Mario Matuschek, Carsten Nickel, Gökhan Sagir, Felix Stockhammer

**Vienna University of Economics and Business**

5408 – Data Processing 2

Dr. Sabrina Kirrane

**Project Report**

# Social media and financial markets

Assessing the impact of social networks utilization during “blacklivesmatter”- turmoils on price performance in the private security market


Project Report by **TeamMatuschek**:

**Mario Matuschek** (h11813240)

**Carsten Nickel** (h11825522)

**Gökhan Sagir** (h1552537)

**Felix Stockhammer** (h11801716)

**Date**

06.07.2020

# Contents

# List of figures

# Project Overview

## Introduction and high-level description

For our Project we decided to discuss a recently widespread topic. Due to the riots and the current chaotic situation in the USA, the importance of social media to carry a message around the globe is highlighted once more. In our project we aim for analyzing whether these unusual circumstances do also have an impact on selected titles traded on the stock exchange. For our analysis we will focus on companies in the "private security" segment and choose the ADT Inc. as representative and wish to answer the research question: "***Does the stock price of ADT Inc. correlate with Twitter sentiment?"***

Focusing on the 5 Vs of Big Data we may assure that our project also fulfills the necessary requirements. We will be streaming real-time twitter data (Velocity) to generate large amounts of data. We will be working with unstructured text data and structured numeric data (Variety) delivered in a JSON format, via the Twitter API using the "tweepy" library in Python. We made use of a NoSQL Database named MongoDB to store the unprocessed tweets as well as the financial data to not only ensure reproducibility to our project, but also scalability. In total, we streamed about 60 MB of raw data, which is represented in roughly 225.000 tweets and the financial data we collected over the timespan of the project. Although it is only 60MB of data, 225.000 tweets fulfill the Volume's requirement of a Big Data Project. Veracity is covered by collecting historical stock prices and real tweets.

As mentioned earlier our aim is to investigate correlations (Value) between the sociopolitical developments in the United States and the stock price of ADT Inc.. As the underlying fundamentals of stock movement are very complex and based on many different components, it is worth mentioning that correlation does not imply causation. To deduce the sociopolitical developments out of unstructured text data one of our main part of the project has been the Natural Language Processing (NLP).

NLP is needed to prepare the unstructured text data in such a way that we can employ several different sentiment analysis methods on it to analyze the sociopolitical development based on social media. These methods range from pre-built analyzers such as TextBlob to Vader Sentiment to machine learning models such as Naive Bayes and Random Forests. A $Chi^2$ Test will be carried out to compare the predictions of the best ML model to the build-in TextBlob sentiment classifier. Additionally, several visualization techniques will be taken to facilitate the comprehension of the data and the analytical steps taken.

Next, the tweets and their sentiment are grouped into distinct time intervals (15 min, 60min) to extract the prevalent average mood of the twitter society. We then combine the two variables of interest (Twitter sentiment and ADI Inc. stock price) and use the Pearson correlation coefficient and corresponding visualizations such as scatter plots to inspect whether the prevalent mood fostered by socio political developments has an impact on the returns/ the stock price of ADT. As the analytical focus has been on

the United States of America which is known to be a diverse country with distinct regional differences, we had a closer look into the regional differences by setting up a choropleth map displaying the average sentiment by state. Additionally, we were interested in digging deeper into the Machine Learning Models using spark's machine learning library MLlib. We constructed several different Sentiment classifier models and evaluate their performance in terms of accuracy.

# Data Sources

## Twitter API

*Further information and documentation can be found at "https://developer.twitter.com/en"*

One of the key elements of our entire project is the Twitter API. It serves as the foundation for retrieving tweets from all around the globe. Representative for the sociopolitical development we filtered for tweets with the keywords "blacklivesmatter", "racism" and "georgefloyd". As not all attributes of the tweets would have been relevantly processed within the scope of our project, we have initially defined a clear amount of aspects that we are interested in and assume to be of create importance. This aspect may potentially limit our perspective on the analysis and the resulting implications and biases will be discussed in a further section of the project report. The selected attributes are: "tweet_text", "tweet_created" (date), "tweet_loc" (location), "followers_count" (follower count), "lang" (language), "state" (US-State), "user_id" and "user_name".

As the cornerstone of our project is sentiment analysis, the "tweet_text" variable is at the core and therefore indispensable. As the used libraries for the sentiment analysis are primarily intended to be applied to the English language, we store the "lang" of the tweets to filter accordingly in further steps. It is also crucial to mention that the employed steps of natural language processing are best applicable to English and still to be enhanced otherwise. The "tweet_created" attribute was used to group tweets by time intervals being one of the major aspects covered in preparation for the analysis between the sentiment and the stock price development. It was initially planned to use the location "tweet_loc" to observe trends among different regions. However, we had to abandon this plan later, as the location attribute was unfortunately too messy to work with. While "tweet_loc" did not translate into satisfactory results, we have decided to add the "state" attribute at a later stage of the project which was used to visualize regional differences within the United States. At the beginning of the project we also had rough plans of centering the analysis around "credible" users and potentially differentiate between "famous" twitter users and "unknown" twitter users with regard to the impact on the stock price of ADT Inc. However, we eventually end up not using it, as the path of the project analysis shifted. Although "user_name" and "user_id" are not particularly relevant for us, we decided to save them as well to ensure the integrity and reproducibility of our analysis.

## Yahoo Finance API

*Further information and documentation can be found at "https://rapidapi.com/apidojo/api/yahoo-finance1/details"*

As we are analyzing the impact of socio-political development on financial markets, we also need a source of reliable financial data. After examining potential data sources, we decided to use the Yahoo Finance as our source of data. As the actual Yahoo Finance Api only provide data download via csv files, we needed a way to access the stock price data in a satisfactory way. On promising way to do so was to use a platform called ***rapidapi.com.*** After signing in we were granted access to authentication keys needed to query the desired data.

We used the possibility and extracted the prices for the desired time intervals (15 min & 1 hour) and stored them into a data frame. Even though we have retrieved neatly prepared data, further pre-processing (timestamp referencing different time zone, etc.) was needed. Finally, we stored the processed data into a MongoDB database.

Although the results using the intermediate step led to satisfactory results, we found the “yfinance” library which also provides a reliable workaround to the Yahoo Finance API’s way of getting the data via csv files. One of the advantages compared to the rapidapi platform is that “yfinance” does neither need any authentication keys nor any sign ups to query the data. As we just found the “yfinance” towards the end of our project and had no issues with the original approach using the rapidapi platform, we decided to continue with the initial approach.

