sf-trader

Interactive terminal trading application for quantitative portfolio management with IBKR integration.

Overview

sf-trader is a command-line trading system that: - Calculates alpha signals from multiple factors (momentum, reversal, beta) - Optimizes portfolio weights using mean-variance optimization - Generates optimal share allocations based on available funds - Executes trades through Interactive Brokers (IBKR)

Installation

```
uv sync
```

Requirements

- Python >= 3.11
- Interactive Brokers account with API access
- IBKR TWS or IB Gateway running and configured

Configuration

Create a `config.yml` file in your project directory to define your trading strategy. Here's the structure:

Example Configuration

```yaml
signals:
  - momentum
  - reversal
  - beta

ic: 0.05

signal-combinator: mean

constraints:
  - full-investment
  - long-only
  - no-buying-on-margin
  - unit-beta

gamma: 450

decimal-places: 4
```

```yaml
ignore-tickers:
  - MONDQ
  - VELO
  - BGXXQ
  - PSTX.CVR
```

Configuration Parameters

`signals` (required) List of alpha signals to use. Available signals: - **`momentum`**: 230-day momentum with 22-day lag (252 days lookback) - **`reversal`**: Short-term reversal based on 22-day returns - **`beta`**: Predicted beta signal

`ic` (required) Information coefficient - expected correlation between signals and future returns (float). - Typical values: 0.01 to 0.10 - Higher values indicate more confidence in signals

`signal-combinator` (required) Method to combine multiple signals. Available combinators: - **`mean`**: Simple average of all signals

`constraints` (required) List of portfolio constraints for optimization. Available constraints: - **`full-investment`**: Sum of weights equals 1 (fully invested) - **`long-only`**: All weights >= 0 (no short positions) - **`no-buying-on-margin`**: Prevents over-leveraging - **`unit-beta`**: Portfolio beta constrained to 1.0

`gamma` (required) Risk aversion parameter for mean-variance optimization (float). - Higher values → more risk-averse portfolios - Typical values: 100 to 1000

`decimal-places` (required) Number of decimal places for weight precision (integer). - Typical value: 4

`ignore-tickers` (optional) List of tickers to exclude from the trading universe. - Useful for removing delisted stocks, problematic tickers, etc.

Usage

Testing with Dry Run

Before executing real trades, always test your configuration with `--dry-run`:

```bash
# Basic dry run with default config.yml
python -m sf_trader run --dry-run

# Dry run with custom config file
python -m sf_trader run --config my_config.yml --dry-run
```

```
# Dry run with Barra prices instead of IBKR
python -m sf_trader run --dry-run --prices barra

# Dry run for a specific trade date
python -m sf_trader run --dry-run --trade-date 2024-01-15
```

The dry run will: 1. Load your configuration 2. Fetch market data and prices 3. Calculate alpha signals 4. Optimize portfolio weights 5. Generate trade list 6. Display portfolio metrics and trades 7. **Skip actual trade execution**

Production Run

After verifying with dry run, execute trades:

```
# Execute trades with default config.yml
python -m sf_trader run

# Execute with custom config
python -m sf_trader run --config my_config.yml

# Execute with Barra prices
python -m sf_trader run --prices barra

# Execute for a specific trade date
python -m sf_trader run --trade-date 2024-01-15
```

Production runs will: - Execute all steps from dry run - Submit limit orders to IBKR - Display execution results

Command Reference

`run` Command Execute the full trading pipeline.

Flags:

Flag	Type	Default	Description
`--config`	Path	`config.yml`	Path to configuration file
`--dry-run`	Flag	`False`	Simulate trades without executing
`--prices`	Choice	`ibkr`	Price source (`ibkr` or `barra`)
`--trade-date`	Date	Today	Trade date in YYYY-MM-DD format

Examples:

```
# Most common: dry run to test
python -m sf_trader run --dry-run
```

```
# Production run with all defaults
python -m sf_trader run

# Custom config and date
python -m sf_trader run --config configs/aggressive.yml --trade-date 2024-03-01

# Use historical Barra prices for backtesting
python -m sf_trader run --dry-run --prices barra --trade-date 2024-01-01
```

`clear-orders` Command Cancel all open orders in IBKR.

