A data analysis dashboard for a cold drink supplier agency should be visually appealing, easy to navigate, and provide actionable insights. Here are key elements to include and tips to make it attractive:

1. Key Performance Indicators (KPIs):

- \*\*Sales Performance:\*\* Total sales revenue, sales by product category, sales trends over time.

- \*\*Inventory Management:\*\* Current stock levels, reorder points, stockouts, and overstock situations.

- \*\*Customer Insights:\*\* Customer segments, customer preferences, repeat purchases.

- \*\*Financial Metrics:\*\* Monthly profit margins, cost of goods sold (COGS), gross profit.

- \*\*Operational Efficiency:\*\* Order fulfillment rates, order processing time, supplier performance.

2. Visualization Techniques:

- Charts and Graphs: Use bar charts, line graphs, and pie charts to visualize sales trends, inventory levels, and customer insights.

- Heatmaps: Show areas of high and low sales performance or inventory turnover.

- Geographical Data: If applicable, display sales data geographically to identify regional trends.

- KPI Widgets: Include widgets displaying key metrics like total revenue, top-selling products, and customer segments.

3. Interactive Features:

- \*\*Drill-Down Functionality:\*\* Allow users to drill down into specific data points for detailed analysis.

- \*\*Filters and Filters:\*\* Include filters for time periods, product categories, customer segments, etc., to customize data views.

- \*\*Hover-over Information:\*\* Provide additional information or tooltips when users hover over data points.

4. Dashboard Design:

- \*\*Color Scheme:\*\* Choose a visually appealing color scheme that aligns with the brand and makes data easy to interpret.

- \*\*Layout:\*\* Organize data elements logically with clear labels and headings for each section.

- \*\*White Space:\*\* Use white space effectively to avoid clutter and improve readability.

- \*\*Responsive Design:\*\* Ensure the dashboard is responsive and optimized for different devices (desktop, tablet, mobile).

5. Actionable Insights:

- Include summary sections or text boxes highlighting key insights and recommendations based on the data.

- Use alerts or notifications for important events such as stockouts or significant sales fluctuations.

6. User-Friendly Navigation:

- Include a navigation menu or tabs for easy access to different sections of the dashboard.

- Provide a search function to quickly find specific data or KPIs.

By incorporating these elements into the dashboard design, you can create an attractive and effective data analysis tool that provides valuable insights to the cold drink supplier agency.

For inventory management in the context of a cold drinks supplier vendor app, you would typically include the following columns:

1. **Item Name:** This column stores the name or description of the cold drink item, such as "Coca-Cola 500ml" or "Sprite 1L."
2. **Quantity in Stock:** Here, you track the current quantity of each item available in your inventory.
3. **Reorder Level:** This column specifies the minimum quantity at which you should reorder the item to avoid stockouts.
4. **Supplier Details:** Include columns for supplier name, contact information, and possibly pricing details if they vary by supplier.
5. **Barcode/QR Code:** If you're implementing barcode scanning or QR code functionality, you'd have a column to store the unique codes for each item.
6. **Purchase Price:** This column records the price at which you purchase each item from the supplier.
7. **Selling Price:** Store the price at which you sell each item to customers, which may vary based on factors like size or promotions.
8. **Profit Margin**: Calculate and display the profit margin for each item, typically calculated as the selling price minus the purchase price, divided by the selling price.
9. **Category/Type:** Categorize items based on types such as carbonated drinks, juices, water, etc., for better organization and reporting.
10. **Expiry Date:** If applicable, include a column to track the expiry date of perishable items to manage stock rotation effectively.
11. **Location/Storage Area:** Specify where each item is stored within your warehouse or storage facility for efficient retrieval.
12. **Notes/Comments:** This column can be used for additional information, such as special handling instructions or notes about specific items.

These columns provide a comprehensive view of your cold drinks inventory, allowing you to track stock levels, manage suppliers, calculate profits, and optimize inventory replenishment processes.

