

17 Kasım 2023

Python Programlama Dili 

Veri Görselleştirme 

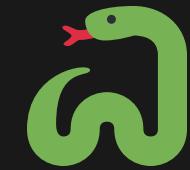
Veri Manipülasyonu ve Analizi 

18 Kasım 2023

Doğrusal Regresyon ile Tahminleme 

Lojistik Regresyon ile Sınıflandırma 

Web Uygulaması 



Python Programlama Dili

- Kurulumlar
- Değişkenler
- Temel Fonksiyonlar
- Veri Tipleri
- Yorum Satırı
- Operatörler

- Koşullar
- Döngüler
- Fonksiyonlar
- Modüller
- Hata Yakalama ve İşleme
- Dosya İşlemleri

Kurulumlar

Python

Download the latest version for Windows

[Download Python 3.12.0](#)

Looking for Python with a different OS? Python for [Windows](#),
[Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python 3.13? [Prereleases](#),
[Docker images](#)

<https://www.python.org/downloads/>

Add python.exe to PATH 

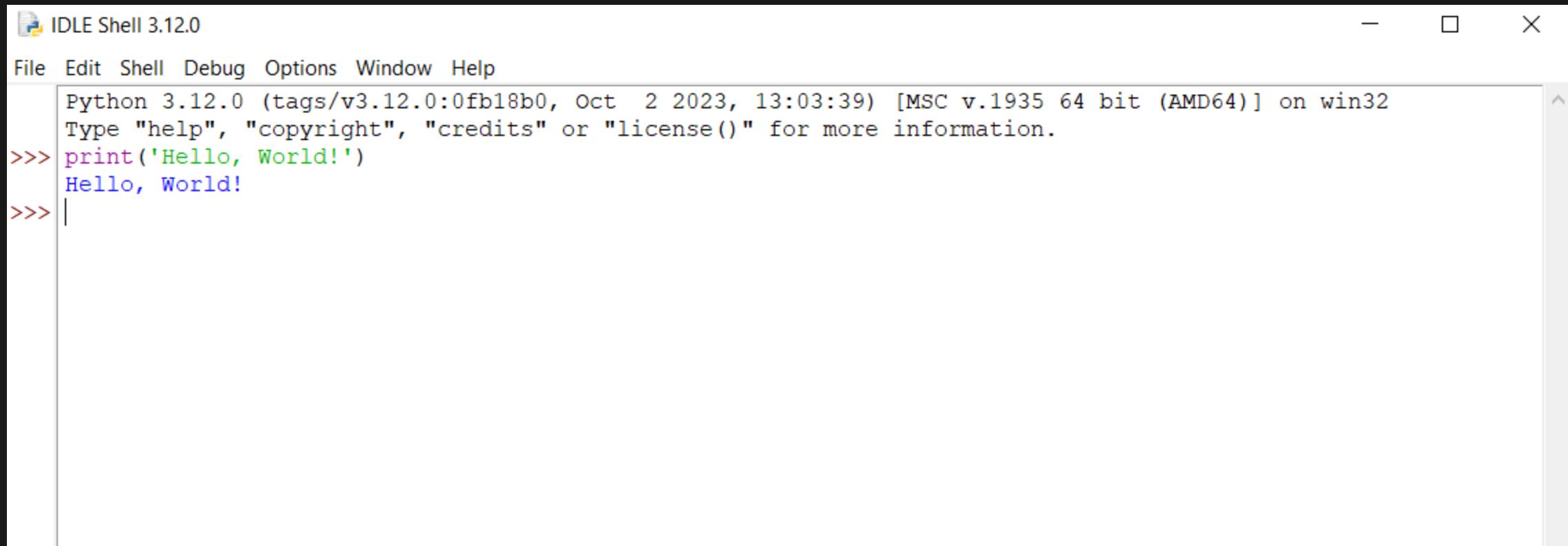
Semantik Versiyonlama (Semantic Versioning)

Major.Minor.Patch

3.12.0

- Major: Potansiyel olarak büyük dil değişiklikleri veya önemli yeni özellikler.
- Minor: Büyük hata düzeltmeleri, yeni özellikler, geliştirmeler ve modüller.
- Patch: Küçük hata düzeltmeleri ve güvenlik düzeltmeleri.

IDLE: Integrated Development and Learning Environment



The screenshot shows the IDLE Shell 3.12.0 interface. The title bar reads "IDLE Shell 3.12.0". The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The main window displays a Python session:

```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print('Hello, World!')
Hello, World!
>>>
```

Visual Studio Code

Code editing.
Redefined.

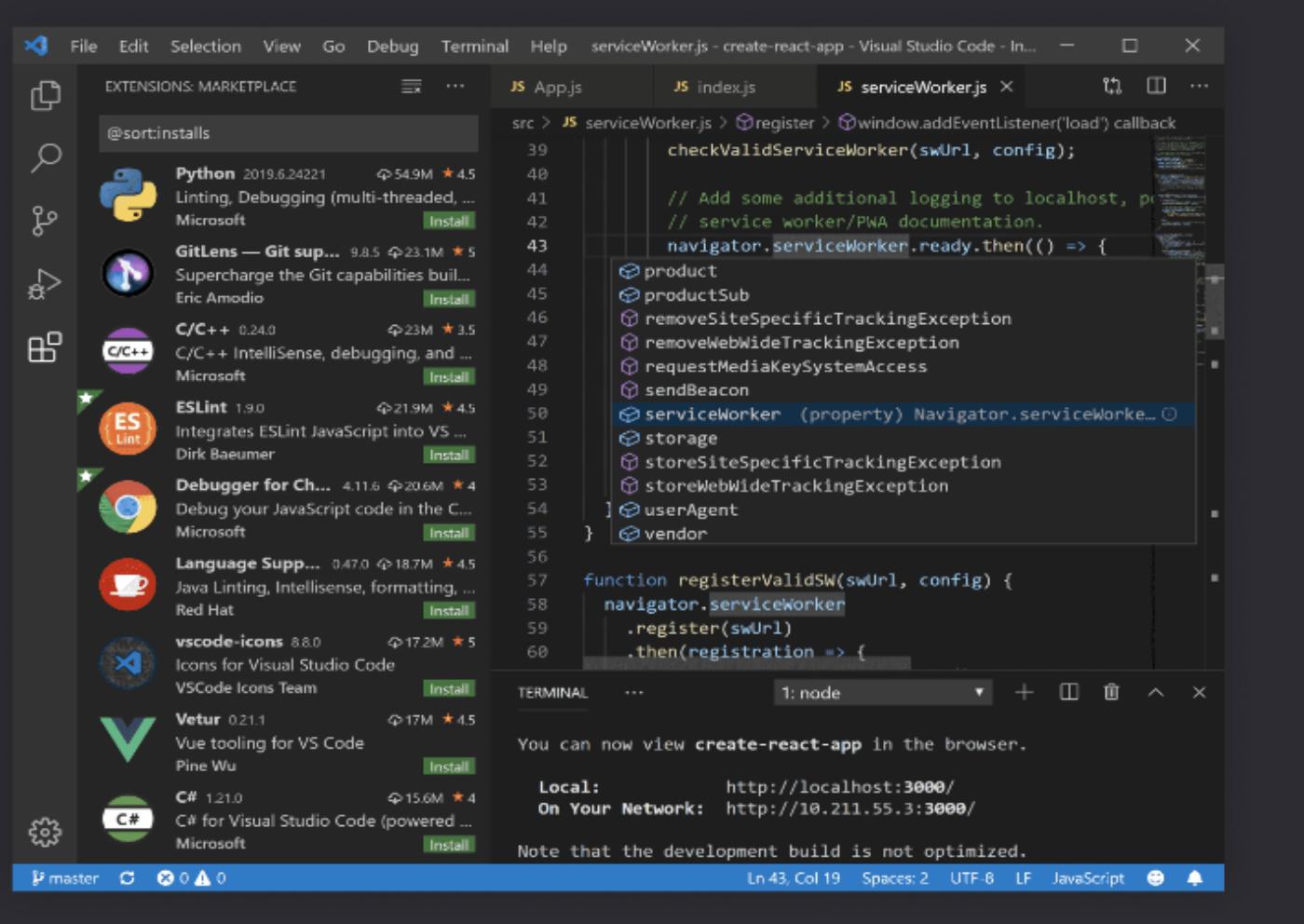
Free. Built on open source. Runs everywhere.

Download for Windows

Stable Build

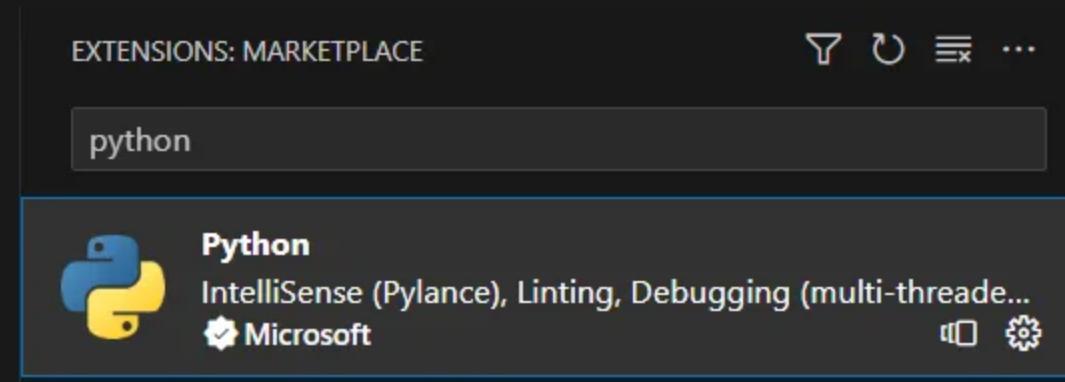
Web, Insiders edition, or other platforms

By using VS Code, you agree to its
[license and privacy statement](#).

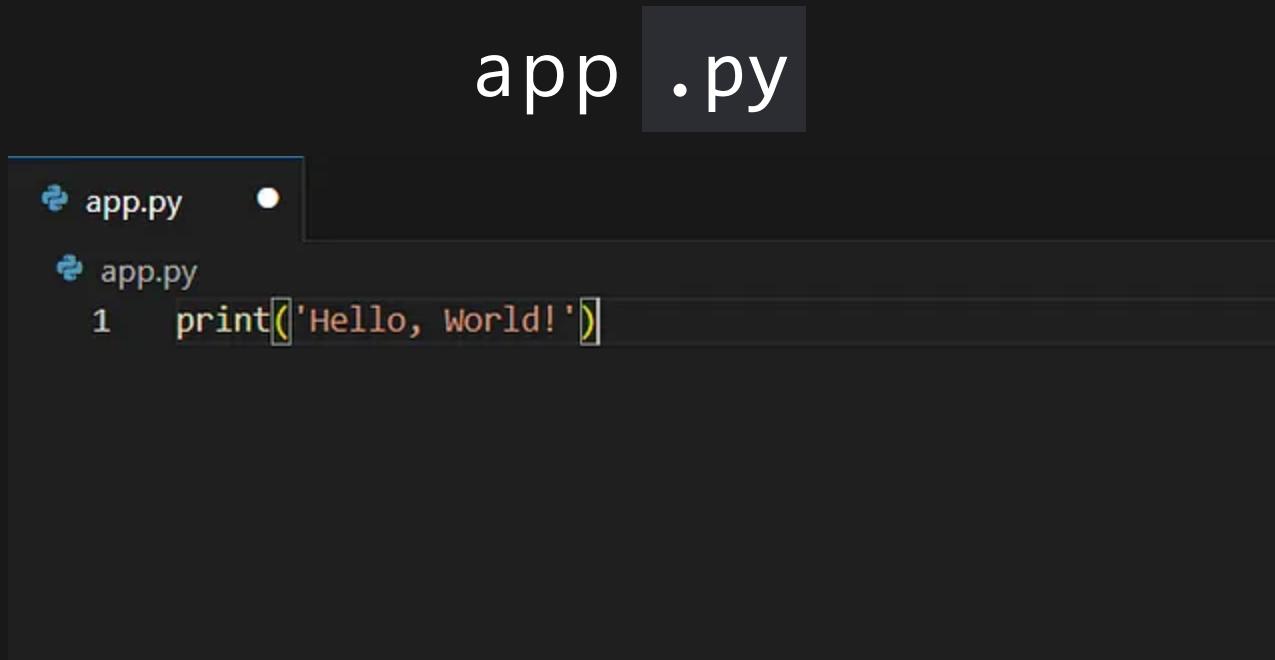


<https://code.visualstudio.com/>

Extensions - Python IntelliSense



Visual Studio Code'da İlk Kod



The image shows a screenshot of the Visual Studio Code interface. At the top, there is a dark gray header bar with the text "app .py" in white. Below this, the main workspace shows a single file named "app.py". The file content is displayed in a light gray code editor area. The code consists of a single line: "1 print('Hello, World!')". The number "1" is in green, indicating it's a line comment, and the rest of the line is in orange, likely representing a string literal.

Değişkenler

```
['Istanbul','Toronto','TK',17,'B77W']

flight = ['Istanbul','Toronto','TK',17,'B77W']

flightnumber1_1 = 'TK'
flightnumber1_2 = 17
aircraft_icao = 'B77W'

print(flight)
```

Değişken İsimlendirme Kuralları

- Özel karakter kullanma ✗
- Sayı ile başlama ✗
- Sayı içermeye ✓
- Python'a ait bir ifade kullanma ✗
- Python'a ait bir ifadenin başına sembol ekleme ✓
- Türkçe harfler ✓ ✗
- Küçük-büyük harflere olan duyarlılık ✓

Değişken İsimlendirme Kuralları - Özel

- camelCase: `flightNumber='TK17'`
- snake_case: `flight_number='TK17'`
- PascalCase: `FlightNumber='TK17'`

Değişkenler ile neler yapılabilir?

- Değişkenler birleştirilebilir.

```
flightnumber1_1 = 'TK'  
flightnumber1_2 = '17'  
flightnumber = flightnumber1_1 + flightnumber1_2
```

- Tırnaklar içerisinde yazılmış bir değişken birden fazla yazdırılabilir.

```
print(flightnumber*3)
```

Değişkenler ile neler yapılabilir?

- Değişkenlerin değerleri kendi aralarında değiştirilebilir.

```
flightnumber1 = 17  
flightnumber2 = 18  
flightnumber1, flightnumber2 = flightnumber2, flightnumber1
```

- Bir değişken silinebilir.

```
del(flightnumber)
```

Temel Fonksiyonlar

Yerleşik Fonksiyonlar

- `print()` : Verilen değeri ekranaya yazdırınmak için kullanılır.
- `type()` : Bir nesnenin türünü döndürmek için kullanılır.
- `help()` : Bir nesnenin belgelemesini görüntülemek için kullanılır.
- `len()` : Bir nesnenin uzunluğunu veya elemanların sayısını döndürmek için kullanılır.
- `input()` : Kullanıcıdan bir giriş almak için kullanılır.

Veri Tipleri

`type()` ?

- **NoneType**: Boş bir değer veya olmayan bir değer.
- **str**: Metin veya karakter dizileri.
- **int**: Tam sayılar.
- **float**: Ondalıklı sayılar.
- **complex**: Gerçek ve sanal bileşenlerden oluşan karmaşık sayılar.
- **bool (boolean)**: Mantıksal değerler.

- **list**: Birden çok elemanı `[]` ile bir arada saklamak için.
- **tuple**: Birden çok elemanı `()` ile bir arada saklamak için. Değiştirilemez.
- **set**: Birden çok elemanı `{ }` ile bir arada saklamak için. Benzersiz.
- **dict**: Anahtar-değer çiftlerini `{ }` ile depolar.

NoneType

```
flightNumber=None  
print(type(flightNumber))
```

str

- Değer, tek tırnak (`'string'`) içine yazarak bir string oluşturulabilir.
- Değer, çift tırnak (`"string"`) içine yazarak bir string oluşturulabilir.
- Değer, üç tırnak (`"""string"""` veya `'''string'''`) içine yazarak bir string oluşturulabilir.

str ile neler yapılabilir? İndeksleme.

```
slogan = 'Widen Your World'
```

- Belirli bir indeksteki karakteri alma

```
slogan[0]
```

- Dilimleme ile alt dize alma

```
slogan[6:10]
```

- Negatif indeksleme

```
slogan[-5:]
```

str ile neler yapılabilir? İndeksleme.

```
slogan = 'Widen Your World'
```

- Pozitif indeksleme

```
slogan[:5]
```

- Sadece çift sayıya sahip indeksleri alma

```
slogan[0:15:2] veya slogan[0::2]
```

- Tersini elde etme

```
slogan[::-1]
```

str ile neler yapılabilir? Formatlama.

```
slogan = 'Widen Your World'
```

- `format()` string metodu ile formatlama:

```
print("{} | Turkish Airlines".format(slogan))
```

- `f`-string formatlama:

```
print(f"{slogan} | Turkish Airlines")
```

str ile neler yapılabilir? Şekil değiştirme.

```
slogan = 'Widen Your World'
```

- String bir ifade özel karakterler ile ayrılabilir.

```
print(*slogan, sep="|")
```

- String bir ifade alt alta yazılabılır.

```
print(*slogan, sep="\n")
```

str ile neler yapılabilir? Şekil değiştirme.

```
slogan = 'Widen Your World'
```

- String bir ifadede boşluklar bırakılabilir.

```
print(*slogan, sep="\t")
```

str ile neler yapılabilir? Bazı metodlar.

```
slogan = 'Widen Your World'
```

```
dir(slogan)
```

- `upper()` : Tüm harfleri büyük hale getirir.

```
print(slogan.upper())
```

- `count()` : Belirtilen alt dizgenin kaç kez geçtiğini sayar.

```
print(slogan.count("o"))
```

str ile neler yapılabilir? Bazı metodlar.

- `replace()` : Belirtilen eski alt dizgeyi yeni alt dizeyle değiştirir.

```
print(slogan.replace("Widen", "W I D E N"))
```

- `join()` : Bir dizeyi birleştirmek için kullanılır.
- `split()` : Bir dizeyi belirli bir ayırıcı (varsayılan olarak boşluk) kullanarak parçalara böler.

```
print("".join(slogan.split(' ')))
```

int

```
fleet_size = 435
print(type(fleet_size))
```

float

```
pi = 3.14  
print(type(pi))
```

complex

```
karmasik = 3 + 2j  
print(type(karmasik))
```

boolean

True : 1, False : 0

```
takeoff = True  
print(type(takeoff))
```

list (liste) []

```
flight = ['Istanbul', 'Toronto', 'TK', 17, 'B77W']
print(type(flight))
```

list (liste) ile neler yapılabilir?

```
flight[0]
```

```
flight[1] = 'Vancouver'
```

- `append()` : Bir listeye eleman ekler.

```
flight.append(True)
```

- `insert()` : Belirtilen indekse bir eleman ekler.

```
flight.insert(1, 'Detroit')
```

- `index()` : Belirtilen değerin indeksini döndürür.

```
flight.index('Vancouver')
```

- `pop()` : Belirtilen indeksteki elemanı listeden çıkarır ve bu elemanı döndürür. Varsayılan: Son eleman.

```
flight.pop(2)
```

tuple (demet) ()

list veri tipinden temel farkları:

- Değiştirilebilirlik: Listeler değiştirilebilir (mutable), demetler değiştirilemez (immutable).
- Parantez kullanımı: Listeler [] ile oluşturulurken, demetler () ile oluşturulur.

```
flight = ('Istanbul', 'Toronto', 'TK', 17, 'B77W')
print(type(flight))
```

set (küme) {}

Temel özelliği: Öğeler benzersiz olmalıdır.

```
flight = {'Istanbul', 'Toronto', 'TK', 17, 'B77W'}  
print(type(flight))
```

dict (sözlük) {}

Key-Value (Anahtar-Değer) çiftinden oluşurlar.

```
flight = {  
    'From': 'Istanbul',  
    'To': 'Toronto',  
    'Flight Number': 'TK17',  
    'Aircraft ICAO': 'B77W'  
}  
print(type(flight))
```

dict (sözlük) ile neler yapılabilir?

```
flight['Flight Number']
```

```
flight['To'] = 'Detroit'
```

- `items()` : Bir sözlüğün tüm anahtar-değer çiftlerini içeren bir liste döndürür.

```
flight.items()
```

- `keys()` : Bir sözlüğün tüm anahtarlarını içeren bir liste döndürür.

```
flight.keys()
```

- `values()` : Bir sözlüğün tüm değerlerini içeren bir liste döndürür.

```
flight.values()
```

- `pop()` : Belirtilen anahtarı içeren öğeyi sözlükten çıkarır ve bu ögenin değerini döndürür

```
flight.pop('Aircraft ICAO')
```

Yorum Satırı

- Yorum satırları, Python yorumlayıcısı tarafından görmezden gelinir.
- Yorum satırları, kodun farklı bölümlerini açıklamak veya geçici olarak devre dışı bırakmak için kullanılabilir.
- Python'da yorum satırları # karakteriyle başlar.
- Çok satırlı yorumlar için her satırın # karakteriyle başlatılması gereklidir. Bunun yerine, üç adet tırnak işaretisi ('' 'veya " ") kullanarak bir string ifadesi içinde yorumlar yazılabılır.

Operatörler

Aritmetik Operatörler

- Toplama: + işaretini ile gerçekleştirilir.

```
revenue = 15823  
cost_of_sales = -11752  
gross_profit = revenue + cost_of_sales
```

- Çıkarma: - işaretini ile gerçekleştirilir.

```
revenue = 15823  
cost_of_sales = 11752  
gross_profit = revenue - cost_of_sales
```

- Çarpma: * işaretini ile gerçekleştirilir.

```
revenue = 15823  
passenger_share = 0.86  
passenger_revenue = revenue * passenger_share
```

- Bölme: / işaretini ile gerçekleştirilir.

```
revenue = 15823  
passenger_revenue = 13586  
passenger_share = passenger_revenue / revenue
```

- Tam Sayı Bölme: // işaretini ile gerçekleştirilir. Bu operatör, ondalık kısmını atar.

```
revenue = 15823
passenger_revenue = 13586
passenger_share = passenger_revenue // revenue
```

- Mod Alma: % işaretini ile gerçekleştirilir. Bu operatör, bir sayının diğer sayıya bölümünden kalanı verir.

```
revenue = 15823
passenger_revenue = 13586
passenger_share = revenue % passenger_revenue
```

- Üs Alma: ****** işaretini ile gerçekleştirilir. Bu operatör, bir sayının üssünü alır.

```
revenue = 15823 ** 2
```

Atama Operatörleri

- Eşittir (=)

```
cost_of_sales = 11752
```

- Topla ve ata (+=), Çıkar ve ata (-=)
- Çarp ve ata (*=), Böl ve ata (/=)
- Mod al ve ata (%=)
- Tam böl ve ata (//=)
- Üs al ve ata (**=)

```
fuel = 4545
cost_of_sales -= fuel
```

Karşılaştırma Operatörleri

True , False

- Eşittir (`==`) `1 == 1`
- Eşit değildir (`!=`) `1 != 1`
- Büyüktür (`>`) `2 > 1`
- Küçüktür (`<`) `2 < 1`
- Büyük veya eşit (`>=`) `1 >= 1`
- Küçük veya eşit (`<=`) `1 <= 1`

Mantıksal Operatörler

- Ve (`and`): Her iki ifadenin de doğru olması durumunda sonucu True döndürür. `True and False`
- Veya (`or`): En az bir ifadenin doğru olması durumunda sonucu True döndürür. `True or False`
- Değil (`not`): Bir ifadeninliğini (tersini) alır. `not True`

Kimlik Operatörleri

- Eşit mi (`is`): İki değişkenin aynı kimliğe sahip olup olmadığını kontrol eder.

```
iata_code_1 = ['ESB','SAW','IST']
iata_code_2 = ['ESB','SAW','IST']

print(iata_code_1 is iata_code_2)

print(id(iata_code_1))
print(id(iata_code_2))
```

- Eşit değil mi (`is not`): İki değişkenin veya nesnenin farklı kimliklere sahip olup olmadığını kontrol eder.

Üyelik Operatörleri

- İçeriyor mu (`in`): Bir elemanın varlığını kontrol eder. Eğer varsa sonuç True, aksi halde sonuç False olur.

```
iata_code = ['ESB','SAW','IST']

if 'IST' in iata_code:
    print('IST listede bulunmaktadır.')
else:
    print('IST listede bulunmamaktadır.')
```

- İçermiyor mu (`not in`): Bir elemanın yokluğunu kontrol eder. Eğer yoksa sonuç True, aksi halde sonuç False olur.

Koşullar

`if`, `elif` ve `else` ifadeleri, belirli koşullara dayalı olarak farklı işlemlerin gerçekleştirilmesini sağlayan bir kontrol yapısıdır. Bu yapının kullanımı, belirli bir şartın doğru veya yanlış olmasına bağlı olarak farklı kod bloklarının çalıştırılmasını sağlar.

```
if kosul_1:  
    # kosul_1 doğru ise burası çalışacak  
elif kosul_2:  
    # kosul_1 yanlış, kosul_2 doğru ise burası çalışacak  
elif kosul_3:  
    # kosul_1 ve kosul_2 yanlış, kosul_3 doğru ise burası çalışacak  
else:  
    # Hiçbir koşul doğru değilse burası çalışacak
```

```
# Kullanıcıdan yaş bilgisini alma
age = int(input("Lütfen yaşıınızı girin: "))
ticket_price = None

# Bilet fiyatını belirleme
if age < 18:
    ticket_price = 2500 # 18 yaş altı indirimli fiyat
elif age >= 18 and age < 65:
    ticket_price = 3000 # 18-65 yaş arası normal fiyat
elif age >= 65:
    ticket_price = 2750 # 65 yaş üstü indirimli fiyat
else:
    print("Geçersiz bir yaş girdiniz.")

# Kullanıcıya bilgi verme
print(f"\nBilet fiyatı: {ticket_price} TL")
```

Döngüler

Döngüler, belirli bir işlemi birçok kez tekrarlamamızı sağlayan yapılardır. Python'da iki tür döngü bulunur:
`for` döngüsü ve `while` döngüsü.

for Döngüsü

For döngüsü, belirli bir koleksiyonun (liste, demet, dize vb.) veya bir aralığın üzerinde gezinmeye olanak sağlar. Her döngü adımında döngünün içindeki işlem koleksiyonun her bir elemanı için tekrarlanır.

```
departures = {
    'Time': ['20:00', '20:00', '20:05', '20:05', '20:05'],
    'Flight': ['ME268', '6E18', 'EK122', 'QR246', 'TK1019'],
    'To': ['BEY', 'BOM', 'DXB', 'DOH', 'PRN'],
    'Airline': ['MEA (Retro Livery)', 'IndiGo', 'Emirates', 'Qatar Airways', 'Turkish Airlines'],
    'Aircraft': ['A320', 'B77W', 'B77W', 'B77W', 'B738']
}

for flight in departures['Flight']:
    print(f"Flight: {flight}")

for flight, time, aircraft in zip(departures['Flight'], departures['Time'], departures['Aircraft']):
    if aircraft == 'B77W':
        print(f"Flight: {flight}, Departure Time: {time}, Aircraft: {aircraft}")
```

`while` Döngüsü

`while` döngüsü, belirli bir koşul doğru olduğu sürece döngünün içindeki işlemleri tekrarlar. Koşul yanlış olduğunda döngü sona erer.

```
length = len(departures['Time'])
index = 0

while index < length:
    time = departures['Time'][index]
    flight = departures['Flight'][index]
    to = departures['To'][index]
    airline = departures['Airline'][index]
    aircraft = departures['Aircraft'][index]

    print(f"Time: {time}, Flight: {flight}, To: {to}, Airline: {airline}, Aircraft: {aircraft}")

    index += 1
```

Döngülerde Özel İfadeler - `break`

Bir döngüyü tamamlamadan çıkmak için kullanılır. Döngü içinde belirli bir koşul yerine geldiğinde, `break` ifadesiyle döngü aniden sona erer ve döngüden çıkarılır. Bu, döngüyü durdurarak geri kalan kod bloğunu atlamaya olanak tanır.

```
revenue_2022 = {
    'Airline':[
        'American Airlines',
        'Delta Airlines',
        'United Airlines',
        'Air France / KLM Group',
        'Lufthansa Group',
        'International Airlines Group (IAG)',
        'Southwest Airlines',
        'Turkish Airlines Group',
        'China Southern Group'
    ],
    'Annual Revenue':[49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
}

for airline, revenue in zip(revenue_2022['Airline'], revenue_2022['Annual Revenue']):
    if airline == 'Turkish Airlines Group':
        print(f'Airline: {airline} - Revenue: {revenue}')
        break
    else:
        print(airline)
```

Döngülerde Özel İfadeler - **continue**

Bir döngünün mevcut yinelemesini atlamak ve bir sonraki yinelemeye geçmek için kullanılır. Döngü içinde belirli bir koşul yerine geldiğinde, continue ifadesiyle döngünün geri kalanı atlanır ve bir sonraki yineleme başlar.

```
revenue_2022 = {
    'Airline':[
        'American Airlines',
        'Delta Airlines',
        'United Airlines',
        'Air France / KLM Group',
        'Lufthansa Group',
        'International Airlines Group (IAG)',
        'Southwest Airlines',
        'Turkish Airlines Group',
        'China Southern Group'
    ],
    'Annual Revenue':[49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
}

for airline, revenue in zip(revenue_2022['Airline'], revenue_2022['Annual Revenue']):
    if airline == 'Lufthansa Group':
        print("***Skipped...***")
        continue
    else:
        print(f'Airline: {airline} - Revenue: {revenue}')
```

Döngülerde Özel ifadeler - pass

Bir bloğun hiçbir şey yapmadığını belirtmek için kullanılır ve genellikle geçici olarak bir yer tutucu olarak kullanılır.

```
revenue_2022 = {
    'Airline':[
        'American Airlines',
        'Delta Airlines',
        'United Airlines',
        'Air France / KLM Group',
        'Lufthansa Group',
        'International Airlines Group (IAG)',
        'Southwest Airlines',
        'Turkish Airlines Group',
        'China Southern Group'
    ],
    'Annual Revenue':[49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
}

for airline in revenue_2022['Airline']:
    if airline == 'Turkish Airlines Group':
        pass
    else:
        print(airline)
```

List Comprehension

Daha okunabilir ve daha az kod.

```
revenue_2022 = {
    'Airline':[
        'American Airlines',
        'Delta Airlines',
        'United Airlines',
        'Air France / KLM Group',
        'Lufthansa Group',
        'International Airlines Group (IAG)',
        'Southwest Airlines',
        'Turkish Airlines Group',
        'China Southern Group'
    ],
    'Annual Revenue':[49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
}

# For
airlines_uppercase = []
for airline in revenue_2022['Airline']:
    airlines_uppercase.append(airline.upper())

print(airlines_uppercase)

# List Comprehension
airlines_uppercase = [airline.upper() for airline in revenue_2022['Airline']]
print(airlines_uppercase)
```

Fonksiyonlar

- Fonksiyon tanımlama
- Argümanlar (*args)
- Keyword Argümanları (**kwargs)
- return
- pass
- lambda

Fonksiyon Tanımlama

```
def fonksiyon_ismi():
    # yapılacaklar
```

```
def hava_durumu_kontrol(ruzgar_hizi, goruntu_mesafesi, yagis_durumu):  
  
    # km/saat  
    if ruzgar_hizi > 30:  
        return "Hava durumu uygun değil. Kalkış izni verilmıyor."  
  
    # m  
    if goruntu_mesafesi < 500:  
        return "Görüş mesafesi yetersiz. Kalkış izni verilmiyor."  
  
    if yagis_durumu:  
        return "Yağışlı hava. Kalkış izni verilmiyor."  
  
    return "Kalkış izni verildi."  
  
ruzgar_hizi = 20  
goruntu_mesafesi = 800  
yagis_durumu = False  
  
sonuc = hava_durumu_kontrol(ruzgar_hizi, goruntu_mesafesi, yagis_durumu)  
print(sonuc)
```

Argümanlar (*args)

Bir fonksiyona değişken sayıda pozisyonel argümanları iletmek için kullanılan bir yapıdır.

```
def filo(*bilgiler):
    print('***Filo***')
    for bilgi in bilgiler:
        print(bilgi)

filo('B787-9','20','2.8')
```

Keyword Argümanları (kwargs)**

Fonksiyona değişken sayıda keyword argümanlarını
geçirme olanağı sağlar.

```
def filo(**bilgiler):
    tip = bilgiler['tip']
    adet = bilgiler['adet']
    ortalama_yas = bilgiler['ortalama_yas']

    print(f'Tip: {tip}\nAdet: {adet}\nOrtalama Yaş: {ortalama_yas}')

filo(tip='B787-9', adet=20, ortalama_yas=2.8)
```

Modüller

```
# Kişisel fonksiyon
def calculate_median(input_list):
    sorted_list = sorted(input_list)
    n = len(sorted_list)

    if n % 2 == 1: # Tek mi?
        median = sorted_list[n // 2]
    else:
        median = (sorted_list[(n // 2) - 1] + sorted_list[n // 2]) / 2

    return median

revenue = [49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
result = calculate_median(revenue)
print("Median:", result)

# Modül
import numpy as np

revenue = [49,45.6,45,29.2,27.5,26.9,24,23.8,18.4,13]
median = np.median(revenue)

print("Median:", median)
```

Kendi modülünüzü yazabilirsiniz.



PyPI: <https://pypi.org/project/isyatirimhisse/4.0.0/>
GitHub: <https://github.com/urazakgul/isyatirimhisse>
Blog: <https://medium.com/@urazakgul/isyatirimhisse-4-0-0-5247d3e0c99b>

Hata Yakalama ve İşleme

- try, except, else, finally
- raise

try, except, else, finally

- **try** : Hata potansiyeli olan bir kod bloğunu tanımlamak için kullanılır. try bloğu içindeki kod çalıştırılır ve eğer bir hata meydana gelirse, hata yakalanır ve ilgili except bloğuna geçilir.
- **except** : Hata durumlarını ele almak için kullanılır. Eğer try bloğu içinde bir hata meydana gelirse, bu blok içindeki kod çalıştırılır.

- **else** : Bu blok, try bloğu içindeki kod hatasız bir şekilde çalıştırıldığında çalışır. Yani, herhangi bir hata olmadığında bu blok içindeki kod çalıştırılır.
- **finally** : Bu blok, her durumda çalıştırılması gereken kodu tanımlamak için kullanılır. Hata olsun veya olmasın, finally bloğu içindeki kod her zaman çalıştırılır.

```
try:  
    # Hata potansiyeli olan bir kod bloğu  
    # Burada bir hata meydana gelebilir  
    pass  
except HataTuru1:  
    # Hata durumunu ele almak için kod  
    pass  
except HataTuru2:  
    # Farklı bir hata durumunu ele almak için kod  
    pass  
else:  
    # Hata olmadığında çalıştırılacak kod  
    pass  
finally:  
    # Her durumda çalıştırılacak kod  
    pass
```

```
try:
    # Kullanıcıdan hisse senedi fiyatını ve net karı al
    hisse_senedi_fiyati = float(input("Hisse Senedi Fiyatı: "))
    net_kar = float(input("Net Kar: "))

    # P/E oranını hesapla
    pe_orani = hisse_senedi_fiyati / net_kar

except ValueError:
    # Kullanıcının sayısal bir değer girmemesi durumunda ValueError ele alınır
    print("Hata: Geçersiz bir sayı girişi yapıldı.")

except ZeroDivisionError:
    # Eğer net kar sıfırsa ZeroDivisionError ele alınır
    print("Hata: Net kar sıfır olamaz.")

except Exception as e:
    # Diğer tüm hatalar için genel bir hata mesajı
    print(f"Hata: {e}")

else:
    # Hata olmadığında çalışacak kod bloğu
    print(f"P/E Oranı: {pe_orani:.2f}")

finally:
    # Her durumda çalışacak kod bloğu
    print("Program sona erdi.")
```

raise

Bir hata ya da istisna durumunu (exception) programın akışı içinde bilerek ortaya çıkarmak için kullanılan bir yapıdır.

```
# Kullanıcıdan hisse senedi fiyatını ve net karı al
hisse_senedi_fiyati = float(input("Hisse Senedi Fiyatı: "))
net_kar = float(input("Net Kar: "))

# Eğer net kar sıfırsa özel bir hata yarat ve raise kullanarak fırlat
if net_kar == 0:
    raise ZeroDivisionError("Hata: Net kar sıfır olamaz.")

# P/E oranını hesapla
pe_orani = hisse_senedi_fiyati / net_kar

# Hata olmadığında çalışacak kod bloğu
print(f"P/E Oranı: {pe_orani:.2f}")

# Her durumda çalışacak kod bloğu
print("Program sona erdi.")
```

Dosya İşlemleri

- Dosya Açma
- Dosya Okuma
- Dosya Yazma
- Dosya Silme

Dosya Açma

```
with open("./1-python-programming-language/kap.txt", encoding='utf-8') as dosya:  
icerik = dosya.read()  
print(icerik)
```

Dosya Okuma

```
with open("./1-python-programming-language/kap.txt", "r", encoding='utf-8') as dosya:  
    icerik = dosya.read()  
    print(icerik)
```

Dosya Yazma

```
with open("./1-python-programming-language/kap.txt", "w", encoding='utf-8') as dosya:  
    dosya.write("09.11.2023 - Ekim 2023 Trafik Sonuçları\n" + icerik)
```

Dosya Silme

```
import os  
  
os.remove("./1-python-programming-language/kap_2.txt")
```



Veri Görselleştirme

We Fly to 127 Countries

EUROPE				AFRICA				MIDDLE EAST				FAR EAST			
43 COUNTRIES, 116 CITIES				39 COUNTRIES, 60 CITIES				13 COUNTRIES, 35 CITIES				22 COUNTRIES, 39 CITIES			
Germany	Russia	Belgium	Moldova	Egypt	Dem. Rep. Congo	Saudi Arabia	UAE	China	Bangladesh	Dhaka	Beijing	Indonesia	Guangzhou	Jakarta	Maldives
Munich	Moscow	Brussels	Chisinau	Cairo	Kinshasa	Jeddah	Dubai	Beijing	Dhaka	Indonesia	Guangzhou	Jakarta	Shanghai	Denpasar	Hong Kong
Frankfurt	Sochi	Bulgaria	Poland	Alexandria	Gabon	Madinah	Abu Dhabi	Guangzhou	Indonesia	Maldives	Shanghai	Jakarta	Shanghai	Denpasar	Hong Kong
Berlin	St. Petersburg	Sofia	Warsaw	Hurghada	Liberreville	Riyadh	Sharjah	Shanghai	Indonesia	Maldives	Taipei	Male	Xian	Singapore	Pakistan
Stuttgart	Kazan	Varna	Belarus	Sharmel-Sheikh	Mali	Dammam	Lebanon	Taipei	Indonesia	Maldives	Hong Kong	Singapore	Singapore	Sri Lanka	Karachi
Düsseldorf	Rostov	Romania	Minsk	Luxor	Bamako	Yanbu	Beirut	Hong Kong	Indonesia	Maldives	Male	Singapore	Colombo	Islamabad	Colombo
Cologne	Ekaterinburg	Bucharest	Slovakia	Algeria	Burkina Faso	El Qassim	Jordan	Xian	Indonesia	Maldives	Tashkent	Tashkent	Uzbekistan	Kyrgyzstan	Kyrgyzstan
Hamburg	Ufa	Ciuj	Kosice	Algiers	Ougadougou	Taif	Amman	Osh	Samarkand	Samarkand	Tashkent	Tashkent	Samarkand	Samarkand	Samarkand
Hanover	Astrakhan	Constanta		Oran	Cote D'Ivoire	Tehran	Qatar	Kazakhstan	Philippines	Philippines	Almaty	Almaty	Almaty	Almaty	Almaty
Nuremberg	Novosibirsk	Hungary		Constantine	Abidjan	Mashad	Doha	Manila	Philippines	Philippines	Manila	Manila	Manila	Manila	Tajikistan
Bremen	Stravropol	Budapest		Tlemcen	Chad	Shiraz	Bahrain	Astana	Tajikistan	Tajikistan	Japan	Japan	Dushanbe	Dushanbe	Osaka
Friedrichshafen	Voronezh	Czech Rep.		Batna	NDjamena	Tabriz	Bahrain	Tokyo	Khujand	Khujand	Kathmandu	Kathmandu	Kathmandu	Kathmandu	Kathmandu
Leipzig	Samara	Prague		South Africa	Benin	Isfahan	Oman	Osaka	Nepal	Nepal	Mumbai	Mumbai	New Delhi	New Delhi	Mumbai
Münster	Krasnodar	Croatia		Johannesburg	Cotonou	Kermanshah	Muscat	Thailand	Ulaanbaatar	Ulaanbaatar	Phuket	Phuket	Thailand	Thailand	Phuket
Baden-Baden	Ukraine	Zagreb		Cape Town	Guinea	Ahwanz	Syria	Bangkok	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Italy	Kiev	Dubrovnik		Durban	Conakry	Israel	Aleppo	Ashgabat	Samarkand	Samarkand	Malta	Malta	Almaty	Almaty	Almaty
Milan	Odessa	Portugal		Nigeria	Mozambique	Tel Aviv	Damascus	Malta	Samarkand	Samarkand	Manila	Manila	Manila	Manila	Manila
Rome	Ivano-Frankivsk	Lisbon		Lagos	Maputo	Iraq	Yemen	Malta	Samarkand	Samarkand	Osaka	Osaka	Osaka	Osaka	Osaka
Venice	Dnepropetrovsk	Porto		Abuja	Niger	Erbil	Aden	Phuket	Samarkand	Samarkand	Philippines	Philippines	Philippines	Philippines	Philippines
Bologna	Kharkiv	Bosnia		Kano	Niamey	Baghdad	Sanaa	Bangkok	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Naples	Kherson	Sarajevo		Port Harcourt	Eritrea	Basra	Sanaa	Ashgabat	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Catania	Zaporijya	Finland		Cameroun	Asmara	Sulaymaniyah	Saudi Arabia	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Turin	Lviv	Helsinki		Douala	Madagascar	Najaf	Abha	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Pisa	Simferopol	Rovaniemi		Yaounde	Antananarivo	Mosul	Iran	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Bari	Donetsk	Norway		Kenya	Mauritius	Kuwait	Urmia	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand	Samarkand
Genoa	Austria	Oslo		Mombasa	Port Louis	Kuwait									
France	Vienna	Albania		Nairobi	Seychelles										
Paris	Salzburg	Tirana		Tanzania	Seychelles										
Lyon	Graz	Ireland		Dar Es Selaam	Libya										
Nice	Northern Cyprus	Dublin		Kilmenjaro	Misurata										
Marseille	Lefkosa	Kosovo		Zanzibar	Benghazi										
Toulouse	Azerbaijan	Pristina		Tunisia	Sebha										
Bordeaux	Ganja	Macedonia		Tunis	Tripoli										
Strasbourg	Nakhichevan	Skopje		Djibouti	Sierra Leone										
UK	Netherlands	Serbia		Djibouti	Freetown										
London	Amsterdam	Belgrade		Ethiopia	Comoros										
Manchester	Rotterdam	Malta		Addis Ababa	Moroni										
Birmingham	Greece	Malta		Ghana	Gambia										
Spain	Athens	Slovenia		Accra	Banjul										
Barcelona	Thessaloniki	Ljubljana		Morocco	Zambia										
Madrid	Sweden	Montenegro		Casablanca	Lusaka										
Malaga	Stockholm	Podgorica		Marrakech	Congo										
Valencia	Gothenburg	Scotland		Rwanda	Point-Noire										
Bilbao	Denmark	Edinburgh		Kigali	Equatorial Guinea										
Switzerland	Copenhagen	Estonia		Somalia	Malabo										
Zurich	Aalborg	Tallinn		Mogadishu											
Geneva	Billund	Latvia		Sudan	Khartoum										
Basel	Georgia	Riga		Uganda	Angola										
	Tbilisi	Lithuania		Entebbe	Luanda										
	Batum	Vilnius		Mauritania	Egypt										
		Luxembourg		Nouakchott	Aswan										
				Senegal	South Sudan										
				Dakar	Juba										
					Sudan										
					Port Sudan										
Future Routes				Future Routes				Future Routes				Future Routes			
9 COUNTRIES, 19 CITIES				9 COUNTRIES, 19 CITIES				9 COUNTRIES, 19 CITIES				9 COUNTRIES, 19 CITIES			
AMERICAS				AMERICAS				AMERICAS				AMERICAS			
50 CITIES				DOMESTIC				DOMESTIC				DOMESTIC			
50 CITIES				DOMESTIC				DOMESTIC				DOMESTIC			

