```
import streamlit as st
from google.generativeai.generative_models import GenerativeModel
from google.generativeai.client import configure
import chromadb
from chromadb.utils import embedding_functions
import pandas as pd
import numpy as np
import json
import requests
from datetime import datetime, timedelta
import time
import re
from typing import Dict, List, Any, Optional
import pickle
import os
from dataclasses import dataclass, asdict
from bs4 import BeautifulSoup
import feedparser
import yfinance as yf
import plotly.express as px
import plotly.graph_objects as go
from sentence_transformers import SentenceTransformer
import asyncio
import aiohttp
from urllib.parse import urljoin, urlparse
import logging

# Configure logging
```

```
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Configure page
st.set_page_config(
    page_title="CogniScout - Adaptive Market Intelligence Agent",
    page_icon="🔍",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Data classes for structured information
@dataclass
class MarketInsight:
    timestamp: datetime
    query: str
    findings: List[str]
    sources: List[str]
    confidence_score: float
    market_segment: str
    trend_direction: str

@dataclass
class UserProfile:
    preferences: Dict[str, Any]
    research_history: List[str]
    feedback_scores: List[float]
    market_segments: List[str]
```

```
    last_updated: datetime

class CogniScoutAgent:
    def __init__(self):
        self.setup_components()
        self.load_user_profile()

    def setup_components(self):
        """Initialize all components of the agent"""
        # Initialize Gemini
        if 'gemini_api_key' in st.secrets:
            configure(api_key=st.secrets['AIzaSyDXV6cAVahejSTL7WOj3bzhvUlWhKyzQ5U'])
        else:
            st.error("Please configure Gemini API key in secrets")
            st.stop()

        self.model = GenerativeModel('gemini-1.5-flash')

        # Initialize ChromaDB for persistent memory
        self.chroma_client = chromadb.PersistentClient(path="./chroma_db")
        # Use the default embedding function compatible with ChromaDB
        self.embedding_function = embedding_functions.DefaultEmbeddingFunction()

        # Create collections for different types of memory
        try:
            self.insights_collection = self.chroma_client.create_collection(
                name="market_insights"
            )
```

```
        except:
            self.insights_collection = self.chroma_client.get_collection(
                name="market_insights"
            )

        try:
            self.user_profile_collection = self.chroma_client.create_collection(
                name="user_profiles"
            )
        except:
            self.user_profile_collection = self.chroma_client.get_collection(
                name="user_profiles"
            )
        # Initialize sentence transformer for embeddings
        self.sentence_transformer = SentenceTransformer('all-MiniLM-L6-v2')

    def load_user_profile(self):
        """Load or create user profile"""
        try:
            results = self.user_profile_collection.get(ids=["default_user"])
            if results['documents']:
                self.user_profile = UserProfile(**json.loads(results['documents'][0]))
            else:
                self.create_default_profile()
        except:
            self.create_default_profile()

    def create_default_profile(self):
```

```
        """Create default user profile"""
        self.user_profile = UserProfile(
            preferences={},
            research_history=[],
            feedback_scores=[],
            market_segments=[],
            last_updated=datetime.now()
        )
        self.save_user_profile()

    def save_user_profile(self):
        """Save user profile to ChromaDB"""
        profile_json = json.dumps(asdict(self.user_profile), default=str)
        self.user_profile_collection.upsert(
            documents=[profile_json],
            ids=["default_user"],
            metadatas=[{"last_updated": str(datetime.now())}]
        )

    def plan_research_strategy(self, query: str) -> Dict[str, Any]:
        """Plan and execute research strategy using Gemini"""
        planning_prompt = f"""
        As CogniScout, an adaptive market intelligence agent, create a comprehensive research plan for: "{query}"

        Break this down into specific sub-tasks:
        1. Key entities and trends to identify
        2. Specific data points to search for
```

```
3. Competitive landscape analysis requirements
4. Market segment focus areas
5. Synthesis approach

Consider the user's research history: {self.user_profile.research_history[-5:]}

Return a structured JSON plan with tasks, data sources, and success criteria.
