

# Fitting the Weibull model

SURVIVAL ANALYSIS IN PYTHON



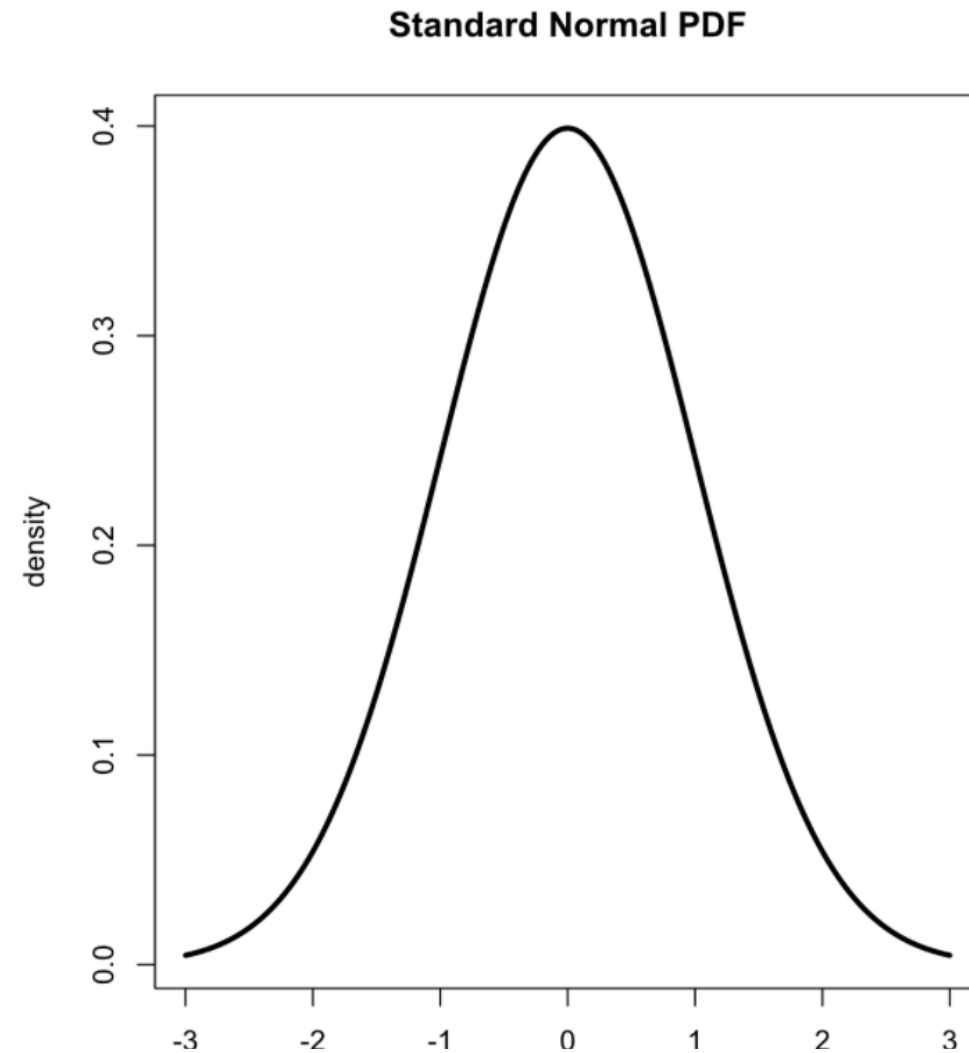
**Shae Wang**

Senior Data Scientist

# Probability distributions

## A probability distribution

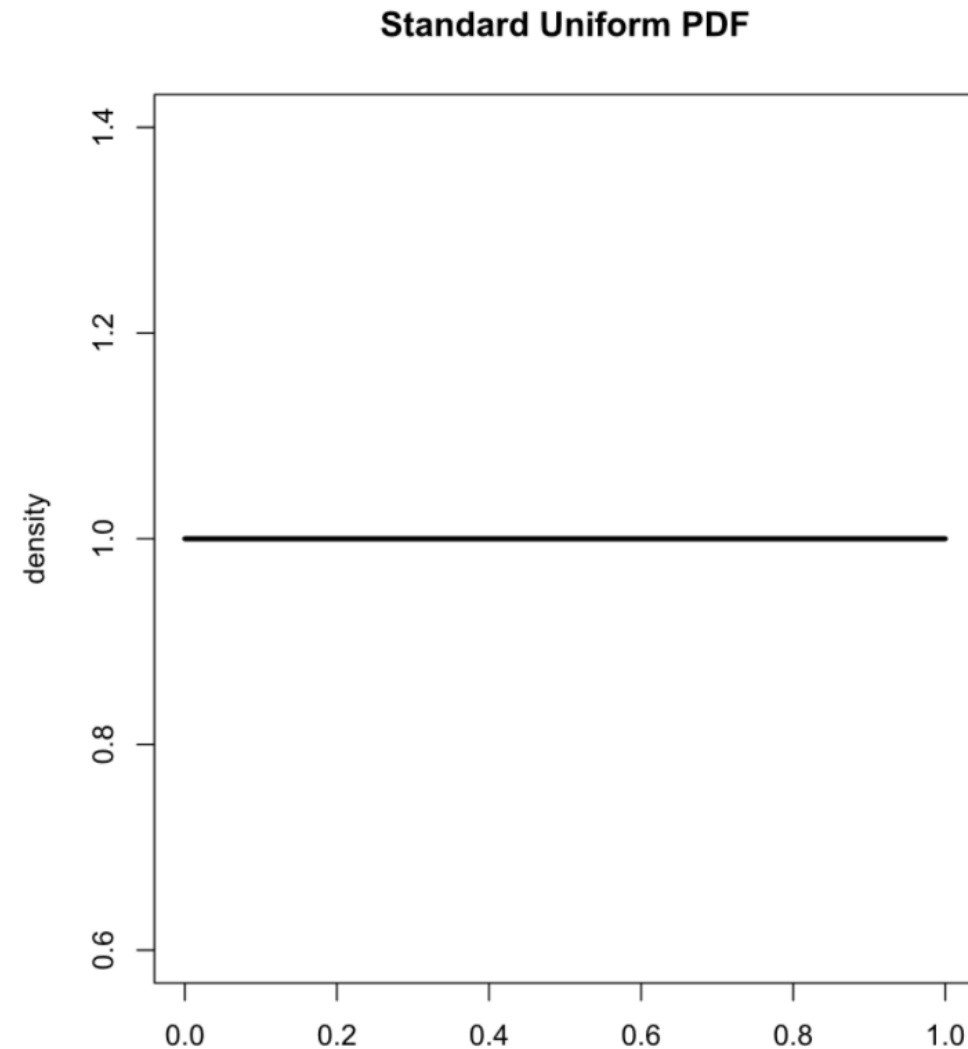
A mathematical function that describes the probability of different event outcomes.



# Probability distributions

## A probability distribution

A mathematical function that describes the probability of different event outcomes.



# Introducing the Weibull distribution

## The Weibull distribution

A continuous probability distribution that models time-to-event data very well (but originally applied to model particle size distribution).

## The Weibull probability density function

$$f(x; \lambda, k) = \frac{k}{\lambda} \left( \frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k}$$

$$x \geq 0, k > 0, \lambda > 0$$

# Introducing the Weibull distribution

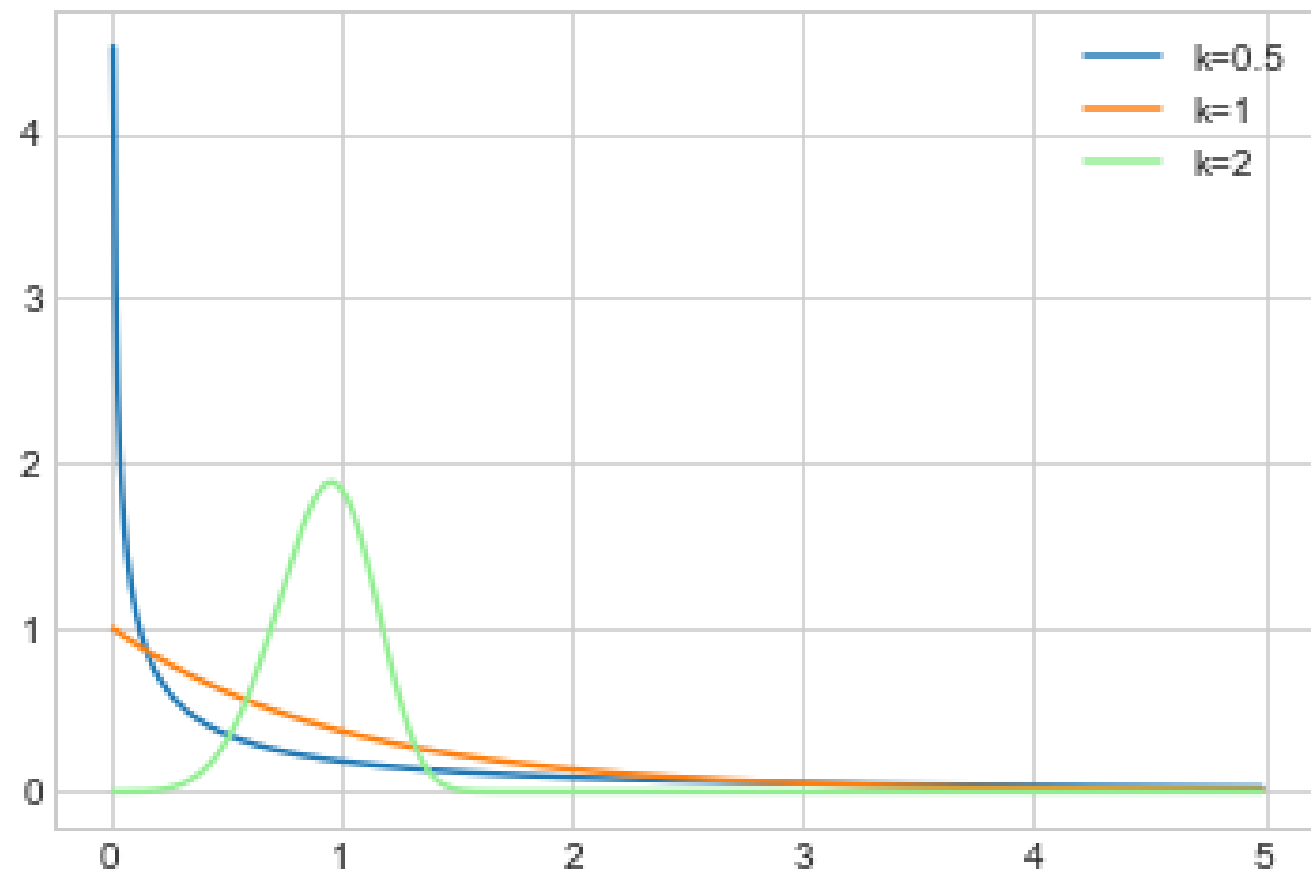
$k$

Determines the shape

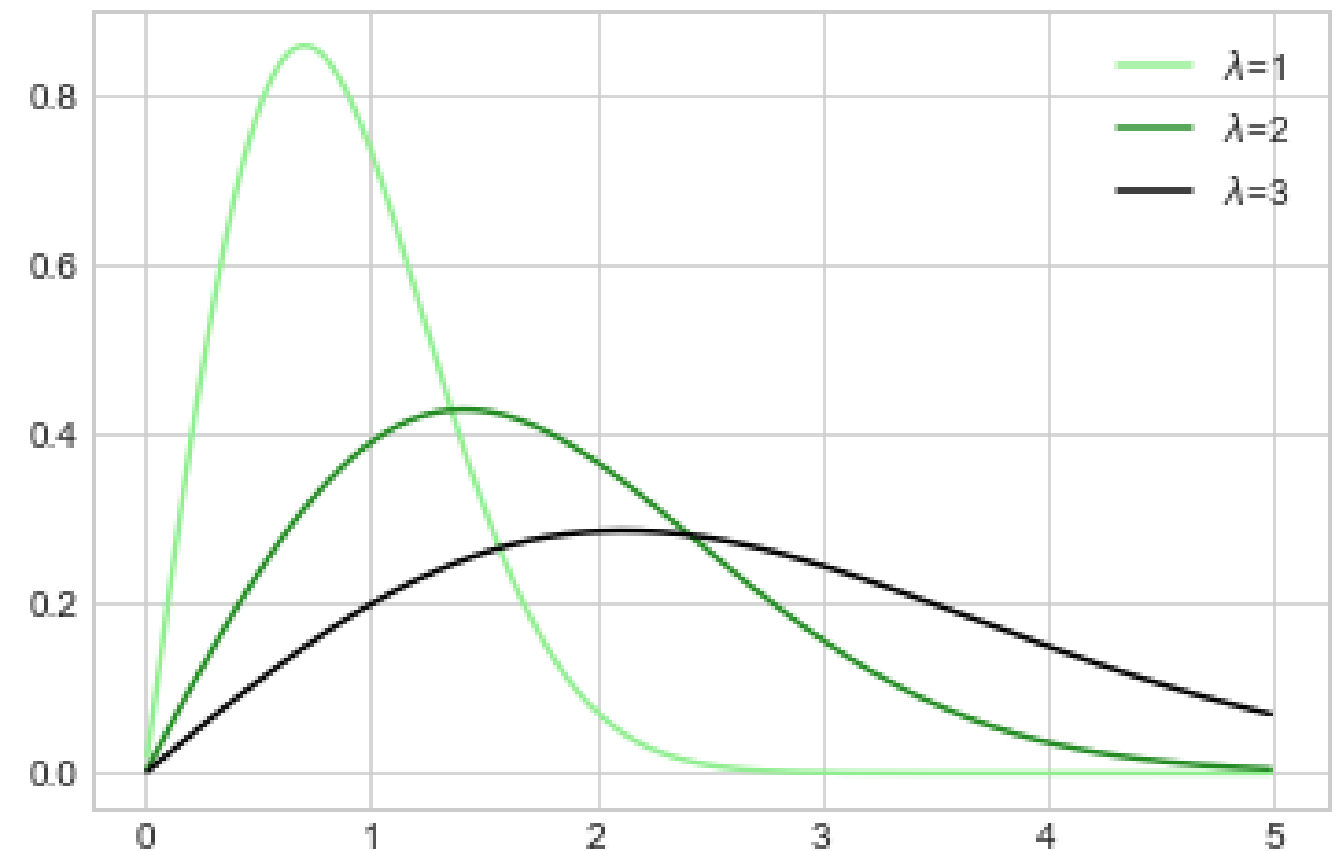
$\lambda$

Determines the scale

Weibull Probability Density Function ( $\lambda=1$ )

