Case Study: noorAI Project

A Self-Directed R&D Project in AI & Natural Language Processing

Author: Zakariae Mandouli
Date: August 2025

1. Project Objective

noorAI is a self-directed R&D project designed to analyze the text of the Quran through a unique, dual-layered approach. The primary goal was to engineer an end-to-end data pipeline that transforms the sacred text into a structured, multi-dimensional dataset. This dataset is then used to explore potential numerical and linguistic patterns using a combination of traditional numerology (Abjad system), unsupervised machine learning, and advanced Natural Language Processing (NLP).

This initiative served as a practical application of modern AI techniques to a classical, unstructured text, demonstrating a full-cycle approach from data engineering and validation to model implementation and analysis.

2. Technology Stack

- **Environment:** Google Colab
- **Core Language:** Python
- **Data Manipulation:** Pandas, NumPy
- **Machine Learning:** Scikit-learn (K-Means Clustering, StandardScaler)
- **Natural Language Processing (NLP):**
  - **Morphological Analysis:** CAMeL Tools
  - **Text Processing:** PyArabic
- **Data Visualization:** Plotly, Matplotlib, Seaborn

3. Methodology & Key Code Snippets

The project involved a multi-stage data pipeline to ensure accuracy and robustness.

A. Data Processing & Abjad Calculation

A robust function was developed to handle the complexities of the Arabic script and accurately calculate the Abjad value for any given text.

```
# THE DEFINITIVE, 100% CORRECT ARABIC ABJAD MAP
abjad_map = {
    'أ': 1, 'ا': 1, 'ء': 1, 'آ': 1, 'إ': 1, 'ئ': 1, 'ؤ': 1, 'ٱ': 1,
    'ب': 2, 'ج': 3, 'د': 4, 'ه': 5, 'و': 6, 'ز': 7, 'ح': 8, 'ط': 9, 'ي': 10,
    'ك': 20, 'ل': 30, 'م': 40, 'ن': 50, 'س': 60, 'ع': 70, 'ف': 80, 'ص': 90,
    'ق': 100, 'ر': 200, 'ش': 300, 'ت': 400, 'ث': 500, 'خ': 600, 'ذ': 700,
    'ض': 800, 'ظ': 900, 'غ': 1000
}
def get_abjad_value(text):
    total_value = 0
    for char in text:
        if char in abjad_map:
            total_value += abjad_map[char]
    return total_value
```

B. Linguistic Root Analysis (Lemmatization)

To enable deep linguistic analysis, the state-of-the-art CAMeL Tools library was used to find the linguistic root (lemma) of every word in the text. This required initializing a pre-trained morphological analyzer.

from camel_tools.morphology.database import MorphologyDB
from camel_tools.morphology.analyzer import Analyzer

```
# --- Part 4: Augment with Word Roots ---
print(">>> Part 4: Augmenting DataFrame with Word Roots...")
db = MorphologyDB.builtin_db('calima-msa-r13')
analyzer = Analyzer(db)
unique_words = word_df['word_text'].unique()
print(f"Analyzing {len(unique_words)} unique words...")
def get_root(word):
    analyses = analyzer.analyze(word)
    return analyses[0].get('lex', word) if analyses else word
```

C. Unsupervised Machine Learning for Pattern Discovery

An unsupervised machine learning model was implemented to automatically discover non-obvious patterns in the data.

- **Algorithm:** The **K-Means clustering** algorithm from Scikit-learn was selected. Its goal is to partition the 6,000+ verses into a pre-defined number of clusters (k), where each verse belongs to the cluster with the nearest mean (centroid).
- **Feature Set:** The model was trained on a multi-dimensional "numerical fingerprint" for each verse, including its word count, total Abjad value, mean word value, and standard deviation.
- **Preprocessing:** Before training, the features were scaled using StandardScaler to ensure that no single feature (like the large total_abjad_value) disproportionately influenced the clustering outcome.
- **Tuning & Visualization:** The number of clusters (n_clusters) was treated as a tunable hyperparameter, with experiments run at k=10 and k=100. The high-dimensional results were visualized in 2D using the **t-SNE** dimensionality reduction technique.

```
# The only change is 'ayah_number' to 'verse_number' in the print statement
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

# Select the numerical features that make up the "fingerprint"
features = ['word_count', 'total_abjad_value', 'mean_word_value',
            'std_dev_word_value', 'max_word_value', 'min_word_value']
verse_features_df = verse_df[features]

# Scale the features. This is crucial for K-Means to work correctly.
scaler = StandardScaler()
scaled_features = scaler.fit_transform(verse_features_df)

# Run the K-Means clustering algorithm to find 10 clusters.
kmeans = KMeans(n_clusters=10, random_state=42, n_init=10)
verse_df['cluster'] = kmeans.fit_predict(scaled_features)
```

4. Custom Exploratory Analysis Functions

To move beyond standard analysis, a suite of custom functions was designed to probe the data for specific, non-obvious patterns. These functions serve as the core analytical instruments of the project.

