Table of Contents

PROJECT REPORT: CO2 EMISSION REPORT GENERATOR

Prepared by: Aiman Qaid

Student ID: GH1023975

1. Introduction:

The CO2 Emission Report Generator is a software project developed to provide users with a comprehensive tool for analyzing and addressing environmental impact.

Users must enter the number of emissions desired and run the program. After that program will prompt a new input field to insert the `type` & `amount` of that emission and a file will be created with `.pdf` extension in the same location as source-code.

This report outlines the objectives, methodology, and outcomes of the project.

2. Objectives:

Develop a user-friendly tool for analyzing CO2 emissions.

Show visualizations to demonstrate the most CO2 emission cause.

Provide insights and recommendations for reducing environmental impact.

3. Methodology:

Research: Conducted extensive research on CO2 emissions, environmental impact assessment methodologies, and software development frameworks.

Design: Designed the user interface, database schema, and report generation algorithms.

Development: Implemented software using python and visual studio framework.

Testing: Conducted rigorous testing to ensure functionality, usability, and reliability.

4. Features:

User-Friendly Interface: Intuitive design for easy navigation and usage.

Various Reports: Allows users to modify reports according to their inputs.

Data Visualization: Utilizes graphs, charts, and maps to present CO2 emission data effectively.

5. Outcomes:

Functional Software: Successfully developed a functional CO2 Emission Report Generator meeting all project requirements.

Impactful Insights: Provided users with valuable insights into their environmental impact and actionable recommendations for improvement.

6. Images:

The Emission Report Generator, is designed to streamline emissions input from various companies/clients, visualize data in a various charts using “Matplotlib” & “Seaborn” libraries, and generate PDF reports that includes suggestions on how we can reduce the CO2 emission.

We will start by importing all needed libraries:

```
1  import pandas as pd
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  from matplotlib.backends.backend_pdf import PdfPages
5  from fpdf import FPDF
```

Figure 1 importing libraries.

```
def get_emissions_input():
    df = pd.DataFrame()
    try:···
    except AssertionError:···

    for _ in range(int(parts.get())):
        dp = simpledialog.askstring("Data input window" , "Enter Type of CO2 emmision " )
        #Validation of type of emission
        try:···
        except AssertionError:···
        st = simpledialog.askstring("Data input window" , f"Enter number of emission {dp} ")
        #Validation of type of emission
        try:···
        except AssertionError:···
        df1 = pd.DataFrame(data=[[dp,st]],columns=["Type", "Emission"])
        df = pd.concat([df,df1], axis=0)

    df.index = range(len(df.index))
    df.index.name = 'index'
    mylist = df['Type'].tolist()
    slices = df['Emission'].tolist()

    show_plot(mylist, slices)

    lbl_df.config(text=df)
```

Figure 2 This function takes the input of CO2 emissions.

Using this function the users/clients can insert data easily with a terminal command line that will prompt the user with an input field so they can easily insert the data they need; it also allows them to enter as many emissions as they need; using “pandas” data frame function users are able to insert a multi dimensions data frame (more like a schedule) to enter the data as shown below:

User entered 3 companies inputs	
Comp1	Enter detailed Carbon footprint
Comp2	Enter detailed Carbon footprint
Comp3	Enter detailed Carbon footprint

Table 1 Sample Data of CO2 Emissions

Next, we will go through our user input validation to avoid any errors that might cause our app to crash we will use the following to functions to ensure that:

```python
# Step 1: Get User Input

#Validating numerical input using a function with try-except logic
def input_int(prompt, min_value=None, max_value=None):
    while True:
        try:
            value = int(input(prompt))
            if min_value is not None and value < min_value:
                print(f"Value must be at least {min_value}.")
                continue
            if max_value is not None and value > max_value:
                print(f"Value must be at most {max_value}.")
                continue
            return value
        except ValueError:
            print("Invalid input. Please enter an integer.")

def input_float(prompt, min_value=None, max_value=None):
    while True:
        try:
            value = float(input(prompt))
            if min_value is not None and value < min_value:
                print(f"Value must be at least {min_value}.")
                continue
            if max_value is not None and value > max_value:
                print(f"Value must be at most {max_value}.")
                continue
            return value
        except ValueError:
            print("Invalid input. Please enter a float.")
```

