Transaction Fraud Insight

Data Analysis and Visualization using

MS SQL SERVER, PYTHON, POWER BI

Project Workflow

1. Project Objective, Tools and Technologies
2. Dataset Overview
3. Data Cleaning & PreProcessing (Python)
4. SQL Query Development (MS SQL Server)
5. Power BI Dashboards and Visualizations
6. Conclusion

Project Objective

To analyze financial transaction data and identify potential fraud using SQL queries, data cleaning with Python, and visualizations in Power BI.

Tools and Technologies:

MS SQL SERVER **PYTHON** **POWER BI**

Dataset Overview

The dataset is sourced from Kaggle. The dataset includes multiple tables containing financial transaction records, customer information, account activity, and patterns of fraudulent transactions.

Transaction Data

- transaction_records.csv: Contains transaction records with details such as transaction ID, date, amount, and customer ID.
- transaction_metadata.csv: Contains additional metadata for each transaction.

- Customer Profiles
- customer_data.csv: Includes customer profiles with information such as name, age, address, and contact details.
- account_activity.csv: Provides details of customer account activity, including account balance, transaction history, and account status.

Dataset Overview

- Fraudulent Patterns
  - fraud_indicators.csv: Contains indicators of fraudulent patterns and suspicious activities.
  - suspicious_activity.csv: Provides specific details of transactions flagged as suspicious.

  - Transaction Amounts
  - amount_data.csv: Includes transaction amounts for each transaction.
  - anomaly_scores.csv: Provides anomaly scores for transaction amounts, indicating potential fraudulence.

  - Merchant Information
  - merchant_data.csv: Contains information about merchants involved in transactions.
  - transaction_category_labels.csv: Provides category labels for different transaction types.

Data Cleaning and PreProcessing (Python)

```
CUSTOMER DATA
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:17: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions1 = pd.read_sql(query1, conn)
  CustomerID           Name  Age       Address
0       1001  Customer 1001   54  Address 1001
1       1002  Customer 1002   35  Address 1002
2       1003  Customer 1003   40  Address 1003
3       1004  Customer 1004   30  Address 1004
4       1005  Customer 1005   46  Address 1005

ACCOUNT ACTIVITY
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:22: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions2 = pd.read_sql(query2, conn)
  CustomerID  AccountBalance   LastLogin
0       1001       9507.2721  2022-01-01
1       1002       7408.7045  2022-01-02
2       1003       1715.3220  2022-01-03
3       1004       3101.5091  2022-01-04
4       1005       5405.7669  2022-01-05
```

*Results mentioned here are the first 5 records.

Data Cleaning and PreProcessing (Python)

```
FRAUD INDICATORS
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:27: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions3 = pd.read_sql(query3, conn)
  TransactionID  FraudIndicator
0             1               0
1             2               0
2             3               0
3             4               0
4             5               0

SUSPICIOUS ACTIVITY
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:32: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions4 = pd.read_sql(query4, conn)
  CustomerID  SuspiciousFlag
0       1001               0
1       1002               0
2       1003               0
3       1004               0
4       1005               0
```

Data Cleaning and PreProcessing (Python)

```
MERCHANT DATA
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:37: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions5 = pd.read_sql(query5, conn)
  MerchantID   MerchantName       Location
0       2001  Merchant 2001  Location 2001
1       2002  Merchant 2002  Location 2002
2       2003  Merchant 2003  Location 2003
3       2004  Merchant 2004  Location 2004
4       2005  Merchant 2005  Location 2005

TRANSACTION CATEGORY LABELS
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:42: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions6 = pd.read_sql(query6, conn)
  TransactionID Category
0             1    Other
1             2   Online
2             3   Travel
3             4   Travel
4             5    Other
```

Data Cleaning and PreProcessing (Python)

```
AMOUNT DATA
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:47: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions7 = pd.read_sql(query7, conn)
   TransactionID  TransactionAmount
0              1            79.4136
1              2            12.0531
2              3            33.3104
3              4            46.1211
4              5            54.0516

ANOMALY SCORES
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:52: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions8 = pd.read_sql(query8, conn)
   TransactionID  AnomalyScore
0              1      0.686699
1              2      0.081749
2              3      0.023857
3              4      0.876994
4              5      0.034059
```

