INSTITUTE FOR ADVANCED COMPUTING

AND SOFTWARE DEVELOPMENT

AKURDI, PUNE-411044

Documentation On

"Unsupervised Anomaly Detection in Cloud Network Traffic"

PG-DBDA AUG-2024

Submitted By:

Group No: 25

Arpita Angad Salunke (248513)

Priyanka Kale (248532)

Mrs. Priyanka Bhor
Project Guide

Mr. Rohit Puranik
Centre Coordinator

DECLARATION

I, the undersigned hereby declare that the project report titled "Unsupervised Anomaly Detection in Cloud Network Traffic" written and submitted by me to Institute For Advanced Computing And Software Development Akurdi Pune, in the fulfilment of requirement for the award of degree of Post Graduate Diploma In Big Data Analytics (PG DBDA) under the guidance of Dr Shantanu S Pathak ,Mrs Priyanka Bhor and Mrs Priti Take is my original work I have not copied any code or content from any source without proper attribution, and I have not allowed anyone else to copy my work.

The project was completed using Python and ML and libraries. The project was developed as part of my academic coursework. I also confirm that the project is original, and it has not been submitted previously for any other academic or professional purpose.

Place :	Signature :
Date :	Name : Arpita Salunke / Priyanka Kale

Acknowledgement

We would like to express our sincere gratitude to everyone who has contributed to the completion of our project. First and foremost, we would like to thank our project guide Mrs. Priyanka Bhor mam for their constant guidance and support throughout the project. We extend our sincere thanks to our respected Centre Co- Ordinator**,** Mr. Rohit Puranik, for allowing us to use the facilities available.

We would also like to express our appreciation to the faculty members of our department for their constructive feedback and encouragement. Their insights and suggestions have helped us to refine our ideas and enhance the quality of our work.

Furthermore, we would like to thank our families and friends for their unwavering support and encouragement throughout our academic journey. Their love and support have been a constant source of motivation and inspiration for us.

Thank you all for your valuable contributions to our project.

Arpita Angad Salunke (248513)

Priyanka Kale (248532)

Abstract

This project analyzes cloud network traffic using unsupervised machine learning techniques to detect anomalies that may signal cyber threats, enabling proactive threat detection and mitigation. For anomaly detection, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is used to identify outliers, which are unusual data points that could indicate attacks. DBSCAN groups normal behaviors while highlighting potential threats. Additionally, Isolation Forest is used to detect outliers based on statistical anomalies in the data.

The effectiveness of these models is evaluated using metrics like silhouette score, which measures how well data points fit within clusters, and WCSS (Within-Cluster Sum of Squares), which assesses cluster tightness. By combining these methods, the project aims to uncover hidden patterns in cloud network traffic, helping security teams detect and mitigate cyber threats before they escalate.

Table Of Contents

Table of Figure:

1 Introduction

1.1 Problem Statement

Unsupervised Anomaly Detection in Cloud Network Traffic

1.2 Scope

The scope of this project is to develop an unsupervised anomaly detection system for cloud network traffic, focusing on identifying unusual patterns that may indicate security threats or system anomalies. It involves data collection, preprocessing, and applying various unsupervised learning algorithms such as Isolation Forest, One-Class SVM, and DBSCAN.

1.3 Aim & Objective

The aim of this project is to develop an unsupervised anomaly detection system to identify abnormal patterns in cloud network traffic, helping to enhance the security and reliability of cloud environments by detecting potential threats

Collect cloud network traffic data and preprocess it for anomaly detection, including cleaning, normalization, and feature extraction. Apply various unsupervised machine learning algorithms, such as Isolation Forest, One-Class SVM, and DBSCAN, for anomaly detection in network traffic. Compare the performance of different algorithms to identify the most effective method for detecting anomalies in cloud network traffic .Use evaluation metrics like silhouette score to assess the accuracy and robustness of the models, even without labeled data. Develop a real-time anomaly detection system capable of identifying unusual traffic patterns and alerting cloud administrators about potential threats.

2 Tools & Techniques

2.1 Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support. Python 3.0, released in 2008, was a major revision that is not completely backward-compatible with earlier versions. Python 2 was discontinued with version 2.7.18 in 2020.

2.2 Numpy

- **NumPy** is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays and matrices, as well as a large collection of mathematical functions to operate on these arrays. NumPy is essential for performing efficient numerical operations on the cloud network traffic dataset, especially when dealing with large amounts of data.
- **Core Functionality**: NumPy introduces the ndarray (N-dimensional array), a highly efficient array structure that allows for fast mathematical operations on large datasets. This is especially useful for anomaly detection in network traffic, where high-performance operations on large datasets are required.
- **Use in Anomaly Detection**: NumPy is used for handling numerical data, performing element-wise operations, and executing mathematical functions such as mean, standard deviation, and correlations. These statistics are often used to define the "normal" behavior of network traffic, making it easier to detect deviations (i.e., anomalies).

2.3 Scikit (Sklearn)

Scikit-learn is one of the most popular Python libraries for machine learning, offering a wide range of algorithms and utilities for supervised and unsupervised learning. For anomaly detection, Scikit-learn provides various models and tools to identify abnormal data points in the dataset.

Key Features of Scikit-learn:

- **Anomaly Detection Algorithms**: Scikit-learn includes several algorithms for anomaly detection, such as:
  - **Isolation Forest**: A tree-based algorithm that isolates anomalies in the feature space.
  - **One-Class SVM**: A Support Vector Machine method for detecting outliers in high-dimensional spaces.
  - **DBSCAN**: A density-based clustering algorithm that can detect anomalies by identifying regions of low density.
- **Preprocessing**: Scikit-learn includes various preprocessing tools for data scaling, normalization, and feature extraction. This is important for preparing the network traffic data before applying anomaly detection algorithms.
- **Model Evaluation**: Scikit-learn also provides tools for evaluating model performance, which can be used to assess the effectiveness of anomaly detection techniques.

Use in Anomaly Detection:

- **Train Anomaly Detection Models**: Algorithms like **Isolation Forest** or **One-Class SVM** are trained on the features extracted from the network traffic data to identify abnormal traffic patterns.
- **Cluster Network Traffic**: Algorithms like **DBSCAN** can group similar traffic patterns and flag outliers as anomalies.
- **Model Tuning and Validation**: Scikit-learn also enables fine-tuning of the model parameters and evaluating its performance (e.g., Silhouette Coefficient) using various metrics.

2.4 Matplotlib

Matplotlib is a powerful plotting library used for data visualization in Python. It allows you to create a variety of static, animated, and interactive visualizations that are essential for interpreting and understanding data.

Key Features of Matplotlib:

- **Versatility**: Matplotlib supports a wide range of plots, including line plots, scatter plots**,** histograms**,** box plots, and heatmaps. These can be used to visualize traffic volume, packet count, and other network features.
- **Customization**: Matplotlib offers extensive customization options (e.g., titles, labels, colors) to make plots informative and easy to understand.
- **Integration with NumPy and Pandas**: Matplotlib works seamlessly with NumPy arrays and Pandas DataFrames , making it easy to create visualizations directly from structured data.

Use in Anomaly Detection :

- **Visualize Data**: Create visual representations of the network traffic data, such as histograms and box plots, to help identify outliers and anomalous behavior in the traffic volume or packet count.
- **Analyze Clusters**: When clustering techniques (e.g., DBSCAN, K-means) are used to identify groups of normal and anomalous traffic, Matplotlib helps visualize the clustering results in a 2D or 3D space.
- **Highlight Anomalies**: Anomalies detected by algorithms like Isolation Forest or One-Class SVM can be visualized using scatter plots, where anomalies are colored differently from normal data points.

