Corona Datenverarbeitung

Prüfungsstudienarbeit des 4. Semesters  
Prüfer: Markus Eider M.Sc.

**Studiengang:**  Angewandte Informatik

**Semester:** SS 2020

**Datum der Abgabe:** 27.05.2020

|  |  |  |
| --- | --- | --- |
| Name | Matrikelnummer | E-Mail |
| Philipp Muhr  Michael Mican  Maximilian Seitz | 00692629 00692390 00692807 | philipp.muhr@stud.th-deg.de michael.mican@stud.th-deg.de maximilian.seitz@stud.th-deg.de |

Erreichte Punktezahl: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Note: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Erstkorrektur Unterschrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Zweitkorrektur Unterschrift: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Inhalt

[1 Einleitung 3](#_Toc41485813)

[1.1 Motivation 3](#_Toc41485814)

[1.2 Problemstellung 3](#_Toc41485815)

[1.3 Lösungsansatz 4](#_Toc41485816)

[2 Methodologie 4](#_Toc41485817)

[2.1 Datenbeschaffung 4](#_Toc41485818)

[2.2 Datenverarbeitung 5](#_Toc41485819)

[2.3 Probleme 8](#_Toc41485820)

[3 Ergebnisse 11](#_Toc41485821)

[4 Schlussfolgerungen 14](#_Toc41485822)

[5 Arbeitsteilung 15](#_Toc41485823)

[6 Anhang 15](#_Toc41485824)

# Einleitung

## Motivation

Das Corona Virus versetzt die gesamte Welt in einen Ausnahmezustand. Bereits im März 2020 wurde eine weltweite Pandemie ausgerufen. Bisher ist zudem unklar warum es in verschiedenen Ländern zu unterschiedlichen Verläufen kommt. Diese Situation führt zu Maßnahmen, wie Maskenpflicht in der Öffentlichkeit, Ein-/Ausreiseverbote oder sogar zu Ausgangssperren innerhalb von „Hot Spots“. In dieser Studienarbeit soll die Ausbreitung des Virus untersucht werden. Um Klarheit gegenüber der Expansion zu erhalten, werden die Gesundheitssysteme, der Gini-Koeffizient und die aktuellen Google Trends der Länder analysiert. Durch einen Vergleich der verschiedenen Daten inklusive automatisierter Visualisierung sollen neue Kenntnisse bezüglich COVID-19 gewonnen werden. Die expliziten Herausforderungen der Studienarbeit werden im folgenden Kapitel, der Problemstellung, behandelt.

## Problemstellung

Damit die Ausbreitung des Corona Virus visualisiert und analysiert werden kann, muss eine geeignete Datenbasis geschaffen werden. Da es sich um die Ausbreitung eines Virus handelt können dementsprechend Daten vom Europäischen Zentrum für die Prävention und die Kontrolle von Krankheiten, kurz ECDC, bezogen werden. Jedoch ist es das Ziel, Korrelationen zu verschiedenen anderen Faktoren festzustellen. Daher reichen die Daten des ECDCs allein nicht aus und es müssen weitere Informationen gewonnen werden. Diese Arbeit wird nicht nur die Ausbreitung des Virus aufzeigen, sondern auch die Zusammenhänge zwischen Corona und dem Zustand des Gesundheitssystems verschiedener Länder. Weiterhin wird die Einkommens- und Vermögensverteilung untersucht, um einen möglichen Zusammenhang zwischen COVID-19 und dem Zustand der Gesellschaft bezüglich Armut und Reichtum festzustellen. Außerdem werden Google Anfragen visualisiert, um aufzuzeigen, wie sich das Interesse der Bevölkerung im Verlauf der Pandemie verhält. Damit die eben vorgestellten Ziele dieser Studienarbeit umgesetzt werden können, muss vorab eine Roadmap für einen Lösungsansatz verfasst werden. Diese wird im folgenden Kapitel beschrieben.

## Lösungsansatz

Da es sich beim Corona Virus um eine Pandemie handelt, muss der gesamte Zeitverlauf möglichst Welweit betrachtet werden. Die Suche nach Geodaten ergab, dass die Corona Verbreitungsdaten vom ECDC bezogen werden. Das Google Interesse, welches sich aus den Suchanfragen der Suchmaschine in den jeweiligen Ländern ergibt, wird von der Google API, mithilfe der Python Bibliothek „pytrends“, bezogen. Außerdem werden die Gini-Koeffizienten (Index für Einkommens- und Vermögensverteilung) der verschiedenen Länder von der „World Bank Group“ als csv Tabelle angefragt. Sowohl die Datenverarbeitung, als auch die Visualisierung, wird in Python realisiert. Es werden aus Übersichtsgründen mehrere Skripte erstellt, die den Download, die Verarbeitung und die Darstellung vollautomatisiert durchführen. Die Skripte sind so konzipiert, dass sie durch das Ausführen einer Windows batch Datei gestartet werden. Die Python Bibliothek „matplotlib“ unterstützt das Zeichnen von Graphen, mit der es möglich ist Diagramme und auch Weltkarten zu erstellen. Durch diese werden unsere finalen Berechnungen schlussendlich visuell dargestellt. Im nächsten Kapitel werden die Techniken genauer erläutert.

# Methodologie

## Datenbeschaffung

Vom ECDC konnten Daten im csv Format heruntergeladen werden. Der Datensatz beinhaltet den Anstieg der Corona Fälle in nahezu allen Ländern weltweit. Diese Daten bilden die Grundlage der Studienarbeit, da alle weiteren Analysen sich auf die Corona Fälle beziehen. Ursprünglich sollten die Daten von der „World Health Organization“ (WHO) bezogen werden. Allerdings befanden sich die Daten der Corona Fälle auf der Website der WHO im pdf Format und waren daher deutlich schwieriger zu verarbeiten als die des ECDC.

Die Datenbeschaffung der Google Suchen erfolgte über eine nicht öffentliche Google API. Um den Zugriff auf diese Schnittstelle zu erleichtern wird die Python Bibliothek pytrends verwendet. Diese sollen das Interesse der Bevölkerung an der Thematik in den einzelnen Ländern wiederspiegeln.

Zusätzlich wurden Daten zum Gini-Koeffizienten einbezogen, um aufzuzeigen, wie sich COVID-19 auf Länder mit unterschiedlich starker Einkommensverteilung auswirkt. Diese Daten kamen von der World Bank ebenfalls im csv Format.

Als letztes wurden von der WHO Werte bezogen, welche die weltweiten Ausgaben für die Gesundheitsysteme in verschiedenen Ländern darstellen. Diese sollen später im Vergleich mit den Corona Fällen Aufschluss geben, ob die per capita Ausgaben für das Gesundheitssystem im Zusammenhang mit den COVID-19 Fällen stehen. Nachdem nun die richtigen Datensätze verfügbar sind, wird auf die Aufbereitung der Daten eingegangen.

## Datenverarbeitung

Die erhaltenen Daten werden automatisiert von den verschiedenen Skripten verarbeitet. Die Software kann über eine batch Datei gestartet werden, welche sich außerdem um die Installation benötigter Python Bibliotheken kümmert. (siehe Code 1)

@echo off

cd scripts

echo Installing requirements...

pip install -r requirements.txt

if %errorlevel%==1 (

    echo Failed to install requirements!

    pause

    exit /B 1

)

echo Requirements installed successfully!

echo Running script...

python main.py

pause

*Code 1: „run.bat“*

Anschließend wird das main.py Skript gestartet welches übergeifend mit den anderen Skripten zusammenarbeitet. Bevor dieses andere Methoden startet, wird die Ordnerstruktur des Projekts erstellt. (siehe Code 2)

def createDir(dirname):

    Path(dirname).mkdir(parents = True, exist\_ok = True)

def createDirs(dirnames):

    for dirname in dirnames:

        createDir(dirname)

def createAllDir():

    createDirs([

        "../dat/temp/",

        "../dat/temp/googleTrends/",

        "../dat/temp/countryBorders/",

        "../dat/temp/giniData/",

        "../out/",

        "../out/caseNumberHistoryPerCountry/",

        "../out/maps/",

        "../out/maps/giniCaseCoef/",

        "../out/maps/giniDeathCoef/",

        "../out/healthSpending/",

        "../out/giniCoefficient/"

    ])

*Code 2: Auszug aus „main.py“. Erstellt die Ordnerstruktur.*

download.py kümmert sich dabei um den Download der unterschiedlichen Datensätze und überschreibt zusätzlich möglicherweise veraltete, schon vorhandene Daten. Im nächsten Schritt werden die Informationen alle durch load.py in den Arbeitspeicher geladen, um sie verarbeiten zu können. Dabei werden einige Daten außerdem gruppiert und als Dictionary zwischengespeichert. Die im Arbeitsspeicher vorliegenden Daten können nun visualisiert werden.

