Final Presentation – *stayhome*

# Epidemic Datathon

# Program

1. **Introduction**
2. **Data**
   a) Preprocessing
   b) Processing
   c) Post-processing
3. **Analysis**
   a) General remarks
   b) Model
      i. Confirmed / Recovered
      ii. Deaths
   c) Performance
4. **Outcome**
   a) Future remarks

# Program

1. **Introduction**
2. **Data**
   a) Preprocessing
   b) Processing
   c) Post-processing
3. **Analysis**
   a) General remarks
   b) Model
      i. Confirmed / Recovered
      ii. Deaths
   c) Performance
4. **Outcome**
   a) Future remarks

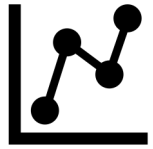
# Introduction

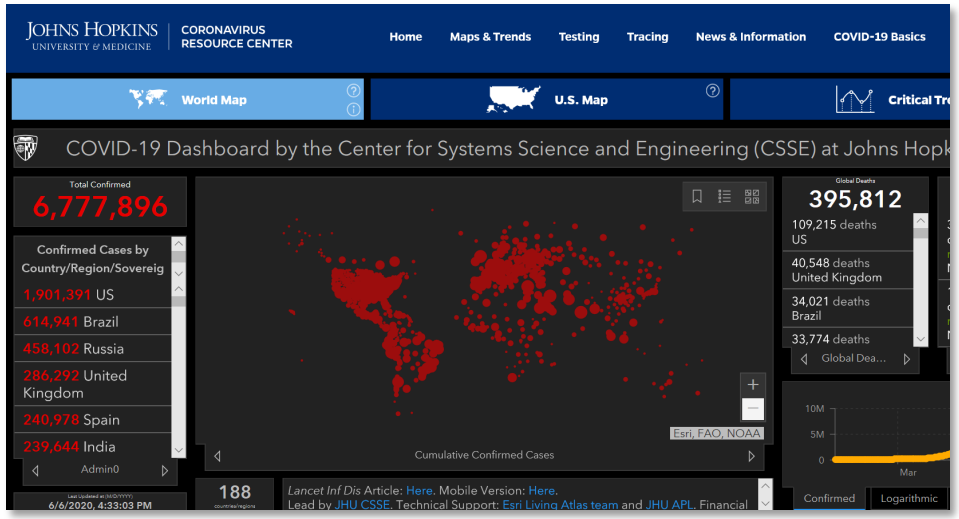
| | |
|---|---|
| **General** | • Global Virus → Pandemic<br>• Time delayed |
| **Geography** | • Tackled in regions, but mostly in countries |
| **Data** | • Collected from health institutes<br>• Datathon based on J.H. University |
| **Importance regional** | • Adapted decision making |
| **Importance global** | • Foresee trends, international studies, international solutions |

# Program

1. **Introduction**
2. **Data**
    a) Collection
    b) Processing
    c) Post-processing
3. **Analysis**
    a) General remarks
    b) Model
        i. Confirmed / Recovered
        ii. Deaths
    c) Performance
4. **Outcome**
    a) Future remarks

# Data – *Collection*






*https://coronavirus.jhu.edu/map.html, last visited 06.06.2020*

**Collection**

- Data is collected by J.H. University

**Accessability**

- Free
- In .csv format

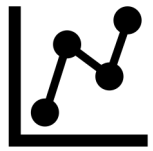
**Cleaning**

- Assumption of “Correctness”
- Not really necessary
- Except for adaptions in publication format of J.H. University

**Sorted?**

- Alphabetically sorted
- Country/region wise → 266 different regions in total

# Data – *Processing*



**Programming Language**

- Python 3

**Read In**

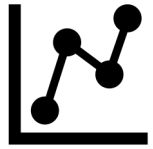
- **Pandas Data Frames** → similar to «dictionary»
- Conversion to *floats*

```
class Data:
    def __init__(self, confirmedFile = None, deathsFile = None,recoveredFile = None,
        self.confirmedFile = str(confirmedFile)
        self.deathsFile = str(deathsFile)
        self.recoveredFile = str(recoveredFile)
        #confirmed cases.
        self.df = self.initialize(self.confirmedFile)
        #death cases.
        self.df_deaths = self.initialize(self.deathsFile)
        #recovered cases.
        self.df_recovered = self.initialize(self.recoveredFile)
```

```
def initialize(self,File = None):
    dataframes = []
    for f in glob.glob(str(File)):
        dataframes.append(pd.read_csv(f))
    df = pd.concat(dataframes)
    df = df.drop(columns=['Lat', 'Long'])
    return df
```

