Veri Toplama

- yfinance, investpy, quandl gibi kütüphaneler kullanılarak, 2005-01-01 tarihinden itibaren aylık getirilere sahip hisse senetleri ve sektör verileri toplanacak.
- Web scraping ile sektörlerin ve hisse senetlerinin listesi çekilecek.

```
#!pip install yfinance
#!pip install scipy==1.14.0
#!pip install --upgrade tsfresh

# Veri Çekme ve İşleme
import yfinance as yf
import pandas as pd
import requests
from io import StringIO
import random

# Web Scraping
from bs4 import BeautifulSoup

# Veri Görselleştirme
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline

def fetch_sectors_names():
    url = "https://stockanalysis.com/stocks/industry/sectors/"
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, "html.parser")
        df=pd.read_html(StringIO(str(soup.find_all("table"))))[0]

    else:
        print(f"Error: Failed to fetch data from page {url}")

    return df

def fetch_industry_names():
    url = "https://stockanalysis.com/stocks/industry/all/"
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, "html.parser")
        df=pd.read_html(StringIO(str(soup.find_all("table"))))[0]

    else:
        print(f"Error: Failed to fetch data from page {url}")
```

```python
    return df

def fetch_data(sectors):
    url = f"https://stockanalysis.com/stocks/sector/{sectors}/"
    response = requests.get(url)

    if response.status_code == 200:
        soup = BeautifulSoup(response.content, "html.parser")
        df=pd.read_html(StringIO(str(soup.find_all("table"))))[0]
        df.drop(columns='No.', inplace=True)

    else:
        print(f"Error: Failed to fetch data from page {url}")

    return df

sectors=fetch_sectors_names()
indusrty=fetch_industry_names()

sectors
```

```
               Sector Name  Stocks Market Cap Div. Yield  PE Ratio  \
0               Financials    1272     11.96T      0.17%     15.88
1               Healthcare    1157  8,050.42B      0.51%     59.35
2               Technology     769     21.51T      0.49%     45.77
3              Industrials     661  5,902.02B      1.19%     29.37
4   Consumer Discretionary     561  8,772.75B      0.75%     29.42
5                Materials     266  2,035.83B      1.71%     27.95
6              Real Estate     263  1,675.38B      3.78%     49.20
7                   Energy     251  3,613.08B      3.18%     13.25
8   Communication Services     245  6,456.13B      1.46%     33.37
9         Consumer Staples     242  4,229.57B      1.53%     29.62
10               Utilities     109  1,642.05B      2.74%     21.42

   Profit Margin 1D Change 1Y Change
0         19.97%    -0.56%    37.57%
1          3.19%    -0.64%    10.10%
2         14.62%    -1.02%    48.02%
3          7.25%    -0.60%    26.47%
4          6.51%    -2.14%    38.69%
5          6.33%    -1.00%    13.76%
6          9.51%    -0.40%    13.67%
7          8.18%    -0.05%    12.47%
8         11.59%    -1.02%    40.95%
9          4.90%    -0.38%    26.62%
10        10.62%    -0.22%    37.92%
```

```python
# Çektiğim verileri, data klasörü içerisinde tutuyorum
#mkdir ..\data\stock_sectors
```

```
fetch_data(sectors='energy').to_csv('../data/stock_sectors/energy.csv')

fetch_data(sectors='financials').to_csv('../data/stock_sectors/financials.csv')

fetch_data(sectors='healthcare').to_csv('../data/stock_sectors/healthcare.csv')

fetch_data(sectors='technology').to_csv('../data/stock_sectors/technology.csv')

fetch_data(sectors='utilities').to_csv('../data/stock_sectors/utilities.csv')

fetch_data(sectors='real-estate').to_csv('../data/stock_sectors/real-estate.csv')

fetch_data(sectors='materials').to_csv('../data/stock_sectors/materials.csv')

fetch_data(sectors='industrials').to_csv('../data/stock_sectors/industrials.csv')

fetch_data(sectors='consumer-staples').to_csv('../data/stock_sectors/consumer-staples.csv')

fetch_data(sectors='consumer-discretionary').to_csv('../data/stock_sectors/consumer-discretionary.csv')

fetch_data(sectors='communication-services').to_csv('../data/stock_sectors/communication-services.csv')
```

- Hangi sütunu baz alacağımı kararlaştırmak için sütun isimlerini yazdırdım
- ['Open'] sütununu baz alacağım

```
aapl_data = yf.download("AAPL")
print(aapl_data.columns)

[*********************100%***********************]  1 of 1 completed

MultiIndex([( 'Close', 'AAPL'),
            (  'High', 'AAPL'),
            (   'Low', 'AAPL'),
            (  'Open', 'AAPL'),
            ('Volume', 'AAPL')],
           names=['Price', 'Ticker'])



aapl_data.head()
```

```
Price          Close      High       Low      Open     Volume
Ticker          AAPL      AAPL      AAPL      AAPL       AAPL
Date
1980-12-12  0.098834  0.099264  0.098834  0.098834  469033600
1980-12-15  0.093678  0.094108  0.093678  0.094108  175884800
1980-12-16  0.086802  0.087232  0.086802  0.087232  105728000
1980-12-17  0.088951  0.089381  0.088951  0.088951   86441600
1980-12-18  0.091530  0.091959  0.091530  0.091530   73449600
```

```
financials = pd.read_csv('../data/stock_sectors/financials.csv')
healthcare = pd.read_csv('../data/stock_sectors/healthcare.csv')
technology = pd.read_csv('../data/stock_sectors/technology.csv')
```

Tarihsel Veri Filtreleme ve En Büyük Firmaları Seçme

```
- Rastgele seçim yapmadan önce 2005 öncesi verisi olan hisseleri
otomatik filtreledim
- En büyük 3 endüstriden (Sağlık, Finans ve Teknoloji) rastgele 500
tane firma seçtim
- Burada herhangi bir işlem yapmama gerek kalmadı çünkü zaten önceden
hem boş sütunları temizledim hem de bütün veriyi 2005 tarihinden
sonrası için ayarladım, kısacası sadece rastgele olarak firma seçmek
kaldı
```

```
# Geçerli semboller (NaN veya float olmayanları aldım)
technology_tickers =
technology['Symbol'].dropna().astype(str).tolist()
financials_tickers = financials['Symbol'].dropna().tolist()
healthcare_tickers = healthcare['Symbol'].dropna().tolist()

# Her bir kategoriden EN BÜYÜK 600 şirket seçtim
technology_biggest = technology_tickers[:600]
financials_biggest = financials_tickers[:600]
healthcare_biggest = healthcare_tickers[:600]

# Tüm şirketleri birleştirdim
all_biggest_tickers = technology_biggest + financials_biggest +
healthcare_biggest

len(all_biggest_tickers)
```

```
1800
```

```
import time

# Semboller için listeyi 250'lik parçalara böldüm ki veri çekmesi
kolay olsun
```

```
chunks = [all_biggest_tickers[i:i+250] for i in range(0,
len(all_biggest_tickers), 250)]

# Veriyi parça parça çektim
all_data = []
for chunk in chunks:
    try:
        data = yf.download(chunk, start='2005-01-01')
        all_data.append(data)
        time.sleep(15)  # 15 saniye bekleme, API limitini aşmamak için
    except yf.download.YFRateLimitError:
        print("Rate limit exceeded. Retrying in 60 seconds...")
        time.sleep(60)
```

```
[*********************100%***********************]  250 of 250
completed
[*********************100%***********************]  250 of 250
completed
[*********************100%***********************]  250 of 250
completed

1 Failed download:
['BRK.B']: YFTzMissingError('$%ticker%: possibly delisted; no timezone
found')
[*********************100%***********************]  250 of 250
completed

1 Failed download:
['AGM.A']: YFTzMissingError('$%ticker%: possibly delisted; no timezone
found')
[*********************100%***********************]  250 of 250
completed

5 Failed downloads:
['TDAC']: YFPricesMissingError('$%ticker%: possibly delisted; no price
data found  (1d 2005-01-01 -> 2025-02-12)')
['JACS.RT', 'CRD.B', 'KFII', 'CRD.A']: YFTzMissingError('$%ticker%:
possibly delisted; no timezone found')
[*********************100%***********************]  250 of 250
completed

1 Failed download:
['BIO.B']: YFPricesMissingError('$%ticker%: possibly delisted; no
price data found  (1d 2005-01-01 -> 2025-02-12)')
[*********************100%***********************]  250 of 250
completed
[*********************100%***********************]  50 of 50 completed
```

```
tickers = technology['Symbol'].tolist()
```

```
# Verileri çek
data = yf.download(tickers, start='2005-01-01')
```

```
TypeError: expected string or bytes-like object, got 'float'.
```

- Bu hatayı aldığım için `yfinance.download()` fonksiyonu bir `float` değeri ile karşılaşıyor, fakat bu fonksiyon yalnızca `string` türündeki sembollerle çalışabilir. `technology[Symbol]` sütununda bazı `NaN` veya `float` değerleri mevcut. Bu nedenle hata veriyor
    a. NaN değerlerini temizledim
    b. Sadece string değerleri aldım
- Ekstra olarak
    - `YFRateLimitError('Too Many Requests. Rate limited. Try after a while.')` hatası nedeniyle aralıklı veri indirdim

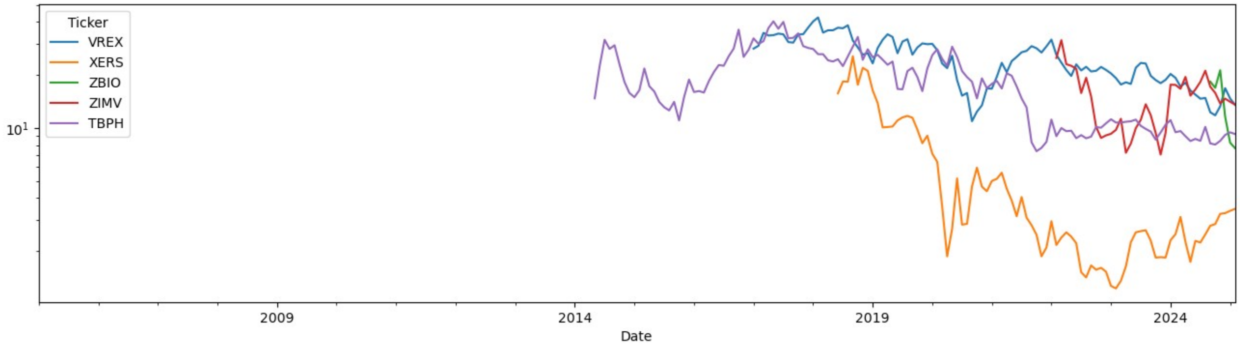
```
# Sonuçları birleştir
final_data = pd.concat(all_data, axis=1)

# "Open" fiyatlarını al
data_open = final_data['Open']

# Günlük verileri aylığa çeviriyoruz.
data_open_monthly = data_open.resample('M').first()

# Sonuç
print(data_open_monthly.head())
```

```
Ticker           AAPL      ACIW        ACN       ADBE        ADI
ADP  \
Date

2005-01-31  0.974731  6.696667  19.106171  31.493821  23.208159
21.876524
2005-02-28  1.159356  7.026667  18.113641  28.444421  22.499747
21.432076
2005-03-31  1.353911  7.833333  18.134912  31.023913  23.340560
21.209859
2005-04-30  1.266639  7.760000  17.702448  33.910000  22.857036
22.277329
2005-05-31  1.089689  6.980000  15.348742  29.870001  21.419059
21.528960

Ticker            ADSK  AFRM  AI   AKAM  ...  SCPH  SLN  SLRN
STXS  THRD  \
Date                                    ...

2005-01-31  38.280256   NaN NaN  13.00  ...   NaN  NaN   NaN
91.240875   NaN
2005-02-28  29.144969   NaN NaN  13.10  ...   NaN  NaN   NaN
86.678833   NaN
```

```
2005-03-31  29.914572   NaN NaN  11.01  ...   NaN  NaN   NaN
86.678833   NaN
2005-04-30  29.690001   NaN NaN  12.72  ...   NaN  NaN   NaN
71.350365   NaN
2005-05-31  31.850000   NaN NaN  11.96  ...   NaN  NaN   NaN
64.598541   NaN

Ticker      TLSI       UTMD  VOR        VXRT  ZYBT
Date
2005-01-31   NaN  14.252260  NaN  867.110930   NaN
2005-02-28   NaN  12.933301  NaN  771.752428   NaN
2005-03-31   NaN  13.708426  NaN  759.906664   NaN
2005-04-30   NaN  13.306285  NaN  740.361116   NaN
2005-05-31   NaN  13.501401  NaN  647.963999   NaN

[5 rows x 1800 columns]
```

- 1800 şirket verisini küçük bir alan kaplaması için parquet veri tipinde sakladım ve gzip şekilden sıkıştırdım

```
data_open_monthly.to_parquet("../data/processed_data/
1800_company_data.gzip", compression="gzip")

data_open_monthly.columns

Index(['AAPL', 'ACIW', 'ACN', 'ADBE', 'ADI', 'ADP', 'ADSK', 'AFRM',
'AI',
       'AKAM',
       ...
       'SCPH', 'SLN', 'SLRN', 'STXS', 'THRD', 'TLSI', 'UTMD', 'VOR',
'VXRT',
       'ZYBT'],
      dtype='object', name='Ticker', length=1800)

# Şirket isimlerini belirleyin
companies = ['VREX', 'XERS', 'ZBIO', 'ZIMV', 'TBPH']

# Veriyi sadece istenen şirketlere göre filtrele
data_open_monthly_filtered = data_open_monthly[companies]

data_open_monthly_filtered.plot(kind='line',figsize=(16,4),logy=True)

<Axes: xlabel='Date'>
```



Hangi Durumda Eksik Verileri Doldurmak Mantıklı?

1. **Az sayıda eksik değer varsa** (örneğin, %10-20 civarında)
   - `ffill().bfill()` yöntemi **önceki ve sonraki değerlere bağlı olarak makul bir dolgu** yapar.

   - Eğer bir şirketin birkaç ay verisi eksikse ama genel trend belliyse, doldurmak **makul bir tahmin sağlar**.

   - **Sonuç: Doğruluk korunur**, özellikle kısa vadeli analizlerde işe yarar.
2. **Şirketin ticareti kesintiye uğramışsa ama sonra devam etmişse**
   - **İlk veriler sıfır olmamalı** çünkü o dönemlerde şirket aktifti ama veriler eksikti.

   - `mean()` kullanımı **güvenli olabilir**, ancak trendi bozabilir.

Hangi Durumda Şirketleri Droplamak Mantıklı?

1. **%70-80 eksik veri varsa**
   - Aylık açılış fiyatları gibi **trend takibi gerektiren bir veride**, aşırı boşluklar **gerçekçi olmayan tahminlere** neden olur.