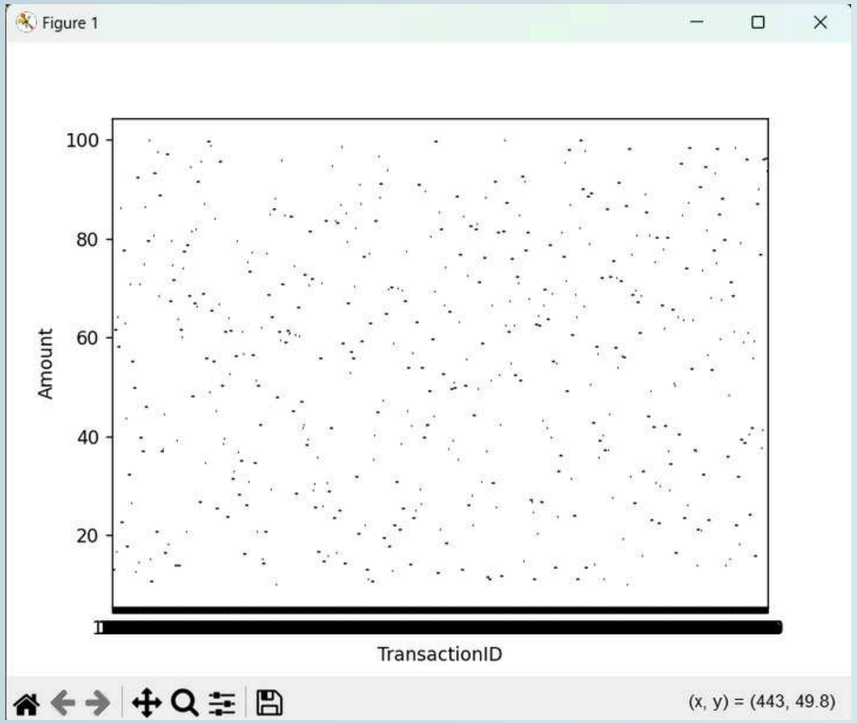
Data Cleaning and PreProcessing (Python)

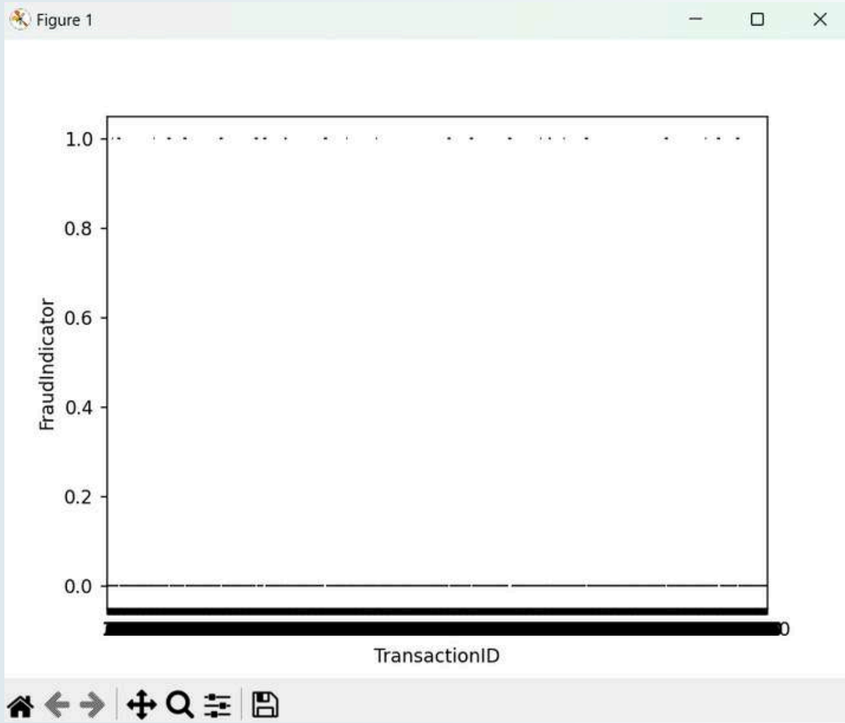
```
TRANSACTION METADATA
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:57: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions9 = pd.read_sql(query9, conn)
   TransactionID                  Timestamp MerchantID
0              1 2022-01-01 00:00:00.0000000       2701
1              2 2022-01-01 01:00:00.0000000       2070
2              3 2022-01-01 02:00:00.0000000       2238
3              4 2022-01-01 03:00:00.0000000       2879
4              5 2022-01-01 04:00:00.0000000       2966

TRANSACTION RECORDS
d:\projects\FinancialTransaction\Python\PythonAndSQLIntegration.py:62: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
  transactions10 = pd.read_sql(query10, conn)
   TransactionID   Amount CustomerID
0              1  55.5303       1952
1             10  17.2454       1574
2            100  13.0972       1514
3           1000  89.9156       1503
4            101  58.9239       1023
```

#Data Cleaning
Drop duplicates
transactions10.drop_duplicates (inplace=True)
# Handle missing values
transactions10.fillna({'column_ name': 0}, inplace=True)

Exploratory Data Analysis using matplotlib and seaborn libraries.





SQL Query Development (MS SQL Server)

List all transaction details (transaction ID, date, amount) along with the customer name and ID.

```
--List all transaction details (transaction ID, date, amount) along with the customer name and ID.
use FinancialTransaction;
SELECT T.TransactionID, A.LastLogin, T.Amount, C.Name, T.CustomerID
FROM     transaction_records AS T INNER JOIN
                  customer_data AS C ON T.CustomerID = C.CustomerID INNER JOIN
                  account_activity AS A ON C.CustomerID = A.CustomerID
```

