In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
#to ignore warnings
import warnings
warnings.filterwarnings('ignore')

# View all columns
pd.set_option('display.max_columns',500)
```

In [2]:

```
data = pd.read_csv("Servicing - Product homework dataset.csv")
#("test.csv")
```

In [3]:

```
data.head()
```

Out[3]:

	user_id	request_id	target_recipient_id	date_user_c
0	77656618388e134648db01eecf7e79ee	2e9f911150e3a79e8d71a35779706e4c	992e0a729d6380d3b50aef5aa7c22572	27/01/201
1	a2497e0c763a7e5640fbf05e53fe0466	69cdf2f9ab2f59d10b636dc86bc9d7b7	02878ea857dbc90b2ed89b8f3488d501	12/10/201
2	759735d092819085c125a5cf81faf24b	5d7d30d709268f22f1a73ddbd6601690	927d3808cdc31d61226ae7c80bc8de16	04/10/201
3	df9627db375322e65f4648ca72f4c630	0df2fc0a4a31595678cd1de3fad57e15	0411b7eb4c220a14876e77da8125f79b	17/10/20
4	5672b2f16063ed75fbb304fee57c024b	7dcbf32659ae5ef61e10e5174a314d7d	dc248a1266709a71e45c40f33056bbb1	12/08/20

Data Cleaning

In [4]:

```
# removing bad user data (where lenght of user_id is incorrect)
data = data[data.user_id.str.len() > 19]
```

In [5]:

```
data.dtypes
```

Out[5]:

```
user_id                          object
request_id                       object
target_recipient_id              object
date_user_created                object
addr_country_code                object
addr_city                        object
recipient_country_code           object
flag_personal_business           object
payment_type                     object
date_request_submitted           object
date_request_received            object
date_request_transferred         object
date_request_cancelled           object
```

```
invoice_value                        object
invoice_value_cancel                 object
flag_transferred                     object
payment_status                       object
ccy_send                             object
ccy_target                           object
transfer_to_self                     object
sending_bank_name                    object
sending_bank_country                 object
payment_reference_classification     object
device                               object
transfer_sequence                   float64
days_since_previous_req             float64
first_attempt_date                   object
first_success_date                   object
dtype: object
```

In [6]:

```python
# Convert string columns
string_columns = [
    "user_id", "request_id", "target_recipient_id", "addr_country_code", "addr_city",
    "recipient_country_code", "flag_personal_business", "payment_type", "payment_status",
    "ccy_send", "ccy_target", "sending_bank_name", "sending_bank_country",
    "payment_reference_classification", "device", "transfer_to_self"
]
data[string_columns] = data[string_columns].astype("string")

# Convert integer columns
integer_columns = ["transfer_sequence", "days_since_previous_req","flag_transferred"]
data[integer_columns] = data[integer_columns].apply(pd.to_numeric, errors='coerce').asty
pe("Int64")

# Convert invoice_value
data["invoice_value"] = (
    data["invoice_value"]
    .replace(r"[^0-9.]", "", regex=True)   # Remove non-numeric characters
    .apply(pd.to_numeric, errors='coerce')  # Convert to numeric
    .astype("float64")
)

# Convert date columns safely
date_columns = [
    "date_user_created", "date_request_submitted", "date_request_received",
    "date_request_transferred", "first_attempt_date", "first_success_date"
]
data[date_columns] = data[date_columns].apply(pd.to_datetime, errors='coerce')

# Check for invalid values
print(data.dtypes)
print("Invalid dates:\n", data[date_columns].isna().sum())
print("Invalid numeric values:\n", data["invoice_value"].isna().sum())
```

```
user_id                                     string
request_id                                  string
target_recipient_id                         string
date_user_created                   datetime64[ns]
addr_country_code                           string
addr_city                                   string
recipient_country_code                      string
flag_personal_business                      string
payment_type                                string
date_request_submitted              datetime64[ns]
date_request_received               datetime64[ns]
date_request_transferred            datetime64[ns]
date_request_cancelled                      object
invoice_value                              float64
invoice_value_cancel                        object
flag_transferred                             Int64
payment_status                              string
ccy_send                                    string
ccy_target                                  string
```

```
transfer_to_self                             string
sending_bank_name                            string
sending_bank_country                         string
payment_reference_classification             string
device                                       string
transfer_sequence                             Int64
days_since_previous_req                       Int64
first_attempt_date                   datetime64[ns]
first_success_date                   datetime64[ns]
dtype: object
Invalid dates:
 date_user_created                0
date_request_submitted         8
date_request_received      21391
date_request_transferred   22629
first_attempt_date            10
first_success_date          4547
dtype: int64
Invalid numeric values:
 22267
```

In [7]:

```
# Merge columns: Take value from invoice_value if present, else take from invoice_value_c
ancel
data['final_invoice_value'] = data['invoice_value'].combine_first(data['invoice_value_can
cel'])

# Convert combined invoice values
data["final_invoice_value"] = (
    data["final_invoice_value"]
    .replace(r"[^0-9.]", "", regex=True)  # Remove non-numeric characters
    .apply(pd.to_numeric, errors='coerce')  # Convert to numeric
    .astype("float64")
)

# Convert to float
#data.final_invoice_value = data.final_invoice_value.astype(float)


# Convert date columns to datetime format
data['date_request_submitted'] = pd.to_datetime(data['date_request_submitted'], errors='
coerce')
data['date_user_created'] = pd.to_datetime(data['date_user_created'], errors='coerce')
data['date_request_transferred'] = pd.to_datetime(data['date_request_transferred'], erro
rs='coerce')

# Calculate the account age (in days)
data['account_age_days'] = (data['date_request_submitted'] - data['date_user_created']).d
t.days

