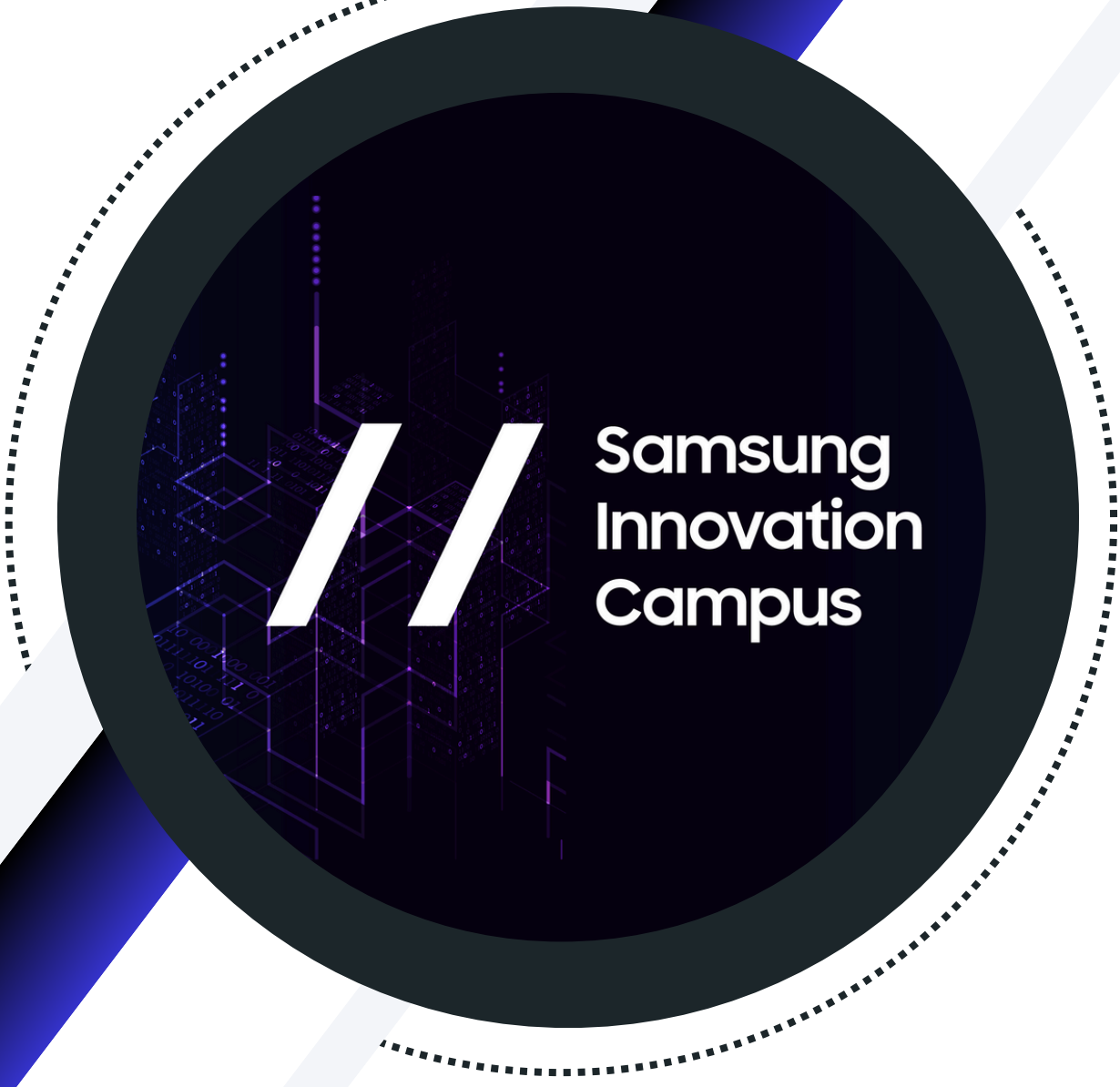
GRADUATION PROJECT REPORT

SIC- 2024

BIG DATA 601



Samsung Innovation Campus

Fraud Detection: Batch and Streaming Processing

Samsung Innovation Campus

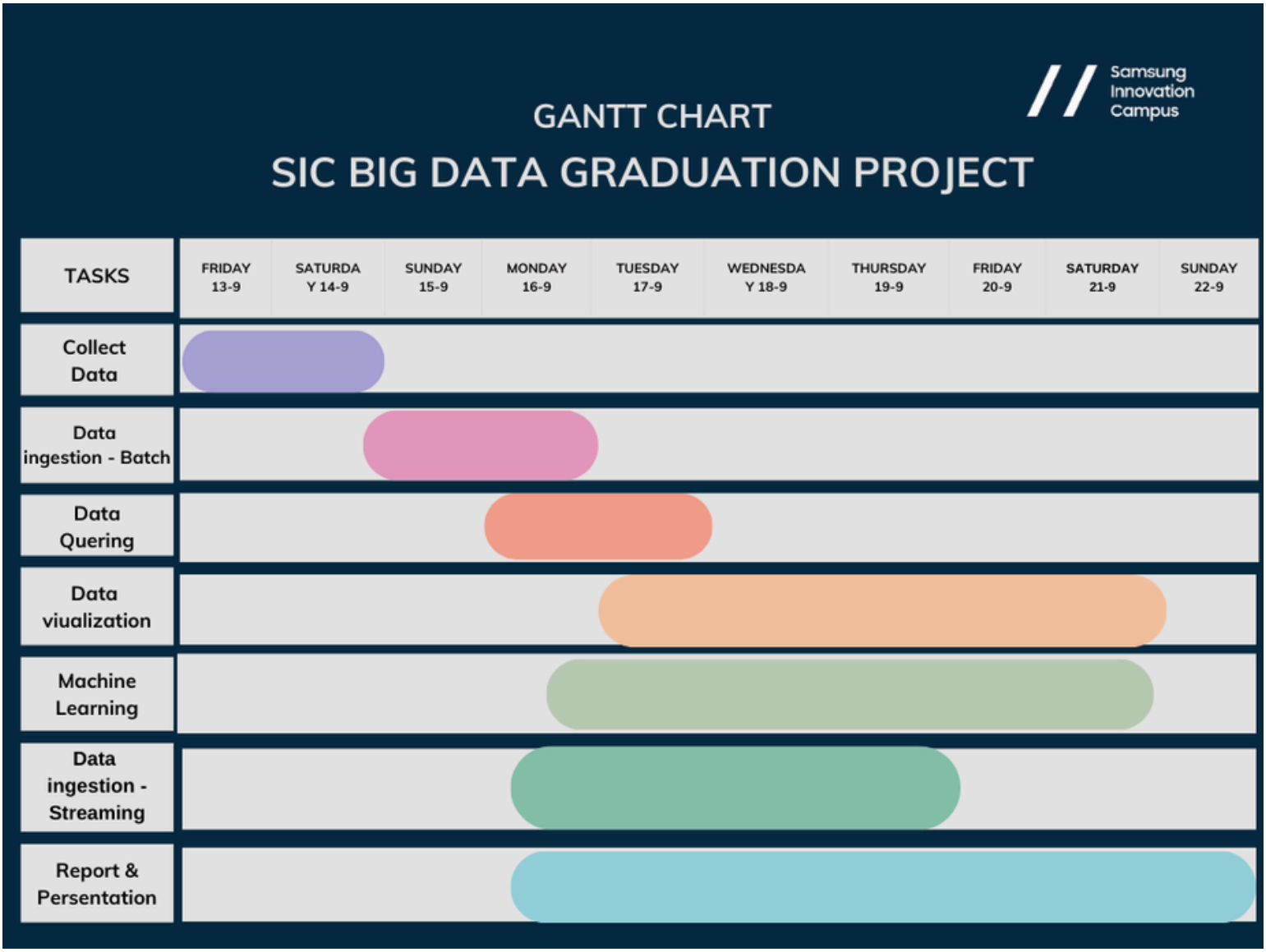
Team Members
Karim Sherif
Salma Mamdoh
Eman AbelSayed
Mohamed Walid
Jana Tamer

Facilitator

Abeer Elmorshedy

Samsung Innovation Campus

PROJECT GANTT CHART



Samsung Innovation Campus

PROJECT OBJECTIVES

To develop a robust batch processing system capable of detecting financial fraud by identifying suspicious transaction patterns in large datasets. By leveraging technologies such as Hadoop HDFS, Apache Spark, Hive, and machine learning algorithms, the system will efficiently process and transform historical financial data, detect anomalies, and generate timely alerts for potentially fraudulent activities. Additionally, the project seeks to provide insights through data visualization and reporting, enabling enhanced decision-making and proactive fraud prevention.

Importance in Business:

This project strengthens financial security by enabling real-time fraud detection, reducing financial losses, and protecting customer trust. It helps businesses identify fraudulent activities quickly, enhancing operational efficiency and compliance with regulations. The use of machine learning and data visualization provides valuable insights into fraud patterns, allowing businesses to refine their risk management strategies. Overall, it promotes better decision-making, optimizes processes, and contributes to long-term financial stability.

Samsung Innovation Campus

DATA DESCRIPTION

Data Retrieval

The dataset utilized for this project was sourced from Kaggle's "Fraud Detection" dataset. It contains labeled transaction data, which is crucial for training and testing fraud detection models. The dataset is publicly accessible .

Dataset Description

This project employs a simulated credit card transaction dataset, spanning the period from January 1, 2019, to December 31, 2020. It includes both legitimate and fraudulent transactions from 1,000 credit card holders who interacted with 800 merchants. The dataset provides detailed transactional information that is essential for developing and training fraud detection models. The dataset consists of over 1 million records with 23 attributes.

Samsung Innovation Campus

DATA DESCRIPTION

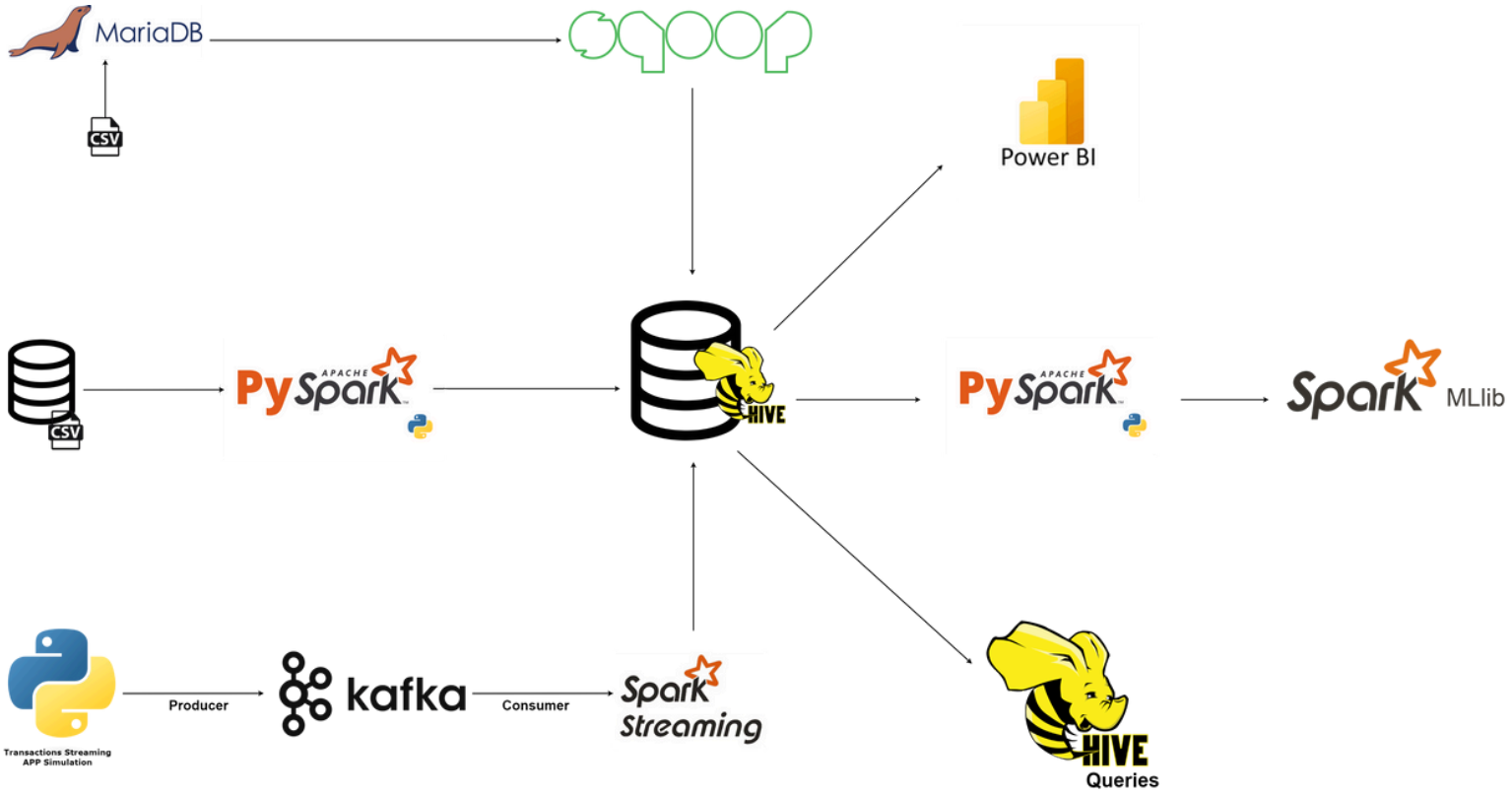
Variable	Description
index	Unique identifier for each transaction record.
trans_date_trans_time	The date and time of the transaction.
c_num	Credit card number of the customer.
merchant	Name of the merchant involved in the transaction.
category	Category of the merchant (e.g., groceries, entertainment).
amt	The transaction amount.
first	First name of the credit card holder.
last	Last name of the credit card holder.
gender	Gender of the credit card holder.
street	Street address of the credit card holder.
city	City where the credit card holder resides.
state	State where the credit card holder resides.
zip	Zip code of the credit card holder.
lat	Latitude coordinate of the credit card holder's location.

Samsung Innovation Campus

DATA DESCRIPTION

Variable	Description
long	Longitude coordinate of the credit card holder's location.
city_pop	Population of the city where the credit card holder resides.
job	Occupation of the credit card holder.
dob	Date of birth of the credit card holder.
trans_num	Unique transaction identifier.
unix_time	UNIX timestamp of the transaction.
merch_lat	Latitude coordinate of the merchant's location.
merch_long	Longitude coordinate of the merchant's location.
is_fraud	Fraud flag indicating whether the transaction is fraudulent (1) or legitimate (0). This serves as the target variable for fraud detection models.

PROJECT ARCHITECTURE



Samsung Innovation Campus

WORKFLOW OVERVIEW

- **Data Ingestion:** Data is ingested from Relational Database, Streams of json data, and CSV File into hive database.
- **Data Integration**: data integrated into unified Schema to be stored into single hive table.
- **Data storage:** data have been stored into hive table Stored on hdfs.
- **Data Querying**: Perform Querires .and analysis to get useful insights from the data.
- **Data Visualization**: Power BI connects to data to create visual reports.
- **Machine Learning**: PySpark, along with Spark MLlib, is used to build and deploy machine learning models that can detect fraudulent activities.

DATA SOURCES

1.Relational Database (Maria DB):

But first we set up our data base in maria DB:

1.1.1 - Create the fraud_detection Database In Maria DB and Hive.

The fraud detection database serves as the primary storage for credit card transaction data.

1.1.2 - Create the transactions Table

The following SQL query creates the transactions table within the fraud_detection database. The table captures various details for each transaction, including the transaction date, merchant information, customer details, and whether the transaction is marked as fraudulent.

```sql
CREATE TABLE transactions (
    idd int,
    trans_date_trans_time DATETIME,
    cc_num BIGINT,
    merchant VARCHAR(100),
    category VARCHAR(100),
    amt DECIMAL(20, 15),
    first VARCHAR(100),
    last VARCHAR(100),
    gender CHAR(1),
    street VARCHAR(255),
    city VARCHAR(100),
    state VARCHAR(100),
    zip VARCHAR(20),
    lat DECIMAL(9, 6),
    longg DECIMAL(9, 6),
    city_pop INT,
    job VARCHAR(255),
    dob DATE,
    trans_num VARCHAR(255) NOT NULL,
    unix_time BIGINT,
    merch_lat DECIMAL(20, 13),
    merch_long DECIMAL(20, 13),
    is_fraud TINYINT(1),
    PRIMARY KEY (trans_num)
);
```

Samsung Innovation Campus

DATA SOURCES

1.1.3 - Load CSV Data into the transactions Table To populate the transactions table with data, a CSV file is loaded using the LOAD DATA command in MySQL. The CSV file is assumed to be in the local directory.

```
LOAD DATA LOCAL INFILE '/home/student/Data/sample_fraud_data.csv'
INTO TABLE transactions
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

Result on terminal

```
MariaDB [fraud_detection]> LOAD DATA LOCAL INFILE '/home/student/Fraud_detection_project/data/fraudTrain.csv'
    -> INTO TABLE transactions
    -> FIELDS TERMINATED BY ','
    -> ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS
    -> ;

Query OK, 1296675 rows affected, 65535 warnings (9 min 19.11 sec)
Records: 1296675  Deleted: 0  Skipped: 0  Warnings: 659884

MariaDB [fraud_detection]>
MariaDB [fraud_detection]> select idd, trans_date_trans_time, cc_num, first  from transactions limit 3;
+--------+---------------------+-----------------+---------+
| idd    | trans_date_trans_time | cc_num          | first   |
+--------+---------------------+-----------------+---------+
| 852047 | 2019-12-15 20:23:10   | 6593250708747804 | Melissa |
|   6766 | 2019-01-05 11:12:15   | 4302480582202074 | David   |
| 889067 | 2019-12-23 19:06:40   | 3583635130604947 | Crystal |
+--------+---------------------+-----------------+---------+
3 rows in set (0.00 sec)
```

Now our Relational DB Data is ready to be ingested

DATA SOURCES

2.Stream of Data coming from python application developed to simulate data streaming.

Python Application:

```python
import sys
import time
import signal
import json
import argparse
import logging
from kafka import KafkaProducer

# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')

def main(kafka_topic, lines_per_batch, filelist, sleep_time):
    # Create a Kafka producer
    producer = KafkaProducer(bootstrap_servers='localhost:9092')

    def signal_handler(signal, frame):
        logging.info('You pressed Ctrl+C!')
        producer.close()
        sys.exit(0)

    signal.signal(signal.SIGINT, signal_handler)
    logging.info('Press Ctrl+C to stop the streaming')

    while True:
        for filename in filelist:
            try:
                with open(filename, 'r') as infile:
                    while True:
                        lines = infile.readlines(lines_per_batch)
                        if not lines:
                            break
                        for line in lines:
                            try:
                                # Parse JSON data
                                data = json.loads(line.strip())
                                # Convert JSON object back to string for Kafka
                                json_str = json.dumps(data)
                                # Send the JSON string as a message to Kafka
                                producer.send(kafka_topic, json_str.encode('utf-8'))
                                producer.flush()  # Ensure the message is sent
                                logging.info("Sent JSON data to %s", kafka_topic)
                            except json.JSONDecodeError:
                                logging.error("Failed to decode JSON from line: %s", line)
                        time.sleep(sleep_time)  # Pause before sending the next batch
            except FileNotFoundError:
                logging.error("File not found: %s", filename)
            except IOError as e:
                logging.error("I/O error occurred while handling file %s: %s", filename, e)

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Stream JSON data to a Kafka topic.')
    parser.add_argument('kafka_topic', help='Kafka topic name')
    parser.add_argument('lines_per_batch', type=int, help='Number of lines to read at a time')
    parser.add_argument('files', nargs='+', help='List of files to stream data from')
    parser.add_argument('--sleep', type=float, default=5, help='Time interval between sending data (in
seconds)')

    args = parser.parse_args()

    main(args.kafka_topic, args.lines_per_batch, args.files, args.sleep)
```

3.Csv file in local machine.

Samsung Innovation Campus

DATA INGESTION

1. SQOOP Data Ingestion:

Getting transactions table from Fraud_detection MySQL Database into Hive.
The following command uses Apache Sqoop to import the transactions table from MySQL into the Hive data warehouse. The Hive table is created in the Fraud_detection Hive database.

```
sqoop import --connect jdbc:mysql://localhost/Fraud_detection \
 --username student --password student \
 --fields-terminated-by '\t' \
 --table transactions \
 --hive-import --hive-database 'Fraud_detection' \
 --hive-table 'transactions' \
 --split-by cc_num
```

Result on terminal:

```
2 ~]$ sqoop import --connect jdbc:mysql://localhost/Fraud_detection \
me student --password student \
-terminated-by '\t' \
transactions \
mport --hive-database 'Fraud_detection' \
able 'transactions' \
y cc_num
sr/local/sqoop/sqoop-1.4.7/../hcatalog does not exist! HCatalog jobs will fail.
$HCAT_HOME to the root of your HCatalog installation.
sr/local/sqoop/sqoop-1.4.7/../accumulo does not exist! Accumulo imports will fail.
$ACCUMULO_HOME to the root of your Accumulo installation.
sr/local/sqoop/sqoop-1.4.7/../zookeeper does not exist! Accumulo imports will fail.
$ZOOKEEPER_HOME to the root of your Zookeeper installation.
17:43:06,683 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
17:43:06,802 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -P instead.
17:43:06,924 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
17:43:06,934 INFO tool.CodeGenTool: Beginning code generation
17:43:07,372 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `transactions` AS t LIMIT 1
17:43:07,408 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM `transactions` AS t LIMIT 1
17:43:07,440 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /home/hadoop/hadoop
/sqoop-student/compile/a8e33e6a831faa9e648cc6c0409c942f/transactions.java uses or overrides a deprecated API.
pile with -Xlint:deprecation for details.
```

DATA INGESTION

2. Kafka With Spark streaming Real Time Data Ingestion:

2.1 Create Kakfka Topic called fraud_traindata to send produced data to.

```
kafka-topics --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic fraud_traindata
```

2.2 Now, we use our python application we created before as a kafka producer to produce streams of fraud_detection.json records to our fraud_traindata Topic.

