

20 MAYIS 2025

VERİ MADENCİLİĞİ PROJE

EMLAK VERİLERİ ANALİZİ

10. GRUP

Mustafa Enes Yeşilyurt G210102063

Yusuf Baran korkmaz B221200384

Süha Sardoğan B201200058

Yiğit Kalaycıoğlu B221200054

Emir Samed Tüfekci B221200056

Emirhan Yıldırım B211200004

Bartu Alp Eren B221200052

Kaan Biber B231200382

Selim Efiloğlu B211200015

Deniz Erdem Aras B221200014

VERİ ÖN İŞLEME

- 1- YÜZDE 90'I BOŞ VE MODELDE ANLAMSIZ SONUÇLAR ÇIKARABİLECEĞİ DÜŞÜNÜLEN VERİ SÜTUNLARI ÇIKARILDI.

```
# %90'ten fazla eksik veriye sahip sütunları tespit et ve sil
threshold = 0.90
missing_ratio = df.isnull().mean() # Her sütunun eksik değer oranını hesapla
columns_to_drop = missing_ratio[missing_ratio > threshold].index # %90'ten fazla eksik olan sütunlar
df_cleaned = df.drop(columns=columns_to_drop) # Bu sütunları veri setinden çıkar

# Çıkarılacak sütunların isimleri
columns_to_drop = [
    "detailUrl", "redirectLink", "detailDescription", "firm",
    "firmuser", "land", "subcategory", "attributes",
    "timeShareName", "isMapHidden", "streetView", "numberOfBuilding",
    "entranceHeight", "bedCount", "roomAndLivingRoom", "floorAreaRatio",
    "listingId", "listingPropertyOrder", "listingId.1", "maximumStay",
    "minimumStay", "guestCount", "entertainmentAllowed"
]

# Belirtilen sütunları veri setinden çıkar
df_cleaned = df.drop(columns=columns_to_drop, errors="ignore")
```

- 2-start_date ve end_date sütunlarının saat kısmı silinip sadece tarih kısmı bırakıldı.

```
# startDate ve endDate sütunlarındaki saat bilgilerini kaldırır, sadece tarih kalsın
df_cleaned["startDate"] = pd.to_datetime(df_cleaned["startDate"], errors='coerce').dt.date
df_cleaned["endDate"] = pd.to_datetime(df_cleaned["endDate"], errors='coerce').dt.date

tools.display_dataframe_to_user(name="Tarih Formatı Güncellenmiş Veri", dataframe=df_cleaned)
```

- 3 Areas sütununun anlamlı verisi olan etiketleri bırakılıp (İstiklal, Karaosman vs.) kalanı silindi.

```
# areas sütunundaki her hizmeteki son "name" değerini çıkar
def extract_last_area_name(area_str):
    try:
        area_list = ast.literal_eval(area_str)
        if isinstance(area_list, list) and area_list:
            return area_list[-1].get("name", None)
    except Exception:
        return None

df_cleaned["areas"] = df_cleaned["areas"].apply(extract_last_area_name)

tools.display_dataframe_to_user(name="Areas Sütunu Düzenlenmiş Veri", dataframe=df_cleaned)
```

- 4- district sütununun anlamlı verileri (Aziziye, Kirazlı , Yalı vs.) bırakılıp kalanı silindi.

```

# district sütunundaki 3. "name" değerini al
def extract_third_district_name(district_str):
    try:
        district_list = ast.literal_eval(district_str)
        if isinstance(district_list, list) and len(district_list) >= 3:
            return district_list[2].get("name", None)
    except Exception:
        return None

df_cleaned["district"] = df_cleaned["district"].apply(extract_third_district_name)

tools.display_dataframe_to_user(name="District Sütunu Düzenlenmiş Veri", dataframe=df_cleaned)

```

5-county sütununun anlamlı verileri bırakılıp(Sapanca, Adapazarı,Karasu vs.) geri kalan veri etiketleri silindi.

```

# county sütunundaki "name" değerini al
def extract_third_district_name(county_str):
    try:
        county_list = ast.literal_eval(county_str)
        if isinstance(county_list, list) and len(county_list) >= 3:
            return county_list[2].get("name", None)
    except Exception:
        return None

df_cleaned["county"] = df_cleaned["county"].apply(extract_third_district_name)

tools.display_dataframe_to_user(name="County Sütunu Düzenlenmiş Veri", dataframe=df_cleaned)

```

6- Tüm satırlardaki veriler Sakarya'da olduğu için city sütünuna Sakarya yazıldı geri kalan etiketler silindi.

```

# tüm city sütununu "Sakarya" olarak ayarla
df_original_full["city"] = "Sakarya"

tools.display_dataframe_to_user(name="City Sütunu Sakarya Olarak Güncellendi", dataframe=df_original_full[["city"]].head(20))

```

7- Residence üzerinden sadece anlamlı veri çekilerek(daire, doubleks, tripleks vs.) geri kalan etiketler silindi.

```

# residence sütunundaki "name" alanını çek
def extract_residence_name(residence_str):
    if pd.isna(residence_str):
        return None
    try:
        residence_dict = ast.literal_eval(residence_str)
        if isinstance(residence_dict, dict):
            return residence_dict.get("name", None)
    except Exception:
        return None

df_original_full["residence"] = df_original_full["residence"].apply(extract_residence_name)

tools.display_dataframe_to_user(name="Residence Sütunu Düzenlendi", dataframe=df_original_full[["residence"]].head(20))

```

8- room ve **linigRoom** sütunlarındaki köşeli parantezleri kaldırarak veriyi sayısal hale getirip modele hazırladık.

```

# "room" ve "livingRoom" sütunlarındaki parantezleri kaldır
columns_to_clean = ["room", "livingRoom"]
for col in columns_to_clean:
    if col in df_from_uploaded.columns:
        df_from_uploaded[col] = df_from_uploaded[col].astype(str).str.replace("(", "", regex=False).str.replace(")", "", regex=False)

tools.display_dataframe_to_user(name="Parantezleri Kaldırılmış Oda Bilgileri", dataframe=df_from_uploaded[columns_to_clean].head(20))

```

9-mapLocation sütununu ikiye ayırarak enlem ve boylam şeklinde güncelleyp ana sütunu sil dik.

```
# maplocation sütununu içleyerek "latitude" ve "longitude" sütunlarına ayırdı
import json

def extract_lat_lon(location_str):
    try:
        loc_dict = json.loads(location_str.replace("'", ""))
        return pd.Series([loc_dict.get("lat"), loc_dict.get("lon")])
    except Exception:
        return pd.Series([None, None])

if "maplocation" in df_from_uploaded.columns:
    df_from_uploaded[["latitude", "longitude"]] = df_from_uploaded[["maplocation"]].apply(extract_lat_lon)

tools.display_dataframe_to_user(name="Latitude ve Longitude Eklendiğ Veri",
                                dataframe=df_from_uploaded[["maplocation", "latitude", "longitude"]].head(20))

# "maplocation" sütununu sil
df_from_uploaded = df_from_uploaded.drop(columns=["maplocation"], errors='ignore')

tools.display_dataframe_to_user(name="MapLocation Sütunu Silinmiş Veri", dataframe=df_from_uploaded.head(20))
```

