Objectives

1. Solve a **binary classification problem** to determine if an image represents a Pokémon.
2. Build an automated, modular workflow using Python for scalability and reproducibility.
3. Evaluate the model with comprehensive performance metrics like accuracy, precision, recall, F1 score, and AUC.

Detailed Workflow and Python Implementation

1. Problem Framing

- The project defines the classification task, specifying performance metrics that measure the success of the model.
- Python is used to set up dynamic computations of metrics during training and testing, enabling real-time monitoring of model performance.

2. Data Gathering

- The notebook automates the collection of datasets:
  - Pokémon images and species data are fetched from **PokeAPI**.
  - Negative examples are sourced from a Digimon dataset scraped from a wiki.
- A **configuration file (`paths.env`)** simplifies dataset management. Python scripts dynamically load file paths, allowing seamless transitions between environments or datasets.

3. Data Exploration

- Python libraries like Pandas and Matplotlib facilitate initial exploration of the datasets. Key tasks include:
  - Summarizing data to check for missing values, class distributions, and other anomalies.
  - Generating visualizations (e.g., histograms, scatter plots) to understand the relationships between features.
- These steps ensure the data is clean, well-structured, and suitable for model training.

4. Data Preparation

- The code processes raw data into a format compatible with deep learning models:
  - Images are resized to a uniform shape and normalized for pixel intensity values.
  - Data is split into training, validation, and testing subsets.
- These preprocessing steps are automated using Python functions, making the pipeline reproducible and efficient.

5. Model Development

- The notebook builds a convolutional neural network (CNN) using **TensorFlow and Keras**, optimized for image classification. Key components include:
  - Convolutional layers to extract spatial features from images.
  - Pooling layers to reduce dimensionality and computational cost.
  - Fully connected layers for binary classification (Pokémon or not).
- The model is compiled with an **Adam optimizer** and a **binary cross-entropy loss function**, ensuring optimal performance on the task.

6. Model Training

- The training process is automated with Python scripts that:
  - Iterate through multiple epochs, updating weights to minimize the loss.
  - Track metrics like accuracy and loss dynamically, storing them for later analysis.
  - Implement callbacks to save the best-performing model and prevent overfitting through early stopping.

7. Model Evaluation and Testing

- Python scripts compute evaluation metrics on the test dataset to assess the model's performance.
- Results are presented using visual tools like confusion matrices and precision-recall curves, offering insights into strengths and weaknesses.
- Additional performance metrics such as F1 score and AUC are calculated to evaluate the model's overall effectiveness.

8. Visualization and Interpretation

- Python scripts generate detailed plots to track training progress (e.g., loss curves) and evaluate results.
- Visualization tools help identify issues such as overfitting or poor generalization, guiding improvements in model design or training strategy.

9. Reusability and Modularity

- The notebook employs modular Python functions for repetitive tasks like data loading, preprocessing, and metric computation.
- The configuration-based setup ensures the workflow can easily adapt to new datasets or related classification tasks.