Ceteris Projektdokumentation Authentic

P3-BI: Projekt Business Intelligence (PS)
FB 4 Informatik, Kommunikation und Wirtschaft
Wirtschaftsinformatik M.Sc.
HTW Berlin

Team:	Ceteris	
Abgabedatum:	10.02.2021	
Semester:	WS 20/21	
Autoren:	Lorenz Wackenhut	573525
	Manu Muttathu	573526
	Peter Grohmann	532499

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Allgemeine Informationen und Hintergrund

Die Ceteris AG wurde im Jahr 2011 gegründet und bietet innovative Lösungen im Bereich Business Analytics und Intelligence an. Aus diesem Grund wurden sie dieses Semester für das partnerschaftliche Studienprojekt des Moduls „Projekt Business Intelligence“ ausgewählt.

Eines der aktuellen Projekte der Ceteris AG findet in Kooperation mit Authentic Network statt. Authentic Network hat eine Methode entwickelt ein fälschungssicheres Plagiatsschutztool auf kostengünstige Art bereitzustellen. Es handelt sich dabei um einen mit dem Handy scannbaren Sticker, welcher das Produkt als authentisch ausweist. Die Ceteris AG ist bei diesem Projekt für das Datenmanagement, die Cloudanbindung und die Datenanalyse zuständig. Um zu testen, wie die IT Landschaft im Rahmen der Projektarbeit mit der Ceteris AG bei einem Normalbetrieb reagiert, wurde die Studentengruppe damit beauftragt einen Lasttest durchzuführen, welcher das Backend mit einem Overload an künstlich generierten Daten befüllt.

Daraus lassen sich folgende Aufgaben extrahieren:

1. Virtuelle Daten generieren
2. Daten an den SQL Server übergeben
3. Lasttest des Backends durchführen
4. Visualisierung eines Dashboardes

Folgende Ressourcen wurden von der Ceteris AG dafür zur Verfügung gestellt:

1. Zugang zur REST-API mit Dokumentation
2. Beispiel Datensatz einer zu übergebenen JSON
3. Zugang zu dem Tool Microsoft Power BI

Im Laufe des Projektes kam es jedoch dazu das Authentic vom Lasttest Abstand nahmen wollte. Das Backend war noch nicht für solch eine Art Test ausgelegt. Daraus resultierend richtet sich der Fokus auf die Entwicklung einer Datastory. In den folgenden Kapiteln werden die einzelnen Schritte zur Realisierung dieser Aufgaben näher beschrieben.

2. IT-Architektur

Die IT-Architektur für das Projekt besteht aus mehreren Services, die in die drei Kategorien Datengenerierung, Datenspeicherung und Datenanalyse eingeteilt werden können. Die von den Autoren angefertigten Entwicklungen beziehen sich ausschließlich auf die erste und dritte Schicht.

