Project Report

On

Bank Transaction Fraud Detection

(Real-time Anomaly Detection and Response System with Kafka, Spark, and LLM Chatbot Integration)

Submitted in partial fulfilment of the Requirements for final project in semester 4

Of

Masters in Data Science

Submitted by:
Sowjanya

CHANDIGARH UNIVERSITY GHARUAN, MOHALI, PUNJAB, INDIA-140413 2019

INDEX

ACKNOWLEDGEMENT

I express my heartfelt gratitude to all those who have been instrumental in the successful completion of my master's degree Data Science Project at Chandigarh University, with a focus on *Bank Transaction Fraud Detection.*

First and foremost, I would like to extend my sincere appreciation to my project guide, ***Mubarak Abdulla - Chief Architect at skan.ai***, for his invaluable guidance, unwavering support, and insightful feedback throughout the duration of this project. His expertise and mentorship have been pivotal in shaping the direction and execution of my research.

I would also like to thank the faculty members of *the Department of Data Science at Chandigarh University* for providing a conducive academic environment and resources that were essential for the successful completion of my project. Their encouragement and constructive criticism played a crucial role in refining my ideas and methodologies.

Special thanks go to ***Tina Sharma and Amanjot Singh*** for being the best team and making our collective efforts count in the success of this project.

Lastly, I express my gratitude to Chandigarh University for providing the platform and resources necessary for pursuing advanced studies in data science. This project has been a significant learning experience, and I am thankful to everyone who has contributed directly or indirectly to its successful completion.

ABSTRACT

For years, fraudulent activities have posed significant challenges across industries like banking, healthcare, and insurance. With the surge in online transactions facilitated by various payment platforms like credit/debit cards, PhonePe, Gpay, and Paytm, the incidence of fraud has escalated. Criminals have honed their skills to exploit loopholes in systems, making it increasingly difficult to ensure secure authentication and protect customers from fraudulent activities. Thus, the development of fraud detection algorithms has become crucial.

The ongoing global shift towards digitization has fuelled a rise in cashless transactions, particularly through credit cards. However, this trend has also exacerbated fraudulent activities, resulting in substantial losses for financial institutions. Consequently, there's a pressing need to distinguish between fraudulent and legitimate transactions. In this context,

we provide a comprehensive overview of various methodologies employed in detecting fraudulent bank transactions.

We employ diverse machine learning algorithms, including logistic regression, Naïve Bayes, and random forest with ensemble classifiers using boosting techniques, to analyse an imbalanced dataset. Our review encompasses existing and proposed models for fraud detection in bank transactions, accompanied by a comparative analysis of these techniques. Additionally, we assess the performance of different classification models based on quantitative metrics such as accuracy, precision, recall, F1 score, support, and confusion matrix.

CHAPTER-1

INTRODUCTION:

1.1 Overview

Fraud, which is a multi-million-dollar business worldwide, has become a significant problem nowadays.

The development of new technologies has resulted in the creation of new ways to commit fraud. The creation of a new information system, in addition to its advantages and benefits,

can create additional opportunities for criminals. In addition to detecting fraud and fraud in organizations, fraud detection techniques and intelligence tools aim to predict future behaviour and reduce the risk of fraud by understanding user and customer behaviour. Banks and financial institutions are striving to speed up the process of identifying fraudsters due to the high costs of fraud.

It is against the law to commit fraud in electronic banking and electronic money transfer. Fraudsters are encouraged and incited to enter this field due to the large volume of monetary and financial transactions and transfers conducted on the Internet and electronic platform. To provide electronic services, it is crucial and important to recognize the identity of individuals.

The purpose of this project was to uncover fraud in the banking system and develop an optimal method for analysing information from the One of the State banks using intelligence tools and model evaluation. The focus is to review the latest fraud detection methodologies and tools. The propose of this is to develop the framework for the fraud detection mechanism, as well as the use of the Python language to implement the developed model.

The simple way to detect this type of fraud is to analyse each transaction and to figure out any variation to the “usual” patterns. As the data sets are not available and the results are not disclosed to the public. The fraud cases should be detected from the available data sets known as the logged data and user behaviour. At present, fraud detection has been implemented by a number of methods such as data mining, statistics, and artificial intelligence.

1.2 Problem Statement

Surge in online banking and cashless transactions has led to a significant increase in fraudulent activities, posing substantial challenges to financial institutions. Traditional rule-based fraud detection systems are often inadequate in identifying sophisticated fraudulent transactions, resulting in substantial financial losses and compromised customer trust. Therefore, there is an urgent need to develop robust machine learning-based algorithms

capable of accurately detecting fraudulent bank transactions in real-time. This project aims to address this challenge by implementing various machine learning techniques on imbalanced datasets, analysing their performance, and identifying the most effective approach for mitigating bank fraud.

1.3 Theoretical Background:

Fraud arises in the financial industry via numerous channels, such as credit cards, e-commerce, phone banking, checks, and online banking. Juniper Research (2020) reports that e-commerce, airline ticketing, money transfer, and banking services will cumulatively lose over $ 200 billion due to online payment fraud between 2020 and 2024. The increased sophistication of fraud attempts and the increasing number of attack vectors have driven these results. We focus on online and mobile payment channels and identity theft fraud (i.e., stealing an individual's personal information to conduct fraud). The aim is to identify external

fraudsters who intend to initiate payments in their interests. As fraudsters gain access to the payment systems as if they were the owners of the accounts, they cannot be identified based on the account access process. However, the fraudster behaves differently during a payment transaction than the account owner and/or the payment has unusual characteristics, such as an unusually high payment amount or transfer to an account in a jurisdiction that does not fit the life context and payment behavior of the customer. The assumption is that algorithms can detect anomalies in behavior during payment transactions.

Types of Internet Fraud

- Email Phishing
- Payment Fraud
- ID Document Forgery
- Identity Theft

Email Phishing: This is a fraud or cybercrime wherein attackers send fake sites and messages to users via email. These emails are seemingly legit and authentic that anyone can misjudge them and enter the vulnerable data that puts them at risk. The best way to prevent email phishing is to avoid entering vulnerable data in these emails until you verify their credentials. And the best way is to ignore these emails or messages that flash on your screen. Traditional methods for phishing involve the use of filters. These filters are primarily of two types, authentication protection, and network-level protection. Authentication protection is through email verification. Network-level protection is through three filters; whitelist, blacklist, and pattern matching. Now all these methods are automated through classical Machine Learning algorithms for classification and regression.

Payment Fraud: These types of fraud are very common in today's card systems for banking. Fraudsters can steal cards, make counterfeit cards, steal Card ID, etc. Once they steal the confidential data of a user, they can buy things, apply for a loan, and pretty much anything they imagine.

ID Document Forgery: Nowadays these criminals and fraudsters can buy ID proof of a person and use that to enter a system, make use of it, and without any impact get out if it. This type of fraud can put many organizations at risk as these fraudsters can get access to their systems by faking an ID Document and cheating them. These fraudsters are skilful in

creating more legit IDs. So old systems which are used to prevent Identity forging are no more capable to detect these forgeries as these patterns need continuous updating. Machine Learning algorithms are the best tool which evolves with more dataset and shows consistent higher detection rates with time.

Identity Theft: Attackers or cybercriminals can hack into their victims accounts and gain access to their credentials like, name, bank account details, email address, passwords, etc. They can use these credentials to cause harm to their victim. There are three types of identity theft: real name theft, account takeover, and synthetic theft.

1.4 Manual Review and Transaction Rules

Nowadays, Machine Learning in Artificial_Intelligence resolves most of the issues that human beings find difficult to deal with. Previously, industries were using a rule-based approach for fraud detection. But due to the popularity and acceptance of A.I, especially by students and Machine Learning in every industry vertical, organizations have moved from the ruled-based fraud detection to ML-based solutions.

Now, we will look at the rule-based fraud detection system and ML-based systems.

Rule-based Approach or Traditional Approach in Fraud Detection Algorithms

In the rule-based approach, the algorithms are written by fraud analysts. They are based on strict rules. If any changes must be made for detecting a new fraud, then they are done manually either by making those changes in the already existing algorithms or by creating new algorithms. In this approach, with the increase in the number of customers and the data, human effort also increases. So, the rule-based approach is time-consuming and costly. Another drawback of this approach is that it is more likely to have false positives. This is an error condition where an output of a test specifies the existence of a particular condition that does not even exist. The output of a transaction depends upon the rules and guidelines made for training the algorithm for non-fraudulent transactions. So, for a fixed risk threshold, if a transaction is rejected where it should not be, it will generate a condition of high rates of false positives. This false-positive condition will result in losing genuine customers.

1.5 ML-based Fraud Detection Algorithms

In the rule-based approach, the algorithms cannot recognize the hidden patterns. Since they are based on strict rules, they cannot predict fraud by going beyond these rules. But in real world, fraudsters are very skilled and can adopt new techniques every time to commit a crime. Therefore, there is a need for a system that can analyse patterns in data and predict and respond to new situations for which it is not trained or explicitly programmed.

Hence, we use Machine Learning for detecting fraud. Here, a machine tries to learn by itself and becomes better by experience. Also, it is an efficient way of detecting fraud because of its fast computing. It does not even require the guidance of a fraud analyst. It helps in

reducing false positives for transactions as the patterns are detected by an automated system for streaming transactions that are in huge volume.

Two most commonly used types of Machine Learning models for detecting fraud in transactions.

1. **Supervised Learning Used in Fraud Detection Algorithms**

Supervised Learning models are trained on tagged outputs. If a transaction occurs, it is tagged as either 'fraud' or 'non-fraud.' Large amounts of such tagged data are fed into the supervised learning model in order to train it in such a way that it gives a valid output. Also, the accuracy of the model's output depends on how well-organized your data is.

2. **Unsupervised Learning Used in Fraud Detection Algorithm**

Unsupervised learning models are built to detect unusual behaviour in transactions which is not detected previously. Unsupervised learning models involve self-learning that helps in finding hidden patterns in transactions. In this type, the model tries to learn by itself, analyses the available data, and tries to find the similarities and dissimilarities between the occurrences of transactions. This helps in detecting fraudulent activities.

So, both these models, supervised and unsupervised, can be used independently or in combination for detecting anomalies in transactions.

Need for the Fraud Detection Machine Learning Algorithms

Human beings always search for methods, tools, or techniques that reduce the human effort for performing a certain task efficiently. In Machine Learning, algorithms are designed in such a way that they try to learn by themselves using past experience. After learning from the past experience, the algorithms become quite capable of reacting and responding to conditions for which they are not explicitly programmed. So, Machine Learning helps a lot when it comes to fraud detection. It tries to identify hidden patterns that help in detecting

fraud which has not been previously recognized. Also, its computation is fast as compared to the traditional rule-based approaches.

1.6 Advantages of using Machine Learning in Fraud Detection?

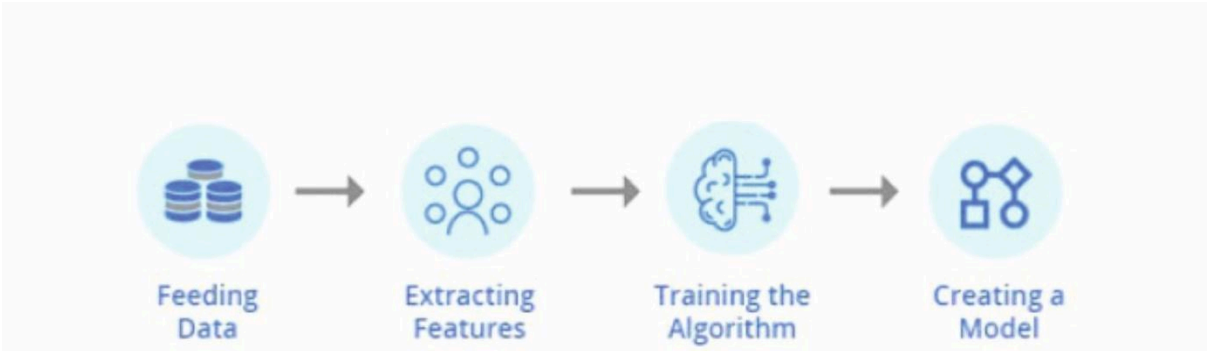
- **Speed:** Machine Learning is widely used because of its fast computation. It analyses and processes data and extracts new patterns from it within no time. For human beings to evaluate the data, it will take a lot of time and evaluation time will increase with the amount of data. Rule-based fraud prevention systems are based on written rules for permitting which type of actions are deemed safe and which one's must raise a flag of suspicion. Now, this Rule-based system is inefficient because it takes much time to write these rules for different scenarios. And that's exactly where Machine Learning based Fraud Detection algorithms succeed in not only learning from these patterns it is capable of detecting new patterns automatically. And it does all of this in a fraction of the time that these rule-based systems could achieve.

- **Scalability:** As more and more data are fed into the Machine Learning-based model, the model becomes more accurate and effective in prediction. Rule-based systems don't evolve by themselves as professionals who developed these systems must write these rules meeting various circumstances. But for Machine Learning based algorithms, a dedicated team of Data Science professionals must be involved in making sure these algorithms are performing as intended.
- **Efficiency:** Machine Learning algorithms perform the redundant task of data analysis and try to find hidden patterns repetitively. Their efficiency is better in giving results in comparison with manual efforts. It avoids the occurrence of false positives which counts for its efficiency. Due to their efficiency in detecting these patterns, the specialists in Fraud detection could now focus on more advanced and complex patterns, leaving the low or moderate level problems to these Machine Learning based algorithms.

CHAPTER 2

2.1 How does a Machine Learning system work for Fraud Detection?

The below picture shows the basic structure of the working of fraud detection algorithms using Machine Learning:



Feeding Data: First, the data is fed into the model. The accuracy of the model depends on the amount of data on which it is trained, more data better the model performs.
For detecting frauds specific to a particular business, we need to input more amounts of data into our model. This will train your model in such a way that it detects fraud activities specific to your business perfectly.

Extracting Features: Feature extraction basically works on extracting the information of each and every thread associated with a transaction process. These can be the location from where the transaction is made, the identity of the customer, the mode of payments, and the network used for transaction.

- **Identity**: This parameter is used to check a customer's email address, mobile number, etc. and it can check the credit score of the bank account if the customer applies for a loan.
- **Location**: It checks the IP address of the customer and the fraud rates at the customer's IP address and shipping address.
- **Mode of Payment**: It checks the cards used for the transaction, the name of the cardholder, cards from different countries, and the rates of fraud of the bank account used.
- **Network**: It checks for the number of mobile numbers and emails used within a network for the transaction.

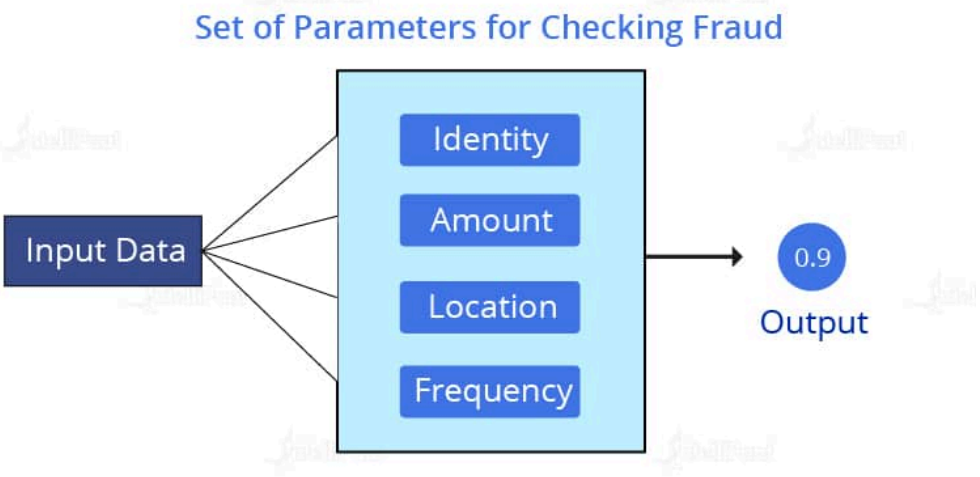
Training the Algorithm: Once a fraud detection algorithm is created, we need to train it by providing customers data so that the fraud detection algorithm learns how to distinguish between 'fraud' and 'genuine' transactions.

Creating a Model: After fraud detection algorithm on a specific dataset is trained, we are ready with a model that works for detecting 'fraudulent' and 'non-fraudulent' transactions in the business.
The advantage of Machine Learning in fraud detection algorithms is that it keeps on improving as it is exposed to more data.
There are many techniques in Machine Learning used for fraud detection. Here, with the help of some use cases, we will understand how Machine Learning is used in fraud detection.

2.2 Techniques of Machine Learning for Fraud Detection Algorithms

1. **Fraud Detection Machine Learning Algorithms Using Logistic Regression**: Logistic Regression is a supervised learning technique that is used when the decision is categorical. It means that the result will be either 'fraud' or 'non-fraud' if a transaction occurs.

