ANOMALY DETECTION IN NETWORK TRAFFIC USING MACHINE LEARNING AND DJANGO

ABSTRACT

During the times of rising cyber attacks, data compromises, and complex patterns of attacks, it has been an utmost priority for organizations and individuals to ensure the security of network systems. Rule-based intrusion detection systems tend to be less adaptable to catch unknown or changing threats, which may cause vulnerabilities in the network infrastructure. This project fills this void by designing a Machine Learning-based anomaly detection system based on Autoencoder and Isolation Forest models. These unsupervised models can determine anomalies from normal traffic flow and are efficient at detecting multiple kinds of intrusions and anomalies without knowledge of the attack signatures. The algorithm is implemented as part of a Django-based web application that provides an easy-to-use interface for users to plug in network parameters and receive instantaneous classification results—either normal or anomalous. The application also includes visualization capabilities and model performance metrics to facilitate better understanding and decision-making. By integrating strong ML algorithms with an end-user-friendly web interface, this project offers an extensible, real-time, and intelligent solution to enhance cybersecurity and proactively detect threats

INTRODUCTION

This project is aimed at anomaly detection in network traffic through Machine Learning methods, namely Autoencoder and Isolation Forest models, to detect abnormal patterns that could potentially suggest cyber threats. Conventional rule-based systems are not able to change with the developing attack strategies and thus become a prerequisite for network security through intelligent anomaly detection. To make the experience interactive and user-centric, the models are implemented into a Django web application where users can provide traffic data and get feedback in real-time on if the activity is normal or anomalous.

This marriage of web integration and machine learning provides an efficient and scalable solution for the improvement of network intrusion detection systems.

Tools and Technologies used

1. **Programming Language:** Python
2. **Machine Learning Libraries:**

• **scikit-learn** – for Isolation Forest and preprocessing

• **TensorFlow / Keras** – for model development and training of Autoencoder models

3. **Data Handling & Visualization:**

• **Pandas, NumPy** – for data manipulation

• **Matplotlib, Seaborn** – for visual analysis and insight into data distributions

4. **Web Framework:**

• Django – for backend development and ML model integration

• HTML, CSS, Bootstrap – for frontend interface

5. **Model Serialization:** Joblib – used to save and load trained ML models

DATASET

Public available network intrusion datasets like https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data used for model training and evaluation.

SYSTEM WORKFLOW

1. **Data Preprocessing**: Raw network traffic data is preprocessed, encoded, and scaled with standard preprocessing methods

2. **Model Training:**

o Autoencoder is trained to acquire normal behavior of the network.

o Isolation Forest is trained to separate anomalies through random partitioning of the feature space.

3. Anomaly Detection: During inference, both models classify new inputs as either normal or anomalous.

4. Web Integration: The trained models are web integrated in a Django web application where users can input feature values and get classification results in real time.

5. Output and Visualization: The output is displayed to the user as Normal or Anomaly, with further data visualization and model insights provided.

SIGNIFICANCE

This project is a scalable, user-friendly, and smart anomaly detection system that can be used in real-world settings. Integrating Machine Learning into a modern web application closes the gap between research and practical deployment, furthering proactive network security and cyber defense.

Advantages

 Unsupervised Learning: No labelled attack data required, allowing the system to adapt to new threats.

 Hybrid Approach: A combination of Autoencoder and Isolation Forest provides better detection accuracy and stability.

 Web Interface Django offers a simple and usable web interface for real-time anomaly detection.

 Scalable & Extendable: Adding new datasets, models, or features can be easily done to the system

 Model Explainability: Visualizations and metrics allow users to interpret model behaviour and trust output.

Disadvantages

 Model Complexity: Training and fine-tuning Autoencoders consume GPU resources and time.

 False Positives: Unsupervised models can sometimes identify normal variations as anomalies.

 Static Thresholds: Thresholds used for detection must be carefully calibrated to ensure continued performance.

PROBLEM STATEMENT

Conventional rule-based intrusion detection systems find it difficult to identify unknown or changing cyber threats in network traffic. Manual inspection is not effective because of the enormous amount and complexity of data. The project focuses on creating an intelligent, ML-driven anomaly detection system with Autoencoder and Isolation Forest, with Django integration for real-time processing.

Objectives

 To create a system that identifies anomalies in network traffic accurately through Machine Learning methodologies.

 To develop and compare Autoencoder and Isolation Forest models for efficient anomaly detection.

 To implement the ML models in an easy-to-use Django web application for real-time input and processing.

 To give visual feedback and classification (normal or anomaly) depending on network data input.

 To help enhance cybersecurity using smart, scalable, and automated detection techniques.

TECHNOLOGIES USED

Programming Language

 Python – for data preprocessing, model development, and backend integration.

Machine Learning & Deep Learning Libraries

 scikit-learn – for applying the Isolation Forest model and evaluation metrics.

 TensorFlow / Keras – for Constructing and training the Autoencoder model.

Data Preprocessing & Analysis

 Pandas – for data manipulation and cleaning.

 NumPy – for numerical operations and array processing.

 LabelEncoder, StandardScaler – for encoding categorical data and scaling features..

Visualization Tools

 Matplotlib – for plotting graphs such as ROC, confusion matrix, etc

 Seaborn – for advanced data visualization and correlation heatmaps.

Web Development

 Django – for constructing the web application and deploying ML models.

 HTML, CSS, Bootstrap – for building a user-friendly and responsive interface..

Dataset

 https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data standard dataset used for training and testing network anomaly detection models.

DATASET DESCRIPTION

The dataset employed in this project is the KDD Cup 1999 dataset, which is among the best-known benchmark datasets used to assess network intrusion detection systems. It was initially developed for the Third International Knowledge Discovery and Data Mining Tools Competition and based on simulated network traffic data gathered in a military setting. Each entry in the dataset is a single network connection and is tagged as either normal or attack, so it can be used for multi-class as well as binary classification. There are 41 input features and 1 target label in the dataset, totaling 42 columns. These characteristics are categorized into three groups: basic features (e.g., duration, protocol type, and service), content features (e.g., amount of failed login attempts or root accesses requests), and traffic features (e.g., the number of connections to the same host or service within a specific time window). The data contains both numerical and categorical values, which need to undergo preprocessing such as label encoding and normalization to be incorporated in machine learning algorithms.

