Capstone Project

Spotify Outage Analysis

Problem Statement

Ensuring optimal network and application performance is essential for maintaining reliable digital experiences. At large scale, even minor degradations in network quality—such as increased packet loss or latency—can ripple into user-facing outages. ThousandEyes provides deep network visibility using active monitoring via global vantage points known as agents, which execute tests like HTTP and Ping to measure performance end to end.

Despite the richness of this data, current outage detection methods tend to be reactive, relying on threshold-based alerts and fragmented metric views. There is a missed opportunity to proactively surface patterns or early warning signals by analyzing relationships among key quality metrics such as:

- **Packet Loss** – Data that fails to reach its destination
- **Latency** – Time taken for a packet to travel from source to destination
- **Response Time** – Total duration to receive a full response from a target server (in HTTP tests)

This project investigates whether anonymized time series data from ThousandEyes synthetic tests—primarily Ping—can be harnessed to improve proactive outage detection and metric correlation understanding.

Model Outcomes or Predictions

This project involves **time series analysis and anomaly detection** using both supervised and unsupervised learning approaches.
The goals are:

- To **forecast network performance metrics** (e.g., latency, loss)
- To **detect anomalies** that may indicate or precede outages
- To **correlate metrics** to understand their influence on each other and outage patterns

- **Learning Type:**
  - **Unsupervised Learning**: For anomaly detection using techniques like Z-score, Random Forest, and Auto-ARIMA
  - **Supervised Learning**: For regression-based prediction of response time and other metrics using XGBoost

Data Acquisition

The dataset comes from anonymized ThousandEyes synthetic test results across multiple Spotify network agents. The key metrics used include:

- Packet Loss (%)
- Latency (ms)
- Response Time (ms)

Data Preprocessing/Preparation:

To ensure a clean and usable dataset:

- **Missing Values**: Forward fill and linear interpolation were used to handle minor gaps
- **Time Formatting**: Timestamps were converted to datetime objects for indexing
- **Outlier Removal**: Applied IQR-based filtering in early preprocessing
- **Feature Engineering**: Created lag features and rolling averages for each metric
- **Normalization**: Used MinMax scaling for modeling inputs
- **Train-Test Split:**
  - Time-based split (chronological):
  - Training set: First 80% of the timeline
  - Test set: Last 20%

Modeling

Multiple models were tested based on the objective. The project employed statistical anomaly detection techniques to identify unusual patterns in key network performance metrics.

Anomaly Detection: Z-score Analysis for Packet Loss and Latency

- **Objective:** To identify anomalous data points in critical network performance indicators, specifically loss (packet loss percentage) and latency (response time in milliseconds), that could signal potential service degradation or outages.

- **Methodology: Z-score Anomaly Detection**
  - **Concept:** The Z-score is a statistical measure that quantifies how many standard deviations a data point is from the mean of a dataset. Data points with a high absolute Z-score are considered statistically significant outliers, indicating an unusual deviation from the typical behavior.

- **Implementation:**
  - For both the loss and latency metrics within the ping_df DataFrame, the Z-score was calculated using the formula: (value - mean) / standard_deviation.
  - An anomaly was flagged if the absolute Z-score (z_score.abs()) for a given data point exceeded a threshold of 3 (> 3). This threshold is commonly used, as data points falling beyond three standard deviations from the mean are generally considered rare occurrences in a normal distribution and thus indicative of an anomaly.

