SAKARYA UYGULAMALI BİLİMLER ÜNİVERSİTESİ

VERİ MADENCİLİĞİ PROJE ÖDEVİ

DERS KODU: BIL 018
ÖĞRETİM GÖREVLİSİ: DR. ÖĞR. ÜYESİ MUHAMMED ALİ NUR ÖZ
AD – SOYAD: AYŞENUR YILDIZ
NO: B200109026
FAKÜLTE: TEKNOLOJİ FAKÜLTESİ
BÖLÜM: BİLGİSAYAR MÜHENDİSLİĞİ

İçindekiler

Şekiller

1. Giriş

Günümüzde ikinci el araç pazarı geniş bir veri yelpazesine sahiptir. Bu nedenle ikinci el araç satıcıları için doğru alım kararı vermek, araç özelliklerine göre doğru fiyat belirlemek önemlidir. Bu gereksinimlerden yola çıkarak ikinci el araç satıcılarının bilinçli alım kararları alabilmelerine ve kârlarını en üst düzeye çıkarmalarına yardımcı olmak üzere veri madenciliği tekniklerini kullanan bir sistem geliştirmek amaçlanmıştır.

Bu projede, Python programlama dili ve veri bilimi alanındaki popüler kütüphaneler kullanılarak, Sahibinden.com üzerinden çekilerek elde edilen veri seti üzerinde çalışılmıştır. Bu veri seti, araçların konum, fiyat, model, üretim yılı, kilometre gibi çeşitli özelliklerini içermektedir. Bu veri üzerinden analizler gerçekleştirerek öneri sistemi geliştirilmiştir. Projenin odak noktası yapay zeka modeli tarafından işlenebilecek şekilde verileri temizlemek, işlemek ve modellemek suretiyle, ikinci el araç satıcıları için en uygun satın alma fiyatını öneren bir yazılım geliştirmektir.

2. Veri Ön İşleme

2.1 Veriyi Yükleme

Ödev kapsamında bize verilen veri setini Google Drive'a yükledim ve Colab platformunda bu veri setini işlemek için gerekli adımları gerçekleştirdim. Gerekli kütüphaneleri import ettikten sonra veri setini Pandas kütüphanesini kullanarak bir dataframe'e dönüştürdüm ve 'veri' adını verdiğim değişkene atadım. İlk olarak, verinin yapısını anlamak için dataframe'in ilk 5 satırını yazdırdım. Bu adım, veriyi incelemeye başlamak için temel bir adımdı ve veri setinin genel yapısını anlamama yardımcı oldu.

```
[2]  # Colab'de Google Drive bağlantısını yap
     from google.colab import drive
     drive.mount("/content/drive")

     Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

[3]  # Veriyi yükle
     veri_yolu = "/content/drive/MyDrive/VeriMadenciligi/arac_ilanlari.csv"

     # Veriyi DataFrame'e yükle
     veri = pd.read_csv(veri_yolu)
```

Şekil 1 Verinin yüklenmesi

```
# Verinin ilk 5 satırını göster
veri.head()
```

	ilan_id	ilan_basligi	fiyat	fiyat_kuru	ilan_tarihi	ilan_kategorisi	arac_tagleri	ilan_konumu	arac_ozellikleri
0	1080850207	DEĞİŞENSİZ YENİ MUAYENELİ RENAULT MEGANE FUL D...	830000.0	TL	2023-05-26 09:23:29.000000	[{'id': 3517, 'label': 'Vasıta'}, {'id': 3530,...	{'Model': '1.5 dCi Icon', 'Marka': 'Renault', ...	İstanbul, Ümraniye	['143.000 km', '2016 model', 'Dizel', '110 hp'...
1	1101850722	KAZA,BOYA,DEĞİŞEN,TRAMER YOK. 80.000 km.	905000.0	TL	2023-05-26 09:23:13.000000	[{'id': 3517, 'label': 'Vasıta'}, {'id': 3530,...	{'Model': '1.5 dCi Icon', 'Marka': 'Renault', ...	İzmir, Konak	['80.000 km', '2017 model', 'Dizel', '110 hp',...
2	1101855061	HATASIZ AYARINDA MASRAFSIZ AUDİ A3	695000.0	TL	2023-05-26 09:22:58.000000	[{'id': 3517, 'label': 'Vasıta'}, {'id': 3530,...	{'Model': 'A3 Sportback 1.6', 'Marka': 'Audi',...	Adana, Seyhan	['177.000 km', '2011 model', 'Benzin & LPG', '...
3	1101851921	2020/A3 35 1.5 TFSİ/150 DESIGN+STRONIC PKT-SUN...	1380000.0	TL	2023-05-26 09:22:35.000000	[{'id': 3517, 'label': 'Vasıta'}, {'id': 3530,...	{'Model': 'A3 Sedan 35 TFSI', 'Marka': 'Audi',...	İstanbul, Sarıyer	['33.000 km', '2020 model', 'Benzin', '150 hp'...
4	1090594643	2015 MODEL	530000.0	TL	2023-05-26 09:21:55.000000	[{'id': 3517, 'label': 'Vasıta'}, {'id': 3530,...	{'Model': '1.4 TDI Trendline', 'Marka': 'Volks...	Kırklareli, Lüleburgaz	['125.000 km', '2015 model', 'Dizel', '75 hp',...

Şekil 2 Verinin ilk 5 satırının gösterilmesi

1.2. Verilerin İncelenmesi

Veri setinin genel özelliklerini elde etmek amacıyla 'check_df' adını verdiğim bir fonksiyon yazdım ve bu fonksiyonu kullanarak veri çerçevesinin yapısal özelliklerini inceledim. İşlenen

veri çerçevesinde toplam 556,756 satır ve 9 sütun bulunuyor. Sütunların veri tipleri incelendiğinde, 'ilan_id' sütunu tam sayı (int64) formatında iken 'fiyat' sütunu ondalıklı sayı (float64) formatında bulunuyor. Diğer sütunlar genellikle string veya metin formatında veri içeriyor. Bunun yanı sıra, veri setinde eksik veri tespit edilmedi; yani sütunlarda null değer bulunmuyor. Bu durum, veri setinin eksiksiz olduğunu gösteriyor. Son olarak, 'Quantiles' bölümü, 'ilan_id' için minimum ve maksimum değerleri, 'fiyat' sütunu için ise yüzdelik dilimlerdeki değerleri gösteriyor.

```python
# Veri seti için genel bilgiler
def check_df(dataframe, head=5):
    print("##################### Shape #####################")
    print(dataframe.shape)
    print("##################### Types #####################")
    print(dataframe.dtypes)
    print("##################### Head #####################")
    print(dataframe.head(head))
    print("##################### Tail #####################")
    print(dataframe.tail(head))
    print("##################### NA #####################")
    print(dataframe.isnull().sum())
    print("##################### Quantiles #####################")
    print(dataframe.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)
```

```python
check_df(veri, head=5)
```

Şekil 3 Veri seti için genel bilgileri elde etmeye yarayan check_df fonksiyonu

Kategorik sütunlar, genellikle metinsel veya kategorik değerler içeren sütunlardır ve geniş bir veri çeşitliliğini temsil ederler. Bu sütunlar, genellikle gruplandırma, segmentasyon veya sınıflandırma gibi işlemlerde kullanılır.

Veri setindeki kategorik sütunlar şunlardır:

- ilan_basligi
- fiyat_kuru
- ilan_tarihi
- ilan_kategorisi
- arac_tagleri
- ilan_konumu
- arac_ozellikleri

Öte yandan, sayısal sütunlar, sayısal değerler içeren ve matematiksel işlemlere uygun olan sütunlardır. Bu tür sütunlar genellikle istatistiksel analizler, regresyon veya yapay zeka modelleri gibi sayısal işlemler için kullanılır.

Veri setindeki sayısal sütunlar ise şunlardır:

- ilan_id
- fiyat

```python
# Kategorik sütunları seçme
kategorik_sutunlar = veri.select_dtypes(include=['object']).columns.tolist()
print(kategorik_sutunlar)
```

```
['ilan_basligi', 'fiyat_kuru', 'ilan_tarihi', 'ilan_kategorisi', 'arac_tagleri', 'ilan_konumu', 'arac_ozellikleri']
```

```python
# Nümerik sütunları seçme
numerik_sutunlar = veri.select_dtypes(include=['int64', 'float64']).columns.tolist()
print(numerik_sutunlar)
```

```
['ilan_id', 'fiyat']
```

Şekil 4 Veri setindeki sayısal ve kategorik sütunların seçimi

Veri setindeki 'arac_tagleri' sütununda her bir satırdaki veri sözlük yapısında bulunuyordu ve bu durum işlemleri zorlaştırıyordu. Bu nedenle, her bir satırdaki sözlük yapısını ele alarak Python'un **ast** kütüphanesinde yer alan **literal_eval** fonksiyonunu kullandım. Bu fonksiyon, metin formatındaki Python veri yapısını gerçek bir Python veri yapısına dönüştürmeme yardımcı oldu. 'arac_tagleri' sütunundaki her bir satırdaki sözlük yapısını böylece analiz edebildim.

Bu analizde, her bir sözlük yapısındaki 'Model' anahtarına karşılık gelen değeri çıkararak yeni bir 'Araç_Modeli' adında bir sütun oluşturdum. Bu işlem, 'arac_tagleri' sütunundaki verileri daha işlenebilir hale getirmeme ve analiz için daha uygun hale getirmeme olanak sağladı.

```python
import ast

# 'arac_tagleri' sütunundaki sözlük yapısını parse etme
veri['arac_tagleri'] = veri['arac_tagleri'].apply(ast.literal_eval)

# 'Model' anahtarına karşılık gelen değerleri çıkararak yeni bir sütun oluşturma
veri['Araç_Modeli'] = veri['arac_tagleri'].apply(lambda x: x['Model'])
```

```python
# Araç modellerinin sayısını yazdırma
model_sayisi = len(veri['Araç_Modeli'].unique())
print(f"Araç modeli sayısı: {model_sayisi}")
```

```
Araç modeli sayısı: 266
```

Şekil 5 'arac_tagleri' sütununu parse ederek Araç_Modeli sütununu oluşturma

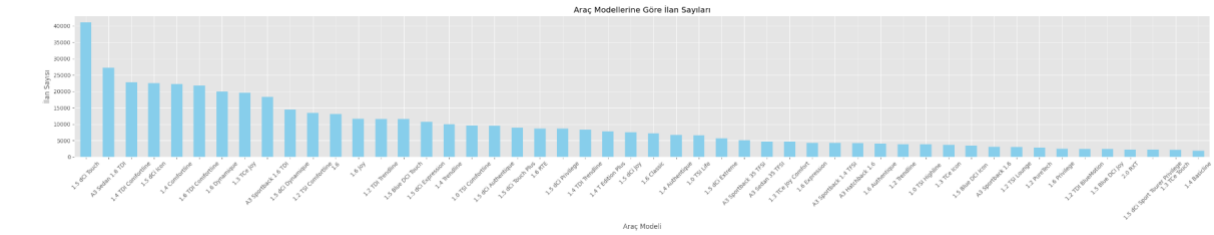
Elde ettiğim 'Araç_Modeli' sütunu üzerinden yapılan görselleştirme çalışması sonucunda, araç modellerinin ilan sayılarını içeren bir grafik oluşturdum. Bu grafikte X ekseni, farklı araç modellerini temsil ederken Y ekseni ise her bir araç modeline karşılık gelen ilan sayısını göstermektedir. Grafik, en yüksek ilan sayısına sahip olan 50 araç modelini listelemektedir.

