In [47]:
```
#WEEK 2 START
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

In [20]:
```
#create pandas DataFrame for financial anomaly data
financial_df = pd.read_csv("~/Analytics-Practicum/data/financial_anomaly_data.csv")
```

In [21]:
```
#print first 5 columns of DataFrame
financial_df.head(5)
```

Out[21]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location
0	1/1/2023 8:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo
1	1/1/2023 8:01	TXN1639	ACC10	15607.89	MerchantH	Purchase	London
2	1/1/2023 8:02	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London
3	1/1/2023 8:03	TXN1438	ACC6	87.87	MerchantE	Purchase	London
4	1/1/2023 8:04	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles

In [22]:
```
#print class, RangeIndex, columns, non-null count, data type, and memory usage information
financial_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 216960 entries, 0 to 216959
Data columns (total 7 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Timestamp        216960 non-null  object 
 1   TransactionID    216960 non-null  object 
 2   AccountID        216960 non-null  object 
 3   Amount           216960 non-null  float64
 4   Merchant         216960 non-null  object 
 5   TransactionType  216960 non-null  object 
 6   Location         216960 non-null  object 
dtypes: float64(1), object(6)
memory usage: 11.6+ MB
```

In [23]:
```
#print shape of DataFrame
financial_df.shape
```

Out[23]: (216960, 7)

In [24]:
```
#print sum of null occurrences of each variable in DataFrame
print(financial_df.isnull().sum())
```

```
Timestamp          0
TransactionID      0
AccountID          0
Amount             0
Merchant           0
TransactionType    0
Location           0
dtype: int64
```

In [25]:
```
#create a new DataFrame excluding null occurrences
new_financial_df = financial_df.dropna()
```

In [26]:
```
#print shape of new DataFrame
new_financial_df.shape
```

Out[26]: (216960, 7)

In [9]:
```
#verify that null occurrences were handled properly
print(new_financial_df.isnull().sum())
```

```
Timestamp          0
TransactionID      0
AccountID          0
Amount             0
Merchant           0
TransactionType    0
Location           0
dtype: int64
```

In [10]:
```
#print number of unique occurrences of each variable in DataFrame
print(f"Number of unique Timestamp: {new_financial_df['Timestamp'].nunique()}")
print(f"Number of unique TransactionID: {new_financial_df['TransactionID'].nunique()}")
print(f"Number of unique AccountID: {new_financial_df['AccountID'].nunique()}")
print(f"Number of unique Amount: {new_financial_df['Amount'].nunique()}")
print(f"Number of unique Merchant: {new_financial_df['Merchant'].nunique()}")
print(f"Number of unique TransactionType: {new_financial_df['TransactionType'].nunique()}")
print(f"Number of unique Location: {new_financial_df['Location'].nunique()}")
```

```
Number of unique Timestamp: 216960
Number of unique TransactionID: 1999
Number of unique AccountID: 15
Number of unique Amount: 214687
Number of unique Merchant: 10
Number of unique TransactionType: 3
Number of unique Location: 5
```

In [27]:
```
#introduce new variables to DataFrame for analysis of certain variables' interactions
new_financial_df['AccountID/Merchant'] = new_financial_df['AccountID'].astype(str) + '_' + new_financial_df['Me
new_financial_df['AccountID/TransactionID'] = new_financial_df['AccountID'].astype(str) + '_' + new_financial_d
new_financial_df['AccountID/Merchant/TransactionID'] = new_financial_df['AccountID'].astype(str) + '_' + new_fi
new_financial_df['TransactionType/Merchant'] = new_financial_df['TransactionType'].astype(str) + '_' + new_fina
new_financial_df['Location/TransactionType'] = new_financial_df['Location'].astype(str) + '_' + new_financial_d
new_financial_df['Merchant/Location'] = new_financial_df['Merchant'].astype(str) + '_' + new_financial_df['Loca
```

In [28]:
```
#verify that new variables have been created successfully
new_financial_df.head(5)
```

Out[28]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location	AccountID/Merchant	AccountID/Transact
0	1/1/2023 8:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo	ACC4_MerchantH	ACC4_TXN
1	1/1/2023 8:01	TXN1639	ACC10	15607.89	MerchantH	Purchase	London	ACC10_MerchantH	ACC10_TXN
2	1/1/2023 8:02	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London	ACC8_MerchantE	ACC8_TX
3	1/1/2023 8:03	TXN1438	ACC6	87.87	MerchantE	Purchase	London	ACC6_MerchantE	ACC6_TXN
4	1/1/2023 8:04	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles	ACC6_MerchantI	ACC6_TXN

In [85]:
```
#convert Timestamp variable to a DateTime object
new_financial_df['Timestamp'] = new_financial_df['Timestamp'].astype(str)
new_financial_df['Timestamp'] = new_financial_df['Timestamp'].str.replace('/', '-', regex=False)
new_financial_df['Timestamp'] = pd.to_datetime(new_financial_df['Timestamp'], format='%Y-%m-%d %H:%M:%S')
```

In [86]:
```
#create distinct features for minute/hour of the day, day of the week, and month
new_financial_df['Minute'] = new_financial_df['Timestamp'].dt.minute
new_financial_df['Hour'] = new_financial_df['Timestamp'].dt.hour
new_financial_df['Day'] = new_financial_df['Timestamp'].dt.dayofweek
new_financial_df['Month'] = new_financial_df['Timestamp'].dt.month
```

In [87]:
```
#verify again that new variables have been created successfully
new_financial_df.head(5)
```

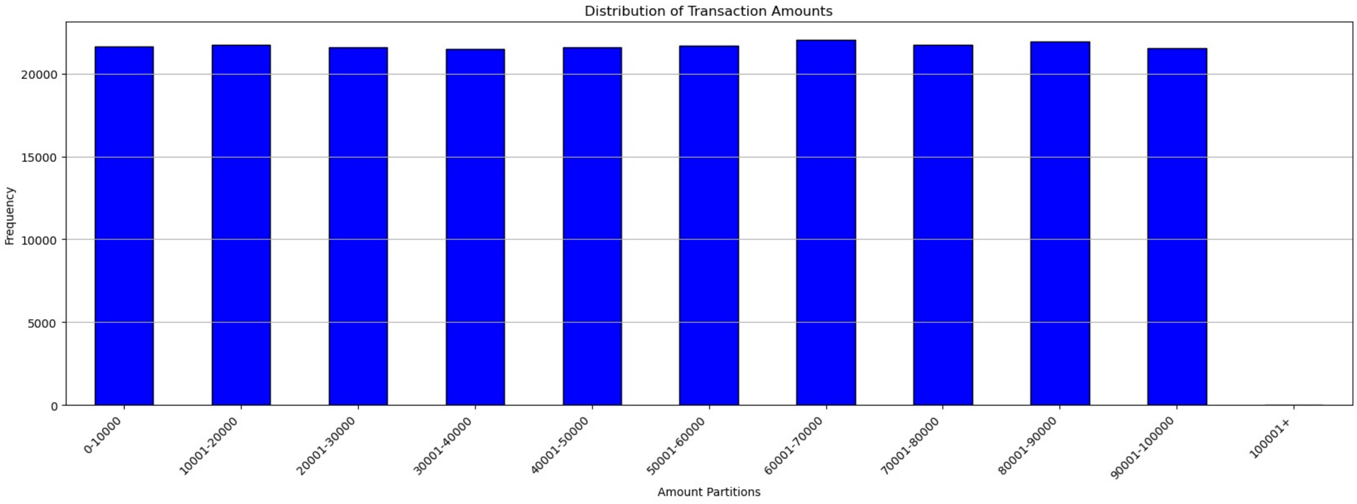
Out[87]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location	AccountID/Merchant	AccountID/Transact
0	2023-01-01 08:00:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo	ACC4_MerchantH	ACC4_TXN
1	2023-01-01 08:01:00	TXN1639	ACC10	15607.89	MerchantH	Purchase	London	ACC10_MerchantH	ACC10_TXN
2	2023-01-01 08:02:00	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London	ACC8_MerchantE	ACC8_TX
3	2023-01-01 08:03:00	TXN1438	ACC6	87.87	MerchantE	Purchase	London	ACC6_MerchantE	ACC6_TXN
4	2023-01-01 08:04:00	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles	ACC6_MerchantI	ACC6_TXN

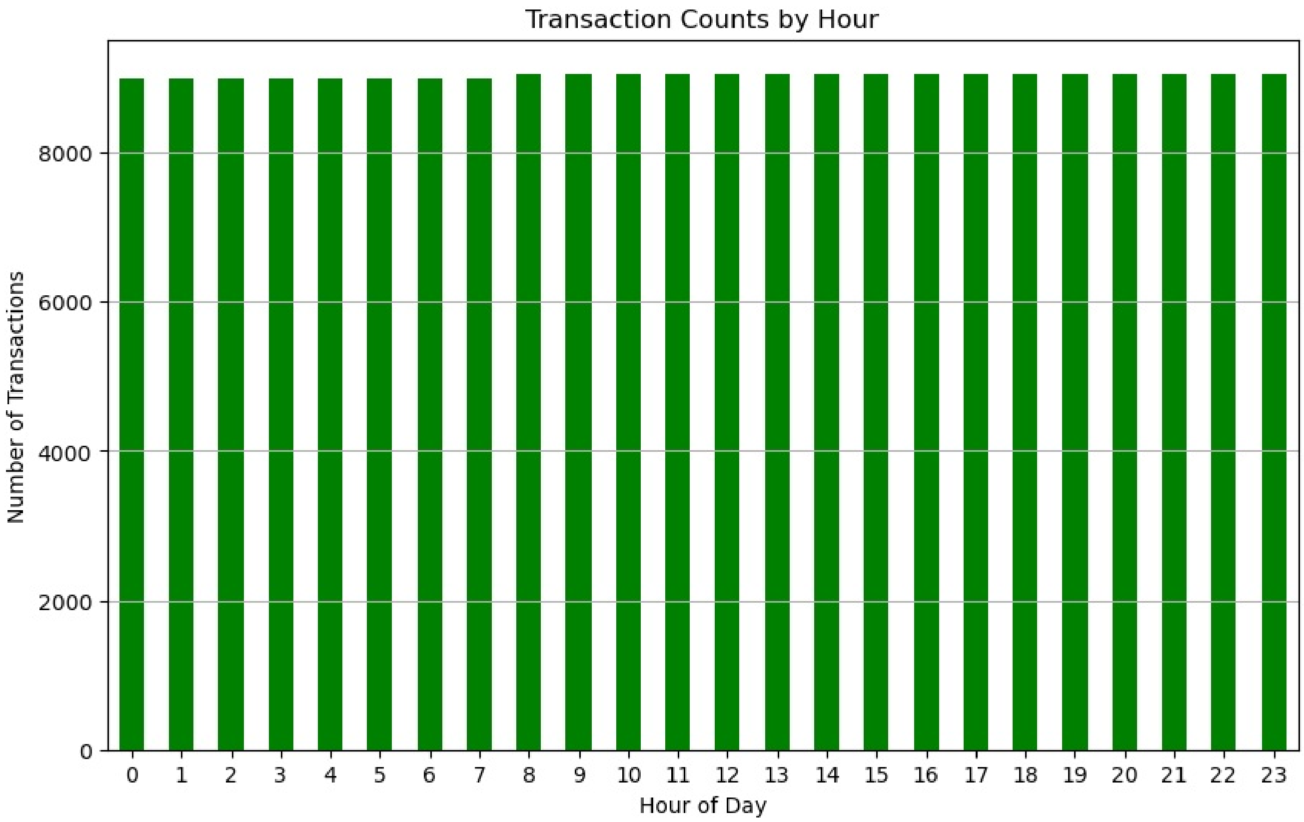
In [88]:
```
#Divide amount variable into appropriately-sized partitions
bins = [0, 10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000, 100000, float('inf')]
labels = ['0-10000', '10001-20000', '20001-30000', '30001-40000', '40001-50000', '50001-60000', '60001-70000',
new_financial_df['Amount_Partitions'] = pd.cut(new_financial_df['Amount'], bins=bins, labels=labels)
```

In [89]:
```
#Construct Bar Graph for distribution of transaction in each amount partition
partition_counts = new_financial_df['Amount_Partitions'].value_counts().reindex(labels)
```

