**Objective:** Develop a fraud detection model.

**Approach:**

I developed a fraud detection model using the provided dataset and implemented it using a random forest classifier. This was a logistic regression problem, and there are various models that can be used to solve such problems, such as the Naïve Bayes classifier, decision tree classifier, KNN classifier, SVM classifier, random forest classifier, etc. I could have utilized deep neural network models for this task, as they often provide better accuracy compared to traditional machine learning models. However, due to the limitations of my laptop's specifications, I used a machine learning approach. Among the available machine learning models, I chose the **random forest classifier** because it is faster compared to other models and less prone to over fitting. This is because it combines multiple decision trees to predict the output, improving the model's robustness and accuracy.

**Process:**

First, I started with the data reading process. Since the company had already provided the dataset, I imported it using the Pandas library.

Data details:

* transactions\_Data.csv: This excel sheet contains transaction details including amounts, timestamps and merchant details.
* cards\_dat.csv: This excel sheet contains credit and debit card details.
* mcc\_codes.json: This JSON file contains standard classification codes for business types.
* train\_fraud\_labels.json: This JSON file contains labels for transactions which are either fraudulent or legitimate.
* users\_data.csv: This excel sheet contains demographics details about customers along with their account details.

transactions\_data = pd.read\_csv("Data/transactions\_data.csv")

cards\_data = pd.read\_csv("Data/cards\_data.csv")

users\_data = pd.read\_csv("Data/users\_data.csv")

with open("Data/mcc\_codes.json", "r") as f:

    mcc\_codes = json.load(f)

with open("Data/train\_fraud\_labels.json", "r") as f:

    fraud\_labels = json.load(f)

fraud\_labels\_df = pd.DataFrame(fraud\_labels['target'].items(), columns=['id', 'label'])

mcc\_codes\_df = pd.DataFrame(mcc\_codes.items(), columns=['mcc', 'data'])

Next, I started merging the dataframes to create a single final dataframe. I observed the dataframes to check for any null values and found that the transaction\_data dataframe had three columns merchant\_state, zip, and errors with a significant number of null values, so I decided to drop them.

transactions\_data = transactions\_data.drop(columns=['merchant\_state', 'zip', 'errors'])

After handling the null values, I merged the data from all three CSV files and two JSON files into a single dataframe, one by one. To merge the dataframes, there needed to be a common column present in both dataframes. Therefore, I decided to rename id to card\_id in the cards\_data dataframe and id to client\_id in the users\_data dataframe.

cards\_data = cards\_data.rename(columns = {'id' : 'card\_id'})

users\_data = users\_data.rename(columns = {'id' : 'client\_id'})

After renaming columns, I merged the data from all three CSV files and two JSON files into a single dataframe.

merged\_data = pd.merge(transactions\_data, cards\_data, on=['card\_id', 'client\_id'], how='left')

merged\_data = pd.merge(merged\_data, users\_data, on=['client\_id'], how='left')

merged\_data = pd.merge(merged\_data, mcc\_codes\_df, on = 'mcc', how = 'left')

Since some data points were missing in the fraud\_labels\_df, I decided to include only those data points that were present in the fraud\_labels\_df.

Total Data points before merging with fraud\_labels\_df = 13305915

Total Data points after merging with fraud\_labels\_df = 8914963

valid\_ids = fraud\_labels\_df['id'].unique()

filtered\_data = merged\_data[merged\_data['id'].isin(valid\_ids)]

merged\_data = pd.merge(filtered\_data, fraud\_labels\_df, on='id', how='left')

After merging all the datasets, I proceeded to pre-process the dataset. I removed all unnecessary symbols, reformatted the date column, and aggregated the features by grouping transactions based on the user ID.

currency\_columns = ['amount', 'per\_capita\_income', 'yearly\_income', 'total\_debt']

for col in currency\_columns:

    if col in merged\_data.columns:

        merged\_data[col] = merged\_data[col].replace({'\$': '', ',': ''}, regex=True).astype(float)

if 'date' in merged\_data.columns:

    merged\_data['date'] = pd.to\_datetime(merged\_data['date'])

    merged\_data['year'] = merged\_data['date'].dt.year

    merged\_data['month'] = merged\_data['date'].dt.month

    merged\_data['day\_of\_week'] = merged\_data['date'].dt.dayofweek

    merged\_data['hour'] = merged\_data['date'].dt.hour

    merged\_data.drop(columns=['date'], inplace=True)

merged\_data['mean\_transaction\_per\_client'] = merged\_data.groupby('client\_id')['amount'].transform('mean')

merged\_data['total\_transaction\_amount'] = merged\_data.groupby('client\_id')['amount'].transform('sum')

