SUMMIT
TECHNOLOGY GROUP

CHITKARA
UNIVERSITY

Anomaly/Pattern Detection from a huge set of data

Team Jarvis

Rahul Mittal

Roll No. - 2010991215
Batch - 2020-2024(BE.CSE)
Contact - 8619970871
Email -
rahul1215@chitkara.edu.in

rahulResume.pdf

Akshit Bansal

Roll No. - 2010991225
Batch - 2020-2024 (BE.CSE)
Contact - 7888500941
Email -
akshit1225@chitkara.edu.in

akshitResume.pdf

Dikshant Mitrav

Roll No. - 2010990203
Batch - 2020-2024(BE.CSE)
Contact - 9416411727
Email -
dikshant0203.be20@chitkara.edu.in

dikshantResume.pdf

Srijan Girdhar

Roll No. - 2010990702
Batch - 2020-2024(BE.CSE)
Contact - 7652931326
Email -
srijan0702.be20@chitkara.edu.in

srijanResume.pdf

Hackathon's Agenda

1. Introduction
2. Project Overview
3. Dataset Information
4. Step 1 - (EDA)
5. Step 2 Feature Engineering
6. Encoding of Data
7. Model Training
8. Saving the Trained Model
9. Running the Model
10. Conclusion

Introduction

- Anomaly detection plays a crucial role in identifying and mitigating potential security threats. It involves identifying patterns or behaviors that deviate significantly from normal expectations.
- This project focuses on detecting anomalies in user login behavior, which can help identify suspicious activities such as unauthorized access attempts or compromised user accounts.
- Our goal is to develop an anomaly detection model that can automatically classify new login events as normal or anomalous based on their features.
- We will be utilising machine learning techniques, to train the anomaly detection model on a labeled dataset.

Project Overview

- Our project involves training an anomaly detection model to classify login events as normal or anomalous.
- The machine learning algorithm we will be using is XGBoost (for midway submission), a powerful and optimized gradient boosting algorithm known for its speed and performance.
- By training our model on a dataset, we aim to create a reliable and accurate system for identifying anomalous login behavior.
- The trained model can be used to automate the detection process, saving time and effort for security analysts.

Dataset Information

```
In [3]: df.head()
```

Out[3]:

	Login Timestamp	User ID	IP Address	Country	Region	City	Browser Name and Version	Device Type	Login Successful
0	2020-02-03 12:43:30.772	-4324475583306591935	10.0.65.171	NO	-	-	Firefox 20.0.0.1618	mobile	False
1	2020-02-03 12:43:43.549	-4324475583306591935	194.87.207.6	AU	-	-	Chrome Mobile 46.0.2490	mobile	False
2	2020-02-03 12:43:55.873	-3284137479262433373	81.167.144.58	NO	Vestland	Urangsvag	Android 2.3.3.2672	mobile	True
3	2020-02-03 12:43:56.180	-4324475583306591935	170.39.78.152	US	-	-	Chrome Mobile WebView 85.0.4183	mobile	False
4	2020-02-03 12:43:59.396	-4618854071942621186	10.0.0.47	US	Virginia	Ashburn	Chrome Mobile WebView 85.0.4183	mobile	False

- Login Timestamp: The date and time when the login attempt took place.
- User ID: A unique identification number assigned to each user attempting to log in.
- IP Address: The distinct address associated with the device used for the login attempt.
- Country: The country of origin for the login attempt.
- Region: The specific region or state associated with the IP address, if available.
- City: The city linked to the IP address, if available.
- Browser Name and Version: The name and version of the web browser used for the login attempt.
- Device Type: The category of device used for the login attempt, such as a mobile device or a desktop computer.
- Login Successful: A binary indicator indicating whether the login attempt was successful (True) or unsuccessful (False).

Step 1 – Exploratory Data Analysis (EDA)

- Exploratory Data Analysis (EDA) is an essential step in understanding the characteristics and patterns of the data before building the anomaly detection model.
- EDA helps identify potential data issues, outliers, relationships between variables, and distribution of features.

EDA Step 1 – Load the Data

- The first step in EDA is to load the raw login data into our analysis environment.
- We typically use pandas, a popular Python and numpy library, to read and manipulate the data.
- Pandas provides powerful tools for data manipulation, cleaning, and summarization.

EDA Step 2 – Data Exploration

- After loading the data, we perform various exploratory analyses to gain insights into the data.
- We can start by examining the structure of the dataset using pandas' methods like info(), and describe(), .shape.
- These methods provide an overview of the dataset, including the column names, data types, summary statistics, and a sample of the data.

```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31269264 entries, 0 to 31269263
Data columns (total 9 columns):
 #   Column                      Dtype
---  ------                      -----
 0   Login Timestamp             object
 1   User ID                     int64
 2   IP Address                  object
 3   Country                     object
 4   Region                      object
 5   City                        object
 6   Browser Name and Version    object
 7   Device Type                 object
 8   Login Successful            bool
dtypes: bool(1), int64(1), object(7)
memory usage: 1.9+ GB
```

```
In [4]: df.describe()
```

Out[4]:

	User ID
count	3.126926e+07
mean	-1.924043e+18
std	4.514276e+18
min	-9.223371e+18
25%	-4.324476e+18
50%	-4.324476e+18
75%	9.119813e+17
max	9.223359e+18

```
In [3]: df.shape
```

```
Out[3]: (31269264, 9)
```

EDA Step 3 – Missing Data

- Missing data is a common issue in datasets and can impact the performance of our anomaly detection model.
- We need to identify missing values in the dataset and handle them appropriately.
- Pandas provides methods like isnull() and fillna() to detect missing values and fill them with suitable values.

```
In [6]: df.isnull().sum()

Out[6]: Login Timestamp                0
        User ID                        0
        IP Address                     0
        Country                        0
        Region                     47409
        City                        8590
        Browser Name and Version       0
        Device Type                 1526
        Login Successful               0
        dtype: int64
```