2.5 Pandas

Pandas is an open-source Python library designed for data manipulation and analysis. It provides flexible **DataFrame** and **Series** structures for working with structured data. While **NumPy** handles raw numerical operations, **Pandas** offers higher-level data manipulation and **preprocessing tools**.

Key Features of Pandas:

- **DataFrame**: A tabular structure, similar to a spreadsheet, that allows easy manipulation and exploration of network traffic data. It makes it simple to work with labeled data and perform operations like filtering, grouping, and sorting.
- **Data Cleaning**: Pandas supports cleaning tasks such as handling missing values**,** removing duplicates**,** and data normalization, which is essential before applying anomaly detection models.
- **Grouping and Aggregation**: Pandas allows grouping data by various criteria, which can be useful when segmenting network traffic data by different categories (e.g., type of traffic or IP address) and detecting anomalies within each group.
- **Integration with NumPy**: Pandas integrates seamlessly with NumPy, enabling efficient numerical operations on structured data.

Use in Anomaly Detection:

- **Data Preprocessing**: Pandas is used for data cleaning (removing irrelevant features, handling missing values) and transformation (scaling, encoding categorical variables).
- **Feature Selection**: It helps in selecting the relevant features (e.g., packet count, traffic volume, bytes transferred) for anomaly detection.
- **Data Exploration**: Pandas is used to analyze and explore the dataset by summarizing statistics, visualizing data distributions, and checking for inconsistencies, which are crucial steps before applying anomaly detection algorithms.

3 Workflow

Data Collection

- **Source:** Gather raw network traffic data from Stanford internet research data Repository
- **Metrics:** Collect metrics like size, protocol type, source/destination IP addresses, source/destination port , Provider, Region , Metadata, Event ,Vpn, etc.

Data Preprocessing

- **Cleaning:** Remove incomplete, corrupted, or irrelevant data.
- **Normalization/Scaling:** Normalize features (e.g., packet size, traffic volume) to bring them into a similar range (e.g., using Min-Max scaling or Standardization).
- **Feature Selection:**Extract relevant features (e.g., number of requests per minute, average flow duration, connection frequency).

Model Selection

- Choose an unsupervised anomaly detection algorithm (depending on the data type and complexity):
  - **Clustering Algorithms:** DBSCAN (identify outliers based on proximity).
  - **Isolation Forest:** Efficient for high-dimensional data and anomaly isolation.
  - **One-Class SVM:** Learns a boundary around normal data and flags outliers.
  - **PCA (Principal Component Analysis):** Dimensionality reduction to find low-variance anomalies.
  - **Statistical Models:** Z-Score for detecting anomalies based on distribution.

Model Training

- **Input Data:** Feed the preprocessed network traffic data into the selected anomaly detection model.
- **Learn Normal Patterns:** The model identifies "normal" traffic patterns and learns their distribution without labeled data.

- **Threshold Setting:** Define an anomaly threshold (based on the model output, such as an anomaly score). Points above the threshold will be flagged as anomalies.

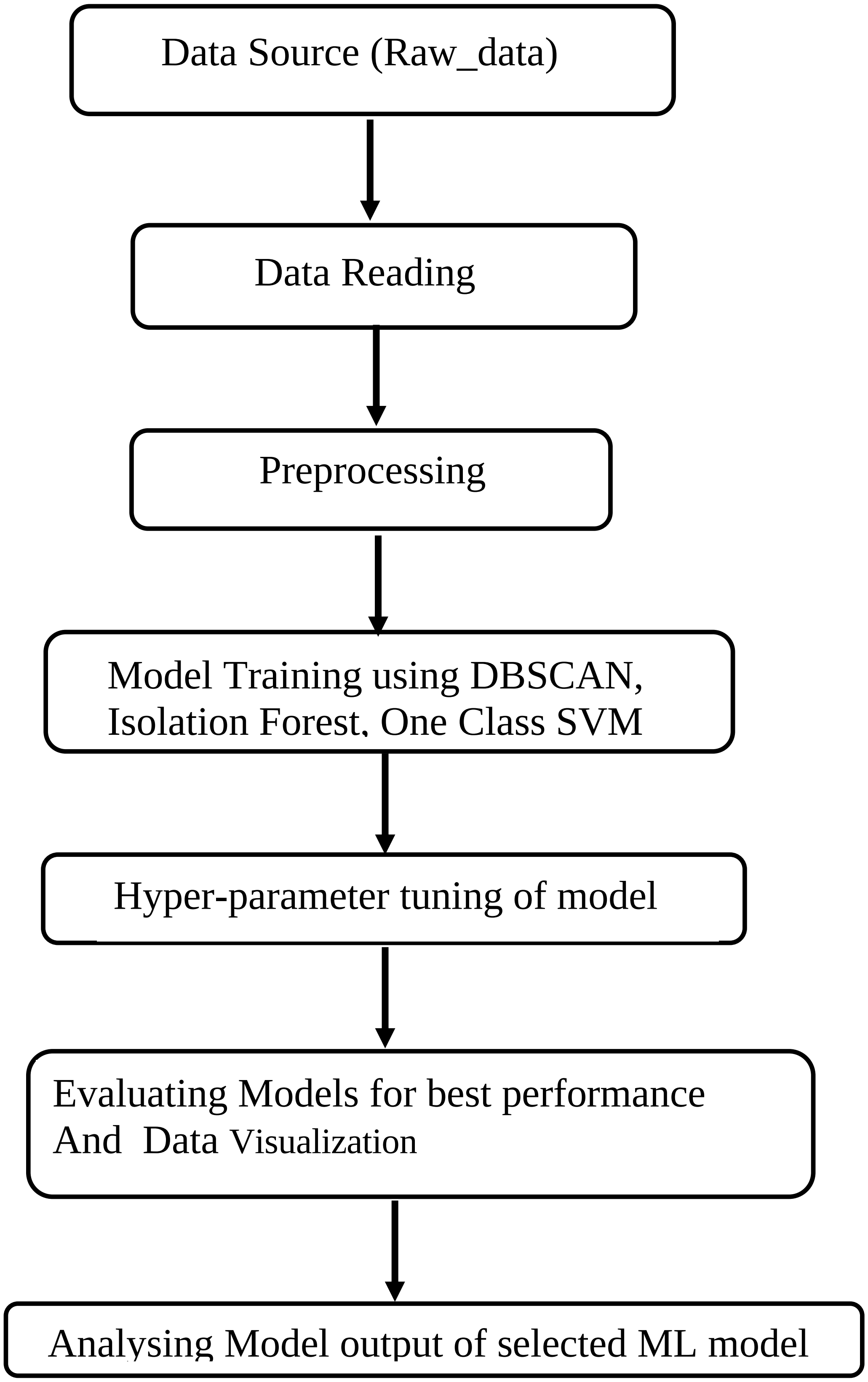
Evaluation Parameter

Silhouette Coefficient :

- **Evaluate clustering quality**: Check how well the algorithm distinguishes between normal and anomalous traffic.
- **Tune the model**: If the Silhouette Coefficient is low, you may want to adjust the number of clusters (e.g., in K-means) or use a different clustering algorithm.
- **Benchmark models**: Compare the effectiveness of different anomaly detection algorithms (clustering-based or otherwise) based on the Silhouette Coefficient score.

Project Workflow

•


Data Source (Raw_data)
Data Reading
Preprocessing
Model Training using DBSCAN, Isolation Forest, One Class SVM
Hyper-parameter tuning of model
Evaluating Models for best performance And Data Visualization
Analysing Model output of selected ML model

Data Source:

Data sourced from the Standford Internet Research Data Repository [2] ,as a zip file containing multiple files in Parquet format. This data may include information such as Source/Destination Ip, Provider, Vpn , Metadata , Event etc.