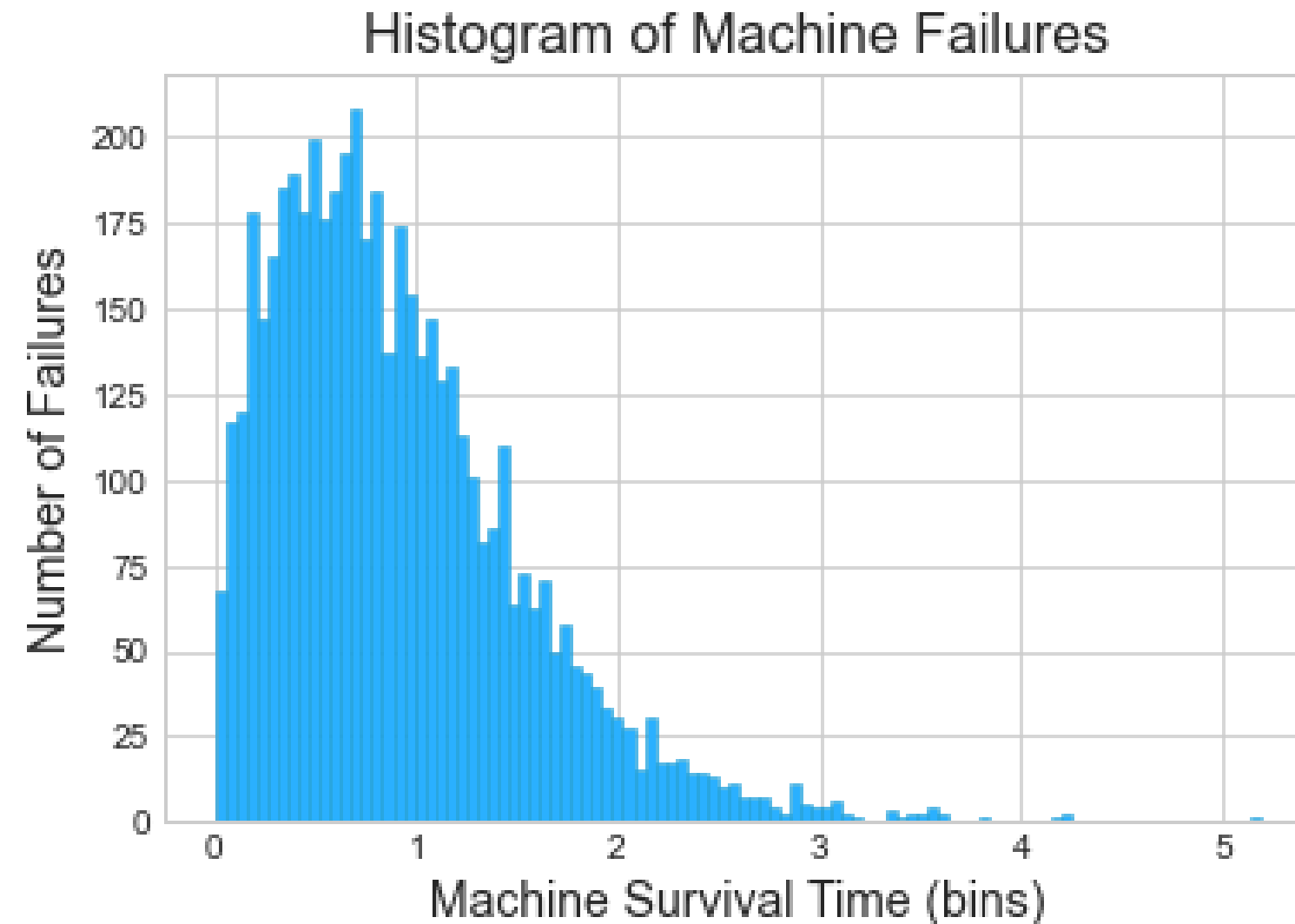


Weibull Probability Density Function ( $k=2$ )



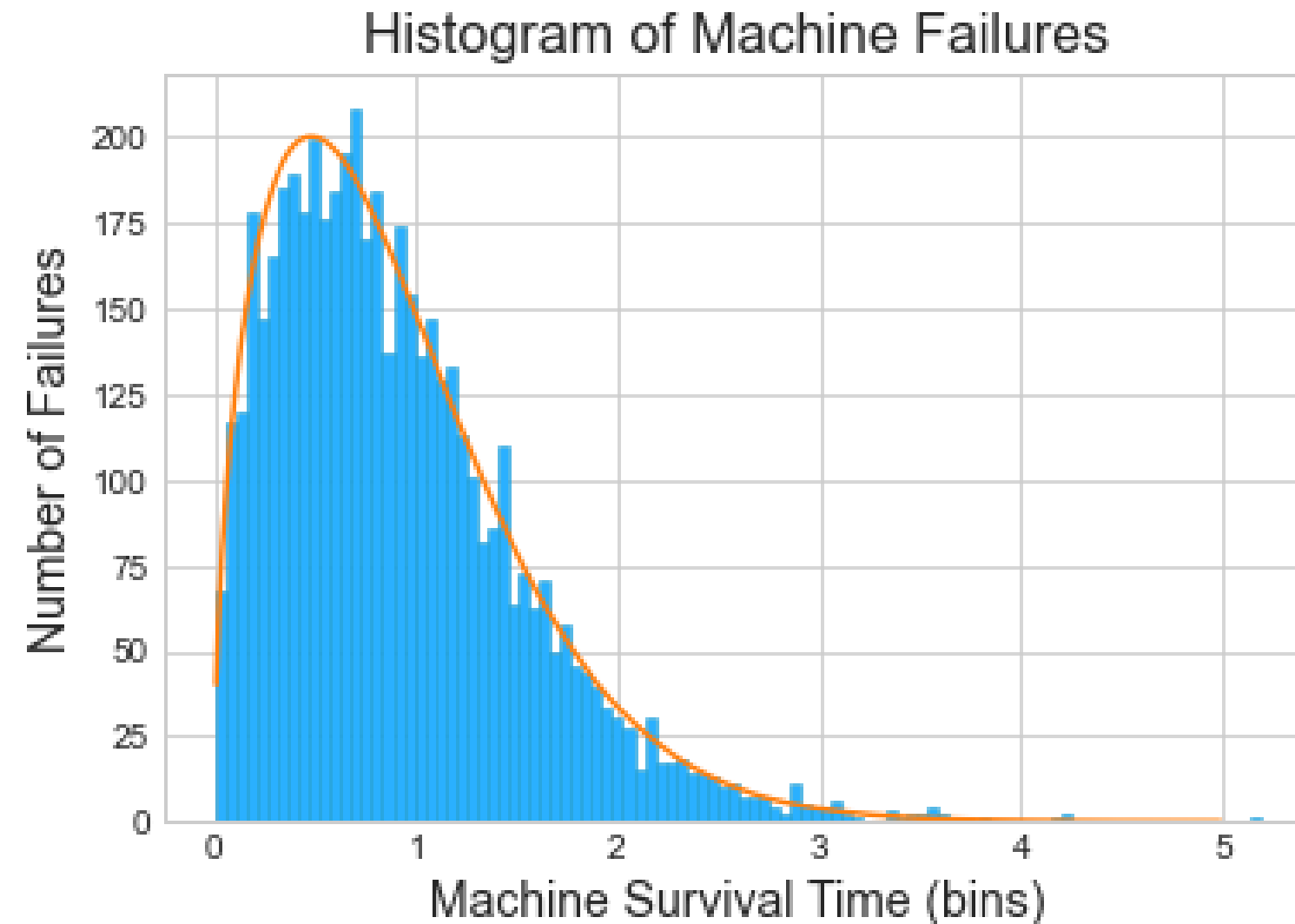
# Fitting the Weibull distribution to data

A company maintains a fleet of machines that are prone to failure...



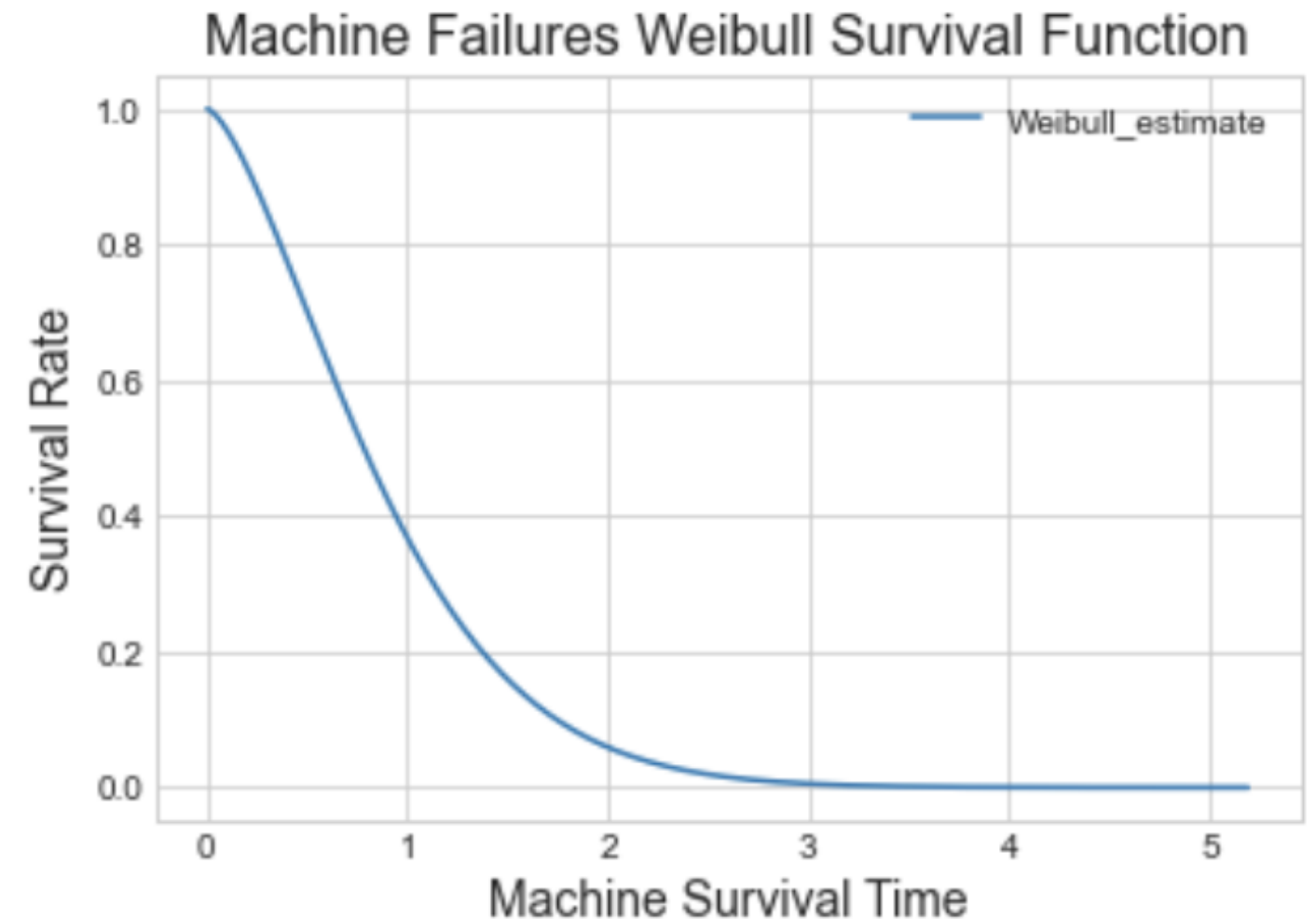
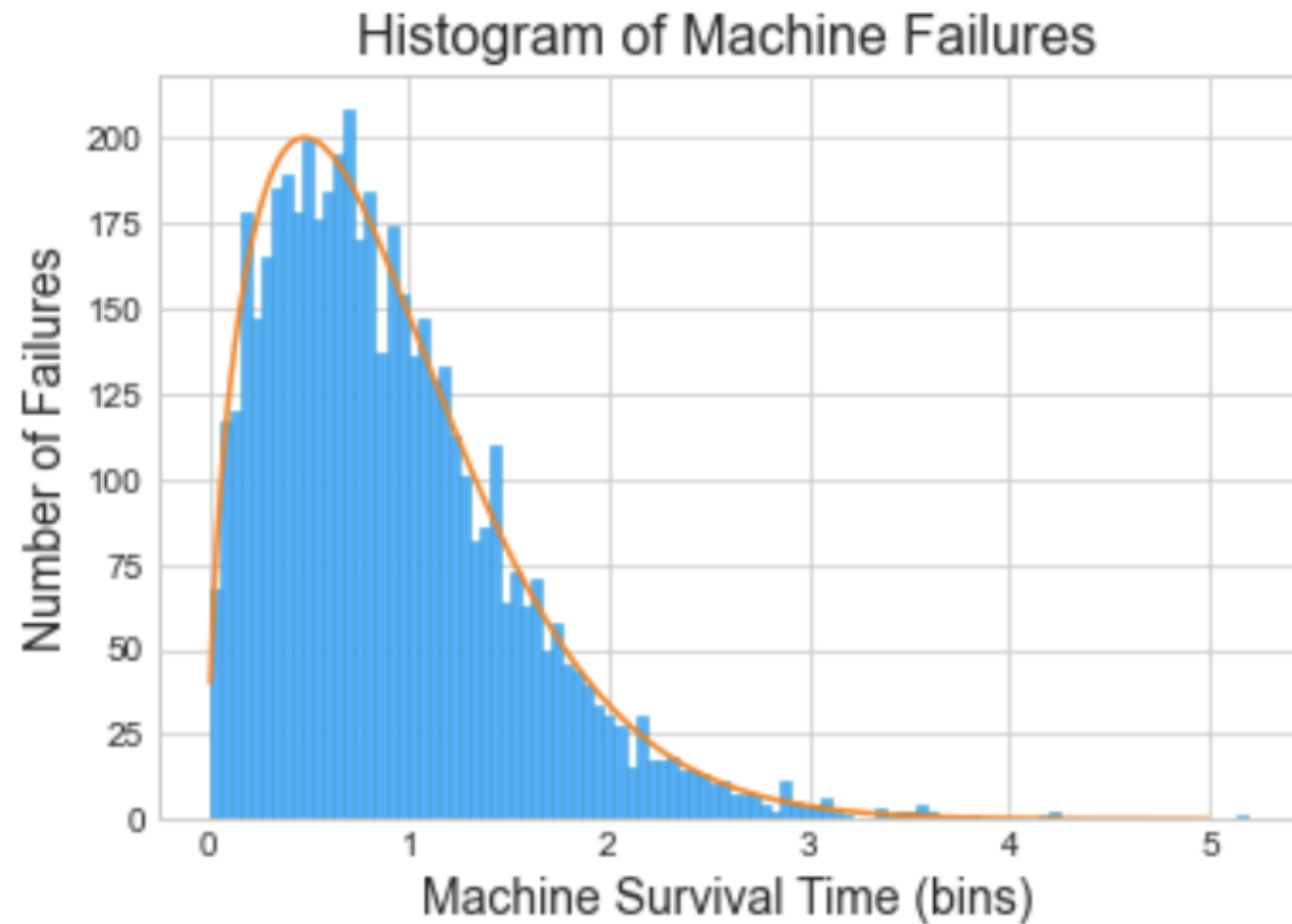
# Fitting the Weibull distribution to data

