***Automated Business Solution with QR code Integration Python Code***

import tkinter as tk

from tkinter import ttk, filedialog, messagebox

import qrcode

from fpdf import FPDF

import gspread

from google.oauth2.service\_account import Credentials

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime

import pymysql

import requests

# Global Variables for Google Sheets Setup

SERVICE\_ACCOUNT\_FILE = None # Initially unset

SPREADSHEET\_ID = None # Initially unset

credentials = None

client = None

sheet = None

root = None

confidential\_window = None

db\_connection = None

db\_cursor = None

PASSWORD = "1234"

# Function to authorize Google Sheets with the provided service account JSON file

def authorize\_google\_sheet():

global credentials, client

if not SERVICE\_ACCOUNT\_FILE:

raise Exception("Service account file is not set. Please provide the JSON file first.")

credentials = Credentials.from\_service\_account\_file(SERVICE\_ACCOUNT\_FILE, scopes=["https://www.googleapis.com/auth/spreadsheets"])

client = gspread.authorize(credentials)

# Function to set the Sheet ID manually

def set\_sheet\_id():

global SPREADSHEET\_ID, sheet

SPREADSHEET\_ID = sheet\_id\_input.get()

if not SPREADSHEET\_ID:

messagebox.showerror("Error", "Sheet ID is required.")

return

try:

if not client:

authorize\_google\_sheet()

sheet = client.open\_by\_key(SPREADSHEET\_ID).sheet1

messagebox.showinfo("Success", f"Successfully connected to the sheet: {SPREADSHEET\_ID}")

except Exception as e:

messagebox.showerror("Error", f"Failed to connect to the sheet: {e}")

# Function to set the Service Account JSON file manually

def set\_service\_account\_file():

global SERVICE\_ACCOUNT\_FILE, credentials, client

SERVICE\_ACCOUNT\_FILE = filedialog.askopenfilename(filetypes=[("JSON Files", "\*.json")])

if not SERVICE\_ACCOUNT\_FILE:

messagebox.showerror("Error", "Service account JSON file is required.")

return

try:

authorize\_google\_sheet()

messagebox.showinfo("Success", "Service account file updated and reauthorized successfully.")

except Exception as e:

messagebox.showerror("Error", f"Failed to authorize with the selected file: {e}")

# Keep your original functionality unchanged

def customer\_discount\_analysis():

try:

data = sheet.get\_all\_records()

df = pd.DataFrame(data)

if df.empty:

messagebox.showerror("Error", "No data available for analysis.")

return

total\_expenses = df.groupby("Customer Name")["Expense"].sum()

top\_customers = total\_expenses.nlargest(5)

top\_customers.plot(kind="bar", title="Top 5 Customers by Total Expense")

plt.xlabel("Customer Name")

plt.ylabel("Total Expense")

plt.show()

except Exception as e:

messagebox.showerror("Error", f"Failed to perform analysis: {e}")

# Keep other existing functionality intact...#sql using

# Password to access the Confidential Options

# Function to connect to MySQL database

# Password to access the Confidential Options

GOOGLE\_SHEET\_URL = "SPREADSHEET\_ID" # Replace with your Google Sheets URL

# Global variables for managing toggle system and MySQL connection

root = None

confidential\_window = None

db\_connection = None

db\_cursor = None

sheet = None # Google Sheets instance

# Function to connect to MySQL database

def connect\_to\_mysql():

global db\_connection, db\_cursor

try:

db\_connection = pymysql.connect(

host="localhost",

user="root",

password="135661",

database="base"

)

db\_cursor = db\_connection.cursor()

db\_cursor.execute("""

CREATE TABLE IF NOT EXISTS customers (

customer\_id INT,

customer\_name VARCHAR(255),

product\_name VARCHAR(255),

expense FLOAT,

quantity\_sold INT

)

""")

db\_connection.commit()

messagebox.showinfo("Success", "Connected to MySQL database!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to connect to MySQL: {e}")

# Function to connect to Google Sheets

def connect\_to\_google\_sheets():

global sheet

try:

scope = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com/auth/drive']

credentials = ServiceAccountCredentials.from\_json\_keyfile\_name('path\_to\_service\_account.json', scope) # Replace with your JSON key file path

client = gspread.authorize(credentials)

sheet = client.open\_by\_url(GOOGLE\_SHEET\_URL).sheet1

messagebox.showinfo("Success", "Connected to Google Sheets!")

except Exception as e:

messagebox.showerror("Error", f"Failed to connect to Google Sheets: {e}")

# Function to save Google Sheets data to MySQL

import pymysql

from tkinter import messagebox

def save\_to\_mysql():

try:

if not sheet:

messagebox.showerror("Error", "Google Sheets is not connected!")

return

# Fetch data from Google Sheets

data = sheet.get\_all\_records()

if not data:

messagebox.showinfo("No Data", "No data available in the Google Sheet to save.")

return

# Save data to MySQL database

for record in data:

customer\_id = record.get("Customer ID")

customer\_name = record.get("Customer Name")

product\_name = record.get("Product Name")

expense = record.get("Expense")

quantity\_sold = record.get("Quantity Sold")

# Check if the record already exists in the database

db\_cursor.execute("""

