User Manual: Unified Data Analysis and Forecasting Classes

July 22, 2025

This document serves as a user manual for the `unified_data_analysis_forecasting.py` script, which consolidates various Python classes for time series analysis, descriptive statistics, and forecasting.

Contents

1 Introduction **2**

2 Prerequisites **2**

3 Installation **3**

4 Code Structure Overview **3**
- 4.1 Plotting Classes 3
- 4.2 Descriptive Statistics & PCA Classes 4
- 4.3 Forecasting Models 4

5 How to Use the Code **5**
- 5.1 Running All Examples 6
- 5.2 Using Individual Classes 6
  - 5.2.1 Plotting Examples 6
  - 5.2.2 Descriptive Statistics & PCA Examples 7
  - 5.2.3 Forecasting Models Examples 8

6 Troubleshooting and Tips **11**

1 Introduction

The `unified_data_analysis_forecasting.py` script provides a powerful toolkit for comprehensive data analysis, with a particular focus on time series. It integrates functionalities for:

- **Data Visualization:** Plotting single/multiple time series, adding recession shading, and analyzing rolling statistics/correlations, along with histograms and density plots.
- **Descriptive Statistics & Dimensionality Reduction:** Generating summary statistics, correlation matrices, performing Principal Component Analysis (PCA), and Multivariate Analysis of Variance (MANOVA). It also includes time series bootstrapping for distributional fit testing.
- **Advanced Forecasting Models:** Implementing various forecasting techniques, including:
  - Minimum Description Length (MDL) based Autoregressive (AR) models.
  - Simple AR(1) models.
  - Unified AR and Support Vector Regression (SVR) forecasters.
  - Enhanced rolling window SVR forecasters with hyperparameter tuning.
  - Adaptive Learning Forecaster for continuous and binary data.
  - Autoregressive Distributed Lag with eXogenous variables (ARDLX) models (OLS and Elastic Net).
  - Echo State Networks (ESN).
  - Backtesting and real-time forecasting utilities for model evaluation.

This unified script aims to provide a convenient and well-structured resource for researchers and analysts working with time series data.

2 Prerequisites

To run the `unified_data_analysis_forecasting.py` script, you will need:

- **Python:** Version 3.7 or higher is recommended.
- **Required Libraries:** The script relies on several popular Python libraries for numerical operations, data manipulation, plotting, and statistical modeling.

3 Installation

Before running the code, you need to install the necessary Python libraries. You can install them using `pip`, Python's package installer.

Open your terminal or command prompt and run the following command:

```
        pip install numpy pandas matplotlib seaborn scikit-learn
           tabulate statsmodels
```

Listing 1: Installation Command

- `numpy`: Fundamental package for numerical computing.
- `pandas`: For data manipulation and analysis, especially with DataFrames.
- `matplotlib`: For creating static, interactive, and animated visualizations.
- `seaborn`: A data visualization library based on matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.
- `scikit-learn`: For machine learning algorithms, including SVR, PCA, and various metrics.
- `tabulate`: For printing data in a nice, tabular format.
- `statsmodels`: For statistical modeling, including AR models and MANOVA.

4 Code Structure Overview

The script is organized into three main sections, each containing related classes:

4.1 Plotting Classes

- `TimeSeriesPlotter`:
  - Handles various time series visualizations.
  - Methods: `plot_single_series`, `plot_multiple_series`, `customize_plot`, `plot_with_shading`, `plot_rolling_statistics`, `plot_rolling_correlation`.
- `TimeSeriesWithHistogram`:
  - Plots a time series alongside its histogram and empirical density function.
  - Method: `plot`.

4.2 Descriptive Statistics & PCA Classes

- `DataAnalyzer`:
  - Performs descriptive statistics, correlation analysis, PCA, and MANOVA.
  - Methods: `descriptive_statistics`, `correlation_matrix`, `principal_component_analysis`, `manova`.
- `TimeSeriesBootstrap`:
  - Generates bootstrap samples for time series data.
  - Tests distributional fit against common distributions (Normal, Exponential, Gamma, Beta).
  - Methods: `test_distribution_fit`, `print_results_table`.