# Architecture

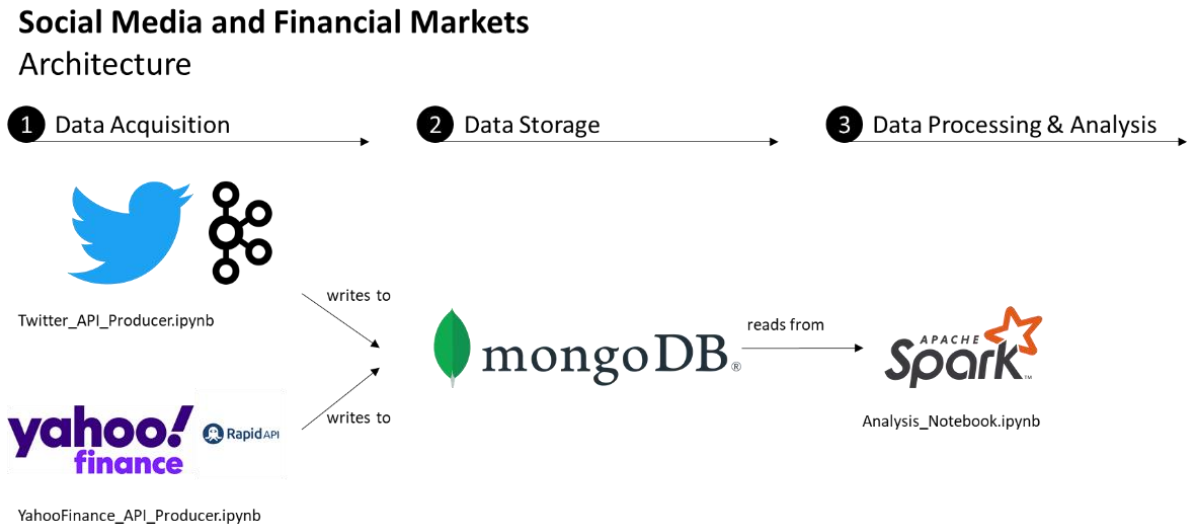


*Figure 1: Architecture of Project*

Like every good project, our Big Data project also requires a proper architecture. Our project is divided into four sections, data collection, data storage, data processing, and data analysis. We have used two different methods to collect the data. We used a Kafka stream to collect the Twitter data and then stored it into our MongoDB database. As for the financial data, other than the Twitter data, it is possible to query historical data via the API. Therefore, we decided not to use a stream but query the desired data. The queried data of the Yahoo Finance API is also stored in MongoDB. The second section deals with the data storage in MongoDB, as it is essential to use a database to ensure scalability of the analysis. After some failed alternatives (more about this in Challenges encountered) we decided to use a distributed cloud database named MongoDB, which turned out to fit perfectly for our needs. The two sections of Data Processing and Data Analysis were carried out with the help of Apache Spark. With its cluster computing, Spark is an excellent framework for Big Data Projects. The data is loaded into the consumer via MongoDB Spark Connector and preprocessed afterwards. Based on the preprocessed data, the data analysis is then performed in the consumer.

# Libraries

During the project we have used the following libraries and functions:

| *Libraries/ Packages* | *Functions/ methods* |
|---|---|

**Producers**

- Tweepy — StreamListener
- Pymongo — MongoClient
- Pyspark.sql — SparkSession
- Http.client — ->connection to RapidAPI for financial data

**Consumer**

- Pyspark
  - streaming — StreamingContext
  - sql — SparkSession
  - sql.types — *
- Pymongo — MongoClient

**Pre-Processing**

- Re
- Json
- string
- Textblob
- Pyspark
  - sql — window
  - sql.functions — udf, lower, col, trim, f
- Nltk
  - sentiment.vader — SentimentIntensityAnalyzer
  - stem.wordnet — WordNetLemmatizer
  - corpus — Stopwords

**Model Building**

- Pyspark
  - ml.feature — Word2Vec, StringIndexer, IDF, HashingTF
  - ml.classification — DecisionTreeClassifier, RandomForestClassifier, Naïve Bayes
  - ml — Pipeline
  - ml.tuning — ParamGridBuilder, CrossValidator

**Visualization**

- Matplotlib
- Seaborn
- Pandas

# Code & Algorithms

*Disclaimer: The overall notebook, forming the basis of this report is reliant on a range of libraries which may need to be installed prior to running the notebook and may run for several hours – "pip install" in any terminal should be ideal for Jupyter notebooks.*

## Data collection – Producers (Twitter & Yahoo)

As a foundation to our analysis we had to collect data from a range of sources as discussed earlier. Two separate notebooks have been used in the course of the project to collect either side of the data separately. Illustratively, we will walk through the most crucial parts of the twitter and Yahoo consumer to elaborate on certain steps.

```
# MongoDB atlas connection
client = pymongo.MongoClient("mongodb+srv://user_project:Project2020@cluster0-bcv7l.mongodb.net/test?retryWrites=true&w=majority
tweets_db = client.tweets
collections = tweets_db.tweets
```

*Figure 2: MongoDB connection*

In order to store the data into a MongoDB database allowing for scalable patch as well as stream processing a connection to the remote database is vital. Tweets / data will be stored in collections in a Json data format and may later on be retrieved as a spark RDD or spark data frame. A stream listener is essential for real-time tweet streaming. In order to fetch the data fitting our needs we wrote the following class method which is shown in a compressed form, neglecting the data dictionary for illustration purposes:

```
def extract_place(self,status):
#Before we start we need to define a dictionary with the states of the US.

    if status.place is not None:
        place = status.place
        if place.country != 'United States':
            return place.country
        elif place.place_type == 'admin':
            return place.name
        elif place.place_type == 'city':
            return states.get(place.full_name.split(', ')[-1])
    #If the status dict has no place info, get the place from the user data
    else:
        place = status.user.location
    try:
        place = place.split(', ')[-1].upper()
    except AttributeError:
        return None
    if place in states:
        return states[place]
    else:
        return None
```

*Figure 3: State filtering*

The above function checks if a tweet is marked with a location and filters all US-based tweets. It is essential to mention that this function has only been implemented towards the end, when our goal of a heatmap became apparent. The exception handling "AttributeError" should avoid any issues with the nested nature of the document structure.

A further crucial aspect that requires proper error handling is the tweet filter in the Listener instance created based on the desired keywords. This error may at all occur because no proper primary or composite key has been defined. We have tried different

approaches, but unfortunately, issues remained and we have decided to solve it otherwise, as it can be seen below.

```
if __name__ == '__main__':

# Error handling, avoid any DuplicateError
while True:
    try:
        myStreamListener = MyStreamListener()
        myStream = tweepy.Stream(auth=api.auth, listener=myStreamListener)
        myStream.filter(track=['blacklivesmatter, racism, georgefloyd'])
    except pymongo.errors.DuplicateKeyError:
        # skip document because it already exists in new collection ("id" is duplicate)
        continue
    #print ("A duplicate tweet id has been recognized, please start again.") --> this print may be used to show it was working.
```

*Figure 4: Stream Listener*

The data collection using the Yahoo Finance API required the use of a http.client setting up the connection. After authentication, a request is sent to the connection in order to provide the data desired. Within this request one can also determine an individualized time frame of real-time or even historic data. Decoding was the final essential step to allow proper processing of the data.

```
# Using yahoo finance api from https://rapidapi.com/apidojo/api/yahoo-finance1?endpoint=5c3d9f98e4b0a62b04251f0e
conn = http.client.HTTPSConnection("apidojo-yahoo-finance-v1.p.rapidapi.com")

# We registered at rapidapi.com and used our key to connect to the api
headers = {
    'x-rapidapi-host': "apidojo-yahoo-finance-v1.p.rapidapi.com",
    'x-rapidapi-key': "b82b3635dfmsh818e1b7b44ba9bdp13613cjsn028f72f0c6e9"
    }

# This statement sends the request
# Note: you can change the interval and range
conn.request("GET", "/market/get-spark?interval=15m&range=1mo&symbols=ADT", headers=headers)

# The response is then retrieved, decoded and saved
res = conn.getresponse()
data = res.read()
adt = json.loads(data.decode("utf-8"))
```

*Figure 5: Yahoo Finance Connection*

To ensure a proper basis of comparing the sentiment with the stock price, we have decided to calculate the corresponding returns over the selected timeframes, which will be grouped in further steps, just as the overall sentiment (for more information refer to the notebook). Below one can see our straightforward chosen approach. Please note, that these operations are performed using pandas data frames due to the fact that the Yahoo Finance API does not provide any insights and documentation on executions with Apache Spark. We created a dichotomous variable returning 1 for a positive return and 0 otherwise, which serve as the foundation to the overall trend developments of the stock data. We have also visualized the stock price development in the given notebook, more details to follow.

```
# Calculate differences to price above
df["diff"] = df["price"].diff()

# Set binary variable "positive difference" to 1 if the difference is positive
df["pos_diff"] = 0
df["pos_diff"].loc[df["diff"]>0] = 1 #ignore the error

# Reset the index
adt_15_min = df.reset_index(drop = True)
```

*Figure 6: Return calculations*

## Sentiment Analysis (TextBlob & Vader Sentiment) – Consumer

With the aim of using Spark's functionality of stream and batch processing, it is necessary to open a connection between the Spark session and the database server in a so called consumer, details and corresponding steps may be regarded in the notebook "Analysis_notebook", which offers a profound guideline. The notebooks contain a detailed elaboration of the pre-processing steps covered and will not be our recent topic of discussion. In contrast, we would like to get a detailed insight into the sentiment analysis and the used tools.

