Lab 1: Suspicious Transaction Data Analysis and Decision Support System

Suspicious transaction detection helps find possible fraud in financial systems. In this lab, we use the transaction and customer data, joining them through customer ID, and apply feature engineering and data preprocessing to make the data ready for analysis. By looking at location, age group, time of day, and transaction type, we can see which transactions are more likely to be suspicious. This helps in deciding where to focus monitoring and improve security.

In [146…
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [147…
```
customer = pd.read_csv("Customer_Master.csv")
transaction = pd.read_csv("transactions.csv")
```

In [148…
```
customer.head(2)
```

Out[148…

	customer_id	age_group	home_location	credit_score	account_age_years	account_type
0	1	26-35	Palpa	714	13	Savings
1	2	26-35	Kathmandu	607	7	Savings

In [149…
```
transaction.head(2)
```

Out[149…

	transaction_id	customer_id	transaction_date	transaction_type	amount	locati
0	TXN20241124104326	727	11/24/2024 15:29	Inward Remittance	13925.72	Nawalpar
1	TXN20241204130277	539	12/4/2024 5:26	ATM Withdrawal	25037.35	Ka

2 rows × 28 columns

Find intersecting columns

In [150…
```
duplicate_cols = set(customer.columns).intersection(set(transaction.columns))
duplicate_cols
```

Out[150…
```
{'account_age_years',
 'account_type',
 'age_group',
 'avg_monthly_income',
 'credit_score',
 'customer_id',
 'employment_status',
 'home_location',
 'international_activity',
 'mobile_banking_user',
 'risk_score',
 'transaction_frequency'}
```

In [151…
```
duplicate_cols = duplicate_cols - {'customer_id'}
duplicate_cols
```

Out[151…
```
{'account_age_years',
 'account_type',
 'age_group',
 'avg_monthly_income',
 'credit_score',
 'employment_status',
 'home_location',
 'international_activity',
 'mobile_banking_user',
 'risk_score',
 'transaction_frequency'}
```

In [152…
```
transaction = transaction.drop(columns= duplicate_cols)
```

In [153…
```
#check duplicate again
duplicate_cols = set(customer.columns).intersection(set(transaction.columns))
duplicate_cols
```

Out[153…
```
{'customer_id'}
```

merge these datasets

In [154…
```
dataset = customer.merge(transaction, on="customer_id", how="inner")
```

In [155…
```
dataset.head(5)
```

Out[155…

	customer_id	age_group	home_location	credit_score	account_age_years	account_type
0	1	26-35	Palpa	714	13	Savings
1	1	26-35	Palpa	714	13	Savings
2	1	26-35	Palpa	714	13	Savings
3	1	26-35	Palpa	714	13	Savings
4	1	26-35	Palpa	714	13	Savings

5 rows × 33 columns

In [156…

```
#check for null
dataset.isna().sum()
```

Out[156…

```
customer_id                       0
age_group                         0
home_location                     0
credit_score                      0
account_age_years                 0
account_type                      0
avg_monthly_income                0
mobile_banking_user               0
primary_device                14202
primary_os                    14202
primary_browser               14202
avg_transaction_amount            0
transaction_frequency             0
employment_status                 0
preferred_transaction_types       0
international_activity            0
risk_score                        0
transaction_id                    0
transaction_date                  0
transaction_type                  0
amount                            0
location                          0
ip_address                        0
device                        79991
os                            79963
browser                       79967
attempt_sequence                  0
time_of_day                       0
transaction_velocity              0
status                            0
auth_method                       0
amount_deviation                  0
is_suspicious                     0
dtype: int64
```

In [157…
```
dataset.shape
```
Out[157…
```
(103500, 33)
```

Data preprocessing

In [158…
```
#device, os and browser have about 77% of null data so we drop those columns
dataset.drop(columns=["device", "os", "browser", "attempt_sequence"], inplace= True
```
In [159…
```
dataset.duplicated().sum()
```
Out[159…
```
np.int64(0)
```
In [160…
```
dataset.isna().sum()
```
Out[160…
```
customer_id                      0
age_group                        0
home_location                    0
credit_score                     0
account_age_years                0
account_type                     0
avg_monthly_income               0
mobile_banking_user              0
primary_device               14202
primary_os                   14202
primary_browser              14202
avg_transaction_amount           0
transaction_frequency            0
employment_status                0
preferred_transaction_types      0
international_activity           0
risk_score                       0
transaction_id                   0
transaction_date                 0
transaction_type                 0
amount                           0
location                         0
ip_address                       0
time_of_day                      0
transaction_velocity             0
status                           0
auth_method                      0
amount_deviation                 0
is_suspicious                    0
dtype: int64
```
In [161…
```
#now filling values with mode for 3 remaining columns
dataset["primary_device"] = dataset["primary_device"].fillna(dataset["primary_devic
dataset["primary_os"] = dataset["primary_os"].fillna(dataset["primary_os"].mode()[0
dataset["primary_browser"] = dataset["primary_browser"].fillna(dataset["primary_bro
```
In [162…
```
dataset.isna().sum().sum()
```

