A blue and white background

Description automatically generated

AI Automation Developer   
Assessment

2025-02-24

**Objective:**

The goal of this assessment is to evaluate your ability to build a machine learning model and an AI agent in Python. You will also be required to document your process, explaining the steps taken, design choices, and the reasoning behind your approach.

**Instructions:**

1. Complete both **Part A** and **Part B** of the assessment.
2. Write clean, well-structured, and well-commented Python code.
3. Provide a written explanation of your approach for each part.
4. Submit your code and documentation in a single file or notebook.

**Part A: Build a Machine Learning Model Using Case Management System Data**

<https://github.com/kanuarj/TitanicMLandFlaskDeployment>

Multi-class Classification by Decision Tree: <https://www.kaggle.com/code/gizemtanriver/multi-class-classification-by-decision-tree>

**Task:**

Develop a machine learning model to predict case outcomes based on data from a **case management system**. The dataset should be structured with at least **three related tables**.

**Dataset Requirements:**

If you do not have access to a real case management system, generate a synthetic dataset that mimics real-world case management data. The dataset should include:

1. **Cases Table**: Contains case details (case ID, case type, status, assignee, priority, creation date, resolution time, description, notes, outcome)
2. **Clients Table**: Contains client details (client ID, name, age, risk level, previous cases, etc.).

Take example of healthcare management system (<https://www.kaggle.com/datasets/anouskaabhisikta/healthcare-management-system>) as example to build such table Cases Table 🡪 Appointments Table, Clients Table 🡪 Patients Table . Download from Kaggle website

**Requirements:**

* **Data Preprocessing**: Merge/join tables appropriately (pd.merge()), handle missing values (<https://www.geeksforgeeks.org/working-with-missing-data-in-pandas/> or https://www.geeksforgeeks.org/imputing-missing-values-before-building-an-estimator-in-scikit-learn/), normalize/scale features (<https://www.geeksforgeeks.org/how-to-normalize-data-using-scikit-learn-in-python/>) , and encode categorical variables (https://www.geeksforgeeks.org/encoding-categorical-data-in-sklearn) if needed.
* **Model Selection**: Train a predictive model (e.g., decision tree model) to predict **case resolution status** (e.g., resolved, pending, escalated). (<https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial>)
* **Evaluation**: Assess model performance using accuracy, precision, recall, and F1-score. (<https://www.kaggle.com/code/haseebwar07/build-a-decision-tree-with-python>)
* **Feature Importance Analysis**: Identify the most influential factors affecting case resolution. (https://www.geeksforgeeks.org/understanding-feature-importance-and-visualization-of-tree-models/)

**Bonus Points:**

* Use SQL or Pandas to extract and preprocess the data efficiently.
* Implement a **time-based** train-test split (e.g., train on older cases, test on newer ones). (<https://stackoverflow.com/questions/50879915/splitting-data-using-time-based-splitting-in-test-and-train-datasets>)
* Deploy the model as an API using Flask or FastAPI. (<https://www.youtube.com/watch?v=MxJnR1DMmsY&t=14s>)

**Part B: Develop an AI Agent for Case Insights (use langchain to create a chatbot that interacts with excel data file)** [**https://www.youtube.com/watch?v=0ModfEMMH2I**](https://www.youtube.com/watch?v=0ModfEMMH2I)[**https://www.youtube.com/watch?v=xQ3mZhw69bc**](https://www.youtube.com/watch?v=xQ3mZhw69bc)

<https://github.com/rajveersinghcse/PetCare-AI-Chatbot>

**Task:**

Create an **AI-powered assistant** that provides useful insights into the case management system data. The assistant should be able to analyse case trends, identify key risk factors, and generate summary statistics.

**Requirements:**

* The AI agent should be implemented as a function or class that interacts with the case management data. (RAG technique)
* The assistant should be capable of answering **key business questions**, such as:
  + What are the most common types of cases? AI-chatbot will retrieve info from model data
  + What factors contribute most to case resolution time? AI-chatbot will retrieve info from model data
  + Which assignees have the highest case resolution rates? AI-chatbot will retrieve info from model data
  + Are there patterns in case escalations based on client demographics or case types? AI-chatbot will retrieve info from model data
* Use **NLP techniques** to analyse notes or case descriptions.
* Provide a written explanation of the approach and implementation.

**Bonus Points:**

* Implement a chatbot interface (e.g., using LangChain or OpenAI's API) that allows users to query the case management system in natural language. ( <https://www.geeksforgeeks.org/build-chatbot-webapp-with-langchain/>**)**

**Submission Requirements:**

* Submit all Python scripts or a Jupyter Notebook (.ipynb) containing your code.
* Include a **README or Markdown** file with explanations of your approach and steps taken.
* Ensure your code is well-commented and modular.
* from langchain\_community.embeddings import HuggingFaceEmbeddings
* from langchain\_community.document\_loaders.hugging\_face\_dataset import (HuggingFaceDatasetLoader,)
* from langchain.indexes import VectorstoreIndexCreator
* from langchain.chains import RetrievalQA
* from langchain\_experimental.agents.agent\_toolkits.csv.base import create\_csv\_agent
* from langchain.llms import OpenAI
* # -- Part B: Develop an AI Agent for Case Insights --
* os.environ["OPENAI\_API\_KEY"] = "sk-abcdef1234567890abcdef1234567890abcdef12"
* # loader = create\_csv\_agent( OpenAI(), path="./data/model\_data.csv", verbose=True, allow\_dangerous\_code=True)
* # Create an index using the loaded documents
* # embeddings = HuggingFaceEmbeddings()
* # index\_creator = VectorstoreIndexCreator(embedding=embeddings).from\_loaders([loader])
* # docsearch = index\_creator.from\_loaders([loader])
* # Create a question-answering chain using the index
* # chain = RetrievalQA.from\_chain\_type(llm=OpenAI(model="gpt-3.5-turbo"), chain\_type="stuff", retriever=docsearch.vectorstore.as\_retriever(), input\_key="question")
* chain = create\_csv\_agent( OpenAI(), path="./data/model\_data.csv", verbose=True, allow\_dangerous\_code=True)
* # render the template
* @app.route("/")
* def index():
* return render\_template("index.html")
* # Posting the user query
* @app.route("/chat", methods=["POST"])
* def chat():
* user\_input = request.form["user\_input"]
* result = chain({"question": user\_input, "chat\_history": []})
* return result["answer"]

A screenshot of a chatbot

AI-generated content may be incorrect.