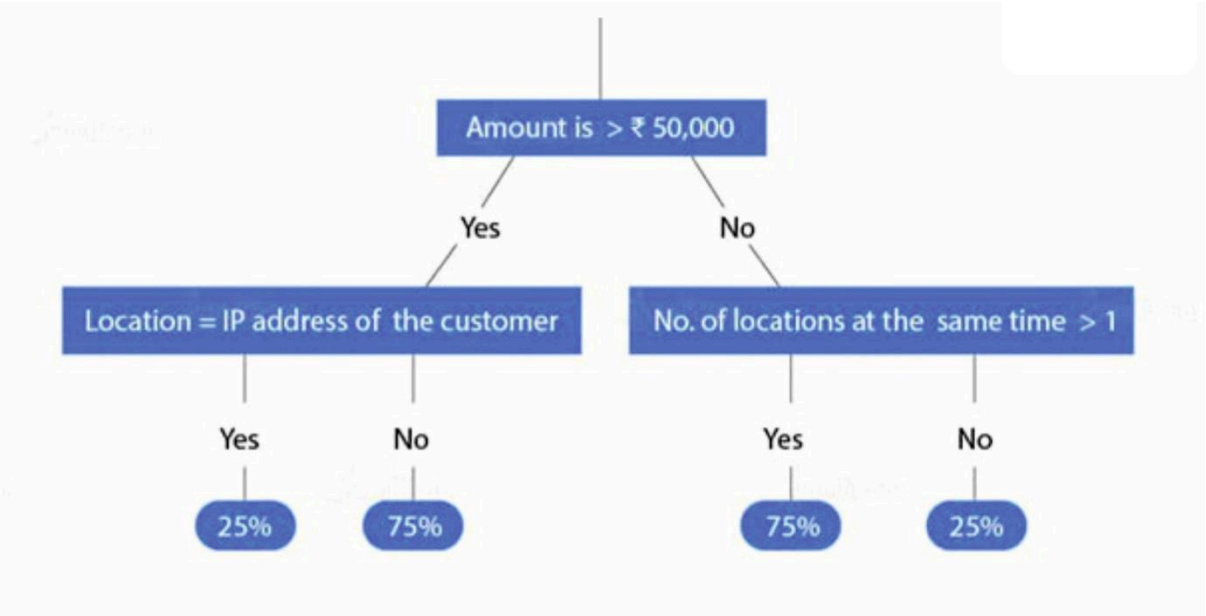
Use Case: Let us consider a scenario where a transaction occurs and we need to check whether it is a 'fraudulent' or 'non-fraudulent' transaction. There will be given set of parameters that are checked and, on the basis of the probability calculated, we will get the output as 'fraud' or 'non-fraud.'



In the above diagram, we can see that the probability calculated is 0.9. This means that there is a 90 percent chance that the transaction is 'genuine' and there is a 10 percent probability that it is a 'fraud' transaction.

2. **Fraud Detection Machine Learning Algorithms Using Decision Tree: Decision Tree** algorithms in fraud detection are used where there is a need for the **classification** of unusual activities in a transaction from an authorized user. These algorithms consist of constraints that are trained on the dataset for classifying fraud transactions.

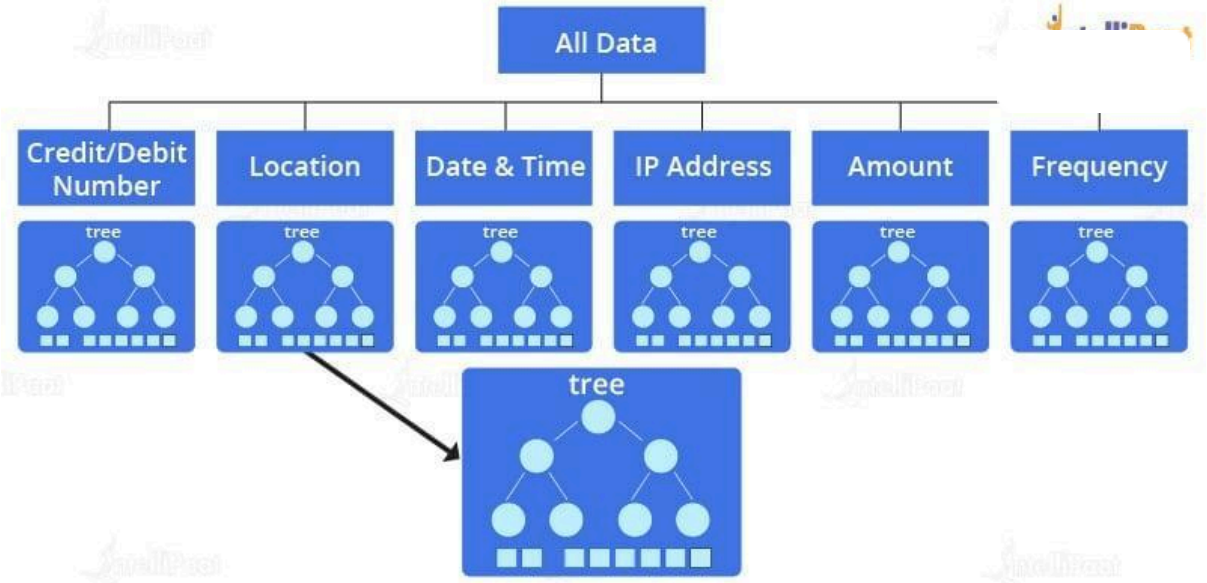
Use Case: Let us consider a scenario where a user makes transactions. We will build a decision tree to predict the probability of fraud based on the transaction made.



'non-fraud.'

3. **Fraud Detection Machine Learning Algorithms Using Random Forest**: Random Forest uses a combination of decision trees to improve the results. Each decision tree checks for different conditions. They are trained on random datasets and, based on the training of the decision trees, each tree gives the probability of the transaction being 'fraud' and 'non-fraud.' Then, the model predicts the result accordingly.

Use Case: Let's consider a scenario where a transaction is made. Now, we will see how the random forest in Machine Learning is used in fraud detection algorithms.



When a request for a transaction is given to the model, it checks for the information like the credit/debit card number, location, date, time, the IP address, the amount, and the frequency of the transaction. All this dataset is fed as an input into the fraud detection algorithm. Then this fraud detection algorithm selects variables from the given dataset that help in splitting up of the dataset. The below diagram shows the splitting up of the dataset into multiple decision trees.

So, the sub-trees consist of variables and the conditions to check those variables for an authorized transaction.

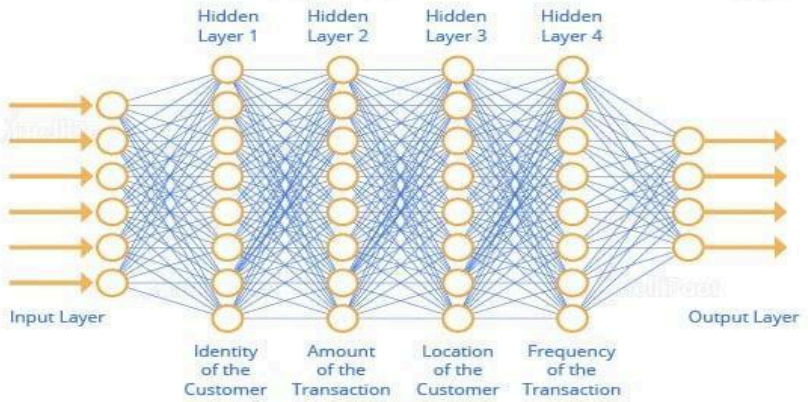
After checking all the conditions, all the sub-trees will give the probabilities for a transaction to be ‘fraud’ and ‘non-fraud.’ Based on the combined result, the model will mark the transaction as ‘fraud’ or ‘genuine.’

4. **Fraud Detection Machine Learning Algorithms Using Neural Networks**: Neural Networks is a concept inspired by the working of a human brain. Neural networks in Deep Learning uses different layers for computation. It uses cognitive computing that helps in building machines capable of using

self-learning algorithms that involve the use of data mining, pattern recognition, and natural language processing. It is trained on a dataset passing it through different layers several times.

It gives more accurate results than other models as it uses cognitive computing and it learns from the patterns of authorized behaviour and thus distinguishes between ‘fraud’ and ‘genuine’ transactions.

Use Case: There are different layers in a neural network that focus on different parameters to make a decision whether a transaction is ‘fraud’ or ‘non-fraud.’ In the below diagram it is shown how the layers of neural networks represent and work on different parameters.



First, the data is fed into the neural network. After that, the Hidden Layer 1 checks the amount of transaction, and similarly other layers check for the location, identity, IP address of the location, the frequency of transaction, and the mode of payment. There can be more business-specific parameters. These individual layers work on these parameters, and computation is done based on the models’ self-learning and past experience to calculate the probabilities for detecting frauds.

2.3 Autoencoder in Anomaly Detection

Autoencoders are a type of neural network architecture widely used for anomaly detection in various applications, including online banking fraud. They work by compressing input data into a lower-dimensional representation and then attempting to reconstruct the original data from this compressed representation. Normal data is learned and reproduced with high fidelity, while anomalies, which deviate significantly from normal data, result in high reconstruction errors. The bottleneck architecture of autoencoders forces them to learn condensed representations, making them effective in distinguishing anomalies from normal data.

AutoEncoders is a neural network that learns to copy its inputs to outputs. In simple words, AutoEncoders are used to learn the compressed representation of raw data. Autoencoders are based on unsupervised machine learning that applies the backpropagation technique and sets the target values equal to the inputs. It does here is simple dimensionality reduction, the same as the PCA algorithm. But the potential benefit is how they treat the non-linearity of data. It allows the Model to learn very powerful generalizations. And it can reconstruct the output back with lower significant loss of information than PCA. This is the advantage of AutoEncoder over PCA. Let us summarize Autoencoder in the below three key points.

- It is an unsupervised ML algorithm similar to PCA.
- It minimizes the same objective function as PCA.
- The neural network target output is its output.
- **Unsupervised –** They do not need labels to train on.
- **data specific –** They can only compress the data similar to what they have been trained on. for example, an autoencoder trained on the human face will not perform well on images of modern buildings. This improvises the difference between auto-encoder and mp3 kind of compression algorithm, which only holds assumptions about sound.
- **Lossy –** Autoencoders are lossy, meaning the decompressed output will be degraded.

Architecture of AutoEncoder

Now let us understand the architecture of Autoencoder and have a deeper insight into the hidden layers. So, in Autoencoder, we add a couple of layers between input and output, and the sizes of this layer are smaller than the input layer.

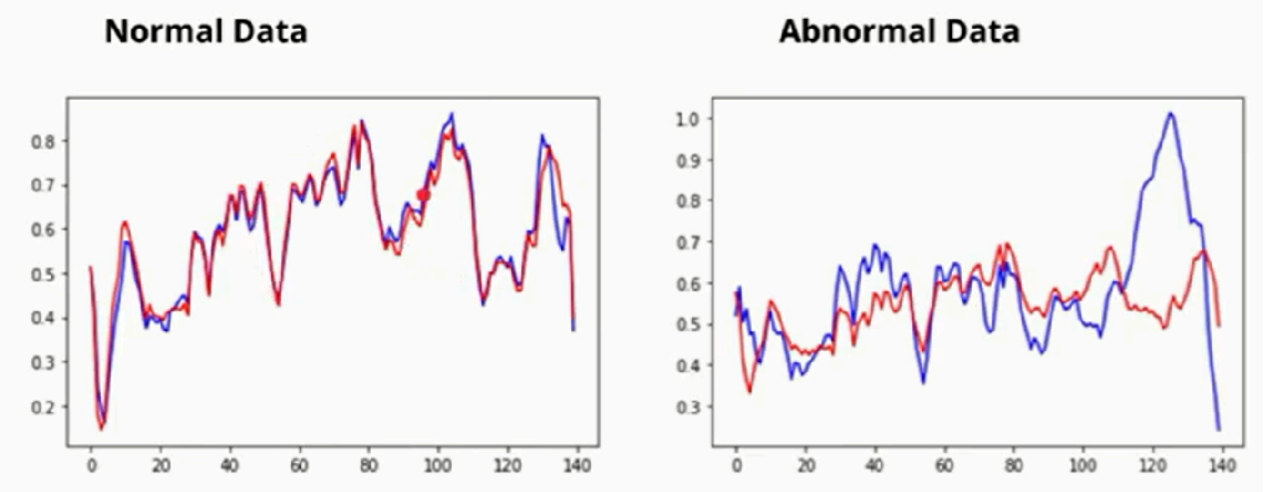
A critical part of the Autoencoder is the bottleneck. The bottleneck approach is a beautifully elegant approach to representation learning specifically for deciding which aspects of obs data are relevant information and which aspects can be thrown away. It describes by balancing two criteria.

1. The compactness of representation, measured as the compressibility number of bits needed to store compressibility.
2. Information the representation retains about some behaviorally relevant variables.

In this case, the difference between input representation and output representation is known as reconstruction error (error between input vector and output vector). One of the predominant use cases of the Autoencoder is anomaly detection.

We pass Autoencoder with majority classes (normal data). The training objective is to minimize the reconstruction error, and the training objective is to minimize this. as training progresses, the model weights for the encoder and decoder are updated. The encoder is a downsampler, and the decoder is an upsampler. Encoder and decoder can be ANN, CNN, or LSTM neural network.

What AutoEncoder does? It learns the reconstruction function that works with normal data, and we can use this Model for anomaly detection. We get low reconstruction error for normal data and high for abnormal data (minority class).



Reconstruction error is the difference between the red and blue lines, and this reconstruction error is undoubtedly high. So, a model will not fit perfectly.

1. Due to the **bottleneck architecture** of the neural network, it is forced to learn a **condensed representation** from which to reproduce the original input.
2. We feed it **only normal transactions**, which it will learn to reproduce with high fidelity.

As a consequence, if a **fraud transaction is sufficiently distinct** from normal transactions, the auto-encoder will have trouble reproducing it with its learned weights, and the subsequent **reconstruction loss will be high**.

Anything above a specific loss (threshold) will be **flagged as anomalous** and thus labeled as fraud.

Autoencoders offer a powerful tool for detecting anomalies in online banking transactions, thereby enhancing the security of financial systems. By understanding the architecture and principles of autoencoders, financial institutions can leverage their capabilities effectively to combat fraud and protect their customers' assets. Continual advancements in machine learning techniques, coupled with a thorough understanding of fraud dynamics, are essential for staying ahead of evolving fraud threats in the digital age.

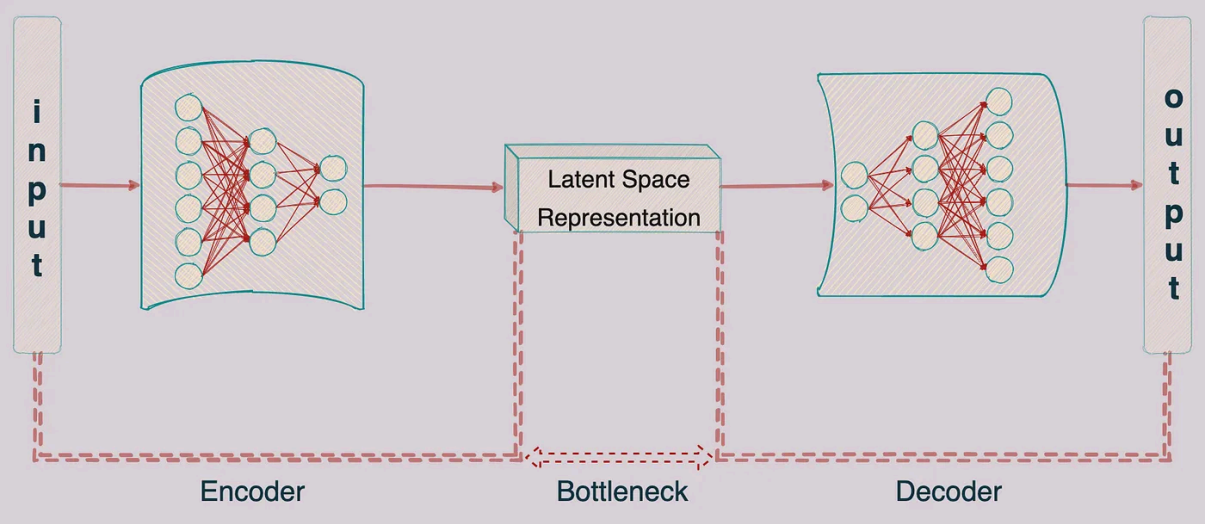
CHAPTER 3

Technologies used:

1 Kafka:

Kafka Streams is a client library for processing and analysing data stored in Kafka. It builds upon important stream processing concepts such as properly distinguishing between event time and processing time, windowing support, and simple yet efficient management and real-time querying of application state.

Kafka Streams has a **low barrier to entry**: You can quickly write and run a small-scale



proof-of-concept on a single machine; and you only need to run additional instances of your application on multiple machines to scale up to high-volume production workloads. Kafka Streams transparently handles the load balancing of multiple instances of the same application by leveraging Kafka's parallelism model.

Some highlights of Kafka Streams:

- Designed as a **simple and lightweight client library**, which can be easily embedded in any Java application and integrated with any existing packaging, deployment and operational tools that users have for their streaming applications.
- Has **no external dependencies on systems other than Apache Kafka itself** as the internal messaging layer; notably, it uses Kafka's partitioning model to horizontally scale processing while maintaining strong ordering guarantees.
- Supports **fault-tolerant local state**, which enables very fast and efficient stateful operations like windowed joins and aggregations.
- Supports **exactly-once** processing semantics to guarantee that each record will be processed once and only once even when there is a failure on either Streams clients or Kafka brokers in the middle of processing.
- Employs **one-record-at-a-time processing** to achieve millisecond processing latency, and supports **event-time based windowing operations** with late arrival of records.
- Offers necessary stream processing primitives, along with a **high-level Streams DSL** and a **low-level Processor API**.

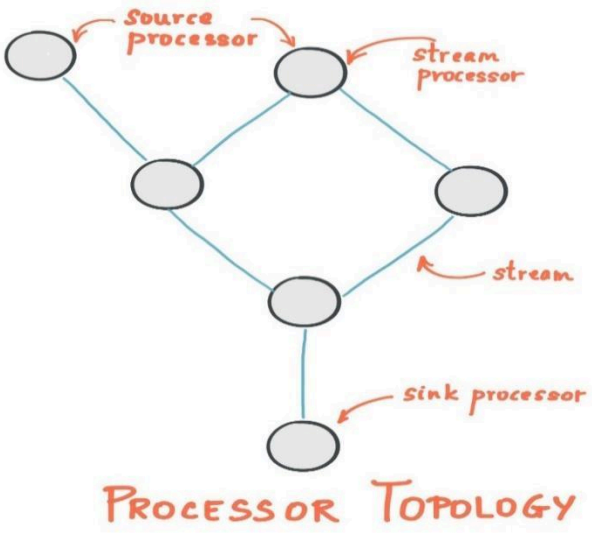
key concepts of Kafka Streams.

