The Discovery of Handwashing

Einführung in die Programmierung für Nicht-Informatiker

Alexander Trey
Marie-Christine Schmitz
Jule Hansen
Lillian Fitzner
Zoe Giese

Übersicht

History

Part I

Marie-Christine Schmitz

Alexander Trey

Part II

Zoe Giese

Lillian Fitzner

Jule Hansen

History

Ignaz Semmelweis

- 1818 - 1865
- Arzt/Gynäkologe
- “Pionier der Hygiene”

- Patienten starben an Kindbettfieber
- 1847 verpflichtende Händedesinfektion

Quelle: https://de.wikipedia.org/wiki/Datei:Ignaz_Semmelweis_1860.jpg

Part I

Marie-Christine Schmitz

```python
# Imports
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
import seaborn as sb

# Read Yearly Deaths by Clinic Dataset
data: pd.DataFrame = pd.read_csv("yearly_deaths_by_clinic.csv")
#print(data)

# Creating a Table with Pandas
data = {
    "year": [
        1841, 1842, 1843, 1844, 1845, 1846,
        1841, 1842, 1843, 1844, 1845, 1846
    ],
    "births": [
        3036, 3287, 3060, 3157, 3492, 4010,
        2442, 2659, 2739, 2956, 3241, 3754
    ],
    "deaths": [
        237, 518, 274, 260, 241, 459,
        86, 202, 164, 68, 66, 105
    ],
    "clinic": [
        "clinic 1", "clinic 1", "clinic 1", "clinic 1", "clinic 1", "clinic 1",
        "clinic 2", "clinic 2", "clinic 2", "clinic 2", "clinic 2", "clinic 2"
    ]
}
```

	year	births	deaths	clinic
0	1841	3036	237	clinic 1
1	1842	3287	518	clinic 1
2	1843	3060	274	clinic 1
3	1844	3157	260	clinic 1
4	1845	3492	241	clinic 1
5	1846	4010	459	clinic 1
6	1841	2442	86	clinic 2
7	1842	2659	202	clinic 2
8	1843	2739	164	clinic 2
9	1844	2956	68	clinic 2
10	1845	3241	66	clinic 2
11	1846	3754	105	clinic 2

```python
# Exploring the Dataset
print("\nClinic information:")
print(clinic.info())
print("\nClinic shape:")
print(clinic.shape)
print("\nClinic size:")
print(clinic.size)
```

```
Clinic information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12 entries, 0 to 11
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   year    12 non-null     int64
 1   births  12 non-null     int64
 2   deaths  12 non-null     int64
 3   clinic  12 non-null     object
dtypes: int64(3), object(1)
memory usage: 512.0+ bytes
None

Clinic shape:
(12, 4)

Clinic size:
48
```

```python
# Deaths per Clinic
deaths_per_clinic = clinic.groupby("clinic")["deaths"].sum()
# Link: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html
print(deaths_per_clinic)
```

```
clinic
clinic 1    1989
clinic 2     691
Name: deaths, dtype: int64
```

```python
#  Calculate proportions of death
prop = clinic.groupby("clinic")[["births", "deaths"]].sum()
prop["death_proportion"] = (prop["deaths"] / prop["births"]) * 100

prop
```

clinic	births	deaths	death_proportion
clinic 1	20042	1989	9.924159
clinic 2	17791	691	3.883986

