Survival Analysis for Individual Patient Data of Acid Sphingomyelinase Deficiency (ASMD)

Background

Sanofi engaged ISMS to develop a patient pool model for Acid Sphingomyelinase Deficiency (ASMD). This disease exhibits significant variability in key parameters, including clinical severity, age of onset, and survival. The variability extends across subtypes (A, A/B, and B), with subtype B further differing by pediatric versus adult onset.

Using mean values to model these parameters would overlook this heterogeneity, leading to loss of detail in patient pool classifications (onset, diagnosis, survival) across age groups. To address this, a probabilistic approach was implemented to better capture variability within and between subgroups.

Input Data

The age-at-diagnosis dataset was extracted from the article.

In [2]:
```
# Load age at diagnosis dataset in Excel
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import lifelines
```

In [3]:
```
diagnosis_age_df = pd.read_excel('age-at-diagnosis-asmd.xlsx')
```

In [23]:
```
diagnosis_age_df.head(54)
```

Out[23]:

	Time since birth (year)	Diagnosis	AB	CH
0	0.10	1	1	0
1	0.28	1	1	0
2	0.36	1	1	0
3	0.45	1	1	0
4	0.45	1	1	0
5	0.54	1	1	0
6	0.81	1	1	0
7	0.98	1	1	0
8	1.16	1	1	0
9	1.42	1	1	0
10	1.78	1	1	0
11	2.04	1	1	0
12	2.13	1	1	0
13	2.48	1	1	0
14	2.93	1	1	0
15	5.93	1	1	0
16	10.82	1	1	0
17	0.17	1	0	1
18	0.28	1	0	1
19	0.49	1	0	1
20	0.82	1	0	1
21	0.91	1	0	1
22	1.44	1	0	1
23	1.65	1	0	1
24	1.76	1	0	1
25	1.76	1	0	1
26	1.97	1	0	1
27	1.97	1	0	1
28	2.82	1	0	1
29	3.03	1	0	1
30	3.05	1	0	1
31	3.05	1	0	1
32	3.14	1	0	1

	Time since birth (year)	Diagnosis	AB	CH
33	4.94	1	0	1
34	5.04	1	0	1
35	7.02	1	0	1
36	8.02	1	0	1
37	8.97	1	0	1
38	16.03	1	0	1
39	18.85	1	0	0
40	21.98	1	0	0
41	25.04	1	0	0
42	29.01	1	0	0
43	30.99	1	0	0
44	35.88	1	0	0
45	42.90	1	0	0
46	44.89	1	0	0
47	47.94	1	0	0
48	48.09	1	0	0
49	50.99	1	0	0
50	51.15	1	0	0
51	56.03	1	0	0
52	66.65	1	0	0
53	77.58	1	0	0

