Frailty and Survival Regression Models in Stan

Michael Issa

October 2024

Table of contents

1 Survival Regression Models

```
import arviz as az
import preliz as pz
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import warnings
import os

from cmdstanpy import CmdStanModel, from_csv

warnings.filterwarnings("ignore")

# Graphic configuration
c_light = "#DCBCBC"
c_light_highlight = "#C79999"
c_mid = "#B97C7C"
c_mid_highlight = "#A25050"
c_dark = "#8F2727"
c_dark_highlight = "#7C0000"

c_light_teal = "#6B8E8E"
c_mid_teal = "#487575"
c_dark_teal = "#1D4F4F"

RANDOM_SEED = 58583389
np.random.seed(RANDOM_SEED)
az.style.use("arviz-whitegrid")

plt.rcParams['font.family'] = 'serif'

plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['axes.titlesize'] = 12

plt.rcParams['axes.spines.top'] = False
plt.rcParams['axes.spines.right'] = False
plt.rcParams['axes.spines.left'] = True
plt.rcParams['axes.spines.bottom'] = True

plt.rcParams['axes.xmargin'] = 0
```

```
plt.rcParams['axes.ymargin'] = 0

plt.subplots_adjust(left=0.15, bottom=0.15, right=0.9, top=0.85)

current_working_directory = os.getcwd()
p_dir = os.path.join(os.path.dirname(current_working_directory), "Frailty_and_Survival_Regres
```

```
<Figure size 720x480 with 0 Axes>
```

2 Exploration of Data

```
retention_df = pd.read_csv(os.path.join(current_working_directory, 'data', 'job_retention.csv

retention_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3770 entries, 0 to 3769
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   gender     3770 non-null   object
 1   field      3770 non-null   object
 2   level      3770 non-null   object
 3   sentiment  3770 non-null   int64
 4   intention  3770 non-null   int64
 5   left       3770 non-null   int64
 6   month      3770 non-null   int64
dtypes: int64(4), object(3)
memory usage: 206.3+ KB
```

```
dummies = pd.concat(
    [
        pd.get_dummies(retention_df["gender"], drop_first=True),
        pd.get_dummies(retention_df["level"], drop_first=True),
        pd.get_dummies(retention_df["field"], drop_first=True),
    ],
    axis=1,
).rename({"M": "Male"}, axis=1)
```

```
retention_df = pd.concat([retention_df, dummies], axis=1).sort_values("Male").reset_index(dr
retention_df.head()
```

	gender	field	level	sentiment	intention	left	month	Male	Low	Medium
0	F	Education and Training	Low	8	5	0	12	False	True	False
1	F	Education and Training	Medium	8	3	1	11	False	False	True
2	F	Education and Training	Low	10	7	1	9	False	True	False
3	F	Education and Training	High	8	2	0	12	False	False	False
4	F	Education and Training	Low	8	8	0	12	False	True	False

```
# Stan data
stan_data_1 = {
  'N': len(retention_df['month']),
  'durations': retention_df['month'],
  'event_observed': retention_df['left'],
}

# Fit the model
KM_model_path = os.path.join(p_dir, 'models', 'kaplan_meier.stan')
KM_model_samples = os.path.join(p_dir, 'models', 'kaplan_meier.csv')
KM_model = CmdStanModel(stan_file=KM_model_path)
```

```
# Fit model
if os.path.exists(KM_model_samples):
    km_fit = from_csv(KM_model_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    km_fit = KM_model.sample(data=stan_data_1, seed=RANDOM_SEED, chains=4, iter_sampling=1,
    km_fit.save_csvfiles(KM_model_samples)
    print("Model run and saved.")
```

```
Model loaded from existing samples.
```

```
km_results = km_fit.km_results

times = np.mean(km_results[:, :, 0], axis=0)
survival_prob = np.mean(km_results[:, :, 1], axis=0)
```

```
km_data = pd.DataFrame({
    'Time': times,
    'Survival Probability': survival_prob,
})
```

```
plt.figure(figsize=(10, 6))

sns.lineplot(x='Time', y='Survival Probability', data=km_data, drawstyle='steps-post', color=

plt.xlabel('Time (Duration)')
plt.ylabel('Survival Probability')
plt.title('Kaplan-Meier Survival Curve with 95% Confidence Interval')
plt.ylim(0, 1)
plt.grid(True)
plt.legend()
plt.show()
```

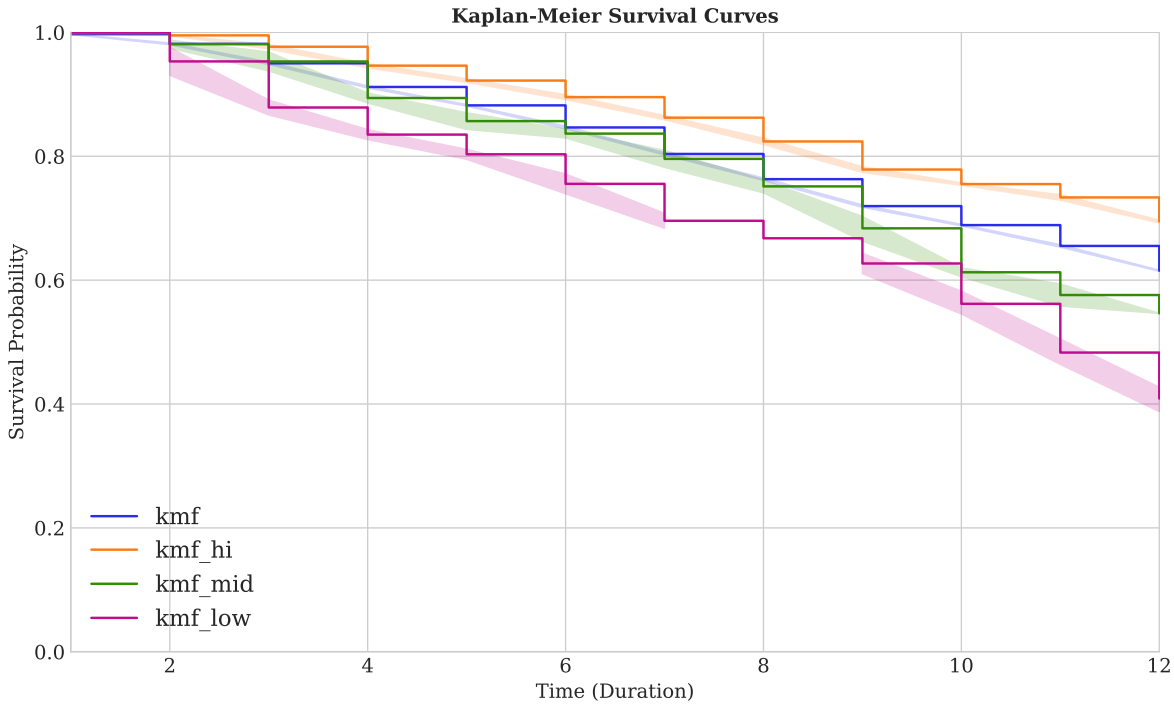
```
KM_model_path = os.path.join(p_dir, 'models', 'kaplan_meier.stan')
KM_model = CmdStanModel(stan_file=KM_model_path)

def fit_model(data, model, random_seed):

    stan_data = {
        'N': len(data["month"]),
        'durations': data["month"],
        'event_observed': data["left"],
    }


    km_fit = model.sample(data=stan_data, seed=random_seed, chains=4, iter_sampling=1, iter_w


    km_results = km_fit.stan_variable('km_results')
    times = np.mean(km_results[:, :, 0], axis=0)
    survival_prob = np.mean(km_results[:, :, 1], axis=0)

    return times, survival_prob

datasets = {
    'kmf': retention_df,
    'kmf_hi': retention_df[retention_df["sentiment"] == 10],
    'kmf_mid': retention_df[retention_df["sentiment"] == 5],
```

```
    'kmf_low': retention_df[retention_df["sentiment"] == 2],
}

plt.figure(figsize=(10, 6))

all_times = []
all_survival_probs = []
labels = []

for label, data in datasets.items():
    times, survival_prob = fit_model(data, KM_model, RANDOM_SEED)
    all_times.append(times)
    all_survival_probs.append(survival_prob)
    labels.append(label)

for i in range(len(all_times)):
    sns.lineplot(x=all_times[i], y=all_survival_probs[i], drawstyle='steps-post', label=label

plt.xlabel('Time (Duration)')
plt.ylabel('Survival Probability')
plt.title('Kaplan-Meier Survival Curves')
plt.ylim(0, 1)
plt.grid(True)
plt.legend()
plt.show()
```

```
16:01:32 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status

chain 3 |          | 00:00 Status


chain 4 |          | 00:00 Status


chain 4 |        | 00:00 Status

chain 3 |        | 00:00 Status
chain 2 |        | 00:01 Statuschain 1 |        | 00:01 Statuschain 1 |     | 00:01 Sampling

chain 2 |     | 00:01 Sampling completedchain 2 |     | 00:01 Sampling completed
```

```
chain 3 |     | 00:01 Sampling completedchain 3 |     | 00:01 Sampling completed



chain 4 |     | 00:01 Sampling completedchain 4 |     | 00:01 Sampling completed
16:01:33 - cmdstanpy - INFO - CmdStan done processing.
16:01:34 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status

chain 3 |          | 00:00 Status


chain 4 |          | 00:00 Statuschain 1 |        | 00:00 Status
chain 2 |        | 00:00 Status


chain 4 |        | 00:01 Status

chain 3 |        | 00:01 Statuschain 1 |     | 00:01 Sampling completedchain 1 |     | 00:01

chain 2 |     | 00:01 Sampling completedchain 2 |     | 00:01 Sampling completed
chain 3 |     | 00:01 Sampling completed



chain 4 |     | 00:01 Sampling completedchain 4 |     | 00:01 Sampling completed
16:01:35 - cmdstanpy - INFO - CmdStan done processing.
16:01:35 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status

chain 3 |          | 00:00 Status


chain 4 |          | 00:00 Status

chain 3 |        | 00:00 Statuschain 1 |        | 00:00 Status


chain 4 |        | 00:01 Status
```

```
chain 2 |        | 00:01 Statuschain 1 |      | 00:01 Sampling completedchain 1 |      | 00:01
chain 2 |     | 00:01 Sampling completed


chain 3 |     | 00:01 Sampling completedchain 3 |     | 00:01 Sampling completed



chain 4 |     | 00:01 Sampling completedchain 4 |     | 00:01 Sampling completed
16:01:37 - cmdstanpy - INFO - CmdStan done processing.
16:01:37 - cmdstanpy - INFO - CmdStan start processing
chain 1 |          | 00:00 Status
chain 2 |          | 00:00 Status

chain 3 |          | 00:00 Status


chain 4 |          | 00:00 Status

chain 3 |        | 00:00 Statuschain 1 |        | 00:00 Status


chain 4 |        | 00:01 Status
chain 2 |        | 00:01 Statuschain 1 |      | 00:01 Sampling completedchain 1 |      | 00:01
chain 2 |     | 00:01 Sampling completed


chain 3 |     | 00:01 Sampling completedchain 3 |     | 00:01 Sampling completed



chain 4 |     | 00:01 Sampling completedchain 4 |     | 00:01 Sampling completed
16:01:38 - cmdstanpy - INFO - CmdStan done processing.
```



