Domácí úkol na BI-BIG

Vypracoval Matyáš Skalický 15.12.2018

Obsah

[Úvod 2](#_Toc532667844)

[Dataset 2](#_Toc532667845)

[Zdroj datasetu: 2](#_Toc532667846)

[Spuštění databázového clusteru 4](#_Toc532667847)

[Spark 4](#_Toc532667848)

[Build image pro spark 4](#_Toc532667849)

[Spuštění master a worker node 4](#_Toc532667850)

[spark-shell 4](#_Toc532667851)

[Připojení spark-shell na master 4](#_Toc532667852)

[HDFS kontejner 4](#_Toc532667853)

[Import dat do databázového clusteru 5](#_Toc532667854)

[Agregace 6](#_Toc532667855)

[1. vytvořit nový dataset, který bude agregovat data z jednoho původního datasetu 6](#_Toc532667856)

[2. vytvořit nový dataset, který bude agregovat data ze dvou původních datasetů najednou 6](#_Toc532667857)

[3. vytvořit nový dataset, který bude agregovat data ze dvou datasetů najednou, z čehož jeden bude výsledkem předchozí agregace a uložit ho zpět do databáze/na file systém 6](#_Toc532667858)

[Export dat ze spark clusteru 6](#_Toc532667859)

[Vyhladávací index 7](#_Toc532667860)

[Nahrání dat a tvorba indexu 7](#_Toc532667861)

[Dotazy na index 8](#_Toc532667862)

[Filtrování - vyhledání všech mužů: 8](#_Toc532667863)

[Třídění - seřadit zákazníky podle útraty 8](#_Toc532667864)

[Wildcard hledání - všichni zákazníci co přišli z Google 8](#_Toc532667865)

[Dashboard 9](#_Toc532667866)

[Závěr 9](#_Toc532667867)

# Úvod

Cílem tohoto úkolu je zopakovat si postupy probrané na cvičení na vlastních datech.

# Dataset

Dataset obsahuje 550 000 záznamů z Black Friday z amerického obchodního řetězce. Obsahuje jak numerické, tak kategorické proměnné. Obsahuje chybějící hodnoty. Bylo dogenerován sloupec Refferal a upraveno Product\_ID na numerickou hodnotu (odstraněním předpony P a úvodních nul). Úpravy datasetu jsou popsány v souboru DatasetTransform.ipynb který se nachází v příloze. Dataset byl pro potřeby úkolu rozdělen na 3 části.

### Zdroj datasetu:

<https://www.kaggle.com/mehdidag/black-friday>

#### Dataset users.csv



RangeIndex: 537577 entries, 0 to 537576Data columns (total 8 columns):User\_ID 537577 int64 - Unikátní identifikátor uživatele.Gender 537577 object - Pohlaví zákazníka (M/F).Age 537577 object - Věková skupina zákazníka.Occupation 537577 int64 - Kategorie dle typu zaměstnání (maskovaná, číslo).City\_Category 537577 object - Kategorie města (maskovaná - hodnoty A, B, C).Stay\_In\_Current\_City\_Years 537577 object - Délka pobytu v současném městě v letech.Marital\_Status 537577 int64 - Manželský status (1=ženatý, 0=jinak).Refferal 537577 object - Stránka, odkud uživatel přišel, když se registroval   
memory usage: 32.8+ MB

#### Dataset products.csv



RangeIndex: 537577 entries, 0 to 537576Data columns (total 5 columns):User\_ID 537577 int64 - Unikátní identifikátor uživatele.Product\_ID 537577 int64 - Unikátní identifikátor produktu.Product\_Category\_1 537577 float64 - Primární kategorie produktu. (maskovaná, číslo)Product\_Category\_2 370591 float64 - Sekundární kategorie produktu. (maskovaná, číslo)Product\_Category\_3 164278 float64 - Tercérní kategorie produktu. (maskovaná, číslo)memory usage: 20.5 MB

#### Dataset user\_purchase.csv



RangeIndex: 537577 entries, 0 to 537576Data columns (total 2 columns):User\_ID 537577 non-null int64- Unikátní identifikátor uživatele.Purchase 537577 non-null int64- Předpokládám že kumulativní suma částky utracené uživatelem v obchodě. Dataset neobsahuje informace o měně.memory usage: 8.2 MB

# Spuštění databázového clusteru

Následující postup vychází z [návodu prezentovaného na 5. cvičení](https://courses.fit.cvut.cz/BI-BIG/tutorials/05/index.html). Postup vyžaduje nainstalovaný a plně funkční [docker-compose](https://docs.docker.com/compose/).

