fontsize=14)
plt.tight_layout()
plt.show()
```



orelasyon ısı haritası, veri setindeki sayısal değişkenler arasındaki ilişki düzeyini görselleştirmek amacıyla oluşturulmuştur. Bu analizde dikkat çeken başlıca bulgular şunlardır:

- `Fiyat` değişkeninin diğer sayısal değişkenlerle olan korelasyonu oldukça zayıftır. Özellikle `Net_Metrekare`, `Brüt_Metrekare` ve `Bina Kat Sayısı` gibi değişkenler ile kira fiyatı arasında anlamlı bir doğrusal ilişki gözlemlenmemiştir.
- `Oda_Sayısı` ile `Banyo_Sayısı` arasında görece yüksek bir korelasyon bulunmaktadır (`0.50`), bu durum beklenen bir ilişkidir.
- Bu tablo, kira fiyatını etkileyen asıl değişkenlerin kategorik yapıda olduğunu veya fiyat verisinde ciddi aykırılıklar bulunduğunu düşündürmektedir.

Sonuç olarak, korelasyon analizi modelleme öncesi sezgisel çıkarımlar açısından önemlidir; ancak burada sayısal değişkenlerin sınırlı açıklayıcılığa sahip olduğu görülmektedir.

Aykırı Fiyatları Sil

Veri setindeki fiyatların dağılımı incelendiğinde, üst sınırda çok yüksek (milyarlarca lira) değerlerin bulunduğu tespit edilmiştir. Bu değerlerin gerçekçi olmaması, veri girişi hatalarına veya istisnai durumlara işaret etmektedir.

Bu nedenle `IQR (Interquartile Range)` yöntemi kullanılarak üst sınır belirlenmiş ve bu sınırı aşan fiyatlar veri setinden çıkarılmıştır. Ayrıca 1000 ₺ altı fiyat değerleri de dikkate alınmamıştır.

Bu temizlik ile modelin aykırı değerlerden etkilenmesi engellenmiş ve daha dengeli bir veri seti elde edilmiştir.

In [58]:
```
# Model Öncesi Veri Seti Hazırlığı
```

In [59]:
```
Q1 = data['Fiyat'].quantile(0.25)
Q3 = data['Fiyat'].quantile(0.75)
IQR = Q3 - Q1
upper_limit = Q3 + 1.5 * IQR

data = data[(data['Fiyat'] >= 1000) & (data['Fiyat'] <= upper_limit)].copy()
```

5. Özellik Dönüştürme ve Veri Hazırlama

Modelleme öncesinde veri setindeki bazı kategorik ya da metinsel değişkenler sayısal forma dönüştürülmüştür. Bu dönüşümler, modelin bu değişkenleri anlayabilmesi ve işleyebilmesi için gereklidir.

Bu bölümde özellikle `Bulunduğu_Kat`, `Binanın_Yaşı` gibi karma yapıya sahip sütunlar ele alınmış ve anlamlı sayısal temsillere dönüştürülmüştür. Bu dönüşümler sayesinde modelleme süreci daha sağlıklı ve kararlı hale getirilmiştir.

In [60]:
```
data["Bulunduğu_Kat"].unique()
```

Out[60]:
```
array(['4.Kat', '3.Kat', '6.Kat', 'Düz Giriş (Zemin)', '12.Kat', '2.Kat',
       'Bilinmiyor', '8.Kat', '5.Kat', '14.Kat', '1.Kat', '17.Kat',
       '9.Kat', 'Yüksek Giriş', '7.Kat', '11.Kat', 'Müstakil',
       'Bahçe Katı', '15.Kat', '10.Kat', 'Bahçe Dublex', '13.Kat',
       'Kot 3 (-3).Kat', '18.Kat', 'Çatı Dubleks', 'Kot 2 (-2).Kat',
       'Bodrum Kat', '16.Kat', 'Çatı Katı', 'Villa Tipi',
       'Kot 1 (-1).Kat', 'Kot 4 (-4).Kat', '21.Kat', '40+.Kat', '19.Kat'],
      dtype=object)
```

In [61]:
```
import re

