Journal of Probability and Statistical Science
24(2), 1-15Aug. 2026

fitPlotpy: A Python Package for Plotting Probability Distribution

Muhammad Osama
Department of Statistics
Quaid-i-Azam University

ABSTRACT

The **fitPlotpy** package offers a comprehensive collection of functions for visualizing univariate probability distributions. It provides an integrated framework for plotting probability density functions, cumulative distribution functions, survival functions, and hazard functions. In addition, the package includes tools to compute and display fundamental descriptive measures of a distribution, such as the mean, median, mode, and measures of dispersion, as well as higher-order moments including skewness and kurtosis. Beyond descriptive and theoretical plots, **fitPlotpy** supports diagnostic visualization through fitted probability density and cumulative distribution plots, quantile–quantile (QQ) plots, and probability–probability (PP) plots. These features are complemented by comprehensive graphical summaries, enabling an effective and detailed overview of distributional characteristics and model fit.

Keyword: Python package; Univariate distributions; Probability density function; Cumulative distribution function; Survival function; Hazard function; Statistical moments; Distribution fitting; Data visualization

1. Introduction

Visualization of probability distributions is the foundational task in statistical analysis. Probability distributions plots enable the researchers to understand the behavior of the random variables, assess the model fits, compare theoretical distributions to empirical data and to link the results efficiently. In recent years, the Python scientific computing ecosystem has matured significantly, offering a rich collection of packages that support statistical modeling, numerical computation, and graphical representation of data.

Packages such as NumPy [1] and SciPy [2] provides fundamental functionality for working with arrays numerical methods and an extensive range of probability distributions. Tools like pandas [3] enable for manipulating and preprocessing of structured data. while statsmodels [4] and PyMC [5] extend the Python for statistical modeling abilities to include generalized linear models and Bayesian inference, respectively.

Python have many packages for visualization. Matplotlib [6] is the most widely-used plotting library and is used to create high-quality 2D plots from datasets. Many data visualization libraries have been developed on top of Matplotlib. Another example of a data visualization library developed using Matplotlib is Seaborn [7], which is designed to create statistics-based graphical displays of data. As well, Plotly [8] and Bokeh [9] are two excellent options for creating data visualizations, including Plotly's ability to create visualizations of data distributions. Some examples of data visualization libraries also include Altair [10] and ggplot [11]; these provide a grammar that assists in creating graphs. Each of these libraries offers Python users additional support when creating data visualizations with simple syntax. Despite this vibrant ecosystem, existing tools often require non-trivial boilerplate to create consistent and customizable plots of probability distributions. For instance, suppose you want to generate kernel density estimate using

- Received February 21, 2026, in final form August 2026.
- Muhammad Osama (corresponding author) is affiliated with Department of Statistics, Quaid-i-Azam University, Pakistan
  Email: muhammadosama0846@gmail.com

the SciPy Stats module, or generate a distribution function using SymPy [12]. In either case, you will have to write a fair amount of duplicated code if you want to create multiple examples using multiple set of parameters.

To address the shortcomings of current general-purpose visualization tools, we present fitPlotpy a Python library that creates an organized and easy to use way to create visualizations of theoretical probability distributions. This library specializes in creating clear, consistent and high-quality graphics for probability distribution i-e (Probability density, cumulative distribution, survival and hazard functions). By permitting the users to specify model parameters directly, fitPlotpy generate custom plots for several types of continuous probability distribution. By providing all tools in one package the user does not require to visualize probability distributions through complex programming. As a result, students, teachers, and researchers can focus on studying distribution behavior, creating parameter-based visualizations, and ensuring reproducibility without getting distracted by complex implementation details.

This paper presents the review of the principal features of the **fitPlotpy** package. The package is available on PyPi https://pypi.org/project/fitPlotpy/0.1.0/. The rest of the paper is organized as: section 2 presents plots for different probability distributions, section 3 discusses the computation of distributional moments, section 4 presents fitted plots for different probability distributions, section 5 presents multiple fitted plots for different probability distributions, section 6 provides data overview and section 7 conclude the paper.

To use the package first we install and then load the package.

```
Pip install fitPlotpy
import fitPlotpy
```

2. Plotting Probability Distribution

2.1 Probability density function (PDF)

The PDF quantifies the relative concentration of probability at each possible value of a continuous random variable. Plotting PDF is an essential because to understand the shape and tail behavior. The PDFs in **fitPlotpy** package are plot using **plot_pdf** function. The function required PDF of a probability distribution and a list of allied parameters, while additional arguments in the **plot_pdf** function can be used to customized the subsequent plot. To demonstrate the use of **plot_pdf** function consider the PDF of generalized exponential distribution, exponentiated Weibull distribution and Kumaraswamy distribution.

The PDF of generalized exponential distribution is given as

$$f(x,\alpha,\lambda) = \alpha\lambda exp(-\lambda x)\big(1-exp(-\lambda x)\big)^{\alpha-1} \quad x \geq 0,\ \alpha > 0, \beta > 0$$

- **Python Code for Plotting PDF of generalized exponential distribution.**

```
from fitPlotpy import plot_pdf
import numpy as np
# PDF
def ge_pdf(x, alpha, lambda_):
   return alpha * lambda_ * np.exp(-lambda_ * x) * (1 - np.exp(-lambda_ * x))**(alpha - 1)

# Parameter sets
param_values = [
   {"alpha": 1, "lambda_": 1},
   {"alpha": 2, "lambda_": 1},
   {"alpha": 3, "lambda_": 0.5},
```

```
    {"alpha": 4, "lambda_": 1.5},
    {"alpha": 5, "lambda_": 2.5},
]

# Plot
plot_pdf(
    pdf=ge_pdf,
    param_list=param_values,
    ylim=(0, 1.5),
    xlim=(0, 6),
    main="PDF GE distribution"
)
```

