Statistical Modelling of Climate Change using Python

Term Project

AG61654

Denny B Justin

20AG36009

Faculty Advisor – Professor Poulomi Ganguli

Abstract

The objective of the study is to analyze rainfall data of San Francisco and to study the rainfall variation in pattern happened in the last few years. Rainfall is fundamental aspects of Earth's climate system. By studying rainfall data probabilistically, we can characterize the statistical distribution of rainfall events, including their frequency, duration, intensity, and spatial distribution. Understanding the pattern of the rainfall is crucial for developing models and designing infrastructure to manage water resources effectively. Rainfall patterns exhibit considerable variability across different geographical regions and time periods. A probabilistic model was generated to study the variation in extreme rainfall data during the last thirty years. My aim is to study the historical data and provide valuable insights on extreme rainfall data. The project aims to bring awareness about statistical tools used in weather modelling and probabilistic analysis.

1 Introduction

Rainfall analysis stands as an important factor across diverse industries, wielding significant influence in modern operations and advancements. Probabilistic analysis on rainfall data helps in assessing and managing risks associated with extreme rainfall events, such as floods and landslides. Engineers and planners use probabilistic rainfall data to design hydraulic structures, drainage systems, and flood control measures. Industries like agriculture rely on rainfall analysis to determine crop growth, while in manufacturing, it is critical for ensuring quality control and the proper functioning of machinery.

Weather stations utilize a range of instruments to measure daily rainfall accurately. These stations employ traditional apparatus, which may be liquid-in-glass or digital, recording rainfall throughout the day. Additionally, many modern stations rely on electronic rainfall sensors, such as rain detectors, to continuously monitor and collect rainfall data. These sensors are often integrated into automated systems and work in conjunction with data loggers or recording devices to capture and store rainfall readings. In this project, I will be focusing on historical precipitation data provided by San Fransico Oceanside station. Analysis of historical precipitation data is essential to identify long-term trends and patterns. This data is crucial for understanding climate change effects, trends in extreme weather events, and informing adaptation strategies.

2 Data

The data for the research were extracted from open-source NOAA [1]. Daily rainfall data collected by San Fransico station from the date 01st January 1960 to 31st December 2020 were taken for analysis. The data has the location details and precipitation data.

3 Analysis and Methodology

Python programming was used to analyze the rainfall data. The data 61-year daily data was cleaned and undergone preprocessing. Extreme events refer to events that occur very rarely having indicating large magnitude of rainfall. The analysis was conducted on extreme events, that is only on top 1% of the events. There were 115 extreme events during 1960 to 2020.

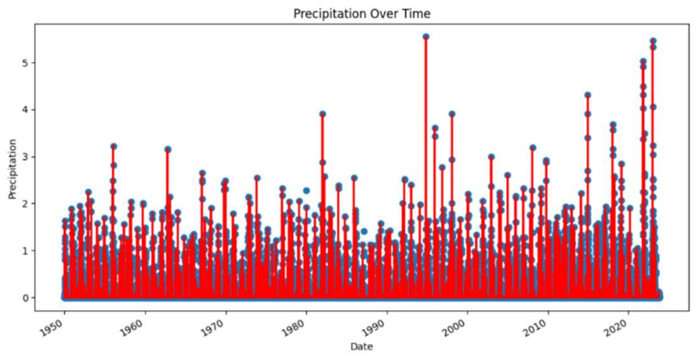


Fig 01

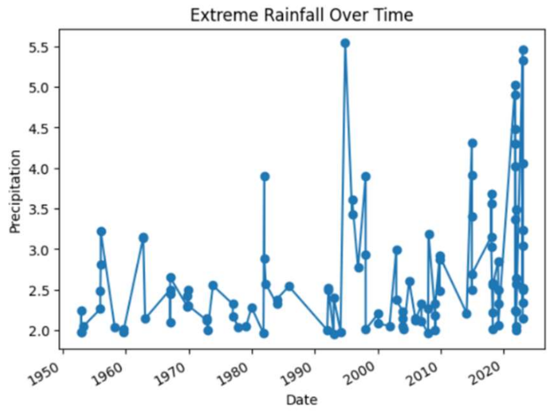
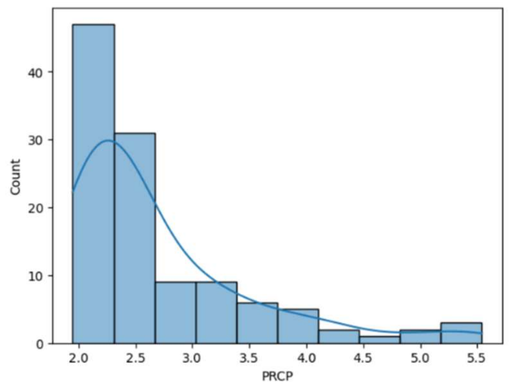