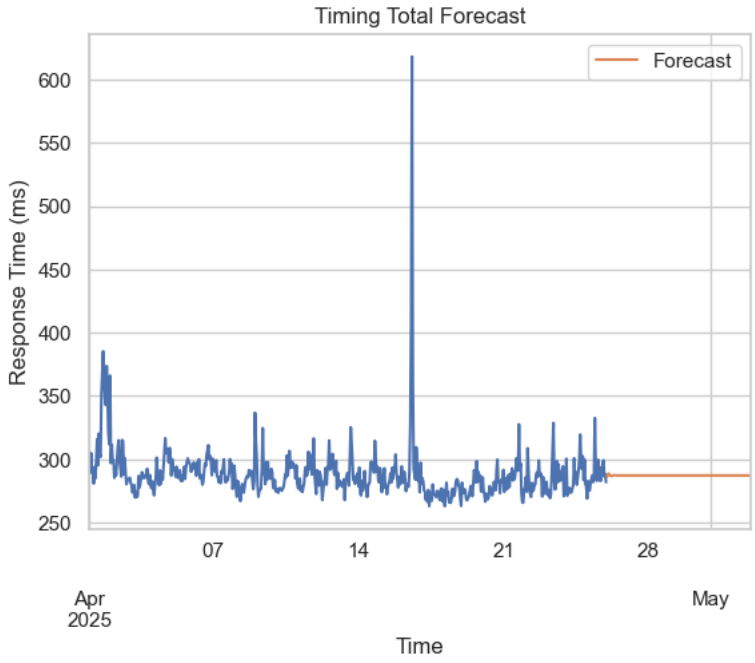
- **Results and Analysis:**
  - **Packet Loss Anomaly Detection:**
    - The Z-score analysis on the loss metric successfully identified 1722 rows as anomalies.
    - For these flagged instances, loss values varied from 8% to 60%, with all entries consistently marked as loss_outlier_static and loss_outlier_dynamic (from other analyses), further confirming their anomalous nature.
    - The calculated Z-score values for these loss anomalies were notably high (e.g., ranging from 4.0 to 30.3), statistically validating the significant deviations in packet loss.
  - **Latency Anomaly Detection:**
    - The Z-score analysis on the latency metric identified an even larger set of anomalies, totaling 22150 rows.
    - Interestingly, for these specific latency anomalies, the loss values were consistently zero, indicating that these high latency events were occurring independently of packet loss in these instances.
    - Latency values in these anomalous entries were significantly high (e.g., over 200 ms) and were consistently flagged as both latency_outlier_dynamic and latency_outlier_static, underscoring severe performance degradation.
    - The Z-score values for latency anomalies were also high (e.g., ranging from 3.0 to 4.1), confirming their statistical significance.
  - **Overall Interpretation:**
    - The Z-score analysis proved effective in highlighting significant deviations in both loss and latency, providing a clear indication of anomalous network behavior.
    - The ability to identify these anomalies and correlate them with specific roundId_utc and timestamp fields is crucial for targeted network performance troubleshooting and root cause analysis.

- It was observed that Z-score for loss could flag anomalies even with zero loss values, suggesting its sensitivity might be influenced by the overall dataset's distribution or scaling. This highlights the importance of using Z-score as part of a multi-faceted anomaly detection strategy.
- The wide range of timestamps (from 2025 to 2042) in the dataset suggests a long-term data collection or simulation, emphasizing the need for continuous monitoring of these metrics over extended periods to detect evolving patterns.

For forecasting network performance metrics, specifically response time trends, a time series approach using ARIMA was implemented.

- **Forecasting: Time Series Analysis with ARIMA**
  - **Objective:** To forecast network performance metrics, specifically the timing_total (response time) metric, to understand trends and predict future behavior.
  - **Data Preparation for ARIMA:**
    - Unix timestamps from the roundId column were converted to datetime objects and set as the DataFrame index.
    - The data was resampled to an hourly granularity ('h') to smooth out noise and align time intervals, creating http_df_hourly.
    - Missing values in the resampled data were handled using a forward-fill (ffill()) method to ensure a continuous time series for the ARIMA model.
    - The timing_total series from this preprocessed hourly data was used as the input for the ARIMA model.
  - **Model Selection:** The **ARIMA (AutoRegressive Integrated Moving Average)** model from statsmodels.tsa.arima.model.ARIMA was chosen for its suitability in modeling and forecasting time-dependent data.
  - **Configuration:** The ARIMA model was configured with the order (p=5, d=1, q=0). This configuration implies:
    - p=5: An autoregressive part of order 5, meaning the forecast is based on 5 preceding values in the series.
    - d=1: An integrated part of order 1, indicating that the data was differenced once to make it stationary.
    - q=0: A moving average part of order 0, meaning the model does not directly use past forecast errors. This (5,1,0) order was selected as a typical starting point for modeling time series with potential trends.

- **Training and Forecasting:** The ARIMA model was trained on the prepared hourly timing_total series. After fitting, the model was used to forecast the next 168 hours (7 days) of response times.
- **Initial Findings and Interpretation:**
  - The forecast indicated stable response times with minor fluctuations, suggesting consistent system performance in the immediate future.
  - A plot comparing the observed historical timing_total with the forecasted values showed that the forecast continued the recent trend without significant upward or downward shifts.