«Cleaning»

# Data – *Postprocessing*



**Desired Output**

**Read Out**

- .csv files
- Specific naming

- Pandas Data Frames → similar to «dictionary»

```
n = 2
t2_prediction = pd.DataFrame(data={'Province/State': newPrediction.df['Province/State'],
                                   'Country': newPrediction.df['Country/Region'],
                                   'Target/Date': self.date[self.length_dates+n-1],
                                   'N': confirmed_and_predicted[:,newPrediction.length_dates-1+n],
                                   'D': deaths_and_predicted[:,newPrediction.length_dates-1+n],
                                   'R': recovered_and_predicted_rearranged[:,newPrediction.length_dates-1+n]})

t2_prediction.to_csv(path2, index=False)
```

# Program

1. **Introduction**
2. **Data**
   a) Preprocessing
   b) Processing
   c) Post-processing
3. **Analysis**
   a) General remarks
   b) Model
      i. Confirmed / Recovered
      ii. Deaths
   c) Performance
4. **Outcome**
   a) Future remarks

# Analysis – *General Remarks*

**Not Perfect Data**

- Insufficient testing
- Time delay
- New method of counting
- Dishonesty
- ….

**Decision for model**

- Overview: Polyfit

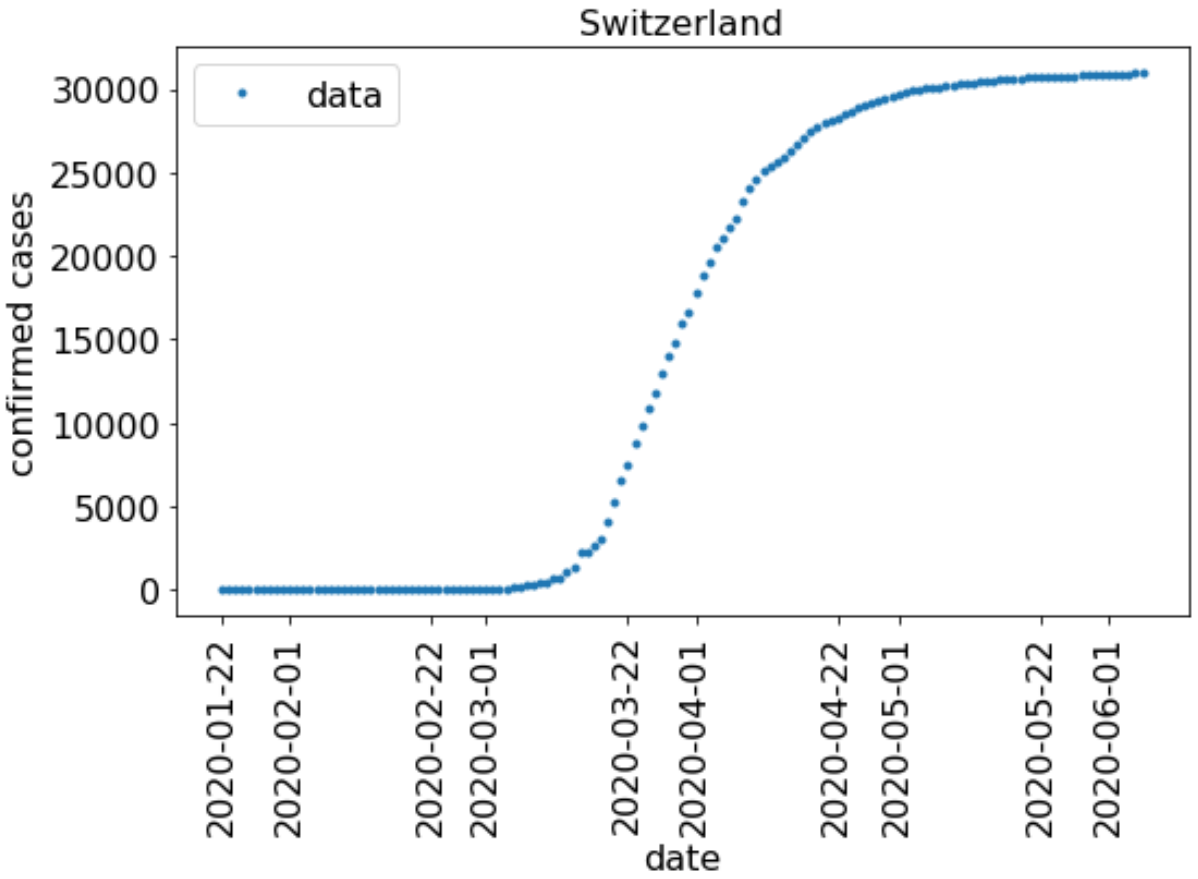
→ Not so good results

- Base Model: Exponential Fit
- Advanced Model for *D*

```
class Prediction(Data):
    def __init__(self,confirmedFile, deathsFile, recoveredFile, PredictionLength):
        super(Prediction, self).__init__(confirmedFile, deathsFile, recoveredFile, PredictionLength)
```

