

# Capturing User Activity with LLMs

Nathan Chiu

nathanchiu123@gmail.com

Summer 2024

## Introduction

Usually, when you use a help menu in an application, there are some common problems:

- You can't find the answer to your question.
- You find an answer, but it is not very specific.

To address these issues, the program uses GPT-4o-Mini, a version of OpenAI that can detect visual images as well as describe what it sees. This program leverages GPT-4o-Mini by:

- Encoding screenshots into base64. Base64 is a scheme for converting binary data to printable ASCII characters.
- Sending a request to OpenAI for a response

## Example of Base64 Encoding



This image of a blue sky would be the equivalent of this base64 string.

```
iVBORw0KGgoAAAANSUhEUgAAAagAAADwCAYAAADixJgxAAAACXBIWXMAAB7CAAewgFu0HU+AAAgAEIEQVR4Xuy9X4hIZX3H/9+9tRBDhCQoxawCiiCoEAgQKJB4sTITEEAgToIg/gBgQKBBISApKhEPAAETB8UQAVKghCi0GzCIUUnZ0qdPZpPnPd/J3bvdP/M49d/effLJn22vrPfecw39/e1j8D1VVrXWuvuqjalekqoeLdTc3bfWLa3f8kDVUEgwTijS4MbhOjcd8zhHuPPuMvY8PLDR0ldrxn4Il6rUPqAes5ryiuuS3PeWb/p10S+v2tx57sqDsT5U4BwnXrMEXG8tudH55R7OECJGb4KRpR5F8VRkMG8B3eTe7yWjs+SwFgcO2sEzfREujUIZrQDG1RG1jBgt5HiAn5F2RLT9F5RM9kXWdpU5tDrJcEZjDrqz/6owq/c3ecmMhDNbMdCG0PMVnTmNquT/TZUn1/Bk9eoQe2UnJgt5fpHTd9Fr1eMhbnxIY/vsa8A74cUCnsinGBTjrNoVgMXymMcUpLleQ4LlbS0OczKDpGFAL/RyGiEGr33ejHbiRCtub21v9YALUwoX6ZoKzKoFAALZ2l8+y2F1yf9ug6Ru9/W2DRts3NgRCIByPJKFT3HpYZ3T7UbHJwzH5qKNpLrQ6rbT7rfk10Se7Sptv9GeWzTSQ6nVv41OdRPnHU8Wiz63UuO2lajc98hMKFcr78EmNvIMk70wLV3biEw5UncUlrdW+zcyoZ3smknA0BjjFF3RHIB67Sz7sM2rmMyMaGjvUM4wO2K3odfvkemaZ8/IY+stsuAohv7o9YuruOHEF/ppqwKbBrE7msdVFL1C7vflnGzyJ98MYzKd4odMpnexMGx9O0iwtSkM1cjT4M/+4pduRZEFHyj07TWQUgYn5zbIEG83Rkio5DEZaxQxAhQQOUBkx4/cLIqpc8yzFRmhXQ6bXNG6PUK1xnGPqPwBgSsmGiBBRMpyRglOjbJXZN3URAw0CqJlrmnDAYYbdfd0CtrUO36VtZ4IlpB1DWZVmcMweALad00QF7IS8LgrqF1eNc7Mzc4uHB7hr92bHiRveFAqZOda3K/AHkiNnALr2DfBZAqRUdLErutkINEmYmD3klZN4ynuR3NR8U7AZawmyZXvOJzARMdQl2xUg+s17toAu8ABXJZI445yeX2u+kY7dTg5jYxo+FilWnfGn5xQEVAIrljNSck2EnJLKvT4XXdG+RO3ShhToKpX2ToEaOfGUwa5CWg8cj0acBocBiU0RyMQFoAx0eoG8n25cUGkUm/dtVCBqF67JFTwvn8f/8Csvl6c5mKBAAJKfJb80JZrwF1wSI7ZDhenFZXZEgT3kZgw4dC2MD3ebqNcCSlyPROq6ie3vUZodhtcAI+ZL/E5kXfD2v4FsgH+5ubtftTW0cYpdyhElrlnhP133VIGoyQY5EakCPHBIC5bTd11GlpeMBxYgXqjUORl9KeustHh0jcZ0IA6cJpgtECgGyGujDcoSvVueT4oFdR8zXyq3x1goE8UanRRhb5lBZOadVYUduwQ2LKntLmOPsAg6vLQFVwWObZIBcol93E/gnscpE06jUels8aL3sQHcYx8sSB7eF1y1enwe5an<[endoffile]>
```

