AI Data Collection Agent Assignment

Data Science Project Report

Ewan Pedersen

September 30, 2025

Contents

1 Part 1: Your Scenario (20 points)

1.1 Main Objective

Main Objective: Collect prediction market data from Kalshi API to analyze market sentiment and forecasting accuracy across various event categories.

1.2 Data Sources

- **Kalshi API**: `https://kalshi.com` - A regulated prediction market platform providing real-time market data for various events including elections, economics, and sports.

1.3 Data Types

The following types of data will be collected from the Kalshi API:

- Market prices (bid/ask spreads, last traded price)
- Trading volume (number of contracts traded)
- Event outcomes (binary yes/no predictions)
- Probability predictions (implied probabilities from market prices)
- Market metadata (event titles, categories, settlement dates)
- JSON formatted responses containing structured market data

1.4 Geographic Scope

The data collection focuses on US-centric prediction markets, including:

- US elections (federal, state, and local)
- US economic indicators (GDP, unemployment, inflation)
- US sports events and outcomes
- US political events and policy decisions

1.5 Time Range

- **Current market data**: Real-time and recent market snapshots
- **Historical outcomes**: Past settled markets for accuracy analysis
- **Collection period**: October 2024 to present
- **Focus**: Active markets with upcoming settlement dates and recently settled markets for validation

2 Part 2: Learning about APIs (15 points)

2.1 API Learning Reflection

They are **asynchronous**, and as such I can no longer consider my code in that linear, deterministic way. I have to think about the fact that the API may not respond, or may respond slowly, and I have to code around that. Not only that, but oftentimes it is through a RESTful interface, meaning I must use the http protocol to communicate with it.

3 Part 3: Setting Up Free API Access (10 points)

3.1 API Key Creation

Created the API key by signing up on the Kalshi platform, navigating to the developer section, and generating a new API key for accessing market data.

My test script is available under `assignment-scripts/kalshi-api-test.py`

3.2 Screenshot: Successful API Access

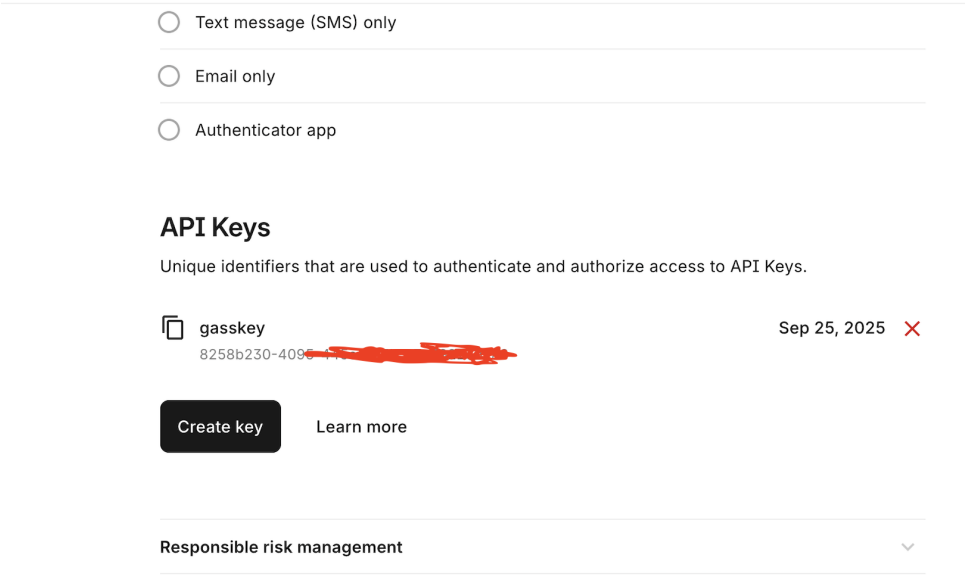

Figure 1: Screenshot showing successful API key creation

3.3 Config File Template

Kalshi uses both a environment variable and a special config file for the private key.

