



**INSTITUTE OF BUSINESS
ADMINISTRATION - IBA**

BIG DATA ANALYTICS

FINAL PROJECT REPORT

SUBMITTED BY:

NEHAL AHMED – 25751

DANIYAL AZHAR – 25793

ABSTRACT

This report presents a project focused on credit card transactional data ingestion, cleaning, exploratory data analysis (EDA), and subsequent storage in a Dockerized container using MongoDB. The project aims to provide a comprehensive understanding of credit card transactional patterns and facilitate insightful analytics through a dashboard.

The project begins with the ingestion of credit card fraud data, which is then cleaned and prepared for analysis using Python. EDA techniques are applied to gain insights into the data, identify trends, and uncover potential fraud patterns.

To ensure scalability, flexibility, and reproducibility, the cleaned data is stored in a Dockerized container utilizing MongoDB as the NoSQL database. Docker provides an isolated and portable environment, enabling easy deployment of the container on various systems.

The stored data serves as the foundation for the development of a metabase dashboard that provides interactive visualizations and analytics. The dashboard leverages the stored data to offer valuable insights and facilitates real-time monitoring of credit card fraud trends.

Overall, this project showcases the end-to-end process of ingesting, cleaning, performing EDA, storing data in a Dockerized container using MongoDB, and utilizing the data for dashboard-driven analytics. It highlights the importance of data quality, scalability, and efficient utilization of resources in delivering valuable insights for credit card fraud detection and prevention.

1. INTRODUCTION

1.1 DATA SET

The dataset utilized in this project is a simulated credit card transaction dataset that encompasses both legitimate and fraudulent transactions. It covers a duration from January 1, 2019, to December 31, 2020, providing a comprehensive and representative snapshot of credit card activities during this time.

The dataset includes transactions conducted by 1000 customers who utilized credit cards for their financial transactions. These customers were selected to represent a diverse range of individuals and their corresponding credit card usage patterns. The transactions were carried out with a pool of 800 different merchants, representing various industries and sectors.

Each transaction entry in the dataset contains several key attributes, including transaction amounts, timestamps, merchant information, customer details, and other relevant transaction-related features. These attributes provide valuable insights into the nature of each transaction and enable the analysis of various aspects such as transaction patterns, frequency, and characteristics.

The dataset includes both legitimate transactions, which represent typical and authorized credit card usage, as well as fraudulent transactions that involve unauthorized or deceptive activities. The inclusion of both types of transactions allows for a comprehensive analysis of credit card fraud patterns, identification of potential fraud indicators, and the development of effective fraud detection and prevention strategies.

By leveraging this simulated credit card transaction dataset, the project aimed to extract meaningful insights, identify patterns and trends, and gain a deeper understanding of credit card fraud dynamics. This understanding can support the development of advanced fraud detection algorithms, the implementation of proactive measures to mitigate risks, and the enhancement of overall security in credit card transactions.

1.2 PROJECT OBJECTIVE AND PROCESS

The project's objective was to analyze credit fraud data through exploratory data analysis (EDA) and data cleaning, with the goal of gaining valuable insights. The project followed a series of steps, including pulling images of Mongo DB and Metabase on Docker. The cleaned data was then dumped into Mongo DB and queries were run on metabase for insights. The next step involved connecting Mongo DB with Metabase to create a dashboard that would provide meaningful insights on the data.

The project first focused on performing EDA to understand the structure, patterns, and potential issues within the credit fraud data. This included identifying missing values, outliers, and anomalies, and ensuring the data was clean and ready for further analysis. To establish the necessary infrastructure, the project pulled the required images of Mongo DB and Metabase on Docker. This step ensured that the data could be stored

and managed effectively and provided a user-friendly interface for data exploration and visualization.

The cleaned credit fraud data was then imported into Mongo DB, a document-oriented database. This allowed for efficient storage, retrieval, and querying of the data, enabling seamless analysis and exploration.

To extract meaningful insights, various queries were executed on the data stored in Mongo DB. These queries were designed to address specific research questions and uncover patterns and trends related to credit fraud. The objective was to obtain actionable insights that would aid in understanding fraudulent activities and improving fraud detection mechanisms.

To facilitate data visualization and create an intuitive dashboard, Mongo DB was connected with the Metabase. This powerful data exploration and visualization tool enabled the creation of a visually appealing and informative dashboard. The dashboard presented key metrics, trends, and visual representations of the analyzed credit fraud data, empowering stakeholders to gain valuable insights and make data-driven decisions.

By achieving these objectives, the project aimed to enhance the understanding of credit fraud patterns, improve fraud detection capabilities, and provide stakeholders with actionable insights to mitigate risks and improve decision-making processes.

1.3 ADVANTAGES OF MONGODB AS THE NOSQL DATA STORAGE

The decision to use MongoDB as the NoSQL data storage tool for our credit card transaction dataset proved advantageous due to its flexible data model, horizontal scalability, and high performance. MongoDB's flexible schema allowed for the seamless integration of new data attributes and accommodated the evolving nature of the dataset without requiring costly schema alterations. The horizontal scalability of MongoDB enabled us to handle the increasing volume and velocity of the dataset by leveraging automatic sharding and distributing data across multiple servers. This ensured optimal performance for real-time transaction processing and facilitated timely fraud detection. Additionally, MongoDB's high performance and indexing mechanisms facilitated efficient query execution, enabling quick and accurate analysis of complex data structures within the dataset.

Furthermore, MongoDB's replication capabilities ensured data availability and reliability. The automatic creation of replica sets across different servers provided data durability and fault tolerance, minimizing the risk of data loss and downtime. Additionally, MongoDB's integration with data analysis tools, such as Metabase, allows for the creation of intuitive dashboards and visualizations, enabling stakeholders to gain valuable insights into correlations between variables such as age and fraud amount, location and amount, and gender and fraud. These insights supported data-driven decision-making, enhanced fraud detection capabilities, and improved overall security and customer protection in credit card transactions.

By leveraging MongoDB as the NoSQL data storage tool for our credit card transaction dataset, we were able to efficiently manage and analyze the data, leading to improved

fraud detection, enhanced decision-making processes, and better security measures. The flexibility, scalability, and performance advantages of MongoDB, combined with its integration capabilities, provided a robust foundation for extracting meaningful insights and facilitating data-driven strategies. Ultimately, MongoDB proved to be a valuable asset in handling credit card transaction data and empowering organizations to make informed decisions based on comprehensive and reliable information.

1.4 LEVERAGING META BASE FOR SEAMLESS INTEGRATION AND INTUITIVE DATA VISUALIZATION

The decision to use Meta base alongside MongoDB for our project was driven by its unique features and suitability for our specific requirements. While other analytics tools like Power BI offer powerful data visualization capabilities, Meta base stood out for its seamless integration with MongoDB and its user-friendly interface.

Meta base provides a direct connection to MongoDB, allowing for real-time data retrieval and analysis. This integration eliminated the need for complex data extraction and transformation processes, streamlining our workflow and saving valuable time. Additionally, Metabase's ability to dynamically query MongoDB collections and generate interactive dashboards made it an ideal choice for exploring and visualizing the credit card transaction data.

In summary, Metabase was chosen for our project alongside MongoDB due to its seamless integration, real-time data retrieval capabilities, and user-friendly interface. The combination of Metabase and MongoDB provided an efficient and accessible solution for exploring, analyzing, and visualizing credit card transaction data, enabling our team to gain valuable insights and make data-driven decisions.

1.5 DOCKER-ENABLED INTEGRATION OF MongoDB AND META BASE

The utilization of Docker in our project played a crucial role in the efficient management and deployment of the MongoDB and Metabase components. Docker provided a containerization platform that allowed us to encapsulate the required dependencies, configurations, and software packages into isolated containers.

By pulling the images of MongoDB and Metabase on Docker, we ensured that the necessary infrastructure was readily available, reducing the complexity of setting up and configuring these tools. Docker's containerization technology enabled us to create lightweight, portable, and reproducible environments, eliminating any potential compatibility issues or dependencies conflicts.

Dumping the cleaned credit card transaction data into MongoDB within a Docker container offered several benefits. Firstly, it provided a consistent and isolated environment, ensuring data integrity and avoiding interference from other processes or applications. Secondly, Docker facilitated the seamless replication and scaling of MongoDB instances, enabling us to handle large volumes of data efficiently and effectively.

Connecting Metabase with MongoDB through Docker allowed us to create a dashboard that fetched data directly from the MongoDB container. Docker's networking capabilities ensured smooth communication between the containers, enabling Metabase to access and analyze the credit card transaction data stored in MongoDB. This integration streamlined the data visualization process, allowing us to create interactive dashboards that provided valuable insights into various correlations within the dataset.

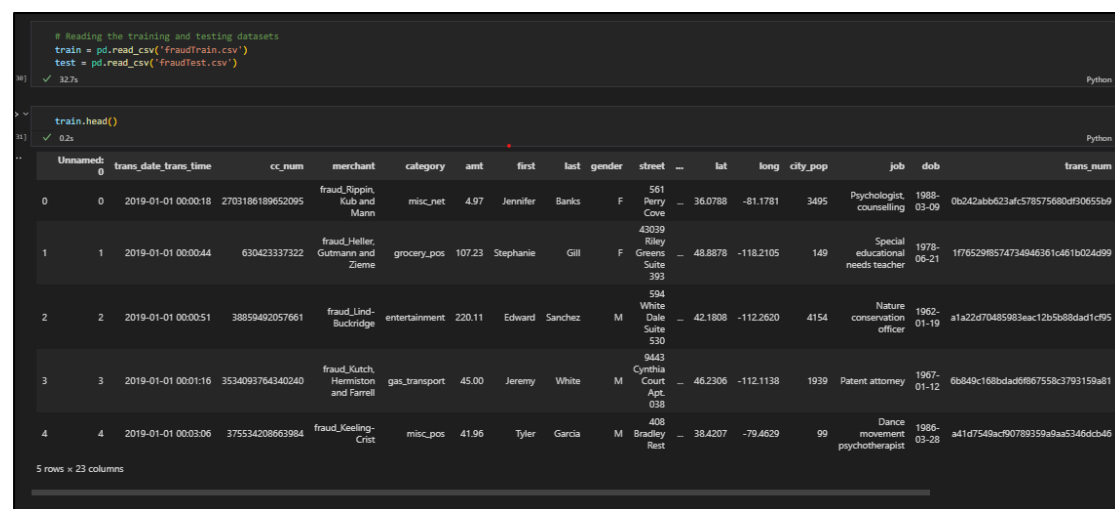
Overall, the utilization of Docker in our project simplified the setup, management, and integration of MongoDB and Metabase. It provided a consistent and reproducible environment, enhanced scalability, and facilitated the seamless connection between these tools, enabling us to efficiently analyze the credit card transaction data and create insightful dashboards. Docker's containerization technology played a vital role in ensuring a smooth and streamlined workflow for our data analysis and visualization processes.

