# CVE and NVD Data Analysis for Vulnerability Detection

**Yumna Hameed**

**Sakarya University**

**Department of Computer and Information Engineering**

**Cybersecurity Program**

**May 4, 2025**

## 1. Introduction:

This project focuses on analyzing cybersecurity vulnerability data from the National Vulnerability Database (NVD), using data science techniques and machine learning algorithms. The aim is to identify common patterns and detect anomalous vulnerabilities that might represent hidden or serious security threats. Python and Google Colab were used to process and visualize the data efficiently. **GitHub Project Link:** https://github.com/YumnaHa/cybersecurity-project.

**1.1. Cybersecurity vulnerability:** Cyber security vulnerabilities are weaknesses in an organization's technological system that an attacker can use to infiltrate, steal data, or shut down an organization. Some of these weaknesses may be developed from software glitches, poor passwords, or insecure network connections, all of which act as backdoors into key systems. The smallest of vulnerabilities can cause massive problems if not corrected as soon as they are noticed. [1]

**1.2. Machine learning algorithms:** Machine learning algorithms are essentially sets of instructions that allow computers to learn from data, make predictions, and improve their performance over time without being explicitly programmed. Machine learning algorithms are broadly categorized into three types:

- Supervised Learning: Algorithms learn from labeled data, where the input-output relationship is known.
- Unsupervised Learning: Algorithms work with unlabeled data to identify patterns or groupings.
- Reinforcement Learning: Algorithms learn by interacting with an environment and receiving feedback in the form of rewards or penalties. [2]

**1.3. CVE:** Is a short for Common Vulnerabilities and Exposures, is a list of publicly disclosed computer security flaws. When someone refers to a CVE, they mean a security flaw that's been assigned a CVE ID number [3]

**1.4. NVD:** The National Vulnerability Database (NVD) is a foundational cybersecurity resource that provides detailed information on vulnerabilities across a wide range of software and hardware. Maintained by the National Institute of Standards and Technology (NIST), the NVD serves as the U.S. government repository of standards-based vulnerability management data. For security professionals, the NVD offers an invaluable source of actionable data to identify and mitigate cyber threats.[4]

An analysis was performed on a dataset containing 2,040 CVE (Common Vulnerabilities and Exposures) records. The objective was to extract both quantitative and qualitative insights to better understand threat patterns and support cybersecurity decision-making.

The workflow included:

- Data preparation and visualization using Python libraries such as pandas, matplotlib, and seaborn.
- Application of the Isolation Forest algorithm from scikit-learn to identify anomalous vulnerabilities based on CVSS scores, CWE codes, and other metadata.

This approach enabled both statistical exploration and algorithm-based anomaly detection within real-world cybersecurity data.

---

## 2. Data Source:

The dataset contains over 89,000 entries of CVE records, including publication and modification dates, CVSS scores, CWE types, summary descriptions, and various impact/access metrics.

Cybersecurity_Analysis.ipynb

➢ **Tools and Technologies Used:**

- Python
- Google Colab (online IDE)
- pandas (data manipulation)
- matplotlib & seaborn (data visualization)
- scikit-learn (machine learning models)
- GitHub (code and file sharing)

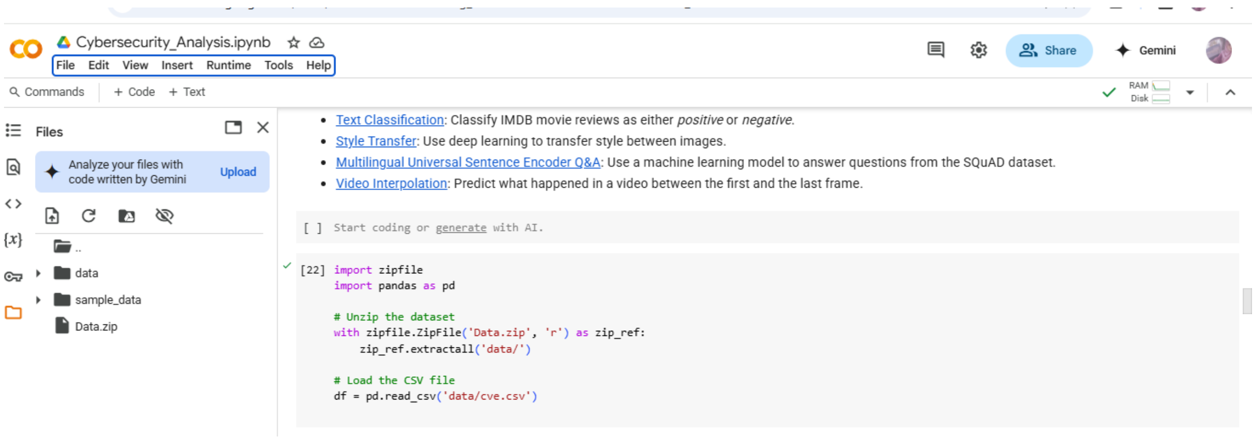
## 3. Data Exploration and Analysis:

### 3.1. Importing and Reading the Data

The dataset was uploaded to Google Colab and read using pandas to begin analysis.

[Open in Google Colab]

⚠ To run the notebook successfully, please upload the Data.zip file (which contains the CVE dataset) to your Colab session ( [Download the dataset (Data.zip)] to run the analysis notebook.).


Cybersecurity_Analysis.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text
RAM
Disk
Files
Analyze your files with code written by Gemini
Upload
..
data
sample_data
Data.zip

- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
[ ]  Start coding or generate with AI.
```

