CREDIT CARD FRAUD DETECTION

CAPSTONE PROJECT

Group 1: Pandas

BY

Niraj Bharatkumar Makwana (000964560)

Akshar Nimeshkumar Patel (000961577)

Sashidhar Kodamagundla (000940889)

INDEX

Problem statement

As digital transactions continue to grow, fraudulent credit card activities have become more complex and harder to detect. Traditional manual methods of identifying fraud are no longer effective or scalable. Therefore, the organization requires an automated solution which helps to detect and prevent fraudulent transactions in real time.

Project vision

This project aims to reduce financial losses and enhance customer trust by delivering faster and more reliable fraud prevention. To achieve this, we have come up with an ML model which helps to prevent fraudulent transactions in real time. With the help of key attributes such as location, amount, time, and customer behavior we can predict the fraud.

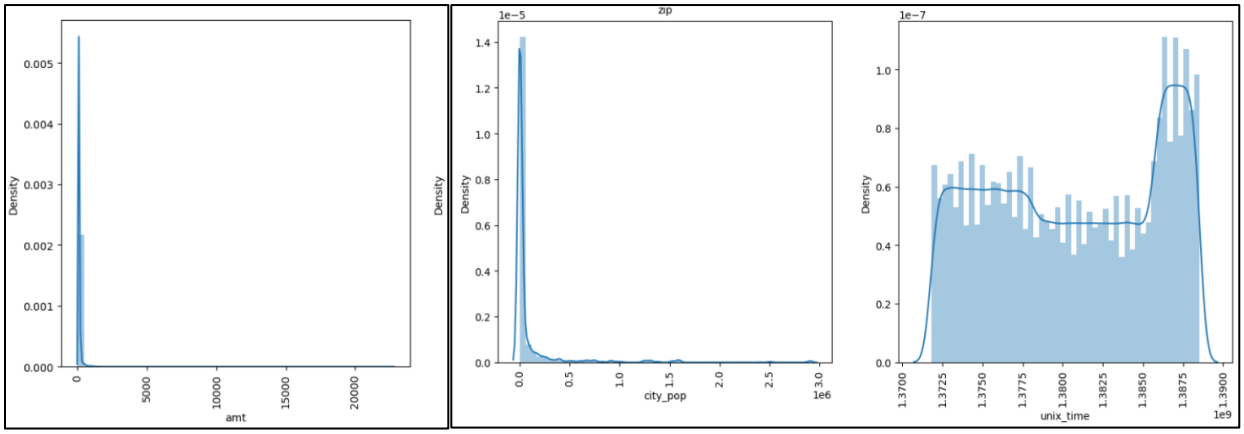
Stakeholder

Financial Institutions / Banks
Credit Card Users / Customers
Fraud Analysts / Investigators
Data Scientists / ML Engineers
QA / Testing Teams
Merchants / Retailers
Cybersecurity Teams
Compliance and Risk Officers
Product Managers
Customer Support Teams
Third-party Payment Processors (e.g., GPay, PayPal,POS)

Descriptive statistics

Descriptive statistics give a quick overview of the data. We looked at transaction amount, city population, and time of transaction. Key stats like mean, standard deviation, and percentiles show how values are spread.

Density plots revealed the data is skewed—not evenly distributed, which can affect model performance. This step helps us understand the data before building machine learning models.



	city_pop	unix_time	amt
count	5.557190e+05	5.557190e+05	555719.000000
mean	8.822189e+04	1.380679e+09	69.392810
std	3.003909e+05	5.201104e+06	156.745941
min	2.300000e+01	1.371817e+09	1.000000
25%	7.410000e+02	1.376029e+09	9.630000
50%	2.408000e+03	1.380762e+09	47.290000
75%	1.968500e+04	1.385867e+09	83.010000
max	2.906700e+06	1.388534e+09	22768.110000

Data preprocessing

In this project, we conducted several preprocessing and data wrangling steps to prepare the dataset for effective modeling and analysis. Initially we loaded raw datasets and began inspection by checking the quick glance of individual variables, checked for missing, Nulls and duplicate values in the dataset. This helped us to identify issues like null values, duplicate records, and irrelevant features.

```
[6]: df = pd.read_csv(r"C:\Users\User\OneDrive - Southern Alberta Institute of Technology\Desktop\Data Science\Machine Larning\Applied Machine Learning Project - International (Capstone)\fraudTest.csv")
     df.head()
```

