Ausgangslage

Der Schweizer Gastro-Markt gilt als gesättigter Markt. Gemäss Recherchen von NZZ unterschätzen die Unternehmen, wie anspruchsvoll der Schweizer Markt ist. So sind Investitionen in ein Restaurant beispielsweise nirgends so kostspielig wie in Zürich. Das Resultat: In der Schweiz sind zwei von drei Restaurantbetrieben defizitär. Insbesondere die hohen Kosten machen den Unternehmen zu schaffen. Laut Analysen der Firma Prognolite entstehen die Kosten zu 49% aus Mitarbeiterlöhnen und zu 27% aus Ausgaben für Lebensmittel. Dies sind beides Positionen, welche sich mit einer akkuraten Ressourcenplanung optimieren lassen.

Das Ziel der vorliegenden Projektarbeit ist es deshalb herauszufinden, wann das Restaurant welche Ressourcen (Mitarbeiter und Lebensmittel) bereitstellen muss. Die Analysen sollen insbesondere zur Ressourcenplanung und zur Lösung folgender Zielkonflikte beitragen:

- Viele Essensresten vs. reduziertes Angebot für Kunden aufgrund ausverkaufter Menüs
- Hohe Mitarbeiterkosten vs. lange Wartezeiten für die Kunden

Unter der Annahme, dass vom erzielten Umsatz auf die benötigen Ressourcen Mitarbeiter und Nahrungsmittel geschlossen werden kann, soll der Datensatz auf Strukturen in den Umsatzzahlen untersucht werden. Es sollen insbesondere folgende Fragen beantwortet werden:

- Wie hoch ist der durchschnittliche Umsatz pro Wochentag für die Zeitspanne 2016-2017?
- Wie entwickelt sich der Umsatz über die Monatstage für die Zeitspanne 2016-2017?
- Wie entwickelt sich der Umsatz im Tagesverlauf?
- Es ist bekannt, dass das Restaurant auch einen Take-Away-Service anbietet. Welchen Einfluss hat das Wetter auf den durchschnittlichen Tagesumsatz?

Quellen:
NZZ: https://www.nzz.ch/wirtschaft/restaurantketten-richten-mit-der-grossen-kelle-an-ld.1376560 (https://www.nzz.ch/wirtschaft/restaurantketten-richten-mit-der-grossen-kelle-an-ld.1376560)
Prognolite. (2019). Inputpräsentation Datensatz. [Unveröff. Präsentation], Prognolite GmbH.

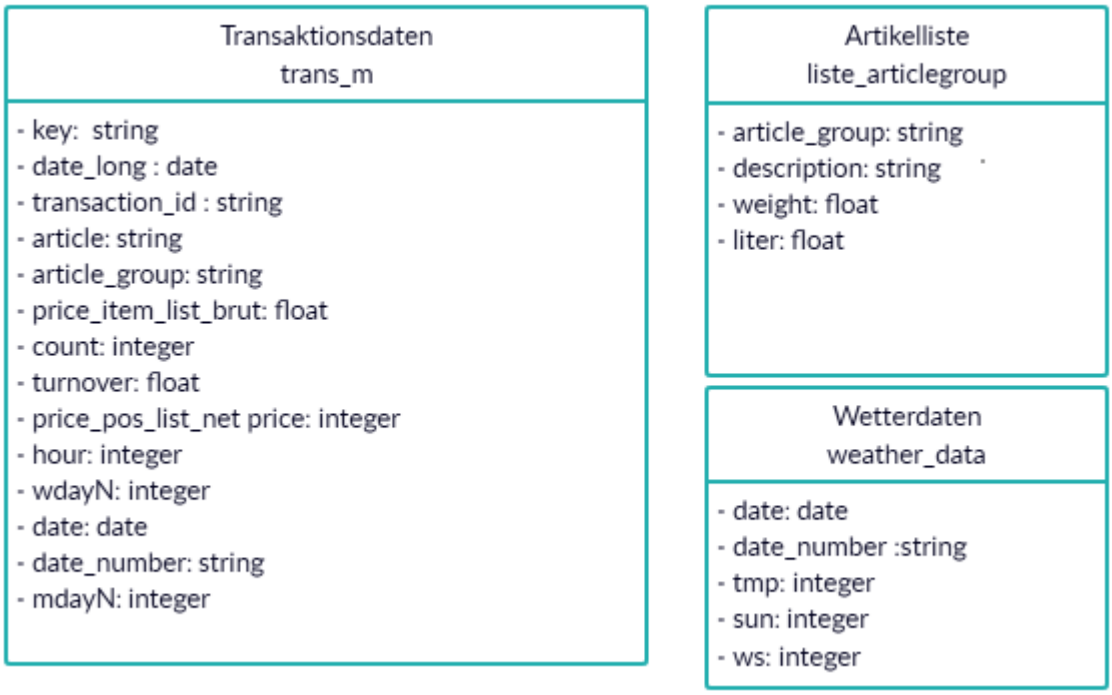
Datenquellen

Bei den verwendeten Daten handelt es sich um Daten eines Restaurants in Zürich. Der Namen des Restaurants ist nicht bekannt. Die Daten stammen von der Firma Prognolite GmbH und wurden uns im Rahmen des Moduls DE_CD01 Customer Data Analytics im Format eines csv-Files zur Verfügung gestellt.

Die vorliegende Projektarbeit wurden basierend auf folgenden drei Datensätzen erstellt:

- **Datensatz trans_m.csv**: Hauptdatensatz mit allen getätigten Kassentransaktionen des Restaurants in den Jahren 2016 und 2017. Der Datensatz enthält Informationen zum Zeitpunkt der Transaktion, Artikelinformationen und Preisen

- **Datensatz liste_articlegroup.csv**: Ergänzungsdatensatz mit Zusatzinformationen zu den Artikelgruppen (manuell angepasst)
- **Datensatz weather_data.csv**: Ergänzungsdatensatz mit Wetterinformationen auf Tagesbasis



Die Variablenbeschreibungen für die drei Datenquellen sind im Kapitel Screenshows verfügbar.

Die Beurteilung, ob es sich bei einem spezifischen Fall um ein "Big Data Problem" handelt, kann man anhand den drei Faktoren Volume (Dateigrösse), Velocity (Geschwindigkeit des Datenwachstums), Variety (Verschiedenheit der Datenquellen und -Formate), beantworten. Im vorliegenden Fall kann die Beurteilung wie folgt gemacht werden:

- Volume: Grundsätzlich gegeben
  Aus System- und Performancegründen wurde hier aber nur ein Datensatz mit reduzierter Zeilenzahl (nur Jahre 2016 und 2017) und reduzierter Spaltenanzahl verwendet.
- Velocity: gegeben
  Die Datenbasis wächst täglich. Es können – zum Beispiel über einen automatisierten ETL-Prozess – periodisch neue Transaktionsdaten eingepflegt werden.
- Variety: nicht gegeben
  Der verwendete Datensatz ist relativ strukturiert. Mit Ausnahme der Wetterdaten kommen alle Informationen aus dem Transaktionssystem des Restaurants.