"""

try:
    response = self.model.generate_content(planning_prompt)

    # Parse the response to extract structured plan
    plan_text = response.text

    # Extract key components using regex and NLP
    entities = self.extract_entities(plan_text)
    data_sources = self.identify_data_sources(plan_text)
    market_segments = self.identify_market_segments(plan_text)

    return {
        "query": query,
        "entities": entities,
        "data_sources": data_sources,
        "market_segments": market_segments,
        "plan_text": plan_text,
        "timestamp": datetime.now()
    }
```

```
        except Exception as e:
            logger.error(f"Error in planning: {e}")
            return {"error": str(e)}

    def extract_entities(self, text: str) -> List[str]:
        """Extract key entities from text"""
        # Simple entity extraction - can be enhanced with NER
        entities = []
        patterns = [
            r'\b[A-Z][a-z]+(?:\s+[A-Z][a-z]+)*\b',  # Proper nouns
            r'\b(?:AI|ML|IoT|5G|blockchain|cryptocurrency)\b',  # Tech terms
            r'\b\d{4}\b',  # Years
        ]

        for pattern in patterns:
            matches = re.findall(pattern, text, re.IGNORECASE)
            entities.extend(matches)

        return list(set(entities))

    def identify_data_sources(self, text: str) -> List[str]:
        """Identify relevant data sources from plan"""
        sources = []
        source_keywords = {
            'news': ['news', 'articles', 'press release'],
            'financial': ['financial', 'earnings', 'revenue', 'stock'],
            'social': ['social media', 'twitter', 'linkedin'],
```

```
        'academic': ['research', 'papers', 'studies'],
        'industry': ['industry report', 'market analysis', 'competitor']
    }

    text_lower = text.lower()
    for source_type, keywords in source_keywords.items():
        if any(keyword in text_lower for keyword in keywords):
            sources.append(source_type)

    return sources

def identify_market_segments(self, text: str) -> List[str]:
    """Identify market segments from plan"""
    segments = []
    segment_keywords = {
        'technology': ['tech', 'software', 'hardware', 'digital'],
        'healthcare': ['health', 'medical', 'pharma', 'biotech'],
        'finance': ['finance', 'banking', 'fintech', 'investment'],
        'retail': ['retail', 'consumer', 'e-commerce', 'shopping'],
        'automotive': ['automotive', 'cars', 'transportation', 'mobility']
    }

    text_lower = text.lower()
    for segment, keywords in segment_keywords.items():
        if any(keyword in text_lower for keyword in keywords):
            segments.append(segment)

    return segments
```

```
def execute_research(self, plan: Dict[str, Any]) -> Dict[str, Any]:
    """Execute research plan with multiple data sources"""
    results = {
        'news_data': [],
        'financial_data': [],
        'social_data': [],
        'academic_data': [],
        'synthesis': '',
        'confidence_score': 0.0
    }

    # Execute research for each data source
    for source in plan.get('data_sources', []):
        if source == 'news':
            results['news_data'] = self.gather_news_data(plan['query'])
        elif source == 'financial':
            results['financial_data'] = self.gather_financial_data(plan['entities'])
        elif source == 'social':
            results['social_data'] = self.gather_social_data(plan['query'])
        elif source == 'academic':
            results['academic_data'] = self.gather_academic_data(plan['query'])

    # Synthesize findings
    results['synthesis'] = self.synthesize_findings(results, plan)
    results['confidence_score'] = self.calculate_confidence_score(results)

    return results
```

```
def gather_news_data(self, query: str) -> List[Dict[str, Any]]:
    """Gather news data from RSS feeds and APIs"""
    news_data = []

    # Sample RSS feeds - replace with actual news APIs
    rss_feeds = [
        'https://feeds.feedburner.com/TechCrunch',
        'https://rss.cnn.com/rss/edition.rss',
        'https://feeds.bbci.co.uk/news/business/rss.xml'
    ]

    for feed_url in rss_feeds:
        try:
            feed = feedparser.parse(feed_url)
            for entry in feed.entries[:5]:  # Limit to 5 entries per feed
                title = str(entry.title) if hasattr(entry, 'title') else ""
                summary = str(entry.summary) if hasattr(entry, 'summary') else ""
                if any(keyword.lower() in title.lower() or
                       keyword.lower() in summary.lower()
                       for keyword in query.split()):
                    news_data.append({
                        'title': title,
                        'summary': summary,
                        'link': entry.link,
                        'published': entry.published,
                        'source': getattr(feed.feed, 'title', 'Unknown Source')
                    })
```

```
            except Exception as e:
                logger.error(f"Error fetching news from {feed_url}: {e}")

        return news_data

    def gather_financial_data(self, entities: List[str]) -> List[Dict[str, Any]]:
        """Gather financial data for relevant entities"""
        financial_data = []

        # Extract potential stock symbols
        stock_symbols = [entity.upper() for entity in entities if len(entity) <= 5]

        for symbol in stock_symbols:
            try:
                ticker = yf.Ticker(symbol)
                info = ticker.info

                if info:
                    financial_data.append({
                        'symbol': symbol,
                        'company_name': info.get('longName', ''),
                        'market_cap': info.get('marketCap', 0),
                        'pe_ratio': info.get('trailingPE', 0),
                        'revenue': info.get('totalRevenue', 0),
                        'sector': info.get('sector', ''),
                        'industry': info.get('industry', '')
                    })
            except Exception as e:
```

```
            logger.error(f"Error fetching financial data for {symbol}: {e}")

    return financial_data

def gather_social_data(self, query: str) -> List[Dict[str, Any]]:
    """Simulate social media data gathering"""
