Case Study: Migrating Complex Alteryx Workflows to Qlik Sense

I recently led a project that presented some fascinating technical challenges. The high-level objective was to migrate several critical Alteryx workflows into Qlik Sense applications, driven by an enterprise-wide initiative to reduce software costs and bolster data security through centralized reporting.

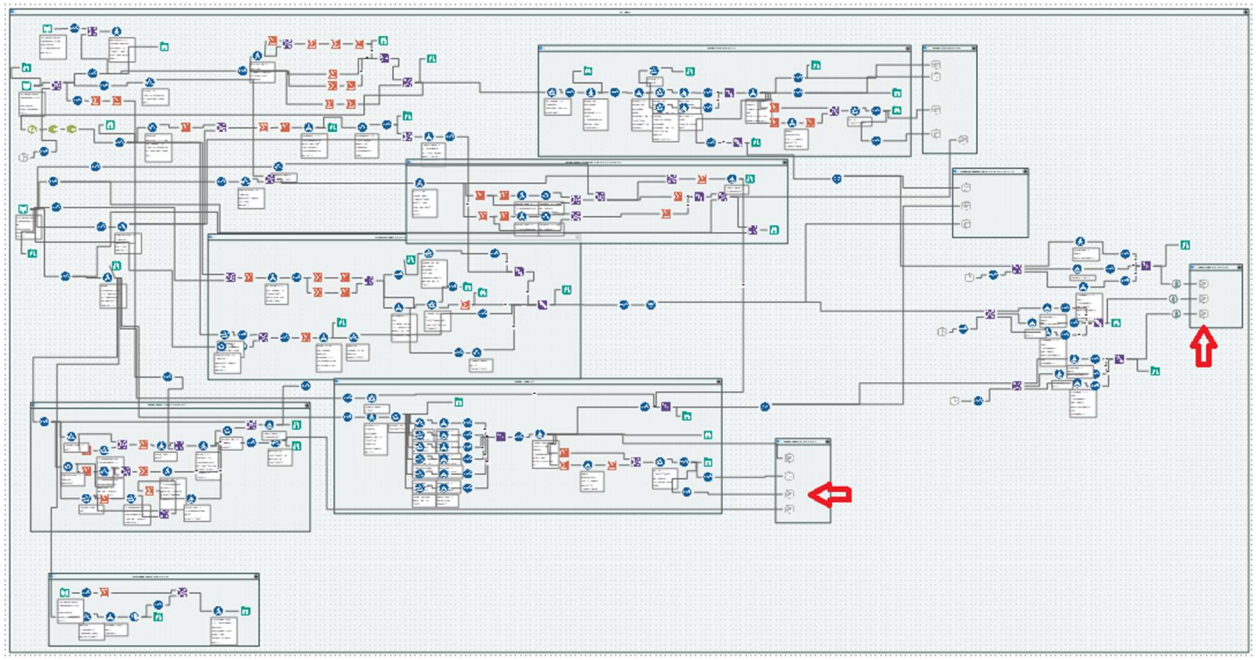
For the most part, these migrations were straightforward "lifts" because the workflows were uncomplicated. However, a handful of these legacy processes were far from simple.

The "Spaghetti" Challenge

This legacy workflow I inherited was a classic example of technical debt. Its layout was reminiscent of a "bowl of spaghetti," characterized by a dense web of crisscrossing connections. It handled a massive volume of data, pulling from various SQL queries in our data lake and several Google Sheets. Similarly, it pushed data out to multiple Google Sheets simultaneously.

To convert this into a clean Qlik Sense application, I first had to unravel the logic and separate the specific branches feeding each output. I achieved this by writing a series of Python scripts designed to manipulate the underlying XML of the Alteryx .yxmd files.

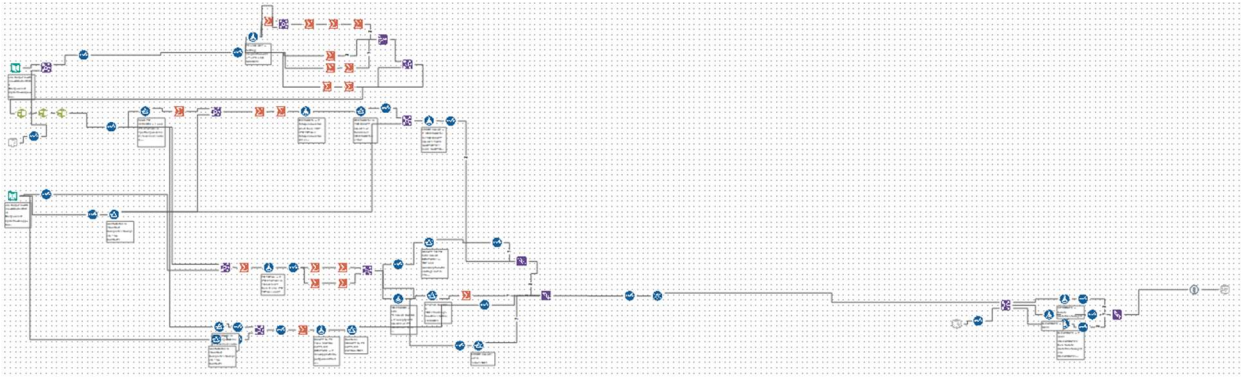
(In the image below, the arrows indicate specific nodes that I will use as examples throughout this document.)