A function for calculating a triangle's area can be written as follows in Python:

# What are LLMs and Transformers

LLMs (Large Language Models) are coded with knowledge of various text datasets to understand and create human-like text and images. Transformers usually encode or decode various text and image data and can extract a long-range of content efficiently.

Here are some examples of what they can do:

- Translations
- Summarizations
- Concisely answer questions.
- Describe visual data such as screenshots

Here are some examples of LLMs and Transformers

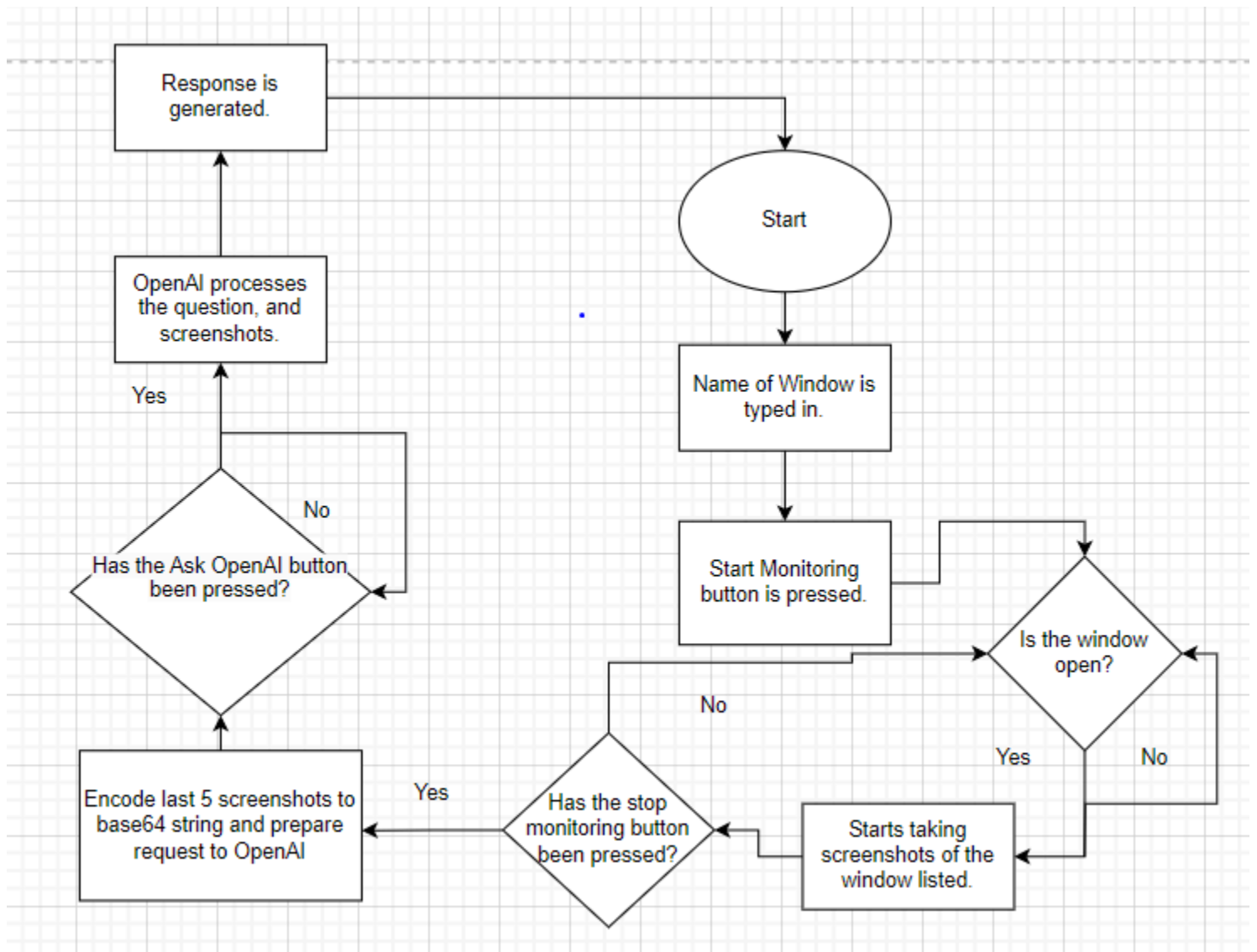
- GPT 4o
- Google's Bert
- Google's ViT
- Microsoft's Swin Transformer

