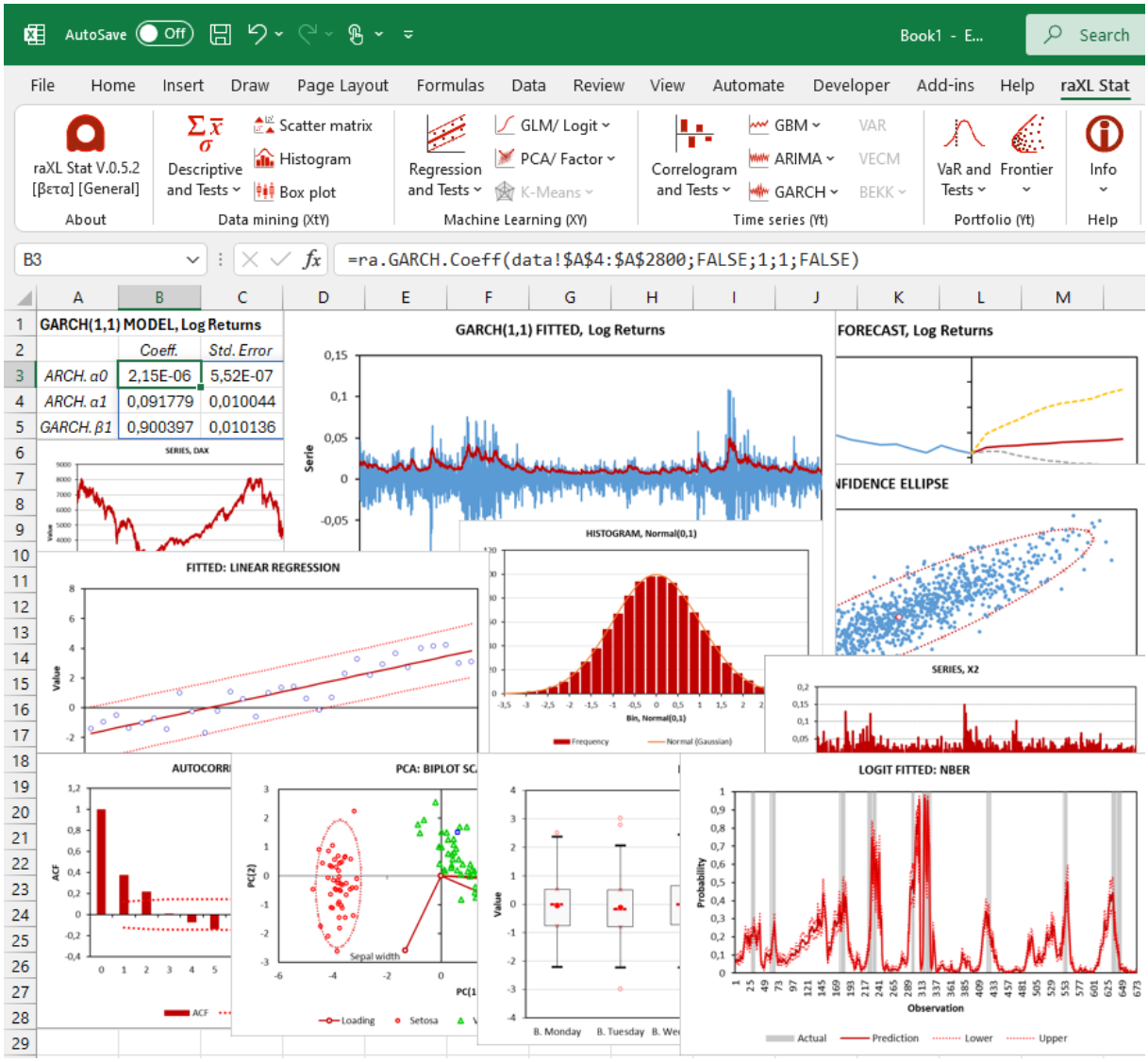
User Manual
raXL Stat v0.5.2 [$\beta\varepsilon\tau\alpha$]

Statistical Add-in for Data Science in Excel



August 27, 2025

Contents

raXL Stat *is an add-in for Microsoft Excel that turns your favorite spreadsheet into a quantitative and predictive analysis software, offering a collection of functions to create statistical, econometric, financial, and mathematical models.* You can call these functions directly from a spreadsheet and they will return the modeling results directly to it.

1 Introduction

raXL Stat is a statistical add-in for Microsoft Excel, developed in .NET with ExcelDna, that transforms spreadsheets into advanced tools for quantitative analysis, econometrics, finance, and time series. This manual details the installation, configuration, and use of all available user-defined functions (UDFs), organized by category, with full descriptions of each function, its purpose, and parameters.

raXL Stat is a statistical analysis software that offers easy-to-use tools to perform and deliver quality work in a short time. It is developed to be used by both beginners and experts. The easiest and most intuitive way to run the functions is through the ribbon menu. If necessary, the user can directly write the functions in the spreadsheet cells or can invoke the functions from VBA (Visual Basic for Application) programming.

2 System Requirements

- **Operating System**: Windows 7 or later.
- **Microsoft Excel**: Versions 2010 to 2024, or Microsoft 365 (32 or 64-bit).
- **Disk Space**: 10 MB free
- **RAM**: Minimum 512 MB (4 GB recommended).
- **Requires**:.NET Framework 4.5.2 or later.

3 Installation

- **Download**:
  - Visit Rubén Apaza's Blog or the GitHub repository.
  - Download the `raXL_Stat_v0.5.2.zip` file.

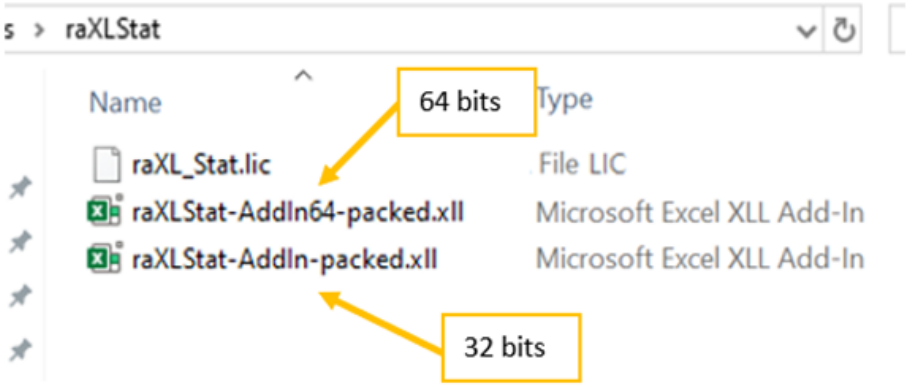


Figure 1: File.xll for Excel(32 or 64-bit)

- **Installation**:
  - The add-in does not require installation; simply open the `.xll` add-in and run it.

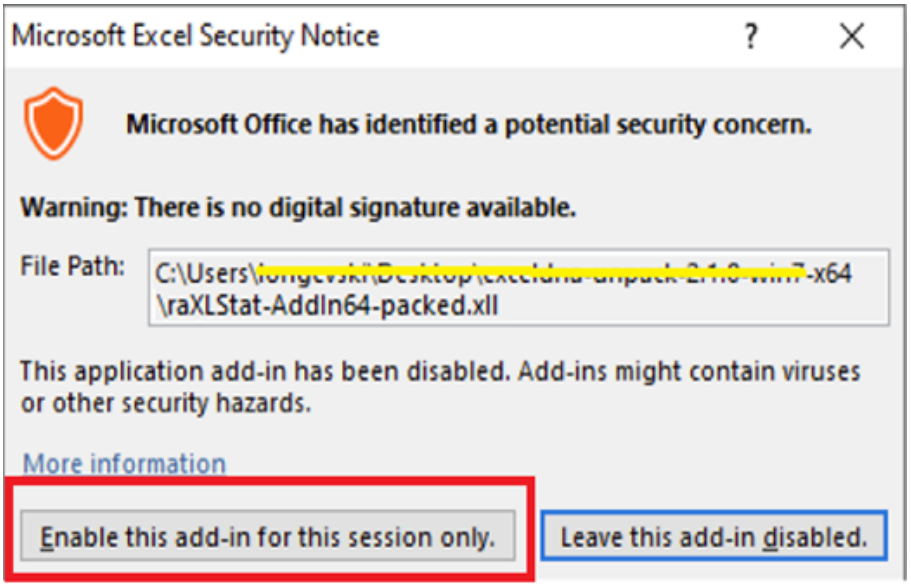


Figure 2: installation and simply open .xll

  - Optional unlock:
    - Locate the add-in file `.xll`, right-click on it,
    - Go to Properties...,
    - Check the box 'Unlock',
    - Click on OK.

- **Verification**:
  - A **raXL Stat** tab will appear in the toolbar, or functions will be available by typing `=ra.` in a cell.

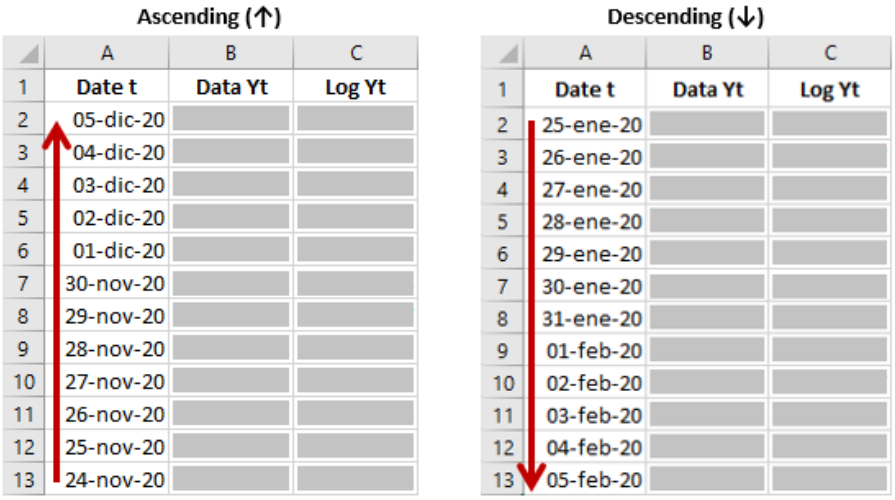
4 License and Activation

- **Verification**: Use `=ra.raXLStat.License()` with the provided password (e.g., associated with Universidad Mayor de San Andrés, UMSA).
- **Supported Versions**: Check with `=ra.raXLStat.Version()`.

5 Getting Started

- **Accessing Functions**:
  - Open Excel, type `=ra.` in a cell to list UDFs, or use the **raXL Stat** tab.
- **Data Preparation**:
  - Organize data in columns with clear headers (e.g., 'Sales', 'Date').
  - Check for missing or non-numeric data using `=ra.MissingData.Info(RangeX)`.
- **Conventions**:

- `RangeX`: Independent variables or data range (e.g., A1:B100).
- `RangeY` or `RangeYt`: Dependent variable or time series (e.g., C1:C100).
- `AscendentYt`: `TRUE` (most recent date at top), `FALSE` (most recent at bottom).
- `Alpha`: Significance level (e.g., 0.05).
- `Lag`: Lag (positive integer).
- `ConstantC`: `TRUE` (include intercept), `FALSE` (exclude intercept), or fixed value.
- `Label`: `TRUE` (include label, default), `FALSE` (no label).



	A	B	C
1	Date t	Data Yt	Log Yt
2	05-dic-20		
3	04-dic-20		
4	03-dic-20		
5	02-dic-20		
6	01-dic-20		
7	30-nov-20		
8	29-nov-20		
9	28-nov-20		
10	27-nov-20		
11	26-nov-20		
12	25-nov-20		
13	24-nov-20		

	A	B	C
1	Date t	Data Yt	Log Yt
2	25-ene-20		
3	26-ene-20		
4	27-ene-20		
5	28-ene-20		
6	29-ene-20		
7	30-ene-20		
8	31-ene-20		
9	01-feb-20		
10	02-feb-20		
11	03-feb-20		
12	04-feb-20		
13	05-feb-20		

Figure 3: Ascending-Descending data

6 Core Functionalities

The UDFs of **raXL Stat** are grouped by category. Each function is detailed with its **Function**, **Description**, and **Parameters**.

6.1 License and System

6.1.1 ra.raXLStat.License

- **Function**: `ra.raXLStat.License()`
- **Description**: Verifies the add-in's license status.

6.1.2 ra.raXLStat.Version

- **Function**: `ra.raXLStat.Version()`
- **Description**: Displays supported Excel versions.

6.2 Descriptive Statistics

6.2.1 ra.Descriptive.Stats

- **Function**: `ra.Descriptive.Stats(RangeX)`
- **Description**: Computes descriptive statistics (mean, median, standard deviation, min, max).

- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).

6.2.2 ra.Correlation.Matrix

- **Function**: `ra.Correlation.Matrix(RangeX, Population)`
- **Description**: Returns the correlation coefficient matrix of multiple data ranges, random variables, or time series.
- **Parameters**:
  - `RangeX`: Range of variables (e.g., A1:C100).
  - `Population`: Boolean. `TRUE` for population correlation, `FALSE` for sample.

6.2.3 ra.Correlation.Matrix.Test

- **Function**: `ra.Correlation.Matrix.Test(RangeX, LowTriang, Alpha)`
- **Description**: Generates a Pearson correlation matrix with options for coefficients, R-squared, p-values, or significance tests.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).
  - `LowTriang`: Integer. 0 (correlation coefficients R), 1 (R-squared), 2 (p-values for R based on t-Stat), 3 (test for R=0).
  - `Alpha`: Double. Significance level (default = 0.05).

6.2.4 ra.Correlation.Column

- **Function**: `ra.Correlation.Column(RangeX, RangeY)`
- **Description**: Returns the correlation coefficients between a dependent variable and multiple independent variables.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).

6.2.5 ra.Covariance.Matrix

- **Function**: `ra.Covariance.Matrix(RangeX, Population)`
- **Description**: Returns the covariance matrix for multiple data ranges, random variables, or time series.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).
  - `Population`: Boolean. `TRUE` for population covariance, `FALSE` for sample.

6.2.6 ra.Average.Column

- **Function**: `ra.Average.Column(RangeX)`
- **Description**: Returns a vector of means for multiple data sets.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).

6.2.7 ra.StdDev.Column

- **Function**: `ra.StdDev.Column(MatrixCovariance)`
- **Description**: Returns a vector of standard deviations from a covariance matrix.
- **Parameters**:
  - `MatrixCovariance`: Covariance matrix of returns.

6.2.8 ra.Average.Rows

- **Function**: `ra.Average.Rows(RangeX)`
- **Description**: Returns a column of means for a multi-column data set.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).

