Intoduction_to_stocksr_package

```r
library(stocksr)
```

Overview

The `stocksr` package offers tools for preprocessing, analyzing, and modeling S&P 500 stock data organized by sector. It simplifies the end-to-end workflow of:

- **Loading and cleaning** historical S&P 500 price data
- **Computing sector-level indices** from company-level stock prices
- **Generating lag features and technical indicators** for forecasting
- **Training Random Forest models** to predict sector index movements
- **Evaluating model performance** using RMSE, MAE, and R^2

Key Features

- **Sector-wise grouping**
  Automatically maps companies to sectors like Technology, Financials, or Health Care.

- **Technical indicator generation**
  Adds moving averages, returns, and momentum indicators for each sector's time series.

- **Random Forest pipeline**
  Trains models using default or customizable parameters via the `caret` package.

- **Performance comparison**
  Computes error metrics (RMSE, MAE, R^2) to benchmark models across different sectors.

- **Visualization**
  Generates plots showing predicted vs actual sector index values and feature importance rankings.

Overview

The `load_and_preprocess_data()` function in the **stocksr** package is designed to load historical S&P 500 stock price data from an Excel file and organize it into **sector-wise groups**.

Each sector is returned as a separate data frame, containing: - One column for `Date` - One column per company within that sector

This function is especially useful for: - Performing **sector-level financial analysis** - Building **machine learning models** on grouped company data

---

What the Function Does

The function performs the following steps:

- **Reads** an Excel file where:
- Rows represent daily price observations
- Columns represent individual companies
- **Maps** each company to its corresponding sector
  (e.g., Energy, Health Care)
- **Creates** a data frame for each sector that includes:
  - `Date` as the first column
  - Stock prices of the companies in that sector
- **Returns** a named list of sector-specific data frames

```
# Load and preprocess
sectors_data <- load_and_preprocess_data(df_cleaned2)

# View available sectors
names(sectors_data)
#> [1] "Energy"      "Health Care"
```

View a Sample Sector

```
# Preview the first few rows of the Energy sector
head(sectors_data$Energy)
#>         Date chevron conocophillips schlumberger
#> 1 2010-01-01   76.99        38.9317        65.09
#> 2 2010-01-04   79.06        40.0828        67.11
#> 3 2010-01-05   79.62        40.1209        67.30
#> 4 2010-01-06   79.63        40.4106        68.80
#> 5 2010-01-07   79.33        40.2505        69.51
#> 6 2010-01-08   79.47        40.6012        70.65
```

Sector Mapping Reference

The `load_and_preprocess_data()` function internally maps companies to the following sectors:

- **Energy:** Exxon Mobil, Chevron, ConocoPhillips, Schlumberger, EOG Resources, etc.
- **Health Care:** Johnson & Johnson, UnitedHealth Group, Eli Lilly, Pfizer, Merck & Company, etc.

Return Value

The function returns a **named list of data frames**, where each name corresponds to a sector.

Each sector data frame contains:

- `Date`: The trading date
- Stock prices for all companies in that sector (one column per company)

Inspect Sector Data Structure

```r
# Check the structure of the Energy sector data
str(sectors_data$Energy)
#> 'data.frame':    2870 obs. of  4 variables:
#>  $ Date           : Date, format: "2010-01-01" "2010-01-04" ...
#>  $ chevron        : num  77 79.1 79.6 79.6 79.3 ...
#>  $ conocophillips: num  38.9 40.1 40.1 40.4 40.3 ...
#>  $ schlumberger  : num  65.1 67.1 67.3 68.8 69.5 ...
```

Calculating Sector Index from S&P 500 Data

The `calculate_sector_index()` function computes the **average stock price across all companies in a sector** for each date, creating a simple sector-level time series. This is especially useful as input for forecasting or machine learning models.

How It Works

- Takes a **sector-specific data frame** (e.g., `sectors_data$Energy`) created using `load_and_preprocess_data()`
- Handles missing values using linear interpolation (`na.approx`) and bidirectional filling (`fill`)
- Computes a row-wise average of all company columns
- Returns a data frame with:
  - `Date`: The trading date
  - `sector_index`: The average of all company prices in that sector

Apply to this Data

```r
# Compute sector index for the Energy sector
tech_index <- calculate_sector_index(sectors_data$Energy)

# View the result
head(tech_index)
#>         Date sector_index
#> 1 2010-01-01     60.33723
#> 2 2010-01-04     62.08427
#> 3 2010-01-05     62.34697
#> 4 2010-01-06     62.94687
#> 5 2010-01-07     63.03017
#> 6 2010-01-08     63.57373
```

Calculating Technical Indicators for Sector Index

The `calculate_technical_indicators()` function in the `stocksr` package enriches a sector index time series with widely used technical indicators. These features are valuable for capturing patterns like momentum, trend, and volatility—key components for predictive modeling in finance.