Performing sentiment analysis is a very heavily debated topic in the data science community. Thereby, there is a wide range of alternatives to apply in diverse projects. We have used 2 pre-built sentiment classifiers, namely TextBlob and Vader Sentiment – both built on top of the Natural language processing library for python called NLTK.

```
# define a sentiment function to apply to the "tweet_text"
def apply_sentiment(sentence):
    temp = TextBlob(sentence).sentiment[0] # only get the polarity
    if temp == 0.0:
        return 0
    elif temp > 0.0:
        return 1
    else:
        return -1

sentiment = udf(apply_sentiment) # use the udf method in order to apply the function to the column of interest in the tweet df.
```

*Figure 8: TextBlob Sentiment*

```
# Import a necessary library in order to also apply built in functions that may be overwritten due to the import of pyspark
# functions.
import operator

# Instantiate the object of the class
analyser = SentimentIntensityAnalyzer()

# Create a function in order to apply it to the tweets in the df
def Vader_Sentiment(sentence):

    # Get the dictionary of polarity scores (positive, negative, neutral & compound - each with a given probability, in total 1)
    sentiment = analyser.polarity_scores(sentence)

    # Get the the label with the highest likelihood (maximum probability)
    return (__builtin__.max(sentiment, key = sentiment.get))

sentiment = udf(Vader_Sentiment)
```

*Figure 7: Vader Sentiment*

In order to apply them to our spark data frame we have designed two user defined functions (udf) that may be applied to specific columns/ features of the dataset. While a basic branching structure is enough for the "apply_sentiment" function based on TextBlob, the set-up of the Vader Sentiment classifier required more functionality in terms of methods to be imported. Whereas function #1 distinguished between "positive", "negative" or "neutral", "Vader_Sentiment" additionally suggest "compound" as an alternative.

Both methods are lexicon-based approaches. This means that they refer to a pre-built already classified collection of words and combine a tokenized sentence' likelihood, based on the individual given probabilities assigned to each word. The most probable sentiment will be returned in the end. It is worth mentioning that the classifiers have been applied to entire sentences, without any tokenization or lemmatization in advance.

Regarding the suggested data distribution (sentiment distribution) of both approaches we have decided to choose the TextBlob sentiment analyzer as our "true" label of the

tweets, given a more realistic data split as it can be seen in the bar plot below. This data split will also play a crucial role in the end, when determining if a machine learning model performed decent or not. Concerning the data labelling, a detailed discussion on biases will follow later in the report.

## Sentiment Analysis (NLP & Machine Learning) – Consumer

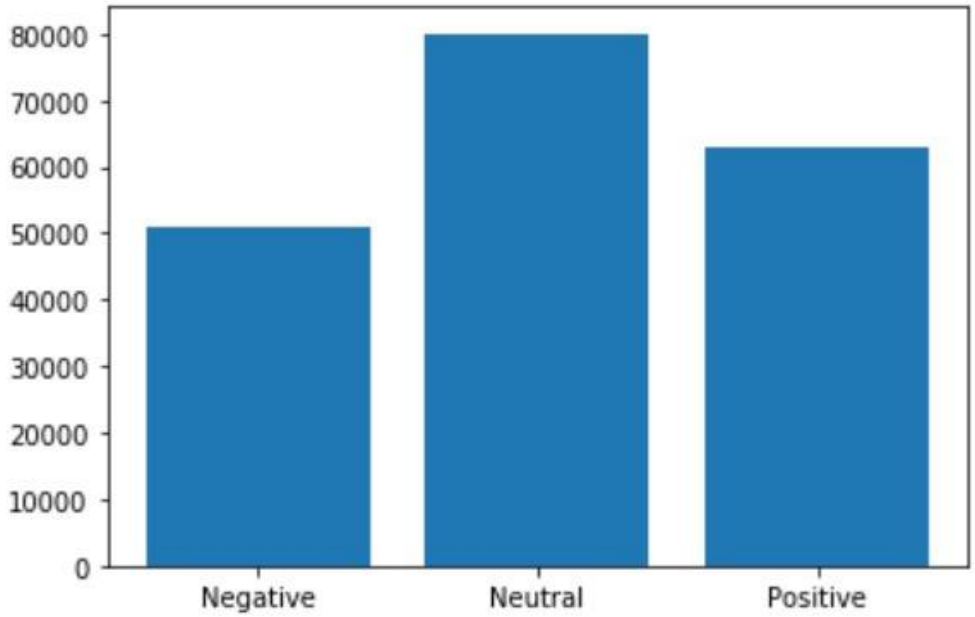


*Figure 9: Bar Plot Sentiment Distribution*

One of our aims of this project was to empirically investigate different machine learning algorithms' performance when classifying sentiment of tweets. We have used: Naïve Bayes, Decision trees and Random Forests. Before setting up the pipeline to those models and training them on our dataset we have made use of the broad spectrum of natural language processing tokenizing and lemmatizing the data. Additionally, also stop words have been removed. All these steps have been covered using user-defined-functions fitting our needs or are based on the NLTK library.

### ***Step One:*** Tokenization

```
# use NLTK to tokenize the tweets into words
word_tokenizer = udf(nltk.word_tokenize)
df_tokenized = df_textblob.withColumn("tokens", word_tokenizer("tweet_text"))
```

*Figure 10: NLTK Tokenizer*

### ***Step Two:*** Stop word removal

```
# customize the stopwords list provided by the nltk corpus
stops = set(stopwords.words("english"))

# supplement the set of stopwords
stops.add("’")
stops.add("https")
stops.add(" ")
stops.add("\n")
stops.add("“")
stops.add(";")
stops.add("&")
stops.add("u")
stops.add("”")
stops.add("…")
stops.add("’’")

def remove_stops(tweet_list):
    cleaned_list = [] # initalize an empty list later filled with words that are not stopwords
    for word in tweet_list:
        if word not in stops:
            cleaned_list.append(word)
        else:
            pass
    return cleaned_list

remove_stops = udf(remove_stops)

df_stops = df_tokenized.withColumn("tokens", remove_stops("tokens"))
```

*Figure 11: Stop Word function*

### ***Step 3:*** Lemmatization

```
def lemmatizationFunct(tweet):
    lemmatizer = WordNetLemmatizer()
    list_lemmatized = [lemmatizer.lemmatize(word) for word in tweet]
    return list_lemmatized
lemmatization = udf(lemmatizationFunct)
df_lem = df_stops.withColumn("tokens", lemmatization("tokens"))
```

*Figure 12: Lemmatization function*

After all those steps we have been ready to dive into supervised machine learning and a common classification problem. Having selected the necessary features of the dataset we had to vectorize our tokens. Vectorization is necessary to transform natural language into numeric data, being a prerequisite for learning a ML-model. In the realm of our project we have consistently made us of the extensive Spark Machine Learning functionalities and used the Word2Vec model for the numerical transformation of the word tokens.

```
# Vectorize the tokens to feed them to the ML models
from pyspark.ml.feature import Word2Vec

# Initliaze the object
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="Array_tokens", outputCol="vector")

# Fit on the data (df_nlp)
model = word2Vec.fit(df_nlp)
model_df = model.transform(df_nlp)
```

*Figure 13: Word2Vec - Vectorization*

The word vectorizer is a so-called Bag-of-Words model. It lists all the words in the data and represents the words based on the relative appearances in the dataset in a separate feature space. This feature space, in which each token has its own relative position is the foundation for the numerical vectors being assign to a tweet. Prior to the pipeline constructions of the model we split the data into a training and testing set. In order to ensure reproducibility, we also set a seed in the split.