```
python3 stream_json.py fraudtraindata 100 /home/student/Fraud_detection_project/data/fraud_train.json
```

Result on terminal:

```
9-20 21:48:52,534 - INFO - <BrokerConnection node_id=0 host=it-bro:9092 <connecting> [IPv4 ('192.168.1.4', 9092)]>: Connection complete.
9-20 21:48:52,535 - INFO - <BrokerConnection node_id=bootstrap-0 host=localhost:9092 <connected> [IPv6 ('::1', 9092, 0, 0)]>: Closing connection
9-20 21:48:53,030 - INFO - Sent JSON data to fraudtraindata
9-20 21:48:58,044 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:04,698 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:09,711 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:14,729 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:19,763 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:24,773 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:29,803 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:34,814 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:39,862 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:44,877 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:49,887 - INFO - Sent JSON data to fraudtraindata
9-20 21:49:54,969 - INFO - Sent JSON data to fraudtraindata
```

DATA INGESTION

2.3 **Finally we created pyspark / spark streaming Application** to pull the data from fraud_traindata topic as a consumer to process, structure and refine it to be inserted into our hive table

Spark Application:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.sql.types import *

spark = SparkSession.builder \
    .appName("KafkaSparkStreaming") \
    .master("local[*]") \
    .config("spark.streaming.stopGracefullyOnShutdown", "true") \
    .config("spark.jars.packeges", "org.apache.spark:spark-sql-kafka-0-10_2.13:3.1.2") \
    .config("spark.sql.warehouse.dir", "/user/hive/warehouse") \
    .config("hive.metastore.uris", "thrift://your-hive-metastore-uri:port") \
    .enableHiveSupport() \
    .getOrCreate()

# Kafka configs
kafka_input_config = {
    "kafka.bootstrap.servers": "localhost:9092",
    "subscribe": "fraudtraindata",
    "startingOffsets": "latest",
    "failOnDataLoss": "false"
}

schema = StructType([
    StructField("trans_date_trans_time", StringType(), True),
    StructField("cc_num", StringType(), True),
    StructField("merchant", StringType(), True),
    StructField("category", StringType(), True),
    StructField("amt", DoubleType(), True),
    StructField("first", StringType(), True),
    StructField("last", StringType(), True),
    StructField("gender", StringType(), True),
    StructField("street", StringType(), True),
    StructField("city", StringType(), True),
    StructField("state", StringType(), True),
    StructField("zip", StringType(), True),
    StructField("lat", DoubleType(), True),
    StructField("long", DoubleType(), True),
    StructField("city_pop", IntegerType(), True),
    StructField("job", StringType(), True),
    StructField("dob", StringType(), True),
    StructField("trans_num", StringType(), True),
    StructField("unix_time", LongType(), True),
    StructField("merch_lat", DoubleType(), True),
    StructField("merch_long", DoubleType(), True),
    StructField("is_fraud", IntegerType(), True)
])
```

```
def hiveInsert(df, epoch_id):
        df.write.mode("append").insertInto("fraud_detection.transactions")

df = spark \
    .readStream \
    .format("kafka") \
    .options(**kafka_input_config) \
    .load()


value_df = df.selectExpr("CAST(value AS STRING) as value")

json_df = value_df.withColumn("value", from_json(col("value"), schema))

flattened_df = json_df.select("value.*") \
    .withColumn("trans_date_trans_time", to_timestamp("trans_date_trans_time", "yyyy-MM-dd HH:mm:ss")) \
    .withColumn("cc_num", col("cc_num").cast("BIGINT")) \
    .withColumn("amt", col("amt").cast("DECIMAL(20,15)")) \
    .withColumn("lat", col("lat").cast("DECIMAL(9,6)")) \
    .withColumn("long", col("long").cast("DECIMAL(9,6)")) \
    .withColumn("merch_lat", col("merch_lat").cast("DECIMAL(20,13)")) \
    .withColumn("merch_long", col("merch_long").cast("DECIMAL(20,13)")) \
    .withColumn("dob", col("dob").cast("DATE"))


myStream = flattened_df.writeStream \
    .foreachBatch(hiveInsert) \
    .option("checkpointLocation", "/Data/hive_checkpoint") \
    .outputMode("append") \
    .start()
myStream.awaitTermination()
```

3. Spark Batch Data Ingestion:

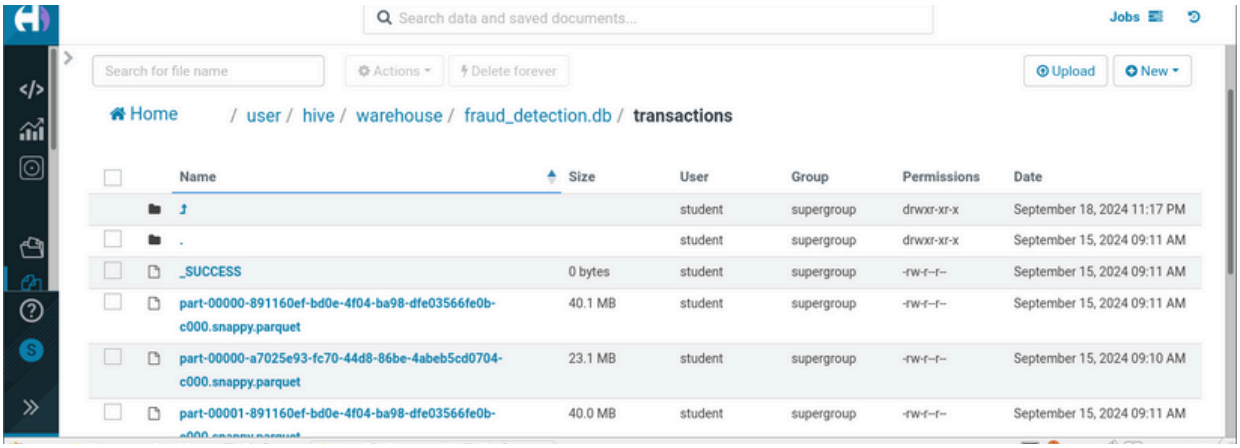
Here, we read the fraudTest.csv file into the fraud_df DataFrame using PySpark, then wrote the DataFrame into the fraud_detection.transactions Hive table.

```
# Reading fraudTest CSV file into a fraud_df DataFrame
fraud_df = spark.read.csv("file:///home/student/Fraud_detection_project/data/fraudTest.csv", header=True)

# Writing fraud_df to a Hive table
fraud_df.write.mode("append").saveAsTable("fraud_detection.transactions")
```

Samsung Innovation Campus

DATA STORAGE

we integrated data from three sources—relational database, streaming data, and CSV flat file—into a single Hive table stored on HDFS. This unified storage enables efficient querying and analysis, consolidating different data types into a structured format for seamless processing.




```
Select * from fraud_detection.transactions LIMIT 10;
```

Query History | Saved Queries | Results (10)

	transactions.cc_num	transactions.merchant	transactions.category	transactions.amt	transactions.first	transactions.las
1	2703186189652095	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks
2	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill
3	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez
4	3534093764340240	fraud_Kutch, Hermiston and Farrell	gas_transport	45.0	Jeremy	White
5	375534208663984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia
6	4767265376804500	fraud_Stroman, Hudson and Erdman	gas_transport	94.63	Jennifer	Conner

Samsung Innovation Campus

DATA QUERYING

Query #1

This query calculates the total amount of fraudulent transactions by summing up the values from the amt column where transactions are marked as fraudulent in the is_fraud column. It returns the total financial loss from all such transactions.

```sql
SELECT SUM(amt) AS total_fraud_loss
FROM transactions
WHERE is_fraud = 1;
```

This result after running query on the sample of data but the acutal loss of fraud trasctions around 4 million doller !

total_fraud_loss
1392307.8800000008

Samsung Innovation Campus

DATA QUERYING

Query #2

This query retrieves the most recent 10 fraudulent transactions from the transactions table. It filters for records where is_fraud is marked as 1, sorts them by the transaction date and time in descending order, and limits the result to the 10 most recent fraud cases

```sql
SELECT *
FROM transactions
WHERE is_fraud = 1
ORDER BY trans_date_trans_time DESC
LIMIT 10;
```

the last 10 fraud transactions happened and the amount loss of this transactions

ery History Saved Queries Results (10)

	transactions.idd	transactions.trans_date_trans_time	transactions.cc_num	transactions.merchant	transactions.category	transactions.amt	transactions.first	transactions.last	transaction
1	1295733	2020-06-21 03:59:46.0	4005676619255478	fraud_Koss and Sons	gas_transport	10.2	William	Perry	M
2	1295399	2020-06-21 01:00:08.0	3524574586339330	fraud_Kassulke PLC	shopping_net	977.01	Ashley	Cabrera	F
3	1295315	2020-06-21 00:07:09.0	3524574586339330	fraud_Kemmer-Buckridge	misc_pos	9.18	Ashley	Cabrera	F
4	1295314	2020-06-21 00:05:03.0	3560725013359375	fraud_Adams, Kovacek and Kuhlman	grocery_net	15.87	Brooke	Smith	F
5	1295257	2020-06-20 23:31:05.0	3573030041201292	fraud_Crist, Jakubowski and Littel	home	222.69	Joanne	Williams	F
6	1295255	2020-06-20 23:29:52.0	3560725013359375	fraud_Fisher-Schowalter	shopping_net	1063.03	Brooke	Smith	F
7	1295219	2020-06-20 23:17:07.0	6564459919350820	fraud_Strosin-Cruickshank	grocery_pos	307.71	Douglas	Willis	M
8	1295118	2020-06-20 22:33:33.0	3560725013359375	fraud_Dach-Nader	misc_net	834.6	Brooke	Smith	F
9	1294576	2020-06-20 18:12:40.0	6564459919350820	fraud_Kerluke-Abshire	shopping_net	965.05	Douglas	Willis	M
10	1293250	2020-06-20 03:48:39.0	3573030041201292	fraud_Rempel Inc	shopping_net	876.1	Joanne	Williams	F

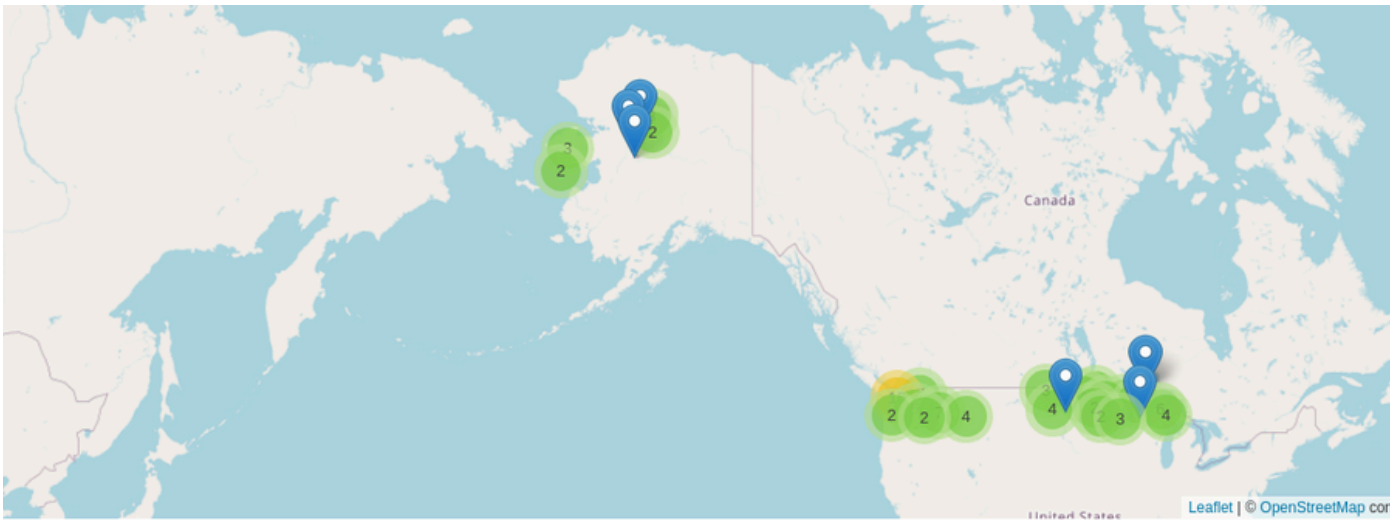
DATA QUERYING

Query #3

This query counts fraudulent transactions at each merchant location (latitude and longitude) and sorts them by the highest fraud occurrences. In a marker map, each location is marked, with larger or more prominent markers indicating more fraud at that location.

```sql
SELECT merch_lat, merch_long, COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY merch_lat, merch_long
ORDER BY fraud_count DESC;
```

the most merchants are located at unaided status and Canada and we used this exploration to check the distance of the customers and merchants because large distance can be alert for fraud transactions



Samsung Innovation Campus

DATA QUERYING

Query #4

This query identifies the top 10 jobs associated with the most fraudulent transactions, counting fraud cases for each job and displaying them in descending order of fraud occurrences

```sql
SELECT job, COUNT(is_fraud) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY job
ORDER BY fraud_count DESC
LIMIT 10;
```

the most jobs that customer in this job do fraud transactions are biomedical , mechanical engineer , etc..

story Saved Queries Results (10)

job	fraud_count
Scientist, biomedical	23
Mechanical engineer	22
Scientist, audiological	22
Film/video editor	20
Trading standards officer	20
Naval architect	19
Materials engineer	19
Quantity surveyor	19
Nurse, children's	18
Probation officer	18

Samsung Innovation Campus

DATA QUERYING

Query #5

This query counts the number of fraudulent transactions for each hour of the day by extracting the hour from the transaction timestamp (trans_date_trans_time). It groups the fraud cases by hour and displays them in descending order based on the number of frauds per hour.

```sql
SELECT HOUR(trans_date_trans_time) AS hour_of_day, COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY HOUR(trans_date_trans_time)
ORDER BY fraud_count DESC;
```

The result means most fraud transactions happens at evening and night hours

hour_of_day	fraud_count
22	679
23	673
1	233
0	227
2	222
3	217
17	32
14	31
12	31
18	27
13	27
16	26
15	25
21	25
19	25

DATA QUERYING

Query #6

This query counts the number of fraudulent transactions for each category in the transactions table. It groups the data by the category column, sorts the categories in descending order based on the number of fraud occurrences, and returns the fraud count for each category.

```sql
SELECT category, COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY category
ORDER BY fraud_count DESC;
```

Top 10 categories have fraud transactions

ery History | Saved Queries | Results (14)

	category	fraud_count
1	grocery_pos	648
2	shopping_net	571
3	misc_net	339
4	shopping_pos	282
5	gas_transport	184
6	misc_pos	93
7	personal_care	92
8	entertainment	88
9	kids_pets	86
10	home	76

DATA QUERYING

Query #7

This query counts the number of fraudulent transactions for each merchant. It groups the results by the merchant column, sorts them in descending order based on the number of fraud occurrences, and returns the fraud count for each merchant.

```sql
SELECT merchant, COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY merchant
ORDER BY fraud_count DESC;
```

Top 10 merchant has been affected by fraud transactions

ery History Saved Queries Results (576)

	merchant	fraud_count
1	fraud_Koepp-Witting	20
2	fraud_Lockman Ltd	19
3	fraud_Stracke-Lemke	19
4	fraud_Kozey-Boehm	19
5	fraud_Hudson-Ratke	18
6	fraud_Schumm, Bauch and Ondricka	18
7	fraud_Casper, Hand and Zulauf	18
8	fraud_Langworth, Boehm and Gulgowski	18
9	fraud_Murray-Smitham	17
10	fraud_Doyle Ltd	17
11	fraud_Towne, Greenholt and Koepp	17

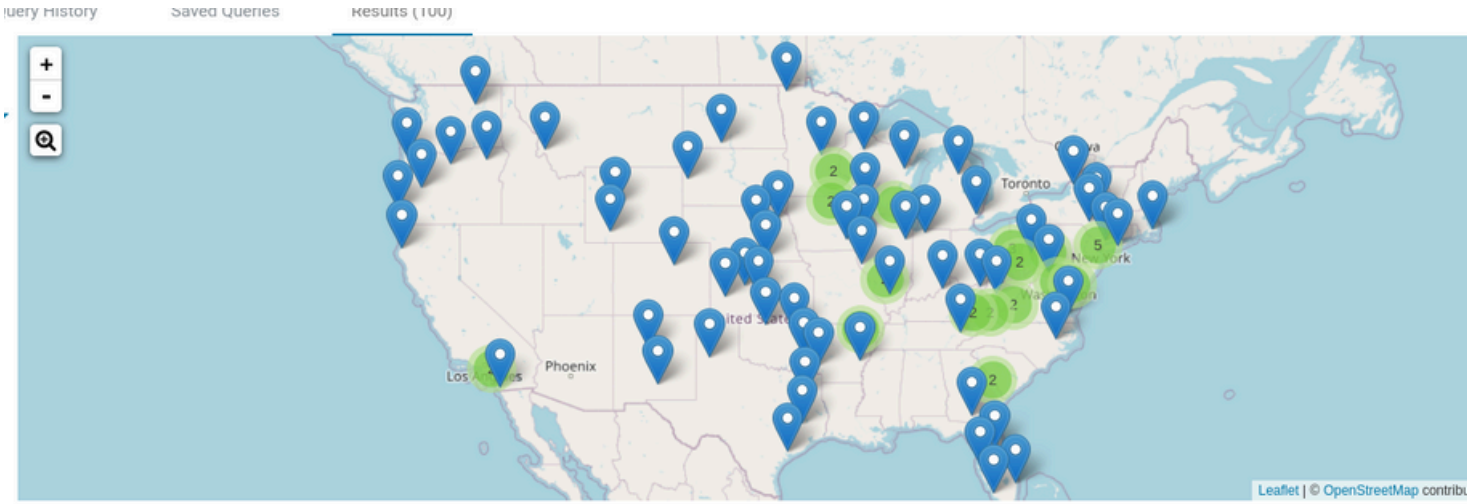
DATA QUERYING

Query #8

This query finds the top 100 locations with the most fraudulent transactions, grouping by latitude, longitude, city, and state. The results are ordered by the number of fraud cases. In a marker map, each location is shown with markers sized or colored according to the fraud count.

```sql
SELECT lat, longg, city, state, COUNT(is_fraud) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY lat, longg, city, state
ORDER BY fraud_count DESC
LIMIT 100;
```

The distribution of the customer in the united states



DATA QUERYING

Query #9

This query counts the number of fraudulent transactions for each city, grouped by both city name and its population (city_pop). It orders the results in descending order based on the number of fraud cases per city and population.

```sql
SELECT city_pop, COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY city , city_pop
ORDER BY fraud_count DESC;
```

Top cities that have fraud detection and the population of each city and we discovered that high population cities have high number of fraud transactions

ery History | Saved Queries | Results (701)

	city	city_pop	fraud_count
1	Houston	2906700	14
2	Warren	134056	11
3	Beaver Falls	28425	11
4	Naples	276002	11
5	Tulsa	413574	11
6	Huntsville	190178	10
7	San Diego	1241364	10
8	Washington	601723	10
9	Detroit	673342	10
10	Hahira	10295	9
11	Hudson	215	9

Samsung Innovation Campus

DATA QUERYING

Query #10

This query identifies the top 10 credit card numbers associated with the most fraudulent transactions. It groups the results by credit card number (cc_num), cardholder's first name (first), and last name (last), counts the number of fraud cases for each, and sorts them in descending order of fraud count.

```sql
SELECT cc_num, first, last, COUNT(is_fraud) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY cc_num, first, last
ORDER BY fraud_count DESC
LIMIT 10;
```

Top 10 customers and each customer how many fraud transaction did

cc_num	first	last	fraud_count
4400011257587661852	Marissa	Powell	9
5289285402893489	Amanda	Adams	9
3517527805128735	Tracy	Conway	9
4629451965224809	Karen	Warren	9
4259996134423	Julie	Johnson	9
6011366578560244	Adam	Stark	8
3519607465576254	Audrey	Gonzalez	8
30033162392091	Kevin	Summers	8
4266200684857219	Joshua	Bryant	8
2227671554547514	Angie	Jones	8

Samsung Innovation Campus

DATA QUERYING

Query #11
This query calculates the average transaction amount (avg_amount) for both fraudulent and non-fraudulent transactions. It groups the results by the is_fraud flag, which indicates whether a transaction is fraudulent (1) or not (0). The result shows the average amount for each fraud category.

```
SELECT is_fraud, AVG(amt) AS avg_amount
FROM transactions
GROUP BY is_fraud;
```

The average amount of fraud transaction higher than the average amount of non-fraud transaction

History | Saved Queries | Results (2)

is_fraud	avg_amount
false	71.67754050073643
true	526.990113550341

DATA QUERYING

Query #12

This query categorizes fraudulent transactions into age groups based on the difference between the current year and the year of birth (dob). It counts the number of fraud cases within each age group and orders the results by the number of fraud cases in descending order. The age groups are defined as follows:
'16-25': Ages less than 25 years.
'25-40': Ages between 25 and 40 years.
'41-60': Ages between 41 and 60 years.
'60-100': Ages over 60 years.

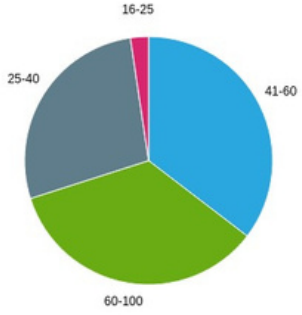
```sql
SELECT
    CASE
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) < 25 THEN '16-25'
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) BETWEEN 25 AND 40 THEN '25-40'
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) BETWEEN 41 AND 60 THEN '41-60'
        ELSE '60-100'
    END AS age_group,
    COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY CASE
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) < 25 THEN '16-25'
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) BETWEEN 25 AND 40 THEN '25-40'
        WHEN YEAR(CURRENT_DATE) - YEAR(dob) BETWEEN 41 AND 60 THEN '41-60'
        ELSE '60-100'
    END
ORDER BY fraud_count DESC;
```

People in range 41-60 who did the most number of fraud transactions

Results (4)

age_group	fraud_count
41-60	935
60-100	916
25-40	729
16-25	62

Results (4)



Samsung Innovation Campus

DATA QUERYING

Query #13

This query finds and lists fraudulent transactions with amounts above the average transaction amount, sorting the results by the amount in descending order.

```sql
WITH avg_amount AS (
    SELECT AVG(amt) AS average_amt
    FROM transactions
),

transactions_above_avg AS (
    SELECT t.trans_date_trans_time, t.cc_num, t.merchant, t.amt, t.city, t.state, t.is_fraud
    FROM transactions t
    JOIN avg_amount a
    ON t.amt > a.average_amt
    WHERE t.is_fraud = 1
)

SELECT trans_date_trans_time, cc_num, merchant, amt, city, state, is_fraud
FROM transactions_above_avg
ORDER BY amt DESC;
```

ery History Saved Queries Results (1,316)

	trans_date_trans_time	cc_num	merchant	amt	city	state	is_fraud
1	2019-01-18 23:20:16.0	4586810168620942	fraud_Pouros-Conroy	1334.07	Edisto Island	SC	true
2	2019-08-17 01:19:31.0	4044436772018844508	fraud_Heathcote, Yost and Kertzmann	1289.89	Dayton	OH	true
3	2019-12-21 23:39:07.0	180040131978916	fraud_Schmidt and Sons	1289.07	North East	PA	true
4	2019-06-29 22:02:22.0	3519232971341141	fraud_Ruecker, Beer and Collier	1268.18	Amsterdam	OH	true
5	2019-02-15 23:47:40.0	503848303379	fraud_Reichert, Rowe and Mraz	1258.71	Saint James City	FL	true
6	2019-01-03 23:41:36.0	4613314721966	fraud_Mohr-Bayer	1254.27	Collettsville	NC	true
7	2020-05-06 23:11:57.0	4536996888716062123	fraud_Fisher Inc	1253.93	Espanola	NM	true
8	2019-02-28 22:35:06.0	4642894980163	fraud_Jast Ltd	1238.3	Clarinda	IA	true
9	2019-12-17 12:17:00.0	30273037698427	fraud_Bins, Balistreri and Beatty	1234.22	Thida	AR	true
10	2019-07-22 22:42:13.0	676148621961	fraud_Heathcote LLC	1230.42	Eagarville	IL	true
11	2019-04-28 22:27:23.0	4587657402165341815	fraud_Bahringer, Schoen and Corkery	1224.1	Pembroke Township	IL	true
12	2019-05-27 23:22:09.0	2242542703101233	fraud_Hudson-Grady	1221.91	Westport	KY	true
13	2019-09-23 23:32:08.0	4457732997086323466	fraud_Rempel Inc	1214.39	Fiddletown	CA	true
14	2019-07-10 22:53:15.0	377234009633447	fraud_Reichert, Huels and Hoppe	1205.19	Shenandoah Junction	WV	true
15	2019-03-16 22:38:47.0	4355790796238264643	fraud_Kozey-Boehm	1204.44	Payson	IL	true
16	2019-12-31 23:59:23.0	213155997615567	fraud_Boyer-Reichert	1203.86	Clune	PA	true
17	2020-02-02 22:54:19.0	6534628260579800	fraud_Kerluke-Abshire	1202.36	Hinesburg	VT	true
18	2020-03-05 07:10:37.0	4198470814557	fraud_Kuhic, Bins and Pfeffer	1201.24	Avoca	IA	true
19	2019-05-01 23:08:11.0	2305336922781618	fraud_Fisher-Schowalter	1193.21	Moulton	IA	true

Samsung Innovation Campus

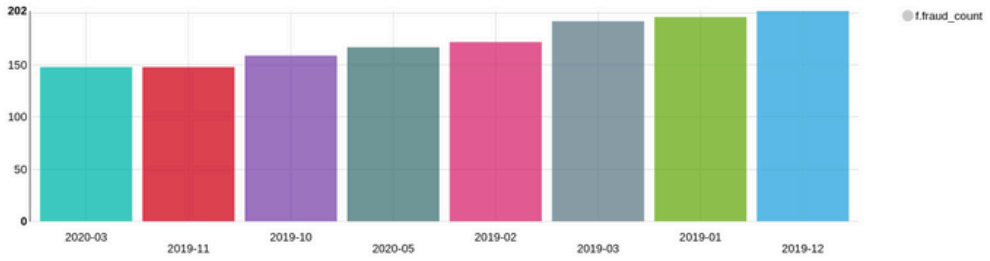
DATA QUERYING

Query #14

This query finds and lists months with fraudulent transaction counts above the monthly average, sorting them by the number of frauds in descending order.

```sql
WITH fraud_transactions_by_month AS (
    SELECT
        from_unixtime(unix_timestamp(trans_date_trans_time, 'yyyy-MM-dd HH:mm:ss'), 'yyyy-MM') AS
month_year,
        COUNT(*) AS fraud_count
    FROM transactions
    WHERE is_fraud = 1
    GROUP BY from_unixtime(unix_timestamp(trans_date_trans_time, 'yyyy-MM-dd HH:mm:ss'), 'yyyy-MM')
),

avg_fraud_per_month AS (
    SELECT AVG(fraud_count) AS avg_fraud_count
    FROM fraud_transactions_by_month
)

SELECT f.month_year, f.fraud_count
FROM fraud_transactions_by_month f
JOIN avg_fraud_per_month a
ON f.fraud_count > a.avg_fraud_count
ORDER BY f.fraud_count DESC;
```

The Most number of fraud transactions happened at December 2019



	f.month_year	f.fraud_count
1	2019-12	202
2	2019-01	196
3	2019-03	192
4	2019-02	172
5	2020-05	167
6	2019-10	159
7	2020-03	148
8	2019-11	148

Samsung Innovation Campus

DATA QUERYING

Query #15

This query categorizes fraudulent transactions into different amount ranges and counts the number of frauds within each range. The ranges are:
'Under $50': Transactions less than $50.
'$50-$200': Transactions between $50 and $200.
'$200-$500': Transactions between $200 and $500.
'Over $500': Transactions over $500.
The results are grouped by these amount ranges and ordered by the count of fraud cases in descending order.

