SigmaRanks – Project Overview

A simple walkthrough of the data-driven trading pipeline

What is SigmaRanks?
SigmaRanks is a streamlined trading research project that builds on *MrktMove*, expanding it into a full, ranked-signal pipeline. While *MrktMove* focused on predicting next-day returns for individual stocks, *SigmaRanks* turns those predictions into structured trading insights — complete with ranking, portfolio simulation, and testing.

It's designed to show how a complete data-driven trading process can be built step-by-step in one clear system — from fetching market data all the way to evaluating results.

In short, SigmaRanks:

- Analyzes and organizes market data into structured features
- Builds ranked trading signals and portfolios
- Tests ideas safely through simulation and backtesting

Bird's-Eye View of the SigmaRanks Pipeline

The SigmaRanks project follows a simple, logical flow — from gathering data to testing trading ideas.
Each step builds on the one before it, forming a complete loop that helps researchers move from raw data to actionable insight.

Pipeline Overview

Fetch → Clean → Feature Build → Modeling → Ranking → Portfolio → Backtesting

---

Fetch – Gather stock and market data from reliable online sources.
Clean – Remove errors and organize the raw data so it's ready for use.
Feature Build – Create meaningful variables that capture patterns in price and volume.
Modeling – Train predictive models that estimate future returns and risk.
Ranking – Sort stocks based on the model's signals to find the strongest opportunities.
Portfolio – Combine ranked signals into a balanced portfolio that manages risk.
Backtesting – Simulate past performance to see how the strategy would have worked in real markets.

Together, these stages make up the **SigmaRanks pipeline** — a repeatable, data-driven framework for developing and testing systematic trading strategies.

Methods and Where They Fit

Each method in SigmaRanks supports a different part of the pipeline.
Together, they make the system more realistic, stable, and informative.

Wyckoff Features – Feature Build Stage

These features look for classic patterns of buying and selling pressure in price and volume data.
They help the model detect when large investors may be accumulating or distributing shares — an early sign of potential market moves.

Market Signaling – Pre-Modeling Stage

Before modeling begins, SigmaRanks summarizes the overall market mood using sentiment and volatility data.
This helps adjust individual stock predictions based on whether the market environment is calm, uncertain, or trending.

Model Types (Quantile Regression) – Modeling Stage

Instead of predicting a single number, quantile regression estimates a range — showing both expected return and uncertainty.
This gives a more complete picture of potential outcomes, helping the ranking and portfolio steps make better decisions.

Monte Carlo Simulation – Portfolio Stage

Here, random scenarios are generated to test how the portfolio would behave under many different market conditions.
It's a way to check if the portfolio can handle volatility and avoid over-reliance on lucky outcomes.

Backtesting – Final Stage

At the end, the system tests the full strategy using past market data.
This step helps confirm whether the signals and portfolio logic would have worked historically before putting any money at risk.

Each of these methods fits naturally into the flow — adding structure, realism, and confidence to every part of the SigmaRanks process.

File Structure Overview

Below is the full directory layout of the SigmaRanks project.
It shows how every file and folder connects the pieces of the pipeline — from raw data collection to model testing.

```
SigmaRanks/

├─ README.md

├─ notes.md

├─ overview.pdf
├─ pyproject.toml
├─ .env.example
├─ config/
│  ├─ base.yaml
│  ├─ modeling.yaml
│  └─ portfolio.yaml
├─ data/
│  ├─ raw/
│  ├─ interim/
│  └─ features/
├─ notebooks/
│  ├─ 00_data_audit.ipynb
│  ├─ 10_quantiles_dev.ipynb
│  └─ 20_backtest_report.ipynb
├─ scripts/
│  ├─ fetch.py
│  ├─ build_features.py
│  ├─ train_quantiles.py
│  ├─ run_backtest.py
│  └─ daily_signal.py
├─ src/sigmaranks/
│  ├─ ingest/
│  │  ├─ loaders.py
│  │  └─ universe.py
│  ├─ features/
│  │  ├─ price_volume.py
│  │  ├─ wyckoff.py
│  │  └─ assemble.py
│  ├─ modeling/
│  │  ├─ quantile_model.py
│  │  ├─ outlook.py
│  │  └─ eval_utils.py
│  ├─ risk/
│  │  ├─ cov_estimation.py
│  │  └─ vol_target.py
│  ├─ portfolio/
│  │  ├─ ranking.py
│  │  ├─ construction.py
│  │  ├─ turnover.py
│  │  └─ mc_sizer.py
│  ├─ execution/
│  │  ├─ cost_models.py
│  │  └─ simulator.py
│  ├─ backtest/
│  │  ├─ engine.py
│  │  └─ metrics.py
│   └─ utils/
│       ├─ io.py
│       ├─ dates.py
│       └─ logging.py
└─ tests/
    ├─ test_features.py
    ├─ test_modeling.py
    ├─ test_portfolio.py
    └─ test_backtest.py
```

This structure is the **backbone of SigmaRanks** — organizing every stage of the process into clear sections.
It keeps data, modeling, and backtesting separate but connected, making the system easy to navigate and extend.