**CODES:**

* **TO GENERATE UNIQUE PRODUCT ID**
* CONCATENATE("PRD", RIGHT("000" & (COUNT(PUCHASES[PRD\_ID]) + 1), 3))
* CONCATENATE("CL", RIGHT("000" & (COUNT(CLIENTS[CL\_ID]) + 1), 3))
* CONCATENATE("BILL NO:", RIGHT("000" & (COUNT(SALES[INVOICE NO.]) + 1), 3))
* CONCATENATE(RIGHT("000" & (COUNT(SALES FORM[INVOICE NO.]) + 1), 3))
* CONCATENATE("INV-", LEFT(UNIQUEID(), 6))
* **TO FETCH DATA FROM ANOTHER TABLE**
* LOOKUP([\_THISROW].[PRD\_ID],"INVENTORY","PRD\_ID","PRODUCT NAME")
* LOOKUP([\_THISROW].[PRD\_ID],"INVENTORY","PRD\_ID","IMAGES")
* **TO GET DATA/ANY DETAILS FROM ANOTHER TABLE BASED ON CONDITION**
* any(select(INVENTORY[PRODUCT NAME],[\_THISROW].[PRD\_ID] = [PRD\_ID]))
* any(select(CLIENTS[CLIENT NAME],[\_THISROW].[CL\_ID] = [CL\_ID]))
* any(select(INVENTORY[SELLING PRICE],[\_THISROW].[PRD\_ID] = [PRD\_ID]))
* any(select(PURCHASES[QTY. IN STOCK],[\_THISROW].[PRD\_ID] = [PRD\_ID]))
* any(select(INVENTORY[PRD\_ID],[\_THISROW].[PRODUCT NAME] = [PRODUCT NAME]))
* **TO DO SUM CALCULATIONS**
* SUM(SELECT( INVENTORY[QTY. IN STOCK], AND( [PRD\_ID] = [\_THISROW].[PRD\_ID], [PRODUCT NAME] = [\_THISROW].[PRODUCT NAME])))
* SUM(SELECT( SALES[TOTAL\_AMT.],[INVOICE NO.] = [\_THISROW].[INVOICE NO.]))
* IF([\_THISROW].[PAYMENT STATUS] = "PARTIAL",SUM(SELECT(SALES[TOTAL\_AMT.], [INVOICE NO.] = [\_THISROW].[INVOICE NO.]))-[AMOUNT PAID],SUM(SELECT(SALES[TOTAL\_AMT.], [INVOICE NO.] = [\_THISROW].[INVOICE NO.])))
* IF([\_THISROW].[PAYMENT STATUS] = "PAID", SUM(SELECT(SALES[TOTAL\_AMT.],[INVOICE NO.] = [\_THISROW].[INVOICE NO.])),0)

**FOR SALES TABLE:**

* SWITCH([PAYMENT STATUS],

"PARTIAL", SUM(SELECT(SALES[TOTAL\_AMT.], [INVOICE NO.] = [\_THISROW].[INVOICE NO.]))-[AMOUNT PAID],

"PAID", "0.00",

"UNPAID", SUM(SELECT(SALES[TOTAL\_AMT.], [INVOICE NO.] = [\_THISROW].[INVOICE NO.])), "0.00")

**FOR PURCHASE TABLE:**

SWITCH([PAYMENT STATUS],

"PARTIAL", SUM(SELECT(PURCHASES[TOTAL PAID AMT.], [INVOICE NO.] = [\_THISROW].[INVOICE NO.]))-[AMOUNT PAID],

"PAID", "0.00",

"UNPAID", [Total Purchased Amt.],"")

* **USED IN ACTION TO SUBTRACT STOCK**
* [QTY. IN STOCK] - SUM(SELECT(Sales[QTY\_SOLD], AND([PRD\_ID] = [\_THISROW].[PRD\_ID], [INVOICE DATE] = MAX(SELECT(Sales[INVOICE DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))
* **VALIDATIONS**
* ISBLANK(select(INVENTORY[PRD\_ID],([PRD\_ID]=[\_THISROW].[PRD\_ID])))
* AND(ISNOTBLANK([Phone]),LEN([Phone]) = 10)
* SELLING PRICE VALIDATION:

*AND([SELLING PRICE] > [PURCHASED PRICE], DECIMAL([SELLING PRICE]) = [SELLING PRICE])*

*MESSAGE: "SELLING PRICE MUST BE HIGHER THAN PURCHASE PRICE"*

* TOTAL PAID AMT. VALIDATION:

*[TOTAL PAID AMT.] = ([PURCHASED PRICE] \* [UPDATED STOCK])*

*MESSAGE: "IT MUST BE QTY. PURCHASED \* PURCHASED PRICE"*

* TOTAL AMT. VALIDATION OF SALES TABLE:

*[TOTAL\_AMT.] = (([QTY\_SOLD] \* [UNIT\_PRICE])-[DISCOUNT])*

*MESSAGE: "IT MUST BE QTY. SOLD \* UNIT PRICE"*

* PURCHASED PRICE VALIDATION OF PURCHASED TABLE:

*[PURCHASED PRICE]<[M.R.P.]*

*MESSAGE: "PURCHASE PRICE MUST BE LESS THAN M.R.P."*

* TO AVOID DUPLICATE USERNAME IN USERS TABLE:

*NOT(IN([USERNAME], USERS[USERNAME]))*

* **TO SELECT ITEMS FROM ITEM TABLE BY USING THIS FORMULA IN VALID IF CONSTRAINT IN SALES TABLE’S PRODUCT NAME**

**PRODUCTS:**

* SELECT(PURCHASES[PRODUCT NAME],TRUE)
* any(select(INVENTORY[PRD\_ID],[\_THISROW].[PRODUCT NAME] = [PRODUCT NAME]))