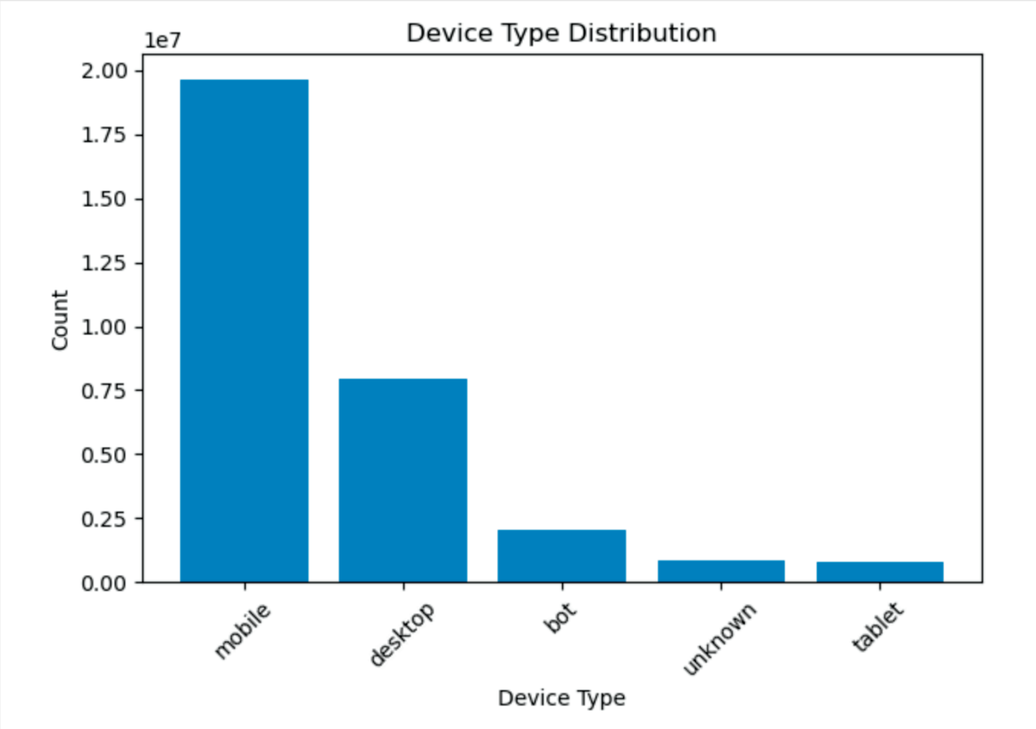
```
In [27]: df.isnull().sum()

Out[27]: Login Timestamp    0
         User ID            0
         IP Address         0
         Country            0
         Device Type        0
         Login Successful   0
         Browser Category   0
         LoginRatio         0
         dtype: int64
```

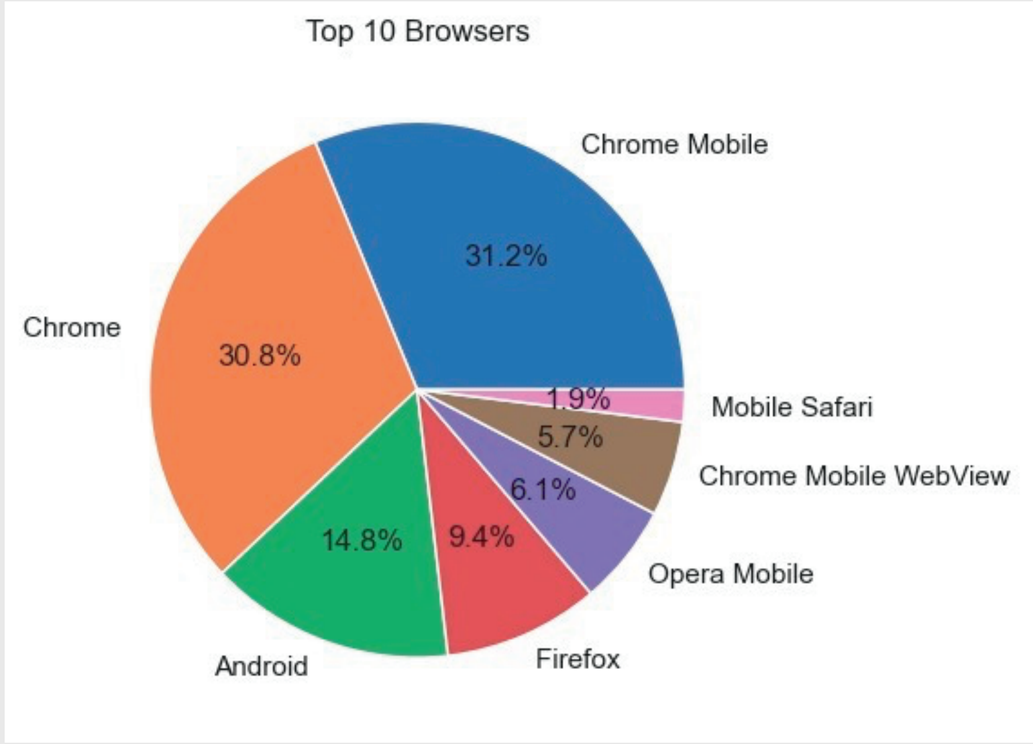
This code snippet replaces missing values with 'unknown', and then further replaces any instances of 'unknown' with 'other' if they were present.

```
In [24]: df['Device Type'] = df['Device Type'].fillna('unknown')
         df['Device Type'] = df['Device Type'].replace('unknown', 'other')
```