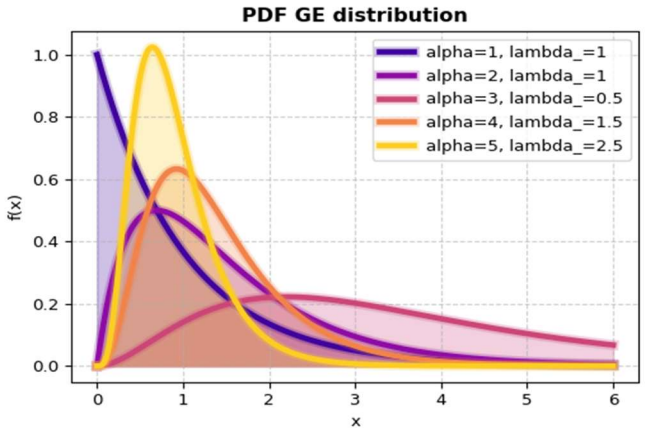


Figure 1: Plot for the PDF of generalized exponential distribution

The PDF exponentiated Weibull distribution is given as

$$f(x,a,b,c) = \alpha\,\beta\,\gamma(\beta x)^{\gamma-1}\,exp(-(\beta x)^{\gamma})\left(1-exp(-(\beta x)^{\gamma})\right)^{\alpha-1}; x \geq 0,\ \alpha > 0, \beta > 0, \gamma > 0$$

- **Python code for plotting the PDF of exponentiated Weibull distribution.**

```
# PDF

def exp_weibull_pdf(x, alpha, beta, gamma):
     t = (beta * x) ** gamma
     pdf = alpha * gamma * beta**gamma * x**(gamma - 1) * (1 - np.exp(-t))
**(alpha - 1) * np.exp(-t)
     return pdf

# Parameter sets
param_values = [
    {"alpha": 1.0, "beta": 1.0, "gamma": 1.0},
    {"alpha": 0.8, "beta": 2.0, "gamma": 1.5},
    {"alpha": 2.0, "beta": 0.8, "gamma": 2.5},
    {"alpha": 3.0, "beta": 1.5, "gamma": 0.5},
    {"alpha": 4.0, "beta": 2.5, "gamma": 1.2},
    {"alpha": 0.5, "beta": 0.7, "gamma": 3.0}
]
```

```
# Plot
plot_pdf(exp_weibull_pdf, xlim=(0, 4), ylim=(0, 2),
        main="PDF EW distribution", param_list=param_values,
        grid=False)
```

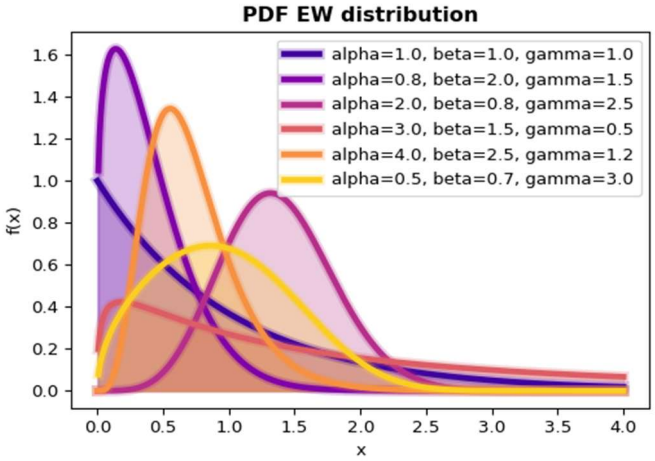


Figure 2: Plot for the PDF of Exponentiated Weibull distribution

The PDF of Kumaraswamy distribution is given as

$$f(x) = abx^{a-1}\,(1-x^a)^{b-1} \quad 0 < x < 1, a > 0, b > 0$$

- **Python code for plotting the PDF of Kumaraswamy distribution.**

```
# PDF
def kumar_pdf(x, a, b):
    pdf = a * b * x**(a-1) * (1 - x**a)**(b-1)
    return pdf

# CDF
param_values = [
    {"a": 4.5, "b": 1.2},
    {"a": 1.0, "b": 0.8},
    {"a": 1.5, "b": 1.6},
    {"a": 2.0, "b": 1.9},
    {"a": 3.0, "b": 2.6}
]

# Colors
colors = ["skyblue", "lightgreen", "violet", "darkgray", "salmon"]

# Plot
plot_pdf(pdf=kumar_pdf, param_list=param_values, xlim=(0, 1),
         ylim=(0, 5), lty="-", shade_colors=colors, lwd = 4,
         main = "PDF KW distribution")
```

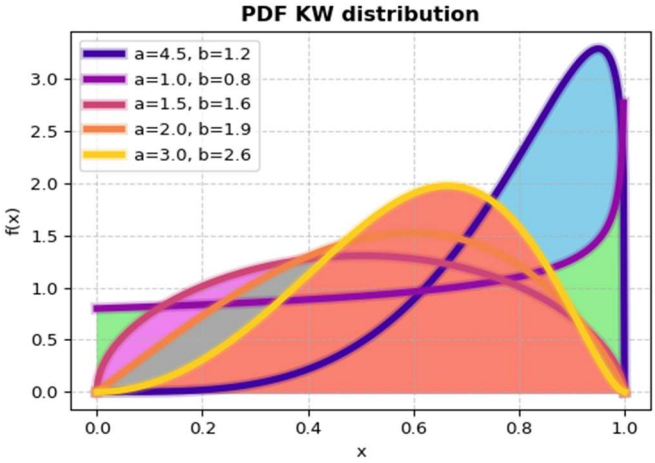


Figure 3: Plot for the PDF of Kumaraswamy distribution

2.2 Cumulative Distribution Function (CDF)

The CDF of a probability distribution is defined as

$$F(x) = P(X \leq x)$$

The CDFs in **fitPlotpy** package are plot using **plot_cdf** function. The function required CDF of a probability distribution and a list of allied parameters, while additional arguments in the **plot_cdf** function can be used to customized the subsequent plot. To demonstrate the use of **plot_cdf** function consider the CDF of generalized exponential distribution, exponentiated Weibull distribution and Kumaraswamy distribution.

