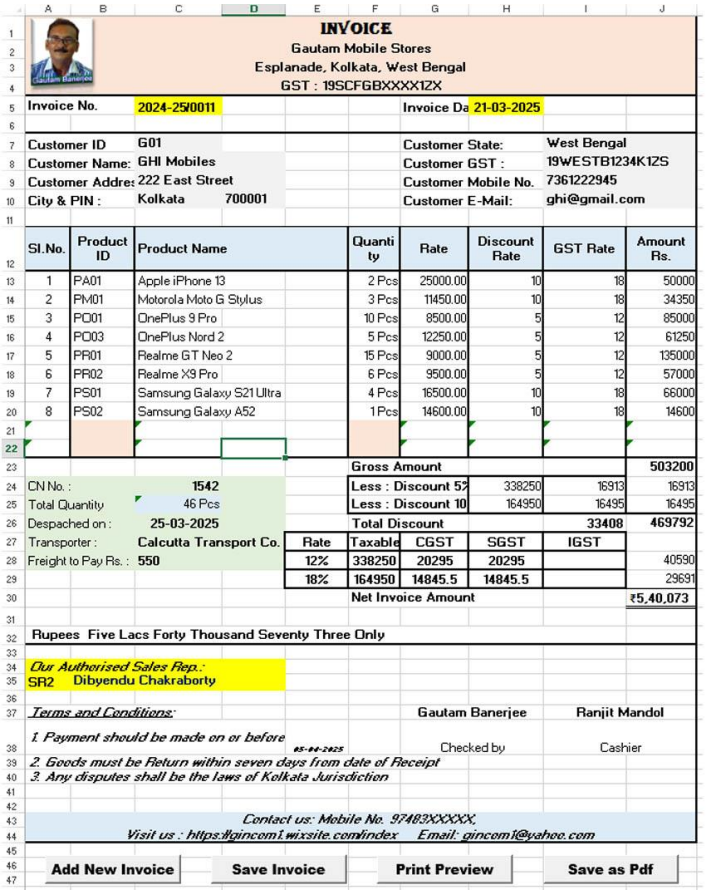
Explanation of the SaveInvoice VBA Code!



INVOICE

Gautam Mobile Stores
Esplanade, Kolkata, West Bengal
GST : 19SCFGBXXXX1ZX

Invoice No.	2024-2500011	Invoice Da	21-03-2025
Customer ID	G01	Customer State:	West Bengal
Customer Name:	GHI Mobiles	Customer GST :	19WESTB1234K1ZS
Customer Addre:	222 East Street	Customer Mobile No.	7361222945
City & PIN :	Kolkata 700001	Customer E-Mail:	ghi@gmail.com

Sl.No.	Product ID	Product Name	Quanti ty	Rate	Discount Rate	GST Rate	Amount Rs.
1	PA01	Apple iPhone 13	2 Pcs	25000.00	10	18	50000
2	PM01	Motorola Moto G Stylus	3 Pcs	11450.00	10	18	34350
3	PO01	OnePlus 9 Pro	10 Pcs	8500.00	5	12	85000
4	PO03	OnePlus Nord 2	5 Pcs	12250.00	5	12	61250
5	PR01	Realme GT Neo 2	15 Pcs	9000.00	5	12	135000
6	PR02	Realme X9 Pro	6 Pcs	9500.00	5	12	57000
7	PS01	Samsung Galaxy S21 Ultra	4 Pcs	16500.00	10	18	66000
8	PS02	Samsung Galaxy A52	1 Pcs	14600.00	10	18	14600

Gross Amount					503200
Less : Discount 5%	338250	16913			16913
Less : Discount 10%	164950	16495			16495
Total Discount		33408			469792

CN No. :	1542
Total Quantity	46 Pcs
Despached on :	25-03-2025
Transporter :	Calcutta Transport Co.
Freight to Pay Rs. :	550

Rate	Taxable	CGST	SGST	IGST	
12%	338250	20295	20295		40590
18%	164950	14845.5	14845.5		29691
Net Invoice Amount					₹5,40,073

Rupees Five Lacs Forty Thousand Seventy Three Only

Our Authorised Sales Rep.:
SR2 Dibyendu Chakraborty

Terms and Conditions:

1. Payment should be made on or before 05-04-2025
2. Goods must be Return within seven days from date of Receipt
3. Any disputes shall be the laws of Kolkata Jurisdiction

Gautam Banerjee — Checked by
Ranjit Mandol — Cashier

Contact us: Mobile No. 97483XXXXX,
Visit us : https://gincom1.wixsite.com/index Email: gincom1@yahoo.com

Add New Invoice | Save Invoice | Print Preview | Save as Pdf

Explanation of the SaveInvoice VBA Code

```
Sub SaveInvoice()
    Dim wsInvoice As Worksheet, wsBillsFile As Worksheet, wsSalesFile As Worksheet
    Dim wsProduct As Worksheet
    Dim i As Long, BillRow As Long, SalesRow As Long, ProductRow As Long
    Dim InvoiceNo As String, lastInvoiceNo As String, newInvoiceNo As String
    Dim lastRow As Long, serialNumber As Integer
    Dim duplicateCheck As Range
    Dim tot_Qty As Double, amt_Disc As Double, Amt_Taxable As Double, Amt_GST As Double
    Dim custGSTIN As String, invoiceDate As Date, finYear As String, totalInvoiceAmount As Double
    Dim rng As Range, cell As Range
    Dim hasProducts As Boolean
    Dim backupPath As String
    Dim ProductID As String, StockIN As Double, StockOUT As Double, StockInHand As Double

    ' Error handling setup
    On Error GoTo ErrorHandler

    ' Initialize worksheet references with validation
    If Not InitializeWorksheets(wsInvoice, wsBillsFile, wsSalesFile) Then Exit Sub

    ' Validate invoice date
    invoiceDate = wsInvoice.Range("H5").Value
    If Not ValidateInvoiceDate(invoiceDate, wsInvoice) Then Exit Sub

    ' Check if products exist in invoice
    If Not InvoiceHasProducts(wsInvoice) Then
        MsgBox "Cannot save invoice with no products. Please add at least one product.", vbExclamation,
"No Products"
        Exit Sub
    End If

    ' Generate invoice number
    finYear = GetFinancialYear(invoiceDate)
    InvoiceNo = GenerateInvoiceNumber(wsBillsFile, finYear, invoiceDate)
    wsInvoice.Range("C5").Value = InvoiceNo

    ' Check for duplicate invoice number
    If InvoiceNumberExists(wsBillsFile, InvoiceNo) Then
        MsgBox "Invoice Number already exists in BillsFile!", vbExclamation, "Error"
        Exit Sub
    End If
```

```
    ' Find next empty rows in destination sheets
    BillRow = wsBillsFile.Cells(wsBillsFile.Rows.Count, 1).End(xlUp).Row + 1
    SalesRow = wsSalesFile.Cells(wsSalesFile.Rows.Count, 1).End(xlUp).Row + 1

    ' Process each product line in the invoice
    i = 13
    Do While wsInvoice.Cells(i, 2).Value <> "" Or wsInvoice.Cells(i, 3).Value <> ""
        If wsInvoice.Cells(i, 2).Value <> "" Or wsInvoice.Cells(i, 3).Value <> "" Then
            ' Save to BillsFile
            SaveToBillsFile wsBillsFile, BillRow, wsInvoice, i, InvoiceNo, invoiceDate, finYear,
totalInvoiceAmount

            ' Save to Product sheet if Product ID exists
            If wsInvoice.Cells(i, 2).Value <> "" Then
                ProductID = wsInvoice.Cells(i, 2).Value
                Set wsProduct = GetProductSheet(ProductID)
                ProductRow = wsProduct.Cells(wsProduct.Rows.Count, 1).End(xlUp).Row + 1

                ' Calculate stock values
                StockOUT = wsInvoice.Cells(i, 6).Value
                StockIN = 0
                StockInHand = GetPreviousStockInHand(wsProduct, ProductRow)

                ' Save to Product sheet
                SaveToProductSheet wsProduct, ProductRow, wsInvoice, i, InvoiceNo, invoiceDate, finYear,
ProductID, StockIN, StockOUT, StockInHand

                BillRow = BillRow + 1
            End If
        End If
        i = i + 1
    Loop

    ' Save summary to SalesFile
    SaveToSalesFile wsSalesFile, SalesRow, wsInvoice, InvoiceNo, invoiceDate, finYear

    ' Post-save operations
    PostSaveOperations wsInvoice

    ' Create backup
    CreateBackup

    MsgBox "Invoice saved successfully!", vbInformation, "Success"
```

```
    Exit Sub

ErrorHandler:
    MsgBox "Error " & Err.Number & ": " & Err.Description & vbCrLf & "Source: " & Err.Source, vbCritical,
"Error"
    Err.Clear
End Sub
```

This VBA subroutine is designed to save invoice data from a worksheet to multiple destination worksheets in an Excel workbook. Here's a detailed breakdown of what it does:

Main Components

1. **Variable Declarations**:
    - Declares worksheet objects for source and destination sheets
    - Declares variables for row counters, invoice numbers, dates, financial year, and stock quantities
    - Declares variables for tracking totals (quantity, discount, taxable amount, GST)
2. **Error Handling**:
    - Sets up an error handler to catch and display any runtime errors

Key Operations

1. **Worksheet Initialization**:
    - Calls ***InitializeWorksheets*** to set references to the invoice, bills file, and sales file worksheets
    - Exits if initialization fails
2. **Validation Checks**:
    - Validates the invoice date using ***ValidateInvoiceDate***
    - Checks if the invoice contains any products using ***InvoiceHasProducts***
    - Exits if any validation fails
3. **Invoice Number Generation**:
    - Determines the financial year from the invoice date
    - Generates a new invoice number using ***GenerateInvoiceNumber***
    - Checks for duplicate invoice numbers using ***InvoiceNumberExists***
4. **Data Processing**:

- Finds the next empty rows in the BillsFile and SalesFile worksheets
- Loops through each product line in the invoice (starting from row 13)
- For each product:
  - Saves to BillsFile using ***SaveToBillsFile***
  - If a Product ID exists, gets the corresponding product sheet using ***GetProductSheet***
  - Calculates stock values (StockOUT from invoice, StockIN as 0, and gets previous StockInHand)
  - Saves to Product sheet using ***SaveToProductSheet***

5. **Summary Save**:
   - Saves invoice summary to SalesFile using ***SaveToSalesFile***
6. **Post-Save Operations**:
   - Performs cleanup/reset operations on the invoice sheet using ***PostSaveOperations***
   - Creates a backup using ***CreateBackup***
   - Displays success message

1. **Core Invoice Processing**

Name	Type	Purpose
SaveInvoice	Sub	Main routine to save invoice data to multiple sheets (BillsFile, Product sheets, SalesFile)
PostSaveOperations	Sub	Updates UI after saving (disables Save button, enables Add New, colors tab green)
CreateBackup	Sub	Saves timestamped backup copy in "\Backups" folder

2. Validation Functions

Name	Purpose	Returns
InitializeWorksheets	Checks if required sheets ("BillsFile", "SalesFile") exist	Boolean
ValidateInvoiceDate	Ensures date is within ±5 days of today	Boolean
InvoiceHasProducts	Checks if invoice has at least one product line	Boolean
InvoiceNumberExists	Verifies if invoice number already exists in BillsFile	Boolean

3. Financial Year & Numbering

Name	Purpose	Example Output
GetFinancialYear	Calculates financial year (April-March)	"2023-24"

Name	Purpose	Example Output
GenerateInvoiceNumber	Creates sequential invoice number (YYYY-YY/000X)	"2023-24/0005"

4. Data Saving Routines

Name	Saves To	Key Features
SaveToBillsFile	BillsFile	Detailed line items, tax calculations (CGST/SGST/IGST)
SaveToProductSheet	Product-specific sheets	Stock updates (IN/OUT/In Hand), transaction history
SaveToSalesFile	SalesFile	Consolidated invoice summary for reporting

5. Product Inventory Helpers

Name	Purpose	Notes
GetProductSheet	Finds/Creates product-specific sheet	Auto-creates with headers if missing
GetPreviousStockInHand	Retrieves last stock balance	Returns 0 for new products

Key Features

1. **Modular Design**: Uses several helper functions (shown by the capitalized function names) to break down complex operations.
2. **Data Validation**: Ensures data integrity through multiple validation checks before processing.
3. **Comprehensive Saving**: Saves data to multiple destinations (BillsFile, Product sheets, SalesFile) for different reporting needs.
4. **Stock Management**: Tracks stock movements by updating product-specific worksheets.
5. **Financial Year Handling**: Incorporates financial year awareness in invoice numbering.

