Hafta 3

Botun Başlaması ile Beraber Kullanıcıdan İsim Alarak Sohbeti Bunun Üzerinden Devam Ettirme

Bu bölümde bot ilk açıldığı zaman kullanıcıya selam verecek ardından kullanıcıdan ismini söylemesini isteyecek ve sonrasında geçecek sohbetleri bunun üzerinden devam ettirecek kodları ekleyeceğim.

1. stories.yml dosyasındaki değişiklikler:

```
- story: Greet and ask for user's name
  steps:
  - action: utter_greet
  - action: utter_ask_name
  - intent: inform_name
  - action: action_greet_user
```

2. rules.yml dosyasındaki değişiklikler:

```
- rule: Ask for name at the beginning
  steps:
  - action: utter_greet
  - action: utter_ask_name
```

3. domain.yml dosyasındaki değişiklikler:

```
  - ask_name
  - inform_name
```

bu satırları intents başlığı altına ekliyoruz.

```
  utter_ask_name:
  - text: "What is your name?"

  utter_greet_user:
  - text: "Nice to meet you, {name}!"
```

bu satırları da responses başlığı altına ekliyoruz.

```
  name:
    type: text
    influence_conversation: true
    mappings:
    - type: from_entity
      entity: name
```

bu satırları da slots başlığı altına ekliyoruz.

Veri Yönetimi ve Manipülasyonu

Projede yazdığımız bu bot; kullanıcılarla daha iyi bir etkileşime girmek için kullanıcı ya da kullanıcının talep edebileceği spesifik alanlarda gerekli birikime ve sahip olması gerekiyor. Bunun için gerekli araştırmaları ve kaynak taramalarından sonra yazdığım bot üzerinde şu geliştirmeleri yapmayı gerekli buldum:

A - Veritabanı dosyasını bir betik ile çalıştıran ardından oluşan veritabanı tablosunun verilerini okunmasını sağlayan bir kod parçası

B - Alınan bu bilgileri uygun bir formata manipüle eden bir algoritma

C - Bu senaryoyu botun eğitimi için entegre eden gerekli diğer adımlar

bu üç adımdan önce gerekli yetkinliğe sahip olmak için `Numpy` ve `Pandas` kütüphanelerine çalıştım.

Veri Yönetimi ve Manipülasyonu için `Numpy` Kütüphanesi Çalışmaları

Literatür taramasından sonra bu iş için ilk olarak `Python - Numpy` kütüphanesini öğrenmem gerektiğini anladım. Bu kütüphaneye çalışırken bir video kaynak ve numpy dokümanını kullandım. Bu kütüphanede çalıştığım noktalar üzerine aşağıda kısa örnekler vereceğim:

- **Numpy `array` yapıları:**

```
import numpy as np

arr1 = np.array([1, 2, 3, 4, 5])

print("tek boyutlunun 4. elemanı :", arr1[4])

arr2 = np.array([[10, 20, 30], [40, 50, 60], [70, 80, 90]])

print("üç boyutlunun 2ye 2si :", arr2[2][2])
```

- **Numpy `arange` işlevi:**

```
print(np.arange(10, 20)) # -> 10 ile 20 arasında değerler sıralar
print(np.arange(10, 20, 3)) # -> 10 ile 20 arasında 3'er 3'er atlayarak değerler
sıralar

print(np.zeros(10)) # -> 10 tane 0 elemanı barındıran bir array (default olarak
double sayı atanır)
print(np.zeros((2, 2))) # -> 2x2'lik arrayi yine 0 ile dolduruyor

print(np.ones(10)) # -> 10 tane 1 elemanı barındıran bir array (default olarak
double sayı atanır)
print(np.ones((2, 2))) # -> 2x2'lik arrayi yine 1 ile dolduruyor

print(np.linspace(0, 100, 5)) # 0 ile 100 arasını 5 eşit parçaya bölerek double
array oluşturur

print(np.eye(4)) # 4x4'lük birim matris oluşturur

print(np.random.randint(0, 10)) # 0 ile 10 arası random bir integer, başlangıcın
default değeri 0'dır
print(np.random.randint(0, 10, 5)) # 5'lik random int array

print(np.random.rand(5)) # 0 ile 1 arasında 5 elemanlı bir array

print(np.random.randn(5)) # 0'ın etrafında gaussioan distribution yaparak 5
elemanlı bir array döner
```

