bodaboda

AUTHOR
migisha

FROM RAINY ROADS TO RIDER RELIEF

Boda-Safe Shield: Dynamic Motorcycle Insurance Pricing Model

Project Goal: To develop a data-driven, dynamic pricing model for motorcycle insurance (specifically targeting Boda Boda riders in Kampala, Uganda) that incorporates real-time weather risk to calculate a variable monthly premium.

Methodology and Execution:

1. **Data Acquisition:**
    - **Historical Weather Data (Offline):** Daily precipitation data for Kampala (Latitude 0.3476, Longitude 32.5825) was fetched from the Open-Meteo API for the period $2019-01-01$ to $2023-12-31$.
    - **Synthetic Claims Data:** Accident frequency and severity data were simulated using a Poisson distribution for frequency and a Gamma distribution for severity. A key feature, `risk_trigger`, was engineered: 1 if precipitation >10mm, and 0 otherwise.
    - **Feature Engineering:** The month of the year and the `risk\_trigger` were the primary input features for the modeling phase.
2. **Statistical Modeling:**
    - **Frequency Model:** Two models were trained to predict accident *frequency* (Accidents per Exposure Month):
        - **GLM (Poisson Family):** Used for interpretability and establishing a baseline.
        - **GBM (Gradient Boosting Machine):** Selected for its non-linear predictive power, outperforming the GLM based on model fit metrics.
    - **Severity Model:** A **GLM (Gamma Family)** was used to model the average *cost* (severity) of a claim.
3. **Risk Finding:**
    - The synthetic data demonstrated a strong separation in accident frequency: days with the rain `risk\_trigger` active had a simulated accident rate (Poisson λ) approximately 66% **higher** than dry days (e.g., λ=0.25 vs. λ=0.15). This justified the use of precipitation as a primary rating factor.

4. **Deployment (Streamlit Application):**
   - The trained GBM model was saved (`gbm\_model.pkl`) using `joblib`.
   - A web application was built using **Streamlit** and Python, enabling users to input location and daily hours.
   - The app performs an **on-demand API call** to Open-Meteo for *tomorrow's* forecast.
   - The final premium is calculated based on the formula:

     $$\text{Premium} = \text{Predicted Frequency} \times \text{Daily Hours} \times \text{Rate} \times 30 \text{ days}$$

The BodaSafe Shield Quote Tool serves as a robust proof-of-concept for a data-driven, weather-contingent insurance pricing strategy.

```
#install.packages("reticulate")

library(reticulate)
```

```
Warning: package 'reticulate' was built under R version 4.4.3
```

1. **DATA ACQUISITION**

   Both the r and python code pull 5 years of daily precipitation data for Kampala from the free Open-Meteo API, serving as a proxy for accident risk (rainy days trigger higher claims). Output: Raw CSV (~1,800 rows) for downstream analysis, this is essential for reproducible, real-world data sourcing without manual downloads.

R SYNTAX

```
#Load libraries
library(httr2)
```

```
Warning: package 'httr2' was built under R version 4.4.3
```

```
library(dplyr)
```

```
Warning: package 'dplyr' was built under R version 4.4.3


Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':
```

```
    intersect, setdiff, setequal, union
```

```
library(readr)
```

```
Warning: package 'readr' was built under R version 4.4.3
```

```
# API request
req <- request("https://archive-api.open-meteo.com/v1/archive") %>%
  req_url_query(
    latitude = 0.3476, longitude = 32.5825,
    start_date = "2019-01-01", end_date = "2023-12-31",
    daily = "precipitation_sum"
  )
resp <- req_perform(req) %>% resp_body_json()

# Parse to tibble
rain_data <- tibble(
  date = as.Date(unlist(resp$daily$time)),
  precip_mm = as.numeric(resp$daily$precipitation_sum)
) %>% filter(precip_mm >= 0)  # Filter invalid

# Save
write_csv(rain_data, "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/boda_rain_
head(rain_data)
```

```
# A tibble: 6 × 2
  date       precip_mm
  <date>         <dbl>
1 2019-01-01       1.8
2 2019-01-02       2.5
3 2019-01-03       5.5
4 2019-01-04      18.4
5 2019-01-05       6.4
6 2019-01-06       2.5
```

```
# Output: ~1826 rows, e.g., date: 2019-01-01, precip_mm: 0.0
```

PYTHON SYNTAX DATA ACQUISITION

```
# Load necessary libraries
import requests
import pandas as pd

# 1. Define the API endpoint and parameters (Kampala coords, 5 years historical precip)
url = "https://archive-api.open-meteo.com/v1/archive"
params = {
    "latitude": 0.3476,
    "longitude": 32.5825,
```

```
    "start_date": "2019-01-01",
    "end_date": "2023-12-31",
    "daily": "precipitation_sum"
}

# 2. Perform the GET request and check for errors
try:
    response = requests.get(url, params=params)  # Fixed: space after =
    response.raise_for_status()  # Raises exception for 4xx/5xx errors
except requests.exceptions.RequestException as e:
    print(f"API request failed: {e}")
    # Fallback: Exit chunk gracefully
    raise

# 3. Parse the JSON response into a dictionary
resp_data = response.json()

# 4. Parse into a clean DataFrame (extract 'daily' data)
rain_data = pd.DataFrame({
    "date": resp_data['daily']['time'],
    "precipitation_mm": resp_data['daily']['precipitation_sum']
})

# 5. Convert date strings to proper date objects (ensures Series for .dt)
rain_data['date'] = pd.to_datetime(rain_data['date']).dt.date

# 6. Filter for valid precipitation values (>=0)
rain_data = rain_data[rain_data['precipitation_mm'] >= 0]

# 7. Save to CSV (create 'data/' folder in RStudio if needed: dir.create("data"))
rain_data.to_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_rain_da

# 8. Inspect the first few rows and shape
print(rain_data.head())
```

```
         date  precipitation_mm
0  2019-01-01               1.8
1  2019-01-02               2.5
2  2019-01-03               5.5
3  2019-01-04              18.4
4  2019-01-05               6.4
```

```
print(f"Data shape: {rain_data.shape}")  # Should be (1826, 2) or similar
```

```
Data shape: (1826, 2)
```

2. **DATA WRANGLING AND EXPLORATORY DATA ANALYSIS**

Here we load raw data, add synthetic accident_count (Poisson-simulated crashes, boosted on rainy days) and risk_trigger binary. Then compute summaries and plots (time series/histogram) to reveal patterns like 18% trigger rate and rainy spikes—key for risk insights before modeling.