Next, I converted non-numeric data into numeric data using a Label Encoder.

from sklearn.preprocessing import LabelEncoder

categorical\_columns = merged\_data.select\_dtypes(include=['object']).columns

for col in categorical\_columns:

    le = LabelEncoder()

    merged\_data[col] = le.fit\_transform(merged\_data[col])

I then scaled down the integer and float data type columns using Standard Scaler for normalization.

from sklearn.preprocessing import StandardScaler

numerical\_columns = merged\_data.select\_dtypes(include=['float64', 'int64']).columns

scaler = StandardScaler()

merged\_data[numerical\_columns] = scaler.fit\_transform(merged\_data[numerical\_columns])

After completing the pre-processing of the data, I had a fully numeric and scaled dataset. I then split the dataset into training and testing sets using train\_test\_split, with the 'label' column as the dependent variable.

from sklearn.model\_selection import train\_test\_split

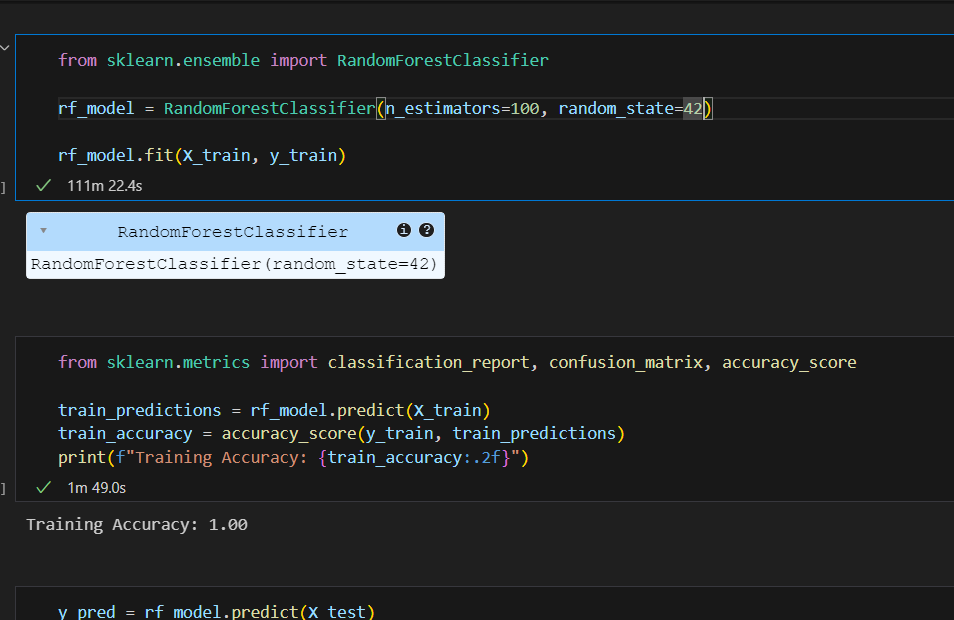
X = merged\_data.drop(columns=['label'])

y = merged\_data['label']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42, stratify=y)

print(f"Training data: {X\_train.shape}, Test data: {X\_test.shape}")

After splitting the dataset, I built a random forest model and checked its accuracy. But I noticed that the model was over fitted due to imbalanced dataset.



To address the issue, I used SMOTE (Synthetic Minority Oversampling Technique) to balance the dataset by generating new data points for the minority class.

I used two strategies to create a balanced dataset:

* 1/1 technique: Created an equal number of data points for category 1.
* 1/2 technique: Created half the number of data points for category 1.

from imblearn.over\_sampling import SMOTE

from collections import Counter

smote = SMOTE(random\_state=42, k\_neighbors=1, sampling\_strategy = 0.5)

X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)

After generating new data points and balancing the dataset, I built a Random Forest classifier model and trained it on the training dataset.

from sklearn.ensemble import RandomForestClassifier

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X\_resampled, y\_resampled)

Finally, I used evaluation metrics such as classification\_report, confusion\_matrix, and accuracy\_score to assess the model's prediction accuracy on the testing dataset.

The model achieved an accuracy of 1.00 (100%) on the training dataset and 1.00 (100%) on the testing dataset.