## Spark

### Build image pro spark

Otevřeme terminál ve složce spark. Bude vytvořen image *spark* který bude využit dle parametrů při spuštění jak pro worker, tak pro master node.

docker build -f spark.df -t spark .

### Spuštění master a worker node

docker-compose up

### spark-shell

#### spuštění spark-shell

Následující příkaz spustí kontejner s přístupem do bashe na popředí.

docker run -it -p 8088:8088 -p 8042:8042 -p 4041:4040 --name driver -h driver spark:latest bash

### Připojení spark-shell na master

V běžícím bashi kontejneru se připojíme na master node spark clusteru.  
  
spark-shell --master spark://<IP adresa mastera>:7077

## HDFS kontejner

Pro spuštění kontejneru s Hadoop HDFS použijeme následující příkaz:

docker run --name hadoop -t -i sequenceiq/hadoop-docker /etc/bootstrap.sh -bash

Pro zjednodušení práce si přidáme do cesty image odkaz na hadoop.

export PATH=$PATH:/usr/local/hadoop/bin/

# Import dat do databázového clusteru

Vytvoříme složku *data* v rootu hadoop image a v rootu HDFS filesystému.

mkdir /datahdfs dfs -mkdir /data

Kopie dat do kontejneru na kterém běží HFDS.

docker cp data/products.csv hadoop:/data/products.csvdocker cp data/user\_purchase.csv hadoop:/data/user\_purchase.csvdocker cp data/users.csv hadoop:/data/users.csv

Vložení dat z filesystému image do HDFS.

hdfs dfs -put /data/products.csv /data/products.csvhdfs dfs -put /data/user\_purchase.csv /data/user\_purchase.csvhdfs dfs -put /data/users.csv /data/users.csv

Načtení csv souboru z HDFS do sparku

**val** users = spark.sqlContext.read.format("csv").option("header", "true").option("inferSchema", "true").load("hdfs://172.17.0.5:9000/data/users.csv") **val** user\_purchase = spark.sqlContext.read.format("csv").option("header", "true").option("inferSchema", "true").load("hdfs://172.17.0.5:9000/data/user\_purchase.csv") **val** products = spark.sqlContext.read.format("csv").option("header", "true").option("inferSchema", "true").load("hdfs://172.17.0.5:9000/data/products.csv")

# Agregace

### 1. vytvořit nový dataset, který bude agregovat data z jednoho původního datasetu

Zjistíme, kolik maximálně uživatel utratil. Vzhledem k tomu, že přepokládám, že se jedná o kumulativní sumu tedy i kolik celkem v obchodě utratil.

// Zjistíme maximální hodnoty sloupce Purchase **val** user\_purchase\_max = user\_purchase.groupBy("User\_ID").max("Purchase")// Zobrazíme výsledekuser\_purchase\_max.show()

### 2. vytvořit nový dataset, který bude agregovat data ze dvou původních datasetů najednou

Zjistíme, kteří zákazníci si koupili nejméně produktů

// Registrace Dataframe jako SQL tabulekusers.registerTempTable("users")products.registerTempTable("products")// SQL dotaz **val** users\_order\_count = spark.sqlContext.sql("SELECT User\_ID, first\_value(Gender) as Gender, first\_value(Age) as Age, first\_value(Occupation) as Occupation, first\_value(City\_Category) as City\_Category, first\_value(Stay\_In\_Current\_City\_Years) as Stay\_In\_Current\_City\_Years, first\_value(Marital\_Status) as Marital\_Status, first\_value(Refferal) as Refferal, COUNT(Product\_ID) as Product\_Count FROM users u FULL JOIN products p USING(User\_ID) GROUP BY User\_ID ORDER BY Product\_Count ASC")// Zobrazíme si výsledek. Volání provede dotaz.users\_order\_count.show()

### 3. vytvořit nový dataset, který bude agregovat data ze dvou datasetů najednou, z čehož jeden bude výsledkem předchozí agregace a uložit ho zpět do databáze/na filesystém

