Estimating Equalization Reserves for Landfill

In this project our job is to take into account various parameters as set by Jacob. Our objective is to derive the aggregate claims distribution over a 1-year time horizon.

In [7]:
```
#loading the dataset
import pandas as pd
data = pd.read_excel("C:\\Users\\sujoydutta\\Downloads\\landfillinsurance.xlsx")
data
```

Out[7]:

	Year	WIS	NONWIS
0	2016	3.10	4.5
1	2016	2.10	NaN
2	2016	10.50	NaN
3	2017	2.00	NaN
4	2018	NaN	125.3
5	2018	NaN	NaN
6	2019	230.05	0.4
7	2019	51.00	NaN
8	2020	0.50	NaN

In [8]:
```
#removing whitespaces
data.columns = data.columns.str.strip()
data.columns
```

Out[8]:
```
Index(['Year', 'WIS', 'NONWIS'], dtype='object')
```

In [12]:
```
# setting inflation rate and years since loss
inflation_rate = 0.03
current_year = 2021
data['Years_Since_Loss'] = current_year - data['Year']
```

In [13]:
```
# Function to adjust loss amounts for inflation
def adjust_loss(amount, years_since):
    return amount * (1 + inflation_rate) ** years_since
```

In [15]:
```
# Adjust loss data for inflation
import numpy as np
data['WIS_Adjusted'] = data.apply(lambda row: adjust_loss(row['WIS'], row['Years_Since_L
data['NONWIS_Adjusted'] = data.apply(lambda row: adjust_loss(row['NONWIS'], row['Years_S
```

In [16]:
```
# Calculating frequency parameter λ
def calculate_frequency(df, column):
    total_claims = df[column].notna().sum()
    num_years = len(df['Year'].unique())
    num_sites = len(df[column].dropna().unique())
    return total_claims / (num_sites * num_years)
```

In [17]:
```
#obtaining frequency
lambda_wis = calculate_frequency(data, 'WIS_Adjusted')
```

```
lambda_nonwis = calculate_frequency(data, 'NONWIS_Adjusted')
```

In [21]:

```
# Calculating severity distribution parameters
from scipy import stats
def fit_severity_distribution(df, column):
    data = df[column].dropna()
    mu, sigma = stats.norm.fit(np.log(data))
    return mu, sigma
```

In [22]:

```
#obtaining severity distribution

mu_wis, sigma_wis = fit_severity_distribution(data, 'WIS_Adjusted')
mu_nonwis, sigma_nonwis = fit_severity_distribution(data, 'NONWIS_Adjusted')
```

In [23]:

```
# Calculating expected value for 2021
def calculate_expected_value(lambda_param, mu, sigma):
    return lambda_param * np.exp(mu + 0.5 * sigma ** 2)
```

In [24]:

```
#Obtaining 2021 values
expected_value_wis = calculate_expected_value(lambda_wis, mu_wis, sigma_wis)
expected_value_nonwis = calculate_expected_value(lambda_nonwis, mu_nonwis, sigma_nonwis)
```

In [25]:

```
# Value at Risk (80% VaR) using CLT
def calculate_var(expected_value, std_dev, confidence_level=0.80):
    z_score = stats.norm.ppf(confidence_level)
    return expected_value + z_score * std_dev
```

In [26]:

```
#obtaining risk values
std_dev_wis = np.sqrt(lambda_wis * (np.exp(sigma_wis ** 2) - 1) * np.exp(2 * mu_wis + si
std_dev_nonwis = np.sqrt(lambda_nonwis * (np.exp(sigma_nonwis ** 2) - 1) * np.exp(2 * mu

var_wis = calculate_var(expected_value_wis, std_dev_wis)
var_nonwis = calculate_var(expected_value_nonwis, std_dev_nonwis)
```

In [27]:

```
# Impact of doubling the frequency
def impact_of_doubling_frequency(lambda_param, mu, sigma):
    new_lambda = 2 * lambda_param
    return calculate_expected_value(new_lambda, mu, sigma)
```

In [28]:

```
#Obtaining impact
new_expected_value_wis = impact_of_doubling_frequency(lambda_wis, mu_wis, sigma_wis)
new_expected_value_nonwis = impact_of_doubling_frequency(lambda_nonwis, mu_nonwis, sigma
```

In [29]:

```
# Confidence Interval for Expected Aggregate Loss
def confidence_interval(expected_value, std_dev, confidence_level=0.80):
    z_score = stats.norm.ppf((1 + confidence_level) / 2)
    margin_of_error = z_score * std_dev
    return expected_value - margin_of_error, expected_value + margin_of_error
```

In [30]:

```
#Obtaining confidence interval
ci_wis = confidence_interval(expected_value_wis, std_dev_wis)
ci_nonwis = confidence_interval(expected_value_nonwis, std_dev_nonwis)
```

In [31]:

```
# Print results
print(f'Frequency parameter (WIS): {lambda_wis}')
print(f'Frequency parameter (NONWIS): {lambda_nonwis}')
print(f'Severity parameters (WIS): mu={mu_wis}, sigma={sigma_wis}')
print(f'Severity parameters (NONWIS): mu={mu_nonwis}, sigma={sigma_nonwis}')
print(f'Expected value (WIS) for 2021: {expected_value_wis}')
print(f'Expected value (NONWIS) for 2021: {expected_value_nonwis}')
print(f'80% VaR (WIS): {var_wis}')
print(f'80% VaR (NONWIS): {var_nonwis}')
print(f'New Expected Value (WIS) with doubled frequency: {new_expected_value_wis}')
print(f'New Expected Value (NONWIS) with doubled frequency: {new_expected_value_nonwis}'
print(f'80% Confidence Interval (WIS): {ci_wis}')
print(f'80% Confidence Interval (NONWIS): {ci_nonwis}')
```

```
Frequency parameter (WIS): 0.2
Frequency parameter (NONWIS): 0.2
Severity parameters (WIS): mu=2.0434640395009747, sigma=1.952707910688474
Severity parameters (NONWIS): mu=1.9046951830731953, sigma=2.3650373355964693
Expected value (WIS) for 2021: 10.387053961164739
Expected value (NONWIS) for 2021: 22.020152791302692
80% VaR (WIS): 140.4767505883692
80% VaR (NONWIS): 699.9799388183317
New Expected Value (WIS) with doubled frequency: 20.774107922329478
New Expected Value (NONWIS) with doubled frequency: 44.040305582605384
80% Confidence Interval (WIS): (-187.70283222915612, 208.47694015148562)
80% Confidence Interval (NONWIS): (-1010.3212265614425, 1054.361532144048)
```