As Welf said, we can skip the data generation step....

1.3 Load the (generated) data and get an overview

In [ ]:
```
import pandas as pd

feature_names = ['n_50', 'n_150', 'n_300', 'd_50', 'd_150', 'd_300', 't_peak', 'x_5
df = pd.read_csv('../data/epidemic_process.csv', names=feature_names)
df.head()
```

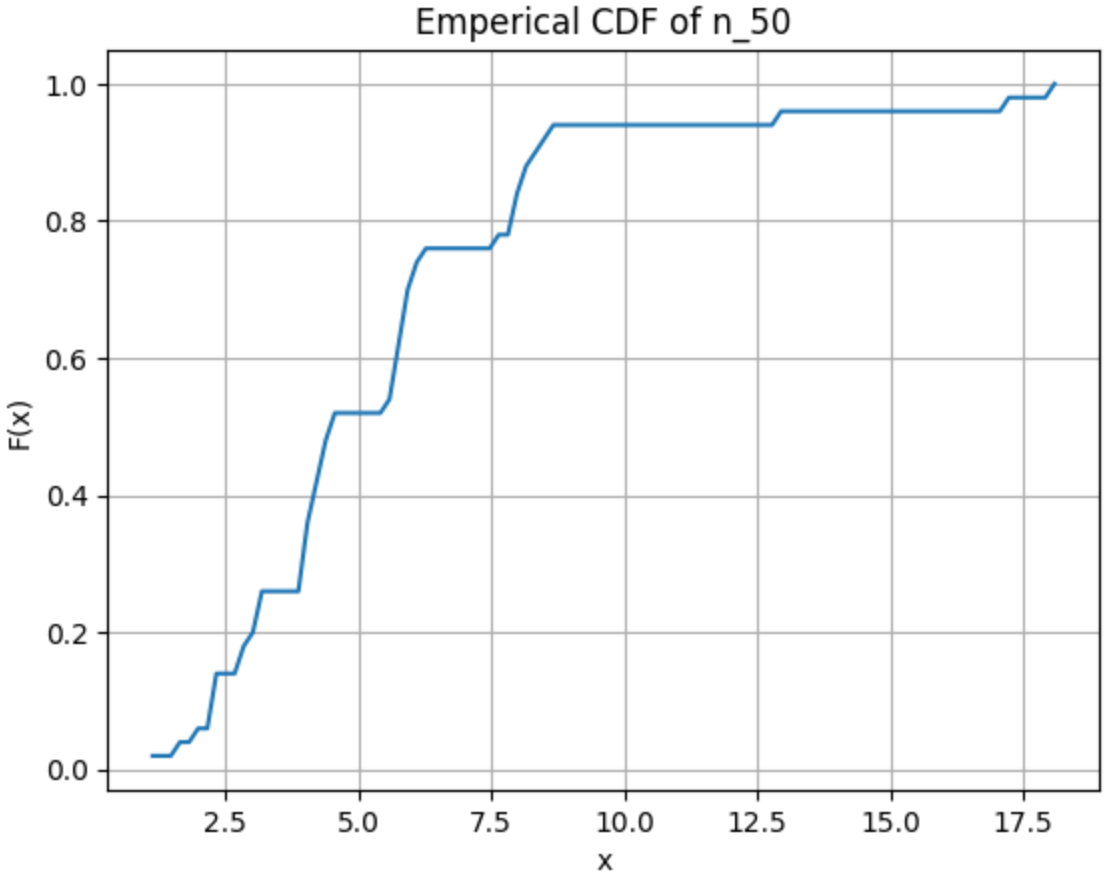
Out[ ]:

	n_50	n_150	n_300	d_50	d_150	d_300	t_peak	x_500
0	5.484789	31.727771	6.910956	1.086088	0.364691	-1.167716	155.625779	0.956636
1	8.578305	34.552430	3.908585	1.464196	-4.350747	-0.664125	120.744595	0.982187
2	1.525187	3.581051	7.528895	0.077954	0.445827	0.030962	357.805297	0.472349
3	6.151596	33.205968	6.199945	1.148558	-1.573964	-0.617079	133.964974	0.961004
4	4.443471	30.324971	8.335484	0.970687	2.765872	-1.672065	162.800388	0.881936

Plot the emperical cumulative probability distributions (CDFs) of the random variables (features).

In [ ]:
```
import matplotlib.pyplot as plt
from statsmodels.distributions.empirical_distribution import ECDF
import numpy as np

def plotcdf(df, f):
    print(df[f].describe())

    x = np.linspace(df[f].min(), df[f].max(), 100)

    ecdf = ECDF(df[f])

    # Plot the CDF
    plt.plot(x, ecdf(x))
    plt.xlabel('x')
    plt.ylabel('F(x)')
    plt.title(f'Emperical CDF of {f}')
    plt.grid(True)
    plt.show()

plotcdf(df, feature_names[0])
```

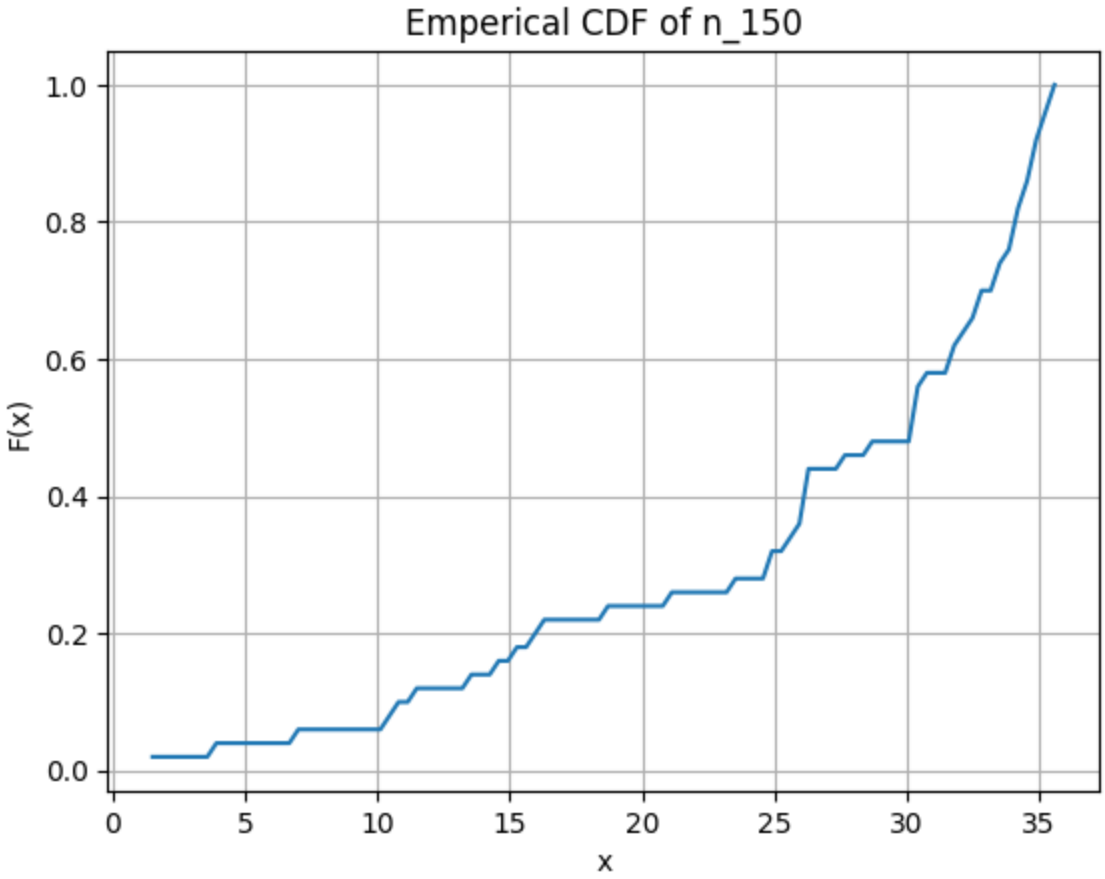
```
count    50.000000
mean      5.510800
std       3.396421
min       1.129245
25%       3.309000
50%       4.422730
75%       6.106346
max      18.092522
Name: n_50, dtype: float64
```



Interpretation: Almost all countries (> 90%) noted approximately 8 infections after 50 days. More than 8 infections were noted in only a few countries. The maximum number of infections after 50 days was approxomately 18.

In [ ]:
```
plotcdf(df, feature_names[1])
```

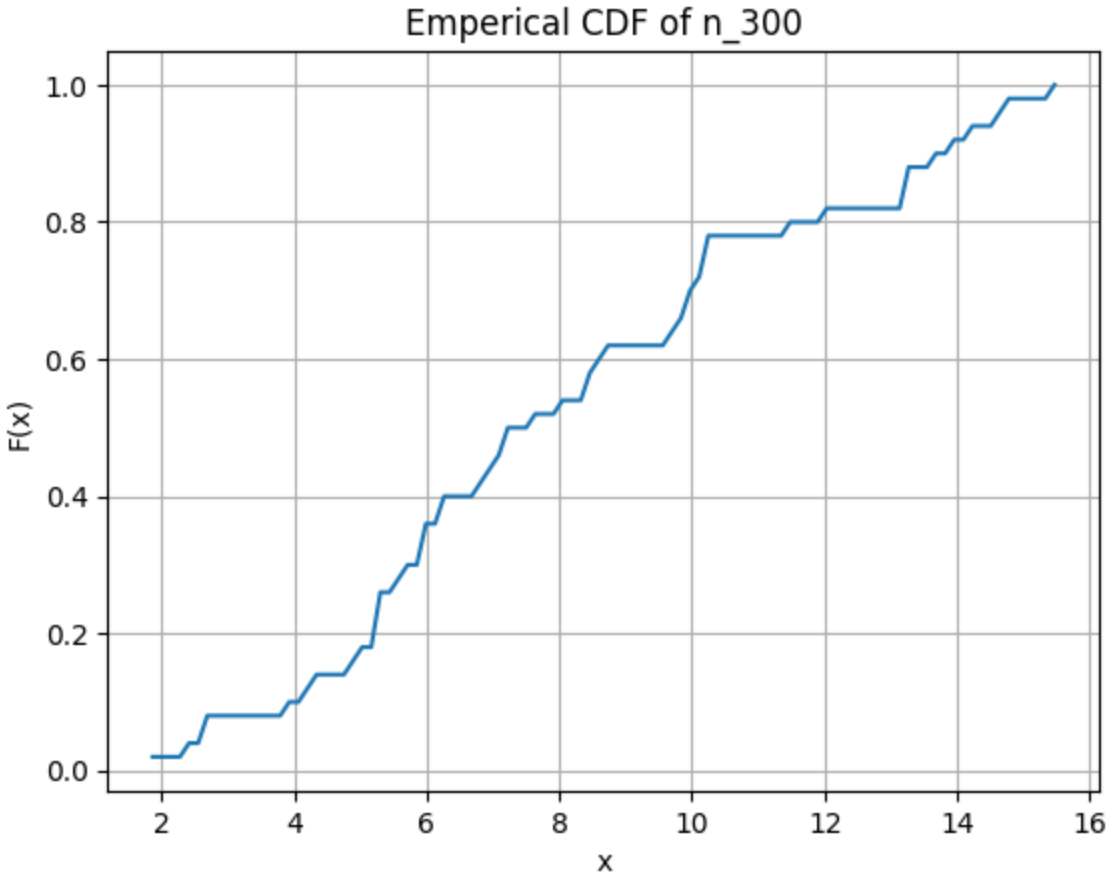
```
count    50.000000
mean     26.084033
std       9.348820
min       1.507993
25%      21.595062
50%      30.329100
75%      33.676859
max      35.575905
Name: n_150, dtype: float64
```



Interpretation: Only 25-30% of the countries noted a comparibly low number of infections of up to 25. Then there the ECDF went steep for another 20-25% where countries noted a number of infections between 25 an 30. The rest of the countries, which is almost 50% of them, got a number of infections between 30 and aproximately 36 after 150 days. Additionaly, none of the countries came with zero infections after 150 days.