2. Credit Card Fraud Analysis

2.1 EDA

Exploratory Data Analysis (EDA) is a crucial step in understanding and gaining insights from our dataset. In this project, we performed EDA on credit card fraud data to uncover patterns, anomalies, and trends. The following steps were taken to conduct a comprehensive EDA:

1. The first step involved loading the test and train datasets. We carefully analyzed the columns in both datasets to review the metrics and gain an initial understanding of the data's structure.



```
# Reading the training and testing datasets
train = pd.read_csv('fraudtrain.csv')
test = pd.read_csv('fraudtest.csv')
```

train.head()

Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	lat	long	city_pop	job	dob	trans_num
0	2019-01-01 00:00:18	2703186189652095	fraud_Rippin Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	36.0788	-81.1781	3495	Psychologist, counseling	1988- 03-09	0b242abb623afc578575580df930655b9
1	2019-01-01 00:00:44	630423337322	fraud_Heller Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greene Suite 393	48.8878	-118.2105	149	Special educational needs teacher	1978- 06-21	1776529f8574734946361c461b024d99
2	2019-01-01 00:00:51	38859492057661	fraud_Lind- Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Suite 530	42.1808	-112.2620	4154	Nature conservation officer	1962- 01-19	a1a22d70485983eac12b5b8bdad1cf95
3	2019-01-01 00:01:16	3534093764340240	fraud_Kutch Hemiston and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	46.2306	-112.1138	1939	Patent attorney	1967- 01-12	6b849c168bdad6f667558c3793159a81
4	2019-01-01 00:03:06	375534208663984	fraud_Keeling- Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	38.4207	-79.4629	99	Dance movement psychotherapist	1986- 03-28	a41d7549ac190789359a9aa5346dcdb46

5 rows x 23 columns

```
test.head()
```

Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	lat	long	city_pop	job	dob	trans_num
0	2020-06-21 12:14:25	2291163933867244	fraud_Kirlin and Sons	personal_care	2.86	Jeff	Elliott	M	351 Darlene Green	—	33.9659	-80.9355	333497	Mechanical engineer	1968-03-19	2da90c7d74bd46a0ca3777415b3ebd3
1	2020-06-21 12:14:33	3573030041201292	fraud_Sporen-Kesbler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union	—	40.3207	-110.4360	302	Sales professional, IT	1990-01-17	324c204407e99f51b0d6ca0055005e7
2	2020-06-21 12:14:53	3598215285024754	fraud_Swaniawski, Nitzsche and Welch	health_fitness	41.28	Ashley	Lopez	F	9333 Valentine Point	—	40.6729	-73.5365	34496	Librarian, public	1970-10-21	c81755dbbbea9d5c77094348a7579be
3	2020-06-21 12:15:15	3591919803438423	fraud_Haley Group	misc_pos	60.05	Brian	Williams	M	32941 Krystal Mill Apt. 552	—	28.5697	-80.8191	54767	Set designer	1987-07-25	2159175b9efe66dc301f149d3d5ab8bc
4	2020-06-21 12:15:17	3528826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	M	5783 Evan Roads Apt. 465	—	44.2529	-85.0170	1126	Furniture designer	1955-07-06	57f021bd3f3288738bb535c302a31b

5 rows x 23 columns

```
train.columns
```

```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category', 'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip', 'lat', 'long', 'city_pop', 'job', 'trans_num', 'unix_time', 'merch_lat', 'merch_long', 'is_fraud'], dtype=object)
```

- During the analysis, we identified an unnecessary column labeled "Unnamed" in the train dataset and a redundant column in the test dataset. These columns were dropped as they provided no meaningful information for our analysis.

```
train=train.drop('Unnamed: 0',axis=1)
```

```
test=test.drop('Unnamed: 0',axis=1)
```

- To consolidate the data, we concatenated the test and train datasets, creating a unified dataset for further analysis. This step allowed us to have a larger dataset, enhancing the statistical significance of our findings.

```
# Combining the train and test datasets for data cleaning and data visualization
data = pd.concat([train, test], axis = 0)
data.head()
```

trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	city	lat	long	city_pop	job	dob	trans_num
2019-01-01 00:00:18	2703186189652095	fraud_Riggins, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	Moravian Falls	-36.0788	-81.1781	3495	Psychologist, counseling	1988-03-09	0b242abb623afc578575680d9f30655b9
2019-01-01 00:00:44	630423337322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	Orient	-48.8878	-118.2105	149	Special educational needs teacher	1978-06-21	1f76529f8574734946361c461b03d4999
2019-01-01 00:00:51	38859492057661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale Suite 530	Malad City	-42.1808	-112.2620	4154	Nature conservation officer	1962-01-19	a1a22d70485983eac12b5b88dad1cf95
2019-01-01 00:01:16	3534093764340240	fraud_Kutch, Hammon and Farrell	gas_transport	45.00	Jeremy	White	M	9443 Cynthia Court Apt. 038	Boulder	-46.2306	-112.1138	1939	Patent attorney	1967-01-12	6b849c168bdad6f67558c3793159a81
2019-01-01 00:03:06	375534208663984	fraud_Keeling-Cret	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Rest	Doe Hill	-38.4207	-79.4629	99	Dance movement psychotherapist	1986-03-28	a41d7549ac90789359a3aa3346dcb46

5 rows x 22 columns

- The column types were reviewed to ensure appropriate data handling. Understanding the data types of each column is vital for performing accurate calculations and appropriate visualizations during the EDA process.

```

data.info()
[17] ✓ 0.1s Python
...
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1852394 entries, 0 to 555718
Data columns (total 22 columns):
 #   Column        Dtype
---  -
 0   trans_date_trans_time  object
 1   cc_num         int64
 2   merchant       object
 3   category       object
 4   amt           float64
 5   first         object
 6   last          object
 7   gender         object
 8   street         object
 9   city          object
10   state         object
11   zip           int64
12   lat           float64
13   long          float64
14   city_pop      int64
15   job           object
16   dob           object
17   trans_num     object
18   unix_time     int64
19   merch_lat     float64
20   merch_long    float64
21   is_fraud      int64
dtypes: float64(5), int64(5), object(12)
memory usage: 325.1+ MB

```

5. Further analysis was carried out on the dataset to explore various aspects, such as distribution, duplication, and missing values. This analysis enabled us to gain deeper insights into the dataset and uncover any patterns or anomalies that might be present.

```

# Checking for duplicate values
data.duplicated().sum()
[18] ✓ 19.7s Python
...
0

# Checking for null values
data.isnull().sum()
[19] ✓ 5.0s Python
...
trans_date_trans_time  0
cc_num                0
merchant              0
category              0
amt                  0
first                 0
last                  0
gender                0
street                0
city                  0
state                 0
zip                   0
lat                   0
long                  0
city_pop              0
job                   0
dob                   0
trans_num             0
unix_time             0
merch_lat             0
merch_long            0
is_fraud              0
dtype: int64

```

```

data.describe()
[20] ✓ 1.0s Python
...

```

	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06	1.852394e+06
mean	4.173860e+17	7.006357e+01	4.881326e+04	3.853931e+01	-9.022783e+01	8.864367e+04	1.358674e+09	3.853898e+01	-9.022794e+01	5.210015e-03
std	1.309115e+18	1.592540e+02	2.688185e+04	5.071470e+00	1.374789e+01	3.014878e+05	1.819508e+07	5.105604e+00	1.375969e+01	7.199217e-02
min	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902742e+01	-1.666716e+02	0.000000e+00
25%	1.800429e+14	9.640000e+00	2.623700e+03	3.466890e+01	-9.679800e+01	7.410000e+02	1.343017e+09	3.474012e+01	-9.689944e+01	0.000000e+00
50%	3.521417e+15	4.745000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.443000e+03	1.357089e+09	3.936890e+01	-8.744069e+01	0.000000e+00
75%	4.642255e+15	8.310000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.374581e+09	4.195626e+01	-8.024511e+01	0.000000e+00
max	4.992346e+18	2.894890e+04	9.992100e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.388534e+09	6.751027e+01	-6.695090e+01	1.000000e+00

6. As part of the data preparation process, we converted the date and time columns into a standardized date/time format. This conversion enabled us to utilize temporal information for extracting meaningful insights and patterns from the data.

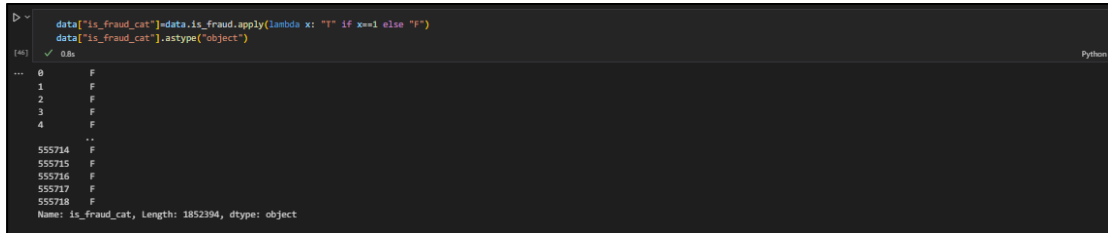
```

data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data['trans_date'] = data['trans_date_trans_time'].dt.strftime('%Y-%m-%d')
data['trans_date'] = pd.to_datetime(data['trans_date'])
data['dob'] = pd.to_datetime(data['dob'])
[21] ✓ 48.1s Python

```

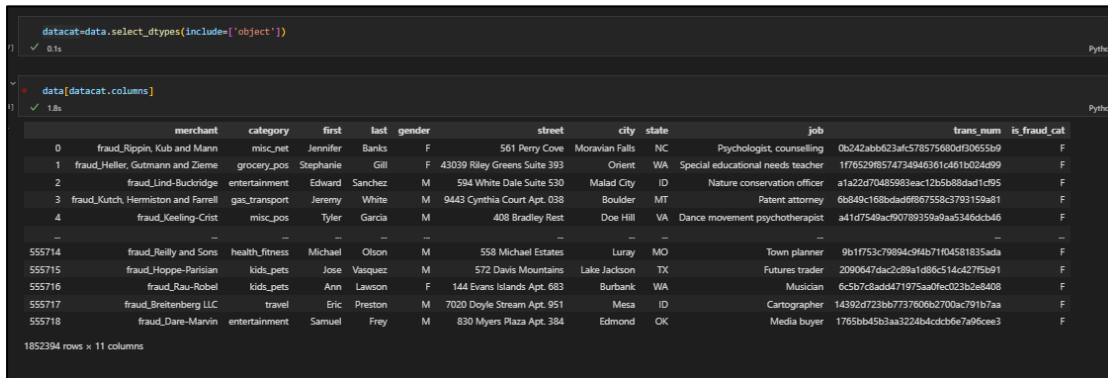

7. In the "Is Fraud" column, we replaced the values of 0 with "F" (indicating false) and 1 with "T" (indicating true). This transformation enhanced the interpretability of the data and made it easier to understand the fraudulent transactions.

```
data["is_fraud_cat"] = data.is_fraud.apply(lambda x: "T" if x==1 else "F")
data["is_fraud_cat"].astype("object")
```



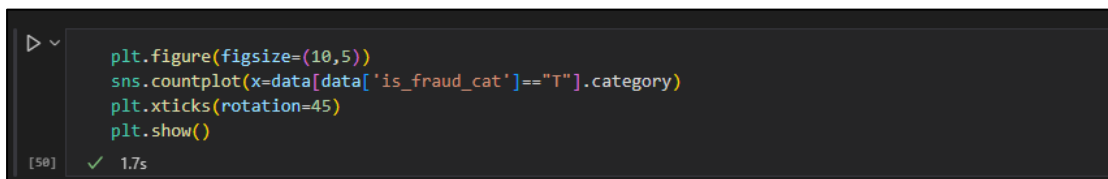
8. Categorical columns were analyzed to gain a better understanding of the distribution and frequency of each category. This analysis provided insights into the composition of the dataset and helped identify potential patterns or relationships.