A company maintains a fleet of machines that are prone to failure...



# From Weibull distribution to survival function

$$f(x; \lambda, k) = \frac{k}{\lambda} \left( \frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k} \rightarrow S(t) = e^{-(t/\lambda)^k}$$





# The knobs: k and lambda

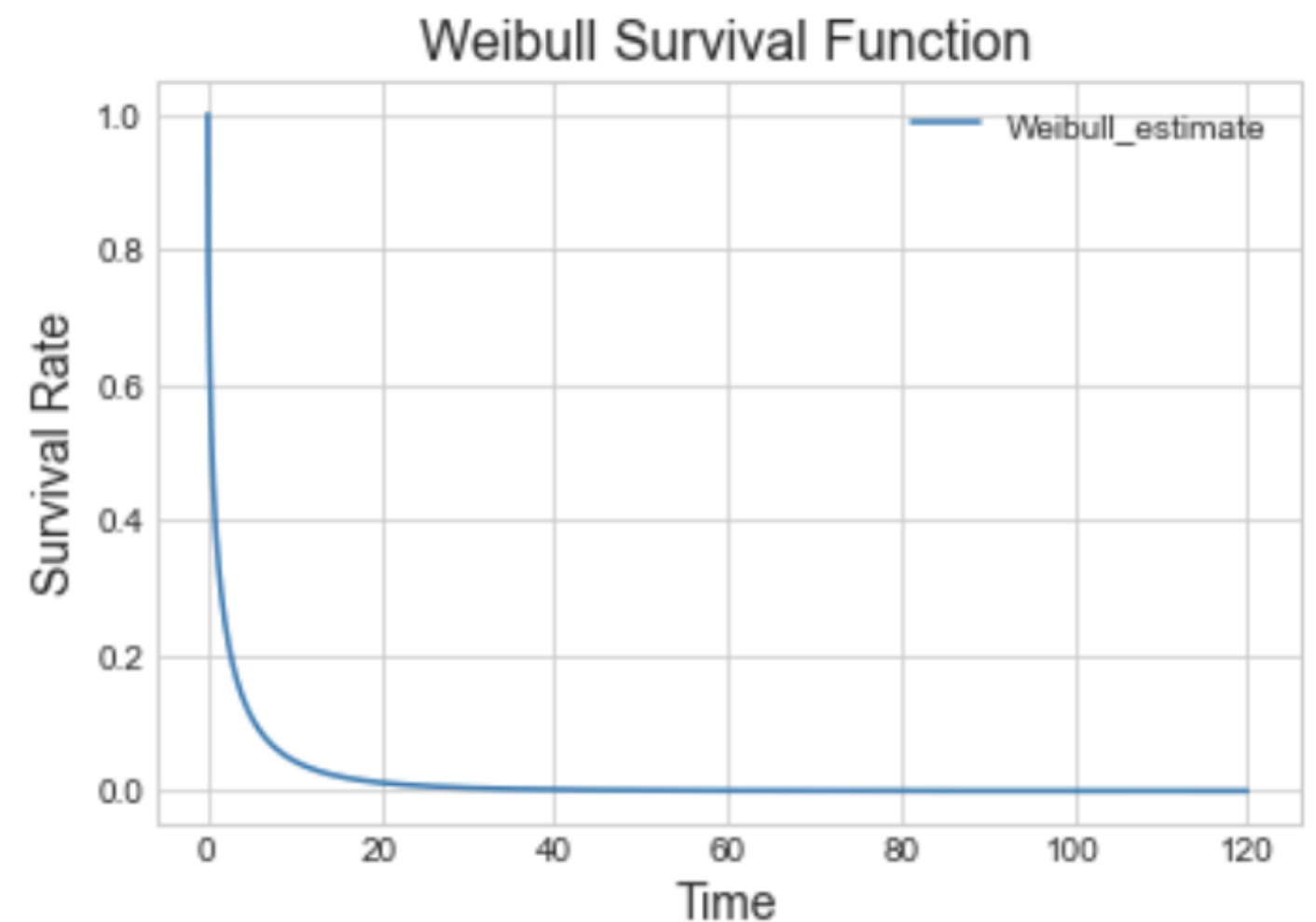
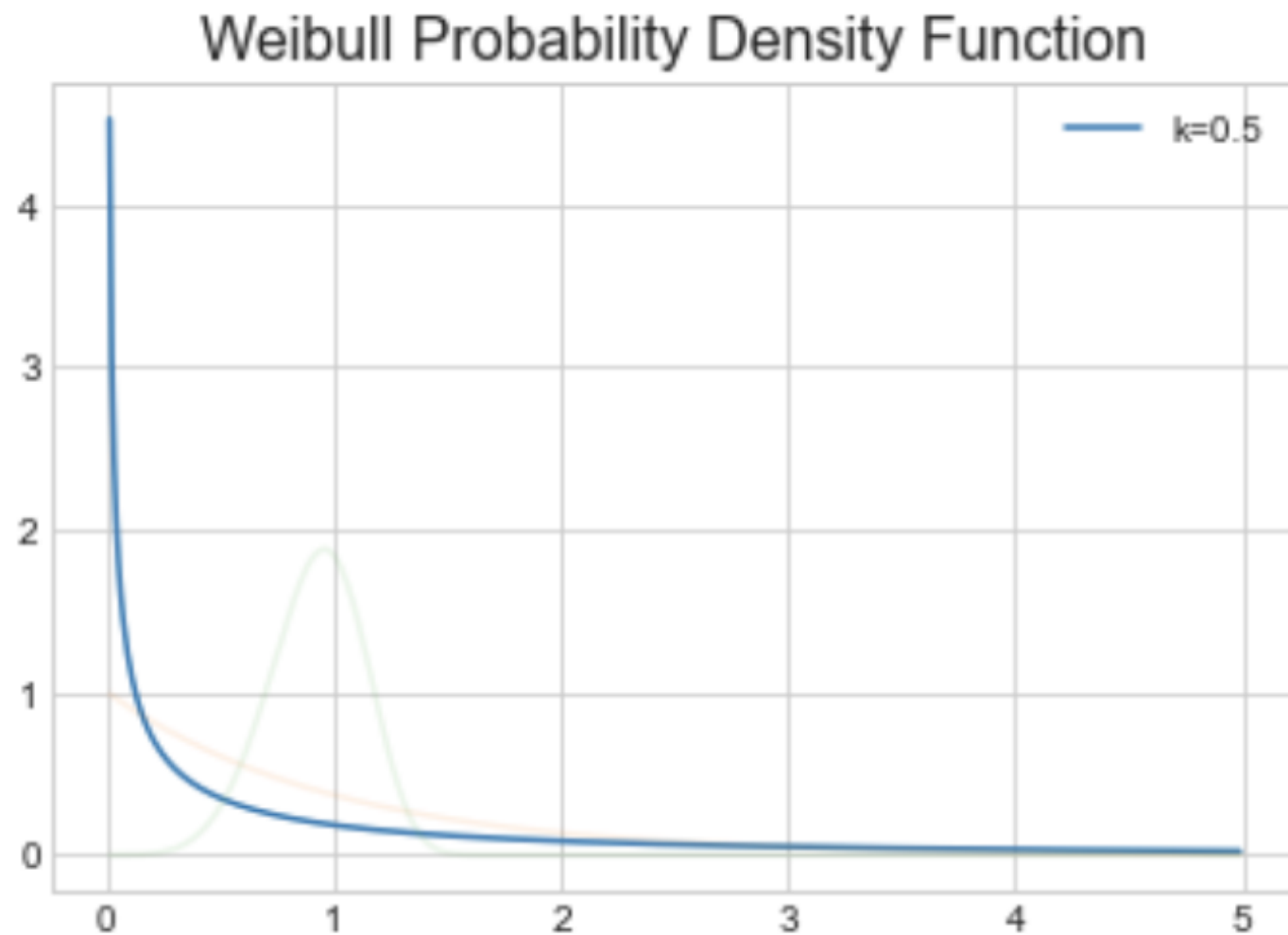
## k and $\lambda$

- k (or  $\rho$ ): determines the shape
- $\lambda$ : determines the scale (indicates when 63.2% of the population has experienced the event)

$$f(x; \lambda, k) = \frac{k}{\lambda} \left( \frac{x}{\lambda} \right)^{k-1} e^{-(x/\lambda)^k} \rightarrow f(x; \lambda, k = 3) = \frac{3}{\lambda} \left( \frac{x}{\lambda} \right)^2 e^{-(x/\lambda)^3}$$