def kat_donustur(kat):
```

```
    # Sayısal katlar
    if re.match(r'^\d+\.Kat$', kat):
        return int(kat.split('.')[0])
    elif kat == '40+.Kat':
        return 40
    elif kat in ['Düz Giriş (Zemin)', 'Yüksek Giriş']:
        return 0
    elif kat == 'Bahçe Katı':
        return -1
    elif kat == 'Bodrum Kat':
        return -2
    elif kat in ['Çatı Katı', 'Çatı Dubleks']:
        return 99
    elif kat == 'Bahçe Dublex':
        return -1
    elif 'Kot 1' in kat:
        return -1
    elif 'Kot 2' in kat:
        return -2
    elif 'Kot 3' in kat:
        return -3
    elif 'Kot 4' in kat:
        return -4
    else:
        return -999  # bilinmeyen/müstakil/villa gibi değerler
```

In [62]:
```
data["Bulunduğu_Kat"] = data["Bulunduğu_Kat"].apply(kat_donustur)
```

Veri setinde yer alan `Bulunduğu_Kat` sütunu, farklı formatlarda metinsel değerler içermekteydi. Bu değerlerin çoğu kat numarasını barındırırken, bazıları ise zemin, bodrum, çatı gibi özel kat tiplerini veya müstakil evleri ifade etmekteydi.

Bu nedenle, modellemede kullanılabilmesi için tüm `Bulunduğu_Kat` verileri belirli kurallara göre sayısal değerlere dönüştürülmüştür. Dönüşüm sırasında:

- `1.Kat`, `5.Kat` gibi değerler → doğrudan sayıya çevrildi (`1`, `5` vb.)
- `Düz Giriş (Zemin)`, `Yüksek Giriş` → `0`
- `Bodrum Kat`, `Kot 1`, `Kot 2`, ... → `-1`, `-2`, ... şeklinde negatif değerlerle temsil edildi
- `Çatı Katı`, `Çatı Dubleks` → `99` ile temsil edildi
- Anlamı belirli olmayan veya modellemede anlamlı karşılığı olmayan değerler (örneğin `Müstakil`, `Villa Tipi`, `Bilinmiyor`) → `-999` olarak kodlandı

Tüm bu işlemler sonucunda `Bulunduğu_Kat` sütunu artık tamamen sayısal hale getirilmiş ve model için kullanılabilir forma getirilmiştir. Hiçbir gözlem NaN olarak bırakılmamıştır.

In [63]:
```
data['Binanın_Yaşı'].unique()
```

Out[63]:
```
array(['21 Ve Üzeri', '4', '0 (Yeni)', '5-10', '16-20', '3', '2', '11-15',
       '1'], dtype=object)
```

In [64]:
```
yas_map = {
    "0 (Yeni)": 0,
    "1": 1,
```

```
    "2": 2,
    "3": 3,
    "4": 4,
    "5-10": 7,
    "11-15": 13,
    "16-20": 18,
    "21 Ve Üzeri": 25,
    "Bilinmiyor": -1  # isteğe bağlı
}

data["Binanın_Yaşı"] = data["Binanın_Yaşı"].map(yas_map)
```

`Binanın_Yaşı` sütunu, veri setinde karışık formatlarda (bazı değerler sayısal, bazıları metin aralıkları) yer almaktaydı. Bu nedenle sütundaki tüm değerler tutarlı ve modellemede anlamlı olacak şekilde sayısal formatlara dönüştürülmüştür.

Kullanılan dönüşüm şu şekildedir:

- `"0 (Yeni)"` → 0
- `"1"`, `"2"`, `"3"`, `"4"` → kendi değerleri
- `"5-10"` → 7
- `"11-15"` → 13
- `"16-20"` → 18
- `"21 Ve Üzeri"` → 25
- `"Bilinmiyor"` → -1

Bu şekilde, `Binanın_Yaşı` sütunu artık tam sayısal hale gelmiş ve modellemeye hazırdır.

Bağımlı ve Bağımsız Değişkenlerin Tanımlanması

In [65]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 18304 entries, 0 to 20325
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Net_Metrekare       18304 non-null  int64  
 1   Brüt_Metrekare      18304 non-null  float64
 2   Oda_Sayısı          18304 non-null  float64
 3   Bulunduğu_Kat       18304 non-null  int64  
 4   Eşya_Durumu         18304 non-null  object 
 5   Binanın_Yaşı        18304 non-null  int64  
 6   Isıtma_Tipi         18304 non-null  object 
 7   Fiyat               18304 non-null  float64
 8   Şehir               18304 non-null  object 
 9   Binanın_Kat_Sayısı  18304 non-null  int64  
 10  Kullanım_Durumu     18304 non-null  object 
 11  Yatırıma_Uygunluk   18304 non-null  object 
 12  Takas               18304 non-null  object 
 13  Tapu_Durumu         18304 non-null  object 
 14  Banyo_Sayısı        18304 non-null  float64
dtypes: float64(4), int64(4), object(7)
memory usage: 2.2+ MB
```

