Data Visualisation and Analysis Application

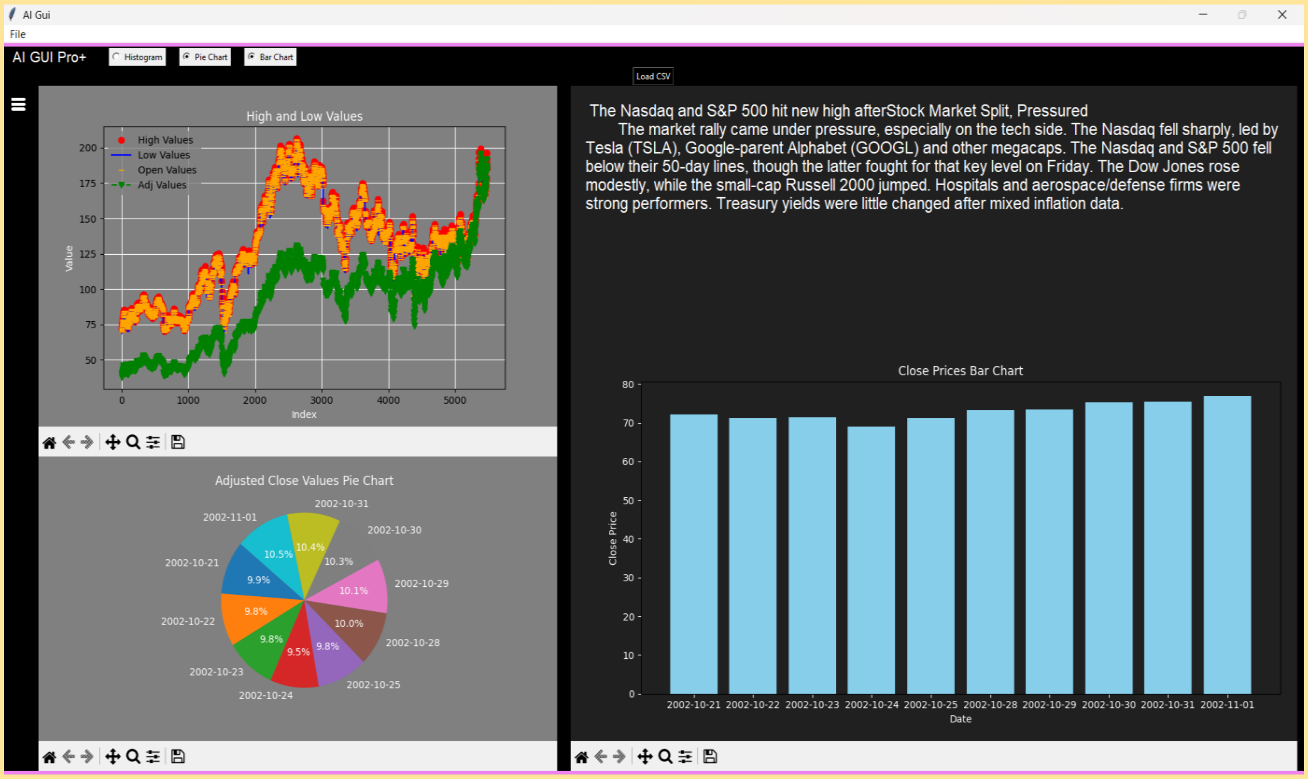
Contents:

Table of Contents

Introduction:

This Report demonstrates Data visualisation and analysis and furious functionalities.The project, aimed to demonstrate a basic Gui design. Implementing UI interface, using Ai tools, Radio buttons, buttons, Text boxes and frames, with graph plots and data table generation.

There is also a csv button handling data. This project leverages Python libraries to create versatile tools for data analysis and visualisation.



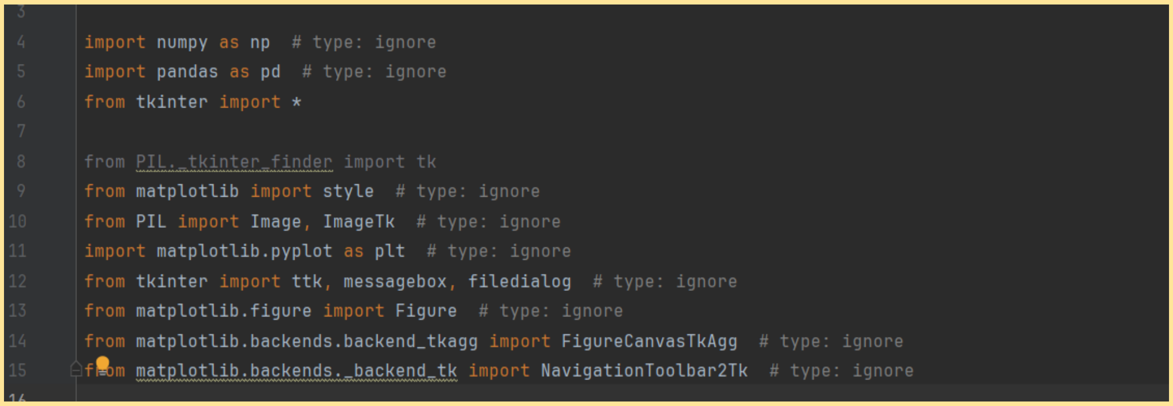
The Nasdaq and S&P 500 hit new high afterStock Market Split, Pressured
The market rally came under pressure, especially on the tech side. The Nasdaq fell sharply, led by Tesla (TSLA), Google-parent Alphabet (GOOGL) and other megacaps. The Nasdaq and S&P 500 fell below their 50-day lines, though the latter fought for that key level on Friday. The Dow Jones rose modestly, while the small-cap Russell 2000 jumped. Hospitals and aerospace/defense firms were strong performers. Treasury yields were little changed after mixed inflation data.

Technologies Used

- **Python**: The application is developed using Python as its primary programming language.
- **Tkinter**: The standard Python interface to the Tk GUI toolkit is used to create the application's graphical user interface (GUI).
- **Pandas**: provides powerful data manipulation and analysis capabilities for CSV files.

- **Matplotlib**: This is a Python library that enables static, animated, and interactive visualisations. It is used here to generate plots such as histograms, pie charts, and bar charts.
- **Pillow (PIL)**: Image processing tasks are accomplished with Pillow (PIL) by resizing and displaying images visually.
- **Intellij(IDE)**: integrated development environment (IDE) developed by JetBrains, including intelligent code completion, refactoring tools, and deep code analysis.
- **CodiumAI**: Excellent for improving Python code, providing robust feedback, and ensuring code quality.
- **Open AI Codex(ChatGPT)**: Great for understanding and solving complex coding problems through human-like conversation.

