Real Time Data Processing and Display using Python

Submitted by

SREESH SOUDARAPU

MANIPAL INSTITUTE OF TECHNOLOGY BENGALURU

JULY-2024

List of contents:

1) Abstract

This project focuses on developing a real-time data processing and display system entirely implemented in Python. The core objective is to process raw Ethernet data received from a server in real-time, perform feature extraction and preprocessing, and apply machine learning algorithms for analysis. Python libraries such as Pandas, NumPy, and scikit-learn are utilized for data manipulation, preprocessing, and model training. The project emphasizes Jupyter Notebook for interactive data visualization. The final output includes a dynamic dashboard displaying processed data and real-time accuracy metrics, showcasing Python's versatility in handling and analysing continuous data streams effectively.

2) Background

a) Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. Often, programmers fall in love with Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective.

b) Jupyter Notebook:

JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.

3) Introduction:

In today’s data-driven landscape, real-time data processing plays a pivotal role in decision-making across various domains. The project “Real-Time Data Processing and Display Using Python” aims to showcase Python’s capabilities in handling streaming data, preprocessing it, applying machine learning algorithms, and dynamically visualizing results. Python, known for its versatility and extensive libraries, serves as the primary tool for implementing these tasks. The project leverages Python's robust libraries such as Pandas, NumPy, Scikit-learn, and Matplotlib to facilitate efficient data manipulation, statistical analysis, and visualization. Additionally, the integration of Jupyter Notebook for interactive development underscores Python's capability to seamlessly integrate with modern data processing pipelines.

The core objectives of this project are twofold: first, to demonstrate the practical application of Python in handling and processing real-time data streams; and second, to showcase its utility in generating actionable insights through machine learning models and visual representations. By implementing these functionalities, the project aims to provide a comprehensive understanding of Python's role in facilitating agile decision-making processes across domains such as Internet of Things (IoT), finance, healthcare, and beyond.

Now the important libraries which are used in the implementation of this project are

1) Pandas: Data manipulation and analysis library.
2) NumPy: Fundamental package for scientific computing with Python, used for array operations.
3) Matplotlib: Comprehensive library for creating static, animated, and interactive visualizations.
4) Scikit-learn: Simple and efficient tools for data mining and data analysis, including machine learning algorithms.
5) Socket: Low-level networking interface used for communication between processes across a network.
6) IPython.display: Utilities for displaying rich content in the IPython environment, useful for dynamic output in notebooks.
7) Plotly: Interactive graphing library that provides a wide range of graphs and charts.
8) PySerial: Python serial port access library.
9) Seaborn: Data visualization library based on Matplotlib, providing a high-level interface for drawing attractive statistical graphics.
10) Standard Python Libraries: Python's built-in modules for handling dates and times, binary data, and other standard tasks.

4) Methodology:

a) Setup a Server:

Use Python's socket library to create a TCP server. The server will generate raw Ethernet data packets and send them to the client.

b) **Data Packet Structure**: Each Ethernet packet will have a destination MAC address, source MAC address, Ethertype, and payload.

c) **Client Setup**: Use Python's socket library to create a TCP client. Connect to the server to receive raw Ethernet data packets.

d) **Feature Extraction**: Extract key features from each Ethernet packet: destination MAC, source MAC, Ethertype, and payload length. Store these features in a structured format for further processing.

e) **Encoding MAC Addresses**: Use LabelEncoder from sklearn. preprocessing to encode destination and source MAC addresses into numerical values.

f) **Feature Scaling**: Use StandardScaler from sklearn.preprocessing to scale features for better performance of machine learning algorithms

g) **Tracking Seen MAC Addresses:** Maintain sets of seen destination and source MAC addresses to update the encoders dynamically with new addresses

h) **Model Selection**: The Machine learning algorithms used in the project are

a) SDGC Classifier (Stochastic Gradient Descent Classifier): SGD is an optimization technique used to find the minimum of a function (in this case, the loss function of the classifier). It iteratively updates the parameters (weights) of the model in the opposite direction of the gradient of the loss function with respect to the weights.

