Assem Albitar

Efficient Data Stream Anomaly Detection project

1. Project Overview

Brief description of the project, goals, and why anomaly detection is important.

2. Algorithm and Approach

Explanation of the Z-score anomaly detection algorithm and the rolling window approach used to detect anomalies.

3. Data Stream Simulation

Details on how the data stream is simulated using regular patterns and noise.

4. Anomaly Detection

Description of how anomalies are detected in real-time based on Z-scores.

5. Optimization and Adaptation

Details on optimizing the detection algorithm using rolling windows and handling concept drift.

6. Visualization

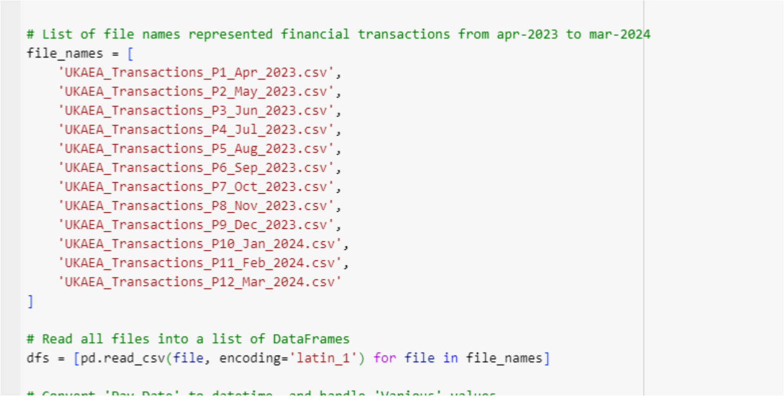
Explanation of how the data stream and detected anomalies are visualized, including plots.

7. Results and Conclusion

Summary of detected anomalies, insights from the process, and performance of the anomaly detection system.

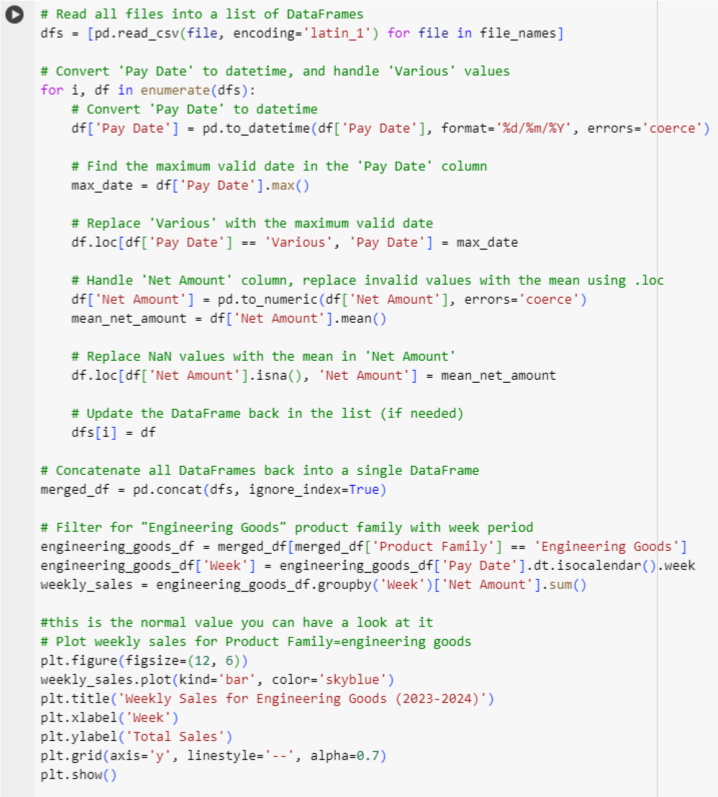
now I will put screenshot for parts of the code and explain it:

1.this is name of the original files (I put the file in .rar file)



