Analysis with both Scraped Data and Onboarding File

financial_analysis.py

```
import pandas as pd
import numpy as np
import re
import os
import requests
import time
import json
from dotenv import load_dotenv
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from bs4 import BeautifulSoup
from serpapi import GoogleSearch
from groq import Groq
from pinecone import Pinecone, ServerlessSpec
import PyPDF2
from tenacity import retry, wait_random_exponential, stop_after_attempt

# Load environment variables
load_dotenv()

# Initialize APIs
client = Groq(api_key=os.getenv("GROQ_API_KEY"))
SERPAPI_API_KEY = os.getenv("SERPAPI_API_KEY")
perplexity_api_key = os.getenv("PERPLEXITY_API_KEY")

# Pinecone setup
pc = Pinecone(api_key=os×getenv("PINECONE_API_KEY"))
index_name = "financial-data-index"

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)

# Sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Cache for market reports
market_report_cache = {}
```

```
# Onboarding PDF file path
ONBOARDING_FILE_PATH = "Tinge/TingeBeauty_onboarding.pdf"

# Mapping topics to categorize tags
tag_mapping = {
    "Chart of accounts": "O",
    "Revenue recognition": "O",
    "Sales channel": "O",
    "Cost of goods sold": "O",
    "Expenses": "O",
    "Sales & marketing expenses": "O",
    "G&A expense": "O",
    "Accounts receivable": "O",
    "Accounts payable": "O",
    "Vendors": "O",
    "Customers": "O",
    "Reconciliation": "O",
    "Growth": "S",
    "Margin": "S",
    "Profitability": "S",
    "Cash": "S",
    "Gender": "P",
    "Role": "P",
    "Education": "P",
    "Financial literacy": "P",
    "Goal": "P",
    "Onboarding": "J",
    "Compliance": "J",
    "Operations": "J",
    "Working capital": "J",
    "Equity capital": "S",
    "Unit economics": "S",
    "Alternative capital": "S",
    "Top of mind": "P",
}

# ---------------------- ONBOARDING HANDLING FUNCTIONS
----------------------

def read_onboarding_text():
    """Reads the onboarding PDF and extracts text."""
    try:
        with open(ONBOARDING_FILE_PATH, "rb") as file:
            reader = PyPDF2.PdfReader(file)
            text = " "×join([page.extract_text() for page in reader.pages])
        return text
```

```
    except Exception as e:
        print(f"Onboarding file read error: {e}")
        return ""

def analyze_onboarding_tags(text):
    """Detects Strategic / Operational / Persona tags from onboarding text."""
    detected_tags = set()
    if text:
        text_lower = text×lower()
        for keyword, tag in tag_mapping.items():
            if keyword.lower() in text_lower:
                detected_tags.add((keyword, tag))
    return list(detected_tags)

def extract_action_items(text):
    """Extracts Action Items from onboarding transcript."""
    pattern = r"•\s*\[\s*\]\s*(.*?)(?:\n|$)"
    matches = re.findall(pattern, text)
    return [match.strip() for match in matches]

def summarize_onboarding(text):
    """Summarizes onboarding by sentiment + detected tags + action items."""
    if not text:
        return {"sentiment": "Unknown", "tags_detected": [],
"action_items_summary": []}
    scores = analyzer.polarity_scores(text)
    sentiment = "Positive" if scores['compound'] > 0.3 else "Negative" if
scores['compound'] < -0.3 else "Neutral"
    tags_detected = analyze_onboarding_tags(text)
    action_items = extract_action_items(text)
    return {
        "sentiment": sentiment,
        "tags_detected": tags_detected,
        "action_items_summary": action_items
    }

def rate_financial_literacy(onboarding_summary):
    """Classifies founder as Strategic / Operational based on onboarding tags."""
    strategic_topics = {"Growth", "Margin", "Profitability", "Cash", "Equity
capital", "Unit economics", "Alternative capital"}
    operational_topics = {"Chart of accounts", "Revenue recognition", "Cost of
goods sold", "Expenses", "Reconciliation", "Vendors", "Accounts receivable",
"Accounts payable"}

    tags_found = {tag for keyword, tag in
onboarding_summary.get('tags_detected', [])}
```

```
    if any(tag == "S" for tag in tags_found):
        return (
            "Strategic",
            "Founder is classified as **Strategic** due to emphasis on growth,
profitability, equity capital, and strategic financial management topics during
onboarding discussions."
        )
    elif any(tag == "O" for tag in tags_found):
        return (
            "Operational",
            "Founder is classified as **Operational** due to heavy focus on day-to-
day operations like reconciliation, expenses, vendors, and cost management
during onboarding."
        )
    else:
        return (
            "Unknown",
            "Not enough onboarding discussion topics detected to confidently
classify founder's financial literacy style."
        )

# ------------------------ SEARCH, SCRAPE, UPLOAD, VITALS FUNCTIONS
------------------------

def search_company_website(company_name):
    """Find official website and founder-related pages using SerpAPI."""
    try:
        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']

        founder_params = {
            "q": f"{company_name} founder OR co-founder",
```

```
            "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]}  # Limit
to 3 founder links
    except Exception as e:
        print(f"Search error: {str(e)}")
        return {'website_url': None, 'founder_urls': []}

def scrape_website(url):
    """Scrape clean text from a given URL."""
    try:
        headers = {'User-Agent': 'Mozilla/5.0'}
        response = requests×get(url, headers=headers, timeout=10)
        soup = BeautifulSoup(response×text, 'html.parser')
        for element in soup(['script', 'style', 'nav', 'footer', 'iframe', 'header', 'aside',
'form']):
            element.decompose()
        main_content = soup×find('main') or soup.find('article') or soup.find('div',
class_=re.compile(r'content|main', re.I))
        text = main_content.get_text() if main_content else soup.get_text()
        text = re.sub(r'\s+', ' ', text).strip()
        return text[:10000]
    except Exception as e:
        print(f"Scraping error: {e}")
        return ""

def embed_data(data):
    """Dummy embedding (for Pinecone upload)."""
    return np.random.rand(768).tolist()

def calculate_key_vitals(df):
    """Calculates all key financial metrics from Income Statement."""
    def safe_divide(n, d):
        return n / d if d != 0 else 0
```

```
    def get_total(name):
        try:
            return float(df.loc[name, 'Total'])
        except:
            return 0

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

    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) *
100

    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 upload_to_pinecone(file_paths):
    """Uploads income statement financials to Pinecone vector database."""
    for path in file_paths:
        try:
            df = pd.read_csv(path)
            if 'Name' in df.columns:
                df = df.set_index('Name')
            df×index = df.index.astype(str).str.strip()

            vitals = calculate_key_vitals(df)

            if not vitals:
                print(f"Skipping {path} - no valid financial metrics calculated")
                continue
```

```
            vector = embed_data(list(vitals.values()))
            index.upsert(
                vectors=[{
                    "id": os.path.basename(path),
                    "values": vector,
                    "metadata": vitals
                }],
                namespace="financial_data"
            )
            print(f"Uploaded {path} to Pinecone successfully.")
        except Exception as e:
            print(f"Upload error for {path}: {str(e)}")

def get_industry_averages():
    """Retrieve industry average financial metrics from Pinecone."""
    try:
        stats = index.describe_index_stats()
        if stats.namespaces.get("financial_data", {}).get("vector_count", 0) == 0:
            print("No vectors found in Pinecone.")
            return {}

        query_result = index×query(vector=[0]*768, top_k=100,
include_metadata=True, namespace="financial_data")
        if not query_result.matches:
            print("No matches found in query.")
            return {}

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

        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

        return industry_averages
    except Exception as e:
        print(f"Industry averages error: {str(e)}")
        return {}

def compare_to_industry_average(uploaded_file, industry_averages):
    """Compare uploaded company's metrics to industry averages."""
    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 financial metrics calculated for uploaded file.")

        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 {}

# ----------------------- MARKET REPORT GENERATION FUNCTION
-----------------------

@retry(wait=wait_random_exponential(min=1, max=60),
stop=stop_after_attempt(3))
def generate_market_report_perplexity(company_name):
```

```
    """Generates a detailed market report for the company (without SWOT
analysis)."""
    if company_name in market_report_cache:
        return market_report_cache[company_name]

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

    url = "https://api.perplexity.ai/chat/completions"

    prompt = f"""
    Please generate a detailed market report for {company_name} including the
following sections:
    - Company Overview
    - Market Size and Growth
    - Target Market
    - Competitive Landscape
    - Key Trends
    - Future Outlook

    Important:
    - DO NOT include SWOT Analysis.
    - Make sure to use bullet points wherever possible for clarity.
    """

    payload = {
        "model": "sonar-pro",
        "messages": [
            {"role": "system", "content": "You are a professional financial research
analyst providing structured reports."},
            {"role": "user", "content": prompt}
        ],
        "max_tokens": 3000,
        "temperature": 0.2
    }

    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 formatting issues if any
```

```
        result = re.sub(r'\[\d+\]', '', result)  # Remove references like [1]
        result = re.sub(r'\n{3,}', '\n\n', result)

        # Cache it
        market_report_cache[company_name] = result

        return result

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

# ------------------------ COMPANY PROFILE GENERATION
------------------------

@retry(wait=wait_random_exponential(min=1, max=60),
stop=stop_after_attempt(3))
def generate_company_profile(company_name):
    """Generate a complete company profile with enhanced founder scraping
and correctly split subtabs."""
    try:
        # Read onboarding info
        onboarding_text = read_onboarding_text()
        onboarding_summary = summarize_onboarding(onboarding_text)

        # Founder Literacy smart scoring
        strategic_topics = {"Growth", "Margin", "Profitability", "Cash", "Equity
capital", "Unit economics", "Alternative capital"}
        operational_topics = {"Chart of accounts", "Revenue recognition", "Cost of
goods sold", "Expenses", "Reconciliation", "Vendors", "Accounts receivable",
"Accounts payable"}

        detected_tags = {keyword for keyword, tag in
onboarding_summary.get('tags_detected', [])}
        strategic_count = len(strategic_topics.intersection(detected_tags))
        operational_count = len(operational_topics.intersection(detected_tags))
        total_count = strategic_count + operational_count

        if total_count == 0:
            founder_rating = "Unknown"
            rating_reason = "Not enough onboarding discussion topics detected to
classify founder's financial literacy."
        elif strategic_count > 0 and operational_count == 0:
            founder_rating = "Strategic"
            rating_reason = "Founder is classified as **Strategic** due to focus on
strategic financial planning like growth, margins, and profitability."
        elif operational_count > 0 and strategic_count == 0:
            founder_rating = "Operational"
```

```
            rating_reason = "Founder is classified as **Operational** due to focus on day-to-day management like reconciliation, vendor handling, and expense control."
        else:
            strategic_pct = round((strategic_count / total_count) * 100)
            operational_pct = 100 - strategic_pct
            founder_rating = f"Strategic {strategic_pct}% / Operational {operational_pct}%"
            rating_reason = f"Founder shows a mixed style: **{strategic_pct}% Strategic** (growth, cash management) and **{operational_pct}% Operational** (reconciliation, cost management) based on onboarding topics."

        # Search for website and founder info
        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 ""

        # Scrape multiple founder pages smartly
        founder_data = []
        if founder_urls:
            for url in founder_urls:
                text = scrape_website(url)
                if text:
                    founder_data.append(text)

        founder_text_combined = "\n".join(founder_data) if founder_data else "No detailed founder information found."

        combined_data = f"""
        MAIN WEBSITE CONTENT:
        {scraped_data}

        FOUNDER INFORMATION SOURCES:
        {founder_text_combined}
        """

        # If scraping fails totally
        if not scraped_data and not founder_data:
            combined_data = "No website or founder data available from public sources."

        # LLM Prompt for clean structured profile
        prompt = f"""
        Create a detailed and very clean company profile for {company_name}
```

```
based on the following data:
        {combined_data}

        Include onboarding insights:
        - Sentiment: {onboarding_summary.get('sentiment', 'Unknown')}
        - Topics: {', '.join([k for k, _ in onboarding_summary.get('tags_detected',
[])])}
        - Action Items: {', '.join(onboarding_summary.get('action_items_summary',
[]))}

        STRUCTURE the report as:

        ## Company Overview
        (Write company overview here...)

        ## Leadership Information
        (Write about the leadership team here...)

        ## Founder Details
        (Write detailed founder background, skills, philosophy here.)

        ## Financial Health
        (Write about revenue streams, margins, cost structures.)

        ## Business Operations
        (Write about how operations are managed, supply chain, logistics.)

        ## Mission and Values
        (Write company's mission, culture, values.)

        ## Market Position
        (Write about competitors, market ranking.)

        Very important:
        - Insert clear section headers like "## Company Overview" etc.
        - NEVER mix content between sections.
        - End one section cleanly before starting next.
        - No duplication across sections.
        - **IMPORTANT**: Exclude Alice Zhang if mentioned anywhere.
        """

        response = client×chat×completions×create(
            messages=[
                {"role": "system", "content": "You are a professional analyst creating
structured clean profiles."},
                {"role": "user", "content": prompt}
            ],
```

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

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

        # REMOVE any <think> or stray AI tags
        profile_text = re.sub(r'<think>.*?</think>', '', profile_text,
flags=re.DOTALL)
        profile_text = profile_text.replace('<think>', '').replace('</think>', '')

        # REMOVE any line mentioning Alice Zhang
        profile_text = re.sub(r".*Alice Zhang.*\n?", "", profile_text,
flags=re.IGNORECASE)

        # Very careful section splitter
        def extract_section(section_title):
            try:
                pattern = rf"## {re.escape(section_title)}\n(.*?)(?=\n## |\Z)"
                match = re×search(pattern, profile_text, re.DOTALL)
                if match:
                    content = match×group(1)×strip()
                    return content if content else "Not available"
                return "Not available"
            except Exception as e:
                print(f"Section extract error for {section_title}: {e}")
                return "Not available"

        # Final profile dictionary with clean splits
        profile = {
            'company_overview': extract_section("Company Overview"),
            'leadership': extract_section("Leadership Information"),
            'founder_details': extract_section("Founder Details"),
            'financial_health': extract_section("Financial Health"),
            'operations': extract_section("Business Operations"),
            'mission_values': extract_section("Mission and Values"),
            'market_position': extract_section("Market Position"),
            'scraped_data': combined_data[:2000] + "..." if len(combined_data) >
2000 else combined_data,
            'financial_literacy': f"{founder_rating} - {rating_reason}",
            'source_urls': {
                'website': website_url,
                'founder_sources': founder_urls
            }
        }
```

```
        return profile

    except Exception as e:
        print(f"Profile generation error: {e}")
        return {
            'company_overview': 'Not available',
            'leadership': 'Not available',
            'founder_details': 'Not available',
            'financial_health': 'Not available',
            'operations': 'Not available',
            'mission_values': 'Not available',
            'market_position': 'Not available',
            'scraped_data': 'No scraped data available',
            'financial_literacy': 'Unknown',
            'source_urls': {}
        }


# ----------------------- COMPANY STANDING / FINANCIAL ANALYSIS (FINAL SWOT REWRITE) -----------------------

@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, company_profile=None):
    """Generates a very detailed, personalized financial and strategic analysis with mandatory founder integration in SWOT."""
    try:
        # Read onboarding info
        onboarding_text = read_onboarding_text()
        onboarding_summary = summarize_onboarding(onboarding_text)

        founder_rating = company_profile.get('financial_literacy', 'Unknown') if company_profile else "Unknown"
        founder_details = company_profile.get('founder_details', '') if company_profile else ""

        profile_context = ""
        if company_profile:
            profile_context = f"""
            Additional Company Context:
            - Company Overview: {company_profile.get('company_overview', 'Not available')}
            - Leadership: {company_profile.get('leadership', 'Not available')}
            - Founder Details: {founder_details}
            - Financial Health: {company_profile.get('financial_health', 'Not
```

```
available')}
        - Business Operations: {company_profile.get('operations', 'Not
available')}
        - Mission and Values: {company_profile.get('mission_values', 'Not
available')}
        - Market Position: {company_profile.get('market_position', 'Not
available')}
        - Founder Financial Literacy: {founder_rating}
        """

    prompt = f"""
    You are a senior financial consultant providing highly strategic and detailed
analysis.

    Based on the following:

    Company Financial Metrics:
    {json.dumps(company_statement, indent=2)}

    Industry Averages:
    {json.dumps(industry_averages, indent=2)}

    Market Report:
    {market_report[:2000]}...

    {profile_context}

    Onboarding Discussion:
    - Sentiment: {onboarding_summary.get('sentiment', 'Unknown')}
    - Key Topics: {', '.join([k for k, _ in
onboarding_summary.get('tags_detected', [])])}
    - Action Items: {', '.join(onboarding_summary.get('action_items_summary',
[]))}

    Please generate a detailed analysis:

    1. **Financial Performance Summary**:
       - Clear paragraph analyzing profitability, efficiency, cash flow.
       - Highlight areas outperforming or underperforming vs industry.

    2. **SWOT Analysis**:
       - **Strengths**: 5-6 clear bullet points, each explained with WHY it's a
strength.
         - ⚡ Mandatory: 1 strength about the founder's style (e.g., strategic/
operational thinking).
       - **Weaknesses**: 5-6 clear bullet points, with WHY each matters.
         - ⚡ If founder style is more weakness-driven (e.g., lack of strategic
```

```
focus), mention it here instead.
        - **Opportunities (Actionable Recommendations)**:
          - 5-7 detailed personalized growth strategies.
          - ⚡ Mandatory: 1 opportunity focused on how founder can evolve/improve for better scaling (e.g., "expand strategic focus through mentorship programs..." etc.)
        - **Threats**: External and internal risks outlined clearly.

      3. **Final Verdict**:
        - Short-term (6 months) and Long-term (2-3 years) business outlook.
        - Base it on financial + market + founder evaluation.

      **Important Rules**:
      - Use bullet points wherever appropriate.
      - Bold important financial terms like **Net Profit Margin**, **Operating Expense Ratio**, etc.
      - No generic advice.
      - No `<think>` or placeholder tags.
      """

      response = client×chat×completions×create(
        messages=[
          {"role": "system", "content": "You are an expert business analyst writing executive financial assessments."},
          {"role": "user", "content": prompt}
        ],
        model="deepseek-r1-distill-llama-70b",
        temperature=0.15,
        max_tokens=4000
      )

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

      # Clean any stray LLM tags
      analysis = re.sub(r'<think>.*?</think>', '', analysis, flags=re.DOTALL)

      return analysis

    except Exception as e:
      return f"Error generating company analysis: {e}"
```

Main.py

```
import streamlit as st
from PIL import Image
import pandas as pd
```

```
from financial_analysis import (
    generate_market_report_perplexity,
    generate_company_profile,
    upload_to_pinecone,
    get_industry_averages,
    compare_to_industry_average,
    analyze_company_standing
)
import os
import time

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

# ------------------------ CUSTOM STYLING ------------------------

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

st.markdown(f""" <style>
/* Global App 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};
}}

.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, textarea, select {{
    border: 1px solid #cccccc !important;
    border-radius: 5px;
    padding: 10px;
    font-size: 16px;
}}

input:focus, textarea:focus, select:focus {{
    outline: none !important;
    border-radius: 5px;
    padding: 10px;
    font-size: 16px;
}}

.stTabs [data-baseweb="tab"] {{
    height: 45px;
    background-color: #ffffff;
    border-radius: 8px 8px 0px 0px;
    font-weight: bold;
    padding: 12px;
    color: {text_color} !important;
    border-bottom: none;
}}

.stTabs [aria-selected="true"] {{
    color: {primary_color} !important;
    border-bottom: 3px solid {primary_color} !important;
}}

h1, h2, h3 {{
    color: {primary_color} !important;
}}

.finbot-image {{
    display: flex;
    justify-content: flex-end;
    align-items: flex-end;
    height: 100%;
}}
```

```
.sidebar-title {{
    color: white !important;
}}

.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)

# ---------------------- SESSION STATE INIT ----------------------

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
if 'profile_generated' not in st.session_state:
    st.session_state.profile_generated = False

# ---------------------- SIDEBAR ----------------------

try:
    main_logo = Image.open("images/main_logo.png")
    overview_image = Image×open("images/overview.jpeg")
    finbot_image = Image×open("images/finbot.jpg")
```

```
except:
    main_logo = None
    overview_image = None
    finbot_image = None

with st.sidebar:
    if main_logo:
        st.image(main_logo, use_container_width=True)
    st.markdown("""
    <h3 class="sidebar-title">mypocketCFO: Your Financial Companion</h3>
    <p>Empowering your financial journey with real-time insights and AI-driven
analysis.</p>
    <h4>Key Features:</h4>
    <ul>
      <li>Automated bookkeeping and reporting</li>
      <li>Cash flow forecasting and budgeting</li>
      <li>Personalized financial advice</li>
      <li>Integration with popular accounting systems</li>
    </ul>
    <p class="highlight">Join us on your journey to financial success!</p>
    """, unsafe_allow_html=True)

# ------------------------ TABS -----------------------

tab1, tab2, tab3 = st.tabs(['Overview', 'Profile', 'Analysis'])

# ------------------------ TAB 1: OVERVIEW -----------------------

with tab1:
    st.header("Overview", help="Market insights and basic company
information")
    col1, col2 = st.columns([2, 1])

    with col1:
        st.subheader("Company Information")
        company_name = st.text_input("Company Name",
                                     value=st.session_state.company_name,
                                     key="company_name_input",
                                     label_visibility="collapsed")
        if st.button("Generate Market Report"):
            if not company_name:
                st.session_state.show_prompt = True
                st.session_state.company_name = ""
                st.session_state.market_report = ""
                st.session_state.profile_generated = False
            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 style="color:red;">Please enter a company name.</
p>', unsafe_allow_html=True)

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

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

# ----------------------- TAB 2: PROFILE -----------------------

with tab2:
    st.header("Company Profile", help="Detailed company and founder profile")

    if st.session_state.company_name:
        if st.button("Generate Comprehensive Profile") or
st.session_state.profile_generated:
            if not st.session_state.profile_generated:
                with st.spinner("Generating company profile..."):
                    try:
                        profile_data =
generate_company_profile(st.session_state.company_name)
                        st.session_state.company_profile = profile_data
                        st.session_state.profile_generated = True
                    except Exception as e:
                        st.error(f"Profile generation error: {str(e)}")

            if st.session_state.company_profile:
                profile_data = st.session_state.company_profile
                st.subheader(f"Profile: {st.session_state.company_name}")
```

```
            # Show source URLs if available
            with st.expander("✔ Data Sources"):
                website = profile_data['source_urls'].get('website', "Not available")
                founders = profile_data['source_urls'].get('founder_sources', [])
                st.markdown(f"**Website:** {website}")
                if founders:
                    st.markdown("**Founder Sources:**")
                    for link in founders:
                        st.markdown(f"- {link}")

            # Correctly split the profile into mini-tabs
            profile_subtab_titles = [
                "Company Overview",
                "Leadership Information",
                "Founder Details",
                "Financial Health",
                "Business Operations",
                "Mission and Values",
                "Market Position"
            ]

            profile_subtab_contents = [
                profile_data.get('company_overview', 'Not available'),
                profile_data.get('leadership', 'Not available'),
                profile_data.get('founder_details', 'Not available'),
                profile_data.get('financial_health', 'Not available'),
                profile_data.get('operations', 'Not available'),
                profile_data.get('mission_values', 'Not available'),
                profile_data.get('market_position', 'Not available')
            ]

            profile_subtabs = st.tabs(profile_subtab_titles)

            for idx, subtab in enumerate(profile_subtabs):
                with subtab:
                    content = profile_subtab_contents[idx]
                    if content and content×strip()×lower() != "not available":
                        for line in content.split('\n'):
                            if line.strip():
                                st.markdown(f"- {line.strip()}")

                    # Special handling for Founder Literacy inside Founder Details
tab
                    if profile_subtab_titles[idx] == "Founder Details":
                        founder_literacy = profile_data.get('financial_literacy',
'Unknown')
```

```
                    st.markdown(f"- **Founder Financial Literacy:**
{founder_literacy}")

            # Downloadable Profile
            full_profile_text = f"""
            # {st.session_state.company_name} Profile

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

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

            ## Founder Details
            {profile_data.get('founder_details', '')}
            Financial Literacy: {profile_data.get('financial_literacy', '')}

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

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

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

            ## Market Position
            {profile_data.get('market_position', '')}
            """

            st.download_button(
                label="Download Full Profile",
                data=full_profile_text,
                file_name=f"{st.session_state.company_name}_profile.md",
                mime="text/markdown"
            )

        else:
            st.info("Please generate a Market Report in the Overview tab first.")
    else:
        st.info("Please enter a company name in the Overview tab first.")

# ----------------------- TAB 3: FINANCIAL ANALYSIS
-----------------------

with tab3:
    col1, col2 = st.columns([3, 1])
```

```
    with col1:
        st.header("Financial Analysis", help="In-depth AI-driven financial and
strategic evaluation")
        st.write("*Analyze your company's financial vitals and get personalized
growth strategies.*")

        company_file = st.file_uploader("Upload your company's Income
Statement CSV", type="csv")

        process_data = st.button("Process Data and Generate Analysis")

    with col2:
        if finbot_image:
            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 style="color:red;">Please enter a company name
first.</p>', unsafe_allow_html=True)
        elif not company_file:
            st.markdown('<p style="color:red;">Please upload your company\'s
Income Statement CSV file.</p>', unsafe_allow_html=True)
        else:
            with st.spinner("Processing uploaded data and generating detailed
financial analysis..."):
                try:
                    # Upload benchmark files to Pinecone
                    industry_files = ["data/income_statement1.csv", "data/
income_statement2.csv"]
                    upload_to_pinecone(industry_files)
                    time.sleep(2)

                    # Fetch industry averages
                    industry_averages = get_industry_averages()

                    if not industry_averages:
                        st.error("Failed to fetch industry benchmarks. Please retry.")
                    else:
                        # Compare uploaded company file
                        comparison_results =
compare_to_industry_average(company_file, industry_averages)

                        st.subheader("Comparison to Industry Averages")
                        comparison_data = []
```

```
                for metric, values in comparison_results.items():
                    comparison_data.append([
                        metric,
                        values['Company Value'],
                        values['Industry Average'],
                        values['Verdict']
                    ])
                df_comparison = pd.DataFrame(comparison_data,
columns=['Metric', 'Company Value', 'Industry Average', 'Verdict'])
                st.dataframe(df_comparison)

                # Generate full company analysis
                st.subheader("Comprehensive Company Analysis")

                company_statement = df_comparison.to_dict()

                market_report = st.session_state.market_report
                company_profile = st.session_state.company_profile if
st.session_state.profile_generated else None

                analysis = analyze_company_standing(
                    company_statement,
                    industry_averages,
                    market_report,
                    st.session_state.company_name,
                    company_profile
                )

                # Clean any stray tags if any
                analysis = analysis.replace('<think>', '').replace('</think>', '')

                st.markdown(analysis)

            except Exception as e:
                st.error(f"Error processing financial analysis: {str(e)}")
```