**Extract Fonksiyonu - Detaylı Açıklama**

**Tarih**: 20 Haziran 2025, Saat: 20:47 +03  
**Konu**: extract Fonksiyonu - Ülke GDP Verisi Çekme  
**Kaynak**: IBM Data Engineering Kursu, ETL ve Web Scraping

**1. Fonksiyonun Genel Bakış**

* **Amaç**: Belirtilen URL’den (Wikipedia’dan ülke GDP verileri) tablo verilerini çeker ve bir Pandas DataFrame’ine dönüştürür.
* **Giriş Parametreleri**:
  + url: Veri çekilecek web sayfası adresi.
  + table\_attribs: DataFrame’in sütun isimlerini içeren liste (['Country', 'GDP\_USD\_millions']).
* **Çıktı**: Ülke isimleri ve GDP (milyon USD) değerlerini içeren bir DataFrame.

**2. Kod ve Adım Adım Açıklama**

**Kod**

def extract(url, table\_attribs):

page = requests.get(url).text

data = BeatifulSoup(page, 'html.parser')

df = pd.DataFrame(columns=table\_attribs)

tables = data.find\_all('tbody')

rows = tables[2].find\_all('tr')

for row in rows:

col = row.find\_all('td')

if len(col) != 0:

if col[0].find('a') is not None and '-' not in col[2]:

data\_dict = {"Country": col[0].a.contents[0],

"GDP\_USD\_millions": col[2].contents[0]}

df1 = pd.DataFrame(data\_dict, index=[0])

df = pd.concat([df, df1], ignore\_index=True)

return df

**Adım Adım Açıklama**

1. **Web Sayfasını İndirme**
   * page = requests.get(url).text
     + requests.get(url): Belirtilen URL’den (Wikipedia arşiv sayfası) HTML içeriğini indirir.
     + .text: İndirilen içeriği metin formatında alır.
     + **Not**: İnternet bağlantısı gerektirir, site erişimi engellenirse hata verebilir.
2. **HTML’yi Parse Etme**
   * data = BeatifulSoup(page, 'html.parser')
     + BeautifulSoup: HTML’yi parse eder ve ağaç yapısı (tree) haline getirir.
     + **Düzeltme**: Kodda BeatifulSoup yazılmış, doğru olan BeautifulSoup. Bu hata düzeltilmeli.
     + 'html.parser': HTML’yi işlemek için kullanılan parser.
3. **Boş DataFrame Oluşturma**
   * df = pd.DataFrame(columns=table\_attribs)
     + Boş bir DataFrame oluşturur, sütun isimleri ['Country', 'GDP\_USD\_millions'] olarak tanımlanır.
     + **Amaç**: Çekilen veriler bu yapıda saklanacak.
4. **Tabloyu Bulma**
   * tables = data.find\_all('tbody')
     + find\_all('tbody'): HTML’de <tbody> etiketlerini bulur (tablo gövdeleri).
     + rows = tables[2].find\_all('tr')
       - tables[2]: Üçüncü <tbody> etiketini seçer (verinin bulunduğu tablo).
       - find\_all('tr'): Bu tablodaki tüm satırları (<tr> etiketleri) alır.
       - **Not**: 2 indeksi spesifik bir tabloyu hedefler, site yapısına bağlıdır.
5. **Satırları İşleme**
   * for row in rows:
     + Her satır (<tr>) için döngü başlatır.
     + col = row.find\_all('td')
       - Satırdaki tüm hücreleri (<td>) bulur.
6. **Veri Filtresi ve Çekme**
   * if len(col) != 0:
     + Hücrelerin boş olmadığını kontrol eder (geçersiz satırları atlar).
   * if col[0].find('a') is not None and '-' not in col[2]:
     + col[0].find('a') is not None: İlk hücrede bir <a> (link) etiketi varsa (ülke isimleri linkli).
     + '-' not in col[2]: GDP sütununda eksik veri (-) yoksa.
     + **Amaç**: Yalnızca geçerli ülke ve GDP verilerini seçer.
   * data\_dict = {"Country": col[0].a.contents[0], "GDP\_USD\_millions": col[2].contents[0]}
     + col[0].a.contents[0]: Ülke ismini <a> etiketinden çeker.
     + col[2].contents[0]: GDP değerini (milyon USD) çeker.
     + **Not**: .contents[0] ilk içeriği alır, birden fazla içerik varsa hata verebilir.
7. **DataFrame’e Ekleme**
   * df1 = pd.DataFrame(data\_dict, index=[0])
     + Çekilen veriyi tek satırlık bir DataFrame’e dönüştürür.
   * df = pd.concat([df, df1], ignore\_index=True)
     + Ana DataFrame’e yeni satırı ekler, indeksleri sıfırlayarak tutarlılık sağlar.
   * **Amaç**: Veriyi biriktirir.
8. **Döngü Kontrolü ve Dönüş**
   * count kullanılmamış, ancak for döngüsü tüm satırları işler.
   * return df: İşlenen verilerle dolu DataFrame’i döndürür.