10-createdDate ve updatedDate kısımlarındaki saat silinip tarih bırakıldı.

```
# Tarih formatları da düzeltmek için saat bilgilerini kaldırır
date_columns = ["createdDate", "updatedDate"]

for col in date_columns:
    if col in df_from_uploaded.columns:
        df_from_uploaded[col] = pd.to_datetime(df_from_uploaded[col], errors="coerce").dt.date

tools.display_dataframe_to_user(name="Tarih Formatı Düzellendiğ Veri", dataframe=df_from_uploaded[date_columns].head(20))
```

11-listingUpdatedDate sütunu içinde aynı işlem yapıldı.

```
# listingUpdatedDate sütunundaki saat bilgisini kaldırır, sadece tarih kalsın
if "listingUpdatedDate" in df_from_uploaded.columns:
    df_from_uploaded["listingUpdatedDate"] = pd.to_datetime(
        df_from_uploaded["listingUpdatedDate"], errors="coerce"
    ).dt.date

tools.display_dataframe_to_user(
    name="Listing Updated Date Düzellendiğ Veri",
    dataframe=df_from_uploaded[["listingUpdatedDate"]].head(20)
)
```

12-sqm değerini ikiye bölgerek net m² ve brüt m² olarak parçaladık.

```
# sqm'den yalnızca ilk iki sayısal değer (net ve gross gibi) ayırt eden fonksiyon
def extract_first_two_numbers(sqm_str):
    if pd.isna(sqm_str):
        return None
    try:
        # Sayı arama: hem tam sayı hem ondalık
        numbers = pd.Series(pd.to_numeric(
            pd.Series(sqm_str.replace(".", ",").replace("-", "")).replace(":", ",").replace(";", ",")
            .replace(":", ",").split(",")).str.extract(r'^(d+\.\?d*)')[0]
        ).dropna().astype(float))
        return numbers.iloc[:2].tolist() if not numbers.empty else None
    except Exception:
        return None

# Uygulama ve dizi halinde sağla
df_from_uploaded["sqm"] = df_from_uploaded[["sqm"]].apply(extract_first_two_numbers)

tools.display_dataframe_to_user(name="İlk İki Sayısal Değerle Düzellendiğ sqm", dataframe=df_from_uploaded[["sqm"]].head(20))
```

13-Bölünen sqm değerlerini net ve brüt olacak şekilde 2 sütuna ayırip orijinal sqm değeri silindi.

```
# son sütundaki iki değeri netSqm ve grossSqm olarak ayıralım
df_from_uploaded["netSqm"] = df_from_uploaded["sqm"].apply(lambda x: x[0] if isinstance(x, list) and len(x) > 0 else None)
df_from_uploaded["grossSqm"] = df_from_uploaded["sqm"].apply(lambda x: x[1] if isinstance(x, list) and len(x) > 1 else None)

tools.display_dataframe_to_user(
    name="netSqm ve grossSqm Sütunları Oluşturuldu",
    dataFrame=df_from_uploaded[["sqm", "netSqm", "grossSqm"]].head(20))

# son sütununu kaldırır
df_from_uploaded = df_from_uploaded.drop(columns=["sqm"], errors="ignore")

tools.display_dataframe_to_user(name="sqm Sütunu Silinmiş Veri", dataFrame=df_from_uploaded.head(20))
```

14- floor sütunu ikiye bölünüp floor_count ve floor_name olarak iki sütüna ayrıldı. Orijinal veri silindi.

```
# floor sütununu 'count' ve 'name' alanlarına ayıran fonksiyon
def extract_floor_part(value, part):
    try:
        parsed = ast.literal_eval(value) if isinstance(value, str) else value
        if isinstance(parsed, dict):
            return parsed.get(part)
    except:
        return None

    # Yeni sütunları oluştur
    df_final['floor_count'] = df_final['floor'].apply(lambda x: extract_floor_part(x, 'count'))
    df_final['floor_name'] = df_final['floor'].apply(lambda x: extract_floor_part(x, 'name'))

    # Sonuçları göster
    tools.display_dataframe_to_user(name="Kat bilgileri ayrılmış (floor_count ve floor_name)", dataFrame=df_final)

    # 'floor' sütununu veri setinden kaldırırız
    df_final.drop(columns=['floor'], inplace=True)

    # Güncellemiş veriyi göster
    tools.display_dataframe_to_user(name="floor sütunu silinmiş veri", dataFrame=df_final)
```

14-Bahçe katı yazanlar 0.kat diye güncellendi

```
# "Bahçe Katı" ifadesini "0. Kat" ile değiştir
df_final['floor_name'] = df_final['floor_name'].replace("Bahçe Katı", "0. Kat")

# Güncellemiş veriyi göster
tools.display_dataframe_to_user(name="floor_name sütununda 'Bahçe Katı' → '0. Kat'", dataFrame=df_final)
```

15- heating sütununun anlamlı verileri alınarak(kombi, yerden ısıtma, soba vs.) kalanı silindi.

```
# heating sütundaki "name" değerini ayıklayalım
def extract_heating_name(heating_str):
    if pd.isna(heating_str):
        return None
    try:
        heating_dict = ast.literal_eval(heating_str)
        return heating_dict.get("name")
    except Exception:
        return None

df_from_uploaded["heating"] = df_from_uploaded["heating"].apply(extract_heating_name)

tools.display_dataframe_to_user(name="Heating Sütunu Düzenlendi", dataFrame=df_from_uploaded[["heating"]].head(20))
```

16-fuel kısmının anlamlı verileri alınarak (doğalgaz, elektrik, gaz vs.) kalanları silindi.

```

# fuel sütunundaki "name" değerini ayırmak için
def extract_fuel_name(fuel_str):
    if pd.isna(fuel_str):
        return None
    try:
        fuel_dict = ast.literal_eval(fuel_str)
        return fuel_dict.get("name")
    except Exception:
        return None

df_from_uploaded["fuel"] = df_from_uploaded["fuel"].apply(extract_fuel_name)

tools.display_dataframe_to_user(name="Fuel Sütunu Düzlenlendi", dataframe=df_from_uploaded[["fuel"]].head(20))