Grafiği incelediğimde, en yüksek ilan sayısına sahip olan ilk üç araç modeli şunlardır:

1. 1.5 dCi Touch
2. A3 Sedan 1.6 TDI
3. 1.4 TDI Comfortine

Bu veriler, hangi araç modellerinin daha popüler olduğunu veya hangi modellerin daha sık tercih edildiğini anlamam için referans noktası oluşturmaktadır.

```python
import matplotlib.pyplot as plt

# Araç modellerine göre ilan sayılarını hesaplama
ilan_sayilari = veri['Araç_Modeli'].value_counts()

# Grafik oluşturma
plt.figure(figsize=(30, 6))
ilan_sayilari.head(50).plot(kind='bar', color='skyblue')
plt.title('Araç Modellerine Göre İlan Sayıları')
plt.xlabel('Araç Modeli')
plt.ylabel('İlan Sayısı')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



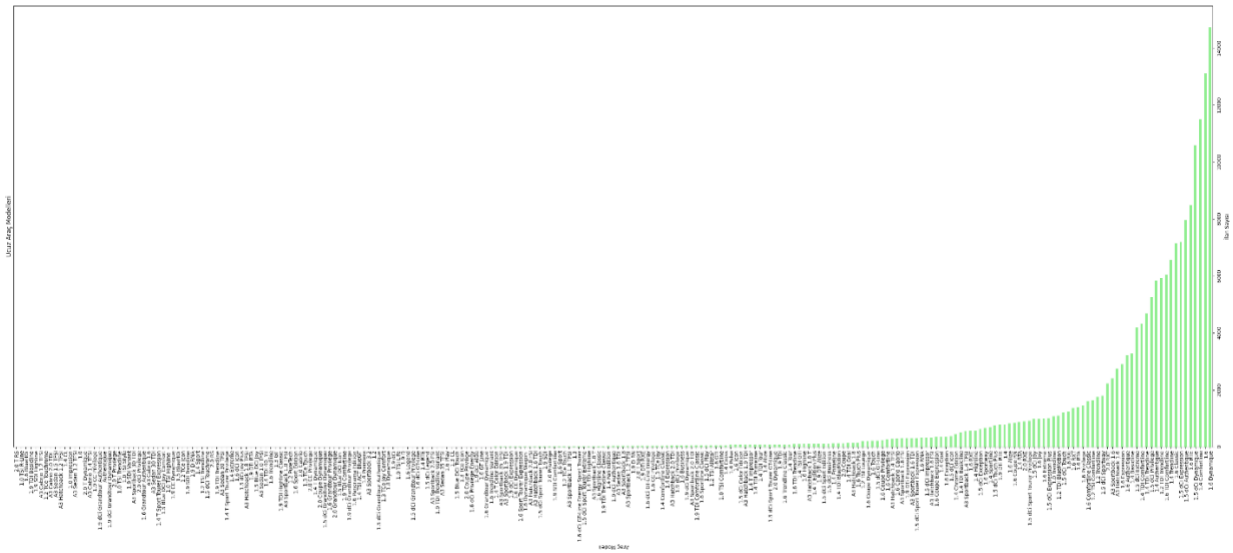
Şekil 6 Araç modellerine göre ilan sayısını gösteren grafik ve kodu

Araç fiyatlarını üç kategoriye - ucuz, orta ve pahalı - ayırdım ve her bir araç modelini bu kategorilere göre sınıflandırdım. Daha sonra her kategoriye ait araç modeli sayılarını içeren grafikler oluşturdum. Bu grafikler ile, her fiyat kategorisine ait araç modeli sayılarını inceledim. Bu inceleme, hangi fiyat aralığında daha fazla araç modelinin bulunduğunu anlamam için faydalı oldu.

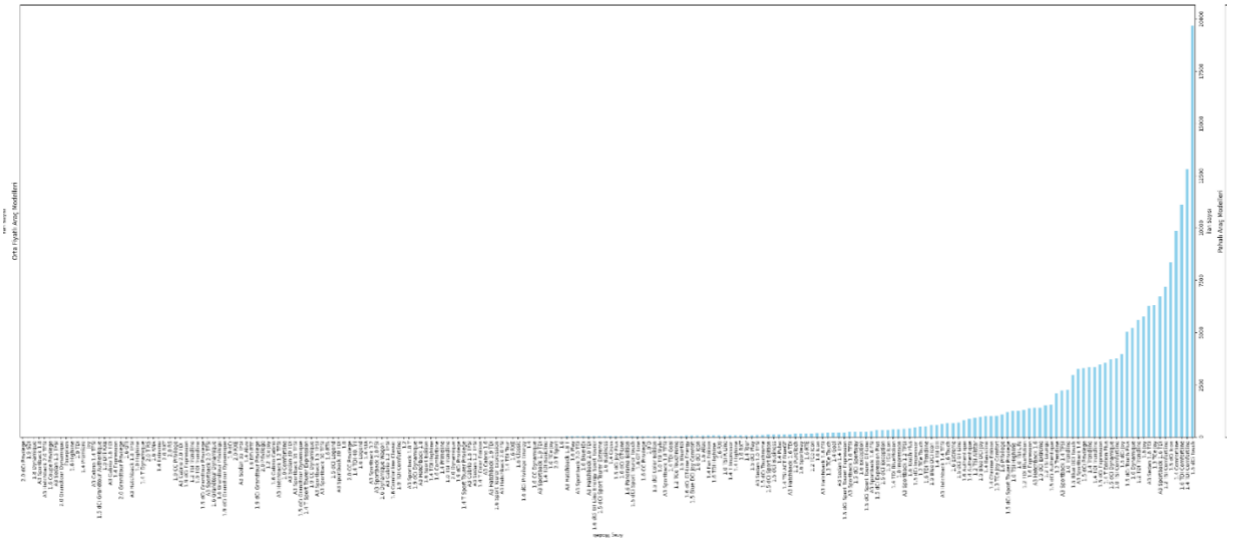
Bu çalışma, fiyat kategorilerine göre araç modeli sayılarını karşılaştırarak fiyat segmentlerinin araç çeşitliliği üzerindeki etkisini anlamama yardımcı oldu.

```python
# Fiyatları kategorilere göre ayırma
ucuz = veri[veri['fiyat'] < veri['fiyat'].quantile(1/3)]
orta = veri[(veri['fiyat'] >= veri['fiyat'].quantile(1/3)) & (veri['fiyat'] < veri['fiyat'].quantile(2/3))]
pahali = veri[veri['fiyat'] >= veri['fiyat'].quantile(2/3)]

# Her kategori için ayrı grafikler oluşturma
plt.figure(figsize=(15, 100))

plt.subplot(3, 1, 1)
ucuz['Araç_Modeli'].value_counts().plot(kind='barh', color='lightgreen')
plt.title('Ucuz Araç Modelleri')
plt.xlabel('İlan Sayısı')
plt.ylabel('Araç Modeli')

plt.subplot(3, 1, 2)
orta['Araç_Modeli'].value_counts().plot(kind='barh', color='skyblue')
plt.title('Orta Fiyatlı Araç Modelleri')
plt.xlabel('İlan Sayısı')
plt.ylabel('Araç Modeli')

plt.subplot(3, 1, 3)
pahali['Araç_Modeli'].value_counts().plot(kind='barh', color='salmon')
plt.title('Pahalı Araç Modelleri')
plt.xlabel('İlan Sayısı')
plt.ylabel('Araç Modeli')

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

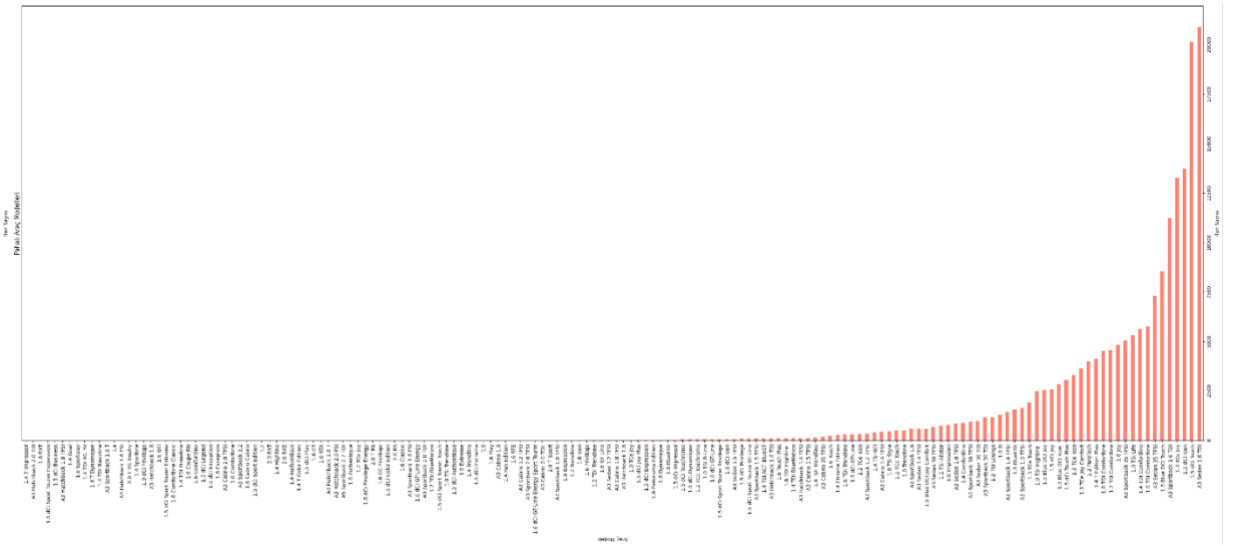
Şekil 7 Araç fiyatlarını üç kategoriye - ucuz, orta ve pahalı - ayırma kodu



Şekil 8 Ucuz olarak sınıflandırılan araç modeli sayılarını gösteren grafik



Şekil 9 Orta olarak sınıflandırılan araç modeli sayılarını gösteren grafik



Şekil 10 Pahalı olarak sınıflandırılan araç modeli sayılarını gösteren grafik

'arac_tagleri' sütunundaki verilerden 'Marka' anahtarına karşılık gelen değerleri çıkartarak yeni bir 'Araç_Markası' sütunu oluşturdum. Daha sonra her bir benzersiz araç markasını belirlemek için veri setinde yer alan 'Araç_Markası' sütununu inceledim.

Bu analiz sonucunda, veri setinde toplamda 6 farklı araç markasının bulunduğunu gözlemledim. Bu markalar şunlardır: 'Renault', 'Audi', 'Volkswagen', 'Opel', 'Citroën' ve 'Isuzu'.

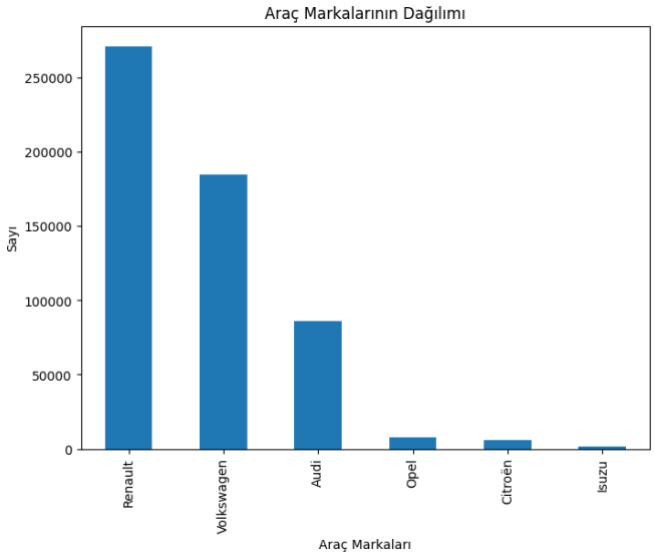
Bu bilgi, veri setindeki araç markalarının çeşitliliğini ve hangi markaların temsil edildiğini anlamamı sağladı.

```
veri['Araç_Markası'] = veri['arac_tagleri'].apply(lambda x: x.get('Marka') if 'Marka' in x else None)
```

```
[19] unique_markalar = veri['Araç_Markası'].nunique()
     print("Unique araç markası sayısı:", unique_markalar)
     unique_markalar = veri['Araç_Markası'].unique()
     print("Araç markaları:")
     print(unique_markalar)

     Unique araç markası sayısı: 6
     Araç markaları:
     ['Renault' 'Audi' 'Volkswagen' 'Opel' 'Citroën' 'Isuzu' None]
```

Şekil 11 'Araç_Markası' sütununun oluşturulması ve veri setindeki markaların gözlemlenmesi

Her bir araç markasının veri setindeki dağılımını gösteren bir grafik oluşturdum. Bu grafik, her markanın veri setinde ne kadar sıklıkta bulunduğunu inceledim.