3 Data Preperation For Survival Regression

You could write this in the transfromed data section of the stan code but it's so much easier to just transform and pass it.

```
intervals = np.arange(12)
n_employees = retention_df.shape[0]
n_intervals = len(intervals)
last_period = np.floor((retention_df.month - 0.01) / 1).astype(int)
employees = np.arange(n_employees)
quit = np.zeros((n_employees, n_intervals))
quit[employees, last_period] = retention_df["left"]
quit = quit.astype(int)
pd.DataFrame(quit)
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0
2	0	0	0	0	0	0	0	0	1	0	0	0

	0	1	2	3	4	5	6	7	8	9	10	11
3	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...
3765	0	0	1	0	0	0	0	0	0	0	0	0
3766	0	0	0	0	0	0	1	0	0	0	0	0
3767	0	0	0	0	0	0	0	0	0	0	0	0
3768	0	0	0	0	0	0	0	0	0	0	0	0
3769	0	0	0	0	0	0	0	0	0	0	0	0

```
exposure = np.greater_equal.outer(retention_df.month.to_numpy(), intervals) * 1
exposure[employees, last_period] = retention_df.month - intervals[last_period]
pd.DataFrame(exposure)
```

	0	1	2	3	4	5	6	7	8	9	10	11
0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	1	1	0	0
3	1	1	1	1	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1	1	1	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...
3765	1	1	1	1	0	0	0	0	0	0	0	0
3766	1	1	1	1	1	1	1	1	0	0	0	0
3767	1	1	1	1	1	1	1	1	1	1	1	1
3768	1	1	1	1	1	1	1	1	1	1	1	1
3769	1	1	1	1	1	1	1	1	1	1	1	1

4 Fit Basic Cox Model with Fixed Effects

```
preds = [
    "sentiment",
    "Male",
    "Low",
    "Medium",
    "Finance",
    "Health",
    "Law",
```

```
    "Public/Government",
    "Sales/Marketing",
]
preds2 = [
    "sentiment",
    "intention",
    "Male",
    "Low",
    "Medium",
    "Finance",
    "Health",
    "Law",
    "Public/Government",
    "Sales/Marketing",
]


stan_data_2 = {
  'N': len(retention_df),
  'P': len(preds),
  'K': n_intervals,
  'X_data': retention_df[preds].astype(int),
  'quit': quit,
  'exposure': exposure,
}

stan_data_3 = {
  'N': len(retention_df),
  'P': len(preds2),
  'K': n_intervals,

  'X_data': retention_df[preds2].astype(int),
  'quit': quit,
  'exposure': exposure,
}

# Cox PH model
cox_model_path = os.path.join(p_dir, 'models', 'log_poisson.stan')
cox_model_base_samples = os.path.join(p_dir, 'models', 'log_poisson_base.csv')
cox_model_intention_samples = os.path.join(p_dir, 'models', 'log_poisson_intentions.csv')
cox_model = CmdStanModel(stan_file=cox_model_path)
```

```
# Fit model
if os.path.exists(cox_model_base_samples):
    cox_fit_base = from_csv(cox_model_base_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    cox_fit_base = cox_model.sample(data=stan_data_2, seed=RANDOM_SEED, chains=4, parallel_c
    cox_fit_base.save_csvfiles(cox_model_base_samples)
    print("Model run and saved.")

if os.path.exists(cox_model_intention_samples):
    cox_fit_intention = from_csv(cox_model_intention_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    cox_fit_intention = cox_model.sample(data=stan_data_3, seed=RANDOM_SEED, chains=4,paralle
    cox_fit_intention.save_csvfiles(cox_model_intention_samples)
    print("Model run and saved.")
```

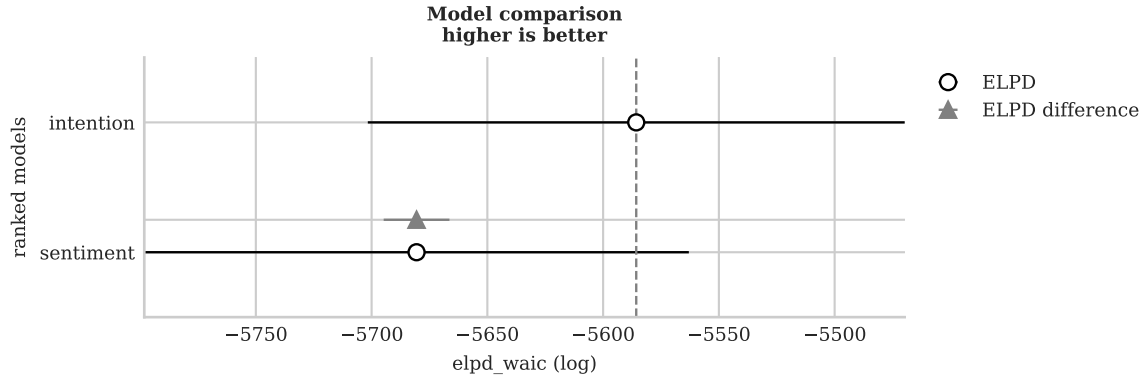
```
Model loaded from existing samples.
Model loaded from existing samples.
```

```
base_idata = az.from_cmdstanpy(
    cox_fit_base,
    log_likelihood="log_lik",
    dims={'beta': ['preds'], 'log_lik': ['individuals', 'intervals'], 'lambda0': ['intervals
    coords={
        'preds': preds,  # Predictor names or indices
        'individuals': list(range(len(retention_df))),  # Coordinates for individuals
        'intervals': list(range(n_intervals))  # Coordinates for intervals
    }
)

base_intention_idata = az.from_cmdstanpy(
    cox_fit_intention,
    log_likelihood="log_lik",
    dims={'beta': ['preds'], 'log_lik': ['individuals', 'intervals'], 'lambda0': ['intervals
    coords={
        'preds': preds2,  # Predictor names or indices
        'individuals': list(range(len(retention_df))),  # Coordinates for individuals
        'intervals': list(range(n_intervals))  # Coordinates for intervals
    }
)
```

```
compare = az.compare({"sentiment": base_idata, "intention": base_intention_idata}, ic="waic")
compare
```

	rank	elpd_waic	p_waic	elpd_diff	weight	se	dse	warning	scale
intention	0	-5585.694107	20.953239	0.00000	0.997046	115.941449	0.000000	False	log
sentiment	1	-5680.578997	20.115744	94.88489	0.002954	117.605617	14.205166	False	log

```
az.plot_compare(compare)
```



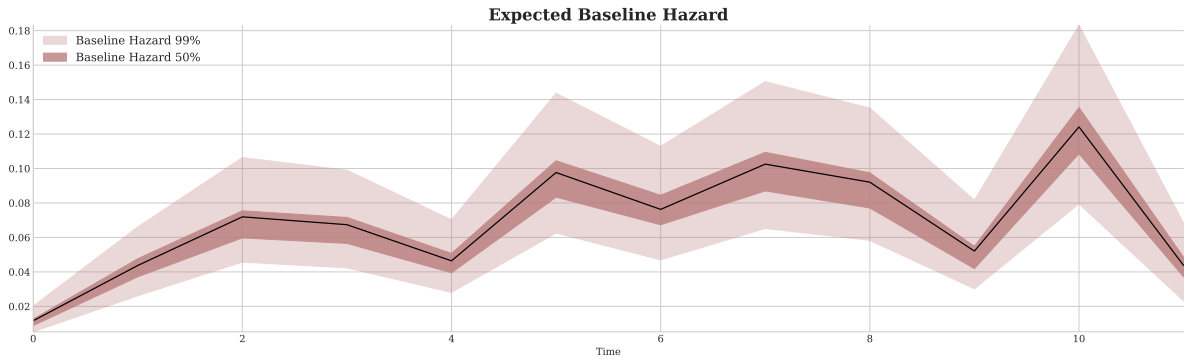
4.1 Interpreting the Model Coefficients

```
m = (
    az.summary(base_idata, var_names=["beta"])
    .reset_index()[["index", "mean"]]
    .rename({"mean": "expected_hr"}, axis=1)
)
m1 = (
    az.summary(base_intention_idata, var_names=["beta"])
    .reset_index()[["index", "mean"]]
    .rename({"mean": "expected_intention_hr"}, axis=1)
)
m = m.merge(m1, left_on="index", right_on="index", how="outer")
m["exp(expected_hr)"] = np.exp(m["expected_hr"])
m["exp(expected_intention_hr)"] = np.exp(m["expected_intention_hr"])
m
```

	index	expected_hr	expected_intention_hr	exp(expected_hr)	exp(expected_int
0	beta[Finance]	0.205	0.232	1.227525	1.261120
1	beta[Health]	0.248	0.235	1.281460	1.264909
2	beta[Law]	0.094	0.071	1.098560	1.073581
3	beta[Low]	0.136	0.154	1.145682	1.166491
4	beta[Male]	-0.039	0.015	0.961751	1.015113
5	beta[Medium]	0.160	0.106	1.173511	1.111822
6	beta[Public/Government]	0.099	0.119	1.104066	1.126370
7	beta[Sales/Marketing]	0.074	0.098	1.076807	1.102963
8	beta[intention]	NaN	0.189	NaN	1.208041
9	beta[sentiment]	-0.109	-0.029	0.896730	0.971416

```
fig, ax = plt.subplots(figsize=(20, 6))
ax.plot(base_idata["posterior"]["lambda0"].mean(("draw", "chain")), color="black")
az.plot_hdi(
    range(12),
    base_idata["posterior"]["lambda0"],
    color= c_mid,
    ax=ax,
    hdi_prob=0.99,
    fill_kwargs={"label": "Baseline Hazard 99%", "alpha": 0.3},
    smooth=False,
)
az.plot_hdi(
    range(12),
    base_idata["posterior"]["lambda0"],
    color=c_mid,
    ax=ax,
    hdi_prob=0.50,
    fill_kwargs={"label": "Baseline Hazard 50%", "alpha": 0.8},
    smooth=False,
)
ax.legend()
ax.set_xlabel("Time")
ax.set_title("Expected Baseline Hazard", fontsize=20)
```

```
Text(0.5, 1.0, 'Expected Baseline Hazard')
```



4.2 Predicting Marginal Effects of CoxPH regression

```
def cum_hazard(hazard):
    """Takes arviz.InferenceData object applies
    cumulative sum along baseline hazard"""
    return hazard.cumsum(dim="intervals")


def survival(hazard):
    """Takes arviz.InferenceData object transforms
    cumulative hazard into survival function"""
    return np.exp(-cum_hazard(hazard))


def get_mean(trace):
    """Takes arviz.InferenceData object marginalises
    over the chain and draw"""
    return trace.mean(("draw", "chain"))
```