### // Registrace dataframu na SQL tabulkuuser\_purchase\_max.withColumnRenamed("max(Purchase)", "Purchase").registerTempTable("user\_purchase\_max").users\_order\_count.registerTempTable("users\_order\_count")// Dotaz **val** user\_average\_purchase\_amount = spark.sqlContext.sql("SELECT User\_ID, Purchase / Product\_Count as average\_purchase FROM users\_order\_count u JOIN user\_purchase\_max USING (User\_ID) ORDER BY average\_purchase DESC")// Zobrazíme výsledekuser\_average\_purchase\_amount.show()

# Export dat ze Spark clusteru

Na HDFS se vytvoří složka /data/users\_order\_count s výsledným csv.

users\_order\_count.coalesce(1).write.format("com.databricks.spark.csv").option("header","true").save("hdfs://172.17.0.5:9000/data/users\_order\_count")

Kopie dat z HDFS do filesystému kontejneru na kterém běží HDFS.

hdfs dfs -get /data/users\_order\_count /data/users\_order\_count

Kopie dat z filesystému kontejneru do filesystému počítače.

docker cp hadoop:/data/users\_order\_count .

# Vyhledávací index

## Nahrání dat a tvorba indexu

Použijeme kontejner s technologií ElasticSearch pro indexaci csv souboru. Zároveň spustíme i kontejner Kibana, který slouží pro vizualizaci a dotazování nad daty. Data do ElasticSearch dostaneme za pomocí docker kontejneru LogStash. [Postup vychází ze cvičení číslo 9](https://courses.fit.cvut.cz/BI-BIG/tutorials/09/index.html).

Jdeme do složky logstash. Kontejnery spustíme na pozadí s použitím příkazu:

docker-compose up -d

Data se načtou dle definice konfigurace v souboru *logstash.conf*:

**input** { **file** **{** **path** **=>** "/datasets/BlackFridayEdit.csv" **start\_position** **=>** "beginning" } **}  
  
filter** { **csv** **{** **separator** **=>** "," **columns** **=>** **[**"User\_ID", "Product\_ID", "Gender", "Age", "Occupation", "City\_Category", "Stay\_In\_Current\_City\_Years", "Marital\_Status", "Product\_Category\_1", "Product\_Category\_2", "Product\_Category\_3", "Purchase", "Refferal"**]** } **mutate** {**convert** **=>** **[**"User\_ID", "integer"**]**} **mutate** {**convert** **=>** **[**"Product\_ID", "integer"**]**} **mutate** {**convert** **=>** **[**"Occupation", "integer"**]**} **mutate** {**convert** **=>** **[**"Product\_Category\_1", "integer"**]**} **mutate** {**convert** **=>** **[**"Product\_Category\_2", "integer"**]**} **mutate** {**convert** **=>** **[**"Product\_Category\_3", "integer"**]**} **mutate** {**convert** **=>** **[**"Purchase", "integer"**]**} **}  
  
output** { **elasticsearch** **{** **hosts** **=>** "http://elasticsearch:9200" **index** **=>** "blackfriday" } **}**

Uživatelské rozhraní Kibana je na adrese: <http://127.0.0.1:5601>. Pro vytvoření indexu zvolíme *Management -> Index Patterns* a do pole “Index pattern” okopírujeme název indexu, tedy “blackfriday”. Následně klikneme next a v time filter zvolíme možnost “I don’t want to use Time Filter” jelikož náš dataset neobsahuje časové značky. Klikneme na Create index pattern a index je tak vytvořen.

## Dotazy na index

Dotazy v Kibaně lze provádět v záložce *Discover*. Konzole se nachází na *Dev Tools -> Console*.

### Filtrování - vyhledání všech mužů:

Dev Tools:

GET /blackfriday/doc/\_search   
{   
 "query": {   
 "match": {   
 "Gender": "M"   
 }   
 }   
}

Kibana: Gender:M

### Třídění - seřadit zákazníky podle útraty

Dev Tools:

GET /blackfriday/doc/\_search  
{  
 "sort": {  
 "Purchase": {  
 "order": "desc"  
 } }  
}