## Overview

The program uses OpenAI to help answer questions posed by users and provides clear answers using context from screenshots and OpenAI's GPT-4o-mini.

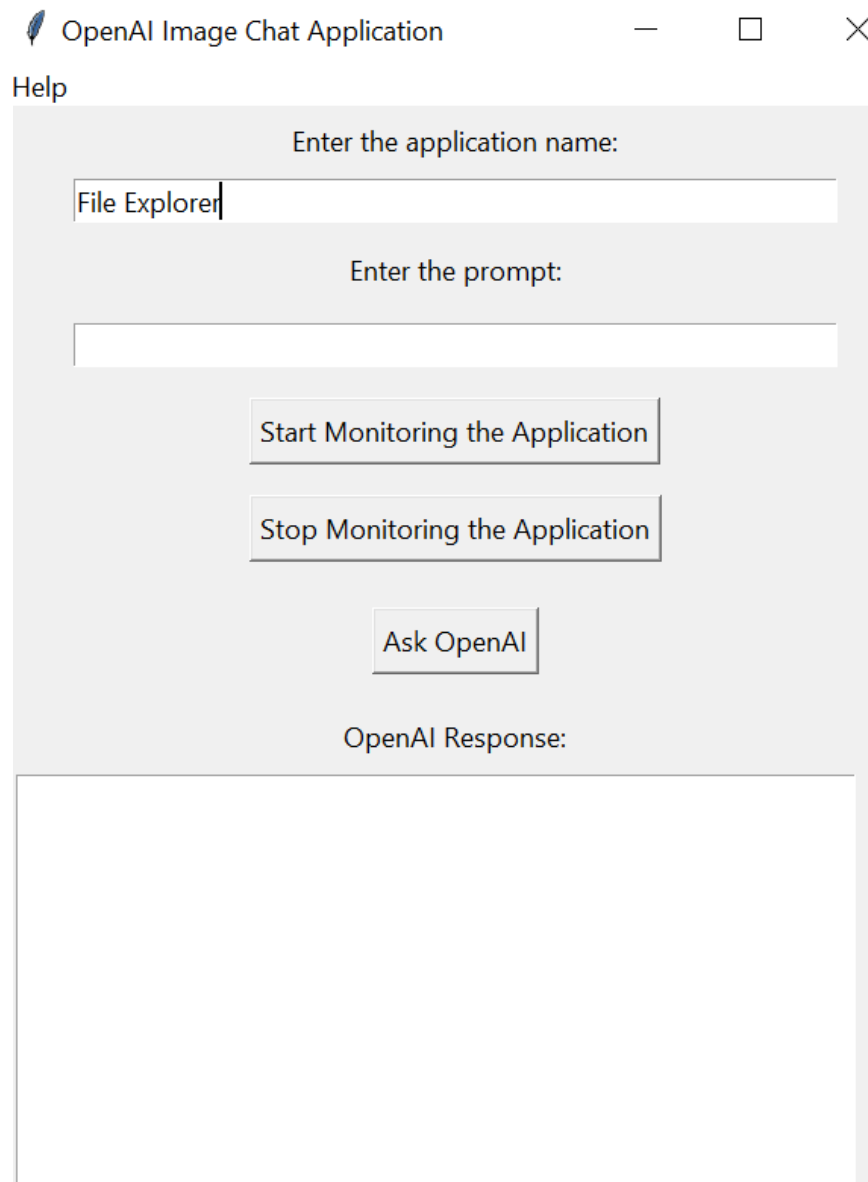
- The program takes screenshots of the user's application at 2-second intervals.
- When the user poses a question, OpenAI scans the last five screenshots taken.
- The program then gives a short and concise answer to the question.