Results | Messages

	TransactionID	LastLogin	amount	name	customerID
478	528	2023-05-16	70.0081	Customer 1501	1501
479	529	2024-04-07	89.7173	Customer 1828	1828
480	53	2022-12-11	42.084	Customer 1345	1345
481	530	2022-08-19	27.2545	Customer 1231	1231
482	531	2024-09-04	42.2782	Customer 1978	1978
483	532	2023-05-11	57.1367	Customer 1496	1496
484	533	2023-09-21	81.4999	Customer 1629	1629
485	534	2023-04-22	49.1099	Customer 1477	1477
486	535	2023-03-01	42.022	Customer 1425	1425
487	536	2024-04-16	80.5191	Customer 1837	1837
488	537	2024-08-11	95.2233	Customer 1954	1954
489	538	2022-05-26	77.116	Customer 1146	1146
490	539	2023-08-14	59.587	Customer 1591	1591
491	54	2023-11-11	87.7761	Customer 1680	1680
492	540	2024-02-03	44.1268	Customer 1764	1764
493	541	2023-05-25	69.2367	Customer 1510	1510
494	542	2023-01-17	44.0946	Customer 1382	1382
495	543	2023-03-15	99.7754	Customer 1439	1439
496	544	2024-08-12	52.9912	Customer 1955	1955
497	545	2022-01-16	68.3657	Customer 1016	1016
498	546	2023-04-05	12.3011	Customer 1460	1460
499	547	2022-03-14	98.6464	Customer 1073	1073
500	548	2024-03-24	85.3141	Customer 1814	1814

Query executed successfully.

SQL Query Development (MS SQL Server)

Find the total number of transactions made by each customer.

```sql
--Find the total number of transactions made by each customer.
SELECT CustomerID, COUNT(*) AS NoOfTransactions
FROM     transaction_records
GROUP BY CustomerID
```

Results | Messages

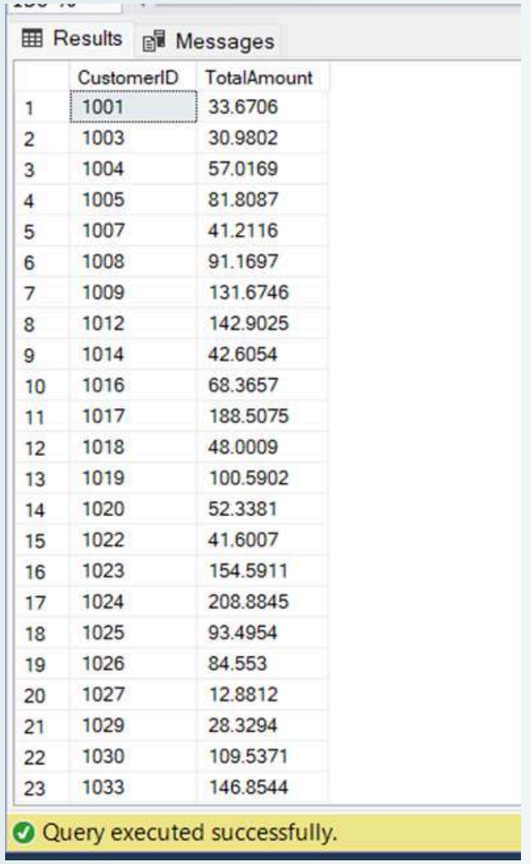
	CustomerID	NoOfTransactions
1	1001	1
2	1003	1
3	1004	2
4	1005	1
5	1007	1
6	1008	1
7	1009	2
8	1012	2
9	1014	1
10	1016	1
11	1017	3
12	1018	1
13	1019	2
14	1020	1
15	1022	1
16	1023	2
17	1024	4
18	1025	1
19	1026	2
20	1027	1
21	1029	1
22	1030	2
23	1033	2

Query executed successfully.

SQL Query Development (MS SQL Server)

Get the total transaction amount for each customer

```sql
--Get the total transaction amount for each customer
SELECT CustomerID, SUM(Amount) AS TotalAmount
FROM     transaction_records
GROUP BY CustomerID
```



Results | Messages

	CustomerID	TotalAmount
1	1001	33.6706
2	1003	30.9802
3	1004	57.0169
4	1005	81.8087
5	1007	41.2116
6	1008	91.1697
7	1009	131.6746
8	1012	142.9025
9	1014	42.6054
10	1016	68.3657
11	1017	188.5075
12	1018	48.0009
13	1019	100.5902
14	1020	52.3381
15	1022	41.6007
16	1023	154.5911
17	1024	208.8845
18	1025	93.4954
19	1026	84.553
20	1027	12.8812
21	1029	28.3294
22	1030	109.5371
23	1033	146.8544

Query executed successfully.

SQL Query Development (MS SQL Server)

Find all transactions above a specific amount threshold

```sql
--Find all transactions above a specific amount threshold
SELECT TransactionID, Amount
FROM     transaction_records
WHERE  (Amount > 10 and Amount <30);
```

Results | Messages

	TransactionID	Amount
1	10	17.2454
2	100	13.0972
3	103	12.4586
4	104	16.651
5	106	13.9151
6	111	22.586
7	119	17.7299
8	124	26.5298
9	130	12.4734
10	135	14.5645
11	139	12.1846
12	147	19.6885
13	149	10.8638
14	150	15.2401
15	152	10.6421
16	16	20.735
17	166	16.5604
18	170	27.4599
19	171	16.4688
20	175	13.5922
21	176	18.043
22	18	27.0549
23	186	13.9461

Query executed successfully.

