

Assessing the Contribution of Ecotourism and Community-Based Ecotourism to Community Resilience in the Pamirs

Furough Shakarmamadova, Philip E. Van Beynen, and Fenda A. Akiwumi

Here is the python script created by Co-Pilot Enterprise® coupled with GPT-5® to produce an Interpretative Phenomenological Analysis (IPA) framework matrix. A WORD document containing 53 informant responses (bullets) to seven questions was attached to the Co-Pilot Enterprise® request to build the IPA framework matrix to determine how the following five themes/outcomes were recognized by the informants in their answers. These themes/ outcomes are community stability, economic stability, environment conservation, cultural conservation and empowerment.

The command was “create an Interpretative Phenomenological Analysis (IPA) framework matrix from the attached document containing seven questions looking for following themes: community stability, economic stability, environment conservation, cultural conservation and empowerment”.

Co-Pilot responded by asking how I wanted the analysis to be reported. I requested the matrix table and heatmap.

1. Methods note (how this IPA-style matrix was built)

- **Idiographic reading:** I read each bullet* as an “informant unit,” noting how participants **make sense** of ecotourism in their context (experience-near phrasing preserved in extracts). [[all interv...ecotourism | Word](#)]
- **Framework matrixing:** I then **indexed** each bullet to the five requested themes (allowing multiple codes per bullet) and summarized patterns by **question** to retain situational context (definition, examples, impacts, future, livelihoods, awareness). [[all interv...ecotourism | Word](#)]
- **Quality checks:** Automated keyword screening supported coverage and counting, followed by manual trimming/interpretation to reduce false positives and select representative extracts. (53 bullets total; see distribution above.)

* Bullet is the informants response.

2. Analysis Code – IPA Framework Matrix (Themes × Evidence × Interpretation)

Generated on 2025-10-31 14:01:20

This document contains a complete, ready-to-run Python analysis script that ingests the interview bullets, codes them to the five themes, produces counts and representative quotes, builds an IPA-style framework matrix, and optionally exports a Word report and heatmap. Paste your own data path into INPUT_DOCX or RAW_TEXT.

```
# -*- coding: utf-8 -*-  
"""
```

```
IPA Framework Matrix - End-to-end Analysis Script  
=====
```

```
This script reproduces the analysis pipeline used to build an IPA-style  
Framework Matrix (Themes x Evidence x Interpretation) from interview bullets.
```

It performs:

- 1) Ingestion of interview text (from .docx or raw string)
- 2) Parsing into Questions (Q1-Q7) and bullet points
- 3) Thematic coding using a simple keyword dictionary (customizable)
- 4) Aggregation of counts by theme and by question
- 5) Selection of representative quotes (per theme)
- 6) Assembly of an IPA Framework Matrix (themes x evidence x interpretation)
- 7) Optional visualization (heatmap) and exports (CSV/PNG)
- 8) Optional report export to Word (.docx)

Prerequisites (for a fresh environment):

```
pip install python-docx pandas matplotlib
```

Notes:

- The keyword approach is intentionally light-touch and should be combined with manual review to align with interpretative phenomenological analysis (IPA).
- Replace INPUT_DOCX or RAW_TEXT with your own data. The parser expects sections labeled as ****Question X. ...**** followed by '-' bullets.

```
"""
```

```
from __future__ import annotations  
import re  
from collections import defaultdict  
from dataclasses import dataclass  
from typing import List, Dict, Tuple
```

```
import pandas as pd  
import matplotlib.pyplot as plt
```

```
# ===== CONFIGURATION =====
```

```
# Path to a .docx containing the interview bullets (optional). If None, falls back to  
RAW_TEXT.
```

```
INPUT_DOCX: str | None = None # e.g., 'all interview answers ecotourism.docx'
```

```
# If you prefer, paste your text here using the same structure as the uploaded file.
```

```
RAW_TEXT: str | None = None
```

```
# Theme keywords (customize as needed)
```

```
THEME_KEYWORDS: Dict[str, List[str]] = {
```

```
    'community_stability': [
```

```
        'community', 'communities', 'local people', 'locals', 'bring more people
```

```

together',
    'association', 'stakeholders', 'awareness', 'education', 'schools', 'soldiers',
    'check point', 'checkpoint', 'awareness raising', 'security', 'stable',
'stability',
    'cross border', 'relationship', 'together', 'members'
],
'economic_stability': [
    'money', 'income', 'benefit', 'sustain their lives', 'jobs', 'seasonal',
    'permanent jobs', 'business', 'export', 'drivers', 'homestays', 'hotels',
    'handicraft', 'financial', 'fundraising', 'create jobs', 'economical',
    'tourism industry', 'market'
],
'environmental_conservation': [
    'environment', 'nature', 'protect', 'protection', 'pollution', 'garbage',
'plastics',
    'green', 'renewable', 'solar', 'wildlife', 'parks', 'national park',
'ecological',
    'deforestation', 'teresk', 'teresken', 'clean', 'clean up', 'minimal impact',
'flora',
    'fauna', 'glaciers', 'rivers', 'lakes', 'trees'
],
'cultural_conservation': [
    'culture', 'cultural', 'historical', 'tradition', 'handicraft', 'local wools',
    'eco cultural', 'eco-cultural'
],
'empowerment': [
    'awareness', 'training', 'teach', 'teaching', 'involved', 'involve',
'volunteer',
    'capacity', 'responsibilities', 'association', 'members', 'information
centers',
    'help communities', 'priority that people benefit', 'explain', 'education
campaign'
],
}

```

```

# Representative quotes per theme to keep
N_REP_QUOTES = 5

```

```

# Output paths
HEATMAP_PNG = 'theme_salience_heatmap.png'
MATRIX_CSV = 'theme_salience_matrix.csv'
REPORT_DOCX = 'ipa_framework_report.docx' # optional report (table + figure)

```

```

# ===== DATA CLASSES =====
@dataclass
class Entry:
    qnum: str
    question: str
    text: str
    themes: List[str]

```

```

# ===== INGESTION HELPERS =====

```

```

def load_text_from_docx(path: str) -> str:
    """Load paragraph text from a .docx file into a single string."""
    from docx import Document
    doc = Document(path)
    parts = []
    for p in doc.paragraphs:
        parts.append(p.text)
    return "\n".join(parts)

# ===== PARSING =====

def parse_questions_and_bullets(doc_text: str) -> Dict[str, Dict[str, List[str]]]:
    """Parse the raw text into a dict mapping qnum -> {question, bullets[]}.
    Expects sections like **Question 1. Define Ecotourism** followed by '-' bullets.
    """
    sections = re.split(r"\*\*Question\s*(\d+)\.?(.*?)\*\*", doc_text, flags=re.S)
    questions: Dict[str, Dict[str, List[str]]] = {}
    for i in range(1, len(sections), 3):
        qnum = sections[i].strip()
        qtext = sections[i+1].strip().replace('\n', ' ').strip()
        content = sections[i+2]
        bullets = [b.strip() for b in re.findall(r"^\s+(.*)", content, flags=re.M)]
        questions[qnum] = { 'question': qtext, 'bullets': bullets }
    return questions

# ===== THEMATIC CODING =====

def code_themes_for_text(text: str, theme_keywords: Dict[str, List[str]]) -> List[str]:
    t = text.lower()
    tags = []
    for theme, kws in theme_keywords.items():
        if any(kw in t for kw in kws):
            tags.append(theme)
    return sorted(set(tags))

def code_entries(questions: Dict[str, Dict[str, List[str]]],
                 theme_keywords: Dict[str, List[str]]) -> List[Entry]:
    entries: List[Entry] = []
    for qnum, data in questions.items():
        for bullet in data['bullets']:
            themes = code_themes_for_text(bullet, theme_keywords)
            entries.append(Entry(qnum=qnum, question=data['question'], text=bullet,
            themes=themes))
    return entries

# ===== AGGREGATION =====

def aggregate_counts(entries: List[Entry]) -> Tuple[Dict[str, int], Dict[str, Dict[str,
int]]]:
    total_by_theme: Dict[str, int] = defaultdict(int)
    by_q_theme: Dict[str, Dict[str, int]] = defaultdict(lambda: defaultdict(int))
    for e in entries:

```

```

        for t in e.themes:
            total_by_theme[t] += 1
            by_q_theme[e.qnum][t] += 1
    return total_by_theme, by_q_theme

# ===== REPRESENTATIVE QUOTES =====

def select_representative_quotes(entries: List[Entry], n_per_theme: int = 5) ->
Dict[str, List[Entry]]:
    rep: Dict[str, List[Entry]] = defaultdict(list)
    # prefer diversity across questions first
    for e in entries:
        for t in e.themes:
            if len(rep[t]) < n_per_theme:
                # favor unique question coverage
                covered_qs = {x.qnum for x in rep[t]}
                if e.qnum not in covered_qs:
                    rep[t].append(e)
    # if still fewer than n, fill regardless of question diversity
    for t in list(rep.keys()) | {t for e in entries for t in e.themes}:
        if len(rep[t]) < n_per_theme:
            for e in entries:
                if t in e.themes and e not in rep[t]:
                    rep[t].append(e)
                    if len(rep[t]) >= n_per_theme:
                        break
    return rep

# ===== FRAMEWORK MATRIX (IPA-style) =====

def build_framework_matrix(rep_quotes: Dict[str, List[Entry]]) -> pd.DataFrame:
    rows = []
    for theme, items in rep_quotes.items():
        evidence_lines = [f"• Q{e.qnum}: {e.text}" for e in items]
        evidence = "\n".join(evidence_lines)
        interpretation = (
            "Interpretative memo placeholder – summarize how participants make sense of"
            "this theme, "
            "linking to context (e.g., stability, infrastructure) and tensions (e.g.,"
            "growth vs. impact). "
        )
        rows.append({
            'Theme': theme.replace('_', ' ').title(),
            'Evidence (illustrative extracts)': evidence,
            'Interpretation (notes)': interpretation
        })
    df = pd.DataFrame(rows, columns=['Theme', 'Evidence (illustrative'
                                     'extracts)', 'Interpretation (notes)'])
    return df

# ===== VISUALIZATION =====

def heatmap_counts(by_q_theme: Dict[str, Dict[str, int]]),

```

```

        theme_order: List[str],
        out_png: str = HEATMAP_PNG,
        cmap: str = 'YlGnBu') -> pd.DataFrame:
    qs = [str(i) for i in range(1, 1+max(map(int, by_q_theme.keys() or ['1'])))]
    mat = []
    for q in qs:
        mat.append([by_q_theme[q].get(t, 0) for t in theme_order])
    df = pd.DataFrame(mat, index=[f"Q{q}" for q in qs], columns=[t.replace('_', '
').title() for t in theme_order])

    fig, ax = plt.subplots(figsize=(10, 5.5))
    im = ax.imshow(df.values, cmap=cmap)
    ax.set_xticks(range(len(df.columns)))
    ax.set_yticks(range(len(df.index)))
    ax.set_xticklabels(df.columns, rotation=30, ha='right')
    ax.set_yticklabels(df.index)
    for i in range(df.shape[0]):
        for j in range(df.shape[1]):
            ax.text(j, i, str(df.iat[i,j]), ha='center', va='center', color='black',
fontsize=9)
    ax.set_title('Theme Salience by Question (counts of coded mentions)')
    fig.colorbar(im, ax=ax, label='Count')
    fig.tight_layout()
    fig.savefig(out_png, dpi=200)
    plt.close(fig)
    return df

# ===== WORD EXPORT (OPTIONAL) =====

def export_to_word(matrix_df: pd.DataFrame, heatmap_png: str | None = None,
                    out_docx: str = REPORT_DOCX,
                    counts_total: Dict[str,int] | None = None,
                    by_q_theme: Dict[str, Dict[str, int]] | None = None) -> str:
    from docx import Document
    from docx.shared import Inches

    doc = Document()
    doc.add_heading('IPA Framework Matrix (Themes x Evidence x Interpretation)',
level=1)
    p = doc.add_paragraph('Auto-generated from coded interview bullets. Evidence are
illustrative extracts; interpretation notes are placeholders to be refined during
analysis.')

    # Add counts summary
    if counts_total:
        doc.add_heading('Theme counts (total mentions)', level=2)
        t = doc.add_table(rows=1, cols=2)
        t.style = 'Table Grid'
        hdr = t.rows[0].cells
        hdr[0].text = 'Theme'
        hdr[1].text = 'Count'
        for theme, ct in sorted(counts_total.items(), key=lambda x: -x[1]):
            r = t.add_row().cells

```

```

        r[0].text = theme.replace('_', ' ').title()
        r[1].text = str(ct)

# Add matrix table
doc.add_heading('Framework Matrix', level=2)
tbl = doc.add_table(rows=1, cols=3)
tbl.style = 'Table Grid'
hdr = tbl.rows[0].cells
hdr[0].text = 'Theme'
hdr[1].text = 'What participants said (illustrative extracts)'
hdr[2].text = 'Interpretative notes'
for _, row in matrix_df.iterrows():
    r = tbl.add_row().cells
    r[0].text = str(row['Theme'])
    r[1].text = str(row['Evidence (illustrative extracts)'])
    r[2].text = str(row['Interpretation (notes)'])

# Add heatmap figure if available
if heatmap_png:
    doc.add_page_break()
    doc.add_heading('Theme Salience Heatmap', level=2)
    try:
        doc.add_picture(heatmap_png, width=Inches(6.5))
    except Exception:
        doc.add_paragraph(f"(Could not embed heatmap image: {heatmap_png})")

doc.save(out_docx)
return out_docx

# ===== MAIN PIPELINE =====

def main():
    # 1) Ingest
    if INPUT_DOCX:
        text = load_text_from_docx(INPUT_DOCX)
    elif RAW_TEXT:
        text = RAW_TEXT
    else:
        raise SystemExit("Please set INPUT_DOCX to a .docx path or provide RAW_TEXT.")

    # 2) Parse
    questions = parse_questions_and_bullets(text)

    # 3) Thematic coding
    entries = code_entries(questions, THEME_KEYWORDS)

    # 4) Aggregation
    total_by_theme, by_q_theme = aggregate_counts(entries)

    # 5) Representative quotes
    rep_quotes = select_representative_quotes(entries, n_per_theme=N_REP_QUOTES)

    # 6) Framework Matrix

```

```

matrix_df = build_framework_matrix(rep_quotes)

# 7) Visualization (optional)
theme_order = list(THEME_KEYWORDS.keys())
heatmap_df = heatmap_counts(by_q_theme, theme_order, out_png=HEATMAP_PNG)
heatmap_df.to_csv(MATRIX_CSV)

# 8) Export to Word (optional)
out_docx = export_to_word(matrix_df, heatmap_png=HEATMAP_PNG,
                           out_docx=REPORT_DOCX,
                           counts_total=total_by_theme,
                           by_q_theme=by_q_theme)

print("Entries:", len(entries))
print("Total by theme:", dict(total_by_theme))
print("Saved heatmap:", HEATMAP_PNG)
print("Saved matrix CSV:", MATRIX_CSV)
print("Saved report DOCX:", out_docx)

if __name__ == '__main__':
    main()

```