```
import pandas as pd

# Load dataset
file_path = "/content/Global_Cybersecurity_Threats_2015-2024.csv"
df = pd.read_csv(file_path)


df=pd.DataFrame(df)

df
```

	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
...	...	...	...	...	...	...	...	...	...	...
2995	UK	2021	Ransomware	Government	51.42	190694	Unknown	Social Engineering	Firewall	52
2996	Brazil	2023	SQL Injection	Telecommunications	30.28	892843	Hacker Group	Zero-day	VPN	26
2997	Brazil	2017	SQL Injection	IT	32.97	734737	Nation-state	Weak Passwords	AI-based Detection	30
2998	UK	2022	SQL Injection	IT	32.17	379954	Insider	Unpatched Software	Firewall	9
2999	Germany	2021	SQL Injection	Retail	48.20	480984	Unknown	Zero-day	VPN	64

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.head()
```

	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-	IT	74.41	810682	Insider	Social	VPN	68

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.tail()
```

	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)
2995	UK	2021	Ransomware	Government	51.42	190694	Unknown	Social Engineering	Firewall	52
2996	Brazil	2023	SQL Injection	Telecommunications	30.28	892843	Hacker Group	Zero-day	VPN	26
2997	Brazil	2017	SQL Injection	IT	32.97	734737	Nation-state	Weak Passwords	AI-based Detection	30
2998	UK	2022	SQL Injection	IT	32.17	379954	Insider	Unpatched Software	Firewall	9
2999	Germany	2021	SQL	Retail	48.20	480984	Unknown	Zero-day	VPN	64

```
df.shape
```

```
(3000, 10)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 10 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Country                           3000 non-null   object 
 1   Year                              3000 non-null   int64  
 2   Attack Type                       3000 non-null   object 
 3   Target Industry                   3000 non-null   object 
 4   Financial Loss (in Million $)     3000 non-null   float64
 5   Number of Affected Users          3000 non-null   int64  
 6   Attack Source                     3000 non-null   object 
 7   Security Vulnerability Type       3000 non-null   object 
 8   Defense Mechanism Used            3000 non-null   object 
 9   Incident Resolution Time (in Hours)  3000 non-null   int64  
dtypes: float64(1), int64(3), object(6)
memory usage: 234.5+ KB
```

```
df.describe()
```

	Year	Financial Loss (in Million $)	Number of Affected Users	Incident Resolution Time (in Hours)
count	3000.000000	3000.000000	3000.000000	3000.000000
mean	2019.570333	50.492970	504684.136333	36.476000
std	2.857932	28.791415	289944.084972	20.570768
min	2015.000000	0.500000	424.000000	1.000000
25%	2017.000000	25.757500	255805.250000	19.000000
50%	2020.000000	50.795000	504513.000000	37.000000
75%	2022.000000	75.630000	758088.500000	55.000000
max	2024.000000	99.990000	999635.000000	72.000000

```
# Fill missing categorical values with "Unknown"
categorical_cols = df.select_dtypes(include=['object']).columns
df[categorical_cols] = df[categorical_cols].fillna("Unknown")


# Fill missing numerical values with the median
numerical_cols = df.select_dtypes(include=['number']).columns
df[numerical_cols] = df[numerical_cols].fillna(df[numerical_cols].median())


# Verify missing values are handled
print(df.isnull().sum())
```

```
Country                           0
Year                              0
Attack Type                       0
Target Industry                   0
Financial Loss (in Million $)     0
Number of Affected Users          0
Attack Source                     0
Security Vulnerability Type       0
Defense Mechanism Used            0
Incident Resolution Time (in Hours)    0
```

```
dtype: int64
```

```
# Ensure column names are properly formatted
df.columns = df.columns.str.strip().str.lower()
```