**CLIENTS:**

* SELECT(CLIENTS[CLIENT NAME],TRUE)
* any(select(CLIENTS[CLIENT NAME],[\_THISROW].[CLIENT NAME] = [CLIENT NAME]))
* **TO CHANGE THE NAME OF BUTTONS**
* IF(CONTEXT("View") = "TABLE\_NAME", "BUTTON NAME", IF(CONTEXT("View") = "TABLE \_NAME", "BUTTON NAME", "SAVE"))
* IF(CONTEXT("View") = "NEW SALES", "NEXT","SAVE")
* **TO CHANGE THE NAME OF COLUMNS ON CERTAIN CONDITIONS**
* IF([PAYMENT STATUS]="PARTIAL","REMAINING AMOUNT","TOTAL AMOUNT")
* [PAYMENT STATUS]="PARTIAL"]
* **TO CHANGE THE BUTTON NAME ON CERTAIN CONDITIONS**
* IF(CONTEXT("View") = "NEW SALES", "NEXT","SAVE")
* IF(OR(CONTEXT("View") = "NEW SALES",CONTEXT("View") = "ALLOCATION FORM\_Form",CONTEXT("View") = "UPDATE STOCK / PURCHASES"),"NEXT","SAVE")
* IF(CONTEXT("View") = "Settings", "LOGIN",IF(OR(CONTEXT("View") = "NEW SALES",CONTEXT("View") = "ALLOCATION FORM\_Form",CONTEXT("View") = "UPDATE STOCK / PURCHASES"),"NEXT","SAVE"))

**CONCATENATE([CLIENT NAME], "-", [INVOICE NO.], "-", TEXT([DATE], "YYYYMMDD"), "-", TEXT([DATE], "HHMMSS"), ".pdf")**

**Description:**

This formula is designed to generate a unique and descriptive file name for PDF invoices, ensuring easy identification and organization of files. By combining the client name, invoice number, date, and time, each file name is distinct and meaningful, preventing any file naming conflicts and making it straightforward to locate specific invoices.

**Usage:**

1. **Unique Identification**: Each file name created with this formula is unique, even if multiple invoices are generated for the same client on the same day. This prevents overwriting and confusion.
2. **Organized Storage**: The structure of the file name makes it easy to organize and search for files in Google Drive or any other storage system.
3. **Instant Recognition**: The inclusion of the client name and invoice number allows for quick recognition of the file’s contents without needing to open it.
4. **Date and Time Stamping**: Adding the date and time provides an exact record of when the invoice was created or updated, which is useful for tracking and auditing purposes.

**CONCATENATE("/Clients Apps/COLD\_DRINK\_MANAGEMENT/INVOICES/", [CLIENT NAME], "-", [INVOICE NO.], "-", TEXT([DATE], "YYYYMMDD"), "-", TEXT([DATE], "HHMMSS"), ".pdf")**

**Description:**

This formula is designed to generate a file path for PDF invoices within a specific directory structure, facilitating organized storage and easy retrieval of documents. By combining a predefined path with dynamic elements such as client name, invoice number, date, and time, each file path is unique and structured, allowing for efficient document management.

**Components:**

* **"/Clients Apps/COLD\_DRINK\_MANAGEMENT/INVOICES/"**: Predefined path indicating the directory where the invoices are stored, ensuring consistent storage location.
* **[CLIENT NAME]**: The name of the client, providing context about the recipient of the invoice and helping in categorization.
* **"-"**: Hyphens used as separators to enhance readability and structure.
* **[INVOICE NO.]**: The unique invoice number, aiding in differentiating invoices and avoiding file name conflicts.
* **TEXT([DATE], "YYYYMMDD")**: Converts the date to a YYYYMMDD format for a standardized and sortable date representation.
* **TEXT([DATE], "HHMMSS")**: Converts the time to a HHMMSS format (24-hour clock), adding an exact timestamp to the file path.
* **".pdf"**: Appends the .pdf file extension, indicating the file type.

**Usage:**

1. **Structured Storage**: The formula ensures that all PDF invoices are stored in a designated directory, promoting a structured and organized storage system.
2. **Unique Identification**: Each file path generated is unique, incorporating dynamic elements like client name, invoice number, date, and time for precise identification.
3. **Efficient Retrieval**: With a predefined path and descriptive file names, retrieving specific invoices becomes faster and more straightforward.
4. **Consistent Directory Structure**: The predefined path maintains a consistent directory structure, making it easy for users to navigate and manage documents.

**[\_THISROW\_BEFORE].[PAYMENT STATUS] <> [\_THISROW\_AFTER].[PAYMENT STATUS]**

This formula is used in the Reset option within AppSheet's Update Behavior settings. Its purpose is to trigger an action when there is a change in the "PAYMENT STATUS" column of a row. Specifically, it is commonly employed to empty the "PRINT INVOICE" column when the "PAYMENT STATUS" changes.

**How It Works:**

* [\_THISROW\_BEFORE].[PAYMENT STATUS]: Refers to the value of the "PAYMENT STATUS" column before any updates are made to the row.
* [\_THISROW\_AFTER].[PAYMENT STATUS]: Refers to the value of the "PAYMENT STATUS" column after updates are applied to the row.