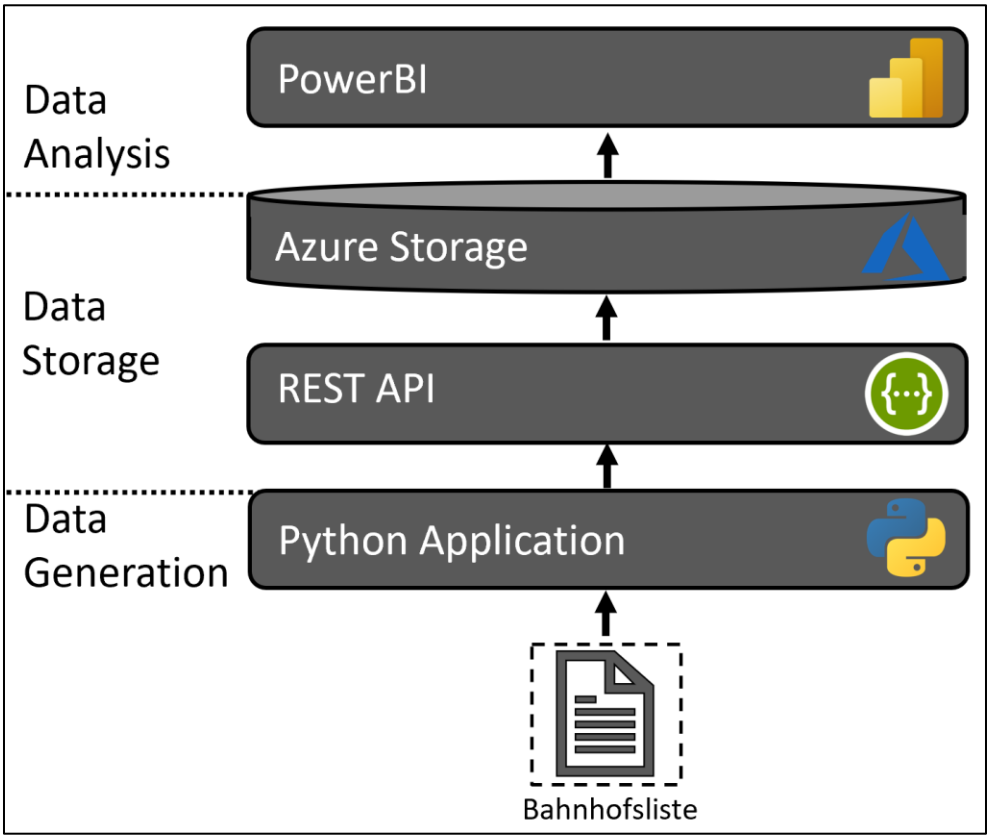

Abbildung 1: IT-Architektur

3. Datengenerierung

Das folgende Kapitel beschreibt die Auswahl der Technologien und die Funktionsweise der entwickelten Anwendung, welche für die Generierung und das Laden der Geodaten verwendet wurde.

3.1. Toolingauswahl

Python ist die De-facto-Sprache für Data Science und ebenso beliebt im Bereich des Data Engineering, unterstützt die meisten Frameworks und Pakete für beide Bereiche und liefert eine zufriedenstellende Performance für die Datenlast in diesem Projekt. Aus diesem Grund wurde Python als Programmiersprache für die Datengenerierung und die Verbindung zur REST-API verwendet.

Als Framework für die Verarbeitung der tabellarischen Daten wurde die beliebte Python-Bibliothek Pandas verwendet. Um die Koordinaten besser verarbeiten zu können wurden ebenfalls die Packages GeoPandas und Shapely eingesetzt, welche die geographischen Funktionalitäten stark erweitern.

Das Modul concurrent.futures bietet eine High-Level-Schnittstelle für die asynchrone Ausführung von Funktionen und wurde verwendet um das Laden der JSON-Pakete durch Multi-Threading zu beschleunigen.

Für den Lasttest wurde eine REST-API implementiert, die auf dem Paket Flask basiert.

3.2. Funktionsweise

Bild 2 zeigt ein UML Klassendiagramm der Applikation. In der folgenden Sektion werden die einzelnen Klassen und Methoden näher beschrieben, wobei sich, wenn auch nicht explizit erwähnt, immer auf Bild 2 bezogen wird.