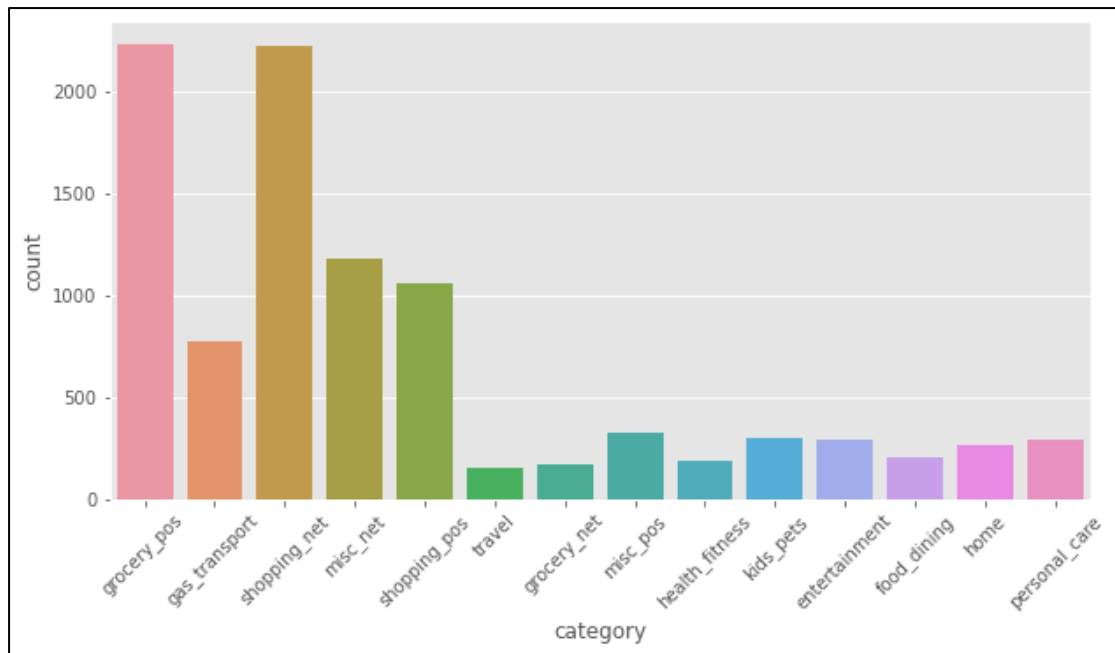
```
datacat = data.select_dtypes(include=['object'])
data[datacat.columns]
```



9. Multiple graphs and visualizations were plotted to facilitate a visual exploration of the data. These graphs allowed us to uncover patterns, identify outliers, and visualize relationships among different variables.

```
plt.figure(figsize=(10,5))
sns.countplot(x=data[data['is_fraud_cat']=="T"].category)
plt.xticks(rotation=45)
plt.show()
```





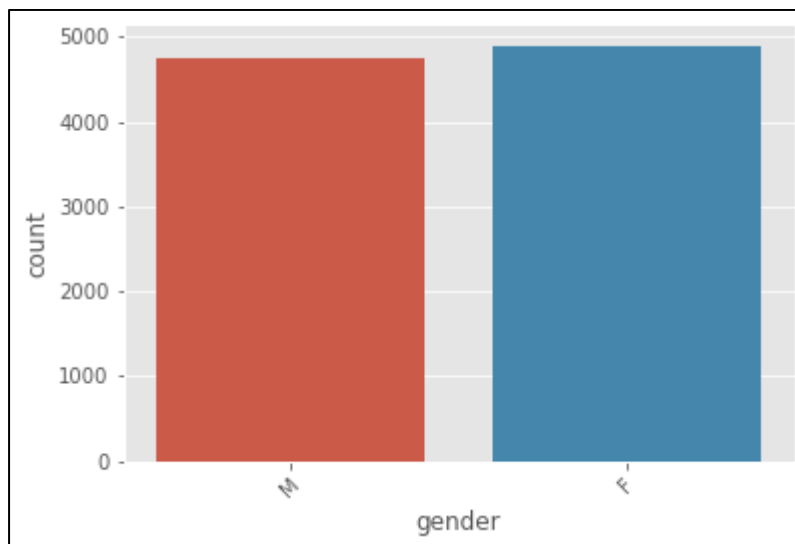
Grocery POS experienced the highest Fraud Count.

```

sns.countplot(x=data[data['is_fraud_cat']=="T"].gender)
plt.xticks(rotation=45)
plt.show()

```

[51] ✓ 0.6s



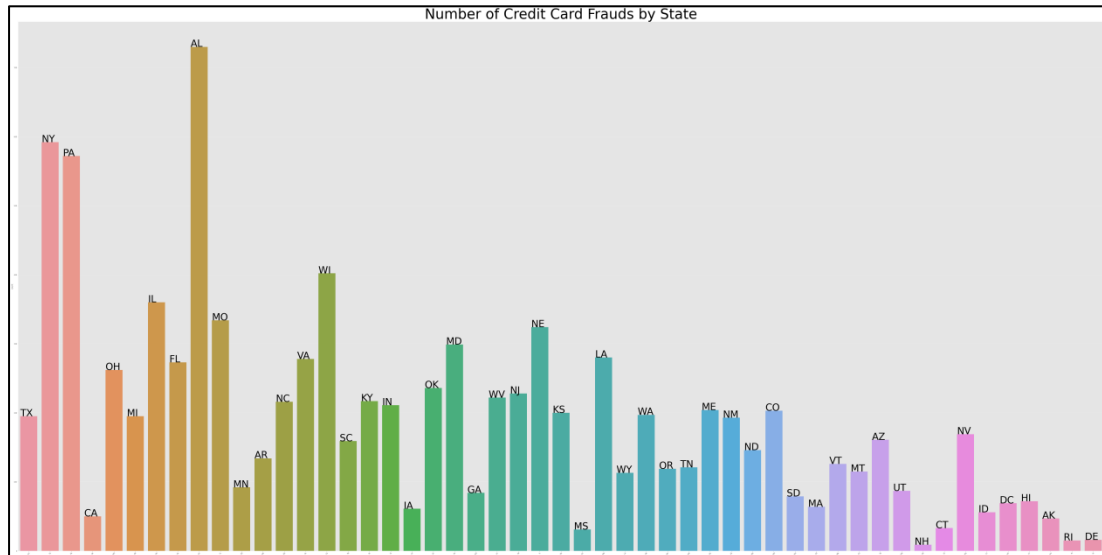
Although the number of females is greater, the male and female genders are nearly balanced in terms of quantity.

```
fig, ax = plt.subplots(figsize=(120,60))
plt.rcParams.update({'font.size': 60})
sns.countplot(x=data[data['is_fraud_cat']=="T"].state)
plt.xticks(rotation=45)
for p, label in zip(ax.patches, data["state"].value_counts().index):
    ax.annotate(label, (p.get_x(), p.get_height()+0.15))
plt.title("Number of Credit Card Frauds by State")
plt.show()
```

[52] ✓ 6.5s

Activate Windows
Go to Settings to activate Windows.

Python



States OH, TX, and LA report the greatest number of credit card frauds.

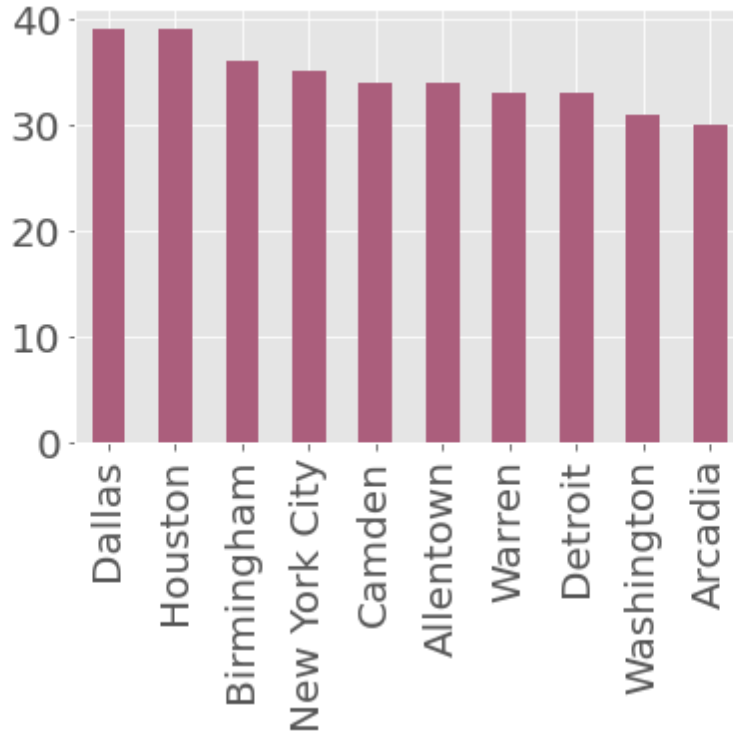
```
def randomcolor():  
    r = random.random()  
    b = random.random()  
    g = random.random()  
    rgb = [r,g,b]  
    return rgb  
  
plt.rcParams.update({'font.size': 20})  
data[data['is_fraud_cat']=="T"]['city'].value_counts(sort=True,ascending=False).head(10).plot(kind="bar",color=randomcolor())  
plt.title("Number of Credit Card Frauds by City")  
plt.show()
```

[57]

✓ 0.6s

Python

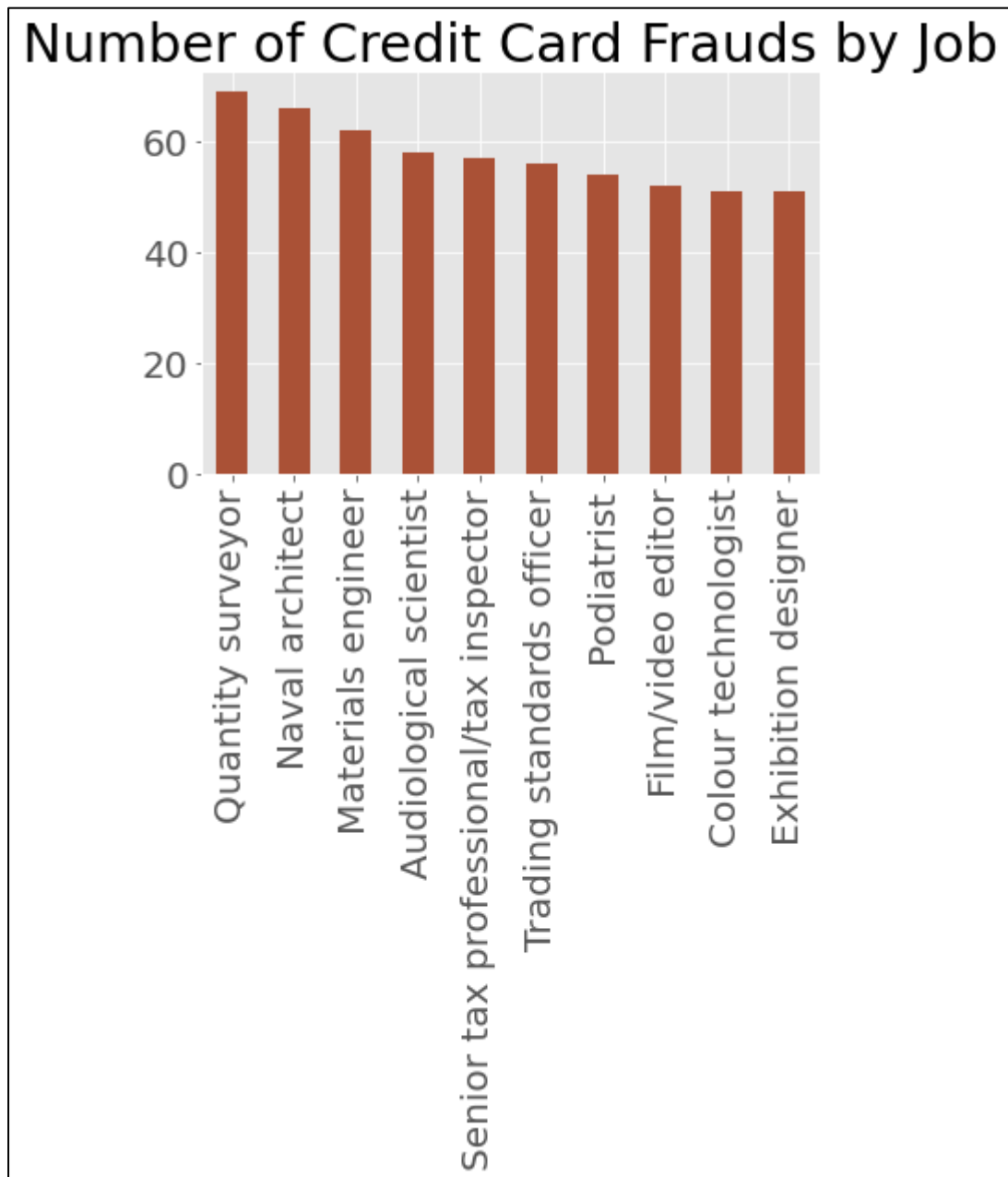
Number of Credit Card Frauds by City



Dallas, Houston, and Birmingham report the most frauds city-wise.

```
data[data['is_fraud_cat']=='T']['job'].value_counts(sort=True,ascending=False).head(10).plot(kind="bar",color=randomcolor())
plt.title("Number of Credit Card Frauds by Job")
plt.show()
```