6.3 Time Series Analysis

Note: ARIMA and GARCH functions use the Maximum Likelihood Estimation (MLE) method together with the Newton-Raphson (NR) optimization algorithm, however, other optimization methods such as Levenberg-Marquardt, BHHH, BFGS and others will be added in development.

6.3.1 ra.Autocorrelation.ACF

- **Function**: `ra.Autocorrelation.ACF(RangeYt, Lag, Interval, Alpha)`
- **Description**: Calculates the autocorrelation function (ACF) for a specified lag k.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `Lag`: Integer. Lag k (positive, default = 0).
  - `Interval`: Boolean. `TRUE` for confidence interval, `FALSE` (default).
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.2 ra.Autocovariance.Matrix

- **Function**: `ra.Autocovariance.Matrix(RangeYt, MaxLag)`
- **Description**: Returns the autocovariance function (ACVF) matrix for a specified maximum lag k.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `MaxLag`: Integer. Maximum lag k (positive).

6.3.3 ra.Partial.Autocorr.PACF

- **Function**: `ra.Partial.Autocorr.PACF(RangeYt, Lag, Interval, Alpha)`
- **Description**: Calculates the partial autocorrelation function (PACF) for a specified lag k.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `Lag`: Integer. Lag k (positive, default = 0).
  - `Interval`: Boolean. `TRUE` for confidence interval, `FALSE` (default).
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.4 ra.LjungBox.Test

- **Function**: `ra.LjungBox.Test(RangeYt, Lag, AutocorrType, Alpha)`
- **Description**: Performs the Ljung-Box test for significant autocorrelation.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `AutocorrType`: Boolean. `TRUE` for ACF (default), `FALSE` for PACF.
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.5 ra.BoxPierce.Test

- **Function**: `ra.BoxPierce.Test(RangeYt, Lag, AutocorrType, Alpha)`
- **Description**: Performs the Box-Pierce test for autocorrelation.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `AutocorrType`: Boolean. `TRUE` for ACF (default), `FALSE` for PACF.
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.6 ra.DickeyFuller.ADF.Test

- **Function**: `ra.DickeyFuller.ADF.Test(RangeYt, AscendentYt, Lag, TermType, Alpha)`
- **Description**: Runs the Augmented Dickey-Fuller (ADF) test for stationarity.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `TermType`: Integer. 0 (no constant), 1 (constant, default), 2 (constant and trend).
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.7 ra.DickeyFuller.ADF.Reg

- **Function**: `ra.DickeyFuller.ADF.Reg(RangeYt, AscendentYt, Lag, TermType)`
- **Description**: Runs the autoregression for the Augmented Dickey-Fuller (ADF) test for stationarity.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `TermType`: Integer. 0 (no constant), 1 (constant, default), 2 (constant and trend).

6.3.8 ra.KPSS.Test

- **Function**: `ra.KPSS.Test(RangeYt, AscendentYt, Lag, TermType, Alpha)`
- **Description**: Performs the KPSS test for stationarity.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `TermType`: Integer. 1 (constant, default), 2 (constant and trend).
  - `Alpha`: Double. Significance level (default = 0.05).

6.3.9 ra.KPSS.Reg

- **Function**: `ra.KPSS.Reg(RangeYt, AscendentYt, Lag, TermType)`
- **Description**: Performs the autoregression for the KPSS test for stationarity.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Lag`: Integer. Number of lags (positive, default = 1).
  - `TermType`: Integer. 1 (constant, default), 2 (constant and trend).

6.3.10 ra.ARIMA.Coeff

- **Function**: `ra.ARIMA.Coeff(RangeYt, AscendentYt, ARp, DiffD, MAq, OptMethod)`
- **Description**: Estimates coefficients for an ARIMA(p,d,q) model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `ARp`: Integer. AR order (default = 1).
  - `DiffD`: Integer. Differencing order (default = 0).
  - `MAq`: Integer. MA order (default = 1).
  - `OptMethod`: String. Estimation method, default "NR" (Newton-Raphson).

6.3.11 ra.ARMA.Fitted

- **Function**: `ra.ARMA.Fitted(RangeYt, AscendentYt, Constant, RangeAR, RangeMA)`
- **Description**: Returns fitted values of an ARMA model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Constant`: Double. Model constant.
  - `RangeAR`: Range of AR (phi) coefficients.
  - `RangeMA`: Range of MA (theta) coefficients.

6.3.12 ra.ARMA.Forecast

- **Function**: `ra.ARMA.Forecast(RangeYt, AscendentYt, Constant, RangeAR, RangeMA, Forecast, Interval, Iterations)`
- **Description**: Forecasts future values using an ARMA model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Constant`: Double. Model constant.
  - `RangeAR`: Range of AR (phi) coefficients.
  - `RangeMA`: Range of MA (theta) coefficients.
  - `Forecast`: Integer. Number of periods to forecast (default = 10).
  - `Interval`: Integer. 0 (no interval, default), 1 (68%), 2 (95%), 3 (99.7%).
  - `Iterations`: Integer. Simulation iterations (default = 1000).

6.3.13 ra.GARCH.Coeff

- **Function**: `ra.GARCH.Coeff(RangeYt, AscendentYt, AlphaP, BetaQ, CondMean, ErrorDist, OptMethod)`
- **Description**: Estimates coefficients for a GARCH(p,q) model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `AlphaP`: Integer. ARCH order (default = 1).
  - `BetaQ`: Integer. GARCH order (default = 1).
  - `CondMean`: Boolean. `TRUE` to include conditional mean, `FALSE` (default).
  - `ErrorDist`: String. Error distribution, default "Normal".
  - `OptMethod`: String. Estimation method, default "NR" (Newton-Raphson).

6.3.14 ra.GARCH.Fitted

- **Function**: `ra.GARCH.Fitted(RangeYt, AscendentYt, RangeAlpha, RangeBeta)`
- **Description**: Returns fitted values for a GARCH(p,q) model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `RangeAlpha`: Range of ARCH (Alpha) coefficients.
  - `RangeBeta`: Range of GARCH (Beta) coefficients.

6.3.15 ra.GARCH.Forecast

- **Function**: `ra.GARCH.Forecast(RangeYt, AscendentYt, RangeAlpha, RangeBeta, Forecast, Interval, Iterations)`
- **Description**: Forecasts volatility using a GARCH(p,q) model.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `RangeAlpha`: Range of ARCH (Alpha) coefficients.
  - `RangeBeta`: Range of GARCH (Beta) coefficients.
  - `Forecast`: Integer. Number of periods to forecast (default = 10).
  - `Interval`: Integer. 0 (no interval), 1 (68%), 2 (95%, default), 3 (99.7%).
  - `Iterations`: Integer. Simulation iterations (default = 1000).

6.3.16 ra.GBM.Brownian.Coeff

- **Function**: `ra.GBM.Brownian.Coeff(RangeYt, AscendentYt)`
- **Description**: Calculates parameters (mean, standard deviation) for a Geometric Brownian Motion (GBM).
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).

6.3.17 ra.GBM.Brownian.Forecast

- **Function**: `ra.GBM.Brownian.Forecast(RangeMuSigma, Initial, Iterations, Forecast, Seed, Interval)`
- **Description**: Forecasts using Geometric Brownian Motion (GBM) with Monte Carlo simulation.
- **Parameters**:
  - `RangeMuSigma`: Range with mean and standard deviation.
  - `Initial`: Double. Initial value.

- `Iterations`: Integer. Number of simulations (default = 100).
- `Forecast`: Integer. Periods to forecast (default = 15).
- `Seed`: Integer. Random seed (default = 1234).
- `Interval`: Integer. 0 (simple random), 1 (68%), 2 (95%, default), 3 (99.7%).

6.3.18 ra.FanChart.Table

- **Function**: `ra.FanChart.Table(RangeY, IntervalType)`
- **Description**: Returns columns for fan chart, trend band, or lines interval plot.
- **Parameters**:
  - `RangeY`: Multi-column data range (e.g., A1:C100).
  - `IntervalType`: Integer. 0 (mean, no interval), 1 (trend band 68%-95%), 2 (fan chart 68%-95%, default).

6.4 Normality Tests

6.4.1 ra.JarqueBera.Test

- **Function**: `ra.JarqueBera.Test(RangeX, Population, Alpha)`
- **Description**: Performs the Jarque-Bera test for normality.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).
  - `Population`: Boolean. `TRUE` for population (default), `FALSE` for sample.
  - `Alpha`: Double. Significance level (default = 0.05).

6.4.2 ra.ShapiroWilk.Test

- **Function**: `ra.ShapiroWilk.Test(RangeX, Alpha)`
- **Description**: Performs the Shapiro-Wilk test for normality using the Royston algorithm (2 to 5000 observations).
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).
  - `Alpha`: Double. Significance level (default = 0.05).

6.4.3 ra.AndersonDarling.Test

- **Function**: `ra.AndersonDarling.Test(RangeX, DistType, Alpha)`
- **Description**: Performs the Anderson-Darling test for normality.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).
  - `DistType`: Integer. 0 (generic), 1 (normal, default), 2 (unmodified normal), 3 (log-normal).
  - `Alpha`: Double. Significance level (default = 0.05).

6.4.4 ra.Skew.S

- **Function**: `ra.Skew.S(RangeX)`
- **Description**: Returns the skewness value of the distribution based on the sample.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).

6.4.5 ra.Kurt.S

- **Function**: `ra.Kurt.S(RangeX)`
- **Description**: Returns the kurtosis value of the distribution based on the sample.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).

6.4.6 ra.Skew.P

- **Function**: `ra.Skew.P(RangeX)`
- **Description**: Returns the skewness value of the distribution based on the population.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).

6.4.7 ra.Kurt.P

- **Function**: `ra.Kurt.P(RangeX, Subtract3)`
- **Description**: Returns the kurtosis value of the distribution based on the population.
- **Parameters**:
  - `RangeX`: Data range (e.g., A1:A100).
  - `Subtract3`: Boolean. `TRUE` to subtract 3 (normal distribution kurtosis = 0), `FALSE` (no subtraction).

6.5 Linear Regression (OLS)

6.5.1 ra.LinearReg.Coeff

- **Function**: `ra.LinearReg.Coeff(RangeX, RangeY, ConstantC, RobustStdErrHC, Alpha)`
- **Description**: Estimates coefficients, standard errors, t-statistics, p-values, and confidence intervals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `RobustStdErrHC`: Integer. 0=HC0, 1=HC1, 2=HC2, 3=HC3, 4=HC4, 5=HAC, 6=OLS (default).
  - `Alpha`: Double. Significance level (default = 0.05).

6.5.2 ra.LinearReg.Fitted

- **Function**: `ra.LinearReg.Fitted(RangeX, RangeY, ConstantC, Interval, Alpha)`
- **Description**: Returns fitted values and optional confidence or prediction intervals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `Interval`: Integer. 0 (no interval, default), 1 (confidence), 2 (prediction).
  - `Alpha`: Double. Significance level (default = 0.05).

6.5.3 ra.LinearReg.Forecast

- **Function**: `ra.LinearReg.Forecast(RangeX, RangeY, RangeXo, ConstantC, Interval, Alpha)`
- **Description**: Generates forecasts for new data with optional intervals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `RangeXo`: New independent variable data (e.g., E1:F10).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `Interval`: Boolean. `TRUE` (include interval), `FALSE` (forecast only, default).
  - `Alpha`: Double. Significance level (default = 0.05).

6.5.4 ra.LinearReg.Residuals

- **Function**: `ra.LinearReg.Residuals(RangeX, RangeY, ConstantC, ResidualType, Alpha, Critical)`
- **Description**: Computes residuals, standardized, studentized, Pearson, DFFITS, leverage, or Cook's distance.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `ResidualType`: Integer. 0=residuals(default), 1=σ, 2=standardized, 3=studentized, 4=Pearson, 5=DFFITS, 6=leverage, 7=Cook's distance.
  - `Alpha`: Double. Significance level for z-Stat and t-Stat (default = 0.3073/2).
  - `Critical`: Double. Critical value for Cook's (4/n), Leverage (3*mu), DFFITS (1).

6.5.5 ra.LinearReg.Residuals.2

- **Function**: `ra.LinearReg.Residuals.2(RangeX, RangeY, ConstantC, MeasureType, Alpha, Critical)`
- **Description**: Generates a table for XY scatter plot and computes diagnostic measures for outlier detection.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `MeasureType`: Integer. 0=residuals-fitted(default), 1=standardized-fitted, 2=studentized-fitted, 3=standardized-leverage, 4=studentized-leverage, 5=studentized-Cook, 6=studentized-DFFITS.
  - `Alpha`: Double. Significance level for z-Stat and t-Stat (default = 0.05).
  - `Critical`: Double. Critical value for Cook's (4/n), Leverage (2*mu), DFFITS (1).

6.5.6 ra.LinearReg.Influence

- **Function**: `ra.LinearReg.Influence(RangeX, RangeY, readjust)`
- **Description**: Calculates Pratt's measure for the relative importance of variables in R^2.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `readjust`: Boolean. `TRUE` to adjust indices to 100% (default), `FALSE` (no adjustment).

6.5.7 ra.LinearReg.Anova

- **Function**: `ra.LinearReg.Anova(RangeX, RangeY, ConstantC)`
- **Description**: Generates an ANOVA table with sums of squares, mean squares, F-statistic, and p-value.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.

6.5.8 ra.LinearReg.Stat

- **Function**: `ra.LinearReg.Stat(RangeX, RangeY, ConstantC)`
- **Description**: Calculates fit metrics such as R^2, adjusted R^2, standard error, Durbin-Watson, LLH, AIC, and BIC.
- **Parameters**:

- `RangeX`: Independent variables (e.g., A1:B100).
- `RangeY`: Dependent variable (e.g., C1:C100).
- `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.

6.5.9 ra.RamseyRESET.Test

- **Function**: `ra.RamseyRESET.Test(RangeX, RangeY, ConstantC, Power, Alpha)`
- **Description**: Performs the Ramsey RESET test for misspecification by evaluating whether polynomial terms of fitted values improve the model.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `Power`: Integer. Maximum polynomial order of fitted values (default = 1 for quadratic).
  - `Alpha`: Double. Significance level (default = 0.05).

6.5.10 ra.RamseyRESET.Reg

- **Function**: `ra.RamseyRESET.Reg(RangeX, RangeY, ConstantC, Power)`
- **Description**: Performs the regression for the Ramsey RESET test.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `Power`: Integer. Maximum polynomial order of fitted values (default = 1 for quadratic).

6.5.11 ra.RecursiveCUSUM

- **Function**: `ra.RecursiveCUSUM(RangeX, RangeY, ConstantC, RecursiveType)`
- **Description**: Performs the recursive CUSUM test to detect structural breaks or parameter instability using recursive residuals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept), or fixed value.
  - `RecursiveType`: Integer. 0 (CUSUM + 5% interval, default), 1 (recursive residuals), 2 (standardized recursive residuals).