SELECT quantity\_sold FROM customers

WHERE customer\_id = %s AND product\_name = %s

""", (customer\_id, product\_name))

result = db\_cursor.fetchone()

if result:

# If record exists, update the quantity\_sold

existing\_quantity = result[0]

new\_quantity = existing\_quantity + quantity\_sold

db\_cursor.execute("""

UPDATE customers

SET quantity\_sold = %s

WHERE customer\_id = %s AND product\_name = %s

""", (new\_quantity, customer\_id, product\_name))

else:

# If no matching record exists, insert a new row with ON DUPLICATE KEY UPDATE

db\_cursor.execute("""

INSERT INTO customers (customer\_id, customer\_name, product\_name, expense, quantity\_sold)

VALUES (%s, %s, %s, %s, %s)

ON DUPLICATE KEY UPDATE quantity\_sold = quantity\_sold + VALUES(quantity\_sold)

""", (customer\_id, customer\_name, product\_name, expense, quantity\_sold))

# Commit the transaction

db\_connection.commit()

messagebox.showinfo("Success", "Data saved to MySQL database successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to save data to MySQL: {e}")

# Function to search customer data from MySQL

def search\_from\_mysql():

customer\_id = customer\_id\_sql\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required to search.")

return

try:

# Fetch all records with the given customer\_id

db\_cursor.execute("SELECT \* FROM customers WHERE customer\_id = %s", (customer\_id,))

records = db\_cursor.fetchall() # Fetch all matching records

if records:

# Prepare the message with all customer details

customer\_details = "\n".join([f"ID: {record[0]}\nName: {record[1]}\nProduct: {record[2]}\n"

f"Expense: {record[3]}\nQuantity Sold: {record[4]}\n"

f"{'-'\*40}" for record in records])

messagebox.showinfo("Customer Found", customer\_details)

else:

messagebox.showinfo("Not Found", "No customer found with the given ID.")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to search customer: {e}")

def update\_mysql():

try:

customer\_id = customer\_id\_sql\_input.get()

new\_expense = float(new\_expense\_input.get())

if not customer\_id or not new\_expense:

messagebox.showerror("Error", "Customer ID and new expense value are required.")

return

db\_cursor.execute("UPDATE customers SET expense = %s WHERE customer\_id = %s", (new\_expense, customer\_id))

db\_connection.commit()

messagebox.showinfo("Success", "Customer expense updated successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to update customer: {e}")

# Function to delete customer data from MySQL

def delete\_from\_mysql():

customer\_id = customer\_id\_sql\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required to delete.")

return

try:

db\_cursor.execute("DELETE FROM customers WHERE customer\_id = %s", (customer\_id,))

db\_connection.commit()

messagebox.showinfo("Success", "Customer data deleted successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to delete customer: {e}")

# Function to open the Confidential window

def open\_confidential\_window():

global confidential\_window

if confidential\_window is None or not tk.Toplevel.winfo\_exists(confidential\_window):

confidential\_window = tk.Toplevel(root)

confidential\_window.title("Confidential Options")

confidential\_window.geometry("500x400")

# Save to MySQL button

ttk.Button(confidential\_window, text="Save Data to MySQL", command=save\_to\_mysql).pack(pady=10)

# Search from MySQL

search\_frame = ttk.LabelFrame(confidential\_window, text="Search from MySQL", padding=10)

search\_frame.pack(fill="x", padx=10, pady=5)

global customer\_id\_sql\_input

customer\_id\_sql\_input = tk.StringVar()

ttk.Label(search\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(search\_frame, textvariable=customer\_id\_sql\_input, width=30).grid(row=0, column=1, padx=5, pady=5)

ttk.Button(search\_frame, text="Search", command=search\_from\_mysql).grid(row=0, column=2, padx=5, pady=5)

# Update MySQL

update\_frame = ttk.LabelFrame(confidential\_window, text="Update MySQL", padding=10)

update\_frame.pack(fill="x", padx=10, pady=5)

global new\_expense\_input

new\_expense\_input = tk.StringVar()

ttk.Label(update\_frame, text="New Expense:").grid(row=1, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(update\_frame, textvariable=new\_expense\_input, width=30).grid(row=1, column=1, padx=5, pady=5)

ttk.Button(update\_frame, text="Update", command=update\_mysql).grid(row=1, column=2, padx=5, pady=5)

# Delete from MySQL

delete\_frame = ttk.LabelFrame(confidential\_window, text="Delete from MySQL", padding=10)

delete\_frame.pack(fill="x", padx=10, pady=5)

ttk.Button(delete\_frame, text="Delete", command=delete\_from\_mysql).pack(pady=10)

# Back Button

ttk.Button(confidential\_window, text="Exit", command=lambda: confidential\_window.destroy()).pack(pady=20)

# Function to prompt for password and proceed

def password\_prompt():

password\_window = tk.Toplevel(root)

password\_window.title("Enter Password")

password\_window.geometry("300x150")

ttk.Label(password\_window, text="Enter Password:", font=("Arial", 12)).pack(pady=10)

password\_input = tk.StringVar()

ttk.Entry(password\_window, textvariable=password\_input, show="\*", width=20).pack(pady=5)

def check\_password():

if password\_input.get() == PASSWORD:

password\_window.destroy()

open\_confidential\_window()

else:

messagebox.showerror("Access Denied", "Incorrect Password")

ttk.Button(password\_window, text="Submit", command=check\_password).pack(pady=10)

# Connect to MySQL and Google Sh

# Add a new frame for Google Sheets connection

# Frame for Google Sheets connection

# Function to reset and allow connecting to a new sheet dynamically

def reset\_google\_connector():

global SERVICE\_ACCOUNT\_FILE, SPREADSHEET\_ID, credentials, client, sheet

SERVICE\_ACCOUNT\_FILE = None

SPREADSHEET\_ID = None

credentials = None

client = None

sheet = None

sheet\_id\_input.set("") # Clear the Sheet ID input

messagebox.showinfo("Reset", "Google Connector has been reset. Please provide new details.")

