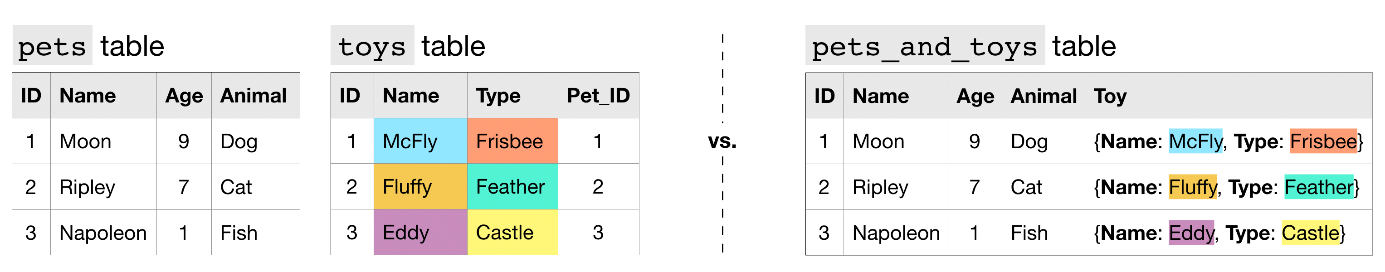
Introduction[¶](https://www.kaggle.com/code/alexisbcook/nested-and-repeated-data" \l "Introduction" \t "_self)

Şimdiye kadar, sayısal türler (tam sayılar, kayan nokta değerleri), dizeler ve DATETIME türü de dahil olmak üzere birçok veri türüyle çalıştınız. Bu eğitimde, iç içe geçmiş ve tekrarlanan verileri nasıl sorgulayacağınızı öğreneceksiniz. Bunlar, BigQuery veri kümelerinde bulabileceğiniz en karmaşık veri türleridir!

# Nested data[¶](https://www.kaggle.com/code/alexisbcook/nested-and-repeated-data" \l "Nested-data" \t "_self)

Evcil hayvanlar ve oyuncakları hakkında bilgi içeren varsayımsal bir veri setini ele alalım. Bu bilgileri iki farklı tabloda düzenleyebiliriz (**pets** tablosu ve **toys** tablosu). **toys** tablosu, her bir oyuncağı ait olduğu evcil hayvanla eşleştirmek için kullanılabilecek bir "**Pet\_ID**" sütunu içerebilir.

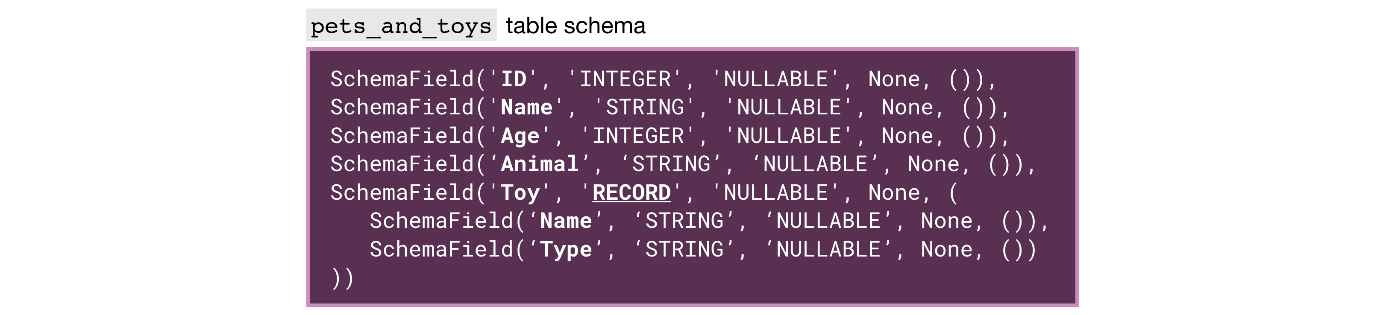
BigQuery'de diğer bir seçenek ise, aşağıdaki **pets\_and\_toys** tablosuna benzer şekilde tüm bilgileri tek bir tabloda düzenlemektir.



Bu durumda, **toys** tablosundaki tüm bilgiler tek bir sütunda (**pets\_and\_toys** tablosundaki 'Toy' sütunu) toplanmıştır. **pets\_and\_toys** tablosundaki 'Toy' sütununu **iç içe (nested)** bir sütun olarak adlandırıyoruz ve 'Name' ve 'Type' alanlarının onun içine yerleştirildiğini söylüyoruz.

