Project Report

on

Anomaly Detection in Web Application Logs Using Machine Learning

Submitted in partial fulfilment of the course of

PG-Diploma
in

Big Data Analytics

From **C-DAC ACTS (Bangalore)**

Guided by:

Mr. Abhishek Chavan

Presented by:

Mr. Arpit Raj	PRN: 230350125013
Mr. Himanshu Warudkar	PRN: 230350125025
Mr. Aishwary Pratap Singh	PRN: 230350125004
Mr. Kulkarni Suhas Sureshrao	PRN: 230350125032

Candidate's Declaration

We hereby certify that the work being presented in the report titled: **Anomaly Detection in Web Application Logs using Machine Learning**, in partial fulfilment of the requirements for the award of PG Diploma Certificate and submitted to the departmentof PG-DBDA of the C-DAC ACTS Bangalore, is an authentic record of our work carriedout during the period, 1st August 2023 to 31th August 2023 under the supervision of Mr.Abhishek Chavan, C-DAC Bangalore. The matter presented in the report has not been submitted by us for the award of any degree of this or any other Institute/University.

Name and Signature of Candidate:

Mr. Arpit Raj	PRN: 230350125013
Mr. Himanshu Warudkar	PRN: 230350125025
Mr. Aishwary Pratap Singh	PRN: 230350125004
Mr. Kulkarni Suhas Sureshrao	PRN: 230350125032

Counter Signed by:

- - - - - - - - - - -

CERTIFICATE

TO WHOMSOEVER IT MAY CONCERN

This is to certify that

Mr. Arpit raj
Mr. Himanshu Warudkar
Mr. Aishwary Pratap Singh
Mr. Kulkarni Suhas Sureshrao

Have successfully completed their project on

Anomaly Detection in Web Application Logs Using Machine Learning

Under the guidance of

Mr. Abhishek Chavan

Acknowledgement

This project **"Anomaly Detection in Web Application Logs Using Machine Learning"** wasa great learning experience for us and we are submitting this work to Advanced Computing Training School (CDAC ACTS).

We all are very glad to mention the name of **Mr. Abhishek Chavan** for his valuable guidance to work on this project. His guidance and support helped us to overcome various obstacles and intricacies during the course of project work.

Our most heartfelt thanks go to **Ms. Shilpa** (Course Coordinator, **PG-DBDA**)who gave all the required support and kind coordination to provide all the necessitieslike extra Lab hours to complete the project and throughout the course up to the last day at C-DAC ACTS, Bangalore.

From,
The whole team.

Table of Content

Abstract

This project focuses on automating anomaly detection in web application logs using machine learning. The aim is to efficiently identify abnormal patterns, including potential security threats, in these logs. The project employs machine learning algorithms like Isolation Forest, SVM, and CatBoost to build predictive models. These models differentiate between normal and anomalous log entries, with a focus on high accuracy and low false positives.

The chosen CatBoost algorithm effectively handles categorical features present in web application logs. These features, like HTTP status codes help pinpoint unusual behavior. The project's methodology includes data preprocessing, feature engineering, model selection, hyperparameter tuning, and performance assessment. Open-source web application log data is used for training and evaluation, with metrics like precision, recall, F1-score guiding performance evaluation.

Incorporating PyCaret and PyOD into the project has significantly streamlined the anomaly detection process in web application logs. PyCaret's automation of tasks such as data preprocessing, feature engineering, and model selection has expedited the development cycle, enabling efficient exploration of algorithms like Isolation Forest, SVM, and CatBoost. This automation enhances the project's agility and accuracy in identifying abnormal patterns. Moreover, PyOD's specialized anomaly detection techniques offer a wider spectrum of options beyond the core algorithms, bolstering the project's ability to detect intricate security threats and anomalies within web application logs. Together, the integration of PyCaret and PyOD empowers the project to deliver a robust and effective solution for web application security.

Results demonstrate the system's ability to detect security breaches and anomalies in web application logs. Automation enables proactive security threat mitigation, enhancing web application security. This project offers a scalable solution by utilizing advanced algorithms and leveraging categorical features, enabling precise anomaly identification and improved security incident response.

Chapter-1

INTRODUCTION

1.1 Web Application

A web application is a type of software program that operates through web browsers over the internet. Unlike traditional software, it doesn't require installation and can be accessed using a simple web address. Hosted on remote servers, web apps provide users with interactive features and dynamic content. They use a client-server architecture: users' browsers (the clients) send requests to servers, which process the requests and send back responses. This enables tasks such as online shopping, social networking, and document editing. Web apps are versatile, platform-independent, and can be accessed from various devices. They're continuously updated by developers and can offer a responsive design that adapts to different screens.

1.2 Logs

The events that take place in a system are recorded in logs. Logs on web application typically record information about HTTP requests and responses, such as the date and time of the request, the source and destination IP addresses, the requested URL, the HTTP method (GET, POST, etc.), the status code the server returned, and other specifics.

Logs are essential for maintaining and fixing web applications. Administrators can use them to find performance problems, security risks, and other issues that might affect the web application's availability or dependability. Organizations can make sure that their web apps are compliant with regulations and shield themselves from fines or legal action by analyzing log data.

1.2.1 Types of Logs

Logs are produced in a wide variety of ways by various systems and programs. The following are some of the most typical sorts of logs:

1] System logs: These logs store details about the operating system, such as warnings, failures, and system events.

2] Application logs: These logs record details on how certain applications behave, such as errors, exceptions, and other events.

3] Security Logs: These logs for security events, such as login attempts, modifications to system configurations, and other security-related events are recorded in these logs.

4] Network logs: These logs include details regarding network activity, such as traffic volume, traffic trends, and other activities connected to the network.

5] Server logs: These logs record details about web applications, such as web traffic, problems, and other server performance-related occurrences.

6] Database logs: These logs record details of database transactions, such as errors, warnings, and other occasions relating to the operation of the database.

7] Audit logs: These logs record details of user activity, such as login attempts, access requests, and other user behavior-related occurrences.

System events, such as startup and shutdown events, hardware events, and other events relating to system performance, are detailed in event logs. Logs are a significant source of data for performance analysis, security monitoring, and troubleshooting in general. Organizations may improve performance, avoid security problems, and optimize their whole IT infrastructure by examining logs to get insights into the behavior of their systems and applications.

1.3 Status Code

A status code is a three-digit code that the server returns to the client to indicate the status of the response to an HTTP request in the context of web applications and HTTP requests. It gives details on the outcome of the client's request and is a component of the HTTP protocol.

Common status codes include the following:

- 200 OK: The request has been completed successfully by the server.
- 301 Permanently Moved: The requested resource has been transferred permanently to another URL.
- 404 Not Found: The server was unable to locate the requested resource.
- 500 Internal Server Error: The server was unable to process the request due to an unforeseen circumstance.

Each status code has a distinct meaning that aids in resolving any problems that may arise with the web application or the client's request.

1.4 Web Application Logs

The information contained in web application logs is useful for understanding user behavior and streamlining web applications. The performance, usability, and security of web applications can be studied using the data that web applications keep, including IP addresses, URLs, user agents, and response codes. The purpose of this project is to use web application logs to learn more about how a web application is used to spot any potential security risks or anomalies. In this project, the log data will be cleaned, preprocessed, and analyzed using Python and other data analysis and machine learning packages.

Cleaning and preprocessing the raw log data is the initial phase in the analysis, and it involves extracting relevant information from the log files, such as IP addresses, timestamps, URLs, and user agents. After the data has been preprocessed, we will examine it using a variety of statistical and visualization approaches to learn more about the web application's usage trends. In the next step, we will use machine learning algorithms to detect anomalies or security threats in the log data. Anomalies can be caused by various factors such as bot traffic, server errors, or malicious attacks, and can have significant impact on the performance and security of the web application. We will use unsupervised learning algorithms such as Isolation Forest and DBSCAN to detect anomalies in the log data.

The volume of data produced by web applications has increased dramatically as a result of an increased reliance on web-based applications and services. A wealth of data may

be found in web application logs that can be utilized to understand user behavior, website performance, and security risks. Organizations that depend on web-based services must now analyze web application logs in order to get insight into their online applications, optimize their infrastructure, and guarantee the security of their systems.

The objective of this project is to use various data analysis and machine learning approaches to examine web application logs and get useful insights from them. Python will be used to do data analysis, alter data, and create machine learning models in order to find trends and abnormalities in web application log data. The pre-processing of the log data, feature engineering, and application of machine learning models to the pre-processed data will all be part of the project's efforts to distinguish between regular and abnormal traffic.

Several well-known Python libraries, including Pandas, NumPy, Matplotlib, and Scikit-Learn, will be used in the project to carry out data analysis and machine learning tasks. We will also use PyCaret, a high-level machine learning library, to streamline the machine learning process and perform hyperparameter tuning to optimize our models' performance. Also, PyOD a library which help to deal with anomaly.

1.5 Problem Statement:

The way we work, live, and conduct business has all been transformed by the internet. Yet as there are more and more online enterprises, cyber threats are a huge worry. In 2020 alone, there were over 5 billion cyberattacks, according to a recent Symantec analysis. In addition to risking the security of online enterprises, these attacks also result in financial losses, harm to brand reputation, and legal liability. An efficient method to track and examine web traffic in order to spot irregularities and stop cyberattacks is web application log analysis. The user's IP address, the requested URL, the date and time of the request, the user agent, and the server response code are all recorded in web application logs for each request made to the server.

Web application log analysis is problematic since it may be a time-consuming and difficult operation, especially for companies that get a lot of requests. It calls for proficiency in data analysis as well as a solid grasp of web application protocols. Also, because web application logs contain a lot of data and abnormalities may be subtle and hard to spot, it might be challenging to find anomalies in the logs.