The target feature comprises a large number of attack types, which are consolidated into four broad categories: Denial of Service (DoS), Remote to Local (R2L), User to Root (U2R), and Probe. Some of the examples of these attacks are 'smurf', 'neptune', 'buffer_overflow', 'guess_passwd', and 'nmap'. For the sake of this project, the classification problem has been reduced to a binary classification, where all attacks are categorized as "anomaly" and normal connections as "normal". Given the size of the complete dataset (around 4.9 million records), the usual subset kddcup.data_10_percent (around 494,021 records) is employed to train and evaluate the models. This subset is adequate for model learning and offers a balanced set of many different attack types. Overall, the KDD Cup 1999 dataset is a robust and well-

organized base for anomaly detection system development and testing within the field of network security.

DATA PREPROCESSING

1. **Removal of Duplicates**

Duplicate records were detected and deleted to remove redundancy and minimize noise in the dataset, providing cleaner input for model training.

2. **Label Encoding of Categorical Features**

Categorical features like protocol_type, service, and flag were changed into numerical format by using Label Encoding so they can be compatible with machine learning algorithms.

3. **Simplification of Target Label**

The multi-class target labels were transformed into binary classes. Normal and attack connections were labeled as "normal" and "anomaly" respectively, reducing the problem to binary classification for anomaly detection.

4. **Feature Scaling**

Standard Scaler was used for all numerical features to standardize the data. This scaling ensures features with zero mean and unit variance, leading to enhanced performance and convergence of models, especially the Autoencoder

5. **Handling Class Imbalance**

The class distribution was examined to determine possible imbalance between normal and anomalous samples. This process ensures that the model is not biased in training and testing..

6. **Train-Test Split**

The dataset was divided into training and test subsets. The training subset was mainly composed of normal records to enable the models particularly the unsupervised Autoencoder to learn patterns of authentic behavior.

The test subset contained both included both normal and anomaly records for performance testing.

MODEL SELECTION

1. **Objective-Oriented Selection**

The main goal of this project is to detect unusual patterns in network traffic without any prior knowledge of known attack signatures. Hence, unsupervised learning models were utilized to efficiently detect novel or unknown anomalies.

2. **Autoencoder (Deep Learning-Based Approach)**

o It is an unsupervised anomaly detection type of neural network.

o It learns how to compress (encode) and reconstruct (decode) regular input data with little loss.

o When anomalous data is input into the model, the reconstruction error is much greater, making the system able to identify it as an anomaly..

3. Ideal for modeling complex, non-linear interactions in high-dimensional data.

4. **Isolation Forest (Tree-Based Approach)**

o Isolation Forest is a fast, unsupervised model which separates anomalies by selecting features and splitting points randomly..

o Anomalies are generally isolated quickly because of their sparse, unique nature.

o Efficient computationally and is effective with large data and low-dimensional or structure

o Less feature scaling sensitive than neural networks.

5. **Hybrid Modeling Strategy**

o The combination of a deep learning model (Autoencoder) and a conventional ensemble approach (Isolation Forest) is able to capture the benefits of both methods.

o The hybrid model enhances robustness and accuracy by both linear and non-linear anomalies handling effectively.

o The collection also supports cross-validation and comparison of outcomes, producing more accurate conclusions.

6. **Model Choice Justification**

o Both models are unsupervised, best suited for scenarios where labelled attack information is scarce or nonexistent..

o They are complementary to each other: Autoencoder extracts deeper feature patterns, whereas Isolation Forest is fast and interpretable anomaly detection.

o They are also apt for incorporation in a Django-based web interface for real-time detection.

ISOLATION FOREST IMPLEMENTATION

The Isolation Forest model was chosen as one of the base models for anomaly detection because it is efficient and effective in detecting outliers in high-dimensional data such as network traffic. It randomly partitions the data and isolates anomalies by shorter path lengths in decision trees

Library and Import

Implemented using the Isolation Forest class from the sklearn.ensemble module in the scikit-learn library.

 Preprocessing

o Categorical features (protocol_type, service, flag) were encoded using Label Encoder.

o All numerical features were scaled using Standard Scaler to normalize the input.

 Label Conversion

Transformed multi-class attack labels into binary labels:

o "normal" for normal traffic

o "anomaly" for all types of attacks

 Model Initialization

o n_estimators = 100 (number of base estimators/trees)

o contamination = 0.1 (ratio of outliers in the data)

o random_state = 42 for reproducible results

 Training Phase

The model was trained on a dataset having mostly normal records to acquire the pattern of legitimate traffic.

 Prediction & Anomaly Scoring

o The model calculated anomaly scores for every instance..

 Instances with scores less than the threshold learned during training were marked as anomalies.

 Performance Evaluation

Model effectiveness was evaluated using:

o Accuracy

o Precision

o Recall

o F1-Score

o Confusion Matrix

 Integration into Web Application

o The trained model was saved using joblib.

o Integrated within the Django backend to enable real-time detection of anomalies based on user input.

 Advantages

o Fast training and inference

o Effective for high-dimensional data

o Unsupervised, so no labeled attack data is required for training

AUTOENCODER IMPLEMENTATION

An Autoencoder is a type of unsupervised deep learning model employed to learn compressed representations of input data based on minimizing the reconstruction error.

In anomaly detection, Autoencoders are trained solely on normal network traffic such that they learn its underlying structure. When an anomalous (unseen or malicious) data is shown, the model does not reconstruct it well, and an output with a very high reconstruction error is generated, which in turn serves as a foundation to label it as an anomaly. This renders Autoencoders very effective for identifying intricate and subtle intrusions in network traffic.

Steps:

 Library and Framework

Implemented with TensorFlow and Keras, which provide high-level APIs to develop and train neural networks effectively

 Input Data Preparation

o The data was preprocessed through label encoding of categorical features and normalization of all numeric features with StandardScaler.

o Training was performed only on normal class data in order to identify patterns of legal network activity.

 Model Architecture

o Input Layer: 41 neurons (equal to number of features)

o Encoder: Dense layers to reduce dimensionality

o Latent Space: Bottleneck layer for the compressed data

o Decoder: Symmetrical layers for decoding the original input

o Activation Functions: Typically ReLU for hidden layers and linear or sigmoid for output layer

 Loss Function & Optimization

o Loss: Mean Squared Error (MSE), characterized by fast and stable convergence

o Optimizer: Adam optimizer, known for fast and stable convergence

 Training

o The model was trained on fixed number of epochs (e.g., 50–100) with a validation split

o Batch normalization and dropout layers were used to avoid overfitting

 Anomaly Detection

o After training, the model was used to reconstruct input samples

o A threshold was set based on reconstruction error from the validation set

o If reconstruction error > threshold → sample labeled as anomaly; else, normal

 Model Evaluation

Checked against metrics like Accuracy, F1-Score, Precision, Recall, and Confusion Matrix on a test set with both normal and anomalous data

 Deployment

o Saved the trained Autoencoder model using joblib or model.save()

o Implemented in the Django backend for real-time prediction via the web interface.

