In [4]:
```
import pandas as pd
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.ensemble import IsolationForest
from sklearn.metrics import classification_report
```

In [5]:
```
data = pd.read_csv('Transaction_anomalies_dataset.csv')
print(data.head())
print(data.shape)
```

```
   Transaction_ID  Transaction_Amount  Transaction_Volume  \
0            T50            1024.820700                   3
1            TK1            1013.952065                   4
2            TX2             976.966893                   1
3            TK4            1040.922354                   2
4            TX4             998.777241                   1

   Average_Transaction_Amount  Frequency_of_Transactions  \
0                  1024.820700                          7
1                  1013.952065                          7
2                   976.966893                          5
3                  1040.922354                          7
4                   998.777241                          5

   Time_Since_Last_Transaction Day_of_Week Time_of_Day  Age  Gender  Income  \
0                           29      Friday     06:00  36    Male  143074
1                           12     Monday     21:00  61    Male  108232
2                           12     Tuesday     21:00  61    Male  108232
3                           28      Sunday     14:00  61    Male  619030
4                            7      Friday     00:00  56  Female  649457

  Account_Type
0      Savings
1      Savings
2      Savings
3      Savings
4      Savings
(1000, 12)
```

In [6]:
```
print(data.isnull().sum())
```

```
Transaction_ID                 0
Transaction_Amount             0
Transaction_Volume             0
Average_Transaction_Amount     0
Frequency_of_Transactions      0
Time_Since_Last_Transaction    0
Day_of_Week                    0
Time_of_Day                    0
Age                            0
Gender                         0
Income                         0
Account_Type                   0
dtype: int64
```

In [7]:
```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Transaction_ID              1000 non-null   object
 1   Transaction_Amount          1000 non-null   float64
 2   Transaction_Volume          1000 non-null   int64
 3   Average_Transaction_Amount  1000 non-null   float64
 4   Frequency_of_Transactions   1000 non-null   int64
 5   Time_Since_Last_Transaction 1000 non-null   int64
 6   Day_of_Week                 1000 non-null   object
 7   Time_of_Day                 1000 non-null   object
 8   Age                         1000 non-null   int64
 9   Gender                      1000 non-null   object
 10  Income                      1000 non-null   int64
 11  Account_Type                1000 non-null   object
dtypes: float64(2), int64(5), object(5)
memory usage: 93.9+ KB
None
```

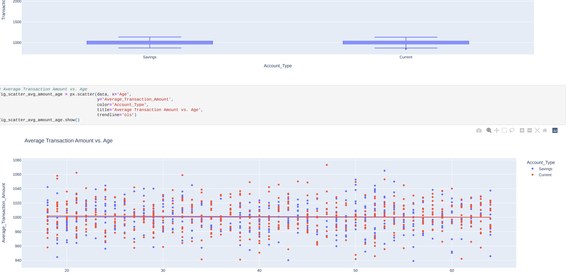
In [8]:
```
print(data.describe())
```

```
       Transaction_Amount  Transaction_Volume  Average_Transaction_Amount  \
count         1000.000000         1000.000000                 1000.000000
mean          1010.123511            2.498000                 1000.662506
std            283.580055            1.119806                   29.633234
min            500.000000            1.000000                  900.000000
25%            966.629700            1.000000                  980.000556
50%           1002.118670            3.000000                 1000.501902
75%           1040.922354            3.000000                 1020.000000
max           3227.454108            4.000000                 1073.154836

       Frequency_of_Transactions  Time_Since_Last_Transaction          Age  \
count                1000.000000                  1000.000000  1000.000000
mean                    5.079000                    15.341000    40.541000
std                     4.245125                     9.361256    11.618953
min                     1.000000                     1.000000    18.000000
25%                     1.000000                     7.000000    31.000000
50%                     5.000000                    16.000000    41.000000
75%                     7.000000                    22.000000    51.000000
max                    19.000000                    29.000000    64.000000

             Income
count  1.000000e+03
mean   8.948238e+05
std    3.016394e+05
min    3.001500e+05
25%    5.573984e+05
50%    9.000000e+05
75%    1.178183e+06
max    1.499676e+06
```

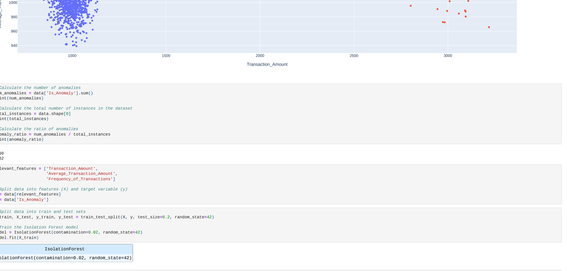
In [9]:
```
# Distribution of Transaction Amount
fig_amount = px.histogram(data, x='Transaction_Amount',
                          nbins=20,
                          title='Distribution of Transaction Amount')
fig_amount.show()
```