*As of 31.12.2020

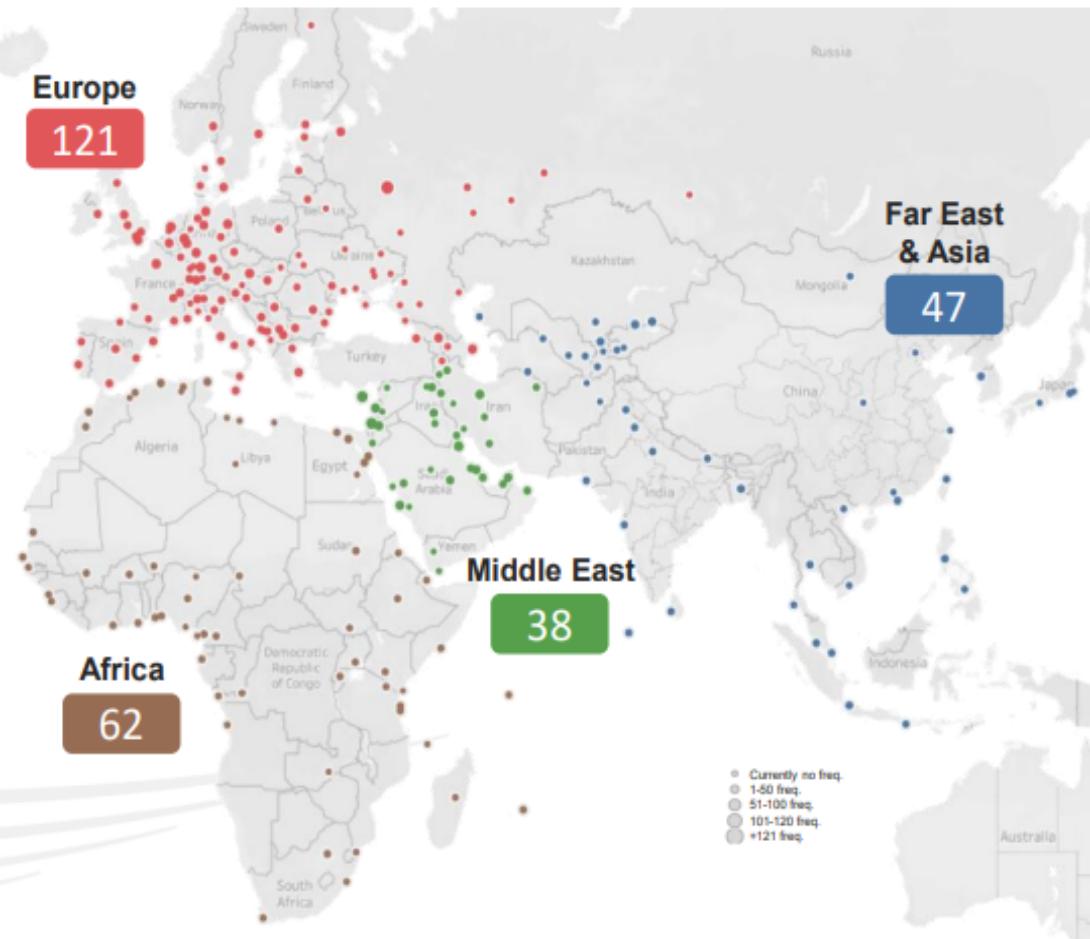
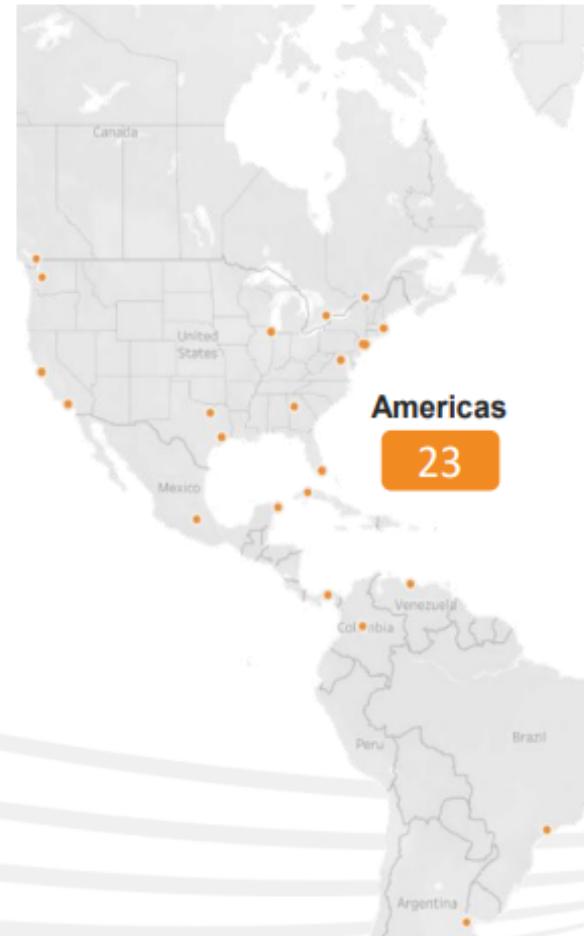
Turkish Airlines ranks #1 in the World by international destinations

Turkish Airlines flies to 291 international destinations in 128 countries¹

Provides 52.2 thousand connection options²

508 international destinations (including offline³)

Our network reaches more than 90% of the world's population, GDP and trade volume⁴



A STAR ALLIANCE MEMBER

¹As of 27.10.2023. ²Meaningful O&D's with detour factor <1.4. ³Including codeshare agreements. ⁴On a country basis. World Bank. Circle sizes represent the number of weekly frequencies.

The boxes show the number of destinations in the corresponding region.

21 New Routes in 2021-23

Americas

Seattle, Newark, Dallas, Vancouver

Europe

Palermo, Bergamo, Krakow, Tivat, Rize-Artvin

Middle East

Kirkuk, Urmia

Africa

Juba, Luanda, Lusaka

Far East & Asia

Cebu, Bukhara, Turkistan, Fergana, Urgench, Aktau, Turkmenbashi

Future Routes

Americas

Detroit, Denver, Orlando, Lima, Santiago, Rio de Janeiro

Europe

Bergen, Glasgow, Iasi, Katowice, Nantes, Newcastle, Timisoara, Bayburt, Yozgat

Middle East

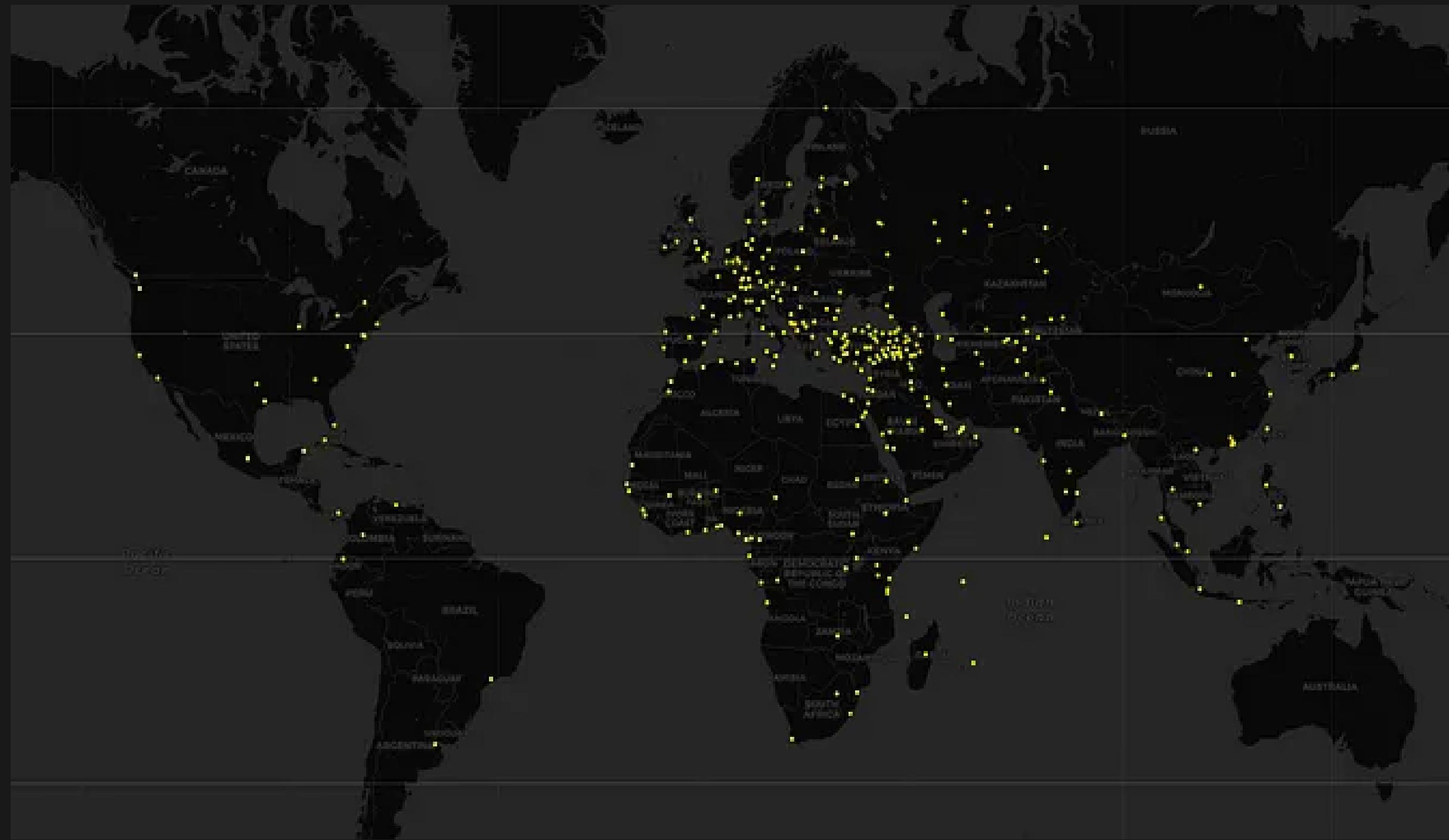
Abha, Salalah

Africa

Aswan, Brazzaville, Hargeisa, Lome, Monrovia, Port Sudan, Windhoek

Far East & Asia

Sydney, Melbourne, Phnom Penh, Atyrau, Osaka





Airways. @PythonMaps

This map shows the world's flight paths and airports. It maps 10,000 airports and 67,663 routes linking those airports.

Data source - <https://openflights.org/data.html>

Matplotlib ve **Seaborn**, Python programlama dili için bir veri görselleştirme kütüphanesidir.

```
pip install matplotlib  
pip install seaborn  
python  
import matplotlib, seaborn  
matplotlib.__version__  
seaborn.__version__
```

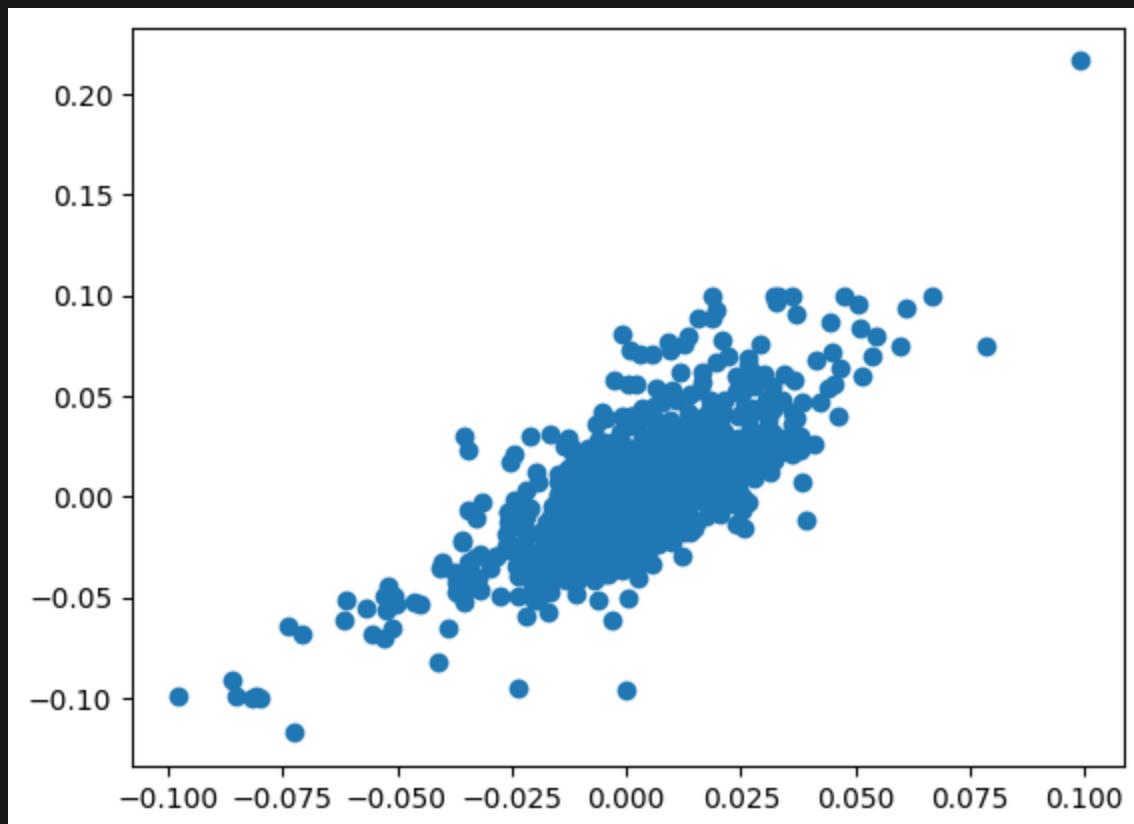
- Scatterplot
- Line Graph
- Histogram
- Bar Chart
- Heatmap

Scatterplot

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# sns.set_style("whitegrid")
# plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda/dataviz.xlsx', sheet_name='scatter')
df['Date'] = pd.to_datetime(df['Date'])
```

```
# Adim-1  
plt.scatter('XU100_R', 'THYAO_R', data=df)  
plt.show()
```



```
# Adım-2
plt.scatter('XU100_R', 'THYAO_R', data=df, color='red')
plt.show()

# Adım-3
plt.scatter('XU100_R', 'THYAO_R', data=df, color='red')
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.show()

# Adım-4
plt.scatter('XU100_R', 'THYAO_R', data=df, color='red')
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()
```

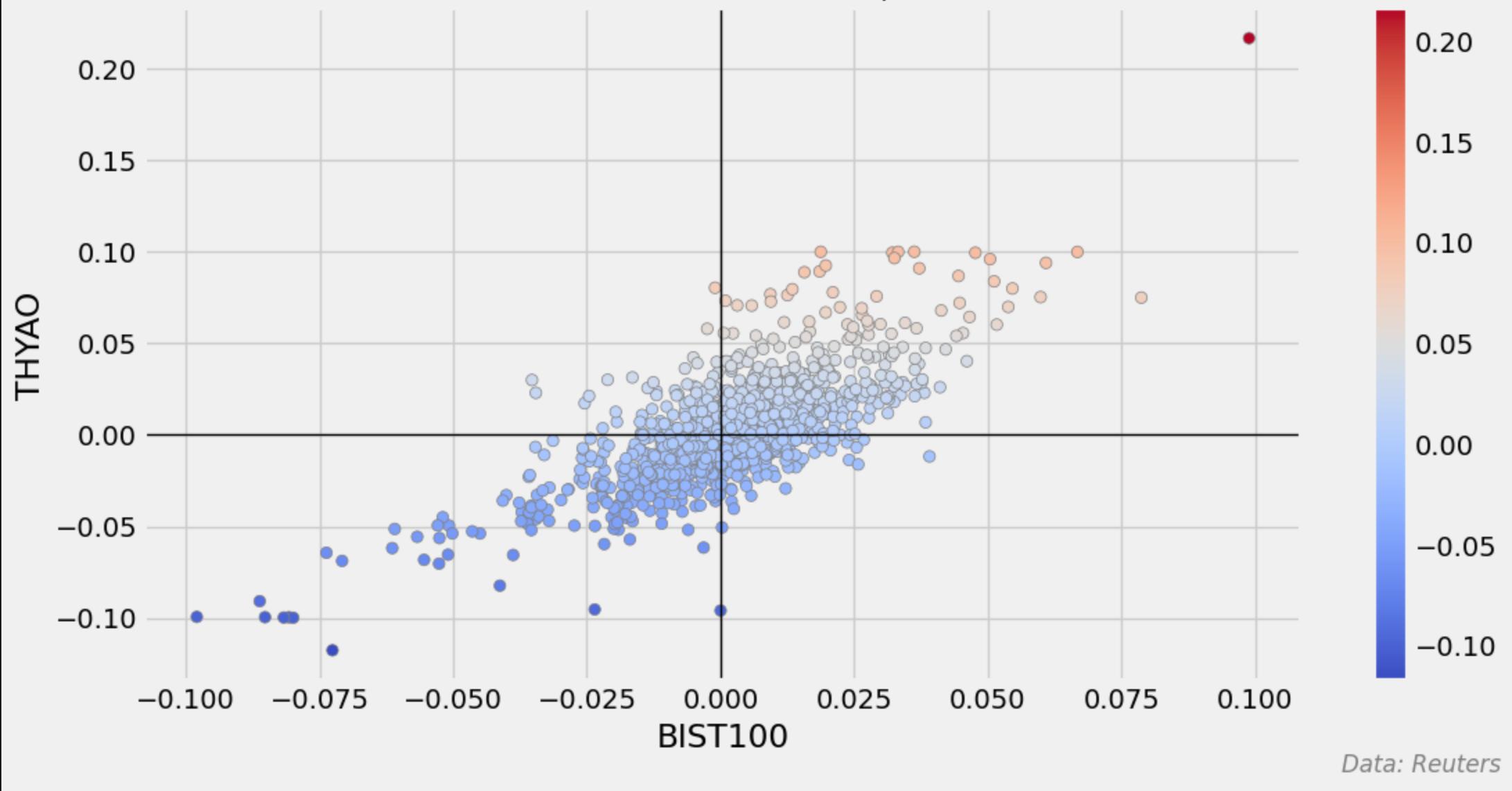
```
# Adım-5
plt.scatter('XU100_R', 'THYAO_R', data=df, color='red')
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.show()

# Adım-6
plt.scatter('XU100_R', 'THYAO_R', data=df, c='THYAO_R', cmap='coolwarm', edgecolors='gray')
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.axhline(0, color='gray', linewidth=0.5)
plt.axvline(0, color='gray', linewidth=0.5)
plt.colorbar(label='BIST100')
plt.show()
```

```
# Adım-7
plt.style.use('fivethirtyeight')

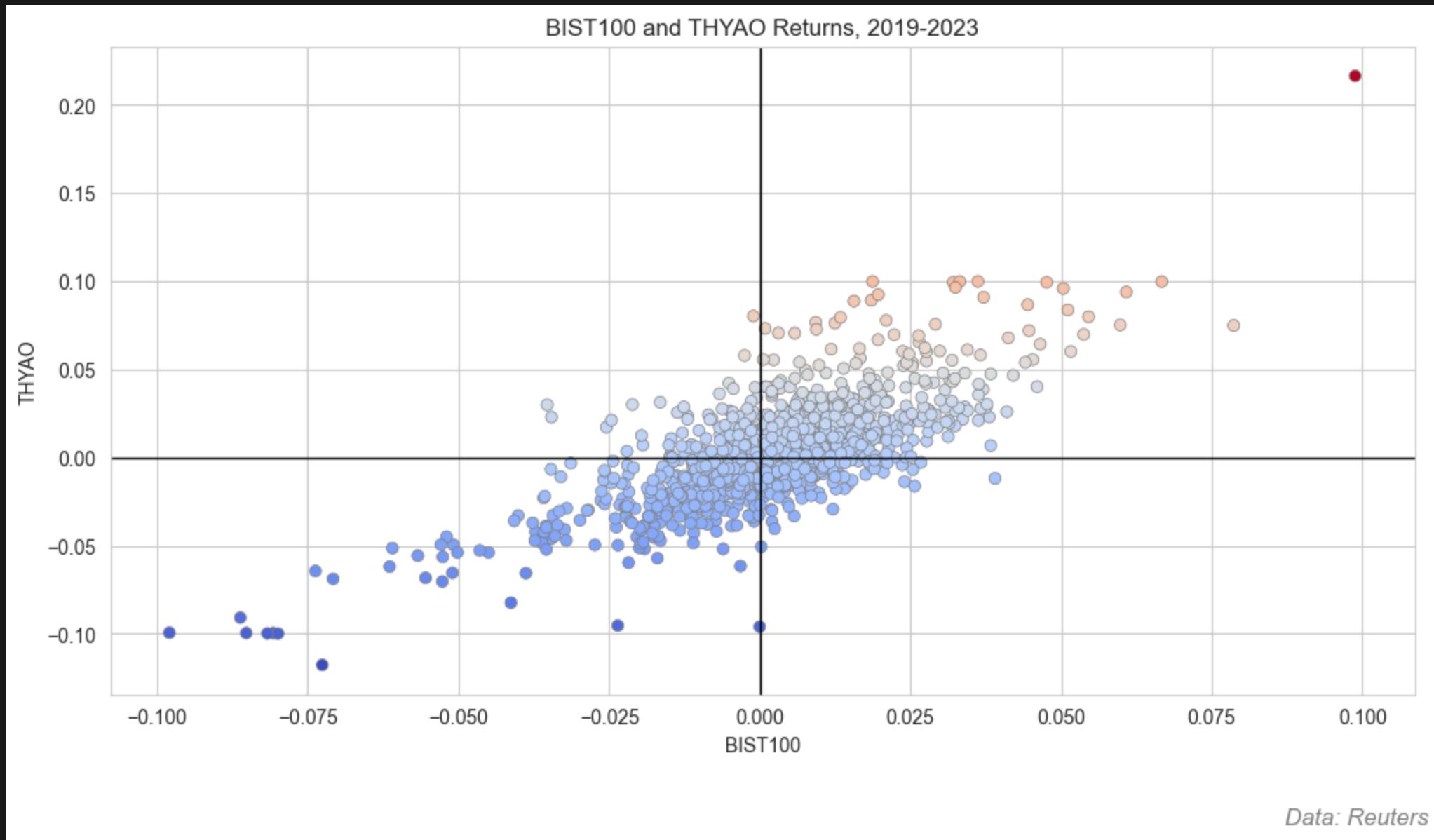
plt.figure(figsize=(12, 6))
plt.scatter('XU100_R', 'THYAO_R', data=df, c='THYAO_R', cmap='coolwarm', edgecolors='gray')
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.figtext(0.8, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic', color='gray')
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.colorbar(label='')
plt.show()
```

BIST100 and THYAO Returns, 2019-2023



```
sns.set_style("whitegrid")

plt.figure(figsize=(12, 6))
scatter = sns.scatterplot(
    x='XU100_R',
    y='THYAO_R',
    data=df,
    hue='THYAO_R',
    palette='coolwarm',
    edgecolor='gray'
)
plt.xlabel('BIST100')
plt.ylabel('THYAO')
plt.title('BIST100 and THYAO Returns, 2019-2023')
plt.text(
    0.9,
    -0.2,
    'Data: Reuters',
    fontsize=12,
    style='italic',
    color='gray',
    transform=plt.gca().transAxes # Metin koordinatlarını eksen koordinatları olarak yorumla
)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.legend().set_visible(False)
plt.show()
```



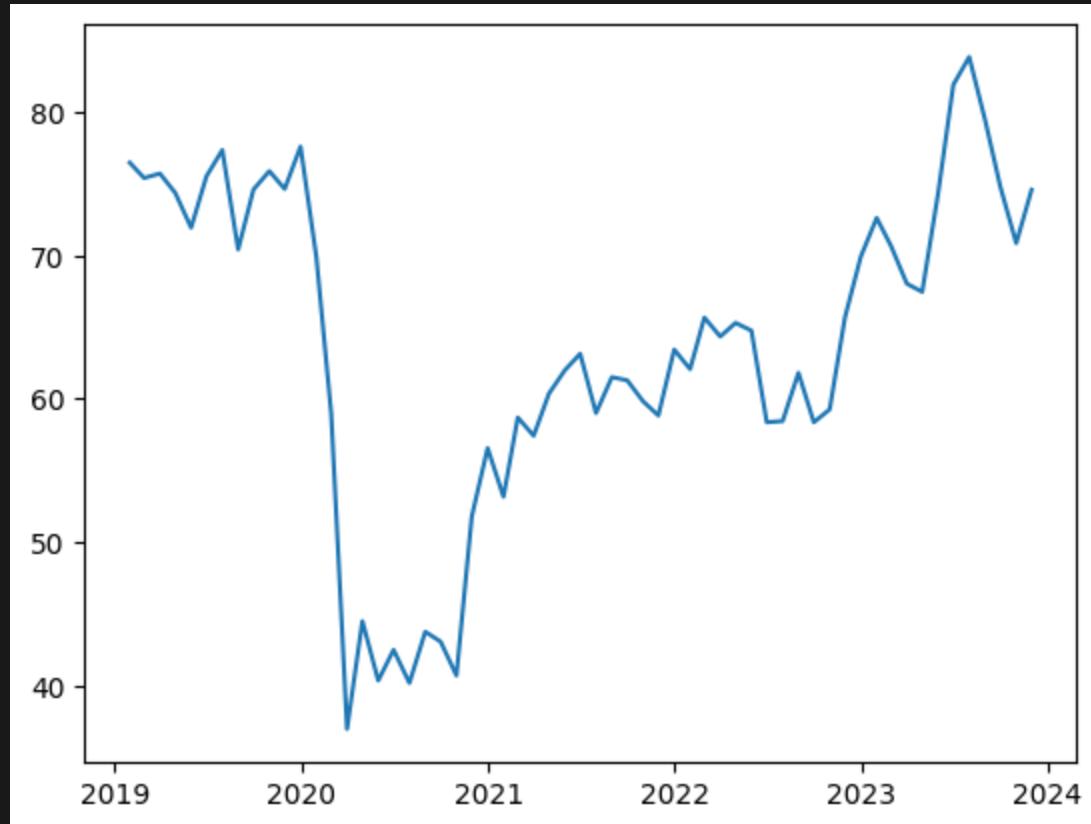
Data: Reuters

Line Graph

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# sns.set_style("whitegrid")
# plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda/dataviz.xlsx', sheet_name='line')
df['Date'] = pd.to_datetime(df['Date'])
```

```
# Adım-1  
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df)  
plt.show()
```



```
# Adım-2
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red')
plt.show()

# Adım-3
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging Markets Airlines Price Return Index')
plt.show()

# Adım-4
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging Markets Airlines Price Return Index')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()
```

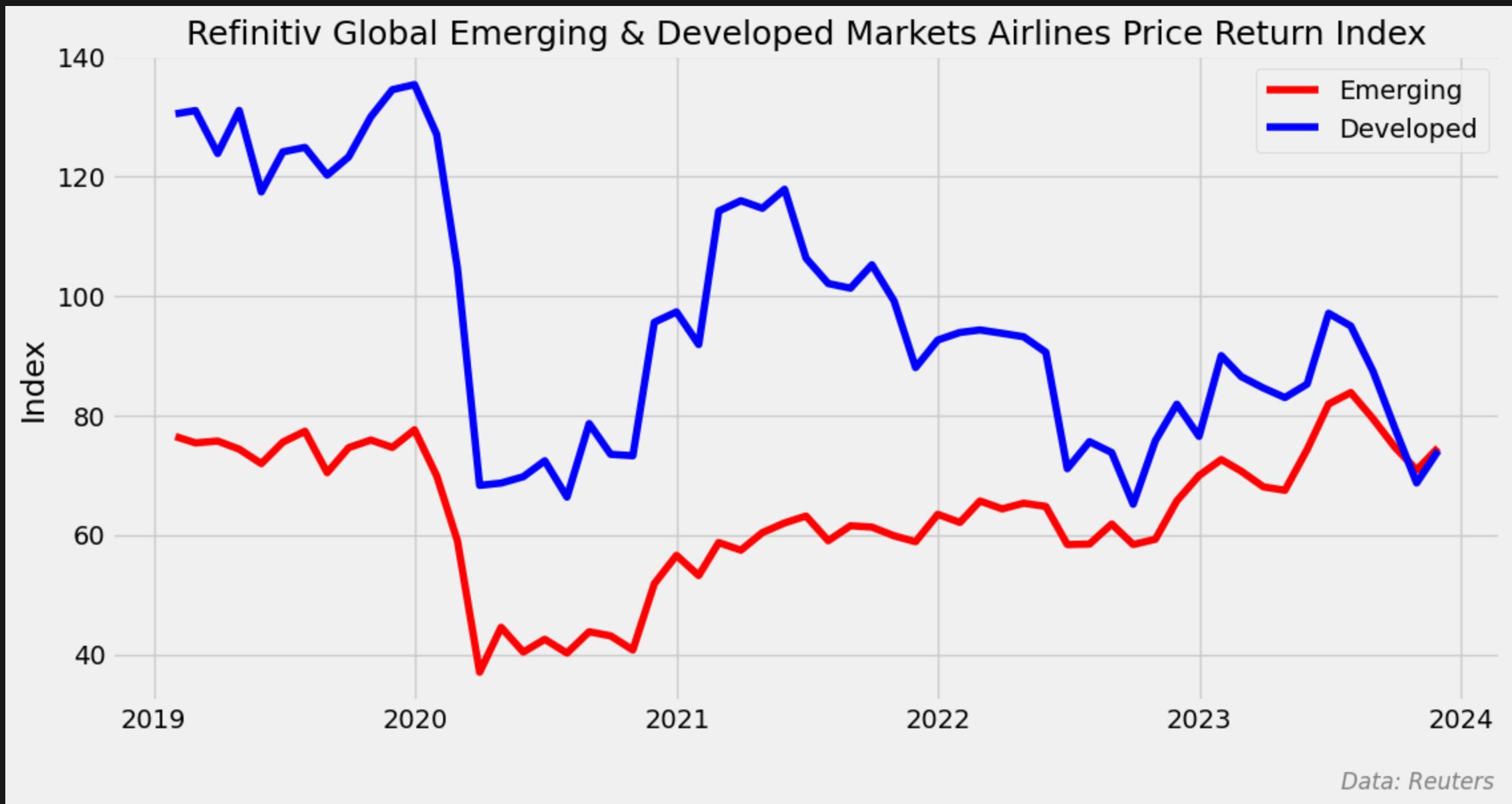
```
# Adım-5
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red')
plt.plot('Date', 'AIRLINE_DEV_INDEX', data=df, color='blue')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging & Developed Markets Airlines Price Return Index')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()

# Adım-6
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red')
plt.plot('Date', 'AIRLINE_DEV_INDEX', data=df, color='blue')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging & Developed Markets Airlines Price Return Index')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.legend()
plt.show()

# Adım-7
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red', label='Emerging')
plt.plot('Date', 'AIRLINE_DEV_INDEX', data=df, color='blue', label='Developed')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging & Developed Markets Airlines Price Return Index')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.legend()
plt.show()
```

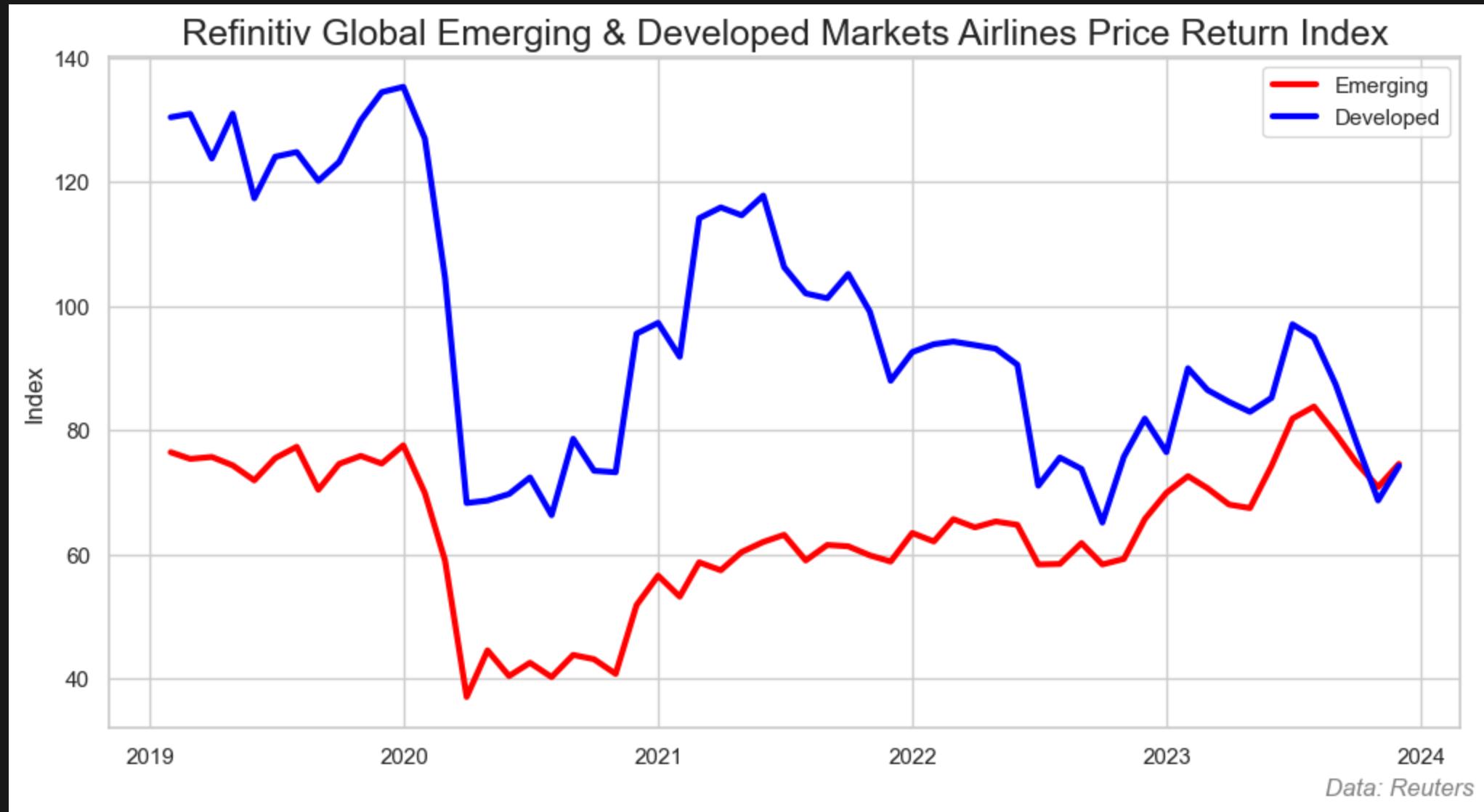
```
# Adm-8
plt.style.use('fivethirtyeight')

plt.figure(figsize=(12, 6))
plt.plot('Date', 'AIRLINE_EM_INDEX', data=df, color='red', label='Emerging')
plt.plot('Date', 'AIRLINE_DEV_INDEX', data=df, color='blue', label='Developed')
plt.ylabel('Index')
plt.title('Refinitiv Global Emerging & Developed Markets Airlines Price Return Index', fontsize=18)
plt.figtext(0.85, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic', color='gray')
plt.legend()
plt.show()
```



```
sns.set_style('whitegrid')

plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='AIRLINE_EM_INDEX', data=df, color='red', label='Emerging', linewidth=3)
sns.lineplot(x='Date', y='AIRLINE_DEV_INDEX', data=df, color='blue', label='Developed', linewidth=3)
plt.ylabel('Index')
plt.xlabel('')
plt.title('Refinitiv Global Emerging & Developed Markets Airlines Price Return Index', fontsize=18)
plt.text(
    0.9,
    -0.1,
    'Data: Reuters',
    fontsize=12,
    fontstyle='italic',
    color='gray',
    transform=plt.gca().transAxes
)
plt.legend()
plt.show()
```

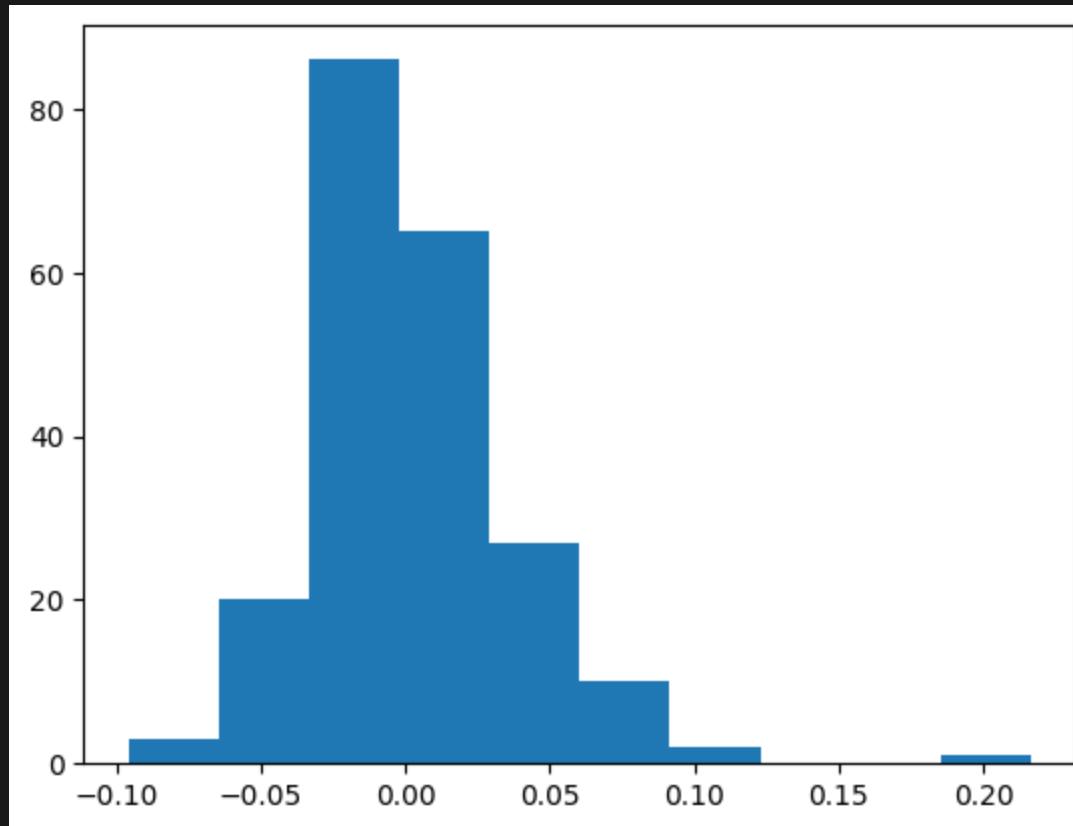


Histogram

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# sns.set_style("whitegrid")
# plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda/dataviz.xlsx', sheet_name='histogram')
df['Date'] = pd.to_datetime(df['Date'])
```

```
# Adım-1  
plt.hist('THYAO_R', data=df)  
plt.show()
```



```
# Adım-2
plt.hist('THYAO_R', data=df, color='red')
plt.show()

# Adım-3
plt.hist('THYAO_R', data=df, color='red')
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.show()

# Adım-4
plt.hist('THYAO_R', data=df, color='red')
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()
```

```
# Adım-5
plt.hist('THYAO_R', data=df, color='red', bins=20)
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()