6.5.12 ra.Acorr.BreuschGodfrey.Test

- **Function**: `ra.Acorr.BreuschGodfrey.Test(RangeX, RangeY, ConstantC, Lag, Alpha, Chi2)`
- **Description**: Performs the Breusch-Godfrey test for autocorrelation in regression residuals up to a specified lag.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Lag`: Integer. Number of lags to test (default = 2).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Chi2`: Boolean. `TRUE` for Chi-squared test (default), `FALSE` for F-Stat.

6.5.13 ra.Acorr.BreuschGodfrey.Reg

- **Function**: `ra.Acorr.BreuschGodfrey.Reg(RangeX, RangeY, ConstantC, Lag)`
- **Description**: Performs the regression for the Breusch-Godfrey test.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Lag`: Integer. Number of lags to test (default = 2).

6.5.14 ra.Acorr.ACF.Test

- **Function**: `ra.Acorr.ACF.Test(RangeYt, Lag, Alpha)`
- **Description**: Tests for significant autocorrelation in residuals using the autocorrelation function (ACF) up to a specified lag.
- **Parameters**:
  - `RangeYt`: Time series or residuals (e.g., A1:A100).
  - `Lag`: Integer. Number of lags to test (default = 1).
  - `Alpha`: Double. Significance level (default = 0.05).

6.5.15 ra.Acorr.DurbinWatson.Stat

- **Function**: `ra.Acorr.DurbinWatson.Stat(RangeYt)`
- **Description**: Calculates the Durbin-Watson statistic to test for first-order autocorrelation in residuals.
- **Parameters**:
  - `RangeYt`: Time series or residuals (e.g., A1:A100).

6.5.16 ra.MulticolLin.VIF

- **Function**: `ra.MulticolLin.VIF(RangeX, Column, MeasureType)`
- **Description**: Calculates the Variance Inflation Factor (VIF) for each independent variable to assess multicollinearity.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).
  - `Column`: Integer. Column index of the matrix X (default = 0 for all columns).
  - `MeasureType`: Integer. 0 (VIF, default), 1 (Tolerance), 2 (R-squared).

6.5.17 ra.MulticolLin.Lambda

- **Function**: `ra.MulticolLin.Lambda(RangeX)`
- **Description**: Calculates the eigenvalues (λ) of the correlation matrix of independent variables to assess multicollinearity.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).

6.5.18 ra.MulticolLin.Kappa

- **Function**: `ra.MulticolLin.Kappa(RangeX)`
- **Description**: Calculates the condition number (Kappa) of the matrix of independent variables to detect multicollinearity.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).

6.5.19 ra.MulticolLin.Nu

- **Function**: `ra.MulticolLin.Nu(RangeX)`
- **Description**: Returns variances associated with principal axes (loadings) or eigenvectors (λ) of the correlation matrix.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).

6.5.20 ra.Leverage

- **Function**: `ra.Leverage(RangeX, Constant, MeasureType)`
- **Description**: Calculates leverage values for each observation to identify influential points in independent variables.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).
  - `Constant`: Boolean. `TRUE` (include intercept, default), `FALSE` (no intercept).
  - `MeasureType`: Integer. 0 (leverage, default), 1 (leverage + 2*mu), 2 (leverage + 2*mu outliers).

6.5.21 ra.Normalized.Distances

- **Function**: `ra.Normalized.Distances(RangeX, Alpha, MeasureType)`
- **Description**: Calculates normalized distances for observations to detect outliers in independent variables.
- **Parameters**:
  - `RangeX`: Multi-column data range (e.g., A1:C100).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `MeasureType`: Integer. 0 (distances, default), 1 (Chi² critical).

6.6 Heteroscedasticity Tests

6.6.1 ra.Het.BreuschPagan.Test

- **Function**: `ra.Het.BreuschPagan.Test(RangeX, RangeY, ConstantC, Alpha, Chi2)`
- **Description**: Performs the Breusch-Pagan test for heteroscedasticity in regression residuals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Chi2`: Boolean. `TRUE` for Chi-squared test (default), `FALSE` for F-Stat.

6.6.2 ra.Het.BreuschPagan.Reg

- **Function**: `ra.Het.BreuschPagan.Reg(RangeX, RangeY, ConstantC)`
- **Description**: Performs the regression for the Breusch-Pagan test.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).

6.6.3 ra.Het.White.Test

- **Function**: `ra.Het.White.Test(RangeX, RangeY, ConstantC, Alpha, Chi2)`
- **Description**: Performs the White test for heteroscedasticity in regression residuals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Chi2`: Boolean. `TRUE` for Chi-squared test (default), `FALSE` for F-Stat.

6.6.4 ra.Het.White.Reg

- **Function**: `ra.Het.White.Reg(RangeX, RangeY, ConstantC)`
- **Description**: Performs the regression for the White test.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).

6.6.5 ra.Het.ARCH.Test

- **Function**: `ra.Het.ARCH.Test(RangeX, RangeY, ConstantC, Lag, Alpha, Chi2)`
- **Description**: Performs the ARCH test for conditional heteroscedasticity in regression residuals.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Lag`: Integer. Number of lags (default = 1).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Chi2`: Boolean. `TRUE` for Chi-squared test (default), `FALSE` for F-Stat.

6.6.6 ra.Het.ARCH.Reg

- **Function**: `ra.Het.ARCH.Reg(RangeX, RangeY, ConstantC, Lag)`
- **Description**: Performs the regression for the ARCH test.
- **Parameters**:
  - `RangeX`: Independent variables (e.g., A1:B100).
  - `RangeY`: Dependent variable (e.g., C1:C100).
  - `ConstantC`: Boolean or Double. `TRUE` (intercept, default), `FALSE` (no intercept).
  - `Lag`: Integer. Number of lags (default = 1).

6.7 Portfolio Analysis

6.7.1 ra.Portfolio.Risk

- **Function**: `ra.Portfolio.Risk(MatrixCovariance, RangeW)`
- **Description**: Calculates the portfolio risk (standard deviation) for a covariance matrix and weights.
- **Parameters**:
  - `MatrixCovariance`: Covariance matrix (e.g., A1:C3).
  - `RangeW`: Weights range (e.g., D1:D3).

6.7.2 ra.Portfolio.Return

- **Function**: `ra.Portfolio.Return(VectorMeanReturns, VectorWeights)`
- **Description**: Returns the expected return of a portfolio.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `VectorWeights`: Vector of portfolio weights.

6.7.3 ra.Portfolio.Weights.Optimal

- **Function**: `ra.Portfolio.Weights.Optimal(VectorMeanReturns, MatrixCovariance, ExpectedReturn)`
- **Description**: Returns the optimal investment weights in a portfolio on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `ExpectedReturn`: Double. Required or expected return.

6.7.4 ra.Portfolio.Weights.Tangency

- **Function**: `ra.Portfolio.Weights.Tangency(VectorMeanReturns, MatrixCovariance, RiskFree)`
- **Description**: Returns the tangency portfolio weights on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `RiskFree`: Double. Risk-free rate.

6.7.5 ra.Portfolio.Market.Line.Effic

- **Function**: `ra.Portfolio.Market.Line.Effic(VectorMeanReturns, MatrixCovariance, RiskFree, FrontierPoints)`
- **Description**: Returns expected return and risk for the Capital Market Line (CML) in a portfolio on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `RiskFree`: Double. Risk-free rate.
  - `FrontierPoints`: Integer. Number of points on the efficient frontier.

6.7.6 ra.Portfolio.Market.Line.Opti

- **Function**: `ra.Portfolio.Market.Line.Opti(VectorMeanReturns, MatrixCovariance, RiskFree, ExpectedReturn)`
- **Description**: Returns the expected risk for the Capital Market Line (CML) optimal portfolio on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `RiskFree`: Double. Risk-free rate.
  - `ExpectedReturn`: Double. Target expected return.

6.7.7 ra.Portfolio.Weights.MinVar

- **Function**: `ra.Portfolio.Weights.MinVar(VectorMeanReturns, MatrixCovariance)`
- **Description**: Returns the minimum variance portfolio weights on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.

6.7.8 ra.Portfolio.Frontier.Efficient

- **Function**: `ra.Portfolio.Frontier.Efficient(VectorMeanReturns, MatrixCovariance, FrontierPoints)`
- **Description**: Returns expected return and risk for the efficient portfolio frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `FrontierPoints`: Integer. Number of points on the efficient frontier.

6.7.9 ra.Portfolio.Frontier.Optimal

- **Function**: `ra.Portfolio.Frontier.Optimal(VectorMeanReturns, MatrixCovariance, FrontierPoints)`
- **Description**: Returns the optimal expected return and risk for a portfolio on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `FrontierPoints`: Integer. Number of points on the efficient frontier.

6.7.10 ra.Portfolio.Risk.Optimal

- **Function**: `ra.Portfolio.Risk.Optimal(VectorMeanReturns, MatrixCovariance, ExpectedReturn`
- **Description**: Returns the optimal expected risk (standard deviation) for a portfolio on the efficient frontier.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `ExpectedReturn`: Double. Target expected return.

6.7.11 ra.Portfolio.Simulation

- **Function**: `ra.Portfolio.Simulation(VectorMeanReturns, MatrixCovariance, IterSamples, Seed)`
- **Description**: Returns risk and return points for a portfolio simulation using Monte Carlo methods.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `IterSamples`: Integer. Number of simulation iterations.
  - `Seed`: Integer. Random seed (e.g., 1234).

6.7.12 ra.Portfolio.Simul.Frontier

- **Function**: `ra.Portfolio.Simul.Frontier(VectorMeanReturns, MatrixCovariance, IterSamples, Seed, LevelZoom)`
- **Description**: Returns efficient risk and return points for a portfolio simulation (experimental).
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `IterSamples`: Integer. Number of simulation iterations.
  - `Seed`: Integer. Random seed (e.g., 1234).
  - `LevelZoom`: Integer. Zoom level for frontier points (e.g., 1, 2).

6.7.13 ra.VaR.Historical

- **Function**: `ra.VaR.Historical(RangeR, Alpha, AscendR)`
- **Description**: Calculates historical Value at Risk (VaR) for a series of returns.
- **Parameters**:
  - `RangeR`: Returns series (e.g., A1:A100).
  - `Alpha`: Double. Confidence level (default = 0.05).
  - `AscendR`: Boolean. `TRUE` for ascending order, `FALSE` (default).

6.7.14 ra.VaR.Parametric

- **Function**: `ra.VaR.Parametric(RangeR, Alpha, DistType)`
- **Description**: Calculates parametric Value at Risk (VaR) for a series of returns.
- **Parameters**:
  - `RangeR`: Returns series (e.g., A1:A100).
  - `Alpha`: Double. Confidence level (default = 0.05).
  - `DistType`: Integer. 0 (normal, default), 1 (t-Student), 2 (lognormal).

6.7.15 ra.VaR.VarCovar

- **Function**: `ra.VaR.VarCovar(VectorMeanReturns, MatrixCovariance, VectorWeights, Alpha)`
- **Description**: Returns the variance-covariance Value at Risk (P VaR) of a portfolio, assuming normality.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `VectorWeights`: Vector of portfolio weights.
  - `Alpha`: Double. Significance level (e.g., 0.01, 0.05, 0.10).

6.7.16 ra.CVaR.Historical

- **Function**: `ra.CVaR.Historical(RangeReturns, Alpha, Excludes)`
- **Description**: Returns the historical Conditional Value at Risk (H CVaR) or expected shortfall of a portfolio.
- **Parameters**:
  - `RangeReturns`: Single or multi-column data range of returns (e.g., A1:C100).
  - `Alpha`: Double. Significance level (e.g., 0.01, 0.05, 0.10).
  - `Excludes`: Boolean. `TRUE` to exclude first and last percentile values, `FALSE` to include.

6.7.17 ra.CVaR.Parametric

- **Function**: `ra.CVaR.Parametric(RangeReturns, Alpha, Population)`
- **Description**: Returns the parametric Conditional Value at Risk (P CVaR) or expected shortfall, assuming normality.
- **Parameters**:
  - `RangeReturns`: Multi-column data range of returns (e.g., A1:C100).
  - `Alpha`: Double. Significance level (e.g., 0.01, 0.05, 0.10).
  - `DistType`: Boolean. `TRUE` for population variance, `FALSE` for sample variance.

6.7.18 ra.CVaR.VarCovar

- **Function**: `ra.CVaR.VarCovar(VectorMeanReturns, MatrixCovariance, VectorWeights, Alpha)`
- **Description**: Returns the variance-covariance Conditional Value at Risk (P CVaR) or expected shortfall, assuming normality.
- **Parameters**:
  - `VectorMeanReturns`: Vector of mean asset returns.
  - `MatrixCovariance`: Covariance matrix of returns.
  - `VectorWeights`: Vector of portfolio weights.
  - `Alpha`: Double. Significance level (e.g., 0.01, 0.05, 0.10).

6.7.19 ra.VaR.BackTest

- **Function**: `ra.VaR.BackTest(RangeR, VaR, Alpha)`
- **Description**: Performs backtesting for Value at Risk (VaR).
- **Parameters**:
  - `RangeR`: Returns series (e.g., A1:A100).
  - `VaR`: VaR series (e.g., B1:B100).
  - `Alpha`: Double. Confidence level (default = 0.05).

6.7.20 ra.VaR.Kupiec.Test

- **Function**: `ra.VaR.Kupiec.Test(RangeR, VaR, Alpha, Label)`
- **Description**: Performs the Kupiec test for Value at Risk (VaR) backtesting.
- **Parameters**:
  - `RangeR`: Returns series (e.g., A1:A100).
  - `VaR`: VaR series (e.g., B1:B100).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.8 Visualization

6.8.1 ra.Histogram.Table

- **Function**: `ra.Histogram.Table(RangeX, CurveType, FrequencyType)`
- **Description**: Generates a table for histogram construction with automatic bins.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `CurveType`: Integer. 0 (no curve, default), 1 (cumulative), 2 (normal).
  - `FrequencyType`: Boolean. `TRUE` for absolute frequency (default), `FALSE` for relative frequency.

6.8.2 ra.Histogram.Table.Bin

- **Function**: `ra.Histogram.Table.Bin(RangeX, CurveType, FrequencyType, Start, Step, Stop)`
- **Description**: Generates a table for histogram construction with user-specified bins.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `CurveType`: Integer. 0 (no curve, default), 1 (cumulative), 2 (normal).
  - `FrequencyType`: Boolean. `TRUE` for absolute frequency (default), `FALSE` for relative frequency.
  - `Start`: Double. Minimum bin value.
  - `Step`: Double. Step size between bins.
  - `Stop`: Double. Maximum bin value.

