NLP
Semester Project

Google Colab Link : Click Here

Group X

"Automated Information Extraction from NIST Databases Using NLP Techniques "

Supervisor : Dr. Gaurav Sharma

Final Presentation

Team Members:

Ashish Singh
(LCS2021016)

Meet Savla
(LCS2021010)

Satvik Dubey
(LCS2021017)

Vikrant Palle
(LCS2021011)

Gopal Singh Chauhan
(LCS2021025)

PROBLEM DEFINITION & UNDERSTANDING

- In today's digital era, the vast volumes of data stored in specialized repositories like the National Institute of Standards and Technology (NIST) databases are crucial for research and development across various fields. However, manually extracting meaningful information from these extensive datasets can be a complex and error-prone task.

- Automated Information Extraction, powered by Natural Language Processing (NLP) techniques, offers a solution to this challenge by streamlining the process of data retrieval and interpretation. By leveraging NLP, we can automate the extraction of valuable insights from unstructured and semi-structured data with greater accuracy and efficiency.

In this presentation, we will cover:

- Problem Definition and Understanding.
- Data Preparation and Analysis
- Model Design and Implementation
- Training and Tuning
- Evaluation and Results
- Discussion and Conclusions

PROBLEM DEFINITION & UNDERSTANDING

The goal of this project is to automate the extraction and preprocessing of vulnerability information from the National Institute of Standards and Technology (NIST) databases. By employing Natural Language Processing (NLP) techniques, the project aims to streamline the process of data retrieval, enhance the quality of extracted information, and facilitate further analysis.

Implementation Details:

Data Collection:

- **Source:** The data is fetched from the NIST National Vulnerability Database (NVD) API.

- **More specifically, the API Used :** https://services.nvd.nist.gov/rest/json/cves/2.0

- **Method:** The API provides detailed information about vulnerabilities including **CVE IDs**, **descriptions**, and associated metrics.

IMPLEMENTATION OVERVIEW & DATA STRUCTURE

- **Pagination:** The code handles pagination to fetch a large volume of data efficiently. It retrieves records in batches, managing API limits and delays to ensure reliable data retrieval.

Structure of API :


```
"cve": {
  "id": "CVE-1999-0095",
  "sourceIdentifier": "cve@mitre.org",
  "published": "1988-10-01T04:00:00.000",
  "lastModified": "2019-06-11T20:29:00.263",
  "vulnStatus": "Modified",
  "cveTags": [],
```


Common Vulnerabilities and Exposures (CVE) Data: The CVE Program's primary purpose is to uniquely identify vulnerabilities and to associate specific versions of code bases (e.g., software and shared libraries) to those vulnerabilities.

DATA PREPARATION & ANALYSIS

The use of CVEs ensures that two or more parties can confidently refer to a CVE identifier (ID) when discussing or sharing information about a unique vulnerability.

Description in CVEs


```
"descriptions": [
  {
    "lang": "en",
    "value": "The debug command in Sendmail is enabled, allowing attackers to execute commands as root."
  },
  {
    "lang": "es",
    "value": "El comando de depuración de Sendmail está activado, permitiendo a atacantes ejecutar comandos como root."
  }
],
```


The **Common Vulnerability Scoring System (CVSS)** is a method used to supply a qualitative measure of severity. CVSS is not a measure of risk.

CVSS v2.0 Ratings

Severity	Severity Score Range
Low	0.0-3.9
Medium	4.0-6.9
High	7.0-10.0

STRUCTURE OF OBTAINED DATA

CVSS Metrics :


```
"metrics": {
  "cvssMetricV2": [
    {
      "source": "nvd@nist.gov",
      "type": "Primary",
      "cvssData": {
        "version": "2.0",
        "vectorString": "AV:N/AC:L/Au:N/C:C/I:C/A:C",
        "accessVector": "NETWORK",
        "accessComplexity": "LOW",
        "authentication": "NONE",
        "confidentialityImpact": "COMPLETE",
        "integrityImpact": "COMPLETE",
        "availabilityImpact": "COMPLETE",
        "baseScore": 10
      },
      "baseSeverity": "HIGH",
      "exploitabilityScore": 10,
      "impactScore": 10,
      "acInsufInfo": false,
      "obtainAllPrivilege": true,
      "obtainUserPrivilege": false,
      "obtainOtherPrivilege": false,
      "userInteractionRequired": false
    }
  ]
},
```


- The **Confidentiality Impact - Complete :** indicates exploiting the vulnerability results in a complete breach of confidentiality (all information is disclosed).

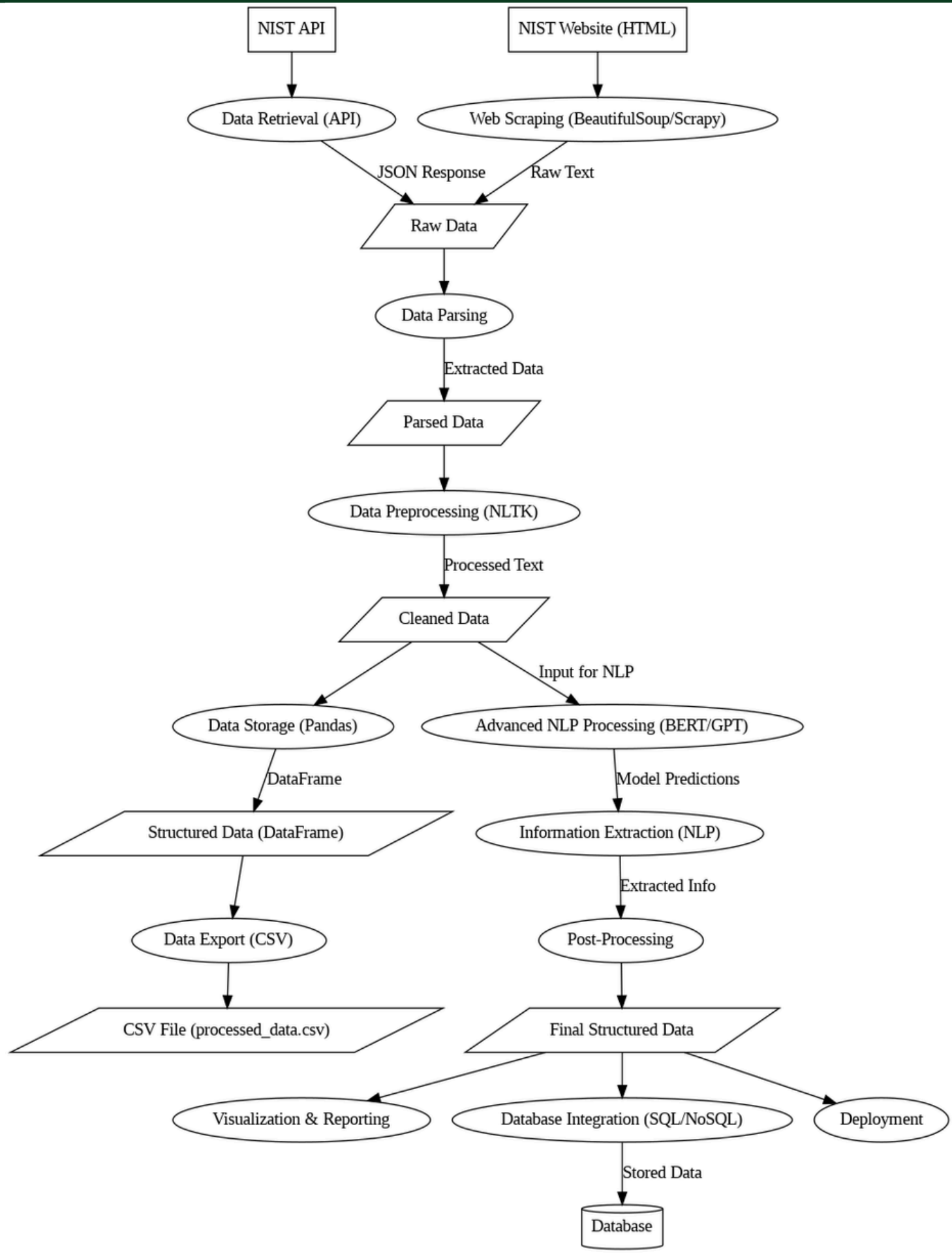
CVSS METRICS

- **Access Vector (AV):**
  - "accessVector": "NETWORK"
  - This indicates that the vulnerability is exploitable from a remote network location.
- **Access Complexity (AC):**
  - "accessComplexity": "LOW"
  - The vulnerability is easy to exploit, with no complex conditions needed.
- **Authentication (Au):**
  - "authentication": "NONE"
  - No authentication is required to exploit the vulnerability.
- **Confidentiality Impact (C):**
  - "confidentialityImpact": "COMPLETE"
  - A successful exploit would result in complete confidentiality breach, meaning all data is accessible to the attacker.
- **Integrity Impact (I):**
  - "integrityImpact": "COMPLETE"
  - A successful exploit would completely compromise the integrity of the system, allowing full modification of data.

CVSS METRICS

- **Availability Impact (A):**
  - "availabilityImpact": "COMPLETE"
  - A successful exploit would completely compromise system availability, rendering the system unusable.
- **Base Score:**
  - "baseScore": 10
  - This is the overall severity score for the vulnerability, ranging from 0 to 10. A score of 10 indicates a critical vulnerability.
- **Exploitability Score:**
  - "exploitabilityScore": 10
  - This score reflects how easy it is to exploit the vulnerability. A score of 10 means it is very easy to exploit.
- **ObtainAllPrivilege:**
  - "obtainAllPrivilege": true
  - Indicates that exploiting the vulnerability allows the attacker to obtain all privileges on the affected system.

DATAFLOW-DIAGRAM



DATA RETRIEVAL

Extracted a total of 21 attributes/fields

Fetching CVE data from the NVD API :

```
# Loop to fetch data from the NVD API
while cur_index < total_results:
    results_per_page = min(max_results_per_page, total_results - cur_index)  # Ca
    url = f"{base_url}/?resultsPerPage={results_per_page}&startIndex={cur_index}"

    try:
        response = requests.get(url)  # Make the API request
        response.raise_for_status()  # Raise an exception for HTTP errors
        data = response.json()  # Parse JSON response

        # Process each vulnerability in the response
        for vulnerability in data.get("vulnerabilities", []):
            cve = vulnerability.get("cve", {})
            cve_id = cve.get("id", "")
            published_date = cve.get("published", "")
            year = published_date.split("-")[0] if published_date else "Unknown"
            description = ""
            os_product = []
            attack_vector = []
            access_vector = []
            severity = "Unknown"
```

This code fetches and processes vulnerability data from the NVD API, extracting details like **CVE ID**, **publication date**, **operating systems**, **attack vectors**, and **severity level.**

DATA RETRIEVAL

Extract operating systems/products from configurations :

```
# Extract operating systems/products from configurations
os_product = extract_os_and_attack_vectors(cve.get("configurations"))
```

Extract severity, attack vector, and access vector from CVSS metrics :

```
metrics = cve.get("metrics", {}).get("cvssMetricV2", [])
if metrics:
    severity = metrics[0].get("baseSeverity", "Unknown")
    attack_vector = metrics[0].get("cvssData", {}).get("vectorString", "Unknown")
    access_vector = metrics[0].get("cvssData", {}).get("accessVector", "Unknown")
```

Creating a DataFrame from the collected CVE data :

```
# Create a DataFrame from the collected data
df = pd.DataFrame({
    "CVE_ID": cve_ids,
    "Description": descriptions,
    "Published_Year": published_years,
    "Operating_System/Product": os_products,
    "Attack_Vector": attack_vectors,
    "Access Vector": access_vectors,
    "Severity": severities
})

# Display the total number of records fetched
print(f"Total records fetched: {len(df)}")
```

OUTPUT AND VISUALIATION OF DATA

	CVE_ID	Description
0	CVE-1999-0095	The debug command in Sendmail is enabled, allo...
1	CVE-1999-0082	CWD ~root command in ftpd allows root access.
2	CVE-1999-1471	Buffer overflow in passwd in BSD based operati...
3	CVE-1999-1122	Vulnerability in restore in SunOS 4.0.3 and ea...
4	CVE-1999-1467	Vulnerability in rcp on SunOS 4.0.x allows rem...

	Published_Year	Operating_System/Product
0	1988	[sendmail:5.58]
1	1988	[ftp:*, ftpcd:*]
2	1989	[bsd:4.2, bsd:4.3]
3	1989	[sunos:*, sunos:4.0, sunos:4.0.1]
4	1989	[sunos:4.0, sunos:4.0.1, sunos:4.0.2, sunos:4....
...	...	...

Processed Description :

	Processed_Description
0	debug command sendmail enabled allowing attack...
1	cwd command ftpd allows root access
2	buffer overflow passwd bsd based operating sys...
3	vulnerability restore sunos earlier allows loc...
4	vulnerability rcp sunos allows remote attacker...