Şekil 12 Araç markalarının dağılımı

1.3. Veri Ön İşleme

Veri ön işleme aşamasında yapılan adımlar genellikle verinin daha kullanışlı ve işlenebilir bir hale getirilmesini sağlamak için yapılmaktadır. Bunun için ilk olarak, 'Araç_Seri' adında yeni bir sütun oluşturdum ve 'arac_tagleri' sütunundan 'Seri' anahtarına karşılık gelen değerler bu sütuna aktardım. Ardından, 'arac_tagleri' sütunu veri setinden sildim.

Daha sonra, 'ilan_kategorisi' sütunu da veri setinden çıkarttım. 'arac_ozellikleri' sütunundaki veriler için önce köşeli parantezleri temizledim ardından virgülle ayrılan özellikleri

gruplayarak ayrı sütunlara dönüştürdüm. Bu yeni sütunları 'Kilometre', 'Model_Yili', 'Yakit_Tipi', 'Motor_Gucu' ve 'Vites_Tipi' olarak adlandırdım.

Son olarak, veri seti bu yeni sütunlarla birleştirdim ve 'arac_ozellikleri' sütununu sildim.

Bu ön işleme adımları, veri setinin daha düzenli hale gelmesini ve analiz için daha uygun bir formata dönüşmesini sağladı. Veri setindeki özelliklerin daha ayrıntılı incelenmesine ve daha doğru analizler yapmama olanak sağladı.

```
[21] veri['Araç_Seri'] = veri['arac_tagleri'].apply(lambda x: x.get('Seri') if 'Seri' in x else None)
```

Şekil 13 Araç_Seri sütununun oluşturulması

```
# 'arac_tagleri' sütununu sil
veri.drop(columns=['arac_tagleri'], inplace=True)
```

Şekil 14 arac_tagleri sütununun silinmesi

```
[25] # Parantezleri kaldırma ve virgüle göre ayrılan özellikleri gruplama
     veri['arac_ozellikleri'] = veri['arac_ozellikleri'].str.replace(r'[\[\]]', '')  # Köşeli parantezleri kaldırma
     ozellikler = veri['arac_ozellikleri'].str.split(', ', expand=True)

     # Virgüle göre ayrılan özellikleri gruplayarak ayırma
     ozellikler = veri['arac_ozellikleri'].str.split(', ', expand=True)

     # Sütun adlarını belirleme
     ozellikler.columns = ['Kilometre', 'Model_Yili', 'Yakit_Tipi', 'Motor_Gucu', 'Vites_Tipi']

     # Verileri birleştirme
     veri = pd.concat([veri, ozellikler], axis=1)
```

Şekil 15 Verideki "[ ]" gibi gereksiz bileşenlerin kaldırılması

```
[28] veri['Kilometre'] = veri['Kilometre'].str.replace(r"[^\w\s]", "")
     veri['Model_Yili'] = veri['Model_Yili'].str.replace(r"[^\w\s]", "")
     veri['Yakit_Tipi'] = veri['Yakit_Tipi'].str.replace(r"[^\w\s]", "")
     veri['Motor_Gucu'] = veri['Motor_Gucu'].str.replace(r"[^\w\s]", "")
     veri['Vites_Tipi'] = veri['Vites_Tipi'].str.replace(r"[^\w\s]", "")
```

Şekil 16 " ' ' " gibi bileşenlerin kaldırılarak verilen sütunların sayısal hale getirilmesi

'fiyat_kuru' sütununda yer alan benzersiz değerleri görüntülemek için 'unique()' fonksiyonunu kullandım. Bu adımın sonucunda, 'fiyat_kuru' sütununda yalnızca 'TL' değerlerinin olduğunu belirledim.

Daha sonra, veri setinden gereksiz veya tekrarlayan bilgileri temizlemek amacıyla 'fiyat_kuru' sütununu 'drop()' fonksiyonu kullanarak sildim.

Bu tür bir işlem, analiz sürecinde veri setindeki gereksiz veya tekrarlayan bilgileri temizlemek ve veri setini daha basit ve odaklı hale getirmek için sıkça kullanılır. 'fiyat_kuru' sütunundaki yalnızca 'TL' değerlerinin olması, bu sütunun analizimiz için gerekli olmadığını gösterdiği için bu adımı uyguladım.

```
[30] unique_fiyat_kuru = veri['fiyat_kuru'].unique()
     print(unique_fiyat_kuru)

     ['TL']
```

```
# 'fiyat_kuru' sütununu sil
veri.drop(columns=['fiyat_kuru'], inplace=True)
veri.head()
```

	ilan_id	ilan_basligi	fiyat	ilan_tarihi	ilan_konumu	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Model_Yili	Yakit_Tipi	Motor_Gucu	Vites_Tipi
0	1080850207	DEĞİŞENSİZ YENİ MUAYENELİ RENAULT MEGANE FUL D...	830000.0	2023-05-26 09:23:29.000000	İstanbul, Ümraniye	1.5 dCi Icon	Renault	Megane	143000 km	2016 model	Dizel	110 hp	Otomatik
1	1101850722	KAZA,BOYA,DEĞİŞEN,TRAMER YOK. 80.000 km.	905000.0	2023-05-26 09:23:13.000000	İzmir, Konak	1.5 dCi Icon	Renault	Megane	80000 km	2017 model	Dizel	110 hp	Otomatik
2	1101855061	HATASIZ AYARINDA MASRAFSIZ AUDİ A3	695000.0	2023-05-26 09:22:58.000000	Adana, Seyhan	A3 Sportback 1.6	Audi	A3	177000 km	2011 model	Benzin LPG	102 hp	Manuel
3	1101851921	2020/A3 35 1.5 TFSİ/150 DESIGN+STRONIC PKT-SUN...	1380000.0	2023-05-26 09:22:35.000000	İstanbul, Sarıyer	A3 Sedan 35 TFSI	Audi	A3	33000 km	2020 model	Benzin	150 hp	Otomatik
4	1090594643	2015 MODEL	530000.0	2023-05-26 09:21:55.000000	Kırklareli, Lüleburgaz	1.4 TDI Trendline	Volkswagen	Polo	125000 km	2015 model	Dizel	75 hp	Manuel

Şekil 17 fiyat_kuru sütununun incelenmesi ve kaldırılması

Veri setindeki gereksiz veya dağınık bilgileri temizleyerek, analiz sürecimin daha düzenli ve anlaşılır olmasını hedefledim bunun için şu adımları gerçekleştirdim:

1. **'ilan_basligi' sütununu veri setinden silme:** Analiz sürecimde 'ilan_basligi' sütununun ihtiyaç duymayacağım veya analizim için gereksiz olduğunu belirledim.
2. **'ilan_konumu' sütunundaki verileri 'il' ve 'ilce' olarak ayrıştırma:** 'ilan_konumu' sütunundaki verileri coğrafi olarak 'il' (şehir) ve 'ilce' (ilçe) şeklinde ayrıştırdım. Bu işlem, coğrafi analizler yapabilmek, lokasyon bazlı filtrelemeler veya gruplamalar yapabilmek adına veriyi daha anlamlı ve kullanışlı hale getirdi.
3. **'ilan_konumu' sütununu veri setinden silme:** 'ilan_konumu' sütununun artık 'il' ve 'ilce' olarak ayrıştırılmış verileri içeren yeni sütunlar oluşturulduktan sonra gereksiz hale geldiğini belirledim. Bu nedenle, orijinal 'ilan_konumu' sütununu veri setinden çıkardım.
4. **'ilan_tarihi' sütunundaki verileri zaman damgasına dönüştürme:** 'ilan_tarihi' sütunundaki tarih verilerini zaman damgası formatına çevirdim.

```
[32] # 'ilan_basligi' sütununu sil
     veri.drop(columns=['ilan_basligi'], inplace=True)
```

```
[33] # 'ilan_konumu' sütununu virgülle ayırarak yeni sütunlara atama
     veri[['il', 'ilce']] = veri['ilan_konumu'].str.split(', ', expand=True)
```

```
[34] # 'ilan_konumu' sütununu sil
     veri.drop(columns=['ilan_konumu'], inplace=True)
```

```
[35] veri['ilan_tarihi'] = pd.to_datetime(veri['ilan_tarihi'])
```

Şekil 18 Veri setindeki gereksiz veya dağınık bilgileri temizleme

Belirli sütunlardaki verileri düzenlemek ve sayısal formata dönüştürmek için şu adımları uyguladım:

1. **'Kilometre' sütunundaki değerlerin düzenlenmesi:** 'Kilometre' sütunundaki ' km' ifadesini kaldırdım ve ardından bu değerleri sayısal formata dönüştürdüm. Bu adım, araçların kullanım mesafelerini kilometre cinsinden daha düzenli bir şekilde elde etmek için yapıldı.

2. **'Model_Yili' sütunundaki verilerin düzenlenmesi:** 'Model_Yili' sütunundaki ' model' ifadesini kaldırarak, sadece yıl bilgisini içeren sayısal bir forma dönüştürdüm. Bu adım, araçların üretim yıllarını yıl cinsinden daha düzenli bir şekilde temsil etmek için yapıldı.
3. **'Motor_Gucu' sütunundaki verilerin düzenlenmesi:** 'Motor_Gucu' sütunundaki ' hp' ifadesini kaldırarak, beygir gücünü ifade eden verileri sayısal bir formata dönüştürdüm. Bu adım, araç motor gücünü sayısal bir biçimde temsil etmek için yapıldı.