In [66]:
```
data.columns
```

Out[66]:
```
Index(['Net_Metrekare', 'Brüt_Metrekare', 'Oda_Sayısı', 'Bulunduğu_Kat',
       'Eşya_Durumu', 'Binanın_Yaşı', 'Isıtma_Tipi', 'Fiyat', 'Şehir',
       'Binanın_Kat_Sayısı', 'Kullanım_Durumu', 'Yatırıma_Uygunluk', 'Takas',
       'Tapu_Durumu', 'Banyo_Sayısı'],
      dtype='object')
```

In [67]:
```
y = data["Fiyat"]
X =data.drop(["Fiyat",'Takas'],axis=1)
```

Modelin hedef değişkeni olarak `Fiyat` sütunu belirlenmiştir. Bu değişken, evin tahmin edilmek istenen fiyat bedelini temsil etmektedir.

Bağımsız değişkenler olarak, `Takas` dışındaki tüm sütunlar modele dahil edilmiştir. `Takas` sütunu, ev sahibinin tercihine yönelik bir bilgi taşıdığından, ev fiyatını doğrudan etkileyebileceği düşünülmediği için modelden çıkarılmıştır.

Kategorik Değişkenlerin Kodlanması (Label Encoder)

In [68]:
```
categorical_cols = X.select_dtypes(include="object").columns

le = LabelEncoder()
for col in categorical_cols:
    X[col] = le.fit_transform(X[col])
```

Ölçeklendirilme (StandardScaler)

In [69]:
```
numerical_cols = [
    "Net_Metrekare",
    "Brüt_Metrekare",
    "Oda_Sayısı",
    "Bulunduğu_Kat",
    "Binanın_Yaşı",
    "Binanın_Kat_Sayısı",
    "Banyo_Sayısı"
]

scaler = StandardScaler()

X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

Modelleme sürecinde bazı algoritmalar (örneğin XGBoost, KNN, vs.) özellikle ölçeklenmiş verilere daha duyarlı çalışmaktadır. Bu nedenle, sayısal değişkenler `StandardScaler` yöntemi ile standartlaştırılmıştır. Bu yöntem, her değişkenin ortalamasını 0, standart sapmasını 1 olacak şekilde dönüştürür.

Ölçekleme yalnızca sürekli sayısal değişkenlere uygulanmıştır:

In [70]:
```
X.head()
```

Out[70]:

	Net_Metrekare	Brüt_Metrekare	Oda_Sayısı	Bulunduğu_Kat	Eşya_Durumu	Binanın_Y
0	-0.014918	-0.007392	0.446566	0.276851	2	2.0369
1	-0.031990	-0.007392	0.446566	0.273013	1	-0.4499
2	-0.041379	-0.007392	-0.581021	0.276851	1	-0.9236
3	-0.083205	-0.007392	-1.608608	0.284527	1	-0.9236
4	0.002154	-0.007392	0.446566	0.261499	1	-0.0947

Eğitim ve Test Verisinin Ayrılması (train_test_split)

In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Modelin performansını değerlendirebilmek için veri seti eğitim ve test olarak ikiye ayrılmıştır. Bu işlem `train_test_split` fonksiyonu ile gerçekleştirilmiştir.

Random Forest Regressor ile Model Eğitimi

In [72]:
```
# Modeli oluştur ve eğitilmesi
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
```

Out[72]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [73]:
```
# Test verisi ile tahmin yap
y_pred_rf = rf_model.predict(X_test)
```

In [74]:
```
# Başarı metrikleri
rmse_rf = np.sqrt(mean_squared_error(y_test, y_pred_rf))
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest RMSE: {rmse_rf:,.2f}")
print(f"Random Forest R2: {r2_rf:.4f}")
```

