Google Reviews Pipeline for POIs (Aveiro)

Goal: Build end-to-end NLP pipeline to fetch, clean, and analyze reviews for Points of Interest (POIs) in Aveiro, using:

- **OSM-derived POIs** (from `pois_aveiro.csv` with EWKB geometry)
- **Google Places API (v1)** for proximity search and review extraction
- **Bilingual text preprocessing** (English & Portuguese with language-aware normalization)
- **NLP feature extraction** (TF-IDF for keywords, VADER sentiment for English)
- **Interactive visualizations** (ratings, sentiment, top places, wordclouds)
- Output **three-stage CSV pipeline**: raw → clean → enriched

> **Note:** Google Places reviews are typically limited to 5 reviews per place. Deduplication by `place_id` prevents duplicate results from overlapping search.

0. Setup & Requirements

Prerequisites

1. A **Google Cloud Project** with **Places API** enabled
2. A valid **API key** (set via environment variable `GOOGLE_API_KEY`)
3. The input file `pois_aveiro.csv` with `geom_pt` or `geom` column (EWKB POINT geometry, SRID=4326)

Key Features

- **Geometry parsing** from EWKB hex to (lon, lat) coordinates
- **Deduplication** across overlapping POI search circles
- **Bilingual NLP** (EN lemmatization + PT stemming with stopword removal)
- **Sentiment analysis** for English reviews (VADER)
- **TF-IDF keyword extraction** on processed text (without stopwords)
- **Multiple output formats** for exploratory analysis

Environment Variables

- `GOOGLE_API_KEY` : Your Google Places API key (fallback hardcoded for demo)

Rate Limits & Quotas

- Request delays are included to avoid hitting rate limits
- Reviews are limited to ~5 per place (Google API constraint)
- Adjust `MAX_TOTAL_POIS` to control overall execution time

In [ ]:
```
import os       # filesystem paths, env vars
INPUT_CSV_PRIMARY = "../../Milestone_2/pois_aveiro.csv"
INPUT_CSV_FALLBACK = "../../Milestone_2/pois_aveiro.csv"
OUTPUT_DIR = "../output"
RAW_REVIEWS_CSV = os.path.join(OUTPUT_DIR, "reviews_raw.csv")
CLEAN_REVIEWS_CSV = os.path.join(OUTPUT_DIR, "reviews_clean.csv")

# Google Places API
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY", "not_a_real_key")

# Fetch configuration
SLEEP_BETWEEN_REQUESTS = 0.1   # seconds
RADIUS_METERS = 25              # tighter radius to match OSM POI
MAX_PLACES_PER_POI = 1000       # max nearby results per POI we will cons
MAX_TOTAL_POIS = 1000           # safety limit for demonstration; set Non

INCLUDED_TYPES = None           # e.g., ["restaurant", "cafe"] or None fo

# Retry configuration
MAX_RETRIES = 3
BACKOFF_FACTOR = 1.6

os.makedirs(OUTPUT_DIR, exist_ok=True)
print("Output directory:", OUTPUT_DIR)
```

```
Output directory: ../output
```

In [ ]:
```
# === Core & Utilities ===
import time    # simple delays between requests
import json    # summary output
import math    # any minor math utilities
import re      # regex for text cleaning
import ast     # safe literal evaluation for parsed dicts

# === HTTP & DataFrames ===
import requests        # Google Places API calls
import pandas as pd    # data manipulation

# === Geometry ===
from shapely import wkb  # EWKB hex POINT parsing

# === Typing helpers ===
from typing import Optional, Tuple, List, Dict, Any

# === NLP & Language ===
import numpy as np                               # numeric ops
import nltk                                      # tokenization/resources
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer, RSLPStemmer
from nltk.tokenize import word_tokenize
try:
    from langdetect import detect                # language detection (opt
    HAVE_LANGDETECT = True
except Exception:
    HAVE_LANGDETECT = False

# === Features & Sentiment ===
from sklearn.feature_extraction.text import TfidfVectorizer
try:
    from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
```

```
    HAVE_VADER = True
    sid = SentimentIntensityAnalyzer()
except Exception:
    HAVE_VADER = False
    sid = None

# === Visualization ===
import matplotlib.pyplot as plt
import seaborn as sns
try:
    from wordcloud import WordCloud
    HAVE_WORDCLOUD = True
except Exception:
    HAVE_WORDCLOUD = False

# === Transformers (optional for PT sentiment) ===
try:
    from transformers import pipeline
    HAVE_TRANSFORMERS = True
except Exception:
    HAVE_TRANSFORMERS = False

# === Topic Modeling (optional for LDA) ===
try:
    from gensim import corpora
    from gensim.models import LdaMulticore
    HAVE_GENSIM = True
except ImportError:
    HAVE_GENSIM = False

print("Imports loaded. Optional libs:", {
    "langdetect": HAVE_LANGDETECT,
    "vader": HAVE_VADER,
    "wordcloud": HAVE_WORDCLOUD,
    "transformers": HAVE_TRANSFORMERS,
    "gensim": HAVE_GENSIM,
})
```

```
/home/miragaia/Documents/5_ANO/ICD/PROJECT_ICD/.venv/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Imports loaded. Optional libs: {'langdetect': True, 'vader': True, 'wordcloud': True, 'transformers': True}
```

1. Geometry Parsing & API Client Functions

Purpose: Define helper functions for geometric transformations and Google Places API interactions.

Process

1. **EWKB to Coordinates**: Parse OSM EWKB hex-encoded POINT geometries (SRID 4326) into standard (lon, lat) tuples
2. **API Headers & Field Masking**: Construct proper HTTP headers with field masks to request specific place attributes (name, ID, reviews, location, rating, type)

3. **Nearby Search**: Query Google Places API for places within a search radius around each POI
4. **Review Extraction**: Parse the API response and flatten nested review structures into rows

Key Functions

- `ewkb_hex_point_to_lonlat()` : Convert EWKB hex to geographic coordinates
- `places_search_nearby()` : Execute Nearby Search query (single page, no pagination)
- `extract_reviews_from_places()` : Extract and flatten reviews from place objects

Notes

- EWKB format commonly appears in PostGIS/OSM exports (e.g., `0101000020E6100000...`)
- Single-page Nearby Search avoids field mask conflicts; pagination available as future enhancement
- Deduplication of places by ID prevents duplicate reviews from overlapping search circles

In [3]:
```
def ewkb_hex_point_to_lonlat(hex_str: str) -> Optional[Tuple[float, float
    """
    Convert EWKB hex POINT (SRID=4326) to (lon, lat).
    """
    if not isinstance(hex_str, str) or not hex_str:
        return None
    try:
        geom = wkb.loads(bytes.fromhex(hex_str))
        if geom.geom_type == "Point":
            return (geom.x, geom.y)
    except Exception:
        pass
    return None


def _places_headers(field_mask: str) -> Dict[str, str]:
    return {
        "X-Goog-FieldMask": field_mask,
        "X-Goog-Api-Key": GOOGLE_API_KEY,
        "Content-Type": "application/json",
    }


def places_search_nearby(
    lat: float,
    lon: float,
    radius: int = 25,
    included_types: Optional[List[str]] = None,
    max_results: int = 20,
) -> List[Dict[str, Any]]:
    """
    Nearby Search (single page) following the Milestone 2 approach.
    Returns a list of place dicts (no pagination).
```

```
    """
    url = "https://places.googleapis.com/v1/places:searchNearby"

    payload: Dict[str, Any] = {
        "locationRestriction": {
            "circle": {
                "center": {"latitude": lat, "longitude": lon},
                "radius": radius,
            }
        },
        "maxResultCount": min(max_results, 20),
    }
    if included_types:
        payload["includedTypes"] = included_types

    field_mask = (
        "places.displayName,places.id,places.reviews,places.location,plac
    )

    try:
        response = requests.post(
            url,
            params={"key": GOOGLE_API_KEY},
            json=payload,
            headers=_places_headers(field_mask),
            timeout=20,
        )
        response.raise_for_status()
    except requests.exceptions.RequestException as e:
        print(f"Error during API request: {e}")
        return []

    data = response.json()
    places = data.get("places", []) if isinstance(data, dict) else []
    return places


def extract_reviews_from_places(places: List[Dict[str, Any]]) -> List[Dic
    rows = []
    for p in places:
        pid = p.get("id")
        pname = (p.get("displayName") or {}).get("text", None)
        prating = p.get("rating")
        ptype = p.get("primaryType")
        ploc = p.get("location") or {}
        reviews = p.get("reviews") or []
        for r in reviews:
            rows.append({
                "place_id": pid,
                "place_name": pname,
                "place_rating": prating,
                "place_primary_type": ptype,
                "place_location": ploc,
                "author_name": (r.get("authorAttribution") or {}).get("di
                "rating": r.get("rating"),
                "review_text": (r.get("text") or {}).get("text", ""),
                "publish_time": r.get("publishTime"),
            })
    return rows
```

2. Load POIs & Fetch Reviews via Nearby Search

Purpose: Iterate over OSM-derived POIs, execute Nearby Search for each location, and aggregate all discovered reviews into a **raw output CSV**.

> ⚠️ **SKIP THIS SECTION** if `reviews_raw.csv` is available and want to work with existing data (see Section 2b instead).

