T.C
DOKUZ EYLÜL ÜNİVERSİTESİ
FEN FAKÜLTESİ
BİLGİSAYAR BİLİMLERİ BÖLÜMÜ

BİTİRME PROJESİ RAPORU

Eyüp Salih ÖZDEMİR

Barış TOPAL

Şiyar UTKAN

Danışman: Resmiye NASİBOĞLU

Mayıs, 2023

İZMİR

ÖZET

Bu bitirme projesi, ikinci el ürünlerin özellikle ikinci el araçların alım-satım işlemlerinde fiyat belirleyici unsurların tespitinde ortaya çıkan zorluklara odaklanmaktadır. Proje, kullanıcı tarafından seçilen ürün belirleyici kriterler ve ilgili ürün modeli için ilan özelliklerinin satış sitelerinden toplanmasını içermektedir. Elde edilen veriler, makine öğrenimi algoritmaları için veri beslemesi olarak kullanılacak ve makine öğrenimi modeli çalıştırıldıktan sonra kullanıcıya geri döndürülecek olan bir ortalama fiyat belirlenecektir. Projenin amacı, kullanıcıların aradıkları ürünün piyasa değerine dayalı olarak fiyat belirlemelerine yardımcı olacak bir mobil uygulama tasarlamaktır.

Anahtar kelimeler: Makine öğrenimi, Veri madenciliği, Android uygulama geliştirme, Kotlin, Coroutine, Asenkron programlama, Python, Beautiful Soup, REST API, FAST API, RetroFit.

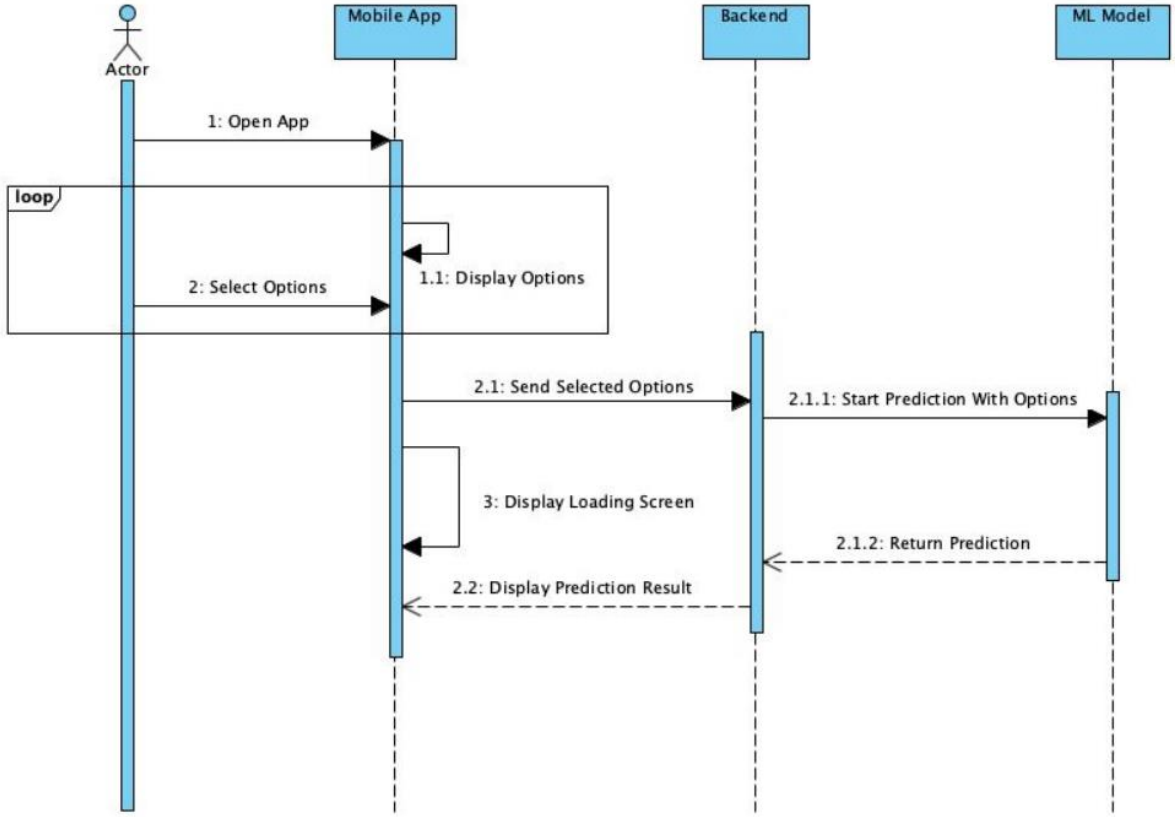
ABSTRACT

This graduation project focuses on the difficulties encountered in determining the price determining factors in the buying and selling processes of second-hand products, especially second-hand vehicles. The project involves collecting advertisement features from sales websites based on user-selected product determining criteria and the corresponding product model. The obtained data will be used as input for machine learning algorithms, and an average price will be determined and returned to the user after running the machine learning model. The aim of the project is to design a mobile application that assists users in determining the price of the desired product based on its market value.

Keywords: Machine learning, Web scraping, Android, Kotlin, Coroutine, Asynchronous programming, Python, Beautiful Soup, REST API, FAST API, RetroFit.