# Data Flow



# How to Use the App






**Step 1:** Run the program and type in the name of the window you want monitored. (Example: File Explorer)



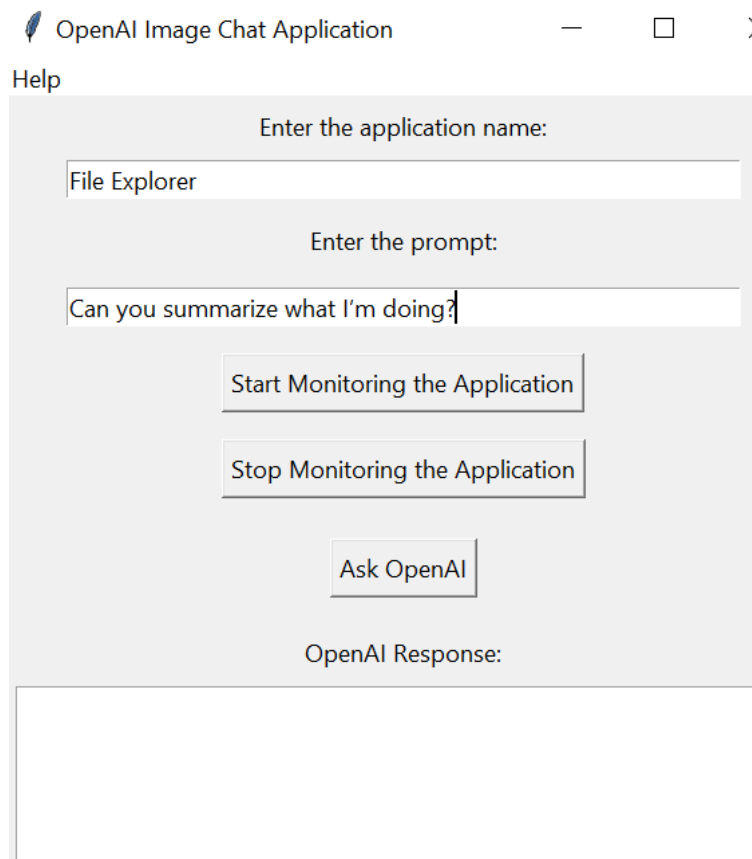
The screenshot shows a window titled "OpenAI Image Chat Application" with standard Windows window controls (minimize, maximize, close). Below the title bar is a "Help" link. The main content area is a light gray panel containing the following elements:

- A label "Enter the application name:" followed by a text input field containing "File Explorer".
- A label "Enter the prompt:" followed by an empty text input field.
- A button labeled "Start Monitoring the Application".
- A button labeled "Stop Monitoring the Application".
- A button labeled "Ask OpenAI".
- A label "OpenAI Response:" followed by a large, empty text area for the response.

