Alteryx Workflow Automation Strategy

CLIP Credit Line Increase Process - Migration & Automation Plan

---

Executive Summary

This document outlines the strategy to automate existing Alteryx workflows for the CLIP process, including migration paths, automation frameworks, and implementation approaches that maintain business logic while improving scalability and maintainability.

Current State: Typical Alteryx CLIP Workflow

```
┌─────────────────────────────────────────────────────────────┐
│              TYPICAL ALTERYX CLIP WORKFLOW                  │
└─────────────────────────────────────────────────────────────┘

INPUT STAGE
├─► Input Data Tool (1): Member Demographics
├─► Input Data Tool (2): Account Data
├─► Input Data Tool (3): Transaction History
├─► Input Data Tool (4): Credit Scores
└─► Input Data Tool (5): Exclusion List (Excel)

    ↓

DATA PREPARATION
├─► Join Tool: Merge member + account data
├─► Filter Tool: Remove closed accounts
├─► Select Tool: Choose relevant columns
├─► Formula Tool: Calculate account age
├─► Formula Tool: Calculate utilization
└─► Sort Tool: Order by member ID

    ↓

ELIGIBILITY FILTERING
├─► Filter Tool: Account age >= 12 months
├─► Filter Tool: Credit score >= 650
```

├─► Filter Tool: No 30+ day delinquencies
├─► Filter Tool: Current balance > $500
├─► Join Tool: Exclude members from exclusion list
└─► Formula Tool: Create eligibility flag

↓

ANALYSIS & CALCULATIONS
├─► Summarize Tool: Calculate avg utilization (6 months)
├─► Formula Tool: Calculate recommended increase %
├─► Formula Tool: Calculate new credit limit
├─► Formula Tool: Calculate post-increase utilization
├─► Multi-Row Formula Tool: Identify trends
└─► Formula Tool: Assign risk categories

↓

BUSINESS RULES APPLICATION
├─► Filter Tool: Apply maximum limit cap
├─► Formula Tool: Apply tiered increase rules
├─► Join Tool: Check cross-account exposure
├─► Filter Tool: Apply income-to-limit ratio
└─► Formula Tool: Generate approval tier

↓

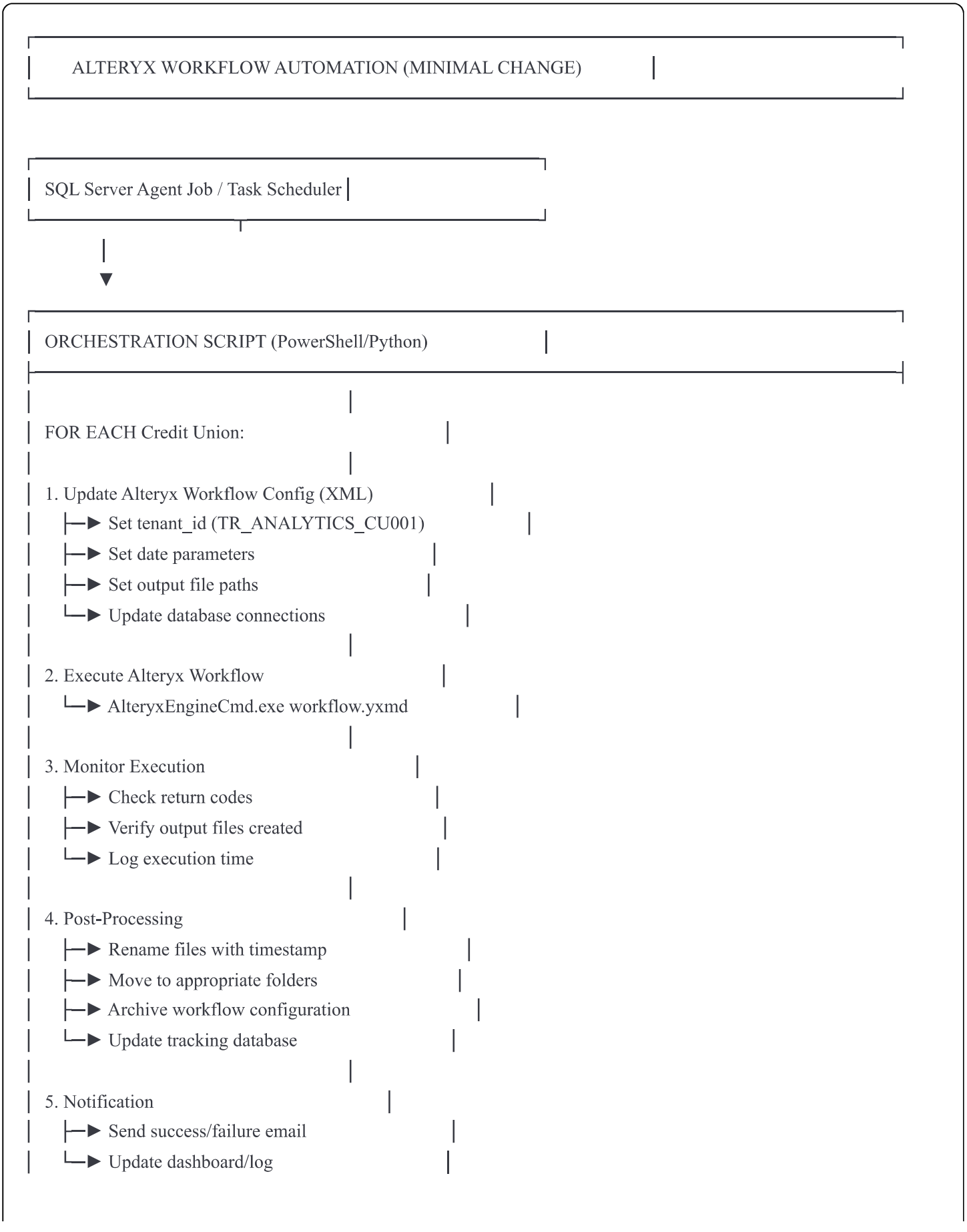
OUTPUT GENERATION
├─► Output Data Tool (1): Approved List → CSV
├─► Output Data Tool (2): Rejected List → CSV
├─► Output Data Tool (3): Analysis Report → Excel
├─► Output Data Tool (4): Executive Summary → Excel
└─► Email Tool: Send notification

