Video Title:

👉 *Python Excel Automation – Complete End-to-End Project (One Script)*

🎬 INTRO (0:00 – 0:30)

Hello everyone,
Welcome back to my channel.

In today's video, we are going to build a **complete Excel Automation project using Python**.

This is **not a small demo**.
This is a **real office-level automation**, done using **just one Python script**.

We will read Excel data, clean it, calculate totals, add business logic, generate a summary, apply colors, create charts, and finally save a professional Excel report — **automatically**.

(pause)

So if you work with Excel in your job, or you are learning Python automation, this video will be extremely useful.

📂 SHOW EXCEL FILE (0:30 – 1:30)

First, let me show you the Excel file we are using.

The file name is **sales_data.xlsx**.

It contains simple columns like Date, Product, Quantity, and Price.

You will also notice that some rows have missing values.

This is very common in real projects, and we will see how Python automatically handles this problem.

(pause)

Now let's move to the Python code.

🧠 WHAT THIS SCRIPT DOES (1:30 – 2:30)

Before jumping into the code, let me explain what this script will do in simple words.

Think of Python as an office assistant.

It will:

- Open the Excel file
- Remove incorrect rows
- Calculate total sales
- Add High, Medium, or Low status
- Create a summary for managers
- Apply colors
- Create charts
- And save everything automatically

No manual Excel work at all.

📦 IMPORT LIBRARIES (2:30 – 3:30)

Now let's start with the code.

First, we import some libraries.

We use **pandas** to read and process Excel data.

Then we use **openpyxl** to apply formatting like colors and charts.

Pandas is great for data handling, but for Excel styling, openpyxl is required.

(pause)

📥 STEP 1 – READ EXCEL FILE (3:30 – 4:30)

In step one, we read the Excel file using pandas.

When we use pd.read_excel, Python loads the entire Excel sheet into a DataFrame.

You can think of a DataFrame as an Excel table inside Python.

Once the file is loaded, we can easily access columns like Quantity or Price.

🧹 STEP 2 – CLEAN DATA (4:30 – 5:30)

Next comes data cleaning.

In real life, Excel files often contain missing values.

Using dropna(), Python automatically removes rows that contain empty cells.

This prevents calculation errors and saves a lot of manual work.

🧮 STEP 3 – CALCULATE TOTAL AMOUNT (5:30 – 6:30)

Now we calculate total sales amount.

The formula is simple:
Quantity multiplied by Price.

Python creates a brand new column called **Total Amount**.

This is exactly the same thing people manually do in Excel, but here it is fully automated.

🏷 STEP 4 – ADD BUSINESS LOGIC (6:30 – 7:30)

Next, we add business logic.

Based on total amount, we categorize sales into:

- High
- Medium
- Low

We define a simple function, and then apply it row by row using pandas.

Python automatically assigns the correct status to each record.

📊 STEP 5 – CREATE SUMMARY REPORT (7:30 – 8:30)

After that, we generate a summary report.

This summary is very useful for managers.

It contains total revenue, number of orders, and how many sales are High, Medium, or Low.

This summary is created as a separate Excel sheet automatically.

💾 STEP 6 – WRITE TO EXCEL (8:30 – 9:30)

Now we write everything to a new Excel file called **Final_Excel_Report.xlsx**.

The first sheet contains cleaned sales data.

The second sheet contains the summary.

At this point, the Excel file is created, but it does not have colors or charts yet.

🎨 STEP 7 – CONDITIONAL FORMATTING (9:30 – 10:30)

In this step, we apply conditional formatting.

High sales are highlighted in green.

Medium sales are highlighted in yellow.

Low sales are highlighted in red.

Python loops through each row and applies colors automatically based on the status.

📈 STEP 8 – CREATE EXCEL CHARTS (10:30 – 12:00)

Now comes one of the most powerful parts — Excel charts.

We create a bar chart to show product-wise total sales.

We also create a pie chart to show sales distribution — High, Medium, and Low.

Both charts are added to a separate sheet called **Charts**.

These charts are created entirely by Python, without touching Excel.

✅ FINAL RESULT (12:00 – 13:00)

Finally, we save the Excel file.

Now let me open the output file.

You can see:

- Cleaned data
- Calculated columns
- Color-coded status
- A management summary
- And professional charts

All created using **one Python script**.

🚀 OUTRO (13:00 – 14:00)

This is how Python can completely replace manual Excel work.