   - `ffill().bfill()` yapıldığında **ilk başta dümdüz bir çizgi olup sonra ani değişim olması**, aslında **tahmin edilen verinin hatalı olabileceğini** gösterir.

   - **Sonuç: Eğer bir şirketin verisi %70-80 eksikse, bu şirketi droplamak genellikle daha güvenlidir.**

Ne Yapmalısın?

Seçenek 1: Çok Eksik Olan Şirketleri Filtreleyerek Dropla

```
missing_ratio = data_open_monthly.isna().mean()
filtered_data = data_open_monthly.loc[:, missing_ratio < 0.3]  #
%30'dan fazla eksik olanları dropla
```

- %70-80 eksik olanları tamamen çıkarırsın, geriye daha sağlam veriler kalır.

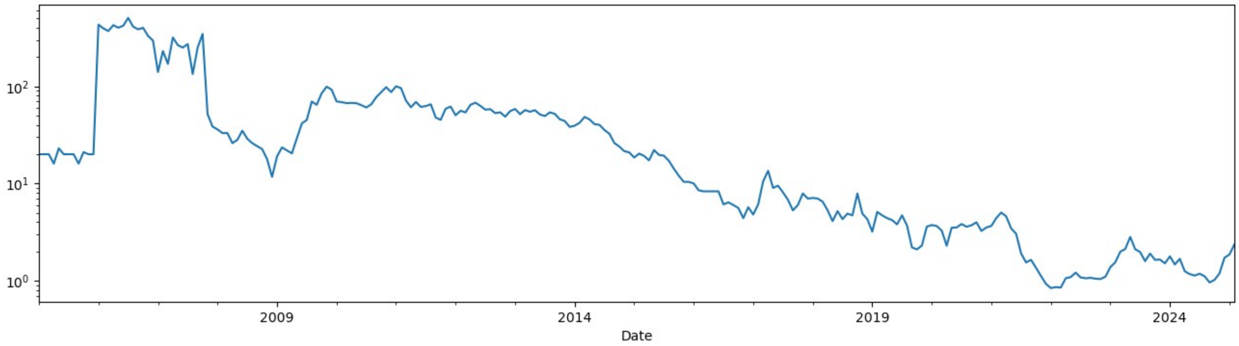
Seçenek 2: Eksik Azsa Doldur `(ffill + bfill)`

```
filtered_data = filtered_data.ffill().bfill()
```

- Sadece az eksik olan şirketler için doldurma yapılır, daha güvenilir sonuç verir.

```
filtered_data = 
pd.read_parquet("../data/processed_data/1800_company_data.gzip")

# Her sütunun eksik veri oranını hesapla
missing_ratio = data_open_monthly.isna().mean()

# %40'tan fazla eksik veri olan sütunları drop et
filtered_data = data_open_monthly.loc[:, missing_ratio < 0.4]

# Kalan verileri ffill ve bfill ile doldur
filtered_data = filtered_data.ffill().bfill()

# Sonucu göster
print(filtered_data.head())

Ticker          AAPL      ACIW        ACN       ADBE        ADI  
ADP  \
Date                                                              

2005-01-31  0.974731  6.696667  19.106171  31.493821  23.208159  
21.876524   
2005-02-28  1.159356  7.026667  18.113641  28.444421  22.499747  
21.432076   
2005-03-31  1.353911  7.833333  18.134912  31.023913  23.340560  
21.209859   
2005-04-30  1.266639  7.760000  17.702448  33.910000  22.857036  
22.277329   
2005-05-31  1.089689  6.980000  15.348742  29.870001  21.419059  
21.528960   

Ticker           ADSK   AKAM       AMAT        AMD  ...      ZYXI  
CATX  CGEN  \
Date                                                ...             

2005-01-31  38.280256  13.00  12.584156  22.110001  ...  0.146753  
53.5  5.01   
2005-02-28  29.144969  13.10  11.735455  15.900000  ...  0.132078  
53.5  6.22   
2005-03-31  29.914572  11.01  12.906074  17.629999  ...  0.141861  
53.5  4.73   
2005-04-30  29.690001  12.72  12.035424  16.309999  ...  0.112511  
53.5  4.10   
2005-05-31  31.850000  11.96  10.945289  14.210000  ...  0.122294  
53.5  3.79   
```

```
Ticker        INFU  LFCR  PHLT   PLX        STXS       UTMD        VXRT
Date
2005-01-31   3.75  6.84  9.25  20.0  91.240875  14.252260  867.110930
2005-02-28   3.75  6.89  9.25  20.0  86.678833  12.933301  771.752428
2005-03-31   3.75  6.99  9.25  20.0  86.678833  13.708426  759.906664
2005-04-30   3.75  7.48  9.25  16.0  71.350365  13.306285  740.361116
2005-05-31   3.75  6.54  9.25  23.0  64.598541  13.501401  647.963999

[5 rows x 900 columns]

filtered_data['PLX'].plot(kind='line',figsize=(16,4),logy=True)

<Axes: xlabel='Date'>
```



Data'yı Long Formata Dönüştürme:

1. **`data_open_monthly.reset_index()`**
   - `long_data = filtered_data.reset_index().melt(id_vars='Date', var_name='Ticker', value_name='Open')` Bu satır, `filtered_data` verisini **uzun formata** dönüştürmek için pandas'ın `melt()` fonksiyonunu kullanır.
   - `filtered_data` bir pandas **Series** objesidir ve indeksi tarihlerdir (aylık bazda).

   - `reset_index()`, bu diziyi bir **DataFrame**'e çevirir.
     - **İndeks** (`Date` sütunu olarak) veriye eklenir.
     - Açılış fiyatları aynı sütunda kalır.
2. **`.melt(id_vars='Date', var_name='Ticker', value_name='Open')`**
   - `melt()` fonksiyonu, veriyi **geniş formattan uzun formata** dönüştürür.
   - **`id_vars='Date'`** → `"Date"` sütununu korur.
   - **`var_name='Ticker'`** → Önceden sütun başlıklarında olan hisse senedi kodlarını `"Ticker"` olarak adlandırır.
   - **`value_name='Open'`** → Açılış fiyatlarını `"Open"` sütununa taşır.

Dönüştürmeden Önce (Geniş Format)

Date	AAPL	MSFT	GOOG
2024-01-01	150	300	2800
2024-02-01	155	310	2900

Dönüştürdükten Sonra (Uzun Format)

Date	Ticker	Open
2024-01-01	AAPL	150
2024-01-01	MSFT	300
2024-01-01	GOOG	2800
2024-02-01	AAPL	155
2024-02-01	MSFT	310
2024-02-01	GOOG	2900

Neden Kullanılır?

- Uzun format, **gruplama, filtreleme ve görselleştirme** işlemleri için daha uygundur.
- `seaborn` veya `matplotlib` ile grafik çizmek için daha kullanışlıdır.
- **Diğer veri setleriyle birleştirme** işlemleri (örneğin, sektör bilgisi eklemek) daha kolay olur.
- Ben 2. ve 3. neden için kullanıyor olacağım

```python
# Uzun formata çevirme
long_data = filtered_data.reset_index().melt(id_vars='Date',
var_name='Ticker', value_name='Open')

# Sektör bilgilerini ekleme
sector_map = {}
for ticker in financials_biggest:
    sector_map[ticker] = 'Financials'

for ticker in healthcare_biggest:
    sector_map[ticker] = 'Healthcare'

for ticker in technology_biggest:
    sector_map[ticker] = 'Technology'

long_data['Sector'] = long_data['Ticker'].map(sector_map)

long_data.head()
```

```
        Date Ticker      Open      Sector
0 2005-01-31   AAPL  0.974731  Technology
1 2005-02-28   AAPL  1.159356  Technology
2 2005-03-31   AAPL  1.353911  Technology
3 2005-04-30   AAPL  1.266639  Technology
4 2005-05-31   AAPL  1.089689  Technology
```

```python
# Datayı kayıt et
long_data.to_csv('../data/processed_data/combined_data.csv',
index=False)

long_formatted_data =
pd.read_csv('../data/processed_data/combined_data.csv')

meaningless_data_difference = long_formatted_data[
    (long_formatted_data["Ticker"].isin(["AAPL", "EYPT"])) &
    (long_formatted_data["Date"].between("2005-01-01", "2006-01-01"))
][["Date", "Ticker", "Open"]]

print(meaningless_data_difference)
```

```
              Date Ticker        Open
0       2005-01-31   AAPL    0.974731
1       2005-02-28   AAPL    1.159356
2       2005-03-31   AAPL    1.353911
3       2005-04-30   AAPL    1.266639
4       2005-05-31   AAPL    1.089689
5       2005-06-30   AAPL    1.200433
6       2005-07-31   AAPL    1.108347
7       2005-08-31   AAPL    1.281084
8       2005-09-30   AAPL    1.414399
9       2005-10-31   AAPL    1.629870
10      2005-11-30   AAPL    1.722558
11      2005-12-31   AAPL    2.074954
206910  2005-01-31   EYPT  420.000000
206911  2005-02-28   EYPT  356.000000
206912  2005-03-31   EYPT  328.000000
206913  2005-04-30   EYPT  270.000000
206914  2005-05-31   EYPT  208.000000
206915  2005-06-30   EYPT  238.000000
206916  2005-07-31   EYPT  244.399994
206917  2005-08-31   EYPT  272.000000
206918  2005-09-30   EYPT  244.000000
206919  2005-10-31   EYPT  276.399994
206920  2005-11-30   EYPT  240.000000
206921  2005-12-31   EYPT  188.000000
```

Log Dönüşümü Uygulaması ve Nedenleri

Veri setinizde farklı ölçeklerde değerler gözlemlenmektedir. Örneğin, bazı Ticker'ların (ör. **AAPL**) değerleri 1 civarındayken, bazıları (ör. **EYPT**) 400 gibi yüksek değerlere sahip. Bu durum, modellerin öğrenme sürecinde farklı ölçekler arasında dengesizliğe yol açabilir.

Log Dönüşümünün Avantajları

- **Ölçek Dengeleme:**
  Log dönüşümü, büyük değerlerin etkisini azaltarak tüm değerleri daha benzer bir

ölçeğe çeker. Bu, özellikle farklı büyüklükteki değişkenlerin (tickerların) analizinde faydalıdır.

- **Varyansın Stabilizasyonu:**
  Log dönüşümü, veri dağılımındaki varyansı stabil hale getirerek aşırı uç değerlerin (outlier) etkisini hafifletebilir.

- **Dağılımın Normalleşmesi:**
  Birçok durumda log dönüşümü, verinin sağa çarpık dağılımını (right-skewed) daha normal bir dağılıma yaklaştırır, bu da bazı makine öğrenmesi algoritmalarının performansını artırabilir.

Dikkat Edilmesi Gerekenler

- **Pozitif Değerler:**
  Log dönüşümünü doğrudan uygulayabilmek için verilerinizin pozitif olması gerekir. Eğer bazı değerler 0 veya negatif ise, `log1p` (yani, log(1 + x)) dönüşümü uygulanabilir.

- **Farklı Ölçekler:**
  Her ticker için log dönüşümünü ayrı ayrı uygulamak daha mantıklı olabilir. Bu sayede, her bir zaman serisinin kendi dağılım özelliklerine göre dönüşüm yapılır.

- **Küçük Değerler:**
  Değerlerin 1'in altında olması durumunda log dönüşümü negatif sonuçlar verecektir. Bu durum modelleme açısından problem oluşturmaz, ancak yorumlanması sırasında dikkatli olunmalıdır.

Sonuç

Eğer bazı değerler diğerlerine göre "mantıksız" ya da "ölçülemeyecek kadar farklı" görünüyorsa, **log dönüşümü uygulamak**:

- Büyük ölçekli değerlerin etkisini azaltır,
- Farklı tickerların değerlerini karşılaştırılabilir hale getirir,
- Makine öğrenmesi ve istatistiksel modellerde daha stabil sonuçlar elde etmenize yardımcı olur.

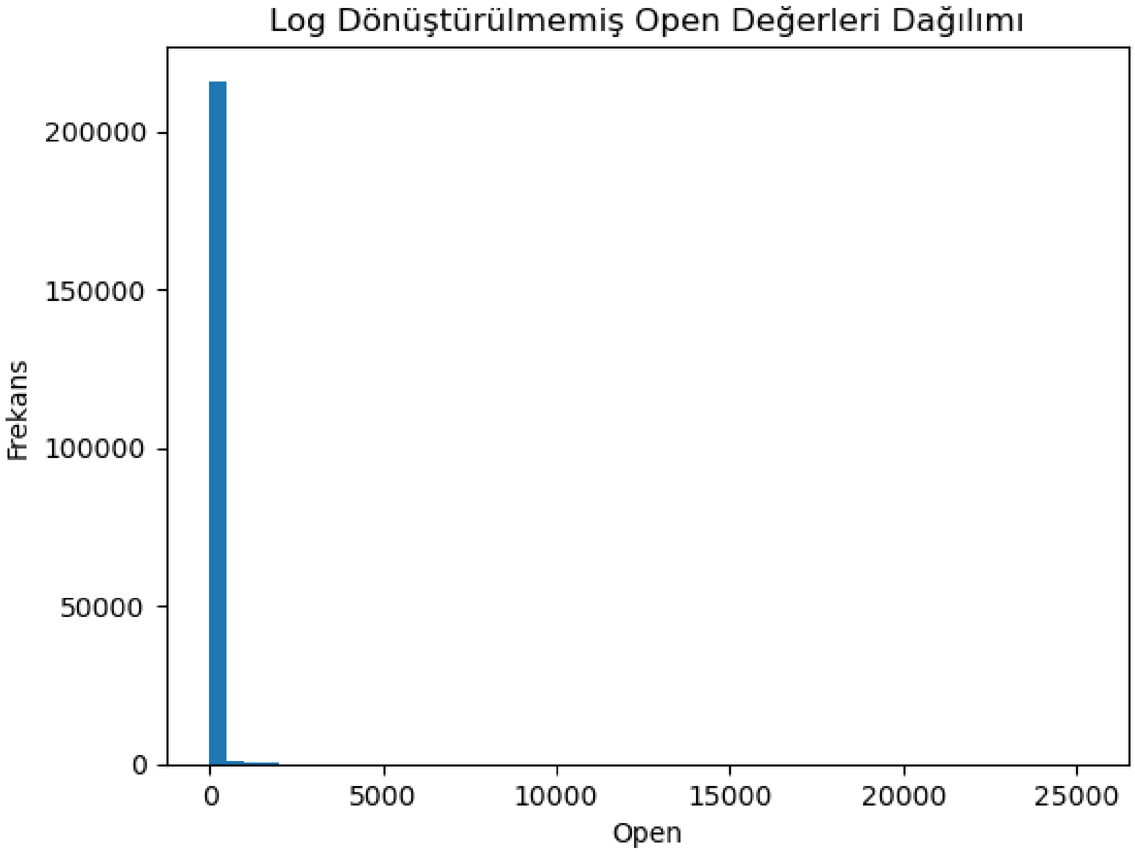
Bu nedenle, veri setinizdeki ölçek farklılıklarını gidermek ve istatistiksel özellikleri (örn. `tsfresh` ile çıkarılacak öznitelikler) daha tutarlı hale getirmek için log dönüşümü mantıklı bir adım olabilir.

```
long_formatted_data["Open_LOG"] =
np.log1p(long_formatted_data["Open"])

#Halihazırdaki Open sütununu güncellemek yerine yeni sütun açmak daha
mantıklı
#çünkü bu kısmı tekrar tekrar çalıştırırsak Open değerleri sürekli
değişiyor olacak.
```