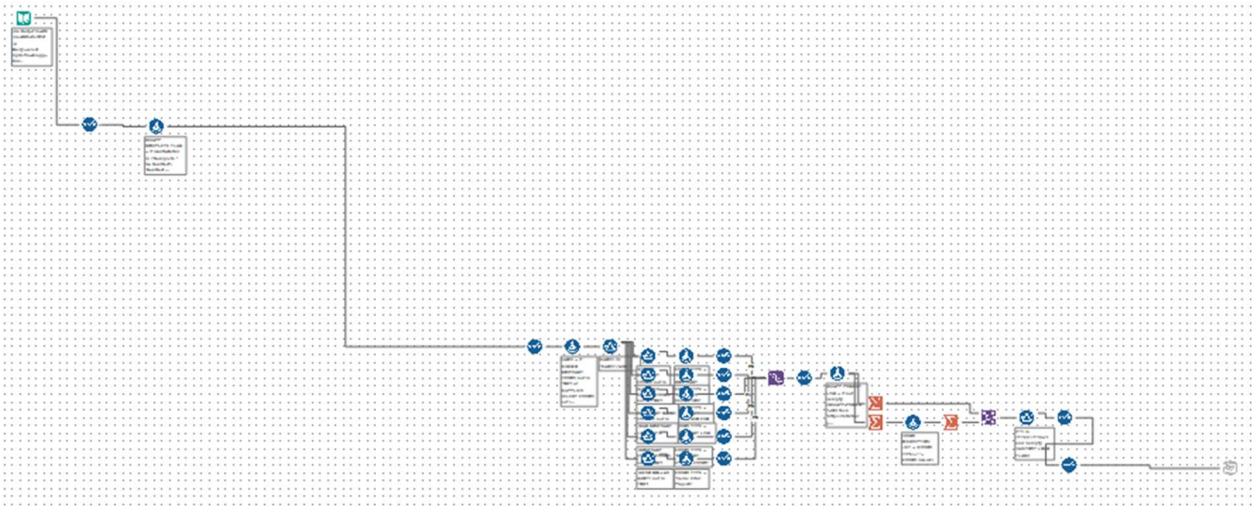


Phase 1: Untangling the Logic

My first step was to develop a Python script that could isolate the paths for each individual output into its own separate Alteryx workflow. By stripping away the secondary clutter, I could reverse-engineer the core business logic of each branch without the visual "noise" of the original file.

Below are two of the resulting "clean" branches:

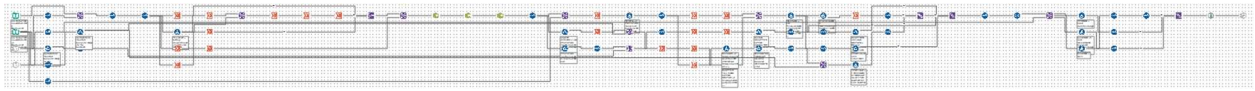


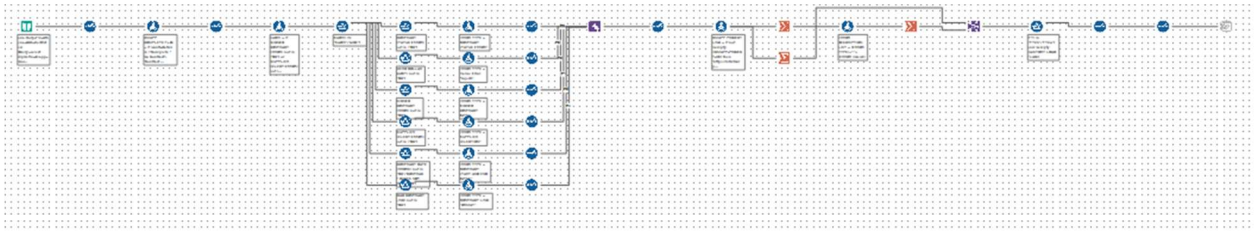


Phase 2: Visual Optimization

Once isolated, the workflows were still a bit disorganized. I updated my Python script to programmatically rearrange the tool coordinates in the XML, enforcing a strict left-to-right flow pattern. This made the logic immediately more digestible for stakeholders.

Here are those same two branches with their new, optimized layouts:





One critical decision I made was to give the Python script an option to retain the original **Tool ID** numbers rather than renumbering them. By keeping the original IDs, I ensured I could easily trace logic back to the master file if I needed to recombine branches within the final Qlik Sense load script.

Phase 3: Translating the Logic

The final phase involved translating Alteryx formulas and filters into Qlik Sense load script commands. To scale this, I created another Python utility that parsed the entire workflow, extracted the logic from Formula and Filter tools, and converted the syntax. The resulting translations were stored in a mapping spreadsheet for validation.

Example: Automated Logic Translation

The complexity of the original Alteryx formula required a robust SQL CASE structure in Qlik Sense.

Original Alteryx Formula:

```
if [contr_status] not in ("Contract Complete") then "CONTRACT DATES NOT REQUIRED"
 elseif[contr_status]="Contract Complete" and isempty([CONTRACT START DATE]) and isempty([CONTRACT END DATE]) then "ERR-CDT-03"
```

```
 elseif[contr_status]="Contract Complete" and isempty([CONTRACT END DATE]) and [m2m_convert]="yes" then "OK"
 elseif[contr_status]="Contract Complete" and isempty([CONTRACT END DATE]) and ([m2m_convert]="no" or
isempty([m2m_convert])) then "ERR-CDT-05"
 elseif[contr_status]="Contract Complete" and isempty([CONTRACT START DATE]) then "ERR-CDT-04"
 elseif[contr_status]="Contract Complete" and [CONTRACT START DATE] > [CONTRACT END DATE] then "ERR-CDT-02"
 elseif[contr_status]="Contract Complete" and [CONTRACT START DATE] < todate('2010-01-01') then "OK"
 elseif[contr_status]="Contract Complete" and [CONTRACT START DATE] > DateTimeAdd(DateTimeToday(),7,"months") then
"ERR-CDT-01"
 elseif[contr_status]="Contract Complete" and [CONTRACT END DATE] < todate('2010-01-01') then "ERR-CDT-06"
 elseif[contr_status]="Contract Complete" and [CONTRACT END DATE] > DateTimeAdd(DateTimeToday(), 5,"years") then "OK"
 elseif[contr_status]="Contract Complete" and [CONTRACT START DATE] < [CONTRACT END DATE] then "OK"
 elseif[contr_status]="Contract Complete" and [CONTRACT START DATE] = [CONTRACT END DATE] then "OK"
 else "ERROR CD" endif
```

Converted Qlik Sense (SQL) Formula:

```
CASE
        WHEN CONTR_STATUS NOT IN ( 'Contract Complete')
                THEN 'CONTRACT DATES NOT REQUIRED'
        WHEN CONTR_STATUS = 'Contract Complete'
                        AND CONTR_EFF_STRT_DT IS NULL
                        AND CONTR_EXPIR_DT IS NULL
                THEN 'ERR-CDT-03'
        WHEN CONTR_STATUS = 'Contract Complete'
                        AND CONTR_EXPIR_DT IS NULL
                        AND M2M_CONVERT = 'yes'
```

```
        THEN 'OK'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EXPIR_DT IS NULL
                AND (M2M_CONVERT = 'no' OR M2M_CONVERT IS NULL)
        THEN 'ERR-CDT-05'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EFF_STRT_DT IS NULL
        THEN 'ERR-CDT-04'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EFF_STRT_DT > CONTR_EXPIR_DT
        THEN 'ERR-CDT-02'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EFF_STRT_DT < DATE_PARSE ('2010-01-01', '%Y-%m-%d')
        THEN 'OK'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EFF_STRT_DT > DATE_ADD ('month', 7, CURRENT_DATE)
        THEN 'ERR-CDT-01'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EXPIR_DT < DATE_PARSE ('2010-01-01', '%Y-%m-%d')
        THEN 'ERR-CDT-06'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EXPIR_DT > DATE_ADD ('year', 5, CURRENT_DATE)
        THEN 'OK'
WHEN CONTR_STATUS = 'Contract Complete'
                AND CONTR_EFF_STRT_DT < CONTR_EXPIR_DT
        THEN 'OK'
WHEN CONTR_STATUS = 'Contract Complete'
```

```
                    AND CONTR_EFF_STRT_DT = CONTR_EXPIR_DT
                THEN 'OK'
            ELSE 'ERROR CD'
END AS CONTRACT_DATE_ERRORS
```

Technical Nuances

The most difficult aspect of these translations involved mapping functions that behave differently across platforms. For instance, Alteryx's CONTAINS function is highly flexible, whereas the Qlik Sense equivalent, MATCH, uses a completely different format and strictness levels. By utilizing Python to automate the bulk of the "heavy lifting," I was able to focus my time on solving these complex syntax edge cases, ultimately delivering a secure and highly performant Qlik Sense environment.