# Create a flag: 1 if the value comes from invoice_value_cancel, else 0
data['is_cancelled'] = data['invoice_value_cancel'].notna().astype(int)

# Synching the city names
data['addr_city'] = data.addr_city.str.lower()

# Transaction time column (how long did it take for transaction to be processed)
data['transaction_time'] = (data['date_request_transferred'] - data['date_request_submitt
ed']).dt.days

# Create a year column and month column for transaction date
data['month'] = data['date_request_submitted'].dt.month_name()
data['year'] = data['date_request_submitted'].dt.year

# High-risk country codes
high_risk_country_codes = {'DZ', 'AO', 'BG', 'KE', 'NG', 'VN', 'SY', 'IR', 'YE', 'CD', '
SS', 'MM'}
#Source: https://www.lawsociety.org.uk/topics/anti-money-laundering/high-risk-third-count
ries-for-aml-purposes
```

```
# Apply high-risk flag using lambda function
data['country_risk_score'] = data['recipient_country_code'].apply(lambda x: 1 if x in hi
gh_risk_country_codes else 0)
```

In [8]:

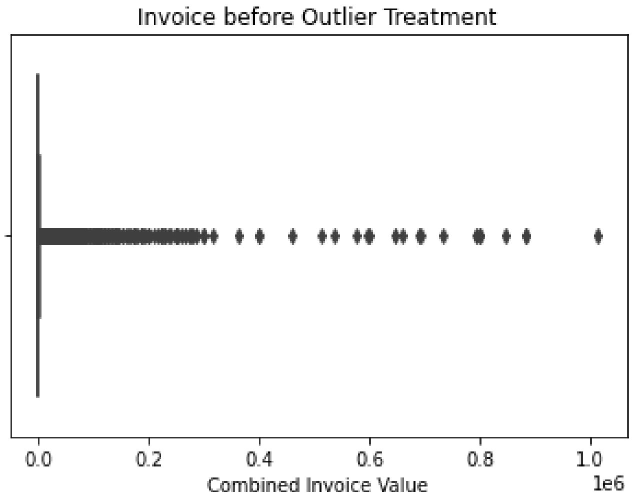
```
# Removing non usuable invoice values

df = data.dropna(subset=['final_invoice_value', 'payment_status', 'transfer_sequence'],a
xis=0)

# Filling in missing values
df['days_since_previous_req'] = df.days_since_previous_req.fillna(0)
```

In [9]:

```
from sklearn.preprocessing import RobustScaler


# Normalize numerical features (e.g., transaction amount)
scaler = RobustScaler()
df['amount_normalized'] = scaler.fit_transform(df[['final_invoice_value']])

# Feature Engineering
df['transaction_hour'] = pd.to_datetime(df['date_request_submitted']).dt.hour
df['transaction_day'] = pd.to_datetime(df['date_request_submitted']).dt.dayofweek
```

In [10]:

```
# Plotting outliers in invoice

x = df.final_invoice_value

#plt.figure(figsize=(6,5))
sns.boxplot(x)

plt.title("Invoice before Outlier Treatment")
plt.xlabel("Combined Invoice Value")
plt.show()
```

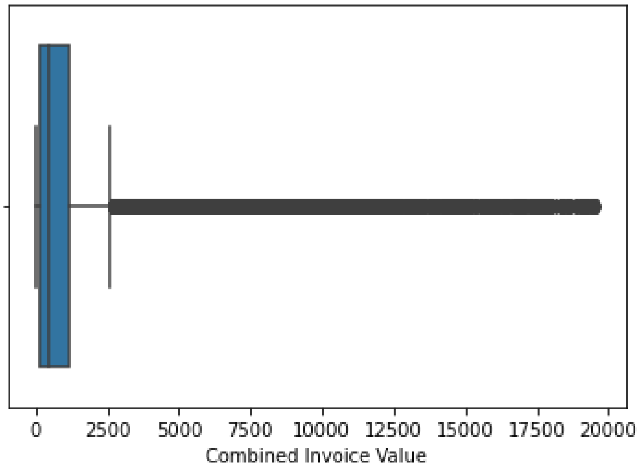


In [11]:

```
# Treating outliers

lower_bound = df['final_invoice_value'].quantile(0.01)   # 1st percentile
upper_bound = df['final_invoice_value'].quantile(0.99)   # 99th percentile

# Remove outliers in invoice_value
df_filt = df[(df['final_invoice_value'] >= lower_bound) & (df['final_invoice_value'] <=
upper_bound)]

plt.xlabel("Invoice Value")
plt.title("Invoice Values after Outlier Treatment")
```

```python
ot_iv = df_filt['final_invoice_value']
sns.boxplot(ot_iv)

plt.xlabel("Combined Invoice Value")
plt.show()
```

Invoice Values after Outlier Treatment



In [12]:

```python
from sklearn.feature_selection import VarianceThreshold

# Define numerical columns
numerical_cols = [
    "invoice_value", "flag_transferred", "transfer_sequence",
    "days_since_previous_req", "final_invoice_value", "account_age_days",
    "transaction_time", "year", "country_risk_score"
]

# Apply Variance Threshold
var_thresh = VarianceThreshold(threshold=0.01)  # Remove near-constant features
X_num = df[numerical_cols]
X_reduced = var_thresh.fit_transform(X_num)

# Get selected feature names
selected_features = X_num.columns[var_thresh.get_support()]
print("Selected Features after Variance Threshold:", selected_features.tolist())
```