This code appears to be part of a larger inventory and billing system, designed to maintain accurate records across multiple interconnected worksheets.

Here's an explanation of each variables:

```
Dim wsInvoice As Worksheet, wsBillsFile As Worksheet, wsSalesFile As Worksheet
    Dim wsProduct As Worksheet
    Dim i As Long, BillRow As Long, SalesRow As Long, ProductRow As Long
    Dim InvoiceNo As String, lastInvoiceNo As String, newInvoiceNo As String
    Dim lastRow As Long, serialNumber As Integer
    Dim duplicateCheck As Range
    Dim tot_Qty As Double, amt_Disc As Double, Amt_Taxable As Double, Amt_GST As Double
    Dim custGSTIN As String, invoiceDate As Date, finYear As String, totalInvoiceAmount As Double
    Dim rng As Range, cell As Range
    Dim hasProducts As Boolean
    Dim backupPath As String
    Dim ProductID As String, StockIN As Double, StockOUT As Double, StockInHand As Double
```

Worksheet Variables

wsInvoice As Worksheet

Represents a worksheet object for the "Invoice" sheet where invoice data is stored or generated.

wsBillsFile As Worksheet

Represents a worksheet object for the "BillsFile" sheet, likely containing billing-related data.

wsSalesFile As Worksheet

Represents a worksheet object for the "SalesFile" sheet, likely containing sales transaction records.

wsProduct As Worksheet

Represents a worksheet object for the "Product" sheet, which contains product details, stock levels, etc.

Loop & Row Counter Variables

i As Long

A loop counter, typically used in For loops to iterate through rows or columns.

BillRow As Long

Keeps track of the current row number in the "BillsFile" worksheet.

SalesRow As Long

Keeps track of the current row number in the "SalesFile" worksheet.

ProductRow As Long

Keeps track of the current row number in the "Product" worksheet.

lastRow As Long

Stores the last used row number in a worksheet.

serialNumber As Integer

Used to assign a serial number (e.g., for line items in an invoice).

Invoice Number Variables

InvoiceNo As String

Holds the current invoice number being processed.

lastInvoiceNo As String

Stores the last generated invoice number (used for incrementing to the next number).

newInvoiceNo As String
Stores the newly generated invoice number.

Range & Cell Variables
duplicateCheck As Range
A range variable used to check if an invoice number already exists (duplicate check).
rng As Range
A generic range variable, used for looping through cells.
cell As Range
Represents a single cell within a loop (e.g., For Each cell In rng).

Financial & Quantity Variables
tot_Qty As Double
Stores the total quantity of items sold (allows decimal values, e.g., 2.5 kg).
amt_Disc As Double
Stores the total discount amount applied to an invoice.
Amt_Taxable As Double
Stores the taxable amount before GST (Goods and Services Tax).
Amt_GST As Double
Stores the GST amount calculated on the taxable amount.
totalInvoiceAmount As Double
Stores the final invoice amount after discounts and taxes.

Customer & Invoice Details
custGSTIN As String
Stores the customer's GST Identification Number (GSTIN).
invoiceDate As Date
Stores the date of the invoice.
finYear As String
Stores the financial year (e.g., "2023-24") for record-keeping.

Product & Stock Management
ProductID As String
Stores the unique identifier for a product (e.g., "PRD001").
StockIN As Double
Tracks the quantity of stock received (allows decimal values).
StockOUT As Double
Tracks the quantity of stock sold or dispatched.
StockInHand As Double
Calculates the remaining stock (StockInHand = StockIN - StockOUT).

Boolean & Path Variables
hasProducts As Boolean
A flag (True/False) to check if there are products listed in an invoice.

backupPath As String
Stores the file path where backups (e.g., invoice copies) are saved.

Summary
These variables are used for:
Worksheet handling (wsInvoice, wsBillsFile, etc.)
Looping & row tracking (i, BillRow, lastRow, etc.)
Invoice numbering (InvoiceNo, newInvoiceNo, etc.)
Financial calculations (tot_Qty, Amt_GST, etc.)
Stock management (ProductID, StockInHand, etc.)
Data validation & checks (duplicateCheck, hasProducts)

Error Handling

```
On Error GoTo ErrorHandler
```

If any error occurs during execution, the code jumps to the ErrorHandler section which displays:

- Error number
- Error description
- Error source

Explanation of the *InitializeWorksheets* Function

```
Private Function InitializeWorksheets(ByRef wsInvoice As Worksheet, ByRef wsBillsFile As Worksheet, ByRef wsSalesFile As Worksheet) As Boolean
    On Error Resume Next
    Set wsInvoice = ThisWorkbook.Sheets("Invoice")
    Set wsBillsFile = ThisWorkbook.Sheets("BillsFile")
    Set wsSalesFile = ThisWorkbook.Sheets("SalesFile")
    On Error GoTo 0

    If wsBillsFile Is Nothing Or wsSalesFile Is Nothing Then
        MsgBox "Required worksheets (BillsFile or SalesFile) are missing!", vbCritical, "Error"
        InitializeWorksheets = False
    Else
        InitializeWorksheets = True
    End If
```

```
End Function
```

This is a **private helper function** that initializes references to three important worksheets in the Excel workbook. Here's a detailed breakdown:

The keyword `ByRef` stands for **"By Reference"**, which means that the function receives a **reference to the original variable** (worksheet objects in this case) rather than a copy of it.

Purpose

The function attempts to set worksheet object references for:

1. `wsInvoice` - The worksheet where invoice data is entered
2. `wsBillsFile` - The worksheet that stores detailed bill records
3. `wsSalesFile` - The worksheet that stores sales summaries

How It Works

1. **Error Handling Setup**:
   ```
   On Error Resume Next
   ```
   - Temporarily suppresses errors (so if a sheet doesn't exist, it won't crash)
2. **Setting Worksheet References**:
   ```
   Set wsInvoice = ThisWorkbook.Sheets("Invoice")
   Set wsBillsFile = ThisWorkbook.Sheets("BillsFile")
   Set wsSalesFile = ThisWorkbook.Sheets("SalesFile")
   ```
   - Attempts to assign each worksheet object to its corresponding sheet name
3. **Restoring Normal Error Handling**:
   ```
   On Error GoTo 0
   ```
   - Turns normal error handling back on
4. **Validation Check**:
   ```
   If wsBillsFile Is Nothing Or wsSalesFile Is Nothing Then
   ```
   - Checks if either BillsFile or SalesFile failed to initialize (are `Nothing`)
   - Notice it doesn't check `wsInvoice` - suggesting these two are mandatory while Invoice sheet might be optional
5. **Error Handling**:
   ```
   MsgBox "Required worksheets (BillsFile or SalesFile) are missing!", vbCritical, "Error"
   InitializeWorksheets = False
   ```
   - Shows an error message if required sheets are missing
   - Returns `False` to indicate failure
6. **Success Case**:
   ```
   Else
      InitializeWorksheets = True
   ```

```
End If
```

- Returns True if both required sheets were found

Key Points

- **ByRef Parameters**: The worksheet references are passed ByRef (by reference), so the function can modify the original variables passed to it.
- **Selective Validation**: Only checks for BillsFile and SalesFile, implying:
  - These are absolutely required for the system to work
  - The Invoice sheet might be optional or its absence handled elsewhere
- **Error Handling Approach**: Uses On Error Resume Next to gracefully handle missing sheets rather than crashing.
- **Boolean Return**: Returns True/False to indicate success/failure, allowing the calling code to decide what to do next.

Typical Usage

This function would be called at the ***start of a procedure*** to ensure all required worksheets are available before proceeding with operations.

Explanation of the *ValidateInvoiceDate* Function

```
Private Function ValidateInvoiceDate(invoiceDate As Date, wsInvoice As Worksheet) As Boolean
    If invoiceDate < Date - 5 Or invoiceDate > Date + 5 Then
        MsgBox "Invalid Invoice Date! Please select a date within 5 days from today.", vbExclamation,
"Invalid Date"
        wsInvoice.Range("H5").Value = Date
        ValidateInvoiceDate = False
    Else
        ValidateInvoiceDate = True
    End If
End Function
```

This is a **private validation function** that checks whether an invoice date falls within an acceptable range. Here's what it does:

Purpose

The function validates that the invoice date is:

- Not more than 5 days in the past
- Not more than 5 days in the future
  from the current date (today).

Parameters

- `invoiceDate`: The date to be validated (passed by value)
- `wsInvoice`: The worksheet containing the invoice (to update the date if invalid)

Return Value

- Returns `True` if the date is valid
- Returns `False` if the date is invalid

Code Breakdown

1. **Date Validation Check**:
   ```
   If invoiceDate < Date - 5 Or invoiceDate > Date + 5 Then
   ```
   - `Date` is a VBA function that returns the current system date
   - Checks if the invoice date is either:
     - Earlier than 5 days ago (`Date - 5`)
     - Later than 5 days from now (`Date + 5`)
2. **Invalid Date Handling**:
   ```
   MsgBox "Invalid Invoice Date! Please select a date within 5 days from today.", vbExclamation, "Invalid D
   ate"
   wsInvoice.Range("H5").Value = Date
   ValidateInvoiceDate = False
   ```
   - Shows an error message to the user
   - Automatically resets the date in cell H5 to today's date
   - Returns `False` indicating validation failed
3. **Valid Date Case**:
   ```
   Else
       ValidateInvoiceDate = True
   End If
   ```
   - Returns `True` if the date is within the acceptable range

Key Features

1. **Business Rule Enforcement**:

    - Implements a business rule that invoices must be dated within ±5 days of the current date
    - This might be for accounting or reporting purposes
2. **User-Friendly Feedback**:
    - Provides clear feedback about what dates are acceptable
    - Automatically corrects invalid dates to today's date
3. **Worksheet Integration**:
    - Directly modifies the worksheet (cell H5) when invalid dates are found
    - This ensures the invoice always has a valid date before proceeding

Typical Usage

This function would be called before processing an invoice to ensure the date is valid. The calling code would check the return value to decide whether to proceed:

```
If Not ValidateInvoiceDate(invoiceDate, wsInvoice) Then Exit Sub
```

Why This Validation?