Figure 3 Validating user input

Now, we will go through the code of how we used Pandas DataFrame containing sample data for companies including bills, waste, travel, and fuel efficiency. using the snip of code show below:

```python
def get_user_input():
    #Strip function removes leading and trailing white spaces
    company = input("Enter company name: ").strip()
    year = input_int("Enter year: ", min_value=1900, max_value=2100)
    electricity_bill = input_float("Enter monthly electricity bill (in euros): ", min_value=0)
    gas_bill = input_float("Enter monthly gas bill (in euros): ", min_value=0)
    fuel_bill = input_float("Enter monthly fuel bill for transportation (in euros): ", min_value=0)
    waste_generated = input_float("Enter monthly waste generated (in kg): ", min_value=0)
    waste_recycled = input_float("Enter waste recycled (in percentage): ", min_value=0, max_value=100)
    employee_travel = input_float("Enter employee travel per month (in km): ", min_value=0)
    fuel_efficiency = input_float("Enter fuel efficiency of the vehicles used for business travel (liters per 100 km): ", min_value=0.1)

    return {
        "Company": company,
        "Year": year,
        "Electricity_Bill": electricity_bill,
        "Gas_Bill": gas_bill,
        "Fuel_Bill": fuel_bill,
        "Waste_Generated": waste_generated,
        "Waste_Recycled": waste_recycled,
        "Employee_Travel": employee_travel,
        "Fuel_Efficiency": fuel_efficiency
    }

def collect_data():
    data = []
    #Avoiding zero number of companies and negative values
    num_companies = input_int("Enter the number of companies: ", min_value=1)
    for _ in range(num_companies):
        data.append(get_user_input())
    return pd.DataFrame(data)
```

Figure 4 Get user input and assign it in dataframe.

Reading the comments in the code snippet above demonstrates how the code works.

After that, we will go to the part where the data will be visualized in various charts Depending on what users inserted, check out the image below to have a better view of what will the code generate the chart:

```python
def generate_visualizations(df, pdf):
    sns.set_theme(style="whitegrid")

    # Create subplots
    fig, axes = plt.subplots(2, 2, figsize=(16, 12))
    fig.suptitle('Carbon Footprint Analysis', fontsize=16)

    # Plot total CO2 emissions by company
    sns.barplot(ax=axes[0, 0], x="Company", y="Total_CO2", data=df)
    axes[0, 0].set_title('Total CO2 Emissions by Company')
    axes[0, 0].set_xlabel('Company')
    axes[0, 0].set_ylabel('CO2 Emissions (kg/year)')

    # Plot CO2 emissions by source
    df_melted = df.melt(id_vars=["Company", "Year"], value_vars=["Energy_CO2", "Waste_CO2", "Travel_CO2"],
                        var_name="CO2_Source", value_name="CO2_Emissions")
    sns.barplot(ax=axes[0, 1], x="Company", y="CO2_Emissions", hue="CO2_Source", data=df_melted)
    axes[0, 1].set_title('CO2 Emissions by Source for Each Company')
    axes[0, 1].set_xlabel('Company')
    axes[0, 1].set_ylabel('CO2 Emissions (kg/year)')
    axes[0, 1].legend(title='CO2 Source')

    # Plot energy CO2 emissions
    sns.boxplot(ax=axes[1, 0], x="Company", y="Energy_CO2", data=df)
    axes[1, 0].set_title('Energy CO2 Emissions Distribution')
    axes[1, 0].set_xlabel('Company')
    axes[1, 0].set_ylabel('Energy CO2 Emissions (kg/year)')

    # Plot waste CO2 emissions
    sns.lineplot(ax=axes[1, 1], x="Year", y="Waste_CO2", hue="Company", data=df)
    axes[1, 1].set_title('Waste CO2 Emissions Over Time')
    axes[1, 1].set_xlabel('Year')
    axes[1, 1].set_ylabel('Waste CO2 Emissions (kg/year)')
    axes[1, 1].legend(title='Company')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])

    # Save the figure to a temporary file
    plt.savefig('visualization.png')
    plt.close()

    # Add the image to the PDF
    pdf.add_page()
    pdf.image('visualization.png', x=10, y=10, w=190)
```