Usage:

```
python -m sf_trader clear-orders
```

This command: - Connects to IBKR - Retrieves all open orders - Cancels each order - Displays cancellation results

Trading Pipeline

The `run` command executes these steps:

1. **Parse configuration** - Load and validate config.yml
2. **Load trading universe** - Get list of tradable securities
3. **Fetch available funds** - Query IBKR account value
4. **Get prices** - Fetch current prices from IBKR or Barra
5. **Filter tradable tickers** - Remove tickers without valid prices
6. **Load asset data** - Historical returns and risk data
7. **Calculate alpha signals** - Compute momentum, reversal, beta signals
8. **Get predicted betas** - Fetch beta forecasts
9. **Optimize portfolio** - Mean-variance optimization with constraints
10. **Generate optimal shares** - Convert weights to share quantities
11. **Compute portfolio metrics** - Risk, return, and exposure analysis
12. **Fetch current positions** - Get existing portfolio from IBKR
13. **Compute trade list** - Calculate required trades (buy/sell)
14. **Display top long positions** - Show top 10 BUY orders by dollar value with current/optimal position details
15. **Execute trades** - Submit limit orders to IBKR (if not dry run)

Extending the System

Creating Custom Signals

Signals are alpha factors that predict future returns. To create a new signal, edit sf_trader/signals.py.

Structure:

A signal is defined using the `Signal` dataclass with three components: - `name`: Identifier for the signal (string) - `expr`: Polars expression that computes the signal - `lookback_days`: Number of days of historical data required (integer)

Example - Creating a Volume Signal:

```
from dataclasses import dataclass
import polars as pl

@dataclass
class Signal:
    name: str
    expr: pl.Expr
    lookback_days: int

# Define your custom signal
volume_signal = Signal(
    name="volume",
    expr=(
        pl.col("volume")
        .rolling_mean(window_size=20)
        .over("barrid")
        .alias("volume")
    ),
    lookback_days=20,
)

# Add to the registry
SIGNALS = {
    "momentum": momentum,
    "reversal": reversal,
    "beta": beta,
    "volume": volume_signal,  # Add your signal here
}
```

Tips: - Use `.over("barrid")` to apply operations per asset - Use `.alias()` to name the output column - Set `lookback_days` to the minimum data needed - Signals should be standardized/normalized for best results

Creating Signal Combinators

Signal combinators merge multiple signals into a single alpha. To create a new combinator, edit sf_trader/combinators.py.

Structure:

A combinator is created using a factory function that returns a `SignalCombinator` instance with: - `name`: Identifier for the combinator (string) - `combine_fn`:

Function that takes signal names and returns a Polars expression

Example - Creating a Max Combinator:

```python
import polars as pl
from sf_trader.models import SignalCombinator

def max_combinator() -> SignalCombinator:
    """Take the maximum value across all signals"""
    def combine_fn(signal_names: list[str]) -> pl.Expr:
        # Create max expression across all signals
        expr = pl.max_horizontal([pl.col(name) for name in signal_names])
        return expr.alias("alpha")

    return SignalCombinator(
        name="max",
        combine_fn=combine_fn,
    )

# Add to the registry
COMBINATORS = {
    "mean": mean_combinator,
    "max": max_combinator,  # Add your combinator here
}
```

Usage in config.yml:

```yaml
signal-combinator: max
```

Tips: - The `combine_fn` receives a list of signal column names - Always alias the final expression as "alpha" - Use Polars expressions for efficient computation - Consider normalization/standardization of signals before combining

Tips

1. **Always start with `--dry-run`** to validate your configuration and review proposed trades
2. **Monitor execution** - Watch IBKR TWS for order fills
3. **Use `clear-orders`** if you need to cancel all pending orders
4. **Test with historical dates** using `--trade-date` and `--prices barra` for backtesting
5. **Adjust `gamma`** to control portfolio concentration (higher = more diversified)
6. **Review ignore-tickers** regularly to exclude problematic securities

Troubleshooting

"Configuration file not found" - Ensure `config.yml` exists in current directory or use `--config` flag

"Failed to connect to IBKR" - Verify TWS or IB Gateway is running - Check API settings are enabled in IBKR - Confirm connection parameters

"No tradable tickers" - Check that prices are available for your universe - Verify tickers are not all in ignore-tickers list - Ensure market is open or use `--prices barra` for testing

License

SilverFund Proprietary