Week 4: Time-Series Forecasting

ARIMA & GARCH Models

MSc Banking and Finance - FinTech Course

September 30, 2025

Abstract

This document provides comprehensive coverage of Week 4's time-series forecasting implementation, encompassing stationarity testing, ARIMA model specification, GARCH volatility modeling, and Monte Carlo simulation for price prediction. Each cell from the associated Jupyter Notebook is examined through theoretical foundations, mathematical formulations, and practical implications for DeFi portfolio management and risk analysis.

Contents

1 Introduction and Environment Setup **3**
1.1 Cell 1: Library Configuration and Dependencies 3

2 Data Preparation and Visualization **3**
2.1 Cell 2: Loading Returns Data and Temporal Indexing 3

3 Stationarity Analysis **4**
3.1 Cell 3: Statistical Tests for Stationarity 4
3.2 Cell 4: Visual Stationarity Confirmation 5

4 Model Identification **6**
4.1 Cell 5: ACF and PACF Analysis 6

5 ARIMA Model Estimation **7**
5.1 Cell 6: ARIMA Model Fitting and Selection 7

6 Model Diagnostics **8**
6.1 Cell 7: Residual Analysis and Validation 8

7 Volatility Modeling **9**
7.1 Cell 8: GARCH Models for Volatility Clustering 9

8 Forecasting and Simulation **10**
8.1 Cell 9: Monte Carlo Price Forecasting 10

9 Model Evaluation **11**
9.1 Cell 10: Backtesting and Performance Metrics 11

10 Summary and Conclusions **12**
10.1 Key Learning Outcomes 12
10.2 Critical Limitations and Considerations 13
10.3 Future Research Directions 13

11 References and Further Reading **13**

12 Appendix: Technical Implementation **14**
12.1 A.1: Python Implementation Guidelines 14
12.2 A.2: Computational Considerations 14
12.3 A.3: Model Deployment Considerations 15

13 Practical Exercises **15**
13.1 Exercise 1: Stationarity Testing 15
13.2 Exercise 2: Model Selection 15
13.3 Exercise 3: Volatility Forecasting 15
13.4 Exercise 4: Monte Carlo Simulation 15
13.5 Exercise 5: Backtesting 15

14 Conclusion **15**

1 Introduction and Environment Setup

1.1 Cell 1: Library Configuration and Dependencies

Theoretical Background

Time-series forecasting in financial markets requires specialized econometric tools that extend beyond standard regression analysis. The computational environment must support:

- **Stationarity Testing:** Statistical procedures to verify time-invariant properties
- **ARIMA Modeling:** Autoregressive integrated moving average frameworks
- **Volatility Clustering:** GARCH models capturing heteroskedasticity
- **Monte Carlo Methods:** Stochastic simulation for uncertainty quantification

The architecture implements a rigorous workflow from exploratory analysis through diagnostic testing to production-grade forecasting systems. Proper seed management ensures reproducibility—critical for academic research and regulatory compliance.

Key Libraries and Their Roles

Library	Purpose
`statsmodels`	ARIMA estimation, diagnostic tests
`arch`	GARCH volatility models
`numpy/pandas`	Numerical computing, data manipulation
`scipy.stats`	Statistical distributions, hypothesis tests
`matplotlib/seaborn`	Professional visualization

Key Takeaways

- Specialized time-series libraries provide robust implementations of complex models
- Warning suppression should be used judiciously—never mask legitimate errors
- Reproducible random seeds are mandatory for academic and production systems
- Visualization configuration establishes professional presentation standards

2 Data Preparation and Visualization

2.1 Cell 2: Loading Returns Data and Temporal Indexing

Theoretical Background

Financial time-series analysis requires proper temporal indexing to enable:

- **Lag Operations:** Computing autocorrelations and partial autocorrelations
- **Rolling Windows:** Calculating moving averages and standard deviations
- **Frequency Conversion:** Aggregating daily to weekly/monthly returns
- **Seasonal Decomposition:** Identifying cyclical patterns

The choice between prices and returns is fundamental: prices are typically non-stationary (unit root processes), while returns are generally stationary—a prerequisite for ARIMA modeling.