Stream Processing Topology

- A stream is the most important abstraction provided by Kafka Streams: it represents an unbounded, continuously updating data set. A stream is an ordered, repayable, and fault-tolerant sequence of immutable data records, where a data record is defined as a key-value pair.
- A stream processing application is any program that makes use of the Kafka Streams library. It defines its computational logic through one or more processor topologies, where a processor topology is a graph of stream processors (nodes) that are connected by streams (edges).
- A stream processor is a node in the processor topology; it represents a processing step to transform data in streams by receiving one input record at a time from its upstream processors in the topology, applying its operation to it, and may subsequently produce one or more output records to its downstream processors.

There are two special processors in the topology:

- **Source Processor**: A source processor is a special type of stream processor that does not have any upstream processors. It produces an input stream to its topology from one or multiple Kafka topics by consuming records from these topics and forwarding them to its down-stream processors.
- **Sink Processor**: A sink processor is a special type of stream processor that does not have down-stream processors. It sends any received records from its up-stream processors to a specified Kafka topic.



.

Time

A critical aspect in stream processing is the notion of time, and how it is modelled and integrated. For example, some operations such as windowing are defined based on time boundaries.

Common notions of time in streams are:

- **Event time** - The point in time when an event or data record occurred, i.e. was originally created "at the source". Example: If the event is a geo-location change

reported by a GPS sensor in a car, then the associated event-time would be the time when the GPS sensor captured the location change.

- **Processing time** - The point in time when the event or data record happens to be processed by the stream processing application, i.e. when the record is being consumed. The processing time may be milliseconds, hours, or days etc. later than the original event time. Example: Imagine an analytics application that reads and processes the geo-location data reported from car sensors to present it to a fleet management dashboard. Here, processing-time in the analytics application might be milliseconds or seconds (e.g. for real-time pipelines based on Apache Kafka and Kafka Streams) or hours (e.g. for batch pipelines based on Apache Hadoop or Apache Spark) after event-time.
- **Ingestion time** - The point in time when an event or data record is stored in a topic partition by a Kafka broker. The difference to event time is that this ingestion timestamp is generated when the record is appended to the target topic by the Kafka broker, not when the record is created "at the source". The difference to processing time is that processing time is when the stream processing application processes the record. For example, if a record is never processed, there is no notion of processing time for it, but it still has an ingestion time.

Kafka Streams assigns a timestamp to every data record via the Timestamp Extractor interface. These per-record timestamps describe the progress of a stream with regards to time and are leveraged by time-dependent operations such as window operations. As a result, this time will only advance when a new record arrives at the processor. We call this data-driven time the stream time of the application to differentiate with the wall-clock time when this application is executing. Concrete implementations of the Timestamp Extractor interface will then provide different semantics to the stream time definition. For example, retrieving or computing timestamps based on the actual contents of data records such as an embedded timestamp field to provide event time semantics and returning the current wall-clock time thereby yield processing time semantics to stream time. Developers can thus enforce different notions of time depending on their business needs.

Finally, whenever a Kafka Streams application writes records to Kafka, then it will also assign timestamps to these new records. The way the timestamps are assigned depends on the context:

- When new output records are generated via processing some input record, for example, context.forward() triggered in the process() function call, output record timestamps are inherited from input record timestamps directly.
- When new output records are generated via periodic functions such as Punctuator#punctuate(), the output record timestamp is defined as the current internal time (obtained through context.timestamp()) of the stream task.
- For aggregations, the timestamp of a resulting aggregate update record will be that of the latest arrived input record that triggered the update.

Aggregations

An aggregation operation takes one input stream or table, and yields a new table by combining multiple input records into a single output record. Examples of aggregations are computing counts or sum.

In the Kafka Streams DSL, an input stream of an aggregatiom can be a KStream or a KTable, but the output stream will always be a KTable. This allows Kafka Streams to update an aggregate value upon the late arrival of further records after the value was produced and emitted. When such late arrival happens, the aggregating KStream or KTable emits a new aggregate value. Because the output is a KTable, the new value is considered to overwrite the old value with the same key in subsequent processing steps.

Windowing

Windowing lets you control how to group records that have the same key for stateful operations such as aggregations or join into so-called windows. Windows are tracked per record key.

Windowing operations are available in the Kafka Streams DSL. When working with windows, you can specify a retention period for the window. This retention period controls how long Kafka Streams will wait for out of order or late arriving data records for a given window. If a record arrives after the retention period of a window has passed, the record is discarded and will not be processed in that window.

Late-arriving records are always possible in the real world and should be properly accounted for in your applications. It depends on the effective time semantics how late records are handled. In the case of processing-time, the semantics are "when the record is being processed", which means that the notion of late records is not applicable as, by definition, no record can be late. Hence, late-arriving records can only be considered as such (i.e. as arriving "late") for event-time or ingestion-time semantics. In both cases, Kafka Streams is able to properly handle late-arriving records.

Duality of Streams and Tables

When implementing stream processing use cases in practice, you typically need both streams and also databases. An example use case that is very common in practice is an e-commerce application that enriches an incoming *stream* of customer transactions with the latest customer information from a database table. In other words, streams are everywhere, but databases are everywhere, too.

Any stream processing technology must therefore provide first-class support for streams and tables. Kafka's Streams API provides such functionality through its core abstractions for streams and tables, which we will talk about in a minute. Now, an interesting observation is that there is actually a close relationship between streams and tables, the so-called stream-table duality. And Kafka exploits this duality in many ways: for example, to make your applications elastic, to support fault-tolerant stateful processing, or to run interactive queries against your application's latest processing results. And, beyond its internal usage, the Kafka Streams API also allows developers to exploit this duality in their own applications.

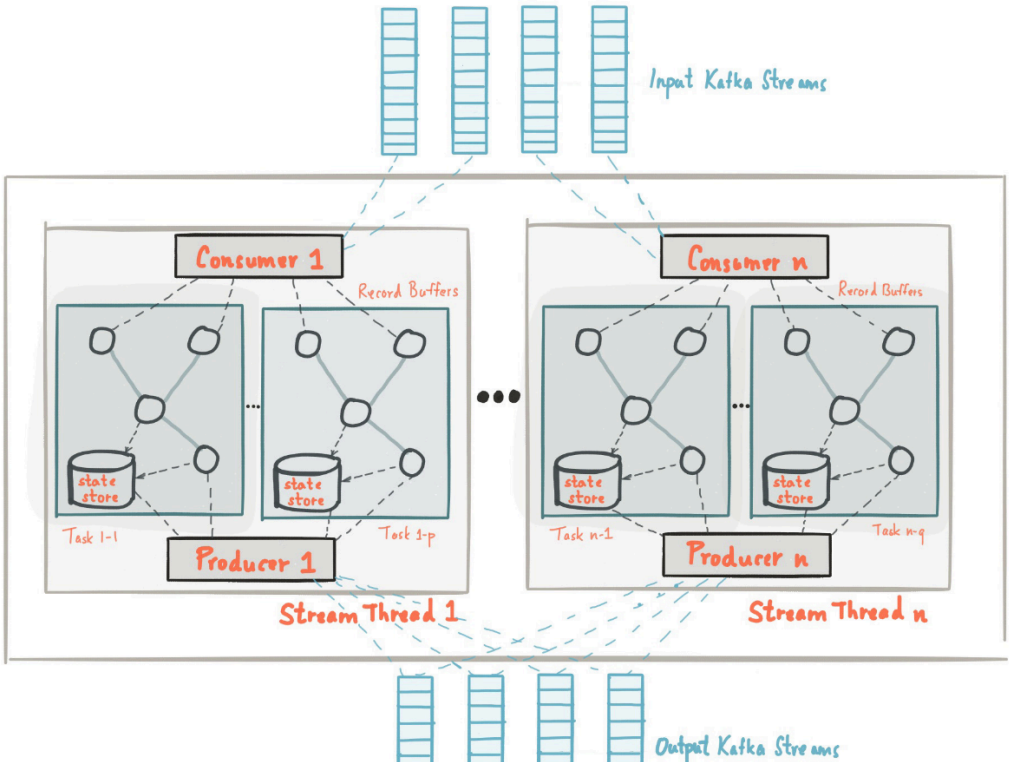
Before we discuss concepts such as aggregations in Kafka Streams, we must first introduce tables in more detail, and talk about the aforementioned stream-table duality. Essentially, this duality means that a stream can be viewed as a table, and a table can be viewed as a stream.

States

Some stream processing applications don't require state, which means the processing of a message is independent from the processing of all other messages. However, being able to maintain state opens many possibilities for sophisticated stream processing applications: you can join input streams, or group and aggregate data records. Many such stateful operators are provided by the Kafka Streams DSL

Kafka Streams provides so-called state stores, which can be used by stream processing applications to store and query data. This is an important capability when implementing stateful operations. Every task in Kafka Streams embeds one or more state stores that can be accessed via APIs to store and query data required for processing. These state stores can either be a persistent key-value store, an in-memory HashMap, or another convenient data structure. Kafka Streams offers fault-tolerance and automatic recovery for local state stores.

Kafka Streams allows direct read-only queries of the state stores by methods, threads, processes, or applications external to the stream processing application that created the state



stores. This is provided through a feature called Interactive Queries. All stores are named, and Interactive Queries exposes only the read operations of the underlying implementation.

2. SPARK:

Apache Spark is a lightning-fast, open-source data-processing engine for machine learning and AI applications, backed by the largest open-source community in big data.

Apache Spark (Spark) easily handles large-scale data sets and is a fast, general-purpose clustering system that is well-suited for PySpark. It is designed to deliver the computational speed, scalability, and programmability required for big data—specifically for streaming data, graph data, analytics, machine learning, large-scale data processing, and artificial intelligence (AI) applications.

Spark's analytics engine processes data 10 to 100 times faster than some alternatives, such as Hadoop for smaller workloads. It scales by distributing processing workflows across large clusters of computers, with built-in parallelism and fault tolerance. It even includes APIs for programming languages that are popular among data analysts and data scientists, including Scala, Java, Python, and R.

Spark was developed in 2009 at UC Berkeley's AMPLab. Today, it is maintained by the Apache Software Foundation and boasts the largest open-source community in big data, with

over 1,000 contributors. It's also included as a core component of several commercial big data offerings.

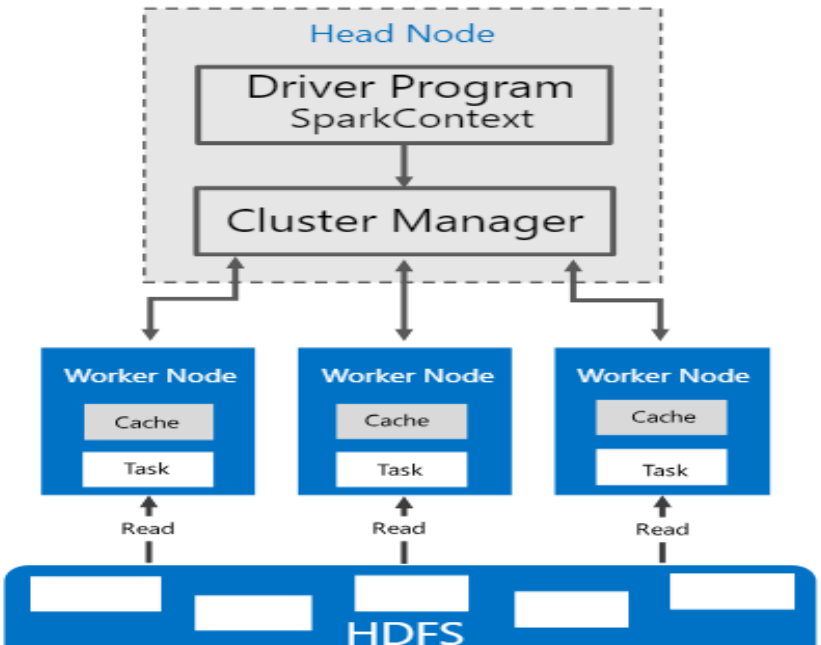
What is Apache Spark?

Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers, either on its own or in tandem with other distributed computing tools. These two qualities are key to the worlds of big data and machine learning, which require the marshalling of massive computing power to crunch through large data stores. Spark also takes some of the programming burdens of these tasks off the shoulders of developers with an easy-to-use API that abstracts away much of the grunt work of distributed computing and big data processing.

Spark was developed in 2009 at UC Berkeley's AMPLab. Today, it is maintained by the Apache Software Foundation and boasts the largest open-source community in big data, with over 1,000 contributors. It's also included as a core component of several commercial big data offerings.

How Apache Spark works

Apache Spark has a hierarchical primary/secondary architecture. The Spark Driver is the primary node that controls the cluster manager, which manages the secondary nodes and delivers data results to the application client.



Based on the application code, Spark Driver generates the *SparkContext*, which works with the cluster manager—Spark's Standalone Cluster Manager or other cluster managers such as Hadoop YARN, Kubernetes, or Mesos—to distribute and monitor execution across the nodes. It also creates Resilient Distributed Datasets (RDDs), which are the key to Spark's remarkable processing speed.

Resilient Distributed Dataset (RDD)

Resilient Distributed Datasets (RDDs) are fault-tolerant collections of elements that can be distributed among multiple nodes in a cluster and worked on in parallel. RDDs are a fundamental structure in Apache Spark.

Spark loads data by referencing a data source or by parallelizing an existing collection with the SparkContext parallelize method of caching data into an RDD for processing. Once data is loaded into an RDD, Spark performs transformations and actions on RDDs in memory—the key to Spark's speed. Spark also stores the data in memory unless the system runs out of memory, or the user decides to write the data to disk for persistence.

Each dataset in an RDD is divided into logical partitions, which may be computed on different nodes of the cluster. And users can perform two types of RDD operations**:** transformations and actions**.** Transformations are operations applied to create a new RDD**.** Actions are used to instruct Apache Spark to apply computation and pass the result back to the driver.

Spark supports a variety of actions and transformations on RDDs. This distribution is done by Spark, so users don't have to worry about computing the right distribution.

Directed Acyclic Graph (DAG)

As opposed to the two-stage execution process in MapReduce, Spark creates a Directed Acyclic Graph (DAG) to schedule tasks and the orchestration of worker nodes across the cluster. As Spark acts and transforms data in the task execution processes, the DAG scheduler facilitates efficiency by orchestrating the worker nodes across the cluster. This task-tracking makes fault tolerance possible, as it reapplies the recorded operations to the data from a previous state.

Data Frames and Datasets

In addition to RDDs, Spark handles two other data types: Data Frames and Datasets.

Data Frames are the most common structured application programming interfaces (APIs) and represent a table of data with rows and columns. Although RDD has been a critical feature to Spark, it is now in maintenance mode. Because of the popularity of Spark's Machine Learning Library (MLlib), Data Frames have taken on the lead role as the primary API for MLlib (a set of machine learning algorithms for scalability plus tools for feature selection and building ML pipelines). This is important to note when using the MLlib API, as Data Frames provide uniformity across the different languages, such as Scala, Java, Python, and R.

Datasets are an extension of Data Frames that provide a type-safe, object-oriented programming interface. Datasets are, by default, a collection of strongly typed JVM objects, unlike Data Frames.

Spark SQL enables data to be queried from Data Frames and SQL data stores, such as Apache Hive. Spark SQL queries return a Data Frame or Dataset when they are run within another language.

Spark Core

Spark Core is the base for all parallel data processing and handles scheduling, optimization, RDD, and data abstraction. Spark Core provides the functional foundation for the Spark libraries, Spark SQL, Spark Streaming, the MLlib machine learning library, and GraphX graph data processing. The Spark Core and cluster manager distribute data across the Spark cluster and abstract it. This distribution and abstraction make handling Big Data very fast and user-friendly.

Spark APIs

Spark includes a variety of application programming interfaces (APIs) to bring the power of Spark to the broadest audience. Spark SQL enables for interaction with RDD data in a relational manner. Spark also has a well-documented API for Scala, Java, Python, and R. Each language API in Spark has its specific nuances in how it handles data. RDDs, Data Frames, and Datasets are available in each language API. With APIs for such a variety of languages, Spark makes big data processing accessible to more diverse groups of people with backgrounds in development, data science, data engineering, and statistics.

Advantages of Apache Spark

Spark speeds development and operations in a variety of ways. Spark will help teams:

- **Accelerate app development**: Apache Spark's Streaming and SQL programming models backed by MLlib and GraphX make it easier to build apps that exploit machine learning and graph analytics.

