Libraries

In [ ]:
```
import math
import numpy as np
import plotly.graph_objects as go

import plotly.io as pio
pio.renderers.default = "notebook_connected"
```

Constants

In [ ]:
```
# Gaussian quadrature constants
# Notation: (w_i, x_i)
GAUSS_LAGUERRE = [
    (0.458964, 0.222847),
    (0.417, 1.188932),
    (0.113373, 2.992736),
    (0.0103992, 5.775144),
    (0.000261017, 9.837467),
    (0.000000898548, 15.982874)
]

# GAUSS_LEGENDRE = [
#     (0.360762, -0.661209),
#     (0.467914, -0.238619),
#     (0.171324, -0.932469),
#     (0.171324, 0.932469),
#     (0.467914, 0.238619),
#     (0.360762, 0.661209)
# ]

# GAUSS_LEGENDRE = [
#     (0.2491470458134028, -0.1252334085114689),
#     (0.2491470458134028, 0.1252334085114689),
#     (0.2334925365383548, -0.3678314989981802),
#     (0.2334925365383548, 0.3678314989981802),
```

```python
#     (0.2031674267230659, -0.5873179542866175),
#     (0.2031674267230659, 0.5873179542866175),
#     (0.1600783285433462, -0.7699026741943047),
#     (0.1600783285433462, 0.7699026741943047),
#     (0.1069393259953184, -0.9041172563704749),
#     (0.1069393259953184, 0.9041172563704749),
#     (0.0471753363865118, -0.9815606342467192),
#     (0.0471753363865118, 0.9815606342467192)
# ]


GAUSS_LEGENDRE = [
    (0.0775059479784248, -0.0387724175060508),
    (0.0775059479784248, 0.0387724175060508),
    (0.077039818164248, -0.1160840706752552),
    (0.077039818164248, 0.1160840706752552),
    (0.0761103619006262, -0.1926975807013711),
    (0.0761103619006262, 0.1926975807013711),
    (0.0747231690579683, -0.2681521850072537),
    (0.0747231690579683, 0.2681521850072537),
    (0.0728865823958041, -0.3419940908257585),
    (0.0728865823958041, 0.3419940908257585),
    (0.0706116473912868, -0.413779204371605),
    (0.0706116473912868, 0.413779204371605),
    (0.0679120458152339, -0.4830758016861787),
    (0.0679120458152339, 0.4830758016861787),
    (0.064804013456601, -0.5494671250951282),
    (0.064804013456601, 0.5494671250951282),
    (0.0613062424929289, -0.6125538896679802),
    (0.0613062424929289, 0.6125538896679802),
    (0.0574397690993916, -0.6719566846141796),
    (0.0574397690993916, 0.6719566846141796),
    (0.0532278469839368, -0.7273182551899271),
    (0.0532278469839368, 0.7273182551899271),
    (0.0486958076350722, -0.7783056514265194),
    (0.0486958076350722, 0.7783056514265194),
    (0.0438709081856733, -0.8246122308333117),
    (0.0438709081856733, 0.8246122308333117),
    (0.038782167974472, -0.8659595032122595),
    (0.038782167974472, 0.8659595032122595),
```

```
    (0.0334601952825478, -0.9020988069688743),
    (0.0334601952825478, 0.9020988069688743),
    (0.0279370069800234, -0.9328128082786765),
    (0.0279370069800234, 0.9328128082786765),
    (0.022245849194167, -0.9579168192137917),
    (0.022245849194167, 0.9579168192137917),
    (0.0164210583819079, -0.9772599499837743),
    (0.0164210583819079, 0.9772599499837743),
    (0.0104982845311528, -0.990726238699457),
    (0.0104982845311528, 0.990726238699457),
    (0.0045212770985332, -0.9982377097105593),
    (0.0045212770985332, 0.9982377097105593)
]
```

PDF, CDF, plotting functions

In [ ]:
```
# Weibull distribution function -- uses Gauss-Laguerre quadratures
def f_Weibull (t, k, mu):
    denomArea = 0
    for node in GAUSS_LAGUERRE:
        denomArea += node[0] * (node[1]**(1/k))

    lbda = mu / denomArea

    return (k/lbda) * ((t/lbda)**(k-1)) * (math.e ** (-(t/lbda)**k))
```

In [ ]:
```
# Integral of Weibull function -- uses Gauss-Legendre quadratures
def cdf (t, k, mu):
    pdf_area = 0
    for node in GAUSS_LEGENDRE:
        u = (t/2)*node[1] + t/2
        du = t/2
        pdf_area += node[0] * f_Weibull(u, k, mu) * du

    return pdf_area
```

In [ ]:
```
def plot (x05, x1, x2, y05, y1, y2, xtitle, ytitle, title, printAsymptote = False):
    fig = go.Figure()
```

```
    fig.add_traces([
        go.Scatter(x=x05, y=y05, mode='lines', marker = {'color' : 'blue'}, name="k = 0.5"),
        go.Scatter(x=x1, y=y1, mode='lines', marker = {'color' : 'red'}, name="k = 1"),
        go.Scatter(x=x2, y=y2, mode='lines', marker = {'color' : 'magenta'}, name="k = 2")
    ])

    if printAsymptote:
        fig.add_traces([go.Scatter(x=[i/100 for i in range(0, 101, 1)], y=[78 for i in range(0, 101, 1)], line_dash ='dash', marker = {'col

    fig.update_layout(
        title_text=title,
        xaxis_title=xtitle,
        yaxis_title=ytitle,
        height=1080*0.5,
        width=1920*0.6,
        font_family="CMU Serif",
        font_size=15,
        title_font_size=25,
        font_color="#0e0f11",
        margin=dict(t=120, b=80)
    )
    fig.show()

# def plot_inverse (x05, x1, x2, y05, y1, y2, xtitle, ytitle, title):
#     fig = go.Figure()
#     fig.add_traces([
#         go.Scatter(x=x05, y=y05, mode='lines', marker = {'color' : 'blue'}, name="k = 0.5"),
#         go.Scatter(x=x1, y=y1, mode='lines', marker = {'color' : 'red'}, name="k = 1"),
#         go.Scatter(x=x2, y=y2, mode='lines', marker = {'color' : 'magenta'}, name="k = 2"),
#         go.Scatter(x=[i/100 for i in range(0, 101, 1)], y=[78 for i in range(0, 101, 1)], line_dash ='dash', marker = {'color' : 'orange'
#     ])
#     fig.update_layout(
#         title_text=title,
#         xaxis_title=xtitle,
#         yaxis_title=ytitle,
#         height=1080*0.5,
#         width=1920*0.6,
#         font_family="CMU Serif",
#         font_size=15,
#         title_font_size=25,
```

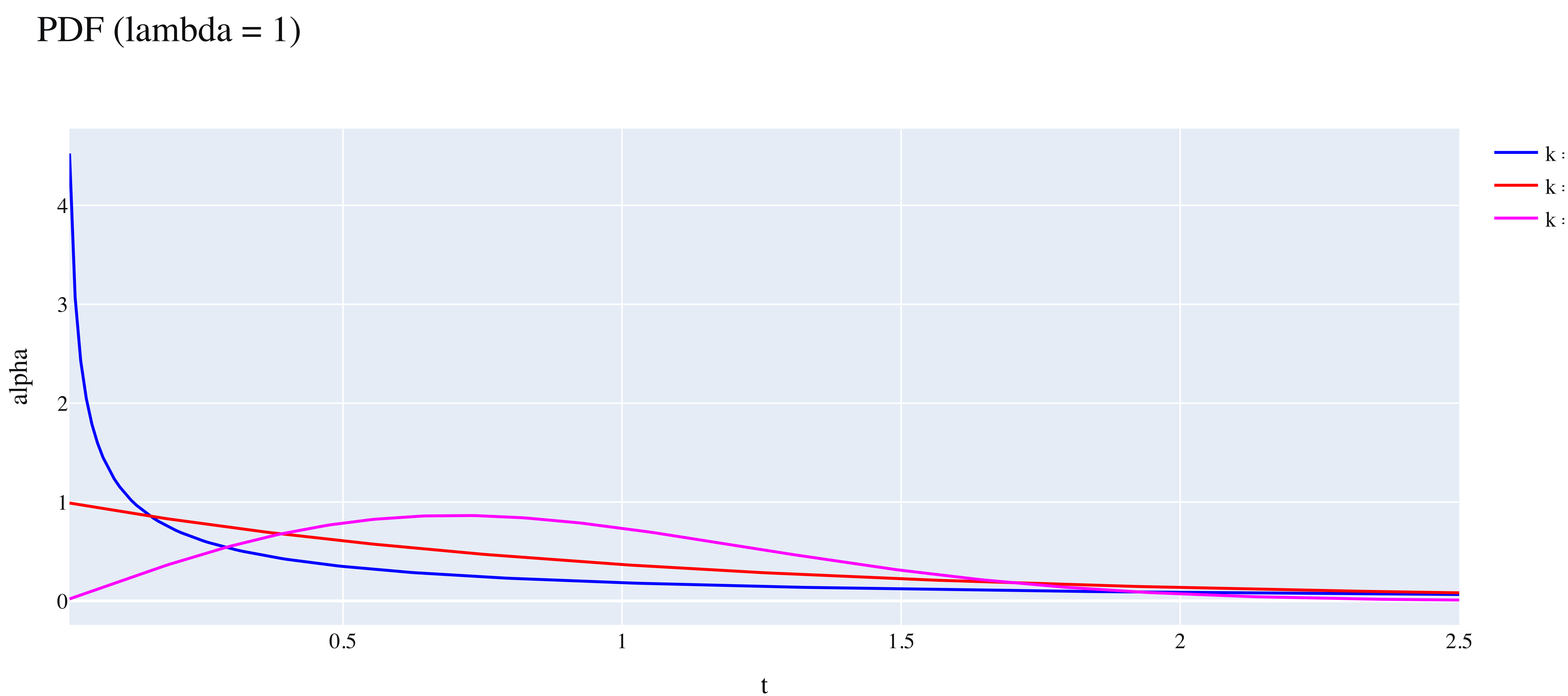
```
#         font_color="#0e0f11",
#         margin=dict(t=120, b=80)
#     )
#     fig.show()
```

Graphs (lambda = 1)

PDF

In [ ]:

```
x, y05, y1, y2 = [], [], [], []

for t in range(1, 251, 1):
    t /= 100
    x.append(t)
    y05.append(f_Weibull(t, 0.5, 2))
    y1.append(f_Weibull(t, 1, 1))
    y2.append(f_Weibull(t, 2, math.sqrt(math.pi)/2))

plot(x, x, x, y05, y1, y2, "t", "alpha", "PDF (lambda = 1)")
```



CDF, survival, inverse survival

In [ ]:
```
# # @title CDF values
# x, y05, y1, y2 = [], [], [], []
# for t in range(1, 251, 1):
#     t /= 100
#     x.append(t)
```

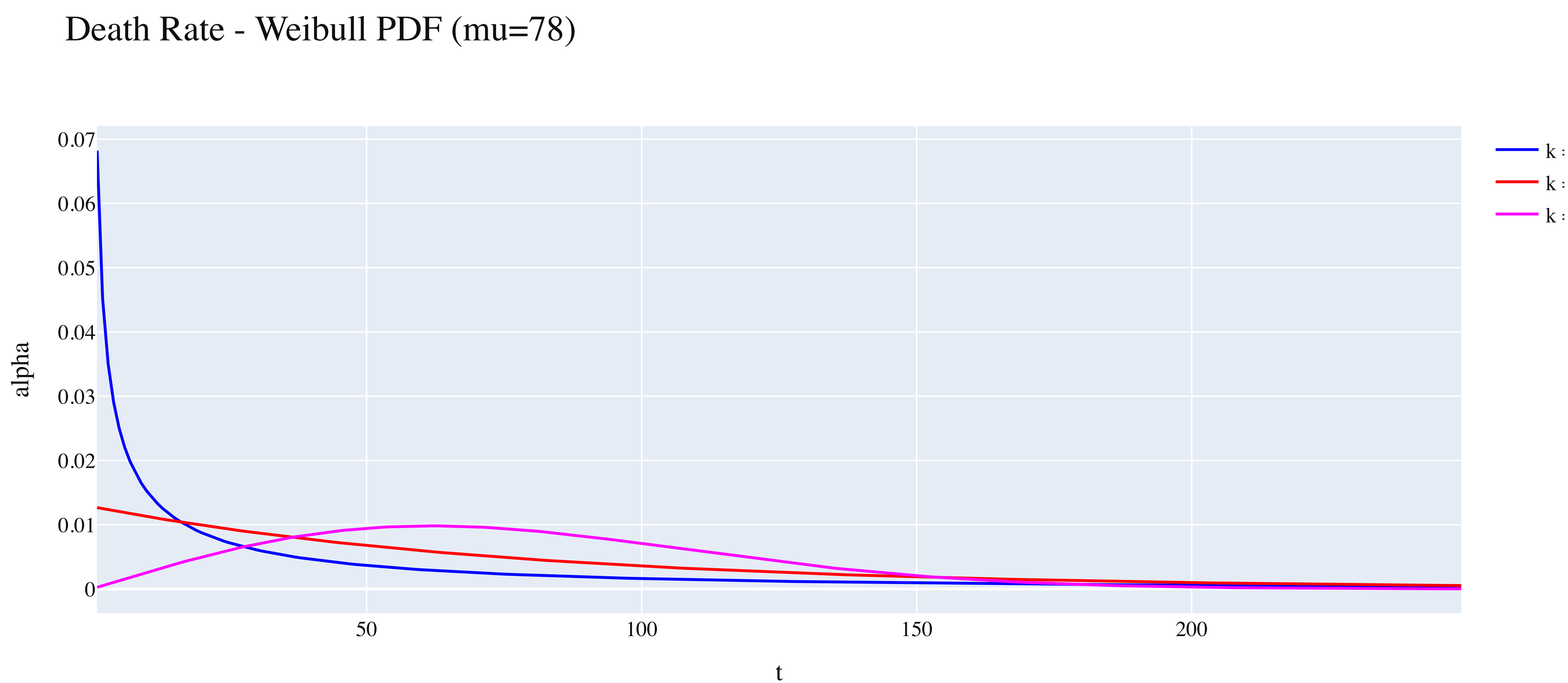
```
#     y05.append(cdf(t, 0.5, 2))
#     y1.append(cdf(t, 1, 1))
#     y2.append(cdf(t, 2, math.sqrt(math.pi)/2))
# plot(x, x, x, y05, y1, y2, "t", "alpha", "CDF (lambda = 1)")

# Survival function
# plot(x, x, x, [1-i for i in y05], [1-i for i in y1], [1-i for i in y2], "t", "alpha", "S(t), lambda = 1")

# Inverse survival function
# plot([1-i for i in y05], [1-i for i in y1], [1-i for i in y2], x, x, x, "alpha", "t", "Inverse survival (lambda = 1)")
```

Graphs (mu = 78)

PDF

In [ ]:

```
x, y05, y1, y2 = [], [], [], []
for t in range(1, 250, 1):
    x.append(t)
    y05.append(f_Weibull(t, 0.5, 78))
    y1.append(f_Weibull(t, 1, 78))
    y2.append(f_Weibull(t, 2, 78))

# print(x[:101:])
# print(y05[:101:])
# print(y1[:101:])
# print(y2[:101:])

plot(x, x, x, y05, y1, y2, "t", "alpha", "Death Rate - Weibull PDF (mu=78)")
```

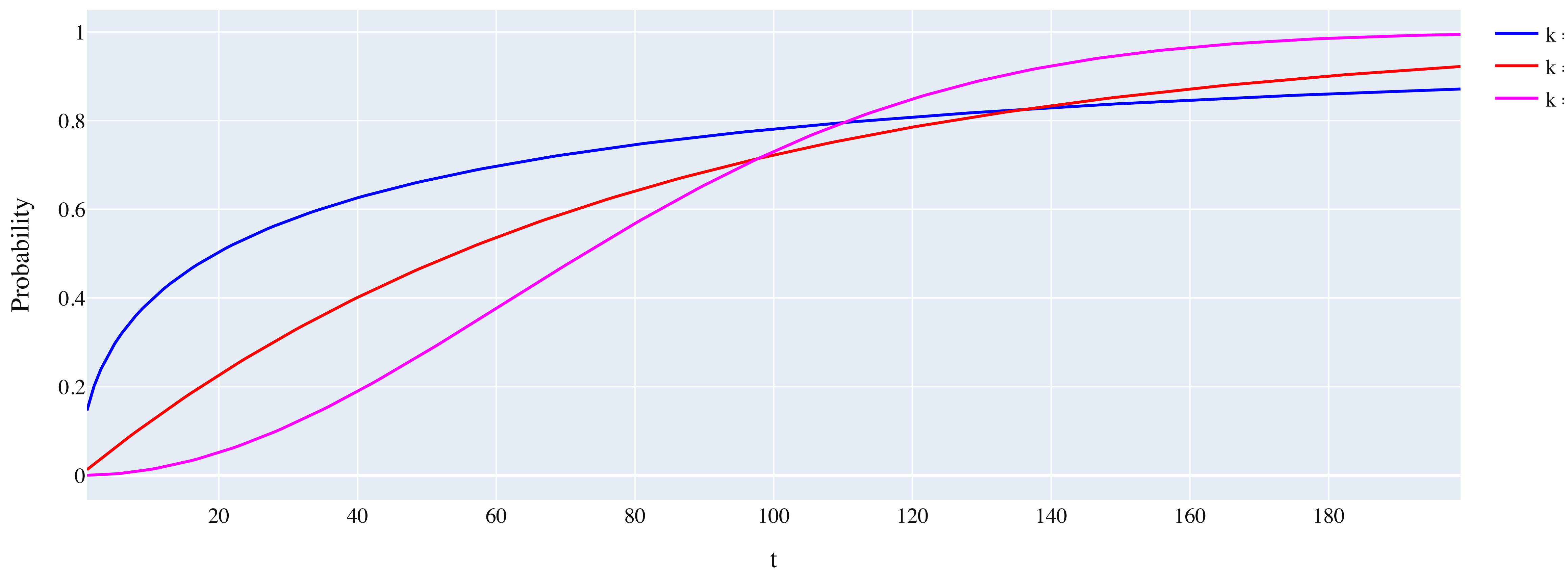


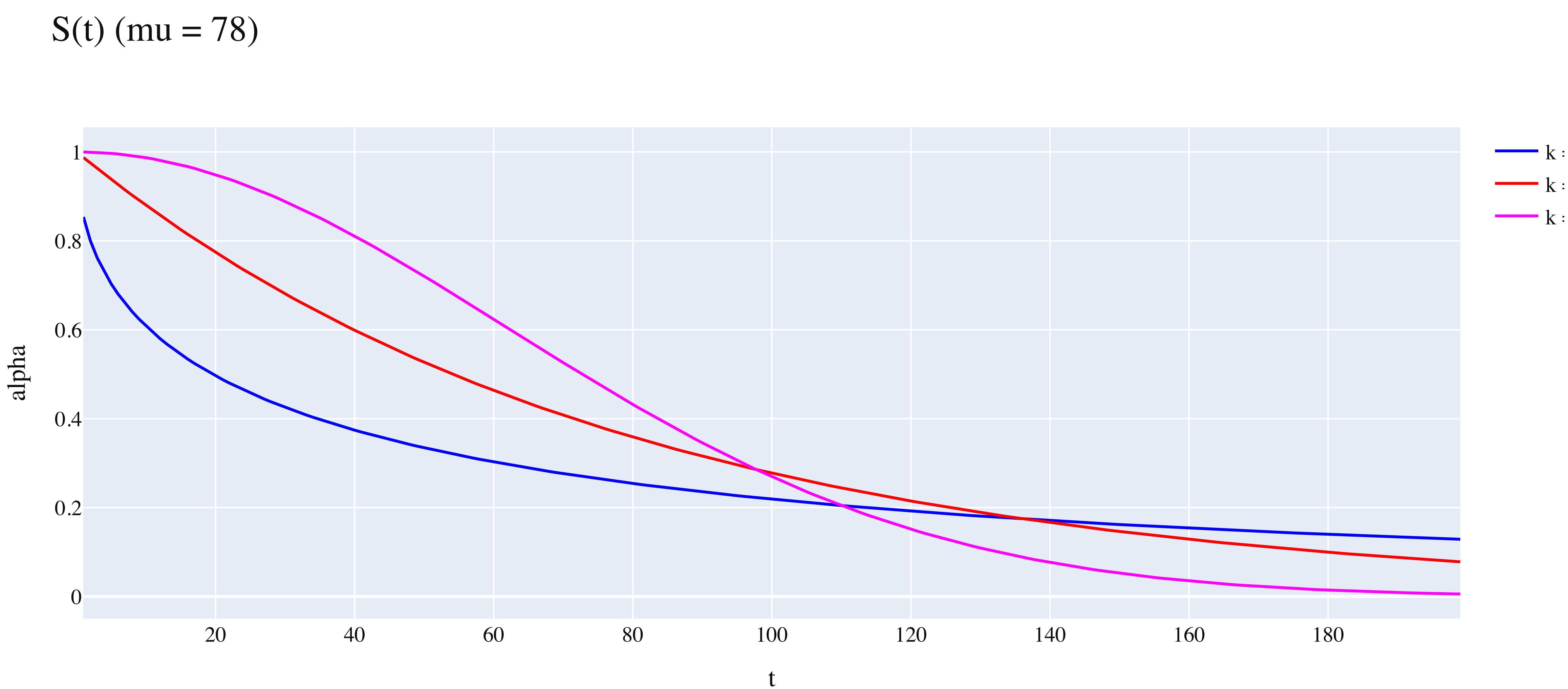
CDF, survival, inverse survival

```
x, y05, y1, y2 = [], [], [], []
for t in range(1, 200, 1):
    x.append(t)
    y05.append(cdf(t, 0.5, 78))
    y1.append(cdf(t, 1, 78))
```

```python
    y2.append(cdf(t, 2, 78))

# CDF mu = 78
plot(x, x, x, y05, y1, y2, "t", "Probability", "Death Rate - Weibull CDF (mu=78)")

# Survival mu = 78
plot(x, x, x, [1-i for i in y05], [1-i for i in y1], [1-i for i in y2], "t", "alpha", "S(t) (mu = 78)")

# Inverse survival mu = 78
# plot([1-i for i in y05], [1-i for i in y1], [1-i for i in y2], x, x, x, "alpha", "t", "inv surv (mu = 78)")
```

Death Rate - Weibull CDF (mu=78)





Root-finding problem

In [ ]:
```
# # Inverse survivability -- solved as root-finding problem using bisection method
# def bisection (a, b, alpha, mu, k):
#     f = lambda t : 1 - cdf(t, k, mu) - alpha
```

```
#     tol = 1.0e-9

#     fa = f(a)
#     fb = f(b)

#     if fa == 0.0: return a

#     if fb == 0.0: return b

#     # print("-> cdf(" + str(a) + ") - " + str(alpha) + " = " + str(fa))
#     # print("-> cdf(" + str(b) + ") - " + str(alpha) + " = " + str(fb))

#     if np.sign(fa) == np.sign(fb):
#         # print("---> NO ROOT AT :", alpha)
#         return None

#     n = int (math.ceil (math.log(abs(b-a)/tol) / math.log(2.0)))

#     # print("---> iterations =", n)

#     for i in range(n):
#         # print("-> CDF(" + str(a) + ") - " + str(alpha) + " = " + str(fa))
#         # print("-> CDF(" + str(b) + ") - " + str(alpha) + " = " + str(fb))

#         c = 0.5 * (a + b)
#         fc = f(c)

#         if fc == 0.0: return c
#         if np.sign(fa) != np.sign(fc):
#             b = c
#             fb = fc
#         elif np.sign(fb) != np.sign(fc):
#             a = c
#             fa = fc

#     # print("---> returning c =", 0.5 * (a+b))
#     return 0.5 * (a+b)

def regula_falsi(a, b, alpha, mu, k):
    f = lambda t : 1 - cdf(t, k, mu) - alpha
```

```
    tol = 1.0e-9
    fa = f(a)
    fb = f(b)

    if fa == 0.0: return a
    if fb == 0.0: return b
    if np.sign(fa) == np.sign(fb): return None

    while True:
        c = b - fb * (b-a)/(fb-fa)
        fc = f(c)

        if fc == 0.0 or abs(fc) < tol: return c
        elif fa * fc < 0:
            b = c
        else:
            a = c
```

Evaluation

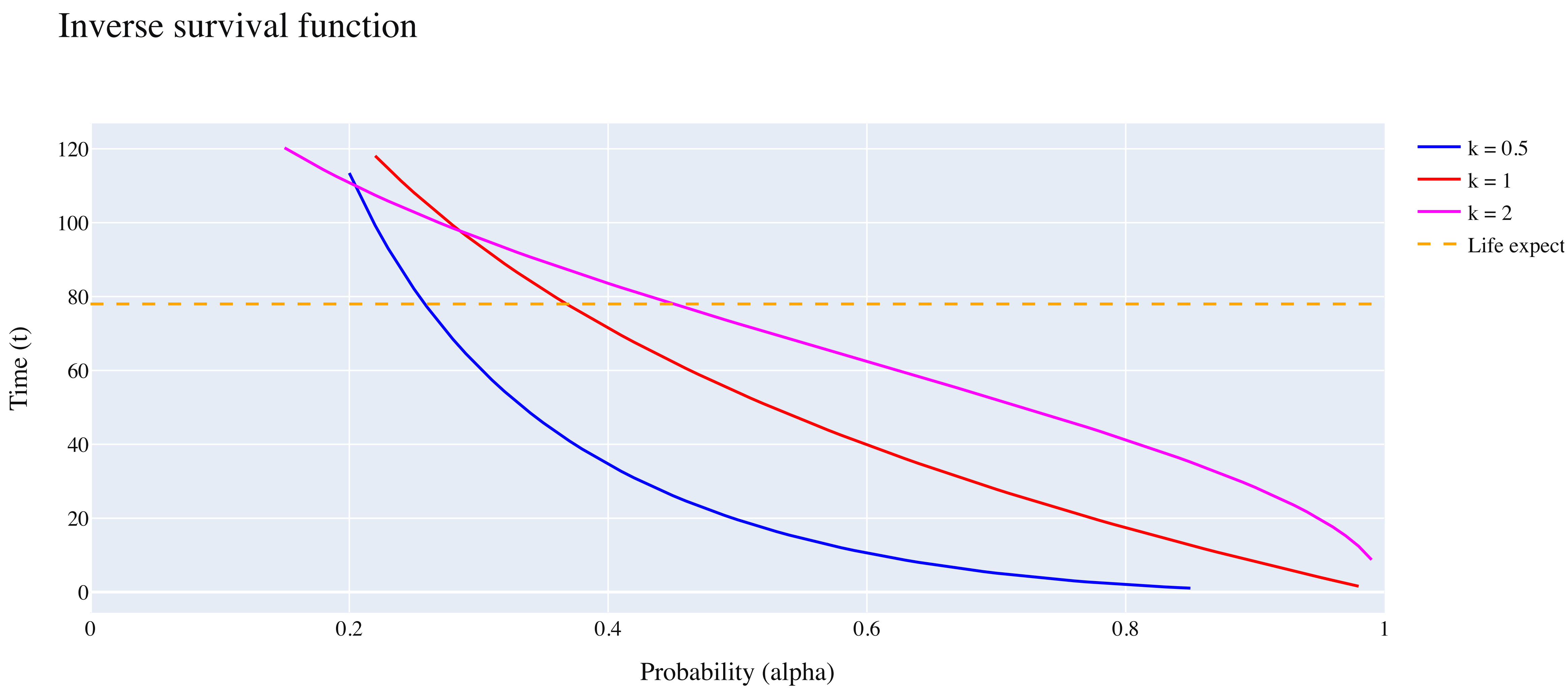
In [ ]:
```
# Lists for values storing
x_values, y_t_05, y_t_1, y_t_2 = [], [], [], []

# Average life expectancy
mu = 78
```

In [ ]:
```
# For alpha in (0,1) with 0.01 step
for alpha in range(1, 101, 1):
    alpha /= 100
    x_values.append(alpha)
    y_t_05.append(regula_falsi(1, 121, alpha, mu, 0.5))
    y_t_1.append(regula_falsi(1, 121, alpha, mu, 1))
    y_t_2.append(regula_falsi(1, 121, alpha, mu, 2))
```

Plotting

In [ ]:
```
# @title Inverse survivability plot
```

```
plot(x_values, x_values, x_values, y_t_05, y_t_1, y_t_2, "Probability (alpha)", "Time (t)", "Inverse survival function", True)
```




```
In [ ]:
```