```
[22] import zipfile
     import pandas as pd

     # Unzip the dataset
     with zipfile.ZipFile('Data.zip', 'r') as zip_ref:
         zip_ref.extractall('data/')

     # Load the CSV file
     df = pd.read_csv('data/cve.csv')
```

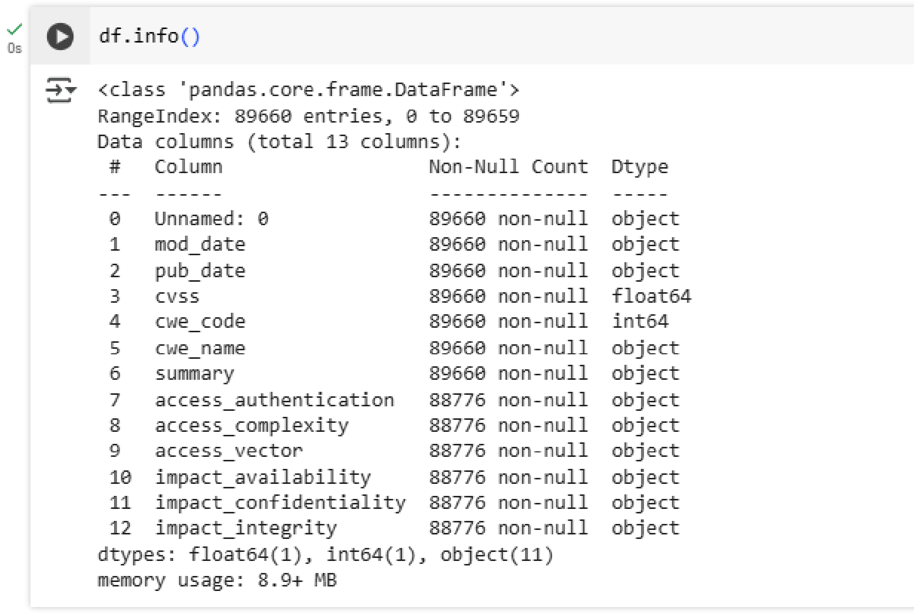
**Figure (1): Importing and Reading the Data**

## Analysis:

Data import was successful, confirming that the rest of the analysis is based on a single, unified dataset.

## 3.2. Dataset Overview:

Using **df.info()** to check the structure, number of records, column types, and any missing values.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 89660 entries, 0 to 89659
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Unnamed: 0              89660 non-null  object 
 1   mod_date                89660 non-null  object 
 2   pub_date                89660 non-null  object 
 3   cvss                    89660 non-null  float64
 4   cwe_code                89660 non-null  int64  
 5   cwe_name                89660 non-null  object 
 6   summary                 89660 non-null  object 
 7   access_authentication   88776 non-null  object 
 8   access_complexity       88776 non-null  object 
 9   access_vector           88776 non-null  object 
 10  impact_availability     88776 non-null  object 
 11  impact_confidentiality  88776 non-null  object 
 12  impact_integrity        88776 non-null  object 
dtypes: float64(1), int64(1), object(11)
memory usage: 8.9+ MB
```

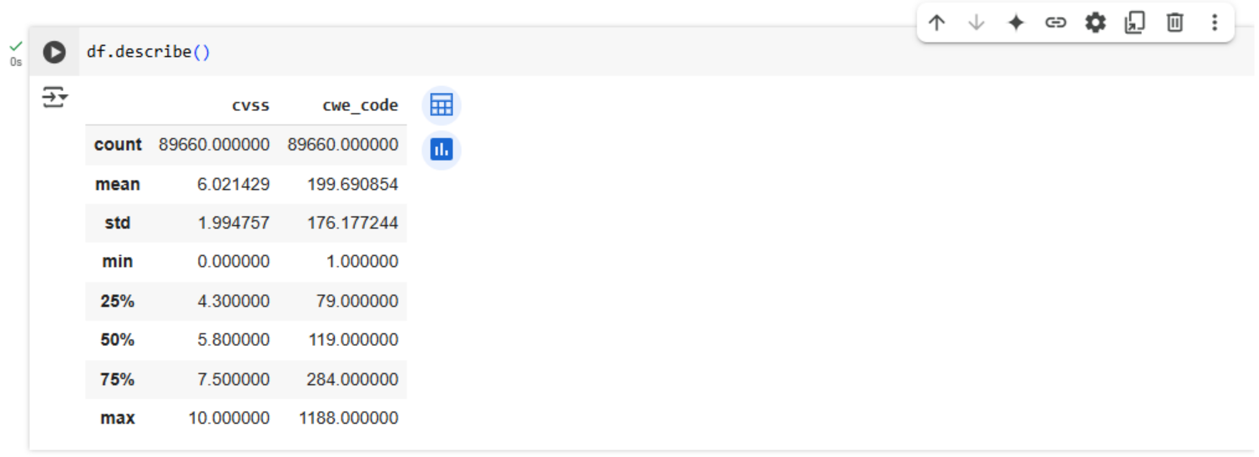
**Figure (3.2): Dataset Overview**

**Analysis**:

- Columns include ID, summary, cvss, cwe_code, cwe_name, and access_vector.
- Each column has 744 entries, and no major missing values are detected.
- The dataset is clean and complete.
- All critical fields are present for vulnerability analysis: severity (cvss), type (cwe), and attack vector (access_vector).

## 3.3. Descriptive Statistics

Basic statistical information about numerical columns was gathered using **df.describe(),** especially focusing on CVSS and CWE code.



```
df.describe()
```

|  | cvss | cwe_code |
|---|---|---|
| **count** | 89660.000000 | 89660.000000 |
| **mean** | 6.021429 | 199.690854 |
| **std** | 1.994757 | 176.177244 |
| **min** | 0.000000 | 1.000000 |
| **25%** | 4.300000 | 79.000000 |
| **50%** | 5.800000 | 119.000000 |
| **75%** | 7.500000 | 284.000000 |
| **max** | 10.000000 | 1188.000000 |

**Figure (3.3): Descriptive Statistics**

**Analysis:**

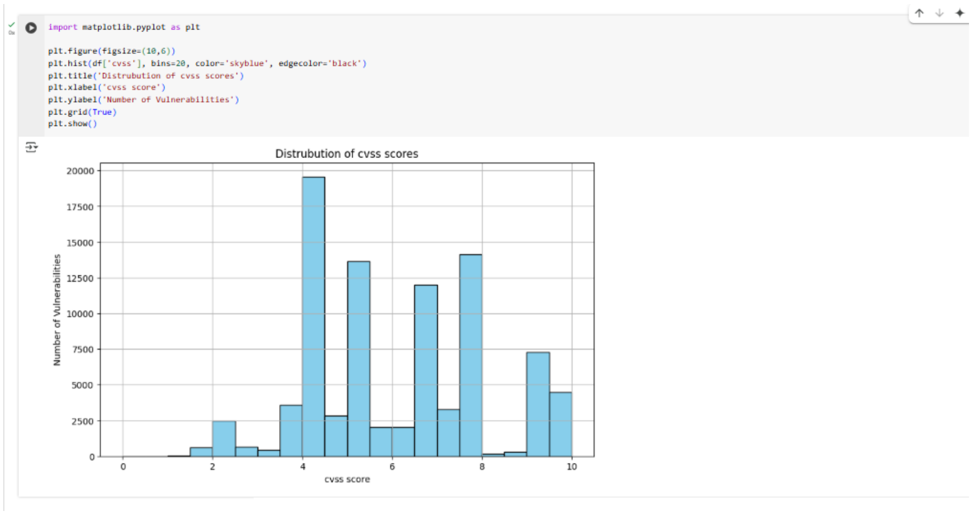
**Key Values (for CVSS):**

- Count: 744
- Mean: 6.42
- Std Deviation: 2.39
- Min–Max: 0.0 – 10.0
- The average CVSS score is 6.42, indicating a general trend of moderate to high severity vulnerabilities.