The CDF of generalized exponential distribution is given as

$$F(x, \alpha, \lambda) = \left(1 - exp(-\lambda x)\right)^{\alpha} \quad \alpha > 0, \lambda > 0$$

- **Python Code for Plotting CDF of generalized exponential distribution.**

```
from fitPlotpy import plot_cdf
import numpy as np
# CDF
def ge_cdf(x, alpha, lambda_):
    return (1 - np.exp(-lambda_ * x))**alpha

# Parameter set
param_values = [
    {"alpha": 1, "lambda_": 1},
    {"alpha": 2, "lambda_": 1},
    {"alpha": 3, "lambda_": 0.5},
    {"alpha": 4, "lambda_": 1.5},
    {"alpha": 5, "lambda_": 2.5},
]

# Plot
plot_cdf(cdf=ge_cdf, param_list=param_values, main = "CDF GE distribution")
```

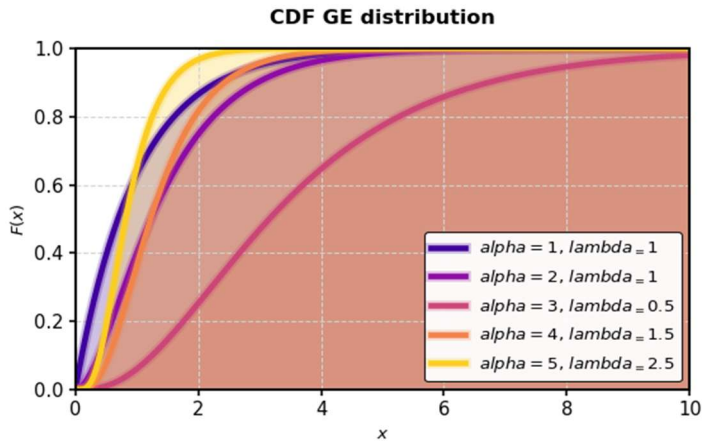


Figure 4: Plot for the CDF of generalized exponential distribution

The CDF of exponentiated Weibull distribution is given as

$$F(x,a,b,c) = \left(1 - exp(-(\beta x)^{\gamma})\right)^{\alpha} \quad \alpha > 0, \beta > 0, \gamma > 0$$

- **Python code for plotting the CDF of exponentiated Weibull distribution.**

```
# CDF
def exp_weibull_cdf(x, alpha, beta, gamma):
    t = (beta * x)**gamma
    return (1 - np.exp(-t))**alpha
# Parameter set
param_values = [
    {"alpha": 1.0, "beta": 1.0, "gamma": 1.0},
    {"alpha": 0.8, "beta": 2.0, "gamma": 1.5},
    {"alpha": 2.0, "beta": 0.8, "gamma": 2.5},
    {"alpha": 3.0, "beta": 1.5, "gamma": 0.5},
    {"alpha": 4.0, "beta": 2.5, "gamma": 1.2},
    {"alpha": 0.5, "beta": 0.7, "gamma": 3.0}
]
# Plot
plot_cdf(exp_weibull_cdf, xlim=(0, 4), main="CDF EW distribution", param_list
=param_values, grid=False)
```

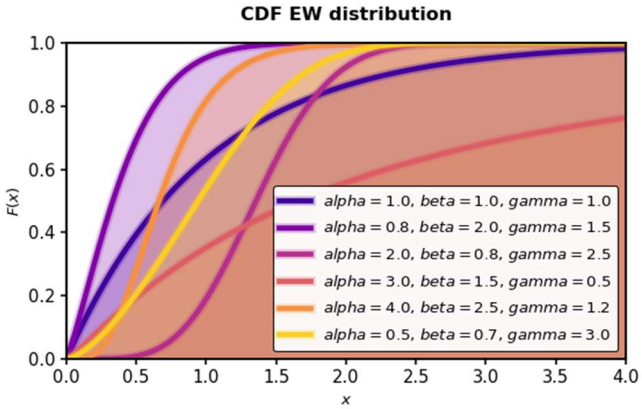


Figure 5: Plot for the CDF of Exponentiated Weibull distribution

The CDF of Kumaraswamy distribution is given as

$$F(x) = 1 - (1 - x^a)^{b-1} \quad a, b > 0$$

- **Python code for the CDF of Kumaraswamy distribution.**

```
# CDF
def kumar_cdf(x, a, b):
    return 1-(1-x**a)**b

# Parameter Set
param_values = [
    {"a": 0.5, "b": 1.2},
    {"a": 1.0, "b": 0.8},
    {"a": 1.5, "b": 1.6},
    {"a": 2.0, "b": 1.9},
    {"a": 3.0, "b": 2.6}
]
# Colors
colors = ["skyblue", "lightgreen", "violet", "darkgray", "salmon"]

# Plot
plot_cdf(cdf=kumar_cdf, param_list=param_values,
         xlim=(0, 1), lty="-", shade_colors=colors, lwd = 4, grid=False,
         main = "CDF KW distribution")
```

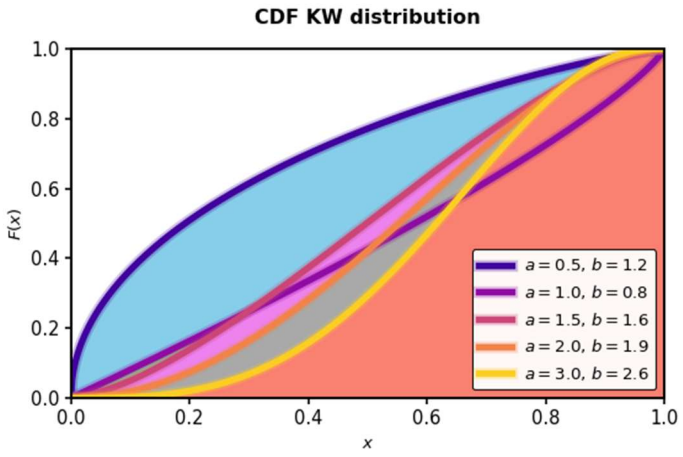


Figure 6: Plot for the CDF of Kumaraswamy distribution

2.3 Survival Function (SF)

The survival function of a probability distribution is defined as;

$$S(x) = 1 - P(X \leq x)$$

The SFs in **fitPlotpy** package are plot using **plot_sf** function. The function required SF of a probability distribution and a of allied parameters, while additional arguments in the **plot_sf** function can be used to customized the subsequent plot. To demonstrate the use of **plot_sf** function consider the SF of generalized exponential distribution, exponentiated Weibull distribution and Kumaraswamy distribution.

