Enhanced Audit and Error Logging System

Overview

Your Repricing Automation Program now has a comprehensive audit and error logging system that will help you assist users effectively.

■ What the System Logs

*1. **User Identification & Session Tracking***

• **Username**: Captured via `getpass.getuser()` (e.g., "DamionMorrison")
• **Computer Name**: Host machine identifier (e.g., "L01403-DATA")
• **Session Start/End**: Complete session lifecycle tracking
• **System Information**: Python version, OS details for troubleshooting

*2. **File Operations***

• **File Imports**: Tracks when users import File1, File2, and templates
• **File Paths**: Records exact file locations accessed
• **File Errors**: Logs permission issues, missing files, corrupt data

*3. **Process Tracking***

• **Process Start/Stop**: When repricing processes begin and end
• **Process Errors**: Failures during data processing, merging, or calculations
• **User Actions**: What the user was trying to do when errors occurred

*4. **Error Categories for Support***

• **USER_ERROR**: User-related issues (wrong files, invalid inputs)
• **SYSTEM_ERROR**: Application/system failures (memory, crashes)
• **FILE_ERROR**: File access, permission, or format issues
• **DATA_ERROR**: Data processing, validation, or calculation errors

■ Support Tools Available

*1. **Error Analysis Tool** (`safe_error_analysis.py`)*

```
python safe_error_analysis.py
```

Provides:
• Summary of all errors in the last 7 days
• Breakdown by error type and affected users
• Most problematic scripts/functions
• Recent successful operations

*2. **User Support Report** (`user_support_report.py`)*

```
python user_support_report.py
# Or modify to check specific user:
# python -c "from user_support_report import generate_user_support_report; generate_user_suppo
```

Provides:
• User-specific error history
• Detailed error context and system information
• Timeline of issues for pattern identification

*3. **Real-time Audit Log Viewer** (In Application)*

• Click "Shared Audit Log" button in the application
• Live updating view of all user activities
• Search functionality to filter specific users or error types

■ Example Error Log Entry

```
Timestamp,User,Script,Message,Status
2025-07-10 11:18:32,DamionMorrison,File1Import,"USER ERROR - FILE_NOT_FOUND | User: DamionMorr
```

■ How to Help Users

*1. **When a User Reports an Issue:**

1. Ask for their **username** and **approximate time** of the error
2. Run the support report: `generate_user_support_report("Username")`
3. Check the audit log for their recent activities

*2. **Common Issue Patterns:**

File Errors

• **Symptoms**: "Cannot import file" or "File not found"
• **Check**: File paths, permissions, OneDrive sync status
• **Log Shows**: Exact file path, file size, error details

Processing Errors

• **Symptoms**: "Process failed" or crashes during repricing
• **Check**: Data format, memory usage, Python environment
• **Log Shows**: Processing stage, data context, system resources

User Errors

• **Symptoms**: Wrong template, invalid data format
• **Check**: File structure, column headers, data types
• **Log Shows**: User actions, file details, validation failures

*3. **Support Workflow:**

```
1. Get user details (name, time of issue)
   ↓
2. Run: python -c "from user_support_report import generate_user_support_report; generate_user
   ↓
3. Review error details and system information
   ↓
4. Check audit log for session context
   ↓
5. Provide targeted assistance based on error type
```

■ Log File Location

```
%OneDrive%/True Community - Data Analyst/Python Repricing Automation Program/Logs/audit_log.cs
```

■ Key Information Captured for Each User

System Context

• Python version compatibility
• Operating system details
• Computer/hostname for environment identification
• Memory and processing context

User Actions

• Exact sequence of operations performed
• Files attempted to access
• Processing options selected
• Timestamp of each action

Error Context

• What the user was trying to accomplish
• Which file caused the issue
• System state at time of error
• Complete error messages and stack traces

■ Quick Commands for Support

Check Recent Errors (All Users)

```
python safe_error_analysis.py
```

Check Specific User Errors

```
python -c "from safe_error_analysis import get_user_errors_safe; errors = get_user_errors_safe
```

View Raw Audit Log (Recent Entries)

```
python check_audit.py
```

This comprehensive logging system ensures you have all the information needed to quickly diagnose and resolve user issues, making support much more efficient and effective!