```
Selected Features after Variance Threshold: ['invoice_value', 'flag_transferred', 'transf
er_sequence', 'days_since_previous_req', 'final_invoice_value', 'account_age_days', 'tran
saction_time', 'year', 'country_risk_score']
```

df_filt[df_filt.user_id == 'd966e072fab4f783c66d30fa2ed4a723'] # This user is a definite flag

Creating Z-scores for all columns

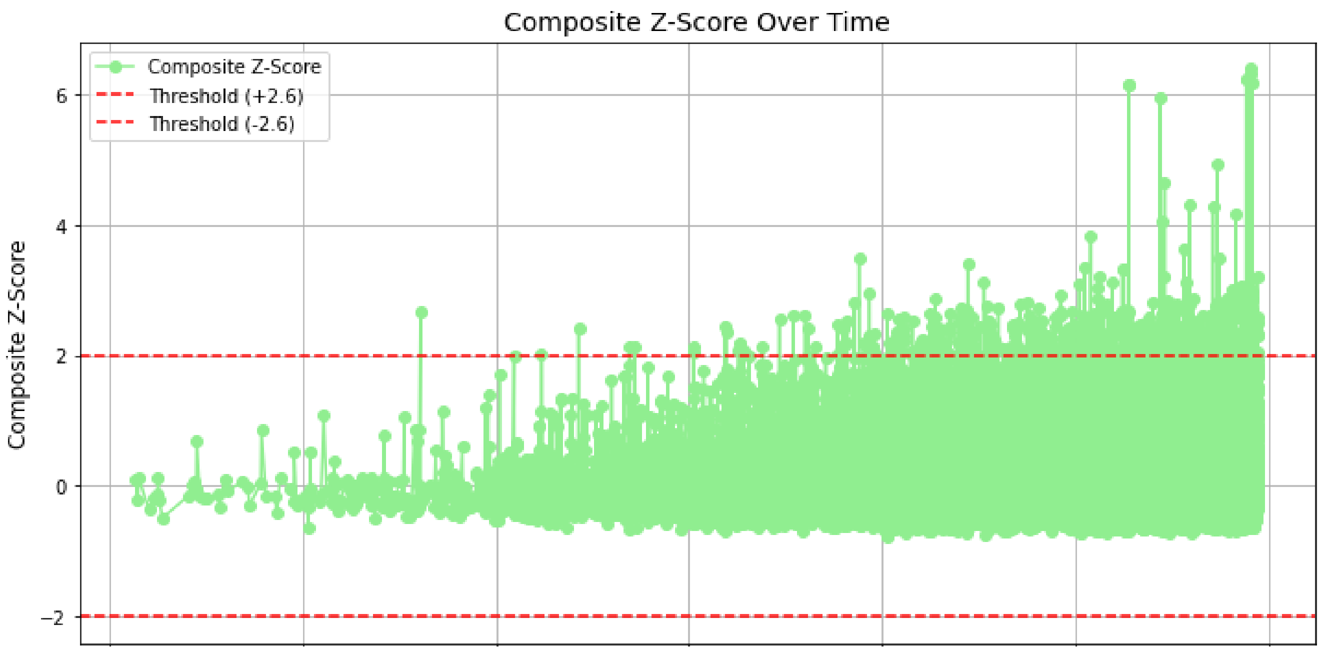
In [13]:

```python
# Define the numerical columns to calculate Z-scores for
num_cols = ['final_invoice_value'
            ,'days_since_previous_req'
            ,'transaction_hour'
            ,'transaction_day'
            ,'country_risk_score'
            ,'transfer_sequence'
            ,'account_age_days'
            #,'amount_normalized'
           ]

for col in num_cols:
    mean_val = df_filt[col].mean()
    std_dev = df_filt[col].std()
```

```
    # Compute Z-Scores for each column
    df_filt[f'z_score_{col}'] = (df_filt[col] - mean_val) / std_dev

    # Flag transactions with Z-Scores greater than 3 (extreme outliers)
    df_filt[f'suspicious_{col}'] = df_filt[f'z_score_{col}'].apply(lambda x: 1 if abs(x)
> 3 else 0)
```

In [14]:

```
# Define weights for each column (adjust as needed)
weights = {
    'final_invoice_value': 0.3,
    'days_since_previous_req': 0.1,
    'transaction_hour': 0.1,
    'transaction_day': 0.1,
    'country_risk_score': 0.2,
    'transfer_sequence': 0.1,
    'account_age_days': 0.1
}

# Get all z_score columns
z_score_cols = [f'z_score_{col}' for col in num_cols]

# Convert all Z-score columns to float32 and handle NaN values
df_filt[z_score_cols] = df_filt[z_score_cols].astype(np.float32)
df_filt[z_score_cols] = df_filt[z_score_cols].fillna(0)  # Replace NaNs with 0

# Compute weighted composite Z-score
df_filt['composite_z_score'] = sum(df_filt[z_col] * weights[col] for z_col, col in zip(z
_score_cols, num_cols))

# Ensure there are no NaN values in the final result
df_filt['composite_z_score'] = df_filt['composite_z_score'].fillna(0)

# Sort data by time
df_filt = df_filt.sort_values(by='date_request_submitted')

# Plot composite Z-score over time
plt.figure(figsize=(12, 6))
plt.plot(df_filt['date_request_submitted'], df_filt['composite_z_score'],
         linestyle='-', marker='o', color='lightgreen', label='Composite Z-Score')
plt.axhline(y=2, color='r', linestyle='--', label='Threshold (+2.6)')
plt.axhline(y=-2, color='r', linestyle='--', label='Threshold (-2.6)')
plt.xlabel("Transaction Date", fontsize=12)
plt.ylabel("Composite Z-Score", fontsize=12)
plt.title("Composite Z-Score Over Time", fontsize=14)
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