Mathematical Formulation

Returns Transformation:
From prices P_t to log returns r_t:

$$r_t = \ln(P_t) - \ln(P_{t-1}) = \ln\left(\frac{P_t}{P_{t-1}}\right) \tag{1}$$

Properties of Log Returns:

- Time-additive: $r_{1,T} = \sum_{t=1}^{T} r_t$
- Approximately symmetric for small values
- Suitable for continuous-time models

Temporal Index Requirements:
Pandas DatetimeIndex with daily frequency ensures:

$$t_i - t_{i-1} = 1 \text{ day}, \quad i = 2, \ldots, T \tag{2}$$

Key Takeaways

- Log returns are preferred for econometric analysis due to stationarity
- Proper datetime indexing enables sophisticated time-series operations
- Visual inspection identifies outliers, structural breaks, and trends
- Cryptocurrency returns exhibit higher volatility than traditional assets

3 Stationarity Analysis

3.1 Cell 3: Statistical Tests for Stationarity

Theoretical Background

A time series $\{y_t\}$ is **strictly stationary** if the joint distribution of $(y_{t_1}, \ldots, y_{t_k})$ is identical to that of $(y_{t_1+\tau}, \ldots, y_{t_k+\tau})$ for all τ and k.

Weak (Covariance) Stationarity requires:

1. $E[y_t] = \mu$ (constant mean)
2. $\text{Var}(y_t) = \sigma^2$ (constant variance)
3. $\text{Cov}(y_t, y_{t-k}) = \gamma_k$ (autocorrelation depends only on lag k)

Stationarity is crucial because:

- ARIMA models require stationary inputs
- Non-stationary series can lead to spurious regressions
- Forecasts from stationary models are theoretically justified

Mathematical Formulation

Augmented Dickey-Fuller (ADF) Test:
Tests for unit root in the regression:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \epsilon_t \tag{3}$$

Hypotheses:

- H_0: $\gamma = 0$ (unit root exists, series is non-stationary)

- H_1: $\gamma < 0$ (no unit root, series is stationary)

Test Statistic:

$$\text{ADF} = \frac{\hat{\gamma}}{\text{SE}(\hat{\gamma})} \tag{4}$$

Compared against critical values from Dickey-Fuller distribution (not standard t-distribution).

KPSS (Kwiatkowski–Phillips–Schmidt–Shin) Test:

Complementary test with reversed hypotheses:

- H_0: Series is stationary
- H_1: Series has unit root

Test statistic based on variance ratio:

$$\text{KPSS} = \frac{\sum_{t=1}^{T} S_t^2}{T^2 \hat{\sigma}_\epsilon^2} \tag{5}$$

where $S_t = \sum_{i=1}^{t} e_i$ are partial sums of residuals.

Combined Interpretation

ADF Result	KPSS Result	Conclusion
Reject H_0	Don't reject H_0	Stationary
Don't reject H_0	Reject H_0	Non-stationary
Reject H_0	Reject H_0	Inconclusive
Don't reject H_0	Don't reject H_0	Inconclusive

Key Takeaways

- Cryptocurrency prices are non-stationary (random walk with drift)
- Log returns are typically stationary (constant mean and variance)
- Both ADF and KPSS tests should be used for robust conclusions
- Stationarity is a prerequisite for valid ARIMA modeling

3.2 Cell 4: Visual Stationarity Confirmation

Theoretical Background

Visual diagnostics complement formal statistical tests by revealing:

- Trending behavior in mean level
- Heteroskedasticity (changing variance)
- Structural breaks and regime shifts
- Seasonal patterns

Rolling statistics provide time-varying estimates that should remain stable for stationary series.