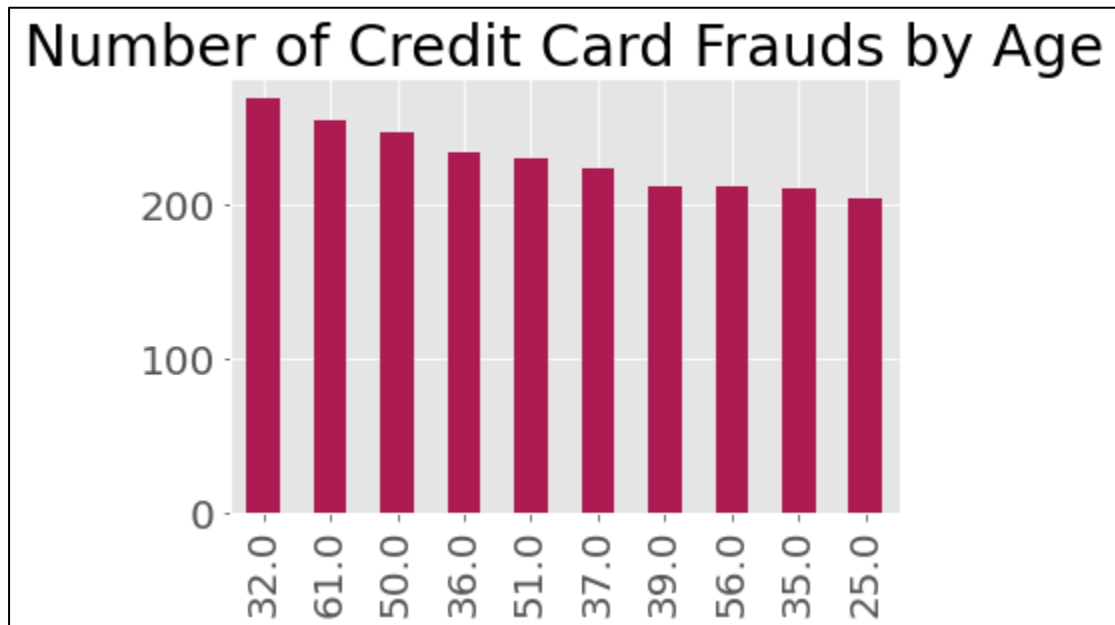
[59] ✓ 1.1s Python



Most frauds occurred in jobs of quantity surveyor followed by naval architect and materials engineer.

```
[63] ✓ 0.3s Python
current_date = datetime.now()
data['Age'] = (current_date - data['dob']).astype('<m8[Y]')

[64] ✓ 1.5s Python
data[data['is_fraud_cat'] == "T"]['Age'].value_counts(sort=True, ascending=False).head(10).plot(kind="bar", color=randomcolor())
plt.title("Number of Credit Card Frauds by Age")
plt.show()
```



By following these steps, we conducted a thorough EDA on the credit card fraud dataset, enabling us to gain valuable insights, identify potential fraud patterns, and inform subsequent analysis and decision-making processes.

2.2 STREAMLINING MongoDB AND METABASE INTEGRATION WITH DOCKER

The images of MongoDB and Metabase were pre-pulled, simplifying the setup process. To initiate the MongoDB side, we only needed to run the MongoDB image and create a container for it. Once the container was up and running, we proceeded to dump the cleaned credit card transaction data into the MongoDB container. This step ensured that our data was securely stored and readily available for analysis.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
apache/superset	latest	28437f75265f	23 hours ago	1.07GB
ajeje93/grafana-mongodb	latest	45474355b5da	3 days ago	1.02GB
metabase/metabase	latest	537adb7b89bb	10 days ago	461MB
grafana/grafana	latest	6c5313ea00cc	3 weeks ago	309MB
apache/hadoop	3	03ceea563502	2 months ago	1.64GB
postgres	latest	f462f91720c0	3 months ago	379MB
mongo	latest	a440572ac3c1	4 months ago	639MB
docker/getting-started	latest	3e4394f6b72f	5 months ago	47MB
alpine/git	latest	22d84a66cda4	6 months ago	43.6MB
bde2020/spark-worker	3.3.0-hadoop3.3	50eb31df8fa4	11 months ago	544MB
bde2020/spark-master	3.3.0-hadoop3.3	925cc6e56b20	11 months ago	544MB
bde2020/hadoop-nodemanager	2.0.0-hadoop3.2.1-java8	4e47dabd148f	3 years ago	1.37GB
bde2020/hadoop-resourcemanager	2.0.0-hadoop3.2.1-java8	3deba4a1885f	3 years ago	1.37GB
bde2020/hadoop-namenode	2.0.0-hadoop3.2.1-java8	839ec11d95f8	3 years ago	1.37GB
bde2020/hadoop-historyserver	2.0.0-hadoop3.2.1-java8	173c52d1f624	3 years ago	1.37GB
bde2020/hadoop-datanode	2.0.0-hadoop3.2.1-java8	df288ee0a7f9	3 years ago	1.37GB
redash/redash	latest	1b48a51810b5	3 years ago	1.31GB
prakhar1989/static-site	latest	f01030e1dcf3	7 years ago	134MB

```
docker run -it --rm --network host -v C:/Users/nehaa/Downloads/BDA_Project:/data mongo
mongoimport --host localhost --port 27017 --db test --collection mycollection --type csv --headerline
--file /data/data.csv
docker run -d -p 3000:3000 --name metabase --link mongodb metabase/metabase
```

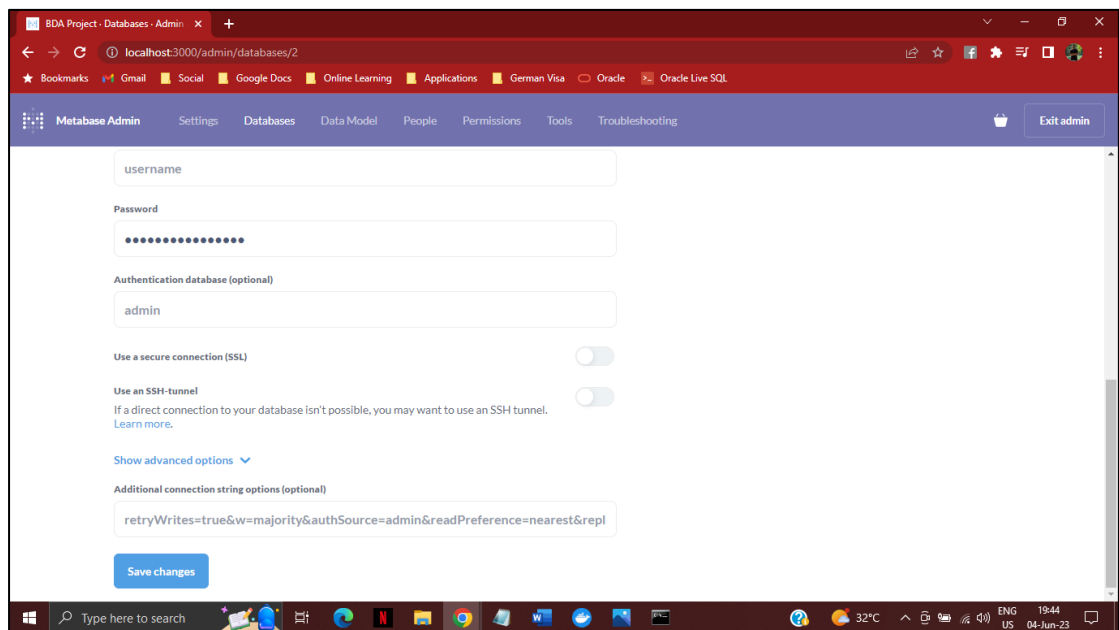
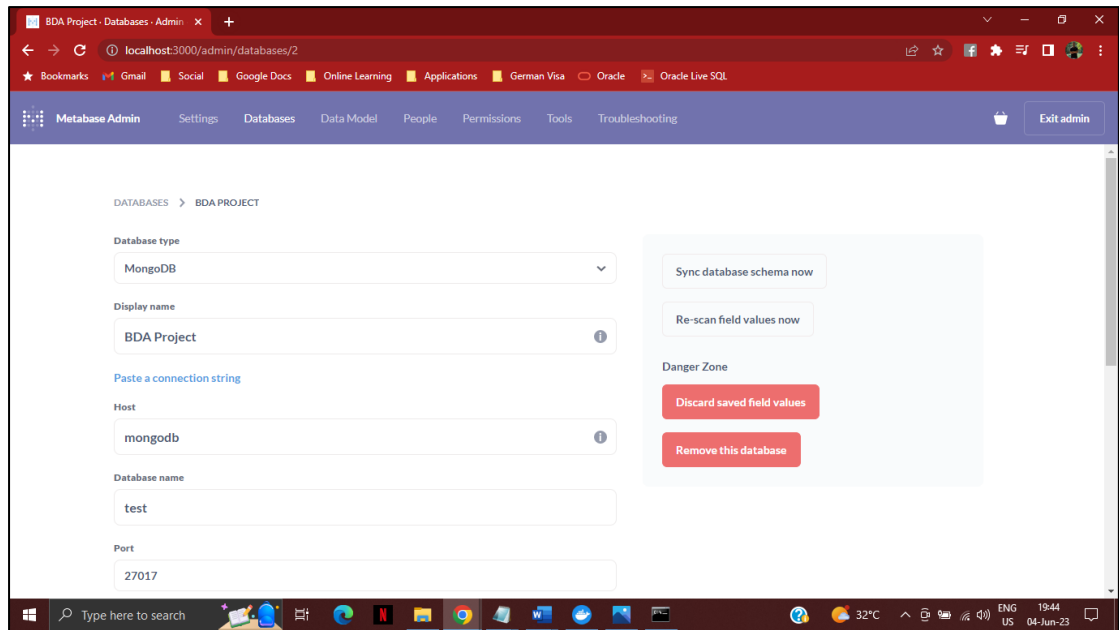
```

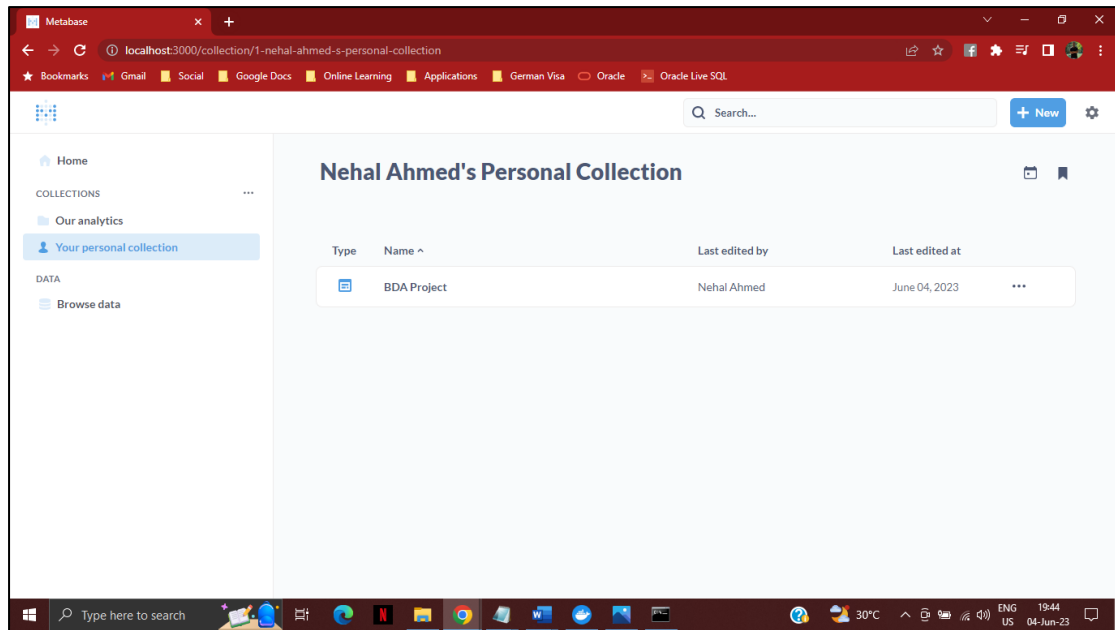
Command Prompt
C:\Users\nehaa>docker run -it --rm --network host -v C:/Users/nehaa/Downloads/BDA_Project:/data mongo mongoimport --host localhost --port 27017 --db test --collection mycollection --type csv --headerline --file /data/data.csv
connected to: mongodb://localhost:27017/
2023-06-04T14:02:39.816+0000 [.....] test.mycollection 13.4MB/461MB (2.9%)
2023-06-04T14:02:42.816+0000 [#.....] test.mycollection 27.6MB/461MB (6.0%)
2023-06-04T14:02:45.817+0000 [##.....] test.mycollection 41.6MB/461MB (9.0%)
2023-06-04T14:02:48.816+0000 [###.....] test.mycollection 55.3MB/461MB (12.0%)
2023-06-04T14:02:51.817+0000 [####.....] test.mycollection 69.7MB/461MB (15.1%)
2023-06-04T14:02:54.816+0000 [#####.....] test.mycollection 83.7MB/461MB (18.1%)
2023-06-04T14:02:57.816+0000 [#####.....] test.mycollection 94.2MB/461MB (20.4%)
2023-06-04T14:03:00.817+0000 [#####.....] test.mycollection 106MB/461MB (22.9%)
2023-06-04T14:03:03.817+0000 [#####.....] test.mycollection 117MB/461MB (25.4%)
2023-06-04T14:03:06.816+0000 [#####.....] test.mycollection 128MB/461MB (27.8%)
2023-06-04T14:03:09.816+0000 [#####.....] test.mycollection 140MB/461MB (30.7%)
2023-06-04T14:03:12.816+0000 [#####.....] test.mycollection 153MB/461MB (33.1%)
2023-06-04T14:03:15.816+0000 [#####.....] test.mycollection 167MB/461MB (36.2%)
2023-06-04T14:03:18.816+0000 [#####.....] test.mycollection 179MB/461MB (38.8%)
2023-06-04T14:03:21.816+0000 [#####.....] test.mycollection 193MB/461MB (41.8%)
2023-06-04T14:03:24.816+0000 [#####.....] test.mycollection 208MB/461MB (44.7%)
2023-06-04T14:03:27.817+0000 [#####.....] test.mycollection 220MB/461MB (47.6%)
2023-06-04T14:03:30.816+0000 [#####.....] test.mycollection 233MB/461MB (50.5%)
2023-06-04T14:03:33.817+0000 [#####.....] test.mycollection 247MB/461MB (53.5%)
2023-06-04T14:03:36.816+0000 [#####.....] test.mycollection 261MB/461MB (56.6%)
2023-06-04T14:03:39.817+0000 [#####.....] test.mycollection 275MB/461MB (59.7%)
2023-06-04T14:03:42.816+0000 [#####.....] test.mycollection 289MB/461MB (62.5%)
2023-06-04T14:03:45.816+0000 [#####.....] test.mycollection 299MB/461MB (64.9%)
2023-06-04T14:03:48.816+0000 [#####.....] test.mycollection 313MB/461MB (67.8%)
2023-06-04T14:03:51.816+0000 [#####.....] test.mycollection 327MB/461MB (70.9%)
2023-06-04T14:03:54.816+0000 [#####.....] test.mycollection 342MB/461MB (74.1%)
2023-06-04T14:03:57.816+0000 [#####.....] test.mycollection 356MB/461MB (77.2%)
2023-06-04T14:04:00.816+0000 [#####.....] test.mycollection 370MB/461MB (80.3%)
2023-06-04T14:04:03.816+0000 [#####.....] test.mycollection 385MB/461MB (83.4%)
2023-06-04T14:04:06.816+0000 [#####.....] test.mycollection 399MB/461MB (86.5%)
2023-06-04T14:04:09.816+0000 [#####.....] test.mycollection 413MB/461MB (89.5%)
2023-06-04T14:04:12.816+0000 [#####.....] test.mycollection 428MB/461MB (92.7%)
2023-06-04T14:04:15.816+0000 [#####.....] test.mycollection 439MB/461MB (95.1%)
2023-06-04T14:04:18.816+0000 [#####.....] test.mycollection 453MB/461MB (98.2%)
2023-06-04T14:04:21.816+0000 [#####.....] test.mycollection 461MB/461MB (100.0%)
2023-06-04T14:04:23.606+0000 [#####.....] 1852394 document(s) imported successfully. 0 document(s) failed to import.
2023-06-04T14:04:23.607+0000

C:\Users\nehaa>docker run -d -p 3000:3000 --name metabase --link mongodb metabase/metabase
8798c78918952c56307a500b6b6a2f0c91eb8e4015b620917007f6b05246e3c