# Analysis – *Confirmed/Recovered*

**6 Step plan**

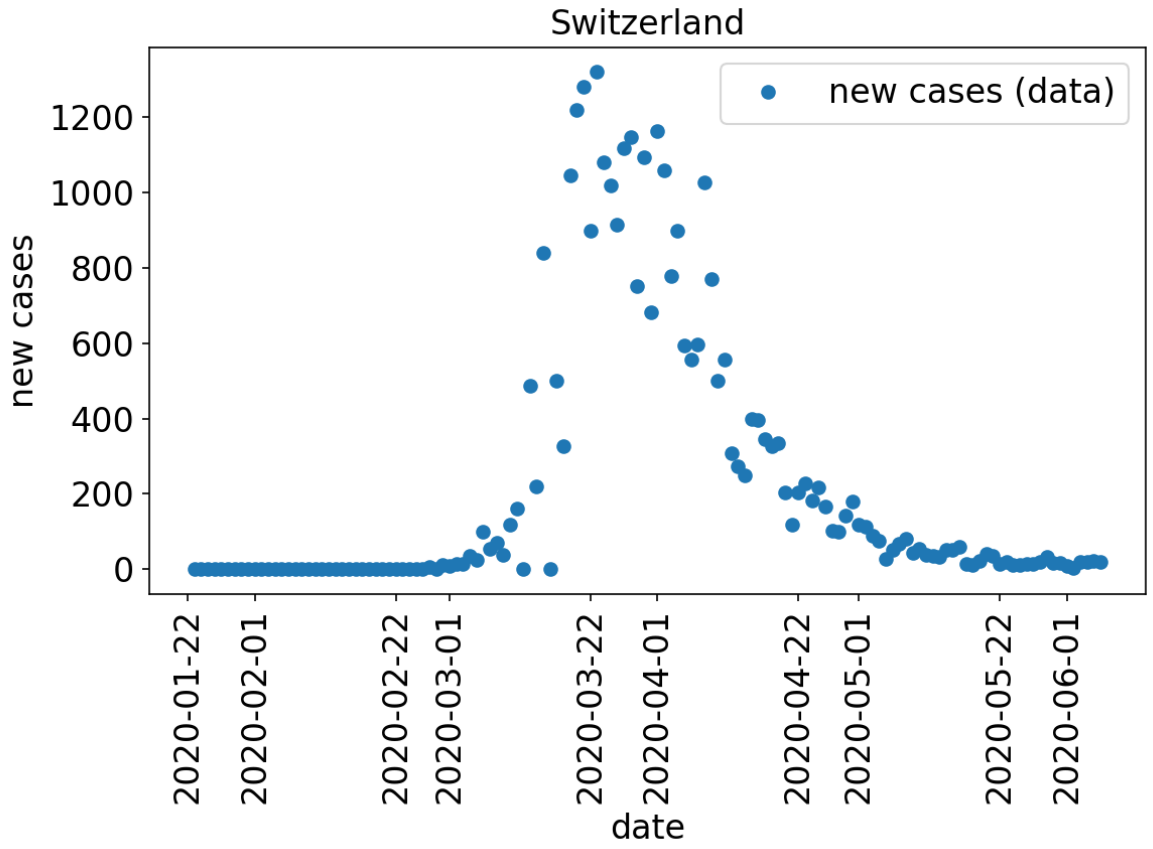



1. Compute daily new cases from number of confirmed cases

# Analysis – *Confirmed/Recovered*

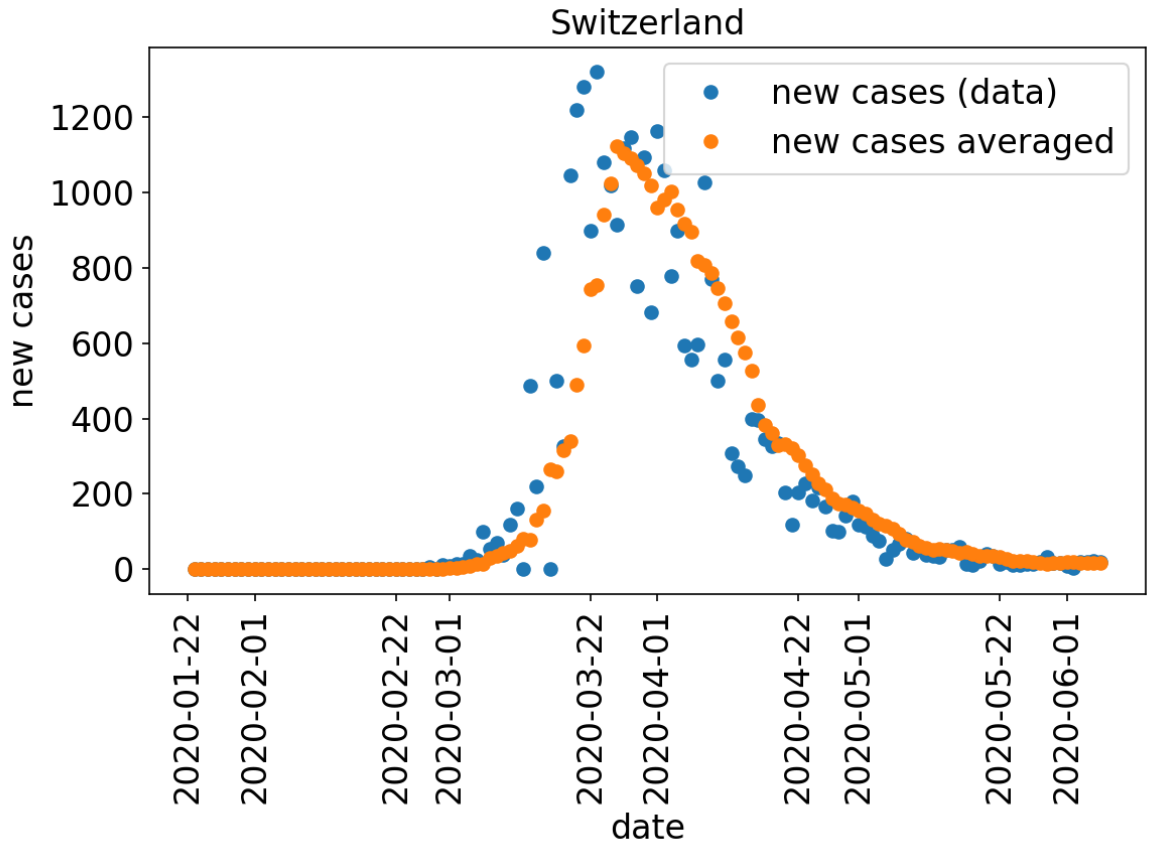
**6 Step plan**

1. Compute daily new cases from number of confirmed cases