# Keep your original GUI functionality

# ...

# QR Code Generation Function

def generate\_qr(data, file\_name):

qr = qrcode.QRCode(version=1, box\_size=10, border=5)

qr.add\_data(data)

qr.make(fit=True)

img = qr.make\_image(fill="black", back\_color="white")

img.save(file\_name)

return file\_name

# PDF Generation Function

def generate\_pdf(data, file\_name):

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, txt="Customer Report", ln=True, align="C")

# Customer details

for key, value in data["customer"].items():

pdf.cell(200, 10, txt=f"{key}: {value}", ln=True)

pdf.ln(10) # Add some space before listing expenses

# Product expenses

pdf.cell(200, 10, txt="Individual Product Expenses:", ln=True)

for product in data["products"]:

pdf.cell(200, 10, txt=f"Product: {product['name']} - Expense: {product['expense']}", ln=True)

pdf.ln(10) # Add space

pdf.cell(200, 10, txt=f"Total Expense: {data['total\_expense']}", ln=True)

pdf.output(file\_name)

return file\_name

# GUI Functions

def product\_estimation\_analysis():

try:

data = sheet.get\_all\_records()

df = pd.DataFrame(data)

if df.empty:

messagebox.showerror("Error", "No data available for analysis.")

return

sales\_trend = df.groupby("Product Name")["Quantity Sold"].sum()

sales\_trend.plot(kind="line", title="Sales Trend by Product")

plt.xlabel("Product Name")

plt.ylabel("Quantity Sold")

plt.show()

except Exception as e:

messagebox.showerror("Error", f"Failed to perform analysis: {e}")

def search\_customer():

customer\_id = customer\_id\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required.")

return

try:

data = sheet.get\_all\_records()

customer\_data = next(

(customer for customer in data if str(customer["Customer ID"]) == str(customer\_id)), None

)

if customer\_data is None:

messagebox.showinfo("Not Found", "Customer not found.")

return

# Gather individual expenses for the customer

customer\_expenses = [customer for customer in data if str(customer["Customer ID"]) == str(customer\_id)]

total\_expense = sum(exp["Expense"] for exp in customer\_expenses)

# Prepare data for displaying and generating the PDF

customer\_info = {

"Customer ID": customer\_data["Customer ID"],

"Customer Name": customer\_data["Customer Name"],

"Total Expense": total\_expense

}

products = [{"name": exp["Product Name"], "expense": exp["Expense"]} for exp in customer\_expenses]

# Show the customer data in a new window

show\_customer\_data\_window(customer\_info, products, total\_expense)

# Store the customer info and products for PDF generation

global selected\_customer\_data

selected\_customer\_data = {"customer": customer\_info, "products": products, "total\_expense": total\_expense}

except Exception as e:

messagebox.showerror("Error", f"Failed to search customer: {e}")

def show\_customer\_data\_window(customer\_info, products, total\_expense):

# New window to display customer info and product expenses

show\_window = tk.Toplevel(root)

show\_window.title(f"Customer: {customer\_info['Customer Name']}")

customer\_label = ttk.Label(show\_window, text=f"Customer ID: {customer\_info['Customer ID']}\n"

f"Customer Name: {customer\_info['Customer Name']}\n"

f"Total Expense: {total\_expense}")

customer\_label.pack(pady=10)

ttk.Label(show\_window, text="Individual Expenses:").pack()

for product in products:

ttk.Label(show\_window, text=f"Product: {product['name']} - Expense: {product['expense']}").pack()

ttk.Button(show\_window, text="Report PDF", command=generate\_customer\_pdf).pack(pady=10)

def generate\_customer\_pdf():

global selected\_customer\_data

if selected\_customer\_data is None:

messagebox.showerror("Error", "No customer selected for PDF generation.")

return

file\_name = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF Files", "\*.pdf")])

if file\_name:

generate\_pdf(selected\_customer\_data, file\_name)

messagebox.showinfo("Success", f"Customer report saved at {file\_name}")

def show\_customers():

try:

data = sheet.get\_all\_records()

if not data:

messagebox.showinfo("No Data", "No customers available.")

return

show\_window = tk.Toplevel(root)

show\_window.title("All Customers")

columns = list(data[0].keys())

tree = ttk.Treeview(show\_window, columns=columns, show="headings")

for col in columns:

tree.heading(col, text=col)

tree.column(col, width=120)

for row in data:

tree.insert("", tk.END, values=list(row.values()))

tree.pack(fill="both", expand=True)

except Exception as e:

messagebox.showerror("Error", f"Failed to fetch customers: {e}")

def generate\_qr\_and\_pdf():