Data Preprocessing:

In this step, The data was cleaned by handling missing values, Python Dictionaries , JSON Strings , scaling features and selecting relevant metrics for anomaly detection.

Model Training:

Anomaly detection models such as Isolation Forest**,** One-Class SVM, and DBSCAN were chosen for detecting outliers .The models were trained on the cleaned dataset, and their performance was evaluated using metrics like silhouette Coefficient to ensure reliable detection of anomalous traffic..

Hyper-Parameter Tuning:

Hyperparameter tuning is the process of selecting the best set of hyperparameters for a machine learning model to improve its performance. Hyperparameters are the parameters that are set before the learning process begins (unlike model parameters, which are learned during training). Properly tuning these hyperparameters can have a significant impact on the model's accuracy, generalization ability, and efficiency.

- Knee method: It is applied to DBSCAN (Density-Based Spatial Clustering of Applications with Noise) to determine the optimal value of the eps (epsilon) hyperparameter. The eps parameter in DBSCAN defines the radius of the neighborhood around a point. DBSCAN is sensitive to the choice of eps, as it determines how points are grouped into clusters and how outliers (noise) are identified.

- Grid Search: for Isolation Forest can be used to tune hyperparameters and optimize the model’s performance, typically for anomaly detection tasks. The Isolation Forest model in scikit-learn has a few hyperparameters that can significantly impact the model's ability to detect anomalies.

Evaluation & Data Visualization:

- Silhouette Coefficient: evaluates the quality of clustering in unsupervised machine learning. It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1
- The results were visualized through scatter plots and histograms to clearly distinguish between normal and anomalous data points using matplotlib library.

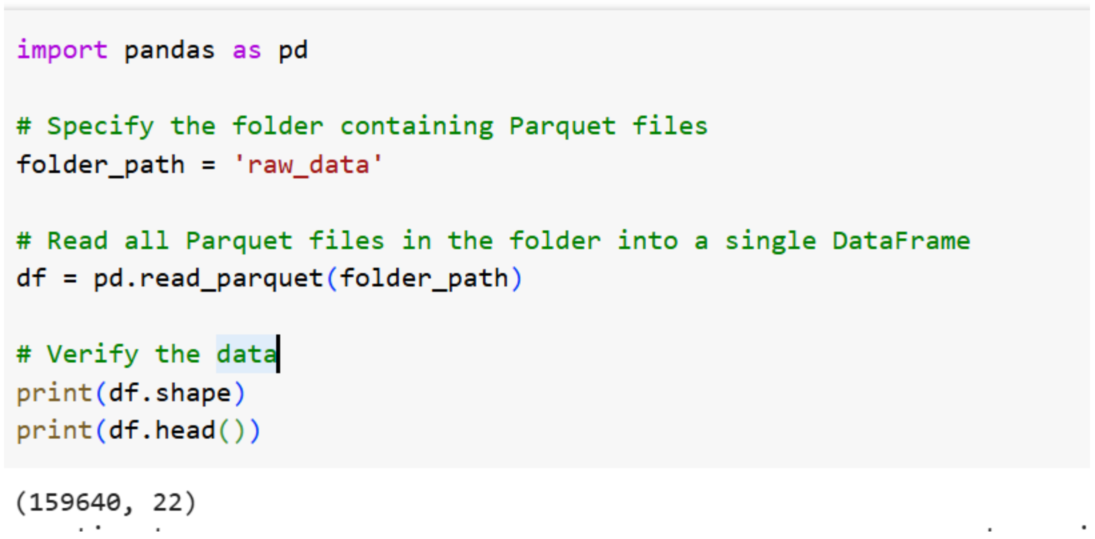
Analysing Model:

DBSCAN	**ISOLATION FOREST**	**ONE CLASS - SVM**
0.549	0.149	**0.605**

One-Class SVM outperforms both **DBSCAN** and **Isolation Forest** in terms of silhouette coefficient, indicating it provides the most meaningful and well-separated anomaly detection. This model has the best-defined boundaries for separating normal and anomalous data points.

4 Figures

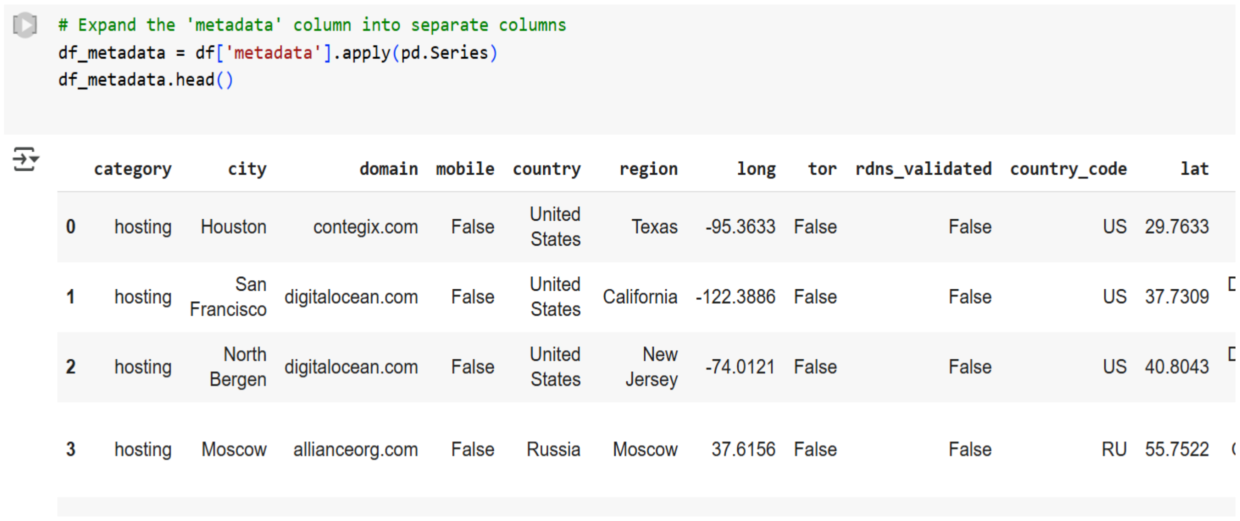
4.1Read all Parquet files in the folder into a single Dataframe



```
import pandas as pd

# Specify the folder containing Parquet files
folder_path = 'raw_data'

# Read all Parquet files in the folder into a single DataFrame
df = pd.read_parquet(folder_path)

# Verify the data
print(df.shape)
print(df.head())
```

```
(159640, 22)
```

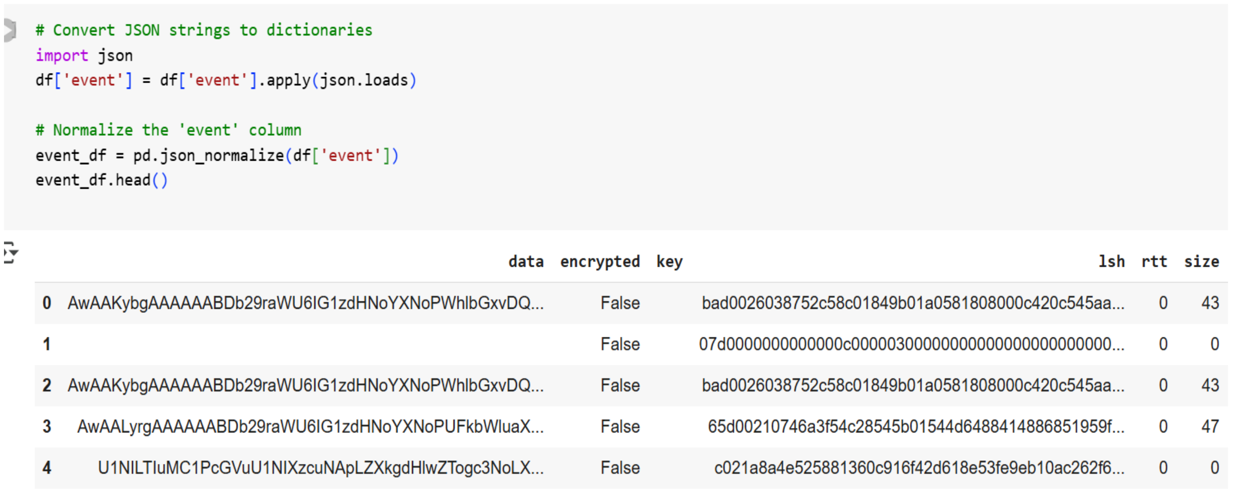
4.2 Handling "Metadata" column(Contains Python Dictionary)



```
# Expand the 'metadata' column into separate columns
df_metadata = df['metadata'].apply(pd.Series)
df_metadata.head()
```