**Use Case:**

* Suppose you have a table where you track sales transactions, and there's a column named "PRINT INVOICE" that indicates whether an invoice should be printed for a particular transaction.
* You want the "PRINT INVOICE" column to be automatically emptied whenever the payment status of a transaction changes. This could be useful, for example, when a payment is marked as "Paid" or "Cancelled," indicating that an invoice may need to be re-evaluated.

**Implementation:**

* In the AppSheet editor, you set up the "PRINT INVOICE" column with the Reset option enabled.
* You enter the formula[\_THISROW\_BEFORE].[PAYMENTSTATUS]<>[\_THISROW\_AFTER].[PAYMENT STATUS] as the reset condition.
* When a row is updated and the "PAYMENT STATUS" changes, the formula triggers the Reset option, causing the "PRINT INVOICE" column to be emptied.

**Benefits:**

* Ensures data accuracy and consistency by clearing outdated or irrelevant information automatically.
* Streamlines workflows by automating tasks based on specific column changes.
* Improves user experience by maintaining clean and up-to-date records.

This formula, combined with the Reset option in AppSheet, provides a powerful way to manage data and automate actions based on dynamic changes in column values.

**IF([\_THISROW\_BEFORE].[PAYMENT STATUS] <> [\_THISROW\_AFTER].[PAYMENT STATUS], FALSE, [\_THISROW].[PRINT INVOICE])**

This formula is designed to control the value of the "PRINT INVOICE" column based on changes in the "PAYMENT STATUS" column within an AppSheet app. It is typically used in the App Formula or Initial Value setting of the "PRINT INVOICE" column to determine whether the column should be set to FALSE (empty) or retain its current value.

**How It Works:**

* **[\_THISROW\_BEFORE].[PAYMENT STATUS]**: Represents the value of the "PAYMENT STATUS" column before any updates are made to the row.
* **[\_THISROW\_AFTER].[PAYMENT STATUS]**: Represents the value of the "PAYMENT STATUS" column after updates are applied to the row.
* **[\_THISROW].[PRINT INVOICE]**: Refers to the current value of the "PRINT INVOICE" column.

**Use Case:**

* Let's say you have a table for tracking sales transactions, and there's a column named "PRINT INVOICE" that determines whether an invoice should be printed for a particular transaction.
* You want to automate the process of clearing the "PRINT INVOICE" column whenever the payment status of a transaction changes. This helps ensure that the correct action is taken based on the payment status update.

**Implementation:**

* In the AppSheet editor, you set up the "PRINT INVOICE" column with the App Formula or Initial Value option.
* You enter the formula

IF([\_THISROW\_BEFORE].[PAYMENT STATUS] <> [\_THISROW\_AFTER].[PAYMENT STATUS], FALSE, [\_THISROW].[PRINT INVOICE]) in the formula field.

* When a row is updated and the "PAYMENT STATUS" changes, the formula evaluates whether to set "PRINT INVOICE" to FALSE (empty) or retain its current value based on the condition.

**Benefits:**

* Offers dynamic control over column values based on changes in other columns.
* Simplifies data management by automating actions in response to specific column updates.
* Improves data accuracy and consistency by ensuring that columns reflect the most relevant information.

By using this formula, you can enhance the functionality of your AppSheet app by intelligently managing column values in response to changing data conditions.

**OR( [\_THISROW\_BEFORE].[PAYMENT STATUS] <> [\_THISROW\_AFTER].[PAYMENT STATUS], [\_THISROW\_BEFORE].[AMOUNT PAID] <> [\_THISROW\_AFTER].[AMOUNT PAID])**

Purpose: This formula is designed to be used in the Reset on Edit setting for a column in an AppSheet table. Specifically, it is used to determine whether the "PRINT INVOICE" column should be reset (emptied) based on changes in either the "PAYMENT STATUS" or "AMOUNT PAID" columns.

**Explanation:**

* [\_THISROW\_BEFORE]: Represents the values of the row before the update.
* [\_THISROW\_AFTER]: Represents the values of the row after the update.
* OR: Logical function that returns TRUE if any of the conditions inside it are TRUE.

**How it Works:**

* The formula checks if the "PAYMENT STATUS" value before the update is different from the value after the update.
* It also checks if the "AMOUNT PAID" value before the update is different from the value after the update.
* If either of these conditions is true, the formula returns TRUE, triggering the reset action for the "PRINT INVOICE" column.
* If both conditions are false, the formula returns FALSE, and the "PRINT INVOICE" column retains its value.

**Use Case:**

* This formula is particularly useful in scenarios where you want to ensure that certain columns are cleared or reset when specific related data changes.
* For instance, in a sales tracking application, you might want the "PRINT INVOICE" column to be reset whenever there is a change in the payment status or the amount paid, indicating that the invoice might need to be reprinted with updated information.