6.8.3 ra.BoxPlot.Table

- **Function**: `ra.BoxPlot.Table(RangeX, Outliers, IQR)`
- **Description**: Generates a table for box-and-whisker plots.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `Outliers`: Boolean. `TRUE` to detect outliers, `FALSE` (default).
  - `IQR`: Double. Interquartile range multiplier (default = 1.5).

6.8.4 ra.Curve.Distribution

- **Function**: `ra.Curve.Distribution(RangeX, Population, Distribution)`
- **Description**: Generates a table of density or frequencies with automatic bins for plotting a curve.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `Population`: Boolean. `TRUE` for population variance, `FALSE` for sample variance.
  - `Distribution`: String. Curve type, default "Normal" (Gaussian).

6.8.5 ra.Bins

- **Function**: `ra.Bins(RangeX, Bins)`
- **Description**: Generates a columnn table of Bins useful for plotting.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `Population`: Integer: Number of the bins. All the bins have the same size.

6.9 Utilities

6.9.1 ra.MissingData.Info

- **Function**: `ra.MissingData.Info(RangeX)`
- **Description**: Identifies missing or non-numeric data in a range.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).

6.9.2 ra.Difference

- **Function**: `ra.Difference(RangeYt, AscendentYt, DifferenceD)`
- **Description**: Returns a column with the difference operation applied to a time series.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `DifferenceD`: Integer. Order of differencing (default = 1).

6.9.3 ra.Interpolate.Point

- **Function**: `ra.Interpolate.Point(x, x1, x2, y1, y2, interpType)`
- **Description**: Performs linear, logarithmic, or harmonic interpolation.
- **Parameters**:
  - `x`: Double. Value to interpolate.
  - `x1`, `x2`: Double. Reference values.
  - `y1`, `y2`: Double. Corresponding values.
  - `interpType`: Integer. 0 (linear), 1 (logarithmic), 2 (harmonic).

6.9.4 ra.Flip

- **Function**: `ra.Flip(RangeX, Flip)`
- **Description**: Flips a vertical range of cells if the condition is true.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `Flip`: Boolean. `TRUE` to flip vertically (default), `FALSE` to duplicate.

6.9.5 ra.Transpose

- **Function**: `ra.Transpose(RangeX, Transpose)`
- **Description**: Transposes a range from vertical to horizontal or vice versa.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `Transpose`: Boolean. `TRUE` to transpose (default), `FALSE` to duplicate.

6.9.6 ra.ShowLag

- **Function**: `ra.ShowLag(RangeYt, AscendentYt, ShowLag)`
- **Description**: Shows a column with lagged values of a time series.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `ShowLag`: Integer. Lag value (positive, default = all lags).

6.9.7 ra.Rate.LN

- **Function**: `ra.Rate.LN(RangeYt, AscendentYt)`
- **Description**: Converts a positive time series into a time series of natural logarithmic (LN) rate of change.
- **Parameters**:
  - `RangeYt`: Time series with positive values (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).

6.9.8 ra.Rate.Log

- **Function**: `ra.Rate.Log(RangeYt, AscendentYt, Base)`
- **Description**: Converts a positive time series into a time series of logarithmic (LOG) rate of change.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).
  - `Base`: Integer. Logarithmic base (default = 10).

6.9.9 ra.Rate.Change

- **Function**: `ra.Rate.Change(RangeYt, AscendentYt)`
- **Description**: Converts a positive time series into a time series of rate of change.
- **Parameters**:
  - `RangeYt`: Time series (e.g., A1:A100).
  - `AscendentYt`: Boolean. `TRUE` for ascending order, `FALSE` (default).

6.9.10 ra.Standardize

- **Function**: `ra.Standardize(RangeX, standardized)`
- **Description**: Returns standardized (mean, sigma) values from a distribution.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `standardized`: Boolean. `TRUE` to standardize (default), `FALSE` to duplicate.

6.9.11 ra.Normalize

- **Function**: `ra.Normalize(RangeX, normalized)`
- **Description**: Returns normalized N(0,1) values from a distribution.
- **Parameters**:
  - `RangeX`: Single or multi-column data range (e.g., A1:C100).
  - `normalized`: Boolean. `TRUE` to normalize (default), `FALSE` to duplicate.

6.10 Binary Models (Logit and Probit)

6.10.1 ra.Logit.Coeff

- **Function**: `ra.Logit.Coeff(rangeX, rangeY, choiceCoeff, Alpha, Label, Iterations, OptMethod)`
- **Description**: Estimates coefficients, standard errors, z-statistics, p-values, and confidence intervals for logistic regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `choiceCoeff`: Integer. 0 (coefficients logit, default), 1 (marginal effects), 2 (odds ratio).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `Iterations`: Integer. Maximum iterations for convergence (default = 20).
  - `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).

6.10.2 ra.Logit.Stat

- **Function**: `ra.Logit.Stat(rangeX, rangeY, Iterations, OptMethod, Label)`
- **Description**: Computes fit metrics like Log-Likelihood (LLH), AIC, BIC, McFadden R^2, and LR-Chi^2 test for logistic regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `Iterations`: Integer. Maximum iterations for convergence (default = 20).
  - `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.3 ra.Logit.Gof

- **Function**: `ra.Logit.Gof(rangeX, rangeY, Iterations, OptMethod, Label, groupsLS)`
- **Description**: Computes goodness-of-fit tests Pearson Chi² or Hosmer-Lemeshow (experimental) for logistic regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `Iterations`: Integer. Maximum iterations for convergence (default = 20).
  - `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `groupsLS`: Integer. Number of groups for Hosmer-Lemeshow test (default = auto).

6.10.4 ra.Logit.Fitted

- **Function**: `ra.Logit.Fitted(rangeX, rangeY, rangeB, Interval, Alpha, Label)`
- **Description**: Predicts the probability of 1 in the dependent variable using logistic regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Logit regression coefficients β.
  - `Interval`: Boolean. `TRUE` (prediction confidence interval), `FALSE` (no interval, default).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.5 ra.Logit.Forecast

- **Function**: `ra.Logit.Forecast(rangeX, rangeY, rangeB, rangeXo, Interval, Alpha, Label)`
- **Description**: Forecasts the probability of 1 in the dependent variable using logistic regression for new data.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Logit regression coefficients β.
  - `rangeXo`: New independent variables data (e.g., E1:F10).
  - `Interval`: Boolean. `TRUE` (prediction confidence interval), `FALSE` (no interval, default).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.6 ra.Logit.Confusion.Matrix

- **Function**: `ra.Logit.Confusion.Matrix(rangeX, rangeY, rangeB, cutoff, Label)`
- **Description**: Calculates the confusion matrix for logistic regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Logit regression coefficients β.
  - `cutoff`: Double. Cutoff probability for positive (1) identification (default = 0.5).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.7 ra.Logit.ROC.Table

- **Function**: `ra.Logit.ROC.Table(rangeX, rangeY, rangeB, intIntervals, Label)`
- **Description**: Estimates forecast performance for logistic regression, suitable for plotting Receiver Operating Characteristic (ROC) curves.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Logit regression coefficients β.
  - `intIntervals`: Integer. Number of intervals for cutoff probability (default = number of observations).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.8 ra.ROC.Table

- **Function**: `ra.ROC.Table(rangeX, rangeY, Iterations, Label)`
- **Description**: Estimates forecast performance for logistic regression, suitable for plotting Receiver Operating Characteristic (ROC) curves.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `Iterations`: Integer. Maximum iterations for logit coefficients and Newton-Raphson convergence (default = 20).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.9 ra.Probit.Coeff

- **Function**: `ra.Probit.Coeff(rangeX, rangeY, choiceCoeff, Alpha, Label, Iterations, OptMethod)`
- **Description**: Estimates coefficients, standard errors, z-statistics, p-values, and confidence intervals for probit regression.
- **Parameters**:

- `rangeX`: Independent variables data (e.g., A1:B100).
- `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
- `choiceCoeff`: Integer. 0 (coefficients probit, default), 1 (odds ratio).
- `Alpha`: Double. Significance level (default = 0.05).
- `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
- `Iterations`: Integer. Maximum iterations for convergence (default = 20).
- `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).

6.10.10 ra.Probit.Stat

- **Function**: `ra.Probit.Stat(rangeX, rangeY, Iterations, OptMethod, Label)`
- **Description**: Computes fit metrics like Log-Likelihood (LLH), AIC, BIC, McFadden R^2, and LR-Chi^2 test for probit regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `Iterations`: Integer. Maximum iterations for convergence (default = 20).
  - `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.11 ra.Probit.Gof

- **Function**: `ra.Probit.Gof(rangeX, rangeY, Iterations, OptMethod, Label, groupsLS)`
- **Description**: Computes goodness-of-fit tests Pearson Chi^2 or Hosmer-Lemeshow (experimental) for probit regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `Iterations`: Integer. Maximum iterations for convergence (default = 20).
  - `OptMethod`: Integer. MLE estimation method, 0 (Newton-Raphson, default).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `groupsLS`: Integer. Number of groups for Hosmer-Lemeshow test (default = auto).

6.10.12 ra.Probit.Fitted

- **Function**: `ra.Probit.Fitted(rangeX, rangeY, rangeB, Interval, Alpha, Label)`
- **Description**: Predicts the probability of 1 in the dependent variable using probit regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Probit regression coefficients β.

- `Interval`: Boolean. `TRUE` (prediction confidence interval), `FALSE` (no interval, default).
- `Alpha`: Double. Significance level (default = 0.05).
- `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.13 ra.Probit.Forecast

- **Function**: `ra.Probit.Forecast(rangeX, rangeY, rangeB, rangeXo, Interval, Alpha, Label)`
- **Description**: Forecasts the probability of 1 in the dependent variable using probit regression for new data.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Probit regression coefficients β.
  - `rangeXo`: New independent variables data (e.g., E1:F10).
  - `Interval`: Boolean. `TRUE` (prediction confidence interval), `FALSE` (no interval, default).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.14 ra.Probit.Confusion.Matrix

- **Function**: `ra.Probit.Confusion.Matrix(rangeX, rangeY, rangeB, cutoff, Label)`
- **Description**: Calculates the confusion matrix for probit regression.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Probit regression coefficients β.
  - `cutoff`: Double. Cutoff probability for positive (1) identification (default = 0.5).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.10.15 ra.Probit.ROC.Table

- **Function**: `ra.Probit.ROC.Table(rangeX, rangeY, rangeB, intIntervals, Label)`
- **Description**: Estimates forecast performance for probit regression, suitable for plotting Receiver Operating Characteristic (ROC) curves.
- **Parameters**:
  - `rangeX`: Independent variables data (e.g., A1:B100).
  - `rangeY`: Dependent variable data, acceptable values 1 or 0 (e.g., C1:C100).
  - `rangeB`: Probit regression coefficients β.
  - `intIntervals`: Integer. Number of intervals for cutoff probability (default = number of observations).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11 Principal Component Analysis (PCA)

6.11.1 ra.PCA.KMO

- **Function**: `ra.PCA.KMO(rangeX, Alpha, Label)`
- **Description**: Returns overall Kaiser-Meyer-Olkin (KMO) measure, Bartlett's chi-square test, p-value, and interpretations.
- **Parameters**:
  - `rangeX`: Columns of variables or features (e.g., A1:C100).
  - `Alpha`: Double. Significance level for Bartlett's test (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.2 ra.PCA.Eigenvalues

- **Function**: `ra.PCA.Eigenvalues(rangeX, method, Label)`
- **Description**: Calculates variances associated with principal components (axes), standardizing input variables first.
- **Parameters**:
  - `rangeX`: Data range with rows as observations and columns as variables (e.g., A1:C100).
  - `method`: Integer. 0 (QR in correlation, default), 1 (Power in correlation), 2 (QR in covariance).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.3 ra.PCA.Eigenvectors

- **Function**: `ra.PCA.Eigenvectors(rangeX, method, Label)`
- **Description**: Calculates principal components or coefficients, assuming standardization of input variables.
- **Parameters**:
  - `rangeX`: Data range with rows as observations and columns as variables (e.g., A1:C100).
  - `method`: Integer. 0 (QR in correlation, default), 1 (Power in correlation), 2 (QR in covariance).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.4 ra.PCA.Loadings

- **Function**: `ra.PCA.Loadings(rangeEVal, rangeEVec, Label, load)`
- **Description**: Calculates principal components or factor loadings, assuming standardization of input variables.
- **Parameters**:
  - `rangeEVal`: Column of eigenvalues.
  - `rangeEVec`: Matrix of eigenvectors.
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `load`: Boolean. `TRUE` (include loadings, default), `FALSE` (no loadings).

6.11.5 ra.PCA.Scores

- **Function**: `ra.PCA.Scores(rangeX, rangeLoad, Label)`
- **Description**: Calculates transformed factor scores (projections of original coordinates onto principal axes), standardizing input variables first.
- **Parameters**:
  - `rangeX`: Data range with rows as observations and columns as variables (e.g., A1:C100).
  - `rangeLoad`: Columns of PCA loadings or eigenvectors.
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.6 ra.PCA.Biplot

- **Function**: `ra.PCA.Biplot(range2Load, range2Score, biplot, rotAngleVect, scaleVecLambda, Label, rotAngleScore)`
- **Description**: Generates data for a PCA biplot with optional axis rotation.
- **Parameters**:
  - `range2Load`: Two columns of loadings for PC1 and PC2 axes or eigenvectors.
  - `range2Score`: Two columns of scores for PC1 and PC2 axes.
  - `biplot`: Boolean. `TRUE` (biplot table output, default), `FALSE` (monoplot table).
  - `rotAngleVect`: Double. Rotation angle in degrees for loadings PC1 and PC2 (default = 0.0).
  - `scaleVecLambda`: Double. Scaling factor for loadings in standard biplot (default = 1.0).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `rotAngleScore`: Double. Rotation angle in degrees for scores PC1 and PC2 (default = 0.0).

6.11.7 ra.PCA.Varimax

- **Function**: `ra.PCA.Varimax(range2Load, iter, Label, prec)`
- **Description**: Performs Varimax rotation on a matrix using Kaiser normalization.
- **Parameters**:
  - `range2Load`: Columns of factor loadings or eigenvectors.
  - `iter`: Integer. Maximum number of iterations (default = 100).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).
  - `prec`: Double. Precision for stopping condition (default = 0.00001).