In [22]:
```
diagnosis_age_df.shape
```

Out[22]:
```
(54, 4)
```

Perform KM survival analysis

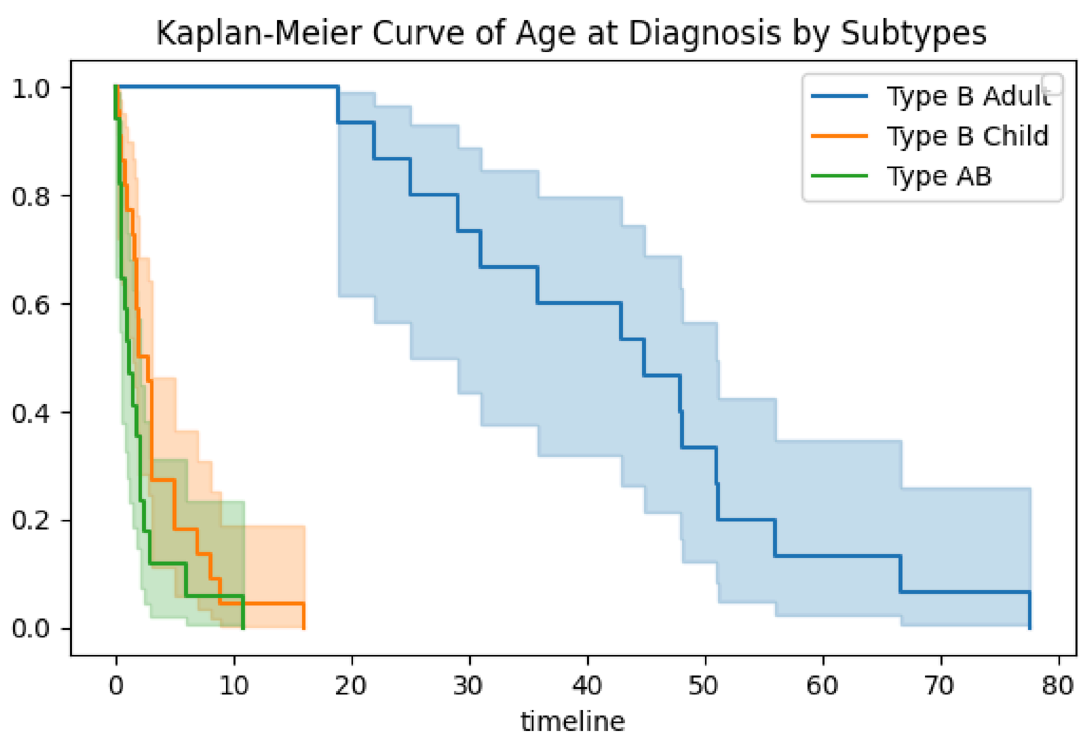
In [5]:
```
from lifelines import KaplanMeierFitter # for Kaplan Meier estimator
from matplotlib import pyplot as plt
from lifelines.utils import median_survival_times
```

In [6]:
```
ax = plt.subplot(111) # To put the legend outside the plot

# Subset by subtypes - index the rows with the condition
# AB == 0 and CH == 0 means type B adult
# AB == 0 and CH == 1 means type B child
# AB == 1 means type AB
type_b_adult = (diagnosis_age_df["AB"] == 0) & (diagnosis_age_df["CH"] == 0)
type_b_child = (diagnosis_age_df["AB"] == 0) & (diagnosis_age_df["CH"] == 1)
```

```
type_ab = (diagnosis_age_df["AB"] == 1)

# Instantiate the class to create an object
kmf = KaplanMeierFitter()

# Define the time and event
T = diagnosis_age_df["Time since birth (year)"]
E = diagnosis_age_df["Diagnosis"]

# Fit the data into the model
kmf.fit(T[type_b_adult], event_observed=E[type_b_adult], label="Type B Adult")
kmf.plot_survival_function(ax=ax, at_risk_counts=True)

kmf.fit(T[type_b_child], event_observed=E[type_b_child], label="Type B Child")
kmf.plot_survival_function(ax=ax, at_risk_counts=False)

kmf.fit(T[type_ab], event_observed=E[type_ab], label="Type AB")
kmf.plot_survival_function(ax=ax, at_risk_counts=False)

plt.tight_layout()
plt.title("Kaplan-Meier Curve of Age at Diagnosis by Subtypes")
plt.legend(loc="best")
```

```
/var/folders/b8/9ymtxc2j7rb00xx34s753cwc0000gn/T/ipykernel_57926/41739333.py:30: UserWarning: No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
  plt.legend(loc="best")
```

Out[6]: `<matplotlib.legend.Legend at 0x13ed89b20>`



Type B Adult									
At risk	15	15	14	11	9	5	2	1	0
Censored	0	0	0	0	0	0	0	0	0
Events	0	0	1	4	6	10	13	14	15

Compare the survival curves of different subtypes using log-rank test

In [7]:
```
from lifelines.statistics import logrank_test
```

In [8]:
```
# Test at the 5% significance level
diag_log_rank = logrank_test(T[type_b_adult],
                             T[type_b_child],
                             E[type_b_adult],
                             E[type_b_child],
                             alpha=0.05)
```

In [9]:
```
diag_log_rank.print_summary()
```

t_0	-1
null_distribution	chi squared
degrees_of_freedom	1
alpha	0.05
test_name	logrank_test

	test_statistic	p	-log2(p)
0	37.05	<0.005	29.70

Perform parametric regression analysis to estimate the survival function

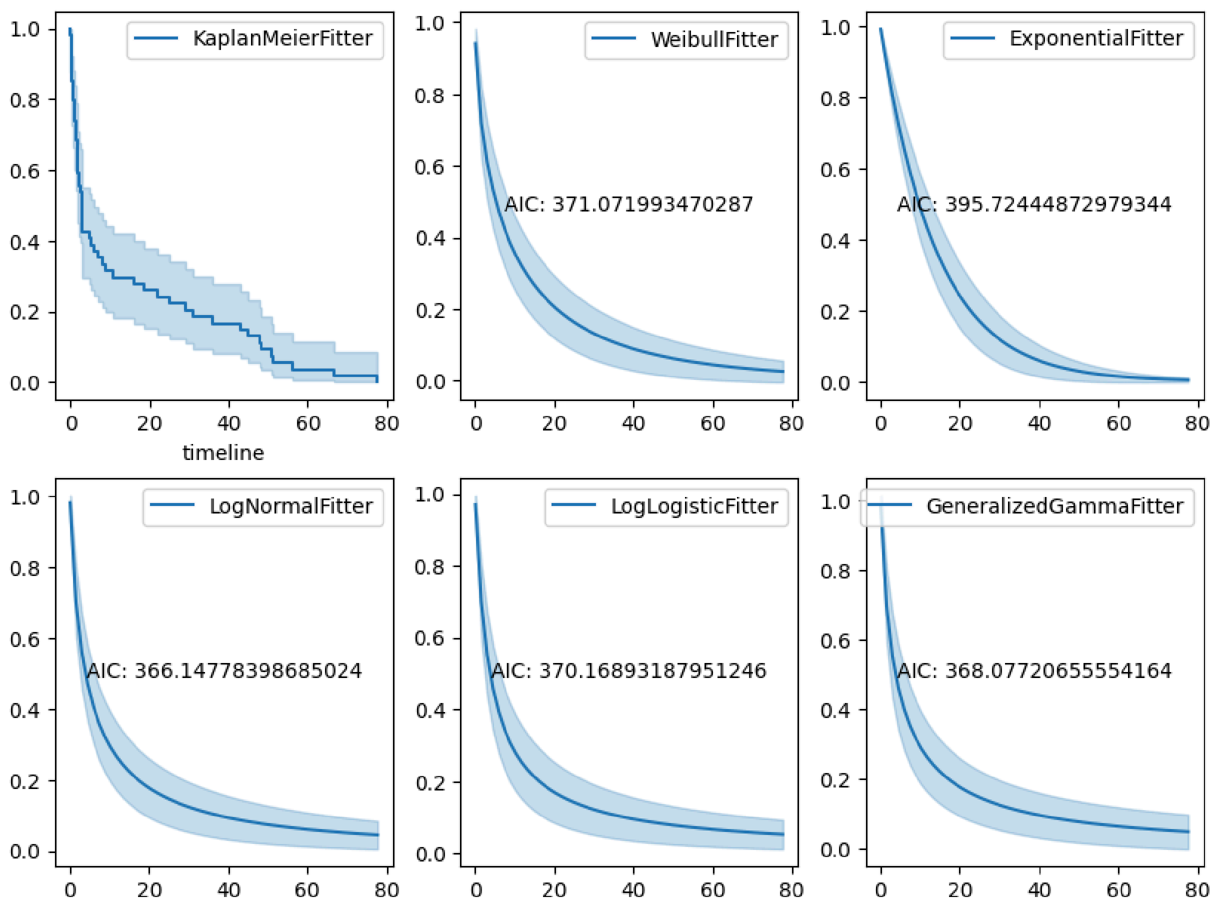
Fit various parametric models to the data and compare the goodness of fit using AIC, including Weibull, Gompertz, Exponential, and Log-normal models.

In [10]:
```
from lifelines import (WeibullFitter, ExponentialFitter, LogNormalFitter, LogLogi
```

In [11]:
```
fig, axes = plt.subplots(2, 3, figsize=(10, 7.5))

# Kaplan-Meier Fitter
kmf_surv = KaplanMeierFitter().fit(T, E, label='KaplanMeierFitter')
kmf_surv.plot_survival_function(ax=axes[0][0])

# Weibull Fitter
wbf_surv = WeibullFitter().fit(T, E, label='WeibullFitter')
wbf_surv.plot_survival_function(ax=axes[0][1])
# Print AIC in plot
axes[0][1].text(0.5, 0.5, f"AIC: {wbf_surv.AIC_}", horizontalalignment='center',

# Exponential Fitter
exf_surv = ExponentialFitter().fit(T, E, label='ExponentialFitter')
exf_surv.plot_survival_function(ax=axes[0][2])
axes[0][2].text(0.5, 0.5, f"AIC: {exf_surv.AIC_}", horizontalalignment='center',
```

```
# Log-Normal Fitter
lnf_surv = LogNormalFitter().fit(T, E, label='LogNormalFitter')
lnf_surv.plot_survival_function(ax=axes[1][0])
axes[1][0].text(0.5, 0.5, f"AIC: {lnf_surv.AIC_}", horizontalalignment='center',

# Log Logistic Fitter
llf_surv = LogLogisticFitter().fit(T, E, label='LogLogisticFitter')
llf_surv.plot_survival_function(ax=axes[1][1])
axes[1][1].text(0.5, 0.5, f"AIC: {llf_surv.AIC_}", horizontalalignment='center',

# Generalized Gamma Fitter
ggf_surv = GeneralizedGammaFitter().fit(T, E, label='GeneralizedGammaFitter')
ggf_surv.plot_survival_function(ax=axes[1][2])
axes[1][2].text(0.5, 0.5, f"AIC: {ggf_surv.AIC_}", horizontalalignment='center',
```

Out[11]: `Text(0.5, 0.5, 'AIC: 368.07720655554164')`



In [12]:
```
# Manually compare AIC values
models = {'Weibull': wbf_surv.AIC_, 'Exponential': exf_surv.AIC_, 'Log-Normal': l
min_model = min(models, key=models.get)
print(f"Model with minimum AIC: {min_model}")
```

```
Model with minimum AIC: Log-Normal
```

Select the best model and estimate the survival function

In [13]:
```
from lifelines.utils import find_best_parametric_model
```

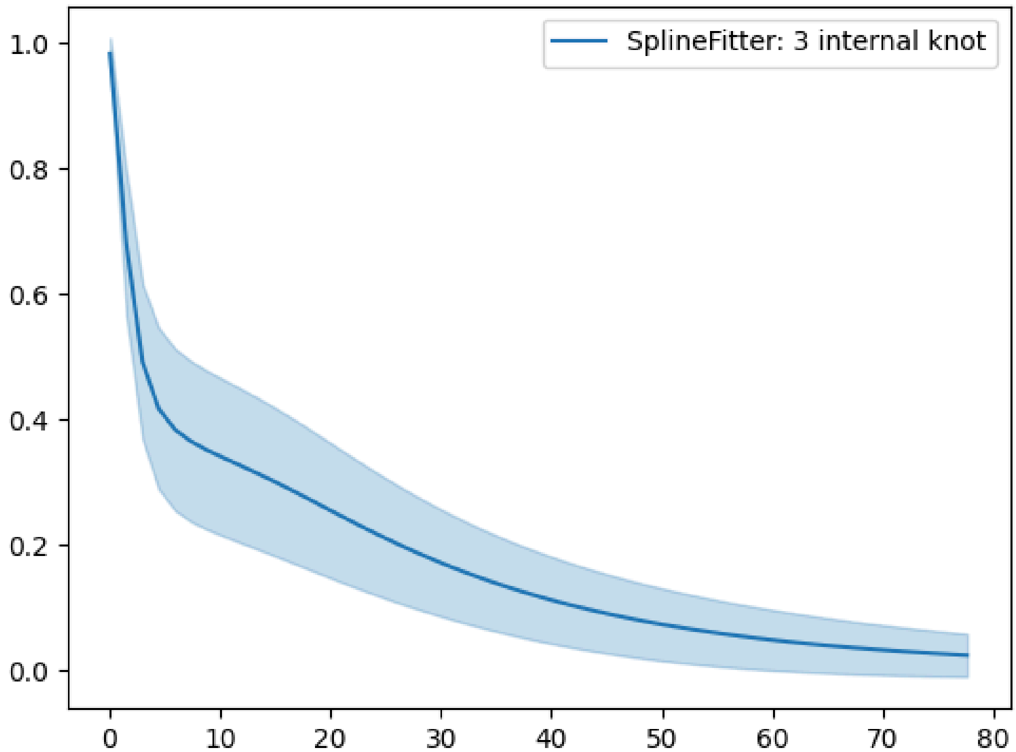
In [14]:
```
best_model, best_aic = find_best_parametric_model(T, E, scoring_method='AIC')
```

In [15]:
```
print(best_model, best_aic)
```

```
<lifelines.SplineFitter:"SplineFitter: 3 internal knot", fitted with 54 total obs
ervations, 0 right-censored observations> 359.53924981090427
```

In [16]:
```
best_model.plot_survival_function() # Plot the survival function of the best mode
```

Out[16]:
```
<Axes: >
```



find_best_parametric_model() can select non-parametric SplineFitter if no parametric model fits well.

Estimate the survival function for each subtype using log-normal model

In [17]:
```
from lifelines import LogNormalAFTFitter # Accelerated Failure Time model

lognorm_aft = LogNormalAFTFitter()  # Weibull Accelerated Failure Time model
lognorm_aft.fit(diagnosis_age_df, duration_col='Time since birth (year)', event_c
```

Out[17]:
```
<lifelines.LogNormalAFTFitter: fitted with 54 total observations, 0 right-censor
ed observations>
```

In [18]:
```
lognorm_aft.print_summary()
```

model	lifelines.LogNormalAFTFitter
duration col	'Time since birth (year)'
event col	'Diagnosis'
number of observations	54
number of events observed	54
log-likelihood	-148.72
time fit was run	2025-02-28 23:34:15 UTC

		coef	exp(coef)	se(coef)	coef lower 95%	coef upper 95%	exp(coef) lower 95%	exp(coef) upper 95%	cmp to	
mu_	**Intercept**	3.69	40.12	0.25	3.20	4.18	24.63	65.35	0.00	14.8
	AB	-3.59	0.03	0.34	-4.26	-2.93	0.01	0.05	0.00	-10.5
	CH	-2.92	0.05	0.32	-3.55	-2.28	0.03	0.10	0.00	-9.0
sigma_	**Intercept**	-0.04	0.96	0.10	-0.23	0.15	0.80	1.16	0.00	-0.3

Concordance	0.75
AIC	305.43
log-likelihood ratio test	64.72 on 2 df
-log2(p) of ll-ratio test	46.68

Get survival (diagnosis in this case) probability for each subtype

In [49]:

```
# Prepare a dataframe for prediction
new_data = pd.DataFrame({'AB': [0, 0, 1],  # Type B adult, Type B child, Type AB
                         'CH': [0, 1, 0]})

# Predict the survival function at 0 to 100 years
future_times = list(np.arange(0, 100, 0.5))

# Predict the survival function
survival_probs = lognorm_aft.predict_survival_function(new_data[['AB', 'CH']], ti
```

In [50]:

```
print(survival_probs)
```

```
             0         1         2
0.0   1.000000  1.000000  1.000000
0.5   0.999997  0.935983  0.793725
1.0   0.999936  0.789023  0.540046
1.5   0.999673  0.648964  0.374499
2.0   0.999064  0.533539  0.268183
...        ...       ...       ...
97.5  0.178538  0.000040  0.000002
98.0  0.177157  0.000039  0.000002
98.5  0.175789  0.000038  0.000002
99.0  0.174435  0.000037  0.000002
99.5  0.173094  0.000036  0.000002

[200 rows x 3 columns]
```

In [51]:
```
# Plot the survival function
ax_lognormal = survival_probs.plot(legend=True)

# Rename legend
ax_lognormal.legend(['Type B Adult', 'Type B Child', 'Type AB'])

plt.title("Survival Function of Log-Normal AFT Model")
plt.ylabel("Survival Probability")
plt.xlabel("Time since birth (year)")
plt.show()
```