• Advantages

o Even able to learn complex non-linear pattern

o Unsupervised and capable of learning new types of anomalies

o Very effective with training on representative, clean normal data

Model Evaluation Metrics

A number of standard evaluation metrics were utilized to measure the performance of the anomaly detection models. As this is a binary classification task identifying whether a network connection is normal or anomalous these metrics are useful for measuring how well the models identify intrusions while reducing false alarms.

1. Accuracy

Accuracy is the proportion of total predictions (both normal and anomaly) that the model got correct.

$$\textbf{Accuracy} = \mathbf{(TP + TN) / (TP + TN + FP + FN)}$$

Where:

 TP = True Positives (correctly predicted anomalies)

 TN = True Negatives (correctly predicted normals)

 FP = False Positives (normal predicted as anomaly)

 FN = False Negatives (anomaly predicted as normal)

2. Precision

Precision measures how many of the predicted anomalies were actually true anomalies.

Precision = TP / (TP + FP)

3. Recall (Sensitivity or True Positive Rate)

Recall shows the number of real anomalies the model was able to identify..

Recall = TP / (TP + FN)

4. F1-Score

F1-Score is the harmonic mean of recall and precision. It offers a balanced score when both false negatives and false positives are important.

F1-Score = 2 * (Precision * Recall) / (Precision + Recall)

5. Confusion Matrix

Confusion matrix is a 2×2 table that shows how many correct and incorrect predictions were made by the model:

Predicted Normal Predicted Anomaly Actual Normal True Negative False Positive Actual Anomaly False Negative True Positive

6. ROC Curve and AUC

The Receiver Operating Characteristic (ROC) curve graphically plots the True Positive Rate (Recall) vs the False Positive Rate. The AUC (Area Under Curve) is a measure of the ability of the model to distinguish between normal and anomaly.

Model Evaluation and Comparison

In order to compare the performance of the models adopted for network anomaly detection, important metrics were analyzed: Precision, Recall, F1 Score, AUC Score, and Average Precision. They give a complete idea of each model's capability to differentiate between normal and abnormal network traffic.

Two models were tried:

- Isolation Forest (Anomaly detection using Tree-based model)
- Autoencoder (Reconstruction model using Neural network-based model)

Metric	Isolation Forest	Autoencoder
Precision	0.976	0.723
Recall	0.992	0.064
F1 Score	0.984	0.117
AUC Score	0.980	0.406
Average Precision	0.995	0.781

WEB INTERFACE USING DJANGO

To make the anomaly detection system user-friendly and accessible, a web-based interface was created with the Django framework. Django is a Python web framework that favors rapid development and elegant, pragmatic design. The interface makes it possible for users to work with the machine learning models without coding, making the solution deployable in real-time to practical environments.

Key Features:

 User Input Form

Django's form handling

system was utilized to create a web form through which users could directly input or upload network traffic feature values

 Backend Integration

The trained models (Autoencoder and Normal Isolation Forest) were imported to the Django backend via joblib. When input data was submitted, the backend preprocessed the input, routed it through the chosen model, and labeled the output as either "Normal" or "Anomaly."

 Real-time Inference

After submitting the form, the outcome is shown on the same or result redirect page, providing real-time feedback to the user..

 Result Visualization

The interface may optionally show other metrics like the anomaly score, confidence levels

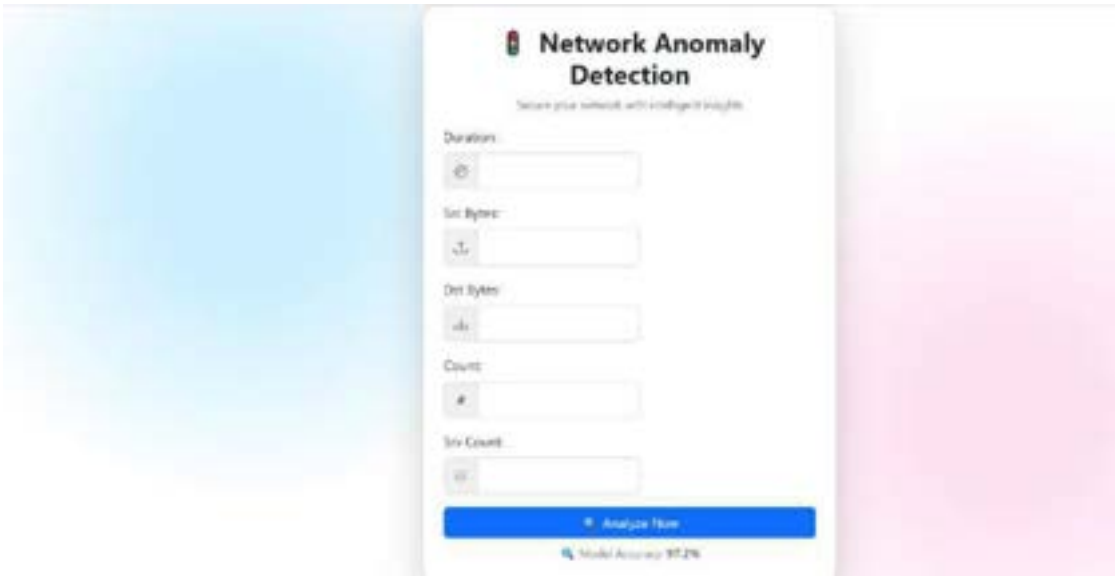
 Responsive Frontend

The interface may optionally show other metrics like the anomaly score, confidence levels.

User Input and Output Interface

The web interface built with Django offers a clean and interactive user interface for users to input network traffic data and get instant results for anomaly detection. The frontend is user experience-oriented, employing contemporary UI styling and visual feedback in order to make the tool usable even by non-technical personnel.

