Currency Trading System

This code implements a complete forex trading system for the EUR/USD currency pair that combines historical backtesting with future price forecasting. Here's a detailed explanation of what it does:

Core Functionality

1. Data Acquisition and Processing

- Fetches historical EUR/USD exchange rate data from Yahoo Finance
- Processes daily closing prices into a structured format for analysis
- Prepares the data for both backtesting and forecasting purposes

2. Trading Strategy Implementation

- Implements a trend-following strategy with retracement entry points
- Identifies market trends by analyzing 30-day price movements
- Enters trades when price retraces against the established trend (buys dips in uptrends, sells rallies in downtrends)
- Uses a fixed 5-day holding period for all trades
- Calculates trade outcomes in pips and dollar profit/loss

3. Backtesting Engine

- Tests the trading strategy on historical data
- Generates entry and exit signals based on the strategy rules
- Calculates performance metrics including win rate, profit/loss, and drawdowns
- Builds an equity curve showing account balance changes over time
- Provides comprehensive statistics on the strategy's historical performance

4. Price Forecasting System

- Creates a predictive model for future EUR/USD prices
- Uses technical indicators (moving averages and volatility) as input features
- Implements a linear regression model trained on recent price data
- Generates day-by-day price forecasts for a user-specified time horizon
- Includes confidence intervals to represent forecast uncertainty

5. Forward-Looking Signal Generation

- Applies the same trading strategy to forecasted prices
- Identifies potential future trading opportunities
- Calculates expected entry/exit points and profit potential
- Provides actionable trade recommendations based on the forecast

Visualization Components

1. Backtest Visualization Suite

- Creates a multi-panel dashboard showing the strategy's historical performance
- Displays price chart with all entry and exit points clearly marked
- Shows the equity curve representing account balance changes over time
- Visualizes drawdowns to highlight periods of capital reduction
- Presents trade outcome statistics with win/loss distribution
- Displays profit distribution to understand trade profitability patterns

2. Trade Sequence Visualization

- Provides detailed views of individual historical trades
- Shows the price action before, during, and after each trade
- Highlights entry and exit points with clear annotations
- Includes trade details such as direction, profit/loss, and outcome

3. Forecast Visualization

- Creates a dedicated chart showing only the forecasted prices
- Includes confidence intervals to represent prediction uncertainty
- Highlights the expected price trend and potential turning points
- Provides a summary of the forecast including expected price change

4. Next Trade Visualization

- Presents a detailed breakdown of the next forecasted trading opportunity
- Shows a zoomed-in view of the expected entry and exit points
- Includes a comprehensive table with all trade details
- Color-codes information based on trade direction and expected outcome

Integration and Workflow

The system ties all these components together into a seamless workflow:

1. **Historical Analysis**: First, it analyzes past data to evaluate the strategy's performance
2. **Performance Assessment**: It generates detailed statistics and visualizations of historical results
3. **Future Projection**: Then, it forecasts future prices based on current market conditions
4. **Opportunity Identification**: It identifies potential trading opportunities in the forecast period
5. **Decision Support**: Finally, it provides actionable trade recommendations with expected outcomes

User Interaction

The system is designed to be user-friendly:

- Users can specify the historical period for backtesting
- They can adjust the forecast horizon to look further into the future
- Position sizing can be customized through the lot size parameter
- All visualizations are automatically generated in a logical sequence
- Results are presented in both visual and numerical formats for easy interpretation

Purpose and Benefits

This trading system serves several key purposes:

1. **Strategy Validation**: It allows traders to validate their strategy on historical data
2. **Performance Analysis**: It provides detailed insights into strategy performance
3. **Forward-Looking Analysis**: It extends beyond historical testing to forecast future opportunities
4. **Decision Support**: It helps traders make informed decisions with clear visualizations
5. **Risk Management**: It provides insights into drawdowns and profit distributions for risk assessment

In essence, this code creates a comprehensive trading system that bridges the gap between historical backtesting and future forecasting, providing traders with both performance validation and actionable trade recommendations for the EUR/USD currency pair.

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from datetime import datetime, timedelta
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import MinMaxScaler
import warnings
import matplotlib.dates as mdates
from matplotlib.gridspec import GridSpec
warnings.filterwarnings('ignore')


# Original functions - kept intact
def fetch_eurusd_data(start="2000-01-01", end=None):
    df = yf.download("EURUSD=X", start=start, end=end, interval="1d")
    df = df[['Close']].dropna().reset_index()
    df.columns = ['Date', 'Close']
    return df


def generate_signals(data, lot_size=0.01):
    pip_value = 0.10 * (lot_size / 0.01)
    signals = []

    for i in range(30, len(data) - 5):  # 5 days for holding period
        last_30 = data['Close'].iloc[i-30:i]
        slope = last_30.iloc[-1] - last_30.iloc[0]  # simple trend slope

        if abs(slope) < 0.001:  # Ignore weak trends
            continue

        trend = 'up' if slope > 0 else 'down'
        entry_price = data['Close'].iloc[i]
        entry_date = data['Date'].iloc[i]
```