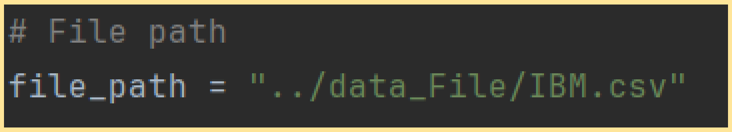
Imported Libraries:



```
import numpy as np  # type: ignore
import pandas as pd  # type: ignore
from tkinter import *

from PIL._tkinter_finder import tk
from matplotlib import style  # type: ignore
from PIL import Image, ImageTk  # type: ignore
import matplotlib.pyplot as plt  # type: ignore
from tkinter import ttk, messagebox, filedialog  # type: ignore
from matplotlib.figure import Figure  # type: ignore
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg  # type: ignore
from matplotlib.backends._backend_tk import NavigationToolbar2Tk  # type: ignore
```

Dataset

The application implements a financial dataset on the stock market using ‘File_path’ `../data_File/IBM.csv`. within the code to generate visual plots on the Gui through a csv file.

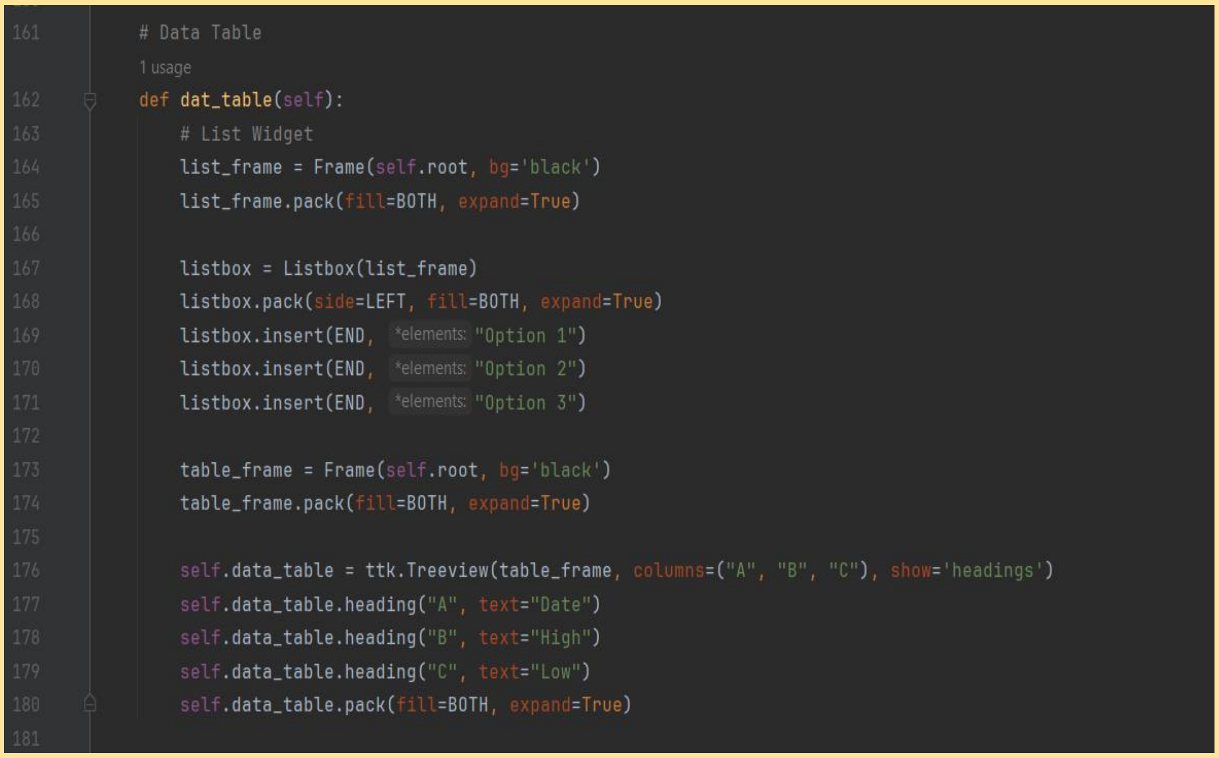


```
# File path
file_path = "../data_File/IBM.csv"
```

While the application may do this it also takes a second variable, ‘File_path’ generated in ‘open_file_dialog’ which is called through a button to load a data_table holding content such as:

- **`High`**: The highest price of the stock during a given period.
- **`Low`**: The lowest price of the stock during a given period.
- **`Open`**: The opening price of the stock.
- **`Adj Close`**: The adjusted closing price of the stock.
- **Date**: The release Date of the occurring information.

It is to be assumed the dataset consists of reliable historical IBM stock data, which provides valuable financial information. Yahoo Finance was used as the source of the data.



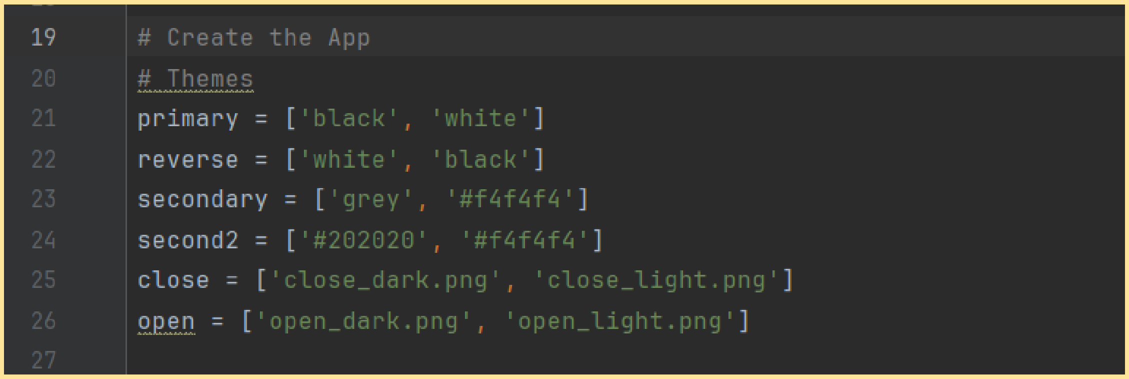
```
    # Data Table
    1 usage
    def dat_table(self):
        # List Widget
        list_frame = Frame(self.root, bg='black')
        list_frame.pack(fill=BOTH, expand=True)

        listbox = Listbox(list_frame)
        listbox.pack(side=LEFT, fill=BOTH, expand=True)
        listbox.insert(END, *elements: "Option 1")
        listbox.insert(END, *elements: "Option 2")
        listbox.insert(END, *elements: "Option 3")

        table_frame = Frame(self.root, bg='black')
        table_frame.pack(fill=BOTH, expand=True)

        self.data_table = ttk.Treeview(table_frame, columns=("A", "B", "C"), show='headings')
        self.data_table.heading("A", text="Date")
        self.data_table.heading("B", text="High")
        self.data_table.heading("C", text="Low")
        self.data_table.pack(fill=BOTH, expand=True)

```

Application Features

Code Architecture

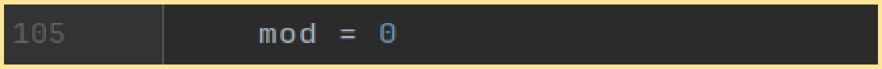
1. Global Variables and Function

- **Themes**: Lists defining colour schemes for different GUI elements such as ‘light’ and ‘Dark’ mode.



```python
# Create the App
# Themes
primary = ['black', 'white']
reverse = ['white', 'black']
secondary = ['grey', '#f4f4f4']
second2 = ['#202020', '#f4f4f4']
close = ['close_dark.png', 'close_light.png']
open = ['open_dark.png', 'open_light.png']
```

Initiated to code through ‘mod’ variable:



```python
105        mod = 0
```

- **File Path**: Which loads CSV files containing financial data, as recently discussed.

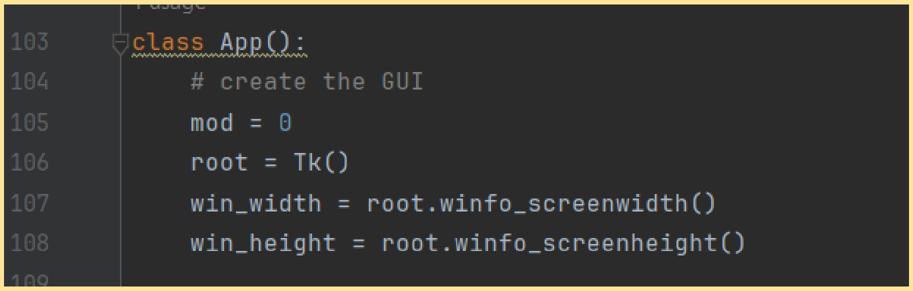
2. Data Visualisation Functions

- **Load_data3**: Loads data and creates a histogram plot showing high, low, open, and adjusted close values.
- **Load_data4**: Creates a pie chart of the top 10 adjusted close values.
- **Load_data5**: Generates a bar chart for the close prices.

3. Application Class

- **App**: The main class responsible for setting up the GUI and handling user interactions.

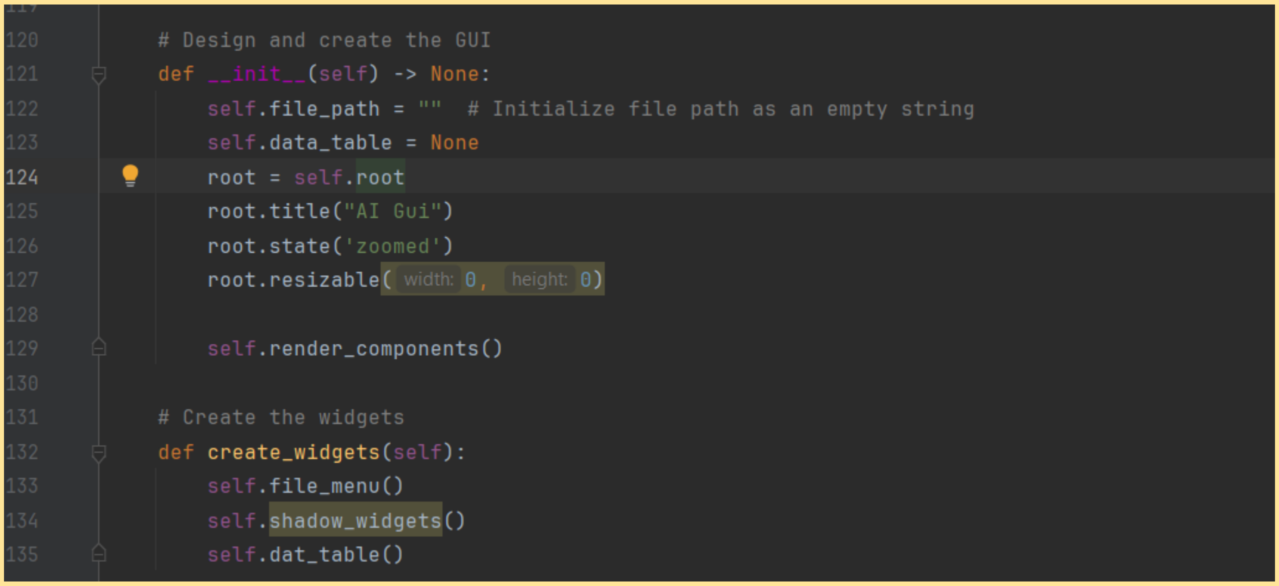
  It's responsible for launching applications,running main-loop and handling feature components.



```
class App():
    # create the GUI
    mod = 0
    root = Tk()
    win_width = root.winfo_screenwidth()
    win_height = root.winfo_screenheight()
```

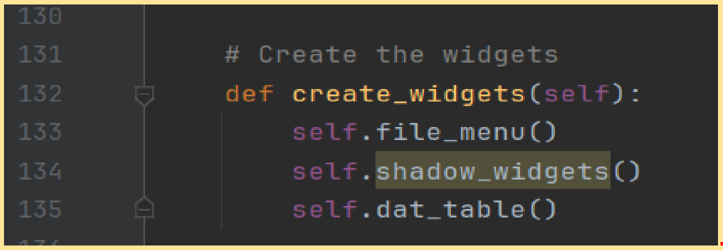
Below we will see some of the class application components:

- **Initialization and Configuration**: Sets up the main window and initialises Gui components.



```
    # Design and create the GUI
    def __init__(self) -> None:
        self.file_path = ""  # Initialize file path as an empty string
        self.data_table = None
        root = self.root
        root.title("AI Gui")
        root.state('zoomed')
        root.resizable( width: 0, height: 0)

        self.render_components()

    # Create the widgets
    def create_widgets(self):
        self.file_menu()
        self.shadow_widgets()
        self.dat_table()
```

- **Widgets Creation**: Includes methods for creating menus, buttons, and data tables, These widgets are essential for user interaction and data display.



```
    # Create the widgets
    def create_widgets(self):
        self.file_menu()
        self.shadow_widgets()
        self.dat_table()
```

- **File Handling**:Handles operations related to file management, such as Methods to open CSV files and display data in a new window.

  See Data Handling for further Information.

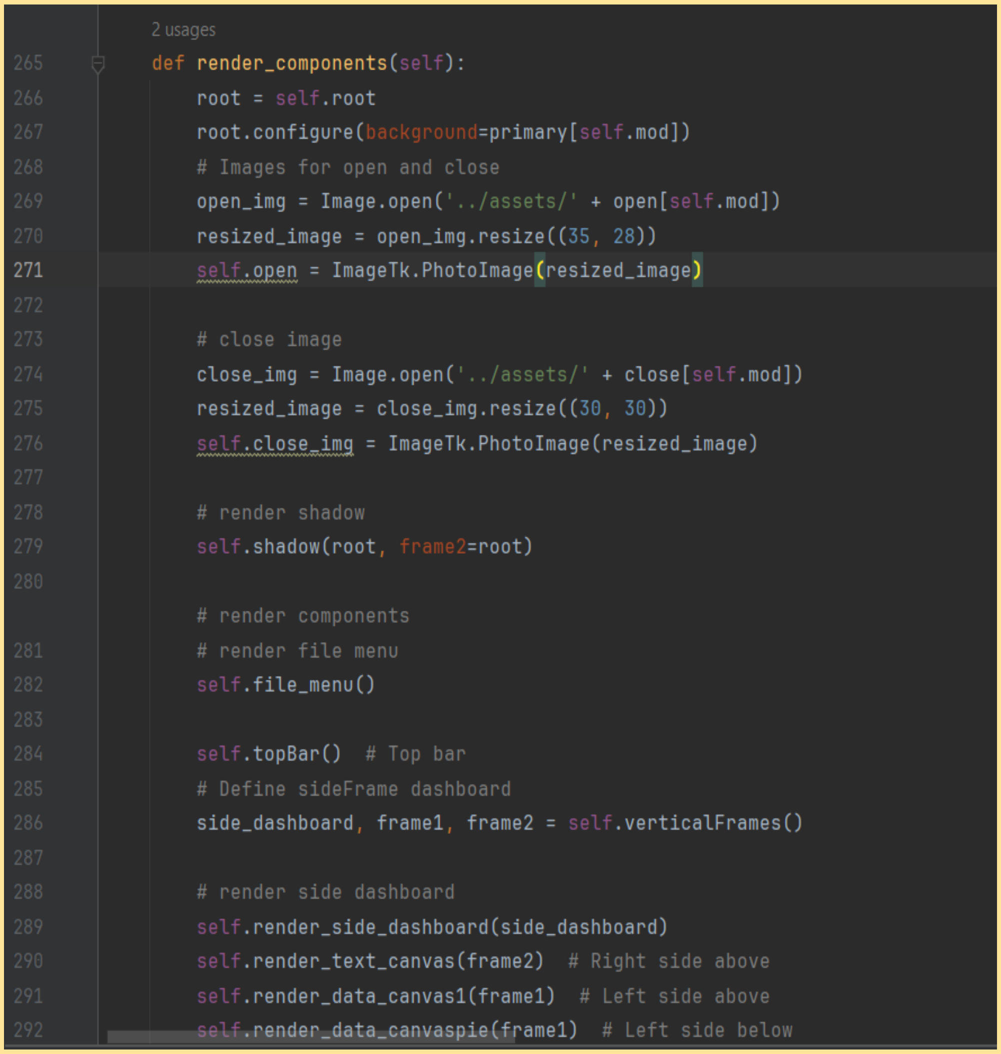
- **Plotting**: Methods to open new windows and render plots using Matplotlib.

  See Data Handling, plot rendering for further Information.

- **UI Customization**: Includes methods for changing themes and updating the GUI layout.

  For further information on themes see Theming.

Components for Gui have been initialised in one function which controls layout placement:

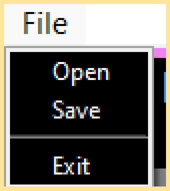


```
    2 usages
    def render_components(self):
        root = self.root
        root.configure(background=primary[self.mod])
        # Images for open and close
        open_img = Image.open('../assets/' + open[self.mod])
        resized_image = open_img.resize((35, 28))
        self.open = ImageTk.PhotoImage(resized_image)

        # close image
        close_img = Image.open('../assets/' + close[self.mod])
        resized_image = close_img.resize((30, 30))
        self.close_img = ImageTk.PhotoImage(resized_image)

        # render shadow
        self.shadow(root, frame2=root)

        # render components
        # render file menu
        self.file_menu()

        self.topBar()  # Top bar
        # Define sideFrame dashboard
        side_dashboard, frame1, frame2 = self.verticalFrames()

        # render side dashboard
        self.render_side_dashboard(side_dashboard)
        self.render_text_canvas(frame2)  # Right side above
        self.render_data_canvas1(frame1)  # Left side above
        self.render_data_canvaspie(frame1)  # Left side below
```

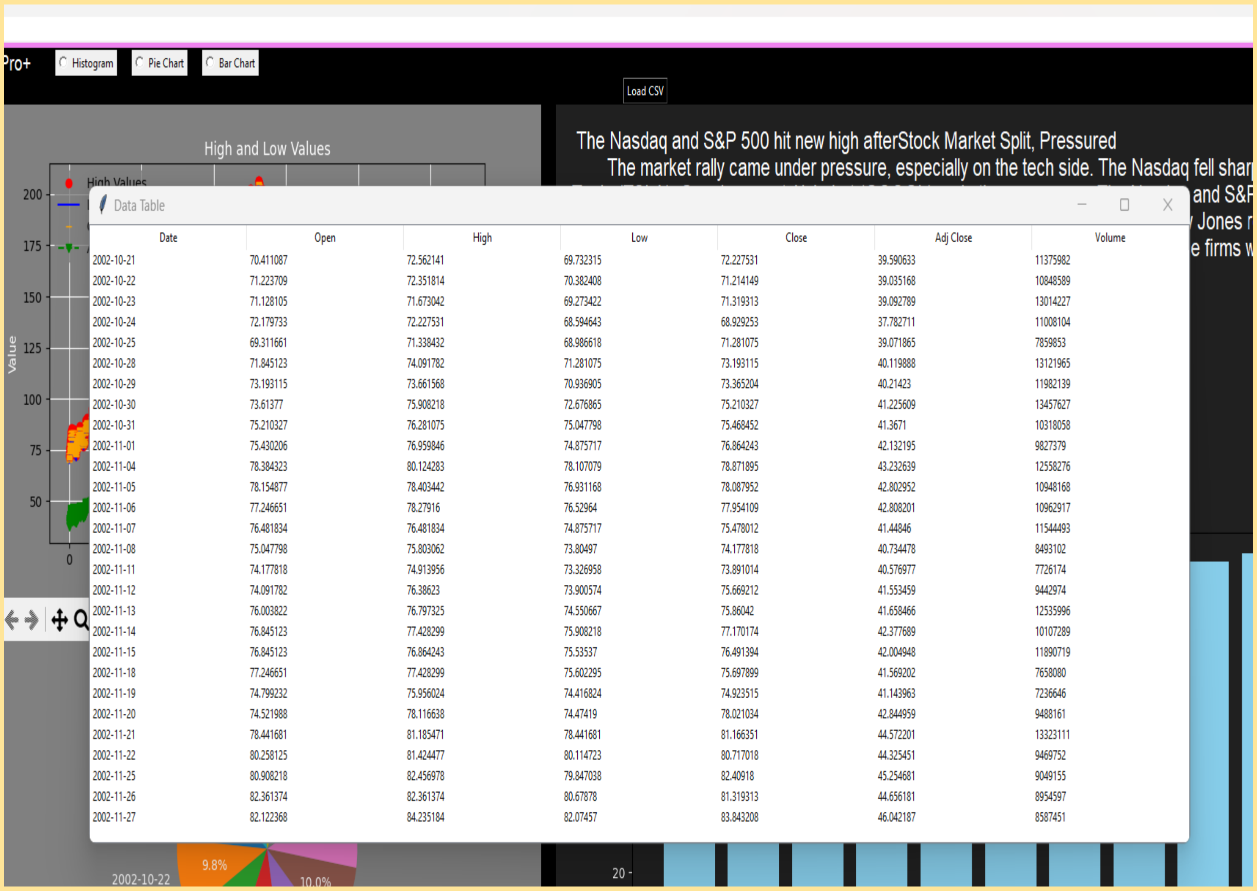
Functionality

GUI Components

- **Menu Bar**: labelled File. Provides options to open, save, and exit files.



- **Data Table**: Displays data from the CSV file, loaded from the load csv button, in a tabular format to a separate window.

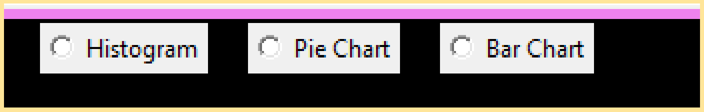


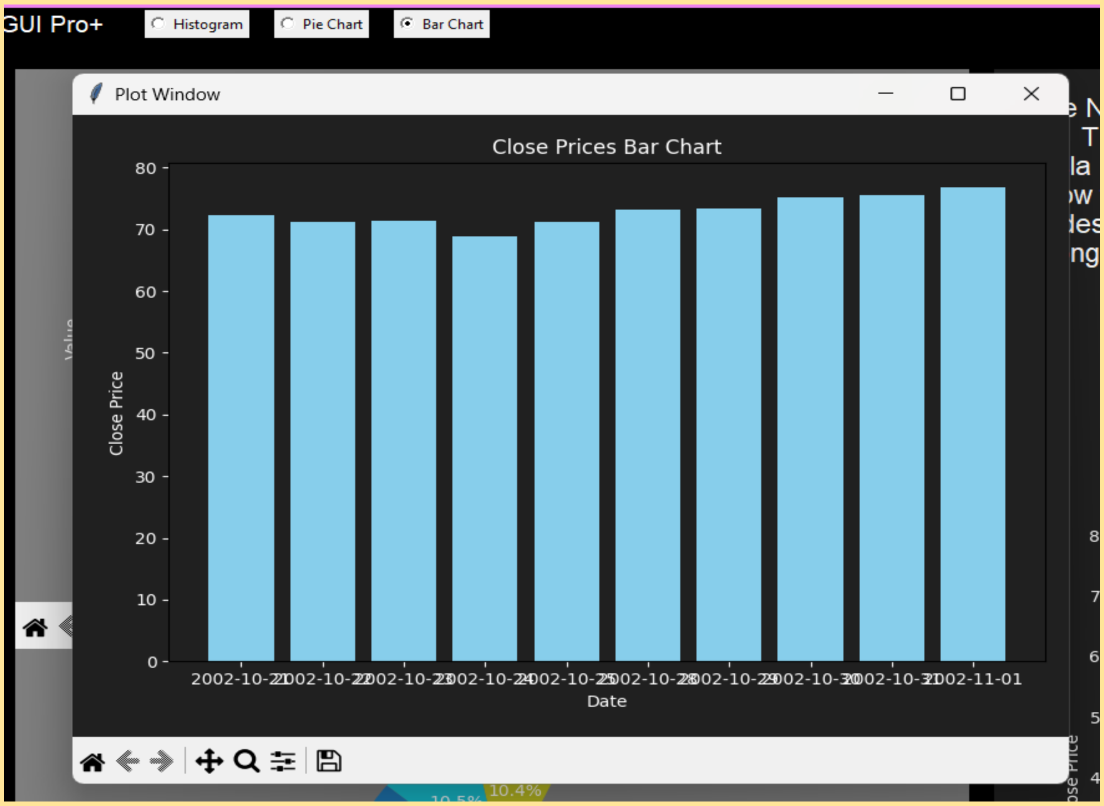
The Nasdaq and S&P 500 hit new high afterStock Market Split, Pressured
The market rally came under pressure, especially on the tech side. The Nasdaq fell shar

Date	Open	High	Low	Close	Adj Close	Volume
2002-10-21	70.411087	72.562141	69.732315	72.227531	39.590633	11375982
2002-10-22	71.223709	72.351814	70.382408	71.214149	39.035168	10848589
2002-10-23	71.128105	71.673042	69.273422	71.319313	39.092789	13014227
2002-10-24	72.179733	72.227531	68.594643	68.929253	37.782711	11008104
2002-10-25	69.311661	71.338432	68.986618	71.281075	39.071865	7859853
2002-10-28	71.845123	74.091782	71.281075	73.193115	40.119888	13121965
2002-10-29	73.193115	73.661568	70.936905	73.365204	40.21423	11982139
2002-10-30	73.61377	75.908218	72.676865	75.210327	41.225609	13457627
2002-10-31	75.210327	76.281075	75.047798	75.468452	41.3671	10318058
2002-11-01	75.430206	76.959846	74.875717	76.864243	42.132195	9827379
2002-11-04	78.384323	80.124283	78.107079	78.871895	43.232639	12558276
2002-11-05	78.154877	78.403442	76.931168	78.087952	42.802952	10948168
2002-11-06	77.246651	78.27916	76.52964	77.954109	42.808201	10962917
2002-11-07	76.481834	76.481834	74.875717	75.478012	41.44846	11544493
2002-11-08	75.047798	75.803062	73.80497	74.177818	40.734478	8493102
2002-11-11	74.177818	74.913956	73.326958	73.891014	40.576977	7726174
2002-11-12	74.091782	76.38623	73.900574	75.669212	41.553459	9442974
2002-11-13	76.003822	76.797325	74.550667	75.86042	41.658466	12535996
2002-11-14	76.845123	77.428299	75.908218	77.170174	42.377689	10107289
2002-11-15	76.845123	76.864243	75.53537	76.491394	42.004948	11890719
2002-11-18	77.246651	77.428299	75.602295	75.697899	41.569202	7658080
2002-11-19	74.799232	75.956024	74.416824	74.923515	41.143963	7236646
2002-11-20	74.521988	78.116638	74.47419	78.021034	42.844959	9488161
2002-11-21	78.441681	81.185471	78.441681	81.166351	44.572201	13323111
2002-11-22	80.258125	81.424477	80.114723	80.717018	44.325451	9469752
2002-11-25	80.908218	82.456978	79.847038	82.40918	45.254681	9049155
2002-11-26	82.361374	82.361374	80.67878	81.319313	44.656181	8954597
2002-11-27	82.122368	84.235184	82.07457	83.843208	46.042187	8587451

- **Plotting Options**: Radio buttons to select between histogram, pie chart, and bar chart.

  Radio buttons consist of 3 options: these buttons will display the corresponding plot in a separate window allowing to individualise plots and interact separately.

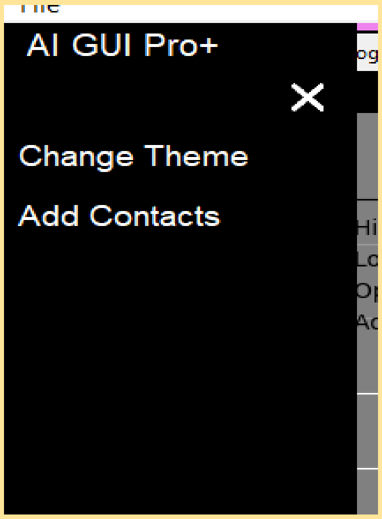




- **CSV Button**: Allows users to load and display CSV files from File_path selected.

  Issue would be that it only loads to table not updating entire GUI information and plots.

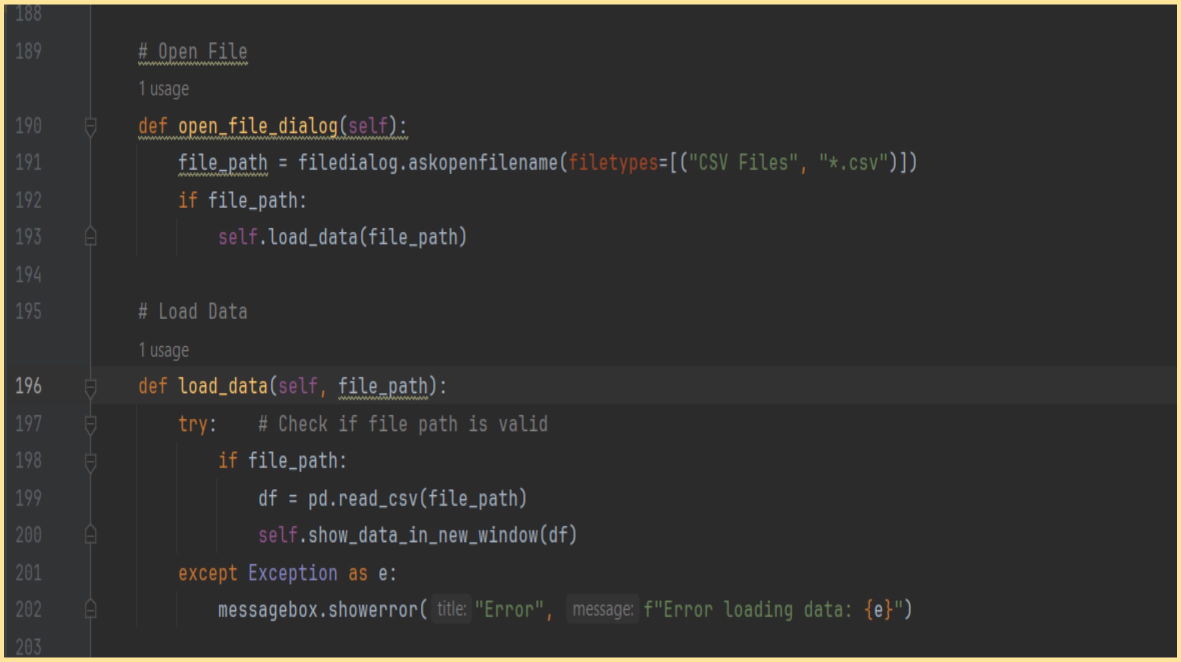
- **Side Dashboard**: Provides additional options such as theme change and contacts.



Data Handling

- **CSV Loading**: Reads data from a CSV file and updates the data table.

  This is where the button takes a file using ‘File_path’ and updates the data table accordingly.

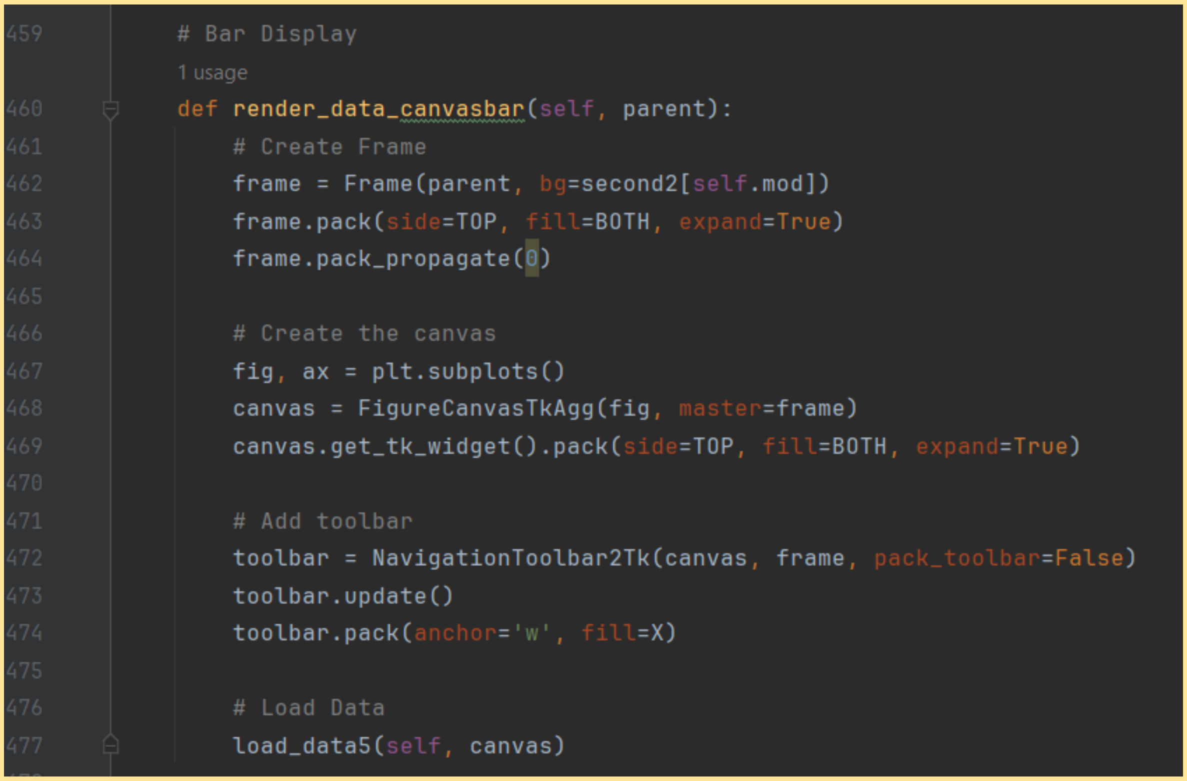


```
# Open File
1 usage
def open_file_dialog(self):
    file_path = filedialog.askopenfilename(filetypes=[("CSV Files", "*.csv")])
    if file_path:
        self.load_data(file_path)

# Load Data
1 usage
def load_data(self, file_path):
    try:    # Check if file path is valid
        if file_path:
            df = pd.read_csv(file_path)
            self.show_data_in_new_window(df)
    except Exception as e:
        messagebox.showerror( title: "Error",  message: f"Error loading data: {e}")
```

- **Plot Rendering**: Uses Matplotlib to create and display various types of charts based on the loaded data.

This is where we will take the load_data(3,4 and 5) and print it. This is used to create the picture-like upload to Gui as well as creating a good separation between data and rendering to the screen.

Example of one rendering plot Function:



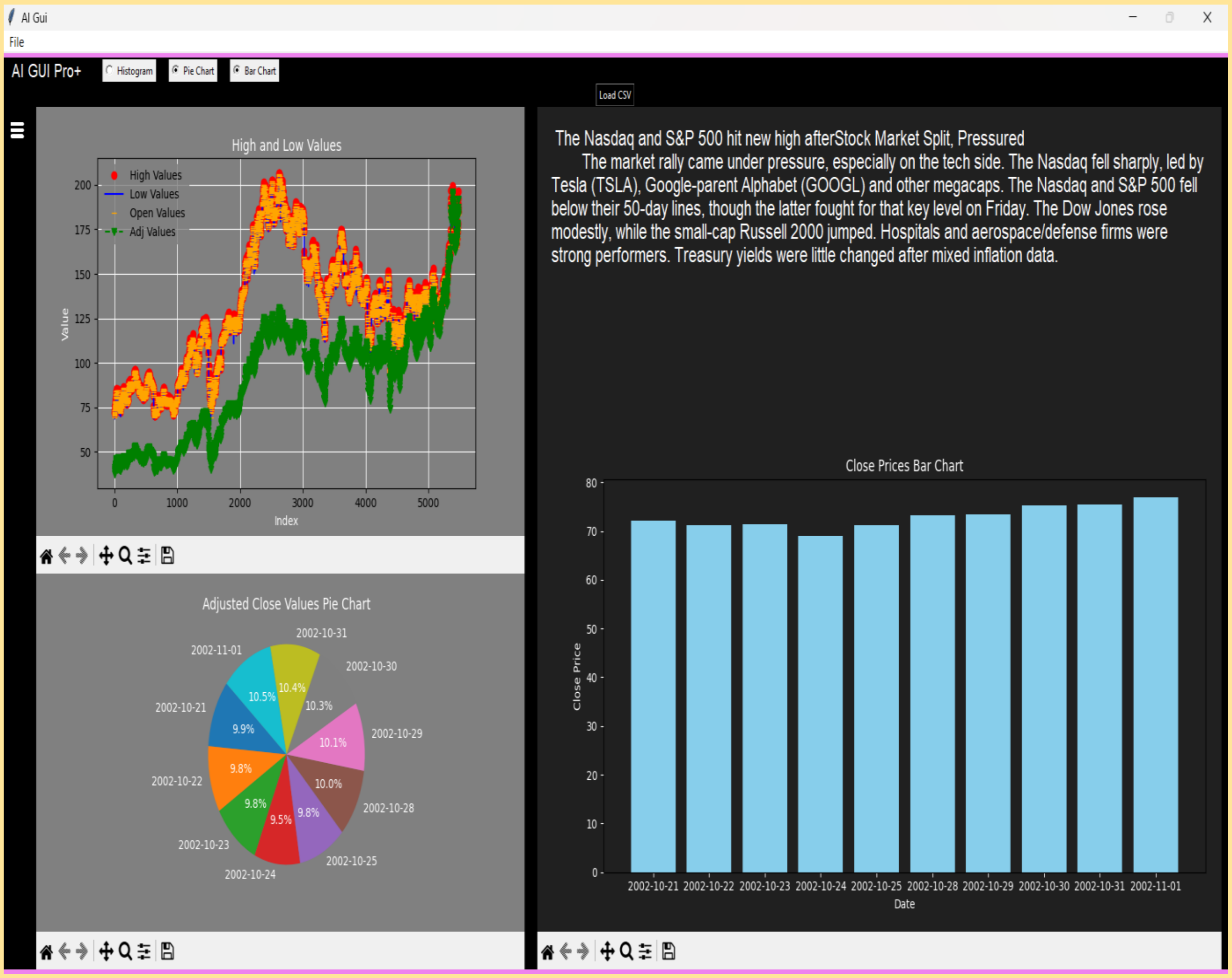
```
    # Bar Display
    1 usage
    def render_data_canvasbar(self, parent):
        # Create Frame
        frame = Frame(parent, bg=second2[self.mod])
        frame.pack(side=TOP, fill=BOTH, expand=True)
        frame.pack_propagate(0)

        # Create the canvas
        fig, ax = plt.subplots()
        canvas = FigureCanvasTkAgg(fig, master=frame)
        canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True)

        # Add toolbar
        toolbar = NavigationToolbar2Tk(canvas, frame, pack_toolbar=False)
        toolbar.update()
        toolbar.pack(anchor='w', fill=X)

        # Load Data
        load_data5(self, canvas)
```

User Interface

The application is designed to maximise user experience, showing key features which include:

- **Responsive Design**: The layout is set to users screen width and height displaying full functionalities of Gui design, meaning the Gui is not adjustable to keep contents from overlapping and causing un-interactive functions.This provides an intuitive navigation experience for the user.

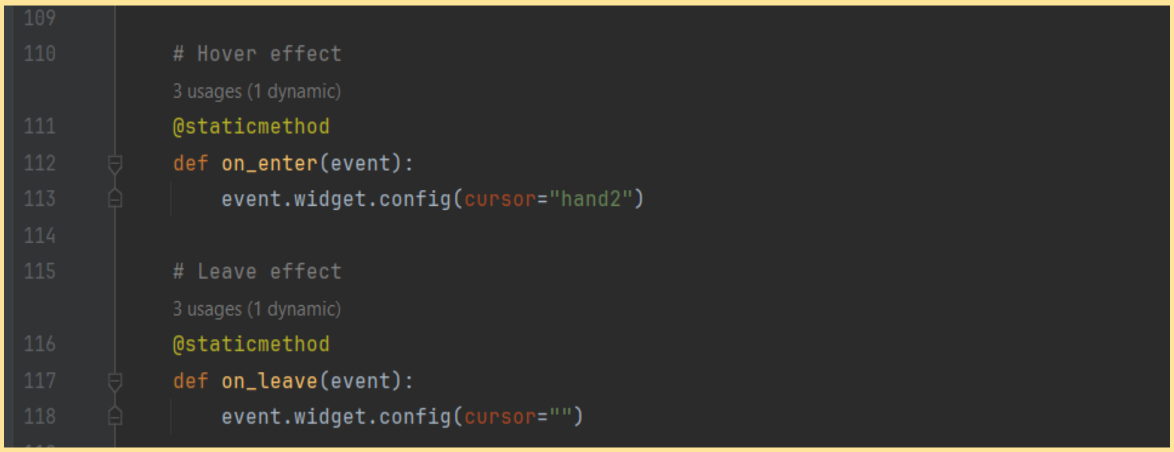
- **Interactive Elements**: Hover effects on buttons and options enhance usability.



The Nasdaq and S&P 500 hit new high afterStock Market Split, Pressured

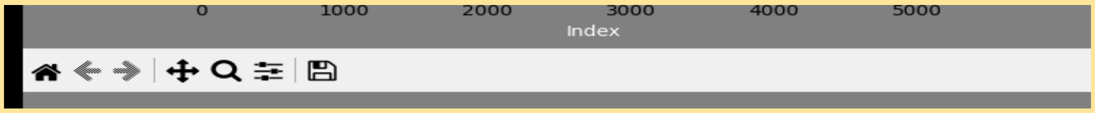
The market rally came under pressure, especially on the tech side. The Nasdaq fell sharply, led by Tesla (TSLA), Google-parent Alphabet (GOOGL) and other megacaps. The Nasdaq and S&P 500 fell below their 50-day lines, though the latter fought for that key level on Friday. The Dow Jones rose modestly, while the small-cap Russell 2000 jumped. Hospitals and aerospace/defense firms were strong performers. Treasury yields were little changed after mixed inflation data.

1. Hover effects include turning the mouse arrow to a hand when selecting a button

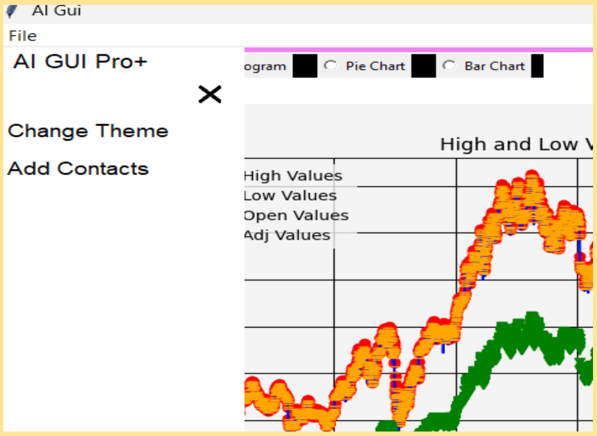


```
    # Hover effect
    3 usages (1 dynamic)
    @staticmethod
    def on_enter(event):
        event.widget.config(cursor="hand2")

    # Leave effect
    3 usages (1 dynamic)
    @staticmethod
    def on_leave(event):
        event.widget.config(cursor="")
```

1. The Gui plots are also enhanced with an interactive option Bar to zoom,move around. The down side here is that while rendering the plots the bar is not fully connected with the data and is mainly showing for display.

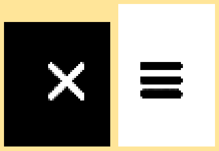


- **Theming**: The application supports different themes, allowing users to switch between light and dark modes.



- **Images**: Images for burger bar, which change based off of theme light or dark.

  Both images contain a primary and referred colour change.

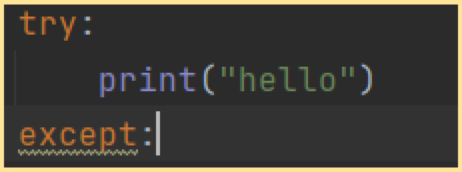


Error Handling

The application includes error handling mechanisms to manage issues related to file loading and data processing. For instance:

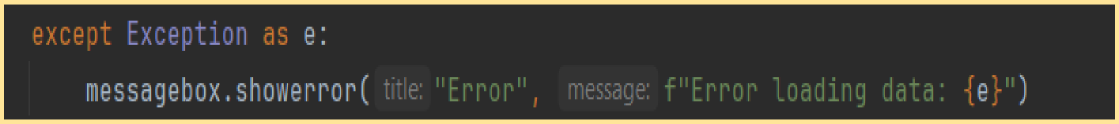
- **File Loading Errors**: Displays an error message if the CSV file cannot be loaded.

  Error handling through means of try and except.



```
try:
    print("hello")
except:
```

- **Data Processing Errors**: Handles exceptions during data manipulation and visualisation, ensuring that the user is informed of any issues.



```
except Exception as e:
    messagebox.showerror( title: "Error",  message: f"Error loading data: {e}")
```

App Improvements & Recommendations

1. **Error Handling**: Implement more specific error messages for different types of exceptions.

   Having try and except operations has made my code more robust and organised being able to trough an error with specific messages also makes it easier as a developer to know when and where an issue has occurred

2. **Data Validation**: Add checks to ensure the CSV file contains the required columns before processing.

   When the file is uploaded to the new window we are welcomed with heading such as, Date, high, low , etc.

   Information that is displayed from the data are added correspondingly to the headings, if data is not of the heading it will not load.

3. **User Experience**: Improve the responsiveness of the GUI components and add tooltips for better user guidance.

   In order to improve the Gui csv button should update Gui data to match the dataset.

   Buttons , radio buttons, theme and interactive dash-boread have advanced user experience to interact with the project limited to only changing them and loading data table functions.

4. **Testing**: Conduct thorough testing with different datasets to ensure robustness and handle edge cases.

   While testing the Gui i have discovered the limitation of my current Gui some of these limitation are:

   - Not resizable Gui fixed to user screen width and height
   - Loads file from button but only to the table visualisation on screen will not update if another data is inserted
   - Interactive bar for plots is not interactive with plots.
   - Menu bar save, and open function is not implemented, exit function works as supposed to.
   - Limited option on visualisation graphs and plots only 3 available

Conclusion:

With the Data Visualization and Analysis Application, Python's capabilities are leveraged to build a GUI-based data analysis tool.

Despite the application's success in implementing key features, there are still areas for improvement, such as error handling and data validation. This functionality could be expanded and refined in future developments.