```
def extract_individual_hazard(idata, i, retention_df, intention=False):
    beta = idata.posterior["beta"]
    if intention:
        intention_posterior = beta.sel(preds="intention")
    else:
        intention_posterior = 0
    hazard_base_m1 = idata["posterior"]["lambda0"]

    full_hazard_idata = hazard_base_m1 * np.exp(
        beta.sel(preds="sentiment") * retention_df.iloc[i]["sentiment"]
```

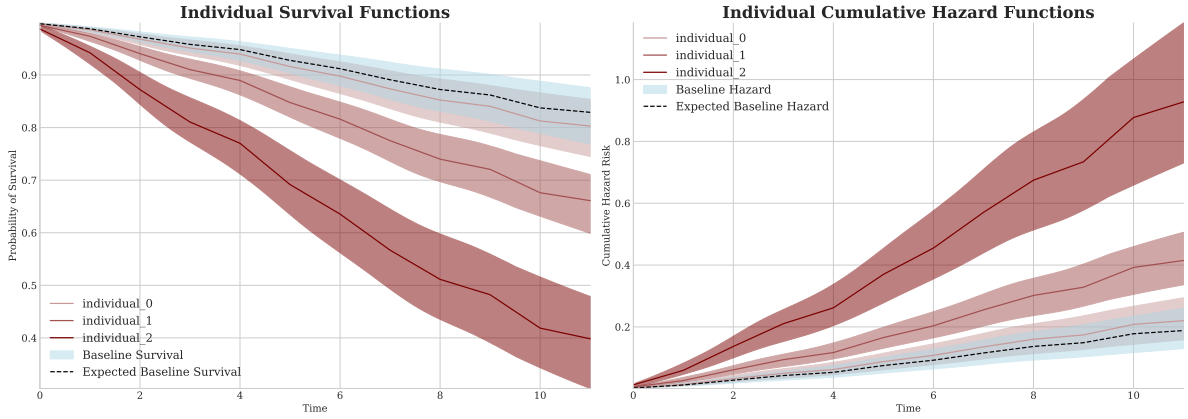
```
        + np.where(intention, intention_posterior * retention_df.iloc[i]["intention"], 0)
        + beta.sel(preds="Male") * retention_df.iloc[i]["Male"]
        + beta.sel(preds="Low") * retention_df.iloc[i]["Low"]
        + beta.sel(preds="Medium") * retention_df.iloc[i]["Medium"]
        + beta.sel(preds="Finance") * retention_df.iloc[i]["Finance"]
        + beta.sel(preds="Health") * retention_df.iloc[i]["Health"]
        + beta.sel(preds="Law") * retention_df.iloc[i]["Law"]
        + beta.sel(preds="Public/Government") * retention_df.iloc[i]["Public/Government"]
        + beta.sel(preds="Sales/Marketing") * retention_df.iloc[i]["Sales/Marketing"]
    )

    cum_haz_idata = cum_hazard(full_hazard_idata)
    survival_idata = survival(full_hazard_idata)
    return full_hazard_idata, cum_haz_idata, survival_idata, hazard_base_m1


def plot_individuals(retention_df, idata, individuals=[1, 300, 700], intention=False):
    fig, axs = plt.subplots(1, 2, figsize=(20, 7))
    axs = axs.flatten()
    colors = [c_light_highlight, c_mid_highlight, c_dark_highlight]
    for i, c in zip(individuals, colors):
        haz_idata, cum_haz_idata, survival_idata, base_hazard = extract_individual_hazard(
            idata, i, retention_df, intention
        )
        axs[0].plot(get_mean(survival_idata), label=f"individual_{i}", color=c)
        az.plot_hdi(range(12), survival_idata, ax=axs[0], fill_kwargs={"color": c})
        axs[1].plot(get_mean(cum_haz_idata), label=f"individual_{i}", color=c)
        az.plot_hdi(range(12), cum_haz_idata, ax=axs[1], fill_kwargs={"color": c})
        axs[0].set_title("Individual Survival Functions", fontsize=20)
        axs[1].set_title("Individual Cumulative Hazard Functions", fontsize=20)
    az.plot_hdi(
        range(12),
        survival(base_hazard),
        color= 'lightblue',
        ax=axs[0],
        fill_kwargs={"label": "Baseline Survival"},
    )
    axs[0].plot(
        get_mean(survival(base_hazard)),
        color='black',
        linestyle="--",
        label="Expected Baseline Survival",
```

```
    )
    az.plot_hdi(
        range(12),
        cum_hazard(base_hazard),
        color='lightblue',
        ax=axs[1],
        fill_kwargs={"label": "Baseline Hazard"},
    )
    axs[1].plot(
        get_mean(cum_hazard(base_hazard)),
        color="black",
        linestyle="--",
        label="Expected Baseline Hazard",
    )
    axs[0].legend()
    axs[0].set_ylabel("Probability of Survival")
    axs[1].set_ylabel("Cumulative Hazard Risk")
    axs[0].set_xlabel("Time")
    axs[1].set_xlabel("Time")
    axs[1].legend()


#### Next set up test-data input to explore the relationship between levels of the variables
test_df = pd.DataFrame(np.zeros((3, 15)), columns=retention_df.columns)
test_df["sentiment"] = [1, 5, 10]
test_df["intention"] = [1, 5, 10]
test_df["Medium"] = [0, 0, 0]
test_df["Finance"] = [0, 0, 0]
test_df["M"] = [1, 1, 1]
test_df
```

	gender	field	level	sentiment	intention	left	month	Male	Low	Medium	Finance	Health	La
0	0.0	0.0	0.0	1	1	0.0	0.0	0.0	0.0	0	0	0.0	0.
1	0.0	0.0	0.0	5	5	0.0	0.0	0.0	0.0	0	0	0.0	0.
2	0.0	0.0	0.0	10	10	0.0	0.0	0.0	0.0	0	0	0.0	0.

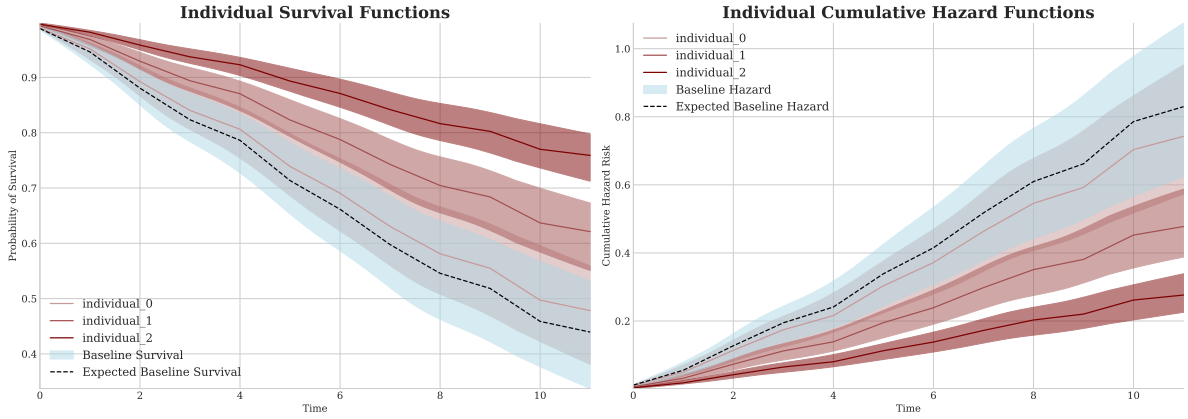
4.3 The Intention Model

```
plot_individuals(test_df, base_intention_idata, [0, 1, 2], intention=True)
```



4.4 The Sentiment Model

```
plot_individuals(test_df, base_idata, [0, 1, 2], intention=False)
```



4.5 Make Predictions for Individual Characteristics

```
def create_predictions(retention_df, idata, intention=False):
    cum_haz = {}
    surv = {}
    for i in range(len(retention_df)):
```

```
        haz_idata, cum_haz_idata, survival_idata, base_hazard = extract_individual_hazard(
            idata, i, retention_df, intention=intention
        )
        cum_haz[i] = get_mean(cum_haz_idata)
        surv[i] = get_mean(survival_idata)
    cum_haz = pd.DataFrame(cum_haz)
    surv = pd.DataFrame(surv)
    return cum_haz, surv


cum_haz_df, surv_df = create_predictions(retention_df, base_idata, intention=False)
surv_df
```

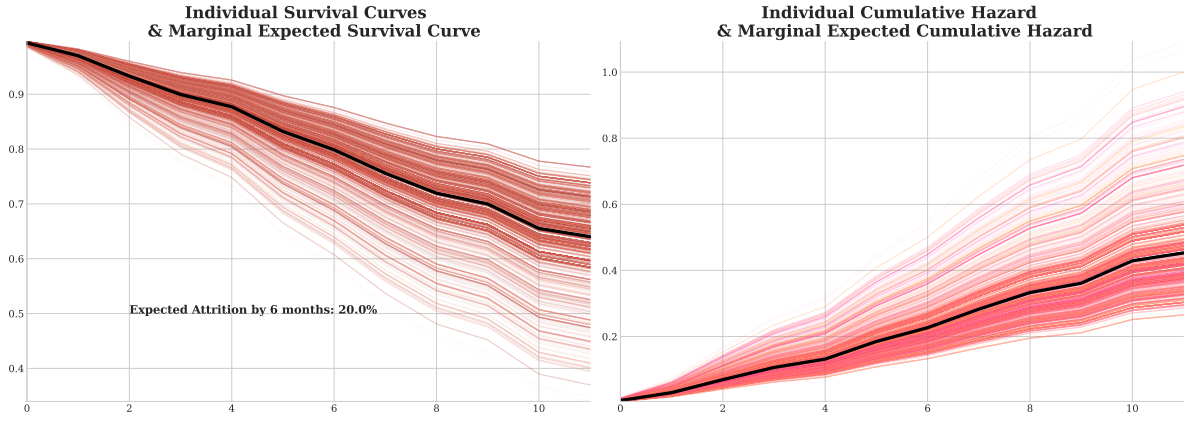
	0	1	2	3	4	5	6	7	8	9
0	0.994495	0.994356	0.995572	0.995178	0.994495	0.994566	0.994495	0.990584	0.992469	0.992
1	0.974127	0.973489	0.979148	0.977311	0.974127	0.974468	0.974127	0.956045	0.964749	0.966
2	0.941523	0.940106	0.952715	0.948615	0.941523	0.942283	0.941523	0.901852	0.920823	0.925
3	0.911988	0.909895	0.928617	0.922517	0.911988	0.913119	0.911988	0.853926	0.881514	0.888
4	0.892185	0.889654	0.912374	0.904963	0.892185	0.893555	0.892185	0.822409	0.855415	0.863
5	0.851931	0.848544	0.879136	0.869128	0.851931	0.853774	0.851931	0.759883	0.803027	0.813
6	0.821758	0.817762	0.854019	0.842139	0.821758	0.823940	0.821758	0.714393	0.764354	0.776
7	0.782887	0.778156	0.821395	0.807191	0.782887	0.785483	0.782887	0.657538	0.715296	0.729
8	0.749550	0.744230	0.793161	0.777055	0.749550	0.752480	0.749550	0.610374	0.673932	0.690
9	0.731341	0.725713	0.777637	0.760525	0.731341	0.734447	0.731341	0.585232	0.651620	0.668
10	0.689687	0.683408	0.741833	0.722522	0.689687	0.693172	0.689687	0.529426	0.601360	0.620
11	0.675569	0.669090	0.729602	0.709580	0.675569	0.679177	0.675569	0.511048	0.584577	0.604

4.6 Sample Survival Curves and their Marginal Expected Survival Trajectory