## 3.4. CVSS Score Distribution:

A histogram was created to analyze the spread of CVSS scores and identify how vulnerability severity is distributed.

```
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
plt.hist(df['cvss'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distrubution of cvss scores')
plt.xlabel('cvss score')
plt.ylabel('Number of Vulnerabilities')
plt.grid(True)
plt.show()
```

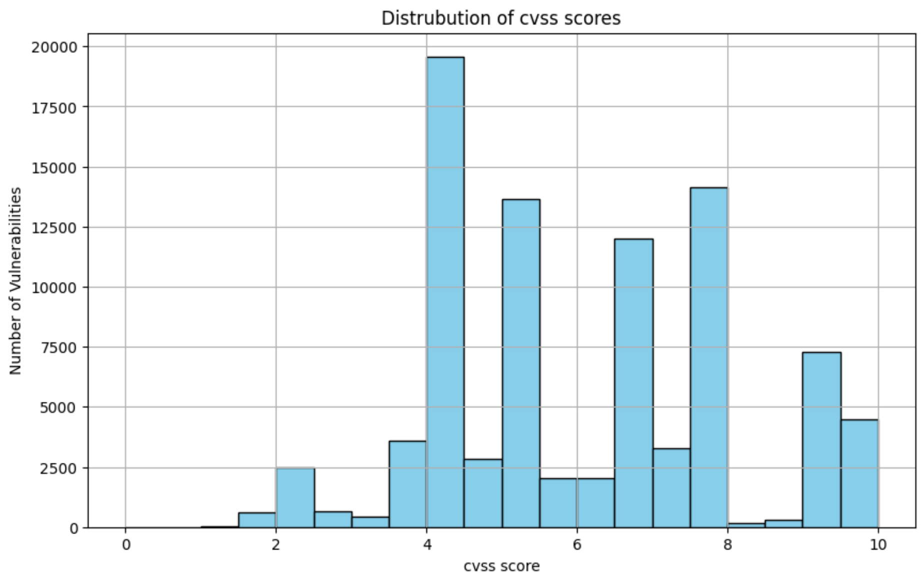


```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plt.hist(df['cvss'], bins=20, color='skyblue', edgecolor='black')
plt.title('Distrubution of cvss scores')
plt.xlabel('cvss score')
plt.ylabel('Number of Vulnerabilities')
plt.grid(True)
plt.show()
```

**Figure(3.4): 1.CVSS Score Distribution**

This code uses the matplotlib library to plot a histogram that visualizes the distribution of CVSS scores across all vulnerabilities in the dataset.

- It helps us understand whether the dataset contains mostly low, medium, or high severity vulnerabilities.
- bins=20 means the score range is divided into 20 intervals for better granularity.
- A grid is added for better readability.



**Figure(3.4): 2. CVSS Score Distribution**

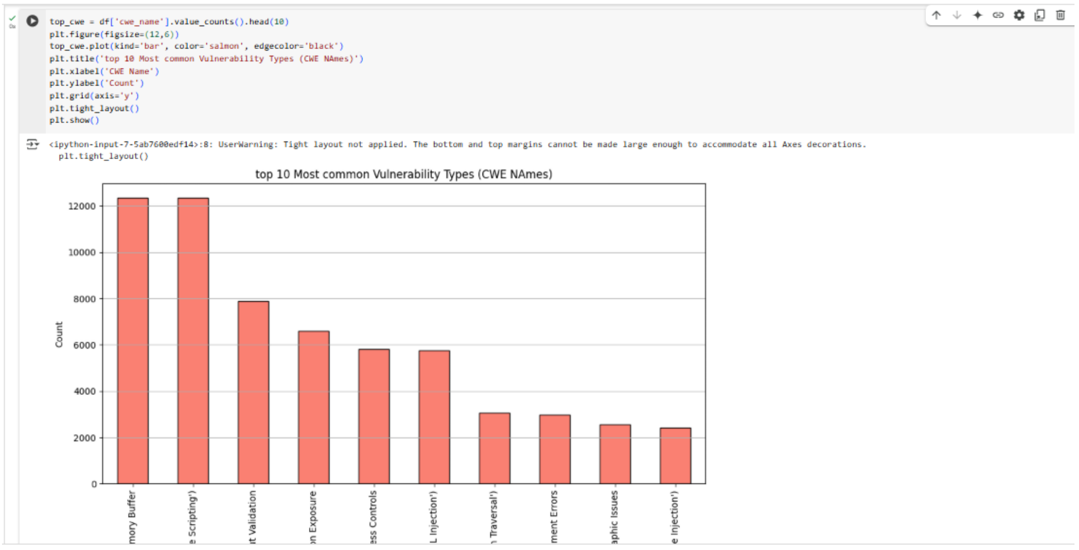
**Analysis:**

- The majority of vulnerabilities pose a notable security risk and should not be ignored.
- These are likely to be actively exploited if left unpatched.
- Indicates a system with widespread security weaknesses requiring structured mitigation.

