Econometrics and Trading Strategy Programming Exercises

Easy Exercises (1-2 hours each)

Data Collection & Cleaning

1. **stock_data_downloader** - Write a Python script using `yfinance` or similar library to download historical OHLCV data for a list of tickers (e.g., AAPL, MSFT, GOOGL) for the last 5 years. Save to CSV.
2. **missing_data_handler** - Create a script that detects missing values in price data, fills gaps using forward-fill or interpolation, and reports data quality metrics (% missing, gaps duration).
3. **corporate_actions_adjuster** - Implement adjustment for stock splits and dividends. Given raw price data and corporate actions list, produce adjusted prices.
4. **data_validator** - Write a validator that checks for data anomalies: negative prices, zero volume, gaps > 3 days, price jumps > 20%, volume spikes > 10x average.

Basic Statistics & Returns

5. **returns_calculator** - Implement functions to calculate simple returns, log returns, and cumulative returns from price series. Compare differences for volatile vs stable stocks.
6. **descriptive_stats** - Calculate and display mean, median, std dev, skewness, kurtosis for returns of multiple assets. Identify which assets have fat tails.
7. **correlation_matrix** - Compute correlation matrix for returns of 10 stocks. Visualize as heatmap. Identify highly correlated pairs ($|\rho| > 0.7$).
8. **rolling_statistics** - Calculate 30-day rolling mean, std dev, min, max for a stock. Plot price with Bollinger Bands (±2 std dev).

Simple Technical Indicators

9. **sma_crossover** - Implement Simple Moving Average (SMA) calculation. Generate buy/sell signals when fast SMA (20-day) crosses slow SMA (50-day).
10. **rsi_indicator** - Calculate Relative Strength Index (RSI) for 14-day period. Identify overbought (>70) and oversold (<30) conditions.
11. **macd_calculator** - Implement MACD (Moving Average Convergence Divergence): MACD line, signal line, histogram. Generate crossover signals.

12. **atr_volatility** - Calculate Average True Range (ATR) for 14-day period to measure volatility. Compare ATR across different assets.

Basic Backtesting

13. **buy_hold_strategy** - Implement buy-and-hold strategy backtester. Calculate total return, annualized return, max drawdown for given period.
14. **equal_weight_portfolio** - Create equal-weight portfolio of 5 stocks. Rebalance monthly. Calculate portfolio returns and compare with individual stocks.
15. **strategy_metrics** - Given a series of strategy returns, calculate: Sharpe ratio, Sortino ratio, max drawdown, win rate, profit factor.

Data Visualization

16. **candlestick_chart** - Create candlestick chart with volume bars for any stock using matplotlib or plotly.
17. **returns_distribution** - Plot histogram and Q-Q plot of returns against normal distribution. Test for normality using statistical tests.
18. **equity_curve** - Given strategy signals and prices, calculate and plot equity curve showing portfolio value over time.

Time Series Basics

19. **stationarity_test** - Implement Augmented Dickey-Fuller test to check if price/returns series is stationary. Test on prices vs returns.
20. **autocorrelation** - Calculate and plot ACF (autocorrelation function) and PACF for returns. Test for serial correlation.

Market Microstructure

21. **bid_ask_spread** - Given tick data with bid/ask prices, calculate time-weighted average spread, relative spread (% of mid-price), and spread volatility.
22. **vwap_calculator** - Calculate Volume-Weighted Average Price (VWAP) for intraday data. Compare execution price against VWAP benchmark.

Portfolio Basics

23. **portfolio_variance** - Given weights, expected returns, and covariance matrix, calculate portfolio expected return and variance.
24. **efficient_frontier_2assets** - For two assets, calculate and plot efficient frontier showing risk-return tradeoff for different weight combinations.
25. **beta_calculator** - Calculate stock beta against market index (S&P 500). Identify high-beta (>1.5) and low-beta (<0.5) stocks.

