A blue circle with white text

AI-generated content may be incorrect.

**Kardan University**

Data mining

Instructor: Sayed Murtaza Kazimi

Student: Aziz Ahmad Afzali

ID:201-2204033

7TH semester BCS

Final project documentation/Business proposal

Handwritten digit recognition using the MNIST dataset.

* **Real-time data capture with Paint**
* Custom dataset creation and preprocessing (grayscale conversion, blurring, thresholding)
* Manual digit annotation and storage
* Dataset loading, visualization, training, and saving using SVM
* A Tkinter GUI with buttons to control each step

Honestly, it’s very close to building your own mini version of MNIST!

1. You might consider adding **error handling** for image saving and directory creation (like using os.makedirs(..., exist=True)).
2. For training feedback, perhaps show a visual of **prediction vs actual** using confusion matrix heatmaps from sklearn.metrics.

3. Eventually, you could integrate a **canvas** in Tkinter instead of launching Paint, for a smoother experience.

**Architectural Diagram**

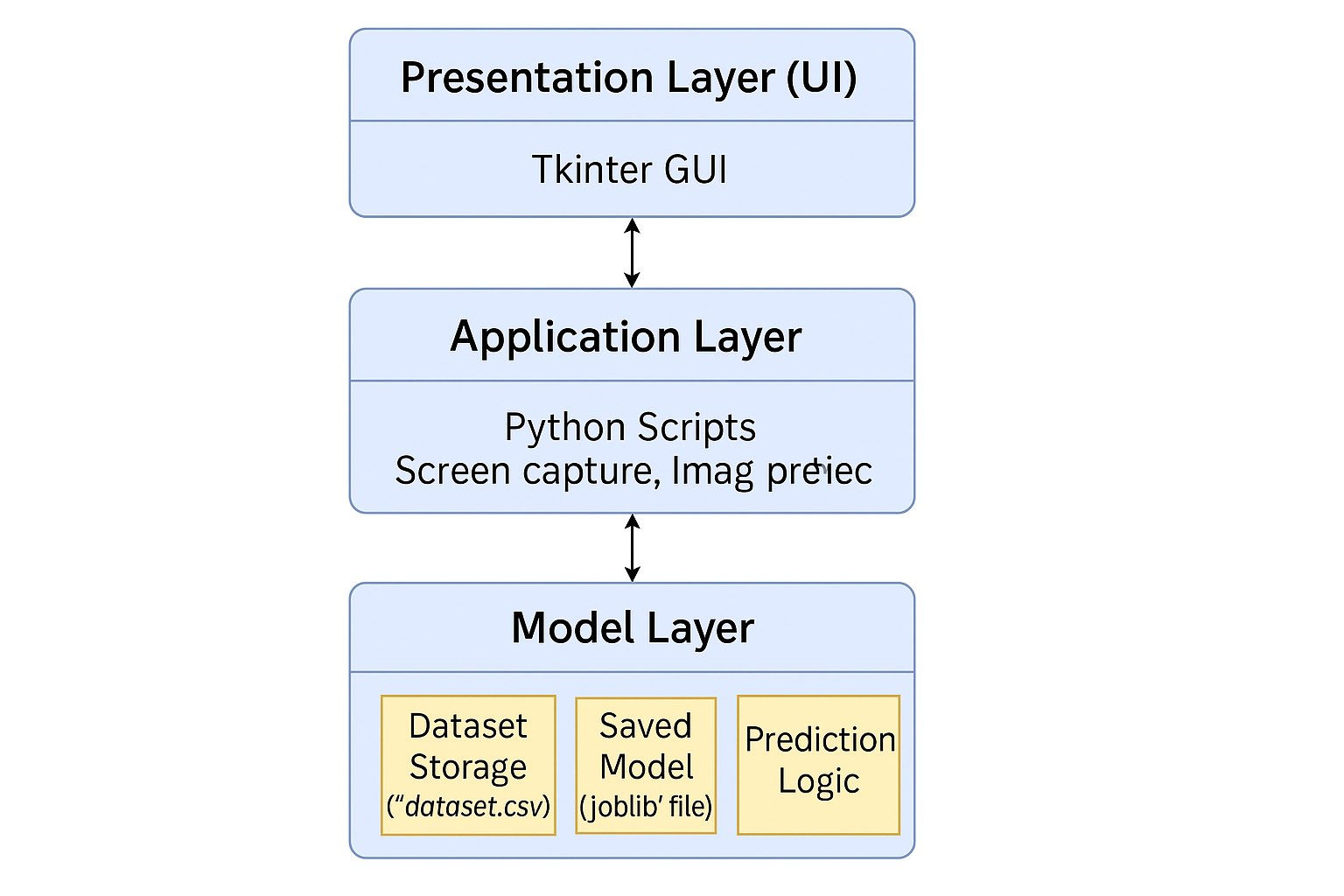
A typical architecture would follow a layered MVC-like pattern:

**Presentation Layer (UI)** → Tkinter GUI for capturing, training, and prediction

**Application Layer** → Python Scripts handling screen capture, image processing, ML training/testing

**Model Layer** → Dataset storage (dataset.csv), saved model (joblib file), prediction logic

You could represent this visually using a tool like draw.io or Lucidchart with clear arrows showing flow between components.



**2. Use Case Design**

**Primary Actor: User (you or app user) Use Cases:**

* **Capture Digit Image**
* **Generate Dataset**
* **Train Model**
* **Predict Digit**
* **Display Accuracy Each use case will have preconditions (e.g. data availability), flow of events (steps taken), and postconditions (e.g. model trained, accuracy shown).**
* **Time Sequence Diagram**
* Example of the "Digit Recognition" process:
* **User → UI (Tkinter) → Image Grab → Save Image**
* **Preprocessing CSV Update**
* **→ Train Model → Evaluate Accuracy → Display Result**

**Cost Estimation**

**Assuming development by a solo developer with open-source tools:**

* **Development Time: ~80 hours**
* **Hourly Rate: e.g., $15/hour**
* **Software Tools: Free (Python, OpenCV, etc.)**
* **Total: ~$1,200 (or equivalent in local currency)**

**Time Estimation**

| **Task** | **Hours** |
| --- | --- |
| Planning & Design | 8 |
| GUI Development | 10 |
| Image Processing | 10 |
| Model Training | 10 |
| Dataset Management | 8 |
| Integration & Testing | 12 |
| Documentation & UI/UX | 6 |
| **Total** | **64** |

**DFD (if required)**

**Level 0 DFD**

* **User → [Digit Recognizer System] → Receives Prediction**
* **System ↔ Dataset / Model Files**
* **Internal Process: Capture → Preprocess → Train/Test → Predict**

**Level 1 DFD would dive deeper into sub-processes.**

* **UI based on UX (Figma)**

**Design a simple Figma prototype with:**

* **Input field for digit label**
* **Buttons for each function (Capture, Train, Predict)**
* **Output field for result and accuracy**
* **Make layout clean, color-coded, with guided instructions**

**Testing Strategy and Techniques**

**Types:**

* **Unit Testing (functions: preprocessing, capture, prediction)**
* **Integration Testing (workflow from image capture to prediction)**
* **System Testing (complete GUI workflow)**
* **Manual Testing (usability, image quality, misclassifications)**

**Tools: Python unit test, pytest, plus manual validation of model predictions.**

**Basic Needs for the Project**

These are the foundational requirements you rely on:

* **Hardware**: A computer with reasonable processing power and camera/screenshot capability.
* **Software Tools**:
  + Python and libraries (OpenCV, Scikit-learn, Tkinter, Pandas, etc.)
  + Paint or drawing interface for input
  + Joblib for model storage
  + Figma for UI/UX mockups
* **Data**: Self-created digit images for training/testing
* **Skillset**: Familiarity with Python, image processing, machine learning, and GUI design

**Challenges Faced**

No project is complete without a few bumps along the way. You likely encountered:

* **Data Quality Issues**: Manually drawn digits may vary significantly in thickness, rotation, or clarity.
* **User Workflow**: Having to toggle Paint and the GUI may reduce usability.
* **Data Preprocessing**: Converting to consistent grayscale, thresholding, and resizing can be error prone.
* **Index Errors**: As seen in your debug code, accessing specific dataset rows can result in exceptions.
* **Model Accuracy**: Limited dataset size can reduce SVM effectiveness, especially with highly variable inputs.
* **GUI Limitations**: Tkinter’s simplicity can sometimes limit advanced UX experiences.

