Pete Gerdsen Project 1: Enron Word Count (Map → Shuffle → Reduce);
Approach: Pure Python (local)

Dataset & Subset

I processed of ~517k Enron emails due to local resource limits; results are representative and reproducible via the same selection method.

Results:

- (Top 20) please,137574, would,116308, energy,109133, power,107913, new,107235, said,98068 more,87391, image,84148, time,81827, about,77569, one,77356, gas,76223, which,75917 company,75682, out,73827, thanks,71049, know,69455, information,66297, get,66169 market,66118
- Cumulative tokens in Top-20: 1,725,259
- Vocabulary size: 441,626 unique words

Methods: (aka Describing my approach):

preprocessing (parse_enron.py)

- Prefer text/plain bodies; fallback to text/html with tag stripping.
- Strip quoted replies (drop lines starting with > and sections like "Original Message").
- Normalize whitespace and remove obvious boilerplate fragments where feasible.
- Deduplicate by SHA-1 of the cleaned body (avoids counting long forward/reply chains repeatedly).
- Write cleaned docs to data/plain_v2/, then point inputs → data/plain_v2.

Word Count Pipeline (main.py) — explicit Map → Shuffle/Combine → Reduce

- MAP: Read each .txt email, lowercase, remove emails/URLs/domains, strip non-ASCII, tokenize on [A–Z/a–z/'], drop tokens ≤2 chars, apply general + Enron-specific stopwords (e.g., *enron, ect, hou*, months/days). Emit (word, 1) pairs.
- SHUFFLE / COMBINE: For each file, aggregate pairs into per-file term frequencies using plain dict increments.
- REDUCE (two-stage):
    - Merge local counts into a batch accumulator; spill partials to partials/part_XXXX.csv every N files to bound memory.
    - Final global reduce: read all partial CSVs and sum counts into global totals.
- Outputs:

- outputs/top20.csv — the 20 most frequent tokens (header + 20 rows)
- outputs/word_counts.csv — full vocabulary (sorted by frequency

Reflection: The final counts come from a cleaned, deduplicated subset of **236,244** emails out of ~**517,401** raw messages (≈45.66% retained). The **Top-20** terms—*please (137,574), would (116,308), energy (109,133), power (107,913), new (107,235), said (98,068), more (87,391), image (84,148), time (81,827), about (77,569), one (77,356), gas (76,223), which (75,917), company (75,682), out (73,827), thanks (71,049), know (69,455), information (66,297), get (66,169), market (66,118)*—together account for **1,725,259** tokens, drawn from a vocabulary of **441,626** unique words. This profile looks like internal corporate email rather than spam: it is dominated by request/coordination language (*please, would, thanks, time, get, know*) and clearly reflects Enron's domain via sector terms (*energy, power, gas, market*). A few artifacts persist: *image* rises because HTML signatures/logos survive HTML→text conversion as placeholders, and thread behavior (forward/reply chains) can repeat prior content even after body-hash deduplication. Generic high-frequency tokens (*one, which, about, out*) remain common in email prose. Overall, the counts point to routine coordination and substantial discussion of energy markets, which is consistent with expectations for the Enron corpus.

Improvement : To push the analysis further, I would first strengthen **thread-aware deduplication** by detecting quoted blocks (From:/Sent:/To: headers and ">" lines) and using Message-ID/References to collapse reply chains so only the newest content segment is counted. Second, I would add **normalization** (lemmatization or careful stemming) so inflected forms merge (e.g., *markets → market; companies → company*), and slightly expand the **domain stoplist** to exclude residual rendering artifacts (e.g., *image*) and organization-specific boilerplate. Third, I would move beyond unigrams to **phrases and weighting**—compute bigrams/trigrams (e.g., *natural gas, power trading, credit risk*) and apply **TF-IDF** or sublinear TF so generic words are down-weighted while domain terms and phrases are surfaced. Finally, for scale and clarity, I would keep the current two-stage Reduce but write **time-sliced partials** (e.g., monthly) before the final merge; this preserves reproducibility, enables trend/topic-drift analysis, and makes re-runs faster if only a slice changes.