# Get inputs

product\_name = product\_name\_input.get()

price = product\_price\_input.get()

expire\_date = product\_expiry\_input.get()

webapp\_url = webapp\_input.get()

if product\_name and price and expire\_date and webapp\_url:

# Generate a dynamic URL for the web app

qr\_url = f"{webapp\_url}?product\_name={product\_name}&expense={price}"

# Generate QR Code

file\_name\_qr = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "\*.png")])

if file\_name\_qr:

qr = qrcode.make(qr\_url)

qr.save(file\_name\_qr)

messagebox.showinfo("QR Code", f"QR Code saved at {file\_name\_qr}")

# Generate PDF

data\_pdf = {"Product Name": product\_name, "Price": price, "Expire Date": expire\_date}

file\_name\_pdf = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF Files", "\*.pdf")])

if file\_name\_pdf:

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, txt="Product Information", ln=True, align="C")

for key, value in data\_pdf.items():

pdf.cell(200, 10, txt=f"{key}: {value}", ln=True, align="L")

pdf.output(file\_name\_pdf)

messagebox.showinfo("PDF", f"PDF saved at {file\_name\_pdf}")

# Notify user

messagebox.showinfo("QR Code Generated", f"Scan the QR code to open the web app and enter ID/Name.")

else:

messagebox.showerror("Error", "All fields are required.")

# Main Application

root = tk.Tk()

root.title("Automated Bussiness Solution With Qr Code Intregation")

# Frame for Working Mode

working\_frame = ttk.LabelFrame(root, text="Working Mode", padding=10)

working\_frame.pack(fill="x", padx=10, pady=5)

# VIP Mode

vip\_frame = ttk.LabelFrame(root, text="VIP Mode - Product QR & PDF", padding=10)

vip\_frame.pack(fill="x", padx=10, pady=5)

product\_name\_input = tk.StringVar()

product\_price\_input = tk.StringVar()

product\_expiry\_input = tk.StringVar()

webapp\_input=tk.StringVar()

ttk.Label(vip\_frame, text="Product Name:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_name\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Price:").grid(row=1, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_price\_input, width=40).grid(row=1, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Expire Date:").grid(row=2, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_expiry\_input, width=40).grid(row=2, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Web App Connect:").grid(row=3, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=webapp\_input, width=40).grid(row=3, column=1, padx=5, pady=5)

ttk.Button(vip\_frame, text="Generate QR Code & PDF", command=generate\_qr\_and\_pdf).grid(row=4, column=0, columnspan=2, padx=5, pady=10)

# Frame for Business Analysis

business\_frame = ttk.LabelFrame(root, text="Business Analysis", padding=10)

business\_frame.pack(fill="x", padx=10, pady=5)

ttk.Button(business\_frame, text="Customer Discount Analysis", command=customer\_discount\_analysis).pack(side="left", padx=10)

ttk.Button(business\_frame, text="Product Estimation Analysis", command=product\_estimation\_analysis).pack(side="left", padx=10)

# Search Customer

search\_frame = ttk.LabelFrame(root, text="Search Customer", padding=10)

search\_frame.pack(fill="x", padx=10, pady=5)

customer\_id\_input = tk.StringVar()

ttk.Label(search\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(search\_frame, textvariable=customer\_id\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

# Separate buttons for Search and PDF Generation

ttk.Button(search\_frame, text="Search Customer", command=search\_customer).grid(row=1, column=0, padx=5, pady=10)

# Show Customers Button

ttk.Button(root, text="Show All Customers", command=show\_customers).pack(padx=10, pady=10)

ttk.Button(root, text="Confidential", command=password\_prompt).pack(padx=2,pady=2)

# Google Sheets Connector Frame

sheet\_connector\_frame = ttk.LabelFrame(root, text="Google Sheet Connector", padding=10)

sheet\_connector\_frame.pack(fill="x", padx=10, pady=5)

# Sheet ID input and buttons

sheet\_id\_input = tk.StringVar()

ttk.Label(sheet\_connector\_frame, text="Sheet ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(sheet\_connector\_frame, textvariable=sheet\_id\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

ttk.Button(sheet\_connector\_frame, text="Set Sheet ID", command=set\_sheet\_id).grid(row=0, column=2, padx=5, pady=5)

# Service Account File input and reset button

ttk.Button(sheet\_connector\_frame, text="Set Service Account File", command=set\_service\_account\_file).grid(row=1, column=0, columnspan=2, pady=10)

ttk.Button(sheet\_connector\_frame, text="Reset Connector", command=reset\_google\_connector).grid(row=1, column=2, pady=10)

# Running the Tkinter loop

#second winow funtionality

# Main application function

connect\_to\_mysql()

connect\_to\_google\_sheets()

# Run the main application

root.mainloop()

***Automated Business Solution with QR code Integration Python Code***

import tkinter as tk

from tkinter import ttk, filedialog, messagebox

import qrcode

from fpdf import FPDF

import gspread

from google.oauth2.service\_account import Credentials

import pandas as pd

import matplotlib.pyplot as plt

from datetime import datetime

import pymysql

import requests

# Global Variables for Google Sheets Setup

SERVICE\_ACCOUNT\_FILE = None # Initially unset

SPREADSHEET\_ID = None # Initially unset

credentials = None

client = None

sheet = None

root = None

confidential\_window = None

db\_connection = None

db\_cursor = None

PASSWORD = "1234"