- **Numpy matris işlemleri:**

```
arrayim = np.arange(30)
print(arrayim.reshape(6, 5)) # verilen arraye ait elemanları 5x5'lik bir matrise
yerleştirir
# burada verilen 2 parametre array için fix olmalı mesela 6x5 = 30 şeklinde...

newArray = np.random.randint(1, 100, 10)
print(newArray)
print(newArray.max()) # arrayin en büyük değerini döner
print(newArray.min()) # arrayin en küçük değerini döner
print(newArray.sum()) # tüm değerlerin toplamını döner
print(newArray.mean()) # tüm değerlerin ortalamasını döner
print(newArray.argmax()) # en büyük değerin indeksini döner
print(newArray.argmin()) # en küçük değerin indeksini döner


detArray = np.random.randint(1, 100, 25)
print(detArray)
detArray = detArray.reshape(5, 5)
print(detArray)

x = np.linalg.det(detArray) # -> detArray arrayinin determinantını bulur
print("detArray'in determinantı: ", x)
```

- **Numpy alternatif syntaxlar:**

```
arr = np.arange(1, 10)

print(arr)
print(arr[1:5]) # -> 1 ile 5. indeks arasındaki sayılar
print(arr[::2]) # -> tüm değerleri bas ama 2 atlaya atlaya bas

arr[0:3] = 25
print(arr)

arr = np.arange(1, 10)
arr2 = arr

arr2[:3] = 100
print(arr2)
print(arr)

# eşitlemeden sonra arr2'yi değiştirsek arr de aynı şekilde değişir (pointer
meselesi yüzünden)

# bu durumun yaşanmaması için:
arr = np.arange(1, 10)
arr2 = arr.copy()

newArray = np.arange(1, 21)
newArray = newArray.reshape(5, 4)

print(newArray[:, :2]) # -> çift boyutlu matrisde tüm satırları alırken
sütunların sadece ilk ikisini alıp döner

arr = np.arange(1, 11)
print(arr > 3) # her indeksteki değeri 3 ile karşılaştırıp true ya da false
ataması yapar

booleanArray = arr > 3

print(arr[booleanArray]) # sadece true olan değerleri tutar
```

Veri Manipülasyonu için `Pandas` Kütüphanesi Çalışmaları

Veri tabanından alınan veriler genel olarak pandas'ın güçlü, etkili işlevleri ve dataframe yapıları ile Python dilinde manipüle edilir. Bu yüzden bu kütüphane üzerinde gerektiği kadar bilgi edinmek için yine aynı video kaynak ve pandas dokümanını kullandım.

Bu kütüphanede çalıştığım noktalar üzerine aşağıda kısa örnekler vereceğim:

1. Series

Series, Pandas kütüphanesinde tek boyutlu diziler olarak kullanılır. Series, etiketli verilere erişim sağlar ve veri analizi yaparken kullanışlıdır. Aşağıda bazı Series oluşturma yöntemleri gösterilmiştir:

```
import pandas as pd
import numpy as np

label_list = ["Ahmet", "Esma", "Leyla", "Yusuf", "Ali"]
data_list = [10, 20, 30, 40, 50]

# Series oluşturma
print(pd.Series(data=data_list, index=label_list))

# Numpy array'den Series oluşturma
npArray = np.array([10, 20, 30, 40, 50])
print(pd.Series(npArray))

# Dictionary'den Series oluşturma
dataDictionary = {"Ahmet": 10, "Esma": 100, "Zehra": 50, "Ali": 5}
print(pd.Series(dataDictionary))
```

2. DataFrame

DataFrame, Pandas kütüphanesinde iki boyutlu veri yapısıdır (matris gibi). DataFrame, satır ve sütunlardan oluşur ve veri analizi için oldukça esnektir.

```
# DataFrame oluşturma
dataFr = pd.DataFrame(data=randn(3, 3), index=["A", "B", "C"], columns=
["Column1", "Column2", "Column3"])
print(dataFr)

# DataFrame'den bir sütunu seçme
print(dataFr["Column3"])

# DataFrame'e yeni sütun ekleme
dataFr["Column4"] = pd.Series(data=randn(3), index=["A", "B", "C"])
print(dataFr)

# DataFrame'de sütun silme
print(dataFr.drop("Column4", axis=1))

# DataFrame'de satır silme
print(dataFr.drop("A", axis=0))
```