**3. Bağlantılar**

* **Banka Lab’ı (Task 2)**: extract() fonksiyonu banka verisi için benzer şekilde <tr> ve <td> etiketlerini işledi. Bu fonksiyon, Wikipedia’dan GDP verisi çekerken aynı mantığı kullanıyor.
* **Fark**: Banka lab’ında tables[1] kullanılmıştı, burada tables[2] hedefleniyor (site yapısına bağlı).

**4. İpuçları**

* **Test**: python3.11 -m pip install requests beautifulsoup4 pandas sqlite3 ile modülleri kontrol et.
* **Hata**: BeatifulSoup yazımını BeautifulSoup olarak düzelt, aksi halde ModuleNotFoundError alır.
* **Not Tutma**: Notion’a kod ve açıklamaları ekle, çıktıyı (Task\_2d\_extract.png) sakla.

**5. Kaynaklar**

* **Ders**: IBM Data Engineering, Web Scraping ve ETL.
* **Dokümantasyon**: [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/), [Pandas](https://pandas.pydata.org/)

**tables[2] Seçimi - Detaylı Açıklama**

**Tarih**: 20 Haziran 2025, Saat: 20:46 +03  
**Konu**: tables[2] Neden Kullanılıyor?  
**Kaynak**: IBM Data Engineering Kursu, Web Scraping

**1. Sorunun Bağlamı**

* **Kod Parçası**:
* tables = data.find\_all('tbody')
* rows = tables[2].find\_all('tr')
* **Sorun**: Neden tables[2] ile üçüncü <tbody> etiketi seçiliyor?
* **Amaç**: Wikipedia’daki GDP verilerinin bulunduğu tablodaki satırları (<tr>) çekmek.

**2. Neden tables[2]?**

* **HTML Yapısı**: Bir web sayfasında birden fazla <tbody> etiketi olabilir. find\_all('tbody'), sayfanın tüm <tbody> etiketlerini bir liste olarak döndürür.
* **İndeksleme**: Python’da liste indeksleri 0’dan başlar:
  + tables[0]: İlk <tbody>.
  + tables[1]: İkinci <tbody>.
  + tables[2]: Üçüncü <tbody>.
* **Sebep**: Hedef verinin (ülke GDP tablosu) üçüncü <tbody> etiketinde olduğu varsayılıyor. Bu, sayfanın HTML yapısına bağlıdır.
  + Wikipedia sayfaları genellikle birden fazla tablo içerir (örneğin, kaynaklar, notlar, ana veri tablosu).
  + tables[2]’nin seçilmesi, veri tablosunun üçüncü <tbody> içinde yer aldığını gösterir.

**3. Neden 2?**

* **Siteye Özel**: https://web.archive.org/web/20230902185326/https://en.wikipedia.org/wiki/List\_of\_countries\_by\_GDP\_%28nominal%29 adresinde HTML yapısı incelendiğinde:
  + İlk <tbody>: Başlık veya başka bir tablo.
  + İkinci <tbody>: Başka bir veri grubu (örneğin, notlar).
  + Üçüncü <tbody>: Ana GDP verilerinin bulunduğu tablo.
* **Doğrulama**: Sayfanın kaynak kodunda <tbody> etiketlerinin sırasına bakılarak bu indeks belirlenmiş. Geliştirici, verinin bu konumda olduğunu tespit etmiş.
* **Banka Lab’ı Bağlantısı (Task 2)**: Banka verisi için tables[1] kullanılmıştı, çünkü o sayfada veri ikinci <tbody>’deydi. Her sayfa yapısı farklıdır.