Input Interface



The input form enables users to provide specific attributes of a network connection, including:

 Duration: Time of the connection

 Src Bytes: Bytes sent from the source to the destination

 Dst Bytes: Bytes sent from the destination to the source

 Count: Number of connections to the same host

 Srv Count: Number of connections to the same service

Output Interface

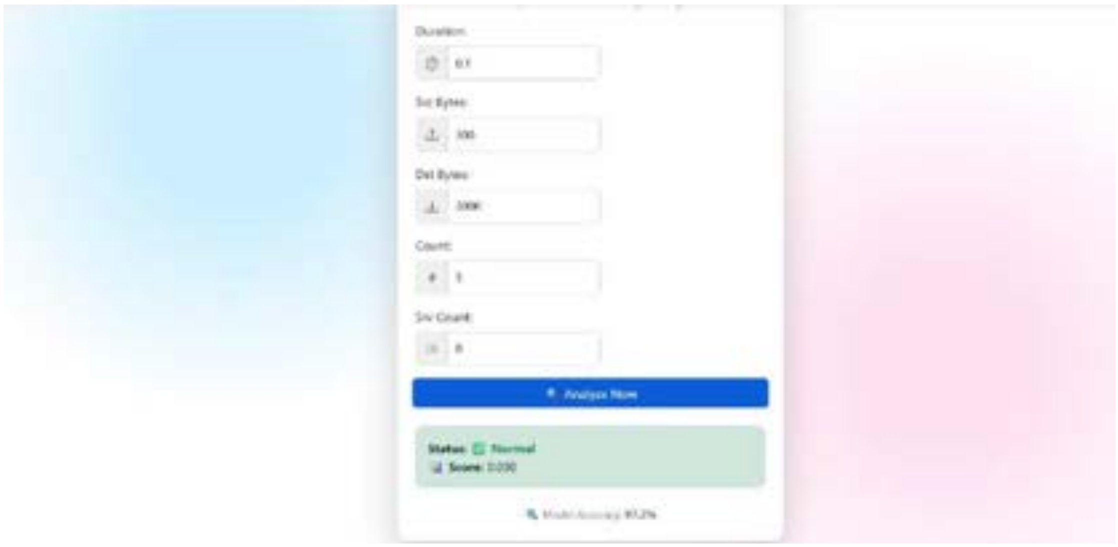
 When the input is submitted, the backend model calculates in real-time the data and provides a prediction outcome that is displayed prominently to the user.

 The output shows:

o Status: Whether the traffic is Normal ✅ or Anomaly ❌

o Score: Anomaly score calculated by the model

o Model Accuracy: A reference marker of the model's evaluation accuracy


Duration: 0.1

Src Bytes: 100

Dst Bytes: 300

Count: 1

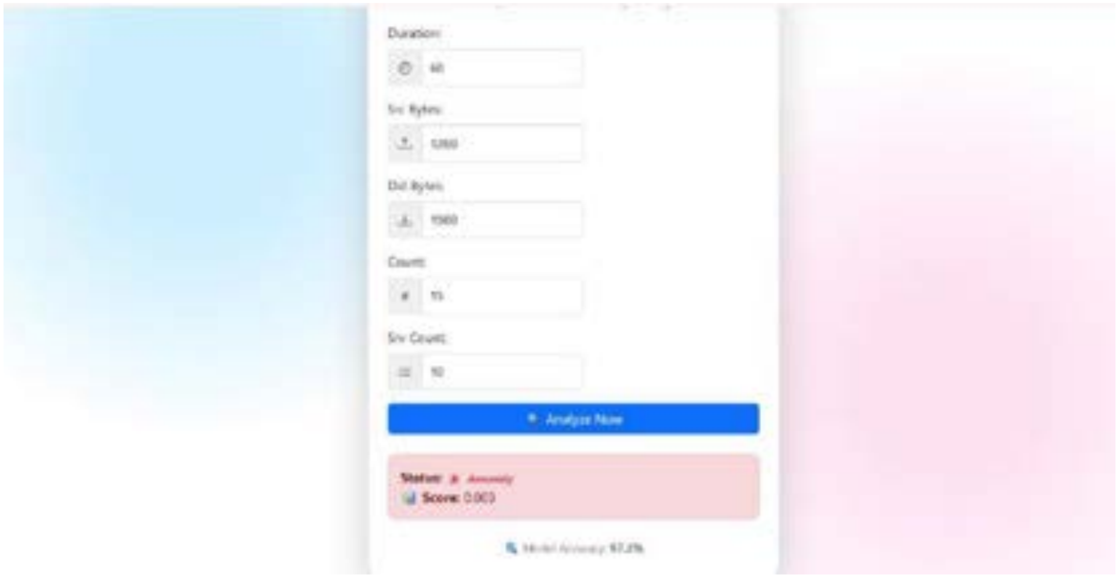
Srv Count: 0

Status: Normal

Score: 0.030

Model Accuracy 97.2%

o The result is shown within a color-coded response card, with green indicating Normal.


Duration: 60

Src Bytes: 1200

Dst Bytes: 1500

Count: 15

Srv Count: 10

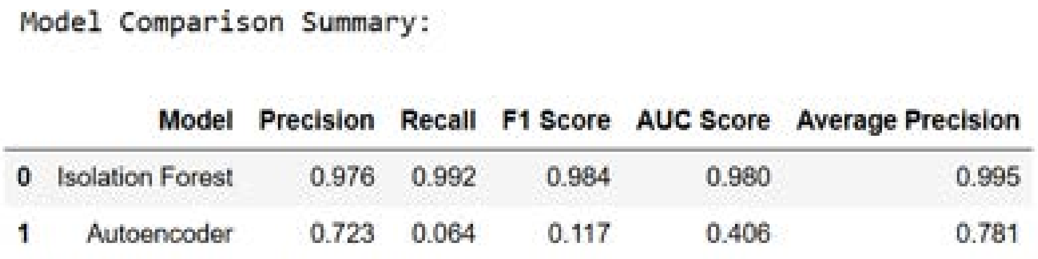
Status: Anomaly

Score: 0.003

Model Accuracy 97.2%

o The result is shown within a color-coded response card, with red indicating Anamoly.

Result Interpretation


Model Comparison Summary:

	Model	Precision	Recall	F1 Score	AUC Score	Average Precision
0	Isolation Forest	0.976	0.992	0.984	0.980	0.995
1	Autoencoder	0.723	0.064	0.117	0.406	0.781

 Isolation Forest performed exceptionally with a Precision value of 97.6% and Recall value of 99.2%, which resulted in a very high F1 Score value of 98.4%, which reflects that it correctly identifies anomalies with the least false positives and false negatives.

 The Autoencoder, although with fair Precision (72.3%), had very low Recall (6.4%), indicating failure to identify a majority of the anomalies and thus a low F1 Score of (11.7%).

 Isolation Forest AUC Score is 0.980, indicating very high ability to differentiate between normal and anomalous traffic, while Autoencoder's AUC of 0.406 is indicative of near-random performance.

 Average Precision also testifies to the strength of Isolation Forest (0.995) over the Autoencoder (0.781), asserting that the Autoencoder tends to classify anomalies as normal traffic.

 As a whole, the results unambiguously reflect that Isolation Forest is the stronger model for anomaly detection in this project, predominantly outperforming the Autoencoder on all key evaluation measures.