File-by-File Explanations

Root Files

README.md — Provides a high-level introduction to the project and quick setup notes. It's the first place readers go to understand SigmaRanks.
notes.md — A scratchpad for research ideas, experiment notes, and testing observations. Helps track changes over time.
overview.pdf — This document. It explains how the project works from start to finish in plain English.
pyproject.toml — Lists all required Python packages and dependencies. It ensures anyone can install and run the project consistently.
.env.example — A template for storing API keys or sensitive credentials safely, without sharing private data.

Config Folder

base.yaml — Holds general settings such as file paths, date ranges, and base parameters.
modeling.yaml — Stores model-related options like quantile levels and feature lists.
portfolio.yaml — Contains portfolio construction and risk settings, such as target volatility and rebalance frequency.

Data Folder

raw/ — Where unprocessed market data is first stored after fetching.
interim/ — Holds cleaned, partially processed data ready for feature building.
features/ — Contains final feature sets used by the models for prediction.

Notebooks Folder

00_data_audit.ipynb — Checks that incoming data looks correct before modeling starts.
10_quantiles_dev.ipynb — Used for developing and tuning the quantile regression models.
20_backtest_report.ipynb — Summarizes and visualizes performance results after backtesting.

Scripts Folder

fetch.py — Pulls the latest market data for all selected stocks and saves it to `data/raw/`.
build_features.py — Creates predictive variables such as moving averages and Wyckoff features.
train_quantiles.py — Trains quantile regression models to estimate expected return and risk.
run_backtest.py — Runs historical simulations to evaluate strategy performance.
daily_signal.py — Produces daily trading signals for live or paper-trading environments.

Source Code – `src/sigmaranks/`

ingest/loaders.py — Handles reading and writing of data files in consistent formats.
ingest/universe.py — Defines which stocks or assets are included in the trading universe.

features/price_volume.py — Builds features from price and volume trends.
features/wyckoff.py — Implements Wyckoff-based buying and selling pressure indicators.
features/assemble.py — Combines multiple feature sources into one structured dataset.

modeling/quantile_model.py — Trains and applies quantile regression models for prediction.
modeling/outlook.py — Summarizes model results into a market outlook or sentiment measure.
modeling/eval_utils.py — Provides helper tools to evaluate model accuracy and stability.

risk/cov_estimation.py — Estimates relationships between assets to manage diversification.
risk/vol_target.py — Adjusts position sizes to maintain a consistent volatility target.

portfolio/ranking.py — Ranks assets based on expected return and risk estimates.
portfolio/construction.py — Builds a portfolio allocation from ranked signals.
portfolio/turnover.py — Measures how frequently portfolio holdings change.
portfolio/mc_sizer.py — Uses Monte Carlo simulation to test and size portfolio positions safely.

execution/cost_models.py — Estimates transaction costs and slippage from trading.
execution/simulator.py — Mimics the trading process to see realistic execution results.

backtest/engine.py — Runs historical simulations across time to test the entire strategy.
backtest/metrics.py — Calculates performance metrics like Sharpe ratio and drawdown.

utils/io.py — Handles reading and saving files efficiently.
utils/dates.py — Manages trading calendars and date conversions.
utils/logging.py — Keeps consistent logs of each run for debugging and tracking progress.

Tests Folder

test_features.py — Ensures all feature-building functions work as intended.
test_modeling.py — Checks model outputs for accuracy and stability.
test_portfolio.py — Validates portfolio ranking and construction logic.
test_backtest.py — Confirms that backtest calculations produce expected results.

How to Run + Final Notes

Running the Pipeline

To execute the full SigmaRanks process, run the scripts below in order:

```
python -m scripts.fetch

python -m scripts.build_features

python -m scripts.train_quantiles

python -m scripts.run_backtest

python -m scripts.daily_signal
```

Each step builds on the last — starting from raw data and ending with tested trading signals.

Configuration

All settings are stored in the `/config` folder as YAML files.
This includes which tickers to analyze, model parameters, and portfolio settings.
You can adjust these without touching the core code, making experimentation simple and safe.

Environment

SigmaRanks can be run on any computer with Python installed and the libraries listed in `pyproject.toml`.
Once the environment is set up, the entire pipeline can be executed from the command line.

Final Notes

SigmaRanks connects every part of a data-driven trading workflow — from raw data to live signals — in one structured system.
It's designed to help researchers and students test ideas clearly, understand performance, and explore systematic trading safely.