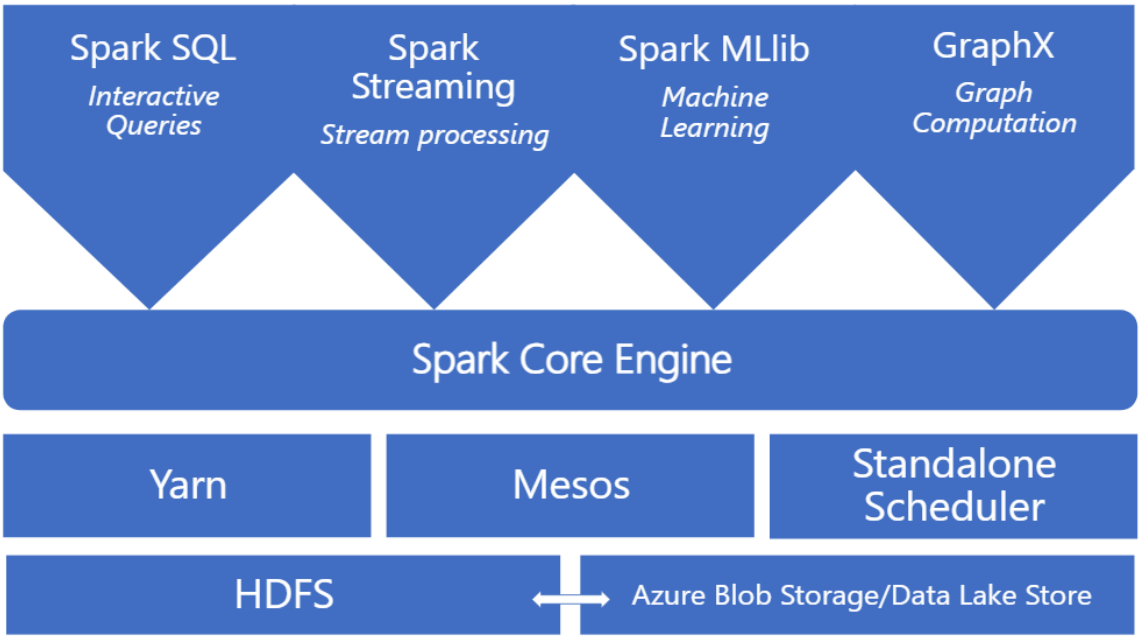
- **Innovate faster**: APIs provide ease of use when manipulating semi-structured data and transforming data.

- **Manage with ease**: A unified engine supports SQL queries, streaming data, machine learning (ML) and graph processing.

- **Optimize with open technologies**: The OpenPOWER Foundation enables GPU, CAPI Flash, RDMA, FPGA acceleration and machine learning innovation to optimize performance for Apache Spark workloads.

- **Process faster**: Spark can be 100x faster than Hadoop (link resides outside ibm.com) for smaller workloads because of its advanced in-memory computing engine and disk data storage.

- **Speed memory access**: Spark can be used to create one large memory space for data processing, enabling more advanced users to access data via interfaces using Python, R, and Spark SQL.



3. MySQL

MySQL is an open-source Relational Database Management System (RDBMS). It is based on Structured Query Language (SQL – a language to manage the database and perform CRUD operations such as create, read, etc., update and delete.). Examples Google, Facebook, Yahoo, and other Tech giants have employed MySQL to enhance their data processing capabilities.

MySQL is essential for LAMP, a web development platform with the Linux operating system, Apache as the web server, MySQL as RDBMS, and PHP as a programming language. Nevertheless, Python or Perl can be used instead of PHP as the programming or scripting language.

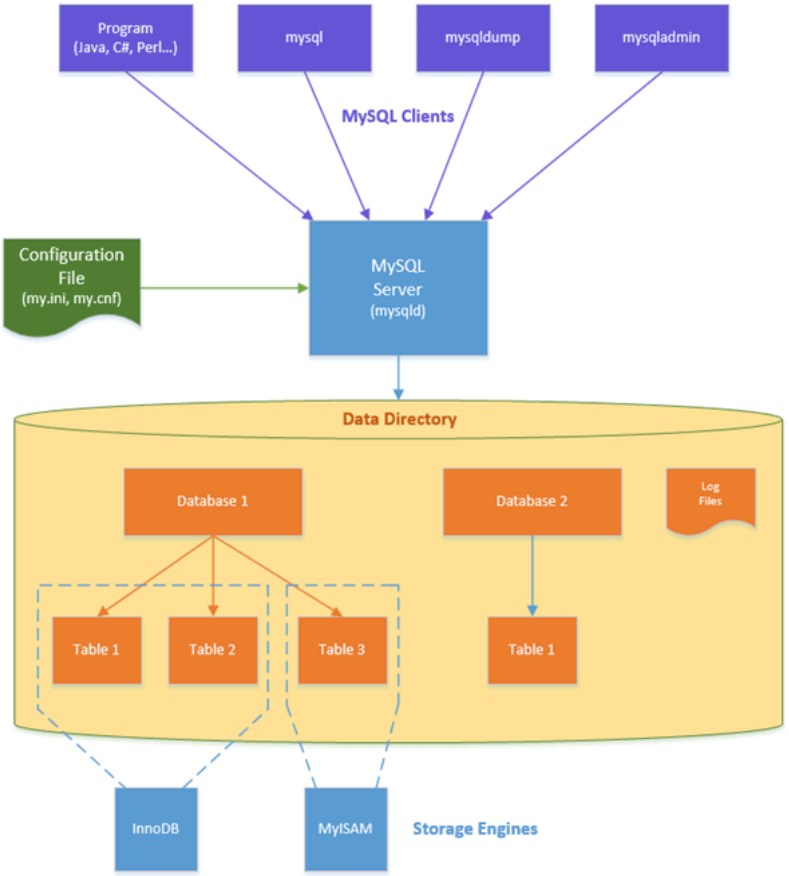
It can be used everywhere where the Data needs to be stored relationally, i.e., in a tabular format. Every table has a primary key, and rows can relate to each other using this primary key.

Architecture of MySQL Server

The architecture of MySQL mainly consists of the following components:

1. MySQL Server
2. MySQL Clients
3. Data Directory
4. Storage Engine

MySQL is based on a Client-Server Model.



1. **MySQL Server:** It is a MySQL instance where the actual data is stored and processed. This component is responsible for processing the incoming queries from MySQL clients and manipulating the database tables. It is also responsible for accepting database connections that are coming from MySQL Clients.

2. **MySQL Clients:** The MySQL clients use utilities for communicating with the MySQL server. In other words, these programs communicate with the MySQL server. MySQL clients include programs like Perl, PHP, Java, MySQL, mysqladmin, and tools such as MySQL dump, mysqlcheck, and myisamchk.
3. **Data Directory:** The Data Directory contains stored data due to ongoing operations over applications/software or servers. It includes Databases, tables, log files, stored procedures.

   It is possible to change the storage location for datadir in case someone runs out of allocated space.
4. **Storage Engine:** A storage engine is a software module RDBMS uses to perform CRUD operations (Create, read, update, and delete).

4 Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets you run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you don't need to rely on what's installed on the host. You can share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers.

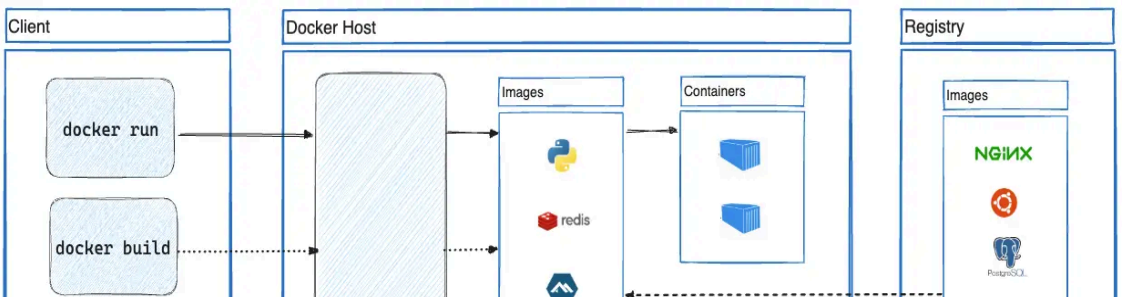
Develop your application and its supporting components using containers.

The container becomes the unit for distributing and testing your application.

When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data centre, a cloud provider, or a hybrid of the two.

Docker architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



5. Large Language Models:

A large language model represents a sophisticated artificial intelligence algorithm that leverages neural network methodologies featuring numerous parameters. Its primary function is to analyse and comprehend human languages or textual information through self-supervised learning techniques. Various applications stem from this model, including text generation, machine translation, summarization, image generation from text, machine coding, chat-bots, and Conversational AI. Examples of such LLM models are Chat GPT by open AI, BERT (Bidirectional Encoder Representations from Transformers) by Google, etc.

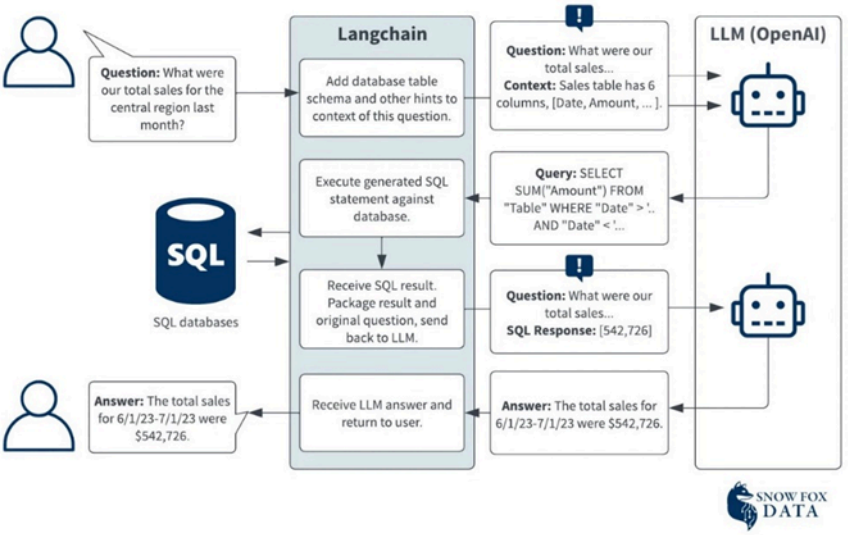


Figure: LLM Architecture

1. **Input Embeddings:** The input text is tokenized into smaller units, such as words or sub-words, and each token is embedded into a continuous vector representation. This embedding step captures the semantic and syntactic information of the input.

2. **Positional Encoding:** Positional encoding is added to the input embeddings to provide information about the positions of the tokens because transformers do not naturally encode the order of the tokens. This enables the model to process the tokens while taking their sequential order into account.

3. **Encoder:** Based on a neural network technique, the encoder analyses the input text and creates a number of hidden states that protect the context and meaning of text data. Multiple encoder layers make up the core of the transformer architecture. Self-attention mechanism and feed-forward neural network are the two fundamental sub-components of each encoder layer.

4. **Self-Attention Mechanism:** Self-attention enables the model to weigh the importance of different tokens in the input sequence by computing attention scores. It allows the model to consider the dependencies and relationships between different tokens in a context-aware manner.

5. **Feed-Forward Neural Network:** After the self-attention step, a feed-forward neural network is applied to each token independently. This network includes fully connected layers with non-linear activation functions, allowing the model to capture complex interactions between tokens.

6. **Decoder Layers:** In some transformer-based models, a decoder component is included in addition to the encoder. The decoder layers enable autoregressive generation, where the model can generate sequential outputs by attending to the previously generated tokens.

7. **Multi-Head Attention:** Transformers often employ multi-head attention, where self-attention is performed simultaneously with different learned attention weights.

This allows the model to capture different types of relationships and attend to various parts of the input sequence simultaneously.

8. **Layer Normalization:** Layer normalization is applied after each sub-component or layer in the transformer architecture. It helps stabilize the learning process and improves the model's ability to generalize across different inputs.

9. **Output Layers:** The output layers of the transformer model can vary depending on the specific task. For example, in language modelling, a linear projection followed by SoftMax activation is commonly used to generate the probability distribution over the next token.

It's important to keep in mind that the actual architecture of transformer-based models can change and be enhanced based on particular research and model creations. To fulfil different tasks and objectives, several models like GPT, BERT, and T5 may integrate more components or modifications.

Examples:

GPT – 3: The full form for GPT is a Generative pre-trained Transformer and this is the third version of such a model hence it is numbered as 3. This is developed by Open AI and you must have heard about Chat GPT which is launched by Open AI and is nothing but the GPT-3 model.

BERT – The full form for this is Bidirectional Encoder Representations from Transformers. This large language model has been developed by Google and is generally used for a variety of tasks related to natural language. Also, it can be used to generate embeddings for a particular text may be to train some other model.

RoBERTa – The full form for this is the Robustly Optimized BERT Pretraining Approach. In the series of attempts to improve the performance of the transformer architecture, RoBERTa is an enhanced version of the BERT model which is developed by Facebook AI Research.

BLOOM – It is the first multilingual LLM generated by the association of the different organizations and researchers who combined their expertise to develop this model which is similar to the GPT-3 architecture.

Use Cases:

1. **Code Generation** – One of the craziest use cases of this service is that it can generate quite an accurate code for a specific task that is described by the user to the model.

2. **Debugging and Documentation of Code** – If you are struggling with some piece of code regarding how to debug it then ChatGPT is your saviour because it can tell you the line of code which are creating issues along with the remedy to correct the same. Also, now you don't have to spend hours writing the documentation of your project you can ask ChatGPT to do this for you.

3. **Question Answering** – As you must have seen that when AI-powered personal assistants were released people used to ask crazy questions to them well you can do that here as well along with the genuine questions.

4. **Language Transfer** – It can convert a piece of text from one language to another as it supports more than 50 native languages. It can also help you correct the grammatical mistakes in your content.

Use Cases:

 **Chatbots:** A Q/A bot is one of the most popular NLP categories to work on these days. It is used by institutional websites, and shopping sites among many others. Prompts on which an AI chatbot Model is trained can largely affect the kind of response a bot generates. An example of what critical information one can add in a prompt can be adding the intent and context of the query so that the bot is not confused in generating relevant answers.

 **Healthcare:** In healthcare, prompt engineers instruct AI systems to summarize medical data and develop treatment recommendations. Effective prompts help AI models process patient data and provide accurate insights and recommendations.

 **Software Development:** Prompt engineering plays a role in software development by using AI models to generate code snippets or provide solutions to programming challenges.

Using prompt engineering in software development can save time and assist developers in coding tasks.

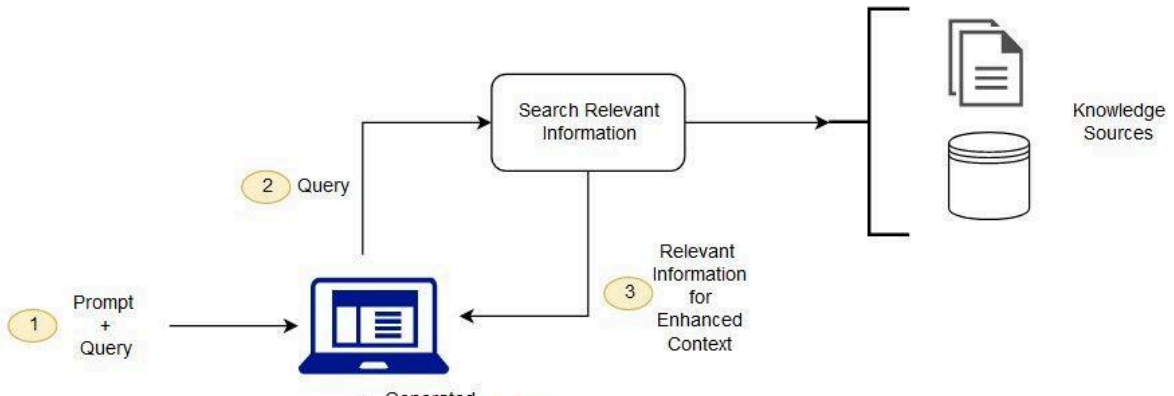
Cybersecurity and computer science: Prompt engineering is used to develop and test security mechanisms. Researchers and practitioners leverage generative AI to simulate cyberattacks and design better defence strategies. Additionally, crafting prompts for AI models can aid in discovering vulnerabilities in software.

6. Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) is a methodology within natural language processing that combines retrieval and generation techniques to enhance the performance of language models. In RAG, external data from various sources, such as document repositories, databases, or APIs, is leveraged to augment the input prompts provided to a model. This approach addresses the limitations of foundation models that are typically trained offline and on general domain corpora, making them less effective for domain-specific tasks.

The process begins by converting documents and user queries into a compatible numerical format using embedding language models. Embedding involves representing text in a vector space with numerical values. RAG model architectures then compare the embeddings of user queries within the vector space of a knowledge library, allowing for the identification of relevant information.

The original user prompt is expanded by appending context retrieved from similar documents within the knowledge library. This enriched prompt, now containing relevant external data, is then presented to the foundation model for further processing. Importantly, RAG provides the flexibility to update knowledge libraries and their associated embeddings asynchronously, allowing for continuous improvement and



adaptation to evolving information. Overall, RAG serves as a powerful tool for knowledge-intensive natural language processing tasks, enabling models to incorporate external information dynamically A model architecture diagram of Retrieval Augmented Generation (RAG) showing how embeddings of user queries and supplemental documents are used to augment foundation model prompts to improve customization.

7. Llama index

Llama Index is a data framework for LLM-based applications to ingest, structure, and access private or domain-specific data. It's available in Python and Typescript

LLMs offer a natural language interface between humans and data. Widely available models come pre-trained on huge amounts of publicly available data like Wikipedia, mailing lists, textbooks, source code and more.

However, while LLMs are trained on a great deal of data, they are not trained on your data, which may be private or specific to the problem you're trying to solve. It's behind APIs, in SQL databases, or trapped in PDFs and slide decks.

Llama Index solves this problem by connecting to these data sources and adding your data to the data LLMs already have. This is often called *Retrieval-Augmented Generation (RAG*).

RAG enables you to use LLMs to query your data, transform it, and generate new insights. You can ask questions about your data, create chatbots, build semi-autonomous agents, and more.

Llama Index provides the following tools:

1. Data connectors ingest your existing data from their native source and format. These could be APIs, PDFs, SQL, and (much) more.

2. Data indexes structure your data in intermediate representations that are easy and performant for LLMs to consume.

3. Engines provide natural language access to your data. For example: - Query engines are powerful retrieval interfaces for knowledge-augmented output. - Chat engines are conversational interfaces for multi-message, “back and forth” interactions with your data.