In [ ]:
```
plotcdf(df, feature_names[2])
```

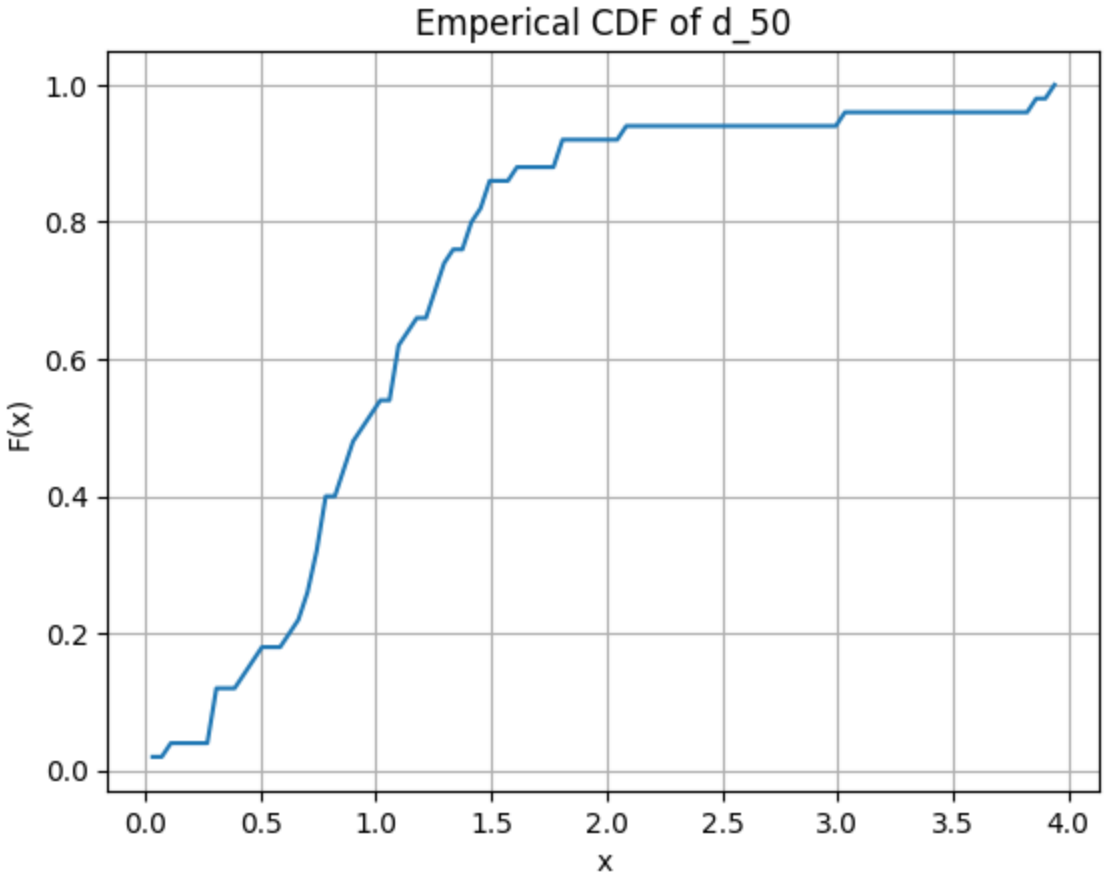
```
count    50.000000
mean      8.145338
std       3.640636
min       1.868088
25%       5.353104
50%       7.347345
75%      10.167518
max      15.466792
Name: n_300, dtype: float64
```



Interpretation: The half of the countries came with a maximum of approximately 7 infections after 300 days. For the rest of the countries, the number of infection ranges from 7 to approximately 15. Additionaly, none of the countries came with zero infections after 300 days.

In [ ]:
```
plotcdf(df, feature_names[3])
```

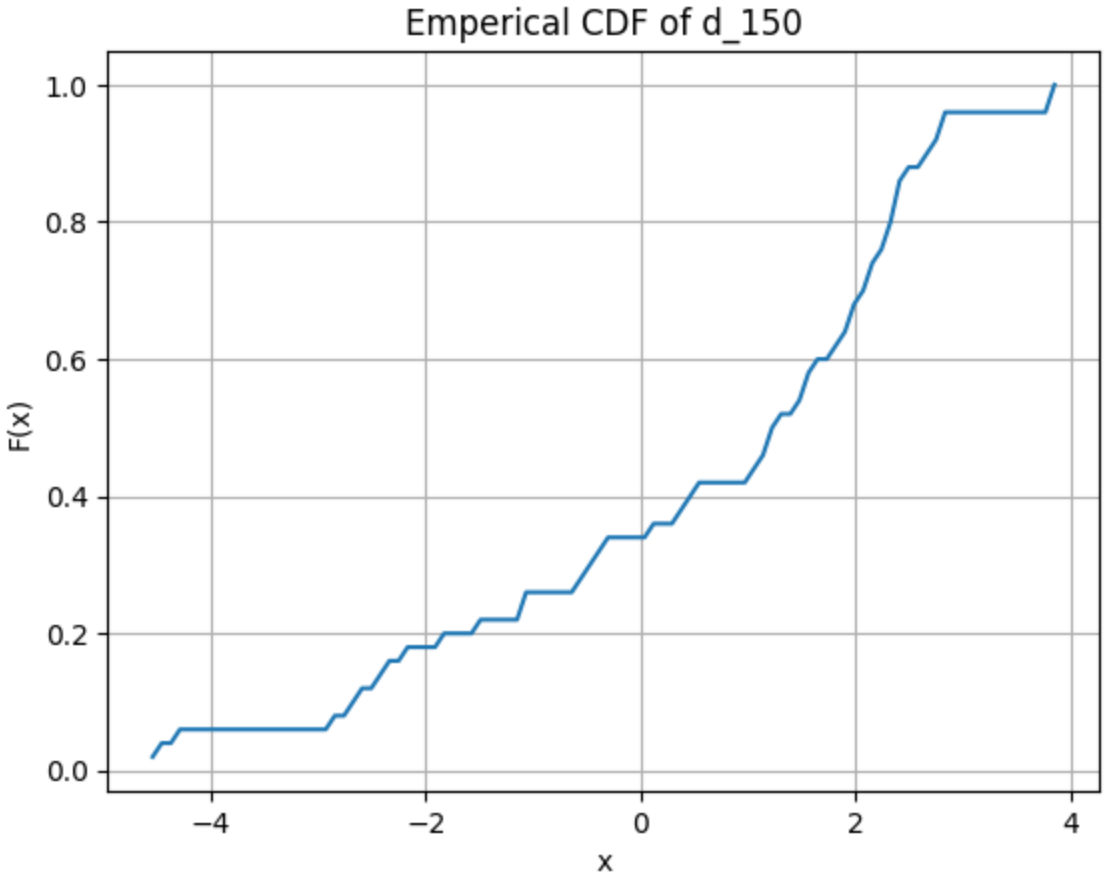
```
count    50.000000
mean      1.097608
std       0.784262
min       0.032972
25%       0.703376
50%       0.952191
75%       1.310096
max       3.940195
Name: d_50, dtype: float64
```



Interpretation: Almost all countries (> 90%) noted approximately 2 new infections on day 50. More than 2 new infections were noted in only a few countries. The maximum number of new infections on day 50 was approxomately 4.

In [ ]:
```
plotcdf(df, feature_names[4])
```

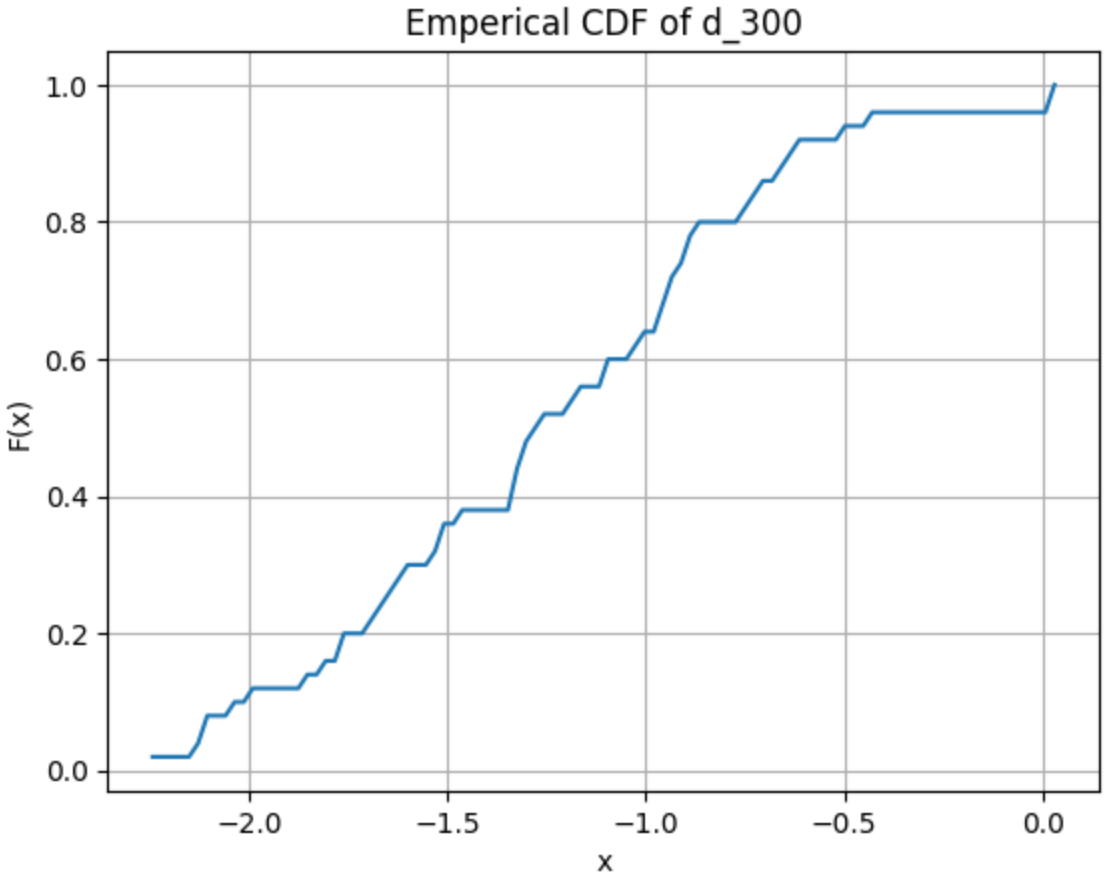
```
count    50.000000
mean      0.507043
std       2.193691
min      -4.539801
25%      -0.997498
50%       1.229447
75%       2.155732
max       3.846155
Name: d_150, dtype: float64
```



Interpretion: Approximately 35% of the countries noted an decreasing or at least no change in infections on day 150. The maximum decrease was about 4.5. However, 65% of the countries noted up to approximately 4 new infections on day 150.