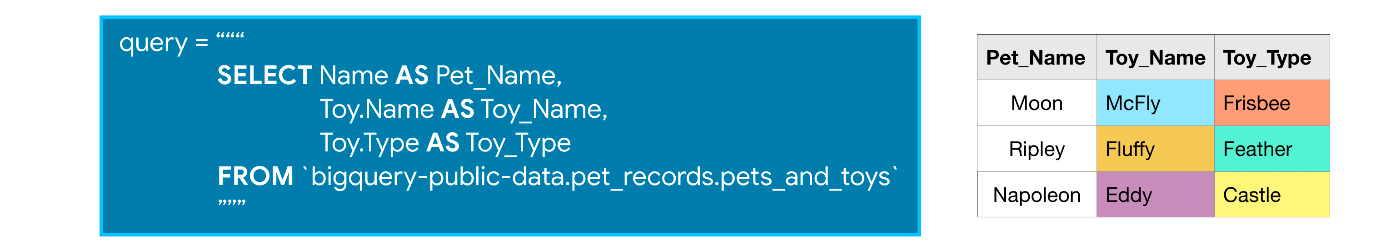
İç içe sütunlar, **STRUCT** (veya **RECORD**) tipindedir. Bu durum, aşağıdaki tablo şemasında da yansıtılmıştır.

Bir tablonun yapısını **şema (schema)** olarak adlandırdığımızı hatırlayın. Tablo şemasını nasıl yorumlayacağınızı gözden geçirmeniz gerekirse, SQL'e Giriş mikro-kursundaki **bu dersi** kontrol edebilirsiniz.



İç içe geçmiş verilere sahip bir sütunu sorgulamak için, her bir alanı onu içeren sütun bağlamında tanımlamamız gerekir:

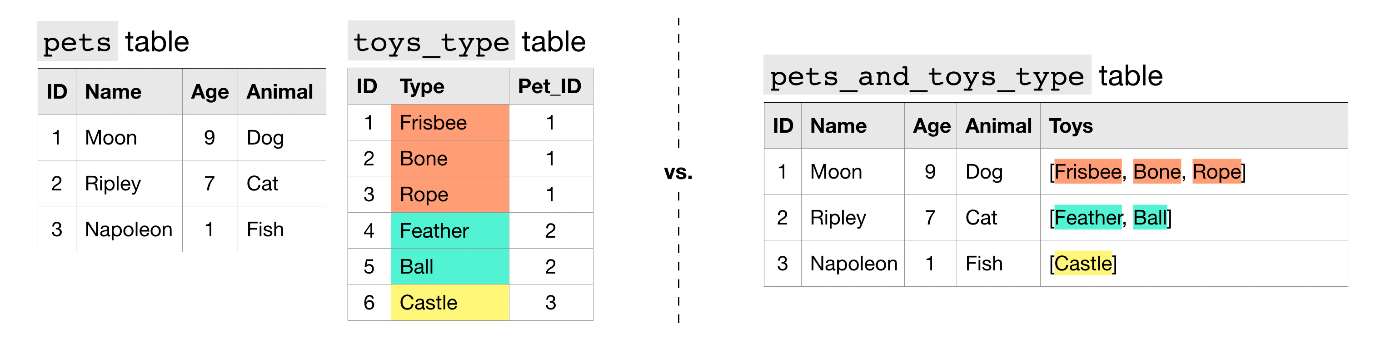
* **Toy.Name**, 'Toy' sütunundaki 'Name' alanını ifade eder.
* **Toy.Type**, 'Toy' sütunundaki 'Type' alanını ifade eder.