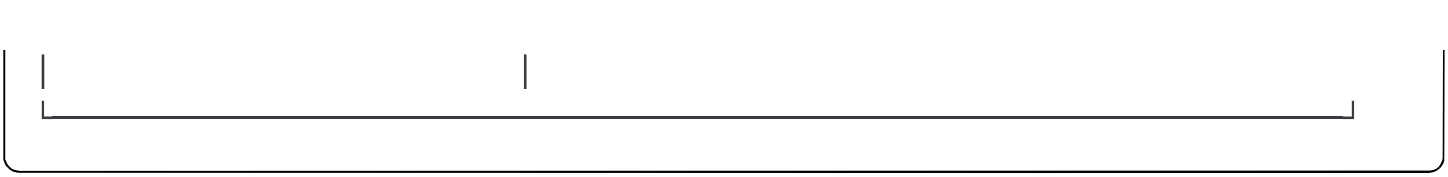
↓

END WORKFLOW

Automation Approach 1: Direct Alteryx Automation

Strategy: Automate Alteryx execution via Command Line





Implementation Files

Folder Structure for Alteryx Automation:

```
CLIP_ALTERYX_AUTOMATION/
│
├── workflows/
│   ├── templates/
│   │   └── CLIP_Master_Template.yxmd       # Template workflow
│   │
│   └── runtime/
│       ├── CU_001_CLIP.yxmd                # Generated workflow for CU_001
│       ├── CU_002_CLIP.yxmd                # Generated workflow for CU_002
│       └── CU_003_CLIP.yxmd
│
├── config/
│   ├── credit_unions.json                  # CU-specific parameters
│   ├── database_connections.json           # Connection strings
│   └── email_config.json                   # Notification settings
│
├── scripts/
│   ├── orchestrator.ps1                    # PowerShell orchestrator
│   ├── workflow_generator.py               # Generates CU-specific workflows
│   ├── alteryx_executor.py                 # Python wrapper for execution
│   └── post_processor.py                   # File handling & notifications
│
├── inputs/
│   ├── exclusion_lists/
│   │   ├── CU_001_exclusions.xlsx
│   │   └── CU_002_exclusions.xlsx
│   │
│   └── reference_data/
│       └── product_configs.xlsx
│
├── outputs/
│   ├── YYYY-MM/
│   │   ├── CU_001/
│   │   │   ├── approved_list.csv
│   │   │   ├── rejected_list.csv
│   │   │   ├── analysis_report.xlsx
│   │   │   └── executive_summary.xlsx
│   │   │
│   │   └── CU_002/
│   │
│   └── archive/
│
├── logs/
```

```
│   ├── execution_logs/
│   │   └── YYYY-MM-DD_execution.log
│   │
│   ├── alteryx_logs/
│   │   └── YYYY-MM-DD_alteryx.log
│   │
│   └── error_logs/
│
└── monitoring/
    ├── dashboard_data.json
    └── performance_metrics.csv
```

PowerShell Orchestration Script Example

```powershell
```

```powershell
# orchestrator.ps1
# Alteryx CLIP Workflow Automation Orchestrator

param(
    [string]$ConfigFile = "config\credit_unions.json",
    [string]$RunDate = (Get-Date -Format "yyyy-MM-dd"),
    [string]$CreditUnion = "ALL"
)

# Configuration
$AlteryxEnginePath = "C:\Program Files\Alteryx\bin\AlteryxEngineCmd.exe"
$LogPath = "logs\execution_logs\$RunDate`_execution.log"
$OutputBasePath = "outputs\$((Get-Date).ToString('yyyy-MM'))"

# Initialize logging
function Write-Log {
    param([string]$Message)
    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $logMessage = "$timestamp - $Message"
    Add-Content -Path $LogPath -Value $logMessage
    Write-Host $logMessage
}

Write-Log "=== CLIP Automation Started ==="
Write-Log "Run Date: $RunDate"

# Load configuration
$config = Get-Content $ConfigFile | ConvertFrom-Json
$creditUnions = if ($CreditUnion -eq "ALL") { $config.credit_unions } else { $config.credit_unions | Where-Object { $_.cu_

# Process each credit union
foreach ($cu in $creditUnions) {
    Write-Log "Processing Credit Union: $($cu.cu_name) ($($cu.cu_id))"

    # Create output directory
    $cuOutputPath = "$OutputBasePath\$($cu.cu_id)"
    New-Item -ItemType Directory -Force -Path $cuOutputPath | Out-Null

    # Generate workflow from template
    Write-Log "Generating workflow for $($cu.cu_id)"
    python scripts\workflow_generator.py --cu-id $cu.cu_id --run-date $RunDate --output-path $cuOutputPath

    $workflowPath = "workflows\runtime\$($cu.cu_id)_CLIP.yxmd"
```

```
    if (Test-Path $workflowPath) {
        # Execute Alteryx workflow
        Write-Log "Executing Alteryx workflow: $workflowPath"

        $startTime = Get-Date
        $process = Start-Process -FilePath $AlteryxEnginePath -ArgumentList $workflowPath -Wait -PassThru -NoNewWindow
        $endTime = Get-Date
        $duration = ($endTime - $startTime).TotalMinutes

        if ($process.ExitCode -eq 0) {
            Write-Log "SUCCESS: Workflow completed in $([math]::Round($duration, 2)) minutes"

            # Post-processing
            Write-Log "Running post-processing"
            python scripts\post_processor.py --cu-id $cu.cu_id --output-path $cuOutputPath

            # Send success notification
            python scripts\send_notification.py --cu-id $cu.cu_id --status "SUCCESS" --duration $duration

        } else {
            Write-Log "ERROR: Workflow failed with exit code $($process.ExitCode)"

            # Send failure notification
            python scripts\send_notification.py --cu-id $cu.cu_id --status "FAILED" --exit-code $process.ExitCode
        }

    } else {
        Write-Log "ERROR: Workflow file not found: $workflowPath"
    }

    Write-Log "---"
}

Write-Log "=== CLIP Automation Completed ==="
```