Result Interpretation – Django Web Application

 The Django application utilizes the Isolation Forest model, which renders extremely high-quality results (F1 Score: 98.4% and Precision: 97.6%).

 When network traffic information is input by users, the system immediately displays whether it's Normal ✅ or Anomaly ❌, depending on the output of the model.

 Along with the result, it also displays an anomaly score and the overall accuracy of the model (97.2%) for the confidence of the user.

 The result is displayed on a clean, color-coded web page—green for normal and red for anomaly, which is simple to comprehend.

 This renders the Django app a quick, easy, and robust tool for real-time detection

of network intrusions.

Visualizations

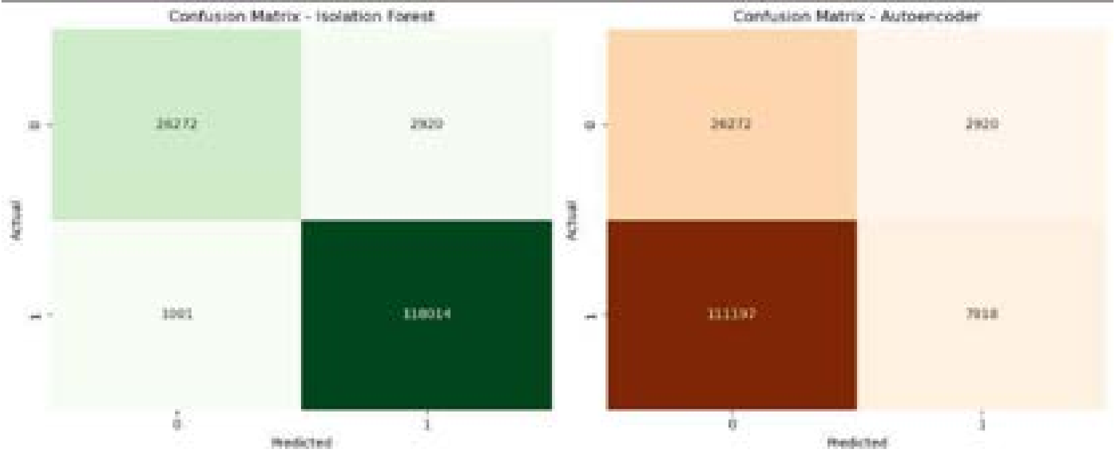
In order to comprehend and analyze the performance of the anomaly detection models more effectively, a number of visualizations were constructed using Matplotlib and Seaborn in Python. These plots facilitated a better understanding of model behavior, accuracy, and detection more easily.

1. Confusion Matrix

 Displays the number of True Positives, True Negatives, False Positives, and False Negatives.

 Utilized to visually verify the accuracy and classification performance of the two models.

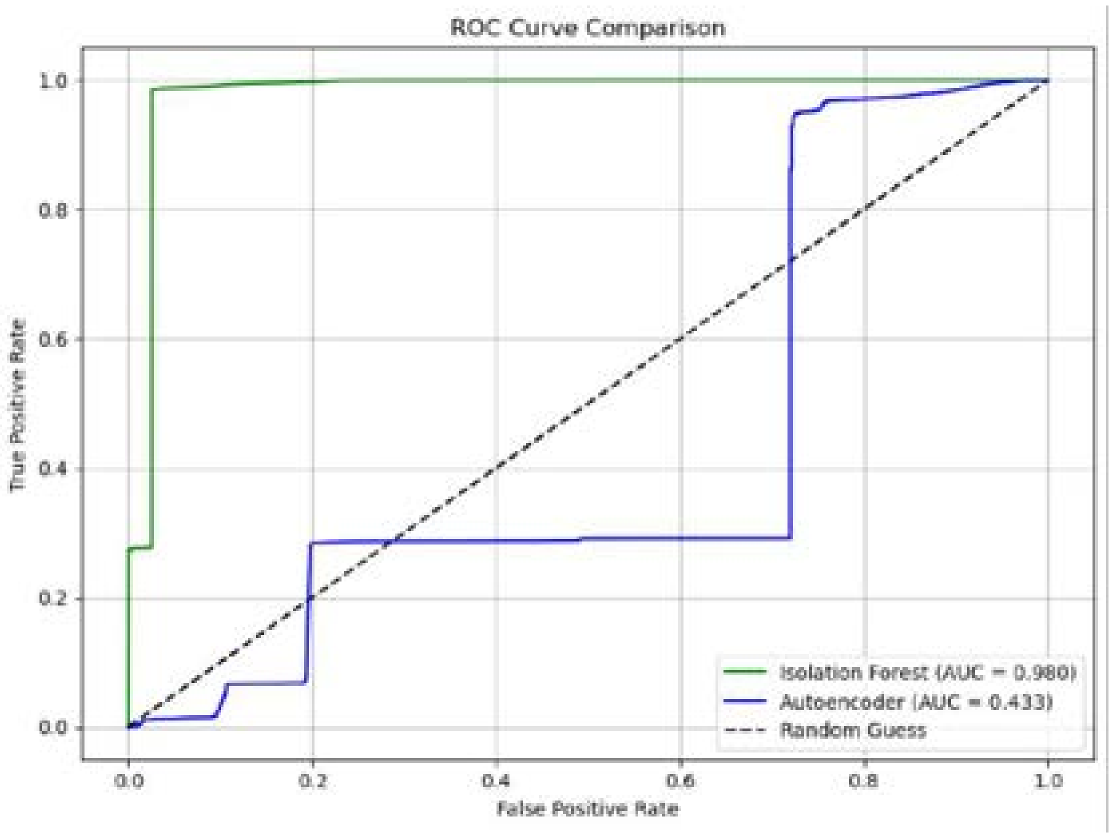
 Observation: Isolation Forest has a significant diagonal dominance, reflecting high accuracy for both normal and anomaly classes.