If you liked this video, please like, share, and subscribe.

In the next videos, I'll cover more Python automation projects like:

- Advanced Excel automation
- Command-line tools
- Real-world office scripts

Thanks for watching, and I'll see you in the next video.

🎯 FINAL TIP FOR YOU

👉 Speak **slowly**
👉 Show Excel after each major step
👉 Pause between sections

If you want, I can also:

- Write **YouTube description + tags**
- Create **thumbnail text**
- Give **interview Q&A from this script**

"""
==
Python Excel Automation Script (Single Script – End to End)
==

WHAT THIS SCRIPT DOES:
---
1. Reads Excel data
2. Removes rows with missing values
3. Calculates total sales amount
4. Adds business status (High / Medium / Low)
5. Creates a summary sheet
6. Applies conditional formatting (colors)
7. Creates Excel charts (Bar + Pie)
8. Saves final professional Excel file

```
INPUT  FILE : sales_data.xlsx
OUTPUT FILE : Final_Excel_Report.xlsx
================================================================
"""

# ============================================================
# IMPORT REQUIRED LIBRARIES
# ============================================================

# pandas:
# Used to read Excel files and perform data operations
import pandas as pd

# load_workbook:
# Used to open an Excel file again for formatting & charts
from openpyxl import load_workbook

# PatternFill:
# Used to apply background colors to Excel cells
from openpyxl.styles import PatternFill

# Chart classes:
# Used to create Excel charts using Python
from openpyxl.chart import BarChart, PieChart, Reference


# ============================================================
# FILE CONFIGURATION
# ============================================================

# Name of the input Excel file
INPUT_FILE = "sales_data.xlsx"

# Name of the output Excel file
OUTPUT_FILE = "Final_Excel_Report.xlsx"


# ============================================================
# STEP 1: READ EXCEL FILE
# ============================================================

# Read Excel file and load it into a DataFrame
# DataFrame = Excel table inside Python
df = pd.read_excel(INPUT_FILE)

# Print confirmation message
```

```
print("📥 Excel file loaded successfully")

# At this point:
# df contains columns: Date, Product, Quantity, Price


# ============================================================
# STEP 2: DATA CLEANING
# ============================================================

# dropna() removes rows with missing (empty) values
# inplace=True means original DataFrame is modified
df.dropna(inplace=True)

# Print confirmation message
print("🧹 Missing rows removed")

# Result:
# Rows where Quantity or Price was empty are deleted


# ============================================================
# STEP 3: ADD CALCULATED COLUMN
# ============================================================

# Create a new column called "Total Amount"
# Formula: Quantity * Price
df["Total Amount"] = df["Quantity"] * df["Price"]

# Print confirmation message
print("🧮 Total Amount column created")

# Example:
# Quantity = 2, Price = 50000 → Total Amount = 100000


# ============================================================
# STEP 4: ADD STATUS COLUMN (BUSINESS LOGIC)
# ============================================================

def sales_status(amount):
    """
    This function decides the sales category
    based on Total Amount value.

    BUSINESS RULES:
    - amount >= 100000 → High
```

```
    - amount >= 50000  → Medium
    - amount < 50000   → Low
    """

    # If sales amount is very high
    if amount >= 100000:
        return "High"

    # If sales amount is medium
    elif amount >= 50000:
        return "Medium"

    # If sales amount is low
    else:
        return "Low"


# apply() runs the function on each row value
# It applies sales_status() to every Total Amount
df["Sales Status"] = df["Total Amount"].apply(sales_status)

# Print confirmation message
print("🏷 Sales Status column added")


# ============================================================
# STEP 5: CREATE SUMMARY REPORT
# ============================================================

# Create a new DataFrame for summary information
summary_df = pd.DataFrame({

    # First column: Description
    "Metric": [
        "Total Revenue",
        "Total Orders",
        "High Sales Orders",
        "Medium Sales Orders",
        "Low Sales Orders"
    ],

    # Second column: Calculated values
    "Value": [
        df["Total Amount"].sum(),              # Sum of all sales
        len(df),                               # Total number of records
        (df["Sales Status"] == "High").sum(),      # Count of High sales