4.3 Forecasting Models

- `MDL_AR_Model`:
  - Implements an Autoregressive (AR) model with lag order selection based on the Minimum Description Length (MDL) principle.
  - Methods: `fit`, `forecast`.
- `AR1_Model`:
  - A simple Autoregressive model of order 1.
  - Methods: `fit`, `forecast`.
- `Backtesting`:
  - Performs backtesting for `MDL_AR_Model` and `AR1_Model` using expanding or rolling windows.
  - Methods: `run_backtest`, `evaluate`, `plot_forecasts_and_errors`.
- `RealTimeForecasting`:
  - Computes forecasts using `MDL_AR_Model` and `AR1_Model` on different sample sizes (full, last 25%, 50%, 75%).
  - Method: `compute_forecasts`.
- `UnifiedForecaster`:
  - A versatile forecaster supporting both AR models (with lag selection) and Support Vector Regression (SVR).
  - Methods: `evaluate`, `get_predictions`, `plot`.
- `EnhancedRollingSVRForecaster`:

  - An advanced rolling window SVR forecaster with detailed data preparation, validation, hyperparameter tuning, and comprehensive evaluation.
  - Methods: `prepare_data`, `fit`, `rolling_predict`, `evaluate_rolling`, `visualize`, `tune_hyperparameters`.

- `AdaptiveLearningForecaster`:
  - Implements an adaptive learning forecasting approach based on ensemble weighting, supporting continuous and binary data.
  - Methods: `forecast`, `update`, `get_weights`.
  - Includes helper functions: `moving_average_model`, `momentum_model`, `majority_vote_model`.
- `ARDLX`:
  - Autoregressive Distributed Lag with eXogenous variables model, supporting OLS and Elastic Net.
  - Methods: `fit`, `predict`.
- `ESN`:
  - Echo State Network for time series forecasting.
  - Methods: `fit`, `predict`.
- `Forecast`:
  - A utility class to encapsulate `ARDLX` and `ESN` models, handling feature construction and model fitting/prediction.
  - Methods: `construct_features`, `fit_models`, `predict`.
- `EvaluationMetrics`:
  - A static class for computing common regression evaluation metrics (MAE, MSE, RMSE).
  - Static Method: `compute_metrics`.

5 How to Use the Code

You can use the provided script in two main ways: by running all the included examples or by importing and using individual classes for your specific needs.

5.1 Running All Examples

The script includes a comprehensive `run_all_examples()` function that demonstrates the usage of almost every class. To run all the examples and see the various plots and printed outputs:

1. **Save the code:** Save the provided Python code as a `.py` file (e.g., `unified_data_analysis_forecast`
2. **Open a terminal/command prompt:** Navigate to the directory where you saved the file.
3. **Execute the script:** Run the script using the Python interpreter:

   ```
   python unified_data_analysis_forecasting.py
   ```

   This will execute the `run_all_examples()` function, generating plots and printing results to your console. Plots will appear in separate windows.

5.2 Using Individual Classes

You can also import and use any of the classes independently in your own Python scripts or Jupyter notebooks.