# Adım-6
plt.hist('THYAO_R', data=df, color='red', bins='auto')
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.show()
```

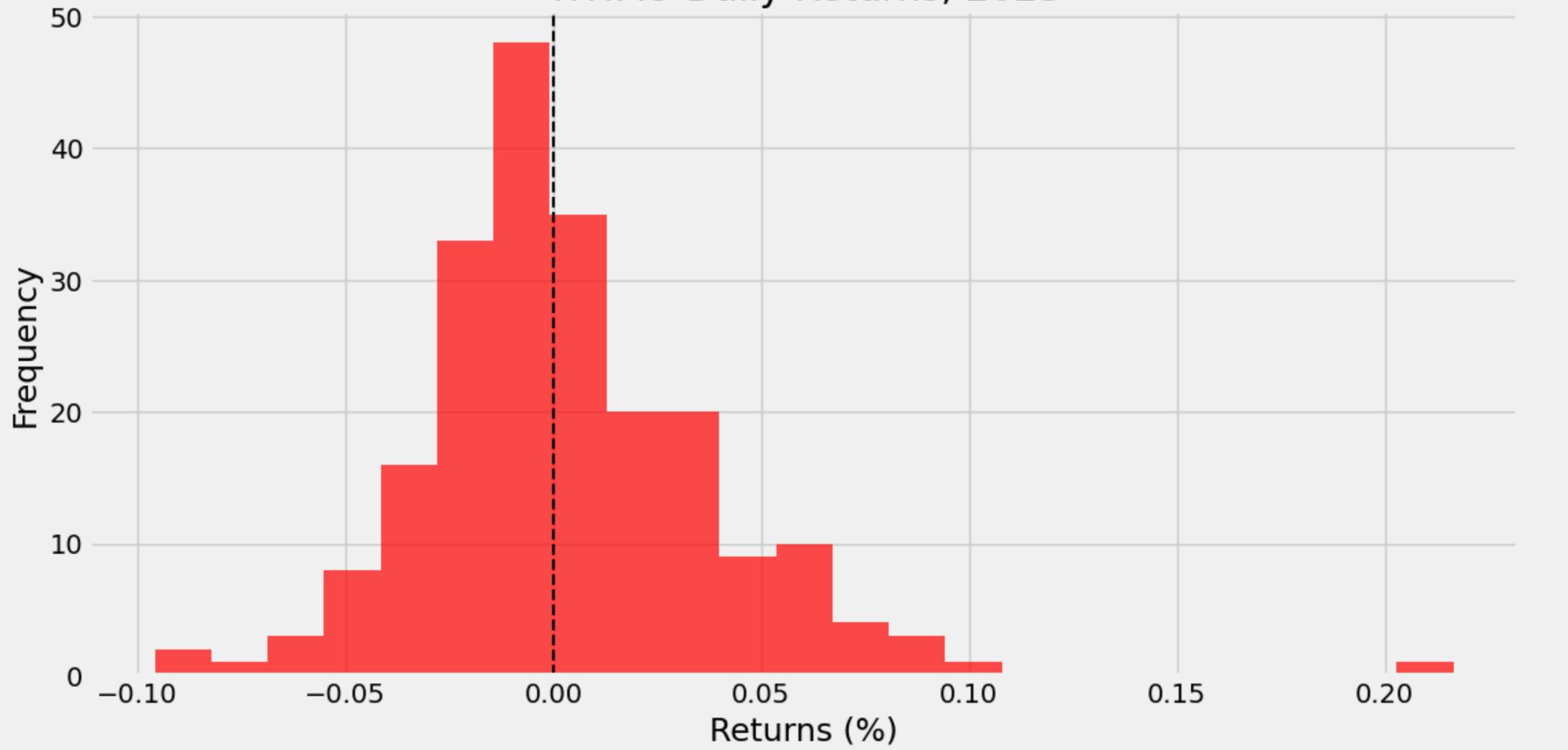
```
# Adım-7
plt.hist('THYAO_R', data=df, color='red', bins='auto')
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.axvline(x=0, color='black', linestyle='--')
plt.show()

# Adım-8
plt.hist('THYAO_R', data=df, color='red', bins='auto', alpha=0.7)
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(0.75, -0.04, 'Data: Reuters', fontsize=12, fontstyle='italic')
plt.axvline(x=0, color='black', linestyle='--')
plt.show()
```

```
plt.style.use('fivethirtyeight')

plt.figure(figsize=(12, 6))
plt.hist('THYAO_R', data=df, color='red', bins='auto', alpha=0.7)
plt.xlabel('Returns (%)')
plt.ylabel('Frequency')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(
    0.9, -0.04, 'Data: Reuters', fontsize=10, fontstyle='italic', color='gray'
)
plt.axvline(x=0, color='black', linestyle='--', linewidth=1.5)
plt.show()
```

THYAO Daily Returns, 2023

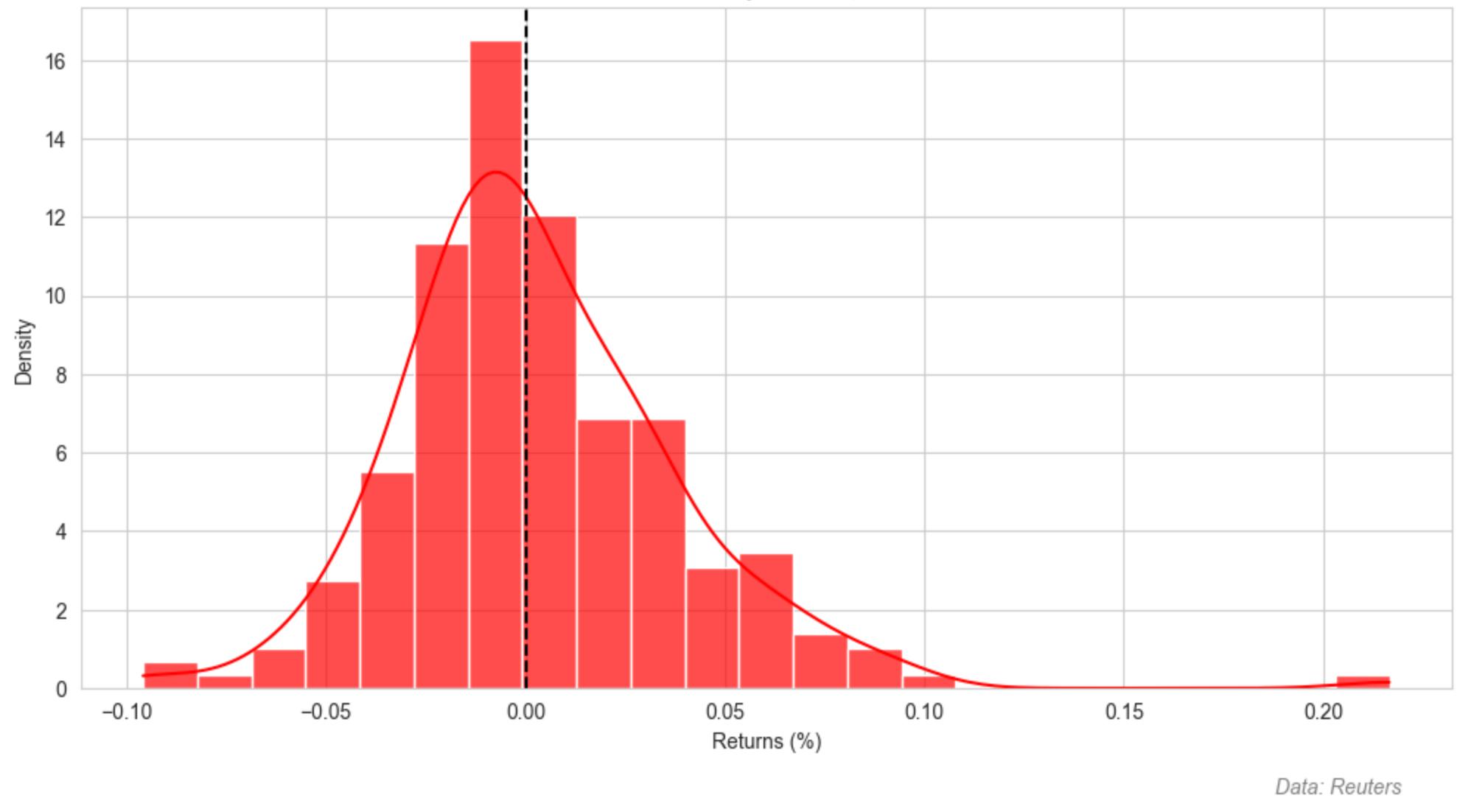


Data: Reuters

```
sns.set_style('whitegrid')

plt.figure(figsize=(12, 6))
sns.histplot(
    data=df['THYAO_R'],
    kde=True, # Kernel Density Estimation, veri dağılımını yumuşak bir şekilde tahmin eder
    color='red',
    bins='auto',
    stat='density',
    alpha=0.7
)
plt.xlabel('Returns (%)')
plt.ylabel('Density')
plt.title('THYAO Daily Returns, 2023')
plt.figtext(
    0.8,
    -0.01,
    'Data: Reuters',
    fontsize=10,
    fontstyle='italic',
    color='gray'
)
plt.axvline(x=0, color='black', linestyle='--', linewidth=1.5)
plt.show()
```

THYAO Daily Returns, 2023

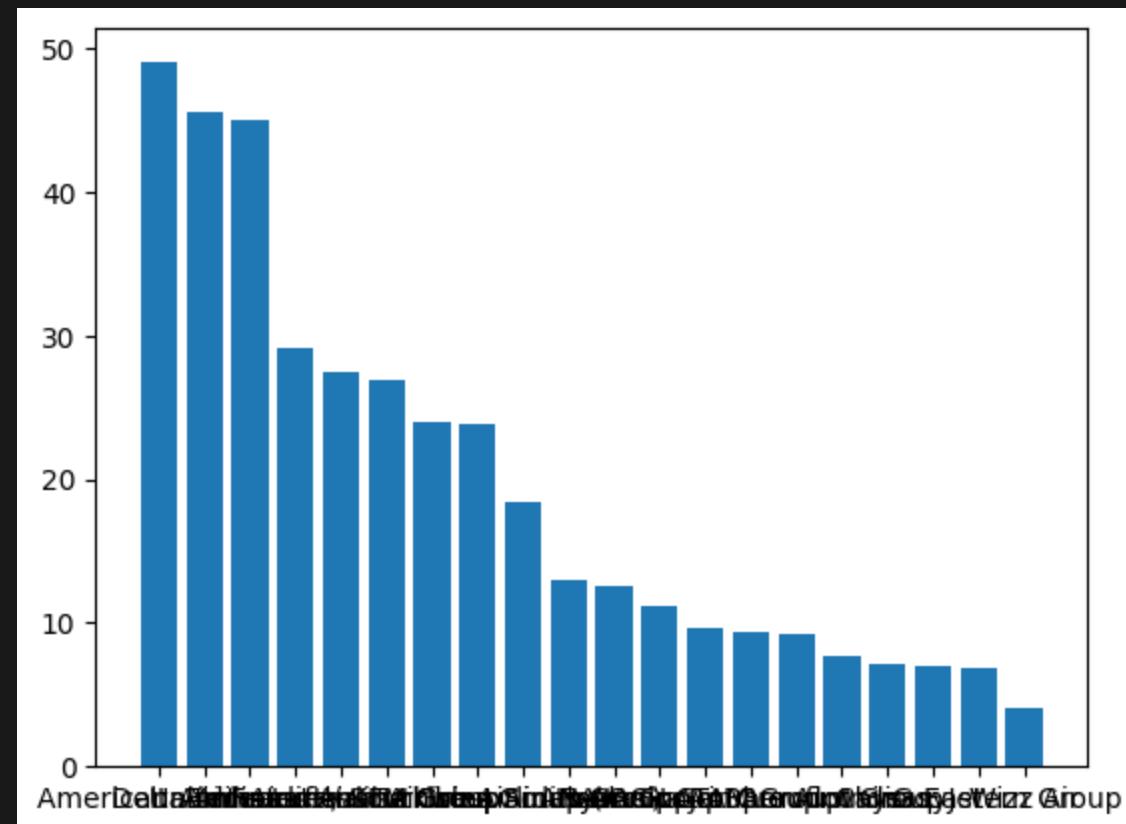


Bar Chart

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# sns.set_style("whitegrid")
# plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda/dataviz.xlsx', sheet_name='bar')
```

```
# Adım-1  
plt.bar('Airline', 'Annual Revenue (USD Billions)', data=df)  
plt.show()
```



```
# Adım-2
plt.barh('Airline', 'Annual Revenue (USD Billions)', data=df)
plt.show()

# Adım-3
df_sorted = df.sort_values(by='Annual Revenue (USD Billions)', ascending=True)
plt.barh('Airline', 'Annual Revenue (USD Billions)', data=df_sorted)
plt.show()

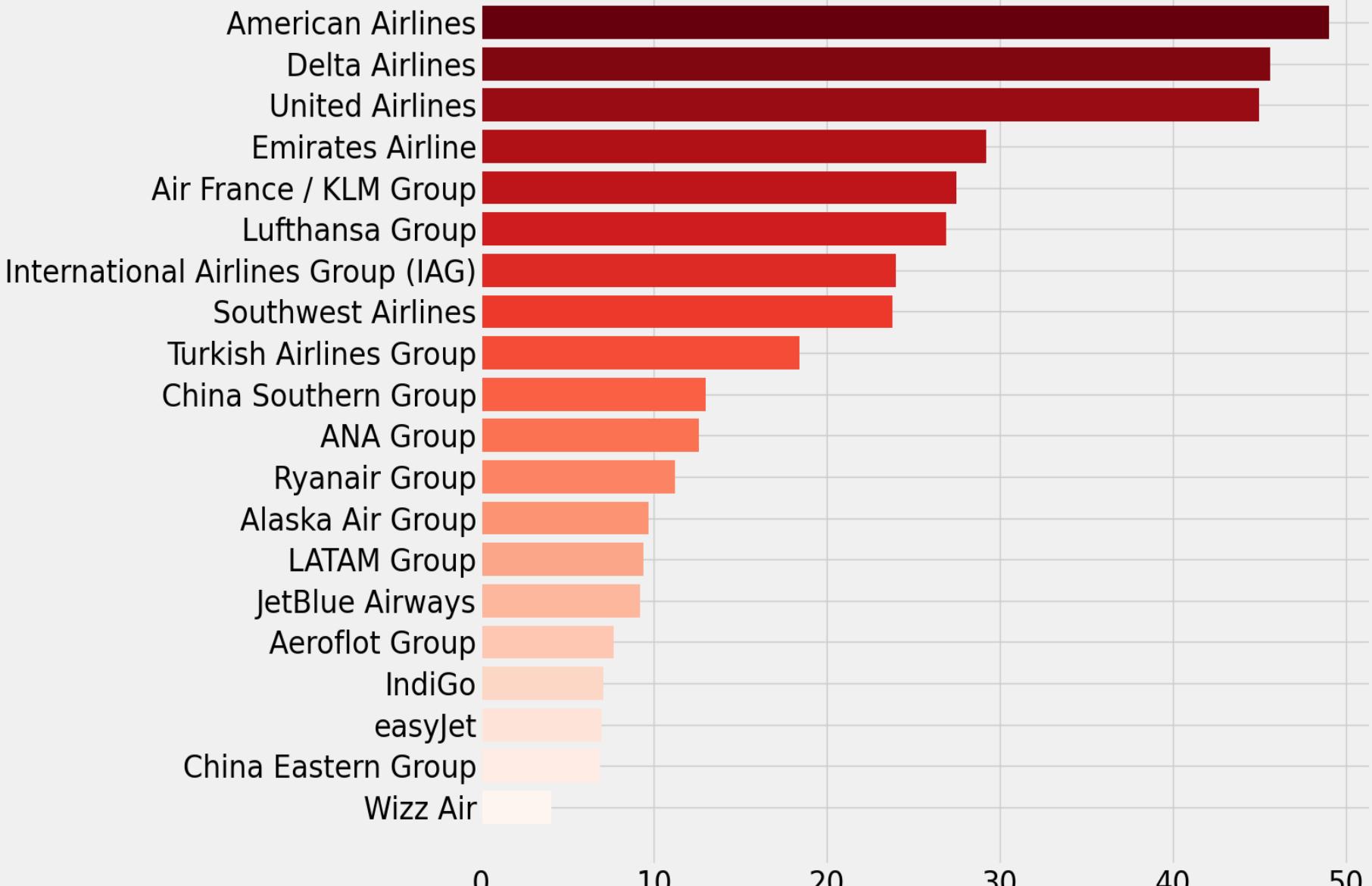
# Adım-4
df_sorted = df.sort_values(by='Annual Revenue (USD Billions)', ascending=True)
colors = plt.cm.Reds(np.linspace(0, 1, len(df_sorted)))
plt.barh('Airline', 'Annual Revenue (USD Billions)', data=df_sorted, color=colors)
plt.show()

# Adım-5
df_sorted = df.sort_values(by='Annual Revenue (USD Billions)', ascending=True)
colors = plt.cm.Reds(np.linspace(0, 1, len(df_sorted)))
plt.barh('Airline', 'Annual Revenue (USD Billions)', data=df_sorted, color=colors)
plt.title('Top 20 Airlines By Revenue in 2022')
plt.figtext(0.65, -0.04, 'Data: Visual Capitalist', fontsize=10, fontstyle='italic', color='gray')
plt.show()
```

```
# Adim-6
plt.style.use('fivethirtyeight')

plt.figure(figsize=(12, 12))
df_sorted = df.sort_values(by='Annual Revenue (USD Billions)', ascending=True)
colors = plt.cm.Reds(np.linspace(0, 1, len(df_sorted)))
plt.barh('Airline', 'Annual Revenue (USD Billions)', data=df_sorted, color=colors)
plt.title('Top 20 Airlines By Revenue in 2022 (USD Billions)', fontsize=32)
plt.figtext(0.8, -0.02, 'Data: Visual Capitalist', fontsize=16, fontstyle='italic', color='gray')
plt.tick_params(axis='both', labelsize=24)
plt.show()
```

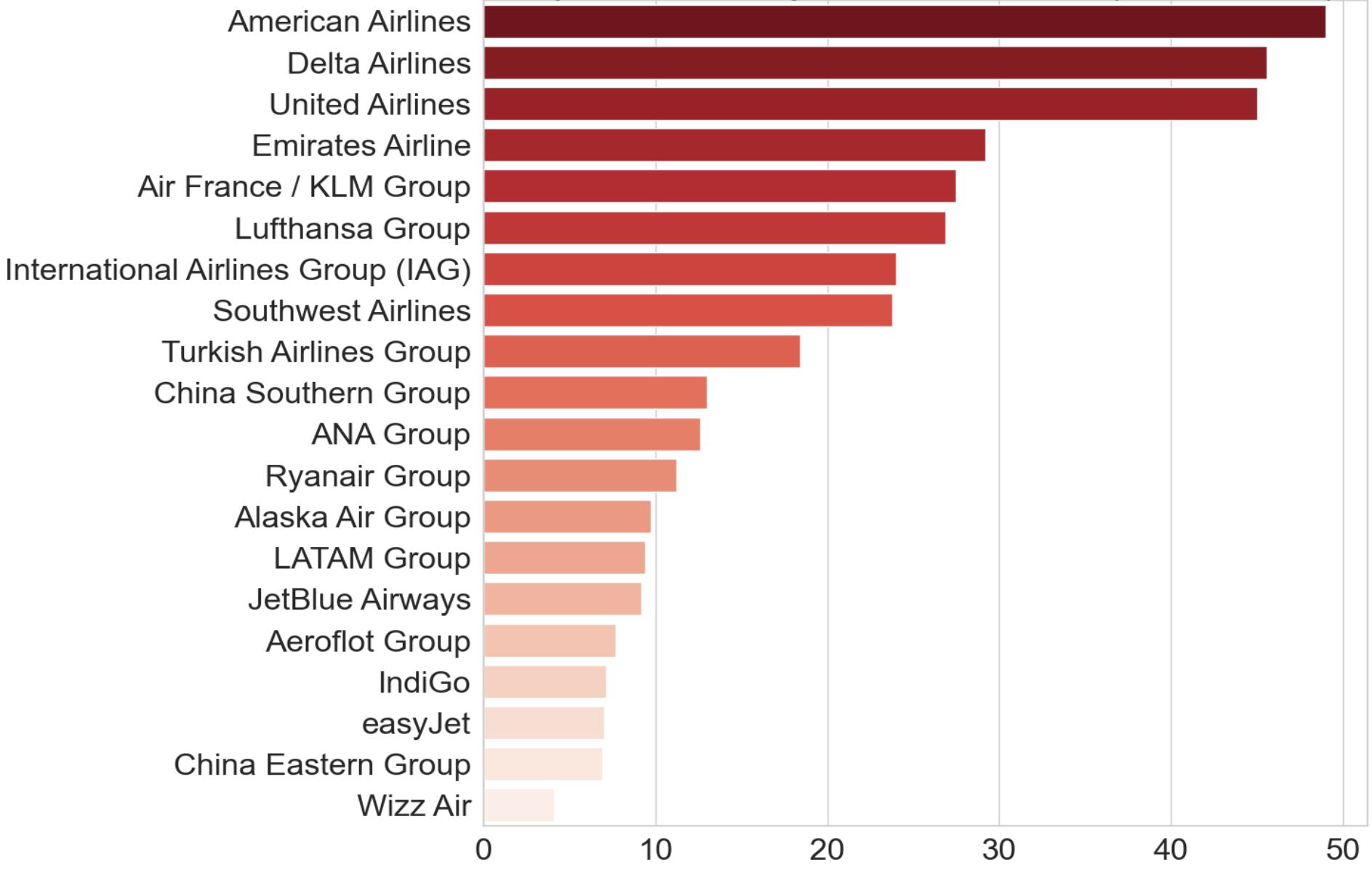
Top 20 Airlines By Revenue in 2022 (USD Billions)



```
sns.set(style="whitegrid", palette="Reds_r")

df_sorted = df.sort_values(by='Annual Revenue (USD Billions)', ascending=False)
colors = sns.color_palette("Reds", n_colors=len(df_sorted))[:-1]
plt.figure(figsize=(12, 12))
sns.barplot(
    x='Annual Revenue (USD Billions)',
    y='Airline',
    data=df_sorted,
    palette=colors,
    hue='Airline',
    dodge=False, # Barlar ayrılacak mı?
    legend=False
)
plt.title('Top 20 Airlines By Revenue in 2022 (USD Billions)', fontsize=28)
plt.figtext(
    0.7,
    -0.02,
    'Data: Visual Capitalist',
    fontsize=16,
    fontstyle='italic',
    color='gray'
)
plt.tick_params(axis='both', labelsize=24)
plt.xlabel('')
plt.ylabel('')
plt.show()
```

Top 20 Airlines By Revenue in 2022 (USD Billions)

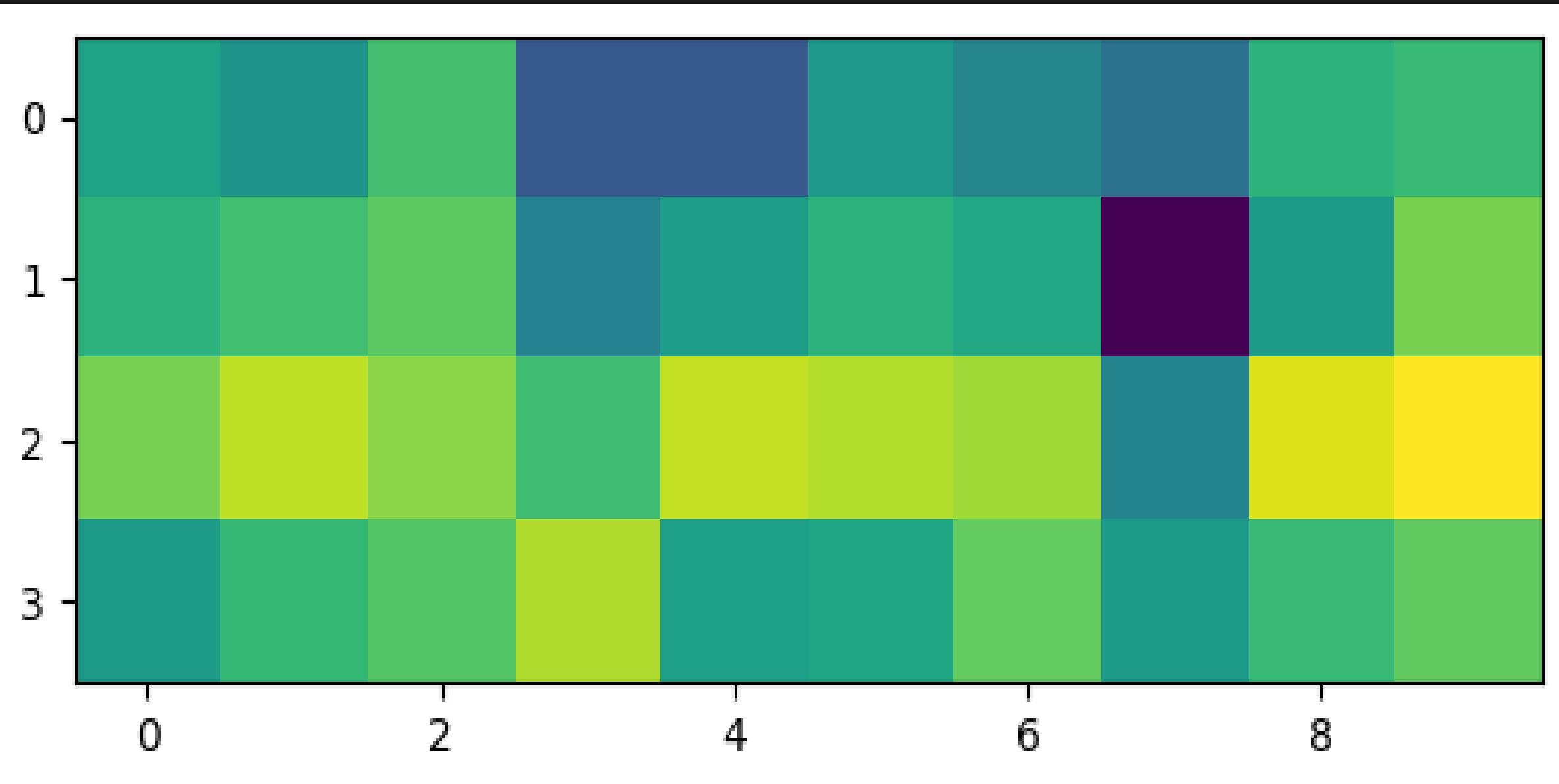


Heatmap

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
# sns.set_style("whitegrid")
# plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda/dataviz.xlsx', sheet_name='heatmap')
df['Date'] = pd.to_datetime(df['Date'])
df['Quarter'] = df['Date'].dt.month
df['Year'] = df['Date'].dt.year
heatmap_data = df.pivot_table(
    values='Net Profit Margin',
    index='Quarter',
    columns='Year',
    aggfunc='mean'
)
```

```
# Adim-1  
plt.imshow(heatmap_data)  
plt.show()
```



```
# Adim-2
plt.imshow(heatmap_data)

plt.yticks(np.arange(len(heatmap_data.index)), [str(year) for year in heatmap_data.index])
plt.xticks(np.arange(len(heatmap_data.columns)), [str(month) for month in heatmap_data.columns])

plt.show()

# Adim-3
plt.figure(figsize=(12, 6))
plt.imshow(heatmap_data)
plt.colorbar(label='Net Profit Margin')
plt.xlabel('Year')
plt.ylabel('Quarter')
plt.title('Net Profit Margin')

plt.yticks(np.arange(len(heatmap_data.index)), [str(year) for year in heatmap_data.index])
plt.xticks(np.arange(len(heatmap_data.columns)), [str(month) for month in heatmap_data.columns])

plt.show()
```

```
# Adım-4
plt.figure(figsize=(12, 6))
heatmap = plt.imshow(heatmap_data, cmap='Reds')
cbar = plt.colorbar(heatmap, label='Net Profit Margin', orientation='horizontal', shrink=0.8)
cbar.set_label('Net Profit Margin', rotation=0, labelpad=15)

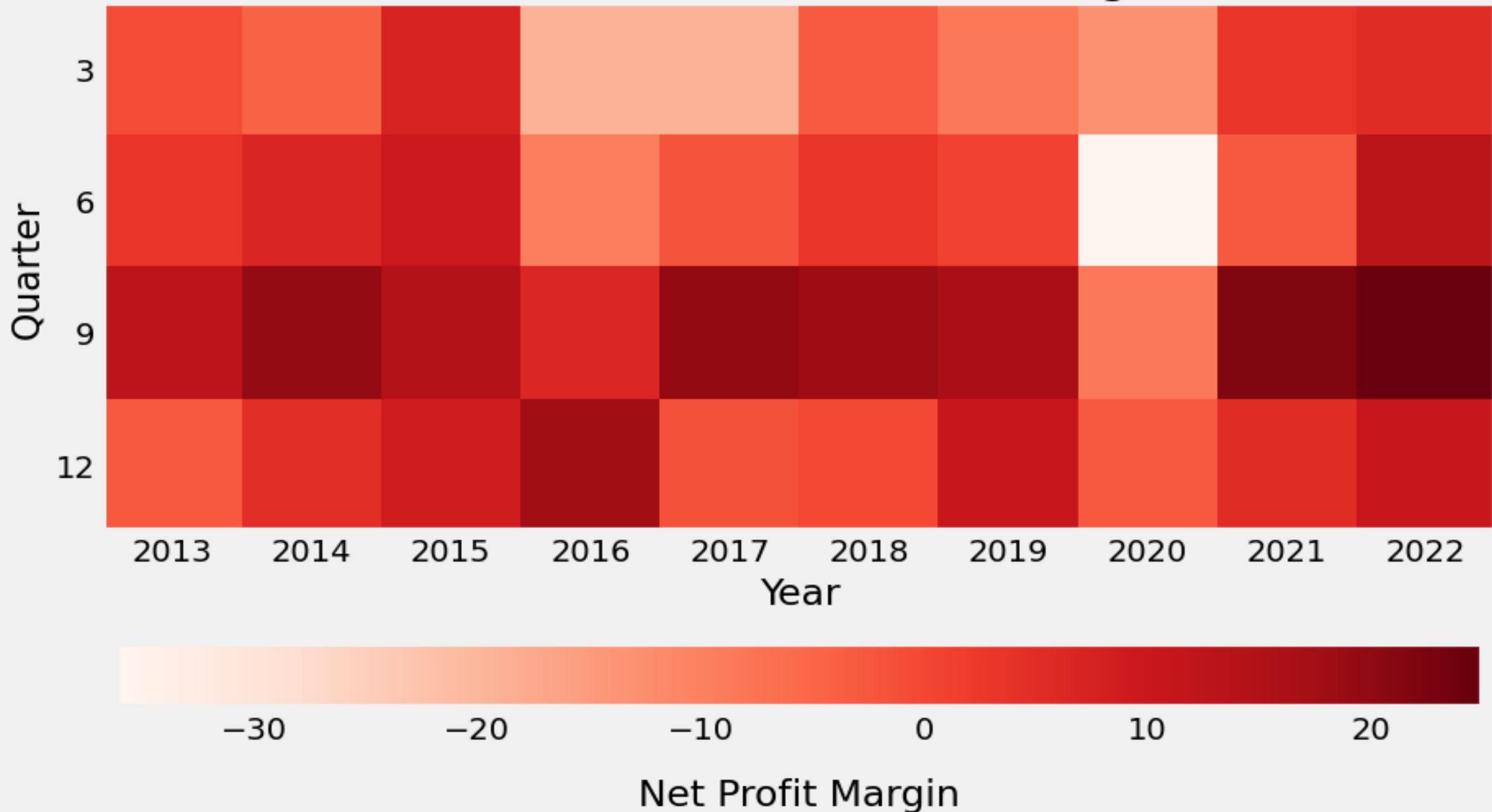
plt.xlabel('Year')
plt.ylabel('Quarter')
plt.title('Turkish Airlines Net Profit Margin')

plt.yticks(np.arange(len(heatmap_data.index)), [str(year) for year in heatmap_data.index])
plt.xticks(np.arange(len(heatmap_data.columns)), [str(month) for month in heatmap_data.columns])

plt.grid(False)

plt.show()
```

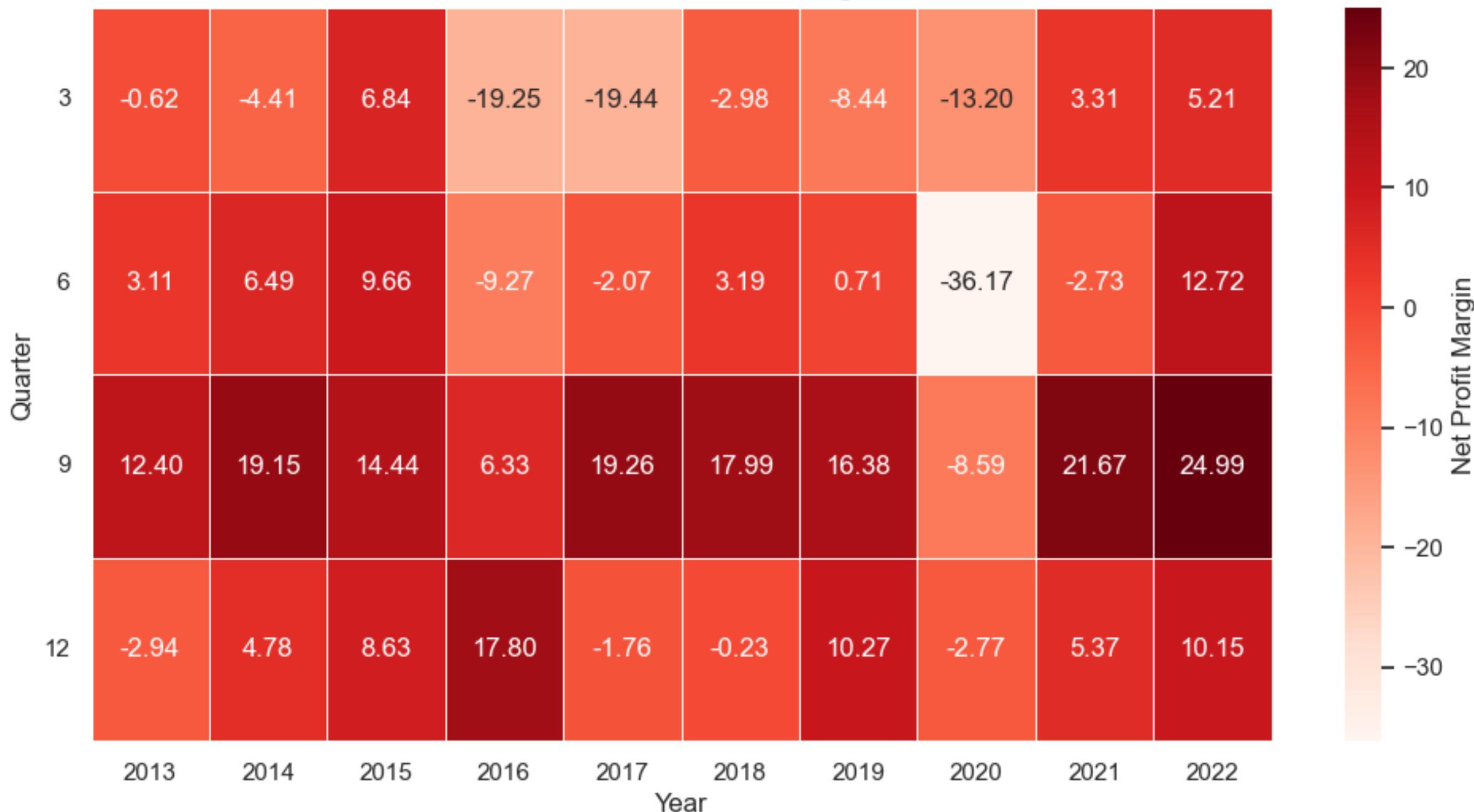
Turkish Airlines Net Profit Margin



```
sns.set(style="whitegrid")

plt.figure(figsize=(12, 6))
sns.heatmap(
    heatmap_data,
    cmap='Reds',
    annot=True, # Sayısal değerler eklenecek mi?
    fmt=".2f",
    linewidths=.5, # Hücreler arasındaki çizgi kalınlığı.
    cbar_kws={'label': 'Net Profit Margin'} # Colorbar keyword arguments
)
plt.xlabel('Year')
plt.ylabel('Quarter')
plt.title('Turkish Airlines Net Profit Margin')
plt.yticks(rotation=0)
plt.xticks(rotation=0)
plt.show()
```

Turkish Airlines Net Profit Margin





Veri Manipülasyonu ve Analizii

Pandas, Python programlama dilinin üzerine inşa edilmiş hızlı, güçlü, esnek ve kullanımı kolay bir açık kaynak veri manipülasyonu ve analizi aracıdır.