Out[162…

```
np.int64(0)
```

Comparing different sorts of features and visualization

In [ ]:

```

```

1. Suspicious Transaction Distribution Across Transaction Types

In [163…

```
count_and_rate = (
    dataset.groupby('transaction_type')['is_suspicious']
    .agg(['sum', 'count', 'mean'])
    .sort_values(by='mean', ascending=False)
)

print(count_and_rate)
```

```
                              sum  count      mean
transaction_type
Mobile Banking Transfer      1425   4775  0.298429
ATM Withdrawal                837   4947  0.169193
Wallet Load - eSewa           608   4514  0.134692
QR Payment                    534   4382  0.121862
Mobile Banking Bill Payment   546   4652  0.117369
Branch Deposit                343   4161  0.082432
Electricity Bill Payment      154   3843  0.040073
Cheque Deposit                144   3626  0.039713
Water Bill Payment            146   3822  0.038200
POS Transaction               141   3836  0.036757
Internet Bill Payment         140   3877  0.036110
Outward Remittance            135   3849  0.035074
Hotel Booking Payment         139   4021  0.034569
Airline Ticket Payment        123   3840  0.032031
Interest Credit               119   3722  0.031972
Mobile Recharge               133   4245  0.031331
School Fee Payment            129   4159  0.031017
Inward Remittance             133   4404  0.030200
Wallet Load - IME Pay         111   3687  0.030106
Wallet Load - Khalti          124   4166  0.029765
Insurance Premium Payment     132   4437  0.029750
Loan Payment                  128   4361  0.029351
Cheque Payment                118   4116  0.028669
Branch Withdrawal             111   4042  0.027462
Cable TV Payment              108   4016  0.026892
```

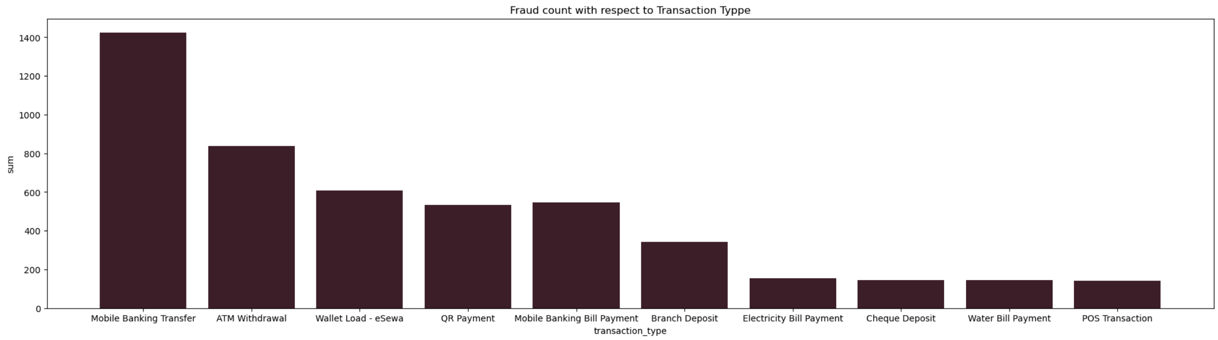
In [164…

```
plt.figure(figsize=(24,6))
sns.barplot(
    data=count_and_rate.head(10),
    x = 'transaction_type',
    y = 'sum',
    color = '#451828'
)
```

```python
plt.title("Fraud count with respect to Transaction Typpe")
plt.plot()
```

Out[164...

```
[]
```



Analysis shows Mobile Banking Transfer has the highest fraud count (1,425 cases), followed by ATM Withdrawal (837 cases) and Wallet Load - eSewa (608 cases).

Solving strategies:

1. Mobile Banking Transfer: Implement mandatory two-factor authentication (2FA) using OTP or biometric verification for all transfers.
   Add transaction velocity limits to block multiple rapid transfers.

2. ATM Withdrawal: Send real-time SMS alerts immediately after each withdrawal. Enable customers to instantly block their card via SMS reply or app if fraud is detected.
   Enforce daily withdrawal limits and chip-and-PIN-only transactions.

3. Wallet Load - eSewa: Implement KYC validation before allowing first-time loads. Add a 24-hour hold on loaded funds for new or suspicious accounts.