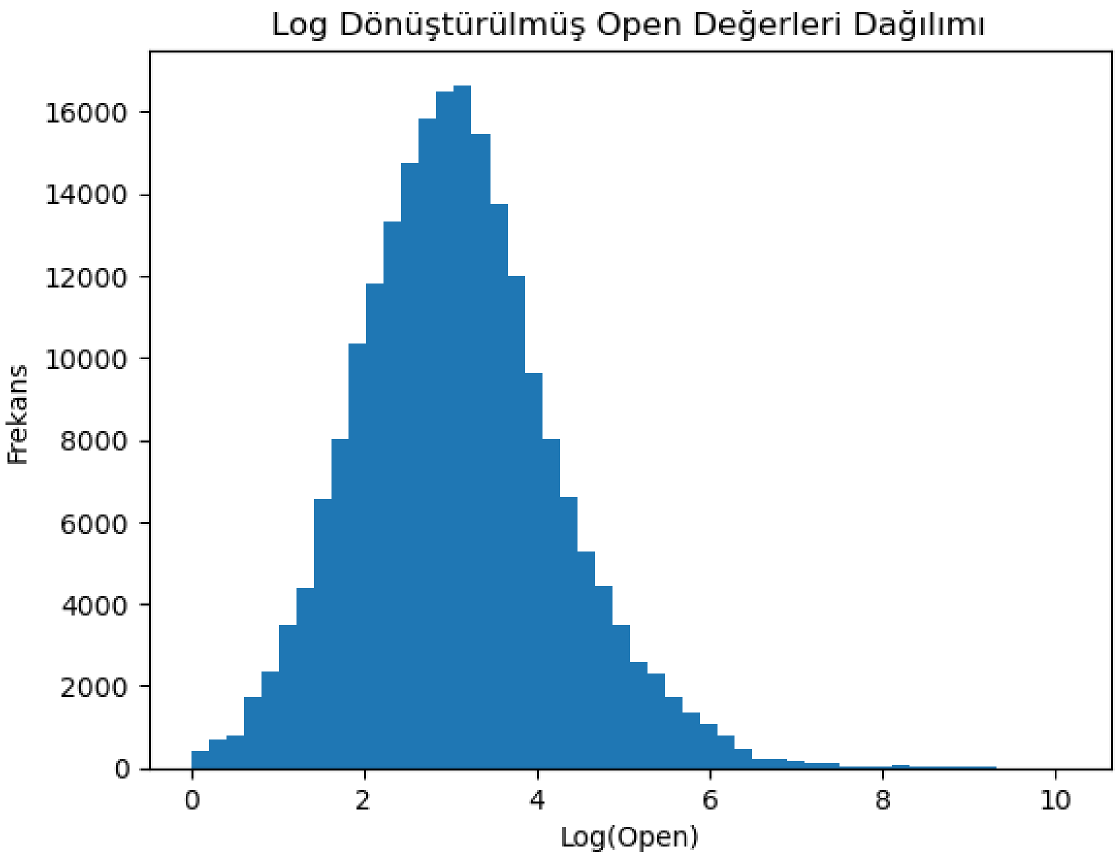
```python
meaningless_data_difference = long_formatted_data[
    (long_formatted_data["Ticker"].isin(["AAPL", "EYPT"])) &
    (long_formatted_data["Date"].between("2005-01-01", "2006-01-01"))
][["Date", "Ticker", "Open_LOG"]]

print(meaningless_data_difference)
```

```
              Date Ticker  Open_LOG
0       2005-01-31   AAPL  0.680432
1       2005-02-28   AAPL  0.769810
2       2005-03-31   AAPL  0.856078
3       2005-04-30   AAPL  0.818298
4       2005-05-31   AAPL  0.737015
5       2005-06-30   AAPL  0.788654
6       2005-07-31   AAPL  0.745904
7       2005-08-31   AAPL  0.824651
8       2005-09-30   AAPL  0.881450
9       2005-10-31   AAPL  0.966934
10      2005-11-30   AAPL  1.001572
11      2005-12-31   AAPL  1.123290
206910  2005-01-31   EYPT  6.042633
206911  2005-02-28   EYPT  5.877736
206912  2005-03-31   EYPT  5.796058
206913  2005-04-30   EYPT  5.602119
206914  2005-05-31   EYPT  5.342334
206915  2005-06-30   EYPT  5.476464
206916  2005-07-31   EYPT  5.502890
206917  2005-08-31   EYPT  5.609472
206918  2005-09-30   EYPT  5.501258
206919  2005-10-31   EYPT  5.625460
206920  2005-11-30   EYPT  5.484797
206921  2005-12-31   EYPT  5.241747
```

```python
plt.hist(long_formatted_data["Open"], bins=50)
plt.title("Log Dönüştürülmemiş Open Değerleri Dağılımı")
plt.xlabel("Open")
plt.ylabel("Frekans")
plt.show()
```



```
plt.hist(long_formatted_data["Open_LOG"], bins=50)
plt.title("Log Dönüştürülmüş Open Değerleri Dağılımı")
plt.xlabel("Log(Open)")
plt.ylabel("Frekans")
plt.show()
```



- **Open Değerleri**: İlk histogramdaki x=0'daki çubuk muhtemelen veri kümesinde çok sayıda sıfır veya çok küçük değerler olduğunu gösterir. Bu, Açık sütununda çok sayıda sıfır veya sıfıra yakın değer varsa meydana gelebilir. Ancak önemli olan, x'teki diğer sayıların çok büyük olmasıdır.

- **Open_LOG Değerleri**: Log dönüşümü (log1p) değerleri daha eşit bir şekilde dağılmıştır ve aşırı uç değerlerin etkisini azaltılmıştır

```
long_formatted_data.to_csv('../data/processed_data/
LOG_combined_data.csv', index=False)

# Oluşturduğumuz tabloyu okuyalaım
combined_data =
pd.read_csv('../data/processed_data/LOG_combined_data.csv')

combined_data.head()

         Date Ticker      Open      Sector  Open_LOG
0  2005-01-31   AAPL  0.974731  Technology  0.680432
1  2005-02-28   AAPL  1.159356  Technology  0.769810
2  2005-03-31   AAPL  1.353911  Technology  0.856078
3  2005-04-30   AAPL  1.266639  Technology  0.818298
4  2005-05-31   AAPL  1.089689  Technology  0.737015
```

- Data-type'larında hata var ve tsfresh kütüphanesi ile çalışamam
- Hatanın nedeni, Date sütunu type olarak obect ancak datetime64 olması gerek

```
print(combined_data.dtypes)

Date          object
Ticker        object
Open         float64
Sector        object
Open_LOG     float64
dtype: object

# Date sütununu datetime64 formuna çevirdim
combined_data['Date'] = pd.to_datetime(combined_data['Date'])

print(combined_data.dtypes)

Date        datetime64[ns]
Ticker              object
Open               float64
Sector              object
Open_LOG           float64
dtype: object
```

Kategorik Değişkenlerle Çalışma

Kategorik veriler, sayısal olmayan ve genellikle kategorilere ayrılan verilerdir. Örneğin, bir şirketin faaliyet gösterdiği sektör gibi kategorik bilgiler, modelleme süreçlerinde sayısal verilere dönüştürülmelidir. Kategorik verileri sayısal hale getirmek için kullanılan yaygın yöntemler **One-Hot Encoding** ve **Label Encoding**'dir.

Sektör Verisi (Nominal Değişken)

Verimdeki **Sektör** bilgisi, **nominal** (isimsel) bir değişkendir. Yani, sektörler arasında **doğal bir sıralama** ya da **ağırlık** yoktur. Örneğin, bir şirketin teknoloji, finans veya sağlık sektöründe olması, diğer sektörlere göre daha üstün ya da daha düşük bir değer taşımaz. Her bir sektör, sadece bir isimsel kategoriyi ifade eder.

One-Hot Encoding

Nominal veriler için, **One-Hot Encoding** yöntemi genellikle tercih edilir. Bu yöntemde, her benzersiz kategori için ayrı bir sütun oluşturulur ve gözlemde o kategoriye ait olan değer **1** ile işaretlenirken, diğerleri **0** olarak kodlanır. Bu, modelin sektöre ait kategorileri sayısal verilere dönüştürmesine olanak tanır ve herhangi bir sıralama veya ağırlık ilişkisi oluşturmaz.

Örneğin, **Sektör** değişkeninde "Teknoloji", "Finans" ve "Sağlık" gibi kategoriler varsa, One-Hot Encoding ile şu şekilde bir dönüşüm yapılır:

Date	Ticker	Open	Sector	Sector_Teknoloji	Sector_Finans	Sector_Sağlık
2022-01-01	AAPL	150	Teknoloji	1	0	0
2022-01-02	TSLA	700	Finans	0	1	0
2022-01-03	PFE	40	Sağlık	0	0	1

Bu şekilde, her sektör için ayrı bir sütun eklenmiş olur ve her gözlem, o sektöre ait olan sütunda **1** değerini alırken, diğer sütunlarda **0** değerini alır. Bu, modelin sektörler arasında herhangi bir sıralama yapmamasını sağlar ve sektörel bilgiler daha doğru bir şekilde temsil edilmiş olur.

```
# One-hot encoding uygulama: 'Sector' sütununu dönüştürüyoruz.
combined_data_encoded = pd.get_dummies(combined_data,
columns=['Sector'])

print(combined_data_encoded.head())

        Date Ticker      Open  Open_LOG  Sector_Financials
Sector_Healthcare  \
0 2005-01-31   AAPL  0.974731  0.680432              False
False
1 2005-02-28   AAPL  1.159356  0.769810              False
False
2 2005-03-31   AAPL  1.353911  0.856078              False
False
3 2005-04-30   AAPL  1.266639  0.818298              False
False
4 2005-05-31   AAPL  1.089689  0.737015              False
False

   Sector_Technology
0              True
1              True
2              True
3              True
4              True

combined_data_encoded.to_csv('../data/processed_data/
encoded_combined_data.csv', index=False)
```

Öznitelik Çıkarımı ve Seçme

Öznitelik Çıkarımı:

tsfresh ile Otomatik Özellik Çıkarımı

Zaman serisi analizinde, veriden istatistiksel özellikler çıkarmak modelin başarısını artırabilir. `tsfresh` kütüphanesi, zaman serisi verilerinden otomatik olarak öznitelik (feature) çıkarımı yaparak analitik süreçleri hızlandırır. Bu süreçte çıkarılacak bazı temel istatistiksel özellikler şunlardır:

- **Ortalama (Mean)**: Verinin genel eğilimini belirler.
- **Standart Sapma (Standard Deviation)**: Verinin ne kadar değişkenlik gösterdiğini gösterir.
- **Otokorelasyon (Autocorrelation)**: Bir zaman serisinin önceki değerleriyle olan ilişkisini ölçer.
- **Minimum ve Maksimum Değerler**: Veri setinin uç noktalarını belirler.
- **Medyan ve Çeyrek Değerler**: Veri dağılımı hakkında bilgi verir.

Ancak, **`tsfresh` ile başarılı bir öznitelik çıkarımı için veri setinde eksik değer bulunmamalıdır**. Eksik değerlerin doldurulması, sağlıklı analiz yapabilmek için kritik bir adımdır.

```
data_open_filled =
pd.read_csv('../data/processed_data/encoded_combined_data.csv')
```

```
data_open_filled.isna().sum()
```

```
Date                 0
Ticker               0
Open                 0
Open_LOG             0
Sector_Financials    0
Sector_Healthcare    0
Sector_Technology    0
dtype: int64
```

```
import tsfresh
from tsfresh.feature_extraction import EfficientFCParameters
```

```
# Extract features using only the 'Open' column
data_extract_features = tsfresh.extract_features(
    data_open_filled,
    column_id='Ticker',
    column_sort='Date',
    column_value='Open_LOG',  # Explicitly specify the value column
    default_fc_parameters=EfficientFCParameters()
)
```

```
Feature Extraction: 100%|██████████| 30/30 [00:50<00:00,  1.68s/it]
```

```
data_extract_features.to_parquet('../data/processed_data/
extracted_features.parquet', compression='gzip')
```

```
data_extract_features.columns
```

```
Index(['Open_LOG__variance_larger_than_standard_deviation',
       'Open_LOG__has_duplicate_max', 'Open_LOG__has_duplicate_min',
       'Open_LOG__has_duplicate', 'Open_LOG__sum_values',
       'Open_LOG__abs_energy', 'Open_LOG__mean_abs_change',
       'Open_LOG__mean_change',
'Open_LOG__mean_second_derivative_central',
       'Open_LOG__median',
       ...
       'Open_LOG__fourier_entropy__bins_5',
       'Open_LOG__fourier_entropy__bins_10',
       'Open_LOG__fourier_entropy__bins_100',
       'Open_LOG__permutation_entropy__dimension_3__tau_1',
       'Open_LOG__permutation_entropy__dimension_4__tau_1',
       'Open_LOG__permutation_entropy__dimension_5__tau_1',
       'Open_LOG__permutation_entropy__dimension_6__tau_1',
       'Open_LOG__permutation_entropy__dimension_7__tau_1',
       'Open_LOG__query_similarity_count__query_None__threshold_0.0',
       'Open_LOG__mean_n_absolute_max__number_of_maxima_7'],
      dtype='object', length=777)
```

```
extracted_data =
pd.read_parquet('../data/processed_data/extracted_features.parquet')
```

```
extracted_data.head()
```

```
      Open_LOG__variance_larger_than_standard_deviation  \
A                                                   0.0
AAPL                                                1.0
AB                                                  0.0
ABBV                                                0.0
ABCB                                                0.0

      Open_LOG__has_duplicate_max  Open_LOG__has_duplicate_min  \
A                             0.0                          0.0
AAPL                          0.0                          0.0
AB                            0.0                          0.0
ABBV                          0.0                          1.0
ABCB                          0.0                          0.0

      Open_LOG__has_duplicate  Open_LOG__sum_values
Open_LOG__abs_energy  \
A                         0.0            913.803852
3586.251889
AAPL                      0.0            764.921929
2870.542166
AB                        0.0            657.495086
```

```
1859.250052
ABBV                      1.0            916.311508
3591.156249
ABCB                      0.0            753.473377
2445.777047

      Open_LOG__mean_abs_change  Open_LOG__mean_change  \
A                      0.063168               0.009434
AAPL                   0.066749               0.019755
AB                     0.062410               0.005599
ABBV                   0.034470               0.008760
ABCB                   0.073705               0.006244

      Open_LOG__mean_second_derivative_central  Open_LOG__median  ...
\
A                                     0.000388          3.640009  ...

AAPL                                 -0.000350          3.155702  ...

AB                                   -0.000013          2.678818  ...

ABBV                                  0.000071          3.666718  ...

ABCB                                  0.000179          3.136306  ...


      Open_LOG__fourier_entropy__bins_5
Open_LOG__fourier_entropy__bins_10  \
A                              0.047540
0.047540
AAPL                           0.047540
0.047540
AB                             0.095013
0.142417
ABBV                           0.047540
0.095013
ABCB                           0.095013
0.095013

      Open_LOG__fourier_entropy__bins_100  \
A                                0.210003
AAPL                             0.210003
AB                               0.426832
ABBV                             0.225655
ABCB                             0.238762

      Open_LOG__permutation_entropy__dimension_3__tau_1  \
A                                              1.703685
AAPL                                           1.606687
AB                                             1.699796
```

```
ABBV                                           1.268576
ABCB                                           1.736663

      Open_LOG__permutation_entropy__dimension_4__tau_1  \
A                                              2.868977
AAPL                                           2.710202
AB                                             2.881841
ABBV                                           2.090456
ABCB                                           2.972795

      Open_LOG__permutation_entropy__dimension_5__tau_1  \
A                                              3.985898
AAPL                                           3.765929
AB                                             3.992900
ABBV                                           2.826554
ABCB                                           4.162395

      Open_LOG__permutation_entropy__dimension_6__tau_1  \
A                                              4.740089
AAPL                                           4.525950
AB                                             4.705068
ABBV                                           3.266698
ABCB                                           4.969204

      Open_LOG__permutation_entropy__dimension_7__tau_1  \
A                                              5.092092
AAPL                                           4.966029
AB                                             5.026873
ABBV                                           3.559127
ABCB                                           5.265551

      Open_LOG__query_similarity_count__query_None__threshold_0.0  \
A                                                   NaN
AAPL                                                NaN
AB                                                  NaN
ABBV                                                NaN
ABCB                                                NaN

      Open_LOG__mean_n_absolute_max__number_of_maxima_7
A                                              5.053794
AAPL                                           5.445788
AB                                             3.700294
ABBV                                           5.239863
ABCB                                           4.162039

[5 rows x 777 columns]
```