```
pip install pandas
python
import pandas as pd
pd.__version__
```

Kütüphanenin ve Veri Setinin İçe Aktarılması

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')

df = pd.read_excel('./2-dataviz-eda asn_data.xlsx')
```

Çalışılacak Sütunların Seçilmesi

- iloc, integer location anlamına gelir ve DataFrame veya Series üzerinde konum tabanlı indeksleme yapılmasına olanak tanır.
- loc, label location anlamına gelir ve DataFrame veya Series üzerinde etiket tabanlı indeksleme yapmak için kullanılan bir indeksleme yöntemidir.

```
# acc. date, type, operator, fat., location, dmg
df = df[[
    'acc. date','type','operator','fat.','location','dmg'
]]

# veya

df = df.iloc[:, [0, 1, 3, 4, 5, 7]]
```

Sütunların Yeniden İsimlendirilmesi

```
df = df.rename(  
    columns={  
        'acc. date':'date',  
        'fat.':'fatalities',  
        'dmg':'damage'  
    }  
)
```

Sütunlara Göre NaN Sayısının Bulunması

```
# Sayı  
df.isna().sum()  
  
# Yüzde  
(df.isna().sum() / len(df)) * 100
```

Sütunların Veri Tipleri

```
df.dtypes
```

Date Kolonunun Kendi Veri Formatına Dönüştürülmesi

```
def fix_date_format(date_str):
    try:
        return pd.to_datetime(date_str, format='%d-%b-%Y')
    except ValueError:
        return None
df['date'] = df['date'].apply(fix_date_format)
for i in range(1, len(df)):
    if pd.isnull(df.loc[i, 'date']):
        prev_date = df.loc[i - 1, 'date']
        next_date = df.loc[i + 1, 'date'] if i + 1 < len(df) else None
        if prev_date is not None:
            df.loc[i, 'date'] = prev_date
        elif next_date is not None:
            df.loc[i, 'date'] = next_date
```

Airbus ve Boeing'in Filtrelenmesi ve İndeksin Sıfırlanması

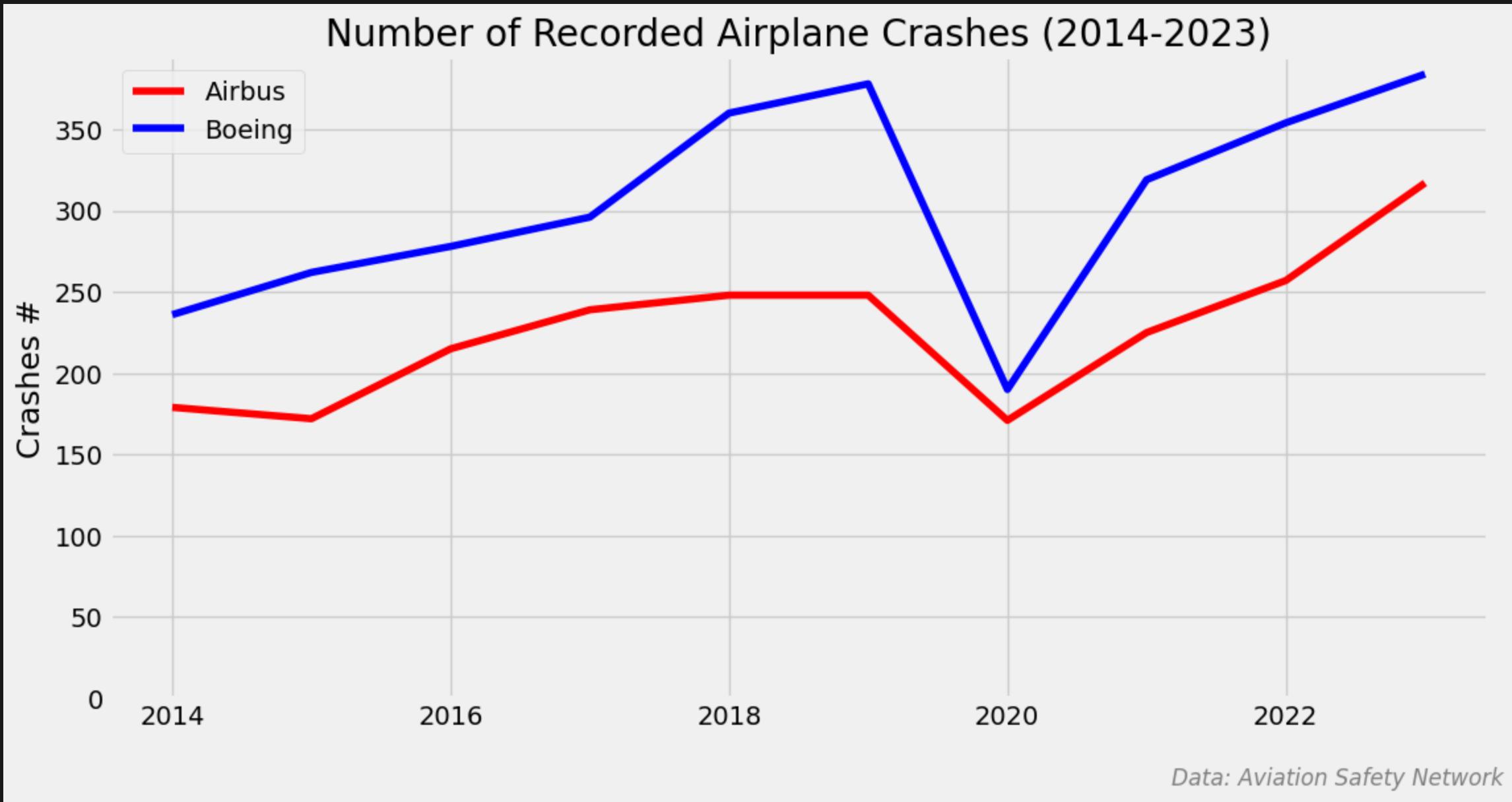
```
airbus_boeing_df = df[df['type'].str.contains('Airbus|Boeing', na=False)]
airbus_boeing_df = airbus_boeing_df.reset_index(drop=True)
airbus_boeing_df.loc[airbus_boeing_df['type'].str.contains('Airbus'), 'type'] = 'Airbus'
airbus_boeing_df.loc[airbus_boeing_df['type'].str.contains('Boeing'), 'type'] = 'Boeing'
```

Yıllara Göre Airbus&Boeing Tipli *Kaydedilen* Uçak Kazası Sayısı

(2014-2023)

```
airbus_boeing_df.loc[:, 'year'] = airbus_boeing_df['date'].dt.year
crashes_by_year = airbus_boeing_df.groupby(['year', 'type']).size().unstack(fill_value=0).reset_index()
```

```
plt.figure(figsize=(12, 6))
plt.plot(crashes_by_year['year'], crashes_by_year['Airbus'], color='red', label='Airbus')
plt.plot(crashes_by_year['year'], crashes_by_year['Boeing'], color='blue', label='Boeing')
plt.ylabel('Crashes #')
plt.title('Number of Recorded Airplane Crashes (2014-2023)')
plt.legend()
plt.ylim(0)
plt.figtext(
    0.75,
    -0.04,
    'Data: Aviation Safety Network',
    ha='left',
    fontsize=12,
    fontstyle='italic',
    color='gray'
)
plt.show()
```

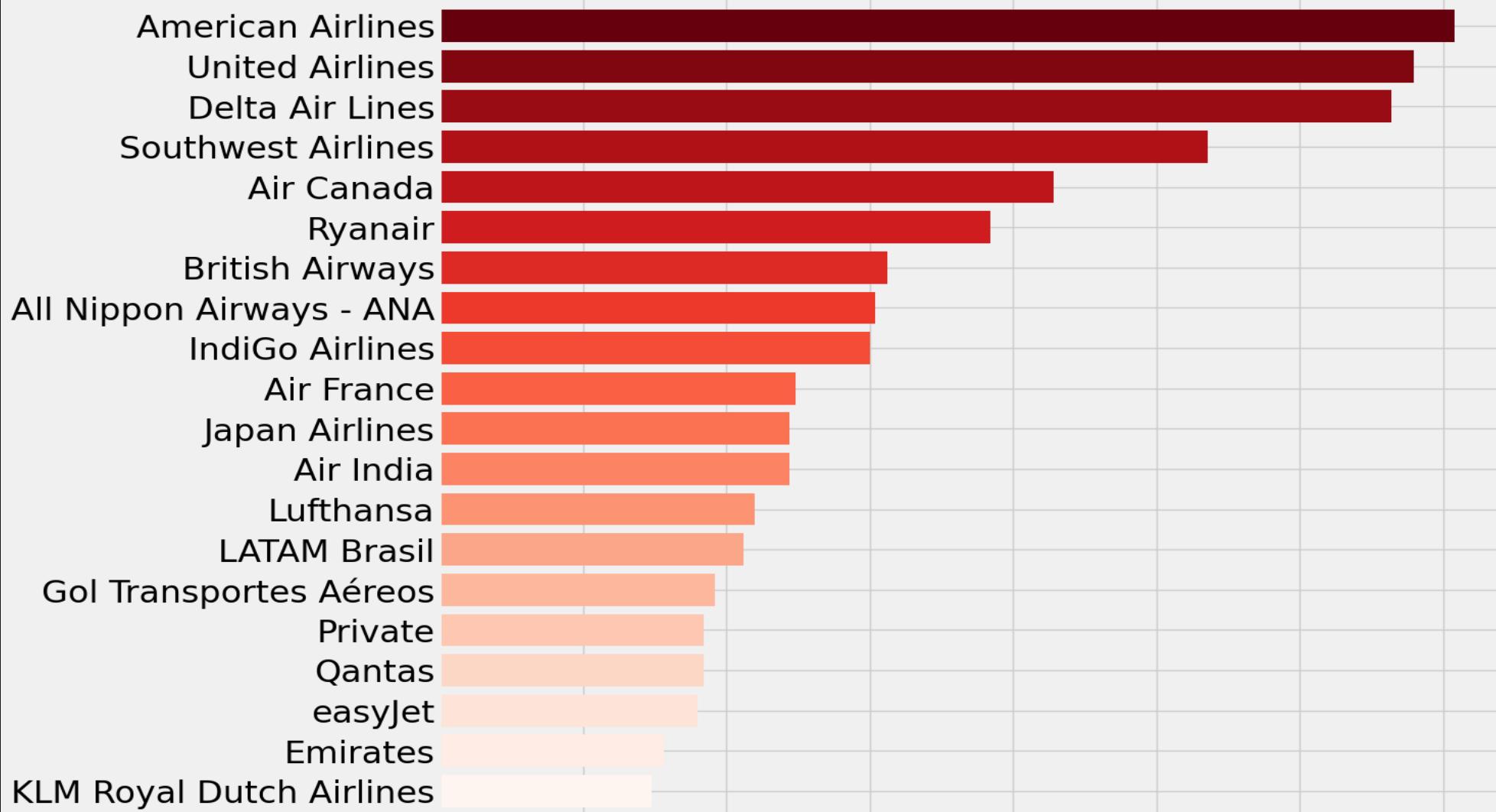


Operatörler Göre Top 20

```
count_by_operator = airbus_boeing_df[ 'operator' ].value_counts().reset_index(name='crashes #')
```

```
plt.figure(figsize=(12, 12))
count_by_operator_sorted = count_by_operator.sort_values(by='crashes #', ascending=True).tail(20)
colors = plt.cm.Reds(np.linspace(0, 1, len(count_by_operator_sorted)))
plt.barh('operator', 'crashes #', data=count_by_operator_sorted, color=colors)
plt.title('Top 20 Operators (limited to Airbus and Boeing) by Number of Crashes\n(2014-2023)', fontsize=28)
plt.figtext(
    0.7,
    -0.02,
    'Data: Aviation Safety Network',
    ha='left',
    fontsize=16,
    fontstyle='italic',
    color='gray'
)
plt.tick_params(axis='both', which='major', labelsize=24)
plt.show()
```

Top 20 Operators (limited to Airbus and Boeing) by Number of Crashes (2014-2023)



Yıllara Göre Airbus&Boeing Tipli Uçak Kazalarında Ölenlerin Sayısı

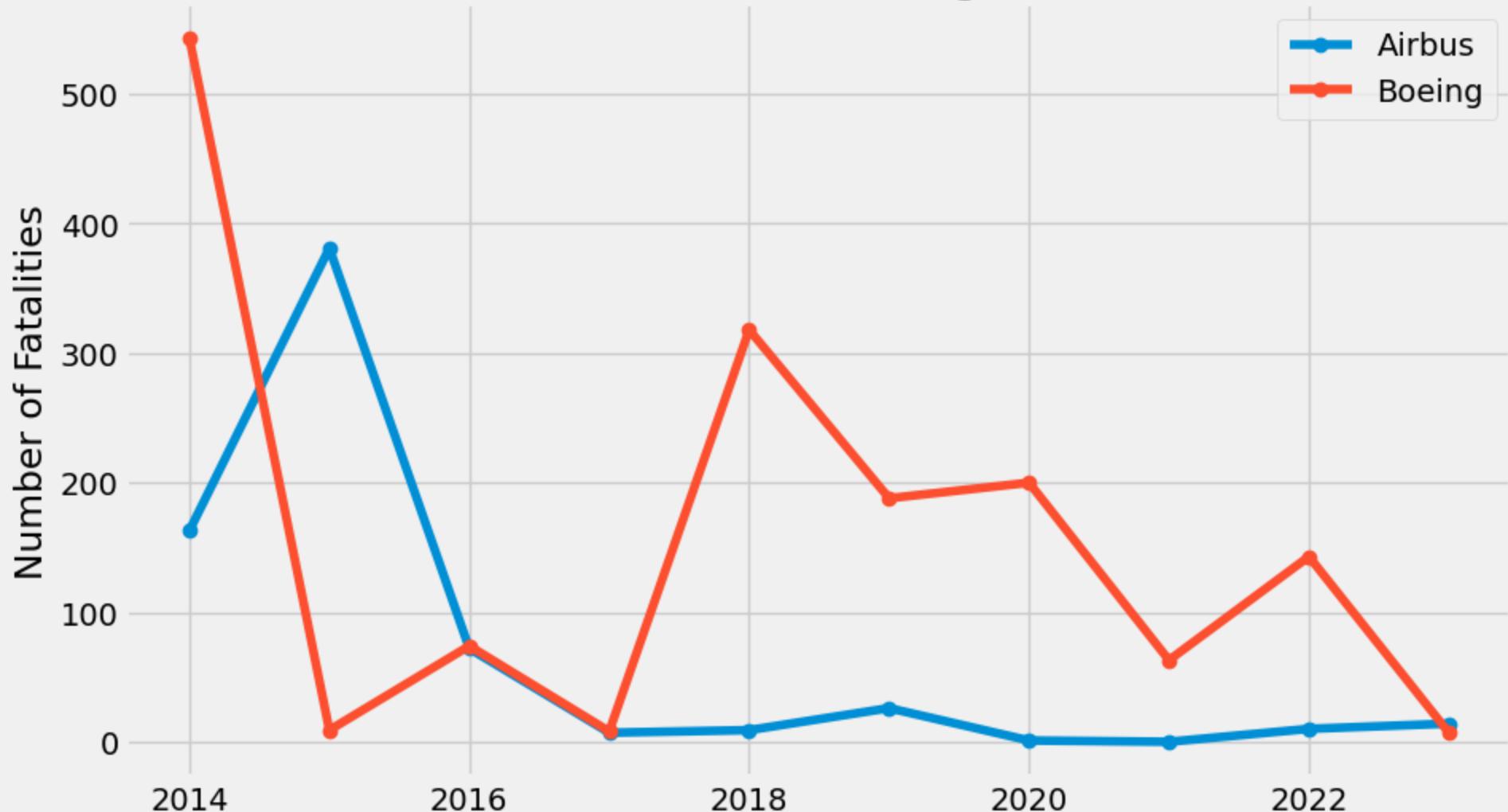
```
airbus_boeing_df['fatalities'] = pd.to_numeric(airbus_boeing_df['fatalities'], errors='coerce')
fatalities_by_year_and_type = airbus_boeing_df.groupby(['year', 'type'])['fatalities'].sum().reset_index()
```

```
plt.figure(figsize=(10, 6))

for aircraft_type in fatalities_by_year_and_type['type'].unique():
    subset = fatalities_by_year_and_type[fatalities_by_year_and_type['type'] == aircraft_type]
    plt.plot(subset['year'], subset['fatalities'], label=aircraft_type, marker='o')

plt.title('Number of Fatalities in Airplane Accidents by Years\n(Limited to Airbus and Boeing, Last 10 Years)')
plt.ylabel('Number of Fatalities')
plt.legend()
plt.figtext(
    0.7,
    -0.03,
    'Data: Aviation Safety Network',
    ha='left',
    fontsize=12,
    fontstyle='italic',
    color='gray'
)
plt.show()
```

Number of Fatalities in Airplane Accidents by Years (Limited to Airbus and Boeing, Last 10 Years)



Data: Aviation Safety Network



Doğrusal Regresyon ile Tahminleme

in the model, such as elasticities or the effects of monetary policy, and usually attempts to measure the validity of the theory against the behavior of observable data. Once suitably constructed, the model might then be used for prediction or analysis of behavior. This book will develop a large number of models and techniques used in this framework.

The **linear regression model** is the single most useful tool in the econometrician's kit. Though to an increasing degree in the contemporary literature, it is often only the departure point for the full analysis, it remains the device used to begin almost all empirical research. This chapter will develop the model. The next several chapters will discuss more elaborate specifications and complications that arise in the application of techniques that are based on the simple models presented here.

William H. Greene

$$Y_i = \beta_1 + \beta_2 X_{2i} + \ldots + \beta_k X_{ki} + u_i$$

X değişkenlerinin değerlerindeki değişimlere Y 'nın
ortalaması nasıl tepki veriyor?

Gelecek Bilanço Dönemi için Modelleme Çalışması: Türk Hava Yolları 2023Q3 Brüt Kar Öngörüsü

Adım Adım Modelleme

- Genel Bakış
- Brüt Kar, Bileşenleri, Tarihsel Bakış
- Trafik Verileri: Ücretli Yolcu Kilometre, Kargo + Posta (Ton)
- Yolcu Hasılatı, Kargo Hasılatı ve Satışlarının Maliyeti Modeli
- Varsayımlar
- Öngörü

Genel Bakış

2023 yılının ilk altı ayında;

- 189 milyar 583 milyon TL'lik hasılata (+66%). USD: +25%.
- 150 milyar 222 milyon TL'lik satışların maliyeti (+63%). USD: +23%.
- 39 milyar 468 milyon TL'lik brüt kar (+77%). USD: 34%.

Brüt Kar, Bileşenleri, Tarihsel Bakış

Brüt Kar = Hasılat - Satışların Maliyeti

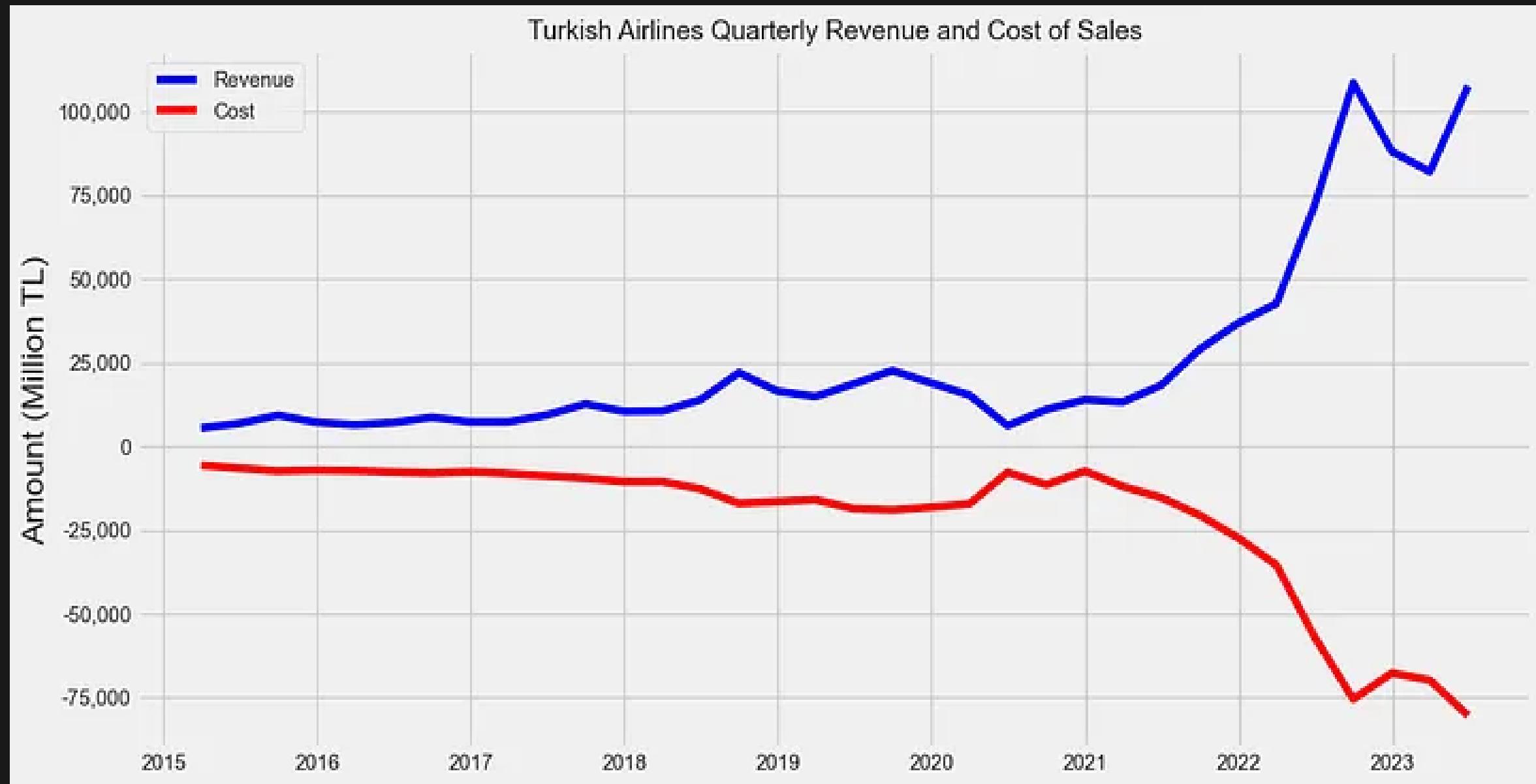
TÜRK HAVA YOLLARI ANONİM ORTAKLIĞI VE BAĞLI ORTAKLIKLARI

30 Haziran 2023 Tarihinde Sona Eren Altı Aylık Hesap Dönemine Ait

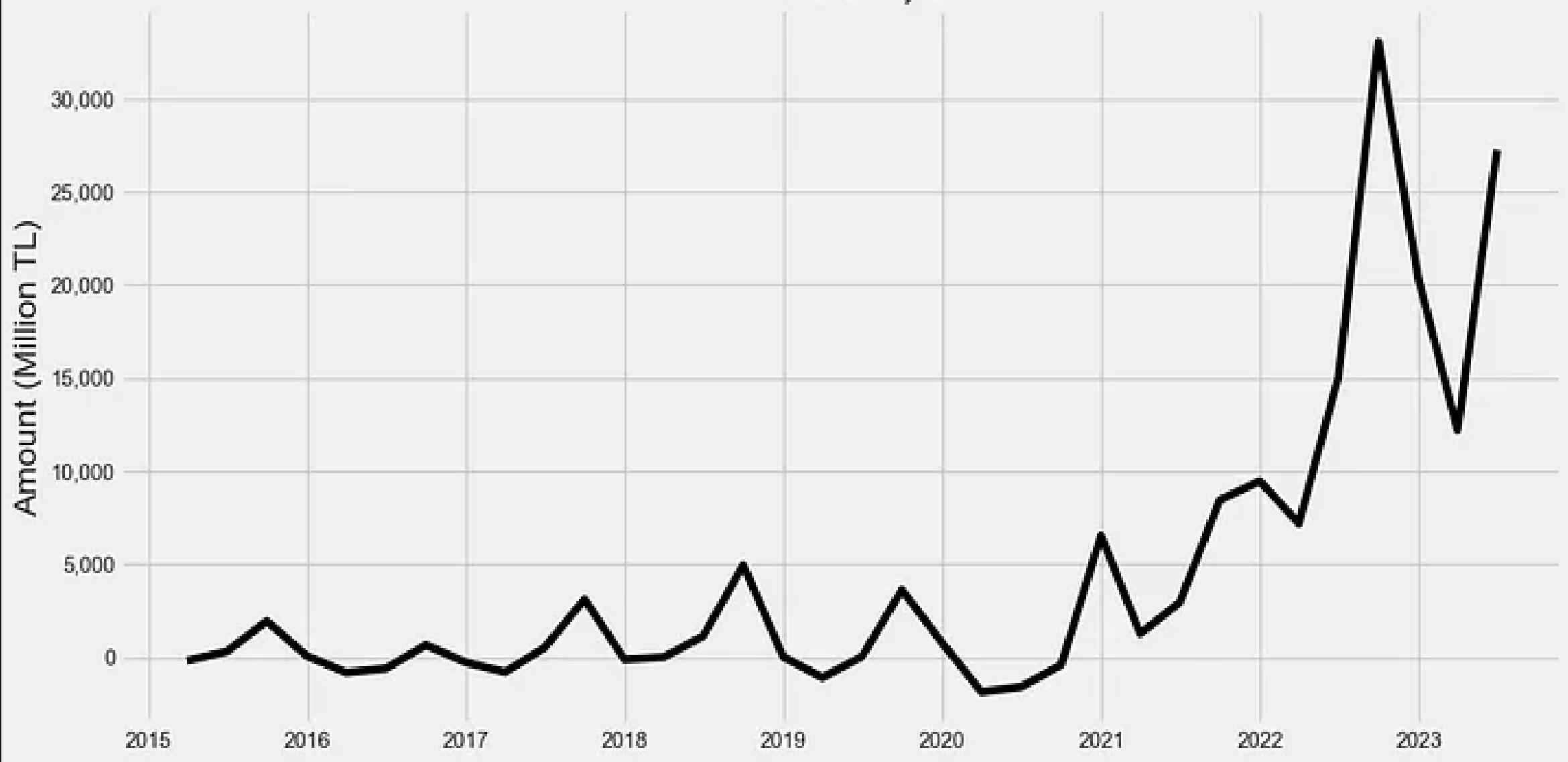
Özet Konsolide Kar veya Zarar ve Diğer Kapsamlı Gelir Tablosu

(Tutarlar, aksi belirtilmemişçe, Milyon Türk Lirası (TL) olarak gösterilmiştir.)

	Dipnot Referansları	Sınırlı Denetimden Geçmiş	Sınırlı Denetimden Geçmemiş	Sınırlı Denetimden Geçmiş	Sınırlı Denetimden Geçmemiş
		1 Ocak- 30 Haziran 2023	1 Nisan - 30 Haziran 2023	1 Ocak- 30 Haziran 2022	1 Nisan - 30 Haziran 2022
KAR VEYA ZARAR KISMI					
Hastalat	19	189.690	107.633	114.583	71.969
Satışların Maliyeti (-)	20	(150.222)	(80.383)	(92.334)	(56.908)
BRÜT KAR		39.468	27.250	22.249	15.061
Genel Yönetim Giderleri (-)	21	(4.424)	(2.580)	(1.995)	(1.031)
Pazarlama Giderleri (-)	21	(17.214)	(9.370)	(9.707)	(5.522)
Esas Faaliyetlerden Diğer Gelirler	22	3.921	2.115	1.070	653
Esas Faaliyetlerden Diğer Giderler (-)	22	(2.461)	(276)	(638)	(476)
ESAS FAALİYET KARI		19.290	17.139	10.979	8.685
Yatırım Faaliyetlerinden Gelirler	23	7.313	4.458	1.614	976
Yatırım Faaliyetlerinden Giderler (-)	23	(285)	(64)	(333)	-
Özkaynak Yöntemiyle Değerlenen Yatırımların Karlanmadan Paylar	3	625	857	179	245
İNANSMAN GELİRİ ÖNCESİ FAALİYET KARI		26.943	22.390	12.439	9.906
Finansman Gelirleri	24	7.685	1.999	2.133	1.645
Finansman Giderleri (-)	24	(10.330)	(5.309)	(2.869)	(1.502)
SÜRDÜRÜLEN FAALİYETLER VERGİ ÖNCESİ KARI		24.298	19.080	11.703	10.049
Sürdürülen Faaliyetler Vergi Gideri		(6.125)	(5.322)	(328)	(894)
Dönem Vergi Gideri	25	(77)	1	(359)	(241)
Ertelenmiş Vergi (Gideri)/Geliri	25	(6.048)	(5.323)	31	(653)
SÜRDÜRÜLEN FAALİYETLER DÖNEM KARI		18.173	13.758	11.375	9.155
DÖNEM KARININ DAĞILIMI					
- Kontrol Gücü Olmayan Paylar		1	4	-	-
- Ana Ortaklık Payları		18.172	13.754	11.375	9.155



Turkish Airlines Quarterly Gross Profit



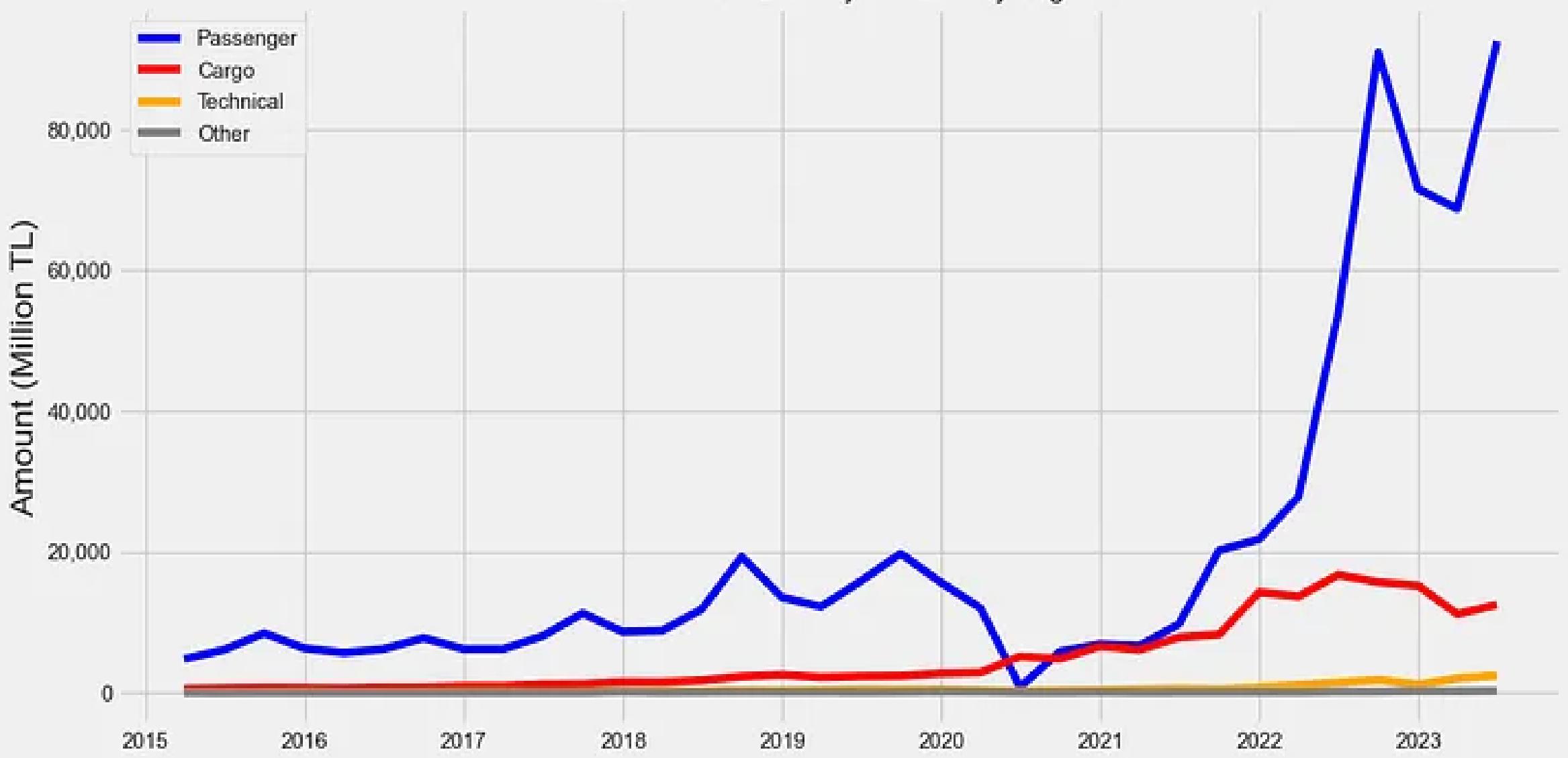
Trafik Verileri:

Ücretli Yolcu Kilometre, Kargo + Posta (Ton)

	1 Ocak - 30 Haziran 2023	1 Nisan - 30 Haziran 2023	1 Ocak - 30 Haziran 2022	1 Nisan - 30 Haziran 2022
Yolcu gelirleri				
Tarifeli hizmetler	160.330	91.757	80.940	53.241
Tarifesiz hizmetler	846	709	563	464
Toplam yolcu gelirleri	161.176	92.466	81.503	53.705
Kargo gelirleri				
Yolcu uçaklarında taşınan	11.383	5.969	12.305	6.859
Kargo uçaklarında taşınan	12.244	6.518	18.074	9.856
Toplam kargo gelirleri	23.627	12.487	30.379	16.715
Toplam yolcu ve kargo gelirleri	184.803	104.953	111.882	70.420
Teknik gelirler				
Diğer gelirler	4.439	2.424	2.504	1.411
Net satışlar	448	256	197	138
Satışların maliyeti (-)	189.690	107.633	114.583	71.969
Brüt kar	(150.222)	(80.383)	(92.334)	(56.908)
	39.468	27.250	22.249	15.061

- Yolcu hasılatı payı: 85.6%
- Kargo hasılatı payı: 11.6%
- Teknik hasılat payı: 2.3%
- Diğer hasılatlar payı: 0.2%

Turkish Airlines Quarterly Revenue by Segment



Turkish Airlines Quarterly Revenue by Segment (%)



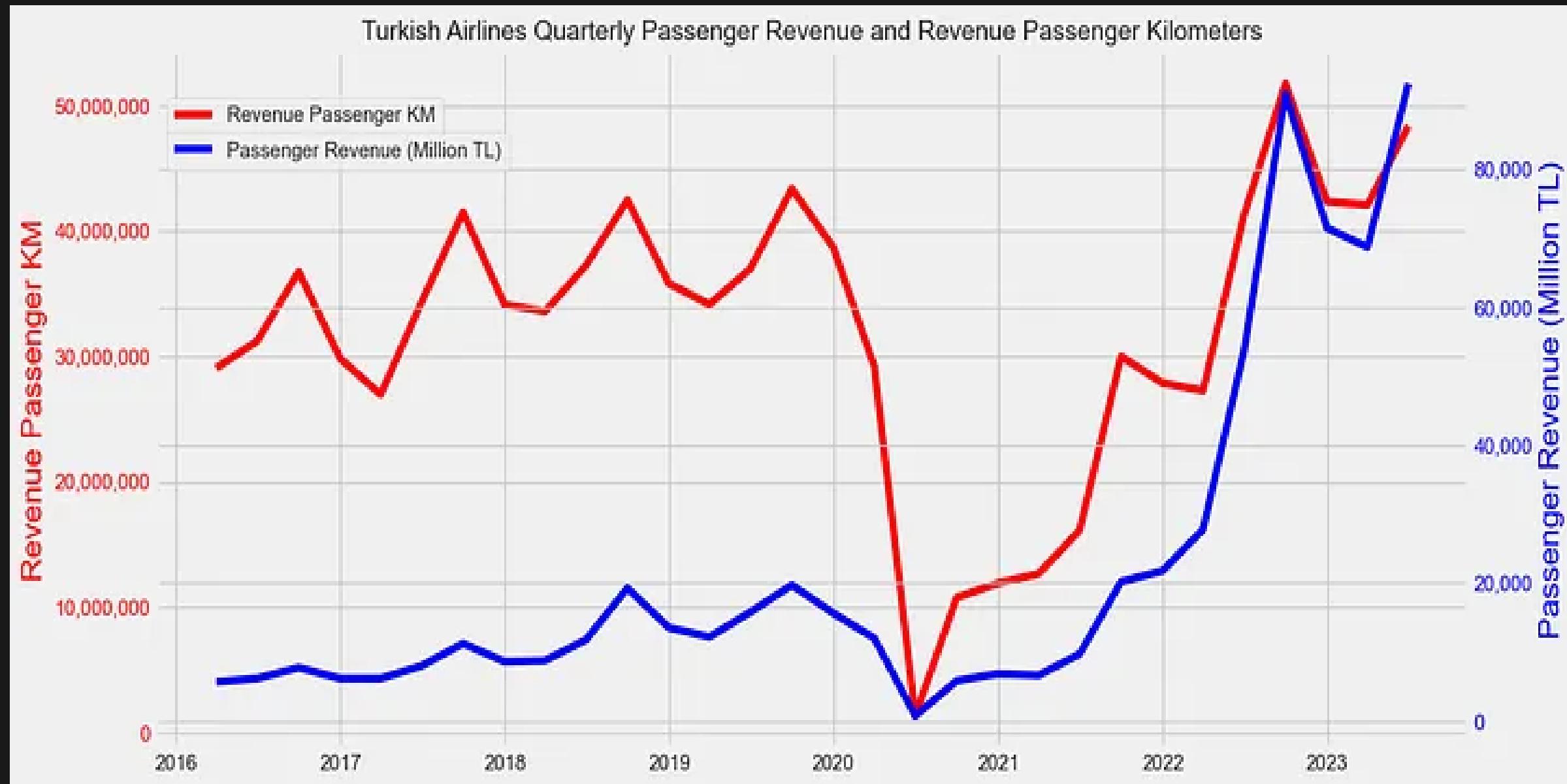
2020Q2: Kargo 83%, Yolcu 13%.

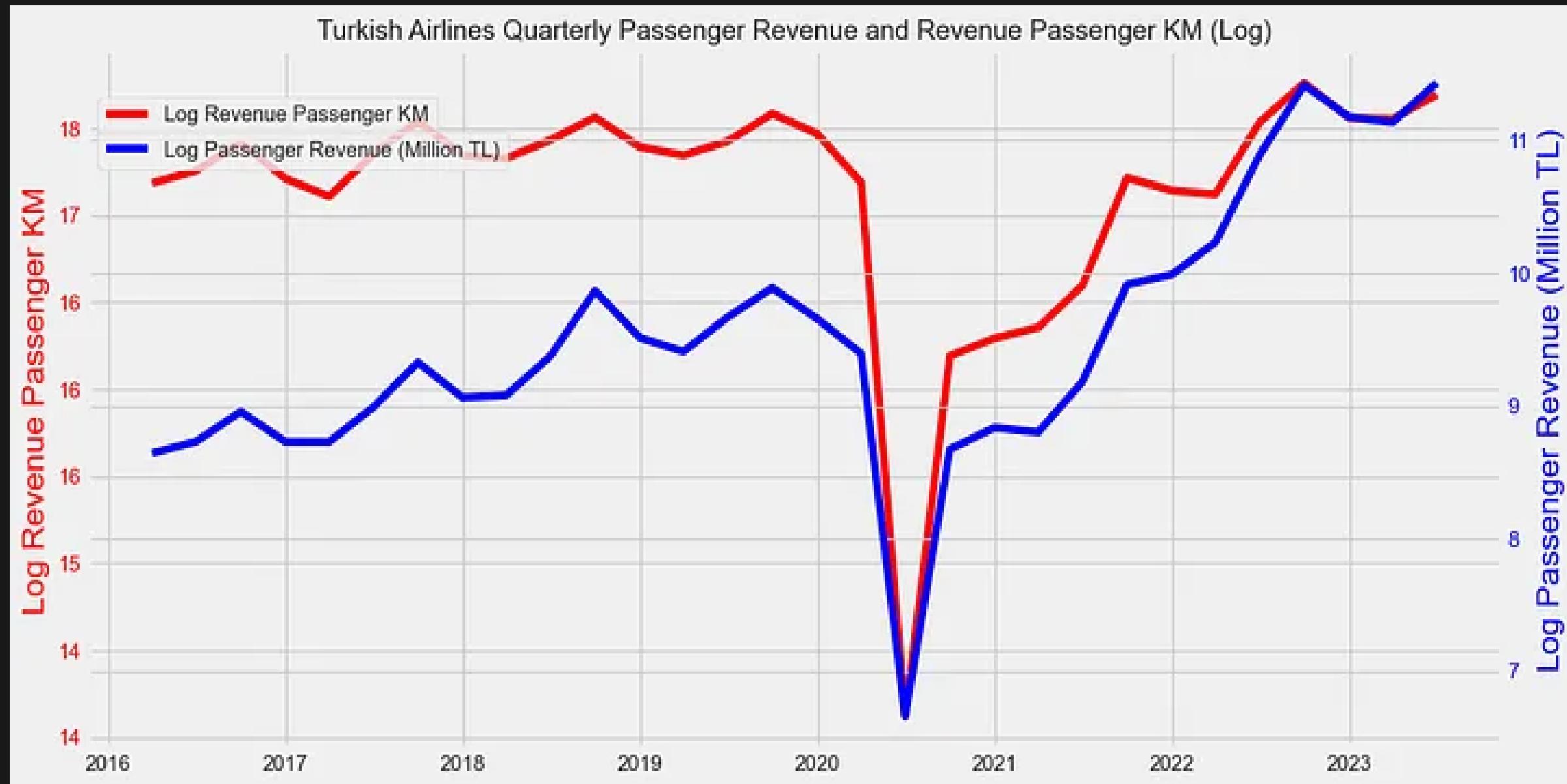
TRAFİK VERİLERİ – EYLÜL 2023

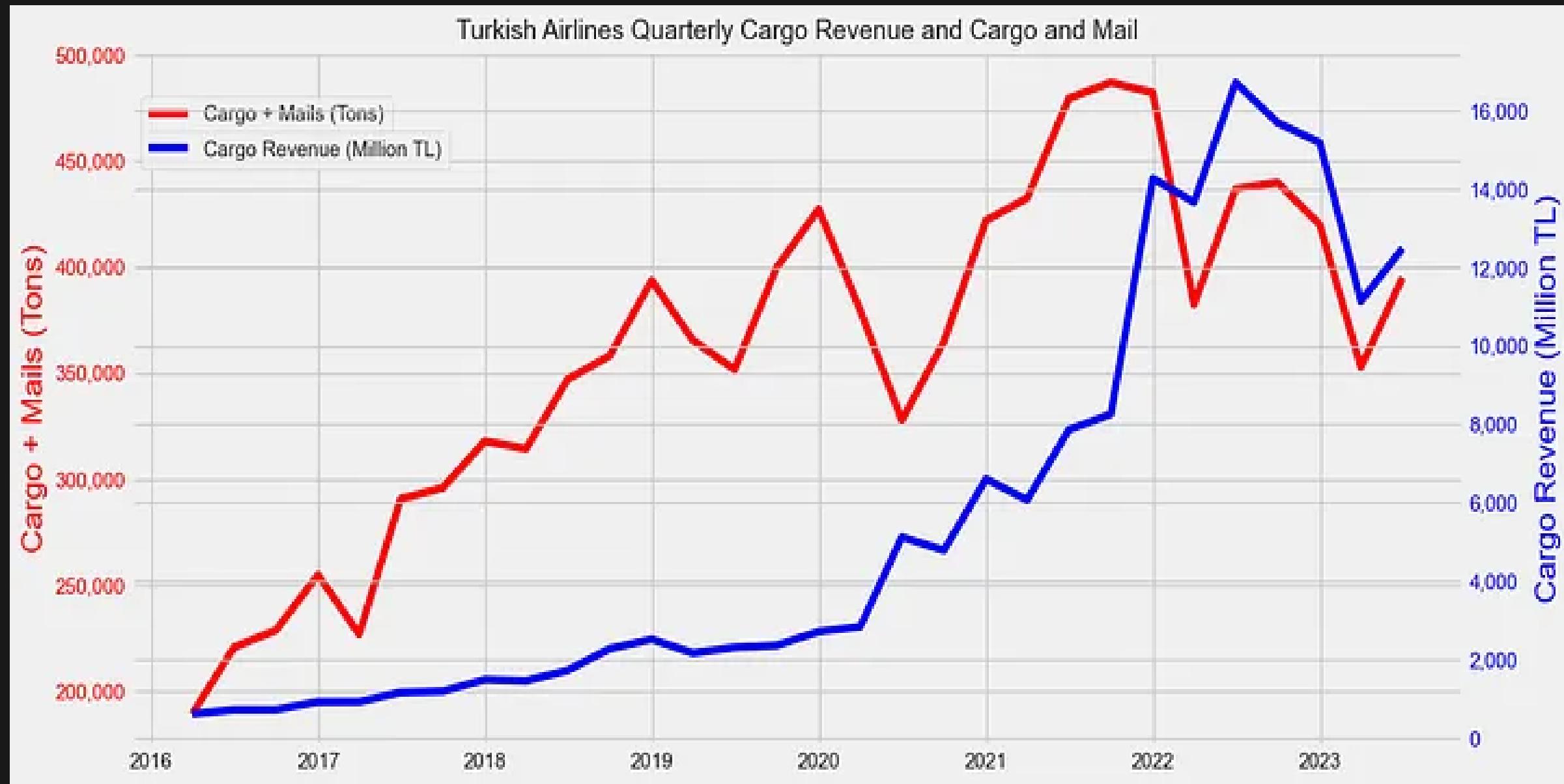
	TOPLAM		
	2022	2023	Değişim (%)
Konma Sayısı(Yolcu Seferleri)	42.909	47.633	11,0%
Arzedilen Koltuk Km ('000)	19.335.074	21.058.143	8,9%
Ücretli Yolcu Km ('000)	16.489.447	17.868.686	8,4%
Yolcu Doluluk Oranı (%)	85,3%	84,9%	-0,4 pt
Yolcu Sayısı	7.272.648	7.943.811	9,2%
Dıştan Dışa Transfer Yolcu Sayısı	2.317.242	2.655.166	14,6%
Kargo + Posta (Ton)	145.414	150.333	3,4%
Uçak Sayısı	390	429	10,0%
Koltuk Kapasitesi	78.909	87.946	11,5%
Uçulan Nokta (Şehir Bazında)	335	339	1,2%
Uçulan Km ('000)	90.034	96.137	6,8%
Uçulan Saat	143.070	154.574	8,0%

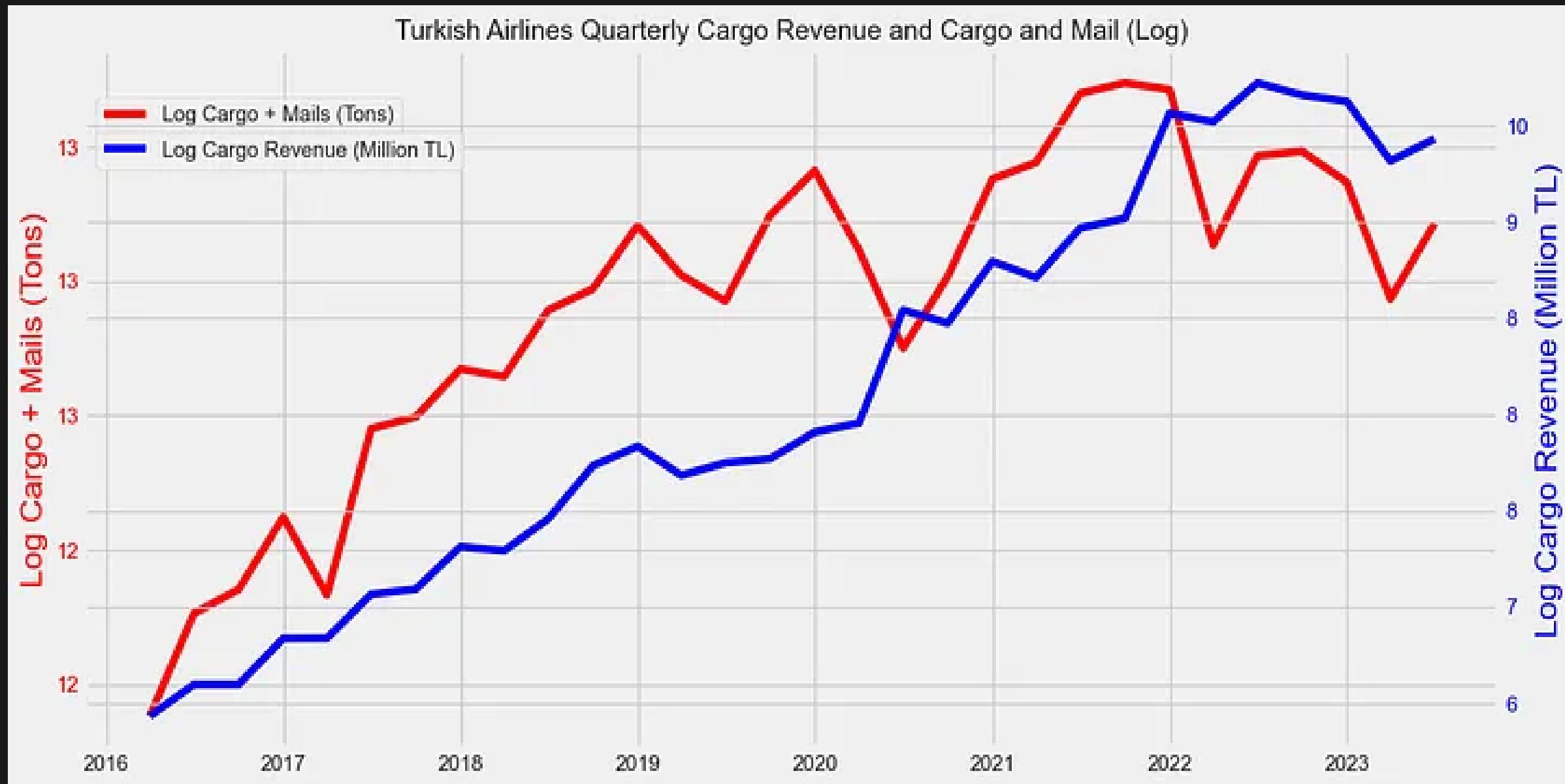
- Yolcu hasılatı için **Ücretli Yolcu Kilometre (Revenue Passenger Kilometers, RPK)**
- Kargo hasılatı için **Kargo + Posta (Ton) (Cargo and Mails (Tons), CMT)**

- Ücretli Yolcu Kilometre, hava yolu şirketinin toplamda kaç ücretli yolcu ile ne kadar mesafe uçtuğunu; diğer bir ifadeyle, arz etmiş olduğu koltuk kilometrenin ne kadarını sattığını göstermektedir. RPK değeri, talebin bir ölçüsüdür.
- 2023 yılının ilk altı ayı: 146,2 milyar RPK.