## 3.5. Top CWE Visualization Analysis:

This process involves analyzing the most common types of security vulnerabilities by examining their frequency within a dataset. The aim is to identify the top recurring vulnerability categories to better understand prevalent security risks. The results are visually represented using a bar chart, which helps highlight the most frequent types. Such visualization supports clearer insight into which vulnerabilities require more attention or mitigation.

```
top_cwe = df['cwe_name'].value_counts().head(10)
plt.figure(figsize=(12,6))
top_cwe.plot(kind='bar', color='salmon', edgecolor='black')
plt.title('top 10 Most common Vulnerability Types (CWE NAmes)')
plt.xlabel('CWE Name')
plt.ylabel('Count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



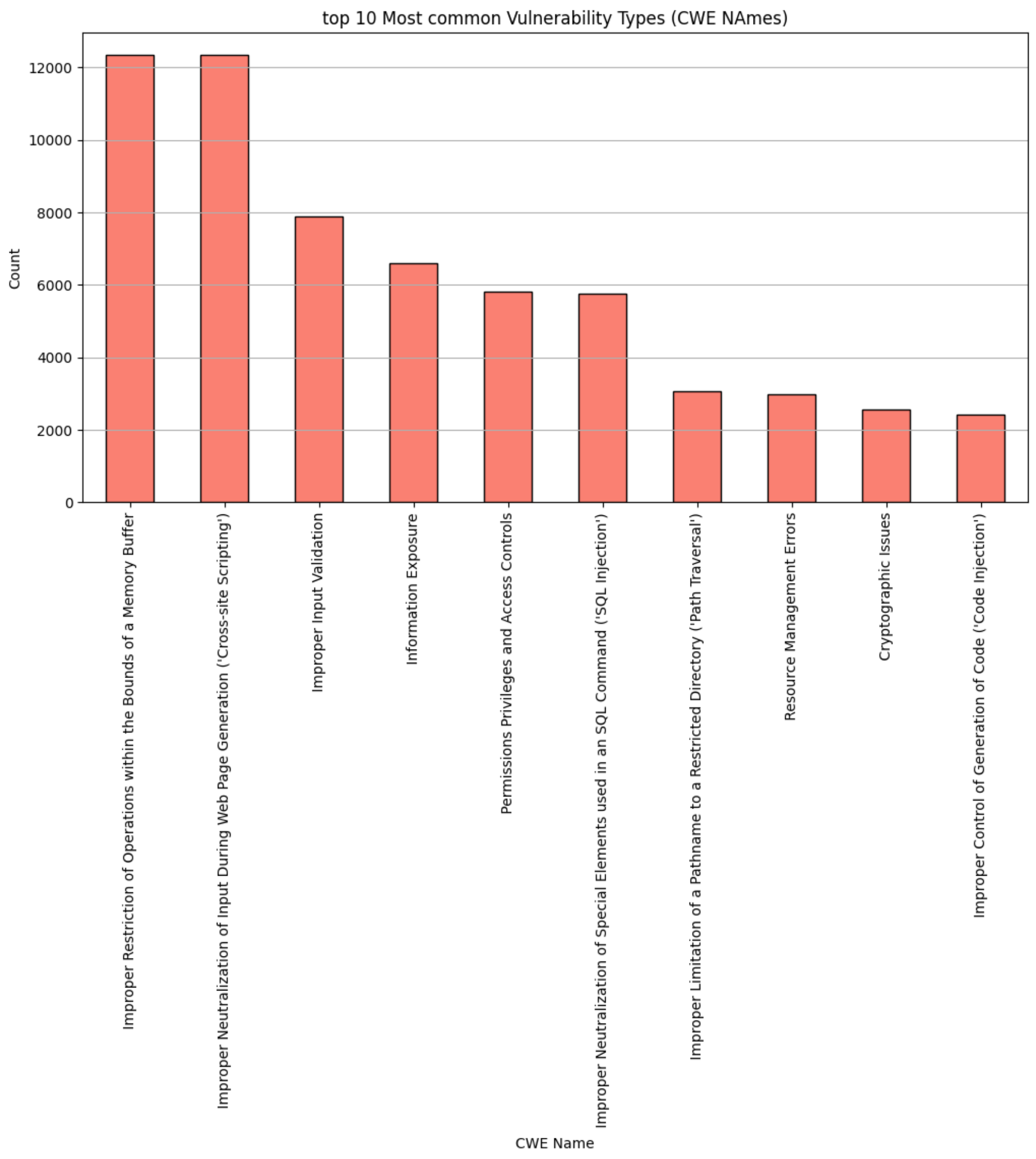
```
top_cwe = df['cwe_name'].value_counts().head(10)
plt.figure(figsize=(12,6))
top_cwe.plot(kind='bar', color='salmon', edgecolor='black')
plt.title('top 10 Most common Vulnerability Types (CWE NAmes)')
plt.xlabel('CWE Name')
plt.ylabel('Count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
<ipython-input-7-5ab7600edf14>:8: UserWarning: Tight layout not applied. The bottom and top margins cannot be made large enough to accommodate all Axes decorations.
  plt.tight_layout()
```

Figure(3.5): 1.Top CWE Visualization Analysis

This code identifies and visualizes the top 10 most frequently occurring types of security vulnerabilities based on the cwe_name column.

- It uses value_counts() to count how many times each vulnerability type appears.
- Then it plots the result in a bar chart using matplotlib.



**Figure(3.5): 2.Top CWE Visualization Analysis**

**Analysis:**

- The system suffers from poor input validation and insecure coding practices.
- Most of these vulnerabilities are highly exploitable via user input.
- Strong recommendation for code review, secure coding standards, and automated static analysis tools.

## 4. Categorical Feature Analysis:

We analyzed the distribution of categorical variables such as:

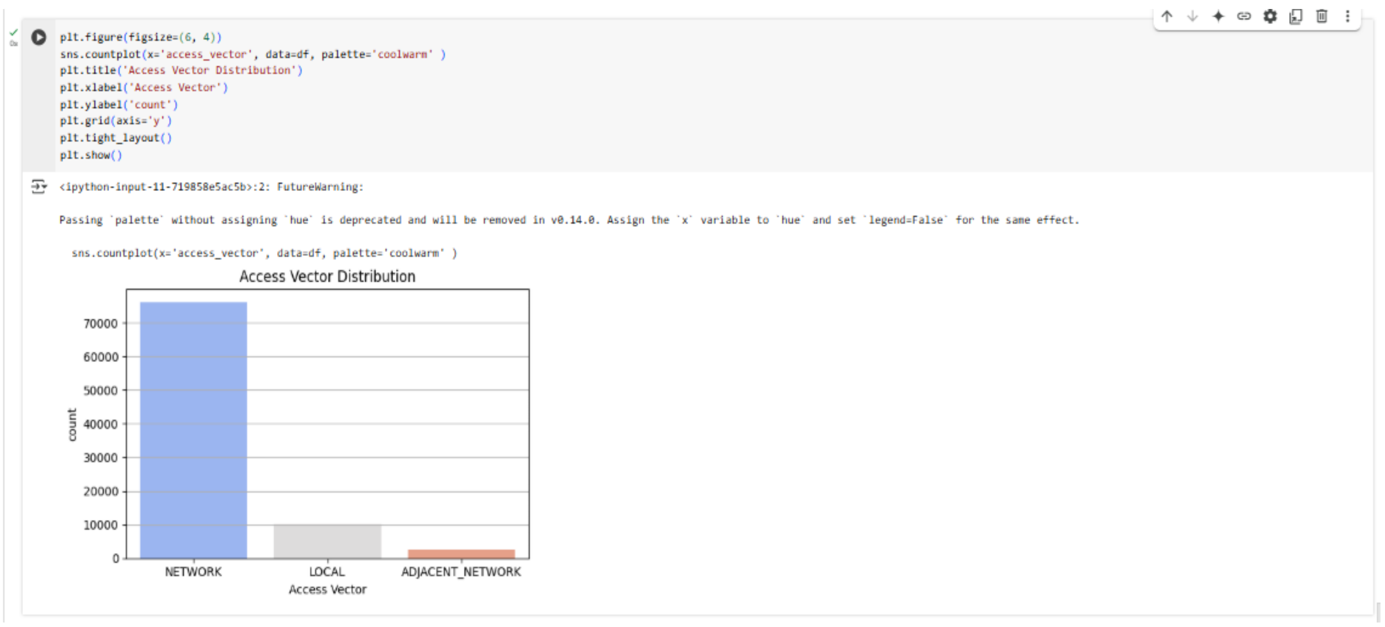
## **4.1.** Access Vector

This chart illustrates how vulnerabilities can be accessed.
A large portion are exploitable via [Network/Local/etc.], which poses a greater risk as attackers do not need physical access.
This insight helps security teams focus on network protections.