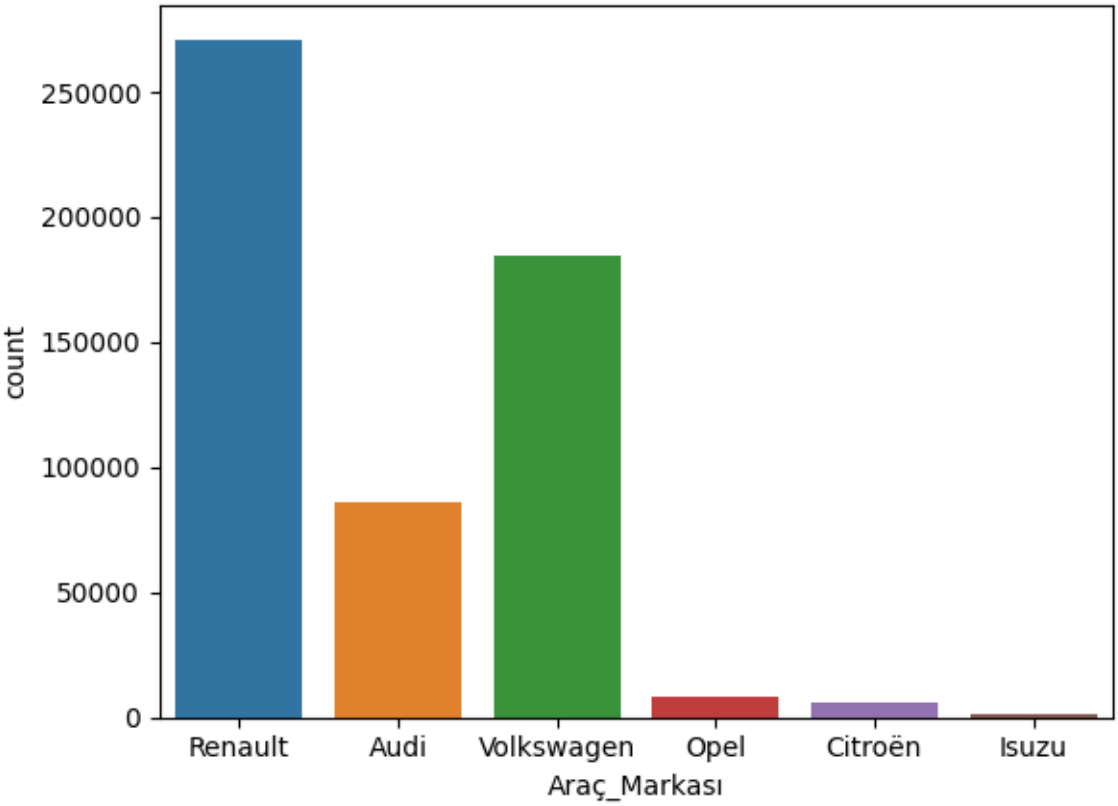
```
[37] # 'Kilometre' sütunundaki ' km' ifadesini kaldırarak boş olmayan değerleri sayısal formata dönüştürme
     veri['Kilometre'] = veri['Kilometre'].str.replace(' km', '')
     veri['Kilometre'] = pd.to_numeric(veri['Kilometre'], errors='coerce').astype('Int64')

[38] # 'Model_Yili' sütununu sayısal formata dönüştürme
     veri['Model_Yili'] = pd.to_numeric(veri['Model_Yili'].str.replace(' model', ''), errors='coerce').astype('Int64')

[39] # 'Model_Gucu' sütununu sayısal formata dönüştürme
     veri['Motor_Gucu'] = pd.to_numeric(veri['Motor_Gucu'].str.replace(' hp', ''), errors='coerce').astype('Int64')
```

Şekil 19 Sayısal formata dönebilecek sütunları sayıala çevirme

Yazdığım cat_summary ve num_summary fonksiyonları ile veri setinizdeki kategorik ve sayısal değişkenlerin özet bilgilerini ve grafiksel gösterimlerini elde ettim.

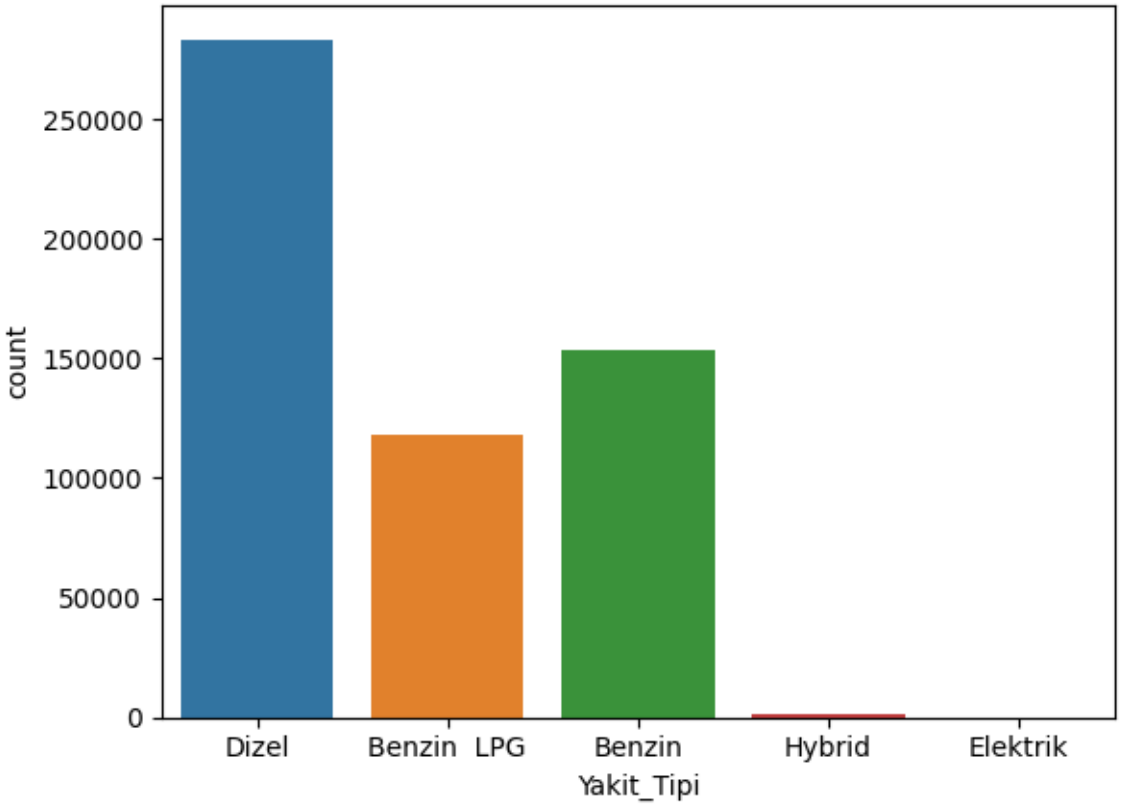
Araç_Markası		Ratio
Renault	271120	48.696377
Volkswagen	184385	33.117739
Audi	86109	15.466201
Opel	7824	1.405283
Citroën	5923	1.063841
Isuzu	1392	0.250020

##



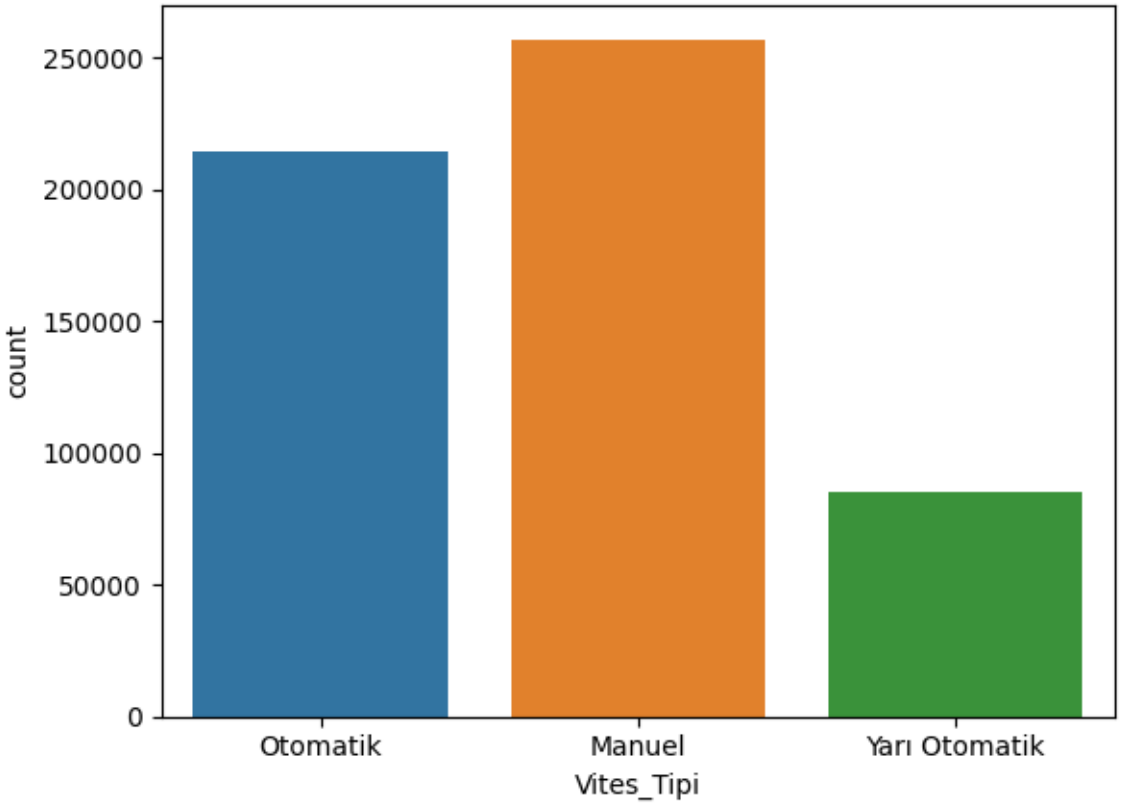
Yakit_Tipi		Ratio
Dizel	283594	50.936856
Benzin	153611	27.590363
Benzin LPG	117898	21.175883
Hybrid	1651	0.296539
Elektrik	1	0.000180

##



Vites_Tipi		Ratio
Manuel	256947	46.150737
Otomatik	214375	38.504300
Yarı Otomatik	85373	15.334006

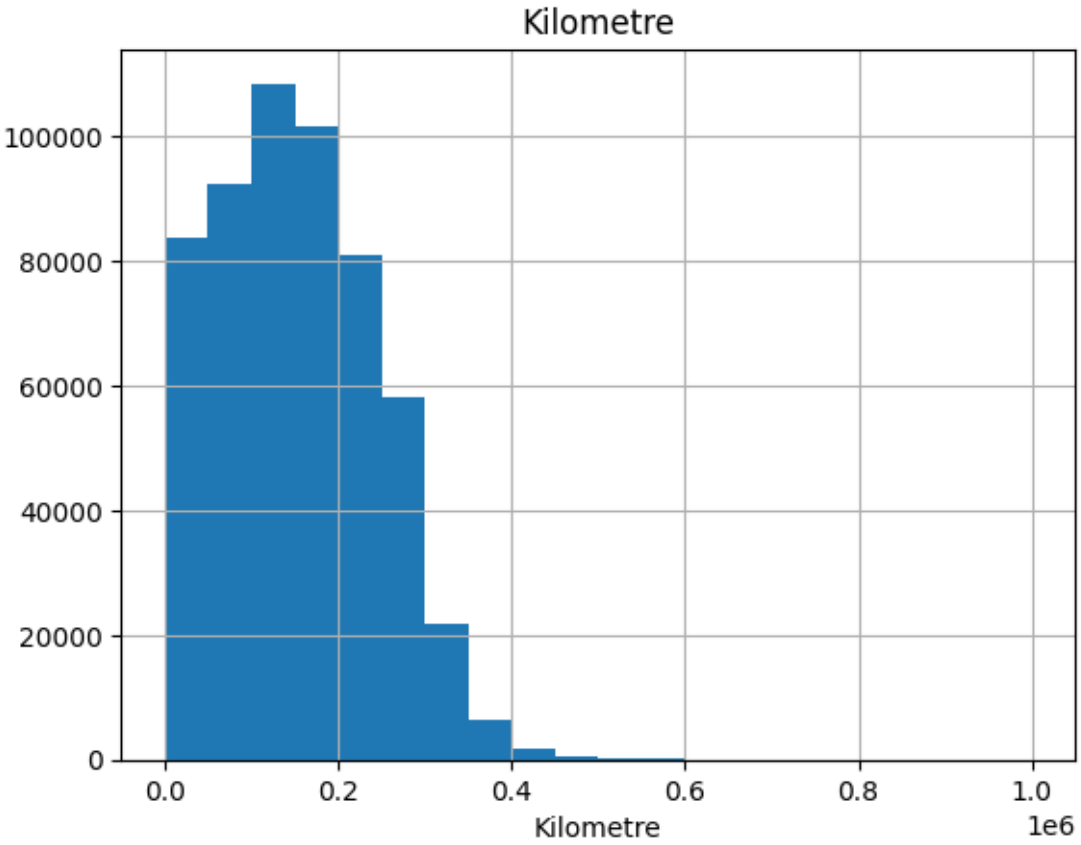
##



Şekil 20 Kategorik değerler için elde edilen çıktılar

count	556755.0
mean	152464.72744
std	91213.688167
min	0.0
5%	15000.0
10%	32000.0
20%	67000.0
30%	95900.0
40%	121000.0
50%	147000.0
60%	173000.0
70%	201000.0
80%	235000.0
90%	275000.0
95%	302165.0
99%	370000.0
max	1000000.0