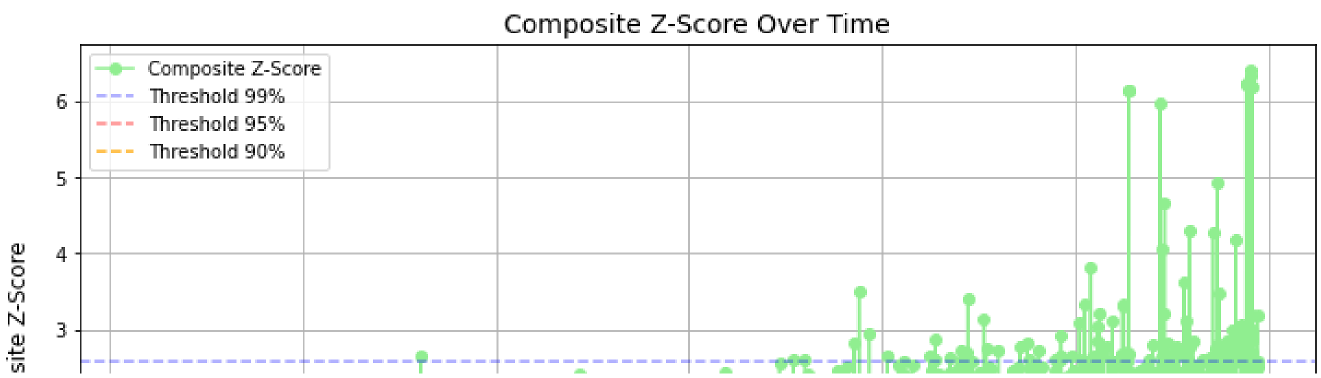


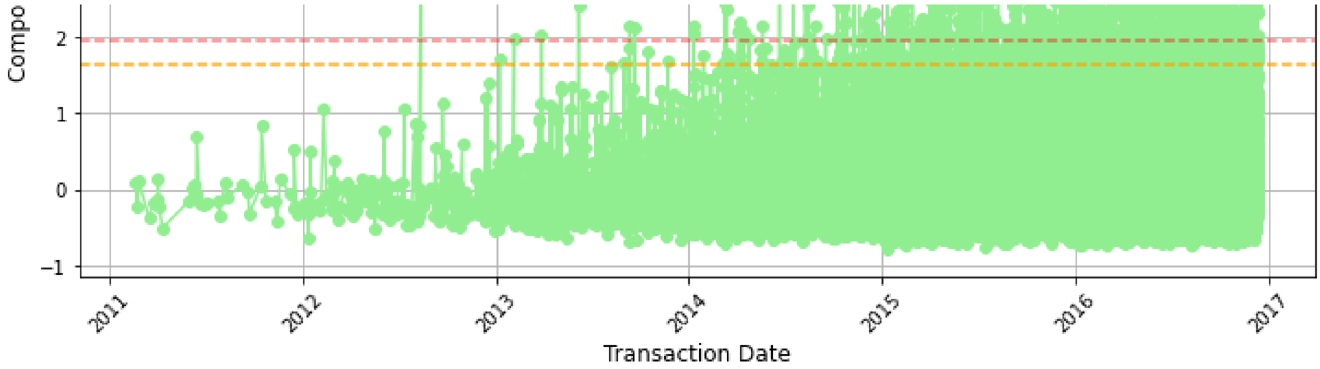
What are the different confidence levels of 90, 95, 99 percents of being default.

Each percentage represents the probability of a transaction being fraudulent. A 90% confidence level indicates a moderate likelihood, 95% suggests a strong probability, and 99% signifies a very high certainty that the transaction is fraudulent.