2. ROC Curve (Receiver Operating Characteristic)

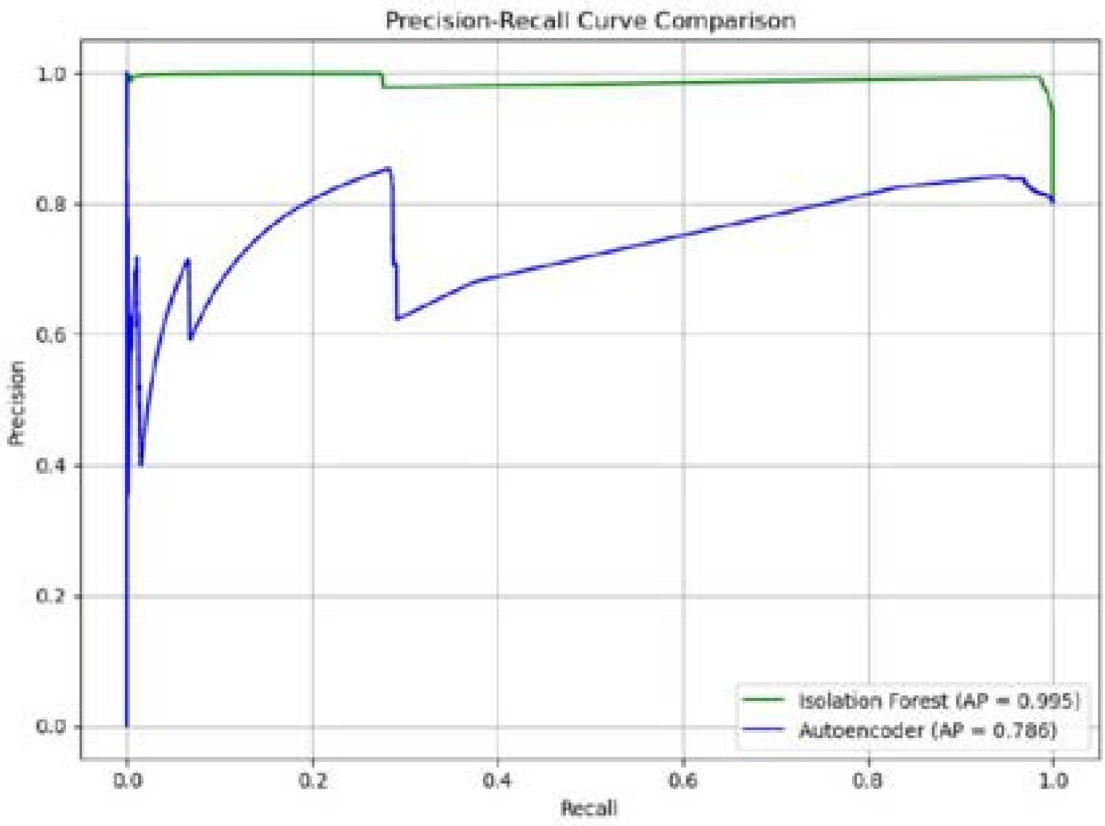
 Plots True Positive Rate (Recall) against False Positive Rate, showing how well the model distinguishes between the two classes.

 The AUC Score (Area Under Curve) for Isolation Forest was 0.98, indicating excellent performance.



3. Precision-Recall Curve

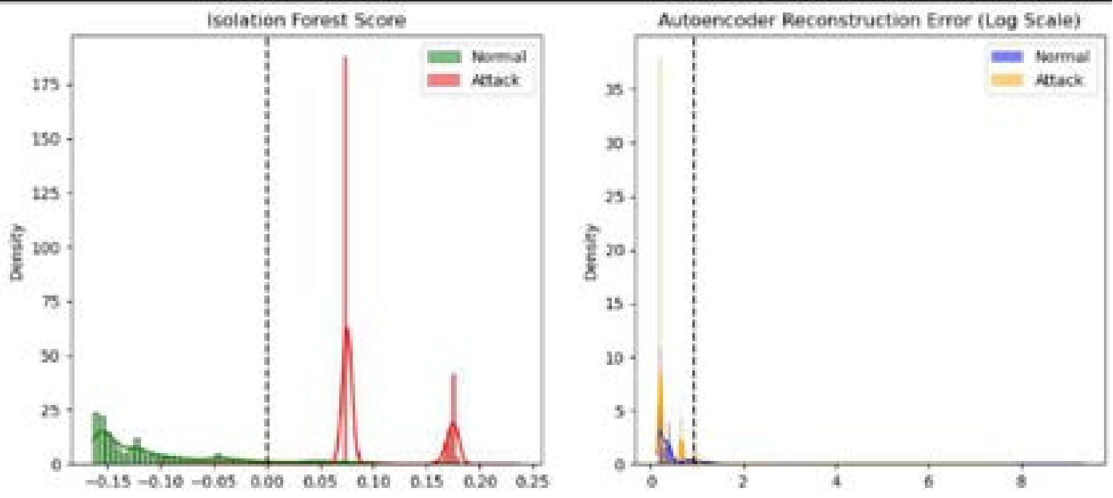
- Illustrates the trade-off between precision and recall for varied threshold settings.
- Helpful in assessing performance on imbalanced datasets, where anomalies are very few compared to normal data.



4. Anomaly Score Distribution

 Histogram or KDE plot of the distribution of anomaly scores across all samples.

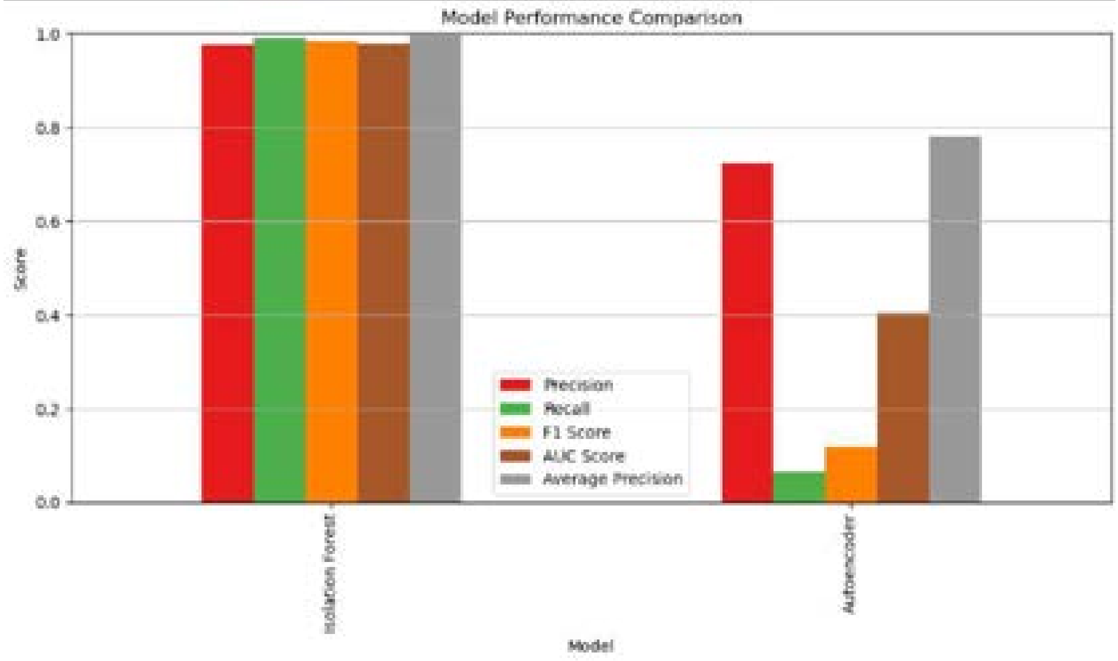
 Assists in determining an appropriate threshold to mark as anomalies



5. Bar Chart – Model Comparison

 A side-by-side bar graph providing measures such as like Precision, Recall, F1 Score, and AUC for both Isolation Forest and Autoencoder.

