



NATIONAL INSTITUTE OF TECHNOLOGY, WARANGAL

DATABASE MANAGEMENT SYSTEM FOR RETAIL GARMENT SHOP

Submitted by:

Name: Sarvesh Charpe

Roll no.: 22EEB0A41

Branch: EEE (Section-A)

Subject: Python Programming

Database Management System for Retail Garment Shop

1.INTRODUCTION

The Python project aims to develop a Database Management System (DBMS) for a retail garment shop named "SHUBHAM COLLECTION." The system is designed to streamline various operations essential for the efficient management of stock, customer data, and billing processes. The key features include adding, deleting, and searching data, as well as printing and sending bills to customers via email.

This project is essential for a retail garment shop as it simplifies tasks like managing inventory, creating bills, and sending them via email. It ensures accurate record-keeping, saves time, and enhances customer service. With features like data entry, deletion, and searching, it streamlines daily operations, making it easier to track product details. The billing module adds efficiency, automating the billing process and improving customer experience. Overall, this project is crucial for the shop's smooth functioning, offering a user-friendly system that optimizes tasks, reduces manual efforts, and contributes to a more organized functioning of the shop.

2.FEATURES

The First Window Displays two options to the user of 'Manage Data' and 'Print Bill'.

By choosing option of 'Manage Data' the user is taken to another window where the user gets multiple options for adding data, displaying data, searching data , and deleting data. The following information is given by user of each product:

- Brand
- Garment Type
- Product Code
- Size
- Colour
- Quantity
- MRP

The information of all the products is stored in a database in MySQL int a table named 'Product_Details' in a Database named 'Shubham_Collection' which contains the columns that are mentioned above.

2.1. Introduction Window

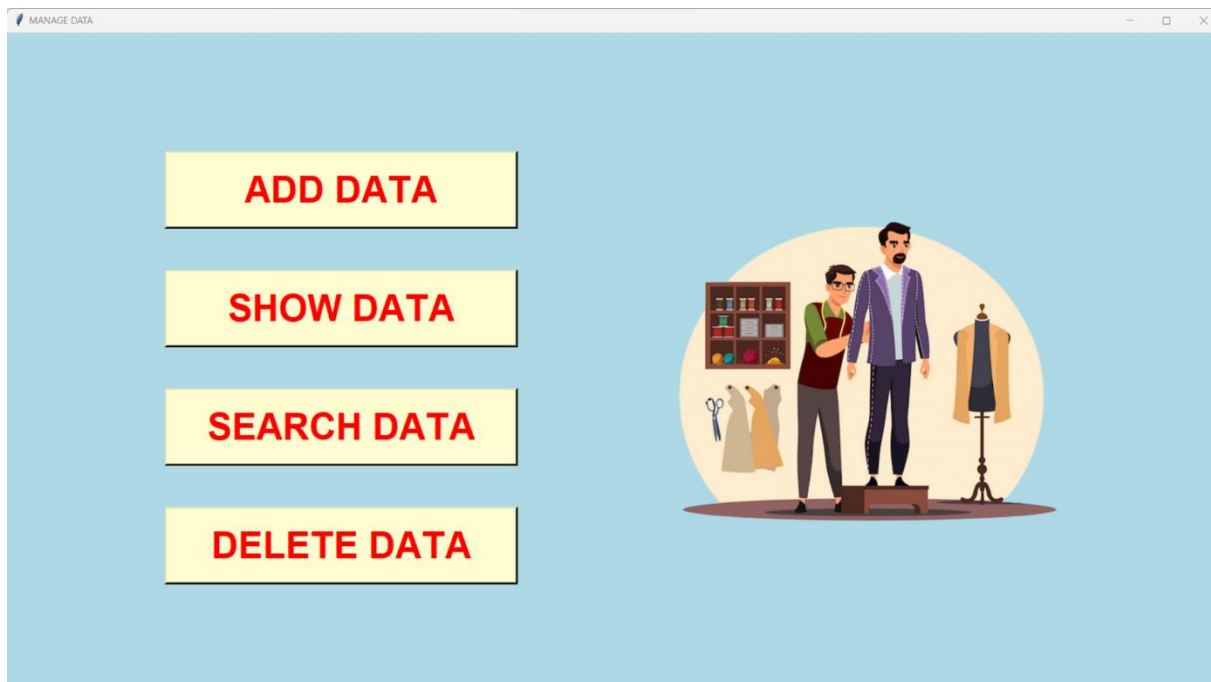


2.2 Manage Stock

By choosing Manage Stock the user is taken to another window.

This window provides four features to the user-

- Adding data to the database
- Showing data (Displaying the data)
- Searching data on basis of certain criterion
- Deleting of data



2.3 ADDING DATA



```
# Create labels and entry fields for data input
tk.Label(root, text="Brand:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=75)
tk.Label(root, text="Garment Type:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=125)
tk.Label(root, text="Product Code:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=175)
tk.Label(root, text="Size:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=225)
tk.Label(root, text="Colour:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=275)
tk.Label(root, text="Quantity:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=325)
tk.Label(root, text="MRP:", font=('Arial', 15, 'bold'), bg='lightblue', fg='black').place(x=50, y=375)

Brand_entry = tk.Entry(root)
Garment_Type_entry = tk.Entry(root)
Product_Code_entry = tk.Entry(root)
Size_entry = tk.Entry(root)
Colour_entry = tk.Entry(root)
Quantity_entry = tk.Entry(root)
MRP_entry = tk.Entry(root)

Brand_entry.place(x=200, y=75)
Garment_Type_entry.place(x=200, y=125)
Product_Code_entry.place(x=200, y=175)
Size_entry.place(x=200, y=225)
Colour_entry.place(x=200, y=275)
Quantity_entry.place(x=200, y=325)
MRP_entry.place(x=200, y=375)

# Function to insert data into the MySQL database
def insert_data():
    Brand = Brand_entry.get()
    Garment_Type = Garment_Type_entry.get()
    Product_Code = Product_Code_entry.get()
    Size = Size_entry.get()
    Colour = Colour_entry.get()
    Quantity = Quantity_entry.get()
    MRP = MRP_entry.get()

    sql = "INSERT INTO Product_Details (Brand, Garment_Type, Product_Code, Size, Colour, Quantity, MRP) VALUES (%s, %s, %s, %s, %s, %s, %s)"
    val = (Brand, Garment_Type, Product_Code, Size, Colour, Quantity, MRP)
    mycursor.execute(sql, val)
    mycon.commit()

# Create a button to trigger the data insertion
tk.Button(root, text="Add Product", command=insert_data, font=('Arial', 25, 'bold'), bg='yellow', fg='red', borderwidth=3, width=10, height=1).place(x=200, y=425)
```

The module is designed to allow users to input and store product details, including information such as brand, garment type, product code, size, colour, quantity, and MRP (Maximum Retail Price). The entered data is then stored in a MySQL database named "Shubham_Collection".

The utilization of Tkinter for GUI and MySQL for data storage is used for making this module function.

On clicking on the button 'Add Product', the data is stored into the database.

2.4 SHOWING DATA

SHOW DATA						
Brand	Garment_Type	Product_Code	Size	Colour	Quantity	MRP
Cambridge	Shirt	SCW800	40	White	1	800
Internity	Jeans	LJ758	32	Blue	1	799
Collars	Formal Pant	FP855	36	Cream	0	999
Red & White	Shirt	RW742	40	Black	1	800
Macpi	Tahirt	HF645	36	Blue	1	450
Peter England	Shirt	PE3059	42	Brown	1	1299
Vituda	Shirt	JDH486	42	white	1	899
Nike	Tshirt	TS8724	38	Violet	2	1200
LUIS VITTON	JEANS	1022	32	BLUE	1	2000
Cambridge	Formal Pant	h473j	36	Grey	0	999
Oxenberry	Shirt	OX6384	42	White	1	1299
Cambridge	Shirt	CH847	40	Yellow	2	999
Encasty	Shirt	EN847	38	Yellow	1	899
Internity	Jeans	FH756	34	Blue	3	900
Internity	Jeans	DF345	34	Black	4	700

```
# Create a cursor object to execute SQL commands
mycursor = mycon.cursor()

# Fetch data from the MySQL table
mycursor.execute("SELECT * FROM Product_Details")
rows = mycursor.fetchall()

# Tkinter window
root = tk.Tk()
root.title("SHOW DATA")
root.geometry("1920x1080")

label1=Label(root,text="SHOW DATA",
              font=('Arial',35,'bold'),
              fg='red'
              )
label1.pack()

# Creating a treeview
tree = ttk.Treeview(root)
tree['show']='headings' #now it will not display first empty column

#selecting theme of the table
s=ttk.Style(root)
s.theme_use("clam")
s.configure("Treeview",
            background="lightblue",
            foreground='black',
            fieldbackground='lightblue',
            rowheight=25
            )

s.configure("",font=('Arial',10,'bold'))
s.configure("Treeview.Heading",font=('Arial',10,'bold'),foreground='red',background='#FFD0D0')

tree["columns"] = tuple([i[0] for i in mycursor.description])

# Formatting columns
for col in tree["columns"]:
    tree.column(col, anchor="center",minwidth=50)
    tree.heading(col, text=col)

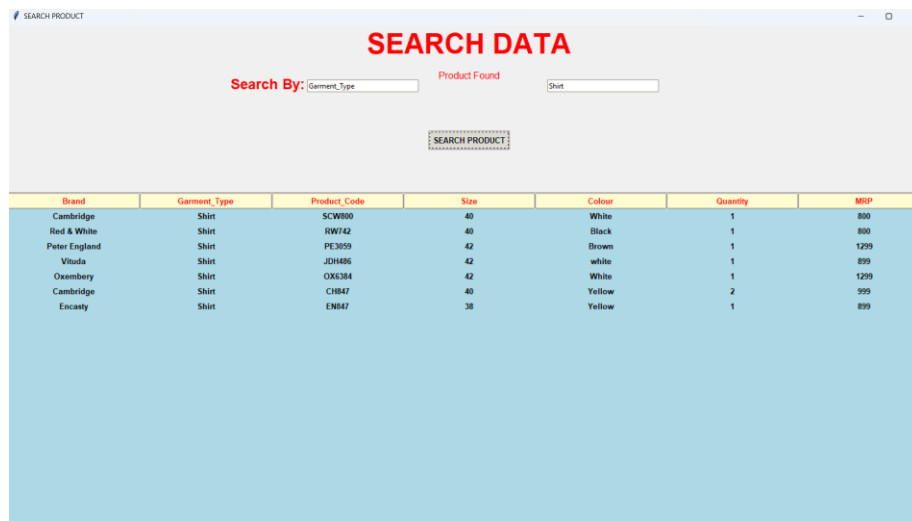
# Inserting data
for row in rows:
    tree.insert("", "end", values=row)

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

root.mainloop()
```

This feature provides a clear and organized representation of product details stored in the MySQL database. The use of Tkinter for GUI and Treeview for tabular data presentation ensures a user-friendly and visually appealing display. This module can be seamlessly integrated into a larger system, offering a valuable tool for monitoring and managing product information in the retail garment shop.