In [ ]:
```
plotcdf(df, feature_names[5])
```

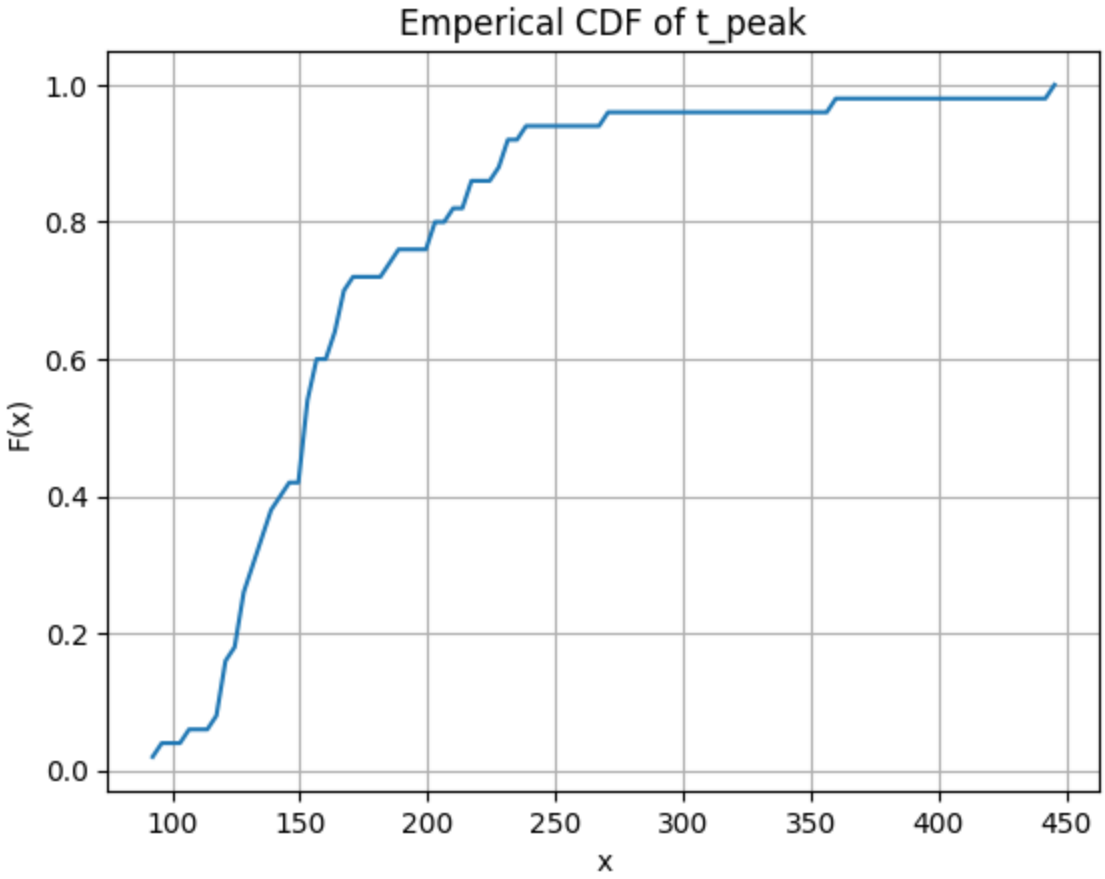
```
count    50.000000
mean     -1.249822
std       0.540493
min      -2.244317
25%      -1.647766
50%      -1.278831
75%      -0.904246
max       0.030962
Name: d_300, dtype: float64
```



Interpretation: Almost all (> 95%) of the countries noted a decrease of infections (less than zero new infections) on day 300. The maximum decrease was about 2.2. Just a few countries noted a small number of new infections on das 300.

In [ ]:
```
plotcdf(df, feature_names[6])
```

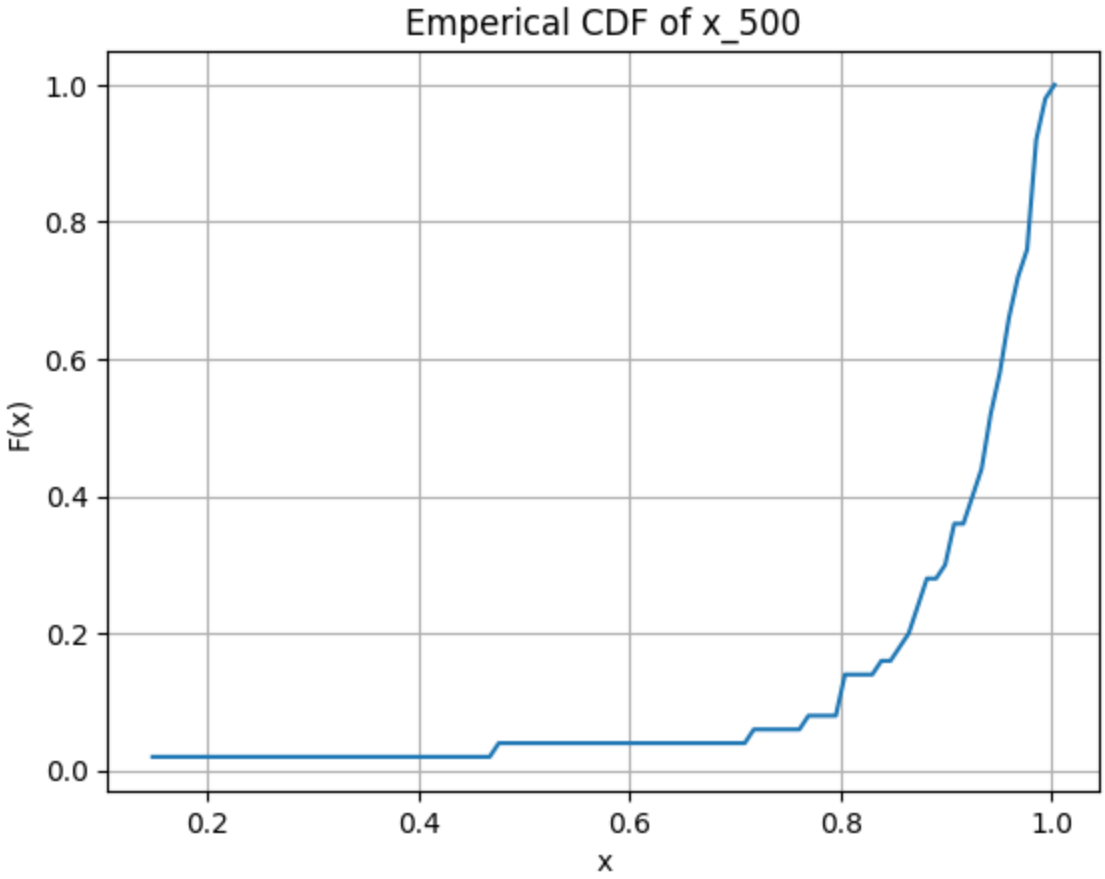
```
count     50.000000
mean     167.165455
std       63.076660
min       92.341618
25%      128.262318
50%      150.599951
75%      187.041275
max      445.305297
Name: t_peak, dtype: float64
```



Interpretation: The half of the countries reached the peak of infections within 150 days. Almost all countries (> 90%) reached the peak within approximately 240 days. Just a few countries reached the peak later, while the maximum was about 445 days.

In [ ]:
```
plotcdf(df, feature_names[7])
```

```
count    50.000000
mean      0.898774
std       0.141293
min       0.147541
25%       0.878408
50%       0.939200
75%       0.974210
max       1.003071
Name: x_500, dtype: float64
```



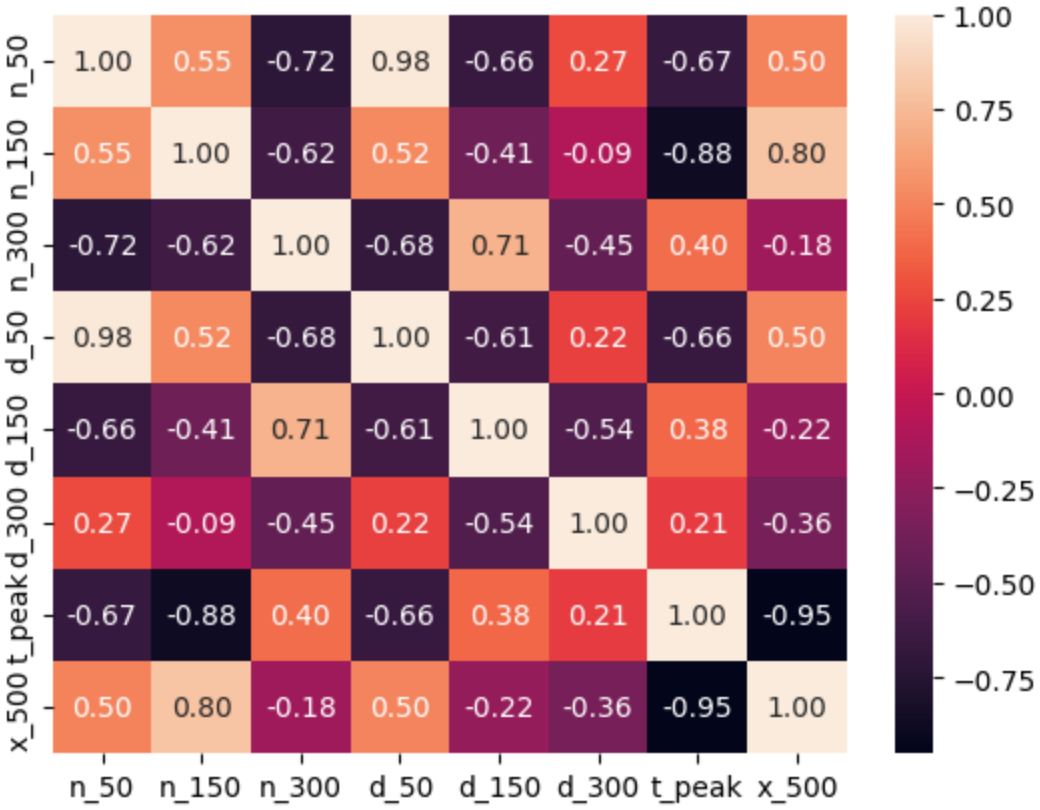
Interpretation: Unfortunately, there was no country with zero deaths after 500 days. Most of the countries (approximately 85%) came with 0.8-1 deaths after 500 days.

Recall, the sample mean value is an approximation of the expected value, or expectation of the random variable. The sample variance is the square of the sample standard deviation std.

Check the correlation between the features.

In [ ]:
```
import seaborn as sns
correlation = df.corr()
sns.heatmap(correlation, annot=True, fmt=".2f")
```

Out[ ]:
```
<AxesSubplot:>
```



Recall, the (sample) correlation of two (samples of) random variables X and Y their (sample) covariance normalized (divided) by the product of their (sample) standard deviation.

Discuss your findings. Your answer goes here.

There are some strong and very strong correlations in the data.

Examples for very strong correlations:

- n_50 and d_50 (positive, almost perfect correlation)
- X_500 and n_150 (positive)
- x_500 and t_peak (negative, alomst perfect correlation)
- t_peak and n_150 (negative)

Examples for strong correlation:

- n_300 and n_50 (negative)
- d_150 and n_50 (negative)
- t_peak and n_50 (negative)
- n_300 and n_150 (negative)
- d_50 and n_300 (negative)
- d_150 and n_300 (negative)
- d_150 and d_50 (negative)
- t_peak and d_50 (negative)

Examples for moderate correlation:

- n_150 and n_50 (positive)
- x_500 and n_50 (positive)
- d_50 and n_150 (positive)
- d_150 and n_150 (negative)
- d_300 and n_300 (positive)
- t_peak and n_300 (negative)
- x_500 and d_50 (positive)
- d_300 and d_150 (negative)

The rest shows just a weak down to no correlation.

The most interesting ones are the ones with the highest correlation, which are n_50/d_50 and x_500/t_peak coming with an amlost perfect correlation with different signs, i.e. positive for n_50/d_50 and negative for x_500/t_peak.

According to the scenario n_50 cannot be smaller than d_50, because d_50 conceptually adds up to n_50. Hence, the correlation is expected to be positive and almost perfect. So it a perfect correlation by design.