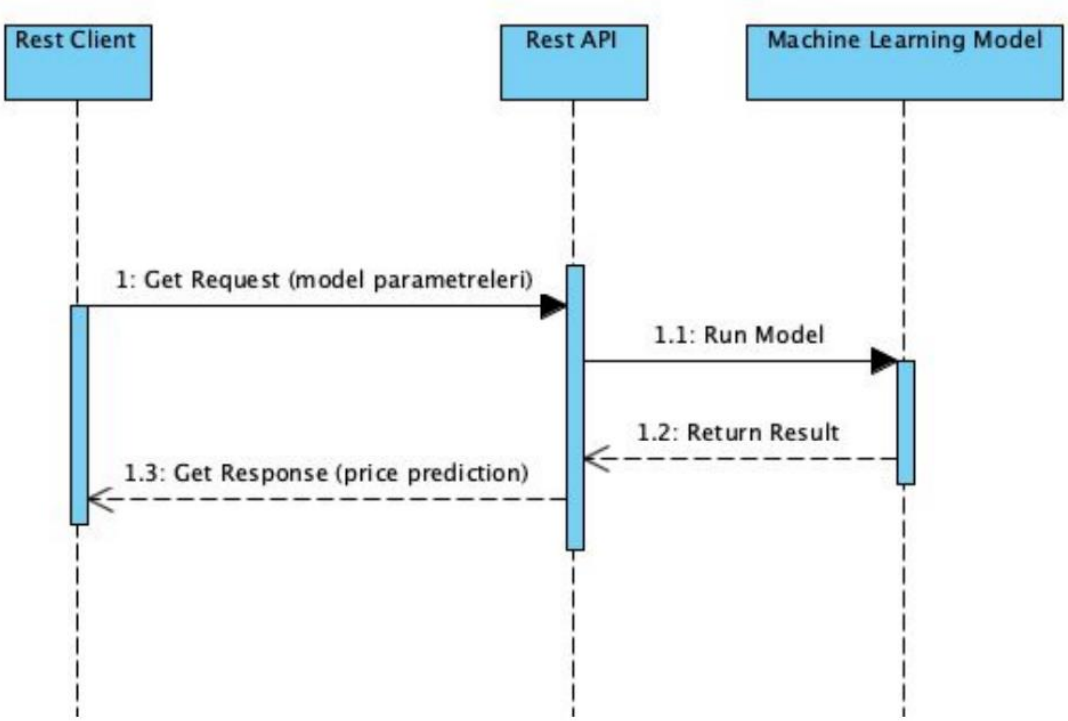
ÇİZELGELER

UML Diyagramı:



Bu diyagram, programın genel akış sürecini göstermektedir.

UML Diyagramı:



Bu diyagram, uygulamanın API ile etkileşim sürecini göstermektedir.

1. PROJEYE GENEL BAKIŞ

1.1. Projenin Konusu

Günümüzde, ikinci el ürünlerin alım-satımı hızla yaygınlaşmaktadır ve bu alanda özellikle ikinci el araçlar önemli bir yer tutmaktadır. Ancak, ikinci el ürünlerin fiyatlandırılması sürecinde, doğru ve gerçekçi fiyatları belirlemek önemli bir zorluk oluşturmaktadır. Piyasada, alıcılar ve satıcılar arasında anlaşmazlıklar, gerçekçi olmayan yüksek fiyatlar ve pahalılık sorunları ortaya çıkmaktadır. Bu durum, alım-satım işlemlerinin verimliliğini ve doğruluğunu olumsuz etkilemektedir. Bu bitirme projesi, ikinci el ürünlerin fiyat belirleme sürecindeki zorlukları ele almaktadır. Temel amacı, alıcı veya satıcı kullanıcıların aradıkları ürün için doğru fiyatları belirleyebilmelerine yardımcı olacak bir mobil uygulama geliştirmektir.

1.2. Giriş

Bu uygulama, piyasa taraması yaparak ilgili ürünün benzerlerinin fiyat ortalamalarını kullanıcıya sunacak ve böylece kullanıcılar daha gerçekçi fiyatlarla alım-satım işlemlerini gerçekleştirebileceklerdir. Projenin ana hedefi, kullanıcının seçtiği ilan özelliklerine göre önceden veri beslemesiyle güçlendirilmiş makine öğrenimi modelini çalıştırmaktır. Mobil uygulama, kullanıcıların güvenilir ve veriye dayalı fiyat bilgilerine erişimini sağlayarak, pahalılık sorununu azaltmayı ve alıcılar ile satıcılar arasında daha sağlıklı bir denge oluşturmayı hedeflemektedir. Projenin tamamlanmasıyla, ikinci el alım-satım sürecinde yaşanan fiyat belirleme zorluklarının üstesinden gelinmesi ve daha doğru, adil ve gerçekçi fiyatlandırmanın sağlanması beklenmektedir.

