Comparison of Strategies and ML Methsd for Cryptocurrency and Traditional Equities.

Manav Agarwal

00. Data API and Interface

This notebook establishes and tests our data collection pipeline for the FinML final project. Since I built my project in a modular fashion, not all functions are here with full code, they are imported from the src folder. I will explain a few algorithms in this and other notebooks as things progress and to make things more readable.

Objectives:

1. Test available data sources (Polygon, CMC, yfinance, etc.)
2. Collect 2+ years of historical data (Jan 2023 - Aug 2025)
3. Validate data quality and completeness
4. Establish caching and fallback
5. Create data interface

0. Setup/Imports

In [1]:

```python
import sys
import os
sys.path.append('../src')
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import json
from pathlib import Path
import logging
import warnings
warnings.filterwarnings('ignore')

# Custom modules
from data.polygon_s3_collector import PolygonS3Collector
from data.unified_collector import UnifiedDataCollector
from data.batch_collect_data import BatchDataCollector

# Setup logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(mes
logger = logging.getLogger(__name__)
```

1. Test Individual Data Sources

In [2]:

```python
def test_polygon_s3():
    print("Testing Polygon S3 Access")
    collector = PolygonS3Collector()
    # Test crypto data
    crypto_test = collector.fetch_crypto_day(
        ticker="X:BTCUSD", date="2024-01-01")
    if crypto_test is not None and not crypto_test.empty:
        print(f"[OK] Crypto data: {len(crypto_test)} records")
        print(f"Date range: {crypto_test.index.min()} to {crypto_test.index.max()}"
        print(f"Columns: {list(crypto_test.columns)}")
    else:
        print("[X] Failed to fetch crypto data")
    # Test equity data
    equity_test = collector.fetch_stock_day(
        ticker="SPY",
        date="2024-01-02")  # Market closed Jan 1)
    if equity_test is not None and not equity_test.empty:
        print(f"[OK] Equity data: {len(equity_test)} records")
        print(f"  Date range: {equity_test.index.min()} to {equity_test.index.max()
    else:
        print("[X] Failed to fetch equity data")
    return collector
polygon_collector = test_polygon_s3()
```

```
Testing Polygon S3 Access
Initialized Polygon S3 client
Endpoint: https://files.polygon.io
Bucket: flatfiles
Loading from cache: data\s3_cache\crypto_2024-01-01.parquet
[OK] Crypto data: 1 records
Date range: 0 to 0
Columns: ['timestamp', 'open', 'high', 'low', 'close', 'volume', 'symbol']
Loading from cache: data\s3_cache\stocks_2024-01-02.parquet
[OK] Equity data: 2 records
  Date range: 0 to 1
```

2. Define Target Symbols and Date Ranges

In [3]:

```python
config = {
    "crypto_symbols": ["BTCUSD", "ETHUSD", "SOLUSD", "ADAUSD", "XRPUSD"],
    "equity_symbols": ["SPY", "QQQ", "IWM", "DIA", "VTI"],
    "start_date": "2023-01-01",
    "end_date": "2025-08-01",
    "regime_change_start": "2025-01-01",  # Reserved for regime analysis
    "cache_dir": "../data/ml_comparison_cache"
}

print("Data Collection Configuration:")
print(f"  Crypto: {config['crypto_symbols']}")
print(f"  Equity: {config['equity_symbols']}")
print(f"  Period: {config['start_date']} to {config['end_date']}")
print(f"  Reserved regime: {config['regime_change_start']} onwards")
```

```
Data Collection Configuration:
  Crypto: ['BTCUSD', 'ETHUSD', 'SOLUSD', 'ADAUSD', 'XRPUSD']
  Equity: ['SPY', 'QQQ', 'IWM', 'DIA', 'VTI']
  Period: 2023-01-01 to 2025-08-01
  Reserved regime: 2025-01-01 onwards
```

3. Batch Data Collection

In [4]:

```
# Initialize batch collector
batch_collector = BatchDataCollector(cache_dir=config['cache_dir'])

print(f"Batch collector initialized")
print(f"Cache directory: {batch_collector.cache_dir}")
```

```
INFO:data.batch_collect_data:Using device: cpu
Initialized Polygon S3 client
Endpoint: https://files.polygon.io
Bucket: flatfiles
Batch collector initialized
Cache directory: ..\data\ml_comparison_cache
```