Şirket Verilerine Endüstri Etiketi (Label) Eklenmesi

Neden Bu İşlemi Yaptık?

Elimizde 777 farklı şirketin piyasa açılış değerlerini içeren `extracted_data` adlı veri seti bulunmaktadır. Ancak, bu şirketlerin hangi sektöre ait olduğunu belirten bir sütun bulunmamaktadır. Bu bilgiyi eklemek, **makine öğrenmesi modellerinin** şirketlerin sektör bazlı analizlerini yapabilmesini sağlar.

Şirketlerin sektörel bilgileri, `combined_data_encoded` veri setinde **one-hot encoding** formatında bulunmaktadır. Bu veri setinde sektörler şu şekilde gösterilmektedir:

- `Sector_Technology` → **True** ise Teknoloji Sektörü
- `Sector_Financials` → **True** ise Finans Sektörü
- `Sector_Healthcare` → **True** ise Sağlık Sektörü

Bu bilgileri kullanarak her şirkete **0, 1 veya 2** olacak şekilde bir **Label** (etiket) sütunu ekledik:

- **0** → Teknoloji Şirketleri
- **1** → Finans Şirketleri
- **2** → Sağlık Şirketleri

Bu sayede `extracted_data` veri setimizde her şirketin ait olduğu sektörü belirten bir `Label` sütunu oluşturulmuş oldu.

Nasıl Yaptık?

1. Şirket-Sektör Bilgisini Çıkar

`combined_data_encoded` veri setinden her şirketin (Ticker) sektörünü belirledik:

```
# Şirketlerin sektörlerini içeren bir DataFrame oluştur
ticker_to_label = combined_data_encoded[['Ticker',
'Sector_Technology', 'Sector_Financials',
'Sector_Healthcare']].drop_duplicates()
```

2. Sektörleri 0, 1, 2 Olarak Kodla

Her sektör için aşağıdaki etiketleri atadık:

```
# Şirketlerin sektörlerini belirleyen fonksiyon
def get_sector_label(row):
    if row['Sector_Financials']:
        return 0  # Teknoloji
    elif row['Sector_Healthcare']:
        return 1  # Finans
    elif row['Sector_Technology']:
        return 2  # Sağlık
    return -1  # Hata kontrolü için

# Yeni Label sütununu ekledik
ticker_to_label['Label'] = ticker_to_label.apply(get_sector_label,
axis=1)

# Sadece farklı Ticker ve Label değerlerini al
ticker_to_label_unique = ticker_to_label[['Ticker',
'Label']].drop_duplicates()

# Eğer şirketler arasında tekrar varsa onları dropla
len(ticker_to_label_unique['Ticker'].unique())
```

3. Şirket İsimleri ile Label Eşleştirmesi Yap

`extracted_data` veri setinde bulunan **row, (index)** satırı, şirket isimlerini içeriyordu. Bu sütunu kullanarak sektör etiketlerini `extracted_data` içerisine ekledik:

```
# Şirket isimleri ile sektörleri eşleştirerek extracted_data'ya Label
sütununu ekleyelim
extracted_data = extracted_data.merge(ticker_to_label_unique,
left_index=True, right_on='Ticker', how='left')
```

4. Eksik Verileri Kontrol Et ve Gereksiz Sütunları Kaldır

Her şirket için bir etiket olup olmadığını kontrol ettik:

```
# Gereksiz fazlalıkları dropladım ve index kısmını resetledim
extracted_data = extracted_data.drop(['Label_x', 'Ticker'],
axis=1).reset_index(drop=True)
```

Sonuç

Bu işlemler sonucunda **extracted_data** veri setine **Label** sütunu eklendi. Bu sütun, her şirketin hangi sektöre ait olduğunu belirtiyor (**0 = Finans, 1 = Sağlık, 2 = Teknoloji**). Böylece, sektör bazlı analizler ve makine öğrenmesi modelleri için daha anlamlı bir veri seti elde edilmiş oldu.

```
combined_data_encoded =
pd.read_csv('../data/processed_data/encoded_combined_data.csv')

# Şirketlerin sektörlerini belirleyen fonksiyon
def get_sector_label(row):
    if row['Sector_Financials']:
        return 0  # Finans
    elif row['Sector_Healthcare']:
        return 1  # Sağlık
    elif row['Sector_Technology']:
        return 2  # Teknoloji
    return -1  # Hata kontrolü için

# Şirketlerin sektörlerini içeren bir DataFrame oluştur
ticker_to_label = combined_data_encoded[['Ticker',
'Sector_Financials', 'Sector_Healthcare',
'Sector_Technology']].drop_duplicates()

# Yeni Label sütunu ekle
ticker_to_label['Label'] = ticker_to_label.apply(get_sector_label,
axis=1)

# Sadece farklı Ticker ve Label değerlerini al
ticker_to_label_unique = ticker_to_label[['Ticker',
'Label']].drop_duplicates()

# Eğer şirketler arasında tekrar varsa onları dropla
len(ticker_to_label_unique['Ticker'].unique())
```

```
900
```

```
# Şirket isimleri ile sektörleri eşleştirerek extracted_data'ya Label
sütununu ekleyelim
extracted_data = extracted_data.merge(ticker_to_label_unique,
left_index=True, right_on='Ticker', how='left')

# Gereksiz fazlalıkları dropladım ve index kısmını resetledim
extracted_data = extracted_data.drop(['Ticker'],
axis=1).reset_index(drop=True)

# Eşleşmeyen şirketler olup olmadığını kontrol et (opsiyonel)
print(extracted_data['Label'].isna().sum(), "şirketin sektörü
bulunamadı.")

extracted_data.head()
```

```
0 şirketin sektörü bulunamadı.
```

```
   Open_LOG__variance_larger_than_standard_deviation  \
0                                                0.0
1                                                1.0
2                                                0.0
3                                                0.0
4                                                0.0

   Open_LOG__has_duplicate_max  Open_LOG__has_duplicate_min  \
0                          0.0                          0.0
1                          0.0                          0.0
2                          0.0                          0.0
3                          0.0                          1.0
4                          0.0                          0.0
```

```
   Open_LOG__has_duplicate  Open_LOG__sum_values  Open_LOG__abs_energy
\
0                      0.0            913.803852           3586.251889

1                      0.0            764.921929           2870.542166

2                      0.0            657.495086           1859.250052

3                      1.0            916.311508           3591.156249

4                      0.0            753.473377           2445.777047


   Open_LOG__mean_abs_change  Open_LOG__mean_change  \
0                   0.063168               0.009434
1                   0.066749               0.019755
2                   0.062410               0.005599
3                   0.034470               0.008760
4                   0.073705               0.006244

   Open_LOG__mean_second_derivative_central  Open_LOG__median  ...  \
0                                  0.000388          3.640009  ...
1                                 -0.000350          3.155702  ...
2                                 -0.000013          2.678818  ...
3                                  0.000071          3.666718  ...
4                                  0.000179          3.136306  ...

   Open_LOG__fourier_entropy__bins_10
Open_LOG__fourier_entropy__bins_100  \
0                            0.047540
0.210003
1                            0.047540
0.210003
2                            0.142417
0.426832
3                            0.095013
0.225655
4                            0.095013
0.238762

   Open_LOG__permutation_entropy__dimension_3__tau_1  \
0                                           1.703685
1                                           1.606687
2                                           1.699796
3                                           1.268576
4                                           1.736663

   Open_LOG__permutation_entropy__dimension_4__tau_1  \
0                                           2.868977
```

```
1                                           2.710202
2                                           2.881841
3                                           2.090456
4                                           2.972795

   Open_LOG__permutation_entropy__dimension_5__tau_1  \
0                                           3.985898
1                                           3.765929
2                                           3.992900
3                                           2.826554
4                                           4.162395

   Open_LOG__permutation_entropy__dimension_6__tau_1  \
0                                           4.740089
1                                           4.525950
2                                           4.705068
3                                           3.266698
4                                           4.969204

   Open_LOG__permutation_entropy__dimension_7__tau_1  \
0                                           5.092092
1                                           4.966029
2                                           5.026873
3                                           3.559127
4                                           5.265551

   Open_LOG__query_similarity_count__query_None__threshold_0.0  \
0                                                NaN
1                                                NaN
2                                                NaN
3                                                NaN
4                                                NaN

   Open_LOG__mean_n_absolute_max__number_of_maxima_7  Label
0                                           5.053794      1
1                                           5.445788      2
2                                           3.700294      0
3                                           5.239863      1
4                                           4.162039      0

[5 rows x 778 columns]
```

```
extracted_data.columns
```

```
Index(['Open_LOG__variance_larger_than_standard_deviation',
       'Open_LOG__has_duplicate_max', 'Open_LOG__has_duplicate_min',
       'Open_LOG__has_duplicate', 'Open_LOG__sum_values',
       'Open_LOG__abs_energy', 'Open_LOG__mean_abs_change',
       'Open_LOG__mean_change',
'Open_LOG__mean_second_derivative_central',
       'Open_LOG__median',
       ...
       'Open_LOG__fourier_entropy__bins_10',
       'Open_LOG__fourier_entropy__bins_100',
       'Open_LOG__permutation_entropy__dimension_3__tau_1',
       'Open_LOG__permutation_entropy__dimension_4__tau_1',
       'Open_LOG__permutation_entropy__dimension_5__tau_1',
       'Open_LOG__permutation_entropy__dimension_6__tau_1',
       'Open_LOG__permutation_entropy__dimension_7__tau_1',
       'Open_LOG__query_similarity_count__query_None__threshold_0.0',
       'Open_LOG__mean_n_absolute_max__number_of_maxima_7', 'Label'],
      dtype='object', length=778)
```

```
# Boş değerler içeren sütunları bulalım
for col in extracted_data.columns:
    if extracted_data[col].isna().sum() > 0:
        print(f"Sütun ismi: {col}, boş değer sayısı:
{extracted_data[col].isna().sum()}")
```

```
Sütun ismi: Open_LOG__friedrich_coefficients__coeff_0__m_3__r_30, boş
değer sayısı: 187
Sütun ismi: Open_LOG__friedrich_coefficients__coeff_1__m_3__r_30, boş
değer sayısı: 187
Sütun ismi: Open_LOG__friedrich_coefficients__coeff_2__m_3__r_30, boş
değer sayısı: 187
Sütun ismi: Open_LOG__friedrich_coefficients__coeff_3__m_3__r_30, boş
değer sayısı: 187
Sütun ismi: Open_LOG__max_langevin_fixed_point__m_3__r_30, boş değer
sayısı: 187
Sütun ismi:
Open_LOG__query_similarity_count__query_None__threshold_0.0, boş değer
sayısı: 900
```

```
# Drop empty columns
extracted_data = extracted_data.dropna(axis=1, how='any')
```

```
extracted_data['Label'].unique()
```

```
array([1, 2, 0], dtype=int64)
```

```
extracted_data.to_csv('../data/processed_data/
extracted_labeled_features.xlsx', index= False)
```

Öznitelik Seçme:

- Çıkarılan özellikler arasından en önemlileri seçmek için
  - L1 regularization (Lasso),
  - Recursive Feature Elimination (RFE)

RFE Sonuçları:

RFE, en önemli 385 özelliği seçmiştir. Aşağıda, Random Forest modeline göre sıralanan en önemli 20 özellik yer almaktadır:

1. **Open_LOG__mean_abs_change**: 0.009998
2. **Open_LOG__fft_coefficient__attr_"real"__coeff_7**: 0.009923
3. **Open_LOG__fft_coefficient__attr_"angle"__coeff_58**: 0.008836
4. **Open_LOG__sum_of_reoccurring_values**: 0.008206
5. **Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.8**: 0.008201

En önemli 20 özellik kullanılarak elde edilen doğruluk: 0.7167

Çapraz doğrulama skorları: [0.70833333, 0.70138889, 0.625, 0.69444444, 0.69444444]

Ortalama çapraz doğrulama skoru: 0.6847

L1 Sonuçları (Lasso):

Lasso (L1 regularization) ile 166 özellik arasından en önemli 20 özellik seçilmiştir. Aşağıda bu özellikler, Lasso katsayılarına göre sıralanmıştır:

1. **Open_LOG__fft_coefficient__attr_"real"__coeff_44**: 2.795158
2. **Open_LOG__fft_coefficient__attr_"real"__coeff_34**: 1.144467
3. **Open_LOG__fft_coefficient__attr_"real"__coeff_18**: 0.940388
4. **Open_LOG__fft_coefficient__attr_"real"__coeff_17**: 0.752983
5. **Open_LOG__fft_coefficient__attr_"imag"__coeff_20**: 0.749958