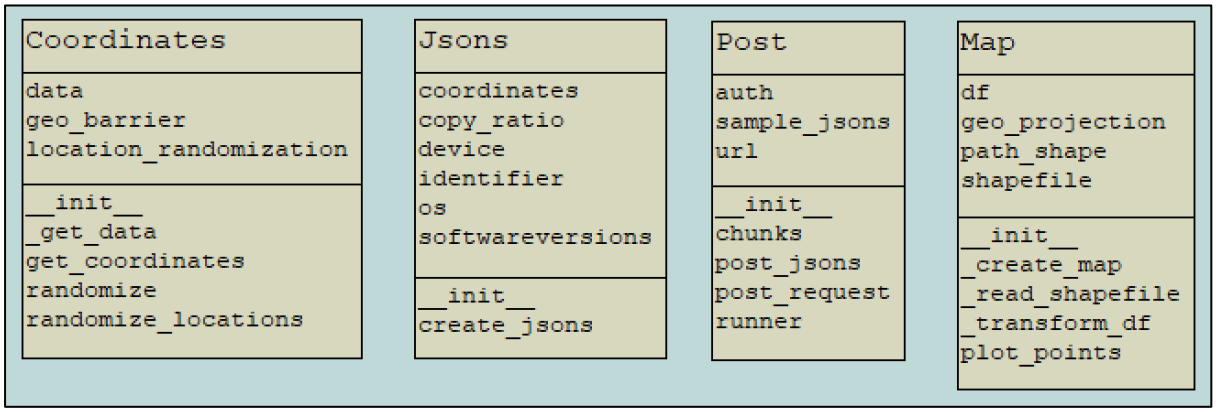

Abbildung 2:UML Klassendiagramm: Applikation

3.2.1. Extraktion der Koordinaten

Alle Methoden zur Extraktion und Transformation der Daten sind Teil der Klasse Coordinates. Im ersten Schritt wird eine Bahnhofsliste mit allen deutschen Bahnhöfen, inklusive geographischer Position, mit Hilfe der Methode _get_data() eingelesen und in der Variable data als Dataframe gespeichert. Anhand der eingestellten Filter (vgl. Konfiguration) wird ein Datensatz gewünschter Länge erstellt, wobei die Auswahl der Einträge dem Prinzip "Ziehen mit Zurücklegen" entspricht. Die Methode randomize_locations() anonymisiert die Breiten- und Längengrade der Koordinaten, indem ein zufälliger Wert innerhalb einer vordefinierten Schranke location_randomization von den Punkten addiert bzw. Subtrahiert wird.

```
def randomize_locations(data):
    """Randomizes the location by adding/substracting a random value"""

    def randomize(df, col, rand_low, rand_high):
        df[col] = (
            df[col]
            .apply(
                lambda x: str(
                    float(x.replace(",", "."))
                    + random.uniform(rand_low * (-1), rand_high)
                )
            )
            .astype("float")
        )
        return df
     rand_low = self.location_randomization
    rand_high = self.location_randomization
    if self.geo_barrier["lon_neg"]:
        rand_low = 0
    if self.geo_barrier["lon_pos"]:
        rand_high = 0
    data = randomize(data, "Longitude", rand_low, rand_high)
    if self.geo_barrier["lat_neg"]:
        rand_low = 0
    if self.geo_barrier["lat_pos"]:
        rand_high = 0
    data = randomize(data, "Latitude", rand_low, rand_high)
    return data
```

Die Methode verhindert ebenfalls, dass Punkte, durch die Streuung der Randomisierung, in ungewollten Gebieten (z.B Wasser) landen, indem durch die Flags [lat_neg; lat_pos; lon_neg; lon_pos] eine Verschiebung in die entsprechende Himmelsrichtung verhindert wird. Die Methode get_coordinates() gibt eine Liste fertig formatierter Punkte zurück.

3.2.2. Erstellung der JSON-Pakete

Die Klasse Jsons bündelt alle Methoden zur Generierung der JSON-Payloads. In der Methode create_jsons() werden die übergebenen Koordinaten mit weiteren Attributen angereichert und im JSON-Format abgespeichert. Das Attribut copy_ratio legt fest, welcher Anteil der Scans mit dem copyclassification:copy Parameter gesendet werden.

```
def create_jsons(self):
    """Creates a sample of json files with specified attributes"""
    choice_pop = ["original", "copy"]
    choice_weights = [1 - self.copy_ratio, self.copy_ratio]
    sample_jsons = []
    for x in range(len(self.coordinates)):
        sample_json = {
            "identifier": self.identifier,
            "copyclassification": random.choices(
                choice_pop, choice_weights, k=1)[0],
            "location": {
                "latitude": self.coordinates[x][1],
                "longitude": self.coordinates[x][0],
            },
            "data": {"key": "value"},
            "softwareInfo": {
                "os": self.os,
                "device": self.device,
                "softwareversion": self.softwareversions,
            },
        }
        sample_jsons.append(sample_json)
    return sample_jsons
```

