Technical Report

AlHaram Analytics

Preprocessing Pipeline for Saudi Arabian
Mobile Application Reviews

Version 0.2.0

Naila Marir
Computer Science Department
Effat University, Jeddah, Saudi Arabia
nmarir@effatuniversity.edu.sa

January 2026

Abstract

AlHaram Analytics is a specialized data preprocessing and analytics system designed to process, clean, and enrich mobile application review data from Saudi Arabian government and religious service applications. The system addresses unique challenges in Arabic text processing, including Arabizi conversion, Islamic calendar event tagging, and multi-language support. The pipeline processes reviews from applications related to Hajj, Umrah, healthcare, transportation, and government services, providing structured data suitable for sentiment analysis and user experience research. This technical report provides comprehensive documentation of the system architecture, components, APIs, and deployment guidelines.

Contents

1 Introduction

1.1 Background

The Kingdom of Saudi Arabia has developed numerous mobile applications to support pilgrims, residents, and visitors. These applications span healthcare (Sehhaty, Asaafni), transportation (Makkah Buses, Haramain Train), government services (Tawakkalna, Nusuk), and religious guidance (Quran apps, Qibla finders). User reviews from these applications provide valuable insights into user experience, service quality, and areas for improvement.

1.2 Problem Statement

Processing Arabic app reviews presents unique challenges:

- **Arabizi usage**: Users frequently write Arabic words using Latin characters and numerals (e.g., “7abibi” for “habibi”)
- **Multi-language content**: Reviews contain Arabic, English, or mixed-language text
- **Islamic calendar relevance**: User behavior varies significantly during Hajj, Ramadan, and Eid periods
- **Username diversity**: Usernames contain Arabic, Latin, emojis, and special characters

1.3 Objectives

1. Standardize and clean user-generated content
2. Detect and classify review languages
3. Tag reviews with relevant Islamic calendar periods
4. Classify applications by service type
5. Predict user gender from usernames (optional)
6. Provide interactive visualization and processing interface

1.4 Target Applications

Table 1: Target Applications by Category

Category	**Applications**
Health Services	Sehhaty, Asaafni
Transportation	Makkah Buses, HHR Train, Tanqul
Government	Tawakkalna, Nusuk, Irshad
Religious	Qibla Finder, Haramain Quran

2 System Architecture

2.1 High-Level Architecture

The AlHaram Analytics system follows a modular architecture with three primary interfaces: Web UI, Command-Line Interface (CLI), and Python API. All interfaces utilize a shared preprocessing pipeline that orchestrates multiple transformation components.

```
+------------------------------------------------------------------+
|                    AlHaram Analytics System                       |
+------------------------------------------------------------------+
|                                                                   |
|  +----------------+  +----------------+  +----------------+       |
|  | Web Interface  |  | CLI Interface  |  |  Python API    |       |
|  |    (Flask)     |  |   (argparse)   |  |   (Module)     |       |
|  +-------+--------+  +-------+--------+  +-------+--------+       |
|          |                   |                   |                |
|          +-------------------+-------------------+                |
|                              |                                    |
|                   +----------v----------+                         |
|                   | PreprocessingPipeline|                        |
|                   +----------+----------+                         |
|                              |                                    |
|  +---------------------------+---------------------------+        |
|  |                           |                           |        |
|  v                           v                           v        |
| +-------------+  +-------------------+  +------------------+      |
| |Preprocessing|  |Feature Engineering|  |Gender Prediction |      |
| +-------------+  +-------------------+  +------------------+      |
| | - Username  |  | - App Normalizer  |  | - HF Classifier  |      |
| | - Language  |  | - Device Mapper   |  | - Ensemble       |      |
| +-------------+  | - Period Tagger   |  +------------------+      |
|                  | - Service Class.  |                            |
|                  +-------------------+                            |
+------------------------------------------------------------------+
```