**4. Potansiyel Riskler**

* **Değişim**: Eğer Wikipedia sayfası güncellenirse veya arşiv sürümü değişirse, <tbody> sıraları kayabilir. Bu durumda hata verebilir.
* **Çözüm**: Daha spesifik bir seçici (örneğin, find('table', {'class': 'wikitable'})) kullanılabilir.

**5. İpuçları**

* **Test**: Kodu çalıştırıp print(len(tables)) ile toplam <tbody> sayısını kontrol et. print(tables[2].prettify()) ile üçüncü tabloyu incele.
* **Not Tutma**: Notion’a bu açıklamayı ekle, ekran görüntüsüyle (örneğin, Task\_2e\_tables.png) destekle.
* **Düzeltme**: Eğer veri yanlış yerdeyse, tables indeksini (0, 1, 2) değiştirerek test et.

**6. Kaynaklar**

* **Ders**: IBM Data Engineering, Web Scraping.
* **Dokümantasyon**: [BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

**Transform Fonksiyonu - Detaylı Açıklama**

**Tarih**: 20 Haziran 2025, Saat: 20:58 +03  
**Konu**: transform Fonksiyonu - GDP Verisi Dönüşümü  
**Kaynak**: IBM Data Engineering Kursu, ETL

**1. Fonksiyonun Genel Bakış**

* **Amaç**: Ülke GDP verilerini milyon USD’den milyar USD’ye dönüştürür, sayısal formata çevirir ve DataFrame’i günceller.
* **Giriş Parametresi**:
  + df: Ülke isimleri ve GDP (milyon USD) değerlerini içeren bir Pandas DataFrame’i.
* **Çıktı**: Dönüştürülmüş ve sütun adı güncellenmiş bir DataFrame.

**2. Kod ve Adım Adım Açıklama**

**Kod**

def transform(df):

GDP\_list = df['GDP\_USD\_millions'].tolist()

GDP\_list = [float("".join(x.split(','))) for x in GDP\_list]

GDP\_list = [np.round(x/1000, 2) for x in GDP\_list]

df['GDP\_USD\_millions'] = GDP\_list

df = df.rename(columns={"GDP\_USD\_millions": "GDP\_USD\_billions"})

return df

**Adım Adım Açıklama**

1. **GDP Verisini Listeye Çevirme**
   * GDP\_list = df['GDP\_USD\_millions'].tolist()
     + df['GDP\_USD\_millions']: DataFrame’deki GDP sütununu seçer.
     + .tolist(): Bu sütunun verilerini bir Python listesine dönüştürür.
     + **Amaç**: Sayısal işlemler için veriyi manipüle edilebilir hale getirmek.
     + **Not**: Veriler başlangıçta string formatında (örneğin, "1,234,567") olabilir.
2. **Virgülleri Kaldırıp Sayıya Çevirme**
   * GDP\_list = [float("".join(x.split(','))) for x in GDP\_list]
     + x.split(','): Her GDP değerindeki virgülleri ayırır (örneğin, "1,234,567" → ["1", "234", "567"]).
     + ""join(...): Virgülsüz bir string oluşturur (örneğin, "1234567").
     + float(...): String’i ondalık sayıya çevirir.
     + **List Comprehension**: Tüm liste elemanlarına bu dönüşümü uygular.
     + **Amaç**: Virgüllü string formatını (örneğin, "1,234,567") sayısal forma (1234567.0) çevirmek.
     + **Not**: Eğer veri boşsa veya geçersizse hata verebilir (try-except ile kontrol edilebilir).
3. **Milyondan Milyara Dönüşüm ve Yuvarlama**
   * GDP\_list = [np.round(x/1000, 2) for x in GDP\_list]
     + x/1000: Her değeri 1000’e böler (milyon USD’den milyar USD’ye dönüşüm).
     + np.round(..., 2): Sonucu 2 ondalık basamağa yuvarlar.
     + **List Comprehension**: Tüm liste elemanlarına bu işlemi uygular.
     + **Amaç**: GDP’yi milyar cinsine çevirip temiz bir format sağlamak (örneğin, 1234567.0 → 1234.57).
     + **Not**: numpy (np) gereklidir, aksi halde hata alır.
4. **DataFrame’i Güncelleme**
   * df['GDP\_USD\_millions'] = GDP\_list
     + Dönüştürülen GDP değerlerini DataFrame’in orijinal sütununa atar.
     + **Amaç**: Veriyi DataFrame’e geri yüklemek.
5. **Sütun Adını Değiştirme**
   * df = df.rename(columns={"GDP\_USD\_millions": "GDP\_USD\_billions"})
     + Sütun adını "GDP\_USD\_millions"’dan "GDP\_USD\_billions"’a günceller.
     + **Amaç**: Dönüşümün sonucunu yansıtmak.
     + **Not**: rename() yeni bir DataFrame döndürür, bu yüzden atama (df =) gereklidir.
6. **Dönüş**
   * return df
     + Dönüştürülmüş DataFrame’i fonksiyonun dışına döndürür.