[6]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	job	dob	trans_num	unix_
0	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	2da90c7d74bd46a0caf3777415b3ebd3	137181
1	1	2020-06-21 12:14:33	3573030041201292	fraud_Sporer-Keebler	personal_care	29.84	Joanne	Williams	F	3638 Marsh Union	...	40.3207	-110.4360	302	Sales professional, IT	1990-01-17	324cc204407e99f51b0d6ca0055005e7	137181
2	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	c81755dbbbea9d5c77f094348a7579be	137181
3	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	2159175b9efe66dc301f149d3d5abf8c	137181
4	4	2020-06-21 12:15:17	3526826139003047	fraud_Johnston-Casper	travel	3.19	Nathan	Massey	M	5783 Evan Roads Apt. 465	...	44.2529	-85.0170	1126	Furniture designer	1955-07-06	57ff021bd3f328f8738bb535c302a31b	137181

```
[8]: #Check Missing values in the dataset
     print(df.info())
     print(df.isnull().sum())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 555719 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Unnamed: 0             555719 non-null  int64
 1   trans_date_trans_time  555719 non-null  object
 2   cc_num                 555719 non-null  int64
 3   merchant               555719 non-null  object
 4   category               555719 non-null  object
 5   amt                    555719 non-null  float64
 6   first                  555719 non-null  object
 7   last                   555719 non-null  object
 8   gender                 555719 non-null  object
 9   street                 555719 non-null  object
 10  city                   555719 non-null  object
 11  state                  555719 non-null  object
 12  zip                    555719 non-null  int64
 13  lat                    555719 non-null  float64
 14  long                   555719 non-null  float64
 15  city_pop               555719 non-null  int64
 16  job                    555719 non-null  object
 17  dob                    555719 non-null  object
 18  trans_num              555719 non-null  object
 19  unix_time              555719 non-null  int64
 20  merch_lat              555719 non-null  float64
 21  merch_long             555719 non-null  float64
 22  is_fraud               555719 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 97.5+ MB
```

```
# Identifying if any column exists with only null values
df.isnull().all(axis=0).any()
```

```
False
```

```
# checking duplicates
sum(df.duplicated(subset = 'cc_num')) == 0
```

```
False
```

No Duplicates and NUll were found in the dataset

Wrangling

We removed unnecessary columns from the dataset to simplify the analysis and focus on relevant features. Columns like names, street address, date of birth, and transaction ID were dropped because they don't help in detecting fraud.

Deleted Unwanted columns from Dataset

```
#Drop Irrelevant Columns
drop_cols = ["Unnamed: 0","trans_date_trans_time" , "trans_num", "first", "last", "street","dob"]
df.drop(columns=drop_cols, inplace=True)
```

Feature engineering

In this step, we performed feature engineering by converting the date of birth to a datetime format, calculating the age of everyone, and splitting the transaction timestamp into separate date and time columns to make the data more useful for analysis.

Feature Engineering (DOB -> Age)

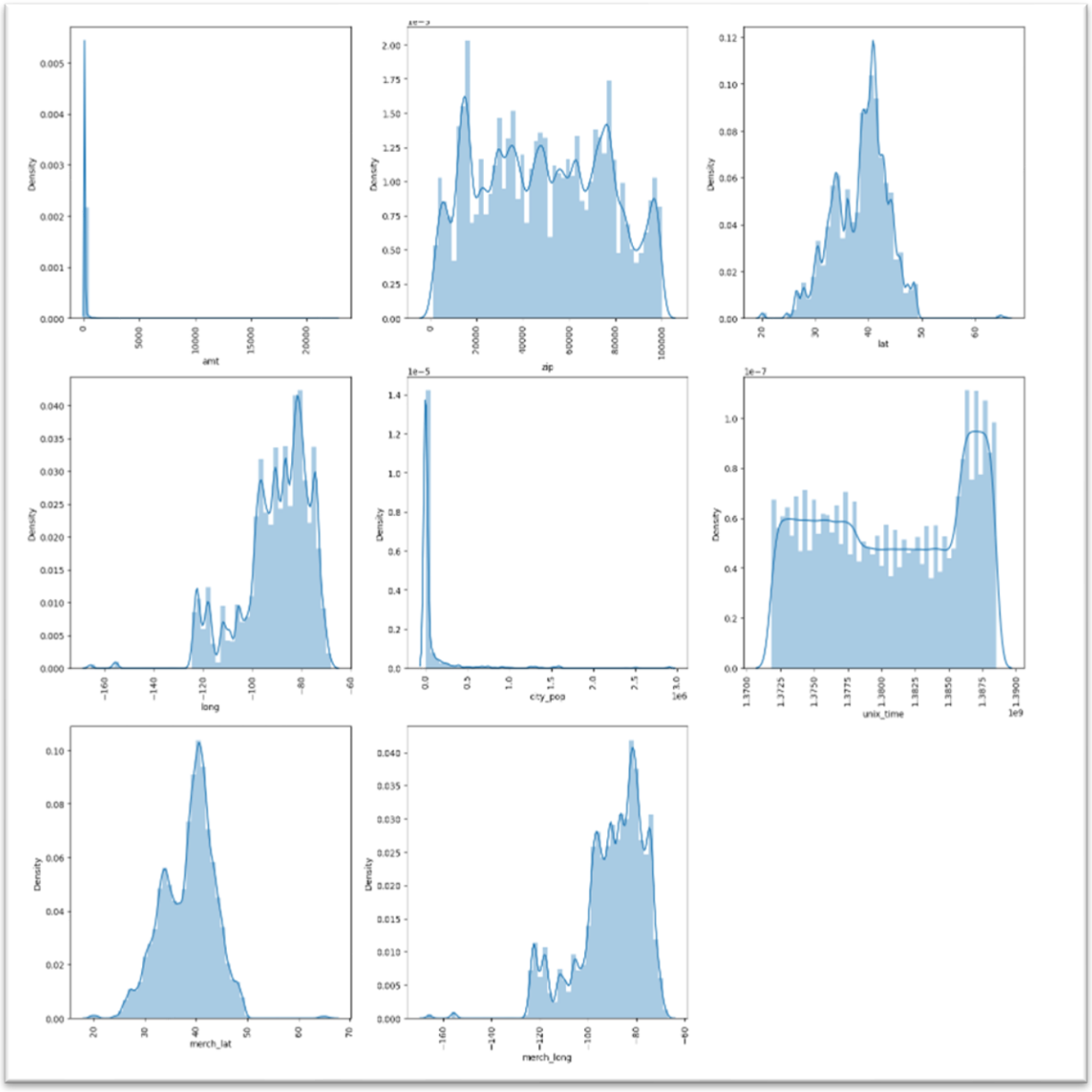
```
#Convert DOB to Age
df['dob'] = pd.to_datetime(df['dob'], errors='coerce')
df['age'] = (pd.to_datetime('today') - df['dob']).dt.days // 365
```

```
df[['date', 'time']] = df['trans_date_trans_time'].str.split(' ', expand=True)
print(df[['date', 'time']].head())
```

```
         date      time
0  2020-06-21  12:14:25
1  2020-06-21  12:14:33
2  2020-06-21  12:14:53
3  2020-06-21  12:15:15
4  2020-06-21  12:15:17
```