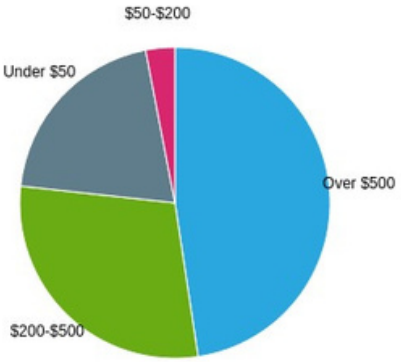
```sql
SELECT
    CASE
        WHEN amt < 50 THEN 'Under $50'
        WHEN amt BETWEEN 50 AND 200 THEN '$50-$200'
        WHEN amt BETWEEN 200 AND 500 THEN
'$200-$500' ELSE 'Over $500'
    END AS amount_range,
    COUNT(*) AS fraud_count
FROM transactions
WHERE is_fraud = 1
GROUP BY CASE
        WHEN amt < 50 THEN 'Under $50'
        WHEN amt BETWEEN 50 AND 200 THEN '$50-$200'
        WHEN amt BETWEEN 200 AND 500 THEN
'$200-$500' ELSE 'Over $500'
    END
ORDER BY fraud_count DESC;
```

The most Fraud Transactions its amount is greater than 500 $

ueries Results (4)

	amount_range	fraud_count
1	Over $500	1258
2	$200-$500	769
3	Under $50	535
4	$50-$200	80

Results (4)



DATA QUERYING

Query #16

This query lists credit card holders with multiple fraudulent transactions, showing the count and total amount of fraud for each. Results are sorted by the number of frauds and total fraud amount, both in descending order.

```sql
SELECT cc_num, first, last, COUNT(trans_num) AS fraud_count, SUM(amt) AS total_fraud_amount
FROM transactions
WHERE is_fraud = 1
GROUP BY cc_num, first, last
HAVING fraud_count > 1
ORDER BY fraud_count DESC, total_fraud_amount DESC;
```

	cc_num	first	last	fraud_count	total_fraud_amount
1	3517527805128735	Tracy	Conway	9	5348.51
2	4629451965224809	Karen	Warren	9	5133.93
3	4259996134423	Julie	Johnson	9	5000.290000000001
4	5289285402893489	Amanda	Adams	9	4573.919999999999
5	4400011257587661852	Marissa	Powell	9	3666.7699999999995
6	4826655832045236	Tami	Forbes	8	6144.11
7	4266200684857219	Joshua	Bryant	8	5683.92
8	3519607465576254	Audrey	Gonzalez	8	4991.51
9	377993105397617	Nathan	Martinez	8	4662.38
10	4806443445305	Eric	Patel	8	4586.67
11	6011366578560244	Adam	Stark	8	4544.42
12	213157767990030	Tara	Campbell	8	3265.2500000000005
13	30033162392091	Kevin	Summers	8	3168.14
14	180048185037117	Mary	Wall	8	3138.9900000000002
15	2227671554547514	Angie	Jones	8	2241.6200000000003
16	4538566639857	Jerry	Kelly	7	5695.410000000001
17	4294930380592	Misty	Rivera	7	5022.75
18	2512312531485080	John	Mccormick	7	4668.76
19	6011382886333463	Martin	Duarte	7	4582.1900000000005

DATA QUERYING

Query # 17

This query analyzes relationships between customers and merchants to identify potential fraud rings.

```sql
WITH FraudConnections AS (
    SELECT
        t.cc_num,
        t.merchant,
        COUNT(*) AS transaction_count,
        SUM(CASE WHEN t.is_fraud = 1 THEN 1 ELSE 0 END) AS fraud_count
    FROM transactions t
    WHERE t.is_fraud = 1
    GROUP BY t.cc_num, t.merchant
)
SELECT
    cc_num,
    COUNT(DISTINCT merchant) AS unique_merchants,
    SUM(fraud_count) AS total_frauds,
    COUNT(*) AS total_transactions,
    (SUM(fraud_count) / COUNT(*)) * 100 AS fraud_rate
FROM FraudConnections
GROUP BY cc_num
HAVING unique_merchants > 1
```

The same customer did a lot of fraud transactions with different merchants

cc_num	unique_merchants	total_frauds	total_transactions	fraud_rate
4400011257587661852	9	9	9	100
4806443445305	7	8	7	114.28571428571428
4538566639857	6	7	6	116.66666666666667
6506116513503136	7	7	7	100
340951438290556	5	6	5	120
3514865930894695	5	6	5	120
3542826960473004	5	6	5	120
3543885983111461	5	6	5	120
4450831335606294	5	6	5	120
6528911529051375	6	6	6	100
6538891242532018	6	6	6	100
6554245334757802	6	6	6	100
4210078554961359092	6	6	6	100
4292902571056973207	6	6	6	100
4301028321766222513	6	6	6	100
4355790796238264643	6	6	6	100
2256234701263057	4	5	4	125
4128027264554082	4	5	4	125
4134456652433447	4	5	4	125

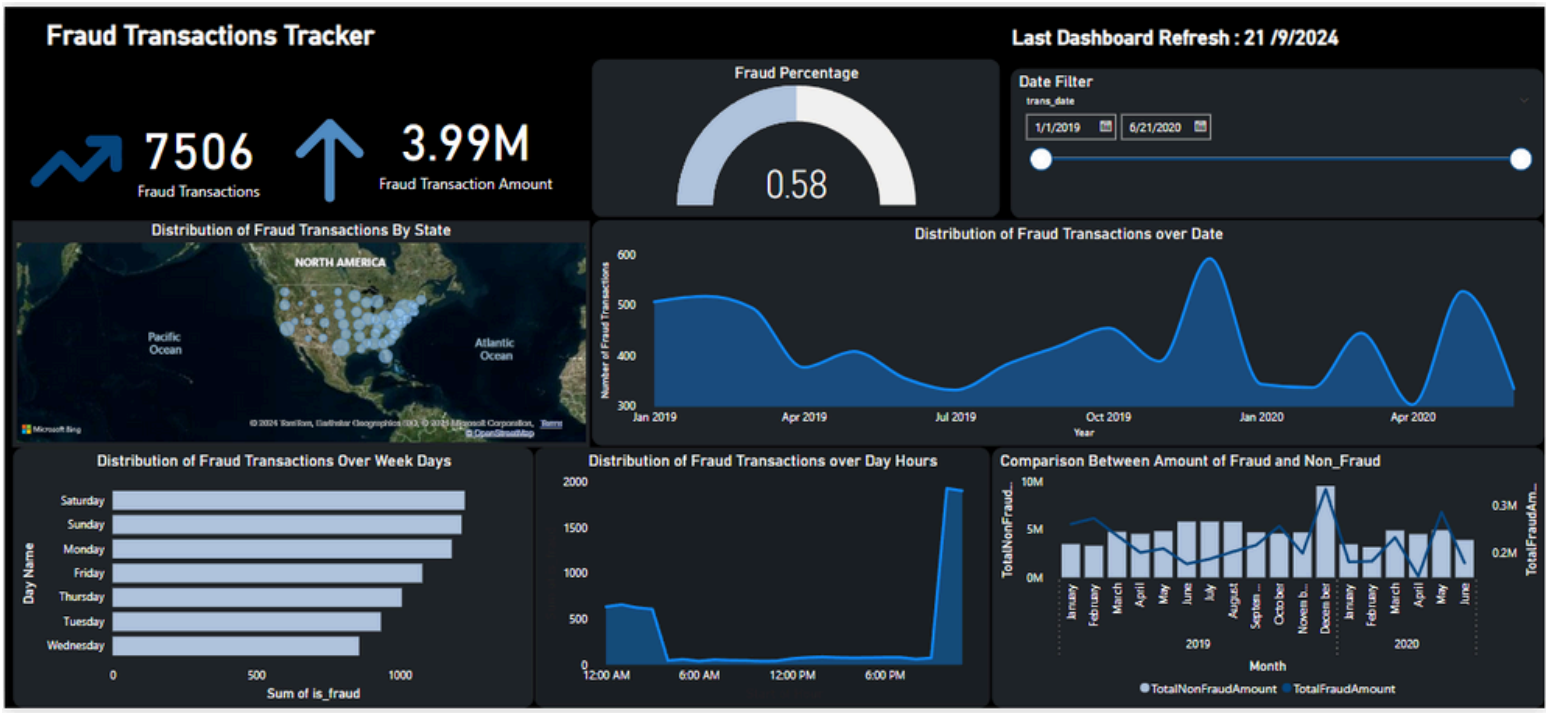
DATA VISUALIZATION

The dashboard titled "Fraud Transactions Tracker" provides an in-depth view of fraudulent transactions over a specific time period.

1. **Fraud Transactions (Top Left)**
   - 7506 indicates the total number of fraud transactions recorded in the dataset.
   - 3.99M represents the total dollar amount involved in these fraudulent transactions.
2. **Fraud Percentage (Center-Top)**
   - A gauge chart shows that 0.58 (58%) of the total transactions within the selected timeframe are fraudulent.
3. **Date Filter (Top Right)**
   - A date range slider allows users to filter the transactions between January 1, 2019, and June 21, 2020. The selected date range is displayed directly above the slider.
4. **Map of Fraud Transactions by State (Left-Center)**
   - A map visualization shows the geographic distribution of fraud transactions across the United States. Larger dots indicate states with higher volumes of fraud. This view helps identify regions where fraud is more prevalent.
5. **Distribution of Fraud Transactions Over Date (Center-Right)**
   - A line chart showing the number of fraud transactions over time, starting from January 2019 to April 2020. Peaks in the graph represent periods when fraud activity was particularly high.
6. **Distribution of Fraud Transactions by Weekday (Bottom-Left)**
   - A bar chart displaying the total fraud transactions for each day of the week. Fraud activity appears to be highest on Saturday and Sunday and lower on weekdays, especially Tuesday and Thursday.
7. **Distribution of Fraud Transactions by Hour (Bottom-Center)**
   - A line chart showing fraud activity by hour of the day. It highlights that most fraud transactions occur late in the day, peaking significantly around 6 PM.
8. **Comparison Between Fraud and Non-Fraud Transactions (Bottom-Right)**
   - A dual-axis bar and line chart comparing the amount of fraud vs. non-fraud transactions month-over-month from January 2019 to June 2020. The bars represent non-fraudulent transaction amounts, and the line represents fraudulent transaction amounts. This chart illustrates how fraud compares to normal transactions on a monthly basis.
9. **Last Dashboard Refresh (Top-Right)**
   - The dashboard was last refreshed on September 21, 2024.

Samsung Innovation Campus

DATA VISUALIZATION



Samsung Innovation Campus

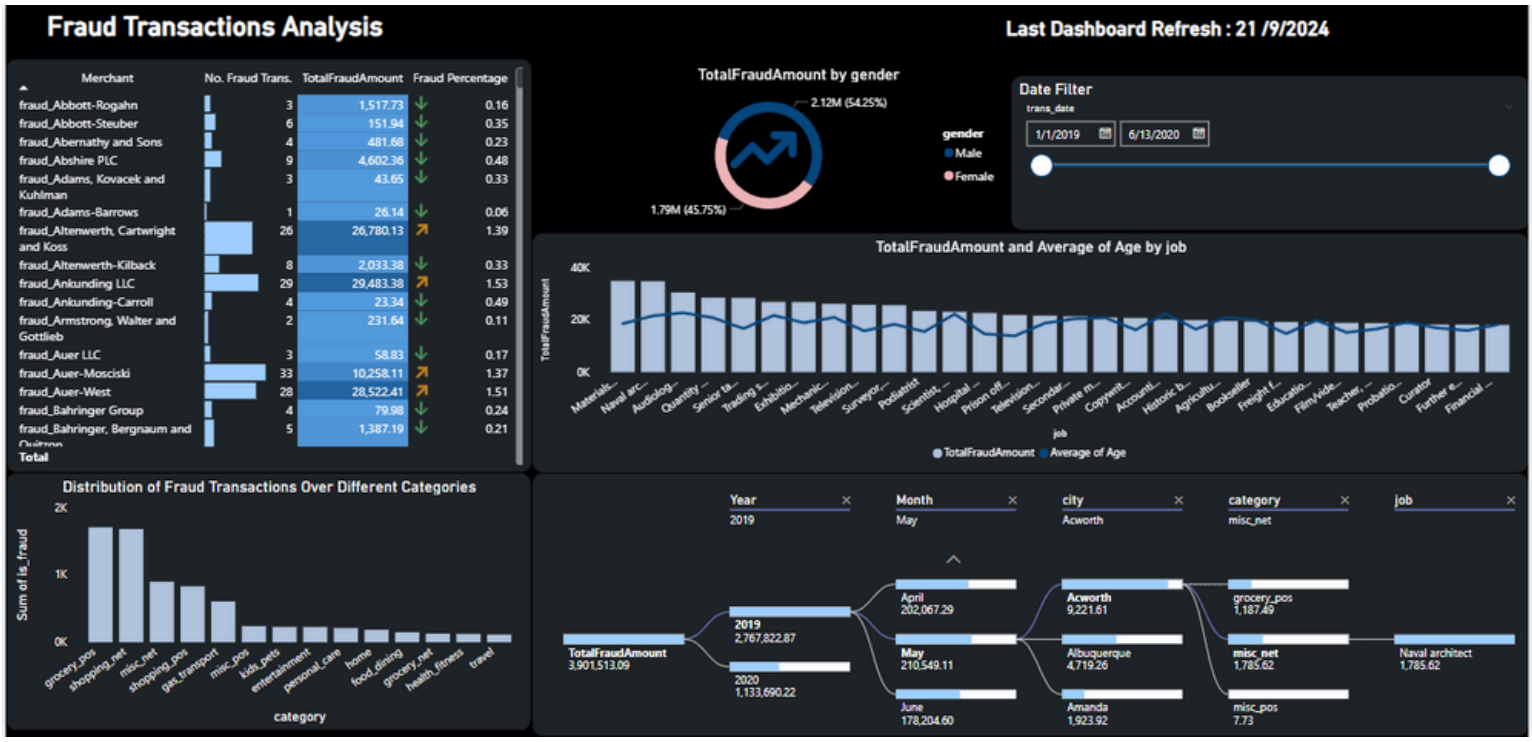
DATA VISUALIZATION

This dashboard, titled "Fraud Transactions Analysis," offers a more detailed breakdown of fraud transactions by merchant, gender, category, and job roles, as well as a visual representation of fraud trends over time.

1. **Merchant Table (Left-Top)**
   - This table lists various merchants involved in fraudulent transactions, showing:
     - No. Fraud Trans.: The number of fraud transactions associated with each merchant.
     - Total Fraud Amount: The total dollar amount of fraud for each merchant.
     - Fraud Percentage: The proportion of total transactions for each merchant that are fraudulent, with arrows indicating whether this percentage has increased or decreased.
   - This section helps identify merchants with higher fraud occurrences and track changes in fraud rates.
2. **Total Fraud Amount by Gender (Center-Top)**
   - A donut chart represents the distribution of total fraud by gender:
     - Male: $2.12M (54.25%)
     - Female: $1.79M (45.75%)
   - This gives insight into the gender split in fraudulent transactions, with males showing a slightly higher fraud amount.
3. **Date Filter (Top Right)**
   - A slider that allows filtering of transactions within the date range of January 1, 2019, to June 13, 2020.
4. **Total Fraud Amount and Average Age by Job (Center)**
   - A bar and line chart showing the Total Fraud Amount (bars) and the Average Age (line) across various job categories.
   - Jobs such as Marketers and Newspaper Carriers exhibit higher total fraud amounts, while other professions have lower fraud rates.
   - The average age across different job roles is also visualized, helping identify whether fraud is more common among certain age groups in specific professions.
5. **Distribution of Fraud Transactions Over Different Categories (Bottom Left)**
   - A bar chart that displays the distribution of fraud across various categories such as grocery, shopping, misc_pos, etc.
   - Categories like grocery_pos and shopping_net have the highest number of fraud transactions, while health_fitness and travel show the lowest.
6. **Fraud Transaction Trends (Bottom-Center)**
   - A flow diagram that tracks fraud trends over different periods, cities, categories, and jobs. It starts from the year and month, drilling down into city, category, and job to provide a detailed view of the fraud amount.
   - For instance:
     - In May 2019, fraud worth $202,067.29 occurred in Acworth under the grocery_pos category, with the Naval Architect job being involved.
   - This allows a granular exploration of fraud patterns across different dimensions.
7. **Last Dashboard Refresh (Top Right)**
   - The dashboard was last refreshed on September 21, 2024.

Samsung Innovation Campus

DATA VISUALIZATION



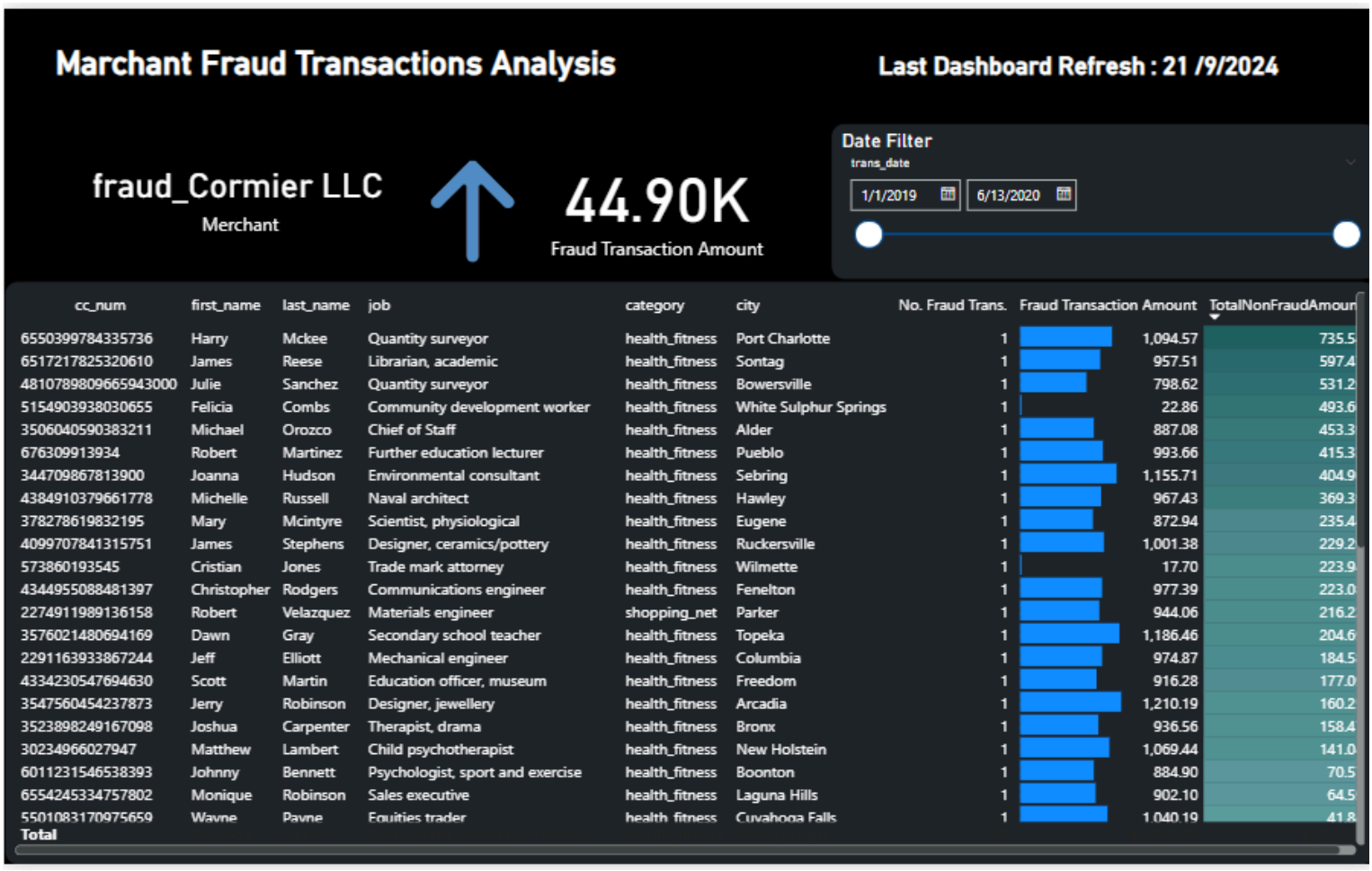
DATA VISUALIZATION

The second dashboard you've shared, titled "Marchant Fraud Transactions Analysis," shows a **drilled-down view** after selecting fraud_Cormier LLC from the previous dashboard. Here's a breakdown of the key insights:

1. **Merchant Details (Top-Left)**
   - The selected merchant is fraud_Cormier LLC.
   - The total fraud transaction amount for this merchant is $44.90K, shown prominently with an upward arrow, highlighting the fraudulent activity associated with this merchant.
2. **Credit Card Holder Information Table (Center)**
   - This table lists the details of customers involved in fraud transactions with this merchant:
     - cc_num: Credit card number.
     - first_name and last_name: Customer's name.
     - job: The occupation of the customer.
     - category: The type of transaction involved (e.g., health_fitness, misc_net).
     - city: The city where the customer resides.
     - No. Fraud Trans.: Number of fraud transactions per customer (all are 1 in this case).
     - Fraud Transaction Amount: The total fraudulent transaction amount for each customer.
     - Total Non-Fraud Amount: The total legitimate (non-fraudulent) transaction amount associated with each customer. This provides context to the fraud amount, showing the overall financial activity of the customer.
3. **Bar Chart (Right)**
   - The Fraud Transaction Amount per customer is visualized through a horizontal bar chart, with the amounts sorted in descending order.
   - The largest fraud transaction is $1,094.57 in Port Charlotte for Harry McKee (Quantity Surveyor), followed by $957.51 in Sontag for James Reese (Librarian, Academic).
4. **Total Non-Fraud Amount (Right)**
   - The Total Non-Fraud Amount column provides a view of how much legitimate spending occurred alongside the fraud. The shading gives a quick comparison, helping identify customers whose fraud may be an anomaly compared to their overall spending.
5. **Date Filter (Top Right)**
   - Similar to the previous dashboard, there's a date filter allowing you to restrict the view to transactions between January 1, 2019, and June 13, 2020.
6. **Last Dashboard Refresh (Top Right)**
   - The dashboard was last refreshed on September 21, 2024, ensuring the data is current.