In preprocessing.py befinden sich Methoden, welche die geladenen Daten verarbeiten und für die visuelle Aufbereitung vorbereiten. Beispielsweise wird in Code 3 die Bevölkerung eines Landes für ein bestimmtes Jahr aus den Bevölkerungsdaten der UN bezogen.

def extractCountryPopulationForYear(populationRaw, year):

    population = {}

    for countryPopulation in populationRaw[year]:

        if countryPopulation["VarID"] == '2':

            country = pycountry.countries.get(

                numeric=countryPopulation["LocID"].zfill(3))

            if country != None:

                population[country.alpha\_2] = float(

                    countryPopulation["PopTotal"])

            else:

                log.logWarning(

                    "Countrycode of " + countryPopulation["Location"] + "couldnt be parsed")

    return population

*Code 3: Auszug aus „preprocessing.py“. Extrahiert die Population des Jahres für alle Länder.*

Die beiden Skripte draw.py und plot.py übernehmen die Visualisierung der Daten. Durch plot.py werden Graphen erstellt, die einen besseren Überblick über die genauen Zahlen geben. Aktuell werden Diagramme zum zeitlichen Ablauf der täglichen und der gesamten Menge der Corona Fälle weltweit erstellt. Auf diesem Graph wird außerdem das Interesse an Corona anhand der Google Suchanfragen dargestellt. Durch draw.py lassen sich ganze Karten erstellen, die einen Farbverlauf beinhalten. In diesem Projekt werden Karten gezeichnet, die die Corona Fälle visualisieren. Ebenfalls wird das Verhältnis der Ausgaben eines Gesundheitsystems pro Einwohner und das des Gini-Koeffizienten und der COVID-19 Fälle visualisiert. Um nun den zeitlichen Verlauf kenntlich zu machen wurden sogenannte GIFs erstellt. Ein GIF verhält sich wie ein Daumenkino und zeigt mehrere Bilder hintereinander. Daher bietet sich diese Darstellungsform an, um den Ablauf der Corona Krise als Weltbild abzubilden. Die Verarbeitung der Daten brachte einige Herausforderungen mit sich, welche im nächsten Kapitel aufgezeigt und behandelt werden.

## Probleme

Zu Beginn des Projekts musste die Art der Visualisierung der Ergebnisse festgelegt werden. Die Herausforderung hierbei war es, visuell mehrere Dimensionen abzubilden. Das Ergebnis dieser Studienarbeit sollte den Verlauf der Corona Pandemie sowohl zeitlich, als auch geographisch unterteilt darstellen. Daher wurde sich in dieser Arbeit für Weltkarten entschieden, die den zeitlichen Verlauf der Krise als GIF ablaufen lassen. Die Karten werden als Shape-Datei verarbeitet, was problematisch für das Zeichnen der Karten war. Shape Dateien bestehen aus mehreren Teilen, die auch einzeln gezeichnet werden müssen (siehe Code 4).

def drawShape(shape, color):

    # Iterate through the parts of the shape

    for i in range(len(shape.parts)):

        partStart = shape.parts[i]

        # Get the ID of the last point of this part

        if i == len(shape.parts) - 1:

            partEnd = len(shape.points)

        else:

            partEnd = shape.parts[i + 1]

        # Draw this part

        x = [i[0] for i in shape.points[partStart:partEnd]]

        y = [i[1] for i in shape.points[partStart:partEnd]]

        plt.fill(x, y, c = color)

*Code 4: Auszug aus „draw.py“.*

Wenn diese nicht separat gezeichnet werden, dann entsteht eine nicht zu identifizierende Grafik. Für weiterführende Informationen werden die Karten und GIFs durch Graphen unterstützt, welche exakte Werte zu den bestimmten Zeitpunkten liefern. Ein weiteres Problem, das es zu lösen galt, war, dass manche csv Dateien im UTF-8 BOM Format ankommen und folglich konvertiert werden müssen. Diese Kodierung führt in den Skripten zu unerwünschten Fehlern. Daher wurde die Byte Abfolge, die am Anfang der Datei steht, entfernt (siehe Code 5). Durch diese Anpassung wird dieselbe Datei im normalen UTF-8 Format abgelegt. Die UTF-8 BOM Kodierung ist mittlerweile in den meisten Fällen überflüssig. Sie dient lediglich dazu die Interpretation verschiedener Zeichenkodierter Daten zu erleichtern.

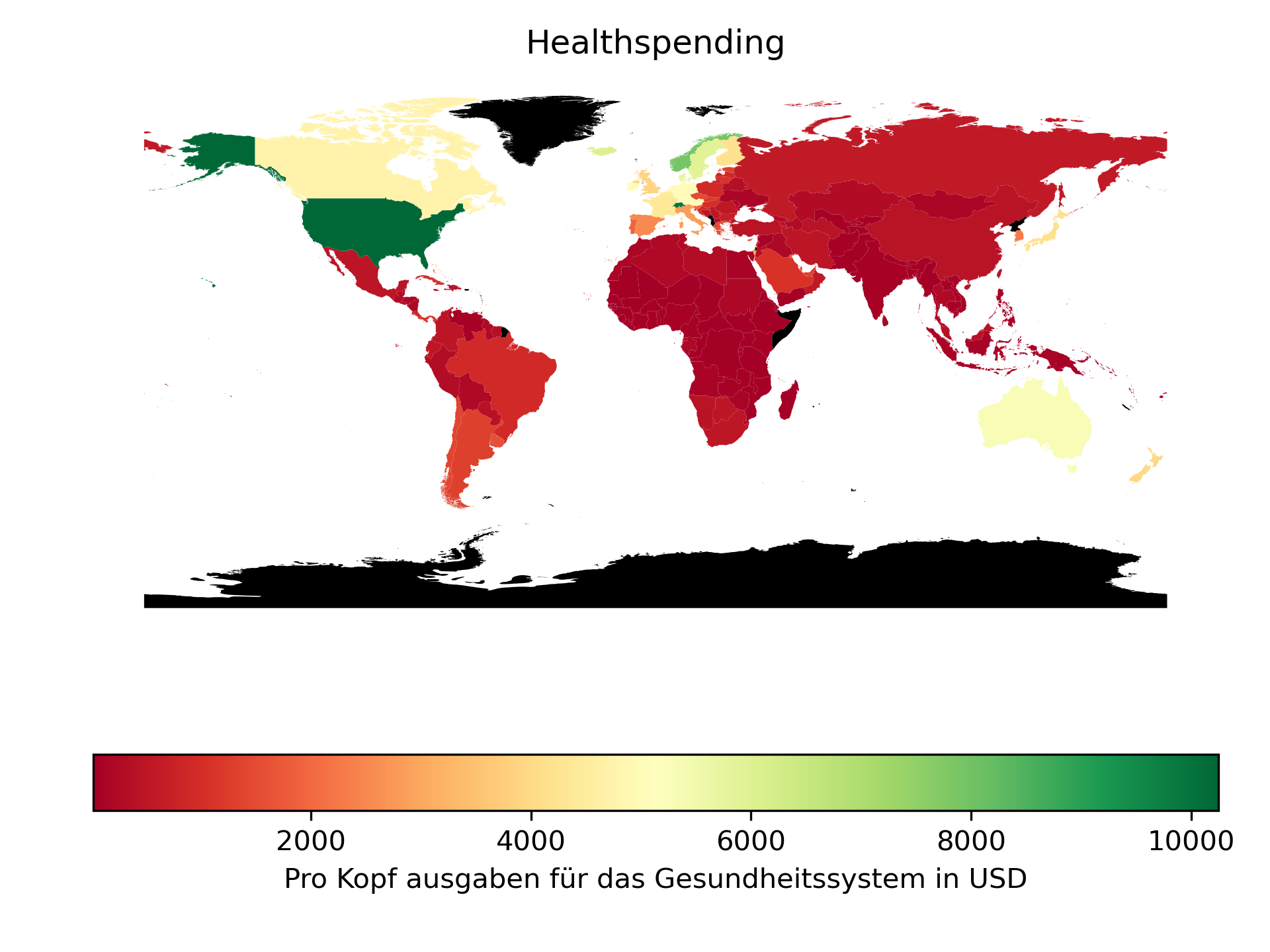
dataFrame = pd.read\_csv(path + "WorldBankGiniIndex.csv", skiprows=4)

dataFrame.to\_csv(path + "WorldBankGiniIndex.csv", index=False)