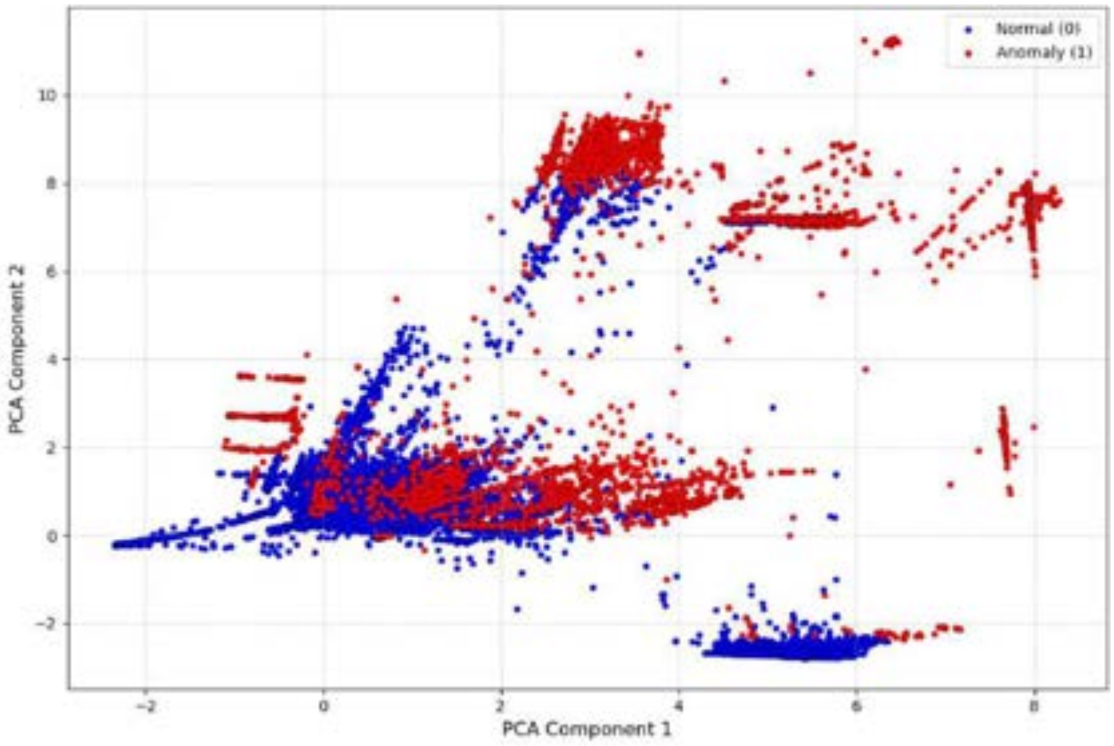
 Clearly displays which model performs better.



6. PCA Plot – Data Separation Visualization

• A scatter plot of 2D data created through Principal Component Analysis (PCA) to project high-dimensional data onto two components.

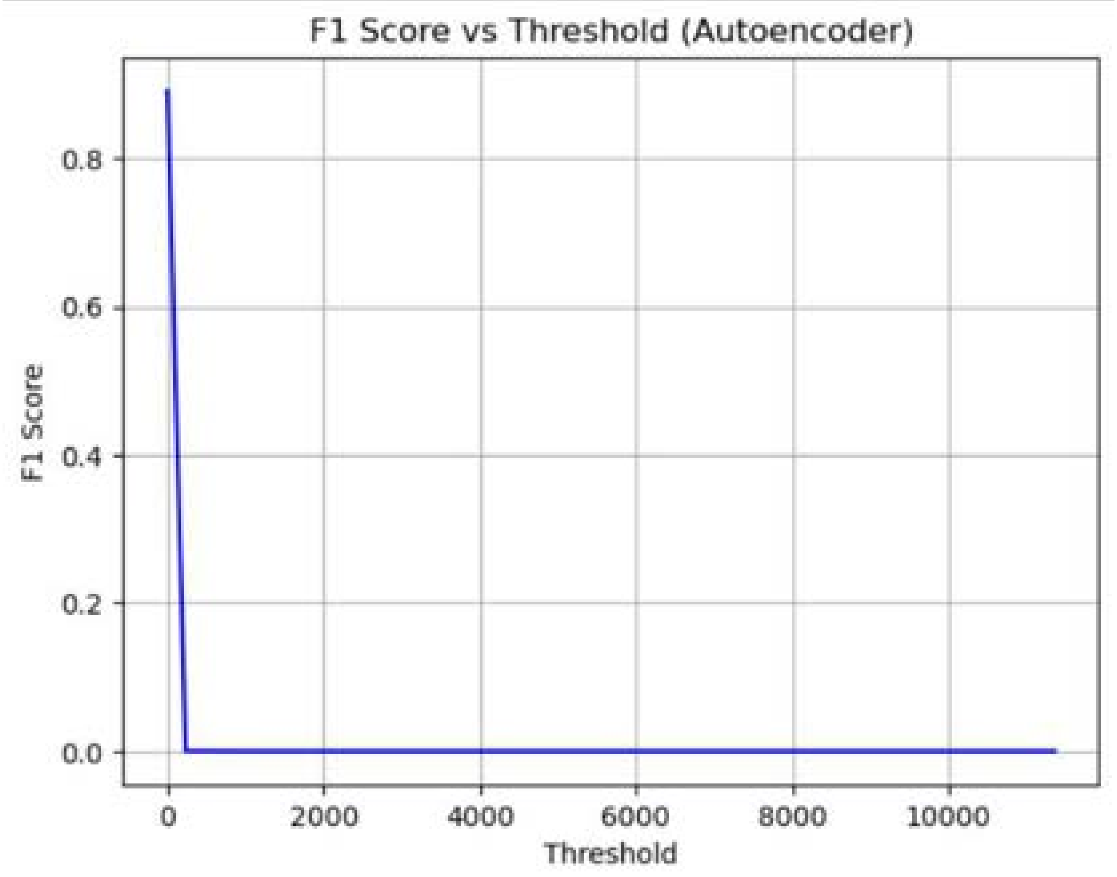
• Visually illustrates the distinction between normal and anomalous data points, and aids in understanding model appropriateness.