**3. Bağlantılar**

* **Banka Lab’ı (Task 2)**: transform fonksiyonu banka verilerinde (örneğin, market cap dönüşümü) benzer şekilde veri formatını değiştirdi. Bu fonksiyon, GDP’yi milyar cinsine çevirerek aynı mantığı kullanıyor.
* **Fark**: Banka lab’ında birim dönüşümü farklıydı, burada 1000’e bölme ve yuvarlama var.

**4. İpuçları**

* **Test**: Küçük bir DataFrame ile test et (örneğin, df = pd.DataFrame({'GDP\_USD\_millions': ['1,234', '5,678']})).
* **Hata**: Eğer np tanınmazsa import numpy as np’yi kontrol et.
* **Not Tutma**: Notion’a kod ve açıklamaları ekle, çıktıyı (Task\_2f\_transform.png) sakla.

**5. Kaynaklar**

* **Ders**: IBM Data Engineering, ETL.
* **Dokümantasyon**: [Pandas](https://pandas.pydata.org/), [NumPy](https://numpy.org/)

**load\_to\_db Fonksiyonu - Detaylı Açıklama**

**Tarih**: 20 Haziran 2025, Saat: 21:08 +03  
**Konu**: load\_to\_db Fonksiyonu - Veriyi Veritabanına Yükleme  
**Kaynak**: IBM Data Engineering Kursu, ETL

**1. Fonksiyonun Genel Bakış**

* **Amaç**: Bir Pandas DataFrame’ini SQLite veritabanına bir tablo olarak kaydeder.
* **Giriş Parametreleri**:
  + df: Yüklenmek istenen veri içeren DataFrame.
  + sql\_connection: Veritabanına bağlanan SQLite bağlantı nesnesi.
  + table\_name: Verinin kaydedileceği tablo adı.
* **Çıktı**: Hiçbir şey döndürmez (void), sadece veritabanına yükleme yapar.

**2. Kod ve Adım Adım Açıklama**

**Kod**

def load\_to\_db(df, sql\_connection, table\_name):

df.to\_sql(table\_name, sql\_connection, if\_exists='replace', index=False)

**Adım Adım Açıklama**

1. **Veriyi Veritabanına Yükleme**
   * df.to\_sql(table\_name, sql\_connection, if\_exists='replace', index=False)
     + to\_sql(): Pandas DataFrame’ini SQL tablosuna dönüştürür ve kaydeder.
     + table\_name: Verinin kaydedileceği tablo adını belirtir (örneğin, 'Countries\_by\_GDP').
     + sql\_connection: SQLite bağlantı nesnesini (örneğin, sqlite3.connect('World\_Economies.db') ile oluşturulmuş) tanımlar.
     + if\_exists='replace': Eğer tablo zaten varsa, üzerine yazar (var olan verileri siler).
     + index=False: DataFrame’in indeksini tabloya eklemez (indeks bilgisi veri değilse gereksizdir).
     + **Amaç**: Veriyi SQLite veritabanına kalıcı olarak kaydeder.
2. **Fonksiyonun Sonu**
   * Fonksiyon açıkça bir dönüş değeri (return) içermez, yükleme işlemini tamamlar ve biter.
   * **Not**: Bağlantı nesnesi açık kalmalı, bu yüzden conn.close() ana kodda çağrılmalı.

**3. Bağlantılar**

* **Banka Lab’ı (Task 2)**: df.to\_sql() ile banka verileri SQLite’a yüklendi, bu fonksiyon aynı mantığı kullanıyor.
* **Fark**: Banka lab’ında if\_exists parametresi farklı olabilir (append gibi), burada replace ile yeniden yazma tercih edilmiş.