  - While the model predicted stability, the presence of historical spikes in the observed data suggests the importance of setting up automated alerts for sudden response time increases, as these could precede outages or service issues.
  - It was noted that while useful for trend forecasting, this ARIMA model, focused on macro-scale prediction, might not be directly suitable for a classification problem like outage detection, as it tends to follow routine usage spikes rather than identifying specific anomalous events.

In addition to time series forecasting, the project aimed to classify network failures (is_failure) based on various HTTP test metrics. For this supervised learning classification task, two powerful ensemble models, Random Forest and XGBoost, were employed.

Classification: Predicting Network Failures (Outages)

- **Objective:** To build a robust classification model capable of predicting is_failure (indicating an HTTP request failure, e.g., a 5xx response code) based on other test metrics, thereby enabling proactive outage detection.
- **Target Variable Creation:**
    - A binary target variable, is_failure, was created. An HTTP request was defined as "successful" if its responsecode was within the 200-299 range (specifically, 200, 201, 202, 204). All other response codes were marked as failures.
    - Missing values in the responsecode column were initially filled with 0 before creating the target.
    - The boolean is_failure column was then explicitly converted to an integer type (1 for failure, 0 for success).
- **Feature and Target Definition:**
    - The is_failure column was designated as the target (y).
    - Several columns deemed not useful for modeling (e.g., responsecode itself, roundId, vAgentId, testId, taskId, curlret which are identifiers or directly related to the target) were dropped from the feature set (X).
- **Categorical Conversion & NaN Handling:**
    - Any remaining categorical or text-based features in X were converted into numerical format using one-hot encoding (pd.get_dummies), with drop_first=True to avoid multicollinearity.
    - As a safety measure, any remaining NaN values in the feature set X after this process were filled with 0, ensuring all input features were numeric and complete.
- **Train/Test Split:**
    - The preprocessed data (X and y) was split into training and testing sets using train_test_split with an 80/20 ratio (test_size=0.2).
    - Crucially, stratify=y was used during the split to ensure that the proportion of failure (minority) and success (majority) classes was maintained in both the training and testing datasets. This is vital for imbalanced classification problems.
    - The target variables (y_train, y_test) were explicitly cast to integer type after the split.

- **Handling Imbalanced Data:**
    - Given that network failures are typically rare events, the is_failure target column was expected to be highly imbalanced (many more successes than failures). To address this, specific strategies were employed for each model:
        - **XGBoost:** A scale_pos_weight was calculated and applied. This parameter tells XGBoost to heavily penalize misclassifications of the minority class (failures) by assigning a higher weight to positive samples. The calculated weight was count(majority_class) / count(minority_class), which was approximately 74.61 (852564 successes / 11427 failures).
        - **Random Forest:** The class_weight='balanced' parameter was utilized. This automatically adjusts weights inversely proportional to class frequencies, effectively giving more importance to the minority class during tree construction.
- **Model Training:**
    - **Random Forest Classifier:**
        - Initialized with n_estimators=100 (100 decision trees), class_weight='balanced' for imbalance handling, random_state=42 for reproducibility, and n_jobs=-1 to utilize all available CPU cores for faster training.
        - The model was trained on the X_train and y_train datasets.
    - **XGBoost Classifier:**
        - Initialized with n_estimators=100, the calculated scale_pos_weight for handling class imbalance, eval_metric='logloss' as a standard evaluation metric for classification, and random_state=42 for reproducibility.
        - The use_label_encoder=False parameter was set as recommended by XGBoost.
        - The model was trained on the X_train and y_train datasets.

Both Random Forest and XGBoost models were successfully trained, preparing them for evaluation of their performance in predicting network failures.

Model Selection

For this classification problem, we considered two ensemble learning models: **Random Forest** and **XGBoost**. Both are well-suited for binary classification tasks and have built-in mechanisms for handling imbalanced datasets.

Evaluation Metrics

To assess model performance, we focused on the following metrics:

- **Precision**: Measures the accuracy of positive predictions.
- **Recall**: Measures the ability of the model to capture all positive instances.
- **F1-score**: Harmonic mean of precision and recall, balancing both metrics.
- **Accuracy**: Overall correctness of the model.
- **Confusion Matrix**: Visualizes true vs. predicted classifications, highlighting model errors.