# Analysis – *Confirmed/Recovered*

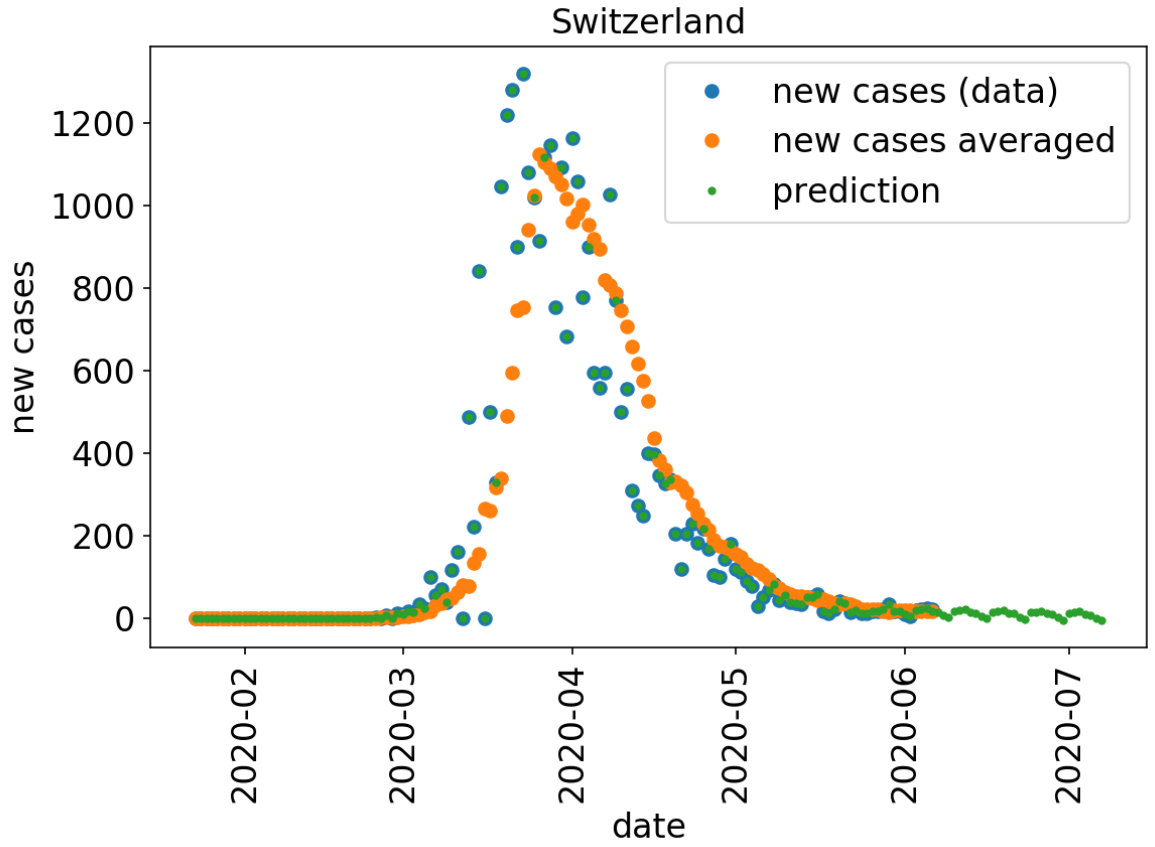
**6 Step plan**




1. Compute daily new cases from number of confirmed cases
2. Calculate average of new cases
3. Predict following 30 days with an exponential fit of the 5 most recent data points averaged