Mathematical Formulation

Rolling Mean:

$$\bar{y}_t(w) = \frac{1}{w} \sum_{i=t-w+1}^{t} y_i \tag{6}$$

Rolling Standard Deviation:

$$s_t(w) = \sqrt{\frac{1}{w-1} \sum_{i=t-w+1}^{t} (y_i - \bar{y}_t(w))^2} \tag{7}$$

where w is the window size (typically 30 days).
Interpretation Guidelines:

- **Stationary:** $\bar{y}_t(w)$ and $s_t(w)$ fluctuate around constant levels
- **Non-stationary:** $\bar{y}_t(w)$ exhibits clear trend; $s_t(w)$ shows systematic variation

Key Takeaways

- Price rolling statistics drift significantly (non-stationary evidence)
- Return rolling statistics fluctuate around stable levels (stationary evidence)
- Visual diagnostics should always accompany formal statistical tests
- Window size selection affects smoothness vs. responsiveness trade-off

4 Model Identification

4.1 Cell 5: ACF and PACF Analysis

Theoretical Background

The Box-Jenkins methodology relies on ACF and PACF plots to identify appropriate ARIMA orders:

- **ACF:** Shows correlation between y_t and y_{t-k} at all lags
- **PACF:** Shows correlation between y_t and y_{t-k} after removing intermediate lag effects

These functions reveal the temporal dependence structure and guide model specification.

Mathematical Formulation

Autocorrelation Function (ACF):

$$\rho_k = \frac{\gamma_k}{\gamma_0} = \frac{\text{Cov}(y_t, y_{t-k})}{\text{Var}(y_t)} \tag{8}$$

Sample estimator:

$$\hat{\rho}_k = \frac{\sum_{t=k+1}^{T} (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2} \tag{9}$$

Partial Autocorrelation Function (PACF):
The PACF at lag k, denoted ϕ_{kk}, is the last coefficient in the regression:

$$y_t = \phi_{k1} y_{t-1} + \phi_{k2} y_{t-2} + \cdots + \phi_{kk} y_{t-k} + \epsilon_t \tag{10}$$

Significance Bounds:
Under white noise hypothesis, approximately 95% of sample autocorrelations fall within:

$$\pm 1.96/\sqrt{T} \tag{11}$$

Model Identification Rules:

ACF Pattern	PACF Pattern	Model
Cuts off after lag q	Decays gradually	MA(q)
Decays gradually	Cuts off after lag p	AR(p)
Decays gradually	Decays gradually	ARMA(p, q)
No significant lags	No significant lags	White Noise

Key Takeaways

- Cryptocurrency returns often resemble white noise (efficient market hypothesis)
- Significant lags suggest exploitable autocorrelation patterns
- ACF/PACF provide initial guidance, but model selection requires formal criteria
- Financial returns rarely exhibit textbook ACF/PACF patterns

5 ARIMA Model Estimation

5.1 Cell 6: ARIMA Model Fitting and Selection

Theoretical Background

ARIMA(p, d, q) models generalize AR, MA, and ARMA processes to handle non-stationary data through differencing. The model combines:

- **AR(p):** Autoregressive component using p past values
- **I(d):** Integration order (number of differences for stationarity)
- **MA(q):** Moving average component using q past errors

For stationary returns, $d = 0$, reducing to ARMA(p, q).

Mathematical Formulation

General ARIMA(p, d, q) Model:

$$\phi(B)(1 - B)^d y_t = \theta(B)\epsilon_t \tag{12}$$

where:

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \quad \text{(AR polynomial)} \tag{13}$$

$$\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q \quad \text{(MA polynomial)} \tag{14}$$

$$B = \text{backshift operator: } By_t = y_{t-1} \tag{15}$$

Specific Models:
AR(1):

$$y_t = \phi_1 y_{t-1} + \epsilon_t \tag{16}$$

Stationary if $|\phi_1| < 1$.
MA(1):

$$y_t = \epsilon_t + \theta_1 \epsilon_{t-1} \tag{17}$$

Always stationary; invertible if $|\theta_1| < 1$.
ARMA(1,1):

$$y_t = \phi_1 y_{t-1} + \epsilon_t + \theta_1 \epsilon_{t-1} \tag{18}$$

Model Selection Criteria:

Akaike Information Criterion:

$$\text{AIC} = -2\ln(\mathcal{L}) + 2k \tag{19}$$

Bayesian Information Criterion:

$$\text{BIC} = -2\ln(\mathcal{L}) + k\ln(n) \tag{20}$$

where:

- $\mathcal{L}$ = maximized likelihood
- k = number of parameters
- n = sample size

BIC penalizes complexity more heavily than AIC. Select model with **lowest** AIC/BIC.