```
from matplotlib import cm

cm_subsection = np.linspace(0, 1, 120)
colors_m = [cm.Reds(x) for x in cm_subsection]
colors = [cm.spring(x) for x in cm_subsection]


fig, axs = plt.subplots(1, 2, figsize=(20, 7))
axs = axs.flatten()
cum_haz_df.plot(legend=False, color=colors, alpha=0.05, ax=axs[1])
```

```
axs[1].plot(cum_haz_df.mean(axis=1), color="black", linewidth=4)
axs[1].set_title(
    "Individual Cumulative Hazard \n & Marginal Expected Cumulative Hazard", fontsize=20
)

surv_df.plot(legend=False, color=colors_m, alpha=0.05, ax=axs[0])
axs[0].plot(surv_df.mean(axis=1), color="black", linewidth=4)
axs[0].set_title("Individual Survival Curves \n  & Marginal Expected Survival Curve", fontsi
axs[0].annotate(
    f"Expected Attrition by 6 months: {100*np.round(1-surv_df.mean(axis=1).iloc[6], 2)}%",
    (2, 0.5),
    fontsize=14,
    fontweight="bold",
)
```

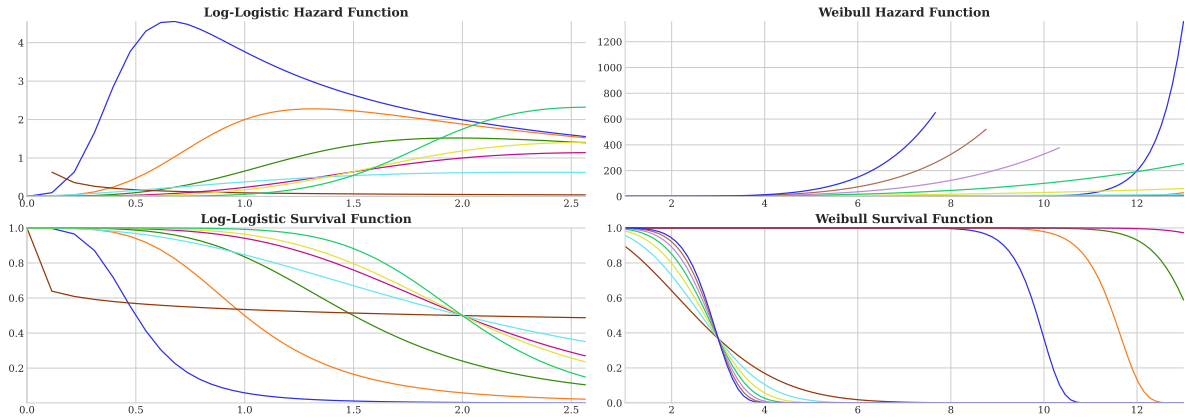
```
Text(2, 0.5, 'Expected Attrition by 6 months: 20.0%')
```



5 Accelerated Failure Time Models

```
from scipy.stats import fisk, weibull_min
fig, axs = plt.subplots(2, 2, figsize=(20, 7))
axs = axs.flatten()


def make_loglog_haz(alpha, beta):
    ## This is the Log Logistic distribution
```

```
    dist = fisk(c=alpha, scale=beta)
    t = np.log(np.linspace(1, 13, 100))  # Time values
    pdf_values = dist.pdf(t)
    sf_values = dist.sf(t)
    haz_values = pdf_values / sf_values
    axs[0].plot(t, haz_values)
    axs[2].plot(t, sf_values)


def make_weibull_haz(alpha, beta):
    dist = weibull_min(c=alpha, scale=beta)
    t = np.linspace(1, 13, 100)  # Time values
    pdf_values = dist.pdf(t)
    sf_values = dist.sf(t)
    haz_values = pdf_values / sf_values
    axs[1].plot(t, haz_values)
    axs[3].plot(t, sf_values)


[make_loglog_haz(4, b) for b in np.linspace(0.5, 2, 4)]
[make_loglog_haz(a, 2) for a in np.linspace(0.2, 7, 4)]
[make_weibull_haz(25, b) for b in np.linspace(10, 15, 4)]
[make_weibull_haz(a, 3) for a in np.linspace(2, 7, 7)]
axs[0].set_title("Log-Logistic Hazard Function", fontsize=15)
axs[2].set_title("Log-Logistic Survival Function", fontsize=15)
axs[1].set_title("Weibull Hazard Function", fontsize=15)
axs[3].set_title("Weibull Survival Function", fontsize=15);
```



```
coords = {
    "intervals": intervals,
    "preds": [
        "sentiment",
        "intention",
        "Male",
        "Low",
        "Medium",
        "Finance",
        "Health",
        "Law",
        "Public/Government",
        "Sales/Marketing",
    ],
}

X = retention_df[
    [
        "sentiment",
        "intention",
        "Male",
        "Low",
        "Medium",
        "Finance",
        "Health",
        "Law",
        "Public/Government",
        "Sales/Marketing",
    ]
].copy()
y = retention_df["month"].values
cens = retention_df.left.values == 0

stan_data_4 = {
  'N': len(X),
  'P': X.shape[1],
  'X_data': X,
  'y': y,
  'cens': cens,
  'is_weibull': 1,
}
```

```
stan_data_5 = {
  'N': len(X),
  'P': X.shape[1],
  'X_data': X,
  'y': np.log(y),
  'cens': cens,
  'is_weibull': 0,
}

aft_path = os.path.join(p_dir, 'models', 'weibull_log_log.stan')
weibull_model_samples = os.path.join(p_dir, 'models', 'weibull_sample.csv')
loglog_model_samples = os.path.join(p_dir, 'models', 'loglog_samples.csv')
aft_model = CmdStanModel(stan_file=aft_path)
```

```
# Fit model
if os.path.exists(weibull_model_samples):
    weibull_fit = from_csv(weibull_model_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    weibull_fit = aft_model.sample(data=stan_data_4, seed=RANDOM_SEED, chains=4, parallel_cha
    weibull_fit.save_csvfiles(weibull_model_samples)
    print("Model run and saved.")

if os.path.exists(loglog_model_samples):
    loglog_fit = from_csv(loglog_model_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    loglog_fit = aft_model.sample(data=stan_data_5, seed=RANDOM_SEED, chains=4,parallel_chai
    loglog_fit.save_csvfiles(loglog_model_samples)
    print("Model run and saved.")
```

```
Model loaded from existing samples.
Model loaded from existing samples.
```

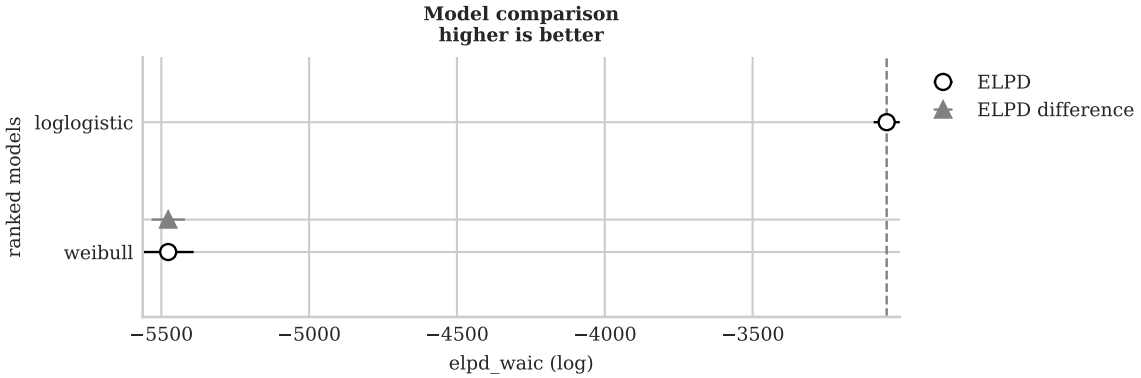
```
weibull_idata = az.from_cmdstanpy(
    weibull_fit,
    log_likelihood="log_lik",
    dims={'beta': ['preds'], },
    coords={
        'preds': preds2,  # Predictor names or indices
    }
```

```
)

loglogistic_idata = az.from_cmdstanpy(
    loglog_fit,
    log_likelihood="log_lik",
    dims={'beta': ['preds'], },
    coords={
        'preds': preds2,  # Predictor names or indices
    }
)
```

```
compare = az.compare({"weibull": weibull_idata, "loglogistic": loglogistic_idata}, ic="waic")
compare
```

	rank	elpd_waic	p_waic	elpd_diff	weight	se	dse	warning
loglogistic	0	-3046.479210	12.375341	0.000000	1.000000e+00	44.023723	0.000000	False
weibull	1	-5476.740427	12.082975	2430.261217	1.837419e-13	86.326833	56.336104	False

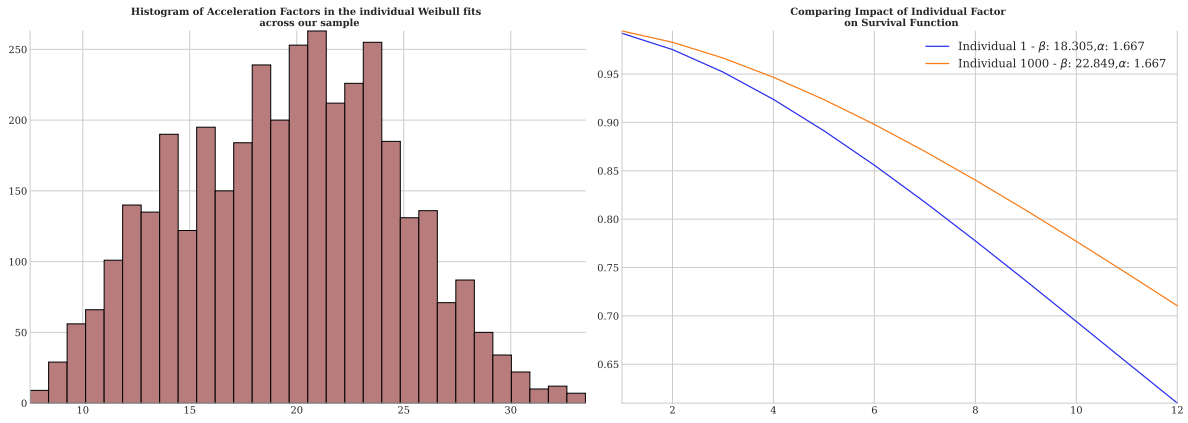
```
az.plot_compare(compare)
```



5.1 Deriving Individual Survival Predictions from AFT models

5.2 Weibull

```
fig, axs = plt.subplots(1, 2, figsize=(20, 7))
axs = axs.flatten()
#### Using the fact that we've already stored expected value for the regression equation
reg = az.summary(weibull_idata, var_names=["reg"])["mean"]
t = np.arange(1, 13, 1)
s = az.summary(weibull_idata, var_names=["s"])["mean"][0]
axs[0].hist(reg, bins=30, ec="black", color=c_mid)
axs[0].set_title(
    "Histogram of Acceleration Factors in the individual Weibull fits \n across our sample"
)
axs[1].plot(
    t,
    weibull_min.sf(t, s, scale=reg.iloc[0]),
    label=r"Individual 1 - $\beta$: " + f"{reg.iloc[0]}," + r"$\alpha$: " + f"{s}",
)
axs[1].plot(
    t,
    weibull_min.sf(t, s, scale=reg.iloc[1000]),
    label=r"Individual 1000 - $\beta$: " + f"{reg.iloc[1000]}," + r"$\alpha$: " + f"{s}",
)
axs[1].set_title("Comparing Impact of Individual Factor \n on Survival Function")
axs[1].legend();
```