# Analysis – *Confirmed/Recovered*

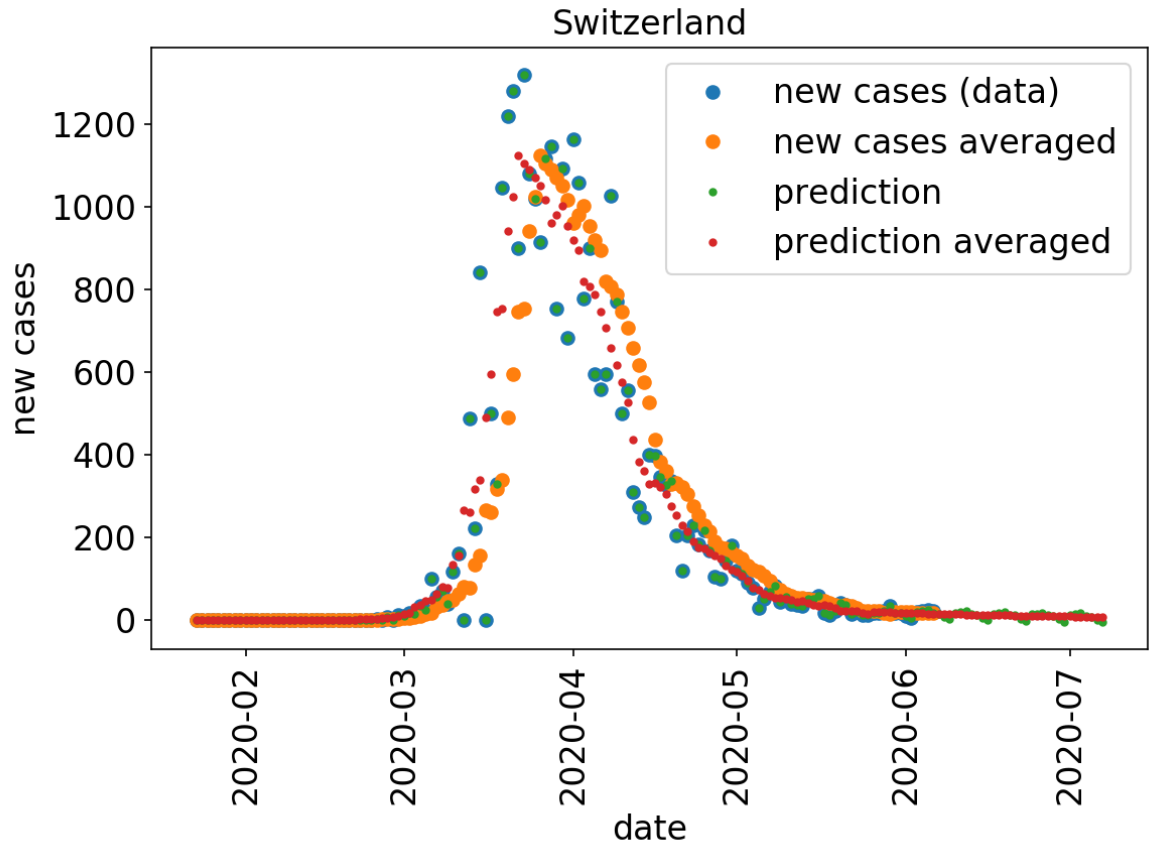
**6 Step plan**




1. Compute daily new cases from number of confirmed cases
2. Calculate average of new cases
3. Predict following 30 days with an exponential fit of the 5 most recent data points averaged
4. Calculate backwards from the average to daily numbers

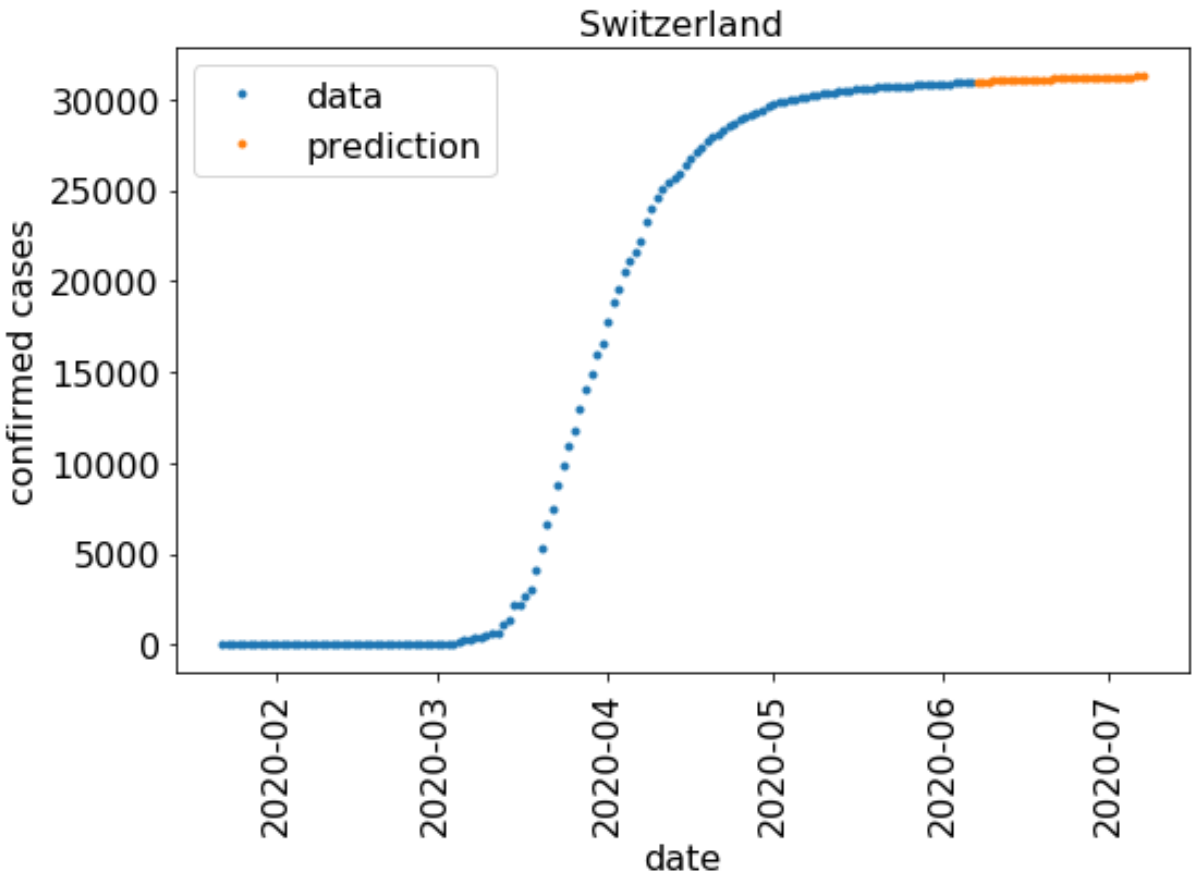
# Analysis – *Confirmed/Recovered*

**6 Step plan**




1. Compute daily new cases from number of confirmed cases
2. Calculate average of new cases
3. Predict following 30 days with an exponential fit of the 5 most recent data points averaged
4. Calculate backwards from the average to daily numbers
5. Compute a center average of the new cases and the predicted new cases

