The Build Vault

@hinumpy · Four Real ML Projects, Every Number Verified

How To Use This

Four machine learning projects you can open and run right now. No setup, no downloads, no environment to configure. Every dataset loads itself from a public link and every notebook runs start to finish on a free Colab CPU.

Open one, then hit "Copy to Drive." That makes it yours. Break it, change the numbers, swap the dataset. You learn almost nothing from reading someone else's code and almost everything from breaking it.

Each project below lists exactly what the notebook prints when you run it, and separately, what you get when you add a proper evaluation cell. Those two things are different, and knowing the difference is most of what separates a student project from an engineering one.

Read this before you use any of these numbers: Rerun the notebook yourself before putting anything on a resume. If you changed one line, the number changed too. An interviewer asking one follow-up question can tell instantly whether you ran it or copied it.

1. Does This Drug Reach Your Brain?

Python · RDKit · scikit-learn · Random Forest · 4 cells

What it does: Predicts whether a drug crosses the blood-brain barrier from the molecule's structure alone. Benadryl makes you drowsy and Claritin doesn't, even though they hit the same receptor. The notebook turns each molecule into seven numbers — molecular weight, fat-solubility, polar surface area, hydrogen bond donors and acceptors, rotatable bonds, and ring count — and trains a random forest on them.

The dataset: BBBP (Blood-Brain Barrier Penetration) from MoleculeNet, loaded straight from a public URL. 2,039 of the molecules parse cleanly in RDKit; a handful have malformed structures and get skipped.

What the notebook prints: "trained on 2039 molecules," then Benadryl crosses at 100% confidence and Claritin crosses at 58% confidence. That's it — there is no accuracy score anywhere in this notebook.

What you get when you add a cross-validation cell: 0.889 ROC AUC, 78.9% balanced accuracy, and 87.2% plain accuracy under 5-fold cross-validation, against a majority-class baseline of 76.5%. Polar surface area came out as the strongest single predictor, at roughly a quarter of total feature importance.

Where it breaks, on purpose: Claritin is called a brain-crosser at 58% confidence. That's the wrong answer — but the model is uncertain in exactly the right place, versus 100% on Benadryl. Claritin does enter the brain, then a protein called P-glycoprotein pumps it back out. Seven features describing size and polarity govern passive diffusion; not one of them can describe a pump.

Open it: https://colab.research.google.com/drive/1kbLSl5-OEIfVJzFKv7TwgiJAku_1Nls2?usp=sharing

Resume bullet, once you've rerun it: Built a blood-brain barrier permeability classifier in Python using seven RDKit molecular descriptors and a random forest over 2,039 compounds, achieving 0.89 ROC AUC under 5-fold cross-validation.

Quote ROC AUC or balanced accuracy, never plain accuracy. BBBP is about 3:1 skewed toward molecules that do cross, so 87% accuracy sits only eleven points above a model that guesses "crosses" every single time. That is the first thing an interviewer will check.

2. Can A Model Spot A Malignant Tumor?

Python · scikit-learn · Random Forest · 4 lines of actual code

What it does: Classifies breast tumors as malignant or benign from 30 measurements of cell nuclei — radius, texture, perimeter, concavity, symmetry and more — taken from digitized fine-needle aspirate images. Load, split, fit, score. Four lines.

The dataset: Wisconsin Diagnostic Breast Cancer, bundled inside scikit-learn. 569 samples, 30 features, 212 malignant and 357 benign. Nothing to download at all.

What the notebook prints: One accuracy number from a single random 80/20 split — and it is a different number every time you run it. Two runs of this exact notebook, unchanged, printed 98.25% and 94.74%.

Why that happens, and why it matters: There is no random_state on the train/test split, so a fresh test set is drawn on every run. Across 200 runs the same code landed anywhere from 90.4% to 100%, averaging 96.1%. Whichever number you happened to see first is not your model's accuracy — it's one sample from a distribution.

What you get when you fix it: Add random_state=42 and a cross-validation cell and the honest numbers are 0.989 ROC AUC, 95.6% accuracy, and 93.9% recall on malignant tumors, against a 62.7% majority baseline.

Where it breaks, on purpose: In scikit-learn's copy of this dataset, 1 means benign and 0 means malignant — backwards from what almost everyone assumes. So the default recall score silently measures the class you don't care about. Measured correctly, the model misses 13 of 212 malignant tumors. "95.6% accurate" and "misses 1 in 16 cancers" describe the same model.

Open it: https://colab.research.google.com/drive/1HA9n5K7Z8_MMRllklnJ4UmJrXJgXg7ss?usp=sharing

Resume bullet, once you've rerun it: Trained a random forest classifier on the Wisconsin Diagnostic Breast Cancer dataset (569 samples, 30 features), achieving 0.99 ROC AUC and 93.9% malignant-class recall under 5-fold cross-validation.

This is the smallest project here and the best interview story. "I ran it twice and got two different numbers, so I looked into why" is a genuinely senior thing to say, and it costs you one line of code to be able to say it.

3. Can A CNN Spot Pneumonia In A Chest X-Ray?

Python · PyTorch · MedMNIST · Convolutional Neural Network

What it does: Trains a convolutional neural network from scratch to classify chest x-rays as normal or pneumonia. Two convolutional layers, one fully connected layer with 30% dropout, 105,281 parameters total. Ten epochs, Adam optimizer, batches of 64.