6.11.8 ra.PCA.Varimax.Angle

- **Function**: `ra.PCA.Varimax.Angle(range2Load, kaiser, Label)`
- **Description**: Calculates the optimal rotation angle (in degrees) for loadings using Varimax criterion and Golden Section Search.
- **Parameters**:

- `range2Load`: Columns of loadings or eigenvectors.
- `kaiser`: Boolean. `TRUE` (use Kaiser normalization, default), `FALSE` (no normalization).
- `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.9 ra.PCA.ErrorMatrix

- **Function**: `ra.PCA.ErrorMatrix(rangeX, range1Load, reproduced, Label)`
- **Description**: Generates matrix reproduced error, for a PC scaled loadings.
- **Parameters**:
  - `rangeX`: Data range with rows as observations and columns as variables (e.g., A1:C100).
  - `range1Load`: Column of loadings for PC1 axes or eigenvectors.
  - `reproduced`: Integer. 0 (reproduced error matrix, default), 1 (reproduced matrix), 2 (correlation matrix), 3 (covariance matrix).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.10 ra.Ellipse.Stat

- **Function**: `ra.Ellipse.Stat(rangeX, rangeY, Alpha, Label)`
- **Description**: Calculates the inside points of a bivariate confidence ellipse plot.
- **Parameters**:
  - `rangeX`: Column with numeric data x (e.g., A1:A100).
  - `rangeY`: Column with numeric data y (e.g., B1:B100).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

6.11.11 ra.Ellipse.Table

- **Function**: `ra.Ellipse.Table(rangeX, rangeY, Alpha, Label)`
- **Description**: Calculates the axis-xy parameters table for a bivariate confidence ellipse plot.
- **Parameters**:
  - `rangeX`: Column with numeric data x (e.g., A1:A100).
  - `rangeY`: Column with numeric data y (e.g., B1:B100).
  - `Alpha`: Double. Significance level (default = 0.05).
  - `Label`: Boolean. `TRUE` (include label, default), `FALSE` (no label).

7 Practical Example: Integrated Analysis

- **Data**:
  - Time series in A1:A100 (prices).
  - Independent variables in B1:C100 (advertising, price).
  - Dependent variable in D1:D100 (sales).
- **Stationarity**:
  - ADF Test: `=ra.DickeyFuller.ADF.Test(A1:A100, TRUE, 1, 1, 0.05)`.
  - KPSS Test: `=ra.KPSS.Test(A1:A100, TRUE, 1, 1, 0.05)`.
- **ARIMA(1,0,1)**:
  - ADF Test: `=ra.ARIMA.Coeff(A1:A100, TRUE, 1, 0, 1, 0.05,0)`.
- **GARCH(1,1)**:
  - ADF Test: `=ra.GARCH.Coeff(A1:A100, TRUE, 1, 1, 0.05; FALSE, 0, 0)`.
- **Regression**:
  - Coefficients: `=ra.LinearReg.Coeff(B1:C100, D1:D100, TRUE, 6, 0.05)`.
  - Forecasts: `=ra.LinearReg.Forecast(B1:C100, D1:D100, E1:F10, TRUE, TRUE, 0.05)`.
  - Specification: `=ra.RamseyRESET.Test(B1:C100, D1:D100, TRUE, 2, 0.05)`.
  - Stability: `=ra.RecursiveCUSUM(B1:C100, D1:D100, TRUE, 0)`.
  - Autocorrelation: `=ra.Acorr.BreuschGodfrey.Test(B1:C100, D1:D100, TRUE, 2, 0.05, TRUE)`.
  - Multicollinearity: `=ra.MulticolLin.VIF(B1:C100, 0, 0)`.
  - Leverage: `=ra.Leverage(B1:C100, TRUE, 0)`.
  - Outliers: `=ra.Normalized.Distances(B1:C100, 0.01, 0)`.
- **Diagnostics**:
  - Residuals: `=ra.LinearReg.Residuals(B1:C100, D1:D100, TRUE, 3, 0.1587, 0.04)`.
  - Heteroskedasticity: `=ra.Het.BreuschPagan.Test(B1:C100, D1:D100, TRUE, 0.05, TRUE)`.
- **Principal Component Analysis (PCA)**:
  - Variances: `=ra.PCA.Eigenvalues(A1:C100, 0, TRUE)`.
- **Portfolio Analysis**:
  - Risk: `=ra.Portfolio.Risk(H1:J3, K1:K3)` (where H1:J3 is covariance matrix, K1:K3 is weights).
  - VaR: `=ra.VaR.Historical(L1:L100, 0.05, TRUE)` (where L1:L100 is returns).
- **Visualization**:
  - Histogram: `=ra.Histogram.Table(D1:D100, 2, FALSE)`.
  - Box Plot: `=ra.BoxPlot.Table(D1:D100, TRUE, 1.5)`.

8 Calling UDFs from VBA Macros

The user-defined functions (UDFs) of **raXL Stat** can be called from VBA macros to automate tasks or integrate with custom workflows. The following VBA example shows how to call the `ra.GARCH.Coeff` function (from the Time Series Analysis category) to estimate GARCH(1,1) coefficients for a data range.

```
Option Explicit
Option Base 1

Function Func_GARCHCoeff() As Variant
    Application.Calculation = xlCalculationManual
    Dim runResult As Variant
    Dim result() As Double
    Dim rngRange As Range
    Dim boolAscend As Boolean
    Dim intP As Integer
    Dim intQ As Integer
    Dim intMuCond As Boolean
    Dim i As Integer

    Set rngRange = ActiveSheet.Range("B2:B1001")
    boolAscend = False
    intP = 1
    intQ = 1
    intMuCond = False

    ReDim result(intP + intQ + 1, 1)
    runResult = Application.Run("ra.GARCH.Coeff", rngRange, boolAscend,
        intP, intQ, intMuCond)

    For i = 1 To intP + intQ + 1
        result(i, 1) = runResult(i, 1)
    Next i

    Func_GARCHCoeff = result
    Application.Calculation = xlCalculationAutomatic
End Function
```

Listing 1: Ejemplo de Código VBA para ra.GARCH.Coeff

Explanation:

- This VBA function calls `ra.GARCH.Coeff` to calculate GARCH(1,1) coefficients for data in the range `B2:B1001`.
- It sets calculation to manual (`xlCalculationManual`) to improve performance, then restores automatic calculation (`xlCalculationAutomatic`) after execution.
- Parameters: `rngRange` (data range), `boolAscend` (FALSE for descending order), `intP` (1 for ARCH order), `intQ` (1 for GARCH order), `intMuCond` (FALSE to exclude mean condition).
- The result is stored in a dynamic array (`result`) and returned as a column of coefficients.
- **To use**: Insert this code into a VBA module (Alt+F11, Insert ¿ Module), then call `=Func_GARCHCoeff()` in an Excel cell, selecting a range for the output (e.g., 3 rows for GARCH(1,1) coefficients).

Note: Ensure the **raXL Stat** add-in is loaded and unlocked (see Troubleshooting) before running the macro. Adjust the range and parameters as needed for your data.

9 Troubleshooting

- **Unblocking the XLL Add-in**:
  - If Excel blocks the **raXL Stat** XLL file (e.g., `raXLStat-AddIn64-packed.xll`) due to security settings, right-click the file in File Explorer, select `Properties`, and under the `General` tab, check the `Unblock` box if available, then click `OK`. Microsoft Support.
  - Alternatively, move the XLL file to a trusted location: In Excel, go to `File > Options > Trust Center > Trust Center Settings > Trusted Locations`, add the folder containing the XLL file, and restart Excel.
  - Ensure the XLL file is compatible with your Excel version (2016, 2019, 2021, or Microsoft 365, 32 or 64 bits).

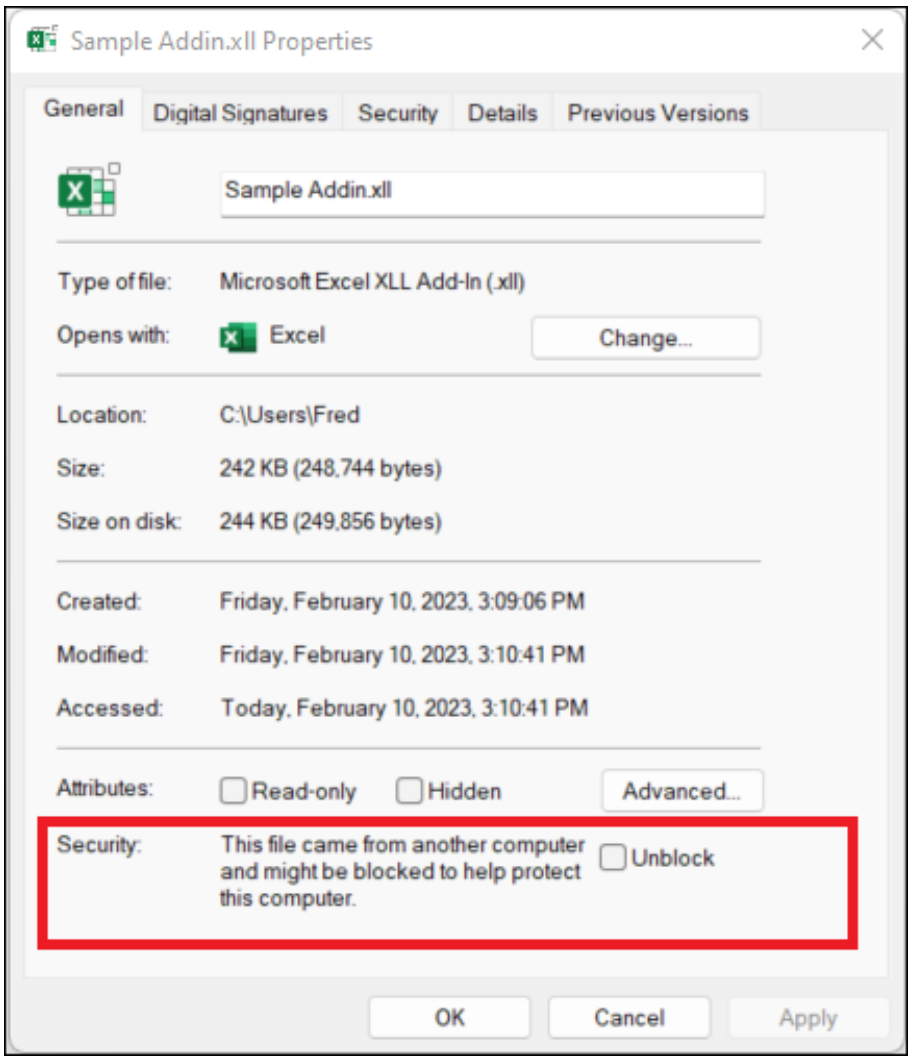


Figure 4: Unblock .xll

- **License Error**:
  - Verify the password with `=ra.raXLStat.License()`.

- **Add-in Not Appearing**:
  - Go to `File > Options > Add-ins`, ensure **raXL Stat** is checked.
  - Download the add-in from the official site: Ruben Apaza.
- **Function Errors**:
  - Verify that ranges contain numeric data with `=ra.MissingData.Info(RangeX)`.
  - Confirm that parameters (e.g., Lag, Alpha) are valid.
- **Slow Performance**:
  - Reduce the range size or use fewer iterations in functions like `ra.GARCH.Forecast`.

10 Additional Resources

- Documentation **raXL Stat** Official Site https://ruben-apaza.blogspot.com/p/raxl-stat.html – Downloads and updates.
- Microsoft Excel Support https://support.microsoft.com/en-us/excel – Help with add-ins.
- Rubén Apaza's Blog https://ruben-apaza.blogspot.com – Tutorials and examples.
- **raXL Stat** GitHub Repository https://github.com/rubenfapaza/raXL-Stat – Source code and documentation.
- Support email rubenfapaza@gmail.com Contact the **raXL Stat** team via the blog for personalized assistance.
- Video tutorial our YouTube channel https://www.youtube.com/c/rubenapaza

11 Basic Statistical Theory

This section presents the statistical foundations underlying the functions of **raXL Stat**.

11.1 Descriptive Statistics

Descriptive statistics summarize the main characteristics of a dataset, providing measures of central tendency, dispersion, shape, and confidence. These are essential for understanding the distribution and guiding subsequent analyses like linear regression.

Mean: Represents the expected central value but is sensitive to outliers. The arithmetic mean is the sum of all values divided by the number of observations:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i. \tag{1}$$

Standard Error: Indicates the variability of the mean across samples, decreasing with larger sample sizes. Measures the precision of the mean as an estimator of the population, calculated as:

$$SE = \frac{s}{\sqrt{n}}, \tag{2}$$

where s is the sample standard deviation and n is the number of observations.

Maximum: The highest value in the dataset. Identifies the upper extreme, useful for detecting outliers or establishing ranges.

Minimum: The lowest value in the dataset. Defines the lower extreme, complementing the range with the maximum.

Median (50th Percentile): The central value when data are ordered; for odd n, it is the value at position $(n+1)/2$; for even n, the average of the two central positions. Measures robust central tendency, less affected by outliers than the mean.

Skewness: Indicates the distribution shape, affecting inferences if non-zero. Measures the symmetry of the distribution:

$$S = \frac{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^3}{s^3}, \tag{3}$$

where s is the standard deviation. Values ¿ 0 indicate positive skewness (right tail), ¡ 0 indicate negative skewness (left tail).

Kurtosis:Evaluates the presence of extreme values, crucial for normality. Measures the thickness of tails and the shape of the distribution:

$$K = \frac{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^4}{s^4}. \tag{4}$$

It is adjusted by subtracting 3 (kurtosis of a normal distribution) for excess values: ¿ 0 indicates heavy tails, ¡ 0 light tails.

Standard Deviation: Quantifies variability, the square root of the variance. Measures the dispersion of data relative to the mean:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}. \tag{5}$$

Confidence Interval (95%): Provides a range of uncertainty, assuming normality. Estimates the range within which the true parameter value (e.g., mean) is expected to fall with 95% confidence:

$$CI = \bar{x} \pm t_{\alpha/2} \cdot \frac{s}{\sqrt{n}}, \tag{6}$$

where $t_{\alpha/2}$ is the critical value from the t-distribution for 95% and $n-1$ degrees of freedom.

Quartile 25%: The value below which 25% of the ordered data lie. Defines the lower bound of the lower half, useful for the interquartile range.

Quartile 75%:The value below which 75% of the ordered data lie.Defines the upper bound of the lower half, complementing the 25% quartile.

These statistics provide a comprehensive view of the data, from central tendency and dispersion to shape and confidence, forming the basis for more advanced analyses.

11.2 Simple Linear Regression

Simple linear regression is a statistical method used to model the relationship between a dependent variable (response) Y and an independent variable (predictor) X. The goal is to find a straight line that best fits the observed data, minimizing the error between the predicted and actual values.