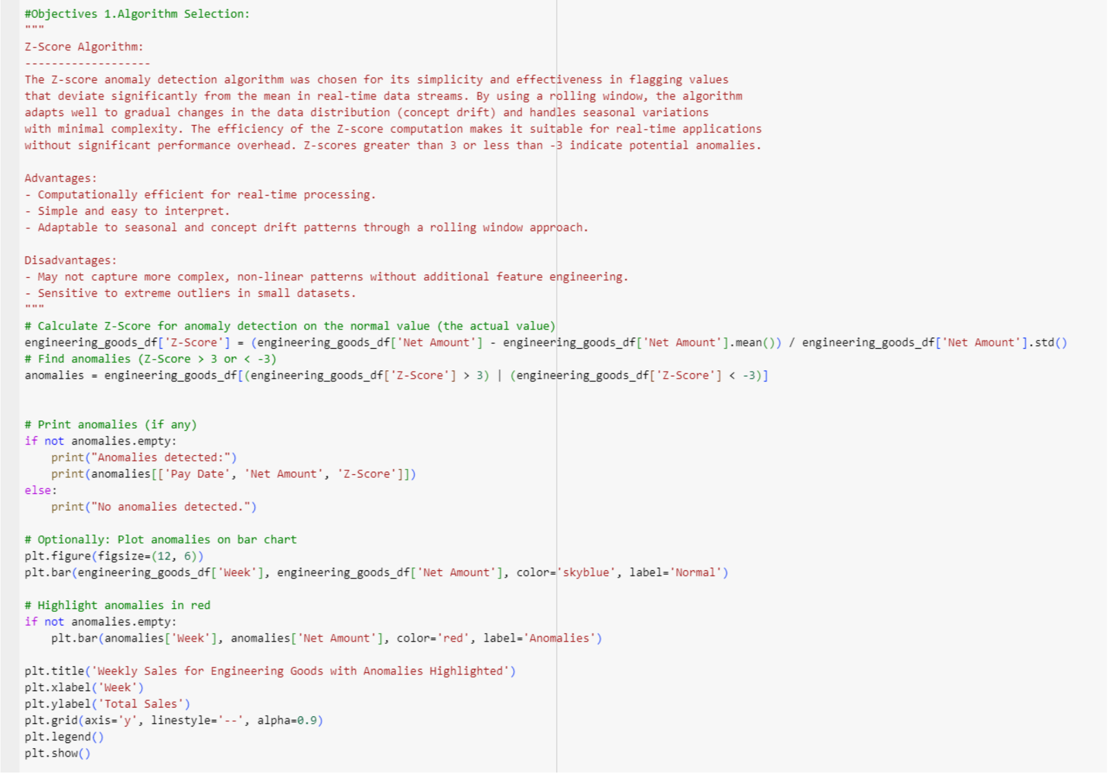
```
# List of file names represented financial transactions from apr-2023 to mar-2024
file_names = [
    'UKAEA_Transactions_P1_Apr_2023.csv',
    'UKAEA_Transactions_P2_May_2023.csv',
    'UKAEA_Transactions_P3_Jun_2023.csv',
    'UKAEA_Transactions_P4_Jul_2023.csv',
    'UKAEA_Transactions_P5_Aug_2023.csv',
    'UKAEA_Transactions_P6_Sep_2023.csv',
    'UKAEA_Transactions_P7_Oct_2023.csv',
    'UKAEA_Transactions_P8_Nov_2023.csv',
    'UKAEA_Transactions_P9_Dec_2023.csv',
    'UKAEA_Transactions_P10_Jan_2024.csv',
    'UKAEA_Transactions_P11_Feb_2024.csv',
    'UKAEA_Transactions_P12_Mar_2024.csv'
]

# Read all files into a list of DataFrames
dfs = [pd.read_csv(file, encoding='latin_1') for file in file_names]

# Convert 'Pay Date' to datetime, and handle 'Various' values
```