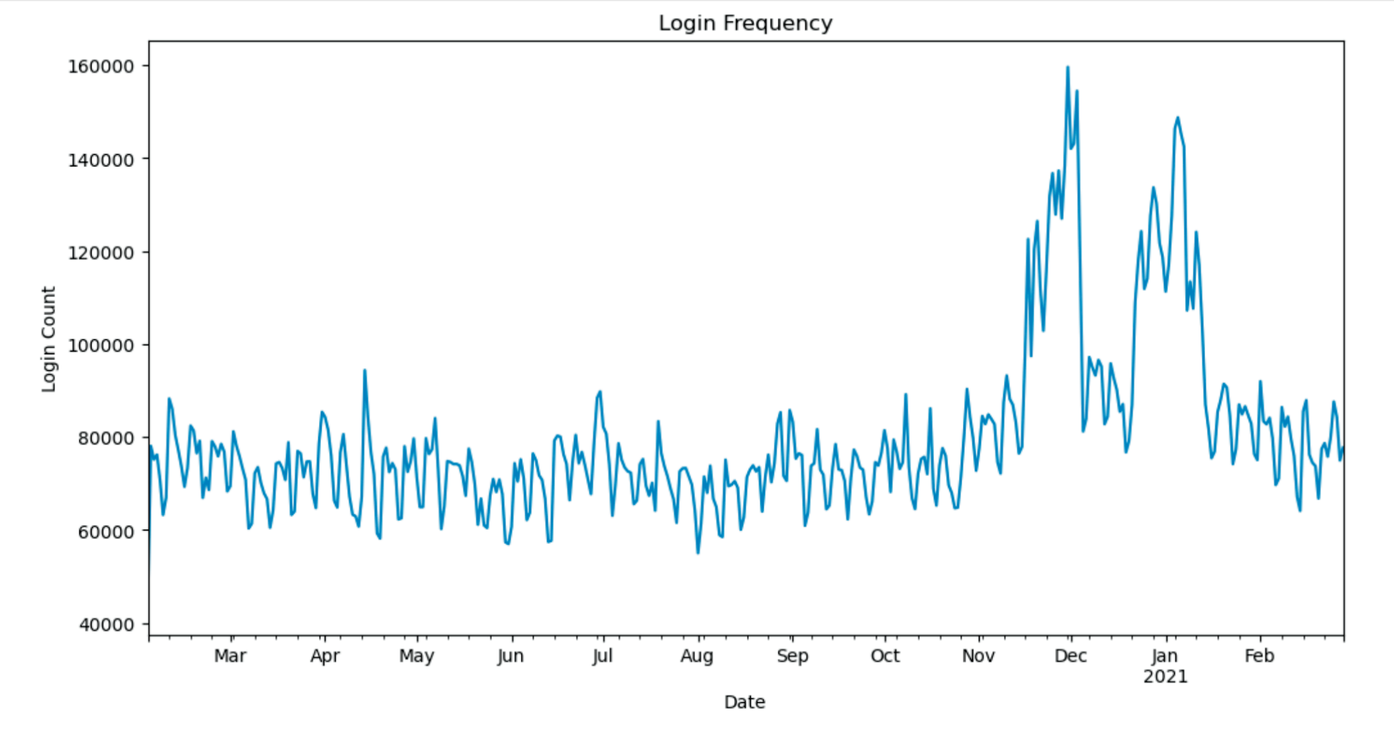
EDA Step 4 – Data Visualization

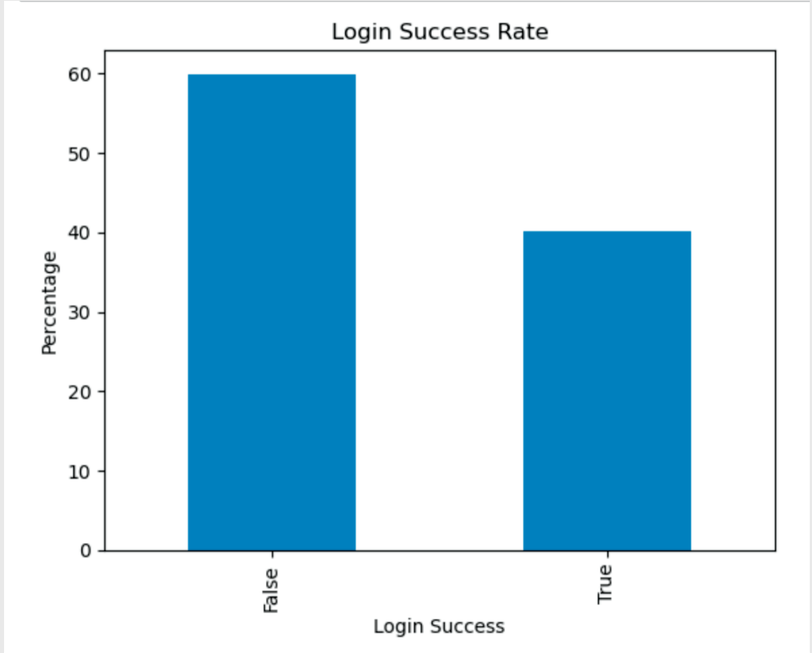
- Data visualisation is a powerful tool for gaining insights and understanding patterns in the data.
- We can use various libraries like Matplotlib and Seaborn to create visualizations such as histograms, bar plots, scatter plots, and heatmaps.
- Visualizations help identify outliers, distribution of features, correlations, and potential patterns.





EDA Step 4 – Data Visualization





- Exploratory Data Analysis (EDA) is a crucial step in understanding the dataset and preparing it for anomaly detection.
- By loading the data, exploring its structure, handling missing values, and visualizing the data, we gain insights and identify patterns.
- EDA helps us make informed decisions during feature engineering, model selection, and anomaly detection.

Step 2 – Feature Engineering

- Feature engineering is a critical step in the anomaly detection project, where we transform and create new features from the raw data.
- Effective feature engineering helps improve the performance of the anomaly detection model by providing relevant and informative input.

Feature Engineering Step 1 – Feature Selection

- Feature selection involves choosing the most relevant features from the available data.
- We aim to include features that have a significant impact on detecting anomalies while excluding irrelevant or redundant ones.

Login Timestamp	User ID	IP Address	Country	Region	City	Browser Name and Version	Device Type	Login Successful

Feature Engineering Step 2 – Feature Creation

- Feature creation involves deriving new features from the existing ones to capture additional information or relationships.
- This step leverages domain knowledge, data understanding, and insights gained during exploratory data analysis

User ID	IP Address	Country	Device Type	Login Successful	Browser Category	LoginRatio	Total Devices Per User	Total IP Addresses Per User	Total Countries Per User	Total Browser Categories Per User	Time Difference	target

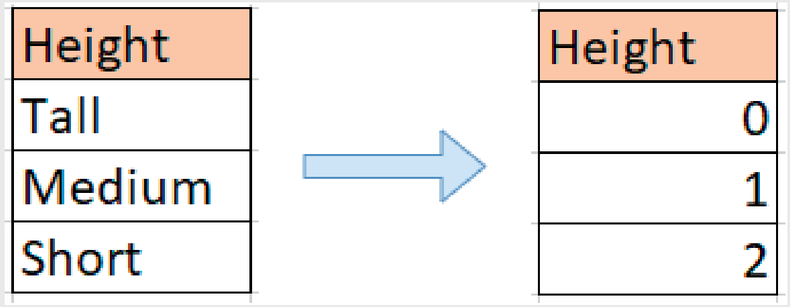
- Feature engineering plays a crucial role in preparing the data for anomaly detection.
- Feature selection, transformation, and creation help enhance the model's performance and capture relevant information.
- Effective feature engineering relies on domain knowledge, data understanding, and iterative exploration.

Step 3 – Encoding of Data

- Encoding of data is a crucial step in preparing categorical variables for machine learning algorithms.
- It involves converting categorical variables into numerical representations that can be understood by the model.
- One of the commonly used techniques is Label Encoding.

Label Encoding

- Label Encoding is a technique used to transform categorical variables into numeric labels.
- Each unique category is assigned a unique integer value.
- Label Encoding is suitable for variables with ordinal relationships or when the number of categories is large.



Label Encoding Considerations

- Encoding of data is essential to convert categorical variables into numerical representations for machine learning algorithms.
- Label Encoding is a technique that assigns unique integer labels to each category.
- Consider the nature of the data and the requirements of the problem when choosing the encoding technique.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17243365 entries, 0 to 17243364
Data columns (total 14 columns):
 #   Column                          Dtype
---  ------                          -----
 0   Login Timestamp                 datetime64[ns]
 1   User ID                         int64
 2   IP Address                      object
 3   Country                         object
 4   Device Type                     object
 5   Login Successful                bool
 6   Browser Category                object
 7   LoginRatio                      float64
 8   Total Devices Per User          int64
 9   Total IP Addresses Per User     int64
 10  Total Countries Per User        int64
 11  Total Browser Categories Per User  int64
 12  Time Difference                 float64
 13  target                          int64
dtypes: bool(1), datetime64[ns](1), float64(2), int64(6), object(4)
memory usage: 1.8+ GB
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17243365 entries, 0 to 17243364
Data columns (total 11 columns):
 #   Column                          Dtype
---  ------                          -----
 0   Country                         int64
 1   Device Type                     int64
 2   Login Successful                int64
 3   Browser Category                int64
 4   LoginRatio                      float64
 5   Total Devices Per User          int64
 6   Total IP Addresses Per User     int64
 7   Total Countries Per User        int64
 8   Total Browser Categories Per User  int64
 9   Time Difference                 float64
 10  target                          int64
dtypes: float64(2), int64(9)
memory usage: 1.5 GB
```

Encoding of Data Summary

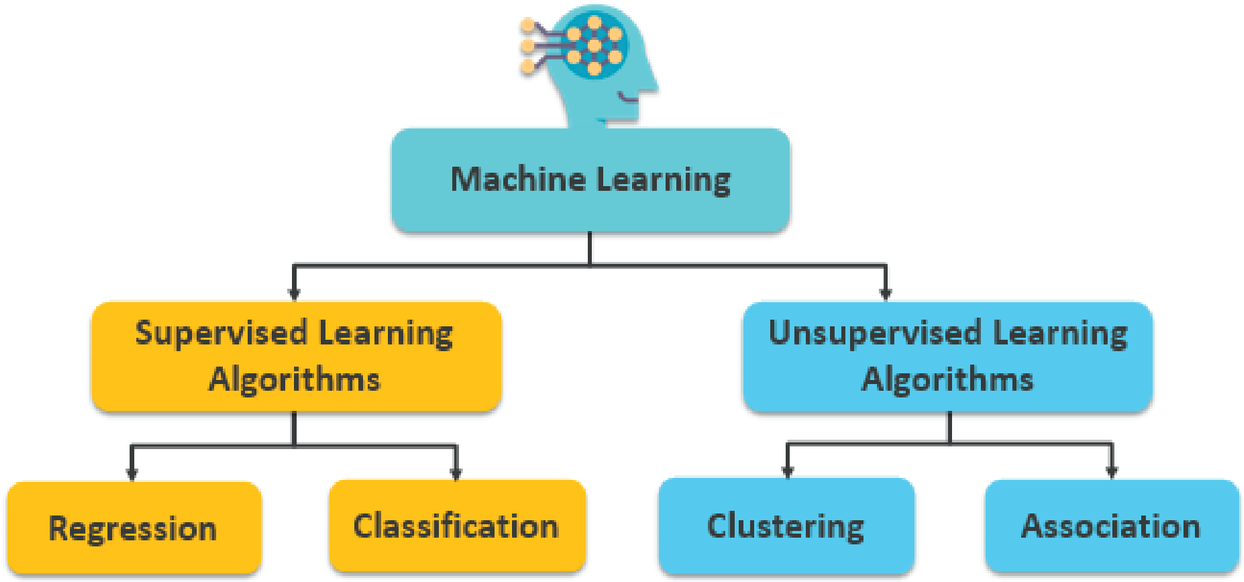
- Encoding of data is essential to convert categorical variables into numerical representations for machine learning algorithms.
- Label Encoding is a technique that assigns unique integer labels to each category.
- Consider the nature of the data and the requirements of the problem when choosing the encoding technique.

	Country	Device Type	Login Successful	Browser Category	LoginRatio	Total Devices Per User	Total IP Addresses Per User	Total Countries Per User	Total Browser Categories Per User	Time Difference	target
8240075	157	2	0	3	1.000000	1	3	1	1	143.822	0
5503482	146	2	0	8	1.166667	2	8	1	4	233.452	2
2686647	202	2	1	3	0.333333	1	2	1	1	0.000	0
724282	146	2	1	17	0.105263	2	8	1	3	149514.910	1
16279654	202	2	1	3	0.500000	1	1	1	1	28.320	0

Model Training

Unsupervised Learning vs. Supervised Learning

- In machine learning, we have two main types of learning: unsupervised learning and supervised learning.
- Unsupervised learning involves finding patterns and structures in data without explicit target labels.
- Supervised learning, on the other hand, utilizes labeled data to learn a mapping between input features and corresponding target labels.



Unsupervised Dataset to Supervised Dataset

- In our anomaly detection project, we transformed the unsupervised dataset into a supervised dataset by adding a target column based on various conditions and business rules.
- By incorporating this target column, we converted the unsupervised anomaly detection problem into a supervised learning problem.
- This allowed us to leverage the power of supervised learning algorithms and train a model to predict the presence or absence of anomalies.

target
0
2
0
1
0

Benefits of Adding a Target Column:

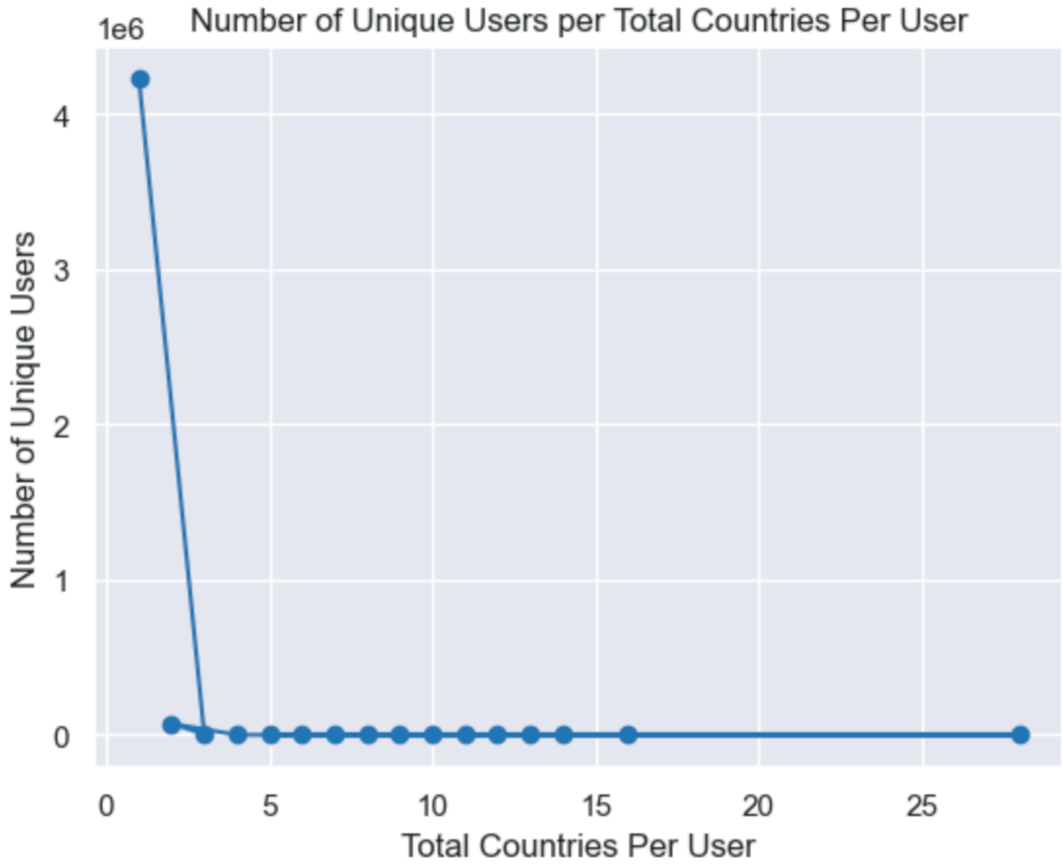
- Enabling Supervised Learning: By adding the target column, we transformed the anomaly detection problem into a supervised learning problem. T
- Facilitating Model Training: With labeled data, we can utilize various supervised learning algorithms, including XGBoost, to train a model that learns the patterns and characteristics of both normal and anomalous login events.
- Evaluation and Performance Metrics: The addition of the target column enables the calculation of performance metrics such as accuracy, precision, which provide insights into the model's performance and effectiveness in detecting anomalies.
- Future Enhancements: The availability of a labeled dataset opens doors for further exploration and experimentation with advanced techniques, such as anomaly detection algorithms that combine unsupervised and supervised learning approaches, ensemble methods, or deep learning architectures.

How our target variable is defined

- We basically made the graphs for the all the unique users and did the EDA and like we came out with the general case (it can vary as well, just assumption).
- So first case unique number of users with the total countries per user, as we can see that most of the users have the total number of countries is not more than 3. So we took the average case as 2