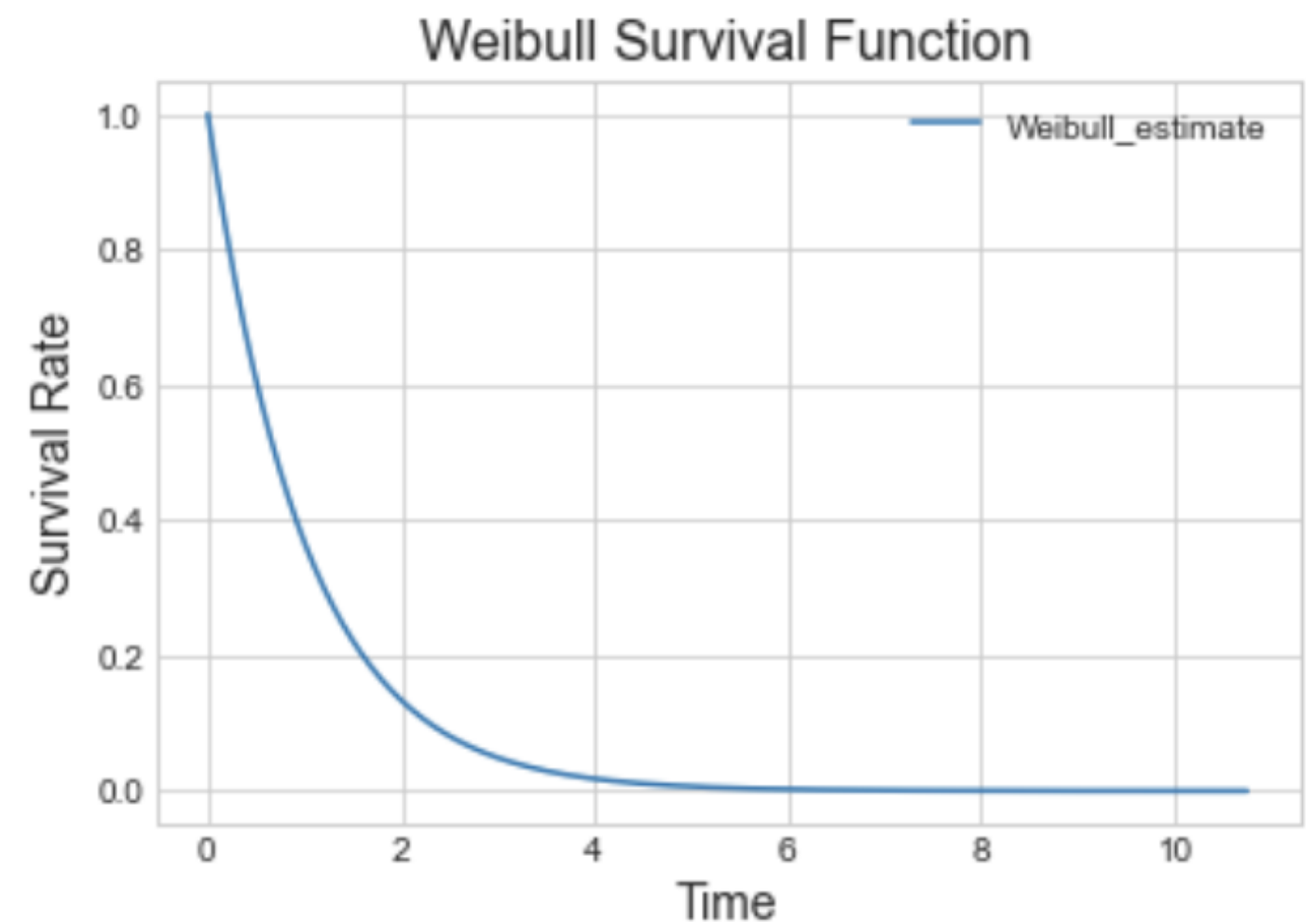
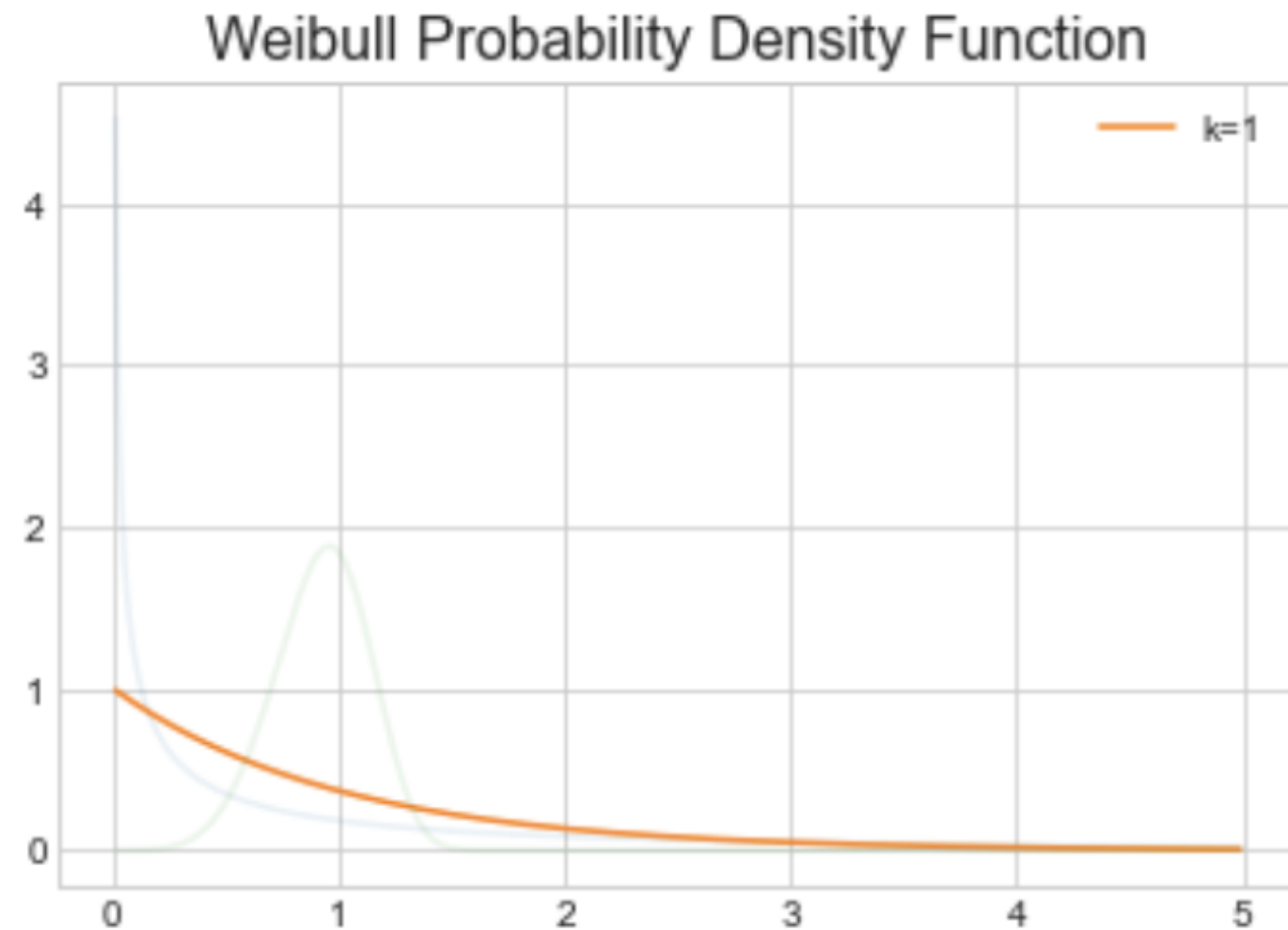
- Weibull distribution: the failure/event rate is proportional to a power of time.

# Interpreting $k$ (or $\rho$ )



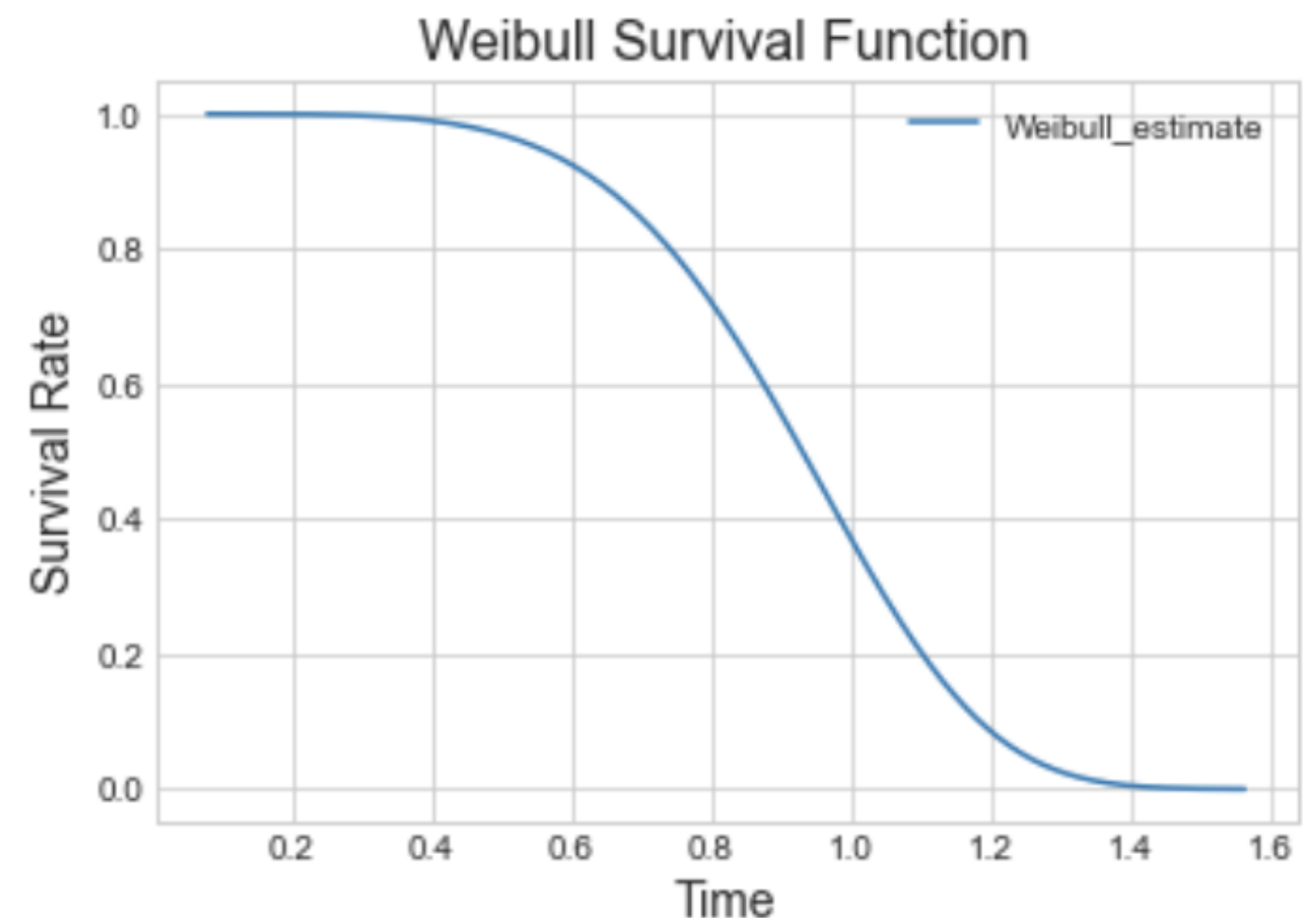
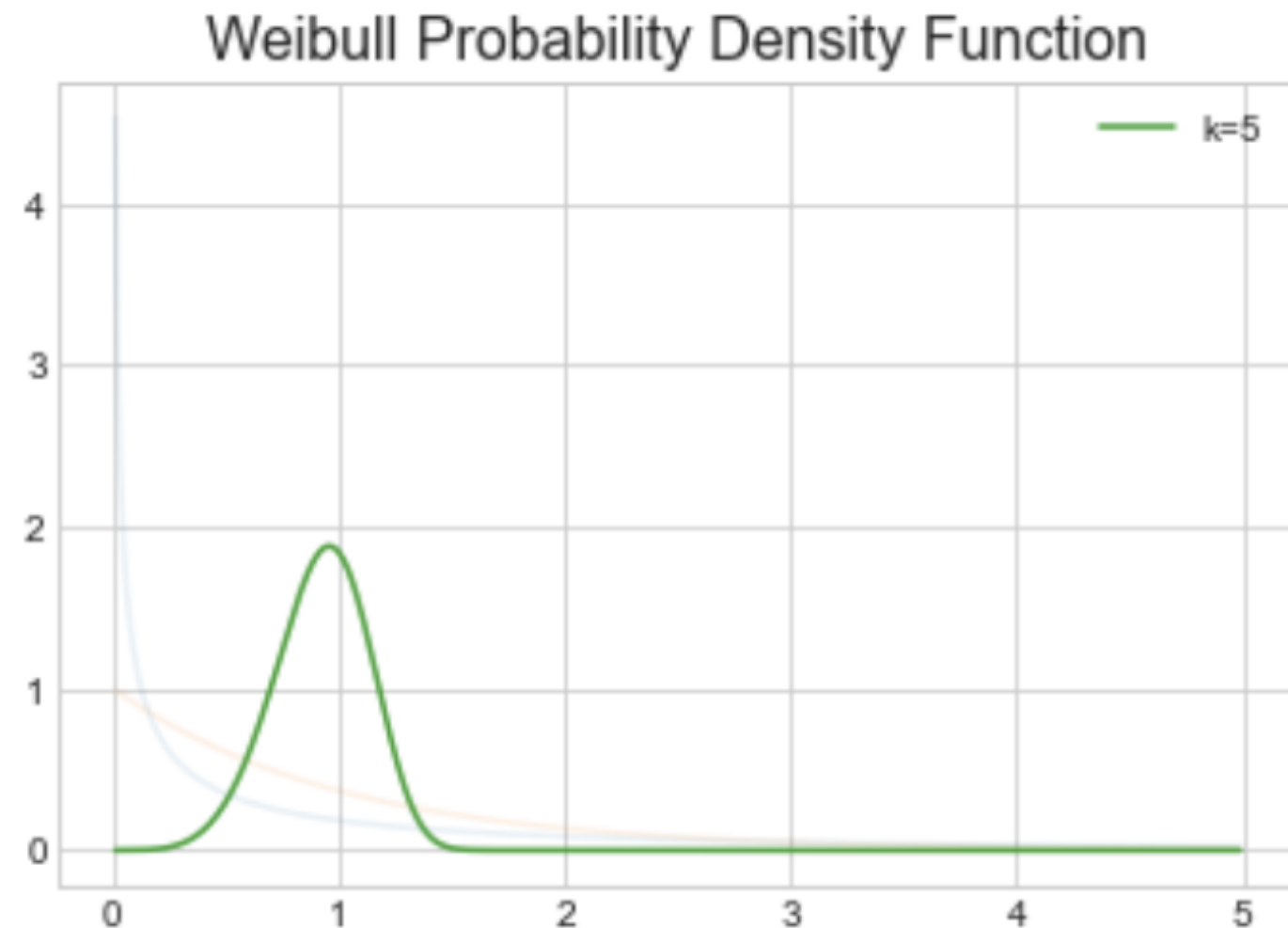
- When  $k < 1$ , the failure/event rate decreases over time.