To overcome these difficulties, we suggest a system for analyzing web application logs that makes use of machine learning algorithms to spot anomalies. A model will be trained by the system using supervised learning methods on a labelled dataset of web application logs, where the labels denote whether a log entry is typical or abnormal. Based on the properties of a new log entry, the trained model will be used to forecast whether it is normal or abnormal.

Businesses will have a quick and easy way to check their web application logs for anomalies and potential cyberattacks according to the suggested solution. Businesses will be able to swiftly spot and respond to abnormal activity, lowering the likelihood of financial losses, reputational harm to their brands, and legal liability.

Chapter-2

LITERARY SURVEY

A literary survey is an important step in any research project, as it involves reviewing the existing literature on a topic to identify what has already been studied and what gaps exist in current knowledge. In the case of a web application log analysis project, a literary survey might involve reviewing existing research on topics related to web application logs, such as log file formats, data preprocessing, log analysis techniques, and so on.

Here are a few recent studies related to web application log analysis that you might consider including in your literary survey:

1] Anomaly Detection in Log Files Using Machine Learning Techniques by Lakshmi Geethanjali Mandagondi: The main purpose of this project is to develop a program using machine learning techniques that can read the test log file and learn what a "normal" log should look like and from that it should be able to determine that a log is “abnormal”. But it should not only be the case for fault finding. The detected anomalies can be of intrigue regardless of whether all tests in the suite have passed so that it will assist with telling us what causes flaws. The test performs the configuration of hardware and checks if the configuration has had the expected result. However, sometimes tests fail because of some unexpected changes occurred. It can also be because of a previous test that has not cleaned up properly.

2] Anomaly detection with Machine learning by Hanna Blomquist and Johanna Möller: The overall purpose of this study was to find a way to identify incorrect data in Sida’s statistics about their contributions. A contribution is the financial support given by Sida to a project. The goal was to build an algorithm that determines if a contribution has a risk to be inaccurate coded, based on supervised classification methods within the area of Machine Learning. A thorough data analysis process was done in order to train a model to find hidden patterns in the data. Descriptive features containing important information about the contributions were successfully selected and used for this task. These included keywords that were retrieved from descriptions of the contributions. Two Machine learning methods, Adaboost and Support Vector Machines, were tested for ten classification models. Each model got evaluated depending on their accuracy of predicting the target variable into its correct class. A misclassified component was more likely to be incorrectly coded and was also seen as an anomaly. The Adaboost method performed better and more steadily on the majority of the models. Six classification models built with the Adaboost method were combined to one final ensemble classifier. This classifier was verified with new unseen data and an anomaly score was calculated for each component. The higher the score, the higher the risk of being anomalous. The result was a ranked list, where the most anomalous components were prioritized for further investigation of staff at Sida.

3] Attack and Anomaly Detection in IoT Sites Using Machine Learning Techniques by Aziza Khalilahmed Shaikh and Prof Govind Negalur: This paper talks about a growing problem in the IoT space is the attack and anomaly detection in the infrastructure of the Internet of Things (IoT). Every domain is using IoT infrastructure more and more, and with that use comes a surge in risks and attacks against those infrastructures. Such attacks and anomalies that can lead to an IoT system failure include Denial of Service, Data Type Probing, Malicious Control, Malicious Operation, Scan, Spying, and Wrong

Setup. Logistic Regression (LR), Decision Tree (DT) and Random Forest (RF) are the machine learning (ML) methods that have been employed in this. Accuracy, precision, recall, f1 score, and area under the receiver operating characteristic curve are the evaluation measures used in performance comparison. For Decision Tree and Random Forest, the system received test accuracy results of 99.4 %. Despite the same accuracy of these algorithms, other criteria show that Random Forest performs significantly better.

4] Implementation of Anomaly Detection Technique Using Machine Learning Algorithms by K. Hanumantha Rao, G. Srinivas, Ankam Damodhar and M. Vikas Krishna: In this paper, we present "machine learning" a method to cascade K-means clustering and the Id3 decision tree learning methods to classifying anomalous and normal activities in a computer network. The K-means clustering method first partitions the training instances into two clusters using Euclidean distance similarity. On each cluster, representing a density region of normal or anomaly instances, we build an ID3 decision tree. The decision tree on each cluster refines the decision boundaries by learning the subgroups within the cluster. Our work studies the best algorithm by using classifying anomalous and normal activities in a computer network with supervised & unsupervised algorithms that have not been used before. We analyze the algorithm that have the best efficiency or the best learning and describes the proposed system of K-means & ID3 Decision Tree.

These studies are just a few examples of the research that has been conducted in the area of web application log analysis. By conducting a comprehensive literary survey, you can identify the key findings and limitations of previous studies, and use this information to inform your own research project.

Chapter-3

ARCHITECTURE AND FLOW CHART

3.1 Architecture of web application log analysis:

The architecture of a web application log analysis project typically involves several components, which may include data collection, storage, preprocessing, analysis, and visualization. Here is a general overview of the architecture of a web application log analysis project:

Data Collection: In this step, web application logs are collected from the web application and stored in a database or file system. The logs can be collected using various methods, such as using log files, web application access logs, or other tools.

Data Storage: The collected web application logs are stored in a database or file system. The database can be a relational database, NoSQL database, or a distributed file system like Hadoop Distributed File System (HDFS).

Data Preprocessing: In this step, the collected data is preprocessed to clean and filter the data. This step involves removing irrelevant or duplicate entries, and converting the data into a standardized format. Data preprocessing techniques can be applied to improve the quality and consistency of the data.

Data Analysis: Data analysis techniques are applied to the preprocessed data to extract insights and patterns. This step involves applying statistical or machine learning algorithms to the data. For instance, log data can be used to identify popular pages, determine user behavior patterns, and detect security breaches.

Data Visualization: Data visualization tools are used to present the insights and patterns in a clear and understandable format. This step involves creating charts, graphs, or other visualizations to display the results of the analysis. Data visualization can help in identifying trends and patterns that might be difficult to detect using other methods.

Deployment: The final step is to deploy the web application log analysis system. The system can be deployed in a cloud-based environment or an on-premise infrastructure, depending on the needs and resources of the organization. The system can be designed to receive data from multiple web applications and can be scalable to handle increasing data volumes.

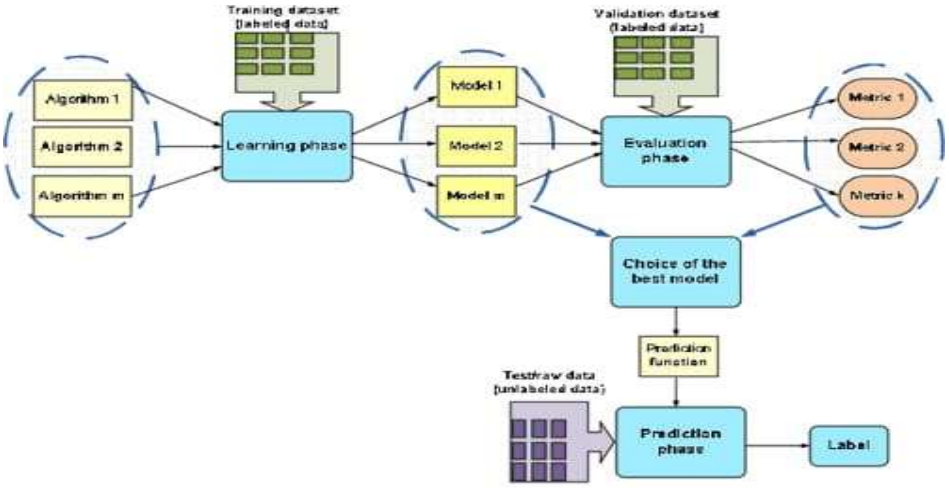


Figure 1. The structure of the supervised machine learning, evaluation and prediction procedures

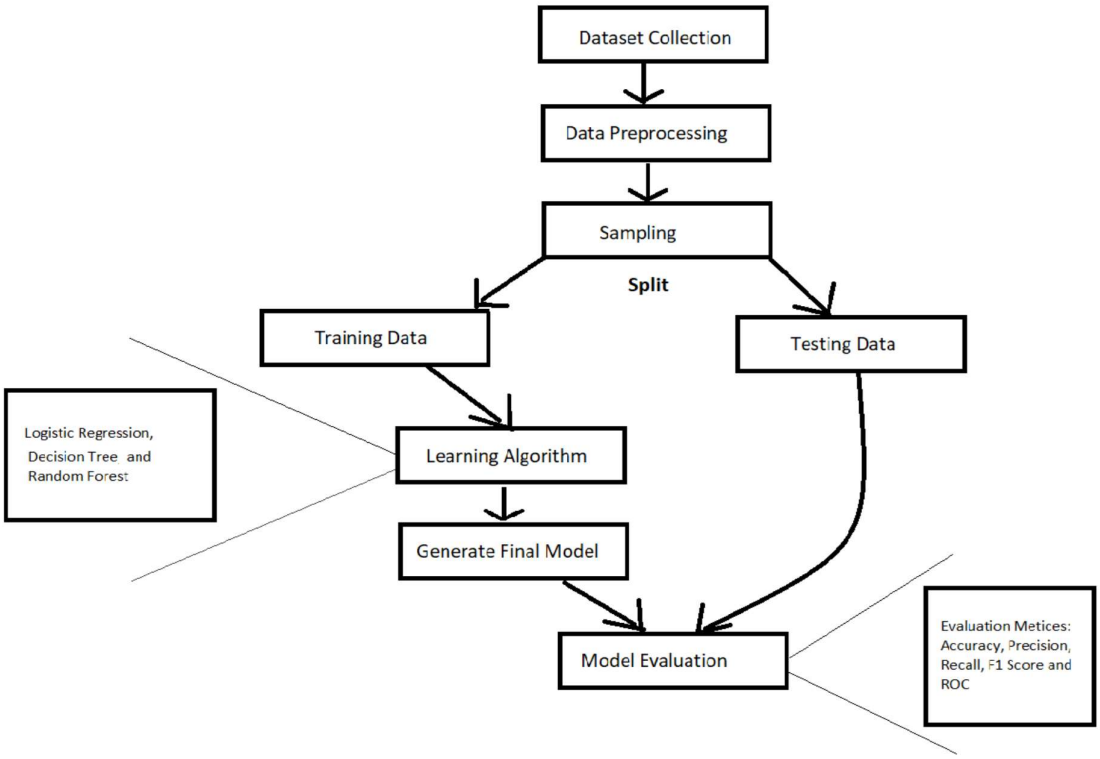


Fig. 1 Overall framework for attack and anomaly detection in web application

Fig. 2 Process of anomaly detection.