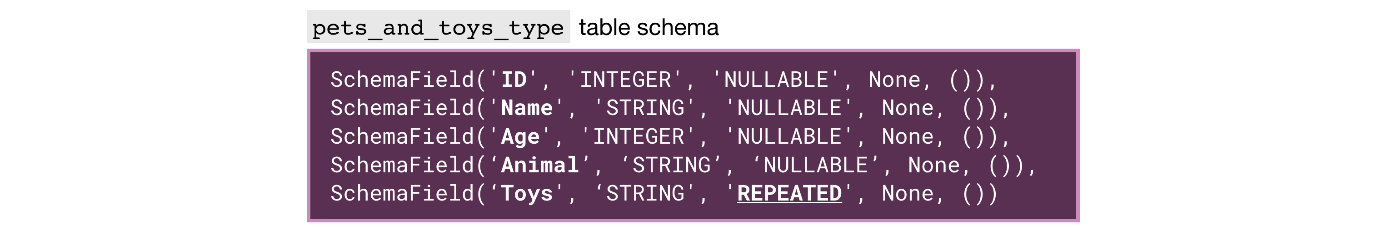
Bunun dışında, her zamanki kurallarımız aynı kalır; sorularımızda başka bir şeyi değiştirmemize gerek yoktur.

# Repeated data[¶](https://www.kaggle.com/code/alexisbcook/nested-and-repeated-data" \l "Repeated-data" \t "_self)

Şimdi, her evcil hayvanın birden fazla oyuncağa sahip olabileceği (daha gerçekçi!) bir durumu ele alalım. Bu durumda, bu bilgileri tek bir tabloya daraltmak için farklı bir veri türünden yararlanmamız gerekiyor.

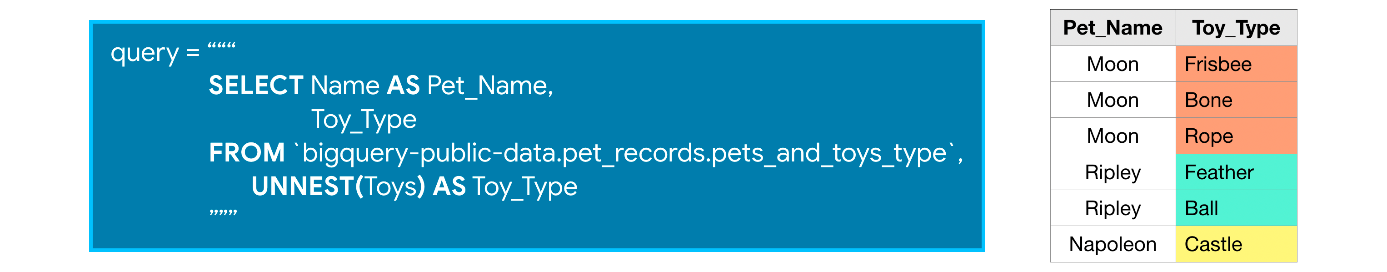


"Oyuncaklar" sütununun tekrarlanan veriler içerdiğini söylüyoruz, çünkü her satır için birden fazla değere izin veriyor. Bu durum, aşağıdaki tablo şemasında da yansıtılıyor; burada "Oyuncaklar" sütununun modu 'TEKRARLANAN' olarak görünüyor.

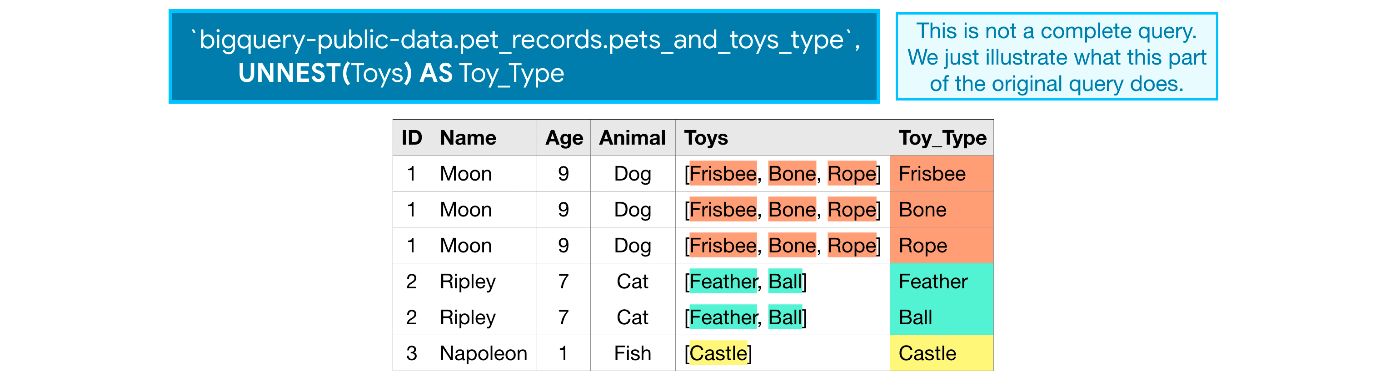


Tekrarlanan bir alandaki her girdi bir **ARRAY**'dir, yani aynı veri tipine sahip (sıfır veya daha fazla) değerin sıralı bir listesidir. Örneğin, Moon the Dog için 'Toys' sütunundaki girdi **[Frisbee, Bone, Rope]**'dur, bu da üç değere sahip bir ARRAY'dir.

