2021. 12. 04.

Projektmunka III.

*Dokumentáció*

Szalai Gergő

LJS2ZQ

szerg97@gmail.com

Tartalom

[1. Feladat (Batch) 3](#_Toc89446990)

[1.1. Source (Scala) 3](#_Toc89446991)

[1.2. Producer (Python) 4](#_Toc89446992)

[1.3. Consumer (Python) 5](#_Toc89446993)

[1.4. Data\_process (Scala) 8](#_Toc89446994)

[1.5. Data\_viewer (Scala) 9](#_Toc89446995)

[2. Feladat (Streaming) 10](#_Toc89446996)

[2.1. Producer (Python) 10](#_Toc89446997)

[2.2. Consumer (Python) 11](#_Toc89446998)

[2.3. Realtime\_process (Scala) 13](#_Toc89446999)

[2.4. Data\_viewer (Scala) 15](#_Toc89447000)

[3. Feladat (AVRO) 17](#_Toc89447001)

[4. Github link 21](#_Toc89447002)

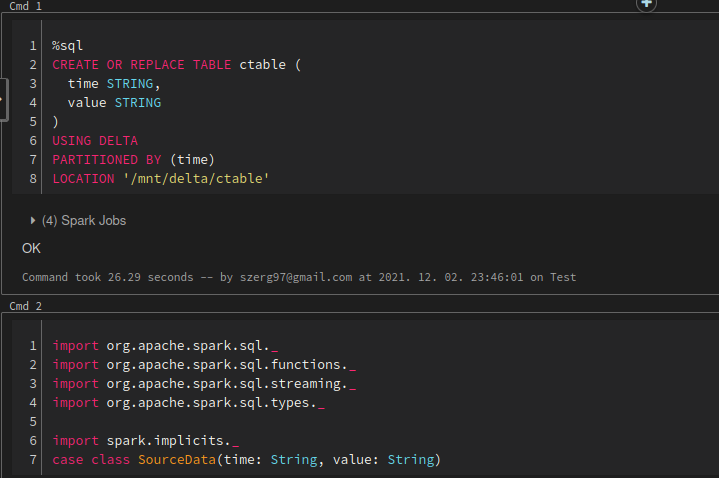
# Feladat (Batch)

A hallgató létrehoz a saját projektjében egy nagy méretű relációs táblában egy új oszlopot. Ebbe generál idő adatokat 0 és 12 óra időpillanatok között. Ez szolgál majd a batch adatok betöltéséhez. Ütemezett/batch alapú betöltéssel ezt lementi egy relációs táblába (MySQL / PostgreSQL), ami a JDBC-vel van hozzákötve a Databricks-hez.

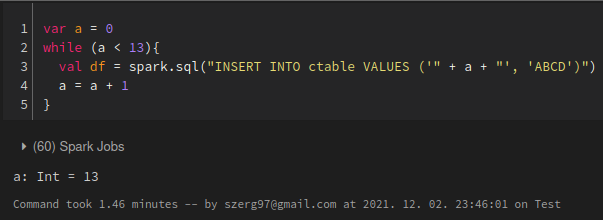
A feladat megoldásához több Notebookot is létrehoztam.

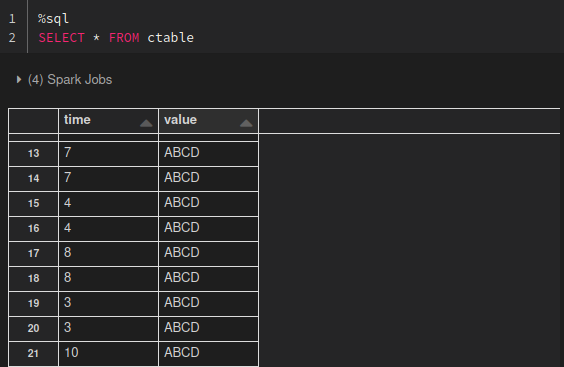
## Source (Scala)

Létrehozok egy relációs táblát, amelyben egyetlen „time”, szöveges típusú oszlop van, valamint importálom a megfelelő névtereket.



Az így kapott táblát egy ciklusban feltöltöm 0-12-ig időkkel.





## Producer (Python)

Kafka Producer segítségével, topicon keresztül szórom az adatokat. Feltelepítem a szükséges függőségeket, majd importálom a megfelelő névtereket.