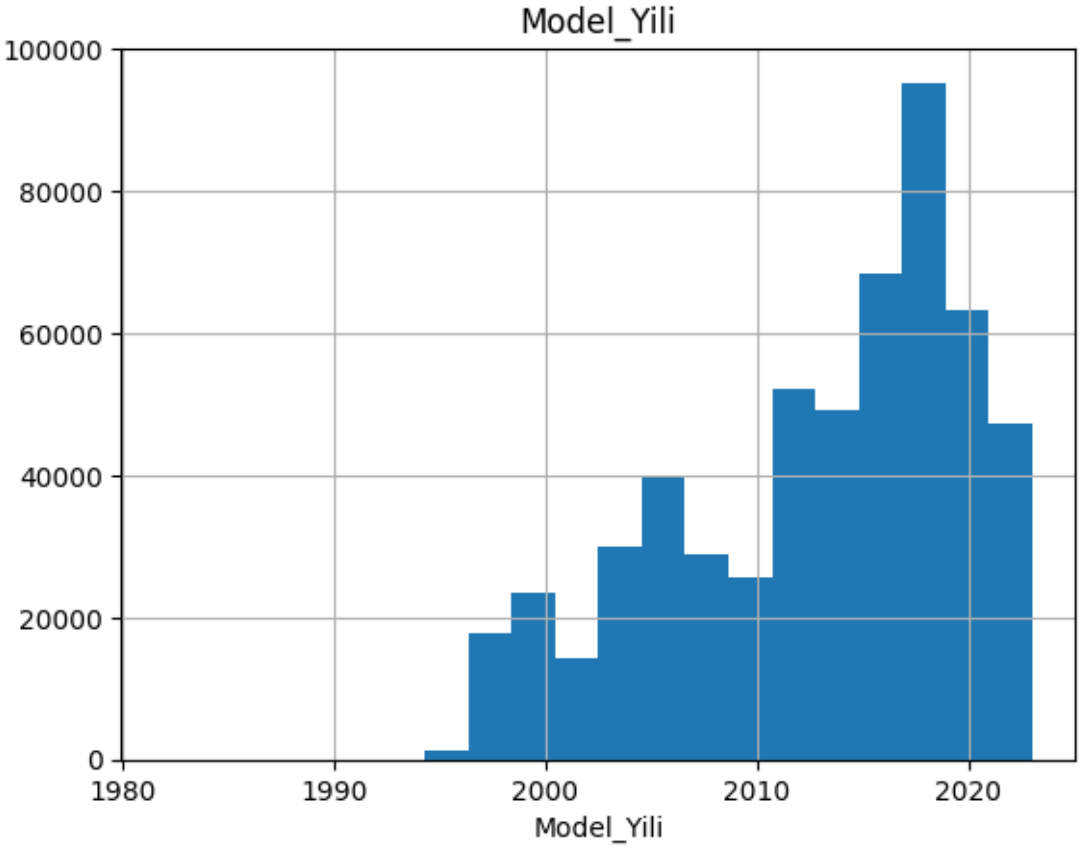
Name: Kilometre, dtype: Float64



count	556755.0
mean	2012.604938
std	6.761917
min	1982.0
5%	1999.0
10%	2002.0
20%	2006.0

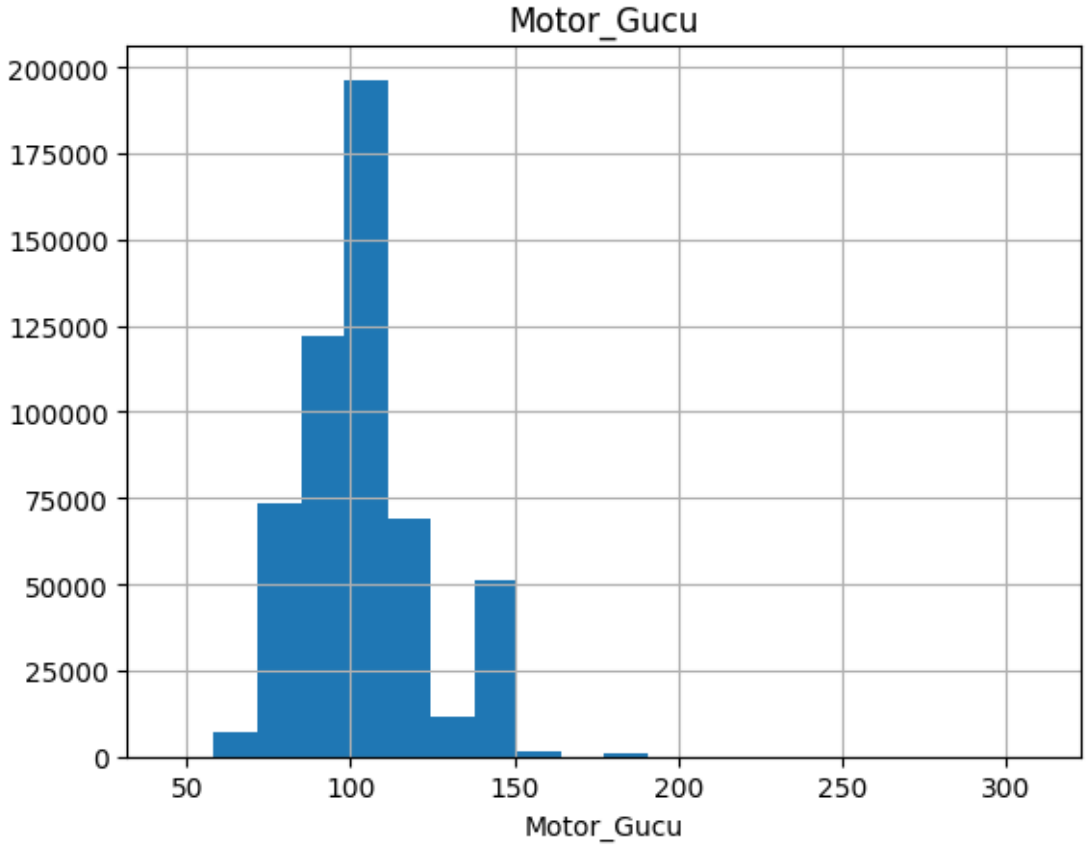
30%	2009.0
40%	2012.0
50%	2014.0
60%	2016.0
70%	2017.0
80%	2018.0
90%	2020.0
95%	2021.0
99%	2023.0
max	2023.0

Name: Model_Yili, dtype: Float64



count	532348.0
mean	103.792788
std	19.334785
min	45.0
5%	75.0
10%	75.0
20%	86.0
30%	90.0
40%	100.0
50%	107.0
60%	110.0
70%	110.0
80%	115.0
90%	140.0
95%	140.0
99%	150.0

max 310.0
Name: Motor_Gucu, dtype: Float64



Şekil 21 Nümerik değerler için elde edilen çıktılar

Her araç modeli için aykırı fiyat değerlerini belirleyerek veri setinden kaldırmak için bir kod bloğu yazdım.

1. Her bir araç modeli için fiyatın ortalama ve standart sapma değerlerini hesaplamak: Her araç modelinin fiyatlarının genel dağılımını anlamak ve her model için fiyatların ortalamasını ve yayılma derecesini belirlemek için bu hesaplamaları yaptım.
2. Her araç modeli için alt ve üst limitleri belirlemek: Ardından, her araç modeli için alt ve üst limitler belirleyerek, bu sınırlar dışında kalan fiyatları aykırı değer olarak işaretledim.
3. Aykırı fiyatları tespit etme ve temizleme: Aykırı değerlere sahip olan araç modelleri ve bunlara ait ilan ID'leri ve fiyatları içeren 'anormal_fiyatlar' listesine ekleyerek aykırı fiyatları tespit ettim. Daha sonra, bu aykırı fiyatları içeren satırları veri setinden kaldırdım. 'reset_index(drop=True)' ile veri setinin indexleri sıfırlandı.

Bu adımların amacı, analizdeki aykırı fiyat değerlerinin veri setine olan etkisini azaltmak ve analitik sürecin güvenilirliğini artırmaktı. Bu şekilde, veri setinin daha tutarlı ve doğru bir şekilde kullanılmasını hedefledim.

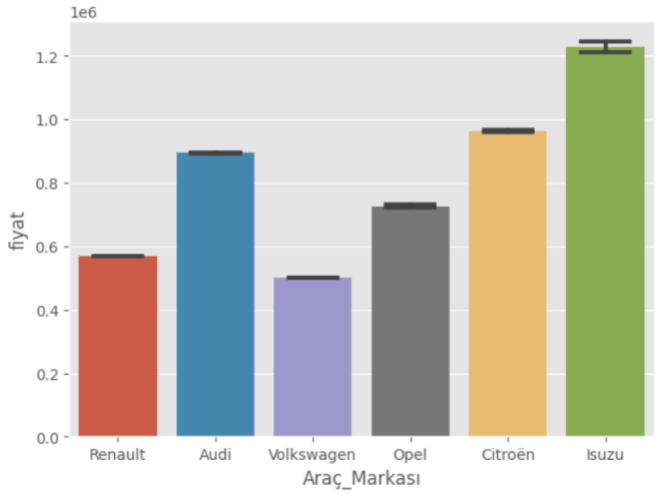
```
[48] #Her model için aykırı fiyat değerlerini hesapla ve aykır değerleri sil
     fiyat_istatistikleri = veri.groupby('Araç_Modeli')['fiyat'].agg(['mean', 'std']).reset_index()

     anormal_fiyatlar = []
     for _, row in fiyat_istatistikleri.iterrows():
         arac_modeli = row['Araç_Modeli']
         ortalama_fiyat = row['mean']
         standart_sapma = row['std']
         alt_limit = ortalama_fiyat - 0.75 * standart_sapma  # Alt limiti belirle
         ust_limit = ortalama_fiyat + 0.80 * standart_sapma  # Üst limiti belirle
         arac_fiyatlar = veri[(veri['Araç_Modeli'] == arac_modeli) & ((veri['fiyat'] < alt_limit) | (veri['fiyat'] > ust_limit))]
         if not arac_fiyatlar.empty:
             anormal_fiyatlar.append((arac_modeli, arac_fiyatlar[['ilan_id', 'fiyat']]))
             veri = veri.drop(arac_fiyatlar.index)  # Aykırı değerleri sil

     veri.reset_index(drop=True, inplace=True)  # Veri setini resetle
```

Şekil 22 Aykırı değerlerin kaldırılması

Araç markalarının fiyatlarının genel dağılımını görselleştirdim. Bu sayede farklı markaların fiyat dağılımlarını karşılaştırarak hangi markaların daha yüksek veya düşük fiyat aralıklarında olduğunu gözlemledim.

```
plt.style.use("ggplot")
fig, ax = plt.subplots(figsize=(7, 5))
sns.barplot(x = veri["Araç_Markası"], y = veri["fiyat"],capsize=0.5)
```

```
<Axes: xlabel='Araç_Markası', ylabel='fiyat'>
```