```
plt.figure(figsize=(20, 6))
partition_counts.plot(kind='bar', color='blue', edgecolor='black')
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount Partitions')
plt.ylabel('Frequency')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y')
plt.show()
```

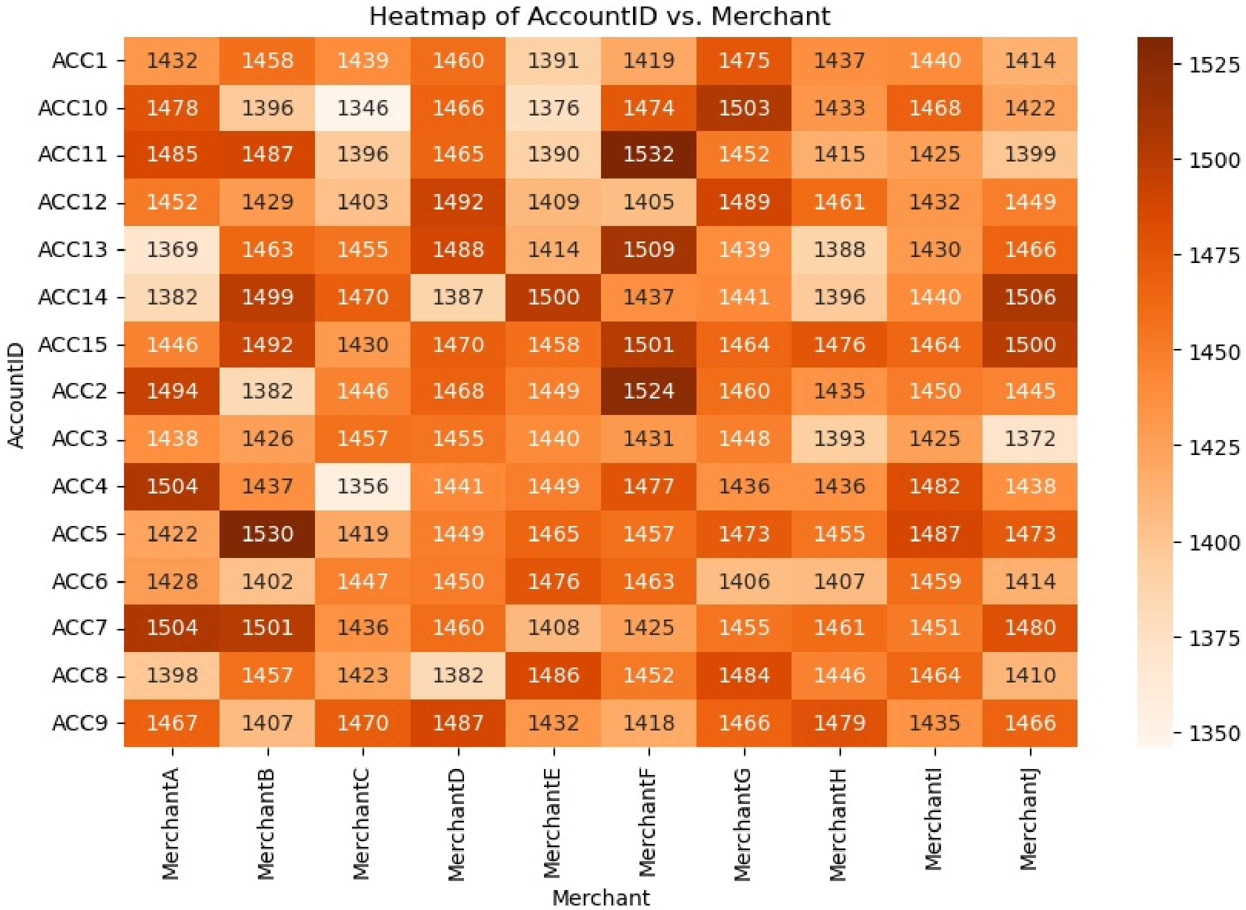


In [90]:
```
#Construct bar graph for total number of transactions per hour
hour_counts = new_financial_df['Hour'].value_counts().sort_index()

plt.figure(figsize=(10, 6))
hour_counts.plot(kind='bar', color='green')
plt.title('Transaction Counts by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=0)
plt.grid(axis='y')
plt.show()
```



In [91]:
```
#Construct heat map to visualize total amounts of each combination of AccountID and Merchant (150 combinations)
pivot_table = pd.crosstab(new_financial_df['AccountID'], new_financial_df['Merchant'])

plt.figure(figsize=(10, 6))
sns.heatmap(pivot_table, annot=True, cmap='Oranges', fmt='d')
plt.title('Heatmap of AccountID vs. Merchant')
plt.xlabel('Merchant')
plt.ylabel('AccountID')
```

