Analysis without Scraped Data nor Onboarding File

Old financial_analysis.py

```
import pandas as pd
import numpy as np
import re
from groq import Groq
import requests
import os
from dotenv import load_dotenv
from tenacity import retry, stop_after_attempt, wait_random_exponential
from pinecone import Pinecone, ServerlessSpec
import time
from serpapi import GoogleSearch
from bs4 import BeautifulSoup
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from collections import defaultdict
import json
from serpapi.google_search import GoogleSearch

load_dotenv()

# Initialize Pinecone
pc = Pinecone(api_key=os.environ.get("PINECONE_API_KEY"))
index_name = "financial-data-index"

# Create index if it doesn't exist
if index_name not in pc.list_indexes().names():
    pc.create_index(
        name=index_name,
        dimension=768,
        metric='cosine',
        spec=ServerlessSpec(
            cloud='aws',
            region='us-east-1'
        )
    )

index = pc×Index(index_name)

# Set up API keys
client = Groq(api_key=os.environ.get("GROQ_API_KEY"))
perplexity_api_key = os.environ.get("PERPLEXITY_API_KEY")
SERPAPI_API_KEY = os.environ×get("SERPAPI_API_KEY")

# Initialize sentiment analyzer
```

```
analyzer = SentimentIntensityAnalyzer()

# Simple cache for market reports
market_report_cache = {}

def search_company_website(company_name):
    """Find official website and founder information using search"""
    try:
        # First search for official website
        website_params = {
            "q": f"{company_name} official website",
            "api_key": SERPAPI_API_KEY,
            "num": 3
        }
        website_search = GoogleSearch(website_params)
        website_results = website_search.get_dict()
        website_url = None

        if website_results.get("organic_results"):
            for result in website_results["organic_results"]:
                if company_name.lower() in result.get("link", "").lower():
                    website_url = result["link"]
                    break
            if not website_url:
                website_url = website_results["organic_results"][0]["link"]

        # Then search for founder information
        founder_params = {
            "q": f"{company_name} founder OR founders OR co-founder OR co-
founders",
            "api_key": SERPAPI_API_KEY,
            "num": 5
        }
        founder_search = GoogleSearch(founder_params)
        founder_results = founder_search.get_dict()
        founder_urls = []

        if founder_results.get("organic_results"):
            for result in founder_results["organic_results"]:
                url = result×get("link", "")
                if url and any(term in url.lower() for term in ['founder', 'about', 'team',
'leadership']):
                    founder_urls.append(url)

        return {
            'website_url': website_url,
            'founder_urls': founder_urls[:3]  # Return top 3 most relevant URLs
```

```
        }

    except Exception as e:
        print(f"Search Error: {str(e)}")
        return {'website_url': None, 'founder_urls': []}

def scrape_website(url):
    """Enhanced website scraping with better content extraction"""
    try:
        headers = {'User-Agent': 'Mozilla/5.0'}
        response = requests×get(url, headers=headers, timeout=10)
        soup = BeautifulSoup(response×text, 'html.parser')

        # Remove unwanted elements
        for element in soup(['script', 'style', 'nav', 'footer', 'iframe', 'header', 'aside',
'form']):
            element.decompose()

        # Prioritize main content areas
        main_content = soup×find('main') or soup.find('article') or soup.find('div',
class_=re.compile(r'content|main', re.I))

        if main_content:
            text = main_content.get_text()
        else:
            text = soup.get_text()

        # Clean up text
        text = re.sub(r'\s+', ' ', text).strip()
        return text[:10000]  # First 10,000 characters for more context

    except Exception as e:
        print(f"Scraping Error: {str(e)}")
        return ""

def extract_founder_info(text, company_name):
    """Extract founder information from scraped text"""
    try:
        # Look for founder-related patterns
        patterns = [
            rf'(founder|co-founder|ceo|creator)\s*(?:of\s*{company_name})?\s*:
\s*([^\n]+)',
            rf'{company_name}\s*(?:was\s*)?founded\s*(?:by|in)\s*([^\n]+?)\s*(?:in|
on|with)',
            rf'([A-Z][a-z]+ [A-Z][a-z]+)\s*(?:is|was)\s*(?:the|a)\s*(?:founder|co-
founder|creator)'
        ]
```

```
        founders = set()
        for pattern in patterns:
            matches = re.finditer(pattern, text, re.IGNORECASE)
            for match in matches:
                name = match×group(1) if 'group' in match.groupdict() else
match.group(2)
                if name and len(name.split()) >= 2:  # At least first and last name
                    founders.add(name.strip())

        return list(founders)

    except Exception as e:
        print(f"Founder extraction error: {str(e)}")
        return []

def analyze_financial_literacy(text):
    """Analyze financial sophistication using text"""
    try:
        scores = analyzer.polarity_scores(text)
        financial_terms = len(re.findall(r'\b(revenue|profit|margin|growth|financial|
investment|capital|ROI|ROE|ROA|EBITDA)\b', text, re.IGNORECASE))

        if financial_terms > 10 and scores['compound'] >= 0.5:
            return "Highly Sophisticated"
        elif financial_terms > 5:
            return "Moderately Sophisticated"
        else:
            return "Basic Understanding"
    except:
        return "Unknown"

@retry(wait=wait_random_exponential(min=1, max=60),
stop=stop_after_attempt(3))
def generate_company_profile(company_name):
    """Generate comprehensive company profile with enhanced founder
information"""
    try:
        # Get website and founder URLs
        urls = search_company_website(company_name)
        website_url = urls['website_url']
        founder_urls = urls['founder_urls']

        # Scrape main website
        scraped_data = scrape_website(website_url) if website_url else "No
website found"
```

```
        # Scrape founder information from multiple sources
        founder_data = []
        for url in founder_urls:
            founder_text = scrape_website(url)
            if founder_text:
                founder_data.append(founder_text)

        # Combine all data for processing
        combined_data = f"""
        MAIN WEBSITE CONTENT:
        {scraped_data}

        FOUNDER INFORMATION SOURCES:
        {' '.join(founder_data)}
        """

        prompt = f"""Please analyze this scraped company data and generate a
detailed, well-structured profile for {company_name}:

        {combined_data}

        The profile should include these sections with rich, specific information:

        ## Company Overview
        - Full legal name
        - Year founded
        - Headquarters location
        - Industry sector
        - Core business activities
        - Major products/services
        - Brief history (1 paragraph)

        ## Leadership Information
        ### Executive Team
        - CEO: Name, tenure, background, education, notable achievements
        - Other C-level executives: Names and brief profiles
        - Board members: Names and affiliations if available

        ### Founder Details (MUST INCLUDE THIS SECTION)
        - Full names of all founders
        - Detailed background for each founder including:
          * Education history
          * Previous professional experience
          * Notable achievements
          * Current roles besides this company
          * Awards and recognition received
        - Founding story (how and why the company was started)
```

```
        - Founder's current involvement with the company

        ## Financial Health Assessment
        - Revenue trends if mentioned
        - Profitability indicators
        - Growth metrics
        - Any available financial ratios
        - Funding history if applicable

        ## Business Operations
        - Key business metrics
        - Operational highlights
        - Geographic reach
        - Employee count if available
        - Major partnerships

        ## Mission and Values
        - Official mission statement
        - Core values
        - Corporate social responsibility initiatives
        - Sustainability practices

        ## Market Position
        - Competitive advantages
        - Market share indicators
        - Industry rankings if available
        - Recent awards/recognitions

        Format the output as clean, organized markdown with proper section
headers.
        Include only information that can be reasonably inferred from the provided
data.
        For missing information, clearly state "Information not available".
        """

        response = client×chat×completions×create(
            messages=[
                {
                    "role": "system",
                    "content": "You are a business intelligence specialist that extracts
and structures detailed company information. Pay special attention to founder
details and ensure they are comprehensive. Provide well-organized output with
clear section headers and bullet points."
                },
                {
                    "role": "user",
                    "content": prompt
```

```
            }
        ],
        model="deepseek-r1-distill-llama-70b",
        temperature=0.2,
        max_tokens=4000
    )

    profile_text = response.choices[0].message.content

    # Process the structured response into sections
    profile = {
        'company_overview': extract_section(profile_text, "Company
Overview"),
        'leadership': extract_section(profile_text, "Leadership Information"),
        'founder_details': extract_section(profile_text, "Founder Details"),
        'financial_health': extract_section(profile_text, "Financial Health
Assessment"),
        'operations': extract_section(profile_text, "Business Operations"),
        'mission_values': extract_section(profile_text, "Mission and Values"),
        'market_position': extract_section(profile_text, "Market Position"),
        'scraped_data': combined_data[:2000] + "..." if len(combined_data) >
2000 else combined_data,
        'financial_literacy': analyze_financial_literacy(combined_data),
        'source_urls': {
            'website': website_url,
            'founder_sources': founder_urls
        }
    }

    return profile

  except Exception as e:
    print(f"Profile generation error: {str(e)}")
    return {
        'company_overview': 'Error generating profile',
        'leadership': 'Not available',
        'founder_details': 'Not available',
        'financial_health': 'Not available',
        'operations': 'Not available',
        'mission_values': 'Not available',
        'market_position': 'Not available',
        'scraped_data': 'Error during data collection',
        'financial_literacy': 'Unknown',
        'source_urls': {}
    }

def extract_section(text, section_title):
```

```
    """Enhanced section extractor that handles nested subsections"""
    patterns = [
        rf'## {section_title}(.*?)(?=## |\Z)',
        rf'{section_title}:(.*?)(?=\n\w+:|$)'
    ]

    for pattern in patterns:
        match = re×search(pattern, text, re.DOTALL | re.IGNORECASE)
        if match:
            content = match×group(1)×strip()
            content = re.sub(r'^\s*[-*]\s*', '', content, flags=re.MULTILINE)
            content = re×sub(r'\n{3,}', '\n\n', content)
            return content if content else "Information not available"

    return "Information not available"

def calculate_key_vitals(df):
    """Calculates key financial vitals based on available row names."""
    def get_row_total(row_name):
        if row_name in df.index:
            try:
                value = df.loc[row_name, 'Total']
                return float(value) if not pd.isna(value) else 0
            except (KeyError, ValueError):
                return 0
        return 0

    total_income = get_row_total('Total Income')
    total_cost_of_goods_sold = get_row_total('Total Cost Of Goods Sold')
    gross_profit = get_row_total('Gross Profit')
    total_expenses = get_row_total('Total Expenses')
    net_profit = get_row_total('Net Profit')
    selling_expenses = get_row_total('Advertising & Selling Expense')
    general_admin_expenses = get_row_total('General & Administrative
expenses')

    # Calculate ratios with proper error handling
    def safe_divide(numerator, denominator):
        return numerator / denominator if denominator != 0 else 0

    gross_margin = safe_divide(gross_profit, total_income) * 100
    operating_expense_ratio = safe_divide(total_expenses, total_income) * 100
    net_profit_margin = safe_divide(net_profit, total_income) * 100
    cost_of_goods_sold_percent = safe_divide(total_cost_of_goods_sold,
total_income) * 100
    selling_expense_ratio = safe_divide(selling_expenses, total_income) * 100
    ga_expense_ratio = safe_divide(general_admin_expenses, total_income) *
```

```
    return {
        "Gross Profit Margin": round(gross_margin, 2),
        "Operating Expense Ratio": round(operating_expense_ratio, 2),
        "Net Profit Margin": round(net_profit_margin, 2),
        "Cost of Goods Sold Ratio": round(cost_of_goods_sold_percent, 2),
        "Selling Expense Ratio": round(selling_expense_ratio, 2),
        "G&A Expense Ratio": round(ga_expense_ratio, 2),
    }

def embed_data(data):
    """Improved embedding function"""
    # In production, replace with actual embedding model
    return np.random.rand(768).tolist()

def upload_to_pinecone(file_paths):
    """Upload financial data to Pinecone with enhanced error handling"""
    for file_path in file_paths:
        try:
            df = pd.read_csv(file_path)
            if 'Name' in df.columns:
                df = df.set_index('Name')
            df×index = df.index.astype(str).str.strip()

            key_vitals = calculate_key_vitals(df)
            if not key_vitals:
                print(f"Skipping {file_path} - no valid metrics calculated")
                continue

            vector = embed_data(list(key_vitals.values()))
            metadata = key_vitals
            record_id = os.path.basename(file_path)

            # Upsert to Pinecone
            index.upsert(
                vectors=[{
                    "id": record_id,
                    "values": vector,
                    "metadata": metadata
                }],
                namespace="financial_data"
            )
            print(f"Successfully uploaded {file_path} to Pinecone")

        except Exception as e:
            print(f"Error uploading {file_path}: {str(e)}")
```

```
def get_industry_averages():
    """Retrieve and calculate industry averages with proper error handling"""
    try:
        # List all vectors in the namespace
        stats = index.describe_index_stats()
        if stats.namespaces.get("financial_data", {}).get("vector_count", 0) == 0:
            print("No data found in Pinecone index")
            return {}

        # Query to get all vectors
        query_result = index×query(
            vector=[0]*768,  # Dummy vector
            top_k=100,
            include_metadata=True,
            namespace="financial_data"
        )

        if not query_result.matches:
            print("No matches found in Pinecone query")
            return {}

        # Extract all metadata
        all_vitals = [match.metadata for match in query_result.matches if
match.metadata]

        if not all_vitals:
            print("No valid metadata found in Pinecone results")
            return {}

        # Calculate averages
        industry_averages = {}
        metric_keys = [
            "Gross Profit Margin",
            "Operating Expense Ratio",
            "Net Profit Margin",
            "Cost of Goods Sold Ratio",
            "Selling Expense Ratio",
            "G&A Expense Ratio"
        ]

        for key in metric_keys:
            values = [float(vitals.get(key, 0)) for vitals in all_vitals if vitals.get(key) is
not None]
            if values:
                industry_averages[key] = round(sum(values) / len(values), 2)
            else:
```

```
            industry_averages[key] = 0.0

        print("Calculated industry averages:", industry_averages)
        return industry_averages

    except Exception as e:
        print(f"Error retrieving industry averages: {str(e)}")
        return {}

def compare_to_industry_average(uploaded_file, industry_averages):
    """Compare company data to industry averages with enhanced validation"""
    try:
        df_company = pd.read_csv(uploaded_file)
        if 'Name' in df_company.columns:
            df_company = df_company.set_index('Name')
        df_company×index = df_company.index.astype(str).str.strip()

        company_vitals = calculate_key_vitals(df_company)
        if not company_vitals:
            raise ValueError("No valid metrics calculated for company")

        comparison_results = {}
        for metric, company_value in company_vitals.items():
            industry_value = industry_averages.get(metric, 0)

            if metric.endswith("Margin"):
                verdict = "Outperforming" if company_value > industry_value else
"Underperforming"
            elif metric.endswith("Ratio"):
                verdict = "Outperforming" if company_value < industry_value else
"Underperforming"
            else:
                verdict = "N/A"

            comparison_results[metric] = {
                "Company Value": round(company_value, 2),
                "Industry Average": round(industry_value, 2),
                "Verdict": verdict,
                "Difference": round(company_value - industry_value, 2)
            }

        return comparison_results

    except Exception as e:
        print(f"Comparison error: {str(e)}")
        return {}
```

```
@retry(wait=wait_random_exponential(min=1, max=60),
stop=stop_after_attempt(3))
def generate_market_report_perplexity(company_name):
    """Generate market report with enhanced error handling"""
    if company_name in market_report_cache:
        return market_report_cache[company_name]

    if not perplexity_api_key:
        return "Error: Perplexity API key is missing."

    url = "https://api.perplexity.ai/chat/completions"
    prompt = f"""Generate a detailed market report for {company_name}
including:
    1. Company Overview
    2. Market Size and Growth
    3. Target Market
    4. Competitive Landscape
    5. Key Trends
    6. SWOT Analysis
    7. Future Outlook
    Use specific numbers and cite sources when possible."""

    payload = {
        "model": "sonar-pro",
        "messages": [
            {
                "role": "system",
                "content": "You are a financial analyst providing accurate, data-driven
insights."
            },
            {
                "role": "user",
                "content": prompt
            }
        ],
        "max_tokens": 3000,
        "temperature": 0.1
    }

    headers = {
        "Authorization": f"Bearer {perplexity_api_key}",
        "Content-Type": "application/json"
    }

    try:
        response = requests×post(url, json=payload, headers=headers)
```

```
        response.raise_for_status()
        result = response.json()["choices"][0]["message"]["content"]

        # Clean up response
        result = re.sub(r'\[\d+\]', '', result)  # Remove citations
        result = re.sub(r'\n{3,}', '\n\n', result)  # Remove excessive newlines
        result = re.sub(r'<think>.*?</think>', '', result, flags=re.DOTALL)  # Remove
AI thinking

        market_report_cache[company_name] = result
        return result

    except Exception as err:
        return f"Error generating report: {str(err)}"

@retry(wait=wait_random_exponential(min=1, max=60),
stop=stop_after_attempt(3))
def analyze_company_standing(company_statement, industry_averages,
market_report, company_name):
    """Generate company analysis with enhanced structure"""
    prompt = f"""Analyze {company_name}'s performance compared to industry
benchmarks:

    Company Data: {json.dumps(company_statement, indent=2)}
    Industry Averages: {json.dumps(industry_averages, indent=2)}
    Market Context: {market_report[:2000]}...

    Provide analysis in this structure:
    ### Financial Performance Summary
    (Brief overview of key findings)

    ### Strengths
    (3 specific strengths with data points)

    ### Weaknesses
    (3 specific weaknesses with data points)

    ### Recommendations
    (3 actionable recommendations)

    ### Overall Verdict
    (Clear conclusion: Outperforming/Underperforming/On Par)

    Remove any internal thinking tags or commentary.
    Use precise numbers and maintain consistency with the provided data."""

    response = client×chat×completions×create(
```

```
        messages=[
            {
                "role": "system",
                "content": "You are a financial analyst providing detailed, data-driven
company analysis. Provide clean output without any internal thinking tags."
            },
            {
                "role": "user",
                "content": prompt
            }
        ],
        model="deepseek-r1-distill-llama-70b",
        temperature=0.1,
        max_tokens=3000
    )

    analysis = response.choices[0].message.content
    # Remove any remaining AI thinking patterns
    analysis = re.sub(r'<think>.*?</think>', '', analysis, flags=re.DOTALL)
    analysis = re.sub(r'\[.*?\]', '', analysis)
    return analysis
```

old main.py

```
import streamlit as st
from PIL import Image
import pandas as pd
from financial_analysis import generate_market_report_perplexity,
compare_to_industry_average, calculate_key_vitals, upload_to_pinecone,
get_industry_averages, analyze_company_standing, generate_company_profile
import os
from io import StringIO
import time

# Set page config
st.set_page_config(page_title="FinBot", layout="wide")

# Define colors
primary_color = "#6bc72e"
text_color = "#333333"

# Custom CSS
st.markdown(f"""
    <style>
    /* Global Styles */
```

```
    .stApp {{
        background-color: #ffffff;
        font-family: 'Arial', sans-serif;
    }}
    .stSidebar {{
        background-color: {primary_color};
        padding: 20px;
    }}
    .stSidebar h3, .stSidebar h4, .stSidebar .highlight {{
        color: white;
        font-size: 1.2em;
        font-weight: bold;
    }}
    .stSidebar p {{
        color: {text_color};
    }}

    /* Button Styles */
    .stButton > button {{
        background-color: white;
        color: {primary_color};
        border: 2px solid {primary_color};
        border-radius: 5px;
        padding: 10px;
        font-size: 16px;
    }}
    .stButton > button:hover, .stButton > button:focus, .stButton > button:active
{{
        background-color: {primary_color} !important;
        color: white !important;
        border: 2px solid {primary_color};
        border-radius: 5px;
        outline: none !important;
        box-shadow: none !important;
    }}

    /* Input Fields without green borders */
    input, textarea, select {{
        border: 1px solid #cccccc !important;
        border-radius: 5px;
        padding: 10px;
        font-size: 16px;
    }}

    /* Remove focus shadow */
    input:focus, textarea:focus, select:focus {{
        outline: none !important;
```

```
    box-shadow: none !important;
}}

/* Expander Styles */
.stExpander {{
    border: 1px solid #d3d3d3;
}}
.stExpander:hover, .stExpander:focus, details[open] summary {{
    border-color: {primary_color} !important;
    color: {primary_color} !important;
    outline: none !important;
}}
details summary:hover {{
    color: {primary_color} !important;
}}

/* Navigation Tabs with single green underline */
.stTabs [data-baseweb="tab"] {{
    height: 45px;
    background-color: #fefefd;
    border-radius: 8px 8px 0px 0px;
    font-weight: bold;
    padding: 12px;
    color: {text_color} !important;
    border-bottom: none;
}}

/* Active tab: Force single green underline */
.stTabs [aria-selected="true"] {{
    color: {primary_color} !important;
    border-bottom: 3px solid {primary_color} !important;
}}

/* Ensure no red focus outline or shadow */
*:focus, *:active, *:hover {{
    outline: none !important;
    box-shadow: none !important;
}}

/* Override any remaining red borders */
[data-baseweb="tab-highlight"] {{
    background-color: transparent !important;
}}

/* Green headings */
h1, h2, h3 {{
    color: {primary_color} !important;
```

```
    }}

    /* FinBot image alignment */
    .finbot-image {{
        display: flex;
        justify-content: flex-end;
        align-items: flex-end;
        height: 100%;
    }}

    /* White text for sidebar title */
    .sidebar-title {{
        color: white !important;
    }}

    /* Market Report Styling */
    .market-report {{
        background-color: #f9f9f9;
        border-left: 5px solid {primary_color};
        padding: 20px;
        margin-top: 20px;
        border-radius: 5px;
        box-shadow: 0 2px 5px rgba(0,0,0,0.1);
        font-family: 'Roboto', sans-serif;
    }}
    .market-report h4 {{
        color: #4a4a4a;
        margin-bottom: 15px;
        font-size: 24px;
        font-weight: 600;
    }}
    .market-report p {{
        line-height: 1.6;
        color: {text_color};
        font-size: 16px;
    }}

    </style>
    """, unsafe_allow_html=True)

# Initialize session state
if 'company_name' not in st.session_state:
    st.session_state.company_name = ""
if 'market_report' not in st.session_state:
    st.session_state.market_report = ""
if 'show_prompt' not in st.session_state:
    st.session_state.show_prompt = False
```

```
if 'company_profile' not in st.session_state:
    st.session_state.company_profile = None

# Load images
main_logo = Image.open("images/main_logo.png")
overview_image = Image×open("images/overview.jpeg")
finbot_image = Image×open("images/finbot.jpg")

# Sidebar with branding and description
with st.sidebar:
    st.image(main_logo, use_container_width=True)
    st.markdown("""
        <h3 class="sidebar-title">mypocketCFO: Your Financial Companion</h3>

        mypocketCFO is an innovative financial management tool designed to
empower small businesses and entrepreneurs. Our AI-driven platform provides
real-time financial insights, predictive analytics, and personalized
recommendations to help you make informed decisions and drive growth.

        <h4>Key features:</h4>

        - Automated bookkeeping and financial reporting
        - Cash flow forecasting and budget optimization
        - Customized financial advice and strategy planning
        - Integration with popular accounting software

        <p class="highlight">Let mypocketCFO be your trusted financial partner
on your journey to success!</p>
    """, unsafe_allow_html=True)

# Tabs for navigation
tab1, tab2, tab3 = st.tabs(["Overview", "Analysis", "Profile"])

with tab1:
    st.header("Overview", help="Company overview and market insights")
    col1, col2 = st.columns([2, 1])

    with col1:
        st.subheader("Company Information")
        st.write("Enter the company name")
        company_name = st.text_input("Company Name",
value=st.session_state.company_name, key="company_name_input",
label_visibility="collapsed")
        if st.button("Enter"):
            if not company_name:
                st.session_state.show_prompt = True
                st.session_state.company_name = ""
```

```
            st.session_state.market_report = ""
        else:
            st.session_state.company_name = company_name
            with st.spinner("Generating market report..."):
                market_report =
generate_market_report_perplexity(company_name)
                if market_report.startswith("Error:"):
                    st.error(market_report)
                    st.session_state.market_report = ""
                else:
                    st.session_state.market_report = market_report
            st.session_state.show_prompt = False

    if st.session_state.show_prompt:
        st.markdown('<p class="prompt-box">Please enter a company name</
p>', unsafe_allow_html=True)

with col2:
    st.image(overview_image, use_container_width=True)

if st.session_state.market_report:
    st.markdown("""
    <div class="market-report">
        <h4>Market Report</h4>
        <p>{}</p>
    </div>
    """.format(st.session_state.market_report), unsafe_allow_html=True)

with tab2:
    col1, col2 = st.columns([3, 1])

    with col1:
        st.header("FinBot", help="AI-powered financial analysis")
        st.write("*FinBot provides a deep dive into financial performance,
identifying key trends and risks.*")

        # File uploader for single company CSV file
        company_file = st.file_uploader("Upload your company's CSV file",
type="csv")

        process_data = st.button("Process Data and Compare")

    with col2:
        st.markdown('<div class="finbot-image">', unsafe_allow_html=True)
        st.image(finbot_image, width=500)
        st.markdown('</div>', unsafe_allow_html=True)
```

```
    if process_data:
        if not st.session_state.company_name:
            st.markdown('<p class="prompt-box">Enter a company name in
Overview first.</p>', unsafe_allow_html=True)
        elif not company_file:
            st.markdown('<p class="prompt-box">Please upload your company\'s
CSV file to compare.</p>', unsafe_allow_html=True)
        else:
            with st.spinner("Processing data..."):
                # Upload industry data to Pinecone
                industry_files = ["data/income_statement1.csv", "data/
income_statement2.csv"]
                upload_to_pinecone(industry_files)

                # Add a delay to ensure data is processed
                time.sleep(2)

                # Get industry averages from Pinecone
                industry_averages = get_industry_averages()

                if not industry_averages:
                    st.error("Failed to retrieve industry data. Please try again.")
                else:
                    # Perform the comparison
                    try:
                        comparison_results =
compare_to_industry_average(company_file, industry_averages)

                        st.subheader("Comparison to Industry Averages")

                        # Display the comparison results in a table
                        data = []
                        for metric, values in comparison_results.items():
                            data.append([metric, values['Company Value'],
values['Industry Average'], values['Verdict']])

                        df = pd×DataFrame(data, columns=['Metric', 'Company Value',
'Industry Average', 'Verdict'])
                        st.dataframe(df)

                        # Generate and display the company standing analysis
                        st.subheader("Company Standing Analysis")
                        company_statement = df×to_dict()
                        market_report = st.session_state.market_report
                        analysis = analyze_company_standing(company_statement,
industry_averages, market_report, st.session_state.company_name)
```

```
                st.markdown(analysis)

            except Exception as e:
                st.error(f"Error processing file: {e}")

with tab3:
    st.header("Company Profile", help="Detailed company and founder insights")

    if st.session_state.company_name:
        if st.button("Generate Comprehensive Profile"):
            with st.spinner("Generating detailed company profile..."):
                try:
                    profile_data =
generate_company_profile(st.session_state.company_name)
                    st.session_state.company_profile = profile_data

                    if st.session_state.company_profile:
                        st.subheader(f"Comprehensive Profile:
{st.session_state.company_name}")

                        # Source information
                        with st.expander("📌 Data Source Information",
expanded=False):
                            st.write("**Main Website:**")
                            st.write(profile_data['source_urls'].get('website', 'Not
available'))

                            st.write("**Founder Information Sources:**")
                            for url in profile_data['source_urls'].get('founder_sources', []):
                                st.write(url)

                            st.write("**Raw Scraped Data Sample:**")
                            st.code(profile_data.get('scraped_data', 'No data available'),
language='text')
                            st.write(f"**Content Sophistication:**
{profile_data.get('financial_literacy', 'Unknown')}")

                        # Main profile sections with added founder section
                        with st.expander("🏢 Company Overview", expanded=True):
                            st.markdown(profile_data.get('company_overview',
'Information not available'))

                        with st.expander("👔 Leadership & Management"):
                            st.markdown(profile_data.get('leadership', 'Information not
available'))

                        with st.expander("👨‍💼 Founder Details", expanded=True):
```

```
                    founder_content = profile_data.get('founder_details',
'Information not available')
                    if founder_content == 'Information not available':
                        st.warning("Could not extract detailed founder information.
Trying alternative methods...")
                        # You could add additional fallback logic here
                    st.markdown(founder_content)

                with st.expander("💰 Financial Health"):
                    st.markdown(profile_data.get('financial_health', 'Information
not available'))

                with st.expander("🏭 Business Operations"):
                    st.markdown(profile_data.get('operations', 'Information not
available'))

                with st.expander("🌟 Mission & Values"):
                    st.markdown(profile_data.get('mission_values', 'Information
not available'))

                with st.expander("📈 Market Position"):
                    st.markdown(profile_data.get('market_position', 'Information
not available'))

                    # Download button with founder info included
                    profile_text = f"""
                    # {st.session_state.company_name} Profile Report

                    ## Company Overview
                    {profile_data.get('company_overview', '')}

                    ## Leadership & Management
                    {profile_data.get('leadership', '')}

                    ## Founder Details
                    {profile_data.get('founder_details', '')}

                    ## Financial Health
                    {profile_data.get('financial_health', '')}

                    ## Business Operations
                    {profile_data.get('operations', '')}

                    ## Mission & Values
                    {profile_data.get('mission_values', '')}

                    ## Market Position
```

```
                    {profile_data.get('market_position', '')}
                    """

                    st.download_button(
                        label="Download Full Profile",
                        data=profile_text,
                        file_name=f"{st.session_state.company_name}_profile.md",
                        mime="text/markdown"
                    )
                else:
                    st.error("Failed to generate comprehensive company profile")
            except Exception as e:
                st.error(f"Profile generation error: {str(e)}")
    else:
        st.info("ℹ️ Please enter a company name in the Overview tab to generate profile")
```