Chapter-4

WORKING

One of the most popular and effective enterprise use-cases which leverage analytics today is log analytics. Nearly every organization today has multiple systems and infrastructure running day in and day out. To effectively keep their business running, these organizations need to know if their infrastructure is performing to its maximum potential. Finding out involves analyzing system and application logs and maybe even applying predictive analytics on log data. The amount of log data involved is typically massive, depending on the type of organizational infrastructure involved and applications running on it.

Gone are the days when we were limited to analyzing a data sample on a single machine due to compute constraints. Powered by big data, better and distributed computing, and frameworks like Apache Spark for big data processing and open-source analytics, we can perform scalable log analytics on potentially billions of log messages daily. The intent of this case study-oriented tutorial is to take a hands-on approach showcasing how we can leverage Spark to perform log analytics at scale on semi-structured log data.

Main Objective: Web Application Log Analysis.

As we mentioned before, Apache Spark is an excellent and ideal open-source framework for wrangling, analyzing and modeling structured and unstructured data. Our main objective is one of the most popular use-cases in the industry log analytics. Server logs are a common enterprise data source and often contain a gold mine of actionable insights and information. Log data comes from many sources in these conditions, such as the web, client and compute servers, applications, user-generated content, and flat files. These logs can be used for monitoring servers, improving business and customer intelligence, building recommendation systems, fraud detection, and much more.
Spark allows you to cheaply dump and store your logs into files on disk, while still providing rich APIs to perform data analysis at scale.

Setting Up Dependencies:

Spark: Apache Spark is an open-source distributed computing system designed for big data processing and analytics. It provides a unified platform for various data processing tasks, including batch processing, interactive queries, real-time streaming, machine learning, and graph processing. Spark is designed to handle large-scale data processing and can efficiently process data in parallel across a cluster of machines.

Environment:

The specifications of the system are mentioned in the table. The programming languages used in the research was Python. Various python and machine learning libraries were used to which were needed for the implementation of the algorithms. The reason behind choosing python is that it is one of the most accessible programming languages available as it has more simplified syntax which gives more emphasis on natural language. Python for developing complex scientific and numeric applications. Python is designed with features to facilitate data analysis and visualization. The data visualization libraries and APIs provided by Python help you to

visualize and present data in a more appealing and effective way.

Operating System	Windows 11
Processor	Intel core i5/Ryzen
RAM	16/8 Gb
Graphics	Nvidia Gtx
Core	6/12

Tb 1. System Configuration

Libraries Used

Scikit: Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Pandas: Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

NumPy: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.

Seaborn: Seaborn provides an API on top of Matplotlib that offers sane choices for plot style and color defaults, defines simple high-level functions for common statistical plot types, and integrates with the functionality provided by Pandas.

Re: is used for regular expression operations. It will likely be used to extract patterns from text data.

Glob: is used to search for files using wildcard patterns. In this case, it seems like the code is expected to process multiple files.

PyCaret: PyCaret (Python Caret) is an open-source, low-code machine learning library designed to streamline the end-to-end machine learning process. It provides a simplified interface for tasks such as data preprocessing, feature engineering, model selection, hyperparameter tuning, and model deployment. PyCaret aims to make it easier for both beginners and experienced data scientists to quickly build and evaluate machine learning models without writing extensive code.

PyOD: PyOD (Python Outlier Detection) is an open-source library for detecting outliers in multivariate data. Outliers are data points that deviate significantly from the rest of the dataset and can distort the results of statistical analyses and machine learning models. PyOD provides various algorithms for outlier detection, making it useful in various domains where identifying anomalous data is crucial, such as fraud detection, network security, and quality control.

Imbalanced-learn: Imbalanced-learn is an open-source library in Python that provides various techniques for addressing imbalanced datasets. It offers a collection of under-sampling, over-sampling, and combination methods to rebalance the class distribution. Some key techniques provided by imbalanced-learn include Random Under-sampling, Random Over-sampling, Synthetic Minority Over-sampling Technique (SMOTE), and more.

Project Workflow Pipeline

Importing Libraries

```
import pandas as pd
import matplotlib.pyplot as plt
#plt.style.use('fivethirtyeight')
from pyspark.context import SparkContext
from pyspark.sql.context import SQLContext
from pyspark.sql.session import SparkSession
sc=SparkContext()
sqlContext=SQLContext(sc)
spark=SparkSession(sc)
from pyspark.sql.functions import col,split, regexp_extract,to_timestamp,year, month, hour, minute, second,date_format
import re #Regular expressions for extracting the features for our project
import glob #The glob function in Python allows you to retrieve a list of file paths that match a specified pattern.
```

Creating Spark Session and Reading the data (Log Files):

Reading the Log Files

```
#sc.stop()
```

```
log_folder = "nginx1"  # "log" folder contains all the log data thus giving its path
#Glob is commonly used to search for files and directories in a directory structure based on specific criteria.
raw_data_files=glob.glob("nginx1/*.gz") #(glob Stands for 'Global')
raw_data_files
```

```
['nginx1\\access.log-20230213.gz',
 'nginx1\\access.log-20230214.gz',
 'nginx1\\access.log-20230215.gz',
 'nginx1\\access.log-20230216.gz',
 'nginx1\\access.log-20230217.gz',
 'nginx1\\access.log-20230218.gz',
 'nginx1\\access.log-20230219.gz',
 'nginx1\\access.log-20230220.gz',
 'nginx1\\access.log-20230221.gz',
 'nginx1\\access.log-20230222.gz',
 'nginx1\\error.log-20230213.gz',
 'nginx1\\error.log-20230214.gz',
 'nginx1\\error.log-20230215.gz',
```

Loading and Viewing the Log Dataset

Given that our data is stored in the following path (in the form of flat files), let's load it into a spark sql DataFrame. We'll do this in steps. The following code loads our disk's log data file names. Now, we'll use sqlContext.read.text() or spark.read.text() to read the text file. This code produces a DataFrame with a single string column called value.

```
log_df1=spark.read.text(raw_data_files)
log_df1.printSchema()
```

```
root
 |-- value: string (nullable = true)
```

Sample log file:

```
log_df1.show(10,truncate=False)
```

```
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|value
|
+------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|143.244.50.172 - - [16/Feb/2023:03:28:45 +0530] "GET /config/getuser?index=0 HTTP/1.1" 400 248 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0"
|
|164.90.235.116 - - [16/Feb/2023:04:11:34 +0530] "GET / HTTP/1.1" 200 5952 "http://14.139.152.12/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36"
|
|66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /robots.txt HTTP/1.1" 404 146 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
|
|66.249.69.126 - - [16/Feb/2023:04:39:52 +0530] "GET /assets/img/favicon.png HTTP/1.1" 200 491 "-" "Googlebot-Image/1.0"
|
|185.221.219.172 - - [16/Feb/2023:05:06:47 +0530] "GET /.git/config HTTP/1.1" 404 548 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.2; Trident/5.0)"
|
|66.249.69.101 - - [16/Feb/2023:05:24:52 +0530] "GET /apim/devportal/site/public/images/logo.svg HTTP/1.1" 304 0 "-" "Googlebot-Image/1.0"
|
|52.167.144.90 - - [16/Feb/2023:06:26:16 +0530] "GET / HTTP/1.1" 200 5952 "-" "Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko; compatible; bingbot/2.0; +http://www.bing.com/bingbot.htm) Chrome/103.0.5060.134 Safari/537.36"
```

Data Wrangling:

In this section, we clean and parse our log dataset to extract structured attributes with meaningful information from each log message.

Log data understanding:
If you're familiar with web application logs, you'll recognize that the data displayed above is in Common Log Format:

The log file entries produced will look something like this:

143.244.50.172 - - [16/Feb/2023:03:28:45 +0530] "GET /config/getuser?index=0 HTTP/1.1" 400 248 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0"

- 143.244.50.172: This is the IP address of the client that made the request to the server. It represents the source of the HTTP request.

  - -: These hyphens represent the identity of the client and the user, which are usually not available in the log. In some cases, you might see these fields populated with more information, but in this log, they're empty.

- [16/Feb/2023:03:28:45 +0530]: This is the timestamp of the request. It provides information about when the request was made. The format is day/month/year:hour:minute:second +timezone.

- "GET /config/getuser?index=0 HTTP/1.1": This part represents the HTTP request line. It consists of three main components:

- GET: The HTTP method used in the request. In this case, it's a GET request, which is used to retrieve information from the server.

- /config/getuser?index=0: The path of the requested resource on the server. This indicates the specific URL that the client is trying to access. In this case, it's requesting the resource located at /config/getuser with the query parameter index set to 0.

- HTTP/1.1: The version of the HTTP protocol being used for the request. In this case, it's HTTP/1.1.

- 400: This is the HTTP status code returned by the server in response to the request. The status code 400 indicates a "Bad Request" error. It means that the server couldn't understand or process the client's request due to a malformed syntax or other client-side error.

- 248: This is the size of the response in bytes. It indicates the amount of data sent from the server to the client in response to the request.

- "-": This hyphen represents the referer header. It indicates the URL of the page that referred the client to the current requested resource. In this case, the hyphen indicates that no referer information was provided.

- "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0": This is the user-agent string sent by the client's web browser. It provides information about the client's browser, operating system, and version. In this case, the user-agent string indicates that the client is using Firefox version 76.0 on a Linux system.

