**Abstract**:

In the realm of modern education and business environments, efficient attendance management systems play a pivotal role in tracking attendance records accurately and conveniently. This abstract outlines the development of an WEBCAM based attendance system integrated with a user-friendly Tkinter interface for seamless interaction and Excel storage for robust data management.

The proposed system utilizes the WEBCAM microcontroller, renowned for its versatility and capability to capture images and process data effectively. Leveraging its Wi-Fi connectivity, the system enables real-time communication with a central server, facilitating instantaneous data transfer and analysis.

The user interface is built using Tkinter, a Python library, offering an intuitive and accessible platform for administrators to manage attendance records effortlessly. Through the interface, users can initiate attendance sessions, view attendance reports, and perform administrative tasks with ease.

Furthermore, the system incorporates Excel as a storage solution for attendance data, ensuring reliability, scalability, and compatibility with existing data management practices. Attendance records captured by the WEBCAM are stored systematically in Excel sheets, enabling seamless integration with other software applications for further analysis and reporting.

**Introduction:**

In today's fast-paced world, the need for efficient attendance management systems has become increasingly critical in various sectors, including education, corporate environments, and public institutions. Traditional methods of attendance tracking, such as manual registers or biometric systems, often prove to be cumbersome, prone to errors, and lacking in versatility. To address these challenges, this project proposes the development of an advanced attendance management system leveraging modern technologies, including the WEBCAM microcontroller, Tkinter GUI framework, and Excel data storage.

The WEBCAM microcontroller is a powerful and versatile device known for its capability to capture images, process data, and communicate over Wi-Fi networks. By harnessing the capabilities of the ESP32-CAM, this project aims to create a sophisticated attendance tracking system that offers real-time data capture, seamless connectivity, and robust performance.

To enhance user interaction and accessibility, the project incorporates Tkinter, a Python library for creating graphical user interfaces (GUIs). The Tkinter interface provides administrators with an intuitive platform to initiate attendance sessions, view attendance reports, and perform administrative tasks efficiently.

Moreover, the project integrates Excel as a storage solution for attendance data. Excel's familiarity, flexibility, and compatibility make it an ideal choice for storing and managing attendance records. By storing attendance data in Excel format, the system ensures ease of access, scalability, and seamless integration with existing data management workflows.

The proposed attendance management system aims to revolutionize the way attendance is tracked and managed in educational institutions, corporate environments, and other organizations. By combining cutting-edge hardware, intuitive software, and efficient data storage mechanisms, the system promises to streamline attendance tracking processes, minimize errors, and improve overall efficiency. Additionally, the system's versatility and scalability make it adaptable to various settings and customizable to meet specific requirements.

In summary, this project seeks to introduce an innovative attendance management system that addresses the shortcomings of traditional methods while harnessing the potential of modern technologies to deliver a seamless and effective solution for attendance tracking and management.

**Literature Survey:**

Attendance management systems have undergone significant advancements in recent years, driven by the integration of emerging technologies and the growing demand for more efficient and user-friendly solutions. This literature survey examines existing research and developments related to attendance management systems, focusing on the integration of hardware, software, and data storage solutions similar to the proposed project.

**WEBCAM Microcontroller Technology:** The WEBCAM microcontroller has gained attention in various applications due to its powerful features, including Wi-Fi connectivity, camera capabilities, and low power consumption. Research by Zhang et al. (2020) highlights the use of ESP32-based systems for remote monitoring and surveillance, showcasing the potential of this technology in data capture and communication.

**Graphical User Interface (GUI) Frameworks**: GUI frameworks such as Tkinter have been widely adopted for creating user-friendly interfaces in Python applications. Studies by Smith et al. (2019) discuss the usability and design considerations of GUI frameworks for enhancing user interaction and experience in software applications.

**Attendance Management Systems in Educational Institutions:** Several studies have explored the implementation of attendance management systems in educational institutions. Research by Gupta and Kumar (2018) evaluates the effectiveness of biometric-based attendance systems in improving attendance monitoring and reducing absenteeism among students.

**Data Storage Solutions:** Excel and other spreadsheet-based solutions have been commonly used for storing and managing attendance data. However, research by Patel et al. (2021) discusses the limitations of traditional data storage methods and proposes alternative approaches, such as cloud-based solutions, for enhanced data management and accessibility.

**Integration of Hardware and Software:** Recent advancements in IoT (Internet of Things) technologies have enabled the integration of hardware and software components in attendance management systems. Research by Li et al. (2022) presents a smart attendance system using IoT devices for real-time data capture and analysis, highlighting the potential for improved efficiency and accuracy.

**Existing method:**

The current method of attendance management relies primarily on manual processes and documentation. In this traditional system, attendance tracking involves the use of paper-based registers or spreadsheets where individuals physically sign in to indicate their presence. This manual approach to attendance management poses several challenges and limitations, including:

**Time-Consuming:** Manual attendance tracking requires significant time and effort from administrators to manage and update attendance records manually. This process is labor-intensive and prone to errors, especially in large organizations or institutions with a high volume of attendees.

**Prone to Errors:** Manual entry of attendance data leaves room for human errors, such as illegible handwriting, data entry mistakes, or unintentional omissions. These errors can lead to inaccuracies in attendance records, affecting the reliability and integrity of the data.

**Limited Accessibility:** Paper-based attendance records are often stored in physical documents or spreadsheets, making it challenging to access and update them remotely. This limitation can hinder the timely availability of attendance information for analysis or decision-making purposes.