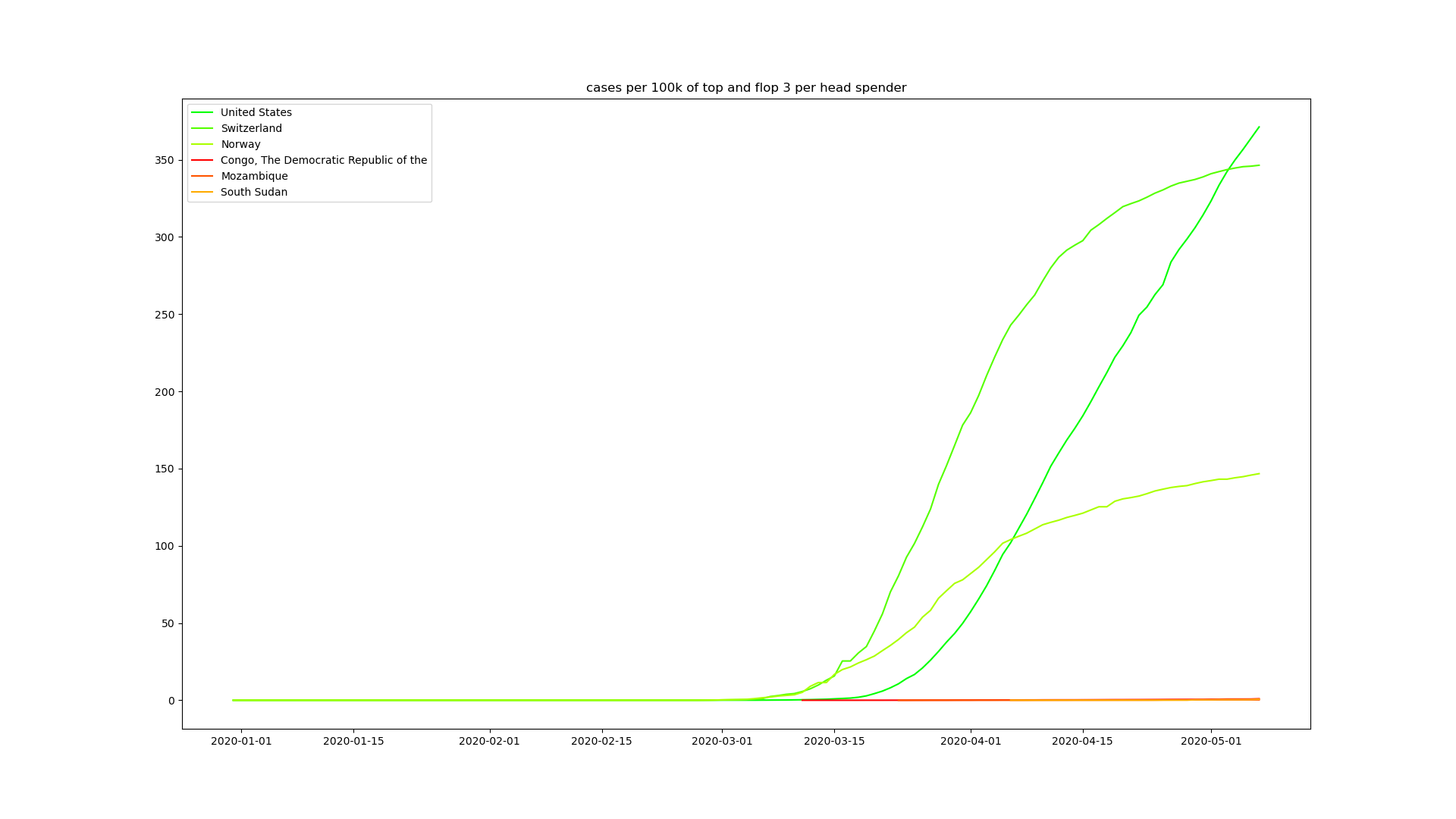
*Code 5: Auszug aus „download.py“. Hier werden die ersten 4 Zeilen inklusive der BOM Byte Folge entfernt.*

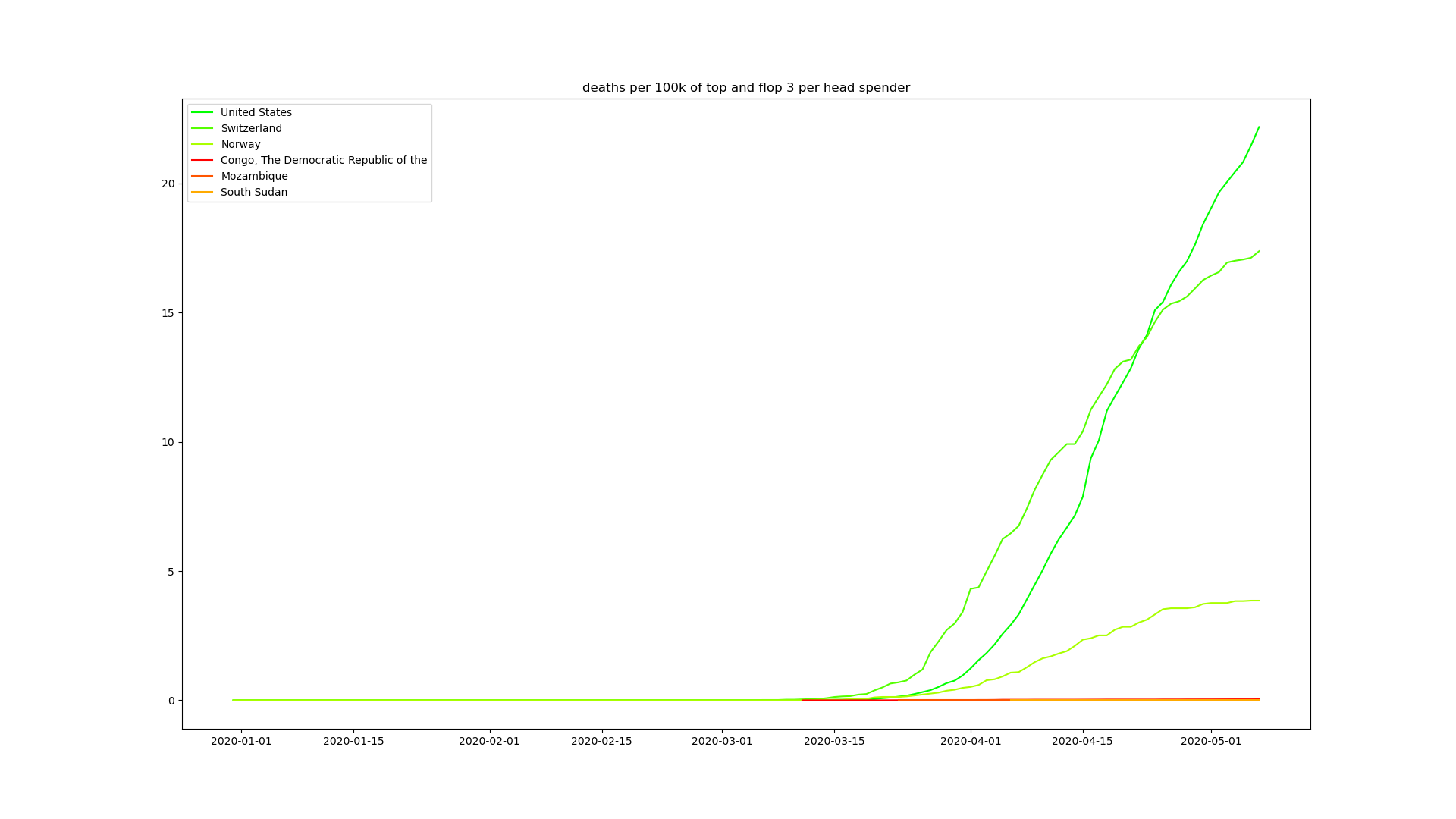
Da sich das Problem durch Löschen der ersten Zeichen der Datei lösen lässt, handelt es sich hier um eine kleinere Komplikation. Die Schwierigkeit bestand lediglich darin den Fehler zu erkennen. Weiterhin gab es bei der Google API Erschwernisse durch deren Sicherheitsmaßnahmen. Hierfür wird eine Schnittstelle verwendet, welche nicht mehr öffentlich dokumentiert ist. Somit musste die Funktionsweise selbst ergründet werden, wobei es zu verschiedenen Authentifikationsschwierigkeiten kam. Als finale Lösung wurde die python Bibliothek pytrends gewählt, welche einen verhältnismäßig leichten Zugang zu dieser Google trends API ermöglicht. Eine weitere Komplikation stellte der ErrorCode 429 - Too many requests dar. Dieser Fehler tritt auf, da die Google API nur eine limitierte Anzahl an Zugriffen zulässt und sich so vor Überlastung schützt. Die Lösung bestand darin eine einminütige Anfragepause einzusetzen. Weiterhin gab der Datensatz des ECDC zwei Länder nicht im richtigen Ländercode zurück und hatte eine Unterscheidung zwischen Japan und einem japanischen Schiff. Das japanische Schiff, auch bekannt als die „Diamond princess - JPG11668“, ist ein Kreuzfahrtschiff, auf dem Corona ausgebrochen war. Diese müssen als Sonderfälle betrachtet werden und der identifizierende Ländercode muss dementsprechend angepasst werden. Hierbei wurde die ISO 3166–alpha2 gewählt. Das ist wichtig, da die Skripte die unterschiedlichen Werte über den Zeitverlauf durch diesen Ländercode zusammenfügen können. Eine weitere Abweichung wurde bei dem Land Kosovo festgestellt. Dieses Land wurde noch nicht von allen Ländern als souveräner Staat anerkannt und deshalb wird vom Skript dieser Ländercode als Falsch interpretiert, da dieser formell noch nicht existiert. Durch Einzelfallbehandlung kann aber auch dieses Problem gelöst werden. Zunächst war eine detaillierte Ansicht für Taiwan geplant. Dies scheiterte aus politischen Gründen, da durch Chinas Einfluss keine vertraulichen Daten für Taiwan auffindbar waren. Das liegt daran, dass China Tawian nicht als autonomen Staat ansieht, sondern als Teil der Republik. Im folgenden Kapitel werden nun die Ergebnisse der Arbeit beschrieben.