3. DataFrame Filtreleme

DataFrame üzerinde çeşitli koşullarla filtreleme yapabiliriz. Filtreleme, belirli koşullara uyan verileri seçmek için kullanılır.

```
df = pd.DataFrame(data=randn(4, 3), index=["A", "B", "C", "D"], columns=["Col1",
"Col2", "Col3"])

# Filtreleme
print(df[df > -1])

# Daha derin filtreleme
print(df[(df["Col1"] > 0) & (df["Col2"] > 0)])
```

4. Multi Index

Çok seviyeli indeksleme (multi index) yaparak daha karmaşık veri yapıları oluşturabiliriz. Bu yöntem, hiyerarşik verilerin yönetimi için kullanışlıdır.

```
outerIndex = ["Group1", "Group1", "Group1", "Group2", "Group2", "Group2",
"Group3", "Group3", "Group3"]
innerIndex = ["Index1", "Index2", "Index3", "Index1", "Index2", "Index3",
"Index1", "Index2", "Index3"]
hierarchy = pd.MultiIndex.from_tuples(list(zip(outerIndex, innerIndex)))

df = pd.DataFrame(randn(9, 3), hierarchy, columns=["Col1", "Col2", "Col3"])
print(df)
```

5. DataFrame İşlemleri

DataFrame üzerinde çeşitli işlemler yapabiliriz. Örneğin, NaN değerleri doldurma veya silme, grup işlemleri, birleştirme gibi işlemler sıklıkla kullanılır.

```
# NaN değerleri silme
arr = np.array([[10, 20, np.nan], [5, np.nan, np.nan], [21, np.nan, 12]])
df = pd.DataFrame(data=arr, index=["index1", "index2", "index3"], columns=
["Col1", "Col2", "Col3"])
print(df.dropna(axis=1))

# NaN değerleri doldurma
print(df.fillna(value=1))

# Groupby işlemi
dataset = {"Departman": ["Bilişim", "İnsan kaynakları", "Üretim", "Üretim",
"Bilişim", "İnsan kaynakları"],
```

```
            "Çalışan": ["Mustafa", "Jale", "Kadir", "Zeynep", "Murat", "Ahmet"],
"Maaş": [3000, 3500, 2500, 4500, 4000, 2000]}
df = pd.DataFrame(dataset)
print(df.groupby("Departman").sum())

# Merge işlemi
dataset1 = {"A": ["A1", "A2", "A3"], "B": ["B1", "B2", "B3"], "anahtar": ["K1",
"K2", "K3"]}
dataset2 = {"X": ["X1", "X2", "X3", "X4"], "Y": ["Y1", "Y2", "Y3", "Y4"],
"anahtar": ["K1", "K2", "K3", "K4"]}
df1 = pd.DataFrame(dataset1)
df2 = pd.DataFrame(dataset2)
print(pd.merge(df1, df2, on="anahtar"))
```

6. Veri İşleme ve Manipülasyon

Verileri işleyip manipüle ederek analize uygun hale getirebiliriz. Bu süreçte lambda fonksiyonları ve pivot tabloları gibi araçlar kullanılır.

```
# Kolon ekleme
df["Col4"] = pd.Series(randn(4), ["A", "B", "C", "D"])
print(df)

# Lambda fonksiyonları ile veri işleme
print(df["Col2"].apply(lambda x: x * 2))

# Veri sıralama
print(df.sort_values("Column2", ascending=False))

# Pivot tablosu oluşturma
df = pd.DataFrame({"Ay": ["Mart", "Nisan", "Mayıs"], "Şehir": ["Sinop", "Cizre",
"İstanbul"], "Nem": [10, 25, 50]})
print(df.pivot_table(index="Ay", columns="Şehir", values="Nem"))
```

7. Veri Seti Okuma ve Yazma

Pandas, farklı veri kaynaklarından veri okumayı ve bu verileri farklı formatlarda kaydetmeyi kolaylaştırır. Aşağıda bazı örnekler verilmiştir:

```
# CSV dosyası okuma ve yazma
dataset = pd.read_csv("USvideos.csv")
print(dataset.head())
dataset.to_csv("USvideosNew.csv", index=False)

# Excel dosyası okuma ve yazma
# excelset = pd.read_excel("file.xlsx")
# excelset.to_excel("fileNew.xlsx")

# Web'den veri okuma
# new = pd.read_html("link")
```

Bu döküman, Pandas kütüphanesinin temel işlevlerini ve kullanımını botu eğitmek için yeterli bir şekilde özetlemektedir.