3.2.3. Senden der Scans

Die Klasse Post wird verwendet, um die zuvor generierten JSON-Payloads and die REST-API zu senden. Um die Performance zu erhöhen, wird das Senden der JSON-Pakete auf mehrere Threads verteilt. Der hierdurch erreichte Speedup wird im Kapitel Lasttest näher erklärt. Die Funktion post_jsons() teilt mit der Hilfsfunktion chunks() die Pakete auf die einzelnen Threads auf und postet diese mittels post_request() and die API. Die Funktion runner() orchestriert die Zuteilung und Ausführung der Threads indem ihr die zu parallelisierende Funktion übergeben wird.
Um sicherzustellen, dass jeder Scan korrekt an die REST-Schnittstelle übermittelt wird, überprüft post_requests() über das Attribut response.ok die Antwort des Servers. Im Falle einer ‘bad response’ wird eine Exception geworfen.
Da kurzzeitige Internetabbrüche die Übertragung der Scans durch das Werfen einer Exception sofort stoppen würden, wurde ein Decorator entwickelt, der die Funktion post_request() eine definierte Anzahl von malen wieder ausführt, bevor das Programm terminiert wird.

```python
def retry(num_of_times=3):
    """Decorator that catches exceptions from a given function
     and retries to execute n times"""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            for i in range(num_of_times):
                try:
                    return func(*args, **kwargs)
                except Exception as e:
                    return_exc = e
                    print(f"Retrying...")
                    sleep(10)
            raise return_exc
        return wrapper
    return decorator
```

3.2.4. Kartenvisualisierung

Um die Einstellung der perfekten Parameter zu erleichtern wurde eine vorläufige Visualisierung der generierten Punkte ermöglicht. Die Klasse Map wird mit einem Dataframe der zuvor erstellten Daten initialisiert und transformiert dies in einen GeoDataframe mittels Geopandas. Dies ermöglicht die Transformation der Punkt-Koordinaten nach dem WGS48-Modell, um eine realistische Krümmung der Erde abzubilden. Die Klasse liest ebenfalls eine Shape-Datei mit einer Abbildung der deutschen Landesgrenzen ein. Abschließend können die Koordinaten mit Matplotlib geplottet werden.

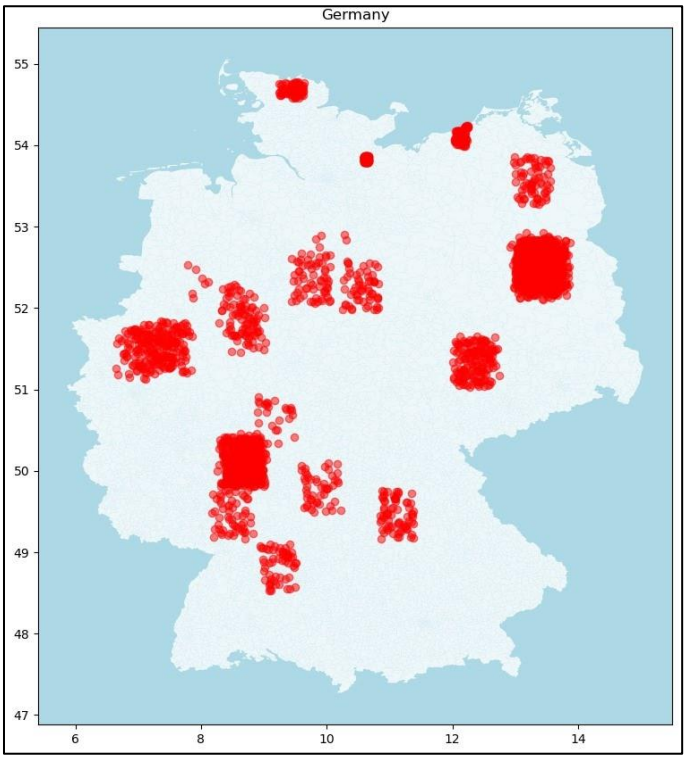

Abbildung 3: Matplotlib Abbildung der Karte

3.3. Konfiguration

Die entwickelte Anwendung kann durch eine YAML Konfigurationsdatei stark angepasst werden. Die allgemeinen Attribute können wie folgt konfiguriert werden:

```yaml
# Gibt an, ob die erstellten Scans an die API gesendet werden sollen
post_to_api: True
# Einstellung zur Kontrolle des zeitabhängigen postens
wait_time: True
# Pfad zur Ausgangsliste
coordinates_name_file: "data/200624_Bahnhofsliste.csv"
# Anteil der als Kopie klassifizierten Scans
copy_ratio: 0.3
# Identifier des Scans
identifier: "0042"
```