Figure 1: AlHaram Analytics System Architecture

2.2 Directory Structure

```
AlharamApplication/
|-- src/alharam_analytics/          # Core library
|   |-- __init__.py
|   |-- pipeline.py                 # Main orchestrator
|   |-- preprocessing/              # Text preprocessing
|   |   |-- username_cleaner.py
|   |   |-- language_detector.py
|   |   |-- text_cleaner.py         # NEW: Arabic text normalization
|   |-- feature_engineering/        # Feature extraction
|   |   |-- app_name_normalizer.py
|   |   |-- device_mapper.py
|   |   |-- period_tagger.py
|   |   |-- service_classifier.py
|   |   |-- text_feature_extractor.py  # NEW: Text metrics
|   |-- sentiment/                  # NEW: Sentiment analysis
|   |   |-- sentiment_analyzer.py   # Deep learning + lexicon
|   |-- gender_prediction/          # ML-based prediction
|   |   |-- hf_gender_classifier.py
|   |   |-- ensemble_predictor.py
|   |-- utils/
|       |-- io_utils.py
|-- webapp/                         # Flask web application
|   |-- app.py
|   |-- templates/
|   |-- static/
|-- config/
|   |-- pipeline_config.yaml
|-- scripts/
|   |-- run_preprocessing.py
|-- docs/
|-- notebooks/
|-- data/
    |-- raw/
    |-- processed/
```

Listing 1: Project Directory Structure

2.3 Design Principles

1. **Modularity**: Each preprocessing step is an independent, reusable component
2. **Scikit-learn Compatibility**: Transformers implement fit/transform interface
3. **Configurability**: YAML-based configuration for all parameters
4. **Extensibility**: Easy to add new preprocessing steps or classifiers
5. **Multi-interface**: Web UI, CLI, and Python API for different use cases

3 Technology Stack

3.1 Core Technologies

3.2 Frontend Technologies

- **HTML5**: Semantic markup
- **CSS3**: Responsive styling with CSS Grid and Flexbox

Table 2: Core Technology Stack

Component	Technology	Version	Purpose
Language	Python	3.10+	Primary development language
Data Processing	pandas	≥2.0.0	DataFrame operations
Excel Support	openpyxl	≥3.1.0	Excel file I/O
Language Detection	langid	≥1.1.6	Text language classification
Islamic Calendar	hijri-converter	≥2.3.0	Gregorian-Hijri conversion
Web Framework	Flask	≥3.0.0	REST API and web interface
ML Models	transformers	≥4.30.0	HuggingFace model inference
Deep Learning	PyTorch	≥2.0.0	Model backend

- **JavaScript (ES6+)**: Async/await, fetch API
- **Font Awesome 6.5.1**: Icon library
- **Google Fonts**: Inter (Latin), Cairo (Arabic)

3.3 Development Tools

Table 3: Development Tools

Tool	Purpose
pytest	Unit testing
black	Code formatting (line length: 88)
ruff	Linting (E, F, I, N, W rules)
setuptools	Package building

4 Core Components

4.1 Username Cleaner

Module: `src/alharam_analytics/preprocessing/username_cleaner.py`

Handles diverse username formats common in Arabic-speaking user bases.

4.1.1 Arabizi Conversion Map

Table 4: Arabizi Character Mapping

Character	Arabic Sound	Name	Example
2	' (glottal stop)	Hamza/Alif	a2mad → ahmad
3	‘ayn	Ain	3ali → ali
5	kh	Kha	5aled → khaled
6	emphatic t	Ta	6ariq → tariq
7	h (pharyngeal)	Ha	a7mad → ahmad
8	gh	Ghain	8areeb → ghareeb
9	emphatic s	Sad	9aber → saber

4.1.2 Processing Steps

1. Convert Arabizi numerals to Latin equivalents
2. Remove trailing digits (e.g., "Hassan855" → "Hassan")
3. Strip punctuation, symbols, and emojis
4. Filter names with fewer than 3 letters → "Anonymous"
5. Handle null/None values gracefully

4.1.3 Usage Example

```python
from alharam_analytics.preprocessing import UsernamePreprocessor

cleaner = UsernamePreprocessor()
df = cleaner.transform(df)  # Adds "clean_name" column
```

Listing 2: Username Cleaner Usage

4.2 Text Cleaner

Module: `src/alharam_analytics/preprocessing/text_cleaner.py`

Comprehensive Arabic text cleaning that extracts information to separate columns rather than removing it.

4.2.1 Design Philosophy

Unlike traditional text cleaners that remove URLs, emojis, and special characters, the AlHaram Text Cleaner preserves all information by extracting elements to dedicated columns. This approach:

- Preserves emojis for sentiment analysis
- Keeps URLs for reference tracking
- Maintains hashtags and mentions for social analysis
- Flags presence of diacritics and elongation patterns

4.2.2 Extraction Features

Table 5: Text Cleaner Extraction Features

Feature	**Output Column**	**Description**
URLs	extracted_urls	List of URLs found in text
Emails	extracted_emails	List of email addresses
Emojis	extracted_emojis	List of emojis with count
Hashtags	extracted_hashtags	Arabic and Latin hashtags
Mentions	extracted_mentions	@username mentions

Table 6: Arabic Character Normalization Mapping

Original	**Normalized**	**Description**
Alef with Madda	Alef	Alef variants unified
Alef with Hamza Above	Alef	Alef variants unified
Alef with Hamza Below	Alef	Alef variants unified
Alef Wasla	Alef	Alef variants unified
Alef Maksura	Ya	Common normalization

4.2.3 Arabic Normalization

4.2.4 Additional Processing

1. **Diacritics (Tashkeel)**: Stripped from clean text but flagged via `has_diacritics` column
2. **Character Elongation**: Repeated characters (3+) collapsed to 2 (e.g., "shukraaaan" → "shukraan"), flagged via `has_elongation`
3. **Zero-width Characters**: Removed (U+200B-U+200D, FEFF, soft hyphen)
4. **Whitespace**: Normalized to single spaces

4.2.5 Usage Example

```python
from alharam_analytics.preprocessing import TextCleaner

cleaner = TextCleaner(
    text_column="Review Text",
    normalize_arabic=True,
    extract_emojis=True,
    strip_diacritics=True
)
df = cleaner.transform(df)

# Output columns added:
# - clean_text: Normalized text
# - extracted_urls, url_count, has_urls
# - extracted_emojis, emoji_count, has_emojis
# - extracted_hashtags, hashtag_count, has_hashtags
# - has_diacritics, has_elongation
```

Listing 3: Text Cleaner Usage

4.3 Language Detector

Module: `src/alharam_analytics/preprocessing/language_detector.py`
Multi-strategy language detection for Arabic/English content.

4.3.1 Detection Algorithm

1. Use langid library for primary classification
2. Check for Arabic Unicode range (U+0600-U+06FF)
3. Check for Latin characters (A-Za-z)
4. Classification:
   - Primarily Arabic characters → "Arabic"

- Primarily Latin characters → “English”
- Both present significantly → “Mixed”
- Neither detected → “Unknown”

4.3.2 Output Categories

Table 7: Language Detection Output Categories

Category	Description
Arabic	Predominantly Arabic script
English	Predominantly Latin script
Mixed	Significant presence of both scripts
Unknown	Unable to determine (emojis only, symbols, etc.)

4.4 Period Tagger

Module: `src/alharam_analytics/feature_engineering/period_tagger.py`

Tags reviews based on Islamic calendar events and Saudi academic calendar.

4.4.1 Islamic Period Definitions

Table 8: Islamic Calendar Period Definitions

Period	Hijri Date	Description
Hajj Season	1-15 Dhul Hijjah	Peak pilgrimage period
Eid al-Adha	10-13 Dhul Hijjah	Festival of Sacrifice
Eid al-Fitr	1-3 Shawwal	End of Ramadan
Ramadan	Full month 9	Fasting month
School Summer	Ministry dates	Saudi academic break
Regular	Other dates	Normal period

4.4.2 Implementation

```python
from hijri_converter import Hijri, Gregorian

def tag_period(gregorian_date):
    hijri = Gregorian(date.year, date.month, date.day).to_hijri()

    if hijri.month == 12 and 1 <= hijri.day <= 15:
        if 10 <= hijri.day <= 13:
            return "Eid al-Adha"
        return "Hajj Season"
    elif hijri.month == 10 and 1 <= hijri.day <= 3:
        return "Eid al-Fitr"
    elif hijri.month == 9:
        return "Ramadan"
    # ... check school summer dates
    return "Regular"
```

Listing 4: Period Tagging Implementation

4.5 Service Classifier

Module: `src/alharam_analytics/feature_engineering/service_classifier.py`

Categorizes applications into service types based on predefined mappings.

```python
SERVICE_MAPPING = {
    "Health Services": [
        "sehhaty", "asaafni"
    ],
    "Reservation/Transport": [
        "makkah buses", "hhr train", "tanqul", "trwayyah"
    ],
    "Government Services": [
        "tawakkalna", "nusuk", "irshad"
    ],
    "Religious": [
        "qibla finder", "haramain quran"
    ],
    "Others": []  # Default category
}
```

Listing 5: Service Classification Mapping

4.6 Text Feature Extractor

Module: `src/alharam_analytics/feature_engineering/text_feature_extractor.py`

Extracts quantitative text features for analysis and machine learning models.

4.6.1 Feature Categories

Table 9: Text Feature Extractor Output

Feature	Column	Description
Length Metrics		
Character count	text_char_count	Total characters
Word count	text_word_count	Total words
Sentence count	text_sentence_count	Approximate sentences
Avg word length	text_avg_word_length	Mean word length
Script Analysis		
Arabic chars	text_arabic_char_count	Arabic character count
Latin chars	text_latin_char_count	Latin character count
Arabic ratio	text_arabic_ratio	Arabic / total alphabetic
Digit count	text_digit_count	Number of digits
Digit ratio	text_digit_ratio	Digits / total chars
Lexical Features		
Unique words	text_unique_word_count	Vocabulary size
Lexical diversity	text_lexical_diversity	Unique / total words (TTR)
Punctuation		
Punctuation count	text_punctuation_count	Total punctuation marks
Exclamation count	text_exclamation_count	Count of !
Question count	text_question_count	Count of ? and Arabic ?

4.6.2 Usage Example

```
from alharam_analytics.feature_engineering import TextFeatureExtractor

extractor = TextFeatureExtractor(
    text_column="Review Text",
    prefix="text_",
    include_ratios=True,
    include_lexical=True
)
df = extractor.transform(df)

# Use features for analysis
avg_arabic = df[’text_arabic_ratio’].mean()
print(f"Average Arabic content: {avg_arabic:.1%}")
```

Listing 6: Text Feature Extractor Usage

4.7 Sentiment Analyzer

Module: `src/alharam_analytics/sentiment/sentiment_analyzer.py`

Deep learning-based sentiment analysis for Arabic app reviews using pre-trained transformer models.

4.7.1 Model Options

Table 10: Available Sentiment Models

Model	**HuggingFace Path**	**Best For**
CAMeL-BERT	CAMeL-Lab/bert-base-arabic-camelbert-mix-sentiment	MSA + Dialectal
AraBERT	aubmindlab/bert-base-arabertv2	Modern Standard Arabic
Multilingual	nlptown/bert-base-multilingual-uncased-sentiment	Mixed content

4.7.2 Output Schema

Table 11: Sentiment Analysis Output Columns

Column	**Type**	**Description**
sentiment	string	Label: positive, neutral, negative
sentiment_score	float	Score from -1 (negative) to +1 (positive)
sentiment_confidence	float	Model confidence from 0 to 1

4.7.3 Architecture

The sentiment analyzer uses a two-tier approach:

1. **Deep Learning (Primary)**: CAMeL-BERT transformer model fine-tuned for Arabic sentiment classification. Processes text in batches with GPU acceleration when available.

2. **Lexicon-Based (Fallback)**: Rule-based analyzer using Arabic and English sentiment lexicons plus emoji sentiment mapping. Used when transformers library is unavailable or GPU memory is insufficient.

4.7.4 Lexicon-Based Fallback

The `SimpleSentimentAnalyzer` uses curated word lists:

- **Positive words**: Arabic (mumtaz, rai', jamil, shukran) and English (excellent, great, love)
- **Negative words**: Arabic (sayyi', fashil, mushkila) and English (bad, terrible, hate)
- **Emoji sentiment**: Positive (thumbs up, heart, smile) and Negative (thumbs down, angry, crying)

4.7.5 Usage Example

```python
from alharam_analytics.sentiment import SentimentAnalyzer

# Deep learning approach
analyzer = SentimentAnalyzer(
    model_name='camel-bert',
    batch_size=16,
    device='cuda'  # or 'cpu'
)
df = analyzer.transform(df)

# Check sentiment distribution
print(df['sentiment'].value_counts(normalize=True))
# positive    0.45
# neutral     0.35
# negative    0.20

# Filter negative reviews for analysis
negative_reviews = df[df['sentiment'] == 'negative']
```

Listing 7: Sentiment Analyzer Usage

4.7.6 Performance Considerations

Table 12: Sentiment Analysis Performance

Method	Speed (1000 rows)	Accuracy
CAMeL-BERT (GPU)	∼30 seconds	High (fine-tuned)
CAMeL-BERT (CPU)	∼5 minutes	High (fine-tuned)
Lexicon-based	∼2 seconds	Moderate

4.8 Gender Prediction (Optional)

Module: `src/alharam_analytics/gender_prediction/`

Ensemble-based gender prediction using HuggingFace transformers.

4.8.1 Models Used

1. **imranali291/genderize**: General name-based classifier
2. **padmajabfrl/Gender-Classification**: Alternative classifier

4.8.2 Ensemble Decision Logic

```python
def predict_ensemble(name):
    pred1, conf1 = model1.predict(name)
    pred2, conf2 = model2.predict(name)

    if pred1 == pred2:
        return pred1  # Agreement
    elif conf1 >= 0.80:
        return pred1  # High confidence model 1
    elif conf2 >= 0.80:
        return pred2  # High confidence model 2
    else:
        return "unknown"  # Disagreement, low confidence
```

Listing 8: Gender Prediction Ensemble Logic

4.8.3 Output Columns

Table 13: Gender Prediction Output Columns

Column	Description
pred_gender_1	Model 1 prediction
pred_score_1	Model 1 confidence (0-1)
pred_gender_2	Model 2 prediction
pred_score_2	Model 2 confidence (0-1)
gender_final	Ensemble decision

5 Data Processing Pipeline

5.1 Pipeline Overview

The `PreprocessingPipeline` class orchestrates all preprocessing steps in a configurable sequence.

```python
from alharam_analytics.pipeline import PreprocessingPipeline

# Initialize
pipeline = PreprocessingPipeline(
    include_gender_prediction=False,
    verbose=True
)

# Run full pipeline
df = pipeline.run("data/raw/reviews.xlsx")

# Run specific steps
df = pipeline.run(
    "data/raw/reviews.xlsx",
    steps=["username", "language", "period"]
)

# Save results
pipeline.save(df, "data/processed/reviews_cleaned.xlsx")
```

Listing 9: Pipeline Usage Example

5.2 Pipeline Steps

Table 14: Pipeline Processing Steps

Step	Component	Input Column	Output Co
1	TextCleaner	Review Text	clean_text,
2	UsernamePreprocessor	User Name	clean_name
3	LanguageDetector	Review Text	language
4	DeviceTypeMapper	Platform	Device Type
5	AppNameNormalizer	Application Name	Application
6	ServiceClassifier	Application Name	Service_Typ
7	TextFeatureExtractor	Review Text	text_* (14 f
8	PeriodTagger	Review Date	period, App_
9	SentimentAnalyzer	Review Text	sentiment, se
10*	GenderEnsemblePredictor	clean_name	gender_final

*Optional step

5.3 Pipeline Execution Flow

6 Web Application

6.1 Overview

The Flask-based web application provides an interactive interface for data preprocessing with real-time preview and step-by-step execution.

- **Entry Point:** `run_webapp.py`
- **Default URL:** `http://localhost:5000`

6.2 User Interface Components

6.2.1 File Upload Section

- Drag-and-drop or click to browse
- Supports .xlsx, .xls, .csv formats
- Maximum file size: 50MB
- Upload progress indication

6.2.2 Pipeline Visualization

Six processing cards with:

- Step icon and name
- Brief description
- Preview button (shows before/after)
- Apply button (executes step)
- Status indicator (pending/applied)

```
Input Data (Excel/CSV)
         |
         v
+------------------------+
| 1. Text Cleaning       |
|    - Arabic normaliz.  |
|    - Extract URLs/emoji|
+----------+-------------+
           |
           v
+------------------------+
| 2. Username Cleaning   |
|    - Arabizi conv.     |
|    - Symbol removal    |
+----------+-------------+
           |
           v
+------------------------+
| 3. Language Detection  |
|    - langid            |
|    - Script analysis   |
+----------+-------------+
           |
           v
+------------------------+
| 4. Device Mapping      |
|    - Platform -> Type  |
+----------+-------------+
           |
           v
+------------------------+
| 5. App Normalization   |
|    - Spelling fixes    |
|    - Name unification  |
+----------+-------------+
           |
           v
+------------------------+
| 6. Service Classify    |
|    - Category assign   |
+----------+-------------+
           |
           v
+------------------------+
| 7. Text Features       |
|    - Word count, ratio |
|    - Lexical diversity |
+----------+-------------+
           |
           v
+------------------------+
| 8. Period Tagging      |
|    - Hijri convert     |
|    - Event matching    |
+----------+-------------+
           |
           v
```

6.2.3 Action Buttons

- **Apply All**: Execute full pipeline
- **Reset**: Revert to original data
- **Download**: Export processed file
- **Change File**: Upload different file

6.3 Session Management

```python
# In-memory session storage
sessions = {
    "20260103143052": {
        "original_df": DataFrame,    # Original uploaded data
        "current_df": DataFrame,     # Current state
        "applied_steps": ["username", "language"],
        "filename": "reviews.xlsx"
    }
}
```

Listing 10: Session Storage Structure

Note: Production deployments should use Redis or database-backed sessions.

7 API Reference

7.1 REST Endpoints

Table 15: REST API Endpoints

Endpoint	Method	Description
/	GET	Main web interface
/upload	POST	Upload data file
/preview/<session_id>/<step_id>	GET	Preview transformation
/apply/<session_id>/<step_id>	POST	Apply single step
/apply-all/<session_id>	POST	Apply all steps
/reset/<session_id>	POST	Reset to original
/download/<session_id>	GET	Download processed file
/stats/<session_id>	GET	Get dataset statistics

7.2 Response Formats

7.2.1 Upload Response

```json
{
  "success": true,
  "session_id": "20260103143052",
  "filename": "reviews.xlsx",
  "rows": 15420,
  "columns": ["Review ID", "Review Date", "Review Text", ...],
  "sample": [...]
}
```

Listing 11: Upload Response Format

7.2.2 Preview Response

```json
{
  "success": true,
  "step": "username",
  "input_column": "User Name",
  "output_column": "clean_name",
  "sample_before": ["Mo7amed123", "fatima_2020", ...],
  "sample_after": ["Mohamed", "fatima", ...],
  "stats": {
    "unique_before": 8542,
    "unique_after": 7831,
    "anonymous_count": 245
  }
}
```

