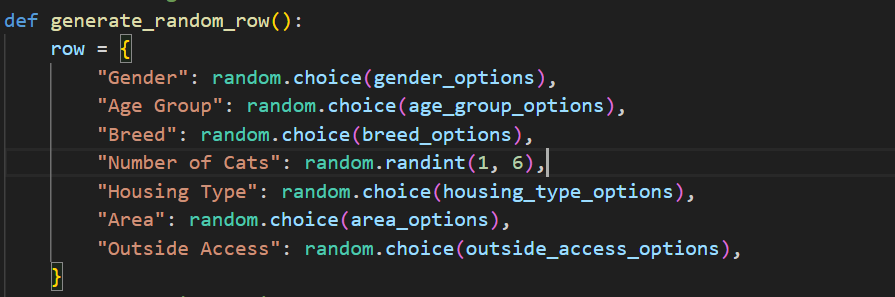
**Project Documentation: Catology**

This project implements a system capable of identifying a cat's breed based on natural language descriptions. The main functionalities include expanding the dataset with new instances, extracting relevant attributes from textual descriptions, identifying breeds using machine learning classifiers, and generating descriptive and comparative textual outputs about cat breeds.

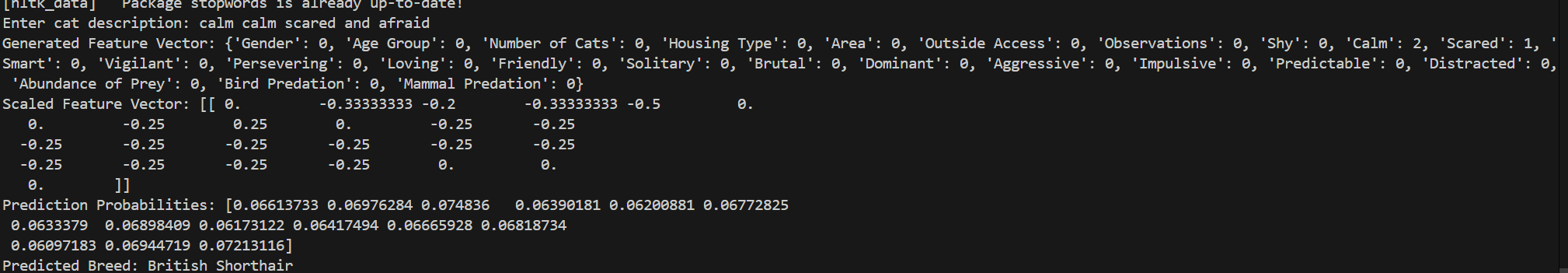
**Main Scripts and Their Roles**

**1. add\_entities.py**

This script is responsible for expanding the dataset by generating synthetic entries to ensure model robustness.

**Key Features**

* **Data Cleaning**: Drops irrelevant columns (Row.names, Timestamp, and Additional Info) from the dataset.
* **Randomized Data Generation**: Synthesizes new data entries for attributes such as gender, age group, breed, housing type, and behavioral traits. Each attribute follows predefined options or ranges.
* **Large-Scale Augmentation**: Generates 5000 new entries and appends them to the cleaned dataset.
* **Output**: Saves the expanded dataset to a new Excel file, ensuring seamless integration into subsequent training processes.
* The script outputs a cleaned and augmented dataset, Updated\_Cats\_database.xlsx, ready for analysis and machine learning tasks.

**2. breed-identify.py**

This script handles the identification of cat breeds from textual descriptions.

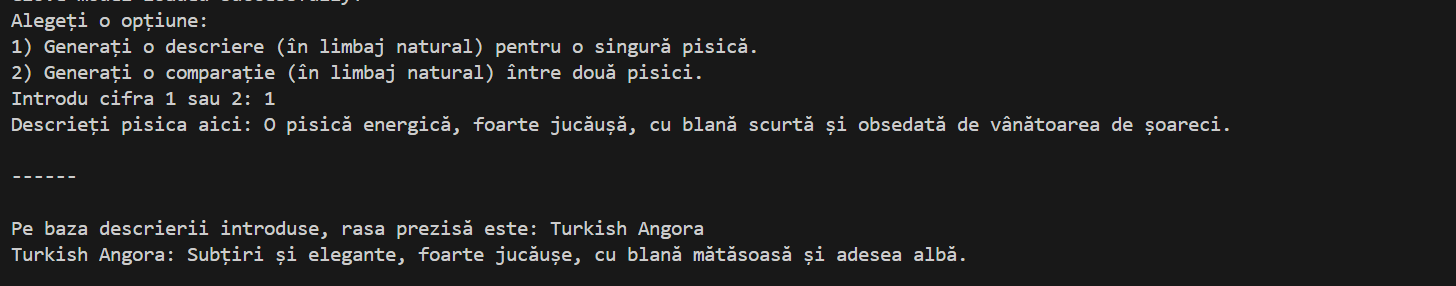
**Key Features**

* **Word Embedding Integration**: Utilizes pre-trained GloVe embeddings to convert text into a 50-dimensional vector representation.
* **Synonym Expansion**: Dynamically enhances tokens by adding synonyms, improving the model's understanding of varied linguistic expressions.
* **Data Preprocessing**:
  + Builds a combined feature vector using numeric and textual inputs.
  + Scales features using a previously fitted scaler.
* **Machine Learning Inference**: Applies a trained neural network model to predict the most likely breed.
* **Interactive Input**: Users provide a description of a cat, and the script outputs the predicted breed with probabilities.