```

Next, we ran the Metabase container, which was seamlessly integrated with MongoDB through Docker networking. This allowed us to establish a connection between the Metabase and the MongoDB container, enabling Meta base to fetch data directly from the database. By leveraging the dynamic query capabilities of Metabase, we were able to generate interactive dashboards and visualizations based on the credit card transaction data.





The screenshot shows the Metabase web application displaying a detailed table of transaction data. The table is titled 'BDA Project / Mycollection'. It includes a search bar, a 'Filter' button, a 'Summarize' button, and a 'Save' button. The table has the following columns: ID, Trans Date, Trans Time, Cc Num, Merchant, Category, Amt, First, Last, Gender, and Street. The table contains 10 rows of data, showing transactions from 2019-01-01 00:00:18 to 2019-01-01 00:07:27. The data is as follows:

ID	Trans Date	Trans Time	Cc Num	Merchant	Category	Amt	First	Last	Gender	Street
647c997f6d9759412de1414d	2019-01-01 00:00:18		2,703,186,189,652,095	fraud_Ripplin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove
647c997f6d9759412de1414e	2019-01-01 00:00:44		630,423,337,322	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Gre
647c997f6d9759412de1414f	2019-01-01 00:01:16		3,534,093,764,340,240	fraud_Kutch, Hermiston and Farrell	gas_transport	45	Jeremy	White	M	9443 Cynthia Cc
647c997f6d9759412de14150	2019-01-01 00:03:06		375,534,208,663,984	fraud_Keeling-Crist	misc_pos	41.96	Tyler	Garcia	M	408 Bradley Res
647c997f6d9759412de14151	2019-01-01 00:04:08		4,767,265,376,804,500	fraud_Stroman, Hudson and Erdman	gas_transport	94.63	Jennifer	Conner	F	4655 David Islar
647c997f6d9759412de14152	2019-01-01 00:00:51		38,859,492,057,661	fraud_Lind-Buckridge	entertainment	220.11	Edward	Sanchez	M	594 White Dale
647c997f6d9759412de14153	2019-01-01 00:05:08		6,011,360,759,745,864	fraud_Corwin-Collins	gas_transport	71.65	Steven	Williams	M	231 Flores Pass:
647c997f6d9759412de14154	2019-01-01 00:05:18		4,922,710,831,011,201	fraud_Herzog Ltd	misc_pos	4.27	Heather	Chase	F	6888 Hicks Stre
647c997f6d9759412de14155	2019-01-01 00:04:42		30,074,693,890,476	fraud_Rowe-Vandervort	grocery_net	44.54	Kelsey	Richards	F	889 Sarah Statio
647c997f6d9759412de14156	2019-01-01 00:06:23		4,642,894,980,163	fraud_Rutherford-Mertz	grocery_pos	24.74	Eddie	Mendez	M	1831 Faith View
647c997f6d9759412de14157	2019-01-01 00:07:27		5,559,857,416,065,248	fraud_Kiehn Inc	grocery_pos	96.29	Jack	Hill	M	5916 Susan Brid

The above screenshots depict the successful execution of running the MongoDB and Metabase containers, the dumping of data into the MongoDB container, and the establishment of the connection between Metabase and MongoDB.

2.3 MULTIPLE NOSQL QUERIES AND CUSTOMIZED DASHBOARDS

We ran multiple NoSQL queries against the credit card transaction dataset stored in MongoDB. Each query was designed to extract specific information and uncover insights related to fraud detection and transaction patterns. These queries allowed us to perform in-depth analysis and gain a comprehensive understanding of the dataset.

To present the findings in a visually appealing and informative manner, we utilized Meta base to create customized dashboards. Each dashboard consisted of multiple

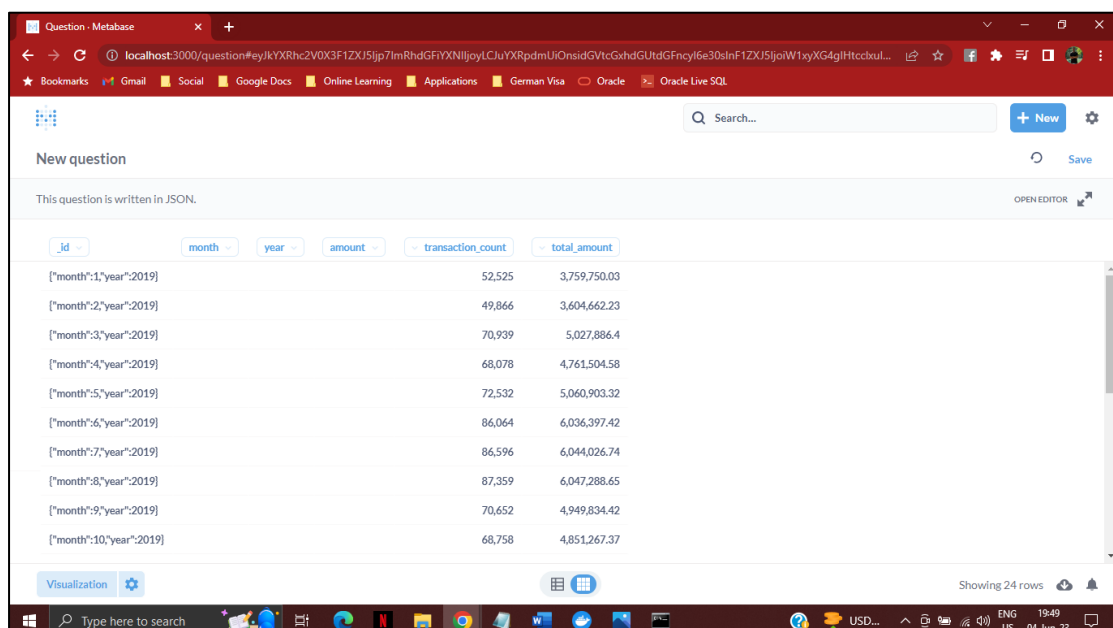
cards, with each card representing a separate query result. These cards were dynamically generated based on the query results, providing real-time insights and enabling stakeholders to easily navigate and interpret the data.

Each card on the dashboard was backed by a unique query that addressed a specific aspect of the credit card transaction data. For example, one card displayed the correlation between age and the amount involved in fraudulent transactions, while another card visualized the geographical distribution of fraudulent activities. By having separate cards for each query, we were able to provide focused insights and facilitate a more granular analysis of the dataset.

The flexibility of Meta base allowed us to customize the appearance and layout of the dashboards, ensuring that the information was presented in a clear and concise manner. We incorporated various visualization techniques such as bar charts, line graphs, and heatmaps to effectively communicate the findings and highlight patterns within the data.

```
1 {
2   {
3     $project: {
4       month: { $month: { $toDate: "$trans_date_trans_time" } },
5       year: { $year: { $toDate: "$trans_date_trans_time" } },
6       amount: "$amt"
7     }
8   },
9   {
10    $group: {
11      _id: { month: "$month", year: "$year" },
12      transaction_count: { $sum: 1 },
13      total_amount: { $sum: "$amount" }
14    }
15  },
16  {
17    $sort: { "_id.year": 1, "_id.month": 1 }
18  }
19 }
```

Month-on-Month Volume and Transactions Count



_id	month	year	amount	transaction_count	total_amount
["month":1,"year":2019]	1	2019		52,525	3,759,750.03
["month":2,"year":2019]	2	2019		49,866	3,604,662.23
["month":3,"year":2019]	3	2019		70,939	5,027,886.4
["month":4,"year":2019]	4	2019		68,078	4,761,504.58
["month":5,"year":2019]	5	2019		72,532	5,060,903.32
["month":6,"year":2019]	6	2019		86,064	6,036,397.42
["month":7,"year":2019]	7	2019		86,596	6,044,026.74
["month":8,"year":2019]	8	2019		87,359	6,047,288.65
["month":9,"year":2019]	9	2019		70,652	4,949,834.42
["month":10,"year":2019]	10	2019		68,758	4,851,267.37

```

1 [
2   {
3     $group: {
4       _id: "$state",
5       transaction_count: { $sum: 1 },
6       total_amount: { $sum: "$amt" }
7     }
8   }
9 ]

```

State Wise Transactions Count & Amount

<input type="text" value="_id"/>	<input type="text" value="transaction_count"/>	<input type="text" value="total_amount"/>
MT	16,806	1,206,232.14
ME	23,433	1,482,788.14
KY	40,981	2,743,359.63
SD	17,574	1,233,010.43
AK	2,963	210,694.99
RI	745	52,397.59
WI	41,738	2,897,850.34
OH	66,627	4,823,536.63
CT	10,979	703,500.88
MD	37,345	2,381,427.49
IN	39,539	2,742,840.09
<input type="button" value="Visualization"/> <input type="button" value="⚙️"/>		

```
1 [
2   {
3     $group: {
4       _id: "$category",
5       transaction_count: { $sum: 1 },
6       total_amount: { $sum: "$amt" }
7     }
8   }
9 ]
```