We now need techniques to parse, match, and extract these attributes from the log data.

Data Parsing and Extraction with Regular Expressions and putting together:

Next, we must parse our semi-structured log data into individual columns. We'll use the special built-in regexp_extract() function to do the parsing. This function matches a column against a regular expression with one or more capture groups, and allows you to extract one of the matched groups. We'll use one regular expression for each field we wish to extract.
Let's now leverage all the regular expression patterns we previously built and use the regex_extract(…) method to build our DataFrame with all of the log attributes neatly extracted in their own separate columns.

Using RE for Extracting the features of log data

```
ip_pattern = r'(^\S+\.[\S+\.]+\S+)\s'
timestamp_pattern = r'\[(\d{2}/\w+/\d{4}:\d{2}:\d{2}:\d{2} \+\d{4})\]'
request_pattern = r'\"(\S+)\s(\S+)\s*(\S*)\"'
status_pattern = r'(\d{3})'
size_pattern = r'\s(\d+) "'
user_agent_pattern = r'"([^"]+)"'
protocol_pattern = r'HTTP/([\d.]+)'
browser_pattern = r'"([^"]+)"$'
```

```
log_df1 = log_df1.select(
                        regexp_extract(col("value"), ip_pattern, 1).alias("Ip_address"),
                        regexp_extract(col("value"), timestamp_pattern, 1).alias("timestamp"),
                        regexp_extract(col("value"), request_pattern, 1).alias("HTTP request line"),
                        regexp_extract(col("value"), request_pattern, 2).alias("user_agent"),
                        regexp_extract(col("value"), request_pattern, 3).alias("protocol"),
                        regexp_extract(col("value"), status_pattern, 1).alias("HTTP status code"),
                        regexp_extract(col("value"), size_pattern, 1).alias("Size of the response in bytes"),
                       )
log_df1 = log_df1.withColumn("timestamp", to_timestamp(col("timestamp"), "dd/MMM/yyyy:HH:mm:ss Z"))
log_df1 = log_df1.withColumn("response_time", (col("timestamp").cast("long") - col("timestamp").cast("long")).cast("int"))

log_df1 = log_df1.withColumn("year", col("timestamp").substr(1, 4))
log_df1 = log_df1.withColumn("month", date_format(col("timestamp"), "MMM"))
log_df1 = log_df1.withColumn("date", col("timestamp").substr(9, 2))
log_df1 = log_df1.withColumn("hour", col("timestamp").substr(12, 2))
log_df1 = log_df1.withColumn("minute", col("timestamp").substr(15, 2))
log_df1 = log_df1.withColumn("seconds", col("timestamp").substr(18, 2))
```

Converting the Spark DataFrame to Pandas DataFrame:

```
log_pandas_df = log_df1.toPandas() #Converting it into Pandas dataframe for better visualization
log_pandas_df
```

	Ip_address	HTTP request line	user_agent	protocol	HTTP status code	Size of the response in bytes	response_time	year	month	date	hour	minute	seconds
0	143.244.50.172	GET	/config/getuser?index=0	HTTP/1.1	143	248	0.0	2023	Feb	16	03	28	45
1	164.90.235.116	GET	/	HTTP/1.1	164	5952	0.0	2023	Feb	16	04	11	34
2	66.249.69.126	GET	/robots.txt	HTTP/1.1	249	146	0.0	2023	Feb	16	04	39	52
3	66.249.69.126	GET	/assets/img/favicon.png	HTTP/1.1	249	491	0.0	2023	Feb	16	04	39	52
4	185.221.219.172	GET	/.git/config	HTTP/1.1	185	548	0.0	2023	Feb	16	05	06	47

Performing Exploratory Data Analysis on Dataframe:

Encoding Feature:

Encoding IPaddress

```
# Function to convert IP address to numerical value
import ipaddress
def convert_ip_to_numeric(ip_str):
    ip = ipaddress.ip_address(ip_str)
    return int(ip)


# Apply the conversion function to the entire column
log_pandas_df['Ip_address'] = log_pandas_df['Ip_address'].apply(convert_ip_to_numeric)
log_pandas_df
```

	Ip_address	HTTP request line	user_agent	protocol	HTTP status code	Size of the response in bytes	response_time	year	month	date	hour	minute	seconds	No of Requests
0	2415145644	GET	/config/getuser?index=0	HTTP/1.1	143	248	0.0	2023	Feb	16	03	28	45	62
1	2757421940	GET	/	HTTP/1.1	164	5952	0.0	2023	Feb	16	04	11	34	5
2	1123632510	GET	/robots.txt	HTTP/1.1	249	146	0.0	2023	Feb	16	04	39	52	20
3	1123632510	GET	/assets/img/favicon.png	HTTP/1.1	249	491	0.0	2023	Feb	16	04	39	52	20

The ipaddress library in Python provides classes for working with IP addresses and networks. It's a part of the Python standard library, which means you don't need to install any additional packages to use it. The library simplifies tasks related to IP address manipulation, validation, and representation.
Internally, the `ipaddress` library converts IP addresses to integers by performing some mathematical operations on the components of the IP address. Let's break down the process step by step:

1. IP Address Components:
An IP address consists of four segments, each representing an octet (8 bits) of the address. For example, in the IPv4 address "192.168.1.1", the octets are 192, 168, 1, and 1.

2. Binary Representation:
Each octet can be represented as an 8-bit binary number. For instance:
- 192 is 11000000 in binary.
- 168 is 10101000 in binary.
- 1 is 00000001 in binary.

3. Concatenation of Binary Segments:
The binary representations of the octets are concatenated to form a 32-bit binary string. In the case of "192.168.1.1", this results in: 11000000 10101000 00000001 00000001

4. Binary-to-Integer Conversion:
The concatenated binary string is then converted to an integer using base-2 (binary) to base-10 (decimal) conversion. This integer value becomes the numeric representation of the IP address.
For the IP address "192.168.1.1", the binary-to-integer conversion would yield:
- Binary: 11000000 10101000 00000001 00000001
- Decimal: 3221225985
So, internally, the library converts each octet of the IP address to its binary representation and then concatenates these binary segments into a larger binary string. This binary string is then

converted to an integer using base-2 to base-10 conversion. The resulting integer is the numeric representation of the IP address.

Encoding Other Categorical Column

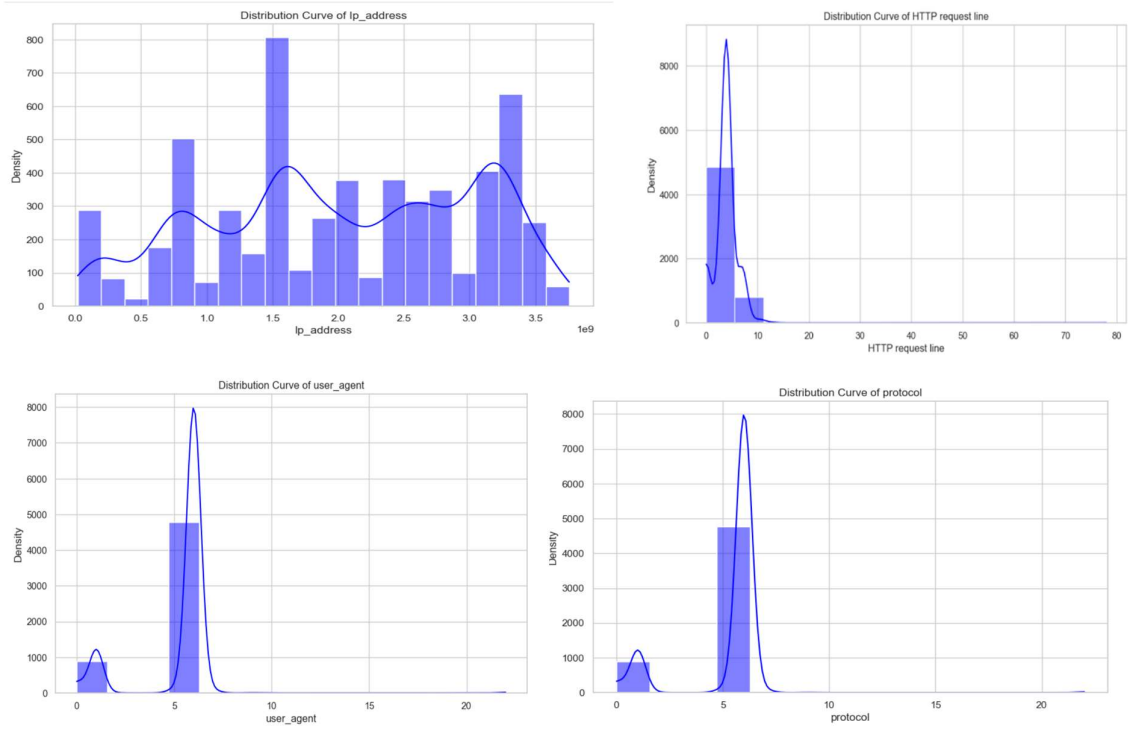
```
from sklearn.preprocessing import LabelEncoder
# Apply Label Encoding to the required column
label_encoder=LabelEncoder()
log_pandas_df['HTTP request line'] = label_encoder.fit_transform(log_pandas_df['HTTP request line'])
log_pandas_df['protocol'] = label_encoder.fit_transform(log_pandas_df['protocol'])
log_pandas_df['user_agent'] = label_encoder.fit_transform(log_pandas_df['protocol'])
log_pandas_df['month'] = label_encoder.fit_transform(log_pandas_df['month'])

log_pandas_df
```

	Ip_address	HTTP request line	user_agent	protocol	HTTP status code	Size of the response in bytes	response_time	year	month	date	hour	minute	seconds	No of Requests
0	2415145644	4	6	6	143	248	0.0	2023	0	16	03	28	45	62
1	2757421940	4	6	6	164	5952	0.0	2023	0	16	04	11	34	5
2	1123632510	4	6	6	249	146	0.0	2023	0	16	04	39	52	20
3	1123632510	4	6	6	249	491	0.0	2023	0	16	04	39	52	20

A LabelEncoder is a utility class in Python, typically found in machine learning libraries such as scikit-learn, that is used for converting categorical data (text or labels) into numerical values. It's particularly useful when dealing with machine learning algorithms that require numerical input, as many algorithms work best with numeric data.