Data visualization in Univariate

Univariate analysis examines one variable at a time to understand its distribution, central tendency, and spread—helping identify patterns, outliers, or skewness in the data.

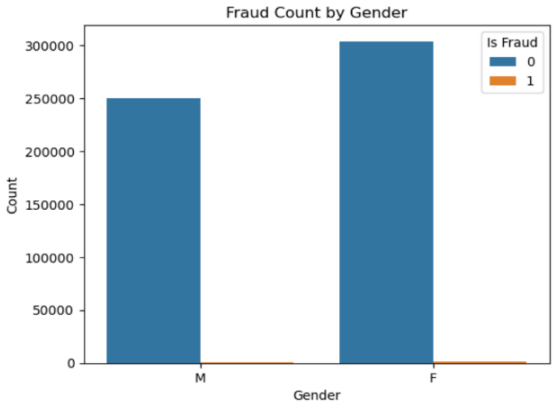


This image shows the distribution patterns of eight key variables using density plots, which reveal how frequently different values occur across the dataset; these are continuous variables, and we can see that **amt, city_pop**, and **unix_time** are highly skewed.

Data visualization in Bivariate:

1. Fraud Count by Gender

The graph shows that males are involved in significantly more fraud cases than females. This suggests that gender may play a role in fraud exposure or behavior, although it should be considered carefully to avoid introducing bias into predictive models.

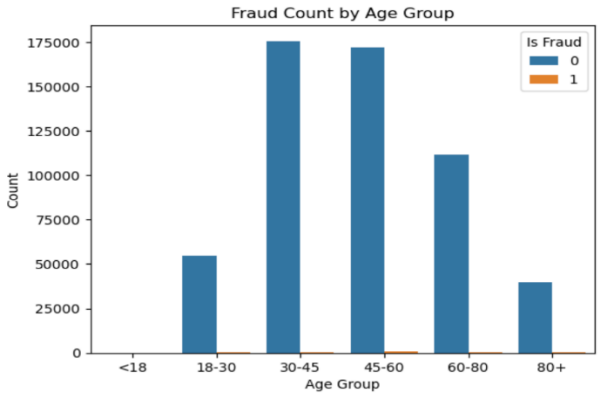


2.Transaction Amount in Fraud

The box plot reveals that fraudulent transactions tend to have higher amounts and greater variability compared to non-fraudulent ones. Many fraud cases involve large sums, indicating that transaction amount is a strong signal for identifying potential fraud.

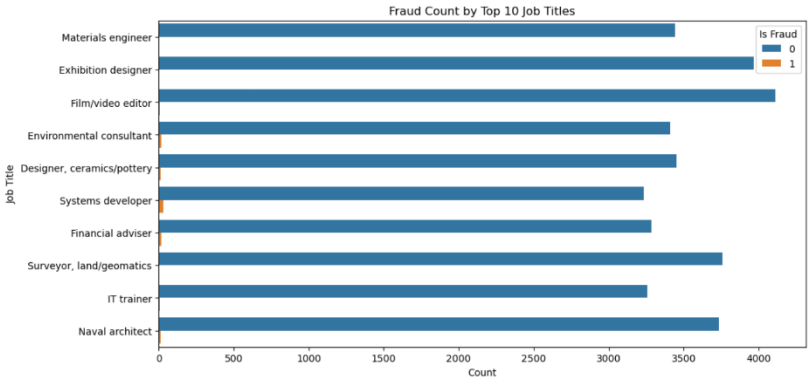
3. Fraud Count by Age Group

Individuals aged 30 to 45 account for the highest number of fraud cases, followed by those aged 45 to 60. This pattern likely reflects the financial activity levels of these age groups, making them more susceptible to fraud or more frequently targeted.