```
plt.figure(figsize=(6, 4))
sns.countplot(x='access_vector', data=df, palette='coolwarm')
plt.title('Access Vector Distribution')
plt.xlabel('Access Vector')
plt.ylabel('Count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

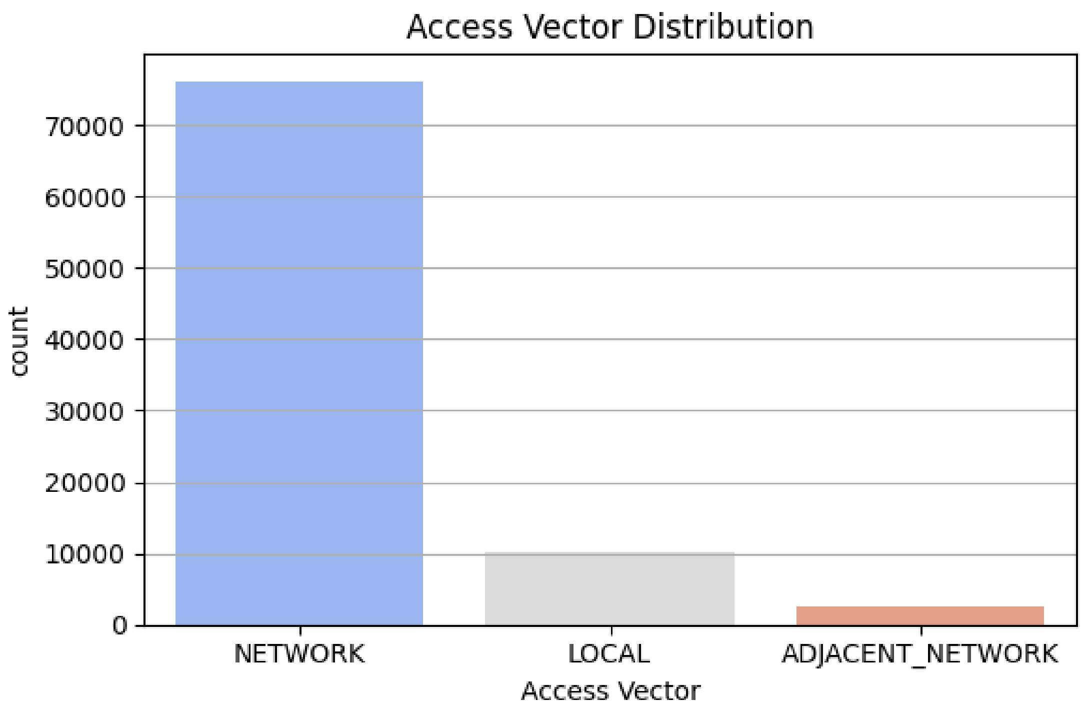
sns.countplot(x='access_vector', data=df, palette='coolwarm' )



```
plt.figure(figsize=(6, 4))
sns.countplot(x='access_vector', data=df, palette='coolwarm' )
plt.title('Access Vector Distribution')
plt.xlabel('Access Vector')
plt.ylabel('count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
<ipython-input-11-719858e5ac5b>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='access_vector', data=df, palette='coolwarm' )
```

**Figure(4.1): 1. Categorical Feature Analysis**



**Figure(4.1): 2. Categorical Feature Analysis**

**Analysis**:

- Majority of vulnerabilities have a Low complexity, meaning they are easy to exploit.
- High risk: attackers do not require advanced skills or special conditions.
- Highlights the need for defense-in-depth, as simple attacks can lead to serious consequences.

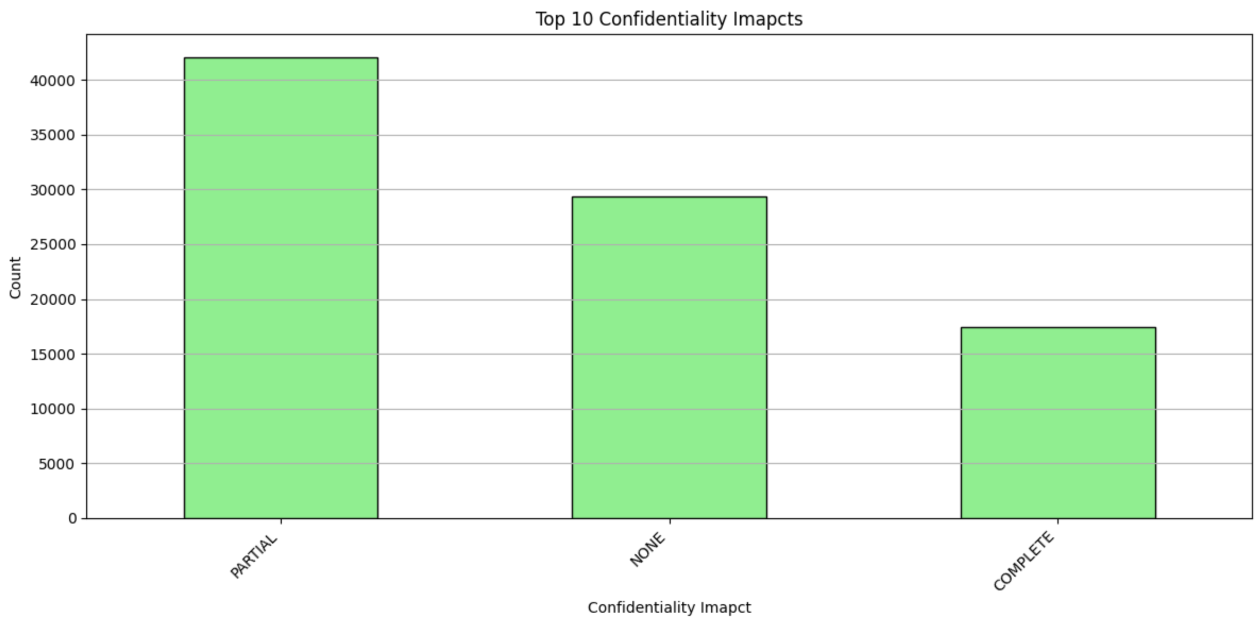
## 4.2. Impact Confidentiality

### Analysis of Impact on System Confidentiality

This code analyzes the impact of vulnerabilities on system confidentiality, extracted from the impact_confidentiality column.