# Interpreting $k$ (or $\rho$ )



- When  $k = 1$ , the failure/event rate is constant over time.

# Interpreting $k$ (or $\rho$ )



- When  $k > 1$ , the failure/event rate increases over time.

# Survival analysis with Weibull distribution

1. Import the `WeibullFitter` class

```
from lifelines import WeibullFitter
```

2. Instantiate a `WeibullFitter` class

```
wb = WeibullFitter()
```

3. Call `.fit()` to fit the estimator to the data

```
wb.fit(durations, event_observed)
```

4. Access `.survival_function_`, `.lambda_`, `.rho_`, `.summary`, `.predict()`

# Example Weibull model

DataFrame name: mortgage\_df

id	duration	paid_off
1	25	0
2	17	1
3	5	0
...	...	...
1000	30	1

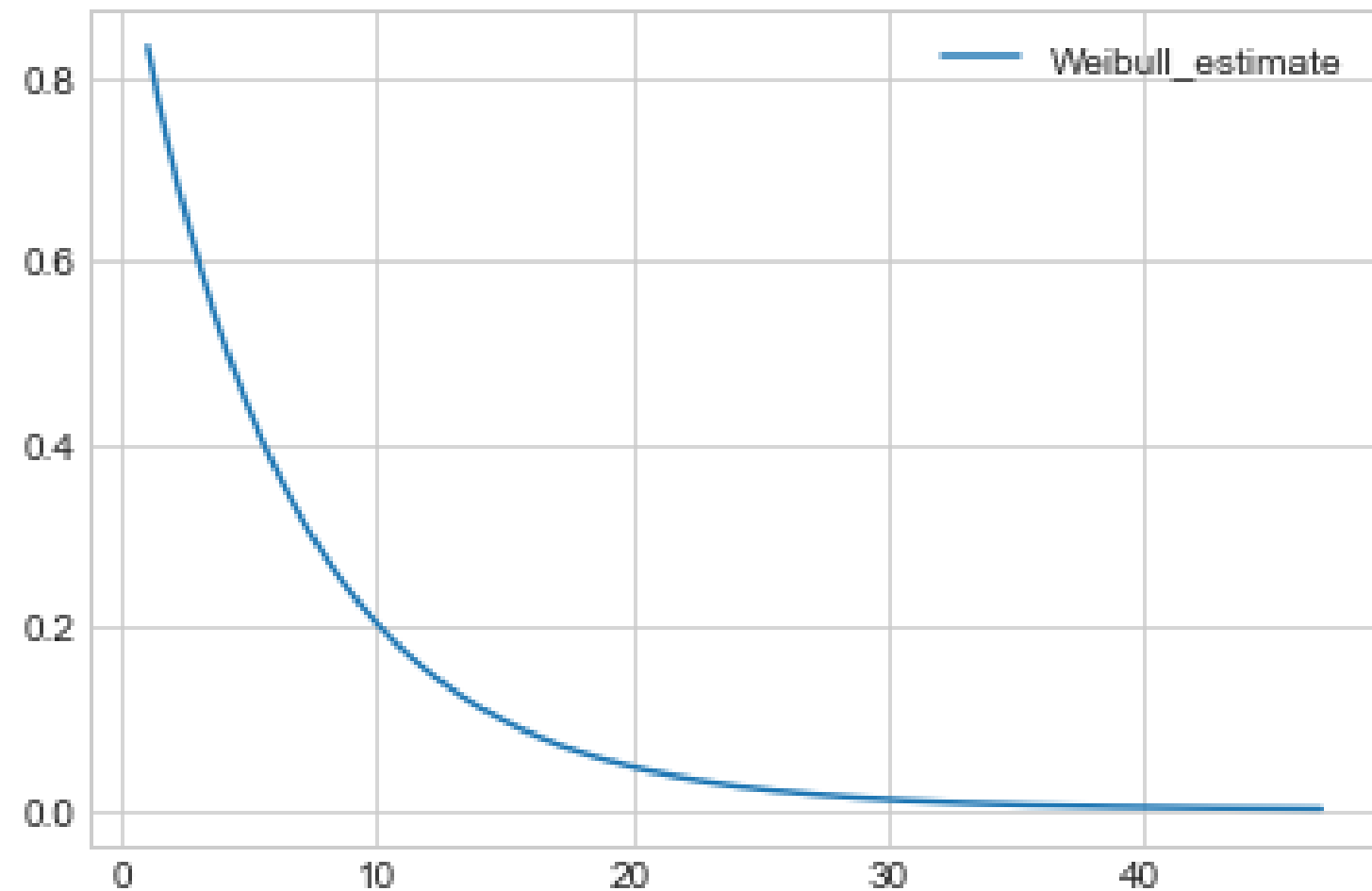
```
from lifelines import WeibullFitter
```

```
wb = WeibullFitter()
```

```
wb.fit(durations=mortgage_df["duration"],  
       event_observed=mortgage_df["paid_off"])
```

# Example Weibull model

```
wb.survival_function_.plot()  
plt.show()
```



```
print(wb.lambda_, wb.rho_)
```

```
6.11  0.94
```

```
print(wb.predict(20))
```

```
0.05
```

**Let's practice!**  
SURVIVAL ANALYSIS IN PYTHON



# Weibull model with covariates

SURVIVAL ANALYSIS IN PYTHON

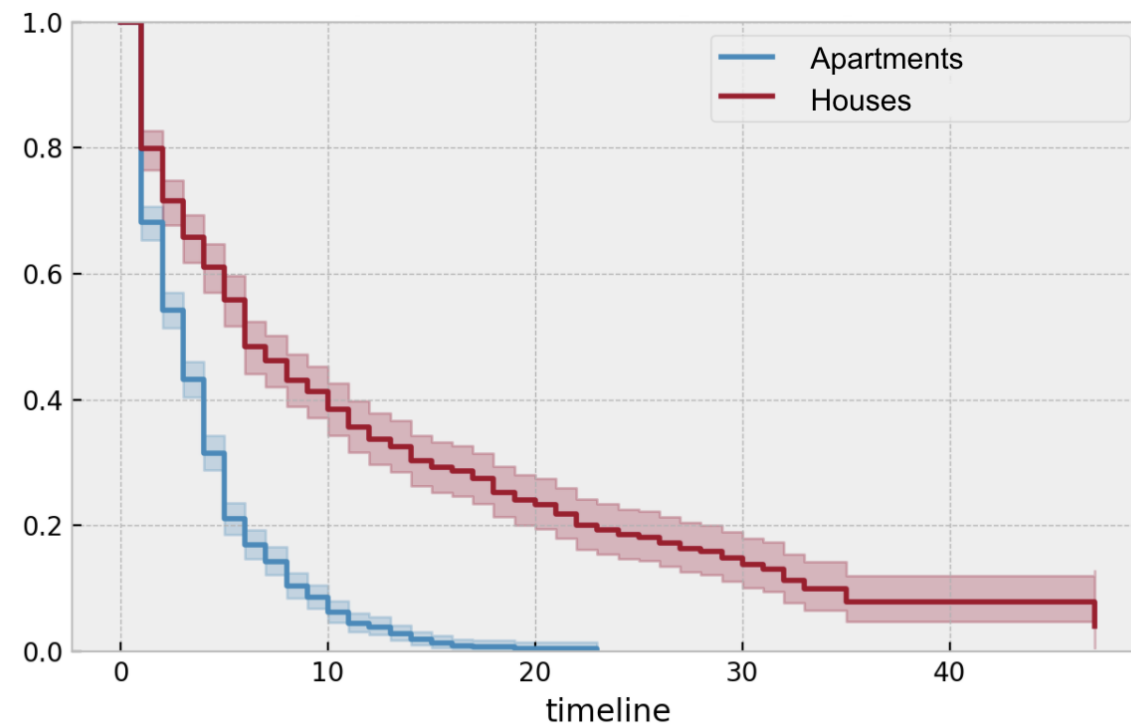


**Shae Wang**

Senior Data Scientist

# Comparing survival functions

Compare groups using the Kaplan-Meier estimator:



Compare groups using the log-rank test:

```
<lifelines.StatisticalResult: logrank_test>  
null_distribution = chi squared  
degrees_of_freedom = 1  
test_name = logrank_test  
test_statistic    p    -log2(p)  
0.09 0.77      0.38
```

**Q: How do we assess if/how one or multiple continuous variables affect the survival function?**

# Survival regression

- A method that models survival functions with covariates
- Quantifies how each covariate affects the survival function

$$Y_i = f(X_i, \beta)$$

$Y_i$  : durations,  $X_i$  : covariates

- Example covariates: age, weight, country

# The Accelerated Failure Time (AFT) model

Population A :  $S_A(t)$

Population B :  $S_B(t)$

$$S_A(t) = S_B(t * \lambda)$$

- $S_B(t)$  is speeding up (accelerating) or slowing down (decelerating) along  $S_A(t)$  by a factor of  $\lambda$ .
- AFT models this acceleration/deceleration relationship based on model covariates.
- When a covariate changes from  $a$  to  $b$ , time-to-event speeds up or slows down by the accelerated failure rate  $\lambda$ .
- Example:  $S_{dog}(t) = S_{human}(t * 7)$

# Data for survival regression

DataFrame example: `mortgage_df`

id	property_type	principal	interest	property_tax	credit_score	duration	paid_off
1	house	1275	0.035	0.019	780	25	0
2	apartment	756	0.028	0.020	695	17	1
3	apartment	968	0.029	0.017	810	5	0
...	...	...	...	...	...	...	...
1000	house	1505	0.041	0.023	750	30	1

# Combining Weibull with AFT: the Weibull AFT model

- DataFrame: `mortgage_df`
- Covariates:
  - `property_type` is **replaced** with a dummy variable:
    - `house` : 1 if "house", 0 if "apartment"
  - `principal`
  - `interest`
  - `property_tax`
  - `credit_score`

1. Import and instantiate the `WeibullAFTFitter` class

```
from lifelines import WeibullAFTFitter  
aft = WeibullAFTFitter()
```

2. Call `.fit()` to fit the estimator to the data

```
aft.fit(df=mortgage_df,  
        duration_col="duration",  
        event_col="paid_off")
```

# Interpreting model output

```
print(aft.summary)
```

```
<lifelines.WeibullAFTFitter: fitted with 1808 observations, 340 censored>
```

		coef	exp(coef)	se(coef)	z	p
lambda_	house	0.04	1.04	0.01	0.99	0.32
	principal	-0.03	0.97	0.22	-1.04	0.30
	interest	0.11	1.11	0.15	1.96	0.05
	property_tax	0.31	1.36	0.27	1.15	0.25
	credit_score	-0.16	0.85	0.14	-2.33	0.02
	Intercept	3.99	54.06	0.41	9.52	<0.0005
rho_	Intercept	0.34	1.40	0.08	3.80	<0.0005

# WeibullAFTFitter with custom formula

Using `formula` to handle custom model covariates:

```
aft.fit(df=mortgage_df,  
        duration_col="duration",  
        event_col="paid_off",  
        formula="principal + interest * house")
```

Analogous to the linear model with interaction term:

$$\beta_1 \text{principal} + \beta_2 \text{interest} + \beta_3 \text{house} + \beta_4 \text{interest} \cdot \text{house}$$



# Interpreting model output

```
print(aft.summary)
```

```
<lifelines.WeibullAFTFitter: fitted with 1808 observations, 340 censored>
```

		coef	exp(coef)	se(coef)	z	p
lambda_	principal	-0.03	0.97	0.22	-1.04	0.30
	interest	0.11	1.11	0.15	1.96	0.05
	house	0.04	1.04	0.01	0.99	0.32
	interest:house	0.06	1.06	0.14	0.42	0.64
	Intercept	3.99	54.06	0.41	9.52	<0.0005
rho_	Intercept	0.34	1.40	0.08	3.80	<0.0005