DATA EXTRACTED - VISUALIZATION

CVE_ID

“DESCRIPTION”

"PUBLISHED_YEAR"

"PUBLISHED_DATES"

"OPERATING_SYSTEM/PRODUCT"

"ATTACK_VECTOR”

"ACCESS_VECTOR"

"SEVERITY"

"AUTHENTICATION"

"IMPACT_SCORE"

"BASE_SCORE"

"OBTAIN_ALL_PRIVILEGE"

"USER_INTERACTION_REQUIRED"

"EXPLOITABILITY_SCORE"

"CVSS_VECTOR_STRING"

"AVAILABILITY_IMPACT"

"INTEGRITY_IMPACT"

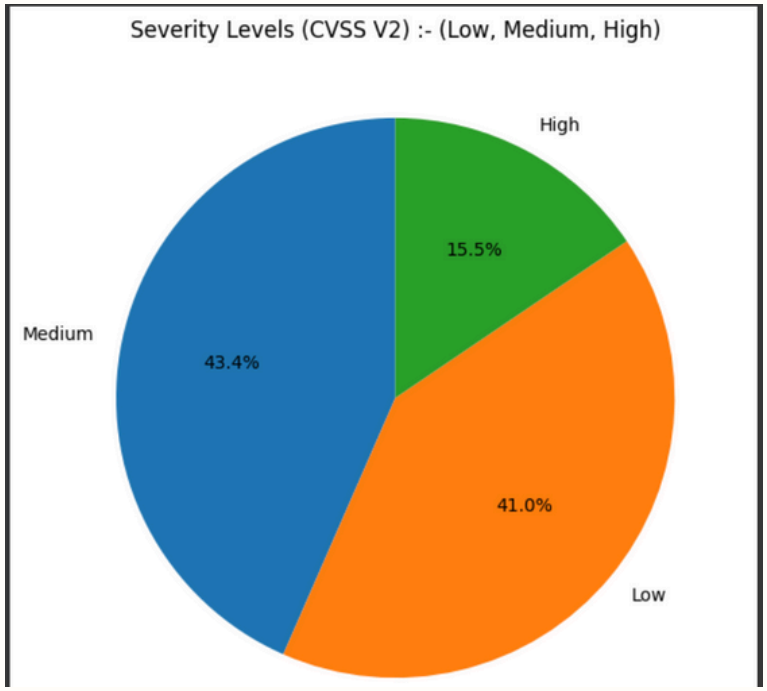
"CONFIDENTIALITY_IMPACT"

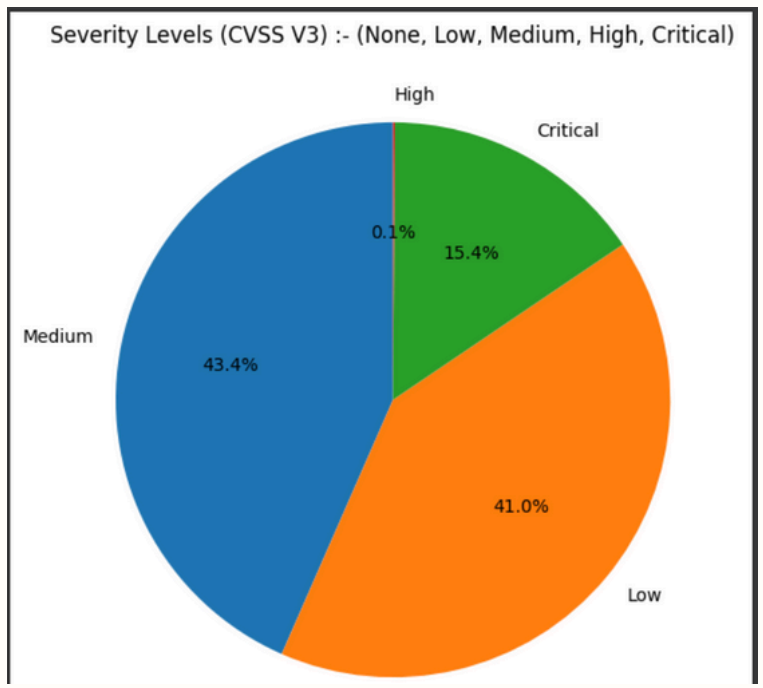
"OBTAIN_USER_PRIVILEGES"

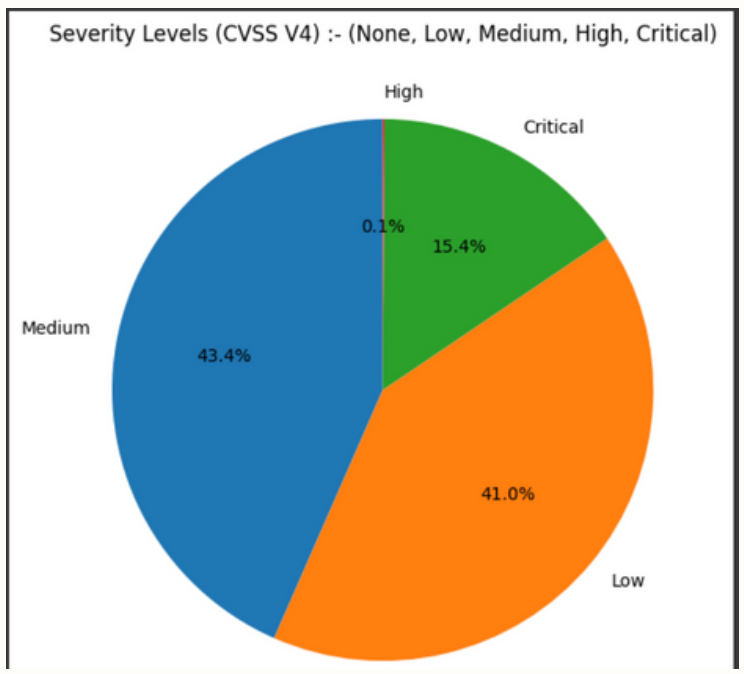
"VULNERABILITY_STATUS"

"LAST_MODIFIED_DATE"

EXPLORATORY DATA ANALYSIS (EDA)

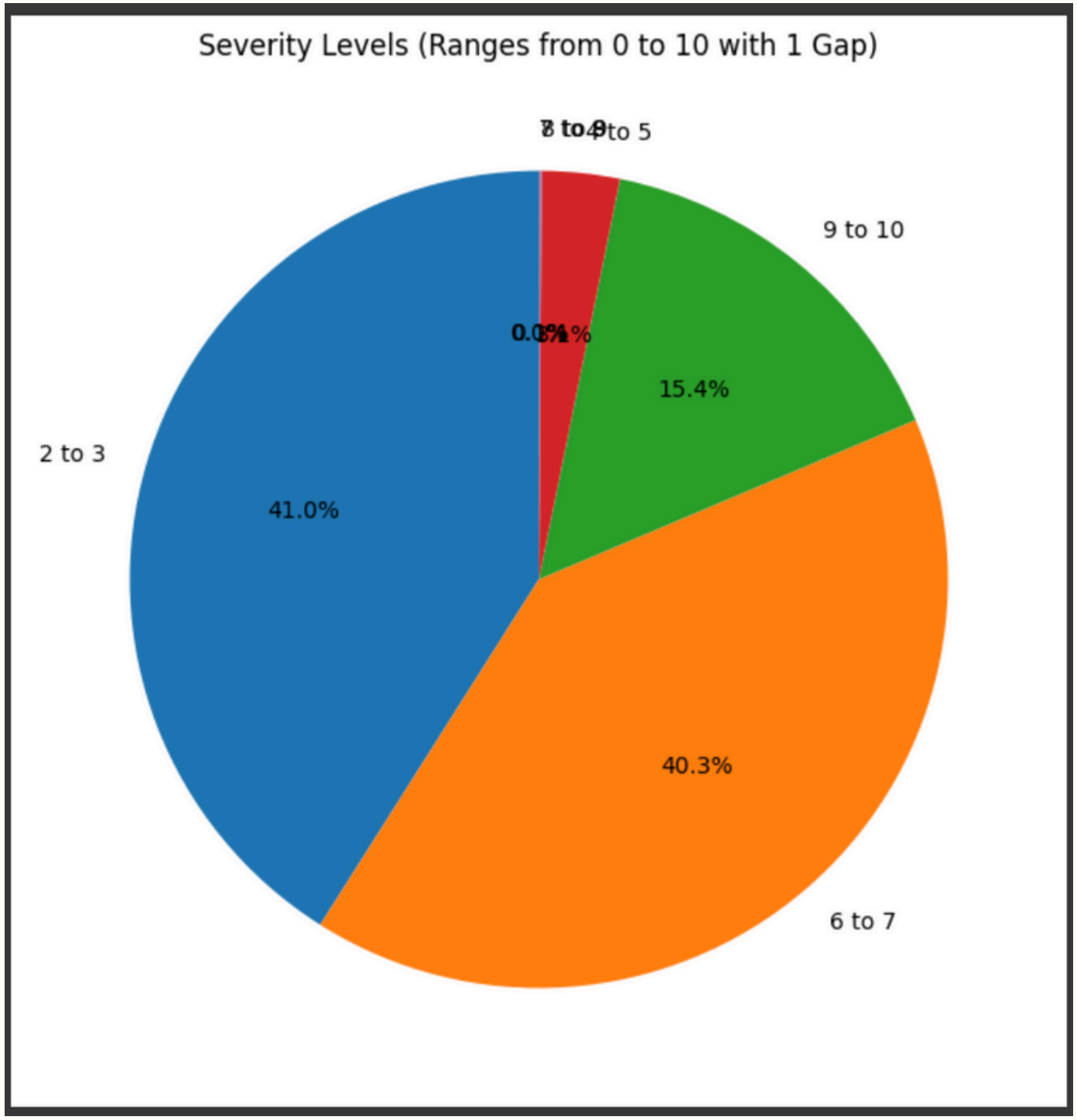






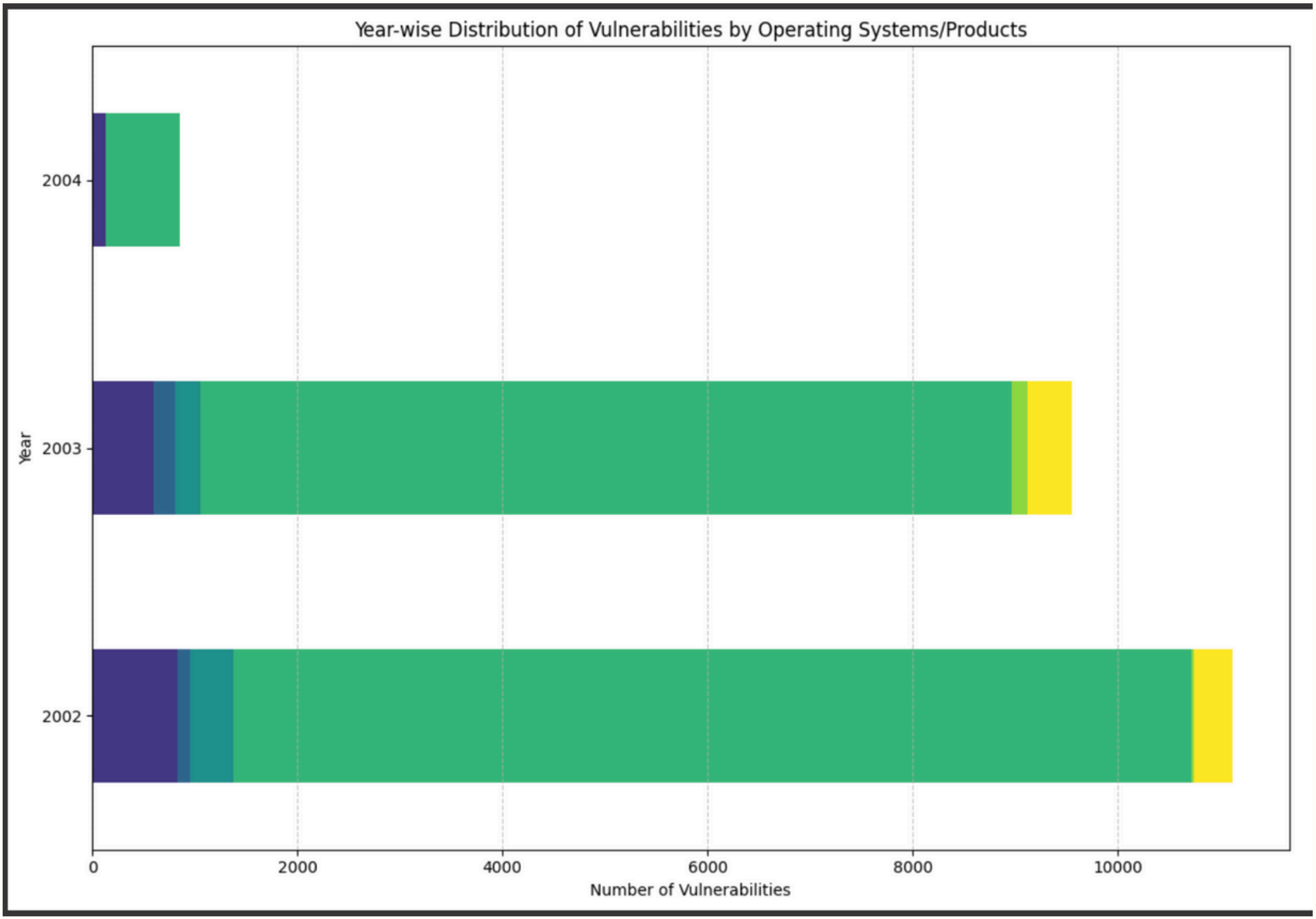
Distribution of Severity Levels according to v2, v3 and v4 versions in the CVE data.