2.read the original files and clean them and then merge them togather



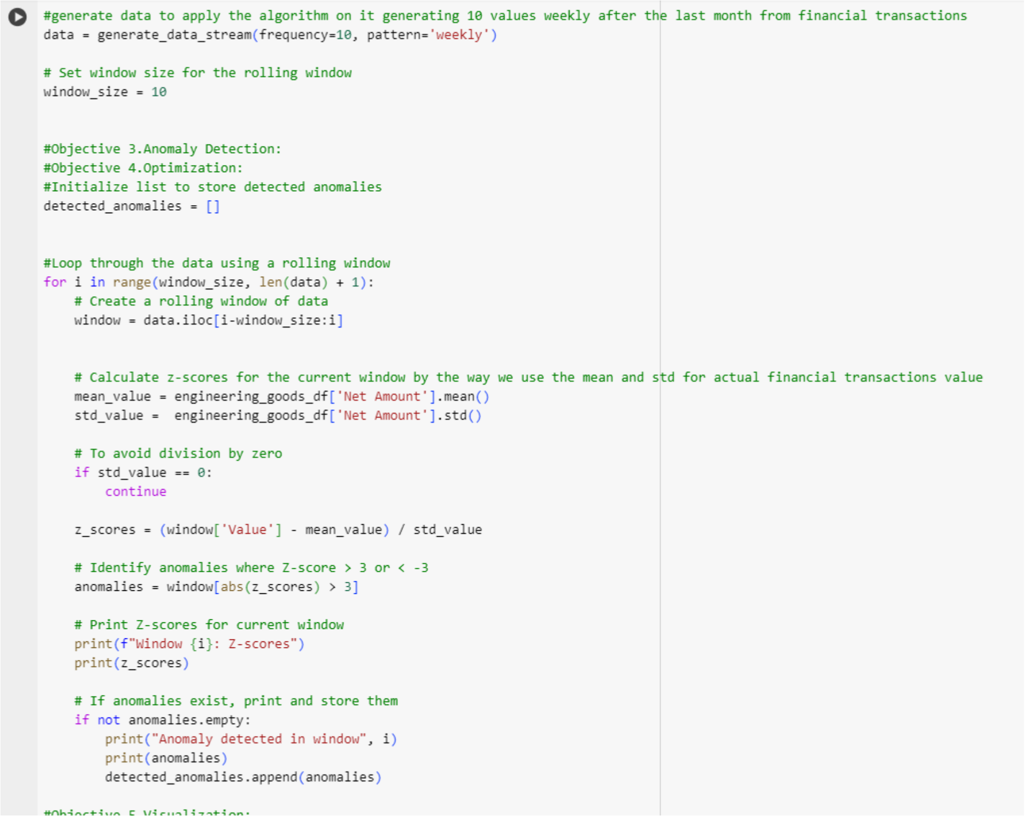
```
# Read all files into a list of DataFrames
dfs = [pd.read_csv(file, encoding='latin_1') for file in file_names]

# Convert 'Pay Date' to datetime, and handle 'Various' values
for i, df in enumerate(dfs):
    # Convert 'Pay Date' to datetime
    df['Pay Date'] = pd.to_datetime(df['Pay Date'], format='%d/%m/%Y', errors='coerce')

    # Find the maximum valid date in the 'Pay Date' column
    max_date = df['Pay Date'].max()

    # Replace 'Various' with the maximum valid date
    df.loc[df['Pay Date'] == 'Various', 'Pay Date'] = max_date

    # Handle 'Net Amount' column, replace invalid values with the mean using .loc
    df['Net Amount'] = pd.to_numeric(df['Net Amount'], errors='coerce')
    mean_net_amount = df['Net Amount'].mean()

    # Replace NaN values with the mean in 'Net Amount'
    df.loc[df['Net Amount'].isna(), 'Net Amount'] = mean_net_amount

    # Update the DataFrame back in the list (if needed)
    dfs[i] = df

# Concatenate all DataFrames back into a single DataFrame
merged_df = pd.concat(dfs, ignore_index=True)

# Filter for "Engineering Goods" product family with week period
engineering_goods_df = merged_df[merged_df['Product Family'] == 'Engineering Goods']