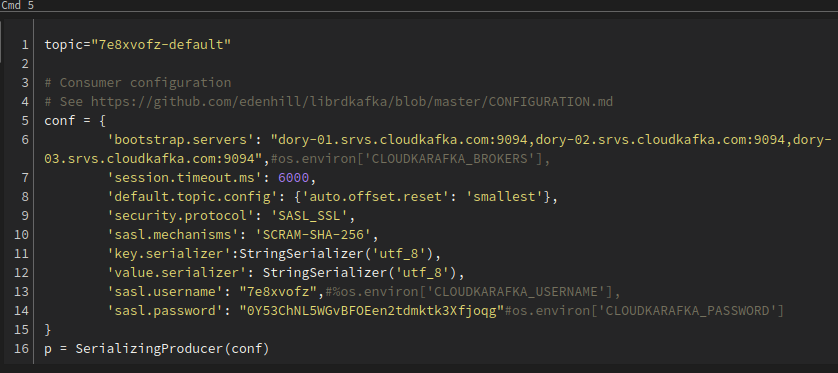
A képen szöveg látható

Automatikusan generált leírás

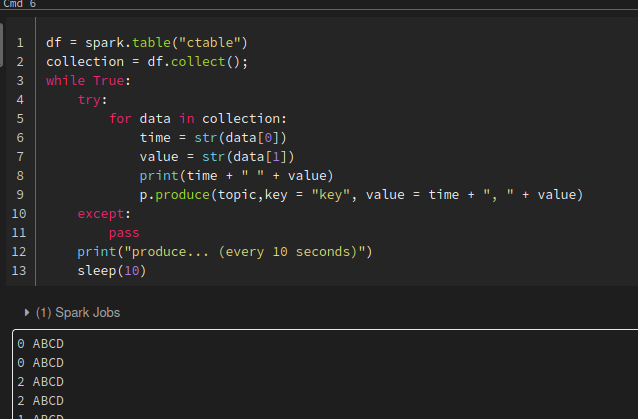
A képen szöveg, képernyőkép, monitor, ezüst látható

Automatikusan generált leírás

A következő lépésben konfigurálom a Producert. Topicnak beállítom a Cloud Karafka szolgáltatáson elhelyezett saját topicomat. A többi beállítás (szerverek, felhasználónév, jelszó) szintén a felhőben lévő értékek alapján készült.



A lényegi programrészben egy DataFrame-et csinálok a korábban létrehozott táblából. Ciklusban bejárom a gyűjteményt, mégpedig úgy, hogy kiveszek belőle egyet, amit aztán továbbításra átadok a Producernek. A tábla sorait 10 mp-ként produkálom.



## Consumer (Python)

A Consumert feliratkoztatom a topicra, hogy várja az adatokat. A Notebookban szintén feltelepítem a Kafka-t, valamint importálom a szükséges elemeket.

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

Ezután jöhet a Consumer konfigurálása a már korábban említett Producerhez hasonlóan, annyi különbséggel, hogy ezesetben több topicot is meghatározhatunk, valamint kap egy group id-t.

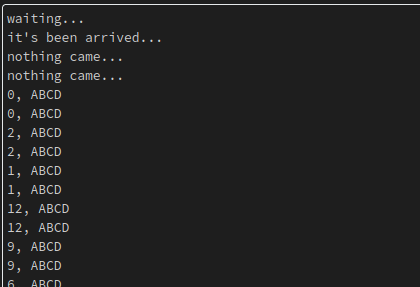
A képen szöveg látható

Automatikusan generált leírás

Beállítom a hostot és portot, majd a socket segítségével figyelem, hogy érkezik-e adat. A program blokkolva van egészen addig, ameddig nem érkezik adat. Amennyiben nem érkezik üzenet, azt jelzem a felhasználó felé. Ha érkezett válasz, azt továbbítom feldolgozásra.