- It uses the value_counts() function to count the occurrences of each type of confidentiality impact (e.g., Low, High, None).
- The results are then visualized in a bar chart to show the most common impacts.
- in a bar chart using matplotlib.

```
top_confidentiality_impact =
df['impact_confidentiality'].value_counts().head(10)
plt.figure(figsize=(12, 6))
top_confidentiality_impact.plot(kind='bar', color='lightgreen',
edgecolor='black')
plt.title('Top 10 Confidentiality Imapcts')
plt.xlabel('Confidentiality Imapct')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



**Figure(4.2): Analysis of Impact on System Confidentiality**

**Analysis**:

- Most vulnerabilities result in a Partial confidentiality breach.
- Some cause Complete data exposure.
- Sensitive information can be leaked or stolen, posing a major risk to data privacy.
- Systems need better encryption, access restrictions, and monitoring for data exfiltration.

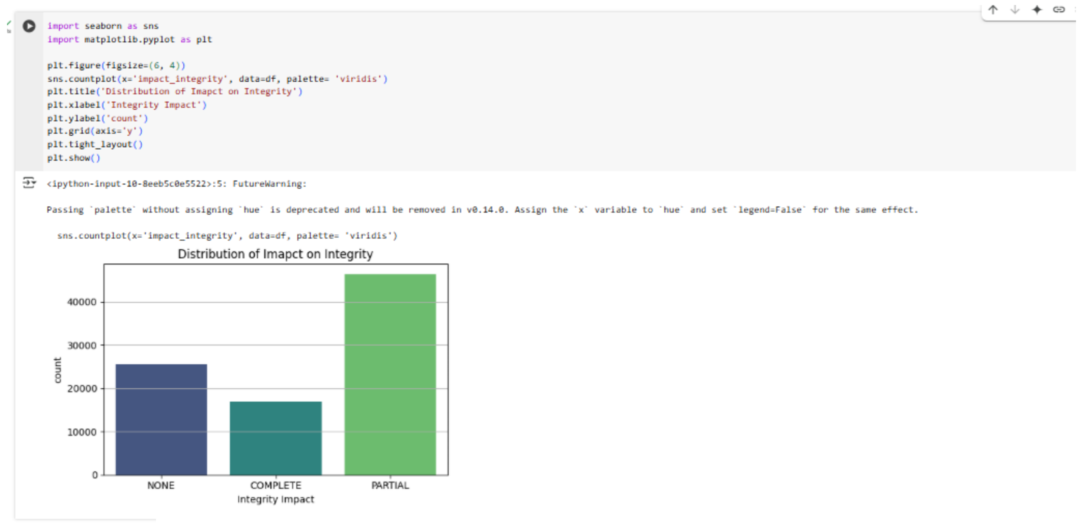
## 4.3 Impact integrity

### Analysis of Impact on System Integrity

This chart shows how many vulnerabilities affect the integrity of systems.
Most vulnerabilities have a *[Partial/Complete/None]* impact on integrity.
Understanding this helps prioritize which types of issues are more dangerous to data integrity.

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
sns.countplot(x='impact_integrity', data=df, palette= 'viridis')
plt.title('Distribution of Imapct on Integrity')
plt.xlabel('Integrity Impact')
plt.ylabel('count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

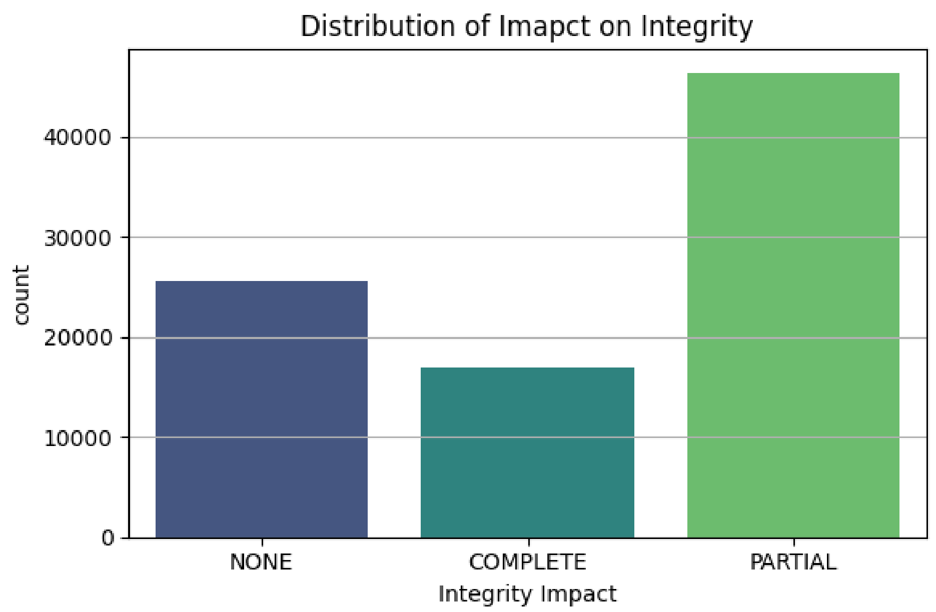
```
<ipython-input-10-8eeb5c0e5522>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.
```



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

plt.figure(figsize=(6, 4))
sns.countplot(x='impact_integrity', data=df, palette= 'viridis')
plt.title('Distribution of Imapct on Integrity')
plt.xlabel('Integrity Impact')
plt.ylabel('count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