Şekil 23 Farklı markaların fiyat dağılımlarının karşılaştırılması

Araçların ilan tarihi ve model yılı bilgilerini kullanarak, veri setindeki araçların yaşını hesapladım. 'ilan_tarihi' sütunundan yıl bilgisini alarak 'Model_Yili' sütunundaki model yılını çıkardım. Bu işlem sonucunda, aracın kaç yaşında olduğunu belirlemek için yeni bir 'Arac_Yasi' sütunu oluşturdum. Bu işlemden sonra Model_Yili sütununa gerek kalmadığı için bu sütunu sildim.

```
[53] # Model yılından ilan tarihini çıkararak aracın yaşını hesaplama
     veri['Arac_Yasi'] = veri['ilan_tarihi'].dt.year - veri['Model_Yili']

     veri.head()
```

	ilan_id	fiyat	ilan_tarihi	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Model_Yili	Yakit_Tipi	Motor_Gucu	Vites_Tipi	il	ilce	Arac_Yasi
0	1080850207	830000.0	2023-05-26 09:23:29	1.5 dCi Icon	Renault	Megane	143000	2016	Dizel	110	Otomatik	İstanbul	Ümraniye	7
1	1101850722	905000.0	2023-05-26 09:23:13	1.5 dCi Icon	Renault	Megane	80000	2017	Dizel	110	Otomatik	İzmir	Konak	6
2	1101851921	1380000.0	2023-05-26 09:22:35	A3 Sedan 35 TFSI	Audi	A3	33000	2020	Benzin	150	Otomatik	İstanbul	Sarıyer	3
3	1090594643	530000.0	2023-05-26 09:21:55	1.4 TDI Trendline	Volkswagen	Polo	125000	2015	Dizel	75	Manuel	Kırklareli	Lüleburgaz	8
4	1101862946	685000.0	2023-05-26 09:21:08	1.5 dCi Icon	Renault	Megane	90500	2013	Dizel	110	Otomatik	İzmir	Bergama	10

Şekil 24 Araç yaşının hesaplanıp veriye eklenmesi

```
[54] # 'Model_Yili' sütununu sil
     veri.drop(columns=['Model_Yili'], inplace=True)
```

Şekil 25 Model_Yili sütununun silinmesi

One-hot encoding, kategorik verileri sayısal forma dönüştürmek için kullanılan bir yöntemdir. Bu yöntem, kategorik bir sütundaki her bir farklı kategoriye ait değerleri, o kategori için yeni bir sütun oluşturarak temsil eder. Bu yeni sütunlar, ilgili kategorideki varlığı veya yokluğu ifade etmek üzere 1 veya 0 değerlerini alır. Bu şekilde, makine öğrenimi modelleri gibi algoritmalar kategorik verilerle çalışabilir. Bunun için bu yöntemi Yakıt "Yakit_Tipi" ve "Vites_Tipi" sütunları için uyguladım.

```
[56] # One-Hot Encoding yaparak Yakit_Tipi sütununu dönüştürme
     veri = pd.get_dummies(veri, columns=['Yakit_Tipi'], prefix='Yakit')
     veri.head()
```

	ilan_id	fiyat	ilan_tarihi	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Motor_Gucu	Vites_Tipi	il	ilce	Arac_Yasi	Yakit_Benzin	Yakit_Benzin LPG	Yakit_Dizel
0	1080850207	830000.0	2023-05-26 09:23:29	1.5 dCi Icon	Renault	Megane	143000	110	Otomatik	İstanbul	Ümraniye	7	0	0	1
1	1101850722	905000.0	2023-05-26 09:23:13	1.5 dCi Icon	Renault	Megane	80000	110	Otomatik	İzmir	Konak	6	0	0	1
2	1101851921	1380000.0	2023-05-26 09:22:35	A3 Sedan 35 TFSI	Audi	A3	33000	150	Otomatik	İstanbul	Sarıyer	3	1	0	0
3	1090594643	530000.0	2023-05-26 09:21:55	1.4 TDI Trendline	Volkswagen	Polo	125000	75	Manuel	Kırklareli	Lüleburgaz	8	0	0	1
4	1101862946	685000.0	2023-05-26 09:21:08	1.5 dCi Icon	Renault	Megane	90500	110	Otomatik	İzmir	Bergama	10	0	0	1

Şekil 26 "Yakit_Tipi" sütunu için one-hot encoding işleminin uygulanması

```
[57] # One-Hot Encoding ile Vites_Tipi sütununu dönüştürme
     veri = pd.get_dummies(veri, columns=['Vites_Tipi'], prefix='Vites')
```

Şekil 27"Vites_Tipi" sütunu için one-hot encoding işleminin uygulanması

'Kilometre' sütunundaki araçların kilometre değerini 'Arac_Yasi' sütunundaki araç yaşına bölerek, her bir aracın kilometre başına düşen yaşını hesapladım. Bu değeri, bir aracın ne kadar kullanıldığını gösteren bir ölçü olarak kullandım.

```
[59] # Yıpranma değeri hesaplama
     veri['Yipranma_Degeri'] = veri['Kilometre'] /  veri['Arac_Yasi']
     veri.head()
```

lometre	Motor_Gucu	il	ilce	Arac_Yasi	Yakit_Benzin	Yakit_Benzin LPG	Yakit_Dizel	Yakit_Elektrik	Yakit_Hybrid	Vites_Manuel	Vites_Otomatik	Vites_Yarı Otomatik	Yipranma_Degeri
143000	110	İstanbul	Ümraniye	7	0	0	1	0	0	0	1	0	20428.571429
80000	110	İzmir	Konak	6	0	0	1	0	0	0	1	0	13333.333333
33000	150	İstanbul	Sarıyer	3	1	0	0	0	0	0	1	0	11000.0
125000	75	Kırklareli	Lüleburgaz	8	0	0	1	0	0	1	0	0	15625.0
90500	110	İzmir	Bergama	10	0	0	1	0	0	0	1	0	9050.0

Şekil 28 Yıpranma değerinin hesaplanması

Veri setini makine öğrenimi modeline uygun hale getirmek için şu ön işlem adımlarını gerçekleştirdim:

1. **Veri Dönüşümü ve Hedef Belirleme:** İlk olarak, veriyi bir dönüşüm işlemine tabi tutarak 'ilan_id' sütununu bıraktım ve 'fiyat' sütununu hedef değişken olarak belirledim. Bu adımda, modelin öğrenmesini istediğimiz çıktı değişkenini belirledim.

2. **Bağımsız ve Bağımlı Değişkenlerin Ayrılması:** Daha sonra, veri setini bağımsız değişkenler (X) ve bağımlı değişken (y) olarak ayırdım. Bu adımda, modelin eğitimde kullanacağı girdi verilerini (bağımsız değişkenler) ve hedef çıktıyı (bağımlı değişken) belirledim.

3. **Eğitim ve Test Setlerine Bölme:** 'train_test_split' fonksiyonunu kullanarak veri setini eğitim ve test alt kümelerine böldüm. Bu adımda, veri setini modelin eğitiminde kullanılacak eğitim seti ve daha sonra performansının test edileceği test seti olarak böldüm.

```
[62] veri_fit = veri.apply(preprocessing.LabelEncoder().fit_transform)
```

```
veri_fit
```

	ilan_id	fiyat	ilan_tarihi	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Motor_Gucu	il	ilce	Arac_Yasi	Yakit_Benzin	Yakit_Benzin LPG	Yakit_Dizel
0	144949	7225	280221	89	4	5	18182	21	1602	5406	7	0	0	1
1	245781	7888	280220	89	4	5	10739	21	1608	2892	6	0	0	1
2	245791	10201	280219	242	0	0	4644	31	1602	4011	3	1	0	0
3	187108	4475	280218	67	5	6	16463	6	926	3260	8	0	0	1
4	245842	5945	280217	89	4	5	12271	21	1608	873	10	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
521008	245854	2664	280225	34	4	5	24789	13	1284	4508	16	0	1	0
521009	245331	6269	280224	22	5	6	7716	10	1602	5224	7	1	0	0
521010	245863	3836	280223	67	5	6	22957	6	271	4330	8	0	0	1
521011	245860	8635	280222	75	4	5	3135	22	1048	1736	3	0	0	1
521012	245853	4772	280222	247	0	0	21212	17	43	3383	16	0	1	0

521013 rows × 20 columns

Şekil 29 Veriyi model için hazırlama- Label encoding işlemi

```
[64] veri_fit.drop(columns=['ilan_id'], inplace=True)
```

```
[65] X = veri_fit.drop('fiyat', axis=1)
     y = veri_fit["fiyat"]

     X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=0.20)
```

```
[66] veri_fit.head()
```

	fiyat	ilan_tarihi	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Motor_Gucu	il	ilce	Arac_Yasi	Yakit_Benzin	Yakit_Benzin LPG	Yakit_Dizel	Yakit_Elektrik	Yak
0	7225	280221	89	4	5	18182	21	1602	5406	7	0	0	1	0	
1	7888	280220	89	4	5	10739	21	1608	2892	6	0	0	1	0	
2	10201	280219	242	0	0	4644	31	1602	4011	3	1	0	0	0	
3	4475	280218	67	5	6	16463	6	926	3260	8	0	0	1	0	
4	5945	280217	89	4	5	12271	21	1608	873	10	0	0	1	0	

Şekil 30 Veriyi model için hazırlama - veri setinin bağımlı ve bağımsız değişkenlerine ayrılması

3. Modellerin Oluşturulması

3.1. Lineer Regresyon

Lineer Regresyon, denetimli öğrenme modellerinden biridir. Bu model, bağımsız değişkenler ile bağımlı değişken arasındaki ilişkiyi tanımlamak için kullanılır. Örneğin, bir veya birden fazla girdi değişkeni (X) ile bir çıktı değişkeni (y) arasındaki istatistiksel ilişkiyi belirlemek amacıyla kullanılır. Bu model, veri setindeki bağımsız değişkenler ile bu değişkenlere karşılık gelen hedef çıktı arasındaki en uygun doğrusal ilişkiyi ifade eden bir denklem oluşturur. Oluşturulan bu denklem doğrultusunda, girdi değişkenlerinden çıktı değişkenini tahmin

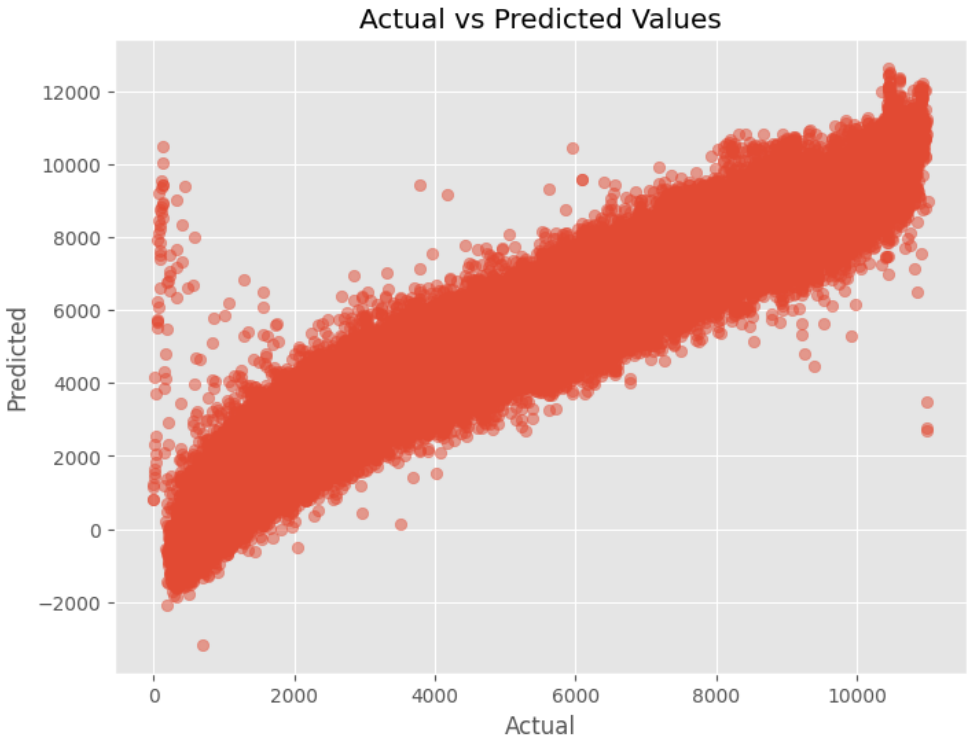
etmeye çalışır. İdeali, bu denklem ile çizilen lineer doğru, veri noktalarının etrafında en iyi uyan doğru olmalıdır.