The ±5 day restriction likely serves several purposes:

- Prevents accidental entry of dates far in past/future
- Maintains accounting period integrity
- Ensures timely invoicing practices
- Matches business requirements for reporting periods

Explanation of the *InvoiceHasProducts* Function

```
Private Function InvoiceHasProducts(wsInvoice As Worksheet) As Boolean
    Dim i As Long
    i = 13
    Do While wsInvoice.Cells(i, 2).Value <> "" Or wsInvoice.Cells(i, 3).Value <> ""
        InvoiceHasProducts = True
        Exit Function
    Loop
    InvoiceHasProducts = False
End Function
```

This function checks whether an invoice worksheet contains any products. Here's a detailed breakdown:

Purpose

The function determines if there are any products listed in the invoice by scanning rows starting from row 13.

Parameters

- `wsInvoice`: The worksheet containing the invoice data to be checked

Return Value

- Returns `True` if at least one product is found (in column B or C)
- Returns `False` if no products are found

Code Breakdown

1. **Initialization**:
   ```
   Dim i As Long
   i = 13
   ```
   - Sets up a row counter starting at row 13 (typical starting row for invoice line items)
2. **Loop Through Rows**:
   ```
   Do While wsInvoice.Cells(i, 2).Value <> "" Or wsInvoice.Cells(i, 3).Value <> ""
   ```
   - Checks columns B (2) and C (3) of each row
   - Continues while either column has content (product ID in B or description in C)
   - The loop condition checks for non-empty cells
3. **Positive Case Handling**:
   ```
   InvoiceHasProducts = True
   Exit Function
   ```
   - If any row has content, immediately:
     - Sets return value to `True`
     - Exits the function (no need to check further rows)
4. **Negative Case Handling**:
   ```
   InvoiceHasProducts = False
   ```
   - Only reached if the loop completes without finding any products
   - Returns `False` indicating no products found

Key Features

1. **Efficient Design**:
    - Exits immediately upon finding the first product
    - Doesn't unnecessarily scan the entire worksheet
2. **Flexible Validation**:
    - Checks both product ID (column B) and description (column C)
    - This accommodates invoices that might use either field
3. **Row Starting Point**:
    - Begins at row 13, suggesting:
        - Rows 1-12 contain invoice headers/static information
        - Product lines start at row 13

Typical Usage

This function would be called before processing/saving an invoice to ensure it contains at least one product line:

```
If Not InvoiceHasProducts(wsInvoice) Then
    MsgBox "Invoice must contain at least one product!", vbExclamation
    Exit Sub
End If
```

Why This Validation?

Prevents:

- Saving empty invoices
- Processing errors downstream
- Accounting discrepancies from blank invoices
- Wasting system resources on invalid invoices

Explanation of the *GenerateInvoiceNumber* Function

```
Private Function GenerateInvoiceNumber(wsBillsFile As Worksheet, finYear As String, invoiceDate As Date) As String
    Dim lastRow As Long, serialNumber As Integer
    Dim lastInvoiceNo As String
```

```
    Dim rng As Range, cell As Range

    lastRow = wsBillsFile.Cells(wsBillsFile.Rows.Count, 4).End(xlUp).Row
    lastInvoiceNo = wsBillsFile.Cells(lastRow, 4).Value
    serialNumber = 0

    If lastRow >= 2 Then
        Set rng = wsBillsFile.Range("D2:D" & lastRow)
        For Each cell In rng
            If Left(cell.Value, 7) = finYear Then
                lastInvoiceNo = cell.Value
            End If
        Next cell
        If lastInvoiceNo <> "" Then serialNumber = Val(Right(lastInvoiceNo, 4))
    End If

    serialNumber = serialNumber + 1
    GenerateInvoiceNumber = finYear & "/" & Format(serialNumber, "0000")
End Function
```

This function creates a new sequential invoice number based on the financial year and previous invoice numbers in the system.

Purpose

Generates a unique invoice number in the format: `YYYY-YY/000X` where:

- `YYYY-YY` is the financial year (e.g., 2023-24)
- `000X` is a 4-digit sequential number

Parameters

- `wsBillsFile`: Worksheet containing existing invoices
- `finYear`: Financial year string (e.g., "2023-24")
- `invoiceDate`: Date of the invoice (not directly used in this function)

Return Value

Returns a new invoice number as a string.

Code Breakdown

1. Initial Setup

```
Dim lastRow As Long, serialNumber As Integer
Dim lastInvoiceNo As String
Dim rng As Range, cell As Range
```

Declares variables for:

- Tracking the last used row
- Storing the serial number component
- Holding the last invoice number
- Range objects for searching

2. Find Last Invoice

```
lastRow = wsBillsFile.Cells(wsBillsFile.Rows.Count, 4).End(xlUp).Row
lastInvoiceNo = wsBillsFile.Cells(lastRow, 4).Value
```

- Finds the last used row in column D (where invoice numbers are stored)
- Gets the invoice number from that row

3. Initialize Serial Number

```
serialNumber = 0
```

Default starting value if no previous invoices exist

4. Process Existing Invoices

```
If lastRow >= 2 Then
    Set rng = wsBillsFile.Range("D2:D" & lastRow)
    For Each cell In rng
        If Left(cell.Value, 7) = finYear Then
            lastInvoiceNo = cell.Value
        End If
    Next cell
    If lastInvoiceNo <> "" Then serialNumber = Val(Right(lastInvoiceNo, 4))
End If
```

- Checks if there are any existing invoices (row 2+)
- Searches all invoice numbers in column D
- Finds the highest number for the current financial year
- Extracts the 4-digit serial number from matching invoices

5. Generate New Number

```
serialNumber = serialNumber + 1
GenerateInvoiceNumber = finYear & "/" & Format(serialNumber, "0000")
```

- Increments the serial number
- Formats the number with leading zeros (e.g., 1 becomes "0001")
- Combines with financial year to create final invoice number

Key Features

1. **Financial Year Tracking**:
    - Numbers are specific to each financial year
    - Each year's numbering starts fresh (0001)
2. **Robust Searching**:
    - Doesn't assume the last row has the highest number
    - Scans all invoices to find the highest number for the current financial year
3. **Consistent Formatting**:
    - Uses 4-digit serial numbers (0001-9999)
    - Standardized format (YYYY-YY/000X)
4. **Error Handling**:
    - Works even with empty worksheet (starts at 0001)
    - Handles cases where no matching financial year exists

Example Output

For financial year "2023-24":

- **First invoice: "2023-24/0001"**
- **Next invoice: "2023-24/0002"**
- **In 2024-25: "2024-25/0001**" (resets counter)

Typical Usage

Called when creating a new invoice to ensure unique, sequential numbering according to financial year.

Explanation of the *InvoiceNumberExists* Function

```
Private Function InvoiceNumberExists(wsBillsFile As Worksheet, InvoiceNo As String) As Boolean
    Dim duplicateCheck As Range
    Set duplicateCheck = wsBillsFile.Columns(4).Find(InvoiceNo, LookIn:=xlValues, LookAt:=xlWhole)
    InvoiceNumberExists = Not (duplicateCheck Is Nothing)
End Function
```

This function checks whether a given invoice number already exists in the BillsFile worksheet.

Purpose

To prevent duplicate invoice numbers by verifying if a proposed invoice number is already present in the system.

Parameters

- `wsBillsFile`: The worksheet containing all existing invoice records
- `InvoiceNo`: The invoice number to check for duplicates

Return Value

- Returns `True` if the invoice number already exists
- Returns `False` if the invoice number is unique

Code Breakdown

1. Variable Declaration

```
Dim duplicateCheck As Range
```

Declares a Range object to store the result of the search.

2. Searching for the Invoice Number

```
Set duplicateCheck = wsBillsFile.Columns(4).Find(InvoiceNo, LookIn:=xlValues, LookAt:=xlWhole)
```

This line performs several important operations:

- `wsBillsFile.Columns(4)`: Searches in column D (4th column) where invoice numbers are stored
- `.Find()` method parameters:
    - `InvoiceNo`: The number to search for
    - `LookIn:=xlValues`: Searches in cell values (not formulas or formats)
    - `LookAt:=xlWhole`: Looks for exact whole-cell matches (not partial matches)

3. Determining the Result

```
InvoiceNumberExists = Not (duplicateCheck Is Nothing)
```

- If the invoice number is found, duplicateCheck contains the cell reference (not Nothing)
- If not found, duplicateCheck is Nothing
- The Not operator converts this to a Boolean result:
    - Found → True (exists)
    - Not found → False (doesn't exist)

Key Features

1. **Efficient Search**:
    - Uses Excel's native Find method which is optimized for quick searching
    - Searches only the relevant column (D/4)
2. **Exact Matching**:
    - xlWhole ensures only exact matches are considered
    - Prevents false positives from similar numbers
3. **Simple Boolean Return**:
    - Clean true/false interface for calling code
    - Easy to use in conditional statements
4. **Minimalist Design**:
    - Does one thing well (existence check)
    - No unnecessary complexity

Typical Usage

Called before saving a new invoice to ensure number uniqueness:

```
If InvoiceNumberExists(wsBillsFile, newInvoiceNo) Then
    MsgBox "This invoice number already exists!", vbExclamation
    Exit Sub
End If
```

Why This Matters

- Maintains data integrity by preventing duplicates
- Critical for accounting and audit trails
- Ensures each invoice has a unique identifier
- Prevents confusion in reporting and tracking