```
plt.show()
```



In [92]:
```
#print a sample of the first 10 values of the cleaned dataset with new variables added
new_financial_df.head(10)
```

Out[92]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location	AccountID/Merchant	AccountID/Transact
0	2023-01-01 08:00:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo	ACC4_MerchantH	ACC4_TXI
1	2023-01-01 08:01:00	TXN1639	ACC10	15607.89	MerchantH	Purchase	London	ACC10_MerchantH	ACC10_TXI
2	2023-01-01 08:02:00	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London	ACC8_MerchantE	ACC8_TX
3	2023-01-01 08:03:00	TXN1438	ACC6	87.87	MerchantE	Purchase	London	ACC6_MerchantE	ACC6_TXI
4	2023-01-01 08:04:00	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles	ACC6_MerchantI	ACC6_TXI
5	2023-01-01 08:05:00	TXN1083	ACC15	13957.99	MerchantC	Transfer	London	ACC15_MerchantC	ACC15_TXI
6	2023-01-01 08:06:00	TXN832	ACC9	4654.58	MerchantC	Transfer	Tokyo	ACC9_MerchantC	ACC9_TX
7	2023-01-01 08:07:00	TXN841	ACC7	1336.36	MerchantI	Withdrawal	San Francisco	ACC7_MerchantI	ACC7_TX
8	2023-01-01 08:08:00	TXN777	ACC10	9776.23	MerchantD	Transfer	London	ACC10_MerchantD	ACC10_TX
9	2023-01-01 08:09:00	TXN1479	ACC12	49522.74	MerchantC	Withdrawal	New York	ACC12_MerchantC	ACC12_TXI

In [21]:
```
#print class, RangeIndex, columns, non-null count, data type, and memory usage information for the updated Data
new_financial_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 216960 entries, 0 to 216959
Data columns (total 18 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   Timestamp                         216960 non-null  datetime64[ns]
 1   TransactionID                     216960 non-null  object
 2   AccountID                         216960 non-null  object
 3   Amount                            216960 non-null  float64
 4   Merchant                          216960 non-null  object
 5   TransactionType                   216960 non-null  object
 6   Location                          216960 non-null  object
 7   AccountID/Merchant                216960 non-null  object
 8   AccountID/TransactionID           216960 non-null  object
 9   AccountID/Merchant/TransactionID  216960 non-null  object
 10  TransactionType/Merchant          216960 non-null  object
 11  Location/TransactionType          216960 non-null  object
 12  Merchant/Location                 216960 non-null  object
 13  Minute                            216960 non-null  int32
 14  Hour                              216960 non-null  int32
 15  Day                               216960 non-null  int32
 16  Month                             216960 non-null  int32
 17  Amount_Partitions                 216960 non-null  category
dtypes: category(1), datetime64[ns](1), float64(1), int32(4), object(11)
memory usage: 26.7+ MB
```

In [22]:

```python
#print number of unique occurrences of newly created variables
print(f"Number of unique AccountID/Merchant: {new_financial_df['AccountID/Merchant'].nunique()}")
print(f"Number of unique AccountID/TransactionID: {new_financial_df['AccountID/TransactionID'].nunique()}")
print(f"Number of unique AccountID/Merchant/TransactionID: {new_financial_df['AccountID/Merchant/TransactionID'
print(f"Number of unique TransactionType/Merchant: {new_financial_df['TransactionType/Merchant'].nunique()}")
print(f"Number of unique Location/TransactionType: {new_financial_df['Location/TransactionType'].nunique()}")
print(f"Number of unique Merchant/Location: {new_financial_df['Merchant/Location'].nunique()}")
print(f"Number of unique Minute: {new_financial_df['Minute'].nunique()}")
print(f"Number of unique Hour: {new_financial_df['Hour'].nunique()}")
print(f"Number of unique Day: {new_financial_df['Day'].nunique()}")
print(f"Number of unique Month: {new_financial_df['Month'].nunique()}")
print(f"Number of unique Amount_Partitions: {new_financial_df['Amount_Partitions'].nunique()}")
```

```
Number of unique AccountID/Merchant: 150
Number of unique AccountID/TransactionID: 29967
Number of unique AccountID/Merchant/TransactionID: 154226
Number of unique TransactionType/Merchant: 30
Number of unique Location/TransactionType: 15
Number of unique Merchant/Location: 50
Number of unique Minute: 60
Number of unique Hour: 24
Number of unique Day: 7
Number of unique Month: 5
Number of unique Amount_Partitions: 11
```

In [23]:

```python
new_financial_df.to_csv('Week_2_Data.csv', index=False)
```

In [24]:

```python
#WEEK 2 END
```

In [25]:

```python
#WEEK 3 START
```

In [93]:

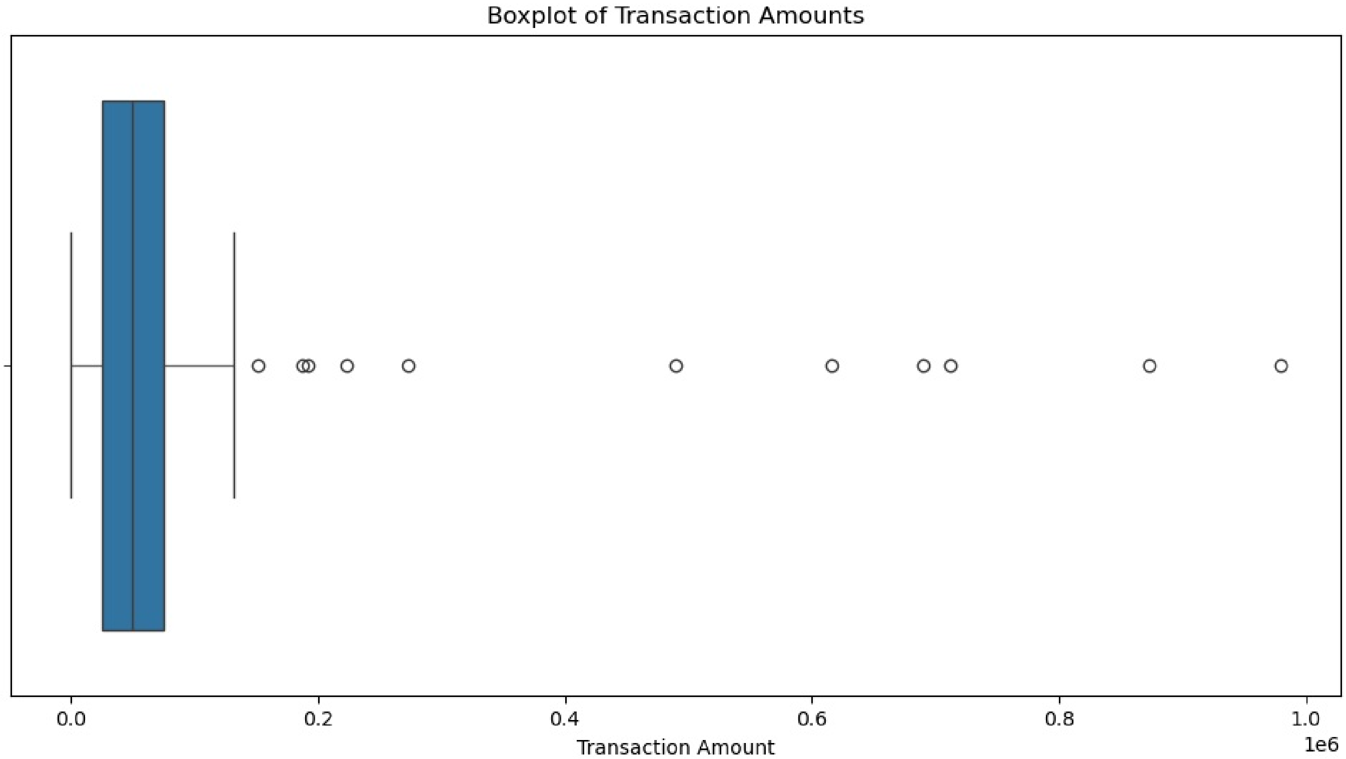
```python
#describe numerical data to better understand these columns
new_financial_df.describe()
```

Out[93]:

	Timestamp	Amount	Minute	Hour	Day	Month
count	216960	216960.000000	216960.000000	216960.000000	216960.000000	216960.000000
mean	2023-04-27 16:24:59.203539968	50090.025108	29.500000	11.517699	3.013274	4.411504
min	2023-01-01 08:00:00	10.510000	0.000000	0.000000	0.000000	1.000000
25%	2023-02-21 23:59:45	25061.242500	14.750000	6.000000	1.000000	2.000000
50%	2023-04-14 15:59:30	50183.980000	29.500000	12.000000	3.000000	4.000000
75%	2023-05-29 07:59:15	75080.460000	44.250000	18.000000	5.000000	5.000000
max	2023-12-05 23:59:00	978942.260000	59.000000	23.000000	6.000000	12.000000
std	NaN	29097.905016	17.318142	6.918770	2.028518	2.976114

In [63]:

```python
plt.figure(figsize=(12, 6))
sns.boxplot(x='Amount', data=new_financial_df)
plt.title('Boxplot of Transaction Amounts')
plt.xlabel('Transaction Amount')
plt.show()
```



In [69]:
```
#print counts of each unique value in each column of the DataFrame
for column in new_financial_df.columns:
    column_count = new_financial_df[column].value_counts()
    print(column_count)
```

```
Timestamp
2023-01-01 08:00:00    1
2023-04-11 18:57:00    1
2023-04-11 18:33:00    1
2023-04-11 18:34:00    1
2023-04-11 18:35:00    1
                      ..
2023-02-20 13:23:00    1
2023-02-20 13:24:00    1
2023-02-20 13:25:00    1
2023-02-20 13:26:00    1
2023-05-31 23:59:00    1
Name: count, Length: 216960, dtype: int64
TransactionID
TXN838     139
TXN1768    139
TXN1658    139
TXN1389    138
TXN340     137
          ...
TXN60       79
TXN891      78
TXN605      78
TXN201      73
TXN799      70
Name: count, Length: 1999, dtype: int64
AccountID
ACC15    14701
ACC5     14630
ACC7     14581
ACC2     14553
ACC9     14527
ACC14    14458
ACC4     14456
ACC11    14446
ACC12    14421
ACC13    14421
ACC8     14402
ACC1     14365
ACC10    14362
ACC6     14352
ACC3     14285
Name: count, dtype: int64
Amount
18010.00    3
34588.69    3
74109.74    3
86099.64    3
```

```
7309.50     3
           ..
56652.57    1
36336.36    1
49174.76    1
71557.91    1
65004.99    1
Name: count, Length: 214687, dtype: int64
Merchant
MerchantF    21924
MerchantG    21891
MerchantD    21820
MerchantB    21766
MerchantI    21752
MerchantA    21699
MerchantJ    21654
MerchantE    21543
MerchantH    21518
MerchantC    21393
Name: count, dtype: int64
TransactionType
Transfer      72793
Purchase      72235
Withdrawal    71932
Name: count, dtype: int64
Location
San Francisco    43613
New York         43378
London           43343
Los Angeles      43335
Tokyo            43291
Name: count, dtype: int64
AccountID/Merchant
ACC11_MerchantF    1532
ACC5_MerchantB     1530
ACC2_MerchantF     1524
ACC13_MerchantF    1509
ACC14_MerchantJ    1506
                   ...
ACC10_MerchantE    1376
ACC3_MerchantJ     1372
ACC13_MerchantA    1369
ACC4_MerchantC     1356
ACC10_MerchantC    1346
Name: count, Length: 150, dtype: int64
AccountID/TransactionID
ACC8_TXN239      22
ACC6_TXN154      20
ACC11_TXN1614    19
ACC11_TXN410     19
ACC1_TXN220      19
                 ..
ACC14_TXN20       1
ACC5_TXN938       1
ACC12_TXN1314     1
ACC3_TXN127       1
ACC2_TXN737       1
Name: count, Length: 29967, dtype: int64
AccountID/Merchant/TransactionID
ACC3_MerchantF_TXN1801     7
ACC11_MerchantJ_TXN1488    6
ACC11_MerchantE_TXN153     6
ACC14_MerchantJ_TXN1389    6
ACC15_MerchantG_TXN220     6
                          ..
ACC10_MerchantH_TXN286     1
ACC7_MerchantF_TXN1587     1
ACC5_MerchantA_TXN1930     1
ACC6_MerchantF_TXN1695     1
ACC3_MerchantG_TXN1807     1
Name: count, Length: 154226, dtype: int64
TransactionType/Merchant
Purchase_MerchantF      7399
Transfer_MerchantG      7354
Transfer_MerchantH      7342
Transfer_MerchantA      7332
Withdrawal_MerchantD    7323
Withdrawal_MerchantI    7308
Transfer_MerchantF      7302
Purchase_MerchantG      7298
Transfer_MerchantB      7291
Transfer_MerchantJ      7286
Purchase_MerchantB      7274
```

```
Purchase_MerchantA      7269
Purchase_MerchantD      7250
Transfer_MerchantD      7247
Withdrawal_MerchantG    7239
Transfer_MerchantI      7238
Withdrawal_MerchantF    7223
Purchase_MerchantE      7216
Purchase_MerchantJ      7216
Transfer_MerchantE      7209
Purchase_MerchantI      7206
Withdrawal_MerchantB    7201
Transfer_MerchantC      7192
Withdrawal_MerchantC    7164
Withdrawal_MerchantJ    7152
Withdrawal_MerchantE    7118
Withdrawal_MerchantH    7106
Withdrawal_MerchantA    7098
Purchase_MerchantH      7070
Purchase_MerchantC      7037
Name: count, dtype: int64
Location/TransactionType
London_Transfer             14653
San Francisco_Transfer      14610
Los Angeles_Transfer        14580
San Francisco_Withdrawal    14515
New York_Transfer           14510
Tokyo_Purchase              14506
San Francisco_Purchase      14488
New York_Purchase           14445
Tokyo_Transfer              14440
New York_Withdrawal         14423
Los Angeles_Purchase        14411
London_Purchase             14385
Tokyo_Withdrawal            14345
Los Angeles_Withdrawal      14344
London_Withdrawal           14305
Name: count, dtype: int64
Merchant/Location
MerchantF_Los Angeles      4476
MerchantD_London           4453
MerchantG_London           4446
MerchantI_Tokyo            4445
MerchantG_New York         4432
MerchantE_San Francisco    4424
MerchantB_Los Angeles      4399
MerchantE_New York         4395
MerchantA_Los Angeles      4394
MerchantH_New York         4393
MerchantA_Tokyo            4393
MerchantB_London           4391
MerchantI_San Francisco    4390
MerchantB_San Francisco    4385
MerchantF_Tokyo            4376
MerchantG_Tokyo            4373
MerchantA_San Francisco    4368
MerchantF_San Francisco    4367
MerchantD_San Francisco    4360
MerchantD_Los Angeles      4360
MerchantF_New York         4356
MerchantJ_Tokyo            4353
MerchantJ_San Francisco    4350
MerchantF_London           4349
MerchantG_San Francisco    4348
MerchantD_Tokyo            4347
MerchantJ_New York         4346
MerchantH_San Francisco    4334
MerchantE_London           4332
MerchantJ_Los Angeles      4332
MerchantB_New York         4332
MerchantA_London           4332
MerchantI_Los Angeles      4330
MerchantH_Tokyo            4311
MerchantC_New York         4310
MerchantI_New York         4302
MerchantD_New York         4300
MerchantC_Tokyo            4296
MerchantG_Los Angeles      4292
MerchantC_San Francisco    4287
MerchantI_London           4285
MerchantC_Los Angeles      4285
MerchantJ_London           4273
MerchantH_London           4267
MerchantB_Tokyo            4259
```

```
MerchantE_Los Angeles      4254
MerchantC_London           4215
MerchantH_Los Angeles      4213
MerchantA_New York         4212
MerchantE_Tokyo            4138
Name: count, dtype: int64
Minute
0     3616
1     3616
32    3616
33    3616
34    3616
35    3616
36    3616
37    3616
38    3616
39    3616
40    3616
41    3616
42    3616
43    3616
44    3616
45    3616
46    3616
47    3616
48    3616
49    3616
50    3616
51    3616
52    3616
53    3616
54    3616
55    3616
56    3616
57    3616
58    3616
31    3616
30    3616
29    3616
14    3616
2     3616
3     3616
4     3616
5     3616
6     3616
7     3616
8     3616
9     3616
10    3616
11    3616
12    3616
13    3616
15    3616
28    3616
16    3616
17    3616
18    3616
19    3616
20    3616
21    3616
22    3616
23    3616
24    3616
25    3616
26    3616
27    3616
59    3616
Name: count, dtype: int64
Hour
8     9060
17    9060
23    9060
22    9060
21    9060
9     9060
19    9060
18    9060
20    9060
16    9060
15    9060
14    9060
13    9060
12    9060
```

```
11    9060
10    9060
0     9000
1     9000
2     9000
3     9000
4     9000
5     9000
6     9000
7     9000
Name: count, dtype: int64
Day
0    31680
1    31680
2    31680
6    31200
3    30240
4    30240
5    30240
Name: count, dtype: int64
Month
3    44640
5    44640
1    44160
4    43200
2    40320
Name: count, dtype: int64
Amount_Partitions
60001-70000     22015
80001-90000     21938
10001-20000     21743
70001-80000     21736
50001-60000     21661
0-10000         21651
40001-50000     21605
20001-30000     21601
90001-100000    21530
30001-40000     21466
100001+            14
Name: count, dtype: int64
```

In [38]:

```
#list variables to be one-hot encoded
'''
one_hot_encoding = [
    'AccountID/Merchant',
    'TransactionType',
    'Location',
    'Amount_Partitions'
]