The almost perfect negative correlation of t_peak and X_500 is not caused by design. However, it gives evidence for countries having a lower death rate, when they reach the peak later. This could give some clues about the targets of the management of infections. Looking at the other

vers strong correlated pairs, makes this more assessable. There is also a very strong positive correlation between x_500 and n_150, meaning that the number of new infections of day 150 could be a good indicator of expected deaths and therefore gives the chance to adjust the management. Moreover, n_150 is very strong negative correlated to t_peak, which the authorities should try to postpone as far as possible. So once again, n_150 seem to be an interesting indicator.

1.4 Try to fit parametric probability distributions

We go through a set of common probability distributions and try to fit any of them to the data. We use the Chi-square and the Kolmogorov-Smirnov tests to check the goodness of fit. *Note that fitting and testing using the same data is actually not quite correct. We should use cross-validation.*

Student: ... I neglected it, too.

Here, some additional implementations I used to generalize the fitting and evaluation of the different distributions:

In [ ]:
```
from scipy import stats
from scipy.stats import norm, poisson, expon, gamma, genextreme, gaussian_kde
from scipy.integrate import quad
import math

def fit(data, func):
    func = getattr(stats, func)
    if func == poisson:
        return [np.mean(data)] # using the mean of the data as parameter
    if func == gaussian_kde:
        return gaussian_kde(data)
    return func.fit(data)

def gaussian_cdf(x, kde):
    cdf_values = np.zeros_like(x)  # Initialize array for CDF values
    for i, xi in enumerate(x):
        cdf_values[i], _ = quad(kde.pdf, -np.inf, xi)  # Integrate KDE's PDF up to
    return cdf_values

def cdf(data, pd, params):
    pd = getattr(stats, pd)
    if pd == gaussian_kde:
        return gaussian_cdf(data, params)
    return pd.cdf(data, *params)

def pdf(data, pd, params):
    pd = getattr(stats, pd)
    if pd == gaussian_kde:
        return params.pdf(data)
    return pd.pdf(data, *params)

class PDF():

    def __init__(self, func, params) -> None:
        self.func = func
        self.params = params

    def calc(self, x):
        return pdf(x, self.func, self.params)

values_must_be_positive =["poisson", "expon", "gamma"]


distributions = ['norm', 'poisson', 'expon', 'gamma', 'genextreme']
```

I got some serious problems with the chi² test. I think, I understood how the test works, but maybe I failed to retrieve the expected data for this test, since it must be categorical and the data we use is continuing. So at this point, it would be helpful to get an example implementation for this. Additionally, the library I used come with some constrains about the expected and observed values in the chi² test. Maybe it is because of the the problems mentioned above, that I ran into this error several times.

I provided an implementation for the chi² tests of the fitted curves below. So I hope you can give me some hints based on this.

However, in order to find the best fit, I made use of another test (Cramer-von-Mises Test) and found a proper fit for all of the features. (after the next cell)

In [ ]:
```
from scipy.stats import chisquare

for f in feature_names:
    print('')
    print(f'Check for feature: {f}')
    # Get x
    x = df[f]
    ecdf = ECDF(x)


    for dist in distributions:
        if x.min() < 0 and dist in values_must_be_positive:
            continue

        print(f'Test: {dist}')

        params = fit(df[f], dist)

        observed = ecdf(df[f].sort_values())
        observed, bins = np.histogram(observed)
        expected = cdf(df[f].sort_values(), dist, params)
        expected, _ = np.histogram(expected, bins)

        try:
            res1 = chisquare(observed, expected)
            if res1.pvalue < 0.05:
                print(f'Not {dist} with 5% significance level')
            else:
                print(f'{dist} with 5% significance level')
        except ValueError:
            print('got Value error')
```

```
Check for feature: n_50
Test: norm
Not norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
got Value error

Check for feature: n_150
Test: norm
got Value error
Test: poisson
got Value error
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
genextreme with 5% significance level

Check for feature: n_300
Test: norm
norm with 5% significance level
Test: poisson
got Value error
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
genextreme with 5% significance level

Check for feature: d_50
Test: norm
Not norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
got Value error

Check for feature: d_150
Test: norm
got Value error
Test: genextreme
Not genextreme with 5% significance level

Check for feature: d_300
Test: norm
```

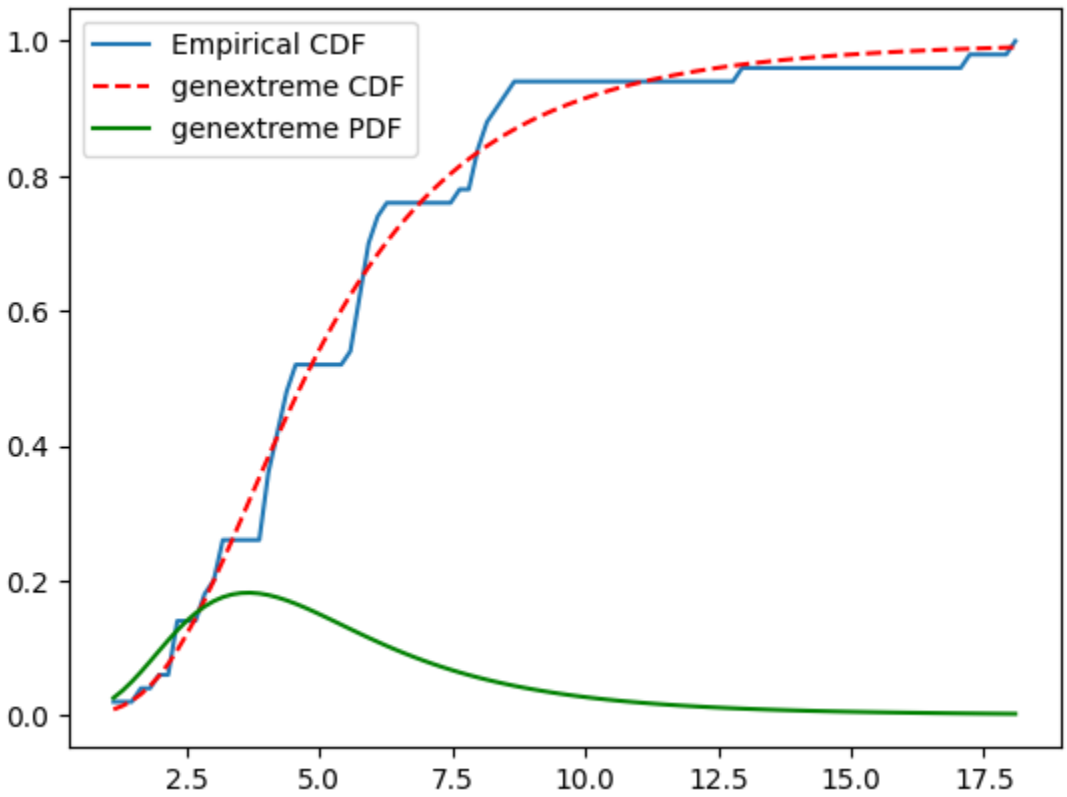
```
Not norm with 5% significance level
Test: genextreme
genextreme with 5% significance level

Check for feature: t_peak
Test: norm
Not norm with 5% significance level
Test: poisson
got Value error
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
got Value error

Check for feature: x_500
Test: norm
got Value error
Test: poisson
Not poisson with 5% significance level
Test: expon
got Value error
Test: gamma
got Value error
Test: genextreme
/media/home/ngundermann/workspace/DML_notebooks/venv/lib/python3.10/site-packages/sc
ipy/stats/_stats_py.py:7985: RuntimeWarning: divide by zero encountered in divide
  terms = (f_obs_float - f_exp)**2 / f_exp
got Value error
```

Here, the implementation for retrieving the best fit.

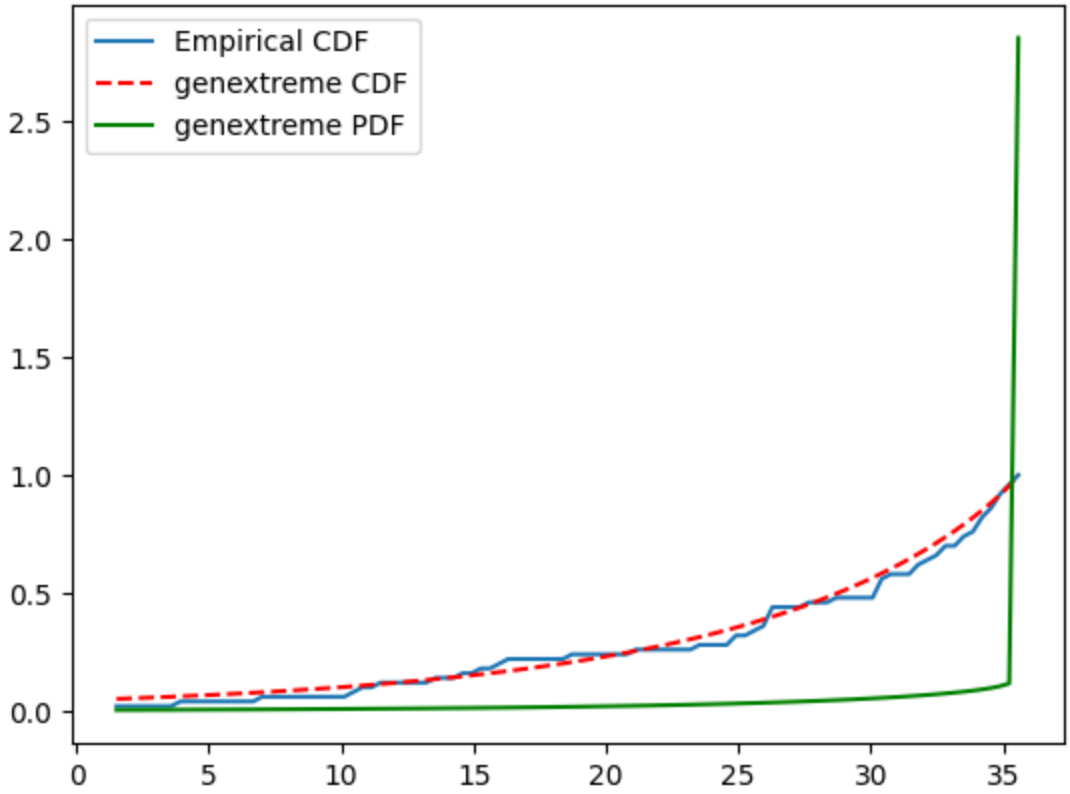
In [ ]:
```
from scipy.stats import cramervonmises, kstest

feature_pdfs = {"n_50":None,
                "n_150":None,
                "n_300":None,
                "d_50":None,
                "d_150":None,
                "d_300":None,
                "t_peak":None,
                "x_500":None,}


feature_pdfs_meta = {"n_50":None,
                "n_150":None,
                "n_300":None,
                "d_50":None,
                "d_150":None,
                "d_300":None,
                "t_peak":None,
                "x_500":None,}

for f in feature_names:
    print('')
    print(f'Check for feature: {f}')
    # Get x
    x = df[f]
    x_values = np.linspace(df[f].min(), df[f].max(), 100)
    ecdf = ECDF(x)


    for dist in distributions:
        if x.min() < 0 and dist in values_must_be_positive:
            continue

        print(f'Test: {dist}')

        params = fit(df[f], dist)
        res1 = kstest(df[f], lambda x: cdf(x, dist, params))
        # check KS-Test
        if res1.pvalue < 0.05:
            print(f'Not {dist} with 5% significance level')
        else:
            res2 = cramervonmises(df[f], lambda x: cdf(x, dist, params))

            # check cramer-von-mises Test
            if res2.pvalue < 0.05:
                print(f'Not {dist} with 5% significance level')
            else:
                print(f'{dist} with 5% significance level')
                if feature_pdfs_meta[f] is None or feature_pdfs_meta[f]['D'] > res1
                    feature_pdfs_meta[f] = {'name':dist,
                                           'D': res1.statistic,
                                           'params':params}


    if feature_pdfs_meta[f] is not None:
```

```
        meta_info = feature_pdfs_meta[f]
        print(f'\nBest fit: {meta_info["name"]} (D: {meta_info["D"]})')
        print(f'Params: {meta_info["params"]}')
        feature_pdfs[f] = PDF(meta_info['name'], meta_info['params'])

        # Plot CDF
        plt.plot(x_values, ecdf(x_values), label='Empirical CDF')

        # # Plot fitted CDF
        plt.plot(x_values, cdf(x_values, dist, params), c='r', label=f'{dist} CDF',

        # # Plot PDF
        plt.plot(x_values, pdf(x_values, dist, params), c='g', label=f'{dist} PDF')

        plt.legend()
        plt.show()
```

