

Veri Bilimi

Otomatik Veri Çekme (Python)

M. Fuat Kına

Veri Bilimi



Veri oluşturma: çekme, toplama, sıfırdan yaratma



Temizleme ve ön işleme



Veri analizi: model kurma, regresyon, sınıflandırma

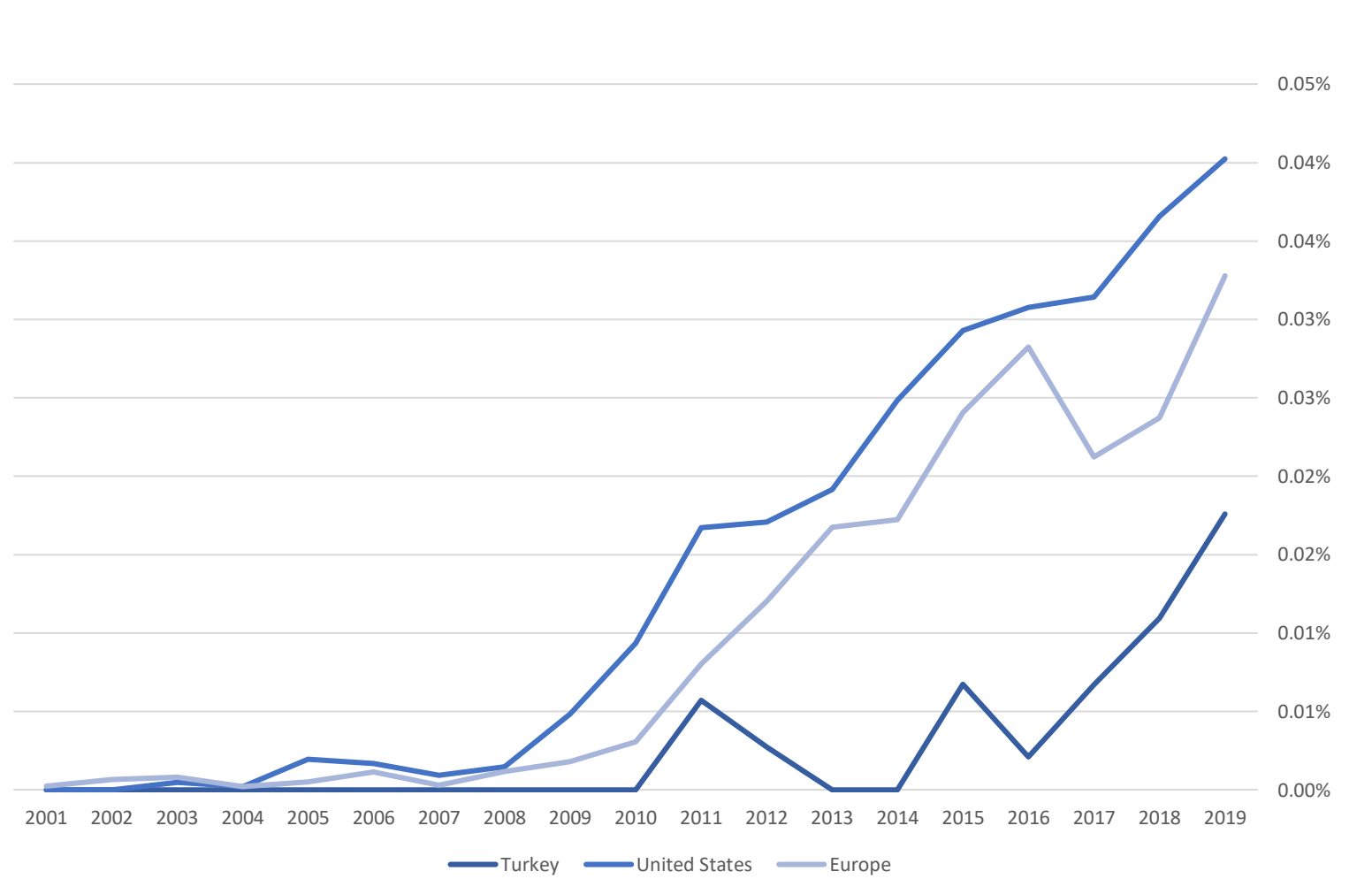


Modelin test edilmesi ve kullanılması: tahmin, optimizasyon, nedensellik, vb.