# Apply one-hot encoding
new_financial_df_encoded = pd.get_dummies(new_financial_df, columns=one_hot_encoding)

# Display the first few rows of the encoded DataFrame
print(new_financial_df_encoded.head())
'''
```

Out[38]:

```
"\none_hot_encoding = [\n    'AccountID/Merchant',\n    'TransactionType',\n    'Location',\n    'Amount_Partit
ions'\n]\n\n# Apply one-hot encoding\nnew_financial_df_encoded = pd.get_dummies(new_financial_df, columns=one_h
ot_encoding)\n\n# Display the first few rows of the encoded DataFrame\nprint(new_financial_df_encoded.head())\n
"
```

In [72]:

```
#print DataFrame info to maintain understanding of DataFrame properties
#new_financial_df_encoded.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 216960 entries, 0 to 216959
Columns: 183 entries, Timestamp to Amount_Partitions_100001+
dtypes: bool(169), datetime64[ns](1), float64(1), int32(4), object(8)
memory usage: 56.5+ MB
```

In [39]:

```
#Retrieve one-hot encoded columns
'''
account_merchant_columns = [col for col in new_financial_df_encoded.columns if 'AccountID/Merchant_' in col]
transaction_type_columns = [col for col in new_financial_df_encoded.columns if 'TransactionType_' in col]
location_columns = [col for col in new_financial_df_encoded.columns if 'Location_' in col]
amount_partitions_columns = [col for col in new_financial_df_encoded.columns if 'Amount_Partitions_' in col]

# Create a dictionary to store correlations
correlation_results = {}

# Iterate through each pair of one-hot encoded columns to compute correlations
for account_merchant in account_merchant_columns:
    for transaction_type in transaction_type_columns:
```

```
        correlation1 = new_financial_df_encoded[account_merchant].corr(new_financial_df_encoded[transaction_typ
        correlation_results[(account_merchant, transaction_type)] = correlation1

for account_merchant in account_merchant_columns:
    for location in location_columns:
        correlation2 = new_financial_df_encoded[account_merchant].corr(new_financial_df_encoded[location])
        correlation_results[(account_merchant, location)] = correlation2

for account_merchant in account_merchant_columns:
    for amount_partitions in amount_partitions_columns:
        correlation3 = new_financial_df_encoded[account_merchant].corr(new_financial_df_encoded[amount_partitio
        correlation_results[(account_merchant, amount_partitions)] = correlation3

for transaction_type in transaction_type_columns:
    for location in location_columns:
        correlation4 = new_financial_df_encoded[transaction_type].corr(new_financial_df_encoded[location])
        correlation_results[(transaction_type, location)] = correlation4

for transaction_type in transaction_type_columns:
    for amount_partitions in amount_partitions_columns:
        correlation5 = new_financial_df_encoded[transaction_type].corr(new_financial_df_encoded[amount_partitio
        correlation_results[(transaction_type, amount_partitions)] = correlation5

for location in location_columns:
    for amount_partitions in amount_partitions_columns:
        correlation6 = new_financial_df_encoded[location].corr(new_financial_df_encoded[amount_partitions])
        correlation_results[(location, amount_partitions)] = correlation6

# Display the results
for (account_merchant, transaction_type), correlation1 in correlation_results.items():
    print(f'Correlation between {account_merchant} and {transaction_type}: {correlation1}')

for (account_merchant, location), correlation2 in correlation_results.items():
    print(f'Correlation between {account_merchant} and {location}: {correlation2}')

for (accout_merchant, amount_partitions), correlation3 in correlation_results.items():
    print(f'Correlation between {account_merchant} and {amount_partitions}: {correlation3}')

for (transaction_type, location), correlation4 in correlation_results.items():
    print(f'Correlation between {transaction_type} and {location}: {correlation4}')

for (transaction_type, amount_partitions), correlation5 in correlation_results.items():
    print(f'Correlation between {transaction_type} and {amount_partitions}: {correlation5}')

for (location, amount_partitions), correlation6 in correlation_results.items():
    print(f'Correlation between {location} and {amount_partitions}: {correlation6}')
    '''
```

Out[39]:

```
"\naccount_merchant_columns = [col for col in new_financial_df_encoded.columns if 'AccountID/Merchant_' in col]
\ntransaction_type_columns = [col for col in new_financial_df_encoded.columns if 'TransactionType_' in col]\nlo
cation_columns = [col for col in new_financial_df_encoded.columns if 'Location_' in col]\namount_partitions_col
umns = [col for col in new_financial_df_encoded.columns if 'Amount_Partitions_' in col]\n\n# Create a dictionar
y to store correlations\ncorrelation_results = {}\n\n# Iterate through each pair of one-hot encoded columns to 
compute correlations\nfor account_merchant in account_merchant_columns:\n    for transaction_type in transactio
n_type_columns:\n        correlation1 = new_financial_df_encoded[account_merchant].corr(new_financial_df_encode
d[transaction_type])\n        correlation_results[(account_merchant, transaction_type)] = correlation1\n\nfor a
ccount_merchant in account_merchant_columns:\n    for location in location_columns:\n        correlation2 = new
_financial_df_encoded[account_merchant].corr(new_financial_df_encoded[location])\n        correlation_results[(
account_merchant, location)] = correlation2\n\nfor account_merchant in account_merchant_columns:\n    for amoun
t_partitions in amount_partitions_columns:\n        correlation3 = new_financial_df_encoded[account_merchant].c
orr(new_financial_df_encoded[amount_partitions])\n        correlation_results[(account_merchant, amount_partiti
ons)] = correlation3\n\nfor transaction_type in transaction_type_columns:\n    for location in location_columns
:\n        correlation4 = new_financial_df_encoded[transaction_type].corr(new_financial_df_encoded[location])\n 
correlation_results[(transaction_type, location)] = correlation4\n\nfor transaction_type in transaction_type_co
lumns:\n    for amount_partitions in amount_partitions_columns:\n        correlation5 = new_financial_df_encode
d[transaction_type].corr(new_financial_df_encoded[amount_partitions])\n        correlation_results[(transaction
_type, amount_partitions)] = correlation5\n\nfor location in location_columns:\n    for amount_partitions in am
ount_partitions_columns:\n        correlation6 = new_financial_df_encoded[location].corr(new_financial_df_encod
ed[amount_partitions])\n        correlation_results[(location, amount_partitions)] = correlation6\n        \n# 
Display the results\nfor (account_merchant, transaction_type), correlation1 in correlation_results.items():\n 
print(f'Correlation between {account_merchant} and {transaction_type}: {correlation1}')\n    \nfor (account_mer
chant, location), correlation2 in correlation_results.items():\n    print(f'Correlation between {account_mercha
nt} and {location}: {correlation2}')\n    \nfor (accout_merchant, amount_partitions), correlation3 in correlati
on_results.items():\n    print(f'Correlation between {account_merchant} and {amount_partitions}: {correlation3}
')\n    \nfor (transaction_type, location), correlation4 in correlation_results.items():\n    print(f'Correlati
on between {transaction_type} and {location}: {correlation4}')\n    \nfor (transaction_type, amount_partitions)
, correlation5 in correlation_results.items():\n    print(f'Correlation between {transaction_type} and {amount_
partitions}: {correlation5}')\n    \nfor (location, amount_partitions), correlation6 in correlation_results.ite
ms():\n    print(f'Correlation between {location} and {amount_partitions}: {correlation6}')\n    "
```

In [40]:

```
'''
correlation_matrix = new_financial_df_encoded[account_merchant_columns + transaction_type_columns].corr()
```

```
# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Heatmap between AccountID/Merchant and TransactionType')
plt.show()
'''
```

Out[40]:

```
'\ncorrelation_matrix = new_financial_df_encoded[account_merchant_columns + transaction_type_columns].corr()\n\n# Create a heatmap\nplt.figure(figsize=(12, 8))\nsns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap=\'coolwarm\', square=True)\nplt.title(\'Correlation Heatmap between AccountID/Merchant and TransactionType\')\nplt.show()\n'
```

In [41]:

```
'''
correlation_matrix = new_financial_df_encoded[transaction_type_columns + location_columns].corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Heatmap between TransactionType and Location')
plt.show()
'''
```

Out[41]:

```
'\ncorrelation_matrix = new_financial_df_encoded[transaction_type_columns + location_columns].corr()\n\n# Create a heatmap\nplt.figure(figsize=(12, 8))\nsns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap=\'coolwarm\', square=True)\nplt.title(\'Correlation Heatmap between TransactionType and Location\')\nplt.show()\n'
```

In [42]:

```
'''
correlation_matrix = new_financial_df_encoded[transaction_type_columns + amount_partitions_columns].corr()

# Create a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Heatmap between TransactionType and Amount_Partitions')
plt.show()
'''
```

Out[42]:

```
'\ncorrelation_matrix = new_financial_df_encoded[transaction_type_columns + amount_partitions_columns].corr()\n\n# Create a heatmap\nplt.figure(figsize=(12, 8))\nsns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap=\'coolwarm\', square=True)\nplt.title(\'Correlation Heatmap between TransactionType and Amount_Partitions\')\nplt.show()\n'
```

In [94]:

```
#create train set (70%) and temporary other set (30%)
train_df, temp_df = train_test_split(new_financial_df, test_size=0.30, random_state=1)

#split the leftover temp set into validation and test sets (50% of 30% each- 15% each)
validation_df, test_df = train_test_split(temp_df, test_size=0.50, random_state=42)

#verify shape of train, validation, and test DataFrames
print(f'Training set shape: {train_df.shape}')
print(f'Validation set shape: {validation_df.shape}')
print(f'Test set shape: {test_df.shape}')
```

```
Training set shape: (151872, 18)
Validation set shape: (32544, 18)
Test set shape: (32544, 18)
```

In [95]:

```
# Save the train set
train_df.to_csv('train_data.csv', index=False)

# Save the validation set
validation_df.to_csv('validation_data.csv', index=False)

# Save the test set
test_df.to_csv('test_data.csv', index=False)

print("DataFrames have been saved as CSV files.")
```

```
DataFrames have been saved as CSV files.
```

In [67]:

```
#END WEEK 3
```

In [14]:

```
#START WEEK 4
```

In [96]:

```
#Apply log transformation to Amount variable
train_df['Amount'] = np.log1p(train_df['Amount'])
```

In [97]:

```
#identify trends in volume of transactions per day per account
train_df['Date'] = train_df['Timestamp'].dt.date
account_activity = train_df.groupby(['Date', 'AccountID']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
```

```
).reset_index()

print(account_activity)
```

```
            Date AccountID  total_transactions  total_amount  average_amount  \
0     2023-01-01      ACC1                  45    465.643415       10.347631
1     2023-01-01     ACC10                  39    401.756535       10.301450
2     2023-01-01     ACC11                  45    466.881311       10.375140
3     2023-01-01     ACC12                  48    494.428680       10.300598
4     2023-01-01     ACC13                  51    537.094450       10.531264
...          ...       ...                 ...           ...             ...
2260  2023-12-05      ACC5                  75    795.515336       10.606871
2261  2023-12-05      ACC6                  53    564.398349       10.649025
2262  2023-12-05      ACC7                  60    632.902432       10.548374
2263  2023-12-05      ACC8                  70    737.592979       10.537043
2264  2023-12-05      ACC9                  80    829.224181       10.365302

      max_transaction  min_transaction
0           11.455237         6.655865
1           11.486849         4.735672
2           11.449300         6.820377
3           11.504645         7.298147
4           11.502063         5.600198
...               ...              ...
2260        11.503390         5.160261
2261        11.505050         7.599867
2262        11.503703         6.454727
2263        11.490852         6.755257
2264        11.489467         6.778694

[2265 rows x 7 columns]
```

In [119…

```
#identify trends in volume of transactions per day per merchant
merchant_activity = train_df.groupby(['Date', 'Merchant']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
).reset_index()

print(merchant_activity)
```

```
            Date   Merchant  total_transactions  total_amount  average_amount  \
0     2023-01-01  MerchantA                  65    693.193900       10.664522
1     2023-01-01  MerchantB                  71    738.537511       10.401937
2     2023-01-01  MerchantC                  83    862.766604       10.394778
3     2023-01-01  MerchantD                  77    807.919846       10.492466
4     2023-01-01  MerchantE                  51    529.932174       10.390827
...          ...        ...                 ...           ...             ...
1505  2023-12-05  MerchantF                 116   1209.703998       10.428483
1506  2023-12-05  MerchantG                  94    966.141520       10.278101
1507  2023-12-05  MerchantH                 103   1102.075913       10.699766
1508  2023-12-05  MerchantI                 110   1165.206592       10.592787
1509  2023-12-05  MerchantJ                 104   1094.717977       10.526134

      max_transaction  min_transaction
0           11.484058         7.952207
1           11.503438         3.548180
2           11.479025         5.491950
3           11.488507         5.600198
4           11.491695         6.264293
...               ...              ...
1505        11.506536         7.657349
1506        11.507756         6.653495
1507        11.503703         7.587767
1508        11.511759         5.992239
1509        11.512097         5.024472

[1510 rows x 7 columns]
```

In [120…

```
#identify trends in volume of transactions per day by location
location_activity = train_df.groupby(['Date', 'Location']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),

).reset_index()

print(location_activity)
```

```
           Date       Location  total_transactions  total_amount  \
0    2023-01-01         London                 146   1530.237727
1    2023-01-01    Los Angeles                 139   1454.136479
2    2023-01-01       New York                 135   1402.617642
3    2023-01-01  San Francisco                 138   1445.854505
4    2023-01-01          Tokyo                 133   1410.359817
..          ...            ...                 ...           ...
750  2023-12-05         London                 173   1808.436815
751  2023-12-05    Los Angeles                 193   2028.860958
752  2023-12-05       New York                 219   2297.486070
753  2023-12-05  San Francisco                 205   2163.578111
754  2023-12-05          Tokyo                 232   2440.682868

     average_amount  max_transaction  min_transaction
0         10.481080        11.502063         5.491950
1         10.461414        11.504645         3.548180
2         10.389760        11.507789         5.600198
3         10.477207        11.504023         4.735672
4         10.604209        11.499541         7.197413
..              ...              ...              ...
750       10.453392        11.506536         5.992239
751       10.512233        11.503571         5.160261
752       10.490804        11.512097         5.024472
753       10.554040        11.507744         6.454727
754       10.520185        11.507756         6.924711