Distribution of each attribute

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

# Set the style for the plots
sns.set(style="whitegrid")

# Select only numerical columns from your DataFrame
numerical_columns = log_pandas_df.select_dtypes(include=[np.number])

# Iterate through numerical columns and create distribution plots
for column in numerical_columns.columns:
    plt.figure(figsize=(10, 6))
    sns.histplot(log_pandas_df[column], kde=True, color='blue')

    # Add labels and title
    plt.title(f"Distribution Curve of {column}")
    plt.xlabel(column)
    plt.ylabel("Density")

    plt.show()
```



Above graph shows the distribution of the feature. It helps us to understand the spread of each feature. We can observe apart from the Ipadress other features are shows the skewness means they are away from the mean value. So, for the further operation we need to scale them so they will show the result like normally distributed data.

Part-1 Using DBSCAN for labeling data and Finding Anomaly and Evaluating Algorithms

DBSCAN Clusturing for finding Anomaly and Label data

```
import numpy as np
from sklearn.metrics import silhouette_score
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
normalized_data = scaler.fit_transform(log_pandas_df)

best_eps = None
best_min_samples = None
best_score = -1

for eps in np.arange(0.5,1.5):
    for min_samples in range(2, 10):
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels = dbscan.fit_predict(normalized_data)

        if len(set(labels)) > 1:
            score = silhouette_score(normalized_data, labels)
            if score > best_score:
                best_score = score
                best_eps = eps
                best_min_samples = min_samples

print(f"Best Silhouette Score: {best_score:.3f}")
print(f"Best eps: {best_eps}")
print(f"Best min_samples: {best_min_samples}")
```

```
Best Silhouette Score: 0.133
Best eps: 0.5
Best min_samples: 2
```

DBSCAN stands for "Density-Based Spatial Clustering of Applications with Noise." It's a popular unsupervised machine learning algorithm used for clustering spatial data points based on their density distribution. DBSCAN is particularly effective at identifying clusters of varying shapes and handling noise.
Density-Based Clustering:
DBSCAN operates by identifying clusters in regions of high data point density. It defines clusters as areas where there are many data points close to each other, separated by areas where there are fewer data points.

Core Points, Border Points, and Noise:
DBSCAN categorizes data points into three main categories:

Core Points: A data point is a core point if it has at least a specified minimum number of neighboring data points within a certain distance (epsilon). Core points are the heart of clusters.
Border Points: A data point is a border point if it's within the epsilon distance of a core point but doesn't have enough neighbors to be considered a core point itself.
Noise Points: Data points that are neither core nor border points are considered noise points.
Cluster Formation:
The algorithm starts with an arbitrary data point and identifies its epsilon neighborhood. If this neighborhood contains enough points (core points), a cluster is formed. The algorithm then iteratively expands the cluster by adding more core and border points until no more points can be added. This process repeats for all core points and their connected components until all data points are assigned to clusters or marked as noise.

DBSCAN requires two main parameters:

Epsilon (ε): The maximum distance between two data points for one to be considered as in the neighborhood of the other.
Minimum Points (MinPts): The minimum number of data points required to form a core point.

```
Number of clusters: 661
Distribution of clusters:
-1      1876
 323     547
 484     127
 19       82
 105      75
        ...
 263       2
 262       2
 261       2
 260       2
 330       2
Name: Cluster, Length: 662, dtype: int64
```

Label the data:

```
selected_columns = ['Ip_address', 'HTTP request line', 'user_agent', 'protocol', 'HTTP status code',
                    'Size of the response in bytes', 'response_time', 'year', 'month', 'date', 'hour',
                    'minute', 'seconds', 'No of Requests']

# Create a subset DataFrame with selected columns
data_subset = log_pandas_df[selected_columns]

# Normalize the data using StandardScaler
scaler = StandardScaler()
normalized_data = scaler.fit_transform(data_subset)

# Instantiate and fit DBSCAN model
dbscan = DBSCAN(eps=0.5, min_samples=2)
labels = dbscan.fit_predict(normalized_data)

# Add cluster labels to the original DataFrame
log_pandas_df['Cluster'] = labels

# Create 'IsAnomaly' column based on cluster labels
log_pandas_df['IsAnomaly'] = (log_pandas_df['Cluster'] == -1).astype(int)

# Add 'Anomaly' column based on 'IsAnomaly'
log_pandas_df['Anomaly'] = np.where(log_pandas_df['IsAnomaly'] == 1, 'Anomaly', 'Not Anomaly')

# Print the number of anomalies
num_anomalies = log_pandas_df['IsAnomaly'].sum()
print(f"Number of anomalies: {num_anomalies}")

# Display the DataFrame with the 'Anomaly' column
log_pandas_df
```

```
Number of anomalies: 1876
```

	Ip_address	HTTP request line	user_agent	protocol	HTTP status code	Size of the response in bytes	response_time	year	month	date	hour	minute	seconds	No of Requests	Cluster	IsAnomaly	Anomaly
0	2415145644	4	6	6	143	248	0.0	2023	0	16	03	28	45	62	0	0	Not Anomaly
1	2757421940	4	6	6	164	5952	0.0	2023	0	16	04	11	34	5	1	0	Not Anomaly
2	1123632510	4	6	6	249	146	0.0	2023	0	16	04	39	52	20	2	0	Not Anomaly
3	1123632510	4	6	6	249	491	0.0	2023	0	16	04	39	52	20	2	0	Not Anomaly
4	3118324652	4	6	6	185	548	0.0	2023	0	16	05	06	47	1	3	0	Not Anomaly

Save dataframe as CSV

```
log_pandas_df.to_csv('Anomaly1.csv', index=True)
```

Now Evaluating the data using Different Algorithm:

```
# Load the data
log_pandas_df = pd.read_csv('Anomaly1.csv')

# Select the columns you want to use for anomaly detection
selected_columns = ['Ip_address', 'HTTP request line', 'user_agent', 'protocol', 'HTTP status code',
                    'Size of the response in bytes', 'response_time', 'year', 'month', 'date', 'hour',
                    'minute', 'seconds', 'No of Requests']
data_subset = log_pandas_df[selected_columns]

# Prepare the target labels
y = log_pandas_df['IsAnomaly']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data_subset, y, test_size=0.2, random_state=42)

# Apply standard scaling to the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply SMOTE to the training data
smote = SMOTE(random_state=42)
X_train_resampled1, y_train_resampled1 = smote.fit_resample(X_train_scaled, y_train)

# Models to evaluate
algorithms = [
    ('Random Forest', RandomForestClassifier(random_state=42)),
    ('XGBoost', XGBClassifier(random_state=42)),
    ('Support Vector Machine', SVC(random_state=42)),
    ('Logistic Regression', LogisticRegression(random_state=42))

]

for name, model in algorithms:
    print(f"Fitting {name}...")
    # Fit the model to the resampled training data
    model.fit(X_train_resampled1, y_train_resampled1)

    # Predict anomaly labels on the scaled test data
    predicted_labels = model.predict(X_test_scaled)

    # Generate a classification report
    classification_rep = classification_report(y_test, predicted_labels)

    # confusion matrix
    conf_matrix = confusion_matrix(y_test, predicted_labels)


    mcc = matthews_corrcoef(y_test, predicted_labels)

    print(f"Classification Report for {name}:\n{classification_rep}")
    print(f"Confusion Matrix for {name}:\n{conf_matrix}")
    print(f"MCC Score for {name}: {mcc}\n")
```

SMOTE (Synthetic Minority Over-sampling Technique):
SMOTE is a technique used to address class imbalance in a dataset. It generates synthetic samples for the minority class by interpolating between existing samples. This helps balance class distribution and improves the performance of machine learning algorithms on imbalanced data.

Random Forest:
Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their predictions to improve accuracy and control overfitting. It's effective for classification and regression tasks, handling various data types and capturing complex relationships in data.

SVM (Support Vector Machine):
SVM is a powerful supervised learning algorithm used for classification and regression. It works by finding a hyperplane that best separates data into different classes. SVM aims to maximize the margin between classes and can handle non-linear relationships through kernel functions.

XGBoost (Extreme Gradient Boosting):
XGBoost is an optimized gradient boosting algorithm known for its speed and performance. It sequentially builds a series of weak learners (typically decision trees) to correct errors made by previous models. XGBoost handles missing data, supports regularization, and has become a popular choice for structured data problems.

Logistic Regression:
Despite its name, logistic regression is used for binary classification. It models the probability that an input belongs to a certain class using the logistic function. It's a simple yet interpretable algorithm that's widely used in various fields due to its efficiency and effectiveness.

Evaluation Report:

```
Fitting Random Forest...
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.94      0.92      0.93       780
           1       0.83      0.88      0.85       368

    accuracy                           0.90      1148
   macro avg       0.88      0.90      0.89      1148
weighted avg       0.90      0.90      0.90      1148

Confusion Matrix for Random Forest:
[[714  66]
 [ 46 322]]
MCC Score for Random Forest: 0.7798068215867826

Fitting XGBoost...
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       780
           1       0.85      0.85      0.85       368

    accuracy                           0.90      1148
   macro avg       0.89      0.89      0.89      1148
weighted avg       0.90      0.90      0.90      1148

Confusion Matrix for XGBoost:
[[723  57]
 [ 54 314]]
MCC Score for XGBoost: 0.7785217608397688
```

Evaluation parameters:
Accuracy
Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. If we have high accuracy for a particular model then the model is best.

Accuracy = TP + TN/TP + FP + FN + TN

Precision
Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to the low false positive rate.

Precision = TP/TP + FP

Recall
Recall is the ratio of correctly predicted positive observations to the all observations in actual class. It has values between 0 and 1. Higher the recall, better performing the model is.

Recall = TP/TP + FN

F1 Score
F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. It ranges from 0 and 1, higher the F1 score better the model is performing.

F1Score = 2∗ (Recall ∗ Precision)/ (Recall + Precision)

Matthews Correlation Coefficient (MCC)
The Matthews Correlation Coefficient (MCC) is a metric used to evaluate the quality of binary classification models. It takes into account true positives, true negatives, false positives, and false negatives. MCC ranges from -1 to +1, where +1 indicates perfect prediction, 0 represents random prediction, and -1 indicates total disagreement between prediction and actual values.

MCC = (TP * TN - FP * FN) / sqrt ((TP + FP) * (TP + FN) * (TN + FP) * (TN + FN))

Both Random Forest and XGBoost achieve an accuracy of 90%, indicating a good overall performance.In terms of precision, Random Forest has slightly higher precision for class 0, while XGBoost has slightly higher precision for class 1. Recall values are quite similar between the two models for both classes. F1-scores are also similar for both classes in both models. The confusion matrices show how the models predicted true positives, false positives, true negatives, and false negatives for each class. The Matthews Correlation Coefficient (MCC) is a measure of the quality of the binary classification, with values closer to 1 indicating better performance. Both models have MCC values above 0.77, suggesting reasonable performance.
Overall, both Random Forest and XGBoost perform well on the task, with minor differences in precision and classification performance.

Hyperparameter Tuning for Better Result:

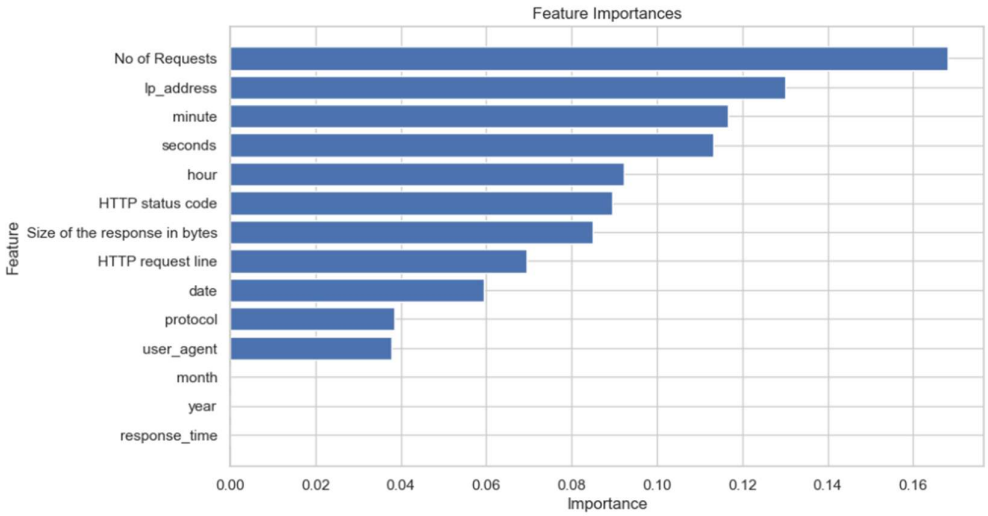
```
from sklearn.model_selection import RandomizedSearchCV
# Random Forest parameter grid
rf_param_dist = {
    'n_estimators': np.arange(100, 500),
    'max_depth':np.arange(10, 100),
    'min_samples_split': np.arange(2, 11),
    'min_samples_leaf': np.arange(1, 11),
    'bootstrap': [True, False]
}

# XGBoost parameter grid
xgb_param_dist = {
    'n_estimators': np.arange(100,500),
    'max_depth': np.arange(3, 11),
    'learning_rate': [0.01, 0.1, 0.2, 0.3],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0]
}

# Initialize RandomForestClassifier and XGBoostClassifier
rf_classifier = RandomForestClassifier(random_state=42)
xgb_classifier = XGBClassifier(random_state=42)

# Initialize RandomizedSearchCV for RandomForestClassifier
rf_random_search = RandomizedSearchCV(estimator=rf_classifier, param_distributions=rf_param_dist,
                                      n_iter=10, scoring='accuracy', cv=5, random_state=42, n_jobs=-1)
rf_random_search.fit(X_train_resampled1, y_train_resampled1)

# Initialize RandomizedSearchCV for XGBoostClassifier
xgb_random_search = RandomizedSearchCV(estimator=xgb_classifier, param_distributions=xgb_param_dist,
                                       n_iter=10, scoring='accuracy', cv=5, random_state=42, n_jobs=-1)
xgb_random_search.fit(X_train_resampled1, y_train_resampled1)

# Print best parameters for RandomForestClassifier
print("Best parameters for RandomForestClassifier:", rf_random_search.best_params_)

# Print best parameters for XGBoostClassifier
print("Best parameters for XGBoostClassifier:", xgb_random_search.best_params_)

# Evaluate the best RandomForestClassifier
best_rf_classifier = rf_random_search.best_estimator_
predicted_rf = best_rf_classifier.predict(X_test_scaled)
print("Classification Report for the Best RandomForestClassifier:")
print(classification_report(y_test, predicted_rf))

# Evaluate the best XGBoostClassifier
best_xgb_classifier = xgb_random_search.best_estimator_
predicted_xgb = best_xgb_classifier.predict(X_test_scaled)
print("Classification Report for the Best XGBoostClassifier:")
print(classification_report(y_test, predicted_xgb))
```

```
Best parameters for RandomForestClassifier: {'n_estimators': 443, 'min_samples_split': 3, 'min_samples_leaf': 1, 'max_depth': 57, 'bootstrap': True}
Best parameters for XGBoostClassifier: {'subsample': 0.8, 'n_estimators': 456, 'max_depth': 6, 'learning_rate': 0.3, 'colsample_bytree': 1.0}
Classification Report for the Best RandomForestClassifier:
              precision    recall  f1-score   support

           0       0.94      0.91      0.93       780
           1       0.83      0.89      0.85       368

    accuracy                           0.90      1148
   macro avg       0.88      0.90      0.89      1148
weighted avg       0.91      0.90      0.90      1148

Classification Report for the Best XGBoostClassifier:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       780
           1       0.92      0.86      0.88       368

    accuracy                           0.93      1148
   macro avg       0.92      0.91      0.92      1148
weighted avg       0.93      0.93      0.93      1148
```

Above we can observe clearly betterment in the result after hyperparameter tuning. Now we can see XGboost performing slightly better after the hyperparameter tuning.

Using RandomForest Looking for the Feature Importance

```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
# Load the data
log_pandas_df = pd.read_csv('Anomaly1.csv')

# Select the columns you want to use for anomaly detection
selected_columns = ['Ip_address', 'HTTP request line', 'user_agent', 'protocol', 'HTTP status code',
                    'Size of the response in bytes', 'response_time', 'year', 'month', 'date', 'hour',
                    'minute', 'seconds', 'No of Requests']

# Prepare the target labels
y = log_pandas_df['IsAnomaly']

# Create a subset of the data
X = log_pandas_df[selected_columns]

# Create a RandomForestClassifier model
model = RandomForestClassifier(random_state=42)

# Fit the model on the entire dataset
model.fit(X, y)

# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame to display feature importances
importance_df = pd.DataFrame({'Feature': selected_columns, 'Importance': feature_importances})
importance_df = importance_df.sort_values(by='Importance', ascending=True)

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(importance_df['Feature'], importance_df['Importance'])
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances')
plt.show()
```



Here we observe that in our data some features are more important some have less

importance which may be less useful so we can find it with feature importance function in randomforest.

```
Fitting Random Forest...
Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.95      0.92      0.94       780
           1       0.85      0.90      0.87       368

    accuracy                           0.92      1148
   macro avg       0.90      0.91      0.90      1148
weighted avg       0.92      0.92      0.92      1148

Confusion Matrix for Random Forest:
[[721  59]
 [ 38 330]]
MCC Score for Random Forest: 0.8096001347719144

Fitting XGBoost...
Classification Report for XGBoost:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       780
           1       0.87      0.86      0.86       368

    accuracy                           0.91      1148
   macro avg       0.90      0.90      0.90      1148
weighted avg       0.91      0.91      0.91      1148

Confusion Matrix for XGBoost:
[[731  49]
 [ 53 315]]
MCC Score for XGBoost: 0.7954668002357856
```

Here we see the improvement in result after dropping the less important feature.

Part-2 Using Pycaret (iForest) for labeling data and Finding Anomaly and Evaluating Algorithms

Detection of anomaly using pycaret module after label encoder

(A) Checking with Isolation Forest

```
from pycaret.anomaly import *
```

```
exp_ano = setup(data=log_pandas_df)
```

	Description	Value
0	Session id	8201
1	Original data shape	(5736, 14)
2	Transformed data shape	(5736, 419)
3	Numeric features	7
4	Categorical features	7
5	Preprocess	True
6	Imputation type	simple
7	Numeric imputation	mean
8	Categorical imputation	mode
9	Maximum one-hot encoding	-1
10	Encoding method	None
11	CPU Jobs	-1
12	Use GPU	False
13	Log Experiment	False
14	Experiment Name	anomaly-default-name
15	USI	1aa1

Assign label Based on anomaly score.

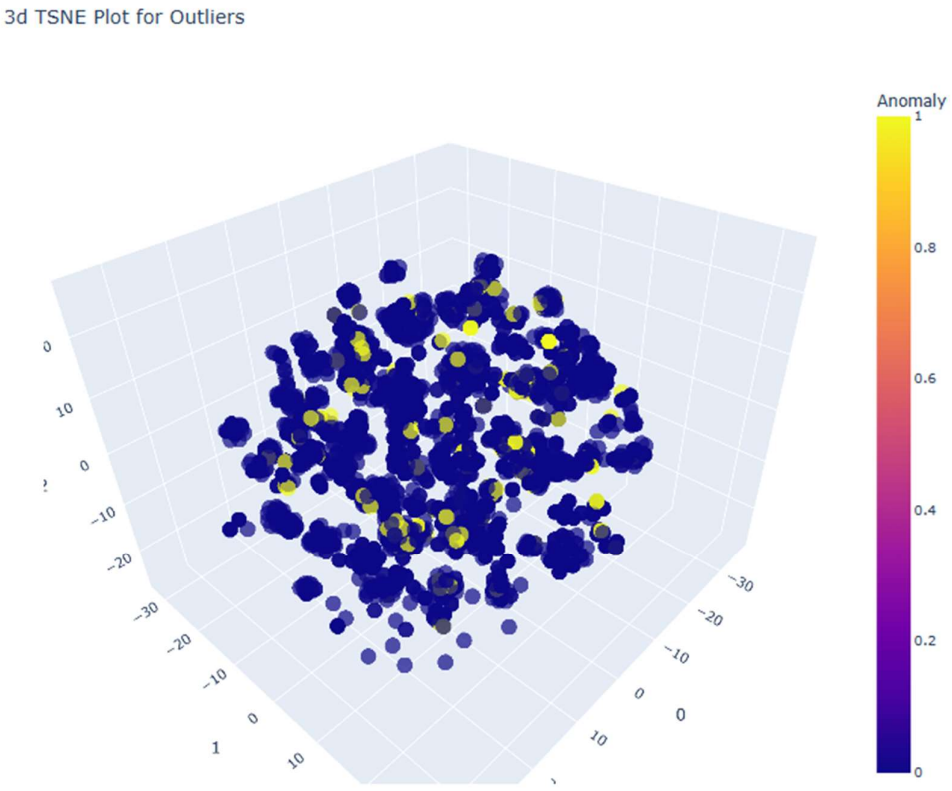
```
result=assign_model(iforest) # 0 = Normal and 1 = Anomaly
result.Anomaly.value_counts()
```

```
0    5449
1     287
Name: Anomaly, dtype: int64
```

```
result
```

	Ip_address	HTTP request line	user_agent	protocol	HTTP status code	Size of the response in bytes	response_time	year	month	date	hour	minute	seconds	No of Requests	Anomaly	Anomaly_Score
0	2415145644	4	6	6	143	248	0.0	2023	0	16	03	28	45	62	0	-0.002548
1	2757421940	4	6	6	164	5952	0.0	2023	0	16	04	11	34	5	0	-0.026349
2	1123632510	4	6	6	249	146	0.0	2023	0	16	04	39	52	20	0	-0.009036
3	1123632510	4	6	6	249	491	0.0	2023	0	16	04	39	52	20	0	-0.007608
4	3118324652	4	6	6	185	548	0.0	2023	0	16	05	06	47	1	0	-0.005265

Visualization of the clusters:



The Isolation Forest (iForest) algorithm is an unsupervised anomaly detection technique that's based on the concept of isolating anomalies within a dataset. It works by constructing a random forest of isolation trees. Each isolation tree isolates a subset of the data by randomly selecting features and then splitting the data points based on these features. Anomalies are expected to require fewer splits to be isolated than normal data points.

The anomaly score provided by the Isolation Forest reflects how easily a data point can be isolated. The basic idea is that anomalies are points that are isolated with fewer splits, indicating that they stand out from the majority of the data. Here's how the anomaly score is calculated:

Isolation Path Length: When a data point traverses down the tree, the number of splits (edges) it takes to isolate that point is recorded as the isolation path length. Anomalies tend to have

shorter path lengths because they're easier to separate from the rest of the data.

Average Path Length: For a given isolation tree, the average isolation path length of all data points within it is calculated. A shorter average path length suggests that the data points in that tree are anomalies.

Anomaly Score: The anomaly score for a data point is then calculated as a function of the average path length across all trees in the forest. A lower anomaly score indicates a higher likelihood of being an anomaly.

We also check with other algorithms like knn and svm both are giving same result as iForest so we continue with iForest.

Now Evaluating the data using Different Algorithm:

Applying CatBoostClassifier

```
from catboost import CatBoostClassifier,Pool
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import matthews_corrcoef
catboost_model_i = CatBoostClassifier(iterations=300, depth=6, learning_rate=0.01, loss_function='Logloss', random_state=24)
catboost_model_i.fit(X_train, y_train, cat_features=['HTTP request line','user_agent','protocol','month','HTTP status code','Size of the response in bytes','year','date','hour','minute','seconds'])
y_predcat = catboost_model_i.predict(X_test)
print("CatBoostClassifier \nAccuracy:",accuracy_score(y_test_cat,y_predcat)*100,"\nPrecision:",precision_score(y_test_cat,y_predcat),"\nRecall Score:",recall_score(y_test_cat,y_predcat),"\nF1 Score:",f1_score(y_test_cat,y_predcat),"\nConfusion Ma
print('''"CatBoostClassifier"''',classification_report(y_test_cat,y_predcat))
mcc_cat = matthews_corrcoef(y_test_cat, y_predcat)
f1_cat = f1_score(y_test_cat, y_predcat)
print("CatBoostClassifier")
print("MCC:", mcc_cat)
print("F1 Score:", f1_cat)
print()
```

Result:

```
CatBoostClassifier
Accuracy: 96.68989547038328
Precision: 0.84
Recall Score: 0.38181818181818183
F1 Score: 0.5250000000000001
Confusion Matrix:
 [[1089    4]
 [  34   21]]
"CatBoostClassifier"              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1093
           1       0.84      0.38      0.53        55

    accuracy                           0.97      1148
   macro avg       0.90      0.69      0.75      1148
weighted avg       0.96      0.97      0.96      1148

CatBoostClassifier
MCC: 0.5533570453318356
F1 Score: 0.5250000000000001
```

CatBoost (Categorical Boosting) is a gradient boosting algorithm designed to work effectively with categorical features in addition to numerical features. It is based on the gradient boosting framework and is specifically optimized for handling categorical data efficiently. Developed by Yandex, CatBoost has gained popularity for its ability to automatically handle categorical variables without the need for extensive preprocessing.

Gradient Boosting Framework: CatBoost is built upon the gradient boosting framework, which is an ensemble technique that combines the strengths of multiple weak learners (typically decision trees) to create a strong predictive model.

Categorical Feature Handling: One of CatBoost's main strengths is its ability to handle categorical features naturally. It uses a technique called "ordered boosting" to treat categorical features effectively. Ordered boosting sorts categorical features by their target statistics (such

as mean target value) for each category, which helps in making optimal splits during tree construction.

Target Encoding: CatBoost uses a combination of target encoding and one-hot encoding to represent categorical variables in a tree-friendly manner. This allows the algorithm to capture meaningful relationships between categorical features and the target variable.

Optimized Tree Building: During tree construction, CatBoost employs a novel algorithm to search for the optimal split points for both numerical and categorical features. This technique is particularly efficient for categorical features, as it minimizes the loss function while accounting for the specific characteristics of categorical data.

Regularization: CatBoost introduces a novel regularization technique called "ordered boosting" to prevent overfitting. This technique penalizes complex models and helps the algorithm generalize well to new data.

In summary, CatBoost is a powerful gradient boosting algorithm that stands out for its built-in handling of categorical variables and its efficient optimization techniques. It's designed to provide high-quality predictive models with minimal preprocessing effort, making it an attractive choice for data scientists and machine learning practitioners working with mixed datasets containing both categorical and numerical features.

Evaluting the Classification Report

```
from sklearn.metrics import classification_report
print('''"RandomForestClassifier"''',classification_report(y_test,y_pred))
print("MCC:", matthews_corrcoef(y_test, y_pred))
print('''"CatBoostClassifier"''',classification_report(y_test_cat,y_predcat))
print("MCC:", matthews_corrcoef(y_test_cat, y_predcat))
print('''"GradientBoostingClassifier"''',classification_report(y_test,y_pred_gb))
print("MCC:", matthews_corrcoef(y_test, y_pred_gb))
print('''"SVM"''',classification_report(y_test,y_predsvm))
print("MCC:", matthews_corrcoef(y_test, y_predsvm))
```

Result:

```
"RandomForestClassifier"              precision    recall  f1-score   support

           0       0.98      0.86      0.92      1102
           1       0.16      0.61      0.25        46

    accuracy                           0.85      1148
   macro avg       0.57      0.74      0.59      1148
weighted avg       0.95      0.85      0.89      1148

MCC: 0.25714700484200037
"CatBoostClassifier"              precision    recall  f1-score   support

           0       0.97      1.00      0.98      1093
           1       0.86      0.35      0.49        55

    accuracy                           0.97      1148
   macro avg       0.92      0.67      0.74      1148
weighted avg       0.96      0.97      0.96      1148

MCC: 0.533872341058555
"GradientBoostingClassifier"                  precision    recall  f1-score   support

           0       0.99      0.92      0.95      1102
           1       0.27      0.74      0.40        46

    accuracy                           0.91      1148
   macro avg       0.63      0.83      0.67      1148
weighted avg       0.96      0.91      0.93      1148

MCC: 0.4133775431881847
"SVM"               precision    recall  f1-score   support

           0       0.98      0.73      0.84      1102
           1       0.08      0.57      0.14        46

    accuracy                           0.73      1148
   macro avg       0.53      0.65      0.49      1148
weighted avg       0.94      0.73      0.81      1148