```
            # Entry on retracement against the trend
            if trend == 'up' and data['Close'].iloc[i] < data['Close'].iloc[i-1]:
                direction = 'Buy'
            elif trend == 'down' and data['Close'].iloc[i] > data['Close'].iloc[i-1]:
                direction = 'Sell'
            else:
                continue

            # Exit 5 days later
            exit_price = data['Close'].iloc[i+4]
            exit_date = data['Date'].iloc[i+4]

            if direction == 'Buy':
                pips = (exit_price - entry_price) * 10000
            else:
                pips = (entry_price - exit_price) * 10000

            signals.append({
                'Entry Date': entry_date,
                'Direction': direction,
                'Exit Date': exit_date,
                'Entry Price': round(entry_price, 5),
                'Exit Price': round(exit_price, 5),
                'Pips': round(pips, 1),
                'Profit ($)': round(pips * pip_value, 2),
                'Outcome': 'Win' if pips > 0 else 'Loss'
            })

        return pd.DataFrame(signals)


    def calculate_profit(signals_df, lot_size=0.01):
        pip_value = 0.10 * (lot_size / 0.01)  # $0.10 per pip for 0.01 lot
        signals_df['Profit ($)'] = signals_df['Pips'] * pip_value  # Calculate profit from pips
        signals_df['Outcome'] = signals_df['Profit ($)'].apply(lambda x: 'Win' if x > 0 else 'Loss')
        return signals_df


    def calculate_equity_curve(signals_df, initial_balance=100):
        signals_df['Equity'] = signals_df['Profit ($)'].cumsum() + initial_balance
        return signals_df


    # ENHANCED VISUALIZATION FUNCTIONS

    def plot_detailed_backtest(data, signals_df, initial_balance=100):
        """
        Plot comprehensive backtest results including:
        - Price chart with entry/exit points
        - Equity curve
        - Trade outcomes
        - Drawdown analysis
        """
        if signals_df.empty:
            print("No signals generated for backtest visualization.")
            return

        # Calculate equity curve if not already done
        if 'Equity' not in signals_df.columns:
            signals_df = calculate_equity_curve(signals_df, initial_balance)

        # Calculate drawdown
        signals_df['Peak'] = signals_df['Equity'].cummax()
        signals_df['Drawdown'] = (signals_df['Equity'] - signals_df['Peak']) / signals_df['Peak'] * 100

        # Create figure with subplots
        fig = plt.figure(figsize=(16, 14))
        gs = GridSpec(4, 2, figure=fig)

        # 1. Price chart with entry/exit points
        ax1 = fig.add_subplot(gs[0:2, 0:2])
        ax1.plot(data['Date'], data['Close'], label="EUR/USD Price", color='blue', alpha=0.6)

        # Plot Buy signals (Entry points)
        buy_signals = signals_df[signals_df['Direction'] == 'Buy']
        if not buy_signals.empty:
            ax1.scatter(buy_signals['Entry Date'], buy_signals['Entry Price'],
                        marker='^', color='green', label='Buy Signal', s=80, zorder=5)
            ax1.scatter(buy_signals['Exit Date'], buy_signals['Exit Price'],
                        marker='x', color='green', label='Buy Exit', s=80, zorder=5)

        # Plot Sell signals (Entry points)
```

```
        sell_signals = signals_df[signals_df['Direction'] == 'Sell']
        if not sell_signals.empty:
            ax1.scatter(sell_signals['Entry Date'], sell_signals['Entry Price'],
                        marker='v', color='red', label='Sell Signal', s=80, zorder=5)
            ax1.scatter(sell_signals['Exit Date'], sell_signals['Exit Price'],
                        marker='x', color='red', label='Sell Exit', s=80, zorder=5)

        # Connect entry and exit points with lines
        for _, row in signals_df.iterrows():
            color = 'green' if row['Outcome'] == 'Win' else 'red'
            ax1.plot([row['Entry Date'], row['Exit Date']],
                     [row['Entry Price'], row['Exit Price']],
                     color=color, linestyle='--', alpha=0.7)

        ax1.set_title('EUR/USD Price with Trading Signals', fontsize=14)
        ax1.set_xlabel('Date')
        ax1.set_ylabel('EUR/USD Price')
        ax1.legend()
        ax1.grid(True)

        # 2. Equity curve
        ax2 = fig.add_subplot(gs[2, 0])
        ax2.plot(signals_df['Entry Date'], signals_df['Equity'],
                 label="Equity Curve", color='blue', linewidth=2)
        ax2.set_title('Equity Curve', fontsize=14)
        ax2.set_xlabel('Date')
        ax2.set_ylabel('Equity ($)')
        ax2.grid(True)

        # 3. Drawdown chart
        ax3 = fig.add_subplot(gs[2, 1])
        ax3.fill_between(signals_df['Entry Date'], signals_df['Drawdown'], 0,
                         color='red', alpha=0.3)
        ax3.set_title('Drawdown (%)', fontsize=14)
        ax3.set_xlabel('Date')
        ax3.set_ylabel('Drawdown (%)')
        ax3.grid(True)