Here's a detailed explanation of the *SaveToBillsFile* subroutine:

```
Private Sub SaveToBillsFile(wsBillsFile As Worksheet, BillRow As Long, wsInvoice As Worksheet, i As Long, InvoiceNo As String, invoiceDate As Date, finYear As String, ByRef totalInvoiceAmount As Double)
    Dim amt_Disc As Double, Amt_Taxable As Double, Amt_GST As Double
    Dim custGSTIN As String

    With wsBillsFile
        ' Invoice & Customer Details
        .Cells(BillRow, 1).Value = wsInvoice.Range("C7").Value   ' Customer_ID
        .Cells(BillRow, 2).Value = wsInvoice.Range("C8").Value   ' Customer_Name
        .Cells(BillRow, 3).Value = wsInvoice.Range("I7").Value   ' State
        .Cells(BillRow, 4).Value = InvoiceNo                     ' Invoice_No
        .Cells(BillRow, 5).Value = invoiceDate                   ' Invoice_Date
        .Cells(BillRow, 6).Value = "SL"                          ' Invoice_Type

        ' Product Details
        .Cells(BillRow, 7).Value = wsInvoice.Cells(i, 2).Value   ' Product_ID
        .Cells(BillRow, 8).Value = wsInvoice.Cells(i, 3).Value   ' Product_Name
        .Cells(BillRow, 9).Value = wsInvoice.Cells(i, 6).Value   ' Quantity
        .Cells(BillRow, 10).Value = wsInvoice.Cells(i, 7).Value  ' Product_Rate
        .Cells(BillRow, 11).Value = wsInvoice.Cells(i, 10).Value ' Product_Value
        .Cells(BillRow, 12).Value = wsInvoice.Cells(i, 8).Value  ' Discount rate

        ' Discount Calculation
        amt_Disc = Round(wsInvoice.Cells(i, 10).Value * wsInvoice.Cells(i, 8).Value / 100, 0)
        .Cells(BillRow, 13).Value = amt_Disc   ' Discount Amount

        ' Taxable Amount Calculation
        Amt_Taxable = wsInvoice.Cells(i, 10).Value - amt_Disc
        .Cells(BillRow, 14).Value = Amt_Taxable  ' Taxable Amount

        .Cells(BillRow, 15).Value = wsInvoice.Cells(i, 9).Value  ' GST rate

        ' GST Calculation
        Amt_GST = Round(Amt_Taxable * wsInvoice.Cells(i, 9).Value / 100, 0)
        custGSTIN = wsInvoice.Range("I8").Value

        ' Check if the customer GSTIN indicates a West Bengal customer
        If Left(Trim(custGSTIN), 2) = "19" Then
            ' Intrastate: Apply CGST and SGST
```

```
            .Cells(BillRow, 16).Value = Amt_GST          ' Total GST Amount (optional)
            .Cells(BillRow, 17).Value = Amt_GST / 2      ' CGST Amount
            .Cells(BillRow, 18).Value = Amt_GST / 2      ' SGST Amount
            .Cells(BillRow, 19).Value = 0                ' IGST Amount (not applicable)
        Else
            ' Interstate: Apply IGST only
            .Cells(BillRow, 16).Value = 0                ' GST Amount (if you want to leave it blank)
            .Cells(BillRow, 17).Value = 0                ' CGST Amount
            .Cells(BillRow, 18).Value = 0                ' SGST Amount
            .Cells(BillRow, 19).Value = Amt_GST          ' IGST Amount
        End If

        ' Invoice Amount Calculation
        .Cells(BillRow, 20).Value = Amt_Taxable + Amt_GST ' Invoice Amount
        totalInvoiceAmount = totalInvoiceAmount + .Cells(BillRow, 20).Value

        ' Sales Rep & Transport Details
        .Cells(BillRow, 23).Value = wsInvoice.Range("C24").Value ' CN No
        .Cells(BillRow, 24).Value = wsInvoice.Range("C25").Value ' No of Item
        .Cells(BillRow, 25).Value = wsInvoice.Range("C26").Value ' Despatched on
        .Cells(BillRow, 26).Value = wsInvoice.Range("C27").Value ' Transporter name
        .Cells(BillRow, 27).Value = wsInvoice.Range("C28").Value ' Freight to Pay
        .Cells(BillRow, 21).Value = wsInvoice.Range("A35").Value ' SalesRep_ID
        .Cells(BillRow, 22).Value = wsInvoice.Range("B35").Value ' SalesRep_Name

        ' Date-related fields
        .Cells(BillRow, 28).Value = Format(invoiceDate, "mmm") ' Month
        .Cells(BillRow, 30).Value = "Q" & (Int((Month(invoiceDate) - 1) / 3) + 1) ' Quarter
        .Cells(BillRow, 31).Value = Year(invoiceDate) ' Year
        .Cells(BillRow, 29).Value = IIf(Day(invoiceDate) <= 15, "1F", "2F") ' Fortnight
        .Cells(BillRow, 32).Value = finYear ' Financial Year
    End With
End Sub
```

Purpose

This subroutine saves individual line items from an invoice to a master BillsFile worksheet, including all relevant product details, taxes, and customer information.

Parameters

- `wsBillsFile`: Target worksheet where bill records are stored
- `BillRow`: Row number where to insert the new record
- `wsInvoice`: Source invoice worksheet
- `i`: Current row number in the invoice being processed

- InvoiceNo: Generated invoice number
- invoiceDate: Date of the invoice
- finYear: Financial year for the invoice
- totalInvoiceAmount (ByRef): Running total that gets updated with each line item

Detailed Breakdown

1. **Customer and Invoice Header Information**:
    - Copies customer ID, name, and state from fixed cells in the invoice (C7, C8, I7)
    - Stores the generated invoice number and date
    - Sets invoice type as "SL" (meaning "Sales")
2. **Product Details**:
    - Copies product ID, name, quantity, rate, and value from the current invoice row
    - Stores discount percentage from column 8 of the invoice
3. **Financial Calculations**:
    - Calculates discount amount (product value × discount %)
    - Computes taxable amount (product value - discount)
    - Calculates GST based on the taxable amount and GST rate
    - Handles different tax scenarios:
        - For West Bengal customers (GSTIN starts with "19"):
            - Splits GST equally as CGST and SGST
        - For other states:
            - Applies full amount as IGST
4. **Invoice Amounts**:
    - Calculates final line item amount (taxable + GST)
    - Updates the running total invoice amount (passed ByRef)
5. **Logistics Information**:
    - Stores consignment note (CN) details
    - Records transportation information
    - Saves sales representative details
6. **Date Analytics**:
    - Extracts and stores various date components:
        - Month name (e.g., "Jan")
        - Fortnight (1F for 1st-15th, 2F for 16th-end)
        - Quarter (Q1-Q4)
        - Calendar year
        - Financial year

Key Features

1. **Comprehensive Data Capture**:
    - Captures all essential invoice elements in one operation
    - Handles both product details and header information
2. **Tax Calculation Logic**:
    - Automatically determines intra-state (CGST+SGST) vs inter-state (IGST) taxation
    - Uses GSTIN prefix (19 for West Bengal) to determine tax treatment
3. **Financial Tracking**:
    - Maintains a running total of the invoice amount
    - Preserves all calculation components (gross, discount, taxable, taxes)
4. **Reporting-Friendly Fields**:
    - Creates derived date fields (month, quarter, fortnight) for easy reporting
    - Stores both calendar and financial year
5. **Structured Organization**:
    - Groups related fields together in the target worksheet
    - Uses consistent column positions for specific data types

Typical Flow

This would be called for each product line in an invoice, with:

- BillRow incrementing for each line item
- i representing the current invoice row being processed
- *totalInvoiceAmount* accumulating the invoice total

The ByRef *totalInvoiceAmount* allows the calling procedure to track the complete invoice value across multiple line items.

Error Handling

Note this subroutine doesn't include explicit error handling, suggesting it relies on the calling procedure's error handling (as seen in the parent SaveInvoice procedure).

Explanation of the *GetProductSheet* Function

```
Private Function GetProductSheet(ProductID As String) As Worksheet
    On Error Resume Next
    Set GetProductSheet = ThisWorkbook.Sheets(ProductID)
```

```
    On Error GoTo 0

    If GetProductSheet Is Nothing Then
        Set GetProductSheet =
ThisWorkbook.Sheets.Add(After:=ThisWorkbook.Sheets(ThisWorkbook.Sheets.Count))
        GetProductSheet.Name = ProductID

        ' Create Product Sheet Headers
        With GetProductSheet
            .Range("A1:P1").Value = Array("Invoice_No", "Invoice_Date", "Invoice_Type", "Customer_ID",
"Customer_Name", _
                                          "Product_ID", "Product_Name", "Product_Rate", "Product_Value", "GST_Rate", _
                                          "Month", "FAYear", "Pieces", "Stock_IN", "Stock_OUT", "Stock_In_Hand")
        End With
    End If
End Function
```

This function retrieves or creates a dedicated worksheet for tracking a specific product's inventory and sales history.

Purpose

To provide a product-specific worksheet for:

1. Maintaining inventory records (stock in/out)
2. Tracking sales history
3. Storing product transaction details

Parameters

- ProductID: The unique identifier for the product (used as sheet name)

Return Value

Returns a Worksheet object for the specified product (either existing or newly created)

Code Breakdown

1. Error Handling Setup

```
On Error Resume Next
```

- Temporarily suppresses errors (for when sheet doesn't exist)

2. Attempt to Get Existing Sheet

```
Set GetProductSheet = ThisWorkbook.Sheets(ProductID)
```

- Tries to find a worksheet with the exact ProductID as its name

3. Restore Error Handling

```
On Error GoTo 0
```

- Returns to normal error handling behavior

4. Create New Sheet if Needed

```
If GetProductSheet Is Nothing Then
```

- Checks if no existing sheet was found

5. Sheet Creation Process

```
Set GetProductSheet = ThisWorkbook.Sheets.Add(After:=ThisWorkbook.Sheets(ThisWorkbook.Sheets.Count))
GetProductSheet.Name = ProductID
```

- Adds new sheet at the end of all existing sheets
- Names it with the ProductID

6. Initialize Sheet Headers

```
With GetProductSheet
    .Range("A1:P1").Value = Array("Invoice_No", "Invoice_Date", "Invoice_Type", "Customer_ID", "Customer_Name", _
                                  "Product_ID", "Product_Name", "Product_Rate", "Product_Value", "GST_Rate", _
                                  "Month", "FAYear", "Pieces", "Stock_IN", "Stock_OUT", "Stock_In_Hand")
End With
```

- Sets up 16 standardized column headers (A-P)
- Creates a complete tracking structure for the product

Key Features

1. **Lazy Initialization**:
    - Only creates the sheet when first needed
    - Avoids creating unused product sheets
2. **Standardized Structure**:
    - Ensures all product sheets have identical column layouts
    - Enables consistent reporting across products
3. **Inventory Tracking**:
    - Dedicated columns for stock movements (IN/OUT/In Hand)
    - Ties stock changes to specific invoices
4. **Sales History**:
    - Records all relevant invoice details
    - Maintains pricing and tax information

5. **Financial Tracking**:
    - Includes financial year (FAYear) for reporting
    - Captures month for period-based analysis

Why This Matters

- Centralizes all product-specific data
- Enables accurate stock tracking
- Supports product performance analysis
- Maintains complete audit trail for each product
- Facilitates inventory reconciliation

The function exemplifies a robust inventory management pattern where each product automatically gets its own tracking worksheet when first encountered in the system.

Explanation of the *GetPreviousStockInHand* Function

```
Private Function GetPreviousStockInHand(wsProduct As Worksheet, ProductRow As Long) As Double
    If ProductRow > 2 Then
        GetPreviousStockInHand = wsProduct.Cells(ProductRow - 1, 16).Value
    Else
        GetPreviousStockInHand = 0
    End If
End Function
```

This function retrieves the previous stock balance for a product from its dedicated worksheet.

Purpose

To get the last recorded "Stock In Hand" value for a product before adding new transactions.

Parameters

- `wsProduct`: The product-specific worksheet
- `ProductRow`: The next available row where new data will be written

Return Value

Returns the previous stock quantity as a Double:

- The value from column P (16) of the previous row if data exists
- 0 if this is the first entry for the product

Code Breakdown

1. Check for Existing Data

```
If ProductRow > 2 Then
```

- Checks if we're past the header row (row 1) and first data row (row 2)
- Row numbers:
    - Row 1: Column headers
    - Row 2: First transaction
    - Row 3+: Subsequent transactions

2. Get Previous Stock Value

```
GetPreviousStockInHand = wsProduct.Cells(ProductRow - 1, 16).Value
```

- If previous data exists (ProductRow > 2):
    - Gets value from column P (16) which contains "Stock_In_Hand"
    - Uses the row before the current insertion point (ProductRow - 1)

3. Default for New Products

```
Else
    GetPreviousStockInHand = 0
End If
```

- If no previous transactions exist (ProductRow ≤ 2):
    - Returns 0 as the starting stock quantity
    - This would be the case for a brand new product

Key Features

1. **Inventory Continuity**:
    - Maintains a running balance of stock quantities
    - Each new transaction starts from the previous balance
2. **Error Prevention**:
    - Handles the case of new products gracefully
    - Won't fail if no previous data exists
3. **Column Reference**:

    - Column 16 (P) is hardcoded as the "Stock_In_Hand" column
    - Matches the header setup in GetProductSheet
4. **Simple Logic**:
    - Straightforward row position check
    - Clear default value for new products

Typical Usage

Used when adding new stock transactions to calculate current stock:

```
Dim currentStock As Double
currentStock = GetPreviousStockInHand(wsProduct, nextRow) + newStockIn - newStockOut
```

Why This Matters

- Essential for accurate inventory tracking
- Ensures each transaction properly updates from the last known quantity
- Provides reliable starting point (0) for new products
- Maintains data integrity in the stock management system

This function works in conjunction with the product worksheet structure created by GetProductSheet, where column P (16) consistently holds the running stock balance.