A - Veritabanı ve Betik Kodu

Bu kısımda elimde hazır bir veritabanı sunucusu olmadığı için sadece bir `sql` dosyası yazıp bunun üzerinden deneme yaptım. Bu oluşturduğum senaryoda bir giyim firmasına ait bazı veritabanı kayıtları (belli başlı ürünlerin satış sayısı) kullanıldı. Kullanıcı alacağı ürün için bota "X ürünü için en çok tercih edilen renk nedir?" veya "X ürünü için ne tercih etmeliyim?" şeklinde prompt verir. Bot da okuduğu bilgiler ışığında uygun cevabı verir.

Basit `sql` tablosunun içeriği:

```
CREATE TABLE IF NOT EXISTS Alisveris
(
    id INTEGER PRIMARY KEY,
    Urun TEXT NOT NULL,
    Kirmizi INTEGER DEFAULT 0,
    Mavi INTEGER DEFAULT 0,
    Yesil INTEGER DEFAULT 0,
    Sari INTEGER DEFAULT 0
);

INSERT INTO Alisveris (Urun, Kirmizi, Mavi, Yesil, Sari) VALUES ('T-Shirt', 10,
5, 3, 7);
INSERT INTO Alisveris (Urun, Kirmizi, Mavi, Yesil, Sari) VALUES ('Pantolon', 4,
8, 6, 2);
INSERT INTO Alisveris (Urun, Kirmizi, Mavi, Yesil, Sari) VALUES ('Ceket', 6, 3,
5, 1);
INSERT INTO Alisveris (Urun, Kirmizi, Mavi, Yesil, Sari) VALUES ('Elbise', 7, 9,
2, 4);
INSERT INTO Alisveris (Urun, Kirmizi, Mavi, Yesil, Sari) VALUES ('Gömlek', 2, 6,
8, 3);
```

- Bu veritabanından veri okuma ve Python kodu ile manipüle etme:

actions.py:

```
def create_db_from_sql(db_path, sql_file_path):

    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()

    # Tablo varsa oluşturmadan geç
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='Alisveris'")
    result = cursor.fetchone()

    if not result:

        with open(sql_file_path, 'r') as file:

            sql_script = file.read()

        cursor.executescript(sql_script)
        conn.commit()

    else:

        print("Table 'Alisveris' already exists, skipping creation.")

    conn.close()

def read_db_to_dataframe(db_path, query):

    conn = sqlite3.connect(db_path)
    df = pd.read_sql_query(query, conn)
    conn.close()

    return df
```

B - Veri Manipülasyonu

Okunan verilerin uygun formata manipülasyonu:

```
def find_favorite_color(row):

    colors = row[['Kirmizi', 'Mavi', 'Yesil', 'Sari']]
    favorite_color = colors.idxmax()  # En yüksek değere sahip rengi bul
    favorite_amount = colors.max()    # O rengin miktarını bul

    return pd.Series([favorite_color, favorite_amount], index=['Renk', 'Miktar'])

def generate_sentences(df):
```

```
    return df.apply(lambda x : f"{x.name} kategorisinde en çok tercih edilen renk
{x['Renk']} rengidir.", axis = 1)

# ... #

class ActionFavoriteColor(Action):

    def name(self) -> Text:

        return "action_favorite_color"

    def run(self, dispatcher: CollectingDispatcher, tracker: Tracker, domain:
Dict[Text, Any]) -> List[Dict[Text, Any]]:

        # Slottan 'product' değerini al
        product = tracker.get_slot('product')
        print(f"DEBUG: Product slot value is: {product}")

        if not product:

            dispatcher.utter_message(text="Please specify a product name.")
            return []

        # Veritabanı ve SQL dosya yolları
        db_path = 'example.db'
        sql_file_path = os.path.join(os.path.dirname(os.path.realpath(__file__)),
'veritabani.sql')

        # Veritabanını oluştur veya veriyi yükle
        create_db_from_sql(db_path, sql_file_path)

        # Veriyi DataFrame'e çevir
        df = read_db_to_dataframe(db_path, "SELECT * FROM Alisveris")

        # Ürüne göre en çok tercih edilen rengi bul
        if product in df['Urun'].values:

            product_df = df[df['Urun'] == product]
            favorite_color_info = product_df.apply(find_favorite_color,
axis=1).iloc[0]
            response = f"The most preferred color in {product} category is
{favorite_color_info['Renk']}."

        else:
            response = f"No information found for category {product}."

        dispatcher.utter_message(text=response)
        return []
```