        (df["Sales Status"] == "Medium").sum(),    # Count of Medium sales
```

```
    (df["Sales Status"] == "Low").sum()        # Count of Low sales
  ]
})

# Print confirmation message
print("📊 Summary report generated")


# ============================================================
# STEP 6: WRITE DATA TO EXCEL
# ============================================================

# ExcelWriter allows writing multiple sheets into one Excel file
with pd.ExcelWriter(OUTPUT_FILE, engine="openpyxl") as writer:

  # Write cleaned sales data to first sheet
  df.to_excel(
    writer,
    sheet_name="Sales Data",    # Sheet name
    index=False                 # Do not write index column
  )

  # Write summary data to second sheet
  summary_df.to_excel(
    writer,
    sheet_name="Summary",
    index=False
  )

# Print confirmation message
print("💾 Data written to Excel")


# ============================================================
# STEP 7: APPLY CONDITIONAL FORMATTING
# ============================================================

# Load the newly created Excel file
wb = load_workbook(OUTPUT_FILE)

# Select the "Sales Data" sheet
ws = wb["Sales Data"]

# ------------------------------------------------------------
# DEFINE CELL COLORS USING HEX CODES
# ------------------------------------------------------------
```

```
# Light green color for High sales
high_fill = PatternFill(
   start_color="C6EFCE",   # HEX code for green
   end_color="C6EFCE",     # Same color for full fill
   fill_type="solid"       # Solid background
)

# Light yellow color for Medium sales
medium_fill = PatternFill(
   start_color="FFEB9C",   # HEX code for yellow
   end_color="FFEB9C",
   fill_type="solid"
)

# Light red color for Low sales
low_fill = PatternFill(
   start_color="FFC7CE",   # HEX code for red
   end_color="FFC7CE",
   fill_type="solid"
)

# ------------------------------------------------------------
# APPLY COLORS ROW BY ROW
# ------------------------------------------------------------

# Start loop from row 2 (row 1 contains headers)
for row in range(2, ws.max_row + 1):

   # Access Sales Status cell (Column F)
   status_cell = ws[f"F{row}"]

   # Apply background color based on status value
   if status_cell.value == "High":
      status_cell.fill = high_fill

   elif status_cell.value == "Medium":
      status_cell.fill = medium_fill

   elif status_cell.value == "Low":
      status_cell.fill = low_fill

# Print confirmation message
print(" 🎨 Conditional formatting applied")


# ============================================================
# STEP 8: CREATE EXCEL CHARTS
```

```
# ============================================================

# Create a new sheet named "Charts"
chart_sheet = wb.create_sheet(title="Charts")


# ------------------------------------------------------------
# BAR CHART: PRODUCT vs TOTAL AMOUNT
# ------------------------------------------------------------

# Create bar chart object
bar_chart = BarChart()

# Set chart title
bar_chart.title = "Product Wise Total Sales"

# Set X and Y axis titles
bar_chart.x_axis.title = "Product"
bar_chart.y_axis.title = "Total Amount"

# Select Total Amount column (Column E)
data = Reference(
    ws,
    min_col=5,        # Column E
    min_row=1,        # Include header
    max_row=ws.max_row
)

# Select Product column (Column B)
categories = Reference(
    ws,
    min_col=2,        # Column B
    min_row=2,        # Skip header
    max_row=ws.max_row
)

# Add data and categories to chart
bar_chart.add_data(data, titles_from_data=True)
bar_chart.set_categories(categories)

# Add bar chart to Charts sheet
chart_sheet.add_chart(bar_chart, "A1")
# ------------------------------------------------------------
# PIE CHART: SALES STATUS DISTRIBUTION
# ------------------------------------------------------------

# Create pie chart object
```

```
pie_chart = PieChart()

# Set chart title
pie_chart.title = "Sales Status Distribution"

# Labels (High / Medium / Low)
labels = Reference(
    wb["Summary"],
    min_col=1,
    min_row=4,
    max_row=6
)

# Values for pie chart
data = Reference(
    wb["Summary"],
    min_col=2,
    min_row=4,
    max_row=6
)

# Add data and labels
pie_chart.add_data(data, titles_from_data=False)
pie_chart.set_categories(labels)

# Add pie chart to Charts sheet
chart_sheet.add_chart(pie_chart, "A20")


# ============================================================
# FINAL SAVE
# ============================================================

# Save the Excel file with all changes
wb.save(OUTPUT_FILE)

# Final confirmation messages
print("📊 Charts created successfully")
print("✅ Excel Automation Completed Successfully")
```