**4. İpuçları**

* **Test**: Küçük bir DataFrame ile test et (örneğin, df = pd.DataFrame({'Country': ['USA'], 'GDP\_USD\_billions': [20.5]})).
* **Hata**: Eğer sql\_connection kapalıysa hata alır, bağlantıyı kontrol et.
* **Not Tutma**: Notion’a kod ve açıklamaları ekle, çıktıyı (Task\_2g\_load\_db.png) sakla.

**5. Kaynaklar**

* **Ders**: IBM Data Engineering, ETL.
* **Dokümantasyon**: [Pandas to\_sql](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_sql.html), [SQLite3](https://docs.python.org/3/library/sqlite3.html)

**run\_query Fonksiyonu - Detaylı Açıklama**

**Tarih**: 20 Haziran 2025, Saat: 21:18 +03  
**Konu**: run\_query Fonksiyonu - SQL Sorgu Çalıştırma  
**Kaynak**: IBM Data Engineering Kursu, ETL ve SQLite3

**1. Fonksiyonun Genel Bakış**

* **Amaç**: Verilen SQL sorgusunu bir SQLite veritabanında çalıştırır ve sonuçları terminale yazdırır.
* **Giriş Parametreleri**:
  + query\_statement: Çalıştırılacak SQL sorgusu (string formatında).
  + sql\_connection: Veritabanına bağlanan SQLite bağlantı nesnesi.
* **Çıktı**: Hiçbir şey döndürmez (void), sadece sorgu ve sonucunu yazdırır.

**2. Kod ve Adım Adım Açıklama**

**Kod**

def run\_query(query\_statement, sql\_connection):

print(query\_statement)

query\_output = pd.read\_sql(query\_statement, sql\_connection)

print(query\_output)

**Adım Adım Açıklama**

1. **Sorguyu Yazdır**
   * print(query\_statement)
     + Girilen SQL sorgusunu terminale yazdırır.
     + **Amaç**: Kullanıcının hangi sorgunun çalıştığını görmesini sağlar (hata ayıklama için faydalı).
     + **Örnek**: SELECT \* FROM Countries\_by\_GDP yazdırılır.
2. **Sorguyu Çalıştırma ve Veri Çekme**
   * query\_output = pd.read\_sql(query\_statement, sql\_connection)
     + pd.read\_sql(): SQL sorgusunu veritabanında çalıştırır ve sonucu bir Pandas DataFrame’ine dönüştürür.
     + query\_statement: Çalıştırılacak sorgu (örneğin, SELECT \* FROM Countries\_by\_GDP).
     + sql\_connection: SQLite bağlantı nesnesi (örneğin, sqlite3.connect('World\_Economies.db')).
     + **Amaç**: Veritabanından çekilen veriyi işlenebilir bir formata (DataFrame) alır.
     + **Not**: Sorgu geçersizse hata verebilir (örneğin, tablo adı yanlışsa).
3. **Sonucu Yazdır**
   * print(query\_output)
     + Çekilen verilerin bulunduğu DataFrame’i terminale yazdırır.
     + **Amaç**: Kullanıcının sorgu sonucunu görsel olarak incelemesini sağlar.
     + **Örnek Çıktı**:
     + Country GDP\_USD\_billions
     + 0 USA 20.5
     + 1 China 14.7
4. **Fonksiyonun Sonu**
   * Fonksiyon açıkça bir dönüş değeri (return) içermez, sadece yazdırma yapar.
   * **Not**: Bağlantı nesnesi açık kalmalı, bu yüzden conn.close() ana kodda çağrılmalı.

**3. Bağlantılar**

* **Banka Lab’ı (Task 2)**: Veritabanından veri çekmek için benzer sorgular kullanıldı (örneğin, SELECT ile banka verileri). Bu fonksiyon, sorgu çalıştırmayı otomatikleştiriyor.
* **Lab**: World\_Economies.db ile Countries\_by\_GDP tablosunu sorgula (örneğin, SELECT \* FROM Countries\_by\_GDP).

**4. İpuçları**

* **Test**: Küçük bir sorgu ile test et (örneğin, run\_query("SELECT \* FROM Countries\_by\_GDP LIMIT 5", conn)).
* **Hata**: Eğer pd tanınmazsa import pandas as pd’yi kontrol et.
* **Not Tutma**: Notion’a kod ve açıklamaları ekle, çıktıyı (Task\_2h\_query.png) sakla.

**5. Kaynaklar**

* **Ders**: IBM Data Engineering, ETL ve SQLite3.
* **Dokümantasyon**: Pandas read\_sql, SQLite3