proportional mehr Tote in Klinik 1

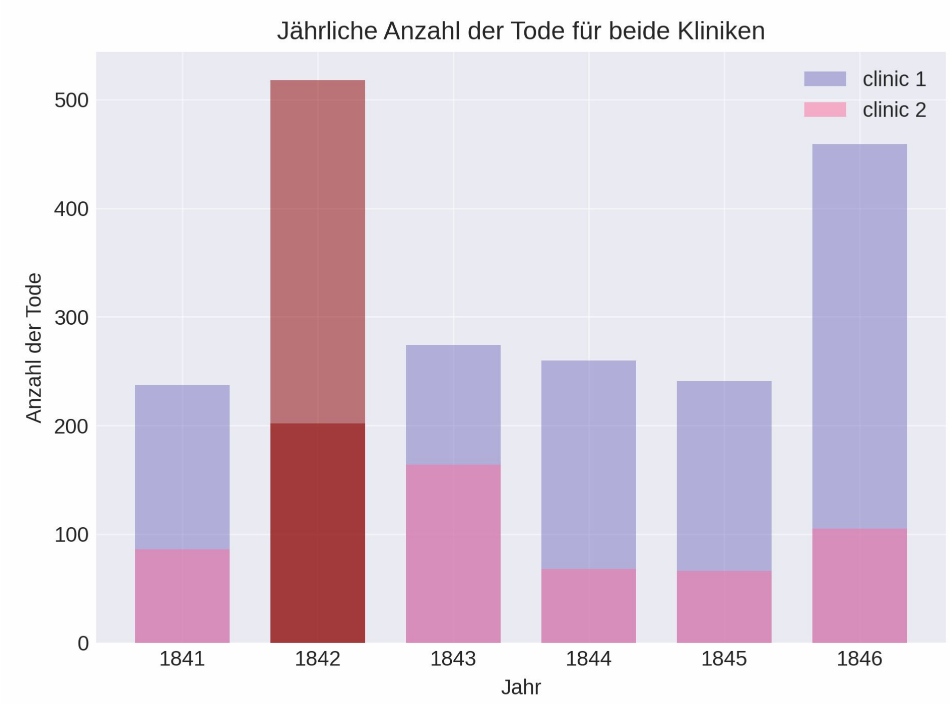
Alexander Trey

```
# Split dataset into clinics
clinic_1 = dataset[dataset['clinic'] == 'clinic 1']
clinic_2 = dataset[dataset['clinic'] == 'clinic 2']

# Assert only one unique entry is present
print(f'Unique clinics in clinic_1 set: {clinic_1["clinic"].unique()}')
print(f'Unique clinics in clinic_2 set: {clinic_2["clinic"].unique()}')
```

```
Unique clinics in clinic_1 set: ['clinic 1']
Unique clinics in clinic_2 set: ['clinic 2']
```