Python Workflow Generator

python

```python
# workflow_generator.py
# Generates Credit Union-specific Alteryx workflows from template

import json
import xml.etree.ElementTree as ET
import argparse
from datetime import datetime

def generate_workflow(cu_id, run_date, output_path):
    """
    Generate CU-specific Alteryx workflow from template
    """

    # Load CU configuration
    with open('config/credit_unions.json', 'r') as f:
        config = json.load(f)

    cu_config = next((cu for cu in config['credit_unions'] if cu['cu_id'] == cu_id), None)

    if not cu_config:
        raise ValueError(f"Credit Union {cu_id} not found in configuration")

    # Load template workflow (XML)
    tree = ET.parse('workflows/templates/CLIP_Master_Template.yxmd')
    root = tree.getroot()

    # Update database connections
    for connection in root.findall(".//Connection"):
        connection_string = connection.find("ConnectionString")
        if connection_string is not None:
            # Replace placeholder with actual tenant schema
            original = connection_string.text
            updated = original.replace(
                "TR_ANALYTICS_PLACEHOLDER",
                cu_config['tenant_id']
            )
            connection_string.text = updated

    # Update input file paths
    for input_tool in root.findall(".//Tool[@ToolID='1']"):  # Input Data tools
        file_path = input_tool.find(".//FilePath")
        if file_path is not None and "exclusion" in file_path.text.lower():
            file_path.text = f"inputs/exclusion_lists/{cu_id}_exclusions.xlsx"
```

```
# Update output file paths
output_mappings = {
    "approved_list": f"{output_path}/{cu_id}_approved_list_{run_date}.csv",
    "rejected_list": f"{output_path}/{cu_id}_rejected_list_{run_date}.csv",
    "analysis_report": f"{output_path}/{cu_id}_analysis_report_{run_date}.xlsx",
    "executive_summary": f"{output_path}/{cu_id}_executive_summary_{run_date}.xlsx"
}

for output_tool in root.findall(".//Tool[@ToolType='OutputData']"):
    output_name = output_tool.get('Name', '').lower().replace(' ', '_')
    if output_name in output_mappings:
        file_path = output_tool.find(".//FilePath")
        if file_path is not None:
            file_path.text = output_mappings[output_name]

# Update formula tools with CU-specific parameters
for formula_tool in root.findall(".//Tool[@ToolType='Formula']"):
    formula_expression = formula_tool.find(".//Expression")
    if formula_expression is not None:
        expr = formula_expression.text

        # Replace eligibility parameters
        expr = expr.replace("{{MIN_CREDIT_SCORE}}", str(cu_config['min_credit_score']))
        expr = expr.replace("{{MIN_ACCOUNT_AGE}}", str(cu_config['min_account_age_months']))
        expr = expr.replace("{{MAX_UTILIZATION}}", str(cu_config['max_utilization_pct']))

        # Replace increase strategy parameters
        expr = expr.replace("{{INCREASE_PCT_TIER1}}", str(cu_config['increase_strategy']['tier1_pct']))
        expr = expr.replace("{{INCREASE_PCT_TIER2}}", str(cu_config['increase_strategy']['tier2_pct']))
        expr = expr.replace("{{MAX_LIMIT}}", str(cu_config['max_credit_limit']))

        formula_expression.text = expr

# Update email tool with CU-specific recipients
for email_tool in root.findall(".//Tool[@ToolType='Email']"):
    to_field = email_tool.find(".//To")
    if to_field is not None:
        to_field.text = ';'.join(cu_config['email_recipients'])

    subject_field = email_tool.find(".//Subject")
    if subject_field is not None:
        subject_field.text = f"CLIP Analysis Complete - {cu_config['cu_name']} - {run_date}"
```

```
    # Save generated workflow
    output_workflow_path = f"workflows/runtime/{cu_id}_CLIP.yxmd"
    tree.write(output_workflow_path, encoding='utf-8', xml_declaration=True)

    print(f"Generated workflow: {output_workflow_path}")
    return output_workflow_path

if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='Generate CU-specific Alteryx workflow')
    parser.add_argument('--cu-id', required=True, help='Credit Union ID')
    parser.add_argument('--run-date', required=True, help='Run date (YYYY-MM-DD)')
    parser.add_argument('--output-path', required=True, help='Output directory path')

    args = parser.parse_args()

    generate_workflow(args.cu_id, args.run_date, args.output_path)
```