Veri Bilimi Uygulama Alanları

- Sınıflandırma ve kategorizasyon
- Tahmine dayalı modelleme
- Duygu ve davranış analizi
- Öneri motorları ve kişiselleştirme sistemleri
- Konuşma sistemleri
- Anomali tespiti
- Kalıp (örüntü) tanımlama
- Otonom sistemler

Veri Bilimi ve Sosyal Bilimler





Hesaplamalı Sosyal Bilimler

- Metin Analizi (NLP)
- Coğrafi Bilgi Sistemleri
- Network Analizi
- Veri oluşturma (e.g. toplumsal hareketler, yoksulluk, göçmen karşıtlığı)

Veri oluşturma



Hazır veri setleriyle çalışmak: World Bank, OECD, GLOCON, TÜİK, etc.



Uygulama programlama arayüzü (application programming interface): Sosyal medya, mesajlaşma uygulamaları



Veri kazıma



İleri veri oluşturma teknikleri: Yapay zeka, makine öğrenmesi, vb.

Uygulama programlama arayüzü (API)

- API'lar Web sitesi operatörleri tarafından sağlanır.
- Erişim limitleri (kullanıcı onaylama, kullanım sınırları)

Sosyal medya

- Facebook (<https://developers.facebook.com/>)
- Twitter (<https://developer.twitter.com/en/docs>)
- YouTube (<https://developers.google.com/youtube/v3>)
- Flickr (<https://www.flickr.com/services/api/>)
- Reddit (<https://www.reddit.com/dev/api/>)
- LinkedIn (<https://www.linkedin.com/developers/>)
- **Mesaj uygulamaları:** Telegram, WhatsApp, Threema, Skype, Discord
- **Akış uygulamaları:** Spotify, Apple Music, Vimeo, Twitch
- **Diğer servisler:** Google Maps, Amazon, Wikipedia

Twitter API

- Kolay erişilebilir
- Zengin veri içeriği
- Retweet, favori, takipçi sayıları, tweet içeriği, biyografi, profil fotoğrafı, vb.
- Academic API, ayda 10 milyon tweet sağlıyor.

Zorluklar:

- Gerçek insanların hesaplarından emin olmak zor
- Bu kişilerin kim olduğu hakkında bilgi eksikliği
- Platforma özgü limitler

Python'da ilgili kütüphane (**tweepy**)

Veri kazıma

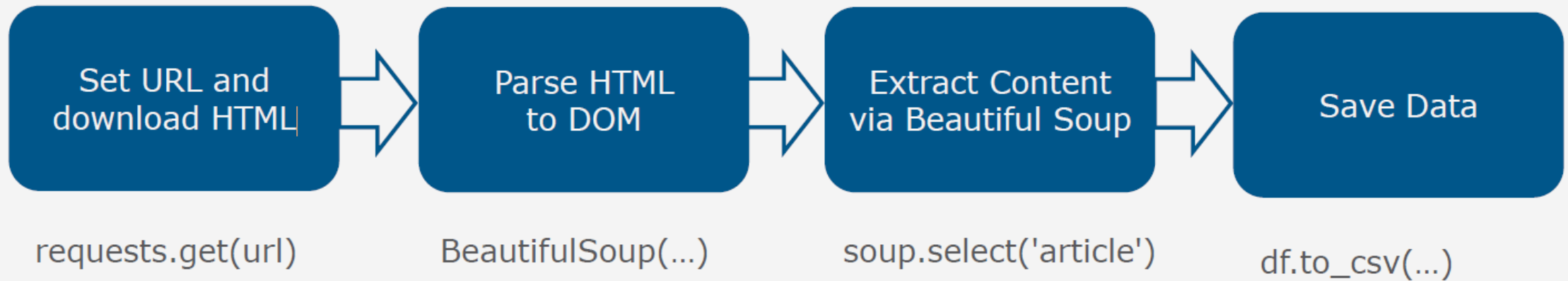
- Sayfayı ve HTML kodunu didikle
- URL'yi (linki) tanımla, ve HTML sayfasını indir
- HTML kodlarını böl, parçala
- İlgili içeriği çıkart
- Veriyi yaz/kaydet