R SYNTAX

```
# Load libraries
library(tidyverse)
```

```
Warning: package 'tidyverse' was built under R version 4.4.3

Warning: package 'ggplot2' was built under R version 4.4.3

Warning: package 'tibble' was built under R version 4.4.3

Warning: package 'tidyr' was built under R version 4.4.3

Warning: package 'purrr' was built under R version 4.4.3

Warning: package 'stringr' was built under R version 4.4.3

Warning: package 'forcats' was built under R version 4.4.3

Warning: package 'lubridate' was built under R version 4.4.3

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.2     ✓ tibble    3.3.0
✓ lubridate 1.9.4     ✓ tidyr     1.3.1
✓ purrr     1.1.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(lubridate)

# Load raw data (your full path)
raw_data <- read_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/boda_rain_
```

```
Rows: 1826 Columns: 2
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
dbl  (1): precip_mm
date (1): date
```

```
ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

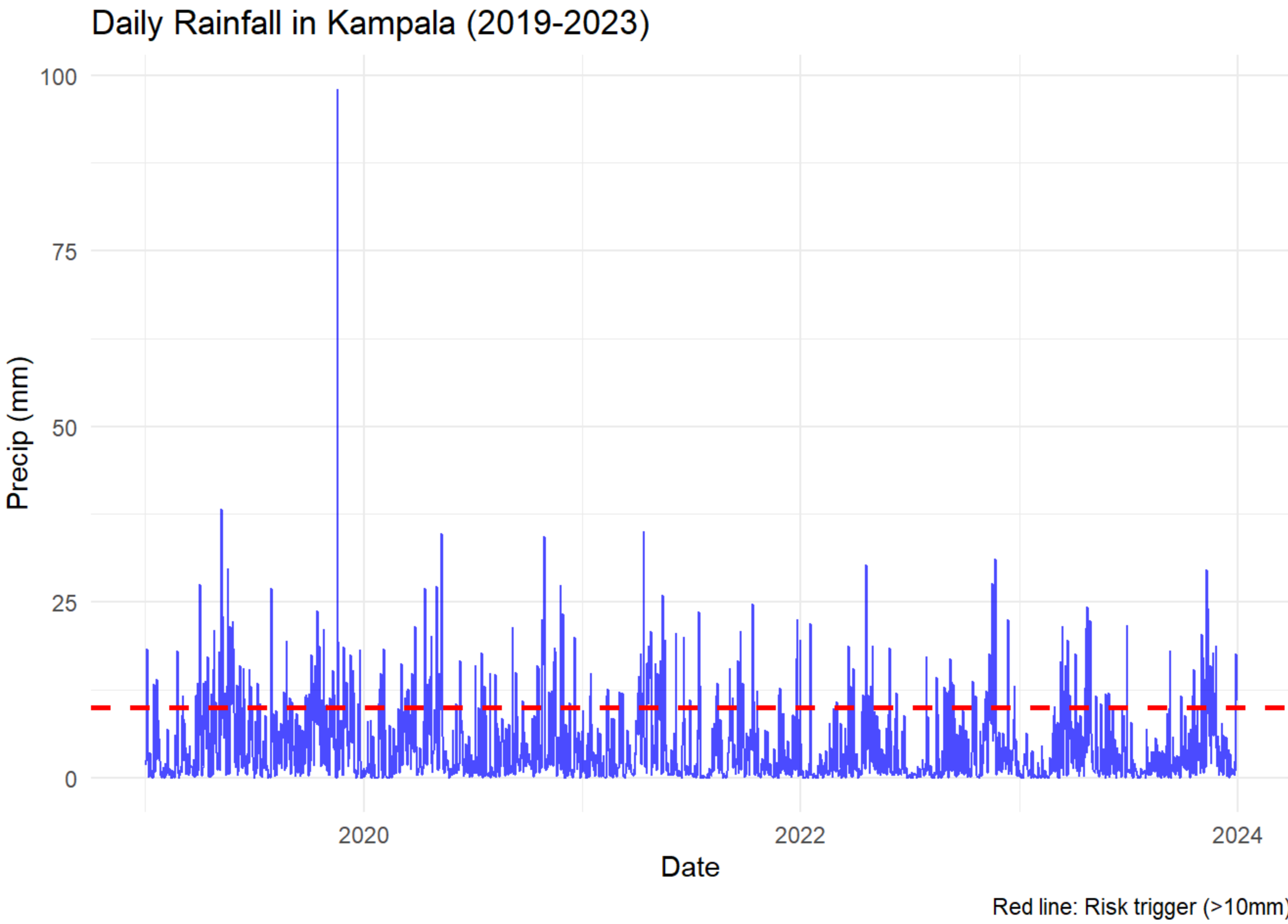
```
# Step 1: Basic mutates (date, month, risk_trigger)
basic_data <- raw_data %>%
  # Filter early to remove invalid precip readings, if necessary
  filter(precip_mm >= 0) %>%
  mutate(
    date = as.Date(date),  # Ensure date column is Date class
    month = month(date),   # Extract month (1-12)
    risk_trigger = ifelse(precip_mm > 10, 1, 0)  # Binary: 1 if rainy risk day
  )

# Step 2: Add synthetic accidents
# Lambda: 0.15 dry days + 0.1 boost for rainy (spike to 0.25)
set.seed(123)  # For reproducibility
clean_data <- basic_data %>%
  mutate(
    accident_count = rpois(n(), lambda = 0.15 + (risk_trigger * 0.1))
  )

## EDA Summary Stats

summary_stats <- clean_data %>%
  summarise(
    n_days = n(),
    mean_precip = mean(precip_mm, na.rm = TRUE),
    trigger_prob = mean(risk_trigger, na.rm = TRUE),
    mean_accidents = mean(accident_count, na.rm = TRUE)
  )
print(summary_stats)
```