MCC: 0.13105625687602246
```

Performance Evaluation of Different Machine Learning Models for Classification:

The Matthews Correlation Coefficient (MCC) is a metric that takes into account true positive, true negative, false positive, and false negative values. It measures the quality of binary (two-class) classifications, where a higher MCC score indicates better performance.

The F1 Score is the harmonic mean of precision and recall. It's also used to evaluate binary classification models, with values ranging from 0 to 1. A higher F1 Score implies better precision and recall balance.

(A)RandomForestClassifier:

MCC (Matthews Correlation Coefficient): 0.3998
F1 Score: 0.4211

The MCC value of 0.3998 indicates a moderate level of correlation between predictions and actual values.
The F1 Score of 0.4211 indicates that the model achieves a balance between precision and recall. It's a reasonable F1 score.

(B)CatBoostClassifier:

MCC: 0.49
F1 Score: 0.45

CatBoost is a gradient boosting algorithm that works well with categorical features. The MCC and F1 Score interpretations are the same as described above. A higher MCC and F1 Score for CatBoost compared to RandomForestClassifier might indicate that CatBoost has better predictive performance on the given dataset.
(C)GradientBoostingClassifier:

MCC: 0.3307
F1 Score: 0.3211

The MCC value of 0.3307 indicates a moderate level of correlation between predictions and actual values.
The F1 Score of 0.3211 suggests that the model's balance between precision and recall is not as strong as desired.

(D)Support Vector Machine (SVM):

MCC: 0.0607
F1 Score: 0.1078

The MCC value of 0.0607 indicates a slight correlation between predictions and actual values.
The F1 Score of 0.1078 suggests that the model's ability to balance precision and recall is relatively weak.

Based on the provided evaluation metrics, the CatBoostClassifier exhibits the highest Matthews Correlation Coefficient (MCC) and F1 Score among the tested classifiers, suggesting its potential as the top-performing model for the given dataset.

Saving the Best Performed model with joblib

```
import joblib
joblib.dump(rf_model_o_N, "anomaly_test.joblib", compress=True)  # Use the 'compress' parameter to reduce size
```

Deployment of the project:

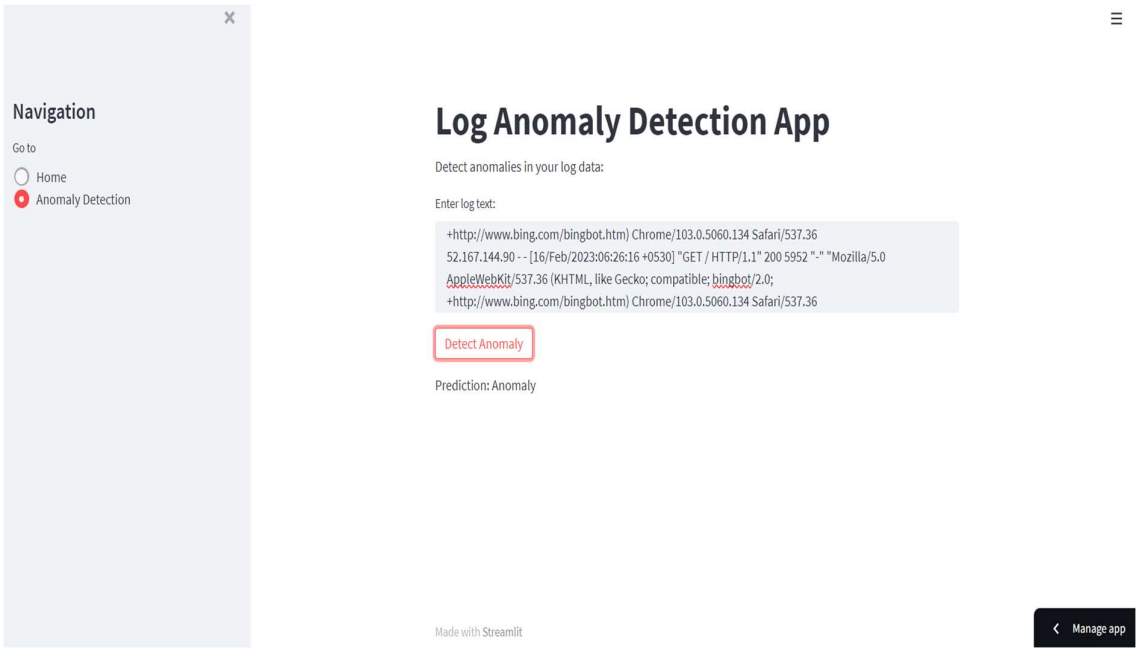
The project deployment of anomaly detection involves the utilization of several key technologies to ensure a seamless and user-friendly experience. Streamlit, a powerful Python library, serves as the foundation for creating interactive and dynamic web applications with minimal effort. Leveraging Streamlit, the anomaly detection model's results and insights are presented to users through an intuitive and visually appealing interface. This interface allows users to easily upload their data, trigger the anomaly detection process, and visualize the detected anomalies and patterns.

GitHub plays a crucial role in the project's deployment pipeline by providing version control and collaboration capabilities. The project's source code, including the anomaly detection model and the Streamlit application script, can be hosted on GitHub repositories. This allows for easy sharing, collaboration, and tracking of changes. Additionally, GitHub's Actions or other CI/CD (Continuous Integration/Continuous Deployment) tools can be configured to automate the deployment process. Whenever new changes are pushed to the repository, the deployment pipeline can be triggered, ensuring that the latest version of the application is available to users.

Pickle, a built-in Python module, is employed for serializing and deserializing the trained anomaly detection model. After the model has been trained on historical data, it can be saved to a pickle file. During the deployment phase, the saved model can be loaded from the pickle file, eliminating the need for retraining the model each time the application is launched. This significantly reduces the computational overhead and allows for quicker response times.

In summary, the project's deployment of anomaly detection brings together Streamlit, GitHub, and Pickle to create an interactive and accessible anomaly detection application. Streamlit provides the user interface, GitHub offers version control and deployment automation, while Pickle optimizes the model loading process. This integration of technologies ensures a seamless deployment experience, enabling users to effortlessly detect anomalies in their data through a user-friendly web application.

Visual Dashboard:



Chapter-5

Applications

Web application log analysis has many applications across different industries and business domains. Here are some of the key applications of web application log analysis:

Website performance optimization: Web application log analysis can help businesses optimize the performance of their websites by identifying slow-loading pages, high traffic times, and other issues that may affect the user experience. By optimizing website performance, businesses can improve user engagement and increase conversions.

Marketing and advertising: Web application log analysis can help businesses identify the most popular pages on their website, as well as the sources of traffic and user behavior patterns. This information can be used to optimize marketing and advertising campaigns to better target the right audience and increase conversions.

Security and fraud prevention: Web application log analysis can help businesses detect and prevent security threats, such as DDoS attacks, hacking attempts, and phishing scams. By analyzing server logs, businesses can identify suspicious activity, block malicious IP addresses, and take other measures to enhance security.

E-commerce and online sales: Web application log analysis can help e-commerce businesses optimize their online stores by identifying customer behavior patterns, such as the most popular products, common search queries, and shopping cart abandonment rates. This information can be used to optimize product offerings, improve website design, and increase sales.

IT operations and infrastructure management: Web application log analysis can help IT teams monitor server and network performance, identify and resolve errors and issues, and optimize infrastructure to improve uptime and reduce downtime.

Conclusion

The project encompasses automating anomaly detection in web application logs through the utilization of machine learning techniques. The primary objective is to efficiently recognize abnormal patterns, particularly potential security threats, within these logs. This is achieved by employing various machine learning algorithms, including Isolation Forest, SVM, and CatBoost, to construct predictive models capable of distinguishing between normal and anomalous log entries with a focus on high accuracy and minimal false positives.

The selection of the CatBoost algorithm is particularly beneficial due to its ability to effectively manage categorical features present in web application logs. These categorical features, such as HTTP status codes, play a crucial role in pinpointing unusual behavior. The project follows a comprehensive methodology that encompasses essential stages like data preprocessing, feature engineering, model selection, hyperparameter tuning, and thorough performance assessment. The evaluation is guided by metrics such as precision, recall, and F1-score, which collectively ensure a robust evaluation of model performance.

The integration of PyCaret and PyOD further enhances the project's anomaly detection capabilities. PyCaret automates crucial tasks, streamlining the workflow by handling data preprocessing, feature engineering, and model selection. This accelerates the overall development process and facilitates the exploration of various algorithms, including Isolation Forest, SVM, and CatBoost. Concurrently, PyOD extends the project's capabilities by offering a broader array of anomaly detection techniques beyond the core algorithms. This combination empowers the project to tackle complex security threats and intricate anomalies prevalent within web application logs, contributing to an enhanced security posture.

Ultimately, the results underscore the project's efficacy in identifying security breaches and anomalies within web application logs. The automation facilitated by PyCaret and PyOD ensures proactive threat mitigation, thus elevating web application security. Through the systematic incorporation of advanced algorithms and the strategic utilization of categorical features, the project presents a scalable and efficient solution for precise anomaly identification and improved security incident response.

While the conclusion regarding the superior performance of CatBoost seems to be given its efficient handling of categorical features, it's important to note that the actual comparative performance should be determined through rigorous experimentation and evaluation of the algorithms using appropriate evaluation metrics.

References

1] Anomaly Detection in Log Files Using Machine Learning Techniques by Lakshmi Geethanjali Mandagondi

2] Anomaly detection with Machine learning by Hanna Blomquist and Johanna Möller:

3] Attack and Anomaly Detection in IoT Sites Using Machine Learning Techniques by Aziza Khalilahmed Shaikh and Prof Govind Negalur

4] Implementation of Anomaly Detection Technique Using Machine Learning Algorithms by K. Hanumantha Rao, G. Srinivas, Ankam Damodhar and M. Vikas Krishna

5] AMixed Clustering Approach for Real-Time Anomaly Detection by Fokrul Alom Mazarbhuiya 1, and Mohamed Shenify 2

6] Analysis of Machine Learning Algorithms for Anomaly Detection on Edge Devices by Aleks Huˇc, Jakob Šalej and Mira Trebar

7] Detection of Anomaly using Machine Learning: A Comprehensive Survey

8] Web Based Anomaly Detection using Machine Learning