5.2.1 Plotting Examples

To use the plotting classes:

```
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from unified_data_analysis_forecasting import TimeSeriesPlotter, TimeSeriesWithHistogram # Assuming the file is named unified_data_analysis_forecasting.py

        # Sample Data for TimeSeriesPlotter
        time_data = pd.to_datetime(pd.date_range(start='2020-01-01', periods=100, freq='M'))
        series_data1 = np.random.rand(100) * 100 + np.sin(np.arange(100)/10) * 20
        series_data2 = np.random.rand(100) * 50 + 50 + np.cos(np.arange(100)/5) * 15
        df_plot = pd.DataFrame({'time': time_data, 'series1': series_data1, 'series2': series_data2})

        # Initialize TimeSeriesPlotter
        plotter = TimeSeriesPlotter(df_plot, title="My Custom Time Series Plot", xlabel="Date", ylabel="Value")

        # Plot a single series with a secondary axis
        plotter.plot_single_series('series1', secondary_series='series2', secondary_ylabel='Secondary Value')
```

```

        # Plot rolling statistics
        plotter.plot_rolling_statistics('series1', window_size=6)

        # Sample Data for TimeSeriesWithHistogram
        df_hist = pd.DataFrame({
            'Value': np.random.normal(loc=100, scale=15, size=300),
            'Date': pd.date_range(start='2024-01-01', periods=300)
        }).set_index('Date')

        # Initialize TimeSeriesWithHistogram
        hist_plotter = TimeSeriesWithHistogram(df_hist)

        # Plot time series with histogram and density
        hist_plotter.plot(series_name='Value', bins=30, kde=True,
        title='My Value Distribution',
        xlabel='Time', ylabel='Observation Count')
```

Listing 2: Plotting Class Usage Example

5.2.2 Descriptive Statistics & PCA Examples

To use the descriptive statistics and PCA classes:

```
        import pandas as pd
        import numpy as np
        from unified_data_analysis_forecasting import DataAnalyzer,
           TimeSeriesBootstrap

        # Sample Data for DataAnalyzer
        data_analyzer = {
            'Feature_A': np.random.randn(50),
            'Feature_B': np.random.randn(50) * 3 + 10,
            'Feature_C': np.random.rand(50) * 5,
            'Group_Var': np.random.choice(['Group1', 'Group2'], size
               =50)
        }
        df_analyzer = pd.DataFrame(data_analyzer)

        # Initialize DataAnalyzer
        analyzer = DataAnalyzer(df_analyzer)

        # Generate descriptive statistics
        desc_stats = analyzer.descriptive_statistics(save_as='csv')

        # Generate correlation matrix
        corr_matrix = analyzer.correlation_matrix(save_as='png')

        # Perform PCA
        pca_results = analyzer.principal_component_analysis(
           n_components=2, save_as='pdf')

        # Perform MANOVA (ensure 'Group_Var' is suitable for MANOVA)
        try:
        manova_results = analyzer.manova(dependent_vars=['Feature_A'
           , 'Feature_B'], independent_var='Group_Var', save_as='txt
           ')
        except ValueError as e:
        print(f"MANOVA usage example error: {e}")

        # Sample Data for TimeSeriesBootstrap
        bootstrap_data = np.random.lognormal(mean=0.5, sigma=0.8,
           size=150) # Example: log-normal data
        ts_bootstrap = TimeSeriesBootstrap(bootstrap_data,
           num_samples=100, block_size=15)

        # Print distributional fit test results
        ts_bootstrap.print_results_table()
```