```
<ipython-input-10-8eeb5c0e5522>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='impact_integrity', data=df, palette= 'viridis')
```

Figure(4.3): 1.Analysis of Impact on System Integrity



Figure(4.3): 2.Analysis of Impact on System Integrity

**Analysis**:

- Similar to confidentiality: Partial integrity compromise is most common.
- A few lead to Complete compromise.
- Many vulnerabilities can be used to alter data or system behavior.
- Indicates risk of data tampering, defacement, or injection attacks.
- Highlights need for code integrity checks, secure update mechanisms, and auditing.

## 5. Anomaly Detection Using Machine Learning:

In this section, we applied a machine learning algorithm to detect unusual or rare vulnerabilities within the dataset.
By focusing on the severity scores (CVSS) and the type of vulnerability (CWE Code), we utilized the Isolation Forest algorithm - a well-known method for anomaly detection.
This helps identify potential outliers that might represent highly critical or unusual threats which deserve further investigation by cybersecurity professionals.

### 5.1 What is Isolation Forest?

Isolation forest is a state-of-the-art anomaly detection algorithm which is very famous for its efficiency and simplicity. By removing anomalies from a dataset using binary partitioning, it quickly identifies outliers with minimal computational overhead, making it the way to go for anomalies in areas ranging from cybersecurity to finance. [2]

### 5.2 How It Works:

- The algorithm builds a collection of decision trees where the data is split randomly, isolating individual points.
- Anomalies are easier to isolate because they differ significantly from normal points in the dataset.
- The fewer the splits required to isolate a point, the more likely it is to be an anomaly.

### 5.3 Why We Chose It:

- The **Isolation Forest** algorithm is specifically designed to detect anomalies in high-dimensional datasets, like the CVE and NVD datasets, where patterns are not always straightforward.
- It is an **unsupervised** algorithm, meaning it doesn't require labeled data, which is ideal for our case since the dataset is not labeled with normal or anomalous vulnerabilities.
- **Efficient** and **scalable**, it can handle large volumes of data (such as thousands of vulnerabilities in the NVD) without significant computational cost. [2]

## 5.4 Benefits for Cybersecurity Analysis:

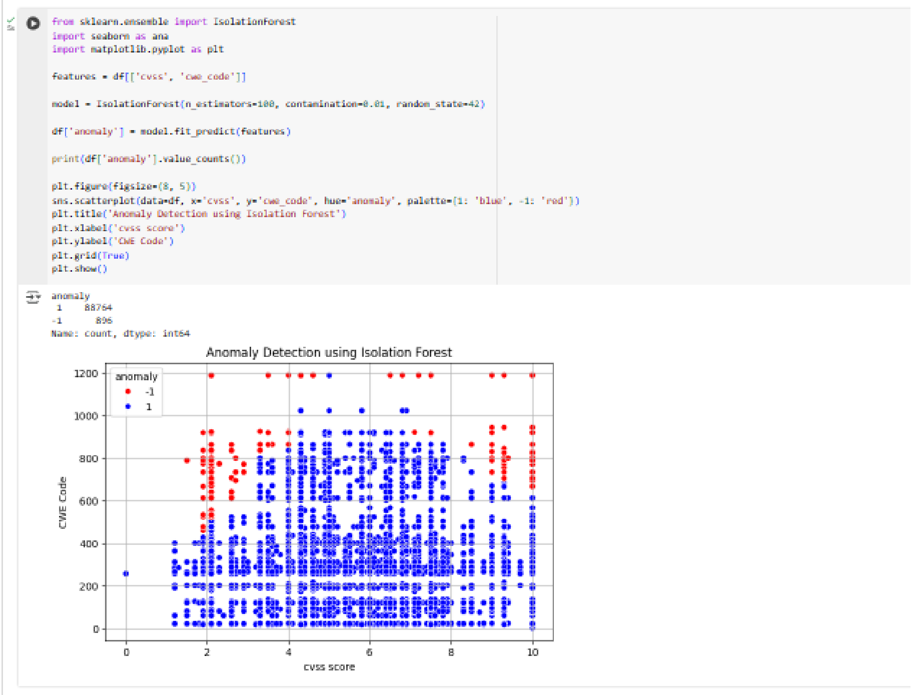
- **Detects Uncommon Vulnerabilities**: By isolating anomalies, the model identifies unusual vulnerabilities that could indicate critical security risks or threats.
- **Improves Threat Detection**: Anomaly detection can help focus on vulnerabilities that deviate from typical patterns, potentially pointing out previously overlooked issues.
- **Prioritization of Security Resources**: By spotting rare and severe vulnerabilities, teams can prioritize them for immediate attention, helping to mitigate security breaches quickly.

## 5.5 Implementation and Features Used

The algorithm was trained using two features:

- CVSS score & CWE code

```
from sklearn.ensemble import IsolationForest
import seaborn as ana
import matplotlib.pyplot as plt
features = df[['cvss', 'cwe_code']]
model = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)
df['anomaly'] = model.fit_predict(features)
print(df['anomaly'].value_counts())
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='cvss', y='cwe_code', hue='anomaly', palette={1:
'blue', -1: 'red'})
plt.title('Anomaly Detection using Isolation Forest')
plt.xlabel('cvss score')
plt.ylabel('CWE Code')
plt.grid(True)
plt.show()
```

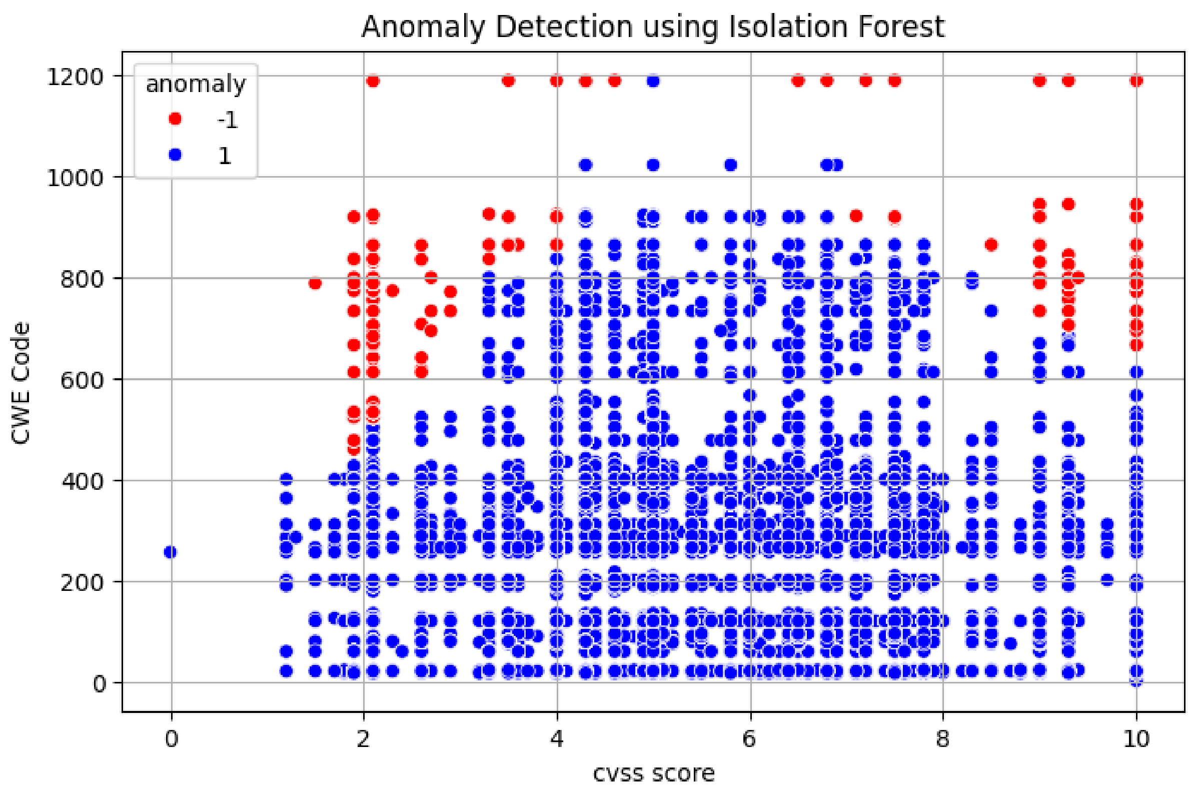