Category Wise Transactions Count & Amount

<div>_id</div>	<div>transaction_count</div>	<div>total_amount</div>
personal_care	130,085	6,250,310.53
shopping_net	139,322	12,112,929.67
gas_transport	188,029	11,935,567.85
grocery_net	64,878	3,483,204.05
entertainment	134,118	8,602,726.58
shopping_pos	166,463	13,135,052.22
kids_pets	161,727	9,303,806.75
food_dining	130,729	6,666,407.52
misc_net	90,654	7,268,761.94
grocery_pos	176,191	20,550,943.97

```

1 [
2   {
3     $addFields: {
4       age: {
5         $floor: {
6           $divide: [
7             { $subtract: [new Date(), { $toDate: "$dob" }] },
8             31536000000|
9           ]
10        }
11      }
12    },
13  },
14  {
15    $group: {
16      _id: "$age",
17      average_amount: { $avg: "$amt" }
18    }
19  }
20 ]

```

Grouping the Avg Volume by Age

▼ _id	▼ average_amount
49	74.53
82	65.64
39	76.24
24	60.01
67	64.1
70	65.32
53	66.78
42	73.95
88	68.18
93	68.01

```

1 [
2   {
3     $addFields: {
4       age: {
5         $floor: {
6           $divide: [
7             { $subtract: [new Date(), { $toDate: "$dob" }] },
8             31536000000
9           ]
10        }
11      }
12    }
13  },
14  {
15    $group: {
16      _id: "$age",
17      volume: { $sum: "$amt" }
18    }
19  }
20 ]
21

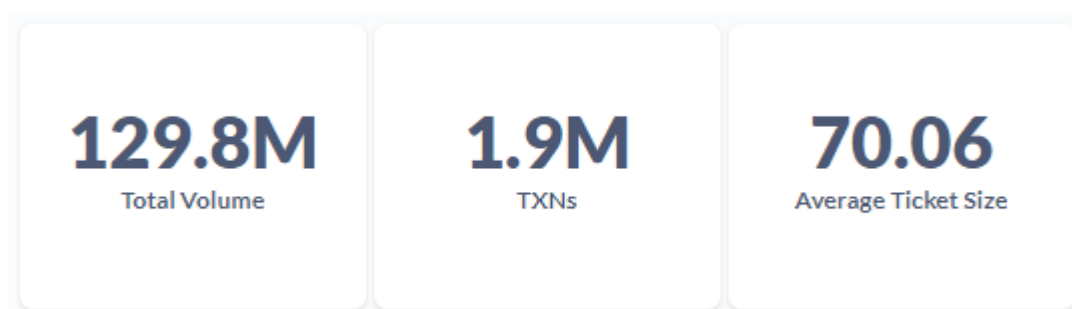
```

Amount By Age

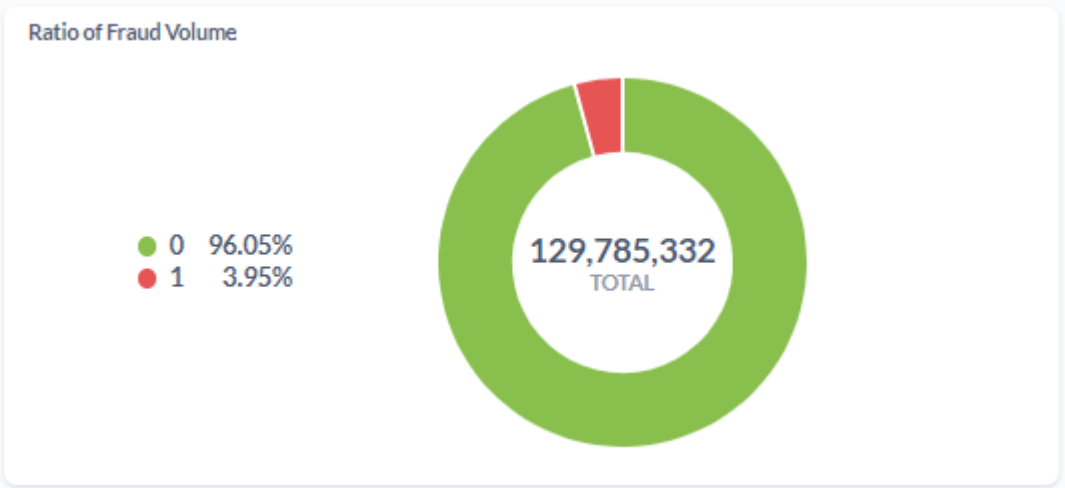
3. Final Dashboard & Summary

As mentioned above, a card (graphical visualization) is made using a query, and the collections of these cards help us build the dashboard. All the cards that were made, which were then eventually used to finalize the dashboard are shown below and their respective queries are mentioned in the Appendix.

All in all, using mongodb and metabase on dockerized containers provide scalability and flexibility. You can easily query your data using metabase and perform visualization through dashboards. The connection to the db is easily performed as well, without any hassle.



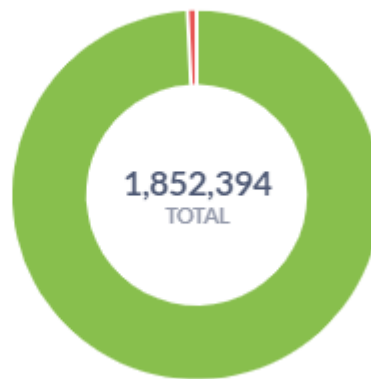
State Wise Stats				
State	Sum of Amt	Count	Average of Amt	% of Fraud Vol
AK	210,694.99	2,963	71.11	14.34
AL	3,810,639.91	58,521	65.12	3.91
AR	3,370,419.27	44,611	75.55	2.9
AZ	1,156,227.12	15,362	75.27	2.81
CA	5,906,869.77	80,495	73.38	3.49
CO	1,537,244.77	19,766	77.77	3.87
CT	703,500.88	10,979	64.08	4.09
DC	385,896.78	5,130	75.22	4.22
DE	4,630.44	9	514.49	100
FL	4,448,375.13	60,775	73.19	3.9
GA	2,601,360.56	37,340	69.67	4.28
...



Ratio of Fraud TXNs

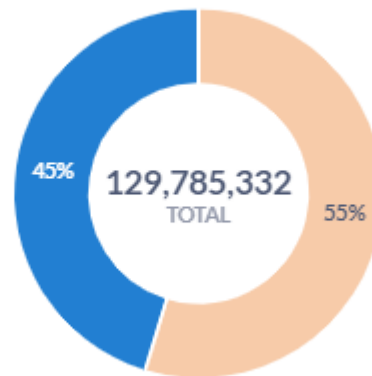
...

0 99.479%
1 0.521%



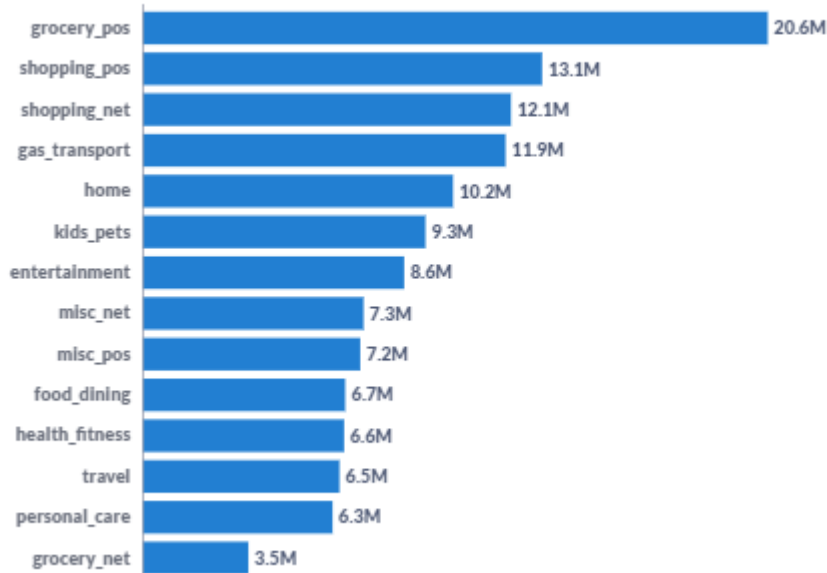
Gender Wise Volume

F
M

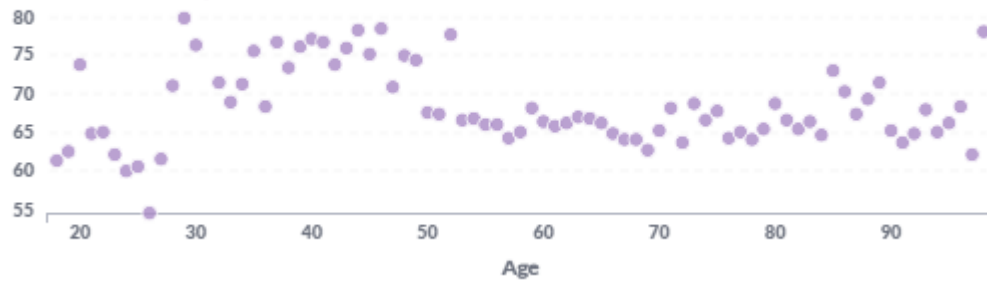


Category Wise Amount

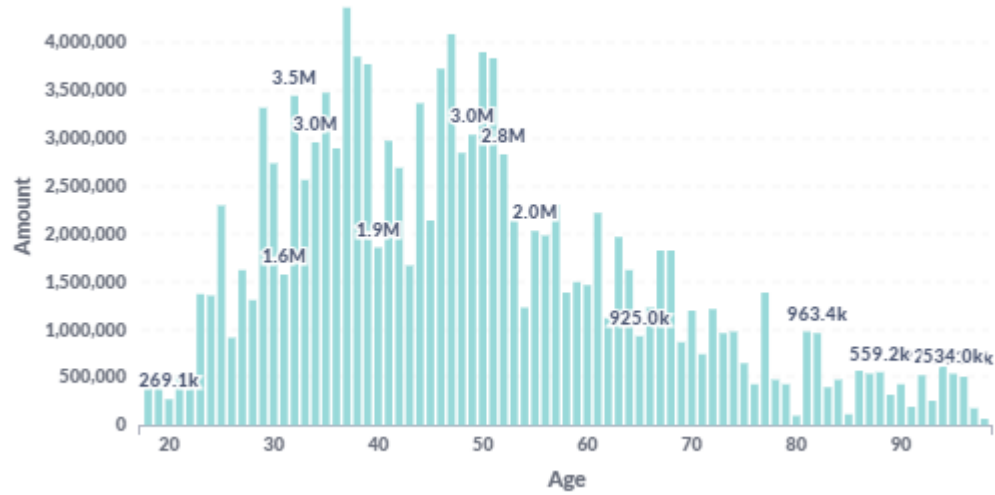
...