# Function to authorize Google Sheets with the provided service account JSON file

def authorize\_google\_sheet():

global credentials, client

if not SERVICE\_ACCOUNT\_FILE:

raise Exception("Service account file is not set. Please provide the JSON file first.")

credentials = Credentials.from\_service\_account\_file(SERVICE\_ACCOUNT\_FILE, scopes=["https://www.googleapis.com/auth/spreadsheets"])

client = gspread.authorize(credentials)

# Function to set the Sheet ID manually

def set\_sheet\_id():

global SPREADSHEET\_ID, sheet

SPREADSHEET\_ID = sheet\_id\_input.get()

if not SPREADSHEET\_ID:

messagebox.showerror("Error", "Sheet ID is required.")

return

try:

if not client:

authorize\_google\_sheet()

sheet = client.open\_by\_key(SPREADSHEET\_ID).sheet1

messagebox.showinfo("Success", f"Successfully connected to the sheet: {SPREADSHEET\_ID}")

except Exception as e:

messagebox.showerror("Error", f"Failed to connect to the sheet: {e}")

# Function to set the Service Account JSON file manually

def set\_service\_account\_file():

global SERVICE\_ACCOUNT\_FILE, credentials, client

SERVICE\_ACCOUNT\_FILE = filedialog.askopenfilename(filetypes=[("JSON Files", "\*.json")])

if not SERVICE\_ACCOUNT\_FILE:

messagebox.showerror("Error", "Service account JSON file is required.")

return

try:

authorize\_google\_sheet()

messagebox.showinfo("Success", "Service account file updated and reauthorized successfully.")

except Exception as e:

messagebox.showerror("Error", f"Failed to authorize with the selected file: {e}")

# Keep your original functionality unchanged

def customer\_discount\_analysis():

try:

data = sheet.get\_all\_records()

df = pd.DataFrame(data)

if df.empty:

messagebox.showerror("Error", "No data available for analysis.")

return

total\_expenses = df.groupby("Customer Name")["Expense"].sum()

top\_customers = total\_expenses.nlargest(5)

top\_customers.plot(kind="bar", title="Top 5 Customers by Total Expense")

plt.xlabel("Customer Name")

plt.ylabel("Total Expense")

plt.show()

except Exception as e:

messagebox.showerror("Error", f"Failed to perform analysis: {e}")

# Keep other existing functionality intact...#sql using

# Password to access the Confidential Options

# Function to connect to MySQL database

# Password to access the Confidential Options

GOOGLE\_SHEET\_URL = "SPREADSHEET\_ID" # Replace with your Google Sheets URL

# Global variables for managing toggle system and MySQL connection

root = None

confidential\_window = None

db\_connection = None

db\_cursor = None

sheet = None # Google Sheets instance

# Function to connect to MySQL database

def connect\_to\_mysql():

global db\_connection, db\_cursor

try:

db\_connection = pymysql.connect(

host="localhost",

user="root",

password="135661",

database="base"

)

db\_cursor = db\_connection.cursor()

db\_cursor.execute("""

CREATE TABLE IF NOT EXISTS customers (

customer\_id INT,

customer\_name VARCHAR(255),

product\_name VARCHAR(255),

expense FLOAT,

quantity\_sold INT

)

""")

db\_connection.commit()

messagebox.showinfo("Success", "Connected to MySQL database!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to connect to MySQL: {e}")

# Function to connect to Google Sheets

def connect\_to\_google\_sheets():

global sheet

try:

scope = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com/auth/drive']

credentials = ServiceAccountCredentials.from\_json\_keyfile\_name('path\_to\_service\_account.json', scope) # Replace with your JSON key file path

client = gspread.authorize(credentials)

sheet = client.open\_by\_url(GOOGLE\_SHEET\_URL).sheet1

messagebox.showinfo("Success", "Connected to Google Sheets!")

except Exception as e:

messagebox.showerror("Error", f"Failed to connect to Google Sheets: {e}")

# Function to save Google Sheets data to MySQL

import pymysql

from tkinter import messagebox

def save\_to\_mysql():

try:

if not sheet:

messagebox.showerror("Error", "Google Sheets is not connected!")

return

# Fetch data from Google Sheets

data = sheet.get\_all\_records()

if not data:

messagebox.showinfo("No Data", "No data available in the Google Sheet to save.")

return

# Save data to MySQL database

for record in data:

customer\_id = record.get("Customer ID")

customer\_name = record.get("Customer Name")

product\_name = record.get("Product Name")

expense = record.get("Expense")

quantity\_sold = record.get("Quantity Sold")

# Check if the record already exists in the database

db\_cursor.execute("""

SELECT quantity\_sold FROM customers

WHERE customer\_id = %s AND product\_name = %s

""", (customer\_id, product\_name))

result = db\_cursor.fetchone()

if result:

# If record exists, update the quantity\_sold

existing\_quantity = result[0]

new\_quantity = existing\_quantity + quantity\_sold

db\_cursor.execute("""