Here's a detailed explanation of the *SaveToProductSheet* subroutine:

```
Private Sub SaveToProductSheet(wsProduct As Worksheet, ProductRow As Long, wsInvoice As
Worksheet, i As Long, InvoiceNo As String, invoiceDate As Date, finYear As String, ProductID As String, _
                    StockIN As Double, StockOUT As Double, StockInHand As Double)
  With wsProduct
    .Cells(ProductRow, 1).Value = InvoiceNo
    .Cells(ProductRow, 2).Value = invoiceDate
    .Cells(ProductRow, 3).Value = "SL"
    .Cells(ProductRow, 4).Value = wsInvoice.Range("C7").Value ' Customer_ID
    .Cells(ProductRow, 5).Value = wsInvoice.Range("C8").Value ' Customer_Name
    .Cells(ProductRow, 6).Value = ProductID
    .Cells(ProductRow, 7).Value = wsInvoice.Cells(i, 3).Value ' Product_Name
    .Cells(ProductRow, 8).Value = wsInvoice.Cells(i, 7).Value ' Product_Rate
    .Cells(ProductRow, 9).Value = wsInvoice.Cells(i, 10).Value ' Product_Value
    .Cells(ProductRow, 10).Value = wsInvoice.Cells(i, 9).Value ' GST_Rate
```

```
        .Cells(ProductRow, 11).Value = Format(invoiceDate, "mmm") ' Month
        .Cells(ProductRow, 12).Value = finYear ' Financial Year
        .Cells(ProductRow, 13).Value = "Pieces"
        .Cells(ProductRow, 14).Value = StockIN
        .Cells(ProductRow, 15).Value = StockOUT
        .Cells(ProductRow, 16).Value = StockInHand + StockIN - StockOUT
    End With
End Sub
```

Purpose

This subroutine records a product transaction to a product-specific worksheet, updating all relevant details including stock movements.

Parameters

- wsProduct: The target product worksheet
- ProductRow: Row number where to insert the record
- wsInvoice: Source invoice worksheet
- i: Current row in invoice being processed
- InvoiceNo, invoiceDate, finYear: Invoice identification
- ProductID: The product being processed
- StockIN, StockOUT: Quantity movements
- StockInHand: Previous stock balance

Detailed Breakdown

1. **Invoice Information**:
```
.Cells(ProductRow, 1).Value = InvoiceNo
.Cells(ProductRow, 2).Value = invoiceDate
.Cells(ProductRow, 3).Value = "SL"  'Sales
```
    - Records basic invoice identifiers
    - "SL" likely stands for "Sales"
2. **Customer Details**:
```
.Cells(ProductRow, 4).Value = wsInvoice.Range("C7").Value ' Customer_ID
.Cells(ProductRow, 5).Value = wsInvoice.Range("C8").Value ' Customer_Name
```
    - Copies customer information from fixed invoice cells
3. **Product Information**:
```
.Cells(ProductRow, 6).Value = ProductID
.Cells(ProductRow, 7).Value = wsInvoice.Cells(i, 3).Value ' Product_Name
.Cells(ProductRow, 8).Value = wsInvoice.Cells(i, 7).Value ' Product_Rate
.Cells(ProductRow, 9).Value = wsInvoice.Cells(i, 10).Value ' Product_Value
.Cells(ProductRow, 10).Value = wsInvoice.Cells(i, 9).Value ' GST_Rate
```
    - Records all product details from the invoice line
    - Includes pricing and tax rate information
4. **Temporal Information**:

```
.Cells(ProductRow, 11).Value = Format(invoiceDate, "mmm") ' Month
.Cells(ProductRow, 12).Value = finYear ' Financial Year
.Cells(ProductRow, 13).Value = "Pieces" ' Unit of measure
```

   - Stores reporting-friendly date information
   - Standardizes unit of measure as "Pieces"

5. **Stock Movement Tracking**:

```
.Cells(ProductRow, 14).Value = StockIN
.Cells(ProductRow, 15).Value = StockOUT
.Cells(ProductRow, 16).Value = StockInHand + StockIN - StockOUT
```

   - Records incoming and outgoing quantities
   - Calculates new stock balance by:
     1. Starting with previous balance (StockInHand)
     2. Adding any incoming stock (StockIN)
     3. Subtracting outgoing stock (StockOUT)

Key Features

1. **Complete Transaction Record**:
   - Captures all aspects of the product transaction
   - Links to original invoice and customer
2. **Inventory Management**:
   - Maintains accurate stock movement records
   - Calculates running stock balance automatically
3. **Standardized Structure**:
   - Consistent with headers created in GetProductSheet
   - Columns match exactly the header array from that function
4. **Reporting-Ready Data**:
   - Includes financial year and month for period analysis
   - Standard unit of measure for consistency
5. **Financial Tracking**:
   - Preserves pricing and tax information
   - Maintains product value data

Typical Flow

This would be called for each product line in an invoice, with:

- StockIN typically 0 for sales invoices
- StockOUT containing the sold quantity
- StockInHand coming from GetPreviousStockInHand

Column Mapping

The columns correspond to the headers created in GetProductSheet:

1. Invoice_No
2. Invoice_Date
3. Invoice_Type
4. Customer_ID
5. Customer_Name
6. Product_ID
7. Product_Name
8. Product_Rate
9. Product_Value
10. GST_Rate
11. Month
12. FAYear
13. Pieces
14. Stock_IN
15. Stock_OUT
16. Stock_In_Hand

This subroutine is a critical component of the inventory management system, ensuring each product's worksheet contains complete historical data for all transactions.

Explanation of the *GetPreviousStockInHand* Subroutine

```
Private Function GetPreviousStockInHand(wsProduct As Worksheet, ProductRow As Long) As Double
    If ProductRow > 2 Then
        GetPreviousStockInHand = wsProduct.Cells(ProductRow - 1, 16).Value
    Else
        GetPreviousStockInHand = 0
    End If
End Function
```

This is a VBA (Visual Basic for Applications) function that calculates the "previous stock in hand" for a product in an Excel worksheet. Let me break it down:

Function Purpose:

The function `GetPreviousStockInHand` retrieves the stock value from the previous row for a given product, or returns 0 if there is no previous row.

Parameters:

1. `wsProduct As Worksheet` - The worksheet object where the product data is stored
2. `ProductRow As Long` - The current row number of the product you're checking

Function Logic:

1. `If ProductRow > 2 Then` - Checks if the current row is greater than row 2
   - If true (meaning there is at least one row above the current one), it:
     - Gets the value from column 16 (column P) of the previous row (`ProductRow - 1`)
     - Assigns this value to the function's return value
2. `Else` - If the current row is 2 or less (meaning no previous product row exists)
   - Returns 0 as the function's value

Technical Details:

- The function returns a 🔢 (decimal number) data type
- Column 16 refers to column P in Excel (A=1, B=2,..., P=16)
- Row numbers start at 1 in Excel, but the data likely starts at row 2 (assuming row 1 has headers)

Typical Use Case:

This would likely be used in inventory management to:

1. Get the ending stock from the previous period
2. Use it as the starting stock for the current period
3. Handle the first entry (row 2) by assuming no previous stock (0)

For example, if called for row 3, it would return the value from P2. If called for row 2, it would return 0.

Explanation of the *SaveToSalesFile* Subroutine

```
Private Sub SaveToSalesFile(wsSalesFile As Worksheet, SalesRow As Long, wsInvoice As Worksheet, _
                            InvoiceNo As String, invoiceDate As Date, finYear As String)
    With wsSalesFile
        .Cells(SalesRow, 1).Value = wsInvoice.Range("C7").Value   ' Customer_ID
        .Cells(SalesRow, 2).Value = wsInvoice.Range("C8").Value   ' Customer_Name
        .Cells(SalesRow, 3).Value = wsInvoice.Range("I7").Value   ' State
        .Cells(SalesRow, 4).Value = InvoiceNo                     ' Invoice_No
        .Cells(SalesRow, 5).Value = invoiceDate                   ' Invoice_Date
        .Cells(SalesRow, 6).Value = "SL"                          ' Invoice_Type
        .Cells(SalesRow, 15).Value = wsInvoice.Range("J30").Value ' Total Invoice Amount

        ' Sales details
        .Cells(SalesRow, 7).Value = wsInvoice.Range("C25").Value   ' Total Quantity
        .Cells(SalesRow, 8).Value = wsInvoice.Range("J23").Value   ' Gross Sales
        .Cells(SalesRow, 9).Value = wsInvoice.Range("I26").Value   ' Disc Amount
        .Cells(SalesRow, 10).Value = wsInvoice.Range("J26").Value  ' Taxable Sales
        .Cells(SalesRow, 11).Value = wsInvoice.Range("J28").Value + wsInvoice.Range("J29").Value ' GST
Amount
        .Cells(SalesRow, 12).Value = wsInvoice.Range("G28").Value + wsInvoice.Range("G29").Value ' CGST
Amount
        .Cells(SalesRow, 13).Value = wsInvoice.Range("H28").Value + wsInvoice.Range("H29").Value ' SGST
Amount
        .Cells(SalesRow, 14).Value = wsInvoice.Range("I28").Value + wsInvoice.Range("I29").Value ' IGST
Amount
        .Cells(SalesRow, 15).Value = wsInvoice.Range("J30").Value  ' Invoice Amount

        ' Sales rep and transport details
        .Cells(SalesRow, 16).Value = wsInvoice.Range("A35").Value   ' SalesRep_ID
        .Cells(SalesRow, 17).Value = wsInvoice.Range("B35").Value   ' SalesRep_Name
```

```
        .Cells(SalesRow, 18).Value = wsInvoice.Range("C24").Value   ' CN No.
        .Cells(SalesRow, 19).Value = wsInvoice.Range("C26").Value   ' CN Date
        .Cells(SalesRow, 20).Value = wsInvoice.Range("C27").Value   ' Transporter Name
        .Cells(SalesRow, 21).Value = wsInvoice.Range("C28").Value   ' Freight

        ' Date-related fields
        .Cells(SalesRow, 22).Value = Format(invoiceDate, "mmm") ' Month
        .Cells(SalesRow, 24).Value = "Q" & (Int((Month(invoiceDate) - 1) / 3) + 1) ' Quarter
        .Cells(SalesRow, 25).Value = Year(invoiceDate) ' Year
        .Cells(SalesRow, 23).Value = IIf(Day(invoiceDate) <= 15, "1F", "2F") ' Fortnight
        .Cells(SalesRow, 26).Value = finYear ' Financial Year
    End With
End Sub
```

This subroutine saves summarized invoice data to a master sales tracking worksheet, consolidating all financial and logistical information for reporting and analysis purposes.

Purpose

To create a comprehensive sales record that:

1. Tracks all customer invoices
2. Maintains financial details (gross sales, discounts, taxes)
3. Records logistics information
4. Enables period-based reporting (monthly, quarterly, etc.)

