Mini-Chapter — Data Acquisition and Ingestion

Why Ingestion Matters

Analyses fail quietly when inputs are wrong. In finance, latency, schema drift, and credential leaks can all destroy trust. Treat ingestion as engineered infrastructure, not a side quest.

APIs vs Scraping

- **APIs:** Stable formats, documented fields, rate limits, auth. Best for reliability.
- **Scraping:** Flexible when no API exists, but fragile. Respect Terms and robots.txt.
- **Rule of thumb:** prefer APIs; scrape only when permitted and necessary.

Secrets Management

- Store keys in `.env`, never in code.
- Load with:

```
from dotenv import load_dotenv
load_dotenv()
import os
key = os.getenv("ALPHAVANTAGE_API_KEY")
```

- Keep `.env` out of version control (`.gitignore`).

Validation First

Minimal, high-value checks:

- **Schema:** required columns present
- **Types:** parse dates, coerce numerics
- **Completeness:** NA counts, shapes
- **Sanity:** ranges, duplicates, monotonic dates

Reproducible Storage

- Save pristine inputs to `data/raw/`
- Timestamp filenames; encode source and parameters
- Log sources and assumptions in your notebook or a `README`

Patterns

1. **Request → Parse → Validate → Save** (fail early)
2. **Retry with backoff** for flaky endpoints
3. **Separate secrets & config** from code
4. **Script it** — no manual copy/paste

Example Validation Snippet

```
req = ['date','adj_close']
types = {'date':'datetime64[ns]','adj_close':'float'}
msgs = validate_df(df, req, types)
if 'missing_cols' in msgs: raise ValueError(msgs['missing_cols'])
```

What to Watch Out For

- Shadow/undocumented API changes
- HTML structure shifts breaking selectors
- Silent float parsing failures (commas, currencies)
- Accidentally committing `.env` or token-bearing notebooks

Where This Fits in the Lifecycle

This stage produces **trusted raw inputs** that power EDA and modeling. Your future self (and teammates) will thank you.