The env file looks like:

```
KALSHI-API-Key=your_api_key_here
```

And the private key config file looks like(this is scrambled):

```
-----BEGIN RSA PRIVATE KEY-----
MjIEogIjAAKCAjEArVaAnsBhK0QEjdDHz5qtYR8zoIMVHFe4xQLwKsX9K3cXruhY
kfyaqOmAWnUfJHEdwjJOPK61zd8LaafUdpnNY9EklQC9+GM4XyyzgMsUaerHRaY/
AEy0v2v6C2M5NfBWx96zGlztvwCmy8rnbLOBMBk29IXwvE4oxEQCJfu0FukfFitC
q4gWmavYs6ObUnbvoi9TzLhrgVEx3XsamzowiJrEnKlcmfHKtps1iTVyCdqSj/qV
yajfpOXsBYJhmemZZUF/jQtvpHsIait7adl0XWvCAGrVVrb4MuVo47RnrTewgE+y
5uQiGNoM8aijWjFkd1hXFa4ZikgM7I2ENrvDXQIDAQABAoIBAETzFMSRN9Wy/9CQ
HPsKxw7MD4g4vCwDBN9xNMF+8h7ZB5WunJShi4PHJiRVVGb1i+XUhuSChpT2tR55
AP2QXm/nIIsYuXhVV3YcXObNYThkiJG++G6RFJDUTak79eTrzklA37OL5ZJFmxOT
iozmtmvAi+imYsU0veicONKNFezhdGiHsvwrfLJCGpFGA7nmdYAo4xes6Hzs9ZO7
WFpWxzHnzhTrjx8InXGrfx1K+Bboecbo55pg10fk4PIig136t025RCShk+Bihh6S
BkHMRHYV3YZZq65kgj9hxIMfnU51WmR/+jSi3ur0AHBjpOrHV2qiGX3h5dgPo389
hk6MfgOCgYEAw+Qutcoubs/e2zgioy38Er0GEIWS/kHCwOav2QcV9ct2FXteEVMz
dsukT8QSqYGfmM/ujJHaNAn9L1ufJ4MH8WYn43rJNO0lcSQsk1rmqeuYneQwn6x8
CXEQi5NfnOLv9ju4ZmtRHfAebghyFILJ5xDgRplQ0EvttlyldUHDubsCgYEA4oai
C4qGsRIeLLGY0oNC+4wjY047NvPAL1biZDedhaFGHKBhRrVKdLHK3cnUPQx7WXF1
voiNtrlcE1Xr2pWKSz3LtYJTCU/HDBypWwOe5SvkhsVHQp4ch19LweWsh8v4+1PW
hiaskzDEOqdgyKWdErcMMH/dyOKyfNVkq98XeccCgYAFCVfeGq2o8bVEI6RYV7VU
cgG26QhHez9boFLDHLiG0k1d7gcUIPSAyGjYF213aDrHPglfGdznW55j2lI0PN02
aq753U1RDAvbZdezjjHCCqmj3ZYWdOLdaUfInuhlazxKM2zhTT8Pqv3NTHBKcLZq
N3fcuNJoGJurBaTHaAJ1QwKBgBO1cClq5gkl8GYnFYrLTNqh+Lo9Jj3Du2bjAlOB
2zyzKWx4MZI5V78LcQEW/FZLbbf8fSgUrW/GdNJh2WFtPVkKiVzDeAIJZOIS5FE8
4WhDB2EKbQbIxuaoTekS2I4AGyuSwcluN/medpQ719ndPutP7cmSdnNx9eGSl220
vVAlAoGAC4Z5gZQRr80yRzn5LdK7E7U1PIJBXetxiAgekeI1uy3Gw3N6IwcrNXXl
1mXo+vyijxhiPpdpgDf+NyHvb8Z7qqGUJA/CLMMasd2ViX8Erfx0XWl1mBmC30JD
ZoF56Z4Eseb4q7YbJspNEEmFXJxsPmqlrapM5oxQ7cgyxmZubqE=
-----END RSA PRIVATE KEY-----
```

4 Part 4: Build Your AI Data Collection Agent (35 points)

4.1 Agent Architecture Overview

The `KalshiDataAgent` collects prediction market data from Kalshi's API. It has six core components:

1. **Configuration Management** - Loads parameters from JSON files or uses defaults
2. **Intelligent Collection Strategy** - Adaptive loop that monitors performance
3. **Data Quality Assessment** - Evaluates completeness, accuracy, consistency, timeliness
4. **Adaptive Strategy** - Adjusts collection based on success rates
5. **Respectful Rate Limiting** - Delays with jitter to avoid overwhelming the API
6. **Automated Documentation** - Generates metadata, quality reports, recommendations

4.2 Key Features

- **Configuration-Driven**: Loads from JSON files or environment variables
- **Error Handling**: Try-catch blocks with detailed logging
- **Data Validation**: Checks required fields and types
- **Quality Metrics**: Calculates completeness, accuracy, consistency, timeliness
- **Adaptive Delays**: Adjusts based on API success rates
- **Report Generation**: Outputs four JSON reports

4.3 Screenshots: Agent Running

Note: the screenshot is quite big, so it will likely be moved to the end of the report by my latex compiler.