Parameters

- `wsSalesFile`: Target sales summary worksheet
- `SalesRow`: Row number for the new record
- `wsInvoice`: Source invoice worksheet
- `InvoiceNo`: Unique invoice identifier
- `invoiceDate`: Date of invoice
- `finYear`: Financial year for reporting

Detailed Breakdown

1. Customer and Invoice Information

```
.Cells(SalesRow, 1).Value = wsInvoice.Range("C7").Value   ' Customer_ID
.Cells(SalesRow, 2).Value = wsInvoice.Range("C8").Value   ' Customer_Name
.Cells(SalesRow, 3).Value = wsInvoice.Range("I7").Value   ' State
.Cells(SalesRow, 4).Value = InvoiceNo                     ' Invoice_No
```

```
.Cells(SalesRow, 5).Value = invoiceDate                ' Invoice_Date
.Cells(SalesRow, 6).Value = "SL"                       ' Invoice_Type (Sales)
```

- Captures key identifiers linking the sale to a customer
- Marks all records as "SL" (Sales) type

2. Financial Information

```
' Core amounts
.Cells(SalesRow, 8).Value = wsInvoice.Range("J23").Value   ' Gross Sales
.Cells(SalesRow, 9).Value = wsInvoice.Range("I26").Value   ' Discount Amount
.Cells(SalesRow, 10).Value = wsInvoice.Range("J26").Value  ' Taxable Sales

' Tax breakdown
.Cells(SalesRow, 11).Value = wsInvoice.Range("J28").Value + wsInvoice.Range("J29").Value ' Total GST
.Cells(SalesRow, 12).Value = wsInvoice.Range("G28").Value + wsInvoice.Range("G29").Value ' CGST
.Cells(SalesRow, 13).Value = wsInvoice.Range("H28").Value + wsInvoice.Range("H29").Value ' SGST
.Cells(SalesRow, 14).Value = wsInvoice.Range("I28").Value + wsInvoice.Range("I29").Value ' IGST

' Final amounts
.Cells(SalesRow, 7).Value = wsInvoice.Range("C25").Value   ' Total Quantity
.Cells(SalesRow, 15).Value = wsInvoice.Range("J30").Value  ' Net Invoice Amount
```

- Maintains complete audit trail of financial calculations:
    - From gross sales → discounts → taxable amount → taxes → net amount
- Preserves tax component details (CGST/SGST for local, IGST for interstate)

3. Logistics and Operational Data

```
' Sales personnel
.Cells(SalesRow, 16).Value = wsInvoice.Range("A35").Value   ' SalesRep_ID
.Cells(SalesRow, 17).Value = wsInvoice.Range("B35").Value   ' SalesRep_Name

' Transportation details
.Cells(SalesRow, 18).Value = wsInvoice.Range("C24").Value   ' CN No. (Consignment Note)
.Cells(SalesRow, 19).Value = wsInvoice.Range("C26").Value   ' CN Date
.Cells(SalesRow, 20).Value = wsInvoice.Range("C27").Value   ' Transporter Name
.Cells(SalesRow, 21).Value = wsInvoice.Range("C28").Value   ' Freight Charges
```

- Tracks responsibility (sales rep)
- Records shipping information for logistics tracking

4. Reporting Period Fields

```
' Date analytics
.Cells(SalesRow, 22).Value = Format(invoiceDate, "mmm") ' Month (e.g., "Jan")
.Cells(SalesRow, 23).Value = IIf(Day(invoiceDate) <= 15, "1F", "2F") ' Fortnight
.Cells(SalesRow, 24).Value = "Q" & (Int((Month(invoiceDate) - 1) / 3) + 1) ' Quarter
.Cells(SalesRow, 25).Value = Year(invoiceDate) ' Calendar Year
.Cells(SalesRow, 26).Value = finYear           ' Financial Year
```

- Creates multiple time dimensions for reporting:
    - Month (text)
    - Fortnight (1F/2F)
    - Quarter (Q1-Q4)
    - Both calendar and financial year

Key Features

1. **Comprehensive Financial Tracking**:
    - Maintains complete sales calculation pipeline
    - Preserves all tax components separately
2. **Dual-Period Reporting**:
    - Supports both calendar year and financial year reporting
    - Multiple time granularities (month, quarter, fortnight)
3. **Operational Visibility**:
    - Links sales to responsible personnel
    - Tracks shipping details
4. **Data Integrity**:
    - All amounts pulled from calculated cells in invoice
    - No recalculation in this routine
5. **Structured Column Layout**:
    - Logical grouping of related fields
    - Consistent positioning across all records

Typical Usage

Called once per invoice (after all line items are processed) to create a summary record in the sales master file.

Why This Matters

- Creates the foundation for sales analysis and reporting
- Enables performance tracking by period, salesperson, region, etc.
- Provides data for financial reporting and tax compliance
- Maintains complete audit trail of all sales transactions

This subroutine serves as the central consolidation point for all sales data in what appears to be a robust inventory and accounting system.

Explanation of the *CreateBackup* Subroutine

```
Private Sub CreateBackup()
```

```
    Dim backupPath As String
    backupPath = ThisWorkbook.Path & "\Backups\"

    ' Create backups folder if it doesn't exist
    If Dir(backupPath, vbDirectory) = "" Then
        MkDir backupPath
    End If

    ' Save backup with timestamp
    ThisWorkbook.SaveCopyAs backupPath & "Backup_" & Format(Now(), "yyyymmdd_hhmmss") &
".xlsm"
End Sub
```

This subroutine creates a timestamped backup copy of the current workbook in a dedicated "Backups" folder.

Purpose

To automatically preserve a version of the workbook:

1. Before or after critical operations (like saving invoices)
2. With unique timestamp to prevent overwrites
3. In an organized backup location

Detailed Breakdown

1. Backup Path Setup

```
backupPath = ThisWorkbook.Path & "\Backups\"
```

- Constructs the backup folder path by:
  - Getting current workbook's directory (`ThisWorkbook.Path`)
  - Appending "Backups" subfolder
- Example result: **D:\Excel Project\Invoices\Backups**

2. Folder Creation

```
If Dir(backupPath, vbDirectory) = "" Then
    MkDir backupPath
End If
```

- Checks if backup folder exists using `Dir` function
- Creates the folder if missing using `MkDir`
- `vbDirectory` flag makes it check for folder existence

3. Saving Backup Copy

```
ThisWorkbook.SaveCopyAs backupPath & "Backup_" & Format(Now(), "yyyymmdd_hhmmss") & ".xlsm"
```

- Creates filename with components:
  - "Backup_" prefix
  - Current timestamp formatted as `YearMonthDay_HourMinuteSecond`
  - .xlsm extension for macro-enabled workbooks
- Example filename: `Backup_20231215_143045.xlsm`
- Uses `SaveCopyAs` which:
  - Saves without changing the active workbook
  - Doesn't break workbook references
  - Preserves all VBA code

Key Features

1. **Automatic Organization**:
   - Dedicated backup folder keeps files separated
   - No manual folder setup required
2. **Timestamp Protection**:
   - Precise to-the-second naming prevents overwrites
   - Sortable date format (yyyymmdd) for easy finding
3. **Non-Destructive**:
   - Original workbook remains active and unchanged
   - Works even if workbook is read-only
4. **Self-Healing**:
   - Creates missing folders automatically
   - No error if backups folder already exists

Typical Usage

Called after successful operations like invoice saving to:

- Create recovery points
- Maintain audit trail
- Enable rollback if needed

Why This Matters

- Critical for data integrity
- Provides disaster recovery

- Maintains historical versions
- Simple but effective version control

The timestamp format (yyyymmdd_hhmmss) is particularly useful because:

1. Files sort chronologically in Explorer
2. Eliminates ambiguity in backup sequencing
3. Works internationally (no month/day confusion)
4. Contains all needed temporal information

Explanation of the *GetFinancialYear* Function

```
' Helper function to calculate Financial Year
Function GetFinancialYear(invoiceDate As Date) As String
    If Month(invoiceDate) >= 4 Then
        GetFinancialYear = Year(invoiceDate) & "-" & Right(CStr(Year(invoiceDate) + 1), 2)
    Else
        GetFinancialYear = (Year(invoiceDate) - 1) & "-" & Right(CStr(Year(invoiceDate)), 2)
    End If
End Function
```

This function calculates the financial year (also called fiscal year) for a given date, following the April-March financial year convention used in many countries including India.

Purpose

To determine the correct financial year string (e.g., "2023-24") for any given date, based on the common April-to-March fiscal year system.

Parameters

- invoiceDate: The date for which we need to determine the financial year

Return Value

Returns a string representing the financial year in "YYYY-YY" format (e.g., "2023-24")

Detailed Breakdown

1. Financial Year Logic

The function uses this decision structure:

```
If Month(invoiceDate) >= 4 Then
    ' April-March period (current year to next year)
Else
    ' January-March period (previous year to current year)
End If
```

2. April-March Period (Month >= 4)

```
GetFinancialYear = Year(invoiceDate) & "-" & Right(CStr(Year(invoiceDate) + 1), 2)
```

- For dates April 1st or later:
    - Start year = current calendar year (e.g., 2023)
    - End year = next calendar year (e.g., 2024)
    - Formatted as "2023-24"
- Example: June 15, 2023 → "2023-24"

3. January-March Period (Month < 4)

```
GetFinancialYear = (Year(invoiceDate) - 1) & "-" & Right(CStr(Year(invoiceDate)), 2)
```

- For dates before April 1st:
    - Start year = previous calendar year (e.g., 2022)
    - End year = current calendar year (e.g., 2023)
    - Formatted as "2022-23"
- Example: February 10, 2023 → "2022-23"

4. String Formatting

- `Right(CStr(Year), 2)` extracts the last two digits of the year
- `CStr()` converts the year number to a string
- The hyphen "-" joins the two year parts

Key Features

1. **Common Fiscal Year Convention**:
    - Matches the April-March financial year used in:
        - India
        - United Kingdom
        - Canada
        - Japan
        - And many other countries

2. **Precise Year Calculation**:
    - Handles the year transition at April 1st
    - Correctly associates Jan-Mar dates with previous year
3. **Consistent Formatting**:
    - Always returns 7-character string (e.g., "2023-24")
    - Standard format for financial reporting
4. **Efficient Implementation**:
    - Simple date arithmetic
    - Minimal string operations

Example Outputs

- January 15, 2023 → "2022-23"
- April 5, 2023 → "2023-24"
- December 20, 2023 → "2023-24"
- March 31, 2024 → "2023-24"
- April 1, 2024 → "2024-25"

Why This Matters

- Essential for financial reporting
- Required for tax calculations
- Important for accounting periods
- Used in invoice numbering (as seen in earlier functions)
- Maintains consistency across financial documents

This function is particularly important in systems that need to organize data by financial year rather than calendar year, which is common in business and accounting applications.