[755 rows x 7 columns]
```

In [99]:
```
train_df.head(5)
```

Out[99]:

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location	AccountID/Merchant	AccountID/Tı
9230	2023-07-01 17:50:00	TXN1858	ACC12	9.064231	MerchantB	Withdrawal	London	ACC12_MerchantB	ACC
41764	2023-01-30 08:04:00	TXN76	ACC9	10.757187	MerchantJ	Transfer	London	ACC9_MerchantJ	A
136513	2023-06-04 03:13:00	TXN847	ACC11	10.996651	MerchantD	Transfer	New York	ACC11_MerchantD	AC(
158548	2023-04-21 10:28:00	TXN852	ACC12	11.204528	MerchantI	Withdrawal	Los Angeles	ACC12_MerchantI	AC(
9929	2023-08-01 05:29:00	TXN1822	ACC1	9.295688	MerchantF	Withdrawal	London	ACC1_MerchantF	AC(

In [100…
```
#drop columns with too many unique values to analyze efficiently
train_df.drop(columns=['TransactionID', 'AccountID/TransactionID', 'AccountID/Merchant/TransactionID', 'AccountI
```

In [101…
```
train_df.head(5)
```

Out[101…

	Timestamp	AccountID	Amount	Merchant	TransactionType	Location	Minute	Hour	Day	Month	Amount_Partitions
9230	2023-07-01 17:50:00	ACC12	9.064231	MerchantB	Withdrawal	London	50	17	5	7	0-10000
41764	2023-01-30 08:04:00	ACC9	10.757187	MerchantJ	Transfer	London	4	8	0	1	40001-50000
136513	2023-06-04 03:13:00	ACC11	10.996651	MerchantD	Transfer	New York	13	3	6	6	50001-60000
158548	2023-04-21 10:28:00	ACC12	11.204528	MerchantI	Withdrawal	Los Angeles	28	10	4	4	70001-80000
9929	2023-08-01 05:29:00	ACC1	9.295688	MerchantF	Withdrawal	London	29	5	1	8	10001-20000

In [138…
```
#One hot encode categorical variables
train_encoded_df = pd.get_dummies(train_df, columns=['AccountID', 'Merchant', 'TransactionType', 'Location', 'A
```

In [139…
```
train_encoded_df.head()
```

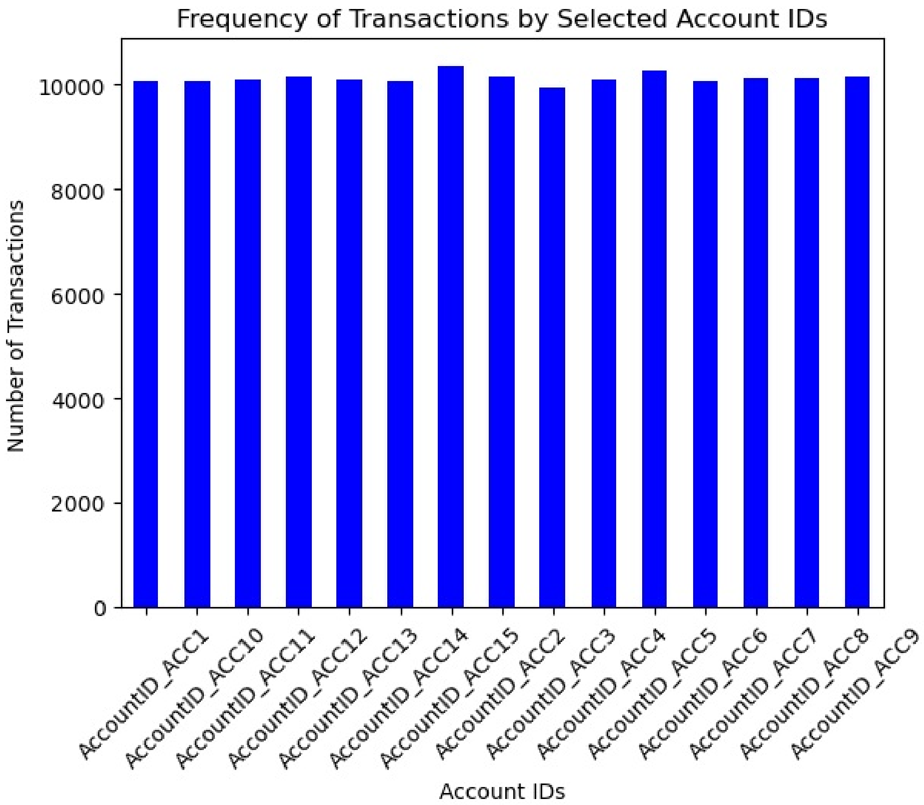
Out[139…

	Timestamp	Amount	Minute	Hour	Day	Month	Date	AccountID_ACC1	AccountID_ACC10	AccountID_ACC11	...	Am
9230	2023-07-01 17:50:00	9.064231	50	17	5	7	2023-07-01	False	False	False	...	
41764	2023-01-30 08:04:00	10.757187	4	8	0	1	2023-01-30	False	False	False	...	
136513	2023-06-04 03:13:00	10.996651	13	3	6	6	2023-06-04	False	False	True	...	
158548	2023-04-21 10:28:00	11.204528	28	10	4	4	2023-04-21	False	False	False	...	
9929	2023-08-01 05:29:00	9.295688	29	5	1	8	2023-08-01	True	False	False	...	

5 rows × 51 columns

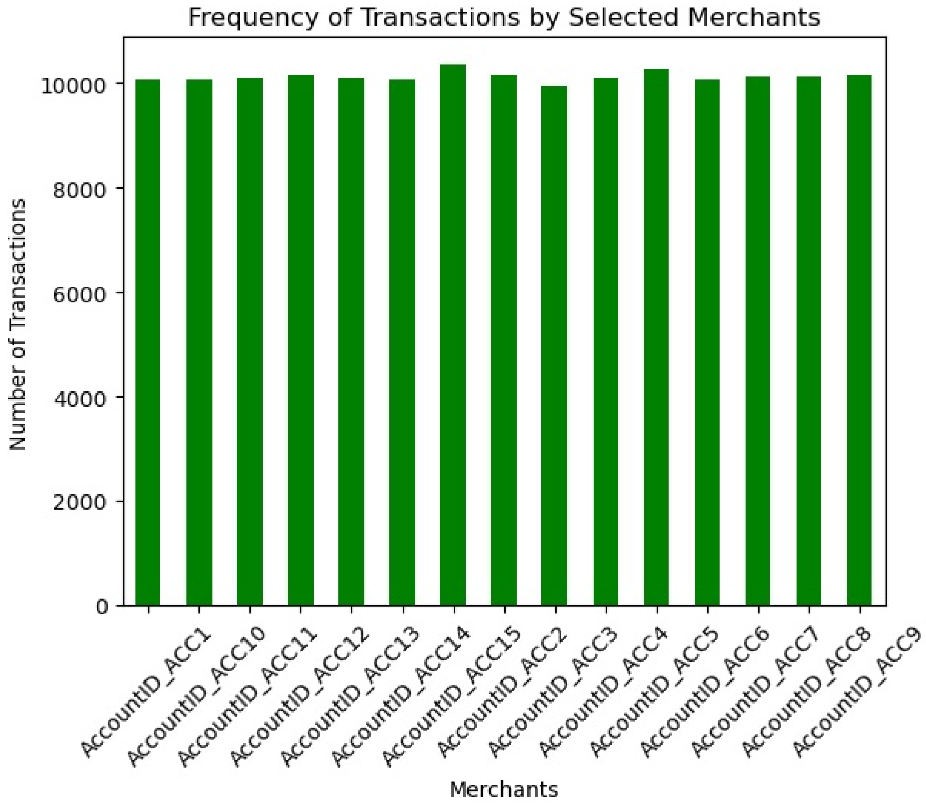
In [111…

```
#Visualize encoded AccountID Data
account_columns = [col for col in train_encoded_df.columns if col.startswith('AccountID_')]
account_counts = train_encoded_df[account_columns].sum()
account_counts.plot(kind='bar', color='blue')
plt.title('Frequency of Transactions by Selected Account IDs')
plt.xlabel('Account IDs')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



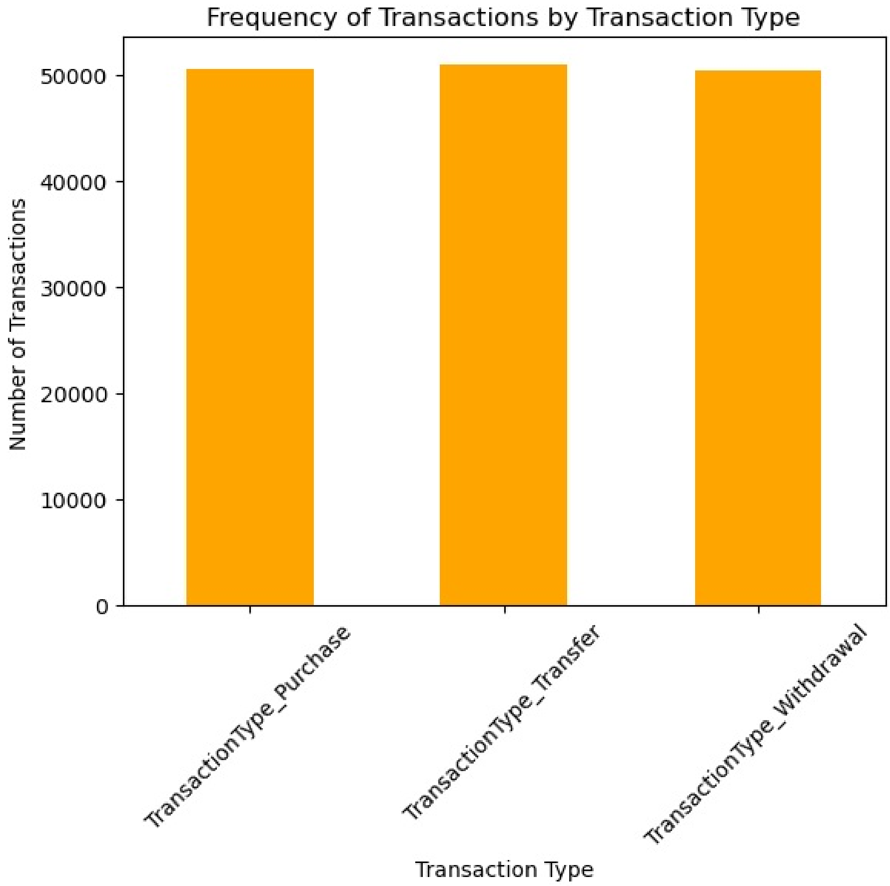
In [112…

```
#Visualize encoded AccountID Data
merchant_columns = [col for col in train_encoded_df.columns if col.startswith('Merchant_')]
merchant_counts = train_encoded_df[account_columns].sum()
merchant_counts.plot(kind='bar', color='green')
plt.title('Frequency of Transactions by Selected Merchants')
plt.xlabel('Merchants')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



In [114…

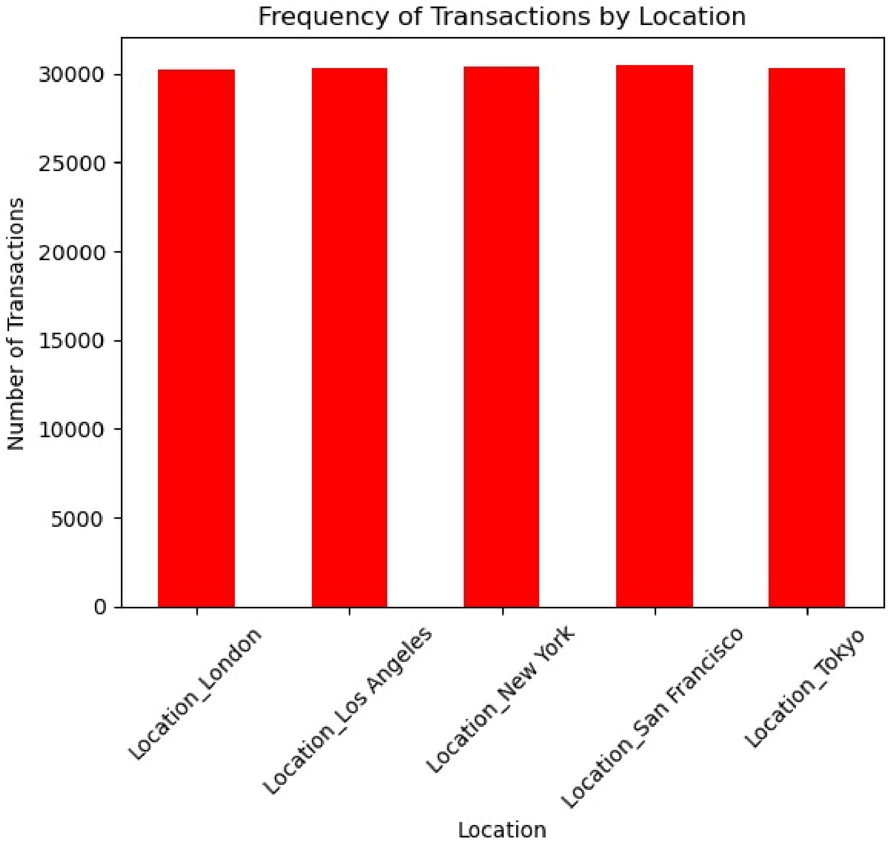
```
#Visualize encoded AccountID Data
TransactionType_columns = [col for col in train_encoded_df.columns if col.startswith('TransactionType_')]
TransactionType_counts = train_encoded_df[TransactionType_columns].sum()
TransactionType_counts.plot(kind='bar', color='orange')
plt.title('Frequency of Transactions by Transaction Type')
plt.xlabel('Transaction Type')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