EXPLORATORY DATA ANALYSIS (EDA)



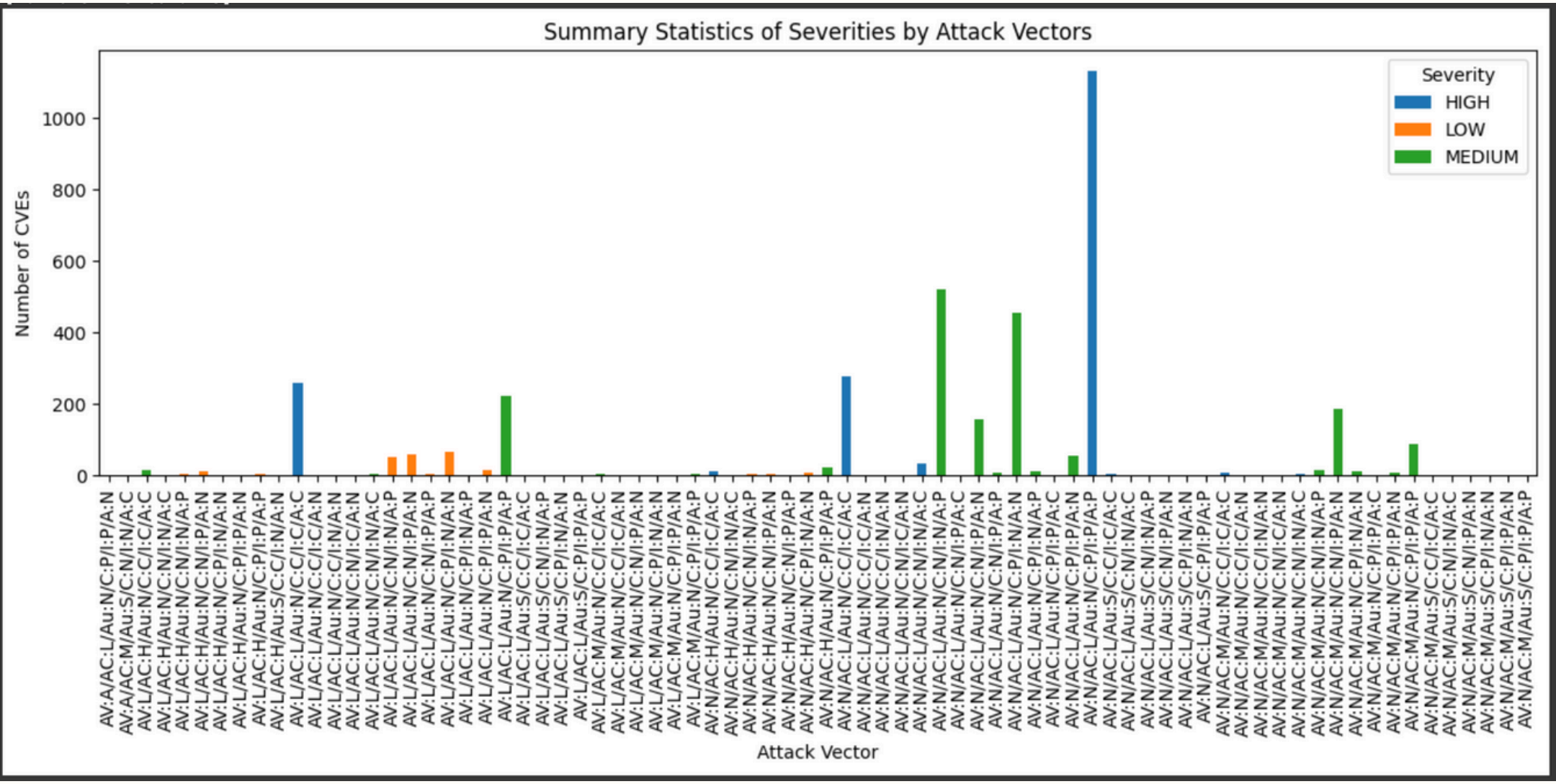
Severity Levels (Ranges from 0 to 10 with 1 Gap)

EXPLORATORY DATA ANALYSIS (EDA)



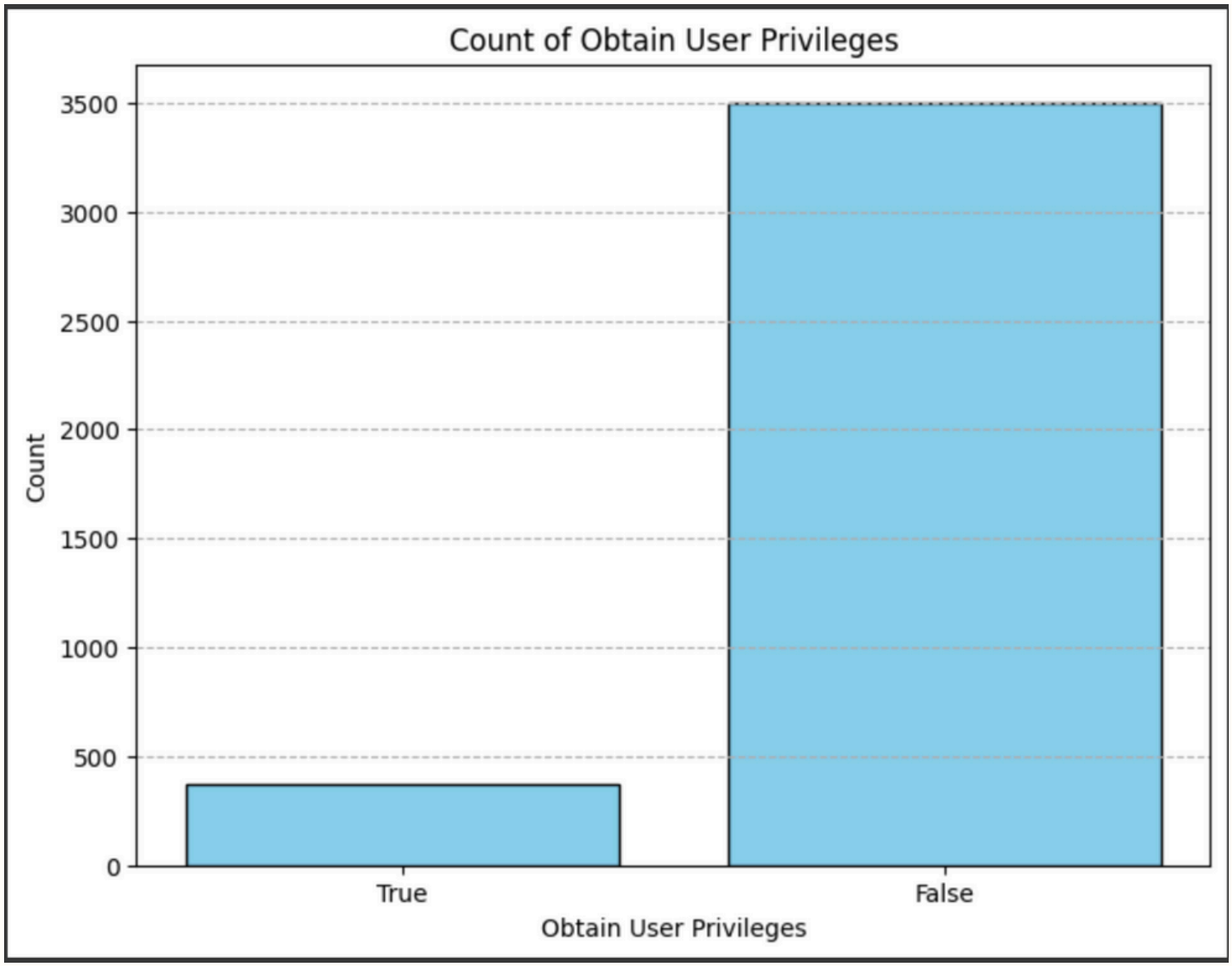
Year-wise Distribution of Vulnerabilities by Operating Systems/Products

EXPLORATORY DATA ANALYSIS (EDA)



Summary Statistics of Severities by Attack Vectors

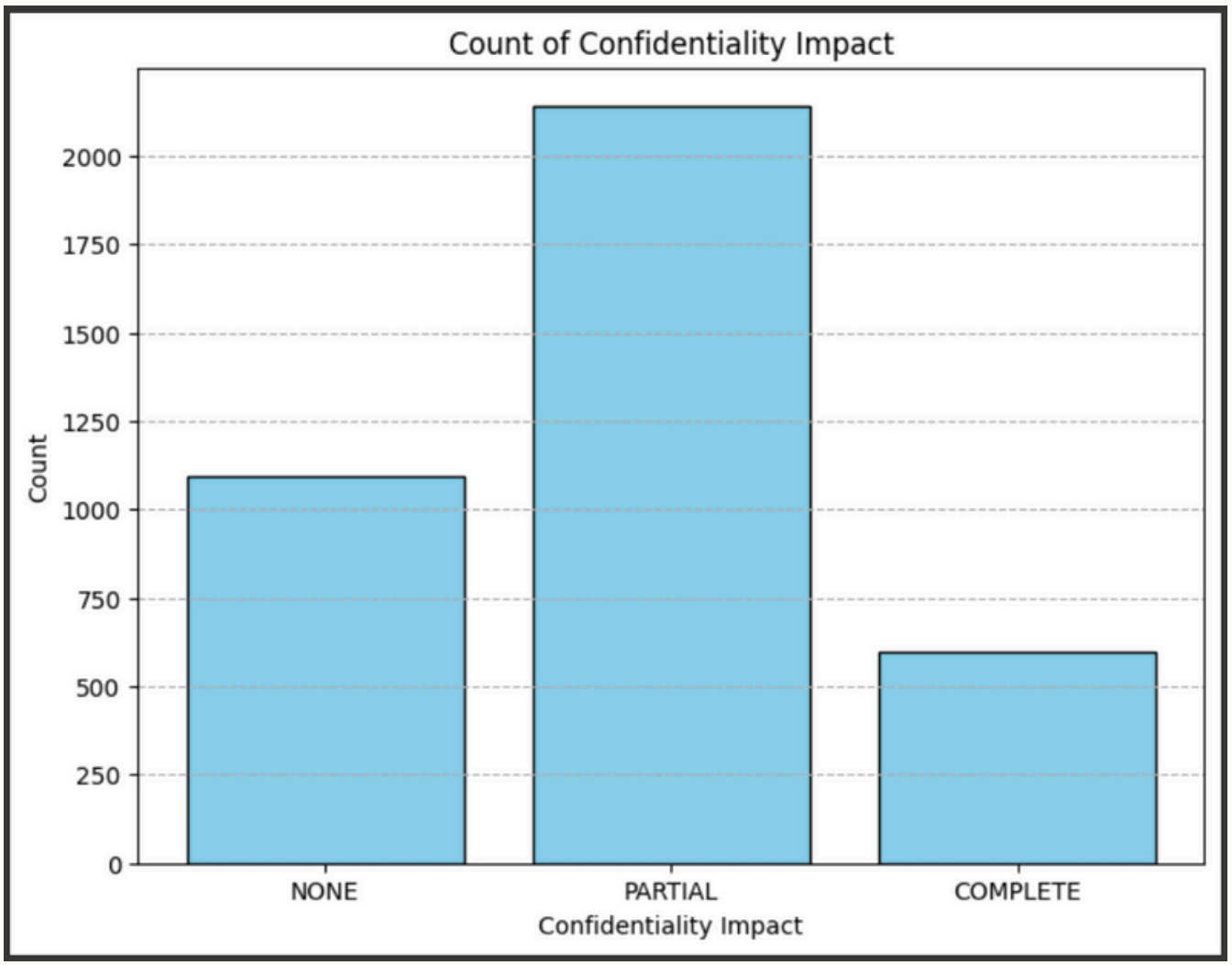
EXPLORATORY DATA ANALYSIS (EDA)