**Motivation Behind the Project**

There’s often something deeper that drives us. This project may have emerged from a few powerful motivations:

* **Learning by Doing**: It’s a hands-on way to explore the entire ML lifecycle—data capture, modeling, UI, and UX.
* **Bridging Academics with Real-World Skills**: Perfect for applying software engineering principles to a tangible product.
* **Empowering Local Solutions**: If digit recognition has educational or accessibility value in your context, this project becomes even more meaningful.
* **Portfolio Power**: A full-stack ML project with GUI and custom dataset creation? That’s résumé gold.
* **Curiosity and Creativity**: Let’s face it, you had fun building something your future self might want to improve or expand.

**🧠** Business **Proposal: Intelligent Handwritten Digit Recognition System**

This project introduces Python-based GUI software that recognizes handwritten digits through machine learning. By capturing user-drawn digits, preprocessing the images, training a custom model, and delivering real-time predictions, this tool bridges human inputs with intelligent recognition. It’s designed to demonstrate machine learning deployment in an accessible, practical application.

**Objectives**

* Develop an end-to-end digit recognition application using **Python, OpenCV, Scikit-learn, and Tkinter**
* Allow users to **manually draw**, **label**, and **train models** based on real-time data
* Build a **scalable and modular system** to serve as a learning tool for AI enthusiasts and a prototype for future digit-based recognition systems (e.g., bank check scanners, form readers, mobile number input, etc.)

**Product Description**

* **Data Collection**: Captures handwritten digits using Paint or custom drawing interface
* **Preprocessing**: Image conversion, filtering, and normalization
* **Model Training**: Uses Support Vector Machines (SVM) trained on labeled custom data
* **Prediction**: GUI-based interface outputs recognized digits and model accuracy

**Market Opportunity**

* **Educational Institutions**: For teaching applied machine learning, image processing, and GUI design
* **Developing Regions**: Manual digit entry is still common; this tool automates recognition tasks
* **Customization Potential**: Can be extended to recognize regional scripts or algebraic symbols

**Cost Analysis**

* Development Tools: *Free & Open-Source*
* Estimated Development Time: *64–80 hours*
* Opportunity Cost: ~$1,200 based on $15/hr. developer rate
* Maintenance: Minimal unless extended to the web or mobile

**Timeline**

| **Phase** | **Duration** |
| --- | --- |
| Requirements & Research | 1 week |
| Development & UI | 3 weeks |
| Testing & Documentation | 1 week |
| Final Deployment | 1 week |

**Growth & Expansion**

1. **Canvas Integration**: Replace Paint with a Tkinter-based drawing pad
2. **Cloud Deployment**: Run the model using Flask + React for broader use
3. **Real-Time Recognition**: From camera or mobile stylus
4. **Multilingual Support**: Recognize regional numeral systems (Pashto, Urdu, Dari digits, etc.)

**Testing Strategy**

* **Unit Tests** for image capture and preprocessing functions
* **Integration Tests** for end-to-end pipeline
* **Manual UX testing** to evaluate ease of use and clarity of feedback
* **Model Evaluation** using confusion matrix and accuracy metrics

**Conclusion**

This project doesn’t just recognize digits—it recognizes the need to empower users with visual learning, smart automation, and AI literacy. With the potential for educational, commercial, and accessibility applications, the handwritten digit recognition tool is an impactful showcase of applied software engineering and machine learning.

The **COCOMO II (Constructive Cost Model)** is an advanced and flexible model used to estimate the cost, effort, and schedule for software development, especially modern, iterative projects like yours.