        # 4. Trade outcomes (Win/Loss)
        ax4 = fig.add_subplot(gs[3, 0])
        outcomes = signals_df['Outcome'].value_counts()
        ax4.bar(['Win', 'Loss'],
                [outcomes.get('Win', 0), outcomes.get('Loss', 0)],
                color=['green', 'red'])
        ax4.set_title('Trade Outcomes', fontsize=14)
        ax4.set_ylabel('Number of Trades')
        for i, v in enumerate([outcomes.get('Win', 0), outcomes.get('Loss', 0)]):
            ax4.text(i, v + 0.5, str(v), ha='center')

        # 5. Profit distribution
        ax5 = fig.add_subplot(gs[3, 1])
        ax5.hist(signals_df['Profit ($)'], bins=20, color='blue', alpha=0.7)
        ax5.axvline(x=0, color='black', linestyle='--')
        ax5.set_title('Profit Distribution', fontsize=14)
        ax5.set_xlabel('Profit ($)')
        ax5.set_ylabel('Frequency')
        ax5.grid(True)

        plt.tight_layout()
        plt.show()

        # Print summary statistics
        print("\n=== Backtest Summary ===")
        print(f"Total Trades: {len(signals_df)}")
        print(f"Win Rate: {outcomes.get('Win', 0) / len(signals_df) * 100:.2f}%")
        print(f"Initial Balance: ${initial_balance:.2f}")
        print(f"Final Balance: ${signals_df['Equity'].iloc[-1]:.2f}")
        print(f"Total Profit: ${signals_df['Equity'].iloc[-1] - initial_balance:.2f}")
        print(f"Max Drawdown: {signals_df['Drawdown'].min():.2f}%")

        return signals_df


    def plot_trade_sequence(signals_df, data, num_trades=10):
        """
        Plot a sequence of trades showing each entry and exit in detail
        """
        if signals_df.empty or len(signals_df) == 0:
            print("No signals available to plot trade sequence.")
            return

        # Limit to the specified number of trades
        trades_to_plot = min(num_trades, len(signals_df))
```

```
        signals_subset = signals_df.head(trades_to_plot)

        # Create a figure with subplots for each trade
        fig, axes = plt.subplots(trades_to_plot, 1, figsize=(12, 4 * trades_to_plot))

        # If only one trade, make axes iterable
        if trades_to_plot == 1:
            axes = [axes]

        for i, (_, trade) in enumerate(signals_subset.iterrows()):
            # Get data for the trade period (plus some context)
            start_idx = data[data['Date'] >= trade['Entry Date'] - timedelta(days=15)].index[0]
            try:
                end_idx = data[data['Date'] >= trade['Exit Date'] + timedelta(days=5)].index[0]
            except IndexError:
                end_idx = len(data) - 1

            trade_data = data.iloc[start_idx:end_idx+1]

            # Plot the price action
            axes[i].plot(trade_data['Date'], trade_data['Close'], color='blue')

            # Highlight entry and exit
            color = 'green' if trade['Outcome'] == 'Win' else 'red'
            marker = '^' if trade['Direction'] == 'Buy' else 'v'

            axes[i].scatter(trade['Entry Date'], trade['Entry Price'],
                            color=color, marker=marker, s=100, zorder=5)
            axes[i].scatter(trade['Exit Date'], trade['Exit Price'],
                            color=color, marker='x', s=100, zorder=5)

            # Connect entry and exit with a line
            axes[i].plot([trade['Entry Date'], trade['Exit Date']],
                         [trade['Entry Price'], trade['Exit Price']],
                         color=color, linestyle='--')

            # Add annotations
            axes[i].annotate(f"{trade['Direction']} Entry",
                             (trade['Entry Date'], trade['Entry Price']),
                             xytext=(10, 10), textcoords='offset points')
            axes[i].annotate(f"Exit: {trade['Pips']} pips",
                             (trade['Exit Date'], trade['Exit Price']),
                             xytext=(10, -15), textcoords='offset points')

            # Set title with trade details
            axes[i].set_title(f"Trade {i+1}: {trade['Direction']} {trade['Entry Date'].strftime('%Y-%m-%d')} to {trade['Exit Date'].strftime
            axes[i].grid(True)

            # Format x-axis dates
            axes[i].xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
            plt.setp(axes[i].xaxis.get_majorticklabels(), rotation=45)

        plt.tight_layout()
        plt.show()


    def plot_forecast_only(forecast_data, confidence_interval=0.01):
        """
        Plot only the forecasted prices with confidence intervals
        """
        if forecast_data.empty:
            print("No forecast data available to plot.")
            return

        plt.figure(figsize=(12, 6))

        # Calculate confidence intervals (simple approach)
        last_price = forecast_data['Close'].iloc[0]
        forecast_data['Upper'] = forecast_data['Close'] * (1 + confidence_interval)
        forecast_data['Lower'] = forecast_data['Close'] * (1 - confidence_interval)

        # Plot the forecast with confidence interval
        plt.plot(forecast_data['Date'], forecast_data['Close'],
                 label="Forecasted EUR/USD", color='blue', linewidth=2)
        plt.fill_between(forecast_data['Date'],
                         forecast_data['Lower'],
                         forecast_data['Upper'],
                         color='blue', alpha=0.2, label=f"{confidence_interval*100}% Confidence Interval")