	category	city	domain	mobile	country	region	long	tor	rdns_validated	country_code	lat
0	hosting	Houston	contegix.com	False	United States	Texas	-95.3633	False	False	US	29.7633
1	hosting	San Francisco	digitalocean.com	False	United States	California	-122.3886	False	False	US	37.7309
2	hosting	North Bergen	digitalocean.com	False	United States	New Jersey	-74.0121	False	False	US	40.8043
3	hosting	Moscow	allianceorg.com	False	Russia	Moscow	37.6156	False	False	RU	55.7522

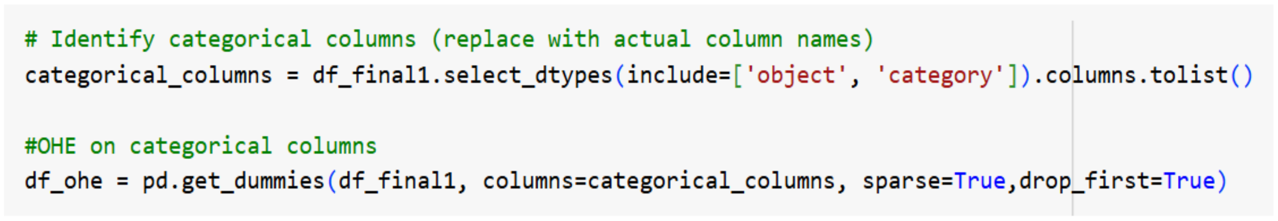
4.3 Handling "Event" Column(Contains JSON Strings)



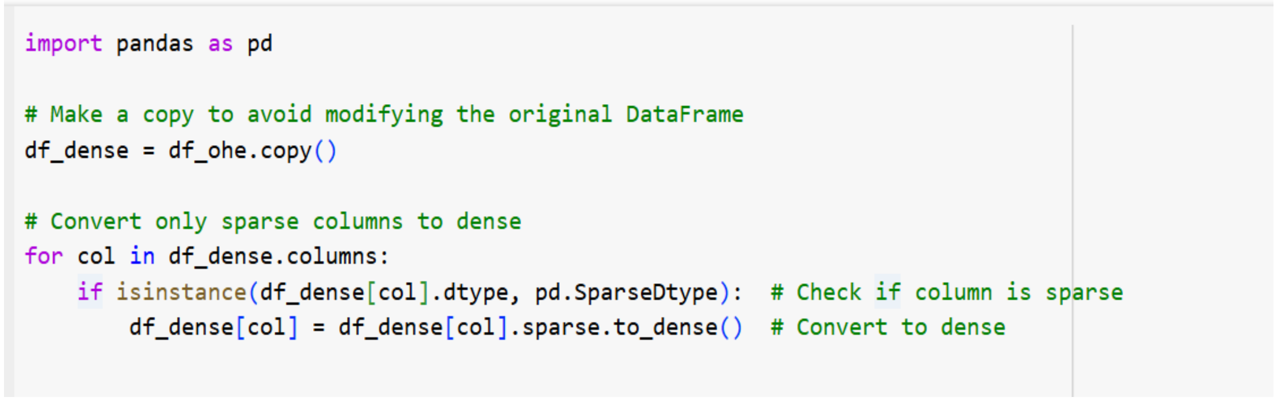
```
# Convert JSON strings to dictionaries
import json
df['event'] = df['event'].apply(json.loads)

# Normalize the 'event' column
event_df = pd.json_normalize(df['event'])
event_df.head()
```

```
                                              data  encrypted  key                                          lsh  rtt  size
0  AwAAKybgAAAAAABDb29raWU6IG1zdHNoYXNoPWhlbGxvDQ...      False  bad0026038752c58c01849b01a0581808000c420c545aa...    0   43
1                                                         False  07d0000000000000c00000300000000000000000000000...    0    0
2  AwAAKybgAAAAAABDb29raWU6IG1zdHNoYXNoPWhlbGxvDQ...      False  bad0026038752c58c01849b01a0581808000c420c545aa...    0   43
3   AwAALyrgAAAAAABDb29raWU6IG1zdHNoYXNoPUFkbWluaX...      False   65d00210746a3f54c28545b01544d6488414886851959f...    0   47
4      U1NILTIuMC1PcGVuU1NIXzcuNApLZXkgdHlwZTogc3NoLX...      False    c021a8a4e525881360c916f42d618e53fe9eb10ac262f6...    0    0
```

4.4 OHE(One Hot Encoding)

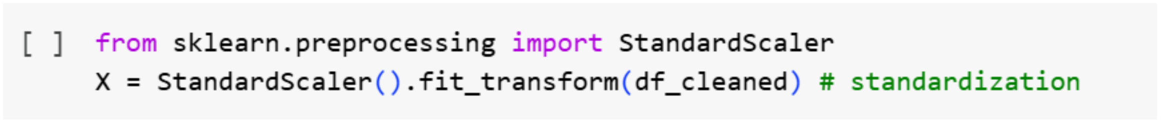


```
# Identify categorical columns (replace with actual column names)
categorical_columns = df_final1.select_dtypes(include=['object', 'category']).columns.tolist()

#OHE on categorical columns
df_ohe = pd.get_dummies(df_final1, columns=categorical_columns, sparse=True,drop_first=True)
```



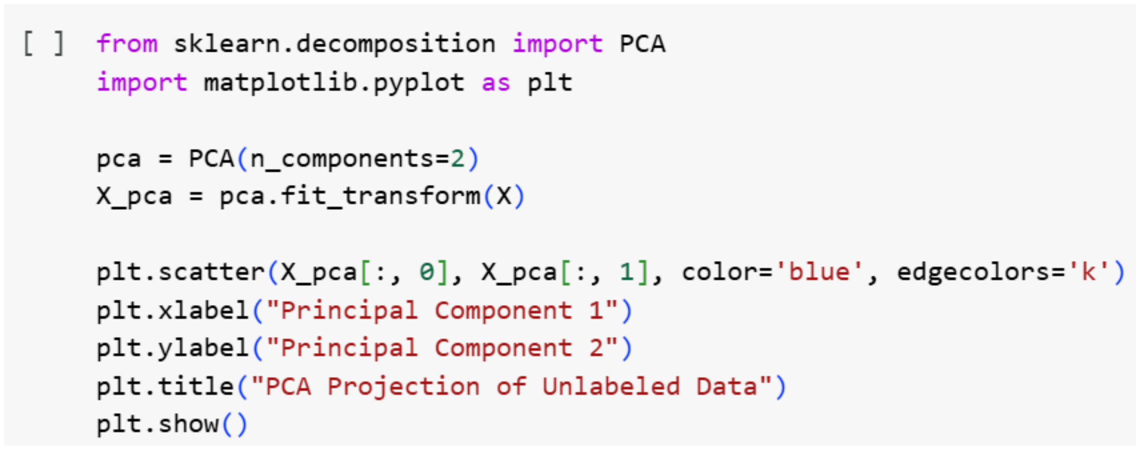
```
import pandas as pd

# Make a copy to avoid modifying the original DataFrame
df_dense = df_ohe.copy()

# Convert only sparse columns to dense
for col in df_dense.columns:
    if isinstance(df_dense[col].dtype, pd.SparseDtype):  # Check if column is sparse
        df_dense[col] = df_dense[col].sparse.to_dense()  # Convert to dense
```

4.5 Standardizing data(Normalizing Data)

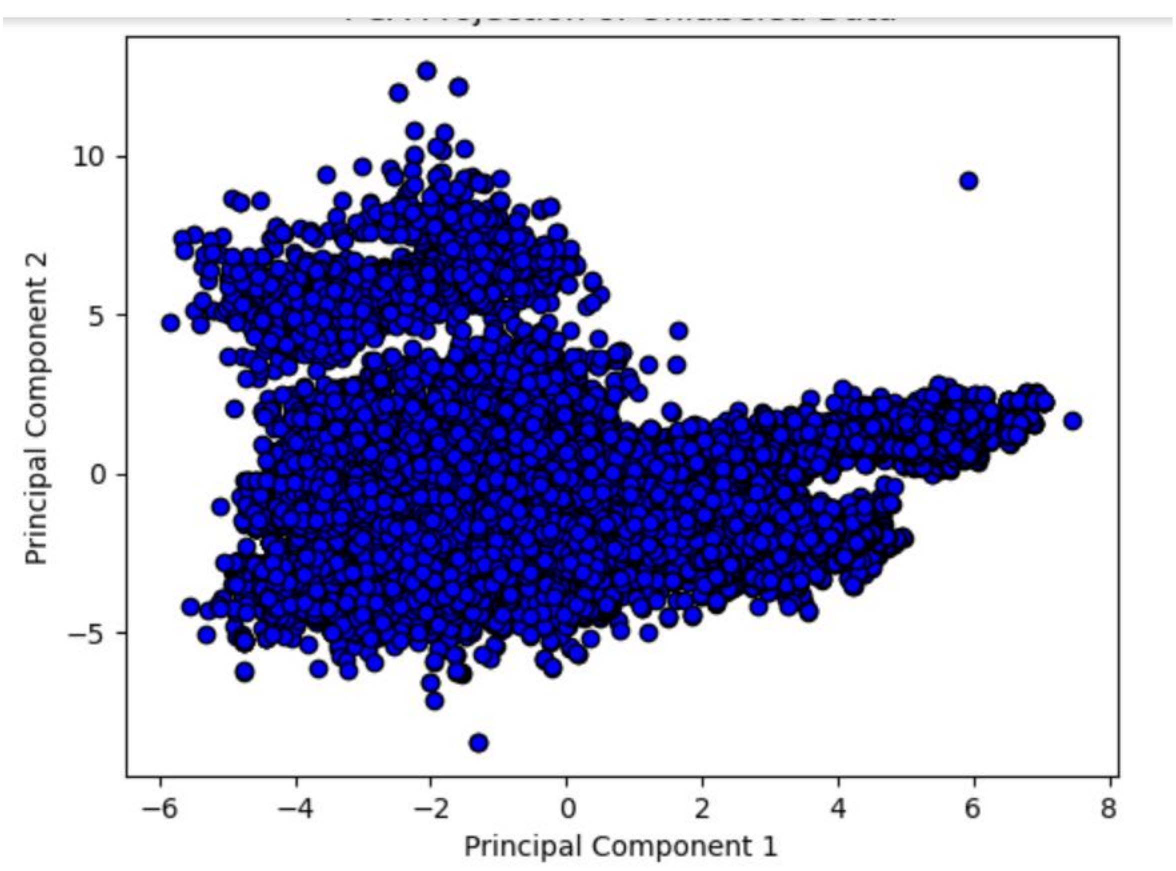


```
[ ]  from sklearn.preprocessing import StandardScaler
     X = StandardScaler().fit_transform(df_cleaned) # standardization
```

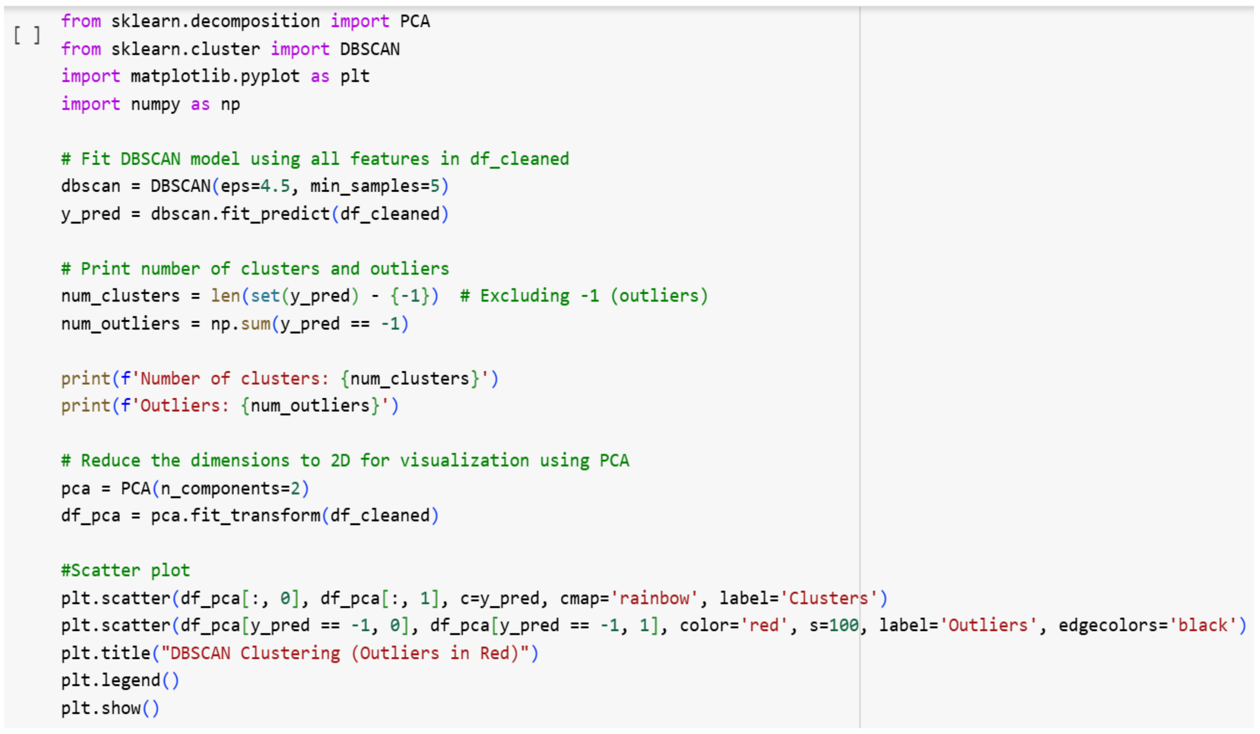
4.6 PCA (To Check Linearity)



```
[ ]  from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt

     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X)

     plt.scatter(X_pca[:, 0], X_pca[:, 1], color='blue', edgecolors='k')
     plt.xlabel("Principal Component 1")
     plt.ylabel("Principal Component 2")
     plt.title("PCA Projection of Unlabeled Data")
     plt.show()
```