In [ ]:

```
def collect_sample_data():
    print("Collecting Sample Data (1 month)")
    sample_start = "2024-01-01"
    sample_end = "2024-01-31"
    results = {}
    # Collect crypto
    for symbol in config['crypto_symbols'][:2]: e
        print(f"\nCollecting {symbol}...")
        ticker = f"X:{symbol}"
        data = batch_collector.crypto_collector.fetch_aggregated_data(symbol=symbol
            start=sample_start, end=sample_end, market='crypto', timeframe='hour')
        if data is not None and not data.empty:
            results[symbol] = data
            print(f"  [OK] {len(data)} records collected")
            print(f"  Date range: {data.index.min()} to {data.index.max()}")
        else:
            print(f"  [X] Failed to collect {symbol}")
    # Just SPY and QQQ for sample equity/indice
    for symbol in config['equity_symbols'][:2]:
        print(f"\nCollecting {symbol}...")
        data = batch_collector.crypto_collector.fetch_aggregated_data(
            symbol=symbol, start=sample_start, end=sample_end, market='stocks', tim
        if data is not None and not data.empty:
            results[symbol] = data
            print(f"  [OK] {len(data)} records collected")
        else:
            print(f"  [X] Failed to collect {symbol}")
    return results

sample_data = collect_sample_data()
```

We get the continuous crypto data and "No data available for 2024-01-01 in stocks" due to holiday, looks like its working like expected.

4. Data Quality Analysis

In [6]:
```
def analyze_data_quality(data_dict):
    print("Data Quality Analysis")
    print("="*130)
    quality_report = pd.DataFrame()
    for symbol, data in data_dict.items():
        report = {
            'symbol': symbol, 'records': len(data), 'start_date': data.index.min(),
            'end_date': data.index.max(), 'missing_values': data.isnull().sum().sum
            'missing_pct': (data.isnull().sum().sum() / (len(data) * len(data.colum
            'zero_volume_pct': (data['volume'] == 0).sum() / len(data) * 100 if 'vo
            'price_range': f"${data['close'].min():.2f} - ${data['close'].max():.2f
            'avg_daily_volume': data['volume'].mean() if 'volume' in data.columns e
        quality_report = pd.concat([quality_report, pd.DataFrame([report])], ignore
    print(quality_report.to_string())
    return quality_report
if sample_data:
    quality_report = analyze_data_quality(sample_data)
```

```
Data Quality Analysis
====================================================================================
==============================================
   symbol  records  start_date  end_date  missing_values  missing_pct  zero_volume_p
ct             price_range  avg_daily_volume
0  BTCUSD      744          0       743               0      0.00000         0.0000
00  $38787.00 - $48633.40      9.894472e+02
1  ETHUSD      744          0       743               0      0.00000         0.0000
00    $2183.61 - $2706.55      6.016779e+03
2     SPY       31          0        30              20      9.21659        16.1290
32      $467.75 - $491.05      5.311149e+07
3     QQQ       31          0        30              20      9.21659        16.1290
32      $396.54 - $428.49      3.086514e+07
```

5. Feature Engineering Test/Preview

In [7]:
```
def preview_features():
    from features.feature_engineering import FeatureEngineer
    print("Feature Engineering Preview")
    print("="*110)

    fe = FeatureEngineer()
    if 'BTCUSD' in sample_data:
        btc_data = sample_data['BTCUSD'].copy()
        # Engineer features
        btc_features = fe.create_features(btc_data, 'BTCUSD')
        print(f"Original columns: {len(btc_data.columns)}")
        print(f"After feature engineering: {len(btc_features.columns)}")
        feature_categories = {
            'Price': [col for col in btc_features.columns if 'return' in col or 'pr
            'Volume': [col for col in btc_features.columns if 'volume' in col],
            'Technical': [col for col in btc_features.columns if any(ind in col for
            'Volatility': [col for col in btc_features.columns if 'volatility' in c
            'Market Structure': [col for col in btc_features.columns if any(ms in c
        for category, features in feature_categories.items():
            print(f"\n{category} Features ({len(features)}):")
            print(f"  {features[:5]}..." if len(features) > 5 else f"  {features}")
        return btc_features
    return None

btc_features = preview_features()
```

```
Feature Engineering Preview
====================================================================================
==========================
Original columns: 7
After feature engineering: 107

Price Features (7):
  ['returns', 'log_returns', 'price_position', 'price_range_position_7', 'price_rang
e_position_14']...

Volume Features (9):
  ['volume', 'volume_sma_7', 'volume_ratio_7', 'volume_sma_14', 'volume_ratio_1
4']...

Technical Features (29):
  ['sma_7', 'ema_7', 'close_to_sma_7', 'close_to_ema_7', 'sma_14']...

Volatility Features (10):
  ['atr', 'atr_ratio', 'volatility_7', 'volatility_ann_7', 'volatility_14']...

Market Structure Features (0):
  []
```