Key Takeaways

- Cryptocurrency returns often require low-order ARIMA models
- Model parsimony (simplicity) is preferred for out-of-sample forecasting
- AIC/BIC balance goodness-of-fit against model complexity
- Overfitting leads to poor forecasting performance

6 Model Diagnostics

6.1 Cell 7: Residual Analysis and Validation

Theoretical Background

A well-specified ARIMA model produces residuals that are **white noise**:

1. $E[\epsilon_t] = 0$ (zero mean)
2. $\text{Var}(\epsilon_t) = \sigma^2$ (constant variance)
3. $\text{Cov}(\epsilon_t, \epsilon_s) = 0$ for $t \neq s$ (no autocorrelation)
4. $\epsilon_t \sim N(0, \sigma^2)$ (normality, for inference)

Residual diagnostics detect model misspecification and assumption violations.

Mathematical Formulation

Ljung-Box Test (Autocorrelation):
Tests joint significance of first h autocorrelations:

$$Q(h) = n(n+2)\sum_{k=1}^{h}\frac{\hat{\rho}_k^2}{n-k} \sim \chi^2_{h-p-q} \tag{21}$$

Hypotheses:

- H_0: No autocorrelation up to lag h
- H_1: Some autocorrelation exists

If p-value > 0.05: Residuals are white noise ✓
Jarque-Bera Test (Normality):

$$JB = \frac{n}{6}\left[S^2 + \frac{(K-3)^2}{4}\right] \sim \chi^2_2 \tag{22}$$

where S = sample skewness, K = sample kurtosis.
ARCH-LM Test (Heteroskedasticity):
Tests for autoregressive conditional heteroskedasticity:

1. Regress $\hat{\epsilon}_t^2$ on lagged $\hat{\epsilon}_{t-i}^2$
2. Test: H_0: all coefficients = 0

Test statistic:

$$\text{ARCH-LM} = nR^2 \sim \chi_q^2 \tag{23}$$

If p-value < 0.05: ARCH effects present $\rightarrow$ Use GARCH model.

Key Takeaways

- Ljung-Box test detects remaining autocorrelation structure
- ARCH effects indicate volatility clustering—common in crypto markets
- Normality violations affect inference but not point estimates
- Failed diagnostics suggest model respecification or robust methods

7 Volatility Modeling

7.1 Cell 8: GARCH Models for Volatility Clustering

Theoretical Background

Financial returns exhibit **volatility clustering**: periods of high volatility followed by high volatility, and similarly for low volatility. ARIMA models assume constant variance—inadequate for crypto markets.

GARCH (Generalized Autoregressive Conditional Heteroskedasticity) models capture time-varying volatility:

- Essential for risk management (VaR, CVaR)
- Improves option pricing accuracy
- Enables volatility forecasting
- Supports dynamic hedging strategies

Mathematical Formulation

GARCH(p, q) Specification:

Mean Equation:

$$r_t = \mu + \epsilon_t, \quad \epsilon_t = \sigma_t z_t, \quad z_t \sim N(0,1) \tag{24}$$

Variance Equation:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 \tag{25}$$

Constraints:

- $\omega > 0$
- $\alpha_i \geq 0$, $\beta_j \geq 0$
- $\sum_{i=1}^{\max(p,q)} (\alpha_i + \beta_i) < 1$ (stationarity)

GARCH(1,1)—Most Common:

$$\sigma_t^2 = \omega + \alpha \epsilon_{t-1}^2 + \beta \sigma_{t-1}^2 \tag{26}$$