```
diff = reg.iloc[1000] - reg.iloc[0]
pchange = np.round(100 * (diff / reg.iloc[1000]), 2)
print(
    f"In this case we could think of the relative change in acceleration \n factor between t
)
```

In this case we could think of the relative change in acceleration
 factor between the individuals as representing a 19.89% increase

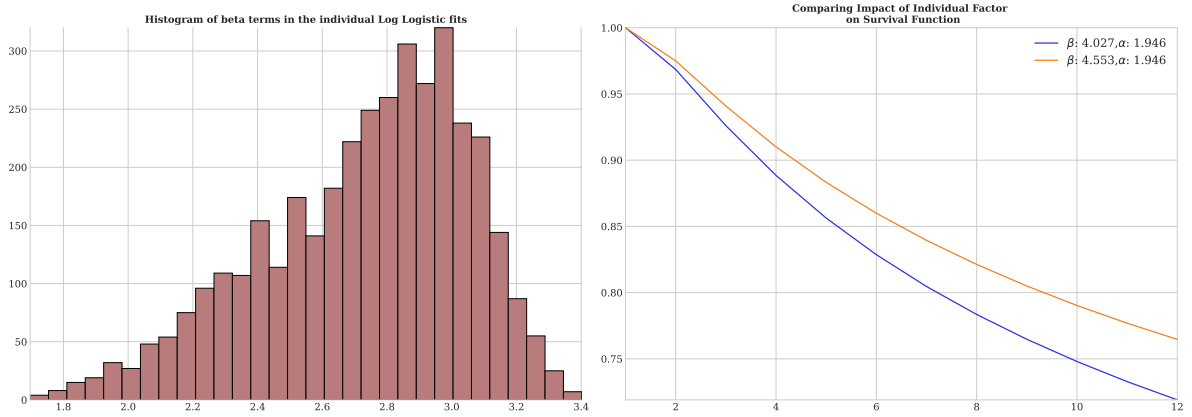
```
reg = az.summary(weibull_idata, var_names=["reg"])["mean"]
s = az.summary(weibull_idata, var_names=["s"])["mean"][0]
t = np.arange(1, 13, 1)
weibull_predicted_surv = pd.DataFrame(
    [weibull_min.sf(t, s, scale=reg.iloc[i]) for i in range(len(reg))]
).T

weibull_predicted_surv
```

	0	1	2	3	4	5	6	7	8	9
0	0.992173	0.994964	0.989363	0.996188	0.985887	0.986597	0.993573	0.991462	0.984744	0.988
1	0.975356	0.984094	0.966610	0.987945	0.955868	0.958054	0.979734	0.973139	0.952352	0.962
2	0.952132	0.968973	0.935420	0.976440	0.915095	0.919214	0.960550	0.947880	0.908489	0.927
3	0.923821	0.950360	0.897770	0.962219	0.866469	0.872778	0.937051	0.917167	0.856388	0.885
4	0.891418	0.928804	0.855187	0.945665	0.812278	0.820871	0.909996	0.882119	0.798604	0.838
5	0.855759	0.904756	0.808967	0.927085	0.754456	0.765293	0.880018	0.843684	0.737297	0.787
6	0.817580	0.878608	0.760245	0.906746	0.694672	0.707601	0.847671	0.802694	0.674312	0.734
7	0.777539	0.850712	0.710025	0.884884	0.634356	0.649140	0.813454	0.759894	0.611214	0.680
8	0.736232	0.821389	0.659186	0.861714	0.574711	0.591052	0.777822	0.715950	0.549298	0.625
9	0.694195	0.790937	0.608495	0.837438	0.516730	0.534290	0.741189	0.671460	0.489610	0.571
10	0.651905	0.759630	0.558606	0.812242	0.461198	0.479623	0.703932	0.626950	0.432955	0.519
11	0.609789	0.727720	0.510071	0.786299	0.408716	0.427652	0.666394	0.582883	0.379921	0.468

5.3 Log Logistic

```
reg = az.summary(loglogistic_idata, var_names=["reg"])["mean"]
s = az.summary(loglogistic_idata, var_names=["s"])["mean"][0]
temp = retention_df
t = np.log(np.arange(1, 13, 1))
## Transforming to the Log-Logistic scale
alpha = np.round((1 / s), 3)
beta = np.round(np.exp(reg) ** s, 3)

fig, axs = plt.subplots(1, 2, figsize=(20, 7))
axs = axs.flatten()
axs[0].hist(reg, bins=30, ec="black", color=c_mid)
```

```
axs[0].set_title("Histogram of beta terms in the individual Log Logistic fits")
axs[1].plot(
    np.exp(t),
    fisk.sf(t, c=alpha, scale=beta.iloc[0]),
    label=r"$\beta$: " + f"{beta.iloc[0]}," + r"$\alpha$: " + f"{alpha}",
)
axs[1].plot(
    np.exp(t),
    fisk.sf(t, c=alpha, scale=beta.iloc[1000]),
    label=r"$\beta$: " + f"{beta.iloc[1000]}," + r"$\alpha$: " + f"{alpha}",
)
axs[1].set_title("Comparing Impact of Individual Factor \n on Survival Function")
axs[1].legend();
```

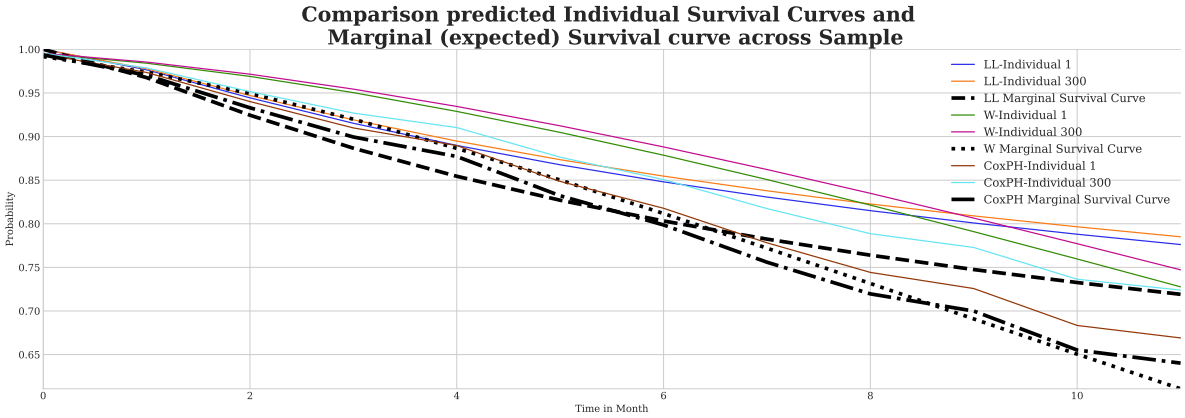



```
loglogistic_predicted_surv = pd.DataFrame(
    [fisk.sf(t, c=alpha, scale=beta.iloc[i]) for i in range(len(reg))]
).T
loglogistic_predicted_surv
```

	0	1	2	3	4	5	6	7	8	9
0	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000
1	0.968448	0.976526	0.960711	0.980638	0.952831	0.954046	0.972428	0.966775	0.950543	0.9561
2	0.926068	0.944373	0.908916	0.953851	0.891818	0.894430	0.935035	0.922328	0.886923	0.8989
3	0.888468	0.915230	0.863875	0.929303	0.839813	0.843460	0.901510	0.883066	0.833005	0.8498
4	0.856282	0.889809	0.825982	0.907676	0.796797	0.801192	0.872548	0.849585	0.788621	0.8088
5	0.828625	0.867607	0.793898	0.888623	0.760889	0.765833	0.847463	0.820909	0.751719	0.7745
6	0.804602	0.848046	0.766379	0.871708	0.730459	0.735814	0.825525	0.796070	0.720551	0.7452
7	0.783496	0.830644	0.742465	0.856558	0.704288	0.709955	0.806134	0.774301	0.693820	0.7199

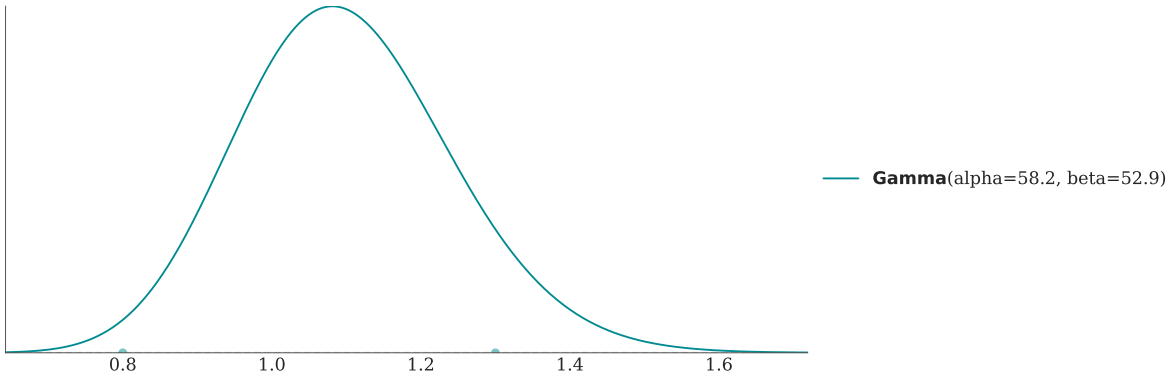
	0	1	2	3	4	5	6	7	8	9
8	0.764757	0.815023	0.721438	0.842875	0.681480	0.687391	0.788826	0.755015	0.670583	0.6978
9	0.747966	0.800886	0.702757	0.830423	0.661377	0.667479	0.773244	0.737767	0.650145	0.6782
10	0.732798	0.788000	0.686010	0.819017	0.643482	0.649736	0.759107	0.722213	0.631987	0.6608
11	0.718999	0.776181	0.670880	0.808507	0.627416	0.633791	0.746196	0.708085	0.615712	0.6450