Obtain User Privileges	Count
True	368
False	3501

Count of Obtain User Privileges

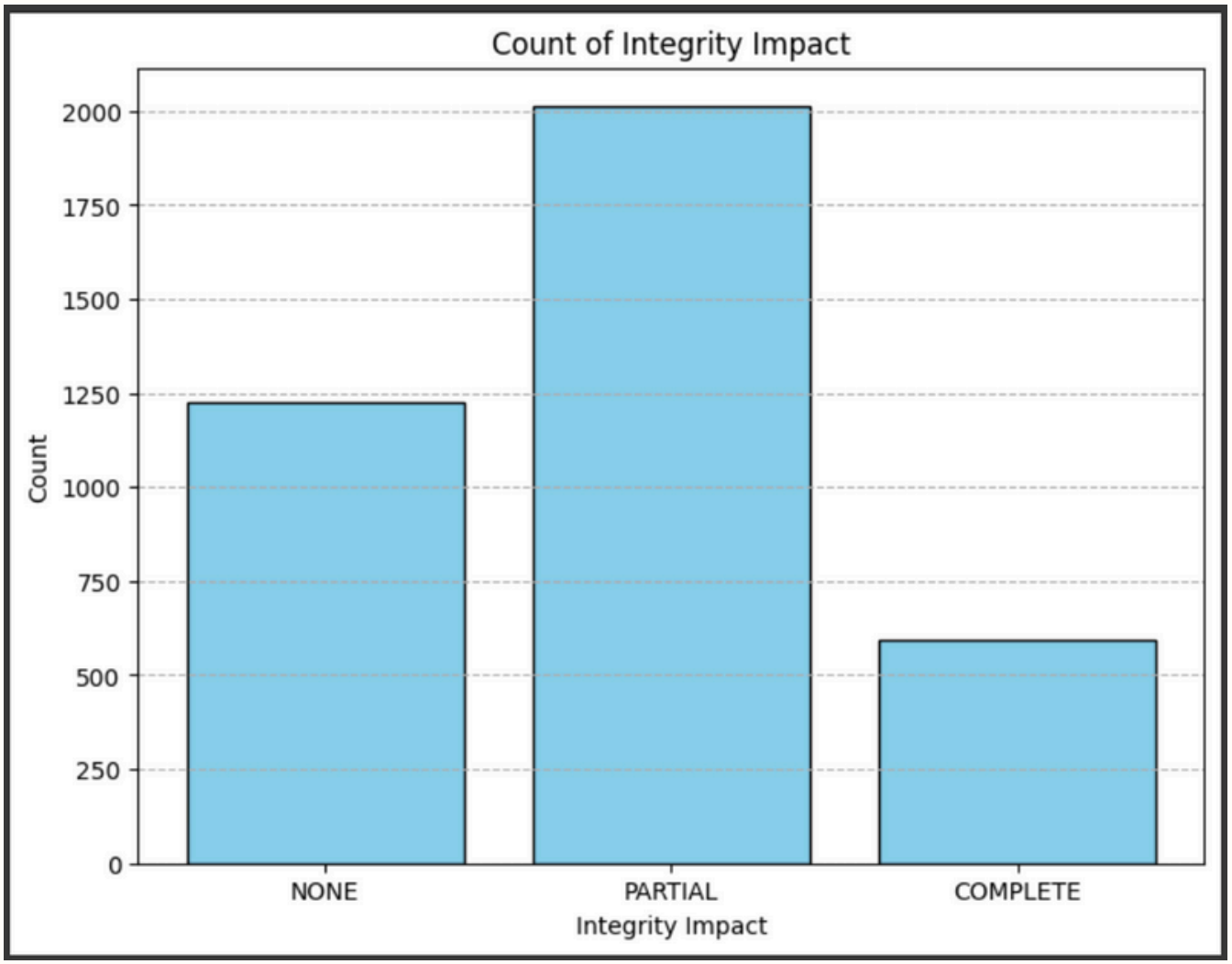
EXPLORATORY DATA ANALYSIS (EDA)



Confidentiality Impact	Count
NONE	1095
PARTIAL	2142
COMPLETE	597

Count of Confidentiality Impact

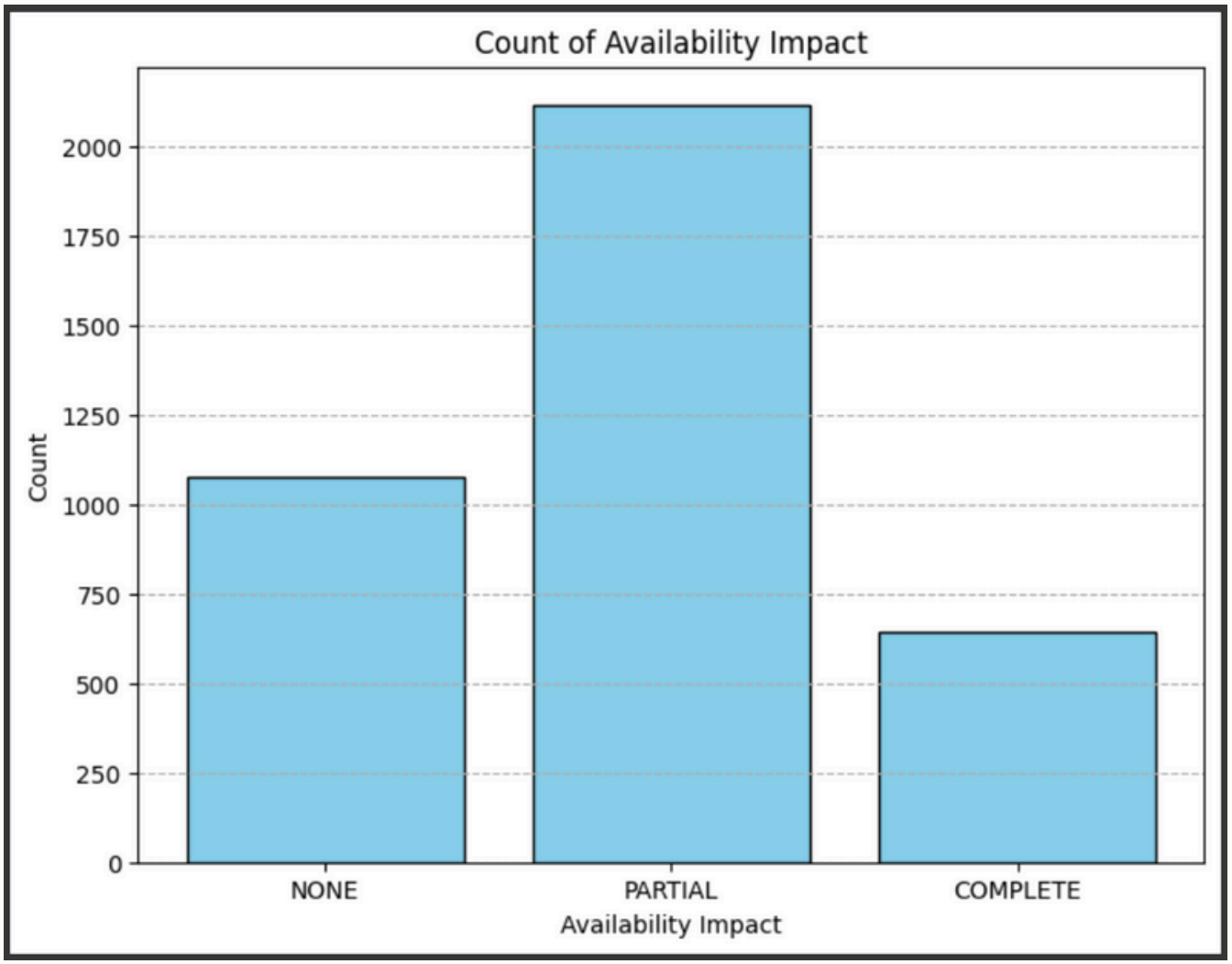
EXPLORATORY DATA ANALYSIS (EDA)



Integrity Impact	Count
NONE	1224
PARTIAL	2014
COMPLETE	596

Count of Integrity Impact

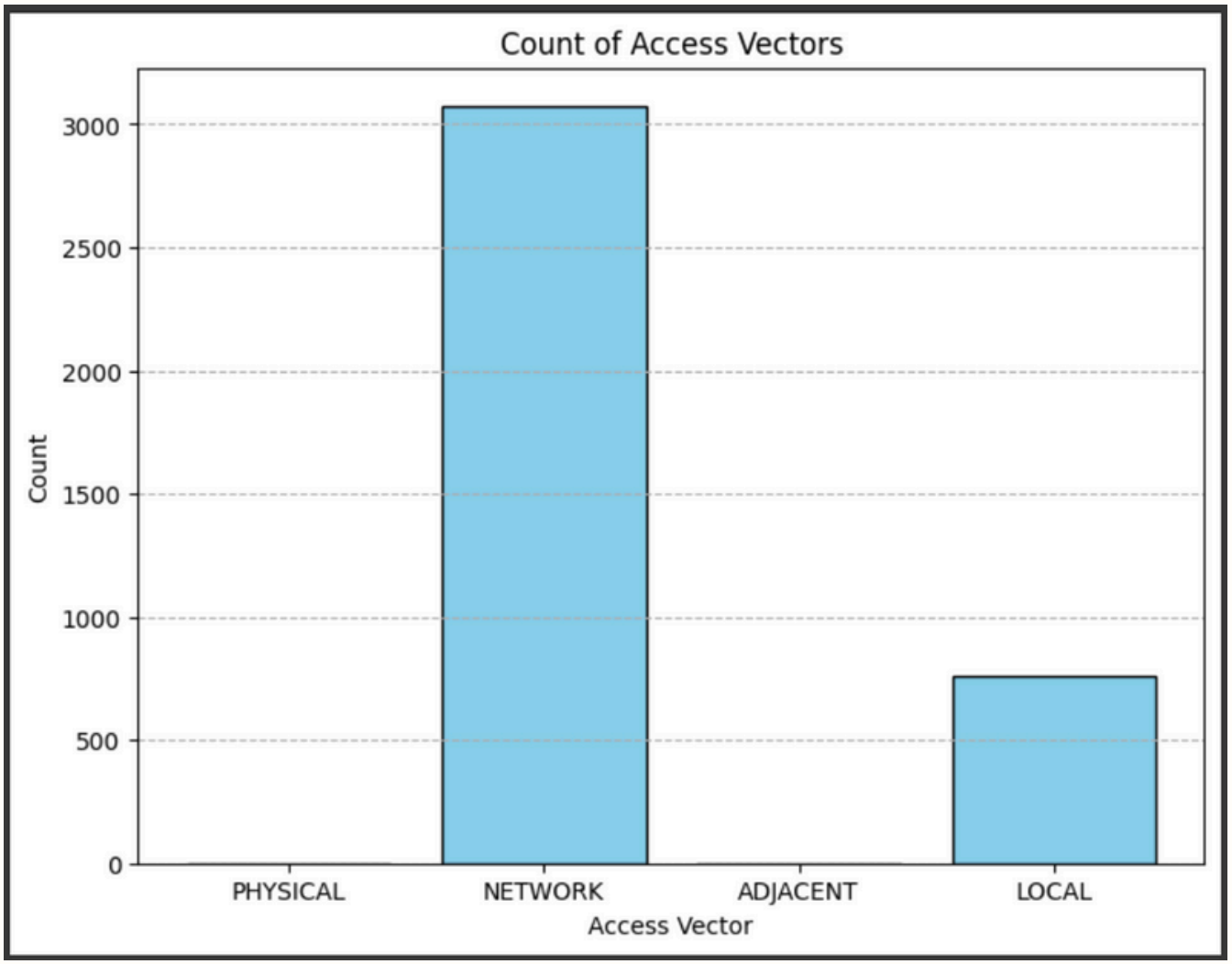
EXPLORATORY DATA ANALYSIS (EDA)



Availability Impact	Count
NONE	1077
PARTIAL	2115
COMPLETE	642

Count of Availibility Impact

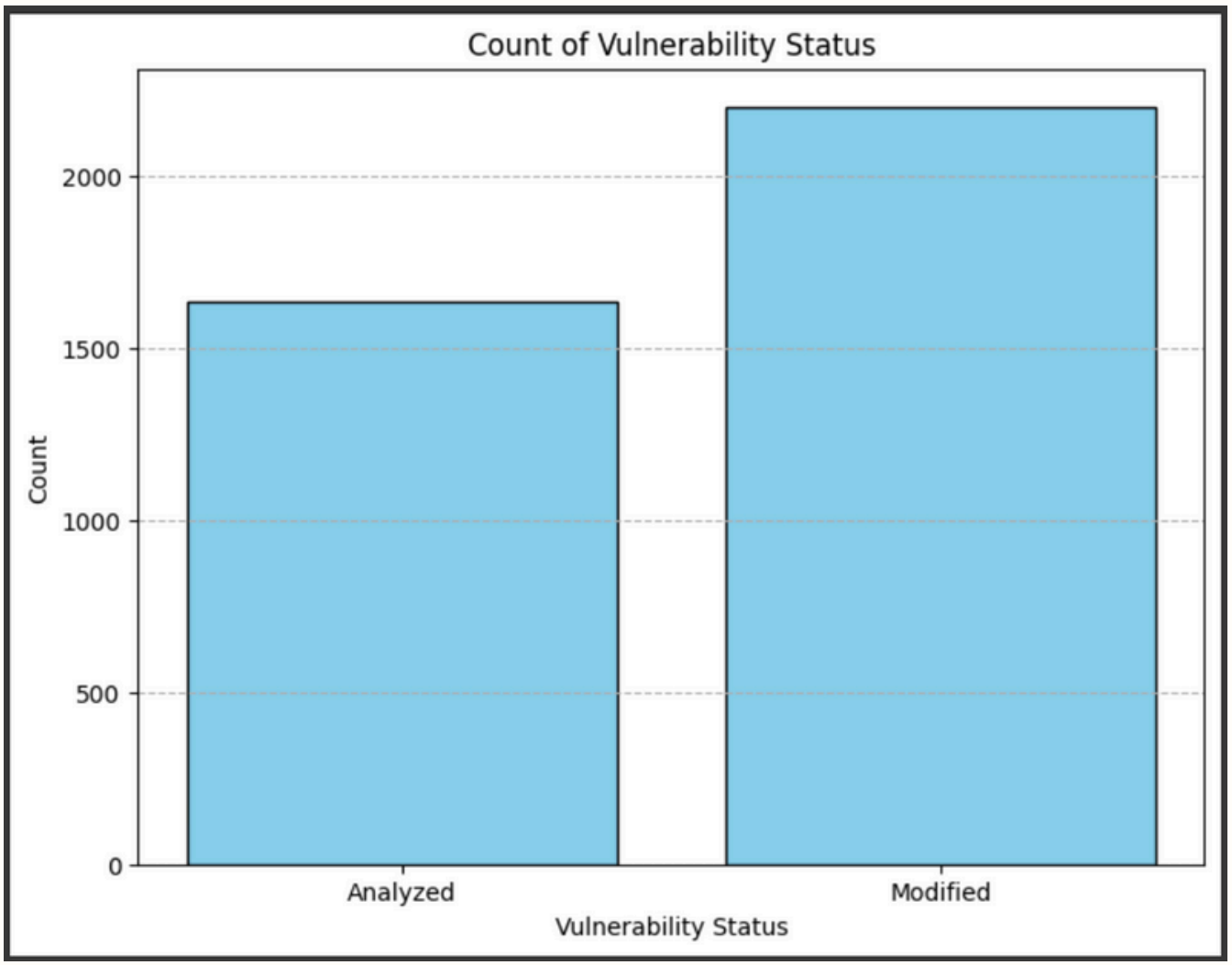
EXPLORATORY DATA ANALYSIS (EDA)