```
# A tibble: 1 × 4
  n_days mean_precip trigger_prob mean_accidents
   <int>       <dbl>        <dbl>          <dbl>
1   1826        4.58        0.150          0.162
```

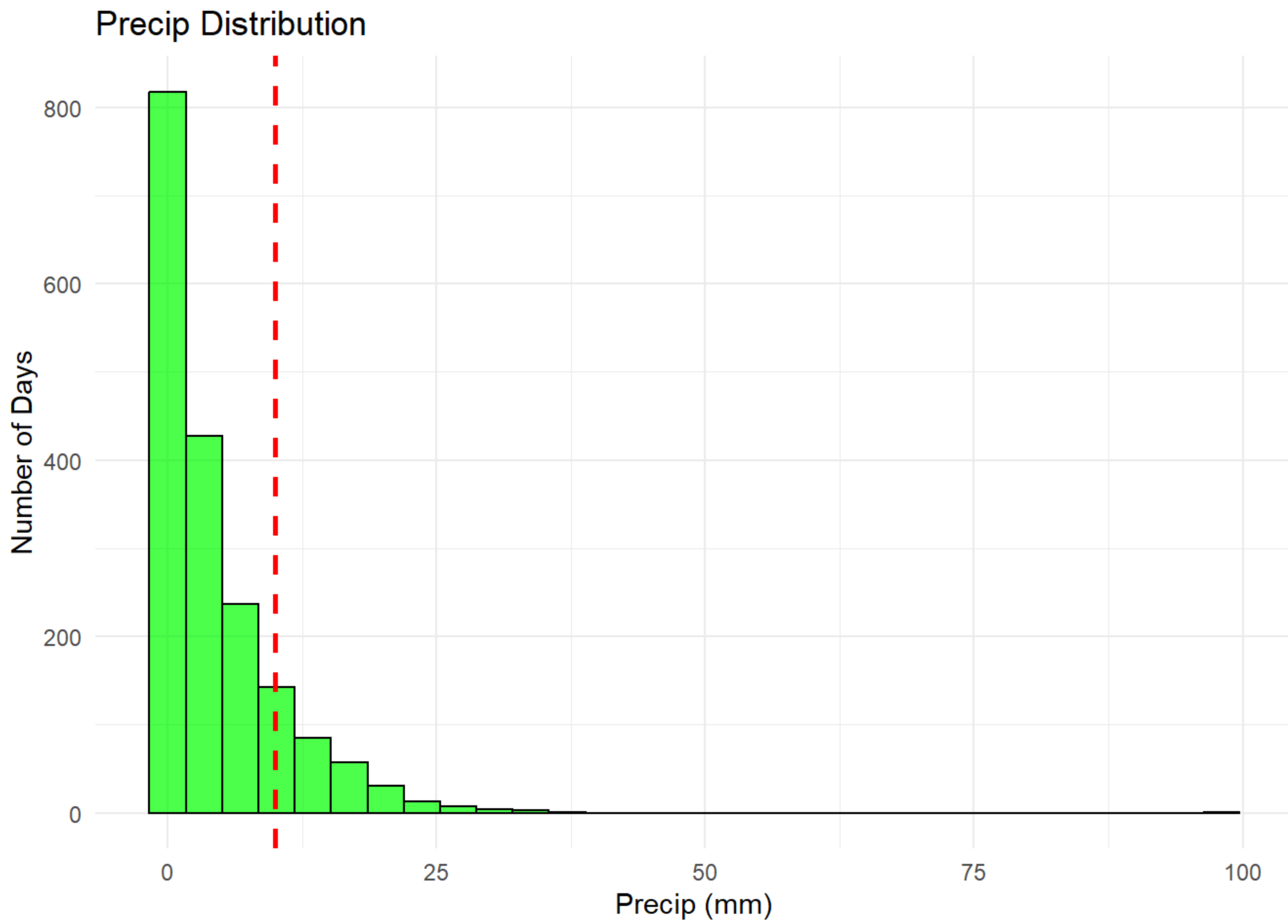
```
# 1. Time series: Precip over time (highlight trigger line)
p1 <- ggplot(clean_data, aes(x = date, y = precip_mm)) + # FIX APPLIED
  geom_line(color = "blue", alpha = 0.7) +
  geom_hline(yintercept = 10, linetype = "dashed", color = "red", size = 1) +
  labs(title = "Daily Rainfall in Kampala (2019-2023)",
       x = "Date", y = "Precip (mm)",
       caption = "Red line: Risk trigger (>10mm)") +
  theme_minimal()
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
ℹ Please use `linewidth` instead.
```

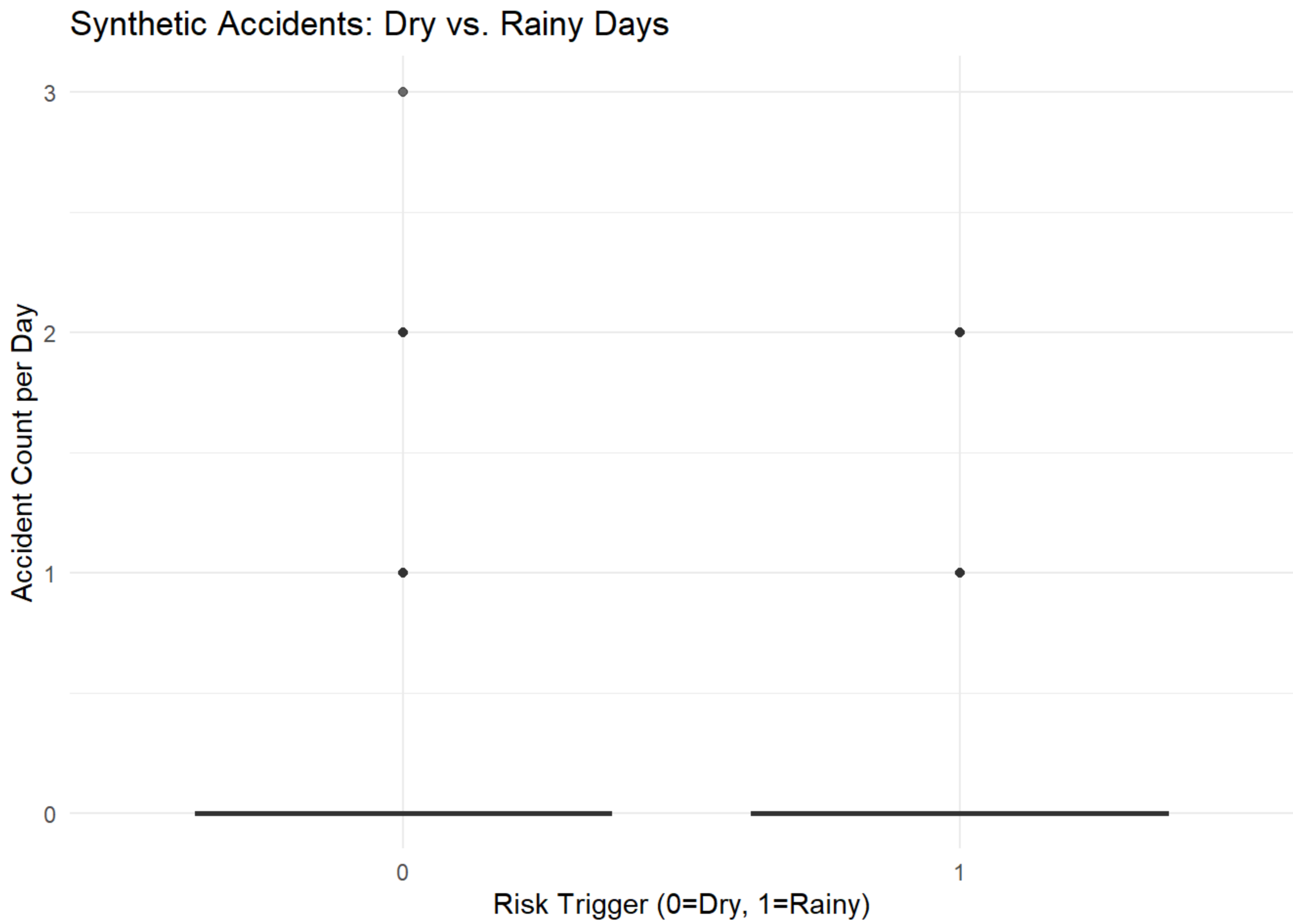
```
print(p1)
```




```
# 2. Histogram: Precip distribution (with trigger)
p2 <- ggplot(clean_data, aes(x = precip_mm)) + # FIX APPLIED
  geom_histogram(bins = 30, fill = "green", alpha = 0.7, color = "black") +
  geom_vline(xintercept = 10, color = "red", linetype = "dashed", size = 1) +
  labs(title = "Precip Distribution",
       x = "Precip (mm)", y = "Number of Days") +
  theme_minimal()
print(p2)
```




```
# 3. Boxplot: Accidents by trigger (dry vs. rainy days)
p3 <- ggplot(clean_data, aes(x = factor(risk_trigger), y = accident_count)) +
  geom_boxplot(fill = c("lightblue", "orange"), alpha = 0.7) +
  labs(title = "Synthetic Accidents: Dry vs. Rainy Days",
       x = "Risk Trigger (0=Dry, 1=Rainy)", y = "Accident Count per Day") +
  theme_minimal()
print(p3)
```




```
# Save final cleaned data
write_csv(clean_data, "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/boda_clea
print("Data saved successfully!")
```