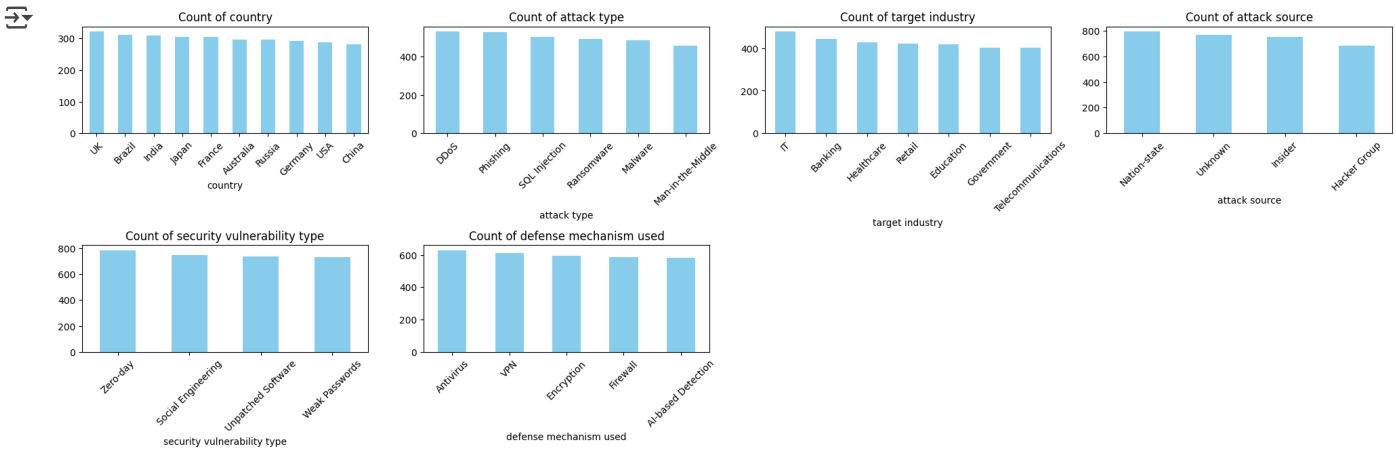
```
# Check column names
print(df.columns)
```

```
Index(['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)'],
      dtype='object')
```

```
# Ensure the target column exists
if "attack type" in df.columns:
    target_column = "attack type"
else:
    raise ValueError("Target column 'Attack Type' not found in dataset.")
```

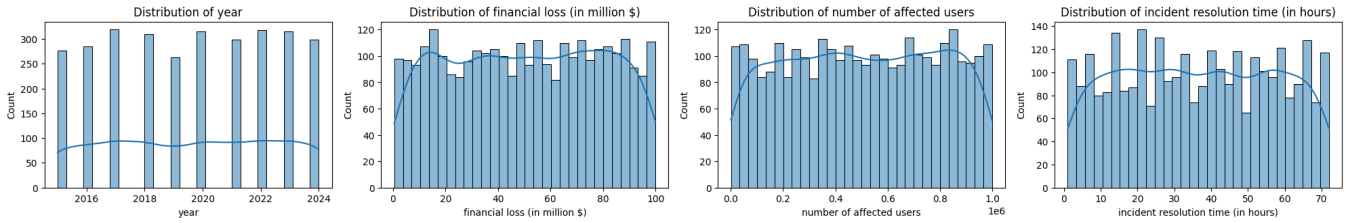
```
import matplotlib.pyplot as plt
import seaborn as sns
# Select categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
```

```
# Plot bar charts for categorical columns
plt.figure(figsize=(20, 15))

for i, col in enumerate(categorical_cols):
    plt.subplot(5, 4, i + 1)  # Adjusting subplot grid dynamically
    df[col].value_counts().plot(kind='bar', color='skyblue')
    plt.title(f'Count of {col}')
    plt.xticks(rotation=45)  # Rotate x-axis labels for readability

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




```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

numerical_cols = df.select_dtypes(include=[np.number]).columns
print("Numerical Columns:", numerical_cols)

# Plot histograms for numerical columns
plt.figure(figsize=(20, 15))

for i, col in enumerate(numerical_cols):
    plt.subplot(5, 4, i + 1)  # Adjusting subplot grid dynamically
    sns.histplot(df[col], kde=True, bins=30)
    plt.title(f'Distribution of {col}')