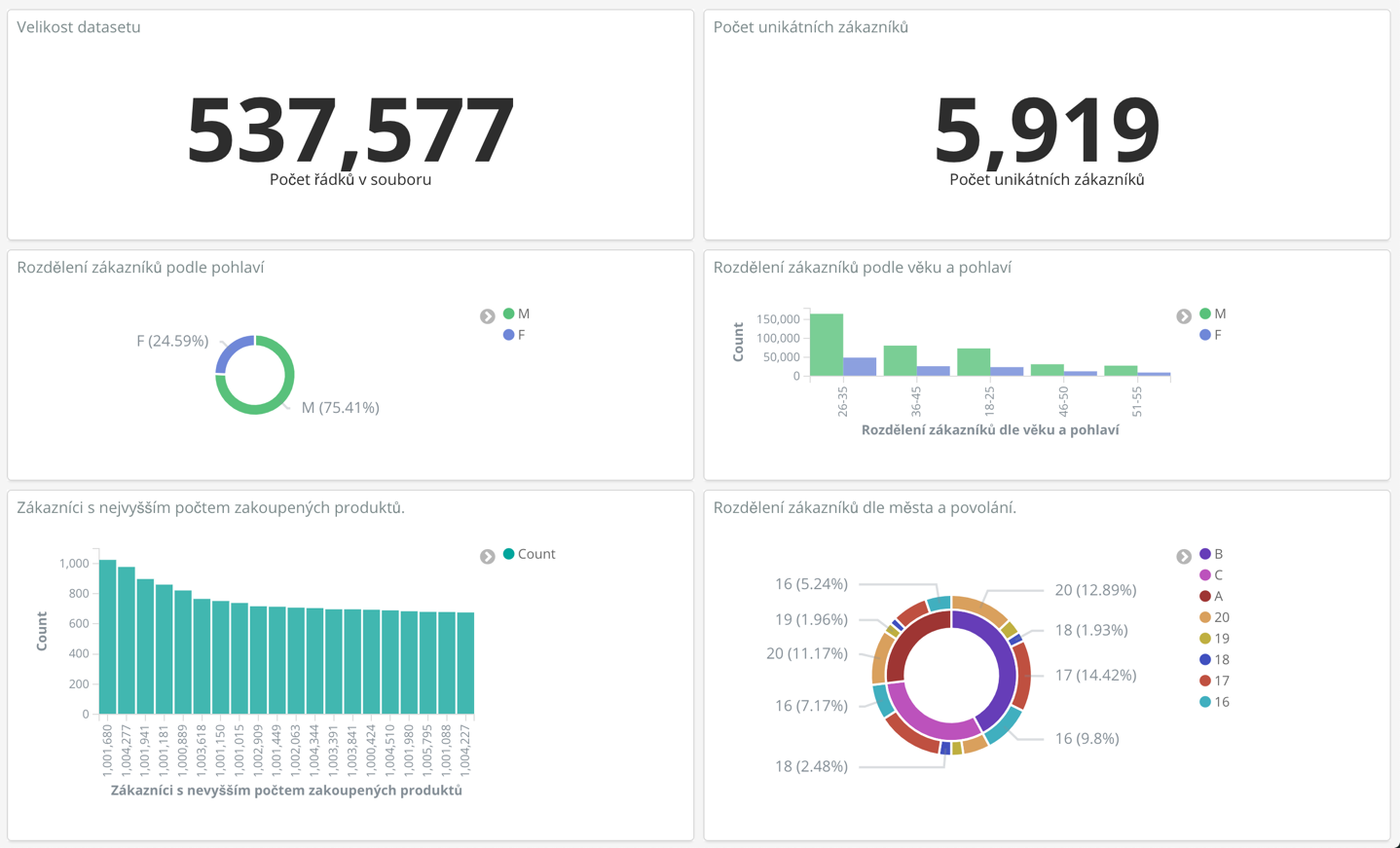
### Wildcard hledání - všichni zákazníci co přišli z Google

Dev Tools:

GET /blackfriday/doc/\_search  
{ "query": { "wildcard": { "Refferal": "www.google.\*" }  }}

Kibana: Refferal:www.google.\*

## Dashboard



Export dashboardu je uložen ve formátu JSON v souboru dashboard\_blackfriday.json ve složce logstash.

# Závěr

Vyzkoušel jsem si vytvořit vlastní Spark cluster s Hadoop HDFS úložištěm a následně ve spark-console udělat pár jednoduchých transformací. Dataset jsem taktéž importoval přes LogStash do ElasticSearch, vytvořil pár vyhledávacích dotazů a v Kibaně následně připravil dashboard s vizualizacemi. Vypracování úkolu mě moc nebavilo, protože jsem prováděl velmi podobné postupy jako na cvičení a v UseCasech. I tak mi tato semestrální práce zabrala větší množství času, než by mi připadalo užitečné. Mám pocit, že celá práce byla spíš než o big data o psaní dokumentace a o vymýšlení SQL dotazů. Radši bych svůj čas strávil děláním něčeho smysluplnějšího.