Python'da ilgili kütüphaneler

Scrapy, BeautifulSoup

Webscraping with BeautifulSoup

- Python library for extracting data from HTML and XML files
- Official documentation: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/#>



URL ve HTML yapıları

- url = path + “/” + searchterm + “?” + parameters

<https://www.aljazeera.com/search/Turkey?sort=date>

```
<html>
  <body>
    <div id="div1" class="class-1">
      <p class="class-1 class-2">Hello World!</p>
      <p>Data Science!</p>
    </div>
  </body>
</html>
```

- html = elements + attributes + text
 - Hypertext markup language
 - Çocuk, ebeveyn, kardeş, gelecek kuşaklar...

Some HTML-Elements

Metadata:

`<head>` collection of metadata
`<title>` title of the document

Sections:

`<body>` main contents of the document
`<section>` section of the document
`<h1>`, `<h2>`, ... headlines

Grouped Contents

`<div>` container
`<p>` paragraph

Links

`<a>` Hyperlink, refers via href-attribute to resource

Lists

`` unordered list
`` ordered list
`` entry of a list

Tables

`<table>` table
`<tr>` row of a table
`<td>` cell of a table

XPATH

Html dosyasını Scrapy ile ayrıştırma

Sunumun bu noktadan sonraki kısmında yer alan kodlar için takip eden kaynaktan faydalandım:
<https://app.datacamp.com/learn/courses/web-scraping-with-python>

- / → Bir nesil ileri git
- [] → Aradığımız elemanın sıralmasını belirt
- // → Tüm gelecek kuşaklar içinde ilerle
- *Xpath = '/html/body/div[2]//table'*

Head elemanını altındaki body elemanını altındaki ikinci div elemanının altındaki tüm tabloları seçer.

Örnekler:

'//p'

'/head/body//div'

'div[3]//p'

- Özellik seçimi:
- *Xpath = '//span[@class="span-class"]'*

Class özelliği 'span_class' olan tüm span elemanlarını seçer.

XPATH

Örnek! 'Data Science' kelime grubunu seç.

```
html =  
"""<html>  
<body>  
  <div>  
    <p>Hello World!</p>  
    <div>  
      <p>Data Science!</p>  
    </div>  
  </div>  
  <div>  
    <p>Thanks for Watching!</p>  
  </div>  
</body>  
</html>"""
```

- Herhangi bir elemanın tüm çocuklarını seç:

Xpath = `'/html/body/*'`

- Body elemanını altındaki tüm gelecek kuşakları seç:

Xpath = `"/html/body//*"`

- HTML dökümanındaki tüm elemanları seç:

Xpath = `"//*"`

- Artık *, /, //, [], @ kullanarak, eleman veya nitelik arayarak ilgili yolu belirtebiliriz.
- Doğrudan eşleştirme yerine birden çok nesne içeren özellikler için (örneğin `<p class = 'class1 class2'>`) içerme fonksiyonunu ("contain") kullanabiliriz.