In [15]:

```
# Define weights for each column (adjust as needed)
weights = {
    'final_invoice_value': 0.3,
    'days_since_previous_req': 0.1,
    'transaction_hour': 0.1,
    'transaction_day': 0.1,
    'country_risk_score': 0.2,
    'transfer_sequence': 0.1,
    'account_age_days': 0.1
}

# Get all z_score columns
z_score_cols = [f'z_score_{col}' for col in num_cols]

# Convert all Z-score columns to float32 and handle NaN values
df_filt[z_score_cols] = df_filt[z_score_cols].astype(np.float32)
df_filt[z_score_cols] = df_filt[z_score_cols].fillna(0)  # Replace NaNs with 0

# Compute weighted composite Z-score
df_filt['composite_z_score'] = sum(df_filt[z_col] * weights[col] for z_col, col in zip(z
_score_cols, num_cols))

# Ensure there are no NaN values in the final result
df_filt['composite_z_score'] = df_filt['composite_z_score'].fillna(0)

# Sort data by time
df_filt = df_filt.sort_values(by='date_request_submitted')

# Plot composite Z-score over time
plt.figure(figsize=(12, 6))
plt.plot(df_filt['date_request_submitted'], df_filt['composite_z_score'],
         linestyle='-', marker='o', color='lightgreen', label='Composite Z-Score')
plt.axhline(y=2.58, color='b', alpha = 0.4,linestyle='--', label='Threshold 99%')
#plt.axhline(y=-2.58, color='r', linestyle='--', label='Threshold -99%')
plt.axhline(y=1.96, color='r', alpha=0.5,linestyle='--', label='Threshold 95%')
plt.axhline(y=1.65, color='orange', alpha=0.9,linestyle='--', label='Threshold 90%')
plt.xlabel("Transaction Date", fontsize=12)
plt.ylabel("Composite Z-Score", fontsize=12)
plt.title("Composite Z-Score Over Time", fontsize=14)
plt.legend()
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```





In [16]:

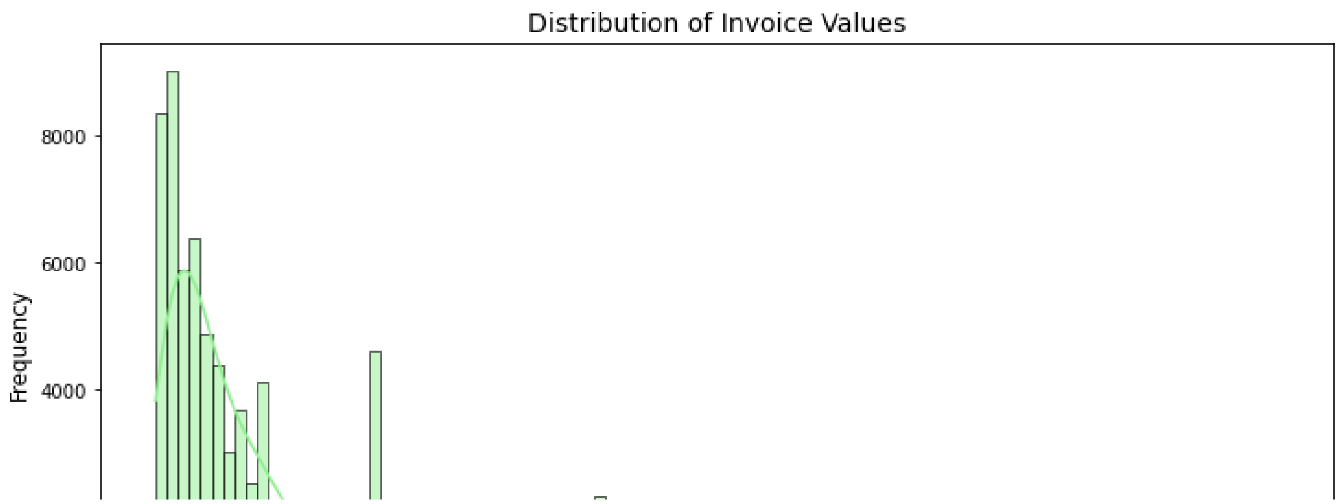
```
# View of suspicious transaction on selected numerical variables (This is a Univariate fl
ag display)

df_sus_zs = df_filt[[
    'suspicious_final_invoice_value'
    ,'suspicious_transaction_hour'
    ,'suspicious_transaction_day'
    ,'suspicious_country_risk_score'
    ,'suspicious_transfer_sequence'
    ,'suspicious_account_age_days'
    ,'suspicious_days_since_previous_req'
]]

df_sus_zs.sum()
```

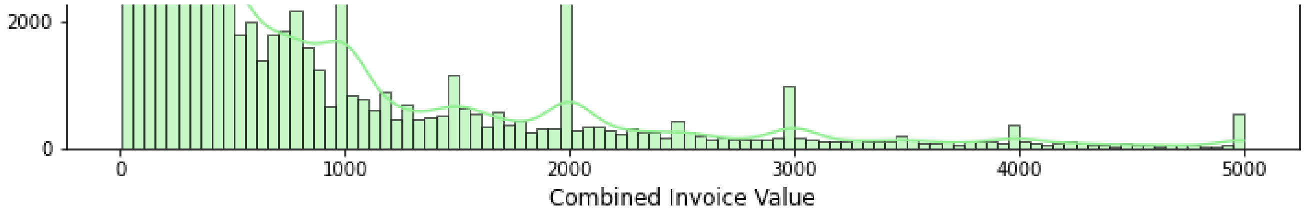
Out[16]:

```
suspicious_final_invoice_value        2548
suspicious_transaction_hour              0
suspicious_transaction_day               0
suspicious_country_risk_score         1395
suspicious_transfer_sequence           255
suspicious_account_age_days           1014
suspicious_days_since_previous_req    1870
dtype: int64
```

In [17]:

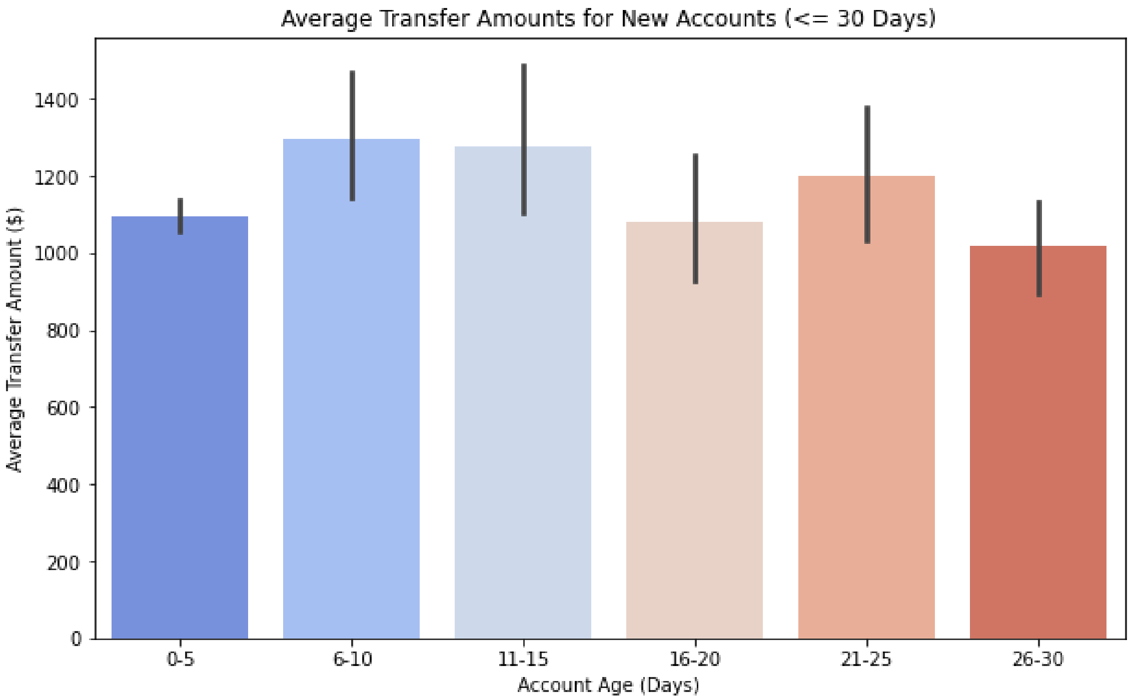
```
# Visualize distribution of 'invoice_value'
plt.figure(figsize=(12, 6))
sns.histplot(df_filt[df_filt['final_invoice_value']<=5000].final_invoice_value,
             kde=True, bins=100, color='lightgreen')
plt.title('Distribution of Invoice Values', fontsize = 14)
plt.xlabel('Combined Invoice Value', fontsize=12)
plt.ylabel('Frequency', fontsize =12)
plt.show()