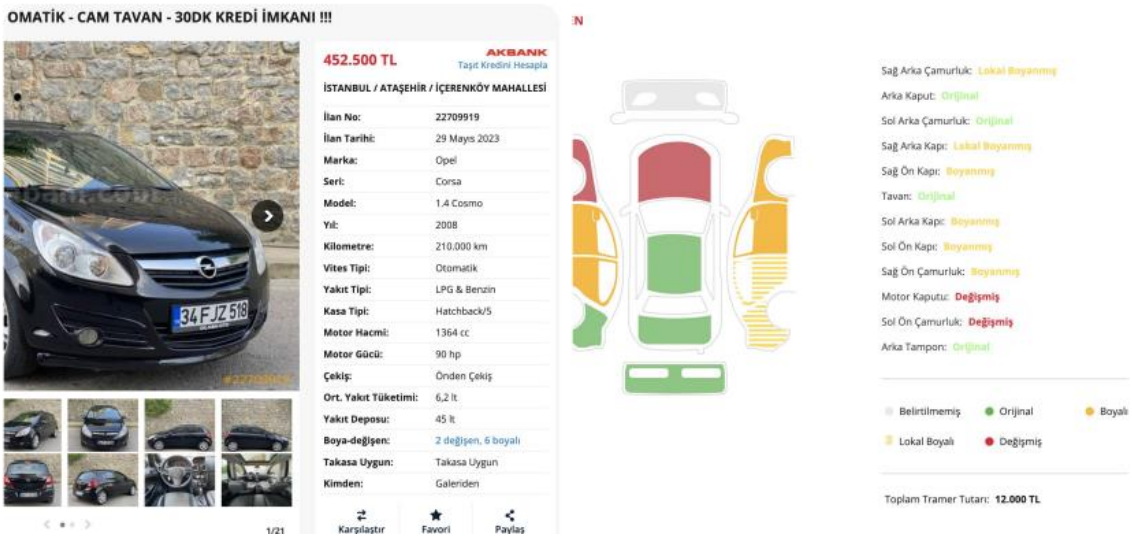
1.3. Yöntem ve Algoritmalar

1.3.1. Genel Yapı

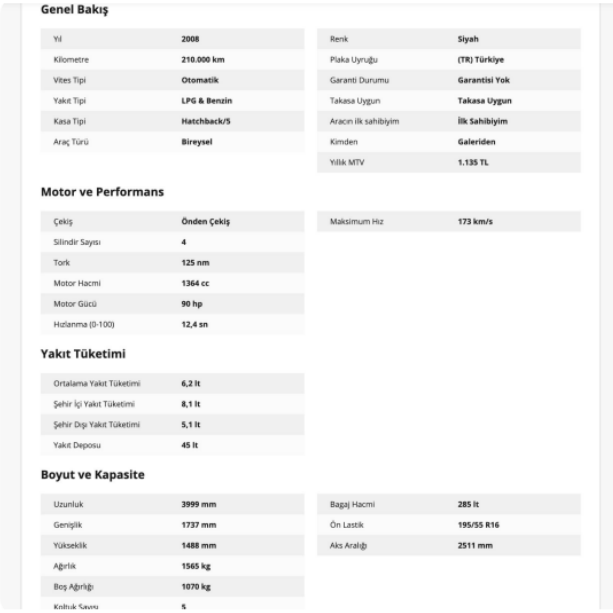
Proje temel düzeyde ele alınacak olursa, uygulamanın bütünü sunucu tarafı (back-end) ve kullanıcı tarafı (front-end) olacak şekilde ikiye ayrılabilir. Sunucu tarafında; veri madenciliği, API ve makine öğrenimi için kullanılacak tüm yapılarda Python dil desteğinden faydalanılmıştır. Kullanıcı tarafında; mobil uygulamanın geliştirilmesi için Kotlin dil desteğinden ve Android işletim sisteminden faydalanılmıştır.

1.3.2. İlgili İlan Verilerinin Toplanması

Bu adımdaki uygulamalar, makine öğrenimi modeline veri beslemesi için kullanılacak olan ilan verilerinin tespitini ve toplanışını içermektedir. Öncelikle veri setinde toplanması hedeflenen her marka için o markaya ait seri listelemeleri, ilgili seriye ait bütün ilan adreslerini veri tabanında toplamak için taranıp kaydedilmiştir. Her ilan, sayfa başında sıralaması ve içeriği ilan sahiplerinin girmiş olduğu araç bilgilerine göre değişmekle birlikte bir genel bakış bölümü ile başlar. Araçların kendi segmentlerindeki fiyat farkının en önemli faktörlerinden biri olan, aracın hasar geçmişi ve kozmetik durumu için hasar kaydı ve tramer bilgisi ilan sayfasının açıklama kısmında yer alır. İlan sahipleri tarafından girilen araca ait teknik bilgiler, ilgili özellik başlıklarının yapay zeka modeline dahil edilmesinin öneminin araştırılması amacıyla geniş bir yelpazede toplanmıştır. İlk adımda toplanılan ilan adreslerine ulaşıp sırasıyla “Genel Bakış”, “Hasar Kaydı ve Tramer” ve “Teknik Bilgiler” kısımlarında yer alan veriler toplanmıştır.



Resim 1: İlanlara ait genel bakış ve hasar kaydı/tramer kısımları.



Resim 2: İlanlara ait teknik bilgiler kısmı.

1.3.3. Veri Ön İşlemesi

1.3.3.1. Eksik ve Hatalı Verilerin Temizlenmesi

Araçların fiyatının belirlenmesinde karar verici nitelikte olduğu için marka, model, seri veya fiyat gibi özelliklerde eksik veriye sahip olan ilanlar veri setinden çıkarılmıştır. Hatalı veri veya başka bir sütunla aynı anlama sahip olma durumlarında kullanışsız hale gelen nitelikler veri setinden çıkarılmıştır.

```
# drop rows which has a NaN value for some attr
df.dropna(subset=['price'], inplace=True)
df.dropna(subset=['marka'], inplace=True)
df.dropna(subset=['model'], inplace=True)
df.dropna(subset=['seri'], inplace=True)
df.dropna(subset=['arac_turu'], inplace=True)

# drop columns
df.drop('motor_hacmi', inplace=True, axis=1) #
df.drop('tramer_kaydi', inplace=True, axis=1) #
df.drop('sanziman', inplace=True, axis=1) # alm
```

Resim 3: NaN değerleri bulunduran satırların setten düşürüldüğü uygulama.

1.3.3.2. Verilerin Tek Tipleştirilmesi

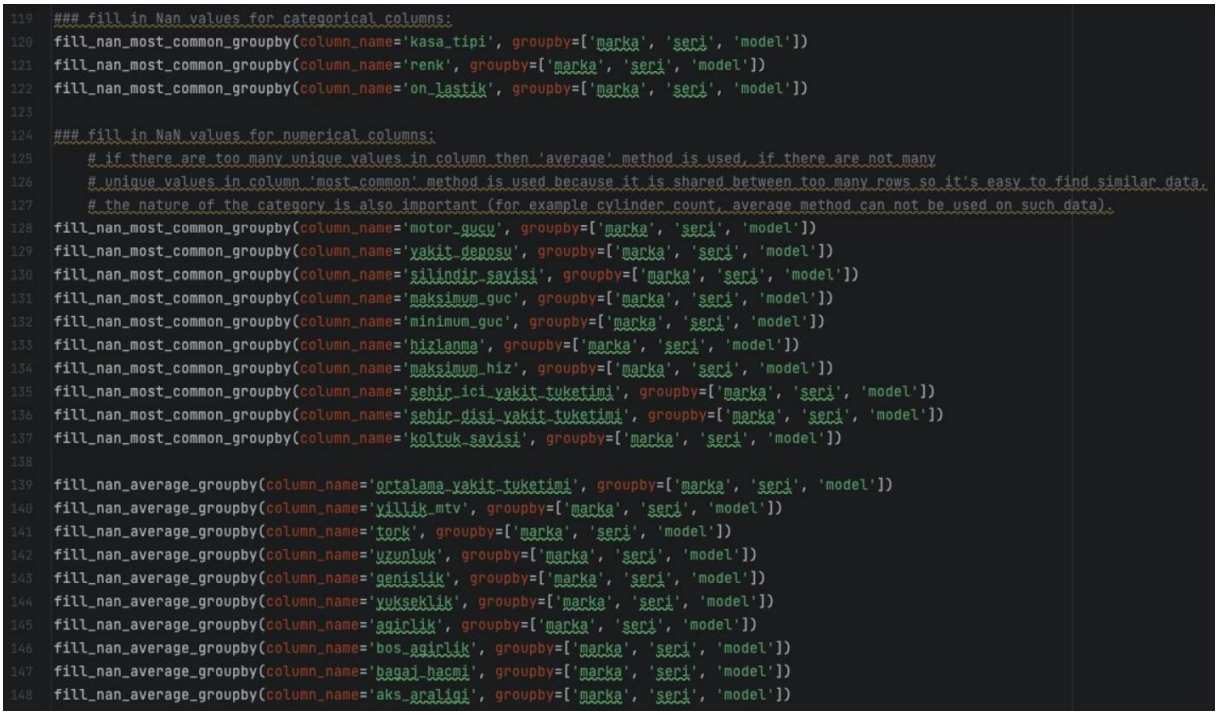
Veri setindeki nitelik değerleri, aynı anlamı taşımalarına rağmen çeşitli şekillerde ifade edilmiş olabilir. Aynı anlama gelen fakat farklı şekilde ifade edilmiş nitelik değerleri, temsil etmeye en uygun görülen şekilde tek tipleştirme uygulanmıştır.

```
85  # change a value with a similar one for a column because it almost has the same meaning
86  df['vites_tipi'] = df['vites_tipi'].replace('Yarı Otomatik', 'Otomatik')
87  df['yakit_tipi'] = df['yakit_tipi'].replace('LPG & Benzin', 'Benzin')
88
89  # group relative categories in a column
90  group_values_in_column(column_name='kasa_tipi', value='Hatchback')
91  group_values_in_column(column_name='kasa_tipi', value='Station wagon')
92  group_values_in_column(column_name='kasa_tipi', value='Coupe')
93  group_values_in_column(column_name='kasa_tipi', value='Roadster')
94  group_values_in_column(column_name='kasa_tipi', value='Sedan')
95  group_values_in_column(column_name='kasa_tipi', value='MPV')
96  group_values_in_column(column_name='renk', value='Gri')
97  group_values_in_column(column_name='renk', value='Yesil')
98  group_values_in_column(column_name='renk', value='Mavi')
99
```

