機器學習與大數據分析技術

Data Analysis and Prediction Using Linear Regression

羅勻瑄、黃世君、劉定睿

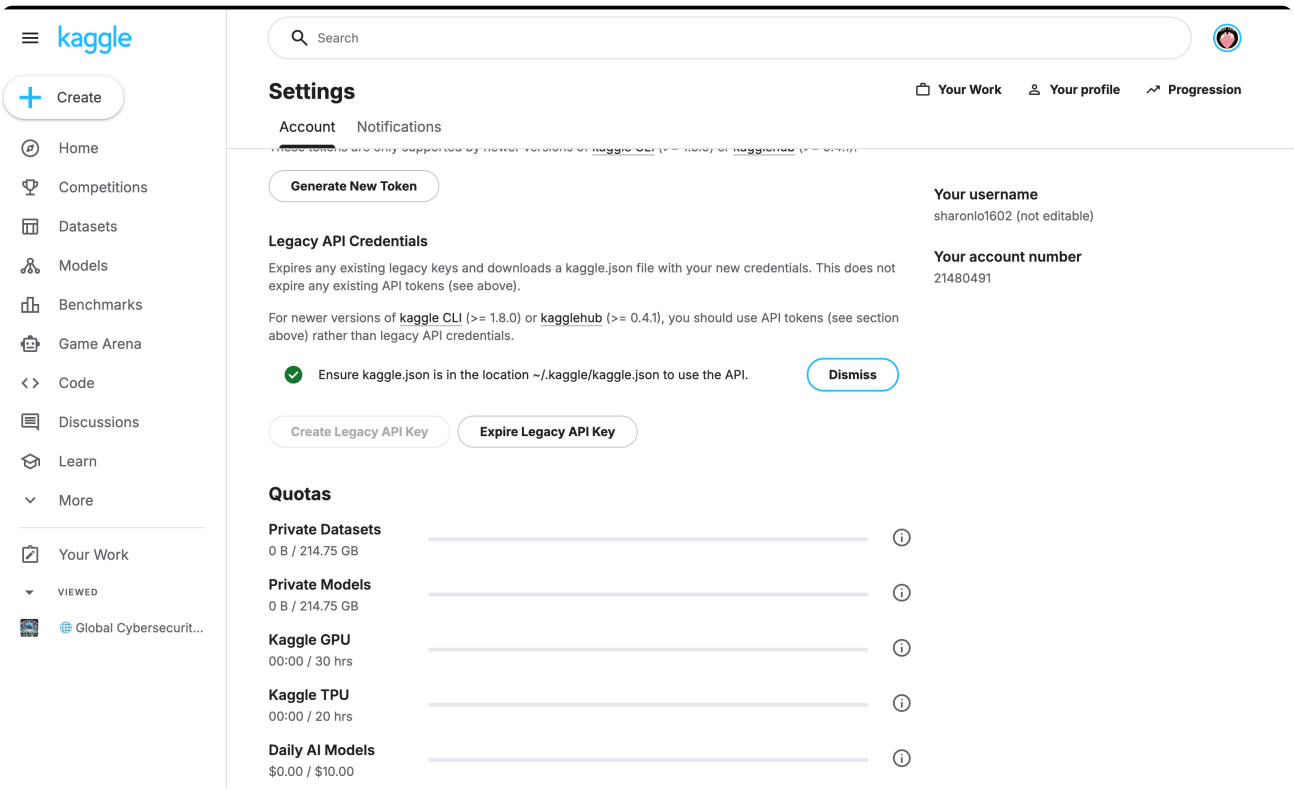
日期：2026-04-02

目錄

實作步驟

1. 去 Kaggle 下載一個數值型資料集

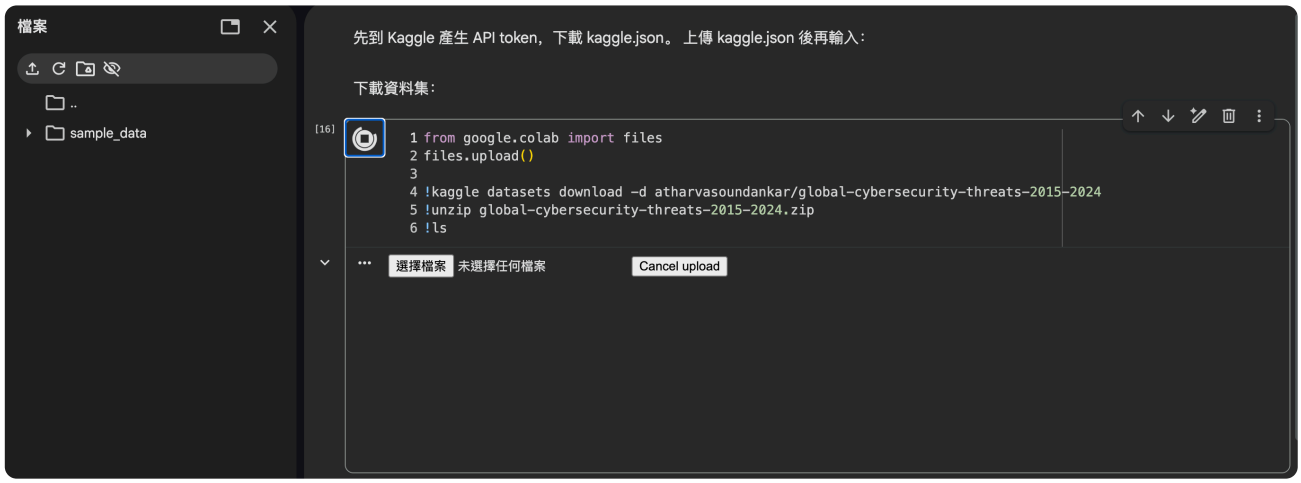
> Kaggel -> 設定 -> Legacy API Credentials
>
> 點擊 Create Legacy API Key 即可產生 kaggel.json 並上傳到colab 做使用



透過以下程式來上傳 kaggle.json 檔

```
from google.colab import files
files.upload()
```

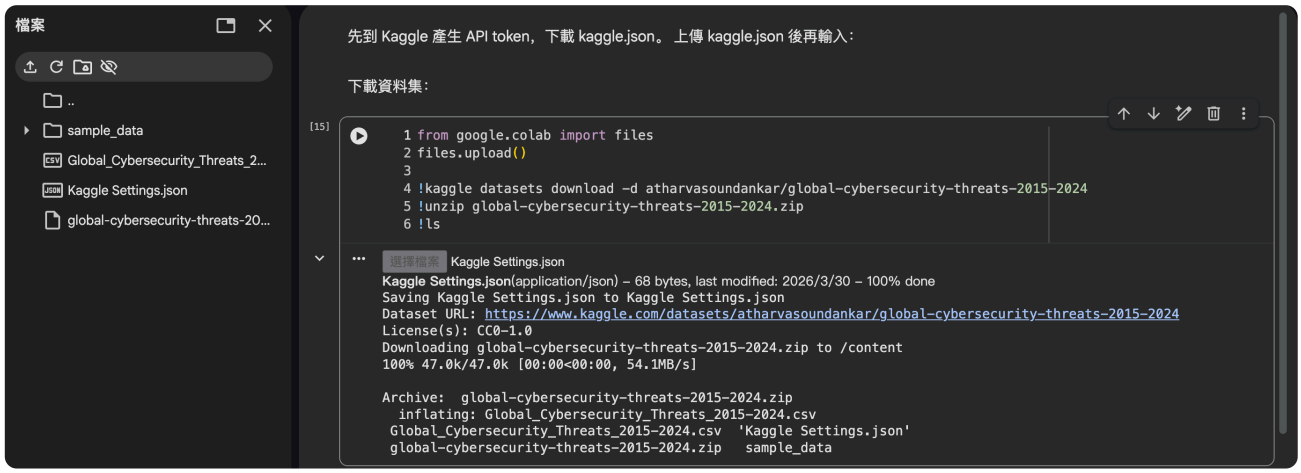
點擊執行後可以點選選擇檔案來上傳 kaggle.json



上傳成功後，後續的程式會下載指定的 kaggle dataset 並解壓縮

```
!kaggle datasets download -d atharvasoundankar/global-cybersecurity-t
!unzip global-cybersecurity-threats-2015-2024.zip
!ls
```