Mathematical Model: The simple linear regression model is expressed as:

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{7}$$

In matrix notation, for n observations, the model can be written as:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{8}$$

where:

- Y: Dependent variable (response).
- X: Independent variable (predictor).
- β_0: Y-intercept (constant term).
- β_1: Slope of the line, indicating the change in Y for each unit change in X.
- ϵ: Error term, capturing variability not explained by the model. It is assumed that $\epsilon \sim N(0, \sigma^2)$.

Interpretation of Coefficients: The coefficients β_0 and β_1 have specific interpretations in the context of the model:

- β_0: Represents the expected value of Y when $X = 0$. It is the point where the regression line intersects the Y-axis. For example, if Y is sales revenue and X is advertising expenditure, β_0 indicates the baseline sales when no advertising is spent (though this interpretation may not always be meaningful if $X = 0$ is outside the data range).
- β_1: Represents the change in the expected value of Y for a one-unit increase in X. For instance, if $\beta_1 = 2$, a one-unit increase in X is associated with a 2-unit increase in Y, on average, assuming all other factors remain constant.

Estimation of Parameters: The parameters β_0 and β_1 are estimated using the least squares method, which seeks to minimize the sum of squared errors between the observed values Y_i and the predicted values $\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$. The objective is to find the values of $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the sum of squared residuals, defined as:

$$S = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2 \tag{9}$$

To find the optimal estimates, we take partial derivatives of S with respect to β_0 and β_1, set them to zero, and solve the resulting system of normal equations:

$$\frac{\partial S}{\partial \beta_0} = -2\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i) = 0 \tag{10}$$

$$\frac{\partial S}{\partial \beta_1} = -2\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i)X_i = 0 \tag{11}$$

Simplifying these equations, we obtain:

$$\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i) = 0 \implies n\beta_0 + \beta_1 \sum_{i=1}^{n} X_i = \sum_{i=1}^{n} Y_i \tag{12}$$

$$\sum_{i=1}^{n}(Y_i - \beta_0 - \beta_1 X_i)X_i = 0 \implies \beta_0 \sum_{i=1}^{n} X_i + \beta_1 \sum_{i=1}^{n} X_i^2 = \sum_{i=1}^{n} X_i Y_i \tag{13}$$

Solving this system of equations, the least squares estimates are:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^{n}(X_i - \bar{X})^2} = \frac{\mathrm{Cov}(X, Y)}{\mathrm{Var}(X)} \tag{14}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X} \tag{15}$$

where $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ and $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$ are the sample means of X and Y, respectively. The formula for $\hat{\beta}_1$ can also be expressed in terms of covariance and variance, highlighting its interpretation as the ratio of the covariance between X and Y to the variance of X. The estimate $\hat{\beta}_0$ ensures that the regression line passes through the point $(\bar{X}, \bar{Y})$, which is the centroid of the data.

The least squares estimators $\hat{\beta}_0$ and $\hat{\beta}_1$ are unbiased, meaning their expected values equal the true parameters β_0 and β_1. Additionally, under the Gauss-Markov theorem, these estimators are the Best Linear Unbiased Estimators (BLUE), provided the model assumptions (linearity, independence, homoscedasticity, and normality) hold. This means they have the smallest variance among all linear unbiased estimators.

Model Assumptions: For inferences based on simple linear regression to be valid, the following assumptions must be met:

1. **Linearity**: The relationship between X and Y is linear.
2. **Independence**: Observations are independent of each other.
3. **Homoscedasticity**: The variance of the errors ϵ is constant for all values of X.
4. **Normality**: The errors ϵ follow a normal distribution with mean zero ($N(0, \sigma^2)$).

Hypothesis Testing in Simple Linear Regression: Hypothesis tests are conducted to assess the significance of the regression model and its parameters. The most common tests are:

Test for the Slope (β_1): To determine whether the predictor X has a significant linear relationship with Y, we test:

- Null hypothesis: $H_0 : \beta_1 = 0$ (no linear relationship between X and Y).
- Alternative hypothesis: $H_a : \beta_1 \neq 0$ (there is a linear relationship).

The test statistic is:

$$t = \frac{\hat{\beta}_1}{\text{SE}(\hat{\beta}_1)} \tag{16}$$

where $\text{SE}(\hat{\beta}_1)$ is the standard error of $\hat{\beta}_1$, calculated as:

$$\text{SE}(\hat{\beta}_1) = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2/(n-2)}{\sum_{i=1}^{n}(X_i - \bar{X})^2}} \tag{17}$$

This t-statistic follows a t-distribution with $n-2$ degrees of freedom. A low p-value (typically < 0.05) leads to rejecting H_0, indicating a significant linear relationship.

Overall Model Significance (F-test): The F-test assesses whether the model explains a significant portion of the variability in Y. The hypotheses are:

- Null hypothesis: $H_0 : \beta_1 = 0$ (the model does not explain variability in Y).
- Alternative hypothesis: $H_a : \beta_1 \neq 0$ (the model explains variability).

The F-statistic is:

$$F = \frac{\text{MSR}}{\text{MSE}} \tag{18}$$

where $\text{MSR} = \frac{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2}{1}$ (mean square regression) and $\text{MSE} = \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n-2}$ (mean square error). The F-statistic follows an F-distribution with 1 and $n-2$ degrees of freedom. A low p-value indicates the model is significant.

Coefficient of Determination (R^2): The coefficient of determination R^2 measures the proportion of the total variability in Y explained by the model. It is calculated as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \tag{19}$$

An R^2 value close to 1 indicates a good fit of the model to the data.

Applications: Simple linear regression is used in various fields, such as economics, social sciences, engineering, and biology, to predict values, analyze trends, and understand relationships between variables.

11.3 Multiple Linear Regression

Multiple linear regression extends simple linear regression to model the relationship between a dependent variable Y and multiple independent variables $X_1, X_2, \ldots, X_p$. The goal is to find a linear equation that best fits the data by accounting for the combined effects of multiple predictors.

Mathematical Model: The multiple linear regression model is expressed as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \epsilon \tag{20}$$

In matrix notation, for n observations and p predictors, the model is:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{21}$$

or explicitly:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} = \begin{bmatrix} 1 & X_{11} & X_{12} & \cdots & X_{1p} \\ 1 & X_{21} & X_{22} & \cdots & X_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & X_{n1} & X_{n2} & \cdots & X_{np} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{22}$$

where:

- Y: Dependent variable (response).
- $X_{11}, X_{12}, \ldots, X_{1p}$: Independent variables (predictors).
- β_0: Y-intercept (constant term).
- $\beta_1, \beta_2, \ldots, \beta_p$: Coefficients representing the change in Y for a one-unit increase in the corresponding X_j, holding all other predictors constant.
- ϵ: Error term, assumed to follow a normal distribution, $\epsilon \sim N(0, \sigma^2)$.

Interpretation of Coefficients: The coefficients in multiple linear regression have the following interpretations:

- β_0: The expected value of Y when all predictors $X_1, X_2, \ldots, X_p$ are equal to zero. This may not always be practically meaningful, depending on the context and range of the predictors.
- β_j (for $j = 1, 2, \ldots, p$): The change in the expected value of Y for a one-unit increase in X_j, holding all other predictors constant. This partial effect isolates the contribution of X_j to Y, controlling for the effects of other variables.

Estimation of Parameters: The parameters $\beta_0, \beta_1, \ldots, \beta_p$ are estimated using the least squares method, which minimizes the sum of squared residuals:

$$S = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = \sum_{i=1}^{n} \left(Y_i - \hat{\beta}_0 - \hat{\beta}_1 X_{i1} - \cdots - \hat{\beta}_p X_{ip} \right)^2 \tag{23}$$

In matrix notation, the model is written as $\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$, where $\mathbf{Y}$ is the vector of observed responses, $\mathbf{X}$ is the design matrix (including a column of ones for the intercept), $\boldsymbol{\beta}$ is the vector of parameters, and $\boldsymbol{\epsilon}$ is the vector of errors. The least squares estimate of $\boldsymbol{\beta}$ is:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \tag{24}$$

This solution requires that $\mathbf{X}^T\mathbf{X}$ is invertible, which holds if the predictors are not perfectly collinear and there are sufficient observations ($n > p + 1$).

The least squares estimators $\hat{\boldsymbol{\beta}}$ are unbiased and, under the Gauss-Markov theorem, are the Best Linear Unbiased Estimators (BLUE) if the model assumptions hold. The variance-covariance matrix of the estimators is given by:

$$\text{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2 (\mathbf{X}^T \mathbf{X})^{-1} \tag{25}$$

where σ^2 is the variance of the error term, estimated as $\hat{\sigma}^2 = \frac{\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2}{n-p-1}$.

Model Assumptions: The assumptions for multiple linear regression are similar to those for simple linear regression. Violations of these assumptions can lead to biased estimates, inefficient inference, or incorrect conclusions. Below, each assumption is described along with common diagnostic tests and methods to check for violations:

1. **Linearity**: The relationship between Y and the predictors $X_1, X_2, \ldots, X_p$ is linear in the parameters.

   Tests and Diagnostics: - Examine residual plots: Plot standardized residuals against predicted values or each predictor. A random scatter around zero indicates linearity; patterns (e.g., curvature) suggest nonlinearity. - Partial regression plots (added variable plots): For each X_j, plot the residuals of Y regressed on all other predictors against the residuals of X_j regressed on all other predictors. A linear trend supports the assumption. - If violated, consider transformations (e.g., log, polynomial terms) or nonlinear models.

2. **Independence**: Observations are independent of each other, meaning errors are not correlated (e.g., no autocorrelation in time series data).

   Tests and Diagnostics: - Durbin-Watson test: Tests for first-order autocorrelation in residuals. The test statistic $DW = \frac{\sum_{i=2}^{n}(e_i - e_{i-1})^2}{\sum_{i=1}^{n} e_i^2}$, where e_i are residuals, ranges from 0 to 4. Values near 2 indicate no autocorrelation; < 2 positive, > 2 negative. - Plot residuals against time or order of data collection to detect patterns. - If violated (common in time series), use generalized least squares or add lagged variables.

3. **Homoscedasticity**: The variance of the errors ϵ is constant for all values of the predictors (no heteroscedasticity).

   Tests and Diagnostics: - Breusch-Pagan test: Regress squared residuals on predictors and test if coefficients are zero. The test statistic is $LM = nR^2$ from this auxiliary regression, following a $\chi^2(p)$ distribution under H_0: constant variance. - White test: Similar but includes interactions and squares of predictors in the auxiliary regression. - Plot residuals vs. fitted values: Fan-shaped or patterned spread indicates **heteroscedasticity**. - If violated, use weighted least squares, robust standard errors, or transformations (e.g., log Y). - ARCH Test: Designed for time series, regresses squared residuals (e_t^2) against their own lags ($e_{t-1}^2, e_{t-2}^2, \ldots, e_{t-p}^2$):

$$e_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i e_{t-i}^2 + \eta_t, \tag{26}$$

   where p is the number of lags. The test statistic is TR^2 (where T is the number of observations), which follows a χ^2 distribution with p degrees of freedom under H_0 (no ARCH effects). Detects conditional heteroscedasticity, common in financial data with variable volatility. A $p < 0.05$ rejects H_0, indicating ARCH effects.

4. **Normality**: The errors ϵ follow a normal distribution with mean zero ($N(0, \sigma^2)$). This is crucial for hypothesis tests and confidence intervals.

   Tests and Diagnostics: - Quantile-Quantile (Q-Q) plot: Plot ordered residuals against theoretical normal quantiles. Deviation from a straight line indicates non-normality. - Shapiro-Wilk test: Tests H_0: residuals are normally distributed. The statistic W is close to 1 for normal data; low p-value rejects normality. - Jarque-Bera test: Based on skewness and kurtosis, $JB = \frac{n}{6}\left(S^2 + \frac{(K-3)^2}{4}\right)$, where S is skewness and K is kurtosis, following $\chi^2(2)$ under H_0. - If violated, consider transformations (e.g., Box-Cox) or robust methods; for large n, central limit theorem may mitigate issues.

5. **No perfect multicollinearity**: The predictors $X_1, X_2, \ldots, X_p$ are not perfectly linearly dependent, ensuring $\mathbf{X}^T\mathbf{X}$ is invertible.

   Tests and Diagnostics: - Variance Inflation Factor (VIF): For each X_j, $VIF_j = \frac{1}{1-R_j^2}$, where R_j^2 is from regressing X_j on other predictors. VIF > 5 or 10 suggests high multicollinearity. - Condition number: The ratio of largest to smallest eigenvalue of $\mathbf{X}^T\mathbf{X}$; values > 30 indicate multicollinearity. - Correlation matrix: High correlations ($|r| > 0.8$) between predictors signal issues. - If violated, remove redundant variables, use principal component analysis, or ridge regression. - **Pratt's Influence**: John W. Pratt (1987) proposed an approach to decompose the contribution of each independent variable to the total R^2 in the presence of multicollinearity. Pratt's influence is measured by the product of the standardized coefficient β_j^* and the simple correlation r_{yj} between the variable X_j and the dependent variable Y, adjusted for the explained variance: Pratt's Influence$_j = \beta_j^* \cdot r_{yj}$.

This identifies which variables have a significant relative impact, even with high correlations among predictors. See Pratt, J. W. (1987), "Dividing the Indivisible: Using Simple Symmetry to Partition Variance Explained," in *Proceedings of the Second International Tampere Conference in Statistics*, for more details.

6. **Model Specification** - Assumption: The model includes relevant variables and the correct functional form (linearity). - Importance: A misspecified model (omitted variables or incorrect functional form) produces biased coefficients. - Test: The Ramsey RESET test checks specification by regressing residuals against powers of predictions. A $p < 0.05$ suggests the model needs adjustments. Checks for omitted variables or incorrect functional form (e.g., nonlinearity) by regressing residuals against powers of predictions ($\hat{Y}^2, \hat{Y}^3, \ldots$). The test statistic is based on:

$$F = \frac{(SSR_r - SSR_u)/q}{SSR_u/(n-k-q)}, \tag{27}$$

where SSR_r is the sum of squared residuals from the restricted model (without powers), SSR_u is the sum of residuals from the unrestricted model (with powers), q is the number of powers, n is the number of observations, and k is the number of parameters in the original model. Alternatively, $n \cdot R^2$ from the auxiliary model is used, following a χ^2 distribution with q degrees of freedom. Detects misspecifications, such as omitted variables or unmodeled nonlinear relationships. A $p < 0.05$ rejects H_0 (model is well-specified), suggesting the model needs adjustments (e.g., including nonlinear terms or additional variables).

7. **Parameter Stability** - CUSUM Test: The cumulative sum (CUSUM) test, proposed by Brown, Durbin, and Evans (1975), evaluates the stability of regression coefficients over time. It is calculated as the cumulative sum of standardized residuals:$S_t = \sum_{i=1}^{t} \frac{e_i}{\hat{\sigma}}$, where e_i are the residuals and $\hat{\sigma}$ is the estimated standard deviation of residuals.