Configuration File Structure

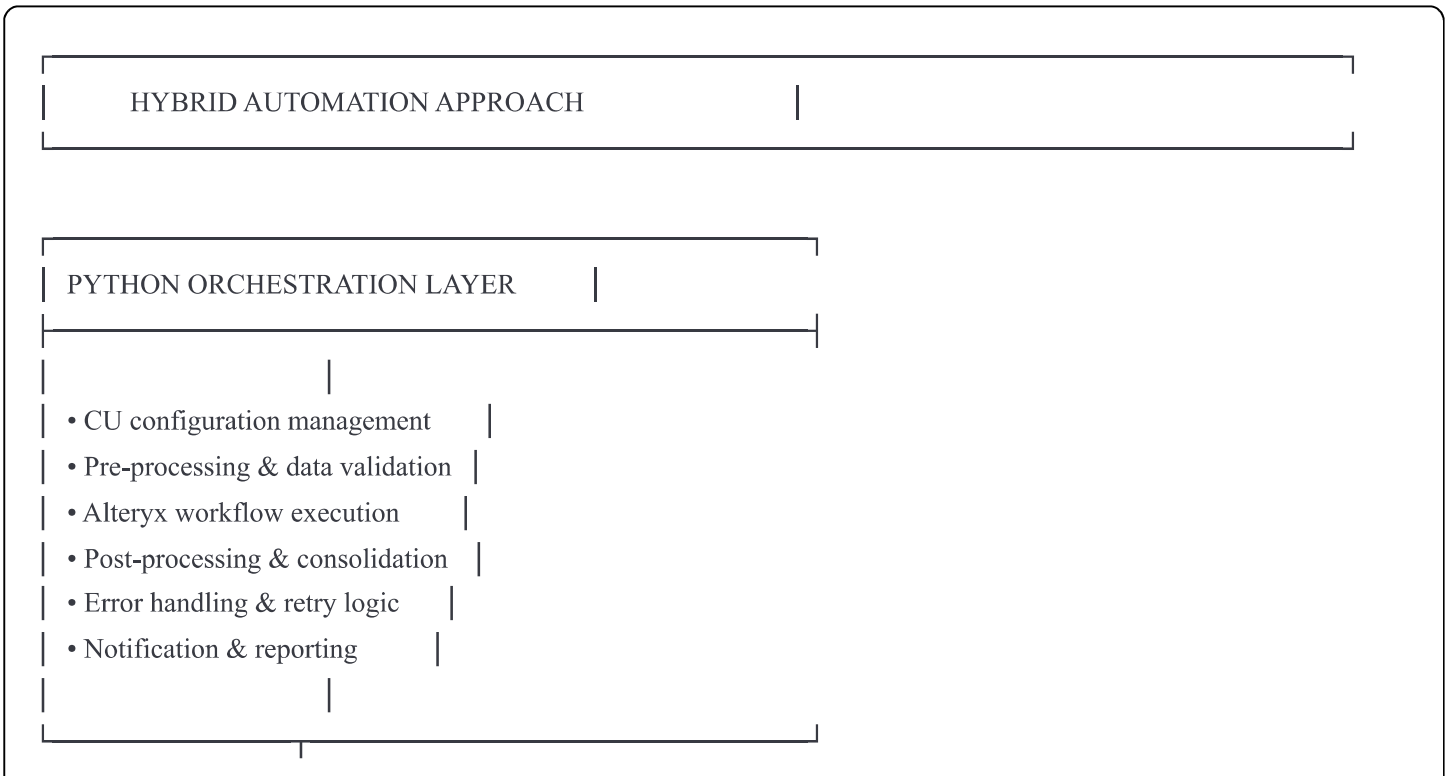
```
json
```

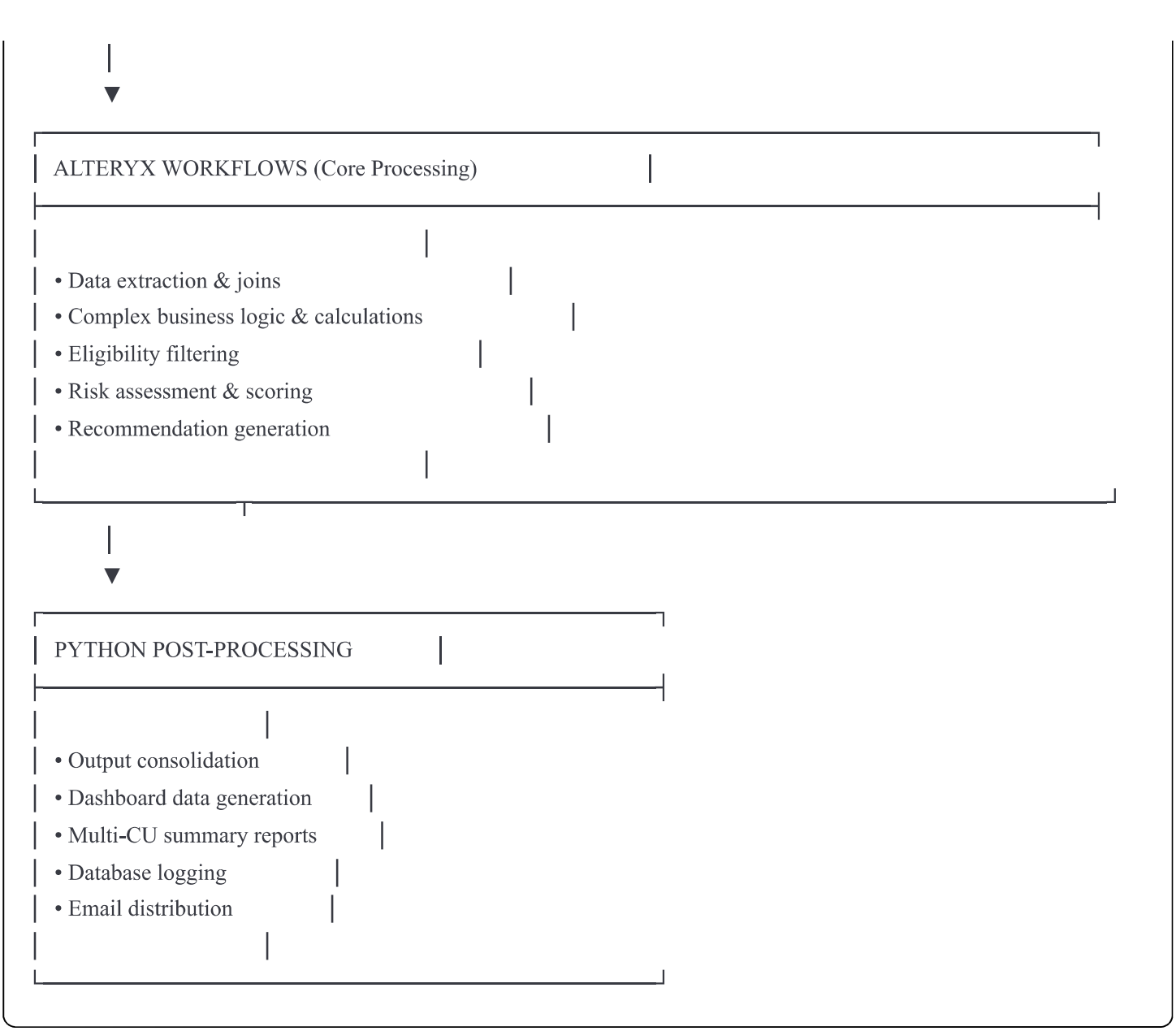
```
{
 "credit_unions": [
  {
   "cu_id": "CU_001",
   "cu_name": "First Example FCU",
   "tenant_id": "TR_ANALYTICS_FIRSTEXAMPLE",
   "min_credit_score": 650,
   "min_account_age_months": 12,
   "max_utilization_pct": 95,
   "increase_strategy": {
    "method": "tiered",
    "tier1_pct": 50,
    "tier1_credit_score_min": 750,
    "tier2_pct": 35,
    "tier2_credit_score_min": 700,
    "tier3_pct": 25,
    "tier3_credit_score_min": 650
   },
   "max_credit_limit": 25000,
   "email_recipients": [
    "analyst@firstexample.com",
    "manager@firstexample.com"
   ],
   "enabled": true
  },
  {
   "cu_id": "CU_002",
   "cu_name": "Second Example CU",
   "tenant_id": "TR_ANALYTICS_SECONDEXAMPLE",
   "min_credit_score": 680,
   "min_account_age_months": 18,
   "max_utilization_pct": 90,
   "increase_strategy": {
    "method": "tiered",
    "tier1_pct": 40,
    "tier1_credit_score_min": 760,
    "tier2_pct": 30,
    "tier2_credit_score_min": 720,
    "tier3_pct": 20,
    "tier3_credit_score_min": 680
   },
   "max_credit_limit": 20000,
   "email_recipients": [
```

```
      "data@secondexample.org"
    ],
    "enabled": true
  }
 ],

 "database_config": {
  "snowflake_account": "your_account.snowflakecomputing.com",
  "database": "CREDIT_UNION_DATA",
  "warehouse": "ANALYTICS_WH",
  "role": "ANALYTICS_ROLE"
 },

 "execution_config": {
  "max_parallel_workflows": 3,
  "timeout_minutes": 60,
  "retry_on_failure": true,
  "max_retries": 2
 }
}
```