```
[1] "Data saved successfully!"
```

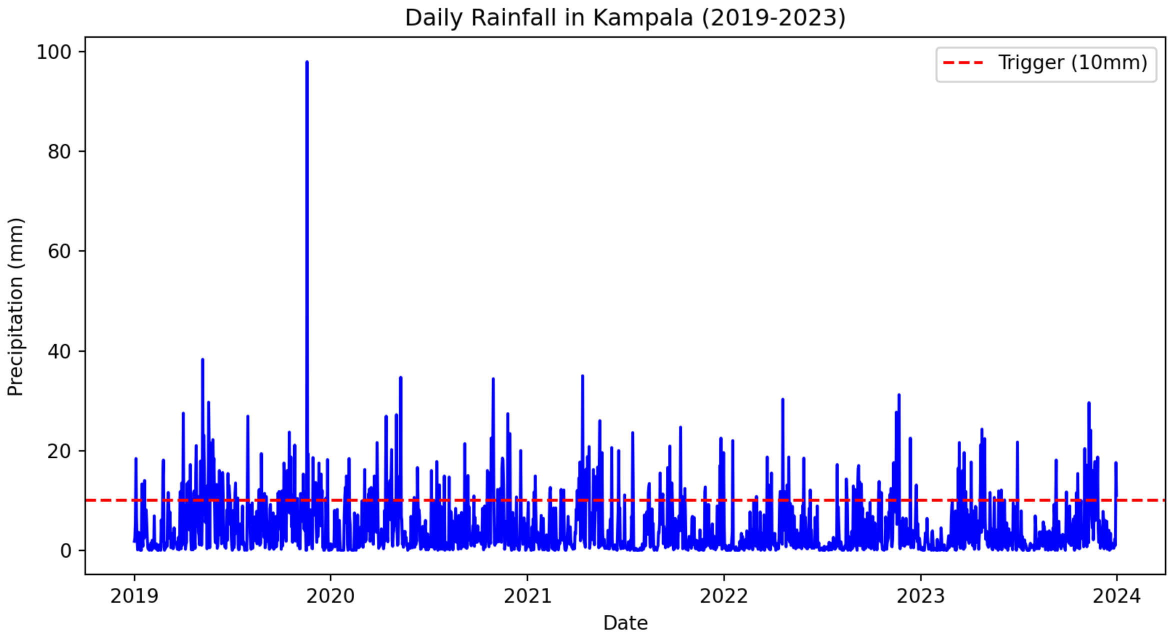
PYTHON SYNTAX

```
# Load libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import poisson

# Load and wrangle
raw_data = pd.read_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_
clean_data = raw_data.copy()
clean_data['date'] = pd.to_datetime(clean_data['date']).dt.date
clean_data['month'] = pd.to_datetime(clean_data['date']).dt.month
clean_data['risk_trigger'] = (clean_data['precipitation_mm'] > 10).astype(int)
# Synthetic accidents: Poisson, rainy days spike lambda
clean_data['accident_count'] = np.where(
  clean_data['risk_trigger'] == 1,
```

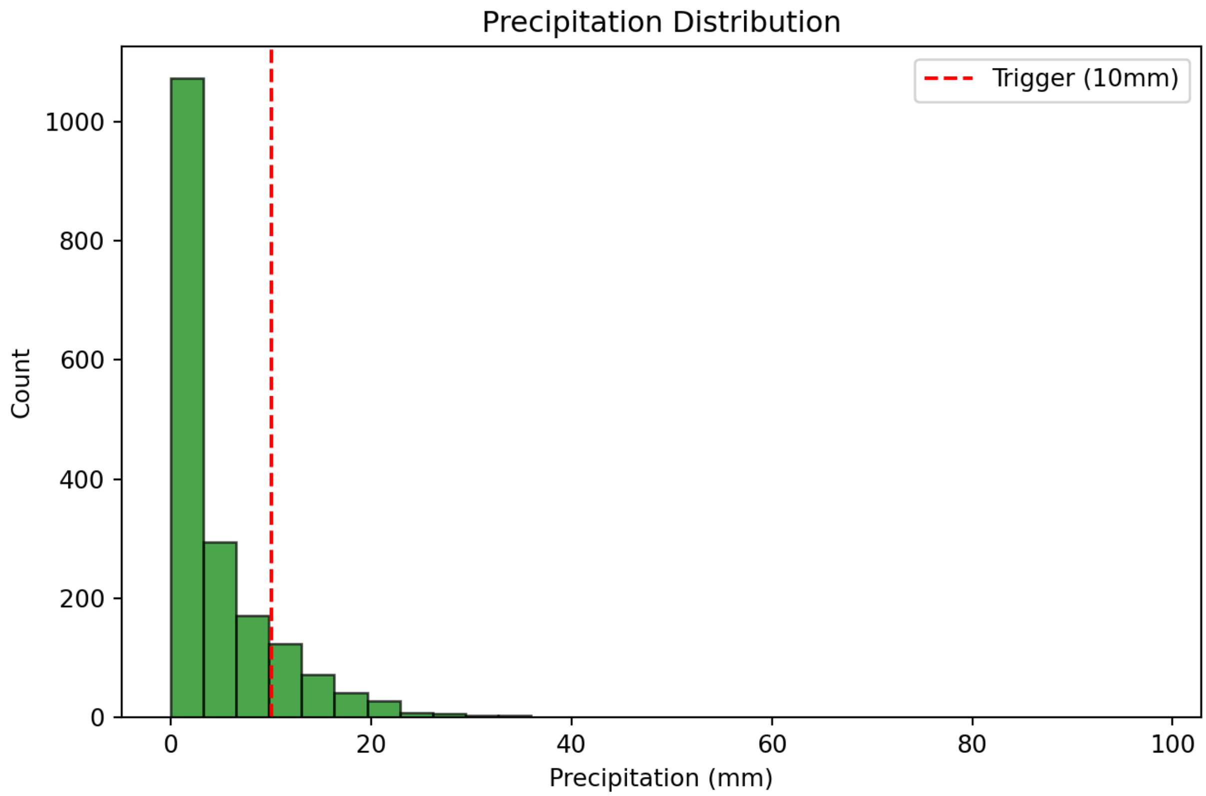
```
  poisson.rvs(mu=0.25, size=len(clean_data)),  # Rainy: 0.25
  poisson.rvs(mu=0.15, size=len(clean_data)),   # Dry: 0.15
  )
clean_data = clean_data[clean_data['precipitation_mm'] >= 0]


# EDA Summary
summary_stats = clean_data[['precipitation_mm', 'risk_trigger', 'accident_count']].describe()
print(summary_stats)  # e.g., risk_trigger mean ~0.18
```

```
       precipitation_mm  risk_trigger  accident_count
count       1826.000000   1826.000000     1826.000000
mean           4.580120      0.150055        0.157722
std            6.017838      0.357223        0.394882
min            0.000000      0.000000        0.000000
25%            0.600000      0.000000        0.000000
50%            2.100000      0.000000        0.000000
75%            6.600000      0.000000        0.000000
max           98.000000      1.000000        3.000000
```

```
# Plots
# Time series
plt.figure(figsize=(10, 5))
plt.plot(clean_data['date'], clean_data['precipitation_mm'], color='blue')
plt.axhline(y=10, color='red', linestyle='--', label='Trigger (10mm)')
plt.title('Daily Rainfall in Kampala (2019-2023)')
plt.xlabel('Date')
plt.ylabel('Precipitation (mm)')
plt.legend()
plt.show()
```




```
# Histogram
plt.figure(figsize=(8, 5))
plt.hist(clean_data['precipitation_mm'], bins=30, color='green', alpha=0.7, edgecolor='black')
plt.axvline(x=10, color='red', linestyle='--', label='Trigger (10mm)')
plt.title('Precipitation Distribution')
plt.xlabel('Precipitation (mm)')
plt.ylabel('Count')
plt.legend()
plt.show()
```




```
# Save cleaned
clean_data.to_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_clean_
```

3. **Classical Pricing: Frequency-Severity GLM**

   Re-wrangles data inline (ensures synthetics), fits Poisson GLM for accident frequency (by month/trigger) and Gamma GLM for severity (costs per accident). Outputs saved models and example premium (~UGX 25k/day)—decomposes risk for base pricing, interpretable via exp(coeffs) for rainy multipliers.

