Anomaly Detection In Financial Data

Daniel Grimm, Gayrat Rakhimov, Suchith Shetty

December 2020

1 Introduction

This project is implementation of anomaly detection on financial data.

1.1 Datasets

2 data sets are used:

- balance_hist_anon.csv - account balance states for certain time periods. It contains 6,427,316 rows of data. It contains 6 columns:
    1. EBIZ_BALANCE_ID - unique id for each account balance state
    2. VALID_FROM - the time from which account balance is valid
    3. VALID_TO - the time till which account balance is valid
    4. GIRONUMBER - account number
    5. AMOUNT - account balance
    6. CURRENCY - currency of account balance
- eurofxref-hist.csv - contains currency exchange rates. It is provided by European Central Bank[1] and the base currency is Euro. The following currencies are used:
    - BGN - Bulgarian lev
    - CHF - Swiss franc
    - CZK - Czech krona
    - EUR - Euro
    - GBP - Great Britain pound
    - HRK - Croatian kuna
    - HUF - Hungarian forint
    - PLN - Poland zloty
    - RON - Romanian leu
    - SEK - Swedish krona
    - USD - United States dollar

2 Overall system architecture

2.1 System architecture

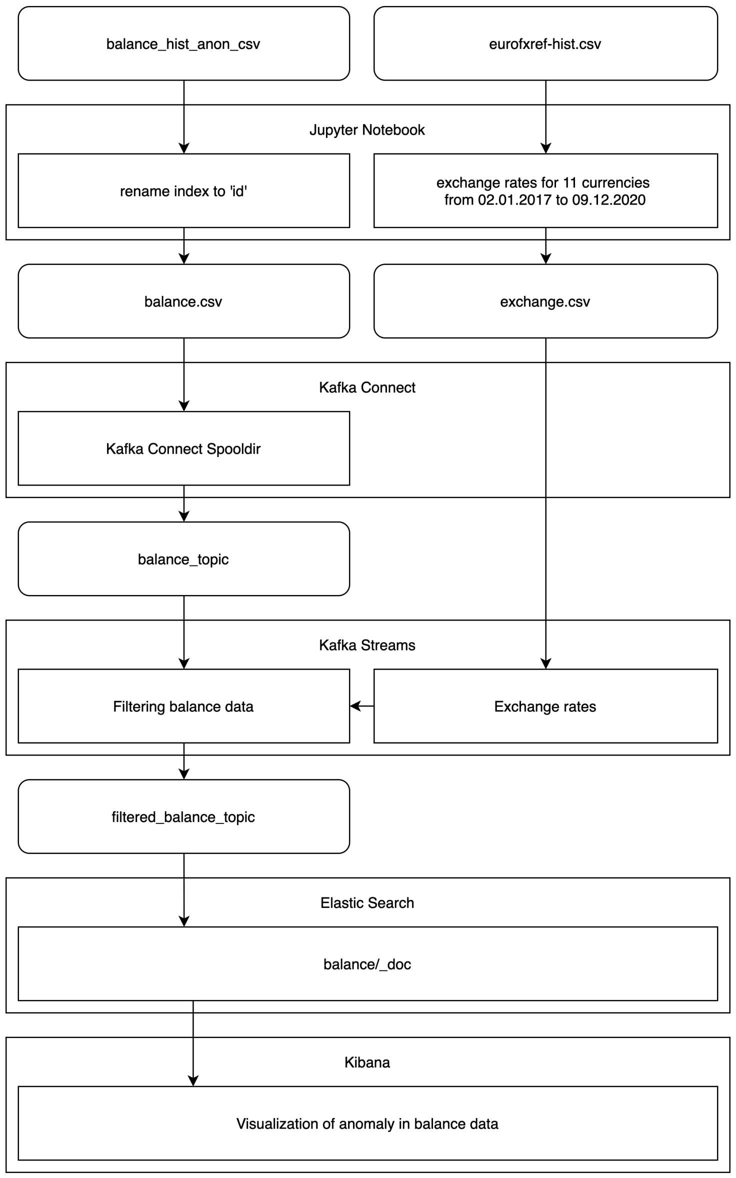


Figure 1: System architecture

2.2 Jupyter Notebook

2.2.1 Modifying balance data

Jupyter Notebook was used to understand balance data and rename first column to 'id'. For debugging purposes, it saves part of data (for example, first 10000 rows).

2.2.2 Modifying exchange rates data

Initially 'eurofxref-hist.csv' contains exchange rates for 42 currencies from 04.01.1999 till 09.12.2020. According to balance data, only exchange rates for 10 currencies from 02.01.2017 till 09.12.2020 is used. It is important to note that this data set does not contain exchange rates RSD (Serbian dinar).

2.2.3 Kafka Producer to read csv data

KafkaProducer.ipynb has the python code to load data from 'balance.csv' to a Kafka topic. Each row is read as a json object using the underlying schema. This was implemented as a first use case to explore use of python for loading data. We then moved on to one of the more sophisticated methods, i.e., Kafka Connect using SpoolDir.