6. Correlation Analysis

In [8]:

```
def analyze_correlations():
    print("Asset Correlation Analysis")
    print("="*40)
    if len(sample_data) < 2:
        print("Need at least 2 assets for correlation analysis")
        return None
    # Prepare returns data
    returns_data = pd.DataFrame()
    for symbol, data in sample_data.items():
        if 'close' in data.columns:
            returns = data['close'].pct_change().dropna()
            returns_data[symbol] = returns
    # Calculate correlation matrix
    if not returns_data.empty:
        # Align indices
        returns_data = returns_data.dropna()
        if len(returns_data) > 0:
            corr_matrix = returns_data.corr()
            print("\nCorrelation Matrix:")
            print(corr_matrix.round(3).to_string())
            # Find highest correlations
            corr_pairs = []
            for i in range(len(corr_matrix.columns)):
                for j in range(i+1, len(corr_matrix.columns)):
                    corr_pairs.append({
                        'pair': f"{corr_matrix.columns[i]}-{corr_matrix.columns[j]}
                        'correlation': corr_matrix.iloc[i, j]})
            corr_pairs = sorted(corr_pairs, key=lambda x: abs(x['correlation']), re
            print("\nTop Correlations:")
            for pair in corr_pairs[:5]:
                print(f"  {pair['pair']}: {pair['correlation']:.3f}")
            return corr_matrix
    return None

corr_matrix = analyze_correlations()
```

```
Asset Correlation Analysis
========================================

Correlation Matrix:
        BTCUSD  ETHUSD    SPY    QQQ
BTCUSD   1.000   0.833  0.043 -0.048
ETHUSD   0.833   1.000  0.032 -0.076
SPY      0.043   0.032  1.000  0.916
QQQ     -0.048  -0.076  0.916  1.000

Top Correlations:
  SPY-QQQ: 0.916
  BTCUSD-ETHUSD: 0.833
  ETHUSD-QQQ: -0.076
  BTCUSD-QQQ: -0.048
  BTCUSD-SPY: 0.043
```

