Pet‑Match Recommender – Project Outline

1 Project Overview

The goal is to build a **pet‑adopter matching prototype** that shows how generative AI and advanced analytics can raise shelter adoption rates. In four weeks our deliverables will be:

- A Streamlit demo that recommends the top-3 pets for any adopter persona.
- Report detailing the business plan, analytics plan, and effect of the tool

The trick is to treat your synthetic pets table as **item data**, invent a small but consistent **adopter-profile table** plus **positive pairs**, and then train a *content-based* or *metric-learning* recommender.
You can layer an LLM on top to generate friendly "why this pet?" explanations.

---

1 Create an adopter table and pseudo-labels

Step	How to do it	Lines of code
1 a. Build 5–10 adopter personas	Use `faker` to sample age, housing, activity, prior-pet; save 5 000–10 000 rows.	`python\nfrom faker import Faker\nfaker = Faker()\nadopters = [{\n 'adopter_id': faker.uuid4(),\n 'age': faker.random_int(21,70),\n 'housing': faker.random_element(['apt','house','farm']),\n 'activity': faker.random_element(['low','mod','high']),\n 'has_prior_pet': faker.boolean()\n} for _ in range(8000)]`

1 b. Define simple matching rules	*If housing=apt ⇒ prefer "Small" breed; if activity=high ⇒ prefer "Working/Active" breeds*, etc.	Write a Spark UDF that scores (adopter, pet) pairs on 0–1.
1 c. Generate positive pairs	Cross-join adopters × ~1 000 random pets, keep rows where rule-score > 0.7 and label = 1; sample same # for label = 0.	Spark: `adopters.crossJoin(pets_silver.sample(0.05)).withColumn("label", udf_score(...))`

You now have **tens of thousands of labelled (adopter, pet) pairs** for supervised training.

---

2 Model options

Model	Why it works label-light	Tooling
Wide-&-Deep tabular	Handles one-hot (breed, color) + numeric (age). Learns interactions without huge data.	TensorFlow Keras (`tf.estimator.DNNLinearCombinedClassifier`).
Siamese / Triplet net	Learns embeddings for pets & adopters; inference = cosine similarity.	PyTorch, 100 k pairs fit in Colab GPU.

LightFM hybrid	Uses content features + (synthetic) interactions; easy to evaluate Precision@k.	`lightfm` Python package.

Pick one; Wide-&-Deep is easiest.

---

3 Streamlit demo flow

adopter dropdown ➜ model.predictTopK(pet_vecs) ➜

LLM tells the story ➜ user clicks 👍 / 👎 ➜ write new feedback row

LLM explanation

explain_prompt = (

f"You are a pet-placement counselor.\n"

f"Adopter profile: {adopter_json}\n"

f"Pet profile: {pet_json}\n"

f"Explain in one short paragraph why this is a good match.")

response = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=[...])

Cache the explanation along with the recommendation.

---

4 Putting it all in Spark + Streamlit

1. **`01_make_adopters.ipynb`**
   Generate adopter table, write `adopters_silver` to shared drive.

2. **`02_make_pairs.ipynb`**
   Cross-join, score rule, create `train_pairs.parquet` (features + label).

3. **`03_train_wd_model.ipynb`**
   Spark → Pandas sample → TensorFlow training → save `pet_model.h5`.

`app.py` (Streamlit)

import tensorflow as tf, pandas as pd, streamlit as st

model = tf.keras.models.load_model("pet_model.h5")

pets = pd.read_parquet("pets_silver")   # small subset in memory

adopter = st.selectbox("Choose adopter persona", adopters_table['adopter_id'])

topk = rank_pets(model, adopter, pets, k=3)

for pet in topk: st.image(pet['photo']); st.write(explain(pet, adopter))

4.

---

5 Why graders will accept synthetic labels

- You document the rule-based generator and label it **"synthetic for prototyping only."**

- The Streamlit demo is interactive and explainable through the LLM.

That combination ticks the "creative + effective code/tool" rubric