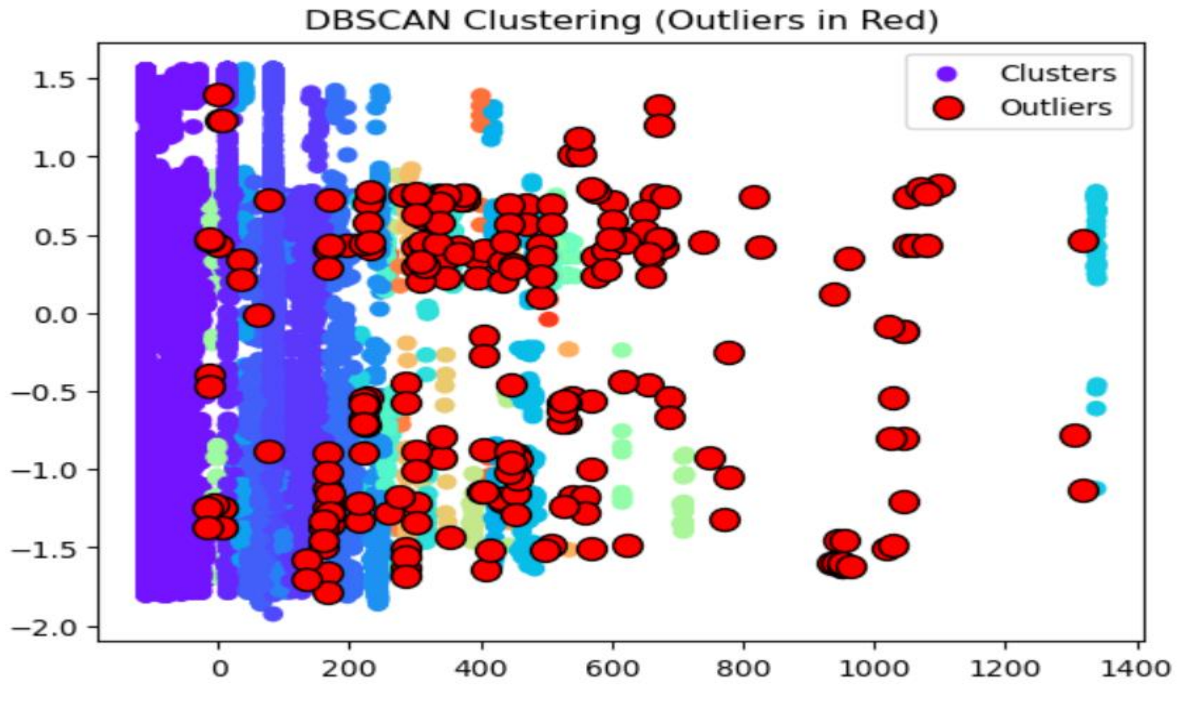


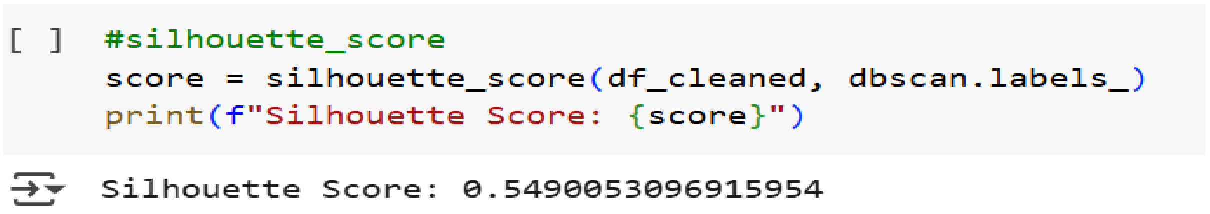
4.7 DBSCAN



```
[ ] from sklearn.decomposition import PCA
    from sklearn.cluster import DBSCAN
    import matplotlib.pyplot as plt
    import numpy as np

    # Fit DBSCAN model using all features in df_cleaned
    dbscan = DBSCAN(eps=4.5, min_samples=5)
    y_pred = dbscan.fit_predict(df_cleaned)

    # Print number of clusters and outliers
    num_clusters = len(set(y_pred) - {-1})  # Excluding -1 (outliers)
    num_outliers = np.sum(y_pred == -1)

    print(f'Number of clusters: {num_clusters}')
    print(f'Outliers: {num_outliers}')

    # Reduce the dimensions to 2D for visualization using PCA
    pca = PCA(n_components=2)
    df_pca = pca.fit_transform(df_cleaned)

    #Scatter plot
    plt.scatter(df_pca[:, 0], df_pca[:, 1], c=y_pred, cmap='rainbow', label='Clusters')
    plt.scatter(df_pca[y_pred == -1, 0], df_pca[y_pred == -1, 1], color='red', s=100, label='Outliers', edgecolors='black')
    plt.title("DBSCAN Clustering (Outliers in Red)")
    plt.legend()
    plt.show()
```

```
Number of clusters: 41
Outliers: 228
```

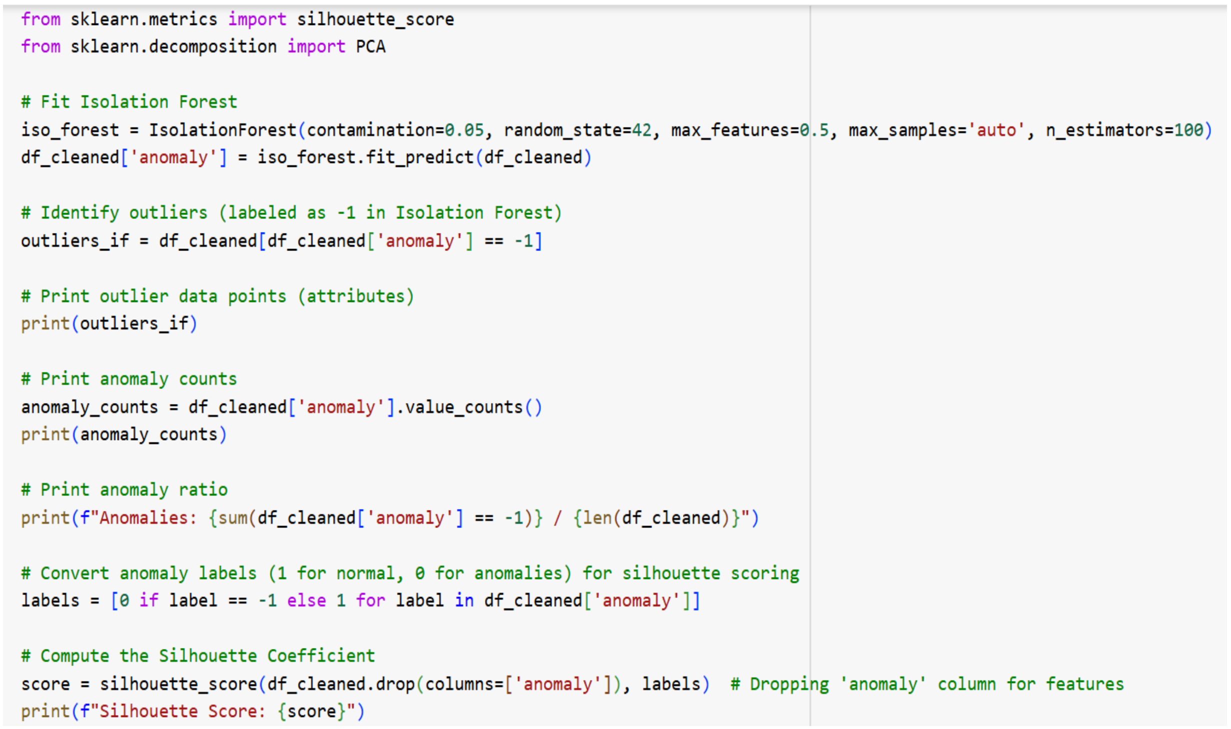




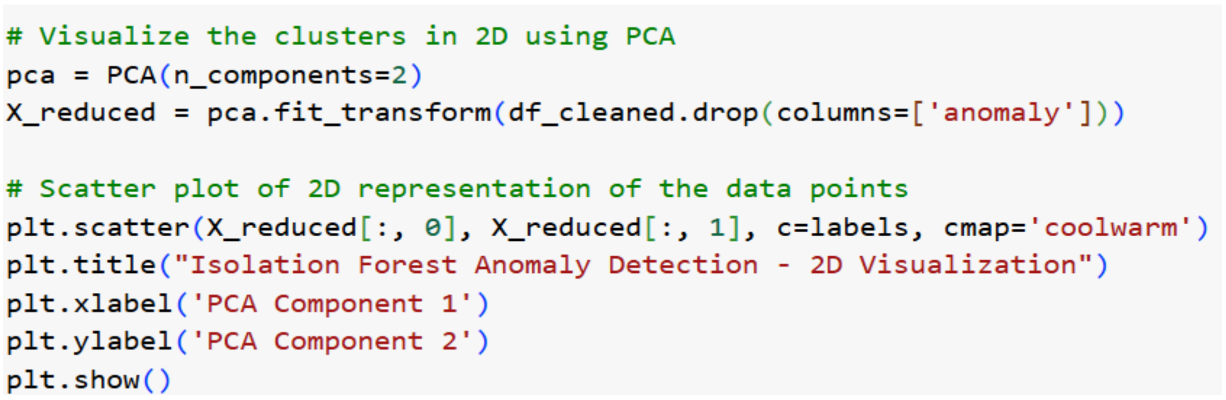
```
[ ]  #silhouette_score
     score = silhouette_score(df_cleaned, dbscan.labels_)
     print(f"Silhouette Score: {score}")
```

```
Silhouette Score: 0.5490053096915954
```

4.7 Isolation Forest Model

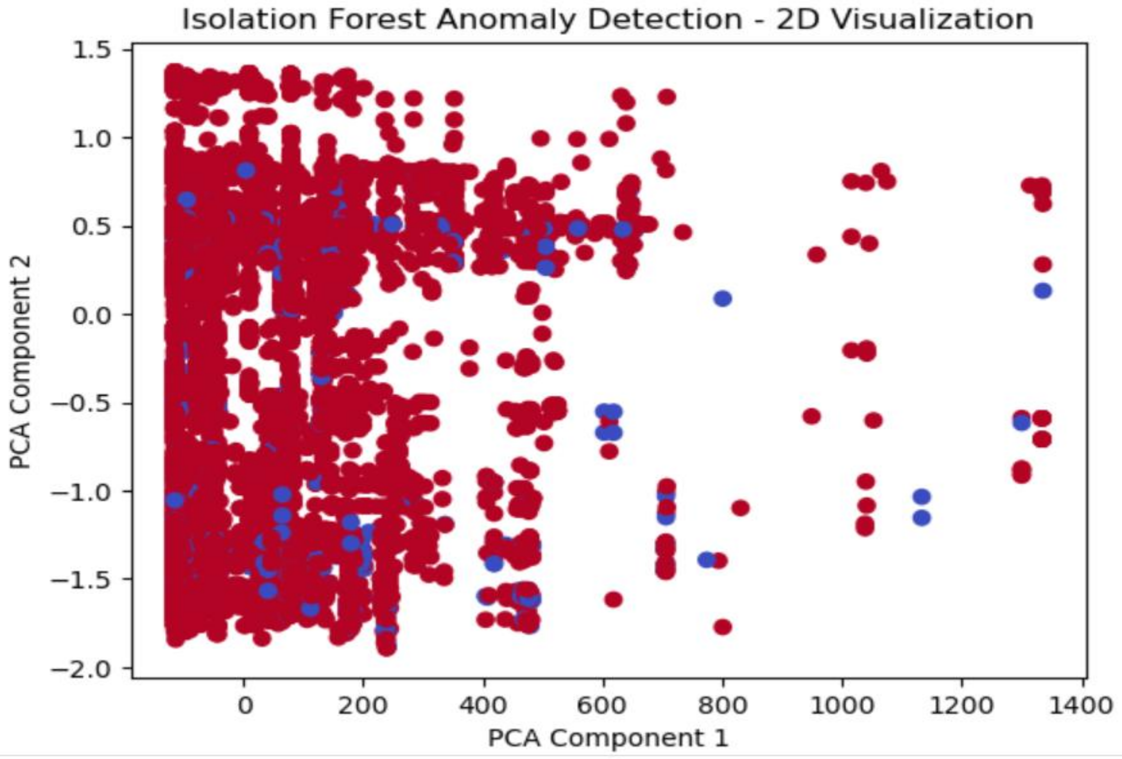


```
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA

# Fit Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42, max_features=0.5, max_samples='auto', n_estimators=100)
df_cleaned['anomaly'] = iso_forest.fit_predict(df_cleaned)

# Identify outliers (labeled as -1 in Isolation Forest)
outliers_if = df_cleaned[df_cleaned['anomaly'] == -1]

# Print outlier data points (attributes)
print(outliers_if)

# Print anomaly counts
anomaly_counts = df_cleaned['anomaly'].value_counts()
print(anomaly_counts)

# Print anomaly ratio
print(f"Anomalies: {sum(df_cleaned['anomaly'] == -1)} / {len(df_cleaned)}")

# Convert anomaly labels (1 for normal, 0 for anomalies) for silhouette scoring
labels = [0 if label == -1 else 1 for label in df_cleaned['anomaly']]

# Compute the Silhouette Coefficient
score = silhouette_score(df_cleaned.drop(columns=['anomaly']), labels)  # Dropping 'anomaly' column for features
print(f"Silhouette Score: {score}")
```