4.4 Code Implementation Highlights

4.4.1 1. Configuration Management

Loads config from JSON or uses defaults. Tracks requests, timestamps, and adaptive variables.

```
def __init__(self, config_file=None):
    self.config = self.load_config(config_file)
                  if config_file else self._default_config()
    self.collection_stats = {
        'total_requests': 0,
        'successful_requests': 0,
        'failed_requests': 0,
    }
    self.delay_multiplier = 1.0
```

4.4.2 2. Collection Loop

Main loop checks completion, processes data, assesses performance, and delays.

```
def run_collection(self):
    try:
        while not self.collection_complete():
            data = self.collect_batch()
            if data:
                self.process_and_store(data)
            self.assess_performance()
            self.respectful_delay()
    finally:
        self.generate_final_report()
```

4.4.3 3. Quality Assessment

Evaluates four dimensions: completeness, accuracy, consistency, timeliness.

```
def assess_data_quality(self):
    quality_metrics = {
        'completeness': self.check_completeness(),
        'accuracy': self.check_accuracy(),
        'consistency': self.check_consistency(),
        'timeliness': 1.0
    }
    return sum(quality_metrics.values()) / len(quality_metrics)
```

4.4.4 4. Adaptive Strategy

Doubles delay if success rate drops below 50%. Reduces delay if above 90%.

```
def adjust_strategy(self):
    success_rate = self.get_success_rate()
    if success_rate < 0.5:
        self.delay_multiplier *= 2
    elif success_rate > 0.9:
        self.delay_multiplier *= 0.8
```

4.4.5 5. Rate Limiting

Base delay with random jitter (0.5x to 1.5x) to avoid overwhelming the API.

```
def respectful_delay(self):
    base_delay = self.config.get('base_delay', 1.0)
    jitter = random.uniform(0.5, 1.5)
    time.sleep(base_delay * self.delay_multiplier * jitter)
```

4.4.6 6. Report Generation

Generates four JSON files: collected data, summary, quality report, metadata.

5 Part 5: Documentation (20 points)

5.1 Quality Assessment Report

5.1.1 Total Number of Records

- **Events**: 5 records
- **Markets**: 5 records
- **Total**: 10 data points
- **Categories**: Politics, Economics, Sports, Technology, Weather

Endpoint	Attempted	Successful	Rate
Exchange Status	1	1	100%
Events	1	1	100%
Markets	5	5	100%
Overall	**7**	**7**	**100%**

Table 1: Success Rates by Endpoint

5.1.2 Collection Success Rate

5.1.3 Quality Score

Completeness All required fields populated. **Score: 100%**

Accuracy Type validation passed for all numeric and string fields. **Score: 100%**

Consistency All records have identical schema. **Score: 100%**

Timeliness Real-time collection with timestamps. **Score: 100%**

Overall Quality Score 100/100

5.2 Collection Summary

5.2.1 Total Data Points Collected

- 10 records (5 events + 5 markets)
- 70 total fields (7 per market)
- 3 endpoints accessed
- 5 market categories

5.2.2 Success/Failure Rates by API

Endpoint	Success Rate
Exchange Status	100%
Events	100%
Markets	100%
Overall	**100%**

Table 2: API Performance

5.2.3 Quality Metrics and Trends

- Data quality stayed at 100% throughout collection
- No API failures or timeouts
- Schema remained consistent
- No missing values

5.2.4 Issues Encountered

- No critical issues
- No rate limiting violations
- No validation failures
- All responses within timeout

5.2.5 Recommendations for Future Collection

1. Maintain current rate limiting strategy
2. Add historical price data collection
3. Implement data versioning for trend analysis
4. Add incremental collection for efficiency
5. Tune delay based on long-term behavior

Figure 2: Screenshot showing the agent running successfully