# Analysis – *Confirmed/Recovered*

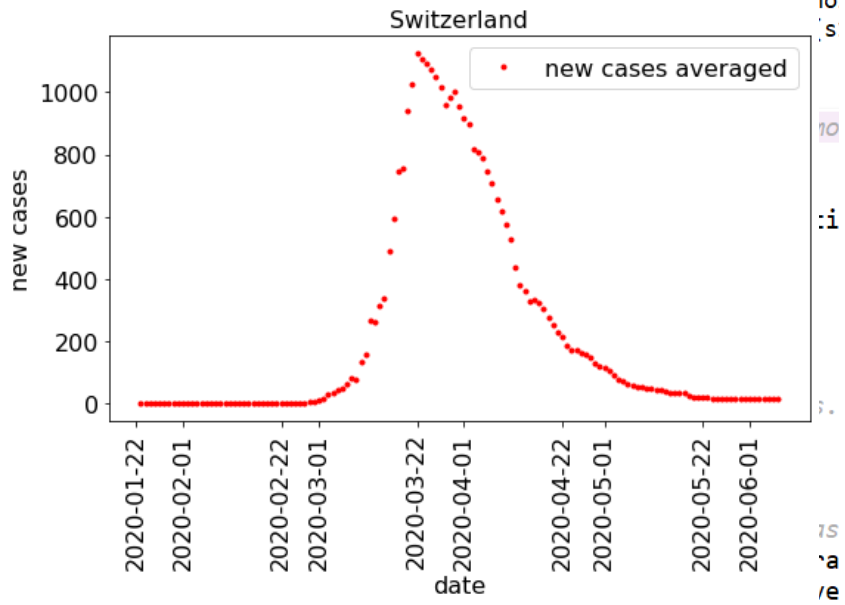


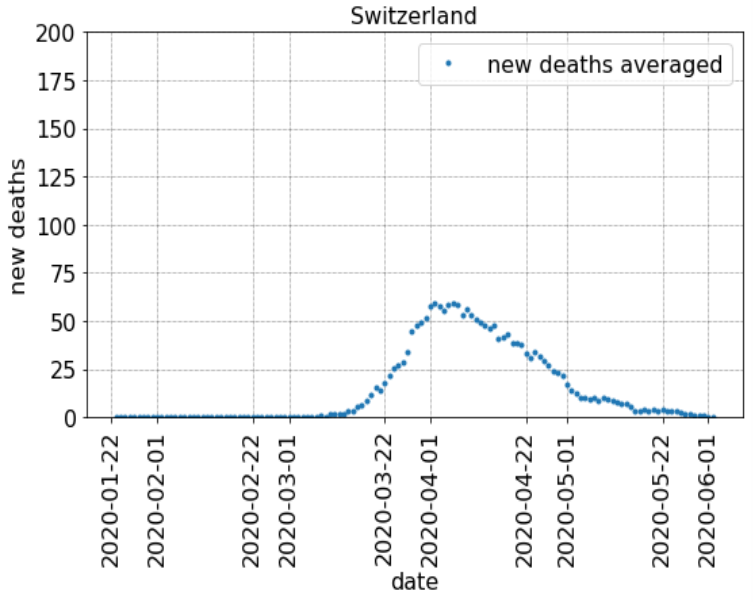

**6 Step plan**

1. Compute daily new cases from number of confirmed cases
2. Calculate average of new cases
3. Predict following 30 days with an exponential fit of the 5 most recent data points averaged
4. Calculate backwards from the average to daily numbers
5. Compute a center average of the new cases and the predicted new cases
6. Sum the predicted new cases center averaged numbers to the confirmed predicted numbers

# Analysis – *Deaths*

```
# Set the noise level.
self.noise_level=2
# Test time shifts in days.
self.delta=np.arange(1.0,20.0,1)
# Test mortailities.
self.mortalities=np.arange(0.001,0.15,0.001)
# Define the prior in data space. ----------
```