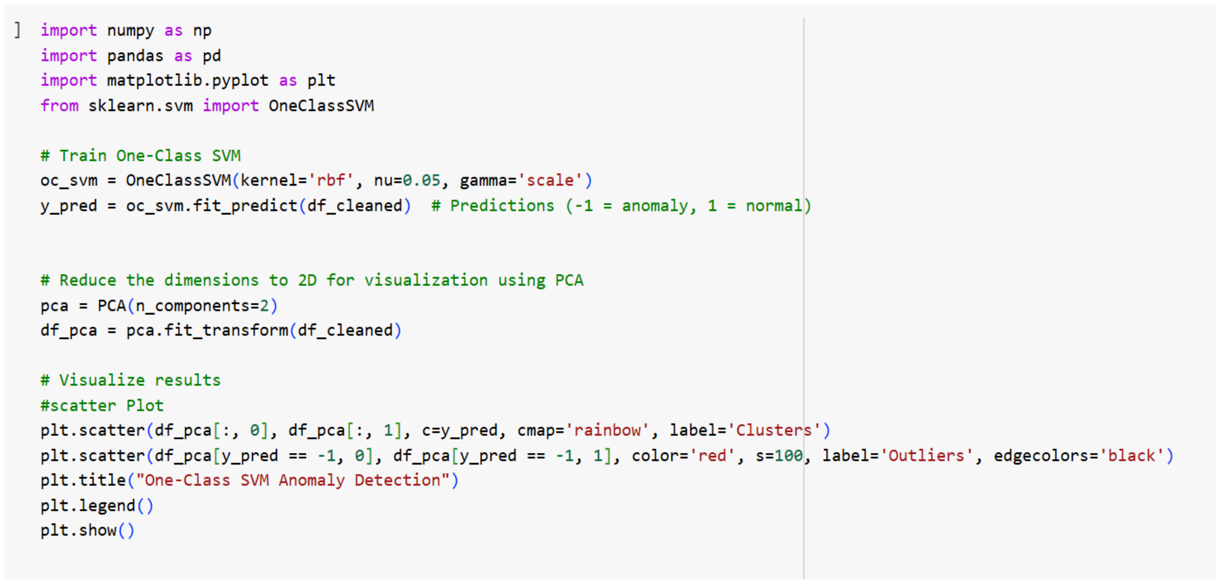


```
# Visualize the clusters in 2D using PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(df_cleaned.drop(columns=['anomaly']))

# Scatter plot of 2D representation of the data points
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=labels, cmap='coolwarm')
plt.title("Isolation Forest Anomaly Detection - 2D Visualization")
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```

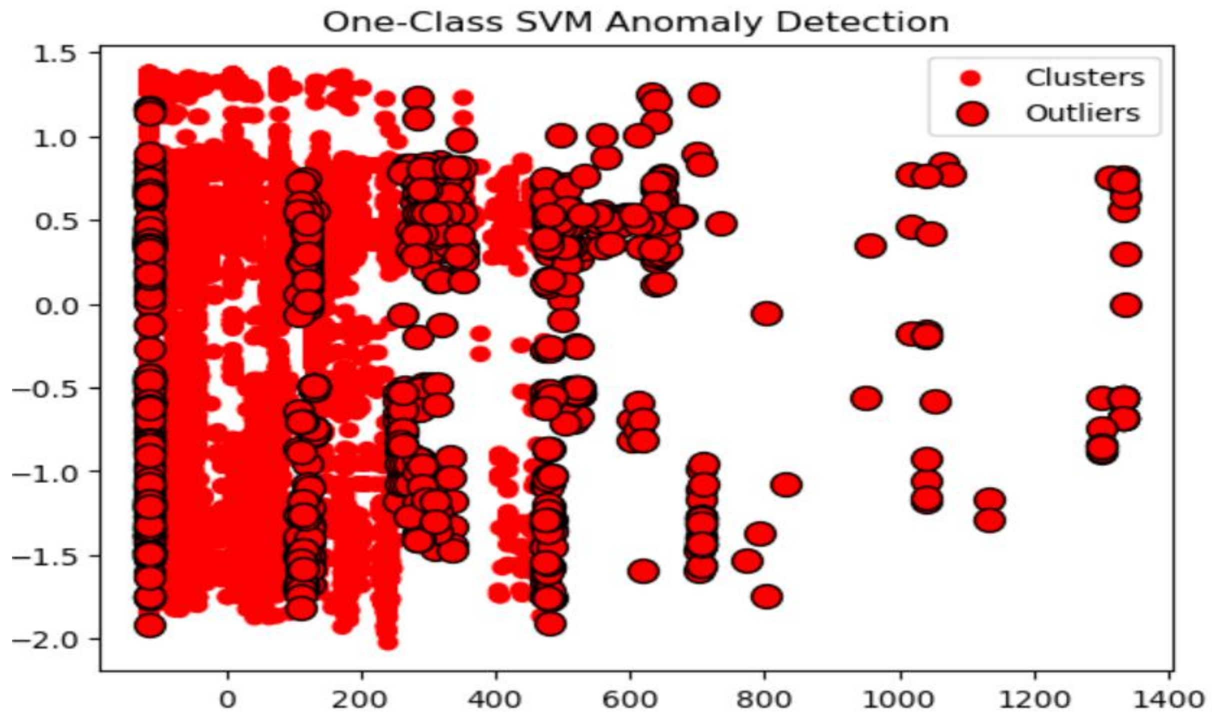
```
Anomalies: 964 / 19310
Silhouette Score: 0.14915304632910834
```



4.8 One-Class SVM



```
] import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
  from sklearn.svm import OneClassSVM

  # Train One-Class SVM
  oc_svm = OneClassSVM(kernel='rbf', nu=0.05, gamma='scale')
  y_pred = oc_svm.fit_predict(df_cleaned)  # Predictions (-1 = anomaly, 1 = normal)


  # Reduce the dimensions to 2D for visualization using PCA
  pca = PCA(n_components=2)
  df_pca = pca.fit_transform(df_cleaned)

  # Visualize results
  #scatter Plot
  plt.scatter(df_pca[:, 0], df_pca[:, 1], c=y_pred, cmap='rainbow', label='Clusters')
  plt.scatter(df_pca[y_pred == -1, 0], df_pca[y_pred == -1, 1], color='red', s=100, label='Outliers', edgecolors='black')
  plt.title("One-Class SVM Anomaly Detection")
  plt.legend()
  plt.show()
```


One-Class SVM Anomaly Detection
Clusters
Outliers
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
0
200
400
600
800
1000
1200
1400

5 Requirements Specification

5.4 Hardware Requirement:

5.4.1 500 GB hard drive (Minimum requirement)
5.4.2 16 GB RAM (Minimum requirement)
5.4.3 PC x64-bit CPU

5.5 Software Requirement:

5.5.1 Windows/Mac/Linux
5.5.2 Python-3.9.1
5.5.3 VS Code/Anaconda/Spyder/Pyspark

6 Future Scope

Exploring Advanced Models: While One-Class SVM performs well, exploring other advanced anomaly detection models like Autoencoders, Isolation Forest with deep learning, or Gaussian Mixture Models (GMM) could lead to even better performance, especially in more complex or high-dimensional data.
Real-Time Anomaly Detection: Moving from batch processing to real-time anomaly detection could enhance the ability to quickly identify and respond to potential security threats or traffic anomalies, improving network performance and security.
Ensemble Methods: Combining the strengths of multiple models (e.g., One-Class SVM, DBSCAN, and Isolation Forest) into an ensemble method might improve overall detection accuracy, allowing the system to adapt to a wider variety of traffic patterns.
Feature Engineering: Further investigation into relevant features such as protocol types, packet sizes, and communication patterns could enhance the detection process. Feature selection or dimensionality reduction techniques like PCA (Principal Component Analysis) may also improve model performance.

7 Conclusion

In conclusion, for anomaly detection in cloud network traffic using DBSCAN, One-Class SVM, and Isolation Forest with the Silhouette Coefficient as the evaluation metric, the performance of each model varies based on data characteristics. DBSCAN works well for dense clusters but is sensitive to parameter choice. One-Class SVM is effective when anomalies are distinct but depends on kernel and regularization settings. Isolation Forest is suitable for high-dimensional data and may perform best in noisy environments. The Silhouette Coefficient helps identify the model with the best separation between normal and anomalous traffic, guiding the selection of the most effective method.

One-Class SVM is the most effective model for anomaly detection in cloud network traffic. It excels in identifying anomalies by defining a decision boundary around normal data, making it particularly suitable when anomalies are rare and distinct. The Silhouette Coefficient confirms that One-Class SVM provides the best separation between normal and anomalous traffic, outperforming DBSCAN and Isolation Forest. This suggests that One-Class SVM is the most appropriate method for detecting anomalies in this specific dataset**.**

8 References

1. https://www.python.org/doc/
2. https://scans.io/study/cloud_watching
3. https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/
4. https://scikit-learn.org/stable/api/index.html
5. https://pandas.pydata.org/docs/reference/index.html#api
6. https://numpy.org/devdocs/reference/index.html#reference
7. https://matplotlib.org/stable/api/pyplot_summary.html
8. https://stackoverflow.com/questions/18837262/convert-python-dict-into-a-dataframe
9. https://stackoverflow.com/questions/50040470/how-to-parse-a-pandas-column-of-json-content-efficiently
10. https://machinelearningmastery.com/model-based-outlier-detection-and-removal-in-python/