What Are Technical Indicators?

The function calculates the following indicators:

Daily Returns* The daily return R_t is computed as the percent change from the previous day's index:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

Where: - P_t is the sector index at time t - R_t is the return on day t

Simple Moving Averages (SMA)* The SMA over n days is:

$$\text{SMA}_t = \frac{1}{n} \sum_{i=0}^{n-1} P_{t-i}$$

Used for smoothing trends over time. The function includes: - SMA over 5, 10, and 20 days

Exponential Moving Averages (EMA)* The EMA assigns more weight to recent prices. It's calculated recursively as:

$$\text{EMA}_t = \alpha \cdot P_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

Where $\alpha = \frac{2}{n+1}$ is the smoothing factor.

The function includes: - EMA over 5 and 10 days

Rolling Volatility The volatility (standard deviation) over a rolling window of n days:

$$\text{Volatility}_t = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n-1} (P_{t-i} - \bar{P})^2}$$

Computed for 5 and 10-day windows.

Relative Strength Index (RSI) The RSI measures the magnitude of recent gains and losses:

$$\text{RSI} = 100 - \left(\frac{100}{1 + \frac{\text{Average Gain}}{\text{Average Loss}}} \right)$$

Used to identify overbought (RSI > 70) or oversold (RSI < 30) conditions.

Lagged Features For time series models, the function creates lagged values:

- $\text{Lag}_k = P_{t-k}$
- $\text{Returns\_Lag}_k = R_{t-k}$ for $k = 1$ to 5

These help capture short-term dependencies and trends.

Apply to Sector Data

```r
tech_features <- calculate_technical_indicators(tech_index)
head(tech_features)
#>         Date sector_index log_price      returns  log_returns      MA5     EMA5
#> 1 2010-03-11     58.99900  4.077520 -0.002252580 -0.002255121 59.05925 58.87591
#> 2 2010-03-12     59.21890  4.081241  0.003727182  0.003720253 59.13379 58.99024
#> 3 2010-03-15     59.08460  4.078970 -0.002267857 -0.002270433 59.09787 59.02170
#> 4 2010-03-16     59.75673  4.090282  0.011375779  0.011311561 59.23829 59.26671
#> 5 2010-03-17     60.55257  4.103512  0.013317886  0.013229982 59.52236 59.69533
#> 6 2010-03-18     60.06397  4.095410 -0.008069022 -0.008101753 59.73535 59.81821
#>       MA10    EMA10     MA20    EMA20     MA50    EMA50        MACD MACD_signal
#> 1 58.29988 58.51287 57.98634 58.36753 59.52566 59.52566 -0.26033348  -0.9402428
#> 2 58.55539 58.64124 58.07375 58.44861 59.50329 59.51363 -0.13249964  -0.7786942
#> 3 58.74668 58.72185 58.15445 58.50918 59.44330 59.49680 -0.04924725  -0.6328048
#> 4 58.92152 58.91001 58.19737 58.62800 59.39149 59.50700  0.10773850  -0.4846962
#> 5 59.17667 59.20866 58.29412 58.81129 59.34361 59.54800  0.33676256  -0.3204044
#> 6 59.39730 59.36417 58.35258 58.93059 59.28428 59.56823  0.44567959  -0.1671876
#>   volatility5 volatility20    lag_1 log_return_lag_1    lag_2 log_return_lag_2
#> 1 0.008368733  0.010355585 59.13220      0.001312047 59.05467     -0.003541283
#> 2 0.004339017  0.010162117 58.99900     -0.002255121 59.13220      0.001312047
#> 3 0.003020810  0.010192372 59.21890      0.003720253 58.99900     -0.002255121
#> 4 0.005608456  0.008966452 59.08460     -0.002270433 59.21890      0.003720253
#> 5 0.007320907  0.009283416 59.75673      0.011311561 59.08460     -0.002270433
#> 6 0.008994578  0.009498558 60.55257      0.013229982 59.75673      0.011311561
#>      lag_5 log_return_lag_5 day_of_week month
#> 1 57.85763     -0.002477155         Thu   Mar
#> 2 58.84620      0.016941865         Fri   Mar
#> 3 59.26417      0.007077591         Mon   Mar
#> 4 59.05467     -0.003541283         Tue   Mar
#> 5 59.13220      0.001312047         Wed   Mar
#> 6 58.99900     -0.002255121         Thu   Mar
```

These engineered features are ready to be used in time series forecasting, machine learning pipelines, or financial dashboarding.