SQL Query Development (MS SQL Server)

List the top 5 transactions with the highest amounts.

```sql
--List the top 5 transactions with the highest amounts.
SELECT TOP (5) TransactionID, Amount
FROM     transaction_records
ORDER BY Amount DESC;
```

Results | Messages

	TransactionID	Amount
1	742	99.8874
2	638	99.8793
3	15	99.8326
4	543	99.7754
5	230	99.7192

SQL Query Development (MS SQL Server)

List details of customers whose transactions were flagged as suspicious.

```sql
--List details of customers whose transactions were flagged as suspicious.
SELECT C.CustomerID, C.Name, C.Age, C.Address, T.TransactionID, S.SuspiciousFlag
FROM     suspicious_activity AS S INNER JOIN
                  customer_data AS C ON C.CustomerID = S.CustomerID INNER JOIN
                  transaction_records AS T ON T.CustomerID = S.CustomerID
WHERE  (S.SuspiciousFlag = 1);
```

Results | Messages

	CustomerID	Name	Age	Address	TransactionID	SuspiciousFlag
1	1330	Customer 1330	63	Address 1330	123	1
2	1365	Customer 1365	49	Address 1365	170	1
3	1599	Customer 1599	18	Address 1599	198	1
4	1095	Customer 1095	27	Address 1095	207	1
5	1387	Customer 1387	31	Address 1387	218	1
6	1678	Customer 1678	33	Address 1678	225	1
7	1395	Customer 1395	32	Address 1395	229	1
8	1432	Customer 1432	36	Address 1432	271	1
9	1770	Customer 1770	36	Address 1770	295	1
10	1678	Customer 1678	33	Address 1678	296	1
11	1157	Customer 1157	60	Address 1157	313	1
12	1157	Customer 1157	60	Address 1157	346	1
13	1713	Customer 1713	57	Address 1713	431	1
14	1330	Customer 1330	63	Address 1330	442	1
15	1644	Customer 1644	51	Address 1644	45	1
16	1573	Customer 1573	35	Address 1573	482	1
17	1507	Customer 1507	39	Address 1507	553	1
18	1573	Customer 1573	35	Address 1573	602	1
19	1770	Customer 1770	36	Address 1770	613	1
20	1758	Customer 1758	38	Address 1758	625	1
21	1770	Customer 1770	36	Address 1770	694	1
22	1630	Customer 1630	42	Address 1630	739	1
23	1678	Customer 1678	33	Address 1678	781	1

Query executed successfully.

SQL Query Development (MS SQL Server)

Find transactions with anomaly scores above a threshold (0.9)

```sql
--Find transactions with anomaly scores above a threshold (e.g., 0.9)

SELECT A.transactionid, A.anomalyscore, T.amount
FROM anomaly_scores A
JOIN transaction_records T ON A.transactionid = T.transactionid
WHERE A.anomalyscore > 0.9;
```

Results | Messages

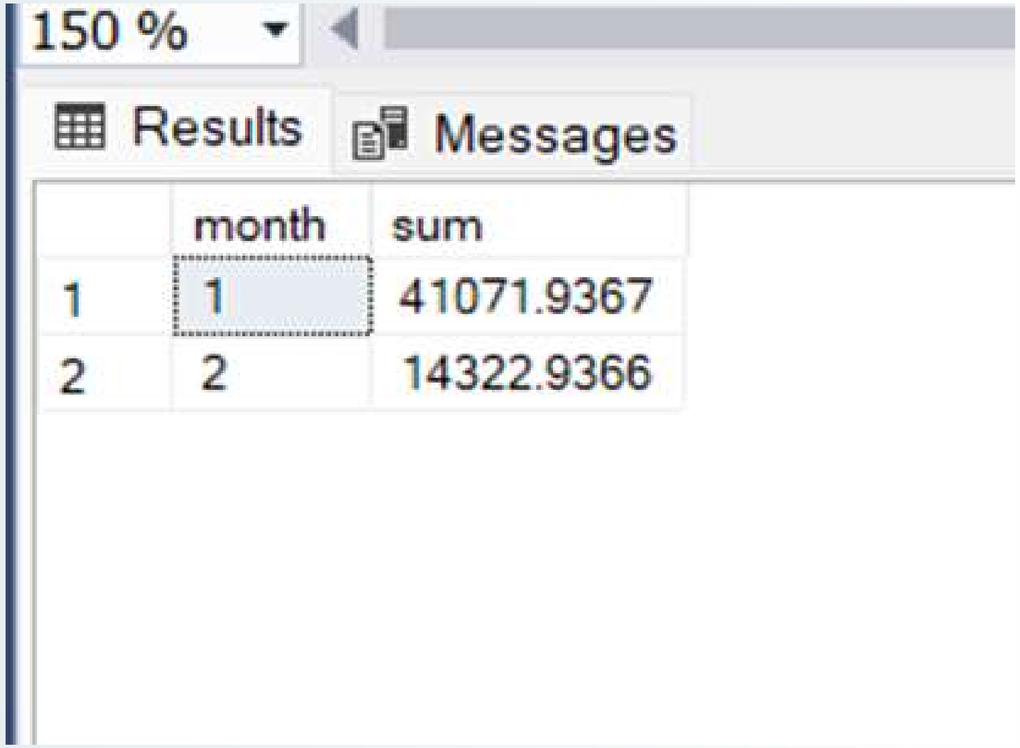
	transactionid	anomalyscore	amount
1	14	0.997339632901219400000000	98.8043
2	39	0.938484553830562800000000	14.6317
3	46	0.992337849640267600000000	79.3579
4	51	0.918156008114674200000000	42.0762
5	74	0.957373586986144000000000	16.2634
6	101	0.960397629346687700000000	58.9239
7	118	0.923014688818437300000000	56.6049
8	150	0.982727036513257800000000	15.2401
9	167	0.957948100514365600000000	36.8725
10	173	0.956696054298193200000000	63.4767
11	183	0.966645463514096100000000	71.6555
12	184	0.934526598934005500000000	76.7448
13	193	0.964499105516097600000000	61.585
14	211	0.970049124736776000000000	67.0313
15	225	0.985410518570541400000000	87.0159
16	227	0.939216165599308900000000	72.1326
17	230	0.958561332812333600000000	99.7192
18	235	0.912297399064296700000000	65.3489
19	261	0.991626338845297500000000	61.3226
20	274	0.950762882928966600000000	35.9241
21	290	0.919952046547111300000000	77.3047