Automation Approach 2: Hybrid (Alteryx + Python)

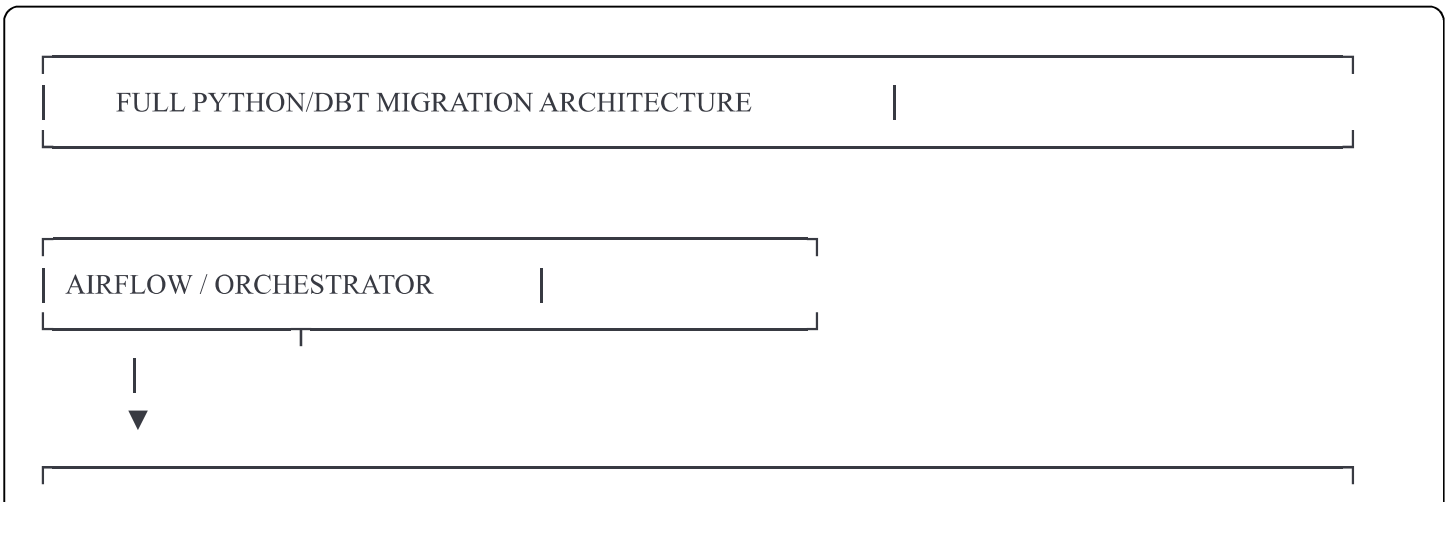
Strategy: Keep Alteryx for complex transformations, add Python orchestration layer