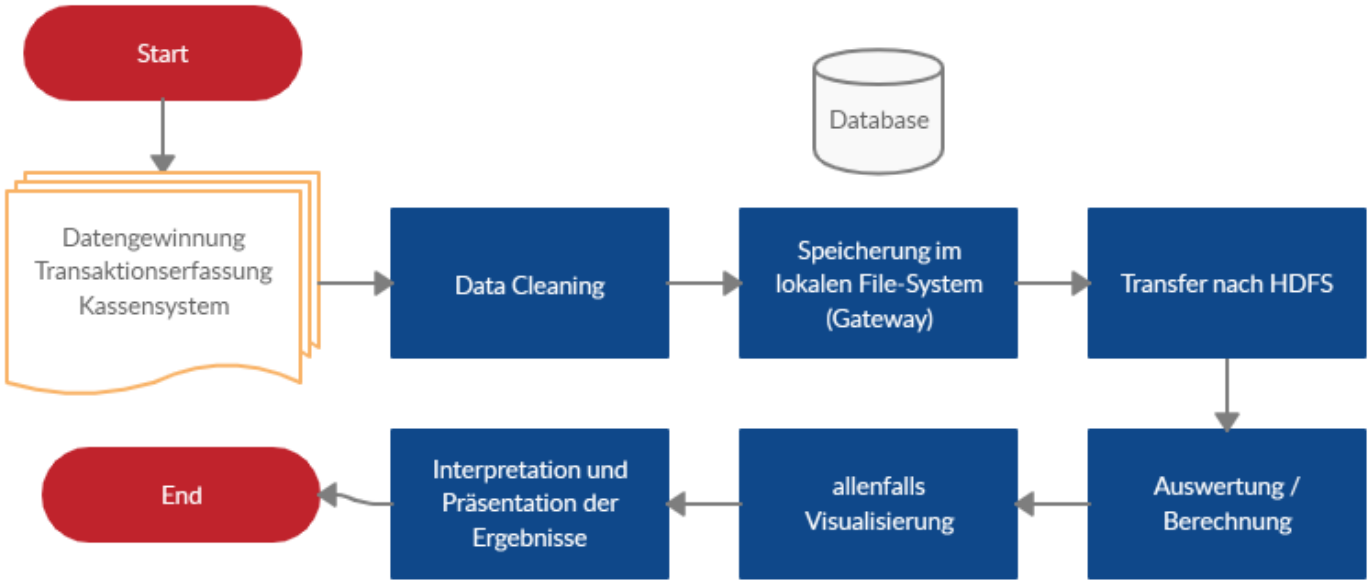
Hinweis: Wahl des Datensatzes
Neben der Tatsache, dass eine optimale Ressourcenplanung aufgrund der Wesentlichkeit der Kosten für ein Restaurant überlebenswichtig ist, war für mich auch eine Situation bei meinem Arbeitsgeber Schindler Aufzüge AG entscheidend für die Datensatzwahl. Wie im vorliegenden Beispiel haben wir auch bei Schindler verschiedene Datenquellen für unsere Verkaufsreports, welche ich als NI/MOD Controller im Monatsabschluss jeweils (mühsam) im Excel konsolidieren und anschliessend auswerten muss. Weil ich die Datensätze von Schindler aus Datenschutzgründen nicht verwenden durfte, habe ich mich für einen ähnlichen Use Case mit transaktionalen Verkaufszahlen entschieden. So kann aufgezeigt werden, wie die Verkaufsdaten in einem Hadoop-Cluster mit Hilfe von PySpark ausgewertet werden könnten.

Quellen:
Prognolite. (2019). Inputpräsentation Datensatz. [Unveröff. Präsentation], Prognolite GmbH.
DataCamp. Big Data Fundamentals with PySpark - Fundamentals of Big Data (2020). Abgerufen am
- 05.02.2020 von https://www.datacamp.com/courses/big-data-fundamentals-via-pyspark (https://www.datacamp.com/courses/big-data-fundamentals-via-pyspark)
Udemy. Hadoop Starter Kit. Abgerufen am 05.02.2020 von https://www.udemy.com/course/hadoopstarterkit/ (https://www.udemy.com/course/hadoopstarterkit/)

Datenfluss

Die Grafik zeigt ein mögliches Datenflussmodell für den Hauptdatensatz “trans_m” und den Datensatz “liste_articlegroup”. Alle Daten werden im Rahmen des Zahlungsvorgangs gewonnen, falls notwendig bearbeitet (z.B. Eliminierung potenzieller #NA-Werte) und anschliessend in einem lokalen Files System gespeichert. In unserem Fall wurden die Daten auf einem Google-Drive gespeichert und über die URL-Angabe des veröffentlichten Files auf das Gateway geladen. Anschliessend werden die Files im HDFS verteilt. In unserem Fall wurden Datenblöcke von maximal 8 MB erstellt.

Die Wetterdaten waren im vorliegenden Projekt direkt im Datensatz enthalten. Anderenfalls könnten die Daten über ein API (Application Programming Interface) von einer Webseite (z.B. https://data.stadt-zuerich.ch/dataset/sid_wapo_wetterstationen (https://data.stadt-zuerich.ch/dataset/sid_wapo_wetterstationen)) geholt werden.



Techniken

Für die Erstellung der Projektarbeit wurden vor allem die nachfolgende Techniken aufgrund der aufgeführten Gründen verwendet. Viele Techniken/Komponenten werden im Kapitel Systemarchitektur beschrieben.

- HDFS: Ermöglicht paralleles Prozessieren dank verteilter Datenspeicherung
- PySpark: Eignet sich gut für die Verarbeitung von grossen Datenmengen
- Spark SQL: Spark-Modul für strukturierte Datenverarbeitung

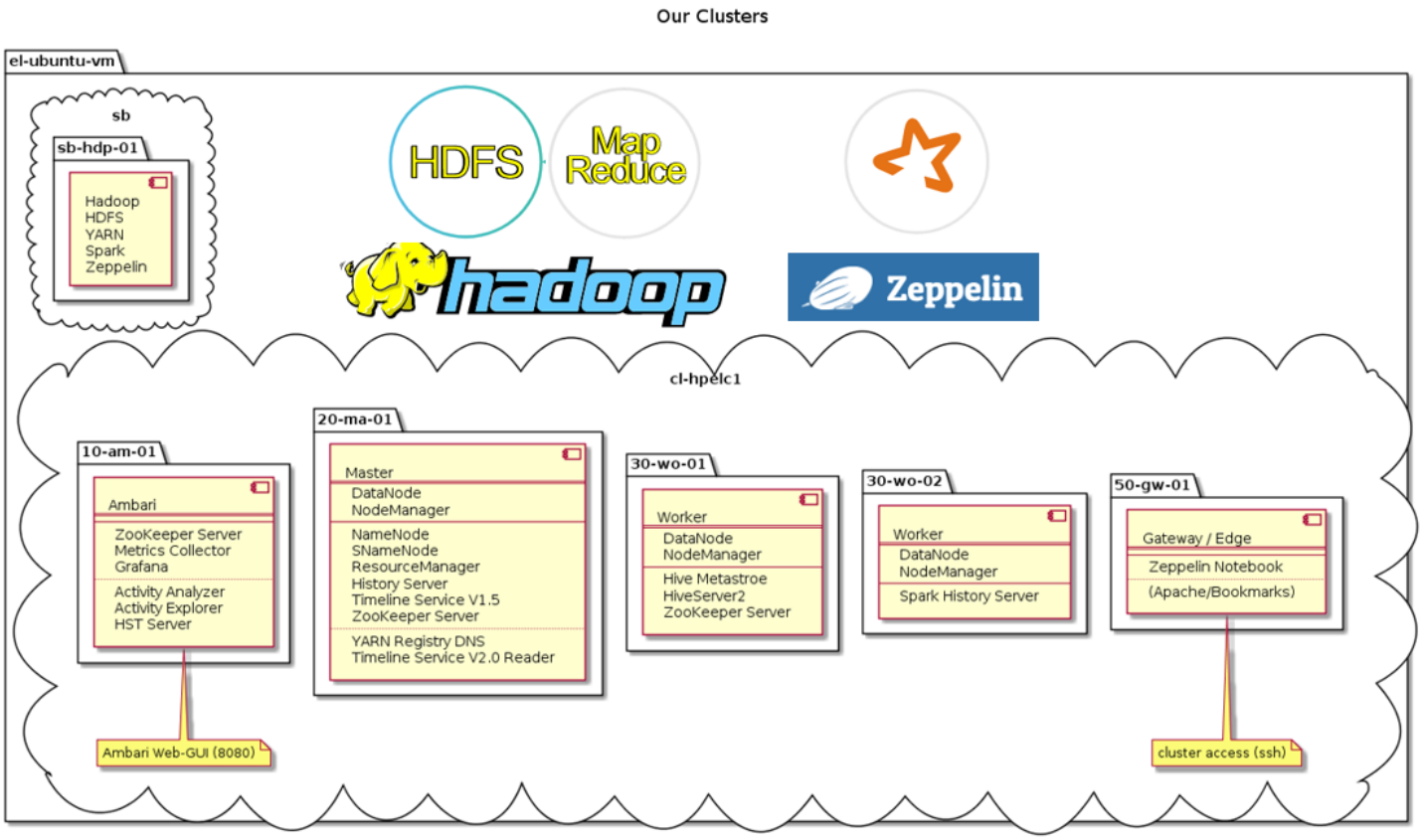
- Spark Data Frames: Ermöglichen einfaches Interagieren mit den Daten in einer Tabellenstruktur
- Shell (Git Bash): Direkter Weg, um mit der Umgebung/System zu verbinden und zu interagieren
- Markdown: Einfache Auszeichnungssprache direkt in Zeppelin Notebook
- Zeppelin: Notebook für Hadoop-Eco-System, welches direkt über Visualisierungsmöglichkeiten verfügt. Eignen sich gut für die Beantwortung der Fragestellungen.

Systemarchitektur

Die untenstehende Grafik zeigt die Systemarchitektur unserer ubuntu-virtual-maschine. Es gibt folgende Komponenten:

- **HDFS** (Hadoop Distributed Files System): Verteilte Datenspeicherung auf Computer-Cluster
- **YARN**: Ressourcennavigator für Datenverarbeitung auf Cluster
- **Map Reduce**: Programmierungsmodell zum parallelen Verarbeiten von grossen Datenmengen auf Cluster
- **Pyspark**: Programmiersprache Python auf Apache Spark spezialisiert für Verarbeitung von grossen Datenmengen
- **Zeppelin**: Notebook für Hadoop-Eco-Systems mit interaktiven Visualisierungsmöglichkeiten

Der Cluster besteht aus fünf verschiedenen Notes. Im Masternode gibt es einen NameNode in welchem Informationen über die Files auf den Workers gespeichert sind (z.B. file name, file size, replication factors, permissions, created by, last modified). In unserem Fall ist der Master auch ein DataNode und der replication factor ist 1. In der Praxis ist der Master häufig kein DataNode und der Standard-Replicationfactor ist drei. Das heisst, die Files werden dreifach abgelegt. Die Workers (Slaves) dienen der Datenverarbeitung.



Quellen:
https://bdl03-2-hs19.it-grossniklaus.ch/doc/bdl03-2/topics/eco-system/index.html (https://bdl03-2-hs19.it-grossniklaus.ch/doc/bdl03-2/topics/eco-system/index.html)
- https://www.udemy.com/course/hadoopstarterkit/ (https://www.udemy.com/course/hadoopstarterkit/)

Arbeitsschritte

Im Projekt konnten folgende Hauptarbeitsschritte eruiert werden:

- Suche und Analyse Datensatz und Definition Fragestellungen
- Import Datensatz in File-System und HDFS
- Bearbeiten Datenschema
- Untersuchung des Datensatzes mit PySpark
- Interpretation der Ergebnisse und Dokumentation

Projektablauf

Die Suche nach einem geeigneten Datensatz dauerte länger als erwartet. Mein Ziel war es, einen Datensatz zu wählen, der mich interessiert und einen Bezug zu meinem Geschäftsumfeld hat (siehe Kapitel Datenquellen). Viele Datenquellen waren zu klein oder zu gross. Mit dem gewählten Datensatz aus dem Modul Customer Analytics habe ich einen Datensatz gefunden, der in mehrere Datensätze unterteilt werden konnte, von der Grösse her passte und gut zu den Problemstellungen in meinem beruflichen Umfeld passt.

Die wohl grösste Herausforderung in der Projektarbeit war der Import der Daten auf das lokale File-System und das HDFS.

- **Lokales File-System**: Der im Unterricht gezeigte Weg für den Import via URL funktioniert nicht auf Anhieb, weil die Verlinkung zum veröffentlichten File auf Google Drive nicht zum direkten Download der Datei führte. Nach langen Recherchen im Internet haben wir einen Workaround gefunden, mit welchem Dank Anpassung der URL die Daten trotzdem direkt von Google Drive importiert werden konnten.
- **HDFS**: Beim Kopieren der Files vom lokalen Filesystem auf das HDFS erhielt ich auf dem HDFS immer leere Dateien (Befehl hdfs dfs -copyFromLocal). Dank Internetrecherchen haben wir herausgefunden, dass der Befehlt zusätzlich mit **"-f"** ergänzt werden muss, dass Dateien mit gleichem Namen (falls schon vorhanden) auf dem HDFS überhaupt ersetzt werden.

Das Bearbeiten des Datensatzes mittels dem Datenschema war relativ schnell erledigt. Allerdings musste ich merken, dass man sich bei der Definition des Datenschemas genau Gedanken machen muss, welches Datenformat man in der Quelle hat. So passierte es mir, dass ich eine Variable als Integer definierte, obwohl sie in der Datenquelle Dezimalstellen enthielt. Das Resultat war eine Tabelle aus Nullwerten (siehe Screenshots). Zudem musste ich einige Variablen von String auf Integer anpassen, um eine korrekte Sortierung bei den Auswertungen sicherstellen zu können.

Die Untersuchung des Datensatzes mit PySpark und PySpark SQL war ein interessanter Arbeitsschritt. Mit PySpark-SQL und den DataFrames konnten die Daten gut bearbeitet werden und mit den Visualisierungstools in Zeppelin relativ ansprechend dargestellt und interpretiert werden.

- Quellen:
Datenimport ab Google Drive: https://www.matthuisman.nz/2019/01/download-google-drive-files-wget-curl.html (https://www.matthuisman.nz/2019/01/download-google-drive-files-wget-curl.html)
Ersetzen bei Befehl “copyFromLocal”: https://stackoverflow.com/questions/36816526/hdfs-dfs-put-with-overwrite/36816685 (https://stackoverflow.com/questions/36816526/hdfs-dfs-put-with-overwrite/36816685)

Prototyp

Mein Ziel vor der Projektarbeit war es, möglichst schnell meine Fragestellungen beantworten zu können und erst im Anschluss den Datenimport, die Abfragen und Darstellungen noch zu verschönern. Dieses Prinzip musste ich vor allem am Anfang verstärkt anwenden: Weil es Probleme beim Importieren der Files auf das lokale Filesystem gab, musste ich auf der Ubuntu-Maschine mit dem File-Manager eine Verbindung auf das Gateway mit dem User Zeppelin erstellen. So konnte ich die Files manuell herunterladen, im entsprechen Verzeichnis speichern und anschliessend auf das HDFS laden. So konnte ich trotz Problemen weiterarbeiten und möglichst rasch einen Prototyp erstellen. Erst im Anschluss konnten wir dank Recherchen die Files direkt ab dem Google Drive importieren (siehe Projektablauf).

Finales Notebook

Die nachfolgenden Abschnitte zeigen die finale Version des Projekt-Notebooks.

1. Import Data auf local file System

Mit dem nachfolgenden Codeblock sollen die drei Filesysteme von meinem Google Drive (Github erlaubt nur Files < 25 MB) auf das lokale File System der Ubuntu-Maschine geholt werden.

Quellen:

- Die Verlinkung zum veröffentlichten File auf Google Drive: https://www.matthuisman.nz/2019/01/download-google-drive-files-wget-curl.html (https://www.matthuisman.nz/2019/01/download-google-drive-files-wget-curl.html)

```
/home/zeppelin
total 5789
-rw-r--r-- 1 zeppelin hadoop 5777367 Feb 12 15:57 100-0.txt
-rw-r--r-- 1 zeppelin hadoop   22118 Feb 11 09:57 bahnhofbenutzer.csv
drwxr-xr-x 2 zeppelin hadoop       2 Feb 13 14:09 bank2
drwxrwxrwx 2 zeppelin hadoop       5 Feb 13 17:10 data
/home/zeppelin/data
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   388    0   388    0     0    202      0 --:--:--  0:00:01 --:--:--   202
100 33.6M    0 33.6M    0     0  12.4M      0 --:--:--  0:00:02 --:--:-- 96.1M
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

```
100   388    0   388    0     0    509      0 --:--:-- --:--:-- --:--:--   509
100  1744  100  1744    0     0   1772      0 --:--:-- --:--:-- --:--:--  1772
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
```

-

2. Daten auf HDFS

Daten vom lokalen Filesystem auf das HDFS übertragen

Quellen:
Dateien ersetzen mit "-f": https://stackoverflow.com/questions/36816526/hdfs-dfs-put-with-overwrite/36816685 (https://stackoverflow.com/questions/36816526/hdfs-dfs-put-with-overwrite/36816685)

```
-rw-r--r--   1 zeppelin hdfs      18181 2020-02-11 10:11 notebook/2EYU4J53S/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-12 07:43 notebook/2EZMMZKZV
-rw-r--r--   1 zeppelin hdfs      46460 2020-02-12 07:43 notebook/2EZMMZKZV/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-15 13:30 notebook/2F18EUUKS
-rw-r--r--   1 zeppelin hdfs     259071 2020-02-15 13:30 notebook/2F18EUUKS/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-12 16:36 notebook/2F1MX4X4W
-rw-r--r--   1 zeppelin hdfs      19670 2020-02-12 16:36 notebook/2F1MX4X4W/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-13 13:37 notebook/2F1XDDJ16
-rw-r--r--   1 zeppelin hdfs     117649 2020-02-13 13:37 notebook/2F1XDDJ16/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-11 09:18 notebook/2F2NBGGCX
-rw-r--r--   1 zeppelin hdfs       4509 2020-02-11 09:18 notebook/2F2NBGGCX/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-11 09:21 notebook/2F2U8PYVS
-rw-r--r--   1 zeppelin hdfs      14214 2020-02-11 09:21 notebook/2F2U8PYVS/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-12 16:15 notebook/2F37VJXNT
-rw-r--r--   1 zeppelin hdfs      36227 2020-02-12 16:15 notebook/2F37VJXNT/note.json
drwxr-xr-x   - zeppelin hdfs          0 2020-02-11 09:34 notebook/2F3CMM8Y5
-rw-r--r--   1 zeppelin hdfs      29188 2020-02-11 09:34 notebook/2F3CMM8Y5/note.json
```

3. DataFrames definieren

Einlesen der drei Datensätze und Übersichtstabelle zeigen

```
|  2|Menu 2|22.66|null|
|  3|Menu 3|99.71|null|
|  4|Menu 4|24.13|null|
|  5|Menu 5|23.13|null|
+---+------+-----+----+
only showing top 5 rows

+----------+-----+---+---+----+
|       _c0|  _c1|_c2|_c3| _c4|
+----------+-----+---+---+----+
|18.01.2016|42387|377| -3|1.96|
|19.01.2016|42388|  0|  0|3.51|
|20.01.2016|42389|  0|  2|3.63|
|21.01.2016|42390|502| -1|1.98|
|22.01.2016|42391|514|  0|1.32|
+----------+-----+---+---+----+
only showing top 5 rows
```

4. Datenschema

Definition der neuen Schemas für die DataFrames

```
root
 |-- _c0: integer (nullable = true)
 |-- _c1: string (nullable = true)
 |-- _c2: integer (nullable = true)
 |-- _c3: integer (nullable = true)
 |-- _c4: string (nullable = true)
 |-- _c5: double (nullable = true)
 |-- _c6: integer (nullable = true)
 |-- _c7: double (nullable = true)
 |-- _c8: double (nullable = true)
 |-- _c9: integer (nullable = true)
 |-- _c10: integer (nullable = true)
 |-- _c11: string (nullable = true)
 |-- _c12: integer (nullable = true)
 |-- _c13: integer (nullable = true)
```

5. DataFrames definieren

Überschreibung der DataFrames mit neuen Schemas

key	date_long	transaction_id	article	article_group
1	2016-01-18	38137553	231	33
2	2016-01-18	38137553	140	1
3	2016-01-18	38137553	486	3
4	2016-01-18	38137554	337	29
5	2016-01-18	38137555	338	29
6	2016-01-18	38137556	275	22
7	2016-01-18	38137557	275	22
8	2016-01-18	38137557	419	24

```
|            2|     Menu 2| 22.66| null|
|            3|     Menu 3| 99.71| null|
|            4|     Menu 4| 24.13| null|
|            5|     Menu 5| 23.13| null|
+-------------+-----------+------+-----+
only showing top 5 rows

+----------+-----------+---+---+----+
|      date|date_number|sun|tmp|  ws|
+----------+-----------+---+---+----+
|2016-01-18|      42387|377| -3|1.96|
|2016-01-19|      42388|  0|  0|3.51|
|2016-01-20|      42389|  0|  2|3.63|
|2016-01-21|      42390|502| -1|1.98|
|2016-01-22|      42391|514|  0|1.32|
+----------+-----------+---+---+----+
only showing top 5 rows
```

6. Überblick über den Datensatz gewinnen

Der Hauptdatensatz trans_m hat gut 421'000 Observationen. Der durchschnittliche Umsatz pro Observation beträgt CHF 16.35, wobei das Minimum bei CHF 0 und das Maximum bei CHF 2450 liegt. Dank printShema () kann geprüft werden, ob die obenstehnden Schema-Mutationen erfolgreich waren.

- Spalte date_long ist zum beispiel ab sofort vom Type "date".
  Die Basisstatistiken helfen, die Daten im DataFrama zu validieren und sich einen Überblick zu verschaffen.

```
+-------+-----------------+

root
 |-- key: string (nullable = true)
 |-- date_long: date (nullable = true)
 |-- transaction_id: string (nullable = true)
 |-- article: string (nullable = true)
 |-- article_group: string (nullable = true)
 |-- price_item_list_brut: float (nullable = true)
 |-- count: integer (nullable = true)
 |-- turnover: float (nullable = true)
 |-- price_pos_list_net: float (nullable = true)
 |-- hour: integer (nullable = true)
 |-- wdayN: integer (nullable = true)
 |-- date: date (nullable = true)
 |-- date_number: string (nullable = true)
 |-- mdayN: integer (nullable = true)
```

7. DataFrame transactions aggregieren und auswerten

Pro Produkt wird im Datensatz ein Tupel generiert. Der Datensatz trans_m enthält also pro Transaktion (transaction_id entspricht Zahlungsvorgang eines Kunden) mehrere Observationen. Dieser Detailierungsgrad ist zu hoch für die Analysen. Mittels einer Aggregation (Beispiel über PySpark oder Execution eines SQL-Queries) wird ein neuer DataFrame erstellt, der ein Tupel pro Transaktion generiert. Dieser DataFrame kann dann für aussagekräftiere Analysen verwendet werden.

Quellen:
Dokumentation:
https://spark.apache.org/docs/2.3.2/api/python/pyspark.sql.html#pyspark.sql.DataFrame.groupBy (https://spark.apache.org/docs/2.3.2/api/python/pyspark.sql.html#pyspark.sql.DataFrame.groupBy)
DataCamp Introduction to PySpark: https://campus.datacamp.com/courses/introduction-to-pyspark/manipulating-data-2?ex=9 (https://campus.datacamp.com/courses/introduction-to-pyspark/manipulating-data-2?ex=9)
DataCamp Big Data Fundamentals with PySpark: https://campus.datacamp.com/courses/big-data-fundamentals-with-pyspark/pyspark-sql-dataframes?ex=5 (https://campus.datacamp.com/courses/big-data-fundamentals-with-pyspark/pyspark-sql-dataframes?ex=5)

```
spark.sparkContext.version 2.3.2.3.1.4.0-315
spark.sparkContext.pythonVer 3.6
spark.sparkContext.sparkUser() zeppelin
```

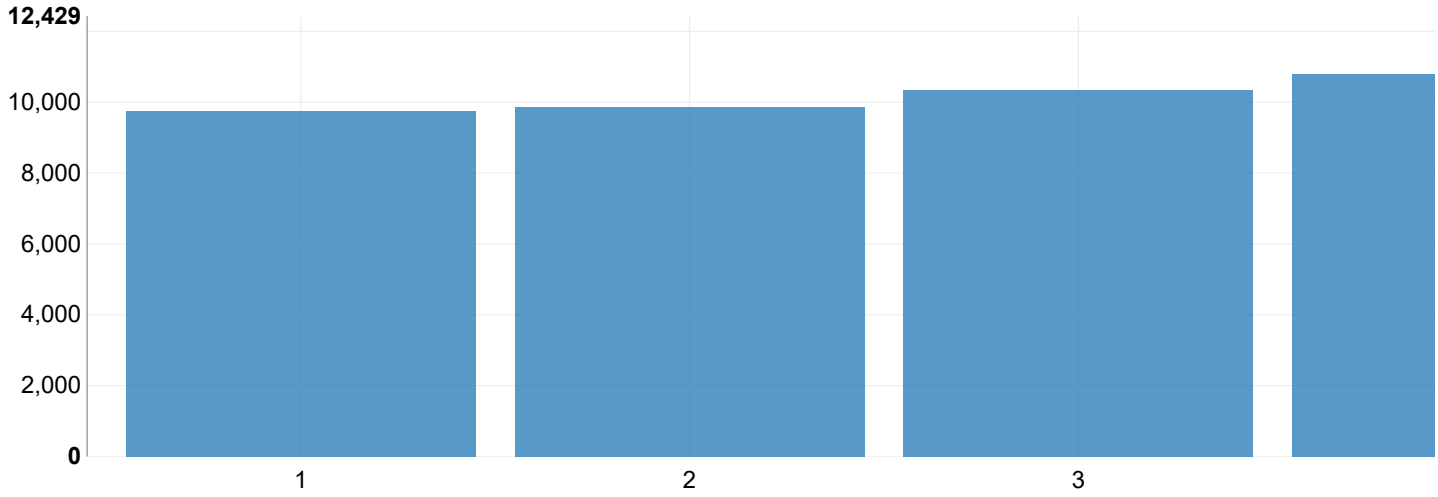
```
+--------------+----------+----+-----+-----+----------+------------------+
|transaction_id| date_long|hour|wdayN|mdayN|      date|     sum(turnover)|
+--------------+----------+----+-----+-----+----------+------------------+
|      38166807|2016-01-22|  15|    5|   22|2016-01-22|             11.75|
|      38166706|2016-01-22|  20|    5|   22|2016-01-22|               0.0|
```

```
|      38173417|2016-01-23|  15|    6|   23|2016-01-23|20.010000228881836|
|      38173453|2016-01-23|  21|    6|   23|2016-01-23|             108.0|
|      38180813|2016-01-24|  21|    7|   24|2016-01-24|193.88999938964844|
+--------------+----------+----+-----+-----+----------+------------------+
only showing top 5 rows

root
 |-- transaction_id: string (nullable = true)
 |-- date_long: date (nullable = true)
 |-- hour: integer (nullable = true)
 |-- wdayN: integer (nullable = true)
 |-- mdayN: integer (nullable = true)
 |-- date: date (nullable = true)
```

```
Durchschnittlicher Umsatz pro Transaktion
+-----------------+
|    avg(turnover)|
+-----------------+
|79.99200768464101|
+-----------------+

Minimaler Umsatz pro Transaktion
+-------------+
|min(turnover)|
+-------------+
|          0.0|
+-------------+

Maximaler Umsatz pro Transaktion
+-------------+
|max(turnover)|
+-------------+
```

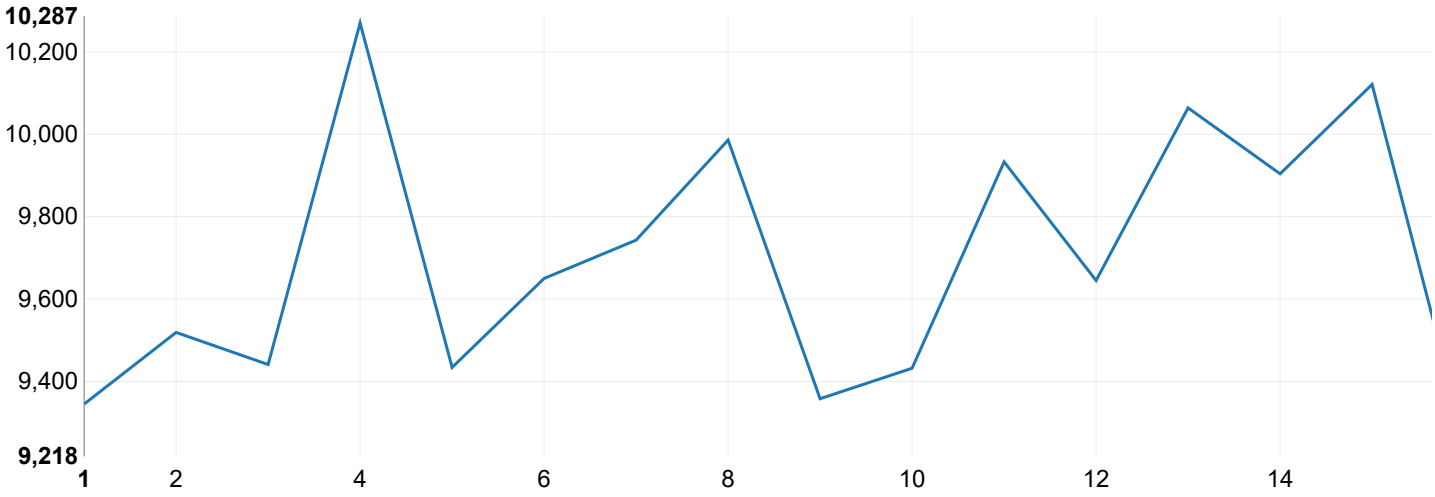
8. Beantwortung Frage 1: Umsatz pro Wochentag

- Wie hoch ist der durchschnittliche Umsatz pro Wochentag für die Zeitspanne 2016-2017?



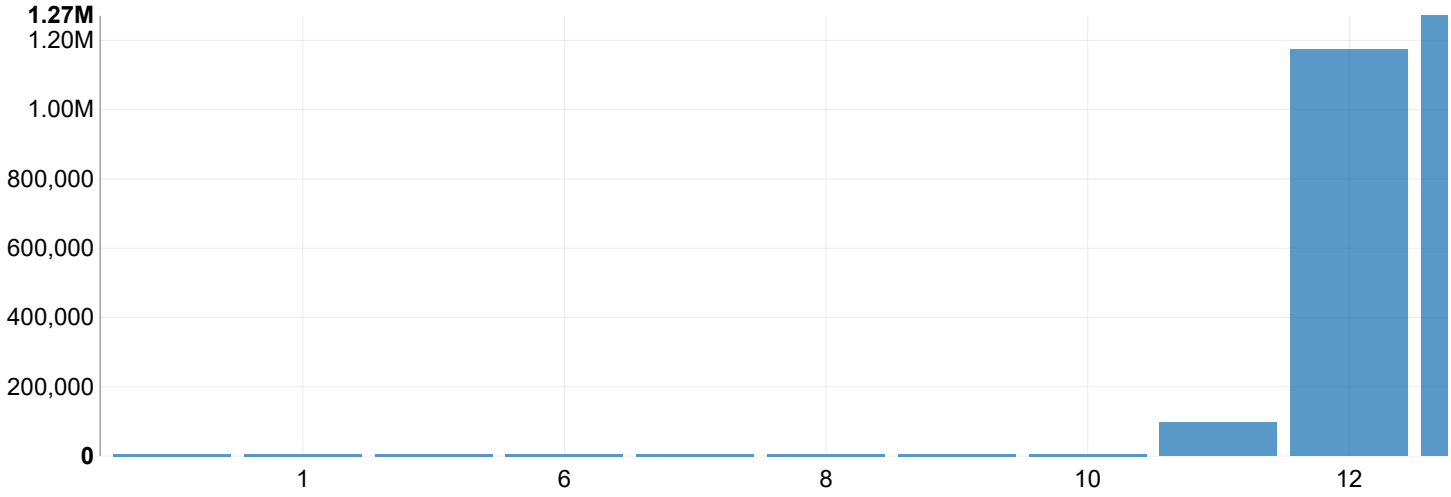
8. Beantwortung Frage 2: Umsatz pro Monatstag

- Wie entwickelt sich der Umsatz über die Monatstage für die Zeitspanne 2016-2017?



8. Beantwortung Frage 3: Umsatz im Tagesverlauf

- Wie entwickelt sich der Umsatz im Tagesverlauf?



8. Beantwortung Frage 4: Einfluss Wetter auf durchschnittlicher Tagesumsatz

- Es ist bekannt, dass das Restaurant auch einen Take-Away-Service anbietet. Welchen Einfluss hat das Wetter auf den durchschnittlichen Tagesumsatz?

Quellen:
https://spark.apache.org/docs/2.3.2/api/python/pyspark.sql.html#pyspark.sql.DataFrame.join (https://spark.apache.org/docs/2.3.2/api/python/pyspark.sql.html#pyspark.sql.DataFrame.join)

```
+----------+-----+-----+------------------+
|      date|wdayN|mdayN|          turnover|
+----------+-----+-----+------------------+
|2016-07-08|    5|    8| 9245.079963684082|
|2017-03-04|    6|    4|12987.409999847412|
|2017-03-27|    1|   27|10120.689978480339|
```

```
|2017-11-06|    1|    6| 10556.19997882843|
|2016-11-12|    6|   12|10744.950000286102|
+----------+-----+-----+------------------+
only showing top 5 rows
```

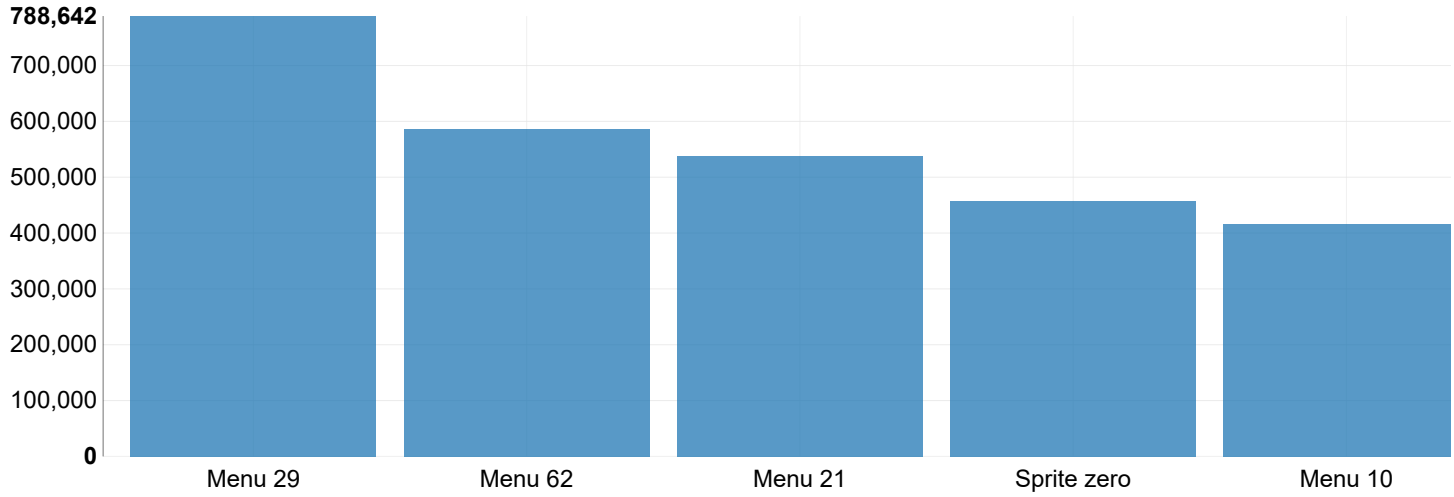
```
+----------+-----------+---+---+----+
|      date|date_number|sun|tmp|  ws|
+----------+-----------+---+---+----+
|2016-01-18|      42387|377| -3|1.96|
|2016-01-19|      42388|  0|  0|3.51|
|2016-01-20|      42389|  0|  2|3.63|
```

```
Der durchschnittliche Tagesumsatz an sonnigen Tagen (Tagen mit > 120 Sonnenminuten) beträgt:
+-----------------+
|    avg(turnover)|
+-----------------+
|9755.728833006468|
+-----------------+

Der durchschnittliche Tagesumsatz an nicht sonnigen Tagen (Tagen mit < 120 Sonnenminuten) beträgt:
+-----------------+
|    avg(turnover)|
+-----------------+
|9686.261330139701|
+-----------------+

Der durchschnittliche Tagesumsatz an warmen Tagen (Tagen mit Durchschnittstemperatur über 0 Grad Celsius) b
eträgt:
+-----------------+
|    avg(turnover)|
```

8. Beantwortung Zusatzfrage:

- Mit welchen Produktgruppen wird am meisten Umsatz erzielt?
  Hinweis: Datentabelle articlegroup list wurde manuell erstellt.

```
+-------------+------------------+
|article_group|          turnover|
+-------------+------------------+
|       shop_1|             83.75|
|  beverage_21| 220.6299991607666|
|   beverage_9|1399.2599983215332|
|           54| 47154.99033641815|
|           15| 3518.430005311966|
+-------------+------------------+
only showing top 5 rows

+-------------+-----------+------+-----+
|article_group|description|weight|liter|
+-------------+-----------+------+-----+
|            1|     Menu 1| 77.58| null|
|            2|     Menu 2| 22.66| null|
|            3|     Menu 3| 99.71| null|
|            4|     Menu 4| 24.13| null|
```

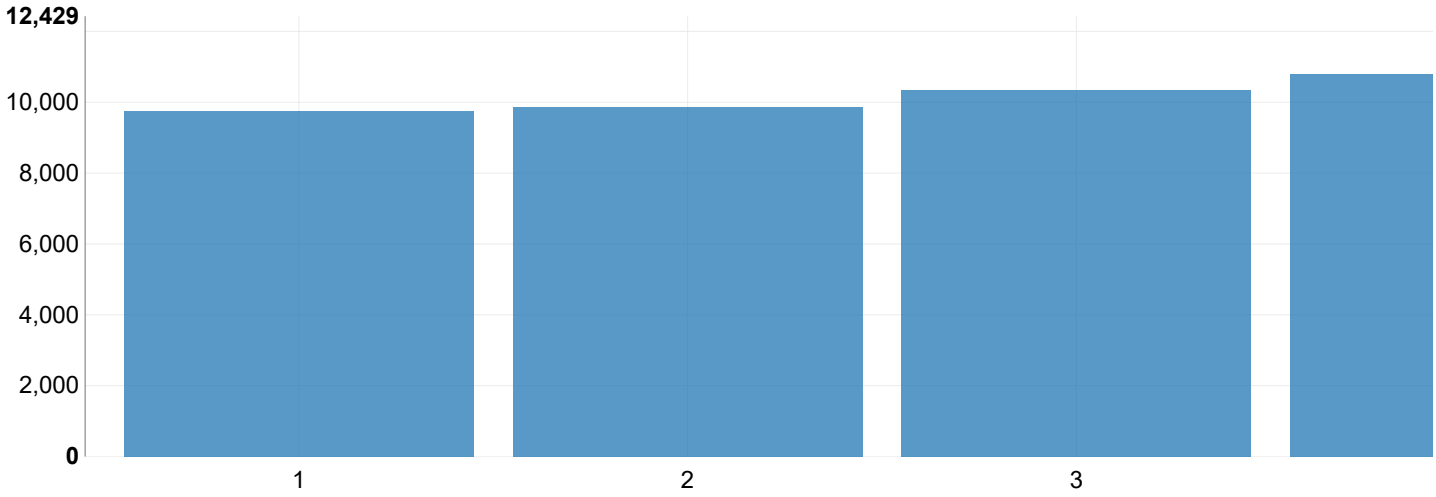


Ergebnisse

Mittels der gewählten Techniken konnten die untenstehenden Fragestellungen beantwortet werden.

Wie hoch ist der durchschnittliche Umsatz pro Wochentag für die Zeitspanne 2016-2017?
Die entwickelte Darstellung veranschaulicht, dass die durchschnittlichen Umsätze pro Wochentag schwanken. Der durchschnittliche Umsatz des umsatzstärksten Tages (Freitag, CHF 12’429) unterscheidet sich deutlich vom umsatzschwächsten Tag (Sonntag, CHF 5’228).

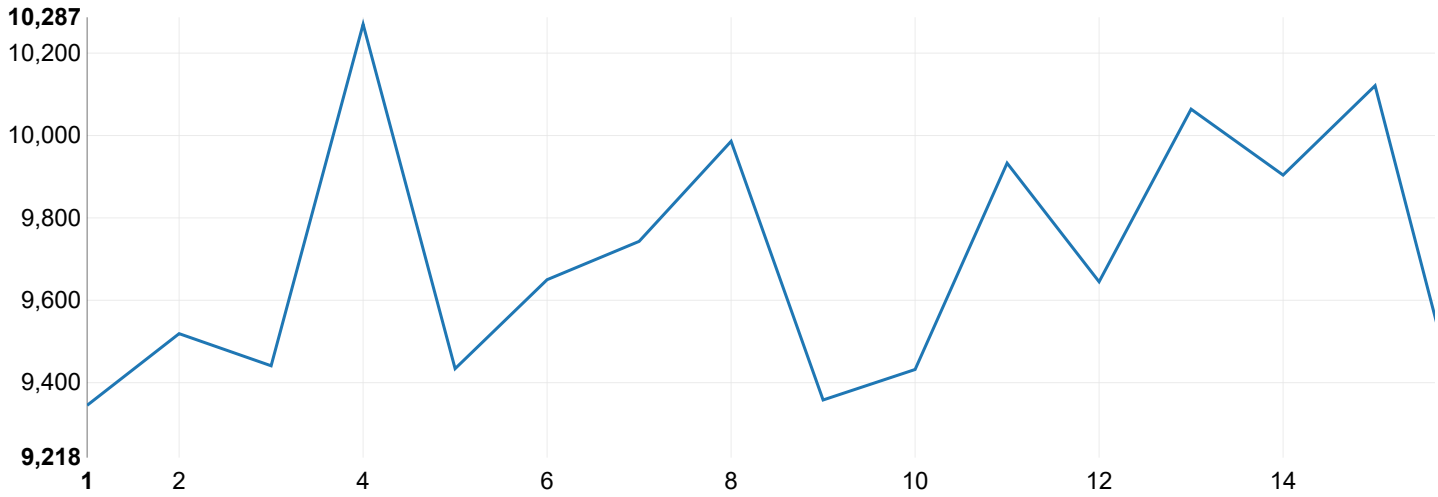
Basierend auf diesen Erkenntnissen kann man sich überlegen, ob die Öffnung des Restaurants an Sonntagen wirklich rentabel ist. Vor allem wenn man bedenkt, dass am Sonntag zusätzliche Mitarbeiterkosten aufgrund von Sonntagszulagen anfallen. Vielleicht wäre es eine Option, nur den Takeaway-Schalter zu öffnen, um Mitarbeiterkosten reduzieren zu können.



Wie entwickelt sich der Umsatz über die Monatstage für die Zeitspanne 2016-2017?

Die durchschnittlichen Umsätze pro Monatstag schwanken relativ stark. Die durchschnittlichen Umsätze liegen zwischen CHF 10’287 (Monatstag 24) und CHF 9’218 (Monatstag 21). Auch wenn auf den ersten

- Blick nicht direkt eine Erklärung für die Schwankungen ersichtlich ist, kann der Monatstag anscheinend eine Rolle für den zu erwartenden Umsatz spielen.

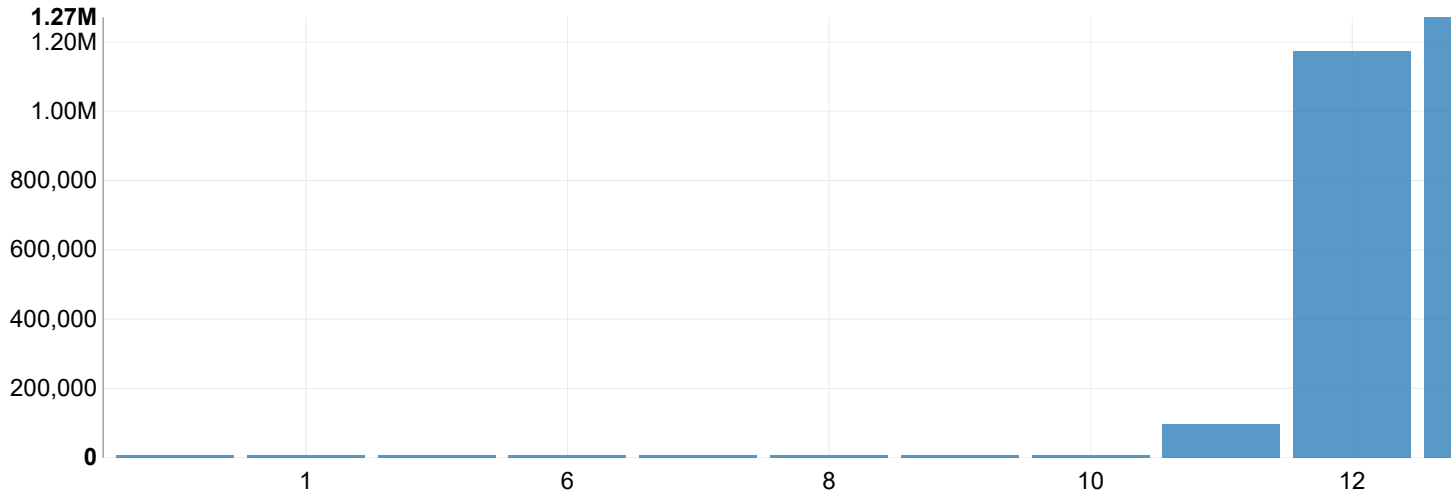


Wie entwickelt sich der Umsatz im Tagesverlauf?

Wenig überraschend wird der grösste Teil des Umsatzes um die Mittagszeit und am Abend erzielt. Aufgrund der Auswertung kann man sagen, dass eine Öffnung des Lokals am Morgen wenig sinnvoll ist und das Lokal möglichst nach 22 Uhr geschlossen werden sollte.

Basierend auf den Ergebnissen können folgende Überlegungen gemacht werden:

- Überprüfung der Öffnungszeiten
- Glättung der Peaks am Mittag mit zusätzlichem Personal (im Stundenlohn) oder Anreize für Kunden schaffen, neben Stosszeiten zu konsumieren (z.B. Spezialofferten für Menüs ab 13:00 Uhr).



Es ist bekannt, dass das Restaurant auch einen Take-Away-Service anbietet. Welchen Einfluss hat das Wetter auf den durchschnittlichen Tagesumsatz?

- Die Analysen haben gezeigt, dass die Umsätze an sonnigen Tagen (Tage > 120 Sonnenminuten) und Tagen mit Durchschnittstemperaturen grösser als 0 Grad Celsius rund CHF 100 höher sind als an nicht sonnigen Tagen bzw. kälteren Tagen. Der Faktor Wetter spielt also nicht eine so grosse Rolle wie erwartet.

```
Der durchschnittliche Tagesumsatz an sonnigen Tagen (Tagen mit > 120 Sonnenminuten) beträgt:
+-----------------+
|    avg(turnover)|
+-----------------+
|9755.728833006468|
+-----------------+

Der durchschnittliche Tagesumsatz an nicht sonnigen Tagen (Tagen mit < 120 Sonnenminuten) beträgt:
+-----------------+
|    avg(turnover)|
+-----------------+
|9686.261330139701|
+-----------------+

Der durchschnittliche Tagesumsatz an warmen Tagen (Tagen mit Durchschnittstemperatur über 0 Grad Celsius) b
eträgt:
+----------------+
|   avg(turnover)|
```

Fazit Ergebnisse

Mit der Beantwortung der Fragestellungen konnte gezeigt werden, dass es durchaus Strukturen für den erzielten Umsatz im Datensatz hat. So kann beispielsweise der Wochentag oder der Monatstag einen Einfluss auf den erzielten Umsatz haben. Basierend auf diesen Erkenntnissen wäre es als nächster Schritt interessant, ein Umsatz-Prognosemodell zu entwickeln (z.B. mit PySpark MLlib). Auf dieser Grundlage könnte dann das Restaurant die Ressourcen entsprechend einplanen und so einen Mehrwert generieren (Kosten sparen).

Erfahrungen

Die grössten Herausforderungen werden im Kapitel Projektablauf direkt erklärt. Nachfolgend sind die Lessons learned aufgeführt, welche vor allem aus diesen Herausforderungen entstanden sind:

- **Datenimport**: Obwohl der im Unterricht gezeigte Import der Daten ziemlich “straightforward” aussah, ist der Datenimportprozess (von verschiedenen Sourcen) keinen Falls zu unterschätzen.
- **Datenschema**: Die Definition des Datenschemas ist zwar programmiert-technisch schnell gemacht, muss aber genaustens durchdacht sein. Die Daten im richtigen Schema auf einem System verfügbar zu machen, braucht mehr Zeit und Konzentration als ursprünglich erwartet (siehe Beispiel Null-Tabelle oder Sortierungsprobleme).
- **Nachschlagen und recherchieren**: Das Recherchieren und Nachschlagen von Befehlen ist bei der Entwicklung eines solchen Projekts absolut zentral. Gute Quellen (Stack Overflow, Dokumentationen) zu kennen und verwenden zu können ist ein Schlüssel für den Programmiererfolg.

- **Datenabfragen**: Auch wenn die Programme fehlerfrei funktionieren ist das Plausibilisieren der Ergebnisse sehr wichtig. Es ist mir mehrmals passiert, dass zwar die Abfrage mit PySpark-SQL funktionierte, ich aber nicht das Resultat erhielt, welches ich suchte (z.B. aufgrund des falschen
- Aggregationslevels). Ein gutes Verständnis der Datenstruktur ist dafür Pflicht.

Screenshots

Screenshot 1: Kontrolle Daten auf lokalem FileSystem via GitBash

Kontrolle mittels Git-Bash, ob Files wirklich auf lokales Filesystem geladen wurden. Dank der Tatsache, dass ich auf dem "root" bin, kann ich die Dateien des Users Zeppelin sehen.

```
root@cl-hpelc1-50-gw-01-lx-ub18.lxd:/home/zeppelin/data# ll
total 34481
drwxr-xr-x 2 zeppelin hadoop        5 Feb 13 15:20 ./
drwxr-xr-x 9 zeppelin hadoop       15 Feb 13 15:15 ../
-rw-rw-r-- 1 zeppelin hadoop     1744 Feb 13 15:20 liste_articlegroup.csv
-rw-rw-r-- 1 zeppelin hadoop 35236143 Feb 13 15:20 trans_m.csv
-rw-rw-r-- 1 zeppelin hadoop    20607 Feb 13 15:20 weather_data.csv
root@cl-hpelc1-50-gw-01-lx-ub18.lxd:/home/zeppelin/data#
```

Screenshot 2: Probleme HDFS

Kontrolle, ob Daten vom lokalen Filesystem auf HDFS übertragen wurden. Problem unten:
Ordner "data" scheint leer zu sein. Die Files wurden jeweils nicht ersetzt. Der Befehl wurde wie folgt angepasst:

- Vorher: hdfs dfs -copyFromLocal $trans_m data/$trans_m
- Nachher: hdfs dfs -copyFromLocal **-f** $trans_m data/$trans_m

```
root@cl-hpelc1-50-gw-01-lx-ub18.lxd:/home/zeppelin/data# su -l zeppelin
$ hdfs dfs -ls
Found 8 items
drwxr-xr-x   - zeppelin hdfs          0 2020-02-13 13:03 .sparkStaging
-rw-r--r--   1 zeppelin hdfs      22118 2020-02-11 10:02 bahnhofbenutzer.csv
drwxr-xr-x   - zeppelin hdfs          0 2020-02-12 16:36 bank2
drwxr-xr-x   - zeppelin hdfs          0 2020-02-13 13:37 conf
drwxr-xr-x   - zeppelin hdfs          0 2020-02-13 14:43 data
-rw-r--r--   1 zeppelin hdfs    5777367 2020-02-12 16:10 examples
drwxr-xr-x   - zeppelin hdfs          0 2020-02-13 13:37 notebook
drwxr-xr-x   - zeppelin hdfs          0 2020-02-10 14:03 test
$
```

Die Files sind nun auch im HDFS vorhanden:

```
-rw-r--r--   1 zeppelin hdfs       1744 2020-02-13 16:50 data/liste_articlegroup.csv
-rw-r--r--   1 zeppelin hdfs   35236143 2020-02-13 16:50 data/trans_m.csv
-rw-r--r--   1 zeppelin hdfs      20607 2020-02-13 16:50 data/weather_data.csv
```

Screenshot 3: Null-Table

Nach der Schema-Mutation trat eine der drei Table nur noch mit null-Werte ein. Dies war damit zu begründen, dass die Spalten "weight" und "liter" im Datenset mit Dezimalstellen gespeichert sind (entspricht Typ double) und ich die Daten ins Format Integer überführen wollte. Nach Anpassung des

- Formats in FloatType wurden die Daten wieder korrekt angezeigt.

```
+-------------+-----------+------+-----+
|article_group|description|weight|liter|
+-------------+-----------+------+-----+
|         null|       null|  null| null|
|         null|       null|  null| null|
|         null|       null|  null| null|
|         null|       null|  null| null|
|         null|       null|  null| null|
+-------------+-----------+------+-----+
```

Screenshot 4: Table Schema

Sortierung des SQL-Queries funktionierte nicht (nach 1 kam nicht Tag 2 sondern Tag 10). Damit Sortierung funktioniert, muss Variable mdayN als integer und nicht wie bis anhin als string definiert werden.

Screenshot 5: Variablenbeschreibung trans_m

Datensatz: trans_m.csv

Field	Description
key	entry number as string
date_long	date of day as date in format dd.mm.yyyy hh:mm
transaction_id	transaction number as string
article	article number as string
article_group	article group number as string
price_item_list_brut	list price per item excluding discount and including MWST as float
count	number of articles per transaction as integer
turnover	turnover per entry in CHF inkl. MWST as float
price_pos_list_net	price per entry in CHF excl. MWST as float
hour	day hour as integer
wdayN	number of day of week 1 = Monday, ... 7 = Sunday as integer
date	date of day as date dd.mm.yyyy
date_number	date of day as string
mdayN	month day as integer

Screenshot 6: Variablenbeschreibung articlelist

Datensatz: liste_articlegroup.csv

Field	Description
article_group0	article number as string
description	description article group as string
weight	number of grams as float
liter	quantity, number of liters as float

Screenshot 7: Variablenbeschreibung weather_data

Datensatz: weather_data.csv

Field	Description
date	date of day as date in format dd.mm.yyyy
date_number	date of day as string
tmp	local temperature [°C] as integer
sun	local number of minutes of sun shine as integer
ws	local wind speed [m/s] as integer