UPDATE customers

SET quantity\_sold = %s

WHERE customer\_id = %s AND product\_name = %s

""", (new\_quantity, customer\_id, product\_name))

else:

# If no matching record exists, insert a new row with ON DUPLICATE KEY UPDATE

db\_cursor.execute("""

INSERT INTO customers (customer\_id, customer\_name, product\_name, expense, quantity\_sold)

VALUES (%s, %s, %s, %s, %s)

ON DUPLICATE KEY UPDATE quantity\_sold = quantity\_sold + VALUES(quantity\_sold)

""", (customer\_id, customer\_name, product\_name, expense, quantity\_sold))

# Commit the transaction

db\_connection.commit()

messagebox.showinfo("Success", "Data saved to MySQL database successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to save data to MySQL: {e}")

# Function to search customer data from MySQL

def search\_from\_mysql():

customer\_id = customer\_id\_sql\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required to search.")

return

try:

# Fetch all records with the given customer\_id

db\_cursor.execute("SELECT \* FROM customers WHERE customer\_id = %s", (customer\_id,))

records = db\_cursor.fetchall() # Fetch all matching records

if records:

# Prepare the message with all customer details

customer\_details = "\n".join([f"ID: {record[0]}\nName: {record[1]}\nProduct: {record[2]}\n"

f"Expense: {record[3]}\nQuantity Sold: {record[4]}\n"

f"{'-'\*40}" for record in records])

messagebox.showinfo("Customer Found", customer\_details)

else:

messagebox.showinfo("Not Found", "No customer found with the given ID.")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to search customer: {e}")

def update\_mysql():

try:

customer\_id = customer\_id\_sql\_input.get()

new\_expense = float(new\_expense\_input.get())

if not customer\_id or not new\_expense:

messagebox.showerror("Error", "Customer ID and new expense value are required.")

return

db\_cursor.execute("UPDATE customers SET expense = %s WHERE customer\_id = %s", (new\_expense, customer\_id))

db\_connection.commit()

messagebox.showinfo("Success", "Customer expense updated successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to update customer: {e}")

# Function to delete customer data from MySQL

def delete\_from\_mysql():

customer\_id = customer\_id\_sql\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required to delete.")

return

try:

db\_cursor.execute("DELETE FROM customers WHERE customer\_id = %s", (customer\_id,))

db\_connection.commit()

messagebox.showinfo("Success", "Customer data deleted successfully!")

except pymysql.MySQLError as e:

messagebox.showerror("Error", f"Failed to delete customer: {e}")

# Function to open the Confidential window

def open\_confidential\_window():

global confidential\_window

if confidential\_window is None or not tk.Toplevel.winfo\_exists(confidential\_window):

confidential\_window = tk.Toplevel(root)

confidential\_window.title("Confidential Options")

confidential\_window.geometry("500x400")

# Save to MySQL button

ttk.Button(confidential\_window, text="Save Data to MySQL", command=save\_to\_mysql).pack(pady=10)

# Search from MySQL

search\_frame = ttk.LabelFrame(confidential\_window, text="Search from MySQL", padding=10)

search\_frame.pack(fill="x", padx=10, pady=5)

global customer\_id\_sql\_input

customer\_id\_sql\_input = tk.StringVar()

ttk.Label(search\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(search\_frame, textvariable=customer\_id\_sql\_input, width=30).grid(row=0, column=1, padx=5, pady=5)

ttk.Button(search\_frame, text="Search", command=search\_from\_mysql).grid(row=0, column=2, padx=5, pady=5)

# Update MySQL

update\_frame = ttk.LabelFrame(confidential\_window, text="Update MySQL", padding=10)

update\_frame.pack(fill="x", padx=10, pady=5)

global new\_expense\_input

new\_expense\_input = tk.StringVar()

ttk.Label(update\_frame, text="New Expense:").grid(row=1, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(update\_frame, textvariable=new\_expense\_input, width=30).grid(row=1, column=1, padx=5, pady=5)

ttk.Button(update\_frame, text="Update", command=update\_mysql).grid(row=1, column=2, padx=5, pady=5)

# Delete from MySQL

delete\_frame = ttk.LabelFrame(confidential\_window, text="Delete from MySQL", padding=10)

delete\_frame.pack(fill="x", padx=10, pady=5)

ttk.Button(delete\_frame, text="Delete", command=delete\_from\_mysql).pack(pady=10)

# Back Button

ttk.Button(confidential\_window, text="Exit", command=lambda: confidential\_window.destroy()).pack(pady=20)

# Function to prompt for password and proceed

def password\_prompt():

password\_window = tk.Toplevel(root)

password\_window.title("Enter Password")

password\_window.geometry("300x150")

ttk.Label(password\_window, text="Enter Password:", font=("Arial", 12)).pack(pady=10)

password\_input = tk.StringVar()

ttk.Entry(password\_window, textvariable=password\_input, show="\*", width=20).pack(pady=5)

def check\_password():

if password\_input.get() == PASSWORD:

password\_window.destroy()

open\_confidential\_window()

else:

messagebox.showerror("Access Denied", "Incorrect Password")

ttk.Button(password\_window, text="Submit", command=check\_password).pack(pady=10)

# Connect to MySQL and Google Sh

# Add a new frame for Google Sheets connection

# Frame for Google Sheets connection

# Function to reset and allow connecting to a new sheet dynamically

def reset\_google\_connector():

global SERVICE\_ACCOUNT\_FILE, SPREADSHEET\_ID, credentials, client, sheet

SERVICE\_ACCOUNT\_FILE = None

SPREADSHEET\_ID = None

credentials = None

client = None

sheet = None

sheet\_id\_input.set("") # Clear the Sheet ID input

messagebox.showinfo("Reset", "Google Connector has been reset. Please provide new details.")