        # Add horizontal line at the starting price
        plt.axhline(y=last_price, color='black', linestyle='--',
                    alpha=0.5, label=f"Current Price ({last_price:.5f})")
```

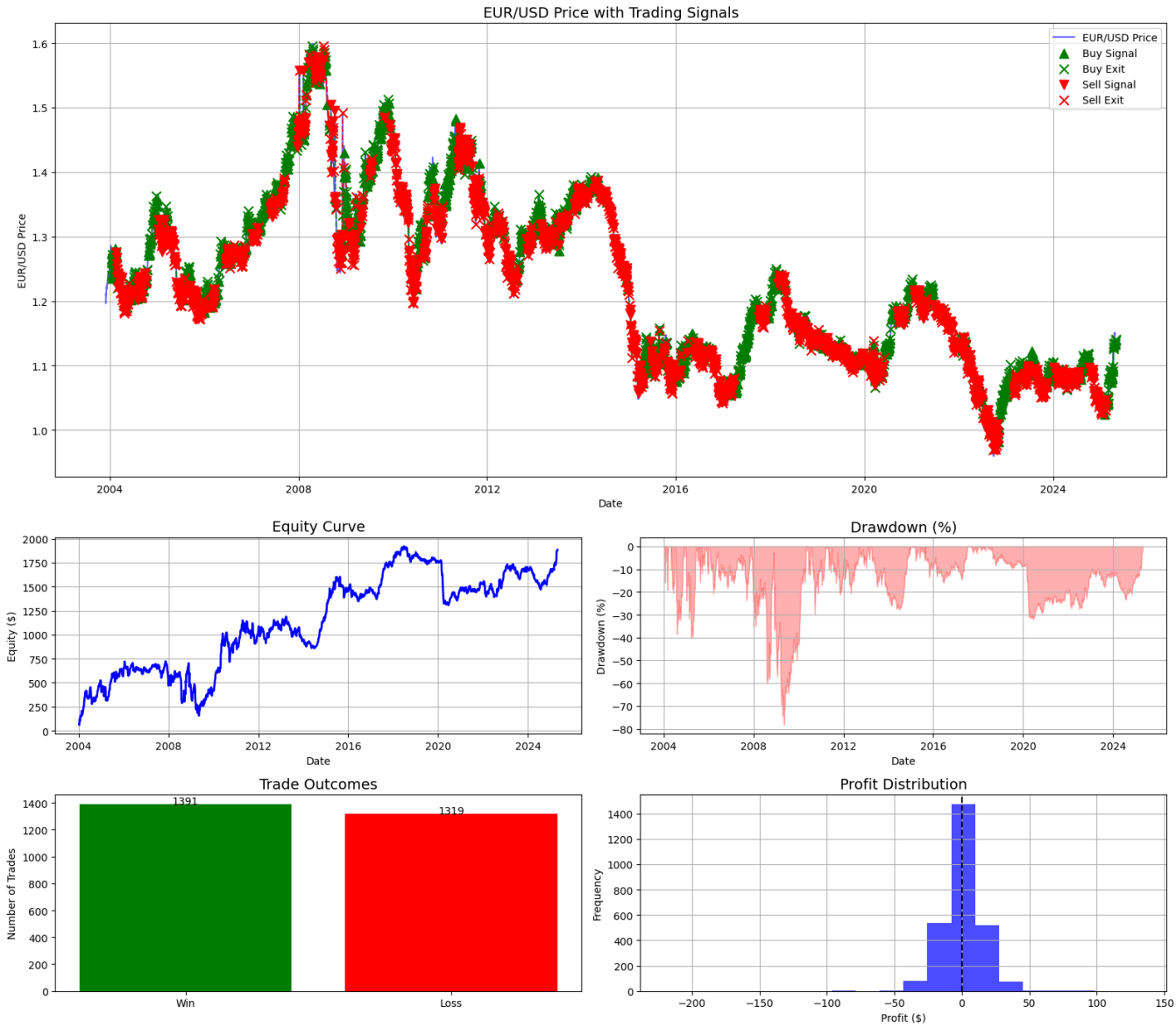
```
        # Formatting
        plt.title('EUR/USD Price Forecast', fontsize=16)
        plt.xlabel('Date', fontsize=12)
        plt.ylabel('EUR/USD Price', fontsize=12)
        plt.grid(True)
        plt.legend()

        # Format x-axis dates
        plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
        plt.xticks(rotation=45)

        plt.tight_layout()
        plt.show()

        # Print forecast summary
        print("\n=== Forecast Summary ===")
        print(f"Forecast Start: {forecast_data['Date'].iloc[0].strftime('%Y-%m-%d')}")
        print(f"Forecast End: {forecast_data['Date'].iloc[-1].strftime('%Y-%m-%d')}")
        print(f"Starting Price: {forecast_data['Close'].iloc[0]:.5f}")
        print(f"Ending Price: {forecast_data['Close'].iloc[-1]:.5f}")
        print(f"Price Change: {(forecast_data['Close'].iloc[-1] - forecast_data['Close'].iloc[0]):.5f} ({((forecast_data['Close'].iloc[-1]/fo


    def plot_next_entry_details(forecast_signals, forecast_data):
        """
        Plot detailed information about the next forecasted entry
        """
        if forecast_signals.empty:
            print("No forecasted signals available.")
            return None