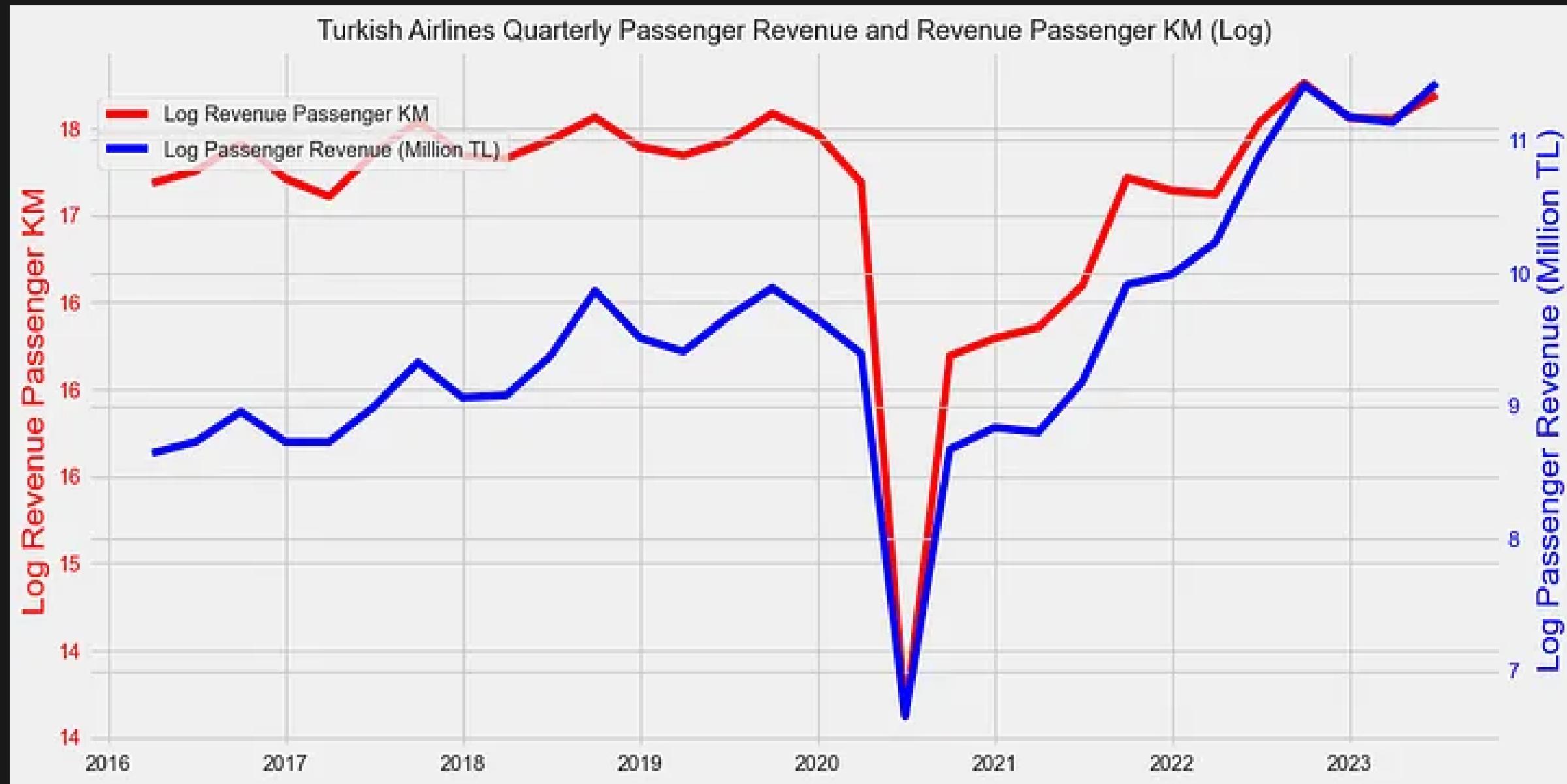
Neden logaritmik dönüşüm kullanırız?

- Doğrusal olmayan bazı ilişkiler doğrusallaştırılabilir.
- Büyük değerler küçük değerlere göre daha az önemli hale gelir. Değişen varyansın sabit varyansa dönüşümü.

Yolcu Hasılatı, Kargo Hasılatı ve Satışlarının Maliyeti Modeli

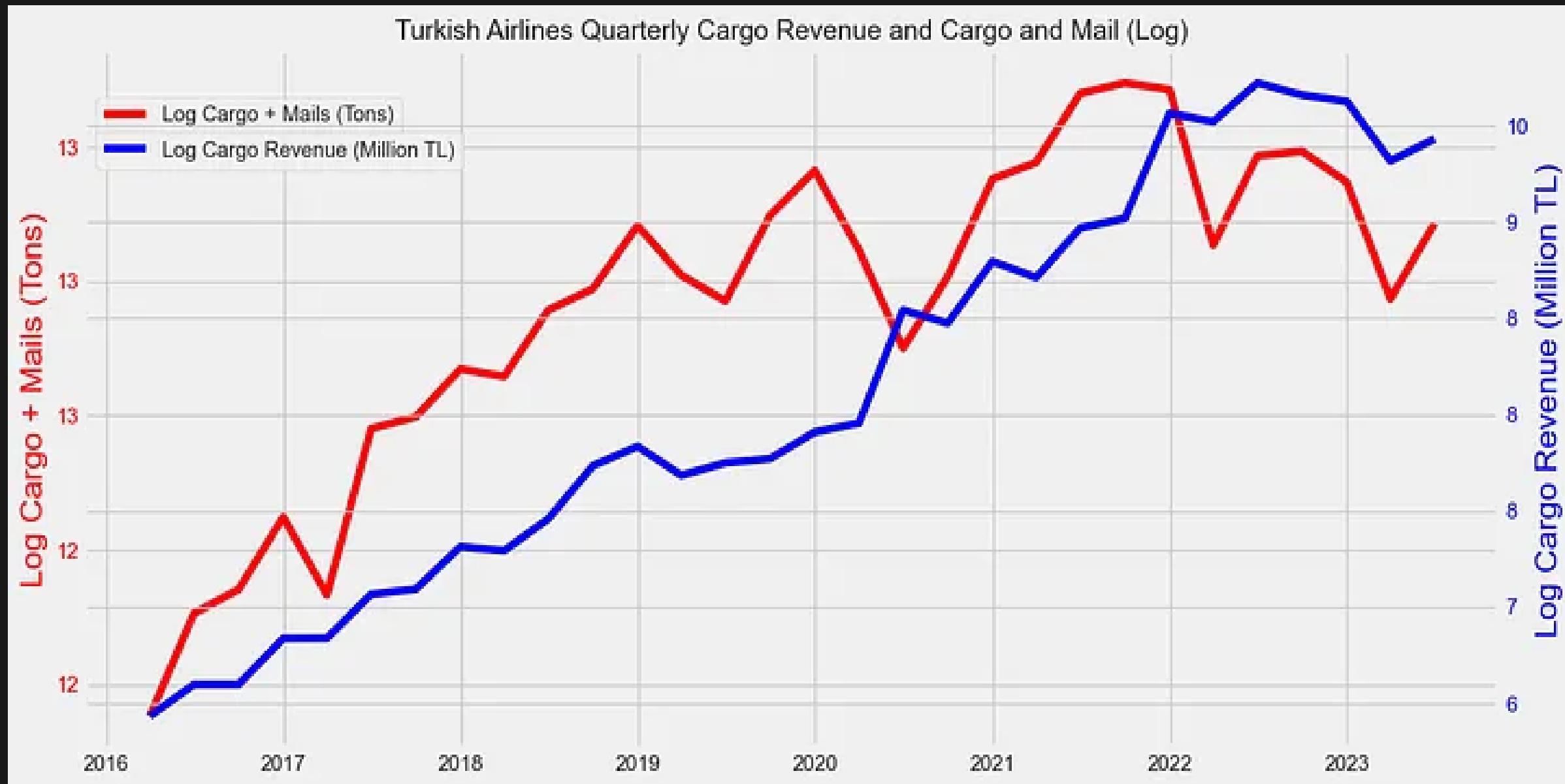
Grafiklerden neleri okuyabiliriz?

- Yolcu hasılatı ve RPK'da 2020 yılındaki kırılmadan önce yataya yakın bir seyir varken, 2020 yılındaki kırılmadan sonra net bir artış trendi göze çarpıyor.



Grafiklerden neleri okuyabiliriz?

- Kargo hasılatı ve CMT'de sürekli bir artış trendi varken, 2020 yılında kırılma yaşıyor ve sonrasında trendin yönünü bir miktar aşağıya çeviriyor.



Sonuç?

Genel olarak 2020 yılı öncesine, 2020 yılına ve 2020 yılı sonrası bir kukla değişken koymayı deneyebiliriz.

Kukla Değişken Tuzağı:

- Eğer modelde sabit parametreye yer veriyorsak kukla değişken sayısının bir eksiği kadar eklemeliyiz.
- Eğer sabit parametreyi dahil etmeyeceksek bu durumda tüm kukla değişkenler modele girebilir.

Otokorelasyon Probleminden Kurtulmak:
Modellere bağımlı değişkenlerin gecikmelerini
ekleyebiliriz.

Dinamik Model:

Bağımsız değişkenler arasında bağımlı ve bağımsız değişkenlerin gecikmeli değerleri de yer almaktadır.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from statsmodels.graphics.tsaplots import plot_acf
import statsmodels.stats.diagnostic as dg
from statsmodels.stats.diagnostic import het_breuschpagan
from scipy.stats import shapiro
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_style("whitegrid")
plt.style.use('fivethirtyeight')
```

```
model_data = pd.read_excel('./3-econometrics/model_data.xlsx', sheet_name=None)

revenue_df = model_data['Revenue']
cost_df = model_data['Cost']
rpk_df = model_data['Traffic'].query("Category == 'TOTAL'")[
    'Date',
    "Revenue Passenger Km ('000)",
    'Cargo and Mail (Tons)'
].dropna()
rpk_df = rpk_df.rename(columns={
    "Revenue Passenger Km ('000)": "RPK",
    "Cargo and Mail (Tons)": "CMT"
})
rpk_df = rpk_df.set_index('Date').resample('Q').sum().reset_index()

master_df = revenue_df.merge(cost_df, on='Date')
master_df['Gross Profit'] = master_df['Revenue'] - abs(master_df['Cost'])
master_df = master_df.merge(rpk_df, on='Date')
master_df['Date'] = pd.to_datetime(master_df['Date'])
```

```
# Dönüşüm

revenue_model_df = master_df[['Date', 'Revenue', 'RPK', 'CMT', 'Passenger', 'Cargo']]
revenue_model_df['Date'] = pd.to_datetime(revenue_model_df['Date'])

revenue_model_df.loc[:, 'Log_Revenue'] = np.log(revenue_model_df['Revenue'])
revenue_model_df.loc[:, 'Log_RPK'] = np.log(revenue_model_df['RPK'])
revenue_model_df.loc[:, 'Log_CMT'] = np.log(revenue_model_df['CMT'])
revenue_model_df.loc[:, 'Log_Passenger'] = np.log(revenue_model_df['Passenger'])
revenue_model_df.loc[:, 'Log_Cargo'] = np.log(revenue_model_df['Cargo'])
```

```
# Kukla

revenue_model_df[ 'Before' ] = (
    revenue_model_df[ 'Date' ] < pd.to_datetime('2020-06-30')
).astype(int)
revenue_model_df[ 'Covid19' ] = (
    revenue_model_df[ 'Date' ] == pd.to_datetime('2020-06-30')
).astype(int)
revenue_model_df[ 'After' ] = (
    revenue_model_df[ 'Date' ] > pd.to_datetime('2020-06-30')
).astype(int)
```

```
# Gecikmeler

revenue_model_df['Log_Passenger_L1'] = revenue_model_df['Log_Passenger'].shift(1)
revenue_model_df['Log_Passenger_L2'] = revenue_model_df['Log_Passenger'].shift(2)
revenue_model_df['Log_Passenger_L3'] = revenue_model_df['Log_Passenger'].shift(3)
revenue_model_df['Log_Passenger_L4'] = revenue_model_df['Log_Passenger'].shift(4)

revenue_model_df['Log_RPK_L1'] = revenue_model_df['Log_RPK'].shift(1)
revenue_model_df['Log_RPK_L2'] = revenue_model_df['Log_RPK'].shift(2)
revenue_model_df['Log_RPK_L3'] = revenue_model_df['Log_RPK'].shift(3)
revenue_model_df['Log_RPK_L4'] = revenue_model_df['Log_RPK'].shift(4)

revenue_model_df['Log_Cargo_L1'] = revenue_model_df['Log_Cargo'].shift(1)
revenue_model_df['Log_Cargo_L2'] = revenue_model_df['Log_Cargo'].shift(2)
revenue_model_df['Log_Cargo_L3'] = revenue_model_df['Log_Cargo'].shift(3)
revenue_model_df['Log_Cargo_L4'] = revenue_model_df['Log_Cargo'].shift(4)

revenue_model_df['Log_CMT_L1'] = revenue_model_df['Log_CMT'].shift(1)
revenue_model_df['Log_CMT_L2'] = revenue_model_df['Log_CMT'].shift(2)
revenue_model_df['Log_CMT_L3'] = revenue_model_df['Log_CMT'].shift(3)
revenue_model_df['Log_CMT_L4'] = revenue_model_df['Log_CMT'].shift(4)

revenue_model_df = revenue_model_df.dropna()
```

Yolcu Hasılatı Modeli

```
X = revenue_model_df[[
    'Log_Passenger_L1',
    'Log_RPK_L1',
    'Log_RPK',
    'Before',
    'After'
]]
y = revenue_model_df['Log_Passenger']
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
print(model.summary())
```

Yolcu hasılatı modelini hangi değişkenler ile açıklıyoruz?

- Logaritmik yolcu hasılatının bir dönem gecikmesi,
- Logaritmik RPK'nın bir dönem gecikmesi,
- Logaritmik RPK,
- 2020 yılı öncesi için belirlenen kukla değişken,
- 2020 yılı sonrası için belirlenen kukla değişken.

OLS Regression Results

=====
 Dep. Variable: Log_Passenger R-squared: 0.990
 Model: OLS Adj. R-squared: 0.987
 Method: Least Squares F-statistic: 457.2
 Date: Sun, 29 Oct 2023 Prob (F-statistic): 5.66e-23
 Time: 23:47:40 Log-Likelihood: 26.244
 No. Observations: 30 AIC: -40.49
 Df Residuals: 24 BIC: -32.08
 Df Model: 5
 Covariance Type: nonrobust
 =====

	coef	std err	t	P> t	[0.025	0.975]
const	-5.4610	1.644	-3.321	0.003	-8.855	-2.067
Log_Passenger_L1	0.7843	0.080	9.762	0.000	0.618	0.950
Log_RPK_L1	-0.8156	0.104	-7.816	0.000	-1.031	-0.600
Log_RPK	1.3296	0.109	12.228	0.000	1.105	1.554
Before	-1.4389	0.379	-3.801	0.001	-2.220	-0.658
After	-1.0633	0.339	-3.137	0.004	-1.763	-0.364

 =====

Omnibus:	3.316	Durbin-Watson:	2.036
Prob(Omnibus):	0.191	Jarque-Bera (JB):	2.521
Skew:	0.710	Prob(JB):	0.284
Kurtosis:	2.985	Cond. No.	2.12e+03

...

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 2.12e+03. This might indicate that there are strong multicollinearity or other numerical problems.

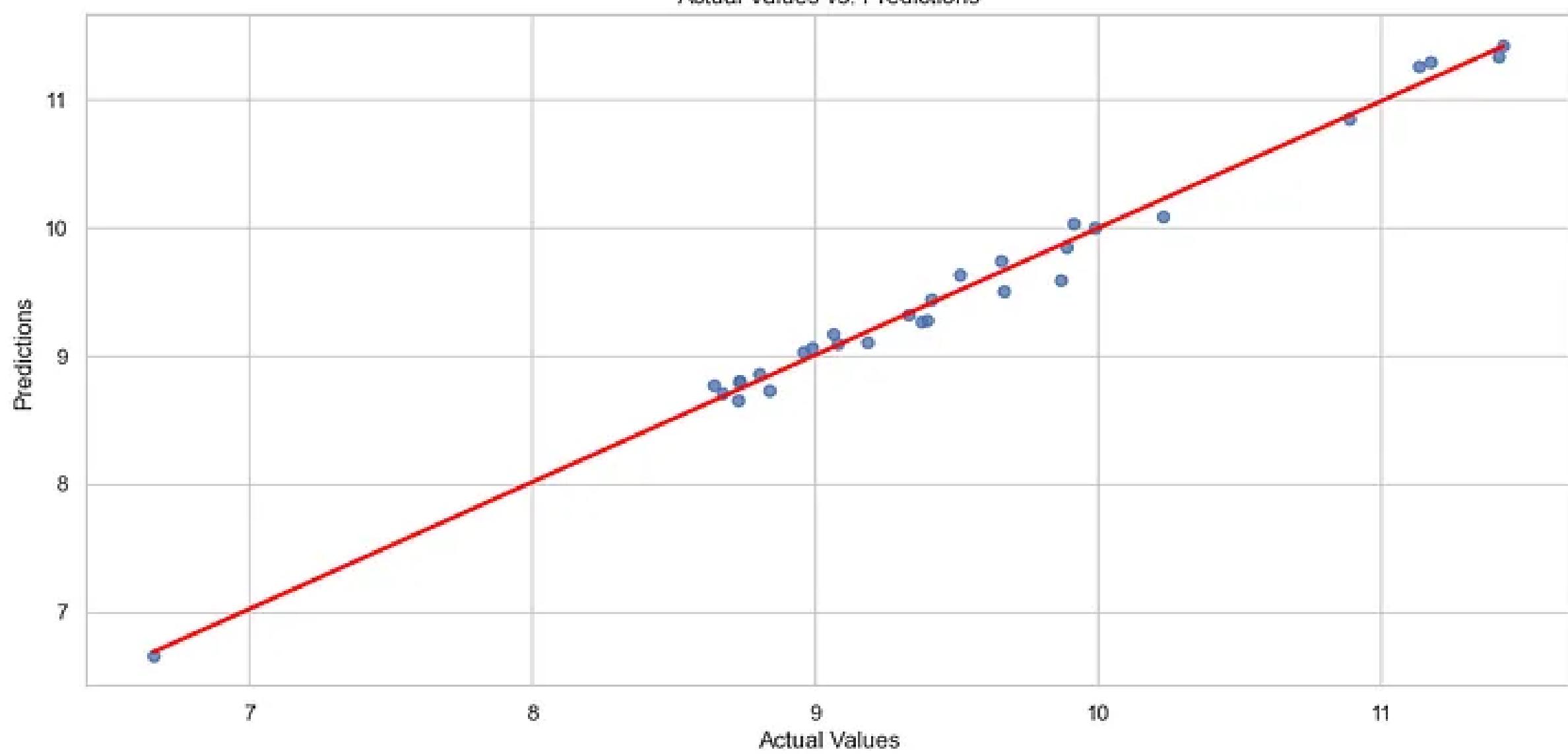
Gerçek ve Tahmin Edilen Değerler (Doğrusallık)

```
residuals = model.resid
fitted_values = model.fittedvalues

predictions = model.predict()

plt.figure(figsize=(12, 6))
sns.regplot(x=y, y=predictions, ci=None, line_kws={"color": "red"})
plt.xlabel("Actual Values")
plt.ylabel("Predictions")
plt.title("Actual Values vs. Predictions")
plt.show()
```

Actual Values vs. Predictions

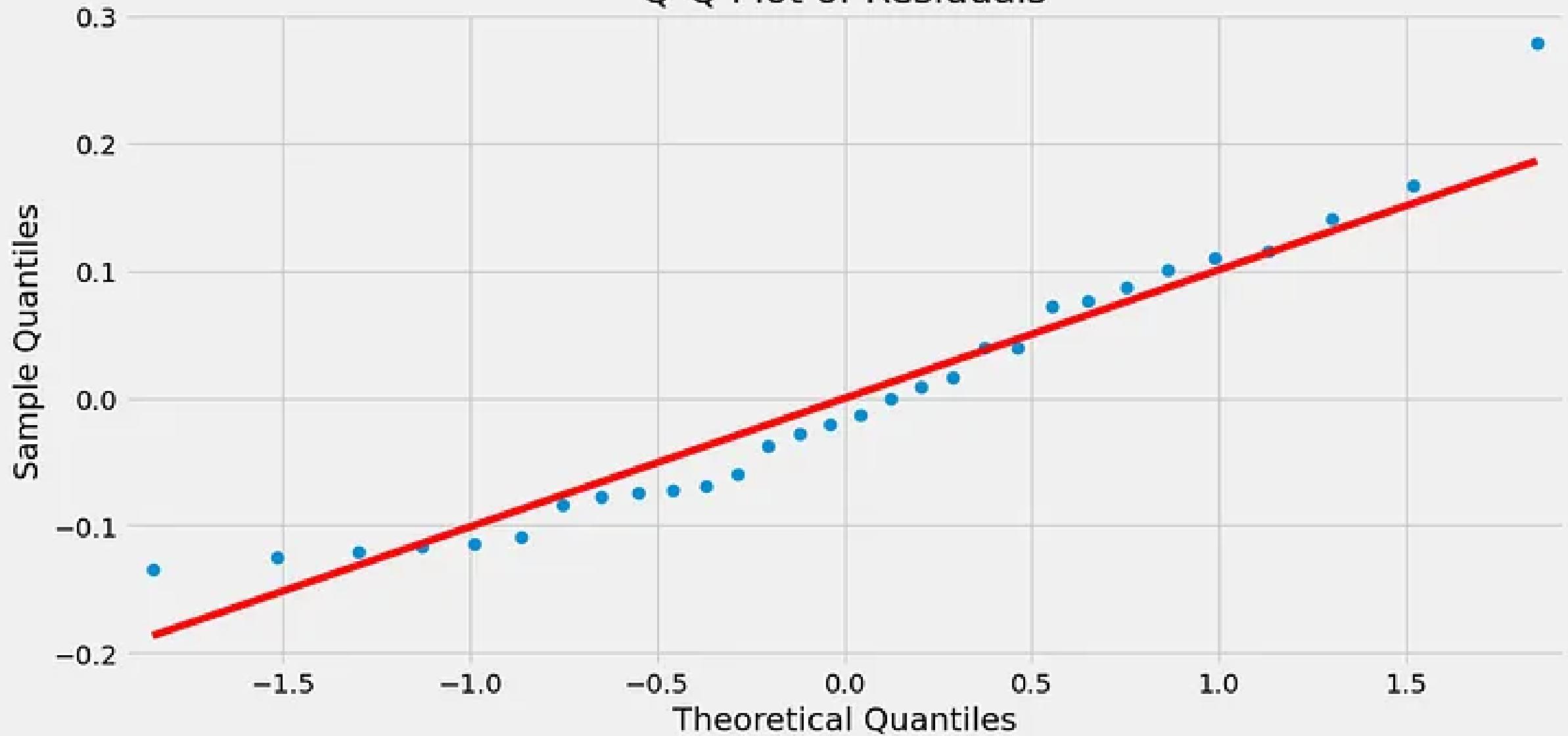


Normallik / Shapiro-Wilk Testi

Örneklem büyüklüğü nispeten küçükse oldukça önemlidir. Çünkü t ve F gibi anlamlılık testleri normallik varsayıımına dayanır.

```
fig, ax = plt.subplots(figsize=(12, 6))
sm.qqplot(residuals, line='s', ax=ax)
ax.set_title("Q-Q Plot of Residuals")
ax.set_xlabel("Theoretical Quantiles")
ax.set_ylabel("Sample Quantiles")
plt.show()
```

Q-Q Plot of Residuals



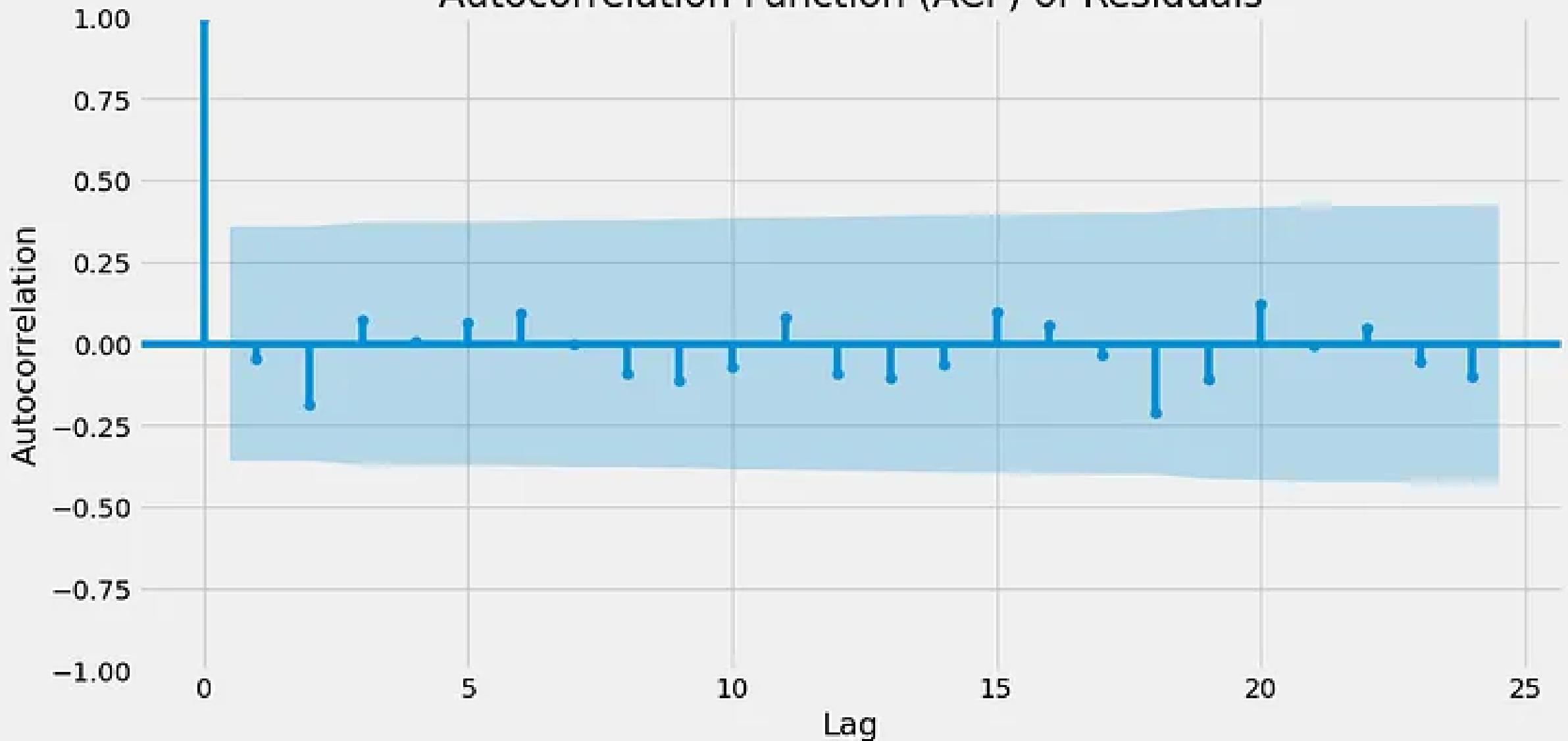
```
alpha = 0.01

sw_statistic, sw_p_value = shapiro(residuals)
if sw_p_value > alpha:
    print("The error terms follow a normal distribution.")
else:
    print("The error terms do not follow a normal distribution.")
```

Otokorelasyon / Breusch-Godfrey Testi

```
fig, ax = plt.subplots(figsize=(12, 6))
plot_acf(residuals, lags=24, ax=ax)
plt.title('Autocorrelation Function (ACF) of Residuals')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
```

Autocorrelation Function (ACF) of Residuals

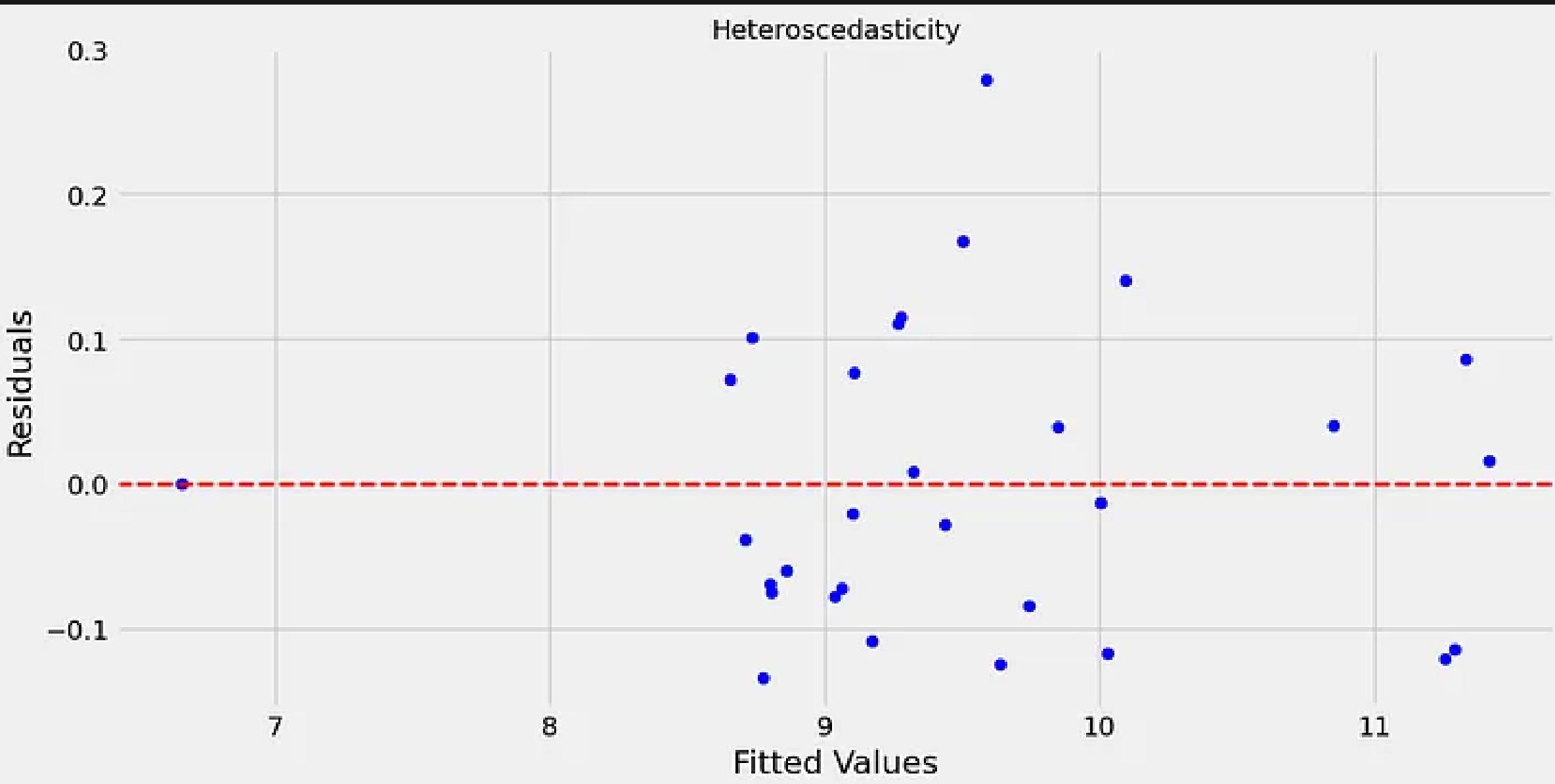


```
alpha = 0.01

bg_p_value = dg.acorr_breusch_godfrey(model, nlags=8)[1]
if bg_p_value > alpha:
    print("There is no autocorrelation among the error terms.")
else:
    print("There is autocorrelation among the error terms.")
```

Değişen Varyans / Breusch-Pagan Testi

```
plt.figure(figsize=(12, 6))
plt.scatter(fitted_values, residuals, color='blue')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title('Heteroscedasticity', fontsize=14)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



```
alpha = 0.01

X = model.model.exog
bp_test_result = het_breuschpagan(residuals, X)

if bp_test_result[1] < alpha:
    print("Heteroskedasticity is present in the model.")
else:
    print("No evidence of heteroskedasticity in the model.")
```

Yolcu Hasılatı - Öngörü

```
input_data = {  
    'Log_Passenger_L1': 11.434596,  
    'Log_RPK_L1': 17.694290,  
    'Log_RPK': np.log(55820821.86),  
    'Before': 0,  
    'After': 1  
}  
  
X = [1]  
X += [  
    input_data['Log_Passenger_L1'],  
    input_data['Log_RPK_L1'],  
    input_data['Log_RPK'],  
    input_data['Before'],  
    input_data['After']  
]  
  
forecasted_log_passenger = model.predict(X)  
forecasted_passenger = np.exp(forecasted_log_passenger)  
  
print("Forecasted Passenger Revenue (Million TL):", forecasted_passenger)
```

Kargo Hasılatı Modeli

```
X = revenue_model_df[[
    'Log_Cargo_L1',
    'Log_Cargo_L2',
    'Log_Cargo_L3',
    'Log_CMT',
    'Before',
    'After'
]]
y = revenue_model_df['Log_Cargo']
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
print(model.summary())
```

OLS Regression Results

=====
 Dep. Variable: Log_Cargo R-squared: 0.990
 Model: OLS Adj. R-squared: 0.987
 Method: Least Squares F-statistic: 371.2
 Date: Sun, 29 Oct 2023 Prob (F-statistic): 1.06e-21
 Time: 23:50:37 Log-Likelihood: 24.954
 No. Observations: 30 AIC: -35.91
 Df Residuals: 23 BIC: -26.10
 Df Model: 6
 Covariance Type: nonrobust
 =====

	coef	std err	t	P> t	[0.025	0.975]
const	-4.8165	1.761	-2.736	0.012	-8.459	-1.174
Log_Cargo_L1	0.5512	0.148	3.716	0.001	0.244	0.858
Log_Cargo_L2	0.4737	0.154	3.083	0.005	0.156	0.791
Log_Cargo_L3	-0.2655	0.146	-1.824	0.081	-0.567	0.036
Log_CMT	0.5742	0.164	3.507	0.002	0.236	0.913
Before	-0.6189	0.131	-4.711	0.000	-0.891	-0.347
After	-0.3561	0.139	-2.568	0.017	-0.643	-0.069

=====
 Omnibus: 6.708 Durbin-Watson: 1.555
 Prob(Omnibus): 0.035 Jarque-Bera (JB): 5.455
 Skew: 0.667 Prob(JB): 0.0654
 ...
 Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.52e+03. This might indicate that there are strong multicollinearity or other numerical problems.

Gerçek ve Tahmin Edilen Değerler (Doğrusallık)

```
residuals = model.resid
fitted_values = model.fittedvalues

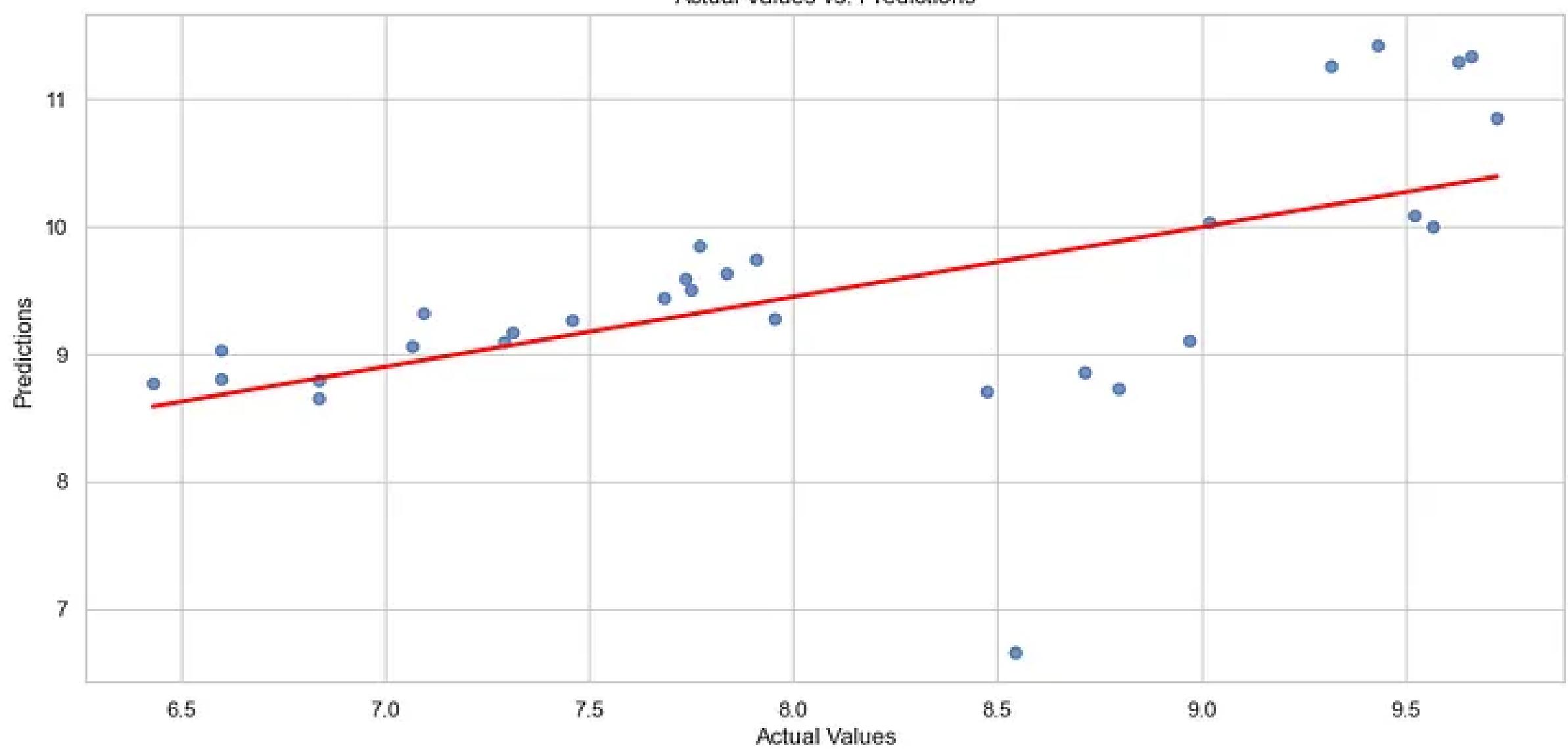
predictions = model.predict()

plt.figure(figsize=(12, 6))
sns.regplot(x=y, y=predictions, ci=None, line_kws={"color": "red"})
plt.xlabel("Actual Values")
plt.ylabel("Predictions")
plt.title("Actual Values vs. Predictions")
plt.show()
```

Kargo hasılatı modelini hangi değişkenler ile açıklıyoruz?

- Logaritmik kargo hasılatının bir dönem gecikmesi,
- Logaritmik kargo hasılatının iki dönem gecikmesi,
- Logaritmik kargo hasılatının üç dönem gecikmesi,
- Logaritmik CMT,
- 2020 yılı öncesi için belirlenen kukla değişken,
- 2020 yılı sonrası için belirlenen kukla değişken.