Interpretation:

- α: Impact of past shocks on current volatility
- β: Persistence of past volatility

- $\alpha + \beta \approx 1$: High persistence (shocks decay slowly)

Unconditional Variance:
For stationary GARCH(1,1):

$$E[\sigma_t^2] = \frac{\omega}{1 - \alpha - \beta} \tag{27}$$

Volatility Forecasting:
h-step ahead variance forecast:

$$\hat{\sigma}^2_{T+h|T} = \bar{\sigma}^2 + (\alpha + \beta)^{h-1}(\sigma^2_{T+1|T} - \bar{\sigma}^2) \tag{28}$$

Converges to unconditional variance as $h \to \infty$.

Key Takeaways

- Cryptocurrencies exhibit strong volatility clustering
- GARCH models capture fat tails and leptokurtosis in returns
- High $\alpha + \beta$ indicates volatility shocks persist
- GARCH improves risk forecasts but may not improve return predictions

8 Forecasting and Simulation

8.1 Cell 9: Monte Carlo Price Forecasting

Theoretical Background

Combining ARIMA (mean) and GARCH (volatility) enables comprehensive forecasting:

1. **ARIMA:** Forecasts expected returns
2. **GARCH:** Forecasts conditional volatility
3. **Monte Carlo:** Generates distribution of future price paths

This approach quantifies forecast uncertainty through confidence intervals—critical for risk management.

Mathematical Formulation

Price Reconstruction from Log Returns:
Given log return forecast $\hat{r}_{T+h}$ and volatility forecast $\hat{\sigma}_{T+h}$:

$$P_{T+h} = P_{T+h-1} \exp(\hat{r}_{T+h} + \hat{\sigma}_{T+h} z_{T+h}) \tag{29}$$

where $z_{T+h} \sim N(0, 1)$.
Monte Carlo Algorithm:
For each simulation $i = 1, \ldots, N$:

1. Initialize: $P_{i,0} = P_T$ (current price)
2. For each forecast step $h = 1, \ldots, H$:

$$P_{i,h} = P_{i,h-1} \exp(\mu_h + \sigma_h z_{i,h}) \tag{30}$$

where:

- μ_h = ARIMA forecast at horizon h
- σ_h = GARCH volatility forecast at horizon h
- $z_{i,h} \sim N(0, 1)$ (independent random shock)

Percentile-Based Confidence Intervals:
For each time h, compute across all N simulations:

$$P_h^{(p)} = \text{percentile}_p(\{P_{1,h}, P_{2,h}, \ldots, P_{N,h}\}) \tag{31}$$

Standard intervals:

- 90% CI: $[P_h^{(5)}, P_h^{(95)}]$
- 50% CI: $[P_h^{(25)}, P_h^{(75)}]$
- Median forecast: $P_h^{(50)}$

Expected Return:

$$\text{Expected Return} = \frac{P_H^{(50)} - P_0}{P_0} \times 100\% \tag{32}$$

Key Takeaways

- Monte Carlo captures full distribution of future outcomes
- Wide confidence intervals reflect high uncertainty in crypto markets
- Median forecast often differs from mean due to skewness
- Volatility forecasts are more reliable than return forecasts

9 Model Evaluation

9.1 Cell 10: Backtesting and Performance Metrics

Theoretical Background

Model validation requires out-of-sample testing to avoid overfitting. The rolling forecast procedure:

1. Split data: 80% training, 20% testing
2. Re-estimate model at each forecast step
3. Generate one-step-ahead predictions
4. Compare against realized values

This simulates real-world forecasting where models adapt to new information.

Mathematical Formulation

Error Metrics:
Root Mean Squared Error:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2} \tag{33}$$

Mean Absolute Error:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t| \tag{34}$$

Mean Absolute Percentage Error:

$$\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \tag{35}$$

Directional Accuracy:
Percentage of correct direction predictions:

$$\text{DA} = \frac{1}{n}\sum_{t=1}^{n} 1[\text{sign}(y_t - y_{t-1}) = \text{sign}(\hat{y}_t - y_{t-1})] \times 100\% \tag{36}$$

In efficient markets, DA $\approx 50\%$.
Coefficient of Determination:

$$R^2 = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2} \tag{37}$$

Naive Forecast Benchmark:
Random walk: $\hat{y}_t^{\text{naive}} = y_{t-1}$
Improvement over Naive:

$$\text{Improvement} = \frac{\text{RMSE}_{\text{naive}} - \text{RMSE}_{\text{model}}}{\text{RMSE}_{\text{naive}}} \times 100\% \tag{38}$$

Key Takeaways

- Out-of-sample evaluation prevents overfitting bias
- Directional accuracy around 50% indicates efficient markets
- RMSE penalizes large errors more than MAE
- Beating naive forecasts is challenging for returns

10 Summary and Conclusions

10.1 Key Learning Outcomes

Theoretical Mastery:

- Stationarity testing provides rigorous foundation for time-series modeling
- ARIMA models capture linear temporal dependencies in returns
- GARCH models address volatility clustering in cryptocurrency markets
- Monte Carlo methods quantify forecast uncertainty comprehensively

Practical Implementation:

- Log returns transformation ensures stationarity for modeling
- ACF/PACF analysis guides initial model specification
- Information criteria (AIC/BIC) enable systematic model selection
- Residual diagnostics validate model assumptions
- Rolling forecasts provide realistic out-of-sample evaluation

DeFi Applications:

- Volatility forecasting for dynamic position sizing
- Risk management through VaR and Expected Shortfall
- Optimal rebalancing strategies based on return predictions
- Liquidity provision pricing in decentralized exchanges

10.2 Critical Limitations and Considerations

Model Limitations:

- Linear ARIMA may miss nonlinear dependencies
- GARCH assumes symmetric volatility response to shocks
- Normal distribution assumption fails to capture fat tails
- Parameter instability during regime shifts

Cryptocurrency-Specific Challenges:

- Limited historical data for model calibration
- Extreme events (flash crashes) not well-captured
- Regulatory announcements cause structural breaks
- Market manipulation and wash trading affect patterns

Practical Implementation Issues:

- Transaction costs and slippage ignored in academic models
- Model risk from specification uncertainty
- Computational intensity of Monte Carlo simulation
- Real-time implementation challenges

10.3 Future Research Directions

Methodological Extensions:

- Asymmetric GARCH models (EGARCH, GJR-GARCH)
- Long-memory models (ARFIMA, FIGARCH)
- Regime-switching frameworks
- Machine learning for nonlinear forecasting

DeFi-Specific Applications:

- On-chain metrics as exogenous variables
- Cross-cryptocurrency volatility spillovers
- DeFi protocol-specific risk factors
- Stablecoin de-pegging prediction

11 References and Further Reading

Foundational Time-Series Econometrics:

- Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time Series Analysis: Forecasting and Control* (5th ed.). Wiley.
- Hamilton, J. D. (1994). *Time Series Analysis*. Princeton University Press.
- Tsay, R. S. (2010). *Analysis of Financial Time Series* (3rd ed.). Wiley.

GARCH Models:

- Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3), 307-327.
- Engle, R. F. (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica*, 50(4), 987-1007.
- Hansen, P. R., & Lunde, A. (2005). A forecast comparison of volatility models. *Journal of Econometrics*, 131(1-2), 97-121.

Cryptocurrency Forecasting:

- Fang, F., Ventre, C., Basios, M., et al. (2022). Cryptocurrency trading: A comprehensive survey. *Financial Innovation*, 8(1), 1-59.
- Katsiampa, P. (2017). Volatility estimation for Bitcoin: A comparison of GARCH models. *Economics Letters*, 158, 3-6.
- Peng, Y., Albuquerque, P. H. M., et al. (2018). The best of two worlds: Forecasting high frequency volatility for cryptocurrencies. *Journal of Economic Dynamics and Control*, 100, 1-19.