4. Data agents are LLM-powered knowledge workers augmented by tools, from simple helper functions to API integrations and more.

5. Application integrations tie Llama Index back into the rest of your ecosystem. This could be Lang Chain, Flask, Docker, ChatGPT

8. Natural Language Processing

Natural language processing (NLP) is a machine learning technology that gives computers the ability to interpret, manipulate, and comprehend human language. Organizations today have large volumes of voice and text data from various communication channels like emails, text messages, social media newsfeeds, video, audio, and more. They use NLP software to automatically process this data, analyse the intent or sentiment in the message, and respond in real time to human communication.

NLP Tasks

Natural language processing (NLP) techniques, or NLP tasks, break down human text or speech into smaller parts that computer programs can easily understand. Common text processing and analysing capabilities in NLP are given below.

1. **Part-f-speech tagging:** This is a process where NLP software tags individual words in a sentence according to contextual usages, such as nouns, verbs, adjectives, or adverbs. It helps the computer understand how words form meaningful relationships with each other.

2. **Word-sense disambiguation:** Some words may hold different meanings when used in different scenarios. For example, the word "bat" means different things in these sentences: .

   Baseball players use a bat to hit the ball.

   With word sense disambiguation, NLP software identifies a word's intended meaning, either by training its language model or referring to dictionary definitions.

3. **Speech recognition:** Speech recognition turns voice data into text. The process involves breaking words into smaller parts and understanding accents, slurs, intonation, and nonstandard grammar usage in everyday conversation.

4. **Machine translation:** Machine translation software uses natural language processing to convert text or speech from one language to another while retaining contextual accuracy. The AWS service that supports machine translation is Amazon Translate.

5. **Named-entity recognition:** This process identifies unique names for people, places, events, companies, and more. NLP software uses named-entity recognition to determine the relationship between different entities in a sentence.

6. **Sentiment analysis:** Sentiment analysis is an artificial intelligence-based approach to interpreting the emotion conveyed by textual data. NLP software analyses the text for words or phrases that show dissatisfaction, happiness, doubt, regret, and other hidden emotions.

9. Streamlit

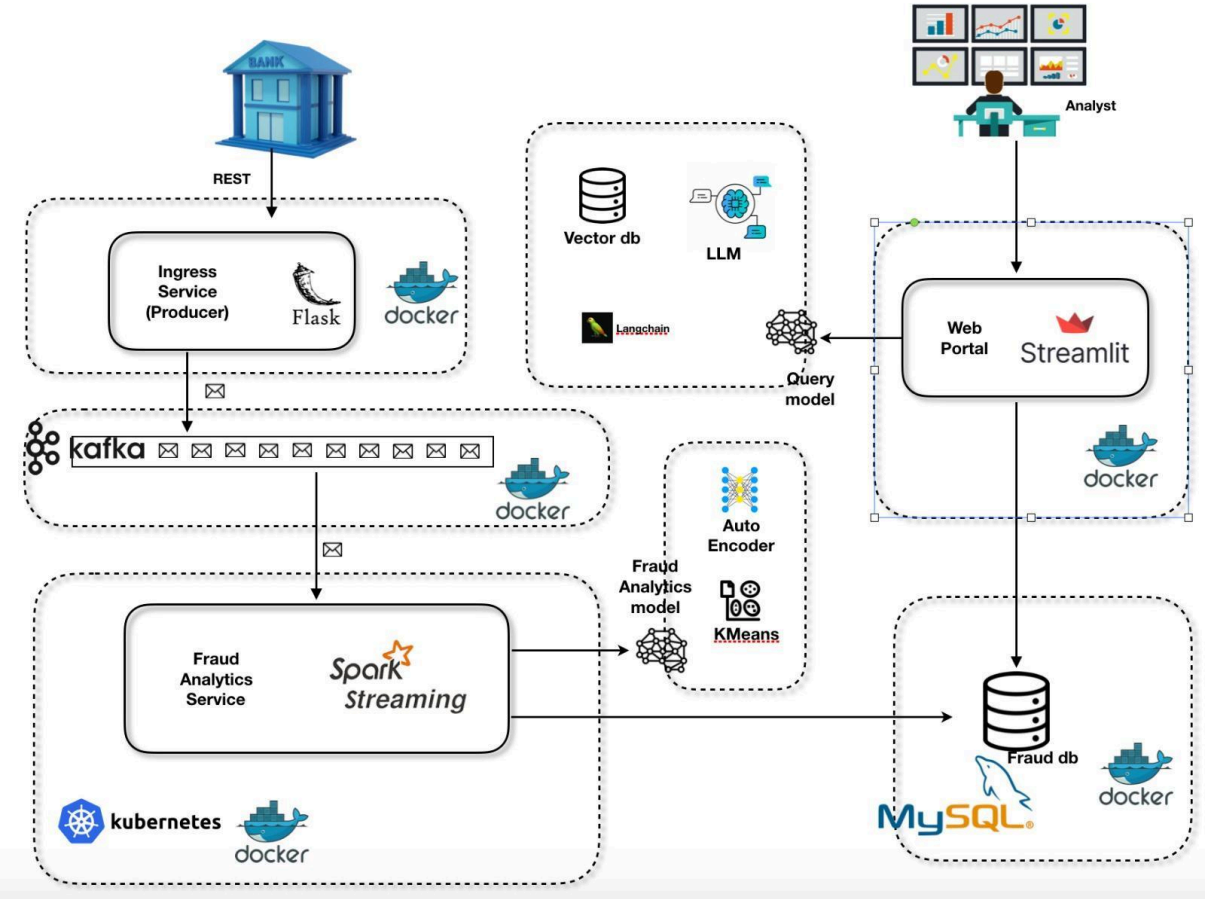
Streamlit is an open-source app framework in python language. It helps us create beautiful web apps for data science and machine learning in a little time. It is compatible with major python libraries such as scikit-learn, keras, PyTorch, latex, numpy, pandas, matplotlib, etc.

Advantages of Streamlit:

1. It embraces python-scripting.
2. Less code is needed to create amazing web-apps.
3. No callbacks are needed since widgets are treated as variables.
4. Data caching simplifies and speeds up computation pipelines.

CHAPTER 4

WIRE DIAGRAM:



Module1: **Data Gathering and Data Preprocessing**

- A proper dataset is searched among various available ones and finalized with the dataset.
- The dataset must be pre-processed to train the model.
- In the preprocessing phase, the dataset is cleaned, and any redundant values, noisy data and null values are removed.
- The Pre-processed data is provided as input to the module.

download data

https://www.kaggle.com/datasets/devondev/financial-anomaly data/data?select=financial_anomaly_data.csv

Data Description

- Timestamp: This column records the date and time when the transaction occurred. It helps in understanding the temporal aspect of transactions, such as patterns over time, frequency, and clustering of activities.

- TransactionID: An identification number assigned to each transaction. It serves as a unique identifier for referencing or tracking specific transactions.

- AccountID: This field represents the unique identifier associated with the bank account involved in the transaction. It links multiple transactions to a specific account, enabling analysis on a per-account basis.

- Amount: The monetary value involved in the transaction. This column provides information about the financial magnitude of each transaction, which is crucial for anomaly detection since unusually high or low values might signify irregularities.

- Merchant: Specifies the entity or business involved in the transaction. This information helps in categorizing transactions (e.g., retail, online, restaurant) and identifying patterns related to specific merchants.

- TransactionType: Describes the nature or category of the transaction, whether it's a withdrawal, deposit, transfer, payment, etc. This column helps in understanding the purpose or direction of the transaction.

- Location: Indicates the place where the transaction occurred. It could be a physical location (e.g., city, country) or an identifier (e.g., store code, online portal), aiding in analyzing geographical spending patterns or detecting anomalies based on unusual transaction locations.

Module 2: Training the model

a) The Pre-processed data is split into training and testing datasets in the 80:20 ratio to avoid the problems of over-fitting and under-fitting.

a. A model is trained using the training dataset with the following algorithms Linear regression, Isolation Forest Classifier, k-means and DB Scan.
b. The trained models are trained with the testing data and results are visualized using bar graphs, scatter plots.
c. The accuracy rates of each algorithm are calculated using different params like RMSE, silhouette score.
d. The algorithm which has provided the better accuracy rate compared to remaining algorithms is taken as final prediction model.

Module 3: Final Prediction model integrated with front end

we have a robust data pipeline set up!

1. Data Retrieval: fetching data via a REST API.

2. Data Streaming: Using Apache Kafka, streaming the data obtained from the REST API. This ensures real-time processing and scalability.

3. Data Processing: Apache Spark is being used to process the data. This could involve various operations like filtering, transformation, and feature engineering.

4. Anomaly Detection: The model built using machine learning techniques, detects anomalies in the data. When anomalies are detected, they are stored or sent somewhere for further analysis.

5. Database Storage: Anomalies are being stored in a MySQL database called "Fraudsdb". This database likely contains tables to store information about detected anomalies.

6. Chatbot Integration: A chatbot is built using an LLM (Long-Short Term Memory) model. This chatbot can answer questions related to anomalies detected by your system. It likely interacts with users to provide insights, explanations, or guidance based on the detected anomalies.

    a) Before using the system, we need to extract metadata about the tables.

    b) The meta data is stored in a document retriever. Once set up, the Data LLM agent is ready to be used and can respond to queries from your users.

    c) User makes an NLP query: user puts question in English language.

    d) System Queries the document retriever

    e) System gets the relevant tables and their metadata needed to answer the question

    f) The System takes the help of an LLM(NSQL) to form the SQL query for your data warehouse to answer the question.

    g) LLM model returns a SQL query

    h) System queries database using the connector

    i) System fetches the relevant data from database

    j) System uses an LLM to summarize the results to answer the question asked by the user.

    k) LLM model sends summarized result

    l) System responds back to the User: The response can be in normal English language or displaying graphs.

System Setup and Execution

Step 1: Setting up the Ingress Service

Technologies Used: Flask, Docker

i. Flask Application (Ingress Service)

Create a new directory for the Ingress Service and within it, create app.py.

C: > Users > theti > Downloads > KafkaSparkStreaming-main > KafkaSparkStreaming-main > FruadAnalytics > producer > app.py

```python
from flask import Flask
from flask import request
from kafka import KafkaProducer
import os
import json

app = Flask(__name__)

@app.route("/add", methods = ['POST'])
def add_contact() :
    content_type = request.headers.get('Content-Type')
    if (content_type == 'application/json'):
        json_data = request.json
        producer = KafkaProducer(bootstrap_servers='kafka:9092')
        producer.send('my-topic', json.dumps(json_data).encode('utf-8'))
        return json_data
    else:
        return 'Content-Type not supported!'

if __name__ == '__main__':
    app.run(debug = True, host = '0.0.0.0')

```

ii. Dockerfile for Ingress Service

Create a Dockerfile in the same directory.

```
# Dockerfile
FROM python:3.9-slim

WORKDIR /app

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY . .

CMD ["python", "app.py"]
```

iii. Requirements.txt for Ingress Service

```
Flask
kafka-python
```

Step 2: Setting up Kafka

Technologies Used: Docker

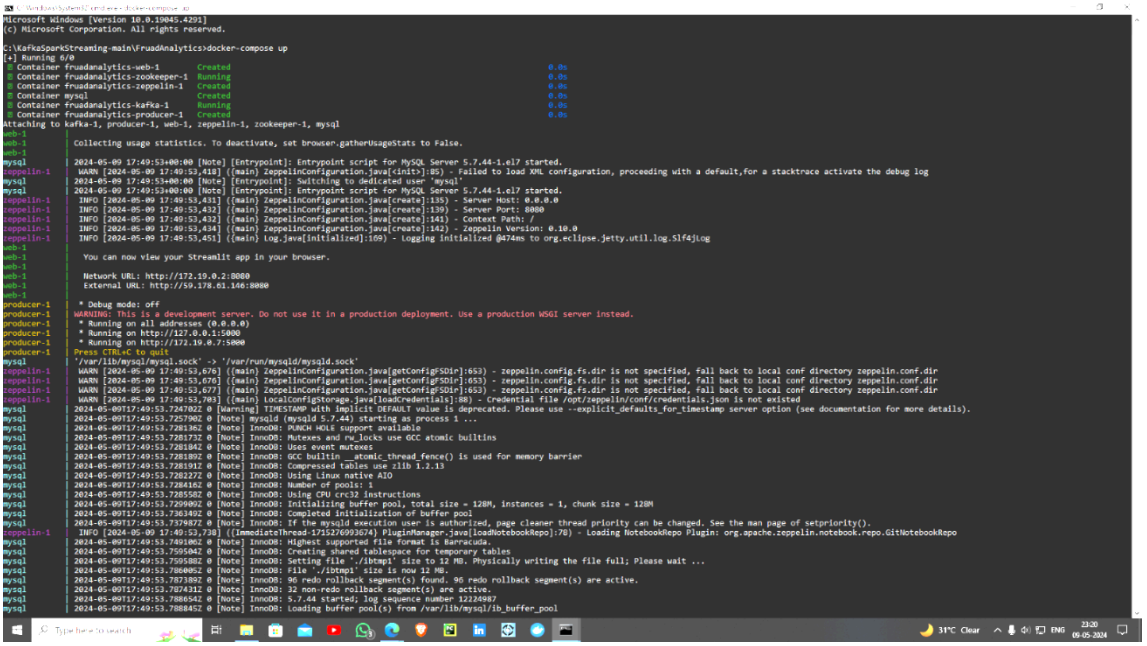
i. Docker Compose for Kafka

Create a docker-compose.yml file to define the Kafka and Zookeeper services.

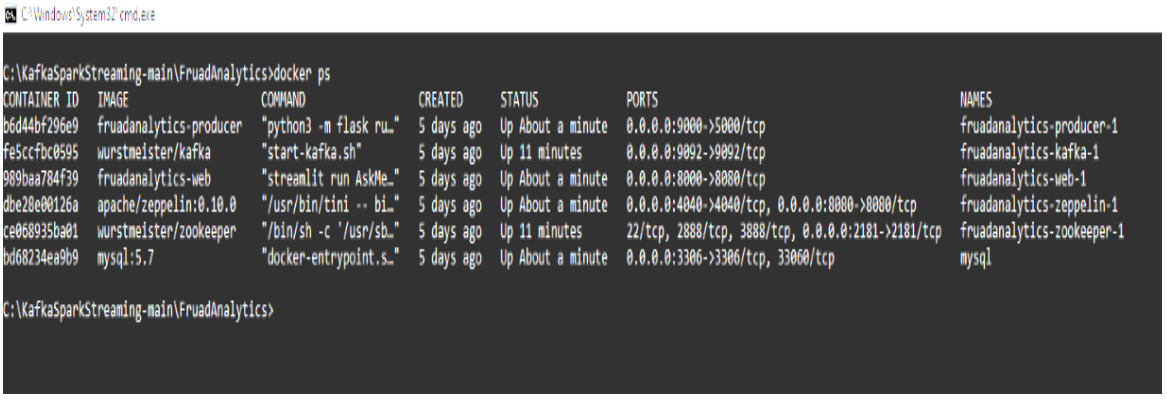
C: > Users > theti > Downloads > KafkaSparkStreaming-main > KafkaSparkStreaming-main > FruadAnalytics > docker-compose.yml

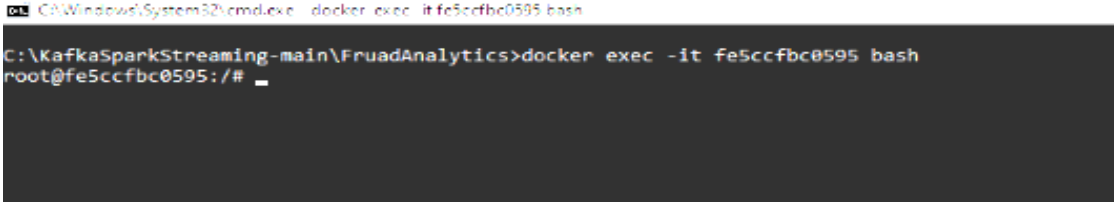
```yaml
version: '2'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    ports:
      - "2181:2181"
    networks:
      - my-net
  kafka:
    image: wurstmeister/kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: kafka
      KAFKA_CREATE_TOPICS: "android:1:1,acceleration:1:1"
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    networks:
      - my-net
    depends_on:
      - zookeeper
  zeppelin:
    ports:
      - 8080:8080
      - 4040:4040
    volumes:
      - "C:/Users/theti/Downloads/KafkaSparkStreaming-main/KafkaSparkStreaming-main/FruadAnalytics/spark-2.4.8-bin-hadoop2.6:/opt/spark"
      - "C:/Users/theti/Downloads/KafkaSparkStreaming-main/KafkaSparkStreaming-main/FruadAnalytics/data:/data"
    environment:
      - SPARK_HOME=/opt/spark
    image: apache/zeppelin:0.10.0
    networks:
      - my-net
  mysql:
    container_name: mysql
```

Do you want to install the r
from Microsoft for docker-


Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.
C:\KafkaSparkStreaming-main\FruadAnalytics>docker-compose up
Attaching to kafka-1, producer-1, web-1, zeppelin-1, zookeeper-1, mysql

2) docker ps


C:\Windows\System32\cmd.exe
C:\KafkaSparkStreaming-main\FruadAnalytics>docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
b6d44bf296e9 fruadanalytics-producer "python3 -m flask ru…" 5 days ago Up About a minute 0.0.0.0:9000->5000/tcp fruadanalytics-producer-1
fe5ccfbc0595 wurstmeister/kafka "start-kafka.sh" 5 days ago Up 11 minutes 0.0.0.0:9092->9092/tcp fruadanalytics-kafka-1
989baa784f39 fruadanalytics-web "streamlit run AskMe…" 5 days ago Up About a minute 0.0.0.0:8000->8080/tcp fruadanalytics-web-1
dbe28e00126a apache/zeppelin:0.10.0 "/usr/bin/tini -- bi…" 5 days ago Up About a minute 0.0.0.0:4040->4040/tcp, 0.0.0.0:8080->8080/tcp fruadanalytics-zeppelin-1
ce068935ba01 wurstmeister/zookeeper "/bin/sh -c '/usr/sb…" 5 days ago Up 11 minutes 22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp fruadanalytics-zookeeper-1
bd68234ea9b9 mysql:5.7 "docker-entrypoint.s…" 5 days ago Up About a minute 0.0.0.0:3306->3306/tcp, 33060/tcp mysql
C:\KafkaSparkStreaming-main\FruadAnalytics>