**Lack of Real-Time Monitoring**: The manual system lacks real-time monitoring capabilities, making it difficult to track attendance trends, identify patterns, or respond promptly to attendance-related issues. Administrators may not have access to up-to-date attendance information, hindering their ability to make informed decisions.

**Difficulty in Data Analysis**: Analyzing attendance data collected manually can be cumbersome and time-consuming. Extracting insights or generating reports from paper-based records may require manual data entry into analysis tools, further delaying the decision-making process.

**Proposed Method:**

The proposed method aims to revolutionize the attendance management process by integrating the ESP32 wireless camera module with Python programming to enable wireless attendance tracking and storage in Excel format. The method comprises several key steps:

WEBCAM Integration: The WEBCAM module will be utilized for its camera capabilities and wireless connectivity. It will capture images of individuals as they enter a designated area, such as a classroom or office space, and transmit these images wirelessly to a central server or computer for processing.

Image Processing with Python: Python programming language, along with OpenCV library, will be used for image processing and analysis. The captured images will be processed to detect faces and recognize individuals present in the scene. This step may involve facial detection and recognition algorithms to identify attendees accurately.

Attendance Tracking: Upon detecting and recognizing faces in the captured images, the system will record attendance data in real-time. Each recognized face will be timestamped and marked as present, and this information will be stored temporarily in memory.

Wireless Communication: The WEBCAM module will communicate wirelessly with the central server or computer using Wi-Fi connectivity. The captured images and attendance data will be transmitted in real-time to the server for further processing and storage.

Storage in Excel Format: The attendance data received by the central server or computer will be formatted and stored in Excel sheets. Each sheet may represent a specific date or session, with columns indicating the name or ID of attendees and their corresponding attendance status (present/absent).

User Interface (Optional): Optionally, a user-friendly interface can be developed using Python's Tkinter library to provide administrators with access to attendance records, reports, and management functionalities. This interface may allow users to view, edit, and export attendance data directly from the Excel files.

Data Analysis and Reporting: Attendance data stored in Excel format can be easily analyzed and used to generate reports or insights using data analysis tools or software. Administrators can track attendance trends, identify patterns, and make data-driven decisions to improve attendance management practices.

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python tkinter is the fastest and easiest way to create GUI applications. Creating a GUI using tkinter is an easy task.

**To create a tkinter Python app:**

Importing the module – tkinter

Create the main window (container)

Add any number of widgets to the main window

Apply the event Trigger on the widgets.

Importing a tkinter is the same as importing any other module in the Python code. Note that the name of the module in Python 2.x is ‘Tkinter’ and in Python 3.x it is ‘tkinter’.

import tkinter

There are two main methods used which the user needs to remember while creating the Python application with GUI.

ython provides various options for developing graphical user interfaces (GUIs). The most important features are listed below.

* Tkinter − Tkinter is the Python interface to the Tk GUI toolkit shipped with Python. We would look at this option in this chapter.
* wxPython − This is an open-source Python interface for wxWidgets GUI toolkit. You can find a complete tutorial on WxPython [here](https://www.tutorialspoint.com/wxpython/index.htm).
* PyQt − This is also a Python interface for a popular cross-platform Qt GUI library. TutorialsPoint has a very good tutorial on PyQt5 [here](https://www.tutorialspoint.com/pyqt/index.htm).
* PyGTK − PyGTK is a set of wrappers written in Python and C for GTK + GUI library. The complete PyGTK tutorial is available [here](https://www.tutorialspoint.com/pygtk/index.htm).
* PySimpleGUI − PySimpleGui is an open source, cross-platform GUI library for Python. It aims to provide a uniform API for creating desktop GUIs based on Python's Tkinter, PySide and WxPython toolkits. For a detaile PySimpleGUI tutorial, click [here](https://www.tutorialspoint.com/python/https:/www.tutorialspoint.com/pysimplegui/index.htm).
* Pygame − Pygame is a popular Python library used for developing video games. It is free, open source and cross-platform wrapper around Simple DirectMedia Library (SDL). For a comprehensive tutorial on Pygame, [visit](https://www.tutorialspoint.com/pygame/index.htm) this link.
* Jython − Jython is a Python port for Java, which gives Python scripts seamless access to the Java class libraries on the local machinehttp: [//www.jython.org](http://www.jython.org/).

There are many other interfaces available, which you can find them on the net.

Tkinter Programming

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

The tkinter package includes following modules −

* Tkinter − Main Tkinter module.
* tkinter.colorchooser − Dialog to let the user choose a color.
* tkinter.commondialog − Base class for the dialogs defined in the other modules listed here.
* tkinter.filedialog − Common dialogs to allow the user to specify a file to open or save.
* tkinter.font − Utilities to help work with fonts.
* tkinter.messagebox − Access to standard Tk dialog boxes.
* tkinter.scrolledtext − Text widget with a vertical scroll bar built in.
* tkinter.simpledialog − Basic dialogs and convenience functions.
* tkinter.ttk − Themed widget set introduced in Tk 8.5, providing modern alternatives for many of the classic widgets in the main tkinter module.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps.

* Import the Tkinter module.
* Create the GUI application main window.
* Add one or more of the above-mentioned widgets to the GUI application.
* Enter the main event loop to take action against each event triggered by the user.