Listing 3: Descriptive Statistics & PCA Usage Example

5.2.3 Forecasting Models Examples

To use the forecasting models:

```
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from unified_data_analysis_forecasting import (
        MDL_AR_Model, AR1_Model, Backtesting, RealTimeForecasting,
        UnifiedForecaster, EnhancedRollingSVRForecaster,
        AdaptiveLearningForecaster, moving_average_model,
           momentum_model, majority_vote_model,
        ARDLX, ESN, Forecast, RollingWindowBacktest,
           EvaluationMetrics
        )

        # General Sample Data for Forecasting
        np.random.seed(123)
        forecast_data = np.cumsum(np.random.normal(0, 0.7, 200)) +
           10 * np.sin(np.linspace(0, 30, 200))
        forecast_data_series = pd.Series(forecast_data)

        print("\n--- MDL_AR_Model and AR1_Model Backtesting ---")
        # Backtesting with a rolling window
        backtesting_rolling = Backtesting(forecast_data, sample_type
           ='rolling', window_size=40)
        backtesting_rolling.run_backtest(forecast_steps=1)
        backtesting_rolling.plot_forecasts_and_errors()

        print("\n--- Real-Time Forecasting ---")
        real_time_forecasting = RealTimeForecasting(forecast_data)
```

```python
    real_time_forecasting.compute_forecasts(forecast_steps=1)

    print("\n--- UnifiedForecaster (AR and SVR) ---")
    # Using AR models
    uf_ar = UnifiedForecaster(forecast_data, R=25, lag_criterion='AIC', verbose=False)
    uf_ar.plot()
    print("UnifiedForecaster AR Metrics:\n", uf_ar.evaluate())

    # Using SVR
    svr_params = {'kernel': 'rbf', 'C': 5, 'epsilon': 0.05, 'gamma': 'scale'}
    uf_svr = UnifiedForecaster(forecast_data, R=25, use_svr=True, svr_params=svr_params, verbose=False)
    uf_svr.plot()
    print("UnifiedForecaster SVR Metrics:\n", uf_svr.evaluate())

    print("\n--- EnhancedRollingSVRForecaster ---")
    # Data for Enhanced SVR
    data_enhanced_svr = np.sin(np.linspace(0, 10*np.pi, 300)) + np.random.normal(0, 0.1, 300)

    forecaster_enhanced_svr = EnhancedRollingSVRForecaster(
    window_size=20, kernel='rbf', C=10, verbose=True
    )

    try:
    X_train, X_test, y_train, y_test = forecaster_enhanced_svr.prepare_data(data_enhanced_svr, train_size=0.7)
    best_params = forecaster_enhanced_svr.tune_hyperparameters()
    forecaster_enhanced_svr.fit()
    predictions, actuals = forecaster_enhanced_svr.rolling_predict(data_enhanced_svr)
    metrics = forecaster_enhanced_svr.evaluate_rolling(data_enhanced_svr)
    print("\nEnhanced SVR Metrics:", metrics)
    forecaster_enhanced_svr.visualize(data_enhanced_svr, predictions)
    except ValueError as e:
    print(f"Enhanced SVR example error: {e}")

    print("\n--- ARDLX and ESN Backtesting ---")
    # Data for ARDLX and ESN
    y_ardlx = pd.Series(np.cumsum(np.random.randn(180)) + 5 * np.sin(np.linspace(0, 25, 180))
    z_ardlx = pd.Series(np.random.randn(180) * 0.5)

    # Parameters
    p_val = 3 # AR lags for y
    q_val = 2 # Lags for z
    delta_val = 2 # Lag for dummy 'd'
```

```

        backtester_ardlx_esn = RollingWindowBacktest(window_size=60, horizon=1)
        try:
        metrics_ardlx_esn, predictions_ardlx_esn, true_values_ardlx_esn = \
        backtester_ardlx_esn.run_backtest(y_ardlx, z_ardlx, p_val, q_val, delta_val)

        if metrics_ardlx_esn:
        print("\nARDLX/ESN Backtest Metrics:")
        for model, model_metrics in metrics_ardlx_esn.items():
        print(f"  {model.capitalize()} Model:")
        for metric_name, value in model_metrics.items():
        print(f"    {metric_name}: {value:.4f}")

        plt.figure(figsize=(14, 7))
        plt.plot(true_values_ardlx_esn, label='Actual', color='blue')
        plt.plot(predictions_ardlx_esn['ols'], '--', label='ARDLX (OLS) Forecast', color='orange')
        plt.plot(predictions_ardlx_esn['elastic_net'], '-.', label='ARDLX (Elastic Net) Forecast', color='green')
        plt.plot(predictions_ardlx_esn['esn'], ':', label='ESN Forecast', color='purple')
        plt.title("ARDLX & ESN Rolling Window Backtest")
        plt.xlabel("Time Index")
        plt.ylabel("Value")
        plt.legend()
        plt.grid(True)
        plt.show()

        except Exception as e:
        print(f"ARDLX/ESN Backtest example error: {e}")


        print("\n--- Adaptive Learning Forecaster ---")
        # Continuous Mode
        continuous_data_alf = np.sin(np.linspace(0, 100, 200) * 0.1) + np.random.normal(0, 0.2, 200)
        forecaster_alf_cont = AdaptiveLearningForecaster(
        models=[moving_average_model, momentum_model],
        mode="continuous", learning_rate=0.1, window_size=10
        )
        forecasts_alf_cont = []
        for i in range(20, len(continuous_data_alf)):
        forecast_val = forecaster_alf_cont.forecast(continuous_data_alf[:i])
        forecasts_alf_cont.append(forecast_val)
        forecaster_alf_cont.update(continuous_data_alf[i])

        print("Adaptive Learning Forecaster (Continuous) - First 5 forecasts:", forecasts_alf_cont[:5])
        print("Adaptive Learning Forecaster (Continuous) - Final weights:", forecaster_alf_cont.get_weights())

        # Binary Mode
        random_walk_alf = np.cumsum(np.random.normal(0, 1, 150))
        binary_data_alf = np.sign(np.diff(random_walk_alf))
        binary_data_alf = np.concatenate(([1], binary_data_alf)) # Ensure same length

        forecaster_alf_bin = AdaptiveLearningForecaster(
        models=[majority_vote_model, momentum_model],
        mode="binary", learning_rate=0.1, window_size=10, threshold=0.0
        )
        forecasts_alf_bin = []
        for i in range(20, len(binary_data_alf)):
        forecast_val = forecaster_alf_bin.forecast(binary_data_alf[:i])
        forecasts_alf_bin.append(forecast_val)
        forecaster_alf_bin.update(binary_data_alf[i])

        print("Adaptive Learning Forecaster (Binary) - First 5 forecasts:", forecasts_alf_bin[:5])
        print("Adaptive Learning Forecaster (Binary) - Final weights:", forecaster_alf_bin.get_weights())
```