In [10]:
```
# Transaction Amount by Account Type
fig_box_amount = px.box(data,
                        x='Account_Type',
                        y='Transaction_Amount',
                        title='Transaction Amount by Account Type')
fig_box_amount.show()
```

In [11]:
```
# Average Transaction Amount vs. Age
fig_scatter_avg_amount_age = px.scatter(data, x='Age',
                                        y='Average_Transaction_Amount',
                                        color='Account_Type',
                                        title='Average Transaction Amount vs. Age',
                                        trendline='ols')
fig_scatter_avg_amount_age.show()
```

In [12]:
```
# Count of Transactions by Day of the Week
fig_day_of_week = px.bar(data, x='Day_of_Week',
                         title='Count of Transactions by Day of the Week')
fig_day_of_week.show()
```




In [13]:
```
print(data.dtypes)
```

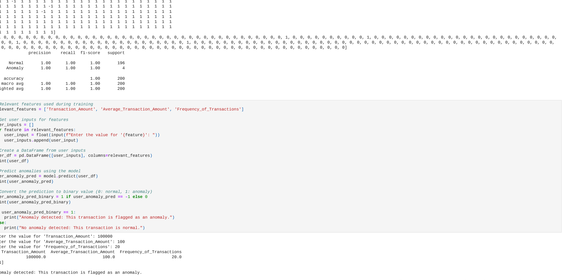
```
Transaction_ID                 object
Transaction_Amount            float64
Transaction_Volume              int64
Average_Transaction_Amount    float64
Frequency_of_Transactions       int64
Time_Since_Last_Transaction     int64
Day_of_Week                    object
Time_of_Day                    object
Age                             int64
Gender                         object
Income                          int64
Account_Type                   object
dtype: object
```

In [14]:
```
data_numeric = data.select_dtypes(include=['float64', 'int64'])

correlation_matrix = data_numeric.corr()

fig_corr_heatmap = px.imshow(correlation_matrix, title='Correlation Heatmap')

fig_corr_heatmap.show()
```




In [15]:
```
# Calculate mean and standard deviation of Transaction Amount
mean_amount = data['Transaction_Amount'].mean()
std_amount = data['Transaction_Amount'].std()

print(mean_amount, std_amount)
```

```
1010.1225107562063 283.5806471553556
```

In [16]:
```
# Define the anomaly threshold (2 standard deviations from the mean)
anomaly_threshold = mean_amount + 2 * std_amount
print(anomaly_threshold)
```

```
1605.2826201800164
```

In [17]:
```
# Flag anomalies
data['Is_Anomaly'] = data['Transaction_Amount'] > anomaly_threshold

data['Is_Anomaly'].value_counts()
```

Out[17]:
```
Is_Anomaly
False    980
True      20
Name: count, dtype: int64
```

In [18]:
```
# Scatter plot of Transaction Amount with anomalies highlighted
fig_anomalies = px.scatter(data, x='Transaction_Amount', y='Average_Transaction_Amount',
                           color='Is_Anomaly', title='Anomalies in Transaction Amount')
fig_anomalies.update_traces(marker=dict(size=2),
                            selector=dict(mode='markers'), marker_size=1)

fig_anomalies.show()
```




In [19]:
```
# Calculate the number of anomalies
num_anomalies = data['Is_Anomaly'].sum()
print(num_anomalies)

# Calculate the total number of instances in the dataset
total_instances = data.shape[0]
print(total_instances)

# Calculate the ratio of anomalies
anomaly_ratio = num_anomalies / total_instances
print(anomaly_ratio)
```

```
20
1000
0.02
```

In [20]:
```
relevant_features = ['Transaction_Amount',
                     'Average_Transaction_Amount',
                     'Frequency_of_Transactions']

# Split data into features (X) and target variable (y)
X = data[relevant_features]
y = data['Is_Anomaly']

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the Isolation Forest model
model = IsolationForest(contamination=0.02, random_state=42)
model.fit(X_train)
```

Out[20]:
```
IsolationForest
IsolationForest(contamination=0.02, random_state=42)
```

In [21]:
```
# Predict anomalies on the test set
y_pred = model.predict(X_test)
print(y_pred)
# Convert predictions to binary values (0: normal, 1: anomaly)
y_pred_binary = [1 if pred == -1 else 0 for pred in y_pred]
print(y_pred_binary)

# Evaluate the model's performance
report = classification_report(y_test, y_pred_binary, target_names=['Normal', 'Anomaly'])
print(report)
```

```
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
  
```