```
unique_values = df["Total Countries Per User"].unique()

# Initialize an empty dictionary to store the results
results = {}

# Iterate over each unique value
for value in unique_values:
    # Filter the data for the current value
    filtered_data = df[df["Total Countries Per User"] == value]
    # Count the number of unique users
    unique_users = filtered_data["User ID"].nunique()
    # Store the result in the dictionary
    results[value] = unique_users

# Print the results
for value, count in results.items():
    print("Total number of unique users with", value, "total countries per user:", cou
```

```
Total number of unique users with 1 total countries per user: 4226824
Total number of unique users with 3 total countries per user: 4885
Total number of unique users with 2 total countries per user: 72293
Total number of unique users with 4 total countries per user: 612
Total number of unique users with 7 total countries per user: 22
Total number of unique users with 5 total countries per user: 146
Total number of unique users with 28 total countries per user: 1
Total number of unique users with 6 total countries per user: 47
Total number of unique users with 10 total countries per user: 2
Total number of unique users with 8 total countries per user: 9
Total number of unique users with 9 total countries per user: 5
Total number of unique users with 11 total countries per user: 2
Total number of unique users with 16 total countries per user: 1
Total number of unique users with 12 total countries per user: 4
Total number of unique users with 13 total countries per user: 2
Total number of unique users with 14 total countries per user: 1
```

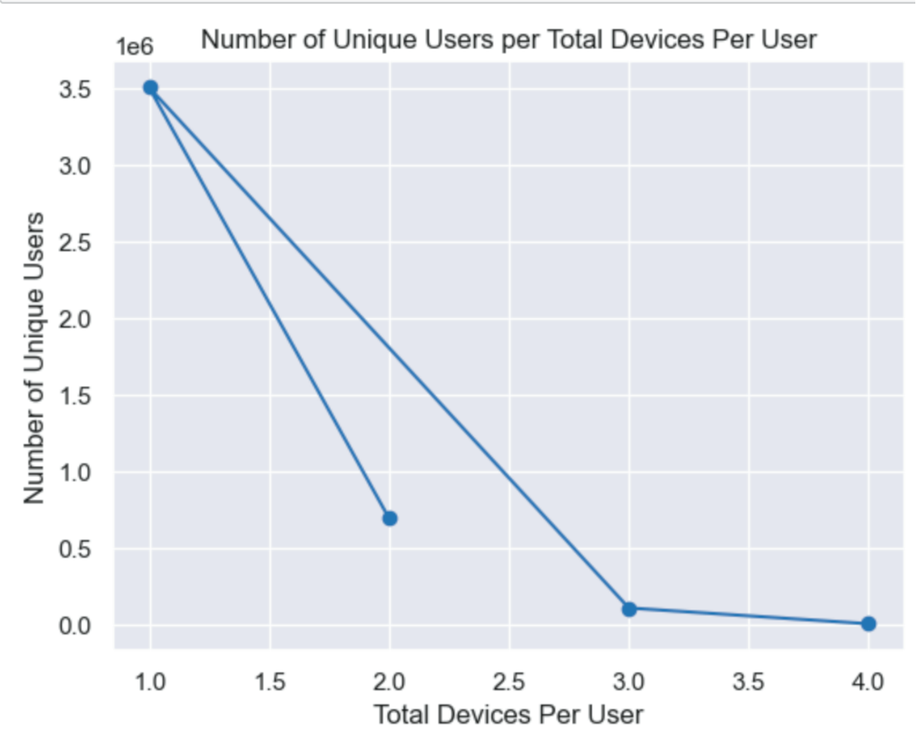


How our target variable is defined

- So second case unique number of users with the total devices per user, as we can see that most of the users have the total number of devices is not more than 3. So we took the average case as3

```
unique_values = df["Total Devices Per User"].unique()

# Initialize an empty dictionary to store the results
results = {}

# Iterate over each unique value
for value in unique_values:
    # Filter the data for the current value
    filtered_data = df[df["Total Devices Per User"] == value]
    # Count the number of unique users
    unique_users = filtered_data["User ID"].nunique()
    # Store the result in the dictionary
    results[value] = unique_users

# Print the results
for value, count in results.items():
    print("Total number of unique users with", value, "Total Devices Per User:", count)
```

```
Total number of unique users with 2 Total Devices Per User: 691895
Total number of unique users with 1 Total Devices Per User: 3507215
Total number of unique users with 3 Total Devices Per User: 104629
Total number of unique users with 4 Total Devices Per User: 1117
```

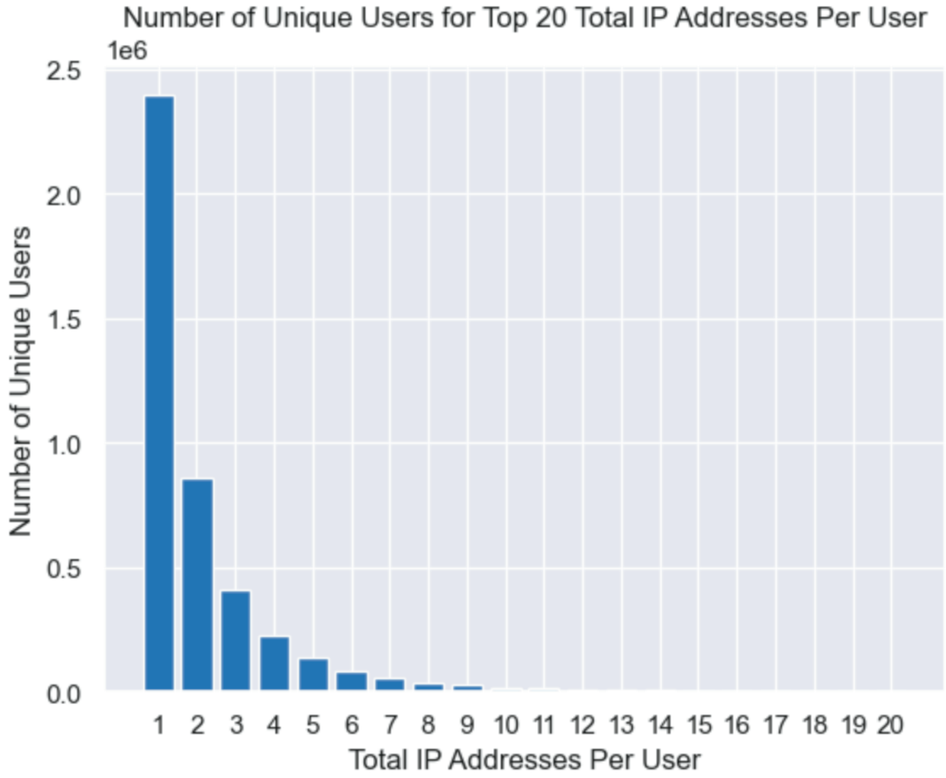


How our target variable is defined

- So third case unique number of users with the total IP Address per user, as we can see that most of the users have the total number of IP Address is not more than 3. So we took the average case as 3-4

```
unique_values = df["Total IP Addresses Per User"].value_counts().nlargest(10).index.tolist()

# Initialize an empty dictionary to store the results
results = {}

# Iterate over each unique value
for value in unique_values:
    # Filter the data for the current value
    filtered_data = df[df["Total IP Addresses Per User"] == value]
    # Count the number of unique users
    unique_users = filtered_data["User ID"].nunique()
    # Store the result in the dictionary
    results[value] = unique_users

# Print the results
for value, count in results.items():
    print("Total number of unique users with", value, "Total IP Addresses Per User:", count)
```

```
Total number of unique users with 1 Total IP Addresses Per User: 2395287
Total number of unique users with 2 Total IP Addresses Per User: 856455
Total number of unique users with 3 Total IP Addresses Per User: 412374
Total number of unique users with 4 Total IP Addresses Per User: 227283
Total number of unique users with 5 Total IP Addresses Per User: 136490
Total number of unique users with 6 Total IP Addresses Per User: 85726
Total number of unique users with 7 Total IP Addresses Per User: 56071
Total number of unique users with 8 Total IP Addresses Per User: 37407
Total number of unique users with 9 Total IP Addresses Per User: 25751
Total number of unique users with 10 Total IP Addresses Per User: 17617
```

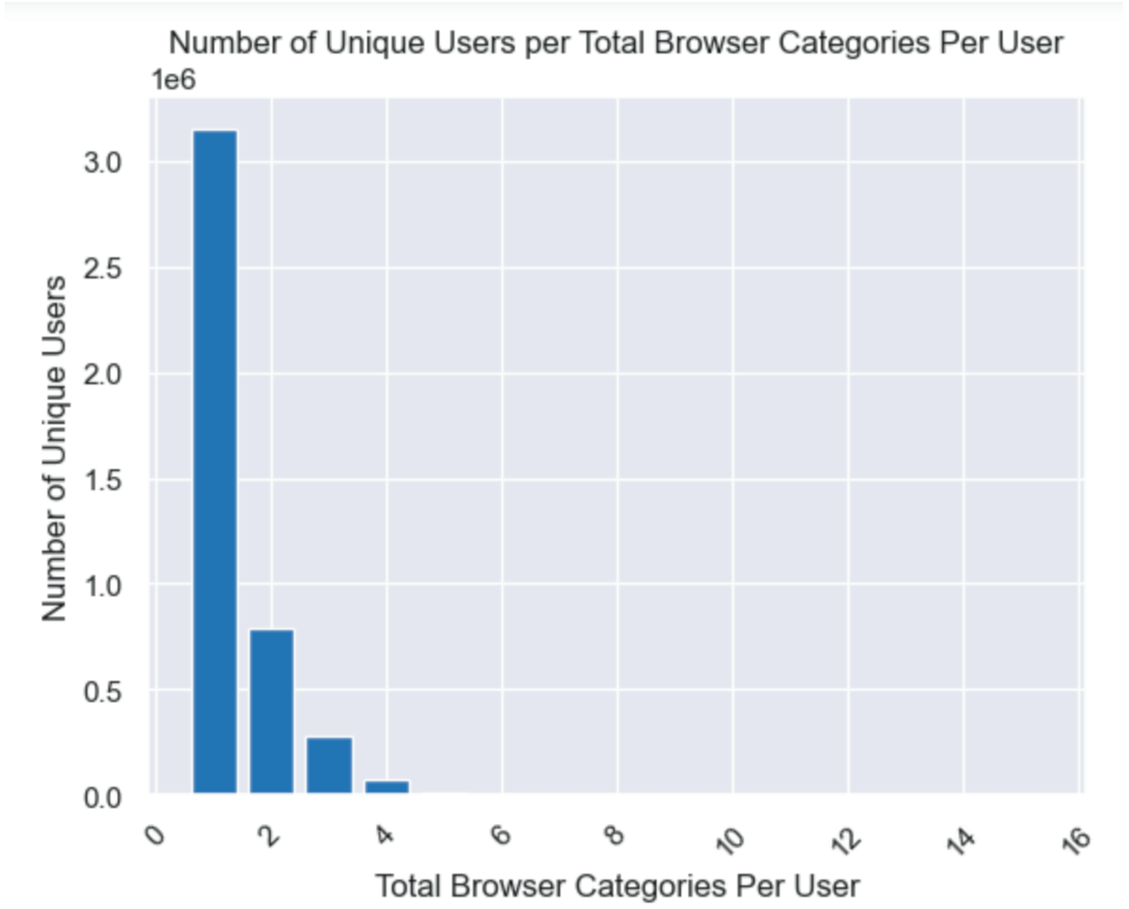


How our target variable is defined

- So fourth case unique number of users with the total browser category per user, as we can see that most of the users have the total number of browser category is not more than 3. So we took the average case as 3

```
: unique_values = df["Total Browser Categories Per User"].unique()

  # Initialize an empty dictionary to store the results
  results = {}

  # Iterate over each unique value
  for value in unique_values:
      # Filter the data for the current value
      filtered_data = df[df["Total Browser Categories Per User"] == value]
      # Count the number of unique users
      unique_users = filtered_data["User ID"].nunique()
      # Store the result in the dictionary
      results[value] = unique_users

  # Print the results
  for value, count in results.items():
      print("Total number of unique users with", value, "Total Browser Categories Per User", count)
```

```
Total number of unique users with 4 Total Browser Categories Per User 74211
Total number of unique users with 1 Total Browser Categories Per User 3149197
Total number of unique users with 2 Total Browser Categories Per User 789208
Total number of unique users with 3 Total Browser Categories Per User 275222
Total number of unique users with 5 Total Browser Categories Per User 14275
Total number of unique users with 7 Total Browser Categories Per User 359
Total number of unique users with 6 Total Browser Categories Per User 2339
Total number of unique users with 10 Total Browser Categories Per User 3
Total number of unique users with 8 Total Browser Categories Per User 34
Total number of unique users with 15 Total Browser Categories Per User 1
Total number of unique users with 9 Total Browser Categories Per User 6
Total number of unique users with 12 Total Browser Categories Per User 1
```



Boosting Algorithms

- Boosting algorithms are a family of machine learning algorithms that combine weak learners into a strong learner.
- They work iteratively by training weak models sequentially, with each subsequent model focusing more on the samples that previous models misclassified.
- The final model is an ensemble of these weak models, where each model's predictions are weighted based on their performance.

XGBoost - Classifier and Regressor

- XGBoost is an optimised gradient boosting algorithm, its strength lies in its ability to handle complex data patterns, large datasets, and high-dimensional features, employs an ensemble of decision trees and uses gradient boosting techniques to improve model performance.
- XGBoost Classifier: It is used for classification tasks, where the target variable is categorical. The algorithm optimises a loss function specific to classification, such as log loss or cross-entropy, and predicts class probabilities.
- XGBoost Regressor: It is used for regression tasks, where the target variable is continuous. The algorithm optimises a loss function specific to regression, such as mean squared error (MSE) or mean absolute error (MAE), and predicts the numerical value.

Why We used XGBoost Regressor

There are several reasons why we chose the XGBoost Regressor for our anomaly detection project:

- Handling Complex Patterns: XGBoost can capture complex relationships between features, making it suitable for detecting anomalies in intricate data patterns.
- Robustness to Noisy Data: XGBoost is less sensitive to noisy data and outliers, allowing it to handle real-world datasets that often contain anomalies or irregularities.
- Scalability: XGBoost efficiently handles large datasets and high-dimensional features, enabling us to process and train the model on substantial amounts of data.
- Performance: XGBoost is known for its high performance and accuracy, consistently delivering competitive results in machine learning competitions and real-world applications.

The following code creates and trains an XGBoost regression model using the XGBRegressor class from the xgboost library. It fits the model to the training data, allowing it to learn the patterns and relationships necessary for making predictions on unseen data.

In [ ]:
```
from xgboost import XGBRegressor
regressor = XGBRegressor()
regressor.fit(X_train, y_train)
```

Evaluation Metrics

To assess the performance of our trained model, we calculate several evaluation metrics.

- Mean Squared Error (MSE): It measures the average squared difference between the predicted and actual values. A lower MSE indicates better performance.
- Root Mean Squared Error (RMSE): It is the square root of MSE and represents the standard deviation of the residuals. RMSE provides a more interpretable metric since it's in the same unit as the target variable.
- Mean Absolute Error (MAE): It calculates the average absolute difference between the predicted and actual values. MAE is less sensitive to outliers compared to MSE.

Evaluation Metrics Calculation

```
Mean Squared Error (MSE): 0.0030612325056118346
Root Mean Squared Error (RMSE): 0.05532840595581834 5
Mean Absolute Error (MAE): 0.006691295544221171 5
```

Future Model Exploration

Random Forest:

- Random Forest is an ensemble learning algorithm that combines multiple decision trees to make predictions.
- It is known for its ability to handle high-dimensional data, handle outliers, and provide feature importance.
- Random Forest can be effective in anomaly detection by capturing complex relationships and identifying patterns in the data.

Support Vector Machines (SVM):

- SVM is a supervised learning algorithm that can be adapted for anomaly detection.
- It separates the data into different classes using a hyperplane, with the aim of maximising the margin between the classes.
- SVM can be useful for identifying anomalies by finding data points that fall on the wrong side of the hyperplane or have a low margin.

Isolation Forest:

- Isolation Forest is an unsupervised learning algorithm specifically designed for anomaly detection.
- It constructs isolation trees by randomly selecting features and splitting points, which makes it efficient for high-dimensional data.
- By isolating anomalies, Isolation Forest can quickly identify anomalies as data points that require fewer splits to separate from the rest.

Saving the Trained Model:

Once the anomaly detection model is trained, it is essential to save it for future use and deployment. In Python, the pickle library provides a convenient way to save machine learning models.