Zusätzlich können folgende Einstellungen für die Kartenvisualisierung getätigt werden:

```yaml
# Pfad zur Shape-Datei
shapefile: "data/shape_germany"
# Anzeigen der Kartenvisualisierung
show_map: False
# Speichern der Kartenvisualisierung
save_map: False
```

Das Posten der Scans erfolgt in sogenannten ‘Runs’. Jeder Run besteht aus einer bestimmten Anzahl an Scans, die nach Städten und Art des Bahnhofs gefiltert werden. Ein Run kann zu einer bestimmten Uhrzeit gepostet werden, um realistische Scans zu simulieren. Zusätzlich kann für jeden Run eine eigene Schranke für die Anonymisierung der Koordinaten, sowie eine Barriere für bestimmte Himmelsrichtungen festgelegt werden.

```yaml
run8:
  time: "12:26"
  city_types:
    - "Metropolbahnhof"
    - "Zentraler S-Bahnhof"
    - "Großstadtknoten"
  city_names:
    - "Hannover, Landeshauptstadt"
    - "Berlin, Stadt"
    - "Leipzig, Stadt"
  num_samples: 958
  location_randomization: 0.3
  geo_barrier:
    lat_neg: False
    lat_pos: False
    lon_neg: False
    lon_pos: False
```

3.4. Lasttest

Die Geschwindigkeit für das Senden der JSON-Pakete ist stark abhängig von der Antwortzeit des Servers. Im Test mit der Authentic-API konnte, trotz Multithreading mit bis zu 20 Workern, aufgrund der unzureichenden Ressourcen des API-Servers deshalb nur eine Geschwindigkeit von circa 20 Posts / Minute erreicht werden.
Um zu bestätigen, dass der Flaschenhals für diese ernüchternden Geschwindigkeit wirklich am Server und nicht an der entwickelten Applikation liegt, wurde eine primitive REST-API zu Testzwecken aufgesetzt.

```
from flask import Flask, jsonify, request
import json

app = Flask(__name__)
app.config["DEBUG"] = True

@app.route("/test", methods=["POST"])
def home():
    return f"<h1>This is a test API.</p>"

app.run()
```

Mit dieser, auf einem Flask-Server basierenden, REST-Schnittstelle konnte das Programm nun ohne Restriktionen getestet werden. Das Senden von 10.000 Scans dauerte nun nur noch circa 87,8 s.

REST-Schnittstelle	Durchschnittliche Zeit für einen Post
Authentic-API	2,8100 s
Flask Server	0,0087 s

Tabelle 1: Schnittstellen Perfomance

Demnach konnte ein Speedup um den Faktor >300 erreicht werden.

3.5. Implementierung

Für die Installation der Skripte wird ein Anaconda Environment empfohlen und alle folgenden Schritte basieren auf dieser Annahme.
Die Ordnerstruktur für den Ordner ‘Code’ im Abgabeordner hat die nachfolgende Struktur:

```
.
├── authentic_api_performance
│   ├── api.py
│   ├── authentic_api_performance.py
│   ├── config.yml
│   ├── data
│   │   └── 200624_Bahnhofsliste.xlsx
│   └── requirements.txt
└── authentic_api_runs
    ├── authentic_api_runs.py
    ├── config_runs.yml
    ├── data
    │   ├── 200624_Bahnhofsliste.csv
    │   └── shape_germany
    │       ├── plz-gebiete.dbf
    │       ├── plz-gebiete.prj
    │       ├── plz-gebiete.shp
    │       └── plz-gebiete.shx
    └── requirements.txt
```

Um die benötigten Pakete für die Python Skripte zu installieren müssen folgende Befehle ausgeführt werden:

```
# Neues Environment erstellen
$ conda create -n env_authentic_api python=3.8.3
# Aktivieren
$ conda activate env_authentic_api
# Pakete installieren
(env_authentic_api)$ pip install -r requirements.txt
```

4. Microsoft Power BI

In diesem Kapitel wird beschrieben wie Power Bi als Datenvisualisierungstool verwendet wurde um, ein erdachtes Fallbeispiel darzustellen.