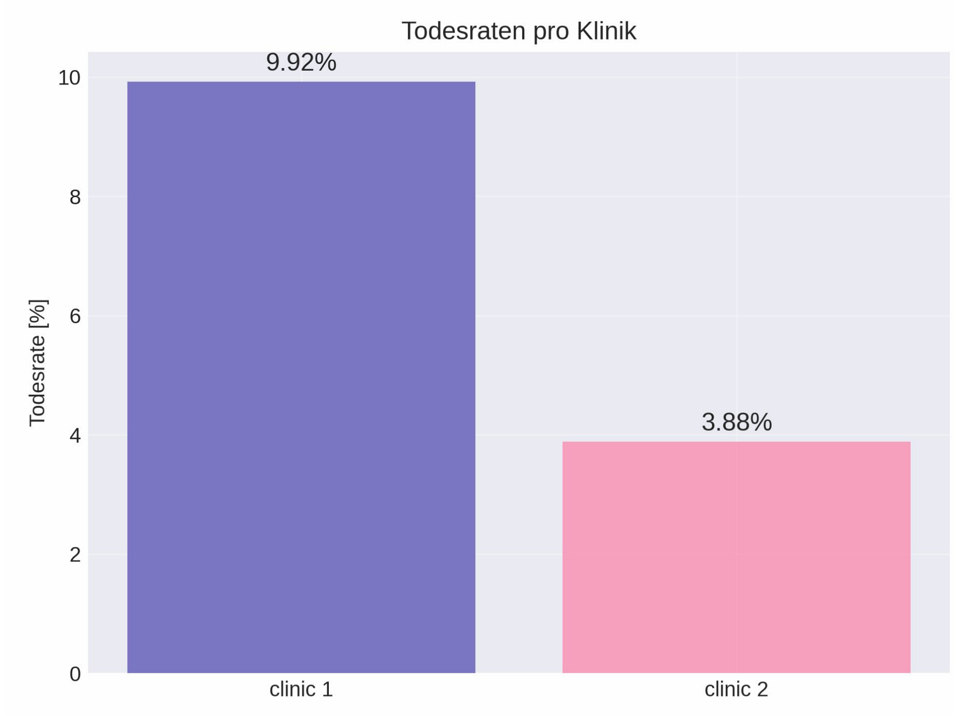
Anzahl der Tode über Jahre

```
# Create figure for deaths vs year for each clinic
fig, ax = plt.subplots()

# Base colors for dhaiz matplotlib stylesheet
colors = {'clinic 1': '#7A76C2',
          'clinic 2': '#ff6e9c98'}
for clinic in [clinic_1, clinic_2]:
    # Save clinic name to variable for color selection
    clinic_name = clinic['clinic'].unique()[0]
    # Save idx of maxmimum deaths to varaible
    idx_max_deaths = clinic['deaths'].idxmax()

    # Highlight peak year in red, others in clinic color
    for idx, row in clinic.iterrows():
        # Select color based on clinic or if its the year of max deaths
        color = 'darkred' if idx == idx_max_deaths else colors[clinic_name]
        # Select label based on clinic name
        label = clinic_name if idx == clinic.index[0] else ""
        # Plot yearly deaths as bar graph
        ax.bar(row['year'], row['deaths'],
               alpha=0.5, color=color, width=0.7,
               label=label)

# Axes settings
ax.set_xlabel('Jahr')
ax.set_ylabel('Anzahl der Tode')
ax.set_title('Jährliche Anzahl der Tode für beide Kliniken')
ax.legend()
ax.grid(True, alpha=0.5)

# Show and save to file
plt.tight_layout()
plt.savefig('yearly_deaths.png', dpi=300)
plt.show()
```



Mortalitätsraten der Klinken

```
fig, ax = plt.subplots()
colors = {'clinic 1': '#7A76C2',
         'clinic 2': '#ff6e9c98'}

# Empty list to store bars (Rectangle elements)
bars = []
mortality_rates = []

# Split dataset by unique clinics (works for sets with more than )
for clinic_name in dataset['clinic'].unique():
    clinic_df = dataset[dataset['clinic'] == clinic_name]
    total_deaths = clinic_df['deaths'].sum()
    total_births = clinic_df['births'].sum()

    # Calculation of proportion of deaths
    mortality_rate = (total_deaths / total_births) * 100
    mortality_rates.append(mortality_rate)
    # Plot proportions of death with bars
    bar = ax.bar(clinic_name, mortality_rate)
    bars.append(bar)

# Add text for accurate display of deathrate to top of bars
for container, rate in zip(bars, mortality_rates):
    for bar in container:
        ax.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.1,
                f'{rate:.2f}%', ha='center', va='bottom', fontsize=12)

# Axes settings
ax.set_ylabel('Mortalitätsrate [%]')
ax.set_title('Mortalitätsrate pro Klinik')
ax.grid(True, alpha=0.3)

# Figure settings
fig.tight_layout()
fig.savefig('deathrates.png', dpi=300)
```



Students T-Test

```
# Statistical Analysis: Proportion Tests for Clinic Comparison

# Calculate mortality rates for each clinic by year
clinic_1_rates = clinic_1['deaths'] / clinic_1['births']
clinic_2_rates = clinic_2['deaths'] / clinic_2['births']

print("=== Statistical Analysis: Comparing Clinic Mortality Rates ===\n")

# Two-sample t-test comparing mortality rates between clinics
alpha = 0.05
t_stat, p_value = stats.ttest_ind(clinic_1_rates, clinic_2_rates)

print(f"Two-sample t-test results:")
print(f"  t-statistic: {t_stat:.4f}")
print(f"  p-value: {p_value:.6f}")
print(f"  Significance level: α = 0.05")
if p_value < alpha:
    print("  Result: Statistically significant")
else:
    print("  Result: Not statistically significant")

# Descriptive statistics
print(f"\nDescriptive Statistics:")
print(f"  Clinic 1 - Mean mortality rate: {clinic_1_rates.mean():.4f}",
      f"({clinic_1_rates.std():.4f})")
print(f"  Clinic 2 - Mean mortality rate: {clinic_2_rates.mean():.4f}",
      f"{clinic_2_rates.std():.4f})")
print(f"  Difference: {clinic_1_rates.mean() - clinic_2_rates.mean():.4f}")
```

T-Tes:

- p-Wert von 0,005
- Nullhypothese wird verworfen

```
=== Statistical Analysis: Comparing Clinic Mortality Rates ===

Two-sample t-test results:
  t-statistic: 3.5797
  p-value: 0.005014
  Significance level: α = 0.05
  Result: Statistically significant

Descriptive Statistics:
  Clinic 1 - Mean mortality rate: 0.0985 (0.0328)
  Clinic 2 - Mean mortality rate: 0.0404 0.0225)
  Difference: 0.0581
```

Monte-Carlo Simulation - Setup

```python
# Monte Carlo Permutation Test for Clinic Mortality Rates

# Set random seed for reproducibility
np.random.seed(42)

# Calculate observed difference in mean mortality rates
observed_diff = clinic_1_rates.mean() - clinic_2_rates.mean()

print("=== Monte Carlo Permutation Test ===\n")
print(f"Observed difference in mean mortality rates: {observed_diff:.4f}")
print(f"(Clinic 1 mean - Clinic 2 mean)")

# Combine all mortality rates for permutation
all_rates = np.concatenate([clinic_1_rates, clinic_2_rates])
n_clinic_1 = len(clinic_1_rates)
n_clinic_2 = len(clinic_2_rates)

# Number of permutations
n_permutations = 10000

# Store permuted differences
permuted_differences = []

print(f"\nRunning {n_permutations:,} permutations...")
for i in range(n_permutations):
    # Randomly shuffle the combined mortality rates
    shuffled_rates = np.random.permutation(all_rates)

    # Split back into two groups of original sizes
    perm_clinic_1 = shuffled_rates[:n_clinic_1]
    perm_clinic_2 = shuffled_rates[n_clinic_1:]

    # Calculate difference in means for this permutation
    perm_diff = perm_clinic_1.mean() - perm_clinic_2.mean()
    permuted_differences.append(perm_diff)

permuted_differences = np.array(permuted_differences)
```

- Permutationstest
- Nullhypothese: Beide Kliniken stammen aus der gleichen Verteilung
- Mortalitätsraten werden zufällig neu sortiert (Permutation)
    - Zuordnung zu Klinik über Index
- Differenz der Mittelwerte nach Permutation werden gespeichert und mit Differenz aus orig. Datensatz verglichen