2.3 Kafka Connect

2.3.1 Why Apache Kafka?

We have decided to use Apache Kafka, because according to Shree et al.(2017), it is "fast, scalable, distributed stream processing platform and fault tolerant messaging system." It is designed for low latency and high throughput[2].

2.3.2 The connector

Kafka Connect connector called Spool Dir was used to read 'balance.csv' into the stream 'balance_topic'. It reads CSV file and converts into JSON format according to the given schema. JSON data is sent to 'balance_topic'. Batch size is 1000 per batch.

2.4 Kafka Streams

According to Bejeck (2018), "Kafka Streams is a library that allows you to perform per-event processing of records". Data is processed as it is available.

2.4.1 Merging balance data with exchange rates data

Exchange rates data is read from 'exchange.csv' into HashMap. It is used to update balance data from 'balance_topic'. For example, when exchange rate is 1 EUR=1.1 USD and account balance 1100 USD is replaced by 1000 EUR.

2.4.2 Filtering balance data

First filtering is made on account balance difference. For this purpose, previous balance amount is persisted by account number. When next account balance state is received, difference is calculated and if the difference is more than predefined limit, it is considered as anomalous. For example, previous balance for the account is 5000 EUR and the current balance is 3000 EUR, the difference is 2000 EUR and it is more than 1000 EUR. This is considered as anomalous and sent to 'filtered_balance_topic'.

2.4.3 Additional ways to filter balance data

We also detect anomalies for every unique customers in the following way: we measure the mean of the previous spendings, and if the unseen data is deviating very strongly from it then we mark it as anomaly. Since data is coming line line by line, the average is recalculated every time when a new data is arriving. There is a predefined threshold for deciding whether we mark the data as anomaly or not. Also we are measuring the differences between the dates, thus we don't allow a transaction after not using the account for one year, or in case of invalid date data.

2.4.4 Identifying Anomaly

Original stream data from 'balance.csv' being loaded into a kafka topic named 'balance_topic' was analysed using multiple stream transformations to identify anomaly. Two new fields, 'ANOMALY' and 'ANOMALY_TYPE', were added during the transformation to indicate whether the record was anomalous or not, and if yes, then what kind of anomaly was detected. Transformed stream was subsequently loaded to ElasticSearch hosted locally and visualized using Kibana.

2.5 ElasticSearch and Kibana

On the consumer side of the pipeline, we chose to use ElasticSearch as the database and Kibana for visualization. ElasticSearch is easily configurable and can overcome issues in real-time data analysis which is otherwise challenging in other NoSQL or relational database services[4]. Kafka consumer module was written to stream the transformed data with information on anomaly to ElasticSearch. Kibana console was used to create, monitor and delete ElasticSearch indices, explore data loaded to ElasticSearch, and create dashboard to visualize the anomaly detection results.

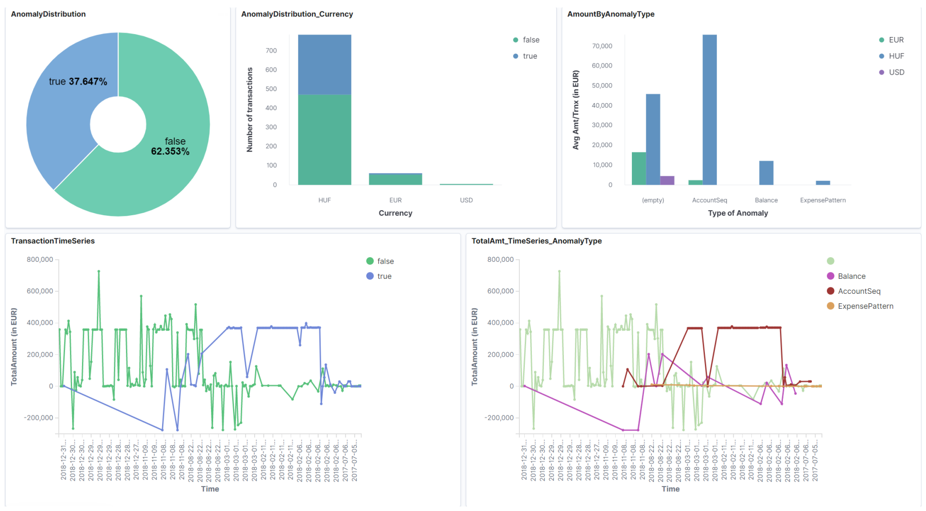


Figure 2: Snapshot of dashboard created in Kibana

3 Experiments

3.1 Merging 2 streams

It was tried to merge the balance data and currency exchange data streams into one using Kafka Streams. However, during the implementation of this idea, several problems were faced. For example, the way how to synchronize daily exchange rates with balance data on the given date was not found. Primitive merging of two streams caused the following problem: the balance data is received before exchange rate data for specific day and balance data could not be updated. For this reason, exchange rate data was read directly from file without creating stream.

3.2 Twitter API

It was tried to use Twitter tweets feed to help find anomalies on balance data. For example, tweets with keyword "HUF/USD", "HUF/EUR", "Economy of Hungary". However, the meaningful use case of tweets feeds was not found and it was decided to not to use them.