7. Threshold vs. F1 Score – Threshold Optimization

A line graph illustrating how the F1 Score varies with varying values of the threshold used for anomaly classification.

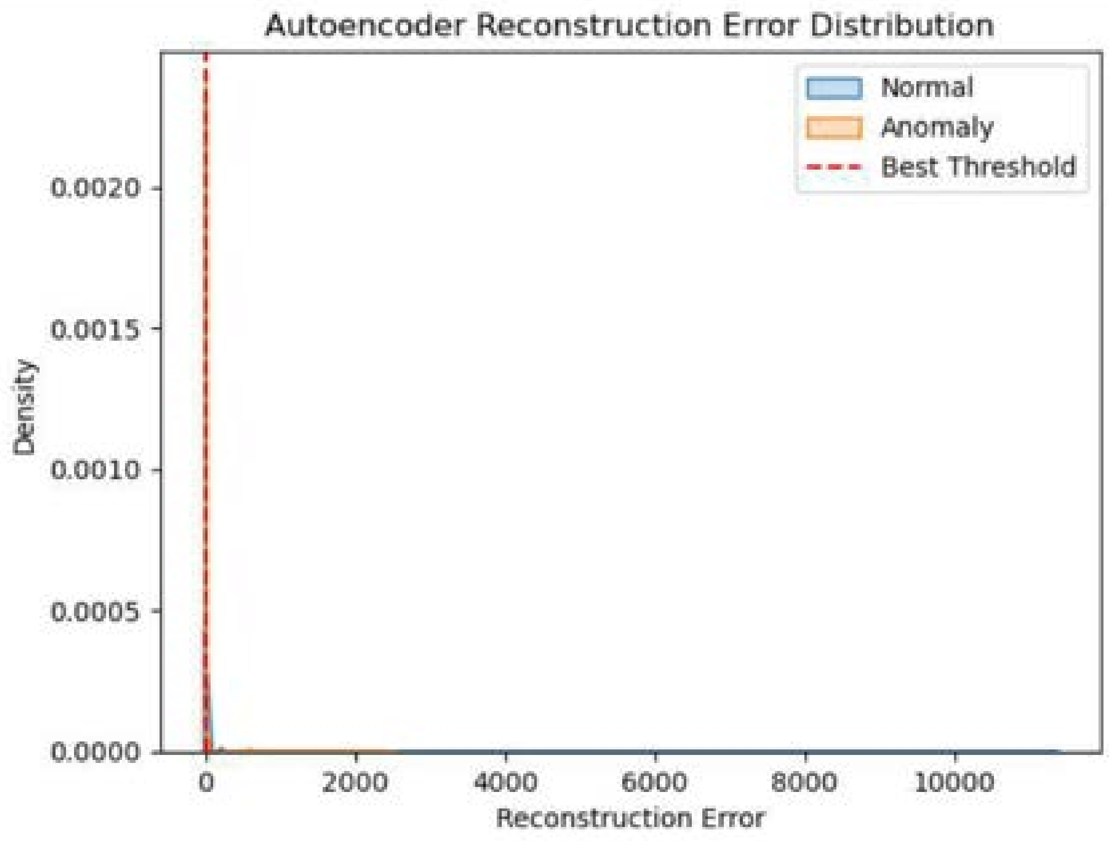
• Aids in finding the ideal threshold value that provides the optimal trade-off between precision and recall for effective anomaly detection.



8. Reconstruction Error – Autoencoder Performance

• A line or histogram plot of reconstruction error between original and reconstructed inputs in the Autoencoder model.

• Explicitly shows that normal data has little reconstruction error, whereas anomalies have greater error, assisting in the determination of a detection threshold.



Network Anomaly Detection App using Streamlit

This is an interactive web application for detecting anomalies in network traffic using Machine Learning. Built with Streamlit, it uses the Isolation Forest algorithm trained on the KDD Cup 1999 dataset.

Technologies:

Python

Streamlit

Scikit-learn

Pandas & NumPy

Plotly & Seaborn

Matplotlib

View Live: https://network-anomaly-app-c3tbeyyqqfwmnoi5efsxvj.streamlit.app

You can:

- Visualize and navigate the dataset
- Make predictions with user input
- Upload your network data (CSV)
- View performance metrics such as ROC and PR curves

Features

Interactive Dashboard – Sleek UI to keep track of network traffic anomalies.

Upload your own CSV files – Try out the model against custom data.

Visualizations – ROC, PR curves, pie charts, histograms, and PCA plots.

Real-Time Prediction – Input network parameters and receive immediate feedback.

Anomaly Score Gauge – Get a visual indication of how risky your data point is.

Top Anomalies Table – View most suspicious traffic patterns.

Challenges Faced

1. High Dimensionality of Data

The dataset of KDD Cup 1999 includes many features, some of which are noisy and redundant. Detection and identification of the most relevant features necessitated thorough preprocessing and analysis.

2. Imbalanced Dataset

A main challenge was the disproportion between regular and abnormal data, which influenced model training as well as making it harder to attain high recall for abnormalities, particularly using the Autoencoder.

3. Threshold Adjustment

Finding the right threshold for anomaly detection was imperative. Incorrect thresholding resulted in misclassifications, necessitating exhaustive experimentation with precision-recall and F1 score curves.

4. Combining ML Models with Django

Loading and executing machine learning models within a real-time Django web application presented issues in data serialization, preprocessing stability, and latency management.

5. Autoencoder Performance

In contrast to Isolation Forest, the Autoencoder had difficulty attaining high recall. Architecture adjustment, hyperparameter tuning, and preventing overfitting took considerable time and testing.

Future Enhancements

1. Real-Time Data Stream Integration

Expand the system to process real-time network traffic with software such as Kafka, SocketIO, or Wireshark, providing ongoing and automatic anomaly detection.

2. Model Ensemble Approach

Merge various models (Isolation Forest + Autoencoder + LSTM) to enhance overall detection accuracy through ensemble methods for increased reliability.

3. Dashboard & Visualization

Construct an enhanced admin dashboard via Chart.js or Plotly to track anomalies, trends, and score distributions over time with interactive charts.

4. Authentication & Role-Based Access

Add user authentication and varied levels of access to the Django web application for secure multi-user functionality.

5. Deployment on Cloud Platforms

Host the web application on Heroku, Render, or AWS for remote access and scalability for deployment in the real world.