BEC/Phishing Detection Project Workflow

This project is a classic end-to-end machine learning pipeline designed to classify incoming emails as either **Legitimate (0)** or **Malicious (1)**, focusing specifically on Business Email Compromise (BEC) and phishing indicators.

The workflow is broken down into four sequential stages, managed across the four Python scripts:

1. Data Generation (Source: `data_simulator.py`)

This initial stage creates a synthetic, structured dataset that simulates the critical features of BEC and phishing emails.

- **Goal:** To provide structured, labeled data (`simulated_emails.csv`) for the model to learn from.
- **Key Features Simulated:**
  - `urgency_score`: Measures manipulative language (e.g., "ASAP," "Urgent").
  - `domain_similarity_score`: Measures how closely the sender's domain matches a legitimate domain (to catch typosquatting).
  - `financial_keyword_count`: Counts terms like "wire transfer," "invoice," or "bank details."
  - `request_type`: Categorical feature for the type of deceptive request (e.g., wire transfer, credential update).
  - `sender_anomaly`: Indicates if the sender's email address is new or anomalous.

2. Feature Engineering & Preprocessing (Source: `feature_engineer.py`)

This stage transforms the raw data into a format suitable for the Random Forest model.

- **Goal:** Clean, encode, and split the data.
- **Key Steps:**
  - **One-Hot Encoding:** The categorical `request_type` feature is converted into numerical columns (e.e. `req_1`, `req_2`) so the model can process it.
  - **Data Splitting:** The dataset is divided into 80% for training the model and 20% for testing its performance (`X_train`, `X_test`, `y_train`, `y_test`).

3. Model Training and Tuning (Source: `train_model.py`)

This is the core machine learning stage where the classification model is built and optimized.

- **Model Used:** Random Forest Classifier (an ensemble method suitable for classification).
- **Critical Tuning Step (Class Weight):** After the initial training phase showed low **Recall** (missing too many threats), we introduced `class_weight={0: 1, 1: 5}`. This instructs the model that misclassifying a Malicious email (False Negative) is 5 times worse than misclassifying a Legitimate email (False Positive). **This shift prioritizes security (high Recall) over perfect precision.**
- **Output:** The trained model is saved as `model.pkl`.

4. Model Deployment and Prediction (Source: `predict_email.py`)

This final stage demonstrates the practical use of the trained model in a production-like environment.

- **Goal:** Load the trained model and use it to classify a new, unseen email record instantly.
- **Process:**
    1. A new, simulated email's features are created.
    2. The features are processed through the same engineering steps used during training (consistency is key).
    3. The saved `model.pkl` is loaded from disk.
    4. The model generates a binary prediction (0 or 1) and a confidence probability.