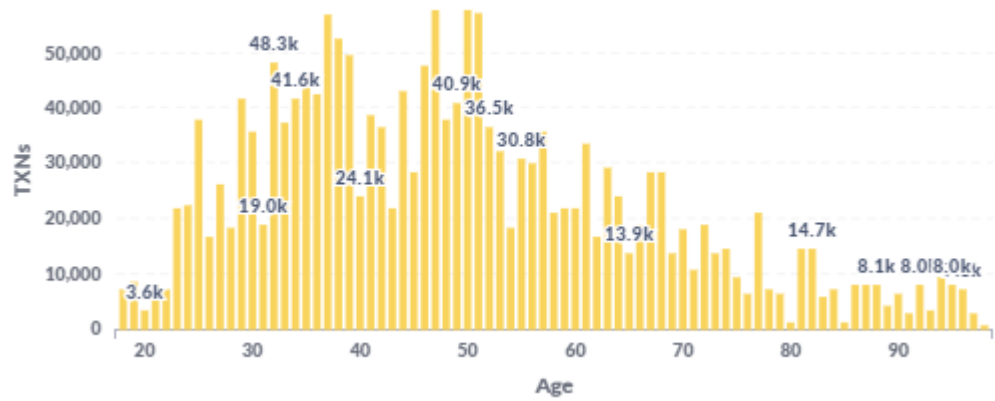
Avg Ticket Size v/s Age



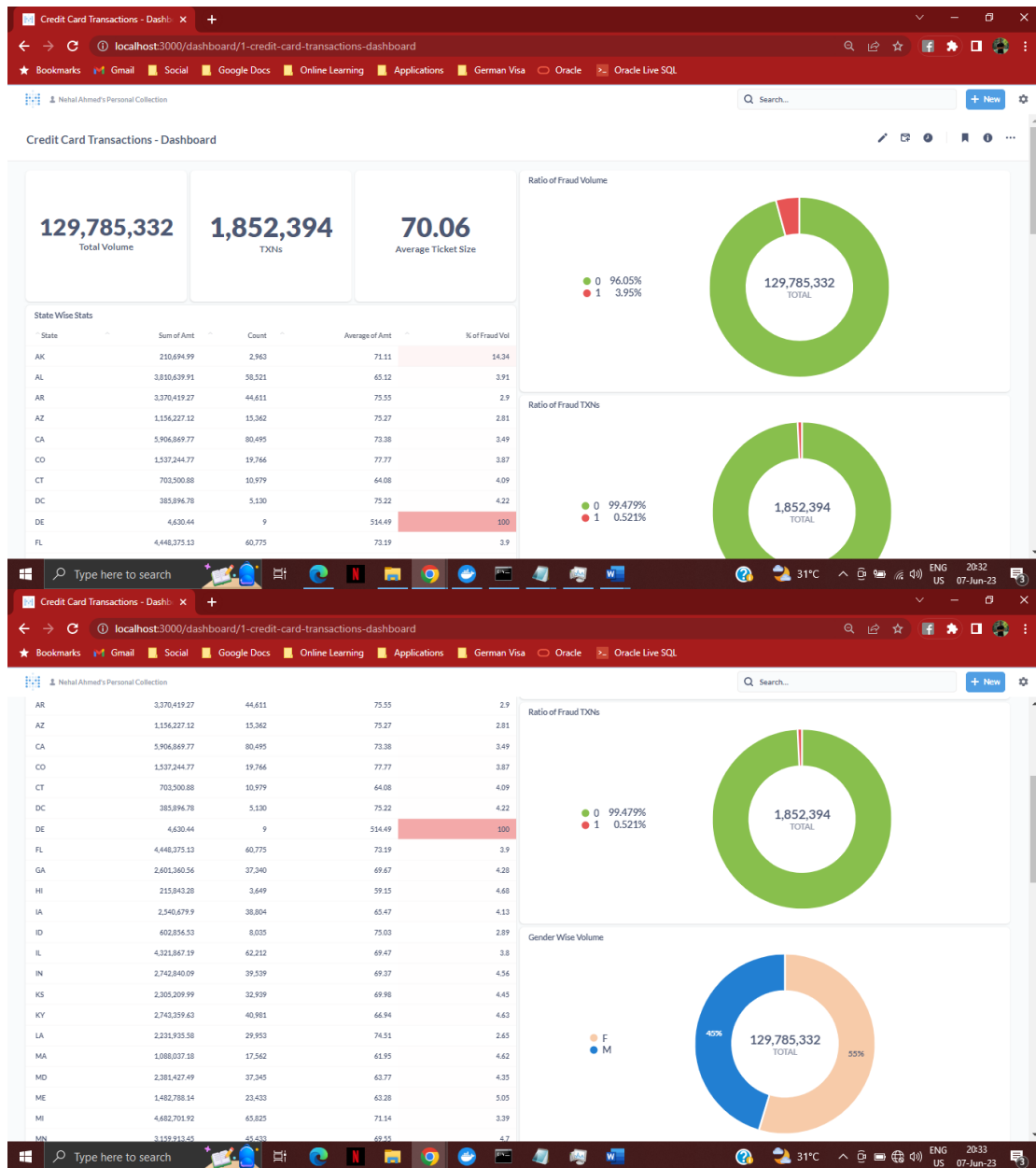
Total Volume by Age

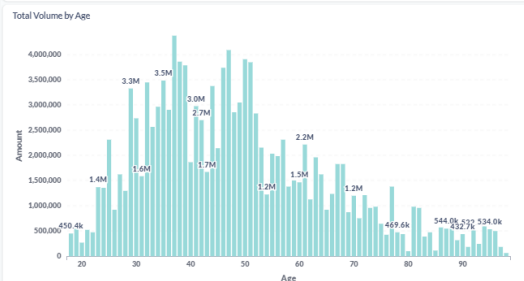
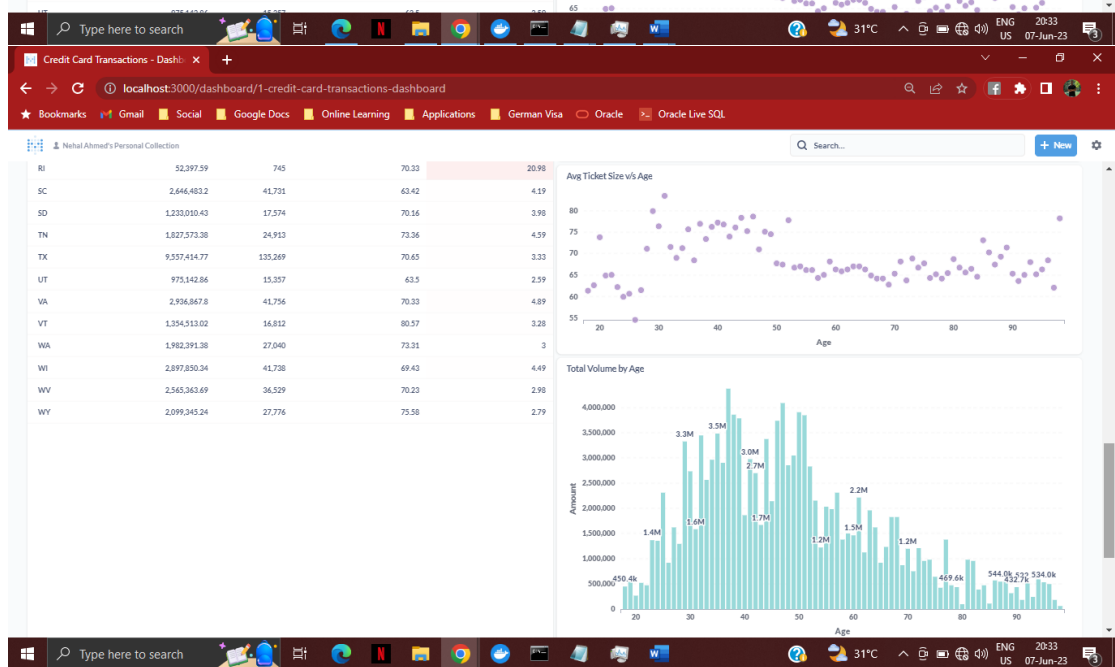
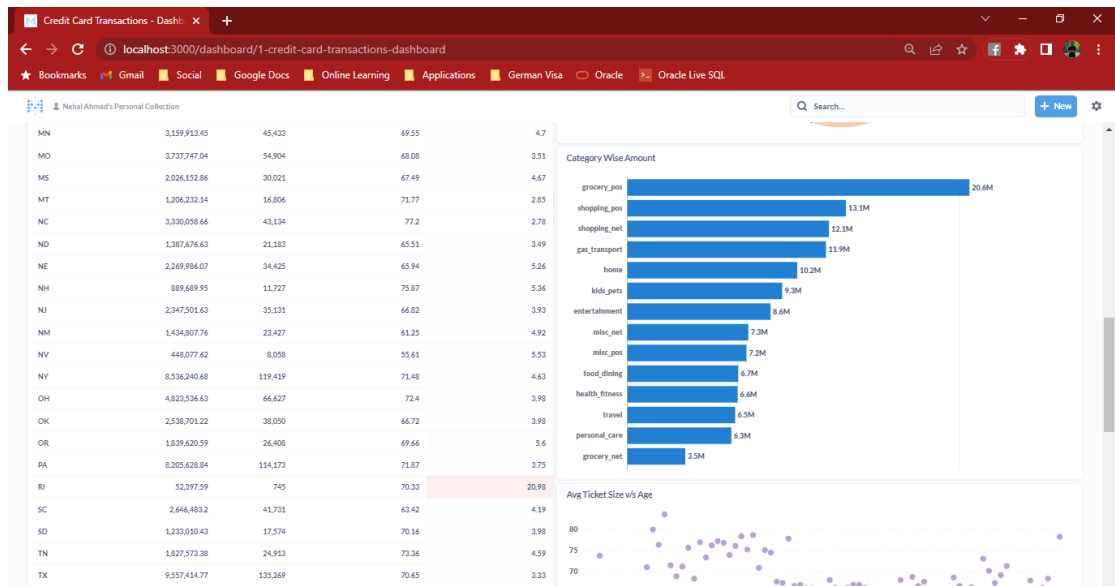


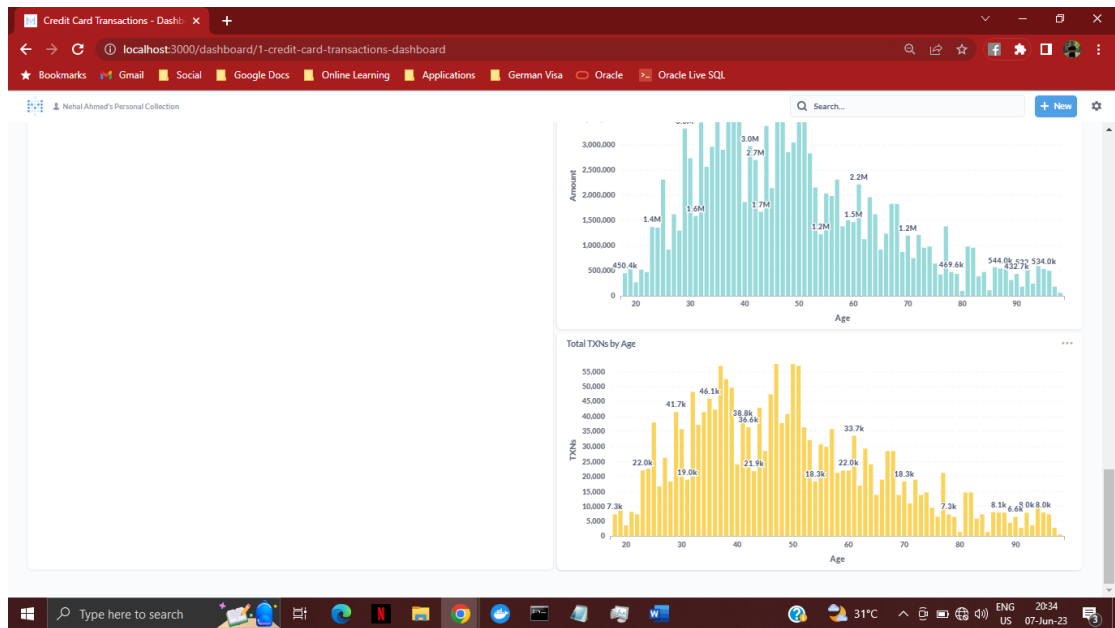
Total TXNs by Age



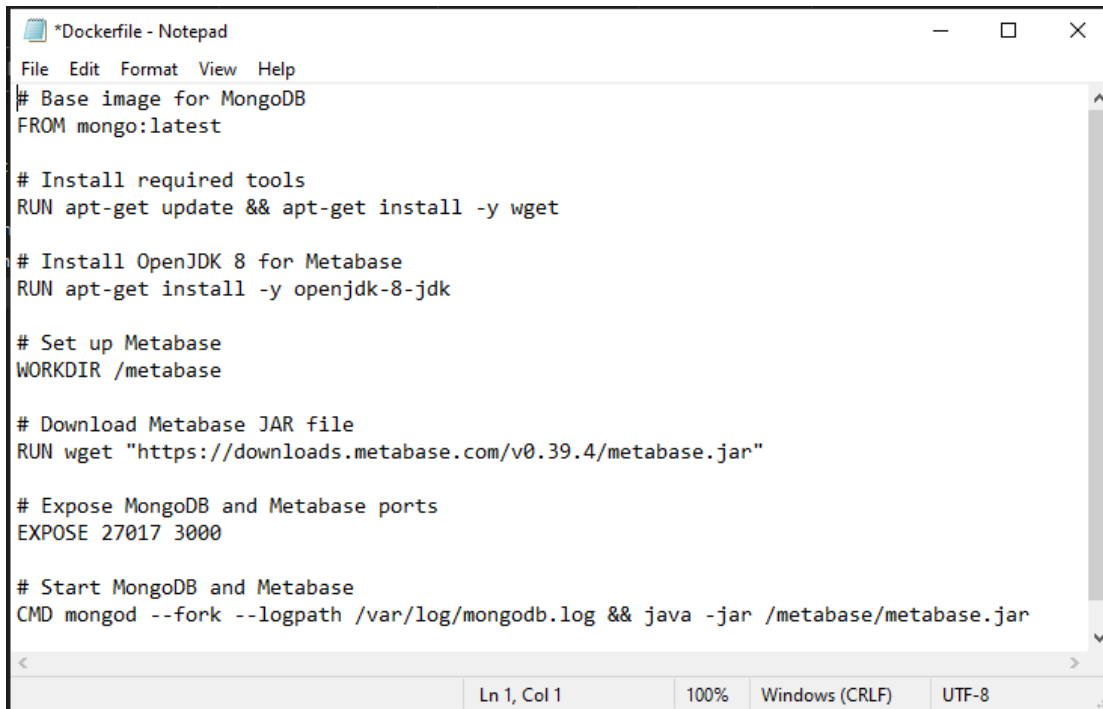
The final dashboard is shown below. The screenshot is divided into several images as it was impossible to fit the whole dashboard in one image.