In [ ]:

```python

```

2. Suspicious Transaction Patterns by Time of Day

In [165...

```python
fraud_by_time = (
    dataset.groupby('time_of_day')['is_suspicious']
    .agg(['sum', 'mean'])
    .sort_values(by = 'sum', ascending=False)
    .reset_index()
)

print(fraud_by_time)
```

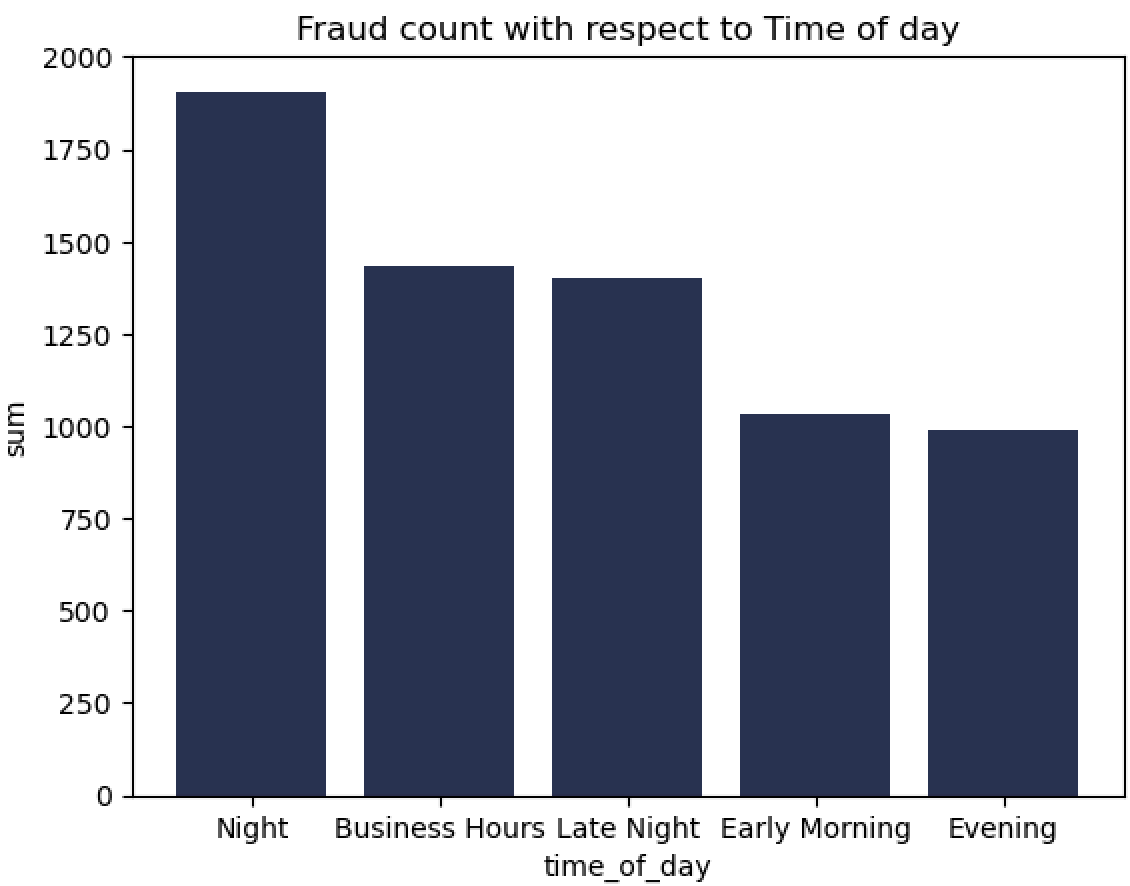
```
      time_of_day   sum      mean
0           Night  1907  0.137115
1  Business Hours  1434  0.042668
2      Late Night  1401  0.054545
3   Early Morning  1031  0.077864
4         Evening   988  0.057920
```

In [166…

```
sns.barplot(
    data = fraud_by_time,
    x = 'time_of_day',
    y = 'sum',
    color= '#252d59'
)
plt.title("Fraud count with respect to Time of day")
plt.plot()
```

Out[166…

```
[]
```



Analysis shows Night has the highest fraud count (1,907 cases, 13.7% rate), followed by Business Hours (1,434 cases, 4.3% rate) and Late Night (1,401 cases, 5.5% rate). Early Morning has 1,031 cases (7.8% rate), and Evening has 988 cases (5.9% rate).

1. Night:

- Implement stricter authentication for all transactions between 11 PM and 5 AM.
- Require step-up authentication (biometric + OTP) for any transaction above a low threshold.
- Add mandatory 5-minute delay with SMS alert before processing high-value night transactions, allowing customers to cancel if unauthorized.

2. Late Night:

- Enable real-time fraud scoring that flags unusual patterns (e.g., user who never transacts late suddenly does).