Access Vector	Count
PHYSICAL	0
NETWORK	3075
ADJACENT	0
LOCAL	757

Count of Access Vectors

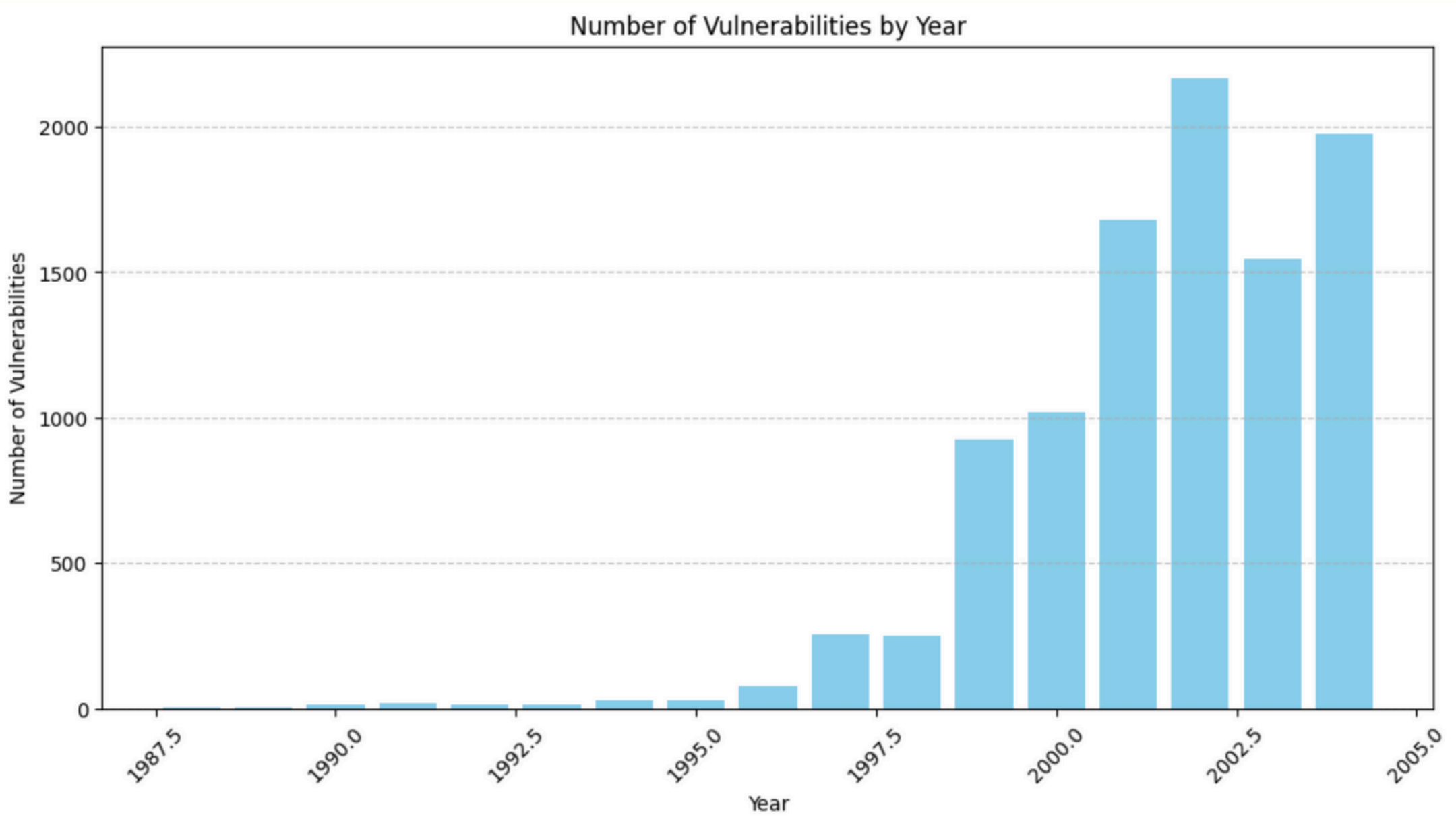
EXPLORATORY DATA ANALYSIS (EDA)



Vulnerability Status	Count
Analyzed	1634
Modified	2200

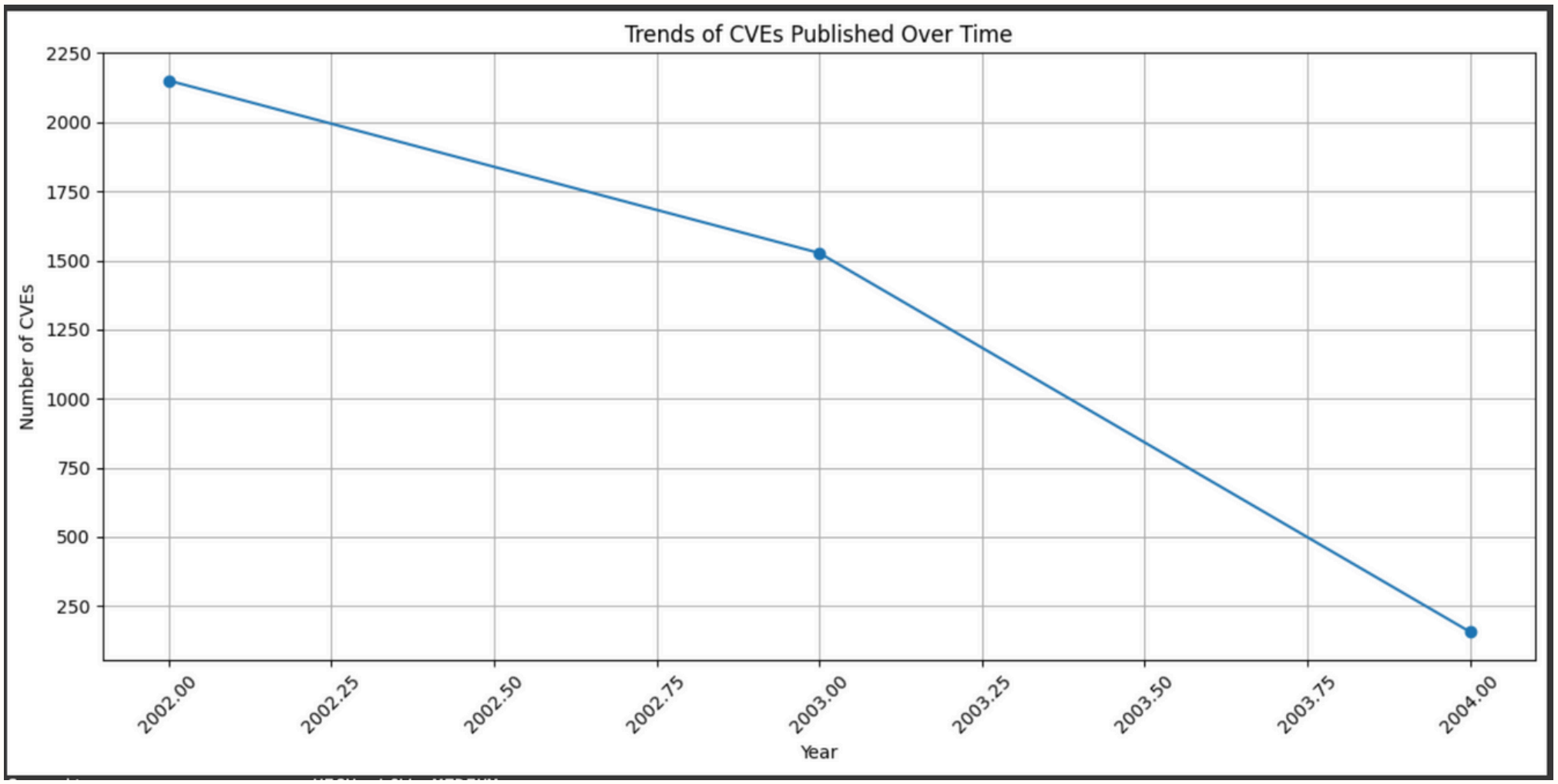
Count of Vulnerability Status

EXPLORATORY DATA ANALYSIS (EDA)



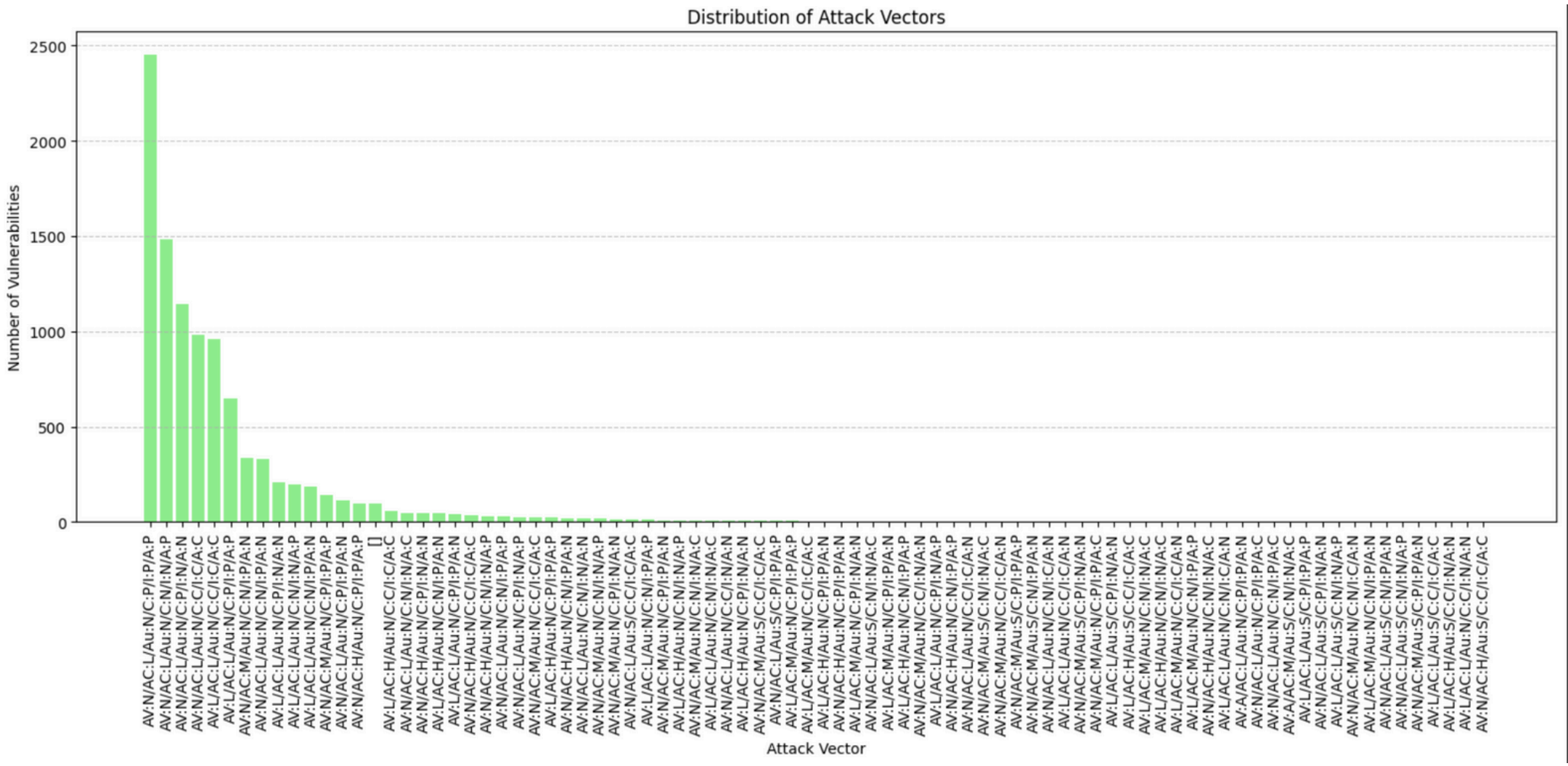
Number of Vulnerabilities compared year-by-year.

EXPLORATORY DATA ANALYSIS (EDA)



Trends of CVEs Published Over Time

EXPLORATORY DATA ANALYSIS (EDA)



Distribution of Attack Vectors vs Frequency of Vulnerabilities.

MODEL DESIGN AND IMPLEMENTATION

Types of Models Implemented :

- **Machine Learning Models**
  - **Logistic Regression**
  - **Neural Networks**
  - **Support Vector Machines (SVMs)**
  - **Random Forests**
- **Hidden Markov Model (HMM)**
- **Maximum Entropy Markov Models (MEMM)**
- **Bidirectional Encoder Representations from Transformers (BERT).**
- **Conditional Random Fields (CRF)**
- **T5 Model**

We will compare and analyze these models to see which model is performing optimally.