Query executed successfully.

SQL Query Development (MS SQL Server)

Summarize the total transaction amount per month.

```sql
--Summarize the total transaction amount per month.
SELECT MONTH(M.Timestamp) AS month, SUM(T.Amount) AS sum
FROM     transaction_records AS T INNER JOIN
                  transaction_metadata AS M ON T.TransactionID = M.TransactionID
GROUP BY MONTH(M.Timestamp)
```



	month	sum
1	1	41071.9367
2	2	14322.9366

SQL Query Development (MS SQL Server)

List merchants with the most transactions flagged as suspicious.

```sql
--List merchants with the most transactions flagged as suspicious.
SELECT M.MerchantID, M.MerchantName, COUNT(T.TransactionID) AS NoOfSusTransactions
FROM     merchant_data AS M INNER JOIN
                  transaction_metadata AS T ON T.MerchantID = M.MerchantID INNER JOIN
                  transaction_records AS Tr ON T.TransactionID = Tr.transactionID INNER JOIN
                  suspicious_activity AS S ON Tr.customerID = S.CustomerID
WHERE  (S.suspiciousflag = 1)
GROUP BY M.MerchantID, M.MerchantName
ORDER BY NoOfSusTransactions DESC;
```

Results | Messages

	MerchantID	MerchantName	NoOfSusTransactions
1	2007	Merchant 2007	1
2	2013	Merchant 2013	1
3	2051	Merchant 2051	1
4	2099	Merchant 2099	1
5	2126	Merchant 2126	1
6	2155	Merchant 2155	1
7	2162	Merchant 2162	1
8	2169	Merchant 2169	1
9	2184	Merchant 2184	1
10	2364	Merchant 2364	1
11	2383	Merchant 2383	1
12	2398	Merchant 2398	1
13	2414	Merchant 2414	1
14	2479	Merchant 2479	1
15	2486	Merchant 2486	1
16	2514	Merchant 2514	1
17	2571	Merchant 2571	1

Query executed successfully.

SQL Query Development (MS SQL Server)

Calculate a "fraud score" for each customer based on the number of suspicious transactions and anomaly scores.

QLQuery1.sql - DE...aksha Bhusnur (57))*

```sql
--Calculate a "fraud score" for each customer based on the number of suspicious transactions and ano
SELECT cd.customerid, cd.name,
       COUNT(Tr.TransactionId) AS suspicious_transactions,
       AVG(ad.anomalyscore) AS average_anomaly_score,
       (COUNT(Tr.TransactionId) * AVG(ad.anomalyscore)) AS fraudScore
FROM customer_data cd
JOIN transaction_records tr ON cd.customerid = tr.customerid
LEFT JOIN suspicious_activity sa ON tr.transactionid = Tr.TransactionId
LEFT JOIN anomaly_scores ad ON tr.transactionid = ad.transactionid
GROUP BY cd.customerid, cd.name
ORDER BY fraudScore DESC;
```

Results | Messages

	customerid	name	suspicious_transactions	average_anomaly_score	fraudScor
1	1825	Customer 1825	6000	0.66212520652747950000000	3972.751
2	1618	Customer 1618	5000	0.57046719790802094600000	2852.335
3	1244	Customer 1244	4000	0.66992663838383293750000	2679.706
4	1796	Customer 1796	4000	0.66041504839952331500000	2641.660
5	1311	Customer 1311	5000	0.50283072686687863400000	2514.153
6	1995	Customer 1995	4000	0.62655910526015435000000	2506.236
7	1920	Customer 1920	3000	0.82076384300864403333333	2462.291
8	1322	Customer 1322	5000	0.49078087211682189600000	2453.904
9	1678	Customer 1678	3000	0.80705562614121403333333	2421.166
10	1506	Customer 1506	3000	0.79845828403848986666666	2395.374
11	1424	Customer 1424	6000	0.39704027768999598333333	2382.241
12	1902	Customer 1902	3000	0.79279157590476693333333	2378.374
13	1800	Customer 1800	4000	0.57489927299478237500000	2299.597
14	1376	Customer 1376	4000	0.56784205799867096500000	2271.368
15	1405	Customer 1405	4000	0.56476460143379397500000	2259.058
16	1764	Customer 1764	3000	0.75152581567140790000000	2254.577
17	1503	Customer 1503	4000	0.55357874365345053750000	2214.314
18	1117	Customer 1117	4000	0.55006263283635260500000	2200.250

Query executed successfully.