Monte-Carlo Simulation - Auswertung

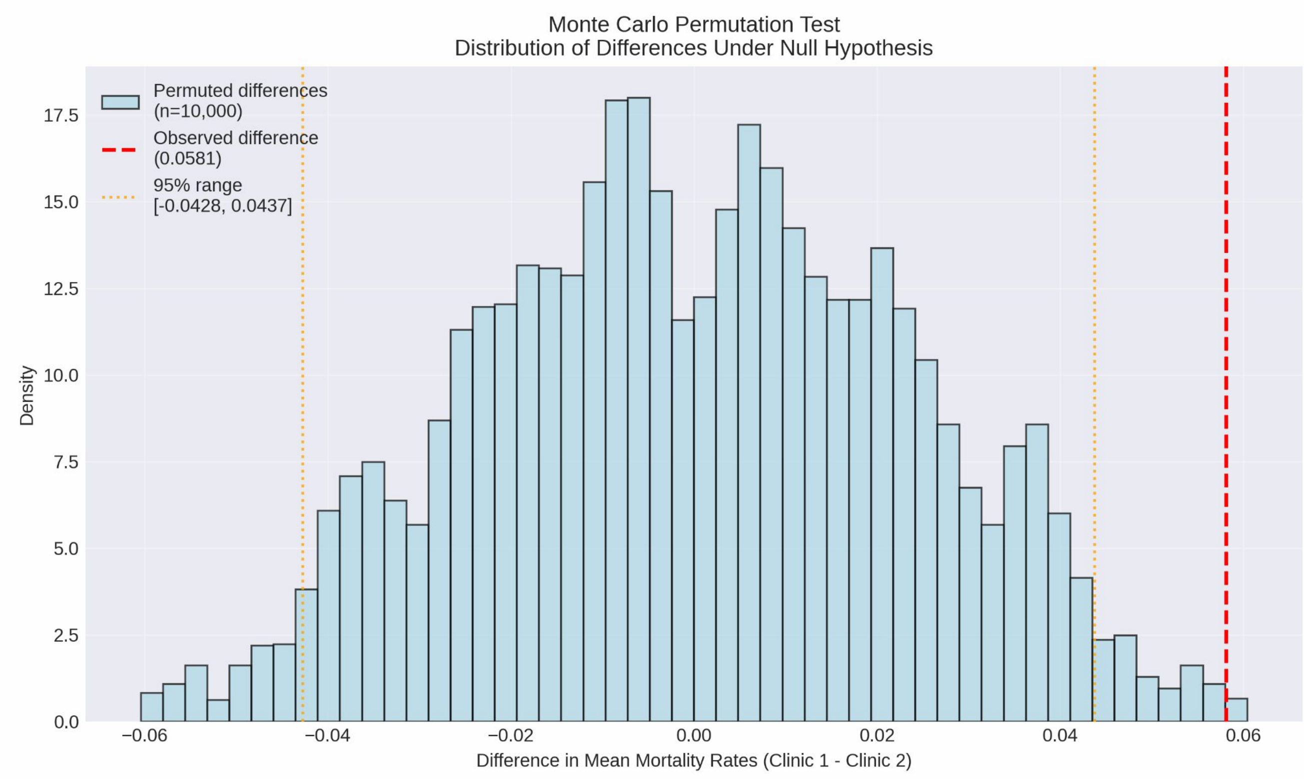
```python
# Calculate p-value (two-tailed test)
# Count how many permuted differences are as extreme or more extreme than observed
p_value_mc = np.sum(np.abs(permuted_differences) >= np.abs(observed_diff)) / n_permutations
print(f"\nMonte Carlo Results:")
print(f"  Permuted differences - Mean: {permuted_differences.mean():.6f}")
print(f"  Permuted differences - Std: {permuted_differences.std():.6f}")
print(f"  Monte Carlo p-value (two-tailed): {p_value_mc:.4f}")
print(f"  Comparison with t-test p-value: {p_value:.6f}")

# Statistical interpretation
print(f"\nInterpretation:")
if p_value_mc < 0.05:
    print(f"  The observed difference ({observed_diff:.4f}) is statistically significant")
    print(f"  at α = 0.05 level (Monte Carlo p = {p_value_mc:.4f})")
    print(f"  We reject the null hypothesis of no difference between clinics")
else:
    print(f"  The observed difference ({observed_diff:.4f}) is not statistically significant")
    print(f"  at α = 0.05 level (Monte Carlo p = {p_value_mc:.4f})")
    print(f"  We fail to reject the null hypothesis")
print(f"  Only {np.sum(np.abs(permuted_differences) >= np.abs(observed_diff))}"
      f"out of {n_permutations:,} permutations")
print(f"  showed a difference as extreme or more extreme than what we observed.")
```

```
Monte Carlo Results:
  Permuted differences - Mean: 0.000098
  Permuted differences - Std: 0.023493
  Monte Carlo p-value (two-tailed): 0.0030
  Comparison with t-test p-value: 0.005014

Interpretation:
  The observed difference (0.0581) is statistically significant
  at α = 0.05 level (Monte Carlo p = 0.0030)
  We reject the null hypothesis of no difference between clinics
  Only 30out of 10,000 permutations
  showed a difference as extreme or more extreme than what we observed.
```

$$\text{p-Wert: } \frac{\text{Anzahl der Permutationen mit größerer Abweichung}}{\text{Gesamtanzahl der Permutationen}}$$