        # Get the next entry (first signal in the forecast)
        next_entry = forecast_signals.iloc[0]

        # Create a DataFrame with details
        next_entry_df = pd.DataFrame({
            'Attribute': [
                'Entry Date', 'Direction', 'Entry Price',
                'Exit Date', 'Exit Price', 'Expected Pips',
                'Expected Profit ($)', 'Trend'
            ],
            'Value': [
                next_entry['Entry Date'].strftime('%Y-%m-%d'),
                next_entry['Direction'],
                f"{next_entry['Entry Price']:.5f}",
                next_entry['Exit Date'].strftime('%Y-%m-%d'),
                f"{next_entry['Exit Price']:.5f}",
                f"{next_entry['Pips']}",
                f"${next_entry['Profit ($)']}",
                'Uptrend' if next_entry['Direction'] == 'Buy' else 'Downtrend'
            ]
        })

        # Create figure with two subplots
        fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6),
                                       gridspec_kw={'width_ratios': [2, 1]})

        # Plot price chart with entry and exit points
        entry_idx = forecast_data[forecast_data['Date'] >= next_entry['Entry Date']].index[0]
        try:
            exit_idx = forecast_data[forecast_data['Date'] >= next_entry['Exit Date']].index[0]
        except IndexError:
            exit_idx = len(forecast_data) - 1

        # Get some data before entry for context
        start_idx = max(0, entry_idx - 5)
        relevant_data = forecast_data.iloc[start_idx:exit_idx+5]

        ax1.plot(relevant_data['Date'], relevant_data['Close'], color='blue')

        # Highlight entry and exit
        color = 'green' if next_entry['Direction'] == 'Buy' else 'red'
        marker = '^' if next_entry['Direction'] == 'Buy' else 'v'

        ax1.scatter(next_entry['Entry Date'], next_entry['Entry Price'],
                    color=color, marker=marker, s=150, zorder=5)
        ax1.scatter(next_entry['Exit Date'], next_entry['Exit Price'],
                    color=color, marker='x', s=150, zorder=5)

        # Connect entry and exit with a line
        ax1.plot([next_entry['Entry Date'], next_entry['Exit Date']],
                 [next_entry['Entry Price'], next_entry['Exit Price']],
                 color=color, linestyle='--')
```

```
        # Add annotations
        ax1.annotate(f"{next_entry['Direction']} Entry",
                     (next_entry['Entry Date'], next_entry['Entry Price']),
                     xytext=(10, 10), textcoords='offset points')
        ax1.annotate(f"Exit: {next_entry['Pips']} pips",
                     (next_entry['Exit Date'], next_entry['Exit Price']),
                     xytext=(10, -15), textcoords='offset points')

        ax1.set_title(f"Next Forecasted Trade: {next_entry['Direction']} on {next_entry['Entry Date'].strftime('%Y-%m-%d')}")
        ax1.grid(True)
        ax1.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
        plt.setp(ax1.xaxis.get_majorticklabels(), rotation=45)

        # Create a table with trade details
        ax2.axis('off')
        table = ax2.table(
            cellText=next_entry_df.values,
            colLabels=['Attribute', 'Value'],
            loc='center',
            cellLoc='center'
        )
        table.auto_set_font_size(False)
        table.set_fontsize(10)
        table.scale(1.2, 1.5)

        # Color the direction row based on buy/sell
        direction_color = 'green' if next_entry['Direction'] == 'Buy' else 'red'
        table[(1, 1)].set_facecolor(direction_color)
        table[(1, 1)].set_text_props(color='white')

        # Color the profit row based on positive/negative
        profit_color = 'green' if next_entry['Profit ($)'] > 0 else 'red'
        table[(6, 1)].set_facecolor(profit_color)
        table[(6, 1)].set_text_props(color='white')

        plt.tight_layout()
        plt.show()

        return next_entry_df


    # FORECASTING FUNCTIONS

    def forecast_prices(data, forecast_days=30):
        """
        Generate price forecasts for the specified number of days
        """
        # Create features for forecasting
        df = data.copy()
        df['MA5'] = df['Close'].rolling(window=5).mean()
        df['MA20'] = df['Close'].rolling(window=20).mean()
        df['MA50'] = df['Close'].rolling(window=50).mean()
        df['Return'] = df['Close'].pct_change()
        df['Volatility'] = df['Return'].rolling(window=20).std()
        df = df.dropna()

        # Prepare features for prediction
        features = ['MA5', 'MA20', 'MA50', 'Volatility']
        X = df[features].values[-60:]  # Use last 60 days for training
        y = df['Close'].values[-60:]

        # Train a simple linear regression model
        model = LinearRegression()
        model.fit(X, y)

        # Generate future dates
        last_date = df['Date'].iloc[-1]
        future_dates = [last_date + timedelta(days=i+1) for i in range(forecast_days)]

        # Initialize forecasted values
        forecasted_prices = []
        last_known_values = df[features].iloc[-1].values.reshape(1, -1)

        # Generate forecasts recursively
        for i in range(forecast_days):
            # Predict next price
            next_price = model.predict(last_known_values)[0]
            forecasted_prices.append(next_price)

            # Update features for next prediction
            # This is a simplified approach - in reality, you'd need more sophisticated feature updates
            ma5 = (last_known_values[0, 0] * 4 + next_price) / 5
```

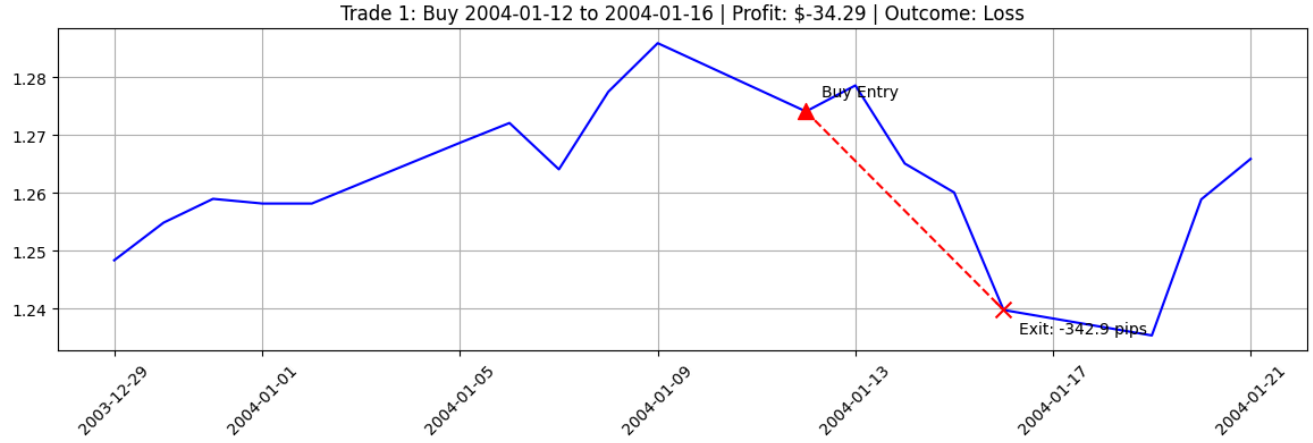
```
            ma20 = (last_known_values[0, 1] * 19 + next_price) / 20
            ma50 = (last_known_values[0, 2] * 49 + next_price) / 50
            volatility = last_known_values[0, 3]  # Keep volatility constant for simplicity

            last_known_values = np.array([[ma5, ma20, ma50, volatility]])

        # Create forecast DataFrame
        forecast_df = pd.DataFrame({
            'Date': future_dates,
            'Close': forecasted_prices
        })

        return forecast_df


    def generate_forecast_signals(historical_data, forecast_data, lot_size=0.01):
        """
        Generate trading signals based on forecasted prices
        """
        # Combine historical and forecast data for analysis
        combined_data = pd.concat([historical_data.tail(30), forecast_data])
        combined_data = combined_data.reset_index(drop=True)

        # Apply the original signal generation logic to the combined data
        signals = []
        pip_value = 0.10 * (lot_size / 0.01)

        for i in range(30, len(combined_data) - 5):
            # Skip if we're not in the forecast period
            if combined_data['Date'].iloc[i] <= historical_data['Date'].iloc[-1]:
                continue

            last_30 = combined_data['Close'].iloc[i-30:i]
            slope = last_30.iloc[-1] - last_30.iloc[0]

            if abs(slope) < 0.001:
                continue

            trend = 'up' if slope > 0 else 'down'
            entry_price = combined_data['Close'].iloc[i]
            entry_date = combined_data['Date'].iloc[i]

            if trend == 'up' and combined_data['Close'].iloc[i] < combined_data['Close'].iloc[i-1]:
                direction = 'Buy'
            elif trend == 'down' and combined_data['Close'].iloc[i] > combined_data['Close'].iloc[i-1]:
                direction = 'Sell'
            else:
                continue

            # Exit 5 days later (if available in forecast)
            if i + 4 < len(combined_data):
                exit_price = combined_data['Close'].iloc[i+4]
                exit_date = combined_data['Date'].iloc[i+4]
            else:
                # If forecast doesn't extend that far, use the last available price
                exit_price = combined_data['Close'].iloc[-1]
                exit_date = combined_data['Date'].iloc[-1]

            if direction == 'Buy':
                pips = (exit_price - entry_price) * 10000
            else:
                pips = (entry_price - exit_price) * 10000

            signals.append({
                'Entry Date': entry_date,
                'Direction': direction,
                'Exit Date': exit_date,
                'Entry Price': round(entry_price, 5),
                'Exit Price': round(exit_price, 5),
                'Pips': round(pips, 1),
                'Profit ($)': round(pips * pip_value, 2),
                'Outcome': 'Win' if pips > 0 else 'Loss',
                'Forecast': True
            })

        return pd.DataFrame(signals) if signals else pd.DataFrame()