Tekrarlanan verileri sorgularken, tekrarlanan verileri içeren sütunun adını bir **UNNEST()** fonksiyonunun içine koymamız gerekir.

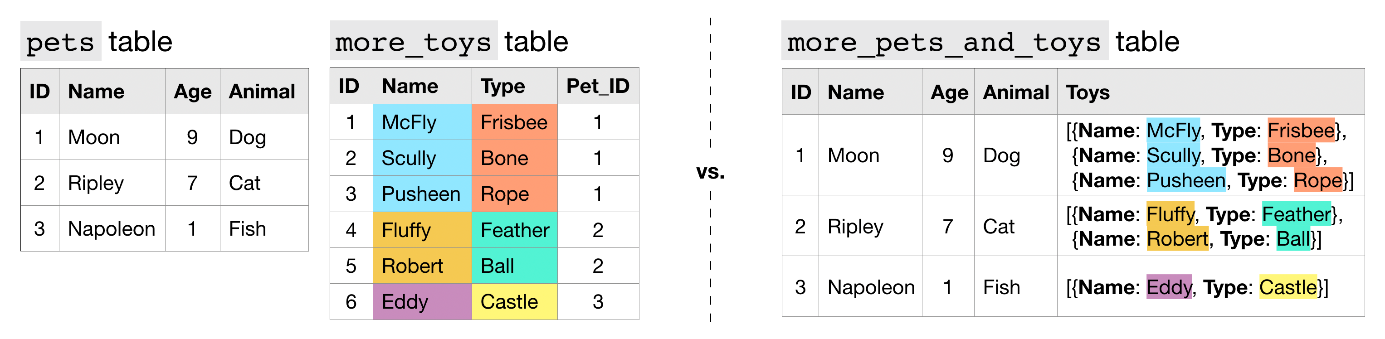


Bu, esasen tekrarlanan verileri (daha sonra tablonun sağ tarafına eklenir) düzleştirir, böylece her satırda bir öğe olur. Bunun bir örneği için aşağıdaki görsele bakın.

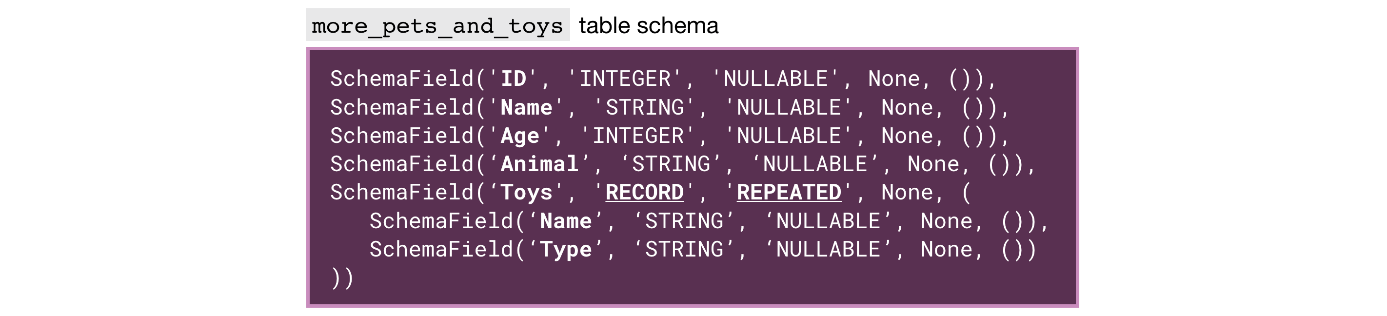


# Nested and repeated data[¶](https://www.kaggle.com/code/alexisbcook/nested-and-repeated-data" \l "Nested-and-repeated-data" \t "_self)

Şimdi, evcil hayvanların birden fazla oyuncağı olabiliyorsa **ve** biz her bir oyuncağın hem adını hem de türünü takip etmek istiyorsak ne olur? Bu durumda, "Toys" sütununu hem **iç içe (nested)** hem de **tekrarlanan (repeated)** yapabiliriz.