Let’s walk through how to estimate your handwritten digit recognition system using **COCOMO II Post-Architecture Model**, which is ideal since you’ve already defined architecture, tools, and tech:

**🧮 1. Effort Estimation (in Person-Months)**

COCOMO II's core formula:

Effort = A × (Size)^E × Π(EM\_i)

* **A** = 2.94 (default coefficient)
* **Size** = estimated code size in KSLOC (thousands of source lines of code)
* **E** = exponent based on scale drivers
* **EM\_i** = effort multipliers (17 cost drivers)

Let’s break it down with **rough assumptions** for your project:

* **Estimated Size**: ~1,000 lines of code = **1.0 KSLOC**
* **Scale Factors** (average): E ≈ 1.05
* **Effort Multipliers** (let's say nominal): Π(EM\_i) ≈ 1.0

Effort = 2.94 × (1.0) ^1.05 × 1.0 ≈ 2.94 Person-Months

**2. Schedule Estimation (in Months)**

Duration = 3.67 × (Effort)^F

Where **F = 0.28 + 0.2 × (E - 1.0)**

Given E = 1.05:

F = 0.28 + 0.2 × 0.05 = 0.29

Duration = 3.67 × (2.94) ^0.29 ≈ 4.2 months

**Cost Estimation (in USD)**

Assuming 1 developer, $15/hour:

2.94 person-months × 160 hours/month = 470.4 hours

470.4 hours × $15 = ~$7,056

This is purely illustrative—actual cost depends on local wage rates, developer experience, and tool usage.

**Gantt chart**

| **Task** | **Start Date** | **End Date** | **Duration** | **Dependencies** |
| --- | --- | --- | --- | --- |
| Requirements Gathering | Day 1 | Day 3 | 3 days | — |
| UI/UX Design (Figma) | Day 4 | Day 7 | 4 days | Requirements |
| Data Collection & Labeling | Day 5 | Day 10 | 6 days | Requirements |
| Image Preprocessing | Day 8 | Day 11 | 4 days | Data Collection |
| Model Training & Evaluation | Day 12 | Day 15 | 4 days | Preprocessing |
| GUI Development (Tkinter) | Day 8 | Day 14 | 7 days | UI/UX Design |
| Integration & Testing | Day 15 | Day 18 | 4 days | GUI + Model |
| Documentation & Reporting | Day 16 | Day 20 | 5 days | Integration |
| Final Presentation & Review | Day 21 | Day 22 | 2 days | Documentation |

**Steps to Apply CPM**

1. **List of All Activities** Example:
   * A: Requirements Gathering
   * B: UI/UX Design
   * C: Data Collection
   * D: Preprocessing
   * E: Model Training
   * F: GUI Development
   * G: Integration & Testing
   * H: Documentation
   * I: Final Presentation

**2.Estimate Duration for Each Task** (in days or hours)

3. **Identify Dependencies** For example:

* B depends on A
* D depends on C
* G depends on E and F

**4.Draw a Network Diagram** Use nodes for tasks and arrows for dependencies.

5.**Calculate Earliest Start (ES), Latest Start (LS), and Slack** Tasks with **zero slack** are on the **critical path**.

**Why It Matters**

* Help you **prioritize** tasks that can’t be delayed
* Optimize **resource allocation**
* Gives a **realistic timeline** for project completion

**PERT chart**:

[Start]

↓

[A] Requirements Gathering (2d)

↓

[B] UI/UX Design (3d) ──┐

↓ │

[C] Data Collection (5d) ──┐

↓ │

[D] Preprocessing (3d) │

↓ ↓

[E] Model Training (3d) [F] GUI Development (5d)

↓ ↓

[G] Integration & Testing (4d)

↓

[H] Documentation (3d)

↓

[I] Final Presentation (2d)

↓

[End]

Each node represents a task, and arrows show dependencies. You can assign three time estimates to each task:

* **Optimistic (O)**
* **Most Likely (M)**
* **Pessimistic (P)**
* PERT formula to estimate duration:
* Expected Time (TE) = (O + 4M + P) / 6

**Use Case Title: Digit Recognition via GUI Interface**

**1. Actors**

* **Primary Actor**: User (Student, Developer, Educator)
* **Secondary Actor**: ML Model / System

**2. Preconditions**

* System is launched successfully
* Paint or drawing interface is available
* Required Python dependencies are installed