```
import pickle
filename = 'regressor_model.pkl'
with open(filename, 'wb') as file:
    pickle.dump(regressor, file)
```

Benefits of Saving Models using Pickle:

- Reusability: Saving the model allows us to reuse it without retraining, saving time and computational resources.
- Deployment: The saved model can be easily deployed in production environments or integrated into other systems for real-time anomaly detection.
- Consistency: By saving the model, we ensure consistent results as the same model can be used across different environments and platforms.
- Collaboration: Saved models can be shared and used by multiple users or systems, promoting collaboration and scalability.

Running the Anomaly Detection Model with FastAPI

FastAPI

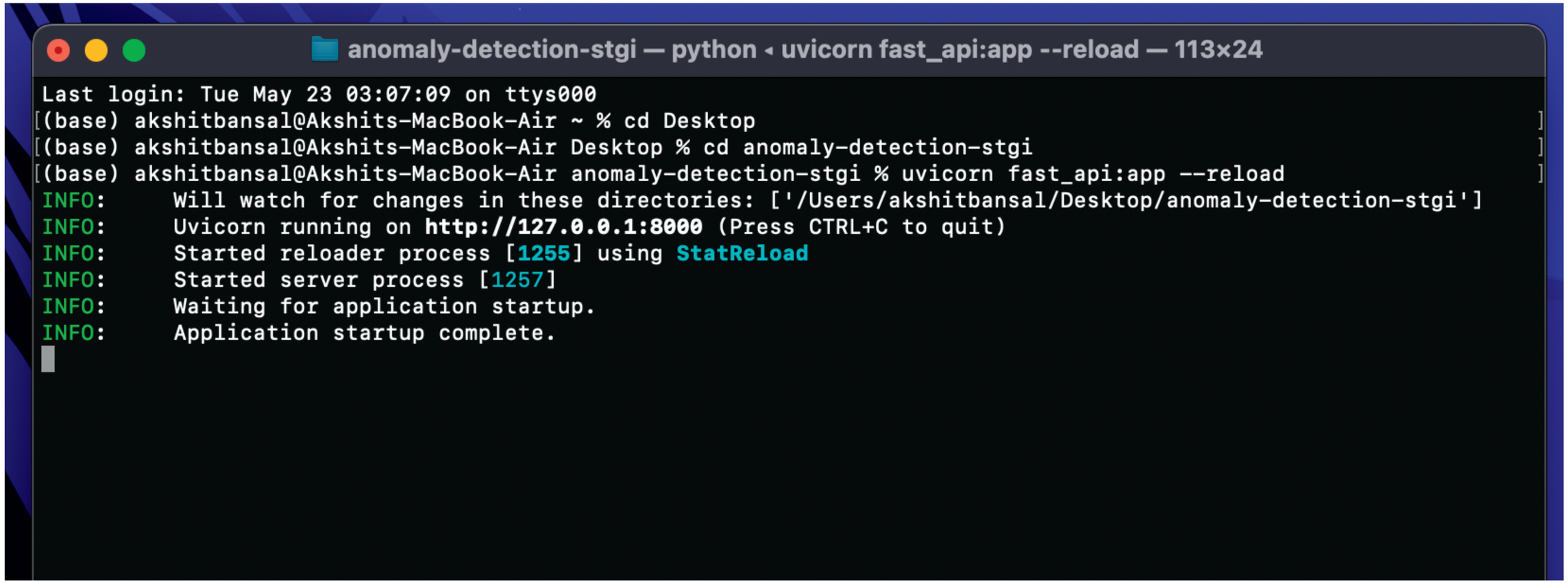
FastAPI is a modern, high-performance web framework for building APIs with Python. It provides a seamless way to deploy our anomaly detection model as a web service, allowing users to make predictions through HTTP requests.

Why are we using FastAPI?

- FastAPI is built on top of Starlette, a high-performance asynchronous web framework. It leverages the power of asynchronous programming to handle a large number of requests efficiently.
- FastAPI provides a clean and intuitive API for defining routes, handling request parameters, and returning responses. It follows the principles of modern Python frameworks, making it easy to understand and work with.
- Type Hinting and Validation: FastAPI leverages Python's type hinting feature to perform automatic request/response validation. It ensures that the data sent to and received from the API conforms to the defined data types and structures.

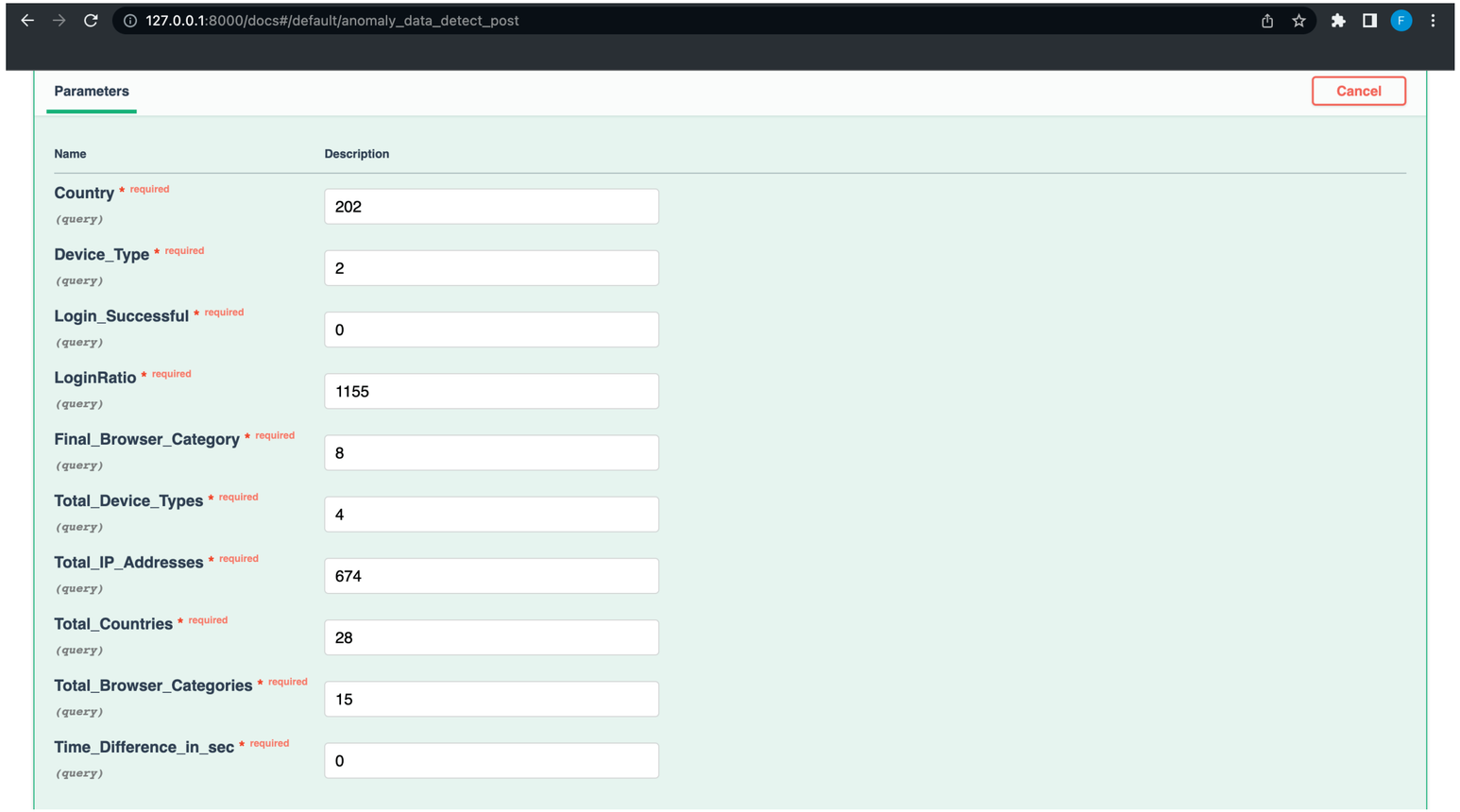
Running the Anomaly Detection Model

- Open the fast_api.py file.
- Open the terminal and navigate to the project folder:
  cd anomaly-detection-project
- Run the FastAPI server:
  uvicorn fast_api:app --reload.



Running the Anomaly Detection Model

- Open your web browser and visit (http://localhost:8000) to ensure the server is running. You should see a "Hello, World!" message.
- Then locate the /docs endpoint and click on it to expand the section.
- Replace the <value> placeholders with the corresponding feature values for anomaly detection.



Running the Anomaly Detection Model

- Click on the execute and check out the anomalous score
- If the score is >= 3 it is considered as anomaly

```
Country                                202.0
Device Type                              2.0
Login Successful                         0.0
Browser Category                         8.0
LoginRatio                            1155.0
Total Devices Per User                   4.0
Total IP Addresses Per User            674.0
Total Countries Per User                28.0
Total Browser Categories Per User       15.0
Time Difference                          0.0
target                                   5.0
Name: 22600, dtype: float64
```

Code	Details
200	**Response body** `{ "anomalous_score": 5.8017683029174805 }`

Conclusion

In conclusion, our anomaly detection project has successfully addressed the challenge of identifying and mitigating potential security threats in user login logs.

The presence of various elements enables the straightforward identification of the cause for the anomaly in a user or session. Specifically, the reason behind the anomaly can be easily determined by considering the target feature.

Through comprehensive data analysis, we gained valuable insights into the dataset and applied feature engineering techniques to extract meaningful information. The project includes steps for installation, data preparation, model training, and running the FastAPI server.

Thank You!