XPATH

Örnek:

```
html = '''
<html>
<body>
  <div id="div1" class="class-1">
    <p class="class-1 class-2">Hello World!</p>
    <div id="div2">
      <p id="p2" class="class-2">Choose
        <a href="http://datascience.com">Data Science!</a>
      </p>
    </div>
  </div>
  <div id="div3" class="class-2">
    <p class="class-2">Thanks for Watching!</p>
  </div>
</body>
</html>'''
```

- Id özelliğini kullanarak "Thanks for Watching!" ifadesini içeren paragraf ögesini seçelim
- Class özelliğini kullanarak "Hello World!" ifadesini içeren paragraf ögesini seçelim
- "Data Science" hiper linki için ilgili bölümün href özelliğini seçelim (elemanın değil özelliğin içeriğini almak istediğimizi unutmayın)

Xpath ile metin çekme

```
<p id="p-example">
  Hello world!
  Try <a href="http://www.datacamp.com">DataCamp</a> today!
</p>
```

- In XPath use `text()`

```
sel.xpath('//p[@id="p-example"]/text()').extract()
# result: ['\n Hello world!\n Try ', ' today!\n']
```

```
sel.xpath('//p[@id="p-example"]//text()').extract()
# result: ['\n Hello world!\n Try ', 'DataCamp', ' today!\n']
```


Spider

```
class DCspider( scrapy.Spider ):
    name = "dcspider"

    def start_requests( self ):
        urls = [ 'https://www.datacamp.com/courses/all' ]
        for url in urls:
            yield scrapy.Request( url = url, callback = self.parse )
    def parse( self, response ):
        links = response.css('div.course-block > a::attr(href)').extract()
        filepath = 'DC_links.csv'
        with open( filepath, 'w' ) as f:
            f.writelines( [link + '/n' for link in links] )
```

Spider

```
class DCspider( scrapy.Spider ):
    name = "dcspider"

    def start_requests( self ):
        urls = [ 'https://www.datacamp.com/courses/all' ]
        for url in urls:
            yield scrapy.Request( url = url, callback = self.parse )
    def parse( self, response ):
        links = response.css('div.course-block > a::attr(href)').extract()
        for link in links:
            yield response.follow( url = link, callback = self.parse2 )
    def parse2( self, response ):
        # parse the course sites here!
```