Resim 4: Veri tek tipleştirme uygulaması.

1.3.3.3. Eksik Verilerin Tamamlanması

Eksik verilere sahip özniteliklerde, verilerin tamamlanması mümkün olduğunda, silme yerine tamamlama yöntemi tercih edilmiştir. Bu sayede veri setindeki boşluklar kapatılarak daha kapsamlı bir analiz imkanı elde edilmiştir. Verinin doğasına göre ortalama değerin atanması veya en sık rastlanan değerin kullanılması tercih edilmiştir.

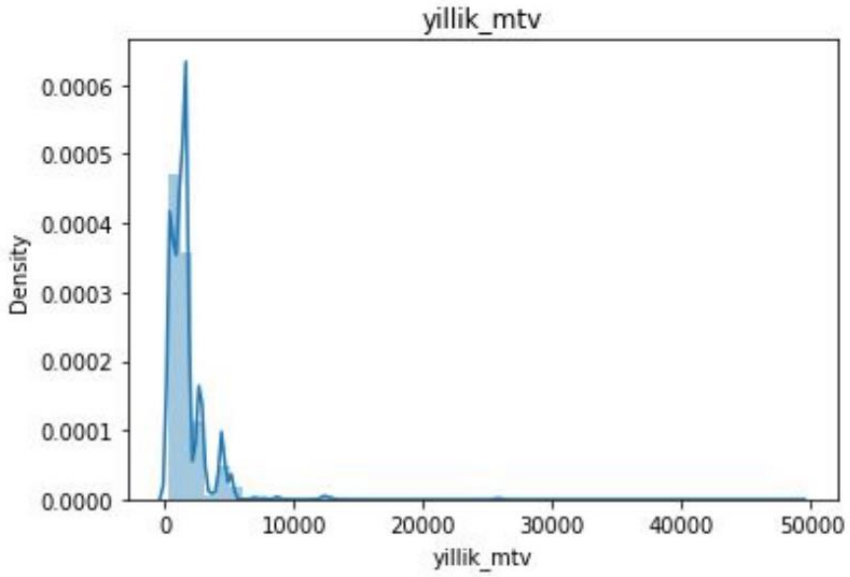


```
### fill in Nan values for categorical columns:
fill_nan_most_common_groupby(column_name='kasa_tipi', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='renk', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='on_lastik', groupby=['marka', 'seri', 'model'])

### fill in NaN values for numerical columns:
    # if there are too many unique values in column then 'average' method is used, if there are not many
    # unique values in column 'most common' method is used because it is shared between too many rows so it's easy to find similar data.
    # the nature of the category is also important (for example cylinder count, average method can not be used on such data).
fill_nan_most_common_groupby(column_name='motor_gucu', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='yakit_deposu', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='silindir_sayisi', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='maksimum_guc', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='minimum_guc', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='hizlanma', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='maksimum_hiz', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='sehir_ici_yakit_tuketimi', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='sehir_disi_yakit_tuketimi', groupby=['marka', 'seri', 'model'])
fill_nan_most_common_groupby(column_name='koltuk_sayisi', groupby=['marka', 'seri', 'model'])

fill_nan_average_groupby(column_name='ortalama_yakit_tuketimi', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='yillik_mtv', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='tork', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='uzunluk', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='genislik', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='yukseklik', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='agirlik', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='bos_agirlik', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='bagaj_hacmi', groupby=['marka', 'seri', 'model'])
fill_nan_average_groupby(column_name='aks_araligi', groupby=['marka', 'seri', 'model'])
```

Resim 5: İlanlara ait teknik bilgiler kısmı.

1.3.3.4. Outlier Verilerin Temizlenmesi

Veri setindeki anormal veya aykırı değerleri belirleyip bunları düzeltme veya çıkarma işlemi uygulanmıştır. Outlier veriler, genellikle diğer verilere göre önemli ölçüde farklı olan ve istatistiksel analizleri etkileyebilecek veri noktalarıdır. Veri setindeki gürültüyü azaltabilir, yanlı veya dengesiz sonuçlara yol açabilecek etkileri en aza indirebilir ve daha sağlıklı bir model oluşturmayı hedefler.



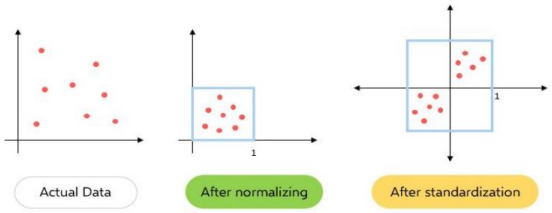
Resim 6: Aykırı/dengesiz sonuçlara yola açabilecek bir özellik örneği.

1.3.4. Özelliklerin Keşfi ve Mühendisliği

1.3.4.1. Nümerik Verilerin Dönüştürülmesi

- Standardizasyon: nümerik değerlerin ortalaması 0 ve standart sapması 1 olacak şekilde dönüştürülmesidir.