## Decision Tree

```
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.feature import StringIndexer, VectorIndexer
import pyspark.sql.types as T
from pyspark.sql.types import Row
from pyspark.ml import Pipeline

# Pipeline construction ---------------------------

# Index the label and fit on data
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(train_set)

# Take the token vector and create a feature vector
featureIndexer = VectorIndexer(inputCol="vector", outputCol="indexedFeatures", maxCategories=10).fit(train_set)

# Initalize the Decision tree classifier
dt = DecisionTreeClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures")

# Construct the pipeline
pipeline_dt = Pipeline(stages=[labelIndexer, featureIndexer, dt])
```

*Figure 14: Decision Tree pipeline*

The first model we have used in our project is a decision tree. When constructing the pipeline we focused on three steps. Firstly, a StringIndex object is created and stored in a variable. Secondly, a VectorIndexer object, setting the number of categories of different words is initialized. It must be mentioned that allowing for only 10 categories of words / feature vectors is a sever limitation to our project but had to be chosen due to the fact that the runtime should be kept to a minimum. Lastly, we surely had to construct the classifier itself before fitting all stages into the pipeline set-up.

Followingly, we trained the algorithm and fitted the model on previously unseen data.

```
# Train the decision tree algorithm and create a model
model_dt = pipeline_dt.fit(train_set)

# Test the model on unseen data
prediction_dt = model_dt.transform(test_set)
```

*Figure 15: Model training and prediction*

As an evaluation metric we have chosen accuracy, as we wanted to minimize the amount of both, false positives and false negative.

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Create the evaluator for multiclass classification
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")

# Evaluates model output - accuracy will be the used metric with the aim of minimizing the false positive & false negatives
evaluator.evaluate(prediction_dt, {evaluator.metricName: "accuracy"})
```

*Figure 16: Model evaluation*

A very unsatisfactory result of only 41,79% of correctly labeled data led us to the decision to employ the other, previously stated models. Next to come is the Random Forrest Classifier.

## Random Forest

As with the previously discussed model, we also had to construct the machine learning pipeline additionally consisting of a labelConverter, that was to ensure the correct data type of created columns and will be used in the pipeline initialization. One key argument, that was inherent to the random forest classifier is the number of trees that should be scanned when training and testing on data. We have chosen the lower end a typical scale reaching from 10 – 100 to avoid crushing the container in which the notebook was hosted.

```
# Import all the necessary packages (also a label to string converter)
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.feature import IndexToString

# lableIndexer and featureIndexer have already been created in the pipeline to the decision tree only supplement now
rf = RandomForestClassifier(labelCol="indexedLabel", featuresCol="indexedFeatures", numTrees=10)

# Labels are converted back to string type
labelConverter = IndexToString(inputCol="prediction", outputCol="predictedLabel",
                               labels=labelIndexer.labels)

# Feed all the pre-processing stages into the pipeline
pipeline_rf = Pipeline(stages=[labelIndexer, featureIndexer, rf, labelConverter])
```

*Figure 17: Random Forest pipeline*

Fitting the data onto the model and calculating the evaluation metric is identical to the previous scenario. However, more important is the fact the model performed worse overall. Indeed, as a little spoiler, it was the worst of all with an accuracy of only 40,87%.

## Naïve Bayes

The best classification algorithm that we used in the course of this project is the Naïve Bayes classifier. The set-up of the analytical approach is identical to the previously seen methods and includes: Data splitting, pipeline construction, model fitting, prediction generation and model evaluation. To our frankness, we ended up with an accuracy of 77,75% being enormously better than a simple 1 category choice that would result in a 30 – 40% accuracy as it could be seen in the bar chart showing the data distribution.

```
# Rename the columns
model_df = model_df.withColumnRenamed("label", "Sentiment")

# split the data again into the 2 sets (randomly in order to ensure reproducability) in order to also have the renaming included
(train_set, test_set) = model_df.randomSplit([0.80, 0.20], seed = 1234)
```

```
from pyspark.ml.feature import IDF, HashingTF
from pyspark.ml.classification import NaiveBayes

# Configuring a ML pipeline for a Naive Bayes consists of: Tokenization, HashingTF & the model (nb)
# Tokenization as part of the ML pipline has already been covered previously and will not be repeated

# Convert textual data into features (vectorizing)
hashingTF = HashingTF(inputCol="Array_tokens", outputCol="features")

# Get the inverse document frequency in the tweets
idf = IDF(minDocFreq=3, inputCol="features", outputCol="idf")

# Set the lables used for the predictions (the most frequent class gets a 0, descending order)
label_stringIdx = StringIndexer(inputCol = "Sentiment", outputCol = "label")

# Create a Nb instance
nb = NaiveBayes()

# Pass all covered steps into the ML pipeline
pipeline_nb = Pipeline(stages=[hashingTF, idf, label_stringIdx, nb])
```

```
# Fit the pipeline on the training data to create a model
model_nb = pipeline_nb.fit(train_set)

# Apply the model to testing data to see how well the algo works in a later step
prediction_nb = model_nb.transform(test_set)
```

```
# Have a look into excerpts from tweets, the label and the predicition from the Naive bayes model.
prediction_df_nb = prediction_nb.select("tweet_text", "label", "prediction")
prediction_df_nb.printSchema()
# Show the top rows
#prediction_df_nb.show(5)
```

```
root
 |-- tweet_text: string (nullable = true)
 |-- label: double (nullable = false)
 |-- prediction: double (nullable = false)
```

```
# Evaluator for multiclass classification, which expects two input columns: prediction and label.
evaluator = MulticlassClassificationEvaluator(predictionCol="prediction")

# Evaluates model output - accuracy will be the used metric with the aim of minimizing the false positive & false negatives
evaluator.evaluate(prediction_df_nb, {evaluator.metricName: "accuracy"})
```

*Figure 18: End-End set-up ML model (Naive Bayes)*

Despite an overall satisfactory result of the model's performance we decided to run a Chi² test, used for two categorical variables, comparing the models to the "actual" true labels provided by TextBlob sentiment analyzer.

```
# X²-Test
crosstab = prediction_df_nb.crosstab("label", "prediction")
# Import necessary libraries
from itertools import chain
from pyspark.mllib.stat import Statistics
from pyspark.mllib.linalg import DenseMatrix
# Run the X²-Test
chi2_results = Statistics.chiSqTest(DenseMatrix(
    numRows=crosstab.count(), numCols=len(crosstab.columns) - 1,
    values=list(chain(*zip(*crosstab.drop(crosstab.columns[0]).collect())))
))
# Show the data in a crosstable
crosstab.show()
# Print the p-value
print(chi2_results.pValue)
# Print the test statistic
print(chi2_results.statistic)
```

```
+----------------+-----+-----+----+
|label_prediction|  0.0|  1.0| 2.0|
+----------------+-----+-----+----+
|             2.0|  936| 1897|7166|
|             1.0| 1013|10393|1164|
|             0.0|12123| 2202|1283|
+----------------+-----+-----+----+