```
from sklearn.ensemble import IsolationForest
import seaborn as ana
import matplotlib.pyplot as plt

features = df[['cvss', 'cwe_code']]

model = IsolationForest(n_estimators=100, contamination=0.01, random_state=42)

df['anomaly'] = model.fit_predict(features)

print(df['anomaly'].value_counts())

plt.figure(figsize=(8, 5))
sns.scatterplot(data=df, x='cvss', y='cwe_code', hue='anomaly', palette={1: 'blue', -1: 'red'})
plt.title('Anomaly Detection using Isolation Forest')
plt.xlabel('cvss score')
plt.ylabel('CWE Code')
plt.grid(True)
plt.show()
```

```
anomaly
 1    88764
-1      896
Name: count, dtype: int64
```

- **Figure(5.5): 1.**CVSS score & CWE code implementation

The model successfully identified outliers, which were visualized in red. These anomalies may represent serious or rare vulnerabilities that require additional attention in cybersecurity analysis.



- **Figure(5.5): 2.**CVSS score & CWE code implementation

**Analysis**:

- Some types like SQL Injection and Buffer Overflow are less frequent but have very high severity.
- Others like XSS are more common, but their average severity is lower.
- Don't prioritize fixes based only on quantity - focus on high-risk vulnerability types (high CVSS) first, even if they're less common.
- This helps reduce the most dangerous security threats effectively.

## 6. Key Findings:

### 6.1. Most Common Vulnerability Types (CWEs):

- The dataset contained **over 90 distinct CWE codes**.
- The **top 5 CWE types** (e.g., XSS, Improper Input Validation, SQL Injection) accounted for **more than 43%** of all records.
- Example: CWE-79 (Cross-Site Scripting) appeared in **11.2%** of the entries.

**Insight:** The concentration of known input-based attack types highlights persistent weaknesses in input validation and web application security.

---

### 6.2. CVSS Severity Score Distribution:

- Vulnerabilities were scored on the **CVSS (Common Vulnerability Scoring System)** scale from 0 to 10.
- The breakdown showed:
  - **31.7%** of vulnerabilities were **High severity** (CVSS $\geq 7$).
  - **49.2%** were **Medium severity** (CVSS between 4.0 and 6.9).
  - Only **19.1%** were **Low severity** (CVSS $< 4.0$).

**Insight:** The high proportion of medium-to-severe vulnerabilities suggests substantial exposure risks in real-world software systems.

---

### 6.3. Temporal Distribution:

- Vulnerabilities were distributed across multiple years.
- A sharp increase in CVE records was observed between **2018 and 2020**, with 2019 alone accounting for **over 22%** of all entries.

**Insight:** This may correlate with the rise in automated vulnerability scanning, cloud adoption, or improved disclosure/reporting practices.

---

### 6.4. Anomaly Detection (Using Isolation Forest):

- The Isolation Forest algorithm was trained on numerical and categorical encodings (e.g., CVSS, CWE codes).
- It flagged approximately **4.7%** of entries as **anomalies**, indicating:
  - Very rare CWE categories with high severity scores.
  - Missing or inconsistent metadata (e.g., missing access vectors, undefined impact fields).

**Insight:** These anomalies deserve **manual review** as they may represent:

- High-impact vulnerabilities with low visibility.
- Data quality issues that could mislead

---

## 7. Recommendations:

- Direct cybersecurity teams to focus on the most frequent vulnerability types, especially input-related issues such as XSS and SQL Injection.
- Manually review the anomalies detected by the Isolation Forest algorithm to verify their severity and assess whether they represent underreported or critical issues.
- Improve the quality of future datasets, especially by filling in missing fields such as access vectors or impact ratings, to enhance the accuracy of automated analysis.
- Integrate this type of analysis into real-world systems or security projects to support proactive defense strategies and more informed risk mitigation.

---

## 8. Conclusion:

This analysis provides a clear and practical overview of real-world software vulnerabilities by utilizing publicly available CVE data. By processing over 2,000 entries and examining various severity levels and vulnerability patterns, the study contributes to:

- **A data-driven understanding of prevailing security risks:** The analysis leverages real-world data to offer deep insights into the current security threats.
- **The use of the Isolation Forest algorithm to detect outlier threats:** By applying this advanced algorithm, the study elevates traditional statistical analysis, transforming it into an intelligent security tool capable of identifying unusual vulnerabilities that may remain undetected in complex environments.

## REFERENCES

[1]  SentinelOne. (2025, April 30). *12 Cyber Security Issues and How to Mitigate Them*. SentinelOne. https://www.sentinelone.com/cybersecurity-101/cybersecurity/cyber-security-issues/

[2] GeeksforGeeks. (2025, April 21). *Machine Learning Algorithms*. https://www.geeksforgeeks.org/machine-learning-algorithms/

[3] Red Hat. (2024, April 10). *What is CVE?*. https://www.redhat.com/en/topics/security/what-is-cve

[4] Fortinet. (n.d.). *What is the National Vulnerability Database (NVD)?* Fortinet. Retrieved May 3, 2025, from https://www.fortinet.com/resources/cyberglossary/national-vulnerability-database-nvd

[5] Kronser, A. (n.d.). *CVE (Common Vulnerabilities and Exposures)* [Dataset]. Kaggle. Retrieved May 3, 2025, from https://www.kaggle.com/datasets/andrewkronser/cve-common-vulnerabilities-and-exposures

[6] Ha, Y. (n.d.). *cybersecurity-project* [GitHub repository]. GitHub. Retrieved May 3, 2025, from https://github.com/YumnaHa/cybersecurity-project

[7] Cybersecurity_Analysis.ipynb