3. Suspicious Transaction Patterns by Location

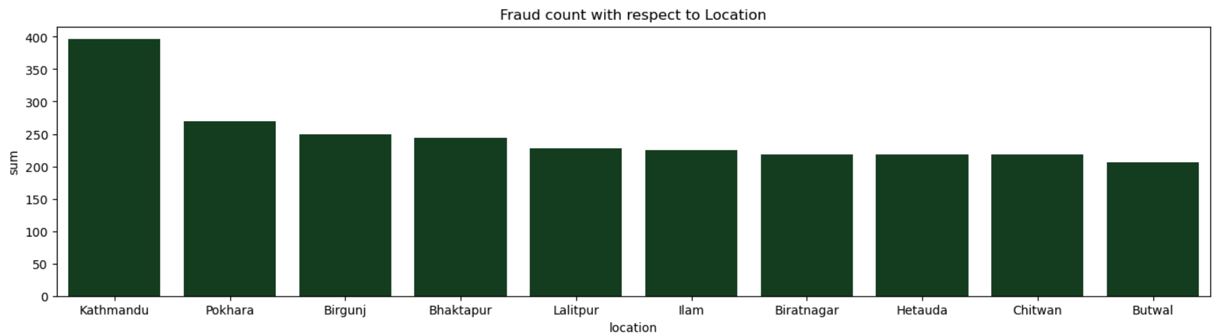
In [208…

```
#location vs fraud
fraud_by_location = (
    dataset.groupby('location')['is_suspicious']
    .agg(['sum', 'mean'])
    .sort_values(by = 'sum', ascending= False)
)

print(fraud_by_location.head(20))
```

```
          sum      mean
location
30        396  0.020643
41        270  0.032032
7         249  0.057546
4         244  0.051597
32        228  0.051213
22        225  0.069103
6         219  0.045540
20        218  0.062662
10        218  0.069338
9         206  0.076637
23        204  0.080473
15        203  0.058233
24        201  0.076021
38        187  0.074118
14        185  0.088900
48        183  0.197198
45        170  0.184783
27        167  0.170234
17        167  0.178419
2         166  0.180043
```

In [168…

```
#figure for this
plt.figure(figsize=(17,4))
sns.barplot(data=fraud_by_location.head(10),
            x= 'location',
            y = 'sum',
            color='#0f451d')
plt.title("Fraud count with respect to Location")
plt.show()
```



Major cities like Kathmandu and Pokhara show high suspicious counts mainly because they have huge transaction volumes, not high fraud risk. Mid-tier cities have moderately higher fraud rates, suggesting weaker controls. Some districts show abnormally high rates, and foreign locations showing 100 percent fraud clearly indicate data or labeling errors.

Potiential Solutions

- Increase monitoring in mid-risk cities (Birgunj, Bhaktapur, Ilam, Chitwan etc.) with stricter verification at merchants and digital payment gateways.
- Improve identity and SIM/account verification in rural districts where high rates suggest weak KYC processes.
- Collaborate with local authorities and banks in high-risk districts to educate merchants and enforce proper documentation.

In [ ]:

4. Suspicious Transaction Patterns by Age Group

In [169…

```
dataset['age_group'] = pd.Categorical(
    dataset['age_group'],
    ordered=True,
    categories=['18-25', '26-35', '36-45', '46-55', '56+']
)

dataset.groupby('age_group')['is_suspicious'].sum()
```

```
C:\Users\Sandesh Khatiwada\AppData\Local\Temp\ipykernel_14668\1775792591.py:7: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  dataset.groupby('age_group')['is_suspicious'].sum()
```

Out[169…

```
age_group
18-25    1070
26-35    2032
36-45    1526
46-55    1104
56+         0
Name: is_suspicious, dtype: int64
```