0.0
33474.16615288508
```

*Figure 19: Chi² test*

Interpreting the output of the statistical test we can confidently confirm that there is a statistically significant difference between the actual and predicted labels. The p-value is 0 and therefore less than any alpha level we could choose (most common is the 5% alpha level). Also, the test statistic 33474 is at unusual high levels. This shows that even with a high degree of accuracy, (predicted) sentiment may very well be extremely different from the actual state of affairs.

## Problems with the sentiment classification

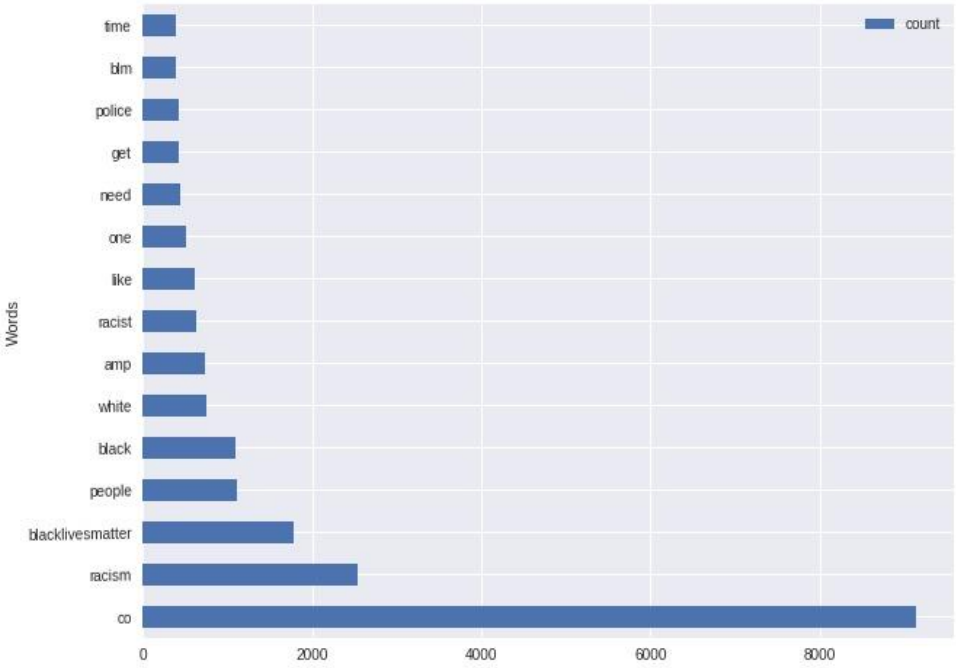


*Figure 20: Most common words (positive sentiment)*

The last section of this chapter should highlight one major issue that may be apparent when classifying the tweets. Looking at the graph below we can see the most used words in the tweets. The x-axis shows the amount of total appearances and the y-axis represents the words in ascending order, the most frequent at the bottom.

Interestingly, the graph looks very similar, be it with regard to positively or negatively classified tweets. A potential reason may be the restrictive set-up we used in terms of

filtering constraint when streaming the data. This may be improved by an amplification of the tweet spectrum. While consistent appearances of words such as “blacklivesmatter” go along with our set-up, for example “co” is a tremendous outlier. Improvements in the data cleansing steps may have potentially alternated the results of this graphical representation. The ratio behind this visualization was to detect the drivers of the sentiment classified in the ML models, but as both positive and negative show common results, adaptions would be required and no proper scientific claims can be stated.

## Data Analysis – Twitter & Finance

As it has already been mentioned several times before we had to group sentiment and the financial data into time intervals such that we are able to compare them and detect whether they correlate or not. Below we can see a code snippet of the steps taken in order to achieve our prospect.

```
# Import necessary libraries to convert the textual sentiment into numeric data
from pyspark.sql.types import FloatType

# Cast the column values
df_textblob = df_textblob.withColumn("Sentiment", df_textblob["Sentiment"].cast(FloatType()))

# Get the sum of the sentiment scores for 15 min intervals
df_15min = df_textblob.groupBy(window("tweet_created", "15 minutes")).sum("Sentiment").alias("Sentiment")

# Now we have a window object with two time stamps, only the first is needed for further grouping.
df_15min.printSchema()
```

```
root
 |-- window: struct (nullable = false)
 |    |-- start: timestamp (nullable = true)
 |    |-- end: timestamp (nullable = true)
 |-- sum(Sentiment): double (nullable = true)
```

```
# Import libraries to re-arrange the data formats in the necessary way
from pyspark.sql.types import StringType, DateType

# convert window object to string in order to split apart
df_15min = df_15min.withColumn("time", df_15min["window"].cast(StringType()))

# This line should only be run once - if the field is missing an error will appear
df_15min = df_15min.drop("window")

#df_15min.printSchema() #--> run if interested

# Create a "time" column as a substring of the long string "window" that we only want the second half of.
df_15min = df_15min.withColumn("time", expr("substring(time, 2, length(Time))"))

# Convert "time" to a timestamp object
df_15min = df_15min.select(to_timestamp(df_15min.time, 'yyyy-MM-dd HH:mm:ss').alias('time'), "sum(Sentiment)")

df_15min.show(truncate = False)
```

```
+-------------------+--------------+
|time               |sum(Sentiment)|
+-------------------+--------------+
|2020-06-26 15:45:00|120.0         |
|2020-06-23 17:45:00|50.0          |
|2020-06-30 18:45:00|213.0         |
|2020-06-22 19:15:00|80.0          |
|2020-06-23 08:45:00|30.0          |
|2020-06-23 14:15:00|50.0          |
|2020-07-02 18:00:00|145.0         |
|2020-06-23 13:45:00|19.0          |
|2020-06-23 10:30:00|11.0          |
|2020-06-26 19:00:00|197.0         |
|2020-06-26 18:30:00|28.0          |
```

*Figure 21: Interval setting*

We converted the sentiment into floating point numbers, as even the numbers have previously still been in a string format resulting from the transformations covered in the ML section of the project. Also, we created a “window” column showing the beginning and end time of the desired interval time frame. Followingly, we converted the window structure back to a string in order to split the time and only keep the end time of the interval. We have chosen the given intervals due to their natural usage in the financial

markets. We applied these steps for both time intervals and also to the TextBlob classified sentiment on the one hand and the best ML model, the Naïve Bayes, on the other hand to see whether correlations may differ between different sentiment bases.

```
# Import necessary libraries
import numpy as np

# Merge the datasets and remove the entry with no growth rate
df_merge = df_15min.join(df_15min_financial, on = "time", how = "rightouter")
df_merge = df_merge.filter((df_merge["sum(Sentiment)"] != 0) & (df_merge["diff"] != np.nan))

#df_merge.show() --> uncomment if requested
```

```
# Filter the df for relevant variables
df_merge = df_merge.select("sum(Sentiment)", "diff", "pos_diff", "price")