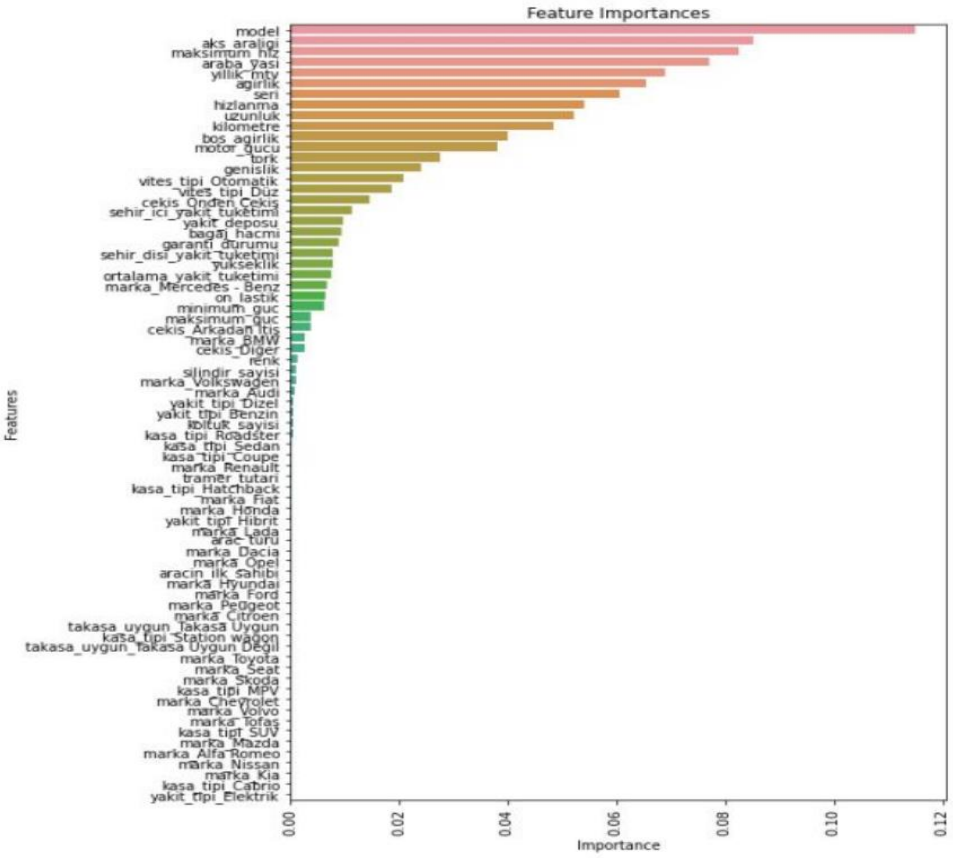
- Normalizasyon: nümerik değerlerin belirli bir aralıkta (genellikle 0 ile 1 arasında) yeniden ölçeklendirilmesidir.



Resim 7: Standardizasyon sonrası istenen veri aralığı yakalanmaktadır.

1.3.4.2. Veri/Boyut İndirgeme

Feature reduction'ın temel amacı, gereksiz, tekrarlayan veya gürültülü öznitelikleri belirleyerek veri setini daha kullanışlı hale getirmektir. modelin performansını iyileştirirken veri setinin anlaşılabilirliğini ve hesaplama verimliliğini artırabilir. Seçilen Öznitelikler: Marka, Seri, Model, Araba Yaşı, Kilometre, Yıllık MTV, Motor Gücü, Vites Tipi, Garanti Durumu.



Resim 8: İlanlarda yer alan özelliklere ait skor/önem tablosu.

1.3.5. Modelin Oluşturulması

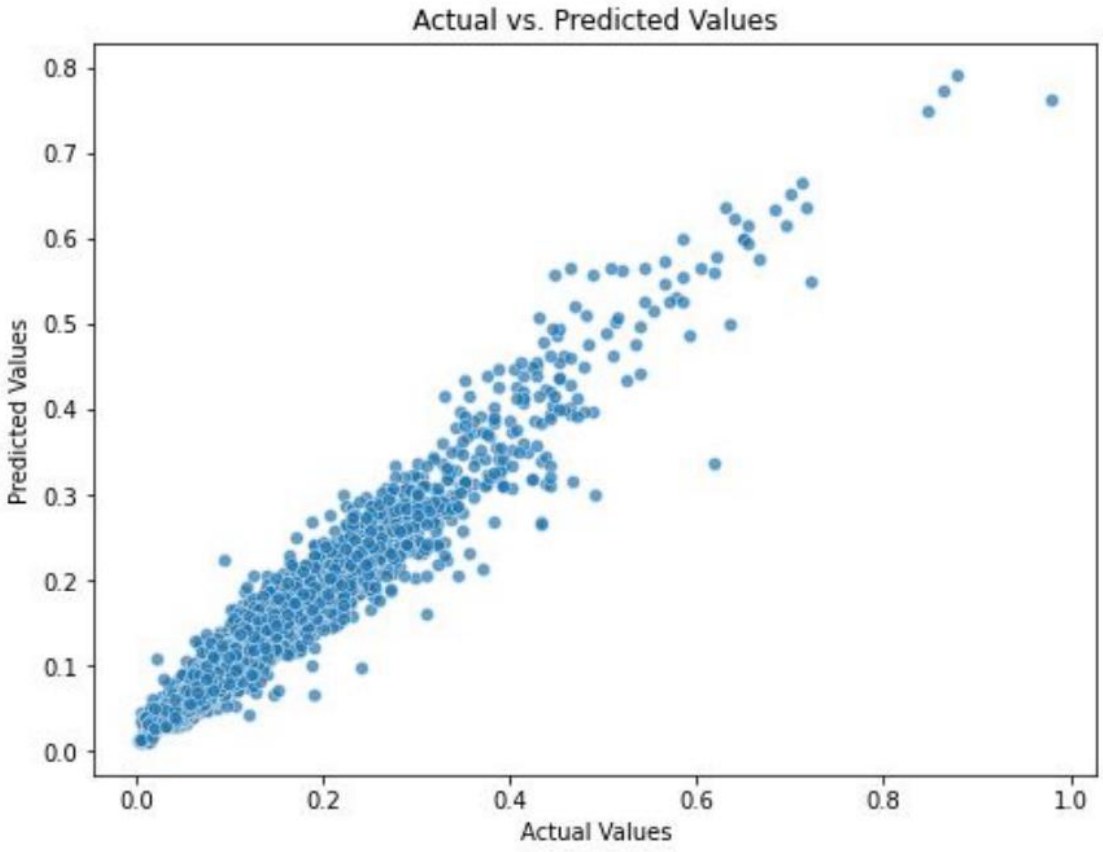
1.3.5.1. Makine Öğrenimi Algoritması

- Outlier ve noisy veri setlerine karşı dayanıklı,
- Büyük ve yüksek boyutlu veri setlerinde başarılı,
- Overfitting yaşanma ihtimali daha düşük,
- Karmaşık non-linear ilişkileri yakalamada başarılı,
- Tpot ile yapılan auto-ml testleri en yüksek başarıyı yakaladı.