In [115…

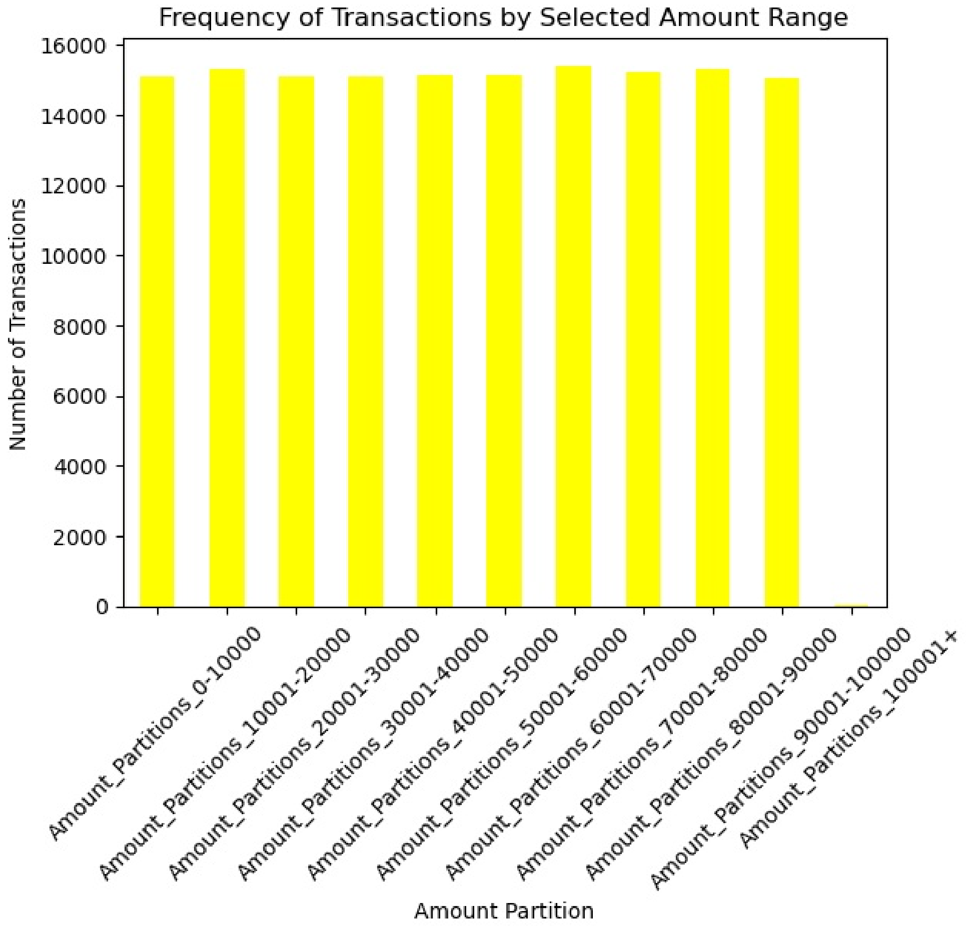
```
#Visualize encoded AccountID Data
location_columns = [col for col in train_encoded_df.columns if col.startswith('Location_')]
location_counts = train_encoded_df[location_columns].sum()
location_counts.plot(kind='bar', color='red')
plt.title('Frequency of Transactions by Location')
```

```
plt.xlabel('Location')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



In [118…

```
#Visualize encoded AccountID Data
amount_partitions_columns = [col for col in train_encoded_df.columns if col.startswith('Amount_Partitions_')]
amount_partitions_counts = train_encoded_df[amount_partitions_columns].sum()
amount_partitions_counts.plot(kind='bar', color='yellow')
plt.title('Frequency of Transactions by Selected Amount Range')
plt.xlabel('Amount Partition')
plt.ylabel('Number of Transactions')
plt.xticks(rotation=45)
plt.show()
```



In [121…
```
#Perform same preprocessing steps on both Validation and Test Sets
#Apply log transformation to Amount variable
validation_df['Amount'] = np.log1p(validation_df['Amount'])
test_df['Amount'] = np.log1p(test_df['Amount'])
```

In [124…
```
#identify trends in volume of transactions per day per account (not printing results to reduce potential for bi
validation_df['Date'] = validation_df['Timestamp'].dt.date
val_account_activity = validation_df.groupby(['Date', 'AccountID']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
).reset_index()
```

In [125…
```
#identify trends in volume of transactions per day per account
test_df['Date'] = test_df['Timestamp'].dt.date
test_account_activity = test_df.groupby(['Date', 'AccountID']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
).reset_index()
```

In [126…
```
#identify trends in volume of transactions per day per merchant
val_merchant_activity = validation_df.groupby(['Date', 'Merchant']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
).reset_index()
```

In [127…
```
#identify trends in volume of transactions per day per merchant
test_merchant_activity = test_df.groupby(['Date', 'Merchant']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),
).reset_index()
```

In [129…
```
#identify trends in volume of transactions per day by location
val_location_activity = validation_df.groupby(['Date', 'Location']).agg(
```

```python
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),

).reset_index()
```

In [130…
```python
#identify trends in volume of transactions per day by location
test_location_activity = test_df.groupby(['Date', 'Location']).agg(
    total_transactions=('Amount', 'count'),
    total_amount=('Amount', 'sum'),
    average_amount=('Amount', 'mean'),
    max_transaction=('Amount', 'max'),
    min_transaction=('Amount', 'min'),

).reset_index()
```

In [131…
```python
#drop columns with too many unique values to analyze efficiently
validation_df.drop(columns=['TransactionID', 'AccountID/TransactionID', 'AccountID/Merchant/TransactionID', 'Acc
#drop columns with too many unique values to analyze efficiently
test_df.drop(columns=['TransactionID', 'AccountID/TransactionID', 'AccountID/Merchant/TransactionID', 'AccountII
```

In [140…
```python
#One hot encode categorical variables
validation_encoded_df = pd.get_dummies(validation_df, columns=['AccountID', 'Merchant', 'TransactionType', 'Loca
test_encoded_df = pd.get_dummies(test_df, columns=['AccountID', 'Merchant', 'TransactionType', 'Location', 'Amou
```

In [141…
```python
# Save the train set
train_encoded_df.to_csv('train_data.csv', index=False)

# Save the validation set
validation_encoded_df.to_csv('validation_data.csv', index=False)

# Save the test set
test_encoded_df.to_csv('test_data.csv', index=False)

print("DataFrames have been saved as CSV files.")
```

```
DataFrames have been saved as CSV files.
```

In [136…
```python
#END WEEK 4
```