整段程式跑完後可以在左邊檔案的地方看到成功新增需要使用的資料集名稱與kaggle.json等檔案



2. 做 EDA（探索式資料分析），而且至少要畫 3 種不同的圖

- import 套件，資料分析所需包含數學統計相關套件、製圖套件

```
# 做EDA（探索式資料分析），而且至少要畫 3 種不同的圖
# ============================================================
# Step 2: 匯入套件
# ============================================================
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import numpy as np
import warnings

warnings.filterwarnings("ignore")

sns.set_theme(style='darkgrid', palette='muted')
plt.rcParams['figure.dpi'] = 120
plt.rcParams['font.size'] = 11
```

- 初步先進行資料檢查，除了確認匯入成功，也先行做簡單基礎統計與檢查缺失，避免後續做巨量資料處理時出錯還需要收斂錯誤。

```
=== 資料形狀 ===
Rows: 3,000  |  Columns: 10

=== 欄位與型別 ===
Country                                object
Year                                    int64
Attack Type                            object
Target Industry                        object
Financial Loss (in Million $)         float64
Number of Affected Users                int64
Attack Source                          object
Security Vulnerability Type            object
Defense Mechanism Used                 object
Incident Resolution Time (in Hours)     int64
dtype: object
```

=== 前 5 筆 ===

	Country	Year	Attack Type	Target Industry	Financial Loss (in Million $)	Number of Affected Users	Attack Source	Security Vulnerability Type	Defense Mechanism Used	Incident Resolution Time (in Hours)
0	China	2019	Phishing	Education	80.53	773169	Hacker Group	Unpatched Software	VPN	63
1	China	2019	Ransomware	Retail	62.19	295961	Hacker Group	Unpatched Software	Firewall	71
2	India	2017	Man-in-the-Middle	IT	38.65	605895	Hacker Group	Weak Passwords	VPN	20
3	UK	2024	Ransomware	Telecommunications	41.44	659320	Nation-state	Social Engineering	AI-based Detection	7
4	Germany	2018	Man-in-the-Middle	IT	74.41	810682	Insider	Social Engineering	VPN	68

=== 基礎統計 ===

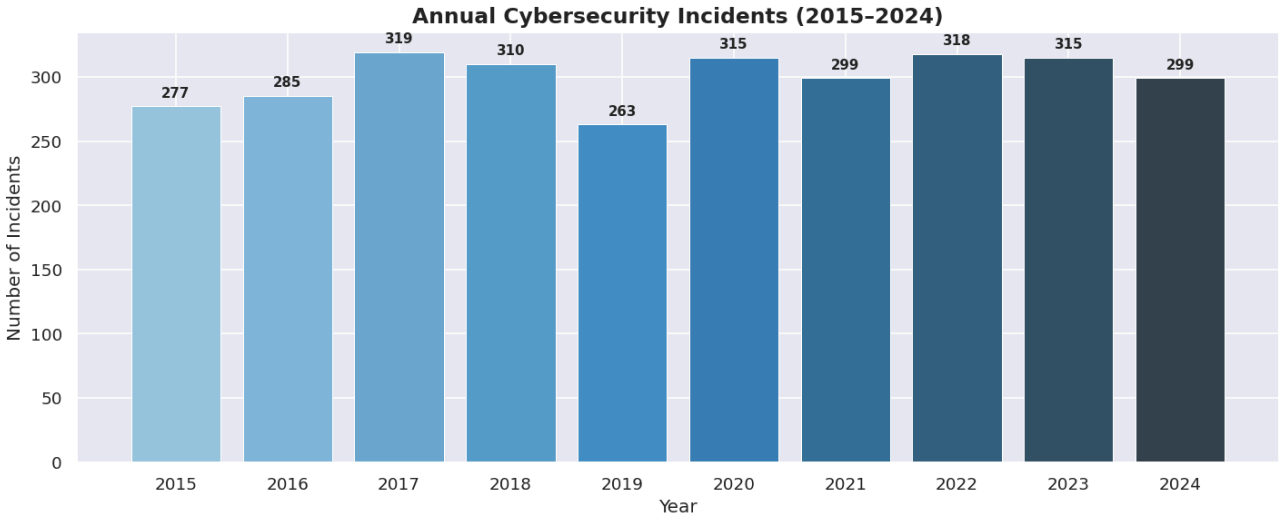
	Country	Year	Attack Type	Target Industry	Financial Loss (in Million $)	Number of Affected Users	Attack Source	Security Vulnerability Type	Defense Mechanism Used	Incident Resolution Time (in Hours)
count	3000	3000.000000	3000	3000	3000.000000	3000.000000	3000	3000	3000	3000.000000
unique	10	NaN	6	7	NaN	NaN	4	4	5	NaN
top	UK	NaN	DDoS	IT	NaN	NaN	Nation-state	Zero-day	Antivirus	NaN
freq	321	NaN	531	478	NaN	NaN	794	785	628	NaN
mean	NaN	2019.570333	NaN	NaN	50.492970	504684.136333	NaN	NaN	NaN	36.476000
std	NaN	2.857932	NaN	NaN	28.791415	289944.084972	NaN	NaN	NaN	20.570768
min	NaN	2015.000000	NaN	NaN	0.500000	424.000000	NaN	NaN	NaN	1.000000