Actual Values vs. Predictions

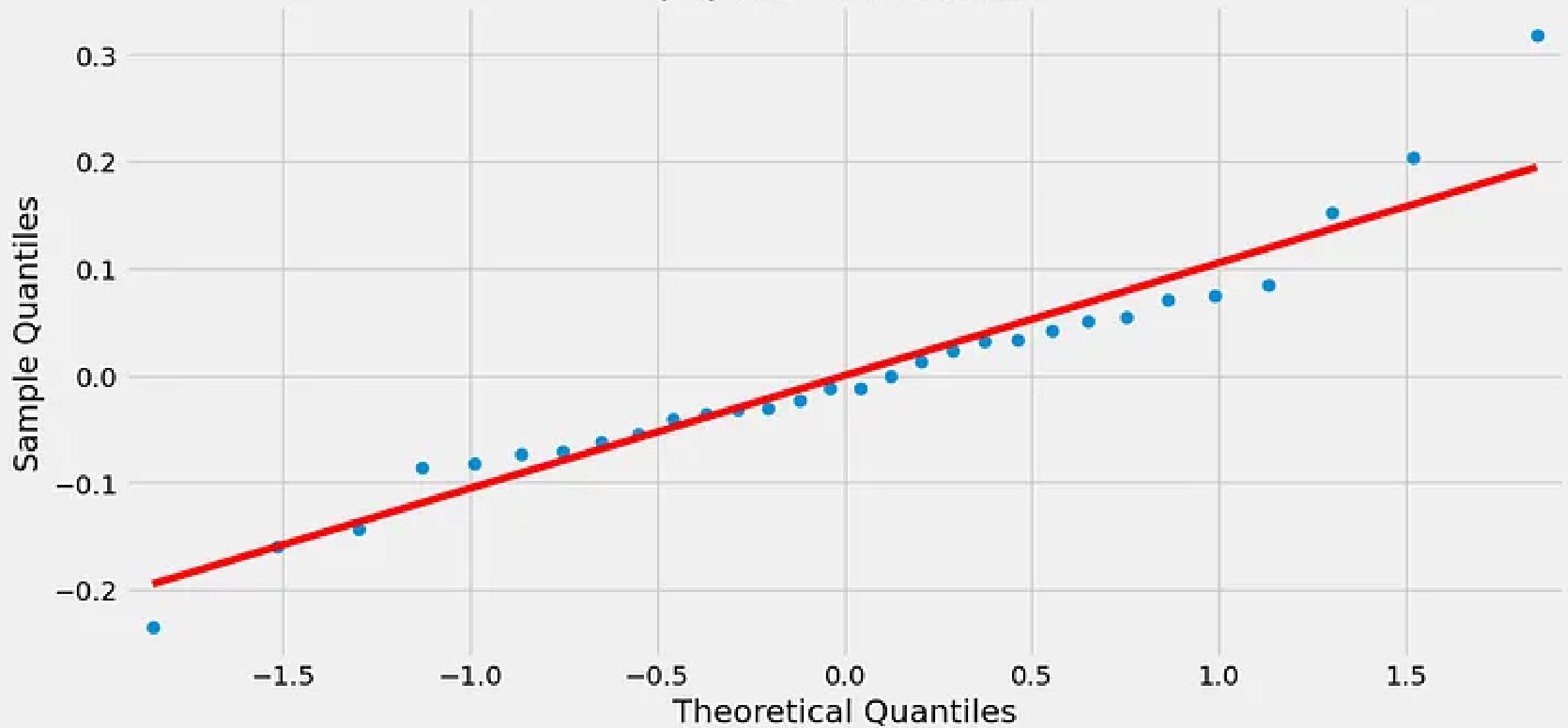


Normallik / Shapiro-Wilk Testi

Örneklem büyüklüğü nispeten küçükse oldukça önemlidir. Çünkü t ve F gibi anlamlılık testleri normallik varsayıımına dayanır.

```
fig, ax = plt.subplots(figsize=(12, 6))
sm.qqplot(residuals, line='s', ax=ax)
ax.set_title("Q-Q Plot of Residuals")
ax.set_xlabel("Theoretical Quantiles")
ax.set_ylabel("Sample Quantiles")
plt.show()
```

Q-Q Plot of Residuals



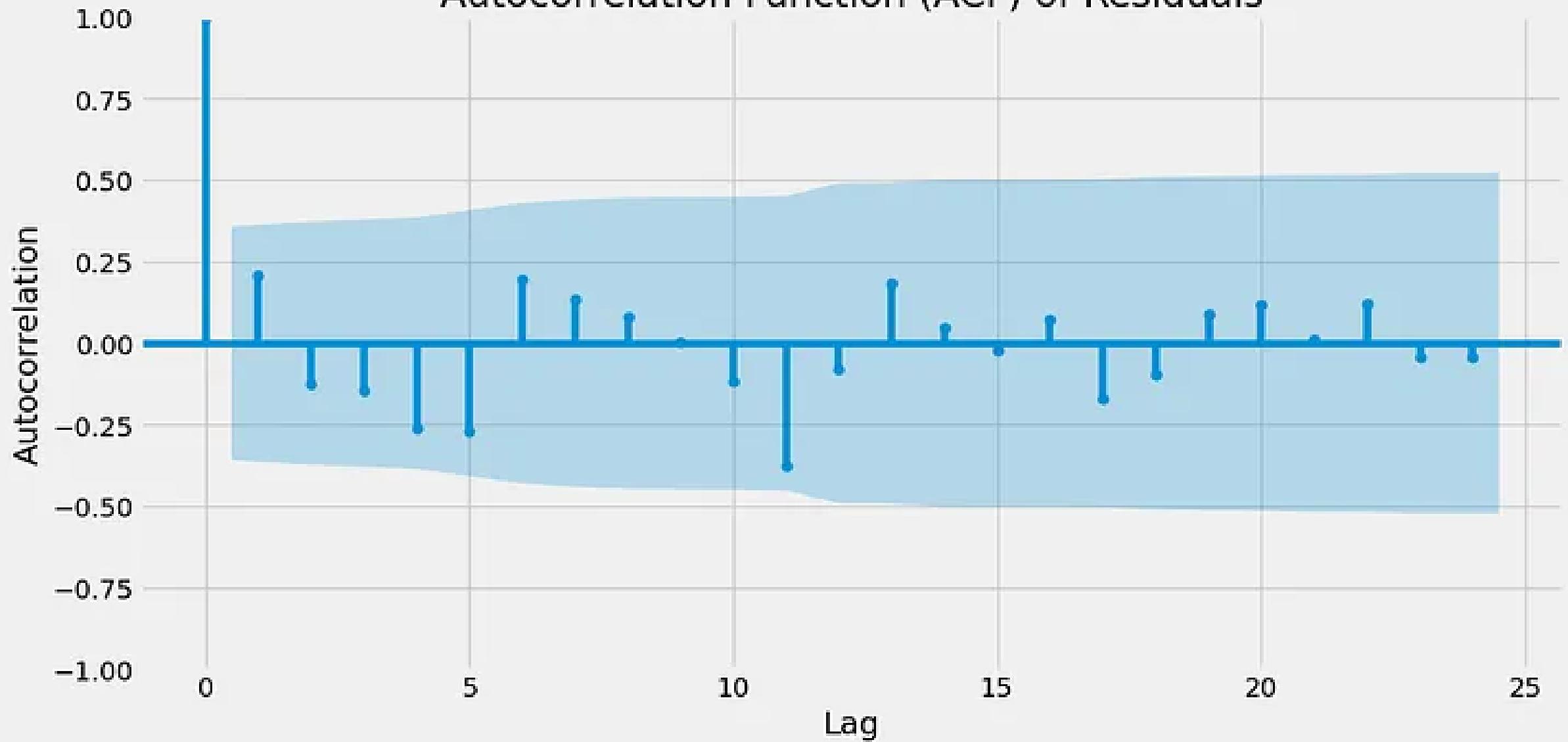
```
alpha = 0.01

sw_statistic, sw_p_value = shapiro(residuals)
if sw_p_value > alpha:
    print("The error terms follow a normal distribution.")
else:
    print("The error terms do not follow a normal distribution.")
```

Otokorelasyon / Breusch-Godfrey Testi

```
fig, ax = plt.subplots(figsize=(12, 6))
plot_acf(residuals, lags=24, ax=ax)
plt.title('Autocorrelation Function (ACF) of Residuals')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
```

Autocorrelation Function (ACF) of Residuals



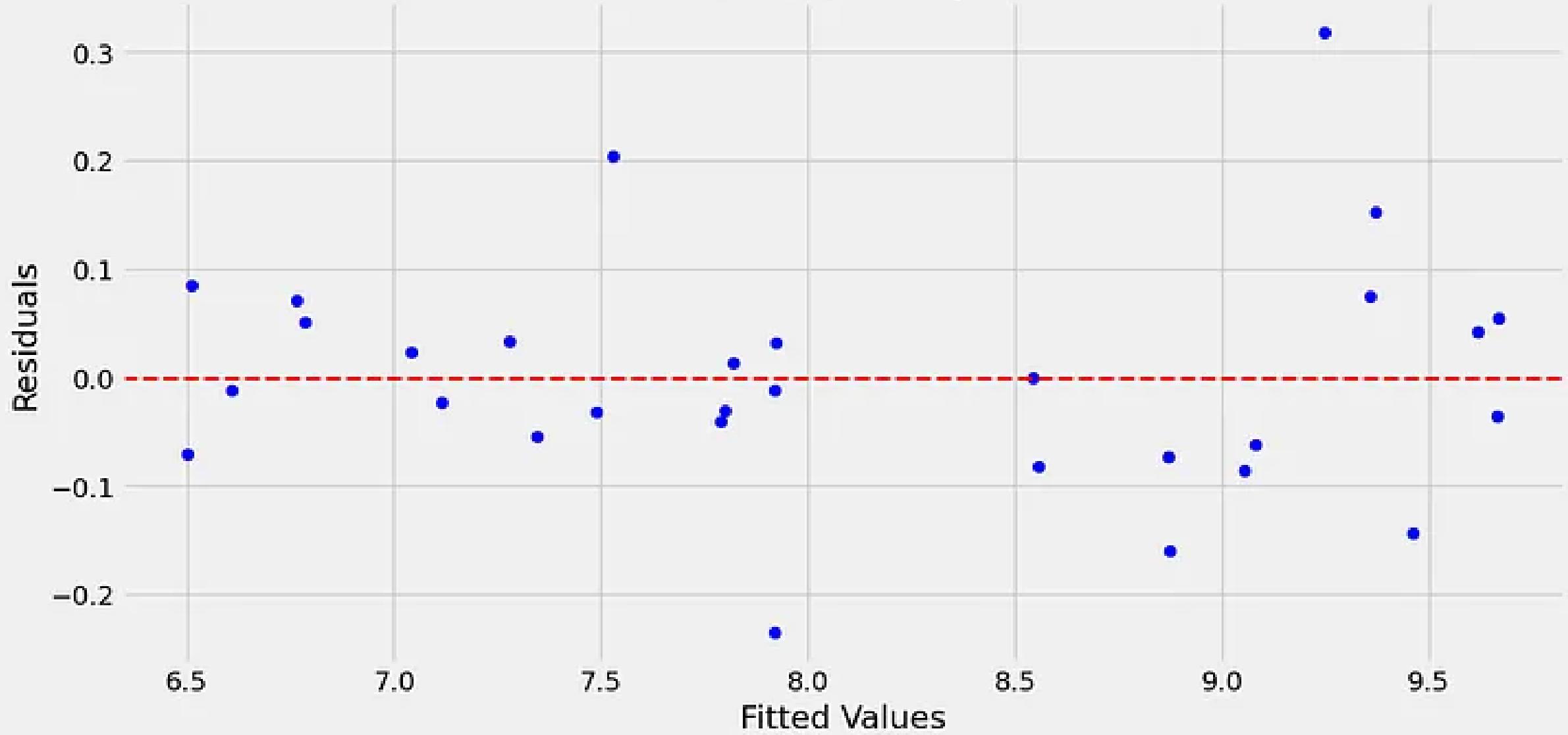
```
alpha = 0.01

bg_p_value = dg.acorr_breusch_godfrey(model, nlags=8)[1]
if bg_p_value > alpha:
    print("There is no autocorrelation among the error terms.")
else:
    print("There is autocorrelation among the error terms.")
```

Değişen Varyans / Breusch-Pagan Testi

```
plt.figure(figsize=(12, 6))
plt.scatter(fitted_values, residuals, color='blue')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title('Heteroscedasticity', fontsize=14)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```

Heteroscedasticity



```
X = model.model.exog
bp_test_result = het_breuschpagan(residuals, X)

if bp_test_result[1] < alpha:
    print("Heteroskedasticity is present in the model.")
else:
    print("No evidence of heteroskedasticity in the model.")
```

Kargo Hasılatı - Öngörü

```
input_data = {  
    'Log_Cargo_L1': 9.432443,  
    'Log_Cargo_L2': 9.318298,  
    'Log_Cargo_L3': 9.627932,  
    'Log_CMT': 12.886602,  
    'Before': 0,  
    'After': 1  
}  
  
X = [1]  
X += [  
    input_data['Log_Cargo_L1'],  
    input_data['Log_Cargo_L2'],  
    input_data['Log_Cargo_L3'],  
    input_data['Log_CMT'],  
    input_data['Before'],  
    input_data['After']  
]  
  
forecasted_log_cargo = model.predict(X)  
forecasted_cargo = np.exp(forecasted_log_cargo)  
  
print("Forecasted Cargo Revenue (Million TL):", forecasted_cargo)
```

Hasılat - Öngörü

Hasılat: X

Yolcu + Kargo Hasılatları: 134,735

Yolcu + Kargo Hasılatlarının Payı (Varsayımlı): 98%

$$X * 0.98 = 134,735$$

$$X = 137,485$$

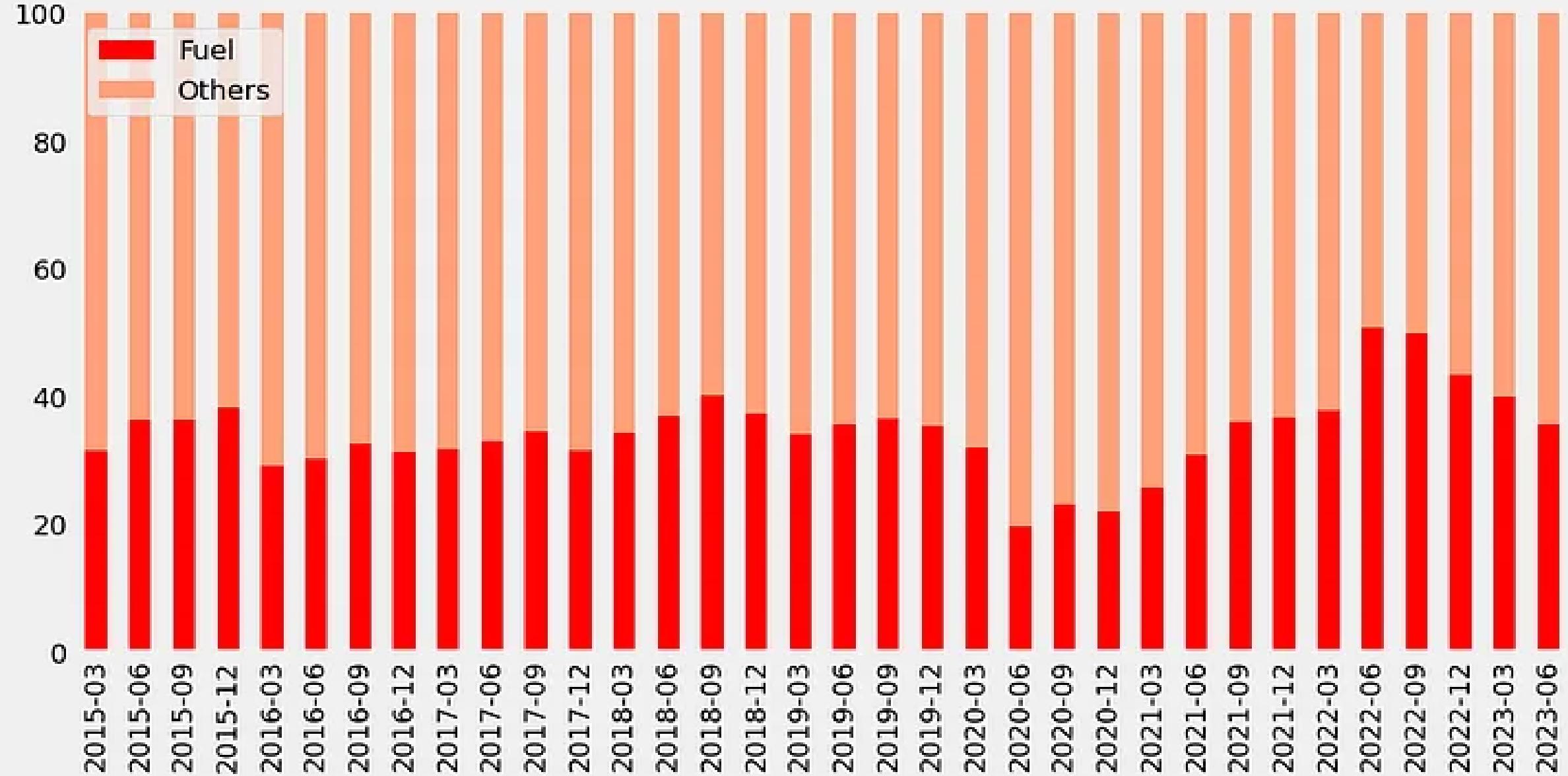
Teknik + Diğer Hasılatlar: 2,750

Hasılat: 137,485

```
forecasted_passenger_cargo = forecasted_passenger + forecasted_cargo  
forecasted_technical_other = (forecasted_passenger_cargo*100/98) - forecasted_passenger_cargo  
forecasted_revenue = forecasted_passenger_cargo + forecasted_technical_other  
  
print("Forecasted Revenue (Million TL):", forecasted_revenue)
```

	1 Ocak - 30 Haziran 2023	1 Nisan - 30 Haziran 2023	1 Ocak - 30 Haziran 2022	1 Nisan - 30 Haziran 2022
Akaryakıt	56.413	28.584	42.235	28.844
Personel	24.267	13.356	10.380	5.572
Amortisman ve itfa payı	18.975	9.943	12.843	6.877
Yer hizmetleri	12.114	7.168	5.973	3.431
Bakım	9.461	4.906	5.517	3.106
Konma ve konaklama	8.501	4.698	4.456	2.584
Yolcu hizmet ve ikram	8.287	4.771	3.708	2.354
Üst geçiş	6.874	3.865	4.141	2.393
Kısa dönemli uçak kirası	1.703	1.144	989	599
Hizmet	1.078	720	298	168
Sigorta	574	296	426	228
Nakliye	541	287	421	234
Aydınlatma, ısıtma, enerji ve su	282	100	201	103
Vergi	255	139	136	71
Kira	207	119	252	139
Uçak kira	158	70	102	77
Haberleşme ve bilişim	73	23	49	26
Sistem kullanım üyelik	64	34	28	18
Diğer	395	160	179	84
	150.222	80.383	92.334	56.908

Turkish Airlines Quarterly Cost of Sales by Segment (%)



Tarihsel ortalama: 35%

Turkish Airlines Quarterly Revenue and Cost of Sales



Satışlarının Maliyeti Modeli

```
cost_model_df = master_df[['Date', 'Revenue', 'Cost']]
cost_model_df['Date'] = pd.to_datetime(cost_model_df['Date'])
cost_model_df['Log_Revenue'] = np.log(cost_model_df['Revenue'])
cost_model_df['Log_Cost'] = np.log(abs(cost_model_df['Cost']))

cost_model_df['Log_Revenue_L1'] = cost_model_df['Log_Revenue'].shift(1)
cost_model_df['Log_Revenue_L2'] = cost_model_df['Log_Revenue'].shift(2)
cost_model_df['Log_Revenue_L3'] = cost_model_df['Log_Revenue'].shift(3)
cost_model_df['Log_Revenue_L4'] = cost_model_df['Log_Revenue'].shift(4)

cost_model_df['Log_Cost_L1'] = cost_model_df['Log_Cost'].shift(1)
cost_model_df['Log_Cost_L2'] = cost_model_df['Log_Cost'].shift(2)
cost_model_df['Log_Cost_L3'] = cost_model_df['Log_Cost'].shift(3)
cost_model_df['Log_Cost_L4'] = cost_model_df['Log_Cost'].shift(4)

cost_model_df = cost_model_df.dropna()
```

```
X = cost_model_df[[  
    'Log_Revenue',  
    'Log_Revenue_L4',  
    'Log_Cost_L2'  
]]  
y = cost_model_df['Log_Cost']  
X = sm.add_constant(X)  
model = sm.OLS(y, X).fit()  
print(model.summary())
```

Satışların maliyeti modelini hangi değişkenler ile açıklıyoruz?

- Logaritmik hasılat,
- Logaritmik hasılatın dört dönem gecikmesi,
- Logaritmik satışların maliyetinin iki dönem gecikmesi.

OLS Regression Results

Dep. Variable:	Log_Cost	R-squared:	0.993
Model:	OLS	Adj. R-squared:	0.992
Method:	Least Squares	F-statistic:	1199.
Date:	Mon, 30 Oct 2023	Prob (F-statistic):	5.57e-28
Time:	00:03:55	Log-Likelihood:	40.125
No. Observations:	30	AIC:	-72.25
Df Residuals:	26	BIC:	-66.65
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P> t	[0.025	0.975]
const	0.8192	0.195	4.205	0.000	0.419	1.220
Log_Revenue	0.6922	0.027	25.421	0.000	0.636	0.748
Log_Revenue_L4	-0.3106	0.046	-6.731	0.000	-0.406	-0.216
Log_Cost_L2	0.5275	0.051	10.302	0.000	0.422	0.633

Omnibus:	6.698	Durbin-Watson:	2.311
Prob(Omnibus):	0.035	Jarque-Bera (JB):	5.187
Skew:	-0.982	Prob(JB):	0.0747
Kurtosis:	3.542	Cond. No.	263.

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

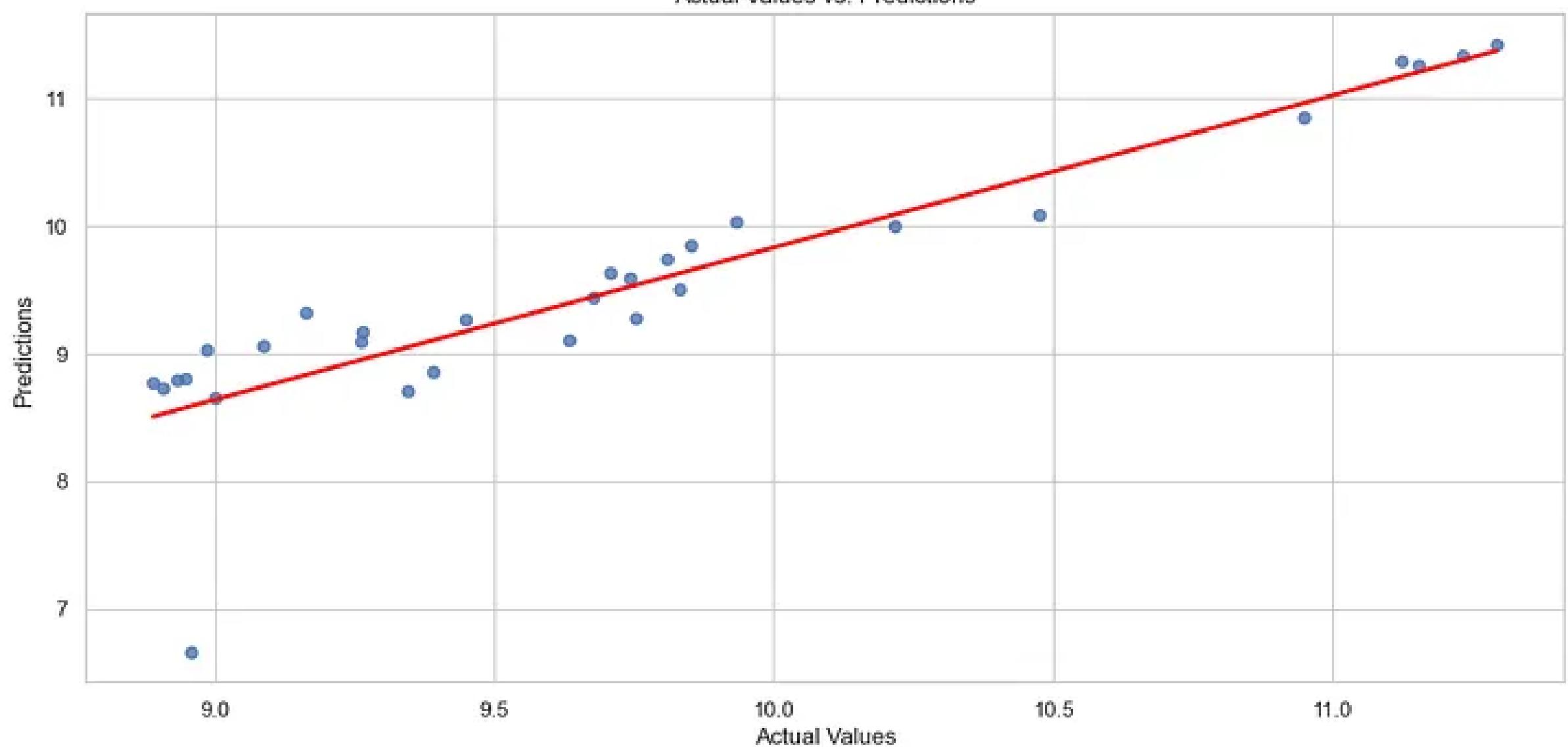
Gerçek ve Tahmin Edilen Değerler (Doğrusallık)

```
residuals = model.resid
fitted_values = model.fittedvalues

predictions = model.predict()

plt.figure(figsize=(12, 6))
sns.regplot(x=y, y=predictions, ci=None, line_kws={"color": "red"})
plt.xlabel("Actual Values")
plt.ylabel("Predictions")
plt.title("Actual Values vs. Predictions")
plt.show()
```

Actual Values vs. Predictions

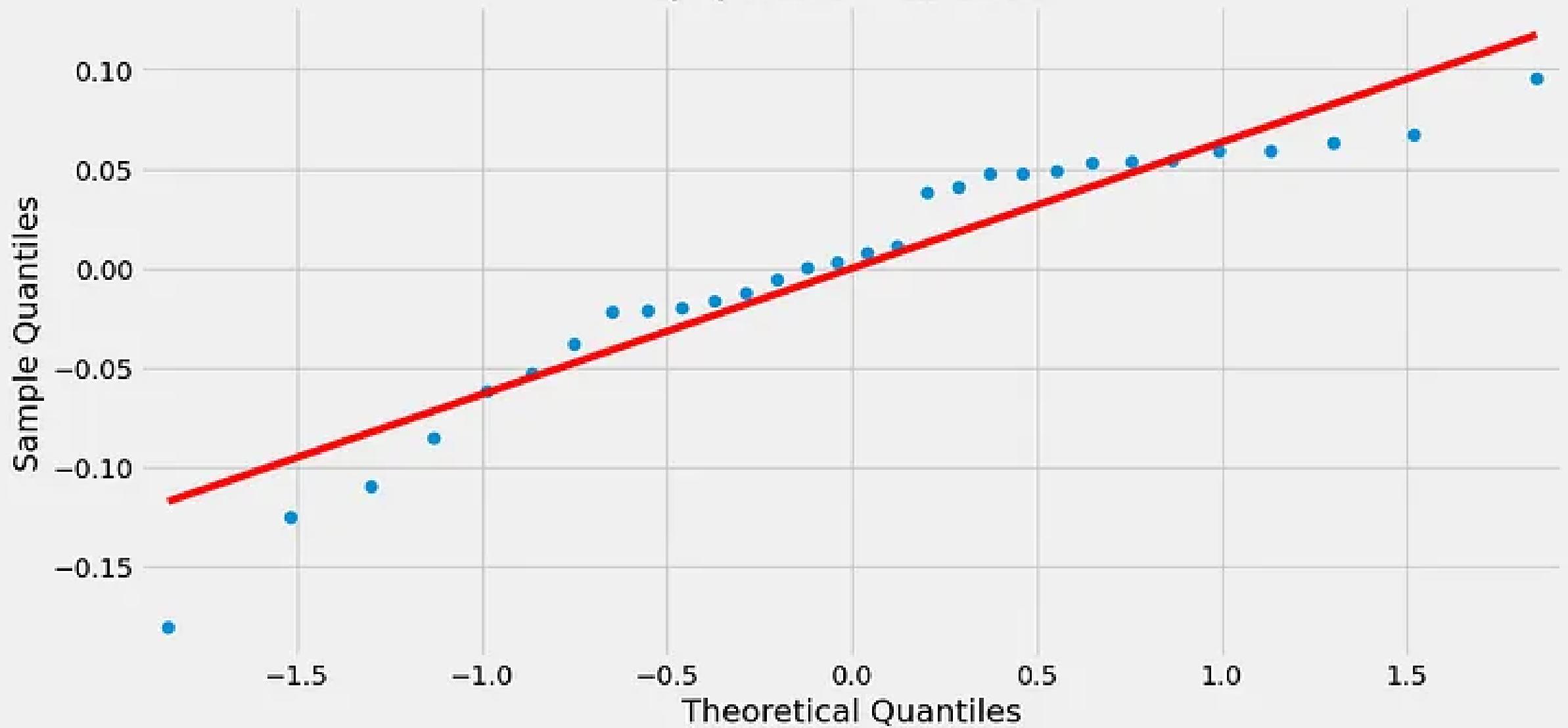


Normallik / Shapiro-Wilk Testi

Örneklem büyüklüğü nispeten küçükse oldukça önemlidir. Çünkü t ve F gibi anlamlılık testleri normallik varsayıımına dayanır.

```
fig, ax = plt.subplots(figsize=(12, 6))
sm.qqplot(residuals, line='s', ax=ax)
ax.set_title("Q-Q Plot of Residuals")
ax.set_xlabel("Theoretical Quantiles")
ax.set_ylabel("Sample Quantiles")
plt.show()
```

Q-Q Plot of Residuals



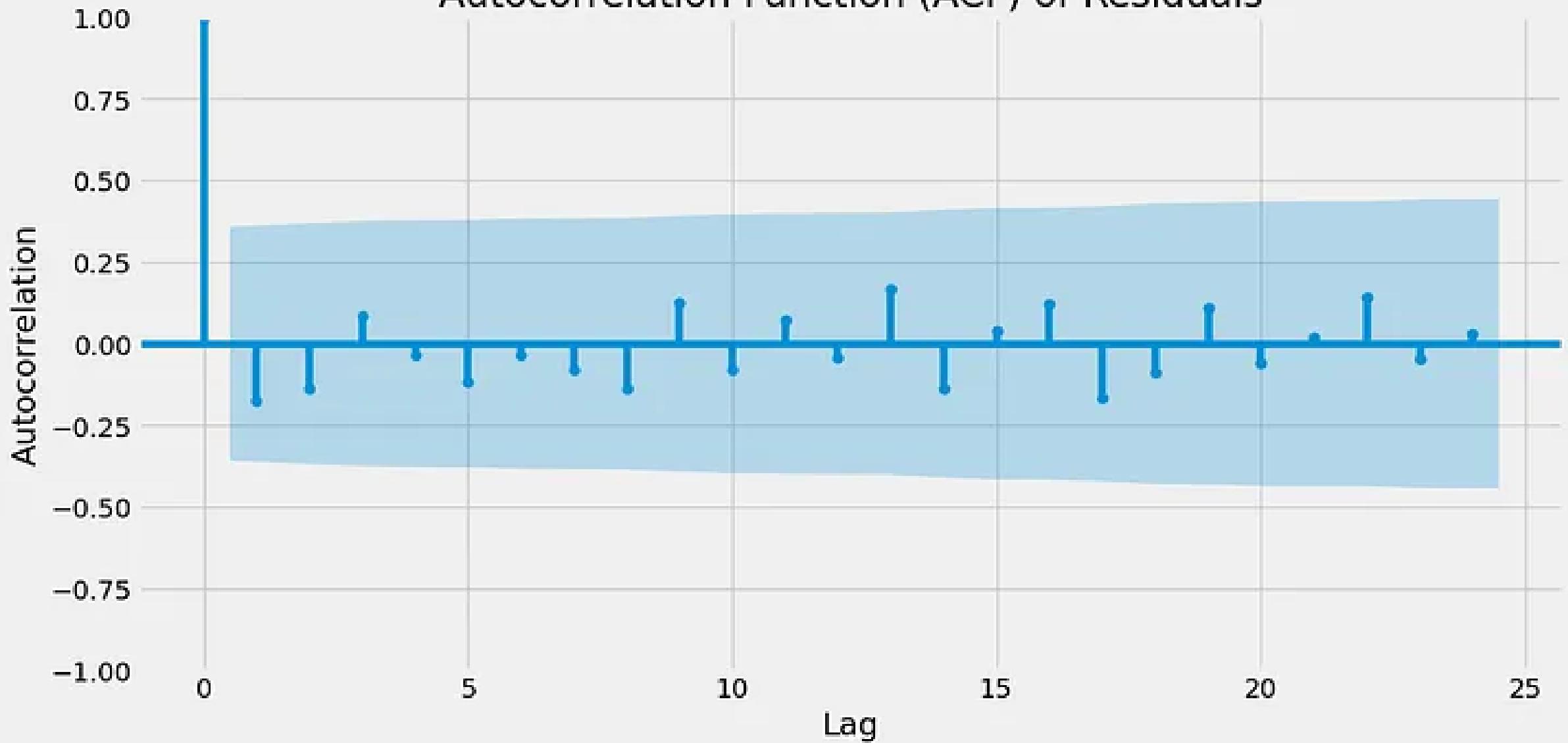
```
alpha = 0.01

sw_statistic, sw_p_value = shapiro(residuals)
if sw_p_value > alpha:
    print("The error terms follow a normal distribution.")
else:
    print("The error terms do not follow a normal distribution.")
```

Otokorelasyon / Breusch-Godfrey Testi

```
fig, ax = plt.subplots(figsize=(12, 6))
plot_acf(residuals, lags=24, ax=ax)
plt.title('Autocorrelation Function (ACF) of Residuals')
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.show()
```

Autocorrelation Function (ACF) of Residuals

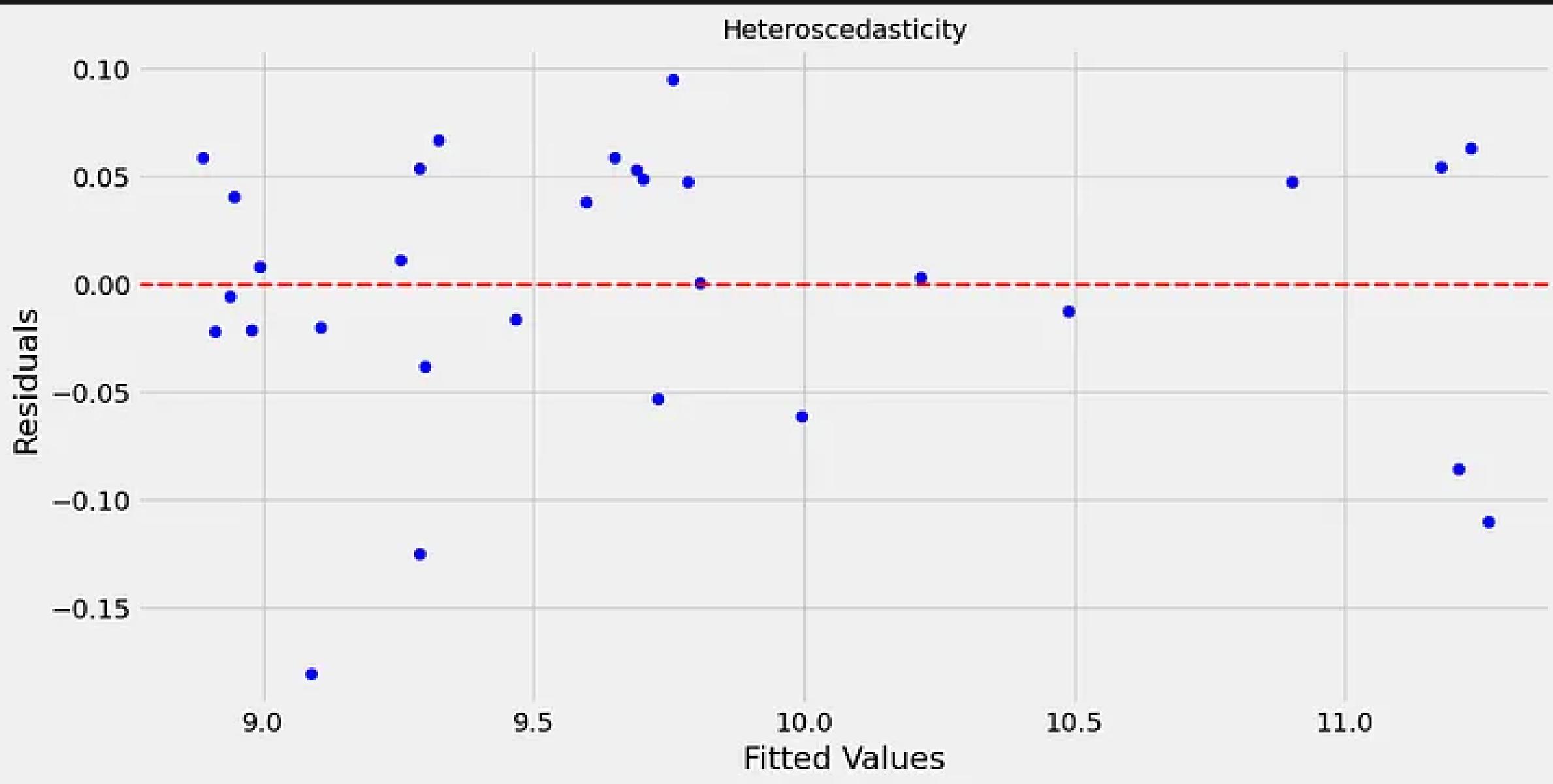


```
alpha = 0.01

bg_p_value = dg.acorr_breusch_godfrey(model, nlags=8)[1]
if bg_p_value > alpha:
    print("There is no autocorrelation among the error terms.")
else:
    print("There is autocorrelation among the error terms.")
```

Değişen Varyans / Breusch-Pagan Testi

```
plt.figure(figsize=(12, 6))
plt.scatter(fitted_values, residuals, color='blue')
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title('Heteroscedasticity', fontsize=14)
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()
```



```
X = model.model.exog
bp_test_result = het_breuschpagan(residuals, X)

if bp_test_result[1] < alpha:
    print("Heteroskedasticity is present in the model.")
else:
    print("No evidence of heteroskedasticity in the model.")
```

Satışlarının Maliyeti - Öngörü

```
input_data = {
    'Log_Revenue': np.log(forecasted_revenue[0]),
    'Log_Revenue_L4': 11.594846,
    'Log_Cost_L2': 11.153948
}

X = [1]
X += [
    input_data['Log_Revenue'],
    input_data['Log_Revenue_L4'],
    input_data['Log_Cost_L2']
]

forecasted_log_cost = model.predict(X)
forecasted_cost = np.exp(forecasted_log_cost)

print("Forecasted Cost of Sales (Million TL):", forecasted_cost)
```

Brüt Kar - Öngörü

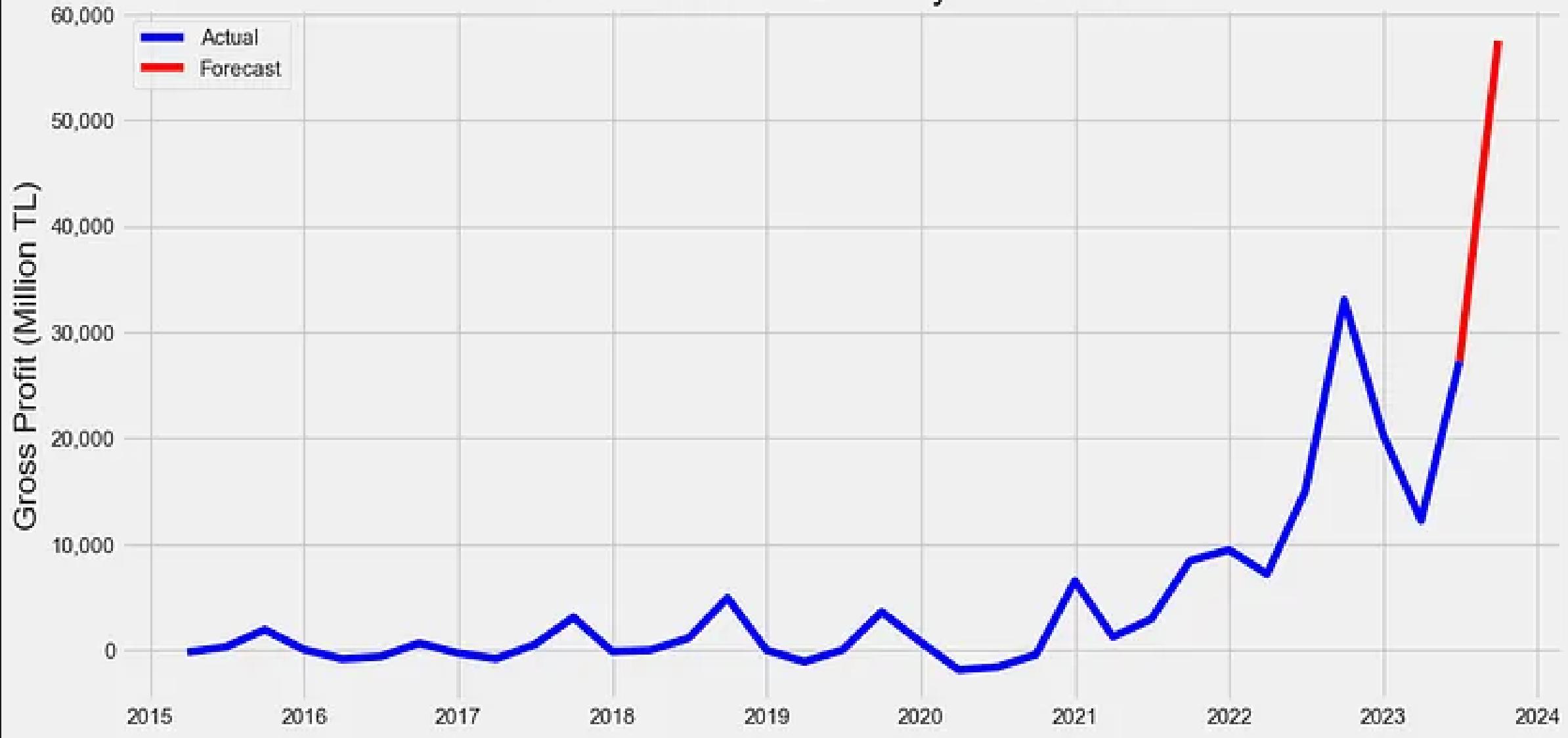
```
print("Forecasted Gross Profit (Million TL):", forecasted_revenue - forecasted_cost)
```

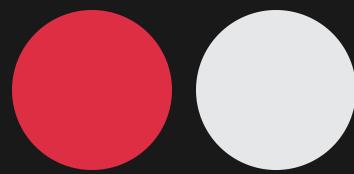
```
forecasted_df = master_df[['Date', 'Gross Profit']]
forecasted_df['Type'] = 'Actual'
new_row = {'Date': '2023-09-30 00:00:00', 'Gross Profit': forecasted_revenue[0] - forecasted_cost[0], 'Type': 'Forecast'}
forecasted_df = forecasted_df.append(new_row, ignore_index=True)
forecasted_df['Date'] = pd.to_datetime(forecasted_df['Date'])

actual_data = forecasted_df[forecasted_df['Date'] <= '2023-06-30']
forecast_data = forecasted_df[forecasted_df['Date'] >= '2023-06-30']
```