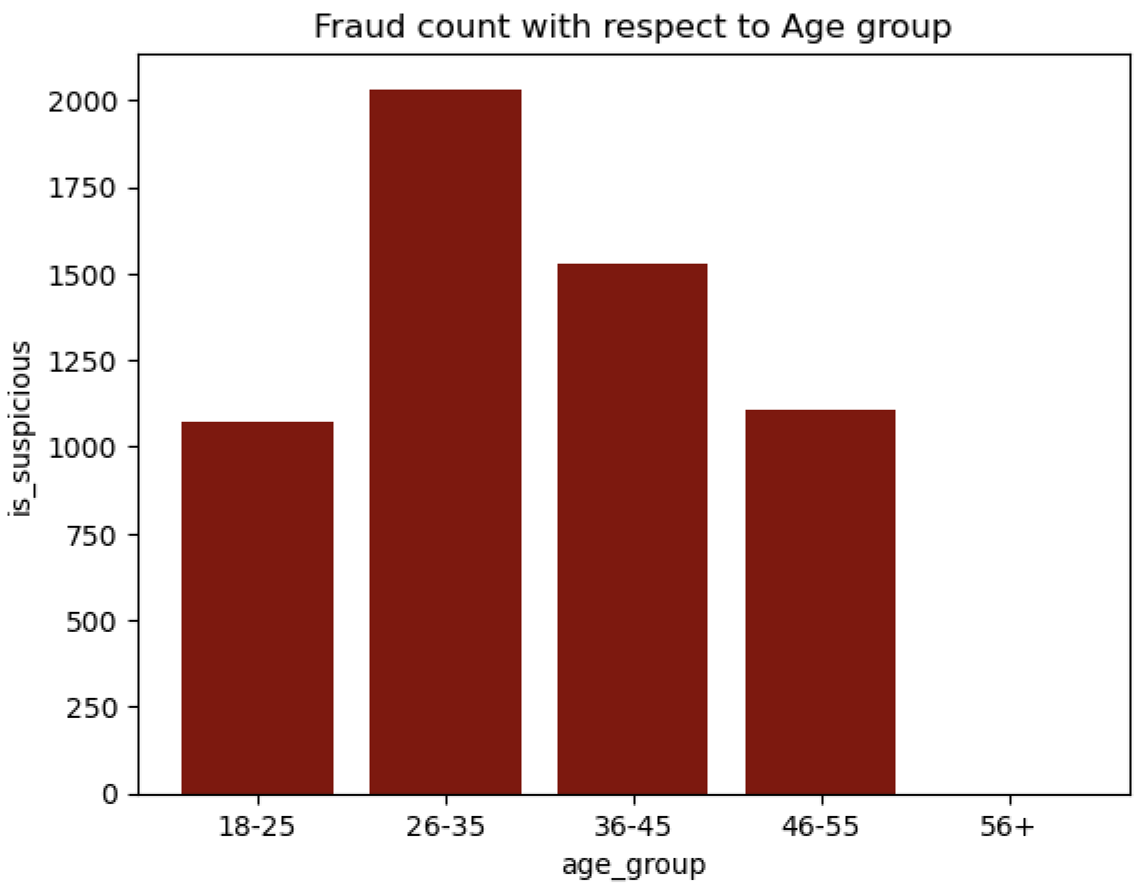
In [170…

```
#make graph for this
# aggregate first
age_fraud = dataset.groupby('age_group', as_index=False)['is_suspicious'].sum()

# plot
sns.barplot(data=age_fraud, x='age_group', y='is_suspicious', color='#8f0c00')
plt.title("Fraud count with respect to Age group")
plt.plot()
```

```
C:\Users\Sandesh Khatiwada\AppData\Local\Temp\ipykernel_14668\2547066393.py:3: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  age_fraud = dataset.groupby('age_group', as_index=False)['is_suspicious'].sum()
```

Out[170…

```
[]
```



Fraud counts peak in the 26–35 and 36–45 age groups, likely because these groups perform the highest number of digital transactions.
The 18–25 and 46–55 groups show moderate suspicious activity.
The 56+ group showing zero fraud is unrealistic and indicates missing data or extremely low digital usage.

Potential Soltions:

- Target high-activity age groups (26–45) with stronger verification, like stricter login checks and spending-pattern monitoring.

- Provide financial literacy and fraud-awareness campaigns specifically for 18–25 users who are easier targets.

In [ ]:

```

```

5. Suspicious Transactions Across Multiple Features('transaction_type', 'time_of_day', 'location', 'age_group')

In [171…

```
combined = (
    dataset.groupby(['transaction_type', 'time_of_day', 'location', 'age_group'])['
    .agg(['sum', 'count'])
    .sort_values(by='sum', ascending=False)
)

print(combined)
```

```
                                                          sum  count
transaction_type        time_of_day    location  age_group
Mobile Banking Transfer Night          Butwal    26-35       13     18
                                       Syangja   26-35       13     14
Wallet Load - eSewa     Night          Lalitpur  26-35       12     18
ATM Withdrawal          Night          Birgunj   36-45       12     13
Mobile Banking Transfer Night          Dhangadhi 26-35       10     12
...                                                         ...    ...
Airline Ticket Payment  Night          London    26-35        0      0
                                                 36-45        0      0
ATM Withdrawal          Business Hours Butwal    56+          0      0
                                       Chitwan   18-25        0      6
Water Bill Payment      Night          Sindhuli  36-45        0      0

[32500 rows x 2 columns]
```

```
C:\Users\Sandesh Khatiwada\AppData\Local\Temp\ipykernel_14668\3887887973.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
  dataset.groupby(['transaction_type', 'time_of_day', 'location', 'age_group'])['is_suspicious']
```

In [172…

```
top5 = combined.sort_values(by='sum', ascending=False).head(5)
top5
```

Out[172…

transaction_type	time_of_day	location	age_group	sum	count
Mobile Banking Transfer	**Night**	**Butwal**	**26-35**	13	18
		Syangja	**26-35**	13	14
Wallet Load - eSewa	**Night**	**Lalitpur**	**26-35**	12	18
ATM Withdrawal	**Night**	**Birgunj**	**36-45**	12	13
Mobile Banking Transfer	**Night**	**Dhangadhi**	**26-35**	10	12

In [173…

```
top5 = combined.head(5).reset_index()
top5
```

Out[173…

	transaction_type	time_of_day	location	age_group	sum	count
0	Mobile Banking Transfer	Night	Butwal	26-35	13	18
1	Mobile Banking Transfer	Night	Syangja	26-35	13	14
2	Wallet Load - eSewa	Night	Lalitpur	26-35	12	18
3	ATM Withdrawal	Night	Birgunj	36-45	12	13
4	Mobile Banking Transfer	Night	Dhangadhi	26-35	10	12

