

# Our Experience Converting an IBM Forecasting Solution from R to IBM SPSS Modeler



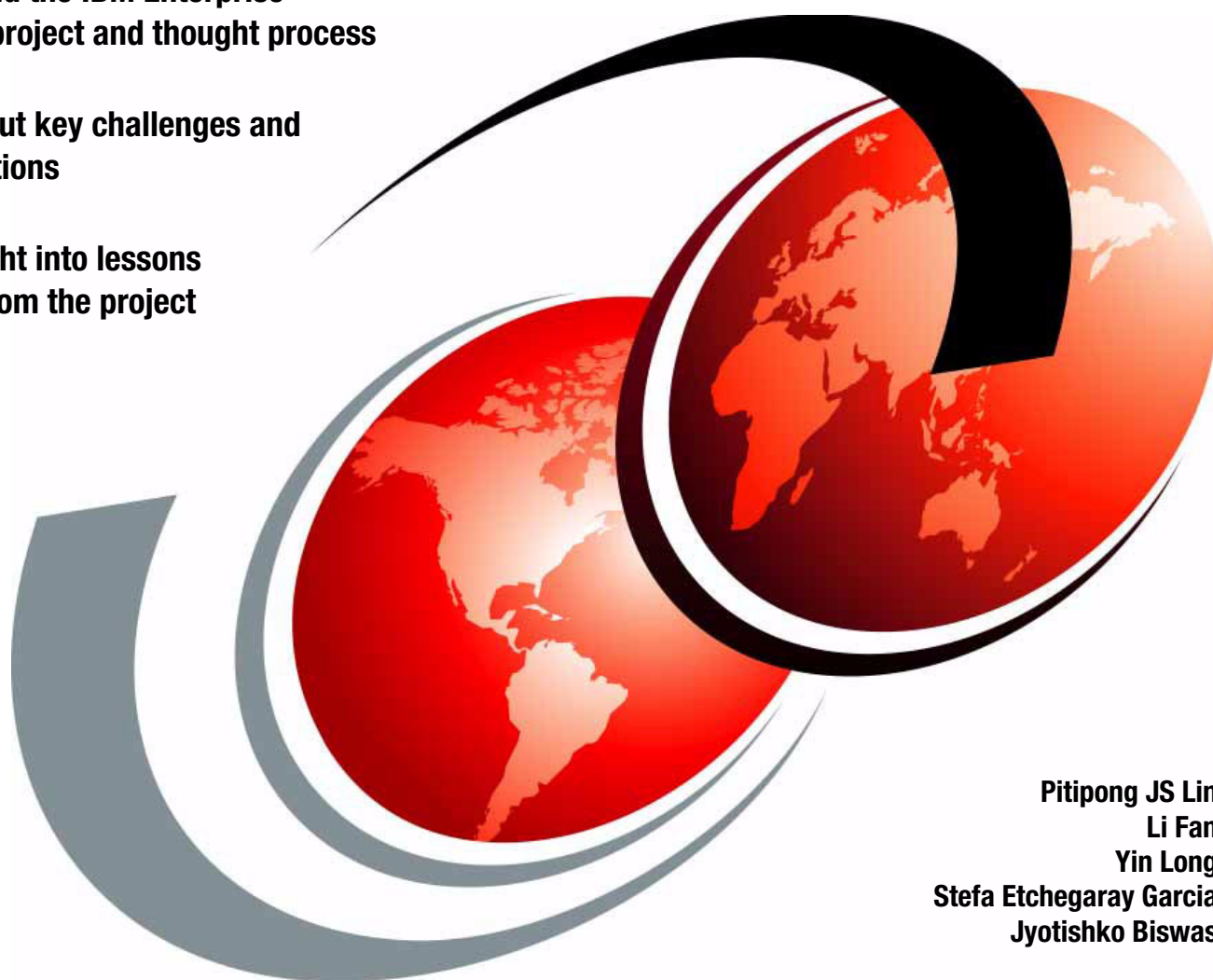
Understand the IBM Enterprise Services project and thought process



Learn about key challenges and their solutions



Gain insight into lessons learned from the project



Pitipong JS Lin  
Li Fan  
Yin Long  
Stefa Etchegaray Garcia  
Jyotishko Biswas





International Technical Support Organization

**Our Experience Converting an IBM Forecasting  
Solution from R to IBM SPSS Modeler**

March 2015

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**First Edition (March 2015)**

This edition applies to IBM SPSS Modeler Server V16.0 (5725-A65) and IBM SPSS Modeler Desktop V16.0 (5725-A64).

**© Copyright International Business Machines Corporation 2015. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
Authors .....	vii
Now you can become a published author, too! .....	viii
Comments welcome .....	ix
Stay connected to IBM Redbooks .....	ix
<b>Chapter 1. Introduction to the domain and the project</b> .....	1
1.1 Business background: Predictive analytics for Sales Transaction Support. ....	2
1.1.1 About the R-based solution and facing long-term viability .....	3
1.2 SPSS Modeler and R comparison .....	4
1.2.1 Data preparation .....	5
1.3 Architecture of the SPSS Modeler-based solution .....	6
1.3.1 Iteration key to solution development .....	10
<b>Chapter 2. Key challenges and resolutions</b> .....	13
2.1 Introduction .....	14
2.2 Switching between forecasting hierarchies .....	14
2.2.1 Finding and using the hierarchy structure dynamically .....	15
2.2.2 Setting SPSS nodes and parameter setup .....	15
2.2.3 Results .....	16
2.3 Handling dynamic streams .....	16
2.3.1 Challenge and approach .....	16
2.3.2 Setting SPSS nodes and parameter setup .....	17
2.3.3 Results .....	19
2.4 Supporting array functionality .....	19
2.4.1 Challenge and approach .....	20
2.4.2 Setting SPSS nodes and parameter setup .....	21
2.4.3 Results .....	24
2.5 Defining variable types .....	25
2.5.1 Challenge and approach .....	25
2.5.2 Setting SPSS nodes and parameter setup .....	25
2.5.3 Results .....	26
2.6 Incorporating automated exception handling .....	27
2.6.1 Challenge and approach .....	28
2.6.2 Setting SPSS nodes, flow chart, and parameter setup .....	28
2.6.3 Results .....	30
2.7 Supporting variable selection .....	31
2.7.1 Challenge and approach .....	32
2.7.2 Setting SPSS nodes and parameter setup .....	33
2.7.3 Results .....	36
2.8 ARIMA and Fourier ARIMA model conversion .....	36
2.8.1 Challenge and approach .....	38
2.8.2 Setting SPSS nodes, flow chart, and parameter setup .....	39
2.8.3 Results .....	47
2.9 AGG: Ensemble models and automatic model selection .....	48
2.9.1 Challenge and approach .....	49

2.9.2 Results .....	58
2.10 Setting parameters .....	59
2.10.1 Challenge and approach .....	59
2.10.2 Setting SPSS nodes and parameter setup .....	60
2.10.3 Results .....	61
<b>Chapter 3. Lessons learned</b> .....	63
3.1 Planning an R to IBM SPSS conversion project .....	64
3.2 Key decisions .....	64
3.3 Risks and mitigation to consider .....	65
3.4 Calling R functions directly .....	66
<b>Related publications</b> .....	67
IBM Redbooks .....	67
Online resources .....	67
Help from IBM .....	67

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

**The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

## COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

# Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Global Business Services®  
IBM®  
Redbooks®

Redpaper™  
Redbooks (logo) ®  
SPSS®

Watson™

The following terms are trademarks of other companies:

Other company, product, or service names may be trademarks or service marks of others.

# Preface

This IBM® Redpaper™ publication presents the process and steps that were taken to move from an R language forecasting solution to an IBM SPSS® Modeler solution. The paper identifies the key challenges that the team faced and the lessons they learned. It describes the journey from analysis through design to key actions that were taken during development to make the conversion successful.

The solution approach is described in detail so that you can learn how the team broke the original R solution architecture into logical components in order to plan for the conversion project. You see key aspects of the conversion from R to IBM SPSS Modeler and how basic parts, such as data preparation, verification, pre-screening, and automating data quality checks, are accomplished.

The paper consists of three chapters:

- ▶ Chapter 1 introduces the business background and the problem domain.
- ▶ Chapter 2 explains critical technical challenges that the team confronted and solved.
- ▶ Chapter 3 focuses on lessons that were learned during this process and ideas that might apply to your conversion project.

This paper applies to various audiences:

- ▶ Decision makers and IT Architects who focus on the architecture, road map, software platform, and total cost of ownership.
- ▶ Solution development team members who are involved in creating statistical/analytics-based solutions and who are familiar with R and IBM SPSS Modeler.

## Authors

This paper was produced by a team of specialists from around the world working with the International Technical Support Organization.

**Pitipong JS Lin** Ph.D. is a Senior Technical Staff Member (STSM) in IBM Enterprise Services (ES) in the US. His expertise is Business Analytics and Reverse Supply Chain. He is a certified lean sigma black belt, business consultant, and project manager. He has end-to-end experience working across supply chain functions in IBM and externally with clients.

**Fan Li** is an IBM Advanced Data Analyst in China. He has 12 years of experience in the data analyst and data mining fields. He has worked at IBM for 10 years. His areas of expertise include data analysis with SPSS and R, focusing on data analysis solution development.

**Yin Long** Ph.D. is an IBM Advanced Analytics Analyst in IBM Singapore. She has four years of research experience and three years of industrial experience in data analysis and optimization. She holds a Ph.D. degree in Operations Research from National University of Singapore. Her areas of expertise include optimization, simulation, and forecasting.

**Stefa Etchegaray Garcia** Ph.D. is an IBM Research Staff Member at the IBM TJ Watson™ Research Center in New York. She has 10 years of experience in statistical analysis, modeling, and theory. She holds a Ph.D. in Statistics from Carnegie Mellon University. Her areas of expertise include time series analysis and forecasting and their applications to problems in business analytics. She has written extensively about the statistical challenges to analyze astronomical data.

**Jyotishko Biswas** is an IBM Advanced Analytics Professional in India. He has eight years of experience in the advanced analytics field. He holds a Masters degree in Economics from Jawaharlal Nehru University. His areas of expertise include predictive modeling, forecasting, and text mining in fields of supply chain and banking. He has written extensively about advanced analytics solutions for supply chain.

Thanks to the following people for their contributions to this project:

Rufus Credle, LindaMay Patterson  
International Technical Support Organization, US

Aliza Heching  
IBM Research, US

Steve Bayline  
IBM Enterprise Services, US

Zhong Shi Wang  
IBM Global Business Solution Center, China

Alex Reutter  
IBM Information Developer, US

Patrick Gibney  
IBM Enterprise Services Consultant, US

Yu Ying Wang  
IBM Global Business Services, China

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks® publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>





# Introduction to the domain and the project

This chapter provides the background for the R to IBM SPSS Modeler conversion project. This chapter contains the following topics:

- ▶ “Business background: Predictive analytics for Sales Transaction Support” on page 2
- ▶ “SPSS Modeler and R comparison” on page 4
- ▶ “Architecture of the SPSS Modeler-based solution” on page 6

## 1.1 Business background: Predictive analytics for Sales Transaction Support

Sales Transaction Support (STS) is an organization in IBM Enterprise Services (ES) that consists of Sales Support and Customer Fulfillment teams. Together, they handle a high volume of transactions daily, including helping to develop a proposal, qualifying a reference during the pre-sales process, and managing the post-sales order and invoice process.

To help the Sales Support and Customer Fulfillment teams, the ES development team created an analytics-based solution to enable a single seller touch-point process that covers the life of a transaction. For IBM sellers and IBM Business Partners, this improvement meant spending less time with sales support activity, reduced cycle time for sales transactions, and fewer handoffs. For clients, this approach reduces the overall cycle time from opportunity through order processing and consistency for all paperwork, including quotes, contracts, and invoices.

The solution models the demand (requests from the IBM Sales team, such as creating the proposal, quoting the price, and writing the contract) for Sales Support resources as a supply chain. This approach differs from classic supply chain models and introduces a number of new and interesting problems. The goal of the project was to balance demand versus supply, requests, and resources, and to ensure that the Sales Support personnel are being deployed and used strategically to meet revenue targets.

Initially for this solution, we used an IBM Research workforce analytics solution. This solution was based on the R language forecasting solution. The solution had the following capabilities:

- ▶ Model business processes to identify factors that drive complexity
- ▶ Forecast demand and transaction volumes by incorporating a complexity index
- ▶ Match skill requirements with forecasted demands and complexity
- ▶ Plan for peak and non-peak workload periods
- ▶ Optimize resource plans to increase productivity and reduce overtime, redundancy, and rework

The solution used IBM Research analytics to optimize the performance of the IBM Sales Support and Customer Fulfillment Centers. The IBM Research solution consisted of a suite of models and tools for resource/skill demand forecasting, capacity planning, gap/glut analysis, and long-term strategic planning.

A key aspect of this solution was the hierarchical demand forecasting that analyzes the number of planning hierarchies that must be considered. The workload is managed for various types of requests. The forecasts are produced at the request-type level, that is, different requests require different skills and take different amounts of time. The users can aggregate this data to see the total volume of requests. Forecasts are produced at the country level, but you can aggregate forecasts to see the total volume of requests. For example, teams can support multiple countries, and a center sees the total volume of requests that flow through that center.

ES Analytics uses hierarchical forecasts to ensure that the forecasts are consistent at all levels of the hierarchy and to determine the best level at which to produce the forecast to maximize accuracy.

### 1.1.1 About the R-based solution and facing long-term viability

In collaboration with IBM Research, IBM ES developed a workforce analytics solution that uses advanced forecasting models. This solution was originally developed by using the R programming language and its statistical computing and graphics capabilities.

The workforce analytics solution combines capabilities that help an organization plan its resources. A wide range of analytics capabilities were used:

- ▶ Descriptive analytics: Visualization of current and historical data through a dashboard
- ▶ Predictive analytics: Forecasts of future workload demand
- ▶ Prescriptive analytics: Optimization of the supply and demand of resources

Development of the R solution consisted of agile, smaller steps toward a larger goal. Development work was performed iteratively, receiving feedback from the users to improve the model development and gain user acceptance and adoption.

Underpinning the development of the forecasting solution are the business impact and benefits. Positive feedback and documented benefits led to further development of the forecasting solution to expand to more varied and complex deployment scenarios and improve forecasting accuracy. A direct result of this extension was the need for a long-term strategy that was based on factors, such as cost, maintainability, and the core competency of the analytics team.

IBM SPSS Modeler is a comprehensive workbench-style statistical modeling software. SPSS Modeler has excellent data handling capabilities that enable the use of large data sets. In addition, SPSS software can carry out complex analyses with high levels of reliability and stability. Many of the individual modeling algorithms in SPSS Modeler are widely available. These advantages and other features of SPSS Modeler made it a natural platform on which to further develop and ultimately deploy the workload forecasting solution.

#### Key decision factors

An important part of the decision process to move the R-based solution into SPSS Modeler was the cost of maintenance. SPSS Modeler is an integrated statistical tool that allows users to carry out complex statistical analyses without having to maintain a complex programming language. SPSS Modeler provides an intuitive interface with a visual-style workbench that allows the analytics team the flexibility to perform knowledge transition and keep maintenance costs low. A key long-term maintenance requirement included a biannual calibration of the forecasting solution. Changes in business trends, data quality issues, and other real-world settings require periodic calibration and adjustment to the code. Finally, SPSS Modeler includes a wide range of descriptive analytics and automated modeling tools that can be integrated with custom extensions. The transition to IBM SPSS Modeler enabled the effective use of the team's analytics skills that helped create a flexible work environment within the organization.

## Evaluation considerations

Multiple points of concern surfaced when we evaluated the technical feasibility of moving the R-based solution into IBM SPSS Modeler:

- ▶ Selecting a statistical software platform

The team considered several options, such as SPSS Modeler with Python, IBM SPSS Statistics, or a combination of SPSS Modeler and SPSS Statistics. The objective was to keep the suite of tools simple while being able to handle the complexity of the solution. SPSS Modeler Version 16 can call R code directly; however, it was not a preferred option because it still requires the organization to develop skills in both R and SPSS competencies.

- ▶ Transitioning from the existing R-based solution in production

The solution on the new software platform needs to yield similar results to those results that are obtained in R while maintaining a seamless transition to the users of the existing solution.

- ▶ Intrinsic capabilities of the R language

The R-based solution used for-loops, arrays, and exception handling. These same capabilities need to be replicated in the new software. These automated control-type capabilities are tricky to implement in SPSS Modeler.

- ▶ Identifying subject matter experts (SME) and reference literature

The original development approach applied a research and development mentality with the use of both academic and commercial literature to enable a state-of-the-art solution. During the original development, SMEs were enlisted for forecasting expertise. Experts or reference material for the transition was not readily available. The transition proved to be a unique challenge for the team that worked on the original project and worked on converting the forecasting solution from R to SPSS Modeler.

## 1.2 SPSS Modeler and R comparison

R is an open source functional programming language that is designed for statistical computing and graphics. New methods for a wide range of statistical applications, such as analytics, are constantly being developed and added to the R repository. R users can use newly developed tools as add-on packages quickly after the tools are developed. The main disadvantage to this quick turnaround is that software and algorithms might not be thoroughly tested before they are put into production.

SPSS however implements methods after thorough testing, and emphasis is put on procedures that are formalized and ready to publish. A downside to publication quality output is that it is harder to use as input for further analysis.

Both R and SPSS Modeler have command-line interfaces, but only SPSS has a menu-driven user interface. Macros and scripts to automate calling procedures can be written in both the R language and SPSS syntax. In both cases, writing user-based functions requires programming knowledge to ensure the required performance and accuracy.

The SPSS Modeler's data editor allows you to enter your data and the attributes of your data, such as variable type, name, missing values, and value labels. The data input management system in SPSS Modeler helps in reading, organizing, and transforming data. For example, data files can be reshaped and automated by using scripts. SPSS Modeler primarily edits one data file at a time, and no limit exists to the number of variables and cases that are allowed in the data file. The limitation is set by the amount of available disk space. With R however, multiple data sets can be handled because the workspace (where variables and data are stored while working) is in the random access memory (RAM). Data management in R is not as "clean" because many different types of data structures can make the data input process difficult.

### 1.2.1 Data preparation

R and SPSS are two statistical software packages that are designed specifically for statistical computing and graphics. Macros and scripts to automate calling procedures can be written in both the R language and by using SPSS syntax. Writing user-based functions in both R and SPSS Modeler requires extensive programming knowledge to ensure the required performance and accuracy.

This section describes the main differences when you use SPSS Modeler and R to prepare data for analysis.

#### Input files

The SPSS Modeler data editor allows the user to enter data and data attributes, such as variable type, name, missing values, and value labels. The data input management system in SPSS Modeler helps in reading, organizing, and transforming data. SPSS Modeler primarily edits one data file at a time, and the number of variables and cases that are allowed in the data file are not limited. The limitation is set by the amount of available disk space.

Multiple data sets can be handled in the R workspace (where variables and data are stored when working), which in turn is in RAM. Data management in R is not as "clean" as in SPSS. Because many types of data structures exist, the data input process can be difficult.

In SPSS Modeler, source nodes (that is, var.files) can be used to import data and edit data attributes. For example, if the default variable storage for a date field is string, a node can be used to override the storage from string to date. Another advantage is that input format can be customized. For example, if the date format in an input data file is *MM/DD/YYYY*, a source node can be used to change the date format in SPSS Modeler to *YYYY-MM-DD*.

#### Types of variables

Continuous data and categorical data are different objects when statistical methods are used in R. A categorical variable in R is known as a factor and factors are stored as integer codes with a set of labels. After the data is loaded, consider whether the data needs to be treated as continuous or categorical. Use re-coding or coercing by using the `factor()` function.

In SPSS Modeler, fields are assigned a measurement level that is based on their storage in a source node. Integer, real, and date fields are assigned the measurement level "Continuous", and string fields are assigned the "Categorical" measurement level. To change the measurement level of data, you can use a Type node, which, in turn, allows procedures to use the various types of data correctly.

## Aggregating and reshaping

Aggregating data in R can be done quickly with the `aggregate()` function. One caveat to using this function is that input must be a data frame object. Data can be collapsed by using a simple function, such as `sum` or `built-in`, or by using user-defined functions. *Reshaping data* refers to structuring data and aggregating it so that certain row entries become column entries and vice versa. The `reshape` R package is a powerful tool that allows for restructuring of data. Reshaping large data sets can be computationally expensive because the data is stored and modified in RAM.

Data can be aggregated in SPSS Modeler by using the Aggregate node. The default methods of aggregation include `sum`, `mean`, `minimum`, `maximum`, `standard deviation`, `median`, `count`, `variance`, and `first and third quartiles`. Custom methods of aggregations are also possible.

The Restructure node in SPSS Modeler can be used to create new fields from one or more categorical fields. However, converting column entries to row entries is not straightforward in SPSS Modeler. You cannot use a single node to convert a row to a column because no corresponding build-in module exists in SPSS Modeler.

## Merging data sets

Merging two data frame objects in R can be done by using the `merge()` function. This function can perform `inner`, `outer`, and `left and right joins`.

In SPSS Modeler, the Merge node can merge two or more data frames. The Merge node can perform `inner`, `outer`, `partial (left or right)`, and `anti joins`.

## Adding the time dimension

Basic time series analysis capabilities in R are included, by default. The time series object `ts()` provides infrastructure for ordered observations that store additional attributes, such as a `time` and `frequency`.

Time Interval node in SPSS Modeler can be used to build time series. You can set the time interval as `year`, `month`, `quarter`, `week`, and so on. Moreover, you can set a cyclic period. For example, `10 periods` is a cycle. This setting is necessary when you build a time series model with seasonal components, for example.

# 1.3 Architecture of the SPSS Modeler-based solution

We chose IBM SPSS Modeler to implement the R-based forecasting solution because SPSS Modeler provides a versatile analytics workbench with similar capabilities to R. Building forecasts repeatedly in SPSS Modeler requires no programming because of the efficient graphical interface. However, development of the custom node for the graphical interface requires extensive programming knowledge. Special care must be taken because specific R control capabilities cannot be replicated exactly. An advantage is that SPSS Modeler includes advanced time series capabilities that can be modified and customized, as needed.