7. Full Data Collection Setup

In [ ]:

```
def collect_crypto_from_yfinance(symbol, start_date, end_date):
    import yfinance as yf

    yf_mapping = {
        'BTCUSD': 'BTC-USD',
        'ETHUSD': 'ETH-USD',
        'SOLUSD': 'SOL-USD',
        'XRPUSD': 'XRP-USD',
        'ADAUSD': 'ADA-USD'
    }

    if symbol not in yf_mapping:
        return None

    try:
        ticker = yf.Ticker(yf_mapping[symbol])
        data = ticker.history(start=start_date, end=end_date, interval='1h')

        if not data.empty:
            # Standardize column names
            data.columns = [col.lower() for col in data.columns]
            data['symbol'] = symbol
            return data
    except Exception as e:
        print(f"yfinance error for {symbol}: {str(e)[:50]}")
        return None

def merge_data_sources(base_data, new_data):
    if not isinstance(base_data.index, pd.DatetimeIndex):
        if 'timestamp' in base_data.columns:
            base_data.index = pd.to_datetime(base_data['timestamp'])

    if not isinstance(new_data.index, pd.DatetimeIndex):
        if 'timestamp' in new_data.columns:
            new_data.index = pd.to_datetime(new_data['timestamp'])

    combined = pd.concat([base_data, new_data])
    combined = combined[~combined.index.duplicated(keep='first')]
    combined = combined.sort_index()

    return combined

def collect_with_multi_source_fallback(symbol, start_date, end_date, cache_dir="../
    from data.collect_ml_comparison_data import MLComparisonDataCollector
    from data.unified_collector import UnifiedDataCollector
    from data.polygon_s3_collector import PolygonS3Collector
    from datetime import timedelta
    print(f"Collecting {symbol} with fallback")
    all_data_sources = {}
    try:
        print("1. Trying MLComparisonDataCollector...")
        ml_collector = MLComparisonDataCollector(cache_dir=cache_dir)
        ml_data = ml_collector.collect_comparison_data()
        if symbol in ml_data and ml_data[symbol] is not None and not ml_data[symbol
            all_data_sources['ml_collector'] = ml_data[symbol]
            print(f"Got {len(ml_data[symbol])} records")
        else:
            print(f"No data from MLComparisonDataCollector")
    except Exception as e:
        print(f"Error: {str(e)[:50]}")

    # 2. Try Polygon S3 directly
    try:
        print("2. Trying Polygon S3...")
        if symbol in ['BTCUSD', 'ETHUSD', 'SOLUSD', 'ADAUSD', 'XRPUSD']:
            polygon_collector = PolygonS3Collector(cache_dir=cache_dir)
            # Use the proper ticker format
            ticker_mapping = {
                'BTCUSD': 'X:BTC-USD',
                'ETHUSD': 'X:ETH-USD',
                'SOLUSD': 'X:SOL-USD',
                'XRPUSD': 'X:XRP-USD',
                'ADAUSD': 'X:ADA-USD'
            }
            ticker = ticker_mapping.get(symbol, f"X:{symbol}")
            try:
                bulk_data = polygon_collector.fetch_bulk_historical(
                    symbols=[symbol],
                    start=start_date,
                    end=end_date,
                    market='crypto'
                )
                if symbol in bulk_data and not bulk_data[symbol].empty:
                    all_data_sources['polygon'] = bulk_data[symbol]
                    print(f"Got {len(bulk_data[symbol])} records")
                else:
                    print(f"No data from Polygon S3")
            except Exception as e:
                print(f"Polygon S3 error: {str(e)[:50]}")
    except Exception as e:
        print(f"Error: {str(e)[:50]}")

    try:
        print("3. Trying UnifiedDataCollector...")
        unified = UnifiedDataCollector(cache_dir=cache_dir)

        if symbol in ['BTCUSD', 'ETHUSD', 'SOLUSD', 'ADAUSD', 'XRPUSD']:
            timespan = 'hour'
        else:
            timespan = 'day'

        unified_data = unified.fetch_data(
            symbol=symbol,
            start=start_date,
            end=end_date,
            timespan=timespan
        )

        if unified_data is not None and not unified_data.empty:
            all_data_sources['unified'] = unified_data
            print(f"Got {len(unified_data)} records")
        else:
            print(f"No data from UnifiedDataCollector")
    except Exception as e:
        print(f"Error: {str(e)[:50]}")

    try:
        print("4. Trying yfinance...")
        yf_data = collect_crypto_from_yfinance(symbol, start_date, end_date)
        if yf_data is not None and not yf_data.empty:
            all_data_sources['yfinance'] = yf_data
            print(f"Got {len(yf_data)} records")
        else:
            print(f"No data from yfinance")
    except Exception as e:
        print(f"Error: {str(e)[:50]}")

    if not all_data_sources:
        print(f"\nFailed to collect any data for {symbol}")
        return None

    print(f"\nMerging {len(all_data_sources)} data sources...")

    sorted_sources = sorted(all_data_sources.items(), key=lambda x: len(x[1]), reve
    merged_data = sorted_sources[0][1].copy()
    print(f"   Base: {sorted_sources[0][0]} with {len(merged_data)} records")
    for source_name, source_data in sorted_sources[1:]:
        print(f"   Merging {source_name} ({len(source_data)} records)...")
        merged_data = merge_data_sources(merged_data, source_data)
    merged_data.columns = [col.lower() for col in merged_data.columns]
    required_cols = ['open', 'high', 'low', 'close', 'volume']
    for col in required_cols:
        if col not in merged_data.columns:
            print(f"   Warning: Missing column {col}, adding with zeros")
            merged_data[col] = 0
    if 'symbol' not in merged_data.columns:
        merged_data['symbol'] = symbol

    print(f"\n✓ Final merged data: {len(merged_data)} records")
    if hasattr(merged_data.index, 'min'):
        print(f"  Date range: {merged_data.index.min()} to {merged_data.index.max()
    return merged_data

def run_full_collection_enhanced(test_mode=False):
    print("FULL DATA COLLECTION")
    if test_mode:
        start = "2024-01-01"
        end = "2024-01-07"
        print(f"Test mode: Collecting {start} to {end}")
    else:
        start = config['start_date']
        end = config['end_date']
        print(f"Full mode: Collecting {start} to {end}")
        print("...")
    from data.collect_ml_comparison_data import MLComparisonDataCollector
    import yfinance as yf
    ml_collector = MLComparisonDataCollector(cache_dir=config['cache_dir'])
    unified = UnifiedDataCollector(cache_dir=config['cache_dir'])
    all_data = {}
    problem_symbols = ['SOLUSD', 'XRPUSD']
    print("\n1. Primary collection with MLComparisonDataCollector...")
    ml_data = ml_collector.collect_comparison_data()

    if ml_data:
        for symbol in config['crypto_symbols'] + config['equity_symbols']:
            if symbol in ml_data and ml_data[symbol] is not None and not ml_data[sy
                all_data[symbol] = ml_data[symbol]
                print(f"{symbol}: {len(ml_data[symbol])} records")

                if symbol in problem_symbols and len(ml_data[symbol]) < 10000:
                    print(f"{symbol} has incomplete data")
            else:
                print(f"{symbol}: No data from primary")

    # Special handling for problematic symbols
    print("\n2. Multi source collection for problems...")
    for symbol in problem_symbols:
        if symbol not in all_data or len(all_data.get(symbol, pd.DataFrame())) < 10
            print(f"\nCollecting {symbol} from multiple sources:")
            complete_data = collect_with_multi_source_fallback(
                symbol=symbol,
                start_date=start,
                end_date=end,
                cache_dir=config['cache_dir']
            )
            if complete_data is not None:
                # Replace or add the complete data
                all_data[symbol] = complete_data
                print(f"{symbol}: Successfully collected {len(complete_data)} recor
            else:
                print(f"{symbol}: Failed to collect sufficient data")

    failed_symbols = [s for s in config['crypto_symbols'] + config['equity_symbols'
                     if s not in all_data or all_data[s] is None or all_data[s].emp
    if failed_symbols:
        print(f"\n3. Standard fallback: {failed_symbols}")

        for symbol in failed_symbols:
            print(f"  Collecting {symbol}...")
            if symbol in config['crypto_symbols']:
                yf_data = collect_crypto_from_yfinance(symbol, start, end)
                if yf_data is not None and not yf_data.empty:
                    all_data[symbol] = yf_data
                    print(f"{symbol}: {len(yf_data)} records via yfinance")
                    continue
            timespan = 'hour' if symbol in config['crypto_symbols'] else 'day'
            alt_symbol = f"X:{symbol}" if symbol in config['crypto_symbols'] else s
            data = unified.fetch_data(
                symbol=alt_symbol,
                start=start,
                end=end,
                timespan=timespan,
                use_cache=True
            )
            if data is not None and not data.empty:
                all_data[symbol] = data
                print(f"{symbol}: {len(data)} records via UnifiedCollector")
            else:
                print(f"{symbol}: Failed all collection attempts")

    # Summary
    print("COLLECTION SUMMARY")
    print("="*80)
    if all_data:
        success_count = len(all_data)
        total_count = len(config['crypto_symbols']) + len(config['equity_symbols'])
        print(f"Successfully collected: {success_count}/{total_count} symbols")
        summary_data = []
        for symbol, data in all_data.items():
            if isinstance(data.index, pd.DatetimeIndex):
                start_date = data.index.min()
                end_date = data.index.max()
            else:
                start_date = pd.to_datetime(data.index.min())
                end_date = pd.to_datetime(data.index.max())

            # Calculate completeness
            days = (end_date - start_date).days
            if symbol in config['crypto_symbols']:
                expected_records = days * 24  # Hourly
            else:
                expected_records = days  # Daily
            completeness = (len(data) / expected_records * 100) if expected_records
            summary_data.append({
                'Symbol': symbol,
                'Records': len(data),
                'Start': start_date.strftime('%Y-%m-%d'),
                'End': end_date.strftime('%Y-%m-%d'),
                'Days': days,
                'Completeness': f"{completeness:.1f}%"
            })
        summary_df = pd.DataFrame(summary_data)
        print("\n" + summary_df.to_string(index=False))
        issues = summary_df[summary_df['Records'] < 5000]
        if not issues.empty:
            print(issues[['Symbol', 'Records', 'Completeness']].to_string(index=Fal
        else:
            print("\nAll symbols have sufficient data!")

        # Save metadata
        metadata = {
            'collection_date': datetime.now().isoformat(),
            'symbols': list(all_data.keys()),
            'start_date': start,
            'end_date': end,
            'test_mode': test_mode,
            'summary': summary_data
        }

        metadata_path = Path(config['cache_dir']) / 'collection_metadata.json'
        with open(metadata_path, 'w') as f:
            json.dump(metadata, f, indent=2)
        print(f"\nMetadata saved to {metadata_path}")

        # Update test_data global variable
        global test_data
        test_data = all_data

        return all_data
    else:
        print("Collection failed for all symbols")
        return None

print("Starting enhanced data collection with multi-source fallback...")
test_data = run_full_collection_enhanced(test_mode=False)
```