In [174…

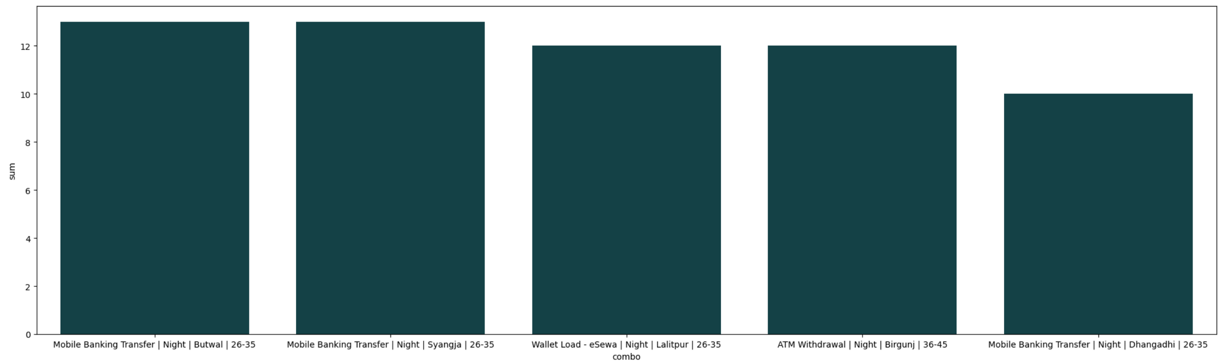
```
# convert everything to string before concatenation
top5 = top5.astype({
    "transaction_type": "string",
    "time_of_day": "string",
    "location": "string",
    "age_group": "string"
})

top5["combo"] = (
    top5["transaction_type"] + " | " +
    top5["time_of_day"] + " | " +
    top5["location"] + " | " +
    top5["age_group"]
)
```

In [175…

```
plt.figure(figsize=(25,7))
sns.barplot(data=top5, x="combo", y="sum", color= "#0f4a4f")
# plt.xticks(rotation=45)
```

Out[175…

```
<Axes: xlabel='combo', ylabel='sum'>
```



The top suspicious combinations all happen at night, mostly through mobile banking transfers, wallet loads, and ATM withdrawals.
The high-risk age group is consistently 26–35, and the risky locations include Butwal, Syangja, Lalitpur, Birgunj, and Dhangadhi.
This pattern shows that fraudsters exploit late-night hours and high-activity digital users.

Potiential Solutions:

- Increase nighttime transaction scrutiny, including stricter OTP/MFA and velocity checks.
- Deploy ATM geo-tracking and withdrawal anomaly alerts in cities like Birgunj.
- Coordinate with local banks in high-risk cities to tighten verification and monitor unusual night activity.

Conclusion from Analysis

Fraud is highest in the 26–45 age group, mostly through mobile banking, wallet loads, and ATM withdrawals, especially at night.
Major cities have high counts due to volume, while some smaller cities show higher fraud rates per transaction.
Nighttime, high-risk locations, and active age groups should be prioritized for monitoring and stronger authentication.

In [201…

```
# view columns data to define type of encoding to perform
# for col in dataset.columns:
#     print("unique values of dataset[",col, "]: ", dataset[col].unique(), "\n")
```

Normalization

In [177…

```
#normalizing avg_monthly_income , amount, credit_score , avg_transaction_amount
from sklearn.preprocessing import MinMaxScaler
ms_avg_monthly_income = MinMaxScaler(feature_range=(0, 1))
ms_amount = MinMaxScaler(feature_range=(0, 1))
ms_credit_score = MinMaxScaler(feature_range=(0, 1))
ms_avg_transaction_amount = MinMaxScaler(feature_range=(0, 1))
ms_amount_deviation = MinMaxScaler(feature_range=(0, 1))
```

In [178…
```
dataset["avg_monthly_income"] = ms_avg_monthly_income.fit_transform(dataset[["avg_m
dataset["amount"] = ms_amount.fit_transform(dataset[["amount"]])
dataset["credit_score"] = ms_credit_score.fit_transform(dataset[["credit_score"]])
dataset["avg_transaction_amount"] = ms_avg_transaction_amount.fit_transform(dataset
dataset["amount_deviation"] = ms_amount_deviation.fit_transform(dataset[["amount_de
```

In [ ]:

Label Encoding

In [179…
```
#label encoding age_group, home_location ,account_type , mobile_banking_user ,  pri
# preferred_transaction_types , location , time_of_day , status , auth_method , is_
```

