# Ecommerce Campaign Analysis with RAG - Cloud MVP

## 🎯 Project Overview

Build an AI-powered analysis tool for ecommerce campaign data that can analyze Excel/CSV files from platforms like Amazon, Flipkart, Blinkit, Instamart, and Zepto. The system uses Retrieval-Augmented Generation (RAG) to provide intelligent insights based on historical analysis patterns and ecommerce best practices.

### Key Features

- **Excel/CSV Upload & Analysis**: Process campaign data from major ecommerce platforms
- **Natural Language Queries**: Ask questions in plain English about your data
- **Intelligent Context**: RAG provides analysis methodologies and platform-specific insights
- **Flexible Goals**: Each team can set their own KPI targets per session
- **100% Cloud-Based**: No local installations needed, works entirely in browser

## 🌐 Updated Architecture (Cloud-Based)

### Technology Stack

- **Development Environment**: GitHub Codespaces (60 hours free/month)
- **Frontend**: Streamlit (Python web framework)
- **Vector Database**: ChromaDB (runs in cloud environment)
- **LLM**: Ollama + Llama 3.1 (runs in Codespaces)
- **Data Processing**: Pandas (Excel/CSV handling)
- **RAG Framework**: LangChain (orchestration)
- **Embeddings**: Local sentence-transformers

### System Flow

```
Excel Upload → Data Processing → Knowledge Base Search →
Context Retrieval → LLM Analysis → Structured Response
```

## 🚀 Getting Started (Zero Installation Method)

### Prerequisites

- GitHub account (free)
- Modern web browser
- Internet connection

## Setup Steps

### Step 1: Create Your Development Environment (Day 1)

1. **Create a new GitHub repository**
   - Go to github.com
   - Click "New repository"
   - Name it: `ecommerce-rag-analyzer`
   - Make it public or private
   - Initialize with README
2. **Open in GitHub Codespaces**
   - In your new repo, click the green "Code" button
   - Click "Codespaces" tab
   - Click "Create codespace on main"
   - Wait for environment to load (2-3 minutes)

### Step 2: Install Dependencies (Day 1)

Once your Codespace is ready, run these commands in the terminal:

```bash
# Install Ollama
curl -fsSL https://ollama.ai/install.sh | sh

# Start Ollama service
ollama serve &

# Pull the AI model (this takes 5-10 minutes)
ollama pull llama3.1:8b

# Install Python dependencies
pip install streamlit pandas openpyxl xlrd plotly chromadb langchain sentence-transformers
```

### Step 3: Clone and Setup Base Project (Day 1-2)

```bash
```

```
# Clone the ragbase foundation
git clone https://github.com/curiousily/ragbase.git ragbase-temp
cp -r ragbase-temp/* .
rm -rf ragbase-temp

# Verify everything works
python -c "import streamlit, pandas, chromadb; print('All dependencies installed!')"
```

## 📋 Updated Project Plan (4 Weeks)

### Week 1: Foundation Setup in Cloud

**Goal**: Get ragbase working in Codespaces and replace PDF with Excel processing

**Tasks**:

- [x] Set up GitHub Codespaces environment
- [x] Install Ollama and dependencies in cloud
- [ ] Test original ragbase with PDFs
- [ ] Replace PDF document loaders with Excel/CSV loaders
- [ ] Modify data preprocessing pipeline for tabular data
- [ ] Test basic Excel upload and processing
- [ ] Verify end-to-end flow (upload → process → query → response)

**Deliverable**: Working system in Codespaces that can upload Excel files and answer basic questions

### Week 2: Excel Integration & Data Processing

**Goal**: Optimize data handling and create proper document structure for RAG

**Tasks**:

- [ ] Implement intelligent data summarization
- [ ] Create proper document chunking for tabular data
- [ ] Add data validation and error handling
- [ ] Implement multiple file format support (XLSX, CSV, XLS)
- [ ] Create data preview and basic statistics
- [ ] Optimize memory usage for large datasets (within Codespaces limits)

**Deliverable**: Robust Excel processing with data insights

### Week 3: Knowledge Base Creation

**Goal**: Build ecommerce-specific knowledge base for intelligent analysis

**Tasks**:

- [ ] Create analysis methodology library
- [ ] Add platform-specific insights (Amazon, Flipkart, etc.)
- [ ] Build query pattern recognition
- [ ] Implement dynamic goal setting per session
- [ ] Create ecommerce analysis templates
- [ ] Test context retrieval accuracy

**Deliverable**: Smart analysis with business context

### Week 4: Polish & Deployment

**Goal**: Production-ready system with deployment options

**Tasks**:

- [ ] Improve UI/UX with better visualizations
- [ ] Add export functionality for analysis reports
- [ ] Implement analysis history and comparison
- [ ] Optimize for Codespaces performance
- [ ] Create user documentation and guides
- [ ] Deploy using Streamlit Cloud or similar free service

**Deliverable**: Deployed MVP accessible via web link

## 📁 Project Structure

```
ecommerce-rag-analyzer/
├── .devcontainer/          # Codespaces configuration
│   └── devcontainer.json     # Environment setup
├── app.py                  # Main Streamlit application
├── data_processing/
│   ├── excel_loader.py       # Excel/CSV processing
│   ├── data_summarizer.py     # Data analysis and summaries
│   └── validators.py         # Data validation
├── rag_engine/
│   ├── knowledge_base.py      # RAG implementation
│   ├── vector_store.py        # ChromaDB operations
│   └── retrieval.py          # Context retrieval logic
├── knowledge_base/
│   ├── analysis_methods.json   # Analysis methodologies
│   ├── platform_insights.json  # Platform-specific knowledge
│   └── query_patterns.json    # Common question patterns
├── prompts/
│   ├── analysis_prompts.py     # LLM prompt templates
│   └── system_prompts.py      # System instructions
├── utils/
│   ├── helpers.py            # Utility functions
│   └── config.py             # Configuration settings
├── requirements.txt          # Python dependencies
└── README.md                 # Project documentation
```

## 🔧 Technical Implementation

### Excel Processing Module

```
python
```

```
# data_processing/excel_loader.py
import pandas as pd
from langchain.schema import Document

class ExcelProcessor:
    def load_excel_as_documents(self, file_path):
        df = pd.read_excel(file_path)
        documents = []

        # Create summary document
        summary = self.create_data_summary(df)
        documents.append(Document(
            page_content=summary,
            metadata={"type": "summary", "source": file_path}
        ))

        # Create row documents for RAG
        for index, row in df.iterrows():
            content = self.row_to_text(row, index)
            documents.append(Document(
                page_content=content,
                metadata={"type": "row", "index": index, "source": file_path}
            ))

        return documents, df

    def create_data_summary(self, df):
        summary = f"""
        Dataset Summary:
        - Total Rows: {len(df)}
        - Total Columns: {len(df.columns)}
        - Columns: {', '.join(df.columns.tolist())}
        - Date Range: {self.get_date_range(df)}
        - Key Metrics: {self.identify_key_metrics(df)}
        """
        return summary
```

## Cloud-Optimized RAG Implementation

```
python
```

```python
# rag_engine/knowledge_base.py
import chromadb
from langchain.llms import Ollama

class EcommerceRAG:
    def __init__(self):
        # Initialize ChromaDB (works in Codespaces)
        self.chroma_client = chromadb.Client()
        self.collection = self.chroma_client.create_collection("ecommerce_data")

        # Initialize Ollama (running in Codespaces)
        self.llm = Ollama(model="llama3.1:8b", base_url="http://localhost:11434")

        self.load_knowledge_base()

    def analyze_with_context(self, df, question, user_goals=None):
        # Detect analysis type
        analysis_type = self.classify_question(question)

        # Retrieve relevant context from vector store
        relevant_docs = self.collection.query(
            query_texts=[question],
            n_results=5
        )

        # Get methodology and platform context
        methodologies = self.get_relevant_methods(analysis_type)
        platform_context = self.get_platform_context(df)

        # Create enriched prompt
        context = self.build_analysis_context(
            df, methodologies, platform_context, user_goals, relevant_docs
        )

        # Generate response using Ollama
        prompt = f"{context}\n\nQuestion: {question}\n\nAnswer:"
        return self.llm(prompt)
```

## 🎮 Day-by-Day Action Plan

### Day 1: Environment Setup

1. ✅ Create GitHub repository
2. ✅ Open in Codespaces
3. ✅ Install Ollama and dependencies

4. ✅ Test basic setup

### Day 2: Base System Test

1. ✅ Clone ragbase
2. ✅ Run original system with PDFs
3. ✅ Understand the code structure
4. ✅ Test Ollama integration

### Day 3-4: Excel Integration

1. 🔄 Replace PDF loader with Excel loader
2. 🔄 Modify Streamlit interface for Excel upload
3. 🔄 Test with sample Excel files
4. 🔄 Debug any issues

### Day 5-7: Basic Functionality

1. 🔄 Implement data summarization
2. 🔄 Create proper document chunking for tabular data
3. 🔄 Test end-to-end flow
4. 🔄 Optimize for performance

## 🎯 Success Metrics

### Week 1 Success Criteria

- [ ] Codespaces environment fully functional
- [ ] Ollama running and responding
- [ ] Excel file upload working
- [ ] Basic data display and summary
- [ ] Simple queries return responses
- [ ] No critical errors in processing

### Week 2 Success Criteria

- [ ] Handles files up to 5,000 rows (Codespaces limit)
- [ ] Proper data validation and error handling
- [ ] Multiple file format support
- [ ] Data insights and visualizations
- [ ] Memory optimization for cloud environment

### Week 3 Success Criteria

- [ ] Intelligent responses with business context
- [ ] Platform-specific recommendations
- [ ] Methodology-based analysis
- [ ] Accurate context retrieval
- [ ] Ecommerce knowledge base integration

### Week 4 Success Criteria

- [ ] Production-ready interface
- [ ] Export functionality
- [ ] Performance optimization for cloud
- [ ] User documentation complete
- [ ] Deployed and accessible via web link

## 📊 Knowledge Base Content

### Analysis Methodologies

- Campaign Performance Analysis Framework
- Budget Optimization Strategies
- Keyword Performance Evaluation
- Seasonal Trend Analysis
- Competitive Analysis Methods
- Attribution Analysis Approaches

### Platform-Specific Insights

- **Amazon**: Sponsored Products vs Sponsored Brands optimization
- **Flipkart**: Category-specific best practices
- **Quick Commerce**: Time-based optimization patterns
- **Cross-Platform**: Budget allocation strategies

## 🛠️ Development Guidelines

### Working in Codespaces

- **Save frequently**: Codespaces auto-saves but commit to Git regularly
- **Monitor usage**: 60 hours free per month
- **Optimize performance**: Close unused tabs, limit concurrent processes
- **Use port forwarding**: For Streamlit app testing

### Git Workflow

```bash
# Regular commits to save progress
git add .
git commit -m "Week 1: Excel processing implemented"
git push origin main

# Create branches for features
git checkout -b feature/excel-loader
git checkout -b feature/knowledge-base
```

### Testing Strategy

- Test with small Excel files first (< 1000 rows)
- Use sample ecommerce data for realistic testing
- Monitor memory usage in Codespaces
- Test different query types and patterns

## 🔍 Example Usage Flow

### Sample Analysis Session

1. **Upload**: User uploads Amazon campaign Excel file via Streamlit interface
2. **Processing**: System loads data, creates summaries, populates vector store
3. **Set Goals**: User inputs target ROAS = 3.0, Focus = Electronics
4. **Query**: User asks "Which campaigns are underperforming and why?"
5. **Analysis**: System retrieves relevant context, applies methodologies
6. **Response**:

   > "Found 8 campaigns below 3.0x ROAS target. Using proven ecommerce analysis methodology: Electronics campaigns show 34% better performance during evening hours. Top issues identified: 1) Broad match keywords consuming 40% budget inefficiently, 2) Creative performance below 1% CTR benchmark, 3) Suboptimal bidding timing. Recommendations: Increase evening bids by 20%, pause underperforming broad keywords, A/B test lifestyle vs product imagery."

## 📈 Deployment Options

### Free Deployment Options

1. **Streamlit Cloud**: Connect GitHub repo, deploy automatically
2. **Railway**: Free tier with GitHub integration
3. **Render**: Free web service hosting

4. **Heroku**: Free tier (limited hours)

### Recommended: Streamlit Cloud

```bash
# Add to requirements.txt
streamlit
pandas
chromadb
langchain
sentence-transformers

# Deploy via streamlit.io with GitHub integration
```

## 💡 Cost Optimization

### Free Tier Maximization

- **Codespaces**: 60 hours/month (plenty for development)
- **Ollama**: No API costs, runs locally in cloud
- **ChromaDB**: No external database costs
- **Streamlit Cloud**: Free hosting for public repos

### Scaling Considerations

- **Week 1-4**: Completely free
- **Production**: May need paid hosting for always-on service
- **Fine-tuning**: Would require paid GPU access later

## 🆘 Troubleshooting

### Common Codespaces Issues

- **Ollama not starting**: Run `ollama serve &` in terminal
- **Memory issues**: Restart Codespace, use smaller datasets
- **Port conflicts**: Use different ports for Streamlit
- **Model download fails**: Check internet connection, retry

### Performance Optimization

- **Large files**: Process in chunks, show progress bars
- **Memory usage**: Clear variables, optimize pandas operations
- **Vector store**: Limit document chunks for large datasets

## 📚 Learning Resources

### Codespaces & Git

- GitHub Codespaces Documentation
- Git Basics Tutorial

### RAG & AI

- LangChain Documentation
- ChromaDB Guide
- Ollama Documentation

### Streamlit Development

- Streamlit Documentation
- Building Data Apps Tutorial

## 🚀 Next Steps

1. **Create GitHub repository** for the project
2. **Open in Codespaces** and set up environment
3. **Follow Day 1-2 setup** instructions
4. **Begin Week 1 implementation** tasks
5. **Schedule weekly progress reviews**

## 📞 Support & Monitoring

- **Codespaces usage**: Monitor hours in GitHub settings
- **Performance**: Use `htop` to monitor resources
- **Git commits**: Regular saves prevent data loss
- **Documentation**: Keep notes on solutions and issues

---

**Project Start Date**: [Today's Date]
**Target Completion**: [Today + 4 weeks]
**Development Environment**: GitHub Codespaces
**Repository**: `https://github.com/[username]/ecommerce-rag-analyzer`
**Deployment**: Streamlit Cloud (Free)