```
Random Forest RMSE: 682,584.40
Random Forest R2: 0.5921
```

İlk model olarak, ağaç tabanlı bir algoritma olan `RandomForestRegressor` kullanılmıştır. Bu model, yüksek performansı ve aşırı öğrenmeye (overfitting) karşı dayanıklı yapısı nedeniyle sıklıkla tercih edilir.

Model, eğitim verisiyle eğitilmiş ve test verisi üzerinde değerlendirilmiştir. Performans ölçütü olarak:

- **RMSE (Root Mean Squared Error)**: Ortalama tahmin hatasının büyüklüğünü ölçer
- **R^2 (Determination Coefficient)**: Modelin değişkenliği ne kadar açıkladığını gösterir

Aşağıda modelin test verisi üzerindeki performansı sunulmuştur:

- **RMSE**: 682,584.40
- **R^2**: 0.5921

GridSearchCV ile Random Forest Hiperparametre Taraması

In [75]:
```
from sklearn.model_selection import GridSearchCV

# Parametre aralıkları (başlangıç için makul ve dengeli)
param_grid_rf = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# GridSearchCV nesnesi
grid_search_rf = GridSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_grid=param_grid_rf,
    scoring='neg_root_mean_squared_error',
    cv=5,
    n_jobs=-1,
    verbose=1
```

```
)

# Eğitim
grid_search_rf.fit(X_train, y_train)

# Sonuçlar
print("En iyi parametreler:", grid_search_rf.best_params_)
print(f"En iyi RMSE (cross-val): {-grid_search_rf.best_score_:,.2f}")
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
En iyi parametreler: {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_spli
t': 2, 'n_estimators': 200}
En iyi RMSE (cross-val): 687,403.22
```

GridSearchCV ile Random Forest Test Seti ile Değerlendirme

In [76]:
```
# En iyi modeli al
best_rf = grid_search_rf.best_estimator_

# Test verisi ile tahmin
y_pred_best_rf = best_rf.predict(X_test)

# Skorlar
rmse_best_rf = np.sqrt(mean_squared_error(y_test, y_pred_best_rf))
r2_best_rf = r2_score(y_test, y_pred_best_rf)

print(f"Test RMSE (best RF model): {rmse_best_rf:,.2f}")
print(f"Test R2 (best RF model): {r2_best_rf:.4f}")
```

```
Test RMSE (best RF model): 677,836.67
Test R2 (best RF model): 0.5977
```

Random Forest modeli, hiperparametre ayarlaması yapılmadan önce belirli bir performans göstermişti. Bu performansı artırmak amacıyla `GridSearchCV` ile hiperparametre optimizasyonu gerçekleştirilmiştir. Kullanılan parametre aralığı şu şekildedir:

- `n_estimators` : [100, 200]
- `max_depth` : [None, 10, 20]
- `min_samples_split` : [2, 5]
- `min_samples_leaf` : [1, 2]

5 katlı çapraz doğrulama (cross-validation) ile en iyi parametre kombinasyonu belirlenmiştir.

En İyi Parametreler:

> {'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 200}

Test Verisi Üzerindeki Performansı:

- **Test RMSE:** 677,836.67

- **Test R^2:** 0.5977

Bu sonuçlara göre, optimize edilmiş Random Forest modeli önceki temel modelden daha iyi sonuç vermiştir. Ancak XGBoost modeli ile karşılaştırıldığında hala biraz daha düşük bir performansa sahiptir.

XGBoost Regressor ile Model Eğitimi

In [77]:
```
# Modeli oluştur ve eğit
xgb_model = XGBRegressor(random_state=42)
xgb_model.fit(X_train, y_train)
```

Out[77]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_roun
ds=None,
             enable_categorical=False, eval_metric=None, feature_typ
es=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=None, max_bin=None, max_cat_threshold=Non
```

In [78]:
```
# Tahmin yap
y_pred_xgb = xgb_model.predict(X_test)
```

In [79]:
```
# Değerlendirme
rmse_xgb = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"XGBoost RMSE: {rmse_xgb:,.2f}")
print(f"XGBoost R2: {r2_xgb:.4f}")
```

```
XGBoost RMSE: 658,662.72
XGBoost R2: 0.6202
```

İkinci model olarak `XGBoost Regressor` kullanılmıştır. Bu algoritma, gradyan artırmalı (gradient boosting) karar ağaçları üzerine kuruludur ve birçok yarışmada üstün performansıyla öne çıkmaktadır.