**Implementation:**

* Navigate to the column settings for "PRINT INVOICE" in the AppSheet editor.
* Find the "Reset on edit" option.
* Enter the formula:  
  OR([\_THISROW\_BEFORE].[PAYMENT STATUS] <> [\_THISROW\_AFTER].[PAYMENT STATUS],[\_THISROW\_BEFORE].[AMOUNT PAID]<>[\_THISROW\_AFTER].[AMOUNT PAID])
* Save the changes.

**Benefits:**

* Ensures data consistency by automatically clearing the "PRINT INVOICE" column when relevant data changes.
* Reduces manual intervention by automating the reset process.
* Enhances the accuracy of printed invoices by ensuring they reflect the most current payment status and amount paid.

By using this formula, you can maintain accurate and up-to-date records in your AppSheet application, improving overall data integrity and user experience.

**OR([PAYMENT STATUS]="UNPAID",[PAYMENT STATUS]="PARTIAL",ISBLANK([PAYMENT STATUS]), AND([PAYMENT STATUS]="PAID",DATE([PAYMENT DATE])=TODAY()))**

#### **Purpose:**

The formula HOUR(NOW() - [Date]) <= 24 is designed to manage the visibility of the delete action button in the Purchase table of an inventory management application. Its primary purpose is to restrict users from deleting purchase records that are older than 24 hours, ensuring that only recent entries can be modified or removed within a specific time frame.

#### **Formula Breakdown:**

* **NOW()**: This function retrieves the current date and time, representing the exact moment when the action is evaluated.
* **[Date]**: This is the column in the Purchase table that stores the date and time when each purchase was added.
* **NOW() - [Date]**: This expression calculates the time difference between the current moment and the recorded purchase time.
* **HOUR()**: This function converts the time difference into hours.
* **<= 24**: This condition ensures that the calculated time difference is 24 hours or less.

#### **Use Case:**

* **Visibility Control**: The formula is applied in the "Only if this condition is true" field of the delete action button in the Purchase table. By doing so, it controls the button's visibility based on the age of the purchase record.
* **Time-Based Restriction**: It provides a mechanism to restrict deletion capabilities to within 24 hours of adding a purchase. This helps maintain data integrity by preventing the removal of older records that might be critical for inventory tracking and auditing purposes.

#### **Implementation Steps:**

1. **Navigate to the Behavior section** in the AppSheet editor.
2. **Select the delete action** associated with the Purchase table.
3. **Enter the formula** in the "Only if this condition is true" field:  
   HOUR(NOW() - [Date]) <= 24

#### **Benefits:**

* **Enhanced Data Security**: By restricting deletions to recent entries, this formula helps in safeguarding older records from accidental or unauthorized deletion.
* **Operational Efficiency**: It ensures that only recent and possibly erroneous entries can be deleted or corrected, promoting accurate record-keeping.
* **Time-Sensitive Actions**: Useful for businesses needing to maintain a strict time-bound control over their inventory data modifications.

This formula effectively balances flexibility in managing recent purchases while ensuring the stability and integrity of historical data within the inventory management system.

**Purpose**

This configuration aims to streamline the process of updating purchase details for items bought from suppliers. It ensures that while adding new purchases, the previously saved values for columns such as `UNIT\_COST`, `MRP`, `RATE`, `GST\_PERCENTAGE`, `GST\_AMOUNT`, and `PURCHASED\_PRICE` are displayed and can be updated in the same view. This setup simplifies data entry and maintains consistency in the purchase records.

**How It Works**

The configuration involves using virtual columns to fetch the latest saved values for the specified columns. These values are then used to set the initial values in the corresponding real columns when a new purchase entry is made. This allows users to see the current values and update them if necessary during the entry process.

**Implementation Steps**

**1. Add Virtual Columns:**

Virtual columns are created to fetch the latest saved values from the `PURCHASES` table based on the `PRD\_ID` (Product ID).

\*\*Previous GST Percentage:\*\*

ANY(SELECT(PURCHASES[GST%], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

\*\*Previous MRP:\*\*

ANY(SELECT(PURCHASES[M.R.P.], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

\*\*Previous Rate:\*\*

ANY(SELECT(PURCHASES[RATE], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

\*\*Previous GST AMT.\*\*

ANY(SELECT(PURCHASES[GST\_AMT.], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

\*\*Previous PURCHASED PRICE:\*\*

ANY(SELECT(PURCHASES[PURCHASED PRICE], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

\*\*Previous SELLING PRICE:\*\*

ANY(SELECT(PURCHASES[SELLING PRICE], AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[DATE] = MAX(SELECT(PURCHASES[DATE], [PRD\_ID] = [\_THISROW].[PRD\_ID])))))

**2. Set Initial Values:**

The initial values for the real columns are set using the values from the virtual columns. This ensures that the form displays the latest saved values when a new purchase entry is made.