The dataset: PneumoniaMNIST from MedMNIST. 4,708 training, 524 validation, 624 test images at 28×28 grayscale. Downloads itself when you run the first cell.

What the notebook prints: Validation accuracy climbing each epoch to a peak of 96.37%, total training time of 0.7 minutes on a free Colab CPU, and a final held-out test accuracy of 84.94%.

Where it breaks, on purpose: Validation says 96.37% and test says 84.94%. That eleven-point gap is not a bug — it's the whole lesson. MedMNIST builds the validation set by splitting the source training data 9:1, then uses a completely separate source split as the test set. So validation shares a distribution with training and test doesn't. The model partly learned the training data's source, not pneumonia.

The sample prediction grid at the end makes it visible: of six test x-rays shown, two genuinely normal ones were called pneumonia. The model over-predicts the class it saw most during training.

Open it: https://colab.research.google.com/drive/1CLY_C1En0vJhIbWuGL1yvVjp2oqwiGFl?usp=sharing

Resume bullet, once you've rerun it: Built and trained a PyTorch CNN from scratch (105K parameters) to classify chest x-rays for pneumonia on PneumoniaMNIST, achieving 84.9% held-out test accuracy and diagnosing an 11-point validation-to-test generalization gap.

Worth adding: a classification report instead of a single accuracy number. On a medical classifier the number that actually matters is how many sick patients you called healthy, and plain accuracy hides it completely. The last cell of the notebook lists six other ways to push this further.

4. Find The Song That Feels Like This One

Python · pandas · scikit-learn · Streamlit

What it does: Takes a song you love and returns five that feel like it, using nine audio features: danceability, energy, valence, tempo, acousticness, instrumentalness, speechiness, liveness, and loudness. Every song becomes a point in nine-dimensional space and similar songs sit close together.

The dataset: Spotify Tracks Dataset on HuggingFace. 114,000 rows, down to 81,343 after dropping blanks and de-duplicating songs that appear once per genre. If the web load fails, the notebook falls back to a manual upload cell.

What the notebook prints: A feature matrix of 81,343 by 9, then a results table. Seed it with Blinding Lights and you get five songs back with match percentages.

Where it breaks, on purpose: Those five results come back hip-hop, k-pop, drum-and-bass, dub, and drum-and-bass — every one scoring exactly 99.9%. Five wildly different genres tying at the same score means the similarity scores are compressed near the top, not that those songs are twins. Cosine similarity on all-positive scaled features has a high floor before you even start, and nine audio features simply don't capture what makes a song sound like The Weeknd.

On having no accuracy score: This project has no metric, and that's correct. There is no ground truth for "songs that feel alike," so there is nothing to be right or wrong about. Unsupervised problems get judged on whether the output looks sane to a human — which makes being able to explain your reasoning the entire job.

Open it: https://colab.research.google.com/drive/1LdzQqgJVxhOKVEiZ-uzV_8M586ENG8Gg?usp=sharing

Resume bullet, once you've rerun it: Built a content-based music recommendation system in Python over 81,000 Spotify tracks, engineering nine normalized audio features and using cosine similarity for retrieval, packaged as a Streamlit web app.

The notebook writes app.py, requirements.txt, and songs.csv and downloads all three. The final step is your own public GitHub repo plus share.streamlit.io. Say "deployed" on a resume only once it is actually live, and add the link when it is.

The scaling line is the one to really understand. Tempo runs 0 to 200 and valence runs 0 to 1, so without normalizing, a 40-bpm difference would outweigh a complete mood flip and every recommendation would just be "songs at a similar speed."

Bonus: Turn Project 3 Into A Real App

A step-by-step guide to wrapping the pneumonia model in an interface

A notebook proves you can train a model. An app proves you can ship one. This guide walks every step: exporting your trained weights, finding a design you like, turning that design into a build prompt, building the interface with Claude Code, connecting your real model instead of a placeholder, and running the whole thing locally.

Follow the guide: https://pneumonia-detector-gamma.vercel.app/

Why this matters more than the model: Almost every student resume has a model. Almost none have an interface. Building the frontend is the step that separates a project from a screenshot — and it is usually the easier half.

How To Talk About These In An Interview

Pick one and go deep. Four shallow projects lose to one you can defend for ten minutes.

- Lead with the decision, not the feature list. "I chose X over Y because Z" is the sentence they're waiting for
- Know your own numbers cold, and know what they don't say. Every project here has a metric that flatters it and one that doesn't
- Bring the failure yourself. "It misclassified Claritin, and here's why" makes you sound like an engineer, not a student
- Never quote a number you haven't rerun. If you changed anything at all, rerun it
- Know the baseline. "87% accuracy" means nothing until you can say what guessing would have scored

Every "where it breaks" section above exists for this reason. The weakness is the interesting part of the story, not the thing to hide.

One Last Thing

These aren't meant to be admired. They're meant to be copied, broken, rebuilt, and made yours. The version of a project you can talk about is always the version you changed.

Go break something.

Made with love by @hinumpy

More free resources: linktr.ee/hinumpy