Example

# note that module name has changed from Tkinter in Python 2

# to tkinter in Python 3

import tkinter

top = tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()

This would create a following window −



When the program becomes more complex, using an object-oriented programming approach makes the code more organized.

import tkinter as tk

class App(tk.Tk):

def \_\_init\_\_(self):

super().\_\_init\_\_()

app = App()

app.mainloop()

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table −

|  |  |
| --- | --- |
| Sr.No. | Operator & Description |
| 1 | [Button](https://www.tutorialspoint.com/python/tk_button.htm)  The Button widget is used to display the buttons in your application. |
| 2 | [Canvas](https://www.tutorialspoint.com/python/tk_canvas.htm)  The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application. |
| 3 | [Checkbutton](https://www.tutorialspoint.com/python/tk_checkbutton.htm)  The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time. |
| 4 | [Entry](https://www.tutorialspoint.com/python/tk_entry.htm)  The Entry widget is used to display a single-line text field for accepting values from a user. |
| 5 | [Frame](https://www.tutorialspoint.com/python/tk_frame.htm)  The Frame widget is used as a container widget to organize other widgets. |
| 6 | [Label](https://www.tutorialspoint.com/python/tk_label.htm)  The Label widget is used to provide a single-line caption for other widgets. It can also contain images. |
| 7 | [Listbox](https://www.tutorialspoint.com/python/tk_listbox.htm)  The Listbox widget is used to provide a list of options to a user. |
| 8 | [Menubutton](https://www.tutorialspoint.com/python/tk_menubutton.htm)  The Menubutton widget is used to display menus in your application. |
| 9 | [Menu](https://www.tutorialspoint.com/python/tk_menu.htm)  The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton. |
| 10 | [Message](https://www.tutorialspoint.com/python/tk_message.htm)  The Message widget is used to display multiline text fields for accepting values from a user. |
| 11 | [Radiobutton](https://www.tutorialspoint.com/python/tk_radiobutton.htm)  The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time. |
| 12 | [Scale](https://www.tutorialspoint.com/python/tk_scale.htm)  The Scale widget is used to provide a slider widget. |
| 13 | [Scrollbar](https://www.tutorialspoint.com/python/tk_scrollbar.htm)  The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes. |
| 14 | [Text](https://www.tutorialspoint.com/python/tk_text.htm)  The Text widget is used to display text in multiple lines. |
| 15 | [Toplevel](https://www.tutorialspoint.com/python/tk_toplevel.htm)  The Toplevel widget is used to provide a separate window container. |
| 16 | [Spinbox](https://www.tutorialspoint.com/python/tk_spinbox.htm)  The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values. |
| 17 | [PanedWindow](https://www.tutorialspoint.com/python/tk_panedwindow.htm)  A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically. |
| 18 | [LabelFrame](https://www.tutorialspoint.com/python/tk_labelframe.htm)  A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts. |
| 19 | [tkMessageBox](https://www.tutorialspoint.com/python/tk_messagebox.htm)  This module is used to display message boxes in your applications. |

Let us study these widgets in detail.

Standard Attributes

Let us look at how some of the common attributes, such as sizes, colors and fonts are specified.

* [Dimensions](https://www.tutorialspoint.com/python/tk_dimensions.htm)
* [Colors](https://www.tutorialspoint.com/python/tk_colors.htm)
* [Fonts](https://www.tutorialspoint.com/python/tk_fonts.htm)
* [Anchors](https://www.tutorialspoint.com/python/tk_anchors.htm)
* [Relief styles](https://www.tutorialspoint.com/python/tk_relief.htm)
* [Bitmaps](https://www.tutorialspoint.com/python/tk_bitmaps.htm)
* [Cursors](https://www.tutorialspoint.com/python/tk_cursors.htm)

Let us study them briefly −

Geometry Management

All Tkinter widgets have access to the specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

* [The pack() Method](https://www.tutorialspoint.com/python/tk_pack.htm) − This geometry manager organizes widgets in blocks before placing them in the parent widget.
* [The grid() Method](https://www.tutorialspoint.com/python/tk_grid.htm) − This geometry manager organizes widgets in a table-like structure in the parent widget.
* [The place() Method](https://www.tutorialspoint.com/python/tk_place.htm) − This geometry manager organizes widgets by placing them in a specific position in the parent widget.

Let us study the geometry management methods briefly −

SimpleDialog

The simpledialog module in tkinter package includes a dialog class and convenience functions for accepting user input through a modal dialog. It consists of a label, an entry widget and two buttons Ok and Cancel. These functions are −

* askfloat(title, prompt, \*\*kw) − Accepts a floating point number.
* askinteger(title, prompt, \*\*kw) − Accepts an integer input.
* askstring(title, prompt, \*\*kw) − Accepts a text input from the user.

The above three functions provide dialogs that prompt the user to enter a value of the desired type. If Ok is pressed, the input is returned, if Cancel is pressed, None is returned.

askinteger

from tkinter.simpledialog import askinteger

from tkinter import \*

from tkinter import messagebox

top = Tk()

top.geometry("100x100")

def show():

num = askinteger("Input", "Input an Integer")

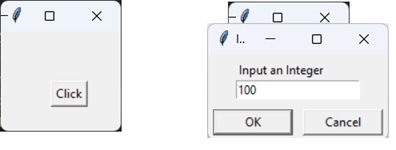
print(num)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following output −



askfloat

from tkinter.simpledialog import askfloat

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

num = askfloat("Input", "Input a floating point number")

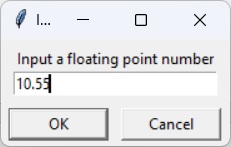
print(num)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following output −



askstring

from tkinter.simpledialog import askstring

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

name = askstring("Input", "Enter you name")

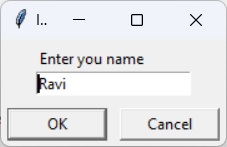
print(name)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following output −



The FileDialog Module

The filedialog module in Tkinter package includes a FileDialog class. It also defines convenience functions that enable the user to perform open file, save file, and open directory activities.