Features of SGDC

- Loss Function: Determines the objective to minimize during training. Common choices include:
- Hinge Loss: Used for Support Vector Machines (SVMs) and linear SVMs.
- Log Loss: Used for logistic regression.
- Modified Huber Loss: Robust to outliers.
- Penalty: Regularization term added to the loss function to penalize large coefficients

b) Random Forest Classifier: Random Forest Classifier is an ensemble learning method for classification tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Features of Random Forest

- Decision tree
- Randomization
- Aggregation

c) Passive aggressive classifier: The Passive-Aggressive Classifier (PA Classifier) is a type of online learning algorithm that is well-suited for large-scale and streaming data scenarios where data arrives sequentially. It belongs to the family of algorithms known as online learning or incremental learning algorithms.

Features of Passive Aggressive Classifier

- Aggressiveness control
- Margin based approach
- Mistake-driven updates
- Online learning

d) Perceptron Classifier: Certainly! The Perceptron algorithm is one of the foundational algorithms in the field of machine learning, specifically in the realm of supervised learning for binary classification tasks. It was developed by Frank Rosenblatt in the late 1950s and is a type of linear classifier.

e) Naïve Bayes Classifier: The Naive Bayes classifier is a probabilistic machine learning model based on Bayes' theorem with strong independence assumptions between the features. It is widely used for classification tasks, especially in natural language processing (NLP) and spam filtering

f) SVM Classifier (Support Vector Machine): The Support Vector Machine (SVM) classifier is a powerful and versatile supervised machine learning algorithm that can perform linear and non-linear classification, regression, and outlier detection tasks. It is particularly effective in high-dimensional spaces and when the number of dimensions (features) is greater than the number of samples

i) **Training and Testing**: Split the processed data into training and test sets. Train the model incrementally as new data arrives. Generate simulated binary target variables for training and testing purposes.

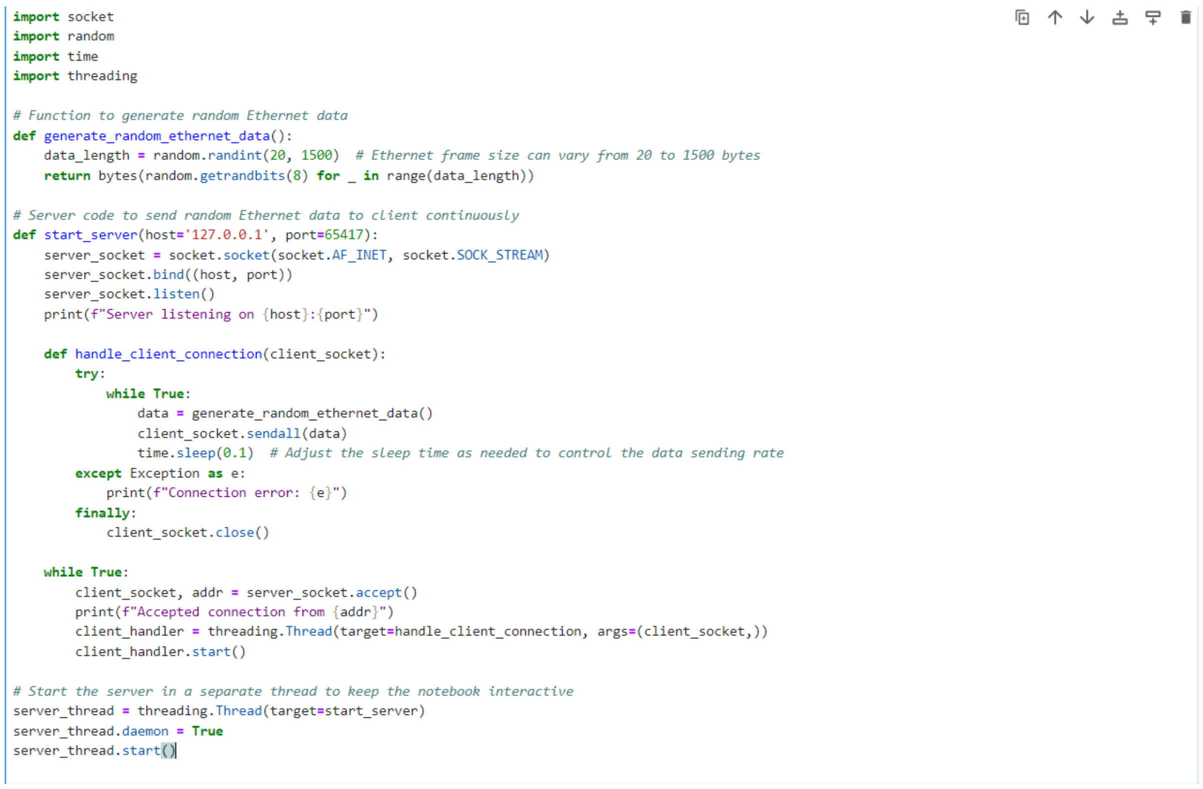
j) **Accuracy Calculation**: calculate the accuracy of the model on the test set.