2.5 SEARCH DATA



```
# Create the main application window
root = tk.Tk()
root.title("SEARCH PRODUCT")
root.geometry("1920x1080")

def search():
    ColumnType = entry1.get()
    query = entry2.get()
    mycursor.execute(f"SELECT * FROM Product_Details WHERE {ColumnType} = '{query}'")
    result = mycursor.fetchall()

    if result:
        error_label.config(text="Product Found", fg='red')
    if not result:
        error_label.config(text="Product Not Found", fg='red')
        return

    for i in tree.get_children():
        tree.delete(i)
    for row in result:
        tree.insert("", "end", values=row)

label1 = tk.Label(root, text="SEARCH DATA", font=('Arial', 35, 'bold'), fg='red')
label1.pack()

label2 = tk.Label(root, text="Search By: ", font=('Arial', 18, 'bold'), fg='red')
label2.place(x=370, y=80)

s = ttk.Style(root)
s.theme_use("clam")
s.configure("Treeview", background="lightblue", foreground='black', fieldbackground='lightblue', rowheight=25)
s.configure(".", font=('Arial', 18, 'bold'))
s.configure("Treeview.Hheading", font=('Arial', 18, 'bold'), foreground='red', background='#FFDD00')

entry1 = ttk.Entry(root, width=30)
entry1.place(x=500, y=90)
entry2 = ttk.Entry(root, width=30)
entry2.place(x=900, y=90)

error_label = tk.Label(root, text="", font=('Arial', 12), fg='red')
error_label.pack(pady=10)

search_button = ttk.Button(root, text="SEARCH PRODUCT", command=search)
search_button.pack(pady=70)

tree = ttk.Treeview(root, columns=("Brand", "Garment_Type", "Product_Code", "Size", "Colour", "Quantity", "MRP"), show="headings")
tree.heading("Brand", text="Brand", anchor=tk.CENTER)
tree.heading("Garment_Type", text="Garment_Type", anchor=tk.CENTER)
tree.heading("Product_Code", text="Product_Code", anchor=tk.CENTER)
tree.heading("Size", text="Size", anchor=tk.CENTER)
tree.heading("Colour", text="Colour", anchor=tk.CENTER)
tree.heading("Quantity", text="Quantity", anchor=tk.CENTER)
tree.heading("MRP", text="MRP", anchor=tk.CENTER)

# Center align data
for col in ("Brand", "Garment_Type", "Product_Code", "Size", "Colour", "Quantity", "MRP"):
    tree.column(col, anchor=tk.CENTER)

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

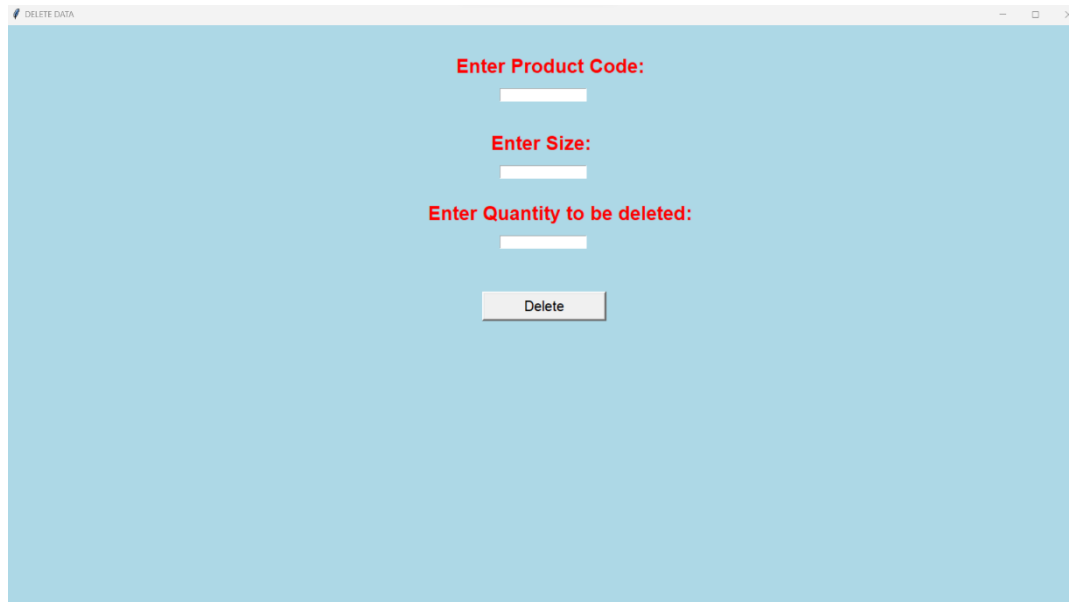
root.mainloop()
```

The data searching feature presented in this Python script provides an interactive and visually appealing way for users to search for specific products within the retail garment shop's database on basis of various criterias. The integration of Tkinter for GUI and Treeview for tabular data presentation enhances the user experience.

2.6 DELETE DATA

The module allows users to connect to a MySQL database, enter specific criteria (product code, size, and quantity), and delete the specified quantity of a product from the database.

Also, this code is linked to billing feature. When a product is purchase by a customer, that data is deleted from the database automatically.



```
def DeleteProduct():
    mycon = co.connect(
        host='localhost',
        user='root',
        passwd='Dhairyag29',
        database='Shubham_Collection'
    )

    # Create a cursor object to execute SQL commands
    mycursor = mycon.cursor()

    error_label = None

    # Function to reduce data in the column's row by one
    def DeletingData():
        product_code = entry_product_code.get()
        size = entry_size.get()
        qty = entry_qty.get()

        # Check if the product exists with the given code and size
        select_query = "SELECT Quantity FROM Product_Details WHERE Product_Code = %s AND Size = %s;"
        mycursor.execute(select_query, (product_code, size))
        result = mycursor.fetchone()

        if result:
            # If quantity is greater than 0, update the quantity
            if result[0] > 0:
                update_query = "UPDATE Product_Details SET Quantity = Quantity - %s WHERE Product_Code = %s AND Size = %s;"
                mycursor.execute(update_query, (qty, product_code, size))
                mycon.commit()
                print(mycursor.rowcount)

                error_label.config(text="Data Deleted Successfully", fg="red")
            else:
                error_label.config(text="Quantity is already 0 for the specified product and size", fg="red")
        else:
            error_label.config(text="Product not found for the specified code and size", fg="red")

    root = Tk()
    root.title("DELETE DATA")
    root.geometry("1020x1080")
    root.config(bg="lightblue")

    # Create a label for product code
    label_product_code = tk.Label(root, text="Enter Product Code: ", font=("Arial", 20, "bold"), fg="red", bg="lightblue")
    label_product_code.place(x=640, y=40)

    # Create an entry for product code
    entry_product_code = tk.Entry(root)
    entry_product_code.place(x=705, y=90)

    # Create a label for size
    label_size = tk.Label(root, text="Enter Size: ", font=("Arial", 20, "bold"), fg="red", bg="lightblue")
    label_size.place(x=690, y=150)

    # Create an entry for size
    entry_size = tk.Entry(root)
    entry_size.place(x=705, y=200)

    label_qty = tk.Label(root, text="Enter Quantity to be deleted: ", font=("Arial", 20, "bold"), fg="red", bg="lightblue")
    label_qty.place(x=600, y=250)
```