engineering_goods_df['Week'] = engineering_goods_df['Pay Date'].dt.isocalendar().week
weekly_sales = engineering_goods_df.groupby('Week')['Net Amount'].sum()

#this is the normal value you can have a look at it
# Plot weekly sales for Product Family=engineering goods
plt.figure(figsize=(12, 6))
weekly_sales.plot(kind='bar', color='skyblue')
plt.title('Weekly Sales for Engineering Goods (2023-2024)')
plt.xlabel('Week')
plt.ylabel('Total Sales')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

3.apply the z-score algorithm to the original datasets:



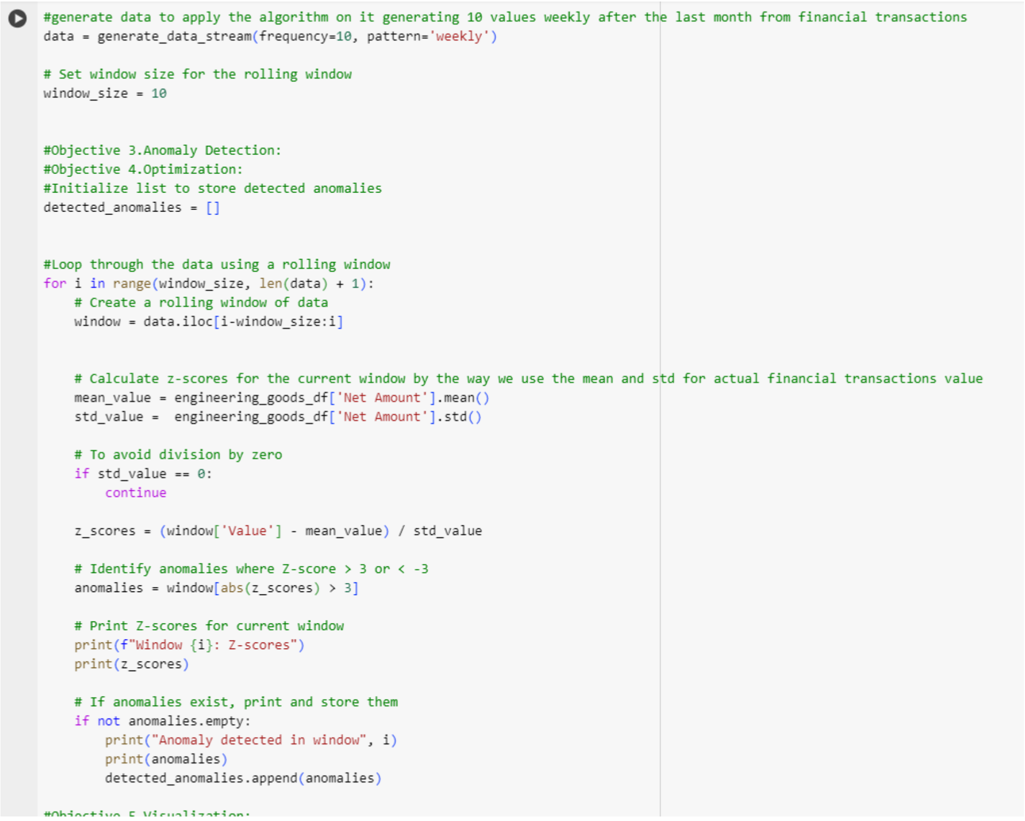
```
#Objectives 1.Algorithm Selection:
"""
Z-Score Algorithm:
------------------
The Z-score anomaly detection algorithm was chosen for its simplicity and effectiveness in flagging values
that deviate significantly from the mean in real-time data streams. By using a rolling window, the algorithm
adapts well to gradual changes in the data distribution (concept drift) and handles seasonal variations
with minimal complexity. The efficiency of the Z-score computation makes it suitable for real-time applications
without significant performance overhead. Z-scores greater than 3 or less than -3 indicate potential anomalies.