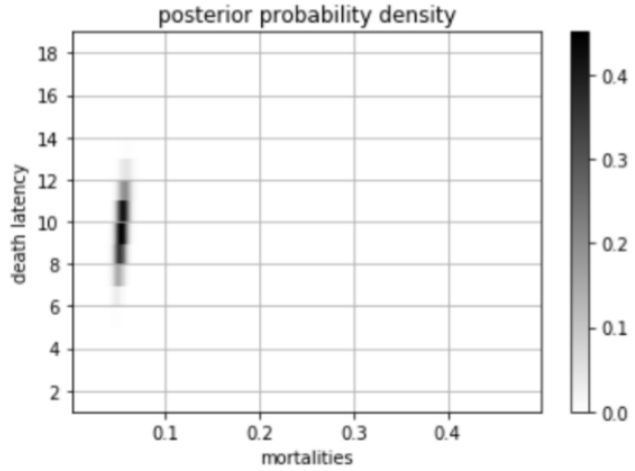


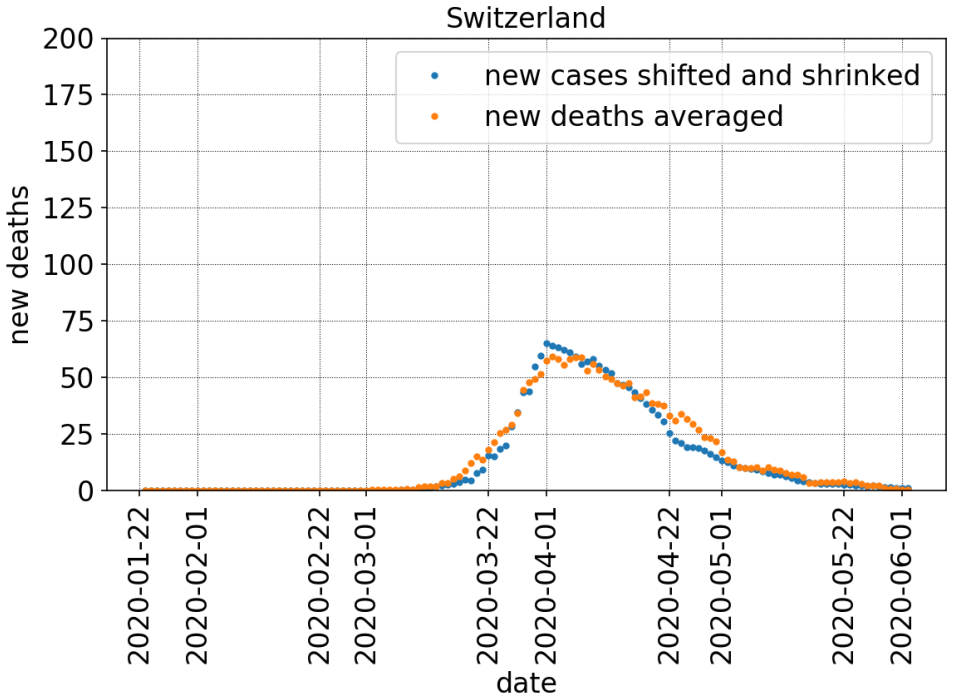




```
-death_latency]*mortality
# Evaluate posterior.
p[n,m]=self.prior_data(s,self.new_deaths_averaged,self.noise_level)
death_latency_index, mortality_index = np.where(p == np.amax(p))
self.death_latency, self.mortality = int(self.delta[death_latency_index]), self.mortalities[mortality_index]
```

- Assumption: Connection between C – D
- Grid search (inverse theory tool)
  - Optimize death latency and mortality
- Time shift and shrink averaged new *C* cases in order to match the averaged new *D*
- Advantage: New development in *C* will translate in the future to *D*

# Analysis – *Deaths*




- Assumption: Connection between C – D
- Grid search (inverse theory tool)
  - Optimize death latency and mortality
- Time shift and shrink averaged new *C* cases in order to match the averaged new *D*
- Advantage: New development in *C* will translate in the future to *D*

# Analysis – *Performance*

**Time**

- Whole process of running the program: ca. 4 sec.
  - Data read in: 0.5 sec
  - *C*/*R*: 1.0 sec
  - *D*: 1.5 sec
  - Data output: 0.5 sec

**Accuracy**

- Good Accuracy for short term C
- Good long term accuracy for *D* and *R*
- Topping the leaderboad several times

**Possible optimization**

- Use gathered data to optimize parameters such as average length and fit length
- Test if our death prediction method is better than an exponential fit

# Program

1. **Introduction**
2. **Data**
   a) Preprocessing
   b) Processing
   c) Post-processing
3. **Analysis**
   a) General remarks
   b) Model
      i. Confirmed / Recovered
      ii. Deaths
   c) Performance
4. **Outcome**
   a) Future remarks

# Outcome – *Future Remarks*

**Thank you for your attention.**

**Questions?**

**Quality of Data**

Poor Data Quality

1. Different counting methods
2. Dishonesty
3. No complete knowledge
4. ….

**Leading to**

→ No modelling of disease
→ Modelling of datapoints

**For future datathons**

→ Maybe problems with *Clean* and *Correct* Data