Process

1. **Load POI CSV**: Read `pois_aveiro.csv` and verify it contains EWKB geometry column (`geom_pt` or `geom`)
2. **Parse Coordinates**: For each POI row, extract (lon, lat) from EWKB hex string
3. **Execute Nearby Search**: Query Google Places API within `RADIUS_METERS` around each POI to discover nearby places
4. **Deduplicate Places**: Track `place_id` globally to prevent duplicate processing from overlapping search circles
5. **Extract Reviews**: For each unique place, flatten nested review structures into individual rows
6. **Attach Context**: Enrich each review with POI metadata (gid, amenity type, source name)
7. **Parse Location**: Convert embedded place location objects to separate `lat` and `lon` columns for convenience
8. **Save Raw Output**: Write all reviews to `reviews_raw.csv` with original fields + POI context

Key Parameters

- `RADIUS_METERS` : Search radius around each POI (25m = roughly OSM POI precision)
- `MAX_PLACES_PER_POI` : Limit results per Nearby Search query (1000 = API max)
- `MAX_TOTAL_POIS` : Safety limit for total POIs processed (set `None` to process all)
- `INCLUDED_TYPES` : Optional list to filter places (e.g., `["restaurant", "cafe"]`), or `None` for all types

Output

- `reviews_raw.csv` : Raw API results with columns: place_id, place_name, author_name, rating, review_text, publish_time, lat, lon, place_rating, place_primary_type, poi_context...
- Deduplication count printed to show unique places discovered

Alternative: If you already have fetched reviews and lost API access, skip to **Section 2b** to load from existing CSV.

In [5]:
```
if os.path.exists(INPUT_CSV_PRIMARY):
    pois_path = INPUT_CSV_PRIMARY
```

```
elif os.path.exists(INPUT_CSV_FALLBACK):
    pois_path = INPUT_CSV_FALLBACK
else:
    raise FileNotFoundError(
        f"Could not find input CSV at {INPUT_CSV_PRIMARY} or fallback {IN
    )

print("Loading:", pois_path)
df_pois = pd.read_csv(pois_path, low_memory=False)
print("Rows:", len(df_pois))

geom_col = None
for cand in ["geom_pt", "geom"]:
    if cand in df_pois.columns:
        geom_col = cand
        break
if geom_col is None:
    raise ValueError("No EWKB point column ('geom_pt' or 'geom') found in

out_rows = []
processed = 0
seen_place_ids = set()

for idx, row in df_pois.iterrows():
    if MAX_TOTAL_POIS and processed >= MAX_TOTAL_POIS:
        break

    hex_point = row.get(geom_col)
    lonlat = ewkb_hex_point_to_lonlat(hex_point) if isinstance(hex_point,
    if not lonlat:
        continue

    lon, lat = lonlat

    places = places_search_nearby(
        lat=lat,
        lon=lon,
        radius=RADIUS_METERS,
        included_types=INCLUDED_TYPES,
        max_results=MAX_PLACES_PER_POI,
    )

    unique_places = []
    for p in places:
        pid = p.get("id")
        if pid and pid not in seen_place_ids:
            unique_places.append(p)
            seen_place_ids.add(pid)

    rows = extract_reviews_from_places(unique_places)

    for r in rows:
        r["poi_row_index"] = idx
        if "gid" in df_pois.columns:
            r["poi_gid"] = row.get("gid")
        for c in ["amenity", "shop", "tourism", "name"]:
            if c in df_pois.columns:
                r[f"poi_{c}"] = row.get(c)

    out_rows.extend(rows)
```

```
    processed += 1

if out_rows:
    df_raw = pd.DataFrame(out_rows)
    def parse_loc(x):
        if isinstance(x, dict):
            return x
        try:
            return ast.literal_eval(x)
        except Exception:
            return {}
    df_raw["place_location"] = df_raw["place_location"].apply(parse_loc)
    df_raw["lat"] = df_raw["place_location"].apply(lambda d: d.get("latit
    df_raw["lon"] = df_raw["place_location"].apply(lambda d: d.get("longi

    df_raw.to_csv(RAW_REVIEWS_CSV, index=False)
    print(f"Saved {len(df_raw)} raw reviews -> {RAW_REVIEWS_CSV}")
    print(f"Total unique places fetched: {len(seen_place_ids)}")
else:
    df_raw = pd.DataFrame()
    print("No reviews fetched.")

display(df_raw.head(10))
```

```
Loading: ../../Milestone_2/pois_aveiro.csv
Rows: 13258
Rows: 13258
Error during API request: 500 Server Error: Internal Server Error for url:
https://places.googleapis.com/v1/places:searchNearby?key=AIzaSyAwme_k4xStL
v2_bLFAckn55hJ1TNF8L8A
Error during API request: 500 Server Error: Internal Server Error for url:
https://places.googleapis.com/v1/places:searchNearby?key=AIzaSyAwme_k4xStL
v2_bLFAckn55hJ1TNF8L8A
Saved 6800 raw reviews -> ../output/reviews_raw.csv
Total unique places fetched: 2356
Saved 6800 raw reviews -> ../output/reviews_raw.csv
Total unique places fetched: 2356
```

	place_id	place_name	place_rating	place_primary_type	
0	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
1	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
2	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
3	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
4	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
5	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
6	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
7	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
8	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
9	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi

2b. Load Existing Reviews from CSV (Skip API Fetch)

Purpose: Load previously fetched reviews from `reviews_raw.csv` when API access is unavailable or when working with existing data.

Use Case

- **API quota exhausted** or API key no longer available
- **Working offline** with previously collected data
- **Reproducible analysis** using a fixed snapshot of reviews

- **Faster iteration** during development (skip expensive API calls)

Process

1. **Check for existing CSV**: Verify `reviews_raw.csv` exists in the output directory
2. **Load DataFrame**: Read CSV with all original columns (place_id, place_name, author_name, rating, review_text, etc.)
3. **Parse location data**: Convert `place_location` from string representation back to dictionary if needed
4. **Extract coordinates**: Ensure `lat` and `lon` columns are properly populated
5. **Validate data**: Check for required columns and print summary statistics

Key Columns Expected

- `place_id` : Google Place ID (unique identifier)
- `place_name` : Name of the place
- `author_name` : Review author
- `rating` : Star rating (1-5)
- `review_text` : Original review text
- `publish_time` : ISO timestamp of review
- `lat` , `lon` : Geographic coordinates
- `place_rating` : Overall place rating
- `place_primary_type` : Place category (restaurant, cafe, etc.)
- `poi_*` columns: Original POI metadata from OSM

Output

- `df_raw` : Pandas DataFrame ready for downstream processing (identical structure to API fetch output)
- Summary statistics printed (total reviews, date range, language distribution preview)

Note: Run **either** Section 2 (API fetch) **OR** Section 2b (load from CSV), not both. Comment out the section you don't need.

In [ ]:
```python
# Load existing reviews from CSV (alternative to API fetch in Section 2)
if os.path.exists(RAW_REVIEWS_CSV):
    print(f"Loading existing reviews from: {RAW_REVIEWS_CSV}")
    df_raw = pd.read_csv(RAW_REVIEWS_CSV, low_memory=False)

    # Parse place_location if it's stored as string
    def parse_loc(x):
        if isinstance(x, dict):
            return x
        if pd.isna(x):
            return {}
        try:
            return ast.literal_eval(str(x))
        except Exception:
            return {}

    # Ensure location parsing and coordinate extraction
```

```
    if "place_location" in df_raw.columns:
        df_raw["place_location"] = df_raw["place_location"].apply(parse_l

    # Extract or verify lat/lon columns
    if "lat" not in df_raw.columns or df_raw["lat"].isna().all():
        df_raw["lat"] = df_raw["place_location"].apply(lambda d: d.get("l
    if "lon" not in df_raw.columns or df_raw["lon"].isna().all():
        df_raw["lon"] = df_raw["place_location"].apply(lambda d: d.get("l

    print(f"Loaded {len(df_raw)} reviews")
    print(f"Unique places: {df_raw['place_id'].nunique()}")
    print(f"Date range: {df_raw['publish_time'].min()} to {df_raw['publis
    print(f"Average rating: {df_raw['rating'].mean():.2f}")

    display(df_raw.head(10))
else:
    raise FileNotFoundError(
        f"CSV file not found: {RAW_REVIEWS_CSV}\n"
        f"Please run Section 2 (API fetch) first to generate the raw revi
        f"or check that the OUTPUT_DIR path is correct."
    )
```

```
Loading existing reviews from: ../output/reviews_raw.csv
Loaded 6800 reviews
Unique places: 1655
Date range: 2009-10-22T19:26:04.476521Z to 2025-12-08T08:31:55.750368399Z
Average rating: 4.32
```

	place_id	place_name	place_rating	place_primary_type	
0	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
1	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
2	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
3	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
4	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
5	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
6	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
7	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
8	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
9	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi

3. Bilingual Text Preprocessing & Normalization

Purpose: Clean raw review text and normalize it using language-aware techniques, producing a `text_processed` column without stopwords for downstream NLP tasks.

Input: Uses `df_raw` DataFrame from either Section 2 (API fetch) or Section 2b (load from CSV)

Process

1. **Text Cleaning**: Remove HTML tags, URLs, excessive whitespace, and non-

alphanumeric characters (preserving accented characters for Portuguese/Spanish)
2. **Language Detection**: Use `langdetect` to identify whether each review is in English (en), Portuguese (pt), or another language; optionally filter to EN/PT only
3. **Language-Specific Tokenization**:
   - **English**: Tokenize → remove EN stopwords → lemmatize (WordNet) → keep tokens > 2 chars
   - **Portuguese**: Tokenize → remove PT stopwords → stem (RSLP - Snowball Stemmer for Portuguese) → keep tokens > 2 chars
   - **Other languages**: Tokenize → remove bilingual stopwords → apply English lemmatization as fallback
4. **Reconstruct Processed Text**: Join tokens back into `text_processed` column (used by TF-IDF and wordcloud)
5. **Compute Token Stats**: Count tokens per review for exploratory analysis
6. **Save Clean Output**: Write to `reviews_clean.csv` with added columns: review_text_clean, lang, tokens, token_count, text_processed

Language-Specific Methods

- **English Lemmatization**: WordNet lemmatizer (reduces "running", "runs" → "run")
- **Portuguese Stemming**: RSLP stemmer (reduces "casas", "casa" → "cas")
- **Stopword Removal**: Pre-computed sets for EN (179 words) and PT (228 words)

Key Columns in Output

- `review_text_clean` : Original text with only punctuation/URLs removed (light cleaning)
- `lang` : Detected language code (en, pt, or null if detection failed)
- `tokens` : List of normalized tokens (stopwords already removed)
- `text_processed` : String reconstructed from tokens (ready for TF-IDF and visualization)

Graceful Fallbacks

- If `langdetect` unavailable: skips language detection, uses bilingual stopword list
- Unknown languages: defaults to English lemmatization
- Missing or empty reviews: handled as empty token lists

Notes

- `text_processed` is the key column for downstream TF-IDF and wordcloud (not `review_text_clean`)
- Stopword removal + lemmatization/stemming significantly reduces noise in keyword extraction

In [ ]:
```
# Stopwords for EN and PT
stop_en = set(stopwords.words('english'))
try:
    stop_pt = set(stopwords.words('portuguese'))
except Exception:
```

```
    stop_pt = set()
stop_all = stop_en.union(stop_pt)

lemmatizer = WordNetLemmatizer()
rslp = RSLPStemmer()


def clean_text(s: str) -> str:
    s = str(s)
    s = re.sub(r"<[^>]+>", " ", s)                # HTML
    s = re.sub(r"http\S+|www\S+", " ", s)        # URLs
    s = re.sub(r"[^\w\sáéíóúàèìòùâêîôûçãõÁÉÍÓÚÀÈÌÒÙÂÊÎÔÛÇÃÕ]", " ", s)  #
    s = re.sub(r"\s+", " ", s)
    return s.strip().lower()


def language_or_none(s: str) -> str:
    if not HAVE_LANGDETECT:
        return None
    try:
        return detect(s)
    except Exception:
        return None


def simple_tokenize(s: str) -> list:
    # Use nltk's word_tokenize; fallback to regex if it fails
    try:
        return [t for t in word_tokenize(s) if t]
    except Exception:
        return re.findall(r"\b\w+\b", s, flags=re.UNICODE)


def tokens_for_lang(s: str, lang: str | None) -> list:
    toks = simple_tokenize(s)
    if lang == 'pt':
        toks = [t for t in toks if t.isalpha() and t not in stop_pt and l
        toks = [rslp.stem(t) for t in toks]
        return toks
    elif lang == 'en':
        toks = [t for t in toks if t.isalpha() and t not in stop_en and l
        toks = [lemmatizer.lemmatize(t) for t in toks]
        return toks
    else:
        # Unknown language: use a bilingual stopword list, no language-sp
        toks = [t for t in toks if t.isalpha() and t.lower() not in stop_
        # Default to English lemmatizer as a light normalization
        toks = [lemmatizer.lemmatize(t) for t in toks]
        return toks


if not df_raw.empty:
    df_clean = df_raw.copy()
    df_clean["review_text_clean"] = df_clean["review_text"].fillna("").ma

    # Language detection (EN/PT only)
    if HAVE_LANGDETECT:
        df_clean["lang"] = df_clean["review_text_clean"].map(language_or_
        df_clean = df_clean[df_clean["lang"].isin(["en", "pt"]).fillna(Tr
    else:
```

```
        df_clean["lang"] = np.nan

    # Tokenization and normalization per language
    df_clean["tokens"] = df_clean.apply(lambda r: tokens_for_lang(r["revi
    df_clean["token_count"] = df_clean["tokens"].map(len)
    # Reconstruct a processed text without stopwords for downstream featu
    df_clean["text_processed"] = df_clean["tokens"].map(lambda toks: " ".

    df_clean.to_csv(CLEAN_REVIEWS_CSV, index=False)
    print(f"Saved clean reviews -> {CLEAN_REVIEWS_CSV}")
else:
    df_clean = pd.DataFrame()

display(df_clean.head(10))
```

```
Saved clean reviews -> ../output/reviews_clean.csv
```

	place_id	place_name	place_rating	place_primary_type	
0	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
1	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
2	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
3	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
4	ChIJgymEZASYIw0RKRkhKH9TfRQ	bp	4.1	gas_station	40.63
5	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
6	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
7	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
8	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi
9	ChIJdZmB9g-jIw0RBVGjwMFX79Y	Café BP Maia	4.3	cafe	'longi

10 rows × 22 columns

4. Feature Extraction & Sentiment Analysis

Purpose: Compute quantitative features from processed reviews (keyword importance via TF-IDF) and bilingual sentiment polarity scores.

Process

1. **TF-IDF Vectorization**: Fit TF-IDF model on `text_processed` (stopword-removed, normalized text)
   - Extract unigrams and bigrams (1-2 word phrases)
   - Minimum document frequency = 2 (appears in ≥2 reviews to reduce noise)
   - Maximum features = 5000 (most frequent/important terms)
   - Aggregate to compute mean TF-IDF per place
2. **Bilingual Sentiment Analysis**
   - **English**: Apply VADER (Valence Aware Dictionary and sEntiment Reasoner) to `review_text_clean`
     - VADER designed for social media text; returns `compound` score in [-1, 1] range
     - Score = -1 (most negative), 0 (neutral), +1 (most positive)
   - **Portuguese**: Use HuggingFace `ysentimiento/bertweet-pt-sentiment` (text classification)
     - We compute a continuous `compound` score as: $\text{compound} = P(\text{POSITIVE}) - P(\text{NEGATIVE})$, using the model's softmax probabilities
     - This yields a value in approximately [-1, 1]; `NEUTRAL` mass is implicit
     - If Transformers are unavailable, fallback to TextBlob polarity in [-1, 1]
   - Both methods preserve punctuation and text emphasis
3. **Per-Place Keyword Extraction**: For each place_id, compute top 10 keywords by mean TF-IDF score
4. **Save Enriched Output**: Write to `reviews_enriched.csv` with added columns: tf_idf_features, sentiment_compound (bilingual)

Key Outputs

- `top_keywords_per_place` : Dictionary mapping place_id → [(keyword, tfidf_score), ...]
- `sentiment_compound` : Float in [-1, 1] for both EN (VADER) and PT (HuggingFace or TextBlob fallback)
- `X` : Sparse TF-IDF matrix (scipy.sparse CSR format) for downstream modeling

Notes

- Use `text_processed` for TF-IDF (no stopwords, normalized) to get meaningful keywords
- Use `review_text_clean` for sentiment (punctuation preserved, needed for VADER emphasis detection)
- Bilingual sentiment enables comparative analysis across languages with consistent scale
- bertweet-pt is stronger for Portuguese than simple lexicon-based approaches
- Both TF-IDF and sentiment results saved to enable further analysis (topic

modeling, emotion classification, etc.)

In [6]:
```
if not df_clean.empty:
    # Reset index so TF-IDF matrix rows align with DataFrame rows
    df_idx = df_clean.reset_index(drop=True)

    # TF-IDF on processed text (stopwords removed, lemmatized/stemmed)
    texts = df_idx["text_processed"].fillna("").tolist()
    tfidf = TfidfVectorizer(max_features=5000, ngram_range=(1,2), min_df=
    X = tfidf.fit_transform(texts)
    vocab = np.array(tfidf.get_feature_names_out())

    # Compute per-place top keywords by mean TF-IDF
    top_keywords_per_place = {}
    for pid, sub in df_idx.groupby("place_id"):
        idx = sub.index
        if len(idx) == 0:
            continue
        vec = X[idx].mean(axis=0)
        arr = np.asarray(vec).ravel()
        top_idx = arr.argsort()[-10:][::-1]
        top_keywords_per_place[pid] = list(zip(vocab[top_idx], arr[top_id

    # Sentiment: VADER for English, HuggingFace (bertweet-pt) for Portugu
    df_clean["sentiment_compound"] = np.nan

    # English sentiment (VADER)
    if HAVE_VADER:
        en_mask = df_clean["lang"].eq("en")
        df_clean.loc[en_mask, "sentiment_compound"] = df_clean[en_mask]["
            lambda s: sid.polarity_scores(s)["compound"]
        )

    # Portuguese sentiment (HuggingFace bertweet-pt-sentiment). Fallback
    pt_mask = df_clean["lang"].eq("pt")
    pt_senti = None
    if HAVE_TRANSFORMERS:
        try:
            pt_senti = pipeline("text-classification", model="pysentimien
        except Exception as e:
            print("Transformers pipeline init failed:", e)
            pt_senti = None

    def pt_compound_score(text: str) -> float:
        if not isinstance(text, str) or not text.strip():
            return np.nan
        if pt_senti is None:
            return np.nan
        try:
            # Use top_k=None to get all class scores (replaces deprecated
            res = pt_senti(text, top_k=None)
            # Expected format: [{'label': 'NEG', 'score': 0.x}, {'label':
            if isinstance(res, list) and len(res) > 0:
                # Build score dict from all returned labels
                scores = {}
                for item in res:
                    label = str(item.get("label", "")).upper()
                    score_val = float(item.get("score", 0.0))
                    # Map various label formats to standard names
```

```
                    if "POS" in label:
                        scores["POSITIVE"] = score_val
                    elif "NEG" in label:
                        scores["NEGATIVE"] = score_val
                    elif "NEU" in label:
                        scores["NEUTRAL"] = score_val

                ppos = scores.get("POSITIVE", 0.0)
                pneg = scores.get("NEGATIVE", 0.0)
                # Compute compound as difference: positive - negative (ra
                compound = float(ppos - pneg)
                return compound
            return 0.0  # Fallback if format unexpected
        except Exception as e:
            # Silent fallback for individual review errors
            return np.nan

    if pt_senti is not None:
        df_clean.loc[pt_mask, "sentiment_compound"] = df_clean.loc[pt_mas
        print("Portuguese sentiment computed with bertweet-pt (HuggingFac
    else:
        try:
            from textblob import TextBlob
            df_clean.loc[pt_mask, "sentiment_compound"] = df_clean.loc[pt
                lambda s: TextBlob(s).sentiment.polarity
            )
            print("Transformers unavailable; Portuguese sentiment via Tex
        except ImportError:
            print("TextBlob not available; Portuguese sentiment skipped.

    # Persist enriched
    df_enriched = df_clean.copy()
    df_enriched.to_csv(os.path.join(OUTPUT_DIR, "reviews_enriched.csv"),
    print("Saved reviews_enriched.csv")
else:
    top_keywords_per_place = {}
    df_enriched = pd.DataFrame()

# Preview keywords for first 3 places
for i, (pid, kws) in enumerate(top_keywords_per_place.items()):
    if i >= 3:
        break
    print(f"Place {pid} top keywords:")
    for w, score in kws:
        print("  ", w, f"{score:.3f}")
```

```
emoji is not installed, thus not converting emoticons or emojis into text.
Install emoji: pip3 install emoji==0.6.0
Device set to use cpu
Device set to use cpu
```

```
Portuguese sentiment computed with bertweet-pt (HuggingFace)
Saved reviews_enriched.csv
Place ChIJ-1iL6JiZIw0RDUrtC7a89ZI top keywords:
   girl 0.134
   happy 0.111
   cabel 0.109
   gabriel 0.106
   first 0.099
   desd 0.071
   result 0.070
   incri 0.065
   destaqu 0.065
   sent bem 0.061
Place ChIJ-6D9iCOXIw0RrrI1s929jkA top keywords:
   cabel 0.294
   ter 0.105
   equip incr 0.103
   real 0.100
   faz 0.099
   trat 0.090
   poss 0.086
   result 0.083
   menin 0.080
   loir 0.077
Place ChIJ-7D2DEyRIw0RajlTjh1ovls top keywords:
   sunset 0.131
   beach 0.116
   served 0.105
   food 0.088
   place right 0.079
   food drink 0.077
   crowded 0.077
   cocktail 0.074
   reccomend 0.073
   friendly 0.071
```

5. Exploratory Data Visualization (Bilingual)

Purpose: Generate visual summaries split by detected language (English vs. Portuguese) to compare patterns across linguistic contexts.

Charts Produced

1. Rating Distributions

- Separate histograms with KDE for English and Portuguese reviews
- Shows star rating (1-5) frequency distribution per language

2. Sentiment Distributions (Bilingual)

- **English**: VADER compound scores [-1, 1] with histogram + KDE
- **Portuguese**: HuggingFace bertweet-pt compound scores [-1, 1] with histogram + KDE
- Enables direct comparison of sentiment polarity across languages

3. Top Places by Review Count

- Bar charts showing top 10 most-reviewed places per language
- Identifies popular venues within each linguistic community

4. Word Clouds (Language-Specific Tokenization)

- **English**: Lemmatized tokens from `text_processed`
- **Portuguese**: Unstemmed tokens (preserves readability) with stopword removal
- Visual representation of most frequent terms per language

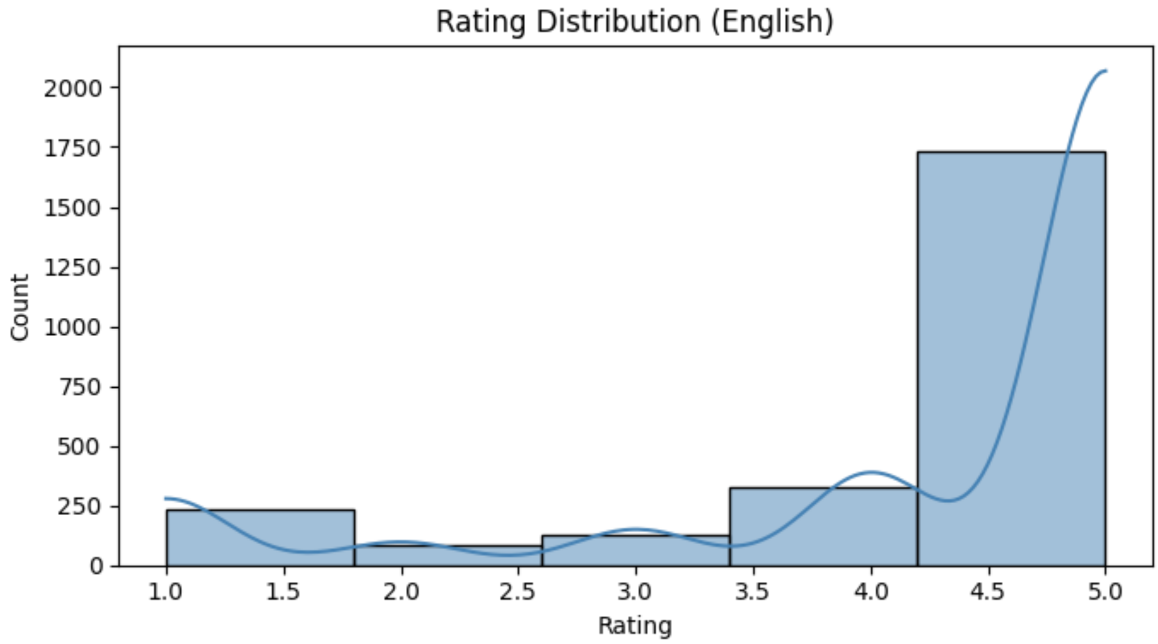
Notes

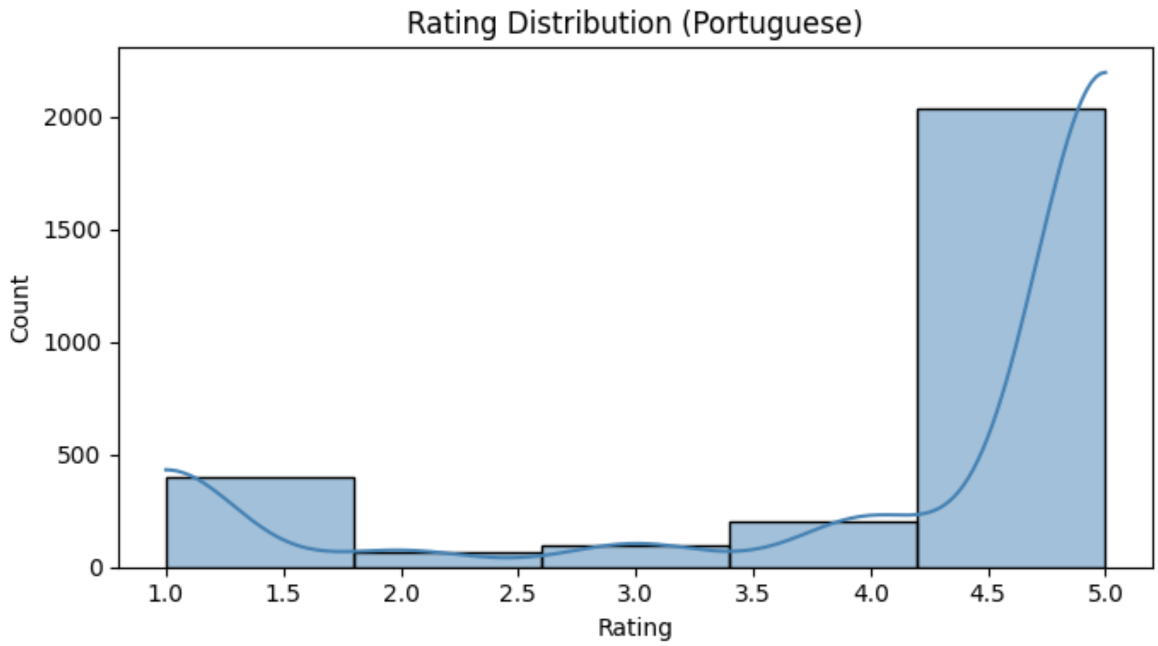
- All visualizations gracefully skip empty language slices with informative messages
- Portuguese sentiment now uses the same [-1, 1] scale as English for direct comparison
- Word clouds use different preprocessing: EN uses lemmas, PT uses raw tokens for better interpretability

In [7]:
```
# Bilingual visualizations and comparative analysis
if df_enriched.empty:
    print("No data to visualize.")
else:
    df_en = df_enriched[df_enriched["lang"].eq("en")]
    df_pt = df_enriched[df_enriched["lang"].eq("pt")]

    # Helper functions for visualizations
    def plot_ratings(df_lang, label):
        if df_lang.empty:
            print(f"No {label} reviews for rating histogram.")
            return
        plt.figure(figsize=(7,4))
        sns.histplot(pd.to_numeric(df_lang["rating"], errors="coerce").dr
        plt.title(f"Rating Distribution ({label})")
        plt.xlabel("Rating")
        plt.ylabel("Count")
        plt.tight_layout(); plt.show()

    def plot_sentiment(df_lang, label, method_name):
        """Plot sentiment distribution for any language with proper label
        if df_lang.empty or df_lang["sentiment_compound"].notna().sum() =
            print(f"No {label} sentiment data to plot.")
            return
        plt.figure(figsize=(7,4))
        sns.histplot(df_lang["sentiment_compound"].dropna(), bins=30, kde
        plt.title(f"Sentiment Distribution ({label}, {method_name})")
        plt.xlabel("Compound Score [-1, 1]")
        plt.ylabel("Count")
        plt.tight_layout(); plt.show()

    def plot_top_places(df_lang, label):
        if df_lang.empty:
            print(f"No {label} reviews for top places.")
            return
        top_places = df_lang["place_name"].value_counts().nlargest(10).re
        top_places.columns = ["place_name", "n_reviews"]
        plt.figure(figsize=(9,5))
        sns.barplot(data=top_places, y="place_name", x="n_reviews", palet
```

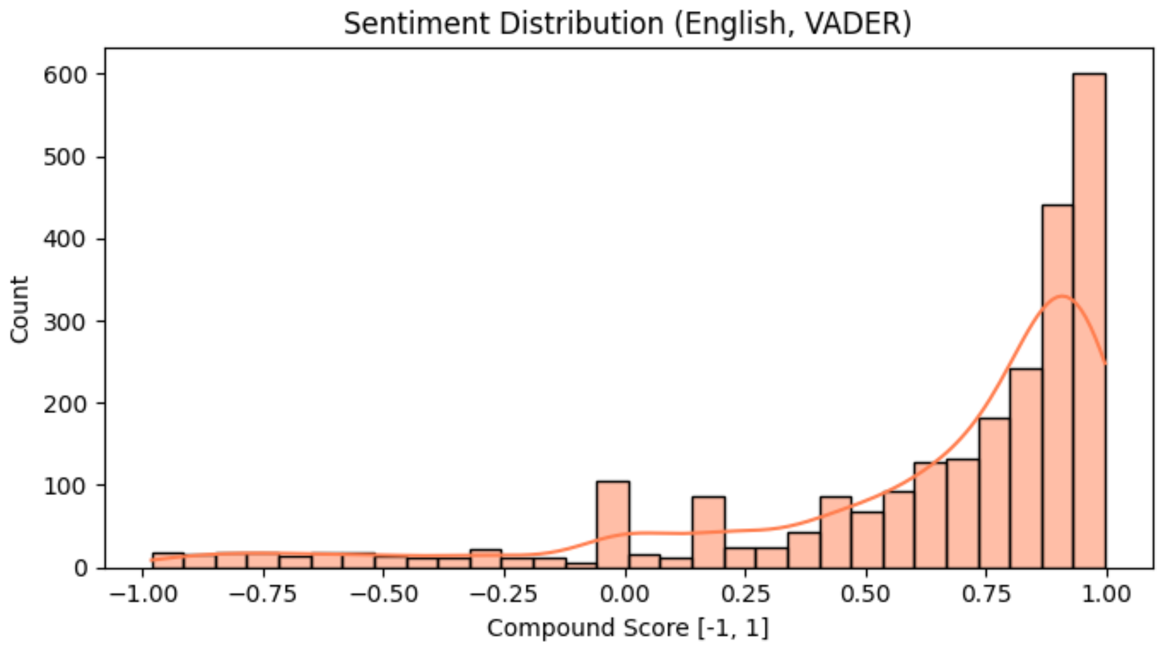
```
        plt.title(f"Top 10 Places by Reviews ({label})")
        plt.xlabel("# Reviews"); plt.ylabel("")
        plt.tight_layout(); plt.show()

    def plot_wordcloud(df_lang, label):
        if not HAVE_WORDCLOUD:
            print("Wordcloud not available: package not installed.")
            return
        if df_lang.empty:
            print(f"No {label} reviews for wordcloud.")
            return
        if label.lower().startswith("portuguese"):
            # Rebuild tokens without stemming for Portuguese wordcloud
            tokens = []
            for txt in df_lang["review_text_clean"].dropna():
                toks = simple_tokenize(txt)
                toks = [t for t in toks if t.isalpha() and t not in stop_
                tokens.extend(toks)
            txt = " ".join(tokens)
        else:
            txt = " ".join(df_lang["text_processed"].dropna().tolist())
        if len(txt.strip()) == 0:
            print(f"No text available for wordcloud ({label}).")
            return
        wc = WordCloud(width=1200, height=600, background_color='white',
        plt.figure(figsize=(12,6))
        plt.imshow(wc, interpolation='bilinear')
        plt.axis('off')
        plt.title(f"WordCloud ({label})")
        plt.show()

    # === 1. Rating Distributions ===
    print("\n=== Rating Distributions ===")
    plot_ratings(df_en, "English")
    plot_ratings(df_pt, "Portuguese")

    # === 2. Sentiment Distributions (Bilingual) ===
    print("\n=== Sentiment Distributions ===")
    plot_sentiment(df_en, "English", "VADER")
    plot_sentiment(df_pt, "Portuguese", "BERTweet-PT")

    # === 3. Top Places by Review Count ===
    print("\n=== Top Places by Review Count ===")
    plot_top_places(df_en, "English")
    plot_top_places(df_pt, "Portuguese")

    # === 4. Word Clouds (Language-Specific) ===
    print("\n=== Word Clouds ===")
    plot_wordcloud(df_en, "English")
    plot_wordcloud(df_pt, "Portuguese")
```

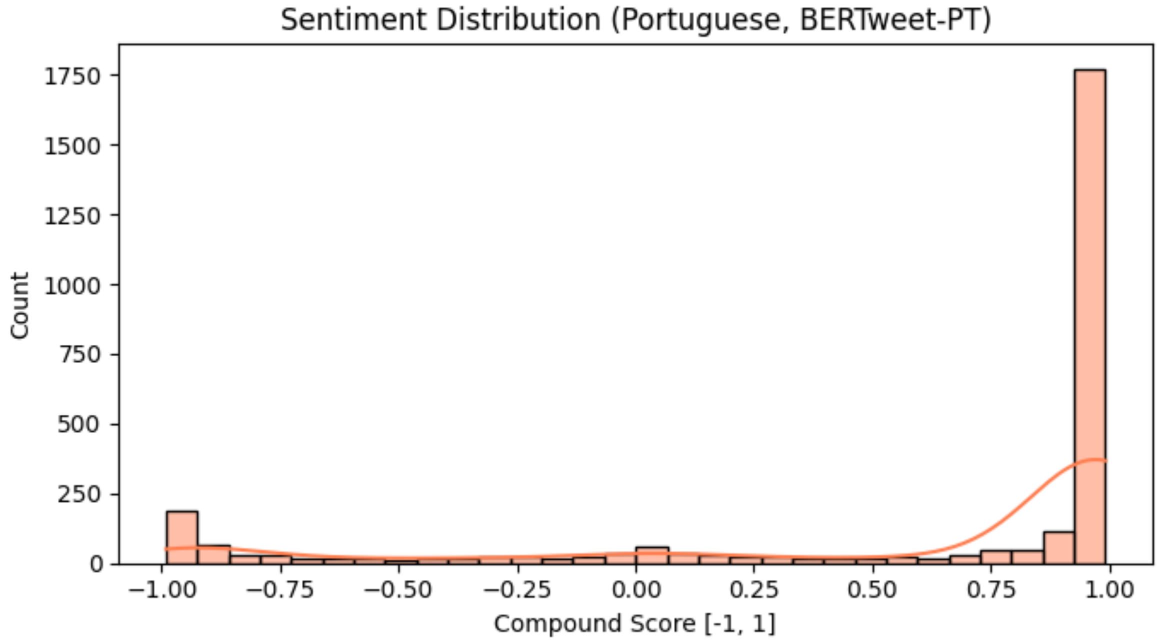
```
=== Rating Distributions ===
```



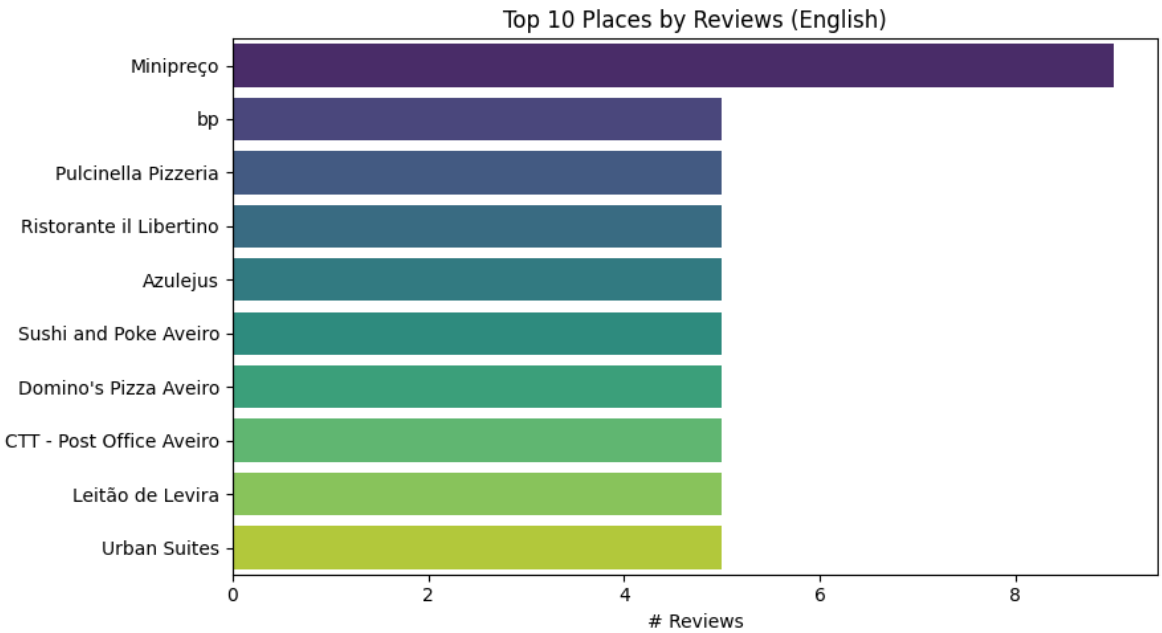



```
=== Sentiment Distributions ===
```

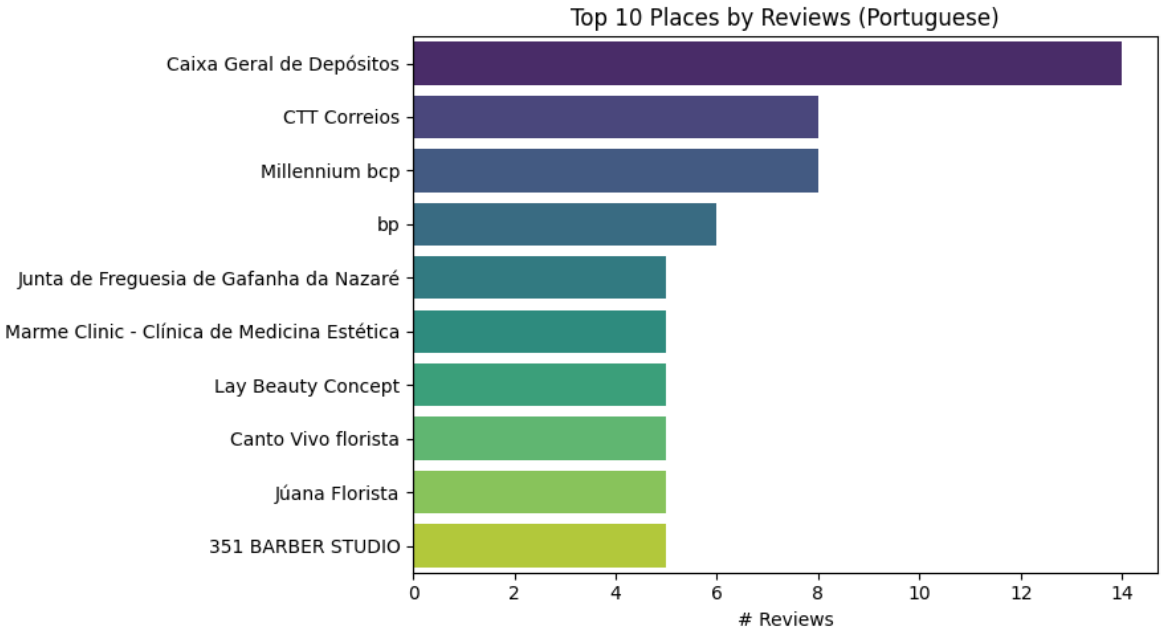




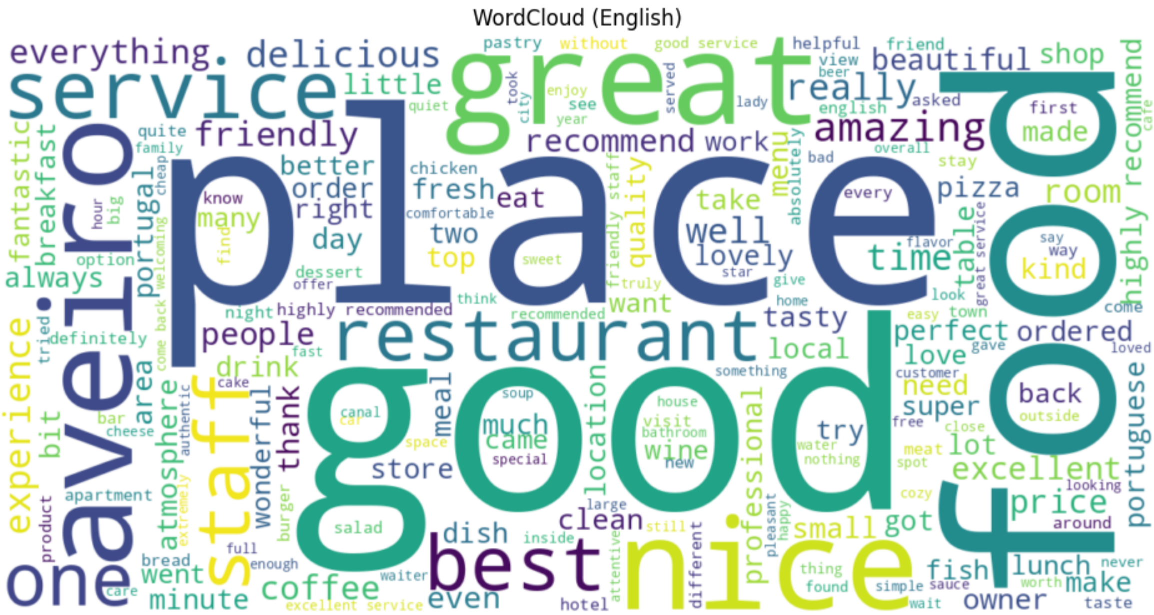

```
=== Top Places by Review Count ===
/tmp/ipykernel_5558/2347132612.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for
the same effect.

  sns.barplot(data=top_places, y="place_name", x="n_reviews", palette="vir
idis")
```




```
/tmp/ipykernel_5558/2347132612.py:39: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be remove
d in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for
the same effect.

  sns.barplot(data=top_places, y="place_name", x="n_reviews", palette="vir
idis")
```


Top 10 Places by Reviews (Portuguese)
Caixa Geral de Depósitos
CTT Correios
Millennium bcp
bp
Junta de Freguesia de Gafanha da Nazaré
Marme Clinic - Clínica de Medicina Estética
Lay Beauty Concept
Canto Vivo florista
Júana Florista
351 BARBER STUDIO
0
2
4
6
8
10
12
14
Reviews

=== Word Clouds ===


WordCloud (English)
place
good
food
great
aveiro
service
staff
nice
best
restaurant
one
everything
delicious
friendly
amazing
recommend
beautiful
really
little
experience
fantastic
breakfast
always
portugal
atmosphere
people
better
order
right
day
fresh
eat
well
lovely
time
pizza
room
highly recommend
tasty
local
perfect
ordered
love
back
super
excellent
price
portuguese
lot
fish
lunch
make
owner
coffee
minute
even
dish
clean
small
got
location
professional
wine
came
much
store
meal
thank
wonderful
drink
area
bit
went
try
need
made
shop
quality
take
work
kind
menu
want
two
top
many


WordCloud (Portuguese)
atendimento
recomendo
sempre
serviço
cliente
loja
excelente
trabalho
pessoa
bem
profissional
melhor
aveiro
tudo
experiência
qualidade
espaço
todo
simpatia
profissionais
dia
fazer
boa
incrível
equipa
adorei
ótimo
local
tempo
resultado
clínica
ainda
produto
simpático
extremamente
nunca
obrigado
ano
fiz
nada
toda
além
casa
onde
grande
profissionalismo
forma
ambiente
tratamento
atencioso
funcionário
super
pouco
parabéns
bastante
bom
cuidado
muita
desde
apena
pois
aqui
outro
outra
atenção
obrigada
preço
bom atendimento
excelente atendimento
recomendo vivamente
excelente profissional
agradável
problema
cabelo
café
disse
dúvida
empresa
confiança
consulta
simpática
assim
senhora
falta
mal
vez
tão
lugar
roupa
faz
conhecimento
melhores
hora
porque
estrela
maravilhosa
pessoal
ter
hoje
parte
tanto
qualquer

6. Save Artifacts & Pipeline Summary

Purpose: Persist all pipeline outputs to disk and generate a summary report for reproducibility and downstream analysis.

Output Files Generated

File	Location	Contains	Purpose
reviews_raw.csv	`../output/`	All extracted reviews with: place_id, place_name, rating, review_text (original), publish_time, lat, lon, place_rating, place_primary_type, poi metadata (gid, amenity, etc.)	Raw API output; enables replay/replay debugging
reviews_clean.csv	`../output/`	raw + cleaned: review_text_clean, detected lang, tokens list, token_count, text_processed (stopwords removed, normalized)	Intermediate format; allows text inspection and validation
reviews_enriched.csv	`../output/`	clean + features: tf_idf features, `sentiment_compound` (EN via VADER; PT via HuggingFace bertweet-pt-sentiment or TextBlob fallback)	Final analytical dataset; ready for downstream modeling/analysis

Downstream Usage

- **reviews_raw.csv**: Audit API responses; validate geometry/location parsing
- **reviews_clean.csv**: EDA, text quality checks, language distribution analysis
- **reviews_enriched.csv**: Train sentiment models, topic analysis

In [12]:
```
summary = {
    "raw_reviews_csv": RAW_REVIEWS_CSV if os.path.exists(RAW_REVIEWS_CSV)
    "clean_reviews_csv": CLEAN_REVIEWS_CSV if os.path.exists(CLEAN_REVIEW
    "enriched_reviews_csv": os.path.join(OUTPUT_DIR, "reviews_enriched.cs
    "n_reviews_raw": int(pd.read_csv(RAW_REVIEWS_CSV).shape[0]) if os.pat
    "n_reviews_clean": int(pd.read_csv(CLEAN_REVIEWS_CSV).shape[0]) if os
}
print(json.dumps(summary, indent=2))
```


```
{
  "raw_reviews_csv": "../output/reviews_raw.csv",
  "clean_reviews_csv": "../output/reviews_clean.csv",
  "enriched_reviews_csv": "../output/reviews_enriched.csv",
  "n_reviews_raw": 6800,
  "n_reviews_clean": 5298
}
```


7. Topic Analysis (LDA)

Purpose: Discover latent themes and topics within review texts using Latent Dirichlet Allocation (LDA), enabling automatic identification of recurring discussion themes across different languages.

Process

1. **Prepare Corpus**:
   - Split reviews by language (English and Portuguese separately)
   - Use `text_processed` (stopword-removed, normalized text) as input
   - Build dictionary of unique tokens per language
2. **LDA Model Training**:
   - Fit separate LDA models for English and Portuguese reviews
   - Number of topics: 5 (configurable)
   - LDA hyperparameters:
     - `passes=10` : Number of iterations over corpus
     - `workers=4` : Parallel processing threads
     - `per_word_topics=True` : Detailed per-word topic assignments
3. **Topic Extraction**:
   - Extract top N words per topic (highest probability)
   - Compute topic distribution per document (review)
   - Identify dominant topic for each review
4. **Visualization & Analysis**:
   - Display top words per topic with probabilities
   - Show topic proportions across dataset
   - Create visualizations of topic coherence and prevalence

Key Outputs

- **Per-language LDA models**: Trained topic models for EN and PT
- **Topic interpretations**: Most probable words defining each topic
- **Document-topic mappings**: Which topics appear in each review
- **Topic statistics**: Prevalence and coherence metrics

In [ ]:
```
if not df_enriched.empty and HAVE_GENSIM:
    print("=== Topic Analysis (LDA) ===\n")

    # Configuration
    NUM_TOPICS = 5
    PASSES = 10
    WORKERS = 4
    MIN_WORD_FREQ = 2

    def train_lda_model(df_lang, lang_name, num_topics=NUM_TOPICS):
        """Train LDA model for a given language."""
        if df_lang.empty or len(df_lang) < 5:
            print(f"Insufficient {lang_name} reviews ({len(df_lang)}) for
            return None, None

        # Convert text_processed to token lists
        texts = [text.split() for text in df_lang["text_processed"].filln
        texts = [[token for token in text if token] for text in texts]  #

        if not texts or all(len(t) == 0 for t in texts):
```

```
            print(f"No tokens available for {lang_name}.")
            return None, None

        # Build dictionary and corpus
        dictionary = corpora.Dictionary(texts)
        dictionary.filter_extremes(no_below=MIN_WORD_FREQ, no_above=0.7,
        corpus = [dictionary.doc2bow(text) for text in texts]

        if not corpus:
            print(f"Empty corpus for {lang_name}.")
            return None, None

        # Train LDA model
        print(f"Training LDA model for {lang_name} ({len(texts)} reviews)
        try:
            lda_model = LdaMulticore(
                corpus=corpus,
                id2word=dictionary,
                num_topics=num_topics,
                random_state=42,
                passes=PASSES,
                workers=WORKERS,
                per_word_topics=True,
                minimum_probability=0.0,
            )
            print(f"✓ {lang_name} LDA model trained successfully.\n")
            return lda_model, dictionary
        except Exception as e:
            print(f"✗ Error training {lang_name} LDA model: {e}\n")
            return None, None

    # Train models for each language
    lda_en, dict_en = train_lda_model(df_en, "English", NUM_TOPICS)
    lda_pt, dict_pt = train_lda_model(df_pt, "Portuguese", NUM_TOPICS)

    # Display topics
    def display_topics(lda_model, lang_name):
        """Print top words for each topic."""
        if lda_model is None:
            print(f"No {lang_name} LDA model available.")
            return

        print(f"\n{'='*60}")
        print(f"  {lang_name.upper()} - TOP TOPICS")
        print(f"{'='*60}")
        topics = lda_model.print_topics(num_words=10)
        for topic_id, topic_words in topics:
            print(f"\nTopic {topic_id}:")
            # Parse and display words with weights
            words_weights = [item.split('*') for item in topic_words.spli
            for weight, word in words_weights:
                word_clean = word.strip('"')
                print(f"  {word_clean:20s}  {float(weight):.4f}")

    # Compute dominant topics for each review
    def get_dominant_topics(df_lang, lda_model, dictionary, lang_name):
        """Extract dominant topic per review."""
        if lda_model is None or df_lang.empty:
            return pd.DataFrame()
```

```
    texts = [text.split() for text in df_lang["text_processed"].filln
    texts = [[token for token in text if token] for text in texts]

    dominant_topics = []
    for idx, text in enumerate(texts):
        bow = dictionary.doc2bow(text)
        if bow:
            topics = lda_model.get_document_topics(bow)
            if topics:
                dominant_topic = max(topics, key=lambda x: x[1])
                dominant_topics.append({
                    "topic_id": dominant_topic[0],
                    "topic_prob": dominant_topic[1],
                })
            else:
                dominant_topics.append({"topic_id": None, "topic_prob
        else:
            dominant_topics.append({"topic_id": None, "topic_prob": 0

    return pd.DataFrame(dominant_topics)

# Visualize topic distributions
def plot_topic_distribution(df_topics, lang_name):
    """Plot topic prevalence."""
    if df_topics.empty or df_topics["topic_id"].isna().all():
        print(f"No topics to plot for {lang_name}.")
        return

    topic_counts = df_topics["topic_id"].value_counts().sort_index()
    plt.figure(figsize=(9, 5))
    plt.bar(topic_counts.index, topic_counts.values, color="mediumpur
    plt.title(f"Topic Distribution ({lang_name})")
    plt.xlabel("Topic ID")
    plt.ylabel("Number of Reviews")
    plt.xticks(topic_counts.index)
    plt.tight_layout()
    plt.show()

# === Display English Topics ===
display_topics(lda_en, "English")

# === Display Portuguese Topics ===
display_topics(lda_pt, "Portuguese")

# === Get and display dominant topics for English ===
if lda_en is not None:
    topics_en = get_dominant_topics(df_en, lda_en, dict_en, "English"
    if not topics_en.empty:
        print(f"\nEnglish Topic Distribution:")
        print(topics_en["topic_id"].value_counts().sort_index())
        plot_topic_distribution(topics_en, "English")

# === Get and display dominant topics for Portuguese ===
if lda_pt is not None:
    topics_pt = get_dominant_topics(df_pt, lda_pt, dict_pt, "Portugue
    if not topics_pt.empty:
        print(f"\nPortuguese Topic Distribution:")
        print(topics_pt["topic_id"].value_counts().sort_index())
        plot_topic_distribution(topics_pt, "Portuguese")
```

```
elif not df_enriched.empty and not HAVE_GENSIM:
    print("Topic Analysis skipped: gensim not installed. Install with: pi
else:
    print("No enriched data available for topic analysis.")
```

```
=== Topic Analysis (LDA) ===

Training LDA model for English (2492 reviews)...
✓ English LDA model trained successfully.

Training LDA model for Portuguese (2805 reviews)...
✓ English LDA model trained successfully.

Training LDA model for Portuguese (2805 reviews)...
✓ Portuguese LDA model trained successfully.


============================================================
  ENGLISH - TOP TOPICS
============================================================

Topic 0:
  food                  0.0290
  great                 0.0220
  service               0.0220
  place                 0.0160
  experience            0.0140
  friendly              0.0140
  staff                 0.0140
  amazing               0.0140
  good                  0.0110
  pizza                 0.0100

Topic 1:
  best                  0.0310
  place                 0.0280
  aveiro                0.0260
  top                   0.0240
  time                  0.0110
  one                   0.0110
  visit                 0.0110
  love                  0.0100
  portuguese            0.0100
  portugal              0.0090

Topic 2:
  good                  0.0440
  food                  0.0250
  service               0.0240
  great                 0.0240
  nice                  0.0220
  place                 0.0210
  price                 0.0200
  staff                 0.0180
  friendly              0.0170
  delicious             0.0130

Topic 3:
  service               0.0170
  one                   0.0120
  time                  0.0120
  recommend             0.0110
  even                  0.0100
  professional          0.0090
  experience            0.0080
```

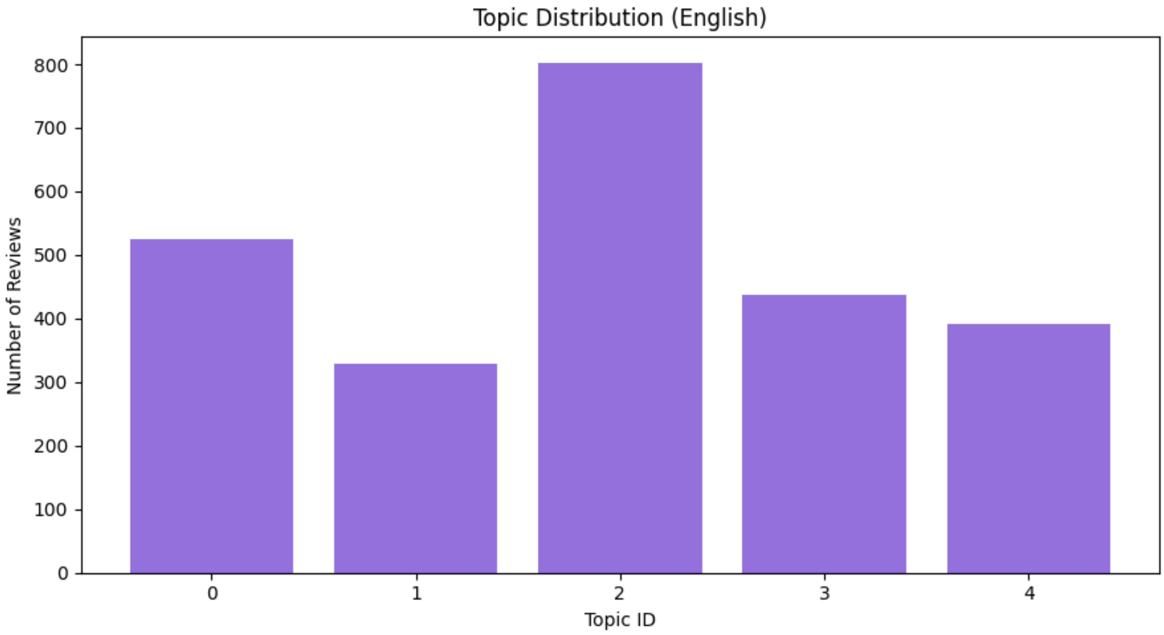
```
  good                 0.0080
  minute               0.0080
  like                 0.0080

Topic 4:
  room                 0.0210
  good                 0.0160
  clean                0.0150
  location             0.0140
  nice                 0.0120
  place                0.0120
  aveiro               0.0110
  well                 0.0110
  great                0.0100
  breakfast            0.0100

============================================================
  PORTUGUESE - TOP TOPICS
============================================================

Topic 0:
  faz                  0.0130
  tod                  0.0120
  dia                  0.0120
  vez                  0.0090
  porqu                0.0090
  outr                 0.0090
  atend                0.0080
  ped                  0.0080
  pod                  0.0070
  diss                 0.0070

Topic 1:
  tod                  0.0160
  compr                0.0120
  aveir                0.0110
  cas                  0.0100
  melhor               0.0090
  dia                  0.0090
  bem                  0.0090
  sempr                0.0090
  client               0.0080
  faz                  0.0070

Topic 2:
  atend                0.0530
  bom                  0.0390
  excel                0.0360
  recom                0.0250
  qual                 0.0230
  serviç               0.0230
  preç                 0.0200
  simpá                0.0190
  boa                  0.0170
  espaç                0.0150

Topic 3:
  atend                0.0260
  client               0.0140
  pesso                0.0130
```

```
  faz                 0.0130
  hor                 0.0100
  ter                 0.0100
  empr                0.0090
  aind                0.0090
  loj                 0.0080
  cas                 0.0080

Topic 4:
  profiss             0.0390
  recom               0.0280
  excel               0.0250
  sup                 0.0210
  atend               0.0200
  sempr               0.0190
  tod                 0.0160
  trabalh             0.0130
  atenci              0.0120
  melhor              0.0120
✓ Portuguese LDA model trained successfully.


============================================================
  ENGLISH - TOP TOPICS
============================================================

Topic 0:
  food                0.0290
  great               0.0220
  service             0.0220
  place               0.0160
  experience          0.0140
  friendly            0.0140
  staff               0.0140
  amazing             0.0140
  good                0.0110
  pizza               0.0100

Topic 1:
  best                0.0310
  place               0.0280
  aveiro              0.0260
  top                 0.0240
  time                0.0110
  one                 0.0110
  visit               0.0110
  love                0.0100
  portuguese          0.0100
  portugal            0.0090

Topic 2:
  good                0.0440
  food                0.0250
  service             0.0240
  great               0.0240
  nice                0.0220
  place               0.0210
  price               0.0200
  staff               0.0180
  friendly            0.0170
```

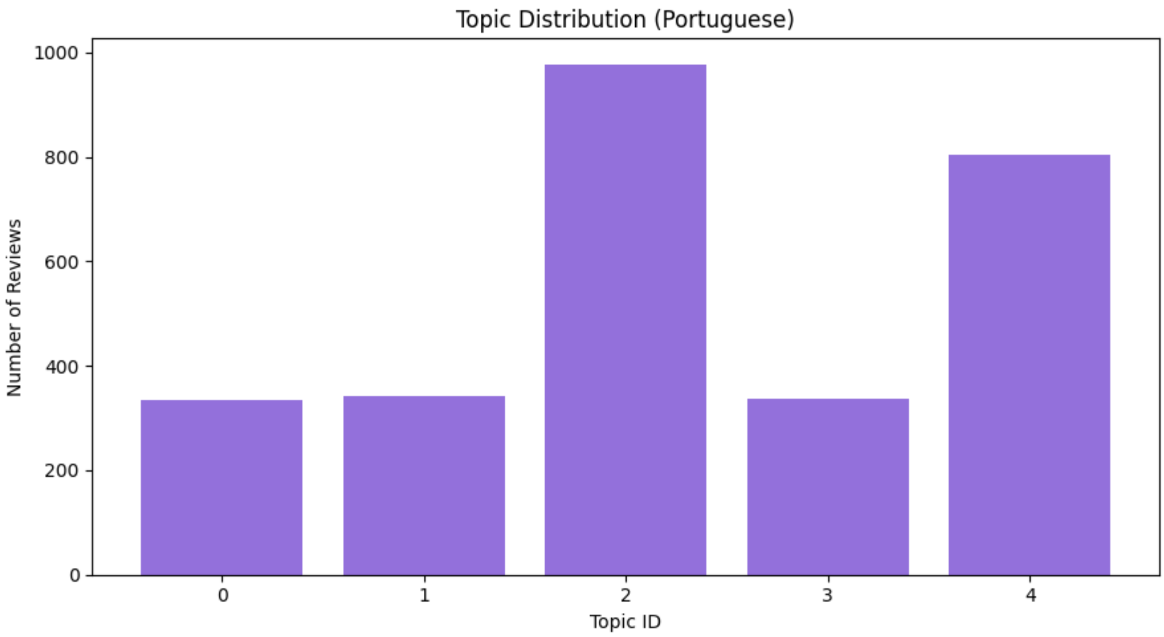
```
  delicious            0.0130

Topic 3:
  service              0.0170
  one                  0.0120
  time                 0.0120
  recommend            0.0110
  even                 0.0100
  professional         0.0090
  experience           0.0080
  good                 0.0080
  minute               0.0080
  like                 0.0080

Topic 4:
  room                 0.0210
  good                 0.0160
  clean                0.0150
  location             0.0140
  nice                 0.0120
  place                0.0120
  aveiro               0.0110
  well                 0.0110
  great                0.0100
  breakfast            0.0100

============================================================
  PORTUGUESE - TOP TOPICS
============================================================

Topic 0:
  faz                  0.0130
  tod                  0.0120
  dia                  0.0120
  vez                  0.0090
  porqu                0.0090
  outr                 0.0090
  atend                0.0080
  ped                  0.0080
  pod                  0.0070
  diss                 0.0070

Topic 1:
  tod                  0.0160
  compr                0.0120
  aveir                0.0110
  cas                  0.0100
  melhor               0.0090
  dia                  0.0090
  bem                  0.0090
  sempr                0.0090
  client               0.0080
  faz                  0.0070

Topic 2:
  atend                0.0530
  bom                  0.0390
  excel                0.0360
  recom                0.0250
  qual                 0.0230
```

```
  serviç              0.0230
  preç                0.0200
  simpá               0.0190
  boa                 0.0170
  espaç               0.0150

Topic 3:
  atend               0.0260
  client              0.0140
  pesso               0.0130
  faz                 0.0130
  hor                 0.0100
  ter                 0.0100
  empr                0.0090
  aind                0.0090
  loj                 0.0080
  cas                 0.0080

Topic 4:
  profiss             0.0390
  recom               0.0280
  excel               0.0250
  sup                 0.0210
  atend               0.0200
  sempr               0.0190
  tod                 0.0160
  trabalh             0.0130
  atenci              0.0120
  melhor              0.0120

English Topic Distribution:
topic_id
0.0    524
1.0    329
2.0    803
3.0    438
4.0    391
Name: count, dtype: int64
```


```
English Topic Distribution:
topic_id
0.0    524
1.0    329
2.0    803
3.0    438
4.0    391
Name: count, dtype: int64
```




```
Portuguese Topic Distribution:
topic_id
0.0    335
1.0    341
2.0    978
3.0    336
4.0    805
Name: count, dtype: int64
```




```
English model topics: 5
Portuguese model topics: 5
```