When deciding whether to use a Subroutine (Sub) **or** Function **in VBA (or any programming language), follow these key criteria:**

1. Use ***a FUNCTION*** When:

- **You need to return a value**
  (e.g., calculations, validations, data lookups)

```
Function GetFinancialYear(date As Date) As String
    ' Returns "YYYY-YY"
```

```
End Function
```

- **You want to reuse logic**
  (e.g., tax calculations, stock updates)

```
Function CalculateGST(amount As Double, rate As Double) As Double
    CalculateGST = amount * rate / 100
End Function
```

- **The code is stateless**
  (Same input always gives same output, no side effects)

2. Use a ***SUBROUTINE*** (Sub) When:

- **You perform actions without returning values**
  (e.g., saving data, formatting sheets, UI updates)

```
Sub SaveToSalesFile()
    ' Writes data to a worksheet
End Sub
```

- **The code has side effects**
  (e.g., modifying worksheets, creating backups)

```
Sub CreateBackup()
    ' Saves a file to disk
End Sub
```

- **You handle events**
  (e.g., button clicks, workbook opens)

```
Sub btnSave_Click()
    ' Triggered by user action
End Sub
```

3. How to Divide Your Code

A. By Responsibility

Task Type	Approach
Data Processing	Functions (e.g., CalculateDiscount)
Data Saving	Subs (e.g., SaveToDatabase)
Validation	Functions (e.g., IsValidEmail)
UI Updates	Subs (e.g., FormatInvoiceTable)

B. By Reusability

- **Functions**: For logic used in multiple places
  (e.g., GetProductSheet is called by both saving and reporting code)
- **Subs**: For one-time operations
  (e.g., PostSaveOperations runs only after saving)

C. By Complexity

- Break down large procedures into smaller functions/subs:

```
Sub ProcessInvoice()
    If Not ValidateInvoice() Then Exit Sub  ' Function
```

```
    SaveInvoiceData()          ' Sub
    UpdateInventory()          ' Sub
    LogTransaction()           ' Sub
End Sub
```

4. Practical Examples from Your Code

Case	Why?
`GetFinancialYear` → **Function**	Returns a value based on input date
`SaveToBillsFile` → **Sub**	Writes data to sheets (no return value)
`InvoiceHasProducts` → **Function**	Returns `True/False` for validation
`CreateBackup` → **Sub**	Performs an action (saving a file)

5. Pro Tips

1. **Naming Conventions**:
    - Functions: Start with verbs (`Get`, `Calculate`, `Is`)
    - Subs: Use action verbs (`Save`, `Format`, `Run`)
2. **Parameter Passing**:
    - Use `ByVal` (default) for functions to avoid side effects.
    - Use `ByRef` in subs when modifying variables.
3. **Error Handling**:
    - Subs typically handle errors (e.g., `On Error GoTo`).
    - Functions often propagate errors to callers.
4. **State Management**:
    - Functions should **not** modify global state.
    - Subs **can** modify worksheets/variables.

When in Doubt, Ask:

1. "Do I need a result?" → **Function**
2. "Am I changing something?" → **Sub**
3. "Will I reuse this?" → **Function**

This approach keeps your code clean, debuggable, and maintainable. The examples from your invoice system demonstrate this perfectly!

✓ Instructions to Use the File:

1. Download the Excel file and the PDF file containing the full VBA code.
2. Save the Excel file as a **Macro-Enabled Workbook** (`.xlsm` format).
3. Open the Excel file and press **Alt + F11** to open the **Visual Basic for Applications (VBA) Editor**.
4. Copy the full code from the PDF and paste it into the VBA editor.
5. Save the file again.

6. Right-click on the **‘Save Invoice’** button and assign the macro named **“SaveInvoice”**.
7. Fill in the data on the **Invoice** sheet and click the **‘Save Invoice’** button to save it.

💬 If you face any issues, feel free to write in the **comment box** below the video!