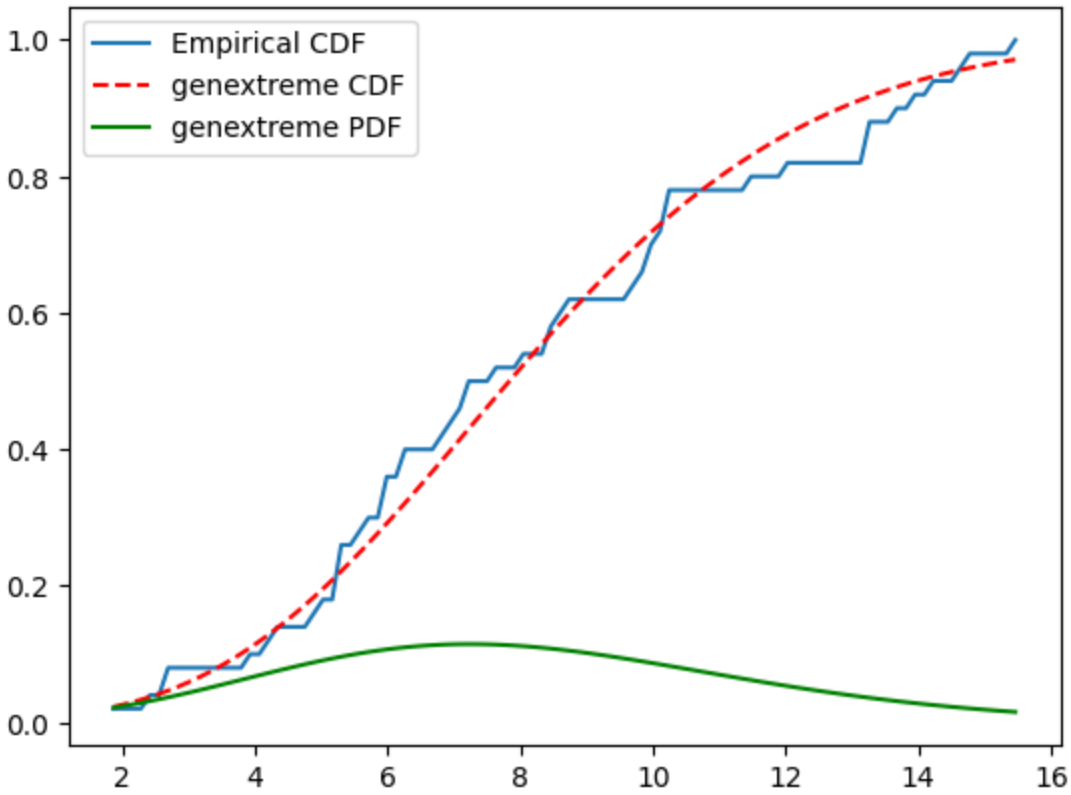
```
Check for feature: n_50
Test: norm
norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.10580729786458315)
Params: (-0.1576646663043923, 3.955832823506072, 2.04799320973682)
```



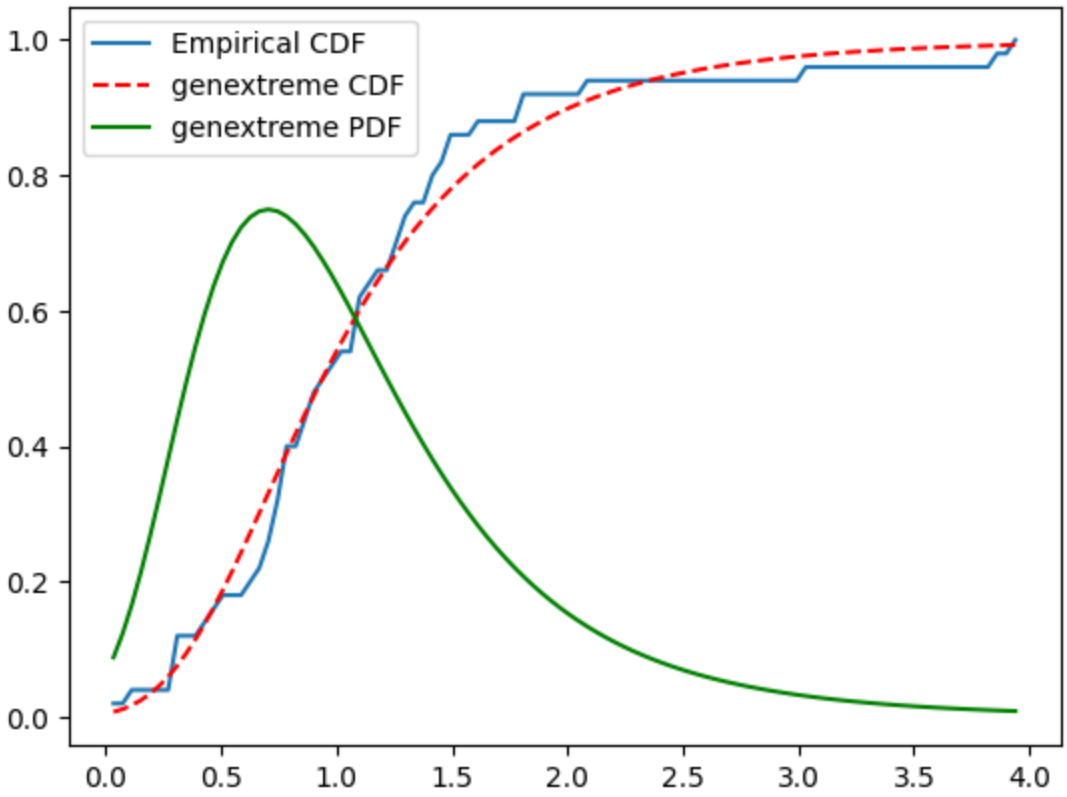
```
Check for feature: n_150
Test: norm
Not norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
Not gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.09845413421706728)
Params: (1.1043653516891614, 25.360698320215405, 11.281320162811323)
```



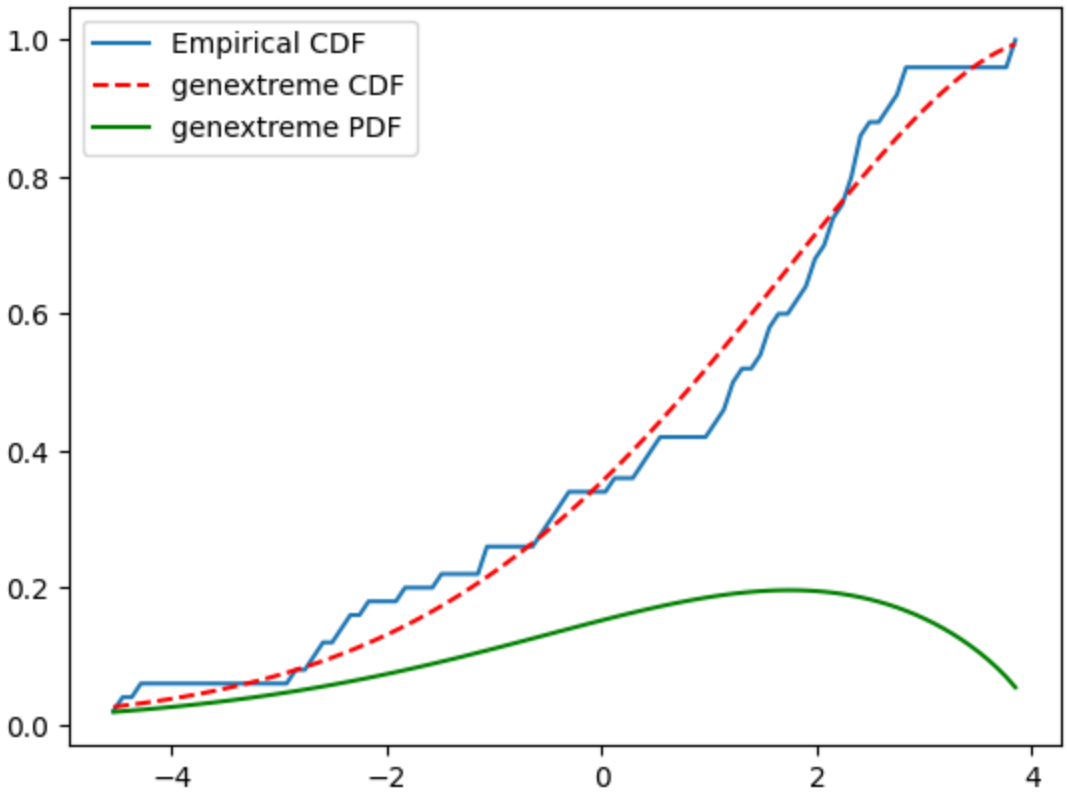

```
Check for feature: n_300
Test: norm
norm with 5% significance level
Test: poisson
poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: gamma (D: 0.08248683752994002)
Params: (6.778259638472212, -1.4798803297973264, 1.4200133660091883)
```



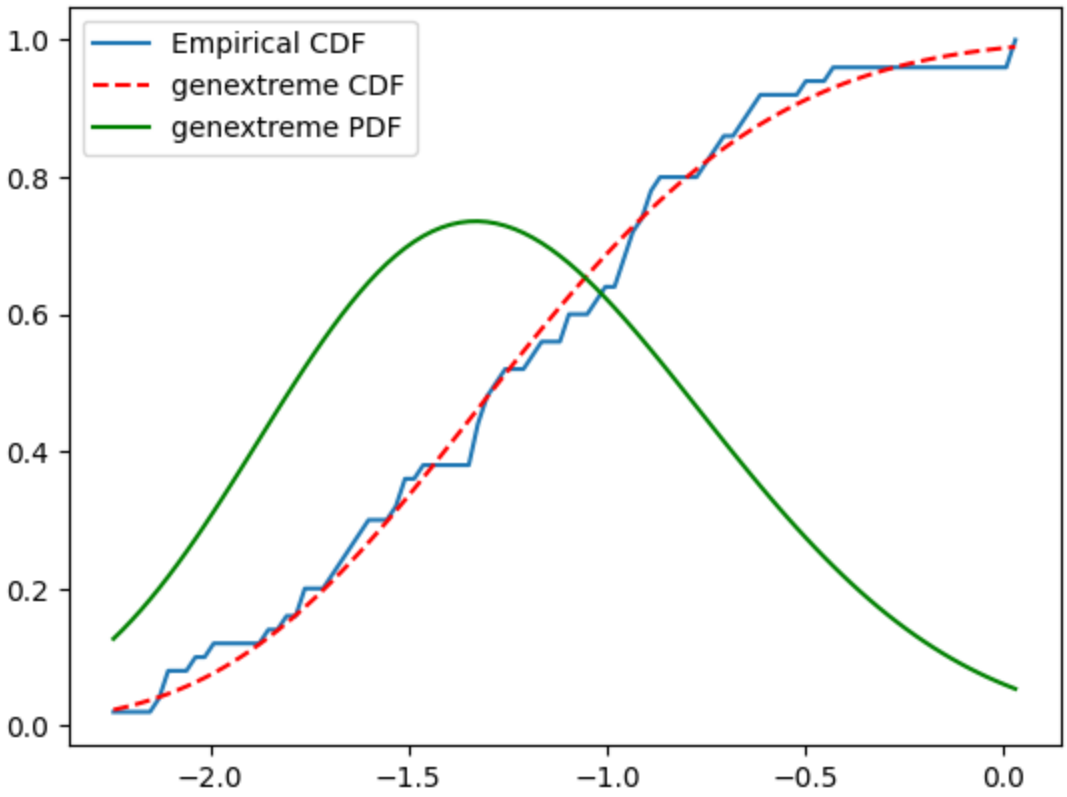

```
Check for feature: d_50
Test: norm
norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.0900096651434627)
Params: (-0.10715612734229667, 0.7534189627585046, 0.49340566919399403)
```