4. Fraud Count by Top 10 Job Titles

The graph highlights that job titles such as materials engineer, historian, and exhibition designer are most associated with fraud cases. This may point to industry-specific vulnerabilities or transaction behaviors linked to certain professions

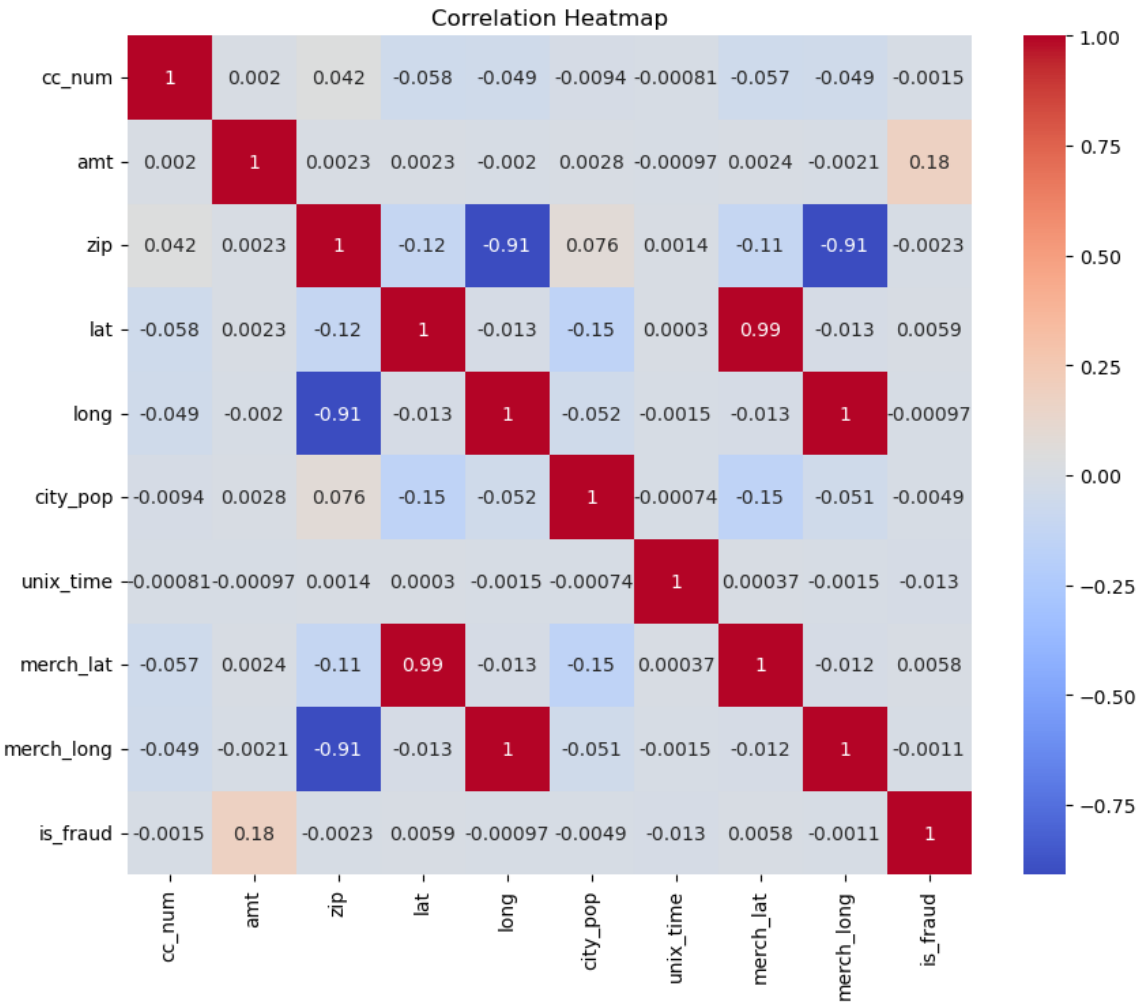


Correlation Heatmap

This heatmap represents the correlation of different variables in the dataset are related to each other. It uses color intensity to represent the strength and direction of these relationships.