=== 缺失值統計 ===

	Missing Count	Missing %

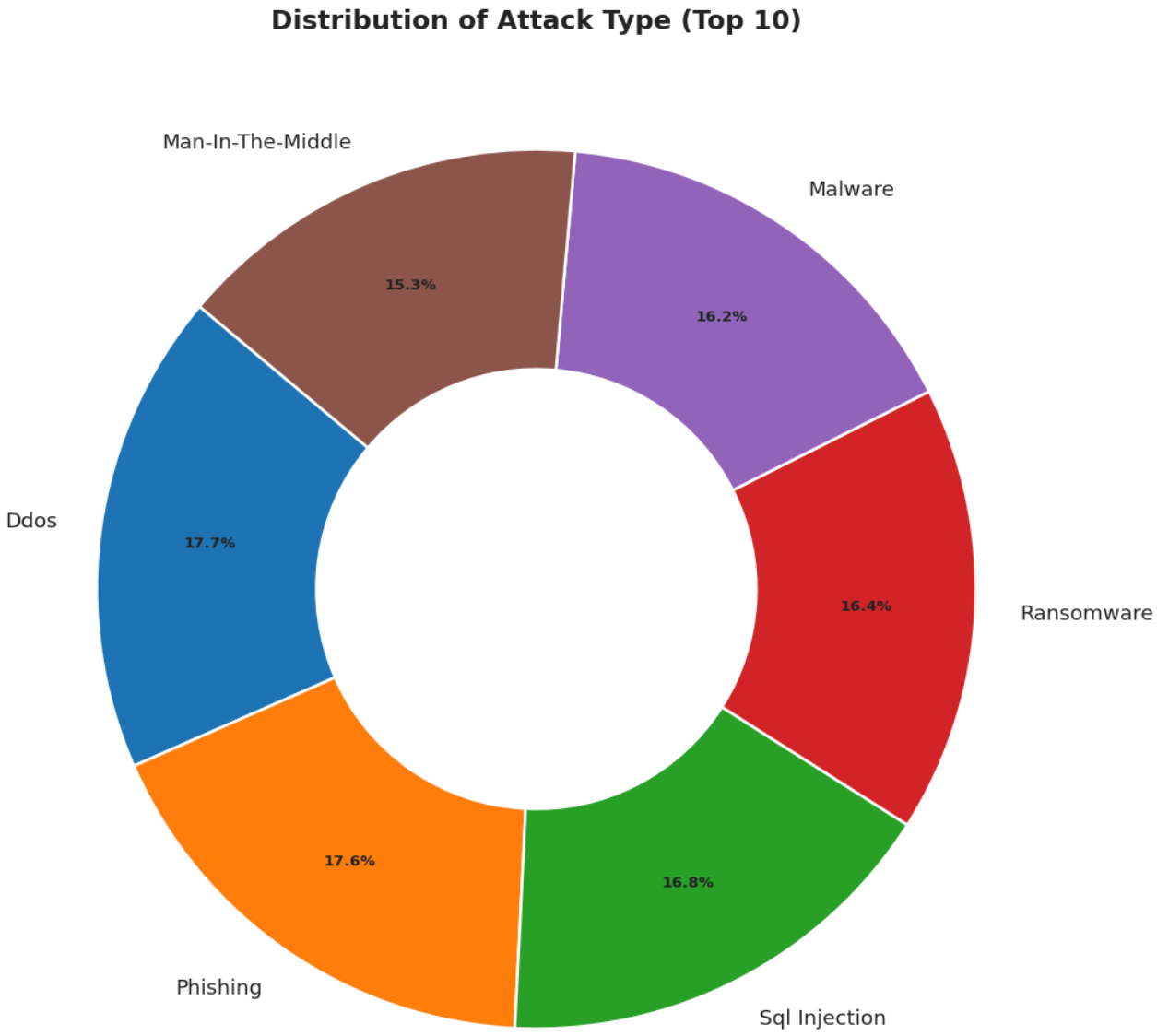
長條圖

長條圖評估了這幾年的網路攻擊趨勢，綜合了所有類型的攻擊類型，可以發現整體來看沒有差太多，但 2020~2022 有略高一點，可能與疫情的在家上班有關，不過無法得知資料集的爬取來源，無法果斷推斷直接原因。