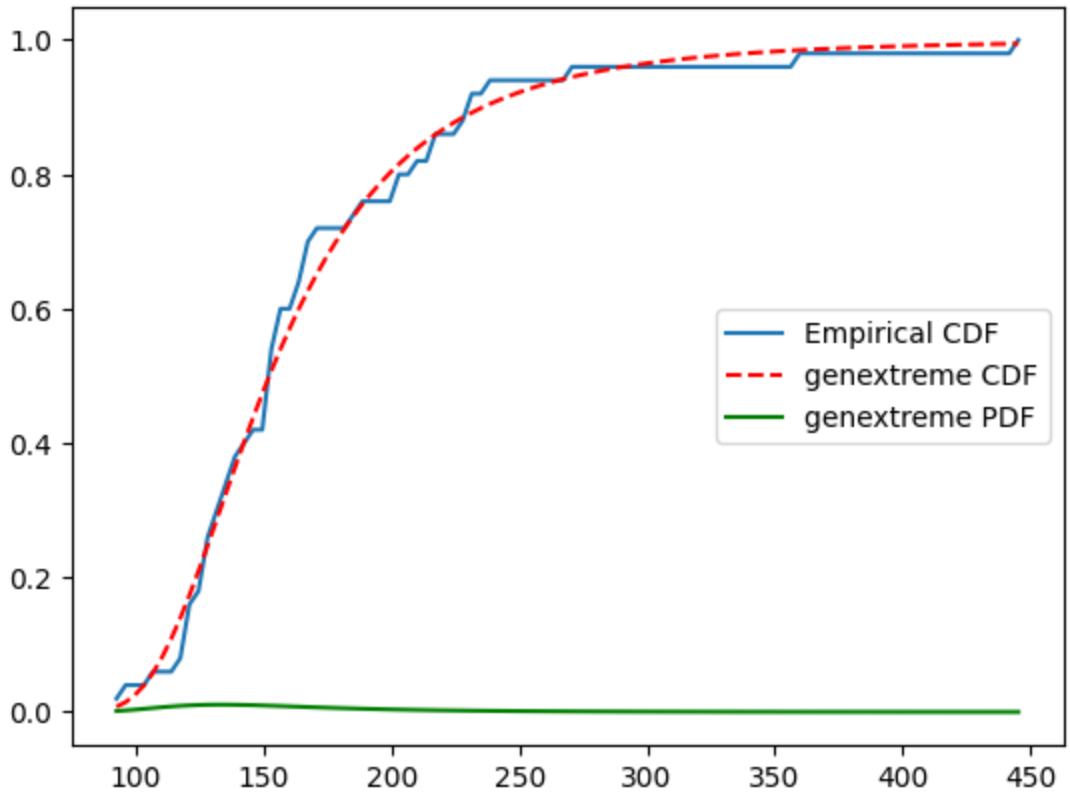

```
Check for feature: d_150
Test: norm
norm with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.10548264421965797)
Params: (0.5985644770709595, 0.08515655630859556, 2.36333034931815)
```



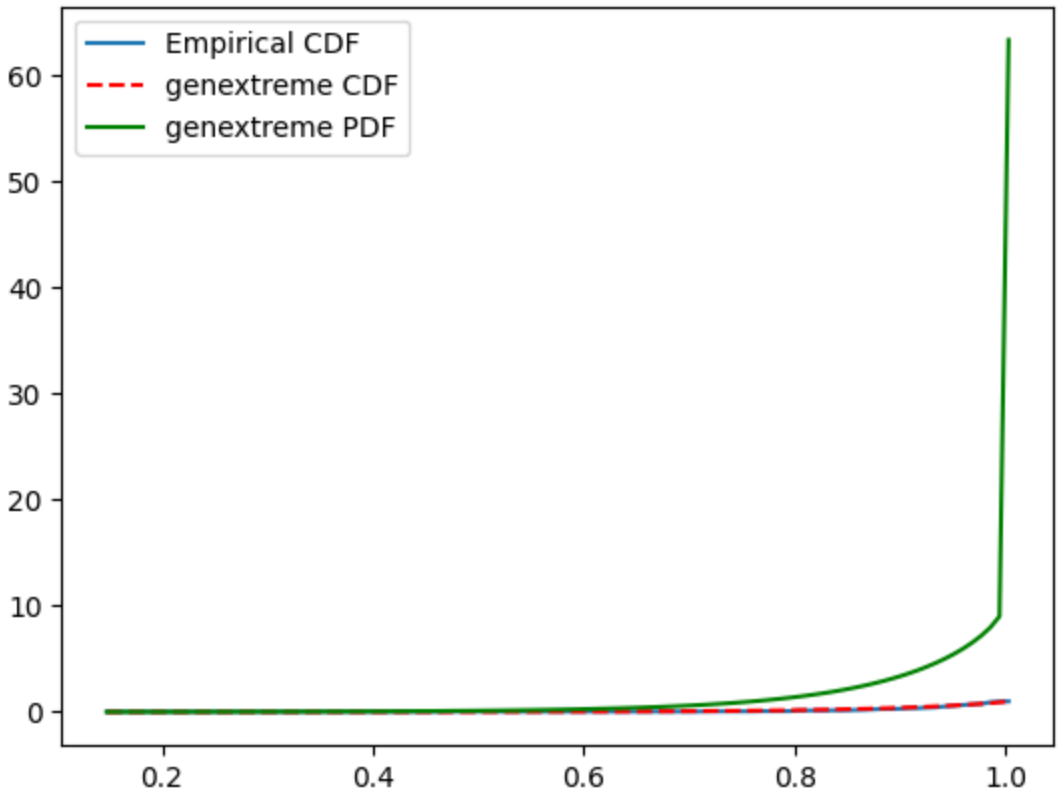

```
Check for feature: d_300
Test: norm
norm with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: norm (D: 0.06049636887514043)
Params: (-1.249822268966371, 0.5350603552002773)
```




```
Check for feature: t_peak
Test: norm
Not norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.0775339768044333)
Params: (-0.192539215177747, 139.08322908707058, 34.321980725567144)
```




```
Check for feature: x_500
Test: norm
Not norm with 5% significance level
Test: poisson
Not poisson with 5% significance level
Test: expon
Not expon with 5% significance level
Test: gamma
Not gamma with 5% significance level
Test: genextreme
genextreme with 5% significance level

Best fit: genextreme (D: 0.11790911267434112)
Params: (1.058815728423074, 0.8926803963611483, 0.11688284833632757)
```


Empirical CDF
genextreme CDF
genextreme PDF
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1.0

Describe the fitted distributions. Your answer goes here.

Answer:

For n_50 we could have fitted a normal distribution, a gamma distribution and an extreme value distribution with significance. The extreme value distribution performed best with a D-statistic of 0.10580729786458315. Comparing the fitted with the emperical CDF it seems to be quite a good fit, taking into account that the emperical cdf is based on just 50 datapoints, what might make it look somewhat like a step function.

For n_150 we could have fitted an extreme value distribution with significance. Comparing the fitted with the emperical CDF it seems to be quite a good fit.

For n_300 we could have fitted a normal distribution, a poisson distribution, a gamma distribution and an extreme value distribution with significance. The fitting to a gamma distribution performed best with a D-statistic of 0.08248683752994002. The fitted curve also looks quite good compared to the emperical one. However, the fit is better for the lower 50% of the CDF.

For d_50 we could have fitted a normal distribution, a gamma distribution and an extreme value distribution with significance. The extreme value distribution performed best with a D-statistic of 0.0900096651434627. Like for n_50 the fitted CDF looks quite good compared to the emperical one.

For d_150 we could have fitted a normal distribution and an extreme value distribution with significance. Here, the extreme value distribution performed best with a D-statistic of 0.10548264421965797. Taking a larger sample could result in a much lower D-statistic and therefore in a much better fit.

For d_300 we could have fitted a normal distribution and an extreme value distribution with significance. Here, the extreme value distribution performed best with a D-statistic of 0.06049636887514043. The low D-statistics indicates quite a good fit to the emperical cdf, which we would also support on a ocular comparison of the fitted and the emperical curve.

For t_peak we could have fitted a gamma distribution and an extreme value distribution with significance. The extreme value distribution performed best with a D-statistic of 0.0775339768044333. Like for d_300 the D-statistic is very low, hence, the fitted CDF looks quite good compared to the emperical one.

For x_500 we could fitted a extreme value distribution with significance. According to the coverage of the fitted and the emperical CDF, the fit was quite good.

1.5 Joint and conditional probabilities, chain rule and Bayes' Theorem

Lets calculate the probability of having a mild start and a disastrous end of the epidemia. We translate that to the joint probability that no more than 4% are infected after 50 days, i.e., N50 ≤ 4% and more than 0.95% die in the end, i.e., X500 > 0.95%.

In [ ]:
```
pos = len(df[(df['n_50'] <= 4) & (df['x_500'] > 0.95)])
print(f'positive cases: {pos}')

all = len(df)
P = pos/all
print(f'P: {P}')
```

```
positive cases: 1
P: 0.02
```

Student: So $P(N_{50} \leq 4\%, X_{500} > 0.95\%)$ ≈ 0.02 in my case, since I worked with only 50 countries instead of 125.

We check the product of the two probabilities $P(N_{50} \leq 4\%)P(X_{500} > 0.95\%)$:

In [ ]:
```
p_n50 = len(df[df['n_50'] <= 4])/len(df)
p_x500 = len(df[df['x_500'] > 0.95])/len(df)

P2 = p_n50 * p_x500
print(f'P2: {P2}')
```

```
P2: 0.1428
```

We are not surprised that the result is different from the joint probability as the random variables N50 and X500 not independent (double-check the results from the correlation analysis).

Student: This holds in my case too

What is the conditional probability of more than 0.95% die in the end given that we know that no more 4% are infected after 50 days.

In [ ]:
```
P_AB = pos/len(df[df['n_50'] <= 4])

print(f'P_AB: {P_AB}')
```

```
P_AB: 0.058823529411764705
```

Student: So $P(X_{500} > 0.95\% | N_{50} \leq 4\%) \approx 0.0588$.

The product or chain rule of probability states that

$$P(A, B) = P(A|B)P(B) (5)$$

$$P(X_{500} > 0.95\%, N_{50} \leq 4\%) = P(X_{500} > 0.95\% | N_{50} \leq 4\%)P(N_{50} \leq 4\%) (6)$$

And we don't get disappointed:

In [ ]:
```
P = P_AB * p_n50
print(f'P: {P}')
```

```
P: 0.02
```

Let us numerically confirm Bayes' Theorem for this case. We'd expect that.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (1)$$

$$P(X_{500} > 0.95\%|N_{50} \leq 4\%) = \frac{P(N_{50} \leq 4\%|X_{500} > 0.95\%)P(X_{500} > 0.95\%)}{P(N_{50} \leq 4\%)} \quad (2)$$

And we don't get disappointed:

In [ ]:
```
P_BA = pos/len(df[df['x_500'] > 0.95])
P_AB = P_BA * p_x500 / p_n50

print(f'P_AB: {P_AB}')
```

```
P_AB: 0.05882352941176469
```

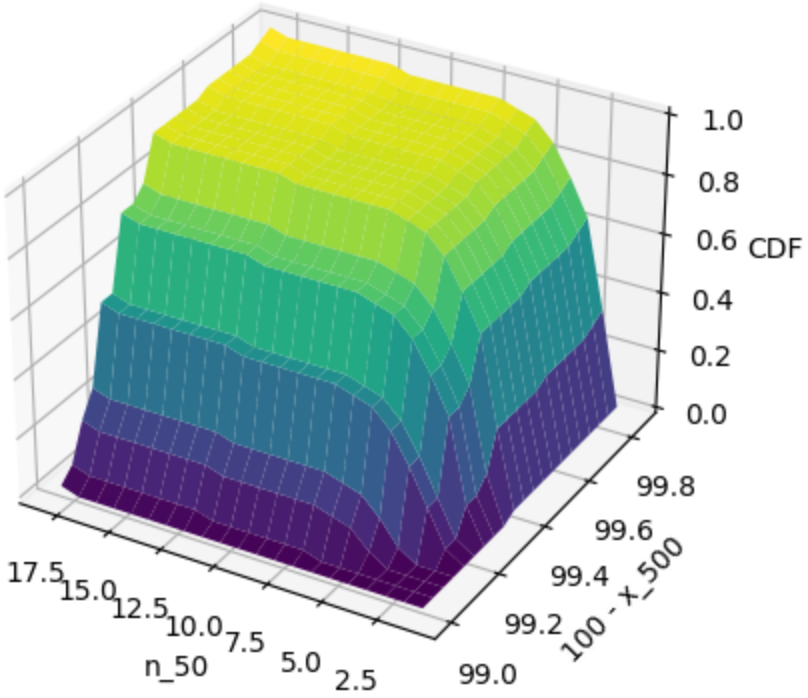
So far, we have looked at the joint (sample) probability of $P(N_{50} \leq n, X_{500} > x)$ for concrete bounds n = 4 and x = 0.95. Let us now understand the joint (sample) probability of $F(n, x) = P(N_{50} \leq n, X_{500} > x)$ as a function of these bounds n and x. Since, $0 \leq X_{500} \leq 100$ this function is equal to joint (sample) cumulative distribution function:

$$F(n, x) = CDF_{N_{50}, 100-X_{500}}(n, 100 - x) \quad (3)$$

$$CDF_{N_{50}, 100-X_{500}}(n, y) = P(N_{50} \leq n, 100 - X_{500} \leq y) \quad (4)$$

In [ ]:
```
# define cdf and F function

def cdf_sample(df,n,y):
    pos_cases = len(df[(df['n_50'] <= n) & (df['x_500'].apply(lambda x: 100-x) <= y
    return pos_cases/len(df)

def F(df, n, x):
    cdf = cdf_sample(df,n,x)
    return cdf
```

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


_, n_50_bin_edges = np.histogram(df['n_50'], bins=20)
_, x_500_bin_edges = np.histogram(100-df['x_500'], bins=20)


# Generate meshgrid for plotting
x_sequence = range(len(n_50_bin_edges))
y_sequence = range(len(x_500_bin_edges))

X, Y = np.meshgrid(n_50_bin_edges, x_500_bin_edges)

# Initialize an empty matrix to store computed values
Z = np.zeros((len(n_50_bin_edges), len(x_500_bin_edges)))

# Compute values for each pair of elements using a nested loop
for x in x_sequence:
    for y in y_sequence:
        Z[x, y] = F(df, n_50_bin_edges[x], x_500_bin_edges[y])

# Plot the surface
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
ax.invert_xaxis()
# Set labels and title
ax.set_xlabel('n_50')
ax.set_ylabel('100 - x_500')
ax.set_zlabel('CDF')


plt.show()
```



Interprete this CDF. Your answer goes here.

Answer:

There is no chance for a country (chance of 0%) to note at most ~4 infections within the first 50 days while having more than ~0.9 deaths after 500 days.

80% of the countries managed to keep the number of infection within the first 50 days to at most ~5 and came with a total number of deaths of at least ~0.6.

It is very likely (~85%) than a country noted at most 7.5 new infections within the first 50 days and comes with a total number of deaths after 500 days of at least ~0.56

There is just a ~20% chance for a country to have at most ~2 new infections within the first 50 days and got at least ~0.8 deaths after 500 days.

(the next ones are hard to see in the figure, because I fixed one variable. To give more exact percentages, I would plot the figures for this cases individually.)

When a country noted 15 infections within the first 50 days, there is a 50% chance of having at least ~0.8 deaths after 500 days. However, when a country noted 3 infections within the first 50 days, there is just a ~33% chance of having at least ~0.8 deaths after 500 days.

When a country had 10 infections within the first 50 days it was very likely (close to 100%) to got a total number of death after 500 day of at least 0.6. However, the chance of getting a number of total deaths of at least 0.8 was about ~50%.

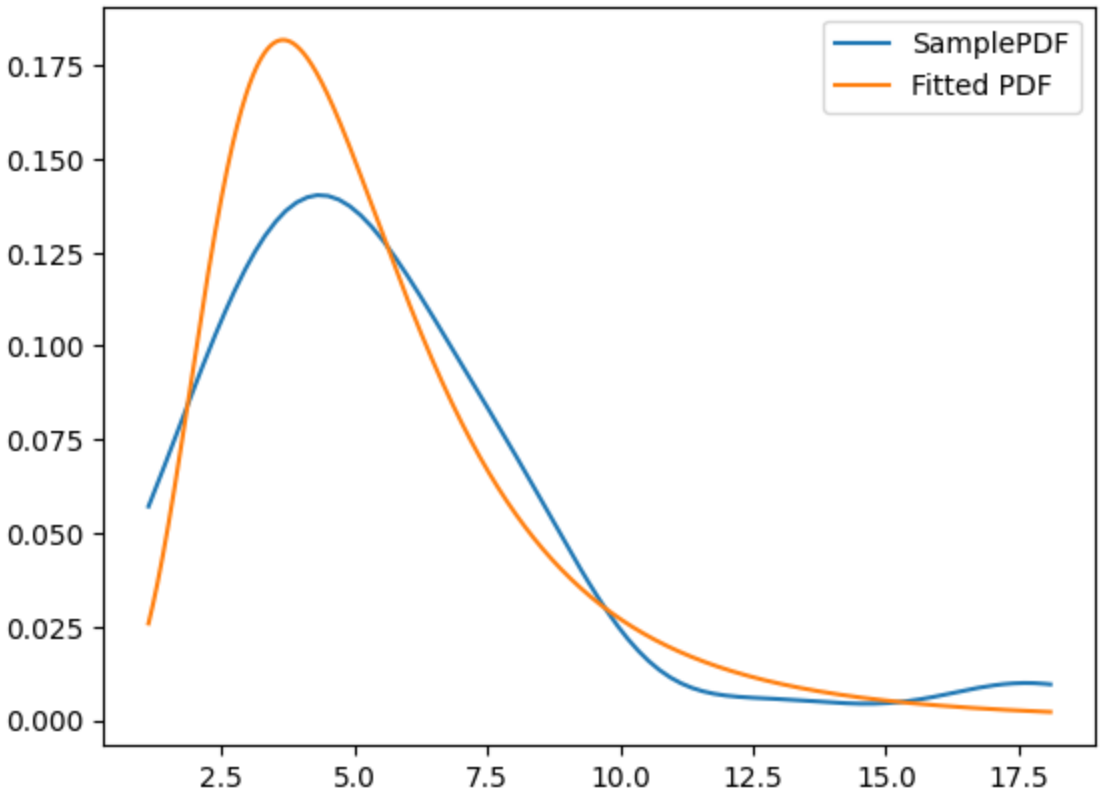
Finally, we compare a function interpolating a sample distribution with the fitted parameterized PDF of that distribution computed earlier, cf. distribution_fitting. We look at N50 and X500 as two examples.

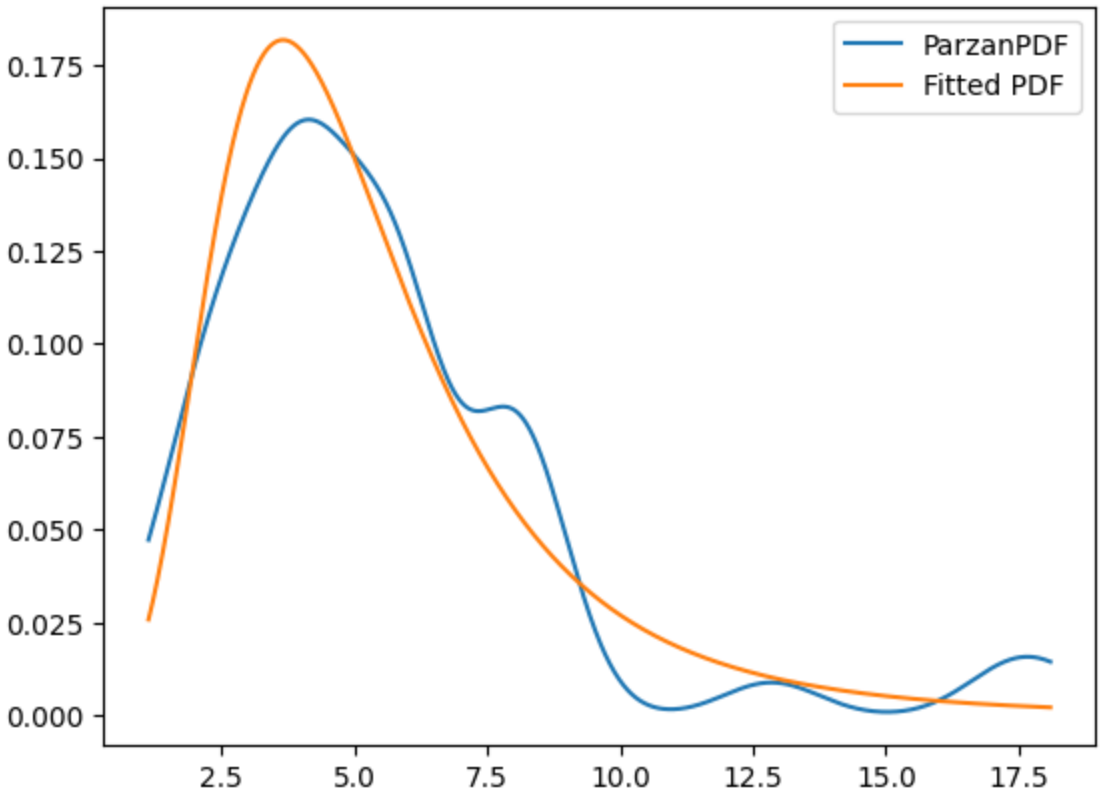
Therefore, we generate auxiliary functions plotPDF for plotting both functions, and two alternatives samplePDF and parzanPDF for interpolating the sample distribution. The former is simple to use and often good enough. The latter allows for fine tuning and is described in detail in the public notebook “Approximate PDFs with Parzen window density estimation”.

In [ ]:
```
from scipy.interpolate import interp1d