**Step 2:** Press the "Start Monitoring" button to take screenshots. Press the "Stop Monitoring" button to stop taking screenshots. The program will keep only the last five screenshots. You should see up to the last 5 screenshots in your files.

 screenshot_20240725104524	7/25/2024 10:45 AM	JPG File
 screenshot_20240725104525	7/25/2024 10:45 AM	JPG File
 screenshot_20240725104528	7/25/2024 10:45 AM	JPG File
 screenshot_20240725104531	7/25/2024 10:45 AM	JPG File
 screenshot_20240725104534	7/25/2024 10:45 AM	JPG File

**Step 3:** Enter a question. (Example: Can you summarize what I'm doing?)



The screenshot shows a window titled "OpenAI Image Chat Application" with standard window controls (minimize, maximize, close). Below the title bar is a "Help" link. The main content area is a light gray box containing the following elements:

- A label "Enter the application name:" followed by a text input field containing "File Explorer".
- A label "Enter the prompt:" followed by a text input field containing "Can you summarize what I'm doing?".
- A button labeled "Start Monitoring the Application".
- A button labeled "Stop Monitoring the Application".
- A button labeled "Ask OpenAI".
- A label "OpenAI Response:" followed by a large, empty text area for the response.



**Step 4:** Press the "Ask OpenAI" button to get a response.

OpenAI Response:

You appear to be using a file manager (likely File Explorer on Windows) to navigate through your Downloads folder. Within this folder, you have several JPG files corresponding to recent screenshots, along with multiple PDF files labeled as "Experiment" and one PNG file. You also seem to be using an application designed for interaction with OpenAI, where you're entering the application name and prompting for details, potentially to monitor and analyze file activity. The second image indicates an issue with accessing one of the screenshot files, as it has either been moved or renamed.

# Demo

Let's use Excel as an example.

## Question:

How can I change this data into a bar graph? What does this bar graph indicate?

Excel cannot provide a specific answer as they do not have a screenshot or image of what the user sees, making it hard for users to find an explanation on their FAQ.

Excel's closest topics are:

- How to create a bar graph.
- How to transfer information from points to a graph.

## ChatGPT Website Response:

1. **Organize Data:**
  - **Column A:** Categories
  - **Column B:** Values
2. **Select Data:**
  - Highlight both columns.
3. **Insert Bar Graph:**
  - Go to **Insert > Bar Chart** > Select your preferred bar chart style.
4. **Customize:**
  - Use **Chart Tools** to adjust titles and labels.