```

17-build kısminın anlamlı verileri alınarak (betonarme, taş bina, ahşap vs.) kalanı silindi.

```

# build sütunundaki "name" değerini ayırmak için
def extract_build_name(build_str):
    if pd.isna(build_str):
        return None
    try:
        build_dict = ast.literal_eval(build_str)
        return build_dict.get("name")
    except Exception:
        return None

df_from_uploaded["build"] = df_from_uploaded["build"].apply(extract_build_name)

tools.display_dataframe_to_user(name="Build Sütunu Düzlenlendi", dataframe=df_from_uploaded[["build"]].head(20))

```

18- build_state sütununun anlamlı verileri (Sıfır, İkinci El) bırakılıp kalanı silindi.

```

# buildState sütunundaki "name" değerini ayırmak için
def extract_build_state(build_state_str):
    if pd.isna(build_state_str):
        return None
    try:
        build_state_dict = ast.literal_eval(build_state_str)
        return build_state_dict.get("name")
    except Exception:
        return None

df_from_uploaded["buildState"] = df_from_uploaded["buildState"].apply(extract_build_state)

tools.display_dataframe_to_user(name="BuildState Sütunu Düzlenlendi", dataframe=df_from_uploaded[["buildState"]].head(20))

```

19-usage kısminın sadece name değerleri alınarak (boş , kiracılı, mülk sahibi) kalanı silindi.

```

# usage sütunundaki "name" değerini ayırmak için
def extract_usage_name(usage_str):
    if pd.isna(usage_str):
        return None
    try:
        usage_dict = ast.literal_eval(usage_str)
        return usage_dict.get("name")
    except Exception:
        return None

df_from_uploaded["usage"] = df_from_uploaded["usage"].apply(extract_usage_name)

tools.display_dataframe_to_user(name="Usage Sütunu Düzlenlendi", dataframe=df_from_uploaded[["usage"]].head(20))

```

20- credit sütununun anlamlı verileri olan (Krediye uygun, Krediye uygun değil) verileri çekiliplik kalanı silindi.

```

# credit sütunundaki "name" değerini ayıralayın
def extract_credit_name(credit_str):
    if pd.isna(credit_str):
        return None
    try:
        credit_dict = ast.literal_eval(credit_str)
        return credit_dict.get("name")
    except Exception:
        return None

df_from_uploaded["credit"] = df_from_uploaded["credit"].apply(extract_credit_name)

tools.display_dataframe_to_user(name="Credit Sütunu Düzenlendi", dataframe=df_from_uploaded[["credit"]].head(20))

```

21-deposit kısmının para birimi ayıklanıldı.

```

# deposit sütunundaki "currencyCode" değerini ayıralayın
def extract_deposit_currency_code(deposit_str):
    if pd.isna(deposit_str):
        return None
    try:
        deposit_dict = ast.literal_eval(deposit_str)
        return deposit_dict.get("currencyCode")
    except Exception:
        return None

df_from_uploaded["deposit"] = df_from_uploaded["deposit"].apply(extract_deposit_currency_code)

tools.display_dataframe_to_user(name="Deposit Sütunu Düzenlendi", dataframe=df_from_uploaded[["deposit"]].head(20))

```

22-barter sütununun anlamlı verileri olan takas durumu(yapılır, yapılmaz) çekilerek kalanları silindi.

```

# barter sütunundaki "name" değerini ayıralayın
def extract_barter_name(barter_str):
    if pd.isna(barter_str):
        return None
    try:
        barter_dict = ast.literal_eval(barter_str)
        return barter_dict.get("name")
    except Exception:
        return None

df_from_uploaded["barter"] = df_from_uploaded["barter"].apply(extract_barter_name)

tools.display_dataframe_to_user(name="Barter Sütunu Düzenlendi", dataframe=df_from_uploaded[["barter"]].head(20))

```

23- rental sütunun anlamlı verisi olan amount değerleri alınarak geri kalan etiketler silindi.

```

# rental sütunundaki "amount" değerini ayıralayın
def extract_rental_amount(rental_str):
    if pd.isna(rental_str):
        return None
    try:
        rental_dict = ast.literal_eval(rental_str)
        return rental_dict.get("amount")
    except Exception:
        return None

df_from_uploaded["rental"] = df_from_uploaded["rental"].apply(extract_rental_amount)

tools.display_dataframe_to_user(name="Rental Sütunu Düzenlendi", dataframe=df_from_uploaded[["rental"]].head(20))

```

24- veri setindeki tüm ilanlar satılık olduğu için category sütunundaki tüm değerlerleri satılık olarak güncelledik.

```
# category sütunundaki tüm değerleri "Satılık" olarak değiştirelim  
df['category'] = "Satılık"  
  
tools.display_dataframe_to_user(name="Category Sütunu Satılık Olarak Güncellendi", dataframe=df_from_uploaded.head(20))
```

25- subcategory kısmındaki typeName alınarak(müstakil, bahçeli vs.) kalanı silindi.

```
# subCategory sütunundaki "typeName" değerini avuçlayalım  
def extract_subcategory_type_name(subcategory_str):  
    if pd.isna(subcategory_str):  
        return None  
    try:  
        subcategory_dict = ast.literal_eval(subcategory_str)  
        return subcategory_dict.get("typeName")  
    except Exception:  
        return None  
  
df_from_uploaded["subCategory"] = df_from_uploaded["subCategory"].apply(extract_subcategory_type_name)  
  
tools.display_dataframe_to_user(name="Subcategory Sütunu Güncellendi", dataframe=df_from_uploaded[['subCategory']].head(20))
```

26- Tüm veriler konut olduğu için mainCategory/deki veriler Konut olarak güncellendi.

```
# mainCategory sütunundaki tüm değerleri "Konut" olarak değiştirelim  
df_from_uploaded["mainCategory"] = "Konut"  
  
tools.display_dataframe_to_user(name="MainCategory Sütunu Konut Olarak Güncellendi", dataframe=df_from_uploaded.head(20))
```

27- housingComplex verisinin name kısmı alınıp diğer etiketler silindi.

```
# 'housingComplex' sütununu düşürt (sadece 'name' değerini al)  
if 'housingComplex' in df_excel.columns:  
    df_excel['housingComplex'] = df_excel['housingComplex'].apply(extract_heating_name)  
  
# Güncellenmiş veriyi göster  
tools.display_dataframe_to_user(name="Güncellenmiş Excel Verisi (housingComplex name)", dataframe=df_excel)
```

28- fee sütununun amount değerleri alınıp temizlendi.

```
# 'fee' sütununda sadece 'amount' değerini al  
df_final['fee'] = df_final['fee'].apply(extract_amount)  
  
# Güncellenmiş veriyi göster  
tools.display_dataframe_to_user(name="Güncellenmiş Veri (fee sadece amount)", dataframe=df_final)
```

29- room and Livingroom toplanarak sayısal hale getirildi.

```
# Tek tırnaklı liste içindeki stringlerden oda+salon toplamını çıkart  
def extract_sum_from_quoted_list(value):  
    try:  
        if isinstance(value, str):  
            # Örn: "[3+1]" → '3+1' → 4  
            value = value.strip("[]").replace("'", "").replace("+", " ").strip()  
            if "+" in value:  
                oda, salon = value.split("+")  
                return int(oda) + int(salon)  
            return None  
    except:  
        return None  
  
    # Uygula  
df_final['roomAndLivingRoom'] = df_final['roomAndLivingRoom'].apply(extract_sum_from_quoted_list)  
  
# Sonuçları tekrar göster  
tools.display_dataframe_to_user(name="roomAndLivingRoom (tek tırnaklı dize türündü)", dataframe=df_final)
```

30- Kategorik veriler excel aracılığıyla standart hale getirildi. Yazım yanlışları düzeltildi.

31- Kategorik verilerden boş olanlar “bilinmiyo” ile, sayısal olanlar ortalama ile dolduruldu.

```
# Kategorik ve sayısal sütunları ayıralım
categorical_cols = df.select_dtypes(include=['object']).columns
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Kategorik verilerde boş olanları "Bilinmiyor" ile doldur
df[categorical_cols] = df[categorical_cols].fillna("Bilinmiyor")

# Sayısal verilerde boş olanları ortalama ile doldur
df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())

# Temizlenmiş dosyayı kaydedelim
output_path = "/mnt/data/temizlenmis_veri.xlsx"
df.to_excel(output_path, index=False)

output_path
```

32- Aykırı değerlere sahip olan (realtyId , no, price,room, livingRoom) IQR yöntemi ile kırılmıştır (%5)

```
from scipy.stats.mstats import winsorize

# Aykırı bulunan sütunlar
columns_to_winsorize = list(outlier_info.keys())

# %5 Winsorizing uygulayalım (alt %5, üst %5)
for col in columns_to_winsorize:
    df[col] = winsorize(df[col], limits=[0.05, 0.05])

# Yeni dosya olarak kaydet
winsorized_path = "/mnt/data/winsorize_edilmiş_veri.xlsx"
df.to_excel(winsorized_path, index=False)

winsorized_path
```

33- Benzersiz kategori sayısı 5'ten az olan sütunlara One-Hot Encoding, ve daha fazla benzersiz değere sahip sütunlara ise Label Encoding uygulanmıştır.

```

from sklearn.preprocessing import LabelEncoder

# Kategorik sütunları yeniden belirle (bazıları sayısal olabilir)
categorical_cols = df.select_dtypes(include='object').columns

# One-hot ve Label encoding uygulanacak sütunları ayıralım
one_hot_cols = [col for col in categorical_cols if df[col].nunique() < 5]
label_encode_cols = [col for col in categorical_cols if df[col].nunique() >= 5]

# One-hot encoding
df = pd.get_dummies(df, columns=one_hot_cols)

# Label encoding
label_encoder = LabelEncoder()
for col in label_encode_cols:
    df[col] = label_encoder.fit_transform(df[col])

# Kodlanmış veriyi kaydet
encoded_path = "/mnt/data/encoded_veri.xlsx"
df.to_excel(encoded_path, index=False)

encoded_path

```

34- Sayısal verilere MIN-MAX ölçeklendirilmesi yapıldı. (Price hariç)

```

# Price dısındaki tüm sayısal sütunları alalım
numeric_cols_to_scale = df.select_dtypes(include=['float64', 'int64']).columns.drop('price', errors='ignore')

# Min-Max Ölçekleyiciyi uygula
scaler = MinMaxScaler()
df[numeric_cols_to_scale] = scaler.fit_transform(df[numeric_cols_to_scale])

# Ölçeklendirilmiş veriyi kaydet
scaled_path = "/mnt/data/minmax_scaled_veri.xlsx"
df.to_excel(scaled_path, index=False)

scaled_path

```

35- Korelasyon hesaplandı.Yüzde 90dan fazla olan ;

| Feature 1 | Feature 2 | Korelasyon Katsayısı |
|-------------------|------------------|----------------------|
| roomAndLivingRoom | room | 0.9763 |
| registerState | landRegisterName | 1.0000 |
| grossSqm | netSqm | 0.9352 |
| currency_TRY | currency_TL | -0.9319 |

Room, RegisterState Ve currency_TRY silindi. Sqm sütunu modelde işe yarayacağı öngörülerek çıkarılmıştır.

```
# Belirtilen sütunları veri setinden çıkar
columns_to_drop = ['room', 'registerState', 'currency_TRY']
df.drop(columns=columns_to_drop, inplace=True)

# Güncellenmiş veri setini kaydet
updated_path = "/mnt/data/yuksek_korelasyon_temizlenmis_veri.xlsx"
df.to_excel(updated_path, index=False)

updated_path
```

36- price sütununa logaritmik dönüşüm uygulandı ve sonuç log_price adında yeni bir sütun olarak eklendi.

```
# price sütununu logaritmik dönüşüme tabi tut (log1p: log(1 + x))
df['log_price'] = np.log1p(df['price'])

# Güncellenmiş veri setini kaydet
log_price_path = "/mnt/data/log_price_eklenmis_veri.xlsx"
df.to_excel(log_price_path, index=False)

log_price_path
```