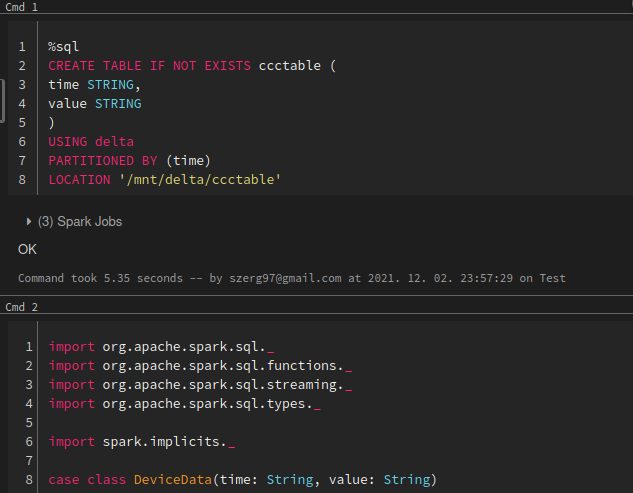
A képen szöveg látható

Automatikusan generált leírás

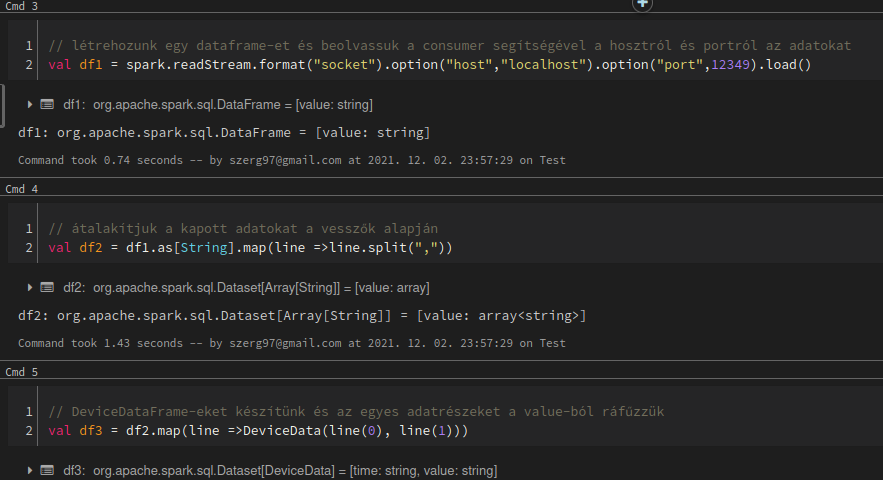


## Data\_process (Scala)

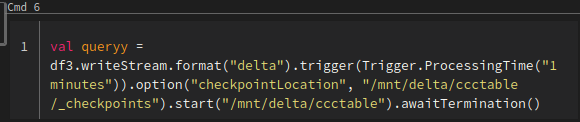
Létrehozom a delta forrástáblát 2 darab mezővel, az forrástáblához hasonlóan „time” és „value” attribútumokkal, majd az ehhez tartozó, a sémára illeszkedő case class-ot.



A Consumerről beolvasom a batch adatokat, csatlakozva a localhost 12349-es portjához. A kapott adatok vesszővel elválasztva vannak, ezért ezek mentén tagolom őket, majd dataframekre fűzöm őket.

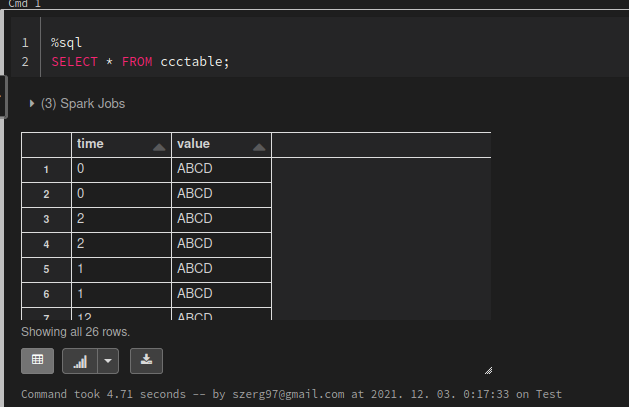


Ezt követően elmentem a céltáblába a kapott adatokat.



## Data\_viewer (Scala)

Ellenőrzésképp lekérdezem a céltábla sorait.



# Feladat (Streaming)

Kap egy API URL-t (streaming adat), az így érkező adatokat át kell alakítani olyan formára, amit a Kafka képes feldolgozni (min. 5 oszlop és 10 coin, ill. tegyen hozzá egy plusz oszlopot).

A feladat megoldása során itt is különböző Notebookokon dolgoztam.

## Producer (Python)

Telepítem a szükséges csomagokat (Kafka) és beimportálom a könyvtárakat.

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

Ezek után következik a Producer konfigurálása az előbbiekhez hasonlóan. Meghatározom a topicot, a szervereket. A session tétlenség esetén fellépő idejét 6 mp-re állítom, a biztonsági protokollt is itt állítom be. StringSerializer-t alkalmazok a kulcs-érték adatokra UTF-8-as kódolással. A felhasználónév és jelszó Cloud Karafka-ból jön.