In [180…
```
from sklearn.preprocessing import LabelEncoder
le_age_group = LabelEncoder()
le_home_location = LabelEncoder()
le_account_type = LabelEncoder()
le_mobile_banking_user = LabelEncoder()
le_primary_device = LabelEncoder()
le_primary_os = LabelEncoder()
le_primary_browser = LabelEncoder()
le_employment_status = LabelEncoder()
le_preferred_transaction_types = LabelEncoder()
le_location = LabelEncoder()
le_time_of_day = LabelEncoder()
le_status = LabelEncoder()
le_auth_method = LabelEncoder()
le_is_suspicious = LabelEncoder()
le_transaction_type = LabelEncoder()
```

In [181…
```
dataset["age_group"] = le_age_group.fit_transform(dataset["age_group"])
dataset["home_location"] = le_home_location.fit_transform(dataset["home_location"])
dataset["account_type"] = le_account_type.fit_transform(dataset["account_type"])
dataset["mobile_banking_user"] = le_mobile_banking_user.fit_transform(dataset["mobi
dataset["primary_device"] = le_primary_device.fit_transform(dataset["primary_device
dataset["primary_os"] = le_primary_os.fit_transform(dataset["primary_os"])
dataset["primary_browser"] = le_primary_browser.fit_transform(dataset["primary_brow
dataset["employment_status"] = le_employment_status.fit_transform(dataset["employme
dataset["preferred_transaction_types"] = le_preferred_transaction_types.fit_transfo
dataset["location"] = le_location.fit_transform(dataset["location"])
dataset["time_of_day"] = le_time_of_day.fit_transform(dataset["time_of_day"])
dataset["status"] = le_status.fit_transform(dataset["status"])
dataset["auth_method"] = le_auth_method.fit_transform(dataset["auth_method"])
dataset["is_suspicious"] = le_is_suspicious.fit_transform(dataset["is_suspicious"])
dataset["transaction_type"] = le_transaction_type.fit_transform(dataset["transactio
```

In [182…
```
label_maps = {
    "age_group": dict(zip(le_age_group.classes_, le_age_group.transform(le_age_grou
    "home_location": dict(zip(le_home_location.classes_, le_home_location.transform
    "account_type": dict(zip(le_account_type.classes_, le_account_type.transform(le
    "mobile_banking_user": dict(zip(le_mobile_banking_user.classes_, le_mobile_bank
    "primary_device": dict(zip(le_primary_device.classes_, le_primary_device.transf
```

```
    "primary_os": dict(zip(le_primary_os.classes_, le_primary_os.transform(le_prima
    "primary_browser": dict(zip(le_primary_browser.classes_, le_primary_browser.tra
    "employment_status": dict(zip(le_employment_status.classes_, le_employment_stat
    "preferred_transaction_types": dict(zip(le_preferred_transaction_types.classes_
    "location": dict(zip(le_location.classes_, le_location.transform(le_location.cl
    "time_of_day": dict(zip(le_time_of_day.classes_, le_time_of_day.transform(le_ti
    "status": dict(zip(le_status.classes_, le_status.transform(le_status.classes_))
    "auth_method": dict(zip(le_auth_method.classes_, le_auth_method.transform(le_au
    "is_suspicious": dict(zip(le_is_suspicious.classes_, le_is_suspicious.transform
    "transaction_type": dict(zip(le_transaction_type.classes_, le_transaction_type.
}
# label_maps
```

In [183…

```
dataset.head()
```

Out[183…

	customer_id	age_group	home_location	credit_score	account_age_years	account_type
0	1	1	21	0.745794	13	3
1	1	1	21	0.745794	13	3
2	1	1	21	0.745794	13	3
3	1	1	21	0.745794	13	3
4	1	1	21	0.745794	13	3

5 rows × 29 columns

Validate data types and properly preprocess the features before applying the algorithm.

In [184…

```
dataset.dtypes
```

Out[184…
```
customer_id                        int64
age_group                          int64
home_location                      int64
credit_score                     float64
account_age_years                  int64
account_type                       int64
avg_monthly_income               float64
mobile_banking_user                int64
primary_device                     int64
primary_os                         int64
primary_browser                    int64
avg_transaction_amount           float64
transaction_frequency              int64
employment_status                  int64
preferred_transaction_types        int64
international_activity              bool
risk_score                         int64
transaction_id                    object
transaction_date                  object
transaction_type                   int64
amount                           float64
location                           int64
ip_address                        object
time_of_day                        int64
transaction_velocity               int64
status                             int64
auth_method                        int64
amount_deviation                 float64
is_suspicious                      int64
dtype: object
```

In [185…
```
dataset['transaction_id'].nunique()
```

Out[185…
```
103492
```

In [186…
```
dataset['transaction_date'].nunique()
```

Out[186…
```
98133
```

In [187…
```
dataset.drop(columns=['ip_address', 'transaction_id', 'transaction_date'], inplace=
```

In [188…
```
#now for algorithm seperate the data
X = dataset.drop(columns=['is_suspicious'])
y = dataset['is_suspicious']
X.ndim
```

Out[188…
```
2
```

Algorithm (Linear and Logistic Regression)