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

```
Numerical Columns: Index(['year', 'financial loss (in million $)', 'number of affected users',
       'incident resolution time (in hours)'],
      dtype='object')
```




```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])


from sklearn.model_selection import train_test_split

X = df.drop(columns=[target_column])  # Features
y = df[target_column]  # Target variable

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

print(f"Training Data: {X_train.shape}, Testing Data: {X_test.shape}")
```

```
Training Data: (2400, 9), Testing Data: (600, 9)
```

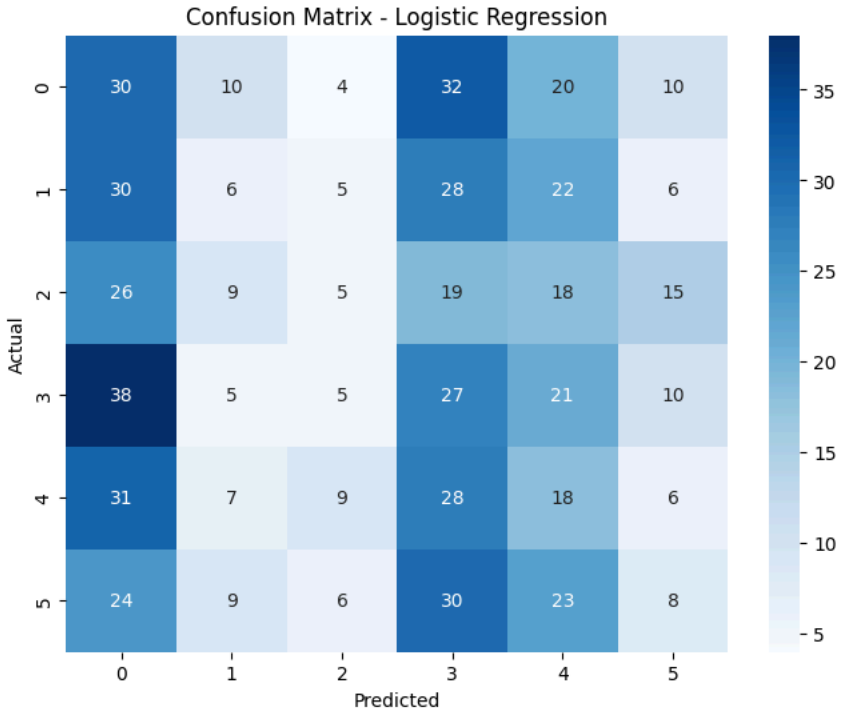
```
from sklearn.preprocessing import StandardScaler

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


from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Define models
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),

}


# Train, Predict, and Evaluate
results = {}
for name, clf in classifiers.items():
    print(f"\nTraining Model: {name}")

    # Train model
    clf.fit(X_train_scaled, y_train)

    # Predict
    y_pred = clf.predict(X_test_scaled)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy: {accuracy:.4f}")
    print(classification_report(y_test, y_pred))

    # Store results
    results[name] = accuracy
```

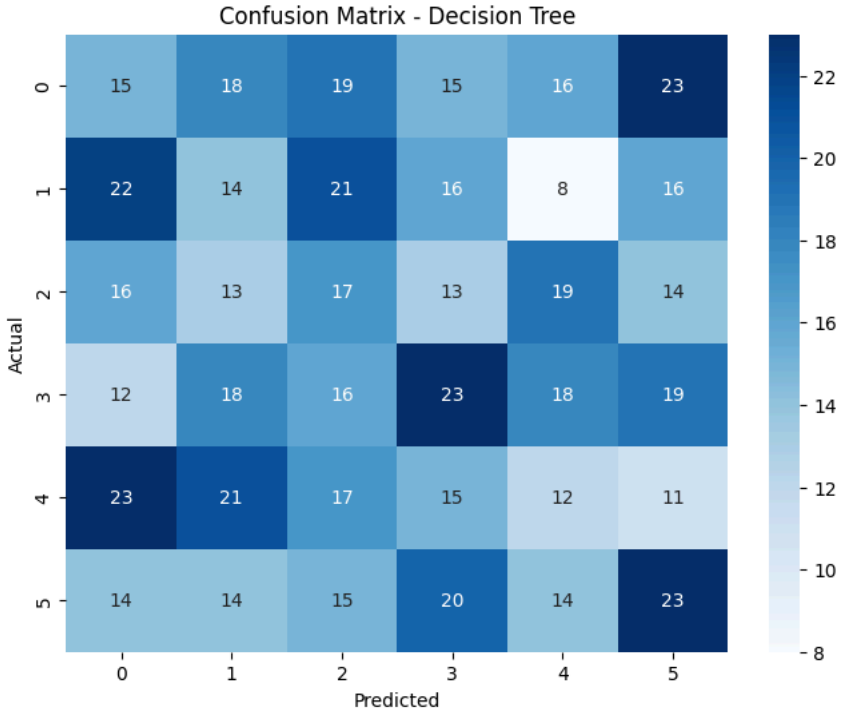
```
Training Model: Logistic Regression
Accuracy: 0.1567
              precision    recall  f1-score   support

           0       0.17      0.28      0.21       106
```

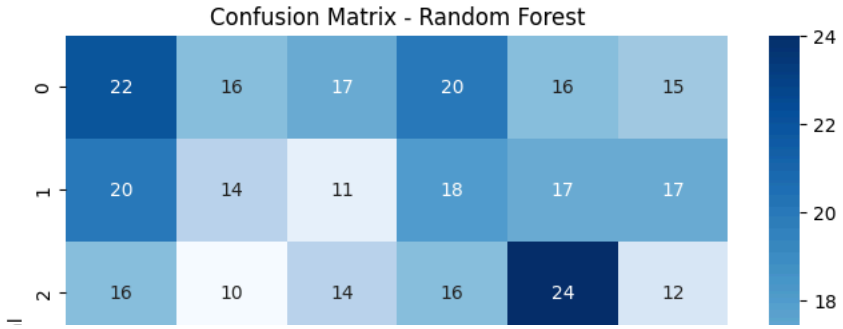
```
                1       0.13      0.06      0.08        97
                2       0.15      0.05      0.08        92
                3       0.16      0.25      0.20       106
                4       0.15      0.18      0.16        99
                5       0.15      0.08      0.10       100

         accuracy                           0.16       600
        macro avg       0.15      0.15      0.14       600
     weighted avg       0.15      0.16      0.14       600


     Training Model: Decision Tree
     Accuracy: 0.1733
                   precision    recall  f1-score   support

                0       0.15      0.14      0.14       106
                1       0.14      0.14      0.14        97
                2       0.16      0.18      0.17        92
                3       0.23      0.22      0.22       106
                4       0.14      0.12      0.13        99
                5       0.22      0.23      0.22       100

         accuracy                           0.17       600
        macro avg       0.17      0.17      0.17       600
     weighted avg       0.17      0.17      0.17       600


     Training Model: Random Forest
     Accuracy: 0.1767
                   precision    recall  f1-score   support

                0       0.19      0.21      0.20       106
                1       0.15      0.14      0.15        97
                2       0.18      0.15      0.16        92
                3       0.20      0.21      0.20       106
                4       0.18      0.20      0.19        99
                5       0.16      0.14      0.15       100

         accuracy                           0.18       600
        macro avg       0.18      0.18      0.17       600
     weighted avg       0.18      0.18      0.18       600
```

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

for name, clf in classifiers.items():
    y_pred = clf.predict(X_test_scaled)
    cm = confusion_matrix(y_test, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
    plt.title(f'Confusion Matrix - {name}')
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

[+]


Confusion Matrix - Logistic Regression
Actual
Predicted
0 1 2 3 4 5
30 10 4 32 20 10
30 6 5 28 22 6
26 9 5 19 18 15
38 5 5 27 21 10
31 7 9 28 18 6
24 9 6 30 23 8
35 30 25 20 15 10 5


Confusion Matrix - Decision Tree
Actual
Predicted
0 1 2 3 4 5
15 18 19 15 16 23
22 14 21 16 8 16
16 13 17 13 19 14
12 18 16 23 18 19
23 21 17 15 12 11
14 14 15 20 14 23
22 20 18 16 14 12 10 8


Confusion Matrix - Random Forest
0 1 2
22 16 17 20 16 15
20 14 11 18 17 17
16 10 14 16 24 12
24 22 20 18