**Use Cases Overview**

| **Use Case ID** | **Name** | **Description** |
| --- | --- | --- |
| UC01 | Capture Digit Image | Captures user’s digit from Paint into the system |
| UC02 | Label & Save Data | Assigns label (0–9) and saves processed image |
| UC03 | Preprocess Image | Converts image to grayscale, resizes, binarizes |
| UC04 | Generate Dataset | Compiles digit images into a CSV dataset |
| UC05 | Train ML Model | Trains classifier using SVM with image dataset |
| UC06 | Predict Digit | Runs real-time digit recognition |
| UC07 | Show Accuracy | Displays model accuracy after evaluation |
| UC08 | Exit Application | Closes system safely |

**Detailed Use Case: UC05 - Train ML Model**

* **Actors Involved**: User, Classifier System
* **Precondition**: Dataset is complete and properly formatted
* **Basic Flow**:
  1. User clicks “Train Model” button
  2. System splits data into training/testing sets
  3. SVM classifier is trained
  4. Model is saved using joblib
  5. Accuracy is displayed
* **Postcondition**: Model is stored and ready for prediction

**Alternate Flows**

* If dataset is insufficient → show error message
* If training fails → prompt user to check dataset or try again

**🎨Figma UI Layout Design: Digit Recognizer**

**🧩 Main Window Layout (Desktop View)**

+------------------------------------------------------+

| Handwritten Digit Recognition (Title) |

------------------------------------------------------

| Label Entry: [ \_\_\_\_\_\_\_\_\_] |

| |

| [🖼️ Open Paint & Capture] [📊 Generate Dataset] |

| |

| [🧠 Train Model] [🔍 Predict Digit] |

| |

| Prediction Output: [ \_\_ 5 \_\_] |

| Accuracy: 96.7% |

------------------------------------------------------

| 📂 Dataset Status: Rows: 250 | Model: Trained ✅ |

+------------------------------------------------------+

**Figma Frame Breakdown**

1. **Top Section**
   * Logo or icon on the left
   * App title centered
   * Light gradient background or clean header bar
2. **From Section**
   * Input field for digit label (0–9)
   * Clear placeholder text like “Enter digit label”
3. **Action Buttons** (with hover states!)
   * Style each button with:
     + Rounded corners
     + Icons (from Figma plugins or Material Symbols)
     + Colors: Orange for capture, Pink for dataset, Blue for train, Green for prediction
4. **Results Panel**
   * Bold, centered prediction output (“Digit: 5”)
   * Accuracy badge or progress bar
   * Dataset summary (row count, trained status)
5. **Footer / Status Bar**
   * Lightweight info area for system tips and updates

**🖌️ Figma UX Style Tips**

* **Font Suggestions**: Inter, Poppins, or Roboto for clean readability
* **Color Palette**:
  + Primary: Indigo or Royal Blue
  + Accent: Orange, Pink, and Teal
  + Background: Light Gray or White
* **Spacing & Alignment**: Use an 8pt grid system for balance
* **Responsiveness**: Create an alternate mobile frame with stacked buttons