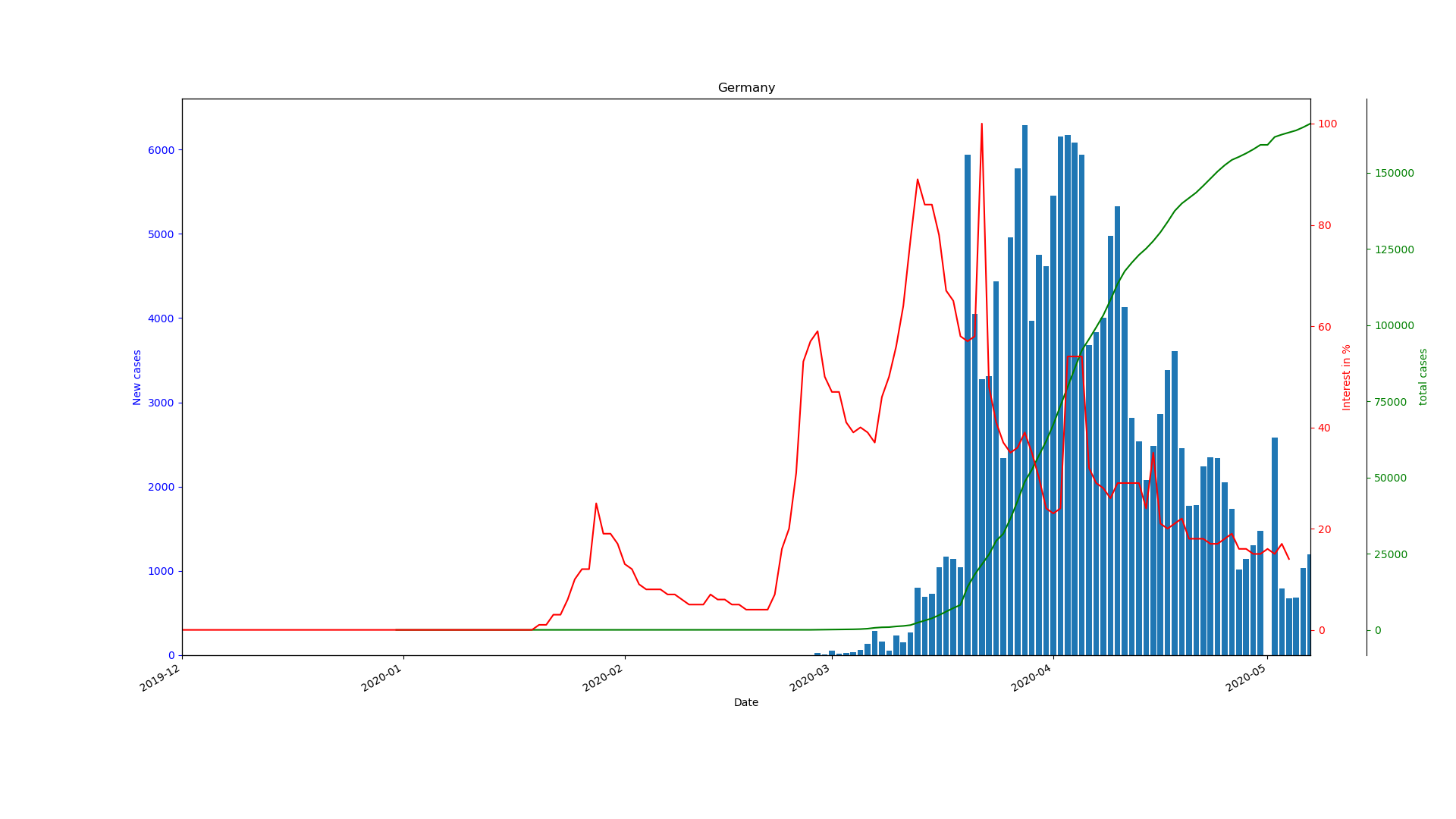
# Ergebnisse

Als Ergebnis präsentiert die Projektarbeit drei Weltkarten im GIF Format. Darunter die Visualisierung der COVID-19 Fälle, der an Corona verstorbenen und den Gini-Koeffizienten im Verhältnis mit den Corona Fällen. Dazu gibt es eine weitere Weltkarte mit den aktuellen Ausgaben für das Gesundheitsystem der Länder pro Kopf gerechnet (siehe Abb. 1).

*(Abb. 1 Gesundheitsausgaben aktuell)*

Ergänzend liegen zu den besten und schlechtesten 3 der Länder mit den Höchsten, beziehungsweise den niedrigsten Ausgaben des Gesundheitssystems Graphen vor, die dementsprechend die neuen Corona-Infizierten und Todesfälle anzeigen (siehe Abb. 2 und 3). *(Abb. 2 Coronafälle der Länder mit den höchsten Ausgaben im Gesundheitswesen)*

*(Abb. 3 Coronatode der Länder mit den höchsten Ausgaben im Gesundheitswesen)*

Anhand dieser Erkenntnisse kann festgestellt werden, dass die Investitionen in das Gesundheitssystem eines Landes keinen direkten Einfluss auf die Anzahl der Corona Erkrankungen hat.Die USA beispielsweise investiert pro Kopf am meisten Geld in das Gesundheitssystem, jedoch verbreitet sich das Corona Virus dort rapide. Auffällig ist, dass besonders die Länder, welche nur sehr wenig Geld in ihr Gesundheitssystem investieren, eine tendentiell geringe Anzahl von Corona-Infizierten anzeigen. Die tatsächliche Anzahl der Corona Patienten könnte in Wirklichkeit deutlich höher ausfallen, da diese aufgrund von beispielsweise mangelnden Corona Tests nicht erfasst werden können und somit nicht in der Statistik beziehungsweise der Visualisierung auftauchen. Weiterhin gibt es ein Diagramm pro Land, das den Zeitverlauf der Fälle insgesamt, der neu gemeldeten Fälle und das Interesse (bemessen an der Anzahl der Google Suchanfragen) an dem Virus, aufzeigt. Bei Betrachtung des Interesses der Bevölkerung an Corona in Deutschland, kann sehr schnell ein Höhepunkt im März festgelegt werden. Das liegt vermutlich

*(Abb. 4 Corona Graph Deutschland)*

daran, dass an diesen Tagen die Ausgangsbeschränkungen in Bayern beschlossen wurden und in diesem Monat die meisten Neuinfektionen verzeichnet wurden. Zusätzlich kann seit Anfang April ein konstanter „Abfall“ des Interesses beobachtet werden (siehe Abb. 4).Dies könnte darauf hindeuten, dass dieser Ausnahmezustand allmählich zur Gewohnheit wird. Im letzten Abschnitt werden sowohl positive als auch negative Aspekte der Darstellung geschildert. Zusätzlich wird diskutiert, was noch verbessert werden könnte.

# Schlussfolgerungen

Zusammenfassend lässt sich sagen, dass sich der Historienverlauf der Corona Ausbreitung sehr gut durch die GIF Bildabfolge darstellen lässt. Die zusätzlichen Diagramme veranschaulichen die Zahlen und Fakten zur COVID-19 Thematik sehr gut. Die größte Herausforderung bestand darin die Daten möglichst wahrheitsgemäß darzustellen. Außerdem ist es kaum möglich alle Daten in einer Karte oder einem Graphen darzustellen. Mehrere visuelle Aufbereitungen werden aufgrund der Menge an verschiedenen Informationen benötigt. Die Farbcodierung der Karten könnten noch verbessert werden. Dies gestaltet sich schwierig, da kleine Differenzen in einer globalen Farbskala nicht ersichtlich werden würden. Die Analyse der Corona Fälle beziehungsweise der Todesfälle im Vergleich zu den pro Kopf Ausgaben der Länder für das Gesundheitssystem ist nicht besonders aussagekräftig, da Länder mit schlechtem Gesundheitssystem Neuinfektionen kaum feststellen und melden können. Für die Analyse des Gesundheitssystems müssten somit andere Kennzahlen und Daten einbezogen werden. Da es sich um ein sehr aktuelles Thema handelt ist es wahrscheinlich, dass die Datenquellen Ihre API-Endpunkte verändern könnten, was ein Problem für das Skript darstellt.

# Arbeitsteilung

Kartenzeichnung, Code Cleanups, Gifs Maximilian Seitz

Schriftliche Auführung der PStA, Gini Koeffizient Download & Verarbeitung Philipp Muhr

Google Trends Analyse, Gesundheitsausgaben Verarbeitung, Corona Ausbreitung Verarbeitung Michael Mican

Datenrecherche, Datenverarbeitung (Code), Download Alle

# Anhang

**Code:**

**main.py**

from pathlib import Path

import plot

import draw

import load

import download

import preprocessing

import log

import os

scriptPath = os.path.dirname(os.path.realpath(\_\_file\_\_))

os.chdir(scriptPath)

log.logInfo("Scriptpath: " + str(scriptPath))

def main():

    log.logInfo("Creating Directories")

    createAllDir()

    log.logInfo("Downloading corona cases")

    download.downloadCoronaCases()

    log.logInfo("Loading corona cases into memory")

    coronaCases = load.loadCoronaCases()

    log.logInfo("Downloading world popoulation")

    download.downloadWorldPopulation()

    log.logInfo("Loading world popoulation into memory")

    population = load.loadPopulationGroupedByYear()

    log.logInfo("Downloading health spending data")

    download.downloadHealthSpendingPerCapita()

    log.logInfo("Loading health spending data into memory")

    healthSpendingDict = load.loadHealthSpendingPerCapita()

    log.logInfo("Plotting topFlop health spender")

    plot.plotTopFlopHealthSpendingCoronaCases(preprocessing.getTopFlopCountries(coronaCases, healthSpendingDict, 3), preprocessing.extractCountryPopulationForYear(population, "2020"))

    log.logInfo("Downloading country borders")

    download.downloadCountryBorders()

    log.logInfo("Downloading Google trends data")

    download.downloadGoogleTrendsData(coronaCases.keys())

    log.logInfo("Loading Google trends data into memory")

    googleTrends = load.loadGoogleTrendsData()

    log.logInfo("Creating Plots")

    plot.plotCaseGoogleTrends(coronaCases, googleTrends)

    log.logInfo("Drawing corona case maps")

    draw.generateCoronaCaseWorldMaps()

    log.logInfo("Downloading Gini-Coefficient")

    download.downloadGiniCoefficient()

    log.logInfo("Loading Gini-Coefficient data into memory")

    giniCoefficient = load.loadGiniData()

    log.logInfo("Creating Gini-Coefficient Plots")

    plot.plotGiniData(giniCoefficient)

    log.logInfo("Creating Gini-Coefficient grouped dictionary")

    newestGiniCoefficientDict = preprocessing.getNewestGiniCoefficientDict(giniCoefficient)

    log.logInfo("Creating top flop gini coef. plot")

    plot.plotTopFlopGiniCoefficientOverview(newestGiniCoefficientDict)

    log.logInfo("Creating gini coef map")

    preprocessing.generateGiniCoefficientMap(newestGiniCoefficientDict)

    log.logInfo("Creating gini-cases coef map")

    preprocessing.generateGiniCoronaMap(coronaCases, newestGiniCoefficientDict, preprocessing.extractCountryPopulationForYear(population, "2020"))

    log.logInfo("Creating health spending per capita map")

    preprocessing.generateHealthSpendingMap(healthSpendingDict)

def createDir(dirname):