A képen szöveg látható

Automatikusan generált leírás

A program logikájában egy API-ból kérem le a coinokat (a limitet 10-re állítva) 10 mp-es időközönként. Amennyiben 200-as response code-ot kapunk (sikeres válasz), a kapott JSON-t átalakítom DataFrame listává és kinyerem az 5 oszlopot, amelyekre szükségem van. Végig megyek minden egyes listaelemen és a Producerrel szórom a már korábban meghatározott topicon.

Itt létrehozok egy új oszlopot (rank) amely úgy jön ki, hogy az aktuális coin „changePercent24Hr” oszlopának értékéhez hozzáadok 100-at. A float típusú változót string forumátumúvá alakítom, hogy az érték feldolgozható legyen.

A képen szöveg látható

Automatikusan generált leírás

## Consumer (Python)

A fogyasztó számára a korábbiakhoz hasonló módszerrel telepítem Kafka-t, valamint importálom a szükséges névtereket.

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

A konfiguráláshoz szükségem van Cloud Karafka-n lévő instance paramétereire, ezeket felhasználva állítom be a Consumert. A Consumer a Producertől eltérő módon egy group id-t is kap. A bejövő adatokat deserializálni kell, ezek ugyanúgy StringSerializerrel, UTF-8-as kódolással történnek, majd feliratkoztatom a topicra.

A képen szöveg látható

Automatikusan generált leírás

Az 1. feladatban leírtakhoz hasonlóan a Consumer itt is a localhostra van kivezetve, azonban ezesetben az előzőtől különböző porton. A működése pedig megegyezik az 1. feladat Consumerével. A konzolon olvashatjuk a beérkező adatokat, miután a következő folyamat lefuttatásra került.

A képen szöveg látható

Automatikusan generált leírás

## Realtime\_process (Scala)

Elsőnek beimportálom a streammel kapcsolatos objektumokat.

A képen szöveg látható

Automatikusan generált leírás

Ezek után egy case class-t hozok létre, hogy a streamelt adatokat ez alapján tudjam mappelni. A mezők típusai megegyeznek a fogadandó adat adattípusaival. Bár azok stringként érkeznek, később parsoljuk őket. Ezt követi a StreamingContext beállítása, majd a context hostjának és portjának beállítása, amellyel beolvasom a Consumerről a kapott adatfolyamot.

A képen szöveg, képernyőkép, fekete, beltéri látható

Automatikusan generált leírás

A kapott válaszban az adatok vesszővel elválasztva szerepelnek, ezért ezek mentén szétválasztom őket. Ahol szükséges, ott konvertálást is alkalmazok.

A képen szöveg látható

Automatikusan generált leírás

Mivel lehetséges, hogy egy adat a streaming következtében többször is szerepel, a kérdéses sorokon számítást végzek és átlagolom az értékeket.

A képen szöveg, képernyőkép, monitor, fekete látható

Automatikusan generált leírás

A végső, átalakított DStreamen elvégzem a már korábban létrehozott „DevData” nevezetű case class attribútumaival a mappelést a bejövő adatra.

Ezután az így létrejött DStreamben lévő oszlopokat vesszővel választom el egymástól, ezáltal a fájlban is így szerepelnek majd.

A képen szöveg látható

Automatikusan generált leírás

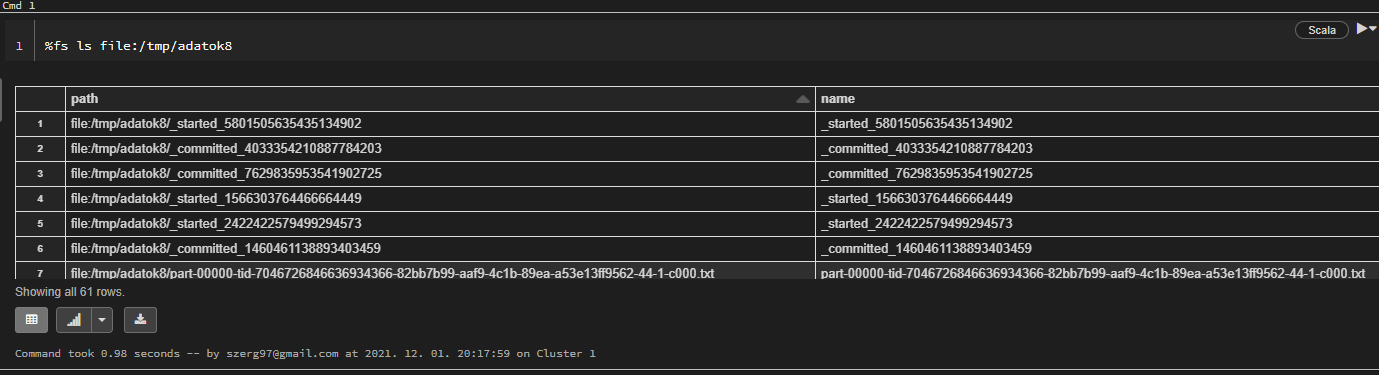
Végül elmentem a módosításokat szövegként a kívánt helyre.