Folgende Case Study wurde erarbeitet:

Beispiel gefälschte Schuhe

+ Es soll dargestellt werden, wie ein Typ Schuh das Land erreicht. Vom Frachtschiff in Rostock abgeladen wurde und erste Chargen rausgefiltert werden. Mit dem Verlauf der Zeit gehen von Rostock aus Richtung Süden immer mehr Fälle mit demselben Produkt ein. Die Fälle werden sich in den Metropolen sammeln.
+ So soll gezeigt werden, wie die gefälschten Waren langsam den Weg an die Kunden finden kann und der Kunde trotzdem noch die „Macht“ hat sich über die Authentizität des Schuhes zu versichern.

Um diese Case Study überhaupt erst visualisieren zu können wurde eine YML Datei mit logisch aufeinanderfolgenden Städten erstellt. Es wurde versucht natürlich erscheinende Uhrzeiten zu wählen in Kombination mit der Abfolge der Städte. Dieser Logik folgend sind die meisten Datenpunkte von Rostock aus nach Süden von Stadt zu Stadt gewandert. Einige Punkte wurden in späteren Runs in bereits erschienen Städte hinzugefügt, um zu verhindern, dass alle Daten gleichzeitig in einer Stadt erscheinen und danach nie wieder.

Im ersten Schritt wurde die Anbindung zum Windows SQL-Server der Ceteris GmbH eingerichtet. Daraufhin wurden die für dieses Projekt interessanten Tabellen in PowerBI geladen. Anschließend wurde das Visualisierungselement „Zuordnung“ mit den Longitudes und Latitudes bestückt. Als Legende diente die Measure „copyclassification“ um die einzelnen Einträge voneinander unterscheiden zu können.

Ein genereller Überblick soll gewährt werden indem das Balkendiagramm „Anzahl der Scans nach Copyclassification“ am rechten Rand des Dashboards dargestellt wird. Es werden die als Originale gelabelt aufgezählt und die Scans welche als Copy gelabelt wurden gegenübergestellt.

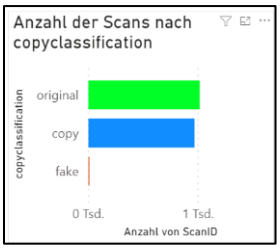

Abbildung 4:Anzahl der Scans nach Copyclassification

Um den Nutzer eine Möglichkeit zu bieten den Zeitraum der Betrachtung einzuschränken wurde aus dem „PowerBI Visuals Store“ das „Time Brush“ Element hinzugefügt. Dies erlaubt es anhand des Datums ein Anfangs- und Endpunkt für die Betrachtung einzustellen.

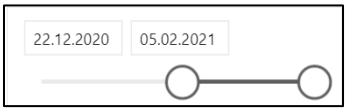

Abbildung 5: Time Brush

Um die gewünschte Animation der Scans über die Karte zu realisieren wurde das Element „Play Axis“ aus dem Store angewandt. Dieses Bedienfeld erlaubt es auf der Karte die Scans iterativ anzuzeigen.

Abbildung 6: Uhrzeit Scan

Abschließend wurde ein allgemeiner Filter für dieses Dashboard angelegt mit welchen man die „Copyclassification“ auf ein bestimmtes Element filtern kann. Demgemäß ist es möglich sich auf der Karte z.B. nur Kopien anzeigen zu lassen.