    # In a real implementation, this would use Twitter API, LinkedIn API, etc.
    social_data = [
        {
            'platform': 'Twitter',
            'content': f'Trending discussions about {query}',
            'sentiment': 'positive',
            'engagement': 1250,
            'timestamp': datetime.now()
        },
        {
            'platform': 'LinkedIn',
            'content': f'Professional insights on {query}',
            'sentiment': 'neutral',
            'engagement': 850,
            'timestamp': datetime.now()
        }
    ]

    return social_data

def gather_academic_data(self, query: str) -> List[Dict[str, Any]]:
    """Simulate academic research data gathering"""
```

```
    # In a real implementation, this would use arXiv API, Google Scholar, etc.
    academic_data = [
        {
            'title': f'Recent Research on {query}',
            'authors': ['Dr. Smith', 'Prof. Johnson'],
            'abstract': f'This paper explores the implications of {query} in modern markets.',
            'publication_date': '2024-01-15',
            'journal': 'Journal of Market Intelligence',
            'citations': 45
        }
    ]

    return academic_data

def synthesize_findings(self, results: Dict[str, Any], plan: Dict[str, Any]) -> str:
    """Synthesize all findings into coherent insights using Gemini"""
    synthesis_prompt = f"""
    As CogniScout, synthesize the following market intelligence data into actionable insights:

    Query: {plan['query']}

    News Data: {json.dumps(results['news_data'], indent=2)}
    Financial Data: {json.dumps(results['financial_data'], indent=2)}
    Social Data: {json.dumps(results['social_data'], indent=2)}
    Academic Data: {json.dumps(results['academic_data'], indent=2)}

    Provide:
    1. Key findings and trends
```

```
    2. Market opportunities and threats
    3. Competitive landscape insights
    4. Strategic recommendations
    5. Risk assessment

    Format as a comprehensive market intelligence report.
    """

    try:
        response = self.model.generate_content(synthesis_prompt)
        return response.text
    except Exception as e:
        logger.error(f"Error in synthesis: {e}")
        return f"Error synthesizing findings: {e}"

def calculate_confidence_score(self, results: Dict[str, Any]) -> float:
    """Calculate confidence score based on data quality and quantity"""
    score = 0.0

    # Score based on data availability
    if results['news_data']:
        score += 0.3
    if results['financial_data']:
        score += 0.3
    if results['social_data']:
        score += 0.2
    if results['academic_data']:
        score += 0.2
```

```
    # Adjust based on data quality
    total_sources = len(results['news_data']) + len(results['financial_data']) + \
                    len(results['social_data']) + len(results['academic_data'])

    if total_sources > 10:
        score *= 1.2
    elif total_sources < 3:
        score *= 0.8

    return min(score, 1.0)

def save_insight(self, insight: MarketInsight):
    """Save market insight to persistent memory"""
    insight_json = json.dumps(asdict(insight), default=str)

    self.insights_collection.add(
        documents=[insight_json],
        ids=[f"insight_{datetime.now().strftime('%Y%m%d_%H%M%S')}"],
        metadatas=[{
            "timestamp": str(insight.timestamp),
            "market_segment": insight.market_segment,
            "confidence_score": insight.confidence_score
        }]
    )

def retrieve_similar_insights(self, query: str, limit: int = 5) -> List[Dict[str, Any]]:
    """Retrieve similar past insights"""
```

```
    try:
        results = self.insights_collection.query(
            query_texts=[query],
            n_results=limit
        )

        insights = []
        documents = results.get('documents')
        distances = results.get('distances')
        if documents and documents[0]:
            for i, doc in enumerate(documents[0]):
                insight_data = json.loads(doc)
                similarity_score = distances[0][i] if distances and distances[0] else None
                insights.append({
                    'insight': insight_data,
                    'similarity_score': similarity_score
                })
        return insights
    except Exception as e:
        logger.error(f"Error retrieving insights: {e}")
        return []

def update_user_profile(self, query: str, feedback_score: Optional[float] = None):
    """Update user profile based on interaction"""
    self.user_profile.research_history.append(query)
    if feedback_score:
        self.user_profile.feedback_scores.append(feedback_score)
    self.user_profile.last_updated = datetime.now()
```

```
    self.save_user_profile()

def self_correction(self, results: Dict[str, Any], plan: Dict[str, Any]) -> Dict[str, Any]:
    """Self-correction mechanism to improve results"""
    if results['confidence_score'] < 0.5:
        # Re-plan with different approach
        correction_prompt = f"""
        The initial research plan for "{plan['query']}" yielded low confidence results.

        Current confidence score: {results['confidence_score']}

        Suggest an improved research strategy with:
        1. Alternative data sources
        2. Different search terms
        3. Modified analysis approach

        Focus on increasing data quality and relevance.
        """

        try:
            response = self.model.generate_content(correction_prompt)
            # Re-execute with improved strategy
            improved_plan = self.plan_research_strategy(f"{plan['query']} {response.text}")
            return self.execute_research(improved_plan)
        except Exception as e:
            logger.error(f"Error in self-correction: {e}")

    return results
```

```
# Initialize the agent
@st.cache_resource
def get_agent():
    return CogniScoutAgent()

# Streamlit UI
def main():
    st.title("🔍 CogniScout: Adaptive Market Intelligence Agent")
    st.markdown("*Advanced AI-powered market research with persistent memory and adaptive learning*")

    # Initialize agent
    agent = get_agent()

    # Sidebar
    with st.sidebar:
        st.header("Agent Status")

        # User profile info
        st.subheader("Your Profile")
        st.metric("Research Queries", len(agent.user_profile.research_history))
        st.metric("Avg Feedback Score",
                  f"{np.mean(agent.user_profile.feedback_scores):.2f}" if agent.user_profile.feedback_scores else "N/A")

        # Recent queries
        if agent.user_profile.research_history:
```

```
        st.subheader("Recent Queries")
        for query in agent.user_profile.research_history[-3:]:
            st.text(f"• {query[:50]}...")

    # Settings
    st.subheader("Settings")
    auto_correction = st.checkbox("Enable Self-Correction", value=True)
    confidence_threshold = st.slider("Confidence Threshold", 0.0, 1.0, 0.5)

# Main interface
col1, col2 = st.columns([2, 1])

with col1:
    st.header("Market Intelligence Query")

    # Query input
    query = st.text_area(
        "Enter your market research question:",
        placeholder="e.g., What are the emerging trends in AI-powered fintech solutions?",
        height=100
    )

    # Quick query examples
    st.subheader("Quick Examples")
    example_queries = [
        "Competitive analysis of electric vehicle market",
        "Emerging trends in remote work technology",
        "Investment opportunities in sustainable energy",
```

```
    "Consumer sentiment towards cryptocurrency",
    "Impact of AI on healthcare industry"
]

selected_example = st.selectbox("Or select an example:", [""] + example_queries)
if selected_example:
    query = selected_example

# Execute research
if st.button("🚀 Execute Research", type="primary"):
    if query:
        with st.spinner("Planning research strategy..."):
            # Step 1: Plan
            plan = agent.plan_research_strategy(query)

            if 'error' in plan:
                st.error(f"Planning error: {plan['error']}")
            else:
                st.success("Research plan created!")

                # Show plan details
                with st.expander("View Research Plan"):
                    st.json(plan)

                # Step 2: Execute
                with st.spinner("Executing research..."):
                    results = agent.execute_research(plan)
```

```
                # Step 3: Self-correction if needed
                if auto_correction and results['confidence_score'] < confidence_threshold:
                    with st.spinner("Applying self-correction..."):
                        results = agent.self_correction(results, plan)

                # Step 4: Display results
                st.header("🎯 Market Intelligence Report")

                # Confidence score
                confidence_color = "green" if results['confidence_score'] > 0.7 else "orange" if results['confidence_score'] > 0.4 else "red"
                st.markdown(f"**Confidence Score:** :{confidence_color}[{results['confidence_score']:.2f}]")

                # Synthesis
                st.subheader("Key Insights & Synthesis")
                st.markdown(results['synthesis'])

                # Data sources
                st.subheader("Data Sources")
                tabs = st.tabs(["News", "Financial", "Social", "Academic"])

                with tabs[0]:
                    if results['news_data']:
                        for item in results['news_data']:
                            st.write(f"**{item['title']}**")
                            st.write(f"Source: {item['source']}")
                            st.write(f"Summary: {item['summary']}")
```

```
                    st.write(f"[Read More]({item['link']})")
                    st.divider()
            else:
                st.info("No news data available for this query.")

        with tabs[1]:
            if results['financial_data']:
                df = pd.DataFrame(results['financial_data'])
                st.dataframe(df)
            else:
                st.info("No financial data available for this query.")

        with tabs[2]:
            if results['social_data']:
                for item in results['social_data']:
                    st.write(f"**{item['platform']}**: {item['content']}")
                    st.write(f"Sentiment: {item['sentiment']} | Engagement:
{item['engagement']}")
                    st.divider()
            else:
                st.info("No social media data available for this query.")

        with tabs[3]:
            if results['academic_data']:
                for item in results['academic_data']:
                    st.write(f"**{item['title']}**")
                    st.write(f"Authors: {', '.join(item['authors'])}")
                    st.write(f"Abstract: {item['abstract']}")
```

```
                        st.write(f"Journal: {item['journal']} | Citations: {item['citations']}")
                        st.divider()
                else:
                    st.info("No academic data available for this query.")

            # Save insight
            insight = MarketInsight(
                timestamp=datetime.now(),
                query=query,
                findings=[results['synthesis']],
                sources=list(results.keys()),
                confidence_score=results['confidence_score'],
                market_segment=plan.get('market_segments', ['general'])[0] if plan.get('market_segments') else 'general',
                trend_direction='neutral'
            )
            agent.save_insight(insight)

            # Update user profile
            agent.update_user_profile(query)

            # Feedback
            st.subheader("Feedback")
            feedback = st.slider("Rate this research quality:", 1, 5, 3)
            if st.button("Submit Feedback"):
                agent.update_user_profile(query, feedback)
                st.success("Feedback submitted!")
    else:
```

```
            st.error("Please enter a research query.")

with col2:
    st.header("Similar Past Insights")

    if query:
        similar_insights = agent.retrieve_similar_insights(query)

        if similar_insights:
            for insight_data in similar_insights:
                insight = insight_data['insight']
                st.write(f"**Query:** {insight['query']}")
                st.write(f"**Confidence:** {insight['confidence_score']:.2f}")
                st.write(f"**Date:** {insight['timestamp'][:10]}")
                st.write(f"**Segment:** {insight['market_segment']}")
                st.divider()
        else:
            st.info("No similar insights found.")
    else:
        st.info("Enter a query to see similar past insights.")

    # Analytics
    st.header("Analytics")

    if agent.user_profile.feedback_scores:
        # Feedback trend
        fig = px.line(
            x=range(len(agent.user_profile.feedback_scores)),
```

```
            y=agent.user_profile.feedback_scores,
            title="Feedback Trend"
        )
        st.plotly_chart(fig, use_container_width=True)

    # Research frequency
    if agent.user_profile.research_history:
        st.metric("Total Researches", len(agent.user_profile.research_history))
        st.metric("This Week", len([q for q in agent.user_profile.research_history if 'week' in q.lower()]))  # Simplified

if __name__ == "__main__":
    main()
```