In [ ]:

```
def diagnose_data_issues():
    print("DATA AVAILABILITY DIAGNOSTIC")
    print("="*80)
    problem_symbols = ['SOLUSD', 'XRPUSD']
    for symbol in problem_symbols:
        print(f"\nDiagnosing {symbol}:")
        test_ranges = [
            ("2023-01-01", "2023-03-01"),
            ("2023-03-01", "2023-06-01"),
            ("2023-06-01", "2023-09-01"),
            ("2023-09-01", "2024-01-01"),
            ("2024-01-01", "2024-06-01"),
            ("2024-06-01", "2025-01-01"),
            ("2025-01-01", "2025-08-01")
        ]
        for start, end in test_ranges:
            try:
                ticker = f"X:{symbol}"
                data = polygon_collector.fetch_aggregated_data(
                    symbol=ticker,
                    start=start,
                    end=end,
                    market='crypto',
                    timeframe='hour'
                )
                if data is not None and not data.empty:
                    print(f"{start} to {end}: {len(data)} records found")
                else:
                    print(f"{start} to {end}: No data")

            except Exception as e:
                print(f"{start} to {end}: Error - {str(e)[:50]}")

        # Try alternative tickers
        print(f"\n  Testing alternative tickers for {symbol}:")
        alt_tickers = [
            f"X:{symbol}",
            symbol.replace("USD", "-USD"),
            symbol.replace("USD", ""),
            symbol[:3] + "-USD",
            symbol[:3]
        ]
        for alt in alt_tickers:
            try:
                data = unified.fetch_data(
                    symbol=alt,
                    start="2024-01-01",
                    end="2024-01-07",
                    timespan='hour',
                    use_cache=False
                )
                if data is not None and not data.empty:
                    print(f"'{alt}': {len(data)} records")
                else:
                    print(f"'{alt}': No data")
            except:
                print(f"'{alt}': Failed")

# Run diagnostics
diagnose_data_issues()
```