* filedialog.asksaveasfilename()
* filedialog.asksaveasfile()
* filedialog.askopenfilename()
* filedialog.askopenfile()
* filedialog.askdirectory()
* filedialog.askopenfilenames()
* filedialog.askopenfiles()

askopenfile

This function lets the user choose a desired file from the filesystem. The file dialog window has Open and Cancel buttons. The file name along with its path is returned when Ok is pressed, None if Cancel is pressed.

from tkinter.filedialog import askopenfile

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

filename = askopenfile()

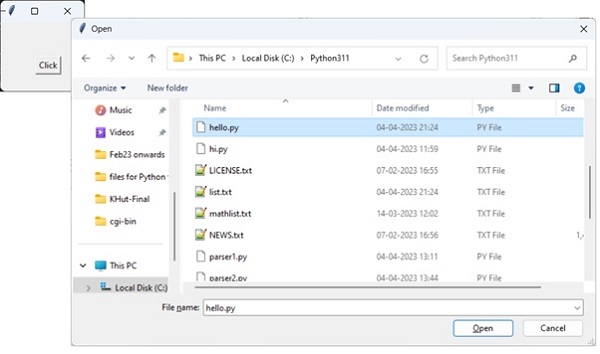
print(filename)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following output −



ColorChooser

The colorchooser module included in tkinter package has the feature of letting the user choose a desired color object through the color dialog. The askcolor() function presents with the color dialog with predefined color swatches and facility to choose custome color by setting RGB values. The dialog returns a tuple of RGB values of chosen color as well as its hex value.

from tkinter.colorchooser import askcolor

from tkinter import \*

top = Tk()

top.geometry("100x100")

def show():

color = askcolor()

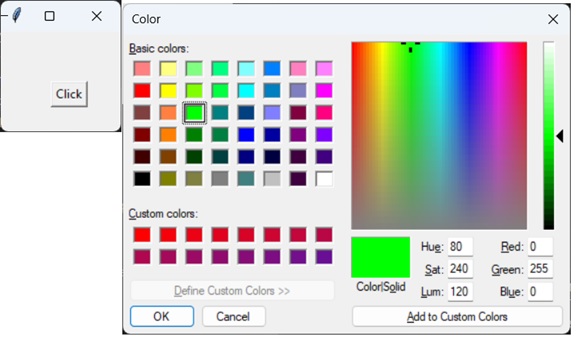
print(color)

B = Button(top, text ="Click", command = show)

B.place(x=50,y=50)

top.mainloop()

It will produce the following output −



((0, 255, 0), '#00ff00')

ttk module

The term ttk stands from Tk Themed widgets. The ttk module was introduced with Tk 8.5 onwards. It provides additional benefits including anti-aliased font rendering under X11 and window transparency. It provides theming and styling support for Tkinter.

The ttk module comes bundled with 18 widgets, out of which 12 are already present in Tkinter. Importing ttk over-writes these widgets with new ones which are designed to have a better and more modern look across all platforms.

The 6 new widgets in ttk are, the Combobox, Separator, Sizegrip, Treeview, Notebook and ProgressBar.

To override the basic Tk widgets, the import should follow the Tk import −

from tkinter import \*

from tkinter.ttk import \*

The original Tk widgets are automatically replaced by tkinter.ttk widgets. They are Button, Checkbutton, Entry, Frame, Label, LabelFrame, Menubutton, PanedWindow, Radiobutton, Scale and Scrollbar.

New widgets which gives a better look and feel across platforms; however, the replacement widgets are not completely compatible. The main difference is that widget options such as "fg", "bg" and others related to widget styling are no longer present in Ttk widgets. Instead, use the ttk.Style class for improved styling effects.

The new widgets in ttk module are −

* Notebook − This widget manages a collection of "tabs" between which you can swap, changing the currently displayed window.
* ProgressBar − This widget is used to show progress or the loading process through the use of animations.
* Separator − Used to separate different widgets using a separator line.
* Treeview − This widget is used to group together items in a tree-like hierarchy. Each item has a textual label, an optional image, and an optional list of data values.
* ComboBox − Used to create a dropdown list of options from which the user can select one.
* Sizegrip − Creates a little handle near the bottom-right of the screen, which can be used to resize the window.

Combobox Widget

The Python ttk Combobox presents a drop down list of options and displays them one at a time. It is a sub class of the widget Entry. Hence it inherits many options and methods from the Entry class.

Syntax

from tkinter import ttk

Combo = ttk.Combobox(master, values.......)

The get() function to retrieve the current value of the Combobox.

Example

from tkinter import \*

from tkinter import ttk

top = Tk()

top.geometry("200x150")

frame = Frame(top)

frame.pack()

langs = ["C", "C++", "Java",

"Python", "PHP"]

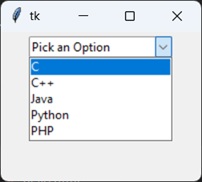
Combo = ttk.Combobox(frame, values = langs)

Combo.set("Pick an Option")

Combo.pack(padx = 5, pady = 5)

top.mainloop()

It will produce the following output −



Progressbar

The ttk ProgressBar widget, and how it can be used to create loading screens or show the progress of a current task.

Syntax

ttk.Progressbar(parent, orient, length, mode)

Parameters

* Parent − The container in which the ProgressBar is to be placed, such as root or a Tkinter frame.
* Orient − Defines the orientation of the ProgressBar, which can be either vertical of horizontal.
* Length − Defines the width of the ProgressBar by taking in an integer value.
* Mode − There are two options for this parameter, determinate and indeterminate.

Example

The code given below creates a progressbar with three buttons which are linked to three different functions.

The first function increments the "value" or "progress" in the progressbar by 20. This is done with the step() function which takes an integer value to change progress amount. (Default is 1.0)

The second function decrements the "value" or "progress" in the progressbar by 20.

The third function prints out the current progress level in the progressbar.

import tkinter as tk

from tkinter import ttk

root = tk.Tk()

frame= ttk.Frame(root)

def increment():

progressBar.step(20)

def decrement():

progressBar.step(-20)

def display():

print(progressBar["value"])

progressBar= ttk.Progressbar(frame, mode='determinate')

progressBar.pack(padx = 10, pady = 10)

button= ttk.Button(frame, text= "Increase", command= increment)

button.pack(padx = 10, pady = 10, side = tk.LEFT)

button= ttk.Button(frame, text= "Decrease", command= decrement)

button.pack(padx = 10, pady = 10, side = tk.LEFT)

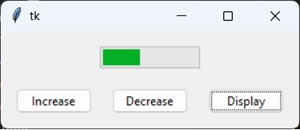
button= ttk.Button(frame, text= "Display", command= display)

button.pack(padx = 10, pady = 10, side = tk.LEFT)

frame.pack(padx = 5, pady = 5)

root.mainloop()

It will produce the following output −



Notebook

Tkinter ttk module has a new useful widget called Notebook. It is a of collection of of containers (e.g frames) which have many widgets as children inside.

Each "tab" or "window" has a tab ID associated with it, which is used to determine which tab to swap to.

You can swap between these containers like you would on a regular text editor.

Syntax

notebook = ttk.Notebook(master, \*options)

Example

In this example, add 3 windows to our Notebook widget in two different ways. The first method involves the add() function, which simply appends a new tab to the end. The other method is the insert() function which can be used to add a tab to a specific position.

The add() function takes one mandatory parameter which is the container widget to be added, and the rest are optional parameters such as text (text to be displayed as tab title), image and compound.

The insert() function requires a tab\_id, which defines the location where it should be inserted. The tab\_id can be either an index value or it can be string literal like "end", which will append it to the end.

import tkinter as tk

from tkinter import ttk

root = tk.Tk()

nb = ttk.Notebook(root)

# Frame 1 and 2

frame1 = ttk.Frame(nb)

frame2 = ttk.Frame(nb)

label1 = ttk.Label(frame1, text = "This is Window One")

label1.pack(pady = 50, padx = 20)

label2 = ttk.Label(frame2, text = "This is Window Two")

label2.pack(pady = 50, padx = 20)

frame1.pack(fill= tk.BOTH, expand=True)

frame2.pack(fill= tk.BOTH, expand=True)

nb.add(frame1, text = "Window 1")

nb.add(frame2, text = "Window 2")

frame3 = ttk.Frame(nb)

label3 = ttk.Label(frame3, text = "This is Window Three")

label3.pack(pady = 50, padx = 20)

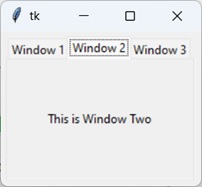
frame3.pack(fill= tk.BOTH, expand=True)

nb.insert("end", frame3, text = "Window 3")

nb.pack(padx = 5, pady = 5, expand = True)

root.mainloop()

It will produce the following output −



Treeview

The Treeview widget is used to display items in a tabular or hierarchical manner. It has support for features like creating rows and columns for items, as well as allowing items to have children as well, leading to a hierarchical format.

Syntax

tree = ttk.Treeview(container, \*\*options)

Options

|  |  |
| --- | --- |
| Sr.No. | Option & Description |
| 1 | columns  A list of column names |
| 2 | displaycolumns  A list of column identifiers (either symbolic or integer indices) specifying which data columns are displayed and the order in which they appear, or the string "#all". |
| 3 | height  The number of rows visible. |
| 4 | padding  Specifies the internal padding for the widget. Can be either an integer or a list of 4 values. |
| 5 | selectmode  One of "extended", "browse" or "none". If set to "extended" (default), multiple items can be selected. If "browse", only a single item can be selected at a time. If "none", the selection cannot be changed by the user. |
| 6 | show  A list containing zero or more of the following values, specifying which elements of the tree to display. The default is "tree headings", i.e., show all elements. |

Example

In this example we will create a simple Treeview ttk Widget and fill in some data into it. We have some data already stored in a list which will be reading and adding to the Treeview widget in our read\_data() function.

We first need to define a list/tuple of column names. We have left out the column "Name" because there already exists a (default) column with a blank name.

We then assign that list/tuple to the columns option in Treeview, followed by defining the "headings", where the column is the actual column, whereas the heading is just the title of the column that appears when the widget is displayed. We give each a column a name. "#0" is the name of the default column.

The tree.insert() function has the following parameters −

* Parent − which is left as an empty string if there is none.
* Position − where we want to add the new item. To append, use tk.END
* Iid − which is the item ID used to later track the item in question.
* Text − to which we will assign the first value in the list (the name).

Value we will pass the the other 2 values we obtained from the list.

The Complete Code

import tkinter as tk

import tkinter.ttk as ttk

from tkinter import simpledialog

root = tk.Tk()

data = [

["Bobby",26,20000],

["Harrish",31,23000],

["Jaya",18,19000],

["Mark",22, 20500],

]

index=0

def read\_data():

for index, line in enumerate(data):

tree.insert('', tk.END, iid = index,

text = line[0], values = line[1:])

columns = ("age", "salary")

tree= ttk.Treeview(root, columns=columns ,height = 20)

tree.pack(padx = 5, pady = 5)

tree.heading('#0', text='Name')

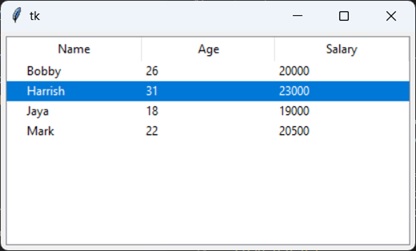
tree.heading('age', text='Age')

tree.heading('salary', text='Salary')

read\_data()

root.mainloop()

It will produce the following output −



Sizegrip

The Sizegrip widget is basically a small arrow-like grip that is typically placed at the bottom-right corner of the screen. Dragging the Sizegrip across the screen also resizes the container to which it is attached to.

Syntax

sizegrip = ttk.Sizegrip(parent, \*\*options)

Example

import tkinter as tk

import tkinter.ttk as ttk

root = tk.Tk()

root.geometry("100x100")

frame = ttk.Frame(root)

label = ttk.Label(root, text = "Hello World")

label.pack(padx = 5, pady = 5)

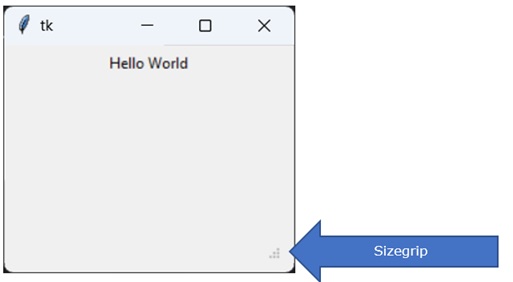
sizegrip = ttk.Sizegrip(frame)

sizegrip.pack(expand = True, fill = tk.BOTH, anchor = tk.SE)

frame.pack(padx = 10, pady = 10, expand = True, fill = tk.BOTH)

root.mainloop()

It will produce the following output −



Separator

The ttk Separator widget is a very simple widget, that has just one purpose and that is to help "separate" widgets into groups/partitions by drawing a line between them. We can change the orientation of this line (separator) to either horizontal or vertical, and change its length/height.

Syntax

separator = ttk.Separator(parent, \*\*options)

The "orient", which can either be tk.VERTICAL or tk.HORIZTONAL, for a vertical and horizontal separator respectively.

Example

Here we have created two Label widgets, and then created a Horizontal Separator between them.

import tkinter as tk

import tkinter.ttk as ttk

root = tk.Tk()

root.geometry("200x150")

frame = ttk.Frame(root)

label = ttk.Label(frame, text = "Hello World")

label.pack(padx = 5)

separator = ttk.Separator(frame,orient= tk.HORIZONTAL)

separator.pack(expand = True, fill = tk.X)

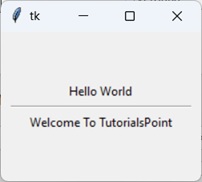
label = ttk.Label(frame, text = "Welcome To TutorialsPoint")

label.pack(padx = 5)

frame.pack(padx = 10, pady = 50, expand = True, fill = tk.BOTH)

root.mainloop()

It will produce the following output −



**Architecture Diagram**

True Face

False Face

Classification Models

Import Credit card Fraud dataset

Exploratory Data Analysis

Import Packages and Libraries

Training data

Testing data

**Spliting Training and Testing data**

**Spliting Training and Testing data**

**MODULES AND DESCRIPTION**

**Importing the packages**

For this project, our primary packages are going to be Pandas to work with data, NumPy to work with arrays, scikit-learn for data split, building and evaluating the classification models, and finally the xgboost package for the xgboost classifier model algorithm. Let’s import all of our primary packages into our python environment.

**Importing Dataset**

The data we are going to use is the Kaggle Credit Card Fraud Detection dataset (https://www.kaggle.com/mlg-ulb/creditcardfraud). It contains features V1 to V28 which are the principal components obtained by PCA. We are going to neglect the time feature which is of no use to build the models. The remaining features are the ‘Amount’ feature that contains the total amount of money being transacted and the ‘Class’ feature that contains whether the transaction is a fraud case or not.

**Analyze Target classes**

Analyze that how many fraud cases and non-fraud cases are there in our dataset. Along with that, let’s also compute the percentage of fraud cases in the overall recorded transactions. We can see that out of 284,807 samples, there are only 492 fraud cases which is only 0.17 percent of the total samples. So, we can say that the data we are dealing with is highly imbalanced data and needs to be handled carefully when modeling and evaluating.

**Splitting of data**

In this process, we are going to define the independent (X) and the dependent variables (Y). Using the defined variables, we will split the data into a training set and testing set which is further used for modeling and evaluating. We can split the data easily using the ‘train\_test\_split’ algorithm in python.

**ER Diagram:**

Main

phot

name

**Conclusion**:

Efficiency and Automation: By leveraging the ESP32-CAM's camera capabilities and wireless connectivity, the system enables automated attendance tracking without the need for manual intervention. Attendees' presence can be detected and recorded in real-time, reducing the time and effort required for attendance management tasks.

Accuracy and Reliability: The use of image processing algorithms and facial recognition techniques ensures accurate identification of attendees, minimizing errors commonly associated with manual attendance tracking methods. The system provides reliable attendance data, enhancing the integrity and reliability of attendance records.

Wireless Connectivity: The wireless communication capabilities of the WEBCAM module enable seamless transmission of attendance data to a central server or computer. Attendances can be captured and recorded wirelessly, providing flexibility and convenience for administrators and attendees alike.