C:\Windows\System32\cmd.exe - docker exec -it fe5ccfbc0595 bash
C:\KafkaSparkStreaming-main\FruadAnalytics>docker exec -it fe5ccfbc0595 bash
root@fe5ccfbc0595:/#

3) **docker exec -it fe5ccfbc0595 bash**

4) **kafka-topics.sh --create --bootstrap-server localhost:9092 --topic my-topic**

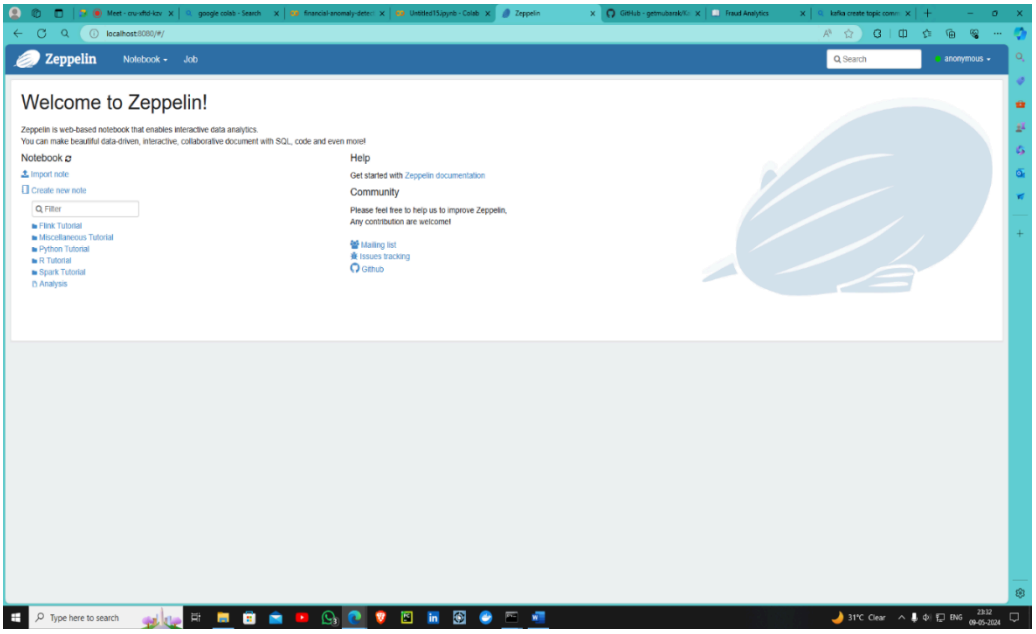
```
C:\KafkaSparkStreaming-main\FruadAnalytics>docker exec -it fe5ccfbc0595 bash
root@fe5ccfbc0595:/# kafka-topics.sh --create --bootstrap-server localhost:9092 --topic my-topic
Created topic my-topic.
root@fe5ccfbc0595:/#
```

5) **kafka-topics.sh --bootstrap-server localhost:9092 --list**

```
C:\KafkaSparkStreaming-main\FruadAnalytics>docker exec -it fe5ccfbc0595 bash
root@fe5ccfbc0595:/# kafka-topics.sh --bootstrap-server localhost:9092 --list
__consumer_offsets
acceleration
android
my-topic
root@fe5ccfbc0595:/#
```

6) **open zeppelin editor (on port 8080)**

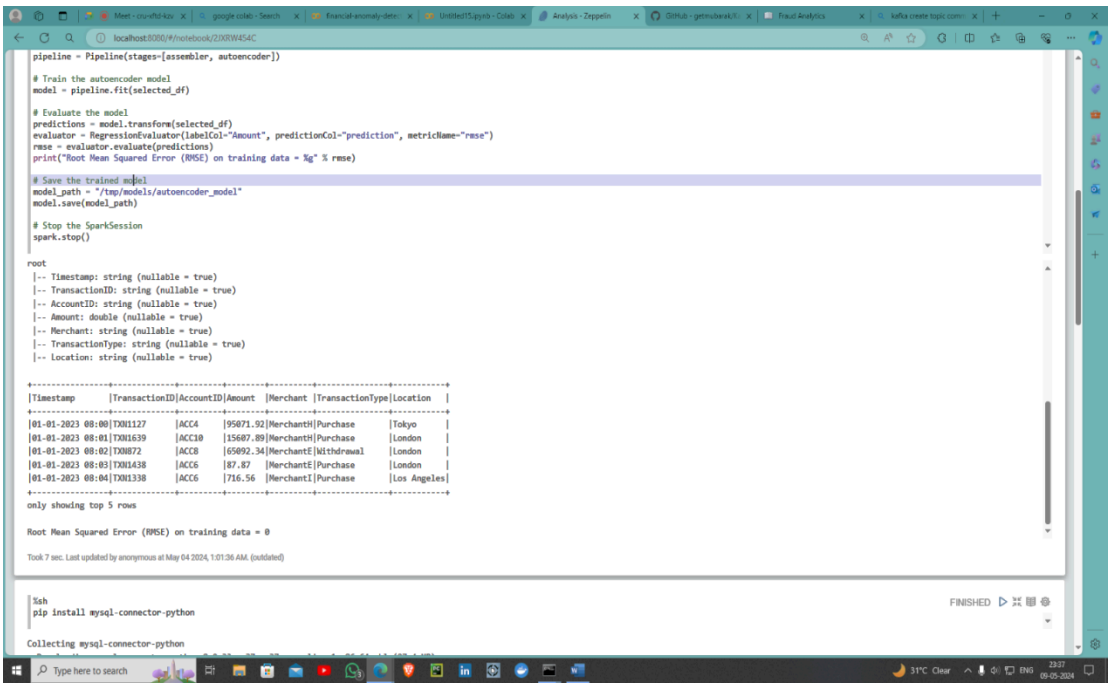
http://localhost:8080/



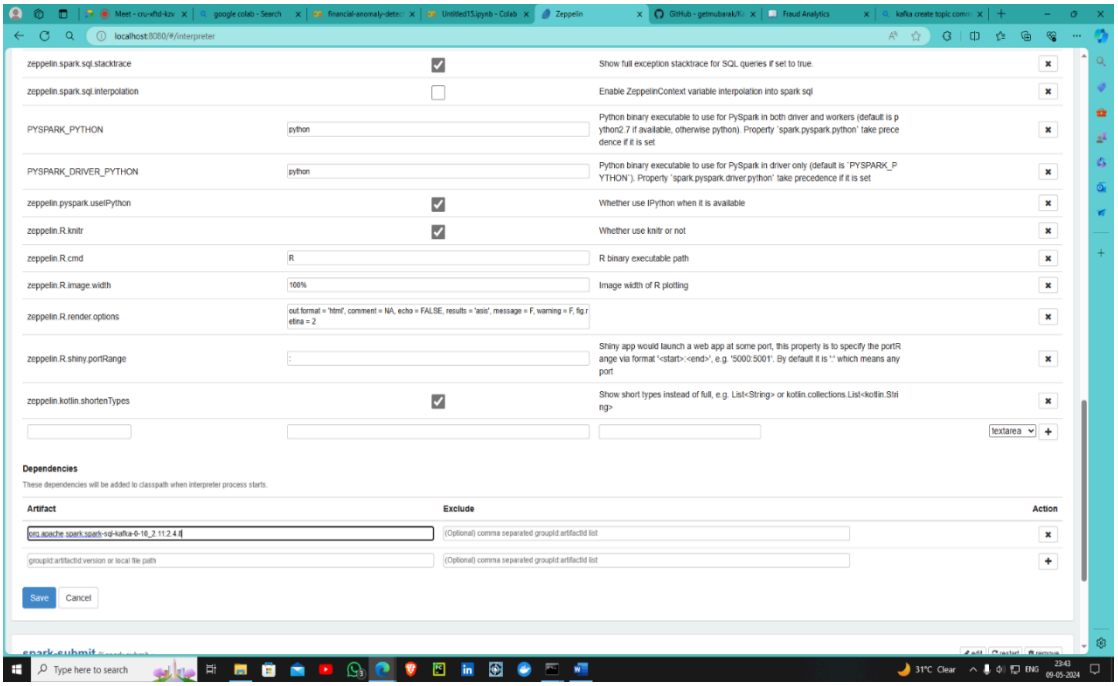
7) Train the model using Linear Regression

```
%spark.pyspark

from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml import Pipeline
from pyspark.sql.functions import from_json, col
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.sql.types import *

# Create SparkSession
spark = SparkSession.builder \
    .appName("AutoencoderTraining") \
    .getOrCreate()

columns = ["Timestamp","TransactionID", "AccountID", "Amount", "Merchant", "TransactionType","Location"]
df = spark.read.csv("/data/financial_anomaly_data.csv",header=True, inferSchema=True)

for i, column in enumerate(columns):
    df = df.withColumnRenamed(df.columns[i], column)
# Display schema and first few rows of the DataFrame
df.printSchema()
df.show(5, truncate=False)

# Data Cleaning and Preprocessing
# Assuming 'Timestamp' is in string format, convert it to datetime
df = df.withColumn("Timestamp", col("Timestamp").cast("timestamp"))

# Filter out rows with null values in 'Amount'
df_cleaned = df.filter(df["Amount"].isNotNull())

#df = df.withColumn("Amount",col("Amount").cast(DecimalType(18,2)))
```

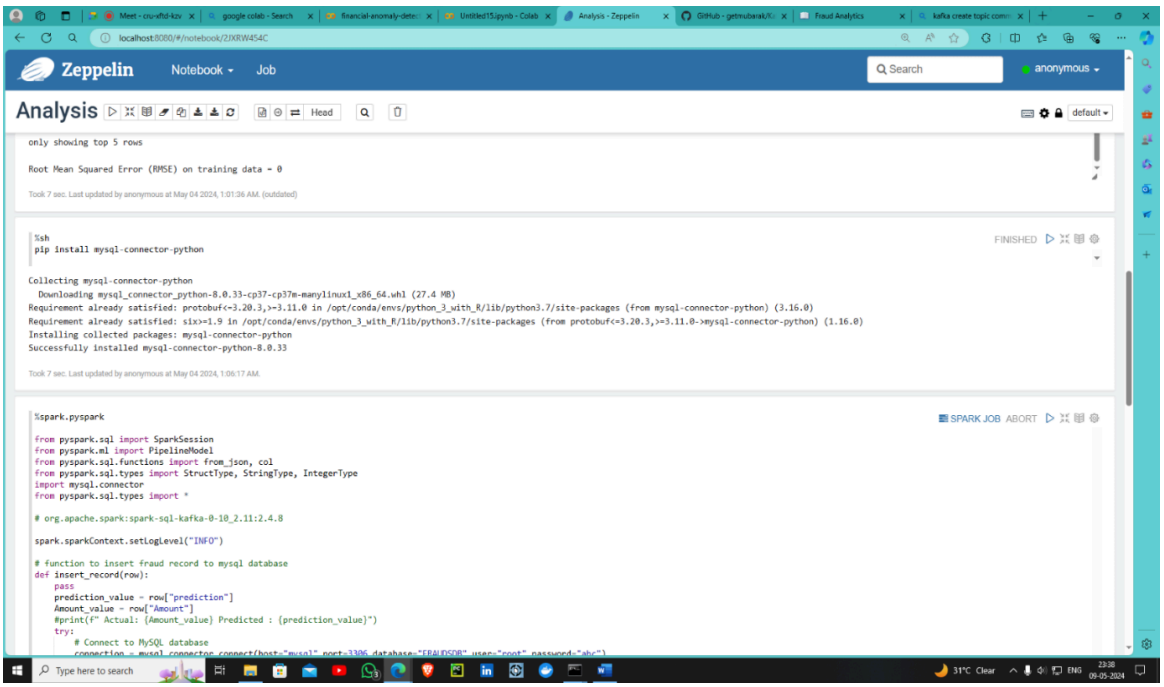
```
 #df = df.withColumn("Amount",col("Amount").cast(DecimalType(18,2)))

 # Assemble features
 assembler = VectorAssembler(inputCols=["Amount"], outputCol="features")

 # Define autoencoder model
 autoencoder = LinearRegression(featuresCol="features", labelCol="Amount")

 # Create pipeline
 pipeline = Pipeline(stages=[assembler, autoencoder])

 # Train the autoencoder model
 model = pipeline.fit(df_cleaned)

 # Evaluate the model
 predictions = model.transform(df_cleaned)
 evaluator = RegressionEvaluator(labelCol="Amount", predictionCol="prediction", metricName="rmse")
 rmse = evaluator.evaluate(predictions)
 print("Root Mean Squared Error (RMSE) on training data = %g" % rmse)

 # Save the trained model
 model_path = "/tmp/models/autoencoder_model"
 model.save(model_path)

 # Stop the SparkSession
 spark.stop()
```



8) Spark interpreter



9) In zeppelin editor install mysql connector



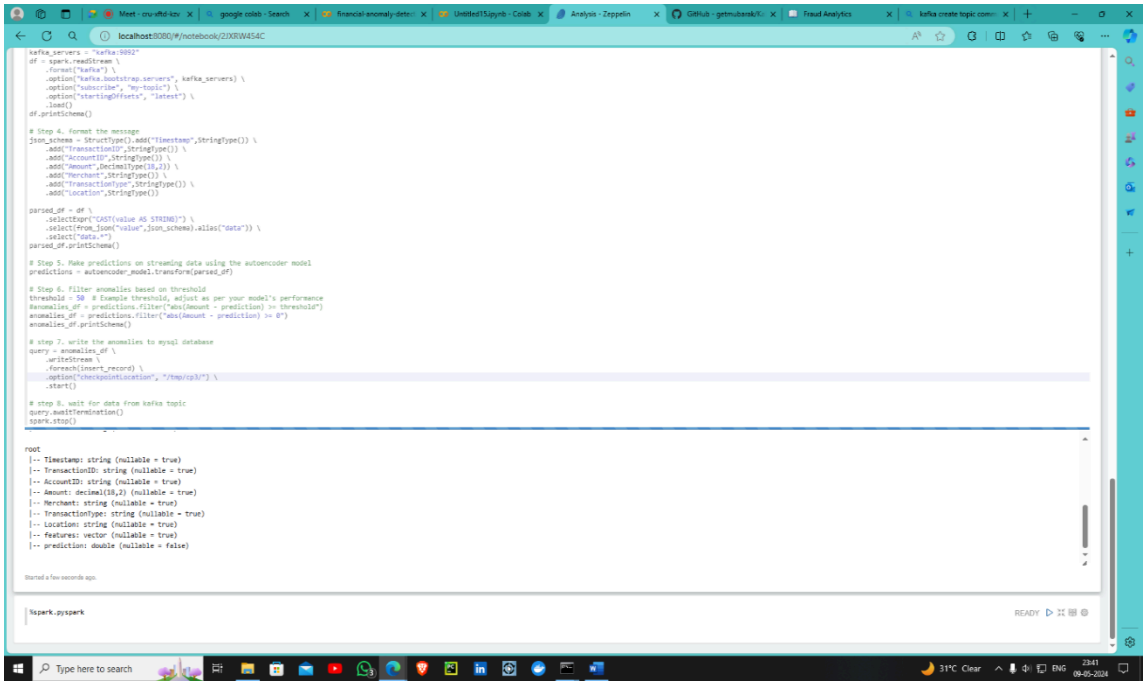
10) Run Spark job

```
%spark.pyspark

from pyspark.sql import SparkSession
from pyspark.ml import PipelineModel
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StringType, IntegerType
import mysql.connector
from pyspark.sql.types import *

# org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.8

spark.sparkContext.setLogLevel("INFO")

# function to insert fraud record to mysql database
def insert_record(row):
    pass
    prediction_value = row["prediction"]
    Amount_value = row["Amount"]
    #print(f" Actual: {Amount_value} Predicted : {prediction_value}")
    try:
        # Connect to MySQL database
        connection = mysql.connector.connect(host="mysql",port=3306,database="FRAUDSDB",user="root",password="abc")

        if not connection.is_connected():
            print("failed to connect to sql")
            return

        print("inserting record" + row["TransactionID"])
        sql_insert_query = "INSERT INTO fraudtrans VALUES ('" +  row["Timestamp"] + "','" + \
        row["TransactionID"] + "','" + \
        row["AccountID"] + "'," + \
```

```
        row["AccountID"] + "'," + \
        str(row["Amount"]) + ",'" + \
        row["Merchant"] + "','" + \
        row["TransactionType"] + "','" + \
        row["Location"] + "')"

        print(sql_insert_query)
        # Create a cursor object to execute SQL queries
        cursor = connection.cursor()
        cursor.execute(sql_insert_query)
        connection.commit()

    except mysql.connector.Error as error:
        print("Error while connecting to MySQL", error)
    finally:
        if 'connection' in locals() and connection.is_connected():
            cursor.close()
            connection.close()

# Step 1. Create Spark Session
# a SparkSession is a unified entry point for working with structured data. It provides a way to interact with various Spark funct
appName = "Kafka Examples2"
master = "local"
spark = SparkSession.builder \
    .master(master) \
    .appName(appName) \
    .getOrCreate()

# Step 2. Load the pre-trained model
model_path = "/tmp/models/autoencoder_model"
autoencoder_model = PipelineModel.load(model_path)
```