# Keep your original GUI functionality

# ...

# QR Code Generation Function

def generate\_qr(data, file\_name):

qr = qrcode.QRCode(version=1, box\_size=10, border=5)

qr.add\_data(data)

qr.make(fit=True)

img = qr.make\_image(fill="black", back\_color="white")

img.save(file\_name)

return file\_name

# PDF Generation Function

def generate\_pdf(data, file\_name):

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, txt="Customer Report", ln=True, align="C")

# Customer details

for key, value in data["customer"].items():

pdf.cell(200, 10, txt=f"{key}: {value}", ln=True)

pdf.ln(10) # Add some space before listing expenses

# Product expenses

pdf.cell(200, 10, txt="Individual Product Expenses:", ln=True)

for product in data["products"]:

pdf.cell(200, 10, txt=f"Product: {product['name']} - Expense: {product['expense']}", ln=True)

pdf.ln(10) # Add space

pdf.cell(200, 10, txt=f"Total Expense: {data['total\_expense']}", ln=True)

pdf.output(file\_name)

return file\_name

# GUI Functions

def product\_estimation\_analysis():

try:

data = sheet.get\_all\_records()

df = pd.DataFrame(data)

if df.empty:

messagebox.showerror("Error", "No data available for analysis.")

return

sales\_trend = df.groupby("Product Name")["Quantity Sold"].sum()

sales\_trend.plot(kind="line", title="Sales Trend by Product")

plt.xlabel("Product Name")

plt.ylabel("Quantity Sold")

plt.show()

except Exception as e:

messagebox.showerror("Error", f"Failed to perform analysis: {e}")

def search\_customer():

customer\_id = customer\_id\_input.get()

if not customer\_id:

messagebox.showerror("Error", "Customer ID is required.")

return

try:

data = sheet.get\_all\_records()

customer\_data = next(

(customer for customer in data if str(customer["Customer ID"]) == str(customer\_id)), None

)

if customer\_data is None:

messagebox.showinfo("Not Found", "Customer not found.")

return

# Gather individual expenses for the customer

customer\_expenses = [customer for customer in data if str(customer["Customer ID"]) == str(customer\_id)]

total\_expense = sum(exp["Expense"] for exp in customer\_expenses)

# Prepare data for displaying and generating the PDF

customer\_info = {

"Customer ID": customer\_data["Customer ID"],

"Customer Name": customer\_data["Customer Name"],

"Total Expense": total\_expense

}

products = [{"name": exp["Product Name"], "expense": exp["Expense"]} for exp in customer\_expenses]

# Show the customer data in a new window

show\_customer\_data\_window(customer\_info, products, total\_expense)

# Store the customer info and products for PDF generation

global selected\_customer\_data

selected\_customer\_data = {"customer": customer\_info, "products": products, "total\_expense": total\_expense}

except Exception as e:

messagebox.showerror("Error", f"Failed to search customer: {e}")

def show\_customer\_data\_window(customer\_info, products, total\_expense):

# New window to display customer info and product expenses

show\_window = tk.Toplevel(root)

show\_window.title(f"Customer: {customer\_info['Customer Name']}")

customer\_label = ttk.Label(show\_window, text=f"Customer ID: {customer\_info['Customer ID']}\n"

f"Customer Name: {customer\_info['Customer Name']}\n"

f"Total Expense: {total\_expense}")

customer\_label.pack(pady=10)

ttk.Label(show\_window, text="Individual Expenses:").pack()

for product in products:

ttk.Label(show\_window, text=f"Product: {product['name']} - Expense: {product['expense']}").pack()

ttk.Button(show\_window, text="Report PDF", command=generate\_customer\_pdf).pack(pady=10)

def generate\_customer\_pdf():

global selected\_customer\_data

if selected\_customer\_data is None:

messagebox.showerror("Error", "No customer selected for PDF generation.")

return

file\_name = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF Files", "\*.pdf")])

if file\_name:

generate\_pdf(selected\_customer\_data, file\_name)

messagebox.showinfo("Success", f"Customer report saved at {file\_name}")

def show\_customers():

try:

data = sheet.get\_all\_records()

if not data:

messagebox.showinfo("No Data", "No customers available.")

return

show\_window = tk.Toplevel(root)

show\_window.title("All Customers")

columns = list(data[0].keys())

tree = ttk.Treeview(show\_window, columns=columns, show="headings")

for col in columns:

tree.heading(col, text=col)

tree.column(col, width=120)

for row in data:

tree.insert("", tk.END, values=list(row.values()))

tree.pack(fill="both", expand=True)

except Exception as e:

messagebox.showerror("Error", f"Failed to fetch customers: {e}")

def generate\_qr\_and\_pdf():