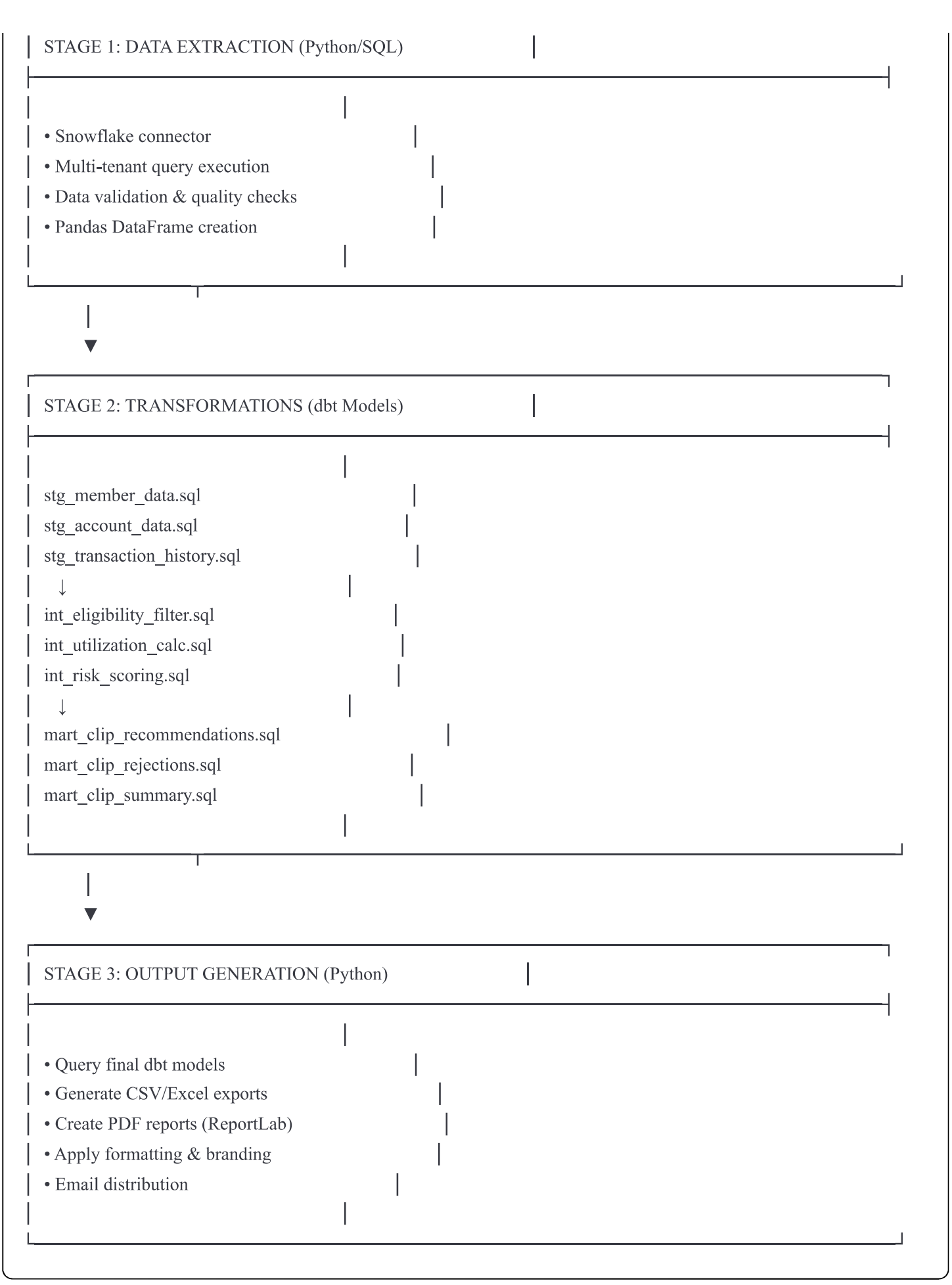




Automation Approach 3: Full Migration to Python/SQL

Strategy: Replicate Alteryx logic in Python/dbt, eliminate Alteryx dependency




STAGE 1: DATA EXTRACTION (Python/SQL)
• Snowflake connector
• Multi-tenant query execution
• Data validation & quality checks
• Pandas DataFrame creation
STAGE 2: TRANSFORMATIONS (dbt Models)
stg_member_data.sql
stg_account_data.sql
stg_transaction_history.sql
↓
int_eligibility_filter.sql
int_utilization_calc.sql
int_risk_scoring.sql
↓
mart_clip_recommendations.sql
mart_clip_rejections.sql
mart_clip_summary.sql
STAGE 3: OUTPUT GENERATION (Python)
• Query final dbt models
• Generate CSV/Excel exports
• Create PDF reports (ReportLab)
• Apply formatting & branding
• Email distribution

Side-by-Side: Alteryx Tool → Python/SQL Equivalent

Alteryx Tool	Python Equivalent	SQL/dbt Equivalent
Input Data	`pd.read_sql()` or `snowflake.connector`	`{{ source() }}` in dbt
Filter	`df[df['column'] > value]`	`WHERE` clause
Select	`df[['col1', 'col2']]`	`SELECT col1, col2`
Formula	`df['new_col'] = df['col1'] * 2`	`col1 * 2 AS new_col`
Join	`pd.merge(df1, df2)`	`JOIN` clause
Sort	`df.sort_values()`	`ORDER BY`
Summarize	`df.groupby().agg()`	`GROUP BY` with aggregations
Multi-Row Formula	`df['new'] = df['col'].shift()`	`LAG()` / `LEAD()` window functions
Union	`pd.concat([df1, df2])`	`UNION ALL`
Output Data	`df.to_csv()` / `df.to_excel()`	`CREATE TABLE AS`
Email	`smtplib` or `sendgrid`	Post-processing script