```
fig, ax = plt.subplots(figsize=(20, 7))
ax.plot(
    loglogistic_predicted_surv.iloc[:, [1, 300]], label=["LL-Individual 1", "LL-Individual 30
)
ax.plot(
    loglogistic_predicted_surv.mean(axis=1),
    label="LL Marginal Survival Curve",
    linestyle="--",
    color="black",
    linewidth=4.5,
)
ax.plot(weibull_predicted_surv.iloc[:, [1, 300]], label=["W-Individual 1", "W-Individual 300"
ax.plot(
    weibull_predicted_surv.mean(axis=1),
    label="W Marginal Survival Curve",
    linestyle="dotted",
    color="black",
    linewidth=4.5,
)
ax.plot(surv_df.iloc[:, [1, 300]], label=["CoxPH-Individual 1", "CoxPH-Individual 300"])
ax.plot(
    surv_df.mean(axis=1),
    label="CoxPH Marginal Survival Curve",
    linestyle="-.",
    color="black",
    linewidth=4.5,
)
ax.set_title(
    "Comparison predicted Individual Survival Curves and \n Marginal (expected) Survival curv
    fontsize=25,
)
ax.set_xlabel("Time in Month")
ax.set_ylabel("Probability")
ax.legend();
```



6 Fit Model with Shared Frailty terms by Individual

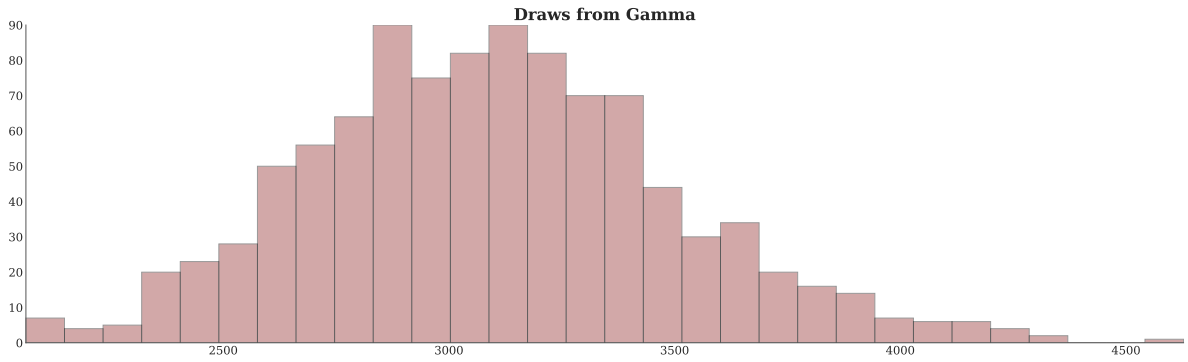
```
pz.style.use('preliz-doc')
params = pz.maxent(pz.Gamma(), 0.80, 1.30, 0.90)
```




```
alpha = params[0].alpha
beta = params[0].beta
params = np.array([alpha, beta])
```

```
fig, ax = plt.subplots(figsize=(20, 6))
ax.hist(
    np.random.gamma(alpha, beta, size=1000),
    ec="black",
    color=c_dark,
    bins=30,
```

```
    alpha=0.4,
)

ax.set_title("Draws from Gamma", fontsize=20);
```




```
female_df = retention_df[retention_df['Male'] == 0]
male_df = retention_df[retention_df['Male'] == 1]

ordered_df = pd.concat([female_df, male_df], axis=0)
ordered_df = ordered_df.reset_index(drop=True)

intervals = np.arange(12)
n_employees = ordered_df.shape[0]
n_intervals = len(intervals)
last_period = np.floor((ordered_df.month - 0.01) / 1).astype(int)
employees = np.arange(n_employees)
quit = np.zeros((n_employees, n_intervals))
quit[employees, last_period] = ordered_df["left"]
quit = quit.astype(int)
quit_df = pd.DataFrame(quit)

exposure = np.greater_equal.outer(ordered_df.month.to_numpy(), intervals) * 1
exposure[employees, last_period] = ordered_df.month - intervals[last_period]
exposure_df = pd.DataFrame(exposure)


preds = [
    "sentiment",
    "intention",
    "Low",
    "Medium",
```

```
    "Finance",
    "Health",
    "Law",
    "Public/Government",
    "Sales/Marketing",
]
preds3 = ["sentiment", "Low", "Medium"]

stan_data_6 = {
    'N': len(ordered_df),
    'N_f': len(female_df),
    'N_m': len(male_df),
    'P': ordered_df[preds].shape[1],
    'K': len(intervals),
    'G': 2,
    'F': len(ordered_df),
    'optm_params': params,
    'X_data_f': female_df[preds],
    'X_data_m': male_df[preds],
    'quit': quit,
    'exposure': exposure,
    'frailty_idx': list(range(1, len(ordered_df) + 1))
}

ordered_df['field_idx'], field_labels = pd.factorize(ordered_df['field'])

stan_data_7 = {
    'N': len(ordered_df),
    'N_f': len(female_df),
    'N_m': len(male_df),
    'P': ordered_df[preds3].shape[1],
    'K': len(intervals),
    'G': 2,
    'F': len(field_labels),
    'optm_params': params,
    'X_data_f': female_df[preds3],
    'X_data_m': male_df[preds3],
    'quit': quit,
    'exposure': exposure,
    'frailty_idx': list(ordered_df['field_idx'] + 1),
}
```

```
frailty_path = os.path.join(p_dir, 'models', 'frailty_log_poisson.stan')
frailty_model_samples = os.path.join(p_dir, 'models', 'frailty_samples.csv')
shared_frailty_model_samples = os.path.join(p_dir, 'models', 'shared_frailty_samples.csv')
frailty_model = CmdStanModel(stan_file=frailty_path)
```

```
# Fit model
if os.path.exists(frailty_model_samples):
    frailty = from_csv(frailty_model_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    frailty = frailty_model.sample(data=stan_data_6, seed=RANDOM_SEED, chains=4, parallel_cha
    frailty.save_csvfiles(frailty_model_samples)
    print("Model run and saved.")

if os.path.exists(shared_frailty_model_samples):
    shared_frailty = from_csv(shared_frailty_model_samples, method='sample')
    print("Model loaded from existing samples.")
else:
    shared_frailty = frailty_model.sample(data=stan_data_7, seed=RANDOM_SEED, chains=4, para
    shared_frailty.save_csvfiles(shared_frailty_model_samples)
    print("Model run and saved.")
```

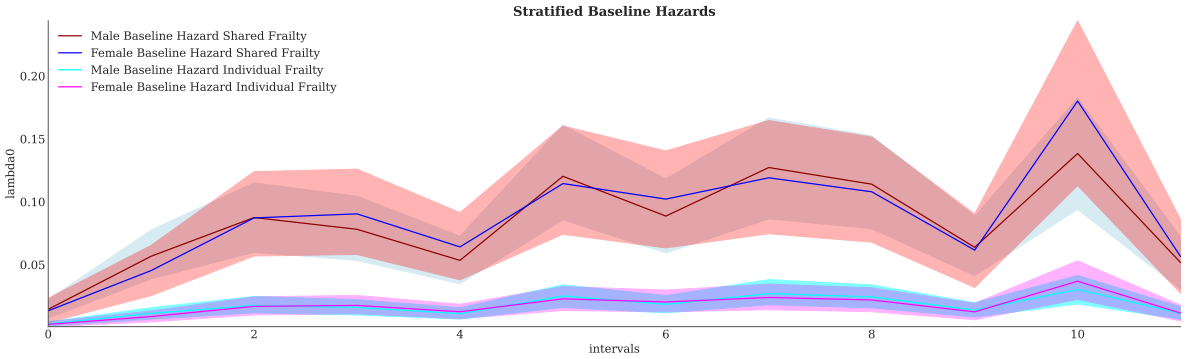
```
Model loaded from existing samples.
Model loaded from existing samples.
```

```
frailty_idata = az.from_cmdstanpy(
    frailty,
    log_likelihood="log_lik",
    dims={
        'beta': ['preds'],
        'log_lik': ['individuals', 'intervals'],
        'lambda0': ['intervals', 'genders'],  # Consistent naming as 'genders'
        'frailty': ['F']
    },
    coords={
        'preds': preds,
        'individuals': list(range(len(retention_df))),
        'intervals': list(range(n_intervals)),
        'genders': ["Male", "Female"],  # Rename 'G' to 'genders'
        'F': list(range(len(retention_df))),
    },
)
```

```
shared_frailty_idata = az.from_cmdstanpy(
    shared_frailty,
    log_likelihood='log_lik',
    dims={
        'beta': ['preds'],
        'log_lik': ['individuals', 'intervals'],
        'lambda0': ['intervals', 'genders'],  # Consistent naming as 'genders'
        'frailty': ['F']
    },
    coords={
        'preds': preds3,
        'individuals': list(range(len(retention_df))),
        'intervals': list(range(n_intervals)),
        'genders': ["Male", "Female"],  # Rename 'G' to 'genders'
        'F': list(range(len(field_labels))),  # Matches field label length
    },
)
```

```
fig, ax = plt.subplots(figsize=(20, 6))
base_m = shared_frailty_idata["posterior"]["lambda0"].sel(genders="Male")
base_f = shared_frailty_idata["posterior"]["lambda0"].sel(genders="Female")
az.plot_hdi(range(12), base_m, ax=ax, color="lightblue", fill_kwargs={"alpha": 0.5}, smooth=
az.plot_hdi(range(12), base_f, ax=ax, color="red", fill_kwargs={"alpha": 0.3}, smooth=False)
get_mean(base_m).plot(ax=ax, color="darkred", label="Male Baseline Hazard Shared Frailty")
get_mean(base_f).plot(ax=ax, color="blue", label="Female Baseline Hazard Shared Frailty")

base_m_i = frailty_idata["posterior"]["lambda0"].sel(genders="Male")
base_f_i = frailty_idata["posterior"]["lambda0"].sel(genders="Female")
az.plot_hdi(range(12), base_m_i, ax=ax, color="cyan", fill_kwargs={"alpha": 0.5}, smooth=Fals
az.plot_hdi(range(12), base_f_i, ax=ax, color="magenta", fill_kwargs={"alpha": 0.3}, smooth=
get_mean(base_m_i).plot(ax=ax, color="cyan", label="Male Baseline Hazard Individual Frailty"
get_mean(base_f_i).plot(ax=ax, color="magenta", label="Female Baseline Hazard Individual Fra


ax.legend()
ax.set_title("Stratified Baseline Hazards")
```

```
Text(0.5, 1.0, 'Stratified Baseline Hazards')
```

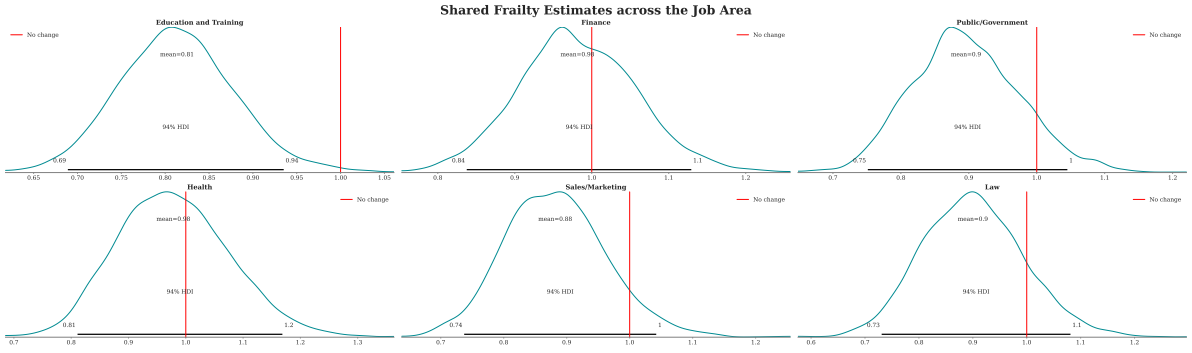