- Sehr kleiner P-Wert: 0,003
- Nullhypothese wird verworfen

Monte-Carlo Simulation - Auswertung



Part II

Zoe Giese

In [1]:

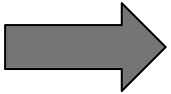
```
#Zoe Giese
import pandas as pd
import matplotlib.pyplot as plt

#read the monthly dataset
monthly = pd.read_csv("monthly_deaths.csv")

#explore monthly dataset via information
print("\nDataset information:")
print(monthly.info())

#explore monthly dataset via head
print("\nFirst 5 rows:")
print(monthly.head())

#explore monthly dataset via tail
print("\nLast 5 rows:")
print(monthly.tail())

#explore monthly dataset via shape
print("\nShape of dataset:")
print(monthly.shape)
```

```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   date    98 non-null     object
 1   births  98 non-null     int64
 2   deaths  98 non-null     int64
dtypes: int64(2), object(1)
memory usage: 2.4+ KB
None

First 5 rows:
         date  births  deaths
0  1841-01-01     254      37
1  1841-02-01     239      18
2  1841-03-01     277      12
3  1841-04-01     255       4
4  1841-05-01     255       2

Last 5 rows:
          date  births  deaths
93  1848-11-01     310       9
94  1848-12-01     373       5
95  1849-01-01     403       9
96  1849-02-01     389      12
97  1849-03-01     406      20

Shape of dataset:
(98, 3)
```

Zoe Giese

In [2]:

```
#Zoe Giese
#calculate proportions of deaths per month
monthly["proportion_deaths"] = monthly["deaths"] / monthly["births"]

#convert data column to datetime
monthly["date"] = pd.to_datetime(monthly["date"])
```



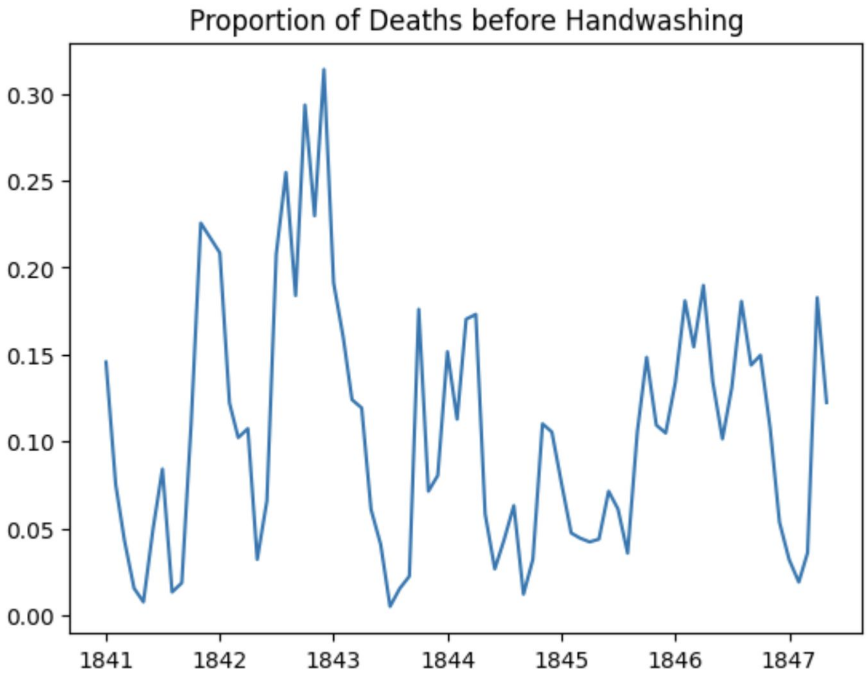
```
Dataset information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   date               98 non-null     datetime64[ns]
 1   births             98 non-null     int64
 2   deaths             98 non-null     int64
 3   proportion_deaths  98 non-null     float64
dtypes: datetime64[ns](1), float64(1), int64(2)
memory usage: 3.2 KB
None
```