SQL Query Development (MS SQL Server)

Rank merchants by their risk score, defined as the ratio of suspicious(fraud) transactions to total transactions.

```sql
--Rank merchants by their risk score, defined as the ratio of suspicious(fraud) transactions to total transactions.
SELECT
    M.MerchantID,
    COUNT(CASE WHEN F.FraudIndicator = 1 THEN F.TransactionID END) AS FraudTransactions,
    COUNT(T.TransactionID) AS TotalTransactions,
    CASE
        WHEN COUNT(T.TransactionID) = 0 THEN 0
        ELSE CAST(COUNT(CASE WHEN F.FraudIndicator = 1 THEN F.TransactionID END) AS FLOAT) / COUNT(T.TransactionID)
    END AS RiskScore
FROM
    merchant_data M
JOIN
    transaction_metadata T ON T.MerchantID = M.MerchantID
LEFT JOIN
    fraud_indicators F ON F.TransactionID = T.TransactionID
GROUP BY
    M.MerchantID
ORDER BY
    RiskScore DESC;
```

Results | Messages

	MerchantID	FraudTransactions	TotalTransactions	RiskScore
1	2022	1	1	1
2	2126	1	1	1
3	2048	1	1	1
4	2495	1	1	1
5	2470	1	1	1
6	2612	1	1	1
7	2492	1	1	1
8	2572	1	1	1
9	2600	1	1	1
10	2541	1	1	1
11	2675	1	1	1
12	2751	1	1	1
13	2815	1	1	1
14	2828	1	1	1
15	2838	1	1	1
16	2898	1	1	1
17	2776	1	1	1
18	2611	1	2	0.5
19	2929	1	2	0.5
20	2960	1	2	0.5
21	2783	1	2	0.5
22	2679	1	2	0.5
23	2583	1	2	0.5
24	2468	1	2	0.5
25	2211	1	2	0.5
26	2282	1	2	0.5
27	2544	1	2	0.5
28	2365	1	2	0.5
29	2328	1	2	0.5
30	2414	1	2	0.5
31	2217	1	2	0.5
32	2169	1	2	0.5
33	2241	1	2	0.5

Query executed successfully.

SQL Query Development (MS SQL Server)

Find customers who have not made any transactions in the past year.

```sql
--Find customers who have not made any transactions in the past year.
SELECT DISTINCT
    cd.customerid,
    cd.name,
    tm.timestamp
FROM
    customer_data cd
LEFT JOIN
    transaction_records tr ON cd.customerid = tr.customerid
JOIN
    transaction_metadata tm ON tm.transactionID = tr.transactionID
WHERE
    tm.timestamp IS NULL
    OR tm.timestamp < DATEADD(YEAR, -1, GETDATE());
```

Results | Messages

	customerid	name	timestamp
1	1001	Customer 1001	2022-02-02 18:00:00.0000000
2	1003	Customer 1003	2022-01-27 16:00:00.0000000
3	1004	Customer 1004	2022-01-19 17:00:00.0000000
4	1004	Customer 1004	2022-02-08 08:00:00.0000000
5	1005	Customer 1005	2022-01-25 22:00:00.0000000
6	1007	Customer 1007	2022-01-04 04:00:00.0000000
7	1008	Customer 1008	2022-01-20 11:00:00.0000000
8	1009	Customer 1009	2022-01-13 05:00:00.0000000
9	1009	Customer 1009	2022-02-08 01:00:00.0000000
10	1012	Customer 1012	2022-01-05 05:00:00.0000000
11	1012	Customer 1012	2022-01-30 19:00:00.0000000
12	1014	Customer 1014	2022-02-09 08:00:00.0000000
13	1016	Customer 1016	2022-01-23 16:00:00.0000000
14	1017	Customer 1017	2022-01-06 21:00:00.0000000
15	1017	Customer 1017	2022-01-12 00:00:00.0000000
16	1017	Customer 1017	2022-02-02 23:00:00.0000000
17	1018	Customer 1018	2022-01-19 10:00:00.0000000
18	1019	Customer 1019	2022-01-06 20:00:00.0000000
19	1019	Customer 1019	2022-01-24 14:00:00.0000000
20	1020	Customer 1020	2022-02-10 18:00:00.0000000
21	1022	Customer 1022	2022-01-09 04:00:00.0000000
22	1023	Customer 1023	2022-01-05 04:00:00.0000000
23	1023	Customer 1023	2022-01-10 03:00:00.0000000
24	1024	Customer 1024	2022-01-08 23:00:00.0000000
25	1024	Customer 1024	2022-01-10 21:00:00.0000000
26	1024	Customer 1024	2022-01-20 02:00:00.0000000
27	1024	Customer 1024	2022-02-08 10:00:00.0000000
28	1025	Customer 1025	2022-01-21 23:00:00.0000000
29	1026	Customer 1026	2022-01-09 17:00:00.0000000
30	1026	Customer 1026	2022-01-18 18:00:00.0000000

Query executed successfully.