The SF of generalized exponential distribution is given as

$$S(x, \alpha, \lambda) = 1 - \left(1 - exp(-\lambda x)\right)^{\alpha} \quad \alpha > 0, \lambda > 0$$

- **Python code for plotting the SF of generalized exponential distribution**

```
from fitPlotpy import plot_sf
import numpy as np
# SF
def ge_sf(x, alpha, lambda_):
    return (1 - np.exp(-lambda_ * x))**alpha
# Parameter Set
param_values = [
    {"alpha": 1, "lambda_": 1},
    {"alpha": 2, "lambda_": 1},
    {"alpha": 3, "lambda_": 0.5},
    {"alpha": 4, "lambda_": 1.5},
    {"alpha": 5, "lambda_": 2.5},
]
# Plot
plot_sf(sf=ge_sf, param_list=param_values, main = "SF GE distribution")
```

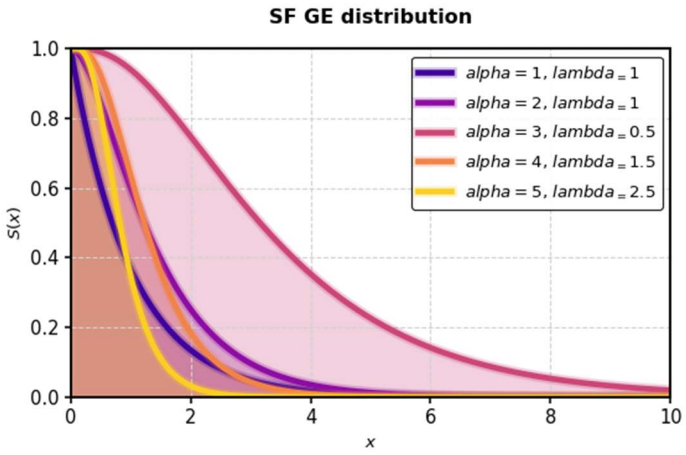


Figure 7: SF plot for the generalized exponential distribution

The SF of exponentiated Weibull distribution is given as

$$S(x,a,b,c) = 1 - \left(1 - exp(-(bx)^c)\right)^a \quad a > 0, b > 0, c > 0$$

- **Python code for plotting the SF of exponentiated Weibull distribution.**

```
# SF
def exp_weibull_sf(x, alpha, beta, gamma):
    t = (beta * x)**gamma
    return (1 - np.exp(-t))**alpha
# Parameter Set
param_values = [
    {"alpha": 1.0, "beta": 1.0, "gamma": 1.0},
    {"alpha": 0.8, "beta": 2.0, "gamma": 1.5},
    {"alpha": 2.0, "beta": 0.8, "gamma": 2.5},
    {"alpha": 3.0, "beta": 1.5, "gamma": 0.5},
    {"alpha": 4.0, "beta": 2.5, "gamma": 1.2},
    {"alpha": 0.5, "beta": 0.7, "gamma": 3.0}
]
```

```
# Plot
plot_sf(
    sf=exp_weibull_sf,
    xlim=(0, 4),
    main="SF EW distribution",
    param_list=param_values,
    grid=False
)
```

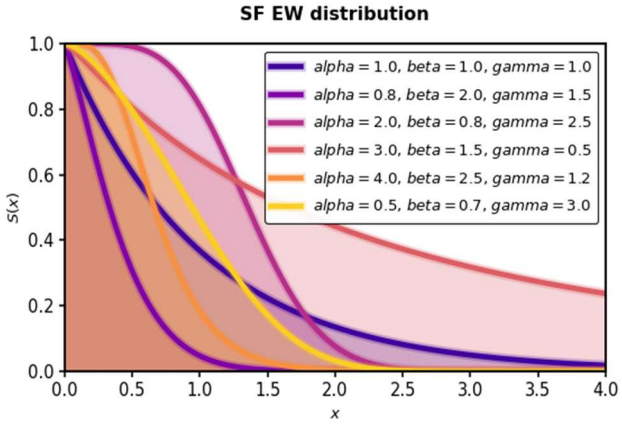


Figure 8: SF plot for the exponentiated Weibull distribution

The SF of Kumaraswamy distribution is given as

$$S(x) = (1 - x^a)^{b-1}; \quad 0 < x < 1, a, b > 0$$

- **Python code for plotting the SF of Kumaraswamy distribution.**

```
# SF
def kumar_sf(x, a, b):
    return 1 - (1 - x**a)**b
# Parameter Set
param_values = [
    {"a": 0.5, "b": 1.2},
    {"a": 1.0, "b": 0.8},
    {"a": 1.5, "b": 1.6},
    {"a": 2.0, "b": 1.9},
    {"a": 3.0, "b": 2.6}
]
# Colors
colors = ["skyblue", "lightgreen", "violet", "darkgray", "salmon"]
# Plot
plot_sf(
    sf=kumar_sf,
    param_list=param_values,
    xlim=(0, 1),
    lty="-",
    shade_colors=colors,
    lwd=4,
    grid=False,
    main = "SF KW distribution"
)
```

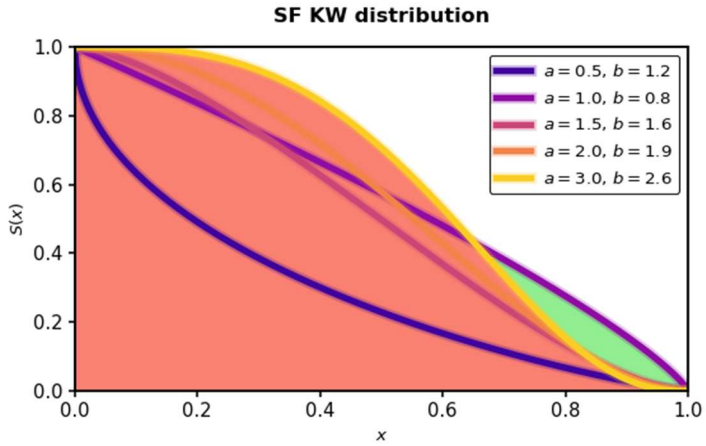


Figure 9: SF plot for the Kumaraswamy distribution

2.4 Hazard Function

If $f(x)$ is the PDF and $S(x)$ is the SF, then the hazard function is defined as

$$h(x) = \frac{f(x)}{S(x)}$$

The HFs in **fitPlotpy** package are plot using **plot_hazard** function. The function required HF of a probability distribution and a list of allied parameters, while additional arguments in the **plot_hazard** function can be used to customized the subsequent plot. To demonstrate the use of **plot_hazard** function consider the HF of generalized exponential distribution, exponentiated Weibull distribution and Kumaraswamy distribution.

