```
import os
import requests
from datetime import datetime, timedelta
from anthropic import Anthropic
import time
import json

class KalshiAlphaBot:
    def __init__(self):
        self.anthropic_key = os.getenv('ANTHROPIC_API_KEY')
        self.claude = Anthropic(api_key=self.anthropic_key)
        self.kalshi_api = "https://api.elections.kalshi.com/trade-api/v2"
        self.kalshi_api_key = os.getenv('KALSHI_API_KEY')

    def get_headers(self):
        return {
            'Authorization': f'Bearer {self.kalshi_api_key}',
            'Content-Type': 'application/json'
        }

    def get_active_markets(self):
        try:
            url = f"{self.kalshi_api}/markets"
            params = {'limit': 200, 'status': 'open'}
            response = requests.get(url, headers=self.get_headers(), params=param
            response.raise_for_status()
            data = response.json()
            markets = data.get('markets', [])
            print(f"Found {len(markets)} active markets")
            return markets
        except Exception as e:
            print(f"Error fetching markets: {e}")
            return []

    def calculate_alpha_score(self, market):
        score = 0
        category = market.get('category', '').lower()
        ticker = market.get('ticker', '').lower()
        title = market.get('title', '').lower()

        allowed_keywords = [
            'fed', 'federal reserve', 'interest rate', 'fomc',
            'inflation', 'cpi', 'pce',
            'unemployment', 'jobs report', 'nonfarm',
```

```
            'gdp', 'recession',
            'election', 'senate', 'congress', 'house', 'vote', 'bill',
            'president', 'trump', 'biden',
            'treasury', 'tax', 'tariff',
            'supreme court', 'scotus'
        ]

        if not any(keyword in title or keyword in category for keyword in allowed
            return 0

        score += 3

        volume = float(market.get('volume', 0))
        if volume > 50000:
            score += 3
        elif volume > 10000:
            score += 2
        elif volume > 5000:
            score += 1

        open_interest = float(market.get('open_interest', 0))
        if open_interest > 50000:
            score += 2
        elif open_interest > 10000:
            score += 1

        close_time = market.get('close_time')
        if close_time:
            try:
                close_date = datetime.fromisoformat(close_time.replace('Z', '+00:
                days_until = (close_date - datetime.now().astimezone()).days
                if 3 <= days_until <= 90:
                    score += 2
                elif days_until < 3:
                    score -= 1
            except:
                pass

        yes_price = market.get('yes_bid', 0)
        yes_prob = yes_price / 100 if yes_price else 0.5
        if 0.20 <= yes_prob <= 0.80:
            score += 2

        return score

    def filter_high_alpha_markets(self, markets, min_score=2):
        scored_markets = []
```

```
    for market in markets:
        score = self.calculate_alpha_score(market)
        if score >= min_score:
            market['alpha_score'] = score
            scored_markets.append(market)
    scored_markets.sort(key=lambda x: x['alpha_score'], reverse=True)
    print(f"Filtered to {len(scored_markets)} high-alpha markets")
    return scored_markets

def research_and_validate(self, market):
    title = market.get('title', '')
    ticker = market.get('ticker', '')
    category = market.get('category', '')
    yes_price = market.get('yes_bid', 0)
    no_price = market.get('no_bid', 0)
    volume = market.get('volume', 0)
    close_time = market.get('close_time', '')

    close_date = "Unknown"
    try:
        close_dt = datetime.fromisoformat(close_time.replace('Z', '+00:00'))
        close_date = close_dt.strftime('%B %d, %Y')
    except:
        pass

    yes_prob = yes_price / 100 if yes_price else 0
    no_prob = no_price / 100 if no_price else 0

    prompt = f"Research this Kalshi market: {title}. Category: {category}. Cu

    try:
        message = self.claude.messages.create(
            model="claude-sonnet-4-20250514",
            max_tokens=2000,
            tools=[{"type": "web_search_20250305", "name": "web_search"}],
            messages=[{"role": "user", "content": prompt}]
        )

        full_response = ""
        for block in message.content:
            if hasattr(block, 'text'):
                full_response += block.text

        recommendation = "PASS"
        if "BUY YES" in full_response.upper():
            recommendation = "BUY YES"
        elif "BUY NO" in full_response.upper():
```

```
            recommendation = "BUY NO"

        confidence = "UNKNOWN"
        if "HIGH" in full_response.upper():
            confidence = "HIGH"
        elif "MEDIUM" in full_response.upper():
            confidence = "MEDIUM"
        elif "LOW" in full_response.upper():
            confidence = "LOW"

        return {'analysis': full_response, 'recommendation': recommendation,
    except Exception as e:
        print(f"Error with analysis: {e}")
        return {'analysis': f"Error: {str(e)}", 'recommendation': "PASS", 'co

def run_daily_scan(self, top_n=5, min_alpha_score=2):
    print(f"\nKALSHI ALPHA BOT")
    print(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")

    print("Fetching active markets...")
    markets = self.get_active_markets()
    if not markets:
        print("No markets found")
        return []

    print(f"\nFiltering for alpha opportunities...")
    high_alpha = self.filter_high_alpha_markets(markets, min_score=min_alpha_

    if not high_alpha:
        print(f"No markets with alpha score >= {min_alpha_score}")
        return []

    top_markets = high_alpha[:top_n]
    print(f"\nResearching top {len(top_markets)} markets...\n")

    results = []
    for i, market in enumerate(top_markets, 1):
        title = market.get('title', 'Unknown')[:70]
        print(f"[{i}/{len(top_markets)}] {title}...")

        analysis = self.research_and_validate(market)

        if not analysis['success']:
            print(f"  Analysis failed")
            continue

        if analysis['recommendation'] != 'PASS':
```

```
                print(f"  {analysis['recommendation']} | {analysis['confidence']}
                results.append({'market': market, 'analysis': analysis})
            else:
                print(f"  PASS")

            if i < len(top_markets):
                time.sleep(20)

        print(f"\nGenerating report...")
        self.generate_report(results)
        return results

    def generate_report(self, results):
        report = []
        report.append("="*80)
        report.append(f"KALSHI ALPHA TRADING REPORT")
        report.append(f"{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
        report.append("="*80)
        report.append(f"\nFound {len(results)} actionable opportunities\n")

        if len(results) == 0:
            report.append("No BUY recommendations today.")

        for i, result in enumerate(results, 1):
            market = result['market']
            analysis = result['analysis']

            title = market.get('title', 'N/A')
            ticker = market.get('ticker', 'N/A')
            yes_price = market.get('yes_bid', 0)
            no_price = market.get('no_bid', 0)
            volume = market.get('volume', 0)

            report.append(f"\nOPPORTUNITY #{i}: {analysis['recommendation']}")
            report.append(f"Confidence: {analysis['confidence']}")
            report.append(f"\nMARKET: {title}")
            report.append(f"Ticker: {ticker}")
            report.append(f"\nCURRENT ODDS:")
            report.append(f"  YES: {yes_price/100:.1%}")
            report.append(f"  NO: {no_price/100:.1%}")
            report.append(f"\nVolume: ${volume:,.0f}")
            report.append(f"\nANALYSIS:")
            report.append(analysis['analysis'])
            report.append("\n" + "-"*80)

        report_text = "\n".join(report)
        print(f"\n{report_text}")
```

```
        return report_text


def main():
    bot = KalshiAlphaBot()
    results = bot.run_daily_scan(top_n=5, min_alpha_score=2)
    print(f"\nScan complete! Found {len(results)} actionable plays.")
    return len(results)


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