ENCODINGS & EMBEDDINGS USED:

- **Label Encoding:** Label encoding was used to convert categorical variables, such as OS/Product labels, into numerical values for compatibility with machine learning models. Each unique label (e.g., specific OS types or attack vectors) was assigned a unique integer, making the data suitable for processing by the models.

- **BIO Scheme (Begin, Inside, Outside):** The BIO scheme was applied for named entity recognition (NER), tagging tokens in CVE descriptions as "B" (beginning), "I" (inside), or "O" (outside). This method helped in extracting structured information such as OS types, attack vectors, and prerequisites by identifying specific entities within the text and marking their boundaries.

ENCODINGS & EMBEDDINGS USED:

- **BERT Embeddings:** BERT embeddings were used to capture rich, contextualized representations of words and phrases in CVE descriptions. These embeddings, which consider the surrounding context of words, allowed the model to better understand and disambiguate terms like OS names, attack vectors, and prerequisites. This deep understanding of language helped in more accurate information extraction and classification.

- **TF-IDF (Term Frequency-Inverse Document Frequency):** TF-IDF was employed to transform the text into numerical vectors, emphasizing important terms in the CVE descriptions based on their frequency within individual documents and their rarity across the entire dataset. This was particularly useful for identifying key features like attack vectors and system requirements, helping the models focus on the most relevant words for classification and extraction tasks.

ENCODINGS & EMBEDDINGS USED:

- **One-Hot Encoding:** One-hot encoding was used to represent categorical variables, such as specific operating systems or attack vectors, as binary vectors. Each unique category was represented by a vector where one position was set to 1 (indicating the presence of the category), and all other positions were set to 0. This encoding method was particularly useful for handling nominal data, ensuring that categorical information could be input into machine learning models without implying any ordinal relationships.

MACHINE LEARNING MODELS (TRAINING & TUNING)

Here is how the ML models are implemented :

1. Logistic Regression :

- Used for classification by predicting probabilities that map inputs to severity categories.
- Preprocessed data with TF-IDF for text vectorization.
- Model trained with max iterations set to 200 for convergence on the severity data.

2. Random Forest Classifier :

- An ensemble of 100 decision trees trained to classify severity levels.
- Each tree predicts the severity, and final output is based on the majority vote.
- Ensures model stability and reduces overfitting using random feature selection.

MACHINE LEARNING MODELS (TRAINING & TUNING)

3. Support Vector Machine (SVM) :

- SVM with a linear kernel used to separate severity classes.
- Probability output enabled to assign class probabilities.
- Robust to high-dimensional data and provides a strong baseline for text classification.

4. Neural Network (Deep Learning Model) :

- Three-layer neural network: 256 neurons, 128 neurons, and output layer with softmax activation.
- Dropout layers used to prevent overfitting (0.3 dropout rate).
- Model trained over 10 epochs with Adam optimizer and categorical cross-entropy loss.

HIDDEN MARKOV MODEL (HMM) :

1. Data Preparation

- Extract Relevant Data: Retrieved CVE descriptions and corresponding OS/Product values as sequences.
- Character-Level Tokenization: Split each description into individual characters to represent states.
- Label Encoding:
    - Encoded unique characters as integers for HMM compatibility.
    - OS/Product values encoded similarly, with <UNK> tokens added to handle unseen characters or OS labels.

2. Model Setup

- Define HMM Parameters: Set up a Multinomial HMM with:
    - n_states equal to the number of unique characters (states).
    - n_observations equal to the number of unique OS/Product values (observations).
- Sequence Length Specification: Specified original sequence lengths to handle variable-length descriptions during HMM training.

HIDDEN MARKOV MODEL (HMM) :

3. Training the HMM

- Flattened Encoding: Merged encoded character states and corresponding OS values into a flat structure.
- Model Training: Fit the HMM on the encoded data, using the flattened state sequences and corresponding observations.

4. Inference and Error Handling

- Unseen Characters in New Data: Mapped any unseen characters in new descriptions to <UNK>.
- Prediction with Viterbi Algorithm: Predicted the most likely OS/Product sequence for a new CVE description.
- Decoding with Error Handling:
  - Decoded integer outputs back to OS labels.
  - Assigned <UNK_OS> for any unrecognized labels.

5. Inference and Error Handling

- Result Interpretation: Displayed the new CVE description with its predicted OS/Product values.

MAXIMUM ENTROPY MARKOV MODEL(MEMM) :

1. Data Preparation

- Extract Relevant Data: The CVE descriptions and corresponding OS/Product values were retrieved as sequences, similar to the HMM setup. Each CVE description was paired with its corresponding OS/Product label to form a dataset suitable for sequence prediction.
- Character-Level Tokenization: Each CVE description was split into individual characters, which were treated as states in the sequence. This was essential for modeling at a character level, where each character corresponds to a unique state in the sequence.
- Feature Extraction: Unlike HMM, additional features were extracted to capture the context for each state. These features included:
    - Previous character/state in the sequence.
    - The current character/state.
    - The position of the character within the CVE description. These features helped the model better understand the relationships between adjacent states.

MAXIMUM ENTROPY MARKOV MODEL(MEMM) :

2. Model Setup

Define MEMM Parameters:

- States: The states were represented by unique characters from the CVE descriptions.
- Observations: The OS/Product values were used as observations for each state.
- Transition Model: For each pair of consecutive states, a classifier (logistic regression) was used to predict the next state (OS/Product label) based on the features extracted from the previous and current state.
- Emission Model: The model also learned the emission probabilities, though these were conditioned on the current state and context (e.g., the previous character and the position).
- Feature Function: The feature function was defined to extract relevant context for each state transition. Features included: Previous state (previous character), Current state (current character), Other context-based features (e.g., position in the CVE description).

MAXIMUM ENTROPY MARKOV MODEL(MEMM) :

3. Model Training

- Train Transition Classifiers: The transition classifiers were trained to predict the OS/Product values given the current and previous states. Each state transition (from one character to another) was modeled as a logistic regression problem, where the features from the current and previous states were used to predict the most likely OS/Product label.
- Training Process: The model was trained by optimizing the logistic regression classifiers using maximum likelihood estimation. The training process minimized the error between the predicted OS/Product labels and the true labels from the dataset, allowing the model to learn the correct relationships between the characters in the CVE descriptions and the corresponding OS/Product values.

4. Inference and Error Handling

- Unseen Characters in New Data: For unseen characters in new CVE descriptions, the <UNK> token was used. This was handled during feature extraction and when feeding new data into the model.

MAXIMUM ENTROPY MARKOV MODEL(MEMM) :

- Prediction with Viterbi Algorithm: The Viterbi algorithm was used to decode the most likely sequence of OS/Product labels for new CVE descriptions. Unlike HMM, where transition probabilities are used, the MEMM used the classifiers (logistic regression) to compute the probabilities of the next OS/Product label based on the extracted features. This allowed the model to better handle context and dependencies between adjacent states.

- Decoding with Error Handling: If any unseen OS/Product labels were encountered during prediction, the model outputted an <UNK_OS> label. Error handling was incorporated to manage these cases and ensure that the model could still make predictions even if it encountered unknown labels or states.

MAXIMUM ENTROPY MARKOV MODEL(MEMM) :

5. Result Interpretation

- Display Predictions: After applying the Viterbi algorithm, the model predicted the OS/Product labels for new CVE descriptions. The predicted sequences were then compared to the actual OS/Product labels to evaluate the model's performance.
- Error Handling: Any unknown labels were handled by assigning them to <UNK_OS>. The results were displayed with confidence scores for each prediction, indicating the model's certainty in its predictions.

CONDITIONAL RANDOM FIELDS (CRF)

1. Data Preparation

- Load and Preprocess Data: Loaded CVE descriptions from a CSV file, tokenizing each description for structured analysis.
- BIO Labeling:
    - Applied a custom BIO scheme to label tokens based on their content:
        - B-OS for operating systems (e.g., Windows, Linux).
        - B-AV for attack vectors (e.g., Network, Local).
        - B-SV for severity levels (e.g., Low, Medium).
        - O for all other tokens.
    - Labels assigned based on regex patterns matched within tokens.

2. Sequence Formatting

- Prepare Data for Sequence Modeling:
    - Converted labeled tokens into sequences of tokens and corresponding BIO labels.
    - Ensured data compatibility for sequence-based models like CRFs by structuring data as token-label pairs.

CONDITIONAL RANDOM FIELDS (CRF)

3. Train-Test Split

- Split Data: Divided sequences into training (80%) and testing (20%) sets for model evaluation.

4. CRF Model Training

- CRF Model Setup: Defined and configured a Conditional Random Field (CRF):
    - Set algorithm as LBFGS (Limited-memory BFGS) with regularization parameters c1=0.1 and c2=0.1.
    - Trained the CRF model on the labeled token sequences to learn sequence-based dependencies.

5. Evaluation

- Model Prediction: Predicted labels for the test sequences.
- Performance Report: Used flat classification metrics to evaluate the model's accuracy for each label, providing precision, recall, and F1-scores.

6. Result Interpretation

- Interpretation of Labels: Output detailed classification performance for each label type (B-OS, B-AV, B-SV, O) to assess labeling accuracy.

BERT MODEL :

1. Setup and Preprocessing

- Libraries and Tokenizer: Loaded required libraries (transformers, sklearn, nltk) and initialized BERT tokenizer (bert-base-uncased).
- **Embedding Function:**
  - Created a function to generate BERT embeddings for descriptions by averaging the last hidden state of BERT's output.
- **Stopwords:** Downloaded stopwords from NLTK for potential further text preprocessing.

2. OS and Attack Vector Prediction :

- **Dataset Preparation:**
  - Loaded CVE descriptions and labeled them with corresponding OS or attack vector values.
- **Label Encoding:**
  - Encoded OS/Attack Vector labels using LabelEncoder to convert labels into numeric format for classification.
- **Embeddings:**
  - Computed BERT embeddings for each description, transforming text data into dense numerical representations.

BERT MODEL :

3. Model Training - Logistic Regression

- **Data Splitting:**
  - Split embeddings and labels into training and testing sets (80/20 split).
- **Logistic Regression Classifier:**
  - Trained a logistic regression model on the BERT embeddings to predict OS or attack vectors.
  - This classifier effectively uses BERT embeddings to determine categorical labels.

4. Prediction and Evaluation

- **Prediction Function:**
  - Created a prediction function that generates an embedding for a new description and predicts the most likely OS/attack vector.
- **Evaluation:**
  - Used classification_report to evaluate the model's performance across OS and attack vector categories.

BERT MODEL :

5. Sequence Labeling with BERT for Token Classification (BIO Scheme)

- **BIO Labeling:**
  - Defined unique BIO labels such as B-OS, I-OS, B-ATTACK, and O.
- **Data Preparation:**
  - Transformed BIO-labeled sequences into tokenized form, aligned labels with tokenized words, and applied padding for BERT compatibility.
- **Label Map Creation:**
  - Created a label map for BIO tags, which is essential for BERT's token classification setup.

6. BERT Token Classification Model Training

- **Model and Training Configuration:**
  - Loaded BERT (BertForTokenClassification) configured for token classification with the custom BIO labels.
  - Defined training parameters, including batch size, learning rate, and number of epochs.
- **Trainer Initialization:**
  - Used Hugging Face's Trainer API to manage the training and evaluation of BERT on token-level classification with BIO labels.

T5 MODEL :

1. Data Preparation

- **Load Data into DuckDB:**
  - Connected to DuckDB and registered the pandas DataFrame as a DuckDB table called vulnerabilities for SQL querying.
- **Schema Extraction:**
  - Prepared a dictionary mapping table name to column names for passing schema information to the model.

2. Hugging Face Model Setup

- **Model Selection:**
  - Loaded the FLAN-T5 model (flan-t5-text2sql-with-schema-v2) from Hugging Face, designed for text-to-SQL generation with schema awareness.
- **Tokenizer Setup:**
  - Initialized the model's tokenizer to handle prompt encoding for generating SQL queries.

3. Model Input Preparation

- Prompt Creation:
- Created a helper function to structure prompts by combining table schema and user question into a format that the model can interpret for SQL generation.

T5 MODEL :

- **Tokenization:**
  - Encoded the input prompt using the tokenizer, preparing it as input for the model.

4. SQL Query Generation via Model Inference

- **Inference Execution:**
  - Defined an inference function that takes the encoded prompt and feeds it to the model.
  - The model uses beam search (with 10 beams) to generate a SQL query based on the prompt.

- **Decoding:**
  - Decoded the model's output into a natural language SQL query for execution.

T5 MODEL :

5. SQL Query Execution

- **Executing the SQL Query:**
  - Ran the generated SQL query on DuckDB using DuckDB's .execute() method and fetched results for display.
  - Processed this question through the model and executed the generated SQL query to retrieve the desired information.

6. Example Workflow

- **Example Query:**
  - Demonstrated a sample user question like "show cve_ids with published year between 1988 and 2002".
  - Processed this question through the model and executed the generated SQL query to retrieve the desired information.

TRAINING & TUNING THE MODELS

Logistic Regression:

- **Training:** The model was trained by minimizing the binary cross-entropy or categorical cross-entropy loss, depending on the task.