Medium Exercises (3-5 hours each)

Advanced Technical Strategies

1. **mean_reversion_strategy** - Implement pairs trading strategy: find cointegrated stock pairs, calculate spread, generate signals when spread deviates > 2 std dev from mean. Backtest on historical data.
2. **momentum_strategy** - Create momentum strategy: rank stocks by 12-month returns (skipping last month), go long top decile, short bottom decile. Rebalance monthly. Calculate performance metrics.
3. **breakout_strategy** - Implement Donchian channel breakout: buy when price breaks above 20-day high, sell when below 20-day low. Add ATR-based position sizing.

Statistical Analysis

4. **regression_analysis** - Perform multi-factor regression: regress stock returns on market, SMB (size), HML (value), momentum factors. Analyze coefficients, R^2, residuals.
5. **capm_validation** - Test CAPM model: estimate betas for 50 stocks, run cross-sectional regression of average returns on betas. Test if high-beta stocks have higher returns.
6. **event_study** - Implement event study framework: calculate abnormal returns around earnings announcements or other events using market model. Test if CAR (cumulative abnormal returns) is significant.

Time Series Models

7. **arima_forecasting** - Fit ARIMA model to returns or volatility series. Use AIC/BIC for model selection. Generate forecasts and evaluate out-of-sample accuracy.
8. **garch_volatility** - Implement GARCH(1,1) model for volatility forecasting. Compare predicted volatility with realized volatility. Use for position sizing.
9. **var_calculation** - Calculate Value at Risk (VaR) using three methods: historical simulation, variance-covariance, Monte Carlo. Compare results at 95% and 99% confidence levels.

Portfolio Optimization

10. **markowitz_optimization** - Implement mean-variance optimization to find optimal portfolio weights. Handle constraints: weights sum to 1, no short selling, max weight per asset. Use historical returns/covariance.
11. **risk_parity** - Create risk parity portfolio where each asset contributes equally to portfolio risk. Compare performance with equal-weight and market-cap weighted portfolios.
12. **black_litterman** - Implement Black-Litterman model combining market equilibrium with investor views. Compare with standard mean-variance optimization.

Machine Learning for Trading

13. **feature_engineering** - Create 50+ features from price/volume data: technical indicators, price patterns, statistical measures, lagged returns. Evaluate feature importance.
14. **classification_model** - Build binary classifier (Random Forest or Gradient Boosting) to predict if next-day return will be positive. Use walk-forward validation. Evaluate precision/recall.
15. **regime_detection** - Use Hidden Markov Model or k-means clustering to detect market regimes (bull/bear/sideways). Adapt strategy parameters based on detected regime.

Hard Exercises (Multiple evenings)

Advanced Backtesting Framework

1. **production_backtester** - Build comprehensive backtesting framework with: realistic transaction costs, slippage modeling, position limits, margin requirements, corporate actions handling, benchmark comparison, Monte Carlo scenario analysis, drawdown control. Test multiple strategies.
2. **high_frequency_backtest** - Implement tick-by-tick backtesting engine for intraday strategies. Handle bid-ask spread, queue position, partial fills, latency simulation. Backtest market-making or statistical arbitrage strategy.
3. **walk_forward_optimization** - Implement walk-forward analysis framework: optimize strategy parameters on training window, test on out-of-sample period, roll forward. Evaluate parameter stability and overfitting.

Advanced Statistical Models

4. **multivariate_garch** - Implement DCC-GARCH (Dynamic Conditional Correlation) or BEKK-GARCH model for modeling time-varying correlations between assets. Use for dynamic portfolio optimization.
5. **cointegration_portfolio** - Build statistical arbitrage strategy based on Johansen cointegration test. Identify cointegrated baskets of 3-5 stocks, trade mean reversion of portfolio. Handle non-stationarity.
6. **structural_breaks** - Implement Chow test and CUSUM test to detect structural breaks in factor models or trading strategies. Automatically adapt when breaks detected.

Machine Learning Production

7. **ml_trading_system** - Build complete ML-based trading system: feature engineering pipeline, model training with cross-validation, hyperparameter tuning, ensemble methods, prediction serving, performance monitoring, model retraining triggers. Backtest with realistic assumptions.
8. **reinforcement_learning_trader** - Implement Deep Q-Learning or PPO agent for trading. State space: market features, actions: long/short/neutral, reward: risk-adjusted returns.

Train on historical data, evaluate on holdout period.