Yukarıdaki more\_pets\_and\_toys tablosunda, "Ad" ve "Tür", "Oyuncaklar" STRUCT'ı içinde yer alan alanlardır ve hem "Oyuncaklar.Ad" hem de "Oyuncaklar.Tür"deki her giriş bir DİZİ'dir.



Örnek bir sorguya bakalım.,



'Toys' sütunu tekrarlandığı için, **UNNEST()** fonksiyonu ile onu düzleştiririz. Düzleştirilmiş sütuna **t** takma adını verdiğimiz için, 'Toys' sütunundaki 'Name' ve 'Type' alanlarına sırasıyla **t.Name** ve **t.Type** olarak başvurabiliriz.

Öğrendiklerinizi pekiştirmek için, aşağıdaki bölümde bu fikirleri gerçek bir veri setine uygulayacağız.

# Example[¶](https://www.kaggle.com/code/alexisbcook/nested-and-repeated-data" \l "Example" \t "_self)

Google Analytics Örnek veri kümesiyle çalışacağız. Bu veri kümesi, Google markalı ürünler satan bir e-ticaret sitesi olan Google Merchandise mağazasını ziyaret edenlerin davranışlarını izleyen bilgiler içeriyor.

ga\_sessions\_20170801 tablosunun ilk birkaç satırını yazdırarak başlıyoruz. (İlgili kodu gizledik. Göz atmak için aşağıdaki "Kod" butonuna tıklayın.) Bu tablo, 1 Ağustos 2017 tarihinde web sitesine yapılan ziyaretleri izliyor.

from google.cloud import bigquery

*# Create a "Client" object*

client = bigquery.Client()

*# Construct a reference to the "google\_analytics\_sample" dataset*

dataset\_ref = client.dataset("google\_analytics\_sample", project="bigquery-public-data")

*# Construct a reference to the "ga\_sessions\_20170801" table*

table\_ref = dataset\_ref.table("ga\_sessions\_20170801")

*# API request - fetch the table*

table = client.get\_table(table\_ref)

*# Preview the first five lines of the table*

client.list\_rows(table, max\_results=5).to\_dataframe()

Using Kaggle's public dataset BigQuery integration.

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:16: UserWarning: Cannot use bqstorage\_client if max\_results is set, reverting to fetching data with the tabledata.list endpoint.

Her alanın açıklaması için bu veri sözlüğüne bakın.

Tabloda birçok iç içe geçmiş alan vardır; bunları veri sözlüğüne (ipucu: sayfada 'RECORD' ifadesinin görüntülenip görüntülenmediğini kontrol edin) veya yukarıdaki tablo önizlemesine bakarak doğrulayabilirsiniz.

Bu tabloya yönelik ilk sorgumuzda "toplamlar" ve "cihaz" sütunlarıyla çalışacağız.

print("SCHEMA field for the 'totals' column:**\n**")

print(table.schema[5])

print("**\n**SCHEMA field for the 'device' column:**\n**")

print(table.schema[7])

SCHEMA field for the 'totals' column:

SchemaField('totals', 'RECORD', 'NULLABLE', None, (SchemaField('visits', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('hits', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('pageviews', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('timeOnSite', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('bounces', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('transactions', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('transactionRevenue', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('newVisits', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('screenviews', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('uniqueScreenviews', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('timeOnScreen', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('totalTransactionRevenue', 'INTEGER', 'NULLABLE', None, (), None), SchemaField('sessionQualityDim', 'INTEGER', 'NULLABLE', None, (), None)), None)

SCHEMA field for the 'device' column:

SchemaField('device', 'RECORD', 'NULLABLE', None, (SchemaField('browser', 'STRING', 'NULLABLE', None, (), None), SchemaField('browserVersion', 'STRING', 'NULLABLE', None, (), None), SchemaField('browserSize', 'STRING', 'NULLABLE', None, (), None), SchemaField('operatingSystem', 'STRING', 'NULLABLE', None, (), None), SchemaField('operatingSystemVersion', 'STRING', 'NULLABLE', None, (), None), SchemaField('isMobile', 'BOOLEAN', 'NULLABLE', None, (), None), SchemaField('mobileDeviceBranding', 'STRING', 'NULLABLE', None, (), None), SchemaField('mobileDeviceModel', 'STRING', 'NULLABLE', None, (), None), SchemaField('mobileInputSelector', 'STRING', 'NULLABLE', None, (), None), SchemaField('mobileDeviceInfo', 'STRING', 'NULLABLE', None, (), None), SchemaField('mobileDeviceMarketingName', 'STRING', 'NULLABLE', None, (), None), SchemaField('flashVersion', 'STRING', 'NULLABLE', None, (), None), SchemaField('javaEnabled', 'BOOLEAN', 'NULLABLE', None, (), None), SchemaField('language', 'STRING', 'NULLABLE', None, (), None), SchemaField('screenColors', 'STRING', 'NULLABLE', None, (), None), SchemaField('screenResolution', 'STRING', 'NULLABLE', None, (), None), SchemaField('deviceCategory', 'STRING', 'NULLABLE', None, (), None)), None)

Aşağıdaki sorguda "tarayıcı" alanına ("cihaz" sütununda yer alır) ve "işlemler" alanına ("toplamlar" sütununda yer alır) cihaz.tarayıcı ve toplamlar.işlemler adını veriyoruz:

*# Query to count the number of transactions per browser*

query = """

SELECT device.browser AS device\_browser,

SUM(totals.transactions) as total\_transactions

FROM `bigquery-public-data.google\_analytics\_sample.ga\_sessions\_20170801`

GROUP BY device\_browser

ORDER BY total\_transactions DESC

"""

*# Run the query, and return a pandas DataFrame*

result = client.query(query).result().to\_dataframe()

result.head()

/opt/conda/lib/python3.7/site-packages/google/cloud/bigquery/client.py:440: UserWarning: Cannot create BigQuery Storage client, the dependency google-cloud-bigquery-storage is not installed.

"Cannot create BigQuery Storage client, the dependency "

| device\_browser | total\_transactions |
| --- | --- |
| 0 | Chrome | 41.0 |
| 1 | Safari | 3.0 |
| 2 | Firefox | 1.0 |
| 3 | Edge | NaN |
| 4 | Coc Coc | NaN |

"Device" ve "totals" sütunlarındaki bilgileri STRUCT'lar olarak (ayrı tablolar yerine) depolayarak, pahalı JOIN işlemlerinden kaçınırız. Bu, performansı artırır ve JOIN anahtarları (ve hangi tabloların tam olarak ihtiyacımız olan verilere sahip olduğu) konusunda endişelenmemize gerek kalmaz.

Şimdi, hem iç içe geçmiş hem de tekrarlanan verilere örnek olarak "hits" sütununu ele alacağız. Şunlardan dolayı:

"hits" bir STRUCT'tır (iç içe geçmiş veriler içerir) ve tekrarlanır,

"hitNumber", "page" ve "type", "hits" sütununun içine,

"pagePath" ise "page" alanının içine yerleştirilmiştir.

Bu alanları aşağıdaki sözdizimiyle sorgulayabiliriz:

*# Query to determine most popular landing point on the website*

query = """

SELECT hits.page.pagePath as path,

COUNT(hits.page.pagePath) as counts

FROM `bigquery-public-data.google\_analytics\_sample.ga\_sessions\_20170801`,

UNNEST(hits) as hits

WHERE hits.type="PAGE" and hits.hitNumber=1

GROUP BY path

ORDER BY counts DESC

"""

*# Run the query, and return a pandas DataFrame*

result = client.query(query).result().to\_dataframe()

result.head()

/opt/conda/lib/python3.7/site-packages/google/cloud/bigquery/client.py:440: UserWarning: Cannot create BigQuery Storage client, the dependency google-cloud-bigquery-storage is not installed.

"Cannot create BigQuery Storage client, the dependency "

| path | counts |
| --- | --- |
| 0 | /home | 1257 |
| 1 | /google+redesign/shop+by+brand/youtube | 587 |
| 2 | /google+redesign/apparel/mens/mens+t+shirts | 117 |
| 3 | /signin.html | 78 |
| 4 | /basket.html | 35 |