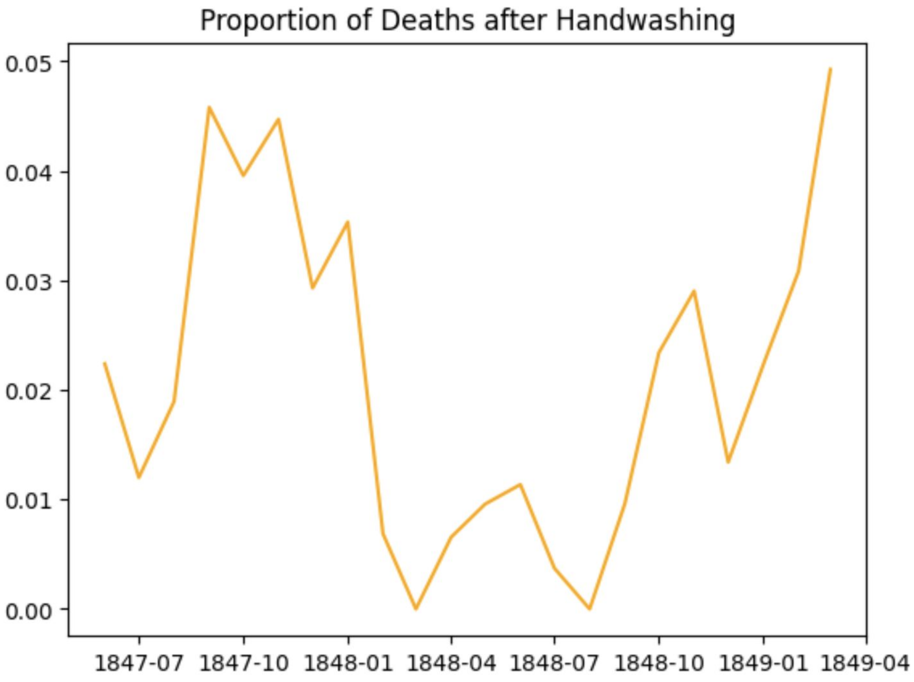
Lillian Fitzner

In [9]:
```
#Lillian Fitzner
#define start of handwashing

# monthly["date"] = pd.to_datetime(monthly["date"])
# plt.figure()
# plt.plot(monthly["date"], monthly["proportion_deaths"])
# plt.show()

start_handwashing = pd.to_datetime("1847-06-01")

#split dataset
before_hw = monthly[monthly["date"] < start_handwashing]
after_hw = monthly[monthly["date"] >= start_handwashing]

print("Before handwashing:", before_hw.shape)
print("After handwashing:", after_hw.shape)

#Plot before handwashing
plt.figure()
plt.plot(before_hw["date"], before_hw["proportion_deaths"])
plt.title("Proportion of Deaths before Handwashing")
plt.show()

#Plot after handwashing
plt.figure()
plt.plot(after_hw["date"], after_hw["proportion_deaths"], color="orange")
plt.title("Proportion of Deaths after Handwashing")
plt.show()
```

```
Before handwashing: (76, 4)
After handwashing: (22, 4)
```