SQL Query Development (MS SQL Server)

Find the transaction categories associated with the highest number of fraud transactions.

```sql
--Find the transaction categories associated with the highest number of fraud transactions.
SELECT C.Category, COUNT(C.TransactionID) AS NoOfFraudTransactions
FROM     transaction_category_labels AS C INNER JOIN
                  fraud_indicators AS F ON F.TransactionID = C.TransactionID
WHERE  (F.FraudIndicator = 1)
GROUP BY C.Category
ORDER BY NoOfFraudTransactions DESC;
```

Results | Messages

	category	NoOfFraudTransactions
1	Online	10
2	Other	10
3	Retail	9
4	Food	9
5	Travel	7

SQL Query Development (MS SQL Server)

Compute the total transaction amount and average transaction value for each customer

```sql
--Compute the total transaction amount and average transaction value for each customer.
SELECT cd.customerid, cd.name,
       SUM(tr.amount) AS total_spent,
       AVG(tr.amount) AS avg_transaction_value
FROM customer_data cd
JOIN transaction_records tr ON cd.customerid = tr.customerid
GROUP BY cd.customerid, cd.name
ORDER BY total_spent DESC;
```

Results | Messages

	customerid	name	total_spent	avg_transaction_value
1	1666	Customer 1666	385.7011	64.2835
2	1825	Customer 1825	352.6797	58.7799
3	1618	Customer 1618	328.7512	65.7502
4	1836	Customer 1836	313.4688	78.3672
5	1424	Customer 1424	294.6036	49.1006
6	1814	Customer 1814	280.796	70.199
7	1480	Customer 1480	276.8078	69.2019
8	1117	Customer 1117	263.8718	65.9679
9	1376	Customer 1376	263.6077	65.9019
10	2000	Customer 2000	253.688	84.5626
11	1311	Customer 1311	251.5552	50.311
12	1132	Customer 1132	246.0974	82.0324
13	1244	Customer 1244	245.5519	61.3879
14	1655	Customer 1655	237.8777	59.4694
15	1501	Customer 1501	227.4484	75.8161
16	1678	Customer 1678	226.8524	75.6174
17	1995	Customer 1995	224.092	56.023
18	1124	Customer 1124	221.1699	73.7233
19	1503	Customer 1503	214.7096	53.6774
20	1176	Customer 1176	212.383	53.0957
21	1024	Customer 1024	208.8845	52.2211
22	1902	Customer 1902	208.5404	69.5134

Query executed successfully.

SQL Query Development (MS SQL Server)

Detect merchants with high-value transactions significantly exceeding their average transaction amount.

```sql
--Detect merchants with high-value transactions significantly exceeding their average transaction amount.
WITH MerchantAvg AS (SELECT M.MerchantID, M.MerchantName, AVG(A.TransactionAmount) AS AvgTransactionAmount
                     FROM      merchant_data AS M INNER JOIN
                               transaction_metadata AS T ON M.MerchantID = T.MerchantID INNER JOIN
                               amount_data AS A ON A.TransactionID = T.TransactionID
                     GROUP BY M.MerchantID, M.MerchantName)
    SELECT M.MerchantID, M.MerchantName, A.TransactionAmount, MA.AvgTransactionAmount
    FROM     merchant_data AS M INNER JOIN
             transaction_metadata AS T ON M.MerchantID = T.MerchantID INNER JOIN
             amount_data AS A ON A.TransactionID = T.TransactionID INNER JOIN
             MerchantAvg AS MA ON M.MerchantID = MA.MerchantID AND A.TransactionAmount > MA.AvgTransactionAmount * 2
    ORDER BY A.TransactionAmount DESC
```

Results | Messages

	merchantID	merchantName	transactionAmount	AvgTransactionAmount
1	2239	Merchant 2239	98.2408	43.1867
2	2235	Merchant 2235	97.4401	45.2919
3	2635	Merchant 2635	96.6389	43.6139
4	2533	Merchant 2533	82.3276	37.4122

SQL Query Development (MS SQL Server)

Detect merchants with high-value transactions significantly exceeding their average transaction amount.