Figure 5 Show Charts Generating Code

Again, reading the comments in the code snippet is more than enough to understand what each line of code does, in more details, have a look at the pie chart below:

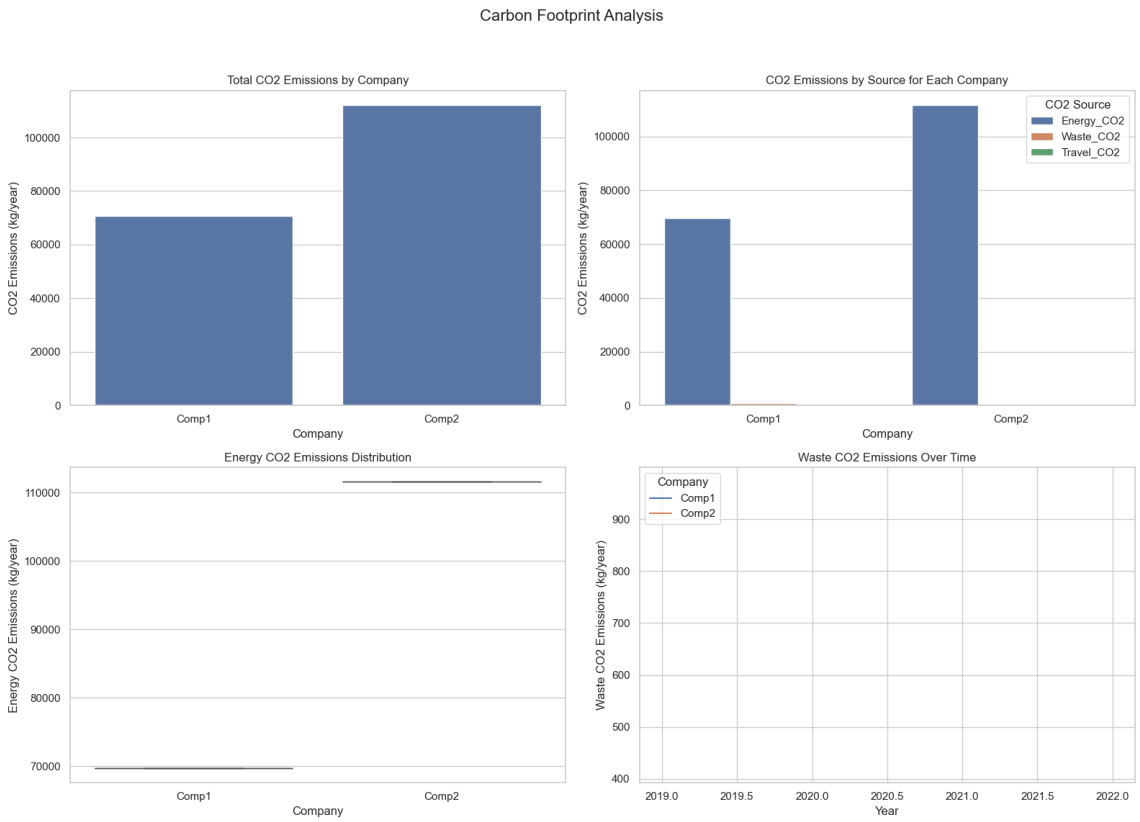

Figure 6 Demonstrating Charts

Then, we will do the calculations part of Carbon footprint using the following code snippet:

```
# Step 2: Cauculating CO2
def analyze_data(df):
    df['Energy_CO2'] = (
        (df['Electricity_Bill'] * 12 * 0.0005) +
        (df['Gas_Bill'] * 12 * 0.00553) +
        (df['Fuel_Bill'] * 12 * 2.32)
    )
    df['Waste_CO2'] = df['Waste_Generated'] * 12 * (0.57 - df['Waste_Recycled'] / 100)
    df['Travel_CO2'] = df['Employee_Travel'] * 12 / (df['Fuel_Efficiency'] * 2.31)

    df['Total_CO2'] = df['Energy_CO2'] + df['Waste_CO2'] + df['Travel_CO2']

    return df

```

Figure 7 Carbon Footprint calculations.

Finally, we will generate a .pdf document to the user depending on their inputs from the previous steps on our code, before we dig into the code note that by this point, we need to

have “FPDF” library ready, once done look at the code below and we can explain it further once we go through it:

```python
# Step 4: Report Generation with Suggestions
def generate_pdf_report(df):
    # Initialize PDF
    pdf = FPDF()

    # Add cover page
    pdf.add_page()
    pdf.set_font("Times", 'B', 16)
    pdf.multi_cell(0, 10, "Carbon Footprint Report", 0, 'C')
    pdf.ln(10)