1.3.5.2. Random Forest Algoritması Sonuçları

● Root Mean Squared Error: 0.019092 Tahmin hatalarının ortalama büyüklüğünü ölçen yaygın olarak kullanılan bir metriktir. Başka bir deyişle, tahminlerimizin gerçek değerlerden ortalama olarak ne kadar sapma gösterdiğini bize anlatır.

● R-squared Score: 0.939086 Bir modelin uygunluğunun göstergesini olarak, 0 ile 1 arasında bir değer alır. Modeldeki bağımsız değişkenler tarafından açıklanan bağımlı değişkendeki (fiyat) varyansın ne kadarını açıkladığının bir ölçütüdür.



Resim 9: Tahmin edilen ve gerçek değerler arasındaki tutarlılık düzeyi grafik yoluyla gösterilmiştir.

1.3.6. API Yapısı

Uygulama ile ilgili sitelerin web hizmetlerine erişmek için bir arabirim tasarımı olarak REST API yapısı uygulandı. REST API yapısı sayesinde, HTTP metotlarını (GET, POST, PUT, DELETE) kullanarak bir web hizmetine veri gönderilerek veya web hizmetinden veri alınarak etkileşim kurulabildi. REST API yapısı, HTTP protokolünü kullandığı için farklı platformlar arasında uyumlu olduğu gözlemlendi.

Uygulamada veri sağlayıcı olarak, FastAPI yazılım çatısı (framework) kullanılarak REST API yapısı oluşturuldu. FastAPI, Python yazılım çatısı olarak kullanıldı ve hızlı ve güçlü bir şekilde REST API oluşturmak için tercih edildi. REST API arayüzünün bitiş noktaları ve bu bitiş noktalarının kullanım şekilleri tanımlandı ve bu tanımlamalar kullanılarak otomatik olarak API belgelemesi oluşturuldu.

REST API'ya erişim için Retrofit kütüphanesi kullanıldı. Retrofit, geliştiricilerin uygulamaları için bir API arabirimi oluşturmasına olanak tanıyan, Android için oluşturulmuş bir yazılım kütüphanesidir. Retrofit, HTTP istekleri yapmak ve bu isteklerin cevaplarını almak için etkin bir şekilde kullanıldı.

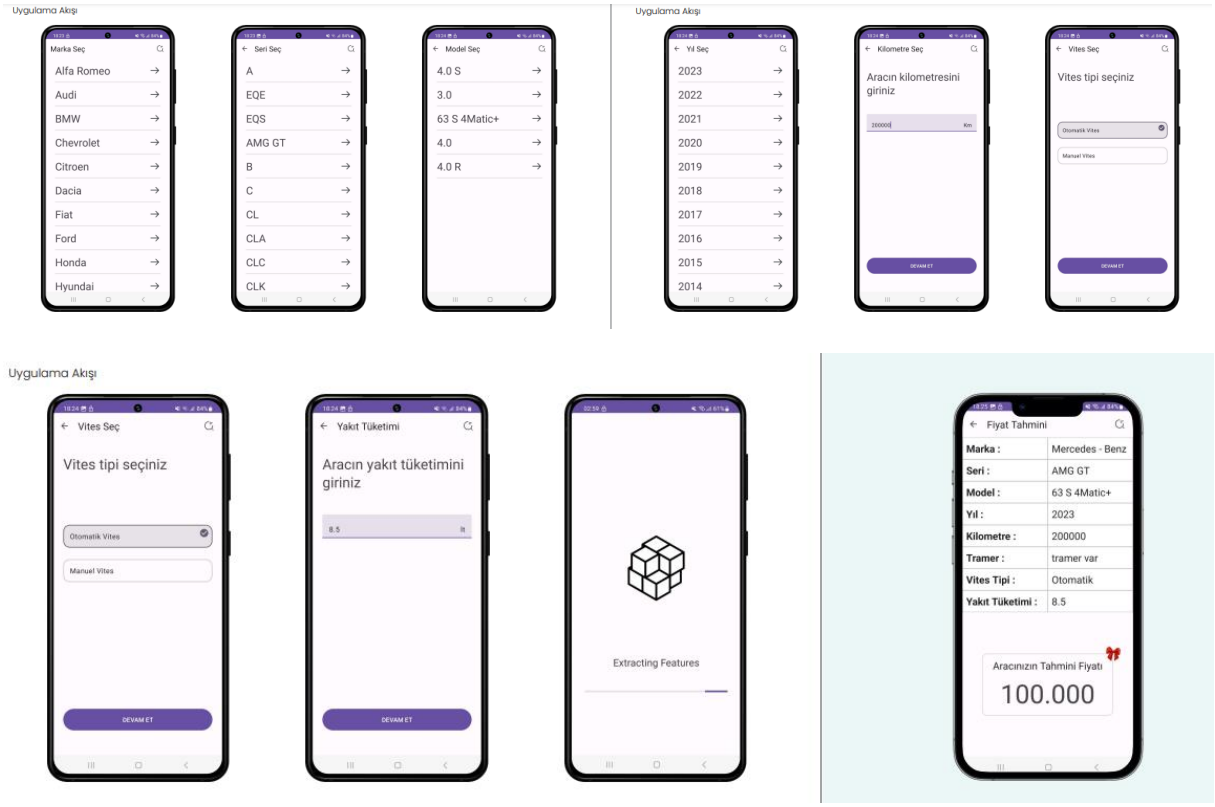
HTTP istekleri oluşturmak, yönetmek ve çözümlemek için OkHttp kullanıldı. OkHttp, Android ve Java uygulamaları için verimli bir HTTP ve HTTP/2 istemcisidir. Retrofit ve OkHttp birlikte kullanılarak, REST API ile hızlı ve güvenilir bir şekilde iletişim kurulması sağlandı.