Diese Schritte führten zu folgendem Ergebnis:

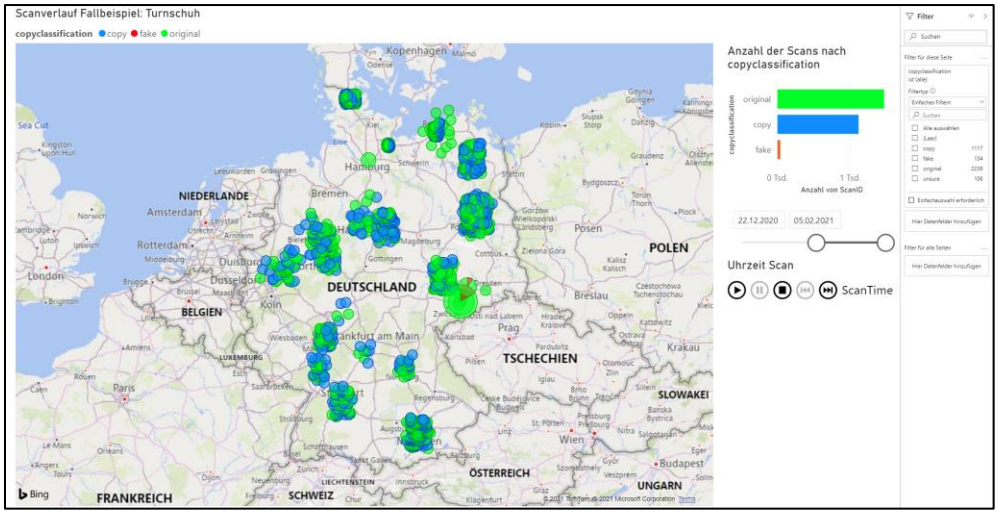

Abbildung 7:Power BI Dashboard

Im Dashboard ist es leider nicht möglich einen Verlauf darzustellen. Es ist somit schwierig den konkreten „Weg“ der Scans nachvollziehen zu können. Da aber diese Funktion von der Bearbeitungsgruppe als wichtig betrachtet wurde, wurde eine alternative Darstellung in Tableau erstellt.

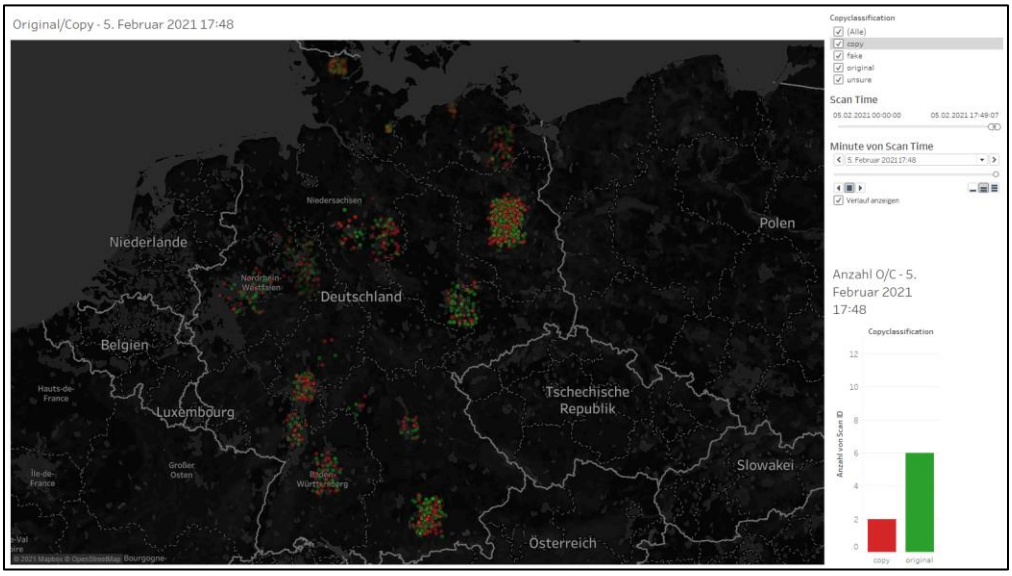

Abbildung 8: Tableau Dashboard

Die Implementierung in Tableau lief analog zu der Visualisierung von PowerBI ab. In Tableau konnte der Verlauf der Scans dargestellt werden.