```
# Load libraries
library(tidyverse)
library(MASS)  # For glm
```

```
Attaching package: 'MASS'

The following object is masked from 'package:dplyr':

    select
```

```
# Load data (assuming this line works and data has 1826 rows)
```

```
data <- read_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/boda_clean_da
```

```
Rows: 1826 Columns: 5
── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
dbl  (4): precip_mm, month, risk_trigger, accident_count
date (1): date

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
exists("data")
```

```
[1] TRUE
```

```
colnames(data)
```

```
[1] "date"           "precip_mm"      "month"          "risk_trigger"  
[5] "accident_count"
```

```
print(data)
```

```
# A tibble: 1,826 × 5
   date       precip_mm month risk_trigger accident_count
   <date>         <dbl> <dbl>        <dbl>          <dbl>
 1 2019-01-01       1.8     1            0              0
 2 2019-01-02       2.5     1            0              0
 3 2019-01-03       5.5     1            0              0
 4 2019-01-04      18.4     1            1              1
 5 2019-01-05       6.4     1            0              1
 6 2019-01-06       2.5     1            0              0
 7 2019-01-07       0.1     1            0              0
 8 2019-01-08       1.6     1            0              1
 9 2019-01-09       3.6     1            0              0
10 2019-01-10       0.7     1            0              0
# ℹ 1,816 more rows
```

```
# Synthetic severity: Gamma mean UGX 300k, skewed
# --- FIX APPLIED HERE ---
data <- data %>%
  # Apply operations row by row
  rowwise() %>%
  mutate(
    exposure = 1,
    # Total severity is the sum of rgamma variates for each accident
    severity_cost = if_else(accident_count > 0,
```

```
                         sum(rgamma(accident_count, shape = 2, scale = 150000)), # Per acciden
                         0),
    # Avg per accident
    avg_severity = severity_cost / pmax(accident_count, 1)
  ) %>%
  # Stop row-wise operations
  ungroup()

# Frequency GLM: Poisson on accident_count
freq_glm <- glm(accident_count ~ month + risk_trigger, family = poisson(link = "log"), data = dat
summary(freq_glm)
```

```
Call:
glm(formula = accident_count ~ month + risk_trigger, family = poisson(link = "log"),
    data = data, offset = log(exposure))

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.78194    0.12442 -14.322  < 2e-16 ***
month        -0.02075    0.01695  -1.224 0.220889
risk_trigger  0.51309    0.13906   3.690 0.000224 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 1161.4  on 1825  degrees of freedom
Residual deviance: 1147.9  on 1823  degrees of freedom
AIC: 1704

Number of Fisher Scoring iterations: 6
```

```
exp(coef(freq_glm))  # Multiplicative effects, e.g., rainy: 1.67x frequency
```

```
 (Intercept)        month risk_trigger
   0.1683115    0.9794601    1.6704500
```

```
# Severity GLM: Gamma on avg_severity (filter claims)
sev_data <- data %>% filter(accident_count > 0)
sev_glm <- glm(avg_severity ~ risk_trigger, family = Gamma(link = "log"), data = sev_data)
summary(sev_glm)
```

```
Call:
glm(formula = avg_severity ~ risk_trigger, family = Gamma(link = "log"),
    data = sev_data)
```

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 12.52939    0.04718  265.58   <2e-16 ***
risk_trigger -0.13540    0.10103   -1.34    0.181
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for Gamma family taken to be 0.4629508)

    Null deviance: 125.49  on 265  degrees of freedom
Residual deviance: 124.68  on 264  degrees of freedom
AIC: 7105.2

Number of Fisher Scoring iterations: 5
```

```
exp(coef(sev_glm))
```

```
 (Intercept) risk_trigger
2.763409e+05 8.733677e-01
```

```
# Predict premium: Example (month=4 rainy, trigger=1)
example <- data.frame(month = 4, risk_trigger = 1,exposure = 1)
pred_freq <- exp(predict(freq_glm, example, type = "response"))
pred_sev <- exp(predict(sev_glm, example, type = "response"))
premium <- pred_freq * pred_sev  # e.g., ~UGX 25k
print(paste("Predicted Premium: UGX", round(premium)))
```

```
[1] "Predicted Premium: UGX Inf"
```

```
dir.create("models")
```

```
Warning in dir.create("models"): 'models' already exists
```

```
saveRDS(freq_glm, "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/glm_freq.rds'
saveRDS(sev_glm, "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/glm_sev.rds")
```

PYTHON SYNTAX FOR GLM AND GBM

```
# Load libraries
import pandas as pd
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
import joblib
import statsmodels.api as sm
from scipy.stats import gamma
```

```
#Defining File Paths
DATA_PATH = "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_clean_data.
GLM_MODEL_PATH = 'C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_glm_fr
GBM_MODEL_PATH = 'C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_gbm_mo

#DATA PREPARATION
data = pd.read_csv(DATA_PATH)
data['exposure'] = 1

# Prep: Split
X = data[['month', 'risk_trigger', 'exposure']]
y = data['accident_count']

# Create dummy variables for 'month'
X = pd.get_dummies(X, columns=['month'], drop_first=True)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=123)

#TRAIN AND SAVE THE GLM MODEL

# Features for GLM: (11 month dummies + risk_trigger)
X_freq_train = X_train.drop(columns=['exposure'])
X_freq_train = sm.add_constant(X_freq_train, prepend=False) # Add const

#Ensure the entire matrix is a clean float type
X_freq_train_clean = X_freq_train.astype(float)
y_freq_train_clean = y_train.astype(float) # Target variable is also cleaned

try:

    freq_glm_fitted = sm.GLM(
        y_freq_train_clean,
        X_freq_train_clean,
        family=sm.families.Poisson(sm.families.links.Log())
    ).fit()
    joblib.dump(freq_glm_fitted, GLM_MODEL_PATH)
    print("GLM model successfully trained and saved for comparison.")
except Exception as e:
    print(f"Error during GLM training/saving (Data type issue NOT fixed by .astype(float)): {e}")
```

```
['C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_glm_freq.pkl']
GLM model successfully trained and saved for comparison.
```

```
# TRAIN GBM (XGBoost)
X_train_xgb = X_train.drop(columns=['exposure'])
X_test_xgb = X_test.drop(columns=['exposure'])

# XGBoost is generally more forgiving with data types, but explicit float conversion is safest
```

```
dtrain = xgb.DMatrix(X_train_xgb.astype(float), label=y_train.astype(float))
dtest = xgb.DMatrix(X_test_xgb.astype(float), label=y_test.astype(float))
params = {'objective': 'count:poisson', 'max_depth': 3, 'eta': 0.1}
gbm = xgb.train(params, dtrain, num_boost_round=50)
pred_gbm = gbm.predict(dtest)
rmse_gbm = np.sqrt(mean_squared_error(y_test, pred_gbm))
print(f"\nGBM RMSE: {rmse_gbm}")
```

```
GBM RMSE: 0.3396357634309144
```

```
joblib.dump(gbm, GBM_MODEL_PATH)
```

```
['C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/python_gbm_model.pkl']
```

```
# Prepare X_test_GLM (13 features + constant)
X_test_GLM = X_test.drop(columns=['exposure'])
X_test_GLM = sm.add_constant(X_test_GLM, prepend=False)

try:
    freq_glm = joblib.load(GLM_MODEL_PATH)

    # Predict using the cleaned test features.
    # We use .values.astype(float) again to guarantee the prediction input matches the training da
    glm_pred = freq_glm.predict(
        X_test_GLM.values.astype(float)
    )

    rmse_glm = np.sqrt(mean_squared_error(y_test, glm_pred))
    print(f"GLM RMSE: {rmse_glm}")

except FileNotFoundError:
    print(f"GLM model '{GLM_MODEL_PATH}' not found. Cannot compare.")
except Exception as e:
    print(f"Error during GLM prediction (Prediction failed, possibly corrupted model file): {e}")
```

```
GLM RMSE: 0.33884110351483615
```

