Course: MSDS-422
Student: Daniel Balette
Kaggle ID: danielnorthwestern
Kaggle Submissions: submission_rf.csv, submission_pca_flawed.csv, submission_pca_fixed.csv, submission_kmeans.csv

Management / Research Question:

Can we automatically recognize handwritten digits (0–9) from 28×28 grayscale images quickly and accurately? People care because robust digit recognition underpins real-world tools—bank check readers, mail sorting, and OCR pipelines. The secondary question is how dimensionality reduction (PCA) and unsupervised clustering (K-Means) compare to a strong supervised baseline in accuracy and speed.

Data Preparation, Exploration, and Visualization:

- Data: Kaggle MNIST CSVs (42,000 labeled training rows; 28,000 test rows; 784 pixel columns).
- Integrity: No missing values; pixel ranges 0–255.
- Reshaping/visuals (optional in appendix): Verified 28×28 layout and spot-checked a few digits.
- Scaling: Not required for Random Forest (RF). For PCA, features are centered internally; pixels share the same scale, so standardization is not critical here.

Model Development:

1) Random Forest (baseline)

- Trained RF on all 784 pixel features.
- Elapsed fit time: ~1.54 s.
- Kaggle public score: 0.96582.

2) PCA —> Random Forest (as originally specified - flawed)

- Design per prompt: Fit PCA on the combined train+test to retain 95% variance —> 154 components.
- Trained RF on these components.
- PCA time: ~0.89 s; RF fit: ~4.68 s.
- Kaggle score: 0.94475.

3) PCA —> Random Forest (corrected, no leakage)

- Fix: Fit PCA only on training data, then transform test data with the learned projection (still 154 comps).
- PCA time: ~0.50 s; RF fit: ~4.65 s.
- Kaggle score: 0.94407.

4) K-Means clustering (unsupervised)

- Ran KMeans(n_clusters=10) on PCA features (154 comps), assigned cluster labels by training-set majority vote.
- Clustering time: ~4.05 s.
- Kaggle score: 0.59092 (expected to be much lower since labels are not used in training).

Model Evaluation:

Accuracy: The RF baseline was best (0.96582). Both PCA+RF variants scored ~0.944, showing a modest accuracy drop with dimensionality reduction. K-Means, as expected, was far lower (0.59092).

Speed: PCA+RF adds ~0.5–0.9 s for PCA and ~4.6–4.7 s to fit RF (still fast overall). The baseline RF is the fastest end-to-end for training.

Feature Reduction: PCA compressed 784 —> 154 features (95% variance), confirming substantial dimensionality reduction.

Experimental Design Flaw & Fix:

Flaw: The assignment's original PCA step learns components on the combined train and test sets, which leaks information from the test distribution into feature construction. This violates a proper training/test regimen and can inflate results.

Fix: Fit PCA only on the training data. Save the fitted transformer and apply pca.transform() to the test set. I re-ran the PCA+RF pipeline with this fix and resubmitted to Kaggle (submission_pca_fixed.csv).

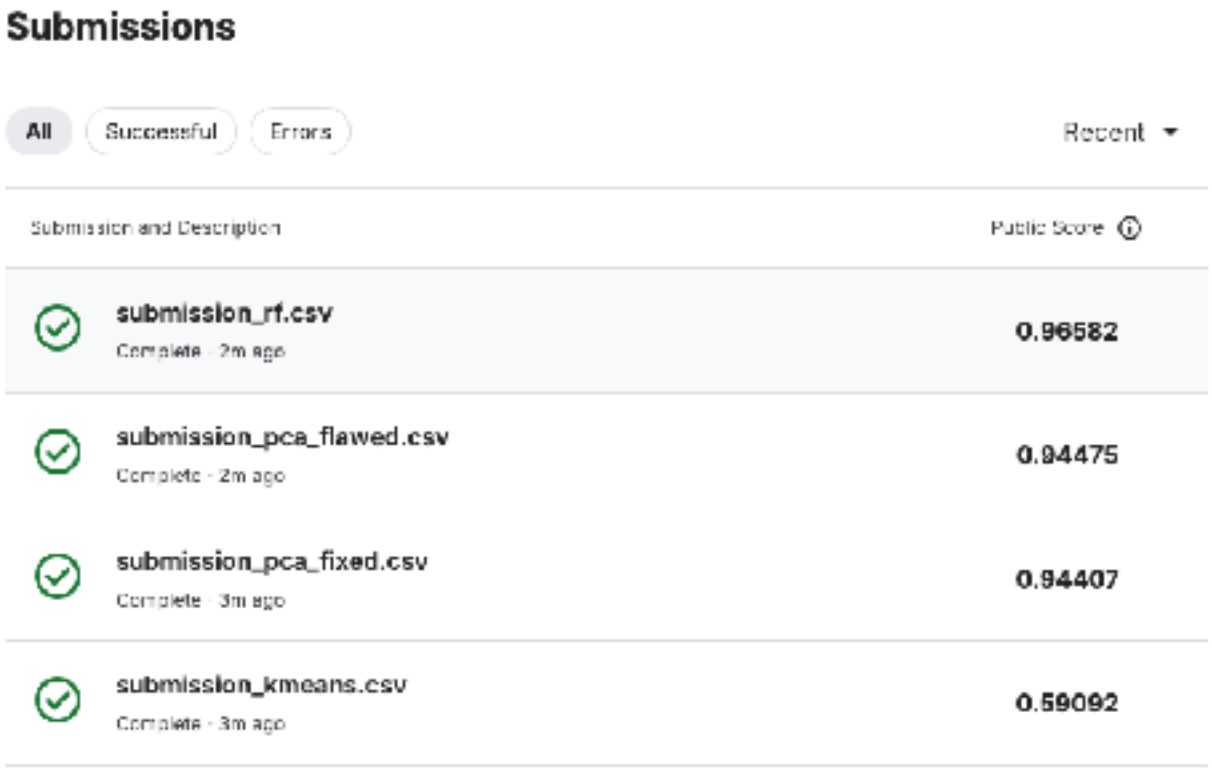
Insights:

- Baseline RF is a strong benchmark on MNIST tabular pixels—simple, fast, and high-accuracy without heavy preprocessing.
- PCA trades a bit of accuracy for efficiency. Reducing to 154 comps preserved 95% variance but cost ~2.2 points of Kaggle accuracy relative to raw pixels. In return, the representation is more compact and may help some algorithms generalize or train faster at larger scales.
- Unsupervised does not equal competitive here. K-Means is useful as a teaching contrast but not a contender for labeled digit recognition.

- Methodology matters. Preventing data leakage (fitting PCA on train only) is essential. The corrected pipeline produced nearly the same score as the flawed run—proof that correctness can be achieved without sacrificing runtime.

Kaggle Results:

- RF baseline: 0.96582 (submission_rf.csv)
- PCA+RF (flawed): 0.94475 (submission_pca_flawed.csv)
- PCA+RF (fixed): 0.94407 (submission_pca_fixed.csv)
- K-Means: 0.59092 (submission_kmeans.csv)



What I would try next:

- Hyperparameters: RF with more trees, max_features='sqrt'/'log2', depth tuning, class balancing.
- Alternative models: A simple CNN (e.g., LeNet-style) typically exceeds 0.99 on MNIST.
- Augmentation: Minor affine transforms could improve generalization for margin cases.