Makine Öğrenmesi

Makine öğrenmesi algoritmaları olarak Random Forest, XGBoost, Naive Bayes, Logistic Regression, KNN ve Decision Tree kullanılmıştır. Aşağıda modeli eğiten kodlar yer almaktadır.

```
# Özellik ve hedef değişken
x = df.drop(columns=["price", "log_price"])
y_log = df["log_price"]
threshold = y_log.median()
y_class = (y_log >= threshold).astype(int)

# Eğitim ve test setlerine ayır
x_train, x_test, y_train, y_test = train_test_split(x, y_class, test_size=0.2, random_state=42)

# KNN için Ölçeklendirme yapılacak, diğerleri için standart X kullanacağız
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Modelleri tanımla
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Random Forest": RandomForestClassifier(random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
}

results = []

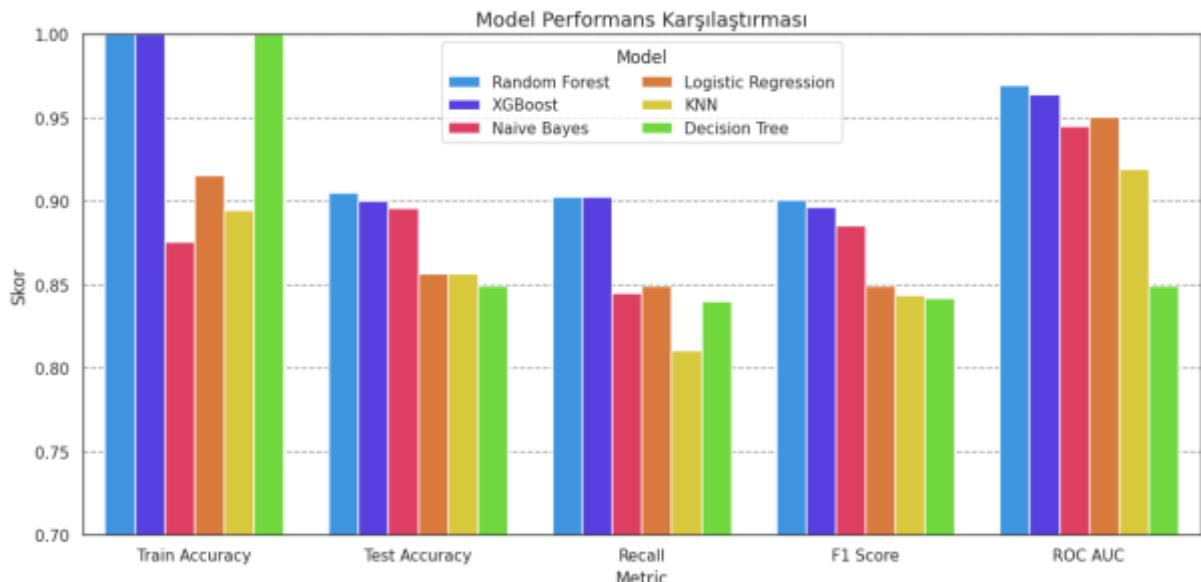
for name, model in models.items():
    print(f"Model çağrılıyor: {name}")
    # KNN için ölçeklendirilmiş veriyi kullan
    if name == "KNN":
        model.fit(x_train_scaled, y_train)
        y_pred_train = model.predict(x_train_scaled)
        y_pred_test = model.predict(x_test_scaled)
        y_proba_test = model.predict_proba(x_test_scaled)[:, 1]
    else:
        model.fit(x_train, y_train)
        y_pred_train = model.predict(x_train)
        y_pred_test = model.predict(x_test)
        y_proba_test = model.predict_proba(x_test)[:, 1]

    # Performans metrikleri
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    recall = recall_score(y_test, y_pred_test)
    f1 = f1_score(y_test, y_pred_test)
    roc_auc = roc_auc_score(y_test, y_proba_test)

    # Overfitting hesapla (train-test accuracy farkı)
    overfitting = train_acc - test_acc

    results.append({
        "Model": name,
        "Train Accuracy": train_acc,
        "Test Accuracy": test_acc,
        "Recall": recall,
        "F1 Score": f1,
        "ROC AUC": roc_auc,
        "Overfitting (Train - Test Acc)": overfitting
    })
```

| --- Modellerin Performans Karşılaştırması --- | | | | | | | |
|---|----------------|---------------|----------|----------|----------|--------------------------------|-----------|
| Model | Train Accuracy | Test Accuracy | Recall | F1 Score | ROC AUC | Overfitting (Train - Test Acc) | |
| Random Forest | 1.000000 | 0.905093 | 0.902913 | 0.900726 | 0.969671 | | 0.094907 |
| XGBoost | 1.000000 | 0.900463 | 0.902913 | 0.896386 | 0.964108 | | 0.099537 |
| Naive Bayes | 0.875579 | 0.895833 | 0.844660 | 0.885496 | 0.945163 | | -0.020255 |
| Logistic Regression | 0.915509 | 0.856481 | 0.849515 | 0.849515 | 0.950490 | | 0.059028 |
| KNN | 0.894676 | 0.856481 | 0.810680 | 0.843434 | 0.918990 | | 0.038194 |
| Decision Tree | 1.000000 | 0.849537 | 0.839806 | 0.841849 | 0.849106 | | 0.150463 |



İlk sonuçlara göre ;

1. En iyi modeller:

- Random Forest ve XGBoost, en yüksek test doğruluğu ve F1 skoruna sahiptir.
- Bu modeller overfitting eğilimi gösterse de (özellikle Decision Tree'ye kıyasla), bu durum çok ciddi seviyede değil.

2. Naive Bayes:

- Basit bir model olmasına rağmen oldukça dengeli sonuçlar vermiştir. Overfitting negatif çıkmıştır.

3. Logistic Regression & KNN:

- İyi alternatifler olarak düşünülebilir, fakat daha karmaşık modeller (RF, XGBoost) karşısında biraz geride kalmıştır.

4. Decision Tree:

- Overfitting problemi belirdir. Train verisinde mükemmel çalışıyor ama test verisinde kötüleşiyor. Bu klasik karar ağacı davranışıdır.