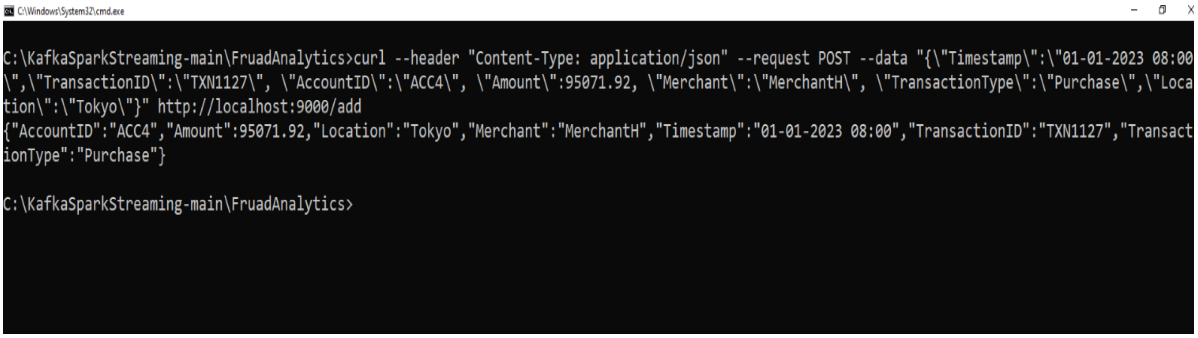
```
autoencoder_model = PipelineModel.load(model_path)

# Step 3. Connect and read from kafka topic
kafka_servers = "kafka:9092"
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_servers) \
    .option("subscribe", "my-topic") \
    .option("startingOffsets", "latest") \
    .load() \
df.printSchema()

# Step 4. format the message
json_schema = StructType().add("Timestamp",StringType()) \
    .add("TransactionID",StringType()) \
    .add("AccountID",StringType()) \
    .add("Amount",DecimalType(18,2)) \
    .add("Merchant",StringType()) \
    .add("TransactionType",StringType()) \
    .add("Location",StringType())

parsed_df = df \
    .selectExpr("CAST(value AS STRING)") \
    .select(from_json("value",json_schema).alias("data")) \
    .select("data.*")
parsed_df.printSchema()

# Step 5. Make predictions on streaming data using the autoencoder model
predictions = autoencoder_model.transform(parsed_df)

# Step 6. Filter anomalies based on threshold
```

```
# Step 6. Filter anomalies based on threshold
threshold = 50  # Example threshold, adjust as per your model's performance
#anomalies_df = predictions.filter("abs(Amount - prediction) >= threshold")
anomalies_df = predictions.filter("abs(Amount - prediction) >= 0")
anomalies_df.printSchema()

# step 7. write the anomalies to mysql database
query = anomalies_df \
    .writeStream \
    .foreach(insert_record) \
    .option("checkpointLocation", "/tmp/cp/") \
    .start()

# step 8. wait for data from kafka topic
query.awaitTermination()
spark.stop()
```



11) run producer (on port 9000)

curl --header "Content-Type: application/json" --request POST --data "{\"Timestamp\":\"01-01-2023 08:00\",\"TransactionID\":\"TXN1127\", \"AccountID\":\"ACC4\", \"Amount\":95071.92, \"Merchant\":\"MerchantH\", \"TransactionType\":\"Purchase\",\"Location\":\"Tokyo\"}" http://localhost:9000/add



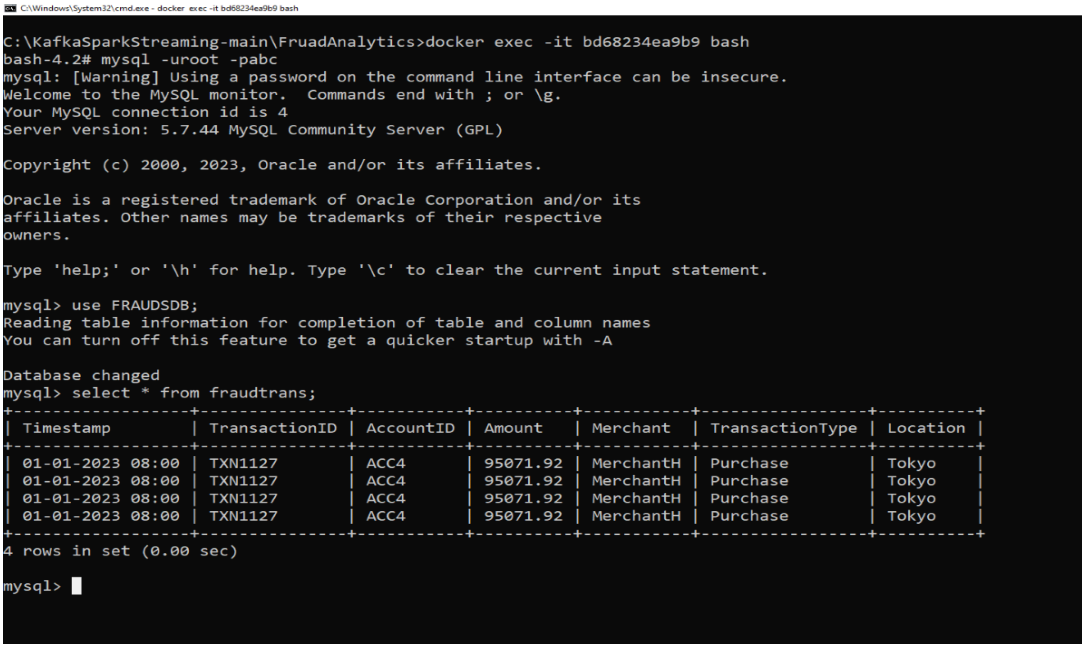
12) check data in SQL

docker exec -it bd68234ea9b9 bash

mysql -uroot -pabc

use FRAUDSDB;

select * from fraudtrans;



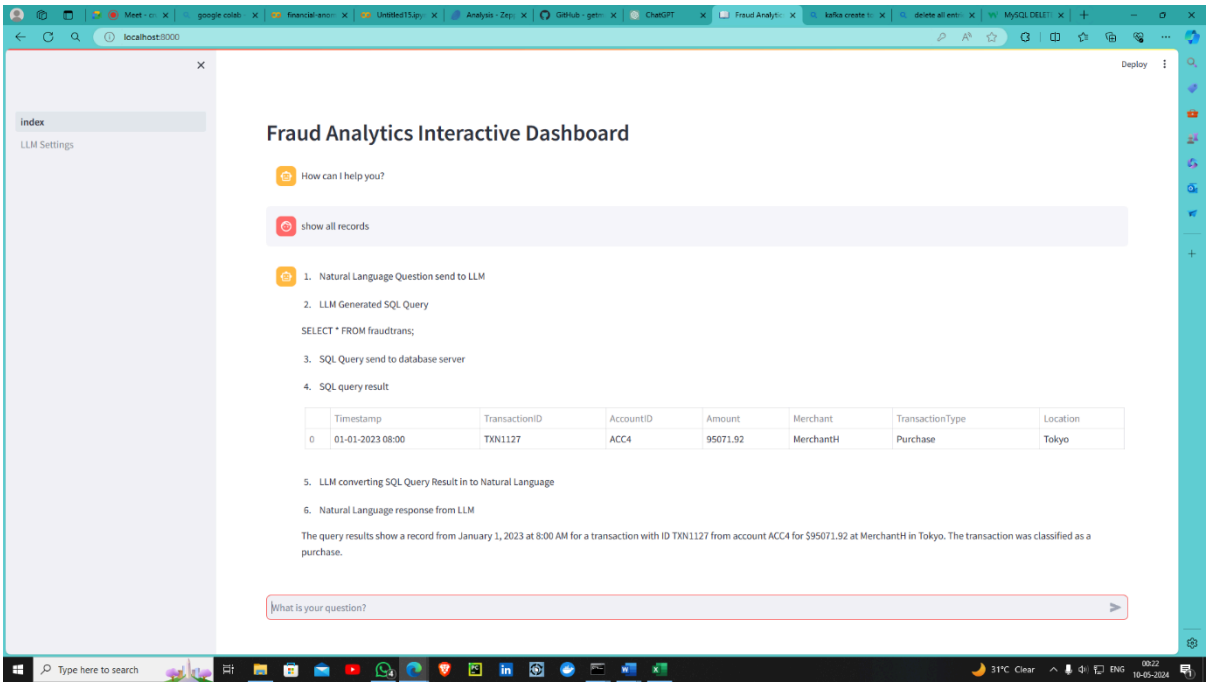
13) see output in web App (on port 8000)

http://localhost:8000

- **configure llm key**
- **run query "show all records"**

Web Application (on port 8000)

- View Data



Trail and testing on different Methods.

Source code using K-Means:

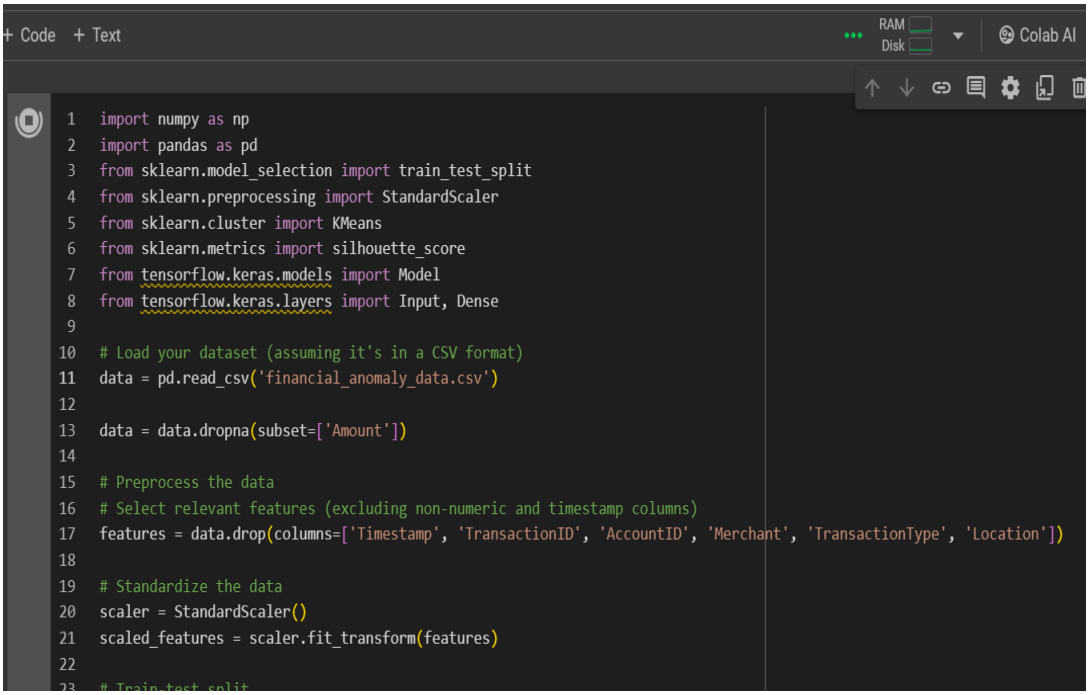
```
%spark.pyspark

from pyspark.sql import SparkSession
from pyspark.ml import PipelineModel
from pyspark.sql.functions import from_json, col
from pyspark.sql.types import StructType, StringType, IntegerType
import mysql.connector
from pyspark.sql.types import *
from pyspark.ml.feature import VectorAssembler, OneHotEncoderEstimator, StringIndexer
from pyspark.ml import Pipeline, PipelineModel

# org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.8

spark.sparkContext.setLogLevel("INFO")

# function to insert fraud record to mysql database
def insert_record(row):
    pass
    prediction_value = row["prediction"]
    Amount_value = row["Amount"]
    #print(f" Actual: {Amount_value} Predicted : {prediction_value}")
    try:
        # Connect to MySQL database
        connection = mysql.connector.connect(host="mysql",port=3306,database="FRAUDSDB",user="root",password="a

        if not connection.is_connected():
            print("failed to connect to sql")
            return

        print("inserting record" + row["TransactionID"])
        sql_insert_query = "INSERT INTO fraudtrans VALUES ('" +  row["Timestamp"] + "','" + \
```

```
# Step 2. Load the pre-trained model
pipelineModel_path = "/tmp/models/clustering_model"
pipelineModel = PipelineModel.load(pipelineModel_path)

# Step 3. Connect and read from kafka topic
kafka_servers = "kafka:9092"
df = spark.readStream \
    .format("kafka") \
    .option("kafka.bootstrap.servers", kafka_servers) \
    .option("subscribe", "my-topic") \
    .option("startingOffsets", "latest") \
    .load()
df.printSchema()

# Step 4. format the message
json_schema = StructType().add("Timestamp",StringType()) \
    .add("TransactionID",StringType()) \
    .add("AccountID",StringType()) \
    .add("Amount",DecimalType(18,2)) \
    .add("Merchant",StringType()) \
    .add("TransactionType",StringType()) \
    .add("Location",StringType())

parsed_df = df \
    .selectExpr("CAST(value AS STRING)") \
    .select(from_json("value",json_schema).alias("data")) \
    .select("data.*")
parsed_df.printSchema()

# Convert categorical columns to numerical using StringIndexer
indexers = [StringIndexer(inputCol=column, outputCol=column+"_index", handleInvalid="keep")
            for column in ["Merchant", "TransactionType", "Location"]]

# One-hot encode the indexed categorical columns
encoder = OneHotEncoderEstimator(inputCols=[indexer.getOutputCol() for indexer in indexers],
                                 outputCols=["Merchant_encoded", "TransactionType_encoded", "Location_encoded"])

# Assemble features
assembler = VectorAssembler(inputCols=["Amount", "Merchant_encoded",
                                       "TransactionType_encoded", "Location_encoded"],
                            outputCol="features")

# Step 5. Make predictions on streaming data using the autoencoder model
predictions = pipelineModel.transform(parsed_df)

# Step 6. Filter anomalies based on threshold
threshold = 50  # Example threshold, adjust as per your model's performance
#anomalies_df = predictions.filter("abs(Amount - prediction) >= threshold")
anomalies_df = predictions.filter("abs(Amount - prediction) >= 0")
anomalies_df.printSchema()

# step 7. write the anomalies to mysql database
query = anomalies_df \
    .writeStream \
    .foreach(insert_record) \
    .option("checkpointLocation", "/tmp/cp/") \
    .start()
```

OUTPUT

```
Silhouette Score: 0.6267644600544695
Anomalies:
                Timestamp TransactionID AccountID    Amount   Merchant  \
3       01-01-2023 08:03       TXN1438      ACC6     87.87  MerchantE
4       01-01-2023 08:04       TXN1338      ACC6    716.56  MerchantI
9       01-01-2023 08:09       TXN1479     ACC12  49522.74  MerchantC
15      01-01-2023 08:15         TXN65      ACC9  98688.82  MerchantH
34      01-01-2023 08:34        TXN113      ACC2  50272.88  MerchantF
...                  ...           ...       ...       ...        ...
216840  31-05-2023 22:00       TXN1318      ACC8    138.05  MerchantG
216853  31-05-2023 22:13       TXN1234     ACC11  51082.70  MerchantA
216896  31-05-2023 22:56       TXN1628     ACC10  48790.06  MerchantC
216934  31-05-2023 23:34       TXN1273     ACC11  99077.99  MerchantG
216948  31-05-2023 23:48        TXN863      ACC3  51110.59  MerchantB

       TransactionType       Location
3             Purchase         London
4             Purchase    Los Angeles
9           Withdrawal       New York
15            Purchase         London
34            Transfer          Tokyo
...                ...            ...
216840        Purchase         London
216853      Withdrawal         London
216896        Transfer         London
216934        Transfer       New York
216948      Withdrawal  San Francisco
```

Source code: MLModelTrainAutoEncoderKMeans



```
+ Code + Text                                                    RAM Disk   Colab AI

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense

# Load your dataset (assuming it's in a CSV format)
data = pd.read_csv('financial_anomaly_data.csv')

data = data.dropna(subset=['Amount'])

# Preprocess the data
# Select relevant features (excluding non-numeric and timestamp columns)
features = data.drop(columns=['Timestamp', 'TransactionID', 'AccountID', 'Merchant', 'TransactionType', 'Location'])

# Standardize the data
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

# Train-test split
```

```python
# Train-test split
X_train, X_test = train_test_split(scaled_features, test_size=0.2, random_state=42)

# Define the autoencoder architecture
input_dim = X_train.shape[1]
encoding_dim = 32  # Choose the size of the encoding layer as per your requirement

input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation='relu')(input_layer)
decoder = Dense(input_dim, activation='relu')(encoder)

# Define the autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder)

# Compile the model
autoencoder.compile(optimizer='adam', loss='mse')

# Train the autoencoder
autoencoder.fit(X_train, X_train, epochs=50, batch_size=128, shuffle=True, validation_data=(X_test, X_tes

# Get the encoded representations
encoded_features = autoencoder.predict(scaled_features)

# Apply k-means clustering on the encoded representations
kmeans = KMeans(n_clusters=2, random_state=42)
```

```python
# Apply k-means clustering on the encoded representations
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(encoded_features)

# Evaluate clustering performance using silhouette score
silhouette_avg = silhouette_score(encoded_features, kmeans.labels_)
print(f"Silhouette Score: {silhouette_avg}")

# Identify anomalies based on clustering
cluster_centers = kmeans.cluster_centers_
distances = np.linalg.norm(encoded_features - cluster_centers[kmeans.labels_], axis=1)
threshold = np.percentile(distances, 95)

# Find anomalies
anomalies = data[distances > threshold]
print("Anomalies:")
print(anomalies)
```

```
Epoch 1/50
1356/1356 [==============================] - 4s 2ms/step - loss: 0.4981 - val_loss: 0.4866
Epoch 2/50
1356/1356 [==============================] - 2s 2ms/step - loss: 0.4952 - val_loss: 0.4866
Epoch 3/50
1356/1356 [==============================] - 3s 2ms/step - loss: 0.4952 - val_loss: 0.4866
Epoch 4/50
1356/1356 [==============================] - 4s 3ms/step - loss: 0.4952 - val_loss: 0.4866
```