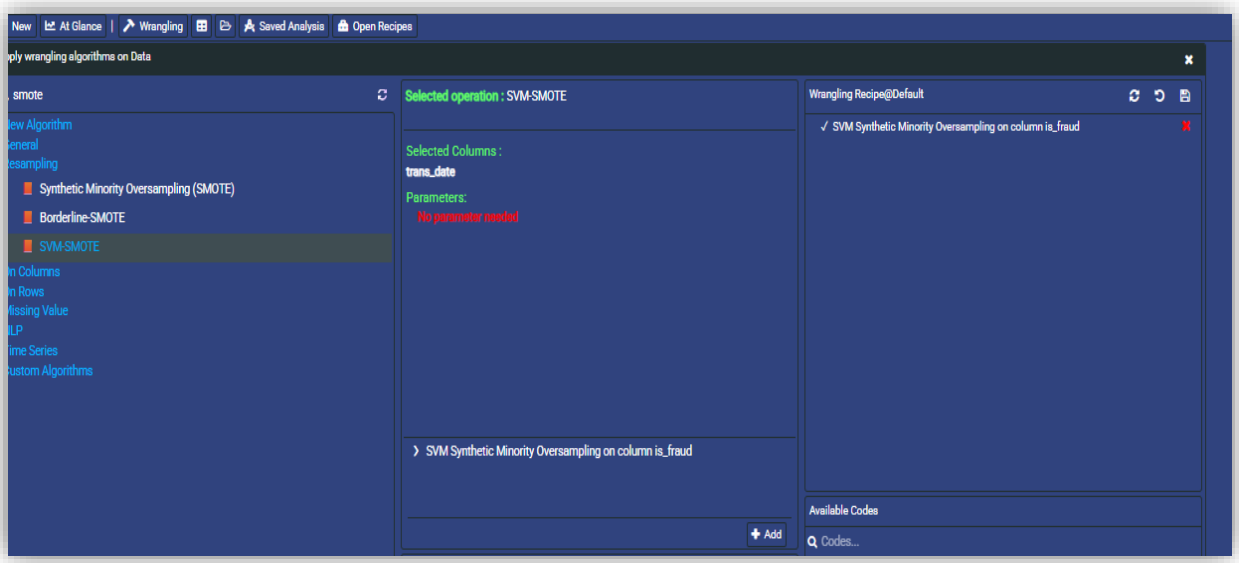
- **Dark Red**: Strong positive correlation (close to +1)
- **Light Red/Orange**: Moderate positive correlation
- **White or Pale Shades**: Weak or no correlation (around 0)
- **Light Blue**: Moderate negative correlation
- **Dark Blue**: Strong negative correlation (close to –1)

The deeper the color, the stronger the relationship—whether positive or negative.

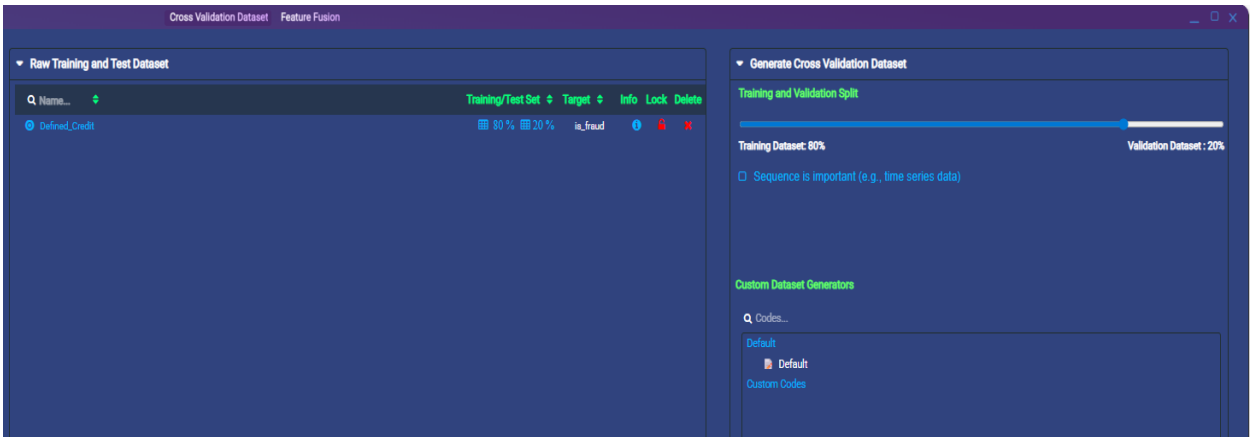


Cross-validation approach

To handle the imbalance data in the target variable **is_fraud**, we applied the **SMOTE** (Synthetic Minority Over-sampling Technique), which generates synthetic examples of the minority class. This helps the model avoid bias toward the majority class, improves its ability to detect fraudulent transactions, and leads to more reliable and fair predictions.



We used an **80-20%** train-test split to evaluate our model's performance, ensuring it learns from a majority of the data while being tested on unseen samples.