Figure 1-1 on page 7 shows the solution components with the forecasting engine as the core analytics engine.

The solution architecture is flexible and scalable enabling quick implementation across multiple geographies and business towers

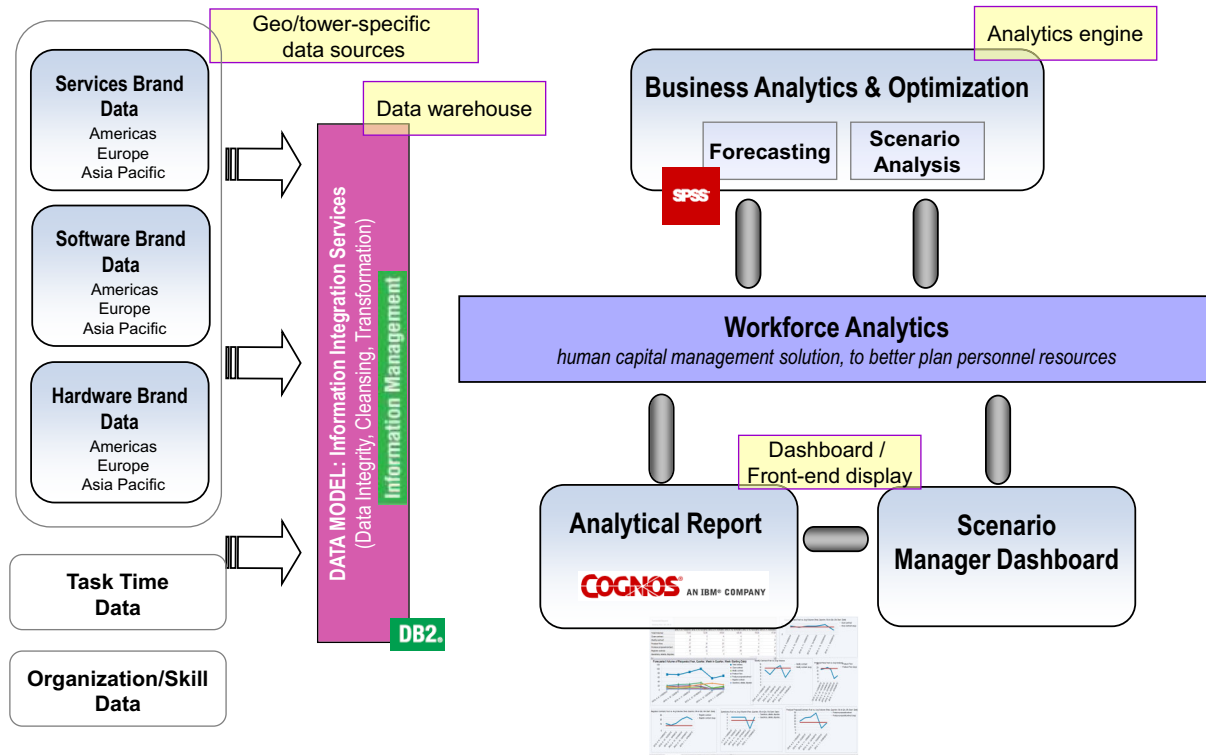


Figure 1-1 Solution architecture

Big data provides key sources of input from disparate existing systems, such as request tools and workflow systems. Due to the diverse products that ES STS supports (hardware, software, and services), the team focused on consolidating and processing the data for downstream business analytics engines. The development team incorporated a dashboard visualization to allow clients or users in the worldwide transaction centers real-time access. These individuals benefited from seeing the information in real time.

In preparation for converting the solution from R to a new statistical software platform, the development team looked at ways to dissect a large complex set of code into smaller manageable modules. The development team arrived at three logical modules:

- Data processing
- Model build
- Forecasting

The solution's logical design modules are shown in Figure 1-2 on page 8.

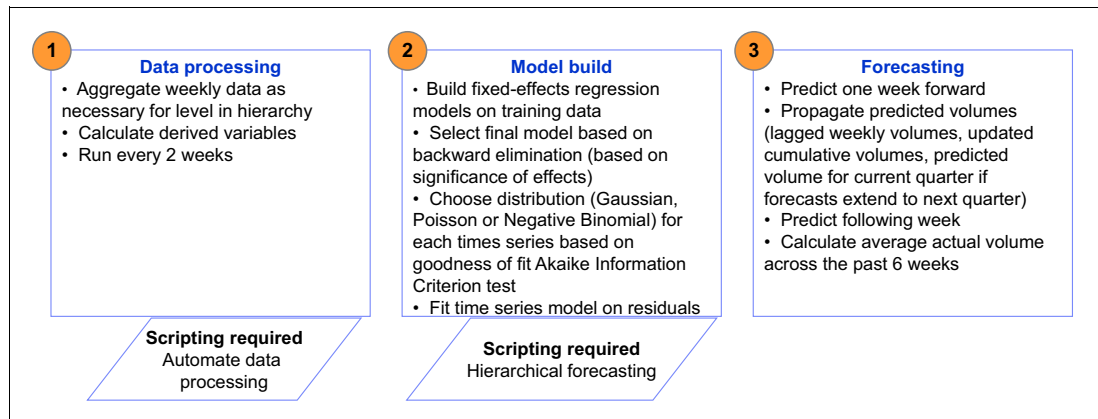


Figure 1-2 Solution logical design modules

By creating these logical design modules, the development team was able to start exploring the detailed design and complexity of the solution.

A design point that quickly surfaced was the need to use programming scripts with SPSS Modeler in data processing and model build. However, it was not apparent that SPSS Modeler was able to run an array or a for-loop, for example.

The team considered using SPSS Modeler with IBM SPSS Statistics or with Python to enable this critical capability. The design requirement forced the team to analyze the function routine further in each solution component. It was a key decision to use SPSS Modeler for its intuitive workbench, while resorting to Statistics or Python, which then reintroduced complexity in long-term code maintenance. Figure 1-3 shows the numerous challenges in the conversion of R to SPSS Modeler that we encountered in the drill-down design.

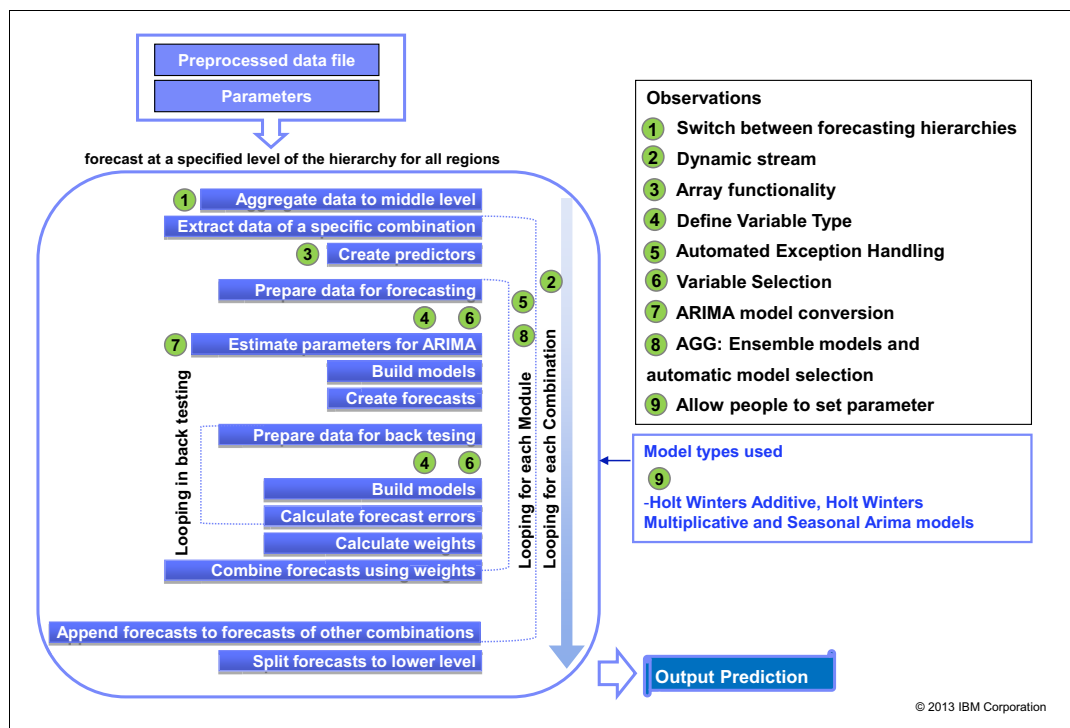


Figure 1-3 SPSS Modeler solution components

Table 1-1 details each observation that was identified during the mapping of the R solution to an SPSS Modeler solution. Chapter 2, “Key challenges and resolutions” on page 13 describes these observations and their resolutions in detail.

*Table 1-1 Observations mapped between R and SPSS Modeler*

Observation	R implementation	SPSS Modeler implementation	Remarks
Switch between forecasting hierarchies	Use loops and functions in R code	Use SPSS script	<ul style="list-style-type: none"> <li>► Hierarchical forecasting needs to build models at different levels</li> <li>► Requires complex scripting to implement in SPSS</li> <li>► Results comparable</li> </ul>
Handling dynamic streams	Use only scripting	Use SPSS scripts and nodes	<ul style="list-style-type: none"> <li>► SPSS can provide the same results as R</li> <li>► Results comparable</li> </ul>
Supporting array functionality	Arrays used frequently in R code	<ul style="list-style-type: none"> <li>► SPSS script has limited array capability</li> <li>► Use field operation nodes</li> <li>► Use the Expression Builder to provide abundant array functions</li> </ul>	<ul style="list-style-type: none"> <li>► Easy to derive fields in SPSS</li> <li>► Easy to understand SPSS node</li> <li>► SPSS provides better results</li> </ul>
Define variable types	Problem is not significant in R. R is flexible in defining variable types	<ul style="list-style-type: none"> <li>► In SPSS, the variable type must be defined in Type node before building a model.</li> <li>► Used script to automatically define variable type in Type node.</li> </ul>	<ul style="list-style-type: none"> <li>► Defining variable type correctly is important to build the correct solution</li> <li>► Manually updating Type node for every Center, Line of Business (LOB), Country, and Request Type combination is not practical. Automated this activity by using scripting</li> <li>► Results comparable</li> </ul>
Automated exception handling	<ul style="list-style-type: none"> <li>► Strong exception handling functionality in R</li> <li>► Certain R tools, such as try(), allow the program to take specific actions when a condition occurs</li> </ul>	Included model development inside the SuperNode. When an execution inside the SuperNode fails, looping of the main stream continues	<ul style="list-style-type: none"> <li>► No automated exception handling mechanism in SPSS</li> <li>► Without SuperNode, if model development for one time series fails, SPSS stream cannot continue</li> <li>► Results comparable</li> </ul>

Observation	R implementation	SPSS Modeler implementation	Remarks
Support variable selection	<ul style="list-style-type: none"> <li>▶ An algorithm based on backward elimination is developed in R</li> <li>▶ Used statistical tests and corresponding p-values for variable selection in R</li> </ul>	<ul style="list-style-type: none"> <li>▶ Difficult to use Statistical tests and corresponding p-values for variable selection in SPSS</li> <li>▶ Used feature selection node to select predictors automatically</li> </ul>	<ul style="list-style-type: none"> <li>▶ Comparable forecast accuracy in SPSS</li> <li>▶ Easy to understand and execute in SPSS</li> <li>▶ SPSS provides better results</li> </ul>
Auto regressive integrated moving average (ARIMA) and Fourier ARIMA model conversion	<ul style="list-style-type: none"> <li>▶ An algorithm based on exhaustive searching is developed in R to select the best ARIMA model</li> <li>▶ Call function spectrum() for spectrum analysis in R</li> </ul>	<ul style="list-style-type: none"> <li>▶ Used ARIMA Expert Modeler that is present in the Time Series node to build seasonal ARIMA model automatically</li> <li>▶ Call R or SPSS Statistics for spectrum analysis</li> </ul>	<ul style="list-style-type: none"> <li>▶ No direct conversion for spectrum analysis in SPSS Modeler</li> <li>▶ SPSS results are as good as R</li> <li>▶ Use SPSS Time Series node, which is easy to implement</li> <li>▶ Results comparable</li> </ul>
Ensemble models. Why not automatic model selection?	<ul style="list-style-type: none"> <li>▶ Ensemble models built</li> <li>▶ Ensemble models reduce over-fitting and offer higher accuracy</li> <li>▶ Weights of forecasts from different models based on their forecast accuracy</li> </ul>	<p>Ensemble models built.</p> <ul style="list-style-type: none"> <li>▶ Weights of forecasts from different models based on their forecast accuracy</li> </ul> <p>SPSS Automated model selection selects a better-fitting model, which can result in over-fitting, therefore, we did not use it</p>	<ul style="list-style-type: none"> <li>▶ AGG accuracy higher than previous models</li> <li>▶ SPSS solution's accuracy is close to R solution accuracy</li> <li>▶ AGG quickly adapts to business change</li> <li>▶ Easy to deploy AGG to new businesses</li> <li>▶ Lesser cost of revisiting models</li> </ul> <p>Results comparable</p>
Allow people to set parameter	R code of the Transaction Center Optimization (TCO) project hardcodes the parameters	<ul style="list-style-type: none"> <li>▶ In interactive mode of SPSS execution, enter the parameter by using a pop-up dialog.</li> <li>▶ In batch mode, enter the parameter by using a text configuration file.</li> </ul>	<ul style="list-style-type: none"> <li>▶ SPSS Modeler has facilities for a user to enter a parameter in both execution modes</li> <li>▶ Each parameter must be a single value; a vector or set is not allowed.</li> </ul> <p>SPSS gives better results</p>

### 1.3.1 Iteration key to solution development

The development approach was to create a new iterative development process that was based on working together with users of the solution. This engaged process was critical to our success. The speed of iteration was directly dependent on the feedback from the users.

This iterative development approach consisted of the following concepts:

- ▶ The development team engages directly with the users of the solution.
- ▶ Measure the key metrics that relate to the usage and understand the true value of the metrics.
- ▶ Modify the solution based on the metrics.

Developing a solution quickly by setting a goal/target and accelerating execution toward that end is not new. However, that approach might not suffice any longer. Iterating to create the solution, measuring the key performance metrics, and steering toward a point of value can help you arrive more quickly at the point of maximum value. Often, the end solution can differ from the original target.

The key to developing the workforce analytics solution was to set a large target that is aligned to the value, break the target into smaller manageable goals, and start with the highest value impact goals (Figure 1-4).

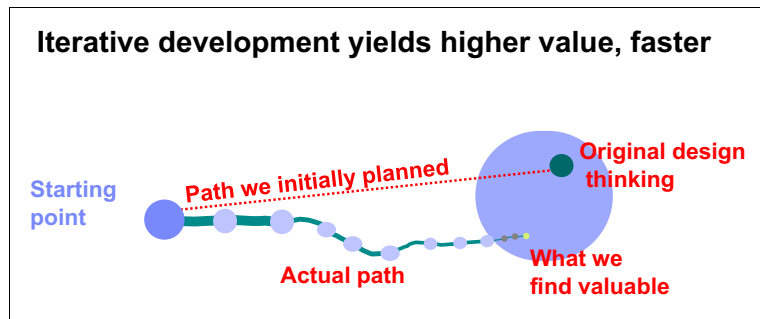


Figure 1-4 Iterative approach to solution development





## Key challenges and resolutions

This chapter describes the key challenges that were identified during the conversion from R to IBM SPSS Modeler of the forecasting solution. The challenges are covered in these sections:

- ▶ “Switching between forecasting hierarchies” on page 14
- ▶ “Handling dynamic streams” on page 16
- ▶ “Supporting array functionality” on page 19
- ▶ “Defining variable types” on page 25
- ▶ “Incorporating automated exception handling” on page 27
- ▶ “Supporting variable selection” on page 31
- ▶ “ARIMA and Fourier ARIMA model conversion” on page 36
- ▶ “AGG: Ensemble models and automatic model selection” on page 48
- ▶ “Setting parameters” on page 59

## 2.1 Introduction

This chapter provides a description of the challenges that the IBM Enterprise Services (ES) team faced when converting the forecasting solution from the R language to IBM SPSS Modeler. Each challenge and its resolution are described in a separate section. Each section includes the following topics:

- ▶ Overview: Introduction to and description of the challenge
- ▶ Example R code: Challenge and summary of conversion approach
- ▶ SPSS solution: Details about setting up SPSS nodes and parameters
- ▶ Result: Summary of the outcome with comparison to the original R implementation

## 2.2 Switching between forecasting hierarchies

Transaction centers perform all back-end sales activities across geographies and brands. Different types of back-end support are needed for different brands. Requests might arrive through various channels and from multiple geographical locations. Requests are disaggregated in a hierarchical structure by using attributes, such as request type, channel, and requesting country. For the different centers and brands, the hierarchy is not unique in the sense that each center/brand combination has unique attributes that impose the hierarchical structure for that specific combination. For example, the countries that request support for software transactions to the centers in Europe differ from those countries that request support for services to the centers in Asia.

The first common task when you deal with hierarchical data is the extraction of the entire hierarchical structure. Retrieving the full tree-like structure consists of finding all of the categories in the different attributes (for example, request types, channels, and country groups). After the hierarchy is extracted, forecasts are generated for each of the series in the hierarchy. In R, we iterate modeling and forecasting through all the levels of the hierarchy.

The example R code in Example 2-1 illustrates the extraction of the hierarchical structure for a data set of data. You retrieve the tree structure before you build a model. Next, a double loop is executed to fit the model to all of the nodes in the hierarchy. In this simple example, a hierarchical structure is extracted given the data. The hierarchy is not predefined, and for different data different structures, it will be obtained.

*Example 2-1 Example R code that extracts the levels of attributes for use in modeling*

---

```
INPUTS data = data.frame(attribute.1, attribute.2), model

a1.vec = unique(data$attribute.1)
a2.vec = unique(data$attribute.2)

for (a1.select in a1.vec) {
  for (a2.select in a2.vec) {
    EXTRACT the subset of data with attribute levels a1.select and a2.select
    FIT      the model to the subset of the data
  }
}
```

---

## 2.2.1 Finding and using the hierarchy structure dynamically

Extracting the levels of attributes, such as request types, channel, and geography, in a data frame in R and further looping through the different combinations by using a for-loop is straightforward. For different data sets, the unique combinations of attributes that define the nodes in a hierarchical tree might differ. Therefore, the tree structure for a certain data set must be learned or extracted from the data. The challenge in SPSS Modeler here is twofold. First, extract the hierarchical structure from the input data and then make it available for processing.

The Aggregate node was used to extract the unique combinations of the data attributes. The Aggregate node allows for extraction of the hierarchical structure by aggregating and recording the levels of the attributes that define the hierarchical levels. However, how to access the extracted hierarchical structure as a script variable in order to loop through the hierarchy when building models is a problem. The main difficulty is that the extracted structure is inside of the processing stream and extracting data from the stream during execution is not possible. To solve this availability problem, we used a Table node that creates and saves a list the levels of the attributes.

## 2.2.2 Setting SPSS nodes and parameter setup

Figure 2-1 shows an SPSS Modeler stream to extract and switch between the levels of a specific attribute (the request type). The same approach was used for the other attributes or hierarchical dimensions, such as channel and country group.

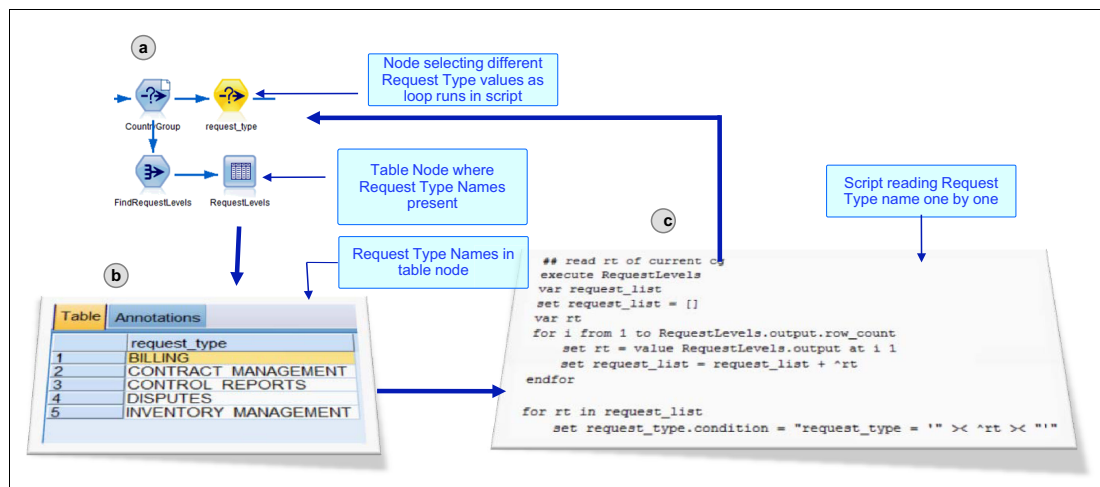


Figure 2-1 Switching between request types

Follow these steps to create the stream with the same functionality as Figure 2-1:

1. Set up nodes (Figure 2-1 item a). Add the following nodes:
  - A Select node that is named CountryGroup that selects data for each country group
  - An Aggregate node that is named FindRequestLevels following the CountryGroup node that extracts request types for each particular country group
  - A Table node that is named RequestLevels that is associated with FindRequestLevels that exports the request types to a table output object

2. Execute the Table node (Figure 2-1 item b).

Execute the Table node RequestLevels in the script, so that the request types are exported and included in the table output object.

3. Read request types (Figure 2-1 on page 15 item 3).

Read the request types from the table output object to a script variable. The script to read request types from the tape output objects is in Example 2-2.

*Example 2-2 Script to read request types*

---

```
set request = value RequestLevels.output at i 1
set request_list = request_list + ^request
```

---

### 2.2.3 Results

The SPSS stream can switch between forecasting hierarchies and perform exactly as the R code. However, this implementation is a bit more complicated than the R implementation.

## 2.3 Handling dynamic streams

Scripting in R allows for automated model building that can be reused and scaled up with little effort. The forecasting solution that was developed in R consists of a set of functions and routines that can be applied to data for different center and brand combinations. The data for a particular center and brand combination contains a specific hierarchical structure that differs from the data of a different center and brand combination. For example, the types of request might vary, as well as the channels and the requesting countries. The forecasting solution in R first learns the hierarchical structure of the data and then proceeds to build automated statistical forecasting models at each node of the hierarchy in the data.