```
plt.figure(figsize=(12, 6))
plt.plot(actual_data['Date'], actual_data['Gross Profit'], label='Actual', color='blue')
plt.plot(forecast_data['Date'], forecast_data['Gross Profit'], label='Forecast', color='red')
plt.ylabel('Gross Profit (Million TL)')
plt.title('Turkish Airlines Quarterly Gross Profit')
plt.legend()
plt.gca().yaxis.set_major_formatter(mticker.StrMethodFormatter('{x:,.0f}'))
plt.show()
```

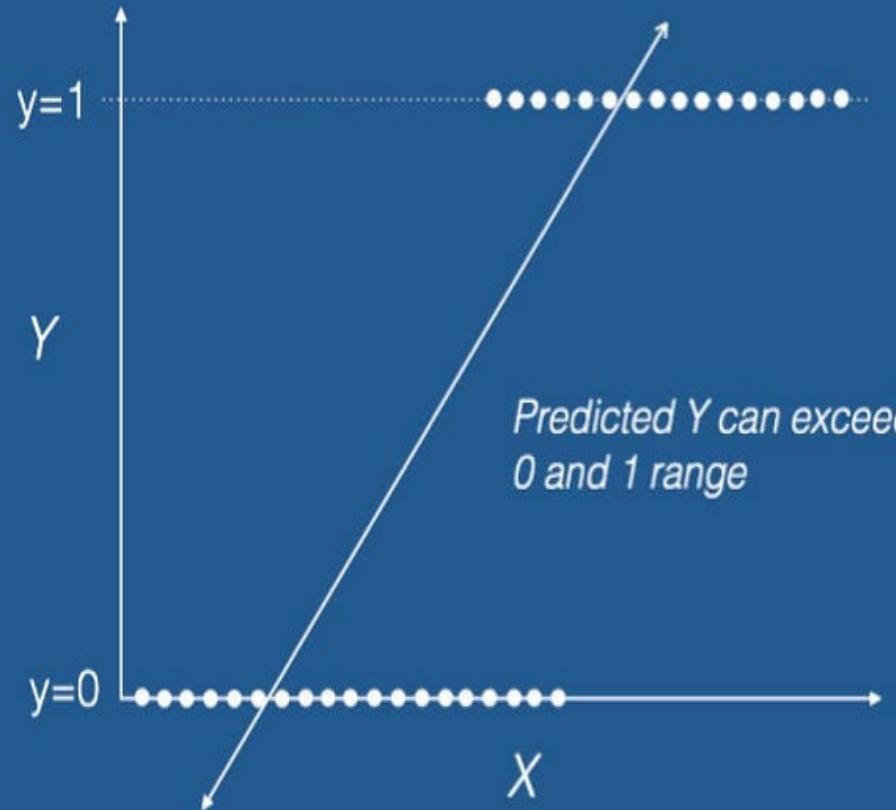
Turkish Airlines Quarterly Gross Profit





Lojistik Regresyon ile Sınıflandırma

Linear Regression



Logistic Regression

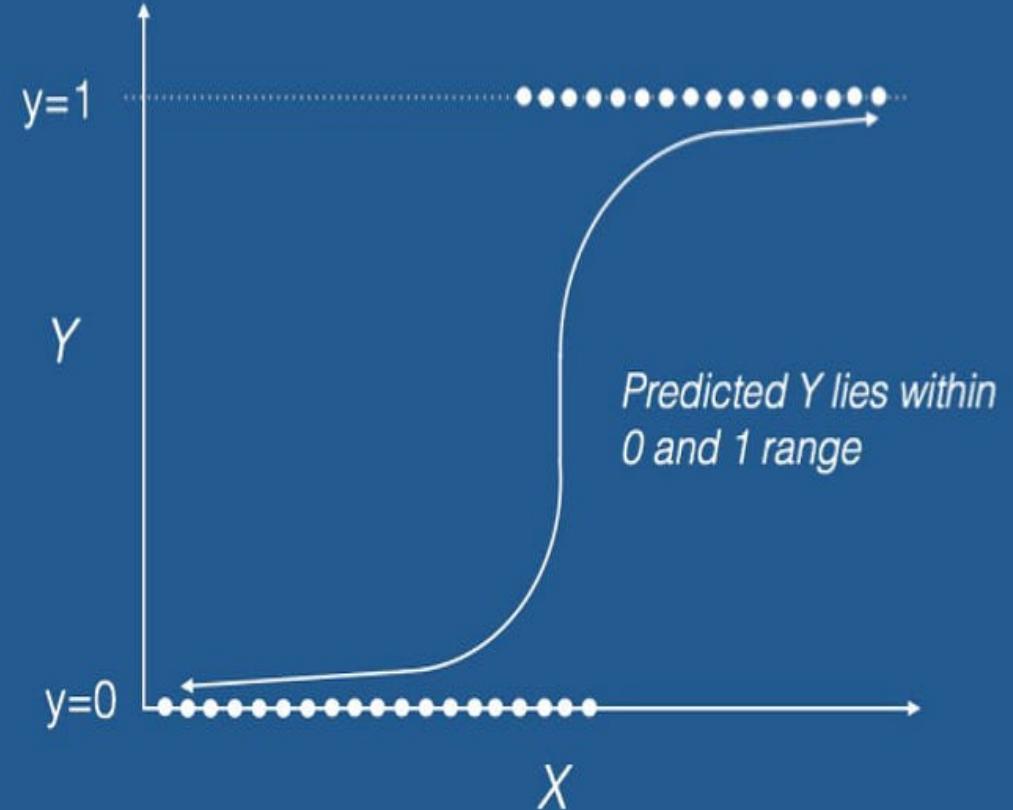


Image: DataCamp

Neden Doğrusal Olasılık Modeli (LPM) Kullanmıyoruz?

1. LPM, bağımlı değişkenin olasılığının bağımsız değişkenlerle doğrusal hareket ettiğini varsayar.
2. Olasılık değerleri 0-1 arasında yer alır ancak LPM bu aralıkta olacağının sözünü vermez.
3. Bağımlı değişken 0-1'lerden oluşuyorsa hata teriminin normalliği varsayıımı geçerliliğini kaybeder.
4. Hata terimi değişen varyanslı olur.

Ne bekliyoruz?

1. Tahmin edilen olasılık daima 0-1 arasında yer almmalıdır.
2. İlişki doğrusal değildir. X büyündükçe P daha düşük hızda 1'e; X küçüldükçe P daha düşük hızda 0'a yaklaşmalıdır.

Türk Hava Yolları Müşterilerine* Ait Tavsiye Değerlendirmelerinin Sınıflandırılması

*[Skytrax/Airline Quality](#)

12. Value for money *



13. Ground service *



14. Seat comfort *



15. Cabin Staff service *



16. Food & beverages *



17. Inflight entertainment *



18. Cabin WiFi & connectivity *



19. Would you recommend this airline?

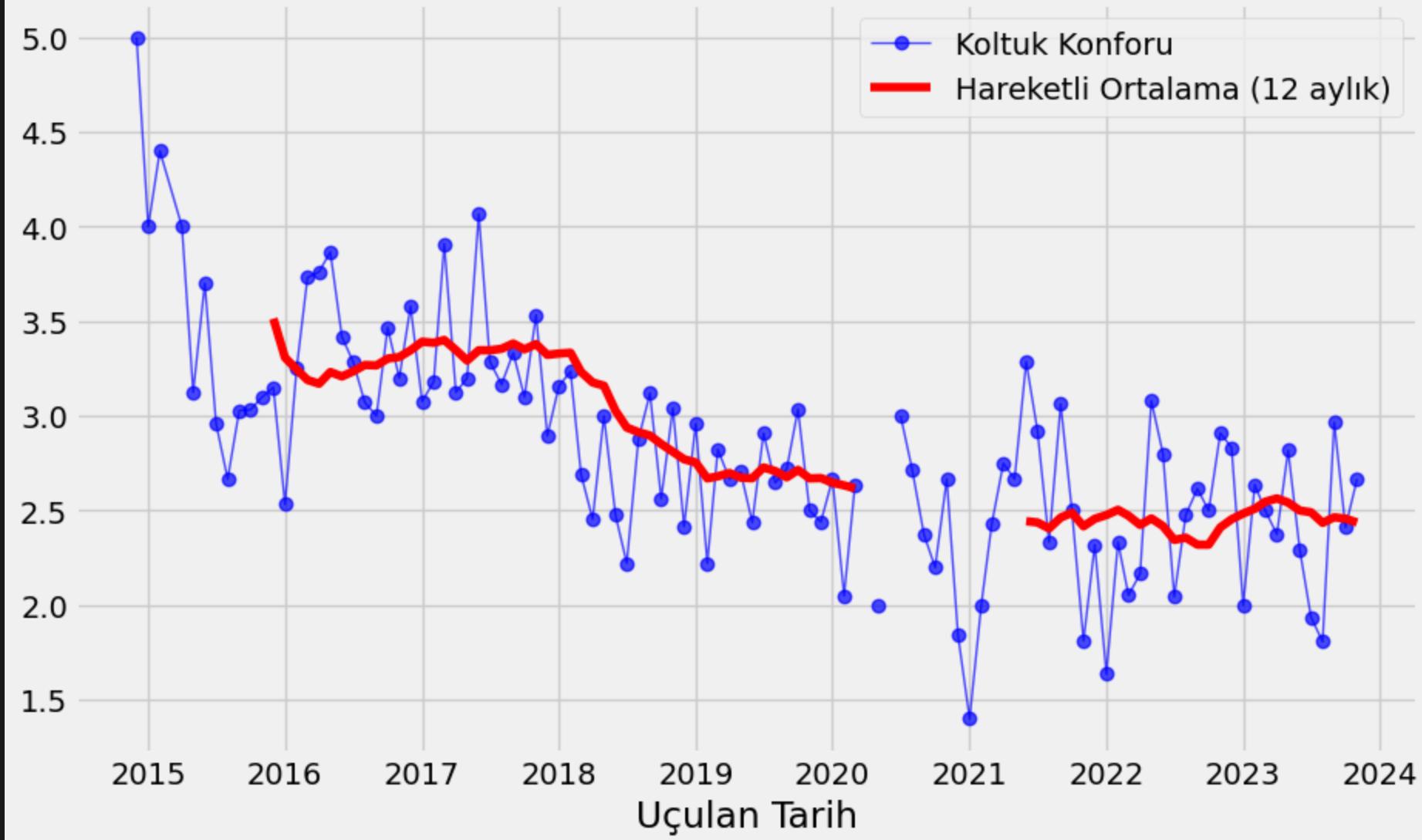
Yes No

Adım Adım Sınıflandırma

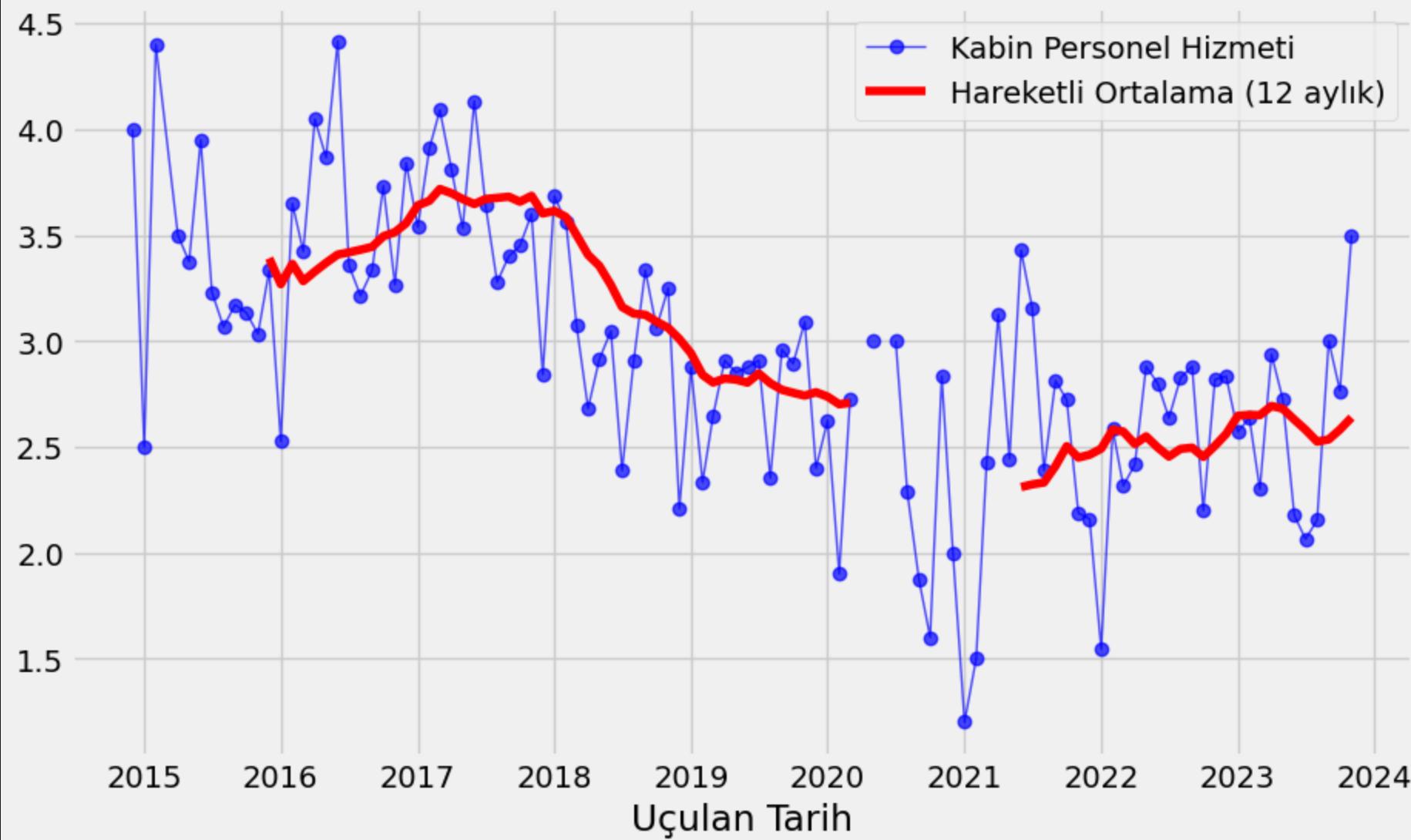
- Genel Bakış
- Sınıflandırma Hazırlığı
- Train-Test
- Confusion Matrix & Accuracy Skoru

Genel Bakış

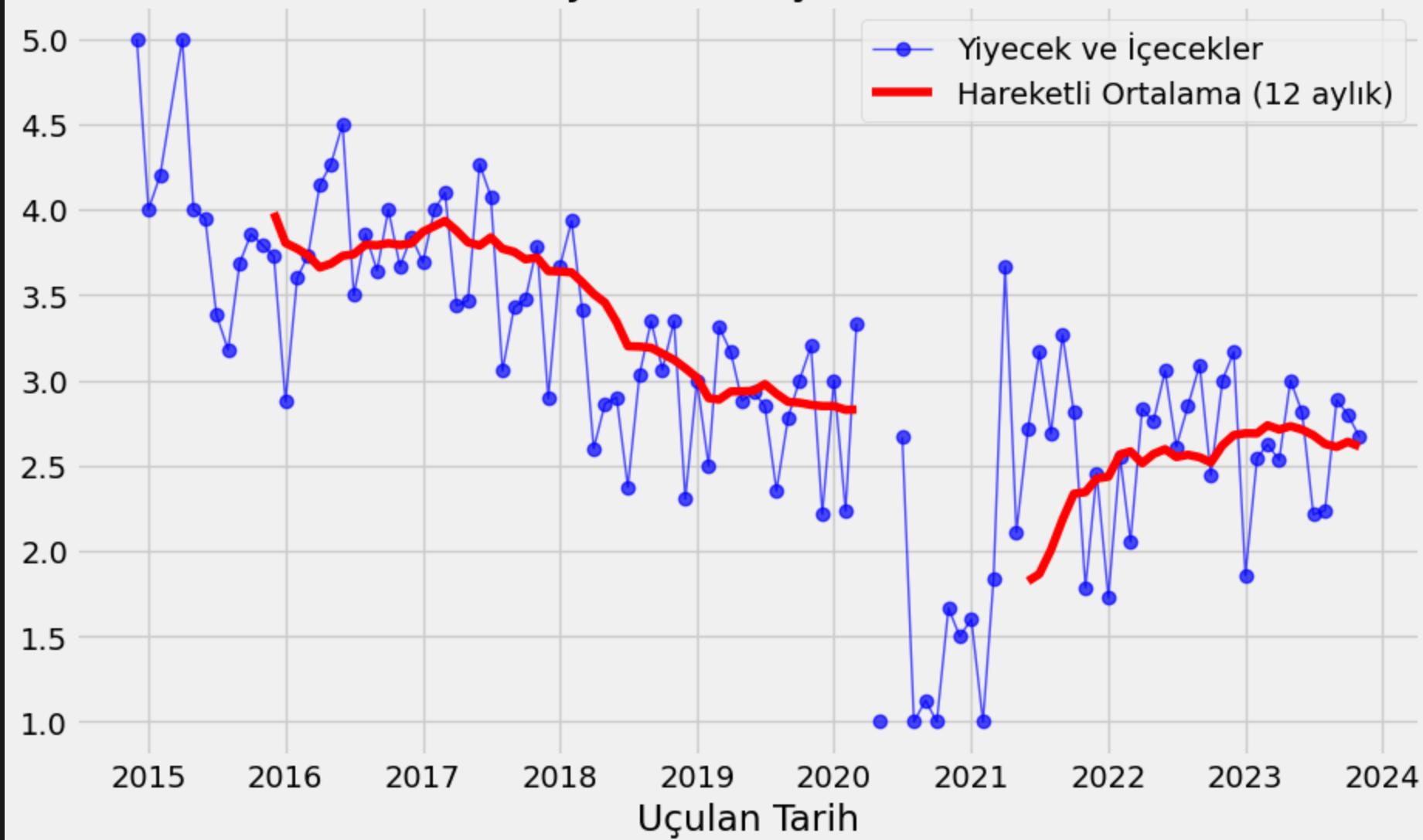
Koltuk Konforu



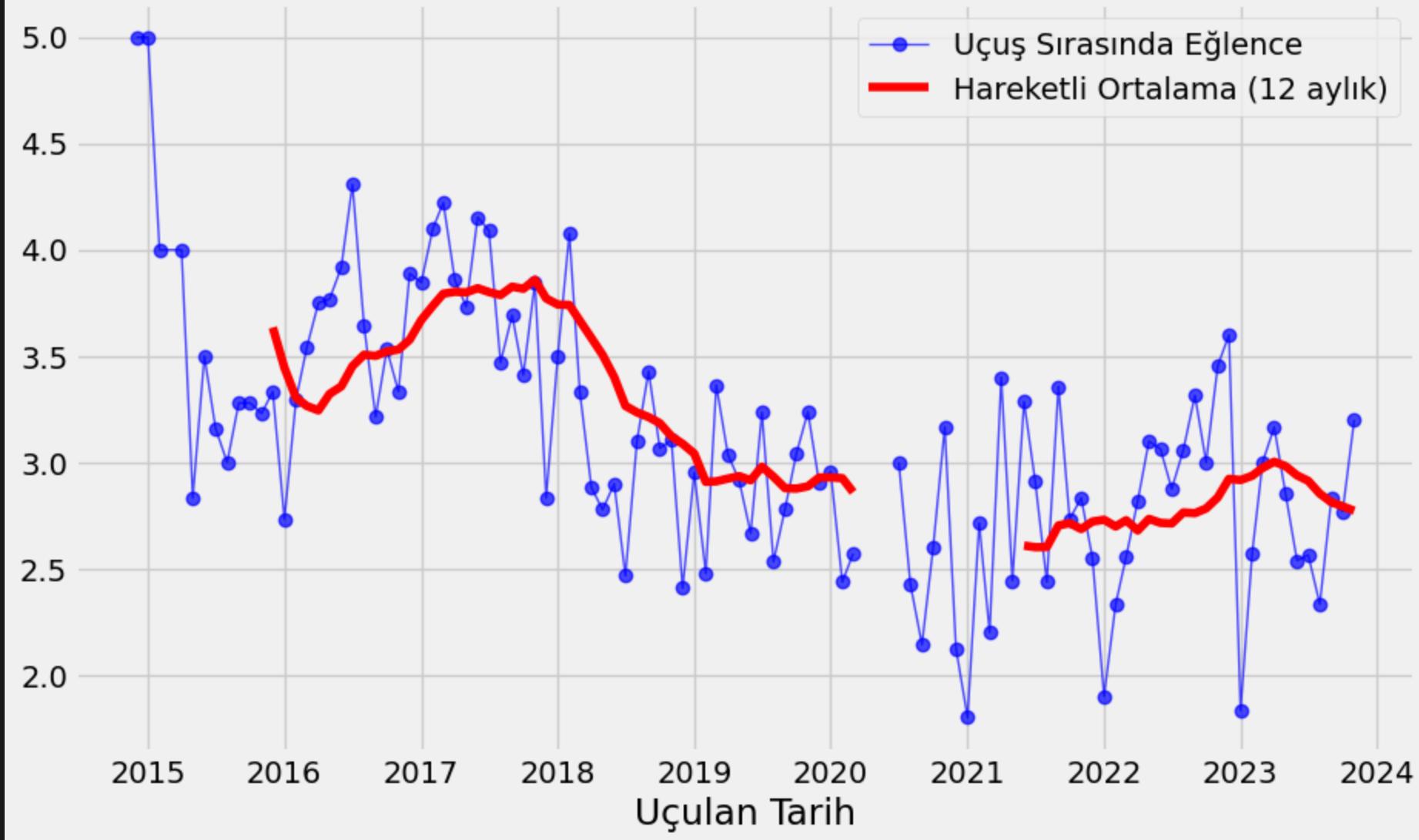
Kabin Personel Hizmeti



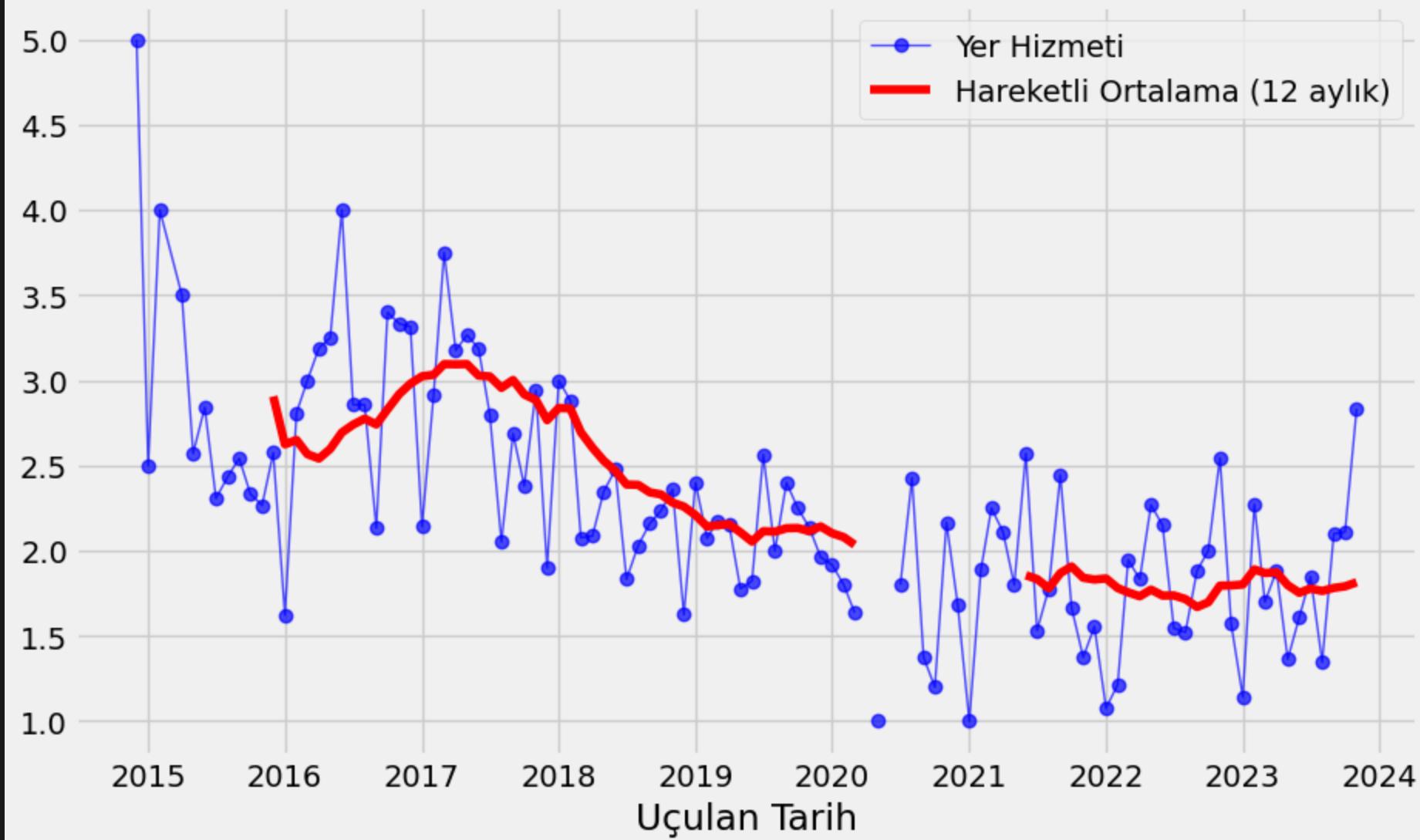
Yiyecek ve İçecekler



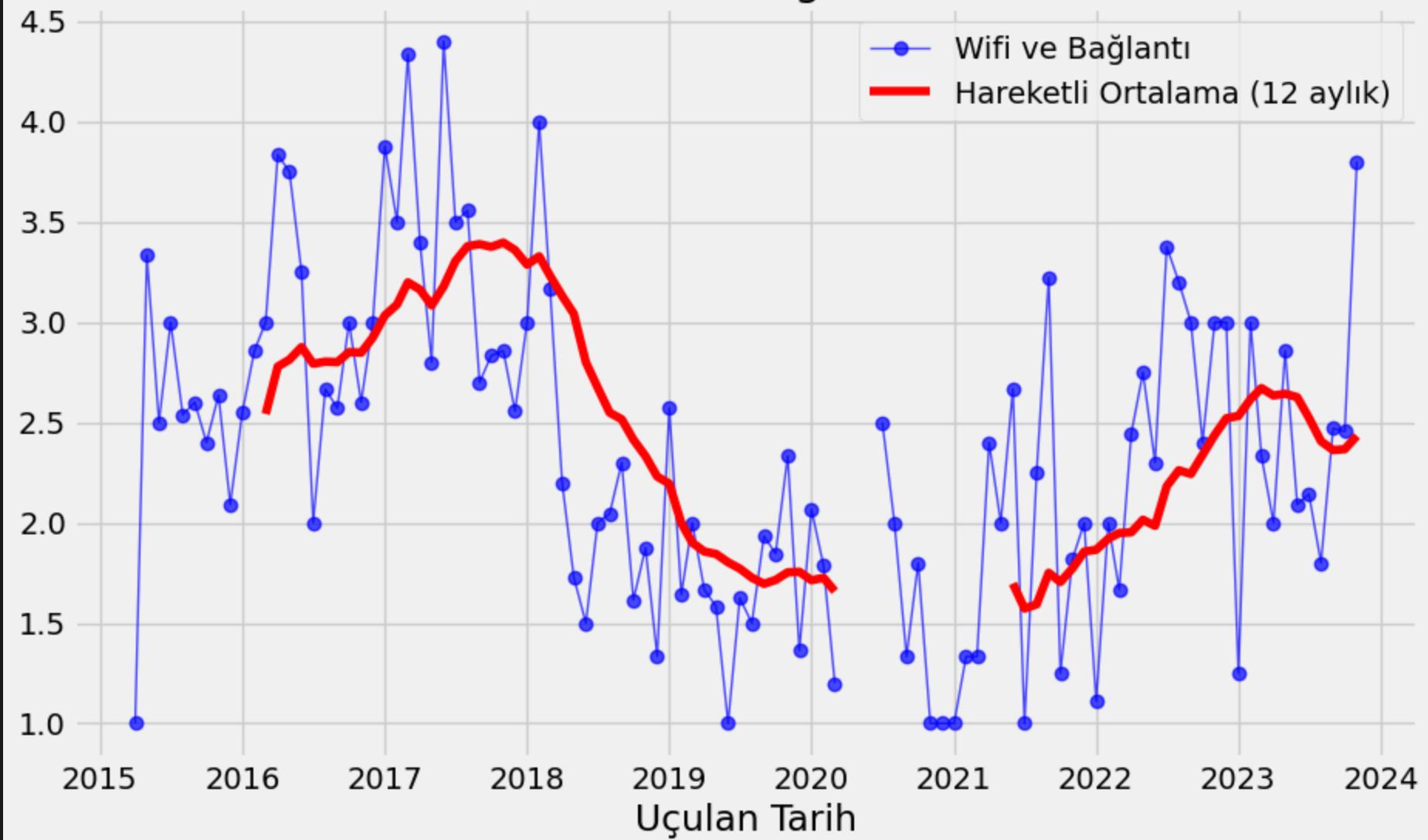
Uçuş Sırasında Eğlence



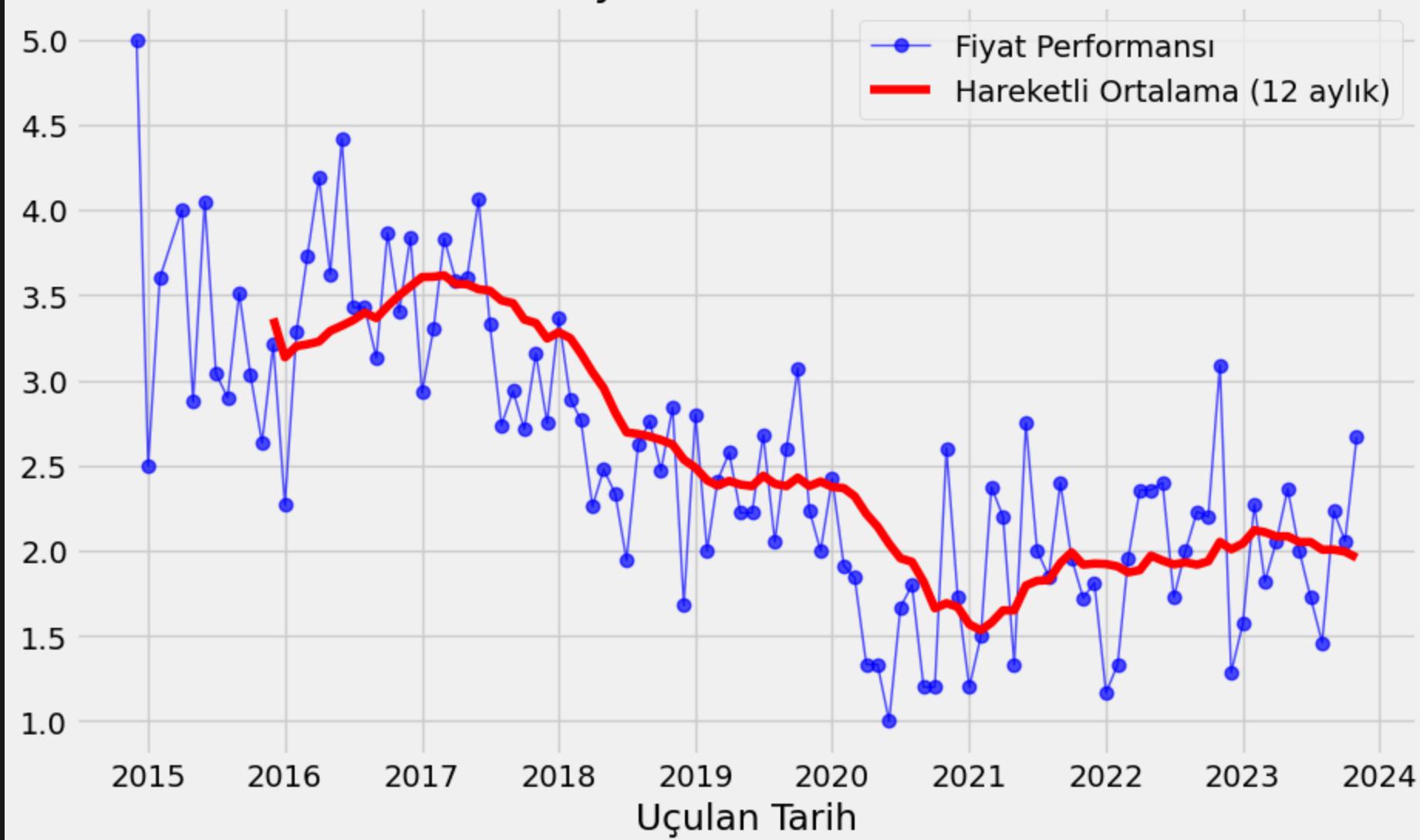
Yer Hizmeti



Wifi ve Bağlantı



Fiyat Performansı



Sınıflandırma Hazırlığı

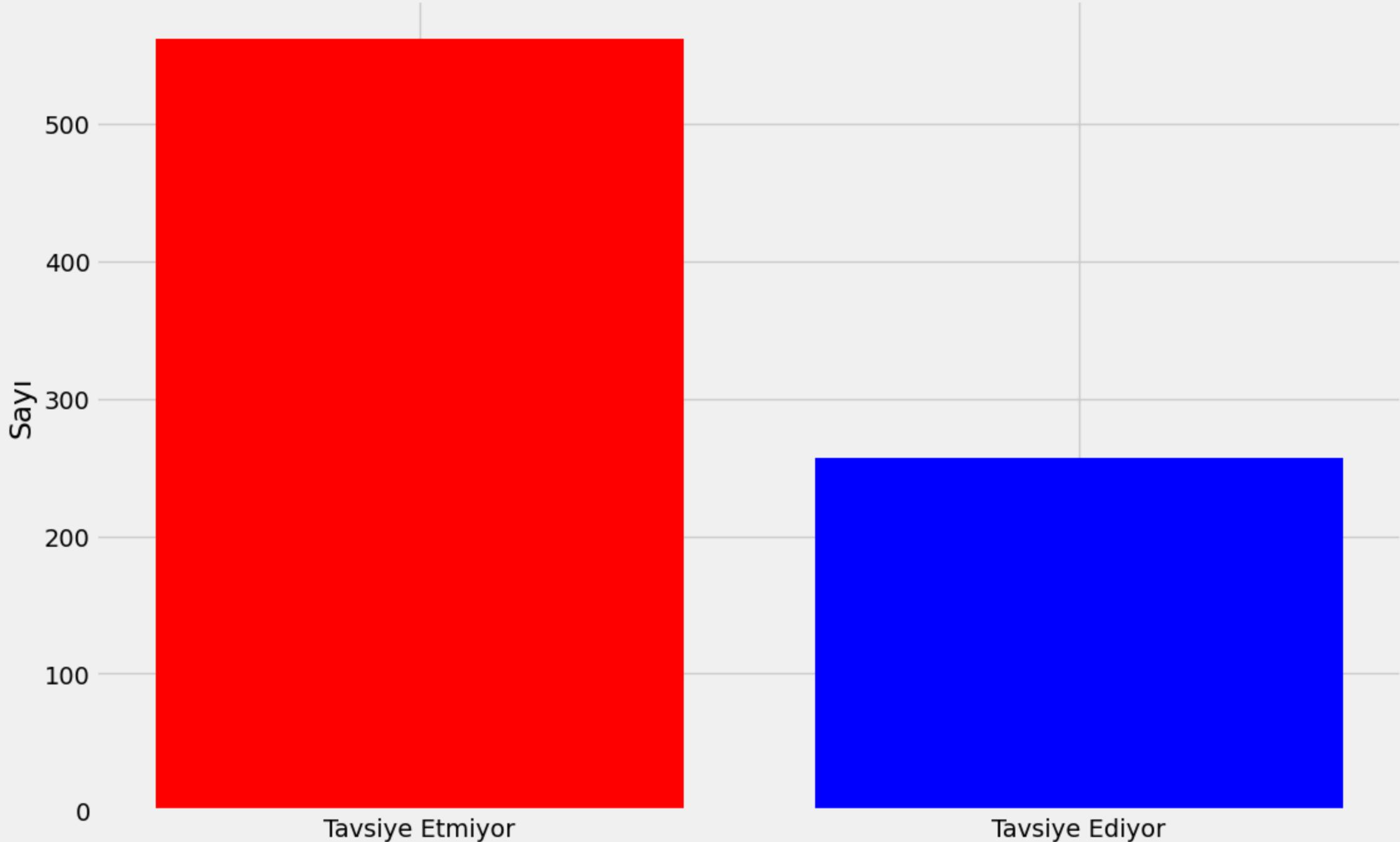
```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

```
df = pd.read_excel('./3-econometrics/model_data_filtered.xlsx')

recommendation_n = df['Tavsiye'].value_counts()

plt.figure(figsize=(12,8))
plt.bar(['Tavsiye Etmiyor', 'Tavsiye Ediyor'], recommendation_n, color=['red', 'blue'])
plt.ylabel('Sayı')
plt.title('Tavsiye Durumu Dağılımı')
plt.show()
```

Tavsiye Durumu Dağılımı



```
not_recommend_n = df['Tavsiye'].value_counts()[0]
recommend_n = df['Tavsiye'].value_counts()[1]

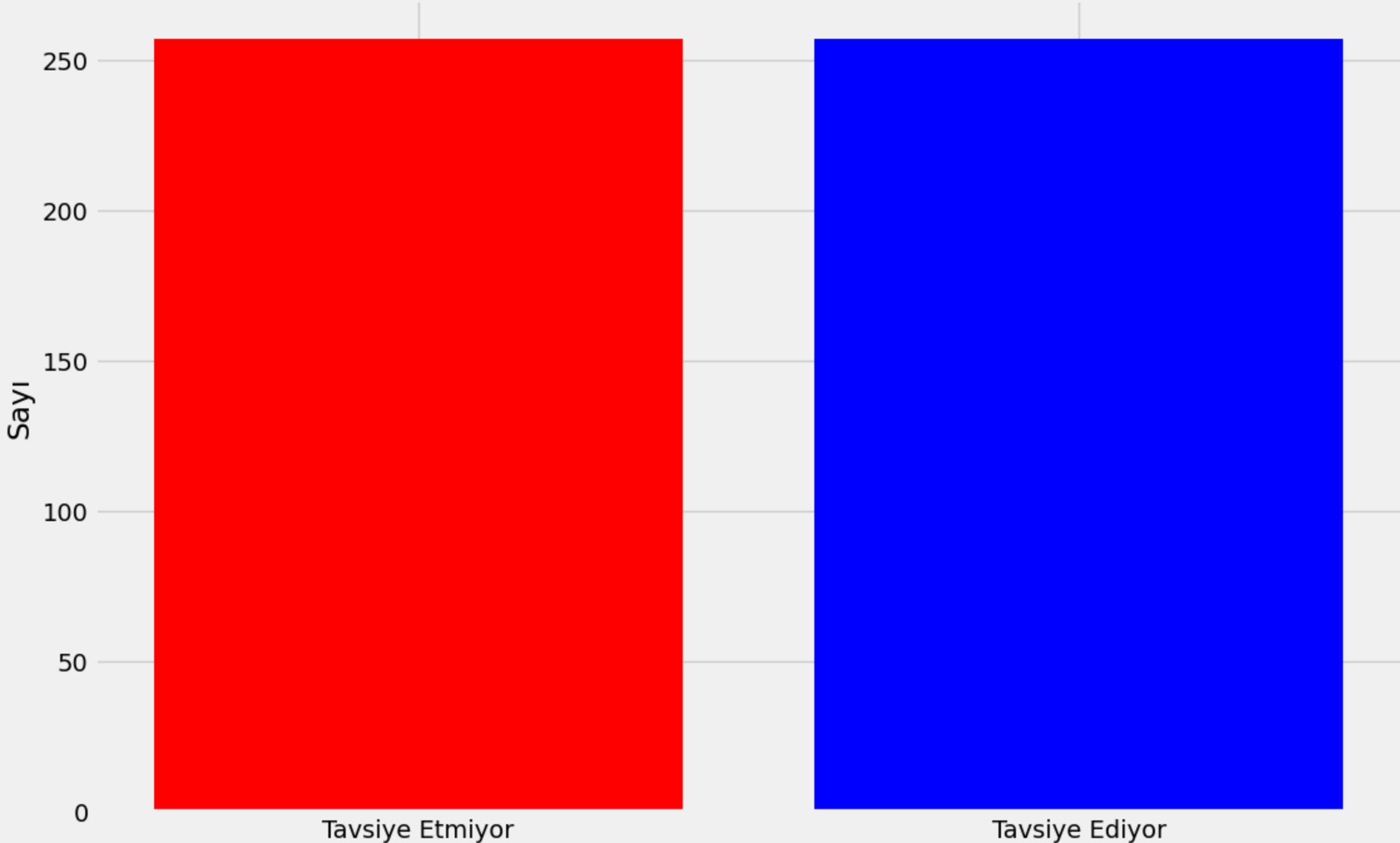
not_recommend_index = df[df['Tavsiye'] == 0].sample(n=min(257, not_recommend_n), random_state=123).index
recommend_index = df[df['Tavsiye'] == 1].index

balanced_df = pd.concat([df.loc[not_recommend_index], df.loc[recommend_index]])
balanced_df = balanced_df.sample(frac=1, random_state=123).reset_index(drop=True)

balanced_recommendation_n = balanced_df['Tavsiye'].value_counts()

plt.figure(figsize=(12,8))
plt.bar(['Tavsiye Etmiyor', 'Tavsiye Ediyor'], balanced_recommendation_n, color=['red', 'blue'])
plt.ylabel('Sayı')
plt.title('Tavsiye Durumu Dağılımı (Dengelenmiş)')
plt.show()
```

Tavsiye Durumu Dağılımı (Dengelenmiş)



Train-Test

Train (Eğitim):

- Modelin veri kümesi üzerinde öğrenmeye çalıştığı aşamadır.
- Model, eğitim veri kümesinde bulunan örnekler üzerinden öğrenir.
- Eğitim veri kümesi, genellikle modelin eğitmek ve belirli bir görevi yerine getirmesini sağlamak için kullanılan verileri içerir.
- Bu veriler genellikle etiketlenmiş verilerdir, yani girdi verileriyle ilişkilendirilmiş hedef çıktılar içerir.