def samplePDF(sample, pd_fit):
    kde = gaussian_kde(sample)
    xi = np.linspace(-100, 100, 1000)
    f = kde.evaluate(xi)
    pdf = interp1d(xi, f, bounds_error=False, fill_value="extrapolate")

    x = np.linspace(min(sample), max(sample), 1000)

    plt.plot(x, pdf(x), label='SamplePDF')
    plt.plot(x, pd_fit.calc(x), label='Fitted PDF')
    plt.legend()
    plt.show()
    return pdf
```

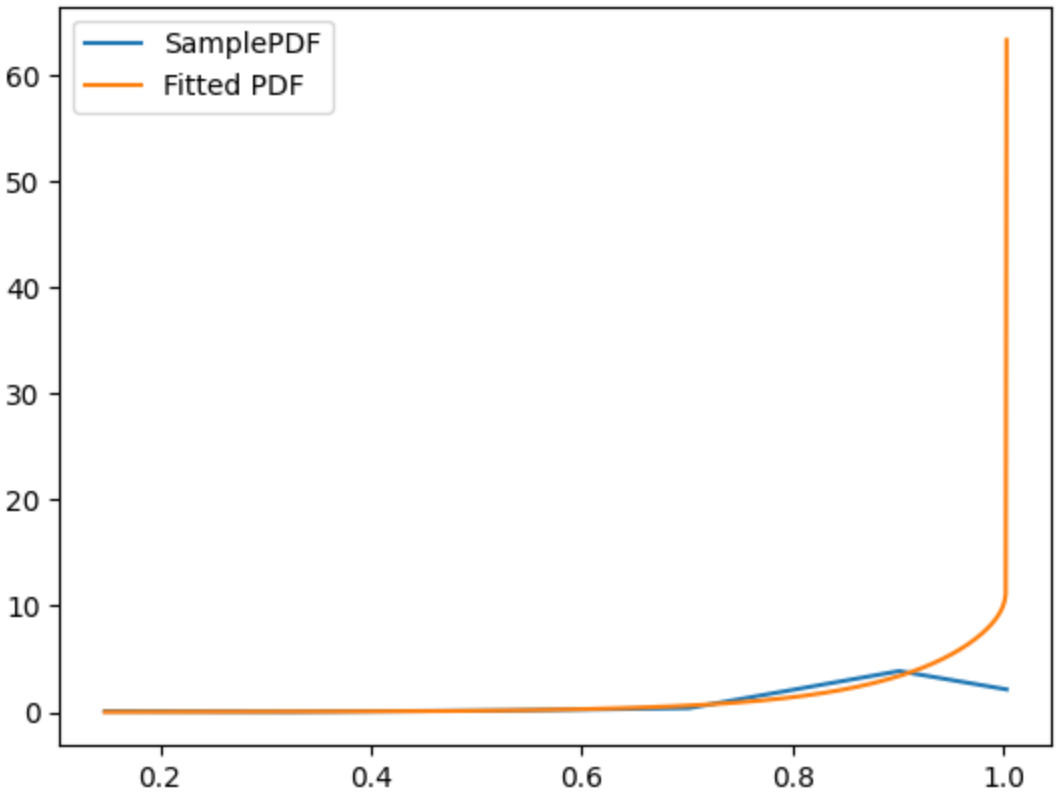
In [ ]:
```
def parzanPDF(sample, a, b, h, pd_fit):
    x = np.linspace(a, b, 1000)
    mu = 0
    sigma = 1
    len_x = len(x)
    len_s = len(sample)
    f = np.zeros((len_x))
    for j in range(len_x):
        xi=x[j]
        for i in range(len_s):
            f[j] = f[j] + norm.pdf((xi-sample[i])/h, mu, sigma)
        f[j] = f[j] / (len_s*h)

    pdf = interp1d(x, f, bounds_error=False, fill_value="extrapolate")

    x = np.linspace(min(sample), max(sample), 1000)

    plt.plot(x, pdf(x), label='ParzanPDF')
    plt.plot(x, pd_fit.calc(x), label='Fitted PDF')
    plt.legend()
    plt.show()
    return pdf
```

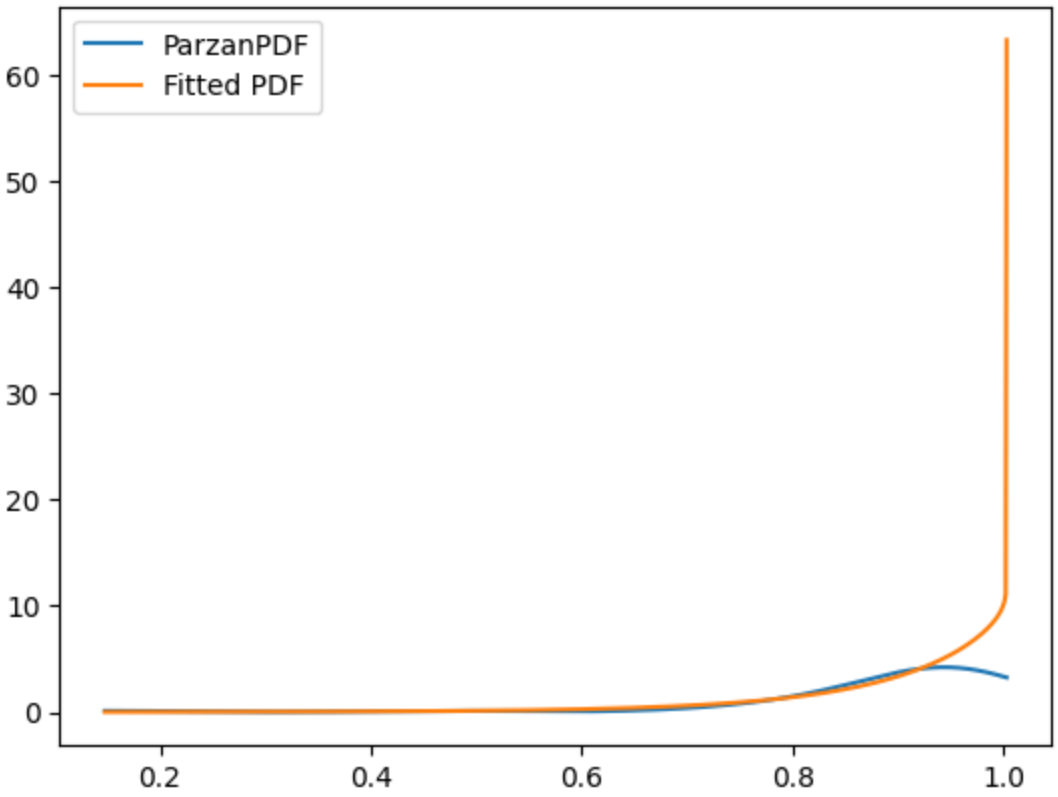
In [ ]:
```
pd_n_50 = feature_pdfs['n_50']
samplePDF_n_50 = samplePDF(df['n_50'], pd_n_50)
PDF_n_50 = parzanPDF(df['n_50'], 0, 25, 0.9, pd_n_50)
```





In [ ]:
```
pd_x_500 = feature_pdfs['x_500']
PDF_x_500 = samplePDF(df['x_500'], pd_x_500)
parzanPDF_n_50 = parzanPDF(df['x_500'], 0, 2, 0.07, pd_x_500)
```


SamplePDF
Fitted PDF
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1.0


ParzanPDF
Fitted PDF
60
50
40
30
20
10
0
0.2
0.4
0.6
0.8
1.0

1.6 Self-entropy of an event and entropy of a distribution

We define a function that calculates the entropy of a sample distribution, based on a interploated parameterless PDF approximation (calculated with samplePDF and parzanPDF, resp.) and the well-known parameterized distribition (calculated with distribution_fitting). We also compute the Kullback-Leibler (KL) divergence in both directions.

For demonstration purpose, we implement numeric integration manually as auxiliary function. It comes in two variants: trapezoid and simpson's 3/8 rule.

In [ ]:
```
def trapez(f, a, b):
    N = int((b - a) * 100)
    x = np.linspace(a, b, N)
    dx = x[1] - x[0]
    y = np.zeros(N)
    for i in range(N):
        z = f(x[i])
        y[i] = z
    w = np.concatenate(([0.5], np.ones(N-2), [0.5]))
    I = np.sum(w * y) * dx
    return I
```

In [ ]:
```
def simpson(f, a, b):
    N = 3 * (b - a) * 100 + 1
    x = np.linspace(a, b, N)
    dx = x[1] - x[0]
    I = 0
    for i in range(0, N - 3, 3):
        z0 = f(x[i])
        z1 = f(x[i + 1])
        z2 = f(x[i + 2])
        z3 = f(x[i + 3])
        I += z0 + 3 * z1 + 3 * z2 + z3
    I *= (3 * dx / 8)
    return I
```

In [ ]:
```
def ite(b,t,f):
    if b:
        return t
    return f
```

In [ ]:
```
import math

a = -25
b = 25
f1  = lambda x : ite(PDF_n_50(x) <= 0, lambda x: 0, lambda x: PDF_n_50(x))(x)
entropy1 = lambda x : -1 * ite(PDF_n_50(x) <= 0, lambda x: 0, lambda x: PDF_n_50(x)

Integral1 = trapez(f1, a, b)
Entropy1 = trapez(entropy1, a, b)

print(f'Integral1 = {Integral1}')
print(f'Entropy1 = {Entropy1}')
```

```
Integral1 = 0.9983712591471883
Entropy1 = 2.394884678336957
```

In [ ]:
```
f2  = lambda x : ite(pd_n_50.calc(x) <= 0, lambda x:0, lambda x:pd_n_50.calc(x))(x)
entropy2 = lambda x : -1 * ite(pd_n_50.calc(x) <= 0, lambda x:0, lambda x:pd_n_50.c

Integral2 = trapez(f2, a, b)
Entropy2 = trapez(entropy2, a, b)

print(f'Integral2 = {Integral2}')
print(f'Entropy2 = {Entropy2}')
```

```
Integral2 = 0.99778015999006
Entropy2 = 2.365217901946731
```

In [ ]:
```
divergence_1_2 = lambda x : ite(PDF_n_50(x) <= 0, lambda x:0, lambda x:PDF_n_50(x)

Divergence_1_2_trapez = trapez(divergence_1_2, a, b)
Divergence_1_2_simposon = simpson(divergence_1_2, a, b)

print(f'Divergence_1_2_trapez = {Divergence_1_2_trapez}')
print(f'Divergence_1_2_simpson = {Divergence_1_2_simposon}')
```

```
Divergence_1_2_trapez = 0.09852748898002434
Divergence_1_2_simpson = 0.09852759548592312
```

In [ ]:
```
eps = 1e-32

divergence_2_1 = lambda x : ite(PDF_n_50(x) <= 0,
                                lambda x: 0,
                                lambda x:pd_n_50.calc(x) * ( math.log(pd_n_50.calc(

Divergence_2_1_trapez = trapez(divergence_2_1, a, b)
Divergence_2_1_simposon = simpson(divergence_2_1, a, b)

print(f'Divergence_2_1_trapez = {Divergence_2_1_trapez}')
print(f'Divergence_2_1_simpson = {Divergence_2_1_simposon}')
```

```
Divergence_2_1_trapez = 0.11802125599756384
Divergence_2_1_simpson = 0.11802131642208097
```

Why is the entropy of N50 assuming is proportional to the interpolated PDF different from the one assuming it is proportional to the fitted PDF? Why are the KL divergences different? Your answers go here.

Answer: The fitted PDF comes with the assumption that there is a normal distribution, whereas the interpolated PDF was interpolated by finding any function that matches the datapoints best. So they have the potential to be different to each other. Hence the entropy is different.

Now that the entropy is different, we can say, that they are indeed different. The KL-divergence are assumed to be different, since the KL-divergence is an asymmertic function by definition.