En önemli 20 özellik kullanılarak elde edilen doğruluk (Lasso): 0.6944

Çapraz doğrulama skorları (Lasso): [0.72222222, 0.76388889, 0.75694444, 0.68055556, 0.72222222]

Ortalama çapraz doğrulama skoru (Lasso): 0.7292

Sonuçların Karşılaştırılması

- **RFE** yöntemi ile seçilen en önemli 20 özelliği kullanarak elde edilen doğruluk (0.7167) ve çapraz doğrulama skoru (0.6847), **Lasso** ile seçilen özellikler ile karşılaştırıldığında daha iyi bir performans göstermektedir.
- **Lasso** yönteminin doğruluk değeri 0.6944 ile biraz daha düşük olsa da, çapraz doğrulama skorları (0.7292) biraz daha yüksek olmuştur.
- Her iki yöntem de önemli özellikleri seçmiş ve farklı doğruluk değerleri elde edilmiştir, ancak genel olarak RFE ile elde edilen sonuçlar biraz daha başarılı olmuştur.

```
# Model ve Veri İşleme Araçları
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report

# Sınıflandırma Modelleri
from sklearn.ensemble import RandomForestClassifier, 
GradientBoostingClassifier
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

# Özellik Seçimi ve Boyut İndirgeme
from sklearn.feature_selection import RFE
from sklearn.decomposition import PCA

# Ekstra: XGBoost Modeli
from xgboost import XGBClassifier

# Veriyi yükle
extracted_labeled_data =
pd.read_csv('../data/processed_data/extracted_labeled_features.xlsx')

# Train ve Test için sütunları seç
X = extracted_labeled_data.drop(columns=['Label'])
y = extracted_labeled_data['Label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)

### 1. Recursive Feature Elimination (RFE) ###
selector_rfe = RFE(RandomForestClassifier(random_state=10))
selector_rfe.fit(X_train, y_train)

# Transform datasets
X_train_rfe = selector_rfe.transform(X_train)
X_test_rfe = selector_rfe.transform(X_test)

# Seçilen feature'ları yazdır
selected_features_rfe = X_train.columns[selector_rfe.get_support()]
print(f"RFE Selected Features ({len(selected_features_rfe)}):
{list(selected_features_rfe)}")
```

```
RFE Selected Features (385): ['Open_LOG__abs_energy',
'Open_LOG__mean_abs_change',
'Open_LOG__mean_second_derivative_central', 'Open_LOG__median',
'Open_LOG__variance', 'Open_LOG__skewness', 'Open_LOG__kurtosis',
'Open_LOG__absolute_sum_of_changes',
'Open_LOG__longest_strike_above_mean', 'Open_LOG__count_below_mean',
'Open_LOG__last_location_of_maximum',
'Open_LOG__first_location_of_maximum',
'Open_LOG__last_location_of_minimum',
'Open_LOG__percentage_of_reoccurring_values_to_all_values',
'Open_LOG__percentage_of_reoccurring_datapoints_to_all_datapoints',
'Open_LOG__sum_of_reoccurring_values',
'Open_LOG__sum_of_reoccurring_data_points', 'Open_LOG__c3__lag_1',
'Open_LOG__cid_ce__normalize_True',
'Open_LOG__cid_ce__normalize_False', 'Open_LOG__quantile__q_0.6',
'Open_LOG__quantile__q_0.7', 'Open_LOG__quantile__q_0.8',
'Open_LOG__quantile__q_0.9', 'Open_LOG__autocorrelation__lag_1',
'Open_LOG__autocorrelation__lag_2',
'Open_LOG__autocorrelation__lag_3',
'Open_LOG__autocorrelation__lag_4',
'Open_LOG__autocorrelation__lag_5',
'Open_LOG__autocorrelation__lag_6',
'Open_LOG__autocorrelation__lag_7',
'Open_LOG__autocorrelation__lag_8',
'Open_LOG__autocorrelation__lag_9',
'Open_LOG__agg_autocorrelation__f_agg_"mean"__maxlag_40',
'Open_LOG__agg_autocorrelation__f_agg_"median"__maxlag_40',
'Open_LOG__agg_autocorrelation__f_agg_"var"__maxlag_40',
'Open_LOG__partial_autocorrelation__lag_1',
'Open_LOG__partial_autocorrelation__lag_2',
'Open_LOG__partial_autocorrelation__lag_3',
'Open_LOG__partial_autocorrelation__lag_6',
'Open_LOG__partial_autocorrelation__lag_8',
'Open_LOG__partial_autocorrelation__lag_9',
'Open_LOG__cwt_coefficients__coeff_0__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_0__w_5__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_0__w_10__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_1__w_20__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_2__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_3__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_3__w_20__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_6__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_6__w_5__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_7__w_20__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_9__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_10__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_10__w_20__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_11__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_12__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_14__w_2__widths_(2, 5, 10, 20)',
'Open_LOG__cwt_coefficients__coeff_14__w_5__widths_(2, 5, 10, 20)',
'Open_LOG__spkt_welch_density__coeff_2',
'Open_LOG__spkt_welch_density__coeff_8',
'Open_LOG__ar_coefficient__coeff_1__k_10',
'Open_LOG__ar_coefficient__coeff_2__k_10',
'Open_LOG__ar_coefficient__coeff_3__k_10',
'Open_LOG__ar_coefficient__coeff_5__k_10',
'Open_LOG__ar_coefficient__coeff_6__k_10',
'Open_LOG__ar_coefficient__coeff_10__k_10',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_0.2__ql_0.0
',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_0.4__ql_0.0
',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.4__ql_0.0'
```

```
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.4__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.4__ql_0.0',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.6__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.6__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.6__ql_0.0',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_0.8__ql_0.0
',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.8__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.8__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.8__ql_0.0',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.0'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_1.0__ql_0.0',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.4__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.4__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.4__ql_0.2',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.6__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.6__ql_0.2',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_0.8__ql_0.2
',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.8__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.8__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.8__ql_0.2',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.2'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_1.0__ql_0.2',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_0.6__ql_0.4'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.6__ql_0.4'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_0.6__ql_0.4',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_0.8__ql_0.4
',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.8__ql_0.4'
```

```
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.4'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.4'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_1.0__ql_0.4',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_0.8__ql_0.6'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.6'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.6'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_1.0__ql_0.6',
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_False__qh_1.0__ql_0.8
',
'Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.8'
,
'Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.8'
,
'Open_LOG__change_quantiles__f_agg_"var"__isabs_True__qh_1.0__ql_0.8',
'Open_LOG__fft_coefficient__attr_"real"__coeff_1',
'Open_LOG__fft_coefficient__attr_"real"__coeff_2',
'Open_LOG__fft_coefficient__attr_"real"__coeff_3',
'Open_LOG__fft_coefficient__attr_"real"__coeff_4',
'Open_LOG__fft_coefficient__attr_"real"__coeff_5',
'Open_LOG__fft_coefficient__attr_"real"__coeff_7',
'Open_LOG__fft_coefficient__attr_"real"__coeff_8',
'Open_LOG__fft_coefficient__attr_"real"__coeff_9',
'Open_LOG__fft_coefficient__attr_"real"__coeff_11',
'Open_LOG__fft_coefficient__attr_"real"__coeff_12',
'Open_LOG__fft_coefficient__attr_"real"__coeff_13',
'Open_LOG__fft_coefficient__attr_"real"__coeff_14',
'Open_LOG__fft_coefficient__attr_"real"__coeff_15',
'Open_LOG__fft_coefficient__attr_"real"__coeff_16',
'Open_LOG__fft_coefficient__attr_"real"__coeff_17',
'Open_LOG__fft_coefficient__attr_"real"__coeff_18',
'Open_LOG__fft_coefficient__attr_"real"__coeff_19',
'Open_LOG__fft_coefficient__attr_"real"__coeff_20',
'Open_LOG__fft_coefficient__attr_"real"__coeff_21',
'Open_LOG__fft_coefficient__attr_"real"__coeff_22',
'Open_LOG__fft_coefficient__attr_"real"__coeff_23',
'Open_LOG__fft_coefficient__attr_"real"__coeff_24',
'Open_LOG__fft_coefficient__attr_"real"__coeff_25',
'Open_LOG__fft_coefficient__attr_"real"__coeff_26',
'Open_LOG__fft_coefficient__attr_"real"__coeff_27',
'Open_LOG__fft_coefficient__attr_"real"__coeff_28',
'Open_LOG__fft_coefficient__attr_"real"__coeff_29',
'Open_LOG__fft_coefficient__attr_"real"__coeff_30',
'Open_LOG__fft_coefficient__attr_"real"__coeff_31',
'Open_LOG__fft_coefficient__attr_"real"__coeff_32',
'Open_LOG__fft_coefficient__attr_"real"__coeff_33',
'Open_LOG__fft_coefficient__attr_"real"__coeff_34',
'Open_LOG__fft_coefficient__attr_"real"__coeff_36',
'Open_LOG__fft_coefficient__attr_"real"__coeff_37',
'Open_LOG__fft_coefficient__attr_"real"__coeff_39',
'Open_LOG__fft_coefficient__attr_"real"__coeff_40',
'Open_LOG__fft_coefficient__attr_"real"__coeff_41',
'Open_LOG__fft_coefficient__attr_"real"__coeff_42',
'Open_LOG__fft_coefficient__attr_"real"__coeff_44',
'Open_LOG__fft_coefficient__attr_"real"__coeff_45',
'Open_LOG__fft_coefficient__attr_"real"__coeff_46',
'Open_LOG__fft_coefficient__attr_"real"__coeff_48',
'Open_LOG__fft_coefficient__attr_"real"__coeff_51',
'Open_LOG__fft_coefficient__attr_"real"__coeff_52',
'Open_LOG__fft_coefficient__attr_"real"__coeff_53',
'Open_LOG__fft_coefficient__attr_"real"__coeff_54',
'Open_LOG__fft_coefficient__attr_"real"__coeff_56',
'Open_LOG__fft_coefficient__attr_"real"__coeff_57',
'Open_LOG__fft_coefficient__attr_"real"__coeff_58',
'Open_LOG__fft_coefficient__attr_"real"__coeff_61',
'Open_LOG__fft_coefficient__attr_"real"__coeff_62',
'Open_LOG__fft_coefficient__attr_"real"__coeff_63',
'Open_LOG__fft_coefficient__attr_"real"__coeff_66',
'Open_LOG__fft_coefficient__attr_"real"__coeff_67',
'Open_LOG__fft_coefficient__attr_"real"__coeff_68',
'Open_LOG__fft_coefficient__attr_"real"__coeff_71',
'Open_LOG__fft_coefficient__attr_"real"__coeff_72',
'Open_LOG__fft_coefficient__attr_"real"__coeff_74',
'Open_LOG__fft_coefficient__attr_"real"__coeff_75',
'Open_LOG__fft_coefficient__attr_"real"__coeff_78',
'Open_LOG__fft_coefficient__attr_"real"__coeff_80',
'Open_LOG__fft_coefficient__attr_"real"__coeff_83',
'Open_LOG__fft_coefficient__attr_"real"__coeff_84',
'Open_LOG__fft_coefficient__attr_"real"__coeff_88',
'Open_LOG__fft_coefficient__attr_"real"__coeff_90',
'Open_LOG__fft_coefficient__attr_"real"__coeff_91',
'Open_LOG__fft_coefficient__attr_"real"__coeff_93',
'Open_LOG__fft_coefficient__attr_"real"__coeff_94',
'Open_LOG__fft_coefficient__attr_"real"__coeff_97',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_1',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_5',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_6',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_8',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_9',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_14',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_16',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_20',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_21',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_22',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_24',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_27',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_28',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_30',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_32',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_40',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_41',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_43',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_44',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_45',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_47',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_48',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_49',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_52',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_53',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_54',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_58',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_59',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_63',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_64',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_66',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_68',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_72',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_73',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_74',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_75',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_76',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_77',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_79',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_80',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_81',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_82',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_85',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_87',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_88',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_89',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_90',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_91',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_92',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_93',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_94',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_95',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_97',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_98',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_99',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_1',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_3',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_4',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_5',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_6',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_7',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_9',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_10',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_12',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_14',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_15',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_16',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_18',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_19',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_21',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_22',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_24',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_27',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_28',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_30',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_32',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_35',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_37',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_40',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_43',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_45',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_47',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_48',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_52',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_56',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_58',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_59',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_62',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_64',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_67',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_71',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_72',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_75',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_76',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_77',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_79',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_80',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_84',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_86',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_87',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_88',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_90',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_91',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_93',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_94',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_95',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_97',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_1',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_2',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_3',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_4',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_6',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_7',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_8',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_9',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_11',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_13',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_14',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_15',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_16',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_17',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_18',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_19',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_20',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_21',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_24',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_25',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_26',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_27',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_28',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_29',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_30',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_31',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_32',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_34',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_35',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_36',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_37',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_40',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_41',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_42',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_44',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_45',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_47',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_48',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_52',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_55',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_56',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_57',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_58',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_61',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_62',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_63',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_64',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_66',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_71',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_72',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_73',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_74',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_75',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_77',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_78',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_81',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_82',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_83',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_84',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_87',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_88',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_89',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_90',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_91',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_93',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_94',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_95',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_96',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_97',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_99',
'Open_LOG__fft_aggregated__aggtype_"centroid"',
'Open_LOG__fft_aggregated__aggtype_"variance"',
'Open_LOG__fft_aggregated__aggtype_"skew"',
'Open_LOG__fft_aggregated__aggtype_"kurtosis"',
'Open_LOG__linear_trend__attr_"slope"',
'Open_LOG__linear_trend__attr_"stderr"',
'Open_LOG__agg_linear_trend__attr_"rvalue"__chunk_len_5__f_agg_"var"',
'Open_LOG__agg_linear_trend__attr_"rvalue"__chunk_len_10__f_agg_"var"'
,
'Open_LOG__agg_linear_trend__attr_"intercept"__chunk_len_5__f_agg_"max
"',
'Open_LOG__agg_linear_trend__attr_"intercept"__chunk_len_10__f_agg_"me
an"',
'Open_LOG__agg_linear_trend__attr_"intercept"__chunk_len_50__f_agg_"mi
n"',
'Open_LOG__agg_linear_trend__attr_"slope"__chunk_len_5__f_agg_"var"',
'Open_LOG__agg_linear_trend__attr_"slope"__chunk_len_50__f_agg_"var"',
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_5__f_agg_"var"',
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_10__f_agg_"min"'
,
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_10__f_agg_"mean"
',
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_50__f_agg_"min"'
,
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_50__f_agg_"mean"
',
'Open_LOG__agg_linear_trend__attr_"stderr"__chunk_len_50__f_agg_"var"'
,
'Open_LOG__energy_ratio_by_chunks__num_segments_10__segment_focus_4',
'Open_LOG__energy_ratio_by_chunks__num_segments_10__segment_focus_5',
'Open_LOG__energy_ratio_by_chunks__num_segments_10__segment_focus_6',
'Open_LOG__energy_ratio_by_chunks__num_segments_10__segment_focus_7',
'Open_LOG__energy_ratio_by_chunks__num_segments_10__segment_focus_9',
'Open_LOG__ratio_beyond_r_sigma__r_0.5',
'Open_LOG__permutation_entropy__dimension_3__tau_1',
'Open_LOG__permutation_entropy__dimension_5__tau_1',
'Open_LOG__permutation_entropy__dimension_6__tau_1',
'Open_LOG__mean_n_absolute_max__number_of_maxima_7']

# RFE'nin seçtiği feature'ları scores sütununa göre listele
results = pd.DataFrame(
    data=selector_rfe.estimator_.feature_importances_,
    index=selected_features_rfe,
    columns=['scores']
)
```