\*\*GST\_PERCENTAGE:\*\*

IF(ISBLANK([\_THISROW].[GST%]), [SAVED GST %], [\_THISROW].[GST%])

\*\*MRP:\*\*

IF(ISBLANK([\_THISROW].[MRP]), [Previous MRP], [\_THISROW].[MRP])

\*\*RATE:\*\*

IF(ISBLANK([\_THISROW].[RATE]), [SAVED RATE], [\_THISROW].[RATE])

\*\*SELLING PRICE:\*\*

IF(ISBLANK([\_THISROW].[SELLING PRICE]), [SAVED SELLING PRICE], [\_THISROW].[SELLING PRICE])

**3. Ensure Columns are Editable:**

Ensure all these columns are editable so that users can change the values if needed during the purchase entry process.

**4. Configure Form View:**

Go to UX > Views and ensure the form view for the `PURCHASES` table includes all necessary columns and is configured to display the initial values. This will make it easier for users to update purchase details accurately.

**Benefits**

* **Streamlined Data Entry:**

Users can see and update previously saved values directly in the form, reducing the likelihood of errors.

* **Consistency:**

Maintains consistency in the purchase records by ensuring that updates are made based on the latest saved values.

* **Efficiency:**

Speeds up the data entry process by pre-filling the form with the most recent values, allowing for quick adjustments.

By implementing this configuration, the purchase entry process becomes more efficient, accurate, and user-friendly.

**To prevent allocating duplicate items to the same salesperson in your "STOCK ALLOCATION" table, you can set up a validation rule using a Data Validity Condition in AppSheet. Here's how to do it:**

### **Step-by-Step Guide**

1. **Go to the Data tab** in your AppSheet editor and select the "STOCK ALLOCATION" table.
2. **Open the column structure** and find the PRD\_ID column.
3. **Set up a Data Validity Condition** for the PRD\_ID column to ensure that no duplicate items are allocated to the same salesperson.

### **Validation Formula**

The formula will check if the combination of USER\_ID and PRD\_ID already exists in the table. If it does, it will invalidate the entry.

1. **Select the PRD\_ID column** in the "STOCK ALLOCATION" table.
2. **In the Data Validity Condition** (or Valid If) section, enter the following formula:

NOT(IN([PRD\_ID],SELECT(STOCK ALLOCATION[PRD\_ID],AND([USER\_ID] = [\_THISROW].[USER\_ID],[ALT\_ID] <> [\_THISROW].[ALT\_ID] ))))

### **Explanation of the Formula**

* SELECT(STOCK ALLOCATION[PRD\_ID], AND([USER\_ID] = [\_THISROW].[USER\_ID], [ALT\_ID] <> [\_THISROW].[ALT\_ID])):
  + This part selects all PRD\_ID values from the "STOCK ALLOCATION" table where the USER\_ID matches the current row's USER\_ID and the ALT\_ID is not the same as the current row's ALT\_ID. This ensures that it does not count the current row in the check.
* IN([PRD\_ID], SELECT(...)):
  + Checks if the current PRD\_ID is in the list of PRD\_ID values returned by the SELECT statement.
* NOT(...):
  + Ensures that the PRD\_ID is not already in the list, meaning it validates only if there are no duplicate items for the same salesperson.

### 

### 

### **Applying the Validation**

* **Save the changes** after entering the formula.
* **Test the validation** by trying to allocate the same item to the same salesperson. The app should now prevent duplicate entries.

### **Example**

Consider the following entries in the "STOCK ALLOCATION" table:

| **ALT\_ID** | **ALLOCATION\_ID** | **USER\_ID** | **USERNAME** | **ALLOCATION**  **DATE** | **PRD\_ID** | **PRODUCT**  **NAME** | **M.R.P.** | **UNIT PRICE** | **QTY.**  **ALLOCATED** | **TOTAL AMT** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4cd4b1b5 | ALL001 | USER001 | Guddu | 6/23/2024 19:51:11 | PRD002 | PEPSI 1000 ML | 50 | 45 | 50 | 2250 |
| d0795965 | ALL002 | USER004 | Golu | 6/23/2024 21:17:59 | PRD005 | MOUNTAIN DEW 1000 ML | 32 | 32 | 90 | 2880 |

* If you try to add another entry with USER\_ID as USER001 and PRD\_ID as PRD002, the validation rule will prevent it.

This setup ensures that each salesperson can only have one unique PRD\_ID in their allocation, preventing duplicate item allocations. If you have further questions or need more specific guidance, feel free to ask!

### **About Section: Locking Initial Values in Stock Allocation Table**

#### **Overview**

In an inventory management application, it is essential to maintain the integrity of initial allocation data for stock management. Specifically, the ALLOTED STOCK and STOCK VALUE columns in the "STOCK ALLOCATION" table should retain their initial values based on the QTY. ALLOCATED and TOTAL AMT. columns, respectively. This section explains the method to ensure these columns do not get updated after their initial values are set.

#### **Solution Explanation**

To achieve this, we set initial values for the ALLOTED STOCK and STOCK VALUE columns and prevent them from being updated after the initial assignment. This is done through a combination of initial value formulas and an action triggered by a workflow to lock the columns.