Hypothesis Testing in Multiple Linear Regression: Hypothesis tests evaluate the significance of individual coefficients and the overall model:

- **Test for Individual Coefficients**: For each β_j, we test $H_0 : \beta_j = 0$ versus $H_a : \beta_j \neq 0$ using the t-statistic:

$$t = \frac{\hat{\beta}_j}{\text{SE}(\hat{\beta}_j)} \tag{28}$$

where $\text{SE}(\hat{\beta}_j)$ is the standard error of $\hat{\beta}_j$, derived from the diagonal of the variance-covariance matrix. The test follows a t-distribution with $n - p - 1$ degrees of freedom.

- **Overall Model Significance (F-test)**: Tests $H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0$ versus H_a: at least one $\beta_j \neq 0$. The F-statistic is:

$$F = \frac{\text{MSR}}{\text{MSE}} = \frac{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2/p}{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2/(n-p-1)} \tag{29}$$

This follows an F-distribution with p and $n - p - 1$ degrees of freedom.

Coefficient of Determination (R^2 and Adjusted R^2): The coefficient of determination R^2 is calculated as in simple linear regression:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2} \tag{30}$$

However, R^2 always increases with additional predictors. The adjusted R^2 accounts for the number of predictors:

$$R^2_{\text{adj}} = 1 - \left(\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2/(n-p-1)}{\sum_{i=1}^{n}(Y_i - \bar{Y})^2/(n-1)} \right) \tag{31}$$

R^2_{adj} penalizes excessive predictors, providing a better measure of model fit.

Applications: Multiple linear regression is widely used in fields like economics, marketing, and social sciences to model complex relationships, predict outcomes, and control for confounding variables.

11.4 Normality Tests

Normality tests assess whether a sample of data or residuals follows a normal distribution, a key assumption in many statistical analyses, such as linear regression and hypothesis testing. These tests are particularly important for validating confidence intervals and p-values.

Jarque-Bera Tes: Combines measures of skewness (S) and kurtosis (K) to evaluate normality. The statistic is calculated as:

$$JB = \frac{n}{6}\left(S^2 + \frac{(K-3)^2}{4}\right), \tag{32}$$

where n is the number of observations, S is skewness (expected to be 0 in a normal distribution), and K is kurtosis (expected to be 3 in a normal distribution). Under the null hypothesis (H_0: data are normal), JB approximately follows a χ^2 distribution with 2 degrees of freedom.Effective for large samples but can be sensitive to deviations in the tails. Interpretation, a $p < 0.05$ rejects H_0, indicating non-normality.

Shapiro-Wilk Test:Measures the correlation between ordered residuals and expected quantiles of a normal distribution. The statistic W is defined as:

$$SW = W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \tag{33}$$

where $x_{(i)}$ are the ordered residuals, a_i are coefficients based on normal quantiles, and $\bar{x}$ is the mean. W ranges from 0 to 1, with 1 indicating a perfectly normal distribution.Highly powerful, especially for small samples (5 to 50 observations), and sensitive to all forms of deviation from normality. Interpretation, a $p < 0.05$ rejects H_0, indicating non-normality.

Anderson-Darling Test:Compares the empirical distribution of data to a theoretical normal distribution, giving more weight to the tails. The statistic A^2 is calculated as:

$$AD = A^2 = -n - \frac{1}{n}\sum_{i=1}^{n}(2i-1)\left[\ln F(x_i) + \ln(1 - F(x_{n+1-i}))\right], \tag{34}$$

where $F(x_i)$ is the cumulative distribution function of a normal fitted to the data, and n is the number of observations. It is compared to specific critical values. More sensitive to deviations in the tails than other tests, ideal for detecting distributions with heavy or light tails. Interpretation, a $p < 0.05$ rejects H_0, suggesting non-normality.

These tests are complementary: Jarque-Bera is useful for large samples, Shapiro-Wilk for small samples, and Anderson-Darling for analyzing tails. If any test rejects normality, transformations (e.g., logarithmic) or non-parametric models may be considered.

11.5 Binary Regression: Logit and Probit

Binary logit (logistic) and probit regression are statistical methods used to model the relationship between a binary dependent variable Y (taking values 0 or 1) and one or more independent variables $X_1, X_2, \ldots, X_p$. These models are used when the outcome is dichotomous, such as success/failure or presence/absence, and aim to predict the probability of the event $Y = 1$.

Mathematical Models Both logistic and probit regression model the probability $P(Y = 1|\mathbf{X})$ as a function of the predictors. The linear predictor is defined as:

$$\eta = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p \tag{35}$$

where $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ are the predictors, β_0 is the intercept, and $\beta_1, \ldots, \beta_p$ are the coefficients.

Logit Regression In logit regression, the probability is modeled using the logistic function:

$$P(Y = 1|\mathbf{X}) = \frac{1}{1 + e^{-\eta}} = \frac{e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p}} \tag{36}$$

The logit, or log-odds, is the linear predictor:

$$\text{logit}(P(Y = 1|\mathbf{X})) = \ln\left(\frac{P(Y = 1|\mathbf{X})}{1 - P(Y = 1|\mathbf{X})}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p \tag{37}$$

Probit Regression In probit regression, the probability is modeled using the cumulative distribution function (CDF) of the standard normal distribution, Φ:

$$P(Y = 1|\mathbf{X}) = \Phi(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p) \tag{38}$$

where $\Phi(z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt$. The probit model assumes the latent variable $Y^* = \eta + \epsilon$, where $\epsilon \sim N(0, 1)$, and $Y = 1$ if $Y^* > 0$, else $Y = 0$.

Interpretation of Coefficients The interpretation of coefficients in logit and probit regression differs from linear regression due to the nonlinear link functions. The coefficients reflect changes in the log-odds (logit) or z-score (probit), but marginal effects provide a more intuitive interpretation by quantifying the change in the probability $P(Y = 1|\mathbf{X})$ for a unit change in a predictor.

- **Logit Regression**:
  - The coefficient β_j represents the change in the log-odds of $Y = 1$ for a one-unit increase in X_j, holding other predictors constant. The **odds ratio** is e^{β_j}, indicating the multiplicative change in odds for a one-unit increase in X_j. For example, if $\beta_j = 0.693$, the odds increase by a factor of $e^{0.693} \approx 2$ per unit increase in X_j.
  - **Marginal Effects**: The marginal effect of X_j on $P(Y = 1|\mathbf{X})$ is the partial derivative of the probability with respect to X_j:

$$\frac{\partial P(Y = 1|\mathbf{X})}{\partial X_j} = \beta_j \cdot P(Y = 1|\mathbf{X}) \cdot (1 - P(Y = 1|\mathbf{X})) \tag{39}$$

    This effect depends on the values of all predictors, as $P(Y = 1|\mathbf{X})$ varies. The marginal effect is often evaluated at the mean of the predictors (average marginal effect) or for specific values of $\mathbf{X}$. For example, if $\beta_j = 0.5$ and $P(Y = 1|\mathbf{X}) = 0.5$ at the mean, the marginal effect is $0.5 \cdot 0.5 \cdot (1 - 0.5) = 0.125$, meaning a one-unit increase in X_j increases the probability by 0.125.

- **Probit Regression**:

- The coefficient β_j represents the change in the z-score of the latent variable Y^* for a one-unit increase in X_j, holding other predictors constant. The effect on probability depends on the normal CDF and varies with $\mathbf{X}$.
- **Marginal Effects**: The marginal effect of X_j is:

$$\frac{\partial P(Y=1|\mathbf{X})}{\partial X_j} = \beta_j \cdot \phi(\eta) \tag{40}$$

where $\phi(\eta) = \frac{1}{\sqrt{2\pi}} e^{-\eta^2/2}$ is the standard normal density at $\eta = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$. Like logit regression, this effect varies with $\mathbf{X}$ and is often computed at the mean or specific values. For instance, if $\beta_j = 0.3$ and $\phi(\eta) = 0.4$ at the mean, the marginal effect is $0.3 \cdot 0.4 = 0.12$, indicating a 0.12 increase in probability per unit increase in X_j.

Marginal effects are particularly useful for comparing the practical impact of predictors across models or datasets, as they directly measure changes in probability rather than log-odds or z-scores. For categorical predictors, marginal effects are often computed as the difference in predicted probabilities when the predictor changes from one category to another.

Estimation of Parameters Unlike linear regression, logit and probit models are estimated using maximum likelihood estimation (MLE) because the binary outcome violates the assumptions of least squares. The likelihood function for n observations is:

$$L(\boldsymbol{\beta}) = \prod_{i=1}^{n} [P(Y_i = 1|\mathbf{X}_i)]^{Y_i} [1 - P(Y_i = 1|\mathbf{X}_i)]^{1-Y_i} \tag{41}$$

The log-likelihood is maximized:

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} [Y_i \ln(P(Y_i = 1|\mathbf{X}_i)) + (1 - Y_i) \ln(1 - P(Y_i = 1|\mathbf{X}_i))] \tag{42}$$

For logit regression, $P(Y_i = 1|\mathbf{X}_i) = \frac{1}{1+e^{-\eta_i}}$, and for probit, $P(Y_i = 1|\mathbf{X}_i) = \Phi(\eta_i)$. The MLE estimates $\hat{\boldsymbol{\beta}}$ are obtained by solving:

$$\frac{\partial \ell}{\partial \boldsymbol{\beta}} = 0 \tag{43}$$

This is typically done numerically using iterative methods like Newton-Raphson or gradient descent, as no closed-form solution exists.

Model Assumptions The assumptions for logit and probit regression differ from linear regression due to the binary outcome and nonlinear link functions:

1. **Correct link function**: The chosen link (logit or probit) appropriately models the relationship between $\mathbf{X}$ and $P(Y = 1)$.
2. **Independence**: Observations are independent.
3. **No perfect separation**: No linear combination of predictors perfectly separates $Y = 1$ from $Y = 0$, as this prevents MLE convergence.
4. **Linearity in the link**: The linear predictor $\eta = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$ correctly specifies the relationship with the log-odds (logit) or z-score (probit).
5. **Adequate sample size**: Sufficient events per predictor to ensure stable estimates (rule of thumb: at least 10 events per parameter).

Diagnostics include: - Hosmer-Lemeshow test (logit): Groups predicted probabilities and tests goodness-of-fit via χ^2. - Residual analysis (e.g., deviance residuals) to check for outliers or model misspecification. - Check for separation by examining convergence issues or extreme coefficient estimates.

Hypothesis Testing

- **Wald Test for Individual Coefficients**: Tests $H_0 : \beta_j = 0$ vs. $H_a : \beta_j \neq 0$ using:

$$z = \frac{\hat{\beta}_j}{\text{SE}(\hat{\beta}_j)} \tag{44}$$

  where $\text{SE}(\hat{\beta}_j)$ is derived from the inverse of the Fisher information matrix. The test follows a standard normal distribution.

- **Likelihood Ratio Test**: Tests nested models (e.g., all $\beta_j = 0$ vs. at least one $\beta_j \neq 0$) using:

$$LR = -2(\ell_{\text{reduced}} - \ell_{\text{full}}) \tag{45}$$

  which follows a χ^2 distribution with degrees of freedom equal to the difference in parameters.

Model Fit and Evaluation Instead of R^2, logit and probit models use pseudo-R^2 measures (e.g., McFadden's R^2) or classification metrics like accuracy, sensitivity, specificity, and the area under the ROC curve (AUC) to evaluate fit and predictive performance.

Applications Logit (logistic) and probit regression are used in fields like medicine (e.g., predicting disease presence), economics (e.g., credit default), and social sciences (e.g., voting behavior) to model binary outcomes and estimate probabilities.

11.6 Theory of Principal Component Analysis

Principal Component Analysis (PCA) is a multivariate statistical technique used to reduce the dimensionality of a dataset while preserving as much variability as possible. It transforms a set of correlated variables $X_1, X_2, \ldots, X_p$ into a smaller set of uncorrelated variables called principal components, which are linear combinations of the original variables. PCA is commonly used for data compression, visualization, and addressing multicollinearity in regression models.

Mathematical Formulation Given a dataset with n observations and p variables, represented as an $n \times p$ data matrix $\mathbf{X}$, PCA seeks to find principal components $\mathbf{Z}_1, \mathbf{Z}_2, \ldots, \mathbf{Z}_k$ (where $k \leq p$) such that:

$$\mathbf{Z}_j = \mathbf{X}\mathbf{a}_j \tag{46}$$

where $\mathbf{a}_j$ is a $p \times 1$ vector of loadings (weights) for the j-th principal component, and $\mathbf{Z}_j$ is the $n \times 1$ vector of scores for the j-th component. The loadings are chosen to maximize the variance of $\mathbf{Z}_j$, subject to the constraints:

- $\mathbf{a}_j^T \mathbf{a}_j = 1$ (unit length to normalize the components).
- $\mathbf{a}_j^T \mathbf{a}_k = 0$ for $j \neq k$ (orthogonality to ensure uncorrelated components).

The variance of the j-th principal component is given by the eigenvalue λ_j of the covariance matrix $\mathbf{S}$ (or correlation matrix if variables are standardized). The first principal component $\mathbf{Z}_1$ explains the largest proportion of variance, the second $\mathbf{Z}_2$ the next largest, and so forth.

Computation of Principal Components PCA is typically performed using the following steps:

1. **Standardize the Data**: If variables have different scales, standardize $\mathbf{X}$ to have mean 0 and variance 1 for each variable to ensure equal contribution to the variance.

2. **Calculate the Covariance Matrix**: Compute the $p \times p$ sample covariance matrix $\mathbf{S} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}$ (if centered) or use the correlation matrix for standardized data.

3. **Eigenvalue Decomposition**: Find the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$ and corresponding eigenvectors $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_p$ of $\mathbf{S}$. The eigenvector $\mathbf{a}_j$ defines the direction of the j-th principal component, and λ_j represents its variance.

   The eigenvalues and eigenvectors satisfy the equation:

$$\mathbf{S}\mathbf{a}_j = \lambda_j \mathbf{a}_j \tag{47}$$

   Estimation of eigenvalues and eigenvectors is typically done using numerical methods such as the QR algorithm, Jacobi method, or singular value decomposition (SVD) for large matrices. The eigenvalues are sorted in descending order to prioritize components with the highest variance.

4. **Select Components**: Choose the first k components that explain a desired proportion of total variance (e.g., 80–90%). The proportion of variance explained by the j-th component is:

$$\text{Proportion} = \frac{\lambda_j}{\sum_{i=1}^{p} \lambda_i} \tag{48}$$

5. **Transform Data**: Compute the principal component scores $\mathbf{Z} = \mathbf{X}\mathbf{A}$, where $\mathbf{A}$ is the $p \times k$ matrix of the first k eigenvectors.

Numerical Example Consider a small dataset with 3 observations and 2 variables:

$$\mathbf{X} = \begin{pmatrix} 2.5 & 2.4 \\ 0.5 & 0.7 \\ 2.2 & 2.9 \end{pmatrix} \tag{49}$$

After centering the data (subtracting the mean of each column), the covariance matrix is:

$$\mathbf{S} = \begin{pmatrix} 1.163 & 1.165 \\ 1.165 & 1.33 \end{pmatrix} \tag{50}$$

The eigenvalues are approximately $\lambda_1 = 2.415$ and $\lambda_2 = 0.079$, and the corresponding eigenvectors are:

$$\mathbf{a}_1 = \begin{pmatrix} -0.681 \\ -0.732 \end{pmatrix}, \quad \mathbf{a}_2 = \begin{pmatrix} -0.732 \\ 0.681 \end{pmatrix} \tag{51}$$

The first principal component explains $\frac{2.415}{2.415+0.079} \approx 96.8\%$ of the total variance, indicating that most of the data's variability can be captured in one dimension.

Interpretation of Principal Components Each principal component is a linear combination of the original variables:

$$Z_j = a_{j1}X_1 + a_{j2}X_2 + \cdots + a_{jp}X_p \tag{52}$$

The loadings $a_{j1}, a_{j2}, \ldots, a_{jp}$ indicate the contribution of each variable to the component. Large absolute values of a_{jk} suggest that variable X_k strongly influences the j-th component. Components are often interpreted by examining the loadings to identify patterns or groupings of variables (e.g., in finance, the first component might capture overall market trends if variables are stock returns).

Assumptions and Considerations PCA does not rely on distributional assumptions but requires the following considerations:

1. **Linearity**: PCA assumes that the relationships among variables are linear, as it is based on variance and covariance.

2. **Variance as Importance**: PCA assumes that variables with higher variance are more important, which may not always align with the research context.

3. **Standardization**: If variables are on different scales, standardization is necessary to avoid dominance by high-variance variables.

4. **Outlier Sensitivity**: PCA is sensitive to outliers, which can distort the covariance matrix and principal components.

Diagnostics include: - Scree plot: Plot eigenvalues against component number to identify the "elbow" where additional components explain little variance. - Cumulative variance explained: Assess how many components are needed to capture a sufficient proportion of variance. - Check for outliers using robust PCA or by examining scores for extreme values.

PCA in Regression PCA is often used in regression to address multicollinearity or reduce dimensionality:

- **Principal Component Regression (PCR)**: Instead of regressing Y on the original predictors $\mathbf{X}$, regress Y on the first k principal components $\mathbf{Z}_1, \ldots, \mathbf{Z}_k$. This reduces multicollinearity since components are orthogonal.

- **Variable Selection**: PCA can identify which variables contribute most to variance, aiding in feature selection.

Applications PCA is widely used in fields like finance (e.g., portfolio analysis), genomics (e.g., gene expression analysis), image processing (e.g., data compression), and social sciences (e.g., survey data reduction). It is particularly valuable for visualizing high-dimensional data in two or three dimensions and addressing multicollinearity in regression models.

11.7 Autocorrelation and Time Series

Autocorrelation measures the linear dependence between Y_t and Y_{t-k}. The autocorrelation function (ACF) is:

$$\rho_k = \frac{\text{Cov}(Y_t, Y_{t-k})}{\sqrt{\text{Var}(Y_t)\text{Var}(Y_{t-k})}}. \tag{53}$$

The partial autocorrelation function (PACF) removes intermediate effects. The Ljung-Box test evaluates significance:

$$Q = n(n+2)\sum_{k=1}^{h} \frac{\rho_k^2}{n-k}. \tag{54}$$

The test hypotheses are:

- H_0: The residuals show no autocorrelation (lags up to h are independent).

- H_1: At least one lag up to h shows significant autocorrelation.

A $p < 0.05$ rejects H_0, indicating autocorrelation.

The Box-Pierce test, an alternative to the Ljung-Box test, also evaluates autocorrelation. Its statistic is:

$$Q = n\sum_{k=1}^{h} \rho_k^2, \tag{55}$$

where n is the number of observations and h is the number of lags. Under H_0 (no autocorrelation), Q approximately follows a χ^2 distribution with h degrees of freedom. It is less powerful than Ljung-Box for small samples but remains useful for time series.

11.8 Unit Root Tests

A stationary series has a constant mean and variance. The Augmented Dickey-Fuller (ADF) test evaluates:

$$\Delta Y_t = \alpha + \beta t + \gamma Y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta Y_{t-i} + \epsilon_t. \tag{56}$$

The test hypotheses are:

- H_0: The series has a unit root (non-stationary).
- H_1: The series has no unit root (stationary).

The coefficient γ is key: if $\gamma = 0$, H_0 is not rejected, while if $\gamma < 0$ and statistically significant (with a p-value ¡ 0.05), H_0 is rejected in favor of H_1. The KPSS test evaluates stationarity under the null hypothesis that the series is stationary around a deterministic trend. The KPSS statistic is calculated as:

$$\text{KPSS} = \frac{1}{T^2} \sum_{t=1}^{T} S_t^2 / s^2(l),$$

where S_t is the cumulative sum of residuals, $s^2(l)$ is a robust estimate of the long-run variance, and T is the number of observations. The test hypotheses are:

- H_0: The series is stationary around a deterministic trend or level.
- H_1: The series is non-stationary (has a unit root).

A $p < 0.05$ rejects H_0, indicating non-stationarity.

The Augmented Dickey-Fuller (ADF) regression estimates the model parameters, including the coefficient γ, which indicates the presence of a unit root. The regression form of the model is useful for diagnosing and correcting non-stationarity, adjusting lag terms to eliminate autocorrelation in the errors. Similarly, the KPSS regression estimates parameters under the null hypothesis of stationarity, adjusting for the deterministic trend and evaluating the contribution of stochastic components.

11.9 ARIMA Models

ARIMA(p,d,q) models time series after applying differencing of order d to achieve stationarity. The reduced form of the model is expressed as:

$$Y_t = c + \sum_{i=1}^{p} \phi_i Y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t, \tag{57}$$

where:

- c is a constant (may include drift if $d > 0$),
- $\sum_{i=1}^{p} \phi_i Y_{t-i}$ represents the autoregressive terms of order 1 to p,
- $\sum_{j=1}^{q} \theta_j \epsilon_{t-j}$ represents the moving average terms of order 1 to q,
- ϵ_t is the white noise error term with zero mean and constant variance.

The forecast for h-steps ahead ($\hat{Y}_{t+h|t}$) is calculated recursively as:

$$\hat{Y}_{t+h|t} = c + \sum_{i=1}^{p} \phi_i \hat{Y}_{t+h-i|t} + \sum_{j=1}^{q} \theta_j \epsilon_{t+h-j}, \tag{58}$$

where:

- $\hat{Y}_{t+h|t}$ is the forecast of Y_{t+h} given information up to t,
- $\hat{Y}_{t+h-i|t}$ uses observed values of Y_{t+h-i} for $h-i \leq 0$ and prior forecasts for $h-i > 0$,
- ϵ_{t+h-j} uses observed residuals for $h-j \leq 0$ and is assumed zero for $h-j > 0$.

For long horizons ($h \to \infty$), the forecast converges to the series mean (or c if there is no drift). AIC and BIC optimize the selection of p, d, and q.

11.10 GARCH Models

GARCH(p,q) models the conditional volatility of a time series. The model equation is:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \tag{59}$$

where:

- ω is the constant,
- $\sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2$ represents the ARCH effects (past shocks),
- $\sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$ represents the persistence of volatility,
- ϵ_t is the error, and σ_t^2 is the conditional variance at time t.

The volatility forecast for h-steps ahead (σ_{t+h}^2) is calculated as:

$$\sigma_{t+h}^2 = \omega + \sum_{i=1}^{p} \alpha_i E[\epsilon_{t+h-i}^2] + \sum_{j=1}^{q} \beta_j \sigma_{t+h-j}^2, \tag{60}$$

where for $h > q$, $E[\epsilon_{t+h-i}^2]$ converges to the unconditional variance $\sigma^2 = \frac{\omega}{1-\sum_{i=1}^{p} \alpha_i - \sum_{j=1}^{q} \beta_j}$, assuming $\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1$ for stationarity. The ARCH test detects volatility effects.

11.11 Geometric Brownian Motion (GBM)

Geometric Brownian Motion (GBM) is a stochastic model widely used in finance to describe the evolution of asset prices, such as stocks or indices. It is based on a continuous stochastic process that assumes logarithmic returns follow a Brownian motion with drift.

GBM Equation: The stochastic differential equation defining GBM is:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{61}$$

where:

- S_t is the asset price at time t,
- μ is the drift rate (expected return),
- σ is the volatility (standard deviation of returns),
- dW_t is an increment of a standard Brownian motion (Gaussian noise with mean 0 and variance dt).

The analytical solution for S_t given an initial price S_0 is:

$$S_t = S_0 \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)t + \sigma W_t\right), \tag{62}$$

where $W_t \sim N(0,t)$ is the cumulative Brownian motion.

Properties: GBM implies that logarithmic returns $\ln(S_t/S_{t-1})$ are normally distributed with mean $\mu - \sigma^2/2$ and variance $\sigma^2\Delta t$.

- The price S_t follows a log-normal distribution, ensuring prices remain positive.
- It is widely used in option pricing (Black-Scholes model) and Monte Carlo simulations.

Forecasting: To simulate future trajectories, GBM is discretized using the Euler-Maruyama method:

$$S_{t+\Delta t} = S_t \exp\left(\left(\mu - \frac{\sigma^2}{2}\right)\Delta t + \sigma\sqrt{\Delta t}Z\right), \tag{63}$$

where $Z \sim N(0,1)$ is a standard normal random variable, and Δt is the time step.

11.12 Portfolio Analysis

Portfolio theory, developed by Harry Markowitz, seeks to optimize the trade-off between risk and expected return of a portfolio of assets. Portfolio risk is measured by its total volatility, expressed as:

$$\sigma_p = \sqrt{\mathbf{w}^T\Sigma\mathbf{w}}, \tag{64}$$

where $\mathbf{w}$ is the vector of asset weights (summing to 1), and Σ is the covariance matrix of asset returns, incorporating both individual variances and diversification effects through covariances. The expected portfolio return is calculated as:

$$\mu_p = \mathbf{w}^T\mathbf{r}, \tag{65}$$

where $\mathbf{r}$ is the vector of expected asset returns. The main goal is to determine the weights $\mathbf{w}$ that minimize σ_p for a given level of μ_p, generating the efficient frontier, a set of optimal portfolios offering the lowest risk for a given return. Diversification reduces unsystematic risk (specific to assets), although systematic risk (market-related) persists. Value at Risk (VaR) measures the maximum expected loss over a time horizon at a given confidence level. The hypotheses associated with VaR tests are:

Historical VaR:

- H_0: Losses do not exceed the VaR at the specified confidence level (e.g., 95%).
- H_1: Losses exceed the VaR with a frequency higher than expected.
- This methodology relies on the empirical distribution of historical returns, assuming past conditions are representative of future behavior.

Parametric VaR:

- H_0: Returns follow the assumed distribution (typically normal) and losses do not exceed the VaR at the confidence level.
- H_1: Returns do not follow the assumed distribution or losses exceed the VaR more frequently.

This approach relies on parameter estimates such as mean and volatility, typically assuming a parametric distribution like the normal.

Backtesting and Kupiec Test:

- Backtesting, a procedure to evaluate the accuracy of a VaR model by comparing predicted losses with actual observed losses over a test period. It involves counting the number of exceptions (days when losses exceed the estimated VaR) and comparing them to the expected confidence level (e.g., 5% for a 95% VaR).
- Kupiec Test, evaluates whether the number of exceptions is consistent with the VaR confidence level. The test statistic is based on a likelihood ratio and follows a χ^2 distribution with 1 degree of freedom. The formula is:

$$LR_{uc} = -2 \ln \left(\frac{(1-p)^{T-x} p^x}{(1-\hat{p})^{T-x} \hat{p}^x} \right), \tag{66}$$

  where:

  - p is the VaR confidence level (e.g., 0.05),
  - T is the total number of observations,
  - x is the number of observed exceptions,
  - $\hat{p} = x/T$ is the observed proportion of exceptions.

  The hypotheses are:

- H_0: The VaR model is adequate (the proportion of exceptions $\hat{p}$ does not significantly differ from p).
- H_1: The VaR model is inadequate (the proportion of exceptions differs from p).

A $p < 0.05$ rejects H_0, indicating that the model is unreliable. See Kupiec, P. H. (1995), "Techniques for Verifying the Accuracy of Risk Measurement Models," in *Journal of Derivatives*, for more details.

References

[1] Box, G. E. P., & Jenkins, G. M. (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day.

[2] Brockwell, P. J., & Davis, R. A. (2002). *Introduction to Time Series and Forecasting*. Springer.

[3] Anderson, T. W. (2003). *An Introduction to Multivariate Statistical Analysis*. Wiley.

[4] Engle, R. F. (1982). Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4), 987-1007.

[5] Markowitz, H. (1952). Portfolio Selection. *The Journal of Finance*, 7(1), 77-91.

[6] Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3), 637-654.

[7] Brown, R. L., Durbin, J., & Evans, J. M. (1975). Techniques for Testing the Constancy of Regression Relationships Over Time. *Journal of the Royal Statistical Society, Series B*, 37(2), 149-192.

[8] Pratt, J. W. (1987). Dividing the Indivisible: Using Simple Symmetry to Partition Variance Explained. *Proceedings of the Second International Tampere Conference in Statistics*.

[9] Kupiec, P. H. (1995). Techniques for Verifying the Accuracy of Risk Measurement Models. *Journal of Derivatives*, 3(2), 73-84.

[10] Fisher, R. A. (1921). On the 'Probable Error' of a Coefficient of Correlation Deduced from a Small Sample. *Metron*, 1, 3-32.

[11] Greene, W. H. (2012). *Econometric Analysis* (7th ed.). Pearson.

[12] Wooldridge, J. M. (2019). *Introductory Econometrics: A Modern Approach* (7th ed.). Cengage Learning.

[13] Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2010). *Multivariate Data Analysis* (7th ed.). Prentice Hall.

[14] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to Linear Regression Analysis* (5th ed.). Wiley.

[15] Tsay, R. S. (2005). *Analysis of Financial Time Series* (2nd ed.). Wiley.

[16] Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). *Data Mining: Practical Machine Learning Tools and Techniques* (4th ed.). Morgan Kaufmann.