- Çıkardığım 385 feature'ın hepsini kullanarak model eğitirsem ortalama 0.73 accuracy alıyordum
- Aynı şekilde en iyi 20 feature'ı seçip model eğitirsem ortalama 0.71 accuracy alıyorum.
- Daha az feature ile aynı oranda doğruluk aldığım için en iyi 20 feature'ı seçtim

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score

# Sonuçları büyükten küçüğe sırala
sorted_results_desc = results.sort_values(by='scores',
ascending=False)

# En önemli 20 feature'ı seç
top_20_features_rf = sorted_results_desc.index[:20]

print(f"En önemli 20 feature (Random Forest önem değerlerine göre):")
for i, feature in enumerate(top_20_features_rf, 1):
    score = sorted_results_desc.loc[feature, 'scores']
    print(f"{i}. {feature}: {score:.6f}")

En önemli 20 feature (Random Forest önem değerlerine göre):
1. Open_LOG__mean_abs_change: 0.009998
2. Open_LOG__fft_coefficient__attr_"real"__coeff_7: 0.009923
3. Open_LOG__fft_coefficient__attr_"angle"__coeff_58: 0.008836
4. Open_LOG__sum_of_reoccurring_values: 0.008206
5.
Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.8:
0.008201
6.
Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.2:
0.007394
7. Open_LOG__fft_coefficient__attr_"real"__coeff_94: 0.007320
```

```
8. Open_LOG__fft_coefficient__attr_"real"__coeff_34: 0.007298
9.
Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.0:
0.007214
10. Open_LOG__percentage_of_reoccurring_values_to_all_values: 0.007091
11.
Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.6:
0.007058
12.
Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.4:
0.006988
13. Open_LOG__fft_coefficient__attr_"real"__coeff_11: 0.006836
14. Open_LOG__fft_coefficient__attr_"real"__coeff_90: 0.006485
15.
Open_LOG__change_quantiles__f_agg_"mean"__isabs_True__qh_1.0__ql_0.4:
0.006401
16. Open_LOG__absolute_sum_of_changes: 0.006284
17. Open_LOG__fft_coefficient__attr_"imag"__coeff_73: 0.006101
18. Open_LOG__fft_coefficient__attr_"real"__coeff_44: 0.005971
19. Open_LOG__fft_coefficient__attr_"imag"__coeff_97: 0.005838
20.
Open_LOG__change_quantiles__f_agg_"var"__isabs_False__qh_1.0__ql_0.6:
0.005837
```

```python
# Bu feature'ları kullanarak yeni veri setleri oluştur
X_train_top20_rf = X_train[top_20_features_rf]
X_test_top20_rf = X_test[top_20_features_rf]

# Model oluştur ve eğit
rf_model_top20 = RandomForestClassifier(random_state=10)
rf_model_top20.fit(X_train_top20_rf, y_train)

# Test seti üzerinde tahmin yap
y_pred_top20_rf = rf_model_top20.predict(X_test_top20_rf)

### 2. L1 Regularization (Lasso) ###
lasso = LogisticRegression(penalty="l1", solver="liblinear",
random_state=10)
lasso.fit(X_train, y_train)

# NaN veya 0 olmayan feature'ları, Lasso ile seç
selected_features_lasso = X_train.columns[lasso.coef_[0] != 0]
X_train_lasso = X_train[selected_features_lasso]
X_test_lasso = X_test[selected_features_lasso]

print(f"Lasso Selected Features ({len(selected_features_lasso)}):
{list(selected_features_lasso)}")
```

```
Lasso Selected Features (166): ['Open_LOG__sum_values',
'Open_LOG__abs_energy', 'Open_LOG__length',
'Open_LOG__longest_strike_above_mean', 'Open_LOG__count_above_mean',
'Open_LOG__count_below_mean', 'Open_LOG__sum_of_reoccurring_values',
'Open_LOG__sum_of_reoccurring_data_points', 'Open_LOG__c3__lag_1',
'Open_LOG__c3__lag_2', 'Open_LOG__c3__lag_3',
'Open_LOG__number_cwt_peaks__n_1', 'Open_LOG__number_cwt_peaks__n_5',
'Open_LOG__number_peaks__n_1', 'Open_LOG__number_peaks__n_3',
'Open_LOG__number_peaks__n_10',
'Open_LOG__spkt_welch_density__coeff_2',
'Open_LOG__fft_coefficient__attr_"real"__coeff_0',
'Open_LOG__fft_coefficient__attr_"real"__coeff_1',
'Open_LOG__fft_coefficient__attr_"real"__coeff_2',
'Open_LOG__fft_coefficient__attr_"real"__coeff_3',
'Open_LOG__fft_coefficient__attr_"real"__coeff_4',
'Open_LOG__fft_coefficient__attr_"real"__coeff_5',
'Open_LOG__fft_coefficient__attr_"real"__coeff_7',
'Open_LOG__fft_coefficient__attr_"real"__coeff_8',
'Open_LOG__fft_coefficient__attr_"real"__coeff_10',
'Open_LOG__fft_coefficient__attr_"real"__coeff_11',
'Open_LOG__fft_coefficient__attr_"real"__coeff_13',
'Open_LOG__fft_coefficient__attr_"real"__coeff_14',
'Open_LOG__fft_coefficient__attr_"real"__coeff_16',
'Open_LOG__fft_coefficient__attr_"real"__coeff_17',
'Open_LOG__fft_coefficient__attr_"real"__coeff_18',
'Open_LOG__fft_coefficient__attr_"real"__coeff_20',
'Open_LOG__fft_coefficient__attr_"real"__coeff_21',
'Open_LOG__fft_coefficient__attr_"real"__coeff_24',
'Open_LOG__fft_coefficient__attr_"real"__coeff_27',
'Open_LOG__fft_coefficient__attr_"real"__coeff_28',
'Open_LOG__fft_coefficient__attr_"real"__coeff_30',
'Open_LOG__fft_coefficient__attr_"real"__coeff_34',
'Open_LOG__fft_coefficient__attr_"real"__coeff_44',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_1',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_2',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_3',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_4',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_5',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_6',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_7',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_8',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_9',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_11',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_12',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_18',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_20',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_22',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_23',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_27',
'Open_LOG__fft_coefficient__attr_"imag"__coeff_34',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_0',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_1',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_2',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_4',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_6',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_9',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_10',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_17',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_24',
'Open_LOG__fft_coefficient__attr_"abs"__coeff_25',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_1',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_2',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_3',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_4',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_5',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_6',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_7',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_8',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_9',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_10',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_12',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_13',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_14',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_15',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_16',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_17',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_18',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_19',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_20',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_21',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_22',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_23',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_24',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_25',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_26',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_27',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_28',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_29',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_30',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_31',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_32',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_33',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_34',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_35',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_36',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_37',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_38',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_40',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_41',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_42',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_43',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_44',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_45',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_46',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_47',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_48',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_49',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_50',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_51',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_52',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_53',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_54',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_55',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_56',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_57',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_58',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_59',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_60',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_61',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_62',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_63',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_64',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_65',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_66',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_67',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_68',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_69',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_70',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_71',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_72',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_73',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_74',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_75',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_76',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_77',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_78',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_79',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_80',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_81',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_82',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_83',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_84',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_85',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_86',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_88',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_89',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_90',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_91',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_92',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_93',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_94',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_95',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_96',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_97',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_98',
'Open_LOG__fft_coefficient__attr_"angle"__coeff_99',
'Open_LOG__fft_aggregated__aggtype_"variance"',
'Open_LOG__range_count__max_1__min_-1',
'Open_LOG__range_count__max_1000000000000.0__min_0']

### 2. L1 Regularization (Lasso) için özellik önem değerlerini
hesaplayalım ###
# L1 katsayılarının mutlak değerlerini alarak özellik önem sıralaması
oluştur
feature_importance_lasso = pd.DataFrame({
    'scores': np.abs(lasso.coef_[0]),
    'names': X_train.columns
})
feature_importance_lasso = feature_importance_lasso.set_index('names')

# Sonuçları büyükten küçüğe sırala
sorted_results_lasso_desc =
feature_importance_lasso.sort_values(by='scores', ascending=False)

# En önemli 20 feature'ı seç
top_20_features_lasso = sorted_results_lasso_desc.index[:20]
print(f"En önemli 20 feature (Lasso katsayılarına göre):")
for i, feature in enumerate(top_20_features_lasso, 1):
    score = sorted_results_lasso_desc.loc[feature, 'scores']
    print(f"{i}. {feature}: {score:.6f}")

En önemli 20 feature (Lasso katsayılarına göre):
1. Open_LOG__fft_coefficient__attr_"real"__coeff_44: 2.795158
2. Open_LOG__fft_coefficient__attr_"real"__coeff_34: 1.144467
3. Open_LOG__fft_coefficient__attr_"real"__coeff_18: 0.940388
4. Open_LOG__fft_coefficient__attr_"real"__coeff_17: 0.752983
5. Open_LOG__fft_coefficient__attr_"imag"__coeff_20: 0.749958
6. Open_LOG__fft_coefficient__attr_"real"__coeff_24: 0.662814
7. Open_LOG__fft_coefficient__attr_"real"__coeff_20: 0.480475
8. Open_LOG__fft_coefficient__attr_"imag"__coeff_18: 0.461297
9. Open_LOG__fft_coefficient__attr_"abs"__coeff_25: 0.423871
10. Open_LOG__fft_coefficient__attr_"abs"__coeff_10: 0.419204
11. Open_LOG__fft_coefficient__attr_"imag"__coeff_8: 0.413403
12. Open_LOG__fft_coefficient__attr_"real"__coeff_16: 0.370956
13. Open_LOG__fft_coefficient__attr_"imag"__coeff_2: 0.338724
14. Open_LOG__fft_coefficient__attr_"real"__coeff_11: 0.326932
15. Open_LOG__fft_coefficient__attr_"imag"__coeff_27: 0.321631
16. Open_LOG__fft_coefficient__attr_"imag"__coeff_7: 0.301130
17. Open_LOG__fft_coefficient__attr_"abs"__coeff_24: 0.281229
```

```
18. Open_LOG__fft_coefficient__attr_"imag"__coeff_4: 0.277839
19. Open_LOG__fft_coefficient__attr_"real"__coeff_28: 0.274034
20. Open_LOG__fft_coefficient__attr_"real"__coeff_7: 0.271834

# Bu feature'ları kullanarak yeni veri setleri oluştur
X_train_top20_lasso = X_train[top_20_features_lasso]
X_test_top20_lasso = X_test[top_20_features_lasso]

# Model oluştur ve eğit (Lojistik Regresyon kullanıyoruz çünkü Lasso
bir özellik seçim yöntemidir)
lr_model_top20 = LogisticRegression(random_state=10)
lr_model_top20.fit(X_train_top20_lasso, y_train)

# Test seti üzerinde tahmin yap
y_pred_top20_lasso = lr_model_top20.predict(X_test_top20_lasso)

D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

# Doğruluk (accuracy) hesapla RFE
accuracy_top20_rf = accuracy_score(y_test, y_pred_top20_rf)
print(f"\nEn önemli 20 feature kullanılarak elde edilen doğruluk:
{accuracy_top20_rf:.4f}")

# Çapraz doğrulama yap RFE
cv_scores_top20_rf = cross_val_score(rf_model_top20, X_train_top20_rf,
y_train, cv=5, scoring='accuracy')
print(f"Çapraz doğrulama skorları: {cv_scores_top20_rf}")
print(f"Ortalama çapraz doğrulama skoru:
{cv_scores_top20_rf.mean():.4f}")

#----------------------------------------------------------------------
---------------------------------------------------

# Doğruluk (accuracy) hesapla Lasoo
accuracy_top20_lasso = accuracy_score(y_test, y_pred_top20_lasso)
print(f"\nEn önemli 20 feature kullanılarak elde edilen doğruluk
(Lasso): {accuracy_top20_lasso:.4f}")

# Çapraz doğrulama yap Lasso
cv_scores_top20_lasso = cross_val_score(lr_model_top20,
X_train_top20_lasso, y_train, cv=5, scoring='accuracy')
print(f"Çapraz doğrulama skorları (Lasso): {cv_scores_top20_lasso}")
print(f"Ortalama çapraz doğrulama skoru (Lasso):
{cv_scores_top20_lasso.mean():.4f}")
```

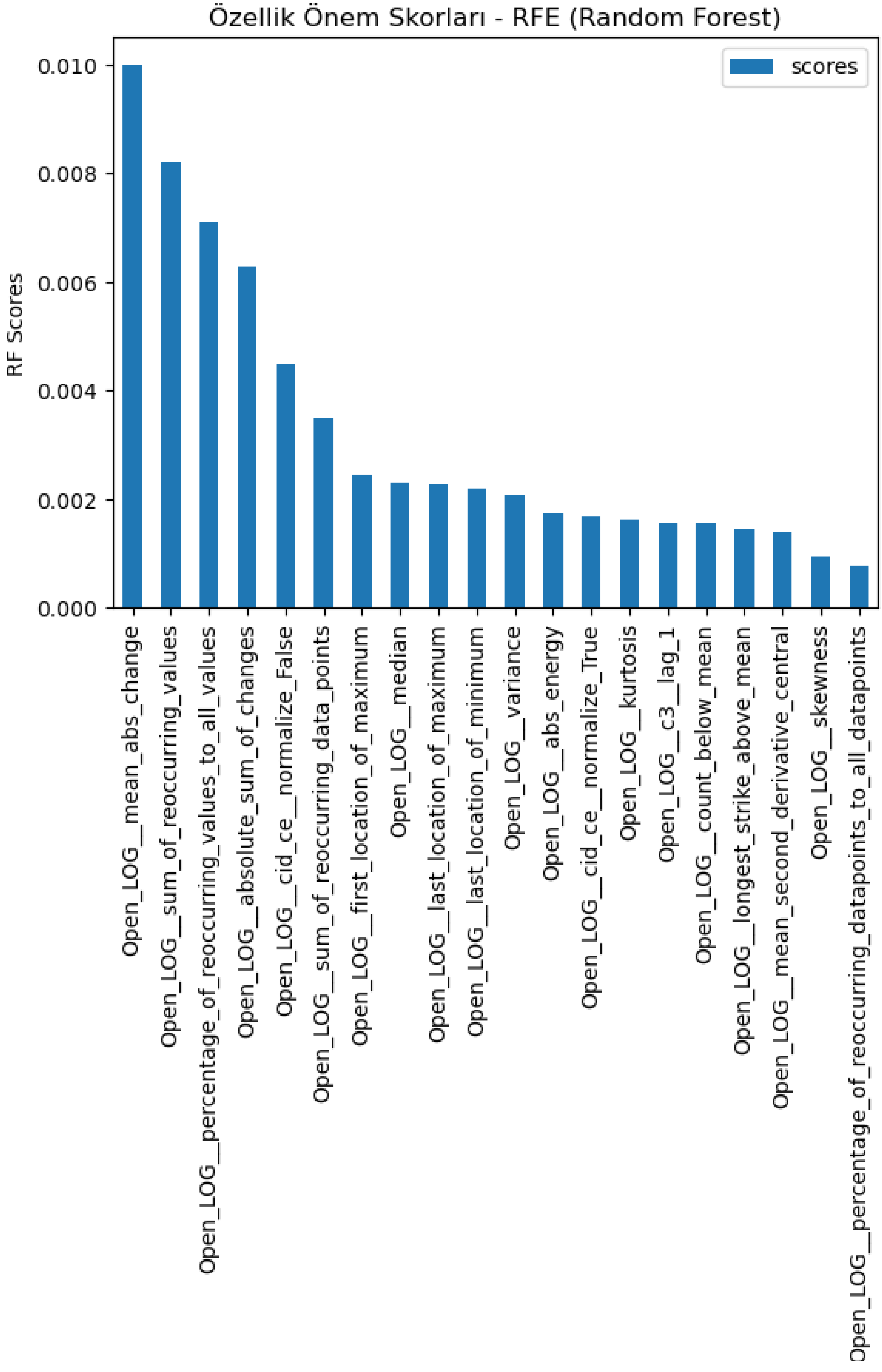
```
En önemli 20 feature kullanılarak elde edilen doğruluk: 0.7167
Çapraz doğrulama skorları: [0.70833333 0.70138889 0.625
0.69444444 0.69444444]
Ortalama çapraz doğrulama skoru: 0.6847

En önemli 20 feature kullanılarak elde edilen doğruluk (Lasso): 0.6944
Çapraz doğrulama skorları (Lasso): [0.72222222 0.76388889 0.75694444
0.68055556 0.72222222]
Ortalama çapraz doğrulama skoru (Lasso): 0.7292

D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
```

```
regression
  n_iter_i = _check_optimize_result(
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```python
# Feature importance values for RFE (Random Forest)
results_of_20_best_feature = pd.DataFrame(
    data=selector_rfe.estimator_.feature_importances_[:20],
    index=selected_features_rfe[:20],
    columns=['scores']
)
# Visualize importance scores
plt.figure(figsize=(14, 6))
results_of_20_best_feature.sort_values(by='scores',
ascending=False).plot.bar(
    title='Özellik Önem Skorları - RFE (Random Forest)'
)
plt.xlabel('Değişkenler')
plt.ylabel('RF Scores')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

```
C:\Users\only_\AppData\Local\Temp\ipykernel_8136\151639754.py:15:
UserWarning: Tight layout not applied. The bottom and top margins
cannot be made large enough to accommodate all axes decorations.
  plt.tight_layout()
```

```
<Figure size 1400x600 with 0 Axes>
```


Özellik Önem Skorları - RFE (Random Forest)
RF Scores
0.010
0.008
0.006
0.004
0.002
0.000
scores
Open_LOG__mean_abs_change
Open_LOG__sum_of_reoccurring_values
Open_LOG__percentage_of_reoccurring_values_to_all_values
Open_LOG__absolute_sum_of_changes
Open_LOG__cid_ce__normalize_False
Open_LOG__sum_of_reoccurring_data_points
Open_LOG__first_location_of_maximum
Open_LOG__median
Open_LOG__last_location_of_maximum
Open_LOG__last_location_of_minimum
Open_LOG__variance
Open_LOG__abs_energy
Open_LOG__cid_ce__normalize_True
Open_LOG__kurtosis
Open_LOG__c3__lag_1
Open_LOG__count_below_mean
Open_LOG__longest_strike_above_mean
Open_LOG__mean_second_derivative_central
Open_LOG__skewness
Open_LOG__percentage_of_reoccurring_datapoints_to_all_datapoints

```
import os
import joblib  # Modeli yüklemek için

# Kayıt dizinini oluştur (eğer yoksa)
save_dir = "../trained_models/"
os.makedirs(save_dir, exist_ok=True)

# Random Forest modelini kaydet
rf_model_path = os.path.join(save_dir, "rf_model_top20.pkl")
joblib.dump(rf_model_top20, rf_model_path)

# Logistic Regression modelini kaydet
lr_model_path = os.path.join(save_dir, "lr_model_top20_lasso.pkl")
joblib.dump(lr_model_top20, lr_model_path)

print(f"Modeller şu dizine kaydedildi: {save_dir}")
```

```
Modeller şu dizine kaydedildi: ../trained_models/
```

Model Geliştirme

```
# Gerekli kütüphaneleri içe aktaralım
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

# Label encoding (eğer label'lar string ise)
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Farklı modelleri tanımla
models = {
    # Temel modeller
    "Logistic Regression": LogisticRegression(max_iter=1000,
random_state=0),
    "SVC": SVC(random_state=0, probability=True),
    "Random Forest": RandomForestClassifier(n_estimators=100,
random_state=10),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100,
random_state=10),

    # Gelişmiş modeller
    "XGBoost": XGBClassifier(n_estimators=100, random_state=10,
use_label_encoder=False, eval_metric='logloss'),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(100, 50),
max_iter=1000, random_state=0),

    # Ek modeller
    "AdaBoost": AdaBoostClassifier(n_estimators=100, random_state=10),
    "Ridge Classifier": RidgeClassifier(random_state=0),
    "KNeighbors": KNeighborsClassifier(n_neighbors=5),
    "Gaussian NB": GaussianNB()
}

import os
import joblib  # Modeli yüklemek için

# Kayıt dizinini oluştur (eğer yoksa)
save_dir = "../trained_models/"
os.makedirs(save_dir, exist_ok=True)

# Sonuçları tutacak liste
results = []

# Her modeli eğit, test et ve accuracy score hesapla
for model_name, model in models.items():
    print(f"\n{model_name} eğitiliyor...")
    model.fit(X_train_rfe, y_train_encoded)  # Modeli eğit
    y_pred = model.predict(X_test_rfe)  # Test verisiyle tahmin yap

    # Model dosya adını oluştur ve kaydet
    model_filename = os.path.join(save_dir, f"{model_name.replace(' ',
'_')}.pkl")
    joblib.dump(model, model_filename)  # Modeli kaydet
    print(f"{model_name} kaydedildi: {model_filename}")

    acc = accuracy_score(y_test_encoded, y_pred)  # Accuracy hesapla
    print(f"{model_name} Accuracy: {acc:.4f}")

    target_names = [str(cls) for cls in label_encoder.classes_]
    # Classification report yazdır
    print(classification_report(y_test_encoded, y_pred,
target_names=target_names))

    # Sonuçları sakla
    results.append((model_name, acc))

# En iyi modeli bul
best_model = max(results, key=lambda x: x[1])
print(f"\nEn iyi model: {best_model[0]} (Accuracy:
{best_model[1]:.4f})")
```

```
Logistic Regression eğitiliyor...

D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as
shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(

Logistic Regression kaydedildi:
../trained_models/Logistic_Regression.pkl
Logistic Regression Accuracy: 0.7111
              precision    recall  f1-score   support

           0       0.81      0.75      0.77        83
           1       0.70      0.62      0.66        48
           2       0.60      0.73      0.66        49

    accuracy                           0.71       180
   macro avg       0.70      0.70      0.70       180
weighted avg       0.72      0.71      0.71       180


SVC eğitiliyor...
SVC kaydedildi: ../trained_models/SVC.pkl
SVC Accuracy: 0.5944
              precision    recall  f1-score   support

           0       0.68      0.77      0.72        83
           1       0.57      0.25      0.35        48
           2       0.48      0.63      0.54        49

    accuracy                           0.59       180
   macro avg       0.58      0.55      0.54       180
weighted avg       0.60      0.59      0.57       180


Random Forest eğitiliyor...
Random Forest kaydedildi: ../trained_models/Random_Forest.pkl
Random Forest Accuracy: 0.7556
              precision    recall  f1-score   support

           0       0.86      0.82      0.84        83
           1       0.81      0.62      0.71        48
           2       0.59      0.78      0.67        49

    accuracy                           0.76       180
   macro avg       0.76      0.74      0.74       180
weighted avg       0.77      0.76      0.76       180


Gradient Boosting eğitiliyor...
Gradient Boosting kaydedildi: ../trained_models/Gradient_Boosting.pkl
Gradient Boosting Accuracy: 0.8167
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        83
           1       0.81      0.79      0.80        48
           2       0.71      0.76      0.73        49

    accuracy                           0.82       180
   macro avg       0.80      0.80      0.80       180
weighted avg       0.82      0.82      0.82       180


XGBoost eğitiliyor...

D:\Anaconda\Lib\site-packages\xgboost\core.py:158: UserWarning:
[16:09:26] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-
autoscaling-group-i-08cbc0333d8d4aae1-1\xgboost\xgboost-ci-windows\
src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)

XGBoost kaydedildi: ../trained_models/XGBoost.pkl
XGBoost Accuracy: 0.8056
              precision    recall  f1-score   support

           0       0.89      0.88      0.88        83
           1       0.82      0.75      0.78        48
           2       0.67      0.73      0.70        49

    accuracy                           0.81       180
   macro avg       0.79      0.79      0.79       180
weighted avg       0.81      0.81      0.81       180


MLP Classifier eğitiliyor...
MLP Classifier kaydedildi: ../trained_models/MLP_Classifier.pkl
MLP Classifier Accuracy: 0.5611
              precision    recall  f1-score   support

           0       0.71      0.64      0.67        83
           1       0.45      0.52      0.48        48
           2       0.47      0.47      0.47        49

    accuracy                           0.56       180
   macro avg       0.54      0.54      0.54       180
weighted avg       0.57      0.56      0.57       180


AdaBoost eğitiliyor...
AdaBoost kaydedildi: ../trained_models/AdaBoost.pkl
AdaBoost Accuracy: 0.7278
              precision    recall  f1-score   support

           0       0.88      0.80      0.84        83
           1       0.88      0.46      0.60        48
           2       0.54      0.88      0.67        49

    accuracy                           0.73       180
   macro avg       0.77      0.71      0.70       180
weighted avg       0.79      0.73      0.73       180


Ridge Classifier eğitiliyor...
Ridge Classifier kaydedildi: ../trained_models/Ridge_Classifier.pkl
Ridge Classifier Accuracy: 0.7944
              precision    recall  f1-score   support

           0       0.88      0.88      0.88        83
           1       0.80      0.67      0.73        48
           2       0.67      0.78      0.72        49

    accuracy                           0.79       180
   macro avg       0.78      0.77      0.77       180
weighted avg       0.80      0.79      0.79       180


KNeighbors eğitiliyor...
KNeighbors kaydedildi: ../trained_models/KNeighbors.pkl
KNeighbors Accuracy: 0.4833
              precision    recall  f1-score   support

           0       0.59      0.72      0.65        83
           1       0.38      0.31      0.34        48
           2       0.31      0.24      0.27        49

    accuracy                           0.48       180
   macro avg       0.43      0.43      0.42       180
weighted avg       0.46      0.48      0.47       180


Gaussian NB eğitiliyor...
Gaussian NB kaydedildi: ../trained_models/Gaussian_NB.pkl
Gaussian NB Accuracy: 0.6000
              precision    recall  f1-score   support

           0       0.65      0.77      0.71        83
           1       0.49      0.42      0.45        48
           2       0.59      0.49      0.53        49

    accuracy                           0.60       180
   macro avg       0.58      0.56      0.56       180
weighted avg       0.59      0.60      0.59       180


En iyi model: Gradient Boosting (Accuracy: 0.8167)
```

```
# Sonuçları DataFrame olarak göster
results_df = pd.DataFrame(results, columns=["Model", "Test Accuracy"])
print("\nModel Performances:")
print(results_df.sort_values(by="Test Accuracy", ascending=False))
```

```
Model Performances:
                  Model  Test Accuracy
3     Gradient Boosting       0.816667
4               XGBoost       0.805556
7      Ridge Classifier       0.794444
2         Random Forest       0.755556
6              AdaBoost       0.727778
0   Logistic Regression       0.711111
9           Gaussian NB       0.600000
1                   SVC       0.594444
5        MLP Classifier       0.561111
8            KNeighbors       0.483333
```

- En iyi sonucu Gradient Boosting verdiği için bu model ile devam edeceğim ve bu modeli geliştirmek için Hiperparametre Optimizasyonu ve Cross-Validation işlemleri yapacağım
- Hiperparametre Optimizasyonu: Grid Search veya Bayesian Optimization yöntemleriyle hiperparametre optimizasyonu yapılacak.
- Cross-Validation: Modelin genelleme performansını artırmak için cross-validation yöntemleri kullanılacak.

```
#!pip install scikit-optimize

from skopt import BayesSearchCV
from skopt.space import Real, Integer

# Model
gb_model = GradientBoostingClassifier()

# Hiperparametre aralığı (Bayesian Search)
param_space = {
    'n_estimators': Integer(50, 500),  # Ağaç sayısı
    'learning_rate': Real(0.01, 0.2, prior='log-uniform'),  # Öğrenme
oranı
    'max_depth': Integer(3, 10),  # Ağaç derinliği
    'subsample': Real(0.5, 1.0),  # Rastgele örnekleme oranı
}

# Bayesian Search
bayes_search = BayesSearchCV(
    gb_model,
    param_space,
    n_iter=10,  # Kaç farklı kombinasyon denenecek, çok zaman aldığı
için 10'a indirdim
    cv=StratifiedKFold(n_splits=5, shuffle=True, random_state=42),
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)

# Modeli eğit
bayes_search.fit(X_train_rfe, y_train_encoded)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
BayesSearchCV(cv=StratifiedKFold(n_splits=5, random_state=42,
shuffle=True),
              estimator=GradientBoostingClassifier(), n_iter=10,
n_jobs=-1,
              scoring='accuracy',
              search_spaces={'learning_rate': Real(low=0.01, high=0.2,
prior='log-uniform', transform='normalize'),
                             'max_depth': Integer(low=3, high=10,
prior='uniform', transform='normalize'),
                             'n_estimators': Integer(low=50, high=500,
prior='uniform', transform='normalize'),
                             'subsample': Real(low=0.5, high=1.0,
prior='uniform', transform='normalize')},
              verbose=2)
```

```
# En iyi modeli kaydet
best_model = bayes_search.best_estimator_
```

```
joblib.dump(best_model,
"../trained_models/Optimized_Gradient_Boosting.pkl")

# En iyi hiperparametreleri yazdır
print("En iyi parametreler:", bayes_search.best_params_)
print("En iyi skor:", bayes_search.best_score_)
```

```
En iyi parametreler: OrderedDict([('learning_rate',
0.07339338249347742), ('max_depth', 7), ('n_estimators', 312),
('subsample', 0.5058257082534684)])
En iyi skor: 0.8125
```

```
from sklearn.model_selection import cross_val_score

# En iyi modeli yükle
best_model =
joblib.load("../trained_models/Optimized_Gradient_Boosting.pkl")
# Cross-Validation ile performansı ölç
cv_scores = cross_val_score(best_model, X_train_rfe, y_train_encoded,
cv=5, scoring='accuracy')

print("Cross-validation skorları:", cv_scores)
print("Ortalama doğruluk:", cv_scores.mean())
```

```
KeyboardInterrupt
```

Sektörel Benzerlik Analizi

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer
import tsfresh
from tsfresh.feature_extraction import EfficientFCParameters
import joblib  # Modeli yüklemek için

# --- 1. Veri İşleme Fonksiyonları ---
def preprocess_open_values(df):
    """ Open değerlerini işleyip log dönüşümü uygular ve 2005'ten
itibaren filtreler. """
    df_data = yf.download(df, start='2005-01-01')

    df_open_data = df_data['Open'] # Sadece open verisini al
    df_open_data = df_open_data.ffill().fillna(0) # İlk ffill, sonra
bütün boş değerleri 0 ile doldur
    df_open_data = df_open_data.resample('M').first() # Günlük
verileri aylığa çevir.

    df_open_data_long =
df_open_data.reset_index().melt(id_vars='Date', var_name='Ticker',
value_name='Open') # Long türüne çevir

    df_open_data_long['Open_LOG'] =
np.log1p(df_open_data_long['Open']) # Open değerlerinin, Log1/y
değerini al
    return df_open_data_long[['Date', 'Ticker', 'Open_LOG']]

# --- 2. Özellik Çıkarma Fonksiyonu ---
def extract_features(df):
    """ Open_LOG üzerinden özellik çıkarır ve seçili olanları
döndürür. """
    features = tsfresh.extract_features(
        df,
        column_id='Ticker',
        column_sort='Date',
        column_value='Open_LOG',
        default_fc_parameters=EfficientFCParameters()
    )

    return features[selected_features_rfe]

# --- 3. Pipeline Tanımlama ---
data_pipeline = Pipeline([
    ('preprocessing', FunctionTransformer(preprocess_open_values)),  #
Open işlemleri
    ('feature_extraction', FunctionTransformer(extract_features)),  #
Tsfresh özellik çıkarımı
])

# --- 4. Model Yükleme ---
#mlp_model = joblib.load("../trained_models/Gradient_Boosting.pkl")  #
Eğittiğin MLP modelini yükle
optimized_gradient_boost_model =
joblib.load("../trained_models/Optimized_Gradient_Boosting.pkl")

def predict_sector(df):
    """ İşlenmiş veriyi modele sokar ve sektöre olan olasılıklarını
döndürür. """
    X_processed = data_pipeline.transform(df)
    probabilities =
optimized_gradient_boost_model.predict_proba(X_processed)[0]  #
Olasılıkları al

    sector_mapping = {0: 'Financials', 1: 'Healthcare', 2:
'Technology'}

    # Yuvarlanmış olasılıkları bir sözlük olarak döndür
    sector_probabilities = {sector_mapping[i]: round(prob, 4) for i,
prob in enumerate(probabilities)}
```

```python
    return sector_probabilities

# --- 6. Kullanım ---
# Örnek olarak seçilen endüstri şirketi verisi (Industrials, Materials vs.)
df_real_estate_companies = pd.read_csv("../data/stock_sectors/real-estate.csv")  # Örnek dosya (inşaat şirketleri)
df_materials_companies = pd.read_csv("../data/stock_sectors/materials.csv")
df_industrials_companies = pd.read_csv("../data/stock_sectors/industrials.csv")
df_energy_companies = pd.read_csv("../data/stock_sectors/energy.csv")

# Şirket sembollerini al
real_estate_tickers = df_real_estate_companies['Symbol'].dropna().tolist()
materials_tickers = df_materials_companies['Symbol'].dropna().tolist()
industrials_tickers = df_industrials_companies['Symbol'].dropna().tolist()
energy_tickers = df_energy_companies['Symbol'].dropna().tolist()

# Şirket isimlerini al (varsa)
real_estate_names = df_real_estate_companies['Company Name'].dropna().tolist()
materials_names = df_materials_companies['Company Name'].dropna().tolist()
industrials_names = df_industrials_companies['Company Name'].dropna().tolist()
energy_names = df_energy_companies['Company Name'].dropna().tolist()

# Sektörleri içeren bir sözlük oluştur
# Rastgele şirketler için bu kısmı çalıştırmak yeterlidir.
sectors = {
    "Real Estate": (real_estate_tickers, real_estate_names),
    "Materials": (materials_tickers, materials_names),
    "Industrials": (industrials_tickers, industrials_names),
    "Energy": (energy_tickers, energy_names)
}

# Rastgele bir sektör seç
selected_sector = random.choice(list(sectors.keys()))
selected_tickers, selected_names = sectors[selected_sector]

# Seçilen sektörden rastgele bir şirket seç
random_index = random.randint(0, len(selected_tickers) - 1)
selected_company_ticker = selected_tickers[random_index]
selected_company_name = selected_names[random_index] if selected_names else "Unknown"
```

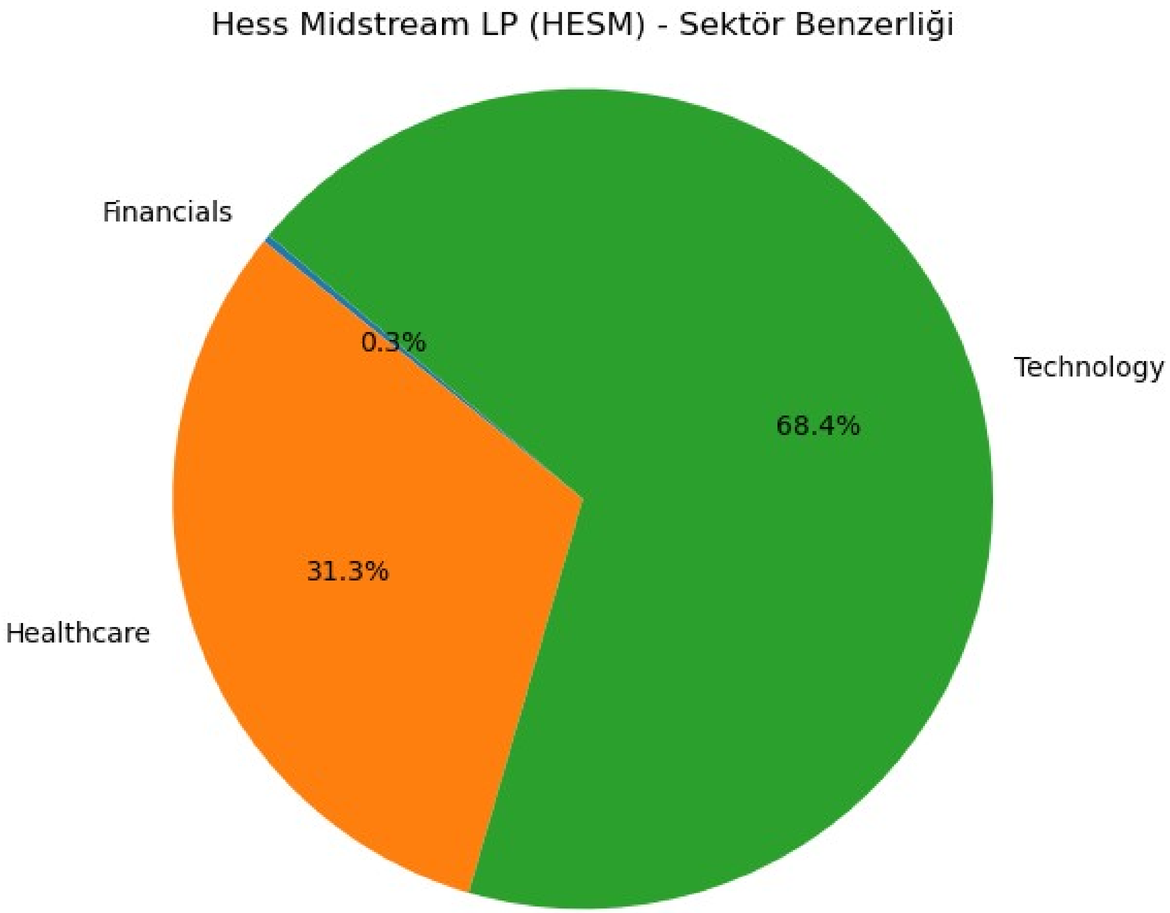
```
# --- Sonuçları Yazdır ---
print(f"Seçilen Şirket: {selected_company_name}
({selected_company_ticker})")
print(f"Şirketin Ait Olduğu Endüstri: {selected_sector}")
```

```
Seçilen Şirket: Hess Midstream LP (HESM)
Şirketin Ait Olduğu Endüstri: Energy
```

```
# Model tahmini yap
predicted_sector = predict_sector(df_selected_company_ticker_random)
```

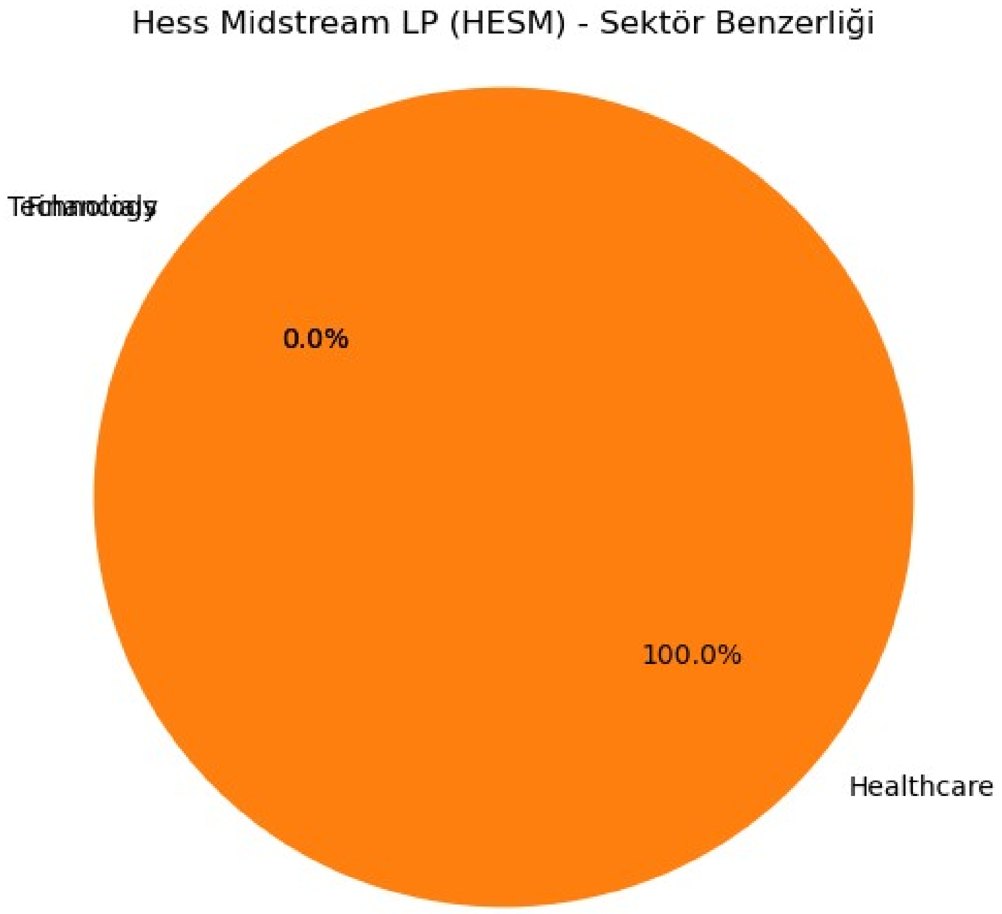
```
[*********************100%***********************]  1 of 1 completed
Feature Extraction: 100%|██████████| 1/1 [00:06<00:00,  6.12s/it]
D:\Anaconda\Lib\site-packages\sklearn\base.py:457: UserWarning: X has
feature names, but GradientBoostingClassifier was fitted without
feature names
  warnings.warn(
```

```
# --- Sonucu Görselleştir ---
# Sözlükten etiketler ve değerler ayrılıyor
labels = list(predicted_sector.keys())
sizes = list(predicted_sector.values())

# Pasta grafiğini oluştur
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title(f"{selected_company_name} ({selected_company_ticker}) -
Sektör Benzerliği")
plt.axis('equal')  # Pasta grafiğinin dairesel görünmesini sağlar
plt.show()
```



Finans Sağlık ya da Teknoloji şirketleri sonuçları:

```
df_healthcare_companies =
pd.read_csv("../data/stock_sectors/healthcare.csv")
healtcare_tickers =
df_healthcare_companies['Symbol'].dropna().tolist()
healtcare_name = df_healthcare_companies['Company
Name'].dropna().tolist()

random_healtcare_company = healtcare_tickers[0]

random_healtcare_company

'LLY'

# Model tahmini yap
predicted_sector = predict_sector(random_healtcare_company)

[*********************100%***********************]  1 of 1 completed
Feature Extraction: 100%|██████████| 1/1 [00:06<00:00,  6.37s/it]
D:\Anaconda\Lib\site-packages\sklearn\base.py:457: UserWarning: X has
feature names, but GradientBoostingClassifier was fitted without
feature names
  warnings.warn(

# --- Sonucu Görselleştir ---
# Sözlükten etiketler ve değerler ayrılıyor
labels = list(predicted_sector.keys())
sizes = list(predicted_sector.values())

# Pasta grafiğini oluştur
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title(f"{selected_company_name} ({selected_company_ticker}) -
Sektör Benzerliği")
plt.axis('equal')  # Pasta grafiğinin dairesel görünmesini sağlar
plt.show()
```



ÖNEMLİ NOT

- Hocam bazen rastgele seçilen şirket verilerinde yetersiz data bulunduğu için tsfresh kütüphanesi öznitelik çıkarırken boş sütunlar çıkarabiliyor
- Bu nedenle model boş sütun değerleriyle çalışamadığı için hata veriyor
- Eğer **model NaN değerleriyle çalışamaz** gibi bir hata alırsanız başka bir şirket seçmeniz gerekiyor