OUTPUT

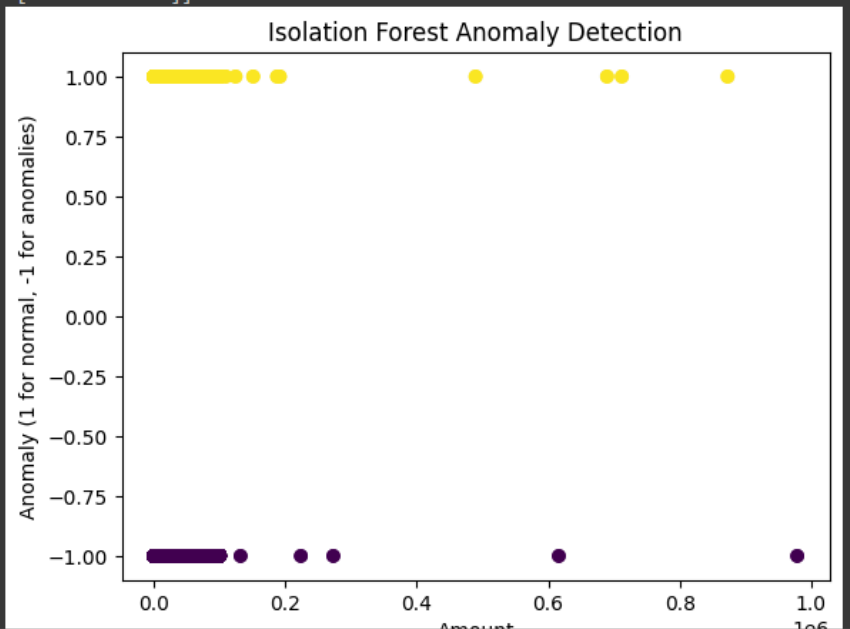
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` wi
  warnings.warn(
Silhouette Score: 0.7736548781394958
Anomalies:
                Timestamp TransactionID AccountID   Amount   Merchant  \
15     01-01-2023 08:15        TXN65      ACC9  98688.82  MerchantH
39     01-01-2023 08:39       TXN187      ACC6  69584.32  MerchantC
91     01-01-2023 09:31      TXN1804      ACC1  67987.68  MerchantG
106    01-01-2023 09:46      TXN1959      ACC1  68107.13  MerchantI
115    01-01-2023 09:55      TXN1728     ACC11  69819.19  MerchantA
...                 ...         ...       ...       ...        ...
216870 31-05-2023 22:30       TXN275     ACC15  69058.74  MerchantH
```

Source Code : Isolation Forest Anomaly Detection

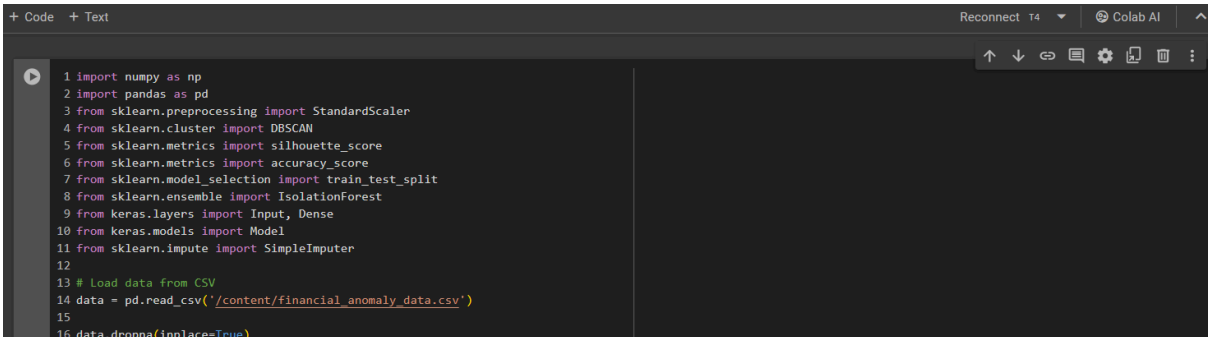
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import classification_report, confusion_matrix

# Load the dataset
df = pd.read_csv('/content/financial_anomaly_data.csv')

# Convert 'Timestamp' column to datetime format
df['Timestamp'] = pd.to_datetime(df['Timestamp'], format='%d-%m-%Y %H:%M')

# Use OneHotEncoder for categorical variables
encoder = OneHotEncoder(sparse=False, drop='first')
encoded_cols = encoder.fit_transform(df[['Merchant', 'TransactionType', 'Location']])
feature_names = encoder.get_feature_names_out(['Merchant', 'TransactionType', 'Location'])
encoded_df = pd.DataFrame(encoded_cols, columns=feature_names)

# Feature engineering - drop original categorical columns
df = pd.concat([df, encoded_df], axis=1)
df.drop(['Merchant', 'TransactionType', 'Location'], axis=1, inplace=True)
df.dropna(inplace=True)

# Fit Isolation Forest model
model = IsolationForest(contamination=0.2, random_state=42)
model.fit(df[['Amount'] + list(encoded_df.columns)])

# Predict anomalies (1 for normal, -1 for anomalies)
df['Anomaly'] = model.predict(df[['Amount'] + list(encoded_df.columns)])

# Read the CSV file containing the new transaction
new_transaction = pd.read_csv('/content/data.csv')

# Convert 'Timestamp' column to datetime format
new_transaction['Timestamp'] = pd.to_datetime(new_transaction['Timestamp'], format='%d-%m-%Y %H:%M')

# Use OneHotEncoder for categorical variables
encoded_cols = encoder.transform(new_transaction[['Merchant', 'TransactionType', 'Location']])
encoded_df = pd.DataFrame(encoded_cols, columns=feature_names)

# Feature engineering - drop original categorical columns
new_transaction = pd.concat([new_transaction, encoded_df], axis=1)
new_transaction.drop(['Merchant', 'TransactionType', 'Location'], axis=1, inplace=True)
```

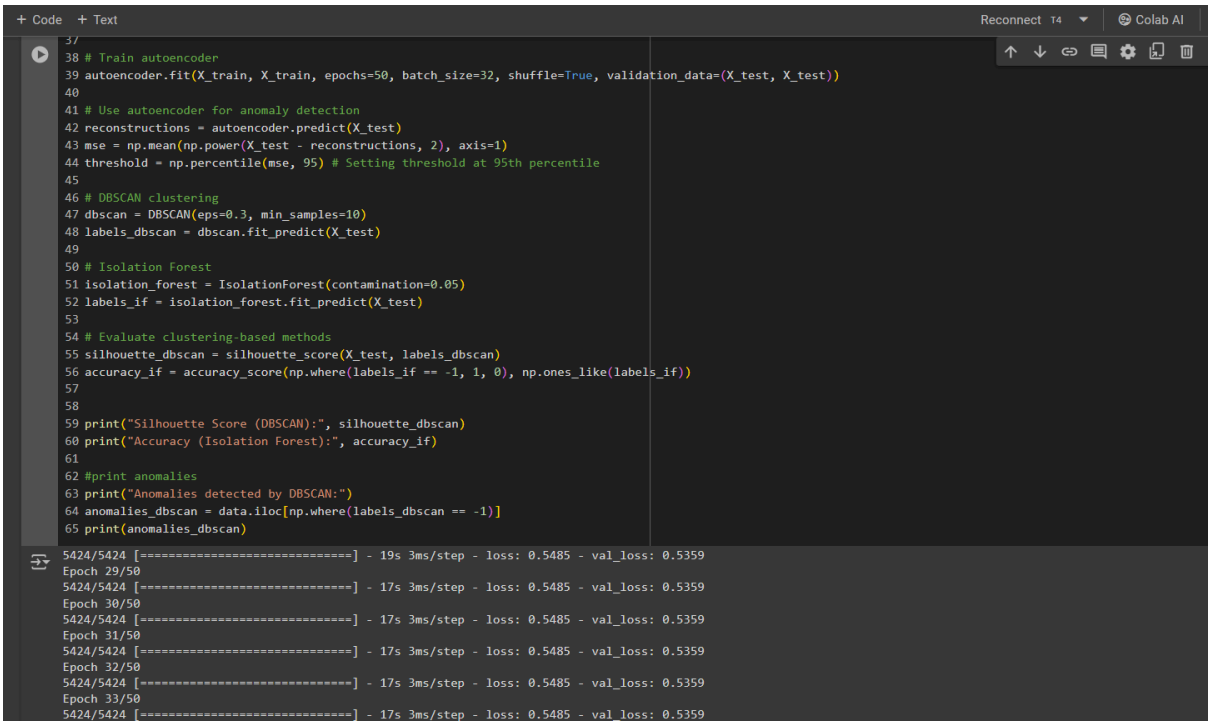
Output:

```
Classification Report:
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00     43406
           1       1.00      1.00      1.00    173594

    accuracy                           1.00    217000
   macro avg       1.00      1.00      1.00    217000
weighted avg       1.00      1.00      1.00    217000


Confusion Matrix:
[[ 43406      0]
 [     0 173594]]
```



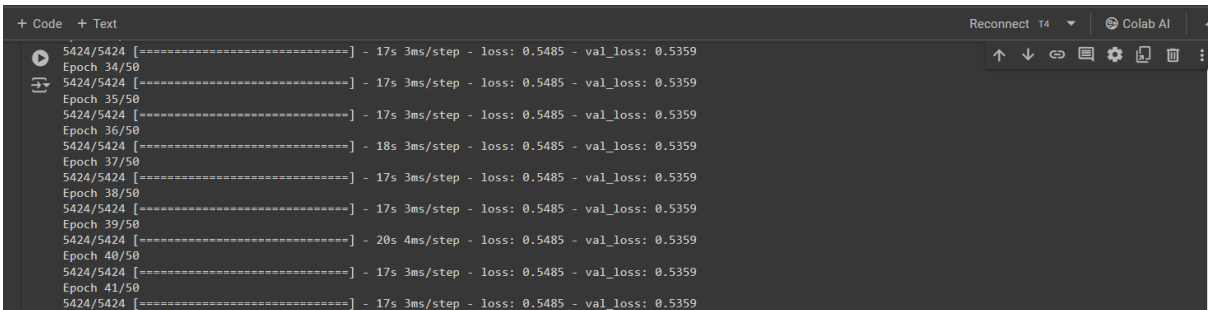
Source Code : DBScan Autoencoder Forest Anomaly Detection





```
+ Code + Text                                                    Reconnect T4   Colab AI

# Train autoencoder
autoencoder.fit(X_train, X_train, epochs=50, batch_size=32, shuffle=True, validation_data=(X_test, X_test))

# Use autoencoder for anomaly detection
reconstructions = autoencoder.predict(X_test)
mse = np.mean(np.power(X_test - reconstructions, 2), axis=1)
threshold = np.percentile(mse, 95) # Setting threshold at 95th percentile

# DBSCAN clustering
dbscan = DBSCAN(eps=0.3, min_samples=10)
labels_dbscan = dbscan.fit_predict(X_test)

# Isolation Forest
isolation_forest = IsolationForest(contamination=0.05)
labels_if = isolation_forest.fit_predict(X_test)

# Evaluate clustering-based methods
silhouette_dbscan = silhouette_score(X_test, labels_dbscan)
accuracy_if = accuracy_score(np.where(labels_if == -1, 1, 0), np.ones_like(labels_if))


print("Silhouette Score (DBSCAN):", silhouette_dbscan)
print("Accuracy (Isolation Forest):", accuracy_if)

#print anomalies
print("Anomalies detected by DBSCAN:")
anomalies_dbscan = data.iloc[np.where(labels_dbscan == -1)]
print(anomalies_dbscan)

5424/5424 [==============================] - 19s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 29/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 30/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 31/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 32/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 33/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
```

OUTPUT



```
+ Code + Text                                                    Reconnect T4   Colab AI

5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 34/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 35/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 36/50
5424/5424 [==============================] - 18s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 37/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 38/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 39/50
5424/5424 [==============================] - 20s 4ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 40/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
Epoch 41/50
5424/5424 [==============================] - 17s 3ms/step - loss: 0.5485 - val_loss: 0.5359
```

CHAPTER-5

CONCLUSION & FUTURE ENHANCEMENT

Nowadays, in the global computing environment, online payments are important, because online payments use only the credential information from the credit card to fulfil an application and then deduct money. Due to this reason, it is important to find the best solution to detect the maximum number of frauds in online systems. Accuracy, Error-rate, Sensitivity and Specificity are used to report the performance of the system to detect the fraud in the bank transactions. In this project three machine learning algorithms are developed to detect the fraud.

Future Enhancement Detection, we did end up creating a system that can, with enough time and data, get very close to that goal. As with any such project, there is some room for improvement here. The very nature of this project allows for multiple algorithms to be integrated together as modules and their results can be combined to increase the accuracy of the final result. This model can further be improved with the addition of more algorithms into it. However, the output of these algorithms needs to be in the same format as the others. Once that condition is satisfied, the modules are easy to add as done in the code. This provides a great degree of modularity and versatility to the project. More room for improvement can be found in the dataset. As demonstrated before, the precision of the algorithms increases when the size of dataset is increased. Hence, more data will surely make the model more accurate in detecting frauds and reduce the number of false positives. However, this requires official support from the banks themselves.
Some suggestions:
Here are some potential future enhancements for fraud detection systems:

1. Advanced Machine Learning Models: Continuously explore and incorporate state-of-the-art machine learning models for anomaly detection. This could include deep learning architectures such as autoencoders, recurrent neural networks (RNNs), or transformers, which might capture more complex patterns in the data.

2. Feature Engineering: Invest in more sophisticated feature engineering techniques to extract meaningful insights from the data. This could involve incorporating domain knowledge, creating new features through dimensionality reduction techniques, or leveraging unsupervised learning methods for feature extraction.

3. Behavioural Analysis: Develop algorithms to model user behaviour over time and detect deviations from normal behaviour. This could involve sequence modelling techniques such as Markov models or recurrent neural networks to capture temporal dependencies in user actions.

4. Graph Analytics: Explore graph-based approaches to fraud detection, especially in scenarios where fraudulent behavior can be represented as anomalous connections or patterns in a network. Graph algorithms such as community detection, centrality measures, or anomaly detection on graphs can be applied to detect suspicious patterns.

5. Real-time Monitoring and Alerting: Enhance real-time monitoring capabilities to quickly identify and respond to emerging fraud patterns. This could involve integrating streaming analytics platforms or implementing custom alerting mechanisms to notify stakeholders of suspicious activities as they occur.

7. Adversarial Defense Mechanisms: Develop robustness against adversarial attacks aimed at evading fraud detection systems. This could involve techniques such as adversarial training, model diversification, or anomaly detection methods specifically designed to detect adversarial behavior.

9. Dynamic Risk Scoring: Implement dynamic risk scoring mechanisms that adaptively adjust risk scores based on evolving fraud patterns and contextual factors. This can help prioritize alerts and interventions for high-risk transactions or activities while minimizing false positives.

BIBLIOGRAPHY

1. B.Meena, I.S.L.Sarwani, S.V.S.S.Lakshmi,” Web Service mining and its techniques in Web Mining” IJAEGT,Volume 2,Issue 1 , Page No.385-389.
2. F. N. Ogwueleka, "Data Mining Application in Credit Card Fraud Detection System", Journal of Engineering Science and Technology, vol. 6, no. 3, pp. 311-322, 2019.
3. G. Singh, R. Gupta, A. Rastogi, M. D. S. Chandel, A. Riyaz, "A Machine Learning Approach for Detection of Fraud based on SVM", International Journal of Scientific Engineering and Technology, vol. 1, no. 3, pp. 194-198, 2019, ISSN ISSN: 2277-1581.
4. K. Chaudhary, B. Mallick, "Credit Card Fraud: The study of its impact and detection techniques", International Journal of Computer Science and Network (IJCSN), vol. 1, no. 4, pp. 31-35, 2019, ISSN ISSN: 2277-5420. M. J. Islam, Q. M. J. Wu, M. Ahmadi, M. A. Sid- Ahmed, "Investigating the Performance of Naive-Bayes Classifiers and KNearestNeighbor Classifiers", IEEE International Conference on Convergence Information Technology, pp. 1541-1546, 2017.
5. R. Wheeler, S. Aitken, "Multiple algorithms for fraud detection" in Knowledge-Based Systems, Elsevier, vol. 13, no. 2, pp. 93-99, 2018.
6. Patil, H. Somavanshi, J. Gaikwad, A. Deshmane, R. Badgujar, "Credit Card Fraud Detection Using Decision Tree Induction Algorithm", International Journal of Computer Science and Mobile Computing (IJCSMC), vol. 4, no. 4, pp. 92-95, 2020, ISSN ISSN: 2320-088X.
7. S. Maes, K. Tuyls, B. Vanschoenwinkel, B. Manderick,"Credit card fraud detection using Bayesian and neural networks", Proceedings of the 1st international naiso congress on neuro fuzzy technologies, pp. 261-270, 2017.
8. S. Bhattacharyya, S. Jha, K. Tharakunnel, J. C.Westland, "Data mining for credit card fraud: A comparative study", Decision Support Systems, vol. 50, no. 3, pp. 602-613, 2019. [10]
9. Y. Sahin, E. Duman, "Detecting credit card fraud by ANN and logistic regression", Innovations in Intelligent Systems and Applications (INISTA) 2018 International Symposium, pp. 315-319, 2018.