A képen szöveg, képernyőkép, képernyő, több látható

Automatikusan generált leírás

## Data\_viewer (Scala)

Ellenőrzésképp kilistázom a fájl mappájában található összes fájlt, majd az egyik fájl tartalmát megjelenítem.



A képen szöveg látható

Automatikusan generált leírás

# Feladat (AVRO)

Egy nyers XML fájlt kell feljuttatni a Databricks-be, mely hibákat tartalmaz. Ezt le kell tárolni AVRO típusú fájlban és a Databricks-en belül legyen feldolgozva.

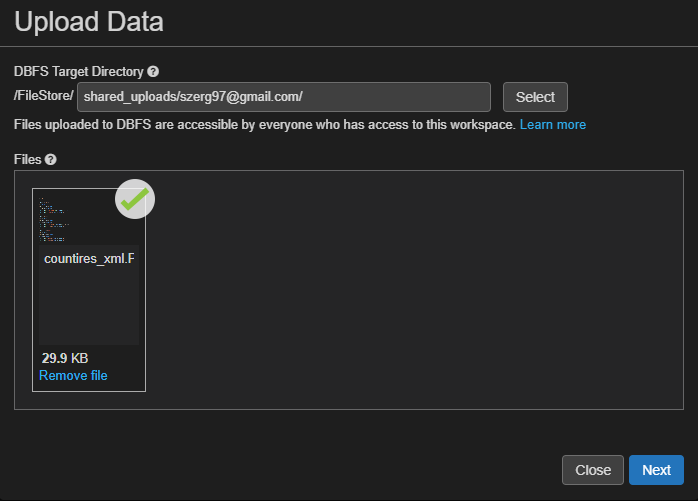
Először létrehoztam magát az XML fájlt, amely a következőképp nézett ki:



Az adatszerkezet különböző országokat tartalmaz. Egy országnak van neve, rangja, egy főre jutó GDP értéke, annak dátuma, valamint a szomszédos országok.

Kijavítandó, hogy a dátum olyan formában jelenjen majd meg, amelyet később el tudunk tárolni adatbázisban: ’dd/MMM/yyyy’ formátum helyett ’yyyy-mm-dd’.

Az XML fájt könnyedén fel lehet tölteni a Databricks fájlrendszerébe, azonban a vele való dolgozáshoz szükséges a megfelelő könyvtár telepítése:



A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

Ezek után beolvasom a fájlt és egy DataFrame-ben eltárolom. Ellenőrzésképp megjelenítem a sémát, majd magát a DataFrame-et is.

A képen szöveg látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

Az adatokon egy egyszerű módosítást hajtok végre: szeretném, ha annak a sornak a „\_name” oszlopa megváltozna „Hungary” -ra, ahol az „Finland” volt.

A képen szöveg, képernyőkép, monitor, képernyő látható

Automatikusan generált leírás

A dátumokat ezennel át kell alakítani a már korábban említett formátumra ahhoz, hogy később akár adatbázisban dátum formátumként el tudjuk tárolni. Az átalakításhoz egy DataFrame-ben (dfff) eltárolom a már korábbi DF (dff) adatait úgy, hogy azok dátum oszlopaiban kicserélem a „/” jeleket kötőjelekre. Ezután egy újabb DF-ben (dffff) teszem be a dátumokat tartalmazó oszlopot, valamint az új (formatteddate) oszlopot, majd megjelenítem.

A képen szöveg, képernyőkép, monitor, fekete látható

Automatikusan generált leírás

Az így kapott DF-et „left join” segítségével kötöm a korábbi DF-hez (dfff). Meghatározom az útvonalat, amelyre az AVRO fájlt kívánom elmenteni. Végül kiválasztom a számomra szükséges oszlopokat, majd elmentem a módosításokat.

A képen szöveg, képernyőkép, monitor, fekete látható

Automatikusan generált leírás

Ellenőrzésképp szeretném látni, hogy az elmentett fájlban valóban sikeresen hajtódott végre az átalakítás:

A képen szöveg látható

Automatikusan generált leírás

# Github link