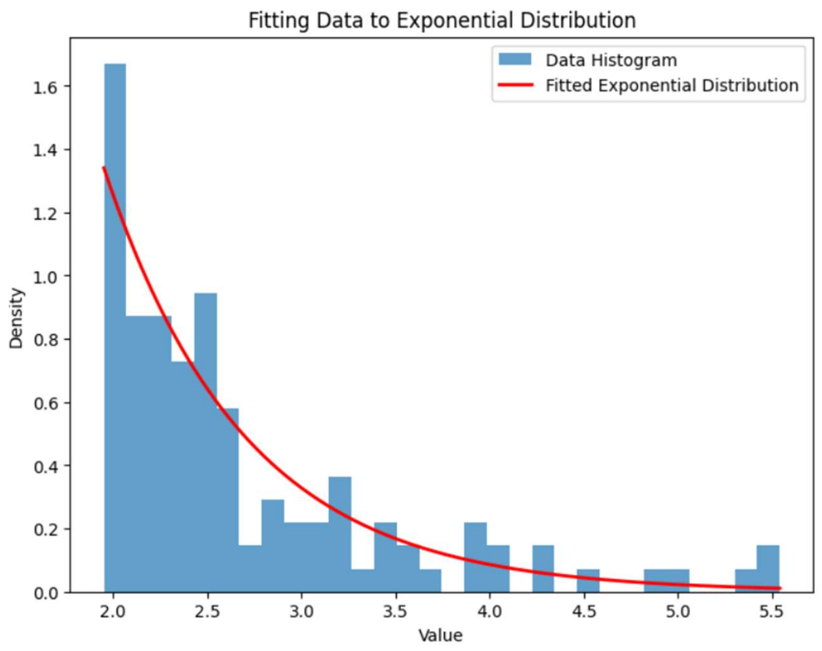
Figure 01 represents the daily rainfall data over the last 60 years. Figure 02 represents the extreme rainfall data that can potentially cause huge damage. Yearly minimum two extreme events occur in the coasts of San Fransico. As it shows, the extreme level rainfall probability have increased drastically indicating serious climate change effect in the area. In this research, will dive into the change in rainfall pattern using statistical methods.

3.1 Statistical Modelling of 60- year data



The frequency of the extreme events is represented here using the histogram. The data is plotted based on the counts of data present in the data rage. The probabilistic distribution is then fitted into various models. For this research we will fit the data into Exponential and Lognormal Models to study the data structure.

Mean: 2.696, Variance: 0.65, Skewness: 1.712, Kurtosis: 2.6



Mean Squared Error (MSE): 5.

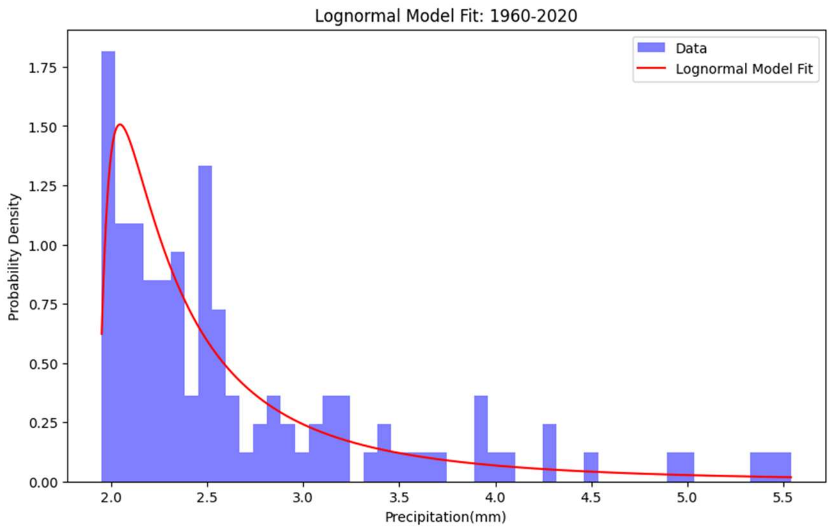
Root Mean Squared Error (RMSE): 2.319

KS Statistic: 0.0819

p-value: 0.40117

AIC: 166.603

BIC: 172.093



AIC: 175.21

BIC: 183.44

KS Statistic: 0.07

p-value: 0.5234

The exponential and lognormal models are fitted in the 1960-2020 extreme event data. The KS test is conducted to understand the goodness of fit of the model. The p-value of both the models are significant showing the model fits well in the data.

In comparing the goodness-of-fit, the AIC value of exponential distribution is lower than that of lognormal model. Also, the BIC value of exponential distribution is lower than that of lognormal model. This indicates that the **1960-2020 extreme event data fits on the exponential model than the log normal model.**

3.2 Statistical Modelling of 1960-1990 data

The 1960-1990 rainfall data is fitted into various statistical model to find the trend followed on specific time priod. This analysis is crutial in understanding the extend of climate change happened in the rainfall data.

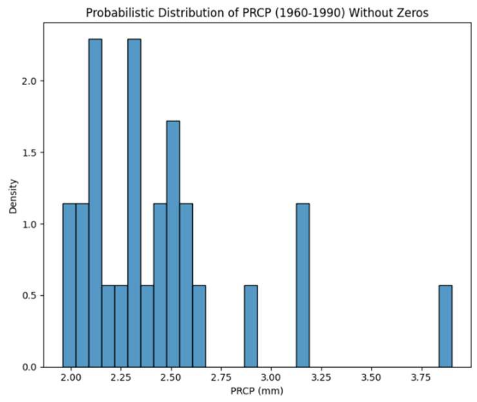
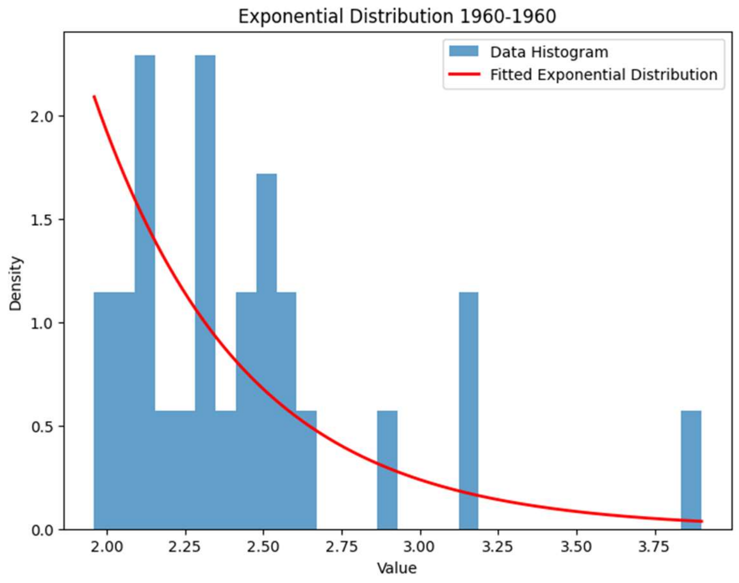


Figure 3 represent the probability distribution of the data over the time series of 1960-1990. Similar to the previous analysis, we need to find the best fitting model for the 1960-1990 dataset.

Mean: 2.44, Variance: 0.1747, Skewness: 1.75, Kurtosis: 3.41

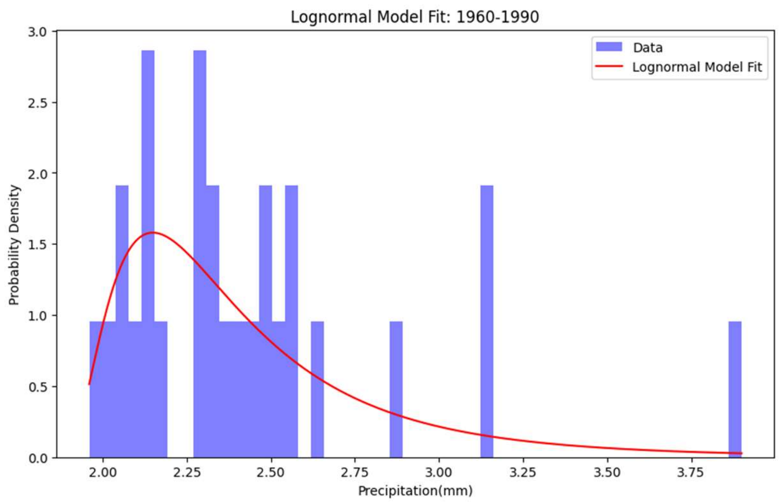


KS Statistic: 0.154

p-value: 0.4956

AIC: 18.24

BIC: 20.83



AIC: 20.70

BIC: 24.59

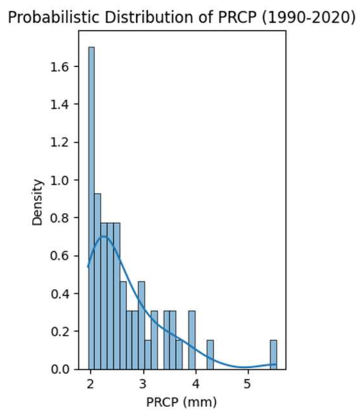
KS Statistic: 0.11

p-value: 0.8497

The AIC and BIC numbers are almost identical, suggesting that the rainfall can be both modeled in the exponential and lognormal method.

3.3 Statistical Modelling of 1990-2020 data

The 1990-2020 rainfall data is fitted into various statistical model to find the trend followed on specific time period. This analysis is crucial in understanding the extend of climate change happened in the rainfall data.

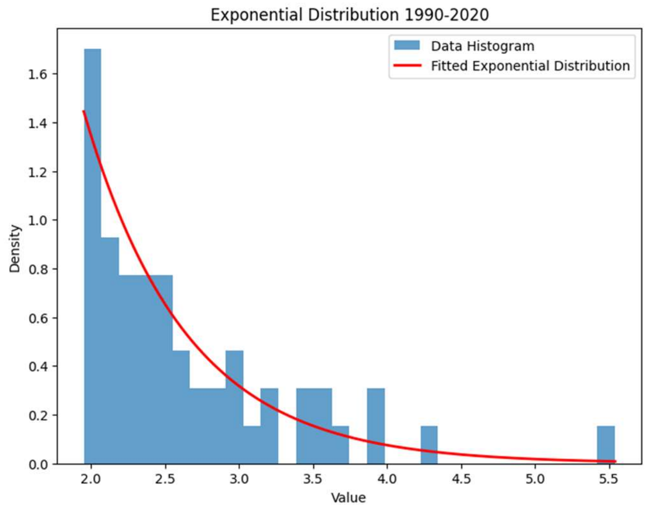


Mean: 2.642592592592593

Variance: 0.49934513031550076

Skewness: 1.7315555521383805

Kurtosis: 3.638868904082732

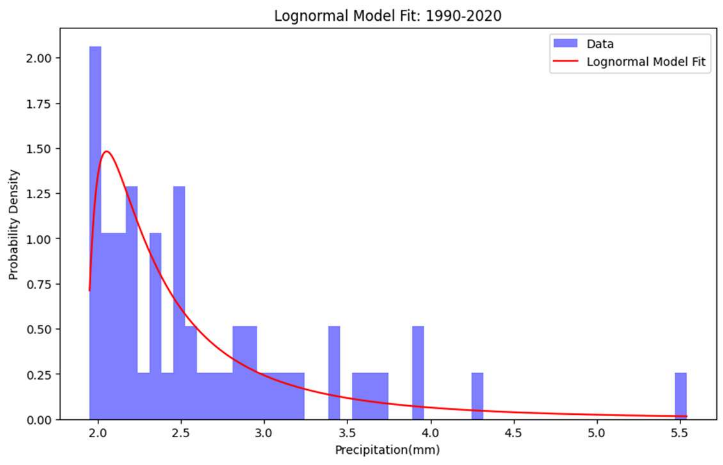


AIC: 20.70

BIC: 24.59

KS Statistic: 0.11

p-value: 0.8497



KS Statistic: 0.065

p-value: 0.96

AIC: 72.33

BIC: 76.308

The AIC and BIC data suggests that the model fit more on the exponential model since AIC and BIC of exponential is lower than that of AIC and BIC of lognormal models. The parameters of exponential data is highly advantage to model the data in exponential than lognormal.

4 Conclusion

The data indicates that the 1960-1990 data is more inclined to provide similar fit for both lognormal and exponential data, where as the 1990-2020 data indicates a slight incline towards the exponential data more. This shows the affect of climate change in rainfall causing the change in trend.

5 Reference

1. National Centers for Environmental Information. (n.d.). Climate Data Online. Retrieved from https://www.ncdc.noaa.gov/cdo-web/search
2. Code and Data Source (n.d.) Google Drive https://drive.google.com/drive/folders/1gfXXWGNjScNhX-hE8GAGuryjvMVQc0Vg?usp=sharing

6 Appendix

```
import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import scipy.stats as stats

import numpy as np

from sklearn.metrics import mean_squared_error
```

```
# Load the dataset

data_path = 'Data File.csv'

data = pd.read_csv('Data File.csv', low_memory=False)



# Display the first few rows of the dataset and the data types of the columns

data.head()

# Convert the 'date' column to datetime format

data['DATE'] = pd.to_datetime(data['DATE'])



# Sort the DataFrame based on the 'date' column (if needed)

data.sort_values(by='DATE', inplace=True)

# Plotting

plt.figure(figsize=(12, 6))  # Set the figure size (width, height)

# Plotting the data

plt.plot(data['DATE'], data['PRCP'], marker='o', linestyle='-')

plt.plot(data['DATE'], data['PRCP'], color='red', linestyle='-', linewidth=2, markersize=4)

# Adding labels and title

plt.xlabel('Date')

plt.ylabel('Precipitation')

plt.title('Precipitation Over Time')



# Beautifying the x-axis date labels

plt.gcf().autofmt_xdate()  # Rotate and align the x labels nicely

# Display the plot

plt.show()
```

```
#Plotting the extreme events

no_zero = data[data['PRCP'] > 0]

extreme_event = no_zero[no_zero['PRCP'] > no_zero['PRCP'].quantile(0.99)]

extreme_events = extreme_event[['DATE', 'PRCP']].copy()

print(extreme_events)

plt.plot(extreme_events['DATE'], extreme_events['PRCP'], marker='o')

#plt.plot(no_zero['DATE'], no_zero['PRCP'], color='red', linestyle='-', linewidth=2, markersize=4)


# Adding labels and title

plt.xlabel('Date')

plt.ylabel('Precipitation')

plt.title('Extreme Rainfall Over Time')


# Beautifying the x-axis date labels

plt.gcf().autofmt_xdate()  # Rotate and align the x labels nicely


# Display the plot

plt.show()

sns.histplot(extreme_events['PRCP'], bins=10, kde=True)


import scipy.stats as stats


#extreme_event = np.array(extreme_event, dtype=np.float64)

ext_prcp = extreme_events['PRCP']

# Fit the data to an Exponential Distribution

loc, scale = stats.expon.fit(ext_prcp)
```

```
# Generate values for the Exponential Distribution using the fitted parameters
xmin, xmax = np.min(ext_prcp), np.max(ext_prcp)
x = np.linspace(xmin, xmax, 1000)
fitted_distribution = stats.expon.pdf(x, loc=loc, scale=scale)

# Plotting the histogram of the data
plt.figure(figsize=(8, 6))
plt.hist(ext_prcp, bins=30, density=True, alpha=0.7, label='Data Histogram')

# Plotting the fitted Exponential Distribution
plt.plot(x, fitted_distribution, 'r-', linewidth=2, label='Fitted Exponential Distribution')

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Fitting Data to Exponential Distribution')
plt.legend()

# Show plot
plt.show()



fitted_distribution = stats.expon(loc=loc, scale=scale)
predicted_values = fitted_distribution.pdf(ext_prcp)



mean_value = np.mean(ext_prcp)
variance = np.var(ext_prcp)
```

```
skewness = stats.skew(ext_prcp)

kurtosis_value = stats.kurtosis(ext_prcp)


print(f"Mean: {mean_value}")

print(f"Variance: {variance}")

print(f"Skewness: {skewness}")

print(f"Kurtosis: {kurtosis_value}")



# Calculate Mean Squared Error (MSE)

mse = mean_squared_error(ext_prcp, predicted_values)

print(f"Mean Squared Error (MSE): {mse}")


# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(mse)

print(f"Root Mean Squared Error (RMSE): {rmse}")


ks_statistic, ks_p_value = stats.kstest(ext_prcp, fitted_distribution.cdf)


print(f"KS Statistic: {ks_statistic}")

print(f"p-value: {ks_p_value}")


log_likelihood = np.sum(fitted_distribution.logpdf(ext_prcp))


# Number of parameters in the model

num_params = 2  # Exponential distribution: loc and scale


# Calculate AIC and BIC
```

```
n = len(ext_prcp)
aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log(n)

print(f"Akaike Information Criterion (AIC): {aic}")
print(f"Bayesian Information Criterion (BIC): {bic}")

alpha = 0.05  # Significance level
if ks_p_value < alpha:
    print("The data does not follow the Exponential Distribution (reject H0)")
else:
    print("The data follows the Exponential Distribution (fail to reject H0)")
    from scipy.stats import laplace
from scipy.optimize import curve_fit

loc, scale = laplace.fit(ext_prcp)

# 4. Plot histogram of data with fitted Laplace distribution
plt.hist(ext_prcp, bins=30, density=True, alpha=0.6, color='y', label='Data')  # Plot histogram

# Generate points for the PDF using the fitted Laplace parameters
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = laplace.pdf(x, loc=loc, scale=scale)
plt.plot(x, p, 'k', linewidth=2, label='Fitted Laplace distribution')  # Plot PDF

plt.title('Fit Double Exponential Distribution to Data')
plt.xlabel('Precipitation')
plt.ylabel('Density')
plt.legend()

plt.show()

print(f"Fitted Laplace distribution parameters (loc, scale): {loc}, {scale}")

predicted_values = laplace.pdf(ext_prcp)



ks_statistic, ks_p_value = stats.kstest(ext_prcp, laplace.cdf)