Advantages:
- Computationally efficient for real-time processing.
- Simple and easy to interpret.
- Adaptable to seasonal and concept drift patterns through a rolling window approach.

Disadvantages:
- May not capture more complex, non-linear patterns without additional feature engineering.
- Sensitive to extreme outliers in small datasets.
"""
# Calculate Z-Score for anomaly detection on the normal value (the actual value)
engineering_goods_df['Z-Score'] = (engineering_goods_df['Net Amount'] - engineering_goods_df['Net Amount'].mean()) / engineering_goods_df['Net Amount'].std()
# Find anomalies (Z-Score > 3 or < -3)
anomalies = engineering_goods_df[(engineering_goods_df['Z-Score'] > 3) | (engineering_goods_df['Z-Score'] < -3)]


# Print anomalies (if any)
if not anomalies.empty:
    print("Anomalies detected:")
    print(anomalies[['Pay Date', 'Net Amount', 'Z-Score']])
else:
    print("No anomalies detected.")

# Optionally: Plot anomalies on bar chart
plt.figure(figsize=(12, 6))
plt.bar(engineering_goods_df['Week'], engineering_goods_df['Net Amount'], color='skyblue', label='Normal')

# Highlight anomalies in red
if not anomalies.empty:
    plt.bar(anomalies['Week'], anomalies['Net Amount'], color='red', label='Anomalies')

plt.title('Weekly Sales for Engineering Goods with Anomalies Highlighted')
plt.xlabel('Week')
plt.ylabel('Total Sales')
plt.grid(axis='y', linestyle='--', alpha=0.9)
plt.legend()
plt.show()
```

4.this is the function for generating values in real time:



```
#generate data to apply the algorithm on it generating 10 values weekly after the last month from financial transactions
data = generate_data_stream(frequency=10, pattern='weekly')

# Set window size for the rolling window
window_size = 10


#Objective 3.Anomaly Detection:
#Objective 4.Optimization:
#Initialize list to store detected anomalies
detected_anomalies = []


#Loop through the data using a rolling window
for i in range(window_size, len(data) + 1):
    # Create a rolling window of data
    window = data.iloc[i-window_size:i]


    # Calculate z-scores for the current window by the way we use the mean and std for actual financial transactions value
    mean_value = engineering_goods_df['Net Amount'].mean()
    std_value =  engineering_goods_df['Net Amount'].std()

    # To avoid division by zero
    if std_value == 0:
        continue

    z_scores = (window['Value'] - mean_value) / std_value

    # Identify anomalies where Z-score > 3 or < -3
    anomalies = window[abs(z_scores) > 3]

    # Print Z-scores for current window
    print(f"Window {i}: Z-scores")
    print(z_scores)

    # If anomalies exist, print and store them
    if not anomalies.empty:
        print("Anomaly detected in window", i)
        print(anomalies)
        detected_anomalies.append(anomalies)

#Objective 5.Visualization:
```

5.call function to generate values and then apply the algorithm on them to see what values are anomaly and what not



```
#generate data to apply the algorithm on it generating 10 values weekly after the last month from financial transactions
data = generate_data_stream(frequency=10, pattern='weekly')

# Set window size for the rolling window
window_size = 10


#Objective 3.Anomaly Detection:
#Objective 4.Optimization:
#Initialize list to store detected anomalies
detected_anomalies = []


#Loop through the data using a rolling window
for i in range(window_size, len(data) + 1):
    # Create a rolling window of data
    window = data.iloc[i-window_size:i]


    # Calculate z-scores for the current window by the way we use the mean and std for actual financial transactions value
    mean_value = engineering_goods_df['Net Amount'].mean()
    std_value =  engineering_goods_df['Net Amount'].std()

    # To avoid division by zero
    if std_value == 0:
        continue

    z_scores = (window['Value'] - mean_value) / std_value

    # Identify anomalies where Z-score > 3 or < -3
    anomalies = window[abs(z_scores) > 3]

    # Print Z-scores for current window
    print(f"Window {i}: Z-scores")
    print(z_scores)

    # If anomalies exist, print and store them
    if not anomalies.empty:
        print("Anomaly detected in window", i)
        print(anomalies)
        detected_anomalies.append(anomalies)

#Objective 5.Visualization:
```