In [189…
```
from sklearn.model_selection import train_test_split
```

In [190...
```
# help(train_test_split)
```

In [191...
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_st
```

In [192...
```
from sklearn.linear_model import LogisticRegression, LinearRegression
```

In [193...
```
lr = LinearRegression()
lg = LogisticRegression()
```

In [194...
```
lr.fit(X_train, y_train)
```

Out[194...
```
▾ LinearRegression
▶ Parameters
```

In [195...
```
lr.score(X_test, y_test)
```

Out[195...
```
0.6133427224337857
```

In [ ]:
```
```

In [196...
```
lg.fit(X_train, y_train)
```

```
D:\Installations\Miniconda\envs\dsml\Lib\site-packages\sklearn\linear_model\_logisti
c.py:473: ConvergenceWarning: lbfgs failed to converge after 100 iteration(s) (statu
s=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=100).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[196...
```
▾ LogisticRegression
▶ Parameters
```

In [197...
```
lg.score(X_test, y_test)
```

Out[197...
```
0.9717171717171718
```

In [ ]:
```
```

Confusion Matrix

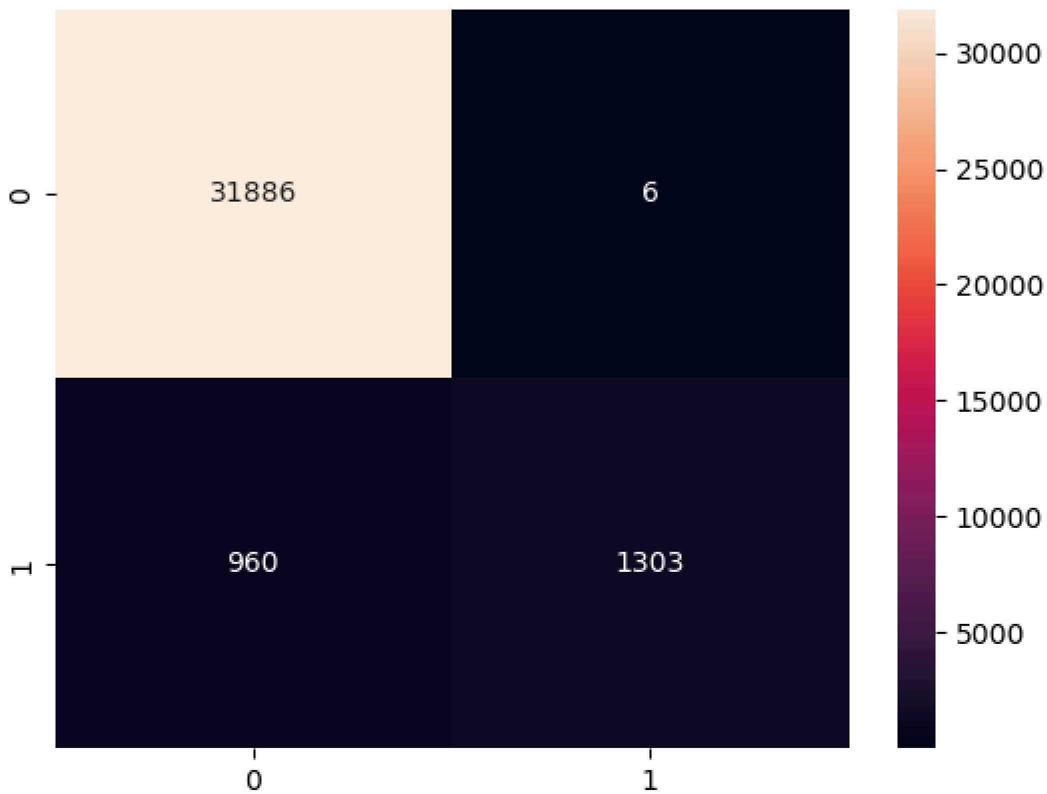
In [198...
```
y_pred = lg.predict(X_test)
```

In [199…
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
confusion_logistic  = confusion_matrix(y_test, y_pred)
confusion_logistic
```

Out[199…
```
array([[31886,     6],
       [  960,  1303]])
```

In [200…
```
sns.heatmap(confusion_logistic, annot=True, fmt="d")
plt.plot()
```

Out[200…
```
[]
```



In [203…
```
!jupyter nbconvert --to webpdf "D:/Github/Data-Science-And-Machine-Learning-Course/
```

```
[NbConvertApp] Converting notebook D:/Github/Data-Science-And-Machine-Learning-Cours
e/Decision Support System/Suspicious_Transaction_Detection_Using_Customer_Transactio
n_Data_Integration.ipynb to webpdf
[NbConvertApp] WARNING | Alternative text is missing on 6 image(s).
[NbConvertApp] Building PDF
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 430808 bytes to D:\Github\Data-Science-And-Machine-Learning-C
ourse\Decision Support System\1_Suspicious_Transaction_Detection.pdf
```

In [ ]: