Module 1: Emotional Arabic Chatbot - Complete Technical Documentation

---

Section 1: Setup & Dependencies Installation

Cells 1-4 | Objective: Initialize development environment with required packages and configuration

Cell 1: Package Installation

Purpose: Automate installation of all required Python packages using pip package manager

Key Operations:

- Define install_package() function that executes pip install commands
- Iterate through required_packages list containing 10 essential packages
- Handle installation errors with try-except blocks
- Display success/warning messages for each package installation
- Total packages: numpy, pandas, scikit-learn, nltk, joblib, matplotlib, seaborn, datasets, requests, openpyxl

Output: Console confirmation showing [SUCCESS] or [WARNING] status for each package

---

Cell 2: Import All Required Libraries

Purpose: Load all installed packages and prepare them for use throughout the entire project

Key Operations:

- Import numerical computing (NumPy, Pandas)
- Import utility modules (re, json, pickle, Path)
- Import machine learning framework (scikit-learn modules)
- Import evaluation metrics (classification_report, confusion_matrix, accuracy_score)
- Import natural language processing (NLTK tokenizer and stopwords)
- Import visualization libraries (Matplotlib, Seaborn)
- Import model serialization (joblib)
- Suppress warning messages for cleaner console output

Output: Confirmation message indicating all libraries imported successfully

---

Cell 3: Download Required NLTK Resources

Purpose: Download language resources required for Natural Language Processing tasks

Key Operations:

- Define nltk_resources list containing three essential resources
- Download 'punkt' tokenizer for sentence and word tokenization
- Download 'stopwords' corpus for common word filtering
- Download 'wordnet' for lemmatization and synonym lookup
- Execute downloads with error handling and quiet mode enabled
- Track download status for each resource

Output: Console messages confirming successful download of each NLTK resource

---

Cell 4: Configure Environment Settings and Initialize Directories

Purpose: Establish reproducible environment configuration and prepare file system structure

Key Operations:

- Configure Matplotlib visualization style to 'seaborn-v0_8-darkgrid'
- Set Seaborn color palette to 'husl' for consistent color scheme
- Initialize RANDOM_SEED = 42 for reproducibility across all random operations
- Set NumPy random seed using np.random.seed()
- Create MODELS_DIR path for saving trained models
- Create DATA_DIR path for storing datasets
- Create directories if they do not exist using mkdir(exist_ok=True)
- Display configuration summary with absolute paths

Output: Environment configuration confirmation with directory paths and random seed value

---

Section 2: Data Loading from Multiple Sources

Cells 5-9 | Objective: Download and load three diverse Arabic datasets from external sources

Cell 5: Setup Kaggle API Configuration for Google Colab

Purpose: Configure authentication credentials for accessing Kaggle API within Google Colab environment

Key Operations:

- Create /root/.kaggle directory for Kaggle credentials
- Copy kaggle.json file from Colab content directory to Kaggle configuration directory
- Set file permissions to 600 (read/write for owner only) for security
- Validate Kaggle API configuration

Output: Confirmation message indicating Kaggle API configuration success

---

Cell 6: Download Required Datasets from Kaggle API

Purpose: Retrieve three distinct Arabic sentiment and review datasets from Kaggle platform

Key Operations:

- Define datasets_to_download dictionary containing Kaggle references
- Download ArSAS (Arabic Sentiment Analysis Corpus) - 10,000+ tweets with sentiment labels
- Download AJGT (Arabic Jordanian General Tweets) - Jordanian dialect tweets with sentiment
- Download HARD (Hotels Arabic Reviews Dataset) - Hotel reviews with sentiment annotations
- Execute Kaggle CLI commands with automatic extraction flag
- Handle download failures with exception handling
- Display status for each dataset download operation

Output: Downloaded datasets stored in ./data/{name}/ directories

---

Cell 7: Load ArSAS Dataset

Purpose: Read and parse ArSAS text file containing Arabic sentiment-labeled tweets

Key Operations:

- Search for .txt files in ./data/arsas/ directory using Path.glob()
- Load first text file using pd.read_csv() with tab delimiter
- Extract dataset dimensions and column names
- Validate dataset structure and availability
- Store dataframe in df_arsas variable
- Handle file not found and parsing exceptions