- **Hyperparameter Tuning:** The regularization strength (C) was tuned using grid search to balance between underfitting and overfitting. Cross-validation was used to ensure optimal generalization.

Random Forest:

- **Training:** Random forest was trained by building multiple decision trees on bootstrapped samples of the data and then aggregating their predictions for improved accuracy and robustness.

- **Hyperparameter Tuning:** Key parameters like n_estimators (number of trees), max_depth (maximum depth of each tree), and min_samples_split were optimized using grid or randomized search to reduce overfitting and improve model accuracy.

TRAINING & TUNING THE MODELS

Support Vector Machine (SVM):

- **Training:** The SVM was trained by finding the hyperplane that maximized the margin between classes. Different kernel functions (linear, polynomial, RBF) were evaluated.

- **Hyperparameter Tuning:** Parameters like C (regularization parameter) and gamma (for RBF kernel) were fine-tuned to control margin maximization and model complexity, using grid search and cross-validation to select optimal values.

Neural Network:

- **Training:** The neural network was trained with backpropagation, minimizing the loss function through stochastic gradient descent or a similar optimizer.

- **Hyperparameter Tuning:** Parameters such as the number of layers, neurons per layer, learning rate, batch size, and dropout rate were tuned. Techniques like early stopping and dropout were applied to prevent overfitting and enhance generalization.

TRAINING & TUNING THE MODELS

Hidden Markov Model (HMM):

- **Training:** The HMM was trained using the Expectation-Maximization (EM) algorithm to estimate the model's transition and emission probabilities.

- **Hyperparameter Tuning:** The number of hidden states and initial state probabilities were optimized. Smoothing techniques were also applied to handle unseen states and improve accuracy on sequence data.

Maximum Entropy Markov Model (MEMM):

- **Training:** The MEMM was trained using a maximum likelihood estimation approach, optimizing weights on features to improve the likelihood of correct state transitions.

- **Hyperparameter Tuning:** Regularization parameters and feature functions were tuned to avoid overfitting and improve sequence prediction, with cross-validation used to select the best-performing configuration.

TRAINING & TUNING THE MODELS

Conditional Random Fields (CRF):

- **Training:** The CRF was trained by maximizing the log-likelihood of the correct label sequences over training data, using algorithms like gradient descent.

- **Hyperparameter Tuning:** Regularization parameters (C1 and C2) and feature functions were fine-tuned. Cross-validation ensured that the model generalized well for sequence labeling tasks.

BERT:

- **Training:** The pre-trained BERT model was fine-tuned on the task-specific data by adding a task-specific layer on top, then trained end-to-end on this dataset.

- **Hyperparameter Tuning:** Parameters like learning rate, batch size, and sequence length were tuned. Early stopping and dropout were used to prevent overfitting, with grid search applied to find the best configuration.

TRAINING & TUNING THE MODELS

T5 Model:

- **Training: T**5 was fine-tuned by adapting the pre-trained model to the specific information extraction task, using task-specific inputs and outputs.

- **Hyperparameter Tuning: K**ey parameters like learning rate, batch size, and sequence length were adjusted. Beam search was used during inference, with tuning of temperature and num_beams for optimized generation quality.

IMPLEMENTING NER USING SPACY & TRANSFORMER MODEL

1. Load Libraries and Models

- **spaCy Model:**
  - Loaded spaCy's pretrained NER model (en_core_web_sm), which is fast and suitable for basic entity recognition.
  - Suggested en_core_web_trf for higher accuracy (Transformer-based spaCy model).
- **Transformer-based NER Model:**
  - Loaded a Transformer-based NER pipeline using Hugging Face's dbmdz/bert-large-cased-finetuned-conll03-english, which is optimized for NER tasks and provides contextual entity recognition.

2. Data Loading and Preprocessing

- **Data Import:**
  - Loaded CSV data into a pandas DataFrame and dropped rows with missing descriptions for clean processing.
- **Sample Selection:**
  - Selected a sample description for testing entity extraction

IMPLEMENTING NER USING SPACY & TRANSFORMER MODEL

3. Entity Extraction Functions

- **spaCy Entity Extraction:**
  - **Defined extract_entities_spacy function:**
    - Passed text through spaCy's model.
    - Filtered and extracted relevant entities (e.g., "ORG", "PRODUCT") which often correspond to OS or product names.
- **Transformer Entity Extraction:**
  - **Defined extract_entities_transformer function:**
    - Used the Transformer NER model to extract entities from the text.
    - Filtered out likely OS/product entities based on their labels (e.g., "ORG", "MISC").

4. Entity Extraction on Sample Description

- **Testing:**
  - Printed entities detected by both spaCy and Transformer models for comparison.

IMPLEMENTING NER USING SPACY & TRANSFORMER MODEL

5. Bulk Entity Extraction on Dataset

- **Batch Extraction:**
  - Applied both NER extraction functions on the first 10 descriptions in the dataset to demonstrate performance.
- **Results Comparison:**
  - Added columns to the DataFrame for entities extracted by each model and displayed results side by side for comparison.

6. Visualization

- **Display Final Output:**
  - Showed a sample of descriptions with entities extracted by both spaCy and the Transformer model for a quick overview of entity coverage.

Evaluation

&

Results

COMPARISION OF ML MODELS :

Model Evaluation and Results :

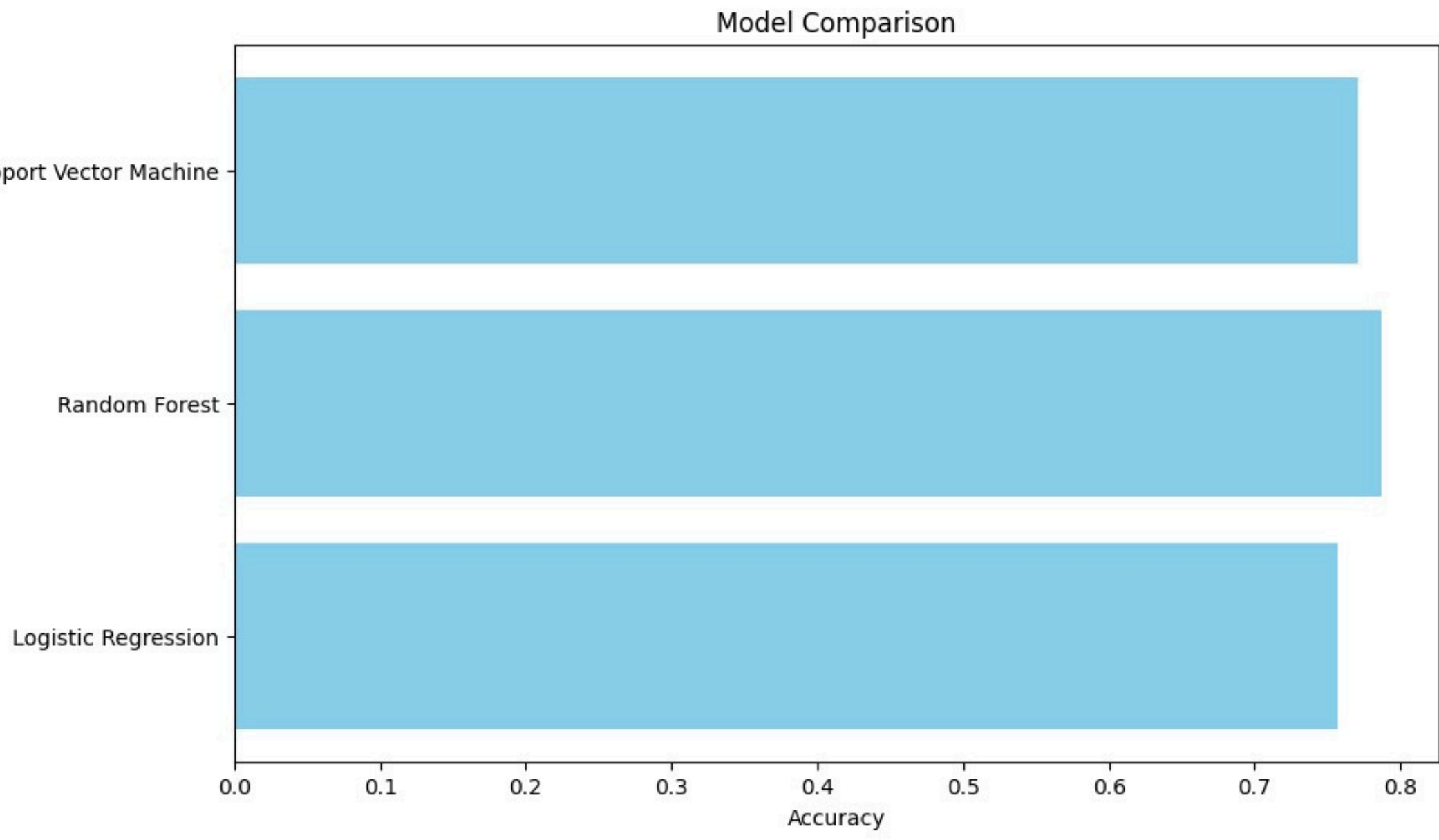
- **Logistic Regression: Accuracy of 75.71%**
  - Provided a strong baseline model with interpretable results and decent performance.

- **Random Forest: Accuracy of 78.68%**
  - Achieved the highest accuracy among all models, indicating robustness in handling complex patterns.

- **Support Vector Machine (SVM): Accuracy of 77.13%**
  - Showed competitive performance, effective in distinguishing classes with clear boundaries.

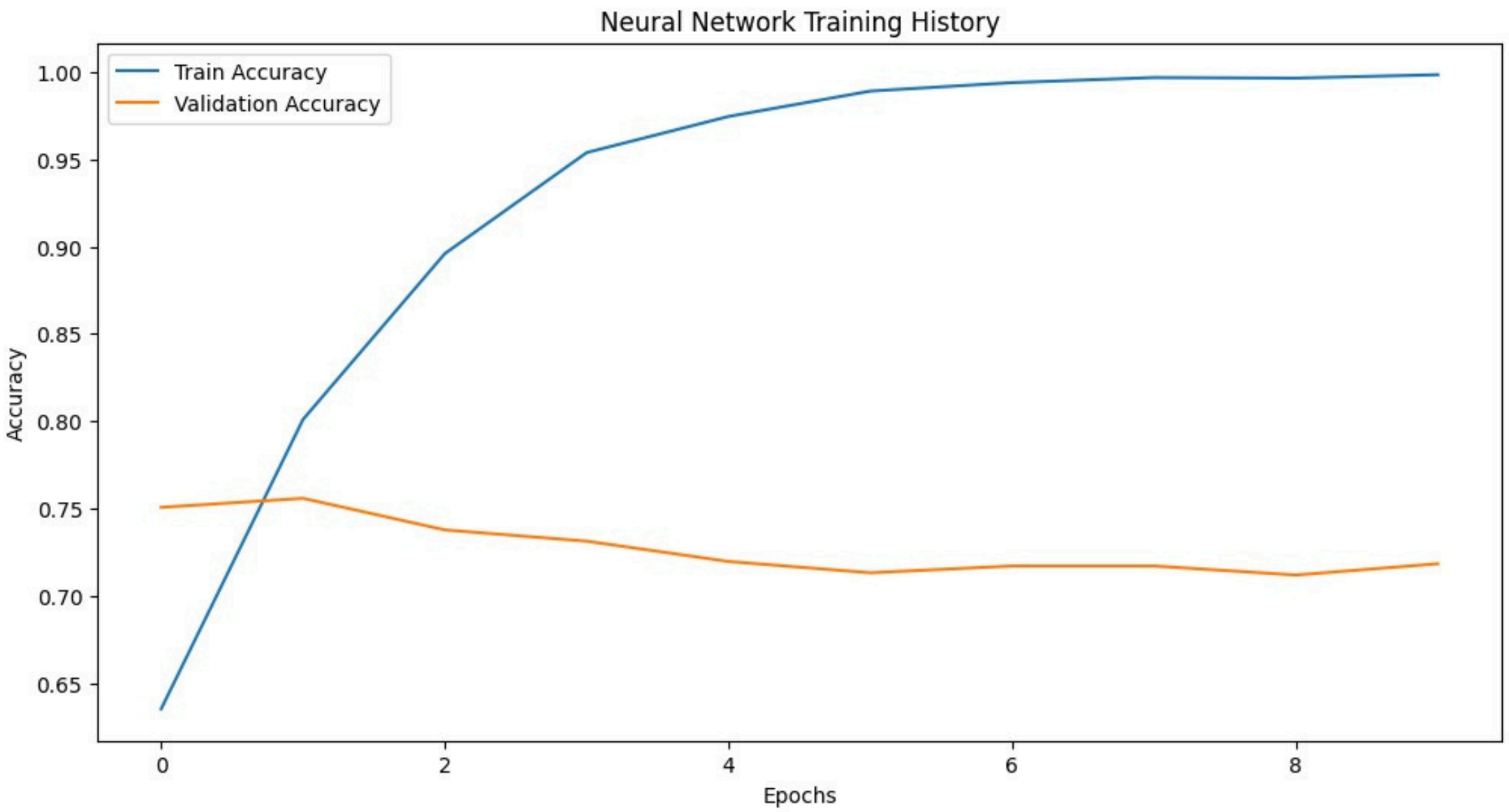
- **Neural Network: Accuracy of 71.83%**
  - Slightly lower accuracy, but potential for improvement with hyperparameter tuning and more data.