Samsung Innovation Campus

DATA VISUALIZATION

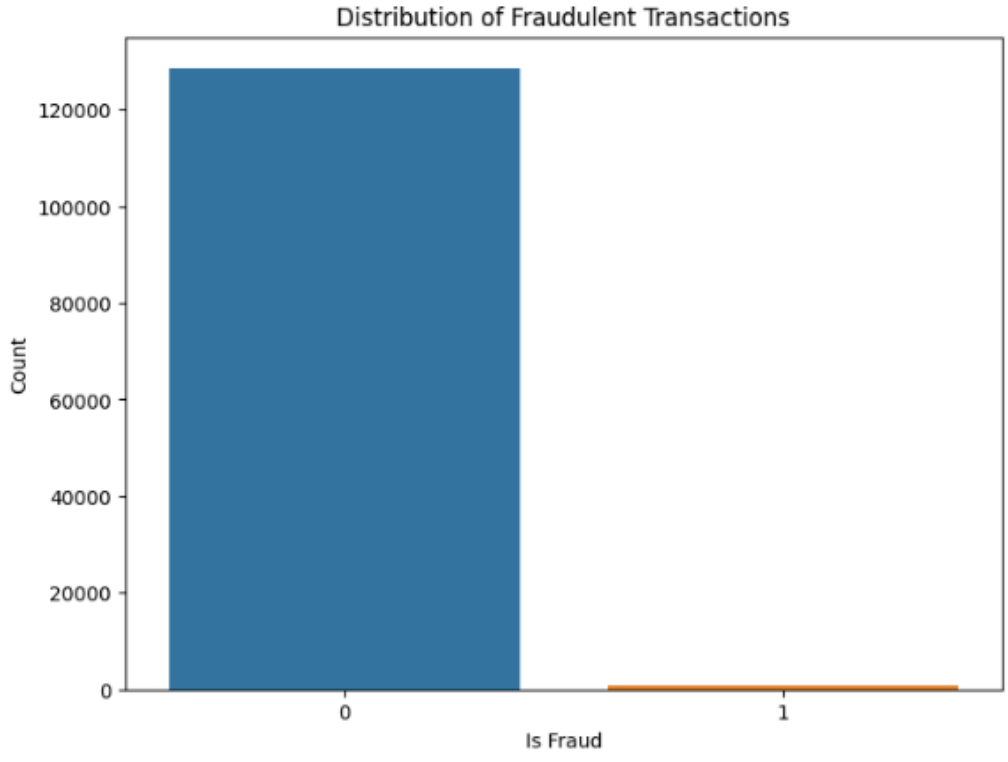


cc_num	first_name	last_name	job	category	city	No. Fraud Trans.	Fraud Transaction Amount	TotalNonFraudAmoun
6550399784335736	Harry	Mckee	Quantity surveyor	health_fitness	Port Charlotte	1	1,094.57	735.5
6517217825320610	James	Reese	Librarian, academic	health_fitness	Sontag	1	957.51	597.4
4810789809665943000	Julie	Sanchez	Quantity surveyor	health_fitness	Bowersville	1	798.62	531.2
5154903938030655	Felicia	Combs	Community development worker	health_fitness	White Sulphur Springs	1	22.86	493.6
3506040590383211	Michael	Orozco	Chief of Staff	health_fitness	Alder	1	887.08	453.3
676309913934	Robert	Martinez	Further education lecturer	health_fitness	Pueblo	1	993.66	415.3
344709867813900	Joanna	Hudson	Environmental consultant	health_fitness	Sebring	1	1,155.71	404.9
4384910379661778	Michelle	Russell	Naval architect	health_fitness	Hawley	1	967.43	369.3
378278619832195	Mary	Mcintyre	Scientist, physiological	health_fitness	Eugene	1	872.94	235.4
4099707841315751	James	Stephens	Designer, ceramics/pottery	health_fitness	Ruckersville	1	1,001.38	229.2
573860193545	Cristian	Jones	Trade mark attorney	health_fitness	Wilmette	1	17.70	223.9
4344955088481397	Christopher	Rodgers	Communications engineer	health_fitness	Fenelton	1	977.39	223.0
2274911989136158	Robert	Velazquez	Materials engineer	shopping_net	Parker	1	944.06	216.2
3576021480694169	Dawn	Gray	Secondary school teacher	health_fitness	Topeka	1	1,186.46	204.6
2291163933867244	Jeff	Elliott	Mechanical engineer	health_fitness	Columbia	1	974.87	184.5
4334230547694630	Scott	Martin	Education officer, museum	health_fitness	Freedom	1	916.28	177.0
3547560454237873	Jerry	Robinson	Designer, jewellery	health_fitness	Arcadia	1	1,210.19	160.2
3523898249167098	Joshua	Carpenter	Therapist, drama	health_fitness	Bronx	1	936.56	158.4
30234966027947	Matthew	Lambert	Child psychotherapist	health_fitness	New Holstein	1	1,069.44	141.0
6011231546538393	Johnny	Bennett	Psychologist, sport and exercise	health_fitness	Boonton	1	884.90	70.5
6554245334757802	Monique	Robinson	Sales executive	health_fitness	Laguna Hills	1	902.10	64.5
5501083170975659	Wayne	Payne	Equities trader	health_fitness	Cuyahoga Falls	1	1,040.19	41.8
Total								

EDA

plot that visualizes the distribution of the target variable (is_fraud), showing the number of fraudulent versus non-fraudulent transactions, notice the majority of legitimate transactions.

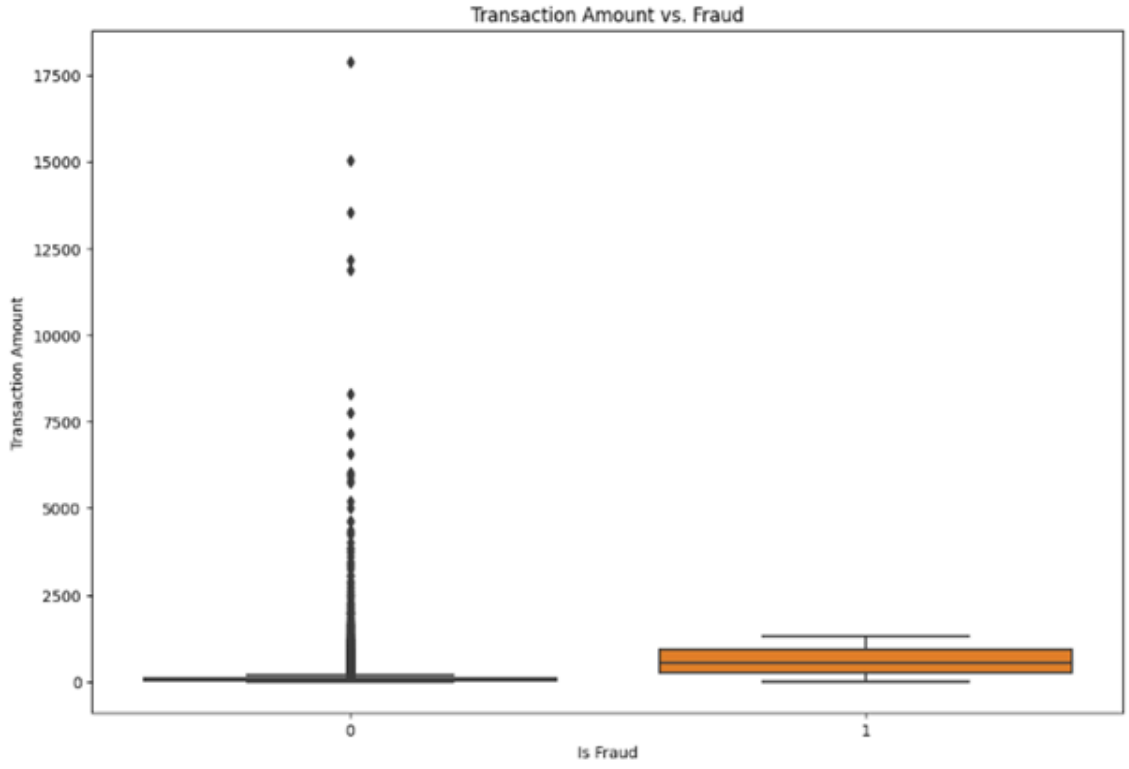
```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Visualize the distribution of the target variable (fraudulent or not)
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.countplot(x='is_fraud', data=train_data_sample)
plt.title('Distribution of Fraudulent Transactions')
plt.xlabel('Is Fraud')
plt.ylabel('Count')
plt.show()
```



EDA

The section uses pandas DataFrame to visualize 10% of training data, creating a boxplot using matplotlib and seaborn to examine the relationship between transaction amounts and fraud status.

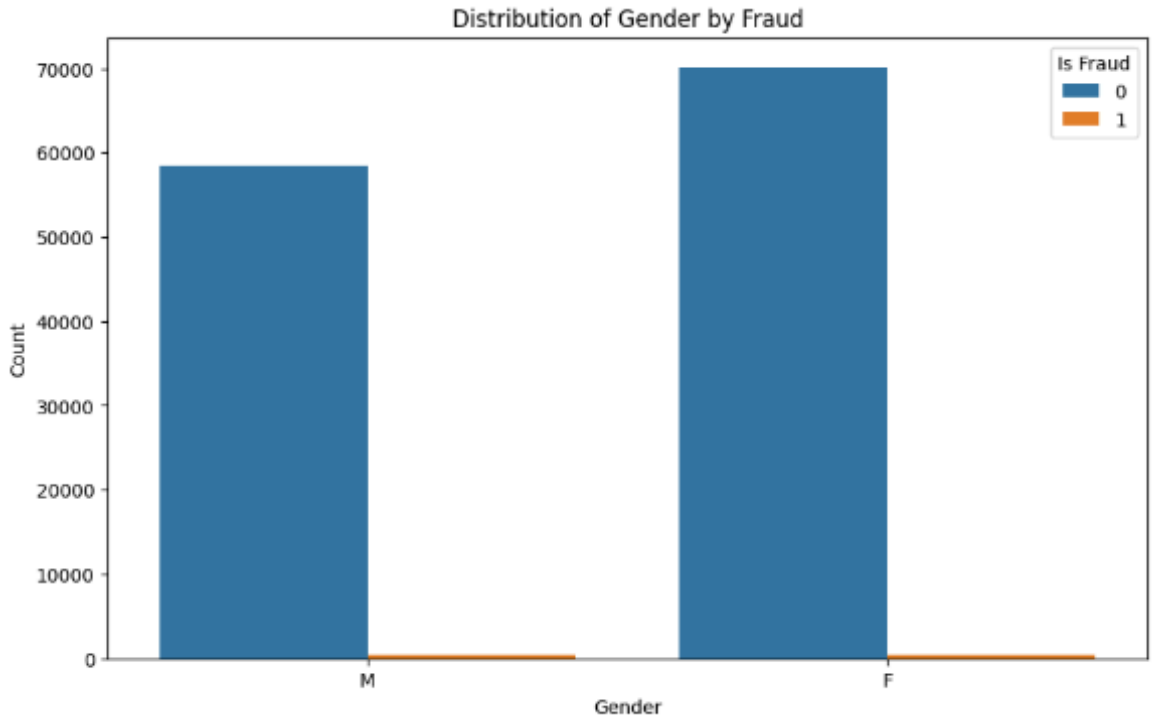
```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Explore the distribution of transaction amounts by fraud status
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 8))
sns.boxplot(x='is_fraud', y='amt', data=train_data_sample)
plt.title('Transaction Amount vs. Fraud')
plt.xlabel('Is Fraud')
plt.ylabel('Transaction Amount')
plt.show()
```



EDA

The section uses matplotlib and seaborn to create a pandas DataFrame, showcasing the distribution of gender features by fraud status, to visualize gender's impact on fraudulent and non-fraudulent transactions.

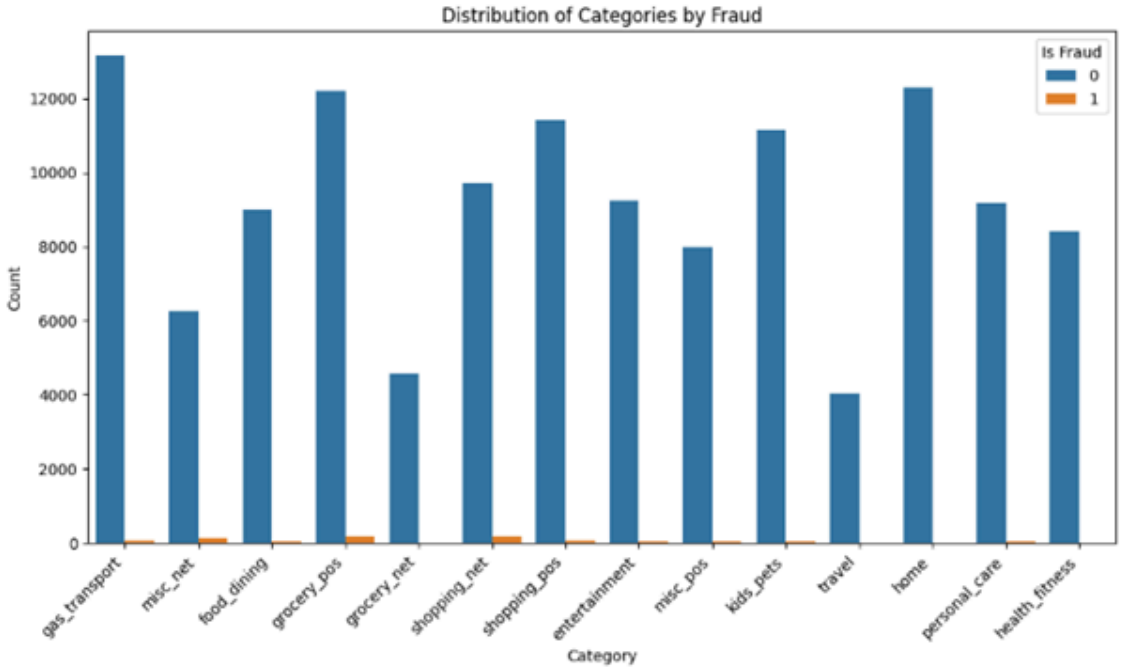
```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Explore categorical features (e.g., gender)
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(x='gender', hue='is_fraud', data=train_data_sample)
plt.title('Distribution of Gender by Fraud')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.legend(title='Is Fraud')
plt.show()
```



EDA

This section uses Matplotlib and Seaborn to create a count plot from 10% of training data, illustrating the distribution of categories based on fraud status.

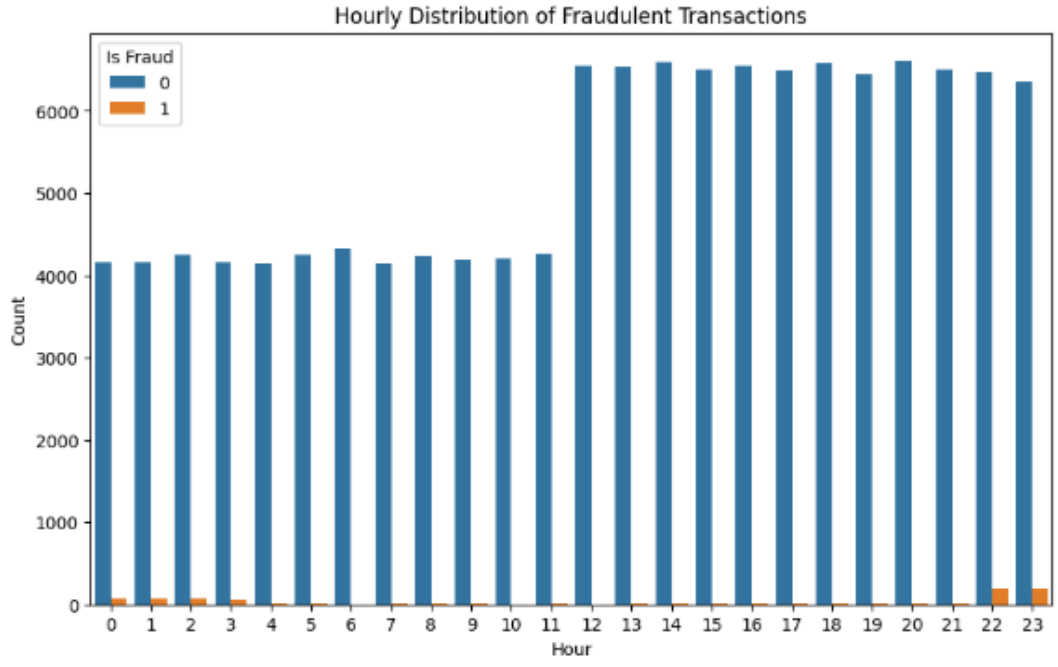
```python
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Explore the distribution of categories by fraud status
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.countplot(x='category', hue='is_fraud', data=train_data_sample)
plt.title('Distribution of Categories by Fraud')
plt.xlabel('Category')
plt.ylabel('Count')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Is Fraud')
plt.show()
```



EDA

The section extracts hour and day of the week from train_data DataFrame, samples 10%, converts to Pandas DataFrame, creates a count plot, and analyzes fraudulent transaction frequency.

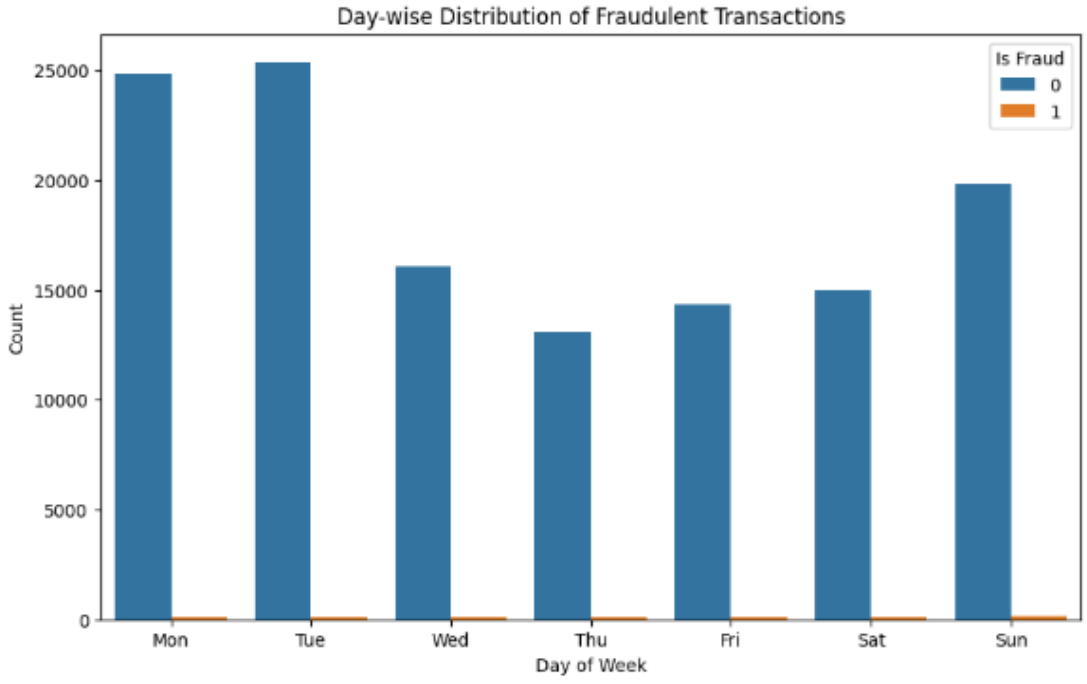
```
from pyspark.sql.functions import hour, dayofweek, col

# Time analysis: Extract hours and days from 'trans_date_trans_time'
train_data = train_data.withColumn('trans_hour', hour(col('trans_date_trans_time')))
train_data = train_data.withColumn('trans_day', dayofweek(col('trans_date_trans_time')))

# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot hourly distribution of fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(x='trans_hour', hue='is_fraud', data=train_data_sample)
plt.title('Hourly Distribution of Fraudulent Transactions')
plt.xlabel('Hour')
plt.ylabel('Count')
plt.legend(title='Is Fraud')
plt.show()
```



EDA

The section uses a Pandas DataFrame to convert 10% training data, generating a count plot to analyze the frequency of fraudulent transactions by day of the week.