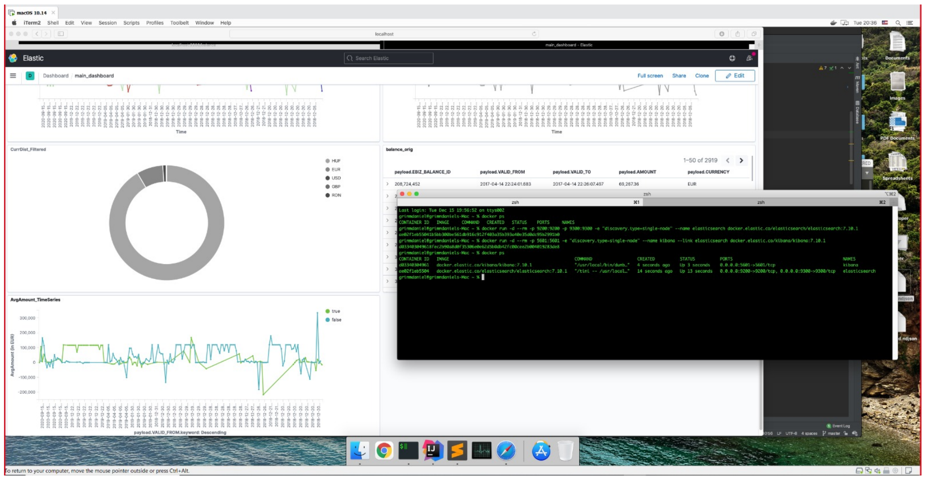
3.3 Remote ElasticSearch server

Twitter all tweets and filtered tweets are stored in remote ElasticSearch server called Bonsai. It was found out that it is convenient way to set up ElasticSearch server in short time. However, according to the project requirements balance data should not be stored in third party storage. Further, only upto 10k records can be loaded to ElasticSearch database hosted in Bonsai under the free version.

Due to these limitations, it was decided to use ElasticSearch and Kibana servers hosted locally on our personal computers.

3.4 Docker containers

Docker provides lightweight virtualization technology, which can be useful if we just want to run applications in separated environments. We don't need to virtualize a whole OS which could be very resource demanding. We decided to use docker containers for Elasticsearch and Kibana. After configuring the ports, we can use them in a very effective way since we don't need to use online solutions like Bonsai. It can be useful also when lot of people are working on the code, since it is very easy to share environments, and set them up. At the end everyone will have the same environment, which makes development a lot easier.



```
Last login: Tue Dec 15 19:56:52 on ttys002
grimmdaniel@grimmdaniels-Mac ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
grimmdaniel@grimmdaniels-Mac ~ % docker run -d --rm -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node" --name elasticsearch docker.elastic.co/elasticsearch/elasticsearch:7.10.1
ae02f1eb55041b5bb300be561db916c912f403a35b393a40e35d0dc95b2991b0
grimmdaniel@grimmdaniels-Mac ~ % docker run -d --rm -p 5601:5601 -e "discovery.type=single-node" --name kibana --link elasticsearch docker.elastic.co/kibana/kibana:7.10.1
d033403049618fec2b90a8d0f35306e0e62d5b0db42fc00cee2b004019283de8
grimmdaniel@grimmdaniels-Mac ~ % docker ps
CONTAINER ID   IMAGE                                                  COMMAND                  CREATED          STATUS         PORTS                                            NAMES
d03340304961   docker.elastic.co/kibana/kibana:7.10.1                 "/usr/local/bin/dumb…"   4 seconds ago    Up 3 seconds   0.0.0.0:5601->5601/tcp                           kibana
ae02f1eb5504   docker.elastic.co/elasticsearch/elasticsearch:7.10.1   "/tini -- /usr/local…"   14 seconds ago   Up 13 seconds  0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp   elasticsearch
grimmdaniel@grimmdaniels-Mac ~ %
```

Figure 3: Workflow with Docker containers

References

[1] European Central Bank. (2020). Euro Foreign Exchange Reference Rates. www.ecb.europa.eu/stats/eurofxref/eurofxref-hist.zip?53a6f7cba144c040b7a2a4a8242d1431.

[2] Shree, R., Choudhury, T., Gupta, S. and Kumar, P., (2017). KAFKA: The modern platform for data management and analysis in big data domain. 2017 2nd International Conference on Telecommunication and Networks (TEL-NET), 1. https://ieeexplore.ieee.org/abstract/document/8343593/

[3] Bejeck, B. (2018). Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API (1st ed.). Manning Publications.

[4] Shah, N., Willick, D. Mago, V. (2018). A framework for social media data analytics using Elasticsearch and Kibana. Wireless Netw. https://doi.org/10.1007/s11276-018-01896-2

[5] Shafi (2018). Docker, Elasticsearch and Kibana (part 1) https://medium.com/@s_tokhi/docker-elasticsearch-and-kibana-c76808ea3ab8

[6] Rahasak, . (2020). Kafka and Zookeeper with Docker. https://medium.com/rahasak/kafka-and-zookeeper-with-docker-65cff2c2c34f