### Interpretation

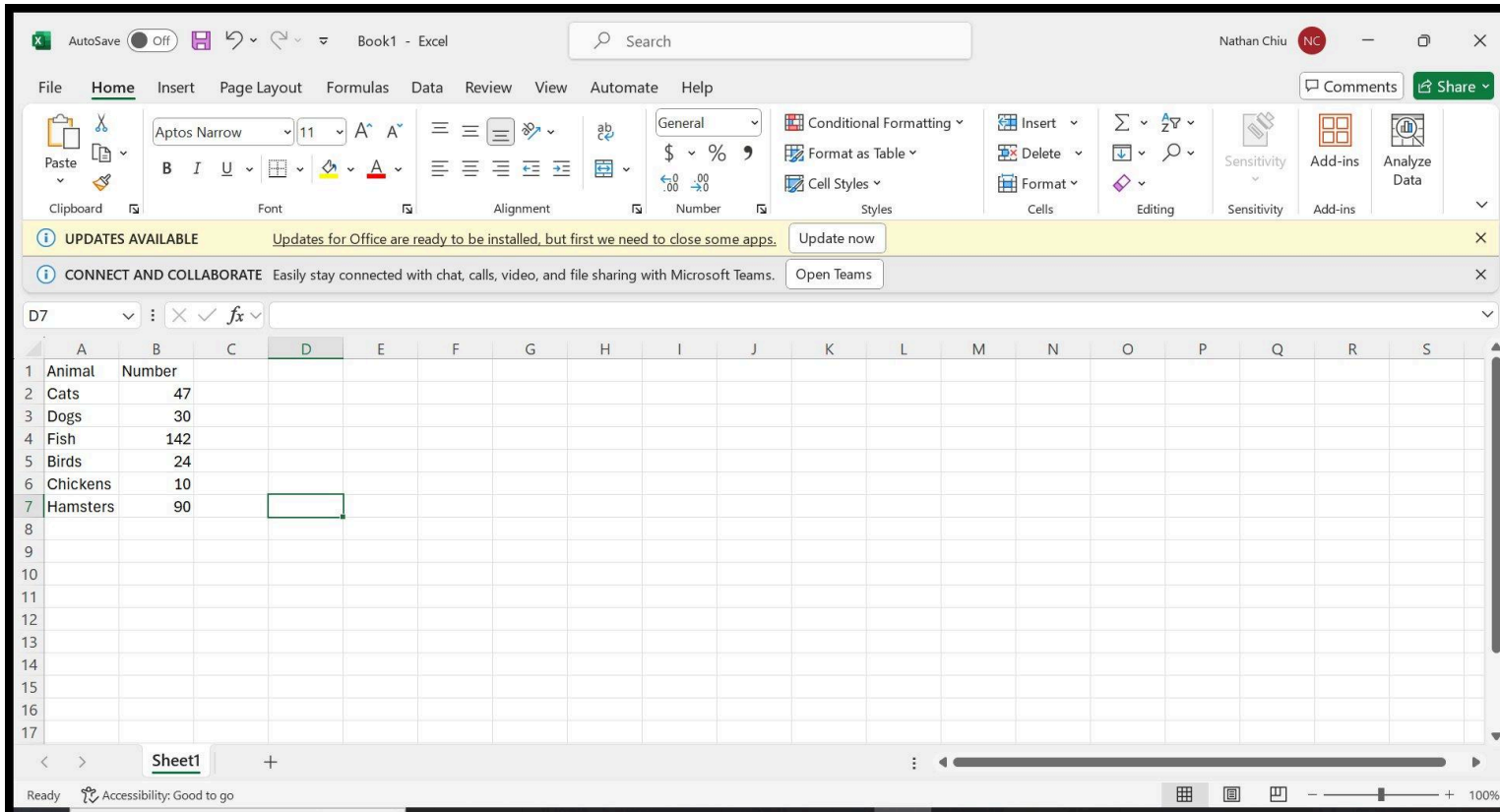
- **Categories:** Represented on the x-axis.
- **Values:** Represented by the length of the bars.

ChatGPT's website does have an answer to this question but doesn't have context. Therefore, it does not give much detail about the answer and is very generic.

Now, I will use my program to monitor my Excel application and then ask GPT-4o-mini the same question.

# Screenshot:

Here is one of the screenshots taken. I input some sample data into excel.



## OpenAI's Response with Context:

To transform the data you've provided into a bar graph using Excel, follow these steps:

### ### Step 1: Organize the Data

Ensure your data is properly organized:

- **Column A**: "Animal"
- **Column B**: "Number"

Your data should look like this:

Animal	Number
Cats	47
Dogs	30
Fish	142
Birds	24
Chickens	10
Hamsters	90

### ### Step 2: Select the Data

1. Click and drag to highlight the cells that contain the data you want to include in the bar graph (cells A1 through B7 in this case).

### ### Step 3: Insert Bar Graph

1. Navigate to the **Insert** tab on the Excel ribbon.
2. In the Charts group, look for either the **Bar Chart** or **Column Chart** icon.
3. Click on the icon to reveal a dropdown menu that lists several chart options.
4. Select a **Clustered Bar** or **Clustered Column** chart (depending on your preference).

### ### Step 4: Customize the Chart

1. Once the chart appears, you can customize it:
  - Click on the chart title to rename it (e.g., "Number of Different Animals").
  - Adjust the axis titles by clicking on the chart and selecting the **Chart Elements** button (the plus sign next to the chart).
  - Add data labels for better visibility if desired.

#### #### Step 5: Analyze the Bar Graph

The bar graph will provide a clear visual representation of the number of animals by type. Here's what you can deduce from it:

- **Comparison of Quantities**: The heights of the bars indicate the relative quantities of each animal. For instance, "Fish" has the highest number, followed by "Hamsters".
- **Insights**: You can quickly see which animals are most and least prevalent. For instance, "Chickens" have the lowest representation, while "Fish" dominate the count.
- **Patterns or Trends**: You may observe patterns, like which types of pets or animals people tend to prefer based on the numbers.

#### #### Step 6: Save Your Work

After customizing your graph, don't forget to save your Excel file to preserve your data and chart.

Following these steps allows you to effectively create a bar graph that visually presents your data, making it easier to interpret and analyze.

## Code Explanation

This part of the code imports all of the modules, allowing many of the functions to be possible.

```
import tkinter as tk
from tkinter import messagebox, Menu, Text, Scrollbar
from PIL import ImageGrab
import base64
import requests
import os
import subprocess
import pyautogui
from win32 import win32gui
import time
import json
import threading
```

This function gives the program the pathway to the user's download folder.

```
def get_downloads_folder():
    return os.path.join(os.path.expanduser("~"), "Downloads")
```

**This function gathers the dimension of the window the user typed in. It takes a screenshot and saves it to the “Downloads” folder**

```
def screenshot():
    current_time = time.strftime("%Y%m%d%H%M%S")
    hwnd = win32gui.GetForegroundWindow()
    if hwnd:
        win32gui.SetForegroundWindow(hwnd)
        time.sleep(1)
        rect = win32gui.GetWindowRect(hwnd)
        im = ImageGrab.grab(bbox=(rect[0], rect[1], rect[2], rect[3]))
        screenshot_path = os.path.join(get_downloads_folder(), f"screenshot_{current_time}.jpg")
        im.save(screenshot_path, "JPEG", quality=85)
        print(f"Screenshot saved at: {screenshot_path}")
        return screenshot_path
    else:
        print('Active window not found!')
        return None
```

**This function is to monitor the window and take screenshots at a two second interval.**

```
def monitor_application(interval=2):
    global monitoring
    while monitoring:
        screenshot_path = screenshot()
        if screenshot_path:
            screenshots_lock.acquire()
            screenshots.append(screenshot_path)
            if len(screenshots) > 5:
                oldest_screenshot = screenshots.pop(0)
                if os.path.exists(oldest_screenshot):
                    os.remove(oldest_screenshot)
                    print(f"Deleted oldest screenshot: {oldest_screenshot}")
            screenshots_lock.release()
        time.sleep(interval)
```



**This function locates the window typed in the box, so it can begin monitoring it.**

```
def start_monitoring():  
    global monitoring  
    app_name = entry_app_name.get()  
    window_title = app_name  
  
    if not app_name:  
        messagebox.showerror("Error", "Please enter the application name.")  
        return  
  
    if not is_application_running(window_title):  
        open_application(app_name)  
  
    monitoring = True  
    monitor_thread = threading.Thread(target=monitor_application)  
    monitor_thread.start()
```

**This function stops monitoring the window.**

```
def stop_monitoring():  
    global monitoring  
    monitoring = False
```

**This function allows the user to close the program.**

```
def quit_app():  
    stop_monitoring()  
    root.destroy()
```

**This function encodes the screenshots to base64.**

```
def encode_image(image_path):  
    with open(image_path, "rb") as image_file:  
        return base64.b64encode(image_file.read()).decode('utf-8')
```

**This function checks if the application is running.**

```
def is_application_running(window_title):  
    hwnd = win32gui.FindWindow(None, window_title)  
    return hwnd != 0
```

**This function causes the window to come to the main screen if it is hidden behind another window.**

```
def open_application(app_name):  
    try:  
        subprocess.Popen(app_name)  
        time.sleep(5)  
    except Exception as e:  
        print(f"Error opening application: {e}")
```