Data Storage and Accessibility: Attendance data is stored in Excel format, offering a familiar and widely used platform for data storage and management. Excel's versatility allows for easy access, manipulation, and analysis of attendance records, facilitating data-driven decision-making and reporting.

Scalability and Customization: The proposed system can be scaled and customized to suit the specific requirements of various organizations and educational institutions. Additional features, such as user interfaces for administrators or integration with existing systems, can be implemented to enhance functionality and usability.

**Bibliography:**

Espressif Systems. "ESP32-CAM." Accessed February 2024. https://www.espressif.com/en/products/modules/esp32-cam

Bradski, Gary, and Adrian Kaehler. "Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library." O'Reilly Media, 2017.

McKinney, Wes. "Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython." O'Reilly Media, 2017.

Brownlee, Jason. "Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python." Machine Learning Mastery, 2019.

Deitel, Paul J., and Harvey M. Deitel. "Python for Programmers." Pearson, 2018.

Walling, J. Richard. "Python Programming for the Absolute Beginner." No Starch Press, 2017.

Microsoft. "Excel." Accessed February 2024. https://www.microsoft.com/en-us/microsoft-365/excel

Tkinter Documentation. Accessed February 2024. https://docs.python.org/3/library/tkinter.html

Zhang, Y., Li, T., & Zhang, Y. (2020). "Design and implementation of remote monitoring system based on ESP32 and cloud computing." In Proceedings of the 6th International Conference on Education Technology, Management and Humanities Science (ETMHS 2020).

Gupta, A., & Kumar, R. (2018). "Automated attendance system using biometric technology." International Journal of Engineering and Advanced Technology (IJEAT), 8(6), 174-178.

**Code:**

import cv2, os, sys, time

import numpy as np

from PIL import Image

from configs import \*

import urllib.request

import cv2, os, sys, pickle

import numpy as np

from PIL import Image

from configs import \*

import cv2, os, sys, pickle

import numpy as np

from PIL import Image

from configs import \*

import cv2, os, sys, pickle

import time

import numpy as np

from configs import \*

import openpyxl

import urllib.request

from tkinter import \*

from PIL import Image, ImageTk

from datetime import date

from datetime import datetime

faceCascade = cv2.CascadeClassifier('bin/haarcascade\_frontalface\_default.xml')

profileCascade = cv2.CascadeClassifier('bin/haarcascade\_profileface.xml')

recognizer = cv2.face\_LBPHFaceRecognizer.create()

cv2.CascadeClassifier('bin/haarcascade\_profileface.xml')

recognizer.read('face\_rec\_saved\_model.yaml')

with open(label\_name\_map\_file, 'rb') as handle:

label\_name\_map = pickle.load(handle)

print("Press 'q' to quit\n\n\n")

#can also use createEigenFaceRecognizer() or createFisherFaceRecognizer() or createLBPHFaceRecognizer()

#Read http://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec\_tutorial.html to understand ML behind

i=0

url = "http://192.168.239.1/cam-mid.jpg"

def ece(d):

path = "ece.xlsx"

wb\_obj = openpyxl.load\_workbook(path)

sheet = wb\_obj.active

sheet = wb\_obj.active

# print the total number of rows

s=sheet.max\_row

if s>0:

s=s+1

print(s)

#here donot declare c0

#writing values to cells

c1 = sheet.cell(s, column = 1)

c1.value = d[0]

c2 = sheet.cell(s , column = 2)

c2.value = d[1]

c3 = sheet.cell(s , column = 3)

c3.value = d[2]

c4 = sheet.cell(s , column = 4)

c4.value = d[3]

#c6 = sheet.cell(s , column = 6) for add column

#c6.value = d[6]

wb\_obj.save("ece.xlsx")

def predictFacesFromWebcam (label2name\_map):

cam = cv2.VideoCapture(url)

d=['0','0','0','0','0']

"""d[0]="jjj"

today = date.today()

now = datetime.now()

current\_time = now.strftime("%H:%M:%S")

d[1]=str(today)

d[2]=current\_time

d[3]="present"

ece(d)"""

i=0

while i==0:

img\_resp=urllib.request.urlopen(url)

imgnp=np.array(bytearray(img\_resp.read()),dtype=np.uint8)

#ret, frame = cap.read()

img\_rgb= cv2.imdecode(imgnp,-1)

gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY)

faces = faceCascade.detectMultiScale(gray, scaleFactor, minNeighbors, cascadeFlags, minSize)

for (x, y, w, h) in faces:

cv2.rectangle(img\_rgb, (x, y), (x+w, y+h), (0, 255, 0), 2)

name\_predicted, confidence = recognizer.predict(cv2.resize(gray[y: y + h, x: x + w], face\_resolution))

print(str(name\_predicted) +' , ' +str(confidence))

if(name\_predicted!=0 and confidence<confidence\_threshold):

print("It is predicted as "+label2name\_map[name\_predicted])

cv2.putText(img\_rgb, label2name\_map[name\_predicted], (x+3,y+h+20), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0,255,0))

d[0]=str(label2name\_map[name\_predicted])

today = date.today()

now = datetime.now()

current\_time = now.strftime("%H:%M:%S")

d[1]=str(today)

d[2]=current\_time

d[3]="present"

ece(d)

i=1

login\_sucess()

cv2.imshow('Video', img\_rgb)

if cv2.waitKey(1) & 0xFF == ord('q'):

print("\nQuitting")

break

cM.release()

def main():

predictFacesFromWebcam (label\_name\_map)

def getFacesAndNames(path):

image\_paths = [os.path.join(path, f) for f in os.listdir(path) if f.endswith(pic\_format)]

faces = []

names = []

count = 0

label2name\_map = {}

name2label\_map = {}

for i in image\_paths:

#Convert to grayscale and get as np\_array

img\_gray = Image.open(i).convert('L')

width, height = img\_gray.size

img = np.array(img\_gray, 'uint8')

#cv2.imshow('q',img)

#Create label for person

#name =str(os.path.split(i)[1].split(".")[0].replace("subject", ""))

person\_name = os.path.split(i)[1].split(".")[0]

print(person\_name)

if person\_name in name2label\_map:

name = name2label\_map[person\_name]

else:

count += 1

name2label\_map[person\_name] = count

name = count

label2name\_map[count] = person\_name

#to detect frontal face

face = faceCascade.detectMultiScale(img, scaleFactor, minNeighbors, cascadeFlags, minSize)

#to detect left side face

#sideface\_left = profileCascade.detectMultiScale(img, scaleFactor, minNeighbors, cascadeFlags, minSize)

#to detect right side face (mirror flip the image and use same cascade)

#sideface\_right = profileCascade.detectMultiScale(np.fliplr(img), scaleFactor, minNeighbors, cascadeFlags, minSize)

#Add all detected faces to the list

for(x, y, w, h) in face:

faces.append(cv2.resize(img[y: y + h, x: x + w], face\_resolution))

names.append(name)

#cv2.imshow("Adding faces to traning set...", img[y: y + h, x: x + w])

#cv2.waitKey(0)

print("Frontal Face found in "+i)

'''for(x, y, w, h) in sideface\_left:

faces.append(cv2.resize(img[y: y + h, x: x + w], face\_resolution))

names.append(name)

#cv2.imshow("Adding faces to traning set...", img[y: y + h, x: x + w])

#cv2.waitKey(0)

print("Left Side Face found in "+i)

for(X, y, w, h) in sideface\_right:

x = width-(X+w) #reflip to unmirror

faces.append(cv2.resize(img[y: y + h, x: x+w], face\_resolution))

names.append(name)

#cv2.imshow("Adding faces to traning set...", img[y: y + h, x: x + w])

#cv2.waitKey(0)

print("Right Side Face found in "+i)'''

return faces, np.array(names), label2name\_map

def train():

faces, names, label\_name\_map = getFacesAndNames('images\_db') #Setup the facial pictures

#cv2.destroyAllWindows()

recognizer.train(faces, names) #Train for facial recognition

recognizer.write(outfile) #Dump the trained model

with open(label\_name\_map\_file, 'wb') as handle:

pickle.dump(label\_name\_map, handle, protocol=pickle.HIGHEST\_PROTOCOL) #Dump the label:name map

def photho1():

username1 = username\_verify.get()

if len(username1)> 0:

video\_capture= cv2.VideoCapture(url) #Set the source webcam

video\_capture .set(3,640)

video\_capture .set(4,480)

name = str(username1)

neram = str(int(time.time()))

i=0

while i<30:

img\_resp=urllib.request.urlopen(url)

imgnp=np.array(bytearray(img\_resp.read()),dtype=np.uint8)

img\_rgb= cv2.imdecode(imgnp,-1)

gray = cv2.cvtColor(img\_rgb, cv2.COLOR\_BGR2GRAY)

cv2.imshow('Video', gray)

if cv2.waitKey(1) & 0xFF == ord('n'):

cv2.imwrite(db\_path+"/"+str(name)+"."+neram+str(i)+".png", gray)

print("Saved as "+str(name)+"."+neram+str(i)+".png"+"\n\n")

i+=1

print("Waiting to capture photo......")

print("\n\nPROCESS STOPPED......")

video\_capture.release()

train()

def name():

global main\_screen1

main\_screen1= Toplevel(login\_screen)

main\_screen1.geometry("766x708")

main\_screen1.configure(background='#3d5705')

global username\_verify

global password\_verify

username\_verify = StringVar()

password\_verify = StringVar()

global username\_login\_entry

global password\_login\_entry

Label(main\_screen1, text="entername",font=("Verdana", 15),bg="#c6eb73",fg="black").place(x=320,y=200)

username\_login\_entry = Entry(main\_screen1,justify=RIGHT, textvariable=username\_verify,font=('Verdana',15,'bold')).place(x=250,y=240)

Button(main\_screen1, text="Login", width=30, height=2,bd=5, command = photho1,bg="#a6ed07",activebackground="#c6eb73").place(x=270,y=300)

def login():

global login\_screen

login\_screen = Tk()

login\_screen.title("Login")

login\_screen.geometry("766x708")

login\_screen.configure(background='#3d5705')

Label(login\_screen, text="Please enter details below to login",bg="#c6eb73", font=("Calibri", 30)).place(x=150,y=10)

Button(login\_screen, text="PHOTO ", width=30, height=2,bd=5, command = name,bg="#a6ed07",activebackground="#c6eb73").place(x=270,y=400)

Button(login\_screen, text="MAIN ", width=30, height=2,bd=5, command =main ,bg="#a6ed07",activebackground="#c6eb73").place(x=570,y=400)

def login\_sucess():

z1=Label(login\_screen , text="Face SUCCESSES ", bg="green",fg="black",font=("calibri", 11))

z1.place(x=350,y=500)

# Designing popup for login invalid password

def password\_not\_recognised():

z1=Label(login\_screen , text="INVALID USER ", bg="red",fg="black",font=("calibri", 11))

z1.place(x=350,y=500)

def main\_account\_screen():

login()

login\_screen.mainloop()

main\_account\_screen()