Test:

- Modelin eğitim aşamasını tamamladıktan sonra, modelin gerçek dünya verileri üzerinde ne kadar iyi performans gösterdiğini değerlendirmek için test aşamasına geçilir.
- Test veri kümesi, modelin daha önce görmediği ve eğitim veri kümesinden farklı olan verileri içerir.
- Bu veri kümesi, modelin genelleme yeteneğini değerlendirmek için kullanılır.
- Model, test veri kümesindeki girişlere dayalı olarak tahminlerde bulunur ve bu tahminlerin doğruluğu değerlendirilir.

Bağımlı-Bağımsız Değişkenler:

```
X = balanced_df.drop('Tavsiye', axis=1)  
y = balanced_df['Tavsiye']
```

Train-Test Ayrımı (Random, 80-20%):

```
x_train, x_test, y_train, y_test = train_test_split(  
    X,  
    y,  
    test_size=0.2,  
    random_state=123  
)
```

Model Oluşturma:

```
model = LogisticRegression()  
model.fit(X_train, y_train)
```

Accuracy (Doğruluk):

```
accuracy = model.score(X_test, y_test)
print("Accuracy: {:.2f}%".format(accuracy * 100))
```

Confusion Matrix (Hata Matrisi):

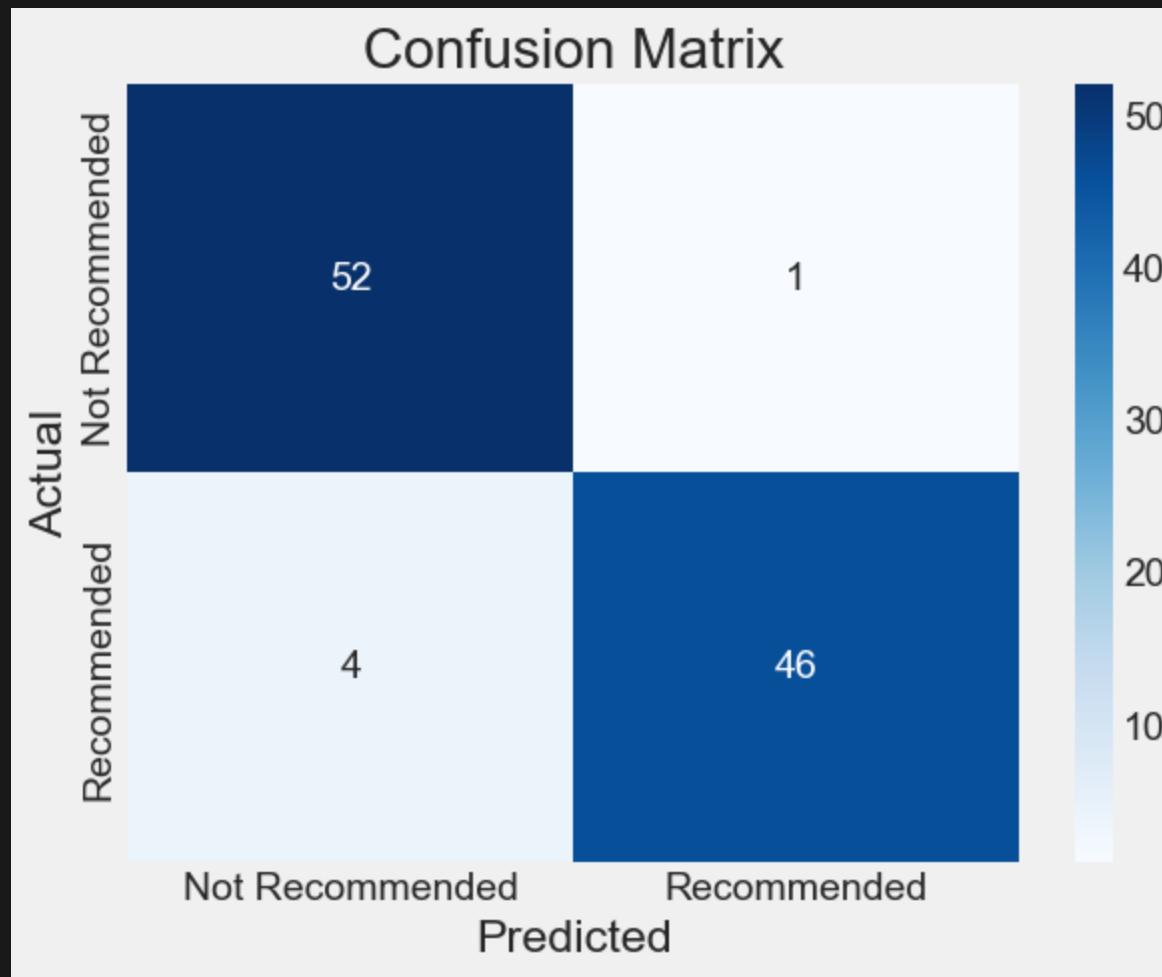
$$\text{Accuracy} = (\text{TP} + \text{TN}) / (\text{TP} + \text{TN} + \text{FP} + \text{FN})$$

		Actual Values	
		Positive	Negative
Predicted Values	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Image: Plat AI

```
y_pred = model.predict(x_test)  
cm = confusion_matrix(y_test, y_pred)
```

```
sns.heatmap(  
    cm, # Gösterilecek matris  
    annot=True, # Her hücredeki değerler gösterilsin  
    fmt="d", # Değerler tam sayı olarak gösterilsin  
    cmap="Blues", # Kullanılacak renk haritası mavinin tonları olsun  
    xticklabels=['Not Recommended', 'Recommended'], # X ekseni etiketleri  
    yticklabels=['Not Recommended', 'Recommended'] # Y ekseni etiketleri  
)  
plt.xlabel('Predicted')  
plt.ylabel('Actual')  
plt.title('Confusion Matrix')  
plt.show()
```



Aşağıdaki varsayımsal puanları veren bir müşteri Türk Hava Yolları'nı tavsiye eder mi?

Koltuk Konforu: 3/5

Kabin Personel Hizmeti: 3/5

Yiyecek ve İçecekler: 4/5

Uçuş Sırasında Eğlence: 4/5

Yer Hizmeti: 2/5

Wifi ve Bağlantı: 3/5

Fiyat Performansı: 4/5

Tavsiye: ?

Kullanıcıdan Girdilerin Alacak Fonksiyonun Yazılması:

```
def predict_recommendation(model):

    koltuk_konforu = int(input('Koltuk Konforu: '))
    kabin_personel = int(input('Kabin Personel Hizmeti: '))
    yiyecek_icecek = int(input('Yiyecek ve İçecekler: '))
    ucus_eglence = int(input('Uçuş Sırasında Eğlence: '))
    yer_hizmet = int(input('Yer Hizmeti: '))
    wifi_baglanti = int(input('Wifi ve Bağlantı: '))
    fiyat_performans = int(input('Fiyat Performansı: '))

    custom_data = {
        'Koltuk Konforu': [koltuk_konforu],
        'Kabin Personel Hizmeti': [kabin_personel],
        'Yiyecek ve İçecekler': [yiyecek_icecek],
        'Uçuş Sırasında Eğlence': [ucus_eglence],
        'Yer Hizmeti': [yer_hizmet],
        'Wifi ve Bağlantı': [wifi_baglanti],
        'Fiyat Performansı': [fiyat_performans]
    }
    custom_df = pd.DataFrame(custom_data)

    prediction = model.predict(custom_df)

    return prediction[0]
```

Kullanıcıdan Girdilerin Alınması ve Sonucun Yazdırılması:

```
result = predict_recommendation(model)

if result == 1:
    print("Tavsiye ediyor.")
else:
    print("Tavsiye etmiyor.")
```



Web Uygulamasi

- Streamlit
- Ekranlar:
 - About
 - Historical Stock Prices
 - Performance
 - Scorecard
 - Traffic
 - Fleet
 - Destinations
 - Competitors
 - Bankruptcy Risk

Streamlit

- Hızlı bir şekilde web uygulaması oluştur ve paylaş.
- %100 Python.
- Front-end deneyimi aramaz.

<https://streamlit.io/>

```
pip install streamlit
```

```
# terminal  
streamlit run your_app_name.py
```

```
import streamlit as st
import yfinance as yf
import pandas as pd
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.graph_objects as go
import folium
from streamlit_folium import st_folium
```

```
st.set_page_config(layout='wide', initial_sidebar_state='expanded')
st.sidebar.image('./4-streamlit-web-app/imgs/thyao.png', use_column_width=True)
```

```
page = st.sidebar.radio("Go to:", [  
    "About",  
    "Historical Stock Prices",  
    "Performance",  
    "Scorecard",  
    "Traffic",  
    "Fleet",  
    "Destinations",  
    "Competitors",  
    "Bankruptcy Risk"  
])
```

About

Turkish Airlines Analysis Dashboard

Welcome to the Turkish Airlines Analysis Dashboard! This interactive web application provides insights into various aspects of Turkish Airlines, including historical stock prices, performance metrics, fleet information, destinations, competitors, and more.

Navigate through the sidebar to explore different sections of the dashboard and gain valuable insights into Turkish Airlines' financial performance, operational data, and market position.

Key Features:

- **Historical Stock Prices:** Visualize the historical stock prices of Turkish Airlines.
- **Performance Metrics:** Analyze the performance of Turkish Airlines and compare it with market indices.
- **Scorecard:** Explore key financial metrics and scorecard data.
- **Traffic Analysis:** Dive into traffic data, including passenger load factors.
- **Fleet Information:** Understand the distribution and age of Turkish Airlines' fleet.
- **Destinations:** Explore Turkish Airlines' flight destinations on an interactive map.
- **Competitor Rankings:** View the evolution of Turkish Airlines' rankings compared to competitors.
- **Bankruptcy Risk:** Assess the bankruptcy risk using the Airscore method.

How to Use:

Use the sidebar on the left to navigate between different sections. Adjust date ranges, select specific metrics, and explore the dynamic visualizations to gain a comprehensive understanding of Turkish Airlines.

Disclaimer:

This dashboard is for informational purposes only and does not constitute financial advice. All data presented here is based on publicly available information, and users are encouraged to conduct their own research and consult with financial professionals before making investment decisions.

```
if page == "About":
    st.title("Turkish Airlines Analysis Dashboard")

    st.write(
        "Welcome to the Turkish Airlines Analysis Dashboard! This interactive web application provides insights "
        "into various aspects of Turkish Airlines, including historical stock prices, performance metrics, fleet "
        "information, destinations, competitors, and more."
    )

    st.write(
        "Navigate through the sidebar to explore different sections of the dashboard and gain valuable insights into "
        "Turkish Airlines' financial performance, operational data, and market position."
    )

    st.header("Key Features:")
    st.markdown(
        "- **Historical Stock Prices:** Visualize the historical stock prices of Turkish Airlines."
        "\n- **Performance Metrics:** Analyze the performance of Turkish Airlines and compare it with market indices."
        "\n- **Scorecard:** Explore key financial metrics and scorecard data."
        "\n- **Traffic Analysis:** Dive into traffic data, including passenger load factors."
        "\n- **Fleet Information:** Understand the distribution and age of Turkish Airlines' fleet."
        "\n- **Destinations:** Explore Turkish Airlines' flight destinations on an interactive map."
        "\n- **Competitor Rankings:** View the evolution of Turkish Airlines' rankings compared to competitors."
        "\n- **Bankruptcy Risk:** Assess the bankruptcy risk using the Airscore method."
    )

    st.header("How to Use:")
    st.write(
        "Use the sidebar on the left to navigate between different sections. Adjust date ranges, select specific metrics, "
        "and explore the dynamic visualizations to gain a comprehensive understanding of Turkish Airlines."
    )

    st.header("Disclaimer:")
    st.write(
        "This dashboard is for informational purposes only and does not constitute financial advice. All data presented here "
        "is based on publicly available information, and users are encouraged to conduct their own research and consult with "
        "financial professionals before making investment decisions."
    )
```

Historical Stock Prices

X

Deploy :



Adj Close Over Time

Go to:

- About
- Historical Stock Prices
- Performance
- Scorecard
- Traffic
- Fleet
- Destinations
- Competitors
- Bankruptcy Risk

Start Date

2023/01/01

End Date

2023/11/12

Interval

Daily

Price Type

Adj Close



```
elif page == "Historical Stock Prices":\n\n    start_date = st.sidebar.date_input('Start Date', pd.to_datetime('2023-01-01'))\n    end_date = st.sidebar.date_input('End Date', pd.Timestamp.now())\n    interval_options = ['Daily', 'Weekly', 'Monthly']\n    selected_interval = st.sidebar.selectbox('Interval', interval_options, index=0)\n    interval_map = {'Daily': '1d', 'Weekly': '1wk', 'Monthly': '1mo'}\n    selected_interval = interval_map[selected_interval]\n    price_columns = ['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume']\n    selected_column = st.sidebar.selectbox('Price Type', price_columns, index=4)\n\n    historical_data = yf.download(\n        tickers='THYAO.IS',\n        start=start_date,\n        end=end_date,\n        interval=selected_interval,\n        progress=False\n    )\n\n    fig = px.line(historical_data, x=historical_data.index, y=selected_column, title=f'{selected_column} Over Time')\n    fig.update_xaxes(title_text="")\n    fig.update_yaxes(title_text=selected_column)\n    fig.update_layout(height=800)\n    st.plotly_chart(fig, use_container_width=True)
```

Performance



```
# Part-1
elif page == "Performance":

    start_date = st.sidebar.date_input('Start Date', pd.to_datetime('2022-01-01'))
    end_date = st.sidebar.date_input('End Date', pd.Timestamp.now())
    interval_options = ['Daily', 'Weekly', 'Monthly']
    selected_interval = st.sidebar.selectbox('Interval', interval_options, index=0)
    interval_map = {'Daily': '1d', 'Weekly': '1wk', 'Monthly': '1mo'}
    selected_interval = interval_map[selected_interval]
    xulas = [
        "THYAO.IS",
        "PGSUS.IS",
        "XU100.IS",
    ]
```

```
# Part-2
data = pd.DataFrame()
for symbol in xulas:
    symbol_data = yf.download(symbol, start=start_date, end=end_date, interval=selected_interval, progress=False)
    data[symbol] = symbol_data['Adj Close']

if selected_interval == '1d':
    annualization_factor = 252
elif selected_interval == '1wk':
    annualization_factor = 52
elif selected_interval == '1mo':
    annualization_factor = 12

returns = data.pct_change()
annualized_returns = (returns.mean() * annualization_factor) * 100
annualized_std_dev = (returns.std() * (annualization_factor ** 0.5)) * 100

performance_options = ["Return-Standard Deviation", "Index"]
selected_performance = st.sidebar.selectbox ("Performance Chart", performance_options)
```

```
# Part-3
if selected_performance == "Return-Standard Deviation":

    fig = px.scatter(x=annualized_std_dev, y=annualized_returns, text=returns.columns)
    fig.update_layout(
        xaxis_title="Annualized Standard Deviation",
        yaxis_title="Annualized Mean Return"
    )
    fig.update_traces(textposition='top center', showlegend=False, marker_size=30)
    fig.update_layout(width=1400, height=800)
    st.plotly_chart(fig, use_container_width=True)

elif selected_performance == "Index":

    cumulative_returns = ((returns + 1).cumprod() * 100)
    cumulative_returns = cumulative_returns.fillna(100)

    fig = px.line(cumulative_returns, x=cumulative_returns.index, y=cumulative_returns.columns, title='')
    fig.update_xaxes(showline=False, showgrid=False, zeroline=False, title_text='')
    fig.update_yaxes(title_text='Index')
    fig.update_layout(height=900, legend={'title_text':''})
    st.plotly_chart(fig, use_container_width=True)
```

Scorecard

X

Deploy :



Brüt Kar Marji

Go to:

- About
- Historical Stock Prices
- Performance
- Scorecard
- Traffic
- Fleet
- Destinations
- Competitors
- Bankruptcy Risk

Select Column to Display

Brüt Kar Marji



2014

2016

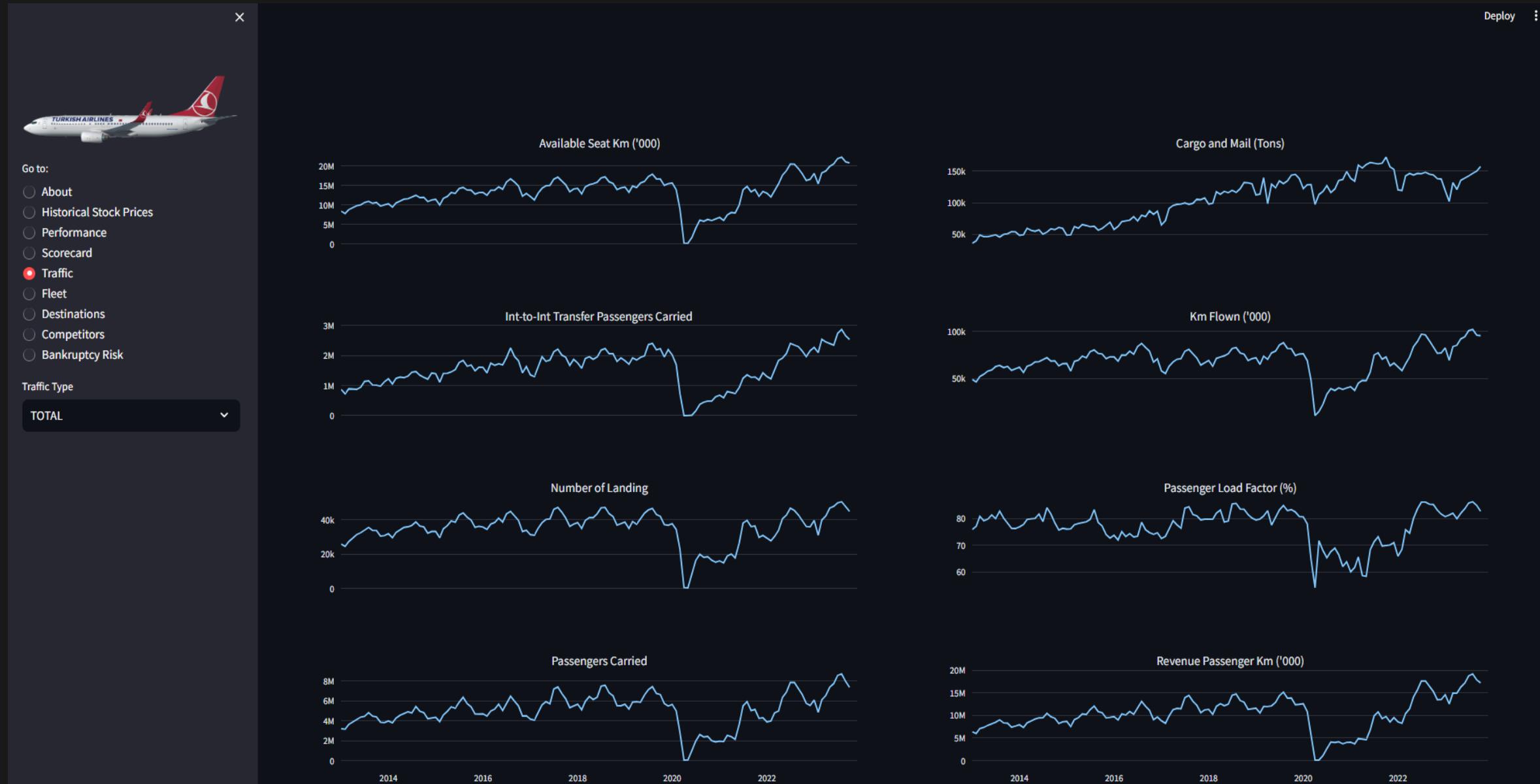
2018

2020

2022

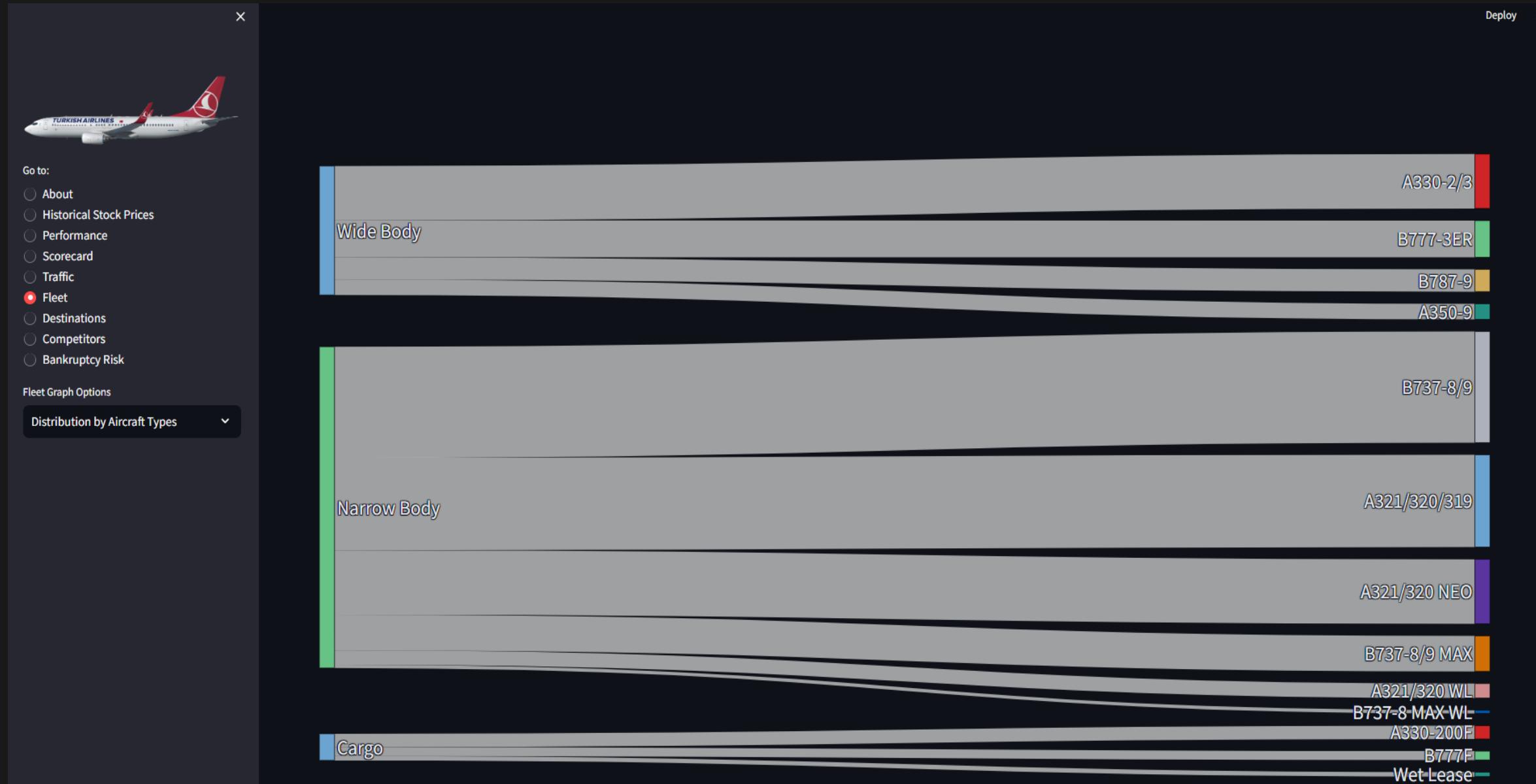
```
elif page == "Scorecard":  
  
    scorecard_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Scorecard')  
    columns = scorecard_df.columns[1:]  
    selected_column = st.sidebar.selectbox('Select Column to Display', columns)  
  
    fig = px.line(scorecard_df, x=scorecard_df['Date'], y=selected_column, title=selected_column)  
    fig.update_xaxes(title_text="")  
    fig.update_yaxes(title_text=selected_column)  
    fig.update_layout(height=900)  
    st.plotly_chart(fig, use_container_width=True)
```

Traffic



```
elif page == "Traffic":\n\n    traffic_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Traffic')\n\n    traffic_options = [\n        "TOTAL", "DOMESTIC", "INTERNATIONAL", "Europe", "Middle East", "Far East", "Africa", "North America", "Central & South America"\n    ]\n    selected_traffic_option = st.sidebar.selectbox("Traffic Type", traffic_options)\n\n    traffic_df = traffic_df[traffic_df['Category'] == selected_traffic_option]\n\n    date_column = traffic_df['Date']\n    data_columns = [col for col in traffic_df.columns if col != 'Date' and col != 'Category']\n\n    cols = 2\n    rows = len(data_columns) // cols + (len(data_columns) % cols > 0)\n    fig = make_subplots(rows=rows, cols=cols, shared_xaxes=True, subplot_titles=data_columns)\n\n    for i, column in enumerate(data_columns):\n        row = i // cols + 1\n        col = i % cols + 1\n\n        if column == "Passenger Load Factor (%):\n            traffic_df[column] = traffic_df[column] * 100\n\n        fig.add_trace(px.line(x=date_column, y=traffic_df[column], title=column).data[0], row=row, col=col)\n\n    fig.update_layout(height=950)\n    st.plotly_chart(fig, use_container_width=True)
```

Fleet



```
# Part-1
elif page == "Fleet":

    fleet_options = ["Distribution by Aircraft Types", "Number of Aircraft and Average Fleet Age"]
    selected_fleet_option = st.sidebar.selectbox("Fleet Graph Options", fleet_options)

    if selected_fleet_option == "Distribution by Aircraft Types":
        fleet_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Fleet')
        nodes = fleet_df['Category'].tolist() + fleet_df['Type'].tolist()
        connections = {
            "Wide Body": fleet_df[fleet_df['Category'] == 'Wide Body']['Type'].tolist(),
            "Narrow Body": fleet_df[fleet_df['Category'] == 'Narrow Body']['Type'].tolist(),
            "Cargo": fleet_df[fleet_df['Category'] == 'Cargo']['Type'].tolist(),
        }
        fig = go.Figure(go.Sankey(
            node=dict(
                pad=15,
                thickness=20,
                line=dict(color="black", width=0.5),
                label=nodes
            ),
            link=dict(
                source=[nodes.index(source) for source, targets in connections.items() for _ in targets],
                target=[nodes.index(target) for source, targets in connections.items() for target in targets],
                value=fleet_df['Total'].tolist()
            )
        ))
        fig.update_layout(
            title_text="",
            title_font=dict(size=24),
            font_size=25,
            height=950,
            title_x=0.5,
            title_y=0.95
        )
        st.plotly_chart(fig, use_container_width=True)
```

```
# Part-2
elif selected_fleet_option == "Number of Aircraft and Average Fleet Age":
    fleet_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Fleet_Age')

    fig = make_subplots(specs=[[{"secondary_y": True}]])

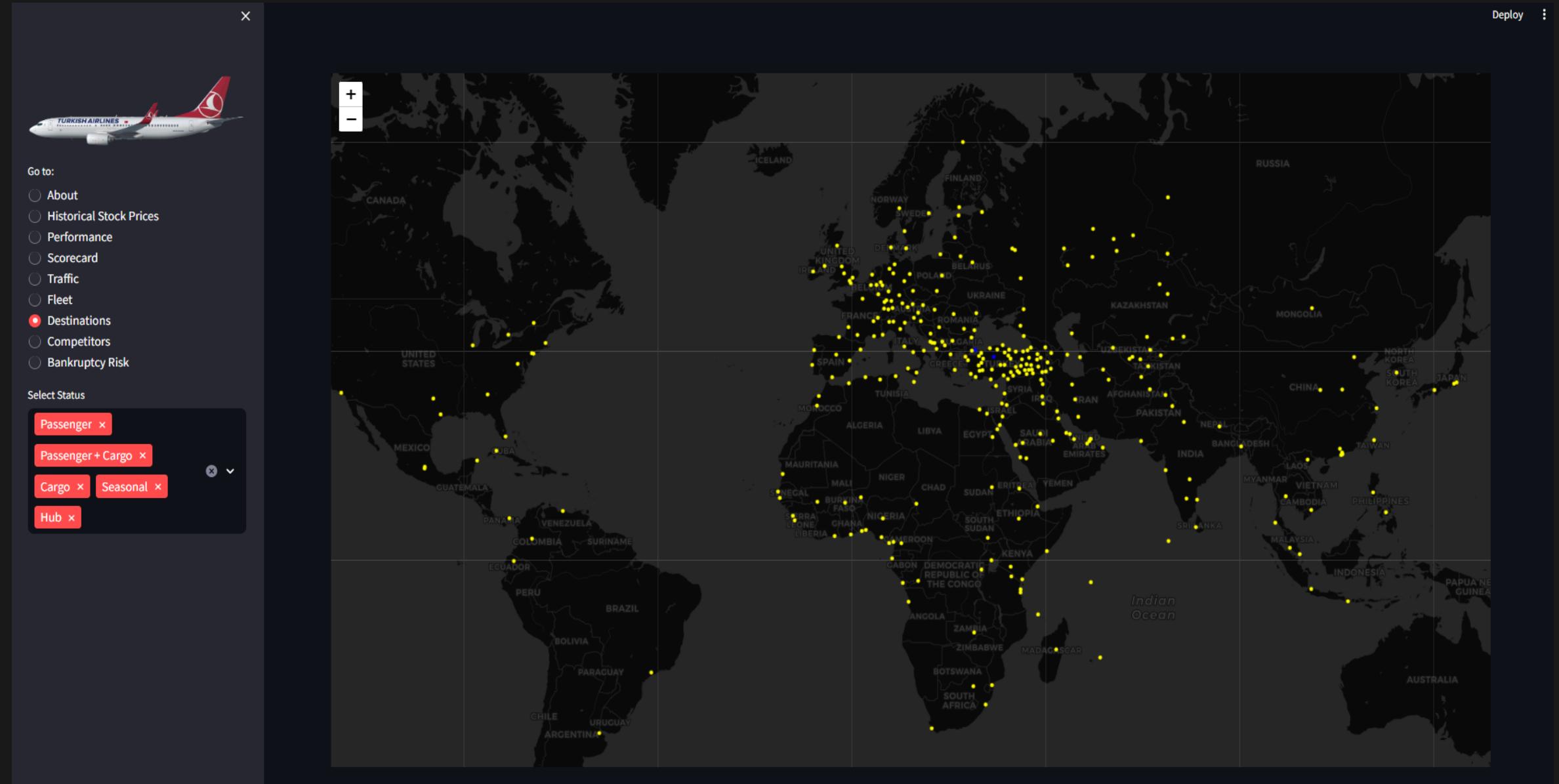
    fig.add_trace(
        go.Scatter(
            x=fleet_df['Date'],
            y=fleet_df['Fleet'],
            mode="lines",
            name="Number of Aircraft",
            line=dict(color="orange", width=4),
        ),
        secondary_y=False,
    )

    fig.add_trace(
        go.Scatter(
            x=fleet_df['Date'],
            y=fleet_df['Fleet Age'],
            mode="lines",
            name="Average Fleet Age",
            line=dict(color="red", width=4),
        ),
        secondary_y=True,
    )

    fig.update_xaxes(title_text="")
    fig.update_yaxes(title_text="Number of Aircraft", secondary_y=False)
    fig.update_yaxes(title_text="Average Fleet Age", secondary_y=True)
    fig.update_layout(
        title="Number of Aircraft and Average Fleet Age Over Time",
        height=900,
        legend=dict(x=0.02, y=0.98),
    )

st.plotly_chart(fig, use_container_width=True)
```

Destinations



```
# Part-1
elif page == "Destinations":

    destinations_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Destinations')

    default_list = ['Passenger', 'Passenger + Cargo', 'Cargo', 'Seasonal']
    selected_status = st.sidebar.multiselect("Select Status", destinations_df['Status'].unique(), default=default_list)
    filtered_destinations_df = destinations_df[destinations_df['Status'].isin(selected_status)]

    status_colors = {
        "Passenger": "yellow",
        "Passenger + Cargo": "yellow",
        "Cargo": "yellow",
        "Seasonal": "yellow",
        "Terminated": "gray",
        "Airport Closed": "gray",
        "Hub": "blue",
        "Focus city": "blue",
    }

    def get_marker_color(status):
        if "Begins" in status:
            return "red"
        else:
            return status_colors.get(status, "black")
```

```
# Part-2
m = folium.Map(tiles='cartodb dark_matter')
for index, row in filtered_destinations_df.iterrows():
    lat = row['Latitude_Decimal']
    lon = row['Longitude_Decimal']
    airport = row['Airport']
    status = row['Status']

    color = get_marker_color(status)

    folium.Circle(
        [lat, lon],
        popup=airport,
        tooltip=airport,
        color=color,
        fill=True,
        fill_color=color,
        fill_opacity=0.5,
    ).add_to(m)

sw = destinations_df[['Latitude_Decimal', 'Longitude_Decimal']].min().values.tolist()
ne = destinations_df[['Latitude_Decimal', 'Longitude_Decimal']].max().values.tolist()
m.fit_bounds([sw, ne])
# st_folium(m, use_container_width=True, height=850)
map_html = m._repr_html_()
st.components.v1.html(map_html, height=850)
```

Competitors



Go to:

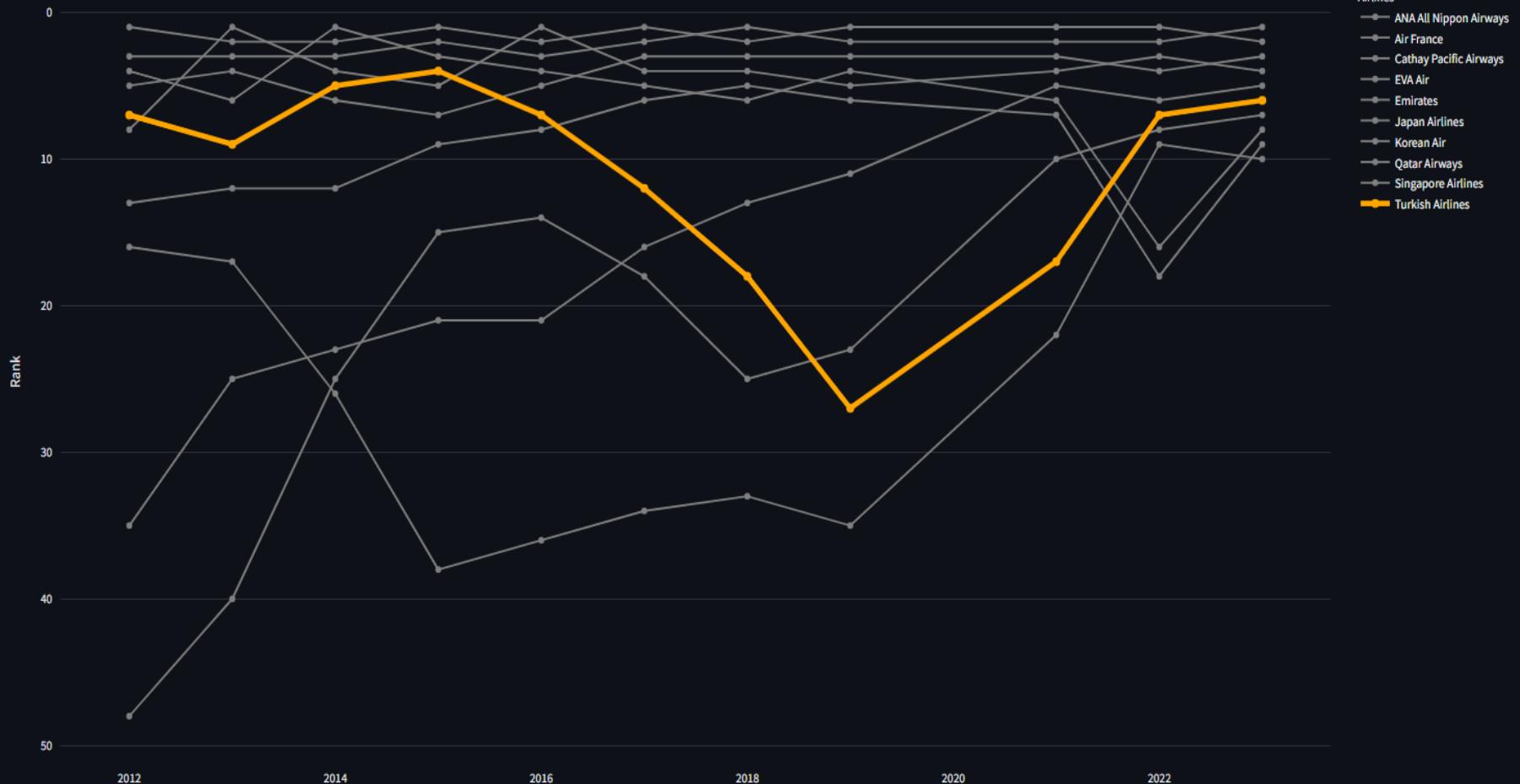
- About
- Historical Stock Prices
- Performance
- Scorecard
- Traffic
- Fleet
- Destinations
- Competitors
- Bankruptcy Risk

Competitors Type

SKYTRAX

▼

Rankings Evolution of the Top 10 Airlines in 2023 (2012-2023)



```
# Part-1
elif page == "Competitors":

    competitors_options = ["SKYTRAX"]
    selected_competitors_option = st.sidebar.selectbox("Competitors Type", competitors_options)

    if selected_competitors_option == 'SKYTRAX':
        df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Skytrax')

        max_year = df['Year'].max()
        filtered_df = df[(df['Year'] == max_year) & (df['Rank'] >= 1) & (df['Rank'] <= 10)]
        filtered_airlines = df[df['Airline'].isin(filtered_df['Airline'])]
        pivot_df = filtered_airlines.pivot(index='Year', columns='Airline', values='Rank')

        fig_title = f"Rankings Evolution of the Top 10 Airlines in {max_year} (2012-{max_year})"

        fig = px.line(pivot_df, x=pivot_df.index, y=pivot_df.columns, markers=True, labels={'Year': '', 'value': 'Rank'})
```

```
# Part-2
turkish_airlines = 'Turkish Airlines'
for airline_name in pivot_df.columns:
    if airline_name == turkish_airlines:
        fig.data[pivot_df.columns.get_loc(airline_name)].line.color = 'orange'
        fig.data[pivot_df.columns.get_loc(airline_name)].line.width = 5
        fig.data[pivot_df.columns.get_loc(airline_name)].update(marker=dict(size=8))
    else:
        fig.data[pivot_df.columns.get_loc(airline_name)].line.color = 'grey'

fig.update_yaxes(autorange="reversed")
fig.update_layout(
    xaxis_title="",
    yaxis_title="Rank",
    legend_title="Airlines",
    hovermode="closest",
    height=850,
    title=fig_title
)
st.plotly_chart(fig, use_container_width=True)
```

Bankruptcy Risk

Deploy



- Go to:
- About
 - Historical Stock Prices
 - Performance
 - Scorecard
 - Traffic
 - Fleet
 - Destinations
 - Competitors
 - Bankruptcy Risk

Method

Airscore

Airscore Over Time



```
# Part-1
elif page == "Bankruptcy Risk":

    method_options = st.sidebar.selectbox("Method", ["Airscore"])

    if method_options == 'Airscore':

        airsore_df = pd.read_excel('./4-streamlit-web-app/thyao_db.xlsx', sheet_name='Airscore')

        airsore_df['X1'] = airsore_df['Interest Expense, Net Non-Operating'] / airsore_df['Total Liabilities']
        airsore_df['X2'] = airsore_df['Net Income Before Taxes'] / airsore_df['KM Flown'] * 0.6214
        airsore_df['X3'] = airsore_df['Total Equity'] / airsore_df['Total Liabilities']
        airsore_df['Airscore'] = (-0.34140) * airsore_df['X1'] + 0.00003 * airsore_df['X2'] + 0.36134 * airsore_df['X3']
        airsore_df['Date'] = pd.to_datetime(airsore_df['Date'])
        airsore_df = airsore_df.set_index('Date')
```

```
# Part-2
fig = px.line(airscore_df, x=airscore_df.index, y='Airscore', title='Airscore Over Time')
    fig.update_xaxes(title_text="")
    fig.update_yaxes(title_text="Airscore")
    fig.update_layout(height=800)
    fig.add_shape(
        type='line',
        x0=airscore_df.index.min(),
        x1=airscore_df.index.max(),
        y0=0.03,
        y1=0.03,
        line=dict(color='green', width=2, dash='dash'),
    )

    fig.add_shape(
        type='line',
        x0=airscore_df.index.min(),
        x1=airscore_df.index.max(),
        y0=-0.095,
        y1=-0.095,
        line=dict(color='red', width=2, dash='dash'),
    )
st.plotly_chart(fig, use_container_width=True)
```

İletişim

- GitHub: <https://github.com/urazakgul>
- E-mail (Kişisel): urazdev@gmail.com
- E-mail (İş): uraz.akgul@riskturk.com

Teşekkür ederim ✉