# Ensure 'days_since_previous_req' is numeric
df_filt['days_since_previous_req'] = pd.to_numeric(df_filt['days_since_previous_req'], e
rrors='coerce')
```





In [18]:

```
# Bin the account age into categories (e.g., 0-5, 6-10, ..., 25-30)
bins = [0, 5, 10, 15, 20, 25, 30]
labels = ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30']
df_filt['account_age_bin'] = pd.cut(df_filt['account_age_days'], bins=bins, labels=label
s, right=False)

# Filter for new accounts (account age <= 30 days) to plot
new_accounts = df_filt[df_filt['account_age_days'] <= 30]

# Plot the average transfer amount for each age bin (categorizing account age)
plt.figure(figsize=(10,6))
sns.barplot(x='account_age_bin', y='invoice_value', data=new_accounts, palette='coolwarm
')
plt.title("Average Transfer Amounts for New Accounts (<= 30 Days)")
plt.xlabel("Account Age (Days)")
plt.ylabel("Average Transfer Amount ($)")
plt.show()
```



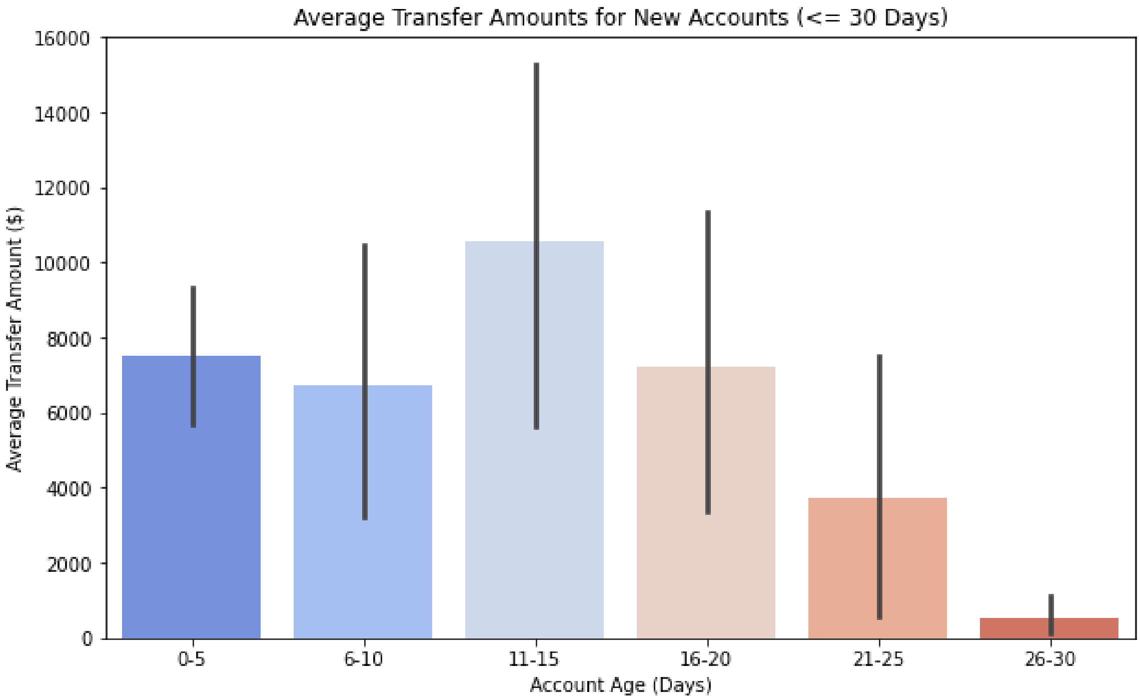
Scaler Isolation forest

In [19]:

```
from sklearn.ensemble import IsolationForest

# Select features for anomaly detection
features = ['amount_normalized', 'transaction_hour', 'transaction_day', 'transfer_sequenc
e','account_age_days'
           ,'country_risk_score', 'days_since_previous_req']
X = df_filt[features]

# Train Isolation Forest
isolation_forest = IsolationForest(contamination=0.03)  # 3% assumed fraud rate
df_filt['anomaly_score'] = isolation_forest.fit_predict(X)

# Flag anomalies
df_filt['is_fraud'] = df_filt['anomaly_score'].apply(lambda x: 1 if x == -1 else 0)
```

In [20]:

```
# Bin the account age into categories (e.g., 0-5, 6-10, ..., 25-30)
bins = [0, 5, 10, 15, 20, 25, 30]
labels = ['0-5', '6-10', '11-15', '16-20', '21-25', '26-30']
df_filt['account_age_bin'] = pd.cut(df_filt['account_age_days'], bins=bins, labels=label
s, right=False)