Below is the Full Code:-

```
Option Explicit

Sub SaveInvoice()
    Dim wsinvoice As Worksheet, wsBillsFile As Worksheet, wsSalesFile As Worksheet
    Dim wsProduct As Worksheet
    Dim i As Long, BillRow As Long, SalesRow As Long, ProductRow As Long
    Dim InvoiceNo As String, lastInvoiceNo As String, newInvoiceNo As String
    Dim lastRow As Long, serialNumber As Integer
    Dim duplicateCheck As Range
    Dim tot_Qty As Double, amt_Disc As Double, Amt_Taxable As Double, Amt_GST As Double
    Dim custGSTIN As String, invoiceDate As Date, finYear As String, totalInvoiceAmount As Double
    Dim rng As Range, cell As Range
    Dim hasProducts As Boolean
    Dim backupPath As String
    Dim ProductID As String, StockIN As Double, StockOUT As Double, StockInHand As Double

    ' Error handling setup
    On Error GoTo ErrorHandler

    ' Initialize worksheet references with validation
    If Not InitializeWorksheets(wsinvoice, wsBillsFile, wsSalesFile) Then Exit Sub

    ' Validate invoice date
    invoiceDate = wsinvoice.Range("H5").Value
    If Not ValidateInvoiceDate(invoiceDate, wsinvoice) Then Exit Sub

    ' Check if products exist in invoice
    If Not InvoiceHasProducts(wsinvoice) Then
        MsgBox "Cannot save invoice with no products. Please add at least one product.", vbExclamation,
"No Products"
        Exit Sub
    End If

    ' Generate invoice number
    finYear = GetFinancialYear(invoiceDate)
    InvoiceNo = GenerateInvoiceNumber(wsBillsFile, finYear, invoiceDate)
    wsinvoice.Range("C5").Value = InvoiceNo
```

```
    ' Check for duplicate invoice number
    If InvoiceNumberExists(wsBillsFile, InvoiceNo) Then
        MsgBox "Invoice Number already exists in BillsFile!", vbExclamation, "Error"
        Exit Sub
    End If

    ' Find next empty rows in destination sheets
    BillRow = wsBillsFile.Cells(wsBillsFile.Rows.Count, 1).End(xlUp).Row + 1
    SalesRow = wsSalesFile.Cells(wsSalesFile.Rows.Count, 1).End(xlUp).Row + 1

    ' Process each product line in the invoice
    i = 13
    Do While wsinvoice.Cells(i, 2).Value <> "" Or wsinvoice.Cells(i, 3).Value <> ""
        If wsinvoice.Cells(i, 2).Value <> "" Or wsinvoice.Cells(i, 3).Value <> "" Then
            ' Save to BillsFile
            SaveToBillsFile wsBillsFile, BillRow, wsinvoice, i, InvoiceNo, invoiceDate, finYear,
totalInvoiceAmount

            ' Save to Product sheet if Product ID exists
            If wsinvoice.Cells(i, 2).Value <> "" Then
                ProductID = wsinvoice.Cells(i, 2).Value
                Set wsProduct = GetProductSheet(ProductID)
                ProductRow = wsProduct.Cells(wsProduct.Rows.Count, 1).End(xlUp).Row + 1

                ' Calculate stock values
                StockOUT = wsinvoice.Cells(i, 6).Value
                StockIN = 0
                StockInHand = GetPreviousStockInHand(wsProduct, ProductRow)

                ' Save to Product sheet
                SaveToProductSheet wsProduct, ProductRow, wsinvoice, i, InvoiceNo, invoiceDate, finYear,
ProductID, StockIN, StockOUT, StockInHand

                BillRow = BillRow + 1
            End If
        End If
        i = i + 1
    Loop

    ' Save summary to SalesFile
    SaveToSalesFile wsSalesFile, SalesRow, wsinvoice, InvoiceNo, invoiceDate, finYear

    ' Post-save operations
```

```
    PostSaveOperations wsinvoice

    ' Create backup
    CreateBackup
    MsgBox "backup file Save Successfully : ", vbOKOnly, "Back-Ups"

    MsgBox "Invoice saved successfully!", vbInformation, "Success"

    Exit Sub

ErrorHandler:
    MsgBox "Error " & Err.Number & ": " & Err.Description & vbCrLf & "Source: " & Err.Source, vbCritical,
"Error"
    Err.Clear
End Sub

' ==================== HELPER FUNCTIONS ====================

Private Function InitializeWorksheets(ByRef wsinvoice As Worksheet, ByRef wsBillsFile As Worksheet,
ByRef wsSalesFile As Worksheet) As Boolean
    On Error Resume Next
    Set wsinvoice = ThisWorkbook.Sheets("Invoice")
    Set wsBillsFile = ThisWorkbook.Sheets("BillsFile")
    Set wsSalesFile = ThisWorkbook.Sheets("SalesFile")
    On Error GoTo 0

    If wsBillsFile Is Nothing Or wsSalesFile Is Nothing Then
        MsgBox "Required worksheets (BillsFile or SalesFile) are missing!", vbCritical, "Error"
        InitializeWorksheets = False
    Else
        InitializeWorksheets = True
    End If
End Function

Private Function ValidateInvoiceDate(invoiceDate As Date, wsinvoice As Worksheet) As Boolean
    If invoiceDate < Date - 5 Or invoiceDate > Date + 5 Then
        MsgBox "Invalid Invoice Date! Please select a date within 5 days from today.", vbExclamation,
"Invalid Date"
        wsinvoice.Range("H5").Value = Date
        ValidateInvoiceDate = False
    Else
        ValidateInvoiceDate = True
    End If
End Function
```

```
Private Function InvoiceHasProducts(wsinvoice As Worksheet) As Boolean
    Dim i As Long
    i = 13
    Do While wsinvoice.Cells(i, 2).Value <> "" Or wsinvoice.Cells(i, 3).Value <> ""
        InvoiceHasProducts = True
        Exit Function
    Loop
    InvoiceHasProducts = False
End Function

Private Function GenerateInvoiceNumber(wsBillsFile As Worksheet, finYear As String, invoiceDate As
Date) As String
    Dim lastRow As Long, serialNumber As Integer
    Dim lastInvoiceNo As String
    Dim rng As Range, cell As Range

    lastRow = wsBillsFile.Cells(wsBillsFile.Rows.Count, 4).End(xlUp).Row
    lastInvoiceNo = wsBillsFile.Cells(lastRow, 4).Value
    serialNumber = 0

    If lastRow >= 2 Then
        Set rng = wsBillsFile.Range("D2:D" & lastRow)
        For Each cell In rng
            If Left(cell.Value, 7) = finYear Then
                lastInvoiceNo = cell.Value
            End If
        Next cell
        If lastInvoiceNo <> "" Then serialNumber = Val(Right(lastInvoiceNo, 4))
    End If

    serialNumber = serialNumber + 1
    GenerateInvoiceNumber = finYear & "/" & Format(serialNumber, "0000")
End Function

Private Function InvoiceNumberExists(wsBillsFile As Worksheet, InvoiceNo As String) As Boolean
    Dim duplicateCheck As Range
    Set duplicateCheck = wsBillsFile.Columns(4).Find(InvoiceNo, LookIn:=xlValues, LookAt:=xlWhole)
    InvoiceNumberExists = Not (duplicateCheck Is Nothing)
End Function

Private Sub SaveToBillsFile(wsBillsFile As Worksheet, BillRow As Long, wsinvoice As Worksheet, i As
Long, _
```

```
                    InvoiceNo As String, invoiceDate As Date, finYear As String, ByRef totalInvoiceAmount As
Double)
  Dim amt_Disc As Double, Amt_Taxable As Double, Amt_GST As Double
  Dim custGSTIN As String

  With wsBillsFile
    ' Invoice & Customer Details
    .Cells(BillRow, 1).Value = wsinvoice.Range("C7").Value   ' Customer_ID
    .Cells(BillRow, 2).Value = wsinvoice.Range("C8").Value   ' Customer_Name
    .Cells(BillRow, 3).Value = wsinvoice.Range("I7").Value   ' State
    .Cells(BillRow, 4).Value = InvoiceNo                  ' Invoice_No
    .Cells(BillRow, 5).Value = invoiceDate                ' Invoice_Date
    .Cells(BillRow, 6).Value = "SL"                       ' Invoice_Type

    ' Product Details
    .Cells(BillRow, 7).Value = wsinvoice.Cells(i, 2).Value   ' Product_ID
    .Cells(BillRow, 8).Value = wsinvoice.Cells(i, 3).Value   ' Product_Name
    .Cells(BillRow, 9).Value = wsinvoice.Cells(i, 6).Value   ' Quantity
    .Cells(BillRow, 10).Value = wsinvoice.Cells(i, 7).Value  ' Product_Rate
    .Cells(BillRow, 11).Value = wsinvoice.Cells(i, 10).Value ' Product_Value
    .Cells(BillRow, 12).Value = wsinvoice.Cells(i, 8).Value  ' Discount rate

    ' Discount Calculation
    amt_Disc = Round(wsinvoice.Cells(i, 10).Value * wsinvoice.Cells(i, 8).Value / 100, 0)
    .Cells(BillRow, 13).Value = amt_Disc   ' Discount Amount

    ' Taxable Amount Calculation
    Amt_Taxable = wsinvoice.Cells(i, 10).Value - amt_Disc
    .Cells(BillRow, 14).Value = Amt_Taxable  ' Taxable Amount

    .Cells(BillRow, 15).Value = wsinvoice.Cells(i, 9).Value  ' GST rate

    ' GST Calculation
    Amt_GST = Round(Amt_Taxable * wsinvoice.Cells(i, 9).Value / 100, 0)
    custGSTIN = wsinvoice.Range("I8").Value

    ' Check if the customer GSTIN indicates a West Bengal customer
    If Left(Trim(custGSTIN), 2) = "19" Then
      ' Intrastate: Apply CGST and SGST
      .Cells(BillRow, 16).Value = Amt_GST         ' Total GST Amount (optional)
      .Cells(BillRow, 17).Value = Amt_GST / 2     ' CGST Amount
      .Cells(BillRow, 18).Value = Amt_GST / 2     ' SGST Amount
      .Cells(BillRow, 19).Value = 0               ' IGST Amount (not applicable)
    Else
```

```
        ' Interstate: Apply IGST only
        .Cells(BillRow, 16).Value = 0              ' GST Amount (if you want to leave it blank)
        .Cells(BillRow, 17).Value = 0              ' CGST Amount
        .Cells(BillRow, 18).Value = 0              ' SGST Amount
        .Cells(BillRow, 19).Value = Amt_GST        ' IGST Amount
    End If

    ' Invoice Amount Calculation
    .Cells(BillRow, 20).Value = Amt_Taxable + Amt_GST ' Invoice Amount
    totalInvoiceAmount = totalInvoiceAmount + .Cells(BillRow, 20).Value

    ' Sales Rep & Transport Details
    .Cells(BillRow, 23).Value = wsinvoice.Range("C24").Value ' CN No
    .Cells(BillRow, 24).Value = wsinvoice.Range("C25").Value ' No of Item
    .Cells(BillRow, 25).Value = wsinvoice.Range("C26").Value ' Despatched on
    .Cells(BillRow, 26).Value = wsinvoice.Range("C27").Value ' Transporter name
    .Cells(BillRow, 27).Value = wsinvoice.Range("C28").Value ' Freight to Pay
    .Cells(BillRow, 21).Value = wsinvoice.Range("A35").Value ' SalesRep_ID
    .Cells(BillRow, 22).Value = wsinvoice.Range("B35").Value ' SalesRep_Name

    ' Date-related fields
    .Cells(BillRow, 28).Value = Format(invoiceDate, "mmm") ' Month
    .Cells(BillRow, 30).Value = "Q" & (Int((Month(invoiceDate) - 1) / 3) + 1) ' Quarter
    .Cells(BillRow, 31).Value = Year(invoiceDate) ' Year
    .Cells(BillRow, 29).Value = IIf(Day(invoiceDate) <= 15, "1F", "2F") ' Fortnight
    .Cells(BillRow, 32).Value = finYear ' Financial Year
  End With
End Sub

Private Function GetProductSheet(ProductID As String) As Worksheet
  On Error Resume Next
  Set GetProductSheet = ThisWorkbook.Sheets(ProductID)
  On Error GoTo 0

  If GetProductSheet Is Nothing Then
    Set GetProductSheet =
ThisWorkbook.Sheets.Add(After:=ThisWorkbook.Sheets(ThisWorkbook.Sheets.Count))
    GetProductSheet.Name = ProductID

    ' Create Product Sheet Headers
    With GetProductSheet
      .Range("A1:P1").Value = Array("Invoice_No", "Invoice_Date", "Invoice_Type", "Customer_ID",
"Customer_Name", _
                                    "Product_ID", "Product_Name", "Product_Rate", "Product_Value", "GST_Rate", _
```

```
                                "Month", "FAYear", "Pieces", "Stock_IN", "Stock_OUT", "Stock_In_Hand")
    End With
  End If
End Function

Private Function GetPreviousStockInHand(wsProduct As Worksheet, ProductRow As Long) As Double
  If ProductRow > 2 Then
    GetPreviousStockInHand = wsProduct.Cells(ProductRow - 1, 16).Value
  Else
    GetPreviousStockInHand = 0
  End If
End Function

Private Sub SaveToProductSheet(wsProduct As Worksheet, ProductRow As Long, wsinvoice As
Worksheet, i As Long, _
                    InvoiceNo As String, invoiceDate As Date, finYear As String, ProductID As String, _
                    StockIN As Double, StockOUT As Double, StockInHand As Double)
  With wsProduct
    .Cells(ProductRow, 1).Value = InvoiceNo
    .Cells(ProductRow, 2).Value = invoiceDate
    .Cells(ProductRow, 3).Value = "SL"
    .Cells(ProductRow, 4).Value = wsinvoice.Range("C7").Value ' Customer_ID
    .Cells(ProductRow, 5).Value = wsinvoice.Range("C8").Value ' Customer_Name
    .Cells(ProductRow, 6).Value = ProductID
    .Cells(ProductRow, 7).Value = wsinvoice.Cells(i, 3).Value ' Product_Name
    .Cells(ProductRow, 8).Value = wsinvoice.Cells(i, 7).Value ' Product_Rate
    .Cells(ProductRow, 9).Value = wsinvoice.Cells(i, 10).Value ' Product_Value
    .Cells(ProductRow, 10).Value = wsinvoice.Cells(i, 9).Value ' GST_Rate
    .Cells(ProductRow, 11).Value = Format(invoiceDate, "mmm") ' Month
    .Cells(ProductRow, 12).Value = finYear ' Financial Year
    .Cells(ProductRow, 13).Value = "Pieces"
    .Cells(ProductRow, 14).Value = StockIN
    .Cells(ProductRow, 15).Value = StockOUT
    .Cells(ProductRow, 16).Value = StockInHand + StockIN - StockOUT
  End With
End Sub

Private Sub SaveToSalesFile(wsSalesFile As Worksheet, SalesRow As Long, wsinvoice As Worksheet, _
                    InvoiceNo As String, invoiceDate As Date, finYear As String)
  With wsSalesFile
    .Cells(SalesRow, 1).Value = wsinvoice.Range("C7").Value   ' Customer_ID
    .Cells(SalesRow, 2).Value = wsinvoice.Range("C8").Value   ' Customer_Name
    .Cells(SalesRow, 3).Value = wsinvoice.Range("I7").Value   ' State
    .Cells(SalesRow, 4).Value = InvoiceNo                     ' Invoice_No
```

```
        .Cells(SalesRow, 5).Value = invoiceDate                    ' Invoice_Date
        .Cells(SalesRow, 6).Value = "SL"                           ' Invoice_Type
        .Cells(SalesRow, 15).Value = wsinvoice.Range("J30").Value ' Total Invoice Amount

        ' Sales details
        .Cells(SalesRow, 7).Value = wsinvoice.Range("C25").Value   ' Total Quantity
        .Cells(SalesRow, 8).Value = wsinvoice.Range("J23").Value   ' Gross Sales
        .Cells(SalesRow, 9).Value = wsinvoice.Range("I26").Value   ' Disc Amount
        .Cells(SalesRow, 10).Value = wsinvoice.Range("J26").Value  ' Taxable Sales
        .Cells(SalesRow, 11).Value = wsinvoice.Range("J28").Value + wsinvoice.Range("J29").Value ' GST
Amount
        .Cells(SalesRow, 12).Value = wsinvoice.Range("G28").Value + wsinvoice.Range("G29").Value ' CGST
Amount
        .Cells(SalesRow, 13).Value = wsinvoice.Range("H28").Value + wsinvoice.Range("H29").Value ' SGST
Amount
        .Cells(SalesRow, 14).Value = wsinvoice.Range("I28").Value + wsinvoice.Range("I29").Value ' IGST
Amount
        .Cells(SalesRow, 15).Value = wsinvoice.Range("J30").Value  ' Invoice Amount

        ' Sales rep and transport details
        .Cells(SalesRow, 16).Value = wsinvoice.Range("A35").Value   ' SalesRep_ID
        .Cells(SalesRow, 17).Value = wsinvoice.Range("B35").Value   ' SalesRep_Name
        .Cells(SalesRow, 18).Value = wsinvoice.Range("C24").Value   ' CN No.
        .Cells(SalesRow, 19).Value = wsinvoice.Range("C26").Value   ' CN Date
        .Cells(SalesRow, 20).Value = wsinvoice.Range("C27").Value   ' Transporter Name
        .Cells(SalesRow, 21).Value = wsinvoice.Range("C28").Value   ' Freight

        ' Date-related fields
        .Cells(SalesRow, 22).Value = Format(invoiceDate, "mmm") ' Month
        .Cells(SalesRow, 24).Value = "Q" & (Int((Month(invoiceDate) - 1) / 3) + 1) ' Quarter
        .Cells(SalesRow, 25).Value = Year(invoiceDate) ' Year
        .Cells(SalesRow, 23).Value = IIf(Day(invoiceDate) <= 15, "1F", "2F") ' Fortnight
        .Cells(SalesRow, 26).Value = finYear ' Financial Year
    End With
End Sub

Private Sub CreateBackup()
    Dim backupPath As String
    backupPath = ThisWorkbook.Path & "\Backups\"

    ' Create backups folder if it doesn't exist
    If Dir(backupPath, vbDirectory) = "" Then
        MkDir backupPath
    End If
```

```
    ' Save backup with timestamp
    ThisWorkbook.SaveCopyAs backupPath & "Backup_" & Format(Now(), "yyyymmdd_hhmmss") &
".xlsm"
End Sub


' Helper function to calculate Financial Year
Function GetFinancialYear(invoiceDate As Date) As String
    If Month(invoiceDate) >= 4 Then
        GetFinancialYear = Year(invoiceDate) & "-" & Right(CStr(Year(invoiceDate) + 1), 2)
        MsgBox "The Financial is : " & GetFinancialYear, vbOKOnly, "Hey..I am Gautam"
    Else
        GetFinancialYear = (Year(invoiceDate) - 1) & "-" & Right(CStr(Year(invoiceDate)), 2)
        MsgBox "The Financial is : " & GetFinancialYear, vbOKOnly, "Hey..I am Gautam"
    End If
End Function
```

Gautam Banerjee for Code Station
For Donation:
Goutam Banerjee, A/c No. 569902010003571,
IFSC Code- UBIN0556998,
Union Bank of India, Barasat
E-mail: gincom1@yahoo.com

Thanks.