1.3.7. Android Uygulama Yapısı

Uygulama mimarisi için belirlenmesi gereken uygulama içi sınırları ve uygulamayı bölümlendirirken takip edilecek ilkeler için standart android uygulama yapısı(material design) takip edildi. Bu uygulama, kullanıcının makine öğrenmesi modeline backend iletişimini sağlayarak erişmesi için bir arayüz görevi üstlenir. Kullanıcı dostu bir tasarım ile, gerekli parametreleri akış halinde alır ve sonunda modelin tahminlediği sonucu kullanıcıya sunarak görevini tamamlar. Böylece, kullanıcının kullanım akışı modelin tahminlemesiyle sonlanır ve istenen sonuç görsel olarak sunulur.
Uygulamanın program yapısında, asenkron programlama tekniği ve Kotlin dilinin sunduğu özellik olan Coroutine uygulamaları kullanıldı. Bu durumda, asenkron programlama ile uygulama verimliliği ve kullanıcı deneyimi iyileştirilmeye çalışılmıştır. Coroutine, Kotlin dilinin bir özelliği olarak kullanıldı ve asenkron programlama için kullanılan, thread tarafından yürütülen görev parçacıklarıdır. Kotlin Coroutine, çoklu iş parçacığı (threading) tekniklerine benzer şekilde çalışır, ancak daha hafif bir yapıya sahip olduğu ve daha az kaynak tükettiği gözlemlenmiştir.

1.4. Uygulama

Uygulama akışınında sırasıyla ürün marka, seri, model, yıl, kilometre, vites ve yakıt tüketimi parametreleri için kullanıcı girdisi alınır. Uygulamadaki son sayfa ile birlikte, kullanıcıdan alınan ilgili araç parametreleri, Rest API aracılığı ile araç fiyat tahmini için tasarladığımız makine öğrenmesi modeline ulaşmış ve yine API üzerinden sonuçları mobil uygulamada kullanıcıya gösterilir.



Resim 10: Mobil uygulamaya ait çalıştırma adımları.

1.5. Ekler

```python
from bs4 import BeautifulSoup
import requests

from car_db import insert_listing_url, ListingUrl

headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.3'
}

test_url = "https://www.arabam.com/ikinci-el/otomobil/renault"
brands = ["alfa-romeo", "audi", "bmw", "chevrolet", "citroen", "dacia", "fiat", "ford", "honda", "hyundai", "kia",
          "lada", "mazda", "mercedes-benz", "nissan", "opel", "peugeot", "renault", "seat", "skoda", "tofas", "toyota",
          "volkswagen", "volvo"]

base_url = "https://www.arabam.com"
second_hand_cars_url = "/ikinci-el/otomobil"

all_models = []


def discover_brand_models(brand_url: str):
    response = requests.get(base_url + brand_url, headers=headers)
    doc = BeautifulSoup(response.text, "html.parser")
    tags = doc.find_all("a", class_="list-item")

    for tag in tags:
        href = tag['href']
        if href.startswith(brand_url+'-'):
            all_models.append(href)

```

```python
def discover_all_model
    for brand in brand
        discover_brand


def add_listing_url_to
    listing_url = List
    listing_url.url =

    insert_listing_url


def gather_car_listing
    brand_listing_resu
    doc = BeautifulSou
    tags = doc.find_al

    for tag in tags:
        a_tag = tag.fi
        if a_tag and '
            add_listin

    next_page_tag = do
    if next_page_tag i
        gather_car_lis
    else:
        print("end of


def gather_all_brands_
    discover_all_model

    for model in all_m
        brand_listing_
        gather_car_lis

gather_all_brands_car_
```

Resim 11: Seçilen markalara ait bütün alt modellerdeki tekil ilanların toplandığı uygulama.

```python
def build_get_details_url(listing_id: str) -> str:
    return f"https://www.arabam.com/advertDetail/details?id={listing_id}"


def scrape_short_description(features_dict: dict, doc: BeautifulSoup) -> dict[str, str]:
    container = doc.find("ul", class_="w100 cf mt12 detail-menu")
    if container is None:
        return None
    tags = container.find_all("li", class_="bcd-list-item")

    for tag in tags:
        spans = tag.find_all('span')
        if len(spans) == 2:
            key = spans[0].text.strip().removesuffix(':')
            value = spans[1].text.strip()
            features_dict[key] = value

    return features_dict


def scrape_details(features_dict: dict) -> dict[str, str]:
    listing_id = features_dict["İlan No"]
    fetch_url = build_get_details_url(listing_id)

    details_response = requests.get(fetch_url, headers=headers)
    details_doc = BeautifulSoup(details_response.text, "html.parser")
    tags = details_doc.find_all("dl", class_="cf vertically-centered-big m0 technical-info pl20")

    for tag in tags:
        spans = tag.find_all('span')
        if len(spans) == 2:
            key = spans[0].text.strip()
            value = spans[1].text.strip()
            features_dict[key] = value
    return features_dict
```

```python


def scrape_price(features_dict: dict, doc: BeautifulSoup) -> dict[str, str]:
    tag = doc.find("p", class_="font-default-plusmore bold ls-03")
    if tag is not None:
        features_dict["price"] = tag.text.strip().removesuffix(" TL")
        return features_dict
    else:
        print("cannot find")


        109
    def scrape_tramer_info(features_dict: dict, doc: BeautifulSoup) -> dict[str, str]:
        wrecked_tag = doc.find("div", class_="cf pl4 pr4 font-default-minus")
        features_dict["is_wrecked"] = False
        if wrecked_tag is not None:
            p_tag = wrecked_tag.find("p")
            if p_tag is not None and p_tag.text.strip() == "Bu araç ağır hasar kayıtlıdır.":
                features_dict["is_wrecked"] = True

        if features_dict["is_wrecked"]:
            return features_dict

        tramer_tag = doc.find("div", class_="cf pl4 pr4 font-default-minus pt20 border-top-gray2")
        if tramer_tag is not None:
            p_tag = tramer_tag.find("p")
            span_tag = tramer_tag.find("span", class_="pl4 bold")
            if p_tag is not None and p_tag.text.strip() == "Tramer tutarı yok":
                features_dict["tramer_kaydi"] = False
                features_dict["tramer_tutari"] = 0
            elif span_tag is not None:
                features_dict["tramer_tutari"] = span_tag.text.strip().removesuffix(" TL")
                features_dict["tramer_kaydi"] = True
            else:
                features_dict["tramer_kaydi"] = None
                features_dict["tramer_tutari"] = None
        else:
            features_dict["tramer_kaydi"] = None
            features_dict["tramer_tutari"] = None

        return features_dict
```