**This function deletes all screenshots taken after the “Ask OpenAI” button is pressed.**

```
def delete_all_screenshots():  
    global screenshots_lock, screenshots  
    screenshots_lock.acquire()  
    for screenshot_path in screenshots:  
        if os.path.exists(screenshot_path):  
            os.remove(screenshot_path)  
    screenshots.clear()  
    screenshots_lock.release()
```

**This function allows OpenAI to gather the screenshots. It provides the prompt to GPT-4o-mini and makes a request for a response.**

```
def ask_openai():
    app_name = entry_app_name.get()
    window_title = app_name
    prompt = entry_prompt.get()

    if not app_name:
        messagebox.showerror("Error", "Please enter the application name.")
        return

    if not is_application_running(window_title):
        open_application(app_name)

    if not screenshots:
        messagebox.showerror("Error", "No screenshots captured.")
        return

    try:
        base64_images = [encode_image(path) for path in screenshots]
        image_messages = [
            {"type": "image_url", "image_url": {"url": f"data:image/jpeg;base64,{image}"}}
            for image in base64_images
        ]
        headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {api_key}"
        }
        payload = {
            "model": "gpt-4o-mini",
            "messages": [
                {
                    "role": "user",
                    "content": [
                        {
                            "type": "text",
                            "text": prompt
                        }
                    ] + image_messages
                }
            ],
            "max_tokens": 500
        }

        # Make request to OpenAI API
        response = requests.post(
            "https://api.openai.com/v1/chat/completions",
            headers=headers,
            json=payload
        )
```

**This function gives a response to the prompt.**

```
if response.status_code == 200:
    response_data = response.json()
    summary = response_data['choices'][0]['message']['content']
    text_output.config(state=tk.NORMAL)
    text_output.delete(1.0, tk.END)
    text_output.insert(tk.END, summary)
    text_output.config(state=tk.DISABLED)
    delete_all_screenshots()
except Exception as e:
    messagebox.showerror("Error", f"An error occurred: {e}")
```

**This function creates an about tab that offers help to the user if they do not know how to use the program.**

```
def show_about():
    messagebox.showinfo("About", "Make sure you have the application open, not minimized or else this app will not be able to scan for it.")
```

**The first part locks the amount of screenshots when the Stop Monitoring button is pressed. The second part creates the GUI and messagebox.**

```
screenshots = []
screenshots_lock = threading.Lock()
monitoring = False

root = tk.Tk()
root.title("OpenAI Image Chat Application")
root.geometry("600x400")
```

**This function creates the window name, the “About” section, and “Start Monitoring” and “Stop Monitoring” buttons, and response section.**

```
menu_bar = Menu(root)
root.config(menu=menu_bar)

help_menu = Menu(menu_bar, tearoff=0)
help_menu.add_command(label="Close App", command=quit_app)
help_menu.add_command(label="Help", command=show_about)
menu_bar.add_cascade(label="Help", menu=help_menu)

label_app_name = tk.Label(root, text="Enter the application name:")
label_app_name.pack(pady=5)

entry_app_name = tk.Entry(root, width=50)
entry_app_name.pack(pady=5)

label_prompt = tk.Label(root, text="Enter the prompt:")
label_prompt.pack(pady=10)
entry_prompt = tk.Entry(root, width=50)
entry_prompt.pack(pady=10)

monitor_button = tk.Button(root, text="Start Monitoring the Application", command=start_monitoring)
monitor_button.pack(pady=10)

stop_button = tk.Button(root, text="Stop Monitoring the Application", command=stop_monitoring)
stop_button.pack(pady=10)

ask_button = tk.Button(root, text="Ask OpenAI", command=ask_openai)
ask_button.pack(pady=20)

label_output = tk.Label(root, text="OpenAI Response:")
label_output.pack(pady=5)

scrollbar = Scrollbar(root)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

text_output = Text(root, wrap=tk.WORD, yscrollcommand=scrollbar.set, height=20, state=tk.DISABLED)
text_output.pack(padx=10, pady=5, fill=tk.BOTH, expand=True)

scrollbar.config(command=text_output.yview)

root.mainloop()
```