#### **Steps to Implement**

* **Set Initial Values for Columns:**
* **ALLOTED STOCK:**
  + - Navigate to Data -> Columns.
    - Select the ALLOTED STOCK column.
    - Set the **Initial Value** formula to: [QTY. ALLOCATED]
* **STOCK VALUE:**
  + - Navigate to Data -> Columns.
    - Select the STOCK VALUE column.
    - Set the **Initial Value** formula to: [TOTAL AMT.]
* **Create an Action to Lock Columns:**
* Navigate to Behavior -> Actions.
* Click "New Action".
* Set the following properties:
  + - **For a record of this table:** STOCK ALLOCATION
    - **Do this:** Data: set the values of some columns in this row
    - **Set these columns:**

ALLOTED STOCK:  
IF(ISBLANK([ALLOTED STOCK]), [QTY. ALLOCATED], [ALLOTED STOCK])

STOCK VALUE:  
  
IF(ISBLANK([STOCK VALUE]), [TOTAL AMT.], [STOCK VALUE])

* **Create a Data Change Workflow:**
* Navigate to Behavior -> Workflows.
* Click "New Workflow Rule".
* Set the following properties:
  + - **Rule Name:** Lock Columns
    - **Table Name:** STOCK ALLOCATION
    - **When:** Updates Only
    - **If this condition is true:**AND(ISNOTBLANK([QTY. ALLOCATED]), ISNOTBLANK([TOTAL AMT.]))
    - **Action:** Data: execute an action on a set of rows
    - **Referenced Action:** Select the action created to lock columns.

#### **Example**

Consider an entry in the "STOCK ALLOCATION" table:

| **ALT\_ID** | **ALLOCATION\_ID** | **USER\_ID** | **USERNAME** | **ALLOCATION**  **DATE** | **PRD\_ID** | **PRODUCT NAME** | **M.R.P.** | **UNIT PRICE** | **QTY. ALLOCATED** | **TOTAL AMT.** | **ALLOTED STOCK** | **STOCK**  **VALUE** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | ALL001 | USER001 | John | 2024-06-24 | PRD002 | PEPSI 1000 ML | 50 | 45 | 50 | 2250 | 50 | 2250 |

After implementing the steps above:

1. The initial values for ALLOTED STOCK and STOCK VALUE are set to 50 and 2250, respectively.
2. If the QTY. ALLOCATED or TOTAL AMT. columns are updated later, the values in ALLOTED STOCK and STOCK VALUE remain unchanged.

This ensures the integrity of the initial stock allocation data, preventing unintentional changes and maintaining accurate records for auditing and reporting purposes.

### **About the Delete Action Condition**

In our inventory management application, we have implemented a validation to ensure that records can only be deleted under specific conditions. This validation helps maintain the integrity of our data and ensures that only authorized users can perform deletion actions within a certain timeframe. The conditions applied are as follows:

#### **Conditions:**

1. **Time Constraint**:
   * Records can only be deleted if they were created within the last 12 hours. This constraint helps prevent the deletion of older records that might be crucial for tracking and auditing purposes.
   * **Formula**: HOUR(NOW() - [Date]) <= 12
     + **Explanation**: This part of the formula calculates the difference between the current time (NOW()) and the record's creation time ([Date]). If this difference is less than or equal to 12 hours, the condition is met.
2. **User Authorization**:
   * Only the user who created the record can delete it. This condition ensures that users can only delete their own records and not those created by others.
   * **Formula**: [USERNAME] = [STAFFNAME]
     + **Explanation**: This part of the formula checks if the USERNAME of the record matches the STAFFNAME of the logged-in user. If the usernames match, the condition is met.

#### **Combined Condition:**

**Formula**:

AND(HOUR(NOW() - [Date]) <= 12,[USERNAME] = [STAFFNAME])

**Explanation**: The AND function ensures that both conditions must be true for the delete action to be allowed. This combined formula effectively restricts deletion to records created within the last 12 hours by the user who created them.

#### **Implementation Steps:**

1. Open the AppSheet editor.
2. Navigate to the table where the delete action is to be applied.
3. Go to the "Actions" section and select the delete action or create a new one.
4. In the "Only if this condition is true" field, enter the combined formula provided above.

By applying these conditions, we ensure that our application's data remains secure and that only authorized and recent records can be deleted, maintaining data accuracy and accountability.

### **Automatic Linking of Stock Allocation to Sales Entries**

#### **Overview**

In a business environment where accurate inventory management is crucial, especially when dealing with multiple users, it is essential to ensure that stock allocations are correctly linked to sales entries. This solution provides a method to automatically reference the correct stock allocation entry (ALT\_ID) from the "STOCK ALLOCATION" table whenever a new sale is made in the "SALES" table.