    # MAIN FUNCTION FOR COMPREHENSIVE ANALYSIS

    def run_comprehensive_analysis(start_date="2000-01-01", forecast_days=30, lot_size=0.01):
        """
        Run a comprehensive analysis with all visualizations:
```

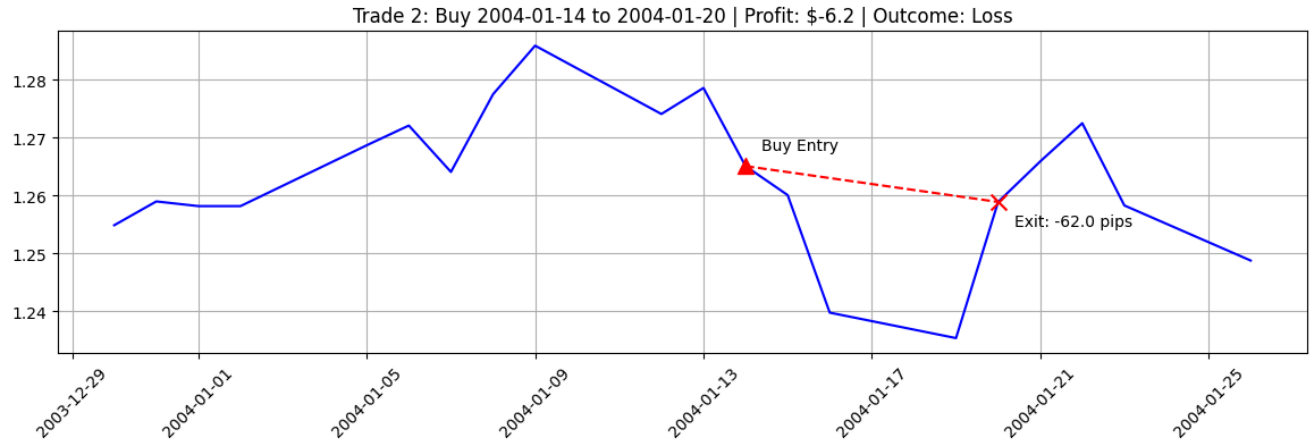
```
        1. Backtest with detailed visualizations
        2. Trade sequence visualization
        3. Forecast visualization
        4. Next entry details
        """
        print("\n=== Fetching EUR/USD Data ===")
        historical_data = fetch_eurusd_data(start=start_date)
        print(f"Data loaded: {len(historical_data)} days from {historical_data['Date'].min()} to {historical_data['Date'].max()}")

        print("\n=== Running Backtest ===")
        signals_df = generate_signals(historical_data, lot_size)
        signals_df = calculate_profit(signals_df, lot_size)
        signals_df = calculate_equity_curve(signals_df)

        print(f"Generated {len(signals_df)} trading signals")

        print("\n=== Visualizing Backtest Results ===")
        plot_detailed_backtest(historical_data, signals_df)

        print("\n=== Visualizing Trade Sequence ===")
        plot_trade_sequence(signals_df, historical_data, num_trades=5)

        print("\n=== Generating Price Forecast ===")
        forecast_data = forecast_prices(historical_data, forecast_days)
        print(f"Generated {len(forecast_data)} days of price forecasts")

        print("\n=== Visualizing Forecast Only ===")
        plot_forecast_only(forecast_data)

        print("\n=== Generating Forecast Signals ===")
        forecast_signals = generate_forecast_signals(historical_data, forecast_data, lot_size)
        print(f"Generated {len(forecast_signals)} forecast signals")

        if not forecast_signals.empty:
            print("\n=== Visualizing Next Entry Details ===")
            next_entry_df = plot_next_entry_details(forecast_signals, forecast_data)
        else:
            print("\nNo forecast signals generated for the specified period.")

        return {
            'historical_data': historical_data,
            'backtest_signals': signals_df,
            'forecast_data': forecast_data,
            'forecast_signals': forecast_signals
        }


    # Example usage
    if __name__ == "__main__":
        results = run_comprehensive_analysis(start_date="2000-01-01", forecast_days=30, lot_size=0.01)
```

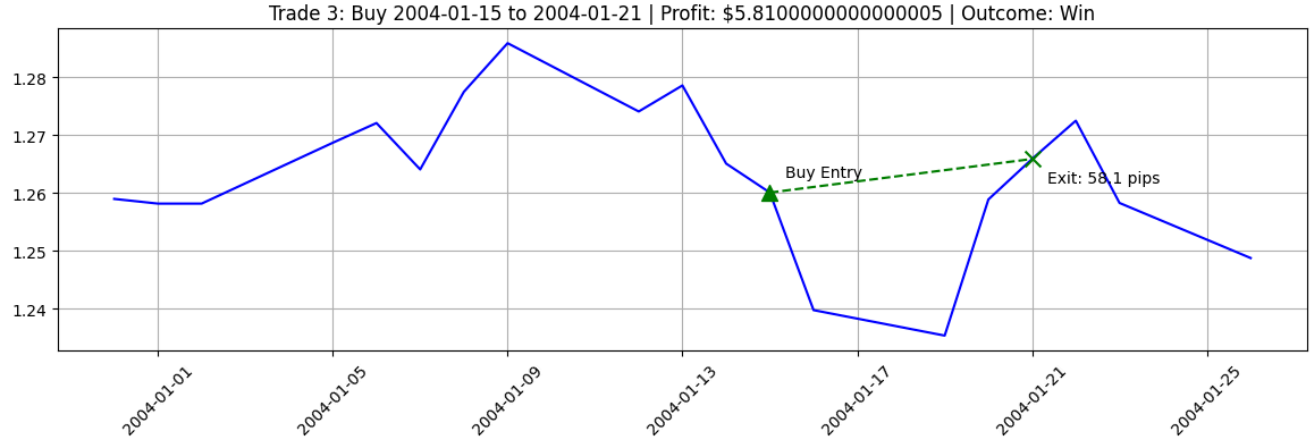
```
=== Fetching EUR/USD Data ===
YF.download() has changed argument auto_adjust default to True
[**********************100%**********************]  1 of 1 completed
Data loaded: 5558 days from 2003-12-01 00:00:00 to 2025-05-02 00:00:00