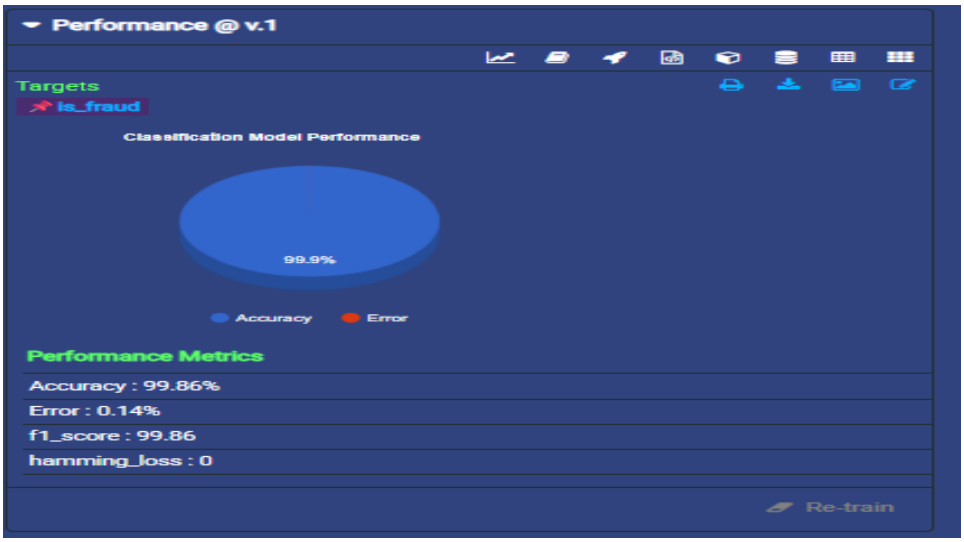
Details of all models built

Initially, we developed and evaluated three different machine learning models to detect fraudulent transactions: XGBoost Classifier, Logistic Regression, and Random Forest Classifier. Each model was assessed based on key performance metrics including accuracy, error rate, and F1 score, which balances precision and recall.

Model	Accuracy (%)	Error Rate (%)	F1 Score (%)
XGBoost Classifier	99.80	0.20	99.80
Logistic Regression	99.63	0.37	99.63
Random Forest	99.86	0.14	99.86

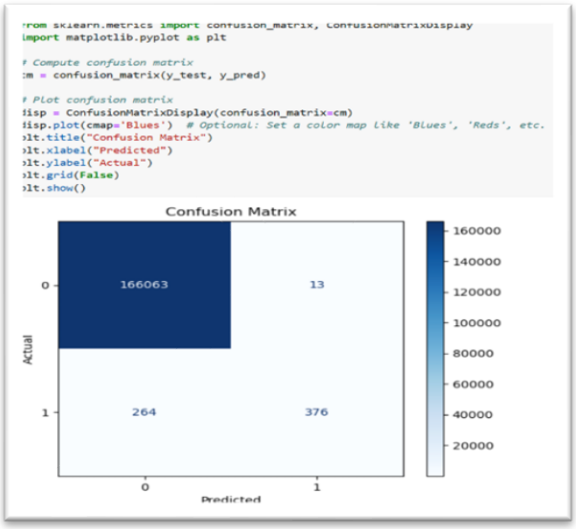
Selection of the Best Model: Justification Based on Evaluation Metrics

After comparing the results, the Random Forest model emerged as the best fit due to its superior accuracy and lower error rate, making it the most reliable choice for detecting fraud in this dataset. Random Forest Classifier performed slightly better, with an accuracy of 99.86%, an error rate of 0.14%, and an F1 score of 99.86%.



Confusion Matrix – Random Forest Classifier

To further evaluate model performance beyond accuracy and F1-score, we analyzed the confusion matrix of our best-performing model—Random Forest Classifier.



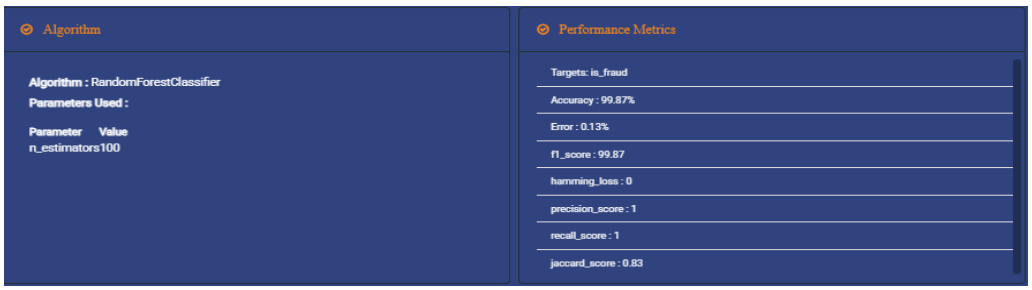
	Predicted: No Fraud	**Predicted: Fraud**
Actual: No Fraud	**True Negatives (TN): 166,063**	**False Positives (FP): 13**
Actual: Fraud	**False Negatives (FN): 264**	**True Positives (TP): 376**

This matrix helps us understand the types of errors the model makes and assess its real-world reliability. The model correctly identified **166,063 non-fraud cases** and **376 fraud cases**. However, it misclassified **13 non-fraud cases as fraud** (false alarms) and missed **264 actual fraud cases**. These insights are critical for improving fraud detection and minimizing both false positives and false negatives.