Migration Mapping: Typical CLIP Workflow

Alteryx Workflow → Python/dbt Equivalent

```python
```

```
# ================================================================
# ALTERYX INPUT TOOLS → PYTHON DATA EXTRACTION
# ================================================================

import pandas as pd
from snowflake.connector import connect

def extract_member_data(tenant_id, run_date):
    """
    Replaces: Alteryx Input Data Tool (Member Demographics)
    """
    conn = connect(
        account='your_account.snowflakecomputing.com',
        user='your_user',
        password='your_password',
        database='CREDIT_UNION_DATA',
        warehouse='ANALYTICS_WH'
    )

    query = f"""
    SELECT
        member_id,
        first_name,
        last_name,
        date_of_birth,
        credit_score,
        annual_income,
        member_since_date
    FROM {tenant_id}.MEMBER
    WHERE member_status = 'ACTIVE'
        AND date_of_birth <= DATEADD(year, -18, CURRENT_DATE())
    """

    df = pd.read_sql(query, conn)
    conn.close()

    return df

def extract_account_data(tenant_id, run_date):
    """
    Replaces: Alteryx Input Data Tool (Account Data)
    """
    conn = connect(
```

```
        account='your_account.snowflakecomputing.com',
        user='your_user',
        password='your_password',
        database='CREDIT_UNION_DATA',
        warehouse='ANALYTICS_WH'
    )

    query = f"""
    SELECT
        account_id,
        member_id,
        account_type,
        account_open_date,
        credit_limit,
        current_balance,
        available_credit,
        payment_due_date,
        last_payment_date,
        last_payment_amount,
        days_past_due,
        charge_off_flag
    FROM {tenant_id}.ACCOUNT
    WHERE account_type = 'CREDIT_CARD'
        AND account_status = 'OPEN'
    """

    df = pd.read_sql(query, conn)
    conn.close()

    return df

# ================================================================
# ALTERYX JOIN TOOLS → PANDAS MERGE
# ================================================================

def merge_member_account_data(member_df, account_df):
    """
    Replaces: Alteryx Join Tool
    """
    merged_df = pd.merge(
        member_df,
        account_df,
        on='member_id',
        how='inner'
```

```
    )

    return merged_df

# ================================================================
# ALTERYX FORMULA TOOLS → PANDAS CALCULATIONS
# ================================================================

def calculate_account_metrics(df, run_date):
    """
    Replaces: Multiple Alteryx Formula Tools
    """
    # Account age calculation
    df['account_age_months'] = (
        (pd.to_datetime(run_date) - pd.to_datetime(df['account_open_date']))
        .dt.days / 30.44
    ).astype(int)

    # Utilization calculation
    df['utilization_pct'] = (
        (df['current_balance'] / df['credit_limit'] * 100)
        .round(2)
    )

    # Available credit percentage
    df['available_credit_pct'] = (
        (df['available_credit'] / df['credit_limit'] * 100)
        .round(2)
    )

    # Days since last payment
    df['days_since_last_payment'] = (
        (pd.to_datetime(run_date) - pd.to_datetime(df['last_payment_date']))
        .dt.days
    )

    return df

# ================================================================
# ALTERYX FILTER TOOLS → PANDAS BOOLEAN INDEXING
# ================================================================

def apply_eligibility_filters(df, config):
    """
```

```
    Replaces: Multiple Alteryx Filter Tools
    """
    # Account age filter
    eligible_df = df[df['account_age_months'] >= config['min_account_age_months']].copy()

    # Credit score filter
    eligible_df = eligible_df[eligible_df['credit_score'] >= config['min_credit_score']]

    # Delinquency filter
    eligible_df = eligible_df[eligible_df['days_past_due'] == 0]

    # Minimum balance filter
    eligible_df = eligible_df[eligible_df['current_balance'] >= config['min_current_balance']]

    # Utilization filter
    eligible_df = eligible_df[eligible_df['utilization_pct'] <= config['max_utilization_pct']]

    # Charge-off exclusion
    eligible_df = eligible_df[eligible_df['charge_off_flag'] == False]

    # Create rejection log for filtered records
    rejected_df = df[~df['account_id'].isin(eligible_df['account_id'])].copy()
    rejected_df['rejection_reason'] = 'Failed eligibility criteria'

    return eligible_df, rejected_df

# ============================================================
# ALTERYX SUMMARIZE TOOL → PANDAS GROUPBY
# ============================================================

def calculate_historical_utilization(df, transaction_df):
    """
    Replaces: Alteryx Summarize Tool for historical averages
    """
    # Calculate 6-month average utilization
    utilization_history = transaction_df.groupby('account_id').agg({
        'utilization_pct': ['mean', 'std', 'min', 'max'],
        'balance': 'mean'
    }).reset_index()

    utilization_history.columns = [
        'account_id',
        'avg_utilization_6mo',
        'std_utilization_6mo',
```

```
        'min_utilization_6mo',
        'max_utilization_6mo',
        'avg_balance_6mo'
    ]

    # Merge back to main dataframe
    df = pd.merge(df, utilization_history, on='account_id', how='left')

    return df

# ============================================================
# ALTERYX MULTI-ROW FORMULA → PANDAS SHIFT/LAG
# ============================================================

def identify_utilization_trends(df):
    """
    Replaces: Alteryx Multi-Row Formula Tool for trend analysis
    """
    # Sort by member and date
    df = df.sort_values(['member_id', 'statement_date'])

    # Create lagged columns
    df['prev_utilization'] = df.groupby('member_id')['utilization_pct'].shift(1)
    df['prev_2_utilization'] = df.groupby('member_id')['utilization_pct'].shift(2)

    # Calculate trend
    df['utilization_trend'] = df['utilization_pct'] - df['prev_utilization']

    # Classify trend
    df['trend_category'] = pd.cut(
        df['utilization_trend'],
        bins=[-float('inf'), -10, 10, float('inf')],
        labels=['Decreasing', 'Stable', 'Increasing']
    )

    return df

# ============================================================
# ALTERYX FORMULA TOOL → RECOMMENDATION ENGINE
# ============================================================

def calculate_credit_limit_increase(df, config):
    """
    Replaces: Alteryx Formula Tools for increase calculation
```

```
    """
    def get_increase_percentage(row):
        """Tiered increase based on credit score"""
        if row['credit_score'] >= 750:
            return config['tier1_increase_pct']
        elif row['credit_score'] >= 700:
            return config['tier2_increase_pct']
        elif row['credit_score'] >= 650:
            return config['tier3_increase_pct']
        else:
            return 0

    # Apply increase percentage
    df['increase_pct'] = df.apply(get_increase_percentage, axis=1)

    # Calculate recommended increase amount
    df['increase_amount'] = (df['credit_limit'] * df['increase_pct'] / 100).round(0)

    # Calculate new credit limit
    df['recommended_new_limit'] = df['credit_limit'] + df['increase_amount']

    # Apply maximum limit cap
    df['recommended_new_limit'] = df['recommended_new_limit'].clip(upper=config['max_credit_limit'])

    # Recalculate actual increase after cap
    df['final_increase_amount'] = df['recommended_new_limit'] - df['credit_limit']

    # Calculate post-increase utilization
    df['projected_utilization'] = (
        (df['current_balance'] / df['recommended_new_limit'] * 100).round(2)
    )

    return df

# ============================================================
# ALTERYX OUTPUT TOOLS → PANDAS TO_CSV/TO_EXCEL
# ============================================================

def generate_outputs(approved_df, rejected_df, output_path, cu_id, run_date):
    """
    Replaces: Multiple Alteryx Output Data Tools
    """
    # Approved list CSV
    approved_file = f"{output_path}/{cu_id}_approved_list_{run_date}.csv"
```

```
    approved_df.to_csv(approved_file, index=False)

    # Rejected list CSV
    rejected_file = f"{output_path}/{cu_id}_rejected_list_{run_date}.csv"
    rejected_df.to_csv(rejected_file, index=False)

    # Analysis report Excel (multiple sheets)
    analysis_file = f"{output_path}/{cu_id}_analysis_report_{run_date}.xlsx"
    with pd.ExcelWriter(analysis_file, engine='xlsxwriter') as writer:
        approved_df.to_excel(writer, sheet_name='Approved', index=False)
        rejected_df.to_excel(writer, sheet_name='Rejected', index=False)

        # Summary statistics
        summary_df = pd.DataFrame({
            'Metric': [
                'Total Accounts Analyzed',
                'Approved Accounts',
                'Rejected Accounts',
                'Approval Rate %',
                'Total Increase Amount',
                'Average Increase Amount',
                'Median Credit Score (Approved)'
            ],
            'Value': [
                len(approved_df) + len(rejected_df),
                len(approved_df),
                len(rejected_df),
                round(len(approved_df) / (len(approved_df) + len(rejected_df)) * 100, 2),
                approved_df['final_increase_amount'].sum(),
                approved_df['final_increase_amount'].mean(),
                approved_df['credit_score'].median()
            ]
        })
        summary_df.to_excel(writer, sheet_name='Summary', index=False)

    print(f"Generated outputs for {cu_id}")
    return approved_file, rejected_file, analysis_file

# ============================================================
# MAIN ORCHESTRATION
# ============================================================

def run_clip_analysis(cu_id, config, run_date):
    """
```

```
    Main function that orchestrates the entire CLIP process
    Replaces: Entire Alteryx workflow
    """
    print(f"Starting CLIP analysis for {cu_id}")

    # Extract data
    member_df = extract_member_data(config['tenant_id'], run_date)
    account_df = extract_account_data(config['tenant_id'], run_date)

    # Merge data
    merged_df = merge_member_account_data(member_df, account_df)

    # Calculate metrics
    metrics_df = calculate_account_metrics(merged_df, run_date)

    # Apply filters
    eligible_df, rejected_df = apply_eligibility_filters(metrics_df, config)

    # Calculate recommendations
    recommendations_df = calculate_credit_limit_increase(eligible_df, config)

    # Generate outputs
    output_path = f"outputs/{run_date[:7]}/{cu_id}"
    os.makedirs(output_path, exist_ok=True)

    generate_outputs(recommendations_df, rejected_df, output_path, cu_id, run_date)

    print(f"Completed CLIP analysis for {cu_id}")

    return recommendations_df, rejected_df
```