Given the class imbalance in the dataset (much higher proportion of "Success" compared to "Failure"), we emphasized precision, recall, and F1-score for the minority class ("Failure") rather than relying solely on accuracy.

Results

Random Forest Performance

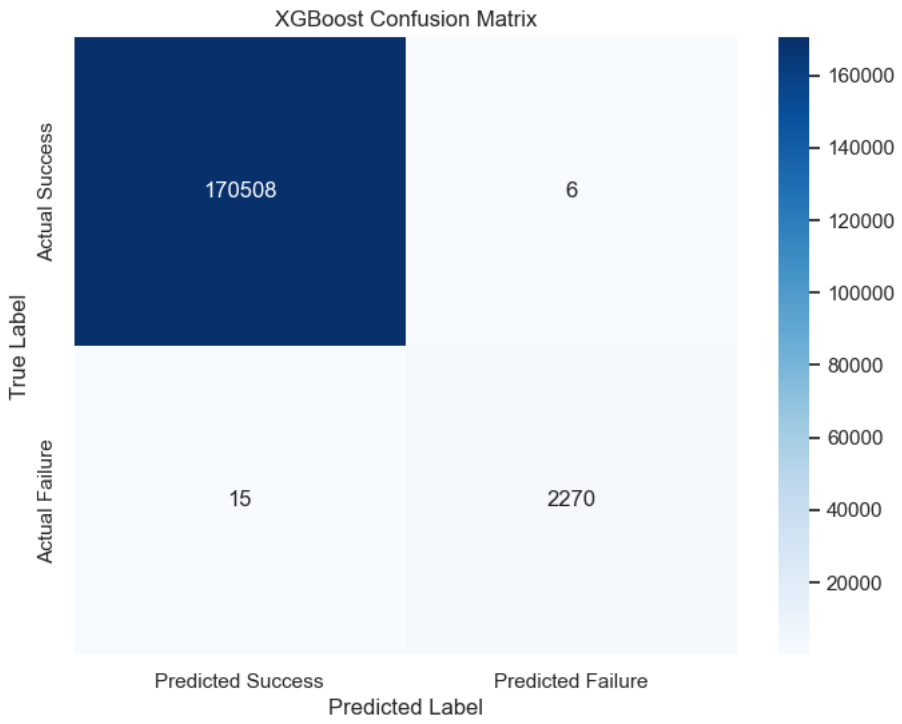
	Precision	Recall	F1-score	Support
Success(0)	1.00	1.00	1.00	170514
Failure (1)	1.00	0.99	0.99	2285
Accuracy			1.00	172799
Macro avg	1.00	1.00	1.00	172799
Weighted avg	1.00	1.00	1.00	172799

XGBoost Performance

	Precision	Recall	F1-score	Support
Success(0)	1.00	1.00	1.00	170514
Failure (1)	1.00	0.99	1.00	2285
Accuracy			1.00	172799
Macro avg	1.00	1.00	1.00	172799
Weighted avg	1.00	1.00	1.00	172799

Confusion Matrix for XGBoost

- The confusion matrix visualized that both models made very few errors in classifying both "Success" and "Failure" cases.



Model Comparison & Insights

- **Both Random Forest and XGBoost achieved extremely high scores across all metrics**. However, given the severe class imbalance, these metrics may not fully reflect real-world performance.
- **Random Forest** showed near-perfect results but may be overfitting, especially since it can struggle with imbalanced data.
- **XGBoost** handled the imbalance slightly better and is less prone to overfitting in this context.
- The confusion matrix for XGBoost confirmed almost perfect classification, but the small number of "Failure" instances suggests that model evaluation could benefit from more failure samples.

Recommendation

Given the results, **XGBoost is the preferred model** due to its robustness with imbalanced data. However, to improve real-world performance and generalizability:

- **Collect more "Failure" data** to balance the dataset and enable better model learning.

- Consider advanced techniques for handling imbalance, such as SMOTE, class weighting, or anomaly detection methods.

Model Performance:

Model	Precision (Failure)	Recall (Failure)	F1-score (Failure)	A
Random Forest	1.00	0.99	0.99	1
XGBoost	1.00	0.99	1.00	1