    Path(dirname).mkdir(parents = True, exist\_ok = True)

def createDirs(dirnames):

    for dirname in dirnames:

        createDir(dirname)

def createAllDir():

    createDirs([

        "../dat/temp/",

        "../dat/temp/googleTrends/",

        "../dat/temp/countryBorders/",

        "../dat/temp/giniData/",

        "../out/",

        "../out/caseNumberHistoryPerCountry/",

        "../out/maps/",

        "../out/maps/giniCaseCoef/",

        "../out/maps/giniDeathCoef/",

        "../out/healthSpending/",

        "../out/giniCoefficient/"

    ])

main()

**download.py**

import requests

import os

import csv

from pytrends.request import TrendReq

import pytrends

from log import printProgressBar

import zipfile

import log

import datetime

import time

import dload

import pandas as pd

def request(source, target):

    log.log("Downloading...")

    result = requests.get(source)

    open(target, "wb").write(result.content)

    log.log("Download finished!")

def downloadWorldPopulation():

    source = "https://population.un.org/wpp/Download/Files/1\_Indicators%20(Standard)/CSV\_FILES/WPP2019\_TotalPopulationBySex.csv"

    target = "../dat/temp/population.csv"

    request(source, target)

def downloadCoronaCases():

    source = "https://opendata.ecdc.europa.eu/covid19/casedistribution/csv"

    target = "../dat/temp/coronaCases.csv"

    request(source, target)

def downloadCountryBorders():

    source = "https://opendata.arcgis.com/datasets/252471276c9941729543be8789e06e12\_0.zip"

    dload.save\_unzip(source, extract\_path='../dat/temp/countryBorders', delete\_after=True)

def downloadGiniCoefficient():

    source = "https://api.worldbank.org/v2/en/indicator/SI.POV.GINI?downloadformat=csv"

    path = "../dat/temp/giniData/"

    target = path + "GiniData.zip"

    request(source, target)

    with zipfile.ZipFile(target, 'r') as zipObj:

        zipObj.extractall(path)

    if(os.path.exists(path + "WorldBankGiniIndex.csv")):

        os.remove(path + "WorldBankGiniIndex.csv")

    for filename in os.listdir(path):

        if(filename == "API\_SI.POV.GINI\_DS2\_en\_csv\_v2\_988343.csv"):

            try:

                os.rename(path + "API\_SI.POV.GINI\_DS2\_en\_csv\_v2\_988343.csv", path + "WorldBankGiniIndex.csv")

            except Exception as error:

                log.logError("Renaming failed - Error: " + str(error))

        else:

            try:

                os.remove(path + filename)

            except Exception:

                log.logError("Removing unused .csv tables failed - Error")

    log.log("Deleting first 4 rows of WorldBankGiniIndex.csv ...")

    dataFrame = pd.read\_csv(path + "WorldBankGiniIndex.csv", skiprows=4)

    dataFrame.to\_csv(path + "WorldBankGiniIndex.csv", index=False)

    log.log("Deleting rows finished!")

def downloadHealthSpendingPerCapita():

    source = "http://apps.who.int/gho/athena/api/GHO/GHED\_CHE\_pc\_US\_SHA2011/?format=csv"

    target = "../dat/temp/healthSpendingPerCapita.csv"

    request(source, target)

def downloadGoogleTrendsData(geoIdArray):

    pytrend = TrendReq()

    maxLength = len(geoIdArray)

    progress = 0

    tooManyRequestGeoIdArray = []

    for geoId in geoIdArray:

        printProgressBar(progress, maxLength,

                         "Downloading Google Trends data for " + geoId)

        geoIdForGoolge = geoId

        #special cases

        if geoId == "UK":

            geoIdForGoolge = "GB-ENG"

        if geoId == "EL":

            geoIdForGoolge = "GR"

        if(geoIdForGoolge != "" and geoIdForGoolge != None and geoIdForGoolge != "N/A"):

            try:

                pytrend.build\_payload(kw\_list=['/m/01cpyy'], timeframe='2019-11-01 ' + datetime.datetime.now().strftime("%Y-%m-%d"), geo=geoIdForGoolge)

                interest\_over\_time\_df = pytrend.interest\_over\_time()

                interest\_over\_time\_df.to\_csv("../dat/temp/googleTrends/" + geoId + ".csv")

            except pytrends.exceptions.ResponseError as error:

                if error.response.status\_code == 429:

                    log.logInfo("Google is mad at us that we make so much requests so were waiting for a minute before continuing (geoId will be redownloaded at the end)")

                    tooManyRequestGeoIdArray.append(geoId)

                    time.sleep(60)

                else:

                    log.logError("Downloading googletrends for geoId " + geoId + " failed - Errormessage was: " + str(error))

            except Exception as error:

                log.logWarning("Downloading googletrends for geoId " + geoId + " failed - Errormessage was: " + str(error))

        progress += 1

    if len(tooManyRequestGeoIdArray) > 0:

        downloadGoogleTrendsData(tooManyRequestGeoIdArray)

**preprocessing.py**

import csv

import log

import pycountry

import draw

def extractCountryPopulationForYear(populationRaw, year):

    population = {}

    for countryPopulation in populationRaw[year]:

        if countryPopulation["VarID"] == '2':

            country = pycountry.countries.get(

                numeric=countryPopulation["LocID"].zfill(3))

            if country != None:

                population[country.alpha\_2] = float(

                    countryPopulation["PopTotal"])

            else:

                log.logWarning(

                    "Countrycode of " + countryPopulation["Location"] + "couldnt be parsed")

    return population

def getTopFlopCountries(coronaCasesDataDict, healthSpendingDict, count):

    for countryKey in healthSpendingDict:

        healthSpendingDict[countryKey] = sorted(healthSpendingDict[countryKey], key=lambda e:

                                                int(e["YEAR"])

                                                )

    # sorted of dicts returns only keys as array

    sortedHealthSpendingCountryKeys = sorted(healthSpendingDict, key=lambda e:

                                             float(

                                                 healthSpendingDict[e][-1]["Numeric"])

                                             )

    topCountries = []

    counter = 0

    for countryKey in reversed(sortedHealthSpendingCountryKeys):

        try:

            country = pycountry.countries.get(alpha\_3=countryKey)

            conutryAlpha2 = country.alpha\_2

            if conutryAlpha2 in coronaCasesDataDict.keys():

                counter += 1

                topCountries.append(

                    {"coronaCases": coronaCasesDataDict[conutryAlpha2], "healthSpending": healthSpendingDict[countryKey], "alpha\_2": conutryAlpha2})

            if counter >= count:

                break

        except Exception as e:

            log.logWarning(

                "Error while preprocessing sortedHealthSpending " + str(e))

    flopCountries = []

    counter = 0

    for countryKey in sortedHealthSpendingCountryKeys:

        try:

            country = pycountry.countries.get(alpha\_3=countryKey)

            conutryAlpha2 = country.alpha\_2

            if conutryAlpha2 in coronaCasesDataDict.keys():

                counter += 1

                flopCountries.append(

                    {"coronaCases": coronaCasesDataDict[conutryAlpha2], "healthSpending": healthSpendingDict[countryKey], "alpha\_2": conutryAlpha2})

            if counter >= count:

                break

        except Exception as e:

            log.logWarning(

                "Error while preprocessing sortedHealthSpending " + str(e))

    return {

        "top": topCountries,

        "flop": flopCountries,

    }

def generateGiniCoefficientMap(newestGiniCoefficientDict):

    dataToDraw = {}

    for countryKey, countryGiniCoefData in newestGiniCoefficientDict.items():

        dataToDraw[countryKey] = countryGiniCoefData["value"]

    draw.generateMaps({"gini-coeff": dataToDraw}, colorMap = 'RdYlGn')

def convertCasesDeathsToTotalCases(coronaCasesDataDict):

    convertedDict = coronaCasesDataDict.copy()

    for countryKey, coronaCasesOfCountry in convertedDict.items():

        totalDeaths = []

        totalCases = []

        sortedCornaCasesOfCountry = sorted(coronaCasesOfCountry, key=lambda e: int(

            e["year"]) \* 10000 + int(e["month"]) \* 100 + int(e["day"]))

        for coronaCasesDaily in sortedCornaCasesOfCountry:

            if len(totalCases) > 0:

                totalCases.append(totalCases[-1] + int(coronaCasesDaily["cases"]))

                coronaCasesDaily["totalCases"] = totalCases[-1]

            else:

                totalCases.append(int(coronaCasesDaily["cases"]))

                coronaCasesDaily["totalCases"] = totalCases[-1]

            if len(totalDeaths) > 0:

                totalDeaths.append(totalDeaths[-1] + int(coronaCasesDaily["deaths"]))

                coronaCasesDaily["totalDeaths"] = totalCases[-1]

            else:

                totalDeaths.append(int(coronaCasesDaily["deaths"]))

                coronaCasesDaily["totalDeaths"] = totalCases[-1]

    return convertedDict

def generateHealthSpendingMap(healthSpendingDict):

    dataForMapGeneration = {"healthspending": {}}

    for countryKey in healthSpendingDict:

        sortedHealthSpendingByYear = sorted(healthSpendingDict[countryKey], key=lambda e:

                                                int(e["YEAR"])

                                                )

        countryKeyConverter = pycountry.countries.get(alpha\_3 = sortedHealthSpendingByYear[-1]["COUNTRY"])

        if countryKeyConverter != None:

            dataForMapGeneration["healthspending"][countryKeyConverter.alpha\_2] = float(sortedHealthSpendingByYear[-1]["Numeric"])

    draw.generateMaps(dataForMapGeneration, colorMap = 'RdYlGn', targetFolder="../out/healthSpending/")

def generateGiniCoronaMap(coronaCasesDataDict, newestGiniCoefficientDict, populationOfYear):

    dataForMapCase = {}

    dataForMapDeaths = {}

    convertedCasesDataDict = convertCasesDeathsToTotalCases(coronaCasesDataDict)

    for countryKey, casesOfCountry in convertedCasesDataDict.items():

        for dayCaseOfCountry in casesOfCountry:

            if dayCaseOfCountry["dateRep"] not in dataForMapCase.keys():

                dataForMapCase[dayCaseOfCountry["dateRep"]] = {}

                dataForMapDeaths[dayCaseOfCountry["dateRep"]] = {}

            if countryKey in newestGiniCoefficientDict.keys() and countryKey in populationOfYear.keys():

                dataForMapCase[dayCaseOfCountry["dateRep"]][countryKey] = newestGiniCoefficientDict[countryKey]["value"]/100 \* (dayCaseOfCountry["totalCases"]/(populationOfYear[countryKey] \* 1000)) \* 100000

                dataForMapCase[dayCaseOfCountry["dateRep"]][countryKey] = newestGiniCoefficientDict[countryKey]["value"]/100 \* (dayCaseOfCountry["totalDeaths"]/(populationOfYear[countryKey] \* 1000)) \* 100000

    draw.generateMaps(dataForMapCase, colorMap = 'RdYlGn', targetFolder="../out/maps/giniCaseCoef/")

    draw.generateMaps(dataForMapDeaths, colorMap = 'RdYlGn', targetFolder="../out/maps/giniDeathCoef/")

def getNewestGiniCoefficientDict(giniDataDictionary):

    returnDict = {}

    for countryKey in giniDataDictionary:

        newestYearWithData = -1

        currentDataValue = -1

        for countryGiniData in giniDataDictionary[countryKey]:

            for year in countryGiniData.keys():

                if year.isdecimal():

                    if countryGiniData[year] != "":

                        newestYearWithData = year

                        currentDataValue = float(countryGiniData[year])

        countryObj = pycountry.countries.get(alpha\_3=countryKey)

        if int(newestYearWithData) > 0 and countryObj != None:

            returnDict[countryObj.alpha\_2] = {

                "countryKey\_alpha2": countryObj.alpha\_2,

                "countryKey\_alpha3": countryObj.alpha\_3,

                "lastYearWithData": newestYearWithData,

                "value": currentDataValue,

            }

        else:

            if int(newestYearWithData) > 0:

                if countryKey.lower() == 'XKX'.lower():

                    returnDict['XK'] = {

                        "countryKey\_alpha2": 'XK',

                        "countryKey\_alpha3": 'XKX',

                        "lastYearWithData": newestYearWithData,

                        "value": currentDataValue,

                    }

                else:

                    log.log(

                        countryKey + " was skipped because it has an invalid alpha 3 country key")

            else:

                log.log(countryKey +

                        " was skipped because it has no Gini-Coefficient data")

    return returnDict

**load.py**

import os

import csv

import log

COMMA = ','

def loadCSV(filepath, delimiter = COMMA):

    dataArray = []

    with open(filepath) as csv\_file:

        csv\_reader = csv.DictReader(csv\_file, delimiter = delimiter)

        for row in csv\_reader:

            dataArray.append(row)

    return dataArray

def loadGroupedCSV(filename, group, delimiter = COMMA):

    dataArray = loadCSV(filename, delimiter)

    groupedDict = {}

    for row in dataArray:

        try:

            groupedDict[row[group]].append(row)

        except KeyError:

            groupedDict[row[group]] = []

            groupedDict[row[group]].append(row)

        except Exception as error:

            log.logError("Error while grouping - Error: " + str(error))

    return groupedDict

def loadPopulationGroupedByYear():

    return loadGroupedCSV('../dat/temp/population.csv', "Time")

def loadHealthSpendingPerCapita():

    return loadGroupedCSV('../dat/temp/healthSpendingPerCapita.csv', "COUNTRY")

def loadCoronaCases(group = "geoId"):

    return loadGroupedCSV('../dat/temp/coronaCases.csv', group)

def loadGiniData():

    return loadGroupedCSV('../dat/temp/giniData/WorldBankGiniIndex.csv', "Country Code")

def loadGoogleTrendsData():

    returnDict = {}

    for filename in os.listdir("../dat/temp/googleTrends/"):

        if(filename.endswith(".csv")):

            countryTrendsData = loadCSV('../dat/temp/googleTrends/' + filename)

            returnDict[filename.replace(".csv", "")] = countryTrendsData

    return returnDict

**plot.py**

import log

import matplotlib.pyplot as plt

from matplotlib.transforms import Bbox

import datetime

import numpy as np

import pycountry

import math

def plotTopFlopGiniCoefficientOverview(newestGiniCoefficientPerCountryDict):

    sortedCountryKeys = sorted(newestGiniCoefficientPerCountryDict,key=lambda e: newestGiniCoefficientPerCountryDict[e]["value"], reverse=True)

    rankingPos = 0

    for countryKey in sortedCountryKeys:

        if rankingPos < 5 or rankingPos > (len(sortedCountryKeys) - 1 - 5):

            red = hex(255)

            green = hex(int(math.floor(rankingPos \* 1/10 \* 255)))

            blue = hex(0)

            if rankingPos > (len(sortedCountryKeys) - 1 - 5):

                red = hex(int(math.floor((len(sortedCountryKeys) - rankingPos) \* 1/8 \* 255)))

                green = hex(255)

                blue = hex(0)

            countryData = newestGiniCoefficientPerCountryDict[countryKey]

            plt.barh(pycountry.countries.get(alpha\_2=countryKey).name + " (" + countryData["lastYearWithData"] + ")", countryData["value"], color="#"+str(red).replace("0x","").zfill(2)+str(green).replace("0x","").zfill(2)+str(blue).replace("0x","").zfill(2))

        elif rankingPos == 5:

            plt.barh("...", 0)

        rankingPos += 1

    figure = plt.gcf()

    figure.set\_size\_inches(19.2, 10.8)

    title = "Best and worst Gini-Coefficient in the world"

    plt.title(title)

    plt.savefig("../out/giniCoefficient/topFlopList.png", bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

    plt.clf()

    plt.close()

def plotTopFlopHealthSpendingCoronaCases(topFlopCountryData, population):

    topAndFlopArray = []

    topAndFlopArray.extend(topFlopCountryData["top"])

    topAndFlopArray.extend(topFlopCountryData["flop"])

    place = 0

    for countryData in topAndFlopArray:

        x = []

        totalCases = []

        yCasesPer100kCitizens = []

        totalDeaths = []

        yDeathsPer100kCitizens = []

        countryPopulation = population[countryData["alpha\_2"]] \* 1000

        if countryPopulation > 0:

            sortedCasesCountryData = sorted(countryData["coronaCases"], key=lambda e: int(

            e["year"]) \* 10000 + int(e["month"]) \* 100 + int(e["day"]))

            for caseData in sortedCasesCountryData:

                x.append(datetime.date(int(caseData["year"]), int(

                    caseData["month"]), int(caseData["day"])))

                if len(totalCases) > 0:

                    totalCases.append(totalCases[-1] + int(caseData["cases"]))

                else:

                    totalCases.append(int(caseData["cases"]))

                if len(totalDeaths) > 0:

                    totalDeaths.append(totalDeaths[-1] + int(caseData["deaths"]))

                else:

                    totalDeaths.append(int(caseData["deaths"]))

                yCasesPer100kCitizens.append((totalCases[-1]/countryPopulation) \* 100000)

                yDeathsPer100kCitizens.append((totalDeaths[-1]/countryPopulation) \* 100000)

            red = hex(85 \* place)

            green = hex(255)

            blue = hex(0)

            if(place > 2):

                red = hex(255)

                green = hex(85 \* (place%3))

                blue = hex(0)

            colorString = "#"+str(red).replace("0x","").zfill(2)+str(green).replace("0x","").zfill(2)+str(blue).replace("0x","").zfill(2)

            countryObj = pycountry.countries.get(alpha\_2=countryData["alpha\_2"])

            plt.figure(1)

            plt.plot\_date(x, yCasesPer100kCitizens, color=colorString, linestyle='solid', marker='None', label=countryObj.name)

            plt.figure(2)

            plt.plot\_date(x, yDeathsPer100kCitizens, color=colorString, linestyle='solid', marker='None', label=countryObj.name)