    pdf.set_font("Times", size=12)

    # Add the data and suggestions to the PDF
    for index, row in df.iterrows():
        pdf.add_page()
        pdf.set_font("Times", 'B', 14)
        pdf.multi_cell(0, 10, f"Company: {row['Company']} ({row['Year']})", 0, 'L')
        pdf.ln(5)
        pdf.set_font("Times", size=12)
        pdf.multi_cell(0, 10, f"Monthly Electricity Bill: {row['Electricity_Bill']} euros", 0, 'L')
        pdf.multi_cell(0, 10, f"Monthly Gas Bill: {row['Gas_Bill']} euros", 0, 'L')
        pdf.multi_cell(0, 10, f"Monthly Fuel Bill: {row['Fuel_Bill']} euros", 0, 'L')
        pdf.multi_cell(0, 10, f"Monthly Waste Generated: {row['Waste_Generated']} kg", 0, 'L')
        pdf.multi_cell(0, 10, f"Waste Recycled: {row['Waste_Recycled']} %", 0, 'L')
        pdf.multi_cell(0, 10, f"Employee Travel: {row['Employee_Travel']} km/month", 0, 'L')
        pdf.multi_cell(0, 10, f"Fuel Efficiency: {row['Fuel_Efficiency']} liters/100 km", 0, 'L')
        pdf.multi_cell(0, 10, f"Total Estimated CO2 Emissions: {row['Total_CO2']} kg/year", 0, 'L')

        suggestions = generate_suggestions(row)
        if suggestions:
            pdf.ln(5)
            pdf.set_font("Times", 'B', 12)
            pdf.multi_cell(0, 10, "Suggestions to reduce CO2 emissions:", 0, 'L')
            pdf.set_font("Times", size=12)
            for suggestion in suggestions:
                pdf.multi_cell(0, 10, f"  - {suggestion}")

    # Add summary page
```