=== Running Backtest ===
Generated 2710 trading signals

=== Visualizing Backtest Results ===
```

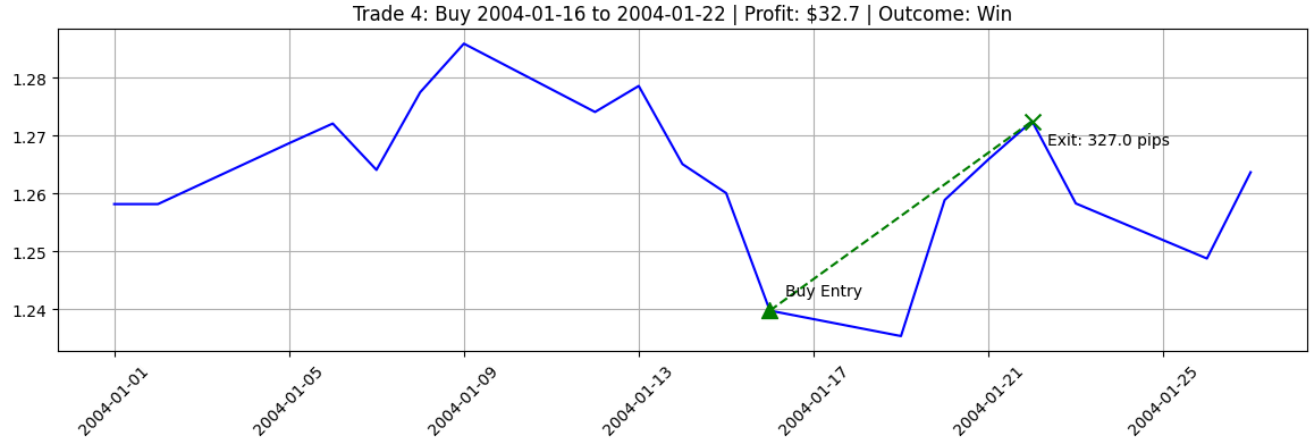


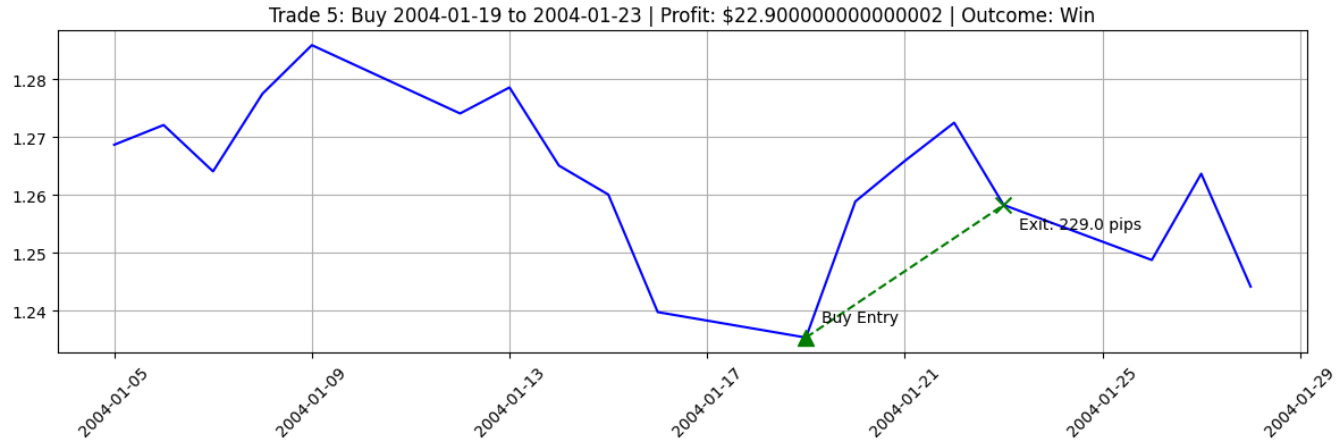

```
=== Backtest Summary ===
Total Trades: 2710
Win Rate: 51.33%
Initial Balance: $100.00
Final Balance: $1883.96
Total Profit: $1783.96
Max Drawdown: -78.33%

=== Visualizing Trade Sequence ===
```












```
=== Generating Price Forecast ===
Generated 30 days of price forecasts

=== Visualizing Forecast Only ===
```