Listing 4: Forecasting Models Usage Example

6 Troubleshooting and Tips

- `ValueError: Data length must be at least X...`: This error indicates that the input data provided to a class or method is too short for the operation it's trying to perform (e.g., fitting an AR model requires at least 2 data points, rolling windows need sufficient history). Adjust your `window_size`, `R`, or the length of your input data accordingly.
- `No numeric columns found...`: Some `DataAnalyzer` methods specifically operate on numeric columns. Ensure your DataFrame contains numeric data, or convert relevant columns to a numeric type.
- `MANOVA example error: Variable 'X' not found in DataFrame.`: Double-check that the `dependent_vars` and `independent_var` lists/strings you pass to the `manova` method exactly match the column names in your DataFrame.
- **Plots not showing up**: If you are running the script outside of an interactive environment (like Jupyter Notebook), plots will open in separate windows. You might need to close one plot window for the next one to appear. Ensure `plt.show()` is called after each plot.

- `RuntimeError: Failed to compute pseudo-inverse...` **(ESN)**: This can happen if the input data to the ESN is highly correlated or has very low variance, leading to a singular matrix. Try adjusting the `ridge_param` (increasing it can help with numerical stability) or ensure your input features are well-scaled and have sufficient variation.

- **Adaptive Learning Forecaster Warnings/Errors**: The `AdaptiveLearningForecaster` is designed to be robust, but individual models within the ensemble might fail if given insufficient or problematic data. The class attempts to handle these gracefully by printing warnings and using `np.nan` or default values. If you see many such warnings, inspect the data being passed to the `forecast` method at those specific iterations.

- **Performance**: Some operations, especially rolling window forecasts or hyperparameter tuning on large datasets, can be computationally intensive. Be mindful of the `window_size`, `num_samples`, and data length.

- **Reproducibility**: For examples involving random number generation (e.g., `np.random.randn`), `np.random.seed()` is used to ensure that results are reproducible across runs. If you want different random data, change or remove the seed.

- **Data Types**: Always ensure your data is in the expected format (e.g., `pd.DataFrame`, `np.ndarray`, `pd.Series`) as specified in the method signatures.

By following this manual, you should be well-equipped to utilize the powerful classes within the `unified_data_analysis_forecasting.py` script for your data analysis and forecasting needs.