Lillian Fitzner

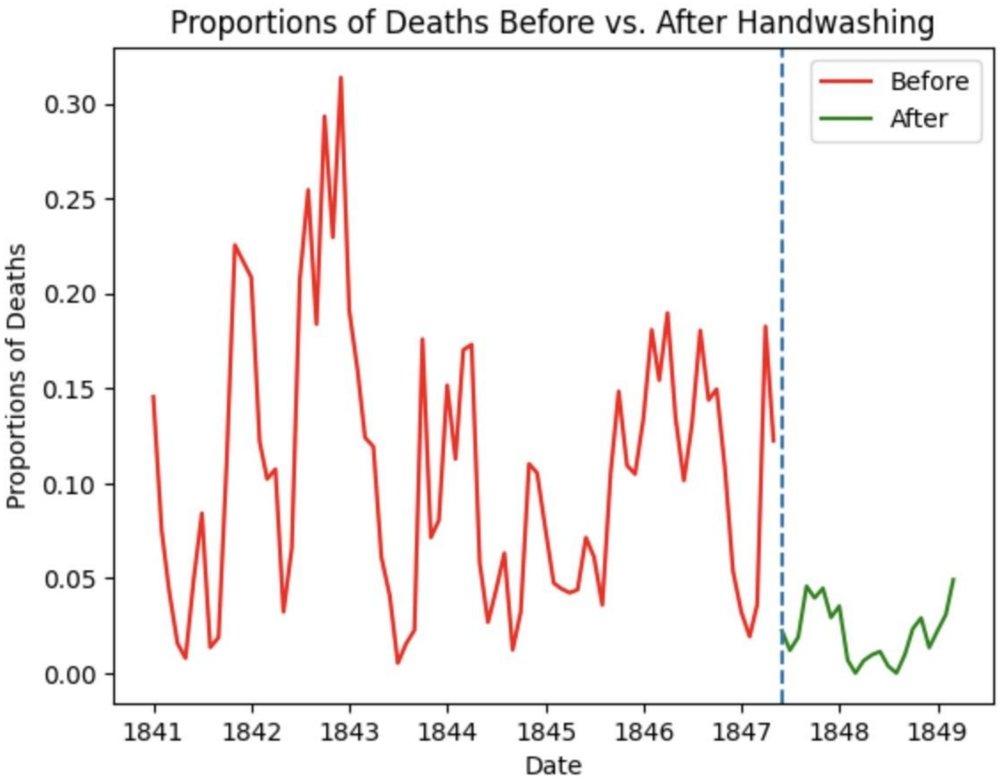




Jule Hansen

In [8]:
```
#Jule Hansen
#Combine the two plots into one plot
plt.figure()

#Plot before handwashing
plt.plot(before_hw["date"], before_hw["proportions_deaths"], label = "Before", color = "red")

#Plot after handwashing
plt.plot(after_hw["date"], after_hw["proportions_deaths"], label = "After", color = "green")

#Mark invention date
plt.axvline(start_handwashing, linestyle = "--")

#Labels and legend
plt.legend()
plt.title("Proportions of Deaths Before vs. After Handwashing")
plt.xlabel("Date")
plt.ylabel("Proportions of Deaths")

plt.show()
```

Jule Hansen



Proportions of Deaths Before vs. After Handwashing
Before
After
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Proportions of Deaths
1841
1842
1843
1844
1845
1846
1847
1848
1849
Date

Jule Hansen

In [9]:
```
#Jule Hansen
#How much did handwashing decrease the proportion of deaths by average
avg_before = before_hw["proportions_deaths"].mean()
avg_after = after_hw["proportions_deaths"].mean()

decrease = avg_before - avg_after
```

In [9]:
```
#Jule Hansen
#Report
print("Average proportions before handwashing:", round(avg_before, 4))
print("Average proportions after handwashing:", round(avg_after, 4))
print("Decrease in proportion due to handwashing:", round(decrease, 4))
```

```
Average proportions before handwashing: 0.105
Average proportions after handwashing: 0.0211
Decrease in proportion due to handwashing: 0.084
```

Vielen Dank für Ihre Aufmerksamkeit!