Looping through the hierarchical dimensions of a data set can be used to build several models at the required nodes. Without looping, each model for a specific node needs to be built one at a time, which makes the task inefficient and time-consuming. Instead, the script is made to run while looping through different parameter settings, such as changing source data during execution and storing forecasts for different subsets of data.

### 2.3.1 Challenge and approach

A main challenge was to have SPSS Modeler traverse the hierarchical dimensions, such as the brand, geography, channel, and request type combinations, in order to use the correct subset of data for each node in the hierarchy. In SPSS Modeler, this task was a challenge because the task requires the stream to dynamically change the nodes and connections to handle the different combinations while the stream is executing. An additional complication was that while looping on the different levels of the hierarchical dimensions, different models can be considered.

An SPSS script was created and embedded in the stream to control the flow of the stream by extracting the hierarchy, iterating over the nodes of the hierarchy, and adjusting the model-fitting parameters. The script allows for the automated definition and control of the loop iterations across all the combinations in the hierarchy.

## 2.3.2 Setting SPSS nodes and parameter setup

The information about the built models is stored in a model list file. It includes the information about where the model is saved, which hierarchy node it is for, and whether it was successfully built. The file is read by the ReadModelList node. The script analyzes each row in the model list. If the required model is built, the build file is loaded as a nugget and inserted into the stream; otherwise, a DeriveNode that derives a value 0 is inserted instead.

Follow these steps to create a similar stream with the same functionality:

1. Create the static part of the stream manually where the nodes and connections will not be changed in the execution looping. The stream includes the data preparing nodes and the output of the forecasting results nodes. The model nuggets nodes and related supporting nodes are currently empty. The script adds these nodes during each execution of the loop (Figure 2-2).

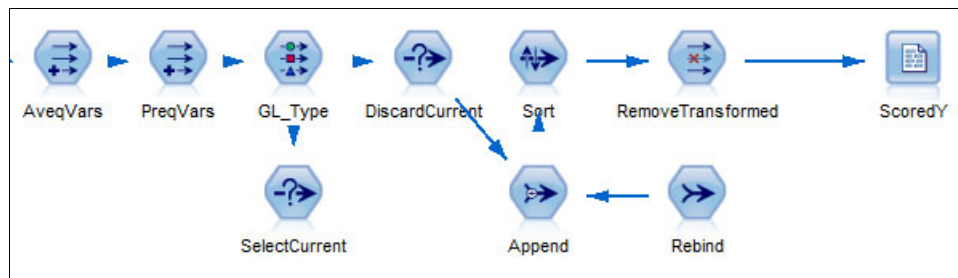


Figure 2-2 Forecasting stream before execution

2. Use the script in Example 2-3 to complete the stream before each loop is executed. For each combination, a different model is built. The SPSS script determines whether the model is successfully built. If successful, the correct model is loaded from the file into the palette and the model is inserted into the scoring stream.

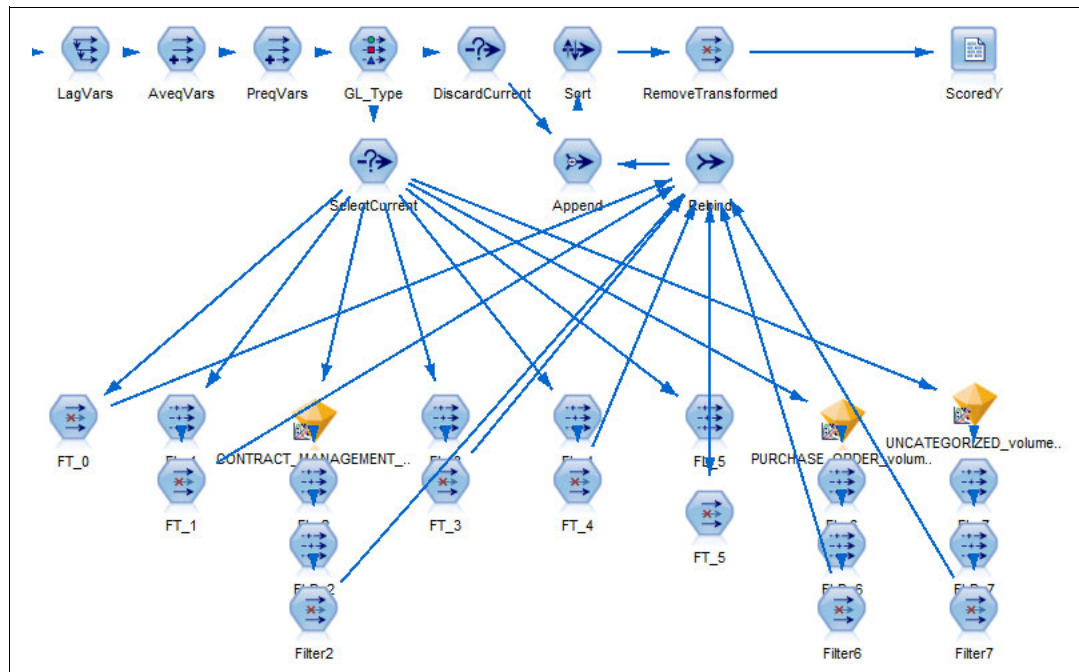
Example 2-3 Example R code to load each built model for prediction

---

```
for (g in 1:length(org.vec))
  for i from 1 to ReadModelList.output.row_count
    ...
    If ts_flag = 'YES' then
      set fullname = ^model_output_directory ><'/'><
        modelname>< '.gm'
      load model ^fullname
      insert model ^ts_target at 620 350+i*60
    else
      var mynode
      set mynode = create derivenode at 620 350+i*60
      rename ^mynode as ^ts_target
      set ^ts_target.new_name = '$TS-' >< ^ts_target
      set ^ts_target.formula_expr = 0
      endif
      connect ^lastnode to ^ts_target
    ...
    clear generated palette
  endfor
endfor
```

---

Figure 2-3 on page 18 showing the dynamic changes to the stream during execution.



3. The stream is cleared after each execution is finished by deleting all the nodes that were added by the script. This action is done so that during the next loop cycle, the script can add the expected model nugget nodes without tangling with the nugget nodes of previous loop iterations. A set in the script is used to store the node name each time that a node is added to the stream. After each execution of the loop, all the node names that are stored in the name set are deleted (Example 2-4).

*Example 2-4 Example R code for prediction and clean models added*

```
# remember the name of created node, for later clean
var n1
set n1 = []
for i from 1 to ReadModelList.output.row_count
  ...
  set modelname = 'GLM_' >< ^org >< '^' >< ^target
  set fullname = ^model_output_directory >< '/' ><
    ^modelname >< '.gm'
  load model ^fullname
  insert model ^target at 1080+i*100 400
  connect 'SelectCurrent' to ^target
  set n1 = n1 + ^target
  ...
  execute ScoredY

  for n in ^n1
    delete ^n
  endfor
endfor
```

Another type of dynamic stream changes the node settings or equations of the stream; otherwise, it is necessary to change the stream structure. For example, change the key fields of an Aggregate node according to the specific hierarchy level.

### 2.3.3 Results

SPSS streams can be modified by a script while the stream is executed. This modification allows an SPSS stream to perform complicated analysis operations that are equivalent to the R code.

## 2.4 Supporting array functionality

An important feature of R is its support for array calculations. In R, an array of values can be considered a variable and operations can be performed on this variable to modify it, expand it, or iterate through its values. The ability to handle array calculations makes R a powerful data processing and analytics tool.

In the forecasting solution, the data is mainly stored in the form of a data frame. A *data frame* is an array that can be used for general data processing functionalities. Example 2-5 shows an example R function that is used to create lagged variables from an input data frame, *dta*. The variables that will be lagged are indexed by *idx*, and the lag size is *M*. The derived lagged variables are appended to the original data frame as new variables and the modified data frame is returned. The *temp.NA* data frame is necessary because no lagged data will be available for the first *M* observations. The *match* function is used to determine the column location of the variables that will be lagged inside of the data frame. The *rbind* function is used to combine two objects by rows, in this case, an empty data frame (that is, a data frame with NAs) with the original data frame that is displaced by *M* units. The index operation *[1:(n-M), idx]* points to the rows of the data frame for the displaced values.

*Example 2-5 Derive 'lag' data field*

---

```
# lag observations by M, idx ... index of variables to be lagged
lagging = function(dta,idx, M)
{
  n = nrow(dta)
  # ensure correct order by date
  dta = dta[order(dta$date),]
  # lag data
  dta.lag = dta[,c(match("date", names(dta)), idx)]
  temp.NA = matrix(NA,nrow=M, ncol=length(idx))
  names(temp.NA) = names(dta)[idx]
  dta.lag[,match(names(dta)[idx], names(dta.lag))]
  = rbind(temp.NA ,as.matrix(dta[1:(n-M),idx]))
  dta.lag = as.data.frame(dta.lag)
  # rename lagged variables
  names(dta.lag)[match(names(dta)[idx], names(dta.lag))]
  = paste(names(dta)[idx], ".lag",M, sep="")
  return(dta.lag)
}
```

---

Example 2-6 assumes that the data `dta` is a data frame that contains weekly volumes. In addition to the volume column, `dta` contains the fields `date`, `year`, and `q`, which are the variables that contain the date, year, and quarter information for the volume record. Again, the variables for which the total previous quarterly volume will be calculated are indexed by `idx`. The R function `aggregate` computes the quarterly total volumes so that the newly created data frame `q.vols` contains the quarterly volumes. The quarter and year variables in `q.vols` are shifted by one unit. Finally, `q.vols` is merged with the original data.

*Example 2-6 Calculate the sum of the volumes for the previous quarter*

---

```
# total volume in previous quarter
previous.q = function(dta,idx)
{
  q.vols = aggregate(dta[,idx],
                     by = list(year=dta$year, q = dta$q),
                     sum)
  ind.4 = q.vols$q==4
  q.vols$q[!ind.4] = as.numeric(q.vols$q[!ind.4])+1
  q.vols$q[ind.4] = 1
  q.vols$year[ind.4] = as.numeric(q.vols$year[ind.4])+1
  # rename variables
  names(q.vols)[-match(c("year", "q"), names(q.vols))]
  = paste(names(dta)[idx], ".prevQ", sep="")
  q.vols = merge(dta[,match(c("date", "year", "q"),
                           names(dta))], q.vols)
  # remove variables 'q' and 'year'
  q.vols = q.vols[,-match(c("year", "q"), names(q.vols))]
  return(q.vols)
}
```

---

In Example 2-6, the R function `Aggregate()` computes the current quarter volume sum and shifts the quarter index variable by one step. Next, it uses the `merge` function to connect the sum of the quarter volume to the original data frame by shifting the quarter index variable.

## 2.4.1 Challenge and approach

The forecasting solution in R used various array operations, such as shifting the array elements, aggregation, and user-defined operations. These same array variable typing and related operations are not available in SPSS. However, a variable type named set that can read and append element operations is readily available in SPSS Modeler. To implement the R array functionality that was needed in the SPSS Modeler environment, we used the `Derive` node, `Filter` node, and other `Field Ops` nodes for data handling. Also, we used scripts to configure the settings and provide necessary functions. In addition, we made extensive use of the `Expression Builder` that is embedded in the SPSS Modeler nodes. The nodes provide abundant array functions, and the `Expression Builder` allowed for the necessary array calculations.

Figure 2-4 on page 21 shows the SPSS Modeler `Expression Builder`. The `Expression Builder` includes a wide variety of functions and the variables (that is, fields) on which the functions can operate.

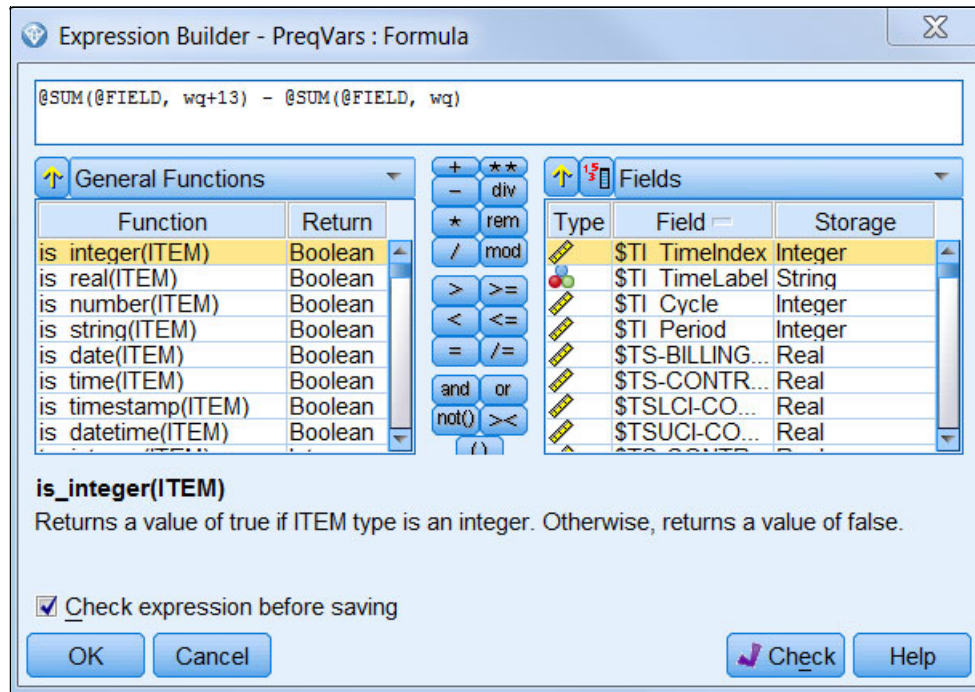


Figure 2-4 SPSS Modeler Expression Builder

## 2.4.2 Setting SPSS nodes and parameter setup

To create the stream that has the same functionality that R provides, perform the following steps:

1. Add a History node into the stream to derive the lag variable from input data. Set the input fields and then set these parameters:
  - Offset =1
  - Span =2
  - Where history is unavailable, leave history undefined.

Figure 2-5 on page 22 shows the History node setting (highlighted in yellow). This setting enables the delay operation to be implemented. Each input variable in the Selected fields will derive two new fields with shift 1 offset and 2 offset.

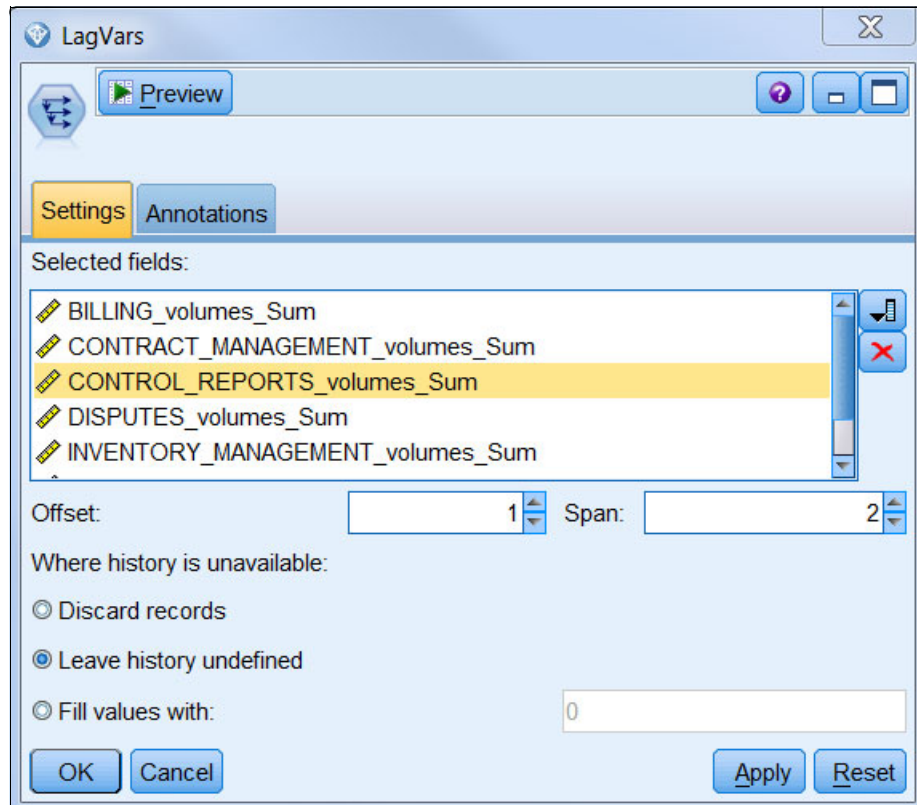


Figure 2-5 Setting the History node

2. Add a Derive node (Figure 2-6 on page 23) into the stream for deriving the sum of the previous quarter variables. Set the Mode to **Multiple**, Derive From (as shown in Figure 2-6 on page 23), and Derive As to **Formula** settings. Modify the formula for the deriving operation by using Expression Builder, and click the **Calculator** icon.

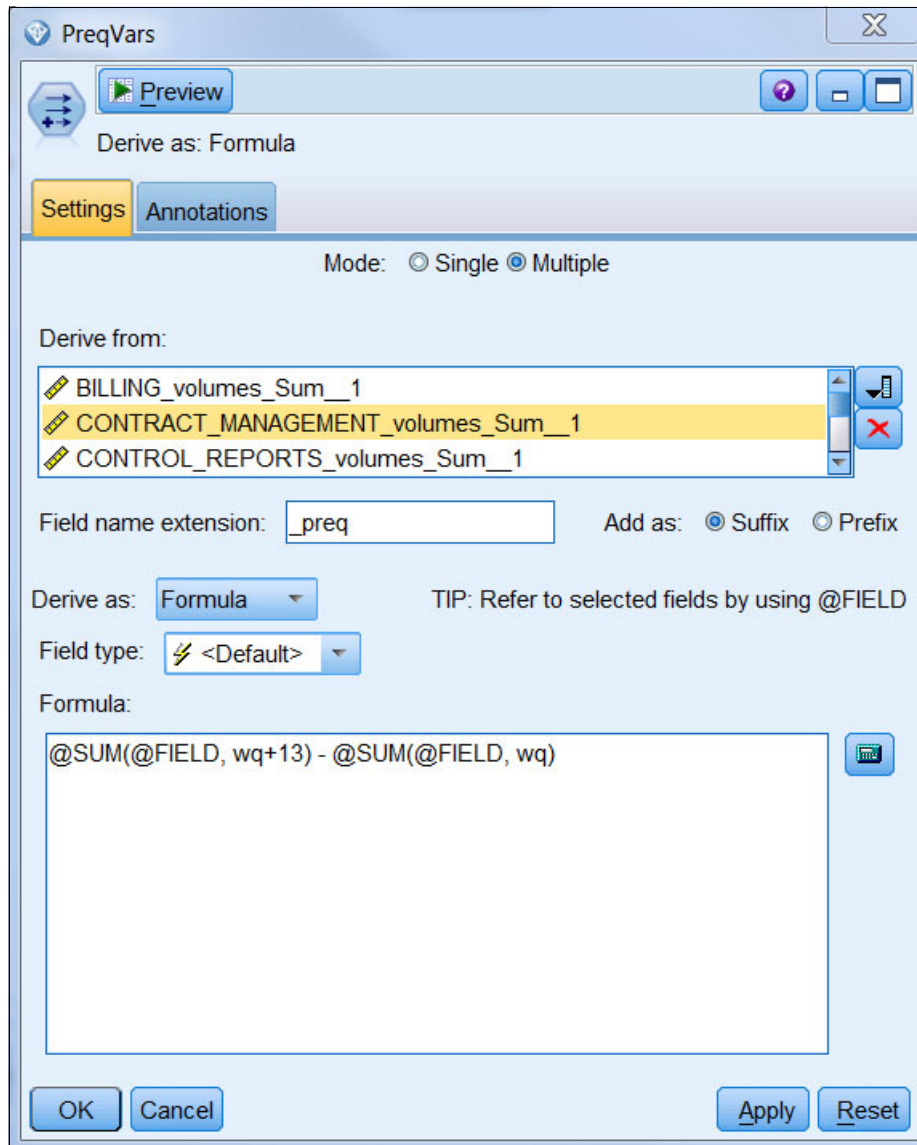


Figure 2-6 Setting the Derive node

3. The formula (Figure 2-7 on page 24) is used to derive new variables that sum up the volumes of the previous quarter for each field that is selected in the “Derive from” setting. The equation  $\text{@SUM}(\text{@FIELD}, \text{wq}+13) - \text{@SUM}(\text{@FIELD}, \text{wq})$  is created by using the Expression Builder. The function  $\text{@SUM}(\text{FIELD}, \text{EXPR})$  returns the sum of the values for FIELD over the last EXPR records, including the current record. And the @FIELD parameter specifies applying the operation on each field in turn.

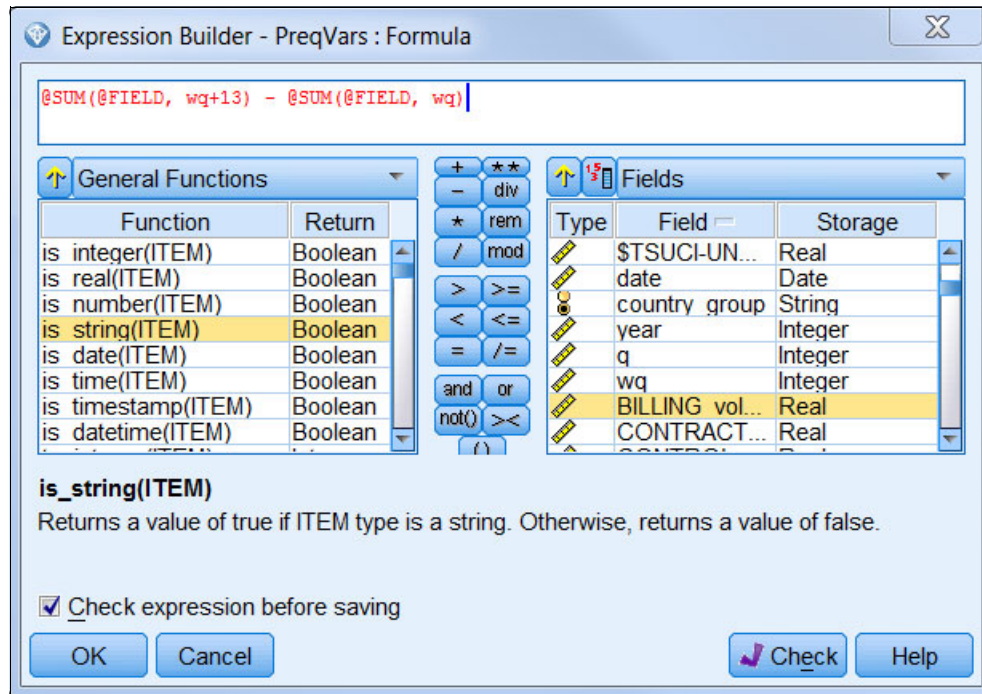


Figure 2-7 Create the formula with Expression Builder

4. Create a script to ensure that the node settings are as expected while the stream loops through the hierarchy branches. The script consists of the code that is shown in Example 2-7.

Example 2-7 Script for node settings

```
set LagVars.fields = ^request_tars
set PreqVars.fields = ^request_tar_lag1
```

Figure 2-8 shows how the same functionality of variable generation (R code) is implemented in SPSS Modeler.

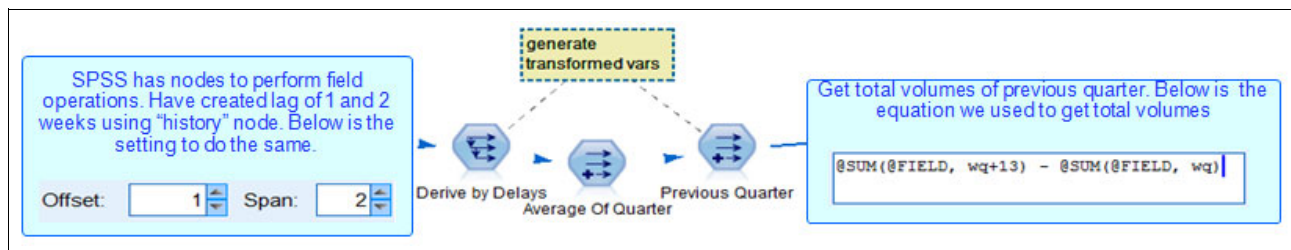


Figure 2-8 Create lag variable and Sum of previous quarter

## 2.4.3 Results

To implement the array calculations, three types of SPSS Modeler capabilities (stream, scripts, and the Expression Builder node) were used:

- The stream with the nodes provides the data processing control.
- The Expression Builder, which is embedded into the nodes, provides abundant functions for array computation.

- The scripts provide programmability, including setting parameters, executing loops, adjusting the stream structure, and expression equations.

With this approach, data transformation is accomplished in SPSS Modeler as in R. It was easier to derive fields in SPSS nodes and easier to understand SPSS nodes.

## 2.5 Defining variable types

We automatically define the type of variables every time that a model is created. Categorical, continuous, and binary variables are examples of variable types.

R has extensive flexibility when loading data and specifying the variable types. Correctly defining the variables in a data set is important for two reasons:

- An analysis procedure returns an incorrect solution or simply does not run if the data that you want to analyze is formatted incorrectly. For example, to include a categorical predictor in a regression procedure, the predictor needs to be set as a factor. If not, the predictor is automatically considered a continuous predictor.
- When you work with data, ensuring that the variables are clearly defined and specified is important so that anyone working with the same data set can tell exactly what was measured. Defining a variable in a data set includes giving the variable a name and specifying its type and the values that the variable can take. Without this information, the data will be harder to understand and use for further analysis.

When you build a statistical model for forecasting, the first step is to obtain the important variables for prediction. For example, historical data is often a good predictor of future data in addition to seasonal and holiday effects. Each of the candidate variables must be included in the correct format so that the code for automated model building runs smoothly and returns the correct fit.

### 2.5.1 Challenge and approach

In SPSS Modeler, we defined the type of predictor and target variables before we built a model. We built the models for different combinations of centers, lines of business (LOBs), request types, and country groups. Predictors of different combinations can differ. Also, the type of the same predictor can differ because data is read from multiple data sources. The type of predictor and target variables need to be defined for every combination. Manually defining variable types for every combination is impractical because we build models for many combinations once every two weeks. Therefore, it was important to identify a method to define the type of variables automatically.

A script was created to define variable types automatically.

### 2.5.2 Setting SPSS nodes and parameter setup

This section explains how the type of variables was defined automatically. Figure 2-9 on page 26 shows the Modeler Type node that was used to define the type of predictors.

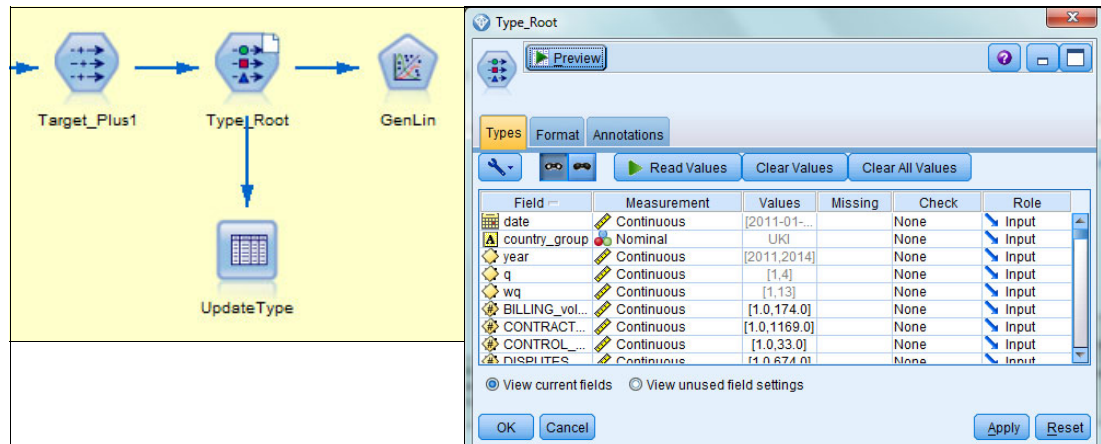


Figure 2-9 Type node to define the type of predictors

Example 2-8 contains the script that was used to define variables that use the Type node automatically.

*Example 2-8 Script to define variables automatically that use the Type node*

```
if ^first_loop = 'yes' then
deleteType_Root
load node C:/Type_Root.nod
positionType_Root connected between Target_Plus1 and GenLin
setfirst_loop = 'no'
endif
execute UpdateType
clear outputs
```

We used the following steps to define the variable types for the different combinations:

- ▶ To build a model of specific combinations, such as center, LOB, country group, and request group, the Type node is deleted by the script.
- ▶ A new Type node, which does not define the type of any field, is inserted in the stream.
- ▶ The UpdateType node is a table node that displays all predictors and target variables for the current combination.
- ▶ The table node provides information about the predictors and target variables of the current combination to the Type node. The type of variables that are displayed by the table node is defined by the Type node.

These actions are repeated for each new combination.

## 2.5.3 Results

SPSS is strict about defining variable types before modeling. Manually defining the type of variables is too time-consuming. Using scripts to successfully define the variable type enabled the activity to be performed automatically.

## 2.6 Incorporating automated exception handling

This section introduces how to use SuperNode and condition-checking rules to handle exceptions automatically in SPSS Modeler.

To build forecasting models for different data sets, incorporating mechanisms into the programs that allow the code to continue processing even if errors or failures occur is necessary. Two main methods of coping with failures in the code are available:

- ▶ The prevention of failures involves how to avoid errors that will break the code. Preventing failures requires the programmer to incorporate rules to help avoid failures.
- ▶ Tolerance deals with how to provide satisfactory answers regardless of errors and failures in the code. However, completely avoiding problems or failure is not possible so inherently failure-tolerant programs are clearly needed.

Not all problems are unexpected. Certain potential problems can be anticipated, for example, the wrong type of input or the non-convergence of a solution. These problems in R code are handled by using conditions. Certain R tools, such as `try()`, allow the program to take specific actions when a condition occurs. For example, if the task is to fit a model to many different data sets, we wanted to continue fitting models to data sets even if one model fails to converge. The `try()` function skips over the error causing inputs and allows the program to continue execution. Boolean state flags can be used to control or monitor the flow of code inside of the `try()` function. State flags aid in transferring control of the computation to another part of the program.

The example R code in Example 2-9 illustrates the use of the `try()` function and conditional statements for automated error handling. A list of data sets and a model fitting routine are passed as inputs and the goal is to apply the model fitting routine to all of the data sets. The loop starts by assigning an initial value to the boolean flag `success.indicator`. Next, the `try()` function is used to fit the model to the current data set. If the model fitting procedure is successful, the boolean state flag `success.indicator` is updated to reflect the success. Failure to update the state flag transfers control of the computation to another part of the program. In this case, an alternative model, which is known as the *constant model*, is fit. In this simple example, the prevention of failure in the code is done by using the `try()` function while tolerance to failures is done by fitting an alternative model.

*Example 2-9 Example R code that uses the try() function*

---

```
INPUTS  success.indicator, data.list, model
OUTPUTS fitted.models
for (D in data.list) {
  INITIALIZE success.indicator = FALSE
  try( {
    FIT    model to the data set D
    UPDATE success.indicator = TRUE
  }, silent = TRUE)
  if (!success.indicator) {
    FIT a constant model to the data set D
  }
  SAVE the fitted model for data set D to fitted.models
}
```

---

## 2.6.1 Challenge and approach

Certain forecast models do not fit a specific data set successfully. In SPSS Modeler, a script can be used to loop through a list of forecast methods or data sets. However, if one of these methods fails, execution of the SPSS Modeler stream stops automatically and an error message is shown. Executing the SPSS stream from the stop point does not occur. In this case, when one model fails, the failure must keep a failure record and continue trying other models.

To solve this problem, a model building part was put inside of a SuperNode in SPSS Modeler. Loop control is coded in the script for the main stream. If one of the forecast models inside of the SuperNode fails, the execution of the SuperNode stops but the execution of the main stream continues. This approach enabled exception handling in SPSS Modeler to occur successfully.

## 2.6.2 Setting SPSS nodes, flow chart, and parameter setup

In this section, an example of how to handle exceptions by using the SuperNode in SPSS Modeler is described. Figure 2-10 shows the Modeler window for the SuperNode that was used to handle exceptions in SPSS.

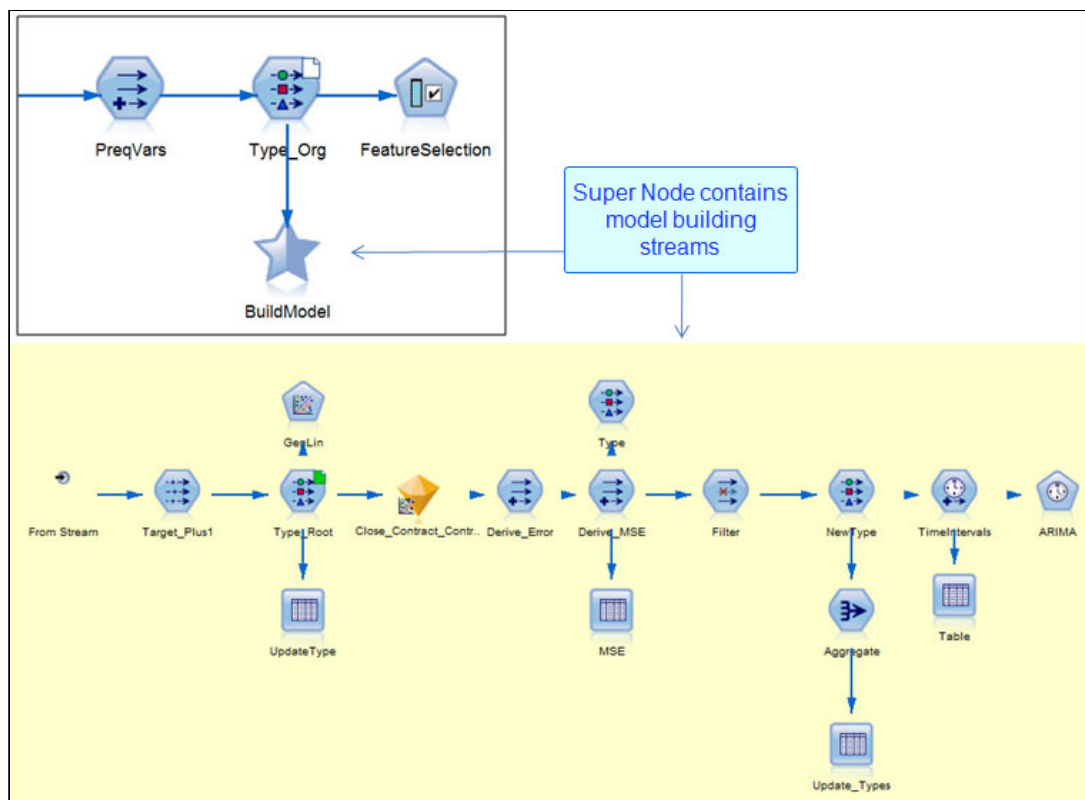


Figure 2-10 Use a SuperNode to handle exceptions in SPSS Modeler

The following steps describe how to set up exception handling in SPSS:

1. Create a stream to build models. In Figure 2-10 on page 28, the stream inside of the yellow rectangle is the model building stream. The pentagon node that is named GenLin is the node that is used to build the Generalized Linear model. The pentagon node that is named autoregressive integrated moving average (ARIMA) is the node that is used to build the ARIMA model.
2. Select the model building stream that was created in step 1, and create a SuperNode. As shown in Figure 2-10 on page 28, the star node that is named BuildModel is a SuperNode. The model building stream is inside of this SuperNode.
3. Edit the SPSS script for the main stream. This script controls the execution of the main stream, that is, outside of the SuperNode. Loops for each country group and request type are coded here. A log file is created before the models are built. This log file records whether the model building for each country group and request type is successful. Example 2-10 shows the SPSS script for the automated exception handling.

---

*Example 2-10 Existing script in SPSS Modeler for automated exception handling*

---

```
for org in org list
  ....
  var FILE
  set filename = ^model output directory >< '/' >< ^org >< ' model list.txt'

  set FILE = open create ^filename
  write FILE 'Target,Distribution,TimeSeries'
  close FILE
set BuildModel.parameters.model list = ^filename
for request in request tars
  set AggregateTarget.keys = ^request
  execute ConstantTarget
  if ConstantTarget.output.row count <= 1 then
    set FILE = open append ^filename
    writeIn FILE ''
    write FILE ^request >< ",NO,NO"
    close FILE
  else
    ...
    execute BuildModel
  endif
endfor
endfor
```

---

Under certain conditions, the model building part fails. We defined certain rules and checks to see whether these rules are met. If the defined rules are not met, a failure record is kept instead of going to the model building part. Otherwise, the SuperNode is executed and the forecast model is built. Sometimes, all of the possible situations that cause a failure of model building cannot be predefined. These unexpected conditions cause an interruption of the execution inside of the SuperNode. In this case, the execution of SuperNode stops, which is recorded in the log file for that specific country group, and the request type is left blank. Execution of the main stream continues to the next country group and request type.

4. Optional: Edit the script inside of the SuperNode. In our case, two models are inside of the SuperNode. The first model is the Generalized Linear model. We can try different distributions and link functions, and the best model is selected out of them. If the Generalized Linear models are built successfully, the best model can be recorded in the log file. The second model is ARIMA. Similarly, we can keep a record in the log file whether the ARIMA model is built successfully.

## 2.6.3 Results

The SuperNode and defined condition-checking rules are used to handle exceptions in SPSS Modeler. By using a SuperNode, the SPSS stream can continue if an unexpected failure happens. Also, by defining condition-checking rules, the condition of a failure can be identified in advance, reducing the interruptions. A log file is created to record the success or failure of modeling. Table 2-1 shows the contents of a sample log file.

*Table 2-1 Log file for country group=AP*

Target	Distribution	Time series
Close contract/contract termination	Gaussian log	Yes
Contract acceptance registration	Gaussian log	Yes
Customer details question	Gaussian log	Yes
Disputes	No	No
Inventory updates	No	No
Modify contract	Gaussian log	Yes
Other		
Product price	Gaussian log	Yes
Produce proposal contract	Gaussian log	Yes

The log file (Table 2-1) is for country group=AP. The table contains the following elements:

- The Target column lists the name of the request types.
- The Distribution column has either Gaussian log, which means that the Generalized Linear model is built successfully with Normal distribution, and the link function is Log, or No, which means that the Generalized Linear model is not built because the input data does not meet the defined checking rule.
- The TimeSeries column has either a Yes, which means that the time series model ARIMA is built successfully, or a No, which means that the ARIMA fails because the input data of the ARIMA model does not meet the defined checking rule. Here, the input data of ARIMA is the residuals of the Generalized Linear model.

**Table column explanation:** The columns Distribution and TimeSeries for Target = Other are blank, which means that the execution of the SuperNode stops and all models fail for an unexpected reason.

The log file can be read in the stream to generate a forecast. For a data set with a failure in model building, the historical volume is used as the forecast.

In conclusion, handling exceptions in SPSS Modeler is not easy because SPSS Modeler does not have an exception-handling mechanism that equals the exception-handling mechanism in R. However, by using a SuperNode and defining condition-checking rules, the SPSS stream of the forecast model can run smoothly.

## 2.7 Supporting variable selection

This section introduces how to select variables in SPSS Modeler by using the Feature Selection node.

For each time series, several candidate models are built. The model that results in the smallest prediction error on a holdout test set is selected to generate future forecasts. The purpose of this procedure is to build a model that has good forecasting performance and that is robust to unseen data. In general, the task of selecting a model involves the selection of the model form and the selection and form of the variables (that is, lags, transformations, and interactions) in the model.

Variable selection improves prediction performance and reduces the size of the candidate set of variables to a subset. The subset includes only the relevant variables for forecasting the response. We make an important distinction between forced and candidate variables:

- *Forced* variables remain in the forecasting model, by default.
- *Candidate* variables are important or relevant for describing the dependent variable but they might not be retained in the forecasting model.

A commonly used variable selection procedure is backward elimination. *Backward elimination* starts from the “full model”, that is, the model that contains all of the candidate variables and the forced variables as predictors. Next, the candidate variables are ranked by using the p-values of a statistical test. The candidate variable with the largest p-value that exceeds a certain threshold is removed from the model. After the insignificant candidate is removed, a new model is obtained and the process of elimination starts again. The procedure can be repeated until all candidate variables in the model are statistically significant.

Many variable selection algorithms include an underlying variable ranking mechanism. By using a ranking of candidate variables, we can focus the decision on the inclusion or retention of each candidate at a time. The variables are ranked by using the p-values of the test that compares the reduction in residual deviance when the candidate variable is removed from the current model. Determining whether a candidate must be removed at each step can be evaluated in several ways. The threshold for the p-value was set at a 5% level. That is, the lowest ranking candidate if its p-value exceeds 0.05 is removed.

In R, the `anova()` function calculates an analysis of variance table for a Generalized Linear model. By using this capability, we can obtain the residual deviance and associated p-value when removing each candidate variable from the current model in the order that is listed in the model formula. This approach requires fitting as many models as there are candidate variables in the formula. The time that is required for the function to perform the calculations scales linearly with the number of candidate predictors and requires a long time to fit models with many predictors.

The example R code (Example 2-11 on page 32) describes the main components of the `model.select()` routine in which the backward elimination procedure is implemented by using the `anova()` function. The threshold of 0.05 is fixed; therefore, it cannot be modified by the user. The candidate variable search procedure ends when all of the candidate variables in the model have p-values that are below the fixed threshold or alternatively, when all of the candidates are dropped.

*Example 2-11 Example R code of the main components of the model.select()*

---

```
INPUTS      response, candidate.variables, forced.variables, model
OUTPUTS     candidate.variables
INITIALIZE  p.vals = 1 (vector of p-values)
            p.max = 1 (index of largest p-value)
while (p.vals[p.max]>0.05 & length(candidate.variables)>0) {
  FIT      model for the response using candidate.variables and forced.variables
  OBTAIN   p-values for the current candidate.variables using anova() function
  UPDATE   p.vals with the largest p-value
            p.max with the index of the largest p-value
  UPDATE   candidate.variables by removing the one with the largest p-value if
exceeds 0.05
  if (p.vals[p.max]>0.05) {
    REMOVE candidate.variable with the largest p-value that is greater than 0.05
  } else{
    break;
  }
}
```

---

## 2.7.1 Challenge and approach

Variable selection in R uses selection criteria that is based on the p-value. It is easy to conduct an ANOVA test by calling `anova()` in R. And, the results that are returned by the `anova()` function contain a p-value. However, implementing the same algorithm in SPSS Modeler is difficult. First, no separate node is available for the ANOVA test in SPSS Modeler. Second, although the ANOVA test is used in certain models in SPSS Modeler (for example, Regression node), extracting this part of the result is difficult. For these two reasons, SPSS Modeler does not have the flexibility to allow us to develop the same variable selection algorithm as in R. We had to identify an equally good or better way to perform variable selection.

In SPSS Modeler, an automated method of variable selection exists in the Feature Selection node that can shortcut this process and narrow the list of candidate predictors. Feature selection in SPSS consists of three steps:

- Screening

Removes unimportant and problematic predictors and cases. Fields that have too much missing data, too little variation, or too many categories, among other criteria, are removed.

- Ranking

Sorts remaining predictors and assigns ranks. Each predictor is paired with the target and an appropriate test of the bivariate relationship between the two is performed. The probability values from these bivariate analyses are turned into an importance measurement by subtracting the p-value of the test from one (therefore a low p-value leads to an importance near one). The predictors are then ranked on importance.

- Selecting

Identifies the important subset of features to use in subsequent models.

The variable selection algorithm in the Feature Selection node differs from the variable selection algorithm. First, R does not include a screening step. To allow for this situation, we skip the screening step when setting up the Feature Selection node in SPSS Modeler. Second, the ranking step in SPSS Modeler also differs. Backward elimination is used in R for variable selection. Variables in R are ranked by using the p-values of the test that compares the reduction in residual deviance when the candidate variable is removed from the current model. The decision on the inclusion or retention of one candidate is considered one at a time. Ranking and selecting procedures are repeated iteratively until all the candidate variables in the model are statistically significant. However, in the Feature Selection node in SPSS Modeler, a statistical test is used to test the bivariate relationship between each candidate predictor and the target. The selection procedure is considered once.

Although the variable selection procedures in R and SPSS differ, in our case, we tried the Feature Selection node and compared the output of the forecast model. We discovered that the forecast accuracy is not affected.

## 2.7.2 Setting SPSS nodes and parameter setup

Implementing variable selection by using the Feature Selection node in SPSS is easy. The steps to set up the Feature Selection node are described:

1. After preparing data and creating the list of target and candidate predictors, add a Feature Selection node to the stream and connect it to the Type node. Figure 2-11 shows the pentagon node that is named FeatureSelection, which is the Feature Selection node that is added to the SPSS stream. Select **FeatureSelection**.

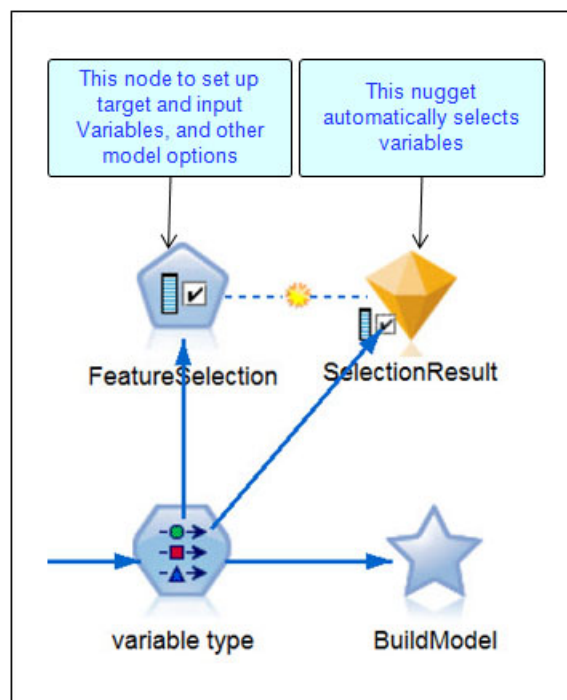


Figure 2-11 FeatureSelection node in SPSS Modeler

2. On the FeatureSelection window (Figure 2-12), edit the Feature Selection node. Select the **Fields** tab. On the Fields tab, select the target variable and inputs. The target variable is the variable to predict. Inputs are the list of candidate variables.

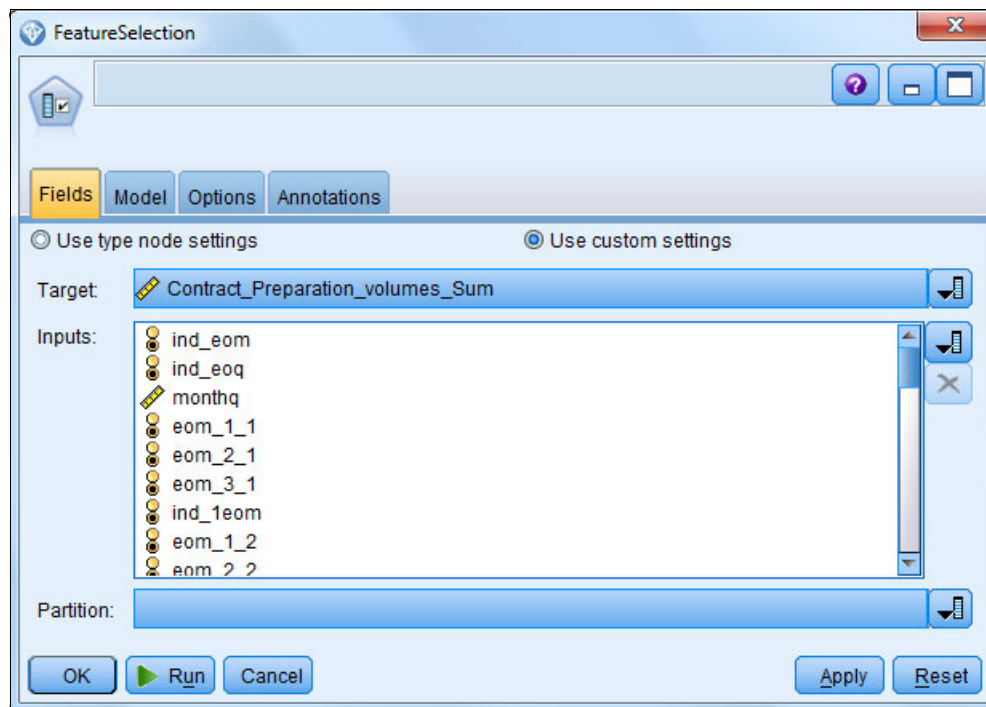


Figure 2-12 Set up the Fields tab in the FeatureSelection node

3. Switch to the **Model** tab (Figure 2-13) in the Model view. Ensure that all the “Screen fields with” are cleared.

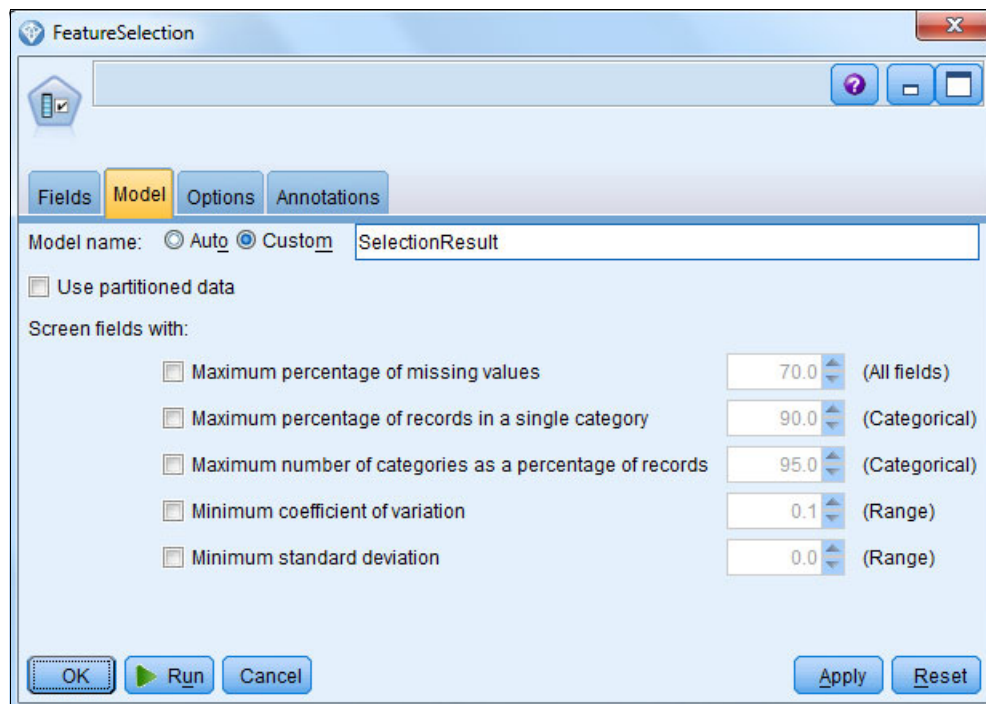


Figure 2-13 Set up Model tab in the FeatureSelection node

4. Switch to the **Options** tab (Figure 2-14). On the Options tab, select **Important** and **Marginal**. Only unimportant variables with an importance of  $<0.8$  are removed.

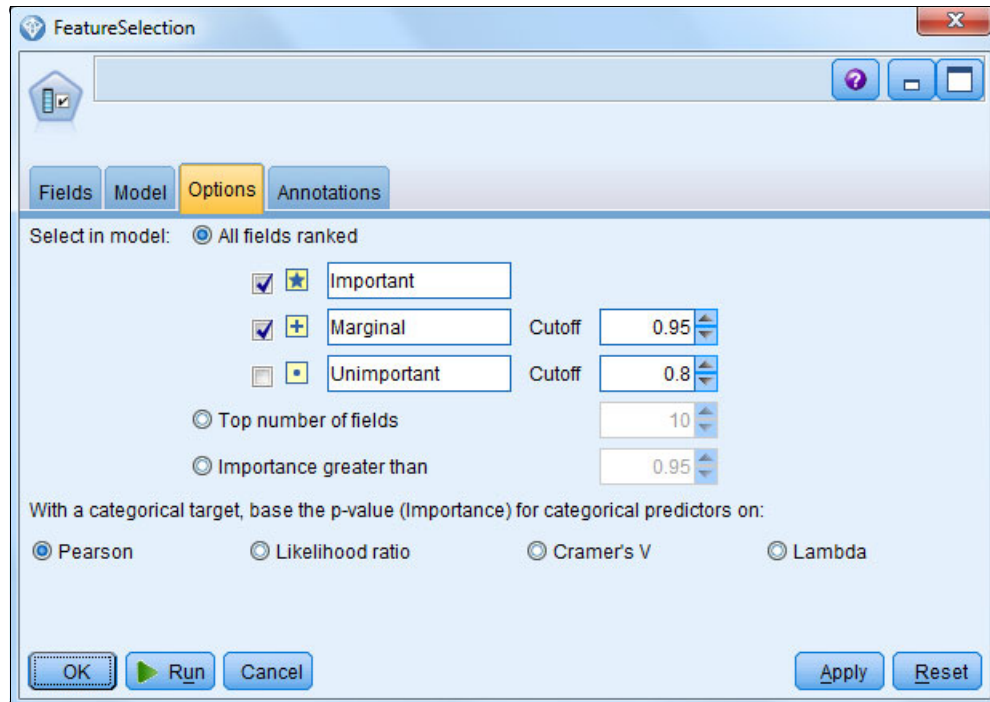


Figure 2-14 Set up the Options tab in the FeatureSelection node

Click **Run** to execute the FeatureSelection node. A nugget with the selected variables is generated.

In Figure 2-13 on page 34, the nugget that is named SelectionResult is the output of the execution of the FeatureSelection node. FeatureSelection acts as a filter node, removing unnecessary fields downstream (with parameters under user control). Our forecasting model is built based on the selected variables from this nugget.

The Feature Selection node can identify the fields that are the most important (most highly related) to a particular target/outcome. Reducing the number of required fields for modeling allows you to develop models more quickly and permits you to explore the data more efficiently. In the point and click interface of the Feature Selection node, we can set up parameters easily. Figure 2-15 on page 36 shows sample results from the generated nugget. We have 42 candidate variables, and 18 fields are selected.

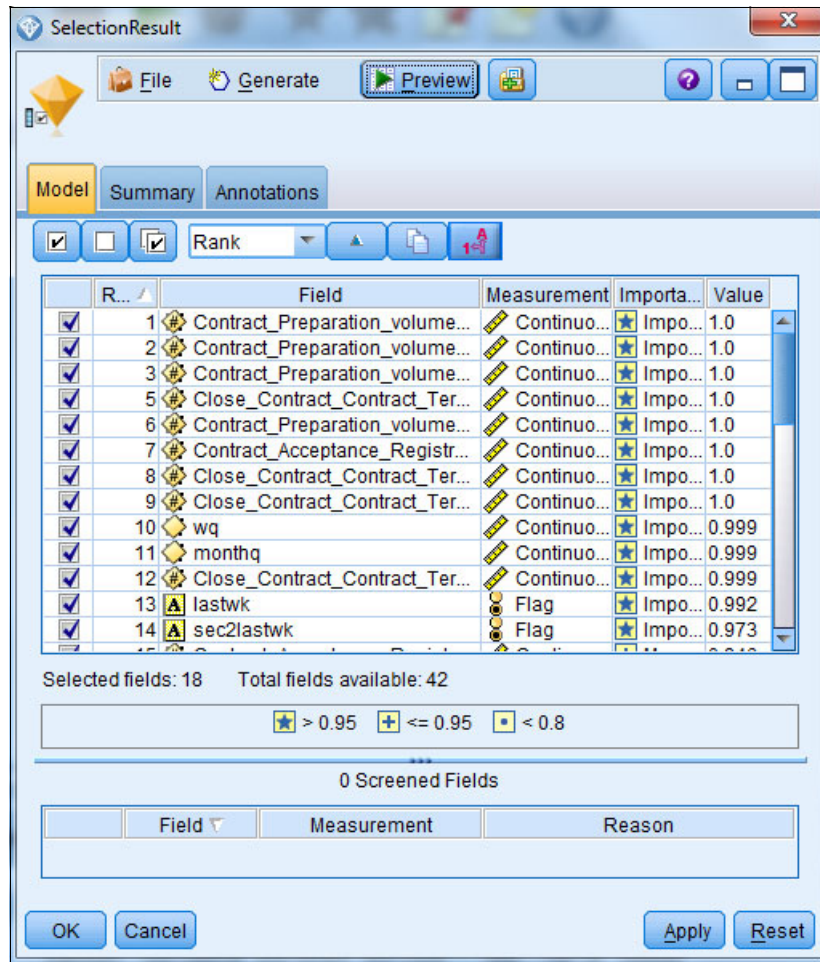


Figure 2-15 Result of Feature Selection

Variables in the FeatureSelection node are ranked by importance. A field with an importance value near one does not guarantee that it is a good predictor of some target field. Another limitation of Feature Selection is that nonlinear relationships will not necessarily be detected by the tests that are used in the FeatureSelection node.

### 2.7.3 Results

SPSS Modeler does not have the same flexibility that is needed to develop a customized variable selection algorithm as in R. However, by taking advantage of the Feature Selection node, the variable selection step can be much easier because we only need to add a node instead of writing an algorithm. By testing the converted forecast model in SPSS Modeler and comparing the forecast accuracy of SPSS Modeler with R, we determined that the forecast accuracy is not affected and that we can provide results that are comparable to the R results.

## 2.8 ARIMA and Fourier ARIMA model conversion

This section introduces how to build an autoregressive integrated moving average (ARIMA) model automatically by using Expert Modeler, how to extract an ARIMA model from Expert Modeler, and how to call R and IBM SPSS Statistics from SPSS Modeler.

The ARIMA model is a widely used time series model for forecasting. In the ARIMA model, observations of an output are assumed to be a linear function of several past observations and random errors or innovations. The ARIMA model describes the underlying relationship between the past and current observations that can be used to forecast the values of the output. This modeling approach is useful when no satisfactory explanatory model underlies the dynamics of the data generating process. An ARIMA model for the data  $\{x_t\}$  is denoted as an ARIMA(p,d,q) model, which is provided by the formula (Figure 2-16).

$$(1-B)^d x_t = y_t$$

$$\hat{y}_t = \mu + \phi_1 y_{t-1} + \dots + \phi_p y_{t-p} - \theta_1 e_{t-1} - \dots - \theta_q e_{t-q}$$

Figure 2-16 ARIMA (p,d,q) model formula

The following variables are used:

- ▶ p is the number of autoregressive terms.
- ▶ d is the number of nonseasonal differences.
- ▶ q is the number of lagged errors in the prediction equation.

The innovations  $\{e_t\}$  are assumed to be a white noise process.

The seasonal ARIMA model is an extension of the ARIMA class to include seasonal behavior. Seasonal patterns exist when data is affected by seasonal factors, for example, the month or quarter of a year. The seasonal ARIMA allows for a relationship between observations for the same quarters in successive years, as well as a relationship between observations for successive quarters in a particular year, for example. Seasonality in the data is assumed to be known and fixed. Many times, seasonality or periodicity is associated with some aspect of the calendar. The seasonal ARIMA model is specified as ARIMA(p, d, q)(P, D, Q)[s] where P is the number of seasonal autoregressive terms, D is the number of seasonal differences, and Q is the number of seasonal-lagged errors.

Data sometimes exhibits cyclic patterns that are not seasonal, such as the rise and fall that do not have a fixed period. For example, business cycles do not have a fixed duration and might not be entirely periodic. Moreover, the length of the current cycle might be unknown beforehand. ARIMA models can describe data that has both seasonal and cyclic behavior. However, long-term cycles are not handled well by the ARIMA framework. One alternative is the Fourier ARIMA where Fourier terms are used to model the seasonal and cyclic patterns while short-term dynamics are modeled by using nonseasonal ARIMA. Fourier ARIMA allows for seasonality and cycles of any length, including non-integer length.

Creating a seasonal ARIMA or Fourier ARIMA model for forecasting involves three main steps:

- ▶ Proposing candidate models
- ▶ Fitting the models to the data
- ▶ Choosing the best model under a specific criterion

Proposing candidate models can be done by an exhaustive search. A list of the orders for the candidate models can be restricted to limit the time and computational resources that are needed to find the optimum. The number of proposed candidate models for a seasonal ARIMA model can be large and usually the nonseasonal and seasonal differencing orders are fixed beforehand. Fitting models are fitted to the data quickly in R by using the `arima()` function. In the case of Fourier ARIMA, the number of Fourier terms is also a parameter that might be fixed beforehand or proposed during the first step. The Fourier terms are included in the model as external regressors. Training data is used to fit the model and test data is used

to evaluate the candidate. A performance criterion, such as the root mean squared error of prediction, can be used to compare forecasts from the learned model and data in the holdout set. The model with the best out-of-sample performance is chosen.

Example 2-12 shows example R code that illustrates an exhaustive search for the optimal seasonal or Fourier ARIMA model. The input data is for a training set and a test set. The candidate models are passed on as a list of nonseasonal orders (`nonseas.ord`) and seasonal orders (`seas.ord`). The selection criterion *crit*, which is used to evaluate the model on a holdout set, needs to be specified. External regressors for the training and the test set might be included and passed on as `xreg.train` and `xreg.test`. In the case of the Fourier ARIMA, the `seas.ord` must be a null list and the `xreg.train` and `xreg.test` must contain the Fourier components. The goal is to find the best model by fitting all of the proposed models and evaluating these in the holdout set. To do this, initialize a variable `best.crit` to a large value. This variable will hold the best value of the criterion. When a proposed model performs better than all the previous ones, the `best.crit` value will be updated and the `best.model` result will be updated to hold the current model.

*Example 2-12 Exhaustive search for optimal ARIMA model*

---

```

INPUTS  train.D, test.D, nonseas.ord, seas.ord, crit, reg.train, xreg.test
OUTPUTS best.model
INITIALIZE best.crit = 1e20
for (i in nonseas.ord) {
  for (j in seas.ord) {
    FIT an ARIMA model of nonseasonal order i and seasonal order j using the
    train.D and external regressors xreg.train
    CALCULATE the criterion crit using the fitted model, the test.D and the external
    regressors xreg.test
    if (current criterion value < best.crit) {
      UPDATE best.crit with the current criterion value
      UPDATE best.model with the current model
    }
  }
}

```

---

Although seasonal and Fourier ARIMA models are flexible in the sense that they can represent several different types of time series, including pure autoregressive (AR), pure moving average (MA), and combined AR and MA (ARMA) series. The major limitation is the assumed linear form of the model. That is, a linear autocorrelation structure is assumed and therefore, no nonlinear patterns in the correlation can be captured by the ARIMA model.

## 2.8.1 Challenge and approach

To build a seasonal ARIMA model, an algorithm that is based on exhaustive searching was developed in R that selects the best ARIMA model. The input data set is split into two parts: training data and test data. The candidate model with the lowest Root Mean Squared Error (RMSE) is selected.

Converting this algorithm directly from R to SPSS was challenging because the algorithm generates a huge number of candidates. As a result, it takes a long time to run. Moreover, based on our experimental testing, the running time in SPSS Modeler is usually longer than in the R solution with the same forecast method. Because we need to consider hundreds of time series for our production run, the long running time of this seasonal ARIMA model was a problem. Considering this condition, we needed to find an easy to build, good-quality model with a short running time. We used ARIMA Expert Modeler in SPSS Modeler. ARIMA Expert

Modeler in SPSS Modeler can select the best seasonal ARIMA model based on the lowest Akaike information criterion (AIC). However, the model selection in ARIMA Expert Modeler is not based on forecast error as in the R model and the model selection in ARIMA Expert Modeler has the problem of over-fitting. The advantage of ARIMA Expert Modeler in SPSS Modeler is that it takes a shorter time to select the model and it is easy to build. Another advantage is that ARIMA Expert Modeler in SPSS searches a broader space than the searching algorithm that is developed in R. We tried ARIMA Expert Modeler in SPSS Modeler and compared the forecast accuracy of the R model and the SPSS model. The forecast accuracy of the R model and the SPSS model were comparable.

Another challenge was to extract the ARIMA model from ARIMA Expert Modeler. After running ARIMA Expert Modeler, we had to keep a record of the ARIMA model  $(p,d,q)(P,D,Q)$ . Next, the fixed model  $(p,d,q)(P,D,Q)$  was applied to different data sets and forecast errors were calculated. However, there was no straightforward way to extract  $ARIMA(p,d,q)(P,D,Q)$  from the output report of ARIMA Expert Modeler in SPSS. The output report from ARIMA Expert Modeler in SPSS Modeler contained the ARIMA model and other information. To solve this problem, we first exported the output report to a text file. Next, we imported the file and removed the unnecessary parts.

The ARIMA model had to be converted by using Fourier transformation. Spectrum analysis is the key step in Fourier transformation. In R, conducting spectrum analysis by calling the `spectrum()` function is easy. However, no corresponding module exists for spectrum analysis in SPSS Modeler. To solve this problem, we tried two approaches:

- Call R

In SPSS Modeler Version 16, R can be called by using a node. The R node can be added in SPSS Modeler and R code about Fourier transformation can be put inside of the R node.

- Call SPSS Statistics

Although spectrum analysis is unavailable in SPSS Modeler, it is available in SPSS Statistics. In SPSS Modeler, connections can be built between SPSS Modeler and SPSS Statistics. Functions that are available in SPSS Statistics can be called in SPSS Modeler.

## 2.8.2 Setting SPSS nodes, flow chart, and parameter setup

This section covers the following items:

- How to build the seasonal ARIMA model automatically by using Expert Modeler
- How to extract the parameters of the ARIMA model from Expert Modeler
- How to perform spectrum analysis by calling R or by calling SPSS Statistics from SPSS Modeler is introduced

## Build the seasonal ARIMA model automatically by using Expert Modeler

The following steps show how to build the seasonal ARIMA model automatically by using ARIMA Expert Modeler:

1. Add a Time Series node to the main stream. Open the Time Series node, as shown in Figure 2-17. Under the Model tab, select **Expert Modeler** as the Method. Click **Criteria**.

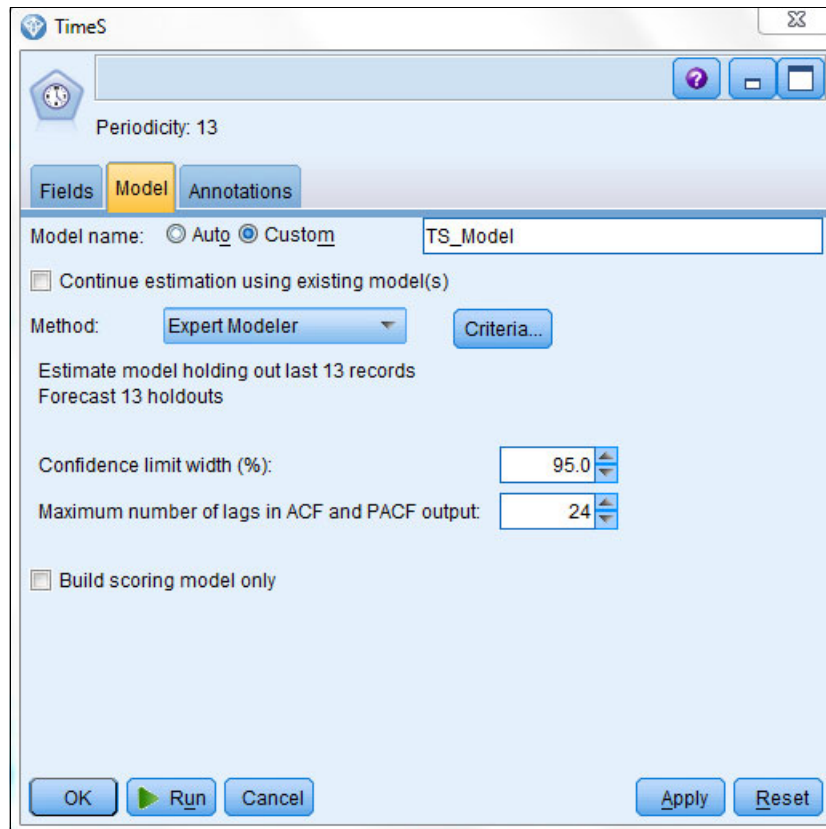


Figure 2-17 Set up ARIMA Expert Modeler in Time Series node in SPSS - Method

2. The Time Series Modeler: Expert Modeler Criteria window opens (Figure 2-18). Under the Model tab, select **ARIMA models only** as the Model Type. To consider seasonal components in the ARIMA model, select **Expert Modeler considers seasonal models**.

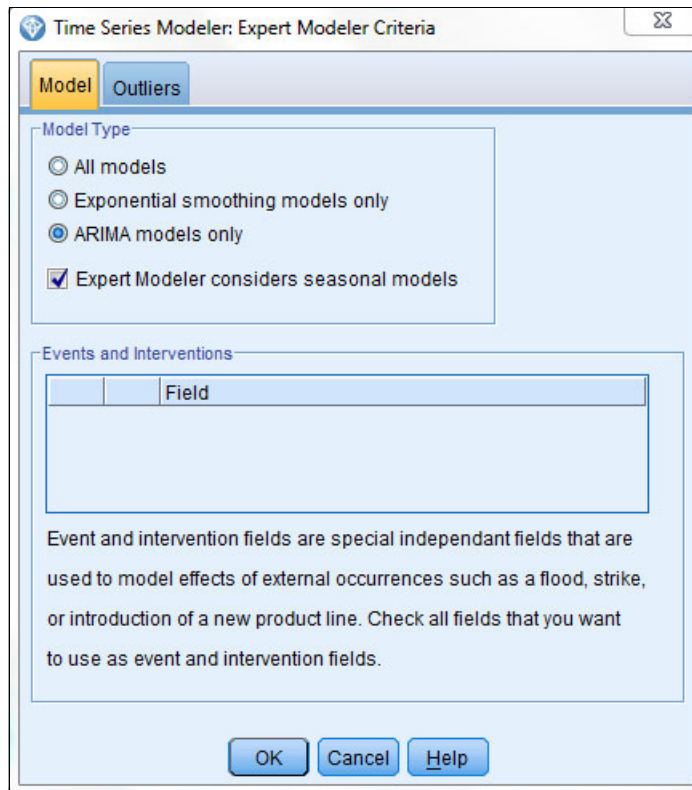


Figure 2-18 Set up ARIMA Expert Modeler in the Time Series node in SPSS - Criteria

Instead of setting up ARIMA Expert Modeler in the Time Series node manually, we can also use an SPSS script to set up the Time Series node.

### Extract the output of ARIMA Expert Modeler

This section shows how to extract the parameters of the ARIMA model from ARIMA Expert Modeler. Follow these steps:

1. Add a Time Series node. Set up a Time Series node. Here, we used an SPSS script (Example 2-13) to set up ARIMA Expert Modeler.

#### Example 2-13 Setting a Time Series node

---

```
set TimeS.inputs=^input_list
set TimeS.method='ExpertModeler'
set TimeS.expert_modeler_method='Arima'
execute TimeS
```

---

The Time Series node is named TimeS (Figure 2-19). Execute the **TimeS** node and the nugget TS\_model to generate the best ARIMA model.

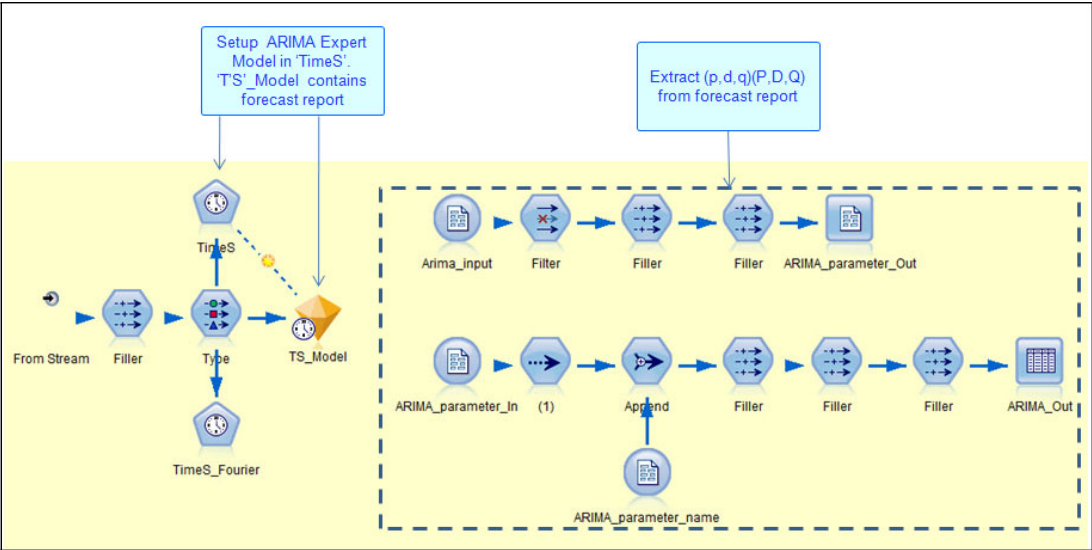


Figure 2-19 Build ARIMA Expert Modeler and extract the parameters of the ARIMA model

- Export the report of the forecast model and save it in a text file. Example 2-14 shows the code that is used to export the forecast report in the TS\_Model node.

*Example 2-14 Export TS\_Model*

```
set FilePath=^TCO_root><"/data/output/Arima_temp.txt"
export TS_Model as ^FilePath format text
```

Figure 2-20 shows a sample forecast report that was exported from ARIMA Expert Modeler. We only wanted to keep the (1,0,0)(1,0,1) ranges.

Target	Model	Predictors	Stationary R**2	R**2	RMSE	MAPE	MAE	MaxAPE	MaxAE	Norm. BIC	Q	df	Sig.
volumes	ARIMA(1,0,0)(1,0,1)	2	0.26	0.26	5.38	48.21	4.16	367.85	16.94	3.53	11.90	15	0.69

Figure 2-20 Sample forecast report

To extract (1,0,0)(1,0,1) from the forecast report, the streams in the rectangle in Figure 2-19 are used to import the report and extract the six parameters (p,d,q)(P,D,Q). Table 2-2 is the ARIMA model that is extracted from the forecast report.

Table 2-2 ARIMA model that is extracted from the forecast report

p	d	q	P	D	Q
1	0	0	1	0	1

3. After extracting the ARIMA model, a record of the extracted model ARIMA(1,0,0)(1,0,1) can be kept and applied to different data sets. ARIMA(1,0,0)(1,0,1) can be evaluated by calculating the average forecast error. Example 2-15 shows the code that is used to set up the parameters of the ARIMA model in the Time Series node by using the parameters that are extracted in Step 2.

---

*Example 2-15 Set up the ARIMA model parameters*

---

```
set TimeS.arima_p = to_integer(^p+1-1)
```

---

The `TimeS.arima_p = ^p` or `TimeS.arima_p = to_integer(^p)` does not give the correct results because SPSS Modeler cannot recognize the variable type automatically. Example 2-16 shows the script that is used to run ARIMA Expert Modeler, extract the model, and use the extracted ARIMA model.

---

*Example 2-16 Existing script in SPSS Modeler to extract the ARIMA model from Expert Modeler*

---

```
if ^method='EM_SARIMA'
  set TimeS.inputs=^input_list
  set TimeS.method='ExpertModeler'
  set TimeS.expert_modeler_method='Arima'
  execute TimeS
  write log 'EM_SARIMA '

  set FilePath=^TCO_root><"/data/output/Arima_temp.txt"
  insert model 'TS_Model'
  export TS_Model as ^FilePath format text
  execute ARIMA_parameter_Out
  execute ARIMA_Out
  delete 'TS_Model'
  delete model TS_Model
  set TimeS.method='Arima'
  set p = value ARIMA_Out.output at 1 1
  set d = value ARIMA_Out.output at 1 2
  set q = value ARIMA_Out.output at 1 3
  set sp = value ARIMA_Out.output at 1 4
  set sd = value ARIMA_Out.output at 1 5
  set sq = value ARIMA_Out.output at 1 6

  set TimeS.arima_p = to_integer(^p+1-1)
  set TimeS.arima_d = to_integer(^d+1-1)
  set TimeS.arima_q = to_integer(^q+1-1)
  set TimeS.arima_sp = to_integer(^sp+1-1)
  set TimeS.arima_sd = to_integer(^sd+1-1)
  set TimeS.arima_sq = to_integer(^sq+1-1)
endif
```

---

## **Spectrum analysis in SPSS Modeler -call R**

Spectrum analysis is a key step in Fourier transformation. However, no spectrum analysis module exists in SPSS Modeler. We cannot perform spectrum analysis directly in SPSS Modeler. Fortunately, there is a spectrum analysis module in SPSS Statistics and also in R. SPSS Statistics and R can be called from SPSS Modeler.

Calling R in SPSS Modeler Version 16 is straightforward by using the following steps:

1. Add an R node to the stream. Figure 2-21 shows the *xreg* node is the R node.

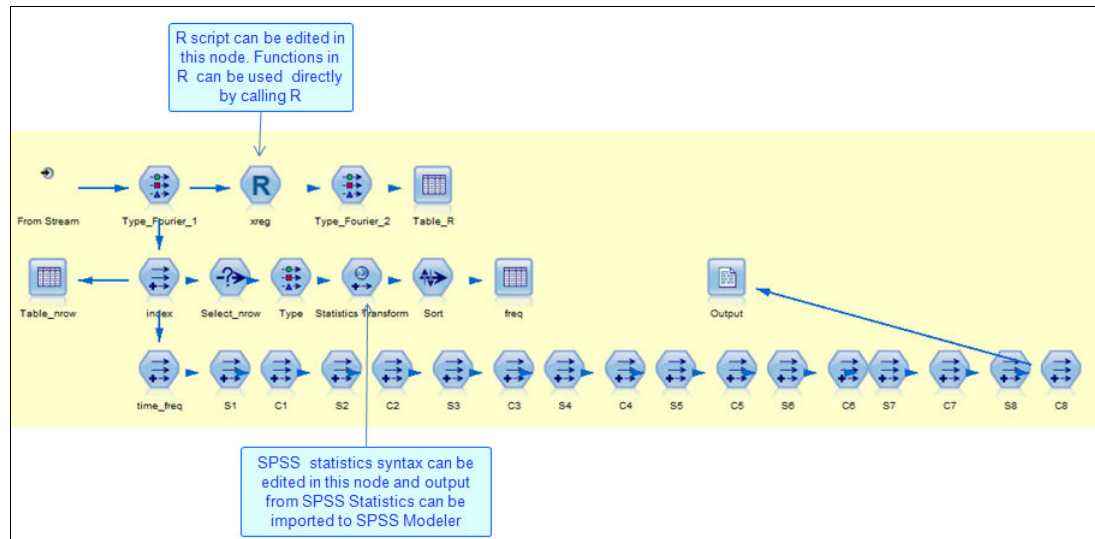


Figure 2-21 Spectrum analysis in SPSS Modeler - call R or call Statistics

2. Copy the R code for the Fourier transformation and paste it in the **R Transform syntax** area under the Syntax tab (Figure 2-22). Functions in R can be called and the output from R is imported to SPSS Modeler through this node.

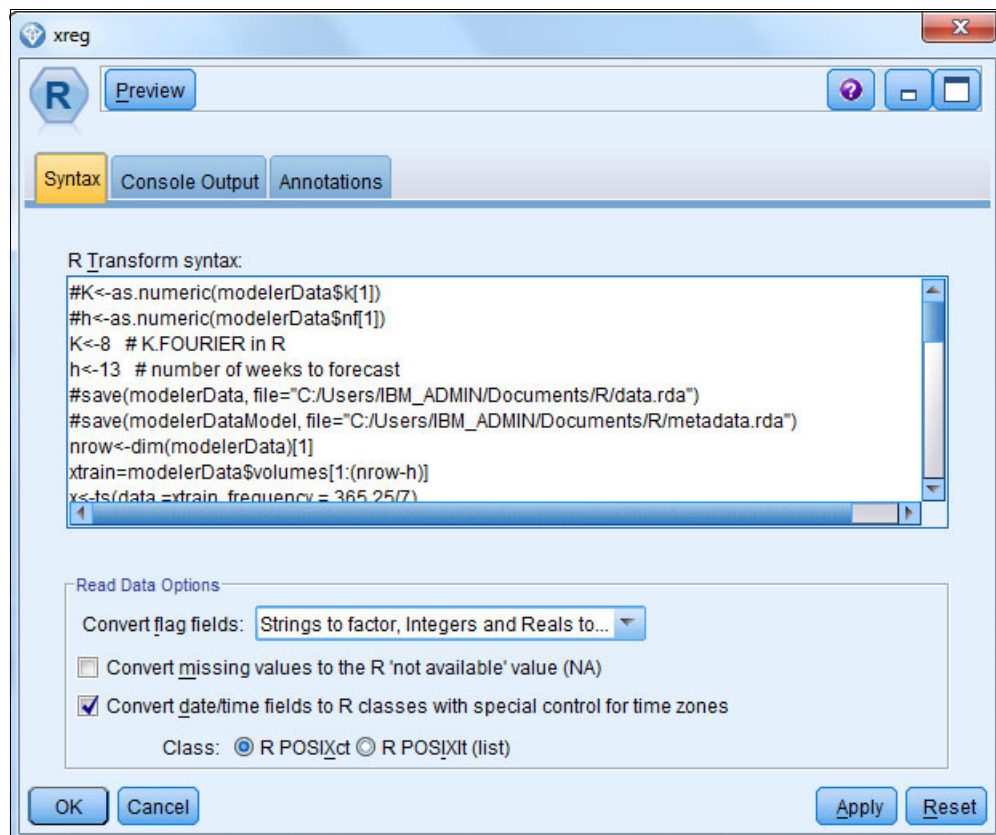


Figure 2-22 Edit syntax in the R node for the Fourier transformation

## Spectrum analysis in SPSS Modeler - call SPSS Statistics

Calling SPSS Statistics for a Fourier transformation is not as straightforward as calling R. The output of spectrum analysis is imported into SPSS Modeler and the following calculation is done by using the SPSS Modeler stream. Use the following steps to perform Fourier transformation by calling IBM SPSS Statistics:

1. Add a Statistics Transform node to the main stream. Figure 2-23 shows a Statistics Transform node that was added. Edit the syntax in the Statistics Transform node. The spectrum analysis is performed by calling SPSS Statistics. The frequencies with the highest spectral density (where the peaks occur) are selected.

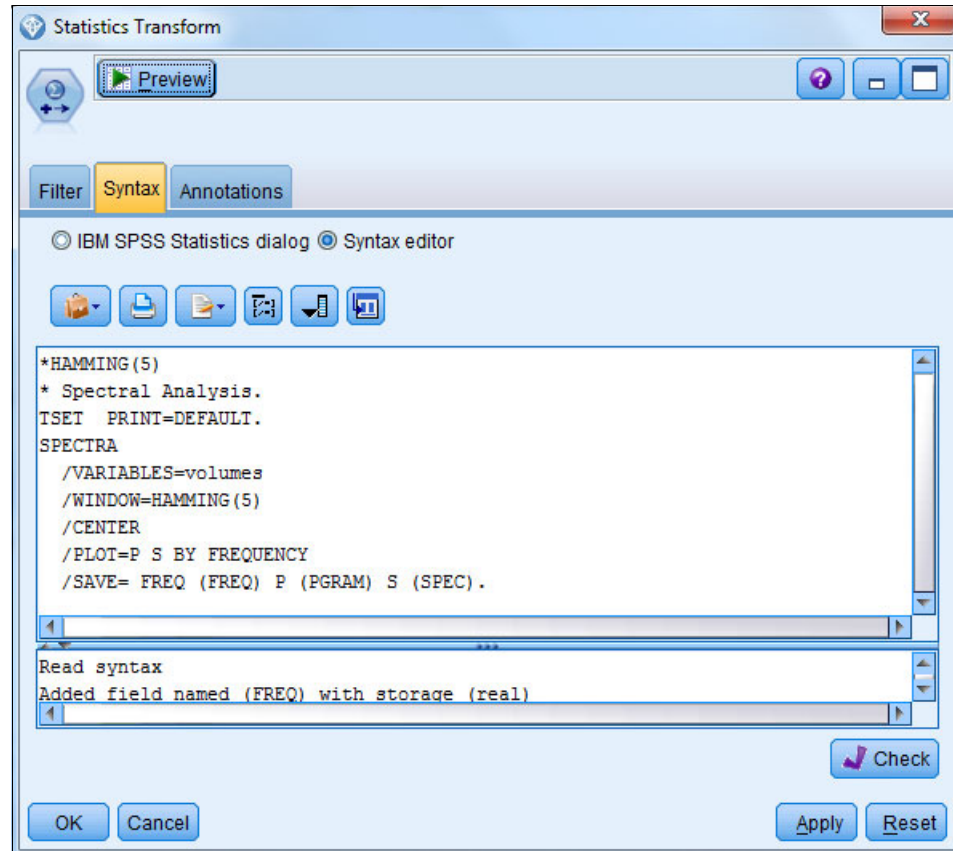


Figure 2-23 Edit the syntax in the Statistics Transform node for spectrum analysis

- Figure 2-24 shows the output of spectrum analysis. The records are sorted according to the spectral density, that is, column PGRAM\_1. In this case, the first eight frequencies in column FREQ are used for Fourier transformation.

	date	country_group	volumes	wq	request_type	index	FREQ	PGRAM_1
1	2012-01-08	CHINA	0	2	Contract_Prepara	2	0.009	197.670
2	2012-01-15	CHINA	0	3	Contract_Prepara	3	0.019	25.205
3	2012-04-29	CHINA	4	5	Contract_Prepara	18	0.157	19.153
4	2012-02-05	CHINA	0	6	Contract_Prepara	6	0.046	15.994
5	2012-04-22	CHINA	9	4	Contract_Prepara	17	0.148	15.877
6	2012-02-12	CHINA	0	7	Contract_Prepara	7	0.056	15.636
7	2012-01-29	CHINA	1	5	Contract_Prepara	5	0.037	10.945
8	2012-09-09	CHINA	6	11	Contract_Prepara	37	0.333	10.907
9	2012-10-07	CHINA	4	2	Contract_Prepara	41	0.370	10.531
10	2012-09-02	CHINA	2	10	Contract_Prepara	36	0.324	9.814
11	2012-05-20	CHINA	2	8	Contract_Prepara	21	0.185	8.977
12	2012-09-30	CHINA	0	1	Contract_Prepara	40	0.361	8.544
13	2012-08-26	CHINA	2	9	Contract_Prepara	35	0.315	8.092
14	2012-07-01	CHINA	1	1	Contract_Prepara	27	0.241	8.082

Figure 2-24 Output of the spectrum analysis from the Statistics Transform node

- After getting the frequencies of the peaks from SPSS Statistics, the calculation of Fourier transformation (Figure 2-25) is conducted in SPSS Modeler. Based on the frequencies from spectral analysis in step 2, Si and Ci can be derived by using the following calculation.

$$S_i = \sin(2\pi f_i t) \quad C_i = \cos(2\pi f_i t) \text{ for } i \text{ from } 1 \text{ to } 8$$

Figure 2-25 Calculation of Fourier transformation

Si and Ci can be used as predictors in the multivariate ARIMA model. Si and Ci are calculated by using an SPSS script, and SPSS Modeler nodes are used to combine all Si and Ci.

Example 2-17 shows the script in SPSS Modeler for the Fourier transformation.

Example 2-17 Existing script in SPSS Modeler

```
if call=='Statistics'
  execute Table_nrow
  set Select_nrow.condition= "index>( "><Table_nrow.output.row_count >< "-" ><
^number_of_weeks_to_forecast><")"
  execute freq
  var deletelist
  set deletelist=[]
  set lastnode = 'time_freq'
  for i from 1 to K
    set freq_i= value freq.output at i freq.output.col-2   var mynode
    set nodenameS="S"><^i
    set nodenameC="C"><^i
    set deletelist=deletelist+^nodenameS
    set deletelist=deletelist+^nodenameC
```

```

set mynode = create derivenode at 100+i*100 270
rename ^mynode as ^nodenameS
connect ^lastnode to ^nodenameS
set lastnode = ^nodenameS
set ^nodenameS.new_name = ^nodenameS
set ^nodenameS.formula_expr = "sin(2*pi*t*><^freq_i><)"
set mynode = create derivenode at 100+i*100+50 270
rename ^mynode as ^nodenameC
connect ^lastnode to ^nodenameC
set lastnode = ^nodenameC
set ^nodenameC.new_name = ^nodenameC
set ^nodenameC.formula_expr = "cos(2*pi*t*><^freq_i><)"
endfor
connect ^mynode to Output
execute Output
disconnect Output
endif

```

---

## 2.8.3 Results

Taking advantage of ARIMA Expert Modeler in SPSS made it possible to convert the Seasonal ARIMA model from R to SPSS easily. And by calling R code or SPSS Statistics, the conversion to Fourier ARIMA is possible. Table 2-3 compares the running time of Seasonal ARIMA and Fourier ARIMA. The average running time of building one Time Series model is calculated. For us, it was the average running time to generate forecasts for one country group and request type combination.

*Table 2-3 Running time comparison - R and SPSS*

Method	Average running time (in minutes)
Seasonal ARIMA in R - exhaustive searching	0.72
Seasonal ARIMA in SPSS - Expert Modeler	0.46
Fourier ARIMA in R - exhaustive searching	0.08
Fourier ARIMA in SPSS (call R) - Expert Modeler	0.76
Fourier ARIMA in SPSS (call Statistics) - Expert Modeler	1.58

As shown in Table 2-3, Seasonal ARIMA in SPSS - Expert Modeler takes a shorter time (36% lower) compared with Seasonal ARIMA in R - exhaustive searching because it takes advantage of ARIMA Expert Modeler.

However, both Fourier ARIMA in SPSS (call R) and Fourier ARIMA in SPSS (call Statistics) take much longer compared with Fourier ARIMA in R - exhaustive searching because building the connection between SPSS Modeler and R (or SPSS Statistics) is time-consuming.

Fourier ARIMA in R - exhaustive searching takes much less time than Seasonal ARIMA in R - exhaustive searching because we only need to search three parameters for Fourier ARIMA. However, we need to search six parameters in Seasonal ARIMA. Although we needed to perform Fourier transformation for the Fourier ARIMA model, the running time of Fourier transformation in R is negligible.

Table 2-4 shows the forecast error comparison of the seasonal ARIMA model in R and SPSS.

Table 2-4 Forecast error comparison of the seasonal ARIMA model in R and SPSS

Tool	Data set	Historical_ vol	Week ahead_1	Week ahead_2	Week ahead_3	Week ahead_4	Week ahead_5	Week ahead_6
R	1	2	65%	55%	66%	53%	70%	71%
SPSS	1	2	63%	62%	63%	63%	63%	60%
R	2	425	14%	12%	12%	13%	13%	15%
SPSS	2	425	11%	10%	13%	11%	18%	18%

It is important to understand the different data sets:

- ▶ Data set 1: brand = Configured Hardware, center = Dalian, Country group = AP, and request type = Contract Preparation
- ▶ Data set 2: brand = Configured Hardware, center = Dalian, Country group = AP, and request type = Pricing Support

We calculated the forecast error for a one-week ahead forecast, a two-week ahead forecast, up to a six-week ahead forecast separately. For data set 1, the forecast error of R and SPSS is around 65% because the average volume in data set 1 is low. Data set 2 has a higher volume, causing a forecast error for both R and SPSS of around 13%. You can see that both the R and SPSS models can provide acceptable forecast accuracy and that the forecast accuracy from R and the forecast accuracy from SPSS are comparable.

The forecast error of R and the forecast error of SPSS Modeler were compared with other data sets. In general, we discovered that the R model provides better forecast accuracy when historical volume is high. But the SPSS model provides better forecast accuracy when the historical volume is low. The difference in the forecast error of these two methods is insignificant. Overall, we discovered that the forecast error of the R model and the forecast error of SPSS Modeler provide comparable forecast accuracy.

## 2.9 AGG: Ensemble models and automatic model selection

This solution combines forecasts from different model families. This solution is preferred over the automatic model selection feature of SPSS Modeler.

*Model averaging* as a technique enables you to average the fits for a number of models, instead of selecting a single best model. The result is a model with excellent prediction capability. This feature is particularly useful for new and unfamiliar models that you do not want to over-fit. When many terms are selected into a model, the fit tends to inflate the estimates. Model averaging tends to shrink the estimates on the weaker terms, yielding better predictions.

*Model selection* has the objective of helping to deal with uncertainty about the model specification. The selected model is restricted to the set of candidate models. The result of a model selection procedure is a single *best* model. The selection criterion can vary but it must be inline with forecasting performance. As a result, forecasts are reported from a single model and underlying this fact is the implicit assumption that the chosen model actually generated the data. The best model however can change every time that new data streams come in. Changing a model for every new update is hard to justify.

The significant advantage of using *aggregated forecasts* is that you do not select a single model, but instead you get strength from the set of available models. Aggregated forecasts are robust because you are averaging over models and weighting the ones with the best performance, for example, better predictive ability.

To create an aggregate forecast, pool forecasts from four types of models:

- ▶ Holt-Winters additive
- ▶ Holt-Winters multiplicative
- ▶ Seasonal ARIMA
- ▶ Fourier ARIMA

Forecasts from each of these models are weighed by a performance measurement and the forecast accuracy, and averaged to obtain the aggregate forecast.

Consider both ARIMA models and exponential smoothing methods. One advantage of the exponential smoothing models is that they can be nonlinear. Series that exhibit nonlinear characteristics, including heteroscedasticity or volatility, might be better modeled by using exponential smoothing models.

In the example R code (Example 2-18), individual forecasts and forecast accuracy measurements from the four models are pooled to obtain the aggregate forecast. Assume that the forecasts from the four specified models are obtained and saved in the objects `fc.HWA`, `fc.HWM`, `fc.SA`, and `fc.FA`. The forecasting horizon is assumed to be fixed and equal for all of the forecasts. In addition, forecast accuracy measurements are obtained and saved in the objects `acc.HWA`, `acc.HWM`, `acc.SA`, and `acc.FA`. The weights are an exponential transformation of the accuracy measurement normalized so that the weights add up to 1. Forecasts with better accuracy will contribute more to the aggregate forecast because their weights will be larger. The final outputs are aggregate forecasts for the set horizon `fc.AGG`.

*Example 2-18 Measure the four models to obtain the aggregate forecast*

---

```
INPUTS  fc.HWA,fc.HWM, fc.SA, fc.FA, acc.HWA, acc.HWM, acc.SA, acc.FA
OUTPUTS fc.AGG
CALCULATE the normalizing constant as  $K = \exp(\text{acc.HWA}) + \exp(\text{acc.HWM}) + \exp(\text{acc.SA}) + \exp(\text{acc.FA})$ 
INITIALIZE the weights as  $\exp(\text{acc})/K$ 
CALCULATE the aggregate forecasts fc.AGG as the weighted average of fc.HWA, fc.HWM, fc.SA, and fc.FA
```

---

The framework that was developed was named AGG because it is a form of model aggregation or model averaging.

## 2.9.1 Challenge and approach

AGG on its own identifies the model family that is better for the current business scenario. AGG combines forecasts of transaction volumes from different model families by using weights. Forecasts from specific models are given higher weight if that model is better for the current business scenario. At the same time, AGG is more dependable because AGG does not rely on only one model family.

Holt Winters Additive, Holt Winters Multiplicative, and Seasonal ARIMA models were built for each center, LOB, request type, and country group combination. The models were built for all combinations once every two weeks. Manually selecting model types for each combination is difficult.

The AGG solution is described in following steps:

1. Create a Time Series node that is named TimeS that is used to build models. Figure 2-26 shows the Time Series node.

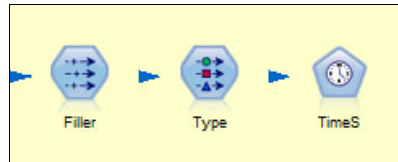


Figure 2-26 TimeS node builds various time series models

2. Click **Criteria** in the Time Series node window (Figure 2-27).

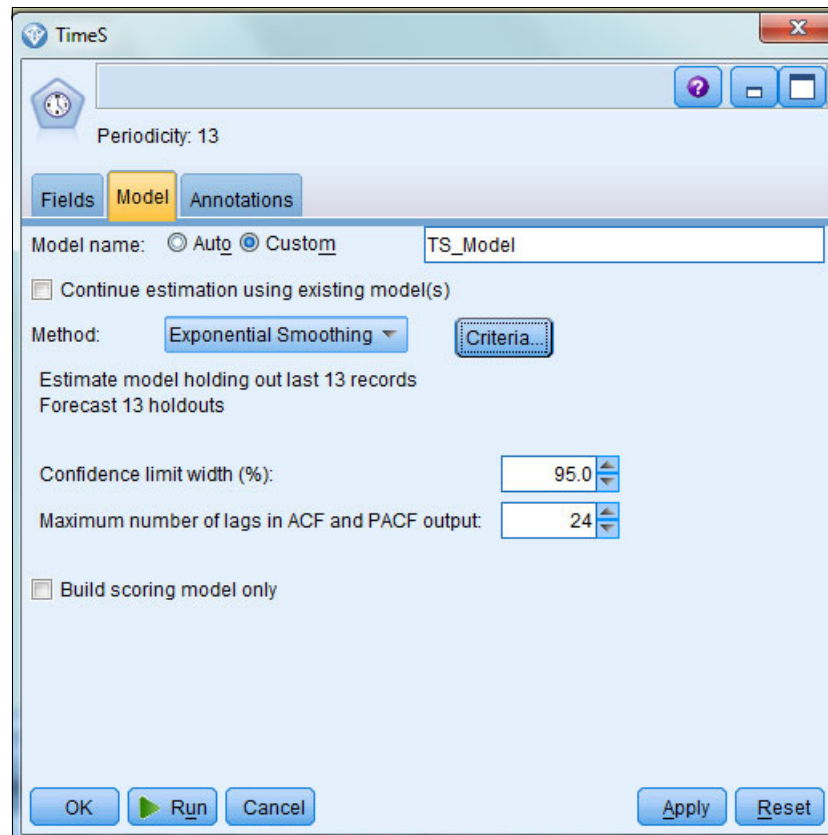


Figure 2-27 Time Series node window

3. The Time Series Modeler: Exponential Smoothing Criteria dialog opens. Specify the model type (Figure 2-28).

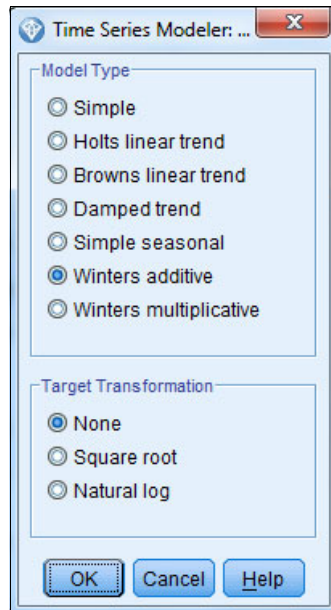


Figure 2-28 Select the model type in the Time Series Modeler: Exponential Smoothing Criteria dialog

Example 2-19 shows the script that was used to select the model family automatically.

*Example 2-19 Script automatically selects a specific model family*

---

```
elseif ^method='HW_Additive'
setTimeS.inputs=[]
    setTimeS.method='Exsmooth'
    setTimeS.exsmooth_model_type = 'WintersAdditive'
executeTimeS
```

---

4. Back testing: For each combination, models from three model families are fitted on 13 different data sets. Forecasts are generated for the next 13 weeks for all 13 back testing data sets. The purpose is to discover how accurate forecasts were from each of the three model families. A model family whose forecasts have higher accuracy gets a higher weight.

The same Time Series node is used for back testing and forecasting. Because 13 back testing runs exist for each combination, a script is used to run them in a loop (Example 2-20).

*Example 2-20 Script to run back testing on 13 data sets in loop*

---

```
for j from (Table_date.output.row_count-Num_testing) to
(Ta-ble_date.output.row_count)
settrain_date=value Table_date.output at j 1
set Remove13WeekAgo.condition = "date_weeks_difference (to_date(' "
>< ^train_date>< "'), date) <= " ><number_of_weeks_to_forecast #><" and
date_weeks_difference(to_date('2012-1-1'), date) >=0"
setTimeIntervals.estimation_num_holdouts = ^min_Num_2
setBuildModel.parameters.method=^method
```

---

5. Calculate the weights of forecasts: For each center, LOB, request type, and country group combination, weights are calculated at the model family week level. For example, there is a weight for a one-week ahead forecast from the Seasonal ARIMA model.

The weight of each model family is calculated by using the following formula (Figure 2-29):

$$W_i = e^{-b \cdot \text{error } m_i}$$

Figure 2-29 Weights for the model family

- $i$  takes values 1 - 13. For example,  $w_{13}$  is the weight of the forecast of the 13th week in the future. Weight  $w_{13}$  will differ for the Seasonal ARIMA, HW Additive, and HW Multiplicative forecasts.
- $b$  is a constant with a fixed value of 10.
- If  $i$  is 1, error  $m_i$  is the average of 13 forecast errors of one week ahead forecasts.

Thirteen forecast errors of one week ahead forecasts are obtained from 13 back testing runs:

- a. Build an error matrix of forecast errors of 13 week ahead forecasts. Forecasts are generated when the model is fitted on the 13 back testing data sets (Figure 2-30).

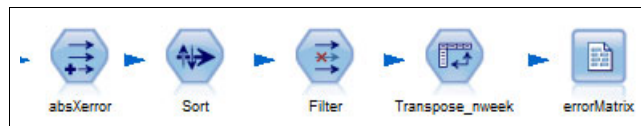


Figure 2-30 Forecast errors from back testing to calculate weights

Example 2-21 shows the script that is used to create the error matrix. The script is run once for every back testing data set. Each time that the script is run, forecast errors for a specific back testing data set are calculated. The forecast errors are for forecasts that are generated when the model is fitted on the back testing data set.

Example 2-21 Script to build the error matrix

```

for if j<Table_date.output.row_count then
set Keep13WeeksTrain.condition = "date_weeks_difference(to_date(' " >
^train_date>< "'), date) > " >< 0 ><" and date_weeks_difference(to_date(' "
>< ^last_date>< "'), date) <= " >< 0
set Remove13WeeksForecast.condition = "date_weeks_difference(to_date(' " >
^train_date>< "'), date) <= " >< 0

executemhisvol
sethisvol=value mhisvol.output at
mhisvol.output.row_countmhisvol.output.column_count
setabsXerror.formula_expr = "abs (volumes-'$TS-volumes')/"><^hisvol
setTranspose_nweek.num_new_fields=^number_of_weeks_to_forecast
executeerrorMatrix
seterrorMatrix.write_mode='Append'
deleteTS_Model
delete model TS_Model
clear outputs
  
```

- b. Use the error matrix (Figure 2-31) to calculate weights. The last Filler node calculates weight by taking the average of forecast errors of forecasts from different back testing data sets.



Figure 2-31 Calculate weights by using the error matrix

Figure 2-32 shows how the Filler node is calculating weight by using the formula (Figure 2-29 on page 52). FIELD in the dialog box is the forecast error.

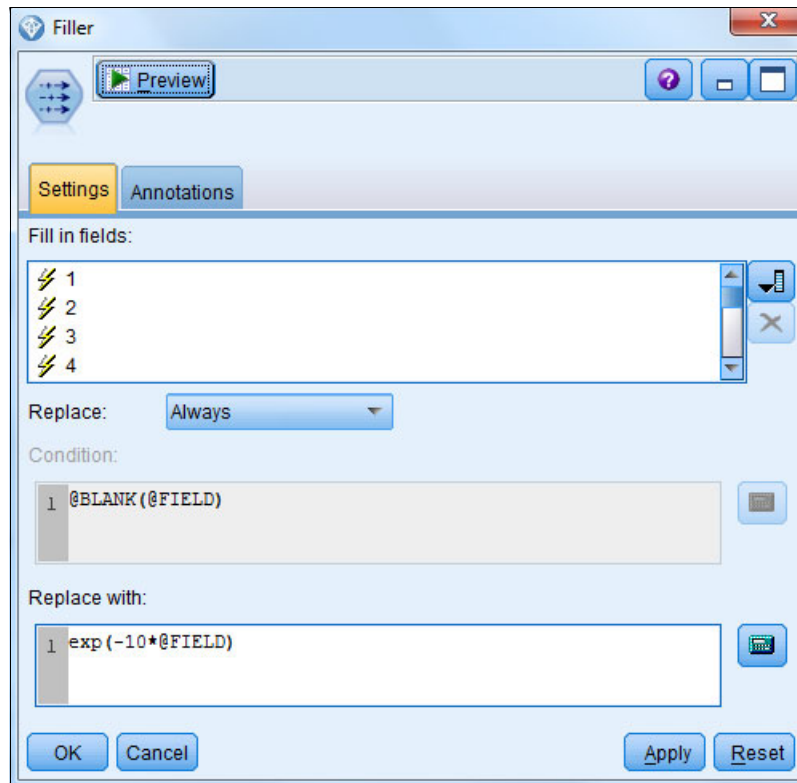


Figure 2-32 Filler node calculating the weight

6. Create forecasts for a 13-week horizon for each model family, and then combine them by using weights to get a final forecast. Example 2-22 shows the script to create a forecast and the processing that is used to combine forecasts.

*Example 2-22 Script for creating the forecast*

```
for j from (Table_date.output.row_count-Num_testing) to
(Ta-ble_date.output.row_count)
settrain_date=value Table_date.output at j 1
set Remove13WeekAgo.condition = "date_weeks_difference(to_date(' " ><
^train_date>< "'), date) <= " ><number_of_weeks_to_forecast #><" and
date_weeks_difference(to_date('2012-1-1'), date) >=0"
setTimeIntervals.estimation_num_holdouts = ^Num_testing
setBuildModel.parameters.method=^method

var log # file for log ^model_output_directory
setlogname = ^TCO_root><"/data/log/">< ^logfilename><".txt"
set log = open append ^logname
writeln log 'calculate weights and forecasts: ' ><^train_date
```

```

close log
executeBuildModel
insert model 'TS_Model' connected between eq_eom and Type_before_Minus

setRename.new_name.forecast='forecast_ '><^method
setRename.new_name.accuracy='accuracy_ '><^method
setRename.new_name.pre_forecast='pre_forecast_ '><^method
setErrorMatrix_input.full_filename="asdf"
setErrorMatrix_input.full_filename=^TCO_root>< "/data/output/errorMatrix.csv"
executeerror_rowcount
setSelect_err_row.condition = "in-dex="><error_rowcount.output.row_count
executeRawForecast_method
deleteTS_Model
delete model TS_Model
clear outputs
endif

```

---

The product of forecast and weight is calculated and the result is named pre\_forecast in the stream (Figure 2-33). The weight is named accuracy in the stream.

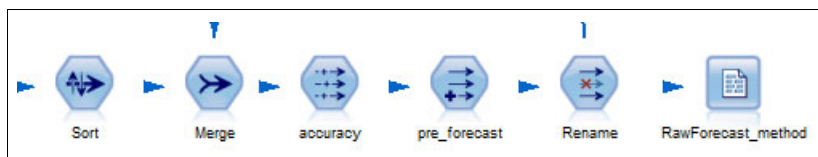


Figure 2-33 Product of weight and forecast calculated

Figure 2-34 shows how the product of the forecast and accuracy is calculated and named as pre\_forecast by using the Derive node.

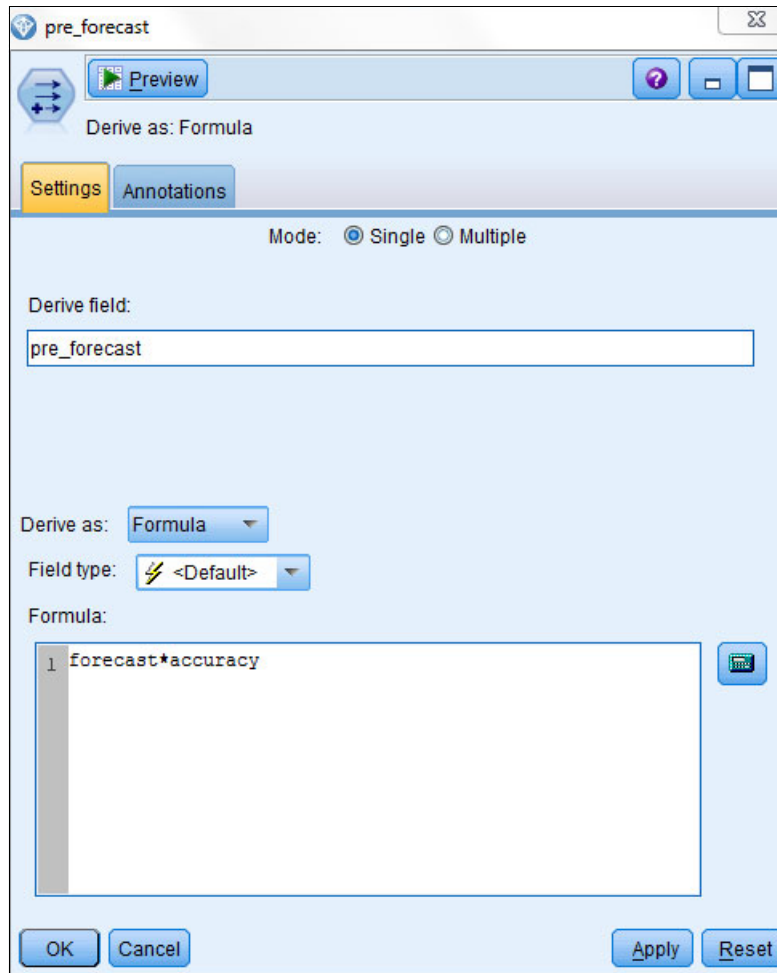


Figure 2-34 Derive node dialog

Forecasts are combined by using weights and the following formula (Figure 2-35).

$$\text{Forecast} = (\text{weight}_{\text{HW Additive}} * \text{forecast}_{\text{HW Additive}} + \text{weight}_{\text{HW Multiplicative}} * \text{forecast}_{\text{HW Multiplicative}} + \text{weight}_{\text{SeasonalArima}}) / (\text{weight}_{\text{HW Additive}} + \text{weight}_{\text{HW Multiplicative}} + \text{weight}_{\text{SeasonalArima}})$$

Figure 2-35 Forecasts are combined by using weights

The combined forecast is calculated by using pre\_forecasts and weights (Figure 2-36).

forecast

Preview

Derive as: Formula

Settings Annotations

Mode: ☒ Single ☐ Multiple

Derive field:

forecast

Derive as: Formula

Field type: <Default>

Formula:

```
1 (pre_forecast_HW_Additive+
2 pre_forecast_HW_Multiplicative+
3 pre_forecast_Seasonal_Arima) /
4 (accuracy_HW_Additive+accuracy_HW_Multiplicative+
5 accuracy_Seasonal_Arima)
```

OK Cancel Apply Reset

Figure 2-36 Combined forecasts calculated

## Automatic model selection in SPSS Modeler: An alternate solution

In SPSS Modeler, automatic model selection functionality is present for time series models. For automatic model selection from all the time series models in SPSS Modeler (by using the Time Series node named TimeS), select **Expert Modeler** as the Method (Figure 2-37) and click **Criteria**.

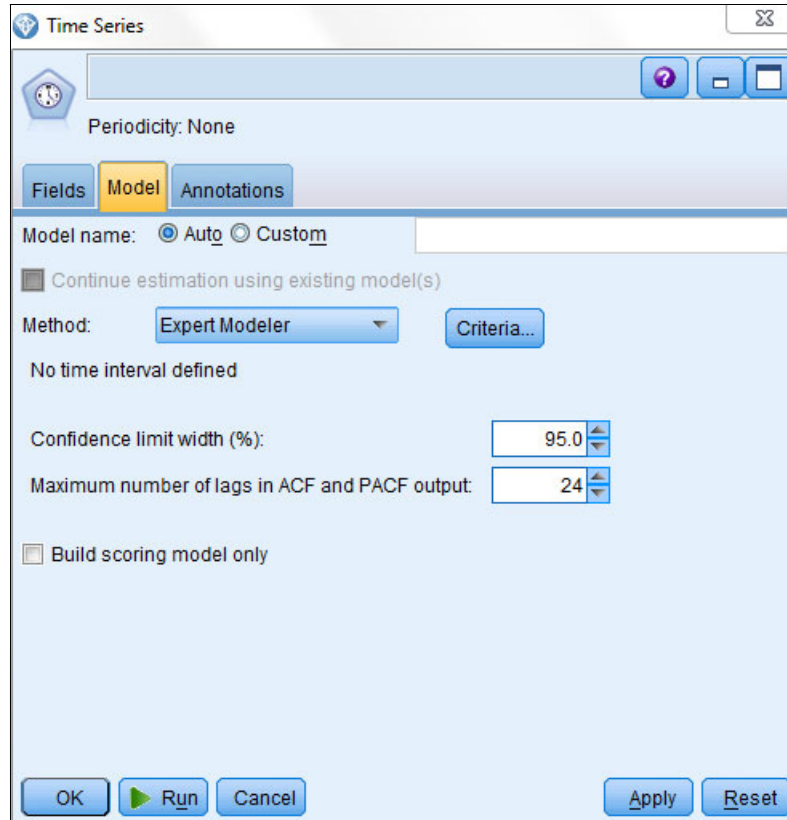


Figure 2-37 Automatic model selection by using Expert Modeler

On the Time Series Modeler: Expert Modeler Criteria window (Figure 2-38), select **All models** for the Model Type, select **Expert Modeler considers seasonal models**, and click **OK**.

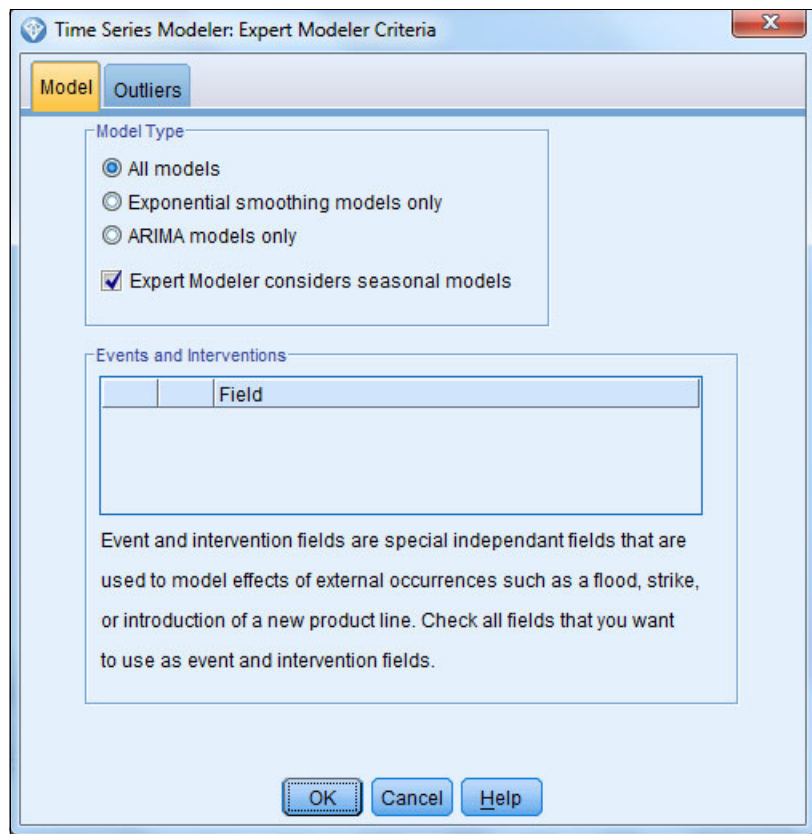


Figure 2-38 Time Series Modeler: Expert Modeler Criteria

The model that best fits the data is selected. This method of model selection can result in over-fitting. This method was not selected. The focus is to create forecasts that are consistently accurate. Therefore, it is better to select the model family that provides higher accuracy.

## 2.9.2 Results

AGG forecast accuracy is higher than previous forecasting solutions. One of the previous forecasting solutions used forecast error to identify the best model from the model families. The model families that were considered were Linear Model, Poisson Regression, and Negative Binomial Regression. The ARIMA model was built on residuals. The previous model did not combine forecasts.

It is frequently observed that the forecast accuracy of forecasts from AGG does not decline when the forecast horizon increases.

We learned that AGG quickly adapts to changes in business environment, resulting in more accurate forecasts. AGG reduces the cost to business for model validation at regular intervals. Because the solution is a combination of model families, it is more readily deployable to other centers, LOBs, or even outside sales support, such as the Procurement and Manufacturing organizations. AGG can also consider the impact of items, such as the sales pipeline, revenue targets, and holidays, on the volume of requests.

## 2.10 Setting parameters

*Parameters* are user-defined variables that are used in programming to control the behavior of a script. Parameters provide information, such as values that control the flow and progress of a routine. Most parameters do not need to be hardcoded in the script. In general, hardcoding parameters restricts the general purpose of a script. It is important to define and implement an interface or a wrapper for these input parameters to make the script flexible and ready for further reuse and modification. Examples of parameters in the forecasting solution are the directory of the data, the directory of the output forecasts, and the number of weeks to forecast ahead (that is, the forecast horizon).

### 2.10.1 Challenge and approach

SPSS Modeler has two execution modes: the user interface mode and the batch mode. The methods for the input parameters for each of them differ:

- ▶ User interface mode

The input parameters are defined and set as prompts in the Parameters tab. Then, each time that the stream starts to execute, a dialog appears for the user to set the input parameters.

- ▶ Batch mode

No user interface system is available. The input parameters are specified within the execution command line. If many parameters exist, they can be saved to a text file. The text file name is put in the command line instead.

The SuperNode script can also specify input parameters. However, these input parameters can only be set through the stream that calls the SuperNode. So, any parameters for the SuperNode stream need to be forwarded by the main stream script.

Another difficulty in developing the parameter interface in SPSS Modeler is how to enter a set of values to one parameter slot, for example, to specify a set of particular holiday names to the stream to use as predictors for the ARIMA model. You cannot define one parameter and store a set of holiday names in that parameter. SPSS Modeler looks at the holiday names set as one character string, instead of a set of strings. To resolve this situation, a text file was created to save the input set as a column of values and then to read the file with a script.

## 2.10.2 Setting SPSS nodes and parameter setup

To define the parameters, open the **Parameters** tab (Figure 2-39). Various parameters are checked as Prompt and used to show details to the user when they start running SPSS Modeler. However, both checked and unchecked parameters are available to the script.

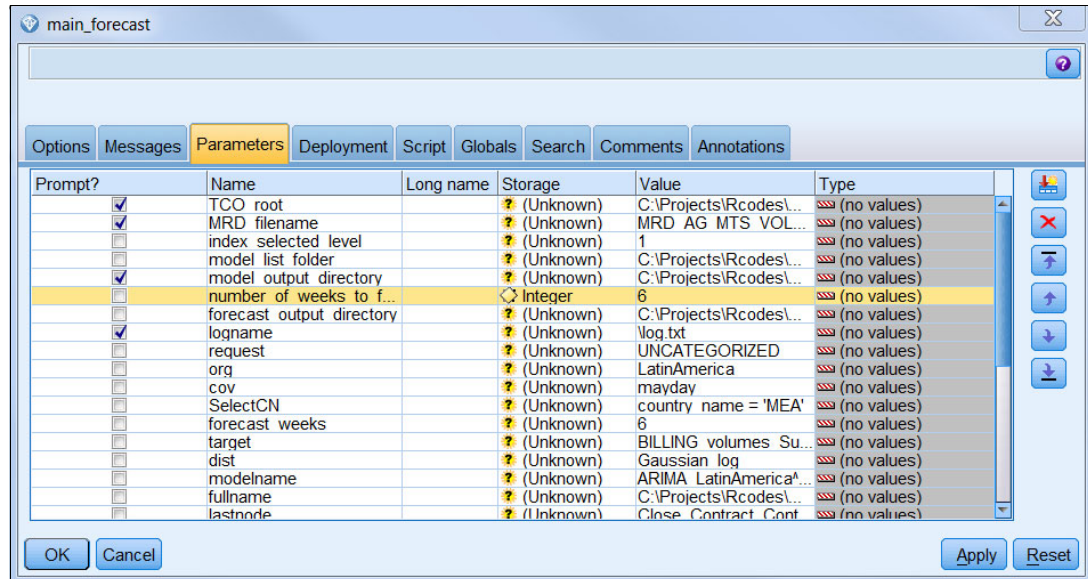


Figure 2-39 Define the input parameters

When the stream is executed in the user interface mode, a parameter input dialog opens (Figure 2-40). The value of parameters can be a string or numeric, but the value cannot be a set of values.

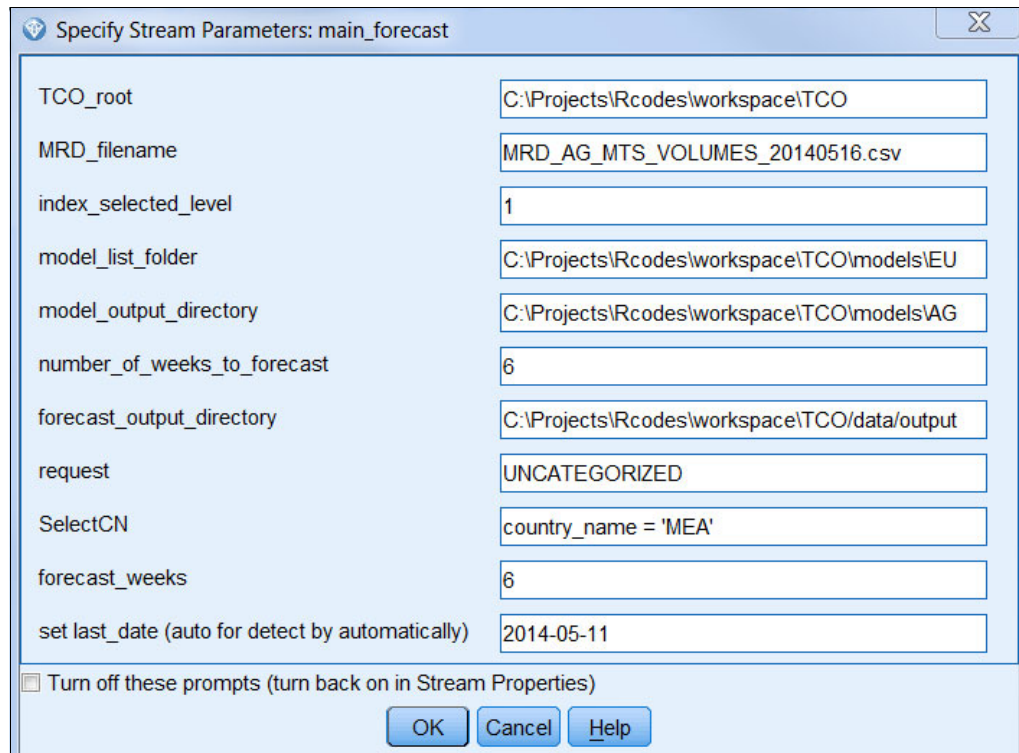


Figure 2-40 The input user interface when the stream starts

When running in batch mode, a text file is used to specify the parameters (Example 2-23).

*Example 2-23 Text parameter specifies the file for batch mode*

---

```
-PTCO_root=/gsa/pokgsa/projects/p/pricepredict2/TC0
-Plocal_root=/gsa/pokgsa/projects/p/pricepredict2/TC0
-PMRD_filename=MRD_AG_US_20140427.csv
-Pmodel_output_directory=/gsa/pokgsa/projects/...
-Pvolumes_filename=AG_US_20140427.csv
-Plog_directory=/gsa/pokgsa/projects/p/pricepredict2/TC0/cmd
-Plast_date=2014-04-27
-Pnumber_of_weeks_to_forecast=6
```

---

In the SuperNode execution, the parameter that is defined by the SuperNode can be set. The stream nodes of the SuperNode can also be set (Example 2-24).

*Example 2-24 Set the parameter of the SuperNode stream*

---

```
set BuildModel.parameters.'GenLin.inputs' =
    SelectionResult.selected_ranked_fields
set BuildModel.parameters.'GenLin.target' = ^request
delete SelectionResult
delete model SelectionResult
set BuildModel.parameters.model_name = ^org ><'^' >< ^request
set BuildModel.parameters.model_directory = ^model_output_directory
execute BuildModel
```

---

Reading a set of values from a text file is no different from reading a set of request types from a data file. For more information, see 2.2, “Switching between forecasting hierarchies” on page 14.

### 2.10.3 Results

SPSS Modeler has capabilities that support user input parameters, by using the user interface mode and via a text file in batch mode. Although SPSS Modeler does not support a combination type for parameters, such as set, this issue was solved in a satisfactory manner.





## Lessons learned

This chapter describes the lessons that were learned from the conversion of the R solution to IBM SPSS Modeler. This chapter includes the following topics:

- ▶ “Planning an R to IBM SPSS conversion project” on page 64
- ▶ “Key decisions” on page 64
- ▶ “Risks and mitigation to consider” on page 65
- ▶ “Calling R functions directly” on page 66

## 3.1 Planning an R to IBM SPSS conversion project

The planning of the R to IBM SPSS conversion project was very important. The planning process included the following key factors:

- ▶ Technical skill

The skills that were required included statistical modeling, creating and understanding R code, and general skills in IBM SPSS Modeler. Beyond these skills, the ability to write scripts in SPSS Modeler was needed.

- ▶ Resource requirements

The project required one or two developers and a technical lead. We preferred that the developers and technical lead were experienced in R, SPSS Modeler, and statistical modeling. We decided to have an expert in each skill on the team.

- ▶ Time

The first few weeks were spent selecting a statistical software platform. First, the team needed to decide between SPSS Modeler and IBM SPSS Statistics, how to perform scripting, and how to handle other issues. The project took approximately six months for development and testing. For an additional six months, the team enhanced the solution, such as deploying the solution on the production server, improving the solution, performing data quality checks on input and output data, and reviewing measurement metrics. Key inputs to the process were historical volumes and output from previous forecasts.

The following factors affected the time that was taken to complete the project:

- ▶ The overall number of businesses that deployed the solution
- ▶ Each business deploying the solution differed from the others
- ▶ The overall complexity of the solution

## 3.2 Key decisions

Before the project started, the development team tried to foresee the possible conversion project requirements. Several important requirements were identified.

The first requirement was that SPSS Modeler include algorithms that were implemented in R code. If the required algorithms were absent, the conversion would be more difficult and even impossible. We identified the two required algorithms, Generalized Linear model (GLM) and autoregressive integrated moving average (ARIMA), which are available in SPSS Modeler. The functions and procedures for data preparation and forecasting have corresponding functions or approaches in SPSS Modeler. One exception is that the stepwise feature selection in the GLM algorithm is not supported by SPSS Modeler. However, we found that this capability is performed by the Feature Selection node of SPSS Modeler.

There was also a requirement for speed and scalability. We discussed the acceptable model build time and forecasting duration with the users. The scale of the target data source was also discussed. We estimated the speed, performance, and scalability of the resulting SPSS Modeler solution, by using our experience and simple experiments. We concluded that this requirement could be satisfied.

We also considered whether the conversion would affect the forecast accuracy. We checked the entire planned forecasting procedure, from the data preparation to the model-building algorithm details, and identified two considerations:

- ▶ The capabilities and algorithms in the two methods were identical.
- ▶ The capability to get an intermediate result by using both methods easily, which was important to track the divergence between the results.

After reviewing these considerations, we were confident that the forecasting accuracy would not drop after the conversion.

### 3.3 Risks and mitigation to consider

There were several risks to the conversion from R to SPSS Modeler. Certain risks were foreseen before the conversion, such as the flexibility of SPSS Modeler. However, other risks were unforeseeable and were discovered during solution development. Two main risks were identified:

- ▶ The flexibility of SPSS Modeler for customized algorithm development

In the R solution, several customized algorithms were developed. Some of these algorithms, for example, the variable selection algorithm that was based on backward elimination and the customized ARIMA model, were difficult to convert to the SPSS model. SPSS Modeler did not allow us to develop the same algorithm as in R. To solve this problem, we used existing modules in SPSS Modeler, in particular, the Feature Selection node and the ARIMA Expert Modeler. However, because the algorithms are not the same, the risk of decreased forecast accuracy after conversion was a concern. The forecast error comparison was carefully analyzed and we discovered that existing SPSS modules were able to provide comparable forecasts to the customized algorithms in R. We learned to take advantage of the existing SPSS Modeler modules, especially when it is difficult to directly convert customized R algorithms. The forecast accuracy of the converted solution must be carefully monitored.

- ▶ Deployment of the converted SPSS solution

Another risk was that parts of the SPSS solution could not be deployed successfully. These risks are two-fold:

- The development environment might not be the same as the deployment environment. Because of the difference in the server environment, your SPSS streams might not run successfully on the deployment server. For example, one module of the SPSS Modeler solution was developed on a local server and it can connect to R/SPSS Statistics successfully. However, the deployment environment was unstable to build connections between SPSS Modeler and R/SPSS Statistics.
- The speed of the SPSS solution in the development environment and the deployment environment might differ. The speed of our solution was faster on the local server than on the production server. A risk exists that the speed of a solution is acceptable in the development environment but that the speed is unacceptable in the deployment environment. To mitigate these two problems, the lesson learned was that an agile strategy, that is, deploying the solution with the solution conversion process, is important. In our case, we saved significant time by deploying and checking the speed of modules of the SPSS solution iteratively.

## 3.4 Calling R functions directly

SPSS Modeler Version 16 supports calling R functions. We considered using this approach instead of implementing everything within SPSS Modeler. After a systemic comparison, it was decided to convert as much as possible to SPSS Modeler for the following reasons:

- ▶ The SPSS Modeler stream is easier to maintain and extend, unless the user is already an R expert.
- ▶ The capability to call R modules is not mature in SPSS Modeler. For example, during one execution of a stream, if the R function calls exceeded 100 iterations (while looping), the execution fails for an unknown reason. This issue was reported to the SPSS development team.
- ▶ R is open source software. If you want to deliver a solution that is developed with R code, you are required to make all the source code in the entire solution open source.

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Using zEnterprise for Smart Analytics: Volume 1 Assessment*, SG24-8007
- ▶ *IBM Information Server: Integration and Governance for Emerging Data Warehouse Demands*, SG24-8126
- ▶ *Governing and Managing Big Data for Analytics and Decision Makers*, REDP-5120
- ▶ *IBM Predictive Maintenance and Quality (Version 2.0)*, TIPS1130

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ IBM SPSS Modeler product page:  
<http://www-01.ibm.com/software/analytics/spss/products/modeler/>
- ▶ The R Project Statistical Computing:  
<http://www.r-project.org>

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)







# Our Experience Converting an IBM Forecasting Solution from R to IBM SPSS Modeler

**Understand the IBM Enterprise Services project and thought process**

**Learn about key challenges and their solutions**

**Gain insight into lessons learned from the project**

This IBM Redpaper publication presents the process and steps that were taken to move from an R language forecasting solution to an IBM SPSS Modeler solution. The paper identifies the key challenges that the team faced and the lessons they learned. It describes the journey from analysis through design to key actions that were taken during development to make the conversion successful.

The solution approach is described in detail so that you can learn how the team broke the original R solution architecture into logical components in order to plan for the conversion project. You see key aspects of the conversion from R to IBM SPSS Modeler and how basic parts, such as data preparation, verification, pre-screening, and automating data quality checks, are accomplished.

The paper consists of three chapters:

- ▶ Chapter 1 introduces the business background and the problem domain.
- ▶ Chapter 2 explains critical technical challenges that the team confronted and solved.
- ▶ Chapter 3 focuses on lessons that were learned during this process and ideas that might apply to your conversion project.

This paper applies to decision makers and IT Architects who focus on the architecture, roadmap, software platform, and total cost of ownership. It also applies to solution development team members who are involved in creating statistical/analytics-based solutions and who are familiar with R and IBM SPSS Modeler.

## **INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

### **BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
**[ibm.com/redbooks](http://ibm.com/redbooks)**