2.7 PRINTING BILL

The screenshot shows a Python Tkinter window titled "BILLING". At the top, there are three input fields: "Product_Code:", "Size:", and "Quantity:". Below these are two buttons: "ADD TO BILL" and "SEND FINAL BILL". A table displays the current bill items:

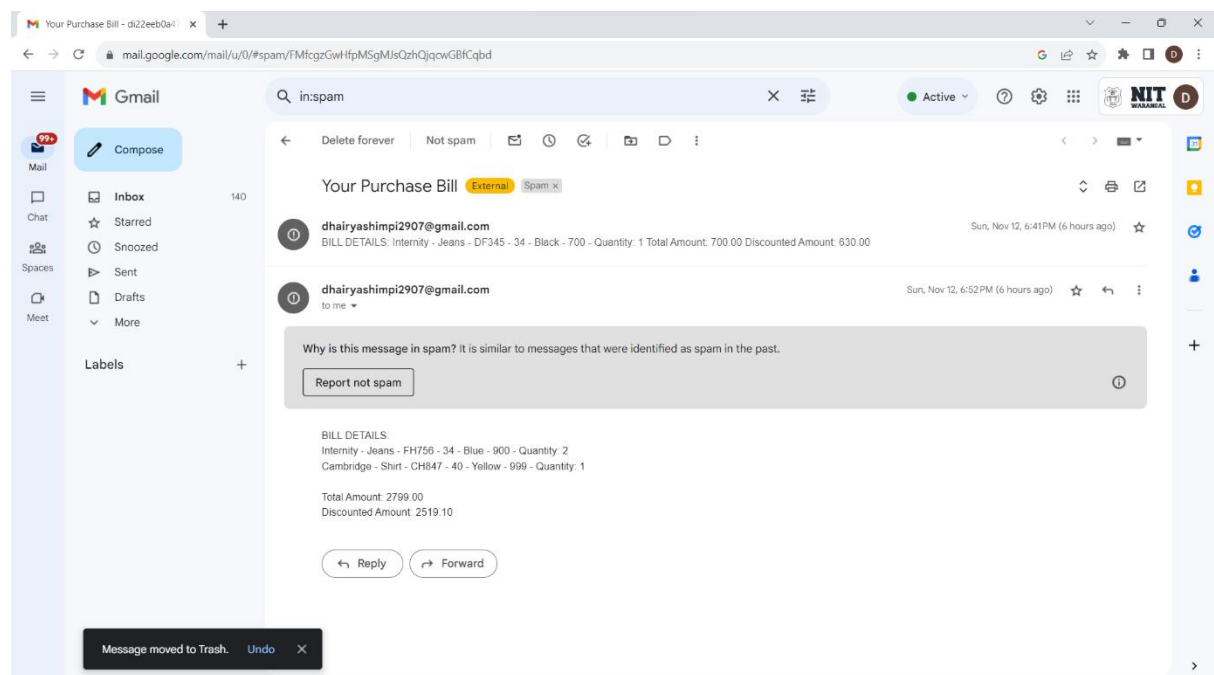
Brand	Garment_Type	Product_Code	Size	Colour	MRP	Quantity
Internity	Jeans	FH756	34	Blue	900	2
Cambridge	Shirt	CH847	40	Yellow	999	1

Below the table, there are input fields for "Discount (%):" (set to 10) and "Email:" (set to di22eeb0a47@student.nitw.ac.in). On the right, a box displays the "Total Amount: 2799.00" and "Discounted Amount: 2519.10".