            place += 1

        else:

            log.logError("There is no countryPopulation data for " + countryData.alpha\_2)

    plt.figure(1)

    figure = plt.gcf()

    figure.set\_size\_inches(19.2, 10.8)

    title = "cases per 100k of top and flop 3 per head spender"

    plt.title(title)

    plt.legend()

    plt.savefig("../out/healthSpending/cases.png", bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

    plt.clf()

    plt.close()

    plt.figure(2)

    figure = plt.gcf()

    figure.set\_size\_inches(19.2, 10.8)

    title = "deaths per 100k of top and flop 3 per head spender"

    plt.title(title)

    plt.legend()

    plt.savefig("../out/healthSpending/deaths.png", bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

    plt.clf()

    plt.close()

def plotCaseGoogleTrends(coronaCaseDataDict, googleTrendsDataDict):

    maxLength = len(coronaCaseDataDict)

    progress = 1

    for countryKey in coronaCaseDataDict:

        log.printProgressBar(progress, maxLength,

                             "Saving plot for " + countryKey)

        xCorona = []

        yCorona = []

        yCoronaTotal = []

        sortedCases = sorted(coronaCaseDataDict[countryKey], key=lambda e: int(

            e["year"]) \* 10000 + int(e["month"]) \* 100 + int(e["day"]))

        for countryDict in sortedCases:

            xCorona.append(datetime.date(int(countryDict["year"]), int(

                countryDict["month"]), int(countryDict["day"])))

            yCorona.append(int(countryDict["cases"]))

            if(len(yCoronaTotal) > 0):

                yCoronaTotal.append(

                    yCoronaTotal[len(yCoronaTotal) - 1] + int(countryDict["cases"]))

            else:

                yCoronaTotal.append(int(countryDict["cases"]))

        fig, ax1 = plt.subplots()

        ax1.bar(xCorona, yCorona)

        ax1.xaxis\_date()

        fig.autofmt\_xdate()

        ax1.set\_xlim([datetime.date(2019, 12, 1), datetime.date.today()])

        ax1.set\_ylabel('New cases')

        ax1.set\_xlabel('Date')

        ax1.yaxis.label.set\_color("blue")

        ax1.tick\_params(axis='y', colors="blue")

        ax3 = ax1.twinx()

        ax3.plot\_date(xCorona, yCoronaTotal, linestyle='solid',

                      marker='None', color='green')

        ax3.set\_ylabel('total cases')

        ax3.yaxis.label.set\_color("green")

        ax3.tick\_params(axis='y', colors="green")

        if coronaCaseDataDict[countryKey][0]["geoId"] in googleTrendsDataDict.keys():

            xTrends = []

            yTrends = []

            for dayData in googleTrendsDataDict[coronaCaseDataDict[countryKey][0]["geoId"]]:

                datetimeTrends = datetime.datetime.strptime(

                    dayData["date"], "%Y-%m-%d")

                xTrends.append(datetimeTrends.date())

                yTrends.append(int(dayData["/m/01cpyy"]))

            ax2 = ax1.twinx()

            ax2.plot\_date(xTrends, yTrends, linestyle='solid',

                          marker='None', color='red')

            ax2.set\_ylabel('Interest in %')

            ax2.yaxis.label.set\_color("red")

            ax2.tick\_params(axis='y', colors="red")

            ax3.spines["right"].set\_position(("axes", 1.05))

        figure = plt.gcf()

        figure.set\_size\_inches(19.2, 10.8)

        title = coronaCaseDataDict[countryKey][0]["countriesAndTerritories"].replace(

            "\_", " ")

        plt.title(title)

        plt.savefig("../out/caseNumberHistoryPerCountry/" + countryKey +

                    ".png", bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

        plt.clf()

        plt.close()

        progress += 1

def plotGiniData(giniDataDict):

    maxLength = len(giniDataDict)

    progress = 1

    for countryKey in giniDataDict:

        giniValue = []

        years = []

        log.printProgressBar(progress, maxLength, "Saving plot for " + countryKey)

        for countryGiniData in giniDataDict[countryKey]:

            for year in countryGiniData.keys():

                if year.isdecimal():

                    if(countryGiniData[year] != ''):

                        giniValue.append(float(countryGiniData[year]))

                        years.append(int(year))

        if len(giniValue) > 0:

            #plt.plot(years, giniValue, marker="o", linestyle= "solid")

            #bildet den den aktuellsten wert ab

            #plt.bar(years[len(years) - 1], giniValue[len(giniValue) - 1])

            #bildet alle werte ab

            plt.bar(years, giniValue)

            plt.ylabel('Gini-Coefficient')

            plt.xlabel('Year')

            # evtl dynamisch setzen?

            plt.xlim([1960, 2020])

            plt.ylim([0, 100])

            figure = plt.gcf()

            figure.set\_size\_inches(19.2, 10.8)

            if(countryKey == ""):

                plt.savefig("../out/giniCoefficient/noCountryCode\_Gini.png",

                            bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

            else:

                plt.savefig("../out/giniCoefficient/" + countryKey +

                            "\_Gini.png", bbox\_inches=Bbox(np.array([[0, 0], [19.2, 10.8]])))

            # plt.show()

            plt.clf()

            plt.close()

        progress += 1

**log.py**

import os

import datetime

import sys

logFolder = "../log/"

def checkPath():

    exists = os.path.exists(logFolder)

    if not exists:

        print("[INFO] " + "A log folder was created here: " + os.path.abspath(logFolder))

        os.makedirs(logFolder)

def logError(logMessage):

    writeLog(logMessage, 3)

    print("[ERROR] " + logMessage, file=sys.stderr)

def logWarning(logMessage):

    writeLog(logMessage, 2)

    print("[WARNING] " + logMessage)

def logInfo(logMessage):

    writeLog(logMessage, 1)

    print("[INFO] " + logMessage)

def log(logMessage):

    writeLog(logMessage, 0)

    print(logMessage)

#type 0 = normal; 1 = Info; 2 = warning; 3 = error;

def writeLog(logMessage, logType=0):

    checkPath()

    typeString = " "

    if logType == 1:

        typeString = " [INFO] "

    elif logType == 2:

        typeString = " [WARNING] "

    elif logType == 3:

        typeString = " [ERROR] "

        with open(logFolder  + "ERROR" + datetime.datetime.now().strftime("%Y-%m-%d")+".log", "a+") as file:

            file.write(datetime.datetime.now().strftime("%d.%m.%Y - %H:%M:%S") + typeString + "- " + logMessage + "\n")

    with open(logFolder + datetime.datetime.now().strftime("%Y-%m-%d")+".log", "a+") as file:

        file.write(datetime.datetime.now().strftime("%d.%m.%Y - %H:%M:%S") + typeString + "- " + logMessage + "\n")

def printProgressBar(progress, maxLength, message=""):

    os.system('cls')

    loadingStr = "["

    for x in range(30):

        if((progress/maxLength) \* 30 >= x):

            loadingStr += "█"

        else:

            loadingStr += "-"