Computational Tools:

- Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. *Proceedings of the 9th Python in Science Conference*.
- Sheppard, K. (2021). *ARCH Documentation*. Retrieved from `https://arch.readthedocs.io`

12 Appendix: Technical Implementation

12.1 A.1: Python Implementation Guidelines

Key Libraries:

```
import numpy as np
import pandas as pd
from statsmodels.tsa.arima.model import ARIMA
from arch import arch_model
from statsmodels.tsa.stattools import adfuller, kpss
```

ARIMA Estimation Example:

```
model = ARIMA(returns, order=(p, d, q))
fitted = model.fit()
forecast = fitted.forecast(steps=30)
```

GARCH Estimation Example:

```
garch = arch_model(returns*100, vol='GARCH', p=1, q=1)
garch_fitted = garch.fit(disp='off')
variance_forecast = garch_fitted.forecast(horizon=30)
```

12.2 A.2: Computational Considerations

Performance Optimization:

- Vectorize operations using NumPy for efficiency
- Cache repeated calculations (e.g., log transformations)
- Use parallel processing for Monte Carlo simulations
- Implement early stopping for convergence

Numerical Stability:

- Scale returns to percentages for GARCH (multiply by 100)
- Check optimizer convergence warnings
- Validate parameter constraints ($\alpha + \beta < 1$)
- Handle edge cases (zero variance periods)

12.3 A.3: Model Deployment Considerations

Production Requirements:

- Automated model retraining schedules
- Real-time data pipelines for live forecasting
- Model versioning and performance tracking
- Fail-safe mechanisms for extreme market conditions

Risk Management Integration:

- Position sizing based on volatility forecasts
- Stop-loss placement using confidence intervals
- Portfolio rebalancing triggers
- Stress testing under historical crisis scenarios

Regulatory Compliance:

- Model documentation and audit trails
- Backtesting results with confidence intervals
- Assumption validation and limitation disclosure
- Independent model validation requirements

13 Practical Exercises

13.1 Exercise 1: Stationarity Testing

Test the stationarity of Bitcoin prices and returns using both ADF and KPSS tests. Interpret the results and explain why returns are preferred for modeling.

13.2 Exercise 2: Model Selection

Fit ARIMA models with orders (0,0,0), (1,0,0), (0,0,1), and (1,0,1) to Ethereum returns. Compare AIC and BIC values to select the best model.

13.3 Exercise 3: Volatility Forecasting

Estimate a GARCH(1,1) model for Dogecoin returns. Calculate the 30-day ahead volatility forecast and interpret the persistence parameter ($\alpha + \beta$).

13.4 Exercise 4: Monte Carlo Simulation

Generate 1000 price paths for Bitcoin over a 30-day horizon using combined ARIMA-GARCH forecasts. Calculate the 90% confidence interval and probability of positive returns.

13.5 Exercise 5: Backtesting

Implement rolling window forecasting for your chosen cryptocurrency. Compare RMSE and directional accuracy against a naive random walk benchmark.

14 Conclusion

Week 4's comprehensive treatment of time-series forecasting provides essential tools for cryptocurrency market analysis. The integration of classical econometric methods (ARIMA) with volatility modeling (GARCH) creates a robust framework for:

- Understanding temporal dependencies in returns
- Forecasting conditional volatility
- Quantifying forecast uncertainty
- Supporting risk management decisions

While these models have limitations—particularly in capturing extreme events and nonlinear dynamics—they remain foundational for quantitative finance. The rigorous testing procedures and diagnostic frameworks ensure model validity and guide interpretation.

Future work should extend these methods to incorporate:

- On-chain analytics and sentiment data
- Regime-switching frameworks for structural breaks
- Machine learning for nonlinear pattern detection
- Cross-asset volatility spillovers

The skills developed here—stationarity testing, model specification, diagnostic analysis, and forecasting—are transferable across asset classes and form the basis for more advanced quantitative strategies in decentralized finance.