**Steps to Produce Inventory Allocation Maximization Data Sets**

1. Connected PowerBI to three dataflows as provided by aaron burton.
   1. **Inventory Dataflow:**
   2. Business Planning > Inventory – Latest Snapshot – Detailed > Inventory for Allocation
   3. **Orders Dataflow:**
   4. Business Planning > Open Orders with Appointments > Open Orders for Inventory Allocation
   5. **Product Master:**
   6. Business Planning > Item Master > Product Master for Allocation
2. Make each dataflow a table in PowerBI with all columns included
3. Exported xlsx excel sheets from PowerBI
4. Create and “age\_joiner” excel file where…
   1. First column is age and rows go from 1 to the max age in the model (currently 120)
   2. Second column is joiner or 1 for all rows
5. Generate SQL script to load the three excel files into SQLite
   1. Convert excel column names into a naming convention that I like
   2. Add a “joiner” column onto the inventory table. Each row is 1
   3. Create a age\_joiner table
6. Loaded the inventory, orders, and item excel files into sqlite
   1. Excel data type can be weird loading to sqlite, be careful.
7. Run the spec\_fix.py script which lives in the Python folder in the Inventory Allocation Maximization folder.
   1. This will remove the bad specs from the spec list in the inventory data
   2. It will also overwrite the existing mvp\_inventory table with a new version that includes the cleaned\_spec column
   3. This script is not capable, yet, of handling multiple items. Its an MVP version with static references. Some of the other required updates are commented in the code
8. Run the create\_products\_table.sql script.
   1. This will create a unique list of products and a formatted excel sheet ready to load directly into the coupa products table
      1. File cant be found @ S:\Supply\_Chain\Analytics\Inventory Allocation Maximization\MVP Data\input tables
   2. It will also create a new table called mvp\_distinct\_inventory\_products which will be useful in the next steps
9. Run the create\_BOM\_table.sql script. I have two scripts, at the moment (4/10/25) it’s the bottom one separated by a --------- on line 43.
   1. This will create a file ready to load directly into coupa’s BOMs table. Can be found @ S:\Supply\_Chain\Analytics\Inventory Allocation Maximization\MVP Data\input tables
10. Run the create\_BOM\_assignments\_table script.
    1. This makes some modifications to the BOM script basically
    2. I had to create a new table as a first step in this script to include the whs\_code the products live at. This will make sure product cant age at warehouses it doesn’t exist at.
       1. I will have to make a production policy for all the products and I only want things to be able to be “produced” aka aged at the warehouse they live at.
11. Run the create\_production\_policies\_table.sql
12. Run the create\_transportation\_policies\_table.sql
13. Run the create\_inventory\_policies\_table.sql
    1. There are a couple tricky things going on in here:
       1. I gather a distinct list of products and duplicate it through a joiner for every whs site in our mvp\_inventory dataset. The result is one product row for every whs.
       2. I use a row\_num function to apply a value that I multiple by 50 for a disposal cost that gets cheaper as the product ages.
       3. I join the actual inventory to that data to pull the total pallets value for the product that does actually exist in our inventory.
       4. This can go anywhere but I still need to deal with the total\_pallet count source number might now be correct. I need to work with Aaron to decide how to handle it. For example, there might be 50 lbs in a lot but the pallet count will be 0. Is that what we want?
14. Run the create\_sites\_policies\_table.sql
15. Run the create\_customers\_policies\_table.sql
16. Run the spec\_enumerate.py
    1. This script preps the mvp\_distinct\_inventory\_products table for creating the groups. This table is created as part of the create\_products\_table.sql script.
    2. The spec\_enumerate.py will take the list of specs and make a column for each spec.
    3. There is manual intervention here that I am not sure how to handle programmatically. The spec\_enumerate.py script will create as many columns as there are specs. Each day when I run this I wont know how many specs it creates. In the next step in the sql I have a join that needs to reference ALL the spec columns. I have to manually update the on statement with all the columns.
17. Run the create\_groups\_sql
    1. The idea is we use the demand to find out what groups we need to create and the mvp\_inventory to take the products we do have an fill them into the needed groups.
    2. I have to run that spec\_enumerate.py script before running the create\_groups.sql. I need to adjust the on statement for the specs on line 56. I need to include ALL the spec columns that exist in mvp\_distinct\_inventory\_products
18. Run the create\_customer\_demand\_table.sql script
    1. It uses stuff from the create groups .sql script and pulls from the orders data.

Something to think about for the future. Right now everything is absed around 1 item. All my joins don’t have to take into account item\_number. Make sure when I expand this I don’t forget about the item\_number join type of thing.