Model, eğitim verileriyle eğitilmiş ve test verileri üzerinde değerlendirilmiştir. Kullanılan başarı ölçütleri:

- **RMSE**: Hata büyüklüğünü ölçer
- **R^2**: Modelin açıklayıcılık gücünü gösterir

Elde edilen sonuçlar:

- **RMSE**: 658,662.72
- **R^2**: 0.6202

GridSearchCV ile XGBoost Hiperparametre Taraması

In [80]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.05, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

xgb_for_grid = XGBRegressor(random_state=42)

grid_search = GridSearchCV(
    estimator=xgb_for_grid,
    param_grid=param_grid,
    scoring='neg_root_mean_squared_error',
    cv=5,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

print("En iyi parametreler:", grid_search.best_params_)
print(f"En iyi RMSE (cross-val): {-grid_search.best_score_:,.2f}")
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
En iyi parametreler: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 200,
'subsample': 0.8}
En iyi RMSE (cross-val): 660,888.70
```

XGBoost modelinin performansını artırmak için hiperparametre optimizasyonu uygulanmıştır. Bu işlemde `GridSearchCV` kullanılarak farklı parametre kombinasyonları denenmiş ve çapraz doğrulama ile en iyi sonuç veren model seçilmiştir.

Denenen parametreler:

- `n_estimators` : [100, 200]
- `max_depth` : [3, 5, 7]
- `learning_rate` : [0.05, 0.1, 0.2]
- `subsample` : [0.8, 1.0]

Bu işlem sonucunda elde edilen en iyi parametreler ve RMSE değeri aşağıda verilmiştir:

- **En iyi parametreler**: {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 200, 'subsample': 0.8}
- **Cross-Validation RMSE**: 660,888.70

GridSearchCV ile XGBoost Test Seti ile Değerlendirme

In [81]:
```
# En iyi model ile test seti üzerinde tahmin
best_xgb = grid_search.best_estimator_
y_pred_best_xgb = best_xgb.predict(X_test)

rmse_best = np.sqrt(mean_squared_error(y_test, y_pred_best_xgb))
r2_best = r2_score(y_test, y_pred_best_xgb)

print(f"Test RMSE (best model): {rmse_best:,.2f}")
print(f"Test R2 (best model): {r2_best:.4f}")
```

```
Test RMSE (best model): 647,896.29
Test R2 (best model): 0.6325
```

XGBoost Regressor modeli için hiperparametre optimizasyonu yapılmıştır. Bu işlem, farklı parametre kombinasyonlarının denenerek en iyi sonuç veren modelin seçilmesini sağlamıştır. Kullanılan parametre aralığı:

- `n_estimators` : [100, 200]
- `max_depth` : [3, 5, 7]
- `learning_rate` : [0.05, 0.1, 0.2]
- `subsample` : [0.8, 1.0]

Bu aralıklar içinde yapılan 5 katlı çapraz doğrulama sonucunda en iyi parametre kombinasyonu belirlenmiş ve model yeniden eğitilmiştir.

En İyi Parametreler:

> {'learning_rate': 0.05, 'max_depth': 7, 'n_estimators': 200, 'subsample': 0.8}

Test Verisi Üzerindeki Performansı:

- **Test RMSE:** 647,896.29
- **Test R^2:** 0.6325

Bu sonuçlar, optimize edilen XGBoost modelinin yüksek doğrulukla kira fiyatlarını tahmin ettiğini göstermektedir. Performansı, temel modele göre önemli ölçüde iyileşmiştir ve diğer modellerin önünde yer almaktadır.

Model Karşılaştırması (Hiperparametre Optimizasyonu Sonrası)

Her iki regresyon modeli de `GridSearchCV` kullanılarak optimize edilmiştir. Optimize edilmiş modeller, test verisi üzerinde aşağıdaki sonuçları vermiştir:

Model	Test RMSE	Test R^2
Random Forest (tuned)	677,836.67	0.5977

Model	Test RMSE	Test R^2
XGBoost (tuned)	647,896.29	0.6325

📌 Yorum:

- RMSE değeri daha düşük olan model daha az hata yapmaktadır.
- R^2 değeri, modelin veri setindeki varyansı ne kadar açıkladığını gösterir.
- XGBoost, her iki metrikte de Random Forest'a göre daha iyi performans göstermiştir.

✅ Sonuç:

Bu nedenle, **final model olarak GridSearch ile optimize edilmiş XGBoost Regressor modeli seçilmiştir**.