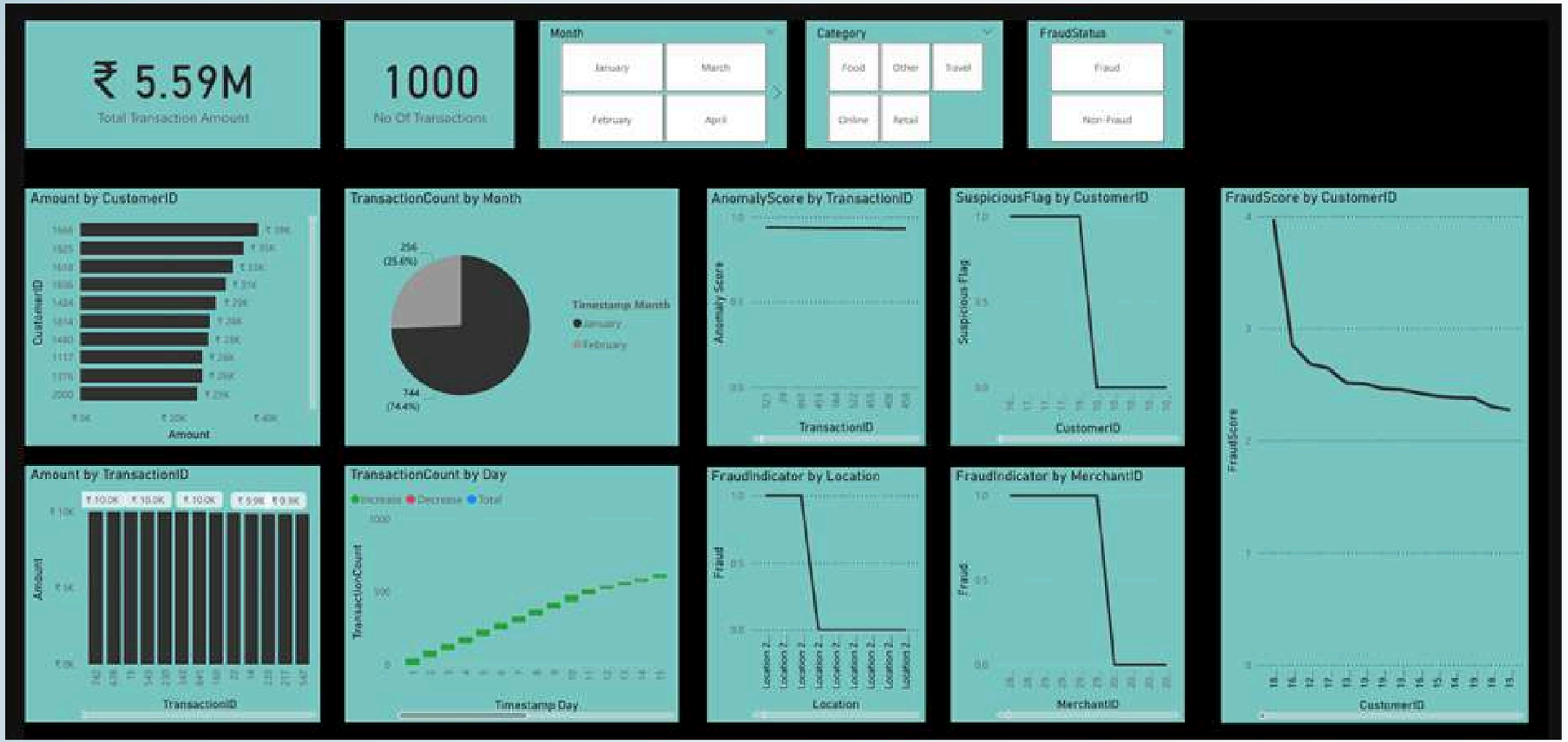
```sql
--Detect merchants with high-value transactions significantly exceeding their average transaction amount.
WITH MerchantAvg AS (SELECT M.MerchantID, M.MerchantName, AVG(A.TransactionAmount) AS AvgTransactionAmount
                     FROM      merchant_data AS M INNER JOIN
                               transaction_metadata AS T ON M.MerchantID = T.MerchantID INNER JOIN
                               amount_data AS A ON A.TransactionID = T.TransactionID
                     GROUP BY M.MerchantID, M.MerchantName)
    SELECT M.MerchantID, M.MerchantName, A.TransactionAmount, MA.AvgTransactionAmount
    FROM     merchant_data AS M INNER JOIN
             transaction_metadata AS T ON M.MerchantID = T.MerchantID INNER JOIN
             amount_data AS A ON A.TransactionID = T.TransactionID INNER JOIN
             MerchantAvg AS MA ON M.MerchantID = MA.MerchantID AND A.TransactionAmount > MA.AvgTransactionAmount * 2
    ORDER BY A.TransactionAmount DESC
```

Results | Messages

	merchantID	merchantName	transactionAmount	AvgTransactionAmount
1	2239	Merchant 2239	98.2408	43.1867
2	2235	Merchant 2235	97.4401	45.2919
3	2635	Merchant 2635	96.6389	43.6139
4	2533	Merchant 2533	82.3276	37.4122

PowerBI Visualizations

PowerBI Visualizations



Conclusion

- The data was thoroughly cleaned and preprocessed using Python, addressing missing values, outliers, and inconsistencies.
- Complex SQL queries were written to identify patterns, trends, and anomalies, highlighting suspicious transactions.
- Power BI dashboards were created to present key insights, providing a visual overview of fraud detection results.

Future Improvements

- Machine Learning Integration: Incorporating machine learning models for predictive fraud detection could improve accuracy. Models like Decision Trees, Random Forests, or Neural Networks can help classify transactions as fraudulent or legitimate with higher precision.
- Real-Time Monitoring: Transitioning from batch processing to real-time transaction monitoring would allow the system to detect fraud as transactions occur
- Anomaly Detection: Implementing more advanced anomaly detection techniques, such as clustering or deep learning models
- Enhanced Data Sources: Integrating external data sources, such as transaction history across different institutions, could provide a more holistic view of potential fraudulent activities.

Thank You!