print(f"KS Statistic: {ks_statistic}")
print(f"p-value: {ks_p_value}")

log_likelihood = np.sum(laplace.logpdf(ext_prcp))

# Number of parameters in the model
num_params = 2  # Exponential distribution: loc and scale

# Calculate AIC and BIC
n = len(ext_prcp)
aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log(n)

print(f"Akaike Information Criterion (AIC): {aic}")
print(f"Bayesian Information Criterion (BIC): {bic}")
```

```
alpha = 0.05  # Significance level
if ks_p_value < alpha:
    print("The data does not follow the Exponential Distribution (reject H0)")
else:
    print("The data follows the Exponential Distribution (fail to reject H0)")
    from scipy.stats import pareto
from scipy.optimize import curve_fit
from scipy.stats import kstest
from scipy.optimize import minimize
from sklearn.metrics import mean_squared_error
import math

# Assuming you have loaded your precipitation data into ext_prcp
# Example:
# ext_prcp = np.array([1.2, 1.5, 2.3, 0.8, 1.9, ...])

# 3. Fit data to Pareto distribution
shape, loc, scale = pareto.fit(ext_prcp)

# 4. Plot histogram of data with fitted Pareto distribution
plt.hist(ext_prcp, bins=30, density=True, alpha=0.6, color='y', label='Data')  # Plot histogram

# Generate points for the PDF using the fitted Pareto parameters
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = pareto.pdf(x, shape, loc=loc, scale=scale)
plt.plot(x, p, 'k', linewidth=2, label='Fitted Pareto distribution')  # Plot PDF
plt.title('Fit Pareto Distribution to Data')
plt.xlabel('Precipitation')
plt.ylabel('Density')
plt.legend()

plt.show()

print(f"Fitted Pareto distribution parameters (shape, loc, scale): {shape}, {loc}, {scale}")

# 5. Calculate goodness-of-fit metrics (e.g., Kolmogorov-Smirnov test)
ks_statistic, ks_p_value = kstest(ext_prcp, 'pareto', args=(shape, loc, scale))
print(f"Kolmogorov-Smirnov test statistic: {ks_statistic}")
print(f"Kolmogorov-Smirnov test p-value: {ks_p_value}")

def neg_log_likelihood(params, data):
    shape, loc, scale = params
    return -np.sum(pareto.logpdf(data, shape, loc=loc, scale=scale))

# Fit Pareto distribution to the data
initial_guess = (1, np.min(ext_prcp), np.max(ext_prcp))  # Initial guess for parameters (shape,
loc, scale)
result = minimize(neg_log_likelihood, initial_guess, args=(ext_prcp,))
shape, loc, scale = result.x

# Calculate number of parameters in the model
k = 3  # Number of parameters (shape, loc, scale)

# Calculate negative log-likelihood and number of observations
```

```
n = len(ext_prcp)

neg_log_likelihood_value = -result.fun

bic = -2 * neg_log_likelihood_value + k * np.log(n)

aic = -2 * neg_log_likelihood_value + 2 * k


print(f"AIC: {aic}")

print(f"BIC: {bic}")

print(f"Fitted Pareto distribution parameters (shape, loc, scale): {shape}, {loc}, {scale}")

# Fitting the data to a lognormal distribution

params_lognorm = lognorm.fit(ext_prcp)

print('Lognormal Model Parameters:', params_lognorm)

pdf_lognorm = lognorm.pdf(np.linspace(ext_prcp.min(), ext_prcp.max(), 1000), *params_lognorm)


# Calculate AIC and BIC

lognorm_aic = -2 * lognorm.logpdf(ext_prcp, *params_lognorm).sum() + 2 * len(params_lognorm)

n = len(ext_prcp)

lognorm_bic = -2 * lognorm.logpdf(ext_prcp, *params_lognorm).sum() + len(params_lognorm) * np.log(n)


ks_statistic, ks_p_value = kstest(ext_prcp, 'lognorm', args=params_lognorm)


# Plotting the lognormal model fit graph

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


plt.hist(ext_prcp, bins=50, density=True, alpha=0.5, color='b', label='Data')

plt.plot(np.linspace(ext_prcp.min(), ext_prcp.max(), 1000), pdf_lognorm, color='red', label='Lognormal Model Fit')
plt.title('Lognormal Model Fit: 1960-2020')
plt.xlabel('Precipitation(mm)')
plt.ylabel('Probability Density')
plt.legend()
plt.show()


# Printing AIC, KS statistic, and p-value
print(f"Akaike Information Criterion (AIC): {lognorm_aic:.2f}")
print(f"Bayesian Information Criterion (BIC): {lognorm_bic:.2f}")
print(f"KS Statistic: {ks_statistic:.2f}")
print(f"p-value: {ks_p_value:.4f}")
extreme_events['DATE'] = pd.to_datetime(extreme_events['DATE'], errors='coerce')


# Split data into two groups
data_1990_2020 = extreme_events[(extreme_events['DATE'].dt.year <= 2020) & (extreme_events['DATE'].dt.year > 1990)]
data_1960_1990 = extreme_events[(extreme_events['DATE'].dt.year <= 1990) & (extreme_events['DATE'].dt.year > 1960)]
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.histplot(data_1990_2020['PRCP'], kde=True, stat="density", bins=30)
plt.title('Probabilistic Distribution of PRCP (1990-2020)')
plt.xlabel('PRCP (mm)')
plt.ylabel('Density')
plt.subplot(1, 2, 2)
sns.histplot(data_1960_1990['PRCP'], kde=True, stat="density", bins=30)
plt.title('Probabilistic Distribution of PRCP (1960-1990)')
plt.xlabel('PRCP (mm)')
plt.ylabel('Density')
plt.tight_layout()
plt.show()
# Filter out entries where PRCP is 0
data_1990_2020_no_zeros = data_1990_2020[data_1990_2020['PRCP'] > 0]
data_1960_1990_no_zeros = data_1960_1990[data_1960_1990['PRCP'] > 0]

# Replotting the probabilistic distributions without zeros
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.histplot(data_1990_2020_no_zeros['PRCP'], stat="density", bins=30)
plt.title('Probabilistic Distribution of PRCP (1990-2020) Without Zeros')
plt.xlabel('PRCP (mm)')
plt.ylabel('Density')

plt.subplot(1, 2, 2)
sns.histplot(data_1960_1990_no_zeros['PRCP'], stat="density", bins=30)
plt.title('Probabilistic Distribution of PRCP (1960-1990) Without Zeros')
plt.xlabel('PRCP (mm)')
plt.ylabel('Density')

plt.tight_layout()
plt.show()
import scipy.stats as stats

prcp_90_20 = data_1990_2020_no_zeros['PRCP']
# Fit the data to an Exponential Distribution
loc, scale = stats.expon.fit(prcp_90_20)
```

```
# Generate values for the Exponential Distribution using the fitted parameters
xmin, xmax = np.min(prcp_90_20), np.max(prcp_90_20)
x = np.linspace(xmin, xmax, 1000)
fitted_distribution = stats.expon.pdf(x, loc=loc, scale=scale)

# Plotting the histogram of the data
plt.figure(figsize=(8, 6))
plt.hist(prcp_90_20, bins=30, density=True, alpha=0.7, label='Data Histogram')

# Plotting the fitted Exponential Distribution
plt.plot(x, fitted_distribution, 'r-', linewidth=2, label='Fitted Exponential Distribution')

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Exponential Distribution 1990-2020')
plt.legend()

# Show plot
plt.show()



fitted_distribution = stats.expon(loc=loc, scale=scale)
predicted_values = fitted_distribution.pdf(prcp_90_20)



mean_value = np.mean(prcp_90_20)
```

```
variance = np.var(prcp_90_20)

skewness = stats.skew(prcp_90_20)

kurtosis_value = stats.kurtosis(prcp_90_20)


print(f"Mean: {mean_value}")

print(f"Variance: {variance}")

print(f"Skewness: {skewness}")

print(f"Kurtosis: {kurtosis_value}")



# Calculate Mean Squared Error (MSE)

mse = mean_squared_error(prcp_90_20, predicted_values)

print(f"Mean Squared Error (MSE): {mse}")


# Calculate Root Mean Squared Error (RMSE)

rmse = np.sqrt(mse)

print(f"Root Mean Squared Error (RMSE): {rmse}")


ks_statistic, ks_p_value = stats.kstest(prcp_90_20, fitted_distribution.cdf)


print(f"KS Statistic: {ks_statistic}")

print(f"p-value: {ks_p_value}")


log_likelihood = np.sum(fitted_distribution.logpdf(prcp_90_20))


# Number of parameters in the model

num_params = 2  # Exponential distribution: loc and scale
```

```
# Calculate AIC and BIC
n = len(prcp_90_20)
aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log(n)

print(f"Akaike Information Criterion (AIC): {aic}")
print(f"Bayesian Information Criterion (BIC): {bic}")

alpha = 0.05  # Significance level
if ks_p_value < alpha:
    print("The data does not follow the Exponential Distribution (reject H0)")
else:
    print("The data follows the Exponential Distribution (fail to reject H0)")

import scipy.stats as stats

prcp_60_90 = data_1960_1990_no_zeros['PRCP']
# Fit the data to an Exponential Distribution
loc, scale = stats.expon.fit(prcp_60_90)

# Generate values for the Exponential Distribution using the fitted parameters
xmin, xmax = np.min(prcp_60_90), np.max(prcp_60_90)
x = np.linspace(xmin, xmax, 1000)
fitted_distribution = stats.expon.pdf(x, loc=loc, scale=scale)

# Plotting the histogram of the data
plt.figure(figsize=(8, 6))
plt.hist(prcp_60_90, bins=30, density=True, alpha=0.7, label='Data Histogram')
```

```
# Plotting the fitted Exponential Distribution
plt.plot(x, fitted_distribution, 'r-', linewidth=2, label='Fitted Exponential Distribution')

# Adding labels and title
plt.xlabel('Value')
plt.ylabel('Density')
plt.title('Exponential Distribution 1960-1990')
plt.legend()

# Show plot
plt.show()



fitted_distribution = stats.expon(loc=loc, scale=scale)
predicted_values = fitted_distribution.pdf(prcp_60_90)



mean_value = np.mean(prcp_60_90)
variance = np.var(prcp_60_90)
skewness = stats.skew(prcp_60_90)
kurtosis_value = stats.kurtosis(prcp_60_90)

print(f"Mean: {mean_value}")
print(f"Variance: {variance}")
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis_value}")
```

```
# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(prcp_60_90, predicted_values)
print(f"Mean Squared Error (MSE): {mse}")

# Calculate Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)
print(f"Root Mean Squared Error (RMSE): {rmse}")

ks_statistic, ks_p_value = stats.kstest(prcp_60_90, fitted_distribution.cdf)

print(f"KS Statistic: {ks_statistic}")
print(f"p-value: {ks_p_value}")

log_likelihood = np.sum(fitted_distribution.logpdf(prcp_60_90))

# Number of parameters in the model
num_params = 2  # Exponential distribution: loc and scale

# Calculate AIC and BIC
n = len(prcp_60_90)
aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log(n)

print(f"Akaike Information Criterion (AIC): {aic}")
print(f"Bayesian Information Criterion (BIC): {bic}")

alpha = 0.05  # Significance level
```

```
if ks_p_value < alpha:
    print("The data does not follow the Exponential Distribution (reject H0)")
else:
    print("The data follows the Exponential Distribution (fail to reject H0)")

# Fitting the data to a lognormal distribution
params_lognorm = lognorm.fit(prcp_90_20)
print('Lognormal Model Parameters:', params_lognorm)
pdf_lognorm = lognorm.pdf(np.linspace(prcp_90_20.min(), prcp_90_20.max(), 1000),
*params_lognorm)

# Calculate AIC and BIC
lognorm_aic = -2 * lognorm.logpdf(prcp_90_20, *params_lognorm).sum() + 2 *
len(params_lognorm)
n = len(prcp_90_20)
lognorm_bic = -2 * lognorm.logpdf(prcp_90_20, *params_lognorm).sum() +
len(params_lognorm) * np.log(n)

ks_statistic, ks_p_value = kstest(prcp_90_20, 'lognorm', args=params_lognorm)

# Plotting the lognormal model fit graph
plt.figure(figsize=(10, 6))

plt.hist(prcp_90_20, bins=50, density=True, alpha=0.5, color='b', label='Data')
plt.plot(np.linspace(prcp_90_20.min(), prcp_90_20.max(), 1000), pdf_lognorm, color='red',
label='Lognormal Model Fit')
plt.title('Lognormal Model Fit: 1990-2020')
plt.xlabel('Precipitation(mm)')
plt.ylabel('Probability Density')
plt.legend()

plt.show()


# Printing AIC, KS statistic, and p-value

print(f"Akaike Information Criterion (AIC): {lognorm_aic:.2f}")

print(f"Bayesian Information Criterion (BIC): {lognorm_bic:.2f}")

print(f"KS Statistic: {ks_statistic:.2f}")

print(f"p-value: {ks_p_value:.4f}")


# Fitting the data to a lognormal distribution

params_lognorm = lognorm.fit(prcp_60_90)

print('Lognormal Model Parameters:', params_lognorm)

pdf_lognorm = lognorm.pdf(np.linspace(prcp_60_90.min(), prcp_60_90.max(), 1000), *params_lognorm)


# Calculate AIC and BIC

lognorm_aic = -2 * lognorm.logpdf(prcp_60_90, *params_lognorm).sum() + 2 * len(params_lognorm)

n = len(prcp_60_90)

lognorm_bic = -2 * lognorm.logpdf(prcp_60_90, *params_lognorm).sum() + len(params_lognorm) * np.log(n)


ks_statistic, ks_p_value = kstest(prcp_60_90, 'lognorm', args=params_lognorm)


# Plotting the lognormal model fit graph

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


plt.hist(prcp_60_90, bins=50, density=True, alpha=0.5, color='b', label='Data')
plt.plot(np.linspace(prcp_60_90.min(), prcp_60_90.max(), 1000), pdf_lognorm, color='red', label='Lognormal Model Fit')

plt.title('Lognormal Model Fit: 1960-1990')

plt.xlabel('Precipitation(mm)')

plt.ylabel('Probability Density')

plt.legend()

plt.show()


# Printing AIC, KS statistic, and p-value

print(f"Akaike Information Criterion (AIC): {lognorm_aic:.2f}")

print(f"Bayesian Information Criterion (BIC): {lognorm_bic:.2f}")

print(f"KS Statistic: {ks_statistic:.2f}")

print(f"p-value: {ks_p_value:.4f}")


# Replotting the probabilistic distributions with a higher number of bins for more detail

plt.figure(figsize=(14, 6))


plt.subplot(1, 2, 1)

sns.histplot(prcp_90_20, kde=True, stat="density", bins=60)  # Increased bin count

plt.title('Refined Probabilistic Distribution of PRCP (1990-2020) Without Zeros')

plt.xlabel('PRCP (mm)')

plt.ylabel('Density')


plt.subplot(1, 2, 2)

sns.histplot(data_1960_1990_no_zeros['PRCP'], kde=True, stat="density", bins=60)  # Increased bin count

plt.title('Refined Probabilistic Distribution of PRCP (1960-1990) Without Zeros')

plt.xlabel('PRCP (mm)')

plt.ylabel('Density')
plt.tight_layout()

plt.show()


# Plotting the Cumulative Distribution Function (CDF) of PRCP

plt.figure(figsize=(12, 6))


plt.subplot(1, 2, 1)

sns.histplot(data_1960_1990['PRCP'], kde=False, stat="probability", element="step", cumulative=True, fill=False, color="blue")

plt.title('Cumulative Distribution of PRCP (1960-1990)')

plt.xlabel('PRCP (mm)')

plt.ylabel('Cumulative Probability')


plt.subplot(1, 2, 2)

sns.histplot(data_1990_2020['PRCP'], kde=False, stat="probability", element="step", cumulative=True, fill=False, color="green")

plt.title('Cumulative Distribution of PRCP (1990-2020)')

plt.xlabel('PRCP (mm)')

plt.ylabel('Cumulative Probability')


plt.tight_layout()

plt.show()

import scipy.stats as stats


# Clean data by dropping NA values for precise plotting

prcp_1960_1990_clean = prcp_60_90.dropna()

prcp_1990_2020_clean = prcp_90_20.dropna()
```

```
# Generating Q-Q plot
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
stats.probplot(prcp_1960_1990_clean, dist="norm", plot=plt)
stats.probplot(prcp_1990_2020_clean, dist="norm", plot=plt)
plt.title('Q-Q Plot')
plt.xlabel('Theoretical Quantiles')
plt.ylabel('Ordered Values')

# Generating P-P plot
plt.subplot(1, 2, 2)
stats.probplot(prcp_1960_1990_clean, dist="uniform", plot=plt)
stats.probplot(prcp_1990_2020_clean, dist="uniform", plot=plt)
plt.title('P-P Plot')
plt.xlabel('Theoretical Probabilities')
plt.ylabel('Ordered Values')

plt.tight_layout()
plt.show()
```