Resim 12: Toplanan bütün tekil ilanların taranarak özelliklere erişilip elde edilen verilerin veri tabanına kaydedildiği uygulama.

```python
from fastapi import FastAPI
from pydantic import BaseModel

from predictor import predicate

app = FastAPI()


class PredictionParameters(BaseModel):
    series: str = None
    model: str = None
    year: str = None
    kilometer: str = None
    mtv: str = None
    power: str = None
    transmission: str = None
    guarantee: str = None


class PredictionResponse(BaseModel):
    price: int


@app.post("/predicate", response_model=PredictionResponse)
def create_item(item: PredictionParameters) -> PredictionResponse:
    isAutomaticTransmission = "1" if item.transmission == "automatic" else "0"
    isManuelTransmission = "1" if isAutomaticTransmission == "0" else "0"

    print(item)
    price = predicate("0", item.series, item.model, item.year, item.kilometer, item.mtv, item.powe
                      isManuelTransmission,
                      isAutomaticTransmission, item.guarantee)

    return PredictionResponse(price=price)
```

Resim 13: Fast API ile mobil uygulama arasındaki etkileşim için gereken isteklerin yürütüldüğü uygulama.

```python
import pandas as pd
from joblib import load

max = 4999000
min = 60000

def predicate(price, seri, model, araba_yasi, kilometre, yillik_mtv, motor_gucu, vites_tipi_Düz, vites_tipi_Otomatik,
              garanti_durumu) -> int:
    data = {
        'price': [price],
        'seri': [seri],
        'model': [model],
        'araba_yasi': [araba_yasi],
        'kilometre': [kilometre],
        'yillik_mtv': [yillik_mtv],
        'motor_gucu': [motor_gucu],
        'vites_tipi_Düz': [vites_tipi_Düz],
        'vites_tipi_Otomatik': [vites_tipi_Otomatik],
        'garanti_durumu': [garanti_durumu]
    }

    rf = load('random_forest_regressor.joblib')

    standardization_scaler = load('std_scaler.joblib')
    normalization_scaler = load('norm_scaler.joblib')
    target_encoder = load('target_encoder.joblib')

    new_data = pd.DataFrame(data)

    new_data = target_encoder.transform(new_data)

    standardization_columns = ['kilometre', 'araba_yasi']
    new_data[standardization_columns] = standardization_scaler.transform(new_data[standardization_columns])

    normalization_columns = ['price', 'motor_gucu', 'yillik_mtv']
    new_data_copy = new_data.copy()
    new_data_copy[normalization_columns] = normalization_scaler.transform(new_data_copy[normalization_columns])
    normalization_columns.remove('price')
    new_data[normalization_columns] = new_data_copy[normalization_columns]

    new_data.drop('price', inplace=True, axis=1)
    pred = rf.predict(new_data)

    original_value = (pred * (max - min)) + min
    return original_value
```

Resim 14: Backend tarafından gelen parametreleri makine öğrenimi modeline ileterek tahminleme yaptıran uygulama.

2. SONUÇ VE DEĞERLENDİRME

Yapılan analizler ve çalışmalar, mobil uygulamanın ikinci el ürünlerin fiyat belirleme sürecinde önemli bir araç olduğunu göstermektedir. Kullanıcılar, aradıkları ürünün piyasa taraması sonucunda ortaya çıkan fiyat ortalamasına göre daha bilinçli bir şekilde fiyat belirleme işlemi yapabilmektedirler. Kullanılan veri setinin kalitesini artırmak, modele daha anlamlı ve işlenebilir bilgiler sunmak, modelin performansını ve genel başarısını iyileştirmek için veri ön işlemesi(preprocessing) ve özellik keşfi/mühendisliği(feature engineering) çalışmaları başarıyla gerçekleştirilmiştir. Geliştirilen mobil uygulama, kullanıcılar için kullanımı kolay ve anlaşılır bir arayüz sunmaktadır. Mobil uygulama, Android işletim sistemi kullanılan en az bir aygıtta ve Android Studio Emülatör ortamında başarıyla test edilmiştir. Geliştirilen programlar için 4 ayrı github yazılım deposu(repository) oluşturulup Android, API, makine öğrenimi ve veri madenciliği çalışmalarını içeren kaynak kodları paylaşılmıştır.

Kaynakça

1. Android Developers' App Architecture,
https://developer.android.com/topic/architecture, 15.11.2022.
2. MVVM Architecture Pattern, https://www.geeksforgeeks.org/mvvm-model-viewviewmodel-architecture-pattern-in-android/, 18.10.2022.
3. Asynchronous programming techniques,
https://kotlinlang.org/docs/asyncprogramming.html, 05.01.2023.
4. Coroutines guide, https://kotlinlang.org/docs/coroutines-guide.html, 05.01.2023.
5. Repository Pattern, https://developer.android.com/codelabs/basic-android-kotlintraining-repository-pattern#3, 22.03.2022.
6. What is a REST API?, https://www.ibm.com/topics/rest-apis#anchor-763184502, 2022.
7. FastAPI, https://fastapi.tiangolo.com/, 2023.
8. Retrofit, https://square.github.io/retrofit/, 2023.
9. OkHttp, https://square.github.io/okhttp/, 2023.
10. Beautiful Soup, https://beautiful-soup-4.readthedocs.io/en/latest/, 2023.
11. Doan T. (2015). Selecting Machine Learning Algorithms Using Regression Models. IEEE International Conference on Data Mining Workshop (ICDMW)
12. Raschka S. (2020). Machine Learning in Python: Main Developmentsand Technology
Trends in Data Science, MachineLearning, and Artificial Intelligence, sayfa 2-44.
13. İşkesen S.E. (2014). Development And Implementation Of A Price Prediction System Using Machine Learning Techniques, sayfa 6-62.
14. https://github.com/baristpl/predictor-ml
15. https://github.com/baristpl/predictor-scrape
16. https://github.com/baristpl/predictor-android
17. https://github.com/baristpl/predictor-python