C - Diğer Dosyalarda Eklenen Konfigürasyonlar

nlu.yml:

```
- regex: product
  examples: |
```

```
    - T-Shirt
    - Pantolon
    - Ceket
    - Elbise
    - Gömlek
    - Çorap

- intent: ask_product_color
  examples: |
    - Which color is most preferred for [T-Shirt](product)?
    - Which color is really most preferred for [T-Shirt](product)?
    - What is the favorite color for [Pantolon](product)?
    - What is the most favorite color for [Pantolon](product)?
    - Tell me the favorite color for [Ceket](product)
    - Tell me the most favorite color for [Ceket](product)
    - What is the most popular color for [Elbise](product)?
    - What is the most popular color for this [Elbise](product)?
    - Which color do people like most for [Gömlek](product)?
    - Which best color do people like most for [Gömlek](product)?
```

burada ilk kez gördüğümüz regex'in işlevi şudur:

Düzenli ifadeler (regex)

Metinlerde belirli desenleri bulmak ve işlemek için kullanılan araçlardır. Regex, metin arama, eşleştirme ve değiştirme gibi işlemler için kullanılır. Veri analizi ve doğal dil işleme (NLP) projelerinde, regex kullanarak belirli desenlere uygun verileri işleyebiliriz.

Regex Kullanımı:

Düzenli ifadeler, özellikle metin verilerinin işlenmesinde çok faydalıdır. Örneğin, bir e-posta listesinden yalnızca belirli bir desene uyan e-postaları çekmek istersek, nlu.yml dosyasında şöyle bir şey kullanabiliriz:

```
nlu:
- regex: email
  examples: |
    - [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}

- intent: provide_email
  examples: |
    - My email address is [john.doe@example.com](email).
    - You can reach me at [jane_smith@example.com](email).
    - Contact me via [alice@example.com](email).
    - My contact is [bob@example.com](email).
    - Send the information to [info@company.com](email).
    - Here is my email: [support@website.org](email).

- intent: ask_email
  examples: |
    - Can you provide your email address?
    - What is your email?
    - Please tell me your email address.
    - I need your email contact.
    - Could you give me your email?
```

```
    - May I have your email address?
```

- `[a-zA-Z0-9._%+-]+`: Kullanıcı adını temsil eder. Küçük ve büyük harfler, rakamlar, nokta (.), alt çizgi (_), yüzde (%), artı (+) ve eksi (-) karakterlerinden bir veya daha fazla olabilir.
- `@`: Tam olarak "@" karakterini temsil eder. Kullanıcı adı ile domain kısmını ayırır.
- `[a-zA-Z0-9.-]+`: Domain adını temsil eder. Küçük ve büyük harfler, rakamlar, nokta (.) ve eksi (-) karakterlerinden bir veya daha fazla olabilir.
- `\.`: Tam olarak bir nokta (.) karakterini temsil eder.
- `[a-zA-Z]{2,}`: Üst seviye domaini (TLD) temsil eder(.com, gmail.com gibi). Küçük ve büyük harflerden en az iki karakter olmalıdır.