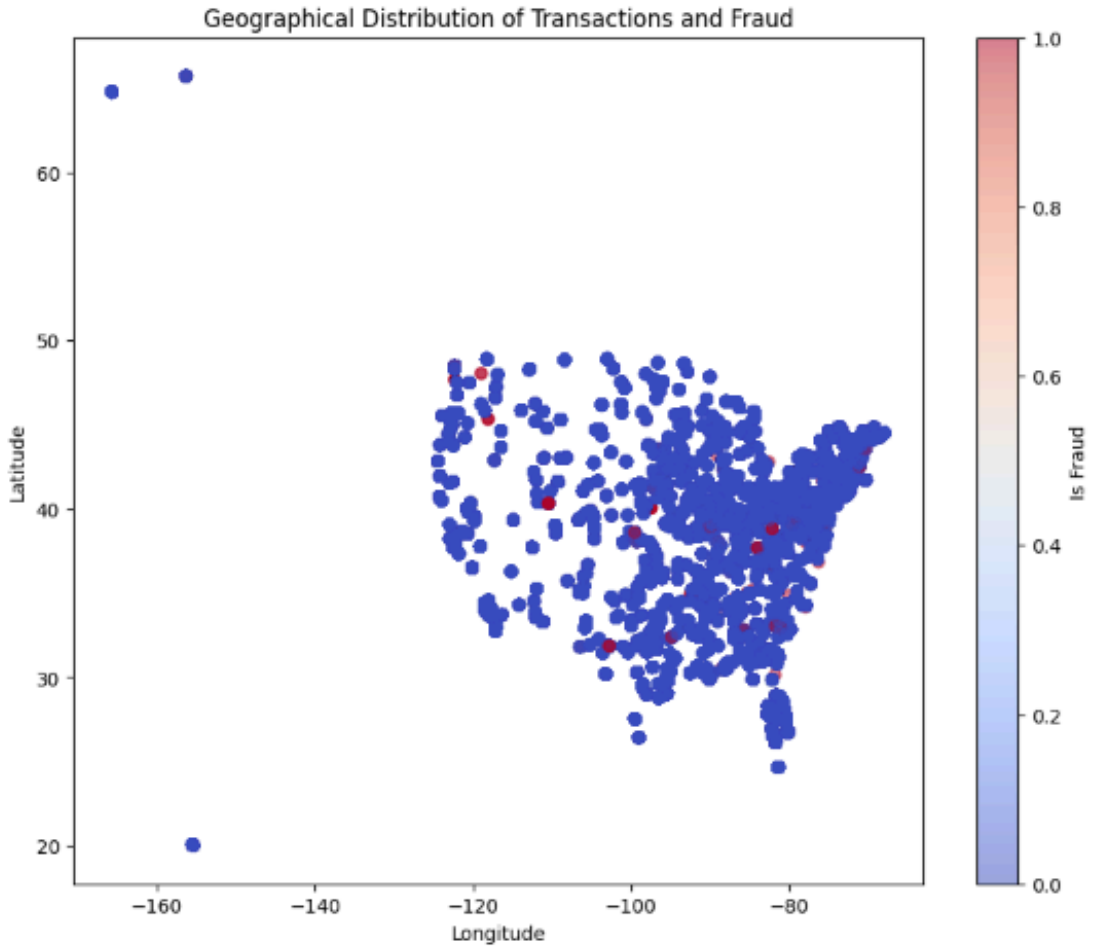
```python
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot day-wise distribution of fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.countplot(x='trans_day', hue='is_fraud', data=train_data_sample)
plt.title('Day-wise Distribution of Fraudulent Transactions')
plt.xlabel('Day of Week')
plt.ylabel('Count')
plt.xticks([0, 1, 2, 3, 4, 5, 6], ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
plt.legend(title='Is Fraud')
plt.show()
```



EDA

This section uses 10% training data to create a Pandas DataFrame, generating a scatter plot to visualize transaction geographical distribution, identifying patterns and indicating fraudulent transactions using color maps.

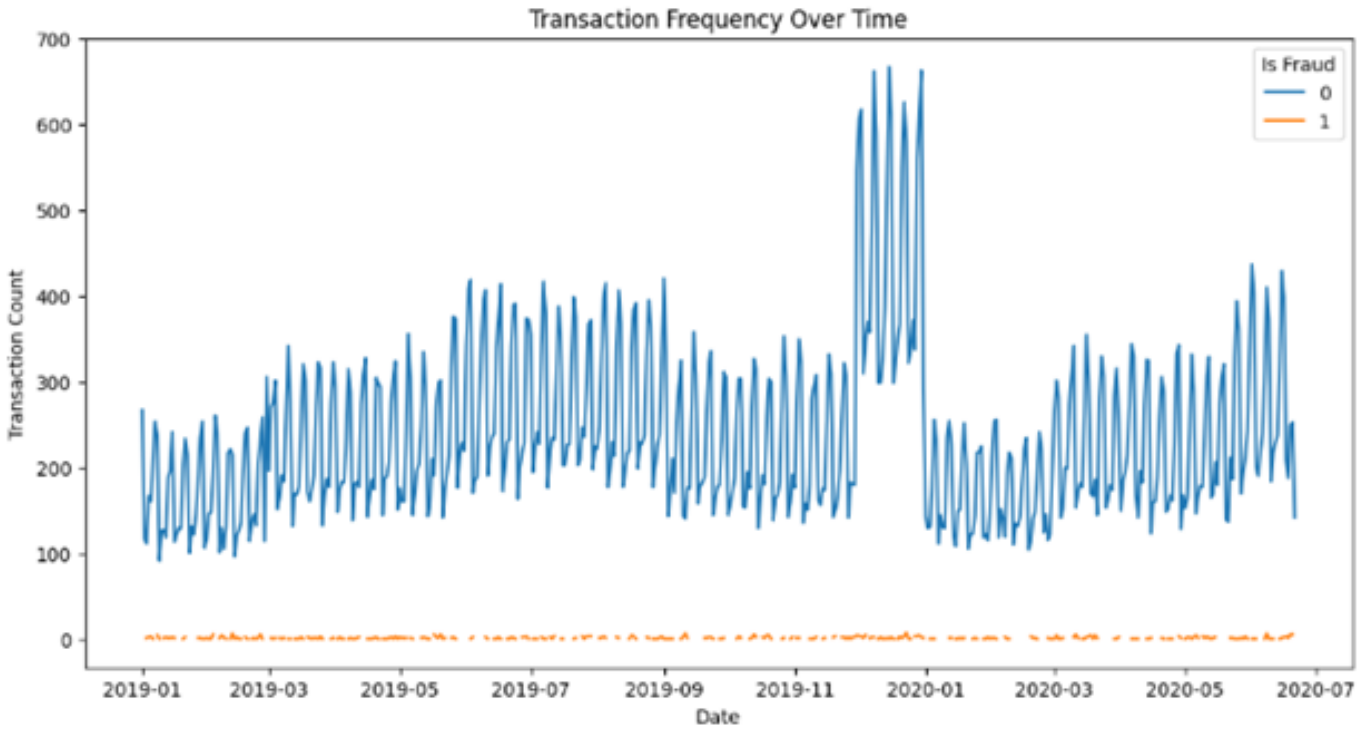
```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Scatter plot of geographical data
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
plt.scatter(train_data_sample['long'], train_data_sample['lat'], c=train_data_sample['is_fraud'],
cmap='coolwarm', alpha=0.5)
plt.title('Geographical Distribution of Transactions and Fraud')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar(label='Is Fraud')
plt.show()
```



EDA

The section converts timestamps to date format, samples 10% data, converts it to a Pandas DataFrame, groups it by date and fraud status, calculates transaction counts, and plots them over time.

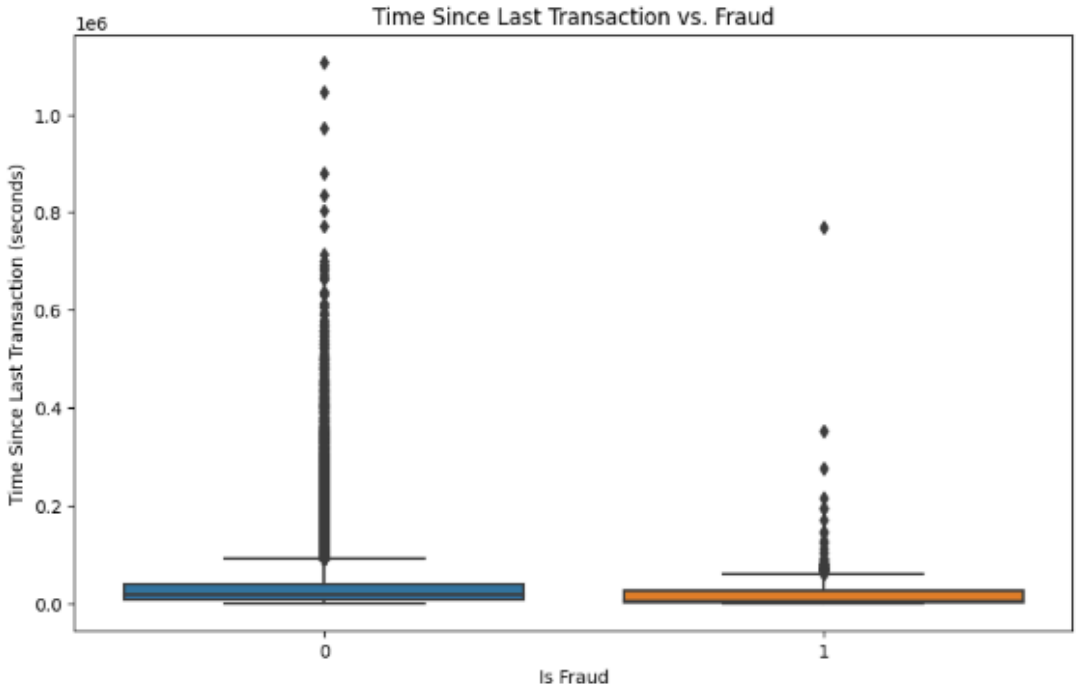
```
from pyspark.sql.functions import to_date, col

# Convert the timestamp to date
train_data = train_data.withColumn('trans_date', to_date(col('trans_date_trans_time')))

# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Group by date and fraud status, then plot
transaction_counts = train_data_sample.groupby(['trans_date', 'is_fraud']).size().unstack()
transaction_counts.plot(kind='line', figsize=(12, 6))
plt.title('Transaction Frequency Over Time')
plt.xlabel('Date')
plt.ylabel('Transaction Count')
plt.legend(title='Is Fraud')
plt.show()
```



EDA

The section calculates Unix time, calculates time difference between consecutive transactions, samples 10% data, converts it to a Pandas DataFrame, and creates a box plot using Matplotlib and Seaborn to compare fraudulent and non-fraudulent transactions.

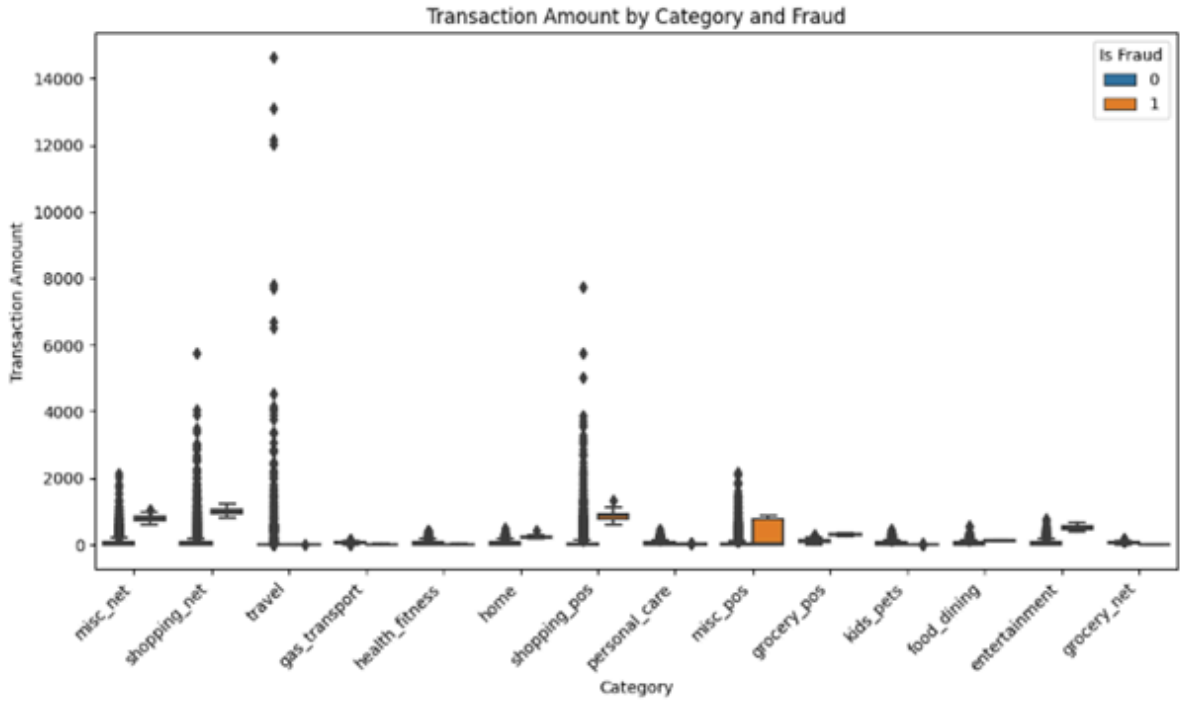
```
from pyspark.sql.window import Window
from pyspark.sql.functions import col, lag, unix_timestamp, from_unixtime

# Convert unix_time to timestamp
train_data = train_data.withColumn('unix_time_ts', from_unixtime(col('unix_time')))

# Calculate the time since the last transaction
window_spec = Window.partitionBy('cc_num').orderBy('unix_time_ts')
train_data = train_data.withColumn('time_since_last_transaction', unix_timestamp(col('unix_time_ts')) -
unix_timestamp(lag(col('unix_time_ts'), 1).over(window_spec)))

# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot time since last transaction vs. fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.boxplot(x='is_fraud', y='time_since_last_transaction', data=train_data_sample)
plt.title('Time Since Last Transaction vs. Fraud')
plt.xlabel('Is Fraud')
plt.ylabel('Time Since Last Transaction (seconds)')
plt.show()
```



EDA

The section uses a Pandas DataFrame to convert 10% training data, creating a box plot to visualize transaction amounts by category, identifying noticeable differences between fraudulent and non-fraudulent transactions.

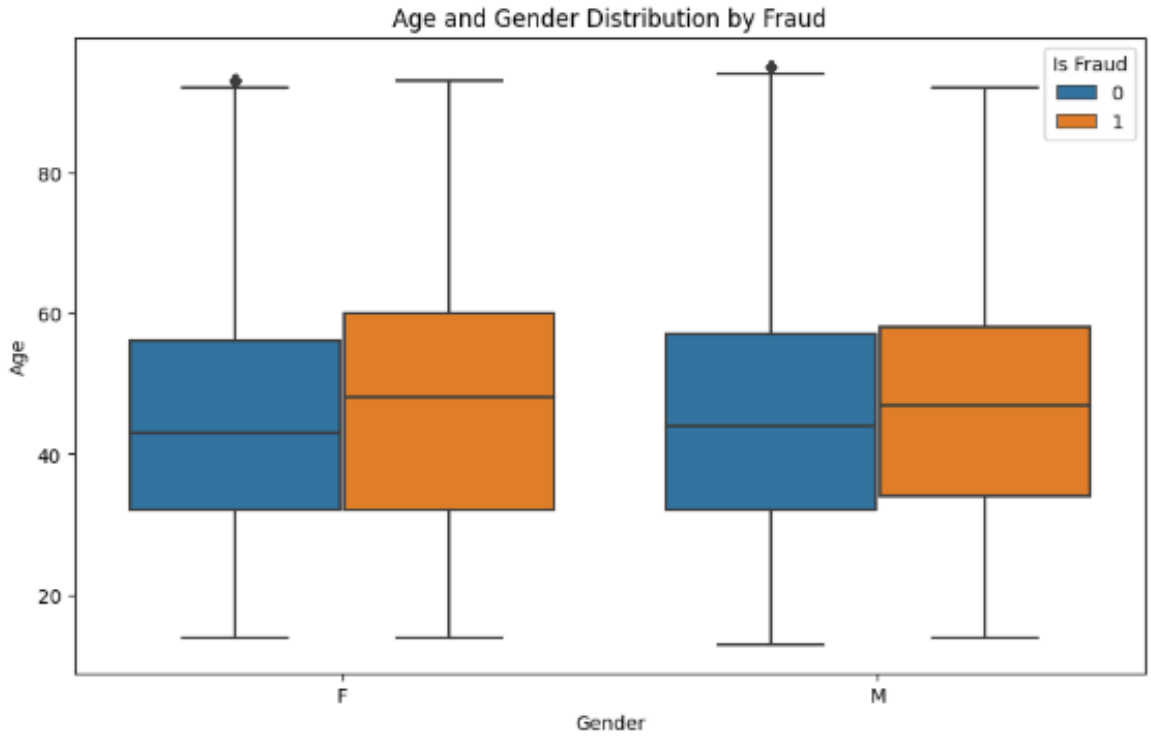
```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot transaction amount by category and fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))
sns.boxplot(x='category', y='amt', hue='is_fraud', data=train_data_sample)
plt.title('Transaction Amount by Category and Fraud')
plt.xlabel('Category')
plt.ylabel('Transaction Amount')
plt.xticks(rotation=45, ha="right")
plt.legend(title='Is Fraud')
plt.show()
```



EDA

The section converts the dob column to date format, calculates age based on transaction date difference, samples 10%, converts to Pandas DataFrame, creates box plots using Matplotlib and Seaborn, and analyzes fraud relation.

```
from pyspark.sql.functions import col, datediff, to_date

# Convert 'dob' to date and calculate 'age'
train_data = train_data.withColumn('dob', to_date(col('dob')))
train_data = train_data.withColumn('age', (datediff(col('trans_date_trans_time'), col('dob')) /
365).cast('int'))

# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot age and gender distribution by fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.boxplot(x='gender', y='age', hue='is_fraud', data=train_data_sample)
plt.title('Age and Gender Distribution by Fraud')
plt.xlabel('Gender')
plt.ylabel('Age')
plt.legend(title='Is Fraud')
plt.show()
```



EDA

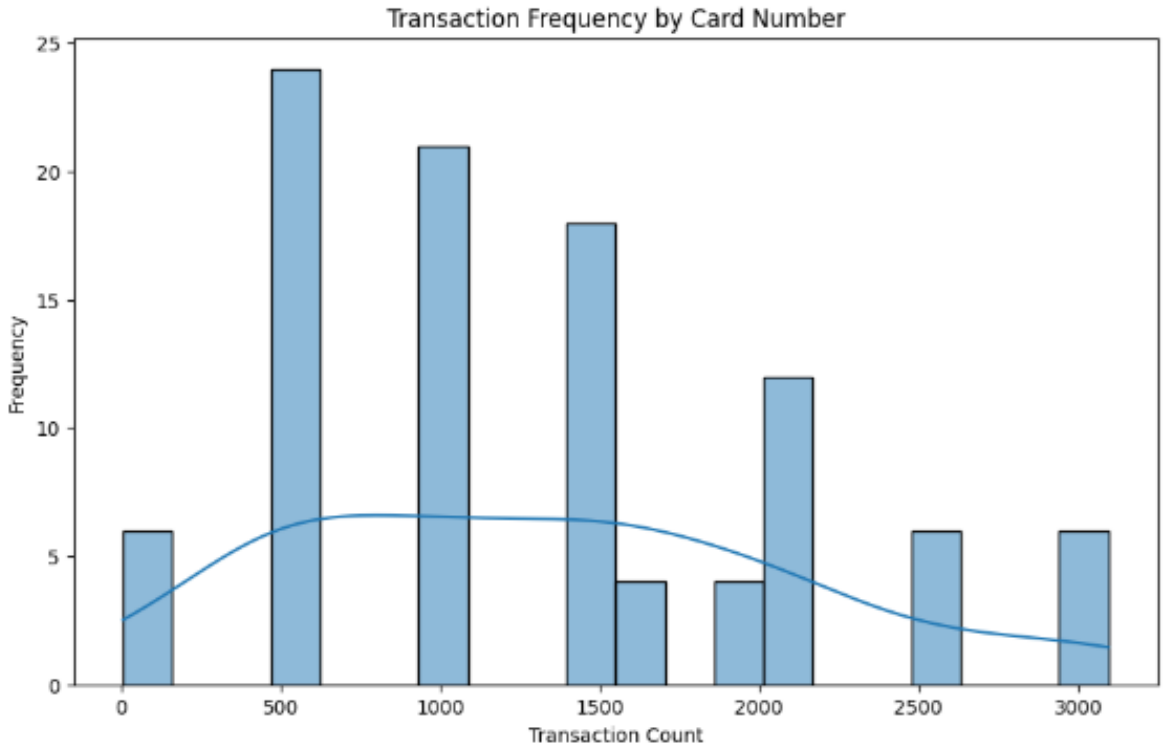
The section uses a Pandas DataFrame and Matplotlib and Seaborn to create a scatter plot analyzing the relationship between city population and transaction amount, identifying fraud patterns.

```
# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot transaction amount vs. city population and fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.scatterplot(x='city_pop', y='amt', hue='is_fraud', data=train_data_sample)
plt.title('Transaction Amount vs. City Population and Fraud')
plt.xlabel('City Population')
plt.ylabel('Transaction Amount')
plt.legend(title='Is Fraud')
plt.show()
```

EDA

The section calculates transaction frequency per credit card number, samples 10%, converts it to a Pandas DataFrame, creates a histogram, and uses Matplotlib and Seaborn for visualization and probability distribution.