SQL Server Agent Job Setup

```
sql
```

```
-- ============================================================
-- SQL Server Agent Job: CLIP Automation
-- ============================================================

USE msdb;
GO

-- Create job
EXEC dbo.sp_add_job
    @job_name = N'CLIP_Monthly_Analysis',
    @enabled = 1,
    @description = N'Automated CLIP credit line increase analysis for all credit unions';

-- Add PowerShell step
EXEC dbo.sp_add_jobstep
    @job_name = N'CLIP_Monthly_Analysis',
    @step_name = N'Execute CLIP Orchestrator',
    @subsystem = N'PowerShell',
    @command = N'
        Set-Location "C:\CLIP_ALTERYX_AUTOMATION"
        .\scripts\orchestrator.ps1 -CreditUnion "ALL" -RunDate (Get-Date -Format "yyyy-MM-dd")
    ',
    @on_success_action = 1,  -- Quit with success
    @on_fail_action = 2;     -- Quit with failure

-- Schedule: Monthly on 1st at 6:00 AM
EXEC dbo.sp_add_schedule
    @schedule_name = N'Monthly_First_Day_6AM',
    @freq_type = 16,  -- Monthly
    @freq_interval = 1,  -- Day 1 of month
    @active_start_time = 60000;  -- 6:00 AM

EXEC dbo.sp_attach_schedule
    @job_name = N'CLIP_Monthly_Analysis',
    @schedule_name = N'Monthly_First_Day_6AM';

-- Assign to server
EXEC dbo.sp_add_jobserver
    @job_name = N'CLIP_Monthly_Analysis',
    @server_name = N'(local)';
GO
```

Comparison Matrix: Automation Approaches

Aspect	Approach 1: Alteryx Automation	Approach 2: Hybrid	Approach 3: Full Migration
Implementation Time	1-2 weeks	4-6 weeks	8-12 weeks
Technical Complexity	Low	Medium	High
Alteryx Dependency	Full	Partial	None
Licensing Cost	High (ongoing)	High (ongoing)	None (after migration)
Maintainability	Medium	Medium-High	High
Version Control	Difficult (XML)	Mixed	Excellent (Git)
Testing	Manual	Semi-automated	Fully automated
Performance	Medium	Medium-High	High
Scalability	Limited	Good	Excellent
Skills Required	Alteryx, PowerShell	Alteryx, Python	Python, SQL, dbt

---

Recommended Implementation Path

Phase 1: Quick Win (Weeks 1-2)

Approach: Alteryx Command-Line Automation

- Implement PowerShell orchestrator
- Create configuration management
- Add basic error handling
- Set up SQL Server Agent jobs

Phase 2: Enhancement (Weeks 3-6)

Approach: Add Python pre/post-processing

- Build workflow generator

- Implement advanced logging
- Create monitoring dashboard
- Add retry logic

Phase 3: Strategic Migration (Weeks 7-12)

Approach: Gradual migration to Python/dbt

- Start with data extraction layer
- Convert simple transformations
- Migrate complex business logic
- Validate outputs match Alteryx

Phase 4: Full Automation (Weeks 13+)

Approach: Complete Python/dbt solution

- Eliminate Alteryx dependency
- Implement full test suite
- Deploy to production
- Decommission Alteryx workflows

Success Metrics & Monitoring

Key Performance Indicators (KPIs)

```python
```

```python
# monitoring_metrics.py

def calculate_automation_metrics(execution_logs):
    """
    Track automation performance and ROI
    """
    metrics = {
        'execution_time': {
            'manual_average_minutes': 180,  # 3 hours manual per CU
            'automated_average_minutes': 15,
            'time_saved_per_run': 165,
            'monthly_runs': 10,  # 10 CUs
            'monthly_time_saved_hours': (165 * 10) / 60
        },

        'reliability': {
            'success_rate_pct': 95,
            'avg_retry_count': 0.2,
            'error_rate_pct': 5
        },

        'data_quality': {
            'records_processed': 50000,
            'validation_pass_rate_pct': 99.5,
            'data_completeness_pct': 99.8
        },

        'business_impact': {
            'accounts_approved_monthly': 13000,
            'total_credit_extended': 15000000,
            'analyst_hours_freed': 27.5
        }
    }

    return metrics
```

Next Steps & Action Items

1. **Immediate (Week 1)**
   - [ ] Inventory all existing Alteryx workflows

- [ ] Document current manual processes
- [ ] Identify CU-specific configurations
- [ ] Set up automation folder structure

2. **Short-term (Weeks 2-4)**
   - [ ] Implement PowerShell orchestrator
   - [ ] Create workflow generator script
   - [ ] Configure SQL Server Agent jobs
   - [ ] Test with 1-2 pilot credit unions
3. **Medium-term (Weeks 5-8)**
   - [ ] Roll out to all credit unions
   - [ ] Build monitoring dashboard
   - [ ] Create documentation
   - [ ] Train team on new process
4. **Long-term (Weeks 9-12)**
   - [ ] Begin Python/dbt migration
   - [ ] Implement comprehensive testing
   - [ ] Phase out Alteryx workflows
   - [ ] Optimize performance

Document Version: 1.0
Last Updated: December 2025
Author: Trellance Data Engineering Team