Model deployment strategy

1. **Model Governance**

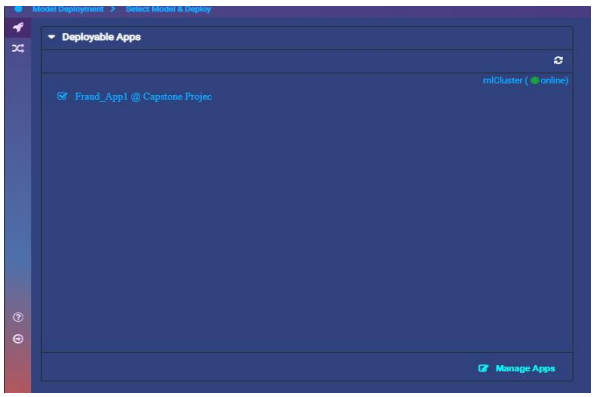
- We reviewed the final model and published it among the team, then accepted the model after evaluating its performance and metrics.



2. Model Deployment

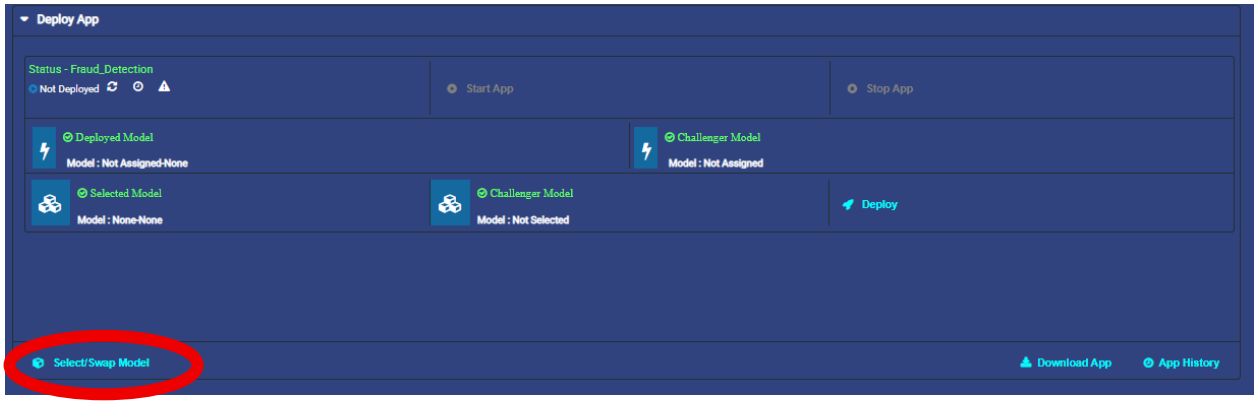
Created a Container

- We navigated to **Manage App > Add Application**, entered a name for the new app, and refreshed the screen to confirm it was added.



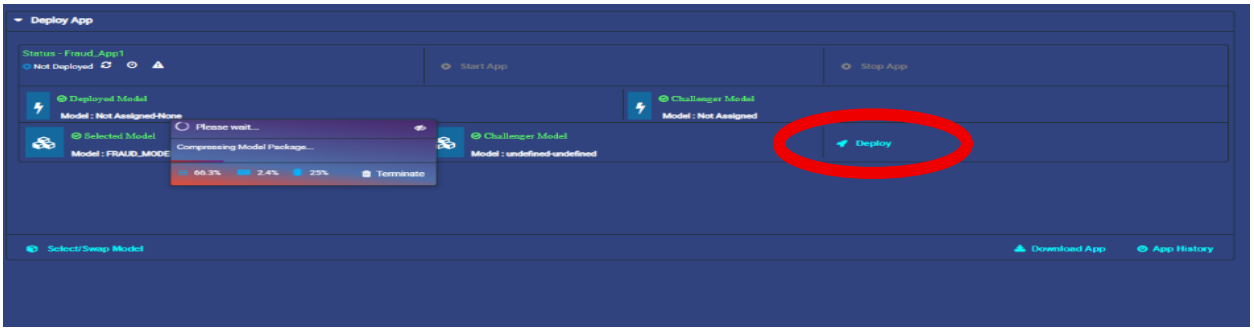
3. Selected the Application

- After the container was created, we clicked on **Select / Swap Model** to begin the deployment process.



4. Deployed the Model

- We selected the appropriate container, chose the model, and initiated deployment.
- We clicked the **Deploy icon** and waited for the deployment to be completed.



5. Returned to Menu

- Once the model was successfully deployed, we returned to the main menu, and select the dashboard tool to proceed with testing.