Backup Config

In [12]:

```
def collect_crypto_from_yfinance(symbol, start_date, end_date):
    """
    Collect crypto data from yfinance as fallback
    Formats to match our standard structure
    """
    import yfinance as yf

    # Map our symbols to yfinance tickers
    yf_mapping = {
        'SOLUSD': 'SOL-USD',
        'XRPUSD': 'XRP-USD',
        'BTCUSD': 'BTC-USD',
        'ETHUSD': 'ETH-USD',
        'ADAUSD': 'ADA-USD'
    }

    if symbol not in yf_mapping:
        print(f"No yfinance mapping for {symbol}")
        return None

    yf_ticker = yf_mapping[symbol]
    print(f"Fetching {symbol} from yfinance as {yf_ticker}...")

    try:
        # Download data with hourly intervals if available
        ticker = yf.Ticker(yf_ticker)

        # Try hourly data first (1h interval)
        try:
            data = ticker.history(
                start=start_date,
                end=end_date,
                interval="1h",
                auto_adjust=True,
                prepost=True
            )
        except:
            # Fall back to daily data if hourly not available
            print(f"  Hourly data not available, using daily...")
            data = ticker.history(
                start=start_date,
                end=end_date,
                interval="1d",
                auto_adjust=True
            )

            # Resample to hourly (forward fill)
            if not data.empty:
                # Create hourly index
                hourly_index = pd.date_range(
                    start=data.index[0],
                    end=data.index[-1] + pd.Timedelta(hours=23),
                    freq='h'
                )
                # Reindex and forward fill
                data = data.reindex(hourly_index, method='ffill')

        if data.empty:
            print(f"  No data returned from yfinance")
            return None

        # Format to match our standard structure
        formatted_data = pd.DataFrame({
            'open': data['Open'],
            'high': data['High'],
            'low': data['Low'],
            'close': data['Close'],
            'volume': data['Volume'],
            'symbol': symbol
        })

        # Add timestamp column if index is not already datetime
        if not isinstance(formatted_data.index, pd.DatetimeIndex):
            formatted_data.index = pd.to_datetime(formatted_data.index)

        formatted_data['timestamp'] = formatted_data.index

        print(f"Retrieved {len(formatted_data)} records from yfinance")
        return formatted_data

    except Exception as e:
        print(f"yfinance error: {str(e)[:100]}")
        return None

# Test yfinance fallback for SOL and XRP
print("Testing yfinance fallback for problematic symbols:")
print("-" * 60)

test_symbols = ['SOLUSD', 'XRPUSD']
yf_data = {}

for symbol in test_symbols:
    data = collect_crypto_from_yfinance(
        symbol=symbol,
        start_date=config['start_date'],
        end_date=config['end_date']
    )
    if data is not None:
        yf_data[symbol] = data
        print(f"{symbol}: Start={data.index.min()}, End={data.index.max()}, Records
    else:
        print(f"{symbol}: Failed to retrieve from yfinance")
```

```
Testing yfinance fallback for problematic symbols:
------------------------------------------------------------
Fetching SOLUSD from yfinance as SOL-USD...
```

```
ERROR:yfinance:$SOL-USD: possibly delisted; no price data found  (1h 2023-01-01 -> 2
025-08-01) (Yahoo error = "1h data not available for startTime=1672531200 and endTim
e=1754006400. The requested range must be within the last 730 days.")
```

```
  No data returned from yfinance
SOLUSD: Failed to retrieve from yfinance
Fetching XRPUSD from yfinance as XRP-USD...
```

```
ERROR:yfinance:$XRP-USD: possibly delisted; no price data found  (1h 2023-01-01 -> 2
025-08-01) (Yahoo error = "1h data not available for startTime=1672531200 and endTim
e=1754006400. The requested range must be within the last 730 days.")
```

```
  No data returned from yfinance
XRPUSD: Failed to retrieve from yfinance
```