Spider

```
import scrapy
from scrapy.crawler import CrawlerProcess

class DC_Chapter_Spider(scrapy.Spider):

    name = "dc_chapter_spider"

    def start_requests( self ):
        url = 'https://www.datacamp.com/courses/all'
        yield scrapy.Request( url = url,
                               callback = self.parse_front )

    def parse_front( self, response ):
        ## Code to parse the front courses page

    def parse_pages( self, response ):
        ## Code to parse course pages
        ## Fill in dc_dict here

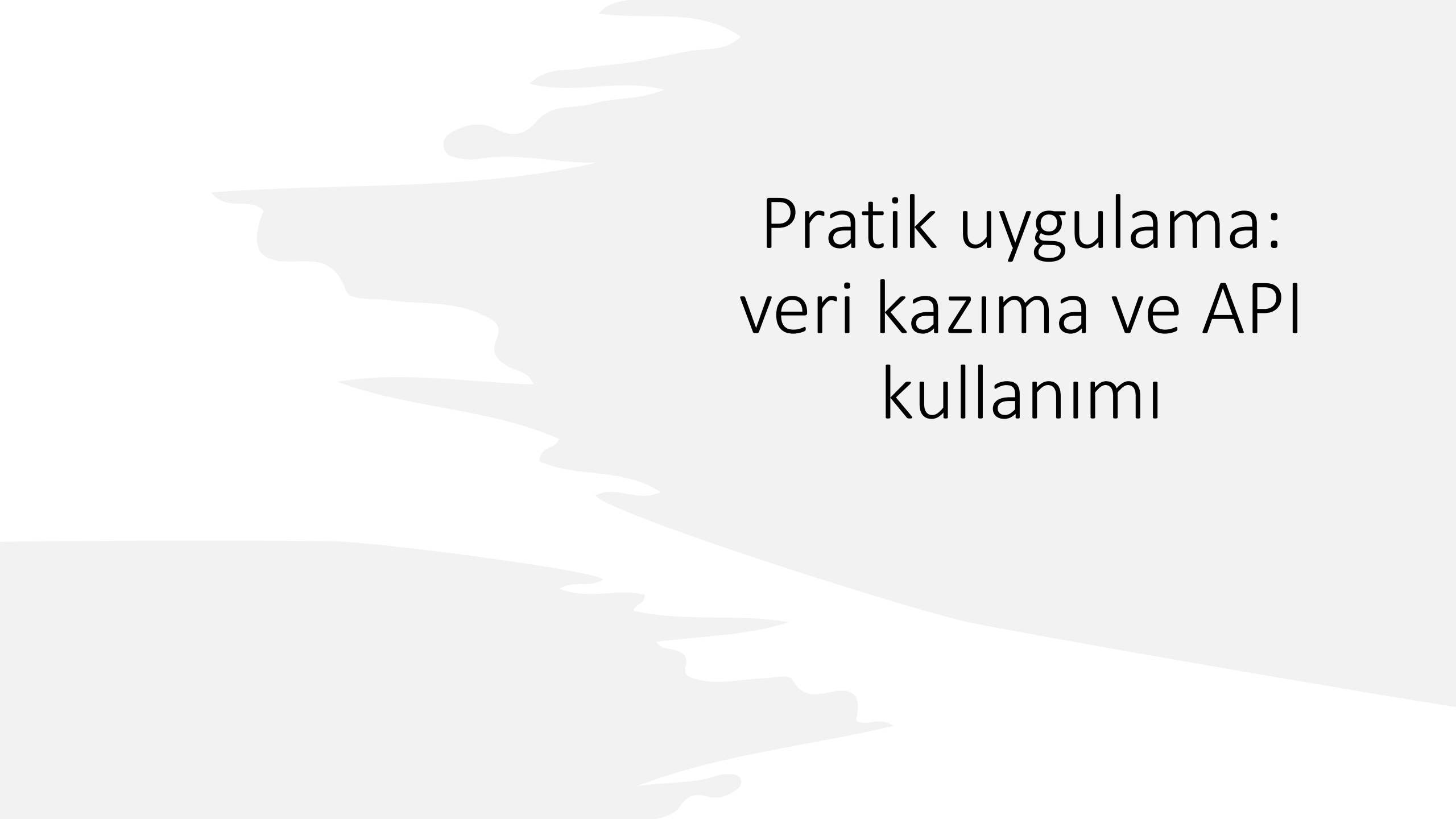
dc_dict = dict()

process = CrawlerProcess()
process.crawl(DC_Chapter_Spider)
process.start()
```

Spider

```
def parse_front( self, response ):  
    # Narrow in on the course blocks  
    course_blocks = response.css( 'div.course-block' )  
    # Direct to the course links  
    course_links = course_blocks.xpath( './a/@href' )  
    # Extract the links (as a list of strings)  
    links_to_follow = course_links.extract()  
    # Follow the links to the next parser  
    for url in links_to_follow:  
        yield response.follow( url = url,  
                                callback = self.parse_pages )
```

```
def parse_pages( self, response ):  
    # Direct to the course title text  
    crs_title = response.xpath('//h1[contains(@class,"title")]/text()')  
    # Extract and clean the course title text  
    crs_title_ext = crs_title.extract_first().strip()  
    # Direct to the chapter titles text  
    ch_titles = response.css( 'h4.chapter__title::text' )  
    # Extract and clean the chapter titles text  
    ch_titles_ext = [t.strip() for t in ch_titles.extract()]  
    # Store this in our dictionary  
    dc_dict[ crs_title_ext ] = ch_titles_ext
```



Pratik uygulama:
veri kazıma ve API
kullanımı