Output: ArSAS dataframe with sample count and column information

---

Cell 8: Load AJGT Dataset from Excel

Purpose: Read AJGT dataset from Excel file format with sentiment annotations

Key Operations:

- Define Excel file path: /content/data/AJGT.xlsx
- Load Excel file using pd.read_excel() with automatic format detection
- Extract dataset dimensions and column structure
- Validate Excel file format and data integrity
- Store dataframe in df_ajgt variable
- Handle file format errors and missing files

Output: AJGT dataframe with sample count and column information

---

Cell 9: Load QADI Dataset from Hugging Face

Purpose: Download and process QADI dataset containing 18 Arabic dialect classifications

Key Operations:

- Import load_dataset function from Hugging Face datasets library
- Load QADI dataset from Hugging Face repository (Abdelrahman-Rezk/Arabic_Dialect_Identification)
- Extract training split from dataset
- Create dialect_mapping dictionary mapping numeric labels (0-17) to dialect names
- Apply dialect mapping to create human-readable dialect column
- Select relevant columns (text, dialect)
- Remove rows with unmapped dialects
- Validate dataset structure and unique dialect count

Output: QADI dataframe containing 18 Arabic dialects (Moroccan, Iraqi, Levantine, Egyptian, Gulf, Saudi, Tunisian, Algerian, Sudanese, Jordanian, Palestinian, Yemeni, Omani, Qatari, Emirati, Kuwaiti, Lebanese, Syrian)

Section 3: Data Preprocessing and Cleaning

Cells 10-11 | Objective: Standardize text format and combine datasets into unified structure

Cell 10: Define Arabic Text Preprocessing Function

Purpose: Create reusable function to clean and normalize Arabic text data

Key Operations:

- Type validation: check if input is string type, return empty string if invalid
- Lowercase conversion: convert all text to lowercase for consistency
- Whitespace normalization: replace multiple spaces with single space using regex
- URL removal: strip URLs and email addresses using pattern matching
- Mention and hashtag removal: remove @ and # symbols while keeping text content
- Arabic text preservation: keep only Arabic Unicode characters (U+0600 to U+06FF)
- Diacritic removal: eliminate Arabic vowel marks (Fatha, Damma, Kasra, etc)
- Repeated character reduction: replace 3+ consecutive identical characters with 2
- Return cleaned normalized text

Test Samples Provided:

- Input: "أناااااا سعيييد جداااااا!! بهذا اليوم الجميل" → Output: "انا سعيد جدا بهذا اليوم الجميل"
- Input: "خخخخخخخ هههههههه ياااه يا نهار" → Output: "خخ هههه ياه يا نهار"
- Input: "@Ahmed_Hamza27 اتمنى صراحة و خصوصا #طال_عمره" → Output: "اتمنى صراحة و خصوصا طال عمره"

Output: Preprocessed text string with all cleaning operations applied

---

Cell 11: Combine All Datasets into Unified Format

Purpose: Merge three separate datasets into single standardized dataframe for model training

Key Operations:

ArSAS Dataset Processing:

- Create emotion_map_arsas dictionary: Positive→joy, Negative→anger, Neutral→neutral, Mixed→neutral
- Iterate through df_arsas rows
- Extract Sentiment_label and check mapping validity
- Create records with fields: text, emotion, dialect (Modern Standard Arabic), source (ArSAS)
- Add records to combined_data list

AJGT Dataset Processing:

- Identify text and label columns dynamically
- Create emotion_map_ajgt dictionary: Positive→joy, Negative→sadness
- Iterate through df_ajgt rows
- Extract sentiment labels and validate against mapping
- Create records with fields: text, emotion, dialect (Jordanian), source (AJGT)
- Add records to combined_data list
- Display count of samples added from AJGT

Final Dataframe Creation:

- Convert combined_data list to pandas DataFrame
- Apply preprocess_arabic_text() function to text column
- Filter rows with text_clean length > 3 characters (remove empty/minimal text)
- Store processed dataframe in emotion_training_data variable

Output: Unified dataframe with total sample count, source distribution, and emotion distribution statistics

---

Section 4: Emotion Detection Model Training

Cells 12-17 | Objective: Build Random Forest classifier for Arabic emotion recognition

Cell 12: Prepare Emotion Training Data

Purpose: Validate and prepare emotion dataset before model training

Key Operations:

- Create copy of emotion_training_data into emotion_data variable
- Display total sample count in emotion dataset
- Count unique emotion classes
- Generate emotion distribution statistics using value_counts()
- Display summary information for verification

Output: Emotion data preparation confirmation with distribution statistics

---

Cell 13: Encode Emotion Labels and Split Data into Train-Test Sets

Purpose: Convert categorical labels to numeric format and partition data for model validation

Key Operations:

- Initialize LabelEncoder for emotion classification
- Fit encoder to emotion_data['emotion'] column
- Transform emotion labels to numeric values (0, 1, 2, ...)
- Display label mapping: numeric value → emotion name
- Prepare feature vector X_emotion from text_clean column
- Prepare target vector y_emotion from emotion_encoded column
- Execute train_test_split with 85% training / 15% testing ratio
- Use stratified sampling (stratify=y_emotion) to maintain emotion distribution
- Set random_state=RANDOM_SEED for reproducibility
- Display split statistics: training sample count, testing sample count

Output: Train-test split confirmation with sample counts and label mapping

---

Cell 14: Convert Text to Numerical Features Using TF-IDF Vectorization

Purpose: Transform text data into numerical feature vectors for machine learning

Key Operations:

- Initialize TfidfVectorizer with optimized hyperparameters
- Set max_features=5000 to limit vocabulary size
- Set min_df=2 to include terms appearing in at least 2 documents
- Set max_df=0.8 to exclude terms in more than 80% of documents
- Set ngram_range=(1,2) to capture both unigrams and bigrams
- Enable sublinear_tf=True for sublinear term frequency scaling
- Set stop_words=None to preserve Arabic stopwords
- Fit vectorizer on X_train_emotion
- Transform both training and testing sets using fitted vectorizer
- Store sparse matrices in X_train_tfidf_emotion and X_test_tfidf_emotion

Output: Vectorization completion confirmation with feature matrix shapes and vocabulary size

---

Cell 15: Train Random Forest Classifier for Emotion Detection

Purpose: Build and train ensemble model for emotion classification

Key Operations:

- Initialize RandomForestClassifier with tuned hyperparameters
- Set n_estimators=200 for 200 decision trees in ensemble
- Set max_depth=20 to limit tree growth
- Set min_samples_split=5 minimum samples required to split node

- Set min_samples_leaf=2 minimum samples required at leaf node
- Set random_state=RANDOM_SEED for reproducibility
- Set n_jobs=-1 to utilize all available CPU cores
- Set verbose=1 to display training progress
- Execute model.fit() on training TF-IDF features and target labels
- Training automatically optimizes decision tree parameters

Output: Model training completion confirmation

Cell 16: Evaluate Emotion Detection Model Performance

Purpose: Calculate comprehensive metrics to assess model quality and reliability

Key Operations:

- Execute emotion_rf_model.predict() on test TF-IDF features
- Calculate accuracy_emotion: percentage of correct predictions
- Calculate precision_emotion: weighted average of precision across emotions
- Calculate recall_emotion: weighted average of recall across emotions
- Calculate f1_emotion: weighted harmonic mean of precision and recall
- Generate confusion matrix for error analysis
- Extract unique labels present in test and prediction sets
- Map numeric labels back to emotion names
- Generate classification_report with precision, recall, F1 per emotion
- Display all metrics in formatted output

Output: Performance metrics table and detailed per-emotion classification report

---

Cell 17: Save Emotion Detection Model and Associated Artifacts

Purpose: Persist trained model and preprocessing components for future use

Key Operations:

- Define file paths for model artifacts in MODELS_DIR
- Serialize emotion_rf_model using pickle.dump()
- Serialize tfidf_vectorizer_emotion using pickle.dump()
- Serialize emotion_encoder using pickle.dump()
- Create emotion_model_summary dictionary containing metadata
- Store model type, sample counts, vocabulary size
- Store performance metrics (accuracy, precision, recall, F1)
- Store emotion class names
- Serialize summary as JSON with UTF-8 encoding
- Display confirmation messages with absolute file paths

Output: Model files saved and summary JSON file with training metadata

---

Section 5: Dialect Recognition Model Training

Cells 18-23 | Objective: Build Linear SVM classifier for Arabic dialect identification

Cell 18: Prepare Dialect Data with Regional Grouping

Purpose: Consolidate 18 individual dialects into 4 geographic regions for better model performance