Listing 12: Preview Response Format

7.3 CLI Interface

```bash
# Basic usage
python scripts/run_preprocessing.py \
    -i data/raw/reviews.xlsx \
    -o data/processed/reviews_cleaned.xlsx

# With gender prediction
python scripts/run_preprocessing.py \
    -i data/raw/reviews.xlsx \
    -o data/processed/reviews_cleaned.xlsx \
    --gender

# Quiet mode (no progress output)
python scripts/run_preprocessing.py \
    -i data/raw/reviews.xlsx \
    -o data/processed/reviews_cleaned.xlsx \
    -q

# Specific steps only
python scripts/run_preprocessing.py \
    -i data/raw/reviews.xlsx \
    -o data/processed/reviews_cleaned.xlsx \
    --steps username language period
```

Listing 13: Command-Line Interface Usage

8 Data Schema

8.1 Input Requirements

8.2 Output Schema

After full pipeline execution:

9 Installation and Deployment

9.1 Requirements

- Python 3.10 or higher

Table 16: Input Data Requirements

Column	**Type**	**Required**	**Description**
Review ID	integer	Yes	Unique identifier
Review Date	datetime	Yes	Review submission date
Review Text	string	Yes	Review content
Rating	integer	No	1-5 star rating
User Name	string	Yes	Reviewer username
Platform	string	Yes	App store (App Store/Google Play)
Application Name	string	Yes	Application name

Table 17: Output Data Schema

Column	**Type**	**Source**	**Description**
Review ID	int	Original	Unique identifier
Review Date	datetime	Original	Submission date
Review Text	string	Original	Review content
Rating	int	Original	Star rating
User Name	string	Original	Original username
clean_text	string	Step 1	Normalized clean text
extracted_emojis	list	Step 1	Emojis found in text
has_urls	bool	Step 1	Contains URLs
clean_name	string	Step 2	Cleaned username
Platform	string	Original	App store
Device Type	string	Step 4	iOS/Android/Other
Application Name	string	Step 5	Normalized app name
Service_Type	string	Step 6	Service category
language	string	Step 3	Arabic/English/Mixed/Unknown
text_word_count	int	Step 7	Word count
text_arabic_ratio	float	Step 7	Arabic character ratio
period	string	Step 8	Islamic calendar period
App_Version_Period	string	Step 8	Quarter (2023Q1, etc.)
sentiment	string	Step 9	positive/neutral/negative
sentiment_score	float	Step 9	Score (-1 to +1)
gender_final*	string	Step 10	Male/Female/unknown

*Only if gender prediction enabled

- pip package manager
- 4GB+ RAM (8GB+ recommended for gender prediction)

9.2 Installation Steps

```bash
# Clone repository
git clone https://github.com/username/AlharamApplication.git
cd AlharamApplication

# Create virtual environment
python -m venv venv
source venv/bin/activate  # Linux/Mac
# venv\Scripts\activate   # Windows

# Install dependencies
pip install -r requirements.txt

# Optional: Install gender prediction dependencies
pip install transformers torch

# Install package in development mode
pip install -e .
```

Listing 14: Installation Commands

9.3 Running the Application

9.3.1 Web Interface

```bash
python run_webapp.py
# Opens browser at http://localhost:5000
```

9.3.2 Command Line

```bash
python scripts/run_preprocessing.py -i input.xlsx -o output.xlsx
```

9.3.3 Python API

```python
from alharam_analytics.pipeline import PreprocessingPipeline

pipeline = PreprocessingPipeline()
df = pipeline.run("input.xlsx")
pipeline.save(df, "output.xlsx")
```

9.4 Configuration

Edit `config/pipeline_config.yaml`:

```yaml
data:
  raw_dir: "data/raw"
  processed_dir: "data/processed"

preprocessing:
  username:
    min_letters: 3
    column_name: "User Name"
    output_column: "clean_name"

gender_prediction:
  enabled: false
  confidence_threshold: 0.60
  high_confidence_threshold: 0.80

logging:
  verbose: true
```