**Usage**

python breed-identify.py

The script predicts and prints the breed name based on the description entered by the user.

**3. breed-nlp-generation.py**

This script generates natural language descriptions for cat breeds and comparative insights between two breeds.

**Key Features**

* **Natural Language Processing**:
  + Processes user inputs to predict breeds using the same pipeline as breed-identify.py.
  + Retrieves pre-defined descriptions for each breed in Romanian.
* **Description Generation**:
  + Provides detailed descriptions of the identified breed based on classification results.
* **Comparative Analysis**:
  + Compares two user-provided descriptions to highlight differences between their respective breeds.

**Usage**

python breed-nlp-generation.py

Users can choose between generating a description for a single breed or a comparative analysis of two breeds.

**System Workflow**

1. **Dataset Preparation**:
   * Use add\_entities.py to expand the dataset with synthetic data.
2. **Model Training**:
   * Train the neural network using the expanded dataset.
   * Save model parameters for inference.
3. **Breed Identification**:
   * Use breed-identify.py to classify breeds based on user input.
4. **NLP-Based Description**:
   * Use breed-nlp-generation.py to generate breed-specific or comparative descriptions.

**Potential Enhancements**

* Integrate additional language support for broader accessibility.
* Enhance the synonym expansion module with domain-specific terminology.
* Incorporate real-time feedback loops for model refinement using user corrections.

**Neural Network Training (nn\_training\_2.py)**

**Technologies Used**

1. **Pretrained Word Embeddings**:
   * **GloVe (Global Vectors for Word Representation)**: GloVe's 50-dimensional vectors are used to convert textual data into numerical embeddings. This captures semantic relationships between words, enabling the model to understand text descriptions more effectively.
   * **gensim**: A Python library is employed to download and manage the GloVe model.
2. **Text Processing**:
   * **NLTK**: Used for tokenization, stopword removal, and synonym expansion via WordNet. This enhances the textual data representation.
3. **Machine Learning Frameworks**:
   * **scikit-learn**: Used for data preprocessing (e.g., scaling and encoding) and managing train-test splits.
   * **imblearn (SMOTE)**: Oversamples minority classes to address imbalances in the dataset, improving the model's ability to generalize.
4. **Data Handling and Plotting**:
   * **Pandas**: For managing and preprocessing tabular data.
   * **Matplotlib**: For visualizing training loss during the model training process.

**Neural Network Implementation**

The neural network consists of the following components:

1. **Input Layer**:
   * Combines numeric data (k features) and GloVe embeddings (50 dimensions) into a single feature vector of size k+50.
2. **Hidden Layers**:
   * Two hidden layers with adjustable sizes (default: 10 neurons each).
   * **Activation Function**: Sigmoid is used in hidden layers to introduce non-linearity.
3. **Output Layer**:
   * Uses a softmax activation to provide probabilities for each class in a multi-class classification problem.
4. **Dropout**:
   * Applied in the first hidden layer to reduce overfitting by randomly setting some neuron activations to zero during training.
5. **Optimization and Regularization**:
   * **Loss Function**: Cross-entropy loss is used for multi-class classification.
   * **Backward Propagation**: Gradient descent is applied to update weights and biases using L2 regularization to avoid overfitting.

**Role of SMOTE (Synthetic Minority Oversampling Technique)**

1. **Problem Addressed**:
   * Imbalanced datasets can lead to biased models, favoring majority classes. This affects the prediction accuracy for minority classes.
2. **Implementation**:
   * SMOTE generates synthetic samples for minority classes by interpolating between existing samples, ensuring a more balanced class distribution.
3. **Benefits**:
   * Improves model performance on underrepresented classes.
   * Enhances generalization by providing diverse training data.

**Training Process**

1. **Data Preparation**:
   * Numeric and text features are combined into a single input vector.
   * Target labels are one-hot encoded for multi-class classification.
2. **Scaling**:
   * All features are normalized using MinMaxScaler to ensure uniformity in value ranges.
3. **Training Loop**:
   * The network is trained over multiple epochs (default: 500).
   * Weights and biases are updated using backpropagation after each epoch.
4. **Evaluation**:
   * Loss and accuracy are monitored on both training and testing datasets.
   * Results are visualized to ensure the model converges appropriately.

**Key Outputs**

1. **Model Persistence**:
   * Trained weights, biases, and the scaler are saved as .pkl files for future use in inference tasks.
2. **Performance Metrics**:
   * The script provides metrics such as test loss and accuracy to evaluate model performance.
3. **Visualization**:
   * A plot of training loss over epochs helps in diagnosing issues like overfitting or underfitting.