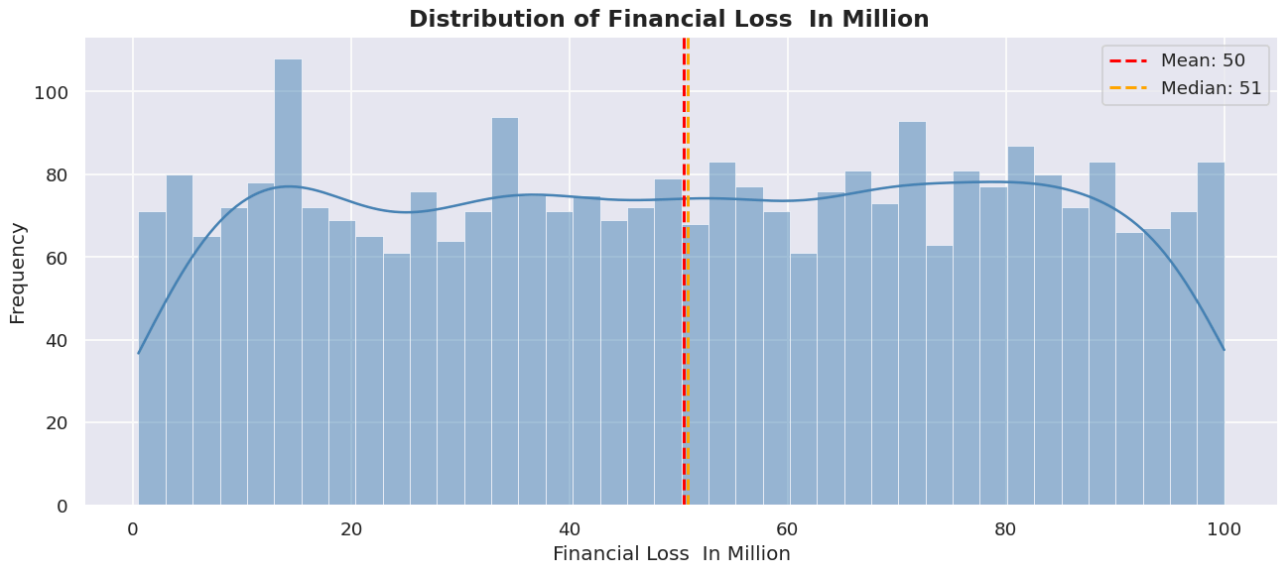


圓餅圖

利用圓餅圖發現所有類型的攻擊分佈都差不多，這可能也並不代表實際上每種類型的攻擊次數就一定差不多，而是取決於資料集來源。

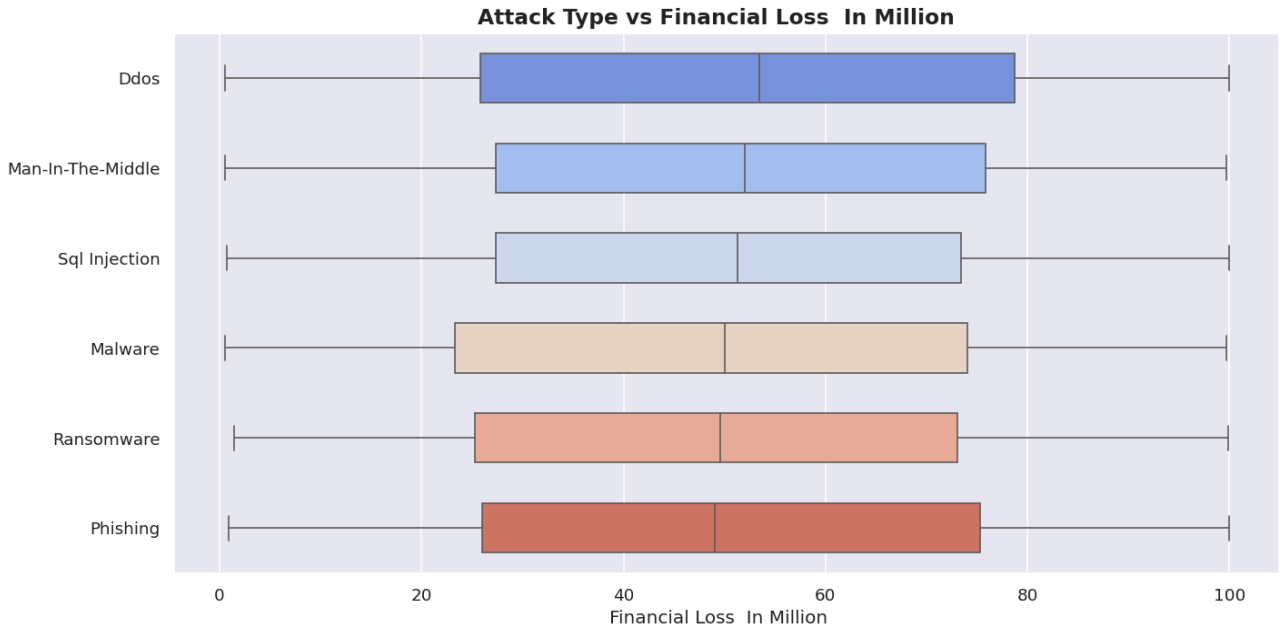


直方圖



水平盒狀圖

可以看到最大與最小的損失值



文字統整

```python
# ============================================================
# Step 5: EDA 文字輸出
# ============================================================

print('=' * 55)
print('          EDA SUMMARY REPORT')
print('=' * 55)
print(f'Total Records     : {len(df):,}')
print(f'Total Features    : {df.shape[1]}')
print(f'Year Range        : {df[year_col].min()} - {df[year_col].max()}')
print(f'Attack Types      : {df[attack_col].nunique()} unique')
print(f'Avg {loss_col:20s}: {df[loss_col].mean():,.2f}')
print(f'Max {loss_col:20s}: {df[loss_col].max():,.2f}')
print(f'Min {loss_col:20s}: {df[loss_col].min():,.2f}')
print('-' * 55)
print('Top 3 Most Frequent Attack Types:')
for i, (name, cnt) in enumerate(attack_counts.head(3).items(), 1):
    print(f'  {i}. {name:30s} ({cnt:,} incidents)')
print('=' * 55)
```

```
=======================================================
          EDA SUMMARY REPORT
=======================================================
Total Records     : 3,000
Total Features    : 12
Year Range        : 2015 - 2024
Attack Types      : 6 unique
Avg financial_loss__in_million___: 50.49
Max financial_loss__in_million___: 99.99
Min financial_loss__in_million___: 0.50
-------------------------------------------------------
Top 3 Most Frequent Attack Types:
  1. Ddos                           (531 incidents)
  2. Phishing                       (529 incidents)
  3. Sql Injection                  (503 incidents)
=======================================================
```

與我們在圖中看到的相符，例如：不同種類攻擊次數沒有太大的改變。

3. 做資料前處理與正規化

- 統一欄位名稱，將可能

```
# ============================================================
# Step 4: 資料清洗與正規化
# ============================================================

# 欄位名稱統一：去空白、小寫、底線
df.columns = (
    df.columns
      .str.strip()
      .str.lower()
      .str.replace(' ', '_')
      .str.replace(r'[^\w]', '_', regex=True)
)

print('清理後欄位名稱: ', df.columns.tolist())

# 確認數值欄位
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = df.select_dtypes(include='object').columns.tolist()
print(f'\n數值欄位: {num_cols}')
print(f'類別欄位: {cat_cols}')

# 移除重複行
df.drop_duplicates(inplace=True)

# 處理缺失值
#    數值型：填中位數；類別型：填眾數
for col in df.columns:
    if df[col].dtype in ['float64', 'int64']:
        df[col].fillna(df[col].median(), inplace=True)
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)

# 統一字串格式（去除空白、轉小寫後首字大寫）
cat_cols = df.select_dtypes(include='object').columns
for col in cat_cols:
    df[col] = df[col].str.strip().str.title()

print("清理後 Shape:", df.shape)
print("剩餘缺失值:", df.isnull().sum().sum())

# 若有 Financial Loss (USD Millions) 欄位，做 log 轉換減少偏態
if 'financial_loss__in_million___' in df.columns:
    df['financial_loss_log'] = np.log1p(df['financial_loss__in_million___'])

# 將 Year 轉為距基準年的相對年份（可選）
df['year_offset'] = df['year'] - df['year'].min()

print(df[['year', 'year_offset']].head())
```

```python
from sklearn.preprocessing import LabelEncoder

# Label Encoding (適合 tree-based 模型)
le = LabelEncoder()
label_cols = ['attack_type', 'target_industry', 'security_vulnerability_type',
              'defense_mechanism_used', 'country']

df_encoded = df.copy()
for col in label_cols:
    if col in df_encoded.columns:
        df_encoded[col + '_Enc'] = le.fit_transform(df_encoded[col].astype(str))

# One-Hot Encoding (適合線性/神經網路模型)
df_ohe = pd.get_dummies(df, columns=label_cols, drop_first=True)

print("Label Encoded Shape:", df_encoded.shape)
print("One-Hot Encoded Shape:", df_ohe.shape)
```

```python
from sklearn.preprocessing import MinMaxScaler, StandardScaler

num_cols = df.select_dtypes(include=['float64', 'int64']).columns.tolist()
print("數值欄位:", num_cols)

# Min-Max Normalization (將值縮放至 [0, 1])
scaler_minmax = MinMaxScaler()
df_minmax = df.copy()
df_minmax[num_cols] = scaler_minmax.fit_transform(df[num_cols])

# Z-score Standardization (均值=0, 標準差=1)
sch_std = StandardScaler()
df_std = df.copy()
df_std[num_cols] = sch_std.fit_transform(df[num_cols])
```

```
清理後欄位名稱: ['country', 'year', 'attack_type', 'target_industry', 'financial_loss__in_million___', 'number_of_affected_users', 'attack_source', 'securit
…
數值欄位: ['year', 'financial_loss__in_million___', 'number_of_affected_users', 'incident_resolution_time__in_hours_']
類別欄位: ['country', 'attack_type', 'target_industry', 'attack_source', 'security_vulnerability_type', 'defense_mechanism_used']
--------------------------------------------
清理後 Shape: (3000, 10)
剩餘缺失值: 0
--------------------------------------------
   year  year_offset
0  2019           4
1  2019           4
2  2017           2
3  2024           9
4  2018           3
--------------------------------------------
Label Encoded Shape: (3000, 17)
One-Hot Encoded Shape: (3000, 34)
--------------------------------------------
數值欄位: ['year', 'financial_loss__in_million___', 'number_of_affected_users', 'incident_resolution_time__in_hours_', 'financial_loss_log', 'year_offset']
```

```
--- Min-Max 正規化後 ---
            year  financial_loss__in_million___  number_of_affected_users  \
count  3000.000                        3000.000                   3000.000
mean      0.508                           0.502                      0.505
std       0.318                           0.289                      0.290
min       0.000                           0.000                      0.000
25%       0.222                           0.254                      0.256
50%       0.556                           0.506                      0.504
75%       0.778                           0.755                      0.758
max       1.000                           1.000                      1.000

       incident_resolution_time__in_hours_  financial_loss_log  year_offset
count                              3000.000            3000.000     3000.000
mean                                  0.500               0.779        0.508
std                                   0.290               0.203        0.318
min                                   0.000               0.000        0.000
25%                                   0.254               0.684        0.222
50%                                   0.507               0.841        0.556
75%                                   0.761               0.934        0.778
max                                   1.000               1.000        1.000
```

- financial_loss_log 異常偏高（mean=0.779，中位數=0.841），代表：
  原始財務損失資料右偏分布（少數超大損失事件拉高平均）
  對數轉換後仍偏高，說明重大資安事件的財務衝擊不成比例地集中在高端
  75% 分位數達 0.934，說明大多數損失集中在高區間

```
--- Z-score 標準化後 ---
            year  financial_loss__in_million___  number_of_affected_users  \
count  3000.000                        3000.000                   3000.000
mean     -0.000                           0.000                      0.000
std       1.000                           1.000                      1.000
min      -1.599                          -1.737                     -1.739
25%      -0.900                          -0.859                     -0.859
50%       0.150                           0.010                     -0.001
75%       0.850                           0.873                      0.874
max       1.550                           1.719                      1.707

       incident_resolution_time__in_hours_  financial_loss_log  year_offset
count                              3000.000            3000.000     3000.000
mean                                 -0.000              -0.000        0.000
std                                   1.000               1.000        1.000
min                                  -1.725              -3.847       -1.599
25%                                  -0.850              -0.466       -0.900
50%                                   0.025               0.309        0.150
75%                                   0.901               0.769        0.850
max                                   1.727               1.092        1.550
```