```
frailty_terms = az.summary(frailty_idata, var_names=["frailty"])
frailty_terms.head()
```

	mean	sd	hdi_3%	hdi_97%	mcse_mean	mcse_sd	ess_bulk	ess_tail	r_hat
frailty[0]	0.903	0.125	0.681	1.150	0.002	0.002	3474.0	2917.0	1.0
frailty[1]	0.924	0.134	0.675	1.171	0.003	0.002	2081.0	2209.0	1.0
frailty[2]	0.919	0.128	0.671	1.155	0.002	0.001	3880.0	2664.0	1.0
frailty[3]	0.907	0.128	0.675	1.144	0.002	0.001	3930.0	2546.0	1.0
frailty[4]	0.896	0.127	0.673	1.143	0.002	0.002	3246.0	1913.0	1.0

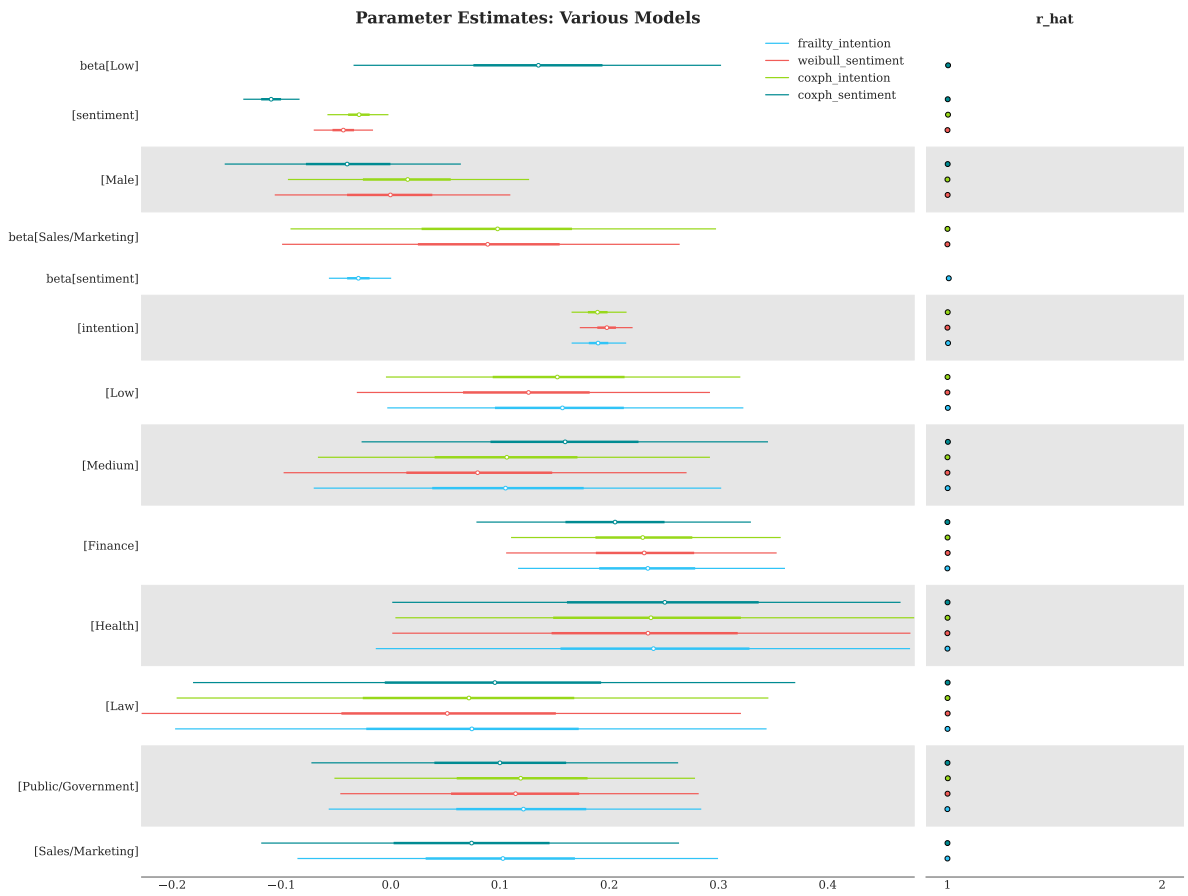
```
axs = az.plot_posterior(shared_frailty_idata, var_names=["frailty"])
axs = axs.flatten()
for ax, label in zip(axs, field_labels):
    ax.set_title(label)
    ax.axvline(1, color="red", label="No change")
    ax.legend()

plt.suptitle("Shared Frailty Estimates across the Job Area", fontsize=30)
```

```
Text(0.5, 0.98, 'Shared Frailty Estimates across the Job Area')
```




```
ax = az.plot_forest(
    [base_idata, base_intention_idata, weibull_idata, frailty_idata],
    model_names=["coxph_sentiment", "coxph_intention", "weibull_sentiment", "frailty_intentio
    var_names=["beta"],
    combined=True,
    figsize=(20, 15),
    r_hat=True,
)

ax[0].set_title("Parameter Estimates: Various Models", fontsize=20)
```

```
Text(0.5, 1.0, 'Parameter Estimates: Various Models')
```




```
temp = retention_df.copy()
temp["frailty"] = frailty_terms.reset_index()["mean"]
(
    temp.groupby(["Male", "sentiment", "intention"])[["frailty"]]
    .mean()
    .reset_index()
    .pivot(index=["Male", "sentiment"], columns="intention", values="frailty")
    .style.background_gradient(cmap="OrRd", axis=None)
    .format(precision=3)
)
```

Table 12

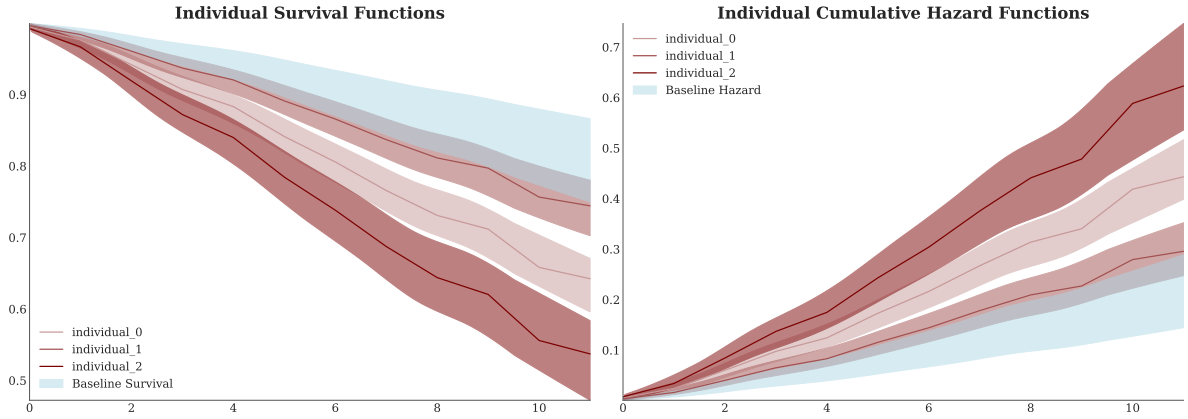
Male	intention sentiment	1
False	1	nan
	2	0.9
	3	0.9
	4	0.9
	5	nan
	6	0.9
	7	0.9
	8	0.9
	9	0.9
	10	0.9
True	1	nan
	2	nan
	3	0.9
	4	nan
	5	0.9
	6	0.9
	7	0.9
	8	0.9
	9	0.9
	10	0.9

7 Interrogating the Cox Frailty Model

```
def extract_individual_frailty(i, retention_df, intention=False):
    beta = frailty_idata.posterior["beta"]
    if intention:
        intention_posterior = beta.sel(preds="intention")
    else:
        intention_posterior = 0
    hazard_base_m = frailty_idata["posterior"]["lambda0"].sel(genders="Male")
    hazard_base_f = frailty_idata["posterior"]["lambda0"].sel(genders="Female")
    frailty = frailty_idata.posterior["frailty"]
    if retention_df.iloc[i]["Male"] == 1:
        hazard_base = hazard_base_m
    else:
```

```
        hazard_base = hazard_base_f

    full_hazard_idata = hazard_base * (
        frailty.sel(F=i).mean().item()
        * np.exp(
            beta.sel(preds="sentiment") * retention_df.iloc[i]["sentiment"]
            + np.where(intention, intention_posterior * retention_df.iloc[i]["intention"], 0)
            + beta.sel(preds="Low") * retention_df.iloc[i]["Low"]
            + beta.sel(preds="Medium") * retention_df.iloc[i]["Medium"]
            + beta.sel(preds="Finance") * retention_df.iloc[i]["Finance"]
            + beta.sel(preds="Health") * retention_df.iloc[i]["Health"]
            + beta.sel(preds="Law") * retention_df.iloc[i]["Law"]
            + beta.sel(preds="Public/Government") * retention_df.iloc[i]["Public/Government"]
            + beta.sel(preds="Sales/Marketing") * retention_df.iloc[i]["Sales/Marketing"]
        )
    )

    cum_haz_idata = cum_hazard(full_hazard_idata)
    survival_idata = survival(full_hazard_idata)
    return full_hazard_idata, cum_haz_idata, survival_idata, hazard_base


def plot_individual_frailty(retention_df, individuals=[1, 300, 700], intention=False):
    fig, axs = plt.subplots(1, 2, figsize=(20, 7))
    axs = axs.flatten()
    colors = [c_light_highlight, c_mid_highlight, c_dark_highlight]
    for i, c in zip(individuals, colors):
        haz_idata, cum_haz_idata, survival_idata, base_hazard = extract_individual_frailty(
            i, retention_df, intention
        )
        axs[0].plot(get_mean(survival_idata), label=f"individual_{i}", color=c)
        az.plot_hdi(range(12), survival_idata, ax=axs[0], fill_kwargs={"color": c})
        axs[1].plot(get_mean(cum_haz_idata), label=f"individual_{i}", color=c)
        az.plot_hdi(range(12), cum_haz_idata, ax=axs[1], fill_kwargs={"color": c})
        axs[0].set_title("Individual Survival Functions", fontsize=20)
        axs[1].set_title("Individual Cumulative Hazard Functions", fontsize=20)
    az.plot_hdi(
        range(12),
        survival(base_hazard),
        color="lightblue",
        ax=axs[0],
        fill_kwargs={"label": "Baseline Survival"},
```