+-----------------------------------------------------------+

| Handwritten Digit Recognition |

|--------------------------------------------------------|

| Label Entry: [ \_\_\_\_\_\_\_\_\_\_\_] |

| |

| [🖌️ Open Paint & Draw] [📁 Generate Dataset] |

| |

| [🚀 Train Model] [🔍 Predict Digit] |

| |

| Prediction Output: [ --] |

| Accuracy: [Waiting to train...] |

|--------------------------------------------------------|

| Status: No data loaded | Model: Not Trained 🔴 |

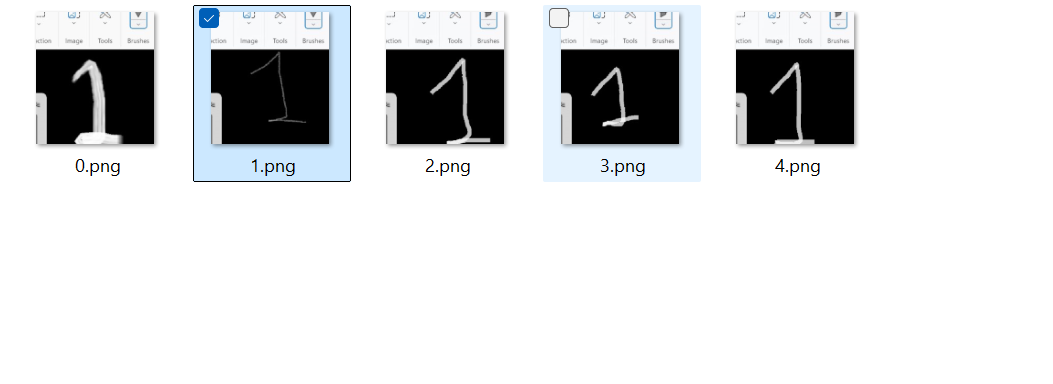
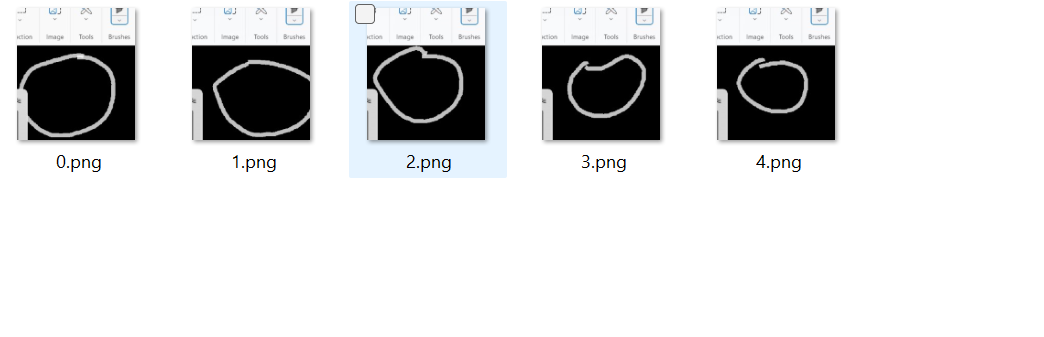
+-----------------------------------------------------------+

Captured images of handwritten digit recognition training the module in python by drawing it in paint program:

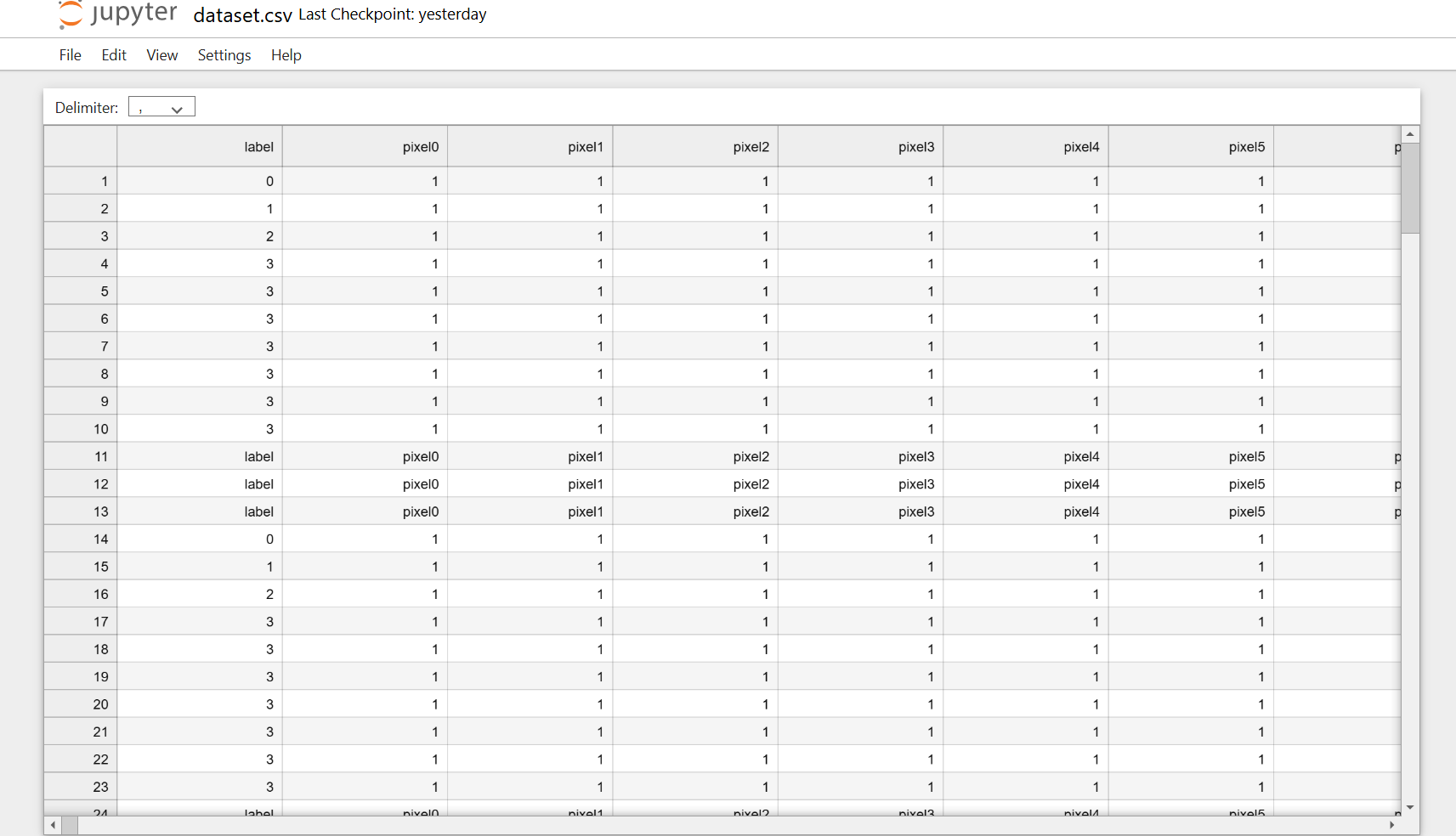
0,1 for five times using loop in python:

images\_folder= "captured\_images/3/"

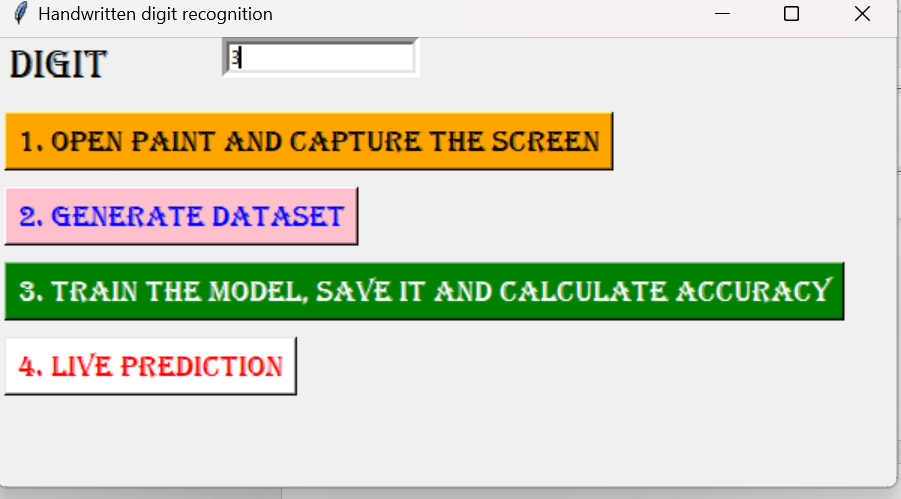
for i in range (0,5):



Generating the dataset in python jupyter :



Converting the module into GUI using python (Tkinter):



|  |
| --- |
| Grantt chart |

Tasks

Week 4

Week 3

Week 2

Week1

|  |
| --- |
| Requirements Gathering |
| UI/UX Design (Figma) |
| Data Collection & Labeling |
| Image Preprocessing |
| Model Training & Evaluation |
| GUI Development (Tkinter) |
| Integration & Testing |
| Documentation & Reporting |
| Final Presentation & Review |

Pert chart: