Learn how to design simulation and optimization models! In this sample chapter, you’ll walk through an in-depth case study using the Monte Carlo simulation to calculate value-at-risk and advanced operations for optimizing an investment portfolio.

“Simulations and Optimizations”

Contents

Index

The Authors

Anil Babu Ankisettipalli, Hansen Chen, Pranav Wankawala

SAP HANA Advanced Data Modeling

392 Pages, 2016, $79.95/€79.95
ISBN 978-1-4932-1236-1

www.sap-press.com/3863
In this chapter, we will discuss how to design and model optimizations and simulations on an SAP HANA platform.

6 Simulations and Optimizations

This chapter explains how to use SAP HANA to design simulation and optimization models through an in-depth case study. While considering an equally distributed portfolio with \( n \) symbols/stock securities, we will simulate value-at-risk with a Monte Carlo simulation using the Geometric Brownian Motion (GBM) model. This chapter also covers complex operations such as matrix inversion, matrix algebra, and the creation of mean-covariance matrices with stock securities on an SAP HANA platform for optimizing an investment portfolio.

Let's begin by reviewing the parameters of the case study that will serve as a running example throughout this chapter.

6.1 Case Study

Many countries have stock markets, such as NASDAQ, S&P, and so on. Companies listed in these markets allow the public to invest in their stocks. These stocks are traded every working day of the stock market, and investment management companies create portfolios to manage the investments in these stocks in order to produce better returns on clients' investments. The following terms and definitions will be found throughout this chapter and case study:

- **Security**
  - The financial instrument of a listed company in a stock market. The terms *symbol* and *stock* are used similarly in this case study.

- **Portfolio**
  - A range of securities in which an organization or individual has invested.

- **Security return**
  - The daily return of a security is the difference between the opening value of the...
Simulations and Optimizations

security and the closing value of the security in the stock market; this is usually a monetary value. Similarly, the monthly return of a security is the difference between the opening value on the first trading day of the month and the closing value on the last trading day of the month.

- **Portfolio return**
  An aggregation of the return on all securities in a portfolio. Similar to a security return, it can be calculated on a daily or monthly basis.

For the purposes of this chapter, in this case study we will create a portfolio that includes the top 40 companies traded in NASDAQ. Henceforth, the word portfolio will refer to a collection of these securities. This portfolio consists of a wide range of securities from IT, healthcare, and retail industries.

The purpose of this case study is to simulate the value-at-risk for this portfolio, a statistical technique that is used to measure the level of financial risk for an investment portfolio over a specified period of time. If the value-at-risk is high, investment management should consider optimizing the portfolio. In subsequent sections, we will cover how to optimize the portfolio by maximizing the portfolio return. We will also define the mathematical models and constraints for optimizing the portfolio in the process of solving this problem.

### Technical implementation

This chapter focuses only on the technical implementation of simulating the value-at-risk and optimizing the portfolio return. We will cover how to perform various operations that are needed for this case study in SAP HANA.

### 6.2 Monte Carlo Simulation of Value-at-Risk

To simulate value-at-risk for the case study, we will use the Monte Carlo simulation technique, a popular problem-solving technique for approximating the probability of particular outcomes through multiple simulations using random variables.

In this case study, we are calculating the approximate value of the portfolio return for a future time using multiple trial runs and random variables. Value-at-risk will be the outcome of this statistical approximation for the future return value.

There are different statistical models that can be used for approximating value-at-risk. Here, we will use the GBM model, which uses random variables along with parameters based on the historic performance of a portfolio.

In this section, we will first look at the various random variable generation techniques in SAP HANA. We will then provide insight into the GBM model before moving on to designing logical models in SAP HANA. Finally, we will look at and compare alternative approaches for implementing and simulating value-at-risk.

#### 6.2.1 Random Variable Generation

**Randomization** is the most important step in calculating value-at-risk outcomes in a Monte Carlo simulation. SAP HANA supports four distinct random distribution sampling techniques to generate random variables through the PAL function `DISTRRANDOM`: uniform, normal, weibull, and gamma.

In this section, we will look at these four random distribution techniques, their equations, parameters, and control tables. Then, to better understand the distribution output of the random variables generated, we will discuss binning functionality.

**Random Distribution Techniques**

In this section, we will look at the four random distribution techniques.

- **Uniform**
  Uniform distribution (also known as rectangular distribution) occurs when all intervals are of the same length between each given minimum and maximum value. The number of sampling values in each interval will be uniform. The probability of uniform distribution is constant. SAP HANA uses the uniform distribution function for generating random variables, like the one shown here:

  \[
  f(x) = \begin{cases} 
  \frac{1}{b-a} & \text{for } a < x < b \\
  0 & \text{for } x \leq a \text{ or } x \geq b 
  \end{cases}
  \]

  The distribution control table to generate uniformly distributed random numbers requires minimum and maximum values. Figure 6.1 shows the distribution control table values as an example.
The output of these random variables in a uniform distribution looks as shown in Figure 6.2. The output shows that the number of random variables in each interval is in a similar range.

**Shape of Distribution**

The bars in Figure 6.2 will change with the next random generation. However, although the numbers generated are random, the overall pattern will be same. This is true for all random generation techniques discussed in this chapter.

**Normal**

A *normal distribution* is defined with a variant of $x$, a mean of $\mu$ and a variance of $\sigma^2$. This distribution usually forms a bell shape. An example of a random variable SAP HANA generates with the normal distribution is as follows:

$$f(x;\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

There are two parameters that control normal distribution random number generation: mean and variance. A mean of 0 and variance of 1 are most commonly used for generating a bell-shaped normal distribution. Figure 6.3 shows the control parameters for the normal distribution.

By generating a chart from the Figure 6.3 control values, you can see normally distributed random numbers, as shown in Figure 6.4.
Simulations and Optimizations

Most Monte Carlo simulations require the normal distribution of random variables. For such simulations, you want each simulation that is normally distributed to have a mean of 0 and variance of 1.

The bin shown for the normal distribution technique shows the overall random sample distribution to be normal while using normal distribution.

**Binning**

Binning is a way to group number values into small groups or “bins”. We will discuss the binning functionality in detail in the Binning section.

Aggregating the random numbers shows a mean of 0 and a variance of 1. This is the technique we will use for the case study.

**Weibull**

The following equation shows a two-parameter weibull probability density function of a random variable of SAP HANA:

\[ f(x; \lambda, \beta) = \begin{cases} \frac{\beta}{\lambda} (\frac{x}{\lambda}) \beta - 1 \left(1 - \frac{x}{\lambda}\right) & x \geq 0 \\ 0 & x < 0 \end{cases} \]

This distribution is used for survival analysis and life distribution analysis. The \( \beta \) parameter effects the shape of the distribution, and \( \lambda \) defines the slope or scale of the distribution. Figure 6.5 shows the weibull distribution input parameters.

![Weibull Distribution Input Parameters](image)

**Gamma**

The following equation defines the gamma probability density function with a shape parameter of \( \beta \) and a scale parameter of \( \theta \) (this is an exponential distribution function):

\[ f(x; \beta, \theta) = \frac{\beta^\beta}{\theta^\beta \Gamma(\beta)} x^{\beta-1} e^{-x/\theta} \text{ for } x > 0 \text{ and } \beta > 0 \]

The parameter control table with gamma distribution is shown in Figure 6.7.

![Gamma Distribution Parameter Values](image)

The result of these parameters is shown in the distribution table in Figure 6.8.
As you can see, these four random variable generation techniques produce different distributions based on their parameters.

In the next section, we will look how these results can be understood in SAP HANA.

**Binning**

It is not easy to understand the distribution of outputs generated by random variable generators by looking directly at the result sets. As an alternative, you can leverage the SAP HANA PAL binning functionality to better understand the distribution. Binning is an effective way to understand the distribution of random variables. All PAL functions are multithreaded and use the processing power of the SAP HANA servers, and using a binning function is an efficient way of manually classifying groups in boilerplate SQL code. You can control the output bins by using various parameters and potentially avoid writing boilerplate code in SQL.

Using the SAP HANA AFL framework, generate a wrapper procedure called `BINNING_VAR_PROC` for the `BINNING` PAL function and use the output of the random distribution as an input to this procedure to better understand the distribution of random numbers. Listing 6.1 provides common code for generating random distributions and viewing the result after binning.

```sql
CREATE PROCEDURE MONTECARLO.RANDOM_DISTRIBUTION_BIN (OUT v_output "MONTECARLO"."PAL_BINNING_RESULT_T") LANGUAGE SQLScript READS SQL DATA WITH RESULT VIEW MONTECARLO.BIN_DISTRIBUTION AS 
BEGIN 
input1= SELECT * from "MONTECARLO"."PAL_DISTRRANDOM_DISTRPARAM_TBL"; 
input2 = SELECT * FROM "MONTECARLO"."PAL_CONTROL_TBL_ANIL"; 
CALL "MONTECARLO"."DISTRRANDOM_PROC" (:input1, :input2, dis_inp); 
bin_input = SELECT "ID", "RANDOM" AS "VAR" FROM :dis_inp; 
control_input = SELECT * FROM "MONTECARLO"."BIN_PAL_CONTROL_TBL"; 
CALL "MONTECARLO"."BINNING_VAR_PROC" (:bin_input, :control_input, v_output); 
END;
```

Listing 6.1 Random Variable Generation Code

In Listing 6.1, `DISTRRANDOM_PROC` will generate a distribution as per the control parameter and distribution parameter tables. Use the output of the random generation as an input for the binning function `BINNING_VAR_PROC` to understand the distribution.

**Random Generation Outputs**

The figures provided for each of the random generation techniques are outputs of the procedure in Listing 6.1, with different control parameter values for random distributions. The result view analysis in SAP HANA Studio is used to produce Figure 6.2, Figure 6.4, Figure 6.6, and Figure 6.8.

In a control input table for generating random numbers, there are three parameters for generating input numbers:
Simulations and Optimizations

- **THREAD_NUMBER**
  For performance, this displays how many threads to use to generate a distribution.

- **SEED**
  Initializes the random generation for pseudo random generation. Here, use 0.

- **NUM_RANDOM**
  Indicates the number of random variables to be generated.

Figure 6.9 is an example of a control table for generating random variables.

![Control Table for All Types of Random Sample Distribution](image)

We will use the normal distribution’s random variables in the next section for the Monte Carlo simulation, define the GBM model along with random variables, and run simulations for the portfolio return. Based on these simulated outcomes, we will calculate the value-at-risk.

### 6.2.2 Simulation Model and Process

A randomly distributed process (also referred as a stochastic process) $S_t$ is said to follow the GBM model if it satisfies the following equation, where $W_t$ is a Brownian motion and $\sigma$ and $\mu$ are constants:

$$dS_t = \mu S_t dt + \sigma S_t dW_t$$

The first part ($\mu S_t dt$) of this differential equation is referred to as the drift in time or trend. The second part of the equation ($\sigma S_t dW_t$) is called random noise and controls the trajectory of the simulation. Because it is a random approximation in time, this is a Monte Carlo simulation.

#### Random Numbers in a GBM Model

In a GBM model, the number of random numbers depends on the input parameters of the date you want to know the value-at-risk. It can be 10 days in advance, 180 days in advance, or higher for long-term securities.

The overall process of a value-at-risk simulation is shown in Figure 6.10.

![Overall Value-At-Risk Simulation Process](image)

We will describe the process seen in Figure 6.10 step by step:

- **Stock quotes**
  In this step, we will discuss the data storage of stock market prices.

- **Select stocks**
  This is the process of creating a portfolio and using that portfolio for a simulation. Because you already have defined a portfolio with the top 40 company securities in it, continue using that portfolio.

- **Calculate return**
  In this step, define the SAP HANA logical model for calculating return.

- **Simulate portfolio**
  In this step, discuss the logic for implementing the simulation model.

- **Calculate value-at-risk**
  In this step, calculate the value-at-risk from the simulation outcomes.

In the following sections, we will implement each of these steps in SAP HANA by creating a stock table to calculate the value-at-risk.

#### Stock Quotes

Figure 6.11 shows the table definition of a stock quote with a sample stock. This table is populated every day from NASDAQ or similar stock markets using SAP HANA ETL tools.

As previously discussed, a portfolio is combination of various securities, usually diversified with the various stocks of a company and its weights. For this case study, we have uniformly distributed the weights of randomly picked stocks to create a portfolio (see Figure 6.12).
Simulations and Optimizations

Figure 6.11 Stock Quote Table Structure

<table>
<thead>
<tr>
<th>Name</th>
<th>SQL Data Type</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>INTEGER</td>
<td>1</td>
</tr>
<tr>
<td>SYMBOL</td>
<td>VARCHAR</td>
<td>10</td>
</tr>
<tr>
<td>WEIGHT</td>
<td>DECIMAL</td>
<td>10.6</td>
</tr>
</tbody>
</table>

Figure 6.12 Portfolio Definition Structure

Figure 6.12 is populated with a portfolio that consists of up to 40 securities. Initially, you will weight each of these securities in your portfolio as 1/40, or 0.025. Once the initial portfolio is created, you can calculate the portfolio return.

Calculate Return

The monthly return of a stock is calculated as the difference between the values of a stock on opening day and on the last day of the month. Figure 6.13 shows a table containing the columns MONTH_LAST and MONTH_FIRST for each month. The difference between these two values for each month is the monthly return.

A portfolio’s monthly and daily return is the sum of the monthly and daily returns of all stocks that belong to a portfolio. A calculation view definition that calculates stock returns is depicted in Figure 6.14.

We will use the calculated daily return to determine the mean and standard deviation of the portfolio, which are used as constants in the GBM simulation.

Figure 6.13 Monthly Return of a Security

Figure 6.14 Calculation View Definition of Portfolio’s Monthly and Daily Returns
Simulate Return

Using the concepts covered in Section 6.1 and Section 6.2.1, Listing 6.2 presents the pseudocode for the Monte Carlo simulation with a GBM model.

```sql
select round(AVG(A_R), 6), round(STDDEV(A_R), 6), count(*), round(STDDEV(A_R)/sqrt(count(*)), 6) into drift_of_return, stddev_of_return, number_of_time_steps, dt from (select DATE_SQL, SUM(DAILY_RETURN*WEIGHT) A_R from "_SYS_BIC"."montecarlo/AN_STOCK_RETURN" where ID = :PORTF_ID and DATE_SQL between :BEGIN_DATE and :END_DATE group by DATE_SQL);

TOPROWS := TIME_STEPS * SERIES;

input1 = SELECT * FROM MONTECARLO.PAL_DISTRRANDOM_DISTRPARAM_TBL;
input2 = SELECT * FROM MONTECARLO.PAL_CONTROL_TBL union select 'NUM_RANDOM', :TIME_STEPS, null, null from dummy;

for k in 1 .. :SERIES do
  CALL MONTECARLO.DISTRRANDOM_PROC(:input1, :input2, v_output);
  eps_list = select top :TIME_STEPS "ID" as row_id, RANDOM as eps from :v_output;
  select top 1 :k, 0, :INIT_RETURN, :INIT_RETURN, 0 from MONTECARLO.SIM_LOOP_WORK_TEMP where SERIES = :k order by TIME_STEPS desc;
end for;
```

Listing 6.2 Pseudocode for Monte Carlo Simulation

The following are inputs used in Listing 6.2:

- **Portfolio ID**
  The portfolio for which you want to simulate the future value-at-risk.

- **Date range**
  The daily returns of the stocks selected for a given date range.

- **Series**
  The number of required simulations.

- **Time steps**
  A future date that requires a value-at-risk to be simulated.

- **Reliability**
  The confidence level for the value-at-risk.

- **Base return**
  The future value of this base return will be simulated.

In Listing 6.2, the initial query retrieves the primary parameters from the stock market data (i.e., the drift and trend that are used for the GBM simulation). Then, simulation loops deduce the future value of the portfolio.

As shown in Listing 6.2, the Monte Carlo simulation for value-at-risk using the GBM model involves simulating the daily stock value for the $n$th day in the future for $m$ iterations. Then, the lowest value of the $n$th day, based on a desired confidence level, is used as the value-at-risk. As discussed earlier, the mean and standard deviation are used as constants for trend/drift in time and random noise (also referred to as volatility).

The output of the simulation procedure is shown in Figure 6.15 for the base value of 100 dollars, the number of the simulation as 1, the number of time steps as 5, and a reliability of 0.99.

The output contains the simulation number in the STEPS column. The TIME_STEPS column contains the day from the end-date. The RETURN_column is the simulated portfolio return. We will use this output to calculate value-at-risk in the next section.
Calculate Value-At-Risk

Value-at-risk is the max value of the reliability percent record. This can be calculated using simple SQL by selecting the top percentage records of the simulation value for the nth day. Figure 6.16 illustrates how to calculate value-at-risk from a simulation result created in the previous section.

You can calculate the percent of records that need to be considered for a simulation based on the user input. If the reliability input for the procedure is 0.99, that means value-at-risk is 99% confidence. In this case, choose 1% of the lowest \( \text{RETURNS}_{-} \) of the simulation. The highest value of this 1% record will be determined as the value-at-risk. Remember, you are looking at only the nth day of the simulation; all of the lowest value selections are only for that day.

Now that you have calculated value-at-risk for the simulation output, you will want to see the distribution of the simulation itself. You used the binning technique to see the distribution of random variables, and you can effectively use the same technique to understand the distribution of the simulation of \( \text{RETURN}_{OF} \). Remember, you are interested in the last day of the simulation.

If you bin the output of the simulated value-at-risk for each of the simulations, the distribution will resemble that shown in Figure 6.17.

![Figure 6.15 GBM Simulation Output](image1)

![Figure 6.16 Value-at-Risk Calculation](image2)

![Figure 6.17 Value-at-Risk Simulation Output Distribution](image3)
ing sections we will try another approach to avoid imperative logic with random distribution.

### 6.2.3 Avoiding Imperative Logic

Imperative logic in a SQLScript stored procedure prohibits SAP HANA from using its full processing power due to the sequential nature of the execution requirements. Some of the logic can be executed using SQL statements that make use of parallel processing in SAP HANA. Using set theory or thinking in sets is another approach to utilizing parallel computing power through SQL processing.

Looking at Listing 6.2, you can see that there are a couple of loops. The first loop is used for calculating the return for each day. Another loop is used for the number of simulations. We call this technique a stepwise simulation, which has a loop simulating each day of a return.

If the number of simulations are large, you can explore generating random distributions once for all simulations and adjust the stepwise processing in the formula for the Monte Carlo simulation. Listing 6.3 is pseudocode for this new logic in which we generate the random variables once for all the simulations. We call this a non-stepwise technique.

```sql
select round(AVG(A_R),6), round(STDDEV(A_R),6), count(*), round(STDDEV(A_R)/sqrt(count(*)),6)
into drift_of_return, stddev_of_return, number_of_time_steps, dt
from (
select DATE_SQL, SUM(DAILY_RETURN*WEIGHT) A_R
from "SYS_BIC"."montecarlo/AN_STOCK_RETURN"
where ID = :PORTF_ID
and DATE_SQL between :BEGIN_DATE and :END_DATE
) group by DATE_SQL;

TOPROWS := TIME_STEPS * SERIES;
input1 = SELECT * FROM MONTECARLO.PAL_DISTRRANDOM_DISTRPARAM_TBL;
input2 = SELECT * FROM MONTECARLO.PAL_CONTROL_TBL
union
select 'NUM_RANDOM', :TOPROWS, null, null from dummy;
CALL MONTECARLO.DISTRRANDOM_PROC(:input1, :input2, v_output);
eps_list = select top :TOPROWS "ID" as row_id, CASE WHEN MOD("ID"+1,:TIME_STEPS) = 0 THEN :TIME_STEPS ELSE MOD("ID"+1,

eps2 = select *times*, sigma*eps*sqrt(dt)*dW, dt*times* t, sigma*eps*sqrt(dt)*"times"* W, sigma, dt,
row_id, drift_of_return from :eps_list;
out1 = select FLOOR((row_id-1)/:TIME_STEPS)+1 as "SERIES", "times" as "TIME_STEPS",
(:INIT_RETURN+INIT_RETURN*(drift_of_return*dt+dW))*exp(
(drift_of_return*t-0.5*sigma*sigma*t) + W) "RETURN_"
from :eps2 order by 1;
```

Listing 6.3 GBM without a Loop

In Listing 6.3, the random variables are generated once; the GBM processing is adjusted to cater for the non-stepwise processing required.

Figure 6.18 depicts the overall GBM output without loops.

**Figure 6.18 Non-Stepwise GBM Output**

```
HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH
The output visualization in Figure 6.18 is created using SAP BusinessObjects BI to create visualizations for comparison. The distribution of the simulation in Figure 6.18 is normal. The leftmost four bars in the distribution are the values below and close to value-at-risk; value-at-risk itself is shown in the visualization. The input for the model is taken from the user through the controls.

You can also see a comparison of both stepwise and non-stepwise simulations side by side to observe the differences in calculating value-at-risk. Figure 6.19 shows a comparison of the output of both models.

![Figure 6.19 Comparison of Stepwise and Non-Stepwise GBM Models](image)

The light gray region indicates that the number of simulated outcomes is less than the value-at-risk. The two charts in Figure 6.19 are for two types of simulations; as noted previously, the random sampling distribution is generated once for the entire simulation and then for each individual simulation. Value-at-risk for each technique is different; the random number being distributed once for each simulation seems to be more realistic than the earlier technique.

Value-at-risk values for each of these models differ by a large amount: 15%. Given that this can be adjusted, there are now two different GBM models: stepwise and non-stepwise (which is without imperative logic). Values of simulated value-at-risk are normal distributions in both simulations, but stepwise is closer to the realistic value and the more preferred method for each simulation.
Size coutContent = 0;
Double w = Double("0");
Double eps = Double("0");
Double dW = Double("0");
Double t = Double("0");
Double constant = Double("0.5");
Double RETURN2 = Double(init_return);
Int32 index1 = 1;
while (coutContent < idC.getSize()) {
    eps = randomC[coutContent];
    dW = stddev_of_return * eps * math::sqrt(dt);
    t = dt * Double(index1);
    w = w + dW;
    Double expInpt = (drift_of_return * t - constant * stddev_of_return * stddev_of_return * t) + w;
    RETURN2 = RETURN2 + Double(rETURN2) * (drift_of_return * dt + dW) * math::exp(expInpt);
    mon_time[table_counter] = index1;
    mon_return[table_counter] = RETURN2;
    mon_var_flg[table_counter] = 0;
    mon_return_org[table_counter] = expInpt;
    mon_series[table_counter] = 0;
    coutContent = coutContent.next();
    table_counter = table_counter.next();
    index1 = index1 + 1;
}
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo1);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo2);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo3);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo4);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo5);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo6);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo7);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo8);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo9);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo10);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo11);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo12);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo13);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo14);
callVaR( times_each, drift_of_return, stddev_of_return, dt, init_return, read1, read2, montecarlo15);
}

 Listing 6.4 LLANG Procedure

In Listing 6.4, SAP HANA parallelizes the execution of logic in the _parallel block. We have implemented GBM in a callVaR function in a LLANG procedure and called it multiple times in the parallel block (_parallel). This enables us to perform a large number of simulations in matter of subseconds. One million Monte Carlo simulations are executed in less than five seconds with this approach.

With this, you have now seen all of the random variable generation techniques in SAP HANA and their use in the Monte Carlo simulation for calculating value-at-risk using the GBM model. You have also seen various alternatives for implementing the simulation and compared the outcomes.

Once an investment manager identifies risk in a portfolio, the next step is to optimize the portfolio for maximise return, which we will cover in the next section.

6.3 Portfolio Optimization

A large number of securities are traded on various bases around the world every day. An investment portfolio is a combination of various securities with distributed weights as per diversification rules. Performance of a stock/security is usually measured by the return of a security for a particular period in time until the time
in question. The volatility of a portfolio is determined by the weighted volatilities of the individual security returns. This volatility is also known as risk. The Sharpe ratio is defined as the ratio of return to risk. The performance of a portfolio is assessed based on values of the return, risk, and Sharpe ratio. A maximum return with minimal risk is an ideal expectation for an investment in a portfolio. In other words, the higher the Sharpe ratio, the higher the return.

Portfolio optimization is the process of maximizing returns and minimizing risk, or maximizing the Sharpe ratio under specified constraints. One of the critical components for optimizing a portfolio is building a variance-covariance matrix that explains the correlation amongst all the securities in a portfolio. The covariance matrix return, risk, and Sharpe ratio are defined as follows:

- **Return**: 
  \[ R = \sum_{i=1}^{n} (W_i \times r_i) \]
  where:
  - \( W_i \) is the weight of the \( i \)th security in a portfolio.
  - \( r_i \) is the return on the \( i \)th security (daily or monthly).
  - \( R \) is the return of a portfolio \( P \).
  - \( n \) is the number of securities in a portfolio.

- **Risk**: 
  \[ \sigma^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} (W_i W_j \sigma_{ij}) \]
  where:
  - \( W_i \) is the weight of the \( i \)th security in a portfolio \( P \).
  - \( \sigma_i \) is the variance of the \( i \)th security in a portfolio \( P \).
  - \( \sigma_j \) is the variance of the \( j \)th security in a portfolio \( P \).
  - \( \rho_{ij} \) is the covariance of the \( i \)th and \( j \)th security in a portfolio \( P \).

- **Sharpe ratio**: 
  This is the ratio of return to risk. In this example, we will maximize the return using matrix algebra, as explained in the following sections.

Figure 6.20 illustrates the overall process to optimize a portfolio.

We will use the same stock definitions and portfolio and return calculations as introduced previously (see Section 6.2.2). The following sections cover the remaining processes involved.

### 6.3.1 Variance-Covariance Matrix

A variance-covariance matrix of a portfolio consists of a correlation of each security in a portfolio with others. A variance-covariance matrix can be designed in two ways in SAP HANA: via a matrix structure or a table structure. The following two sections look at each option in detail.

#### Matrix Structure

As shown in Figure 6.21, a matrix structure is dependent on the number of securities. For the matrix structure table design, if you know the securities of a portfolio beforehand, you can define the physical data model of a table as a matrix structure as shown in Figure 6.21. If you want to be flexible for optimizing and reoptimizing purposes and there are different securities to optimize, consider using SAP HANA flexible tables. A flexible table allows you to add columns dynamically. In SAP HANA, you can use the `SCHEMA FLEXIBILITY` expression for this option and add the columns while performing optimization processes. Listing 6.5 displays the set SQL statements that will define a flexible table.

![Figure 6.20 Portfolio Optimization Process](image)

![Figure 6.21 Matrix Structure](image)

For the matrix structure table design, if you know the securities of a portfolio beforehand, you can define the physical data model of a table as a matrix structure as shown in Figure 6.21. If you want to be flexible for optimizing and reoptimizing purposes and there are different securities to optimize, consider using SAP HANA flexible tables. A flexible table allows you to add columns dynamically. In SAP HANA, you can use the `SCHEMA FLEXIBILITY` expression for this option and add the columns while performing optimization processes. Listing 6.5 displays the set SQL statements that will define a flexible table.
CREATE COLUMN TABLE COVARIANCE(
  SECURITY VARCHAR(15),
  SECURITY_A DECIMAL(6,3),
  SECURITY_B DECIMAL(6,3)
) WITH SCHEMA FLEXIBILITY;

INSERT INTO MONTECARLO.COVARIANCE VALUES ('SECURITY_A', 0.6, 0.7);

Listing 6.5 Flexible Table

With the INSERT statements in Listing 6.6, you can add new columns to this table. The new columns are SECURITY_C and SECURITY_D, with NVARCHAR(5000) for the type.

INSERT INTO MONTECARLO.COVARIANCE (SECURITY, SECURITY_A, SECURITY_B, SECURITY_C) VALUES ('SECURITY_A', 0.6, 0.7, 0.8);
INSERT INTO MONTECARLO.COVARIANCE (SECURITY, SECURITY_A, SECURITY_B, SECURITY_C, SECURITY_D) VALUES ('SECURITY_A', 0.6, 0.7, 0.8, TO_DOUBLE(-0.3));

Listing 6.6 Flexible Table Population with New Columns

Irrespective of the value of the new field type, the new column type will be NVARCHAR. This requires changing additional type conversion operations to numeric. In Figure 6.22, the columns inserted with INSERT statements are created with NVARCHAR data types.

Figure 6.22 Table Columns with Flexibility Option

For the portfolio, we have considered more than 2,000 securities, but it would be difficult to manage the table structure with so many columns. In such a case, the table structure can be used instead of the matrix structure. The other advantage of a matrix structure is that building a covariance matrix is possible with the SAP HANA PAL multivariate statistics function, MULTIVARSTAT.

Table Structure

A table structure has consistent columns irrespective of the number of securities and grows with the number of rows. In Figure 6.23, note that only a number of rows differ based on the number of securities in the portfolio.

<table>
<thead>
<tr>
<th>SYMBOL1</th>
<th>SYMBOL2</th>
<th>Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security A (ABC)</td>
<td>Security B (JP)</td>
<td>8.5</td>
</tr>
<tr>
<td>Security A (ABC)</td>
<td>Security C (AP)</td>
<td>6.2</td>
</tr>
<tr>
<td>Security A (ABC)</td>
<td>Security D (SAP)</td>
<td>8.2</td>
</tr>
<tr>
<td>Security A (ABC)</td>
<td>Security E (IBM)</td>
<td>6.6</td>
</tr>
</tbody>
</table>

Figure 6.23 Table Structure

In the next section, we will see how we can implement and populate a covariance matrix in the columnar table structure.

Columnar Table

The columnar structure is easier with respect to maintaining a structure; however, because it is not in a matrix format, you need to rely on low-level data structures of LLANG on SAP HANA or RLANG for the matrix operations. Because SAP HANA supports R integration, it would be simplest to perform some of these activities with R.

Also, it is easier to populate covariance values into a columnar table structure using the following formula. For a calculation, you need to sum the squares of the returns for the two securities and use the correlation calculation formula shown here:

\[
\rho(x, y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}
\]

The SQL query shown in Listing 6.7 will calculate covariance for securities.

SELECT ID, RUNID, SYMBOL_A, SYMBOL_B, NUM1 as "CORR", COMMONSIZE, "TYPE" FROM
FROM
SELECT :portfId as "ID", :runid as "RUNID", SYMBOL_A, SYMBOL_B, round(VAR_A,6) VAR_A, round(VAR_B,6) VAR_B, round((psum / n - sum1 /n * sum2 / n), 6) as "NUM1",
sqrt((sum1sq - power(sum1, 2.0) / n) * (sum2sq - power(sum2, 2.0) / n)) as "NUM2",
round(((psum - (sum1 * sum2 / n)) /sqrt((sum1sq - power(sum1, 2.0) / n) * (sum2sq - power(sum2, 2.0) / n))),6) AS "CORR",
n AS "COMMONSIZE", 'M' as "TYPE"
FROM

Listing 6.7 Calculating Covariance Matrix Using SQL

Covariance Table Data
With a large number of securities in a portfolio and the easy population of covariance values, a table structure suits our portfolio optimization. The covariance table structure is shown in Figure 6.24.

Figure 6.24 Covariance Table Structure

We will populate the table in Figure 6.24 using the SQL in Listing 6.7 for calculating the correlation factor. Figure 6.25 shows some of the first records of the covariance table.

![Sample Output of Covariance Table](image)

The stronger the positive correlation between the two securities, the closer the value of \( CORR \) is to 1. Positively correlated stocks exhibit similar upward or downward movements more often than not. Negative correlation is signified by a value closer to -1. This indicates a strong inverse movement between two securities. This information is crucial for the process of optimization and portfolio planning.

Bidirectional Covariance Values

The SQL in Listing 6.7 will populate bidirectional covariance values into the table. You will have two entries for the covariance of securities ABC and C corporation, with \( SYMBOL_A \) AS 'ABC' and \( SYMBOL_B \) AS 'C', and another entry with the reverse combination with \( SYMBOL_A \) AS 'C' and \( SYMBOL_B \) AS 'ABC'.
Use SAP BusinessObjects BI tools to visualize covariance analysis based on the correlation value. Figure 6.26 shows a visualization for the top positive and negative correlated securities based on the correlation factor.

In Figure 6.26, notice at the top of the image that if the correlation factor is greater than 0.79 (strong positive correlation), the securities have a similar trend of returns. Similarly, negative correlated securities at the bottom of the image exhibit the same pattern.

In this section, we have seen the various options for creating a covariance matrix. We populated the covariance matrix between securities in a portfolio in a tabular format for this case study. We now know how we will use the covariance matrix for maximizing the portfolio return by defining the matrix algebra models in the next section.

### 6.3.2 Modeling for Optimization Constraints and Equations

Portfolio optimization was proposed in 1970 by Harry Max Markowitz using the covariance matrix and matrix algebra. In the next sections, we will go through the model and how we will derive the equations for optimization execution with an example.

#### Optimization Model

In this section, we will discuss matrix algebra and derive the optimization equation for maximizing the return.

The following formula (formula 1) is based on the portfolio optimization theory:

\[
\begin{bmatrix}
2V & R \\
R & 0
\end{bmatrix}
\begin{bmatrix}
W \\
\lambda
\end{bmatrix}
= \begin{bmatrix}
0 \\
1
\end{bmatrix}
\]

\[(1)\]

In this example, \(V\) stands for variance matrix, \(W\) is the weight, \(R\) stands for the return, and \(r^*\) is the expected return of the portfolio.

You can rewrite this equation for calculating the inverse matrix as follows (formula 2):

\[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 1 \\
0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
\lambda_1 / 2 \\
\lambda_2 / 2 \\
\lambda_2 / 2
\end{bmatrix}
= \begin{bmatrix}
r^* \\
1 \\
0
\end{bmatrix}
\]

\[(2)\]

Finally, let’s represent the above equation (2) in a simplified format (formula 3):

\[
(A) \times (W) = R^*\]

\[(3)\]

Where \(A\) is the return and variance matrix, \(W\) is the weight matrix, and \(R^*\) is the return matrix.

Let’s run through an example of a variance matrix and a return matrix to derive the equations for an optimization based on the preceding simplified formula (3). Let’s consider an example of a three-by-three variance matrix \(V\), as shown in Table 6.1.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Symbol A</th>
<th>Symbol B</th>
<th>Symbol C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol A</td>
<td>0.5</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>Symbol B</td>
<td>0.2</td>
<td>0.6</td>
<td>0.3</td>
</tr>
<tr>
<td>Symbol C</td>
<td>0.1</td>
<td>0.3</td>
<td>0.7</td>
</tr>
</tbody>
</table>

Table 6.1 Example Variance Matrix
Similarly, let’s consider a return matrix for symbols in V for R, as shown in Table 6.2.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol A</td>
<td>5</td>
</tr>
<tr>
<td>Symbol B</td>
<td>4</td>
</tr>
<tr>
<td>Symbol C</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 6.2  Return Matrix Example

Using the example V and R matrix, A is as shown in Figure 6.27.

Figure 6.27  Matrix Example Including the Returns and Variance Matrix

The inverse matrix of A⁻¹ is shown in Figure 6.28.

Figure 6.28  Inverse Matrix of Example A

Given that the expected return is an input, you can derive the weights from formula 2, as shown in Figure 6.29.

As shown in the first two columns of Figure 6.29, the inverse matrix will calculate each weight as defined here:

\[
W_1 = 0.5 \times r^* - 1.611111 \\
W_2 = (1.29 - \text{E}16) \times r^* + 0.222222 \\
W_3 = -0.5 \times r^* + 2.388889
\]

If the calculated weights \(W_1, W_2,\) and \(W_3\) are less than 0, then you need to iterate the process, because you should consider \(W > 0\) as an optimizing constraint.

Once you have the weights calculated for the inverse matrix, write the weights matrix for formula 2 and the variance matrix \(V\) for formula 1, as follows:

\[
W = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix} = \begin{bmatrix} a_1r^* + b_1 \\ a_2r^* + b_2 \\ a_3r^* + b_3 \end{bmatrix}
\]

\[
V = \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix}
\]

Next, we want to show that the expected return is always between the minimum and maximum of \(R\) as follows (you will use this in the Sharpe ratio equation):

\[
\text{Return} = r^* \text{ (min}(R) \leq \text{max}(R))
\]

Now, write the defining formula for risk, which is related to the previous formula:

\[
\text{Risk} = W^T V W = \begin{bmatrix} a_1r^* + b_1 \\ a_2r^* + b_2 \\ a_3r^* + b_3 \end{bmatrix} \begin{bmatrix} \sigma_{11} & \sigma_{12} \\ \sigma_{21} & \sigma_{22} \end{bmatrix} \begin{bmatrix} a_1r^* + b_1 \\ a_2r^* + b_2 \\ a_3r^* + b_3 \end{bmatrix}
\]

\[
= \lambda \frac{1}{2} A^1 + B^1 - C
\]

Once you have defined the risk, you can use the Sharpe ratio equation for optimizing the returns:
The Sharpe ratio equation will be used to optimize returns. As you have seen in this section, in order to optimize the Sharpe ratio, you first need to perform a matrix inversion operation using the covariance matrix built in Section 6.3.1. Then, use the preceding equation for optimization. The next section will cover the inverse matrix.

## Inverse Matrix

Now that we have defined the covariance matrix in tabular form in the Covariance Table Data section, you need to transform the data into the matrix form and perform an inverse matrix operation. You will use R to transform and retrieve the inverse matrix.

To begin, define the input table types of an RLANG procedure. Figure 6.30 shows the input table type.

**Figure 6.30 Inverse Matrix Input Table Type**

You only need two output values to optimize the portfolio for each symbol, as explained in the previous section. Figure 6.31 displays the output table type.

**Figure 6.31 Output of Inverse Matrix Procedure**

Next, let’s look at the RLANG procedure code. Listing 6.8 displays the RLANG procedure that will transform the covariance matrix in the table structure to the matrix structure and calculate the inverse matrix.

```r
library(plyr) library(reshape) library(reshape2) library(data.table)

Cov_list<-data.table(covariance_list)
Cov_matrix<-cast(Cov_list, SYMBOL_A ~ SYMBOL_B)
Cov_inverse_matrix<-solve(Cov_matrix)
Cov_inverse_matrix_frame<-cbind(row.names(Cov_inverse_matrix),Cov_inverse_matrix[,c(1:2)])
result<-as.data.frame(Cov_inverse_matrix_frame)
```

**Listing 6.8 Transform the Structure and Perform the Inverse Matrix Operation**

The previously mentioned R function `solve` will calculate the inverse of the matrix, and `cast` will transform the function from the table format to the matrix structure. Finally, select the first two values for each security for optimization and send them back to SAP HANA (see Figure 6.32 for the output values of Listing 6.8).

**Figure 6.32 Output of Inverse Matrix for Optimization**
Insert the output in Figure 6.32 into the WEIGHT_PARAMETER table, which will be used to calculate the expected result in the next section.

**Calculate Expected Return**

Next, we will minimize and maximize the return using the derived equation in R by passing the relevant parameters of A, B, and C calculated in SAP HANA and leveraging the optimized function in R to calculate the expected return.

Begin by defining the input table type for the RLANG procedure, as shown in Figure 6.33.

![Figure 6.33 Input Table Type to Optimize Function](image)

You can then calculate A, B, and C for the model as follows using SAP HANA SQL (see Listing 6.9).

```sql
select sum(A) as A, sum(B) as B, sum(C) as C, sum(max_return) as MAX_RETURN, sum(min_return) as MIN_RETURN from (select sum(CORR*IR_A*IR_B) as A, sum(CORR*(IR_A*IO_B+IR_B*IO_A)) as B, sum(CORR*IO_A*IO_B) as C, null as max_return, null as min_return from (select M.SYMBOL_A, M.SYMBOL_B, M.CORR as CORR, WP_A.INVERSE.Return as IR_A, WP_A.INVERSE.ONE as IO_A, WP_B.INVERSE.Return as IR_B, WP_B.INVERSE.ONE as IO_B from "MONTECARLO"."COV_MATRIX_NN" as M inner join "MONTECARLO"."WEIGHT_PARAMETR" as WP_A on M.SYMBOL_A = WP_A.SYMBOL inner join "MONTECARLO"."WEIGHT_PARAMETR" as WP_B on M.SYMBOL_B = WP_B.SYMBOL)
```

**Listing 6.9 Calculation Equation Parameters in SQL**

In addition to the Listing 6.9 code, also select the minimum and maximum returns of the securities to pass them as an input to the RLANG procedure. Listing 6.10 is the pseudocode of the RLANG procedure.

```r
library(data.table)
p<-data.table(r_input)
A<-p[1,A]
B<-p[1,B]
C<-p[1,C]
max_return<-p[1,MAX_RETURN]
min_return<-p[1,MN_RETURN]
f< function(x) x/(A*x^2+B*x+C)
xopt_max< optimize(f=f,interval=c(min_return,max_return),tol=0.0000001, maximum=T)
xopt_min< optimize(f=f,interval=c(min_return,max_return),tol=0.0000001)
expected< matrix(c(xopt_max$maximum,xopt_max$objective,xopt_min$minimum,xopt_min$objective), nrow=1, ncol=4)
colnames(expected)<-c("OPTIMIZED_EXPECTED_RETURN","OPTIMIZED_SHARPE_RATIO","MINIMUM_EXPECTED_RETURN","MINIMUM_SHARPE_RATIO")
result< as.data.frame(expected)
```

**Listing 6.10 Optimize the Return**

Note that the `optimize` function definition is defined in the model, which is followed by calling the minimize and maximize optimized functions. The outputs of the optimized functions are maximized and minimized returns with their respective Sharpe ratios.

Redistribute the portfolio by assigning weights, using the maximum expected return `OPTIMIZED_EXPECTED_RETURN` from the procedure in Listing 6.10 to achieve maximum returns. You will store this output in a table in SAP HANA.

**Calculate Weights**

As per the model, a new weight can be calculated based on the maximized expected return and the inverse matrix as $a_r^r + b_1$ for a security. Use following SQL statements to calculate the same:

```sql
calculated_weights = select :portfId, :runid, :expected_return, SYMBOL, INVERSE_RETURN * :expected_return + INVERSE.ONE from "MONTECARLO"."WEIGHT_PARAMETER"
```

The output looks like that shown in Figure 6.34 for securities in a portfolio.
The overall expected return of the portfolio with the new weights calculated in Figure 6.34 can be calculated as shown in Listing 6.11.

```sql
portfolio_return =
    select :portfId, :runid, :expected_return, var_, :expected_return/var_
    from (select sum(CORR*WEIGHT_A*WEIGHT_B) as var_
        from (select M.CORR as CORR, W_A.WEIGHT as WEIGHT_A, W_B.WEIGHT as WEIGHT_B
            from "MONTECARLO"."COV_MATRIX_NN" as M
            inner join :calculated_weight as W_A
                on M.SYMBOL_A = W_A.SYMBOL and W_A.ID = :portfId and W_A.RUNID = :runid and W_A.EXPECTED_RETURN = :expected_return
            inner join :calculated_weight as W_B
                on M.SYMBOL_B = W_B.SYMBOL and W_B.ID = :portfId and W_B.RUNID = :runid and W_B.EXPECTED_RETURN = :expected_return)
        )
    )
```

The output of Listing 6.11 looks like that shown in Figure 6.35.

In this section, we performed an inverse matrix operation to get the weights for each security. We then derived the optimization equation from the matrix algebra and performed the matrix transformation. Finally, we calculated the maximum expected returns using the optimization function.

We still need to recalculate the portfolio weights until the constraints of optimization are met, which we will cover in the next section.

### 6.3.3 Executing Optimization Models

One of the constraints of a portfolio is that there should not be any negative weights; that is, weights should be greater than 0.

A new portfolio can be created by removing the negatively weighted securities from recalculated weights and performing the same iterative process of optimizing until you reach a portfolio optimized without negative stocks.

In this example, a portfolio optimization simulation exercise with 2,742 securities is executed 12 times to remove all negative stocks—and finishes in under 10 minutes. Figure 6.36 shows a set of optimal portfolios with a defined level of expected return (known as an efficient frontier) from which the calculated Sharpe ratio and risk derived from the simulation are depicted.
6.4 Summary

In this chapter, we discussed how to perform a Monte Carlo simulation to calculate value-at-risk using a GBM model. During the simulation process, we presented various random variable generation and imperative logic techniques. We also described how to perform matrix algebra based on portfolio optimization in SAP HANA. During this process, we also looked at matrix data design and matrix operations such as the inverse matrix and using optimized functions.

This completes our journey through advanced data modeling in SAP HANA. In this book, we looked at SAP HANA modeling capabilities, how to build complex models for complex logic, and how to scale these models for high performance. Together, we explored how the SAP HANA platform allows us to perform predictive modeling using R and PAL together for predictive models. We concluded the book by looking at a simulation and optimization case study for performing data structure operations in SAP HANA. With this book, you should now be able to model complex logic in SAP HANA, build predictive models with R, PAL, and more, and utilize performance tools to enhance your SAP HANA data models.
Contents

1 SAP HANA Data Models ........................................................................... 21
  1.1 SAP HANA Database Architecture Overview ....................................... 21
  1.2 SAP HANA Modeling Paradigms ............................................................ 22
    1.2.1 Client and Data Connection .......................................................... 23
    1.2.2 Modeled Views ........................................................................... 24
    1.2.3 Stored Procedures ....................................................................... 24
    1.2.4 C++ (Application Function Libraries) ......................................... 25
    1.2.5 L Language ................................................................................ 25
    1.2.6 R Language ............................................................................... 25
  1.3 Information Views .............................................................................. 26
    1.3.1 Attribute Views ........................................................................... 26
    1.3.2 Analytic Views ............................................................................ 37
    1.3.3 Calculation Views ....................................................................... 52
  1.4 Analytic Privileges ............................................................................ 67
    1.4.1 Classical XML-Based Analytic Privilege ....................................... 67
    1.4.2 SQL-Based Analytic Privilege ..................................................... 73
  1.5 Stored Procedures .............................................................................. 75
    1.5.1 SQLScript Procedures .................................................................. 75
    1.5.2 L Procedures ................................................................................ 81
    1.5.3 R Procedures ............................................................................... 82
  1.6 Application Function Library .............................................................. 86
    1.6.1 Business Function Library ............................................................ 86
    1.6.2 Predictive Analysis Library ............................................................ 88
  1.7 Summary ........................................................................................... 90

2 Modeling Complex Logic ........................................................................ 93
  2.1 Achieving Recursive Logic with Hierarchies ......................................... 93
    2.1.1 Creating Hierarchies with Tables ................................................ 94
    2.1.2 Creating a Hierarchy in an Attribute or Calculation View ............. 97
    2.1.3 Hierarchy View Attributes ............................................................ 100
  2.2 Transposing Columns and Rows .......................................................... 110
    2.2.1 Column-to-Row Transposition ..................................................... 110
    2.2.2 Row-to-Column Transposition .................................................... 116
    2.2.3 Reversing a Matrix ...................................................................... 118
## Contents

2.2.4 Merging Data from Multiple Records ........................................ 121
2.2.5 Splitting Strings ................................................................. 122
2.3 Using cube() with Hierarchies .................................................. 123
2.4 Calculating Running Total ......................................................... 127
2.5 Calculating Cumulative Sum ..................................................... 131
2.6 Filtering Data Based on Ranking ................................................. 134
2.6.1 Using a Subquery ................................................................ 134
2.6.2 Using Window Functions ...................................................... 135
2.6.3 Manipulating Concatenated Virtual Columns ......................... 136
2.6.4 Using a Rank Node in a Calculation View ............................... 137
2.7 Controlling Join Paths via Filters ............................................. 138
2.8 Full Outer Join in a Calculation View ....................................... 143
2.9 Making Dynamic Queries in a Stored Procedure ....................... 148
2.9.1 Changing Tables Dynamically ............................................. 148
2.9.2 Changing Filters Dynamically ............................................. 150
2.9.3 Changing Output Columns Dynamically .................................. 152
2.10 Showing History Records Side By Side .................................... 153
2.11 Sample Data ............................................................................ 158
2.11.1 Using RAND() ..................................................................... 158
2.11.2 Using SrowidS .................................................................... 158
2.11.3 Using Identity Columns ..................................................... 159
2.11.4 Using LIMIT/OFFSET ......................................................... 160
2.11.5 Using the TABLESAMPLE SYSTEM .................................. 160
2.12 Using a Vertical Union to Join Tables ...................................... 161
2.13 Sorting Records ....................................................................... 163
2.13.1 Sorting IP Addresses ........................................................ 163
2.13.2 Sorting with Exceptions .................................................... 166
2.13.3 Sorting with User-Defined Rules ....................................... 167
2.14 Finding Missing Values .......................................................... 168
2.14.1 Using the NOT IN Clause .................................................... 169
2.14.2 Using a Self-Join .............................................................. 170
2.14.3 Using a Vertical Union ....................................................... 171
2.14.4 Using Window Functions ................................................... 171
2.15 Using Window Functions for Complex Grouping ..................... 172
2.16 Joining Based on a Date Sequence ......................................... 178
2.17 Using a Nested Calculation View .......................................... 185
2.18 Summary ................................................................................ 191

3 Scaling for Large Datasets ............................................................... 193

3.1 Partitioning ................................................................................. 193
3.1.1 Round-Robin Partitioning .................................................... 194
3.1.2 Range Partitioning .............................................................. 194
3.1.3 Hash Partitioning .............................................................. 195
3.1.4 Two-Level Partitioning ....................................................... 197
3.2 Using Input Parameters to Enforce Pruning ............................... 198
3.3 Creating an Index ................................................................. 201
3.4 Analyzing Query Performance with Tools ................................ 205
3.4.1 Explain Plan ......................................................................... 205
3.4.2 Visualize Plan ................................................................. 207
3.4.3 Performance Trace ......................................................... 208
3.5 Enforcing Execution Paths ....................................................... 214
3.6 Using a Union with Constant Values Instead of a Join ............ 218
3.7 Manipulating Joins in an Analytic View .................................... 224
3.7.1 Breaking a Union of Dimension Tables ............................... 225
3.7.2 Making Nonequi Joins ....................................................... 230
3.7.3 Modifying Tables ............................................................ 232
3.8 Time Traveling ........................................................................ 236
3.8.1 History Tables ................................................................. 237
3.8.2 Simulated History Tables .................................................. 240
3.9 Storing Temporary Data .......................................................... 243
3.10 Calculating Count Distinct ..................................................... 247
3.11 Using Cached Views ............................................................... 250
3.11.1 Defining a Result Cache .................................................... 251
3.11.2 Defining a View Cache .................................................... 254
3.12 Summary ................................................................................ 258

4 Basic Predictive Modeling ............................................................. 259

4.1 Predictive Analytics Lifecycle in SAP HANA .............................. 259
4.1.1 Commonly Used Models .................................................... 261
4.1.2 Predictive Algorithms in SAP HANA .................................. 264
4.1.3 Application Function Library ............................................. 266
4.1.4 Business Example ............................................................ 269
4.2 Data Exploration ..................................................................... 270
4.2.1 Understanding Sales Data .................................................. 270
4.2.2 Correlation and Autocorrelation ....................................... 279
4.2.3 Deterministic Variables ..................................................... 285
Index

_SYS_BI.M_TIME_DIMENSION, 199
$rowid$, 158
$validfrom$, 239
2-D geometries, 57

A

ABC analysis, 90
Affinity propagation, 297
AFL, 25, 86, 308, 349
framework, 267
AFM, 310
Aggregated column, 54
Aggregation, 122, 144
level, 145
node, 29, 54, 57, 120, 153
Algorithm
parameters, 267
specific results, 327
ALTER VIEW, 254
Analytic privileges, 45, 67
create, 67
Analytic view, 24, 34, 37, 66, 71, 129, 153
data foundation, 38
history records, 156
manipulating joins, 225
modifying tables, 232
nonequi joins, 231
Ancestor-descendant relationship, 108
Ancestors, 105
Anomalies, 294
Anomaly detection, 88, 293
Application Function Library → AFL
Application Function Modeler (AFM), 265, 308, 329
Application functions, 86
APPLY_FILTER, 151
ARIMA, 300, 307
variants, 300
ARIMAX, 300, 302, 305
Array type, 82
Ascendants, 105
Assess, 263
Assets-related functions, 86
Assign restrictions, 68, 72
Association
algorithms, 89
analysis, 298
Asymmetric distribution, 274
Attribute definition, 33
Attribute view, 24, 26, 34, 66
dimensions, 37
linked, 229
Auto commit, 237
Autocorrelation, 279, 282
Automatic analytics, 311

B

Base return, 355
Batch predictive modeling, 329, 330
BFL, 25, 87
function categories, 86
library, 88
Bidirectional covariance values, 369
Binning, 89, 348, 357
Bi-variate geometric regression, 89
Bi-variate natural logarithmic regression, 89
Block, 82
Bottleneck, 214
Brown’s exponential smoothing, 301
Business algorithms, 86
Business Function Library → AFL
Business understanding, 261

C

C++, 23, 24, 25, 90
Cache
filter values, 255
hints, 256
invalidation period, 51, 253
limited columns, 255
no cache, 252
refreshing, 256
Cache (Cont.)
  result cache, 251
time controlled, 252
view cache, 250, 254, 257
Calculate
  before aggregation, 43
  expected return, 376
  profit, 179
  return, 351, 352
weights, 377
Calculated
  attribute, 47
column, 43, 48, 203
measures, 66
Calculation engine, 24, 55, 320
plan operators, 66, 161
scenario, 47
Calculation view, 24, 34, 47, 52, 66, 111, 116, 187, 267, 353
filtering, 141
model, 328
nested, 185
type, 53
Calendar table, 34, 128, 132, 199
Call pattern, 212
Cardinality, 179
Cartesian join, 111
Cash flow-related functions, 87
Catalog procedures, 73
Categorical type variable, 292
CE, VERTICAL_UNION, 161
CHAI D decision tree, 89
Chain designing, 318
Character-based analysis, 338
Children, 103
Churn analysis, 278
Classical XML-based analytic privilege, 67
Classification, 297, 327, 333
algorithms, 88
Clients, 23
Cluster analysis, 88
Clustering, 297, 338
algorithms, 88
Column
  adding comments, 38
eengine, 206
type, 43
  view, 34, 96
Columnar table, 367
Commit ID, 237
Complex
  filters, 151
  logic, 21, 82, 93
  processing logic, 60
Composite index, 203
Concat attributes, 51
Concatenated
  column, 134
  virtual columns, 136
Concatenation, 248
Constant values, 221
Constraints, 370
Control
  parameters, 299
table, 350
Correlated securities, 370
Correlation, 279, 326, 338
Cost-calculation functions, 87
Count columns, 247
Count distinct, 247
  aggregation, 247
  count(), 144
Countermeasure, 57
Covariance matrix, 287
return, 364
Covariance table data, 368
Covariance-variance matrix
  matrix structure, 365
table structure, 367
CRISP-DM, 261
Cross client, 50
Crosstabs smoothing, 301
Cumulate, 87
Cumulative sum, 131
Cyclic effects, 276
Cygwin Xwin Server, 210
D
Daily return, 341
Data
  analysis, 272
  Cont.
Column (Cont.)
  view, 34, 96
  Columnar table, 367
  Commit ID, 237
  Complex
    filters, 151
    logic, 21, 82, 93
    processing logic, 60
  Composite index, 203
  Concat attributes, 51
  Concatenated
    column, 134
    virtual columns, 136
  Concatenation, 248
  Constant values, 221
  Constraints, 370
  Control
    parameters, 299
    table, 350
  Correlated securities, 370
  Correlation, 279, 326, 338
  Cost-calculation functions, 87
  Count columns, 247
  Count distinct, 247
    aggregation, 247
    count(), 144
  Countermeasure, 57
  Covariance matrix, 287
  return, 364
  Covariance table data, 368
  Covariance-variance matrix
    matrix structure, 365
    table structure, 367
  CRISP-DM, 261
  Cross client, 50
  Crosstabs smoothing, 301
  Cumulate, 87
  Cumulative sum, 131
  Cyclic effects, 276
  Cygwin Xwin Server, 210
Data (Cont.)
  category, 49, 53
  connection, 23
  exploration, 260, 270
  filtering, 134
  foundation, 38, 41, 59
  frame, 326
  granularity, 279
  loading, 260
  merge, 121
  mining, 259
  partitions, 332, 333
  preparation, 261, 262, 291
  provision, 242
  pruning, 195, 198
  types, 324
  volume, 198
  Data definition language (DDL), 203
  Data modeling, 21
  complex logic, 93
data.frame, 83
  Database Shared Library (DBSL), 23
Datasets, 193
Date
dimension, 36
  hierarchy, 125
  range, 355
  sequence, 178
DDL statement, 78
Decision tree, 89
Declarative logic, 65
Decumulate, 87
Default
  client, 49
  schema, 51, 63
  Definer’s rights, 63
Density-based spatial clustering of
  applications with noise (DBSCAN), 88
Dependent variables, 237
Deployment, 262
Descendants, 105
Descendants(), 96
Deterministic variables, 285
Developer mode, 31
Dict, 82
Difference, 105
Difference-sign test, 284
Dimensions, 26, 66, 123
tables, 111
Distribution, 357
Double exponential smoothing, 301
Drift in time, 350
Duplication, 185
Dynamic
  filter, 148, 150
  output columns, 148, 152
  queries, 148
tables, 148
time warping, 338
E
Edit hierarchy, 29
Embedded model, 78
Enable history, 51
Equations, 370
Equijoin, 230
Errors, 29
ETL, 260, 351
jobs, 254
Evaluation, 262
Execution
  path, 148, 214
  plan, 40, 48, 59, 170, 205
  Exemplars, 297
  Expert analytics, 315
  Explain plan, 205, 206
  Explore, 263
Exponential
  regression, 89
  smoothing, 316
F
Filters, 56, 138, 140, 158, 200
cache, 255
controlling join paths, 138
Optimization (Cont.)
strategies, 48
OR operator, 140
Ordinal or ranking scale, 291
Outer join, 40, 138

P
Package privilege, 69
PAL, 25, 63, 86, 88, 259, 268, 319, 333, 343, 366
architecture, 334
calling a procedure, 90
function categories, 88
model consumption, 326
scaling, 335
stratification, 321
time series functions, 300
using R, 337
Parallel
coordinates, 317
execution, 82
processing, 331, 361
Parent, 28, 103
Parent-child
hierarchies, 28, 97
relationship, 94, 109
Parsing, 164
Partitioning, 193
by column, 57
Hash, 250
moving, 197
range, 194
round-robin, 194
strategies, 193
tables, 196, 337
two level, 197
Payment-related functions, 86
Pearson correlation, 270, 277
Performance
tools, 205
trace, 205, 208, 212, 217
Permanent table, 246
temporary data, 244
Placeholders, 77

Q
Quarter-to-date (QTD), 133
Query
performance, 205
reductions, 331
query results cache, 247
QUERY_NODE, 101
QUERY_NODE_NAME, 101

R
R, 24, 25, 82, 83, 319, 324, 361
multiple servers, 333, 335
packages, 85, 319
Random
noise, 350

Polyomial regression, 89
Portfolio, 341, 342
equally distributed, 341
ID, 355
optimization process, 363, 365
return, 342
Predictive algorithms, 15, 264
Predictive analysis, 88
applications, 21
Predictive Analysis Library (PAL) → PAL
Predictive analytics lifecycle, 259
Predictive data, 291
Predictive modeling
advanced, 319
basic, 259
data partitions, 332
tasks, 297
Preprocessing algorithms, 89, 324
Preprocessor server, 21
Price version, 173, 174
Primitive types, 324
Principal Component Analysis (PCA), 289, 291
Privilege validity, 68
Programming languages, 75
Projection node, 56, 61
Purchase cost, 178
Python, 23

Index
Models, 263, 274, 327
creation and maintenance, 299
scoring, 300
validation, 299
Modify, 263
Monte Carlo simulation, 342
Month-to-date (MTD), 133
Multidimensional expression (MDX), 102
Multidimensional reporting, 45
Multilevel partitions, 332
Multiple
linear regression, 89
Parent, 28, 103
Multithread processing, 329
Multivariant statistical analysis, 287
Multivariate statistics, 366

N
Naive Bayes, 89
Name
only, 28
path, 28
server, 21
Navigation function, 102, 106, 107
Nested calculation view, 185
Nodes, 28, 106
Nominal or classificatory scale, 291
Nonequi joins, 224, 230
operators, 231
Non-stepwise simulation, 358
Normal distribution, 344
Normalizing by decimal scaling, 292
NOT EXISTS clause, 170
NOT IN clause, 169

O
Object privileges, 69
OLAP engine, 24, 52, 224
Optimization, 341, 364
applications, 21
constraints and equations, 370
model, 371, 379

Random (Cont.)
numbers, 344
walk, 326
Random distribution, 343
shape, 344
Random variable, 292
generation, 343
Randomization, 343
Range, 194
test, 284
Rank
filtering data, 134
comparison, 182
partitions, 332
Real-time
cache, 252
mode, 252
model consumption, 329
model execution, 329
Recommendation, 312
Recursion, 93
Recursive logic, 93
Reference models, 68
Referential join, 40, 41, 59, 138
Regression, 298, 327, 333
Relational optimization, 45, 47
Relevance table, 290
Reliability, 355
Repository procedure, 72
Result cache, 251
RESULT_NODE, 101
RESULT_NODE_NAME, 102
resultcache_clear.reconfig, 251
resultcache_enabled, 251
resultcache_maximum_value_size_in_bytes, 251
resultcache_minimum_query_execution_time_in_milliseconds, 252
resultcache_request_timeout_in_milliseconds, 251
Retention, 254, 256, 257
Return, 364
Return-to-risk, 364

Optimizer (Cont.)
strategies, 48
OR operator, 140
Ordinal or ranking scale, 291
Outer join, 40, 138

P
Package privilege, 69
PAL, 25, 63, 86, 88, 259, 268, 319, 333, 343, 366
architecture, 334
calling a procedure, 90
function categories, 88
model consumption, 326
scaling, 335
stratification, 321
time series functions, 300
using R, 337
Parallel
coordinates, 317
execution, 82
processing, 331, 361
Parent, 28, 103
Parent-child
hierarchies, 28, 97
relationship, 94, 109
Parsing, 164
Partitioning, 193
by column, 57
Hash, 250
moving, 197
range, 194
round-robin, 194
strategies, 193
tables, 196, 337
two level, 197
Payment-related functions, 86
Pearson correlation, 270, 277
Performance
tools, 205
trace, 205, 208, 212, 217
Permanent table, 246
temporary data, 244
Placeholders, 77

Q
Quarter-to-date (QTD), 133
Query
performance, 205
reductions, 331
query results cache, 247
QUERY_NODE, 101
QUERY_NODE_NAME, 101

R
R, 24, 25, 82, 83, 319, 324, 361
multiple servers, 333, 335
packages, 85, 319
Random
noise, 350
Index

TOP/LIMIT, 216
Top-down forecasting, 331
Trace configuration, 208
Training reductions, 331
Transparent filter, 57
Transposition, 110
columns to rows, 110, 115
rows to columns, 116
Trend, test, 284
Triple exponential smoothing, 301

U
UL, 76
Uniform distribution, 343
Union, 104, 114, 144, 223, 225
node, 54, 147
with constant values, 218
UnionAll, 104
Unique index, 201
Univariate statistical analysis, 274
Unsupervised
clustering, 327
learning, 269, 321, 327, 329, 335
Upper quartile, 274
UTC timestamp, 237, 239

V
Value-at-risk, 342, 351, 356
randomization, 343
Variance, 274
Variance-covariance matrix, 365
Vector machine, 302
Vertical union, 161
performance, 162
View cache, 254
Visualize plan, 207
Volatility, 355, 364

W
Weibull distribution, 346
Weighted score table, 90
Weights, 377, 379
White noise test, 269, 285
Window functions, 131, 134, 135, 176, 233
complex grouping, 172
LEAD and LAG, 128
missing values, 171
Wrapper function, 266

X
XS engine, 21
xterm, 210
X-Windows, 210

Y
Year-to-data (YTD), 131

Z
Z-score normalization, 292
Anil Babu Ankisettipalli has more than 17 years of experience in developing application and technology software. Since 2010, he has designed and developed SAP HANA applications and products using predictive analytics, machine learning, and data mining algorithms. At SAP Labs, he uses data science and big data technologies within the SAP HANA platform to solve critical problems of SAP customers.

Hansen Chen is an experienced database expert and graduate of Peking University in China. In 2006, he joined Business Objects, where he worked on Web Intelligence and the semantic layer. He joined the SAP Strategic Customer Engagements group as a senior specialist in 2011, and worked with customers in co-innovation projects on top of the SAP HANA platform. His specific area of SAP HANA focus is on modeling and performance tuning.

Pranav Wankawala has more than 12 years of experience in software engineering and developing enterprise applications. He started working on SAP HANA in 2009, when it was still a research project, and designed and architected one of the first SAP products that ran on the platform. Today, he leads a team of highly skilled engineers and technologists that help SAP customers improve their day-to-day business with SAP HANA.

We hope you have enjoyed this reading sample. You may recommend or pass it on to others, but only in its entirety, including all pages. This reading sample and all its parts are protected by copyright law. All usage and exploitation rights are reserved by the author and the publisher.