- financial_loss_log 的 min = -3.847 是最值得注意的異常：
  Z-score 達 -3.847 遠超過 3σ 門檻（通常 |z| > 3 視為離群）
  代表存在財務損失極小的邊緣事件（可能是初期偵測到的小規模攻擊、或未造成實質損失的探測行為）
  反之 max 僅 1.092，說明高損失端相對收斂，不存在極端超大損失離群點

4. 建立一個 簡單線性迴歸模型（Simple Linear Regression）

再來建立簡單線性迴歸模型，以了解單一特徵是否能有效預測目標變數。

此次模型設定中，選擇 number_of_affected_users 作為自變數（feature, X），financial_loss__in_million___ 作為應變數（target, y），藉此分析受影響使用者數量是否能用來預測資安事件造成的財務損失。

```
target = "financial_loss__in_million___"
feature = "number_of_affected_users"
```

首先，使用 train_test_split() 將資料集切分為訓練集與測試集，其中訓練集占 80%，測試集占 20%，並設定 random_state=42，以確保實驗結果具可重現性。

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

接著，由於不同欄位的數值尺度可能不同，因此使用 StandardScaler 對自變數進行標準化處理。標準化的方式是先以訓練集資料計算平均值與標準差，再將相同轉換套用到測試集，以避免資料洩漏問題。

完成資料切分與正規化後，使用 LinearRegression() 建立簡單線性迴歸模型，並以訓練集資料進行模型訓練。

模型訓練完成後，再利用測試集進行預測，得到預測結果 y_pred，最後使用均方誤差（MSE）與決定係數（R^2）評估模型效能，並額外輸出模型的截距（Intercept）與迴歸係數（Coefficient），以觀察線性關係的方向與強度。

```
model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE =", mse)
print("R² =", r2)
print("Intercept =", model.intercept_)
print("Coefficient =", model.coef_[0])
```

本次程式流程如下：先匯入模型建立所需的套件，包括資料切分、標準化、線性迴歸與評估指標。
接著指定自變數與目標變數，完成訓練集與測試集切分，之後利用 StandardScaler 對資料進行標準化，再以 LinearRegression 建立模型並進行預測，最後輸出 MSE、R^2、截距與係數，作為模型分析依據。

以下為完整的程式碼片段截圖：

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

target = "financial_loss__in_million___"
feature = "number_of_affected_users"

X = df[[feature]]
y = df[target]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model = LinearRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE =", mse)
print("R² =", r2)
print("Intercept =", model.intercept_)
print("Coefficient =", model.coef_[0])
```

```
MSE = 808.9102336252276
R² = -0.00034208499642574
Intercept = 50.42505416666667
Coefficient = -0.10800766450206266
```

這段程式是將簡單線性迴歸的結果畫成圖，方便觀察模型擬合情況。首先使用未標準化的 X_train 和 y_train 重新訓練一個 LinearRegression() 模型，讓圖上的 X 軸保留原始數值，較容易解讀。
接著將 X_test 排序後再進行預測，避免回歸線因資料順序混亂而顯得不平順。最後以散佈圖呈現實際測試資料，並疊加模型的回歸線。

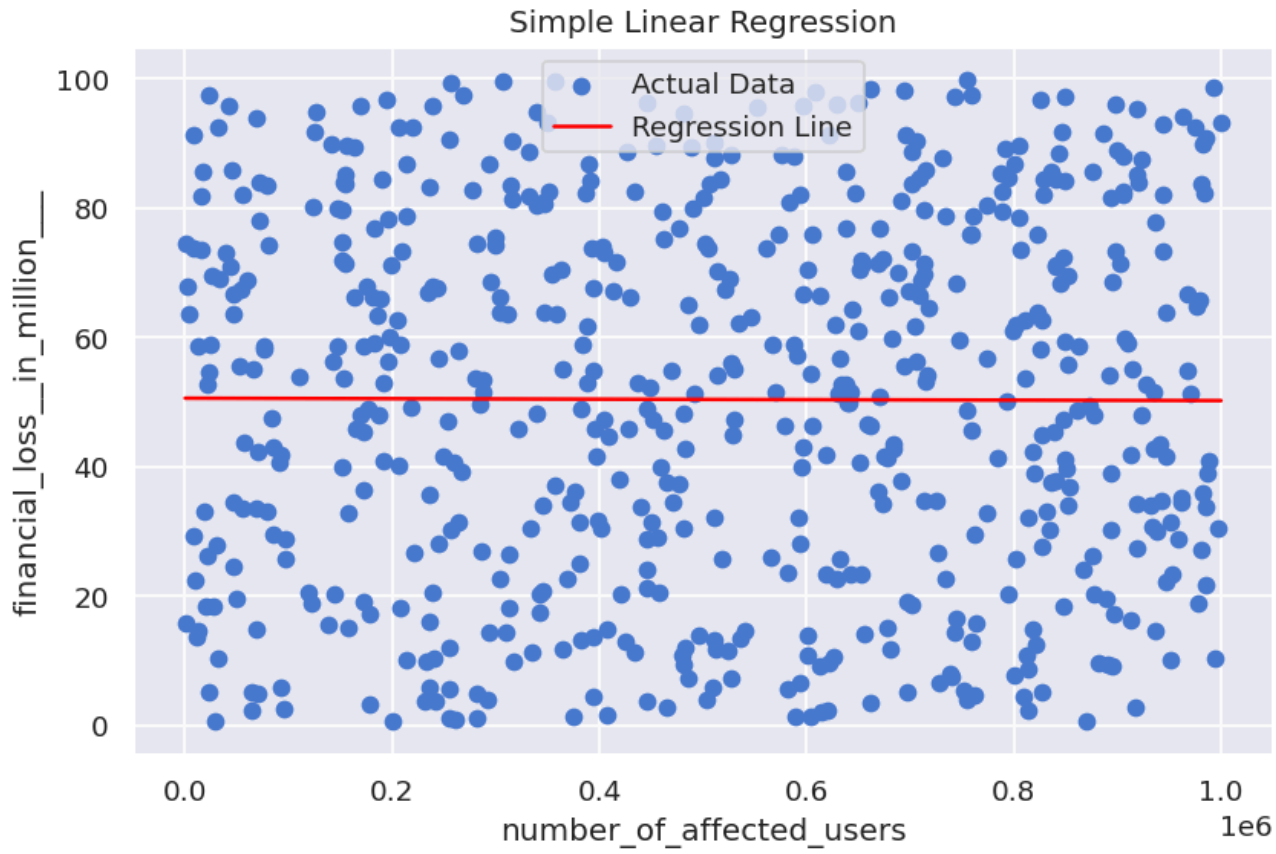
4.1 將簡單線性迴歸的結果畫成圖，方便觀察模型擬合情況

```python
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import numpy as np

# 用原始未標準化資料畫圖比較直觀
plot_model = LinearRegression()
plot_model.fit(X_train, y_train)

# 先把 X_test 排序，畫出來的線才不會亂
X_test_sorted = X_test.sort_values(by=feature)
y_pred_line = plot_model.predict(X_test_sorted)

plt.figure(figsize=(8,5))
plt.scatter(X_test, y_test, label="Actual Data")
plt.plot(X_test_sorted, y_pred_line, color="red", label="Regression Line")
plt.xlabel(feature)
plt.ylabel(target)
plt.title("Simple Linear Regression")
plt.legend()
plt.show()
```

從圖中可以看出，資料點分布相當分散，沒有隨著 number_of_affected_users 增加而呈現明顯上升或下降趨勢。

紅色回歸線幾乎接近水平，表示此特徵對 financial_loss__in_million___ 的影響非常有限。

這也和前面的 R^2 接近 0 且為負值的結果一致，代表單靠 number_of_affected_users 無法有效預測財務損失。



5. 用 MSE 和 R^2 評估這個模型

本研究的簡單線性迴歸模型評估結果如下：

- MSE = 808.9102336252276

- R^2 = -0.00034208499642574
- Intercept = 50.42505416666667
- Coefficient = -0.10800766450206266

首先，MSE（Mean Squared Error, 均方誤差）用來衡量模型預測值與實際值之間的平均平方差，數值越小代表模型預測效果越好。
本次模型的 MSE 為 808.91，顯示模型在預測財務損失時仍存在明顯誤差。
不過，MSE 的大小仍需搭配資料本身的數值範圍一起解讀，因此更重要的指標是 R^2。

R^2（Coefficient of Determination, 決定係數）用來衡量模型對目標變數變異的解釋能力。
理想情況下，R^2 越接近 1，代表模型越能解釋資料；若接近 0，表示模型解釋能力很弱；若小於 0，則表示模型甚至比直接以平均值作為預測還差。
本次模型的 R^2 為 -0.000342，不但非常接近 0，且為負值，代表 number_of_affected_users 幾乎無法有效解釋 financial_loss__in_million___ 的變化，模型預測能力非常有限。

此外，模型的截距為 50.4251，代表在特徵經過標準化後位於基準位置時，模型預測的財務損失約為 50.43。迴歸係數為 -0.1080，表示在此模型中，number_of_affected_users 與 financial_loss__in_million___ 呈現非常微弱的負向關係，也就是受影響使用者數量增加時，預測財務損失反而略微下降。
然而，由於 R^2 幾乎等於 0，表示這個負向關係並不具明顯解釋力，因此不能視為穩定或有意義的線性關聯。

綜合以上結果，可以得出結論：以 number_of_affected_users 作為單一自變數建立的簡單線性迴歸模型，無法有效預測 financial_loss__in_million___。
這代表受影響使用者數量本身不足以單獨解釋財務損失，財務損失可能還受到其他因素影響，例如攻擊類型、攻擊來源、產業別、漏洞類型、防禦機制或事件處理時間等。
因此，若要進一步提升模型表現，應建立多元線性迴歸模型，納入更多特徵一起分析。

6. 再建立一個多元線性迴歸模型（Multiple Linear Regression），並做特徵選擇

1. 特徵選擇：使用相關係數 (Correlation)

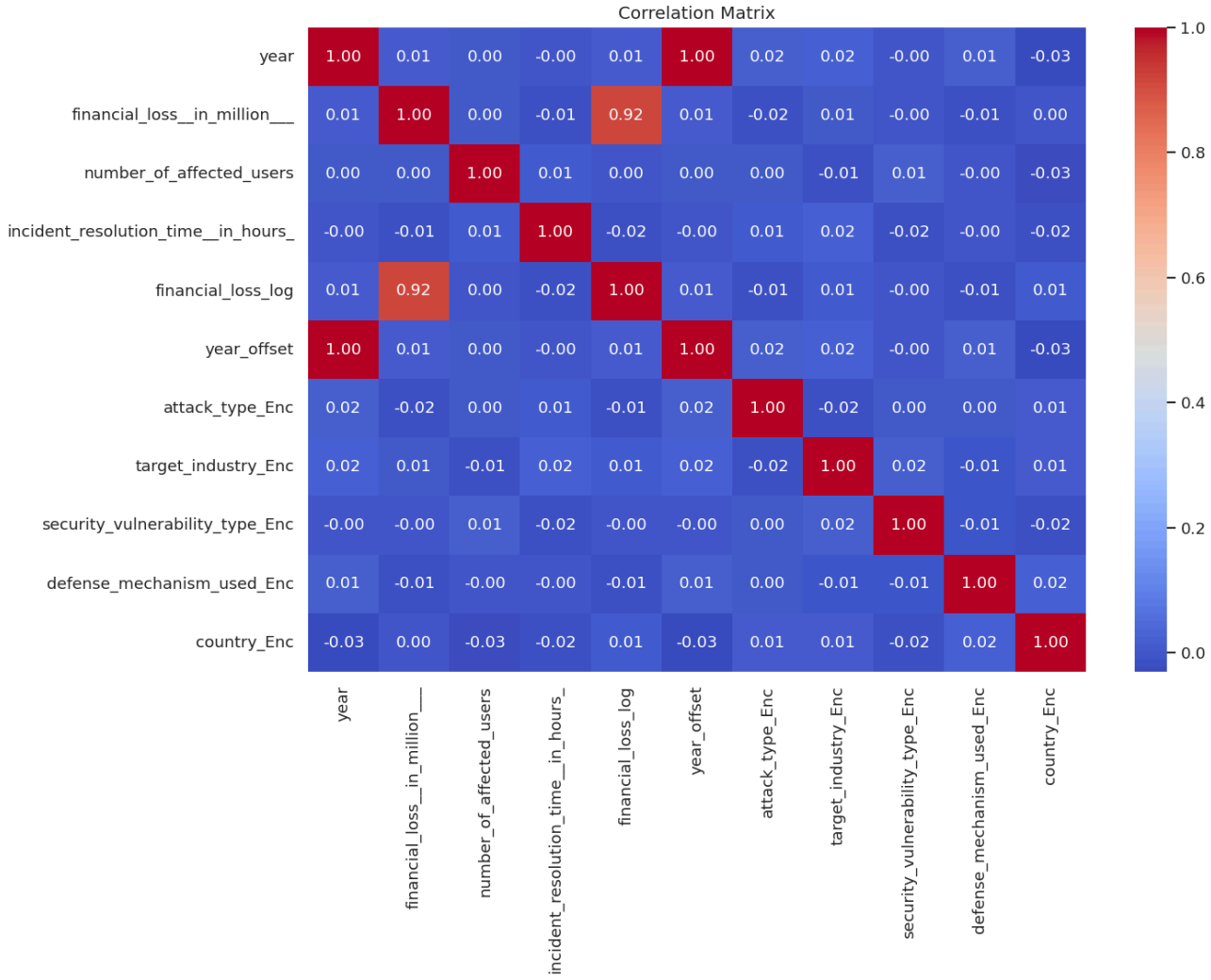
使用 df_encoded 資料表計算 Pearson 相關係數矩陣 (Correlation
並透過 Heatmap 進行視覺化呈現。
觀察到以下現象：

- 目標變數轉換：financial_loss_log 與原始損失具備極高相關性 (0.92)，證實了 Log 轉換能有效保留原始數據趨勢，同時處理偏態分佈。

- 外部特徵關聯性：其餘特徵如 attack_type_Enc (-0.018)、incident_resolution_time (-0.013) 與 year_offset (0.011) 與目標變數之相關係數絕對值均 低於 0.02。

- 統計推論：本資料集之特徵(包含：攻擊類型 (Attack Type)、處理時間 (Resolution Time)、年份 (Year) 以及 受影響用戶數 (Affected Users))與財務損失之間呈現相關性極低的現象。

```
# 計算相關係數矩陣
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))

corr_matrix = df_encoded.corr(numeric_only=True)

sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Matrix")
plt.show()

# 找出與目標變數相關性較高的特徵
target_col = 'financial_loss__in_million___'
if target_col in corr_matrix.columns:
    target_corr = corr_matrix[target_col].sort_values(ascending=False)
    print("與財務損失的相關性排序：\n", target_corr)
else:
    print(f"找不到欄位 {target_col}")
```



與財務損失的相關性排序：

financial_loss__in_million___ 1.000000

financial_loss_log 0.915794

year 0.010581

year_offset 0.010581

target_industry_Enc 0.005423

number_of_affected_users 0.001787

country_Enc 0.001085

security_vulnerability_type_Enc -0.004783

defense_mechanism_used_Enc -0.011496

incident_resolution_time__in_hours_ -0.012671

attack_type_Enc -0.018050

Name: financial_loss__in_million___, dtype: float64

2. 定義X與Y並拆分資料

- 採用 One-Hot Encoding
  本步驟捨棄了 Label Encoding 資料，改為使用 df_ohe (One-Hot Encoding)。這是因為線性模型在處理「攻擊類型」或「產業」等類別資料時，若使用數字編碼會產生「大小順序」的誤導。透過 Dummy Variables (0與1) 的處理，模型能獨立評估如 Malware 或 Phishing 等特定特徵對財務損失的影響力。

- 目標變數轉換 (Log Transformation)：
  延續前處理階段的策略，模型預測的目標定為 financial_loss_log。此做法符合統計學中處理偏態分佈 (Skewness) 的原則，能有效降低極端值對迴歸線段的拉扯，使模型預測更為穩定。

- 資料拆分 (Train-Test Split) ：
  為了驗證模型的泛化能力，本研究將資料集以 80% 訓練集 (Training Set) 與 20% 測試集 (Testing Set) 的比例進行拆分，並設定固定隨機種子 (random_state=42) 以確保實驗結果的可重複性。

模型執行流程:
使用 Scikit-learn 的 LinearRegression 演算法：
Fit (訓練階段)：利用訓練集資料計算每個特徵的權重係數 (Coefficients) 與截距項。
Predict (預測階段)：將測試集的特徵輸入訓練好的模型，產出預測值 y_pred，準備與實際值 y_test 進行對照評估。

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# 1. 重新定義 X 與 y (改用 Log 轉換後的 y)
# 挑選相關性相對「較高」的特徵
features = [
    'year_offset',
    'incident_resolution_time__in_hours_',
    'attack_type_Malware',
    'attack_type_Phishing',
    'target_industry_Healthcare'
]
X = df_ohe[features]
y = df_ohe['financial_loss_log']  # 改用 Log 欄位

# 2. 拆分資料
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra

# 3. 訓練多元模型
multi_model = LinearRegression()
multi_model.fit(X_train, y_train)

# 4. 預測
y_pred = multi_model.predict(X_test)
```

7. 用 MSE、R²、Adjusted R² 來評估多元模型

1. 計算評估指標

- 均方誤差 (MSE):
  0.7575由於目標變數已進行 Log 轉換，此數值代表對數尺度下的平均誤差平方。雖然數值較小，但需結合 R^2 進行綜合判定。
- 決定係數 (R^2 Score):
  0.0006此指標代表模型僅能解釋資料中約 0.06% 的變異。極低的 R^2 值證實了自變數與目標變數之間幾乎不存在顯著的線性相關性。
- 調整後決定係數 (Adjusted R^2 Score):
  -0.0078這是本評估中最關鍵的指標。依據講義 p.43 之理論，當加入的特徵變數無法提供有效的解釋能力時，Adjusted R^2 會因樣本數與特徵數的懲罰項而下降。本結果出現 負值，明確指出該模型目前的特徵組合表現甚至劣於直接預測平均值。

觀察

特徵解釋力極限：

- 根據相關係數矩陣與 Adjusted R^2 的交叉驗證，顯示「攻擊類型」、「年份」與「處理時間」等外部因素對於預測「財務損失」的線性貢獻度極低。
- 資料隨機性分析：
  觀測其 Min-Max 正規化後的分布（平均值均在 0.5 左右），顯示此 Kaggle 資料集可能具備高度隨機性或為模擬生成的數據（Synthetic Data），這導致線性模型難以從中擷取規律性的斜率。

```
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# 計算 MSE 與 R²
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
n = X_test.shape[0]
p = X_test.shape[1]
adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1)) # 講義公式 [cite: 366,

print(f"MSE: {mse:.4f}")
print(f"R2 Score: {r2:.4f}")
print(f"Adjusted R2 Score: {adjusted_r2:.4f}")
```

MSE: 0.7575

R2 Score: 0.0006

Adjusted R2 Score: -0.0078

2. 視覺化結果：實際值 vs. 預測值

水平分佈現象 (Underfitting)：

觀測結果顯示，預測點並未如預期分佈在紅色斜線周圍，而是呈現明顯的 「水平雲狀 (Horizontal Cloud)」。這代表模型對於不同程度的財務損失，產出的預測值皆集中在特定的平均區間（對數尺度的 5.0 附近）。

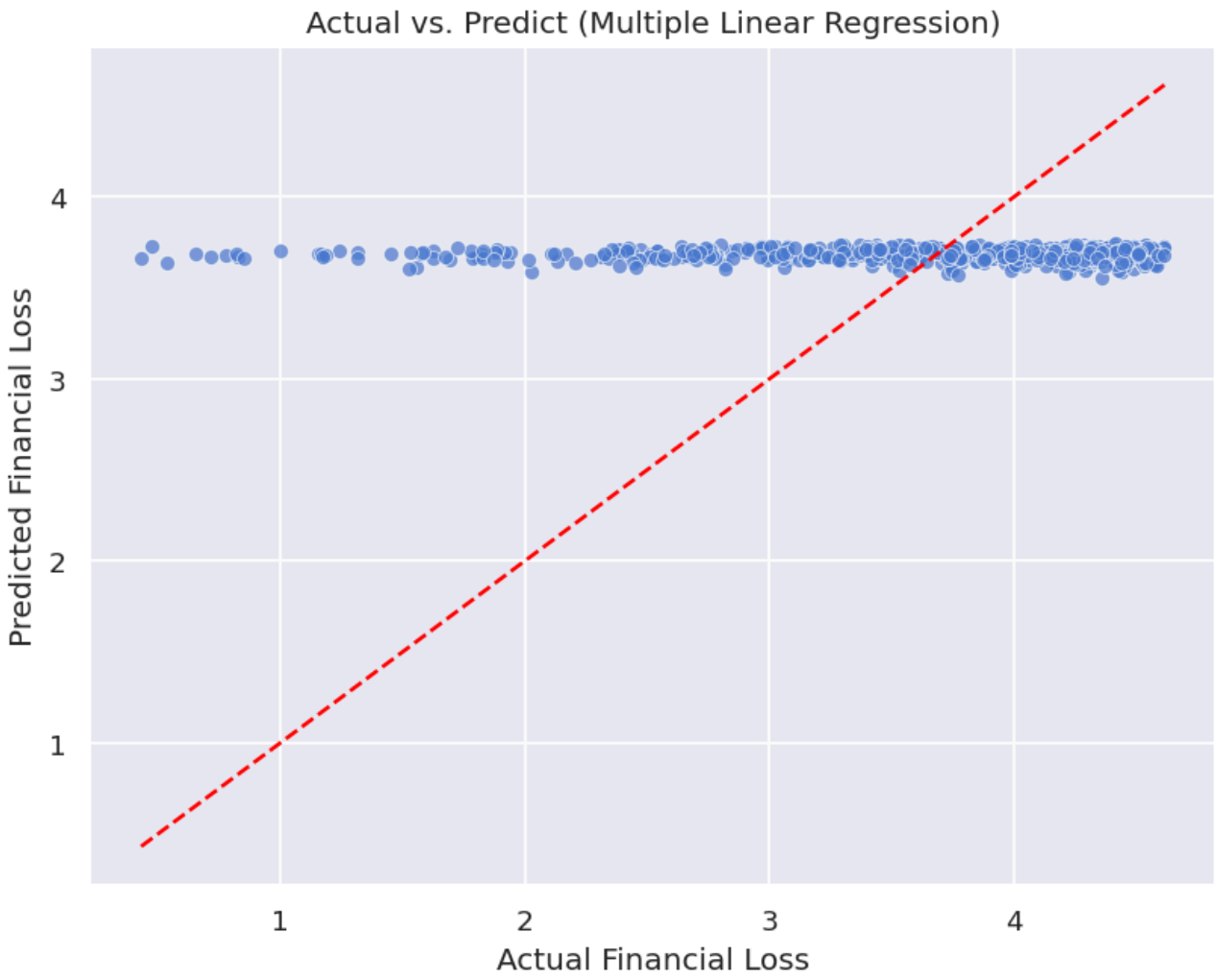
偏誤分析 (Bias Analysis)：

此現象證實了模型存在顯著的 高偏差 (High Bias)。由於輸入特徵（攻擊類型、年份等）與目標變數間的線性相關性微弱，線性迴歸演算法為了最小化整體誤差 (MSE)，傾向於預測資料的平均值，而非捕捉其變動趨勢。

Log 轉換之影響：

受惠於前處理階段的 Log 轉換，座標軸尺度已成功從原始的數百萬級距壓縮至對數區間，這避免了極端離群值對圖表造成的視覺扭曲，使數據分佈更具統計觀測意義。

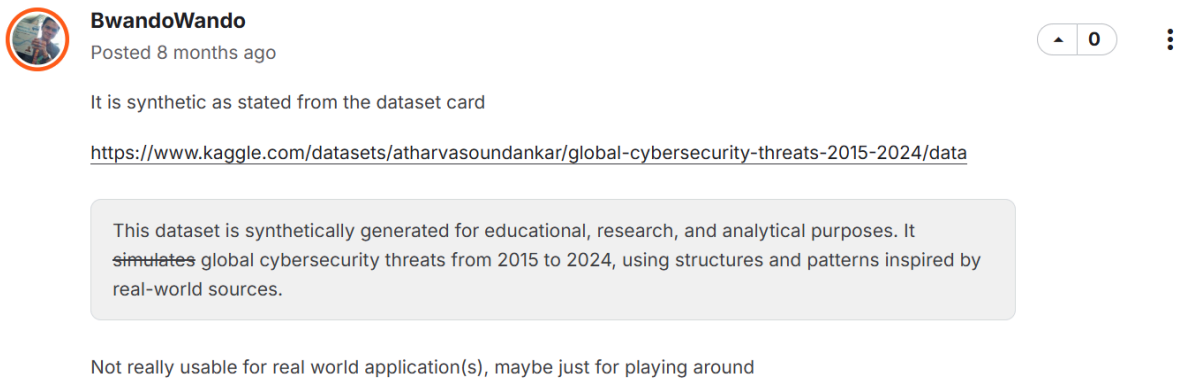
```
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color=
plt.xlabel("Actual Financial Loss")
plt.ylabel("Predicted Financial Loss")
plt.title("Actual vs. Predict (Multiple Linear Regression)")
plt.show()
```



Data Insights

1. 你從資料中發現了哪些趨勢或模式？

- 根據一開始的資料 EDA 製圖我們發現每年發生的攻擊類型分布有點太平均，不太像是真實環境發生的情境，後來我們回去找該資料集的討論發現此資料集並非真實世界攻擊發生之統計，而是生成用於訓練的資料集，這也反映出我們做分析時遇到的一些不合理情境，可能我們的分析才是找出資料集有問題之處，不過也透過此次機會學習到遇到分析出不合理的結果時，一方面檢視自己的分析方法，另一方面也要質疑資料來源的正確性與真實性。



回到正題，我們發現的趨勢是攻擊次數、種類及發生年份沒有太多變動，也正是因為如此才發現資料集本身的問題，如果以此議題推論，在疫情時（2020~2022）可能要有較多的攻擊次數，並且可能會集中在 Malware, Ransomeware, Phishing 這些類型上，因為遠距上班攻擊者有較多攻擊面可以投入竊取資料的管道。

2. 哪些變數對預測影響最大？

根據相關係數分析結果，可以發現和 financial_loss__in_million___ 關聯最高的欄位是 financial_loss_log，其相關係數約為 0.9158。
不過，financial_loss_log 是由原本的財務損失欄位經過對數轉換後得到的衍生欄位，所以雖然它和目標變數的相關性很高，但不能直接當作真正獨立的影響因素來解釋。

如果只看原始資料中的其他特徵，**整體來說幾乎沒有哪個變數和財務損失有明顯的線性關係**。
例如 year 和 year_offset 的相關係數大約是 0.0106，number_of_affected_users 約為 0.0018，incident_resolution_time__in_hours_ 約為 -0.0127，attack_type_Enc 約為 -0.0181。
這些數值都非常接近 0，**代表它們對財務損失的線性影響非常小**。

在多元線性迴歸的部分，我們進一步選用了 year_offset、incident_resolution_time__in_hours_、attack_type_Malware、attack_type_Phishing 和 target_industry_Healthcare 等特徵來建立模型，希望透過多個變數一起預測財務損失。
但最後的結果顯示，模型的 R^2 只有 0.0006，Adjusted R^2 更是 -0.0078，表示**即使加入多個特徵後，模型的解釋能力還是非常有限**。

綜合來看，若不把 financial_loss_log 這種由目標變數轉換而來的欄位算進去，這份資料中其實沒有哪個外部變數對財務損失有特別明顯的預測效果。
也就是說，這個資料集裡的財務損失變化，可能不是由單一變數主導，而是具有比較高的隨機性，或是受到資料中沒有收錄的其他因素影響。

3. 簡單線性迴歸和多元線性迴歸的結果差在哪裡？

- 簡易對照表

評估指標	簡單線性迴歸 (SLR)	多元線性迴歸 (MLR)	差異分析
自變數 (X)	僅 1 個 (affected_users)	5 個以上 (多元特徵)	MLR 資訊量較大，嘗試捕捉更多維度的規律。
目標變數 (y)	原始財務損失金額	Log 轉換後金額	MLR 處理了偏態 (Skewness)，讓預測尺度更合理。
MSE (均方誤差)	808.91	0.7575	MLR 顯著勝出。透過 Log 轉換大幅收斂預測誤差。
R^2	-0.0003	0.0006	MLR 略微提升。多變數提供的解釋力優於單一變數
Adjusted R^2	未計算	-0.0078	MLR 更具統計嚴謹度，揭示了加入無效變數的懲罰。

- 主要差異
  預測精確度 (MSE)：
  多元線性迴歸透過對目標變數進行 Log 轉換，成功解決了資料極端值導致的高偏態問題，使預測誤差（MSE）從 808.91 降至 0.75，數值表現上較為穩定。
  模型解釋力 (R^2 與 Adjusted R^2)：

雖然兩者之 R^2 皆趨近於 0，但多元迴歸透過 Adjusted R^2 進行了自我修正（講義 p.43）。負值的 Adjusted R^2 證明即使加入更多特徵（如攻擊類型、產業），對於具備高隨機性的財務損失數據，線性模型的預測力仍有其侷限。

類別資料處理方式：

多元迴歸採用了 One-Hot Encoding (Dummy Variables) 處理類別資料，比簡單迴歸僅使用單一數值欄位更能公平評估不同「攻擊類型」或「產業別」對損失的邊際影響。

- 結論

  實驗結果顯示，多元線性迴歸在「預測尺度」與「資訊豐富度」上優於簡單線性迴歸。

  然而，由於兩者之 R^2 與 Adjusted R^2 均極低，且視覺化圖表呈現不甚理想，這證實了本資料集之財務損失與現有特徵間的線性相關性極其微弱。