Random Forest Modeling for Sector Index Prediction

The `build_random_forest_model()` function trains a **Random Forest regression model** to predict a sector's daily index using engineered features such as returns, lag values, and technical indicators. It evaluates the model using standard performance metrics and provides feature importance to interpret the model.

What is a Random Forest?

A **Random Forest** is an ensemble machine learning algorithm that builds multiple decision trees and combines their predictions. In regression tasks, it averages the results from each tree to provide a robust, stable prediction.

Why use Random Forests for financial data?

- Handles non-linear relationships between features and response
- Robust to noise and overfitting

- Provides built-in feature importance for interpretation
- Performs well on datasets with many engineered variables (like technical indicators)

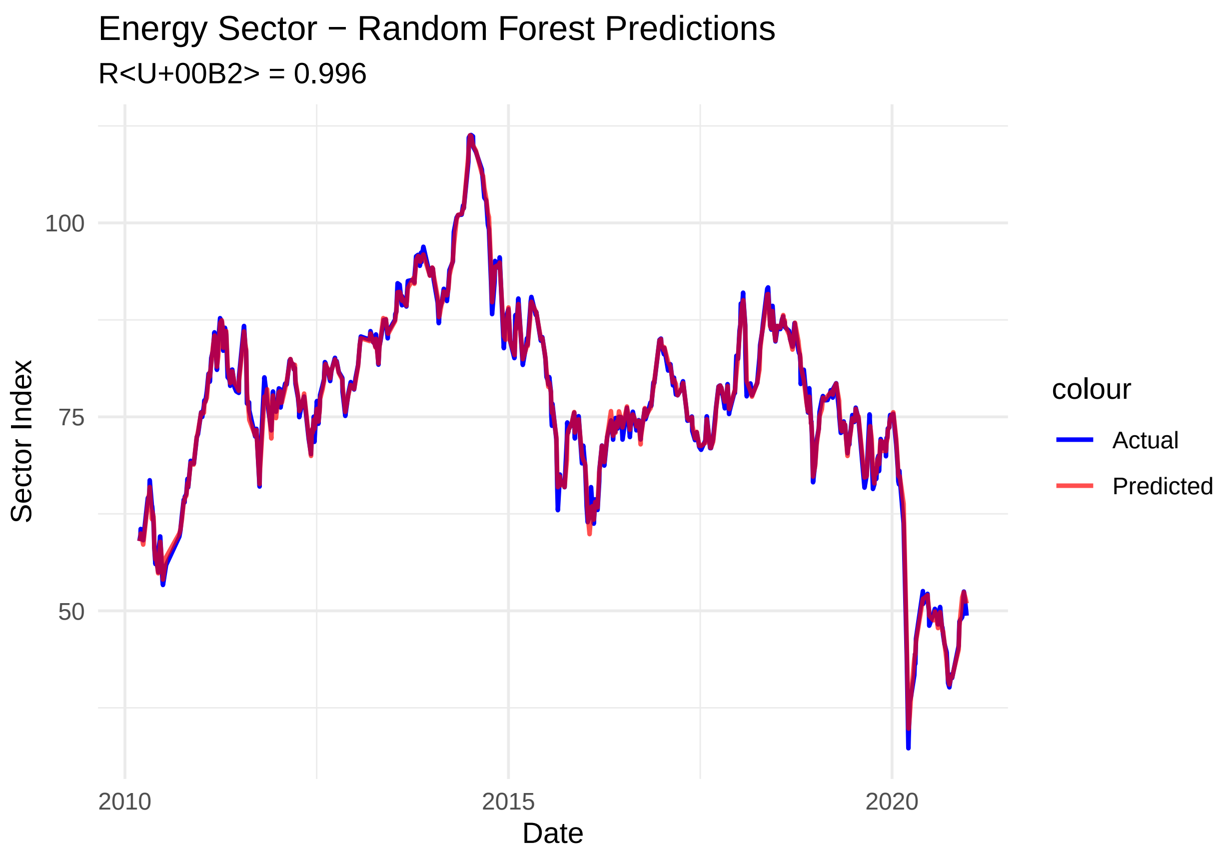
How the Function Works

The `build_random_forest_model()` function performs the following steps:

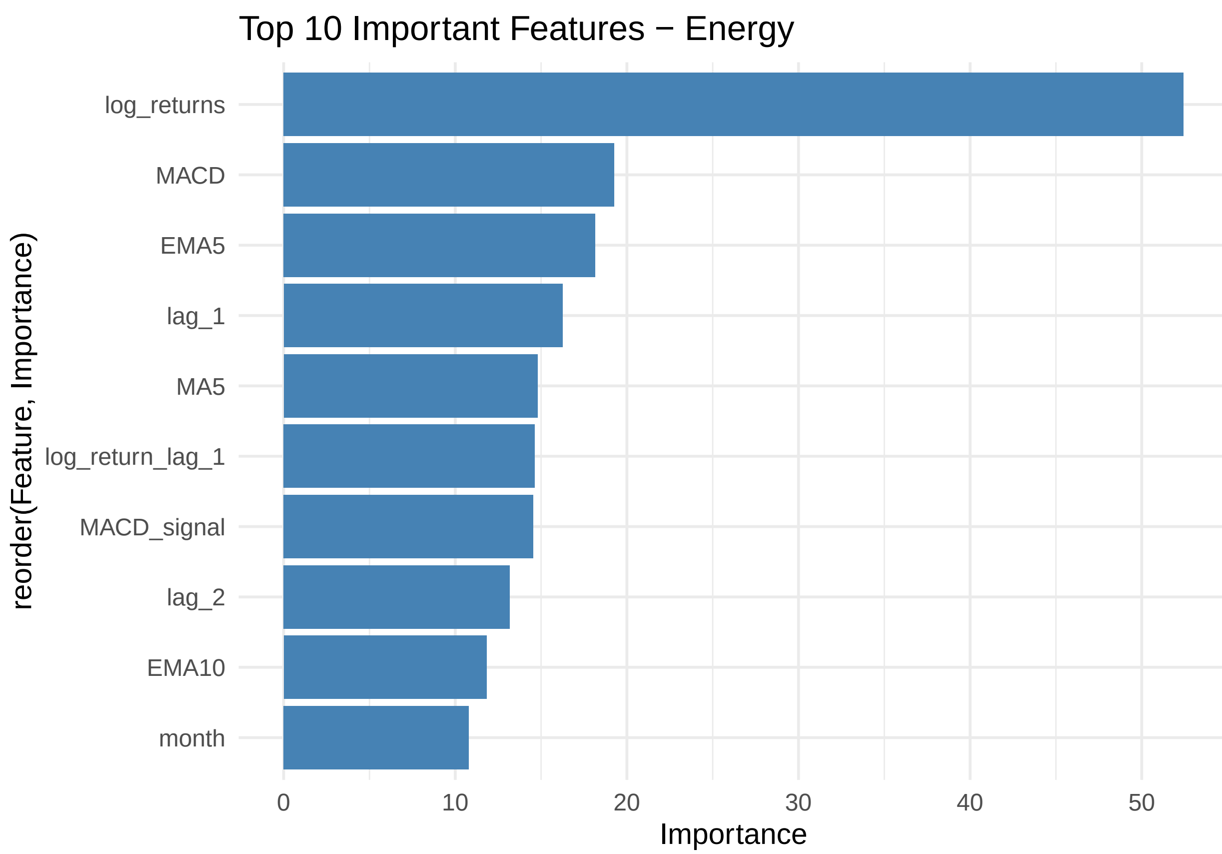
1. **Input**: A data frame with:
   - `Date`
   - `sector_index` (target)
   - Engineered features (technical indicators, returns, lags, etc.)
2. **Split the data**:
   - 80% for training
   - 20% for testing
   - Chronological order is preserved to respect time-dependence
3. **Train the model**:
   - A Random Forest is trained on the training set using `randomForest::randomForest()`
   - Parameters include `ntree = 100`, `mtry = sqrt(p)`, `nodesize = 5`
4. **Evaluate**:
   - Calculates:
     - **RMSE** (Root Mean Squared Error)
     - **MAE** (Mean Absolute Error)
     - R^2 (Coefficient of Determination)
   - Plots actual vs predicted values
5. **Feature Importance**:
   - Displays top variables ranked by `IncNodePurity`

Apply to Your Data

```r
# Train and evaluate the model
rf_results <- build_random_forest_model(tech_features, "Energy")
#>
#> --- Building Random Forest model for Energy sector ---
#> Model performance:
#> RMSE: 0.7691
#> MAE: 0.5274
#> R<U+00B2>: 0.9963
```



Energy Sector − Random Forest Predictions
R<U+00B2> = 0.996
Sector Index
100
75
50
2010
2015
2020
Date
colour
Actual
Predicted



Output

The function returns a named list:

Name	Description
`model`	The trained `randomForest` object
`rmse`	Root Mean Squared Error on the test set
`mae`	Mean Absolute Error on the test set
`r2`	R-squared score on the test set
`feature_importance`	Data frame ranking features by `IncNodePurity`

This modeling function is a key part of the `stocksr` pipeline and can be extended to all sectors for comparative performance analysis.

```r
# Step 3: View results
rf_results$rmse
#> [1] 0.7690529
rf_results$r2
#> [1] 0.9963346
head(rf_results$feature_importance)
#> NULL
```