The module enables users to generate a bill for purchased items, apply discounts, and send the final bill to customers via email. The Tkinter library is used for creating a graphical user interface (GUI), and the smtplib library facilitates email communication.

The billing and email module presented in this Python script provides a comprehensive solution for generating bills, applying discounts, and sending the final bill to customers via email.

The image shown below is the image of the bill received by the customer.




```

def bill():
    Product_Code = entry1.get()
    Size = entry2.get()
    Qty = entry3.get()

    query = "SELECT Brand, Garment_Type, Product_Code, Size, Colour, MRP FROM Product_Details WHERE Product_Code=%s AND Size=%s"
    mycursor.execute(query, (Product_Code, Size))
    result = mycursor.fetchall()

    for row in result:
        row_with_qty = row + (Qty,)
        product_list.append(row_with_qty)
        tree.insert("", "end", values=row_with_qty)

    # Clear the entry fields
    entry1.delete(0, tk.END)
    entry2.delete(0, tk.END)
    entry3.delete(0, tk.END)

def send_final_bill():
    total_amount = sum(float(row[-2]) * float(row[-1]) for row in product_list)
    discount_percentage = entry_discount.get()

    # Check if a discount percentage is provided
    if discount_percentage:
        discount_amount = (float(discount_percentage) / 100) * total_amount
        discounted_total = total_amount - discount_amount
        total_label.config(text=f"Total Amount: {total_amount:.2f} \n Discounted Amount: {discounted_total:.2f}")
    else:
        total_label.config(text=f"Total Amount: {total_amount:.2f}")

    # Get the customer's email
    customer_email = entry_email.get()

    # Send the final bill via email
    send_bill_via_email(customer_email, product_list, total_amount, discount_percentage)

    # Update the product quantity in the database
    for row in product_list:
        update_query = "UPDATE Product_Details SET Quantity = Quantity - %s WHERE Product_Code = %s AND Size = %s;"
        mycursor.execute(update_query, (row[-1], row[2], row[3]))
        mycon.commit()

    # Clear the product list
    product_list.clear()

```

```

def send_bill_via_email(customer_email, product_list, total_amount, discount_percentage):
    # Your email credentials
    email_address = 'dhairyashimp12987@gmail.com'
    email_password = 'dick vndy kczo dkes'

    # Create the email message (parameter) customer_email: Any
    msg = MIMEMu
    msg['From'] = email_address
    msg['To'] = customer_email
    msg['Subject'] = 'Your Purchase Bill'

    # Create the body of the email
    bill_text = "BILL DETAILS:\n"
    for row in product_list:
        bill_text += f"{row[0]} - {row[1]} - {row[2]} - {row[3]} - {row[4]} - {row[5]} - Quantity: {row[6]}\n"

    bill_text += f"\nTotal Amount: {total_amount:.2f}"

    # Check if a discount percentage is provided
    if discount_percentage:
        discount_amount = (float(discount_percentage) / 100) * total_amount
        discounted_total = total_amount - discount_amount
        bill_text += f"\nDiscounted Amount: {discounted_total:.2f}"

    msg.attach(MIMEText(bill_text, 'plain'))

    # Set up the SMTP server
    with smtplib.SMTP('smtp.gmail.com', 587) as server:
        server.starttls()
        server.login(email_address, email_password)

        # Send the email
        server.sendmail(email_address, customer_email, msg.as_string())

    print(f"Bill sent to {customer_email} via email.")

```

CONCLUSION-

In summary, the Python project we've created for "SHUBHAM COLLECTION," a clothing store, is like a smart helper for the shop. It helps in keeping track of all the clothes they have, making bills for customers, and even sending those bills through email.

The part where we manage data lets the shop easily add, see, find, and remove information about the clothes they have in stock. This helps in keeping everything organized, like a digital catalog of their products.

The billing part is like a virtual cashier. It creates bills for customers, applies any discounts, and can even send the bill directly to the customer's email. This not only makes the billing process smoother but also makes customers happy by giving them a clear record of their purchase.

This Python project is a valuable tool for the clothing store. It makes managing stock and dealing with customers much easier, saving time and keeping things running smoothly. It is like a modern and efficient assistant for the shop's daily tasks.