In [13]:

```python
def merge_data_sources(primary_data, fallback_data):
    if primary_data is None or primary_data.empty:
        return fallback_data
    if fallback_data is None or fallback_data.empty:
        return primary_data
    if not isinstance(primary_data.index, pd.DatetimeIndex):
        primary_data.index = pd.to_datetime(primary_data.index)
    if not isinstance(fallback_data.index, pd.DatetimeIndex):
        fallback_data.index = pd.to_datetime(fallback_data.index)
    primary_start = primary_data.index.min()
    primary_end = primary_data.index.max()

    print(f"  Primary data: {primary_start} to {primary_end} ({len(primary_data)} r
    print(f"  Fallback data: {fallback_data.index.min()} to {fallback_data.index.ma
    fallback_before = fallback_data[fallback_data.index < primary_start]
    fallback_after = fallback_data[fallback_data.index > primary_end]
    combined = pd.concat([fallback_before, primary_data, fallback_after], axis=0)
    # Remove duplicates, keeping first (primary data)
    combined = combined[~combined.index.duplicated(keep='first')]
    # Sort by index
    combined = combined.sort_index()
    print(f"  Merged data: {combined.index.min()} to {combined.index.max()} ({len(c

    return combined

print("\nTesting data merging for SOL and XRP:")

primary_sol = test_data.get('SOLUSD') if 'test_data' in locals() else None
primary_xrp = test_data.get('XRPUSD') if 'test_data' in locals() else None

if 'SOLUSD' in yf_data:
    print("\nMerging SOL data:")
    merged_sol = merge_data_sources(primary_sol, yf_data['SOLUSD'])
    if merged_sol is not None:
        print(f"  Final SOL: {len(merged_sol)} total records")

if 'XRPUSD' in yf_data:
    print("\nMerging XRP data:")
    merged_xrp = merge_data_sources(primary_xrp, yf_data['XRPUSD'])
    if merged_xrp is not None:
        print(f"  Final XRP: {len(merged_xrp)} total records")
```

```
Testing data merging for SOL and XRP:
============================================================
```