Lineer regresyon, girdi ve çıktı arasındaki ilişkiyi anlamak, öngörmek ve açıklamak için kullanılır. Model, girdi değişkenlerin çıktı değişkeni üzerindeki etkisini ve bu etkilerin büyüklüğünü analiz etmeye yardımcı olur.

```python
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, r2_score

linear_reg = LinearRegression()

linear_reg.fit(X_train, y_train)

score_lineer = linear_reg.score(X_test, y_test)
print(f"Model Score: {score_lineer}")

y_pred = linear_reg.predict(X_test)

mse_lineer = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse_lineer}")

r_squared_lineer = r2_score(y_test, y_pred)
print(f"R-squared (R^2): {r_squared_lineer}")

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted Values")
plt.show()
```

Şekil 31 Lineer regresyon kodu

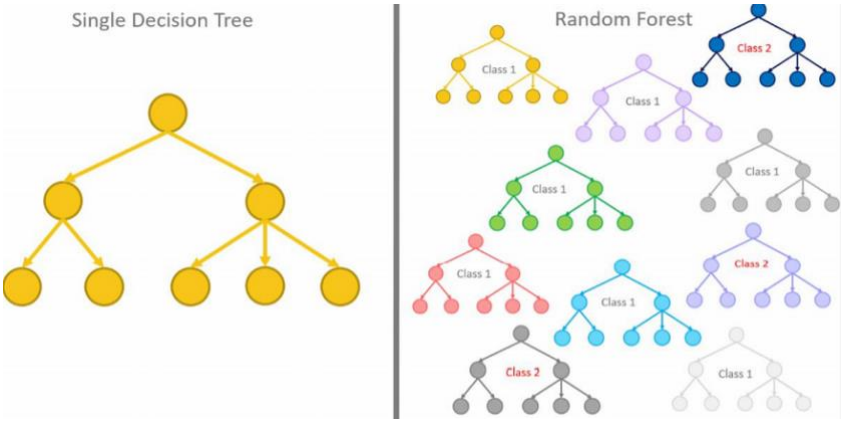
Model Score: 0.910373775582667
Mean Squared Error (MSE): 625864.1324235535
R-squared (R^2): 0.910373775582667



Şekil 32 Lineer regresyon için elde edilen çıktı

3.2. Random Forest Regresyonu

Random Forest algoritması, denetimli sınıflandırma ve regresyon problemleri için kullanılan bir algoritmadır. Birden fazla karar ağacı oluşturarak sınıflandırma veya regresyon işleminde doğruluğu artırmayı hedefler. Bu algoritma, birbiriyle bağımsız çalışan birçok karar ağacının bir araya gelerek en yüksek performansı veren değeri seçmesini sağlar.



Şekil 33 Random forest algoritmik yapısı

Görselde gösterildiği gibi, Random Forest algoritması birden fazla karar ağacını içerir. Bu ağaçlar, veri setinden rastgele örneklem alarak ve rastgele özellikler seçerek eğitilir. Karar ağaçları modelin bir parçası olarak çalışır ve her ağaç veriye farklı bakış açılarından değerlendirme yapar. Ardından, tüm bu ağaçların tahminlerini birleştirerek en iyi sonucu veren çıktıyı belirler.

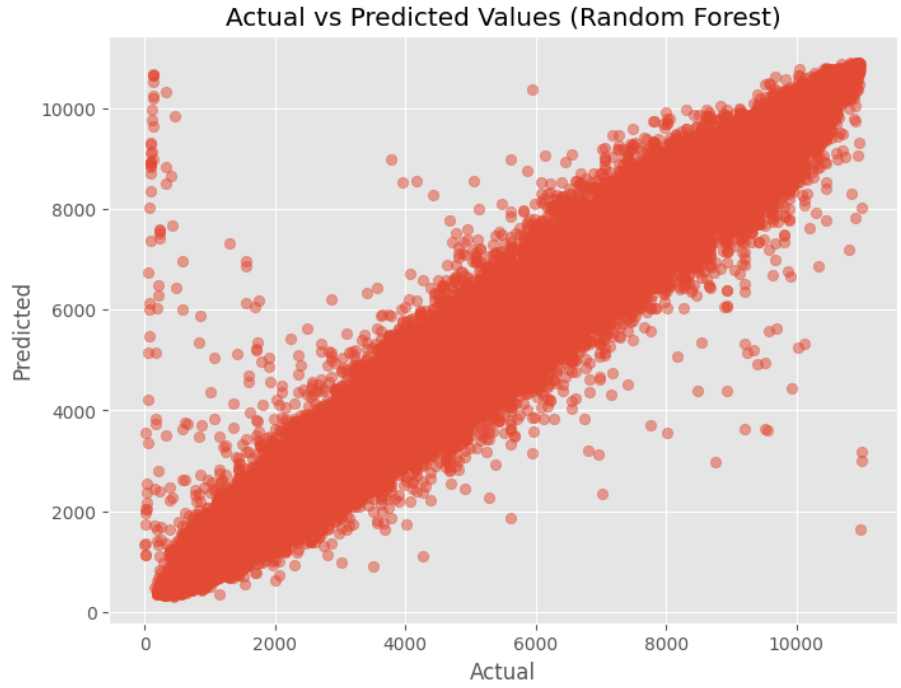
Random Forest algoritmasının karar ağaçları algoritmasından farkı, kök düğümü bulma ve düğümleri bölme işlemlerinin rastgele seçilmesidir. Bu, her bir ağacın farklı özelliklere odaklanmasını sağlar ve genellikle tek bir karar ağacına göre daha iyi genelleme yapabilir.

Ağaç sayısı arttıkça, modelin doğruluğu genellikle artar. Bu algoritma, veri setlerindeki karmaşıklığı ele almak ve daha güçlü tahminler yapabilmek için etkili bir yöntemdir.

```python
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

rf_reg = RandomForestRegressor()

rf_reg.fit(X_train, y_train)

score_rfr = rf_reg.score(X_test, y_test)
print(f"Model Score: {score_rfr}")


y_pred = rf_reg.predict(X_test)

mse_rfr = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse_rfr}")

r_squared_rfr = r2_score(y_test, y_pred)
print(f"R-squared (R^2): {r_squared_rfr}")

# Plotting actual vs predicted values
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted Values (Random Forest)")
plt.show()
```

Şekil 34 Random Forest Regresyonu kodu

Model Score: 0.9655251733180504
Mean Squared Error (MSE): 240739.3330693285
R-squared (R^2): 0.9655251733180504



Şekil 35 Random Forest regresyonu için çıktılar

3.3. XGB Regressor

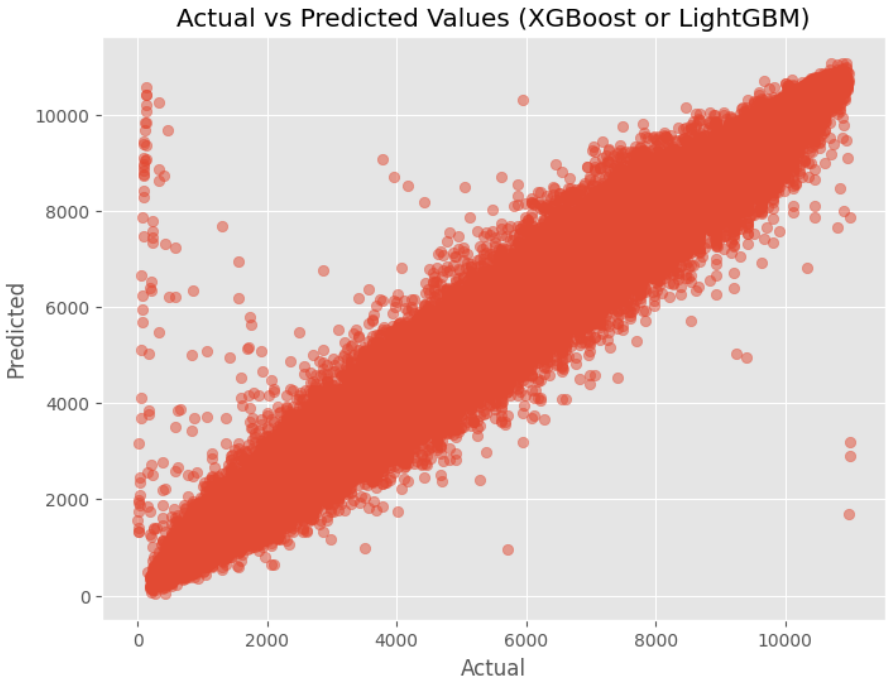
XGBoost, Gradient Boosting algoritmasının optimize edilmiş ve yüksek performanslı bir versiyonudur. 2016'da Tianqi Chen ve Carlos Guestrin tarafından sunulan "XGBoost: A Scalable Tree Boosting System" makalesiyle literatüre girmiştir. Bu algoritmanın dikkate değer özellikleri arasında yüksek tahmin gücü, aşırı öğrenmeyi engelleyebilme yetisi, eksik verileri etkili bir şekilde yönetebilme ve hızlı çalışabilme özelliği bulunmaktadır. Tianqi'ye göre, XGBoost diğer popüler algoritmalardan 10 kat daha hızlı çalışmaktadır.

Yazılım ve donanım optimizasyon tekniklerinin uygulanmasıyla, daha az kaynak kullanarak üst düzey sonuçlar elde etmeyi amaçlar. Karar ağacı tabanlı algoritmalar arasında en iyi performansa sahip olduğu bilinir. XGBoost, bu özellikleriyle öne çıkar ve genellikle makine öğrenimi projelerinde tercih edilen bir modeldir. Bu algoritmanın, özellikle büyük ölçekli veri setleri üzerinde etkili ve hızlı çalıştığı bilinmektedir.

```python
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
import matplotlib.pyplot as plt

xgb_reg = XGBRegressor()

xgb_reg.fit(X_train, y_train)

score_xgb = xgb_reg.score(X_test, y_test)
print(f"Model Score: {score_xgb}")

y_pred = xgb_reg.predict(X_test)

mse_xgb = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse_xgb}")

r_squared_xgb = r2_score(y_test, y_pred)
print(f"R-squared (R^2): {r_squared_xgb}")


plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted Values (XGBoost or LightGBM)")
plt.show()
```

Şekil 36 XGB Regressor kodu

Model Score: 0.9663446717740085
Mean Squared Error (MSE): 235016.73688171722
R-squared (R^2): 0.9663446717740085



Şekil 37 XGB Regressor için elde edilen sonuçlar

3.4. Gradient Boosting Regressor

Gradient Boosting Regressor, bir makine öğrenimi algoritmasıdır ve regresyon problemleri için kullanılır. Temelde Gradient Boosting yöntemini kullanan bir regresyon modelidir. Bu algoritma, ensemble (bir araya getirme) tekniğini kullanarak birden çok zayıf tahmin modelini birleştirir ve böylece daha güçlü ve genelleştirilmiş bir tahmin modeli oluşturur. Gradient Boosting Regressor, ardışık olarak oluşturulan zayıf tahmin modellerini bir araya getirerek, bir önceki modelin yapamadığı hataları düzeltmeye çalışır.

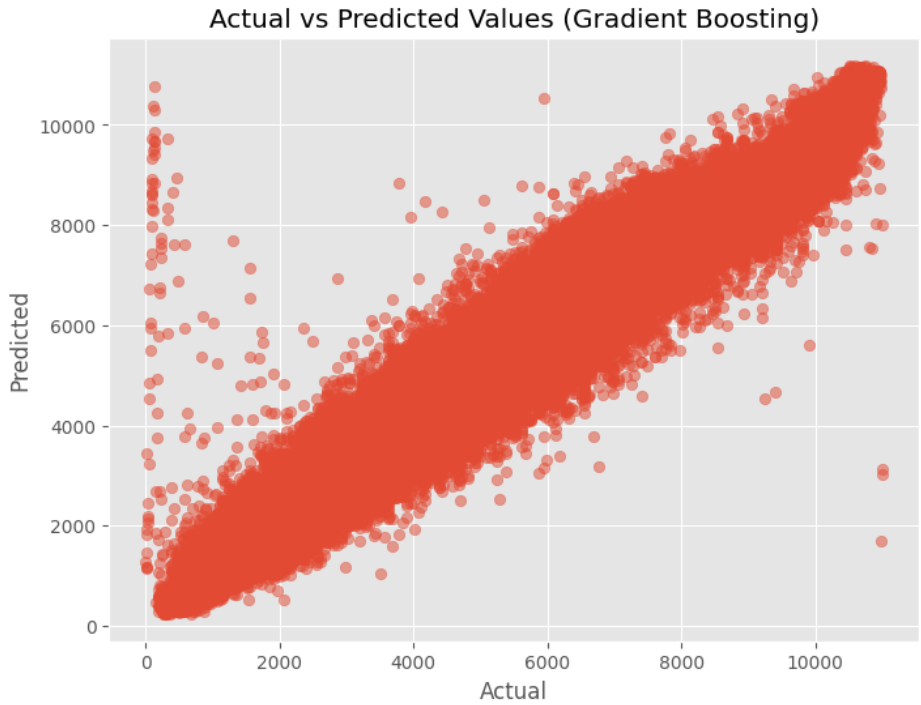
Çalışma prensibi, ardışık ağaçlar oluşturarak yapılır. Her ağaç, önceki ağacın hatalarını düzeltmeye odaklanır. Yani, her bir ağaç, önceki ağaçların tahmin hatalarına odaklanarak veri setinin arta kalan özelliklerini daha iyi öğrenmeye çalışır. Gradient Boosting Regressor, tahmin edilen değerler ile gerçek değerler arasındaki farkı minimize etmeye çalışır. Bu farkı minimize etmek için de gradyan iniş algoritmasını kullanarak modeli eğitir. Bu sayede, veri setindeki ilişkileri öğrenir ve özellikle regresyon problemlerinde yüksek tahmin doğruluğu sağlamayı hedefler.

Bu algoritma, özellikle karmaşık ilişkileri ve gürültülü veri setlerini işlemek için etkili bir seçenektir ve regresyon problemlerinde genellikle tercih edilen bir modeldir.

```python
from sklearn.ensemble import GradientBoostingRegressor
import matplotlib.pyplot as plt

gb_reg = GradientBoostingRegressor()

gb_reg.fit(X_train, y_train)

y_pred_gb = gb_reg.predict(X_test)

score_gbr = gb_reg.score(X_test, y_test)
print(f"Gradient Boosting Score: {score_gbr}")

mse_gbr = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error (MSE): {mse_gbr}")

r_squared_gbr = r2_score(y_test, y_pred)
print(f"R-squared (R^2): {r_squared_gbr}")


plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred_gb, alpha=0.5)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Actual vs Predicted Values (Gradient Boosting)")
plt.show()
```

Şekil 38 Gradient Boosting Regressor kodu

Gradient Boosting Score: 0.9566170890366925
Mean Squared Error (MSE): 235016.73688171722
R-squared (R^2): 0.9663446717740085



Şekil 39 Gradient Boosting Regressor için sonuçlar

4. Öneri Sistemi

Araç fiyatlarını tahmin etmek için 4 farklı regresyon modeli oluşturup sonuçları karşılaştırdıktan sonra en iyi sonuç aldığım modellerden biri olan Gradient Boosting Regresyonunu kullanarak araç fiyat öneri sistemi oluşturdum. Bu sistemde daha önceden eğittiğim modellerden farklı olarak daha az değişken ile çalıştım. Kullanıcı bu verileri girdiğinde model tarafından belirli bir fiyat tahmini yapılmasını sağladım.

1. **Veri Setinin Hazırlanması:** İlk aşamada, farklı regresyon modellerini eğitmek için gerekli veri seti hazırlandı. Kategorik veriler, **LabelEncoder** kullanılarak numerik değerlere dönüştürüldü. Eksik değerler, veri setinden çıkarılarak temizlenmiş bir veri kümesi elde edildi.

```
veri.head()
```

	fiyat	Araç_Modeli	Araç_Markası	Araç_Seri	Kilometre	Yakit_Tipi	Motor_Gucu	Vites_Tipi	il	ilce	Araç_Yasi
0	830000.0	1.5 dCi Icon	Renault	Megane	143000	Dizel	110	Otomatik	İstanbul	Ümraniye	7
1	905000.0	1.5 dCi Icon	Renault	Megane	80000	Dizel	110	Otomatik	İzmir	Konak	6
2	695000.0	A3 Sportback 1.6	Audi	A3	177000	Benzin LPG	102	Manuel	Adana	Seyhan	12
3	1380000.0	A3 Sedan 35 TFSI	Audi	A3	33000	Benzin	150	Otomatik	İstanbul	Sarıyer	3
4	530000.0	1.4 TDI Trendline	Volkswagen	Polo	125000	Dizel	75	Manuel	Kırklareli	Lüleburgaz	8

Şekil 40 Öneri sistemi için kullanılan veri seti

2. **Regresyon Modellerinin Oluşturulması:** Gradient Boosting Regresyon modeli oluşturuldu ve veri seti ile eğitim yapıldı.

```
import pandas as pd
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import LabelEncoder

# Label Encoding işlemi
label_encoders = {}
categorical_cols = ['Araç_Modeli', 'Araç_Markası', 'Araç_Seri', 'Yakit_Tipi', 'Vites_Tipi', 'il', 'ilce']
for col in categorical_cols:
    label_encoders[col] = LabelEncoder()
    veri[col] = label_encoders[col].fit_transform(veri[col])

veri = veri.dropna()

# Özellikler ve hedef değişkeni seçme
X = veri.drop('fiyat', axis=1)
y = veri['fiyat']

# Modeli oluşturma ve eğitme
model = GradientBoostingRegressor()
model.fit(X, y)
```

```
▾ GradientBoostingRegressor
GradientBoostingRegressor()
```

Şekil 41 Gradient Boosting Regresyon modekli kodu

3. **Araç Fiyat Öneri Sistemi:** Seçilen Gradient Boosting Regresyon modeli, daha az değişkenle eğitilmiş olarak kullanıcıdan alınan girişlere dayalı olarak araç fiyatlarını tahmin etmek için kullanıldı. Bu sistem, kullanıcının belirttiği verilere göre önerilen bir fiyatı geri döndürdü.

```
[31] # Kullanıcıdan giriş alarak tahmin yapma
     print("Lütfen giriş yapınız:")
     input_data = {}
     for col in X.columns:
         val = input(f"{col}: ")
         input_data[col] = [val]

     input_df = pd.DataFrame(input_data)

     # Label Encoding işlemi
     for col in categorical_cols:
         input_df[col] = label_encoders[col].transform(input_df[col])

     # Tahmin yapma
     predicted_price = model.predict(input_df)
     print(f"Tahmin Edilen Fiyat: {predicted_price[0]}")
```

```
Lütfen giriş yapınız:
Araç_Modeli: 1.5 dCi Icon
Araç_Markası: Renault
Araç_Seri: Megane
Kilometre: 143000
Yakit_Tipi: Dizel
Motor_Gucu: 110
Vites_Tipi: Otomatik
il: İstanbul
ilce: Ümraniye
Araç_Yasi: 7
Tahmin Edilen Fiyat: 974628.2501793152
```

Şekil 42 Araç fiyat öneri sistemi

5. Sonuçlar

Dört farklı regresyon modelini uyguladım ve elde ettiğim sonuçlar şöyle:

Linear Regression: MSE değeri 625864.13 ve R-squared değeri 0.910374.

Random Forest: MSE değeri 240739.33 ve R-squared değeri 0.965525.

XGBRegressor: MSE değeri 235016.73 ve R-squared değeri 0.966345.

Gradient Boosting: MSE değeri 235016.73 ve R-squared değeri 0.966345.

Sonuçlara göre, Random Forest, XGBRegressor ve Gradient Boosting modelleri benzer MSE değerlerine sahip ve veriyi yüksek bir doğrulukla açıklıyorlar. Özellikle XGBRegressor ve Gradient Boosting modelleri, diğerlerine kıyasla daha düşük bir MSE ve daha yüksek bir R-squared değeri elde ederek veriyi daha iyi açıklıyor gibi görünüyor. Bu modeller, veriyi daha iyi tahmin edebilir ve daha güçlü bir performans sergileyebilir. Linear Regression modeli ise diğer modellere kıyasla daha yüksek bir hata ve daha düşük bir doğruluk oranıyla çalışmaktadır, bu da veriyi diğer modellere göre daha az iyi açıkladığını göstermektedir.

```
results = {
    'Model': ['Linear Regression', 'Random Forest', 'XGBRegressor', 'Gradient Boosting'],
    'MSE': [mse_lineer, mse_rfr, mse_xgb, mse_gbr],
    'R-squared': [r_squared_lineer, r_squared_rfr, r_squared_xgb, r_squared_gbr]
}

results_df = pd.DataFrame(results)
print(results_df)
```

```
              Model            MSE  R-squared
0  Linear Regression  625864.132424   0.910374
1      Random Forest  240739.333069   0.965525
2       XGBRegressor  235016.736882   0.966345
3  Gradient Boosting  235016.736882   0.966345
```

Şekil 43 Sonuçlar

Kaynakça

1. "Lineer Regresyon" - Machine Learning Türkiye - Medium: https://medium.com/machine-learning-t%C3%BCrkiye/lineer-regresyon-c7f9fb611605
2. "Makine Öğrenmesinde Random Forest Algoritması" - Ece Akdağlı - Medium: https://ece-akdagli.medium.com/makine-%C3%B6%C4%9Frenmesinde-random-forest-algoritmas%C4%B1-a79b044bbb31
3. "XGBoost Nasıl Çalışır?" - Veri Bilimi Okulu: https://www.veribilimiokulu.com/xgboost-nasil-calisir/
4. “Gradient Boosting Regresyon Örneği" - scikit-learn Örnekler Dökümantasyonu: https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html
5. Asilkan, Özcan. (2008). "Veri Madenciliği Kullanılarak İkinci El Otomobil Pazarında Fiyat Tahmini". Akdeniz Üniversitesi Açık Erişim Arşivi. Bağlantı: http://acikerisim.akdeniz.edu.tr/xmlui/handle/123456789/5004