9. **nlp_sentiment_strategy** - Build sentiment-based trading strategy: scrape financial news/Twitter, perform sentiment analysis using transformers (FinBERT), aggregate sentiment scores, generate trading signals. Backtest alpha generation.

Advanced Portfolio Management

10. **dynamic_asset_allocation** - Implement tactical asset allocation system using economic indicators (yield curve, VIX, PMI, unemployment) to dynamically shift between stocks/bonds/commodities. Include transaction cost optimization and tax considerations.

Testing Environment & Data Sources

Data Sources

Free Historical Data:

- **yfinance** (Yahoo Finance) - stocks, ETFs, indices, cryptocurrencies
- **pandas-datareader** - FRED, World Bank, OECD economic data
- **Alpha Vantage** - stocks, forex (free tier: 5 calls/min)
- **Quandl/Nasdaq Data Link** - futures, economic indicators
- **MOEX API** - Russian stocks (you're familiar with this!)
- **cryptocompare** - cryptocurrency data

Sample Datasets:

- S&P 500 constituents (current and historical)
- Fama-French factors (Dartmouth library)
- Kenneth French data library (factor returns)
- Quantopian/Zipline example datasets

For Testing:

```
import yfinance as yf
import pandas as pd

# Download S&P 500 data
spy = yf.download('SPY', start='2010-01-01', end='2023-12-31')

# Multiple tickers
tickers = ['AAPL', 'MSFT', 'GOOGL', 'AMZN']
data = yf.download(tickers, start='2015-01-01')
```

Required Libraries

Core Data Science:

```
pip install numpy pandas scipy matplotlib seaborn
```

Financial Data:

```
pip install yfinance pandas-datareader alpha_vantage
```

Technical Analysis:

```
pip install ta-lib  # or pandas-ta, tulipy
```

Statistical/Econometric:

```
pip install statsmodels arch scikit-learn
```

Backtesting:

```
pip install backtrader zipline-reloaded bt vectorbt
```

Machine Learning:

```
pip install scikit-learn xgboost lightgbm tensorflow
```

Portfolio Optimization:

```
pip install PyPortfolioOpt cvxpy
```

Development Environment Setup

Recommended Structure:

```
trading_exercises/
├── data/
│   ├── raw/          # Downloaded data
│   ├── processed/    # Cleaned data
│   └── features/     # Engineered features
├── strategies/
│   ├── momentum.py
│   ├── mean_reversion.py
│   └── ml_based.py
├── backtesting/
│   ├── engine.py
```

```
│   ├── metrics.py
│   └── visualization.py
├── notebooks/        # Jupyter notebooks for exploration
├── tests/            # Unit tests
└── utils/
    ├── data_loader.py
    ├── indicators.py
    └── portfolio.py
```

Jupyter Setup:

```
pip install jupyter notebook jupyterlab
jupyter lab
```

Validation & Testing

Easy Exercises:

- Calculate metrics on known datasets (verify against published results)
- Visual inspection of plots and indicators
- Compare your indicators with TA-Lib implementations
- Test on simple synthetic data where answer is known

Medium Exercises:

- Backtest results should be reasonable (Sharpe 0.5-2.0 for good strategies)
- Strategy should make economic sense (why would it work?)
- Test on different time periods and assets
- Compare with published academic results
- Out-of-sample testing is crucial

Hard Exercises:

- Walk-forward validation shows consistent performance
- Monte Carlo simulation of strategy returns
- Stress testing under different market conditions
- Transaction cost sensitivity analysis
- Parameter stability analysis

Performance Metrics Reference

Returns Metrics:

- Total Return: (Final Value - Initial Value) / Initial Value
- Annualized Return: (1 + Total Return)^(252/Days) - 1
- CAGR: (Final/Initial)^(1/Years) - 1

Risk Metrics:

- Volatility: std(returns) × √252
- Max Drawdown: max(peak - trough) / peak
- VaR (95%): 5th percentile of returns distribution

Risk-Adjusted:

- Sharpe Ratio: (Return - RiskFree) / Volatility
- Sortino Ratio: (Return - RiskFree) / Downside Deviation
- Calmar Ratio: CAGR / Max Drawdown
- Information Ratio: (Portfolio Return - Benchmark) / Tracking Error