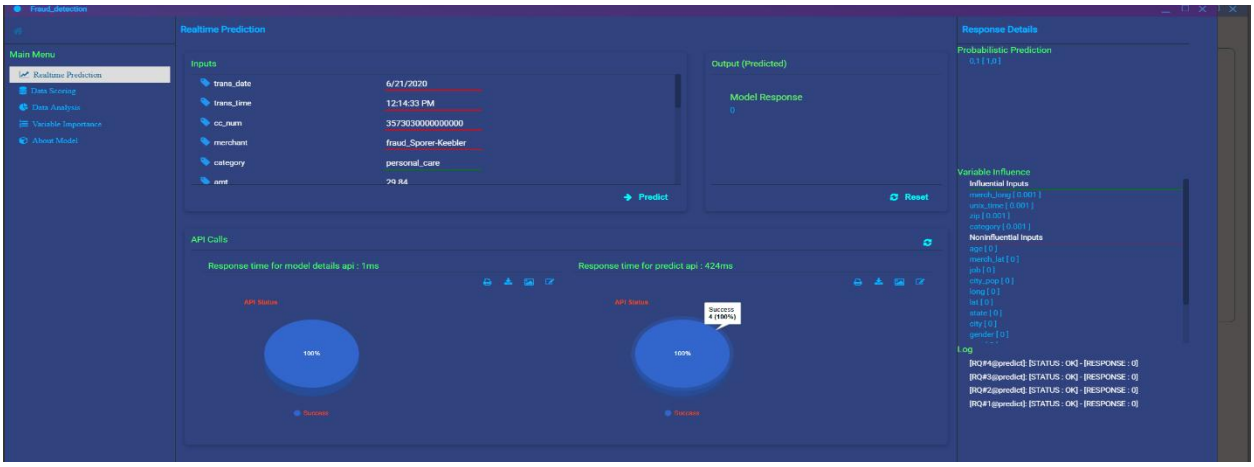
Dashboard & Real-Time Prediction

1. Explored Real-Time Prediction

- We tested the model by entering both new and historical input values and called the API to generate predictions.
- This helped us identify which input features were most influential for the output.

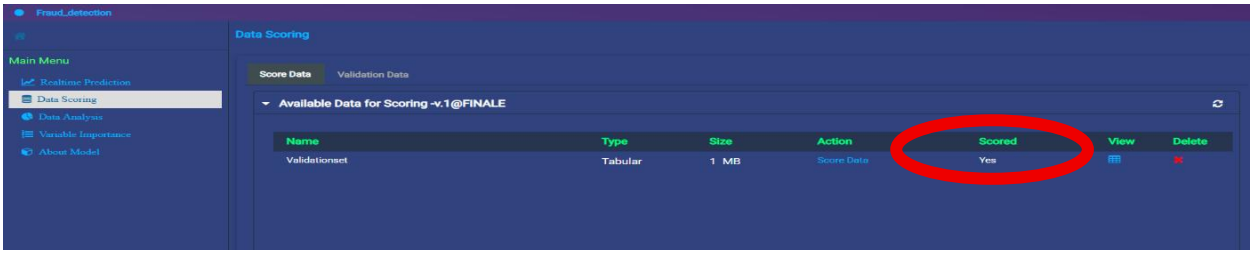
2. Ran Multiple API

- We used the **Predict** option to input different values and ran the prediction 4–5 times to observe how the API responded.



3. Completed Batch Scoring

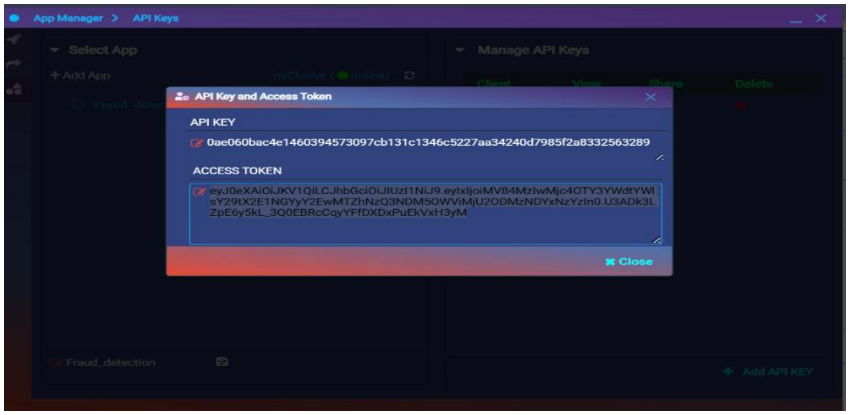
- We went through **Batch Scoring > Validate Data > Convert to Score Data**, and changed the status from **Yes** to **No**.



- We selected our app under **Manage App**, clicked **View**, and accessed the deployment details.

5. Retrieved API Key and HTML

- From the app view, we obtained the **API key** and **HTML snippet** for integration.



6. Final Prediction Test

- We clicked on **Predict Item** to run final tests and confirm the model's output behavior.