k) **Display Features:** Use pandas DataFrame to display the processed features in a tabular format.

l) **Plot Accuracy vs Time**: Use matplotlib to plot accuracy over time, creating a dynamic, stock market-like visualization. Customize the plot with labels, grids and references.

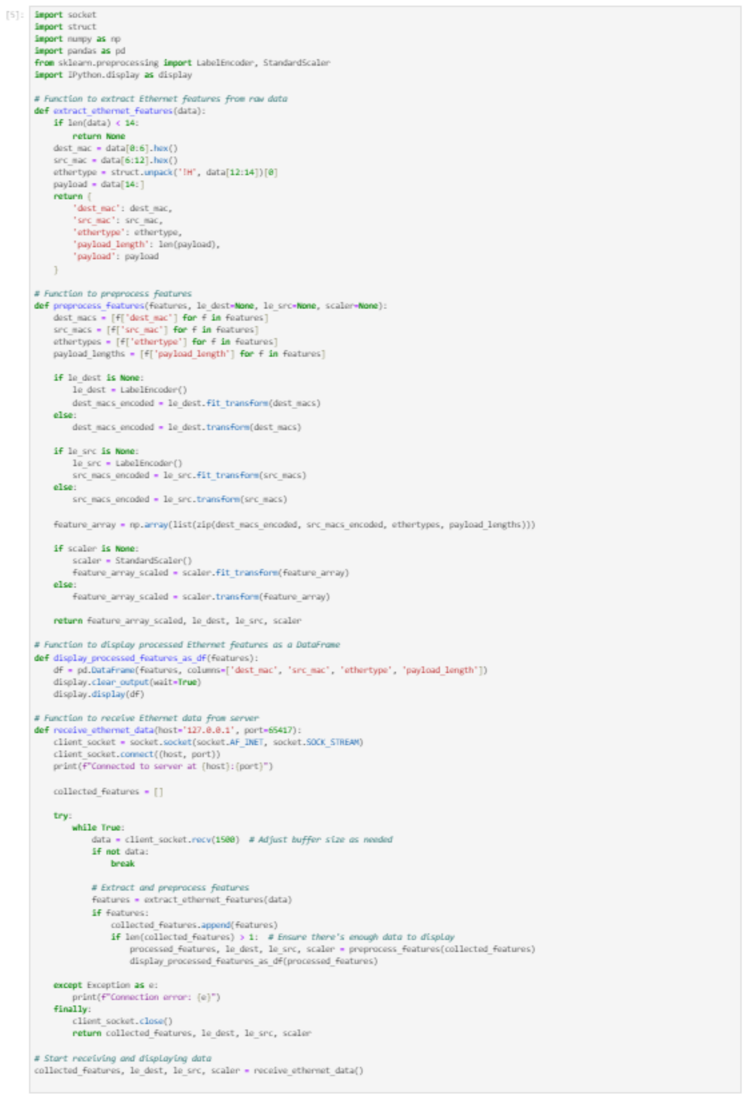
5) CODE SNIPETS OF THE PROJECT:

a) **Server code**:



```python
import socket
import random
import time
import threading

# Function to generate random Ethernet data
def generate_random_ethernet_data():
    data_length = random.randint(20, 1500)  # Ethernet frame size can vary from 20 to 1500 bytes
    return bytes(random.getrandbits(8) for _ in range(data_length))

# Server code to send random Ethernet data to client continuously
def start_server(host='127.0.0.1', port=65417):
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen()
    print(f"Server listening on {host}:{port}")

    def handle_client_connection(client_socket):
        try:
            while True:
                data = generate_random_ethernet_data()
                client_socket.sendall(data)
                time.sleep(0.1)  # Adjust the sleep time as needed to control the data sending rate
        except Exception as e:
            print(f"Connection error: {e}")
        finally:
            client_socket.close()

    while True:
        client_socket, addr = server_socket.accept()
        print(f"Accepted connection from {addr}")
        client_handler = threading.Thread(target=handle_client_connection, args=(client_socket,))
        client_handler.start()

# Start the server in a separate thread to keep the notebook interactive
server_thread = threading.Thread(target=start_server)
server_thread.daemon = True
server_thread.start()
```

b) Client code:

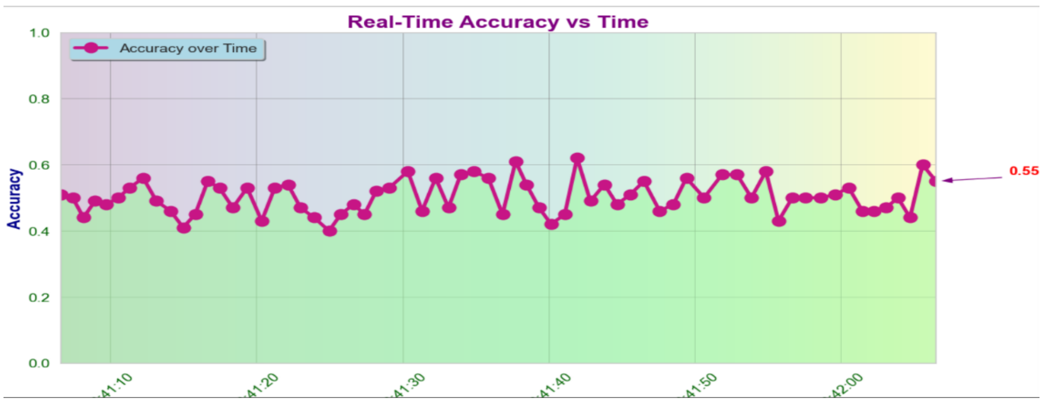


```
[5]: import socket
     import struct
     import numpy as np
     import pandas as pd
     from sklearn.preprocessing import LabelEncoder, StandardScaler
     import IPython.display as display

     # Function to extract Ethernet features from raw data
     def extract_ethernet_features(data):
         if len(data) < 14:
             return None
         dest_mac = data[0:6].hex()
         src_mac = data[6:12].hex()
         ethertype = struct.unpack('!H', data[12:14])[0]
         payload = data[14:]
         return {
             'dest_mac': dest_mac,
             'src_mac': src_mac,
             'ethertype': ethertype,
             'payload_length': len(payload),
             'payload': payload
         }

     # Function to preprocess features
     def preprocess_features(features, le_dest=None, le_src=None, scaler=None):
         dest_macs = [f['dest_mac'] for f in features]
         src_macs = [f['src_mac'] for f in features]
         ethertypes = [f['ethertype'] for f in features]
         payload_lengths = [f['payload_length'] for f in features]

         if le_dest is None:
             le_dest = LabelEncoder()
             dest_macs_encoded = le_dest.fit_transform(dest_macs)
         else:
             dest_macs_encoded = le_dest.transform(dest_macs)

         if le_src is None:
             le_src = LabelEncoder()
             src_macs_encoded = le_src.fit_transform(src_macs)
         else:
             src_macs_encoded = le_src.transform(src_macs)

         feature_array = np.array(list(zip(dest_macs_encoded, src_macs_encoded, ethertypes, payload_lengths)))

         if scaler is None:
             scaler = StandardScaler()
             feature_array_scaled = scaler.fit_transform(feature_array)
         else:
             feature_array_scaled = scaler.transform(feature_array)

         return feature_array_scaled, le_dest, le_src, scaler

     # Function to display processed Ethernet features as a DataFrame
     def display_processed_features_as_df(features):
         df = pd.DataFrame(features, columns=['dest_mac', 'src_mac', 'ethertype', 'payload_length'])
         display.clear_output(wait=True)
         display.display(df)

     # Function to receive Ethernet data from server
     def receive_ethernet_data(host='127.0.0.1', port=65417):
         client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
         client_socket.connect((host, port))
         print(f"Connected to server at {host}:{port}")

         collected_features = []

         try:
             while True:
                 data = client_socket.recv(1500)  # Adjust buffer size as needed
                 if not data:
                     break

                 # Extract and preprocess features
                 features = extract_ethernet_features(data)
                 if features:
                     collected_features.append(features)
                     if len(collected_features) > 1:  # Ensure there's enough data to display
                         processed_features, le_dest, le_src, scaler = preprocess_features(collected_features)
                         display_processed_features_as_df(processed_features)

         except Exception as e:
             print(f"Connection error: {e}")
         finally:
             client_socket.close()
             return collected_features, le_dest, le_src, scaler

     # Start receiving and displaying data
     collected_features, le_dest, le_src, scaler = receive_ethernet_data()
```

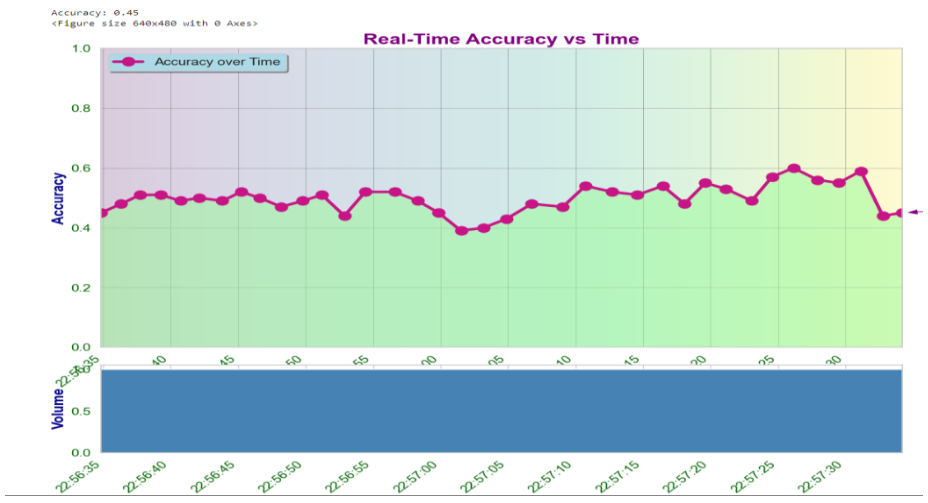
6) Output of the Project:

The output of the project is the real time graphs for different Machine learning Algorithms

a) SDGC Classifier:

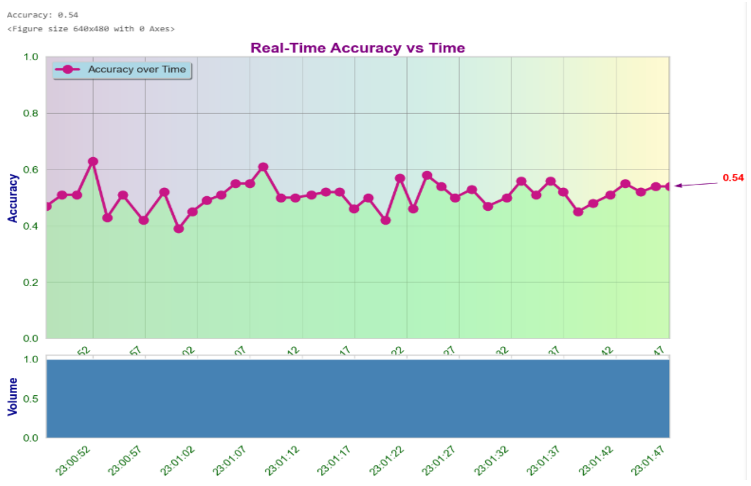


b) Random Forest Classifier:



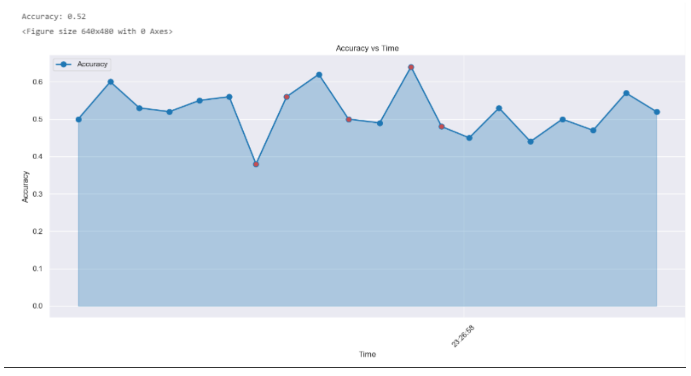
```
Accuracy: 0.45
<Figure size 640x480 with 0 Axes>
```

c) Passive Aggressive Classifier:



```
Accuracy: 0.54
<Figure size 640x480 with 0 Axes>
```

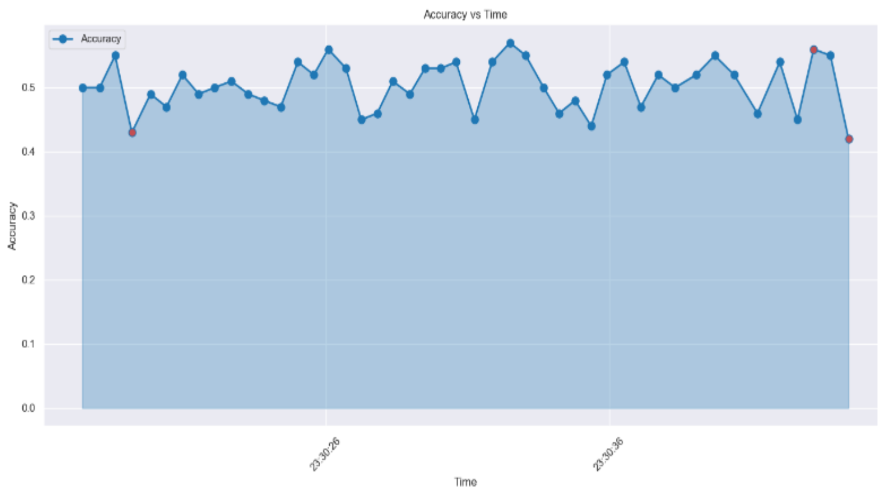
d) Perceptron Classifier:



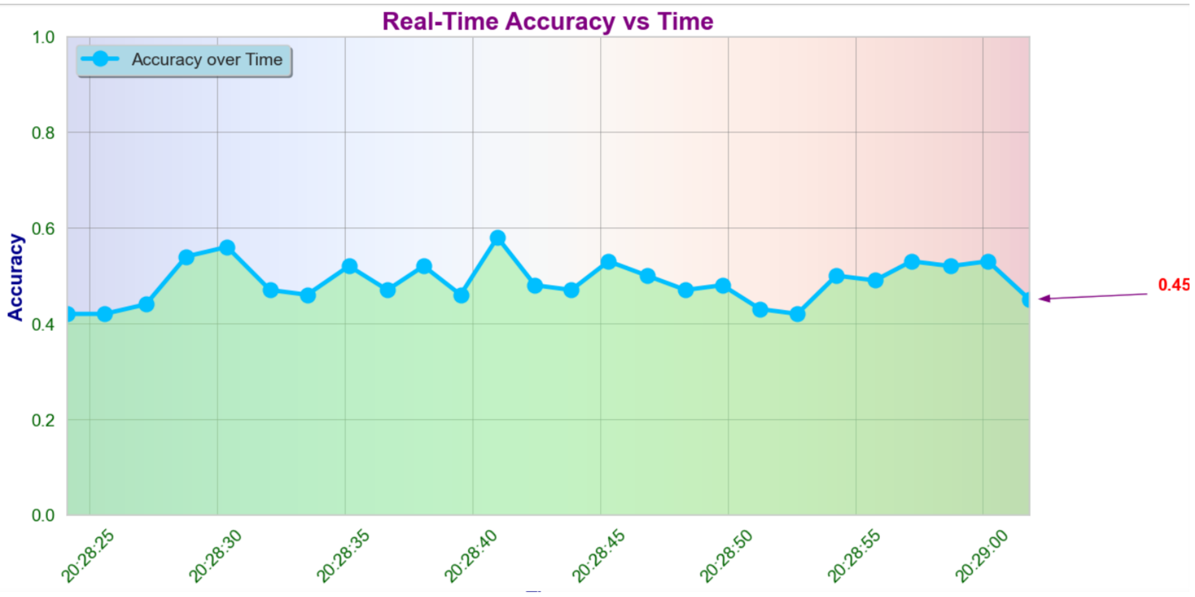
```
Accuracy: 0.52
<Figure size 640x480 with 0 Axes>
```

e) Naïve Bayes Classifier:

```
Accuracy: 0.42
<Figure size 640x480 with 0 Axes>
```



f) Support Vector Machine classifier:



7) Significance:

Real-time data processing plays a pivotal role in various domains, including finance, healthcare, and network security. Its significance lies in providing immediate insights and facilitating rapid decision-making. For instance, in financial services, real-time detection of fraudulent transactions enhances security measures. Beyond security, businesses benefit from dynamic adaptation to market trends and consumer behaviors, leading to more effective strategies and heightened customer satisfaction. Integrating advanced technologies and machine learning techniques ensures scalability and flexibility, bridging the gap between theory and practical application. Moreover, real-time analytics enhance network security by promptly detecting anomalies and potential threats, while interactive visualizations improve user engagement and customer service. Economically, real-time processing reduces storage costs and minimizes latency, providing organizations with a competitive edge.

8) Conclusion:

this project successfully demonstrates the powerful potential of real-time data processing and visualization to drive rapid and informed decision-making across various domains. By integrating machine learning techniques with efficient data handling and processing capabilities, the project showcases how immediate insights can enhance operational efficiency, security, and responsiveness. The application of these technologies not only enables quick detection and response to anomalies and trends but also provides a scalable solution to handle the increasing volume of data in a dynamic environment. Through the development and implementation of this project, we have highlighted the critical importance of real-time analytics in today's fast-paced world, underscoring its significance in improving outcomes, enhancing customer satisfaction, and maintaining competitive advantage. The hands-on experience gained and the practical applications explored serve as a valuable contribution to both educational and professional fields, paving the way for future innovations in real-time data processing and analytics.

9) References

- https://www.python.org/
- https://jupyter.org/
- https://ieeexplore.ieee.org/abstract/document/5492955

- https://softlandia.fi/en/blog/real-time-data-processing-with-python-technology-evaluation
- https://www.kdnuggets.com/python-in-finance-real-time-data-streaming-within-jupyter-notebook
- https://www.geeksforgeeks.org/data-processing-with-pandas/
- https://quix.io/docs/kb/what-is-quix.html
- Sauter, N. K., Hattne, J., Grosse-Kunstleve, R. W., & Echols, N. (2013). New Python-based methods for data processing. *Acta Crystallographica Section D: Biological Crystallography*, *69*(7), 1274-1282.
- Unpingco, J., 2014. Python for Signal Processing. *Python for Signal Processing. https://doi. org/10.1007/978-3-319-01342-8*.
- McKinney, Wes. *Python for data analysis*. " O'Reilly Media, Inc.", 2022.