#### **Key Benefits**

* **Accurate Inventory Tracking:** Ensures that each sale is linked to the correct stock allocation, helping to maintain accurate inventory levels.
* **Efficiency:** Automates the process of linking sales to stock allocations, reducing manual errors and saving time.
* **Scalability:** Suitable for businesses with multiple users, ensuring that each user's sales are correctly tracked against their allocated stock.

#### **How It Works**

This solution leverages AppSheet's capabilities to create a reference link between the "SALES" table and the "STOCK ALLOCATION" table based on the product ID (PRD\_ID) and username (USERNAME). When a sale is made, the ALT\_ID from the stock allocation table is automatically populated in the sales table, ensuring accurate tracking and management of inventory.

#### **Implementation Steps**

1. **Add ALT\_ID Column to Sales Table:**

* Navigate to Data > Columns in your AppSheet app.
* Select the "SALES" table and add a new column named ALT\_ID.
* Set the column type to "Ref" and reference the "STOCK ALLOCATION" table.

1. **Set App Formula for ALT\_ID:**

* In the AppFormula field of the ALT\_ID column in the "SALES" table, enter the following formula:  
    
  ANY(FILTER("STOCK ALLOCATION",AND([PRD\_ID] = [\_THISROW].[PRD\_ID],[USERNAME] = [\_THISROW].[USERNAME])))
* This formula ensures that the ALT\_ID from the stock allocation table is automatically linked to the sales entry based on the product ID and username.

1. **Testing and Verification:**
   1. Create new sales entries and verify that the ALT\_ID is correctly populated.
   2. Ensure that the linkage between the sales and stock allocation entries works as expected.

#### **Conclusion**

By implementing this solution, businesses can significantly enhance their inventory management processes. The automatic linking of stock allocations to sales ensures accurate tracking, reduces manual errors, and improves overall efficiency in managing inventory. This solution is particularly beneficial for businesses with multiple users, as it maintains precise stock levels and allocation records.

**SOME KPIS & METRICS ESSENTIAL FOR INVENTORY MANAGEMENT**

**OR**

**FOR TRACKING**

**.Stock Turnover Rate (also known as Inventory Turnover Ratio):**

The Stock Turnover Rate, also known as the Inventory Turnover Ratio, is a financial metric that measures how efficiently a company manages its inventory. It indicates how many times a company's inventory is sold and replaced *(means the cycle of selling the inventory and then restocking it with new inventory)* over a specific period, usually a year. This ratio helps assess whether a company has excessive inventory compared to its sales, which can tie up capital and increase holding costs, or if it efficiently moves its inventory, indicating good sales performance.

### **Formula:**

The Stock Turnover Rate can be calculated using the following formula:



**COMPONENTS:**

1. **Cost of Goods Sold (COGS):**
   * The direct costs attributable to the production of the goods sold by a company. This includes the cost of materials and labor directly used to create the product.
2. **Average Inventory:**
   * The average amount of inventory held during a period. It can be calculated as: Average Inventory=Beginning Inventory+Ending Inventory2\text{Average Inventory} = \frac{\text{Beginning Inventory} + \text{Ending Inventory}}{2}Average Inventory=2Beginning Inventory+Ending Inventory​

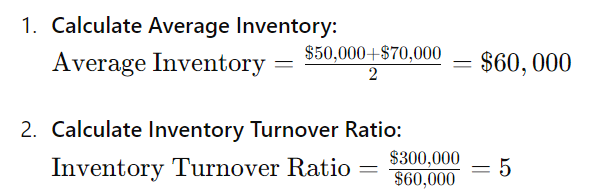
### 

### 

### **Example:**

Let's say a company has the following data for the year 2023:

* Beginning Inventory: $50,000
* Ending Inventory: $70,000
* Cost of Goods Sold (COGS): $300,000



This means the company's inventory turned over five times during the year.

### **Interpretation:**

* **High Turnover Ratio:** Indicates strong sales or effective inventory management. It suggests that the company sells its inventory quickly.
* **Low Turnover Ratio:** May indicate overstocking, obsolescence, or weak sales. This can tie up capital and increase storage and holding costs.

### **Improving Inventory Turnover:**

To improve the Stock Turnover Rate, companies can:

* **Enhance Sales Efforts:** Boosting sales through marketing and promotions.
* **Optimize Inventory Levels:** Implementing just-in-time (JIT) inventory management.
* **Streamline Supply Chain:** Improving the efficiency of the supply chain to reduce lead times.

By monitoring and managing the Inventory Turnover Ratio, companies can improve their operational efficiency and profitability.

**QUESTION TO ASK FROM ANAND BHAIYA**

RATE NIKALTE TIME DISCOUNT INCLUDE KARTE HAI KI NAHI

To get some ideas regarding dashboard solve sales insights project of power bi

[Power BI Project For Beginners | Sales Insights Data Analysis Project - 1 - Problem Statement](https://www.youtube.com/watch?v=hhZ62IlTxYs&list=PLeo1K3hjS3uva8pk1FI3iK9kCOKQdz1I9)