8. Final Data Export and Validation

In [ ]:

```python
# Final data validation and export
def validate_and_export_data(data_dict, export_dir="../data/processed"):
    print("FINAL DATA VALIDATION AND EXPORT")
    print("="*80)
    export_path = Path(export_dir)
    export_path.mkdir(parents=True, exist_ok=True)
    validation_results = []

    for symbol, data in data_dict.items():
        if data is None or data.empty:
            print(f" {symbol}: No data to export")
            continue
        issues = []
        # Check for required columns
        required_cols = ['open', 'high', 'low', 'close', 'volume']
        missing_cols = [col for col in required_cols if col not in data.columns]
        if missing_cols:
            issues.append(f"Missing columns: {missing_cols}")
        # Check for data gaps
        if isinstance(data.index, pd.DatetimeIndex):
            time_diff = data.index.to_series().diff()
            if symbol in config['crypto_symbols']:
                gaps = time_diff[time_diff > pd.Timedelta(hours=2)]
            else:
                gaps = time_diff[time_diff > pd.Timedelta(days=4)]
            if len(gaps) > 0:
                issues.append(f"{len(gaps)} time gaps detected")
        # Check for outliers
        if 'close' in data.columns:
            returns = data['close'].pct_change()
            extreme_returns = returns[abs(returns) > 0.5]  # 50% moves
            if len(extreme_returns) > 0:
                issues.append(f"{len(extreme_returns)} extreme price moves")
        # Check for zero/negative prices
        price_cols = ['open', 'high', 'low', 'close']
        for col in price_cols:
            if col in data.columns:
                invalid_prices = data[data[col] <= 0]
                if len(invalid_prices) > 0:
                    issues.append(f"{len(invalid_prices)} invalid {col} prices")
        # Export the data to parquet for caching
        file_path = export_path / f"{symbol.lower()}_data.parquet"
        data.to_parquet(file_path)
        # Record validation results
        validation_results.append({
            'Symbol': symbol,
            'Records': len(data),
            'Start': data.index.min(),
            'End': data.index.max(),
            'Issues': '; '.join(issues) if issues else 'None',
            'Status': 'Warning' if issues else 'Clean',
            'File': str(file_path)
        })

        status = '[X]' if issues else '[O]'
        print(f"{status} {symbol}: Exported {len(data)} records to {file_path.name}
        if issues:
            for issue in issues:
                print(f"    - {issue}")

    validation_df = pd.DataFrame(validation_results)
    report_path = export_path / "data_validation_report.csv"
    validation_df.to_csv(report_path, index=False)

    # Summary statistics
    print("EXPORT SUMMARY")
    print("="*80)
    print(validation_df[['Symbol', 'Records', 'Status']].to_string(index=False))

    clean_count = len(validation_df[validation_df['Status'] == 'Clean'])
    warning_count = len(validation_df[validation_df['Status'] == 'Warning'])

    print(f"\nClean datasets: {clean_count}")
    print(f"Datasets with warnings: {warning_count}")

    return validation_df

# Validate and export all collected data
if 'test_data' in locals() and test_data:
    validation_report = validate_and_export_data(test_data)

    # Final summary
    print("DATA COLLECTION PIPELINE COMPLETE")
    print("="*80)
    print(f"Successfully collected and exported {len(test_data)} datasets")
    print(f"Data saved to: ../data/processed/")
else:
    print(" No data available to export.")
```

8. Data Export and Summary

In [16]:

```
def create_summary_report(data_dict):
    print("Data Collection Summary Report")
    print("="*80)
    if not data_dict:
        print("No data to summarize")
        return
    summary = []
    for symbol, data in data_dict.items():
        if isinstance(data, pd.DataFrame) and not data.empty:
            # Handle both DatetimeIndex and regular datetime columns
            if isinstance(data.index, pd.DatetimeIndex):
                start_date = data.index.min()
                end_date = data.index.max()
            elif 'timestamp' in data.columns:
                start_date = pd.to_datetime(data['timestamp'].min())
                end_date = pd.to_datetime(data['timestamp'].max())
            else:
                # Try to convert index to datetime
                try:
```

```python
                    start_date = pd.to_datetime(data.index.min())
                    end_date = pd.to_datetime(data.index.max())
                except:
                    start_date = None
                    end_date = None
            summary.append({
                'Symbol': symbol, 'Records': len(data),
                'Start': start_date.strftime('%Y-%m-%d') if start_date else 'N/A',
                'End': end_date.strftime('%Y-%m-%d') if end_date else 'N/A',
                'Days': (end_date - start_date).days if start_date and end_date els
                'Avg Close': f"${data['close'].mean():.2f}" if 'close' in data.colu
                'Volatility': f"{data['close'].pct_change().std() * np.sqrt(252) *
    summary_df = pd.DataFrame(summary)
    print(summary_df.to_string())
    # Save to CSV
    summary_path = Path(config['cache_dir']) / 'data_summary.csv'
    summary_df.to_csv(summary_path, index=False)
    print(f"\nSummary saved to {summary_path}")
    return summary_df
if test_data:
    summary = create_summary_report(test_data)
```

```
Data Collection Summary Report
================================================================================
   Symbol  Records       Start         End  Days  Avg Close Volatility
0  SOLUSD  1354459  2023-01-01  2025-08-01   943    $108.58       2.2%
1  XRPUSD  1311440  2023-01-01  2025-08-01   943      $1.05       2.2%
2  BTCUSD    22656  2023-01-01  2025-08-01   943  $59001.45       8.1%
3  ETHUSD    22656  2023-01-01  2025-08-01   943   $2449.31      10.4%
4  ADAUSD    22656  2023-01-01  2025-08-01   943      $0.51      15.9%
5     SPY      942  2023-01-03  2025-08-01   941    $507.89      13.3%
6     QQQ      942  2023-01-03  2025-08-01   941    $429.45      17.5%
7     IWM      942  2023-01-03  2025-08-01   941    $200.35      18.3%
8     DIA      942  2023-01-03  2025-08-01   941    $385.15      11.8%
9     VTI      942  2023-01-03  2025-08-01   941    $252.17      13.6%

Summary saved to ..\data\ml_comparison_cache\data_summary.csv
```