# Rename the column
df_merge = df_merge.withColumnRenamed("sum(Sentiment)", "Sentiment")
```

*Figure 22: Data Merge*

Merging the datasets (financial & tweets) after the proper interval setting was the final step to perform such that our research question:

***Does the stock price of ADT Inc. correlate with Twitter sentiment?***

May be answered very clearly. On the one hand side, we used the Bravais-Pearson correlation coefficient as a matter of determination. On the other hand, a scatter plot should serve as a visual confirmation of the numeric data.

| | Sentiment | diff |
|---|---|---|
| Sentiment | 1.000000 | -0.139694 |
| diff | -0.139694 | 1.000000 |

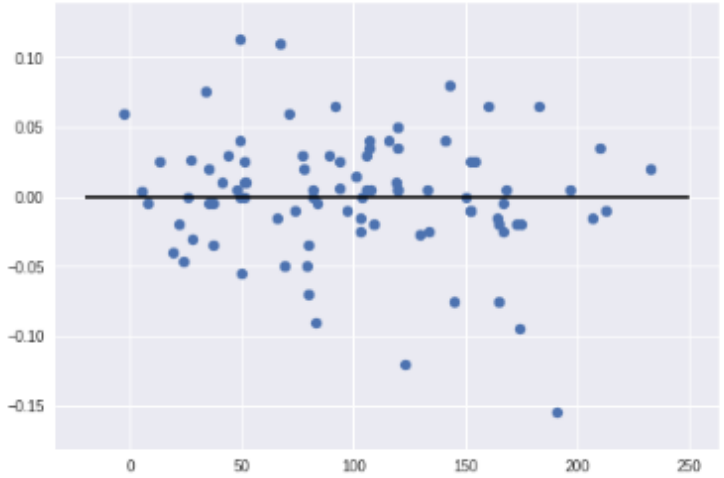


*Figure 23: Correlation Analysis*

Looking at the outcome of the analysis we can see that neither the numeric nor the visuals allow for the identification of any similarities. In fact, we can claim a ***very week negative linear relationship*** between the twitter sentiment and the returns of ADT Inc. in a time interval of 15 min. From our point of view this seems to be sensical. With a more positive mood among the population, the need for security “decreases”, therefore the stock price depreciates due to underutilization. Despite the general tendency to centrality, mean reversion around a return of 0%, there are also some outliers to both sides of the graphic, which strengthen the negative correlation ratio even further. While the sum of the interval sentiment is very large (roughly 200) the return is at -0.15%. Contrastingly, with a low sentiment of roughly 50, the return is positive at about 0.10%. Similar tendencies could be retrieved looking at the 1-hour time frame or also

the usage of the Naïve Bayes sentiment classification with minor non-significant fluctuations to the up- and downside – the relationship remains very weak. In order to get a more detailed insight into both parts of the analysis we also visualized their daily over-time developments in separate graphs, similarities may be detected thereafter, or potentially a more profound reasoning may be possible.

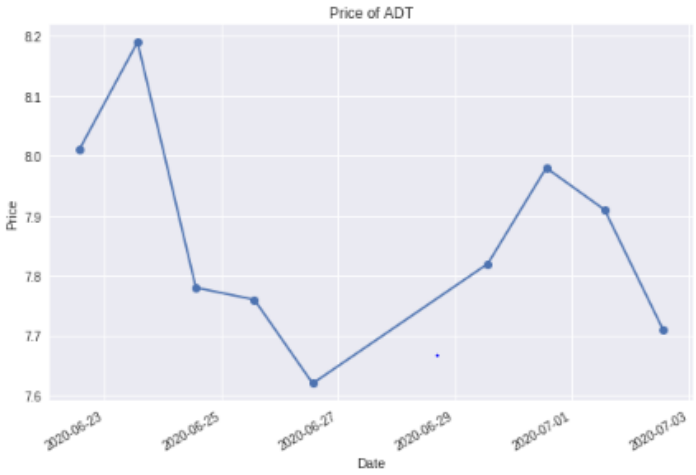


*Figure 25: Stock price ADT Inc.*

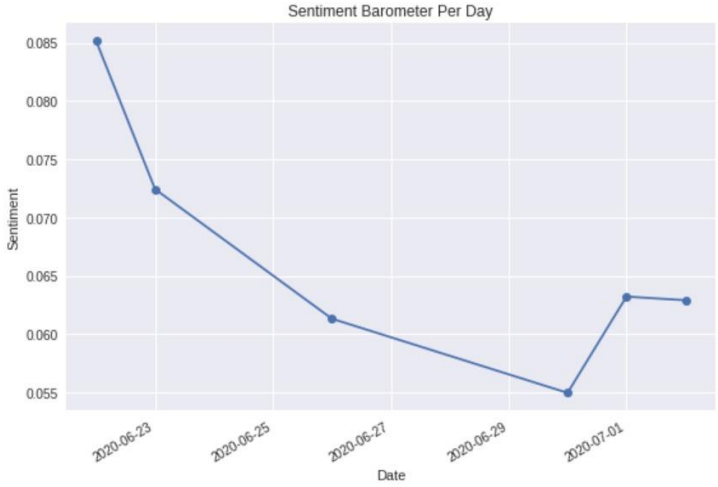


*Figure 24: Sentiment Barometer (daily)*

Before elaborating on the graphics above it is crucial to explain the labeled axis. While the chart on the left-hand side is a common financial graphics with price on the y-axis and time on the x-axis, the sentiment barometer portrays the average daily sentiment on the y-axis. A decrease in the right chart means a less positive, hence a more negative sentiment.

While we could see previously, an increase in positivity in social media leads to a decrease in the returns of ADT Inc. now we are confronted with the opposite scenario – a more negative environment causes the prices to tumble. The previous negative relation is now inverted and became weakly positive. A further aspect that must be mentioned is that the throughs and spikes in the sentiment tend to either lag or precede the stock price, making any potential predictions also very insecure and difficult. This uncertainty and the missing clarity about their relationship led us to the decision to discard and future price predictions solely on twitter sentiment.

As we are only “regressing” stock prices/ returns on twitter sentiment when running a linear regression model we would suffer large omitted variable bias due to the vast range of potential implicative factors on the Y variable (price/ return) as well as the sentiment. To potentially elicit one of the missing variables we had a closer look into regionality of sentiment.

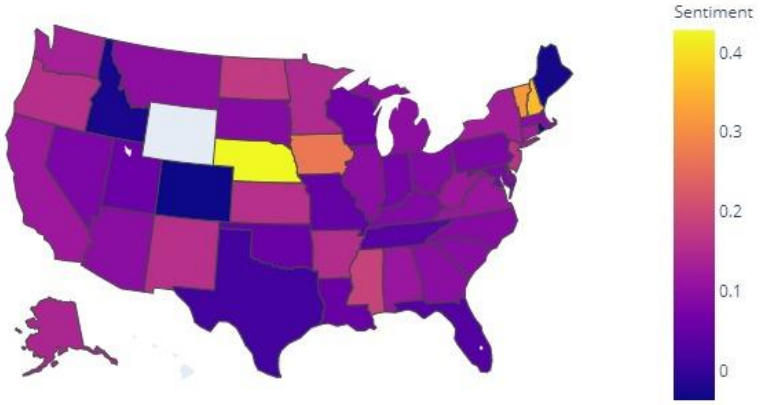


*Figure 26: Choropleth map United States of America*

Frankly, we have could see regional differences in the sentiment distribution across the United Stated of America. While especially the states around Minnesota, where the George Floyd incident happened, experienced wide-ranging sentiment differences. We can see a slight differentiation between the east and the west coast as well as the south, the north and the center of the US. Differences in purchasing power which are inherent to different regions may impact the likelihood of purchasing stock and may therefore differently impact the price development of ADT Inc.. Surely these are just assumptions which would require further testing but already hint at potential areas of future work to elaborate more on this topic.

## Conclusion

To conclude, we see that it is not possible to find a clear pattern of similarity between twitter sentiment and ADT Inc. stock price/ returns. In fact, different time frames yield different results and make it difficult to construct one comprehensive statement about their relationship.

However, we can see that there may be many more other reasons eventually playing a major role in sentiment distribution, hence also the impacts on the returns/ stock prices. Be it solely different classification algorithms, or also deeply rooted attributes such as regional differences causing different financial health, etc.

# Legal and Ethical Issues – guidelines and challenges

Working with the Twitter API raises the question how ethical it is to use Tweets that can often be ascribed to people that might not be aware that their usage of the platform enables third parties (such as us) to have access to their opinions. For this we must differentiate between the legal and ethical aspects. On the legal side, Twitter users need to accept Twitters Terms of Service, which states that “Twitter does not disclose personally-identifying information to third parties except in accordance with our Privacy Policy.” (Twitter 2020a). So, users are made aware that even their personally-identifying information could be used by third parties, but only if these parties handle the data with care.

To ensure this, before using the Twitter API, it is necessary to apply for a developer account which includes providing information about how the API will be used. Moreover, there is a separate Developer Policy and a Developer Agreement which Developers must accept. These include regulations on sensitive geo-data (which we do not need for our project apart from the US state the user comes from) and rate-limits. (see. Twitter 2020b,c) We followed these legal guidelines to the best of our knowledge. Moreover, there are additional guidelines for dealing with private messages or using Twitter content for advertising purposes. These do not apply to our project, as we will not publish any particular tweets and we are not dealing with private messages.

As the official Yahoo Finance API is no longer useable, we created an account at RapidAPI to use the free of charge Yahoo Finance API by “apidojo”. This provides access to stock data from finance.yahoo.com which is publicly available (as you would expect from stock data). There are no Terms of Use available for this API, so some legal aspects unfortunately remain unclear. We focussed our effort on collecting the price in certain intervals from a specific company, namely ADT. We did not need any other information about the company but only the price that their share was trading at on the stock market.

On the ethical side, we focused our efforts on the time the tweet was created and the sentiment of the tweet itself. We followed the main principles of ethical Data Science, illustrated by Accenture (Accenture, 2020). In particular, we excluded usernames from our project as soon as possible. We put emphasis on keeping only the part of the data that we needed for further analysis. For example, we excluded non-English tweets, as they were not useful for our processing steps and model building for sentiment analysis.

Another legal aspect arises with the usage of icons for visualizing the architecture behind our project. For this, we used the logos of the service providers, who’s services we used to complete this project. Twitter has special guidelines for the usage of their logo (only use a blue or a white version etc.), which we carefully followed. Moreover, Yahoo’s mother concern Verizon Media provides additional info on the use of their logo. The same accounts for the usage of logos of Mongo DB, Apache Spark and Kafka. All guidelines and sources of the icons are linked below. We followed the guidelines to the best of our knowledge.

# Biases & limitations of our work

## Biases

*Further information and guidelines can be found at "https://towardsdatascience.com/5-types-of-bias-how-to-eliminate-them-in-your-machine-learning-project-75959af9d3a0"*