Bizim örneğimizde de formül kullanmadan direkt düz ifadeler yazdık, çünkü ürün isimleri spesifik isimlerdir. Bu şekilde formüllerle ifade edilmeye çalışırsa anlamsız olur. Ama şu noktaya dikkat edilmeli, mesela `nlu` dosyasında ***Çorap*** ürünü olsa da database tablomuzda o ürün yoktu. Böyle bir durumda ***actions.py*** dosyasındaki şu satırların çalışmasını bekleriz:

```
        if product in df['Urun'].values:

            # code #

        else:
            response = f"No information found for category {product}."
```

domain.yml:

`intents` başlığı altına,

```
   - ask_product_color
```

`response` başlığı altına,

```
   utter_favorite_color:
   - text: "Which color do you prefer for this product?"
```

`entities` başlığı altına,

```
   - product
```

`slots` başlığı altına,

```
   product:
     type: text
     influence_conversation: true
     mappings:
     - type: from_entity
       entity: product
```

ve son olarak `actions` başlığı altına da aşağıdaki satırı ekliyoruz:

```
  - action_favorite_color
```

Bu Adımlarda Aldığım Hatalar ve Çözümleri

Hata 1: `No module named 'pandas'`

Hata Mesajı:

```
ModuleNotFoundError: No module named 'pandas'
```

Neden: Bu hata, `pandas` modülünün yüklenmemiş olmasından kaynaklanır.

Çözüm:

```
pip install pandas
```

Bu komut ile `pandas` modülünü yükleyerek hatayı giderdim. Aslında bu kütüphane kendi bilgisayarımda yüklüydü ama sanırım RASA için virtual environment kullandığımız için orası için ayriyeten bunu indirmemiz gerekti.

Hata 2: `Failed to execute custom action 'action_favorite_color'`

Hata Mesajı:

```
Failed to execute custom action 'action_favorite_color'
```

Neden: Bu hata, özel bir eylemi çalıştırmaya çalışırken oluşur. Muhtemel nedenler arasında eksik veya yanlış yapılandırılmış veritabanı bağlantısı olabilir.

Çözüm:
Veritabanı dosyasının ve SQL dosyasının doğru konumda olduğundan emin oldum ve `actions.py` dosyasını kontrol ettim. Normalde iki dosya da aynı dizinde olmasına rağmen actions.py veritabani.sql dosyasını bulamıyordu. Daha sonra `os` kütüphanesini kullanarak işimi sağlama aldığımda çalışmaya başladı.

actions.py:

```
        db_path = 'example.db'
        sql_file_path = os.path.join(os.path.dirname(os.path.realpath(__file__)),
'veritabani.sql')
```

Hata 3: `ClientResponseError: [Errno 2] No such file or directory`

Hata Mesajı:

```
ClientResponseError: [Errno 2] No such file or directory: 'veritabani.sql'
```

Neden: Bu hata, SQL dosyasının doğru konumda olmadığını belirtir.

Çözüm: SQL dosyasının doğru konumda olduğundan emin oldum ve `actions.py` dosyasında doğru dosya yolunu belirttim.

```
# SQL dosyasının yolunu kontrol et
sql_file_path = os.path.join(os.path.dirname(os.path.realpath(__file__)),
'veritabani.sql')
```

Hata 4: `Bota prompt olarak verilen girdinin ürün ismi olarak algılanması`

Hata görüntüsü:

bu sorun `domain.yml` dosyasında yaptığım bir hatadan kaynaklıydı. Sorunu `actions.py` dosyasında aradığım için çözümünü bulmam uzun sürdü.

```
  mood:
    type: text
    influence_conversation: true
    mappings:
      - type: from_text

  product:
    type: text
    influence_conversation: true
    mappings:
    - type: from_entity
      entity: product
```

`mood` slotunun `mappings: - type:` özelliği `from_text` olarak ayarlandığı için kullanıcının girdiği tüm promptu bu slota atar. Ancak bunu engellemek istiyorsak `nlu.yml`'de',

```
- intent: ask_product_color
  examples: |
    - Which color is most preferred for [T-Shirt](product)?
    - What is the favorite color for [Pantolon](product)?
    - Tell me the favorite color for [Ceket](product)
    - What is the most popular color for [Elbise](product)?
    - Which color do people like most for [Gömlek](product)?
```

bu şekilde anahtar kelimeleri belirleyip slot atamasını sağlama alıp `domain.yml`'de slotun `mappings: - type:` özelliği `from_entity` olarak ayaranmalıdır.

Sonuç olarak botun bir database ile etkileşim senaryosu üzerine çalıştığım bu haftada bota eklediğim özellikler ve karşılaştığım sorunlar bunlardı.