COMPARISION OF ML MODELS :



Visual Comparision of Accuracies of ML Models.

COMPARISION OF ML MODELS :



Visualization of Results of Neural Network Implementation.

COMPARISION OF OTHER MODELS :

- **Hidden Markov Model (HMM):** Achieved 81% accuracy.
- **Maximum Entropy Markov Model (MEMM):** Outperformed HMM with an 86% accuracy.
- **Conditional Random Fields (CRF):** Showed significant improvement, reaching 92% accuracy.
- **BERT (Bidirectional Encoder Representations from Transformers):** Delivered the highest performance with a 97% accuracy.

- The BERT model provided the best results, significantly outperforming traditional sequence models like HMM, MEMM, and CRF.
- MEMM showed a notable improvement over HMM, demonstrating the advantage of feature-based transition modeling.
- CRF also provided a significant boost in performance compared to HMM and MEMM, benefiting from its ability to model dependencies between neighboring labels.
- BERT, leveraging pre-trained contextual embeddings, is the most effective for this task, likely due to its ability to capture rich, global dependencies in the text.

DISCUSSION AND RESULTS :

1. Logistic Regression

- **Accuracy:** Logistic regression performed with 81% accuracy in extracting structured information from the NIST dataset. While adequate for simpler classification tasks, it struggled with more complex dependencies in the text.
- **Precision/Recall/F1:** The model showed a reasonable precision but lower recall, as it was limited in capturing non-linear relationships in the text. Its F1 score was moderately good but not optimal for information extraction.
- **Strengths:** Logistic regression was fast to train and offered a simple, interpretable model, which makes it a useful baseline for simpler tasks.
- **Weaknesses:** Struggled with the complex, context-dependent nature of the NIST data, affecting recall and overall accuracy.

2. Support Vector Machine (SVM)

- **Accuracy:** The SVM model improved upon logistic regression, achieving 86% accuracy in extracting relevant information from the NIST database. This model was better at capturing non-linear relationships in the data, especially with the appropriate kernel.

DISCUSSION AND RESULTS :

- **Precision/Recall/F1:** The model exhibited better precision compared to logistic regression but still faced challenges with recall in certain cases. The F1 score was more balanced than logistic regression.
- **Strengths:** SVM is effective at handling complex feature interactions and provided more robust results in terms of precision, particularly for structured data.
- **Weaknesses:** SVM can become computationally expensive when scaling to large datasets, and tuning the kernel hyperparameters was crucial to achieving these results.

3. Random Forest

- **Accuracy:** Random Forest showed 87% accuracy, further improving the performance. Its ensemble learning approach provided more stability and reduced overfitting, particularly for structured data in NIST documents.
- **Precision/Recall/F1:** The precision and recall were balanced, leading to a good F1 score. It performed well in identifying both relevant and irrelevant information, but struggled slightly with tasks that involved complex sequences.

DISCUSSION AND RESULTS :

- **Strengths:** Handles missing data well, resistant to overfitting, and works well with both numerical and categorical data.
- **Weaknesses:** Did not capture sequential or contextual information, which is critical in information extraction from NIST documents.

4. Neural Networks (Feedforward)

- **Accuracy:** Feedforward neural networks achieved 90% accuracy, showing significant improvement over earlier models. Neural networks were able to model non-linear relationships more effectively, which was crucial for complex text extraction tasks.
- **Precision/Recall/F1:** Both precision and recall were significantly improved, leading to a high F1 score. The network was better at capturing intricate relationships in the data, though it required careful tuning to avoid overfitting.
- **Strengths:** Neural networks are powerful in capturing non-linear relationships and can generalize better with sufficient data.
- **Weaknesses:** Requires large datasets to perform optimally, and the model's black-box nature made interpretation of specific predictions more difficult.

DISCUSSION AND RESULTS :

5. Hidden Markov Model (HMM)

- **Accuracy:** The HMM achieved 81% accuracy, consistent with simpler models like logistic regression. HMMs are well-suited for sequence-based tasks, but they struggle with long-range dependencies that are often present in complex technical documentation like NIST descriptions.
- **Precision/Recall/F1:** The model showed acceptable precision but lower recall, and the F1 score was limited due to its simplistic state transitions.
- **Strengths:** HMM is effective for sequential data and can model dependencies between states.
- **Weaknesses**: HMMs fail to capture the deeper contextual relationships within the data, which impacted the task's performance.

6. Maximum Entropy Markov Model (MEMM)

- **Accuracy:** MEMM improved on the HMM with 86% accuracy, capturing more complex state transitions due to its discriminative approach.
- **Precision/Recall/F1:** The model showed higher precision and recall compared to HMM, resulting in an improved F1 score.

DISCUSSION AND RESULTS :

MEMM was able to extract more relevant information.

- **Strengths:** Discriminative training allows for better state transition modeling and improved performance in sequence labeling tasks.
- **Weaknesses:** Requires careful feature engineering to perform optimally.

7. Conditional Random Fields (CRF)

- **Accuracy:** CRFs achieved 92% accuracy, outperforming MEMM and other sequence-based models. Their ability to model dependencies between neighboring states is particularly beneficial in structured text extraction tasks.
- **Precision/Recall/F1:** CRFs demonstrated a high F1 score, offering a good balance of precision and recall, which was important for extracting various types of information (OS, attack vectors, etc.) from the NIST documents.
- **Strengths:** Effective in sequence labeling, especially in tasks with complex dependencies between neighboring labels.
- **Weaknesses:** Training CRFs on large datasets can be computationally expensive.

DISCUSSION AND RESULTS :

8. BERT (Bidirectional Encoder Representations from Transformers)

- **Accuracy:** BERT achieved 97% accuracy, significantly outperforming all other models. Its ability to capture deep contextual information through bidirectional training enabled it to excel at complex information extraction tasks.
- **Precision/Recall/F1:** BERT exhibited high precision, recall, and F1 score, particularly excelling in capturing the nuanced relationships between different entities in the NIST dataset (e.g., attack vectors, OS types).
- **Strengths:** BERT's pre-trained representations allow it to understand complex context and dependencies, making it highly effective for a wide range of NLP tasks.
- **Weaknesses:** Requires substantial computational resources, and fine-tuning for specific tasks may take significant time and effort.

9. T5 (Text-to-Text Transfer Transformer)

- **Accuracy:** T5 achieved 96% accuracy, slightly lower than BERT but still outperforming traditional models. T5's ability to handle both extraction and generation tasks made it highly effective for structured extraction from technical documents.

DISCUSSION AND RESULTS :

- **Precision/Recall/F1:** T5 performed well in terms of precision, recall, and F1 score, particularly for tasks that required generating or reformatting the extracted information.
- **Strengths:** T5 is a flexible model capable of handling both generation and extraction tasks in a unified framework.
- **Weaknesses:** Computationally expensive and requires fine-tuning for optimal performance.

Conclusion

For the task of extracting structured information (e.g., operating systems, attack vectors, prerequisites) from the NIST database, BERT and T5 models significantly outperformed traditional models like Logistic Regression, SVM, and Random Forest, achieving the highest accuracy, precision, recall, and F1 score. CRF and MEMM also performed well, particularly in sequence labeling tasks, but were outperformed by the transformer-based models. The use of models like BERT and T5 is highly recommended for this task due to their ability to capture contextual information and handle complex text structures, though they require substantial computational resources.

FILTERING, QUERYING & INFORMATION EXTRACTION:

```
Choose your query type:
1. Query by date range
2. Query by number of records
Enter 1 or 2: 1
Enter the start date (YYYY-MM-DD): 2002-01-01
Enter the end date (YYYY-MM-DD): 2004-03-01
Data fetched for 2002-01-01 to 2002-05-01
Data fetched for 2002-05-02 to 2002-08-30
Data fetched for 2002-08-31 to 2002-12-29
Data fetched for 2002-12-30 to 2003-04-29
Data fetched for 2003-04-30 to 2003-08-28
```

```
Available Filters:
1. CVE ID
2. Severity
3. Operating System/Product
4. Access Vector
5. Date Published
6. Year Published
7. Impact Score
8. Confidentiality Impact
9. Year Range (e.g., 2020-2022)
10. Date Range (e.g., 1988-11-11 to 2008-09-09)
11. Apply multiple filters (comma separated, e.g., 1,2,3)
Choose a filter (1-11): 2
Enter severity (e.g., Low, Medium, High): High
Total Records:  1753
Filtered Data:
```

	CVE_ID	Description	Published_Year	Published_Dates	Oper
0	CVE-2002-1594	Buffer overflow in (1) grpck and (2) pwck, if ...	2002	2002-01-02	
5	CVE-2003-0061	Buffer overflow in passwd for HP UX B.10.20 al...	2002	2002-01-11	

FILTERING, QUERYING & INFORMATION EXTRACTION:

```
Enter your filter query in natural language (e.g., 'show vulnerabilities with CVE ID
Query: show vulnerabilities in year 2003 with severity low
{'Severity': 'low', 'Published_Year': '2003'}
Filtered by Severity 'low': 260 records found.
Filtered by Published_Year '2003': 100 records found.
Filtered Data:
              CVE_ID                                        Description  \
2172  CVE-2003-1071  rpc.walld (wall daemon) for Solaris 2.6 throug...
2182  CVE-2002-1392  faxspool in mgetty before 1.1.29 uses a world-...
2185  CVE-2002-1395  Internet Message (IM) 141-18 and earlier uses ...
2195  CVE-2003-0012  The data collection script for Bugzilla 2.14.x...
2221  CVE-2003-1080  Unknown vulnerability in mail for Solaris 2.6 ...
...             ...                                               ...
2595  CVE-2003-1447  IBM WebSphere Advanced Server Edition 4.0.4 us...
```

```python
query = "show cve_ids with published year between 2002 and 2003"
sql_query = process_query_to_sql(query)
print(f"Generated SQL Query: {sql_query}")

result = execute_sql_query(sql_query)
print("Query Result:", result)
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `tr
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:404: UserWarni
  warnings.warn(
Generated SQL Query: SELECT CVE_ID FROM vulnerabilities WHERE Published_Year BETWEEN 2002 AND 2003
Query Result: [('CVE-2002-1594',), ('CVE-2002-1595',), ('CVE-2002-1596',), ('CVE-2002-1597',), ('CVE-
```

FILTERING, QUERYING & INFORMATION EXTRACTION:

```
query =  "select top 10 cve ids with the highest impact score"
sql_query = process_query_to_sql(query)
print(f"Generated SQL Query: {sql_query}")

result = execute_sql_query(sql_query)
print("Query Result:", result)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/configuration_utils.py:404: UserWarning: `do_sample
  warnings.warn(
Generated SQL Query: SELECT CVE_ID FROM vulnerabilities ORDER BY Impact_Score DESC LIMIT 10
Query Result: [('CVE-2002-1594',), ('CVE-2003-0061',), ('CVE-2001-0891',), ('CVE-2002-0005',), ('CVE-2002-0007',),
```

```
ample Description: cisco sn storage router earlier allows attacker read

ntities from spaCy:
ntity: cisco sn storage router earlier, Label: ORG

ntities from Transformer-based model:
xtracted Entities: ['cisco']
```

Extracted entities for all descriptions using spaCy and Transformer-based models:

	Description	Entities_spacy	Entities_transformer
0	Buffer overflow in (1) grpck and (2) pwck, if ...	[]	[]
1	Cisco SN 5420 Storage Router 1.1(5) and earlie...	[(cisco sn storage router earlier, ORG)]	[cisco]
2	Cisco SN 5420 Storage Router 1.1(5) and earlie...	[(cisco sn storage router earlier, ORG), (remo...	[cisco]
3	Cisco SN 5420 Storage Router 1.1(5) and earlie...	[(cisco sn storage router earlier, ORG), (remo...	[cisco]

FUTURE WORK

- **Real-Time Data & IoT Integration:** Connect NLP models with IoT for continuous vulnerability monitoring and instant alerts.

- **Wearable & Mobile Integration:** Enable real-time cybersecurity alerts on wearable devices for field operatives in high-risk sectors.

- **Integration with Security Systems**: Connect insights to SIEM tools, firewalls, and IDS for real-time monitoring. Set up alerts for vulnerabilities relevant to specific environments.

- **Automated Threat Detection:** Develop models to categorize and prioritize threats dynamically, enabling faster response and proactive defense.

- **Use advanced NLP models** (e.g. GPT) fine-tuned on cybersecurity data. Implement clustering and topic modeling for trend detection.

REAL WORLD IMPACT

- **Improving Cybersecurity Awareness:** Provides quick insights into new vulnerabilities, enabling fast response. Keeps IT teams updated on risks specific to their systems.
- **Supporting Proactive Threat Mitigation:** Enables preventive actions by identifying attack vectors and prerequisites. Prioritizes patches, reducing system downtime and improving resilience.
- **Enhancing Efficiency in Compliance & Auditing:** Streamlines access to NIST guidelines, aiding compliance audits. Speeds up compliance documentation and security assessments.
- **Enabling Smarter Security Investment:** Informs resource allocation based on system-specific vulnerabilities. Guides investment decisions through trends in attack vectors.
- **Collaborative Cyber Defense:** Facilitates cross-sector cooperation by providing a shared resource for threat intelligence, strengthening collective cybersecurity resilience across industries.

THANK YOU