Sonuç olarak Decision Tree algoritmasındaki overfitting durumunun çözülmesi ve Random forest ile XGBoost modellerinin performanslarının artırılması için GridSearchCV hiperparametre optimizasyonu yapılmıştır.

```

# --- Random Forest
rf_params = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2],
    "max_features": ["sqrt", "log2"]
}
rf = RandomForestClassifier(random_state=42)
rf_grid = GridSearchCV(rf, rf_params, cv=5, scoring="f1", n_jobs=-1, verbose=1)
rf_grid.fit(X_train, y_train)

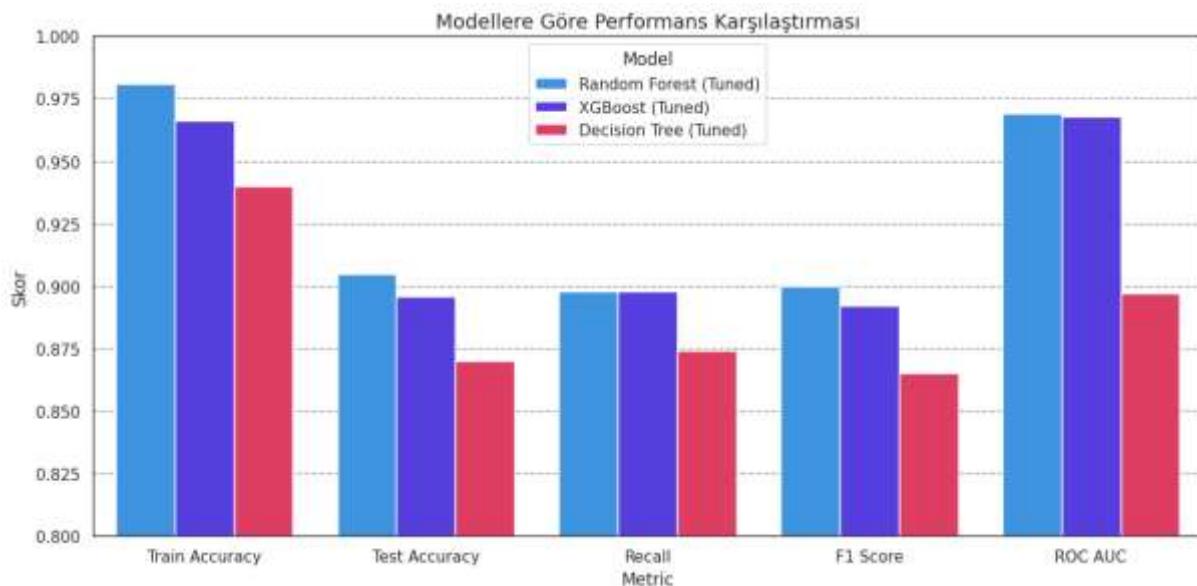
# --- XGBoost
xgb_params = {
    "n_estimators": [100, 200],
    "max_depth": [3, 6, 10],
    "learning_rate": [0.01, 0.1],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0]
}
xgb = XGBClassifier(use_label_encoder=False, eval_metric="logloss", random_state=42)
xgb_grid = GridSearchCV(xgb, xgb_params, cv=5, scoring="f1", n_jobs=-1, verbose=1)
xgb_grid.fit(X_train, y_train)

# --- Decision Tree
dt_params = {
    "max_depth": [None, 5, 10, 20],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "criterion": ["gini", "entropy"]
}
dt = DecisionTreeClassifier(random_state=42)
dt_grid = GridSearchCV(dt, dt_params, cv=5, scoring="f1", n_jobs=-1, verbose=1)
dt_grid.fit(X_train, y_train)

```

Hiperparametre optimizasyonu yapıldıktan sonra aşağıdaki sonuçlar alınmıştır.

| --- Hiperparametre Ayarlı Modellerin Performansı --- | | | | | | | |
|--|----------------|---------------|--------|----------|---------|--------------------------------|-------|
| Model | Train Accuracy | Test Accuracy | Recall | F1 Score | ROC AUC | Overfitting (Train - Test Acc) | |
| Random Forest (Tuned) | 0.981 | 0.905 | 0.898 | 0.900 | 0.969 | | 0.076 |
| XGBoost (Tuned) | 0.962 | 0.896 | 0.893 | 0.891 | 0.968 | | 0.067 |
| Decision Tree (Tuned) | 0.940 | 0.870 | 0.874 | 0.865 | 0.897 | | 0.070 |



Sonuç olarak;

1. Random Forest (Tuned)

- En iyi **F1 Skoru** ve **Test Doğruluğu**.
- ROC AUC değeri çok yüksek → sınıfları ayırma başarısı güçlündür.
- Hafif overfitting var (0.981 vs 0.905), ama kabul edilebilir düzeydedir.
- Genel olarak en iyi modeldir.**

◆ 2. XGBoost (Tuned)

- Performansı Random Forest'a çok yakındır.
- Overfitting oranı daha düşük (0.067), bu da **daha genel geçer** bir model olduğunu gösterir.

◆ 3. Decision Tree (Tuned)

- Hiperparametre tuning sonrasında overfitting ciddi şekilde azaltılmıştır.
- Fakat genel başarı metrikleri diğer iki modele göre daha düşüktür.

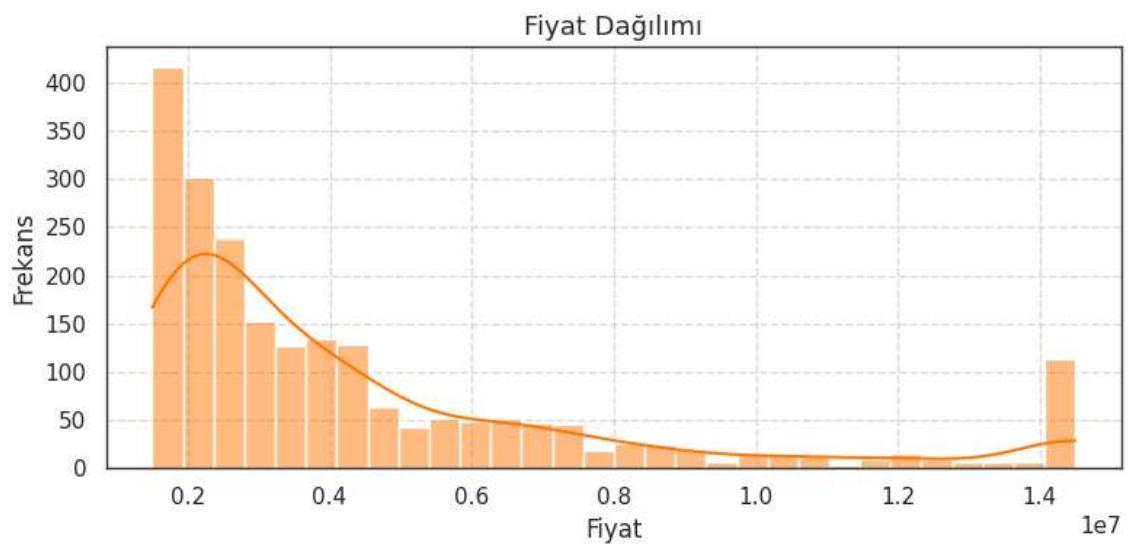
Özellik Mühendisliği

- Net metrekare başına fiyatı içerecek şekilde yeni bir özellik oluşturulmuş ve modele katkılarına bakılmıştır.

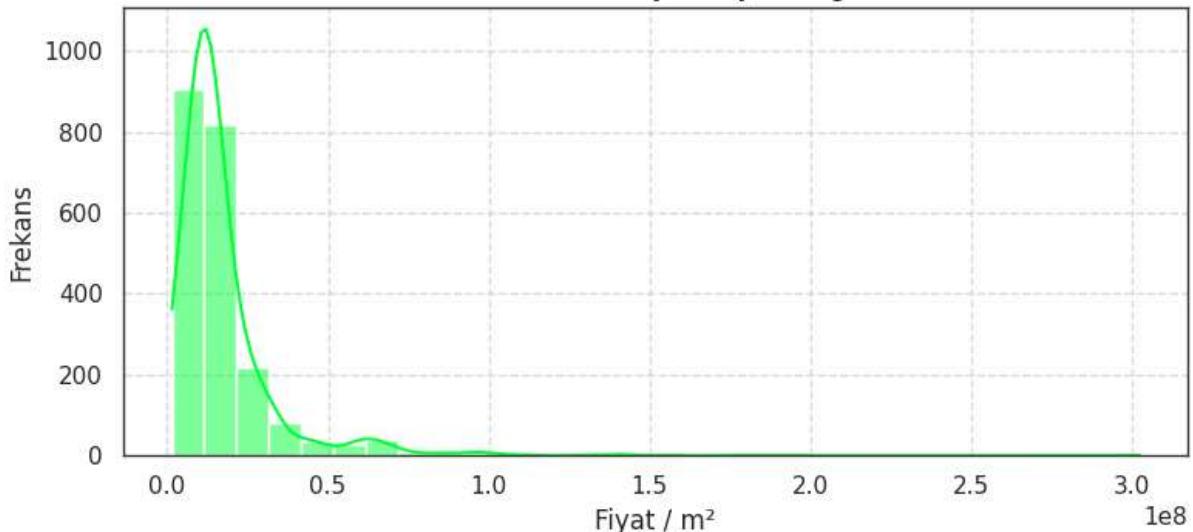
```
df["price_per_sqm"] = df["price"] / df["netSqm"].replace(0, 1)
```

| Model | Accuracy | Recall | F1 Score | ROC AUC |
|---------------------|----------|--------|----------|---------|
| Random Forest | 0.933 | 0.942 | 0.930 | 0.983 |
| XGBoost | 0.965 | 0.971 | 0.964 | 0.995 |
| Naive Bayes | 0.544 | 0.937 | 0.662 | 0.691 |
| Logistic Regression | 0.870 | 0.850 | 0.862 | 0.948 |
| KNN | 0.850 | 0.806 | 0.836 | 0.914 |
| Decision Tree | 0.926 | 0.942 | 0.924 | 0.927 |

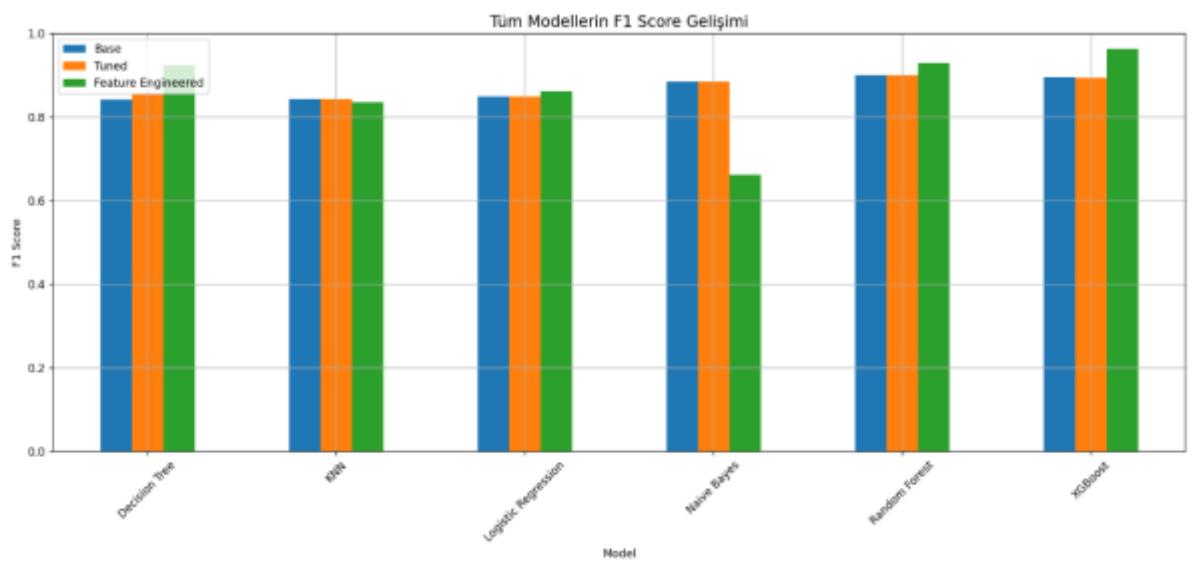
- Eklenen yeni özellik sonucunda Random Forest, XGBoost, Decision Tree ve Logistic Regression algoritmalarının performanslarında artış gözlenmiştir. Fakat KNN algoritmasında hafif bir düşüş yanında Naive Bayes algoritmasında ciddi bir düşüş yaşanmıştır. Bunun sebebi ise eklenen Naive Bayes algoritmasının verilerin normal dağıldığını varsaymasından kaynaklanmaktadır. Fakat yeni eklediğimiz sütunda veriler büyük varyansa sahip olduğundan bu varsayıımı çürütmektedir.



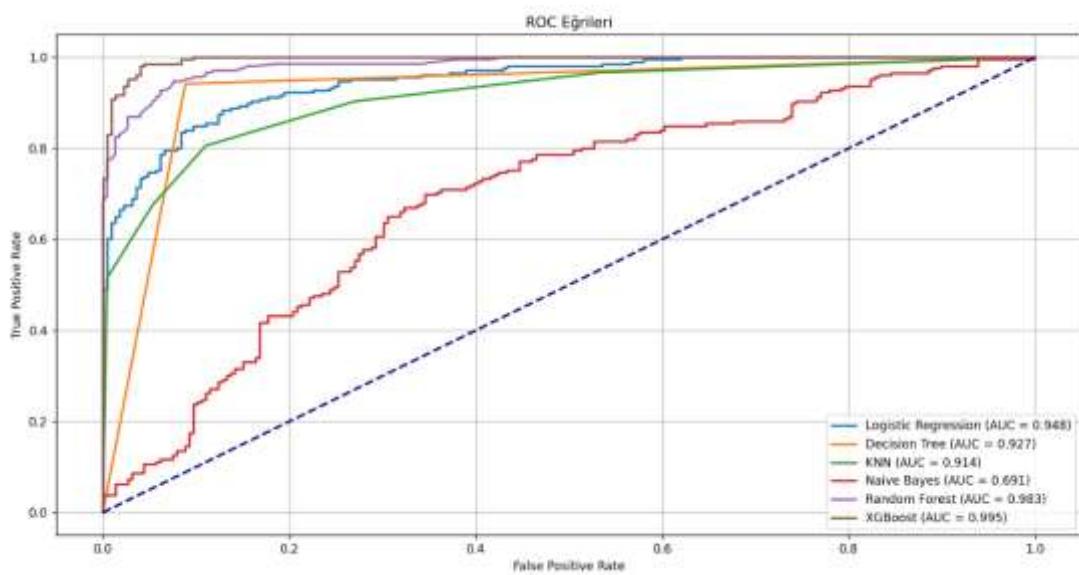
Net Metrekare Başına Fiyat Dağılımı



Göründüğü üzere sütundaki veriler sağa çarpık şekilde dağılmaktadır. Bu yüzden Naive Bayes modelinde F1 skorunda ciddi bir düşüş yaşanmıştır.