# Get inputs

product\_name = product\_name\_input.get()

price = product\_price\_input.get()

expire\_date = product\_expiry\_input.get()

webapp\_url = webapp\_input.get()

if product\_name and price and expire\_date and webapp\_url:

# Generate a dynamic URL for the web app

qr\_url = f"{webapp\_url}?product\_name={product\_name}&expense={price}"

# Generate QR Code

file\_name\_qr = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG Files", "\*.png")])

if file\_name\_qr:

qr = qrcode.make(qr\_url)

qr.save(file\_name\_qr)

messagebox.showinfo("QR Code", f"QR Code saved at {file\_name\_qr}")

# Generate PDF

data\_pdf = {"Product Name": product\_name, "Price": price, "Expire Date": expire\_date}

file\_name\_pdf = filedialog.asksaveasfilename(defaultextension=".pdf", filetypes=[("PDF Files", "\*.pdf")])

if file\_name\_pdf:

pdf = FPDF()

pdf.add\_page()

pdf.set\_font("Arial", size=12)

pdf.cell(200, 10, txt="Product Information", ln=True, align="C")

for key, value in data\_pdf.items():

pdf.cell(200, 10, txt=f"{key}: {value}", ln=True, align="L")

pdf.output(file\_name\_pdf)

messagebox.showinfo("PDF", f"PDF saved at {file\_name\_pdf}")

# Notify user

messagebox.showinfo("QR Code Generated", f"Scan the QR code to open the web app and enter ID/Name.")

else:

messagebox.showerror("Error", "All fields are required.")

# Main Application

root = tk.Tk()

root.title("Automated Bussiness Solution With Qr Code Intregation")

# Frame for Working Mode

working\_frame = ttk.LabelFrame(root, text="Working Mode", padding=10)

working\_frame.pack(fill="x", padx=10, pady=5)

# VIP Mode

vip\_frame = ttk.LabelFrame(root, text="VIP Mode - Product QR & PDF", padding=10)

vip\_frame.pack(fill="x", padx=10, pady=5)

product\_name\_input = tk.StringVar()

product\_price\_input = tk.StringVar()

product\_expiry\_input = tk.StringVar()

webapp\_input=tk.StringVar()

ttk.Label(vip\_frame, text="Product Name:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_name\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Price:").grid(row=1, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_price\_input, width=40).grid(row=1, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Expire Date:").grid(row=2, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=product\_expiry\_input, width=40).grid(row=2, column=1, padx=5, pady=5)

ttk.Label(vip\_frame, text="Web App Connect:").grid(row=3, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(vip\_frame, textvariable=webapp\_input, width=40).grid(row=3, column=1, padx=5, pady=5)

ttk.Button(vip\_frame, text="Generate QR Code & PDF", command=generate\_qr\_and\_pdf).grid(row=4, column=0, columnspan=2, padx=5, pady=10)

# Frame for Business Analysis

business\_frame = ttk.LabelFrame(root, text="Business Analysis", padding=10)

business\_frame.pack(fill="x", padx=10, pady=5)

ttk.Button(business\_frame, text="Customer Discount Analysis", command=customer\_discount\_analysis).pack(side="left", padx=10)

ttk.Button(business\_frame, text="Product Estimation Analysis", command=product\_estimation\_analysis).pack(side="left", padx=10)

# Search Customer

search\_frame = ttk.LabelFrame(root, text="Search Customer", padding=10)

search\_frame.pack(fill="x", padx=10, pady=5)

customer\_id\_input = tk.StringVar()

ttk.Label(search\_frame, text="Customer ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(search\_frame, textvariable=customer\_id\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

# Separate buttons for Search and PDF Generation

ttk.Button(search\_frame, text="Search Customer", command=search\_customer).grid(row=1, column=0, padx=5, pady=10)

# Show Customers Button

ttk.Button(root, text="Show All Customers", command=show\_customers).pack(padx=10, pady=10)

ttk.Button(root, text="Confidential", command=password\_prompt).pack(padx=2,pady=2)

# Google Sheets Connector Frame

sheet\_connector\_frame = ttk.LabelFrame(root, text="Google Sheet Connector", padding=10)

sheet\_connector\_frame.pack(fill="x", padx=10, pady=5)

# Sheet ID input and buttons

sheet\_id\_input = tk.StringVar()

ttk.Label(sheet\_connector\_frame, text="Sheet ID:").grid(row=0, column=0, padx=5, pady=5, sticky="w")

ttk.Entry(sheet\_connector\_frame, textvariable=sheet\_id\_input, width=40).grid(row=0, column=1, padx=5, pady=5)

ttk.Button(sheet\_connector\_frame, text="Set Sheet ID", command=set\_sheet\_id).grid(row=0, column=2, padx=5, pady=5)

# Service Account File input and reset button

ttk.Button(sheet\_connector\_frame, text="Set Service Account File", command=set\_service\_account\_file).grid(row=1, column=0, columnspan=2, pady=10)

ttk.Button(sheet\_connector\_frame, text="Reset Connector", command=reset\_google\_connector).grid(row=1, column=2, pady=10)

# Running the Tkinter loop

#second winow funtionality

# Main application function

connect\_to\_mysql()

connect\_to\_google\_sheets()

# Run the main application

root.mainloop()