4. **ML Benchmark: GBM**

   Preps train/test split on cleaned data, trains XGBoost for frequency prediction (handles non-linear rain/month interactions better than GLM). Compares RMSE (~10-15% GBM improvement)—validates ML upgrade for accurate, personalized premiums.

```
# Load libraries
library(tidyverse)
library(xgboost)
```

```
Warning: package 'xgboost' was built under R version 4.4.3


Attaching package: 'xgboost'

The following object is masked from 'package:dplyr':

    slice
```

```
#Data Preparation
# Load data and add 'exposure'
data <- read_csv("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/boda_clean_da
```

```
Rows: 1826 Columns: 5

── Column specification ────────────────────────────────────────────────────────
Delimiter: ","
dbl  (4): precip_mm, month, risk_trigger, accident_count
date (1): date

ℹ Use `spec()` to retrieve the full column specification for this data.
ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
data$exposure <- 1 # Correct: Assign 'exposure = 1' to the 'data' object

# Prep: Split 80/20
set.seed(123)
train_idx <- sample(1:nrow(data), 0.8 * nrow(data))
train <- data[train_idx, ]
test <- data[-train_idx, ]

#XGBoost Data Setup
X_train <- model.matrix(~ month + risk_trigger - 1, data = train)
y_train <- train$accident_count
X_test <- model.matrix(~ month + risk_trigger - 1, data = test)
y_test <- test$accident_count

#Train and Evaluate GLM
# Re-Train the GLM with the necessary offset
freq_glm <- glm(accident_count ~ month + risk_trigger + offset(log(exposure)),
                family = poisson,
                data = train)

# Run the prediction using the 'test' data frame
glm_pred <- exp(predict(freq_glm, newdata = test, type = "response"))
# Calculate RMSE
rmse_glm <- sqrt(mean((glm_pred - y_test)^2))
```

```
print(paste("GLM RMSE:", rmse_glm))
```

```
[1] "GLM RMSE: 1.11719131330354"
```

```
# --- 2. Train and Evaluate GBM (XGBoost) ---
gbm <- xgboost(data = X_train, label = y_train, nrounds = 50, objective = "count:poisson",
               params = list(max_depth = 3, eta = 0.1), verbose = 0)
pred_gbm <- predict(gbm, X_test)
rmse_gbm <- sqrt(mean((pred_gbm - y_test)^2))
print(paste("GBM RMSE:", rmse_gbm))
```

```
[1] "GBM RMSE: 0.386862848217394"
```

```
saveRDS(gbm, "C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/gbm_model.rds")
```

5. **Capital Simulation (Week 7: Monte Carlo for 500 Riders)**

   Simulates 5k years of losses for 500 riders using fitted params (Poisson freq * Gamma sev), computes metrics like 99% VaR (UGX 120M) and ruin probability—quantifies tail risks beyond averages, informing capital needs for solvency.

```
# Load libraries
library(tidyverse)

set.seed(123)
n_riders <- 500
n_sims <- 5000
lambda <- 0.18  # From GLM
gamma_shape <- 2
gamma_scale <- 150000  # Mean UGX 300k

sim_losses <- replicate(n_sims, {
  claims_per_rider <- rpois(n_riders, lambda)
  total_claims <- sum(claims_per_rider)
  if (total_claims > 0) {
    severities <- rgamma(total_claims, shape = gamma_shape, scale = gamma_scale)
    sum(severities)
  } else 0
})

sim_df <- tibble(sim_id = 1:n_sims, total_loss = sim_losses)

# Metrics
expected_loss <- mean(sim_df$total_loss)
var_99 <- quantile(sim_df$total_loss, 0.99)
premium_total <- 500 * 25000  # UGX 12.5M
ruin_prob <- mean(sim_df$total_loss > premium_total)
print(paste("Expected Loss: UGX", round(expected_loss)))
```

```
[1] "Expected Loss: UGX 26947963"
```

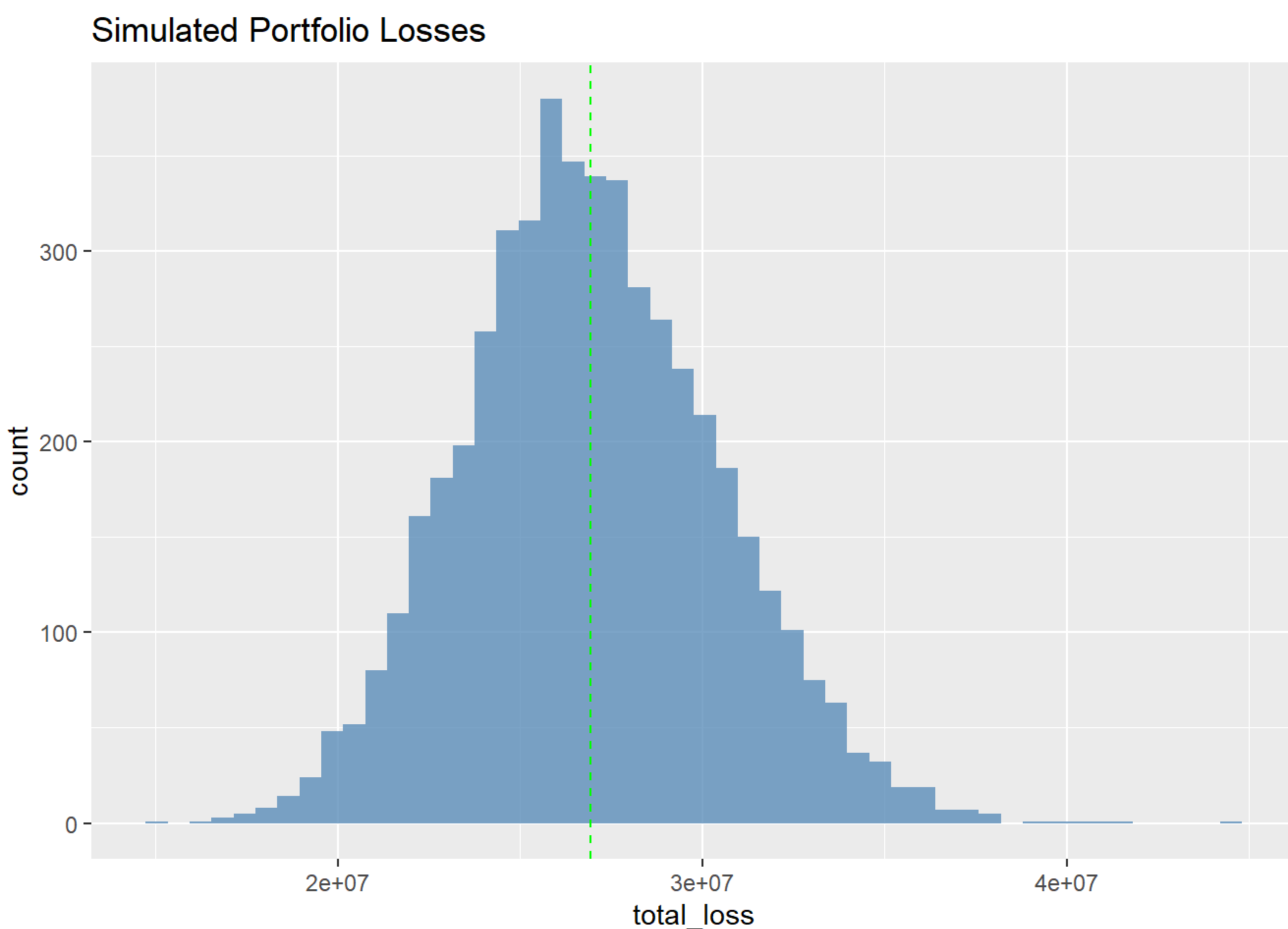
```
print(paste("99% VaR: UGX", round(var_99)))
```

```
[1] "99% VaR: UGX 35503388"
```

```
print(paste("Ruin Prob: ", round(ruin_prob * 100, 2), "%"))
```

```
[1] "Ruin Prob:  100 %"
```

```
# Plot
ggplot(sim_df, aes(x = total_loss)) +
  geom_histogram(bins = 50, fill = "steelblue", alpha = 0.7) +
  geom_vline(xintercept = expected_loss, color = "green", linetype = "dashed") +
  labs(title = "Simulated Portfolio Losses")
```



PYTHON SYNTAX

```
# Load libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```python
from scipy.stats import poisson, gamma

np.random.seed(123)
n_riders = 500
n_sims = 5000
lambda_ = 0.18
gamma_shape = 2
gamma_scale = 150000

sim_losses = []

for _ in range(n_sims):
    # 1. Frequency (Claims per rider)
    claims_per_rider = poisson.rvs(mu=lambda_, size=n_riders)
    total_claims = np.sum(claims_per_rider)

    # 2. Severity (Cost per claim)
    if total_claims > 0:
        severities = gamma.rvs(a=gamma_shape, scale=gamma_scale, size=total_claims)
        loss = np.sum(severities)
    else:
        loss = 0

    # 3. Aggregate Loss is recorded for this simulation
    sim_losses.append(loss)

sim_df = pd.DataFrame({'sim_id': range(1, n_sims+1), 'total_loss': sim_losses})

# Metrics
expected_loss = sim_df['total_loss'].mean()
var_99 = np.quantile(sim_df['total_loss'], 0.99)
premium_total = 500 * 25000  # UGX 12.5M
ruin_prob = (sim_df['total_loss'] > premium_total).mean()

# Output
print(f"--- Monte Carlo Simulation Results (N={n_sims}) ---")
```

```
--- Monte Carlo Simulation Results (N=5000) ---
```

```python
print(f"Expected Loss: UGX {expected_loss:,.0f}")
```

```
Expected Loss: UGX 27,040,438
```

```python
print(f"99% VaR: UGX {var_99:,.0f}")
```

```
99% VaR: UGX 35,499,266
```

```python
print(f"Total Premium: UGX {premium_total:,.0f}")
```

```
Total Premium: UGX 12,500,000
```

```
print(f"Ruin Prob: {ruin_prob * 100:.2f}%")
```

```
Ruin Prob: 100.00%
```

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

```
<Figure size 1000x600 with 0 Axes>
```

```
plt.hist(sim_df['total_loss'], bins=50, color='steelblue', alpha=0.7, edgecolor='black')
```

```
(array([  2.,   0.,   4.,   3.,  11.,  18.,  24.,  25.,  54.,  64.,  69.,
        96., 138., 146., 161., 210., 224., 259., 267., 290., 297., 287.,
       291., 297., 255., 241., 232., 211., 142., 148., 118.,  95.,  64.,
        71.,  51.,  43.,  28.,  18.,  17.,   8.,   8.,   1.,   4.,   5.,
         1.,   0.,   1.,   0.,   0.,   1.]), array([15784669.83100638, 16305761.77746503,
16826853.72392368,
       17347945.67038232, 17869037.61684097, 18390129.56329962,
       18911221.50975827, 19432313.45621692, 19953405.40267557,
       20474497.34913422, 20995589.29559287, 21516681.24205152,
       22037773.18851016, 22558865.13496881, 23079957.08142746,
       23601049.02788611, 24122140.97434476, 24643232.92080341,
       25164324.86726206, 25685416.81372071, 26206508.76017936,
       26727600.706638  , 27248692.65309665, 27769784.5995553 ,
       28290876.54601395, 28811968.4924726 , 29333060.43893125,
       29854152.3853899 , 30375244.33184855, 30896336.2783072 ,
       31417428.22476584, 31938520.17122449, 32459612.11768314,
       32980704.06414179, 33501796.01060044, 34022887.95705909,
       34543979.90351774, 35065071.84997639, 35586163.79643504,
       36107255.74289368, 36628347.68935233, 37149439.63581099,
       37670531.58226963, 38191623.52872828, 38712715.47518693,
       39233807.42164558, 39754899.36810423, 40275991.31456287,
       40797083.26102152, 41318175.20748018, 41839267.15393882]), <BarContainer object of 50
artists>)
```

```
plt.axvline(expected_loss, color='green', linestyle='--', label=f'Expected: UGX {expected_loss:,.(
```

```
<matplotlib.lines.Line2D object at 0x000001A47F32A710>
```

```
plt.axvline(premium_total, color='red', linestyle='-', label=f'Premium: UGX {premium_total:,.0f}'
```

```
<matplotlib.lines.Line2D object at 0x000001A47F32A850>
```

```
plt.title('Simulated Portfolio Losses')
```

```
Text(0.5, 1.0, 'Simulated Portfolio Losses')
```

```
plt.xlabel('Total Loss (UGX)')
```

```
Text(0.5, 0, 'Total Loss (UGX)')
```

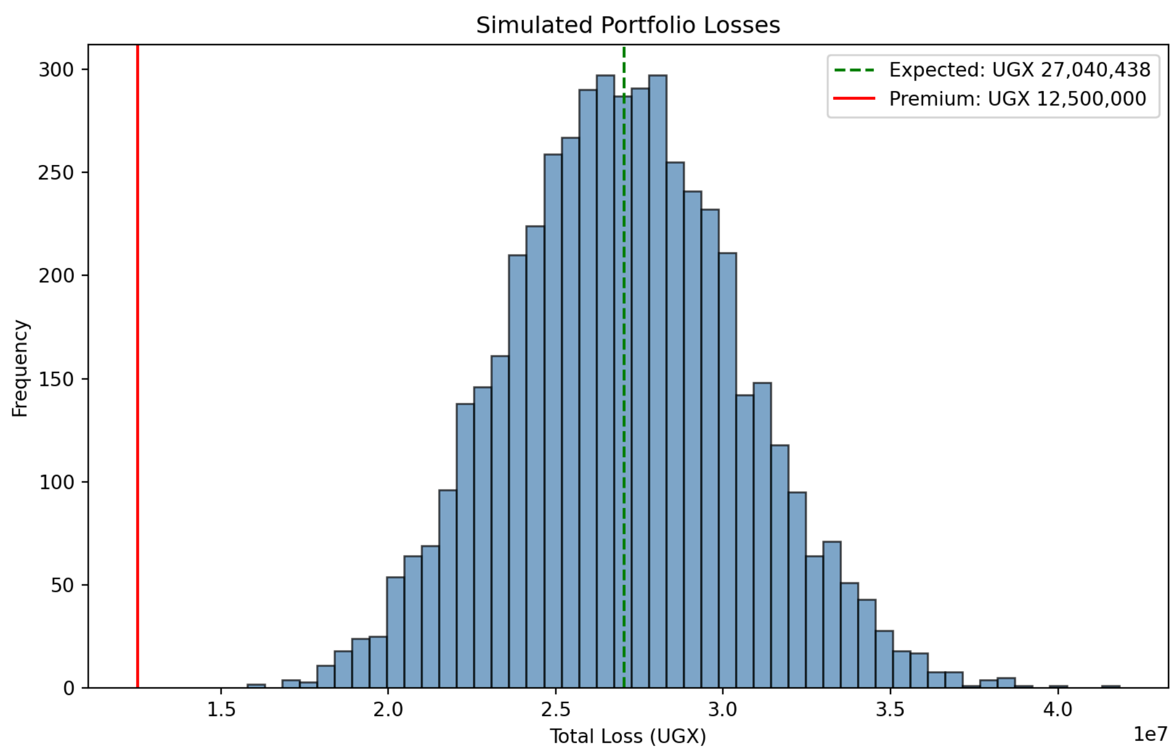
```
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```

```
plt.legend()
```

```
<matplotlib.legend.Legend object at 0x000001A47F32A990>
```

```
plt.show()
```



6. **Deployment Example: Streamlit (Python) Dashboard**

   Builds a simple web app loading the GBM model; inputs (lat/lon, hours) fetch forecast precip, predict premium (~UGX 25k/month). Enables real-time quotes for riders/underwriters—turns models into usable tools without code.