# Filter for new accounts (account age <= 30 days) to plot
new_accounts_fraud = df_filt[(df_filt['account_age_days'] <= 30) & (df_filt['is_fraud']
> 0)]

# Plot the average transfer amount for each age bin (categorizing account age)
plt.figure(figsize=(10,6))
sns.barplot(x='account_age_bin', y='invoice_value', data=new_accounts_fraud, palette='co
olwarm')
plt.title("Average Transfer Amounts for New Accounts (<= 30 Days)")
plt.xlabel("Account Age (Days)")
plt.ylabel("Average Transfer Amount ($)")
plt.show()
```



In [21]:

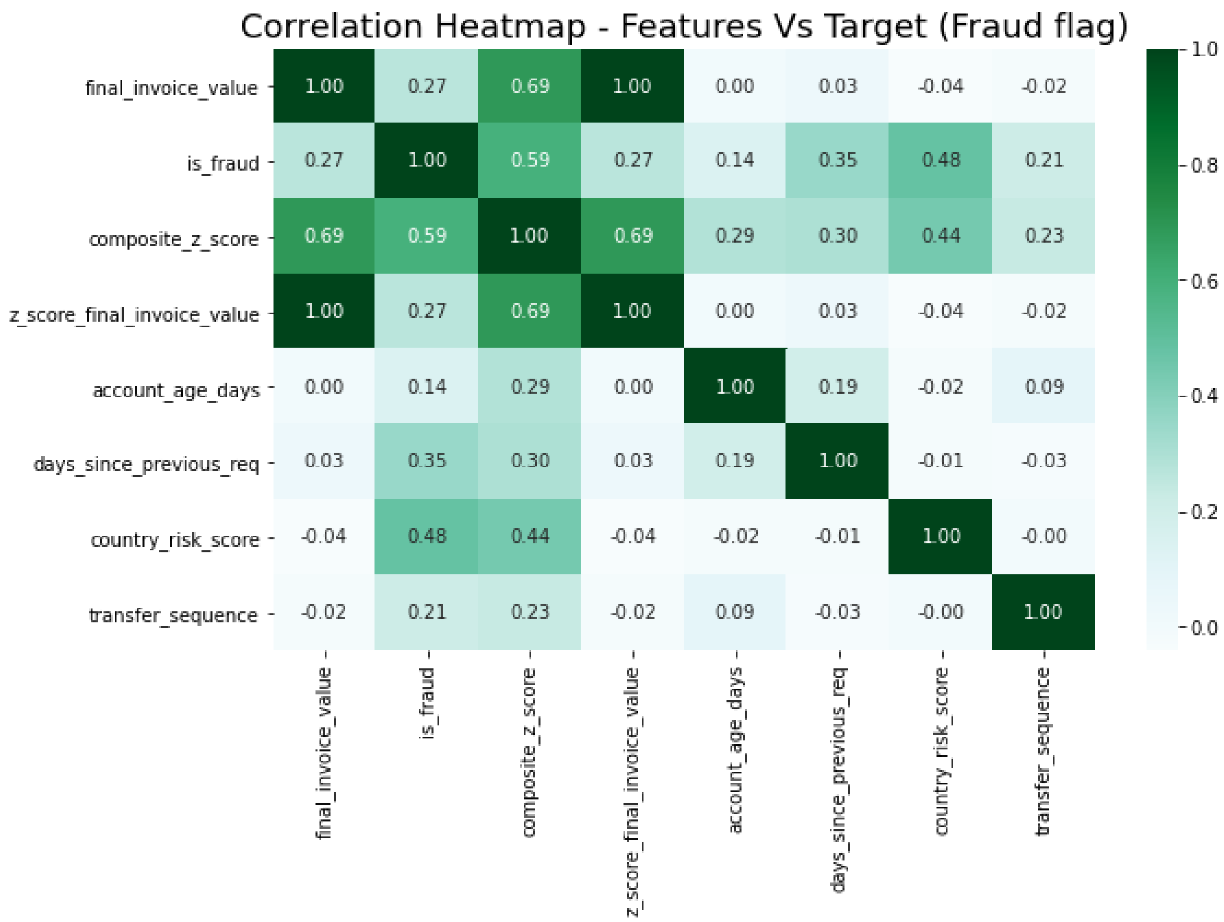
```
# 'is_fraud' is the fraud flag column after detection
total_transactions = len(df_filt)

# Count transactions labeled as fraud
fraud_transactions = df_filt['is_fraud'].sum()

# Fraud identified with z-scores
fraud_transactions_z_score_max = df_filt['suspicious_final_invoice_value'].sum()

# Calculate percentage
fraud_percentage = (fraud_transactions / total_transactions) * 100
z_fraud_percentage = (fraud_transactions_z_score_max / total_transactions) * 100

print(f"Total Transactions: {total_transactions}")
print(f"Fraud Transactions (IsolationForest): {fraud_transactions}")
print(f"Fraud Transactions (Z-scores): {fraud_transactions_z_score_max}")
print(f"Percentage of Fraud Using IsolationForest Model: {fraud_percentage:.2f}%") # Fra
ud percentage with IsolationForest
print(f"Percentage of Fraud Using Z-scores: {z_fraud_percentage:.2f}%") # Fraud percentag
e with z-scores
```

```
Total Transactions: 97981
Fraud Transactions (IsolationForest): 2940
Fraud Transactions (Z-scores): 2548
```