Listing 15: Configuration File Example

10 Performance Considerations

10.1 Processing Times

Table 18: Processing Time Estimates

Step	Time per 1000 rows	Notes
Text Cleaning	∼1-2 seconds	Regex extraction
Username Cleaning	∼0.5 seconds	String operations
Language Detection	∼2-3 seconds	langid inference
Device Mapping	∼0.1 seconds	Dictionary lookup
App Normalization	∼0.2 seconds	String matching
Service Classification	∼0.2 seconds	Dictionary lookup
Text Feature Extraction	∼1 second	Character analysis
Period Tagging	∼1-2 seconds	Hijri conversion
Sentiment (GPU)	∼30 seconds	CAMeL-BERT inference
Sentiment (CPU)	∼5 minutes	CAMeL-BERT inference
Sentiment (Lexicon)	∼2 seconds	Word matching
Gender Prediction	∼30-60 seconds	Transformer inference

10.2 Memory Usage

Table 19: Memory Usage Estimates

Dataset Size	Without Gender	With Gender
10,000 rows	∼200 MB	∼2 GB
50,000 rows	∼500 MB	∼3 GB
100,000 rows	∼1 GB	∼4 GB

10.3 Optimization Recommendations

1. **Batch Processing**: Process large files in chunks

2. **Disable Gender Prediction**: Skip if not needed (saves 80% time)

3. **Selective Steps**: Run only required preprocessing steps

4. **Caching**: Cache HuggingFace models locally

11 Future Enhancements

11.1 Implemented Features (v0.2.0)

1. **Text Cleaning**: Extraction-based Arabic text normalization preserving all information
2. **Text Feature Extraction**: Quantitative metrics including Arabic ratio, lexical diversity
3. **Sentiment Analysis**: Deep learning (CAMeL-BERT) with lexicon fallback

11.2 Planned Features

1. **Topic Modeling**: Automatic topic extraction from reviews using LDA or BERTopic
2. **Database Backend**: PostgreSQL for persistent storage
3. **User Authentication**: Multi-user support with sessions
4. **Batch Processing API**: Async processing for large datasets
5. **Export Formats**: Add PDF report generation
6. **Aspect-Based Sentiment**: Fine-grained sentiment on specific aspects (UI, performance, etc.)

11.3 Technical Improvements

1. **Redis Sessions**: Replace in-memory session storage
2. **Celery Workers**: Background task processing
3. **Docker Deployment**: Containerized deployment
4. **API Rate Limiting**: Production-ready API protection
5. **Logging**: Structured logging with ELK stack integration

A Arabizi Character Mapping

Table 20: Complete Arabizi Character Mapping

Number	Arabic Sound	Name	Example
2	’ (glottal stop)	Hamza/Alif	sa2al → sa’al
3	‘ayn	Ain	3arab → arab
5	kh	Kha	5air → khair
6	emphatic t	Ta	6areq → tareq
7	h (pharyngeal)	Ha	7ubb → hubb
8	gh	Ghain	8areeb → ghareeb
9	emphatic s	Sad	9abr → sabr

Table 21: Hijri Calendar Months and Notable Events

Month #	Name	Notable Events
1	Muharram	Islamic New Year
2	Safar	—
3	Rabi' al-Awwal	Mawlid (Prophet's Birthday)
4	Rabi' al-Thani	—
5	Jumada al-Awwal	—
6	Jumada al-Thani	—
7	Rajab	Isra and Mi'raj
8	Sha'ban	—
9	Ramadan	Fasting Month
10	Shawwal	Eid al-Fitr
11	Dhul Qa'dah	—
12	Dhul Hijjah	Hajj, Eid al-Adha

B Islamic Calendar Reference

References

1. Flask Documentation: https://flask.palletsprojects.com/
2. HuggingFace Transformers: https://huggingface.co/docs/transformers/
3. Hijri Converter: https://hijri-converter.readthedocs.io/
4. langid.py: https://github.com/saffsd/langid.py
5. pandas Documentation: https://pandas.pydata.org/docs/

Document generated: January 2026
AlHaram Analytics v0.2.0