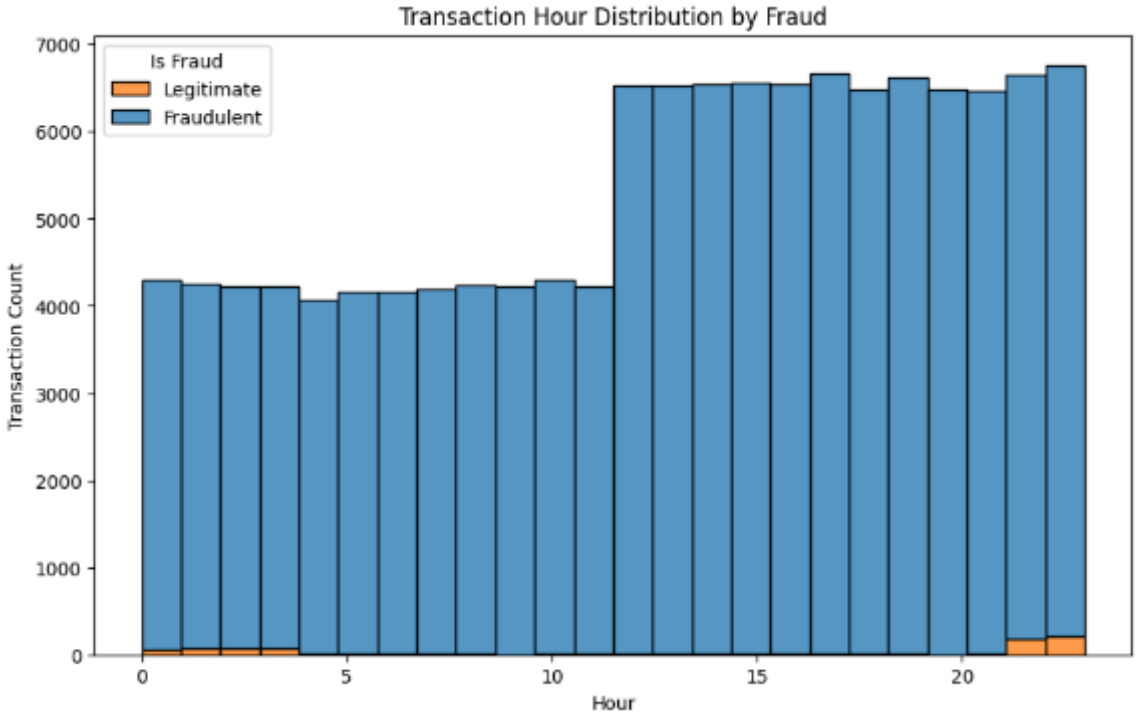
```python
from pyspark.sql.functions import col

# Count the number of transactions per card number
card_transaction_counts = train_data.groupBy('cc_num').count()

# Sample a smaller subset of the data for visualization
card_transaction_counts_sample = card_transaction_counts.sample(fraction=0.1, seed=42).toPandas()

# Plot transaction frequency by card number
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(card_transaction_counts_sample['count'], bins=20, kde=True)
plt.title('Transaction Frequency by Card Number')
plt.xlabel('Transaction Count')
plt.ylabel('Frequency')
plt.show()
```



EDA

The section extracts the hour from the trans_date_trans_time column, samples 10%, converts it to a Pandas DataFrame, creates a stacked histogram, and uses Matplotlib and Seaborn to identify fraudulent activity patterns.

```python
from pyspark.sql.functions import hour, col

# Extract the hour from 'trans_date_trans_time'
train_data = train_data.withColumn('trans_hour', hour(col('trans_date_trans_time')))

# Sample a smaller subset of the data for visualization
train_data_sample = train_data.sample(fraction=0.1, seed=42).toPandas()

# Plot transaction hour distribution by fraud
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.histplot(data=train_data_sample, x='trans_hour', hue='is_fraud', multiple='stack', bins=24)
plt.title('Transaction Hour Distribution by Fraud')
plt.xlabel('Hour')
plt.ylabel('Transaction Count')
plt.legend(title='Is Fraud', labels=['Legitimate', 'Fraudulent'])
plt.show()
```



Samsung Innovation Campus

FEATURE ENGINEERING

The data types of 'transaction date and time' and 'Date of Birth' were converted into datetime formats before removing columns. The 'is_fraud' column was converted to an integer type due to certain functions not supporting string types, using the following code.

```
combined['trans_date_trans_time'] = pd.to_datetime(combined['trans_date_trans_time'])
combined['dob'] = pd.to_datetime(combined['dob'])
combined['is_fraud'] = combined['is_fraud'].astype(int)
```

We then created the ‘Age’ column using ‘transaction date and time’ and ‘Date of Birth’ before removing them like so:

```
combined['age'] = (combined['trans_date_trans_time'] - combined['dob']).dt.days // 365
```

We manipulated the columns this way since the age of the card holder will probably be influential in our model instead of the ‘Date of Birth’

Samsung Innovation Campus

DATA PREPROCESSING

We removed unwanted column such as identifiers like ‘c_num’ as follows:

```
combined = combined.drop(columns=['Unnamed: 0','cc_num','merchant','trans_num','unix_time','first',
                                  'last','street','zip' , 'trans_date_trans_time' , 'date' , 'dob' ,
'lat' , 'long' , 'merch_lat' , 'merch_long','time', 'job' ,  'city'])
```

'Date' was excluded, as we want to predict fraudulent transactions for the future, and adding 'date' will not add much value to our model.

The 'city' variable was excluded due to the dataset's over 900 unique cities, making dummy variables cumbersome. The'state' column, representing the cardholder's residence state, was used, making 'city' redundant.

The 'job' variable was removed from the dataset after re-checking, as it did not significantly improve prediction accuracy due to its cumbersome nature in creating dummy variables.

Samsung Innovation Campus

DATA PREPROCESSING

To simplify the process of converting to dummy variables later, we observed the categories we have using the code below:

```
combined['category'].value_counts()
```

The data set consists of 14 categories, which were combined into similar categories like 'transportation' using the code below to finally have 5 categories instead of 14

```
processed_df['category'] = processed_df['category'].replace(['gas_transport', 'travel'],
'transportation')
processed_df['category'] = processed_df['category'].replace(['home',  'personal_care', 'kids_pets'],
'household')
processed_df['category'] = processed_df['category'].replace(['shopping_pos', 'shopping_net' ,
'grocery_pos' , 'grocery_net'], 'shopping&grocery')
processed_df['category'] = processed_df['category'].replace(['entertainment', 'misc_pos' ,'misc_net' ],
'entertainment&others')
processed_df['category'] = processed_df['category'].replace(['food_dining', 'health_fitness'],
'food&health')
```

DATA PREPROCESSING

Most of the ML algorithms don't deal with string data types, hence, we need 'encode' it with integers instead of strings using the code below:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
processed_df['gender'] = le.fit_transform(processed_df['gender'])
# 1 for males and 0 for females

processed_df.head()
```

	category	amt	gender	state	city_pop	is_fraud	age	trans_hour
0	household	2.86	1	SC	333497	0	52	12
1	household	29.84	0	UT	302	0	30	12
2	food&health	41.28	0	NY	34496	0	49	12
3	entertainment&others	60.05	1	FL	54767	0	32	12
4	transportation	3.19	1	MI	1126	0	65	12

Samsung Innovation Campus

DATA PREPROCESSING

Regarding the ‘category’ and ‘state’ columns we one-hot encoding and then concatenating the new dummy variables to our dataframe using the code below:

```
# Generate dummy variables for 'category' and 'state' columns
category_dummies = pd.get_dummies(processed_df['category'], prefix='category', drop_first=True)
state_dummies = pd.get_dummies(processed_df['state'], prefix='state', drop_first=True)

# Convert dummies to integer type instead of boolean
category_dummies = category_dummies.astype(int)
state_dummies = state_dummies.astype(int)

# Concatenate the original DataFrame with the dummies DataFrames
processed_df = pd.concat([processed_df, category_dummies, state_dummies], axis=1)

# Drop the original 'category' and 'state' columns
processed_df = processed_df.drop(columns=['category', 'state'])
```

Samsung Innovation Campus

DATA PREPROCESSING

The analysis reveals significant differences in numerical columns, such as 'city_pop' in thousands and 'age' in two digits, which our algorithms may prioritize due to their greater variation. We will use standardization as follows:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit and transform the specified columns
processed_df[['amt' , 'city_pop' ,'age' , 'trans_hour']] = scaler.fit_transform(processed_df[['amt' ,
'city_pop','age' , 'trans_hour']])
```

Now the dataset looks like this

	amt	gender	city_pop	is_fraud	age	trans_hour
0	-0.421990	1	0.812151	0	0.356011	-0.118273
1	-0.252575	0	-0.293019	0	-0.906621	-0.118273
2	-0.180740	0	-0.179602	0	0.183834	-0.118273
3	-0.062878	1	-0.112365	0	-0.791836	-0.118273
4	-0.419918	1	-0.290286	0	1.102112	-0.118273

MACHINE LEARNING

Introduction

This part outlines the implementation of 4 machine learning models for fraud detection. The dataset was highly imbalanced, and different techniques were applied to handle this imbalance.

The fraud detection dataset was highly imbalanced:

- Legitimate Transactions: 99.42%
- Fraudulent Transactions: 0.58%

To address this, we applied oversampling on the minority class (is_fraud = 1) to balance the dataset for model training in logistic regression & XGBoost, while using increased weights for the minority class for Random Forest and SVM.

Oversampling is a technique used to address class imbalance in datasets by increasing the number of samples in the minority class. This is typically done by randomly duplicating existing minority class samples or generating new synthetic samples

Increasing the weight of the minority class is a technique used to address class imbalance in machine learning models. By assigning higher weights to the minority class, the model is encouraged to pay more attention to its patterns, thereby reducing bias towards the majority class. This approach helps improve the detection of minority class instances, which is crucial in applications such as fraud detection

MACHINE LEARNING

LOGISTIC REGRESSION

Logistic Regression is a supervised machine learning algorithm that is majorly used for binary classification. It predicts the probability of an input belonging to a particular class by modeling the relationship between the dependent and one or more independent variables using a logistic function

Logistic Regression Features:

• Binary Classification: Used for problems with binary outcomes like yes/no, true/false, or 0/1.

• Logistic Function: Uses the logistic (sigmoid) function to map predicted values to probabilities, producing outputs between 0 and 1.

• Linear Relationship: Models a linear relationship between input features and outcome log-odds.

• Interpretability: Coefficients represent the change in log-odds of the outcome for a one-unit change in the predictor variable.

Logistic Regression finds its application in many areas such as finance and social sciences due to its simplicity and efficiency hence effectiveness.

MACHINE LEARNING

LOGISTIC REGRESSION

In this section, we address the issue of class imbalance in the dataset by oversampling the minority class. This is done to ensure that the model has sufficient data to learn from both classes, which improves the performance and generalization of the model. Below is the code used to achieve this:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
# Step 1: Oversample the minority class (is_fraud = 1)
# Separate the majority and minority classes
from pyspark.sql.functions import col

# Separate the majority and minority classes
majority_class = df.filter(col("is_fraud") == 0)
minority_class = df.filter(col("is_fraud") == 1)

# Oversample the minority class by replicating rows
oversample_factor = int(majority_class.count() / minority_class.count())

# Use union to concatenate the minority class multiple times
oversampled_minority = minority_class
for _ in range(oversample_factor - 1):
    oversampled_minority = oversampled_minority.union(minority_class)

# Combine oversampled minority class and majority class
balanced_df = majority_class.union(oversampled_minority)

# Verify the new class distribution
balanced_df.groupBy("is_fraud").count().show()
```

```
[6]
...  [Stage 12:=====================================================>(763 + 1) / 764]
     +--------+-------+
     |is_fraud|  count|
     +--------+-------+
     |       0|1842743|
     |       1|1833690|
     +--------+-------+
```

MACHINE LEARNING

LOGISTIC REGRESSION

We selected all columns except is_fraud as features and used VectorAssembler to combine them into a single feature vector. The result is stored in a new column called "features" in the DataFrame. This prepares the data for machine learning models.

```
from pyspark.ml.feature import VectorAssembler

# Select features and assemble them into a single vector
feature_columns = [col for col in balanced_df.columns if col != 'is_fraud']
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_df = assembler.transform(balanced_df)
```

In this section, we split the data into training and test sets and train a Logistic Regression model using PySpark. Below is the code used for these tasks:

```
# Split the data into training and test sets (80% training, 20% test
train_df, test_df = assembled_df.randomSplit([0.8, 0.2], seed=42)
from pyspark.ml.classification import LogisticRegression

# Initialize Logistic Regression model
lr = LogisticRegression(featuresCol="features", labelCol="is_fraud")

# Train the model
lr_model = lr.fit(train_df)
# Make predictions on the test data
predictions = lr_model.transform(test_df)
```

MACHINE LEARNING

LOGISTIC REGRESSION

Logistic Regression Findings

Logistic Regression is a supervised machine learning algorithm that is majorly used for binary classification. It predicts the probability of an input belonging to a particular class by modeling the relationship

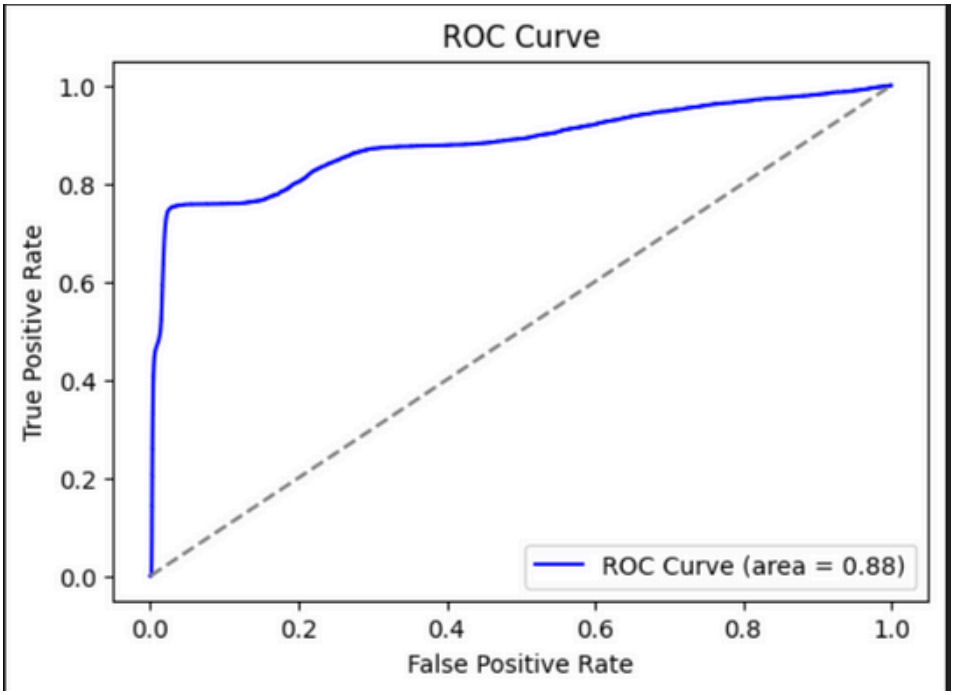
CASE	PRECISION	RECALL	F1-SCORE
LEGITIMATE	0.79	0.93	0.85
FRAUDULENT	0.91	0.75	0.83
ACCURACY	0.84		
AUC	0.86		

MACHINE LEARNING

LOGISTIC REGRESSION

In this section, we evaluate the performance of the Logistic Regression model using the ROC (Receiver Operating Characteristic) Curve. The ROC Curve is a graphical representation of the true positive rate (TPR) versus the false positive rate (FPR) at various threshold settings.

```
import pandas as pd
import matplotlib.pyplot as plt
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from sklearn.metrics import roc_curve

# Compute false positive rate (fpr), true positive rate (tpr) and thresholds
fpr, tpr, thresholds = roc_curve(preds_pd['is_fraud'],
preds_pd['probability_fraud'])
# Plot ROC Curve
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
plt.title('ROC Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

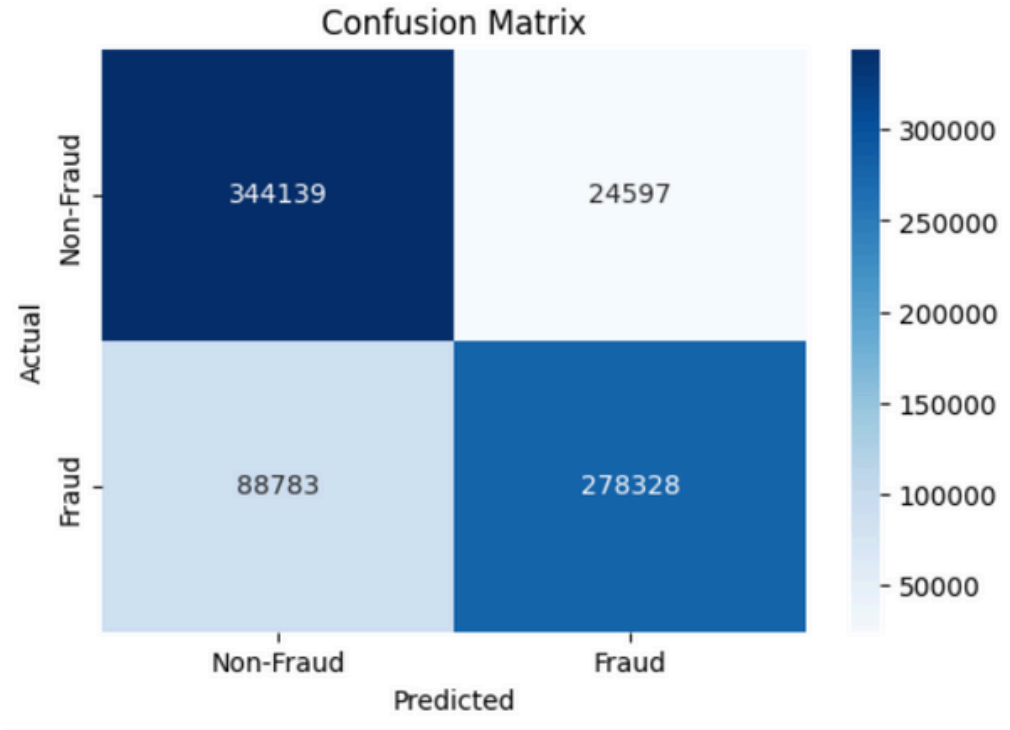


MACHINE LEARNING

LOGISTIC REGRESSION

The confusion matrix provides a summary of the prediction results of the classification model by showing the counts of true positive, true negative, false positive, and false negative predictions. This helps in understanding the performance of the model in detail.

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

# Compute confusion matrix
cm = confusion_matrix(preds_pd['is_fraud'], preds_pd['prediction'])

# Plot confusion matrix
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['Non-Fraud', 'Fraud'], yticklabels=['Non-
Fraud', 'Fraud'])
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

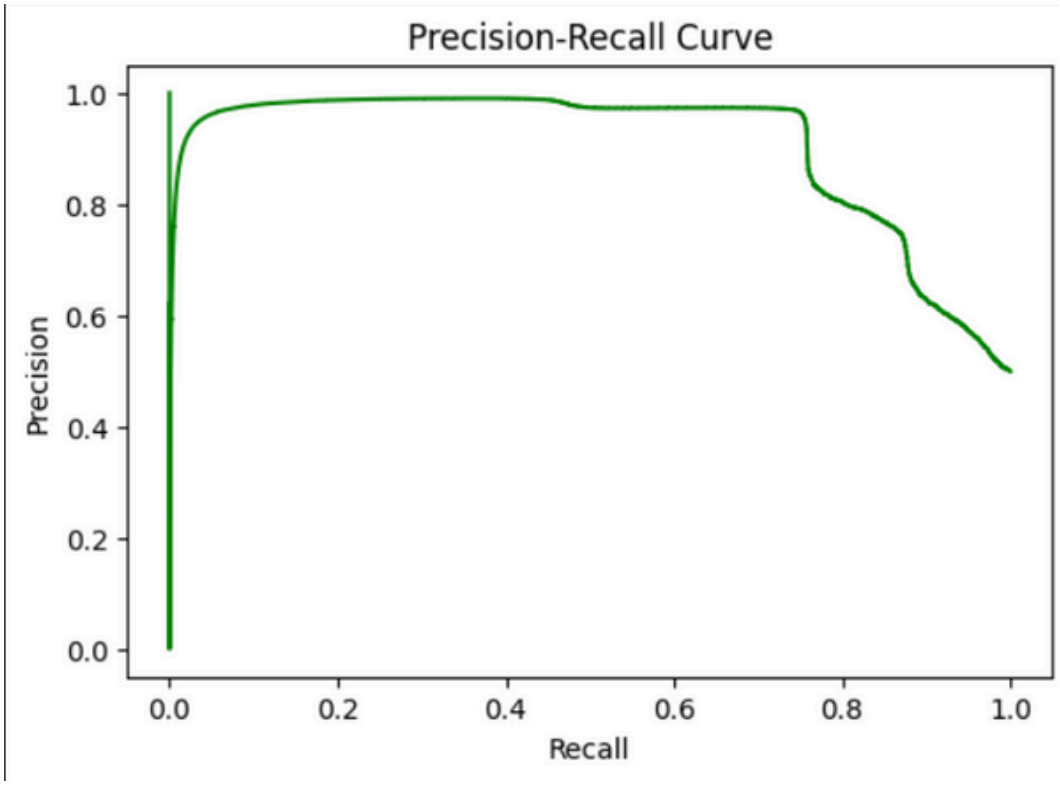


MACHINE LEARNING

LOGISTIC REGRESSION

The Precision-Recall (PR) Curve is used to evaluate the performance of a classification model, especially in cases of imbalanced datasets. It plots precision (the ratio of true positive predictions to the total predicted positives) against recall (the ratio of true positive predictions to the total actual positives) for different threshold values.

```
from sklearn.metrics import precision_recall_curve

# Compute precision and recall
precision, recall, thresholds_pr = precision_recall_curve(preds_pd['is_fraud'],
preds_pd['probability_fraud'])

# Plot Precision-Recall Curve
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='green')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



MACHINE LEARNING

RANDOM FOREST

Random Forest is an ensemble learning technique applied for classification and regression. This versatile, ever-used machine learning algorithm creates a great number of trees for developing the general performance of prediction, at the same time reducing overfitting. It works by creating many decision trees during the training process and outputting either the mode of classes-in case of classification problems-or the mean prediction regarding regression-from all the generated trees. Some of the key characteristics of Random Forest are as follows:

• Ensemble Learning: It combines the predictions over a set of decision trees to get a more general and stable result.
• Bagging Method: It does bootstrap aggregating, or bagging, by creating diversity in trees through training every tree on different sub-samples of the data at random.
• Random Feature Selection: During each split of the tree, a random subset of features is taken for consideration for model robustness and reduction in overfitting.
• Versatility: It can be used for both classification and regression problems; hence, flexibility for a variety of tasks in machine learning.

Random Forest is particularly applicable for big datasets with higher dimensionalities. It may provide insight into feature importance that can also help in interpretability of the model.

MACHINE LEARNING

RANDOM FOREST

Hyperparameters in Random Forest

Several of the following hyperparameters tuning can be very influential in providing a good performance of the Random Forest model. For instance:

1. n_estimators: This is the number of trees in the forest. Usually, increasing the number of trees improves the performance of the model but increases computational costs.
2. max_depth: The maximum depth of each tree. It regulates complexity of the model and helps to avoid overfitting.

Therefore, to be able to determine what parameter we will use to determine the most suitable parameters, we will apply the randomized search method.

Randomized Search Algorithm

Randomized Search optimizes hyperparameters for machine learning model performance by sampling a fixed number of hyperparameter combinations from a given distribution, unlike Grid Search.
Randomized Search Characteristics
• Efficient: Reduces computational cost by sampling a subset of hyperparameter combinations.
• Flexibility: Allows different distributions for hyper-parameters, adaptable to various data and models.
• Scalability: Can handle large datasets and complex models by identifying promising hyper-parameters' settings early.
• Performance: Provides as good hyper-parameters' settings as Grid Search, especially when some hyper-parameters have minimal effect on model performance.

MACHINE LEARNING

RANDOM FOREST

Findings

For the optimal number of estimators in our random forest model, we do a randomized search best practices, which returned 47 trees as the best option and a maximum depth of 10.

We can show this by using 47 trees and changing the maximum depth between 1 and 15 using this code:

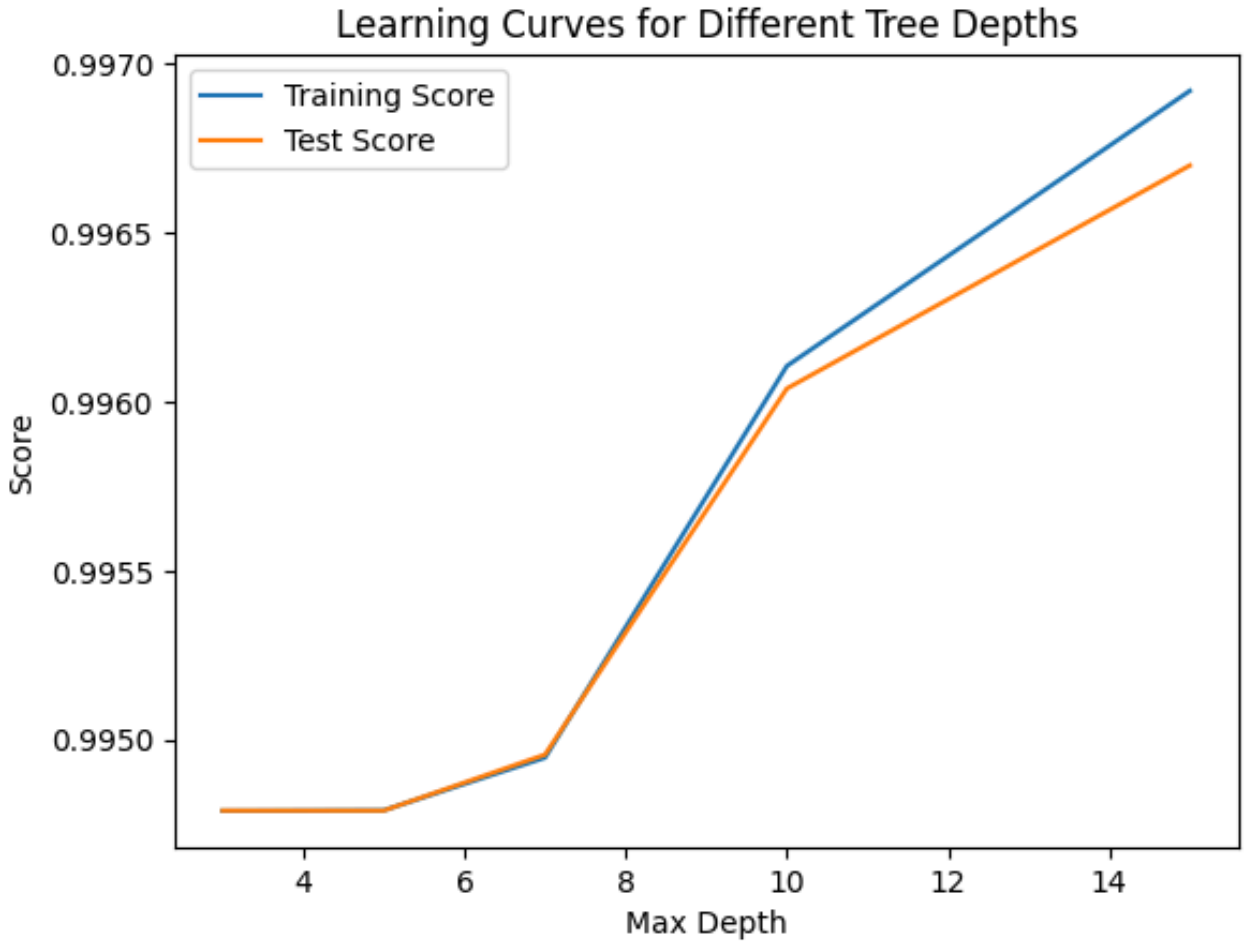
```
import matplotlib.pyplot as plt