- **find_root_sum_combinations()**: This function analyzes the frequency of all word roots to discover if the occurrence count of a major root is the exact sum of the counts of several less frequent roots.
- **find_approximate_sum_for_word()**: A more flexible version of the sum function that finds combinations of root frequencies that sum to a value *within a given error margin* (e.g., 10%) of a specific target word's frequency.
- **find_root_digit_sequence_matches()**: This function searches for numerological coincidences by identifying if the ratio between the occurrence counts of any two roots matches the digit sequence of a special number.
- **find_constants_in_quran_string()**: A novel function that creates a single, massive "Grand Number String" by concatenating the Abjad value of every word in the Quran, and then scans this string for the digit sequences of special constants.

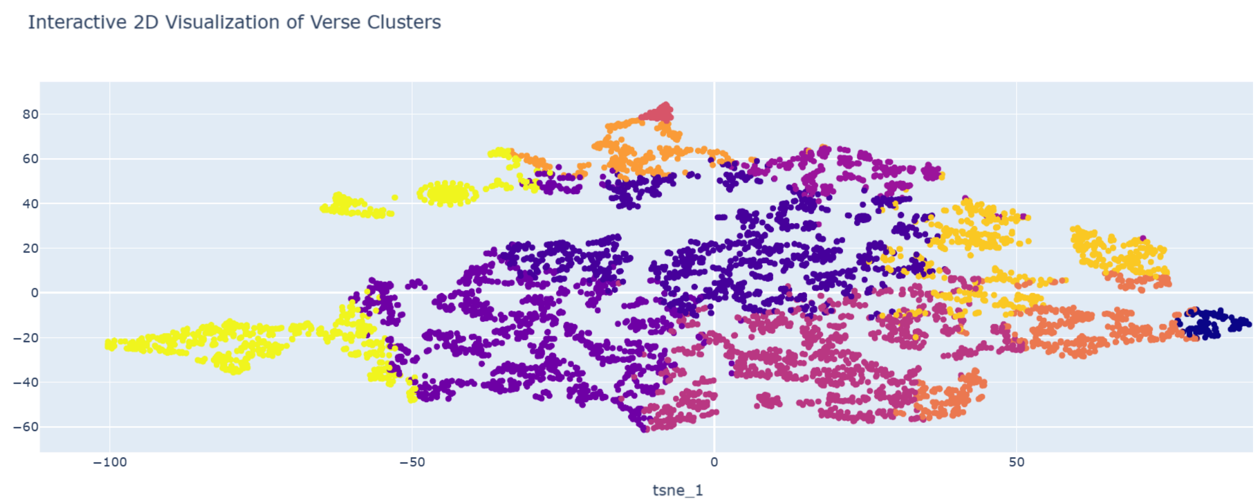
The **Special Constants** used for the ratio and sequence matching functions include:

- **Mathematical Constants:**
  - Pi (π) ≈ 3.14159
  - Euler's Number (e) ≈ 2.71828
- **Physical Constants (numerical values):**
  - Fine-Structure Constant (α) ≈ 7.297×10^{-3}
  - Speed of Light (c) ≈ 2.997×10^{8} m/s
  - Planck Constant (h) ≈ 6.626×10^{-34} J·s
  - Gravitational Constant (G) ≈ 6.674×10^{-11} $m^3 \cdot kg^{-1} \cdot s^{-2}$

5. Results & Visualizations

Results will be added as soon as patterns emerge, as the project is ongoing.

The analysis phase involves using the generated data to explore the clusters and mathematical relationships. Key outputs include interactive visualizations of the verse clusters, allowing for a deep dive into the data.



Data Validation: The data pipeline's accuracy was rigorously validated against established benchmarks, such as the Abjad value of the Basmala (786) and other key phrases, confirming the reliability of the results.

```
validate_known_values()

--- Running Validation Against Established Benchmarks ---

1. Basmala (S1:V1):
   - Our Calculated Value: 786
   - Established Value: 786
   - Result: ✅ Match

2. The word 'Allah' (الله):
   - Our Calculated Value: 66
   - Established Value: 66
   - Result: ✅ Match

3. The word 'Al-Hadid' (ٱلْحَدِيد):
   - Our Calculated Value: 57
   - Established Value: 57
   - Result: ✅ Match

4. Surah Al-Ikhlas (Letter-by-letter value):
   - Our Calculated Value: 1788
   - This is for internal consistency. You can now use this value for your creative analysis.
```

6. Conclusion

The noorAI project successfully demonstrates the ability to engineer a complex data pipeline for a non-standard, unstructured text. It showcases practical, hands-on skills in Python, data validation, unsupervised machine learning, and advanced Arabic NLP. The suite of custom analysis functions provides a rich foundation for further exploratory analysis and pattern discovery.