Theory suggests five different types of biases: sample bias, exclusion bias, observer bias, prejudice and measurement bias. Sample bias is a phenomenon which is simply unavoidable. Even within the scope of our project we have been highly exposed to it. As already hinted at previously, we only looked at a small fraction of a larger universe in determining relations between sentiment and stock price developments. As the financial market is impacted by psychology to a large extent, we would need to include many more sources of opinion and expression to get a full picture of the impact that sentiment has on financial data.

Furthermore, the experimenter bias, tending to look for what we want to see, has unknowingly influenced us. Already at the beginning of the project we have believed there would rather not be any comparative pattern in the data, hence our methodology and tools used may be biased accordingly.

In addition, the prejudice bias is especially relevant when it comes to the ML algorithms which have been trained on unbalanced data. This may impact the future labelling and may be improved by using under-sampling/ over-sampling approaches to obtain a more even-handed distribution of examples.

Exclusion bias did not play a crucial role during the project, as it was mainly non-determining features such as "id" that have been omitted in due course. Also, measurement bias was non-problematic to our use case given the fact that data was collected from internationally renowned institutions, that have their own well-established frameworks and methodologies, surely bias on their side is uncontrollable to us.

## Limitations

Throughout the course of the project many of our decisions have been based on heaps of gut feeling as already hinted at in the biases section. Believing that only selected attributes could be of relevance is a potential hinderance to our conducted analysis. All this refers to the selection of features when streaming the tweets, while the exclusion biases discussed previously applies to the data cleaning process and its corresponding steps covered.

Moreover, we used the sentiment derived from the TextBlob functionality to train our machine learning models to classify sentiment which may also be prone to errors. This means that the reliability of our models builds on the reliability of the TextBlob algorithm and should therefore be taken with care.

A further aspect that must be considered when discussing the ML-algorithms is the lack of hyperparameter optimization. We have only used the default settings for all models and did not apply, for example, cross-validation approaches to improve our models. This step may have been necessary to also eliminate some of the uncertainty originating from the problem with the TextBlob classifier mentioned before.

Lastly, analysing sentiment and its effect on stock markets may not be regarded within a few weeks. A more profound and reliable analysis requires a longer time horizon allowing for flexibility and fluctuations and potentially many more aspects, which we may not even be aware of now.

# Challenges encountered

### Data Storage / connection to MongoDB

One of the biggest Challenges we were facing happened to be the one on how to store the data and how to load it back into the Spark Consumer. Since we wanted to have the sociopolitical mood for a certain amount of time in order to be able to analyze the impact on the financial market we not only needed to stream the tweets but also store them into in a way that meets the requirements of a big data project appropriately. The first approach, which was to just write the data into simple csv files and then import them into the Consumer, was not much of a problem but was far away from satisfactory. The second approach, which was to write the tweets from the producer into MYSQL worked after a couple of attempts, but unfortunately only on Marios local environment. Since we were unable to connect the local MYSQL database to the remote Universities Jupyter notebook environment, we had the same issue with than having to load the data into the Consumer as csv file, which would have been again far from satisfactory. At that process we were really perceiving the negative aspects of the distance learning since we can imagine that the troubleshooting with the local/remote environment problem would not have been taken more than a couple of minutes with the help of an experienced programmer which unfortunately we are not (yet..😉) at this point. But we would not be Data Science SBWL students if we would just give up and revert to the unsatisfactory approach of CSV files.

We dig deeper into other Data Storage Options and came across MongoDB, a distributed NoSQL cloud Database which provides up to 500 MB of free storage. With the help of many Youtube videos Carsten was able to set up the connection to MongoDB, though using a work around of insert and find statements to write and read from the MongoDB. It worked for the time being, but the approach was not the so called “state of the art”, specially not in terms of scalability. Gökhan eventually managed to set up a scalable and satisfactory Connection to the MongoDB and even got a small introduction into mandarin while digging for the solution.

### Sentiment labeling

As the for our Machine Learning Models, we had some difficulties finding reliable labeled data to train the models on to be able to classify the Sentiment. Although there are some datasets online with labeled tweets, there is no certainty that the labels are accurate, so we decided to not use them. Instead we used the Sentiment determined by the TextBlob library to train the ML models. Since we had no labeled data, we had no way to determine the accuracy of the TextBlob. we will get back to this in the limitations section. As we used 80% of the tweets as a training set, we have been left with only 20% of the tweets to extract the sociopolitical mood. Therefore, comparing the aggregated sum of the Tweet Sentiment between the TextBlob and the naive bayes classifier would result into somewhat distorted results. We decided to neglect this issue for this project as comparing the correlation of the two classifier methods have result into similar outcomes and we expect no major changes if applying the model to the whole set.

For future projects, you could deal with this matter by just storing a separate training set of tweets so the model can be applied to the whole tweet set of interest.

In terms of the Sentiment classifiers TextBlob and Vader we had the issue that many of the tweets have been labeled neutral. This has been an issue since we trained our ML models on the predicted sentiment of the TextBlob. Consequently, the ML models also labeled many neutral as neutral which do not provide real added value for our analysis.

As for our analysis we encountered some problems due to the multi labeling of the Sentiment. As we have three classes for the TextBlob and the ML models and even four for Vader (compound) we have not been able to apply typical statistical approaches for binary variables (logistic regression etc., see future work).

**Grouping by interval**

Another challenge we encountered was to group the Tweets by the desired interval to conduct our correlation analysis on. While we quickly found a solution to aggregate the sum of the Sentiment score for each hour using the pyspark.sql.functions function hour. Unfortunately, it did not provide a 15min interval option. After some further research we eventually found a satisfactory solution using the window function which allowed us to group tweets by the desired time window of 15 minutes.

# Experience gained

**Mario Matuschek:**

By carrying out this project I got involved in my first Big Data project. It was an extraordinary experience working with scalable frameworks for the first time. I admire the fact that especially the use of Apache Spark and Kafka are tools highly sought after providing ourselves with a clear comparative advantage.

Furthermore, this project has shed light on many more possibilities that exist in the vast space of data science and artificial intelligence, of which the applied machine learning section is only a minor part, and fostered myself to eagerly look for projects during the next months and years.

This project has enabled myself to get exposed to the entire range of expectations that one would have about data scientists. We cleansed the data, developed model, visualized the data and communicated results – an extraordinary experience.

**Felix Stockhammer:**

Switching from small data projects to the first big data project for this course, many additional challenges arose. Where do we get live data from? Where do we store it? It was clear that the scope of this project exceeded the ones I have previously worked on. But to face these challenges, our combined efforts of looking for solutions, that often led to other challenges, were needed, and I think that we all learned much over the course of working on this project.

In particular, I will never forget our excessive virtual collaborative programming sessions where a big sigh of relief was noticeable every time the code ran and didn't lead to another error.

Moreover, seeing the combined sentiment and financial data visualized for the first time was a relief, as we had compressed our huge dataset to the factors that we were interested in. Overall, I think that we can be very happy with the experience gained out of this project.

**Gökhan Sagir:**

The first two courses DP1 and Data Analytics were very interesting therefore I was eager with which topics we would have to deal with in the DP2 course. Through this course I made great improvements in my Python and SQL skills. It was also the first time for me to work with Apache Spark, Kafka and Mongo DB. Thereby I collected knowledge in many fields to manage analyzing Big and Scalable Data.

There have been many moments where we had desperately tried to solve an error for hours. Finding a solution for these problems enabled me to improve my Programming and mainly my Searching abilities in Chinese Websites 😊. The Project was a great experience to get in touch with Spark and Big Data and has also shown me that it was the right decision to attend this course.

**Carsten Nickel:**

As this SBWL is my first insight into the Section of Data Science without notable experience in programming, this project has been a challenging but nevertheless interesting and very rewarding experience. As this course has been my first experience with Big Data, I have not yet worked with any scalable big data frameworks such as Apache Spark and Kafka.

While Big Data so far has been a classic buzzword for me and not personally tangible yet, the course and especially the practical application during the project allowed me to gain a lot of experience in this particular area of Data Science.

Furthermore, by being in charge of setting up MongoDB I was able to gain my first actual practical experience with distributed cloud databases which have been really absorbing to work on and beyond that encouraged me to use MongoDB in private projects in the future as well.

Finally, I would like to point out that this project has been a group project which deserves to be called as one. During the whole duration of the project all our group members have been supporting each other and helped with occurring problems despite that some of them have not been in their remit.

All these aspects are the basis for a remarkable learning experience during the project which I appreciate very much.

## Future Work

The field of sentiment analysis and the goal of using these valuable insights for predicting the stock market is persistent ever since social networks prevails. Also, we think that our project could be very well amplified.

Firstly, we would suggest training and evaluating many more algorithms for tweet classification such as a logistic regression. Also already labeled balanced tweets (positive and negative, not neutral) should be used to see whether the performance of models differ. Furthermore, a very prominent suggestion is the use of neural networks (within the scope of deep learning) be it Long-Short-Term Memory or convolutional neural nets (Ghoneim, 2019). Libraries such as Tensorflow and Keras would be the most reliable and famous tools to use in this case.

Also, as already mentioned, if the reliability of the data is given or a respectable trend may be detected, predictive analytics (using regression methods) could be employed for future stock price predictions. The regression models used could also be very wide-ranging from a simple linear regression, to a more flexible lasso regression model.

## Credits

Finally, we also want to acknowledge the work of some people by whose code and solution approaches we got inspired during the execution of our project.

The foundation of the Kafka producer and the Spark Consumer is built on the code provided at Vienna University of Economics and Business course 5408 “Data Processing 2: Scalable Data Processing, Legal & Ethical Foundations of Data Science”, which is conducted by Dr. Sabrina Kirrane.

Also, want to acknowledge a github by Krishna Yerrasetty which helped us setting up the connection between the producer and the MongoDB collection. The said github can be found via this link (https://github.com/kimoyerr/twitterbot). As mentioned in the challenges encountered, The last missing piece for our MongoDB connection to work has been found on a mandarin website (https://www.coder.work/article/530074).

Regarding the location of the tweets which we aspired to obtain to look for regional differences, a stackoverflow post by a user named bross (https://stackoverflow.com/questions/54936118/getting-location-of-tweets-in-twitter-api-and-grouping-by-state ) has helped us the setting the state variable in our twitter producer.

For the Natural Language Processing part, an article by Nagesh Singh Chauhan (https://towardsdatascience.com/natural-language-processing-in-apache-spark-using-nltk-part-2-2-5550b85f3340) has helped us and performed as a guideline for our Processing.

Although posts on stack overflow and similar websites (such as those mentioned) have been helpful during the project, we do not want to neglect official documentations such as Spark MLlib (http://spark.apache.org/docs/latest/ml-guide.html ), which performed as guidance on setting up the machine learning models.

Despite getting inspired and using some snippets of the code mentioned above, the project has been carried out by us independently and therefore we claim this project to be our intellectual property.

## References

Accenture (2020): Universal principles of Data Ethics, https://www.accenture.com/_acnmedia/pdf-24/accenture-universal-principles-data-ethics.pdf (last access: 04.07.2020)

Apache Software Foundation (2020): Machine Learning Library (MLlib) Guide, http://spark.apache.org/docs/latest/ml-guide.html (last access: 04.07.2020)

Coder.work (2020): mongodb - Pyspark, https://www.coder.work/article/530074 (last access.05.07.2020)

Github (2020): Twitterbot by *Krishna Yerrasetty*, https://github.com/kimoyerr/twitterbot (last access: 04.07.2020)

Medium (2019): Natural language processing in Apache Spark using NLTK (part 2/2) by *Nagesh Singh Chauhan*, https://towardsdatascience.com/natural-language-processing-in-apache-spark-using-nltk-part-2-2-5550b85f3340 (last access: 04.07.2020)

Medium (2019): Sentiment analysis for text with Deep Learning, by *Salma Ghoneim*, https://towardsdatascience.com/sentiment-analysis-for-text-with-deep-learning-2f0a0c6472b5 (last access: 04.07.2020)

Medium (2019): 5 Types of bias & how to eliminate them in your machine learning project, https://towardsdatascience.com/5-types-of-bias-how-to-eliminate-them-in-your-machine-learning-project-75959af9d3a0 (last access: 04.07.2020)

Stackoverflow (2019): Getting location of tweets in Twitter API ad grouping by State by *bross*, https://stackoverflow.com/questions/54936118/getting-location-of-tweets-in-twitter-api-and-grouping-by-state (last access: 04.07.2020)

Twitter (2020a): Twitter Terms of Services, https://twitter.com/en/tos (last access: 27.06.2020)

Twitter (2020b): Twitter Developer Policy, https://developer.twitter.com/en/developer-terms/policy (last access: 27.06.2020)

Twitter (2020c): Twitter Developer Agreement, https://developer.twitter.com/en/developer-terms/agreement (last access: 27.06.2020)

**Company emblems and guidelines used for architecture overview:**

Apache Spark logo, by Apache Software Foundations, https://spark.apache.org/images/spark-logo.png (last access: 30.06.2020)

Apache Kafka Project Logo, by Apache Software Foundations, https://de.wikipedia.org/wiki/Datei:Apache_kafka.svg (last access: 30.06.2020)

Apache Software Foundation (2020): Apache Software Foundation Trademark usage, http://www.apache.org/foundation/marks/faq/#insideuniv (last access: 04.07.2020)

MongoDB Main Logo, by MongoDB Inc., https://company-30077.frontify.com/d/ghqwg6pjpJrq/mongodb-identity#/basics/mongodb-icons (last access: 30.06.2020)

MongoDB (2020): MongoDB Brand Resources, https://www.mongodb.com/brand-resources (last access: 04.07.2020)

RapidApi logo, by RapidApi, https://www.vectorlogo.zone/logos/rapidapi/index.html (last access: 30.06.2020)

Twitter bird Logo 2012, by Twitter, https://de.wikipedia.org/wiki/Datei:Twitter_bird_logo_2012.svg (last access: 30.06.2020)

Twitter (2020d): Twitter Brand Guidelines Version 2.0, https://about.twitter.com/content/dam/about-twitter/company/brand-resources/en_us/Twitter_Brand_Guidelines_V2_0.pdf (last access: 04.07.2020)

Yahoo! Finance logo, by Yahoo!, https://www.logo.wine/logo/Yahoo!_Finance (last access: 30.06.2020)

Verizon Media (2020): Verizon Media Brands Permissions policy, https://www.verizonmedia.com/policies/us/en/verizonmedia/permissions/requests/index.html (last access: 04.07.2020)