Trading Metrics:

- Win Rate: % of profitable trades
- Profit Factor: Gross Profit / Gross Loss
- Average Win/Loss Ratio
- Recovery Factor: Net Profit / Max Drawdown
- Expectancy: (Win% × Avg Win) - (Loss% × Avg Loss)

Common Pitfalls to Avoid

Survivorship Bias:

- Don't use current index constituents for historical backtests
- Account for delisted/bankrupt companies

Look-Ahead Bias:

- Only use information available at decision time
- Be careful with data alignment (close-to-close returns)

Data Snooping:

- Test on out-of-sample data
- Don't optimize on full dataset
- Use walk-forward validation

Overfitting:

- Limit number of parameters
- Use regularization in ML models
- Cross-validation is essential
- Simple strategies often work better

Transaction Costs:

- Always include realistic commissions (0.1-0.5 bps for stocks)
- Model slippage for large orders
- Include bid-ask spread for frequent trading
- Consider market impact for large positions

Unrealistic Assumptions:

- Can't trade at exact close price
- Partial fills for large orders
- Liquidity constraints
- Margin requirements

Sample Test Cases

For Moving Average Crossover:

```
# Known test case
prices = pd.Series([10, 11, 12, 13, 14, 15, 14, 13, 12, 11])
sma5 = prices.rolling(5).mean()
# Should give: [NaN, NaN, NaN, NaN, 12.0, 13.0, 13.6, 13.8, 13.6, 13.0]
```

For Sharpe Ratio:

```
# Constant positive returns should give high Sharpe
returns = pd.Series([0.01] * 252)  # 1% daily
sharpe = returns.mean() / returns.std() * np.sqrt(252)
# Should be very high (infinite theoretically)
```

For Beta:

```
# Stock perfectly correlated with market should have beta = 1
market_returns = np.random.randn(252)
stock_returns = market_returns  # Perfect correlation
beta = np.cov(stock_returns, market_returns)[0,1] / np.var(market_returns)
# Should be exactly 1.0
```

Benchmark Datasets

For Strategy Testing:

- 2008-2009: Financial crisis (stress test)
- 2010-2017: Bull market (trending strategies)
- 2018: High volatility (mean reversion test)
- 2020: COVID crash and recovery (extreme conditions)
- 2022: Bear market (defensive strategies)

Asset Classes:

- US Large Cap: S&P 500 (SPY)
- Small Cap: Russell 2000 (IWM)
- International: MSCI EAFE (EFA)
- Bonds: AGG, TLT
- Commodities: GLD, USO
- Crypto: BTC-USD, ETH-USD

Learning Progression

Phase 1 (Easy):

- Master data handling and basic statistics
- Implement simple indicators
- Understand return calculations
- Basic visualization

Phase 2 (Medium):

- Statistical analysis and hypothesis testing
- Time series modeling
- Portfolio optimization theory
- Machine learning basics

Phase 3 (Hard):

- Production-grade backtesting
- Advanced statistical models
- ML in production
- Risk management systems

Recommended Reading

Econometrics:

- "Analysis of Financial Time Series" - Ruey S. Tsay
- "Statistics and Data Analysis for Financial Engineering" - Ruppert & Matteson

Trading Strategies:

- "Algorithmic Trading" - Ernest P. Chan
- "Quantitative Trading" - Ernest P. Chan
- "Advances in Financial Machine Learning" - Marcos López de Prado

Portfolio Theory:

- "Active Portfolio Management" - Grinold & Kahn
- "Expected Returns" - Antti Ilmanen

Russian Market Specific

For MOEX data:

```
import aiomoex
import requests

# Get historical data
with requests.Session() as session:
    data = aiomoex.get_board_history(session, 'SBER')
    df = pd.DataFrame(data)
```

Popular Russian tickers for testing:

- SBER, GAZP, LKOH, ROSN (blue chips)
- VTBR, GMKN, NVTK (large cap)
- MOEX (the exchange itself)

This exercise list provides comprehensive coverage from basic data analysis through advanced quantitative finance and algorithmic trading, all implementable with Python and free data sources on your Debian system.