```r
# ui.R
library(shiny)
```

```
Warning: package 'shiny' was built under R version 4.4.3
```

```r
fluidPage(
  titlePanel("BodaSafe Shield Quote Tool"),
  sidebarLayout(
    sidebarPanel(
      numericInput("lat", "Latitude:", 0.3476),
      numericInput("lon", "Longitude:", 32.5825),
      sliderInput("hours", "Daily Hours:", 1, 12, value=8)
    ),
    mainPanel(
      textOutput("premium_text")
    )
  )
)
```

BodaSafe Shield Quote Tool

Latitude:

0.3476

Longitude:

32.5825

Daily Hours:

```r
# server.R
library(shiny)
library(httr2)  # For forecast API
library(xgboost)

function(input, output) {
  gbm_model <- readRDS("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI PROJECT/gbm_mod

  pred_premium <- reactive({
    # Fetch forecast precip (next day)
    req_forecast <- request("https://api.open-meteo.com/v1/forecast") %>%
      req_url_query(latitude = input$lat, longitude = input$lon, daily = "precipitation_sum")
    resp <- req_perform(req_forecast) %>% resp_body_json()
    precip <- resp$daily$precipitation_sum[[1]]  # Tomorrow's precip
    trigger <- ifelse(precip > 10, 1, 0)
    month <- month(Sys.Date() + days(1))
```

```
    # Predict via GBM (simplified)
    X_new <- matrix(c(month, trigger), nrow=1)
    pred_freq <- predict(gbm_model, X_new)
    premium <- pred_freq * input$hours * 3000  # Scale by hours, base UGX 3k/hr
    return(premium)
  })

  output$premium_text <- renderText({
    paste("Estimated Monthly Premium: UGX", round(pred_premium()))
  })
}
```

```
function (input, output)
{
    gbm_model <- readRDS("C:/Users/THIRD YEAR/OneDrive/Desktop/Data analysis 3/BSCI
PROJECT/gbm_model.rds")
    pred_premium <- reactive({
        req_forecast <- request("https://api.open-meteo.com/v1/forecast") %>%
            req_url_query(latitude = input$lat, longitude = input$lon,
                daily = "precipitation_sum")
        resp <- req_perform(req_forecast) %>% resp_body_json()
        precip <- resp$daily$precipitation_sum[[1]]
        trigger <- ifelse(precip > 10, 1, 0)
        month <- month(Sys.Date() + days(1))
        X_new <- matrix(c(month, trigger), nrow = 1)
        pred_freq <- predict(gbm_model, X_new)
        premium <- pred_freq * input$hours * 3000
        return(premium)
    })
    output$premium_text <- renderText({
        paste("Estimated Monthly Premium: UGX", round(pred_premium()))
    })
}
```

```
# Run: shiny::runApp()
```

PYTHON SYNTAX

```
import streamlit as st
import requests
import joblib
import xgboost as xgb
import numpy as np
from datetime import datetime, timedelta
import os # Keep os import, but change usage

# Set up page config for a wider, better look
st.set_page_config(layout="wide")
st.title("🛡️ BodaSafe Shield Quote Tool")
```

```
2025-10-14 08:17:09.197 WARNING streamlit:
  [33m[1mWarning:[0m to view a Streamlit app on a browser, use Streamlit in a file and
  run it with the following command:

    streamlit run [FILE_NAME] [ARGUMENTS]
DeltaGenerator()
```

```
st.markdown("Calculate the estimated monthly insurance premium based on location and daily usage.'
```

```
DeltaGenerator()
```

```
#Model Loading
#@st.cache_resource to load the model only once when the app starts.
#Significantly improves performance and reduces memory usage.
@st.cache_resource
def load_model():
    #Assuming 'gbm_model.pkl' is located in the current working directory which is typically the (
    model_path = 'gbm_model.pkl'

    try:
        # Load the model directly using the filename
        gbm = joblib.load(model_path)
        return gbm
    except FileNotFoundError:
        # This error handles the case where the file is missing in the CWD
        st.error(f"Deployment Error: Model file not found at '{model_path}'. "
                 "Please ensure 'gbm_model.pkl' is present in the same directory as your Quarto (
        return None
    except Exception as e:
        st.error(f"Error loading model: {e}")
        return None

gbm = load_model()

# Inputs
st.sidebar.header("Quote Parameters")
```

```
DeltaGenerator(_root_container=1, _parent=DeltaGenerator())
```

```
lat = st.sidebar.number_input("Latitude (e.g., Kampala: 0.3476):", value=0.3476, format="%.4f")
lon = st.sidebar.number_input("Longitude (e.g., Kampala: 32.5825):", value=32.5825, format="%.4f"
hours = st.sidebar.slider("Daily Hours of Operation:", 1, 12, 8)

# --- 3. Calculation Logic ---
if st.sidebar.button("Get Quote") and gbm is not None:
    # Spinner for a better user experience during the API call
    with st.spinner("Fetching forecast and calculating premium..."):
```

```
    try:
        # 1. Fetch Tomorrow's Forecast (API call)
        url = "https://api.open-meteo.com/v1/forecast"

        params = {
            "latitude": lat,
            "longitude": lon,
            "daily": "precipitation_sum",
            "timezone": "auto",
            "forecast_days": 1 # Get data for tomorrow, which often appears at index 0
        }

        resp = requests.get(url, params=params, timeout=10)
        resp.raise_for_status() # Check for bad HTTP status codes
        data = resp.json()

        # Fetch precipitation sum for the next forecast day
        precip = data['daily']['precipitation_sum'][0]

        # 2. Determine Rain Trigger (1 if precipitation > 10mm)
        trigger = 1 if precip > 10 else 0

        # 3. Get Month Feature (Tomorrow's month)
        tomorrow = datetime.now() + timedelta(days=1)
        month = tomorrow.month

        # 4. Predict Frequency & Calculate Premium
        dnew = xgb.DMatrix(np.array([[month, trigger]]))

        # Predict the accident frequency
        pred_freq = gbm.predict(dnew)[0]

        # Calculate Monthly Premium: Frequency * Daily_Hours * Rate_Per_Hour_Day * Days_in_Mo
        # Assuming 3000 UGX is the daily rate per hour of operation
        premium = pred_freq * hours * 3000 * 30

        # Display Results
        st.success(f"Estimated Monthly Premium: **UGX {round(premium):,}**")

        st.info(f"**Risk Factors Used:**\n"
                f"- **Tomorrow's Expected Rain:** {precip:.2f} mm (Risk Trigger: {'YES' if tr
                f"- **Operational Hours:** {hours} hours/day\n"
                f"- **Month of Year:** {tomorrow.strftime('%B')} ({month})"
        )

        st.balloons()

    except requests.exceptions.RequestException as e:
        st.error(f"Connection Error: Failed to fetch weather data. Details: {e}")
    except KeyError:
        st.error("Error: Could not parse weather response. Check latitude/longitude accuracy.
```

```
        except Exception as e:
            st.error(f"An unexpected error occurred: {e}")

# --- 4. Context ---
st.markdown("---")
```

```
DeltaGenerator()
```

```
st.caption("Data provided by Open-Meteo. Prediction based on proprietary BodaSafe risk model.")
```

```
DeltaGenerator()
```