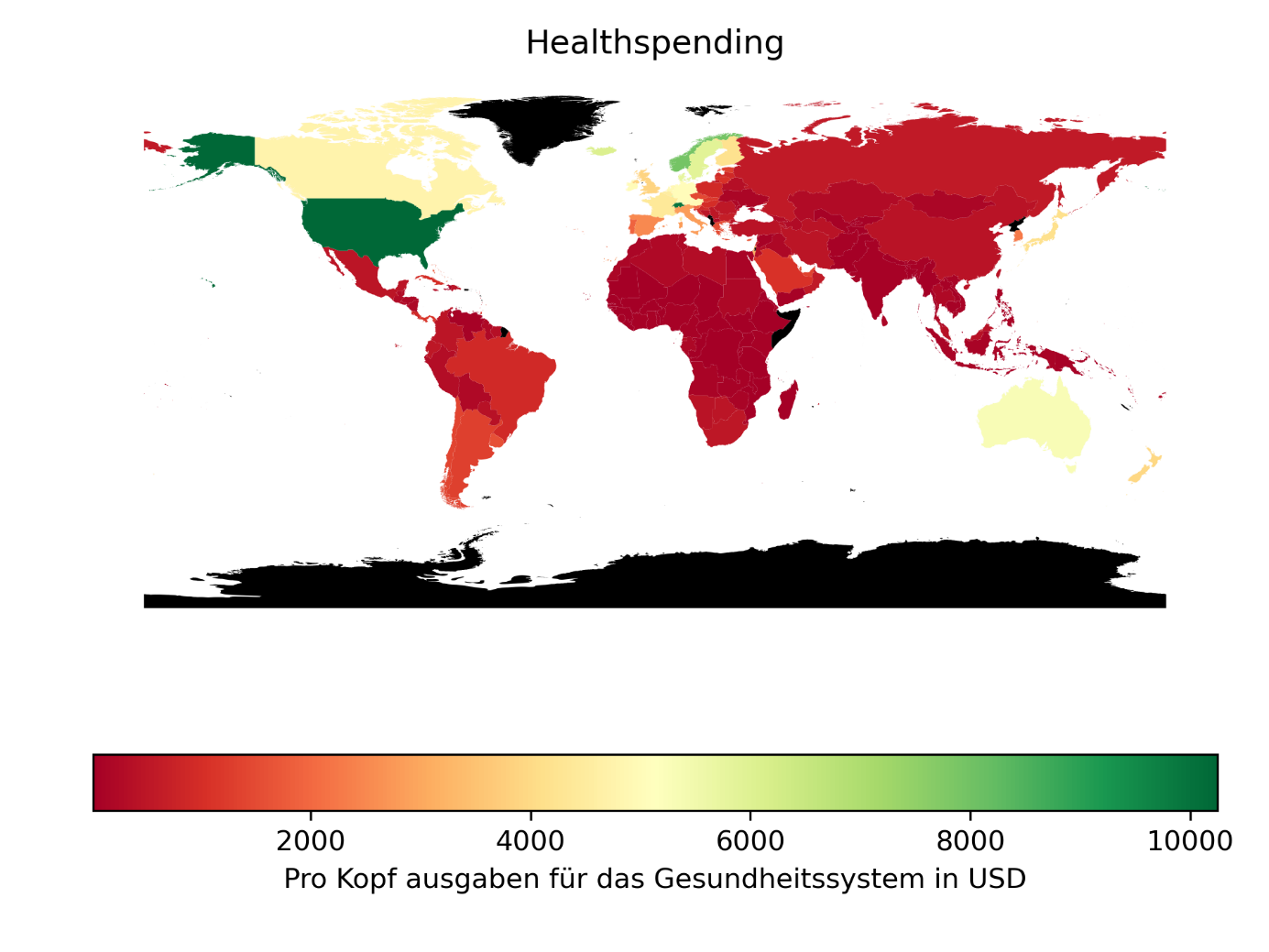
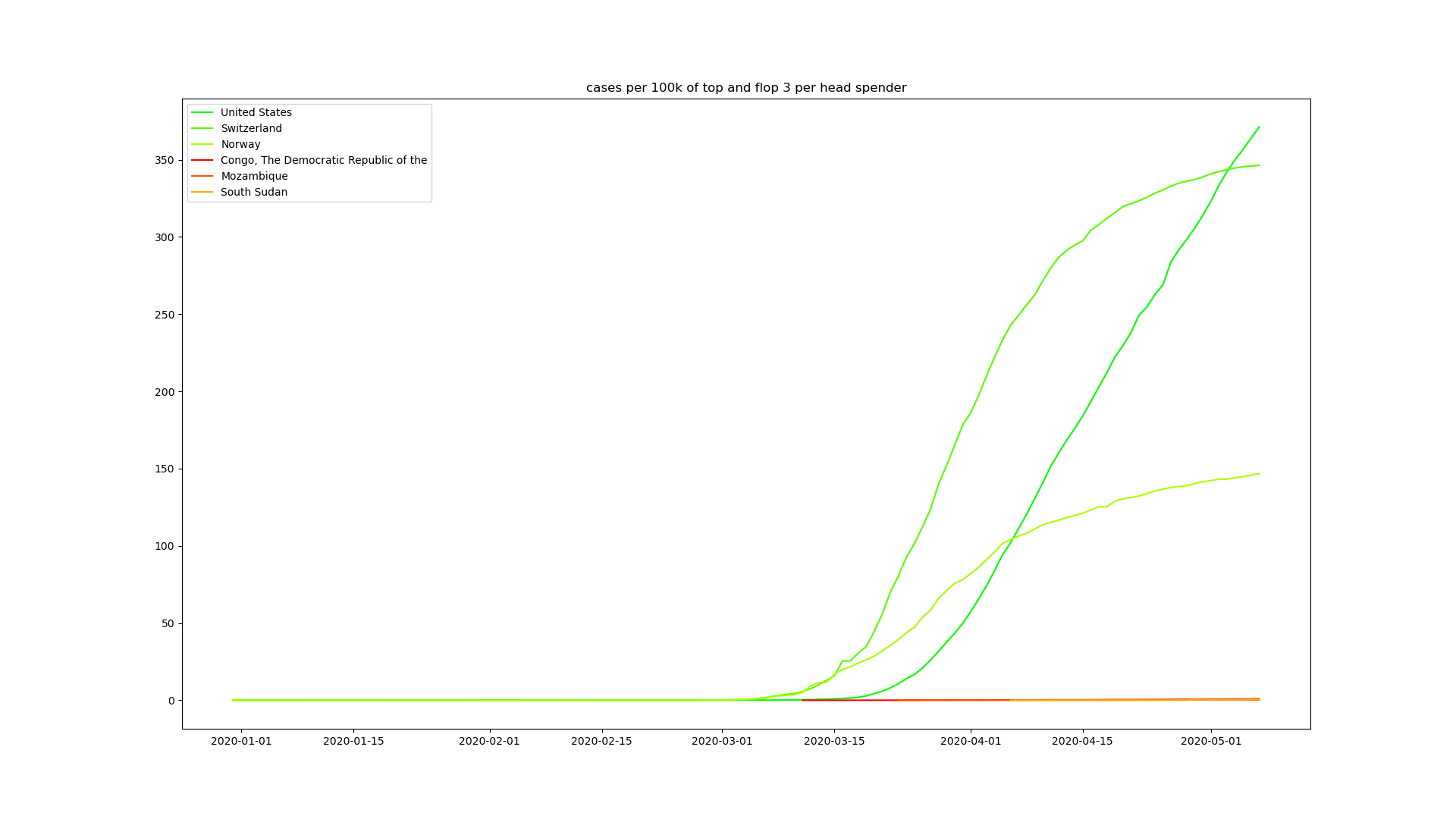
    loadingStr += "]"

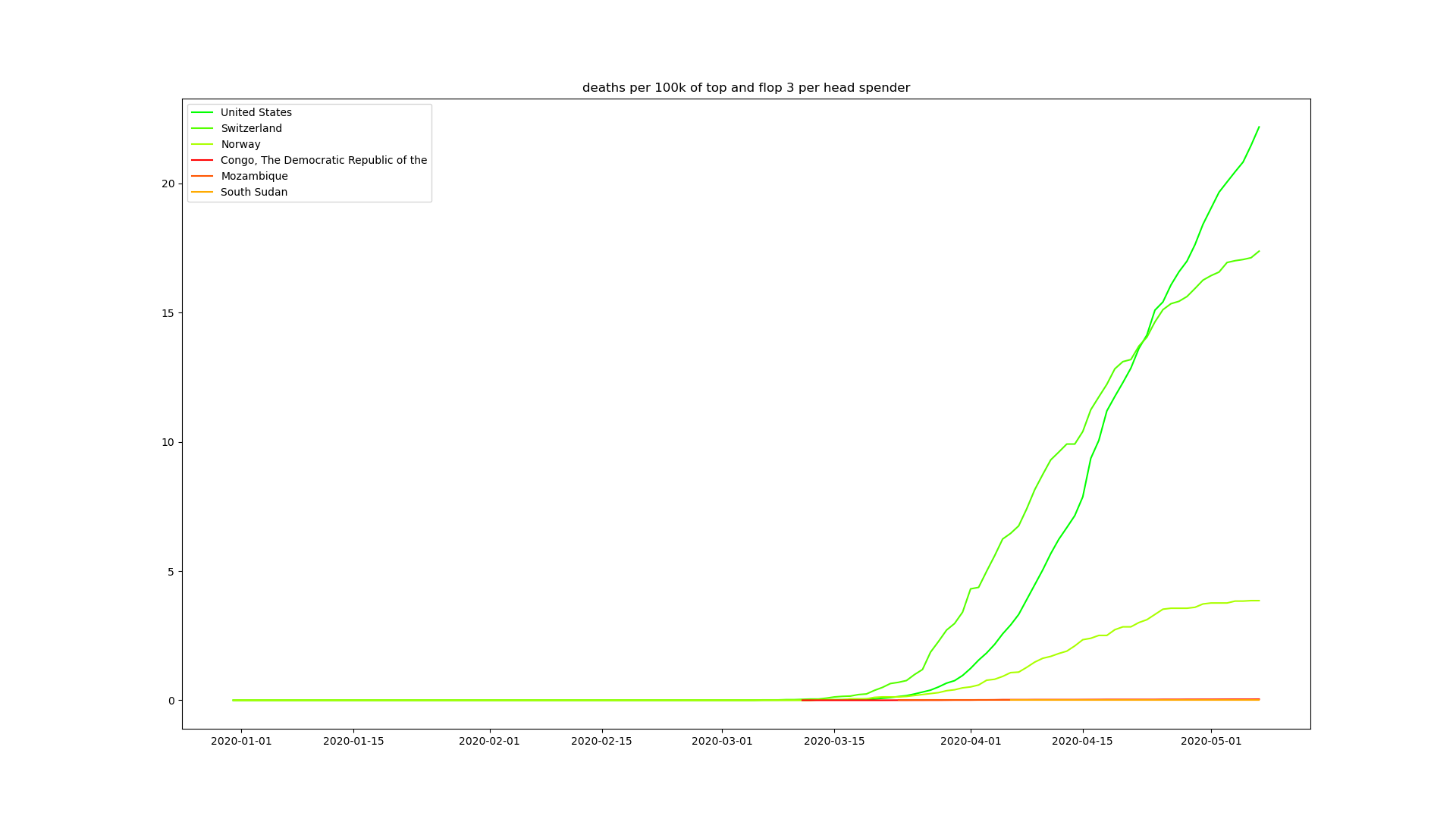
    print(loadingStr)

    print("Finished: "+str(progress)+"/"+str(maxLength))

    print(message)

**Abbildungen:**

**Gesundheitsausgaben**

****

**Fälle + Interesse**

(siehe Dateianhang)

**Gini Koeffizienten**

(siehe Dateianhang)

**GIFs**

(siehe Dateianhang)