depths = [3, 5, 7, 10, 15]
train_scores = []
test_scores = []

for depth in depths:
    rf = RandomForestClassifier(max_depth=depth, n_estimators=47)
    rf.fit(X_train, y_train)
    train_scores.append(rf.score(X_train, y_train))
    test_scores.append(rf.score(X_test, y_test))

plt.plot(depths, train_scores, label='Training Score')
plt.plot(depths, test_scores, label='Test Score')
plt.xlabel('Max Depth')
plt.ylabel('Score')
plt.title('Learning Curves for Different Tree Depths')
plt.legend()
plt.show()
```

MACHINE LEARNING

RANDOM FOREST

Findings

The output from the previous code is shown below:



The gain in accuracy by moving from 10 to 15 is very minor, with an increase of almost 0.0005 at depth 15 compared to 10.

This is no significant improvement; hence, we stick to a maximum depth of 10 to keep the trees simpler with acceptable accuracy.

Samsung Innovation Campus

MACHINE LEARNING
RANDOM FOREST

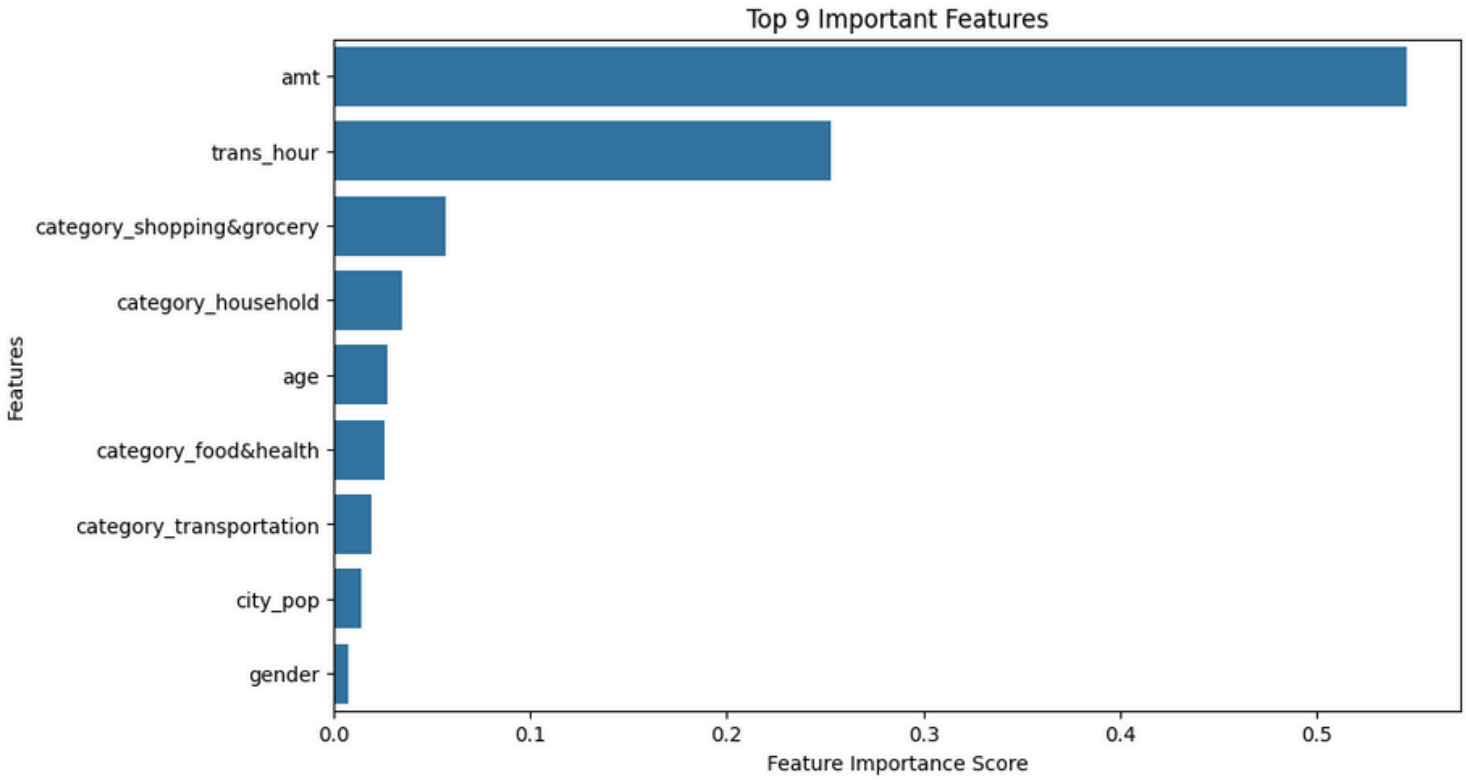
Feature Importance

Feature importance in Random Forest is a measure of each feature's contribution to model prediction, calculated by evaluating the decrease in impurity each time a feature is used to split data.

```python
#feature importance random forest
feature_imp = pd.Series(model.feature_importances_,index=columns).sort_values(ascending=False)
import seaborn as sns
import matplotlib.pyplot as plt

# Sort feature importance in descending order and select the top 9
top_n = 9
sorted_indices = feature_imp.sort_values(ascending=False)[:top_n]

# Visualize feature importance for the top 9 features
plt.figure(figsize=(10, 6))  # Adjust the figure size for clarity
sns.barplot(x=sorted_indices, y=sorted_indices.index)

# Add labels and title to the graph
plt.xlabel('Feature Importance Score')
plt.ylabel('Features')
plt.title("Top 9 Important Features")
plt.show()
```

Samsung Innovation Campus

MACHINE LEARNING
RANDOM FOREST



The amount is the most crucial factor in fraudulent transaction prediction, followed by the transaction's hour. Other critical categories are also identified in previous research.

MACHINE LEARNING

RANDOM FOREST

Evaluation Metrics

We will use measures such as recall and accuracy to evaluate our mode.

```
from sklearn.metrics import classification_report

# Making predictions on the test set
y_pred = model.predict(X_test)

# Generate the classification report
report = classification_report(y_test, y_pred)
print(report)
```

MACHINE LEARNING

RANDOM FOREST

Evaluation Metrics

The code above shows the following results:

CASE	PRECISION	RECALL	F1-SCORE
LEGITIMATE	1	0.97	0.99
FRAUDULENT	0.15	0.88	0.25
ACCURACY	0.97		
AUC	0.98		

The accuracy metric may not accurately represent our situation due to an imbalanced dataset. The recall of fraudulent transactions, initially 0.47, was adjusted to 0.88 by increasing fraud weight.

Recall is crucial for detecting fraudulent transactions, ensuring most activities are identified, reducing undetected fraud and financial consequences. Prioritizing recall helps capture as many fraudulent transactions as possible, despite potential false positives.

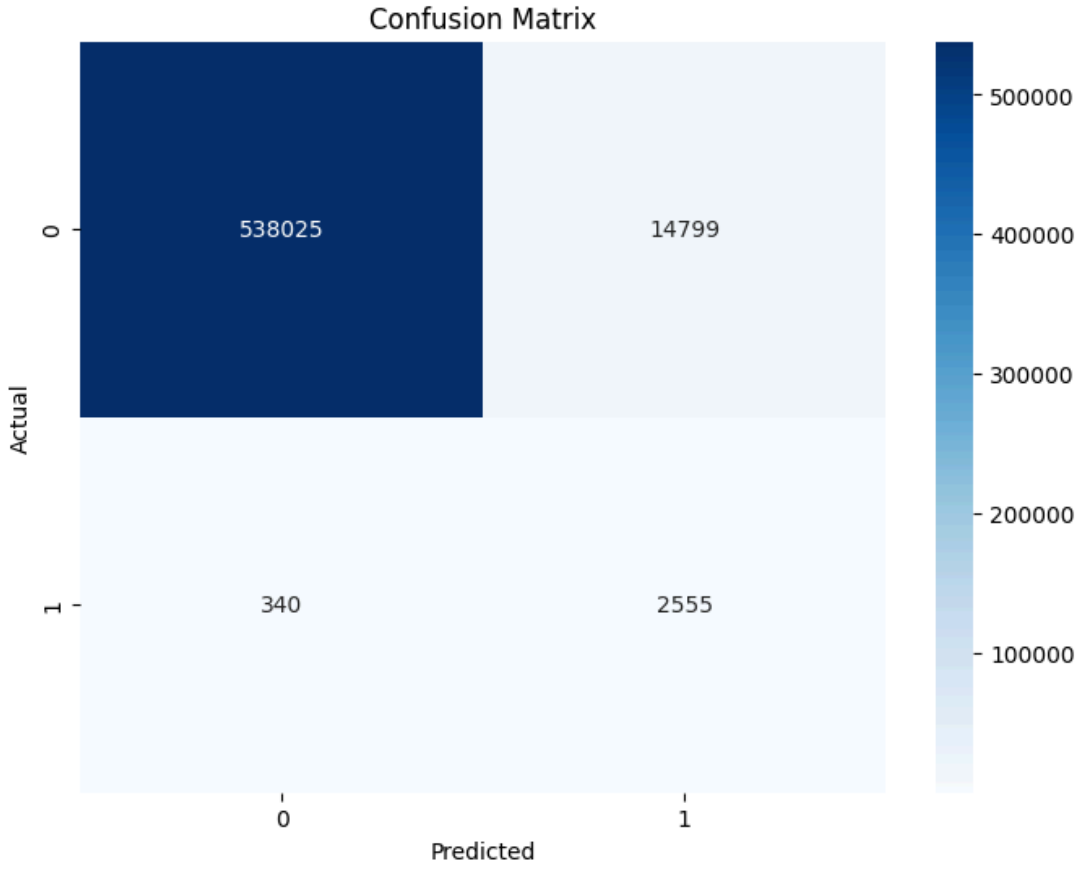
MACHINE LEARNING

RANDOM FOREST

Evaluation Metrics

This code displays the confusion matrix:

```python
from sklearn.metrics import classification_report

# Making predictions on the test set
y_pred = model.predict(X_test)

# Generate the classification report
report = classification_report(y_test, y_pred)
print(report)
```



MACHINE LEARNING

SVM

Support Vector Machine (SVM)

The Support Vector Machine is a powerful 'supervised' machine learning algorithm which is used both for classification and in regression analysis, though it is applied more commonly to classification. Support Vector Machine's primary objective is to find out the best hyperplane that separates the data points of different classes in an N-dimensional space.

The important features of support vector machines are:

Optimal Hyperplane: It involves the generation of a hyperplane or a set of hyperplanes in higher-dimensional space, wherein the margin between the classes gets maximized.

Support Vectors: These are data points nearest to the hyperplane, upon which it rests. They are very important in setting up the position and orientation of the hyperplane.

Kernel Trick: SVM is very efficient in performing nonlinear classification by doing an implicit mapping of input features into high-dimensional feature space with the so-called kernel trick.

Robustness to Overfitting: Thanks to margin maximization, SVM is expected to be more robust against overfitting attacks, particularly in high-dimensional spaces.

MACHINE LEARNING

SVM

SVM Findings

SVMs are well-known to be inefficient and very computationally expensive when dealing with big data. Thus, it took the model fitting almost three hours to compute its results. We have decided to show this algorithm on only a 10% sample of the entire dataset, then we split the data to fit the model.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

sample_df = processed_df.sample(frac=0.1, random_state=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size = 0.3, random_state = 0)

model = SVC(kernel='rbf' , class_weight='balanced' , random_state = 1)
model.fit(X_train, y_train)
```

To evaluate the model we used this code to calcualte the same metrics as above:

```
from sklearn.metrics import classification_report

y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred)
print(report)
```

MACHINE LEARNING

SVM

The output from the previous code shows that:

CASE	PRECISION	RECALL	F1-SCORE
LEGITIMATE	1	0.97	0.99
FRAUDULENT	0.14	0.78	0.23
ACCURACY	0.97		
AUC	0.96		

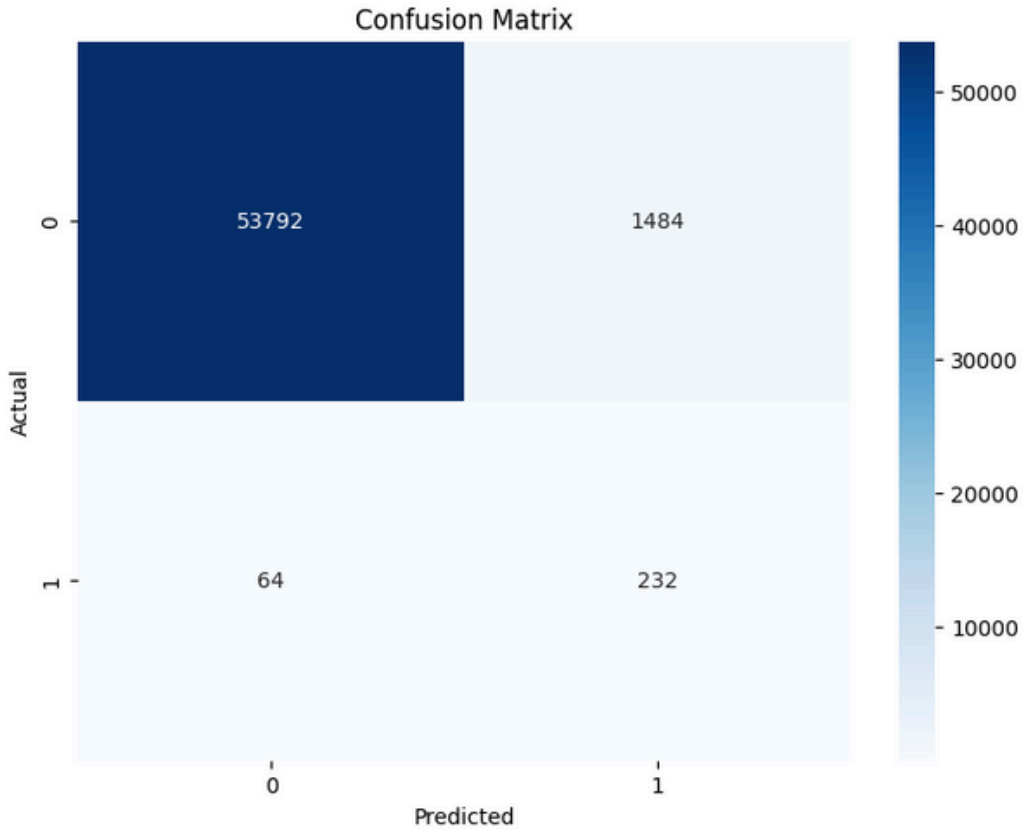
The recall isn't better than the random forest while the accuracy remain the same, so far we can say that the random forest is better so far when it comes to performance and time efficiency.

MACHINE LEARNING

SVM

The confusion matrix was also calculated using the following code:

```
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



MACHINE LEARNING

XGBOOST

XGBoost is a parallelizable, efficient version of gradient boosting used in machine learning for improved model performance and computational power in large data volumes.

XGBoost Features:

• Gradient Boosting Framework: Minimizes prediction errors for improved model accuracy.
• Parallel Processing: Speeds up training process by building trees in parallel.
• Regularization: Incorporates L1(Lasso) and L2(ridge) regularization to prevent overfitting and enhance model generalization.
• Handling Missing Data: Automatically learns best imputation strategy during training.
• Flexibility: Supports various objective functions and evaluation metrics for a wide range of machine learning tasks.

MACHINE LEARNING

XGBOOST

This PySpark code groups the dataset by the is_fraud column and counts the occurrences of each value, displaying the distribution of fraudulent (is_fraud = 1) and legitimate (is_fraud = 0) transactions.

```
# Group by the 'is_fraud' column and count occurrences of each value
df.groupBy('is_fraud').count().show()
```

```
[Stage 3:==============>                                          (1 + 3) / 4]
+--------+-------+
|is_fraud|  count|
+--------+-------+
|       1|   9651|
|       0|1842743|
+--------+-------+
```

MACHINE LEARNING

XGBOOST

This PySpark code performs oversampling to balance an imbalanced dataset by replicating rows of the minority class (is_fraud = 1). It combines the oversampled minority class with the majority class to create a balanced dataset, which is then verified using a class distribution check.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
# Step 1: Oversample the minority class (is_fraud = 1)
# Separate the majority and minority classes
from pyspark.sql.functions import col

# Separate the majority and minority classes
majority_class = df.filter(col("is_fraud") == 0)
minority_class = df.filter(col("is_fraud") == 1)

# Oversample the minority class by replicating rows
oversample_factor = int(majority_class.count() /
minority_class.count())
# Use union to concatenate the minority class multiple times
oversampled_minority = minority_class
for _ in range(oversample_factor - 1):
    oversampled_minority = oversampled_minority.union(minority_class)

# Combine oversampled minority class and majority class
balanced_df = majority_class.union(oversampled_minority)

# Verify the new class distribution
balanced_df.groupBy("is_fraud").count().show()
```

```
[Stage 12:=====================================================>(761 + 3) / 764]
+--------+-------+
|is_fraud|  count|
+--------+-------+
|       0|1842743|
|       1|1833690|
+--------+-------+
```

MACHINE LEARNING

XGBOOST

This PySpark code selects all feature columns from the balanced dataset, excluding the is_fraud column, and uses VectorAssembler to combine these features into a single vector column named features. This transformation prepares the data for machine learning models.

```
from pyspark.ml.feature import VectorAssembler

# Select features and assemble them into a single vector
feature_columns = [col for col in balanced_df.columns if col != 'is_fraud']
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_df = assembler.transform(balanced_df)
```

This PySpark code samples 10% of the data for model training using the sampling method, then converts the selected data to a pandas DataFrame. The features (X) are extracted as an array from the features column, and the target variable (y) is extracted from the is_fraud column to prepare the data for use in an XGBoost model.

```
# Sample 10% of the data
sampled_df = assembled_df.sample(fraction=0.1, seed=42)
# Import required libraries
import numpy as np
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Select the 'features' and 'label' columns
pandas_df = sampled_df.select("features", "is_fraud").toPandas()

# Separate the features and target variable
X = np.array(pandas_df['features'].tolist())
y = pandas_df['is_fraud'].values
```

MACHINE LEARNING

XGBOOST

This code splits the data into training and testing sets using an 80-20 ratio. It then creates DMatrix objects for XGBoost and sets parameters for the binary classification task, including max_depth, eta, and objective. The model is trained for 100 boosting rounds, predictions are made on the test set, and the accuracy of the model is calculated and displayed.

```python
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create the DMatrix for XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set XGBoost parameters
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    'max_depth': 6,
    'eta': 0.3
}

# Train the model
xgboost_model = xgb.train(params, dtrain, num_boost_round=100)
# Make predictions
y_pred = xgboost_model.predict(dtest)
predictions = [1 if val > 0.5 else 0 for val in y_pred]

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
[16]
...  Accuracy: 98.88%
```

MACHINE LEARNING

XGBOOST

This section of the code generates predictions from the XGBoost model by classifying values above 0.5 as fraudulent transactions (1) and others as legitimate (0). It then evaluates the model's performance by calculating the accuracy score based on the test data and prints the accuracy as a percentage.

```
# Make predictions
y_pred = xgboost_model.predict(dtest)
predictions = [1 if val > 0.5 else 0 for val in y_pred]

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
[16]
...  Accuracy: 98.88%
```

This code calculates the Area Under the Curve (AUC) for the Receiver Operating Characteristic (ROC) by predicting probabilities (y_pred_prob) from the XGBoost model. The AUC metric evaluates the model's ability to distinguish between classes, with higher values indicating better performance. The AUC score is printed with four decimal precision.

```
from sklearn.metrics import roc_auc_score

# Make predictions (using probabilities instead of binary labels
y_pred_prob = xgboost_model.predict(dtest)

# Calculate AUC
auc = roc_auc_score(y_test, y_pred_prob)
print(f"AUC: {auc:.4f}")
```

```
[17]
...  AUC: 0.9989
```

MACHINE LEARNING
XGBOOST

This code converts the predicted probabilities (y_pred_prob) into binary labels using a threshold of 0.5, where values greater than 0.5 are classified as fraud (1) and others as non-fraud (0). The classification_report from Scikit-learn is then generated to provide detailed performance metrics such as precision, recall, F1-score, and support for both classes.

```python
from sklearn.metrics import classification_report

# Make predictions (using 0.5 threshold to convert probabilities to binary labels)
y_pred_labels = [1 if val > 0.5 else 0 for val in y_pred_prob]

# Print classification report
print(classification_report(y_test, y_pred_labels))
```

CASE	PRECISION	RECALL	F1-SCORE
LEGITIMATE	1	0.98	0.99
FRAUDULENT	0.98	1	0.99
ACCURACY	0.99		
AUC	0.98		

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We installed PySpark to enable distributed data processing and machine learning using Python.

```
!pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.2.tar.gz (317.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.3/317.3 MB 4.8 MB
```

We imported key libraries for building machine learning models with PySpark, including feature scaling, classifiers, regressors, and evaluation tools. Then, we initialized a Spark session to set up the Spark environment for running machine learning tasks.

```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StandardScaler, StringIndexer, VectorAssembler
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, LinearSVC
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator, RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# Initialize Spark session
spark = SparkSession.builder \
    .appName("ML") \
    .getOrCreate()
```

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We read a CSV file into a Spark DataFrame using the spark.read.csv() method, enabling automatic schema detection and including headers. Then, we displayed the first five rows of the DataFrame using df.show(5).

```
# Read the CSV file
df = spark.read.csv("/kaggle/input/mlmodel/big_processed_data.csv", header=True,
inferSchema=True)

# Show the first few rows
df.show(5)
```

```
+-------------------+------+-------------------+--------+-------------------+-------------------+-------------------+-------------------+----------
|                amt|gender|           city_pop|is_fraud|                age|         trans_hour|category_food&health|category_household|category_sh
+-------------------+------+-------------------+--------+-------------------+-------------------+-------------------+-------------------+----------
| -0.4219900050245071|     1|  0.8121507374895556|       0| 0.3560107822556813|-0.11827295158156441|                  0|                  1|
|-0.25257503550326565|     0|-0.29301932477338205|       0|-0.9066212225780189|-0.11827295158156441|                  0|                  1|
|-0.18074007437194609|     0|-0.17960170113251442|       0|0.18383369068744945|-0.11827295158156441|                  1|                  0|
|-0.06287799216610096|     1|-0.11236509094288968|       0|-0.7918364948658644|-0.11827295158156441|                  0|                  0|
| -0.4199178426841806|     1|-0.29028621013689393|       0|  1.102111512384686|-0.11827295158156441|                  0|                  0|
+-------------------+------+-------------------+--------+-------------------+-------------------+-------------------+-------------------+----------
only showing top 5 rows
```

We grouped the data by the 'is_fraud' column and counted the occurrences of each value (fraud and non-fraud) using groupBy().count(). Then, we displayed the results with show().

```
# Group by the 'is_fraud' column and count occurrences of each value
df.groupBy('is_fraud').count().show()
```

```
[Stage 3:>
+--------+-------+
|is_fraud|  count|
+--------+-------+
|       1|   9651|
|       0|1842743|
+--------+-------+
```

Samsung Innovation Campus

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We imported a combination of libraries for both PySpark and traditional machine learning tasks using scikit-learn and XGBoost. The key components include VectorAssembler, which combines features into a single vector; LogisticRegression for binary classification in PySpark; and BinaryClassificationEvaluator to assess model performance in binary tasks. The Pipeline class streamlines the stages in a machine learning workflow, while SparkSession and col facilitate Spark operations and DataFrame column access. We also used train_test_split from scikit-learn to divide data into training and test sets, and xgboost for gradient boosting classification. Finally, classification_report provides a summary of precision, recall, F1-score, and accuracy for each class. This setup enables a hybrid approach, leveraging both PySpark and scikit-learn/XGBoost for effective machine learning tasks.