Finally, in order to run both the dockers together in the future, we need to create a docker file. The screenshots for creating docker file is shown below. The docker-compose.yml file was also created.



```
*Dockerfile - Notepad
File Edit Format View Help
# Base image for MongoDB
FROM mongo:latest

# Install required tools
RUN apt-get update && apt-get install -y wget

# Install OpenJDK 8 for Metabase
RUN apt-get install -y openjdk-8-jdk

# Set up Metabase
WORKDIR /metabase

# Download Metabase JAR file
RUN wget "https://downloads.metabase.com/v0.39.4/metabase.jar"

# Expose MongoDB and Metabase ports
EXPOSE 27017 3000

# Start MongoDB and Metabase
CMD mongod --fork --logpath /var/log/mongodb.log && java -jar /metabase/metabase.jar

Ln 1, Col 1    100%    Windows (CRLF)    UTF-8
```



```
docker-compose - Notepad
File Edit Format View Help
version: '3'
services:
  mongodb:
    image: mongo:latest
    volumes:
      - mongodb_data:/data/db
      - C:/Users/HP/Downloads/project:/data
    ports:
      - 27017:27017

  metabase:
    image: metabase/metabase:v0.39.4
    ports:
      - 3000:3000
    depends_on:
      - mongodb
    environment:
      - "MB_DB_TYPE=mongo"
      - "MB_DB_DBNAME=bda"
      - "MB_DB_PORT=27017"
      - "MB_DB_HOST=mongodb"
      - "MB_DB_USER=admin"
      - "MB_DB_PASS=admin"
    volumes:
      - metabase_data:/metabase-data

volumes:
  mongodb_data:
  metabase_data:

Ln 1, Col 1    100%    Windows (CRLF)    UTF-8
```

4. Appendix

Containers

Images

Volumes

Dev Environments BETA

Learning Center

Extensions

Add Extensions

Search

Only show running containers

	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	mongodb 5ead8dd06e51	mongo	Running	27017-27017	3 days ago	
<input type="checkbox"/>	metabase 8796c7891895	metabase/metabase	Running	3000-3000	3 days ago	

Showing 2 items

Task Manager

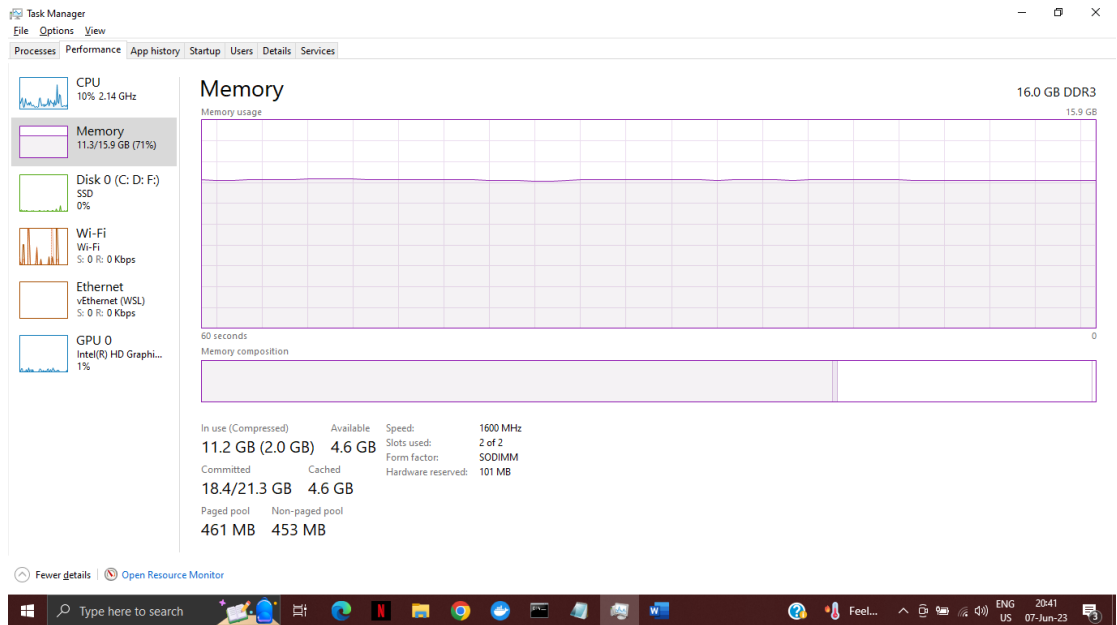
File Options View

Processes Performance App history Startup Users Details Services

Name	Status	9% CPU	70% Memory	1% Disk	0% Network	0% GPU	GPU engine	Power usage	Power usage t...
Vmmem		1.7%	3,567.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Google Chrome (12)		0.2%	711.2 MB	0 MB/s	0.1 Mbps	0%	GPU 0 - 3D	Very low	Very low
> Oracle RDBMS Kernel Executable		0.1%	336.4 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
> Microsoft Edge (14)		0.1%	225.4 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D	Very low	Very low
> Microsoft Word (3)		0.1%	152.8 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D	Very low	Very low
> Antimalware Service Executable		0.4%	123.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Docker Desktop (5)		0.1%	97.8 MB	0 MB/s	0 Mbps	0%	GPU 0 - 3D	Very low	Very low
> Windows Explorer		0.4%	57.8 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Desktop Window Manager		0.4%	50.1 MB	0 MB/s	0 Mbps	0.1%	GPU 0 - 3D	Very low	Very low
com.docker.build		0%	41.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Docker Desktop Backend		0%	41.4 MB	0.1 MB/s	0 Mbps	0%		Very low	Very low
Secure System		0%	37.8 MB	0 MB/s	0 Mbps	0%		Very low	Very low
Docker Desktop API Proxy		0%	32.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Service Host Diagnostic Policy ...		0%	30.2 MB	0 MB/s	0 Mbps	0%		Very low	Very low
MoUSO Core Worker Process		0%	28.8 MB	0 MB/s	0 Mbps	0%		Very low	Very low
vpknet-bridge		0.7%	24.9 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Microsoft Windows Search Inde...		0%	22.7 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> msrdc.exe (21)		0.1%	22.5 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Task Manager		1.3%	22.1 MB	0 MB/s	0 Mbps	0%		Very low	Very low
> Service Host: DCOM Server Proc...		0%	15.1 MB	0 MB/s	0 Mbps	0%		Very low	Very low

Fewer details

End task



About

Device specifications

Device name	DESKTOP-EBK6FP1
Processor	Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz 2.60 GHz
Installed RAM	16.0 GB
Device ID	5E7CD123-3336-42C0-90A4-E716782634DC
Product ID	00342-41306-86300-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Copy

Rename this PC

Windows specifications

Edition	Windows 10 Home Single Language
Version	22H2
Installed on	01-Dec-20
OS build	19045.2965
Experience	Windows Feature Experience Pack 1000.19041.1000.0

Data Source: <https://www.kaggle.com/datasets/kartik2112/fraud-detection>

Queries for each card (visualization):

Total Volume

```
[
  {
    "$group": {
      "_id": null,
      "sum": {
        "$sum": "$amt"
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "sum": true
    }
  }
]
```

TXNs

```
[
  {
    "$group": {
      "_id": null,
      "count": {
        "$sum": 1
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "count": true
    }
  }
]
```

Average Ticket Size

```
[
  {
    "$group": {
      "_id": null,
      "avg": {
        "$avg": "$amt"
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "avg": true
    }
  }
]
```

State Wise Stats

```
[
  {
    "$group": {
      "_id": {
        "state": "$state"
      },
      "sum": {
        "$sum": "$amt"
      },
      "count": {
        "$sum": 1
      },
      "avg": {
        "$avg": "$amt"
      },
      "% of Fraud Vol~sum~where": {
        "$sum": {
          "$cond": {
            "if": {
              "$eq": [
                "$is_fraud",
                1
              ]
            },
            "then": "$amt",
            "else": 0
          }
        }
      },
      "% of Fraud Vol~sum": {
        "$sum": "$amt"
      }
    },
    {
      "$addFields": {
        "% of Fraud Vol": {
          "$multiply": [
            {
              "$cond": [
                {
                  "$eq": [
                    "$% of Fraud Vol~sum",
                    0
                  ]
                },
                null,
                {
                  "$divide": [
                    "$% of Fraud Vol~sum~where",
                    "$% of Fraud Vol~sum"
                  ]
                }
              ]
            },
            100
          ]
        }
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "state": "$_id.state",
      "sum": true,
      "count": true,
      "avg": true,
      "% of Fraud Vol": true
    }
  }
]
]
```

Ratio of Fraud Volume

```
[
  {
    "$group": {
      "_id": {
        "is_fraud": "$is_fraud"
      },
      "sum": {
        "$sum": "$amt"
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "is_fraud": "$_id.is_fraud",
      "sum": true
    }
  }
]
```

Ratio of Fraud Volume

```
[
  {
    "$group": {
      "_id": {
        "is_fraud": "$is_fraud"
      },
      "count": {
        "$sum": 1
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "is_fraud": "$_id.is_fraud",
      "count": true
    }
  }
]
```

Gender Wise Volume

```
[
  {
    "$group": {
      "_id": {
        "gender": "$gender"
      },
      "sum": {
        "$sum": "$amt"
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "gender": "$_id.gender",
      "sum": true
    }
  }
]
```

Category Wise Amount

```
[
  {
    "$group": {
      "_id": {
        "category": "$category"
      },
      "sum": {
        "$sum": "$amt"
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  },
  {
    "$project": {
      "_id": false,
      "category": "$_id.category",
      "sum": true
    }
  },
  {
    "$sort": {
      "sum": -1,
      "category": 1
    }
  }
]
```

Total Volume By Age

```
[
  {
    $addFields: {
      age: {
        $floor: {
          $divide: [
            { $subtract: [new Date(), { $toDate: "$dob" }] },
            31536000000
          ]
        }
      }
    },
  },
  {
    $group: {
      _id: "$age",
      volume: { $sum: "$amt" }
    }
  }
]
```

Avg Ticket Size v/s Age

```
[
  {
    $addFields: {
      age: {
        $floor: {
          $divide: [
            { $subtract: [new Date(), { $toDate: "$dob" }] },
            31536000000
          ]
        }
      }
    },
  },
  {
    $group: {
      _id: "$age",
      average_amount: { $avg: "$amt" }
    }
  }
]
```

Total TXNs by Age

```
[
  {
    $addFields: {
      age: {
        $floor: {
          $divide: [
            { $subtract: [new Date(), { $toDate: "$dob" }] },
            31536000000
          ]
        }
      }
    },
  },
  {
    $group: {
      _id: "$age",
      count: { $sum: 1 }
    }
  }
]
```