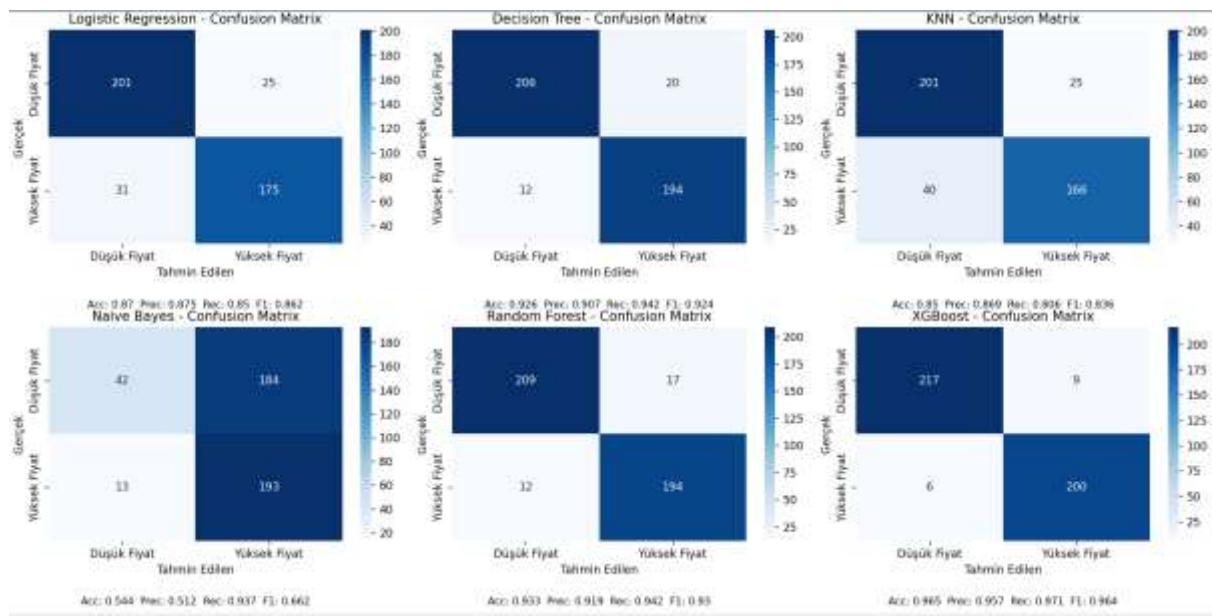


Bu grafikte temel algoritmalar, hiperparametre optimizasyonu sonrası ve özellik eklenmiş hallerindeki F1 skoru değişimleri net bir şekilde görülmektedir.



Tüm modellerin ROC eğrileri, temel olarak mavi kesikli “şans çizgisi”nin (AUC=0.5) oldukça üzerinde. Bu da modellerin rastgele tahminden çok daha iyi performans gösterdiğini ifade etmektedir.

XGBoost (AUC = 0.995) ve **Random Forest (AUC = 0.983)** neredeyse mükemmel yakın ayırt etme başarısına sahiptir. Gerçek sınıf etiketlerini ayırmada son derece güçlündür.



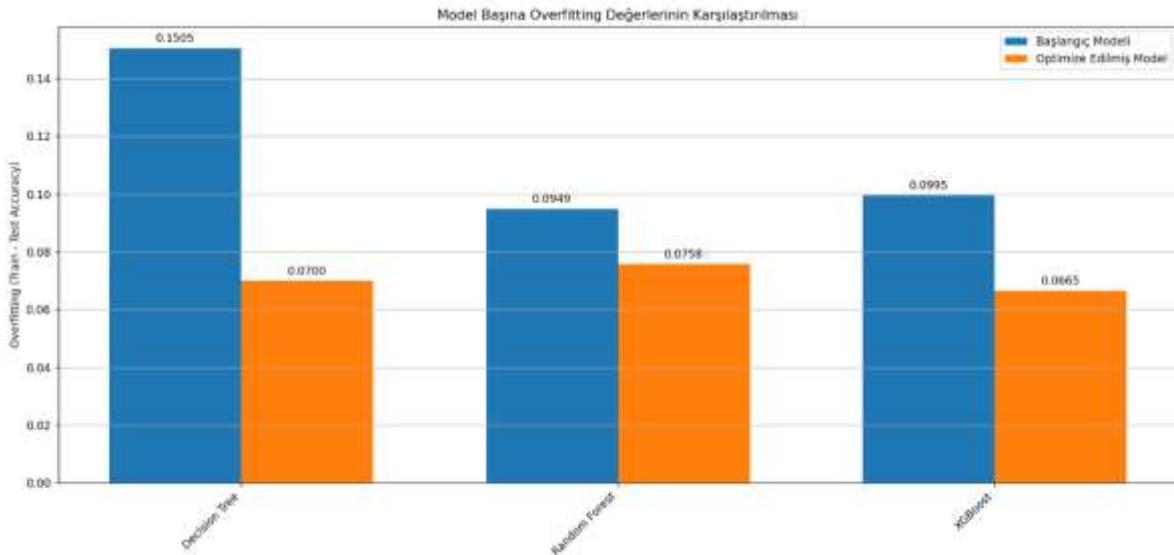
1. **XGBoost, Random Forest ve Decision Tree** gibi modellerin karmaşıklık matrislerinde doğru sınıflandırma sayıları oldukça fazladır.

- Bu, modellerin hem **düşük fiyatlı** hem de **yüksek fiyatlı** evleri ayırt etmede oldukça başarılı olduğunu gösteriyor.

2. Çoğu modelde **düşük ve yüksek fiyatlı evler için dengeli bir başarı** var (True Positive \approx True Negative).

3. Ancak bazı modellerde bu denge bozuluyor:

- **Naive Bayes**, yüksek fiyatlı evleri çok iyi tahmin ederken (193 doğru), düşük fiyatlıları tanıtmakta başarısız (sadece 42 doğru).
- Bu, modelin **bir sınıfı aşırı eğilimli** olduğunu, yani **dengesiz tahmin yaptığı**nı gösteriyor.



Bu grafikte ise gridsearchCV hiperparametre optimizasyonu sonucunda overfitting durumlarındaki değişim görünmektedir. Decision Tree algoritmasındaki overfittingin hiperparametre optimizasyonu sonucunda düşüğü görülmektedir.