Key Operations:

- Create region_map dictionary with dialect-to-region mappings
- Egypt_Sudan region: Egyptian, Sudanese
- Levant region: Lebanese, Syrian, Jordanian, Palestinian, Levantine
- Gulf region: Saudi, Kuwaiti, Qatari, Emirati, Omani, Yemeni, Gulf, Iraqi
- North_Africa region: Moroccan, Tunisian, Algerian, Libyan
- Apply region mapping to qadi_df['dialect'] column
- Remove rows with unmapped dialects using dropna()
- Verify regional class distribution
- Store processed dataframe in dialect_training_data variable

Output: Regional dialect distribution statistics

Cell 19: Encode Dialect Labels and Split Data into Train-Test Sets

Purpose: Encode regional labels numerically and partition data for model validation

Key Operations:

- Initialize LabelEncoder for dialect regions
- Fit encoder to dialect_training_data['region'] column
- Transform region labels to numeric values (0, 1, 2, 3)
- Display label mapping: numeric value → region name
- Prepare feature vector X_dialect from text column (raw, not preprocessed)
- Prepare target vector y_region from region-encoded labels
- Execute train_test_split with 90% training / 10% testing ratio
- Use stratified sampling to maintain region distribution
- Set random_state=RANDOM_SEED for reproducibility
- Display split statistics: training sample count, testing sample count

Output: Train-test split confirmation with sample counts and label mapping

---

Cell 20: Convert Text to Numerical Features for Dialect Model

Purpose: Transform dialect text into numerical feature vectors using TF-IDF

Key Operations:

- Initialize TfidfVectorizer with larger vocabulary for dialect classification
- Set max_features=20000 to capture extensive vocabulary
- Set ngram_range=(1,2) for unigrams and bigrams
- Set min_df=5 to include terms in at least 5 documents

- Enable sublinear_tf=True for improved term frequency scaling
- Fit vectorizer on X_train_d (training dialect text)
- Transform both training and testing sets
- Store sparse matrices in X_train_tfidf_d and X_test_tfidf_d

Output: Vectorization completion with feature matrix shapes and vocabulary size

---

Cell 21: Train Balanced Linear SVM Classifier for Dialect Recognition

Purpose: Build Support Vector Machine model with class balancing for uneven region distribution

Key Operations:

- Initialize SGDClassifier (Stochastic Gradient Descent) implementing linear SVM
- Set loss='hinge' for linear SVM loss function
- Set penalty='l2' for L2 regularization
- Set alpha=1e-4 (learning rate for regularization strength)
- Set class_weight='balanced' to automatically adjust weights for class imbalance
- Set random_state=RANDOM_SEED for reproducibility
- Set n_jobs=-1 to utilize all CPU cores
- Set max_iter=50 for maximum training iterations
- Execute model.fit() on training TF-IDF features and regional labels
- Class balancing ensures minority regions receive equal importance

Output: Model training completion confirmation

Cell 22: Evaluate Dialect Detection Model Performance

Purpose: Calculate accuracy metrics and generate per-region performance report

Key Operations:

- Execute dialect_model.predict() on test TF-IDF features
- Calculate accuracy_dialect: overall percentage of correct predictions
- Generate classification_report with precision, recall, F1 per region
- Display metrics in formatted output for each geographic region
- Handle zero_division parameter to avoid warnings

Output: Overall accuracy metric and detailed per-region classification report

---

Cell 23: Save Dialect Recognition Model and Associated Artifacts

Purpose: Serialize trained dialect model and preprocessing components for production use

Key Operations:

- Define file paths for dialect model artifacts in MODELS_DIR
- Serialize dialect_model using joblib.dump()
- Serialize dialect_vectorizer using joblib.dump()
- Serialize region_encoder using joblib.dump()
- Create dialect_model_summary dictionary containing metadata
- Store model type, sample counts, vocabulary size, accuracy, regions

- Serialize summary as JSON with UTF-8 encoding
- Display confirmation messages with absolute file paths

Output: Dialect model files saved and summary JSON file with training metadata

---

Section 6: Integration & Testing Pipeline

Cells 24-25 | Objective: Load models and create unified end-to-end prediction pipeline

Cell 24: Load Trained Models and Create Unified Prediction Function

Purpose: Retrieve saved models from disk and define function for integrated emotion-dialect inference

Key Operations:

Model Loading:

- Load emotion_detection_model.pkl containing trained Random Forest classifier
- Load emotion_tfidf_vectorizer.pkl for emotion text vectorization
- Load emotion_label_encoder.pkl for emotion numeric-to-label conversion
- Load dialect_model.pkl containing trained Linear SVM classifier
- Load dialect_vectorizer.pkl for dialect text vectorization
- Load dialect_encoder.pkl for dialect numeric-to-label conversion

Prediction Function Definition:

- Function name: predict_emotion_and_dialect(text)
- Input parameter: Arabic text string
- Process input text through preprocess_arabic_text() function
- Vectorize cleaned text using emotion_vectorizer.transform()
- Execute emotion_model.predict() to get numeric emotion prediction
- Extract emotion_model.predict_proba() for confidence scores
- Convert numeric prediction to emotion label using inverse_transform()
- Calculate emotion_confidence as maximum probability value
- Vectorize cleaned text using dialect_vectorizer.transform()
- Execute dialect_model.predict() to get numeric dialect prediction
- Convert numeric prediction to region label using inverse_transform()
- Calculate dialect_confidence from decision_function() output
- Return dictionary with: input_text, cleaned_text, emotion, emotion_confidence, dialect, dialect_confidence

Output: Confirmation message indicating successful model loading and function creation

---

Cell 25: Test Unified Emotion and Dialect Prediction Pipeline

Purpose: Validate integrated pipeline functionality on diverse test samples

Key Operations:

- Define test_samples list containing 5 Arabic text samples

- Sample 1: "أنا سعيد جداً بهذا النبأ الرائع" (Modern Standard Arabic - positive)
- Sample 2: "والله ما في احسن من هيك يا صاحبي" (Levantine dialect - positive)
- Sample 3: "انا تعبان من كل شي والحياة صعبة" (Modern Standard Arabic - negative)
- Sample 4: "غاضب جداً من هذا الموضوع السيء" (Modern Standard Arabic - angry)
- Sample 5: "يا رب تنجح في الامتحان بتاعك يا حبيبي" (Levantine dialect - hopeful)
- Iterate through each test sample
- Execute predict_emotion_and_dialect() function on each sample
- Extract and display predictions: input text, emotion label, emotion confidence
- Extract and display predictions: dialect region, dialect confidence
- Format output for readability with separator lines
- Display pipeline testing completion confirmation

Output: Predictions table showing emotion classification, dialect identification, and confidence scores for each test sample

---

Model Architecture Summary

Emotion Detection Model

- **Model Type**: Random Forest Classifier (200 trees)
- **Feature Extraction**: TF-IDF Vectorizer (5000 features, 1-2 grams)
- **Input**: Preprocessed Arabic text
- **Output**: Emotion class (joy, anger, sadness, neutral) + confidence score
- **Training Data**: ~5000-10000 Arabic tweets from ArSAS and AJGT
- **Expected Accuracy**: 75-85%

Dialect Recognition Model

- **Model Type**: Linear SVM with SGDClassifier
- **Feature Extraction**: TF-IDF Vectorizer (20000 features, 1-2 grams)
- **Input**: Raw Arabic text (not preprocessed)
- **Output**: Dialect region (Egypt_Sudan, Levant, Gulf, North_Africa) + confidence score
- **Training Data**: 10000+ Arabic sentences from QADI dataset
- **Expected Accuracy**: 80-90%

Integration Pipeline

- **Input**: Any Arabic text string
- **Preprocessing**: Text normalization and cleaning
- **Parallel Processing**: Simultaneous emotion and dialect inference
- **Output**: Unified dictionary with emotion, dialect, and confidence metrics
- **Processing Time**: <1 second per sample on CPU

Project Completion Checklist

✅ Section 1: Environment setup with all dependencies installed
✅ Section 2: Three datasets loaded from Kaggle and Hugging Face
✅ Section 3: Arabic text preprocessing function defined and tested
✅ Section 4: Emotion detection model trained and evaluated

✅ Section 5: Dialect recognition model trained and evaluated
✅ Section 6: Unified prediction pipeline created and tested

Total Cells: 25
Total Lines of Code: 1000+
Total Documentation: Comprehensive inline comments
Model Artifacts: 6 saved files (models, vectorizers, encoders)