```
    )
    az.plot_hdi(
        range(12),
        cum_hazard(base_hazard),
        color="lightblue",
        ax=axs[1],
        fill_kwargs={"label": "Baseline Hazard"},
    )
    axs[0].legend()
    axs[1].legend()


plot_individual_frailty(retention_df, [0, 1, 2], intention=True)
```




```
retention_df.iloc[0:3, :]
```

	gender	field	level	sentiment	intention	left	month	Male	Low	Medium
0	F	Education and Training	Low	8	5	0	12	False	True	False
1	F	Education and Training	Medium	8	3	1	11	False	False	True
2	F	Education and Training	Low	10	7	1	9	False	True	False

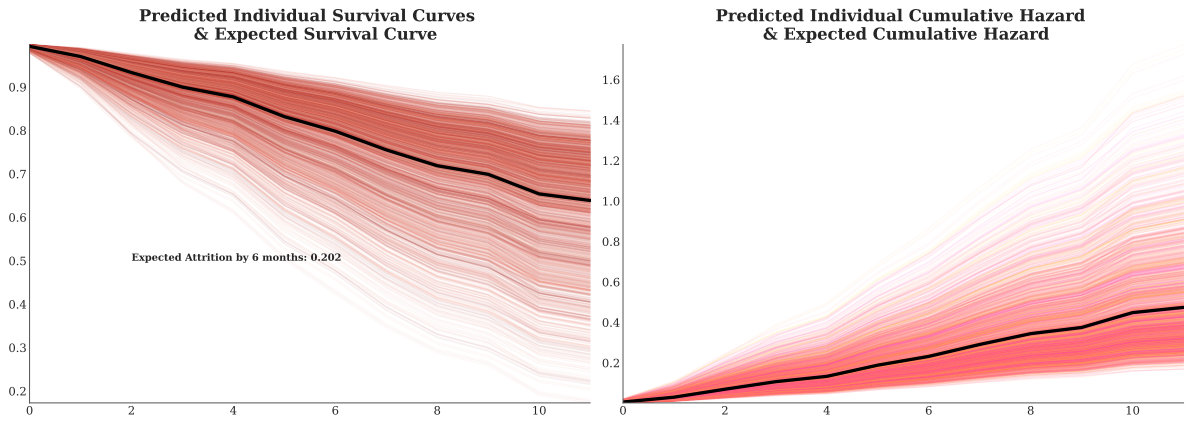
```
def create_predictions(retention_df, intention=False):
    cum_haz = {}
    surv = {}
    for i in range(len(retention_df)):
        haz_idata, cum_haz_idata, survival_idata, base_hazard = extract_individual_frailty(
            i, retention_df, intention
```

```
        )
        cum_haz[i] = get_mean(cum_haz_idata)
        surv[i] = get_mean(survival_idata)
    cum_haz = pd.DataFrame(cum_haz)
    surv = pd.DataFrame(surv)
    return cum_haz, surv


cum_haz_frailty_df, surv_frailty_df = create_predictions(retention_df, intention=True)
surv_frailty_df
```

	0	1	2	3	4	5	6	7	8	9
0	0.994556	0.996365	0.992358	0.997342	0.990454	0.990598	0.995474	0.994158	0.989497	0.9914
1	0.976164	0.984034	0.966659	0.988308	0.958499	0.959124	0.980155	0.974456	0.954441	0.9626
2	0.941784	0.960767	0.919169	0.971172	0.900008	0.901449	0.951381	0.937699	0.890533	0.9097
3	0.907167	0.937054	0.872055	0.953601	0.842719	0.844917	0.922231	0.900797	0.828350	0.8577
4	0.883206	0.920473	0.839871	0.941238	0.804025	0.806696	0.901944	0.875320	0.786576	0.8223
5	0.841317	0.891119	0.784477	0.919207	0.738289	0.741728	0.866252	0.830905	0.716087	0.7619
6	0.805701	0.865777	0.738252	0.900031	0.684312	0.688316	0.835668	0.793280	0.658668	0.7119
7	0.765764	0.836917	0.687413	0.877998	0.625922	0.630461	0.801096	0.751245	0.597055	0.6573
8	0.730911	0.811318	0.643918	0.858284	0.576814	0.581753	0.770677	0.714702	0.545688	0.6110
9	0.711763	0.797083	0.620380	0.847249	0.550582	0.555709	0.753860	0.694673	0.518426	0.5862
10	0.658210	0.756579	0.555919	0.815554	0.480025	0.485554	0.706413	0.638904	0.445748	0.5186
11	0.642250	0.744299	0.537113	0.805857	0.459813	0.465430	0.692147	0.622350	0.425118	0.4991

```
cm_subsection = np.linspace(0, 1, 120)
colors_m = [cm.Reds(x) for x in cm_subsection]
colors = [cm.spring(x) for x in cm_subsection]


fig, axs = plt.subplots(1, 2, figsize=(20, 7))
axs = axs.flatten()
cum_haz_frailty_df.plot(legend=False, color=colors, alpha=0.05, ax=axs[1])
axs[1].plot(cum_haz_frailty_df.mean(axis=1), color="black", linewidth=4)
axs[1].set_title(
    "Predicted Individual Cumulative Hazard \n & Expected Cumulative Hazard", fontsize=20
)

surv_frailty_df.plot(legend=False, color=colors_m, alpha=0.05, ax=axs[0])
axs[0].plot(surv_frailty_df.mean(axis=1), color="black", linewidth=4)
```

```
axs[0].set_title("Predicted Individual Survival Curves \n  & Expected Survival Curve", fontsi
axs[0].annotate(
    f"Expected Attrition by 6 months: {np.round(1-surv_frailty_df.mean(axis=1).iloc[6], 3)}"
    (2, 0.5),
    fontsize=12,
    fontweight="bold",
);
```



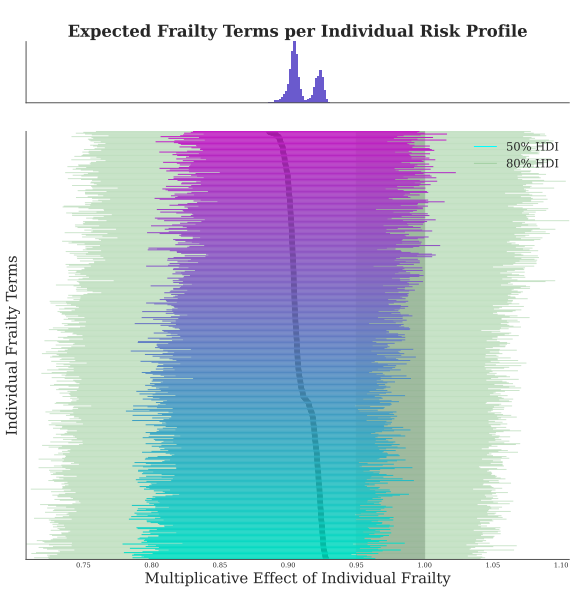
7.1 Plotting the effects of the Frailty Terms

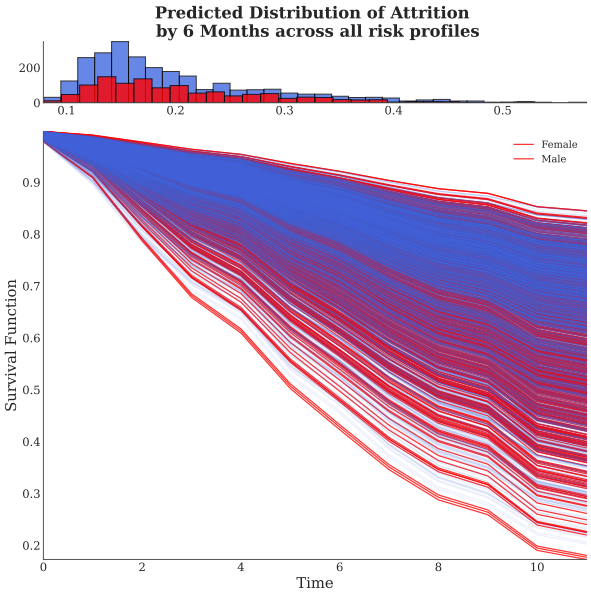
```
beta_individual_all = frailty_idata["posterior"]["frailty"]
predicted_all = beta_individual_all.mean(("chain", "draw"))
predicted_all = predicted_all.sortby(predicted_all, ascending=False)
beta_individual = beta_individual_all.sel(F=range(500))
predicted = beta_individual.mean(("chain", "draw"))
predicted = predicted.sortby(predicted, ascending=False)
ci_lb = beta_individual.quantile(0.025, ("chain", "draw")).sortby(predicted)
ci_ub = beta_individual.quantile(0.975, ("chain", "draw")).sortby(predicted)
hdi = az.hdi(beta_individual, hdi_prob=0.5).sortby(predicted)
hdi2 = az.hdi(beta_individual, hdi_prob=0.8).sortby(predicted)
```

```
cm_subsection = np.linspace(0, 1, 500)
colors = [cm.cool(x) for x in cm_subsection]

fig = plt.figure(figsize=(20, 10))
gs = fig.add_gridspec(
    2,
```

```
    2,
    height_ratios=(1, 7),
    left=0.1,
    right=0.9,
    bottom=0.1,
    top=0.9,
    wspace=0.05,
    hspace=0.05,
)

# Create the Axes.
ax = fig.add_subplot(gs[1, 0])
ax.set_yticklabels([])
ax_histx = fig.add_subplot(gs[0, 0], sharex=ax)
ax_histx.set_title("Expected Frailty Terms per Individual Risk Profile", fontsize=20)
ax_histx.hist(predicted_all, bins=30, color="slateblue")
ax_histx.set_yticklabels([])
ax_histx.tick_params(labelsize=8)
ax.set_ylabel("Individual Frailty Terms", fontsize=18)
ax.tick_params(labelsize=8)
ax.hlines(
    range(len(predicted)),
    hdi.sel(hdi="lower").to_array(),
    hdi.sel(hdi="higher").to_array(),
    color=colors,
    label="50% HDI",
    linewidth=0.8,
)
ax.hlines(
    range(len(predicted)),
    hdi2.sel(hdi="lower").to_array(),
    hdi2.sel(hdi="higher").to_array(),
    color="green",
    alpha=0.2,
    label="80% HDI",
    linewidth=0.8,
)
ax.set_xlabel("Multiplicative Effect of Individual Frailty", fontsize=18)
ax.legend()
ax.fill_betweenx(range(len(predicted)), 0.95, 1.0, alpha=0.4, color="grey")

ax1 = fig.add_subplot(gs[1, 1])
```

```
f_index = retention_df[retention_df["gender"] == "F"].index
index = retention_df.index
surv_frailty_df[list(range(len(f_index)))].plot(ax=ax1, legend=False, color="red", alpha=0.8)
surv_frailty_df[list(range(len(f_index), len(index), 1))].plot(
    ax=ax1, legend=False, color="royalblue", alpha=0.1
)
ax1_hist = fig.add_subplot(gs[0, 1])
f_index = retention_df[retention_df["gender"] == "F"].index
ax1_hist.hist(
    (1 - surv_frailty_df[list(range(len(f_index), len(index), 1))].iloc[6]),
    bins=30,
    color="royalblue",
    ec="black",
    alpha=0.8,
)
ax1_hist.hist(
    (1 - surv_frailty_df[list(range(len(f_index)))].iloc[6]),
    bins=30,
    color="red",
    ec="black",
    alpha=0.8,
)
ax1.set_xlabel("Time", fontsize=18)
ax1_hist.set_title(
    "Predicted Distribution of Attrition \n by 6 Months across all risk profiles", fontsize=
)
ax1.set_ylabel("Survival Function", fontsize=18)
ax.scatter(predicted, range(len(predicted)), color="black", ec="black", s=30)

# Create a manual legend without Line2D
ax1.legend(["Female", "Male"], loc="upper right", fontsize=12)
```


Expected Frailty Terms per Individual Risk Profile
50% HDI
80% HDI
Individual Frailty Terms
0.75
0.80
0.85
0.90
0.95
1.00
1.05
1.10
Multiplicative Effect of Individual Frailty


Predicted Distribution of Attrition
by 6 Months across all risk profiles
200
0
0.1
0.2
0.3
0.4
0.5
Female
Male
Survival Function
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
10
Time