**Let's practice!**  
SURVIVAL ANALYSIS IN PYTHON

# Visualization and prediction with Weibull model

SURVIVAL ANALYSIS IN PYTHON



**Shae Wang**

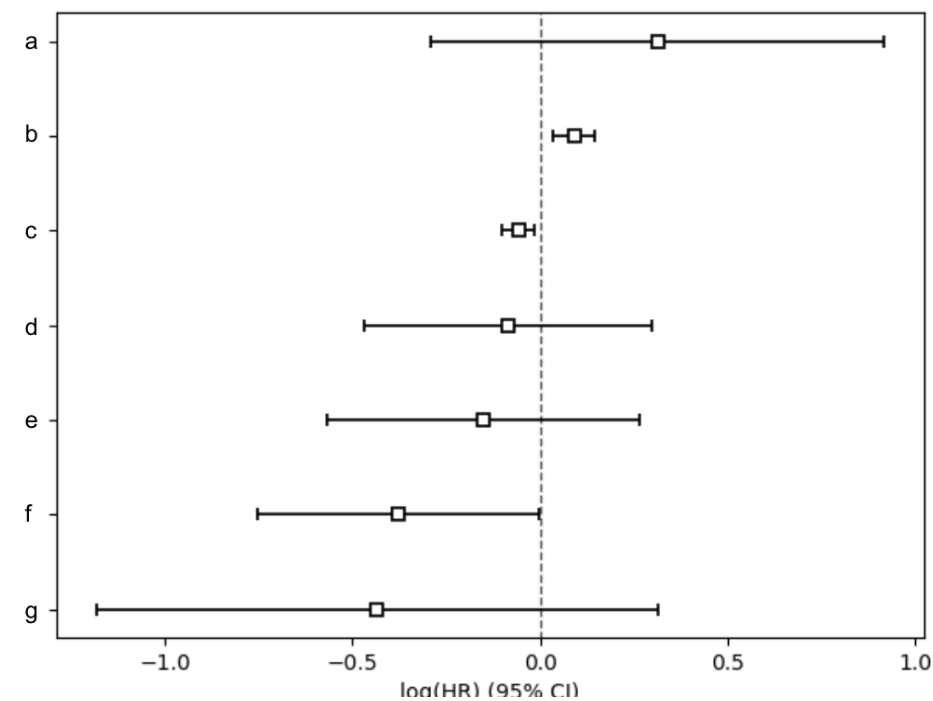
Senior Data Scientist

# .plot()

Returns a plot of the coefficients and their ranges from the 95% confidence intervals.

```
aft.plot()  
plt.show()
```

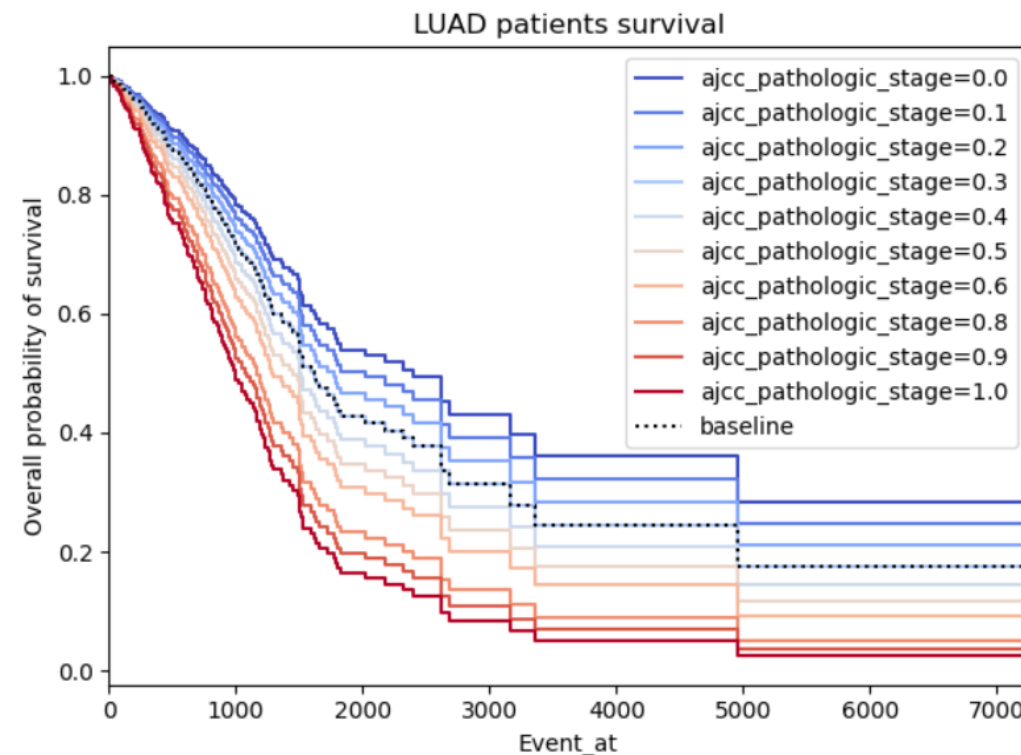
Sample plot:



# .plot\_partial\_effects\_on\_outcome()

Returns a plot comparing the baseline survival curve versus what happens when covariates are varied over values.

```
aft.plot_partial_effects_on_outcome(covariates, values)
plt.show()
```

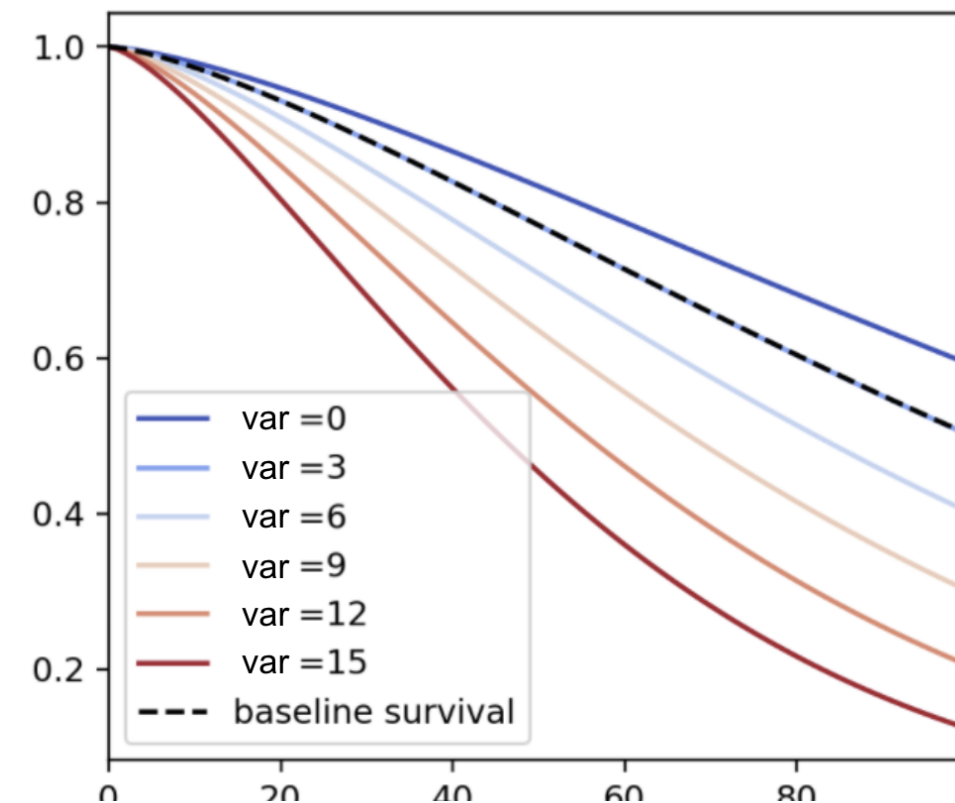


# How to plot partial effects?

```
.plot_partial_effects_on_outcome()
```

- **covariates** (string or list): covariate(s) in the original dataset that we wish to vary.
- **values** (1d or 2d iterable): values we wish the covariate to take on.
- **Baseline survival curve**: predicted survival curve at all average values in the original dataset.

```
aft.plot_partial_effects_on_outcome(  
    covariates='var',  
    values=[0, 3, 6, 9, 12, 15]  
)  
plt.show()
```



# How to plot partial effects?

- Hard-code values:

```
aft.plot_partial_effects_on_outcome(  
    covariates='a',  
    values=[0, 3, 6]  
)
```

- Multiple covariates:

```
aft.plot_partial_effects_on_outcome(  
    covariates=['a', 'b'],  
    values=[[1, 2], [1, 3], [2, 3]]  
)
```

- Use a range function:

```
aft.plot_partial_effects_on_outcome(  
    covariates='a',  
    values=np.arange(10)  
)
```

- Custom formula:

- Necessary transformations (interactions, one-hot encoding, etc.) will be made internally and automatically.

# Mortgage example

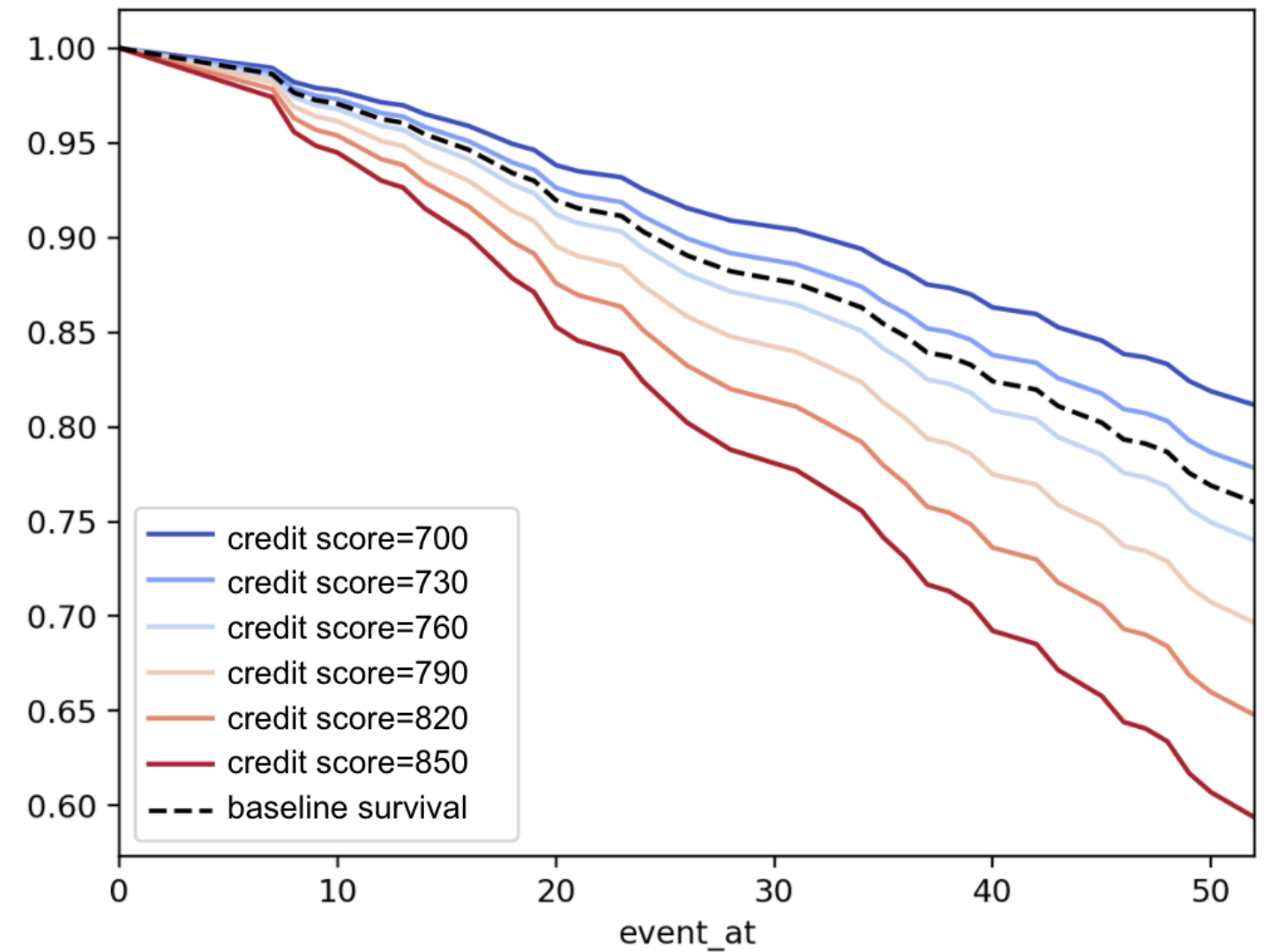
DataFrame example: `mortgage_df`

id	house	principal	interest	property_tax	credit score	duration	paid_off
1	1	1275	0.035	0.019	780	25	0
2	0	756	0.028	0.020	695	17	1
3	0	968	0.029	0.017	810	5	0
...	...	...	...	...	...	...	...
1000	1	1505	0.041	0.023	750	30	1



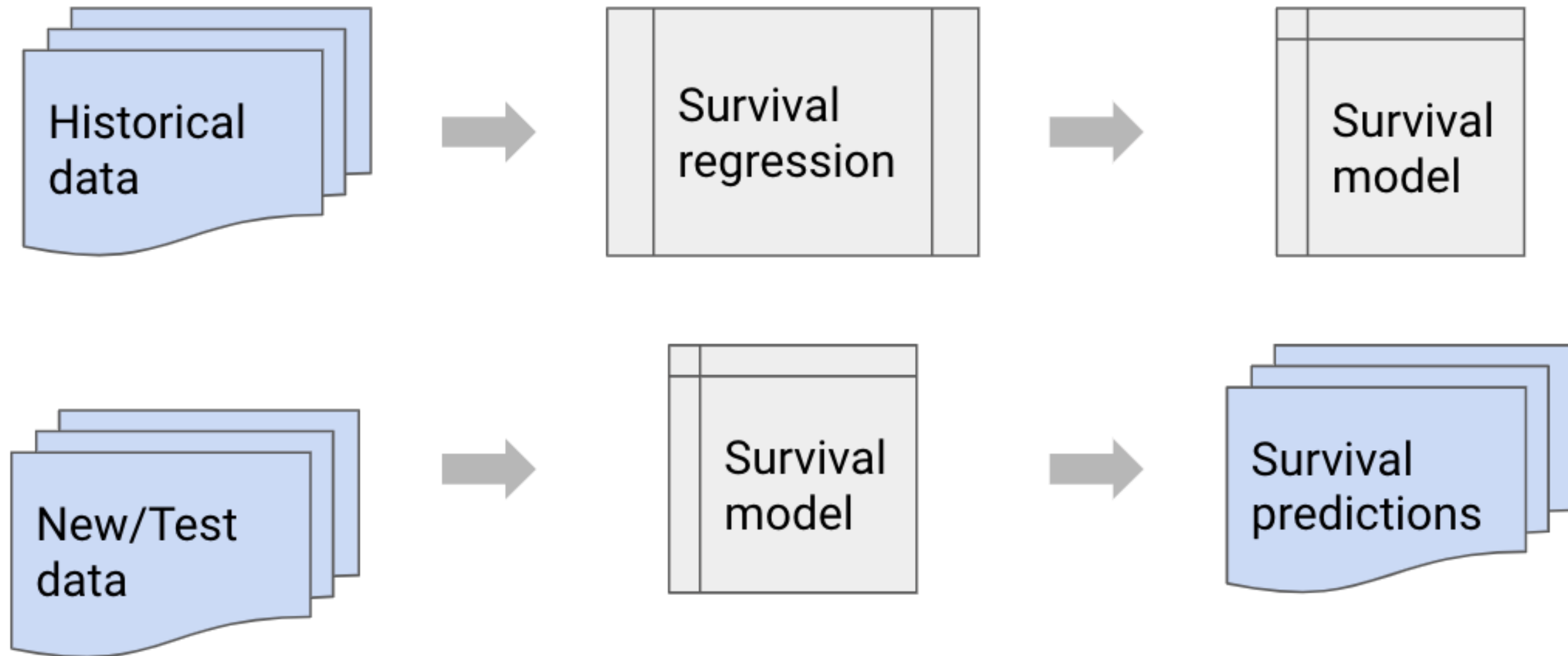
# Mortgage example

```
aft.plot_partial_effects_on_outcome(  
    covariates='credit score',  
    values=np.arange(700, 860, 30)  
)  
plt.show()
```



# Predict survival functions

- Survival curves vary based on their covariates' values.



# Predict survival functions

Predict survival functions of individuals based on covariate values.

```
.predict_survival_function()
```

Arguments:

- `X` (`np` array or `DataFrame`): covariates. If a `DataFrame`, columns can be in any order.

Predict median survival durations of individuals based on covariate values.

```
.predict_median()
```

Arguments:

- `df` (`np` array or `DataFrame`): covariates. If a `DataFrame`, columns can be in any order.

# Conditional after current durations

Predict survival function or median survival duration conditional after current duration.

- `.predict_survival_function(X, conditional_after)`
- `.predict_median(df, conditional_after)`

Example:

```
aft.predict_median(new_subject)
```

```
4.0
```

```
aft.predict_median(new_subject, conditional_after=[2])
```

```
2.0
```

**Let's practice!**  
SURVIVAL ANALYSIS IN PYTHON

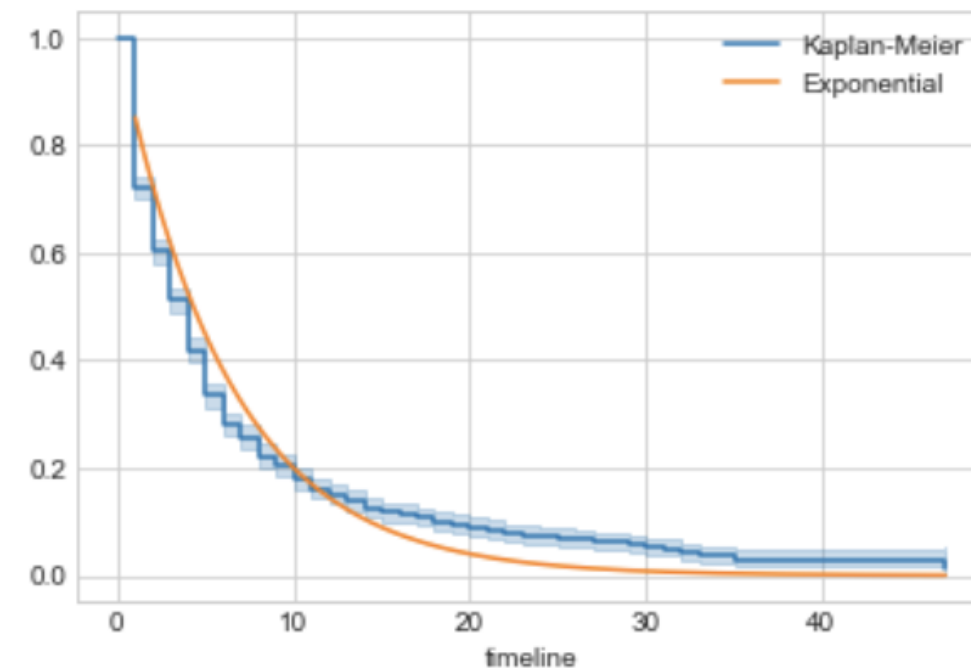
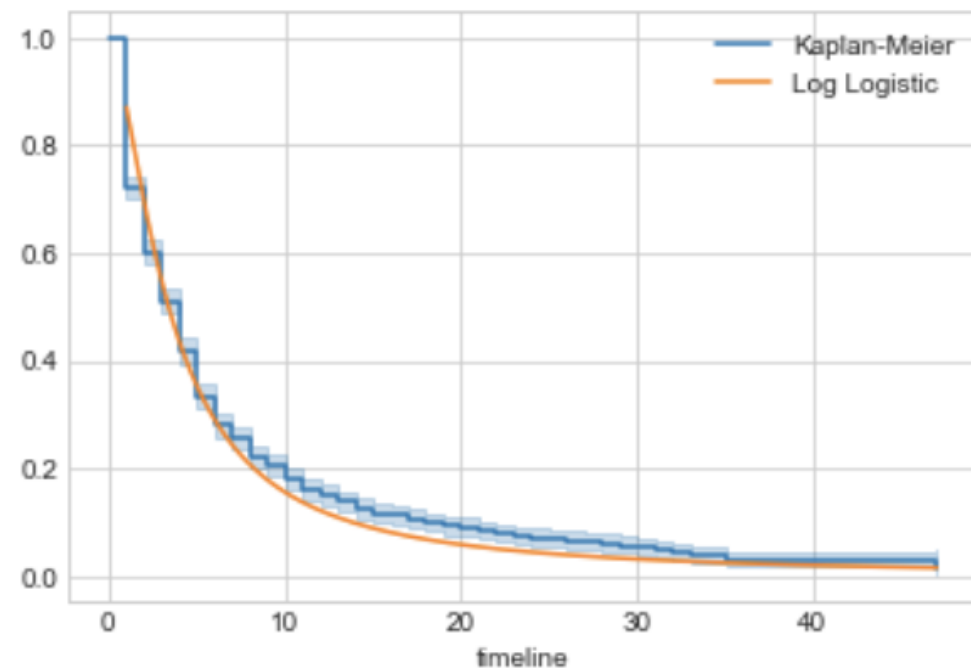
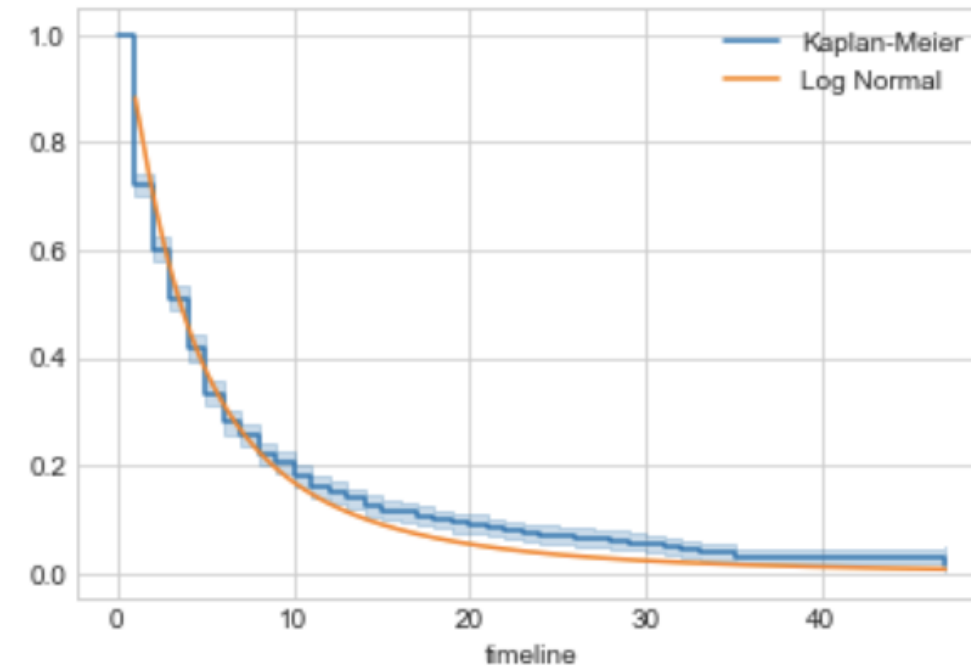
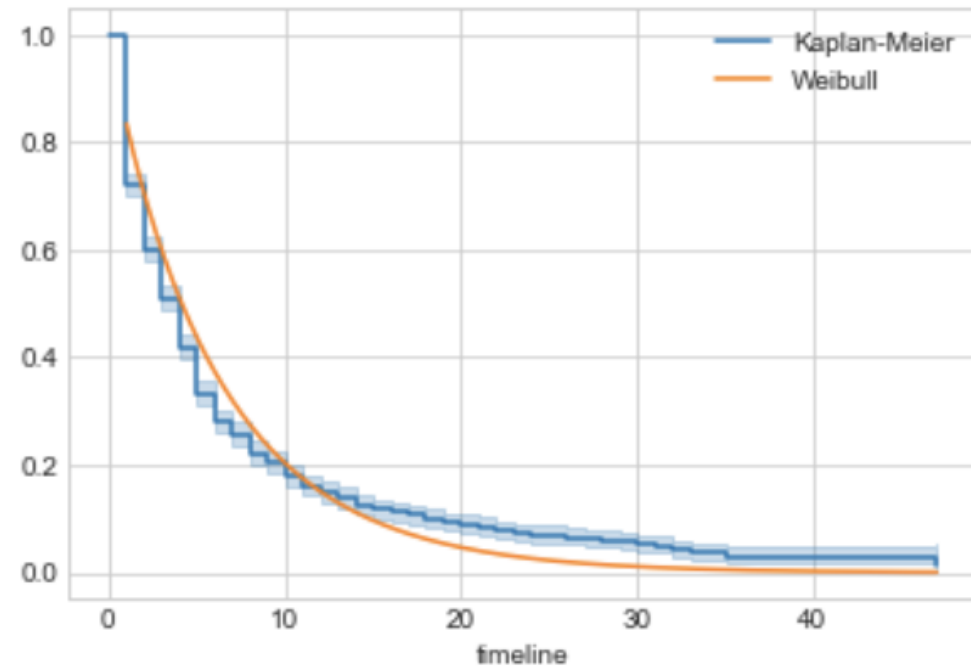
# Other distributions and model selection

SURVIVAL ANALYSIS IN PYTHON



**Shae Wang**  
Senior Data Scientist

# Which model fits the data the best?



# Choosing parametric models

- **Non-parametric modeling** (i.e. the Kaplan-Meier model)
  - Describes the data accurately because it's distribution-free
  - Is not smooth/continuous/differentiable
- **Parametric modeling** (i.e. the Weibull model)
  - Parametric statistics will usually give us more information
  - When the wrong model is used, they lead to significantly biased conclusions



# Common parametric survival models

- The Weibull model

```
from lifelines import WeibullFitter
```

- The Exponential model

```
from lifelines import ExponentialFitter
```

- The Gamma model

```
from lifelines import GeneralizedGammaFitter
```

- The Log Normal model

```
from lifelines import LogNormalFitter
```

- The Log Logistic model

```
from lifelines import LogLogisticFitter
```

# The Akaike Information Criterion (AIC)

- AIC: An estimator of **prediction error** and **relative quality of statistical models** for a given set of data.
- Estimates the relative amount of information lost by a given model and penalizes large number of estimated parameters.
  - The less information a model loses, the higher the quality of that model.
  - The fewer parameters (less complex) a model is, the higher the quality of that model.
- Given a set of candidate models for the data, the one with **the minimum AIC value** is the preferred model.

# Using the AIC for model selection

Step 1) Fit parametric models in `lifelines`

Step 2) Print and compare each model's `AIC_` property

Step 3) The lowest AIC value is preferred

```
from lifelines import WeibullFitter,  
                      ExponentialFitter,  
                      LogNormalFitter
```

```
wb = WeibullFitter().fit(D, E)  
exp = ExponentialFitter().fit(D, E)  
log = LogNormalFitter().fit(D, E)
```

```
print(wb.AIC_, exp.AIC_, log.AIC_)
```

```
215.9091    216.1183    202.3498
```

# find\_best\_parametric\_model()

- `find_best_parametric_model()` : a built-in `lifelines` function to automate AIC comparisons between parametric models.
- Iterates through each parametric model available in `lifelines`.

## How to use it?

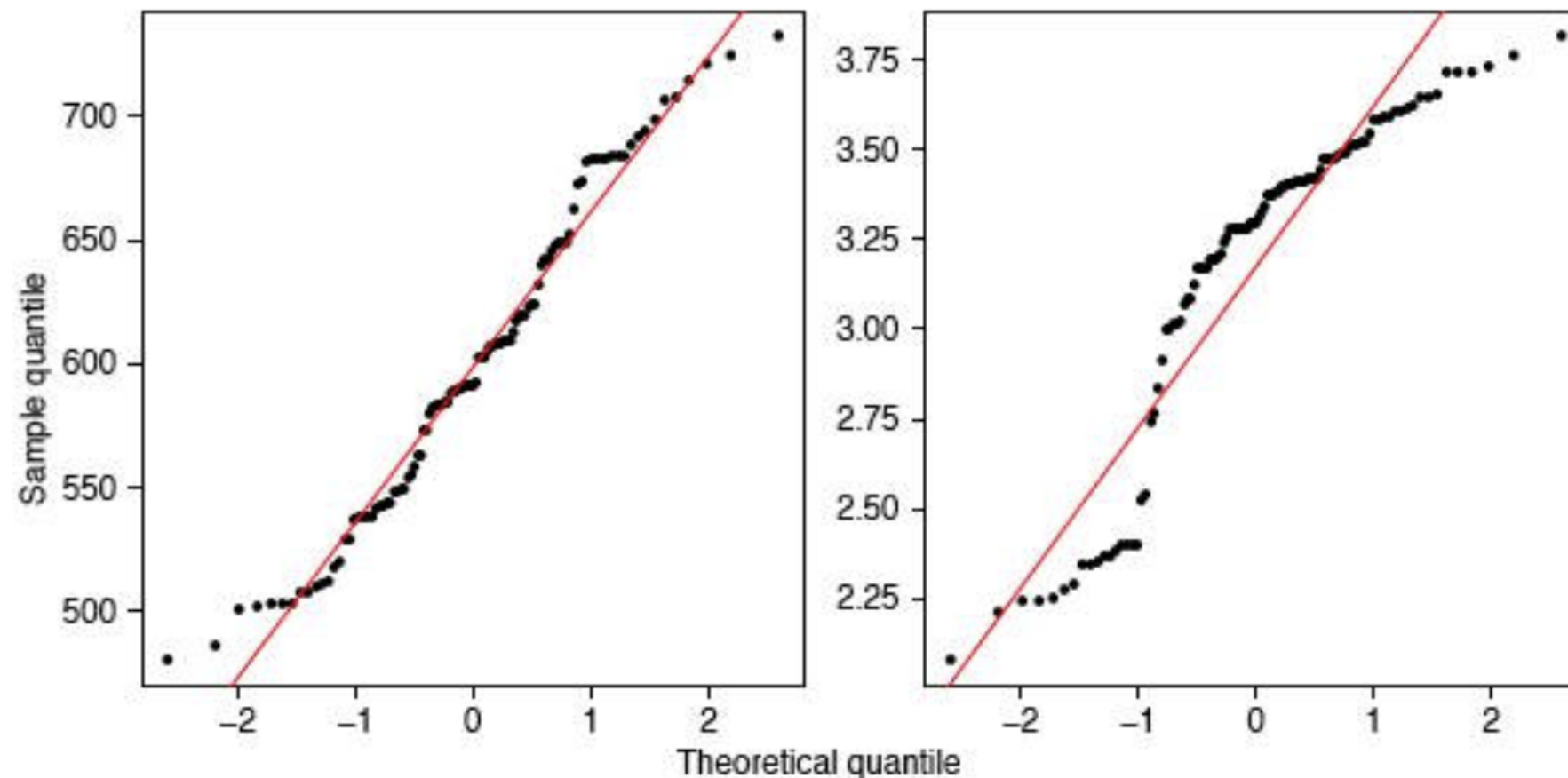
- `T` : durations, `E` : censorship

```
best_model, best_aic_ = find_best_parametric_model(event_times=T,  
                                                    event_observed=E,  
                                                    scoring_method="AIC")  
  
print(best_model)
```

```
<lifelines.WeibullFitter:"Weibull_estimate",  
fitted with 686 total observations, 387 right-censored observations>
```

# The QQ plot

- QQ plot: Compares two probability distributions by plotting their quantiles against each other.
- If the two distributions being compared are similar, the points in the QQ plot will approximately lie on the line  $y = x$ .



# Using QQ plots for model selection

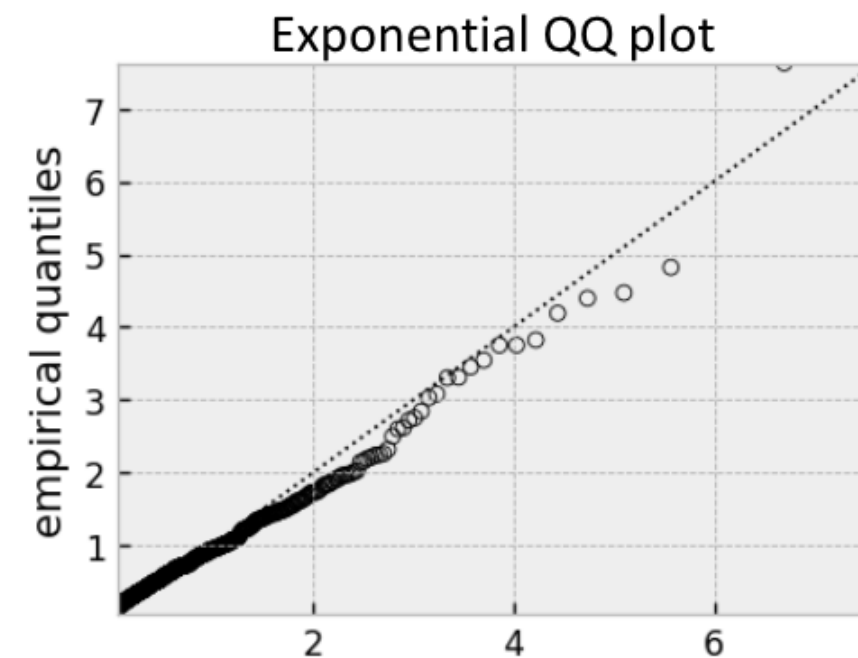
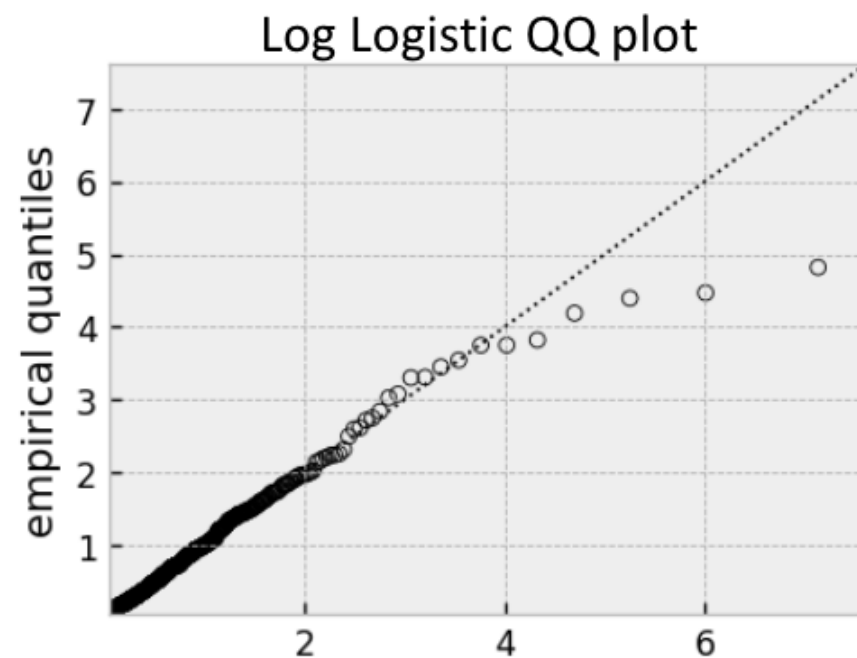
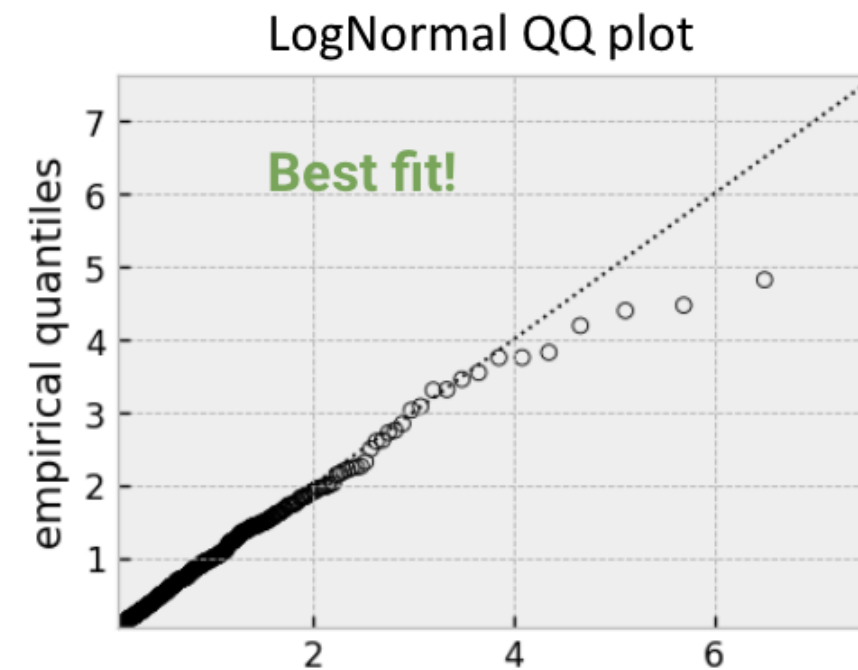
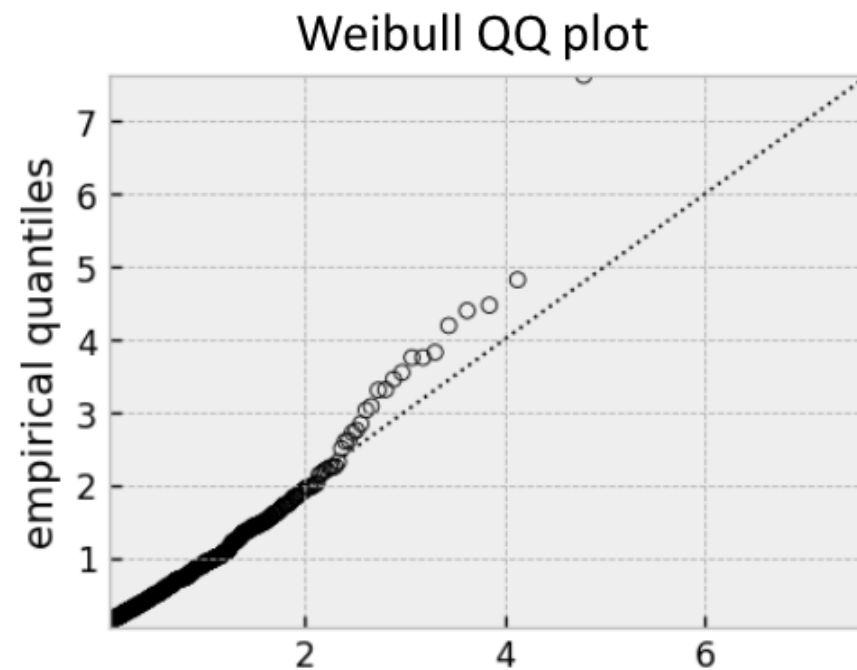
Step 1) Fit parametric models in `lifelines` .

Step 2) Plot the QQ plot of each model.

Step 3) The QQ plot closest to  $y = x$  is preferred.

```
from lifelines.plotting import qq_plot
for model in [WeibullFitter(),
               LogNormalFitter(),
               LogLogisticFitter(),
               ExponentialFitter()]:
    model.fit(T, E)
    qq_plot(model)
plt.show()
```

# Using QQ plots for model selection



**Let's practice!**  
SURVIVAL ANALYSIS IN PYTHON