The HF of generalized exponential distribution is given as

$$H(x) = \frac{\alpha\lambda exp(-\lambda x)\big(1 - exp(-\lambda x)\big)^{\alpha-1}}{1 - \big(1 - exp(-\lambda x)\big)^{\alpha}}$$

- **Python code for plotting the HF of generalized exponential distribution.**

```
from fitPlotpy import plot_hazard
import numpy as np
# PDF
def ge_pdf(x, alpha, lambda_):
    return alpha * lambda_ * np.exp(-lambda_ * x) * (1 - np.exp(-lambda_ * x)
)**(alpha - 1)

# CDF
def ge_cdf(x, alpha, lambda_):
    return (1 - np.exp(-lambda_ * x))**alpha

# Parameter Set
param_values = [
    {"alpha": 1, "lambda_": 1},
    {"alpha": 2, "lambda_": 1},
    {"alpha": 3, "lambda_": 0.5},
    {"alpha": 4, "lambda_": 1.5},
    {"alpha": 5, "lambda_": 2.5},
]
```

```
# Plot
plot_hazard(
    pdf=ge_pdf,
    cdf=ge_cdf,
    param_list=param_values,
    main = "HF GE distribution"
)
```

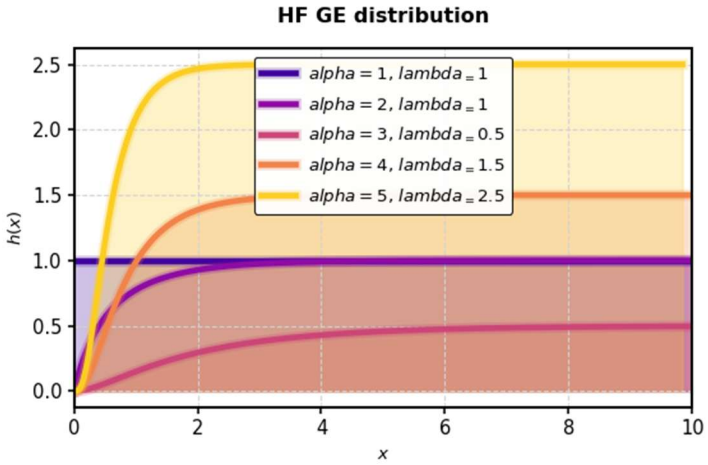


Figure 10: Plot for HF of the generalized exponential distribution

The HF of the exponentiated Weibull distribution is given as

$$H(x) = \frac{\alpha\,\beta\,\gamma(\beta x)^{\gamma-1}\,exp(-(\beta x)^{\gamma})\left(1-exp(-(\beta x)^{\gamma})\right)^{\alpha-1}}{1-\left(1-exp(-(bx)^{c})\right)^{a}}$$

- **Python code for plotting the HF of exponentiated Weibull distribution.**

```
# PDF
def exp_weibull_pdf(x, alpha, beta, gamma):
    t = (beta * x)**gamma
    return (
        alpha * gamma * beta**gamma * x**(gamma - 1)
        * np.exp(-t)
        * (1 - np.exp(-t))**(alpha - 1)
    )

# CDF
def exp_weibull_cdf(x, alpha, beta, gamma):
    t = (beta * x)**gamma
    return (1 - np.exp(-t))**alpha

# Parameter Set
param_values = [
    {"alpha": 1.0, "beta": 1.0, "gamma": 1.0},
    {"alpha": 0.8, "beta": 2.0, "gamma": 1.5},
    {"alpha": 2.0, "beta": 0.8, "gamma": 2.5},
    {"alpha": 3.0, "beta": 1.5, "gamma": 0.5},
]
```

```
# Plot
plot_hazard(
    pdf=exp_weibull_pdf,
    cdf=exp_weibull_cdf,
    xlim=(0, 4),
    main="HF EW distribution",
    param_list=param_values,
    grid=False
)
```

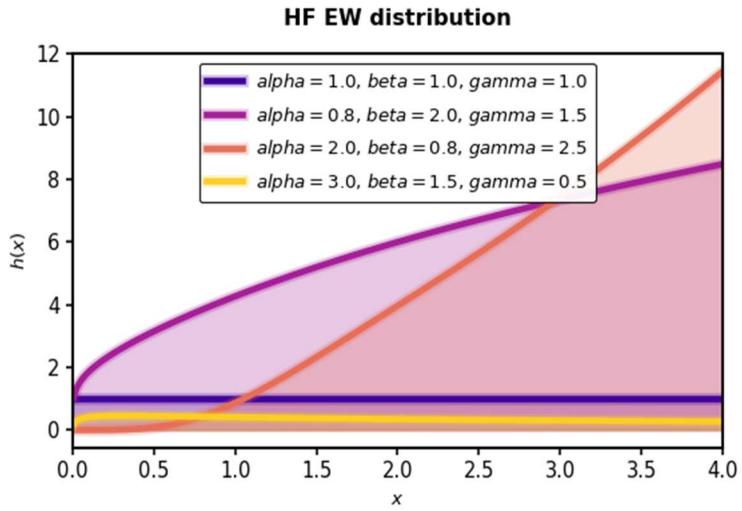


Figure 11: Plot for HF of the exponentiated Weibull distribution

The HF of the Kumaraswamy distribution is given as

$$H(x) = \frac{abx^{a-1}\,(1-x^a)^{b-1}}{(1-x^a)^{b-1}}$$

- **Python code for plotting the HF of Kumaraswamy distribution.**

```
# PDF
def kumar_pdf(x, a, b):
    return a * b * x**(a - 1) * (1 - x**a)**(b - 1)

# CDF
def kumar_cdf(x, a, b):
    return 1 - (1 - x**a)**b

# Parameter Set
param_values = [
    {"a": 0.5, "b": 1.2},
    {"a": 1.0, "b": 0.8},
    {"a": 1.5, "b": 1.6},
    {"a": 2.0, "b": 1.9},
]
```

```
# Plot
plot_hazard(
    pdf=kumar_pdf,
    cdf=kumar_cdf,
    xlim=(0, 1),
    ylim=(0, 10),
    param_list=param_values,
    grid=False
)
```

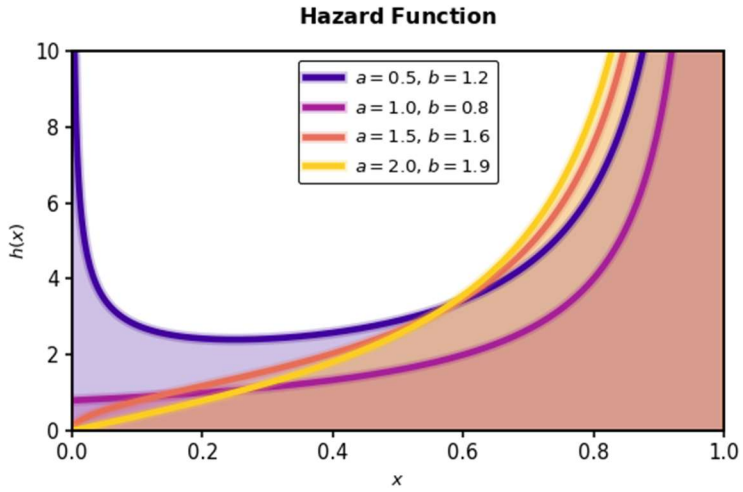


Figure 12: Plot for HF of the Kumaraswamy distribution

3. Moments of Probability distribution

Probability distributions can be described through their key numerical characteristics, known as moments. The first moment, or mean, indicates the central location of the distribution. The second central moment, called the variance, measures the spread by quantifying the average squared distance of observations from the mean. Higher-order moments, such as skewness and kurtosis, provide additional insight into the shape of the distribution skewness reflects the degree of asymmetry, while kurtosis measures the heaviness of the tails or the sharpness of the peak. In addition to these moments, other measures of central tendency, including the median and mode, are also important for understanding distributional behavior. All these quantities can be derived once the probability density function (PDF) and cumulative distribution function (CDF) of the distribution are specified. The **fitPlotpy** package provides the **dist_moments** function to compute these moments directly. To demonstrate the use of **dist_moments** function, consider the generalized exponential distribution and exponentiated Weibull distribution.

- **Python code for finding moments of the generalized exponential distribution.**

```
from fitPlotpy import dist_moments
import numpy as np
# PDF
def ge_pdf(x, a, b):
    return a * b * np.exp(-b*x) * (1 - np.exp(-b*x))**(a-1)
# CDF
def ge_cdf(x, a, b):
    return (1 - np.exp(-b*x))**a
```

```
# Parameters
params = (1.5, 2.6)
moments = dist_moments(pdf=ge_pdf, cdf=ge_cdf, support=(0, np.inf), params=pa
rams)
print(moments)
##        Mean  Variance  Skewness  Kurtosis    Median      Mode
## 0  0.492451  0.170795   1.74746  7.703537  0.382364  0.153589
```

- **Python code for finding moments of the exponentiated Weibull distribution.**

```
# PDF
def exp_weibull_pdf(x, alpha, beta, gamma):
    if x <= 0:
        return 0.0
    t = (beta * x)**gamma
    return alpha * gamma * beta**gamma * x**(gamma - 1) * (1 - np.exp(-t))**(
alpha - 1) * np.exp(-t)

# CDF
def exp_weibull_cdf(x, alpha, beta, gamma):
    if x <= 0:
        return 0.0
    t = (beta * x)**gamma
    return (1 - np.exp(-t))**alpha

# Parameters
params = (2.3, 1.5, 1.8)
moments = dist_moments(exp_weibull_pdf,
                       exp_weibull_cdf,
                       support=(0, np.inf),
                       params=params)
print(moments)
##        Mean  Variance  Skewness  Kurtosis    Median      Mode
## 0  0.819994  0.103992  0.611833  3.459568  0.786427  0.714524
```

3.1 Plotting Fitted Distribution with Diagnostic Plots

To plot fitted distributions with diagnostic plots such as fitted PDF, fitted CDF, against the empirical CDF, quantile-quantile (QQ) plot and probability-probability (PP) plot. The **fitPlotpy** package provide the function **plot_fitted** which automatically generates all these plots by defining the PDF, CDF and quantile function of a distribution. To illustrate the use of **plot_fitted** function we consider Weibull distribution and gamma distribution. First, we plot for Weibull distribution. The PDF of Weibull distribution is given by

$$f(x,a,b) = abx^{a-1}exp(-bx^a) \quad x \geq 0,\ a > 0, b > 0$$

the CDF of Weibull distribution is given as

$$F(x,a,b) = 1 - exp(-bx^a)$$

and the quantile function respectively

$$Q(p) = \left(-\frac{1}{b}\ln(1-p)\right)^{1/a}, \quad 0 < p < 1.$$

- **Python code for fitted plots of the Weibull distribution.**

```
from fitPlotpy import plot_fitted
import numpy as np
from scipy.stats import weibull_min
# Weibull PDF
def weibull_pdf(x, a, b):
    return weibull_min.pdf(x, c=a, scale=b)
# Weibull CDF
def weibull_cdf(x, a, b):
    return weibull_min.cdf(x, c=a, scale=b)
# Weibull Quantile Function (Inverse CDF)
def weibull_quantile(p, a, b):
    return weibull_min.ppf(p, c=a, scale=b)
# Generate sample data
np.random.seed(2)
shape = 2.0  # estimated shape parameter 'a'
scale = 1.0  # estimated scale parameter 'b'
data = np.random.weibull(a=shape, size=60) * scale

#  Estmiated Parameters
params = {"a": shape, "b": scale}
# Plot
plot_fitted(pdf=weibull_pdf,
            cdf=weibull_cdf,
            ppf=weibull_quantile,
            params=params,
            bins=6,
            data=data)
```

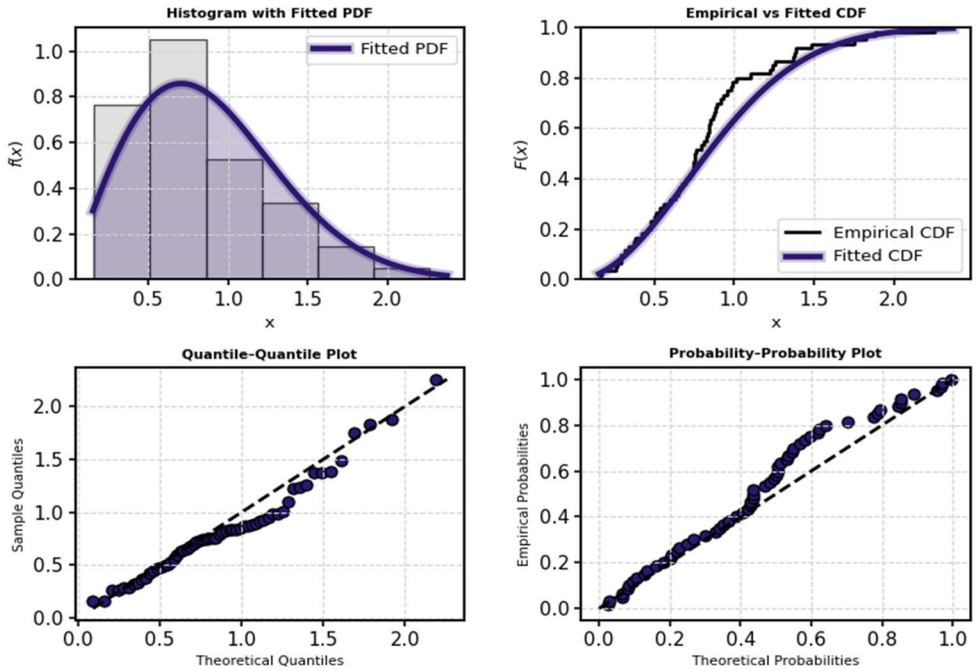


Figure 13: Fitted plots for Weibull distribution

- **Python code for fitted plots of the Gamma distribution.**

```
from fitPlotpy import plot_fitted
import numpy as np
from scipy.special import gamma, gammainc, gammaincinv

# PDF
def gamma_pdf(x, alpha, beta):
    return (beta**alpha / gamma(alpha)) * (x**(alpha - 1)) * np.exp(-beta * x
)

# CDF
def gamma_cdf(x, alpha, beta):
    return gammainc(alpha, beta * x)

# Quantile Function (Inverse CDF)
def gamma_quantile(p, alpha, beta):
    return gammaincinv(alpha, p) / beta

np.random.seed(2)
data = np.random.gamma(shape=2, scale=1, size=30)
params = {"alpha": 2, "beta": 1} # Estimated parameters

plot_fitted(pdf=gamma_pdf, cdf=gamma_cdf, ppf=gamma_quantile,
            params=params, bins=6, data=data)
```

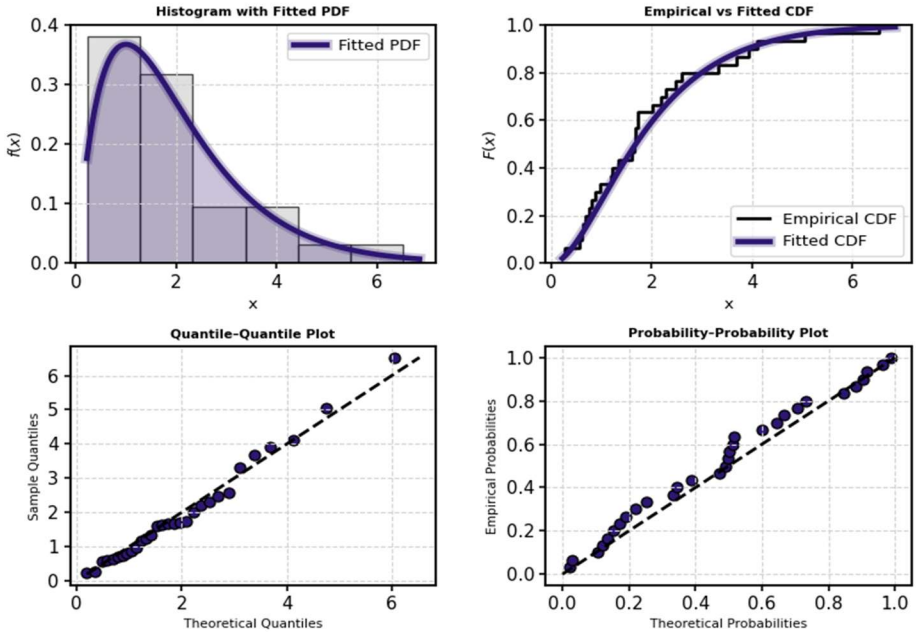


Figure 14: Fitted plots for gamma distribution

3.2 Plotting Multiple Fitted Distribution with Diagnostic Plots

To plot multiple fitted distributions with diagnostics plots such as fitted PDFs fitted CDFs against the empirical CDF. The **fitPlotpy** package provides the function **plot_multi_fitted** which automatically generates all these plots by defining PDF, CDF and quantile function of a distribution. To demonstrate the use of **plot_multi_fitted** consider different probability distributions such as exponential distribution, Weibull distribution, generalized exponential and Rayleigh distribution.

- **Python code for multiple fitted PDFs and CDFs.**

```
from fitPlotpy import plot_multi_fitted
import numpy as np
# Exponential
def exp_pdf(x, lam):
    return lam * np.exp(-lam*x)
def exp_cdf(x, lam):
    return 1 - np.exp(-lam*x)
# Weibull
def weib_pdf(x, c, scale):
    return (c/scale)*(x/scale)**(c-1) * np.exp(-(x/scale)**c)
def weib_cdf(x, c, scale):
    return 1 - np.exp(-(x/scale)**c)
# Generalized Exponential
def ge_pdf(x, alpha, lam):
    return alpha * lam * np.exp(-lam*x) * (1 - np.exp(-lam*x))**(alpha-1)
def ge_cdf(x, alpha, lam):
    return (1 - np.exp(-lam*x))**alpha
# Rayleigh
def rayleigh_pdf(x, sigma):
    return (x / sigma**2) * np.exp(-x**2 / (2*sigma**2))
def rayleigh_cdf(x, sigma):
    return 1 - np.exp(-x**2 / (2*sigma**2))
# ----------------- Sample Data -----------------
np.random.seed(1)
data = np.random.weibull(1.5, 50)

# ----------------- Distribution List -----------------
dist_list = [
    {"name": "Exponential", "pdf": exp_pdf, "cdf": exp_cdf, "params": {"lam":
1/np.mean(data)}},
    {"name": "Weibull", "pdf": weib_pdf, "cdf": weib_cdf, "params": {"c": 1.5
, "scale": data.mean()}},
    {"name": "Gen. Exponential", "pdf": ge_pdf, "cdf": ge_cdf, "params": {"al
pha": 1.2, "lam": 0.8}},
    {"name": "Rayleigh", "pdf": rayleigh_pdf, "cdf": rayleigh_cdf, "params":
{"sigma": 1.0}}
]
# Plot
plot_multi_fitted(data, dist_list)
```

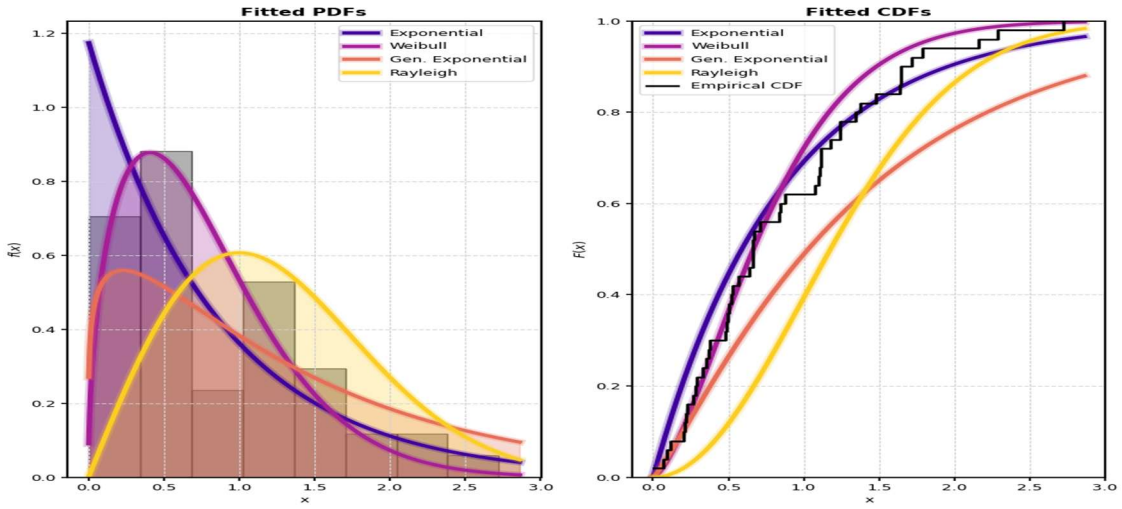


Figure 15: Fitted Multiple PDFs and CDFs

3.4 Plotting Data Overview

To plot histogram, boxplot, kernel density and violin plot of the observed data **fitPlotpy** package provides function **plot_data** which automatically generates all these plots. To demonstrate the use of **plot_data** function we generate random data from exponential distribution.

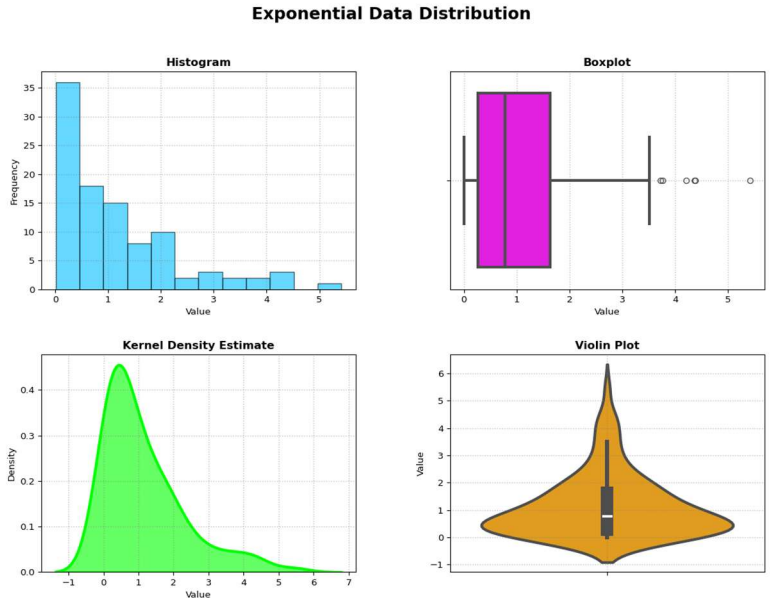


Figure 16: Data Overview

4. Conclusion

This paper introduces **fitPlotpy**, a Python package developed for the visualization of probability distributions. The package primarily facilitates the graphical representation of continuous probability distributions by providing tools to plot probability density functions (PDFs), cumulative distribution functions (CDFs), survival functions (SFs), and hazard functions (HFs) in a consistent, flexible, and customizable framework. Beyond theoretical visualization, **fitPlotpy** offers comprehensive diagnostic capabilities for fitted distributions, including fitted PDF(s), fitted versus empirical CDF(s), quantile–quantile (QQ) plot(s), and probability–probability (PP) plot(s), all accessible through a unified and user-friendly function interface. The package also incorporates functions to compute essential distributional moments, thereby strengthening its applicability for statistical modeling and analytical tasks. Furthermore, **fitPlotpy** provides an integrated data exploration module that generates key graphical summaries such as histograms, boxplots, kernel density plots, and Violin plots, offering users a complete and coherent framework for distribution analysis and model assessment.

References

[1]. Bokeh Development Team (2018). *Bokeh: Interactive Visualizations for Python*, 2018.
[2]. Harris, C. R. et al. (2020). *NumPy: A fundamental package for array computing in Python*.
[3]. Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*.
[4]. Meurer, A. et al. (2017). *SymPy: Symbolic Computing in Python*.
[5]. McKinney, W. (2015). *pandas: A foundational Python library for data analysis*, 2010.
[6]. Plotly Technologies Inc., *Plotly.py*.
[7]. Salvatier, J., Wiecki, T. V., & Fonnesbeck, C. (2016). *PyMC3: Bayesian Estimation using MCMC*.
[8]. Seabold, S. & Perktold, J. (2010). *statsmodels: Econometric and statistical modeling with Python*.
[9]. VanderPlas, J. et al. (2018). *Altair: Declarative Statistical Visualization in Python*.
[10]. Virtanen, P. et al. (2020). *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*.
[11]. Waskom, M. (2021). *Seaborn: Statistical Data Visualization*.
[12]. Yankov, M. & Templ, M. (2013). *ggplot for Python*.