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline
# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import col  # Ensure this line is present
from sklearn.model_selection import train_test_split
import xgboost as xgb
import numpy as np
from sklearn.metrics import classification_report
```

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We performed undersampling to balance the dataset by first separating it into the majority class (non-fraud, is_fraud = 0) and the minority class (fraud, is_fraud = 1). Next, we reduced the majority class by randomly sampling it to match the size of the minority class. We then recombined the undersampled majority class with the minority class to create a balanced dataset. Finally, we verified the new class distribution using groupBy() and count() to ensure an equal representation of fraud and non-fraud cases.

```
# Step 1: Perform undersampling on the full dataset
# Separate majority and minority classes from the full data
majority_class = df.filter(col("is_fraud") == 0)
minority_class = df.filter(col("is_fraud") == 1)

# Perform undersampling by limiting the majority class to the size of the minority class
undersampled_majority = majority_class.sample(withReplacement=False, fraction=
(minority_class.count() / majority_class.count()))

# Combine undersampled majority class and minority class
balanced_df = undersampled_majority.union(minority_class)

# Check the new class distribution
balanced_df.groupBy('is_fraud').count().show()
```

```
[Stage 12:============================>                            (4 + 4) / 8]
+--------+-----+
|is_fraud|count|
+--------+-----+
|       0| 9579|
|       1| 9651|
+--------+-----+
```

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

In this step, we split the undersampled data into training and test sets using an 80-20 split. The training set (train_df) will contain 80% of the data, while the test set (test_df) will contain the remaining 20%. A seed value of 42 was set to ensure the split is reproducible.

```
# Step 2: Split the undersampled data into training and test sets
train_df, test_df = balanced_df.randomSplit([0.8, 0.2], seed=42)
```

We assembled the features into a single vector for both the training and test sets using VectorAssembler, which is essential for preparing data for machine learning models in PySpark. First, we selected all columns except the label (is_fraud) as input features. For the training data (train_df), we created a feature vector while retaining only the "features" and "is_fraud" columns. We applied the same process to the test data (test_df), assembling the features into a vector and keeping the label for model evaluation. This ensures the data is ready for machine learning algorithms that require feature vectors as input.

```
# Step 3: Assemble features into a single vector for training and test sets
feature_columns = [col for col in balanced_df.columns if col != 'is_fraud']

# VectorAssembler for training data
assembler_train = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_train_df = assembler_train.transform(train_df).select("features", "is_fraud")

# VectorAssembler for test data
assembler_test = VectorAssembler(inputCols=feature_columns, outputCol="features")
assembled_test_df = assembler_test.transform(test_df).select("features", "is_fraud")
```

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We converted the PySpark DataFrames (assembled_train_df and assembled_test_df) into Pandas DataFrames for use with XGBoost. This involved using the .toPandas() function to perform the conversion. We then separated the features and labels for the training set, extracting feature vectors from the "features" column into a NumPy array (X_train) and the labels (is_fraud) into y_train. Similarly, we processed the test set to obtain X_test and y_test. This preparation ensures the data is in the required format (NumPy arrays) for training and testing with the XGBoost model.

```
# Step 4: Convert to pandas DataFrames for use with XGBoost
train_pandas_df = assembled_train_df.toPandas()
test_pandas_df = assembled_test_df.toPandas()

# Separate features and labels for training and test sets
X_train = np.array(train_pandas_df['features'].tolist())
y_train = train_pandas_df['is_fraud'].values

X_test = np.array(test_pandas_df['features'].tolist())
y_test = test_pandas_df['is_fraud'].values
```

Samsung Innovation Campus

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

We prepared data and trained the XGBoost model by first creating DMatrix objects for the training (dtrain) and test (dtest) sets to facilitate efficient training. We then set parameters, including the objective as 'binary:logistic' for binary classification, the evaluation metric as 'logloss', a maximum depth of 6 to control tree complexity, and a learning rate (eta) of 0.3. Finally, we trained the model using xgb.train() with 100 boosting rounds to optimize performance.

```
# Step 5: Create DMatrix for XGBoost
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Set XGBoost parameters
params = {
    'objective': 'binary:logistic',
    'eval_metric': 'logloss',
    'max_depth': 6,
    'eta': 0.3
}

# Train the XGBoost model
xgboost_model = xgb.train(params, dtrain, num_boost_round=100)
```

We made predictions and evaluated model performance by using the trained XGBoost model to predict probabilities on the test set (dtest). We then converted these predicted probabilities to binary labels using a threshold of 0.5. Finally, we printed a classification report with classification_report(), summarizing precision, recall, F1-score, and accuracy for the predictions compared to the true labels (y_test).

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

```python
# Step 6: Make predictions on the test set
y_pred_prob = xgboost_model.predict(dtest)

# Convert predicted probabilities to binary labels (0.5 threshold)
y_pred_labels = [1 if val > 0.5 else 0 for val in y_pred_prob]

# Step 7: Evaluate the model performance
print(classification_report(y_test, y_pred_labels))
```

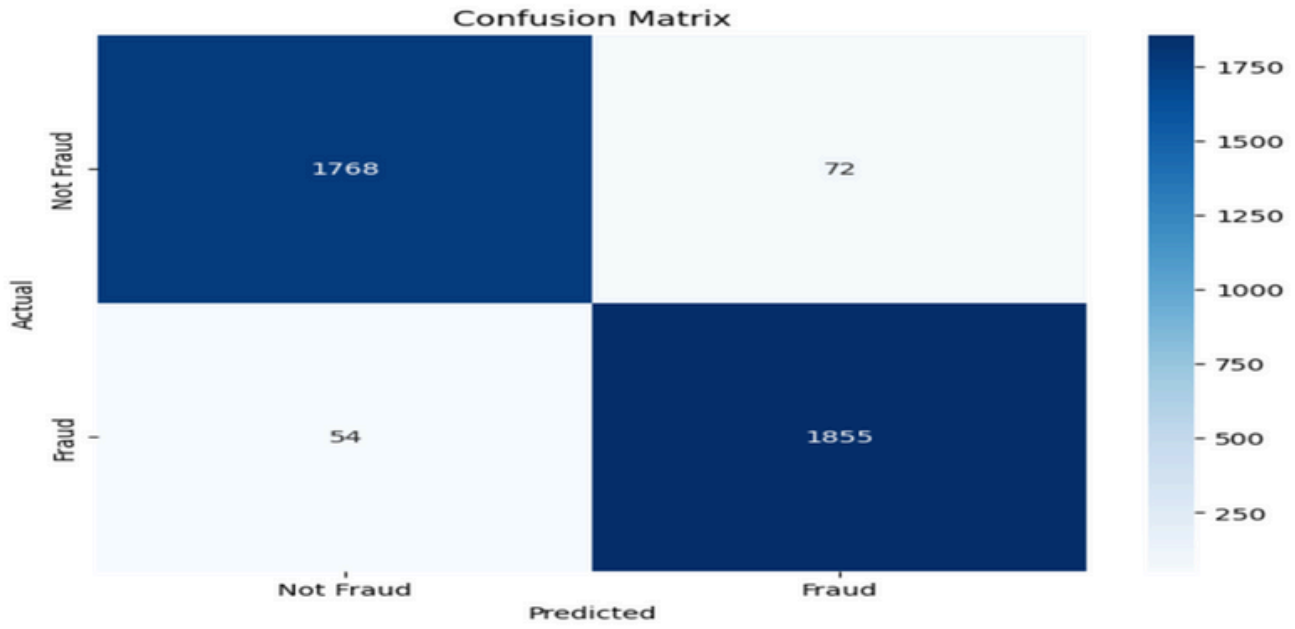
```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97      1840
           1       0.96      0.97      0.97      1909

    accuracy                           0.97      3749
   macro avg       0.97      0.97      0.97      3749
weighted avg       0.97      0.97      0.97      3749
```

We visualized model performance using a confusion matrix and an ROC curve. For the confusion matrix, we computed the matrix to evaluate prediction performance and visualized it with a heatmap, labeling the axes as 'Actual' and 'Predicted' for clarity. For the AUC and ROC curve, we calculated the AUC (Area Under the Curve) using roc_auc_score() and generated the ROC curve by plotting the false positive rate (FPR) against the true positive rate (TPR), including a diagonal line for reference to indicate random performance. These visualizations provide valuable insights into the model's classification accuracy and discriminative ability.

MACHINE LEARNING

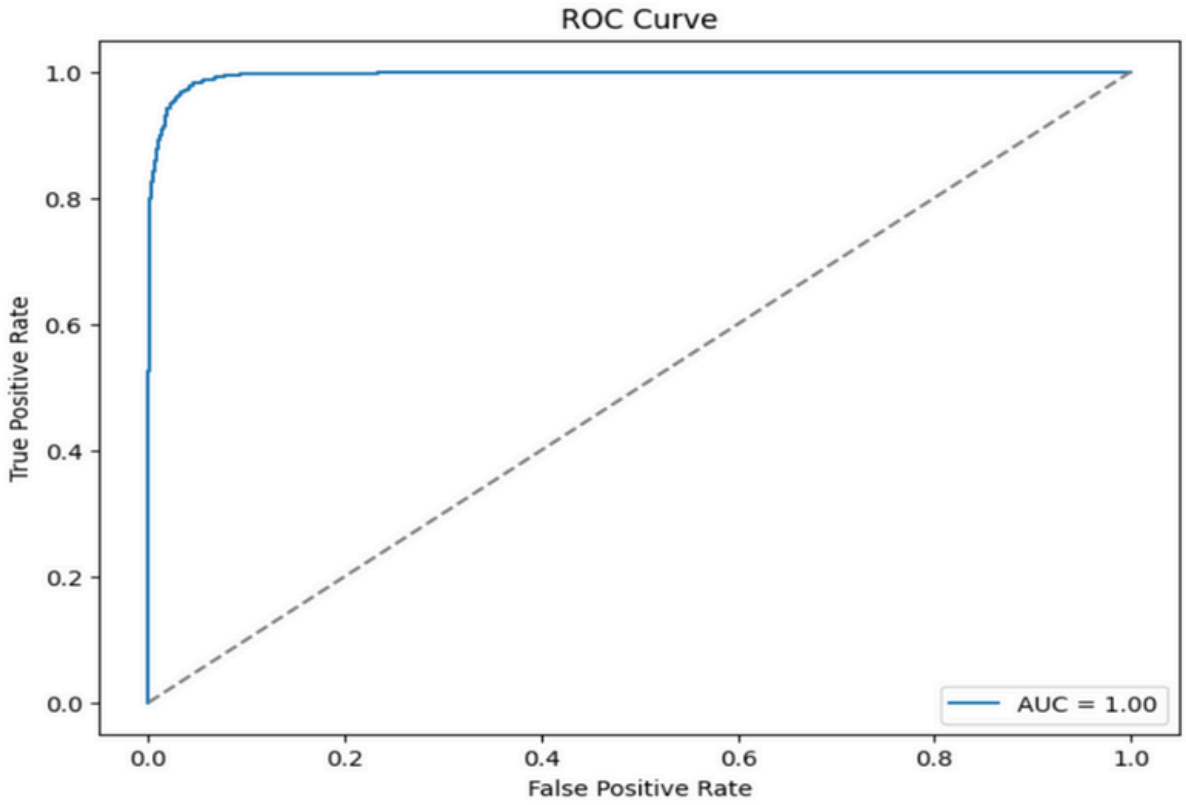
XGBOOST USING UNDER SAMPLING

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, roc_curve
import seaborn as sns

# Step 8: Confusion Matrix visualization
conf_matrix = confusion_matrix(y_test, y_pred_labels)

plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=['Not Fraud', 'Fraud'], yticklabels=['Not
Fraud', 'Fraud'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()

# Step 9: AUC and ROC Curve
auc = roc_auc_score(y_test, y_pred_prob)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```



MACHINE LEARNING

XGBOOST USING UNDER SAMPLING



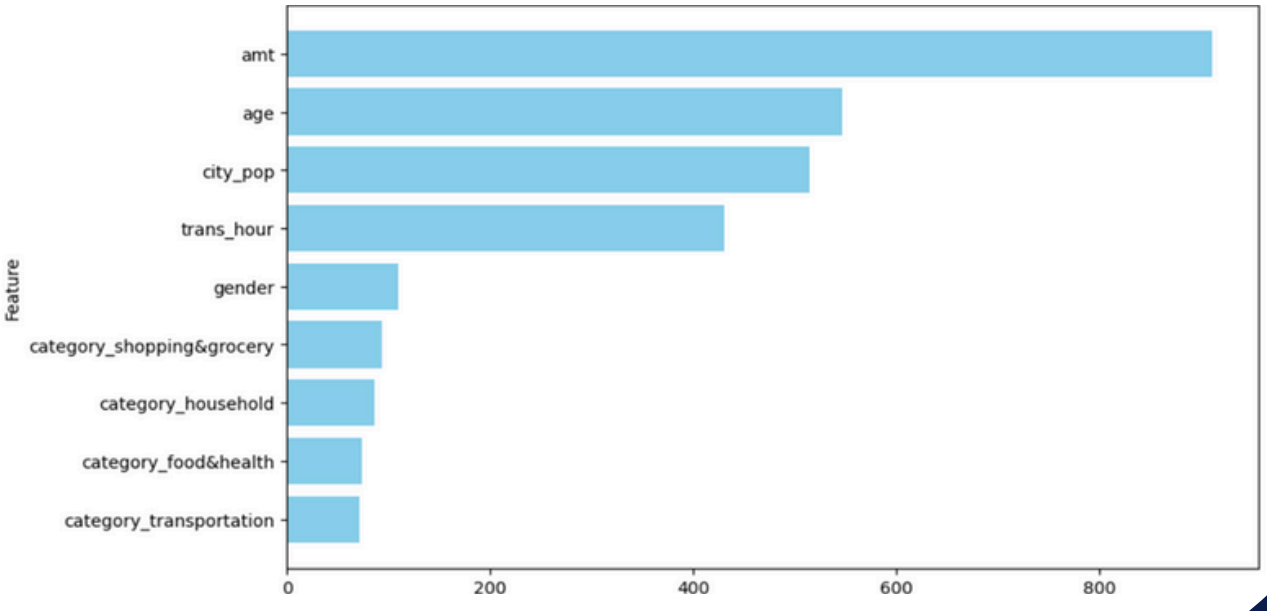
we extracted and visualized feature importance scores from the XGBoost model. First, we created a list of feature column names from the original DataFrame, excluding the target variable (is_fraud). Next, we retrieved feature importance scores using the get_score() method and sorted them into a list of tuples based on their importance. We selected the top 9 features and mapped their indices to their real names. Finally, we plotted these top features in a horizontal bar chart, labeling the axes and inverting the y-axis to display the most important feature at the top. This visualization highlights the features that significantly contribute to the model's predictions.

Samsung Innovation Campus

MACHINE LEARNING

XGBOOST USING UNDER SAMPLING

```python
# Step 1: Get feature names from the original DataFrame (df)
feature_columns = [col for col in df.columns if col != 'is_fraud']  # List of original feature column names

# Step 2: Get feature importance scores from the XGBoost model
feature_importance = xgboost_model.get_score(importance_type='weight')

# Step 3: Convert feature importance dictionary to a sorted list of tuples (feature_index, importance score)
sorted_importance = sorted(feature_importance.items(), key=lambda x: x[1], reverse=True)

# Step 4: Map feature indices (e.g., 'f0', 'f1') to real column names
top_9_features = sorted_importance[:9]  # Top 9 features

# Extract real feature names using the index
top_9_real_feature_names = [feature_columns[int(f[0][1:])] for f in top_9_features]  # Extract index from 'f0',
'f1', etc.
top_9_importance_values = [x[1] for x in top_9_features]

# Step 5: Plot the top 9 feature importances with real names
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plt.barh(top_9_real_feature_names, top_9_importance_values, color='skyblue')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.title('Top 9 Most Important Features')
plt.gca().invert_yaxis()  # To display the most important feature on top
plt.show()
plt.show()
```



Samsung Innovation Campus

MACHINE LEARNING
XGBOOST

This code evaluates the model's performance using two key metrics:

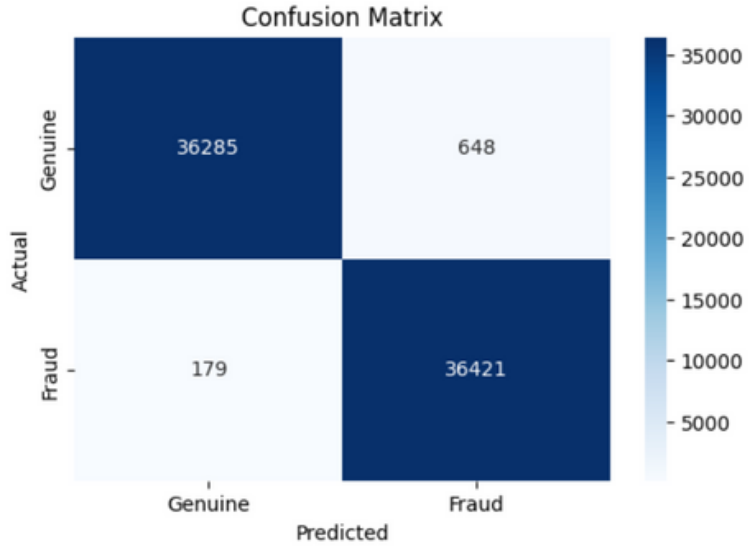
1. Confusion Matrix:

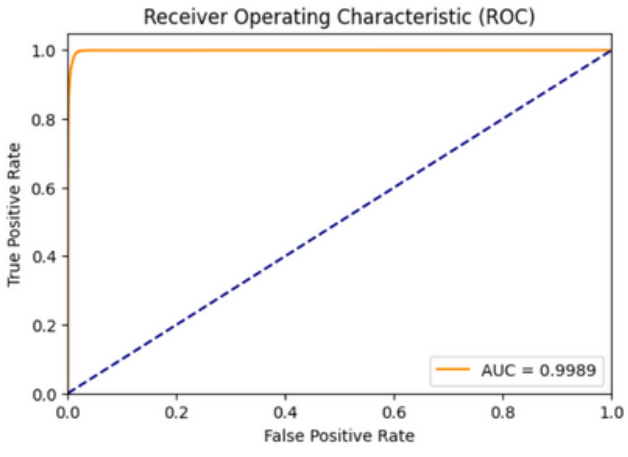
A confusion matrix is generated to visualize the classification performance, showing the counts of true positives, true negatives, false positives, and false negatives. The matrix is plotted using Seaborn's heatmap, with labels for "Genuine" and "Fraud" transactions.

2. ROC Curve and AUC:

The Receiver Operating Characteristic (ROC) curve is plotted to show the trade-off between the true positive rate (TPR) and false positive rate (FPR) across different classification thresholds. The Area Under the Curve (AUC) is also calculated and displayed on the graph to quantify the model's ability to distinguish between fraud and genuine transactions.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve
# Make predictions (using 0.5 threshold to convert probabilities to binary labels)
y_pred_labels = [1 if val > 0.5 else 0 for val in y_pred_prob]
# 1. Confusion Matrix
cm = confusion_matrix(y_test, y_pred_labels)
# Plot the confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Genuine", "Fraud"], yticklabels=
["Genuine", "Fraud"])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion Matrix')
plt.show()
# 2. ROC Curve and AUC
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
auc = roc_auc_score(y_test, y_pred_prob)
# Plot the ROC curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f'AUC = {auc:.4f}', color='darkorange')
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```





Samsung Innovation Campus

MACHINE LEARNING

COMPARISON

MODEL	RECALL
LOGISITC REGRESSION	0.75
RANDOM FOREST	0.88
SVM	0.78
XGBOOST	1

MACHINE LEARNING

COMPARISON

As we are primarily concerned with recall, we can see that the XGboost model has the best recall. As a result, we may use this mode to forecast fraudulent transactions.

XGboost is famous for its ability to work effectively with unbalanced datasets because of its sequential approach to learning and improving weak learners.

MACHINE LEARNING

RECOMMENDATIONS

In our analysis of predicting fraudulent transactions within an imbalanced dataset, we applied four machine learning algorithms: Random Forest, Support Vector Machines, Logistic Regression, and XGBoost. Among these, XGBoost demonstrated the highest recall, making it the most effective model for identifying fraudulent transactions. Consequently, we recommend utilizing XGBoost for future predictions to enhance the detection of fraudulent activities.

Additionally, feature importance analysis using the Random Forest algorithm revealed that the most critical features for prediction are the transaction amount and the transaction hour. These insights should be leveraged to further refine and optimize the predictive model.
Does this capture everything you wanted to convey?