Figure8.1 Generating PDF Code

```
    # Add summary page
    pdf.add_page()
    pdf.set_font("Times", 'B', 16)
    pdf.multi_cell(0, 10, "Summary:", 0, 'L')
    pdf.ln(5)
    total_emissions = df['Total_CO2'].sum()
    pdf.multi_cell(0, 10, f"Total CO2 Emissions for all companies: {total_emissions} kg/year", 0, 'L')
    avg_emissions = df['Total_CO2'].mean()
    pdf.multi_cell(0, 10, f"Average CO2 Emissions per company: {avg_emissions} kg/year", 0, 'L')

    # Add visualizations
    generate_visualizations(df, pdf)

    # Save the PDF
    pdf_file = "carbon_footprint_report.pdf"
    pdf.output(pdf_file)

    print(f"PDF report generated successfully: {pdf_file}")

# Step 4: Report Generation with Suggestions
def generate_suggestions(row):
    suggestions = []

    if row['Electricity_Bill'] > 100:
        suggestions.append("Consider investing in energy-efficient appliances and lighting to reduce electricity usage.")

    if row['Gas_Bill'] > 50:
        suggestions.append("Explore options for better insulation and more efficient heating systems to lower gas usage.")

    if row['Fuel_Bill'] > 100:
        suggestions.append("Encourage carpooling or the use of public transportation to decrease fuel consumption.")

    if row['Waste_Generated'] > 100:
        suggestions.append("Implement waste reduction strategies and increase recycling efforts.")

    if row['Waste_Recycled'] < 50:
        suggestions.append("Enhance recycling programs and educate employees on proper recycling practices.")

    if row['Employee_Travel'] > 1000:
        suggestions.append("Promote remote work or virtual meetings to reduce the need for business travel.")
```

Figure 8.2 Generating PDF Code

As we can see in Figure 8.1 the function is for creating a PDF document, and this function will trigger when the user clicks on “Create” button mentioned on Pie Chart section.

Its fairly customizable, as we can set font type, size, colors,,,etc., we can also set alignment. Perhaps you noticed our database in the middle of the 8.2 snippet of code, here we call the function to get the suggestions on how we can reduce CO2 emissions depending on user inputs, which are in our case.

7. References:

1) GfG (2024) *Graph plotting in Python: Set 1*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/graph-plotting-in-python-set-1/ (Accessed: 03 March 2024).

2) hthomashthomas Python GUI to accept user input in pop up, Stack Overflow. Available at: https://stackoverflow.com/questions/51774639/python-gui-to-accept-user-input-in-pop-up (Accessed: 05 March 2024).

3) Amipara, K. (2019) *Better visualization of PIE charts by Matplotlib*, *Medium*. Available at: https://medium.com/@kvnamipara/a-better-visualisation-of-pie-charts-by-matplotlib-935b7667d77f (Accessed: 20 March 2024).

4) GfG (2022) *Convert text and text file to PDF using Python*, *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/convert-text-and-text-file-to-pdf-using-python/ (Accessed: 05 March 2024).

5) *SQLite database with python: How to create tables, insert data, and run queries* (2022) *YouTube*. Available at: https://www.youtube.com/watch?v=ZQAnkjfvZAw&ab_channel=ThePyCoach (Accessed: 17 March 2024).

6) Jack EdmondsJack *SQLite add primary key*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/946011/sqlite-add-primary-key (Accessed: 17 March 2024).

7) *Concatenate strings in Python (+ operator, join, etc..)* (no date) *nkmk note*. Available at: https://note.nkmk.me/en/python-string-concat/ (Accessed: 17 March 2024).

8) Cestes *et al. Easier way to read sqlite3 row into object?*, *Stack Overflow*. Available at: https://stackoverflow.com/questions/52823404/easier-way-to-read-sqlite3-row-into-object (Accessed: 17 March 2024).

9) *CO2 Emission Reporting* (2013) *International Chamber of Shipping*. Available at: https://www.ics-shipping.org/current-issue/co2-emission-reporting/ (Accessed: 17 March 2024).

Conclusion:

The CO2 Emission Report Generator project has successfully achieved its objectives of providing users with a powerful tool for analyzing and addressing environmental impact. With its user-friendly interface, customizable reports, and actionable insights, it empowers individuals and organizations to make informed decisions and contribute to sustainability efforts.