```
Percentage of Fraud Using IsolationForest Model: 3.00%
Percentage of Fraud Using Z-scores: 2.60%
```

In [22]:

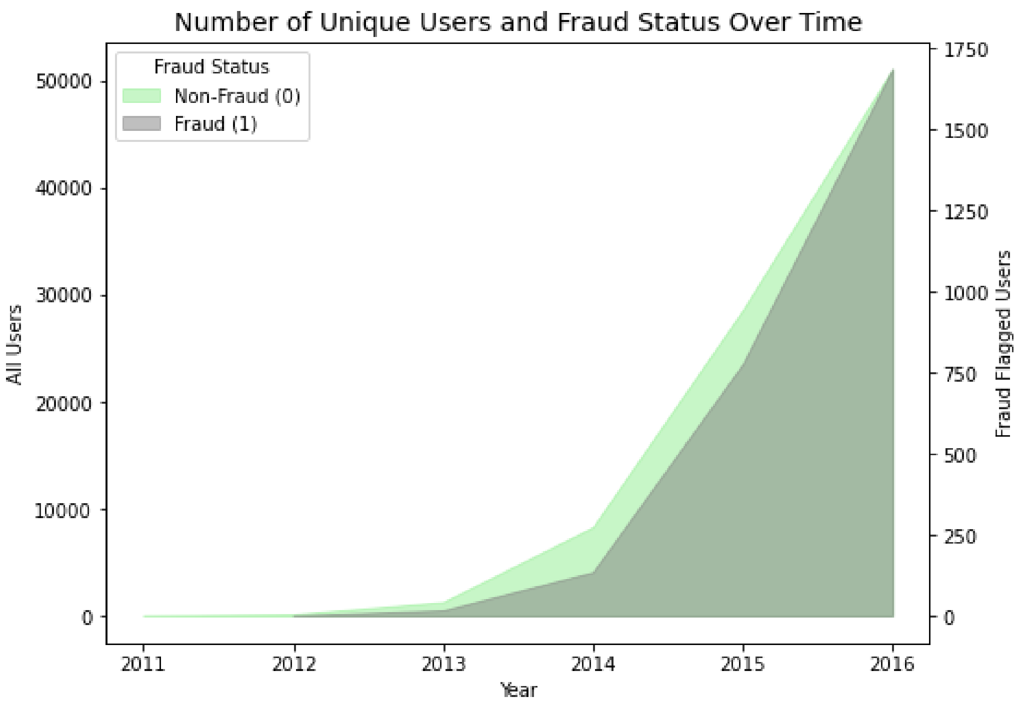
```python
# Select selected features
df_filt_zs = df_filt[[
    'final_invoice_value'
    ,'is_fraud'
    ,'composite_z_score'
    ,'z_score_final_invoice_value'
    ,'account_age_days'
    ,'days_since_previous_req'
    ,'country_risk_score'
    ,'transfer_sequence'
]]

# Plot correlation for the selected features
plt.figure(figsize=(10,6))
corr = df_filt_zs.corr()
#corr = df_filt.corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap='BuGn')

plt.title("Correlation Heatmap - Features Vs Target (Fraud flag)", fontsize=18)

# Customize tick font size
plt.xticks(fontsize=10)  # Increase font size for x-axis ticks
plt.yticks(fontsize=10)  # Increase font size for y-axis ticks
plt.show()
```



In [23]:

```python
# Assuming you have already run the following to get your grouped DataFrame:
df_grouped = df_filt.groupby(['is_fraud', 'year'])['user_id'].nunique().reset_index()

# Pivot the DataFrame to get a column for is_fraud 0 and 1
df_pivot = df_grouped.pivot(index='year', columns='is_fraud', values='user_id')

# Create the plot
fig, ax1 = plt.subplots(figsize=(8, 6))
```

```
# Create a secondary axis for is_fraud
ax2 = ax1.twinx()

# Plot the fraud status (secondary axis) as an area chart
ax1.fill_between(df_pivot.index, df_pivot[0], color='lightgreen', alpha=0.5, label='Non-
Fraud (0)')
ax2.fill_between(df_pivot.index, df_pivot[1], color='grey', alpha=0.5, label='Fraud (1)'
)

# Set labels and titles
ax1.set_title('Number of Unique Users and Fraud Status Over Time', fontsize=14)
ax1.set_xlabel('Year')
ax1.set_ylabel('All Users', color='black')
ax2.set_ylabel('Fraud Flagged Users', color='black')

# Create the combined legend from both axes
handles, labels = ax1.get_legend_handles_labels()
handles2, labels2 = ax2.get_legend_handles_labels()

# Combine the legends
ax1.legend(handles=handles + handles2, labels=labels + labels2, title='Fraud Status', lo
c='upper left')

# Show the plot
plt.show()
```



In [24]:

```
#Create Flag for Fraud Detection on Composite Z-score
df_filt['is_fraud_z90'] = np.where(df_filt["composite_z_score"] > 1.65, 1, 0) # 90% Conf
idence
df_filt['is_fraud_z95'] = np.where(df_filt["composite_z_score"] > 1.96, 1, 0) # 95% conf
idence
df_filt['is_fraud_z99'] = np.where(df_filt["composite_z_score"] > 2.58, 1, 0) # 99% conf
idence
```

In [25]:

```
print(f"95% confidence: {df_filt.is_fraud_z90.sum()}")
print(f"99% confidence: {df_filt.is_fraud_z95.sum()}")
print(f"90% confidence: {df_filt.is_fraud_z99.sum()}")
```

```
95% confidence: 1159
99% confidence: 565
90% confidence: 119
```

In [26]:

In [26]:

```
print(f"Isolation Forest Method: {df_filt.is_fraud.sum()}")
```

```
Isolation Forest Method: 2940
```

In [ ]:

```

```